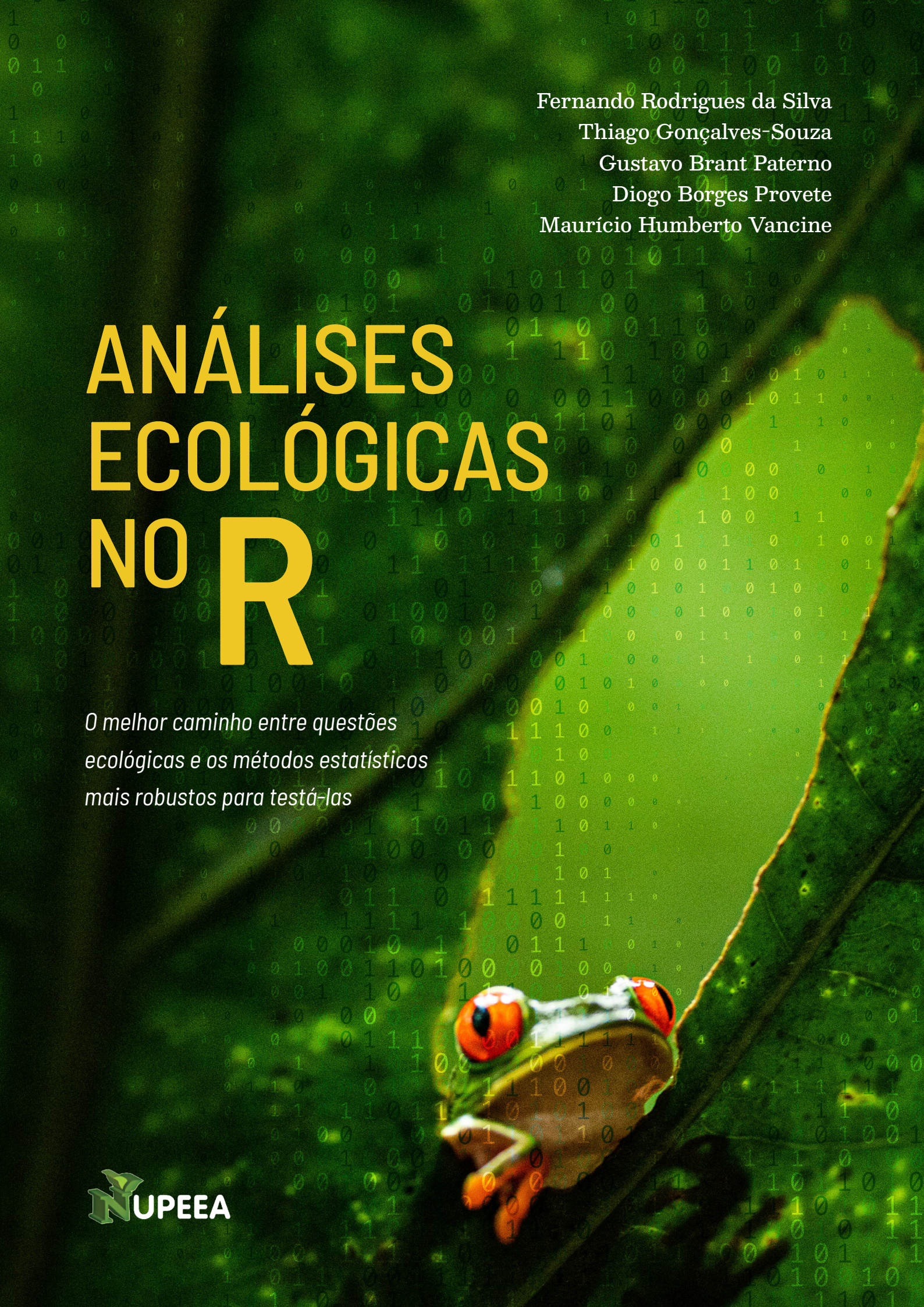
Fernando Rodrigues da Silva
Thiago Gonçalves-Souza
Gustavo Brant Paterno
Diogo Borges Provete
Maurício Humberto Vancine

ANÁLISES ECOLÓGICAS NO R

O melhor caminho entre questões ecológicas e os métodos estatísticos mais robustos para testá-las



NUPEEA

ANÁLISES ECOLÓGICAS NO R

Fernando Rodrigues da Silva
Thiago Gonçalves-Souza
Gustavo Brant Paterno
Diogo Borges Provete
Maurício Humberto Vancine

ANÁLISES ECOLÓGICAS NO R

O melhor caminho entre questões ecológicas e os métodos estatísticos mais robustos para testá-las

NUPEEA

canal6 editora
1ª Edição • 2022
Bauru/SP

Recife – Pernambuco – Brasil

Primeira edição publicada em 2022 por NUPEEA
https://analises-ecologicas.netlify.app/

Editor
Prof. Dr. Ulysses Paulino de Albuquerque

Revisão
Autores

Diagramação
Erika Woelke

Fotos
Filipe Coimbra, Jeremy Bezanger e Nikhil Thomas/Unsplash

Foto da capa
Birger Strahl/Unsplash

Apoio

Instituto Nacional de Ciência e Tecnologia
Etnobiologia, Bioprospecção e Conservação da Natureza

Coedição

canal6 editora

ABEC Brasil
Associação Brasileira de Editores Científicos

Associado CBL Câmara Brasileira do Livro

Dados Internacionais de Catalogação na Publicação (CIP)
(BENITEZ Catalogação Ass. Editorial, MS, Brasil)

A553
1.ed.

Análises ecológicas no R / Fernando Rodrigues da Silva... [et al.] ; [editor] Ulysses Paulino de Albuquerque. – 1.ed. – Recife, PE : São Paulo : Nupeea : Canal 6, 2022.
640 p.; 21 x 30 cm.

Outros autores : Thiago Gonçalves-Souza, Gustavo Brant Paterno, Diogo Borges Provete, Maurício Humberto Vancine.
Bibliografia.
ISBN 978-85-7917-563-3 (e-book)
ISBN 978-85-7917-564-0 (impresso)
DOI 10.52050/9788579175633

1. Análises dados – Ecologia. 2. Linguagem de programação (Computadores). 3. Métodos científicos. 4. Tecnologia – Aspectos ambientais.

03-2022/212

CDD 001.42

Índice para catálogo sistemático:
1. Análises ecológicas : Métodos científicos 001.42

Bibliotecária responsável: Aline Graziele Benitez CRB-1/3129

Dedicamos esse livro às nossas companheiras que nos apoiaram durante o longo processo de construção do livro, e aos nossos filhos e filhas pelas distrações e alegrias quando estávamos longe dos infinitos erros durante as compilações.

Sumário

Prefácio

A evolução dos computadores pessoais e a ampliação do acesso a estes e à Internet têm transformado o jeito como aprendemos e ensinamos. Especificamente sobre a investigação da natureza através de métodos científicos, a maior oferta de ferramentas gratuitas para análise de dados e maior facilidade de acesso a dados talvez estejam entre as maiores transformações ocorridas nos últimos 20 anos. Hoje é relativamente comum, mesmo entre cientistas em formação, conversar sobre novos pacotes, códigos de programação, repositório de dados de acesso público e reprodutibilidade de análises. Para isso ter acontecido, além da evolução tecnológica, também foi preciso que muitas pessoas estivessem dispostas a ensinar como usar essas ferramentas. Este livro é uma dessas contribuições para o contínuo avanço do ensino de métodos computacionais, com um foco específico em análise de dados ecológicos através da linguagem R.

Além da abrangência dos tópicos abordados (desde teste T até análises geoespaciais, passando por diversidade filogenética), há outro aspecto marcante neste livro – e que o torna diferente em relação ao material comumente encontrado em sites, listas, blogs e manuais: a visão dos autores sobre como os códigos devem ser consequências das perguntas que a pesquisa pretende responder (veja detalhes no Capítulo 1). Essa visão tem como consequência um livro que do começo ao fim conecta teoria ecológica, métodos científicos, análises quantitativas e programação. Isso é feito de modo explícito através de exemplos claros e didáticos que apresentam contexto e dados reais, um ou mais exemplos de perguntas que poderiam ser feitas, predições relacionadas às perguntas e a teoria em questão, além das variáveis que poderiam ser utilizadas nas análises. O texto que descreve essas partes é intercalado com pedaços organizados e claros de código e gráficos, o que torna a leitura dos capítulos bastante fluida e dinâmica, principalmente para quem gosta de executar os códigos no seu computador conforme lê os capítulos. É como uma aula prática guiada. Um terceiro aspecto marcante do livro é que boa parte das análises também é explicada conceitualmente. Por exemplo, quando o teste T é introduzido, isso não é feito somente com códigos que permitem o cálculo da estatística. Há explicações sobre o funcionamento e premissas da análise. Isso nos estimula a tentarmos entender o que acontece por trás dos códigos.

Quando eu comecei a ensinar análise de dados ecológicos para discentes de graduação em Ecologia em 2012, na UNESP de Rio Claro, eu dizia para as turmas que nós tínhamos um grande privilégio: um livro texto específico para nossa área e em português – "Princípios de Estatística em Ecologia" (Gotelli & Ellison) e uma apostila de onde eu tirava algumas instruções e alguns exercícios em R também em português. A apostila, elaborada pelos pesquisadores Diogo B. Provete, Fernando R. da Silva e Thiago Gonçalves-Souza, era o embrião deste livro (veja detalhes sobre o histórico do livro no Capítulo 1) e um dos alunos para quem eu dizia sobre o privilégio era o Maurício Vancine. Portanto, é uma satisfação pessoal enorme escrever este prefácio, não só por conhecer quatro dos cinco autores e ter usado uma versão anterior do material, como também por considerar o livro uma das fontes mais abrangentes sobre análise de dados em R. Eu sugiro que você marque o link do livro como um favorito no seu navegador de Internet.

Apesar de os autores não terem separado os capítulos em partes, podemos considerar que há dois grupos de capítulos. No primeiro grupo – que inclui dos capítulos 1 ao 6, nós somos apresentados aos aspectos mais gerais da estrutura do livro, seus objetivos e sobre o funcionamento da linguagem R. Neste grupo, nós aprendemos desde como instalar o R e o RStudio (Capítulo 3), até como produzir gráficos bonitos e informativos (Capítulo 6). No segundo grupo de capítulos (do 7 ao 15), temos contato com análises específicas e atualmente usadas em ecologia, incluindo desde modelos lineares simples (Capítulo 7), até análise de dados geoespaciais complexos (Capítulo 15), passando por diferentes tipos de estratégia para descrever a biodiversidade (Capítulos 10-14).

Este tipo de estrutura permite que o livro possa ser usado como um curso completo ou como guia que consultamos quando esquecemos de como fazer algo. Por exemplo, eu sempre consulto o Capítulo 5, principalmente quando vou usar a função `map` do pacote `purrr`. Ou seja, é possível ler quase todos os capítulos de modo independente, pois todos têm uma seção de pré-requisitos ou uma seção de introdução/contextualização em que os objetivos do capítulo são apresentados.

Entretanto, se você está começando a aprender R, sugiro que leia os capítulos conforme eles são apresentados, principalmente dos capítulos 1 ao 6. Depois, você pode ir para os capítulos que mais te interessam. E já que o assunto é aprendizagem, no Capítulo 1 os autores apresentam ideias criativas e úteis para quem tem interesse em ensinar e aprender com o livro. Gostei bastante das sugestões para grupos de pessoas que querem aprender de forma autônoma, por meio de grupos de estudo. O Capítulo 2 apresenta uma maneira bastante interessante de se usar um dos métodos científicos mais usados – o método hipotético-dedutivo – com fluxogramas para: (a) identificar variáveis relevantes e como elas afetam umas às outras, (b) melhorar (quando necessário) o desenho amostral, (c) facilitar a escolha de análises estatísticas, e (d) melhorar a interpretação e comunicação dos dados e análises. Os capítulos 4 e 5 são essenciais para quem ainda não tem muita familiaridade com o uso do R ou para quem quer revisitar aspectos fundamentais do funcionamento da linguagem. O Capítulo 4 inclui, por exemplo, explicações sobre console, script, operadores, objetos, funções, pacotes, ajuda e principais erros. O Capítulo 5 apresenta uma das maiores inovações surgidas nos últimos anos na comunidade que trabalha com R, "uma filosofia de design, gramática e estruturas" agrupadas em conjuntos de pacotes sob nome guarda-chuva *tidyverse*. Podemos entender o *tidyverse* como um "dialeto novo" para a linguagem R, onde *tidy* quer dizer "organizado, arrumado, ordenado", e verse é "universo" (Capítulo 5). Para quem não usa ou não se acostumou a usar o operador pipe `%>%` para o encadeamento de funções, o Capítulo 5 é bastante útil, pois uma parte subsequente do livro usa essa abordagem. O Capítulo 6 é uma consequência natural do Capítulo 5, pois usa a lógica da gramática dos gráficos, que está fortemente ligada ao *tidyverse*, e a porta de entrada para os próximos capítulos mais específicos sobre análise de dados. Afinal, não fazemos (ou pelo menos não devemos fazer) nenhuma análise de dados sem antes explorá-los graficamente.

Para quem já tem pelo menos alguma familiaridade com a linguagem R, o segundo conjunto de capítulos (do 7 ao 15) pode ser acessado mais diretamente. Aqui também é possível pensar numa estrutura. Do capítulo 7 ao 9, o foco é em métodos estatísticos usados para explorar e estimar a associação entre variáveis. Enquanto os capítulos 7 e 8 são totalmente focados em modelos lineares univariados – em que a variável resposta é um único vetor numérico, o Capítulo 9 trata de métodos em que a variável resposta ou variável de interesse é uma matriz – i.e., as chamadas técnicas multidimensionais ou multivariadas. Eu gostei de ler estes três capítulos na sequência em que são apresentados. É assim que eu ensino também – começamos com uma variável resposta e uma variável explanatória e vamos adicionando variáveis e tipos de variáveis, primeiro no lado direito da equação (a parte explanatória) e depois na parte esquerda da equação (a parte que está sendo modelada).

Do capítulo 10 ao 14, aprendemos como estimar as diferentes facetas da biodiversidade – taxonômica, funcional e filogenética, e os componentes alfa, beta e gama – e como podemos usar algumas dessas medidas para fazer inferências sobre processos de montagem de comunidades. O Capítulo 13 sobre diversidade filogenética merece ser destacado; é um dos mais completos e abrangentes do livro. Apesar de não ser exatamente minha área de atuação, acho que boa parte dos métodos atualmente mais usados está contemplada ali. Em todos estes capítulos há explicações sobre aspectos teóricos que suportam o uso das técnicas em diferentes situações. Tudo é feito com dados que podem ser acessados. Portanto, se você ainda não coletou os dados do seu projeto, mas já sabe quais análises pretende usar, estes capítulos são uma fonte excelente para sua aprendizagem. Não precisa esperar até ter todos os dados em mãos.

Finalmente, o Capítulo 15 fecha o livro nos apresentado uma introdução aos principais conceitos sobre a manipulação e visualização de dados geoespaciais no R. Ainda é comum para quem trabalha com ecologia e quer usar técnicas de análises de dados geoespaciais como algo complementar convidar especialistas da área para uma parceria. O Maurício Vancine já não deve nem conseguir atender todos os pedidos de parceria. Aqui está então uma boa oportunidade para você aprender a dar os primeiros passos, produzir os primeiros mapas, fazer predições espaciais sem ter de depender da ajuda de pessoal especializado. Mas mais importante que isso, o capítulo fornece a base conceitual e terminologia apropriada para o bom uso deste tipo de análise – eu sempre me confundo com a terminologia associada ao sistema de referência de coordenadas e unidades.

Há quem diga que a velocidade com que a tecnologia e a ciência avançam tende a tornar livros e manuais sobre métodos rapidamente obsoletos. Não acho que isso vai acontecer com o livro "Análises Ecológicas no R". Os métodos apresentados são bem estabelecidos e (alguns) aceitos pela comunidade científica há muitos anos e incluem técnicas modernas que ainda estão sendo entendidas e absorvidas por profissionais que atuam em ecologia. Que privilégio ter um livro como este em português e gratuito! Obrigado aos autores por isso. Boa leitura e prática.

Tadeu Siqueira
Instituto de Biociências
Universidade Estadual Paulista - UNESP

Sobre os autores

Fernando Rodrigues da Silva

Piadista descomedido de Barueri/SP, pai da Ária, companheiro da Winter, responsável pela Filomena (gata), Zé (gato), Cenoura (cachorra) e Chica (cachorra), admirador de botecos, viagens, mato, cerveja, poker, basquete e prosa com os amigos. Formado em Ciências Biológicas na UNESP-Assis e com mestrado e doutorado em Biologia Animal pela UNESP-São José do Rio Preto. Realizou doutorado sanduíche na SUNY – College of Environmental Science and Forest. Atualmente é Professor Associado II no Departamento de Ciências Ambientais, UFSCAR-Sorocaba e trabalha na linha de pesquisa de ecologia de comunidades, metacomunidades, macroecologia, biogeografia e história natural de anfíbios.

Site: http://fernandoecologia.wix.com/fernandorodrigues

Thiago Gonçalves-Souza

Capixaba de Cachoeiro do Itapemirim/ES, pai do Lucas, companheiro da Natália, praticante de Muay Thai, Boxe e da arte de provar cervejas. Formado em Ciências Biológicas na Escola Superior São Francisco de Assis/ES e com mestrado e doutorado em Biologia Animal pela UNESP-São José do Rio Preto. Realizou doutorado sanduíche na University of Guelph (Canadá) e pós-doutorado na UNICAMP. Atualmente é Professor Adjunto III no Departamento de Biologia da Universidade Federal Rural de Pernambuco, trabalhando nas linhas de pesquisa de ecologia de comunidades, ecologia funcional, macroecologia e metacomunidades.

Site: https://thiagocalvesouza.wixsite.com/ecofun

Gustavo Brant Paterno

Compositor de cantigas e violeiro, nascido em Ribeirão Preto/SP, pai do Rudá e da Maria Flor, e apaixonado por Mila. Admira profundamente a incrível diversidade da vida na Terra. Gosta de trilha, música, skate, surf, fotografia de estrelas e programação. É Ecólogo com muito orgulho pela formação na UFRN/Natal, com mestrado e doutorado em Ecologia pela mesma universidade. Realizou sanduíche na Macquarie University (Austrália). Possui grande interesse em Ciência aberta e software livre e é embaixador do projeto Open Science Framework (OSF). Atualmente é pesquisador de pós-doutorado na Faculty of Forest Sciences and Forest Ecology da Universidade de Gottingen (Alemanha). Atua na interface entre ecologia evolutiva, biodiversidade-funcionamento de ecossistemas e ecologia da restauração.

Site: https://gustavopaterno.netlify.app/

Diogo Borges Provete

Capixaba da gema, nascido em Bom Jesus do Norte/ES (na verdade registrado nessa cidade e nascido em Bom Jesus do Itabapoana/RJ do outro lado da ponte, porque no ES não tinha hospital), pai orgulhoso da Manuela e esposo apaixonado da Lilian. Apreciador das boas Ale Belgas e Inglesas e vinhos Chilenos e Brasileiros. Já fui nadador, jogador de futebol, e um péssimo jogador de Xadrez. Dizem que gosto de livros e HQs. Graduado em Ciências Biológicas na Universidade Federal de Alfenas-MG, com mestrado em Biologia Animal na UNESP-São José do Rio Preto e Doutorado em Ecologia e Evolução na Universidade Federal de Goiás. Atualmente é Professor Adjunto I na Universidade Federal de Mato Grosso do Sul. Meu programa de pesquisa tenta integrar evolução fenotípica e processos micro- e macroevolutivos para entender padrões de distribuição de espécies em escala de metacomunidades, especialmente em ambientes de água doce.

Site: http://diogoprovete.weebly.com

Maurício Humberto Vancine

Caipira do interior de Socorro/SP, pai do Dudu, companheiro da Japa, amante de música instrumental, livros, games, softwares livres, programação, uma boa cerveja, além de um dedo de cachaça e uma longa prosa. Mais recentemente tenta não levar muitos tombos ao aprender a andar de skate depois dos 30. Graduado em Ecologia e mestre em Zoologia, ambos pela UNESP-Rio Claro. Atualmente é doutorando no PPG de Ecologia, Evolução e Biodiversidade da UNESP-Rio Claro e atua na linha de pesquisa de Ecologia Espacial, Ecologia da Paisagem, Modelagem Ecológica e Ecologia de Anfíbios.

Site: https://mauriciovancine.github.io/

Revisores e colaboradores

Expressamos nossos sinceros agradecimentos aos pesquisadores, alunos e colegas indicados abaixo, pelo imprescindível trabalho de avaliação, revisão e crítica do conteúdo deste livro.

- **André Padial** - Setor de Ciências Biológicas, Universidade Federal do Paraná, Curitiba, Paraná
- **Adriano Sanches Melo** - Instituto de Biociências, Universidade Federal do Rio Grande do Sul (UFRGS), campus do Vale, Porto Alegre, Rio Grande do Sul
- **Beatriz Milz** - Programa de Pós-Graduação em Ciência Ambiental, Instituto de Energia e Ambiente, Universidade de São Paulo (PROCAM/IEE/USP), São Paulo, São Paulo
- **Felipe Sodré Mendes Barros** - Departamento de Geografia, Instituto Superior Antonio Ruiz de Montoya, Instituto Misionero de Biodiversidad - IMiBio, Ambiental Analytics, Misiones, Argentina
- **Ingrid da Silva Lima** - Programa de Etnobiologia e Conservação da Natureza (PPGEtno), Universidade Federal Rural de Pernambuco (UFRPE), Recife, Pernambuco
- **Maurício Cetra** - Departamento de Ciências Ambientais, Universidade Federal de São Carlos (UFSCar), campus Sorocaba, São Paulo
- **Marcos Rafael Severgnini** - Programa de Pós Graduação em Ecologia e Conservação, Universidade Federal de Mato Grosso do Sul, Campo Grande, Mato Grosso do Sul
- **Marcos Robalinho Lima** - Departamento de Biologia Animal e Plantas, Universidade Estadual de Londrina, Londrina, Paraná
- **Michel Varajão Garey** - Instituto Latino Americano de Ciências da Vida e da Natureza (ILACVN), Universidade Federal da Integração Latino-Americana (UNILA), Foz do Iguaçu, Paraná
- **Paulo Mateus Martins Sobrinho** - Programa de Etnobiologia e Conservação da Natureza (PPGEtno), Universidade Federal Rural de Pernambuco (UFRPE), Recife, Pernambuco
- **Paulo Sérgio Monteiro Ferreira** - Secretaria do Estado da Educação da Paraíba, Paraíba
- **Pedro Henrique Albuquerque Sena** - Centro de Pesquisas Ambientais do Nordeste (Cepan), Recife, Pernambuco
- **Reginaldo Augusto Farias de Gusmão** - Programa de Etnobiologia e Conservação da Natureza (PPGEtno), Universidade Federal Rural de Pernambuco (UFRPE), Recife, Pernambuco
- **Victor Satoru Saito** - Departamento de Ciências Ambientais, Universidade Federal de São Carlos (UFSCar), campus São Carlos, São Paulo

Capítulo 1

Introdução

1.1 Histórico deste livro

Este livro foi estruturado a partir da apostila elaborada pelos pesquisadores Diogo B. Provete, Fernando R. da Silva e Thiago Gonçalves-Souza para ministrar o curso *Estatística aplicada à ecologia usando o R* no PPG em Biologia Animal da UNESP de São José Rio Preto/SP, em abril de 2011. Os três pesquisadores eram então alunos do PPG em Biologia Animal quando elaboraram o material disponibilizado na apostila Estatística aplicada à ecologia usando o R. A proposta de transformar a apostila em livro sempre foi um tópico recorrente desde 2011, e concretizado agora, um pouco mais de 10 anos depois.

Neste período, Diogo, Fernando e Thiago foram contratados pela Universidade Federal de Mato Grosso do Sul, Universidade Federal de São Carlos campus Sorocaba, e Universidade Federal Rural de Pernambuco, respectivamente. Nestes anos eles ofertaram diferentes versões do curso *Estatística aplicada à ecologia usando o R* para alunos de graduação e pós-graduação em diferentes instituições do Brasil. A possibilidade da oferta destes novos cursos fortaleceu a ideia de transformar a apostila em um livro com base nas experiências dos pesquisadores em sala de aula.

Considerando que novas abordagens ecológicas vêm sendo descritas e criadas a uma taxa elevada nos últimos anos, era de se esperar que as informações disponíveis na apostila estivessem defasadas após uma década. Por este motivo, Diogo, Fernando e Thiago convidaram outros dois pesquisadores, Gustavo B. Paterno da Georg-August-University of Göttingen e Maurício H. Vancine do PPG em Ecologia, Evolução e Biodiversidade da UNESP de Rio Claro/SP, que são referências no uso de estatística em ecologia usando o R. Com o time completo, passaram mais de um ano realizando reuniões, compartilhando scripts e pagando cerveja para os coautores por capítulos atrasados até chegarem nesta primeira versão do livro.

> **Importante**
> Este livro também pode ser acessado gratuitamente de forma online pelo link Análises Ecológicas no R. Da mesma forma, as respostas dos exercícios propostos em cada capítulo podem ser acessados no link Análises Ecológicas no R: Exercícios e Soluções. Destacamos ainda que devido a atualizações regulares dos pacotes, pode ser que algumas funções ou mesmo as saídas das funções mudem de estrutura, requerendo revisão dos códigos. Dessa forma, possíveis atualizações estarão disponíveis nesses links on-line.

1.2 Objetivo deste livro

Nossa proposta com este livro é produzir um conteúdo que possa ser utilizado tanto por quem quer se aprofundar em análises comumente utilizadas em ecologia, quanto por quem não tem nenhuma ou poucas habilidades quantitativas. Para isso, traçamos o melhor caminho, pelo menos do nosso ponto de vista, entre questões ecológicas e os métodos estatísticos mais robustos para testá-las. Guiar seus passos nesse caminho (nem sempre linear) necessita que você utilize um requisito básico: o de se esforçar para caminhar. O nosso esforço, em contrapartida, será o de indicar as melhores direções para que você adquira certa independência em análises ecológicas. Um dos nossos objetivos é mostrar que o conhecimento de teorias ecológicas e a formulação de questões apropriadas são o primeiro passo na caminhada rumo à compreensão da lógica estatística. Não deixe que a estatística

se torne a "pedra no seu caminho." Em nossa opinião, programas com ambiente de programação favorecem o entendimento da lógica estatística, uma vez que cada passo (lembre-se de que você está caminhando em uma estrada desconhecida e cheia de pedras) precisa ser coordenado, ou seja, as linhas de código (detalhes abaixo) precisam ser compreendidas para que você teste suas hipóteses. No entanto, tome cuidado ao copiar deliberadamente scripts sem entender cada um dos passos da análise ou gráfico realizado.

A primeira parte deste livro pretende utilizar uma estratégia que facilita a escolha do teste estatístico apropriado, por meio da seleção de questões/hipóteses claras e da ligação dessas hipóteses com a teoria e o método (veja Figura 2.1 no Capítulo 2). Enfatizamos que é fundamental ter em mente aonde se quer chegar, para poder escolher o que deve ser feito. Posteriormente à escolha de suas questões, é necessário transferir o contexto ecológico para um contexto meramente estatístico (hipótese nula/alternativa). A partir da definição de uma hipótese nula, partiremos para a aplicação de cada teste estatístico (de modelos lineares generalizados a análises multivariadas) utilizando a linguagem R.

Antes de detalhar cada análise estatística, apresentaremos o funcionamento básico da utilização da linguagem R e os tipos de distribuição estatística que são essenciais para a compreensão dos testes estatísticos. Para isso, organizamos um esquema que chamamos de "estrutura lógica" que facilita a compreensão dos passos necessários para testar suas hipóteses (veja Figura 2.1 no Capítulo 2) (Gonçalves-Souza, Provete, et al. 2019).

1.3 O que você não encontrará neste livro

Aprofundamento teórico, detalhes matemáticos e explicação dos algoritmos são informações que infelizmente não serão abordadas neste livro. O foco aqui é a explicação de como cada teste funciona (teoria e procedimentos matemáticos básicos) e sua aplicação em testes ecológicos usando scripts na linguagem R. Recomendamos aos (as) leitores (as) que consultem os livros indicados no final deste capítulo caso desejem maior aprofundamento teórico e prático.

1.4 Por que usar o R?

Os criadores do R o chamam de uma linguagem e ambiente de programação estatística e gráfica (Venables & Ripley 2002). A linguagem R também é chamada de programação "orientada ao objeto" (*object oriented programming*), o que significa que utilizar o R envolve basicamente a criação e manipulação de objetos em um terminal, em que o usuário tem de dizer exatamente o que deseja que o programa execute, ao invés de simplesmente clicar em botões. E vem daí uma das grandes vantagens em se usar o R: o usuário tem total controle sobre o que está acontecendo e também tem de compreender o que deseja antes de executar uma análise. Além disso, o R permite integração com outros programas escritos em Fortran, C++, Python e Java, permitindo que os usuários possam aplicar novas metodologias sem ter que aprender novas linguagens.

Na página pessoal do Prof. Nicolas J. Gotelli existem vários conselhos para um estudante iniciante de ecologia. Dentre esses conselhos, o Prof. Gotelli menciona que o domínio de uma linguagem de programação é uma das habilidades mais importantes, porque dá liberdade ao ecólogo para executar tarefas que vão além daquelas disponíveis em pacotes estatísticos comerciais. Além disso, a maioria das novas análises propostas nos mais reconhecidos periódicos em ecologia normalmente

são implementadas na linguagem R, e os autores geralmente incluem o código fonte no material suplementar dos artigos, tornando a análise acessível e reprodutível. A partir do momento que essas análises ficam disponíveis (seja por código fornecido pelo autor ou por implementação em pacotes pré-existentes), é mais simples entendermos a lógica de análises complexas, especialmente as multivariadas, utilizando nossos próprios dados, realizando-as passo a passo. Sem a utilização do R, normalmente temos que contatar os autores que nem sempre são tão acessíveis.

Especificamente em Ecologia, o uso da linguagem R para análise de dados cresceu enormemente nas duas últimas décadas. Em um artigo de revisão, Lai et al. (2019) analisaram mais de 60.000 artigos revisados por pares publicados em 30 periódicos de Ecologia durante um período de 10 anos. O número de estudos usando R aumentou linearmente de 11,4% em 2008 para 58,0% em 2017, e os 10 principais pacotes utilizados e ordenados por maior frequência de uso foram: `lme4`, `vegan`, `nlme`, `ape`, `MuMIn`, `MASS`, `mgcv`, `ade4`, `multcomp` e `car`. Os autores afirmam que a crescente popularidade do R promoveu a ciência aberta na pesquisa ecológica, melhorando a reprodutibilidade das análises e o fluxo de trabalho, principalmente quando scripts e códigos foram incluídos e compartilhados nos artigos. Eles finalizam dizendo que a partir dos resultados encontrados, a linguagem R é um componente significativo das análises no campo da Ecologia.

Uma última vantagem é que por ser um software livre, a citação do R em artigos é permitida e até aconselhável. Para saber como citar o R, digite `citation()` na linha de comando. Para citar um pacote específico, digite `citation()` com o nome do pacote entre aspas dentro dos parênteses. Mais detalhes sobre citações podem ser vistos no Capítulo 4. Neste ponto, esperamos ter convencido você leitor(a), de que aprender a utilizar o R tem inúmeras vantagens. Entretanto, provavelmente vai ser difícil no começo, mas continue e perceberá que o investimento vai valer à pena no futuro.

1.5 Indo além da linguagem de programação para a Ecologia

Um ponto em comum em que todos os autores deste livro concordaram em conversas durante sua estruturação, foi a dificuldade que todos tivemos quando estávamos aprendendo a linguagem:

1. Como transcrever os objetivos (manipulação de dados, análises e gráficos) em linguagem R
2. Como interpretar os resultados das análises estatísticas do R para os objetivos ecológicos

Num primeiro momento, quando estamos aprendendo a linguagem R é muito desafiador pensar em como estruturar nossos códigos para que eles façam o que precisamos: importar dados, selecionar linhas ou colunas, qual pacote ou função usar para uma certa análise ou como fazer um gráfico que nas nossas anotações são simples, mas no código parece impossível. Bem, não há um caminho fácil nesse sentido e ele depende muito da experiência e familiaridade adquirida com o tempo de uso da linguagem, assim como outra língua qualquer, como inglês ou espanhol. Entretanto, uma dica pode ajudar: estruture seus códigos antes de partir para o R. Por exemplo, escreva um papel os pontos que deseja executar em seus códigos, como se estivesse explicando para alguém os passos que precisa para realizar as tarefas. Depois disso, transcreva para o script (arquivo onde os códigos são escritos, mas não se preocupe, iremos explicar esse conceito no Capítulo 4) esses pontos em formato de texto. Por fim, traduza isso em linguagem R. Pode parecer maçante e cansativo no começo, mas isso o ajudará a ter maior domínio da linguagem, sendo que esse passo se tornará desnecessário quando se adquire bastante experiência.

Uma vez que esta barreira inicial foi transposta e você conseguiu obter os primeiros resultados de suas análises com valores de estatísticas, parâmetros estimados, valores de p e R^2, gráficos, e etc., como interpretamos à luz da teoria ecológica? Esse ponto é talvez um dos mais complicados. Com o tempo, ter um valor final de uma estatística ou gráfico à partir da linguagem R é relativamente simples, mas o que esse valor ou gráfico significam para nossa hipótese ecológica é o ponto mais complexo. Essa dificuldade por ser por inexperiência teórica (ainda não lemos muito sobre um aspecto ecológico) ou inexperiência científica (ainda temos dificuldade para expandir nossos argumentos de forma indutiva). Destacamos esse ponto porque ele é fundamental no processo científico e talvez seja o principal aspecto que diferencia os cientistas de outros profissionais: sua capacidade de entendimento dos padrões à partir dos processos e mecanismos atrelados. Nesse ponto, quase sempre recorremos aos nossos orientadores ou colegas mais experientes para nos ajudar, mas é natural e faz parte do processo de aprendizado de uso da linguagem R junto à Ecologia como Ciência. Entretanto, contrapomos a importância dessa extrapolação para não nos tornarmos apenas especialistas em linguagem R sem a fundamental capacidade de entendimento do sistema ecológico que estamos estudando.

1.6 Como usar este livro

Os conteúdos apresentados em cada capítulo são independentes entre si. Portanto, você pode utilizar este livro de duas formas. A primeira é seguir uma ordem sequencial (capítulos 1, 2, 3, ...) que recomendamos, principalmente, para as pessoas que não possuem familiaridade com a linguagem R. A segunda forma, é selecionar o capítulo que contém a análise de seu interesse e mudar de um capítulo para outro sem seguir a sequência apresentada no livro.

Com exceção dos capítulos 2, 3, 4, 5, 6 e 15, os outros capítulos foram elaborados seguindo a mesma estrutura, contendo uma descrição da análise estatística (aspectos teóricos) e exemplos relacionados com questões ecológicas que podem ser respondidas por esta análise e exercícios. Todos os exemplos são compostos por: i) uma descrição dos dados utilizados, ii) pergunta e predição do trabalho, iii) descrição das variáveis resposta(s) e preditora(s), e iv) descrição e explicação das linhas de código do R necessárias para realização das análises. A maioria dos exemplos utilizados são baseados em dados reais que já foram publicados em artigos científicos ou são dados coletados por um dos autores deste livro. Nós recomendamos que primeiro você utilize estes exemplos para se familiarizar com as análises e a formatação das linhas e colunas das planilhas. Em seguida, faça os exercícios propostos no final de cada capítulo, e por fim, utilize seus próprios dados para realizar as análises. Esta é a melhor maneira de se familiarizar com as linhas de código do R.

Importante

Muitas das métricas ou índices apresentados neste livro não foram traduzidas para o português, porque seus acrônimos são clássicos e bem estabelecidos na literatura ecológica. Nestes casos, consideramos que a tradução poderia confundir as pessoas que estão começando a se familiarizar com a literatura específica. Além disso, optamos por manter a versão padrão em alguns gráficos utilizados nos Capítulos 7 ao 15, principalmente aqueles gráficos que são "output" de análises como, por exemplo, visualização de normalidade de resíduos, homogeneidade de variâncias, entre outros. Em geral, esses gráficos são usados no processo de decisão de algum passo da análise e não possuem qualidade de publicação.

Como o usuário vai obter o mesmo gráfico quando replicar as análises propostas aqui ou suas próprias análises, julgamos ser mais didático manter a versão original, em inglês.

Realçamos que não estamos abordando todas as possibilidades disponíveis, e existem muitos outros pacotes e funções no R que realizam as mesmas análises. Contudo, esperamos que o conteúdo apresentado permita que os(as) leitores(as) adquiram independência e segurança para que possam caminhar sozinhos(as) na exploração de novos pacotes e funções para responderem suas perguntas biológicas e ecológicas.

1.7 Como ensinar e aprender com esse livro

Uma forma bastante interessante de aprender ou aprofundar seu conhecimento sobre um tema é a partir de grupos de estudo. Aproveitando as dinâmicas de estudos que os próprios autores fizeram em seus laboratórios (seja como discente ou professor), sugerimos abaixo alguns formatos que podem ser usados por um grupo de discentes (sem a presença de um orientador) ou pelo laboratório. É importante ressaltar que esses formatos não são os únicos que podem ser testados. O leitor pode juntar ideias de diferentes propostas ou mesmo usar parte das propostas e inserir suas próprias ideias, tendo como base as características do grupo que irá se reunir.

1.7.1 Em laboratórios ou grupos de pesquisa

Líder aleatório

Cada capítulo é sorteado para um integrante do grupo que ficará responsável por estudar, apresentar e enviar outros materiais que julgar necessário. Neste formato, existem duas possibilidades interessantes. A primeira é de um grupo de estudantes que é iniciante em determinado tema (e.g., análise multivariada) e, desse modo, todos integrantes serão estimulados a participarem do processo de ensino e de aprendizagem. O segundo ponto interessante é para grupos heterogêneos onde pessoas diferentes possuem domínio de diferentes ferramentas. Neste caso, é importante que mesmo que determinado integrante seja especialista na análise X, ele poderá aleatoriamente ter que aprender e ensinar a análise Y. Como resultado, espera-se que os grupos de estudo neste formato tenham ampla discussão, uma vez que integrantes com baixo, médio ou alto conhecimento em determinada análise serão tanto professores como aprendizes.

Líder especialista I (discentes como líderes)

Cada capítulo é liderado pelo "maior especialista" naquele determinado assunto, que ficará responsável por organizar toda dinâmica do grupo. O ideal é que especialistas distintos liderem a discussão de diferentes capítulos, para que todos os membros do grupo sejam líderes em no mínimo um capítulo.

Líder especialista II (orientador ou pós-doc)

O orientador (ou pós-doc ou ambos) selecionam os capítulos sobre o assunto de interesse (ou todos os capítulos do livro) e se reúne regularmente para discussão com discentes. Além da leitura dos capítulos, o líder pode enviar atividades extras ou desafios para estimular que os discentes leiam o

conteúdo e também executem comandos no R. Por exemplo, em cada capítulo, o desafio pode ser criar hipóteses sobre um tema de estudo, gerar dados fictícios (ou usar dados reais disponíveis) e analisar os dados com determinado teste estatístico no R.

Líder especialista I ou II integrando com teoria (específico para capítulos 8 a 15)

Esta proposta expande o Líder especialista I uma vez que não estará focada somente na análise estatística apresentada no livro. Além de inserir este componente analítico, o líder irá apresentar o arcabouço teórico ecológico que é geralmente usado em estudos que utilizaram determinada análise. Por exemplo, o capítulo que apresenta regressão linear seria combinado com a teoria de biogeografia de ilhas para entender a relação espécies-área. Desse modo, não seria apresentado somente as especificidades da regressão linear, mas também a teoria de biogeografia de ilhas.

1.7.2 Em disciplinas da graduação ou pós-graduação

Atividade em grupo em sala invertida

O professor pode sortear diferentes grupos que ficarão responsáveis por cada capítulo do livro (a depender do conteúdo da disciplina). Cada componente do grupo pode ficar responsável por diferentes partes do capítulo. Por exemplo, se a disciplina for de gráficos, um discente pode discutir a estrutura das funções do pacote `ggplot2`, outro discente pode apresentar a conexão entre tipos de variáveis e gráficos, enquanto um terceiro discente se responsabiliza por executar os comandos dos gráficos no R. As atividades devem ser realizadas e apresentadas antes da aula teórica/prática sobre aquela temática, e o docente ficará responsável por mediar as apresentações e discussões.

Sala convencional

O professor pode usar o livro como material didático seguindo o conteúdo de acordo com a disciplina em questão, seja ela da linguagem R, de análises univariadas, multivariadas ou espaciais. Além disso, o professor pode fornecer dados para os discentes (ou estimular que discentes usem os próprios dados) e replicar gráficos e análises usando os scripts fornecidos no livro.

1.8 Livros que recomendamos para aprofundamento teórico

- **Hands-On Programming with R (Grolemund 2014)**: Esse livro é para quem quer se aprofundar e aprender a programar em R, com exemplos práticos. Nas palavras do autor, ele disse que escreveu o livro para não programadores, com o intuito de fornecer uma introdução amigável à linguagem R. Nele, é apresentado como carregar dados, montar e desmontar objetos de dados, navegar no sistema de ambiente do R, escrever funções e usar as ferramentas de programação do R para a solução de problemas práticos de ciência de dados. O livro está disponível nesse link.
- **R for Data Science: Import, Tidy, Transform, Visualize, and Model Data (Wickham & Grolemund 2017)**: também conhecido como R4DS, esse livro é uma das primeiras referências sobre *tidyverse* e de Ciência de Dados no R. O livro aborda as principais etapas de importação, conversão, exploração e modelagem de dados e comunicação dos resultados. Ele apresenta uma compreensão do ciclo da ciência de dados, juntamente com as ferramentas básicas necessárias para gerenciar os detalhes sobre cada etapa do ciclo.

Cada seção do livro é combinada com exercícios para ajudar na fixação do conteúdo. Os principais tópicos são: i) transformar conjuntos de dados em um formato conveniente para análise, ii) programação com ferramentas poderosas do R para resolver problemas de dados com maior clareza e facilidade, iii) examinar os dados, gerar hipóteses e testá-las rapidamente, iv) gerar modelos que forneçam um resumo dos dados e que capture "sinais" no conjunto de dados, e v) aprendar R Markdown para integrar texto, código e resultados. O livro está disponível nesse link.

- **A primer of ecological statistics, 2ª ediçaõ (Gotelli & Ellison 2012)**: este livro traz um apanhado geral sobre desenhos amostrais voltados para experimentação e uma introdução à estatística multivariada. Existe uma tradução para o português da primeira edição, chamada "Princípios de Estatística em Ecologia" que saiu pela ed ArtMed em 2010. Este é uma excelente referência para quem quer começar a estudar estatística básica, especialmente com aplicações em ecologia.

- **Experimental Design and Data Analysis for Biologists (Quinn & Keough 2002)**: Outro excelente livro introdutório sobre estatística com exemplos práticos para ecologia e um dos preferidos dos autores deste livro aqui. Ele aborda os modelos lineares mais comuns vistos em disciplinas de bioestatística, tais como regressão e ANOVA, mas também traz uma boa introdução sobre GLMs e métodos mais modernos de análise de dados. Mas o mais importante, a lógica de ensino dos métodos segue muito o que preconizamos neste livro e não podemos recomendá-lo o bastante para quem está começando a estudar estatística.

- **The R book, 2ª edição (Crawley 2012)**: Livro que vai do básico ao avançado, tem informações sobre linguagem R, estatística univariada, multivariada e modelagem. Relativamente fácil de compreender. Capítulos trazem funções para criação e manipulação de gráficos passo-a-passo.

- **Numerical ecology, 3 ª edição (Legendre & Legendre 2012)**: Este é o manual teórico essencial e leitura obrigatória para entender mais a fundo qualquer análise multivariada. Esta nova edição traz um capítulo novo sobre análises multiescalares em ecologia de comunidade, com exemplos de aplicação de *Moran Eigenvector Maps* (MEMs).

- **Biological Diversity: Frontiers in Measurement and Assessment (Magurran & McGill 2012)**: Livro editado com vários capítulos sobre medidas tradicionais e alternativas de biodiversidade. Também atualiza medidas de estimativa de diversidade, uma revisão sobre diversidade funcional e filogenética. Esse é uma boa porta de entrada para entender os aspectos teóricos e meandros da análise de dados de biodiversidade.

- **Mixed effects models and extensions in ecology with R (Zuur et al. 2009)**: Este continua sendo a melhor introdução para modelos lineares generalizados (e de efeito misto), modelos de mínimos quadrados generalizados, Modelos Aditivos Generalizados para biólogos e ecólogos. O livro contém vários capítulos em que o funcionamento dos modelos é explicado de maneira bastante atraente, mantendo a matemática no mínimo. Todos os exemplos são com dados reais produzidor por ecólogos. Este é um bom livro intermediário para quem quer se aprofundar nas análises mais modernas feitas em ecologia.

- **Geocomputation with R (Lovelace 2020)**: Esse livro tornou-se rapidamente a principal referência sobre manipulação, visualização, análise e modelagem de dados geoespaciais no R. O livro é dividido em três partes: i) fundamentos, ii) extensões e iii) aplicações. A

parte um é voltada para a fundamentação dos dados geográficos no R, descrevendo a natureza dos conjuntos de dados espaciais e métodos para manipulá-los, assim como a importação/exportação de dados geográficos e a transformação de sistemas de referência de coordenadas. A Parte II representa métodos que se baseiam nessas fundações, abrange a criação de mapas avançados (incluindo mapeamento da web), "pontes" para GIS, compartilhamento de código reproduzível e como fazer validação cruzada na presença de autocorrelação espacial. A Parte III aplica o conhecimento adquirido para resolver problemas do mundo real, incluindo representação e modelagem de sistemas de transporte, localização ideal para lojas ou serviços e modelagem ecológica. Os exercícios no final de cada capítulo fornecem as habilidades necessárias para lidar com uma série de problemas geoespaciais. As soluções para cada capítulo e materiais complementares estão disponíveis nesse link e o livro nesse link.

Recomendamos ainda para o amadurecimento em análises ecológicas as seguintes leituras: Manly (1991), Pinheiro & Bates (2000), Scheiner & Gurevitch (2001), Burnham & Anderson (2014), Venables & Ripley (2002), Zar (2010), Zuur et al. (2007), Zuur et al. (2010), James et al. (2013), Fox et al. (2015), Thioulouse et al. (2018) e Touchon (2021).

Capítulo 2

Voltando ao básico: como dominar a arte de fazer perguntas cientificamente relevantes

Capítulo originalmente publicado por Gonçalves-Souza, Provete, Garey, da Silva & Albuquerque (2019), in Methods and Techniques in Ethnobiology and Ethnoecology (publicação autorizada por Springer, licença 5230220198680).

2.1 Introdução

Aquele que ama a prática sem teoria é como um marinheiro que embarca em um barco sem um leme e uma bússola e nunca sabe onde pode atracar - Leonardo da Vinci.

Qual é a sua pergunta? Talvez esta seja a frase que pesquisadores mais jovens ouvem quando começam suas atividades científicas. Apesar de aparentemente simples, responder a esta pergunta se torna um dos maiores desafios da formação científica. Seja na pesquisa quantitativa ou qualitativa, todo processo de busca de conhecimento parte de uma questão/problema formulada pelo pesquisador no início desse processo. Esta questão guiará o pesquisador em todas as etapas da pesquisa. No caso específico de pesquisa quantitativa, a questão é a porta de entrada de uma das formas mais poderosas de pensar cientificamente: o método hipotético-dedutivo (MHD) definido por Karl Popper (1959). Este capítulo propõe uma maneira de pensar sobre hipóteses (geradas dentro do MHD) para melhorar o pensamento estatístico usando um fluxograma que relaciona variáveis por ligações causais. Além disso, argumentamos que você pode facilmente usar fluxogramas para: i) identificar variáveis relevantes e como elas afetam umas às outras; ii) melhorar (quando necessário) o desenho experimental/observacional; iii) facilitar a escolha de análises estatísticas; e iv) melhorar a interpretação e comunicação dos dados e análises.

2.2 Perguntas devem preceder as análises estatísticas

2.2.1 Um bestiário[1] para o teste de hipóteses (Você está fazendo a pergunta certa?)

A maioria dos alunos e professores de ciências biológicas possuem aversão à palavra "estatística." Não surpreendentemente, enquanto a maioria das disciplinas acadêmicas que compõem o "STEM" (termo em inglês para aglomerar *Ciência, Tecnologia, Engenharia e Matemática*) têm uma sólida formação estatística durante a graduação, cursos de ciências biológicas têm um currículo fraco ao integrar o pensamento estatístico dentro de um contexto biológico (Metz 2008). Esses cursos têm

1 O bestiário é uma literatura do Século XII que descrevia animais (reais ou imaginários) com uma visão divertida e fantasiosa. Parte da interpretação continha lições de moral dos monges católicos que escreviam e ilustravam os bestiários. Longe de ser uma lição de moral ou mesmo uma descrição fantasiosa, usamos o termo bestiário neste livro para expressar o que há de mais importante e fantástico (na visão dos autores) para fazer um bom teste de hipóteses.

sido frequentemente ministrados sem qualquer abordagem prática para integrar os alunos em uma plataforma de solução de problemas (Horgan et al. 1999). Infelizmente, a Etnobiologia, Ecologia e Conservação (daqui em diante EEC) não são exceções. Talvez mais importante, uma grande preocupação durante o treinamento estatístico de estudantes de EEC é a necessidade de trabalhar com problemas complexos e multidimensionais que exigem soluções analíticas ainda mais complicadas para um público sem experiência em estatística e matemática. Por este motivo, muitos pesquisadores consideram a estatística como a parte mais problemática de sua pesquisa científica. Argumentamos neste capítulo que a dificuldade de usar estatística em EEC está associada à ausência de uma plataforma de solução de problemas gerando hipóteses claras que são derivadas de uma teoria. No entanto, concordamos que há um grande desafio em algumas disciplinas como a Etnobiologia para integrar esta abordagem direcionada por hipóteses, uma vez que foi introduzida apenas recentemente – veja Phillips & Gentry (1993) e Albuquerque & Hanazaki (2009). Devido à falta de uma plataforma de solução de problemas, frequentemente percebemos que alunos/pesquisadores na EEC geralmente têm dificuldades de responder perguntas básicas para uma pesquisa científica, tais como:

1. Qual é a principal teoria ou raciocínio lógico do seu estudo?
2. Qual é a questão principal do seu estudo?
3. Qual é a sua hipótese? Quais são suas predições?
4. Qual é a unidade amostral, variável independente e dependente do seu trabalho? Existe alguma covariável?
5. Qual é o grupo controle?

Como selecionar qualquer teste estatístico sem responder a essas cinco perguntas? A estrutura estatística frequentista fornece uma maneira de ir progressivamente suportando ou falseando uma hipótese (Neyman & Pearson 1933, Popper 1959). A decisão de rejeitar uma hipótese nula é feita usando um valor de probabilidade (geralmente $P < 0,05$) calculado pela comparação de eventos observados com observações repetidas obtidas a partir de uma distribuição nula.

Agora, vamos ensinar através de um exemplo e apresentar um "guia para o pensamento estatístico" que conecta alguns elementos essenciais para executar qualquer análise multivariada (ou univariada)(Figura 2.1). Primeiro, imagine que você observou os seguintes fenômenos na natureza: i) "indivíduos de uma população tradicional selecionar algumas plantas para fins médicos" e ii) "manchas monodominantes da árvore *Prosopis juliflora*, uma espécie invasora em várias regiões." Do lado da etnobiologia, para entender como e porque o conhecimento tradicional é construído, existe uma teoria ou hipótese (por exemplo, hipótese de aparência: Gonçalves et al. 2016) explicando os principais processos que ditam a seleção da planta (Figura 2.1a). Então, você pode fazer uma ou mais perguntas relacionadas àquele fenômeno observado (Figura 2.1b). Por exemplo, como a urbanização afeta o conhecimento das pessoas sobre o uso de plantas medicinais em diferentes biomas? Do lado ecológico/conservação, para entender por que espécies introduzidas afetam as espécies nativas locais, você precisa entender as teorias do nicho ecológico e evolutiva (MacDougall et al. 2009, Saul & Jeschke 2015). Você pode perguntar, por exemplo, como as plantas exóticas afetam a estrutura de comunidades de plantas nativas? Questões complexas ou vagas dificultam a construção do fluxograma de pesquisa (ver descrição abaixo) e a seleção de testes estatísticos. Em vez disso, uma pergunta útil deve indicar as variáveis relevantes do seu estudo, como as independentes e dependentes, covariáveis, unidade amostral e a escala espacial de interesse (Figura 2.1b). No exemplo etnobiológico fornecido, a urbanização e o conhecimento das pessoas são as variáveis independentes e dependentes, respectivamente. Além disso, este estudo tem uma escala ampla, pois compara biomas diferentes. A próxima etapa é construir a hipótese biológica (Figura 2.1c), que indicará a associação entre variáveis independentes e dependentes. No exemplo etnobiológico, a hipótese é que i) "a urbanização afeta o conhecimento das pessoas sobre o uso de plantas medicinais," enquanto a

hipótese ecológica é que ii) "espécies exóticas afetam a estrutura de comunidades de plantas nativas." Observe que isso é muito semelhante à questão principal. Mas você pode ter múltiplas hipóteses (Platt 1964) derivado de uma teoria. Depois de selecionar a hipótese biológica (ou científica), é hora de pensar sobre a derivação lógica da hipótese, que é chamada de predição ou previsão (Figura 2.1d). Os padrões preditos são uma etapa muito importante, pois após defini-los você pode operacionalizar suas variáveis e visualizar seus dados. Por exemplo, a variável teórica "Urbanização" pode ser medida como "grau de urbanização ao longo das áreas urbanas, periurbanas e rurais" e "conhecimento das pessoas" como "o número e tipo de espécies de plantas úteis usadas para diferentes doenças." Assim, a predição é que o grau de urbanização diminua o número e tipo de espécies de plantas conhecidas utilizadas para fins medicinais. No exemplo ecológico, a variável "espécies exóticas" pode ser medida como "a densidade da planta exótica *Prosopis juliflora*" e "Estrutura da comunidade" como "riqueza e composição de espécies nativas." Depois de operacionalizar o seu trabalho à luz do método hipotético-dedutivo (HDM), o próximo passo é "pensar estatisticamente" sobre a hipótese biológica formulada (ver Figura 2.1e, f).

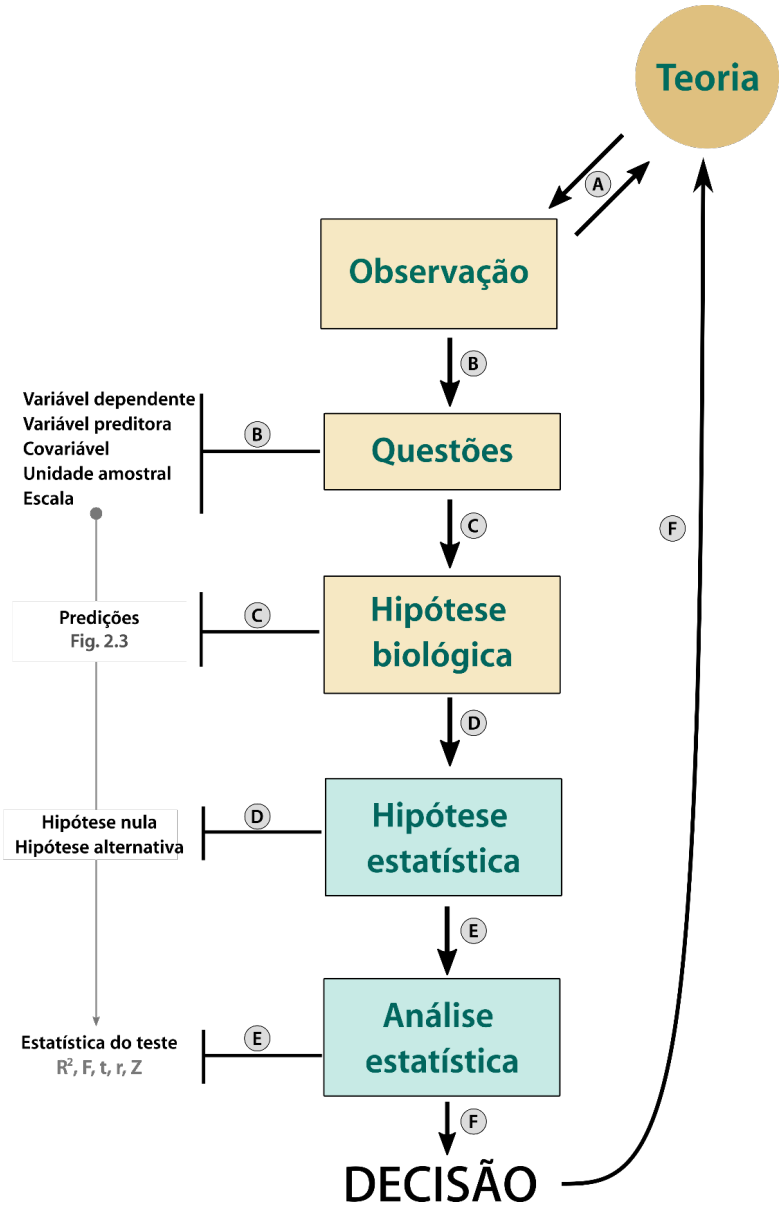


Figura 2.1: Um guia para o pensamento estatístico combinando o método hipotético-dedutivo (a – d, i) e estatística frequentista (e – i). Veja também a Figura 1 em Underwood (1997), Figura 1 em Ford (2000) e Figura 1.3 em Legendre & Legendre (2012).

Então, você precisa definir as hipóteses estatísticas nula (H0) e alternativa (H1). Duas "hipóteses estatísticas" diferentes podem ser derivadas de uma hipótese biológica (Figura 2.1e). Portanto, nós usamos o termo "hipótese estatística" entre aspas, porque as chamadas hipóteses estatísticas são predições *sensu stricto*, e muitas vezes confundem jovens estudantes. A hipótese estatística nula representa uma ausência de relação entre as variáveis independentes e dependentes. Depois de definir a hipótese estatística nula, você pode derivar uma ou várias hipóteses estatísticas alternativas, que demonstram a(s) associação(ões) esperada(s) entre suas variáveis (Figura 2.1e). Em nosso exemplo, a hipótese nula é que "o grau de urbanização não afeta o número de espécies de plantas úteis conhecidas pela população local." Por sua vez, a hipótese alternativa é que "o grau de urbanização afeta o número de espécies de plantas úteis conhecidas pela população local." Depois de operacionalizar suas variáveis e definir o valor nulo e hipóteses alternativas, é hora de visualizar o resultado esperado (Figura 2.2, Caixa 1) e escolher um método estatístico adequado. Por exemplo, se você deseja comparar a diferença na composição de plantas úteis entre áreas urbanas, periurbanas e rurais, você pode executar uma PERMANOVA (Gonçalves-Souza, Garey, et al. 2019) que usa uma estatística de teste chamada *pseudo-F*. Então, você deve escolher o limite de probabilidade (o valor P) do teste estatístico para decidir se a hipótese nula deve ou não deve ser rejeitada (Gotelli & Ellison 2012). Se você encontrar um $P < 0,05$, você deve rejeitar a hipótese estatística nula (urbanização não afeta o número e a composição das plantas). Por outro lado, um $P > 0,05$ indica que você não pode rejeitar a hipótese nula estatística. Assim, a estatística do teste e o valor P representam a última parte do teste de hipótese estatística, que é a decisão e conclusões apropriadas que serão usadas para retroalimentar a teoria principal (Figura 2.1g – i). Generalizando seus resultados e falseando (ou não) suas hipóteses, os estudos buscam refinar a construção conceitual da teoria, que muda constantemente (Figura 1i, Ford 2000). No entanto, há um ponto crítico nesta última frase, porque a significância estatística não significa necessariamente relevância biológica [ver discussão em Gotelli e Ellison (2012) e Martínez-Abraín (2008)]. Nas palavras de Ford (2000): "as estatísticas são usadas para iluminar o problema, e não para apoiar uma posição." Além disso, o procedimento de teste de hipótese tem alguma incerteza, que pode influenciar resultados "falso-positivos" (erro tipo 1) e "falso-negativos" (erro tipo 2) (Whitlock & Schluter 2015). Para simplificar, não discutiremos em detalhes os prós e contras da estatística frequentista, bem como métodos alternativos (por exemplo, Bayesiana e Máxima Verossimilhança), e questões filosóficas relativas ao "valor P" (para uma discussão sobre esses tópicos, consulte o fórum em Ellison et al. (2014).

Caixa 1. Tipo de variáveis e visualização de dados. Conforme descrito na Seção 2.3, o fluxograma é essencial para conectar variáveis relevantes para a pesquisa. Para aproveitar ao máximo esta abordagem, você pode desenhar suas próprias predições gráficas para te ajudar a pensar sobre diferentes possibilidades analíticas. Aqui, nós fornecemos uma descrição completa dos tipos de variáveis que você deve saber antes de executar qualquer análise estatística e representar seus resultados. Além disso, mostramos uma breve galeria (Figura 2.2) com exemplos de boas práticas em visualização de dados (Figura 2.3b, veja também figuras em Gonçalves-Souza, Garey, et al. 2019). Além de conectar diferentes variáveis no fluxograma, você deve distinguir o tipo de variável. Primeiro você deve identificar as variáveis independentes (também conhecidos como explicativas ou preditoras) e dependentes (também conhecidas como resposta). A variável independente é aquela (ou aquelas) que prevê ou afeta a variável resposta (por exemplo, a fertilidade do solo é a variável independente capaz de afetar a abundância de uma espécie de planta focal, a variável dependente). Além disso, uma covariável é uma variável contínua que pode

afetar tanto a variável resposta quanto a independente (ou ambos), mas geralmente não é do interesse do pesquisador. Depois de definir as variáveis relevantes, conectando-as no fluxograma, é hora de diferenciar seu tipo: i) quantitativa ou contínua, e ii) categórica ou qualitativa (Figura 2.2, Caixa 1). O tipo de variável irá definir que tipo de figura você pode selecionar. Por exemplo, se você está comparando duas variáveis contínuas ou uma variável contínua e uma binária, a melhor maneira de visualizá-los (Figura 2.2) é um gráfico de dispersão (Figura 2.2). A linha representa os valores preditos pelo modelo estatístico usado (por exemplo, linear ou logístico). Se você está interessado em comparar a gama de diferentes atributos (ou a descrição de qualquer variável numérica) entre as variáveis categóricas (por exemplo, espécies ou populações locais), um gráfico de halteres (do inglês Dumbbell plot) é uma boa opção (Figura 2.2). Histogramas também podem ser usados para mostrar a distribuição de duas variáveis contínuas de dois grupos ou fatores (Figura 2.2). No entanto, se você quiser testar o efeito de uma variável categórica independente (como em um desenho de ANOVA) sobre uma variável dependente, boxplots (Figura 2.2) ou gráficos de violino podem resumir essas relações de maneira elegante. Conjuntos de dados multivariados, por sua vez, podem ser visualizados com ordenação (Figura 2.2) ou gráficos de agrupamento (não mostrados). Existe um site abrangente apresentando várias maneiras de visualizar dados chamado datavizproject.

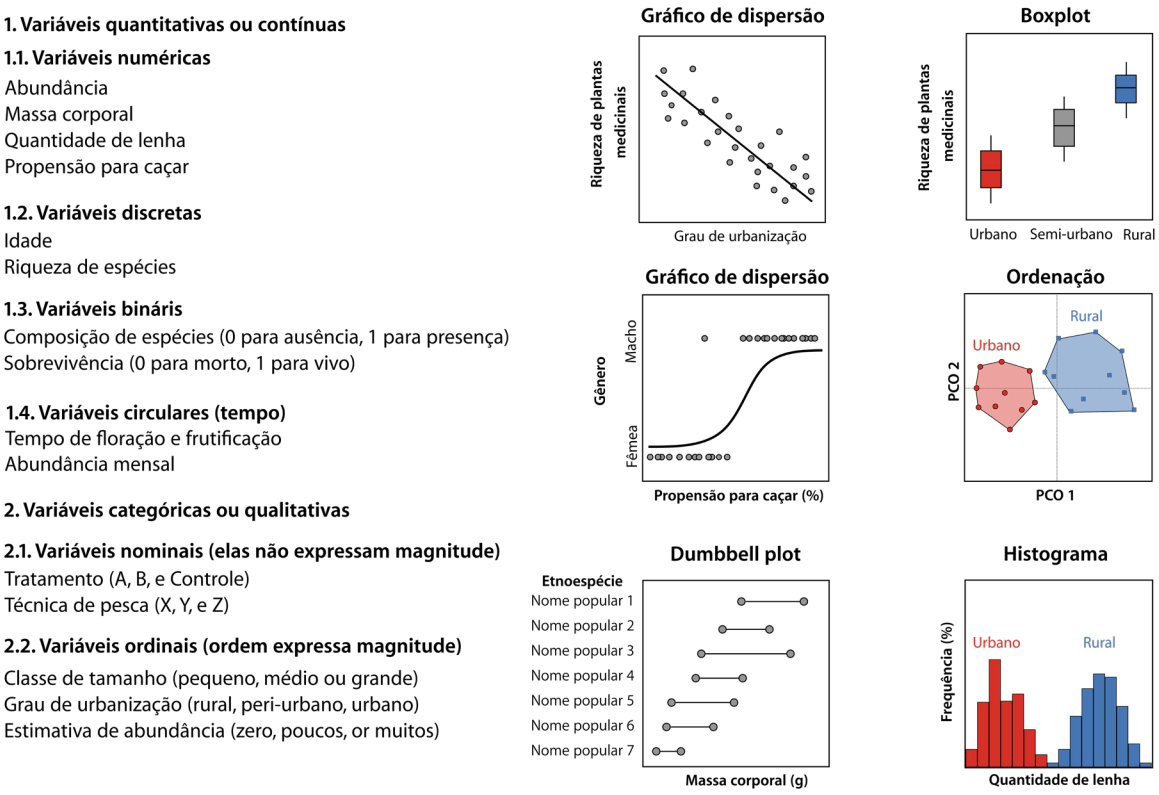


Figura 2.2: (A) Tipos de variáveis e (B) visualização de dados para representar a relação entre variáveis independentes e dependentes ou covariáveis.

2.3 Fluxograma: Conectando variáveis para melhorar o desenho experimental e as análises estatísticas

McIntosh e Pontius (2017) afirmaram que o pensamento estatístico representado na Figura 2.1 inclui quatro etapas importantes: i) quais perguntas você investigaria (Seção 2.4), ii) como e onde coletar os dados (Ruxton & Colegrave, 2016), iii) quais fatores devem ser considerados e como eles afetam suas variáveis de interesse (e como elas afetam umas às outras), e iv) qual análise estatística você deve usar e como interpretar e comunicar os resultados (Seção 2.4). No entanto, a etapa (3) deve ser feita antes de coletar os dados. Por exemplo, se você está interessado na investigação dos benefícios das matas ciliares para as espécies nativas de peixes, quais variáveis devem ser incluídas no estudo? Se você escolher rios com e sem mata ciliar como única variável preditora, seu projeto de amostragem irá omitir outras variáveis de confusão, como ordem do rio e carbono orgânico do solo a montante. Vellend (2016) nomeou este problema como o "problema de três caixas", que se refere à limitação em inferir que X (variável independente) causa variação em Y (variável dependente) quando outras variáveis criam ou ampliam a correlação entre X e Y (ver Figura 2 em Ruxton & Colegrave 2016). Uma ferramenta útil para compreender a relação entre todas as variáveis relevantes do seu estudo é um fluxograma. No "fluxograma de pesquisa" (ver também Magnusson & Mourão 2004) proposto aqui, variáveis dependentes (também conhecidas como resposta) e independentes (ou preditora), bem como covariáveis são representadas como caixas (com formas distintas: Figura 2.3). Além disso, você pode usar uma seta para representar uma (possível) via causal indicando força e sinal (positivo ou negativo) da variável preditora na variável dependente (Figura 2.3). Ao fazer isso, você pode melhorar o desenho experimental ou observacional incluindo ou controlando variáveis de confusão o que, por sua vez, pode ajudar a separar a contribuição relativa de diferentes variáveis preditoras em seu sistema. Mais importante, fazer conexões entre variáveis melhora sua capacidade de visualizar o "Quadro geral" de sua pesquisa, o que pode afetar seu experimento, análise estatística e revisão da literatura. Na verdade, Arlidge et al. (2017) argumentam que fluxogramas facilitam a construção de narrativas, melhorando: i) a definição de múltiplas hipóteses, ii) coleta, interpretação e disseminação de dados e iii) a comunicação do conteúdo do estudo. Você também pode ler os livros de Magnusson et al. (2004) para entender mais como usar fluxogramas para auxiliar análises estatísticas e Ford (2000) que recomenda o uso de uma abordagem analítica para fomentar o desenvolvimento da pesquisa. Além disso, o fluxograma de pesquisa pode ser usado como uma ferramenta forte para contemplar os conselhos de Ford (2000), que foram: i) definir a pergunta da pesquisa, ii) definir a teoria a ser usada, iii) definir a técnica de investigação (por exemplo, experimento, observação de campo), iv) definir as medições, v) definir como fazer inferência, e vi) interpretar, generalizar e sintetizar a partir de dados que, por sua vez, são usados para refinar a teoria e modificar (quando necessário) questões futuras (Figura 2.1).

Pergunta principal: qual o impacto da invasão sobre a comunidade nativa e propriedades ecossistêmicas?

Prediction 1: A planta exótica *Prosopis juliflora* reduz a diversidade beta comunidades de plantas nativas

Prediction 2: *Prosopis juliflora* modifica a composição de plantas medicinais e reduz as taxas de decomposição e estoque de carbono

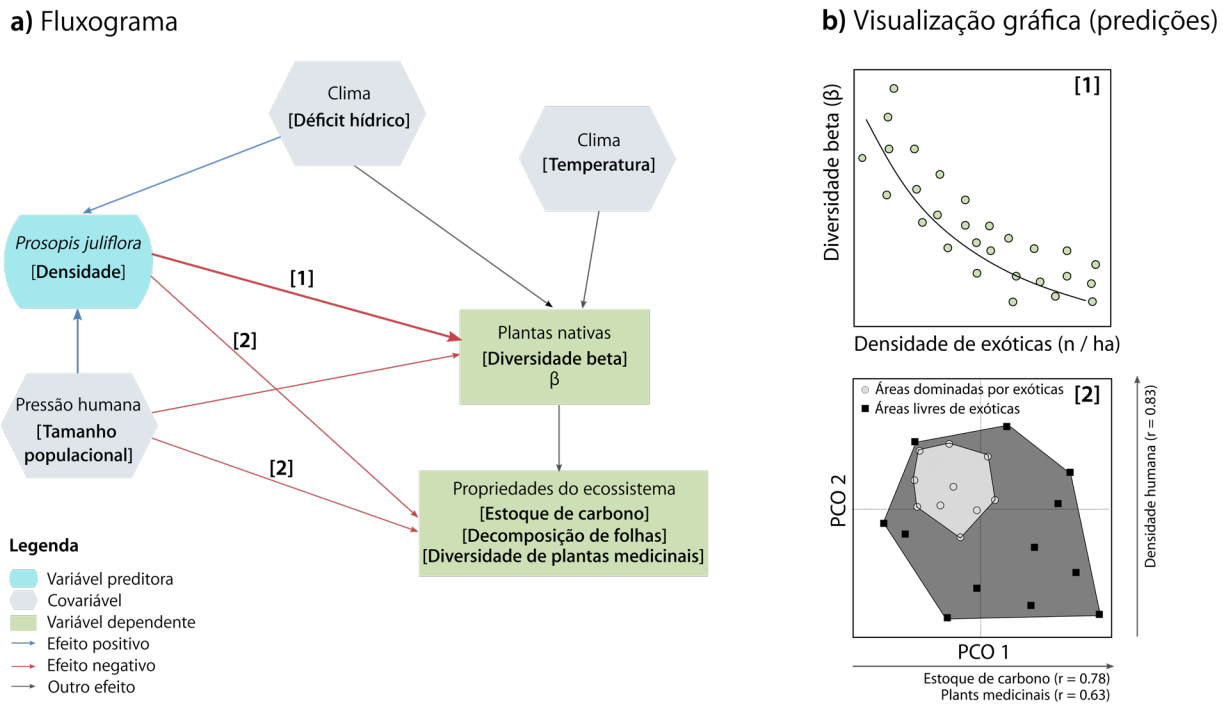


Figura 2.3: Exemplo de como usar um fluxograma para melhorar o entendimento do sistema estudado. A pergunta teórica "Qual é o impacto da invasão na comunidade nativa e nas propriedades do ecossistema?" pode gerar duas predições: i) a planta exótica Prosopis juliflora reduz a diversidade beta de comunidades de plantas nativas, e ii) Prosopis juliflora modifica a composição das comunidades de plantas e reduz o estoque de carbono e as taxas de decomposição. Após selecionar suas predições, você pode construir um fluxograma conectando as variáveis relevantes e as associações entre elas. Além disso, você pode usar as informações na **Caixa 1** *para identificar que tipo de variável você irá coletar e quais figuras podem ser usadas (b).*

2.4 Questões fundamentais em etnobiologia, ecologia e conservação

> *As teorias são generalizações. As teorias contêm perguntas. Para algumas teorias, as perguntas são explícitas e representam o que a teoria pretende explicar. Para outras, as questões são implícitas e se relacionam com a quantidade e tipo de generalização, dada a escolha de métodos e exemplos usados por pesquisadores na construção da teoria. As teorias mudam continuamente, à medida que exceções são encontradas às suas generalizações e como questões implícitas sobre método e opções de estudos são expostas. - E. David Ford (2000)*

Como argumentamos antes, uma questão relevante e testável precede as análises estatísticas. Assim, apresentamos a seguir 12 questões que podem estimular pesquisas futuras na ECC. Observe, no entanto, que não queremos dizer que eles são as únicas questões relevantes a serem testadas na EEC - ver, por exemplo, Sutherland et al. (2013) para uma avaliação completa da pesquisa de ponta em

Ecologia; e Caixa 6.1 em Pickett et al. (2007)[2]. Especificamente, essas questões são muito amplas e podem ser desenvolvidas em perguntas, hipóteses e predições mais restritas. Depois de cada questão teórica, apresentamos um estudo que testou essas hipóteses bem como as variáveis relevantes que podem estimular estudos futuros.

(a) Como o uso da terra afeta a manutenção da biodiversidade e a distribuição de espécies em diferentes escalas espaciais?

Exemplo: Vários estudos em diferentes ecossistemas e escalas investigaram como o uso da terra afeta a biodiversidade. No entanto, destacamos um estudo comparando os efeitos globais do uso da terra (por exemplo, densidade populacional humana, paisagem para usos humanos, tempo desde a conversão da floresta) em espécies terrestres (por exemplo, mudança líquida na riqueza local, dissimilaridade composicional média) (Newbold et al. 2015).

(b) Qual é o impacto da invasão biótica nas comunidades nativas e propriedades do ecossistema?

Exemplo: Investigar como o estabelecimento de espécies exóticas afetam a riqueza de espécies do receptor, comunidades nativas, bem como isso afeta a entrega dos serviços ecossistêmicos. Estudos anteriores controlaram a presença de espécies invasoras ou registros históricos comparados (estudos observacionais) dessas espécies e como elas impactam a biodiversidade. Além disso, há algum esforço em compreender os preditores de invasibilidade (por exemplo, produto interno bruto de regiões, densidade populacional humana, litoral continental e ilhas) (Dawson et al. 2017).

(c) Como o declínio do predador de topo afeta a entrega de serviços ecossistêmicos?

Exemplo: Investigar como a remoção de grandes carnívoros afeta o fornecimento de serviços ecossistêmicos, como o sequestro de carbono, doenças e controle de danos às colheitas. Estudos anteriores investigaram esta questão controlando a presença de predadores de topo ou comparando registros históricos (estudo observacionais) de espécies e vários preditores (por exemplo, perda e fragmentação de habitat, conflito entre humanos e espécies caçadas, utilização para a medicina tradicional e superexploração de presas) (Ripple et al. 2014).

(d) Como a acidificação dos oceanos afeta a produtividade primária e teias alimentares em ecossistemas marinhos?

Exemplo: Estudos recentes testaram os efeitos individuais e interativos da acidificação e do aquecimento do oceano nas interações tróficas em uma teia alimentar. A acidificação e o aquecimento foram manipulados pela mudança dos níveis de CO2 e temperatura, respectivamente. Estudos anteriores demonstraram que elevação de CO2 e temperatura aumentou a produtividade primária e afetou a força do controle de cima para baixo exercido por predadores (Goldenberg et al. 2017).

(e) Como podemos reconciliar as necessidades da sociedade por recursos naturais com conservação da Natureza?

Exemplo: Existe uma literatura crescente usando abordagens de paisagem para melhorar a gestão da terra para reconciliar conservação e desenvolvimento econômico. Os estudos possuem diversos objetivos, mas em geral eles usaram o engajamento das partes interessadas, apoio institucional, estruturas eficazes de governança como variáveis preditoras e melhorias ambientais (por exemplo,

2 Após a publicação original deste capítulo, Ulysses Albuquerque e colaboradores publicaram artigo sugerindo questões fundamentais em Etnobiologia: Albuquerque et al. 2019, Acta Bot. Bras. 33, 2.

conservação do solo e da água, cobertura vegetal) e socioeconômicas (renda, capital social, saúde pública, emprego) como variáveis dependentes (Reed et al. 2017).

(f) Qual é o papel das áreas protegidas (UCs) para a manutenção da biodiversidade e dos serviços ecossistêmicos?

Exemplo: Houve um trabalho considerável na última década comparando a eficácia das UCs para a conservação da biodiversidade. Embora esta questão não esteja completamente separada da questão anterior, o desenho dos estudos é relativamente distinto. Em geral, os pesquisadores contrastam o número de espécies e o fornecimento de serviços ecossistêmicos (por exemplo, retenção de água e solo, sequestro de carbono) entre áreas legalmente protegidas (UCs) e não protegidas (Xu et al. 2017).

(g) Como integrar o conhecimento científico e das pessoas locais para mitigar os impactos negativos das mudanças climáticas e do uso da terra na biodiversidade?

Exemplo: Eventos climáticos extremos podem ter forte impacto sobre rendimento agrícola e produção de alimentos. Autores recentes têm argumentado que esse efeito pode ser mais forte para os pequenos agricultores. Estudos futuros podem investigar como a precipitação e a temperatura afetam o rendimento agrícola e como os agricultores tradicionais ou indígenas lidam com esse impacto negativo. Sistemas de agricultura tradicional têm menor erosão do solo e emissões de N2O / CO2 do que as monoculturas e, portanto, podem ser vistos como uma atividade de mitigação viável em um mundo em constante mudança (Niggli et al. 2009, Altieri & Nicholls 2017).

(h) Como as mudanças climáticas afetam a resiliência e estratégias adaptativas em sistemas socioecológicos?

Exemplo: A mudança do clima altera tanto a pesca quanto a agricultura em todo o mundo, o que por sua vez obriga os humanos a mudar suas estratégias de cultivo. Estudos recentes têm argumentado que a agricultura em alguns países enfrentará riscos com as mudanças climáticas. Esses estudos comparam diferentes sistemas de produção, de agricultura convencional a outros tipos empregados por populações locais. Por exemplo, há uma forte conexão entre i) espécies ameaçadas e sobrepesca, ii) índice de desenvolvimento humano (IDH) e dependência média da pesca e aquicultura. Além disso, há evidências de que a biodiversidade pode amortecer os impactos das mudanças climáticas aumentando a resiliência da terra (Niggli et al. 2009, Altieri & Nicholls 2017, Blanchard et al. 2017). Uma abordagem interessante é investigar como as populações locais lidam com esses desafios em termos de percepções e comportamento.

(i) Como a invasão biológica afeta espacial e temporalmente a estrutura e funcionalidade dos sistemas sócio-ecológicos?

Exemplo: Muitos estudos demonstraram que espécies invasoras têm consequências biológicas, econômicas e sociais negativas. Aqui, da mesma forma que a pergunta B, os pesquisadores controlaram a presença de espécies invasoras ou utilizaram registros históricos. No entanto, trabalhos recentes quantificam não apenas a riqueza e composição de espécies nativas, mas também atributos funcionais de animais/vegetais que afetam diretamente o fornecimento de serviços ecossistêmicos como abastecimento (comida, água), regulação (clima, controle de inundações), suporte (ciclagem de nutrientes, formação do solo) e cultural (ecoturismo, patrimônio cultural) (Chaffin et al. 2016). Mas, espécies invasoras podem provocar efeitos positivos no sistema sócio-ecológico aumentando a disponibilidade de recursos naturais, impactando como as pessoas gerenciam e usam a biodiversidade local.

(j) Qual é a relação entre as diversidades filogenética e taxonômica com a diversidade biocultural?

Exemplo: Estudos recentes mostraram que existe um padrão filogenético e taxonômico nos recursos que as pessoas incorporam em seus sistemas sócio-ecológicos, especialmente em plantas medicinais. Existe uma tendência para as pessoas, em diferentes partes do mundo, para usar plantas próximas filogeneticamente para os mesmos propósitos. Aqui, os pesquisadores podem testar o quanto isso afeta a diversidade de práticas em um sistema sócio-ecológico considerando o ambiente, bem como sua estrutura e funções (Saslis-Lagoudakis et al. 2012, Saslis-Lagoudakis et al. 2014).

(k) Quais variáveis ambientais e sócio-políticas mudam a estrutura e funcionalidade dos sistemas sócio-ecológicos tropicais?

Exemplo: Testar a influência das mudanças ambientais afetadas pela espécie humana (por exemplo, fogo, exploração madeireira, aquecimento) em espécies-chave e, consequentemente, como esse efeito em cascata pode afetar outras espécies e serviços ecossistêmicos (por exemplo, armazenamento de carbono, ciclo da água e dinâmica do fogo) (Lindenmayer & Sato 2018).

(l) Os atributos das espécies influenciam como as populações locais distinguem plantas ou animais úteis e não-úteis?

Exemplo: Investigar se a população local possui preferência ao selecionar espécies de animais ou plantas. Você pode avaliar se grupos diferentes (por exemplo, turistas) ou populações locais (por exemplo, pescadores) selecionam espécies com base em atributos das espécies. Estudos recentes têm mostrado uma ligação potencial entre planta (por exemplo, cor, folha, floração) e pássaro (por exemplo, cor, vocalização) e alguns serviços culturais do ecossistema, como estética, recreativa e espiritual/religiosa (Goodness et al. 2016).

Como você notou, as questões eram mais teóricas e, consequentemente, você pode derivar predições testáveis (usando variáveis) a partir delas (Figuras 2.1, 2.3). Por exemplo, da questão *"Como o uso da terra afeta a manutenção da biodiversidade e distribuição de espécies em diferentes escalas?"* podemos derivar duas predições diferentes: i) densidade populacional (variável operacional de uso da terra) muda a composição de espécies e reduz a riqueza de espécies na escala da paisagem (predição derivada da hipótese da homogeneização biótica: Solar et al. 2015); ii) a composição dos atributos funcionais das plantas é diferente em remanescentes florestais com diferentes matrizes (cana-de-açúcar, gado, cidade, etc.).

2.5 Considerações Finais

Conte-me seus segredos
E faça-me suas perguntas
Oh, vamos voltar para o início
Correndo em círculos, perseguindo caudas
Cabeças em uma ciência à parte
Ninguém disse que seria fácil
(...) Desfazendo enigmas
Questões da ciência, ciência e progresso
- O Cientista, Coldplay

Este é um trecho de uma música da banda britânica de rock Coldplay, do álbum de 2002 *A Rush of Blood to the Head*. A letra é uma comparação incrível entre a ciência e os altos e baixos de um relacionamento fadado ao fracasso. A banda traz uma mensagem surpreendentemente clara de que como cientistas, nós deveríamos frequentemente fazer perguntas, voltar ao início após descobrir que estávamos errados (ou não) e que corremos em círculos tentando melhorar nosso conhecimento. A banda descreveu de uma forma tão precisa o quão cíclico (mas não repetitivo) é o método científico. Como disse a canção: não é fácil, mas aprender como fazer boas perguntas é um passo essencial para a consolidação do conhecimento. Ao incluir o teste de hipótese no EEC, podemos ser mais precisos. Definitivamente, isso não significa que a ciência descritiva seja inútil. Ao contrário, o desenvolvimento da ECC e principalmente da Etnobiologia, foi construído sobre uma linha de frente descritiva, o que significa que foi valioso para a fundação da Etnobiologia como disciplina consolidada (Ethnobiology Working Group 2003, Stepp 2005). No entanto, estudos recentes defendem que a etnobiologia deve dialogar com disciplinas com maior respaldo teórico, como ecologia e biologia evolutiva para melhorar a pesquisa sobre biodiversidade biodiversidade (Albuquerque & Ferreira Júnior 2017). Por sua vez, incorporando o conhecimento local em ecologia e evolução irá certamente refinar seu próprio desenvolvimento, que em última análise beneficia a conservação biológica (Saslis-Lagoudakis & Clarke 2013). Além disso, há uma necessidade urgente de formar jovens pesquisadores em filosofia e metodologia da ciência, bem como comunicação e produção científica (Albuquerque 2013). Como comentário final, acreditamos que a formação dos alunos em EEC precisa de uma reavaliação que necessariamente volta aos conceitos e métodos básicos. Assim, os pesquisadores podem combinar o método hipotético-dedutivo com pensamento estatístico usando um fluxograma de pesquisa para ir além da descrição básica.

Capítulo 3

Pré-requisitos

3.1 Introdução

O objetivo deste capítulo é informar como fazer a instalação dos Programas R e RStudio, além de descrever os pacotes e dados necessários para reproduzir os exemplos do livro.

3.2 Instalação do R

Abaixo descrevemos os sete passos necessários para a instalação do programa R no seu computador (Figura 3.1):

1. Para começarmos a trabalhar com o R é necessário baixá-lo na página do R Project. Então, acesse esse site http://www.r-project.org
2. Clique no link **download R**
3. Na página *CRAN Mirros (Comprehensive R Archive Network)*, escolha uma das páginas espelho do Brasil mais próxima de você para baixar o programa
4. Escolha agora o sistema operacional do seu computador (passos adicionais existem para diferentes distribuições Linux ou MacOS). Aqui faremos o exemplo com o Windows
5. Clique em **base** para finalmente chegar à página de download com a versão mais recente do R
6. Clique no arquivo **Download R (versão mais recente) for Windows** que será instalado no seu computador
7. Abra o arquivo que foi baixado no seu computador e siga os passos indicados para finalizar a instalação do programa R

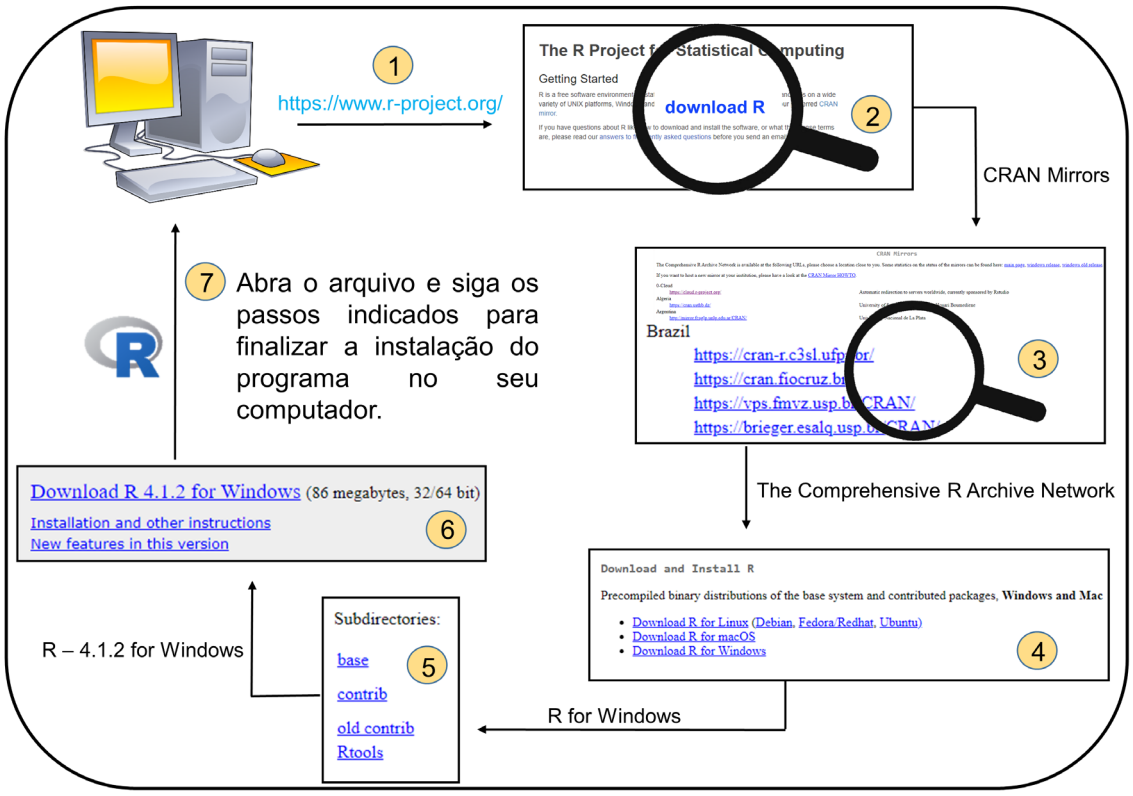


Figura 3.1: Esquema ilustrativo demonstrando os passos necessários para instalação do programa R no computador. Fonte das figuras: imagem computador e imagem da lupa.

Importante
Para o Sistema Operacional (SO) Windows, alguns pacotes são dependentes da instalação separada do Rtools40. Da mesma forma, GNU/Linux e MacOS também possuem dependências de outras bibliotecas para pacotes específicos, mas que não abordaremos aqui. Essas informações de dependência geralmente são retornadas como erros e você pode procurar ajuda em fóruns específicos.

3.3 Instalação do RStudio

O RStudio possui algumas características que o tornam popular: várias janelas de visualização, marcação e preenchimento automático do script, integração com controle de versão, dentre outras funcionalidades.

Abaixo descrevemos os cinco passos necessários para a instalação do RStudio no seu computador (Figura 3.2):

1. Para fazer o download do RStudio, acessamos o site https://www.rstudio.com/
2. Clique em **download**
3. Escolha a versão gratuita
4. Escolha o instalador com base no seu sistema operacional
5. Abra o arquivo que foi baixado no seu computador e siga os passos indicados para finalizar a instalação do programa RStudio

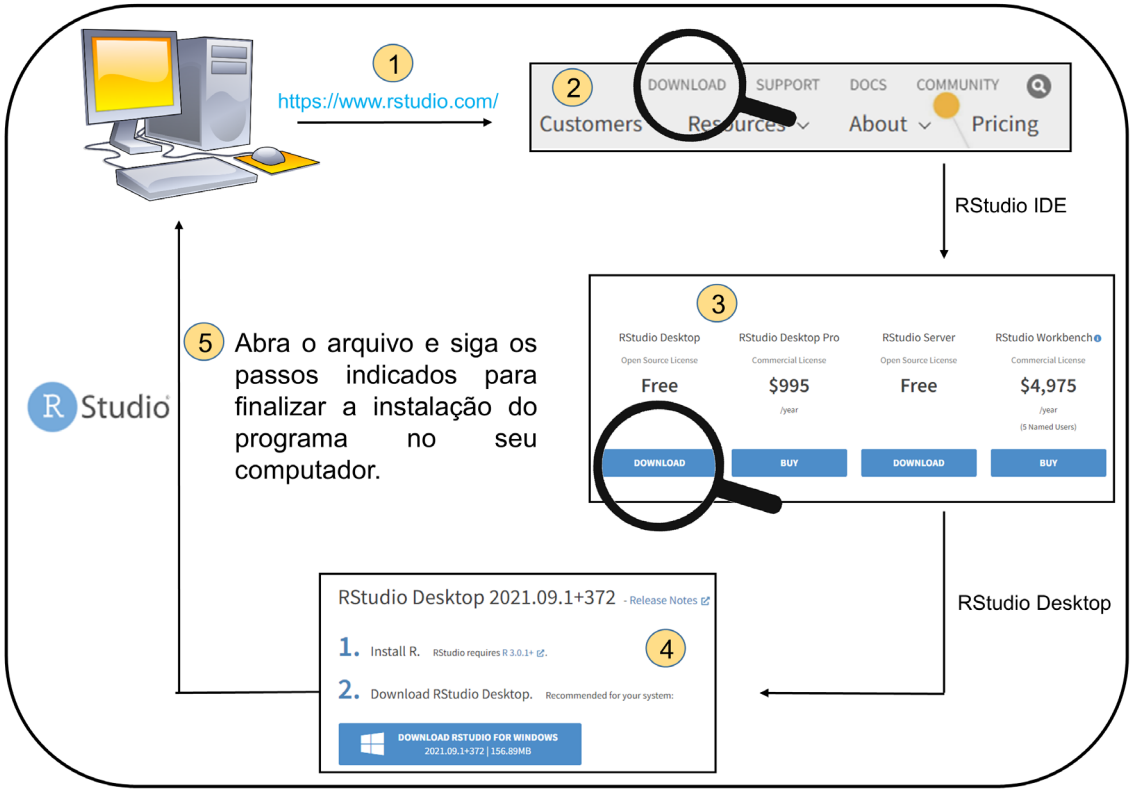


Figura 3.2: Esquema ilustrativo demonstrando os passos necessários para instalação do programa RStudio no computador. Fonte das figuras: imagem computador e imagem da lupa.

3.4 Versão do R

Todas os códigos, pacotes e análises disponibilizados no livro foram realizados no Programa R versão 4.1.2 (10-12-2021).

3.5 Pacotes

Descrevemos no Capítulo 4 o que são e como instalar os pacotes para realizar as análises estatísticas no R.

> **Importante**
> Criamos o pacote `ecodados` que contém todas as informações e dados utilizados neste livro. Assim, recomendamos que você instale e carregue este pacote no início de cada capítulo para ter acesso aos dados necessários para executar as funções no R.

Abaixo, listamos todos os pacotes utilizados no livro. Você pode instalar os pacotes agora ou esperar para instalá-los quando ler o Capítulo 4 e entender o que são as funções `install.packages()`, `library()` e `install_github()`. Para fazer a instalação, você vai precisar estar conectado à internet.

```
install.packages(c("ade4", "adespatial", "ape", "bbmle", "betapart",
"BiodiversityR", "car", "cati", "datasauRus", "devtools", "DHARMa", "dplyr",
"ecolottery", "emmeans", "factoextra", "FactoMineR", "fasterize", "FD",
"forcats", "geobr", "generalhoslem", "GGally", "ggExtra", "ggforce",
"ggplot2", "ggpubr", "ggrepel", "ggspatial", "glmmTMB", "grid", "gridExtra",
"here", "hillR", "iNEXT", "janitor", "kableExtra", "knitr", "labdsv",
"lattice", "leaflet", "lmtest", "lsmeans", "lubridate", "mapview", "MASS",
"MuMIn", "naniar", "nlme", "ordinal", "palmerpenguins", "performance",
"pez", "phyloregion", "phytools", "picante", "piecewiseSEM", "purrr",
"pvclust", "raster", "readr", "reshape2", "rgdal" , "Rmisc", "rnaturalearth",
"RVAideMemoire", "sciplot", "sf", "sidrar", "sjPlot", "spData", "spdep",
"stringr", "SYNCSA", "tibble", "tidyr", "tidyverse", "tmap", "tmaptools",
"TPD", "vcd", "vegan", "viridis", "visdat", "mvabund", "rdist", "udunits2"),
dependencies = TRUE)
```

Diferente dos pacotes anteriores que são baixados do CRAN, alguns pacotes são baixados do GitHub dos pesquisadores responsáveis pelos pacotes. GitHub é um repositório remoto de códigos que permite controle de versão, muito utilizado por desenvolvedores e programadores. Nestes casos, precisamos carregar o pacote `devtools` para acessar a função `install_github()`. Durante as instalações destes pacotes, algumas vezes o R irá pedir para você digitar um número indicando os pacotes que você deseja fazer update. Neste caso, digite 1 para indicar que ele deve atualizar os pacotes dependentes antes de instalar os pacotes requeridos.

```
library(devtools)
install_github("paternogbc/ecodados")
install_github("mwpennell/geiger-v2")
install_github("fawda123/ggord")
install_github("jinyizju/V.PhyloMaker")
install_github("ropensci/rnaturalearthhires")
```

3.6 Dados

A maioria dos exemplos do livro em dados reais extraídos de artigos científicos que já foram publicados ou dados que foram coletados por um dos autores deste livro. Em alguns casos, os dados foram simulados para facilitar a interpretação dos resultados de algumas análises estatísticas. Todos os dados, publicados ou simulados, estão disponíveis no pacote `ecodados`. Além disso, em cada capítulo fazemos uma breve descrição dos dados para facilitar a compreensão sobre o que é variável resposta ou preditora, como essas variáveis estão relacionadas com as perguntas e predições do exemplo.

Capítulo 4

Introdução ao R

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(ecodados)

## Dados necessários
intror_anfibios_locais <- ecodados::intror_anfibios_locais
```

4.1 Contextualização

O objetivo deste capítulo é apresentar os aspectos básicos da linguagem R para a realização dos principais passos para a manipulação, visualização e análise de dados. Abordaremos aqui as questões básicas sobre a linguagem R, como: i) R e RStudio, ii) funcionamento da linguagem, iii) estrutura e manipulação de objetos, iv) exercícios e v) principais livros e material para se aprofundar nos seus estudos.

Todo processo de aprendizagem torna-se mais efetivo quando a teoria é combinada com a prática. Assim, recomendamos fortemente que você, leitor(a) acompanhe os códigos e exercícios deste livro, ao mesmo tempo que os executa em seu computador e não só os leia passivamente. Além disso, se você tiver seus próprios dados é muito importante tentar executar e/ou replicar as análises e/ou gráficos. Por motivos de espaço, não abordaremos todas as questões relacionadas ao uso da linguagem R neste capítulo. Logo, aconselhamos que você consulte o material sugerido no final do capítulo para se aprofundar.

Este capítulo, na maioria das vezes, pode desestimular as pessoas que estão iniciando, uma vez que o mesmo não apresenta os códigos para realizar as análises estatísticas. Contudo, ele é essencial para o entendimento e interpretação do que está sendo informado nas linhas de código, além de facilitar a manipulação dos dados antes de realizar as análises estatísticas. Você perceberá que não usará este capítulo para fazer as análises, mas voltará aqui diversas vezes para relembrar qual é o código ou o que significa determinada expressão ou função usada nos próximos capítulos.

4.2 R e RStudio

Com o R é possível manipular, analisar e visualizar dados, além de escrever desde pequenas linhas de códigos até programas inteiros. O R é a versão em código aberto de uma linguagem de programação chamada de S, criada por John M. Chambers (Stanford University, CA, EUA) nos anos 1980 no Bell Labs, que contou com três versões: Old S (1976-1987), New S (1988-1997) e S4 (1998), utilizada na IDE S-PLUS (1988-2008). Essa linguagem tornou-se bastante popular e vários produtos comerciais que a usam ainda estão disponíveis, como o SAS.

No final dos anos 1990, Robert Gentleman e Ross Ihaka (ambos da Universidade de Auckland, Nova Zelândia), iniciaram o desenvolvimento da versão livre da linguagem S, a linguagem R, com o seguinte histórico: Desenvolvimento (1997-2000), Versão 1 (2000-2004), Versão 2 (2004-2013), Versão 3 (2013-2020) e Versão 4 (2020). Para mais detalhes do histórico de desenvolvimento das linguagens S e

R, consultar Wickham (2013). Atualmente a linguagem R é mantida por uma rede de colaboradores denominada *R Core Team*. A origem do nome R é desconhecida, mas reza a lenda que ao lançarem o nome da linguagem os autores se valeram da letra que vinha antes do S, uma vez que a linguagem R foi baseada nela e utilizaram a letra "R." Outra história conta que pelo fato do nome dos dois autores iniciarem por "R," batizaram a linguagem com essa letra, vai saber.

Um aspecto digno de nota é que a linguagem R é uma linguagem de programação interpretada, assim como o Python, mas contrária a outras linguagens como C e Java, que são compiladas. Isso a faz ser mais fácil de ser utilizada, pois processa linhas de código e as transforma em linguagem de máquina (código binário que o computador efetivamente lê), apesar desse fato diminuir a velocidade de processamento.

Para começarmos a trabalhar com o R é necessário baixá-lo na página do **R Project**. Os detalhes de instalação são apresentados no Capítulo 3. Reserve um tempo para explorar esta página do R-Project. Existem vários livros dedicados a diversos assuntos baseados no R. Além disso, estão disponíveis manuais em diversas línguas para serem baixados gratuitamente.

Como o R é um software livre, não existe a possibilidade de o usuário entrar em contato com um serviço de suporte de usuários, muito comuns em softwares pagos. Ao invés disso, existem várias listas de e-mails que fornecem suporte à comunidade de usuários. Nós, particularmente, recomendamos o ingresso nas seguintes listas: R-help, R-sig-ecolog, R-br e discourse.curso-r. Os dois últimos grupos reúnem pessoas usuárias brasileiras do programa R.

Apesar de podermos utilizar o R com o IDE (Ambiente de Desenvolvimento Integrado - *Integrated Development Environment*) RGui que vem com a instalação da linguagem R para usuários Windows (Figura 4.1) ou no próprio terminal para usuários Linux e MacOS, existem alguns IDEs específicos para facilitar nosso uso dessa linguagem.

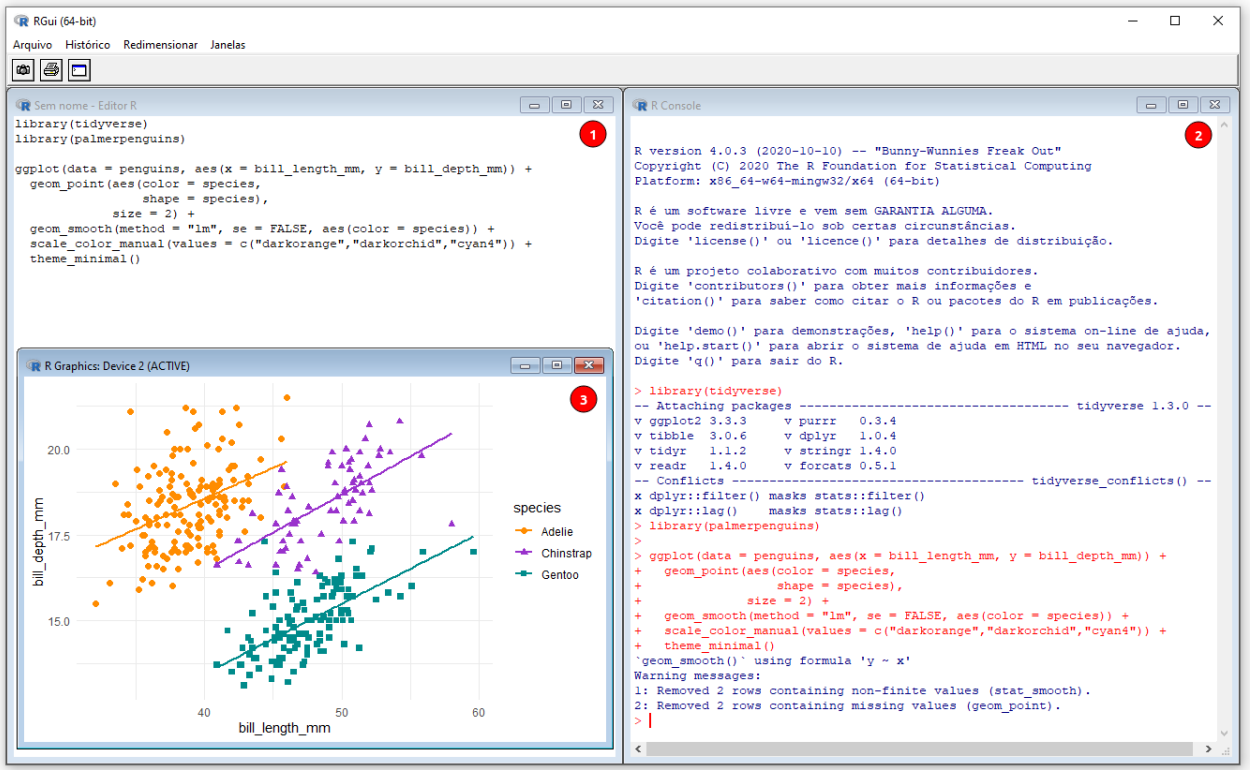


Figura 4.1: Interface do RGui. Os números indicam: (1) R Script, (2) R Console, e (3) R Graphics.

Dessa forma, nós que escrevemos este livro utilizamos o IDE RStudio e assumimos que você que está lendo fará o mesmo.

O RStudio permite diversas personalizações, grande parte delas contidas em `Tools > Global options`. Incentivamos as leitoras e leitores a "fuçar" com certa dose de cuidado, nas opções para personalização. Dentre essas mudanças, destacamos três:

1. `Tools > Global options > Appearance > Editor theme`: para escolher um tema para seu RStudio
2. `Tools > Global options > Code > [X] Soft-wrap R source files`: com essa opção habilitada, quando escrevemos comentários longos ou mudamos a largura da janela que estamos trabalhando, todo o texto e o código se ajustam a janela automaticamente
3. `Tools > Global options > Code > Display > [X Show Margis] e Margin column (80)`: com essa opção habilitada e para esse valor (80), uma linha vertical irá aparecer no script marcando 80 caracteres, um comprimento máximo recomendado para padronização dos scripts

O RStudio permite também trabalhar com projetos. **Projeto do RStudio** é uma forma de organizar os arquivos de scripts e dados dentro de um diretório, facilitando o compartilhamento de fluxo de análises de dados e aumentando assim a reprodutibilidade. Podemos criar um Projeto do RStudio indo em `File > New Project` ou no ícone de cubo azul escuro que possui um R dentro com um um círculo verde com um sinal de + na parte superior esquerda ou ainda no canto superior direito que possui cubo azul escrito `Project` que serve para gerenciar os projetos e depois em `New Project`. Depois de escolher uma dessas opções, uma janela se abrirá onde escolhemos uma das três opções: i) `New Directory` - para criar um diretório novo com diversas opções, ii) `Existing Directory` - para escolher um diretório já existente, e iii) `Version Control` - para criar um projeto que será versionado pelo `git` ou `Subversion`.

> **Importante**
> Para evitar possíveis erros é importante instalar primeiro o software da linguagem R e depois o IDE RStudio.

4.3 Funcionamento da linguagem R

Nesta seção, veremos os principais conceitos para entender como a linguagem R funciona ou como geralmente utilizamos o IDE RStudio no dia a dia, para executar nossas rotinas utilizando a linguagem R. Veremos então: i) console, ii) script, iii) operadores, iv) objetos, v) funções, vi) pacotes, vii) ajuda (*help*), viii) ambiente (*environment/workspace*), ix) citações e x) principais erros.

Antes de iniciarmos o uso do R pelo RStudio é fundamental entendermos alguns pontos sobre as janelas e o funcionamento delas no RStudio (Figura 4.2).

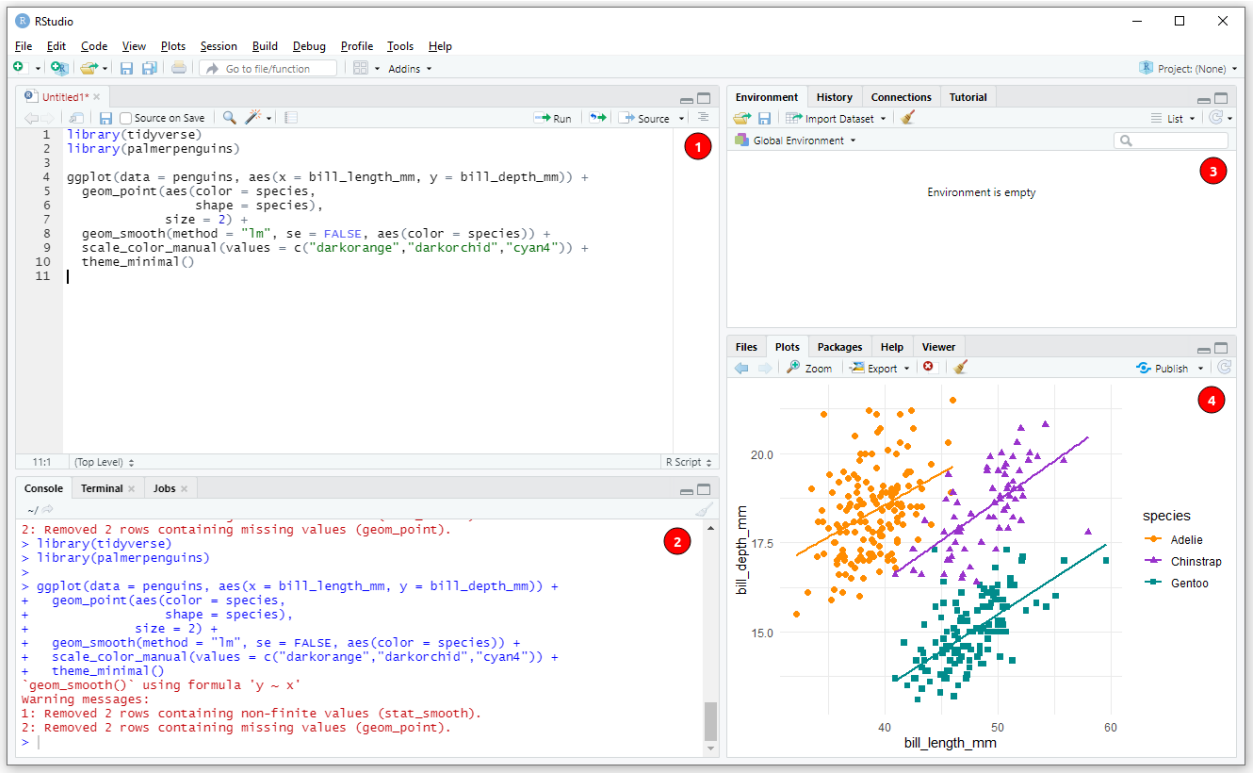


Figura 4.2: Interface do RStudio. Os números indicam: (1) janela com abas de Script, R Markdown, dentre outras; (2) janela com abas de Console, Terminal e Jobs; (3) janela com abas de Environment, History, Conections e Tutorial; e (4) janela com abas de Files, Plots, Packages, Help e Viewer.

Detalhando algumas dessas janelas e abas, temos:

- **Console**: painel onde os códigos são rodados e vemos as saídas
- **Editor/Script**: painel onde escrevemos nossos códigos em R, R Markdown ou outro formato
- **Environment**: painel com todos os objetos criados na sessão
- **History**: painel com o histórico dos códigos rodados
- **Files**: painel que mostra os arquivos no diretório de trabalho
- **Plots**: painel onde os gráficos são apresentados
- **Packages**: painel que lista os pacotes
- **Help**: painel onde a documentação das funções é exibida

No RStudio, alguns atalhos são fundamentais para aumentar nossa produtividade.

- **F1**: abre o painel de *Help* quando digitado em cima do nome de uma função
- **Ctrl + Enter**: roda a linha de código selecionada no script
- **Ctrl + Shift + N**: abre um novo script
- **Ctrl + S**: salva um script
- **Ctrl + Z**: desfaz uma operação
- **Ctrl + Shift + Z**: refaz uma operação
- **Alt + -**: insere um sinal de atribuição (<-)
- **Ctrl + Shift + M**: insere um operador pipe (%>%)
- **Ctrl + Shift + C**: comenta uma linha no script - insere um (#)

- **Ctrl + I**: indenta (recuo inicial das linhas) as linhas
- **Ctrl + Shift + A**: reformata o código
- **Ctrl + Shift + R**: insere uma sessão (# ————————-)
- **Ctrl + Shift + H**: abre uma janela para selecionar o diretório de trabalho
- **Ctrl + Shift + F10**: reinicia o console
- **Ctrl + L**: limpa os códigos do console
- **Alt + Shift + K**: abre uma janela com todos os atalhos disponíveis

4.3.1 Console

O console é onde a versão da linguagem R instalada é carregada para executar os códigos da linguagem R (Figura 4.2 - janela 2). Na janela do console aparecerá o símbolo `>`, seguido de uma barra vertical `|` que fica piscando (cursor), onde digitamos ou enviamos nossos códigos do script. Podemos fazer um pequeno exercício: vamos digitar `10 + 2`, seguido da tecla `Enter` para que essa operação seja executada.

```
10 + 2
#> [1] 12
```

O resultado retorna o valor `12`, precedido de um valor entre colchetes. Esses colchetes demonstram a posição do elemento numa sequência de valores. Se fizermos essa outra operação `1:42`, o R vai criar uma sequência unitária de valores de 1 a 42. A depender da largura da janela do console, vai aparecer um número diferente entre colchetes indicando sua posição na sequência: antes do número 1 vai aparecer o `[1]`, depois quando a sequência for quebrada, vai aparecer o número correspondente da posição do elemento, por exemplo, `[41]`.

```
1:42
#> [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
#> [41] 41 42
```

Podemos ver o histórico dos códigos executados no console na aba **History** (Figura 4.2 - janela 3).

4.3.2 Scripts

Scripts são arquivos de texto simples, criados com a extensão (terminação) `.R` (Figura 4.2 janela 1). Para criar um script, basta ir em `File > New File > R Script`, ou clicar no ícone com uma folha branca e um círculo verde com um sinal de `+`, logo abaixo de `File`, ou ainda usando o atalho `Ctrl + Shift + N`.

Uma vez escrito os códigos no script podemos rodar esses códigos de duas formas: i) todo o script de uma vez, clicando em **Source** (que fica no canto superior direito da aba script) ou usando o atalho `Ctrl + Shift + Enter`; ou ii) apenas a linha onde o cursor estiver posicionado, independentemente de sua posição naquela linha, clicando em **Run** ou usando o atalho `Ctrl + Enter`.

Devemos sempre salvar nossos scripts, tomando por via de regra: primeiro criar o arquivo e depois ir salvando nesse mesmo arquivo a cada passo de desenvolvimento das análises (não é raro o RStudio fechar sozinho e você perder algum tempo de trabalho). Há diversos motivos para se criar um script:

continuar o desenvolvimento desse script em outro momento ou em outro computador, preservar trabalhos passados, ou ainda compartilhar seus códigos com outras pessoas. Para criar ou salvar um script basta ir em `File > Save`, escolher um diretório e nome para o script e salvá-lo. Podemos ainda utilizar o atalho `Ctrl + S`.

Em relação aos scripts, há ainda os comentários, representados pelos símbolos `#` (hash), `#'` (hash-linha) e `#>` (hash-maior). A diferença entre eles é que para o segundo e terceiro, quando pressionamos a tecla `Enter` o comentário `#'` e `#>` são inseridos automaticamente na linha seguinte. Linhas de códigos do script contendo comentários em seu início não são lidos pelo console do R. Se o comentário estiver no final da linha, essa linha de código ainda será lida. Os comentários são utilizados geralmente para: i) descrever informações sobre dados ou funções e/ou ii) suprimir linhas de código.

É interessante ter no início de cada script um cabeçalho identificando o objetivo ou análise, autor e data para facilitar o compartilhamento e reprodutibilidade. Os comentários podem ser inseridos ou retirados das linhas com o atalho `Ctrl + Shift + C`.

```
#' ---
#' Título: Capítulo 04 - Introdução ao R
#' Autor: Maurício Vancine
#' Data: 11-11-2021
#' ---
```

Além disso, podemos usar comentários para adicionar informações sobre os códigos.

```
## Comentários
# O R não lê a linha do código depois do # (hash).
42 # Essas palavras não são executadas, apenas o 42, a resposta para questão fundamental da vida, o universo e tudo mais.
#> [1] 42
```

Por fim, outro ponto fundamental é ter boas práticas de estilo de código. Quanto mais organizado e padronizado estiver seus scripts, mais fácil de entendê-los e de procurar possíveis erros. Existem dois guias de boas práticas para adequar seus scripts: Hadley Wickham e Google.

Ainda em relação aos scripts, temos os *Code Snippets* (Fragmentos de código), que são macros de texto usadas para inserir rapidamente fragmentos comuns de código. Por exemplo, o *snippet* `fun` insere uma definição de função R. Para mais detalhes, ler o artigo do RStudio Code Snippets.

```
# fun {snippet}
fun
name <- function(variables) {

}
```

Uma aplicação bem interessante dos *Code Snippets* no script é o `ts`. Basta digitar esse código e em seguida pressionar a tecla `Tab` para inserir rapidamente a data e horário atuais no script em forma de comentário.

```
# ts {snippet}
# Thu Nov 11 18:19:26 2021 ----------------------------
```

4.3.3 Operadores

No R, podemos agrupar os operadores em cinco tipos: aritméticos, relacionais, lógicos, atribuição e diversos. Grande parte deles são descritos na Tabela 4.1.

Tabela 4.1: Principais operadores no R.

Operador	Tipo	Descrição
+	Aritmético	Adição
-	Aritmético	Subtração
*	Aritmético	Multiplicação
/	Aritmético	Divisão
%%	Aritmético	Resto da divisão
%/%	Aritmético	Divisão inteira
^ ou **	Aritmético	Expoente
>	Relacional	Maior
<	Relacional	Menor
>=	Relacional	Maior ou igual
<=	Relacional	Menor ou igual
==	Relacional	Igualdade
!=	Relacional	Diferença
!	Lógico	Lógico NÃO
&	Lógico	Lógico elementar E
|	Lógico	Lógico elementar OU
&&	Lógico	Lógico E
||	Lógico	Lógico OU
<- ou =	Atribuição	Atribuição à esquerda
<<-	Atribuição	Super atribuição à esquerda
->	Atribuição	Atribuição à direita
->>	Atribuição	Super atribuição à direita
:	Diversos	Sequência unitária
%in%	Diversos	Elementos que pertencem a um vetor
%*%	Diversos	Multiplicar matriz com sua transposta
%>%	Diversos	Pipe (pacote magrittr)
|>	Diversos	Pipe (R base nativo)
%–%	Diversos	Intervalo de datas (pacote lubridate)

Como exemplo, podemos fazer operações simples usando os operadores aritméticos.

```
## Operações aritméticas
10 + 2 # adição
#> [1] 12
```

```
10 * 2 # multiplicação
#> [1] 20
```

Precisamos ficar atentos à prioridade dos operadores aritméticos:

PRIORITÁRIO `()` > `^` > `* ou /` > `+ ou -` **NÃO PRIORITÁRIO**

Veja no exemplo abaixo como o uso dos parênteses muda o resultado.

```
## Sem especificar a ordem
# Segue a ordem dos operadores.
1 * 2 + 2 / 2 ^ 2
#> [1] 2.5

## Especificando a ordem
# Segue a ordem dos parenteses.
((1 * 2) + (2 / 2)) ^ 2
#> [1] 9
```

4.3.4 Objetos

Objetos são palavras às quais são atribuídos dados. A atribuição possibilita a manipulação de dados ou armazenamento dos resultados de análises. Utilizaremos os símbolos `<` (menor), seguido de `-` (menos), sem espaço, dessa forma `<-`. Também podemos utilizar o símbolo de igual (`=`), mas não recomendamos, por não fazer parte das boas práticas de escrita de códigos em R. Podemos inserir essa combinação de símbolos com o atalho `Alt + -`. Para demonstrar, vamos atribuir o valor `10` à palavra `obj_10`, e chamar esse objeto novamente para verificar seu conteúdo.

```
## Atribuição - símbolo (<-)
obj_10 <- 10
obj_10
#> [1] 10
```

> **Importante**
> Recomendamos sempre verificar o conteúdo dos objetos chamando-os novamente para confirmar se a atribuição foi realizada corretamente e se o conteúdo corresponde à operação realizada.

Todos os objetos criados numa sessão do R ficam listados na aba **Environment** (Figura 4.2 - janela 3). Além disso, o RStudio possui a função *autocomplete*, ou seja, podemos digitar as primeiras letras de um objeto (ou função) e em seguida apertar `Tab` para que o RStudio liste tudo que começar com essas letras.

Dois pontos importantes sobre atribuições: primeiro, o R sobrescreve os valores dos objetos com o mesmo nome, deixando o objeto com o valor da última atribuição.

```
## Sobrescreve o valor dos objetos
obj <- 100
obj
#> [1] 100

## O objeto 'obj' agora vale 2
obj <- 2
obj
#> [1] 2
```

Segundo, o R tem limitações ao nomear objetos:

- nome de objetos só podem começar por letras (`a-z` ou `A-Z`) ou pontos (`.`)
- nome de objetos só podem conter letras (`a-z` ou `A-Z`), números (`0-9`), underscores (`_`) ou pontos (`.`)
- R é *case-sensitive*, i.e., ele reconhece letras maiúsculas como diferentes de letras minúsculas. Assim, um objeto chamado "resposta" é diferente do objeto "RESPOSTA"
- devemos evitar acentos ou cedilha (`ç`) para facilitar a memorização dos objetos e também para evitar erros de codificação (*encoding*) de caracteres
- nomes de objetos não podem ser iguais a nomes especiais, reservados para programação (`break`, `else`, `FALSE`, `for`, `function`, `if`, `Inf`, `NA`, `NaN`, `next`, `repeat`, `return`, `TRUE`, `while`)

Podemos ainda utilizar objetos para fazer operações e criar objetos. Isso pode parecer um pouco confuso para os iniciantes, mas é fundamental aprender essa lógica para passar para os próximos passos.

```
## Definir dois objetos
va1 <- 10
va2 <- 2

## Operações com objetos e atribuicão
adi <- va1 + va2
adi
#> [1] 12
```

4.3.5 Funções

Funções são códigos preparados para realizar uma tarefa específica de modo simples. Outra forma de entender uma função é: códigos que realizam operações em argumentos. Devemos retomar ao conceito do ensino médio de funções: os dados de entrada são argumentos e a função realizará alguma operação para modificar esses dados de entrada. A estrutura de uma função é muito similar à sintaxe usada em planilhas eletrônicas, sendo composta por:

nome_da_função(argumento1, argumento2, ...)

1. **Nome da função**: remete ao que ela faz
2. **Parênteses**: limitam a função
3. **Argumentos**: valores, parâmetros ou expressões onde a função atuará
4. **Vírgulas**: separam os argumentos

Os argumentos de uma função podem ser de dois tipos:

1. **Valores ou objetos**: a função alterará os valores em si ou os valores atribuídos aos objetos
2. **Parâmetros**: valores fixos que informam um método ou a realização de uma operação. Informa-se o nome desse argumento, seguido de "=" e um número, texto ou TRUE ou FALSE

Alguns exemplos de argumentos como valores ou objetos.

```
## Funções - argumentos como valores
sum(10, 2)
#> [1] 12

## Funções - argumentos como objetos
sum(va1, va2)
#> [1] 12
```

Vamos ver agora alguns exemplos de argumentos usados como parâmetros. Note que apesar do valor do argumento ser o mesmo (10), seu efeito no resultado da função `rep()` muda drasticamente. Aqui também é importante destacar um ponto: i) podemos informar os argumentos sequencialmente, sem explicitar seus nomes, ou ii) independente da ordem, mas explicitando seus nomes. Entretanto, como no exemplo abaixo, devemos informar o nome do argumento (i.e., parâmetro), para que seu efeito seja o que desejamos.

```
## Funções - argumentos como parâmetros
## Repetição - repete todos os elementos
rep(x = 1:5, times = 10)
#>  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5

## Repetição - repete cada um dos elementos
rep(x = 1:5, each = 10)
#>  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4
4 4 4 4 4 5 5 5 5 5 5 5 5 5 5
```

Um ponto fundamental e que deve ser entendido é o fluxo de atribuições do resultado da operação de funções a novos objetos. No desenvolvimento de qualquer script na linguagem R, grande parte da estrutura do mesmo será dessa forma: atribuição de dados a objetos > operações com funções > atribuição dos resultados a novos objetos > operações com funções desses novos objetos > atribuição dos resultados a novos objetos. Ao entender esse funcionamento, começamos a entender como devemos pensar na organização do nosso script para montar as análises que precisamos.

```
## Atribuicão dos resultados
## Repetição
rep_times <- rep(1:5, times = 10)
rep_times
```

```
#>  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5

## Somar e atribuir
rep_times_soma <- sum(rep_times)
rep_times_soma
#> [1] 150

## Raiz e atribuir
rep_times_soma_raiz <- sqrt(rep_times_soma)
rep_times_soma_raiz
#> [1] 12.24745
```

Por fim, é fundamental também entender a origem das funções que usamos no R. Todas as funções são advindas de pacotes. Esses pacotes possuem duas origens.

1. pacotes já instalados por padrão e que são carregados quando abrimos o R (*R Base*)
2. pacotes que instalamos e carregamos com funções

4.3.6 Pacotes

Pacotes são conjuntos extras de funções para executar tarefas específicas, além dos pacotes instalados no *R Base*. Existe literalmente milhares de pacotes (~19,000 enquanto estávamos escrevendo esse livro) para as mais diversas tarefas: estatística, ecologia, geografia, sensoriamento remoto, econometria, ciências sociais, gráficos, *machine learning*, etc. Podemos verificar este vasto conjunto de pacotes pelo link que lista por nome os pacotes oficiais, ou seja, que passaram pelo crivo do **CRAN**. Existem ainda muito mais pacotes em desenvolvimento, geralmente disponibilizados em repositórios do **GitHub** ou **GitLab**.

Podemos listar esses pacotes disponíveis no **CRAN** com esse código.

```
## Número atual de pacotes no CRAN
nrow(available.packages())
#> [1] 18977
```

Primeiramente, com uma sessão do R sem carregar nenhum pacote extra, podemos verificar pacotes carregados pelo *R Base* utilizando a função `search()`.

```
## Verificar pacotes carregados
search()
```

Podemos ainda verificar todos pacotes instalados em nosso computador com a função `library()`.

```
## Verificar pacotes instalados
library()
```

No R, quando tratamos de pacotes, devemos destacar a diferença de dois conceitos: instalar um pacote e carregar um pacote. A instalação de pacotes possui algumas características:

- Instala-se um pacote apenas uma vez
- Precisamos estar conectados à internet

- O nome do pacote precisa estar entre aspas na função de instalação
- Função (CRAN): `install.packages()`

Vamos instalar o pacote `vegan` diretamente do CRAN, que possui funções para realizar uma série de análise em ecologia (veja mais no Capítulo 10). Para isso, podemos ir em `Tools > Install Packages...`, ou ir na aba **Packages** (Figura 4.2 - janela 4), procurar o pacote e simplesmente clicar em "*Install*." Podemos ainda utilizar a função `install.packages()`.

```
## Instalar pacotes
install.packages("vegan")
```

Podemos conferir em que diretórios um pacote será instalado com a função `.libPaths()`.

```
## Diretórios de intalação dos pacotes
.libPaths()
#> [1] "/home/mude/R/x86_64-pc-linux-gnu-library/4.1" "/usr/local/lib/R/site-
library"
#> [3] "/usr/lib/R/site-library"                      "/usr/lib/R/library"
```

> **Importante**
> Uma vez instalado um pacote, não há necessidade de instalá-lo novamente. Entretanto, todas às vezes que iniciarmos uma sessão no R, precisamos carregar os pacotes com as funções que precisamos utilizar.

O carregamento de pacotes possui algumas características:

- Carrega-se o pacote toda vez que se abre uma nova sessão do R
- Não precisamos estar conectados à internet
- O nome do pacote não precisa estar entre aspas na função de carregamento
- Funções: `library()` ou `require()`

Vamos carregar o pacote `vegan` que instalamos anteriormente. Podemos ir na aba **Packages** (Figura 4.2 janela 4) e assinalar o pacote que queremos carregar ou utilizar a função `library()`.

```
## Carregar pacotes
library(vegan)
```

Como dissemos, alguns pacotes em desenvolvimento encontram-se disponíveis em repositórios do GitHub, GitLab e Bioconductor. Para instalar pacotes do GitHub, por exemplo, precisamos instalar e carregar o pacote `devtools`.

```
## Instalar pacote devtools
install.packages("devtools")

## Carregar pacote devtools
library(devtools)
```

Uma vez instalado e carregado esse pacote, podemos instalar o pacote do GitHub, utilizando a função `devtools::install_github()`. Precisamos atentar para usar essa forma "nome_usuario/

nome_repositorio," retirados do link do repositório de interesse. Como exemplo, podemos instalar o pacote `ecodados` do repositório do GitHub paternogbc/ecodados e depois utilizar a função `library()` para carregá-lo.

```
## Instalar pacote do github
devtools::install_github("paternogbc/ecodados")

## Carregar pacote do github
library("ecodados")
```

Pode ser que em algumas circunstâncias precisaremos instalar pacotes com versões específicas para algumas análises. A forma mais simples de fazer isso é instalar um pacote a partir de um arquivo compactado `.tar.gz`. Para isso podemos ir à base do CRAN e realizar o download: https://cran.r-project.org/src/contrib/Archive/. Para exemplificar, vamos instalar o pacote `vegan 2.4.0`.

```
## Download do arquivo .tar.gz
download.file(url = "https://cran.r-project.org/src/contrib/Archive/vegan/
vegan_2.4-0.tar.gz",
              destfile = "vegan_2.4-0.tar.gz", mode = "auto")

## Instalar o pacote vegan 2.4.0
install.packages("vegan_2.4-0.tar.gz", repos = NULL, type = "source")
```

Podemos ver a descrição de um pacote com a função `packageDescription()`.

```
## Descrição de um pacote
packageDescription("vegan")
```

A maioria dos pacotes possui conjuntos de dados que podem ser acessados pela função `data()`. Esses conjuntos de dados podem ser usados para testar as funções do pacote. Se estiver com dúvida na maneira como você deve preparar a planilha para realizar uma análise específica, entre na Ajuda (*Help*) da função e veja os conjuntos de dados que estão no exemplo desta função. Como exemplo, vamos carregar os dados `dune` do pacote `vegan`, que são dados de observações de 30 espécies vegetais em 20 locais.

```
## Carregar dados de um pacote
library(vegan)
data(dune)
dune[1:6, 1:6]
#>   Achimill Agrostol Airaprae Alopgeni Anthodor Bellpere
#> 1        1        0        0        0        0        0
#> 2        3        0        0        2        0        3
#> 3        0        4        0        7        0        2
#> 4        0        8        0        2        0        2
#> 5        2        0        0        0        4        2
#> 6        2        0        0        0        3        0
```

Se por algum motivo precisarmos desinstalar um pacote, podemos utilizar a função `remove.packages()`. Já para descarregar um pacote de uma sessão do R, podemos usar a função `detach()`.

```
## Desinstalar um pacote
remove.packages("vegan")

## Descarregar um pacote
detach("package:vegan", unload = TRUE)
```

E um último ponto fundamental sobre pacotes, diz respeito à atualização dos mesmos. Os pacotes são atualizados com frequência, e infelizmente (ou felizmente, pois as atualizações podem oferecer algumas quebras entre pacotes), não se atualizam sozinhos. Muitas vezes, a instalação de um pacote pode depender da versão dos pacotes dependentes, e geralmente uma janela com diversas opções numéricas se abre perguntando se você quer que todos os pacotes dependentes sejam atualizados. Podemos ir na aba **Packages** (Figura 4.2 - janela 4) e clicar em "*Update*" ou usar a função `update.packages(checkBuilt = TRUE, ask = FALSE)` para atualizá-los, entretanto, essa é uma função que costuma demorar muito para terminar de ser executada.

```
## Atualização dos pacotes
update.packages(checkBuilt = TRUE, ask = FALSE)
```

Para fazer a atualização dos pacotes instalados pelo GitHub, recomendamos o uso do pacote `dtupdate`.

```
## Atualização dos pacotes instalados pelo GitHub
dtupdate::github_update(auto.install = TRUE, ask = FALSE)
```

Destacamos e incentivamos ainda uma prática que achamos interessante para aumentar a reprodutibilidade de nossos códigos e scripts: a de chamar as funções de pacotes carregados dessa forma `pacote::função()`. Com o uso dessa prática, deixamos claro o pacote em que a função está implementada. Esta prática é importante por que com frequência pacotes diferentes criam funções com mesmo nome, mas com características internas (argumentos) diferentes. Assim, não expressar o pacote de interesse pode gerar erros na execução de suas análises. Destacamos aqui o exemplo de como instalar pacotes do GitHub do pacote `devtools`.

```
## Pacote seguido da função implementada daquele pacote
devtools::install_github()
```

4.3.7 Ajuda (*Help*)

Um importante passo para melhorar a usabilidade e ter mais familiaridade com a linguagem R é aprender a usar a ajuda (*help*) de cada função. Para tanto, podemos utilizar a função `help()` ou o operador `?`, depois de ter carregado o pacote, para abrir uma nova aba (Figura 4.2 janela 4) que possui diversas informações sobre a função de interesse. O arquivo de ajuda do R possui geralmente nove ou dez tópicos, que nos auxiliam muito no entendimento dos dados de entrada, argumentos e que operações estão sendo realizadas. Abaixo descrevemos esses tópicos:

- **Description**: resumo da função
- **Usage**: como utilizar a função e quais os seus argumentos
- **Arguments**: detalha os argumentos e como os mesmos devem ser especificados
- **Details**: detalhes importantes para se usar a função
- **Value**: mostra como interpretar a saída (*output*) da função (os resultados)

- **Note**: notas gerais sobre a função
- **Authors**: autores da função
- **References**: referências bibliográficas para os métodos usados para construção da função
- **See also**: funções relacionadas
- **Examples**: exemplos do uso da função. Às vezes pode ser útil copiar esse trecho e colar no R para ver como funciona e como usar a função.

Vamos realizar um exemplo, buscando o `help` da função `aov()`, que realiza uma análise de variância (veja detalhes no Capítulo 7).

```
## Ajuda
help(aov)
?aov
```

Além das funções, podemos buscar detalhes de um pacote específico, para uma página simples do `help` utilizando a função `help()` ou o operador `?`. Entretanto, para uma opção que ofereça uma descrição detalhada e um índice de todas as funções do pacote, podemos utilizar a função `library()`, mas agora utilizando o argumento `help`, indicando o pacote de interesse entre aspas.

```
## Ajuda do pacote
help(vegan)
?vegan

## Help detalhado
library(help = "vegan")
```

Podemos ainda procurar o nome de uma função para realizar uma análise específica utilizando a função `help.search()` com o termo que queremos em inglês e entre aspas.

```
## Procurar por funções que realizam modelos lineares
help.search("linear models")
```

Outra ferramenta de busca é a página rseek, na qual é possível buscar por um termo não só nos pacotes do R, mas também em listas de e-mails, manuais, páginas na internet e livros sobre o programa.

4.3.8 Ambiente (*Environment*)

O ambiente *(environment),* como vimos, é onde os objetos criados são armazenados. É fundamental entender que um objeto é uma alocação de um pequeno espaço na memória RAM do nosso computador, onde o R armazenará um valor ou o resultado de uma função, utilizando o nome dos objetos que definimos na atribuição. Sendo assim, se fizermos a atribuição de um objeto maior que o tamanho da memória RAM do nosso computador, esse objeto não será alocado, e a atribuição não funcionará, retornando um erro. Existem opções para contornar esse tipo de limitação, mas não a abordaremos aqui. Entretanto, podemos utilizar a função `object.size()` para saber quanto espaço nosso objeto criado está alocando de memória RAM.

```
## Tamanho de um objeto
object.size(adi)
#> 56 bytes
```

Podemos listar todos os objetos criados com a função `ls()` ou `objects()`.

```
## Listar todos os objetos
ls()
```

Podemos ainda remover todos os objetos criados com a função `rm()` ou `remove()`. Ou ainda fazer uma função composta para remover todos os objetos do *Environment*.

```
## Remover um objeto
rm(adi)

## Remover todos os objetos criados
rm(list = ls())
```

Quando usamos a função `ls()` agora, nenhum objeto é listado.

```
## Listar todos os objetos
ls()
#> character(0)
```

Toda a vez que fechamos o R os objetos criados são apagados do **Environment**. Dessa forma, em algumas ocasiões, por exemplo, análises estatísticas que demoram um grande tempo para serem realizadas, pode ser interessante exportar alguns ou todos os objetos criados.

Para salvar todos os objetos, ou seja, todo o *Workspace*, podemos ir em `Session -> Save Workspace As...` e escolher o nome do arquivo do *Workspace*, por exemplo, "meu_workspace.RData". Podemos ainda utilizar funções para essas tarefas. A função `save.image()` salva todo *Workspace* com a extensão `.RData`.

```
## Salvar todo o workspace
save.image(file = "meu_workspace.RData")
```

Depois disso, podemos fechar o RStudio tranquilamente e quando formos trabalhar novamente, podemos carregar os objetos criados indo em `Session -> Load Workspace...` ou utilizando a função `load()`.

```
## Carregar todo o workspace
load("meu_workspace.RData")
```

Entretanto, em algumas ocasiões, não precisamos salvar todos os objetos. Dessa forma, podemos salvar apenas alguns objetos específicos usando a função `save()`, também com a extensão `.RData`.

```
## Salvar apenas um objeto
save(obj1, file = "meu_obj.RData")

## Salvar apenas um objeto
save(obj1, obj2, file = "meus_objs.RData")

## Carregar os objetos
load("meus_objs.RData")
```

Ou ainda, podemos salvar apenas um objeto com a extensão `.rds`. Para isso, usamos as funções `saveRDS()` e `readRDS()`, para exportar e importar esses dados, respectivamente. É importante

ressaltar que nesse formato `.rds`, apenas um objeto é salvo por arquivo criado e que para que o objeto seja criado no *Workspace* do R, ele precisa ser lido e atribuído à um objeto.

```
## Salvar um objeto para um arquivo
saveRDS(obj, file = "meu_obj.rds")

## Carregar esse objeto
obj <- readRDS(file = "meu_obj.rds")
```

4.3.9 Citações

Ao utilizar o R para realizar alguma análise em nossos estudos, é fundamental a citação do mesmo. Para saber como citar o R em artigos, existe uma função denominada `citation()`, que provê um formato genérico de citação e um BibTeX para arquivos LaTeX e R Markdown.

```
## Citação do R
citation()
#>
#> To cite R in publications use:
#>
#>   R Core Team (2021). R: A language and environment for statistical
computing. R Foundation for Statistical
#>   Computing, Vienna, Austria. URL https://www.R-project.org/.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {R: A Language and Environment for Statistical Computing},
#>     author = {{R Core Team}},
#>     organization = {R Foundation for Statistical Computing},
#>     address = {Vienna, Austria},
#>     year = {2021},
#>     url = {https://www.R-project.org/},
#>   }
#>
#> We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See
#> also 'citation("pkgname")' for citing R packages.
```

No resultado dessa função, há uma mensagem muito interessante: "See also `citation("pkgname")` for citing R packages." Dessa forma, aconselhamos, sempre que possível, claro, citar também os pacotes utilizados nas análises para dar os devidos créditos aos desenvolvedores e desenvolvedoras das funções implementadas nos pacotes. Como exemplo, vamos ver como fica a citação do pacote `vegan`.

```
## Citação do pacote vegan
citation("vegan")
```

Podemos ainda utilizar a função `write_bib()` do pacote `knitr` para exportar a citação do pacote no formato `.bib`.

```
## Exportar uma citação em formato .bib
knitr::write_bib("vegan", file = "vegan_ex.bib")
```

4.3.10 Principais erros de iniciantes

Errar quando se está começando a usar o R é muito comum e faz parte do aprendizado. Entretanto, os erros nunca devem ser encarados como uma forma de desestímulo, mas sim como um desafio para continuar tentando. Todos nós, autores deste livro inclusive, e provavelmente usuários mais ou menos experientes, já passaram por um momento em que se quer desistir de tudo. Jovem aprendiz de R, a única diferença entre você que está iniciando agora e nós que usamos o R há mais tempo são as horas a mais de uso (e ódio). O que temos a mais é experiência para olhar o erro, lê-lo e conseguir interpretar o que está errado e saber buscar ajuda.

Dessa forma, o ponto mais importante de quem está iniciando é ter paciência, calma, bom humor, ler e entender as mensagens de erros. Recomendamos uma prática que pode ajudar: caso não esteja conseguindo resolver alguma parte do seu código, deixe ele de lado um tempo, descanse, faça uma caminhada, tome um banho, converse com seus animais de estimação ou plantas, tenha um pato de borracha ou outro objeto inanimado (um dos autores tem um sapinho de madeira), explique esse código para esse pato (processo conhecido como Debug com Pato de Borracha), logo a solução deve aparecer.

Listaremos aqui o que consideramos os principais erros dos iniciantes no R.

1. Esquecer de completar uma função ou bloco de códigos

Esquecer de completar uma função ou bloco de códigos é algo bem comum. Geralmente esquecemos de fechar aspas `""` ou parênteses `()`, mas geralmente o R nos informa isso, indicando um símbolo de `+` no console. Se você cometeu esse erro, lembre-se de apertar a tecla `esc` do seu computador clicando antes com o cursor do mouse no console do R.

```
sum(1, 2
  +
#> Error: <text>:3:0: unexpected end of input
#> 1: sum(1, 2
#> 2:   +
#>    ^
```

2. Esquecer de vírgulas dentro de funções

Outro erro bastante comum é esquecer de acrescentar a vírgula `,` para separar argumentos dentro de uma função, principalmente se estamos compondo várias funções acopladas, i.e., uma função dentro da outra.

```
sum(1 2)
#> Error: <text>:1:7: unexpected numeric constant
#> 1: sum(1 2
#>           ^
```

3. Chamar um objeto pelo nome errado

Pode parecer simples, mas esse é de longe o erro mais comum que pessoas iniciantes comentem. Quando temos um script longo, é de se esperar que tenhamos atribuído diversos objetos e em algum momento atribuímos um nome do qual não lembramos. Dessa forma, quando chamamos o objeto ele não existe e o console informa um erro. Entretanto, esse tipo de erro pode ser facilmente identificado, como o exemplo abaixo.

```
obj <- 10
OBJ
#> Error in eval(expr, envir, enclos): object 'OBJ' not found
```

4. Esquecer de carregar um pacote

Esse também é um erro recorrente, mesmo para usuários mais experientes. Em scripts de análises complexas, que requerem vários pacotes, geralmente esquecemos de um ou outro pacote. A melhor forma de evitar esse tipo de erro é listar os pacotes que vamos precisar usar logo no início do script.

```
## Carregar dados
data(dune)

## Função do pacote vegan
decostand(dune[1:6, 1:6], "hell")
#> Error in decostand(dune[1:6, 1:6], "hell"): could not find function
"decostand"
```

Geralmente a mensagem de erro será de que a função não foi encontrada ou algo nesse sentido. Carregando o pacote, esse erro é contornado.

```
## Carregar o pacote
library(vegan)

## Carregar dados
data(dune)

## Função do pacote vegan
decostand(dune[1:6, 1:6], "hell")
#>    Achimill  Agrostol Airaprae  Alopgeni  Anthodor  Bellpere
#> 1 1.0000000 0.0000000        0 0.0000000 0.0000000 0.0000000
#> 2 0.6123724 0.0000000        0 0.5000000 0.0000000 0.6123724
#> 3 0.0000000 0.5547002        0 0.7337994 0.0000000 0.3922323
#> 4 0.0000000 0.8164966        0 0.4082483 0.0000000 0.4082483
#> 5 0.5000000 0.0000000        0 0.0000000 0.7071068 0.5000000
#> 6 0.6324555 0.0000000        0 0.0000000 0.7745967 0.0000000
```

5. Usar o nome da função de forma errônea

Esse erro não é tão comum, mas pode ser incômodo às vezes. Algumas funções possuem nomes no padrão "Camel Case," i.e., com letras maiúsculas no meio do nome da função. Isso às vezes pode confundir, ou ainda, as funções podem ou não ser separadas com `.`, como `row.names()` e `rownames()`. No Capítulo 5 sobre *tidyverse*, veremos que houve uma tentativa de padronização nos

nomes das funções para "Snake Case," i.e, todas as funções possuem letras minúsculas, com palavras separadas por *underscore* `_`.

```
## Soma das colunas
colsums(dune)
#> Error in colsums(dune): could not find function "colsums"
## Soma das colunas
colSums(dune)
#> Achimill Agrostol Airaprae Alopgeni Anthodor Bellpere Bromhord Chenalbu
Cirsarve Comapalu Eleopalu Elymrepe Empenigr
#>       16       48        5       36       21       13       15        1
2        4       25       26        2
#> Hyporadi Juncarti Juncbufo Lolipere Planlanc  Poaprat  Poatriv Ranuflam
Rumeacet Sagiproc Salirepe Scorautu Trifprat
#>        9       18       13       58       26       48       63       14
18       20       11       54        9
#> Trifrepe Vicilath Bracruta Callcusp
#>       47        4       49       10
```

6. Atentar para o diretório correto

Muitas vezes o erro é simplesmente porque o usuário(a) não definiu o diretório correto onde está o arquivo a ser importado ou exportado. Por isso é fundamental sempre verificar se o diretório foi definido corretamente, geralmente usando as funções `dir()` ou `list.files()` para listar no console a lista de arquivos no diretório. Podemos ainda usar o argumento `pattern` para listar arquivos por um padrão textual.

```
## Listar os arquivos do diretório definido
dir()
list.files()

## Listar os arquivos do diretório definido por um padrão
dir(pattern = ".csv")
```

Além disso, é fundamental ressaltar a importância de verificar se o nome do arquivo que importaremos foi digitado corretamente, atentando-se também para a extensão: `.csv`, `.txt`, `.xlsx`, etc.

4.4 Estrutura e manipulação de objetos

O conhecimento sobre a estrutura e manipulação de objetos é fundamental para ter domínio e entendimento do funcionamento da linguagem R. Nesta seção, trataremos da estrutura e manipulação de dados no R, no que ficou conhecido como modo *R Base*, em contrapartida ao *tidyverse*, tópico tratado no Capítulo 5. Abordaremos aqui temas chaves, como: i) atributos de objetos, ii) manipulação de objetos unidimensionais e multidimensionais, iii) valores faltantes e especiais, iv) diretório de trabalho e v) importar, conferir e exportar dados tabulares.

4.4.1 Atributo dos objetos

Quando fazemos atribuições de dados no R (`<-`), os objetos gerados possuem três características.

1. **Nome**: palavra que o R reconhece os dados atribuídos
2. **Conteúdo**: dados em si
3. **Atributos**: modos (*natureza*) e estruturas (*organização*) dos elementos

Vamos explorar mais a fundo os **modos** e **estruturas** dos objetos. Vale ressaltar que isso é uma simplificação, pois há muitas classes de objetos, como funções e saídas de funções que possuem outros atributos.

Podemos verificar os atributos dos objetos com a função `attributes()`.

```
## Atributos
attributes(dune)
#> $names
#>  [1] "Achimill" "Agrostol" "Airaprae" "Alopgeni" "Anthodor" "Bellpere"
"Bromhord" "Chenalbu" "Cirsarve" "Comapalu"
#> [11] "Eleopalu" "Elymrepe" "Empenigr" "Hyporadi" "Juncarti" "Juncbufo"
"Lolipere" "Planlanc" "Poaprat"  "Poatriv"
#> [21] "Ranuflam" "Rumeacet" "Sagiproc" "Salirepe" "Scorautu" "Trifprat"
"Trifrepe" "Vicilath" "Bracruta" "Callcusp"
#>
#> $row.names
#>  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
"15" "16" "17" "18" "19" "20"
#>
#> $class
#> [1] "data.frame"
```

Modo dos objetos

A depender da natureza dos elementos que compõem os dados e que foram atribuídos aos objetos, esses objetos podem ser, de forma simples um dos cinco modos: numérico do tipo inteiro (*integer*), numérico do tipo flutuante (*double*), texto (*character*), lógico (*logical*) ou complexo (*complex*).

A atribuição de números no R pode gerar dois tipos de modos: **integer** para números inteiros e **double** para números flutuantes ou com decimais.

```
## Numérico double
obj_numerico_double <- 1

## Modo
mode(obj_numerico_double)
#> [1] "numeric"

## Tipo
typeof(obj_numerico_double)
#> [1] "double"
```

A título de praticidade, ambos são incorporados como o modo *numeric*, com o tipo *double*, a menos que especifiquemos que seja inteiro com a letra `L` depois do número, representando a palavra *Larger*, geralmente usando para armazenar números muito grandes.

```
## Numérico integer
obj_numerico_inteiro <- 1L

## Modo
mode(obj_numerico_inteiro)
#> [1] "numeric"

## Tipo
typeof(obj_numerico_inteiro)
#> [1] "integer"
```

Além de números, podemos atribuir textos, utilizando para isso aspas `""`.

```
## Caracter ou string
obj_caracter <- "a" # atencao para as aspas

## Modo
mode(obj_caracter)
#> [1] "character"
```

Em algumas situações, precisamos indicar a ocorrência ou não de um evento ou uma operação. Para isso, utilizamos as palavras reservadas (`TRUE` e `FALSE`), chamadas de variáveis booleanas, pois assumem apenas duas possibilidades: falso (0) ou verdadeiro (1). Devemos nos ater para o fato dessas palavras serem escritas com letras maiúsculas e sem aspas.

```
## Lógico
obj_logico <- TRUE # maiusculas e sem aspas

## Modo
mode(obj_logico)
#> [1] "logical"
```

Por fim, existe um modo pouco utilizado que cria números complexos (raiz de números negativos).

```
## Complexo
obj_complexo <- 1+1i

## Modo
mode(obj_complexo)
#> [1] "complex"
```

Podemos verificar o modo dos objetos ou fazer a conversão entre esses modos com diversas funções.

```
## Verificar o modo dos objetos
is.numeric()
is.integer()
is.character()
```

```
is.logical()
is.complex()

## Conversões entre modos
as.numeric()
as.integer()
as.character()
as.logical()
as.complex()

## Exemplo
num <- 1:5
num
mode(num)

cha <- as.factor(num)
cha
mode(cha)
```

Estrutura dos objetos

Uma vez entendido a natureza dos modos dos elementos dos objetos no R, podemos passar para o passo seguinte e entender como esses elementos são estruturados dentro dos objetos.

Essa estruturação irá nos contar sobre a organização dos elementos, com relação aos modos e dimensionalidade da disposição desses elementos (Figura 4.3). De modo bem simples, os elementos podem ser estruturados em cinco tipos:

1. **Vetores e fatores**: homogêneo (*um modo*) e unidimensional (*uma dimensão*). Um tipo especial de vetor são os fatores, usados para designar variáveis categóricas
2. **Matrizes**: homogêneo (*um modo*) e bidimensional (*duas dimensões*)
3. **Arrays**: homogêneo (*um modo*) e multidimensional (*mais de duas dimensões*)
4. **Data frames**: heterogêneo (*mais de um modo*) e bidimensional (*duas dimensões*)
5. **Listas**: heterogêneo (*mais de um modo*) e unidimensional (*uma dimensão*)

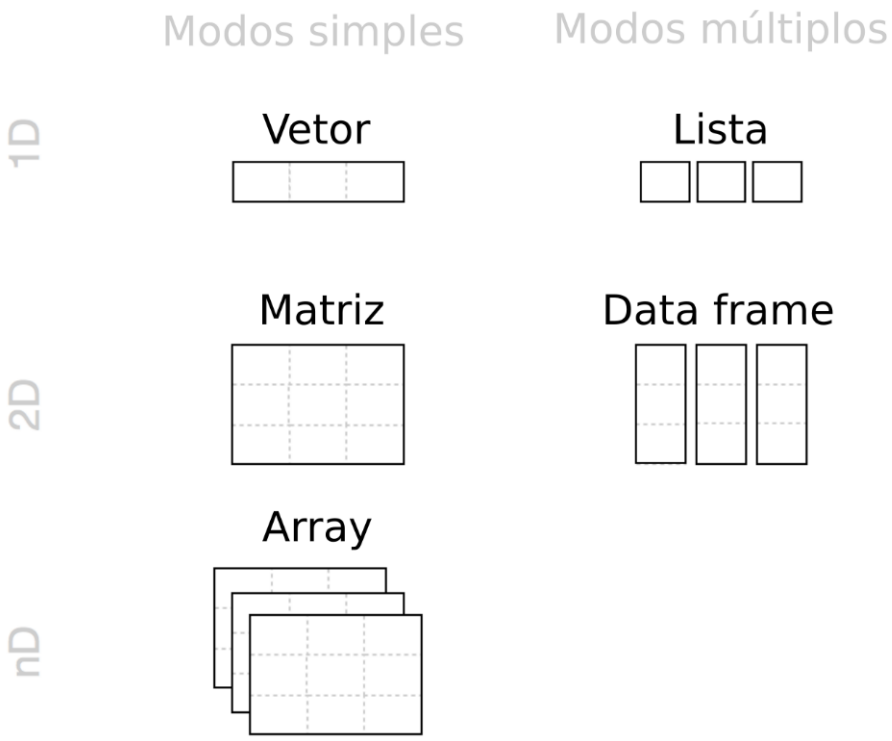


Figura 4.3: Estruturas de dados mais comuns no R: vetores, matrizes, arrays, listas e data frames. Adaptado de: Grolemund (2014).

Vetor

Vetores representam o encadeamento de elementos numa sequência unidimensional. No Capítulo 2 na Figura 2.2, vimos o conceito de variável aleatória e seus tipos. No R, essas variáveis podem ser operacionalizadas como vetores. Dessa forma, essa estrutura de dados pode ser traduzida como medidas de uma variável numérica (discretas ou contínuas), variável binária (booleana - TRUE e FALSE) ou descrição (informações em texto).

Há diversas formas de se criar um vetor no R:

1. Concatenando elementos com a função `c()`
2. Criando sequências unitárias `:` ou com a função `seq()`
3. Criando repetições com a função `rep()`
4. "Colar" palavras com uma sequência numérica com a função `paste()` ou `paste0()`
5. Amostrando aleatoriamente elementos com a função `sample()`

```
## Concatenar elementos numéricos
concatenar <- c(15, 18, 20, 22, 18)
concatenar
#> [1] 15 18 20 22 18

## Sequência unitária (x1:x2)
sequencia <- 1:10
sequencia
#>  [1]  1  2  3  4  5  6  7  8  9 10

## Sequência com diferentes espaçamentos
```

```
sequencia_esp <- seq(from = 0, to = 100, by = 10)
sequencia_esp
#>  [1]   0  10  20  30  40  50  60  70  80  90 100

## Repetição
repeticao <- rep(x = c(TRUE, FALSE), times = 5)
repeticao
#>  [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE

## Cola palavra e sequência numérica
colar <- paste("amostra", 1:5)
colar
#> [1] "amostra 1" "amostra 2" "amostra 3" "amostra 4" "amostra 5"

## Amostragem aleatória
amostragem <- sample(x = 1:100, size = 10)
amostragem
#>  [1] 45 23 76 63 47 31 68 73 69  5
```

Como os vetores são homogêneos, i.e., só comportam um modo, quando combinamos mais de um modo no mesmo objeto ocorre uma dominância de modos. Existe, dessa forma, uma **coerção** dos elementos combinados para que todos fiquem iguais. Essa dominância segue essa ordem:

`DOMINANTE` character > double > integer > logical `RECESSIVO`

Além disso, podemos utilizar as conversões listadas anteriormente para alterar os modos. Vamos exemplificar combinando os vetores criados anteriormente e convertendo-os.

```
## Coerção
c(colar, amostragem)
#>  [1] "amostra 1" "amostra 2" "amostra 3" "amostra 4" "amostra 5" "45"
"23"        "76"        "63"        "47"
#> [11] "31"        "68"        "73"        "69"        "5"

## Conversão
as.numeric(repeticao)
#>  [1] 1 0 1 0 1 0 1 0 1 0
```

Fator

O fator representa medidas de uma variável categórica, podendo ser nominal ou ordinal. É fundamental destacar que fatores no R devem ser entendidos como um vetor de **integer**, i.e., ele é composto por números inteiros representando os níveis da variável categórica.

Para criar um fator no R usamos uma função específica `factor()`, na qual podemos especificar os **níveis** com o argumento `level`, ou fazemos uma conversão usando a função `as.factor()`. Trabalhar com fatores no *R Base* não é das tarefas mais agradáveis, sendo assim, no Capítulo 5 usamos a versão *tidyverse* usando o pacote `forcats`. Destacamos ainda a existência de fatores nominais para variáveis

categóricas nominais e fatores ordinais para variáveis categóricas ordinais, quando há ordenamento entre os níveis, como dias da semana ou classes de altura.

```
## Fator nominal
fator_nominal <- factor(x = sample(x = c("floresta", "pastagem",
                                         "cerrado"),
                                   size = 20, replace = TRUE),
                        levels = c("floresta", "pastagem", "cerrado"))
fator_nominal
#>  [1] cerrado  cerrado  floresta pastagem pastagem cerrado  cerrado
pastagem cerrado  floresta floresta floresta pastagem
#> [14] pastagem cerrado  cerrado  pastagem cerrado  floresta pastagem
#> Levels: floresta pastagem cerrado

## Fator ordinal
fator_ordinal <- factor(x = sample(x = c("baixa", "media", "alta"),
                                   size = 20, replace = TRUE),
                        levels = c("baixa", "media", "alta"),
                        ordered = TRUE)
fator_ordinal
#>  [1] alta  alta  baixa media baixa media alta  media baixa media baixa
media alta  baixa media media alta  media baixa baixa
#> Levels: baixa < media < alta

## Conversão
fator <- as.factor(x = sample(x = c("floresta", "pastagem", "cerrado"),
                              size = 20, replace = TRUE))
fator
#>  [1] cerrado  pastagem floresta floresta cerrado  cerrado  pastagem
cerrado  pastagem cerrado  pastagem cerrado  pastagem
#> [14] pastagem floresta cerrado  pastagem pastagem cerrado  floresta
#> Levels: cerrado floresta pastagem
```

Matriz

A matriz representa dados no formato de tabela, com linhas e colunas. As linhas geralmente representam unidades amostrais (locais, transectos, parcelas) e as colunas representam variáveis numéricas (discretas ou contínuas), variáveis binárias (TRUE ou FALSE) ou descrições (informações em texto).

Podemos criar matrizes no R de duas formas. A primeira delas dispondo elementos de um vetor em um certo número de linhas e colunas com a função `matrix()`, podendo preencher essa matriz com os elementos do vetor por linhas ou por colunas alterando o argumento `byrow`.

```
## Vetor
ve <- 1:12

## Matrix - preenchimento por linhas - horizontal
ma_row <- matrix(data = ve, nrow = 4, ncol = 3, byrow = TRUE)
ma_row
#>      [,1] [,2] [,3]
#> [1,]    1    2    3
#> [2,]    4    5    6
#> [3,]    7    8    9
#> [4,]   10   11   12

## Matrix - preenchimento por colunas - vertical
ma_col <- matrix(data = ve, nrow = 4, ncol = 3, byrow = FALSE)
ma_col
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]    4    8   12
```

A segundo forma, podemos combinar vetores, utilizando a função `rbind()` para combinar vetores por linha, i.e., um vetor embaixo do outro, e `cbind()` para combinar vetores por coluna, i.e., um vetor ao lado do outro.

```
## Criar dois vetores
vec_1 <- c(1, 2, 3)
vec_2 <- c(4, 5, 6)

## Combinar por linhas - vertical - um embaixo do outro
ma_rbind <- rbind(vec_1, vec_2)
ma_rbind
#>       [,1] [,2] [,3]
#> vec_1    1    2    3
#> vec_2    4    5    6

## Combinar por colunas - horizontal - um ao lado do outro
ma_cbind <- cbind(vec_1, vec_2)
ma_cbind
#>      vec_1 vec_2
#> [1,]     1     4
#> [2,]     2     5
#> [3,]     3     6
```

Array

O array representa combinação de tabelas, com linhas, colunas e dimensões. Essa combinação pode ser feita em múltiplas dimensões, mas apesar disso, geralmente é mais comum o uso em Ecologia para três dimensões, por exemplo: linhas (unidades amostrais), colunas (espécies) e dimensão (tempo). Isso

gera um "cubo mágico" ou "cartas de um baralho," onde podemos comparar, nesse caso, comunidades ao longo do tempo. Além disso, arrays também são muito comuns em morfometria geométrica ou sensoriamento remoto.

Podemos criar arrays no R dispondo elementos de um vetor em um certo número de linhas, colunas e dimensões com a função `array()`. Em nosso exemplo, vamos compor cinco comunidades de cinco espécies ao longo de três períodos.

```
## Array
ar <- array(data = sample(x = c(0, 1), size = 75, rep = TRUE),
            dim = c(5, 5, 3))
ar
#> , , 1
#>
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    0    1    1    0
#> [2,]    1    0    1    1    0
#> [3,]    1    1    1    1    1
#> [4,]    0    1    0    1    0
#> [5,]    1    0    1    0    0
#>
#> , , 2
#>
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    0    0    0    1    0
#> [2,]    0    1    0    1    0
#> [3,]    1    1    1    1    0
#> [4,]    0    0    0    0    1
#> [5,]    0    0    0    0    0
#>
#> , , 3
#>
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    0    1    1    1    1
#> [2,]    0    0    0    0    0
#> [3,]    0    1    1    1    1
#> [4,]    0    0    1    0    0
#> [5,]    0    1    0    1    0
```

Data frame

O data frame também representa dados no formato de tabela, com linhas e colunas, muito semelhante à matriz. Mas diferentemente das matrizes, os data frames comportam mais de um modo em suas colunas. Dessa forma, as linhas do data frame ainda representam unidades amostrais (locais, transectos, parcelas), mas as colunas agora podem representar descrições (informações em texto), variáveis numéricas (discretas ou contínuas), variáveis binárias (TRUE ou FALSE) e variáveis categóricas (nominais ou ordinais).

A forma mais simples de se criar data frames no R é através da combinação de vetores. Essa combinação é feita com a função `data.frame()` e ocorre de forma horizontal, semelhante à função `cbind()`. Sendo assim, todos os vetores precisam ter o mesmo número de elementos, ou seja, o mesmo comprimento. Podemos ainda nomear as colunas de cada vetor. Outra forma, seria converter uma matriz em um data frame, utilizando a função `as.data.frame()`.

```
## Criar três vetores
vec_ch <- c("sp1", "sp2", "sp3")
vec_nu <- c(4, 5, 6)
vec_fa <- factor(c("campo", "floresta", "floresta"))

## Data frame - combinar por colunas - horizontal - um ao lado do outro
df <- data.frame(vec_ch, vec_nu, vec_fa)

df
#>   vec_ch vec_nu   vec_fa
#> 1    sp1      4    campo
#> 2    sp2      5 floresta
#> 3    sp3      6 floresta

## Data frame - nomear as colunas
df <- data.frame(especies = vec_ch,
                 abundancia = vec_nu,
                 vegetacao = vec_fa)
df
#>   especies abundancia vegetacao
#> 1      sp1          4     campo
#> 2      sp2          5  floresta
#> 3      sp3          6  floresta

## Data frame - converter uma matriz
ma <- matrix(data = ve, nrow = 4, ncol = 3, byrow = TRUE)
ma
#>      [,1] [,2] [,3]
#> [1,]    1    2    3
#> [2,]    4    5    6
#> [3,]    7    8    9
#> [4,]   10   11   12

df_ma <- as.data.frame(ma)
df_ma
#>   V1 V2 V3
#> 1  1  2  3
#> 2  4  5  6
#> 3  7  8  9
#> 4 10 11 12
```

Lista

A lista é um tipo especial de vetor que aceita objetos como elementos. Ela é a estrutura de dados utilizada para agrupar objetos, e é geralmente a saída de muitas funções.

Podemos criar listas através da função `list()`. Essa função funciona de forma semelhante à função `c()` para a criação de vetores, mas agora estamos concatenando objetos. Podemos ainda nomear os elementos (objetos) que estamos combinando.

Um ponto interessante para entender data frames, é que eles são listas, em que todos os elementos (colunas) possuem o mesmo número de elementos, ou seja, mesmo comprimento.

```
## Lista
lista <- list(rep(1, 20), # vector
              factor(1, 1), # factor
              cbind(c(1, 2), c(1, 2))) # matrix
lista
#> [[1]]
#>  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
#>
#> [[2]]
#> [1] 1
#> Levels: 1
#>
#> [[3]]
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    2

## Lista - nomear os elementos
lista_nome <- list(vector = rep(1, 20), # vector
              factor = factor(1, 1), # factor
              matrix = cbind(c(1, 2), c(1, 2))) # matrix
lista_nome
#> $vector
#>  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
#>
#> $factor
#> [1] 1
#> Levels: 1
#>
#> $matrix
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    2
```

Funções

Uma última estrutura de objetos criados no R são as funções. Elas são objetos criados pelo usuário e reutilizados para fazer operações específicas. A criação de funções geralmente é um tópico tratado num segundo momento, quando o usuário de R adquire certo conhecimento da linguagem. Aqui abordaremos apenas seu funcionamento básico, diferenciando sua estrutura para entendimento e sua diferenciação das demais estruturas.

Vamos criar uma função simples que retorna à multiplicação de dois termos. Criaremos a função com o nome `multi`, à qual será atribuída uma função com o nome `function()`, com dois argumentos `x` e `y`. Depois disso abrimos chaves `{}`, que é onde iremos incluir nosso bloco de código. Nosso bloco de código é composto por duas linhas, a primeira contendo a operação de multiplicação dos argumento com a atribuição ao objeto `mu` e a segunda contendo a função `return()` para retornar o valor da multiplicação.

```
## Criar uma função
multi <- function(x, y){

  mu <- (x * y)
  return(mu)

}
multi
#> function(x, y){
#>
#>   mu <- (x * y)
#>   return(mu)
#>
#> }

## Uso da função
multi(42, 23)
#> [1] 966
```

4.4.2 Manipulação de objetos unidimensionais

Vamos agora explorar formas de manipular elementos de objetos unidimensionais, ou seja, vetores, fatores e listas.

A primeira forma de manipulação é através da **indexação**, utilizando os operadores `[]`. Com a indexação podemos acessar elementos de vetores e fatores por sua posição. Utilizaremos números, sequência de números ou operações booleanas para retornar partes dos vetores ou fatores. Podemos ainda retirar elementos dessas estruturas com o operador aritmético `-`.

No exemplo a seguir, iremos fixar o ponto de partida da amostragem da função `sample()`, utilizando a função `set.seed(42)` (usamos 42 porque é a resposta para a vida, o universo e tudo mais - O Guia do Mochileiro das Galáxias, mas poderia ser outro número qualquer). Isso permite que o resultado da amostragem aleatória seja igual em diferentes computadores.

```r
## Fixar a amostragem
set.seed(42)

## Amostrar 10 elementos de uma sequência
ve <- sample(x = seq(0, 2, .05), size = 10)
ve
#>  [1] 1.80 0.00 1.20 0.45 1.75 0.85 1.15 0.30 1.90 0.20

## Seleciona o quinto elemento
ve[5]
#> [1] 1.75

## Seleciona os elementos de 1 a 5
ve[1:5]
#> [1] 1.80 0.00 1.20 0.45 1.75

## Retira o décimo elemento
ve[-10]
#> [1] 1.80 0.00 1.20 0.45 1.75 0.85 1.15 0.30 1.90

## Retira os elementos 2 a 9
ve[-(2:9)]
#> [1] 1.8 0.2
```

Podemos ainda fazer uma seleção condicional do vetor. Ao utilizarmos operadores relacionais, teremos como resposta um vetor lógico. Esse vetor lógico pode ser utilizado dentro da indexação para seleção de elementos.

```r
## Quais valores são maiores que 1?
ve > 1
#>  [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE

## Selecionar os valores acima de 1 no vetor ve
ve[ve > 1]
#> [1] 1.80 1.20 1.75 1.15 1.90
```

Além da indexação, temos algumas funções que nos auxiliam em algumas operações com objetos unidimensionais, listadas na Tabela 4.2.

Tabela 4.2: Funções para verificação e resumo de dados unidimensionais.

Função	Descrição
`max()`	Valor máximo
`min()`	Valor mínimo
`range()`	Amplitude
`length()`	Comprimento
`sum()`	Soma

Função	Descrição
`cumsum()`	Soma cumulativa
`prod()`	Produto
`sqrt()`	Raiz quadrada
`abs()`	Valor absoluto
`exp()`	Expoente
`log()`	Logaritmo natural
`log1p()`	Logaritmo natural mais 1 *log(x + 1)*
`log2()`	Logaritmo base 2
`log10()`	Logaritmo base 10
`mean()`	Média
`mean.weighted()`	Média ponderada
`var()`	Variância
`sd()`	Desvio Padrão
`mediam()`	Mediana
`quantile()`	Quantil
`quarters()`	Quartil
`IQR()`	Amplitude interquartil
`round()`	Arredondamento
`sort()`	Ordenação
`order()`	Posição ordenada
`rev()`	Reverso
`unique()`	Únicos
`summary()`	Resumo estatístico
`cut()`	Divide variável contínua em fator
`pretty()`	Divide variável contínua em intervalos
`scale()`	Padronização e centralização
`sub()`	Substitui caracteres
`grep()`	Posição de caracteres
`any()`	Algum valor?
`all()`	Todos os valores?
`which()`	Quais valores?
`subset()`	Subconjunto
`ifelse()`	Operação condicional

Para listas, também podemos usar a indexação `[]` para acessar ou retirar elementos.

```
## Lista
li <- list(elem1 = 1, elem2 = 2, elem3 = 3)

## Acessar o primeiro elemento
li[1]
#> $elem1
#> [1] 1

## Retirar o primeiro elemento
li[-1]
#> $elem2
#> [1] 2
#>
#> $elem3
#> [1] 3
```

Podemos ainda usar a indexação dupla `[[]]` para acessar os valores desses elementos.

```
## Acessar o valor do primeiro elemento
li[[1]]
#> [1] 1

## Acessar o valor do segundo elemento
li[[2]]
#> [1] 2
```

Para listas nomeadas, podemos ainda utilizar o operador `$` para acessar elementos pelo seu nome.

```
## Acessar o primeiro elemento
li$elem1
#> [1] 1
```

E ainda podemos utilizar funções para medir o comprimento dessa lista, listar os nomes dos elementos ou ainda renomear os elementos: `length()` e `names()`.

```
## Comprimento
length(li)
#> [1] 3

## Nomes
names(li)
#> [1] "elem1" "elem2" "elem3"

## Renomear
names(li) <- paste0("elemento0", 1:3)
li
#> $elemento01
#> [1] 1
#>
#> $elemento02
```

```
#> [1] 2
#>
#> $elemento03
#> [1] 3
```

4.4.3 Manipulação de objetos multidimensionais

Da mesma forma que para objetos unidimensionais, podemos manipular elementos de objetos multidimensionais, ou seja, matrizes, data frames e arrays.

Novamente, a primeira forma de manipulação é através da indexação, utilizando os operadores `[]`. Com a indexação podemos acessar elementos de matrizes, data frames e arrays por sua posição. Podemos ainda retirar elementos dessas estruturas com o operador aritmético `-`.

Entretanto, agora temos mais de uma dimensão na estruturação dos elementos dentro dos objetos. Assim, utilizamos números, sequência de números ou operação booleanas para retornar partes desses objetos, mas as dimensões têm de ser explicitadas e separadas por **vírgulas** para acessar linhas e colunas. Essa indexação funciona para matrizes e data frames. Para arrays, especificamos também as dimensões, também separadas por vírgulas para acessar essas dimensões.

```
## Matriz
ma <- matrix(1:12, 4, 3)
ma
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]    4    8   12

## Indexação
ma[3, ] # linha 3
#> [1]  3  7 11
ma[, 2] # coluna 2
#> [1] 5 6 7 8
ma[1, 2] # elemento da linha 1 e coluna 2
#> [1] 5
ma[1, 1:2] # elementos da linha 1 e coluna 1 e 2
#> [1] 1 5
ma[1, c(1, 3)] # elementos da linha 1 e coluna 1 e 3
#> [1] 1 9
ma[-1, ] # retirar a linha 1
#>      [,1] [,2] [,3]
#> [1,]    2    6   10
#> [2,]    3    7   11
#> [3,]    4    8   12
ma[, -3] # retirar a coluna 3
#>      [,1] [,2]
#> [1,]    1    5
```

```
#> [2,]    2    6
#> [3,]    3    7
#> [4,]    4    8
```

Para data frames, além de utilizar números e/ou sequências de números dentro do operador `[]` simples, podemos utilizar o operador `[[]]` duplo para retornar apenas os valores de uma linha ou uma coluna. Se as colunas estiverem nomeadas, podemos utilizar o nome da coluna de interesse entre aspas dentro dos operadores `[]` (retornar coluna) e `[[]]` (retornar apenas os valores), assim como ainda podemos utilizar o operador `$` para data frames. Essas últimas operações retornam um vetor, para o qual podemos fazer operações de vetores ou ainda atualizar o valor dessa coluna selecionada ou adicionar outra coluna.

```
## Criar três vetores
sp <- paste("sp", 1:10, sep = "")
abu <- 1:10
flo <- factor(rep(c("campo", "floresta"), each = 5))

## data frame
df <- data.frame(sp, abu, flo)
df
#>       sp abu      flo
#> 1    sp1   1    campo
#> 2    sp2   2    campo
#> 3    sp3   3    campo
#> 4    sp4   4    campo
#> 5    sp5   5    campo
#> 6    sp6   6 floresta
#> 7    sp7   7 floresta
#> 8    sp8   8 floresta
#> 9    sp9   9 floresta
#> 10 sp10  10 floresta

## [] - números
df[, 1]
#>  [1] "sp1"  "sp2"  "sp3"  "sp4"  "sp5"  "sp6"  "sp7"  "sp8"  "sp9"  "sp10"

## [] - nome das colunas - retorna coluna
df["flo"]
#>         flo
#> 1     campo
#> 2     campo
#> 3     campo
#> 4     campo
#> 5     campo
#> 6  floresta
#> 7  floresta
#> 8  floresta
#> 9  floresta
```

```
#> 10 floresta

## [[]] - nome das colunas - retorna apenas os valores
df[["flo"]]
#>  [1] campo    campo    campo    campo    campo    floresta floresta
floresta floresta floresta
#> Levels: campo floresta

## $ funciona apenas para data frame
df$sp
#>  [1] "sp1"  "sp2"  "sp3"  "sp4"  "sp5"  "sp6"  "sp7"  "sp8"  "sp9"  "sp10"

## Operação de vetors
length(df$abu)
#> [1] 10

## Converter colunas
df$abu <- as.character(df$abu)
mode(df$abu)
#> [1] "character"

## Adicionar ou mudar colunas
set.seed(42)
df$abu2 <- sample(x = 0:1, size = nrow(df), rep = TRUE)
df
#>      sp abu      flo abu2
#> 1   sp1   1    campo    0
#> 2   sp2   2    campo    0
#> 3   sp3   3    campo    0
#> 4   sp4   4    campo    0
#> 5   sp5   5    campo    1
#> 6   sp6   6 floresta    1
#> 7   sp7   7 floresta    1
#> 8   sp8   8 floresta    1
#> 9   sp9   9 floresta    0
#> 10 sp10  10 floresta    1
```

Podemos ainda fazer seleções condicionais para retornar linhas com valores que temos interesse, semelhante ao uso de filtro de uma planilha eletrônica.

```
## Selecionar linhas de uma matriz ou data frame
df[df$abu > 4, ]
#>    sp abu      flo abu2
#> 5 sp5   5    campo    1
#> 6 sp6   6 floresta    1
#> 7 sp7   7 floresta    1
#> 8 sp8   8 floresta    1
#> 9 sp9   9 floresta    0
```

```
df[df$flo == "floresta", ]
#>       sp abu      flo abu2
#> 6   sp6   6 floresta    1
#> 7   sp7   7 floresta    1
#> 8   sp8   8 floresta    1
#> 9   sp9   9 floresta    0
#> 10 sp10  10 floresta    1
```

Além disso, há uma série de funções para conferência e manipulação de dados que listamos na Tabela 4.3.

Tabela 4.3: Funções para verificação e resumo de dados multidimensionais.

Função	Descrição
`head()`	Mostra as primeiras 6 linhas
`tail()`	Mostra as últimas 6 linhas
`nrow()`	Mostra o número de linhas
`ncol()`	Mostra o número de colunas
`dim()`	Mostra o número de linhas e de colunas
`rownames()`	Mostra os nomes das linhas (locais)
`colnames()`	Mostra os nomes das colunas (variáveis)
`str()`	Mostra as classes de cada coluna (estrutura)
`summary()`	Mostra um resumo dos valores de cada coluna
`rowSums()`	Calcula a soma das linhas (horizontal)
`colSums()`	Calcula a soma das colunas (vertical)
`rowMeans()`	Calcula a média das linhas (horizontal)
`colMeans()`	Calcula a média das colunas (vertical)
`table()`	Tabulação cruzada
`t()`	Matriz ou data frame transposto

4.4.4 Valores faltantes e especiais

Valores faltantes e especiais são valores reservados que representam dados faltantes, indefinições matemáticas, infinitos e objetos nulos.

1. **NA (Not Available)**: significa dado faltante ou indisponível
2. **NaN (Not a Number)**: representa indefinições matemáticas
3. **Inf (Infinito)**: é um número muito grande ou um limite matemático
4. **NULL (Nulo)**: representa um objeto nulo, sendo útil para preenchimento em aplicações de programação

```
## Data frame com elemento NA
df <- data.frame(var1 = c(1, 4, 2, NA), var2 = c(1, 4, 5, 2))
df
#>   var1 var2
#> 1    1    1
#> 2    4    4
#> 3    2    5
#> 4   NA    2

## Resposta booleana para elementos NA
is.na(df)
#>       var1  var2
#> [1,] FALSE FALSE
#> [2,] FALSE FALSE
#> [3,] FALSE FALSE
#> [4,]  TRUE FALSE

## Algum elemento é NA?
any(is.na(df))
#> [1] TRUE

## Remover as linhas com NAs
df_sem_na <- na.omit(df)
df_sem_na
#>   var1 var2
#> 1    1    1
#> 2    4    4
#> 3    2    5

## Substituir NAs por 0
df[is.na(df)] <- 0
df
#>   var1 var2
#> 1    1    1
#> 2    4    4
#> 3    2    5
#> 4    0    2

## Desconsiderar os NAs em funções com o argumento rm.na = TRUE
sum(1, 2, 3, 4, NA, na.rm = TRUE)
#> [1] 10

## NaN - not a number
0/0
#> [1] NaN
log(-1)
#> [1] NaN
```

```
## Limite matemático
1/0
#> [1] Inf

## Número grande
10^310
#> [1] Inf

## Objeto nulo
nulo <- NULL
nulo
#> NULL
```

4.4.5 Diretório de trabalho

O diretório de trabalho é o endereço da pasta (ou diretório) de onde o R importará ou exportar nossos dados.

Podemos utilizar o próprio RStudio para tal tarefa, indo em `Session > Set Work Directory > Choose Directory...` ou simplesmente utilizar o atalho `Ctrl + Shift + H`.

Podemos ainda utilizar funções do R para definir o diretório. Para tanto, podemos navegar com o aplicativo de gerenciador de arquivos (e.g., Windows Explorer) até nosso diretório de interesse e copiar o endereço na barra superior. Voltamos para o R e colamos esse endereço entre aspas como argumento da função `setwd()`. É fundamental destacar que no Windows é necessário inverter as barras (`\` por `/` ou duplicar elas `\\`).

Aconselhamos ainda utilizar as funções `getwd()` para retornar o diretório definido na sessão do R, assim como as funções `dir()` ou `list.files()` para listagem dos arquivos no diretório, ambas medidas de conferência do diretório correto.

```
## Definir o diretório de trabalho
setwd("/home/mude/data/github/livro_aer/dados")

## Verificar o diretório
getwd()

## Listar os arquivos no diretório
dir()
list.files()
```

Outra forma de definir o diretório é digitar a tecla `tab` dentro da função `setwd("tab")`. Quando apertamos a `tab` dentro das aspas conseguimos selecionar o diretório manualmente, pois abre-se uma lista de diretório que podemos ir selecionando até chegar no diretório de interesse.

```
## Mudar o diretório com a tecla tab
setwd("`tab`")
```

4.4.6 Importar dados

Uma das operações mais corriqueiras do R, antes de realizar alguma análise ou plotar um gráfico, é a de importar dados que foram tabulados numa planilha eletrônica e salvos no formato .csv, .txt ou .xlsx. Ao importar esse tipo de dado para o R, o formato que o mesmo assume, se nenhum parâmetro for especificado, é o da classe `data frame`, prevendo que a planilha de dados possua colunas com diferentes modos.

Existem diversas formas de importar dados para o R. Podemos importar utilizando o RStudio, indo na janela **Environment** (Figura 4.2 - janela 3) e clicar em "**Importar Dataset**".

Entretanto, aconselhamos o uso de funções que fiquem salvas em um script para aumentar a reprodutibilidade do mesmo. Dessa forma, as três principais funções para importar os arquivos nos três principais extensões (.csv, .txt ou .xlsx) são, respectivamente: `read.csv()`, `read.table()` e `openxlsx::read.xlsx()`, sendo o último do pacote `openxlsx`.

Para exemplificar como importar dados no R, vamos usar os dados de comunidades de anfíbios da Mata Atlântica (Vancine et al. 2018). Faremos o download diretamente do site da fonte dos dados.

Vamos antes escolher um diretório de trabalho com a função `setwd()`, e em seguida criar um diretório com a função `dir.create()` chamado "dados." Depois, vamos mudar nosso diretório para essa pasta e criar mais um diretório chamado "tabelas," e por fim, definir esse diretório para que o conteúdo do download seja armazenado ali.

```
## Escolher um diretório
setwd("/home/mude/data/github/livro_aer")

## Criar um diretório 'dados'
dir.create("dados")

## Escolher diretório 'dados'
setwd("dados")

## Criar um diretório 'tabelas'
dir.create("tabelas")

## Escolher diretório 'tabelas'
setwd("tabelas")
```

Agora podemos fazer o download do arquivo `.zip` e extrair as tabelas usando a função `unzip()` nesse mesmo diretório.

```
## Download
download.file(url = "https://esajournals.onlinelibrary.wiley.com/action/downl
oadSupplement?doi=10.1002%2Fecy.2392&file=ecy2392-sup-0001-DataS1.zip",
              destfile = "atlantic_amphibians.zip", mode = "auto")

## Unzip
unzip(zipfile = "atlantic_amphibians.zip")
```

Agora podemos importar a tabela de dados com a função `read.csv()`, atribuindo ao objeto `intror_anfibios_locais`. Devemos atentar para o argumento `encoding`, que selecionamos aqui como `latin1` para corrigir um erro de caracteres, que o autor dos dados cometeu quando publicou esse data paper. Geralmente não precisamos especificar esse tipo de informação, mas caso depois de importar os dados e na conferência, os caracteres como acento não estiverem formatados, procure especificar no argumento `encoding` um padrão para corrigir esse erro.

```
## Importar a tabela de locais
intror_anfibios_locais <- read.csv(
                          "dados/tabelas/ATLANTIC_AMPHIBIANS_sites.csv",
                          encoding = "latin1")
```

Esse arquivo foi criado com separador de decimais sendo `.` e separador de colunas sendo `,`. Caso tivesse sido criado com separador de decimais sendo `,` e separador de colunas sendo `;`, usaríamos a função `read.csv2()`.

Para outros formatos, basta usar as outras funções apresentadas, atentando-se para os argumentos específicos de cada função.

Outra forma de importar dados, principalmente quando não sabemos exatamente o nome do arquivo e também para evitar erros de digitação, é utilizar a tecla `tab` dentro das aspas da função de importação. Dessa forma, conseguimos ter acesso aos arquivos do nosso diretório e temos a possibilidade de selecioná-los sem erros de digitação.

```
## Importar usando a tecla tab
intror_anfibios_locais <- read.csv("`tab`")
intror_anfibios_locais
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
intror_anfibios_locais <- ecodados::intror_anfibios_locais
head(intror_anfibios_locais)
```

4.4.7 Conferência dos dados importados

Uma vez importados os dados para o R, geralmente antes de iniciarmos qualquer manipulação, visualização ou análise de dados, fazemos a conferência desses dados. Para isso, podemos utilizar as funções listadas na Tabela 4.3.

Dentre as funções de verificação, destacamos a importância destas funções apresentadas abaixo para saber se as variáveis foram importadas e interpretadas corretamente e reconhecer erros de digitação, por exemplo.

```
## Primeiras linhas
head(intror_anfibios_locais)

## Últimas linhas
tail(intror_anfibios_locais)
```

```
## Número de linhas e colunas
nrow(intror_anfibios_locais)
#> [1] 1163
ncol(intror_anfibios_locais)
#> [1] 25
dim(intror_anfibios_locais)
#> [1] 1163   25
## Nome das linhas e colunas
rownames(intror_anfibios_locais)
colnames(intror_anfibios_locais)

## Estrutura dos dados
str(intror_anfibios_locais)

## Resumo dos dados
summary(intror_anfibios_locais)

## Verificar NAs
any(is.na(intror_anfibios_locais))
which(is.na(intror_anfibios_locais))

## Remover as linhas com NAs
intror_anfibios_locais_na <- na.omit(intror_anfibios_locais)
```

> 👉 **Importante**
> A função `na.omit()` retira a **linha inteira** que possui algum `NA`, inclusive as colunas que possuem dados que você não tem interesse em excluir. Dessa forma, tenha em mente quais dados você realmente quer remover da sua tabela.

Além das funções apresentadas, recomendamos olhar os seguintes pacotes que ajudam na conferência dos dados importados: `Hmisc`, `skimr` e `inspectDF`.

4.4.8 Exportar dados

Uma vez realizado as operações de manipulação ou tendo dados que foram analisados e armazenados num objeto no formato de data frame ou matriz, podemos exportar esses dados do R para o diretório que definimos anteriormente.

Para tanto, podemos utilizar funções de escrita de dados, como `write.csv()`, `write.table()` e `openxlsx::write.xlsx()`. Dois pontos são fundamentais: i) o nome do arquivo tem de estar entre aspas e no final dele deve constar a extensão que pretendemos que o arquivo tenha, e ii) é interessante utilizar os argumentos **row.names = FALSE** e **quote=FALSE**, para que o arquivo escrito não tenha o nome das linhas ou aspas em todas as células, respectivamente.

```
## Exportar dados na extensão .csv
write.csv(intror_anfibios_locais_na, "ATLANTIC_AMPHIBIAN_sites_na.csv",
```

```
          row.names = FALSE, quote = FALSE)

## Exportar dados na extensão .txt
write.table(intror_anfibios_locais_na, "ATLANTIC_AMPHIBIAN_sites_na.txt",
            row.names = FALSE, quote = FALSE)

## Exportar dados na extensão .xlsx
openxlsx::write.xlsx(intror_anfibios_locais_na,
                     "ATLANTIC_AMPHIBIAN_sites_na.xlsx",
                     row.names = FALSE, quote = FALSE)
```

4.5 Para se aprofundar

Listamos a seguir livros e links com material que recomendamos para seguir com sua aprendizagem em *R Base*.

4.5.1 Livros

Recomendamos aos interessados(as) os livros: i) Crawley (2012) *The R Book*, ii) Davies (2016) *The Book of R: A First Course in Programming and Statistics*, iii) Gillespie e Lovelace (2017) *Efficient R programming*, iv) Holmes e Huber (2019) *Modern Statistics for Modern Biology*, v) Irizarry e Love (2017) *Data Analysis for the Life Sciences with R*, vi) James et al. (2013) *An Introduction to Statistical Learning: with Applications in R*, vii) Kabacoff (2011) *R in Action: Data analysis and graphics with R*, viii) Matloff (2011) *The Art of R Programming: A Tour of Statistical Software Design*, ix) Long e Teetor (2019) *R Cookbook* e x) Wickham (2019) *Advanced R*.

4.5.2 Links

Existem centenas de ferramentas online para aprender e explorar o R. Dentre elas, indicamos os seguintes links (em português e inglês):

Introdução ao R

- An Introduction to R - Douglas A, Roos D, Mancini F, Couto A, Lusseau D
- A (very) shortintroduction to R - Paul Torfs & Claudia Brauer
- R for Beginners - Emmanuel Paradis

Ciência de dados

- Ciência de Dados em R - Curso-R
- Data Science for Ecologists and Environmental Scientists - Coding Club

Estatística

- Estatística Computacional com R - Mayer F. P., Bonat W. H., Zeviani W. M., Krainski E. T., Ribeiro Jr. P. J
- Data Analysis and Visualization in R for Ecologists - Data Carpentry

Miscelânea

- Materiais sobre R - Beatriz Milz
- R resources (free courses, books, tutorials, & cheat sheets) - Paul van der Laken

4.6 Exercícios

4.1 Use o R para verificar o resultado da operação `7 + 7 ÷ 7 + 7 x 7 - 7`.

4.2 Verifique através do R se `3x2³` é maior que `2x3²`.

4.3 Crie dois objetos (qualquer nome) com os valores 100 e 300. Multiplique esses objetos (função `prod()`) e atribuam ao objeto `mult`. Faça o logaritmo natural (função `log()`) do objeto `mult` e atribuam ao objeto `ln`.

4.4 Quantos pacotes existem no CRAN nesse momento? Execute essa combinação no Console: `nrow(available.packages(repos = "http://cran.r-project.org"))`.

4.5 Instale o pacote `tidyverse` do CRAN.

4.6 Escolha números para jogar na mega-sena usando o R, nomeando o objeto como `mega`. Lembrando: são 6 valores de 1 a 60 e atribuam a um objeto.

4.7 Crie um fator chamado `tr`, com dois níveis ("cont" e "trat") para descrever 100 locais de amostragem, 50 de cada tratamento. O fator deve ser dessa forma `cont, cont, cont, ...., cont, trat, trat, ...., trat`.

4.8 Crie uma matriz chamada `ma`, resultante da disposição de um vetor composto por 1000 valores aleatórios entre 0 e 10. A matriz deve conter 100 linhas e ser disposta por colunas.

4.9 Crie um data frame chamado `df`, resultante da composição desses vetores:

- `id: 1:50`
- `sp: sp01, sp02, ..., sp49, sp50`
- `ab: 50 valores aleatórios entre 0 a 5`

4.10 Crie uma lista com os objetos criados anteriormente: `mega`, `tr`, `ma` e `df`.

4.11 Selecione os elementos ímpares do objeto `tr` e atribua ao objeto `tr_impar`.

4.12 Selecione as linhas com ids pares do objeto `df` e atribua ao objeto `df_ids_par`.

4.13 Faça uma amostragem de 10 linhas do objeto `df` e atribua ao objeto `df_amos10`.

4.14 Amostre 10 linhas do objeto `ma`, mas utilizando as linhas amostradas do `df_amos10` e atribua ao objeto `ma_amos10`.

4.15 Una as colunas dos objetos `df_amos10` e `ma_amos10` e atribua ao objeto `dados_amos10`.

Soluções dos exercícios.

Capítulo 5

Tidyverse

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(tidyverse)
library(here)
library(ggplot2)
library(purrr)
library(tibble)
library(dplyr)
library(tidyr)
library(stringr)
library(readr)
library(forcats)
library(palmerpenguins)
library(lubridate)

## Dados
penguins <- palmerpenguins::penguins
penguins_raw <- palmerpenguins::penguins_raw
tidy_anfibios_locais <- ecodados::tidy_anfibios_locais
```

5.1 Contextualização

Como todo idioma, a linguagem R vem passando por transformações nos últimos anos. Grande parte dessas mudanças estão dentro do paradigma de Ciência de Dados (*Data Science*), uma nova área de conhecimento que vem se moldando a partir do desenvolvimento da sociedade em torno da era digital e da grande quantidade de dados gerados e disponíveis pela internet, de onde advém os pilares das inovações tecnológicas: *Big Data*, *Machine Learning* e *Internet of Things*. A grande necessidade de computação para desenvolver esse novo paradigma colocaram o R e o python como as principais linguagens de programação frente a esses novos desafios. Apesar de não serem as únicas ferramentas utilizadas para esse propósito, elas rapidamente se tornaram uma das melhores escolhas, dado vários fatores como: ser de código-aberto e gratuitas, possuir grandes comunidades contribuidoras, ser linguagens de interpretação (orientadas a objeto) e relativamente fáceis de serem aprendidas e aplicadas.

Essas mudanças e expansões na utilização da linguagem R para a Ciência de Dados começaram a ser implementadas principalmente devido a um pesquisador: Hadley Wickham, que iniciou sua contribuição à comunidade R com o desenvolvimento do já consagrado pacote `ggplot2` (Wickham 2016) para a composição de gráficos no R (ver mais no Capítulo 6), baseado na gramática de gráficos (Wilkinson & Wills 2005). Depois disso, Wickham dedicou-se ao desenvolvimento do pensamento de uma nova abordagem dentro da manipulação de dados, denominada **Tidy Data** (Dados organizados) (Wickham 2014), na qual focou na limpeza e organização dos mesmos. A ideia postula que dados estão `tidy` quando: i) variáveis estão nas colunas, ii) observações estão nas linhas e iii) valores estão nas células, sendo que para esse último, não deve haver mais de um valor por célula (Figura 5.2).

A partir dessas ideias, o *tidyverse* foi operacionalizado no R como uma coleção de pacotes que atuam no fluxo de trabalho comum da ciência de dados: importação, manipulação, exploração, visualização, análise e comunicação de dados e análises (Wickham et al. 2019) (Figura 5.1). O principal objetivo do *tidyverse* é aproximar a linguagem para melhorar a interação entre ser humano e computador sobre dados, de modo que os pacotes compartilham uma filosofia de design de alto nível e gramática, além da estrutura de dados de baixo nível (Wickham et al. 2019). As principais leituras sobre o tema no R são os artigos "*Tidy Data*" (Wickham 2014) e "*Welcome to the Tidyverse*" (Wickham et al. 2019), e o livro *"R for Data Science"* (Wickham & Grolemund 2017), além do *Tidyverse* que possui muito mais informações.

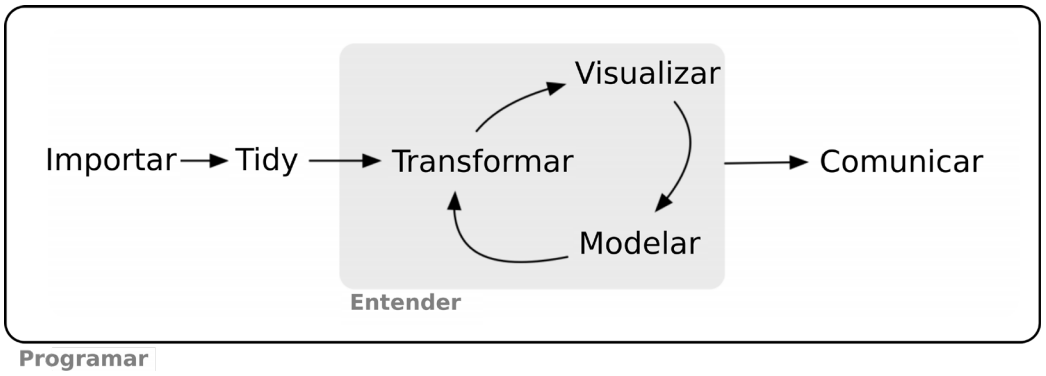


Figura 5.1: Modelo das ferramentas necessárias em um projeto típico de ciência de dados: importar, organizar, entender (transformar, visualizar, modelar) e comunicar, envolto à essas ferramentas está a programação. Adaptado de Wickham & Grolemund (2017).

5.2 *tidyverse*

Uma vez instalado e carregado, o pacote `tidyverse` disponibiliza um conjunto de ferramentas através de vários pacotes. Esses pacotes compartilham uma filosofia de design, gramática e estruturas. Podemos entender o *tidyverse* como um "dialeto novo" para a linguagem R, onde *tidy* quer dizer organizado, arrumado, ordenado, e *verse* é universo. A seguir, listamos os principais pacotes e suas funcionalidades.

- `readr`: importa dados tabulares (e.g., `.csv` e `.txt`)
- `tibble`: implementa a classe `tibble`
- `tidyr`: transformação de dados para `tidy`
- `dplyr`: manipulação de dados
- `stringr`: manipulação de caracteres
- `forcats`: manipulação de fatores
- `ggplot2`: possibilita a visualização de dados
- `purrr`: disponibiliza ferramentas para programação funcional

Além dos pacotes principais, fazemos também menção a outros pacotes que estão dentro dessa abordagem e que trataremos ainda neste capítulo, em outro momento do livro, ou que você leitor(a) deve se familiarizar. Alguns pacotes compõem o *tidyverse* outros são mais gerais, entretanto, todos estão envolvidos de alguma forma com ciência de dados.

- `readxl` e `writexl`: importa e exporta dados tabulares (.xlsx)
- `janitor`: examina e limpa dados sujos

- `DBI`: interface de banco de dados R
- `haven`: importa e exporta dados do SPSS, Stata e SAS
- `httr`: ferramentas para trabalhar com URLs e HTTP
- `rvest`: coleta facilmente (raspagem de dados) páginas da web
- `xml2`: trabalha com arquivos XML
- `jsonlite`: um analisador e gerador JSON simples e robusto para R
- `hms`: hora do dia
- `lubridate`: facilita o tratamento de datas
- `magrittr`: provê os operadores pipe (`%>%`, `%$%`, `%<>%`)
- `glue`: facilita a combinação de dados e caracteres
- `rmarkdown`: cria documentos de análise dinâmica que combinam código, saída renderizada (como figuras) e texto
- `knitr`: projetado para ser um mecanismo transparente para geração de relatórios dinâmicos com R
- `shiny`: framework de aplicativo Web para R
- `flexdashboard`: painéis interativos para R
- `here`: facilita a definição de diretórios
- `usethis`: automatiza tarefas durante a configuração e desenvolvimento de projetos (Git, 'GitHub' e Projetos RStudio)
- `data.table`: pacote que fornece uma versão de alto desempenho do `data.frame` (importar, manipular e expotar)
- `reticulate`: pacote que fornece ferramentas para integrar Python e R
- `sparklyr`: interface R para *Apache Spark*
- `broom`: converte objetos estatísticos em *tibbles* organizados
- `modelr`: funções de modelagem que funcionam com o pipe
- `tidymodels`: coleção de pacotes para modelagem e aprendizado de máquina usando os princípios do *tidyverse*

Destacamos a grande expansão e aplicabilidade dos pacotes rmarkdown, knitr e bookdown, que permitiram a escrita deste livro usando essas ferramentas e linguagem de marcação, chamada Markdown.

Para instalar os principais pacotes que integram o *tidyverse* podemos instalar o pacote `tidyverse`.

```
## Instalar o pacote tidyverse
install.packages("tidyverse")
```

Quando carregamos o pacote `tidyverse` podemos notar uma mensagem indicando quais pacotes foram carregados, suas respectivas versões e os conflitos com outros pacotes.

```
## Carregar o pacote tidyverse
library(tidyverse)
```

Podemos ainda listar todos os pacotes do *tidyverse* com a função `tidyverse::tidyverse_packages()`.

```
## Listar todos os pacotes do tidyverse
tidyverse::tidyverse_packages()
#>  [1] "broom"         "cli"           "crayon"        "dbplyr"
"dplyr"         "dtplyr"        "forcats"
```

```
#>  [8] "googledrive"   "googlesheets4" "ggplot2"       "haven"         "hms"
"httr"          "jsonlite"
#> [15] "lubridate"     "magrittr"      "modelr"        "pillar"
"purrr"         "readr"         "readxl"
#> [22] "reprex"        "rlang"         "rstudioapi"    "rvest"
"stringr"       "tibble"        "tidyr"
#> [29] "xml2"          "tidyverse"
```

Também podemos verificar se os pacotes estão atualizados, senão, podemos atualizá-los com a função `tidyverse::tidyverse_update()`.

```
## Verificar e atualizar os pacotes do tidyverse
tidyverse::tidyverse_update(repos = "http://cran.us.r-project.org")
```

Todas as funções dos pacotes tidyverse usam **fonte minúscula** e **_ (**underscore**)** para separar os nomes internos das funções, seguindo a mesma sintaxe do Python ("*Snake Case*"). Neste sentido de padronização, é importante destacar ainda que existe um guia próprio para que os scripts sigam a recomendação de padronização, o *The tidyverse style guide*, criado pelo próprio Hadley Wickham. Para pessoas que desenvolvem funções e pacotes existe o *Tidyverse design guide* criado pelo *Tidyverse team*.

```
## Funções no formato snake case
read_csv()
read_tsv()
as_tibble()
left_join()
group_by()
```

Por fim, para evitar possíveis conflitos de funções com o mesmo nome entre pacotes, recomendamos fortemente o hábito de usar as funções precedidas do operador `::` e o respectivo pacote. Assim, garante-se que a função utilizada é referente ao pacote daquela função. Segue um exemplo com as funções apresentadas anteriormente.

```
## Funções seguidas de seus respectivos pacotes
readr::read_csv()
readr::read_tsv()
tibble::as_tibble()
dplyr::left_join()
dplyr::group_by()
```

Seguindo essas ideias do novo paradigma da **Ciência de Dados**, outro conjunto de pacotes foi desenvolvido, chamado de `tidymodels` que atuam no fluxo de trabalho da análise de dados em ciência de dados: separação e reamostragem, pré-processamento, ajuste de modelos e métricas de performasse de ajustes. Por razões de espaço e especificidade, não entraremos em detalhes desses pacotes.

Seguindo a estrutura da Figura 5.1, iremos ver nos itens das próximas seções como esses passos são realizados com funções de cada pacote.

5.3 here

Dentro do fluxo de trabalho do *tidyverse*, devemos sempre trabalhar com **Projetos do RStudio** (ver Capítulo 4). Junto com o projeto, também podemos fazer uso do pacote `here`. Ele permite construir caminhos para os arquivos do projeto de forma mais simples e com maior reprodutibilidade.

Esse pacote cobre o ponto de mudarmos o diretório de trabalho que discutimos no Capítulo 4, dado que muitas vezes mudar o diretório com a função `setwd()` tende a ser demorado e tedioso, principalmente quando se trata de um script em que várias pessoas estão trabalhando em diferentes computadores e sistemas operacionais. Além disso, ele elimina a questão da fragilidade dos scripts, pois geralmente um script está com os diretórios conectados exatamente a um lugar e a um momento. Por fim, ele também simplifica o trabalho com subdiretórios, facilitando importar ou exportar arquivos para subdiretórios.

Seu uso é relativamente simples: uma vez criado e aberto o RStudio pelo Projeto do RStudio, o diretório automaticamente é definido para o diretório do projeto. Depois disso, podemos usar a função `here::here()` para definir os subdiretórios onde estão os dados. O exemplo da aplicação fica para a seção seguinte, quando iremos de fato importar um arquivo tabular para o R. Logo abaixo, mostramos como instalar e carregar o pacote `here`.

```
## Instalar
install.packages("here")

## Carregar
library(here)
```

5.4 readr, readxl e writexl

Dado que possuímos um conjunto de dados e que geralmente esse conjunto de dados estará no formato tabular com umas das extensões: .csv, .txt ou .xlsx, usaremos o pacote `readr` ou `readxl` para importar esses dados para o R. Esses pacotes leem e escrevem grandes arquivos de forma mais rápida, além de fornecerem medidores de progresso de importação e exportação, e imprimir a informação dos modos das colunas no momento da importação. Outro ponto bastante positivo é que também classificam automaticamente o modo dos dados de cada coluna, i.e., se uma coluna possui dados numéricos ou apenas texto, essa informação será considerada para classificar o modo da coluna toda. A classe do objeto atribuído quando lido por esses pacotes é automaticamente um `tibble`, que veremos melhor na seção seguinte. Todas as funções deste pacote são listadas na página de referência do pacote.

Usamos as funções `readr::read_csv()` e `readr::write_csv()` para importar e exportar arquivos .csv do R, respectivamente. Para dados com a extensão .txt, podemos utilizar as funções `readr::read_tsv()` ou ainda `readr::read_delim()`. Para arquivos tabulares com a extensão .xlsx, temos de instalar e carregar dois pacotes adicionais: `readxl` e `writexl`, dos quais usaremos as funções `readxl::read_excel()`, `readxl::read_xlsx()` ou `readxl::read_xls()` para importar dados, atentado para o fato de podermos indicar a aba com os dados com o argumento `sheet`, e `writexl::write_xlsx()` para exportar.

Se o arquivo .csv foi criado com separador de decimais sendo `.` e separador de colunas sendo `,`, usamos as funções listadas acima normalmente. Caso seja criado com separador de decimais sendo

`,` e separador de colunas sendo `;`, devemos usar a função `readr::read_csv2()` para importar e `readr::write_csv2()` para exportar nesse formato, que é mais comum no Brasil.

Para exemplificar como essas funções funcionam, vamos importar novamente os dados de comunidades de anfíbios da Mata Atlântica (Vancine et al. 2018), que fizemos o download no Capítulo 4. Estamos usando a função `readr::read_csv()`, indicando os diretórios com a função `here::here()`, e a classe do arquivo é `tibble`. Devemos atentar para o argumento `locale = readr::locale(encoding = "latin1")`,que selecionamos aqui como `latin1` para corrigir um erro de caracteres, que o autor dos dados cometeu quando publicou esse data paper.

```
## Importar locais
tidy_anfibios_locais <- readr::read_csv(
    here::here("dados", "tabelas", "ATLANTIC_AMPHIBIANS_sites.csv"),
    locale = readr::locale(encoding = "latin1")
)
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
tidy_anfibios_locais <- ecodados::tidy_anfibios_locais
head(tidy_anfibios_locais)
```

Para se aprofundar no tema, recomendamos a leitura do Capítulo 11 *Data import* de Wickham & Grolemund (2017).

5.5 tibble

O `tibble` (`tbl_sf`) é uma versão aprimorada do data frame (`data.frame`). Ele é a classe aconselhada para que as funções do tidyverse funcionem melhor sobre conjuntos de dados tabulares importados para o R.

Geralmente, quando utilizamos funções *tidyverse* para importar dados para o R, é essa classe que os dados adquirem. Além da importação de dados, podemos criar um *tibble* no R usando a função `tibble::tibble()`, semelhante ao uso da função `data.frame()`. Podemos ainda converter um `data.frame` para um `tibble` usando a função `tibble::as_tibble()`. Entretanto, em alguns momentos precisaremos da classe `data.frame` para algumas funções específicas, e podemos converter um `tibble` para `data.frame` usando a função `tibble::as_data_frame()`.

Existem duas diferenças principais no uso do `tibble` e do `data.frame`: impressão e subconjunto. Objetos da classe `tibbles` possuem um método de impressão que mostra a contagem do número de linhas e colunas, e apenas as primeiras 10 linhas e todas as colunas que couberem na tela no console, além dos modos ou tipos das colunas. Dessa forma, cada coluna ou variável, pode ser do modo numbers (`int` ou `dbl`), character (`chr`), logical (`lgl`), factor (`fctr`), date + time (`dttm`) e date (`date`), além de outras inúmeras possibilidades.

Todas as funções deste pacote são listadas na página de referência do pacote.

```
## Tibble - impressão
tidy_anfibios_locais
```

Para o subconjunto, como vimos no Capítulo 4, para selecionar colunas e linhas de objetos bidimensionais podemos utilizar os operadores `[]` ou `[[]]`, associado com números separados por vírgulas ou o nome da coluna entre aspas, e o operador `$` para extrair uma coluna pelo seu nome. Comparando um `data.frame` a um `tibble`, o último é mais rígido na seleção das colunas: ele nunca faz correspondência parcial e gera um aviso se a coluna que você está tentando acessar não existe.

```
## Tibble - subconjunto
tidy_anfibios_locais$ref
#> Warning: Unknown or uninitialised column: `ref`.
#> NULL
```

Por fim, podemos "espiar" os dados utilizando a função `tibble::glimpse()` para ter uma noção geral de número de linhas, colunas, e conteúdo de todas as colunas. Essa é a função *tidyverse* da função R Base `str()`.

```
## Espiar os dados
tibble::glimpse(tidy_anfibios_locais[, 1:10])
```

Para se aprofundar no tema, recomendamos a leitura do Capítulo 10 *Tibbles* de Wickham & Grolemund (2017).

5.6 magrittr (pipe - %>%)

O operador pipe `%>%` permite o encadeamento de várias funções, eliminando a necessidade de criar objetos para armazenar resultados intermediários. Dessa forma, pipes são uma ferramenta poderosa para expressar uma sequência de múltiplas operações.

O operador pipe `%>%` vem do pacote `magrittr`, entretanto, todos os pacotes no tidyverse automaticamente tornam o pipe disponível. Essa função torna os códigos em R mais simples, pois podemos realizar múltiplas operações em uma única linha. Ele captura o resultado de uma declaração e o torna a primeira entrada da próxima declaração, então podemos pensar como "EM SEGUIDA FAÇA" ao final de cada linha de código.

Todas as funções deste pacote são listadas na página de referência do pacote.

A principal vantagem do uso dos pipes é facilitar a depuração (*debugging* - achar erros) nos códigos, porque seu uso torna a linguagem R mais próxima do que falamos e pensamos, uma vez que evita o uso de funções dentro de funções (funções compostas, lembra-se do fog e gof do ensino médio? Evitamos eles aqui também).

Digitar `%>%` é um pouco chato, dessa forma, existe um atalho para sua inserção nos scripts: `Ctrl + Shift + M`.

Para deixar esse tópico menos estranho a quem possa ver essa operação pela primeira vez, vamos fazer alguns exemplos.

```
## R Base - sem pipe
sqrt(sum(1:100))
#> [1] 71.06335
```

```
## Tidyverse - com pipe
1:100 %>%
    sum() %>%
    sqrt()
#> [1] 71.06335
```

Essas operações ainda estão simples, vamos torná-las mais complexas com várias funções compostas. É nesses casos que a propriedade organizacional do uso do pipe emerge: podemos facilmente ver o encadeamento de operações, onde cada função é disposta numa linha. Apenas um adendo: a função `set.seed()` fixa a amostragem de funções que geram valores aleatórios, como é o caso da função `rpois()`.

```
## Fixar amostragem
set.seed(42)

## Base R - sem pipe
ve <- sum(sqrt(sin(log10(rpois(100, 10)))))
ve
#> [1] 91.27018

## Fixar amostragem
set.seed(42)

## Tidyverse - com pipe
ve <- rpois(100, 10) %>%
    log10() %>%
    sin() %>%
    sqrt() %>%
    sum()
ve
#> [1] 91.27018
```

O uso do pipe vai se tornar especialmente útil quando seguirmos para os pacotes das próximas duas seções: `tidyr` e `dplyr`. Com esses pacotes faremos operações em linhas e colunas de nossos dados tabulares, então podemos encadear uma série de funções para manipulação, limpeza e análise de dados.

Há ainda três outras variações do pipe que podem ser úteis em alguns momentos, mas que para funcionar precisam que o pacote `magrittr` esteja carregado:

- `%T>%`: retorna o lado esquerdo em vez do lado direito da operação
- `%$%`: "explode" as variáveis em um quadro de dados
- `%<>%`: permite atribuição usando pipes

Para se aprofundar no tema, recomendamos a leitura do Capítulo 18 *Pipes* de Wickham & Grolemund (2017).

Importante
A partir da versão do R 4.1+ (18/05/2021), o operador pipe se tornou nativo do R. Entretanto, o operador foi atualizado para `|>`, podendo ser inserido com o mesmo atalho `Ctrl + Shift + M`, mas necessitando uma mudança de opção em `Tools > Global Options > Code > [x] Use native pipe operator, |> (requires R 4.1+)`, requerendo que o RStudio esteja numa versão igual ou superior a 1.4.17+.

5.7 tidyr

Um conjunto de dados `tidy` (organizados) são mais fáceis de manipular, modelar e visualizar. Um conjunto de dados está no formato `tidy` ou não, dependendo de como linhas, colunas e células são combinadas com observações, variáveis e valores. Nos dados tidy, as variáveis estão nas colunas, observações estão nas linhas e valores estão nas células, sendo que para esse último, não deve haver mais de um valor por célula (Figura 5.2).

1. Cada variável em uma coluna
2. Cada observação em uma linha
3. Cada valor como uma célula

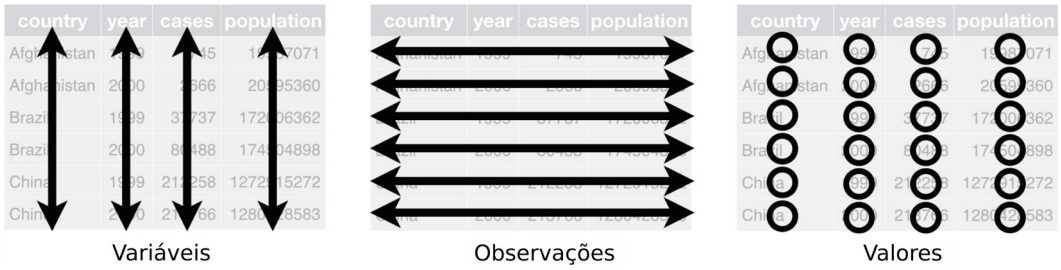


Figura 5.2: As três regras que tornam um conjunto de dados tidy. Adaptado de Wickham & Grolemund (2017).

Todas as funções deste pacote são listadas na página de referência do pacote.

Para realizar diversas transformações nos dados, a fim de ajustá-los ao formato `tidy` existe uma série de funções para: unir colunas, separar colunas, lidar com valores faltantes (`NA`), transformar a base de dados de formato longo para largo (ou vice-e-versa), além de outras funções específicas.

- `unite()`: junta dados de múltiplas colunas em uma coluna
- `separate()`: separa caracteres em múltiplas colunas
- `separate_rows()`: separa caracteres em múltiplas colunas e linhas
- `drop_na()`: retira linhas com `NA` do conjunto de dados
- `replace_na()`: substitui `NA` do conjunto de dados
- `pivot_wider()`: transforma um conjunto de dados longo (*long*) para largo (*wide*)
- `pivot_longer()`: transforma um conjunto de dados largo (*wide*) para longo (*long*)

5.7.1 palmerpenguins

Para exemplificar o funcionamento dessas funções, usaremos os dados de medidas de pinguins chamados *palmerpenguins*, disponíveis no pacote `palmerpenguins`.

```
## Instalar o pacote
install.packages("palmerpenguins")
```

Esses dados foram coletados e disponibilizados pela Dra. Kristen Gorman e pela Palmer Station, Antarctica LTER, membro da Long Term Ecological Research Network.

O pacote `palmerpenguins` contém dois conjuntos de dados. Um é chamado de `penguins` e é uma versão simplificada dos dados brutos. O segundo conjunto de dados é `penguins_raw` e contém todas as variáveis e nomes originais. Ambos os conjuntos de dados contêm dados para 344 pinguins, de três espécies diferentes, coletados em três ilhas no arquipélago de Palmer, na Antártica. Destacamos também a versão traduzida desses dados para o português, disponível no pacote `dados`.

Vamos utilizar principalmente o conjunto de dados `penguins_raw`, que é a versão dos dados brutos.

```
## Carregar o pacote palmerpenguins
library(palmerpenguins)
```

Podemos ainda verificar os dados, pedindo uma ajuda de cada um dos objetos.

```
## Ajuda dos dados
?penguins
?penguins_raw
```

5.7.2 glimpse()

Primeiramente, vamos observar os dados e utilizar a função `tibble::glimpse()` para ter uma noção geral dos dados.

```
## Visualizar os dados
penguins_raw

## Espiar os dados
dplyr::glimpse(penguins_raw)
```

5.7.3 unite()

Primeiramente, vamos exemplificar como juntar e separar colunas. Vamos utilizar a função `tidyr::unite()` para unir as colunas. Há diversos parâmetros para alterar como esta função funciona, entretanto, é importante destacar três deles: `col` nome da coluna que vai receber as colunas unidas, `sep` indicando o caractere separador das colunas unidas, e `remove` para uma resposta lógica se as colunas unidas são removidas ou não. Vamos unir as colunas "*Region*" e "*Island*" na nova coluna "*region_island*".

```
## Unir colunas
penguins_raw_unir <- tidyr::unite(data = penguins_raw,
```

```
                                  col = "region_island",
                                  Region:Island,
                                  sep = ", ",
                                  remove = FALSE)
head(penguins_raw_unir[, c("Region", "Island", "region_island")])
#> # A tibble: 6 × 3
#>   Region Island    region_island
#>   <chr>  <chr>     <chr>
#> 1 Anvers Torgersen Anvers, Torgersen
#> 2 Anvers Torgersen Anvers, Torgersen
#> 3 Anvers Torgersen Anvers, Torgersen
#> 4 Anvers Torgersen Anvers, Torgersen
#> 5 Anvers Torgersen Anvers, Torgersen
#> 6 Anvers Torgersen Anvers, Torgersen
```

5.7.4 separate()

De forma contrária, podemos utilizar as funções `tidyr::separate()` e `tidyr::separate_rows()` para separar elementos de uma coluna em mais colunas. Respectivamente, a primeira função separa uma coluna em novas colunas conforme a separação, e a segunda função separa uma coluna, distribuindo os elementos nas linhas. Novamente, há diversos parâmetros para mudar o comportamento dessas funções, mas destacaremos aqui quatro deles: `col` coluna a ser separada, `into` os nomes das novas colunas, `sep` indicando o caractere separador das colunas, e `remove` para uma resposta lógica se as colunas separadas são removidas ou não. Vamos separar a coluna "*Stage*" nas colunas "*stage*" e "*egg_stage*".

```
## Separar colunas
penguins_raw_separar <- tidyr::separate(data = penguins_raw,
                                        col = Stage,
                                        into = c("stage", "egg_stage"),
                                        sep = ", ",
                                        remove = FALSE)
head(penguins_raw_separar[, c("Stage", "stage", "egg_stage")])
#> # A tibble: 6 × 3
#>   Stage              stage egg_stage
#>   <chr>              <chr> <chr>
#> 1 Adult, 1 Egg Stage Adult 1 Egg Stage
#> 2 Adult, 1 Egg Stage Adult 1 Egg Stage
#> 3 Adult, 1 Egg Stage Adult 1 Egg Stage
#> 4 Adult, 1 Egg Stage Adult 1 Egg Stage
#> 5 Adult, 1 Egg Stage Adult 1 Egg Stage
#> 6 Adult, 1 Egg Stage Adult 1 Egg Stage

## Separar colunas em novas linhas
penguins_raw_separar_linhas <- tidyr::separate_rows(data = penguins_raw,
                                                    Stage,
                                                    sep = ", ")
```

```
head(penguins_raw_separar_linhas[, c("studyName", "Sample Number",
                                     "Species", "Region", "Island",
                                     "Stage")])
```

5.7.5 drop_na() e replace_na()

Valor faltante (`NA`) é um tipo especial de elemento que são discutidos no Capítulo 4 e são relativamente comuns em conjuntos de dados. Em *R Base*, vimos algumas formas de lidar com esse tipo de elemento. No formato `tidyverse`, existem também várias formas de lidar com eles, mas aqui focaremos nas funções `tidyr::drop_na()` e `tidyr::replace_na()`, para retirar linhas e substituir esses valores, respectivamente.

```
## Remover todas as linhas com NAs
penguins_raw_todas_na <- tidyr::drop_na(data = penguins_raw)
head(penguins_raw_todas_na)

## Remover linhas de colunas específicas com NAs
penguins_raw_colunas_na <- tidyr::drop_na(data = penguins_raw,
                                          any_of("Comments"))
head(penguins_raw_colunas_na[, "Comments"])
#> # A tibble: 6 × 1
#>   Comments
#>   <chr>
#> 1 Not enough blood for isotopes.
#> 2 Adult not sampled.
#> 3 Nest never observed with full clutch.
#> 4 Nest never observed with full clutch.
#> 5 No blood sample obtained.
#> 6 No blood sample obtained for sexing.

## Substituir NAs por outro valor
penguins_raw_subs_na <- tidyr::replace_na(data = penguins_raw,
                                          list(Comments = "Unknown"))
head(penguins_raw_subs_na[, "Comments"])
#> # A tibble: 6 × 1
#>   Comments
#>   <chr>
#> 1 Not enough blood for isotopes.
#> 2 Unknown
#> 3 Unknown
#> 4 Adult not sampled.
#> 5 Unknown
#> 6 Unknown
```

5.7.6 pivot_longer() e pivot_wider()

Por fim, trataremos da pivotagem ou remodelagem de dados. Veremos como mudar o formato do nosso conjunto de dados de longo (*long*) para largo (*wide*) e vice-versa. Primeiramente, vamos ver como partir de um dado longo (*long*) e criar um dado largo (*wide*). Essa é uma operação semelhante à "Tabela Dinâmica" das planilhas eletrônicas. Consiste em usar uma coluna para distribuir seus valores em outras colunas, de modo que os valores dos elementos são preenchidos corretamente, reduzindo assim o número de linhas e aumentando o número de colunas. Essa operação é bastante comum em Ecologia de Comunidades, quando queremos transformar uma lista de espécies em uma matriz de comunidades, com várias espécies nas colunas. Para realizar essa operação, usamos a função `tidyr::pivot_wider()`. Dos diversos parâmetros que podem compor essa função, dois deles são fundamentais: `names_from` que indica a coluna de onde os nomes serão usados e `values_from` que indica a coluna com os valores.

```
## Selecionar colunas
penguins_raw_sel_col <- penguins_raw[, c(2, 3, 13)]
head(penguins_raw_sel_col)
#> # A tibble: 6 × 3
#>   `Sample Number` Species                                 `Body Mass (g)`
#>             <dbl> <chr>                                             <dbl>
#> 1               1 Adelie Penguin (Pygoscelis adeliae)                3750
#> 2               2 Adelie Penguin (Pygoscelis adeliae)                3800
#> 3               3 Adelie Penguin (Pygoscelis adeliae)                3250
#> 4               4 Adelie Penguin (Pygoscelis adeliae)                  NA
#> 5               5 Adelie Penguin (Pygoscelis adeliae)                3450
#> 6               6 Adelie Penguin (Pygoscelis adeliae)                3650

## Pivotar para largo
penguins_raw_pivot_wider <- tidyr::pivot_wider(
                                        data = penguins_raw_sel_col,
                                        names_from = Species,
                                        values_from = `Body Mass (g)`)
head(penguins_raw_pivot_wider)
#> # A tibble: 6 × 4
#>   `Sample Number` `Adelie Penguin (Pygoscelis adeliae)` `Gentoo penguin
(Pygoscelis papua)` `Chinstrap penguin (Pygoscelis…`
#>             <dbl>                                <dbl>
<dbl>                          <dbl>
#> 1               1                                 3750
4500                           3500
#> 2               2                                 3800
5700                           3900
#> 3               3                                 3250
4450                           3650
#> 4               4                                   NA
5700                           3525
#> 5               5                                 3450
5400                           3725
```

```
#> 6               6                                  3650
4550                            3950
```

De modo oposto, podemos partir de um conjunto de dados largo (*wide*), ou seja, com várias colunas, e queremos que essas colunas preencham uma única coluna, e que os valores antes espalhados nessas várias colunas sejam adicionados um embaixo do outro, numa única coluna, no formato longo (*long*). Para essa operação, podemos utilizar a função `tidyr::pivot_longer()`. Novamente, dos diversos parâmetros que podem compor essa função, três deles são fundamentais: `cols` indicando as colunas que serão usadas para serem pivotadas, `names_to` que indica a coluna de onde os nomes serão usados e `values_to` que indica a coluna com os valores.

```
## Selecionar colunas
penguins_raw_sel_col <- penguins_raw[, c(2, 3, 10:13)]
head(penguins_raw_sel_col)

## Pivotar para largo
penguins_raw_pivot_longer <- tidyr::pivot_longer(
                             data = penguins_raw_sel_col,
                             cols = `Culmen Length (mm)`:`Body Mass (g)`,
                             names_to = "medidas", values_to = "valores")
head(penguins_raw_pivot_longer)
```

Para se aprofundar no tema, recomendamos a leitura do Capítulo 12 *Tidy data* de Wickham & Grolemund (2017).

5.8 dplyr

O `dplyr` é um pacote que facilita a manipulação de dados, com uma gramática simples e flexível (por exemplo, como filtragem, reordenamento, seleção, entre outras). Ele foi construído com o intuito de obter uma forma mais rápida e expressiva de manipular dados tabulares. O `tibble` é a versão de data frame mais conveniente para se usar com pacote `dplyr`.

Todas as funções deste pacote são listadas na página de referência do pacote.

5.8.1 Gramática

Sua gramática simples contém funções verbais para manipulação de dados, baseada em:

- Verbos: `mutate()`, `select()`, `filter()`, `arrange()`, `summarise()`, `slice()`, `rename()`, etc.
- Replicação: `across()`, `if_any()`, `if_all()`, `where()`, `starts_with()`, `ends_with()`, `contains()`, etc.
- Agrupamento: `group_by()` e `ungroup()`
- Junções: `inner_join()`, `full_join()`, `left_join()`, `right_join()`, etc.
- Combinações: `bind_rows()` e `bind_cols()`
- Resumos, contagem e seleção: `n()`, `n_distinct()`, `first()`, `last()`, `nth()`, etc.

Existe uma série de funções para realizar a manipulação dos dados, com diversas finalidades: manipulação de uma tabela, manipulação de duas tabelas, replicação, agrupamento, funções de vetores, além de muitas outras funções específicas.

- `relocate()`: muda a ordem das colunas
- `rename()`: muda o nome das colunas
- `select()`: seleciona colunas pelo nome ou posição
- `pull()`: seleciona uma coluna como vetor
- `mutate()`: adiciona novas colunas ou resultados em colunas existentes
- `arrange()`: reordena as linhas com base nos valores de colunas
- `filter()`: seleciona linhas com base em valores de colunas
- `slice()`: seleciona linhas de diferente formas
- `distinct()`: remove linhas com valores repetidos com base nos valores de colunas
- `count()`: conta observações para um grupo com base nos valores de colunas
- `group_by()`: agrupa linhas pelos valores das colunas
- `summarise()`: resume os dados através de funções considerando valores das colunas
- `*_join()`: funções que juntam dados de duas tabelas através de uma coluna chave

5.8.2 Sintaxe

As funções do `dplyr` podem seguir uma mesma sintaxe: o `tibble` será sempre o primeiro argumento dessas funções, seguido de um operador pipe (`%>%`) e pelo nome da função que irá fazer a manipulação nesses dados. Isso permite o encadeamento de várias operações consecutivas mantendo a estrutura do dado original e acrescentando mudanças num encadeamento lógico.

Sendo assim, as funções verbais não precisam modificar necessariamente o *tibble* original, sendo que as operações de manipulações podem e devem ser atribuídas a um novo objeto.

```
## Sintaxe
tb_dplyr <- tb %>%
    funcao_verbal1(argumento1, argumento2, ...) %>%
    funcao_verbal2(argumento1, argumento2, ...) %>%
    funcao_verbal3(argumento1, argumento2, ...)
```

Além de `data.frames` e `tibbles`, a manipulação pelo formato `dplyr` torna o trabalho com outros formatos de classes e dados acessíveis e eficientes como `data.table`, SQL e Apache Spark, para os quais existem pacotes específicos.

- *dtplyr*: manipular conjuntos de dados `data.table`
- *dbplyr*: manipular conjuntos de dados SQL
- *sparklyr*: manipular conjuntos de dados no Apache Spark

5.8.3 palmerpenguins

Para nossos exemplos, vamos utilizar novamente os dados de pinguins *palmerpenguins*. Esses dados estão disponíveis no pacote `palmerpenguins`. Vamos utilizar principalmente o conjunto de dados `penguins`, que é a versão simplificada dos dados brutos `penguins_raw`.

```
## Carregar o pacote palmerpenguins
library(palmerpenguins)
```

5.8.4 relocate()

Primeiramente, vamos reordenar as colunas com a função `dplyr::relocate()`, onde simplesmente listamos as colunas que queremos mudar de posição e para onde elas devem ir. Para esse último passo há dois argumentos: `.before` que indica a coluna onde a coluna realocada deve se mover antes, e o argumento `.after` indicando onde deve se mover depois. Ambos podem ser informados com os nomes ou posições dessas colunas com números.

```
## Reordenar colunas - nome
penguins_relocate_col <- penguins %>%
    dplyr::relocate(sex, year, .after = island)
head(penguins_relocate_col)

## Reordenar colunas - posição
penguins_relocate_ncol <- penguins %>%
    dplyr::relocate(sex, year, .after = 2)
head(penguins_relocate_ncol)
```

5.8.5 rename()

Podemos renomear colunas facilmente com a função `dplyr::rename()`, onde primeiramente informamos o nome que queremos que a coluna tenha, seguido do operador `=` e a coluna do nosso dado ("nova_coluna = antiga_coluna"). Também podemos utilizar a função `dplyr::rename_with()`, que faz a mudança do nome em múltiplas colunas, que pode depender ou não de resultados booleanos.

```
## Renomear as colunas
penguins_rename <- penguins %>%
    dplyr::rename(bill_length = bill_length_mm,
                  bill_depth = bill_depth_mm,
                  flipper_length = flipper_length_mm,
                  body_mass = body_mass_g)
head(penguins_rename)
#> # A tibble: 6 × 8
#>   species island    bill_length bill_depth flipper_length body_mass sex     year
#>   <fct>   <fct>           <dbl>      <dbl>          <int>     <int> <fct>  <int>
#> 1 Adelie  Torgersen        39.1       18.7            181      3750 male    2007
#> 2 Adelie  Torgersen        39.5       17.4            186      3800 female  2007
#> 3 Adelie  Torgersen        40.3       18              195      3250 female  2007
#> 4 Adelie  Torgersen        NA         NA               NA        NA <NA>    2007
#> 5 Adelie  Torgersen        36.7       19.3            193      3450 female  2007
#> 6 Adelie  Torgersen        39.3       20.6            190      3650 male    2007

## mudar o nome de todas as colunas
penguins_rename_with <- penguins %>%
```

```
    dplyr::rename_with(toupper)
head(penguins_rename_with)
```

5.8.6 select()

Outra operação bastante usual dentro da manipulação de dados tabulares é a seleção de colunas. Podemos fazer essa operação com a função `dplyr::select()`, que seleciona colunas pelo nome ou pela sua posição. Aqui há uma série de possibilidades de seleção de colunas, desde utilizar operadores como `:` para selecionar intervalos de colunas, `!` para tomar o complemento (todas menos as listadas), além de funções como `dplyr::starts_with()`, `dplyr::ends_with()`, `dplyr::contains()` para procurar colunas com um padrão de texto do nome da coluna.

```
## Selecionar colunas por posição
penguins_select_position <- penguins %>%
    dplyr::select(3:6)
head(penguins_select_position)
#> # A tibble: 6 × 4
#>   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
#>            <dbl>         <dbl>             <int>       <int>
#> 1           39.1          18.7               181        3750
#> 2           39.5          17.4               186        3800
#> 3           40.3          18                 195        3250
#> 4           NA            NA                  NA          NA
#> 5           36.7          19.3               193        3450
#> 6           39.3          20.6               190        3650

## Selecionar colunas por nomes
penguins_select_names <- penguins %>%
    dplyr::select(bill_length_mm:body_mass_g)
head(penguins_select_names)
#> # A tibble: 6 × 4
#>   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
#>            <dbl>         <dbl>             <int>       <int>
#> 1           39.1          18.7               181        3750
#> 2           39.5          17.4               186        3800
#> 3           40.3          18                 195        3250
#> 4           NA            NA                  NA          NA
#> 5           36.7          19.3               193        3450
#> 6           39.3          20.6               190        3650

## Selecionar colunas por padrão
penguins_select_contains <- penguins %>%
    dplyr::select(contains("_mm"))
head(penguins_select_contains)
#> # A tibble: 6 × 3
#>   bill_length_mm bill_depth_mm flipper_length_mm
#>            <dbl>         <dbl>             <int>
```

```
#> 1           39.1          18.7                181
#> 2           39.5          17.4                186
#> 3           40.3          18                  195
#> 4           NA            NA                   NA
#> 5           36.7          19.3                193
#> 6           39.3          20.6                190
```

5.8.7 pull()

Quando usamos a função `dplyr::select()`, mesmo que para apenas uma coluna, o retorno da função é sempre um `tibble`. Caso precisemos que essa coluna se torne um vetor dentro do encadeamento dos `pipes`, usamos a função `dplyr::pull()`, que extrai uma única coluna como vetor.

```
## Coluna como vetor
penguins_select_pull <- penguins %>%
    dplyr::pull(bill_length_mm)
head(penguins_select_pull, 15)
#>  [1] 39.1 39.5 40.3   NA 36.7 39.3 38.9 39.2 34.1 42.0 37.8 37.8 41.1 38.6 34.6
```

5.8.8 mutate()

Uma das operações mais úteis dentre as operações para colunas é adicionar ou atualizar os valores de colunas. Para essa operação, usaremos a função `dplyr::mutate()`. Podemos ainda usar os argumentos `.before` e `.after` para indicar onde a nova coluna deve ficar, além do parâmetro `.keep` com diversas possibilidades de manter colunas depois de usar a função `dplyr::mutate()`. Por fim, é fundamental destacar o uso das funções de replicação: `dplyr::across()`, `dplyr::if_any()` e `dplyr::if_all()`, para os quais a função fará alterações em múltiplas colunas de uma vez, dependendo de resultados booleanos.

```
## Adicionar colunas
penguins_mutate <- penguins %>%
    dplyr::mutate(body_mass_kg = body_mass_g/1e3, .before = sex)
head(penguins_mutate)

## Modificar várias colunas
penguins_mutate_across <- penguins %>%
    dplyr::mutate(across(where(is.factor), as.character))
head(penguins_mutate_across)
```

5.8.9 arrange()

Além de operações em colunas, podemos fazer operações em linhas. Vamos começar com a reordenação das linhas com base nos valores das colunas. Para essa operação, usamos a função `dplyr::arrange()`. Podemos reordenar por uma ou mais colunas de forma crescente ou decrescente usando a função `desc()` ou o operador `-` antes da coluna de interesse. Da mesma forma que na função `dplyr::mutate()`, podemos usar as funções de replicação para ordenar as linhas para várias colunas de uma vez, dependendo de resultados booleanos.

```
## Reordenar linhas - crescente
penguins_arrange <- penguins %>%
    dplyr::arrange(body_mass_g)
head(penguins_arrange)

## Reordenar linhas - decrescente
penguins_arrange_desc <- penguins %>%
    dplyr::arrange(desc(body_mass_g))
head(penguins_arrange_desc)

## Reordenar linhas - decrescente
penguins_arrange_desc_m <- penguins %>%
    dplyr::arrange(-body_mass_g)
head(penguins_arrange_desc_m)

## Reordenar linhas - multiplas colunas
penguins_arrange_across <- penguins %>%
    dplyr::arrange(across(where(is.numeric)))
head(penguins_arrange_across)
```

5.8.10 filter()

Uma das principais e mais usuais operações que podemos realizar em linhas é a seleção de linhas através do filtro por valores de uma ou mais colunas, utilizando a função `dplyr::filter()`. Para realizar os filtros utilizaremos grande parte dos operadores relacionais e lógicos que listamos na Tabela 4.1 no Capítulo 4, especialmente os lógicos para combinações de filtros em mais de uma coluna. Além desses operadores, podemos utilizar a função `is.na()` para filtros em elementos faltantes, e as funções `dplyr::between()` e `dplyr::near()` para filtros entre valores, e para valores próximos com certa tolerância, respectivamente. Por fim, podemos usar as funções de replicação para filtro das linhas para mais de uma coluna, dependendo de resultados booleanos.

```
## Filtrar linhas
penguins_filter <- penguins %>%
    dplyr::filter(species == "Adelie")
head(penguins_filter)

## Filtrar linhas
penguins_filter_two <- penguins %>%
    dplyr::filter(species == "Adelie" & sex == "female")
head(penguins_filter_two)

## Filtrar linhas
penguins_filter_in <- penguins %>%
    dplyr::filter(species %in% c("Adelie", "Gentoo"),
                  sex == "female")
head(penguins_filter_in)
```

```
## Filtrar linhas - NA
penguins_filter_na <- penguins %>%
    dplyr::filter(!is.na(sex) == TRUE)
head(penguins_filter_na)

## Filtrar linhas - intervalos
penguins_filter_between <- penguins %>%
    dplyr::filter(between(body_mass_g, 3000, 4000))
head(penguins_filter_between)

## Filtrar linhas por várias colunas
penguins_filter_if <- penguins %>%
    dplyr::filter(if_all(where(is.integer), ~ . > 200))
head(penguins_filter_if)
```

5.8.11 slice()

Além da seleção de linhas por filtros, podemos fazer a seleção das linhas por intervalos, indicando quais linhas desejamos, usando a função `dplyr::slice()`, e informando o argumento `n` para o número da linha ou intervalo das linhas. Essa função possui variações no sufixo muito interessantes: `dplyr::slice_head()` e `dplyr::slice_tail()` seleciona as primeiras e últimas linhas, `dplyr::slice_min()` e `dplyr::slice_max()` seleciona linhas com os maiores e menores valores de uma coluna, e `dplyr::slice_sample()` seleciona linhas aleatoriamente.

```
## Seleciona linhas
penguins_slice <- penguins %>%
    dplyr::slice(n = c(1, 3, 300:n()))
head(penguins_slice)

## Seleciona linhas - head
penguins_slice_head <- penguins %>%
    dplyr::slice_head(n = 5)
head(penguins_slice_head)

## Seleciona linhas - max
penguins_slice_max <- penguins %>%
    dplyr::slice_max(body_mass_g, n = 5)
head(penguins_slice_max)

## Seleciona linhas - sample
penguins_slice_sample <- penguins %>%
    dplyr::slice_sample(n = 30)
head(penguins_slice_sample)
```

5.8.12 distinct()

A última operação que apresentaremos para linhas é a retirada de linhas com valores repetidos com base nos valores de uma ou mais colunas, utilizando a função `dplyr::distinct()`. Essa função por padrão retorna apenas a(s) coluna(s) utilizada(s) para retirar as linhas com valores repetidos, sendo necessário acrescentar o argumento `.keep_all = TRUE` para retornar todas as colunas. Por fim, podemos usar as funções de replicação para retirar linhas com valores repetidos para mais de uma coluna, dependendo de resultados booleanos.

```
## Retirar linhas com valores repetidos
penguins_distinct <- penguins %>%
    dplyr::distinct(body_mass_g)
head(penguins_distinct)
#> # A tibble: 6 × 1
#>   body_mass_g
#>         <int>
#> 1        3750
#> 2        3800
#> 3        3250
#> 4          NA
#> 5        3450
#> 6        3650

## Retirar linhas com valores repetidos - manter as outras colunas
penguins_distinct_keep_all <- penguins %>%
    dplyr::distinct(body_mass_g, .keep_all = TRUE)
head(penguins_distinct_keep_all)

## Retirar linhas com valores repetidos para várias colunas
penguins_distinct_keep_all_across <- penguins %>%
    dplyr::distinct(across(where(is.integer)), .keep_all = TRUE)
head(penguins_distinct_keep_all_across)
```

5.8.13 count()

Agora entraremos no assunto de resumo das observações. Podemos fazer contagens resumos dos nossos dados, utilizando para isso a função `dplyr::count()`. Essa função contará valores de uma ou mais colunas, geralmente para variáveis categóricas, semelhante à função *R Base* `table()`, mas num contexto *tidyverse*.

```
## Contagens de valores para uma coluna
penguins_count <- penguins %>%
    dplyr::count(species)
penguins_count
#> # A tibble: 3 × 2
#>   species       n
#>   <fct>     <int>
```

```
#> 1 Adelie      152
#> 2 Chinstrap    68
#> 3 Gentoo      124

## Contagens de valores para mais de uma coluna
penguins_count_two <- penguins %>%
    dplyr::count(species, island)
penguins_count_two
#> # A tibble: 5 × 3
#>   species   island        n
#>   <fct>     <fct>     <int>
#> 1 Adelie    Biscoe       44
#> 2 Adelie    Dream        56
#> 3 Adelie    Torgersen    52
#> 4 Chinstrap Dream        68
#> 5 Gentoo    Biscoe      124
```

5.8.14 group_by()

Uma grande parte das operações feitas nos dados são realizadas em grupos definidos por valores de colunas com dados categóricas. A função `dplyr::group_by()` transforma um `tibble` em um `tibble grouped`, onde as operações são realizadas "por grupo." Essa função é utilizada geralmente junto com a função `dplyr::summarise()`, que veremos logo em seguida. O agrupamento não altera a aparência dos dados (além de informar como estão agrupados). A função `dplyr::ungroup()` remove o agrupamento. Podemos ainda usar funções de replicação para fazer os agrupamentos para mais de uma coluna, dependendo de resultados booleanos.

```
## Agrupamento
penguins_group_by <- penguins %>%
    dplyr::group_by(species)
head(penguins_group_by)

## Agrupamento de várias colunas
penguins_group_by_across <- penguins %>%
    dplyr::group_by(across(where(is.factor)))
head(penguins_group_by_across)
```

5.8.15 summarise()

Como dissemos, muitas vezes queremos resumir nossos dados, principalmente para ter uma noção geral das variáveis (colunas) ou mesmo começar a análise exploratória resumindo variáveis contínuas por grupos de variáveis categóricas. Dessa forma, ao utilizar a função `dplyr::summarise()` teremos um novo `tibble` com os dados resumidos, que é a agregação ou resumo dos dados através de funções. Da mesma forma que outras funções, podemos usar funções de replicação para resumir valores para mais de uma coluna, dependendo de resultados booleanos.

```
## Resumo
penguins_summarise <- penguins %>%
    dplyr::group_by(species) %>%
    dplyr::summarize(body_mass_g_mean = mean(body_mass_g, na.rm = TRUE),
                     body_mass_g_sd = sd(body_mass_g, na.rm = TRUE))
penguins_summarise
#> # A tibble: 3 × 3
#>   species   body_mass_g_mean body_mass_g_sd
#>   <fct>                <dbl>          <dbl>
#> 1 Adelie               3701.           459.
#> 2 Chinstrap            3733.           384.
#> 3 Gentoo               5076.           504.

## Resumo para várias colunas
penguins_summarise_across <- penguins %>%
    dplyr::group_by(species) %>%
    dplyr::summarize(across(where(is.numeric), ~ mean(.x, na.rm = TRUE)))
penguins_summarise_across
#> # A tibble: 3 × 6
#>   species   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g  year
#>   <fct>              <dbl>         <dbl>             <dbl>       <dbl> <dbl>
#> 1 Adelie              38.8          18.3              190.       3701. 2008.
#> 2 Chinstrap           48.8          18.4              196.       3733. 2008.
#> 3 Gentoo              47.5          15.0              217.       5076. 2008.
```

5.8.16 bind_rows() e bind_cols()

Muitas vezes teremos de combinar duas ou mais tabelas de dados. Podemos utilizar as funções R Base `rbind()` e `cbind()`, como vimos no Capítulo 4. Entretanto, pode ser interessante avançar para as funções `dplyr::bind_rows()` e `dplyr::bind_cols()` do formato *tidyverse*. A ideia é muito semelhante: a primeira função combina dados por linhas e a segunda por colunas. Entretanto, há algumas vantagens no uso dessas funções, como a identificação das linhas pelo argumento `.id` para a primeira função, e a conferência do nome das colunas pelo argumento `.name_repair` para a segunda função.

```
## Selecionar as linhas para dois tibbles
penguins_01 <- dplyr::slice(penguins, 1:5)
penguins_02 <- dplyr::slice(penguins, 51:55)

## Combinar as linhas
penguins_bind_rows <- dplyr::bind_rows(penguins_01, penguins_02,
                                       .id = "id")
head(penguins_bind_rows)

## Combinar as colunas
penguins_bind_cols <- dplyr::bind_cols(penguins_01, penguins_02,
```

```
                                    .name_repair = "unique")
head(penguins_bind_cols)
```

5.8.17 *_join()

Finalmente, veremos o último conjunto de funções do pacote `dplyr`, a junção de tabelas. Nessa operação, fazemos a combinação de pares de conjunto de dados tabulares por uma ou mais colunas chaves. Há dois tipos de junções: junção de modificação e junção de filtragem. A junção de modificação primeiro combina as observações por suas chaves e, em seguida, copia as variáveis (colunas) de uma tabela para a outra. É fundamental destacar a importância da coluna chave, que é indicada pelo argumento `by`. Essa coluna deve conter elementos que sejam comuns às duas tabelas para que haja a combinação dos elementos.

Existem quatro tipos de junções de modificações, que são realizadas pelas funções: `dplyr::inner_join()`, `dplyr::left_join()`, `dplyr::full_join()` e `dplyr::right_join()`, e que podem ser representadas na Figura 5.3.

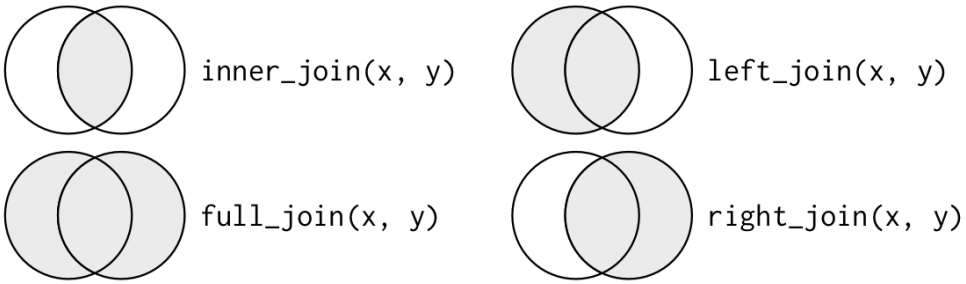


Figura 5.3: Diferentes tipos de joins, representados com um diagrama de Venn. Adaptado de Wickham & Grolemund (2017).

Considerando a nomenclatura de duas tabelas de dados por `x` e `y`, temos:

- `inner_join(x, y)`: mantém apenas as observações em `x` e em `y`
- `left_join(x, y)`: mantém todas as observações em `x`
- `right_join(x, y)`: mantém todas as observações em `y`
- `full_join(x, y)`: mantém todas as observações em `x` e em `y`

Aqui, vamos demostrar apenas a função `dplyr::left_join()`, combinando um `tibble` de coordenadas geográficas das ilhas com o conjunto de dados do penguins.

```
## Adicionar uma coluna chave de ids
penguin_islands <- tibble(
    island = c("Torgersen", "Biscoe", "Dream", "Alpha"),
    longitude = c(-64.083333, -63.775636, -64.233333, -63),
    latitude = c(-64.766667, -64.818569, -64.733333, -64.316667))

## Junção - left
penguins_left_join <- dplyr::left_join(penguins, penguin_islands,
                                       by = "island")
head(penguins_left_join)
```

Já o segundo tipo de junção, a junção de filtragem combina as observações da mesma maneira que as junções de modificação, mas afetam as observações (linhas), não as variáveis (colunas). Existem dois tipos.

- `semi_join(x, y)`: mantém todas as observações em `x` que têm uma correspondência em `y`
- `anti_join(x, y)`: elimina todas as observações em `x` que têm uma correspondência em `y`

De forma geral, *semi-joins* são úteis para corresponder tabelas de resumo filtradas de volta às linhas originais, removendo as linhas que não estavam antes do join. Já *anti-joins* são úteis para diagnosticar incompatibilidades de junção, por exemplo, ao verificar os elementos que não combinam entre duas tabelas de dados.

5.8.18 Operações de conjuntos e comparação de dados

Temos ainda operações de conjuntos e comparação de dados.

- `union(x, y)`: retorna todas as linhas que aparecem em `x`, `y` ou mais dos conjuntos de dados
- `interesect(x, y)`: retorna apenas as linhas que aparecem em `x` e em `y`
- `setdiff(x, y)`: retorna as linhas que aparecem `x`, mas não em `y`
- `setequal(x, y)`: retorna se `x` e `y` são iguais e quais suas diferenças

Para se aprofundar no tema, recomendamos a leitura do Capítulo 13 *Relational data* de Wickham & Grolemund (2017).

5.9 stringr

O pacote `stringr` fornece um conjunto de funções para a manipulação de caracteres ou *strings*. O pacote concentra-se nas funções de manipulação mais importantes e comumente usadas. Para funções mais específicas, recomenda-se usar o pacote `stringi`, que fornece um conjunto mais abrangente de funções. As funções do `stringr` podem ser agrupadas em algumas operações para tarefas específicas como: i) correspondência de padrões, ii) retirar e acrescentar espaços em branco, iii) mudar maiúsculas e minúsculas, além de muitas outras operações com caracteres.

Todas as funções deste pacote são listadas na página de referência do pacote.

Demonstraremos algumas funções para algumas operações mais comuns, utilizando um vetor de um elemento, com o *string* "penguins".

Podemos explorar o comprimento de *strings* com a função `stringr::str_length()`.

```
## Comprimento
stringr::str_length(string = "penguins")
#> [1] 8
```

Extrair um *string* por sua posição usando a função `stringr::str_sub()` ou por um padrão com `stringr::str_extract()`.

```
## Extrair pela posição
stringr::str_sub(string = "penguins", end = 3)
#> [1] "pen"

## Extrair por padrão
stringr::str_extract(string = "penguins", pattern = "p")
#> [1] "p"
```

Substituir *strings* por outros *strings* com `stringr::str_replace()`.

```
## Substituir
stringr::str_replace(string = "penguins", pattern = "i",
                     replacement = "y")
#> [1] "penguyns"
```

Separar *strings* por um padrão com a função `stringr::str_split()`.

```
## Separar
stringr::str_split(string = "p-e-n-g-u-i-n-s", pattern = "-",
                   simplify = TRUE)
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
#> [1,] "p"  "e"  "n"  "g"  "u"  "i"  "n"  "s"
```

Inserir espaços em brancos pela esquerda, direita ou ambos com a função `stringr::str_pad()`.

```
## Inserir espacos em branco
stringr::str_pad(string = "penguins", width = 10, side = "left")
#> [1] "  penguins"
stringr::str_pad(string = "penguins", width = 10, side = "right")
#> [1] "penguins  "
stringr::str_pad(string = "penguins", width = 10, side = "both")
#> [1] " penguins "
```

Também podemos remover espaços em branco da esquerda, direita ou ambos, utilizando `stringr::str_trim()`.

```
## Remover espacos em branco
stringr::str_trim(string = " penguins ", side = "left")
#> [1] "penguins "
stringr::str_trim(string = " penguins ", side = "right")
#> [1] " penguins"
stringr::str_trim(string = " penguins ", side = "both")
#> [1] "penguins"
```

Podemos também alterar minúsculas e maiúsculas em diferentes posições do *string*, com várias funções.

```
## Alterar minúsculas e maiúsculas
stringr::str_to_lower(string = "Penguins")
#> [1] "penguins"
stringr::str_to_upper(string = "penguins")
#> [1] "PENGUINS"
```

```
stringr::str_to_sentence(string = "penGuins")
#> [1] "Penguins"
stringr::str_to_title(string = "penGuins")
#> [1] "Penguins"
```

Podemos ainda ordenar os elementos de um vetor por ordem alfabética de forma crescente ou decrescente, usando `stringr::str_sort()`.

```
## Ordenar
stringr::str_sort(x = letters)
#>  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
"r" "s" "t" "u" "v" "w" "x" "y" "z"
stringr::str_sort(x = letters, dec = TRUE)
#>  [1] "z" "y" "x" "w" "v" "u" "t" "s" "r" "q" "p" "o" "n" "m" "l" "k" "j"
"i" "h" "g" "f" "e" "d" "c" "b" "a"
```

Podemos ainda utilizar essas funções em complemento com o pacote `dplyr`, para alterar os *strings* de colunas ou nome das colunas.

```
## Alterar valores das colunas
penguins_stringr_valores <- penguins %>%
    dplyr::mutate(species = stringr::str_to_lower(species))

## Alterar nome das colunas
penguins_stringr_nomes <- penguins %>%
    dplyr::rename_with(stringr::str_to_title)
```

Para se aprofundar no tema, recomendamos a leitura do Capítulo 14 *Strings* de Wickham & Grolemund (2017).

5.10 forcats

O pacote `forcats` fornece um conjunto de ferramentas úteis para facilitar a manipulação de fatores. Como dito no Capítulo 4, usamos fatores geralmente quando temos dados categóricos, que são variáveis que possuem um conjunto de valores fixos e conhecidos. As funções são utilizadas principalmente para: i) mudar a ordem dos níveis, ii) mudar os valores dos níveis, iii) adicionar e remover níveis, iv) combinar múltiplos níveis, além de outras operações.

Todas as funções deste pacote são listadas na página de referência do pacote.

Vamos utilizar ainda os dados `penguins` e `penguins_raw` para exemplificar o uso do pacote `forcats`.

```
## Carregar o pacote palmerpenguins
library(palmerpenguins)
```

Primeiramente, vamos converter dados de *string* para fator, utilizando a função `forcats::as_factor()`.

```
## String
forcats::as_factor(penguins_raw$Species) %>% head()
```

Podemos facilmente mudar o nome dos níveis utilizando a função `forcats::fct_recode()`.

```
## Mudar o nome dos níveis
forcats::fct_recode(penguins$species, a = "Adelie", c = "Chinstrap",
                    g = "Gentoo") %>% head()

#> [1] a a a a a a
#> Levels: a c g
```

Para inverter os níveis, usamos a função `forcats::fct_rev()`.

```
## Inverter os níveis
forcats::fct_rev(penguins$species) %>% head()
#> [1] Adelie Adelie Adelie Adelie Adelie Adelie
#> Levels: Gentoo Chinstrap Adelie
```

Uma operação muito comum com fatores é mudar a ordem dos níveis. Quando precisamos especificar a ordem dos níveis, podemos fazer essa operação manualmente com a função `forcats::fct_relevel()`.

```
## Especificar a ordem dos níveis
forcats::fct_relevel(penguins$species, "Chinstrap", "Gentoo",
                     "Adelie") %>% head()
#> [1] Adelie Adelie Adelie Adelie Adelie Adelie
#> Levels: Chinstrap Gentoo Adelie
```

Como vimos, a reordenação dos níveis pode ser feita manualmente. Mas existem outras formas automáticas de reordenação seguindo algumas regras, para as quais existem funções específicas.

- `forcats::fct_inorder()`: pela ordem em que aparecem pela primeira vez
- `forcats::fct_infreq()`: por número de observações com cada nível (decrescente, i.e., o maior primeiro)
- `forcats::fct_inseq()`: pelo valor numérico do nível

```
## Níveis pela ordem em que aparecem
forcats::fct_inorder(penguins$species) %>% head()
#> [1] Adelie Adelie Adelie Adelie Adelie Adelie
#> Levels: Adelie Gentoo Chinstrap

## Ordem (decrescente) de frequência
forcats::fct_infreq(penguins$species) %>% head()
#> [1] Adelie Adelie Adelie Adelie Adelie Adelie
#> Levels: Adelie Gentoo Chinstrap
```

Por fim, podemos fazer a agregação de níveis raros em um nível utilizando a função `forcats::fct_lump()`.

```
## Agregação de níveis raros em um nível
forcats::fct_lump(penguins$species) %>% head()
```

```
#> [1] Adelie Adelie Adelie Adelie Adelie Adelie
#> Levels: Adelie Gentoo Other
```

Podemos ainda utilizar essas funções em complemento com o pacote `dplyr` para fazer manipulações de fatores nas colunas de `tibbles`.

```
## Transformar várias colunas em fator
penguins_raw_multi_factor <- penguins_raw %>%
    dplyr::mutate(across(where(is.character), forcats::as_factor))
```

Para se aprofundar no tema, recomendamos a leitura do Capítulo 15 *Factors* de Wickham & Grolemund (2017).

5.11 lubridate

O pacote `lubridate` fornece um conjunto de funções para a manipulação de dados de data e horário. Dessa forma, esse pacote facilita a manipulação dessa classe de dado no R, pois geralmente esses dados não são intuitivos e mudam dependendo do tipo de objeto de data e horário. Além disso, os métodos que usam datas e horários devem levar em consideração fusos horários, anos bissextos, horários de verão, além de outras particularidades. Existem diversas funções nesse pacote, sendo as mesmas focadas em: i) transformações de data/horário, ii) componentes, iii) arredondamentos, iv) durações, v) períodos, vi) intervalos, além de muitas outras funções específicas.

Todas as funções deste pacote são listadas na página de referência do pacote.

Apesar de estar inserido no escopo do *tidyverse*, este pacote não é carregado com os demais, requisitando seu carregamento solo.

```
## Carregar
library(lubridate)
```

Existem três tipos de dados data/horário:

- **Data**: tempo em dias, meses e anos `<date>`
- **Horário**: tempo dentro de um dia `<time>`
- **Data-horário**: tempo em um instante (data mais tempo) `<dttm>`

Para trabalhar exclusivamente com horários, podemos utilizar o pacote `hms`.

É fundamental também destacar que algumas letras terão um significado temporal, sendo abreviações de diferentes períodos em inglês: **y**ear (ano), **m**onth (mês), **w**eak (semana), **d**ay (dia), **h**our (hora), **m**inute (minuto), e **s**econd (segundo).

Para acessar a informação da data e horários atuais podemos utilizar as funções `lubridate::today()` e `lubridate::now()`.

```
## Extrair a data nesse instante
lubridate::today()
#> [1] "2022-02-22"

## Extrair a data e tempo nesse instante
```

```
lubridate::now()
#> [1] "2022-02-22 23:44:45 -03"
```

Além dessas informações instantâneas, existem três maneiras de criar um dado de data/horário.

- De um *string*
- De componentes individuais de data e horário
- De um objeto de data/horário existente

Os dados de data/horário geralmente estão no formato de *strings*. Podemos transformar os dados especificando a ordem dos seus componentes, ou seja, a ordem em que ano, mês e dia aparecem no *string*, usando as letras `y` (ano), `m` (mês) e `d` (dia) na mesma ordem, por exemplo, `lubridate::dmy()`.

```
## Strings e números para datas
lubridate::dmy("03-03-2021")
#> [1] "2021-03-03"
```

Essas funções também aceitam números sem aspas, além de serem muito versáteis e funcionarem em outros diversos formatos.

```
## Strings e números para datas
lubridate::dmy("03-Mar-2021")
lubridate::dmy(03032021)
lubridate::dmy("03032021")
lubridate::dmy("03/03/2021")
lubridate::dmy("03.03.2021")
```

Além da data, podemos especificar horários atrelados a essas datas. Para criar uma data com horário adicionamos um underscore (`_`) e `h` (hora), `m` (minuto) e `s` (segundo) ao nome da função, além do argumento `tz` para especificar o fuso horário (tema tratado mais adiante nessa seção).

```
## Especificar horários e fuso horário
lubridate::dmy_h("03-03-2021 13")
#> [1] "2021-03-03 13:00:00 UTC"
lubridate::dmy_hm("03-03-2021 13:32")
#> [1] "2021-03-03 13:32:00 UTC"
lubridate::dmy_hms("03-03-2021 13:32:01")
#> [1] "2021-03-03 13:32:01 UTC"
lubridate::dmy_hms("03-03-2021 13:32:01", tz = "America/Sao_Paulo")
#> [1] "2021-03-03 13:32:01 -03"
```

Podemos ainda ter componentes individuais de data/horário em múltiplas colunas. Para realizar essa transformação, podemos usar as funções `lubridate::make_date()` e `lubridate::make_datetime()`.

```
## Dados com componentes individuais
dados <- tibble::tibble(
    ano = c(2021, 2021, 2021),
    mes = c(1, 2, 3),
    dia = c(12, 20, 31),
```

```
    hora = c(2, 14, 18),
    minuto = c(2, 44, 55))

## Data de componentes individuais
dados %>%
    dplyr::mutate(data = lubridate::make_datetime(ano, mes, dia,
                                                  hora, minuto))
#> # A tibble: 3 × 6
#>     ano   mes   dia  hora minuto data
#>   <dbl> <dbl> <dbl> <dbl>  <dbl> <dttm>
#> 1  2021     1    12     2      2 2021-01-12 02:02:00
#> 2  2021     2    20    14     44 2021-02-20 14:44:00
#> 3  2021     3    31    18     55 2021-03-31 18:55:00
```

Por fim, podemos criar datas modificando entre data/horário e data, utilizando as funções `lubridate::as_datetime()` e `lubridate::as_date()`.

```
## Data para data-horário
lubridate::as_datetime(today())
#> [1] "2022-02-22 UTC"

## Data-horário para data
lubridate::as_date(now())
#> [1] "2022-02-22"
```

Uma vez que entendemos como podemos criar dados de data/horário, podemos explorar funções para acessar e definir componentes individuais. Para essa tarefa existe uma grande quantidade de funções para acessar partes específicas de datas e horários.

- `year()`: acessa o ano
- `month()`: acessa o mês
- `month()`: acessa o dia
- `yday()`: acessa o dia do ano
- `mday()`: acessa o dia do mês
- `wday()`: acessa o dia da semana
- `hour()`: acessa as horas
- `minute()`: acessa os minutos
- `second()`: acessa os segundos

```
## Extrair
lubridate::year(now())
#> [1] 2022
lubridate::month(now())
#> [1] 2
lubridate::month(now(), label = TRUE)
#> [1] Feb
#> Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < Oct < Nov < Dec
lubridate::day(now())
```

```
#> [1] 22
lubridate::wday(now())
#> [1] 3
lubridate::wday(now(), label = TRUE)
#> [1] Tue
#> Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
lubridate::second(now())
#> [1] 45.69728
```

Além de acessar componentes de datas e horários, podemos usar essas funções para fazer a inclusão de informações de datas e horários.

```
## Data
data <- dmy_hms("04-03-2021 01:04:56")

## Incluir
lubridate::year(data) <- 2020
lubridate::month(data) <- 01
lubridate::hour(data) <- 13
```

Mais convenientemente, podemos utilizar a função `update()` para alterar vários valores de uma vez.

```
## Incluir vários valores
update(data, year = 2020, month = 1, mday = 1, hour = 1)
#> [1] "2020-01-01 01:04:56 UTC"
```

Muitas vezes precisamos fazer operações com datas, como: adição, subtração, multiplicação e divisão. Para tanto, é preciso entender três classes importantes que representam intervalos de tempo.

- **Durações**: representam um número exato de segundos
- **Períodos**: representam unidades humanas como semanas e meses
- **Intervalos**: representam um ponto inicial e final

Quando fazemos uma subtração de datas, criamos um objeto da classe `difftime`. Essa classe pode ser um pouco complicada de trabalhar, então dentro do *lubridate* usamos funções que convertem essa classe em duração, da classe `Duration`. As durações sempre registram o intervalo de tempo em segundos, com alguma unidade de tempo maior entre parênteses. Há uma série de funções para tratar dessa classe.

- `duration()`: cria data em duração
- `as.duration()`: converte datas em duração
- `dyears()`: duração de anos
- `dmonths()`: duração de meses
- `dweeks()`: duração de semanas
- `ddays()`: duração de dias
- `dhours()`: duração de horas
- `dminutes()`: duração de minutos
- `dseconds()`: duração de segundos

```
## Subtração de datas
tempo_estudando_r <- lubridate::today() - lubridate::dmy("30-11-2011")

## Conversão para duração
tempo_estudando_r_dur <- lubridate::as.duration(tempo_estudando_r)

## Criando durações
lubridate::duration(90, "seconds")
#> [1] "90s (~1.5 minutes)"
lubridate::duration(1.5, "minutes")
#> [1] "90s (~1.5 minutes)"
lubridate::duration(1, "days")
#> [1] "86400s (~1 days)"

## Transformação da duração
lubridate::dseconds(100)
#> [1] "100s (~1.67 minutes)"
lubridate::dminutes(100)
#> [1] "6000s (~1.67 hours)"
lubridate::dhours(100)
#> [1] "360000s (~4.17 days)"
lubridate::ddays(100)
#> [1] "8640000s (~14.29 weeks)"
lubridate::dweeks(100)
#> [1] "60480000s (~1.92 years)"
lubridate::dyears(100)
#> [1] "3155760000s (~100 years)"
```

Podemos ainda utilizar as durações para fazer operações aritméticas com datas como adição, subtração e multiplicação.

```
## Somando durações a datas
lubridate::today() + lubridate::ddays(1)
#> [1] "2022-02-23"

## Subtraindo durações de datas
lubridate::today() - lubridate::dyears(1)
#> [1] "2021-02-21 18:00:00 UTC"

## Multiplicando durações
2 * dyears(2)
#> [1] "126230400s (~4 years)"
```

Além das durações, podemos usar períodos, que são extensões de tempo não fixados em segundos como as durações, mas flexíveis, com o tempo em dias, semanas, meses ou anos, permitindo uma interpretação mais intuitiva das datas. Novamente, há uma série de funções para realizar essas operações.

- `period()`: cria data em período
- `as.period()`: converte datas em período

- `seconds()`: período em segundos
- `minutes()`: período em minutos
- `hours()`: período em horas
- `days()`: período em dias
- `weeks()`: período em semanas
- `months()`: período em meses
- `years()`: período em anos

```
## Criando períodos
period(c(90, 5), c("second", "minute"))
#> [1] "5M 90S"
period(c(3, 1, 2, 13, 1), c("second", "minute", "hour", "day", "week"))
#> [1] "20d 2H 1M 3S"

## Transformação de períodos
lubridate::seconds(100)
#> [1] "100S"
lubridate::minutes(100)
#> [1] "100M 0S"
lubridate::hours(100)
#> [1] "100H 0M 0S"
lubridate::days(100)
#> [1] "100d 0H 0M 0S"
lubridate::weeks(100)
#> [1] "700d 0H 0M 0S"
lubridate::years(100)
#> [1] "100y 0m 0d 0H 0M 0S"
```

Além disso, podemos fazer operações com os períodos, somando e subtraindo.

```
## Somando datas
lubridate::today() + lubridate::weeks(10)
#> [1] "2022-05-03"

## Subtraindo datas
lubridate::today() - lubridate::weeks(10)
#> [1] "2021-12-14"

## Criando datas recorrentes
lubridate::today() + lubridate::weeks(0:10)
#>  [1] "2022-02-22" "2022-03-01" "2022-03-08" "2022-03-15" "2022-03-22"
"2022-03-29" "2022-04-05" "2022-04-12" "2022-04-19"
#> [10] "2022-04-26" "2022-05-03"
```

Por fim, intervalos são períodos de tempo limitados por duas datas, possuindo uma duração com um ponto de partida, que o faz preciso para determinar uma duração. Intervalos são objetos da classe `Interval`. Da mesma forma que para duração e períodos, há uma série de funções para realizar essas operações.

- `interval()`: cria data em intervalo
- `%--%`: cria data em intervalo
- `as.interval()`: converte datas em intervalo
- `int_start()`: acessa ou atribui data inicial de um intervalo
- `int_end()`: acessa ou atribui data final de um intervalo
- `int_length()`: comprimento de um intervalo em segundos
- `int_flip()`: inverte a ordem da data de início e da data de término em um intervalo
- `int_shift()`: desloca as datas de início e término de um intervalo
- `int_aligns()`: testa se dois intervalos compartilham um ponto final
- `int_standardize()`: garante que todos os intervalos sejam positivos
- `int_diff()`: retorna os intervalos que ocorrem entre os elementos de data/horário
- `int_overlaps()`: testa se dois intervalos se sobrepõem
- `%within%`: testa se o primeiro intervalo está contido no segundo

```
## Criando duas datas - início de estudos do R e nascimento do meu filho
r_inicio <- lubridate::dmy("30-11-2011")
filho_nascimento <- lubridate::dmy("26-09-2013")
r_hoje <- lubridate::today()

## Criando intervalos - interval
r_intervalo <- lubridate::interval(r_inicio, r_hoje)

## Criando intervalos - interval %--%
filho_intervalo <- filho_nascimento %--% lubridate::today()

## Operações com intervalos
lubridate::int_start(r_intervalo)
#> [1] "2011-11-30 UTC"
lubridate::int_end(r_intervalo)
#> [1] "2022-02-22 UTC"
lubridate::int_length(r_intervalo)
#> [1] 322876800
lubridate::int_flip(r_intervalo)
#> [1] 2022-02-22 UTC--2011-11-30 UTC
lubridate::int_shift(r_intervalo, duration(days = 30))
#> [1] 2011-12-30 UTC--2022-03-24 UTC
```

Uma operação de destaque é verificar a sobreposição entre dois intervalos.

```
## Verificar sobreposição - int_overlaps
lubridate::int_overlaps(r_intervalo, filho_intervalo)
#> [1] TRUE

## Verificar se intervalo está contido
r_intervalo %within% filho_intervalo
#> [1] FALSE
```

```
filho_intervalo %within% r_intervalo
#> [1] TRUE
```

Podemos ainda calcular quantos períodos existem dentro de um intervalo, utilizando as operações de `/` e `%/%`.

```
## Períodos dentro de um intervalo - anos
r_intervalo / lubridate::years()
#> [1] 10.23014
r_intervalo %/% lubridate::years()
#> [1] 10

## Períodos dentro de um intervalo - dias e semandas
filho_intervalo / lubridate::days()
#> [1] 3071
filho_intervalo / lubridate::weeks()
#> [1] 438.7143
```

Ainda podemos fazer transformações dos dados para períodos e ter todas as unidades de data e tempo que o intervalo compreende.

```
## Tempo total estudando R
lubridate::as.period(r_intervalo)
#> [1] "10y 2m 23d 0H 0M 0S"

## Idade do meu filho
lubridate::as.period(filho_intervalo)
#> [1] "8y 4m 27d 0H 0M 0S"
```

Por fim, fusos horários tendem a ser um fator complicador quando precisamos analisar informações instantâneas de tempo (horário) de outras partes do planeta, ou mesmo fazer conversões dos horários. No `lubridate` há funções para ajudar nesse sentido. Para isso, podemos utilizar a função `lubridate::with_tz()` e no argumento `tzone` informar o fuso horário para a transformação do horário.

Podemos descobrir o fuso horário que o R está considerando com a função `Sys.timezone()`.

```
## Fuso horário no R
Sys.timezone()
#> [1] "America/Sao_Paulo"
```

No R há uma listagem dos nomes dos fusos horários que podemos utilizar no argumento `tzone` para diferentes fusos horários.

```
## Verificar os fuso horários
length(OlsonNames())
#> [1] 595
head(OlsonNames())
#> [1] "Africa/Abidjan"     "Africa/Accra"       "Africa/Addis_Ababa"
"Africa/Algiers"     "Africa/Asmara"
#> [6] "Africa/Asmera"
```

Podemos nos perguntar que horas são em outra parte do globo ou fazer as conversões facilmente no `lubridate`.

```
## Que horas são em...
lubridate::with_tz(lubridate::now(), tzone = "America/Sao_Paulo")
#> [1] "2022-02-22 23:44:45 -03"
lubridate::with_tz(lubridate::now(), tzone = "GMT")
#> [1] "2022-02-23 02:44:45 GMT"
lubridate::with_tz(lubridate::now(), tzone = "Europe/Berlin")
#> [1] "2022-02-23 03:44:45 CET"

## Altera o fuso sem mudar a hora
lubridate::force_tz(lubridate::now(), tzone = "GMT")
#> [1] "2022-02-22 23:44:45 GMT"
```

Para se aprofundar no tema, recomendamos a leitura do Capítulo 16 *Dates and times* de Wickham & Grolemund (2017).

5.12 purrr

O pacote `purrr` implementa a *Programação Funcional* no R, fornecendo um conjunto completo e consistente de ferramentas para trabalhar com funções e vetores. A programação funcional é um assunto bastante extenso, sendo mais conhecido no R pela família de funções `purrr::map()`, que permite substituir muitos *loops for* por um código mais sucinto e fácil de ler. Não focaremos aqui nas outras funções, pois esse é um assunto extremamente extenso.

Todas as funções deste pacote são listadas na página de referência do pacote.

Um *loop for* pode ser entendido como uma iteração: um bloco de códigos é repetido mudando um contador de uma lista de possibilidades. Vamos exemplificar com uma iteração bem simples, onde imprimiremos no console os valores de 1 a 10, utilizando a função `for()`, um contador `i` em um vetor de dez números `1:10` que será iterado, no bloco de códigos definido entre `{}`, usando a função `print()` para imprimir os valores.

A ideia é bastante simples: a função `for()` vai atribuir o primeiro valor da lista ao contador `i`, esse contador será utilizado em todo o bloco de códigos. Quando o bloco terminar, o segundo valor é atribuído ao contador `i` e entra no bloco de códigos, repetindo esse processo até que todos os elementos da lista tenham sido atribuídos ao contador.

```
## Loop for
for(i in 1:10){
    print(i)
}
#> [1] 1
#> [1] 2
#> [1] 3
#> [1] 4
#> [1] 5
```

```
#> [1] 6
#> [1] 7
#> [1] 8
#> [1] 9
#> [1] 10
```

Com essa ideia em mente, a programação funcional faz a mesma operação utilizando a função `purrr::map()`. O mesmo *loop for* ficaria dessa forma.

```
## Loop for com map
purrr::map(.x = 1:10, .f = print)
#> [1] 1
#> [1] 2
#> [1] 3
#> [1] 4
#> [1] 5
#> [1] 6
#> [1] 7
#> [1] 8
#> [1] 9
#> [1] 10
#> [[1]]
#> [1] 1
#>
#> [[2]]
#> [1] 2
#>
#> [[3]]
#> [1] 3
#>
#> [[4]]
#> [1] 4
#>
#> [[5]]
#> [1] 5
#>
#> [[6]]
#> [1] 6
#>
#> [[7]]
#> [1] 7
#>
#> [[8]]
#> [1] 8
#>
#> [[9]]
#> [1] 9
#>
```

```
#> [[10]]
#> [1] 10
```

Nessa estrutura, temos:

```
map(.x, .f)
```

- `.x`: um vetor, lista ou data frame
- `.f`: uma função

Num outro exemplo, aplicaremos a função `sum()` para somar os valores de vários elementos de uma lista.

```
## Função map
x <- list(1:5, c(4, 5, 7), c(1, 1, 1), c(2, 2, 2, 2, 2))
purrr::map(x, sum)
#> [[1]]
#> [1] 15
#>
#> [[2]]
#> [1] 16
#>
#> [[3]]
#> [1] 3
#>
#> [[4]]
#> [1] 10
```

Há diferente tipos de retornos da família `purrr::map()`.

- `map()`: retorna uma lista
- `map_chr()`: retorna um vetor de *strings*
- `map_dbl()`: retorna um vetor numérico (*double*)
- `map_int()`: retorna um vetor numérico (*integer*)
- `map_lgl()`: retorna um vetor lógico
- `map_dfr()`: retorna um data frame (por linhas)
- `map_dfc()`: retorna um data frame (por colunas)

```
## Variações da função map
purrr::map_dbl(x, sum)
#> [1] 15 16  3 10
purrr::map_chr(x, paste, collapse = " ")
#> [1] "1 2 3 4 5" "4 5 7"     "1 1 1"     "2 2 2 2 2"
```

Essas funcionalidades já eram conhecidas no *R Base* pelas funções da família *`apply()`*: `apply()`, `lapply()`, `sapply()`, `vapply()`, `mapply()`, `rapply()` e `tapply()`. Essas funções formam a base de combinações mais complexas e ajudam a realizar operações com poucas linhas de código, para diferentes retornos.

Temos ainda duas variantes da função `map()`: `purrr::map2()` e `purrr::pmap()`, para duas ou mais listas, respectivamente. Como vimos para a primeira função, existem várias variações do sufixo para modificar o retorno da função.

```
## Listas
x <- list(3, 5, 0, 1)
y <- list(3, 5, 0, 1)
z <- list(3, 5, 0, 1)

## Função map2
purrr::map2_dbl(x, y, prod)
#> [1]  9 25  0  1

## Função pmap
purrr::pmap_dbl(list(x, y, z), prod)
#> [1]  27 125   0   1
```

Essas funções podem ser usadas em conjunto para implementar rotinas de manipulação e análise de dados com poucas linhas de código, mas que não exploraremos em sua completude aqui. Listamos dois exemplos simples.

```
## Resumo dos dados
penguins %>%
    dplyr::select(where(is.numeric)) %>%
    tidyr::drop_na() %>%
    purrr::map_dbl(mean)
#>    bill_length_mm     bill_depth_mm flipper_length_mm       body_mass_g
year
#>          43.92193          17.15117         200.91520        4201.75439
2008.02924

## Análise dos dados
penguins %>%
    dplyr::group_split(island, species) %>%
    purrr::map(~ lm(bill_depth_mm ~ bill_length_mm, data = .x)) %>%
    purrr::map(summary) %>%
    purrr::map("r.squared")
#> [[1]]
#> [1] 0.2192052
#>
#> [[2]]
#> [1] 0.4139429
#>
#> [[3]]
#> [1] 0.2579242
#>
#> [[4]]
#> [1] 0.4271096
#>
```

```
#> [[5]]
#> [1] 0.06198376
```

Para se aprofundar no tema, recomendamos a leitura do Capítulo 21 *Iteration* de Wickham & Grolemund (2017).

5.13 Para se aprofundar

Listamos a seguir livros que recomendamos para seguir com sua aprendizagem em R e *tidyverse*.

5.13.1 Livros

Recomendamos aos interessados(as) os livros: i) Oliveira e colaboradores (2018) Ciência de dados com R, ii) Grolemund (2018) *The Essentials of Data Science: Knowledge Discovery Using R*, iii) Holmes e Huber (2019) Modern Statistics for Modern Biology, iv) Ismay e Kim (2020) Statistical Inference via Data Science: A ModernDive into R and the Tidyverse, v) Wickham e Grolemund (2017) R for Data Science: Import, Tidy, Transform, Visualize, and Model Data, vi) Zumel e Mount (2014) *Practical Data Science with R Paperback*, vii) Irizarry (2017) Introduction to Data Science: Data Analysis and Prediction Algorithms with R, e viii) Irizarry (2019) Introduction to Data Science.

5.13.2 Links

Para acesso na internet, indicamos os seguintes tutoriais:

- Ciência de Dados em R
- *tidyverse*
- *STHDA - Statistical tools for high-throughput data analysis*
- Estatística é com R! - *tidyverse*
- *Tidyverse Skills for Data Science in R*
- Getting Started with the Tidyverse
- Tidyverse Basics
- Manipulando Dados com dplyr e tidyr

5.14 Exercícios

5.1 Reescreva as operações abaixo utilizando pipes `%>%`.

- `log10(cumsum(1:100))`
- `sum(sqrt(abs(rnorm(100))))`
- `sum(sort(sample(1:10, 10000, rep = TRUE)))`

5.2 Use a função `download.file()` e `unzip()` para baixar e extrair o arquivo do data paper de médios e grandes mamíferos: ATLANTIC MAMMALS. Em seguida, importe para o R, usando a função `readr::read_csv()`.

5.3 Use a função `tibble::glimpse()` para ter uma noção geral dos dados importados no item anterior.

5.4 Compare os dados de penguins (*palmerpenguins::penguins_raw* e *palmerpenguins::penguins*). Monte uma série de funções dos pacotes `tidyr` e `dplyr` para limpar os dados e fazer com que o primeiro dado seja igual ao segundo.

5.5 Usando os dados de penguins (*palmerpenguins::penguins*), calcule a correlação de Pearson entre comprimento e profundidade do bico para cada espécie e para todas as espécies. Compare os índices de correlação para exemplificar o Paradoxo de Simpson.

5.6 Oficialmente a pandemia de COVID-19 começou no Brasil com o primeiro caso no dia 26 de fevereiro de 2020. Calcule quantos anos, meses e dias se passou desde então. Calcule também quanto tempo se passou até você ser vacinado.

Soluções dos exercícios.

Capítulo 6

Visualização de dados

Pré-requisitos do capítulo

Pacotes que serão utilizados neste capítulo.

```
## Pacotes
library(tidyverse)
library(palmerpenguins)
library(datasauRus)
library(gridExtra)

## Dados
penguins <- palmerpenguins::penguins

## Edição dos nomes das colunas para português
colnames(penguins) <- c("especies", "ilha", "comprimento_bico",
                        "profundidade_bico", "comprimento_nadadeira",
                        "massa_corporal", "sexo", "ano")
```

6.1 Contextualização

A visualização de dados através de gráficos frequentemente é a melhor forma de apresentar e interpretar informações ecológicas pois sintetiza os dados e facilita o entendimento de padrões. Geralmente, os gráficos são necessários em quase todas as análises estatísticas, além de enriquecer a argumentação e discussão de hipóteses levantadas para publicações, trabalhos de consultoria, TCCs, dissertações, teses, entre outros.

Existem vários tipos de gráficos para representar os padrões em seus dados para diferentes finalidades. Esses diferentes tipos de gráficos podem até mesmo ser usados para representar o mesmo tipo de dado. Nesta seção, focaremos nos gráficos mais simples para representar uma ou duas variáveis (i.e., gráficos bidimensionais). Dependendo do tipo de variável (categórica ou contínua – veja os tipos de variáveis na Figura 2.2 do Capítulo 2), os gráficos mais indicados para representar os dados mudam. De forma simplificada, os gráficos são representações dos nossos dados tabulares, de modo que os eixos representam as colunas e as feições (pontos, linhas, barras, caixas, etc.) representam as linhas.

Comumente nos gráficos são representados uma ou duas colunas, quando muito três, em gráficos de três dimensões. Para mais colunas, partimos para dados agregados que podem ser vistos no Capítulo 9 de análises multivariadas. Além disso, a utilização de mais de duas colunas pode estar relacionado com outras partes estéticas (e.g., `aes()` no pacote `ggplot2`) do gráfico como cor, forma ou tamanho de pontos, linhas ou outras feições.

Dessa forma, dedicamos esse capítulo inteiramente a apresentar os principais conceitos, como a gramática de gráficos, e uma apresentação geral que pode funcionar como "um guia de bolso" de gráficos, uma vez que apresentamos os principais tipos de gráficos para análises ecológicas e estatísticas. Além disso, no último tópico focamos na finalização (ajustes finos) de gráficos para publicações. Este capítulo fornece a base conceitual necessária para entender a visualização gráfica de dados apresentada ao longo do livro.

Existe uma ampla gama de pacotes para fazer gráficos no R, sendo esse um ponto muito forte dessa linguagem. Além disso, essa grande disponibilidade de pacotes e funções permitem a visualização dos mais diferentes tipos de dados, o que torna a linguagem R detentora de alta praticidade, uma vez que a maior parte dos pacotes possui uma sintaxe relativamente simples para a apresentação de gráficos excelentes e de ótima qualidade. Mais adiante no Capítulo 15, ampliamos a discussão da visualização gráfica com ferramentas para visualização de dados geoespaciais no R.

Este capítulo foi organizado em quatro partes: i) principais pacotes, ii) gramática dos gráficos, iii) tipos de gráficos (um guia de bolso para visualização de vários gráficos no R), e iv) edição de gráficos com alta qualidade para publicação. Apesar de apresentarmos diferentes pacotes com grande potencial para visualização gráfica, focaremos neste capítulo no pacote `ggplot2`, talvez o pacote mais utilizado e com maior gama de possibilidades de criação de excelentes gráficos.

> **Importante**
> Neste capítulo, vamos ensinar gradativamente os comandos para construir gráficos e como editar suas diferentes camadas. Por este motivo, o leitor não verá uma padronização nos temas dos gráficos (fonte, cor, fundo do painel, etc.), uma vez que os gráficos sem edição (padrão do ggplot2) possuem temas pré-definidos. Desse modo, para fazer um gráfico com maior qualidade (chamamos aqui de versão "com ajustes finos") será necessário adicionar comandos extras que são exatamente o foco deste capítulo. Em alguns comandos básicos o leitor vai perceber que devido aos dados utilizados, os eixos vão possuir palavras em inglês misturadas com as palavras em português. Optamos por não traduzir esses nomes para preservar o padrão que será visualizado nos primeiros comandos que cada leitor irá fazer em seu computador.

Usaremos os dados disponíveis no pacote `palmerpenguins` para exemplificar as funções do `ggplot2`, descrito no Capítulo 5.

6.2 Principais pacotes

A seguir, apresentamos uma listagem dos principais pacotes para visualização de dados no R, além das principais funções desses pacotes.

- *graphics*: é o pacote padrão (*default*) do R para produzir gráficos simples, porém útil para visualizações rápidas de quase todos as classes de objetos. Possui funções como: `plot()`, `hist()`, `barplot()`, `boxplot()`, `abline()`, `points()`, `lines()` e `polygon()`. Destacamos que a função `plot()` pode estar presente em diversos pacotes
- *ggplot2*: pacote integrado ao *tidyverse* (Capítulo 5), possui uma sintaxe própria baseada na gramática de gráficos por camadas (*layers*), necessitando de funções específicas para objetos de classes diferentes, demandando geralmente mais tempo para a construção dos códigos. Possui funções como `ggplot()`, `aes()`, `geom_*()`, `facet_*()`, `stats_*()`, `coord_*()` e `theme_*()`, que são conectadas pelo operador `+`
- *ggplot2 extentions*: conjunto de pacotes que adicionam diversas expansões ao pacote `ggplot2`. Exemplos: *gganimate*, *GGally*, *patchwork* e *esquisse*

- *visdat*: cria visualizações preliminares de dados exploratórios de um conjunto de dados inteiro para identificar problemas ou recursos inesperados usando pacote `ggplot2`. Possui diversas funções específicas: `vis_dat()` para visão geral dos dados, `vis_miss()` para visão de dados faltantes (`NA`) e `vis_compare()` para visualizar a diferença entre dados
- *ggpubr*: pacote que fornece funções simplificadas para criar e personalizar gráficos para publicação, baseados no `ggplot2`. Possui funções específicas: `gghistogram()`, `ggdensity()`, `ggboxplot()`, `ggviolin()`, `ggbarplot()` e `ggscatter()`
- lattice: pacote para visualização de dados inspirado nos gráficos treliça (do inglês *Trellis*, geralmente para dados com muitas variáveis que geram uma matriz retangular de gráficos). Também possui funções específicas: `xyplot()`, `histogram()`, `densityplot()`, `barchart()`, `bwplot()` e `dotplot()`. O pacote latticeExtra disponibiliza algumas possibilidades a mais para esse pacote
- plotly: pacote para criar gráficos interativos da web por meio da biblioteca gráfica de JavaScript de código aberto plotly.js. Também possui funções específicas: `plot_ly()`, `add_histogram()`, `add_bars()`, `add_boxplot()`, `add_markers()`, `add_paths()`, `add_lines()` e `add_polygons()`

6.3 Gramática dos gráficos

O livro *The Grammar of Graphics* (Wilkinson & Wills 2005) utiliza uma analogia da linguística para criar uma "gramática" para a visualização gráfica. Segundo ele, a língua se torna expressiva pelo fato da gramática criar um sistema de regras que tornam as declarações com significado conhecido. De maneira semelhante, a ideia da *gramática dos gráficos* cria regras para representação gráfica dos dados a partir de atributos estéticos (do inglês *aesthetic*) como cor, forma e tamanho que definem a geometria dos objetos, como pontos, linhas e barras (Wickham 2016). Além disso, esta gramática reconhece que tais elementos podem ser organizados em camadas, tal como construímos um mapa com diferentes camadas como elevação, hidrografia, rodovias, limites políticos, etc.

Inspirado pela *gramática dos gráficos* proposta por Wilkinson & Wills (2005), Hadley Wickham criou o pacote `ggplot2`, onde "gg" representa a contração de *Grammar of Graphics* (Wickham 2016). As camadas nesta gramática (*layered-grammar*) são organizadas da seguinte forma:

- **Camada 1** - dados `ggplot()`: são as informações no formato `data.frame` que serão usadas nas diferentes camadas nas funções `aes()`, `geom_*()`, `stat_*()`, `facet_*()` e `scale_*()`
- **Camada 2** - mapeamento `aes()`: atributos estéticos, determina que colunas do `data.frame` serão usadas para as representações geométricas, assim como tamanho, forma, cor, preenchimento e transparência
- **Camada 3** - definição da geometria `geom_*()`: define o tipo de gráfico, como pontos, boxplots, violino, linhas, polígonos, entre outros
- **Camada 4** - transformações estatísticas `stat_*()`: modificam, quando necessário, os dados que serão incluídos no gráfico, além de produzir estatísticas como regressões
- **Camada 5** - sistema de coordenadas `coords_*()`: descreve como as coordenadas dos dados são mapeadas para o plano do gráfico
- **Camada 6** - facetas `facets_*()`: especifica como a visualização dos elementos `aes()` são divididos em diferentes "janelas gráficas"

- **Camada 7** - escala `scale_*()`: permite o controle das características visuais (cor, forma e tamanho) dos elementos declarados em `aes()`
- **Camada 8** - temas `theme*()`: controla a aparência visual dos elementos do gráfico, como fontes, cores e legenda

Estruturalmente podemos observar a estrutura da Figura 6.1.

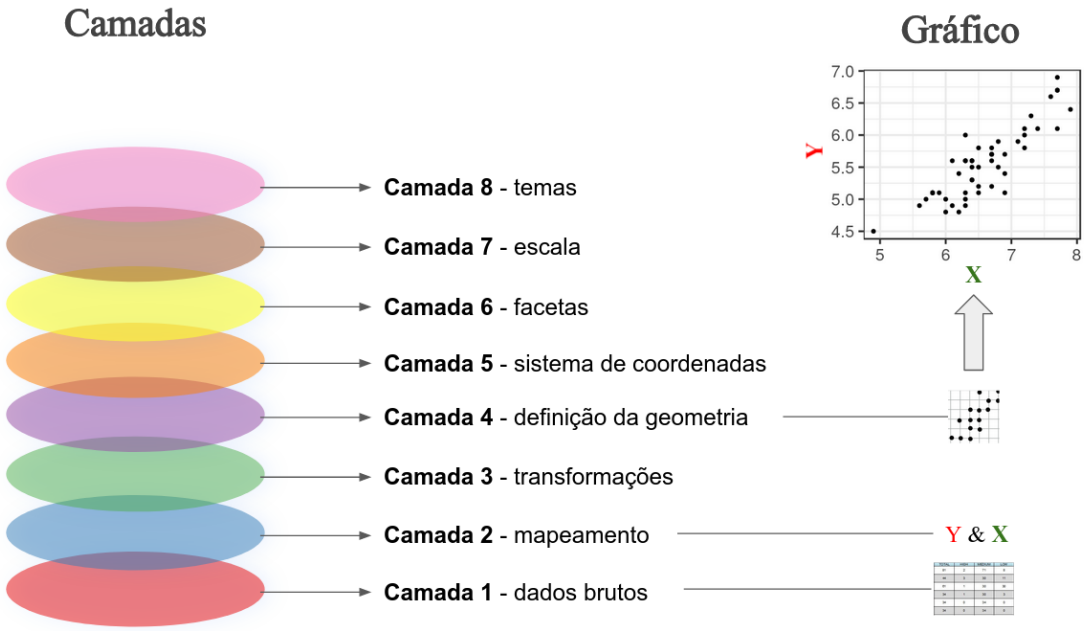


Figura 6.1: Esquema gráfico ilustrando as camadas que definem a strutura de organização aditiva da gramática dos gráficos (ggplot2). No exemplo, a partir de um banco de dados, o mapeamento de quais colunas representam o eixo Y e X e de um atributo gráfico (pontos) é possível construir um gráfico de dispersão que ilustra a relação quantitativa entre a variável Y e X.

Em resumo, o mapeamento gráfico do `ggplot2` segue a seguinte estrutura (Wickham & Grolemund 2017):

```
ggplot(data = <DATA>) +
    <GEOM_FUNCTION>(
        mapping = aes(<MAPPINGS>),
        stat = <STAT>,
        position = <POSITION>
        ) +
    <COORDINATE_FUNCTION> +
    <FACET_FUNCTION> +
    <SCALE_FUNCTION> +
    <THEME_FUNCTION>
```

6.4 Tipos de gráficos

Nesta seção, listamos os principais tipos de gráficos e fazemos uma descrição de quantas colunas e o tipo de variável que eles representam.

- **Histograma (*histogram*)**: distribuição de frequência de uma coluna para dados contínuos (cores diferentes podem representar espécies, populações ou grupos distintos)
- **Gráfico de densidade (*density plot*)**: distribuição da densidade de uma coluna para dados contínuos (assim como no histograma, cores diferentes podem ser utilizadas para representar espécies, populações ou grupos distintos)
- **Gráfico de dispersão (*scatter plot*) e gráfico de linha**: relação entre valores de duas colunas para dados contínuos (X e Y)
- **Diagrama de pontos (*dot plot*)**: distribuição da quantidade de valores agrupados de uma coluna para dados contínuos
- **Gráfico de setores (*pie chart* e *donut chart*)**: representação da quantidade de valores de uma coluna para dados categóricos, geralmente em proporção ou porcentagem
- **Gráfico de barras (*bar plot*)**: representação da quantidade de valores de uma ou mais colunas para dados categóricos
- **Gráfico de caixa (*box plot* e *violin plot*)**: distribuição de valores contínuos de uma coluna (Y) para dois ou mais fatores categóricos de outra coluna (X) no formato de caixas e também no formato de "violinos" (considerando a variação e distribuição)
- **Gráfico pareado (*pairs plot*)**: relação entre valores de duas colunas para dados contínuos (X e Y), para colunas par-a-par

Para facilitar a compreensão das regras da gramática dos gráficos, cada tipo de gráfico segue a mesma estrutura de organização, que respeita as camadas de informação descritas na seção anterior. Podemos perceber, portanto, que algumas camadas não são necessárias dependendo do tipo de gráfico ou do conjunto de dados que pretendemos analisar.

Nos exemplos a seguir, a *versão padrão* se refere à representação determinada no *default* das funções do pacote `ggplot2`. Desse modo, somente informamos as variáveis que serão utilizadas dentro de cada camada e a forma geométrica (i.e., tipo de gráfico) desejada. Porém, para cada tipo gráfico apresentamos funções e argumentos para ajustes finos e personalizados.

6.4.1 Histograma (*histogram*)

O histograma é um gráfico extremamente popular, sendo bastante útil para visualizar a distribuição de variáveis contínuas. É bem provável que você já tenha visto um histograma quando aprendeu pela primeira vez a famosa *distribuição normal* (Figura 6.2).

```
## Dados
dist_normal <- data.frame(x = rnorm(10000, mean = 0, sd = 1))

## Histograma de uma variável contínua
ggplot(data = dist_normal, aes(x = x)) +
    geom_histogram()
```

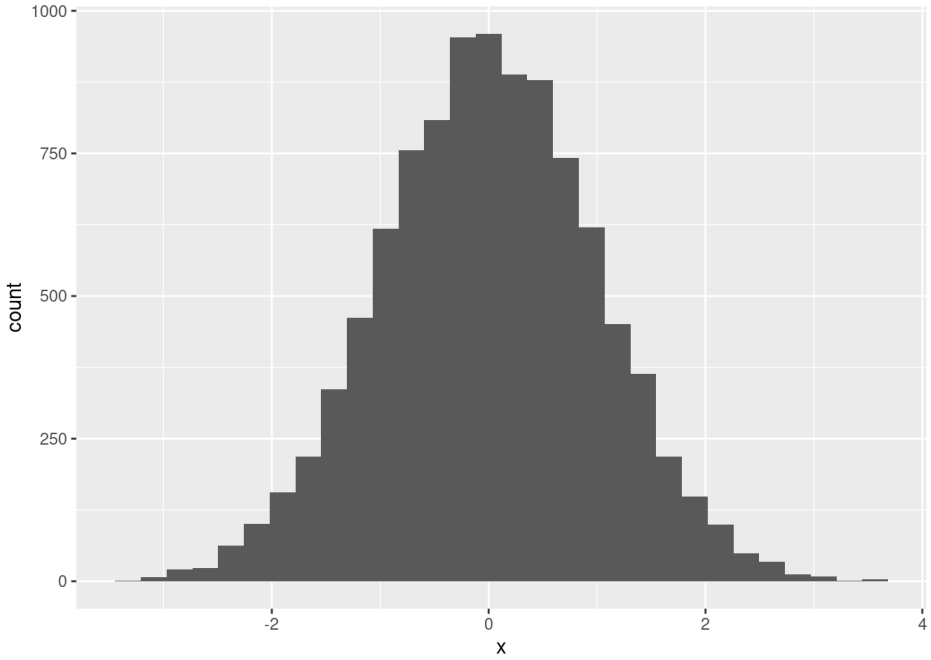


Figura 6.2: Histograma de uma distribuição normal com 10000 observações.

Neste histograma é possível entender que a maioria dos valores da variável `X` do objeto `dist_normal` estão próximos ao valor da média, i.e., zero. Em Ecologia, os histogramas são utilizados para visualizar, por exemplo, a variação morfológica entre espécies (subespécies, gênero, famílias, etc.), variação de parâmetros populacionais entre diferentes espécies ou dentro da mesma espécie em diferentes localidades.

Versão padrão

Vamos utilizar o conjunto de dados `palmerpenguins` para construir um histograma da distribuição da variável *`comprimento_nadadeira`* com a função `geom_hitogram()`. Esta função utiliza uma variável contínua no eixo X e a frequência de cada categoria de intervalo de valores no eixo Y. O gráfico a seguir representa a frequência de uma variável (neste caso, a medida de todos os pinguins, independente da espécie) (Figura 6.3).

```
## Histograma da coluna flipper_length_mm
ggplot(data = penguins, aes(x = comprimento_nadadeira)) +
    geom_histogram()
```

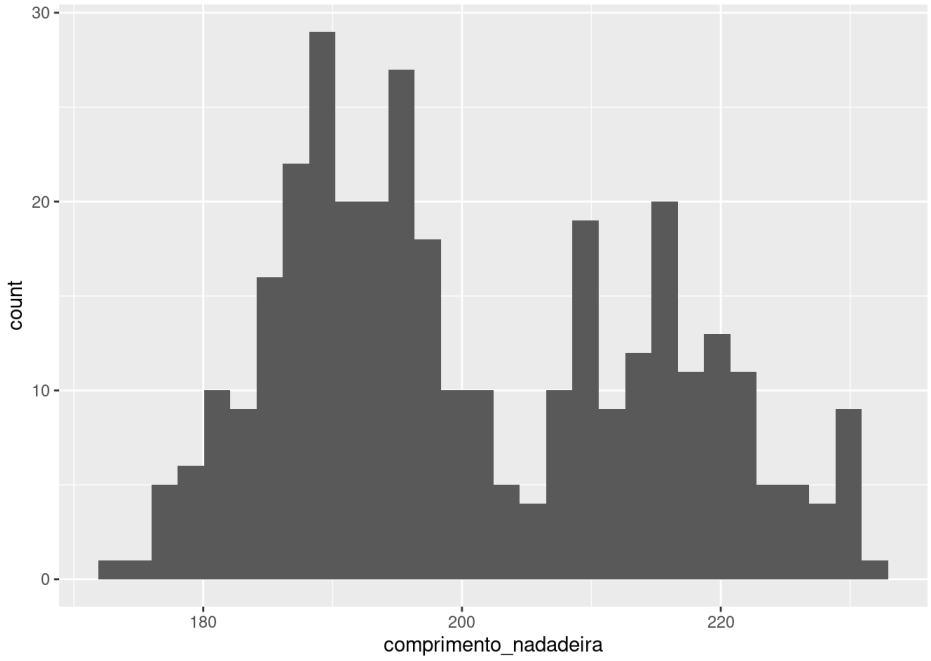


Figura 6.3: Histograma da variável `comprimento_nadadeira`.

Definindo o número de classes

Vamos utilizar o argumento `bins` para definir em quantas classes a variável `x` deve ser dividida (Figura 6.4).

```
## Histograma com 10 classes
ggplot(data = penguins, aes(x = comprimento_nadadeira)) +
    geom_histogram(bins = 10) +
    labs(title = "10 classes")

## Histograma com 30 classes
ggplot(data = penguins, aes(x = comprimento_nadadeira)) +
    geom_histogram(bins = 30) +
    labs(title = "30 classes")
```

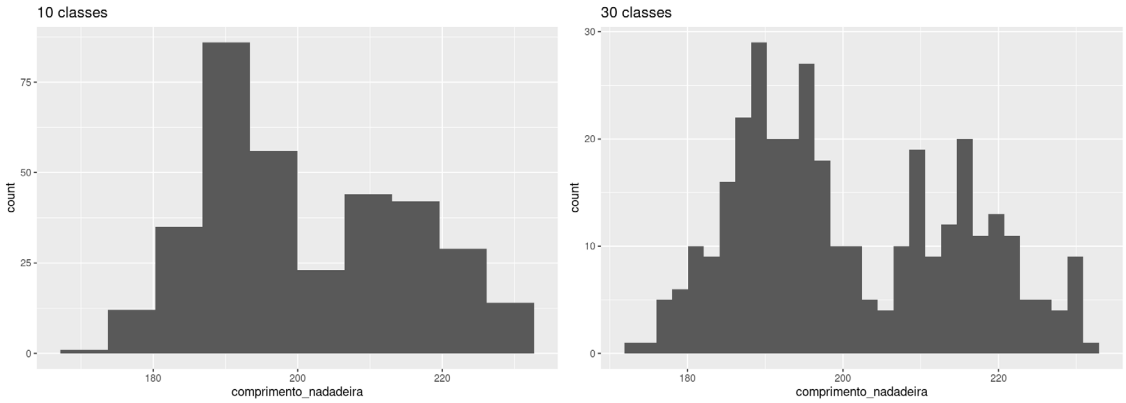


Figura 6.4: Histograma da variável `comprimento_nadadeira` para diferentes classes de divisão.

Comparando múltiplas categorias

Se quisermos comparar a distribuição de uma variável contínua entre diferentes categorias, podemos utilizar o argumento `fill` para colorir o gráfico. No exemplo abaixo, utilizamos cores diferentes para ilustrar a distribuição da variável X (`x = comprimento_nadadeira`) entre espécies diferentes (`fill = especies`) (Figura 6.5).

```
## Histograma com cores para diferentes categorias com sobreposição
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_histogram(alpha = .4) +
    labs(title = "Com sobreposiçao")

## Histograma com cores para diferentes categorias sem sobreposição
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_histogram(position = "dodge") +
    labs(title = "Sem sobreposiçao")
```

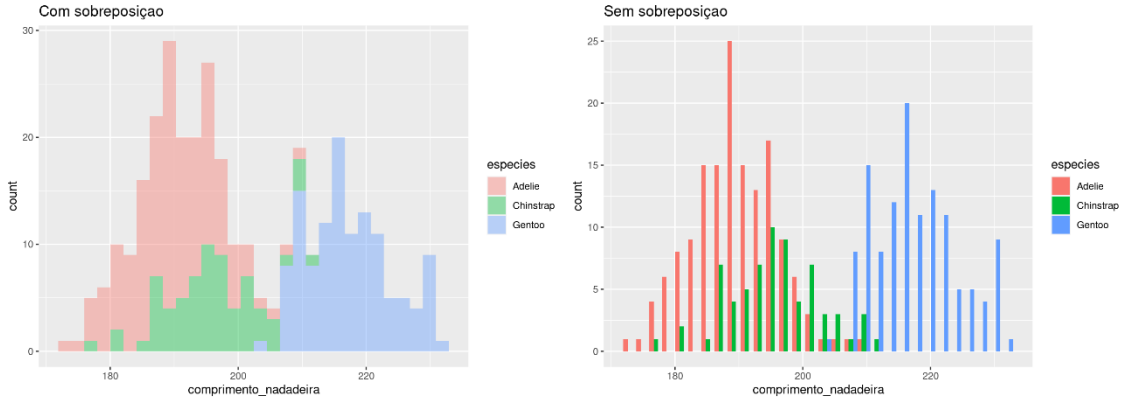


Figura 6.5: Histograma da variável `comprimento_nadadeira` para diferentes espécies com e sem sobreposição.

Ajustes finos (versão personalizada)

```
## Histograma exemplo
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_histogram(alpha = .4, position = "identity") +
    scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    theme_bw(base_size = 14) +
    labs(x = "Comprimento da nadadeira (mm)",
         y = "Frequência absoluta", fill = "Espécies")
```

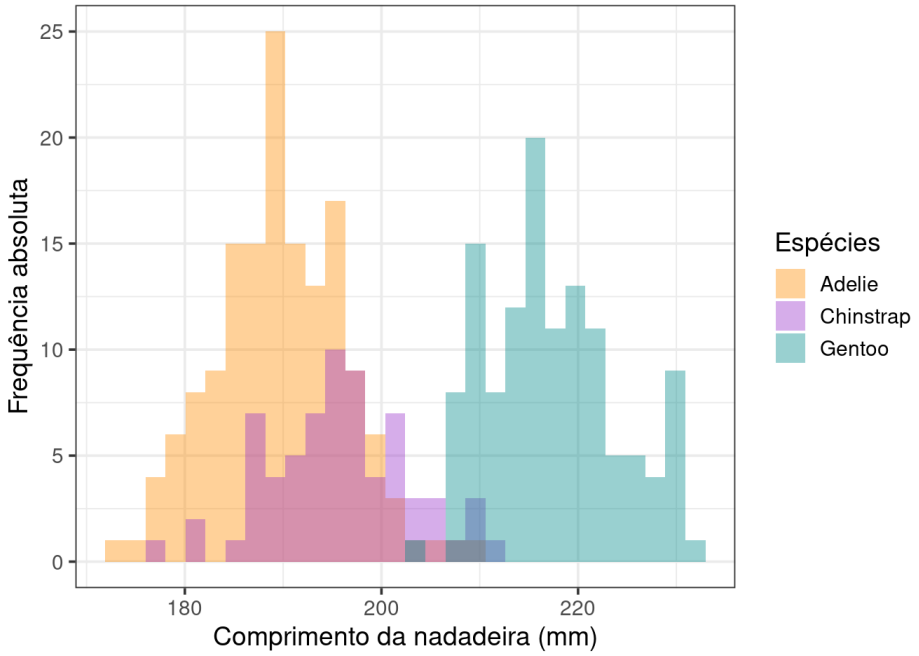


Figura 6.6: Histograma da variável `comprimento_nadadeira` com ajustes finos.

Principais camadas utilizadas na função `geom_histogram()`

- `aes()`
- Eixo X (`x`): variável contínua (`comprimento_nadadeira`)
- Preenchimento (`fill`): variável categórica (`especies`) que define as cores tendo como base o número de níveis dentro desta categoria
- `geom()`
- `geom_histogram()`: para que a variável contínua seja plotada como histograma
- Transparência das linhas e preenchimentos (`alpha`): 0.5 (varia de 0, transparência máxima, a 1, sem transparência)
- Posição das barras: o argumento `position` define se as barras devem ser inseridas de maneira sobreposta (`position = "identity"`) ou não (`position = "dodge"`)
- `scale()`
- `scale_fill_manual()`: para definir manualmente as cores
- `theme()`
- `theme_bw()`: para selecionar o tema com fundo branco
- `labs()`: para personalizar os títulos dos eixos X e Y, e da legenda

6.4.2 Gráfico de densidade (*density plot*)

Nesta seção aprenderemos a criar um gráfico de densidade no R utilizando o `ggplot2`. Assim como o histograma, o **gráfico de densidade** é utilizado para visualizar a distribuição de uma variável contínua em intervalos. Esse gráfico é uma variação do histograma que utiliza Kernel Smoother e, além de ser muito útil para visualizar distribuições, pode ser usado para testar várias hipóteses ecológicas, como descrito no Capítulo 14.

Versão padrão

Vamos utilizar o conjunto de dados `palmerpenguins` para plotar a distribuição da variável *`comprimento_nadadeira`* em um gráfico de densidade. Utilizaremos a função `geom_density()` para plotar uma variável no eixo X (Figura 6.7).

```
## Gráfico de densidade
ggplot(data = penguins, aes(x = comprimento_nadadeira)) +
    geom_density()
```

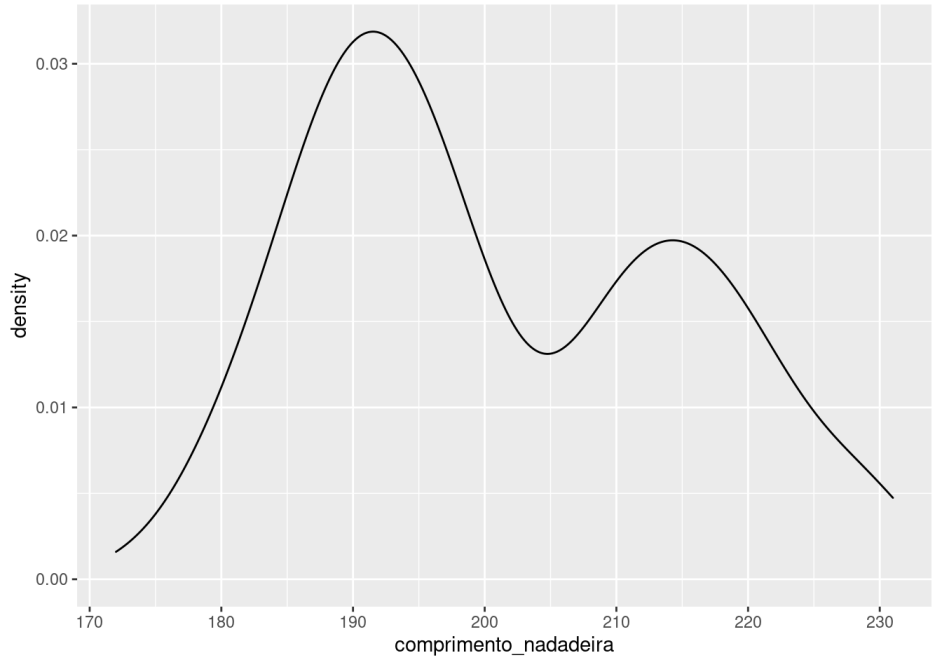


Figura 6.7: Gráfico de densidade da variável `comprimento_nadadeira`.

Além da versão de densidade em linha, é possível utilizar o argumento `fill` para definir a cor de preenchimento do gráfico e o argumento `alpha` para definir a transparência do preenchimento. Utilizamos ainda o argumento `color` para definir a cor da linha (Figura 6.8).

```
## Argumento fill
ggplot(data = penguins, aes(x = comprimento_nadadeira)) +
    geom_density(fill = "cyan4")

## Argumento fill, color e alpha
ggplot(data = penguins, aes(x = comprimento_nadadeira)) +
    geom_density(fill = "cyan4", color = "black", alpha = .4)
```

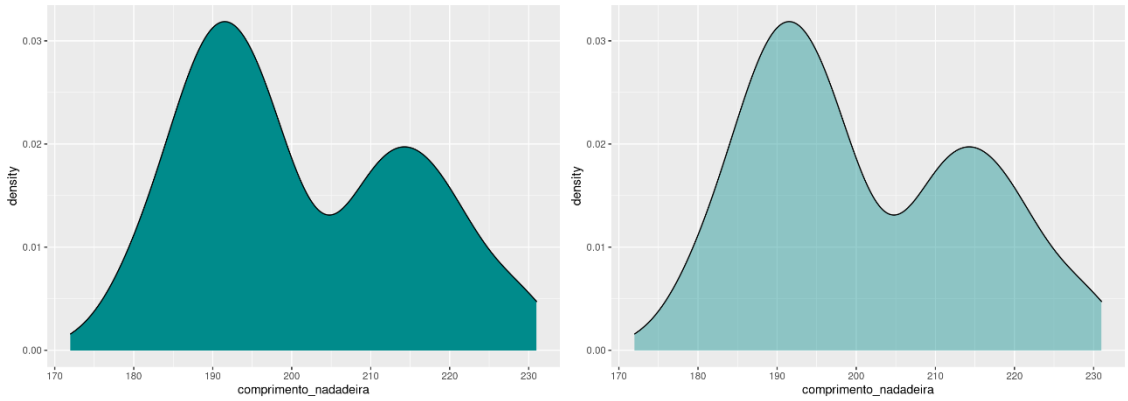


Figura 6.8: Gráfico de densidade da variável `comprimento_nadadeira` com cor e preenchimento.

Comparando múltiplas categorias

Em algumas situações queremos comparar a distribuição de uma variável contínua entre diferentes categorias. Dessa forma, podemos utilizar o argumento `fill` para colorir o gráfico. No exemplo abaixo, utilizamos cores diferentes para ilustrar a distribuição da variável x entre espécies diferentes (`fill = especies`)(Figura 6.9).

```
## O argumento fill preenche cada nível da coluna "especies" (sem
transparência: alpha = 1)
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_density() +
    scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    labs(title = "Sem transparência")

## Gráfico de densidade com cores para diferentes categorias com sobreposição
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_density(alpha = .4) +
    scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    labs(title = "Com transparência")
```

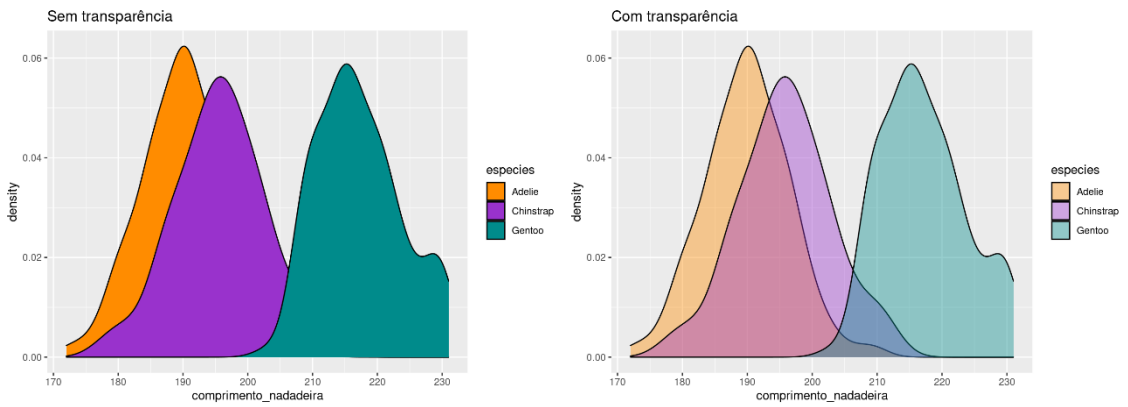


Figura 6.9: Gráfico de densidade da variável `comprimento_nadadeira` para diferentes espécies com e sem transparência.

Ajustes finos (versão personalizada)

```
## Gráfico de densidade exemplo
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_density(alpha = .4) +
    scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    scale_x_continuous(breaks = seq(from = 160, to = 240, by = 10),
                       limits = c(160, 240)) +
    scale_y_continuous(breaks = seq(from = 0, to = .07, by = .01)) +
    theme_bw(base_size = 14) +
    labs(x = "Comprimento da nadadeira (mm)", y = "Densidade",
         fill = "Espécies")
```

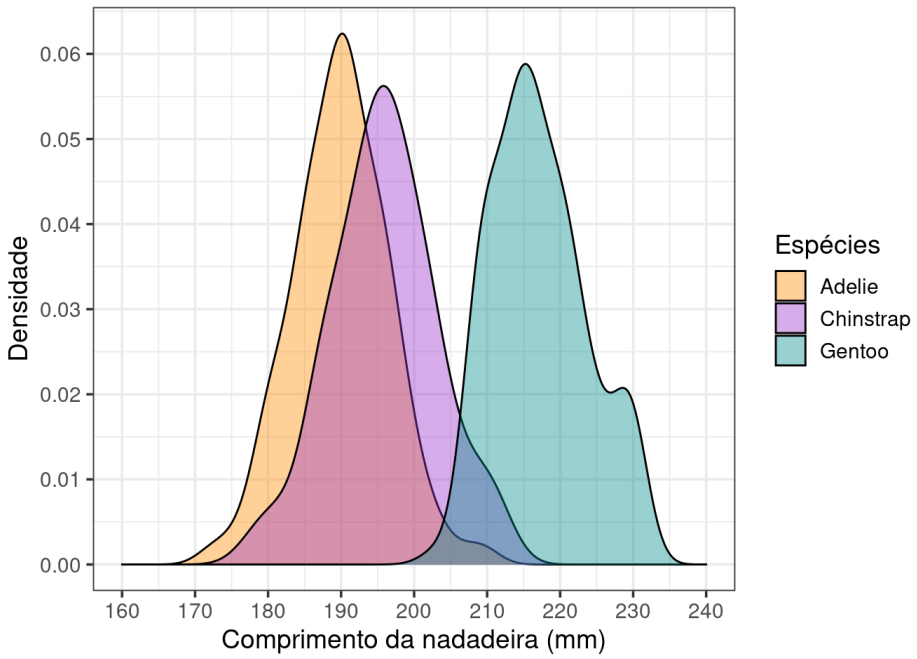


Figura 6.10: Gráfico de densidade da variável `comprimento_nadadeira` com ajustes finos.

Principais camadas utilizadas na função `geom_density()`

- `aes()`
- Eixo X (`x`): variável contínua (`comprimento_nadadeira`)
- Preenchimento (`fill`): variável categórica (`especies`) que define as cores tendo como base o número de níveis dentro desta categoria
- `geom()`
- `geom_density()`: para que a variável contínua seja plotada como densidade
- Transparência das linhas e preenchimentos (`alpha`): 0.5 (varia de 0, transparência máxima, a 1, sem transparência)
- `scale()`
- `scale_fill_manual()`: para definir manualmente as cores de preferência do usuário
- `scale_x_continuous()` e `scale_y_continuous()`: determinam os limites (valores mínimos e máximos) para os dois eixos e, além disso, os intervalos entre os valores (`breaks`)
- `theme()`

- `theme_bw()`: para selecionar o tema com fundo branco
- `labs()`: para personalizar os títulos dos eixos X e Y, e da legenda

6.4.3 Diagrama de pontos (*dot plot*)

Uma alternativa ao histograma e ao gráfico de densidade é o diagrama de pontos (Dot plot, apesar de ser relativamente menos usado em Ecologia.

Versão padrão

Vamos utilizar o conjunto de dados `palmerpenguins` para visualizar a distribuição da variável comprimento_nadadeira com o diagrama de pontos com a função `geom_dotplot()` (Figura 6.11).

```
## Gráfico de pontos
ggplot(data = penguins, aes(x = comprimento_nadadeira)) +
    geom_dotplot(dotsize = .6)
```

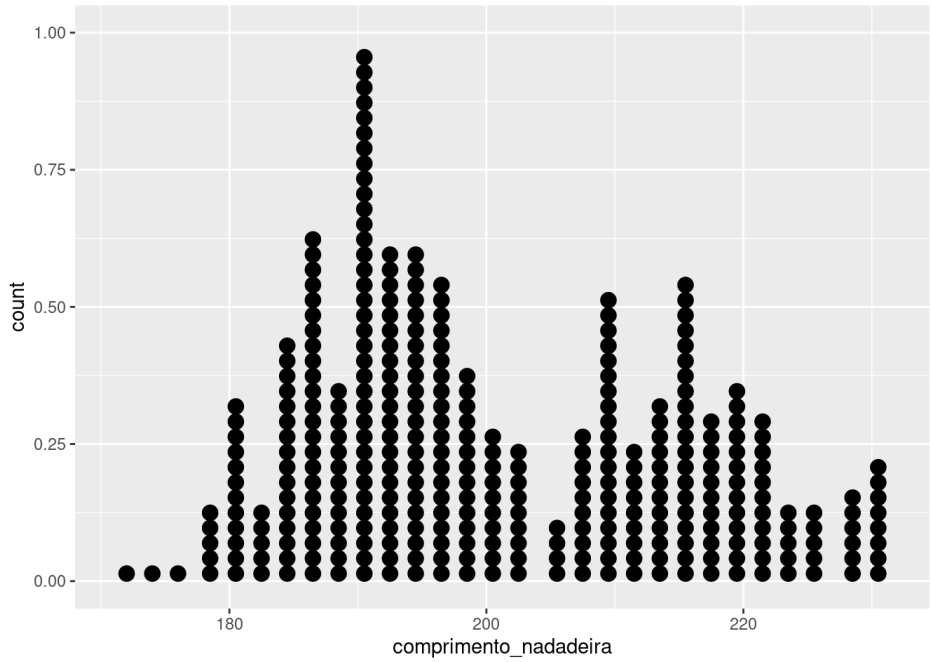


Figura 6.11: Diagrama de pontos da variável `comprimento_nadadeira`.

Comparando múltiplas categorias

Assim como nas funções `geom_histogram()` e `geom_density()`, é possível comparar categorias na função `geom_dotplot()` utilizando o argumento `fill`, bem como os argumentos `color`, `alpha` e `dotsize` (Figura 6.12).

```
## O argumento fill preenche cada nível da coluna "especies" (sem
transparência: alpha = 1)
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_dotplot(dotsize = .9)
```

```
## Diagrama de pontos com cores para diferentes categorias com sobreposição
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_dotplot(dotsize = .7, color = "black", alpha = .4)
```

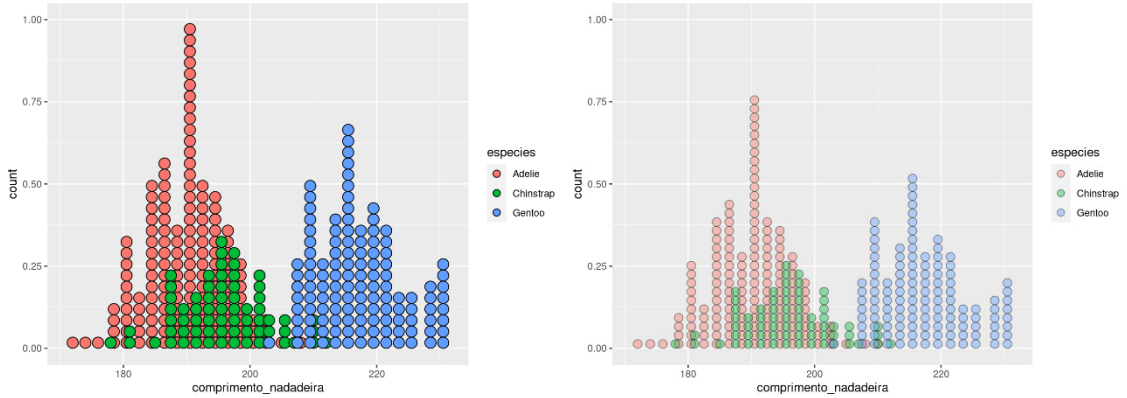


Figura 6.12: Diagrama de pontos da variável `comprimento_nadadeira` para diferentes espécies com e sem transparência.

Ajustes finos (versão personalizada)

```
## Diagrama de pontos exemplo
ggplot(data = penguins, aes(x = comprimento_nadadeira,
                            fill = especies)) +
    geom_dotplot(color = "black", alpha = .7, position = "dodge") +
    scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    scale_x_continuous(breaks = seq(from = 170, to = 240, by = 10),
                       limits = c(170, 240)) +
    scale_y_continuous(breaks = seq(from = 0, to = 1.4, by = .2),
                       limits = c(0, 1.4)) +
    theme_bw(base_size = 14) +
    labs(x = "Comprimento da nadadeira (mm)", y = "Frequência",
         fill = "Espécies")
```

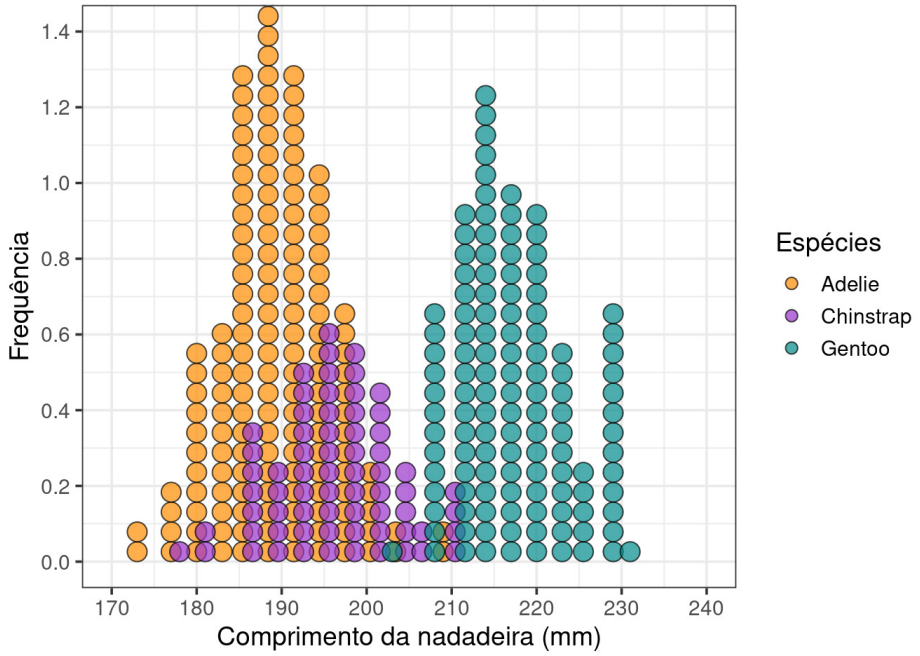


Figura 6.13: Diagrama de pontos da variável `comprimento_nadadeira` com ajustes finos.

> **Importante**
> Uma das limitações do diagrama de pontos é que a sobreposição dos pontos pode não permitir a visualização apropriada desses valores sobrepostos entre diferentes grupos quando comparados.

Principais camadas utilizadas na função `geom_dotplot()`

- `aes()`:
- Eixo X (`x`): variável contínua (`comprimento_nadadeira`)
- Preenchimento (`fill`): variável categórica (`especies`) que define as cores tendo como base o número de níveis dentro desta categoria
- `geom():`
- `geom_dotplot()`: para que a variável contínua seja plotada como diagrama de pontos
- Transparência dos pontos (`alpha`): 0.5 (varia de 0, transparência máxima, a 1, sem transparência)
- Cor da borda do ponto (`color`): valor padrão (se não for especificado) é `black`
- Tamanho dos pontos (`dotsize`): valor padrão (se não for especificado) é 1
- Posição dos pontos: o argumento `position` define se os pontos devem ser inseridos de maneira sobreposta (`position = "identity"`) ou não (`position = "dodge"`)
- `scale()`:
- `scale_fill_manual()` para definir manualmente as cores de preferência do usuário
- `scale_x_continuous()` e `scale_y_continuous()` determinam os limites (valor mínimo e máximo) para os dois eixos e, além disso, os intervalos entre os valores (`breaks`)
- `theme()`:
- `theme_bw()` para selecionar o tema com fundo branco
- `labs()` para personalizar os títulos dos eixos X e Y, e da legenda

6.4.4 Gráfico de barras (*bar plot*)

O gráfico de barras é um dos gráficos mais usados em artigos e livros de Ecologia, uma vez que permite comparar valores absolutos ou médios (combinados com alguma medida de variação, como desvio padrão) de uma variável contínua entre diferentes níveis de uma variável categórica.

Versão padrão

O gráfico de barras utiliza retângulos para representar uma variável contínua ou a contagem de uma variável categórica, sendo que o comprimento dos retângulos é proporcional ao valor que ele representa. Por exemplo, é possível comparar qual a quantidade de indivíduos medidos para cada espécie de pinguim (Figura 6.14).

```
## Número de indivíduos coletados
penguins_count <- penguins %>%
    dplyr::count(especies)
penguins_count
#> # A tibble: 3 × 2
#>   especies      n
#>   <fct>     <int>
#> 1 Adelie      152
#> 2 Chinstrap    68
#> 3 Gentoo      124

## Gráfico de barras
ggplot(data = penguins_count, aes(x = especies, y = n)) +
    geom_bar(stat = "identity")
```

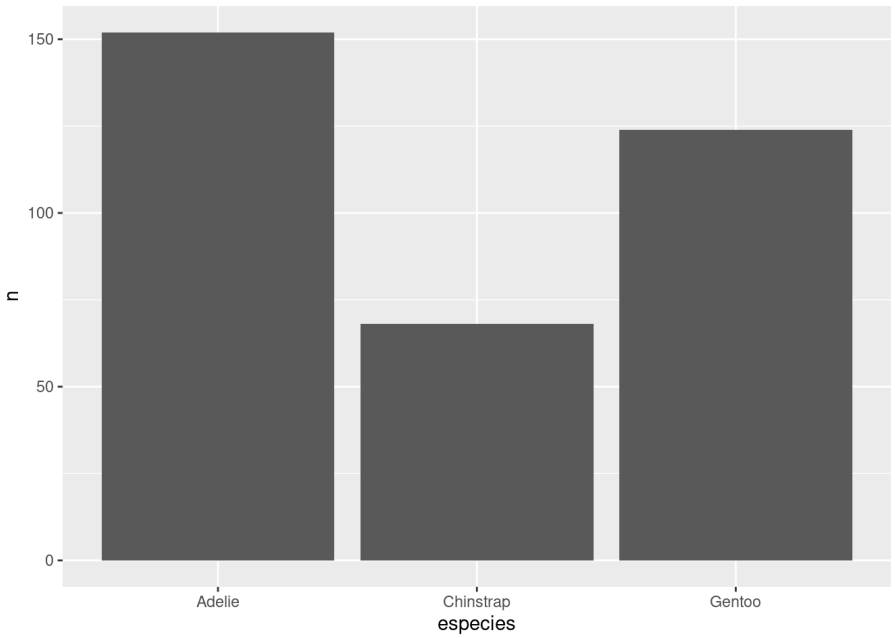


Figura 6.14: Gráfico de barras indicando a quantidade de indivíduos medidos de cada espécie de pinguim.

Além disso, é possível alterar as cores (`color`) e o preenchimento (`fill`) das barras, bem como sua transparência (`alpha`) e largura (`width`), como demonstrado nos próximos quatro gráficos (Figura 6.15).

```
## Modificando o preenchimento
ggplot(data = penguins_count, aes(x = especies, y = n)) +
    geom_bar(stat = "identity", fill = "cyan4")

## Modificando a cor e o preenchimento
ggplot(data = penguins_count, aes(x = especies, y = n)) +
    geom_bar(stat = "identity", color = "cyan4", fill = "white")

## Modificando a largura da barra = .75
ggplot(data = penguins_count, aes(x = especies, y = n)) +
    geom_bar(stat = "identity", width = .75) +
    labs(title = "Largura = .75")

## Modificando a largura da barra = .25
ggplot(data = penguins_count, aes(x = especies, y = n)) +
    geom_bar(stat = "identity", width = .25) +
    labs(title = "Largura = .25")
```

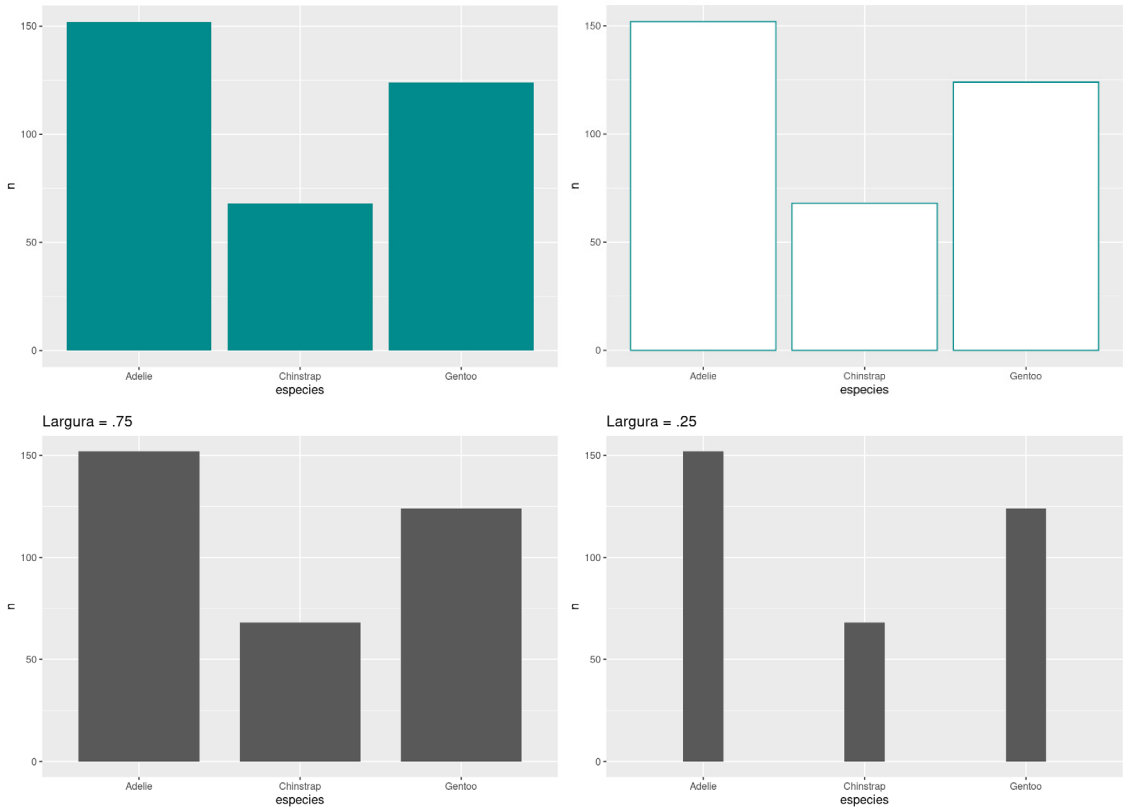


Figura 6.15: Gráficos de barras indicando a quantidade de indivíduos medidos de cada espécie de pinguim, modificando cor, preenchimento, transparência e largura das barras.

Outra possibilidade para representação do gráfico de barras é inverter a direção das barras com a função `coord_flip()` (Figura 6.16).

```
## Barras vertical
ggplot(data = penguins_count, aes(x = especies, y = n)) +
    geom_bar(stat = "identity", width = .6)
```

```
## Barras horizontal
ggplot(data = penguins_count, aes(x = especies, y = n)) +
    geom_bar(stat = "identity", width = .6) +
    coord_flip()
```

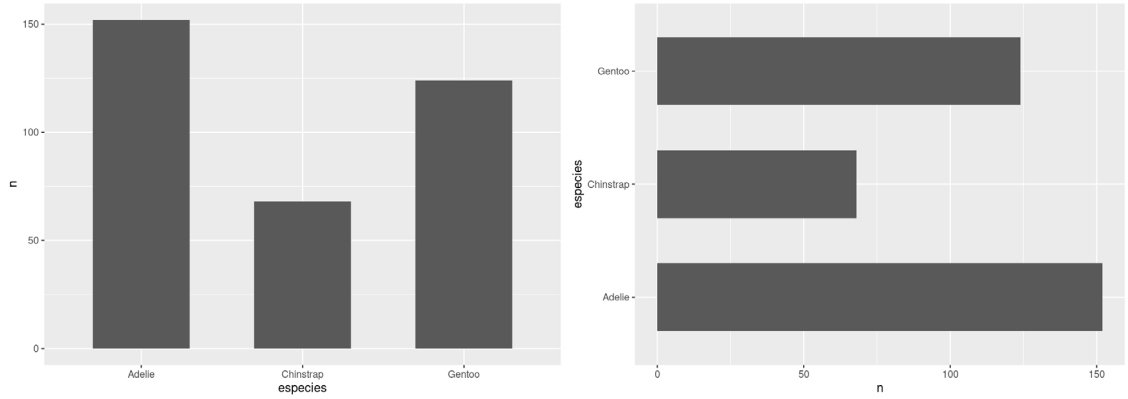


Figura 6.16: Gráficos de barras indicando a quantidade de indivíduos medidos de cada espécie de pinguim, invertendo a direção das barras.

É possível utilizar variáveis categóricas para definir cores e preenchimento e ilustrar, por exemplo, tratamentos ou espécies diferentes com os argumentos `fill` e `color` (Figura 6.17).

```
## Gráfico de barras com preenchimento colorido
ggplot(data = penguins_count, aes(x = especies, y = n,
                                  fill = especies)) +
    geom_bar(stat = "identity")
```

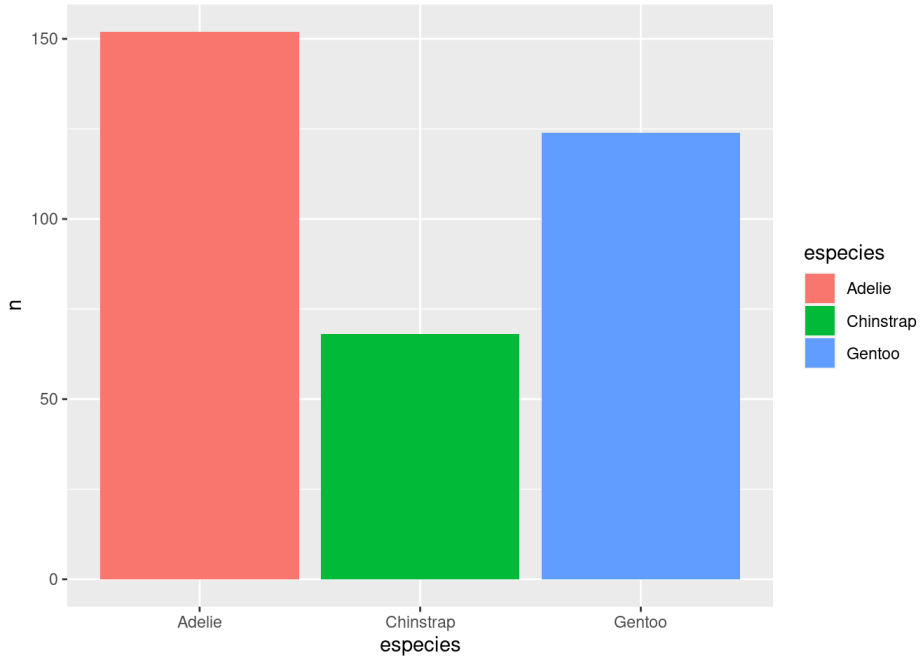


Figura 6.17: Gráfico de barras indicando a quantidade de indivíduos medidos de cada espécie de pinguim, para diferentes espécies.

Adicionando medidas de variação

Em algumas comparações, utilizar somente os valores absolutos pode não ser a visualização mais apropriada, como, por exemplo, em desenhos de ANOVA (Capítulo 7). Desse modo, ao invés do valor máximo da barra representar o valor absoluto (e.g., número de indivíduos de uma espécie), ele vai representar o valor médio. Além disso, linhas adicionais (chamadas barras de erro) vão representar alguma medida de variação como desvio padrão, erro padrão, intervalo de confiança, entre outros (Figura 6.18)

```
## Calcular o desvio padrão por espécie
penguins_media <- penguins %>%
    dplyr::group_by(especies) %>%
    dplyr::summarise(media = mean(comprimento_nadadeira, na.rm = TRUE),
                     desvio = sd(comprimento_nadadeira, na.rm = TRUE))

## Gráfico de barras com desvio padrão
ggplot(data = penguins_media, aes(x = especies, y = media,
                                  fill = especies)) +
    geom_bar(stat = "identity", alpha = .4) +
    geom_errorbar(aes(ymin = media-desvio, ymax = media+desvio),
                  width = .1) +
    geom_point()
```

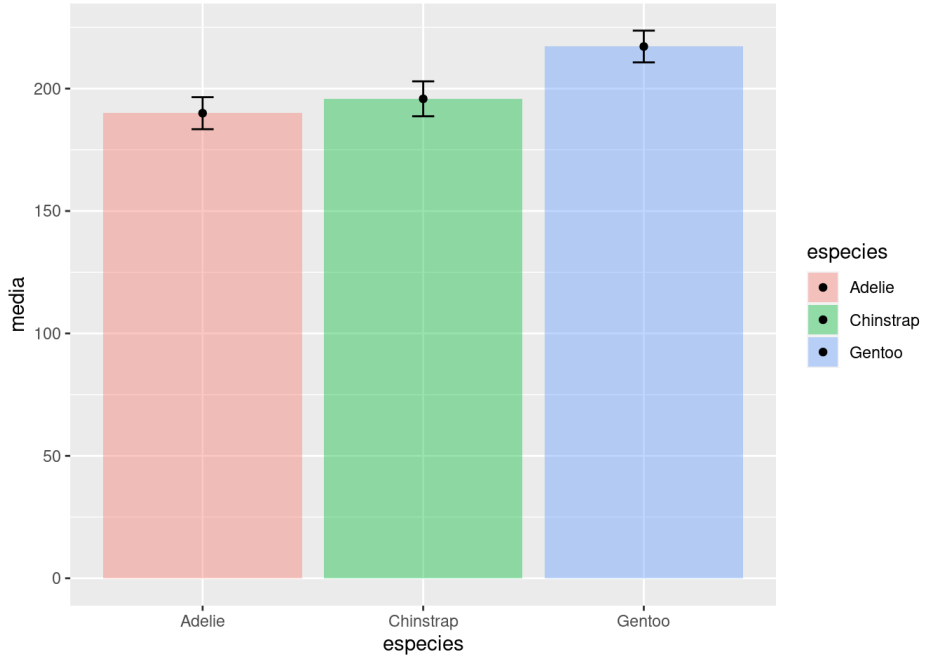


Figura 6.18: Gráfico de barras indicando os valores médios e o desvio padrão da variável `comprimento_nadadeira` para cada espécie de pinguim.

Ajustes finos (versão personalizada)

```
## Gráfico de barra exemplo
ggplot(data = penguins_count, aes(x = especies, y = n,
```

```
                                  fill = especies)) +
    geom_bar(stat = "identity") +
    geom_label(aes(label = n), fill = "white") +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 14) +
    labs(x = "Espécies", y = "Número de indivíduos", fill = "Espécies")
```

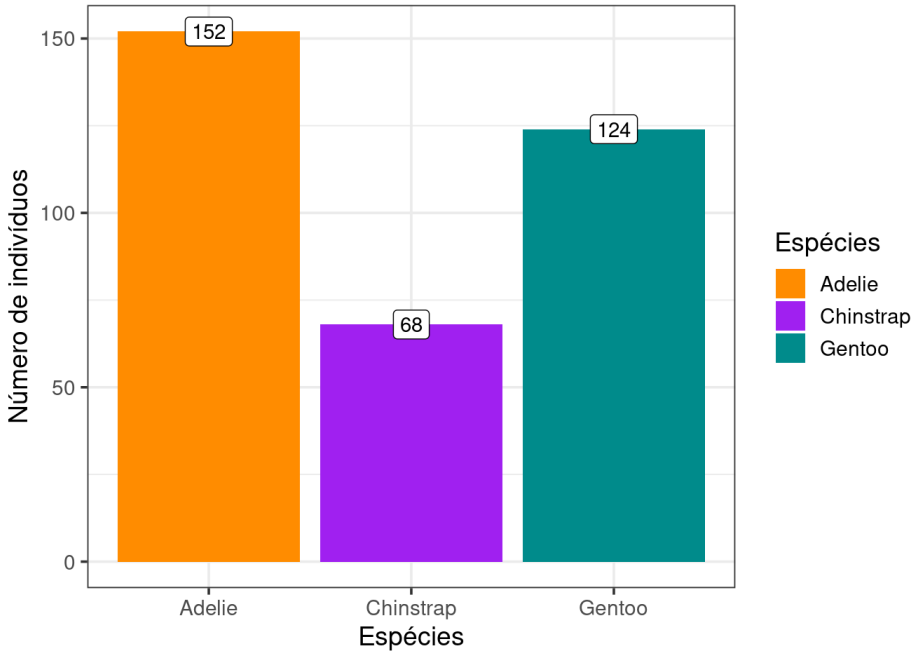


Figura 6.19: Gráfico de barras indicando a quantidade de indivíduos medidos de cada espécie de pinguim, com ajustes finos.

6.4.5 Gráfico de setores (*pie chart* e *donut chart*)

Além do gráfico de barras, o gráfico de setores representa uma alternativa para comparar a proporção entre categorias. Tais gráficos podem ser representados como *pie charts* ou *donut charts*, como demonstrado abaixo. No exemplo abaixo, utilizamos a mesma comparação realizada no item acima. Porém, os valores de contagem (número de indivíduos por espécie) devem ser transformados previamente em proporção.

Gráfico de setores (*pie chart*)

Gráfico de setores do tipo *pie* (Figura 6.20).

```
## Cálculo da proporção - pie
penguins_prop <- penguins %>%
    dplyr::count(especies) %>%
    dplyr::mutate(prop = round(n/sum(n), 4)*100)

## Gráfico de setores
ggplot(data = penguins_prop, aes(x = "", y = prop, fill = especies)) +
    geom_bar(stat = "identity", color = "white") +
```

```
    geom_text(aes(label = paste0(prop, "%")), color = "white",
              position = position_stack(vjust = .5), size = 6) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    coord_polar("y", start = 0) +
    theme_void() +
    labs(fill = "Espécies")
```

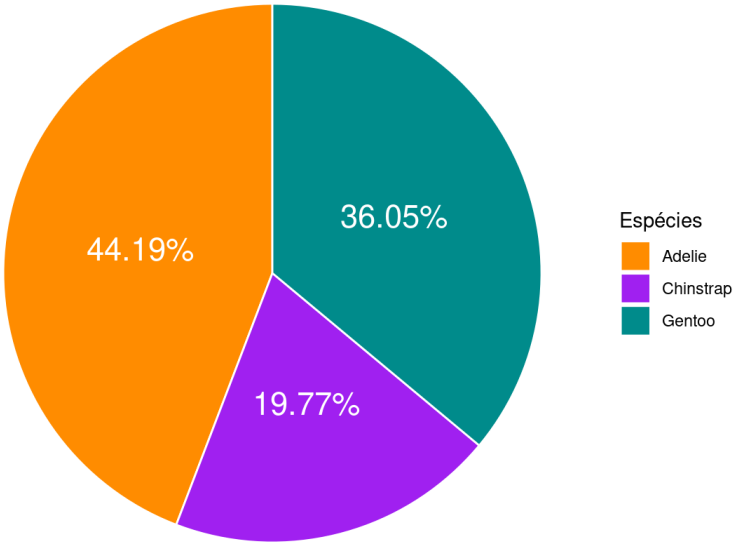


Figura 6.20: Gráfico de setores indicando a proporção de indivíduos medidos de cada espécie de pinguim.

Gráfico de setores (*donut chart*)

Gráfico de setores do tipo *donuts* (Figura 6.21).

```
## Gráfico de setores - donut
ggplot(data = penguins_prop, aes(x = 2, y = prop, fill = especies)) +
    geom_bar(stat = "identity") +
    geom_text(aes(label = paste0(prop, "%")), color = "white",
              position = position_stack(vjust = .5), size = 4) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    xlim(0, 2.5) +
    coord_polar(theta = "y", start = 0) +
    theme_void() +
    theme(legend.position = c(.5, .5),
          legend.title = element_text(size = 20),
          legend.text = element_text(size = 15)) +
    labs(fill = "Espécies")
```

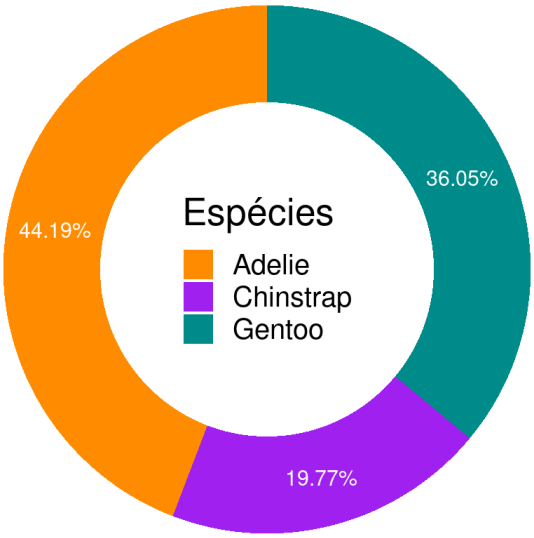


Figura 6.21: Gráfico de setores do tipo donuts indicando a proporção de indivíduos medidos de cada espécie de pinguim.

Comparando gráficos de setores com gráfico de barras

O mesmo conjunto de dados pode ser visualizado de diferentes formas. Não diferente, a comparação da proporção de ocorrências de diferentes categorias pode ser feita de várias maneiras. Abaixo, apresentamos a comparação da proporção de indivíduos por cada uma das três espécies no banco de dados `penguins` (Figura 6.22).

```
## Gráfico de barras - vertical
g_bar_v <- ggplot(data = penguins_prop, aes(x = especies, y = prop,
                                            fill = especies)) +
    geom_bar(stat = "identity") +
    geom_label(aes(label = prop), fill = "white") +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw() +
    theme(legend.position = "none") +
    labs(title = "Gráfico de Barras (Vertical)", x = "Espécies",
         y = "Número de indivíduos", fill = "Espécies")

## Gráfico de barras - horizontal
g_bar_h <- ggplot(data = penguins_prop, aes(x = especies, y = prop,
                                            fill = especies)) +
    geom_bar(stat = "identity") +
    geom_label(aes(label = prop), fill = "white") +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    coord_flip() +
    theme_bw() +
    theme(legend.position = "none") +
    labs(title = "Gráfico de Barras (Horizonal)", x = "Espécies",
         y = "Número de indivíduos", fill = "Espécies")
```

```
## Gráfico de setores - pie
g_pie <- ggplot(data = penguins_prop, aes(x = "", y = prop,
                                          fill = especies)) +
    geom_bar(stat = "identity", color = "white") +
    geom_text(aes(label = paste0(prop, "%")), color = "white",
              position = position_stack(vjust = .5), size = 3) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    coord_polar("y", start = 0) +
    theme_void() +
    labs(title = "Pie chart", fill = "Espécies")

## Gráfico de setores - donut
g_donut <- ggplot(data = penguins_prop, aes(x = 2, y = prop,
                                            fill = especies)) +
    geom_bar(stat = "identity") +
    geom_text(aes(label = paste0(prop, "%")), color = "white",
              position = position_stack(vjust = .5), size = 2) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    coord_polar(theta = "y", start = 0) +
    xlim(0, 2.5) +
    theme_void() +
    theme(legend.position = "none") +
    labs(title = "Donut chart", fill = "Espécies")

## Combinação dos gráfigos
grid.arrange(g_bar_v, g_bar_h, g_pie, g_donut, nrow = 2)
```

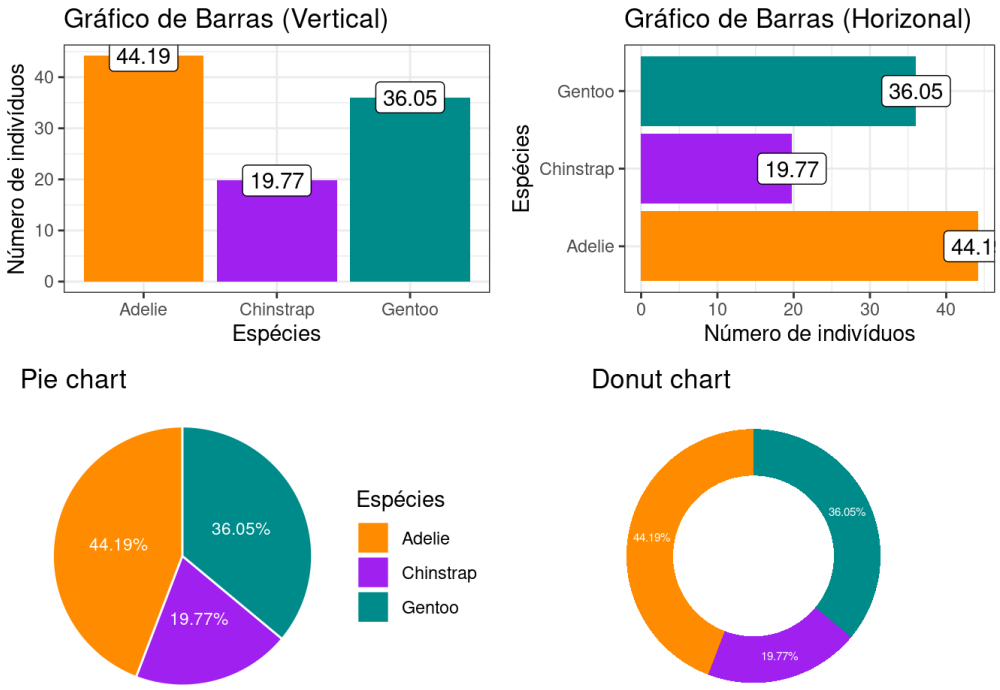


Figura 6.22: Comparação da proporção de indivíduos por cada uma das três espécies dos dados feita através de gráfico de barras, setores do tipo pie e setores do tipo donuts.

Principais camadas utilizadas no gráfico de barras e de setores: `geom_bar()`

- `aes()`
- Eixo X (`x`): variável categórica (`especies`)
- Eixo Y (`y`): variável contínua (`comprimento_nadadeira`)
- Preenchimento (`fill`): a variável categórica (`especies`) define a cor do preenchimento e os níveis dentro desta categoria determinam o número de cores que devem ser indicadas no `scale_fill_manual()`
- `geom()`
- `geom_bar()`: para que as variáveis categóricas sejam plotadas como gráficos de barra ou setores
- Transparência das barras (`alpha`): 0.5 (varia de 0, transparência máxima, a 1, sem transparência)
- `stat`: é necessário usar o argumento `identity` quando os valores do eixo Y são adicionados pelo usuário
- `geom_label()`: forma geométrica que adiciona rótulo dos valores absolutos das barras por categoria (`especies`)
- `geom_errorbar()`: `ymin` e `ymax` delimitam os valores mínimos e máximos, respectivamente, das barras de erro. Tais valores são representados pelo valor da média menos (no caso do ymin) ou mais (no caso do ymax) o valor do intervalo de confiança, desvio ou erro padrão
- `coord`
- `coord_polar()`: sistema de coordenadas para gerar barras circulares sobrepostas (*stacked*) que são usadas nos gráficos de setores (*pie chart* e *donut chart*)
- o argumento `start = 0` indica o local de início do gráfico que, neste caso, começa na "hora" 0 em um "relógio" de 12 horas
- `scale()`
- `scale_fill_manual()`: para definir manualmente as cores de preferência do usuário
- `theme()`
- `theme_bw()`: para selecionar o tema com fundo branco
- `labs()`: para personalizar os títulos dos eixos X e Y, e da legenda.

6.4.6 Gráfico de caixa (*boxplot*)

O gráfico de caixa, também conhecido como boxplot, e amplamente utilizado nos artigos e livros de Ecologia, é uma visualização gráfica que sintetiza informações importantes de dados contínuos como mediana e variação (Figura 6.23) para diferentes níveis de uma variável categórica.

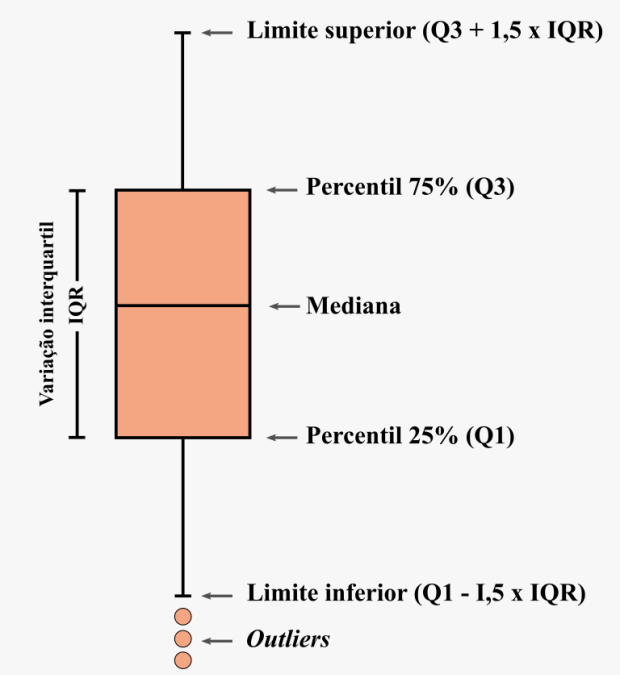


Figura 6.23: Estrutura e elementos do boxplot.

Versão padrão

Vamos plotar uma variável contínua (`comprimento_nadadeira`) no eixo y em função de uma variável categórica no eixo x (`especies`). A definição de qual coluna do bando de dados é a x e qual é a y é feita dentro do na função `aes()` (Figura 6.24).

```
## Gráfico de caixas das coluna comprimento_nadadeira e espécies
ggplot(data = penguins, aes(y = comprimento_nadadeira, x = especies)) +
    geom_boxplot()
```

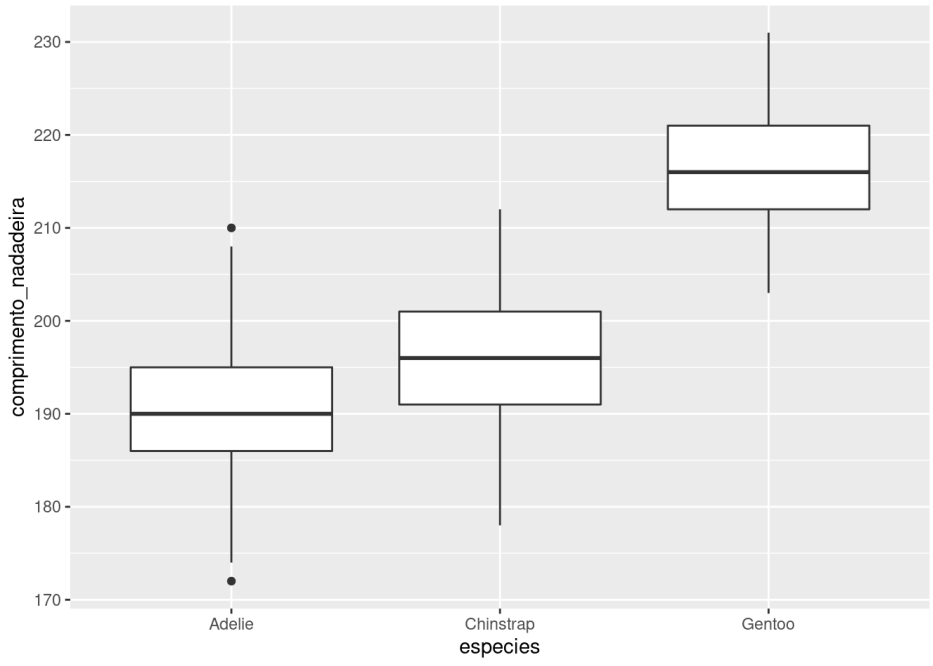


Figura 6.24: Gráfico de caixa para a variável `comprimento_nadadeira` para cada espécie de pinguim.

É possível destacar, se houver, os pontos referentes aos *outliers* (valores discrepantes) com o argumento `outlier.color`. Caso tenha interesse, é possível também remover os outliers do gráfico (Figura 6.25).

```
## Destaque dos outliers
ggplot(data = penguins, aes(y = comprimento_nadadeira, x = especies)) +
    geom_boxplot(outlier.color = "red") +
    labs(title = "Outliers vermelhos")

## Remoção dos outliers
ggplot(data = penguins, aes(y = comprimento_nadadeira, x = especies)) +
    geom_boxplot(outlier.shape = NA) +
    labs(title = "Outliers removidos")
```

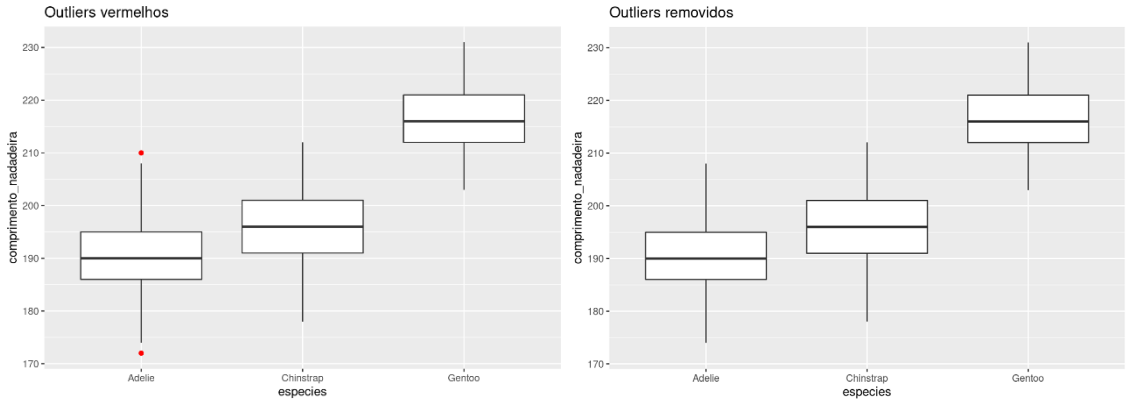


Figura 6.25: Gráfico de caixa para a variável `comprimento_nadadeira` para cada espécie de pinguim, destacando e removendo os outliers.

Outra opção para os gráficos do tipo boxplot é utilizar o argumento `notch = TRUE` para produzir diagramas de caixa entalhados (*notched*). Estes diagramas são úteis para inferir de forma aproximada se existe diferença significativa entre as médias dos grupos (Figura 6.26).

```
## Gráfico com caixa entalhadas
ggplot(data = penguins, aes(y = comprimento_nadadeira, x = especies)) +
    geom_boxplot(notch = TRUE) +
    labs(title = "Caixas entalhadas")
```

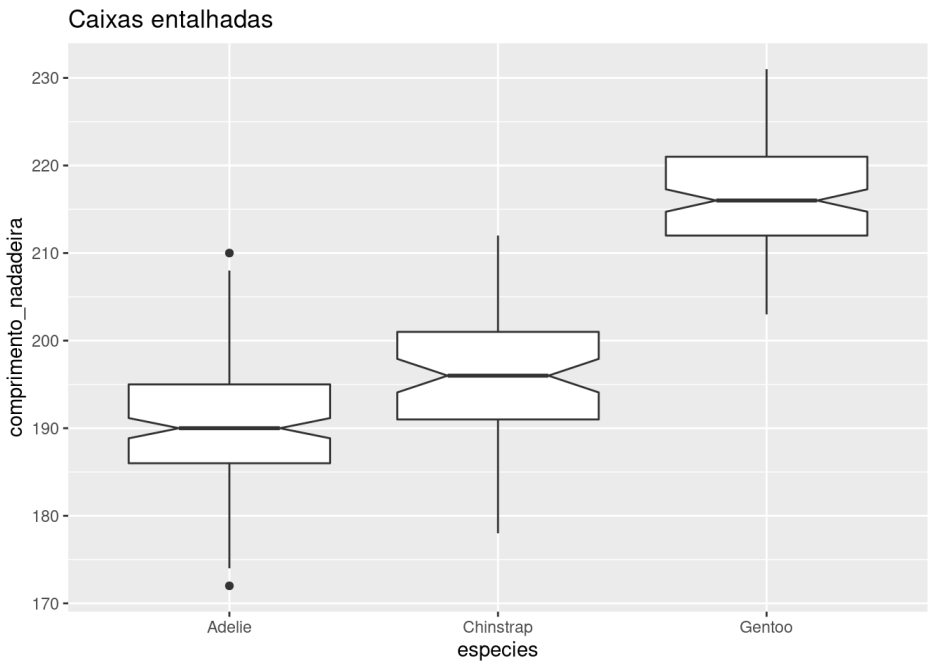


Figura 6.26: Gráfico de caixa para a variável `comprimento_nadadeira` para cada espécie de pinguim, com entalhamentos.

Comparando múltiplas categorias

No exemplo abaixo, utilizamos cores diferentes para ilustrar espécies diferentes através do argumento `fill = especies` (Figura 6.27).

```
## Modificando o preenchimento
ggplot(data = penguins,
       aes(y = comprimento_nadadeira, x = especies, fill = especies)) +
    geom_boxplot()
```

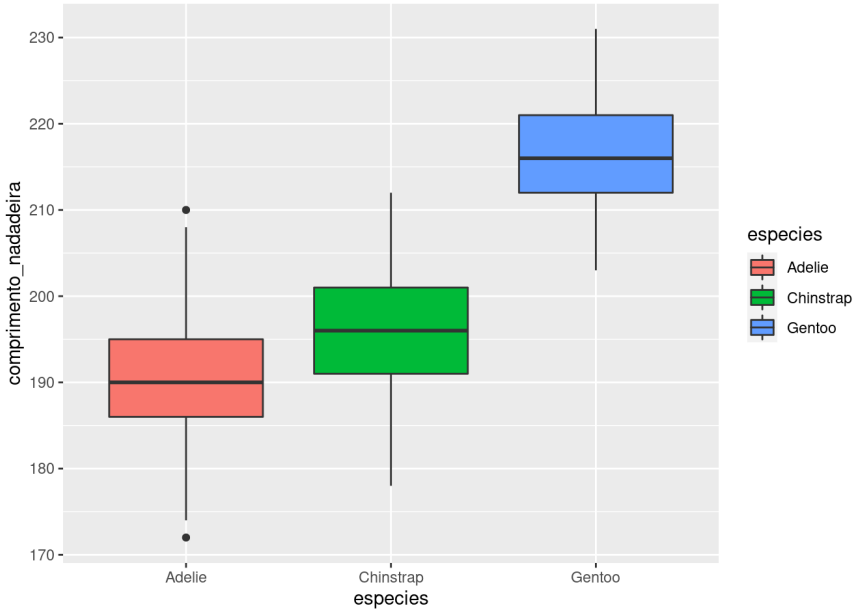


Figura 6.27: Gráfico de caixa para a variável `comprimento_nadadeira` para cada espécie de pinguim, com preenchimentos diferentes.

Combinando boxplot com pontos (*jitter*)

Podemos ainda acrescentar pontos para mostrar a distribuição dos dados, fazendo uma "agitação" (*jitter*) dos pontos sobre as caixas (Figura 6.28).

```
## Boxplot com jitters
ggplot(data = penguins, aes(y = comprimento_nadadeira,
                            x = especies,
                            fill = especies)) +
    geom_boxplot() +
    geom_jitter(size = .5)
```

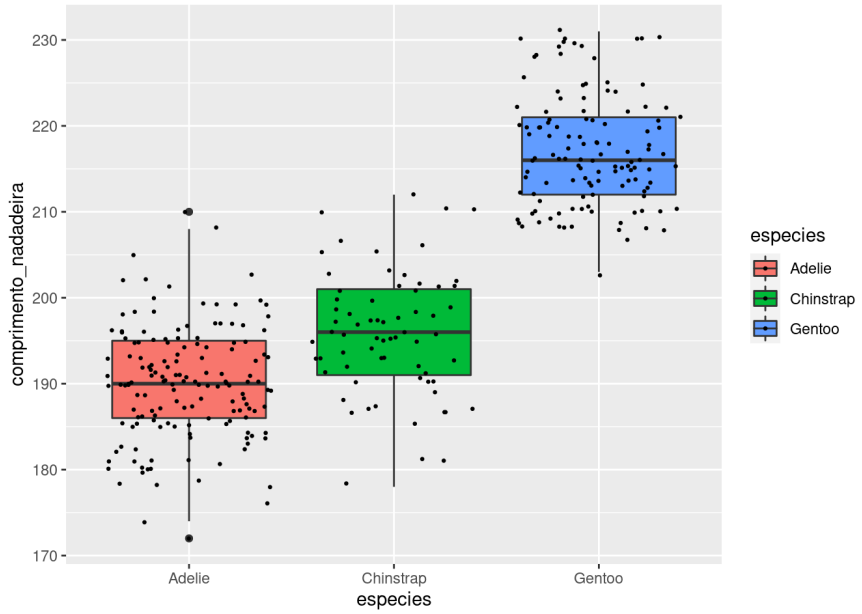


Figura 6.28: Gráfico de caixa para a variável `comprimento_nadadeira` para cada espécie de pinguim, com distribuição dos dados.

Gráfico de violino (*violin plot*) como alternativa ao boxplot

Além das caixas no boxplot, podemos utilizar o formato de "violino" (gráfico de violino) para representar a variação dos dados contínuos para as categorias. A informação adicional ao boxplot que o gráfico de violino permite visualizar é a densidade e distribuição dos pontos, assim como apresentamos acima no gráfico de densidades `geom_density()`. A diferença é que a densidade é espelhada e, desse modo, podemos visualizar os intervalores dos dados com maior ou menor concentração de valores (Figura 6.29).

```
## Gráfico de violino
ggplot(data = penguins, aes(y = comprimento_nadadeira, x = especies,
                            fill = especies)) +
    geom_violin() +
    geom_jitter(size = .5)
```

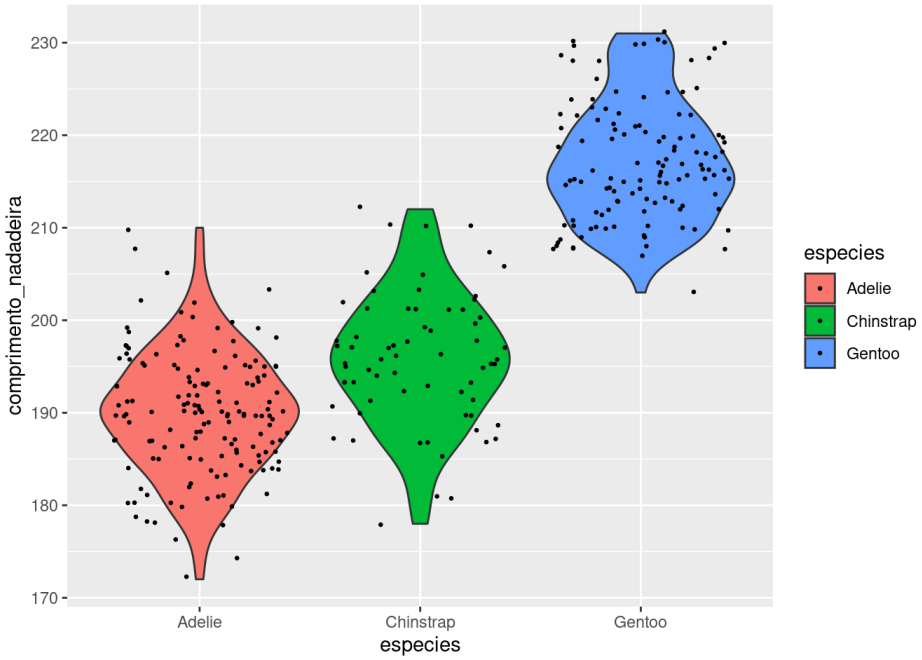


Figura 6.29: Gráfico de violino para a variável `comprimento_nadadeira` para cada espécie de pinguim, com distribuição dos dados.

É possível também combinar boxplot e gráfico de violino em um único gráfico (Figura 6.30).

```
## Combinando o gráfico de violino com o de caixas
ggplot(data = penguins, aes(y = comprimento_nadadeira, x = especies,
                            fill = especies)) +
    geom_violin() +
    geom_boxplot(width = .1, fill = "gray")
```

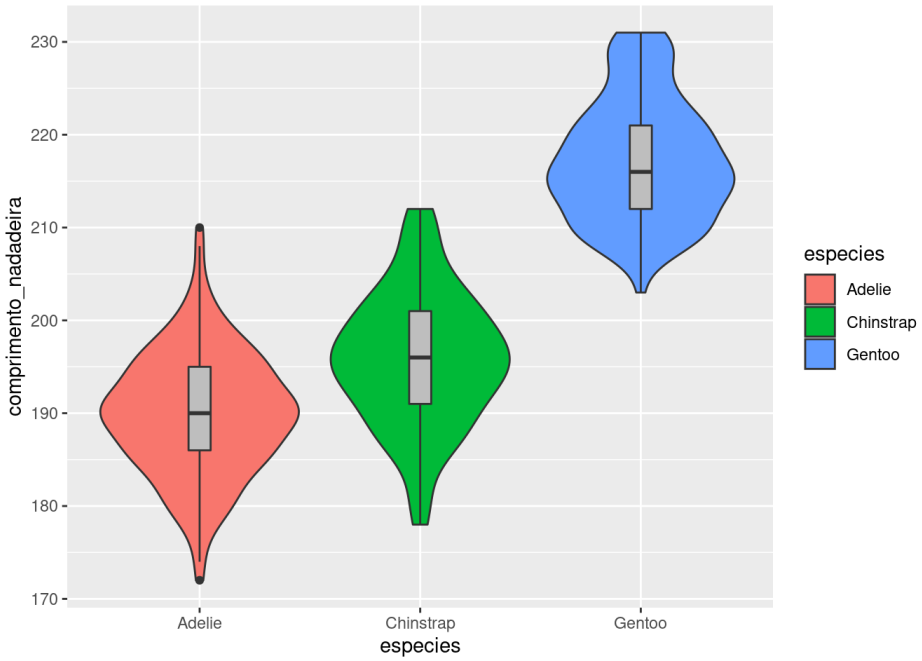


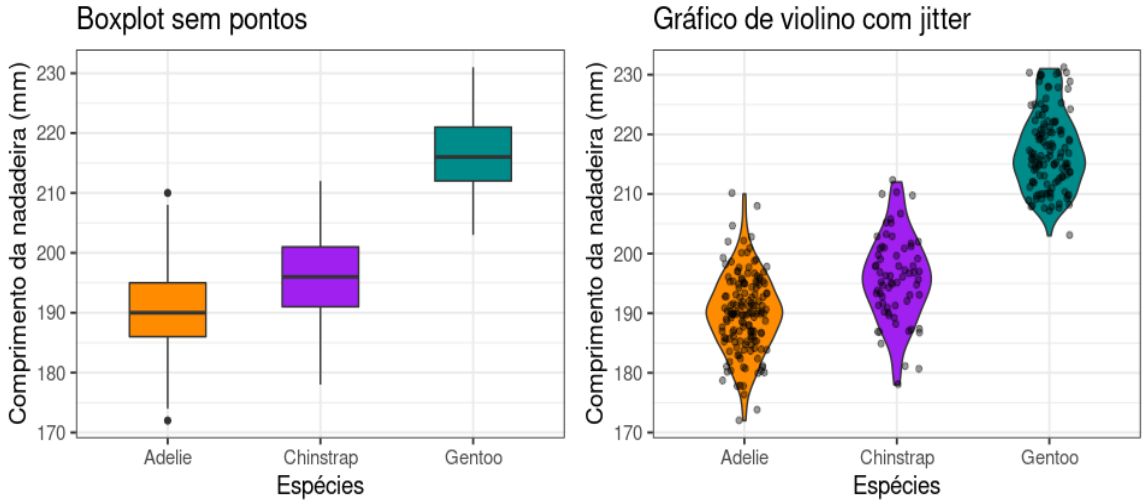
Figura 6.30: Gráfico de caixa e de violino para a variável `comprimento_nadadeira` para cada espécie de pinguim.

Ajustes finos (versão personalizada)

```
## Gráfico de caixas exemplo
ggplot(data = penguins, aes(x = especies, y = comprimento_nadadeira,
                            fill = especies)) +
    geom_boxplot(width = .5, show.legend = FALSE) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 14) +
    labs(title = "Boxplot sem pontos", x = "Espécies",
         y = "Comprimento da nadadeira (mm)")

## Gráfico de violino exemplo
ggplot(data = penguins, aes(x = especies, y = comprimento_nadadeira,
                           fill = especies)) +
    geom_violin(width = .5, show.legend = FALSE) +
    geom_jitter(alpha = .4, show.legend = FALSE,
                position = position_jitter(width = .15, seed = 0)) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 14) +
    labs(title = "Gráfico de violino com jitter", x = "Espécies",
         y = "Comprimento da nadadeira (mm)")

## Gráfico de caixas e violino exemplo
ggplot(data = penguins, aes(x = especies, y = comprimento_nadadeira,
                            fill = especies)) +
    geom_violin(width = .5, show.legend = FALSE) +
    geom_boxplot(width = .3, fill = "gray", show.legend = FALSE) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 14) +
    labs(title = "Gráfico de violino com boxplot", x = "Espécies",
         y = "Comprimento da nadadeira (mm)")
```



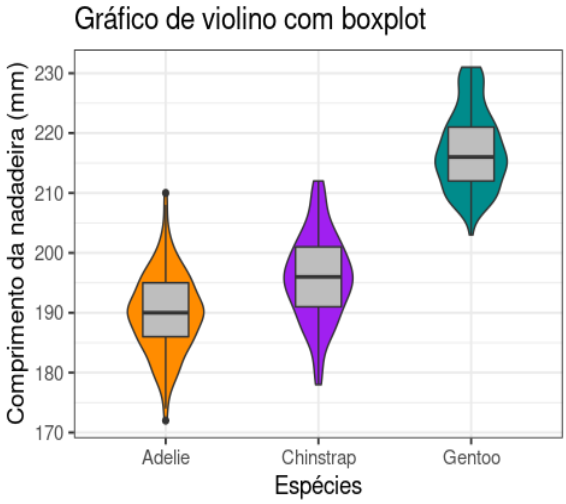


Figura 6.31: Gráfico de caixa combinado ou não com Gráfico de violino e jitter para a variável `comprimento_nadadeira` para cada espécie de pinguim.

Principais camadas utilizadas no `geom_boxplot()` e `geom_violin()`

- `aes()`
- Eixo X (`x`): variável categórica (`especies`)
- Eixo Y (`y`): variável contínua (`comprimento_nadadeira`)
- Preenchimento (`fill`): a variável categórica (`especies`) define a cor do preenchimento e os níveis dentro desta categoria determinam o número de cores que devem ser indicadas no `scale_fill_manual()`.
- `geom():`
- `geom_boxplot()`: para que as variáveis contínuas e categóricas sejam plotadas como gráficos de caixas ou violinos
- `width`: largura das barras ou dos pontos (valor padrão: width = 1)
- `fill`: pode definir uma cor padrão (caso não tenha utilizado o *fill* dentro do argumento `aes()`) como `fill = "gray"`
- `notch`: para utilizar a caixa entalhada o argumento deve ser `notch = TRUE`, a escolha padrão da função `geom_boxplot()` é `notch = FALSE`
- `geom_violin()`: assim como nas outras formas geométricas, é possível controlar largura, cor, preenchimento e transparências dos violinos
- `geom_jitter()`: esta função basicamente "agita" aleatoriamente os pontos para evitar a sobreposição de valores idênticos. Esta função produz a mesma representação se usar a função `geom_point(position = "jitter")`
- `scale()`
- `scale_fill_manual()`: para definir manualmente as cores de preferência do usuário
- `theme()`
- `theme_bw()`: para selecionar o tema com fundo branco
- `labs()`: para personalizar os títulos dos eixos X e Y, e da legenda

6.4.7 Gráfico de dispersão (*scatter plot*)

O gráfico de dispersão (em inglês, *scatter plot*) é famoso na Ecologia por ser a visualização preferida para representar a relação entre área e riqueza de espécies. Neste gráfico, os eixos X e Y são

representados por variáveis contínuas. Em especial, os gráficos de dispersão são usados para representar os resultados testados por análises estatísticas como regressão linear, ANCOVA, mantel, PCA, PCoA, nMDS entre outros (para mais detalhes, consultar os Capítulos 7, 8 e 9).

Versão padrão

Vamos utilizar o conjunto de dados *palmerpenguins* para construir um gráfico de dispersão relacionando as variáveis `comprimento_bico` e `comprimento_nadadeira` com a função `geom_point()`. Esta função utiliza variável contínuas no eixo X e no Y. O gráfico mostra os pares ordenados das medidas das duas variáveis (Figura 6.32).

```
## Gráfico de dispersão das coluna comprimento_nadadeira e comprimento_bico
ggplot(data = penguins, aes(x = comprimento_bico,
                            y = comprimento_nadadeira)) +
    geom_point()
```

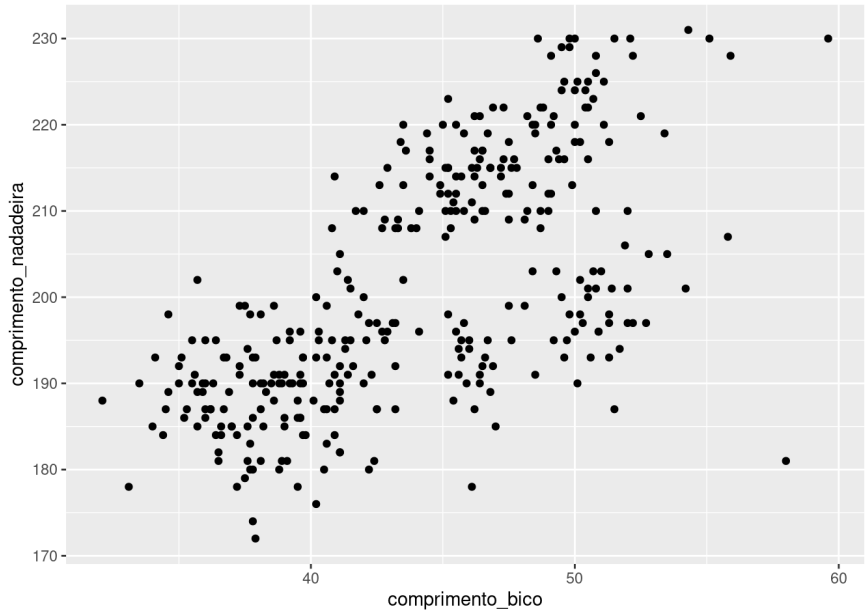


Figura 6.32: Gráfico de dispersão relacionando as variáveis `comprimento_bico` e `comprimento_nadadeira`.

Definindo a cor, tamanho, forma e preenchimento dos pontos

Da mesma forma que para os outros gráficos, podemos alterar cor, tamanho, forma e preenchimento dos pontos (Figura 6.33).

```
## Cor e tamanho dos pontos
ggplot(data = penguins, aes(x = comprimento_bico,
    y = comprimento_nadadeira)) +
    geom_point(color = "cyan4", size = 4) +
    labs(title = "Sem transparência")

## Cor, tamanho dos pontos e transparência
ggplot(data = penguins, aes(x = comprimento_bico,
    y = comprimento_nadadeira)) +
```

```
    geom_point(color = "cyan4", size = 4, alpha = .4) +
    labs(title = "Com transparência")
```

Figura 6.33: Gráfico de dispersão relacionando as variáveis `comprimento_bico` e `comprimento_nadadeira` para cada espécie de pinguim, com alterações na cor, tamanho e preenchimento dos pontos.

A forma dos pontos permite dois controles importantes: a forma em si (símbolos como círculo, quadrado, etc.) e a possibilidade de preenchimento da forma. A figura a seguir discrimina esses símbolos e o valor que deve ser utilizado para desenhar a forma preferida. É importante notar que os símbolos 21 a 25 possuem dois argumentos: (i) `color`, que, na verdade é a cor da borda do símbolo e (ii) `fill`, a cor que define o preenchimento do símbolo. O tipo de símbolo é definido pelo argumento `shape` (Figura 6.34).

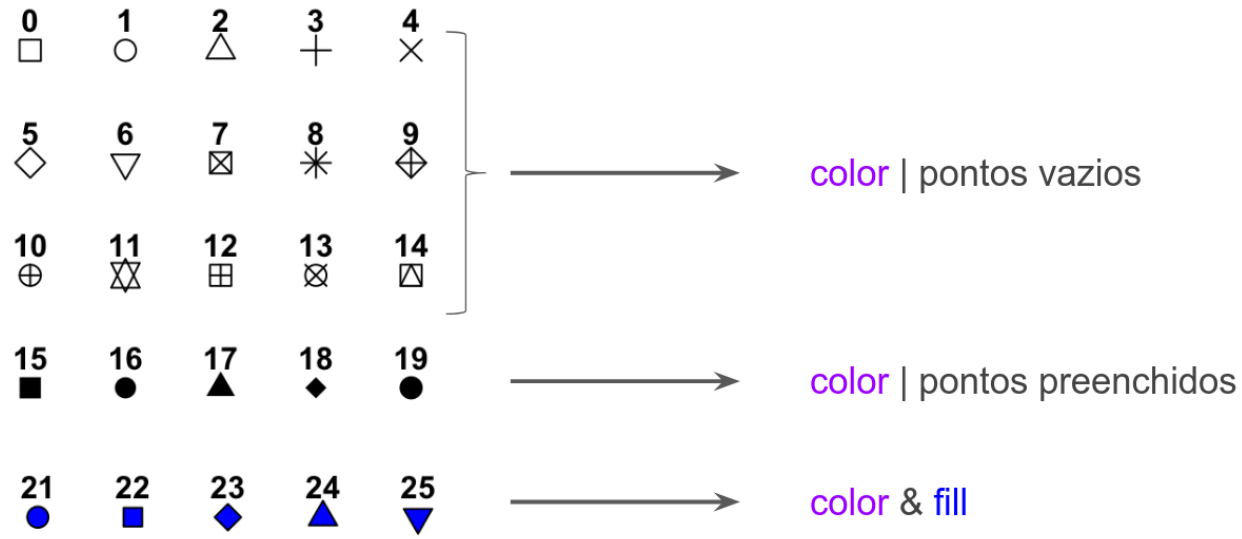


Figura 6.34: Tipos de símbolos disponíveis para representar pontos.

Assim, é possível controlar cores, formas e preenchimento combinado os argumentos `shape`, `fill` e `color` com a função `scale_manual()`. É importante notar que para os símbolos entre 15 e 20 só podemos controlar o argumento cor, enquanto os símbolos entre 21 e 25 podemos controlar a cor e o preenchimento (Figura 6.35).

```
## Formato e tamanho
ggplot(data = penguins, aes(x = comprimento_bico,
```

```
    y = comprimento_nadadeira)) +
    geom_point(shape = 1, size = 4)

## Formato e tamanho para espécies
ggplot(data = penguins, aes(x = comprimento_bico,
    y = comprimento_nadadeira, color = especies)) +
    geom_point(shape = 19, size = 4)

## Formato e tamanho e cor
ggplot(data = penguins, aes(x = comprimento_bico,
    y = comprimento_nadadeira, fill = especies)) +
    geom_point(shape = 21, size = 4, color = "black")
```

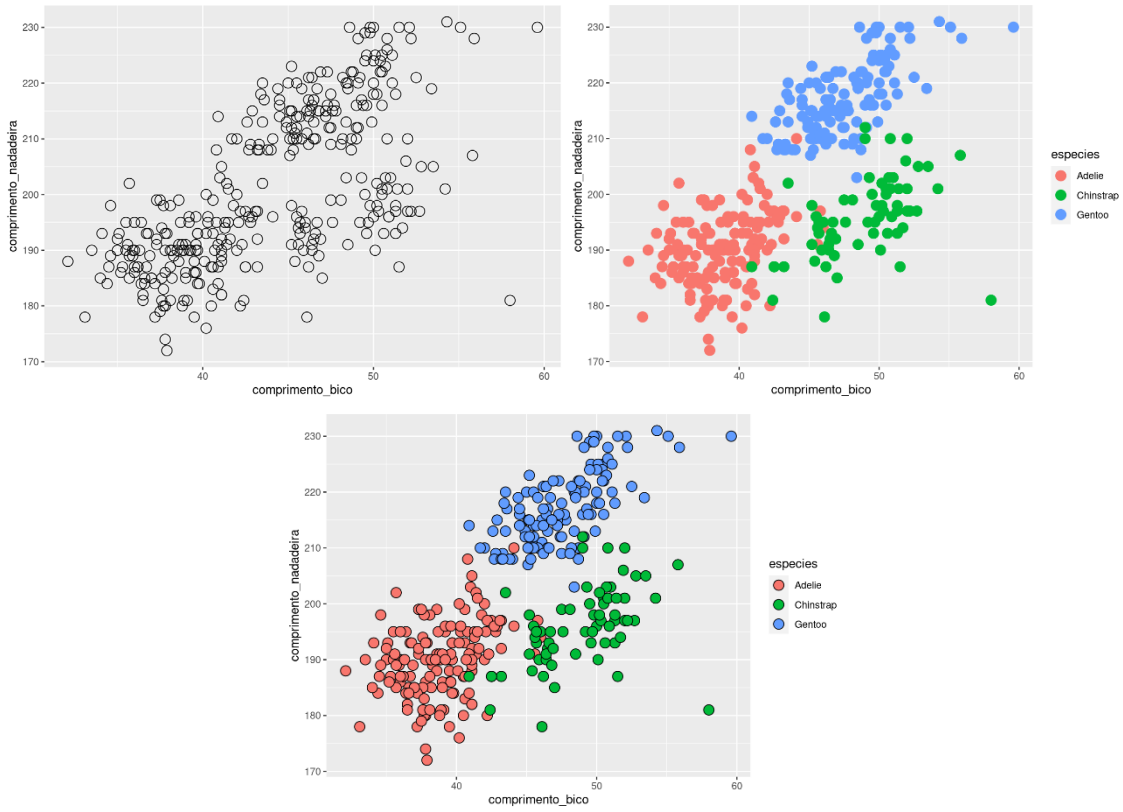


Figura 6.35: Gráfico de dispersão relacionando as variáveis `comprimento_bico` e `comprimento_nadadeira` para cada espécie de pinguim, com variações na cor e no preenchimento dos pontos.

Definindo linhas de ajuste

Quando usamos modelos estatísticos como, por exemplo, `lm()`, `glm()`, `gam()`, entre outros (veja mais nos Capítulos 7 e 8), podemos utilizar os valores preditos para demonstrar a relação entre as variáveis X e Y. No `ggplot2`, a função `geom_smooth()` faz esse ajuste com certa simplicidade (Figura 6.36).

```
## Linha de ajuste
ggplot(data = penguins, aes(x = comprimento_bico,
    y = comprimento_nadadeira)) +
```

```
    geom_point(shape = 20, size = 4, color = "black") +
    geom_smooth(method = lm)
```

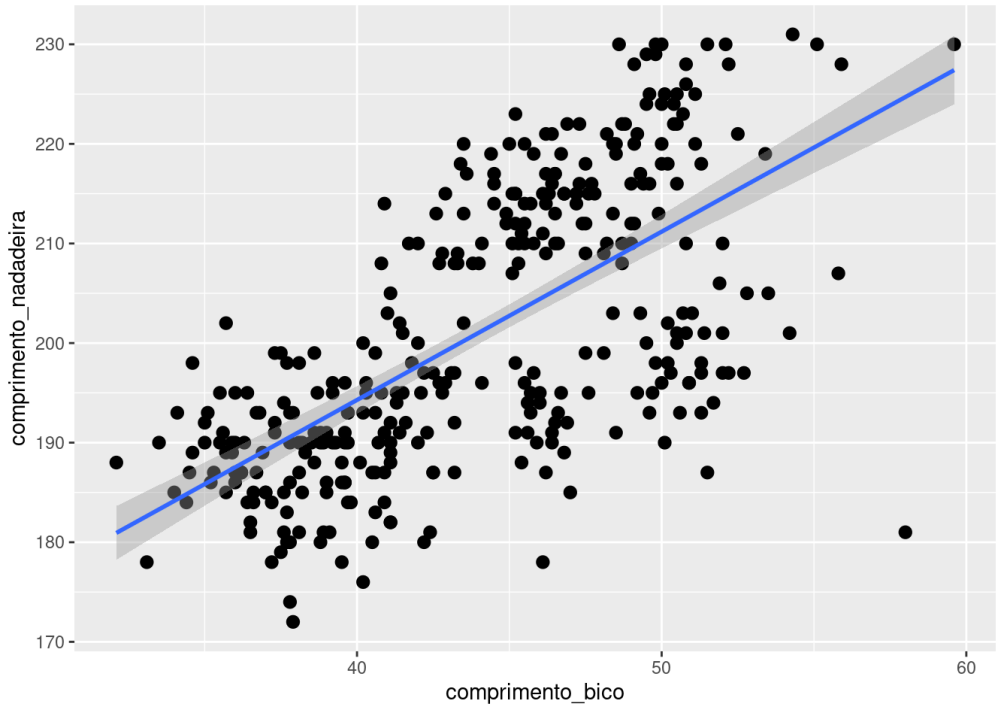


Figura 6.36: Gráfico de dispersão relacionando as variáveis `comprimento_bico` e `comprimento_nadadeira` para cada espécie de pinguim, com linhas de ajustes estatísticos.

Ajustes finos (versão personalizada)

```
## Gráfico de dispersão exemplo
ggplot(data = penguins, aes(x = comprimento_bico,
                            y = comprimento_nadadeira,
                            color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 16) +
    labs(x = "Comprimento do bico (mm)",
         y = "Comprimento da nadadeira (mm)",
         color = "Espécies", shape = "Espécies")
```

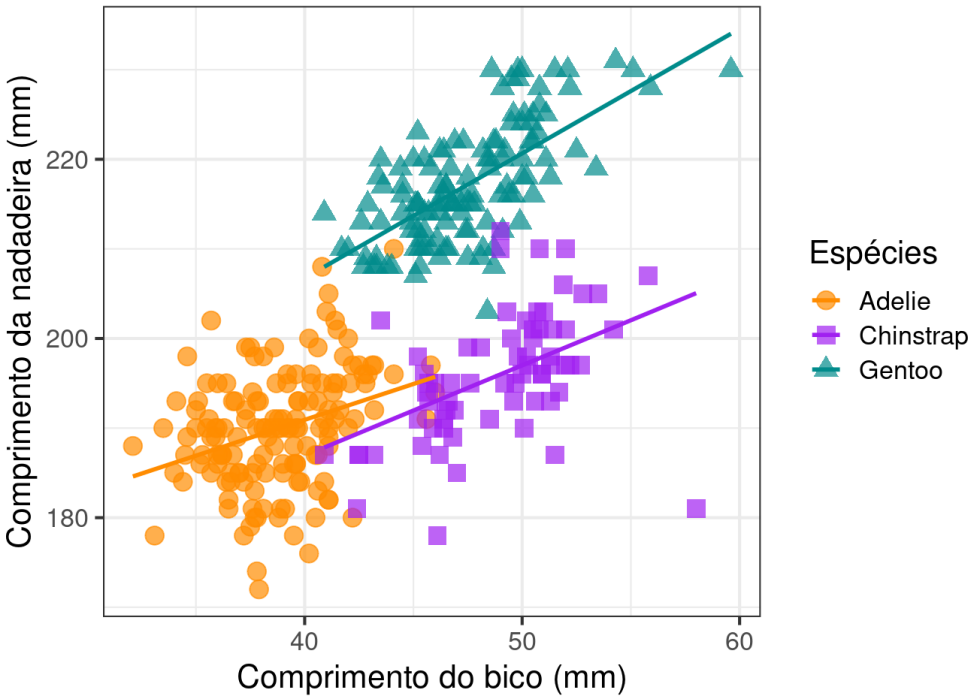


Figura 6.37: Gráfico de dispersão relacionando as variáveis `comprimento_bico` e `comprimento_nadadeira` para cada espécie de pinguim, com ajustes finos.

Além disso, podemos relacionar dados não tão usuais. Recomendamos a leitura do artigo de Matejka & Fitzmaurice (2017) que apresenta as armadilhas típicas que dados podem gerar quando evitamos de visualizá-los previamente (Figura 6.38).

```
## Gráfico do dinossauro
datasaurus_dozen %>%
    dplyr::filter(dataset == "dino") %>%
    ggplot() +
    aes(x = x, y = y) +
    geom_point(colour = "black", fill = "black",
               size = 4, alpha = .7, shape = 21) +
    theme_bw() +
    theme(axis.title = element_text(size = 24),
          axis.text.x = element_text(size = 20),
          axis.text.y = element_text(size = 20))
```

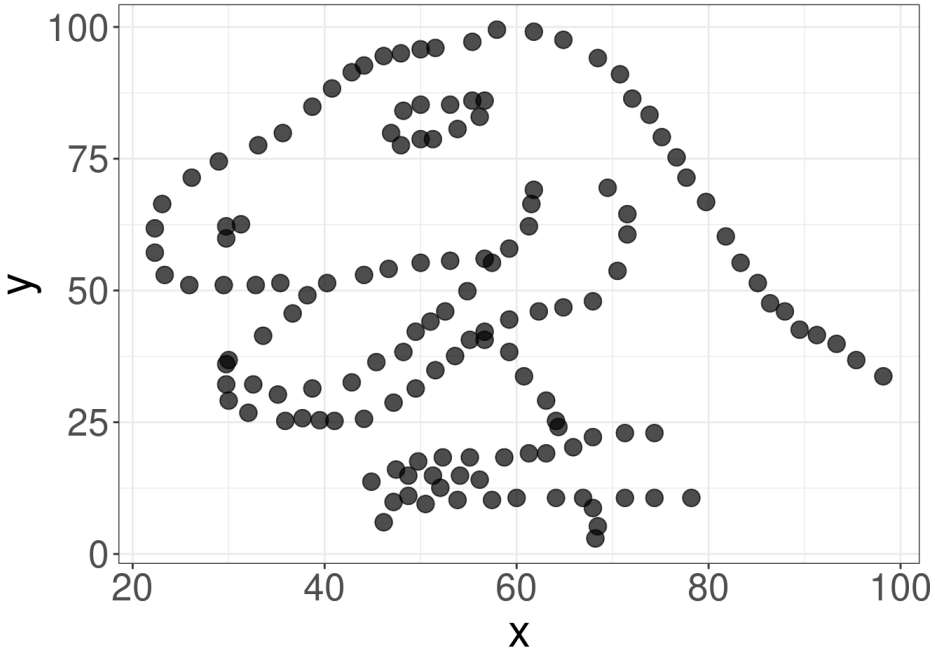


Figura 6.38: Gráfico de dispersão relacionando as variáveis `x` e `y` formando um dinossauro.

6.4.8 Visualização de múltiplos gráficos pareados

Muitas vezes precisamos plotar a relação de mais de uma variável, e muitas vezes essas variáveis são de mais de um tipo (contínua, categórica, etc.). O gráfico mais indicado nesses casos é o gráfico pareado (Emerson et al. 2013) que nos auxilia a ter uma visão geral do conjunto de dados e de suas interrelações. Esse gráfico também é chamado de *pairs plot* ou correlograma.

Gráfico pareado com variáveis contínuas

A função `ggpairs()` do pacote `GGally` permite criar múltiplos gráficos pareados comparando as variáveis contínuas no seu conjunto de dados. Além de demonstrar gráficos de dispersão de cada par de variáveis, ela apresenta gráficos de densidade de cada variável individualmente e, além disso, os valores de correlação entre os pares analisados com ou sem uma potencial variável categórica (neste caso, `especies`) (Figura 6.39).

```
## Gráfico pareado com variáveis contínuas
penguins %>%
    dplyr::select(massa_corporal, comprimento_bico,
                  profundidade_bico, comprimento_nadadeira) %>%
    GGally::ggpairs(aes(color = penguins$especies)) +
    scale_colour_manual(values = c("darkorange", "purple", "cyan4")) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw()
```

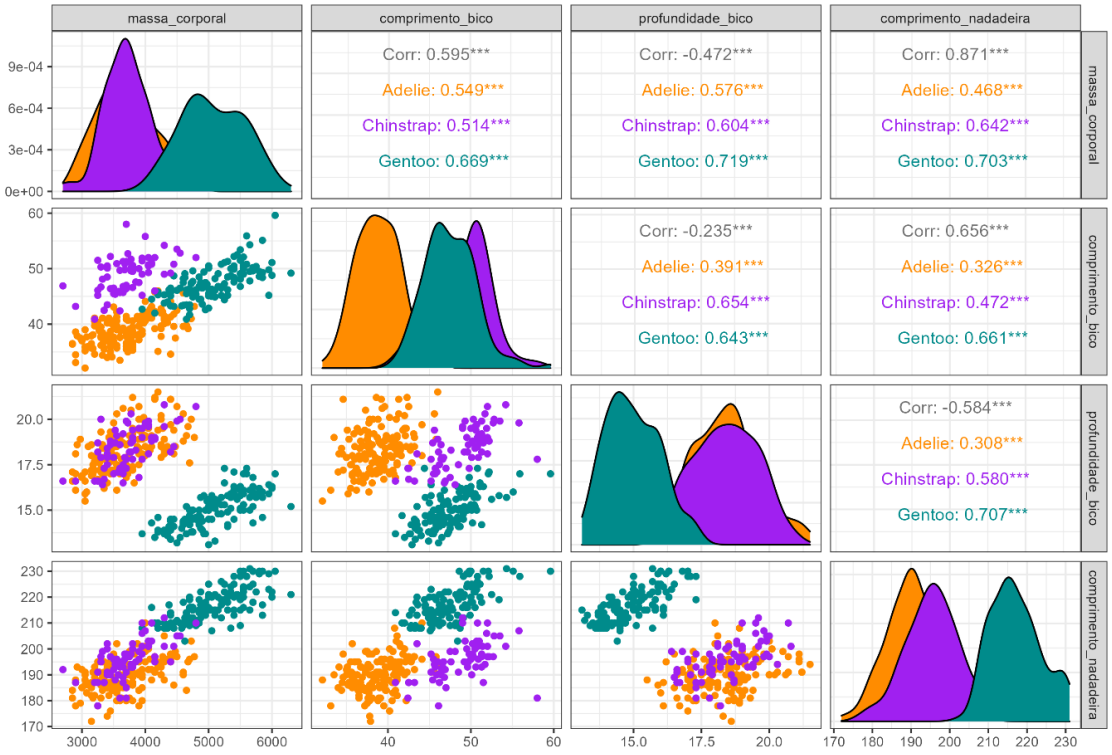


Figura 6.39: Gráfico pareado relacionando várias variáveis contínuas para as espécies de pinguins.

Gráfico pareado com vários tipos de variáveis

Como alternativa, a função `ggpairs()` permite também incluir variáveis categóricas nas comparações. Neste caso, ela reconhece o tipo de gráfico (boxplot, dispersão, etc.) a partir dos modos das variáveis (Figura 6.40).

```
## Gráfico pareado com variáveis contínuas e categóricas
penguins %>%
    dplyr::select(sexo, massa_corporal, comprimento_bico,
                  profundidade_bico, comprimento_nadadeira) %>%
    GGally::ggpairs(aes(color = sexo)) +
    scale_colour_manual(values = c("darkorange", "purple", "cyan4")) +
    scale_fill_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw()
```

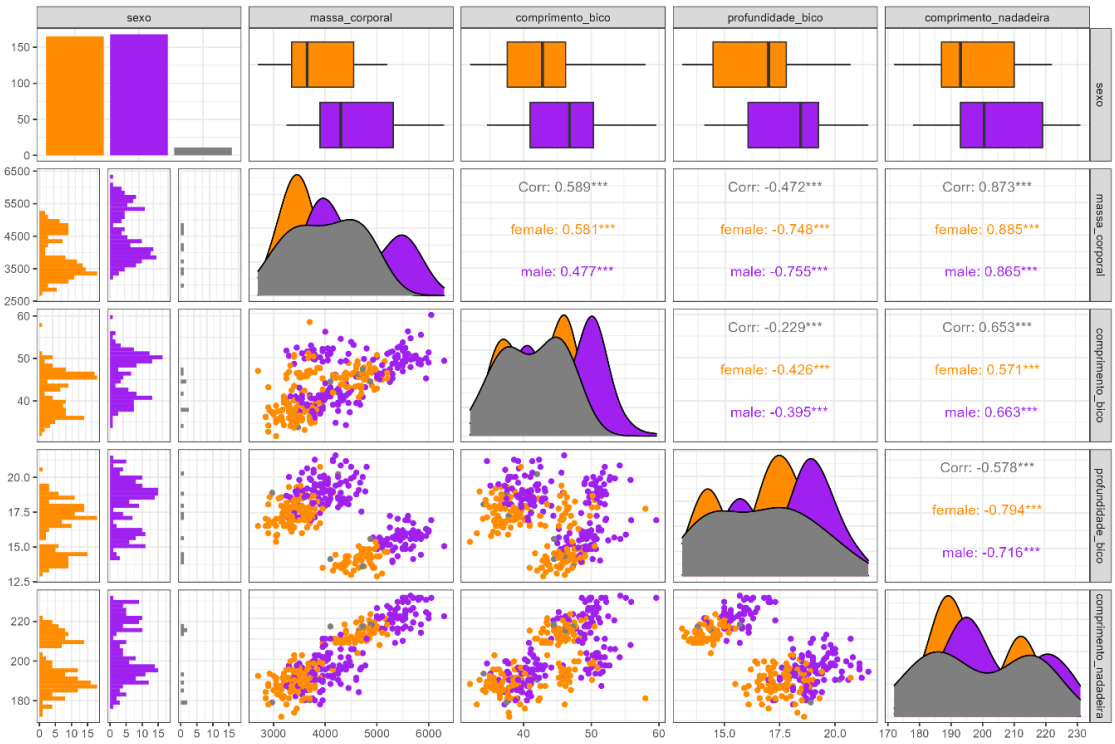


Figura 6.40: Gráfico pareado relacionando várias variáveis contínuas e categóricas para as espécies de pinguins.

6.4.9 Erros comuns dos usuários do `ggplot2` e como evitá-los

Abaixo, apresentamos uma lista não exaustiva dos erros mais comuns que cometemos (e vimos muitos usuários cometerem) ao fazer gráficos no ggplot2.

- Utilizar ajuste manual nas funções `scale_shape_manual()`, `scale_color_manual()` ou `scale_fill_manual()` sem indicar no argumento `aes()` as variáveis que devem definir cada um desses elementos gráficos
- Utilizar ajuste manual na função `scale_size_manual()` indicando uma variável categórica em vez de numérica
- Número de cores indicadas como valores no `scale_fill_manual()` ou `scale_color_manual()`: ao definir as cores de maneira personalizada (ou seja, não usando o padrão da função) é muito comum utilizarmos o número de cores usados por algum tutorial ou livro. Com frequência, o exemplo seguido e seus dados não possuem o mesmo número de cores dos nossos dados. Deste modo, você pode usar códigos no R para ajudar a quantificar o número de cores necessárias. Por exemplo, para os dados `penguins`, o código a seguir indica o número de cores necessárias: `length(levels(penguins\$especies))`. Assim, será necessário indicar três cores diferentes dentro da função `scale_*()`.
- Função `geom_smooth()`: como explicado acima, a função `geom_smooth()` é muito útil (e simples) para gerar as linhas de ajuste (*best fit*) típicas de modelos lineares e não lineares. Porém, fique alerta ao usar, por exemplo, `geom_smooth(method = lm)`, o modelo linear utilizado para testar sua predição foi o `lm()`. Se tiver utilizado `glm()`ou `gam()` o ajuste deve ser produzido a partir desses modelos.
- Uso incorreto do modo das variáveis: neste caso, o usuário utilizar uma variável numérica (por exemplo, 1, 2 e 3) como variável categórica. Neste caso, é preciso transformar a variável numérica em variável categóricas (antes de fazer o ggplot2 ou dentro do `aes()`).

Veja exemplos (Figura 6.41).

```
## Figura incorreta, sem a transformação da variável ano
penguins %>%
    ggplot(aes(x = ano, y = comprimento_bico)) +
    geom_boxplot() +
    theme_bw() +
    labs(title = "Figura incorreta")

## Figura correta, com transformação interna da variável ano
penguins %>%
    ggplot(aes(x = factor(ano), y = comprimento_bico)) +
    geom_boxplot() +
    theme_bw() +
    labs(title = "Figura correta com transformação interna")

## Figura correta, com transformação prévia da variável ano
penguins %>%
    mutate(ano_f = as.factor(ano)) %>%
    ggplot(aes(x = ano_f, y = comprimento_bico)) +
    geom_boxplot() +
    theme_bw() +
    labs(title = "Figura correta com transformação prévia")
```

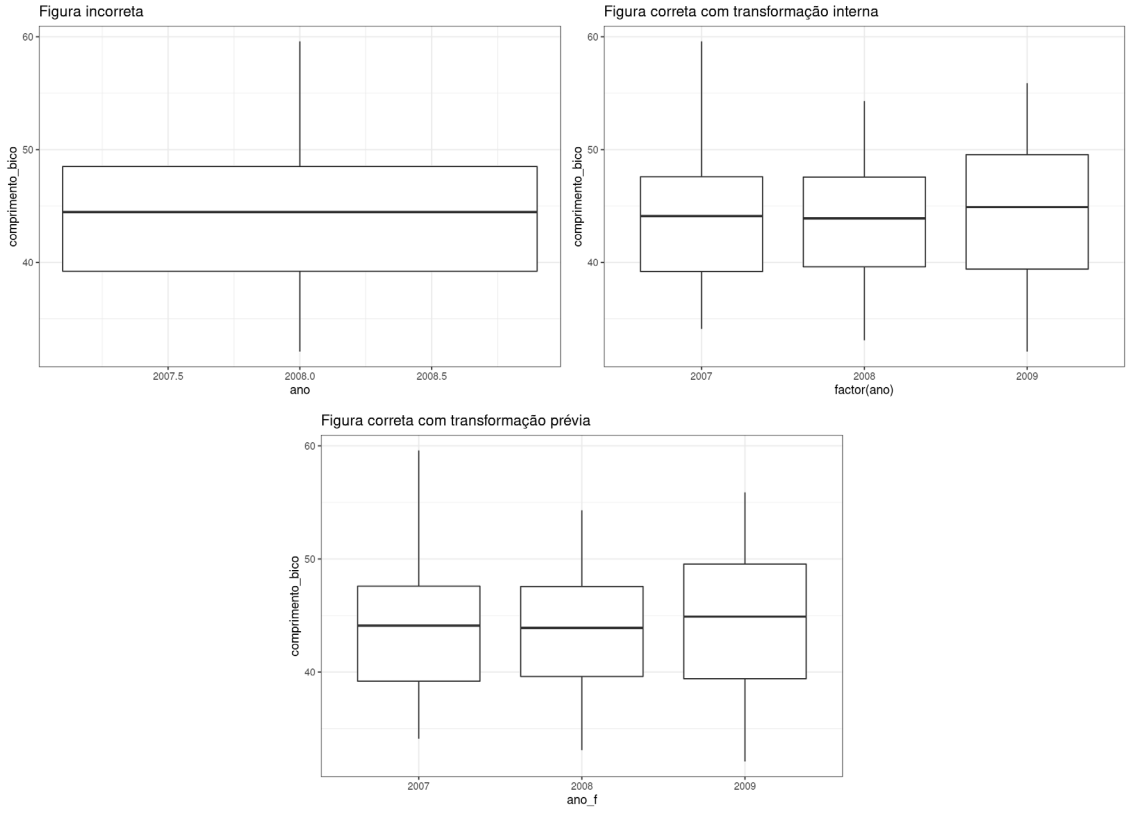


Figura 6.41: Gráfico mostrando erros comuns na criação de gráficos no pacote ggplot2.

6.5 Finalização de gráficos para publicação

6.5.1 Posição, cores e fonte da legenda

É possível controlar a posição, cores e fonte da legenda em diversos locais com alguns argumentos dentro da função `theme()`:

- `legend.position`: controla a posição na área do gráfico: `top`, `right`, `bottom`, `left` ou `none`. Além disso, é possível inserir a legenda internamente no gráfico indicando as posições nos eixos X e Y
- `legend.box`: determina as características do retângulo onde a legenda é inserida: `legend.box.background` (combinado com `element_rect()`) e `legend.box.margin` (combinado com `margin()`)
- `legend.text`: controla a cor e tamanho da legenda (as duas informações devem ser inseridas dentro da função `element_text()`)
- `legend.title`: personaliza a cor e tamanho da legenda também dentro da função `element_text()`

Vejamos alguns exemplos (Figura 6.42).

```
## Legenda acima
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 14) +
    theme(legend.position = "top") +
    labs(title = "Legenda acima do gráfico",
         x = "Comprimento do bico (mm)",
         y = "Profundidade do bico (mm)", color = "Espécies",
         shape = "Espécies")

## Leganda abaixo
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 14) +
    theme(legend.position = "bottom") +
    labs(title = "Legenda abaixo do gráfico",
         x = "Comprimento do bico (mm)",
```

```
         y = "Profundidade do bico (mm)", color = "Espécies",
         shape = "Espécies")

## Sem legenda
ggplot(data = penguins,
       aes(x = comprimento_bico,  y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 14) +
    theme(legend.position = "none") +
    labs(title = "Sem legenda", x = "Comprimento do bico (mm)",
         y = "Profundidade do bico (mm)", color = "Espécies",
         shape = "Espécies")

## Legenda personalizada
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 14) +
    theme(legend.position = "right",
          legend.text = element_text(size = 14, colour = "red"),
          legend.title = element_text(face = "bold"),
          legend.box.background = element_rect(color="red", size=2),
          legend.margin = margin(6, 6, 6, 6)) +
    labs(title = "Legenda personalizada", x = "Comprimento do bico (mm)",
         y = "Profundidade do bico (mm)", color = "Espécies",
         shape = "Espécies")

## Legenda interna
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 16) +
    theme(legend.position = c(.2, .8),
          legend.title = element_blank(),
          legend.key =  element_blank(),
          legend.background = element_blank(),
```

```
        legend.text = element_text(size = 12, face = "bold")) +
  labs(title = "Legenda interna", x = "Comprimento do bico (mm)",
       y = "Profundidade do bico (mm)", color = "Espécies",
       shape = "Espécies")
```

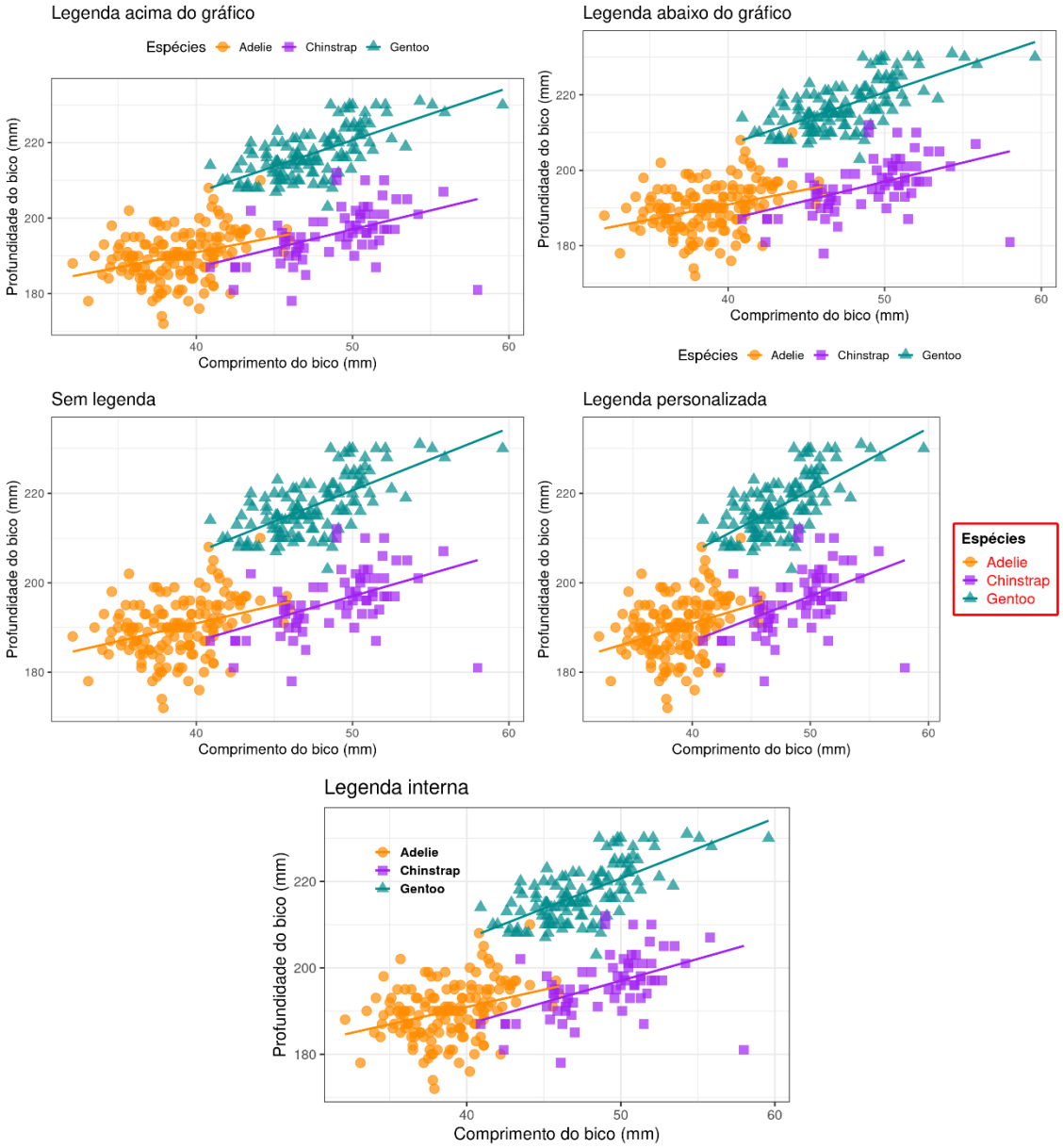


Figura 6.42: Gráfico mostrando a finalização de gráficos para publicação em relação à posição, cores e fonte da legenda no pacote ggplot2.

6.5.2 Elementos gráficos: eixo, fonte, gride

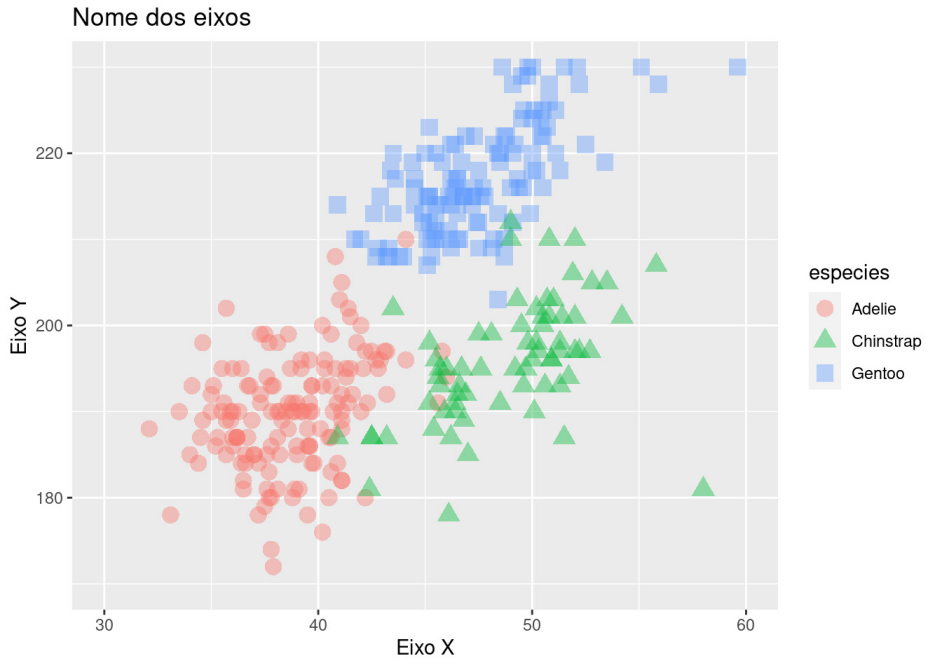
O gráfico padronizado (sem edição extra) geralmente não traz elementos mínimos para publicação em revistas, livros ou periódicos. Além do controle da posição, cor e tamanho da legenda, é fundamental personalizar os seguintes elementos: eixo, fonte e gride.

- Eixos
- Variação: define limites mínimos e máximos para os eixos X (`xlim()`) e Y (`ylim()`)
- Intervalo: define o valor intervalo entre os números dos eixos X e Y

Vejamos mais alguns exemplos (Figura 6.43).

```
## Nome dos eixos
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .4) +
    ylim(170, 230) +
    xlim(30, 60) +
    labs(title = "Nome dos eixos", x = "Eixo X", y = "Eixo Y")

## Intervalo dos eixos
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .4) +
    scale_x_continuous(limits = c(30, 60), breaks = seq(30, 60, 2)) +
    labs(title = "Intervalo dos eixos", x = "Eixo X", y = "Eixo Y")
```



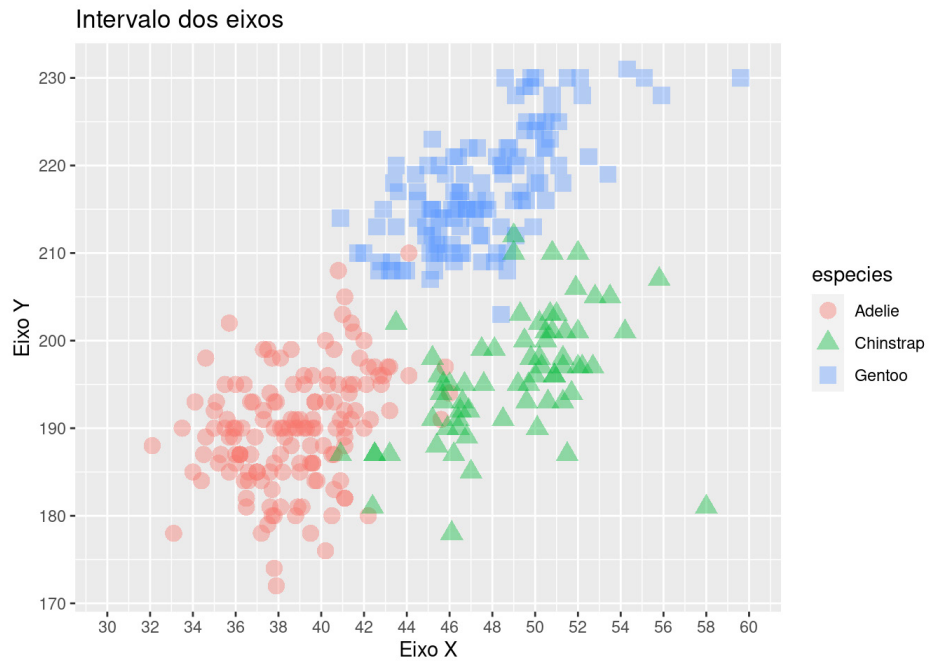


Figura 6.43: Gráficos mostrando a finalização de gráficos para publicação em relação aos elementos gráficos de eixo, fonte e gride no pacote ggplot2.

- Fonte dos eixos X e Y
- Tipo
- Tamanho
- Cor
- Face (itálico, negrito, etc.)
- Ângulo

Vejamos outros exemplos com mudanças nos eixos (Figura 6.44).

```
## Cor e fonte dos eixos
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .4) +
    theme(axis.title.x = element_text(face = "bold", size = 20,
          colour = "cyan4"),
          axis.text.x = element_text(size = 14),
          axis.title.y = element_text(face = "bold", size = 20,
          colour = "cyan4"),
          axis.text.y = element_text(size = 14)) +
    labs(title = "Cor e fonte dos eixos", x = "Eixo X", y = "Eixo Y")

## Intervalo e ângulos do texto dos eixos
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .4) +
    scale_x_continuous(limits = c(20, 60), breaks = seq(20, 60, 2)) +
```

```
    theme(axis.title.x = element_text(face = "bold", size = 20,
          colour = "cyan4"),
          axis.text.x = element_text(size = 14, angle = 45),
          axis.title.y = element_text(face = "bold", size = 20,
          colour = "cyan4"),
          axis.text.y = element_text(size = 14)) +
    labs(title = "Intervalo e ângulos do texto dos eixos",
          x = "Eixo X", y = "Eixo Y")
```

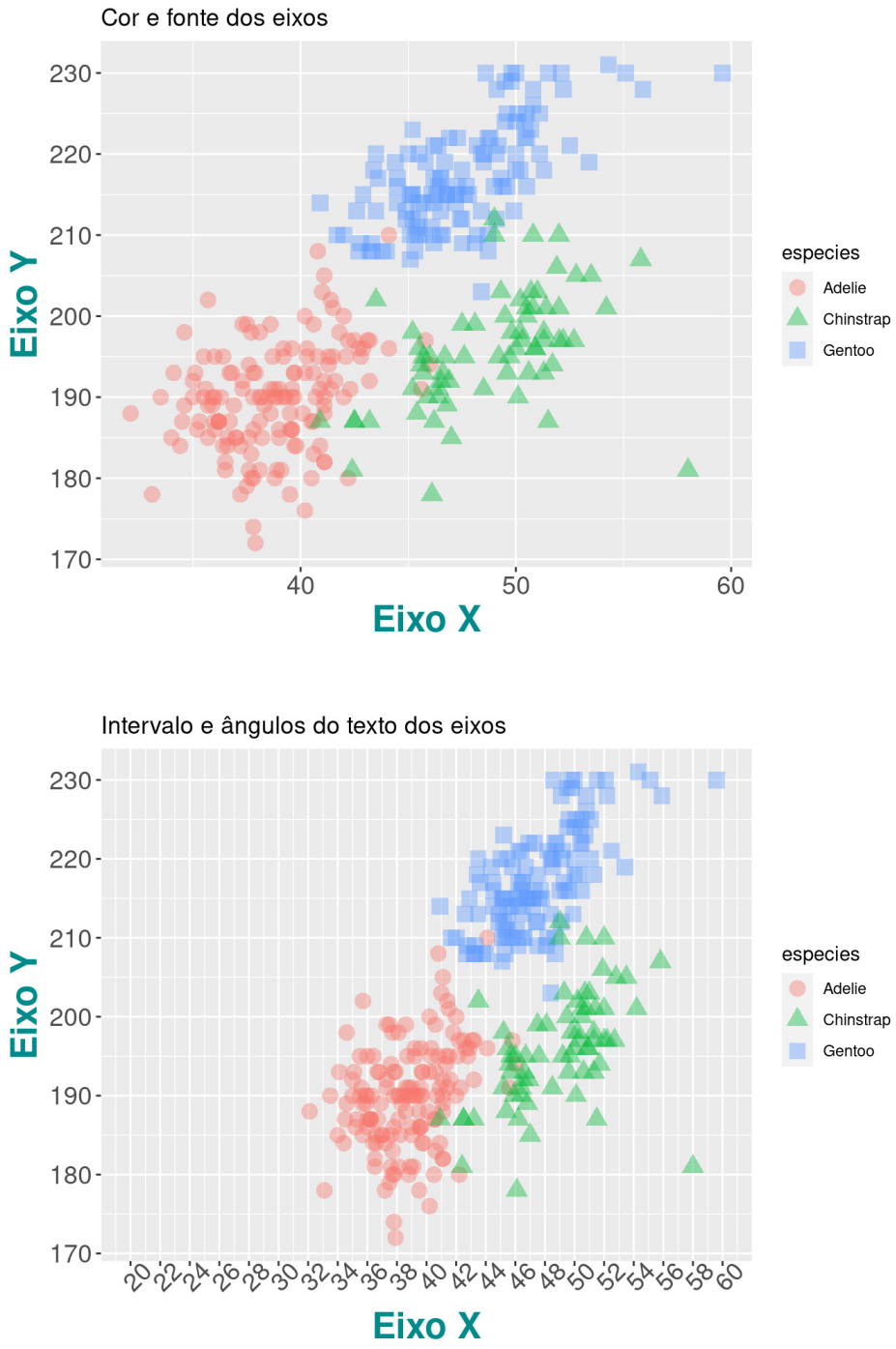


Figura 6.44: Gráficos mostrando a finalização de gráficos para publicação em relação aos elementos gráficos de eixo, fonte e gride no pacote `ggplot2`.

- Gride
- Linhas de grade principais (`panel.grid.major`)
- Linhas de grade secundárias (`panel.grid.minor`)
- Borda do gráfico (`panel.border`)

Vejamos outros exemplos com mudanças ao gride (Figura 6.45).

```
## Linhas de grade principais
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .4) +
    scale_x_continuous(limits = c(30, 60), breaks = seq(30, 60, 5)) +
    theme(axis.title.x = element_text(face = "bold", size = 16),
          axis.text.x = element_text(size = 12),
          axis.title.y = element_text(face = "bold", size = 16),
          axis.text.y = element_text(size = 12),
          panel.grid.minor = element_blank()) +
    labs(title = "Linhas de grade principais", x = "Eixo X",
         y = "Eixo Y")

## Retirar linhas de grade
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .4) +
    scale_x_continuous(limits = c(30, 60), breaks = seq(30, 60, 5)) +
    theme(axis.title.x = element_text(face = "bold", size = 16),
          axis.text.x = element_text(size = 12),
          axis.title.y = element_text(face = "bold", size = 16),
          axis.text.y = element_text(size = 12),
          panel.grid.minor = element_blank(),
          panel.grid.major = element_blank()) +
    labs(title = "Retirar linhas de grade", x = "Eixo X", y = "Eixo Y")

## Borda do gráfico
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = 0.5) +
    scale_x_continuous(limits = c(30, 60), breaks = seq(30, 60, 5)) +
    theme(axis.title.x = element_text(face = "bold", size = 16),
          axis.text.x = element_text(size = 12),
          axis.title.y = element_text(face = "bold", size = 16),
          axis.text.y = element_text(size = 12),
          panel.grid.minor = element_blank(),
          panel.grid.major = element_blank(),
          panel.border = element_rect(size = 2, colour = "black",
```

```
                                        fill = NA)) +
    labs(title = "Borda do gráfico", x = "Eixo X", y = "Eixo Y")

## Borda do gráfico
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .4) +
    scale_x_continuous(limits = c(30, 60), breaks = seq(30, 60, 5)) +
    theme(axis.title.x = element_text(face = "bold", size = 16),
          axis.text.x = element_text(size = 12),
          axis.title.y = element_text(face = "bold", size = 16),
          axis.text.y = element_text(size = 12),
          panel.grid.minor = element_blank(),
          panel.grid.major = element_blank(),
          axis.line = element_line(size = 1)) +
    labs(title = "Borda do gráfico", x = "Eixo X", y = "Eixo Y")
```

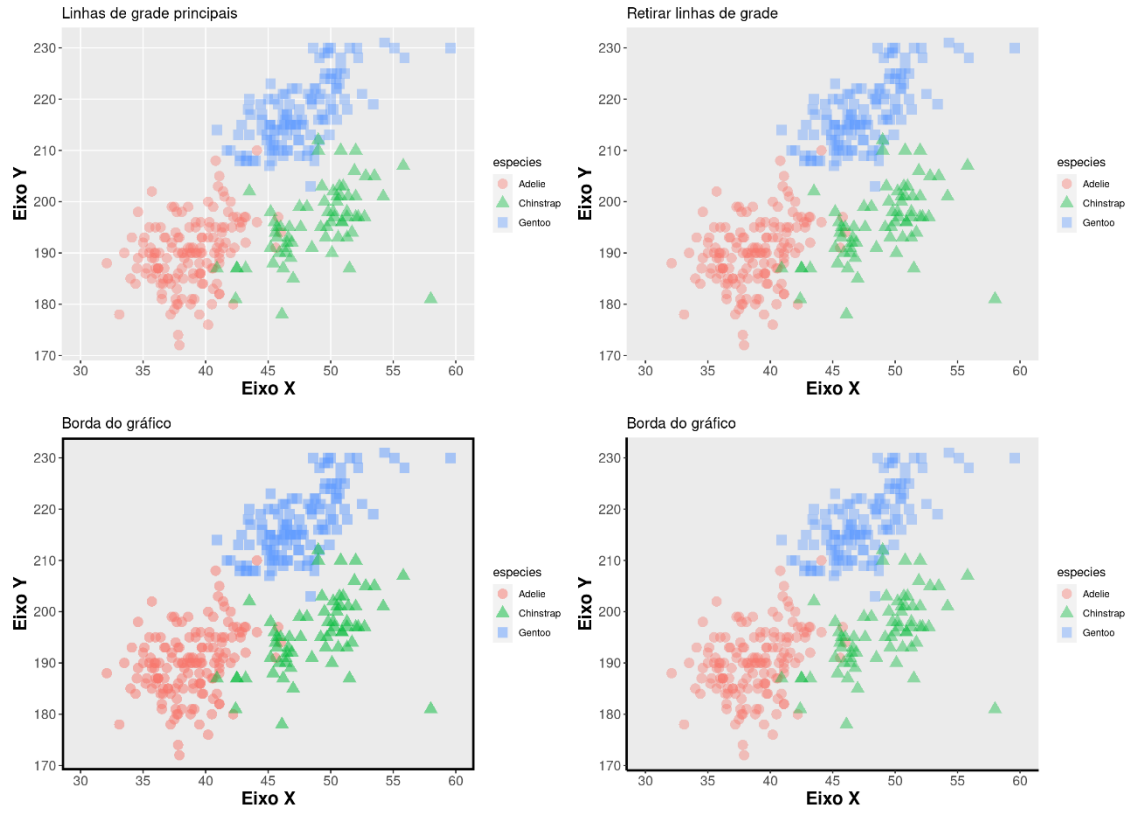


Figura 6.45: Gráficos mostrando a finalização de gráficos para publicação em relação ao gride no pacote `ggplot2`.

6.5.3 Temas personalizados `ggtheme()`

Existem vários temas criados dentro do universo `ggtheme()` que podem facilitar a escolha de um modelo com ótima qualidade para publicação. Abaixo, demonstramos os modelos mais utilizados (Figura 6.46).

```
## theme_gray
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_gray(base_size = 16) +
    labs(title = "theme_gray()", x = "Comprimento do bico (mm)",
         y = "Profundidade do bico (mm)", color = "Espécies",
         shape = "Espécies")
## theme_bw()
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_bw(base_size = 16) +
    labs(title = "theme_bw()", x = "Comprimento do bico (mm)",
         y = "Profundidade do bico (mm)", color = "Espécies",
         shape = "Espécies")
## theme_classic()
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme_classic(base_size = 16) +
    labs(title = "theme_classic()", x = "Comprimento do bico (mm)",
         y = "Profundidade do bico (mm)", color = "Espécies",
         shape = "Espécies")
```

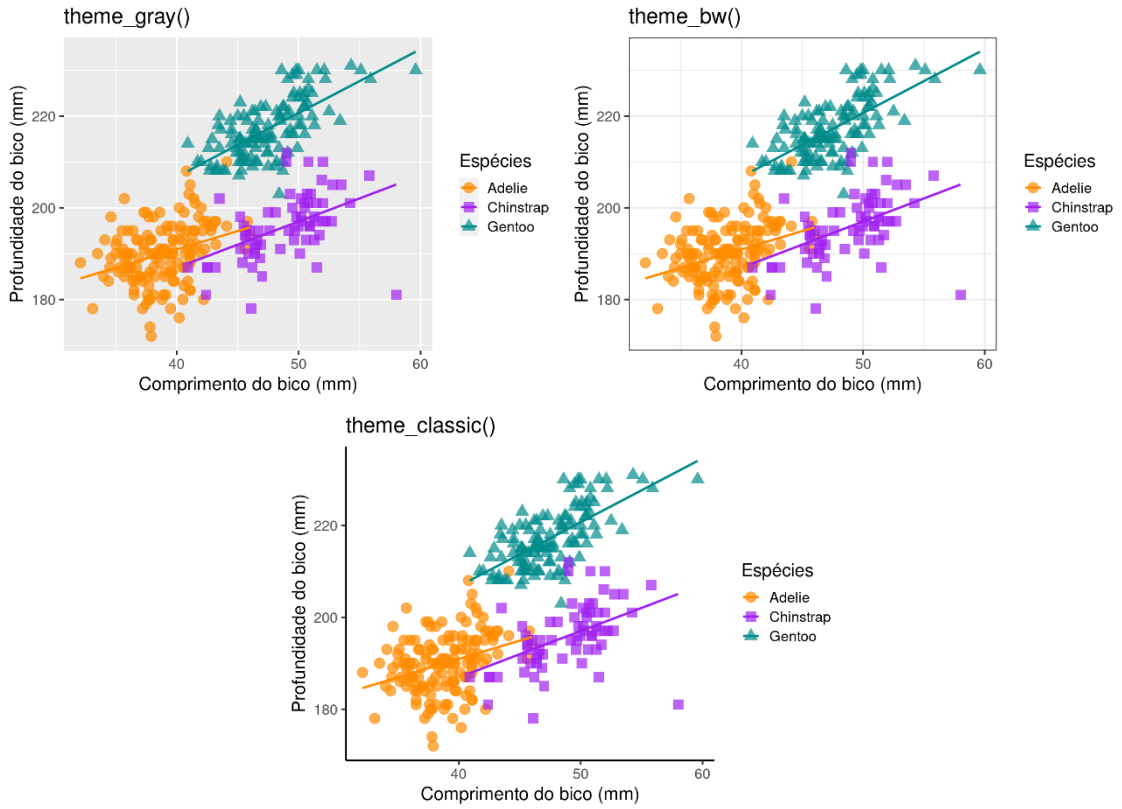


Figura 6.46: Gráficos mostrando a finalização de gráficos para publicação em relação ao `ggtheme()` no pacote `ggplot2`.

6.5.4 Criando seu próprio `theme_custom()`

Por fim, é possível criar um tema personalizado como uma função. Assim, o usuário pode controlar todos os elementos gráficos em um único código. O maior benefício de personalizar uma função é que não será necessário fazer os ajustes finos em todos os gráficos que tiver construindo, o que pode representar grande economia de tempo e linhas de código (Figura 6.47).

```
## Criar uma função com os ajustes finos
tema_personalizado <- function(){

    # Defina uma fonte
    font <- "Times" # Digite names(pdfFonts()) no console do R para ver a
lista de fontes disponíveis

    theme(

        # Defina elementos do gride
        panel.grid.major = element_line(colour = "#d3d3d3"),
        panel.grid.minor = element_blank(),
        axis.ticks = element_blank(),
```

```
        panel.border = element_rect(colour = "black", fill = NA,
                                    size = .5),

        # Defina elementos textuais
        # Título
        plot.title = element_text(
            family = font,            # Fonte
            size = 20,                # Tamanho da fonte
            face = 'bold',            # Tipo de fonte
            hjust = 0,                # Alinhamento horizontal
            vjust = 2),               # Alinhamento vertical

        # Subtítulo
        plot.subtitle = element_text(
            family = font,            # Fonte
            size = 14),               # Tamanho da fonte

        # Rúbrica
        plot.caption = element_text(
            family = font,            # Fonte
            size = 10,                # Tamanho da fonte
            hjust = 1),               # Alinhamento horizontal

        # Título dos eixos
        axis.title = element_text(
            family = font,            # Fonte
            size = 14),               # Tamanho da fonte

        # Texto dos eixos
        axis.text = element_text(
            family = font,            # Fonte
            size = 14)              # Tamanho da fonte

    )}

## Gráfico usando a função de tema criada
ggplot(data = penguins,
       aes(x = comprimento_bico, y = comprimento_nadadeira,
           color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    tema_personalizado() +
    labs(title = "Tema personalizado", x = "Comprimento do bico (mm)",
         y = "Profundidade do bico (mm)", color = "Espécies",
         shape = "Espécies", caption = "Fonte = palmerpinguins")
```

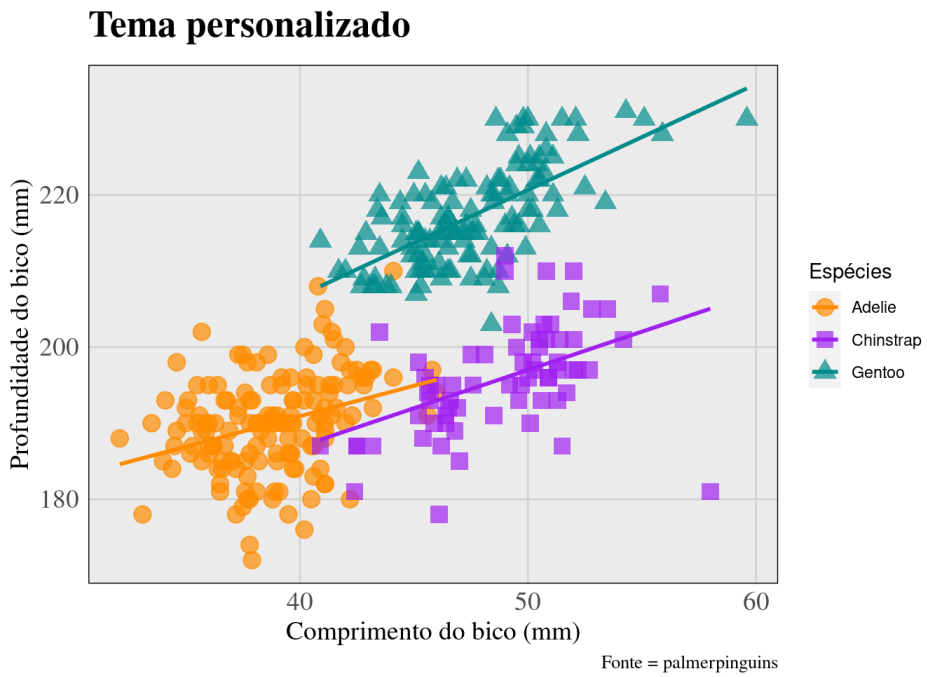


Figura 6.47: Gráficos mostrando a finalização de gráficos para publicação em relação à criação de um tema usando o `theme_custom()` no pacote `ggplot2`.

6.5.5 Exportando gráficos com alta qualidade com a função `ggsave()`

O último passo para construir gráficos com qualidade de publicação é exportar em um formato específico, como .png, .pdf ou .svg (entre outros). A função `ggsave()` não só permite que você tenha o controle sobre o formato, mas também sobre a qualidade e tamanho desejados com os seguintes argumentos:

- `width`: largura do gráfico
- `height`: altura do gráfico
- `units`: unidade de medida (cm, mm) do gráfico para definir largura e tamanho
- `dpi`: resolução ou qualidade da imagem, medida em pontos por polegada (*dots per inch*) (padrão = 300)

Vejamos um último exemplo (Figura 6.48).

```
## Gráfico
g1 <- ggplot(data = penguins,
             aes(x = comprimento_bico, y = comprimento_nadadeira,
                 color = especies, shape = especies)) +
    geom_point(size = 4, alpha = .7) +
    geom_smooth(method = "lm", se = FALSE) +
    scale_shape_manual(values = c(19, 15, 17)) +
    scale_color_manual(values = c("darkorange", "purple", "cyan4")) +
    theme(legend.position = c(.1, .1),
          legend.title = element_blank(),
          legend.key =  element_blank(),
          legend.background = element_blank()) +
    tema_personalizado() +
```

```
    labs(x = "Comprimento do bico (mm)", y = "Profundidade do bico (mm)",
         color = "Espécies", shape = "Espécies")
g1
```

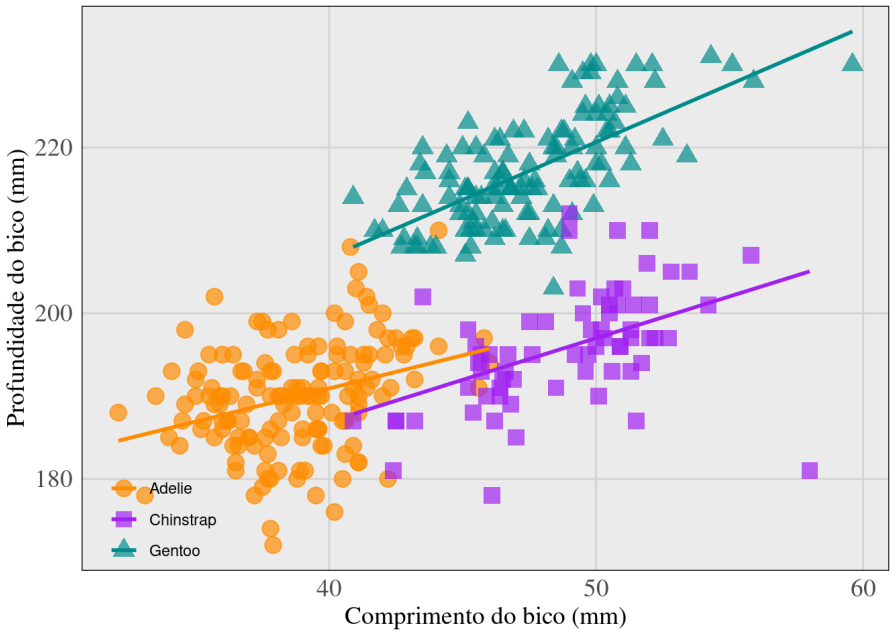


Figura 6.48: Gráfico criado para a exportação no pacote `ggplot2`.

Podemos exportar esse gráfico de diferentes formas.

```
## Exportar no formato PDF
ggsave(filename = "g1.pdf",
       plot = g1,
       width = 15,
       height = 15,
       dpi =300,
       units = "cm")

## Exportar no formato PNG
ggsave(filename = "g1.png",
       plot = g1,
       width = 15,
       height = 15,
       dpi =300,
       units = "cm")

## Exportar no formato SVG
ggsave(filename = "g1.svg",
       plot = g1,
       width = 15,
       height = 15,
       dpi =300,
       units = "cm")
```

6.6 Para se aprofundar

Listamos abaixo livros e links que recomendamos para seguir com sua aprendizagem em R e gráficos.

6.6.1 Livros

Recomendamos aos interessados(as) os livros: i) Chang (2018) *R Graphics Cookbook*, ii) Healy (2018) *Data Visualization: a practical introduction*, iii) Rahlf (2019) *Data Visualisation with R: 111 Examples*, iv) Sievert (2020) *Interactive web-based data visualization with R, plotly, and shiny*, v) Wickham (2016) *ggplot2: elegant graphics for data analysis*, vi) Wilke (2019) *Fundamentals of Data Visualization*, vii) Wilkinson e colaboradores (2005) *The Grammar of Graphics*.

6.6.2 Links

Existem centenas de ferramentas online para aprender visualização de dados e `ggplot2`. Tais ferramentas são bastante variadas, incluindo desde a escolha de tipos de gráficos à seleção de cores. Dentre elas, indicamos os seguintes links em inglês:

Visualização

- *Data Visualization with R* - Rob Kabacoff
- *The Data Visualisation Catalogue*
- *The R Graph Gallery*
- *From Data to Viz*
- *Data Viz Project*
- *QuickChart*
- *Chart.js*
- drawdata.xyz

Seleção de cores (paletas)

- *Color Brewer: color advice*
- *Wes Anderson Palletes*
- *Choosing colors for data visualization*

6.7 Exercícios

6.1 Utilizando o banco de dados `penguins` compare o comprimento do bico entre as diferentes espécies de pinguins. Utilize um gráfico de caixa (boxplot) para ilustrar a variação intraespecífica e possíveis outliers nos dados. Para melhorar o seu gráfico, lembre-se de nomear os dois eixos corretamente, definir um tema e posicionar a legenda.

6.2 Utilizando o banco de dados `penguins` faça um histograma com a distribuição da massa corporal para cada uma das espécies. Utilize uma cor de preenchimento para cada espécie.

6.3 Utilizando o banco de dados `penguins`, faça três gráficos com o mesmo eixo Y e eixo X. Coloque o comprimento das nadadeiras no eixo Y e as espécies de pinguins no eixo X. No primeiro gráfico, utilize o `geom_jitter()` para plotar os dados brutos. No segundo gráfico, utilize o `geom_violin()` para mostrar a distribuição de densidade dos dados. No terceiro gráfico, utilize o `geom_boxplot()` para destacar a mediana e os quartis.

6.4 Se você conseguiu resolver o exercício 6.3, agora dê um passo a mais e compare os três gráficos lado a lado utilizando a função `grid.arrange()`. Lembre-se de colocar um título informativo em cada um dos gráficos antes de juntá-los em uma prancha única. Ao comparar os três tipos de gráficos, qual você considera mais informativo? Experimente combinar mais de um "geom" (camadas) e produzir gráficos ainda mais interessantes.

6.5 Utilize o conjunto de dados `ecodados::anova_dois_fatores` para construir um gráfico de barras com a média e o erro padrão do Tempo (Tempo para eliminar a droga do corpo) no eixo Y em função da variável Pessoas (XX ou XY) e Idade (jovem ou idoso). Antes de fazer o gráfico leia com atenção a descrição do mesmo através do comando `?ecodados::anova_dois_fatores`. Uma dica, utilize `fill` dentro do `aes()` para representar um dos fatores (ex. Pessoas). O outro fator você pode representar no eixo X. Veja se consegue, se não conseguir pode olhar a cola com a solução para aprender como é feito. Outra dica, pesquise sobre a função `stat_summary()` ela irá te ajudar a calcular a média e o erro padrão dentro do comando que gera o gráfico.

6.6 Utilize o banco de dados `penguins` para criar um gráfico de dispersão entre o tamanho da nadadeira (eixo Y) e a massa corporal (eixo X). Utilize o argumento `fill` para ilustrar com cores as diferenças entre os sexos e utilize a função `facte_grid()` para criar um gráfico separado para cada espécie de pinguim. Se você não conhece essa função, dê uma olhada no help `?facet_grid()`. Você também pode utilizar a função `drop_na()` para excluir os dados faltantes da coluna `sexo`.

Soluções dos exercícios.

Capítulo 7

Modelos lineares

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(ecodados)
library(car)
library(ggpubr)
library(ggforce)
library(lsmeans)
library(lmtest)
library(sjPlot)
library(nlme)
library(ape)
library(fields)
library(tidyverse)
library(vegan)
library(rdist)

## Dados
CRC_PN_macho <- ecodados::teste_t_var_igual
CRC_LP_femea <- ecodados::teste_t_var_diferente
Pareado <- ecodados::teste_t_pareado
correlacao_arbustos <- ecodados::correlacao
dados_regressao <- ecodados::regressoes
dados_regressao_mul <- ecodados::regressoes
dados_anova_simples <- ecodados::anova_simples
dados_dois_fatores <- ecodados::anova_dois_fatores
dados_dois_fatores_interacao <- ecodados::anova_dois_fatores
dados_dois_fatores_interacao2 <- ecodados::anova_dois_fatores_interacao2
dados_bloco <- ecodados::anova_bloco
dados_ancova <- ecodados::ancova
data("mite")
data("mite.xy")
coords <- mite.xy
colnames(coords) <- c("long", "lat")
data("mite.env")
```

Importante
Estatísticas frequentistas como as que serão abordadas neste capítulo são baseadas em testes estatísticos (e.g., F, t, χ^2, etc.), que são resultados numéricos do teste e possuem um valor de probabilidade (valor de P) associado com este teste (Gotelli & Ellison 2012). **O valor de P** mede a probabilidade que os valores observados ou mais extremos seriam encontrados caso a hipótese nula seja verdadeira (veja Capítulo 2). Ao longo do livro usaremos o critério convencional de rejeitar a hipótese nula quando P < 0.05. Contudo, sugerimos a leitura destes artigos artigos (Barber & Ogle 2014, Burnham & Anderson

2014, Murtaugh 2014, White et al. 2014, Halsey 2019, Muff et al. 2021) que discutem as limitações e problemas associados ao valor de P.

7.1 Teste T (de Student) para duas amostras independentes

Uma das perguntas mais comuns em estatística é saber se há diferença entre as médias de dois grupos ou tratamentos. Para responder a esta pergunta, William Sealy Gosset, químico da cervejaria Guinness, desenvolveu em 1908 o Teste T que é uma estatística que segue uma distribuição t de *Student* para rejeitar ou não uma hipótese nula de médias iguais entre dois grupos.

$$t = \frac{(\acute{X}_1 - \acute{X}_2)}{\sqrt{\frac{2S_p^2}{n}}}$$

Onde:

- $\acute{X}1$ - $\acute{X}2$ = diferença entre as médias de duas amostras
- S_p^2 = desvio padrão das amostras
- *n* = tamanho das amostras

Premissas do Teste t

- As amostras devem ser independentes
- As unidades amostrais são selecionadas aleatoriamente
- Distribuição normal (gaussiana) dos resíduos
- Homogeneidade da variância

Importante

- Zar (2010) indica que o Teste T é robusto mesmo com moderada violação da normalidade, principalmente se o tamanho amostral for alto.
- Caso as variâncias não sejam homogêneas, isso deve ser informado na linha de comando, pois o denominador da fórmula acima será corrigido.

Avaliação das premissas

Uma das maneiras de avaliarmos as premissas de normalidade e homogeneidade da variância relacionadas às análises do teste T, ANOVA e regressões lineares simples e múltiplas é o uso da inspeção gráfica da distribuição dos resíduos (Figura 7.1) (Zuur et al. 2010).

- A premissa de homogeneidade da variância pode ser avaliada através do gráfico de dispersão dos resíduos (eixo X) pelos valores preditos da variável resposta (eixo Y) (Figura 7.1A). A distribuição dos resíduos será homogênea se não observarmos nenhum padrão na distribuição dos pontos (i.e., forma em V, U ou funil).
- A premissa de normalidade dos resíduos pode ser avaliada através do gráfico de *quantis-quantis* (QQ-plots). A distribuição dos resíduos será normal se os pontos estiverem próximos à reta (Figura 7.1B).

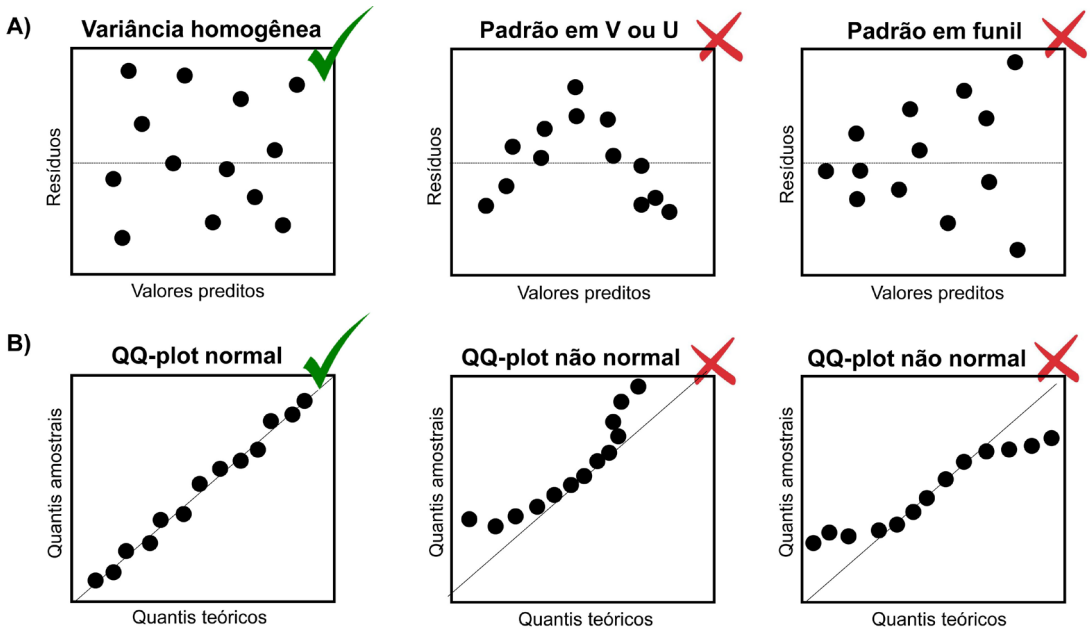


Figura 7.1: Inspeção gráfica da homogeneidade da variância (A) e normalidade dos resíduos (B). Os símbolos verdes indicam que os gráficos em que os resíduos apresentam distribuição homogênea e normal, enquanto os símbolos vermelhos indicam os gráficos em que os resíduos violam as premissas do teste.

Exemplo prático 1 - Teste T para duas amostras com variâncias iguais

Explicação dos dados

Neste exemplo, avaliaremos o comprimento rostro-cloacal (CRC em milímetros) de machos de *Physalaemus nattereri* (Anura:Leptodactylidae) amostrados em diferentes estações do ano com armadilhas de interceptação e queda na Região Noroeste do Estado de São Paulo (da Silva & Rossa-Feres 2010).

Pergunta

- O CRC dos machos de *P. nattereri* é maior na estação chuvosa do que na estação seca?

Predições

- O CRC dos machos será maior na estação chuvosa porque há uma vantagem seletiva para os indivíduos maiores durante a atividade reprodutiva

Variáveis

- Data frame com os indivíduos (unidade amostral) nas linhas e CRC (mm - variável resposta contínua) e estação (variável preditora categórica) como colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variáveis preditoras e respostas nas colunas

Análise

Vamos olhar os dados usando a função `head()`.

```
## Cabeçalho dos dados
head(CRC_PN_macho)
#>    CRC Estacao
#> 1 3.82 Chuvosa
#> 2 3.57 Chuvosa
#> 3 3.67 Chuvosa
#> 4 3.72 Chuvosa
#> 5 3.75 Chuvosa
#> 6 3.83 Chuvosa
```

Vamos verificar a normalidade dos resíduos usando o QQ-plot (Figura 7.2).

```
## Teste de normalidade
residuos <- lm(CRC ~ Estacao, data = CRC_PN_macho)
qqPlot(residuos)
#> [1] 22 26
```

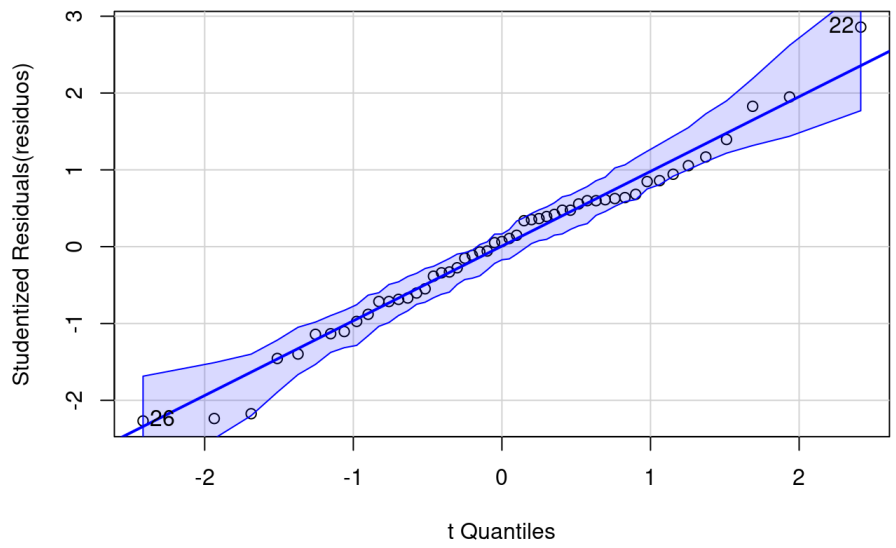


Figura 7.2: Normalidade dos resíduos usando o QQ-plot.

Os pontos estão próximos à reta, indicando que a distribuição dos resíduos é normal (veja Figura 7.1).

Outra possibilidade é usar os testes de Shapiro-Wilk e Levene para verificar a normalidade e a homogeneidade da variância, respectivamente.

Importante

A Hipótese Nula (H0) destes testes é que a distribuição é normal ou homogênea:

- Valor de p < 0.05 significa que os dados **não apresentam** distribuição normal ou homogênea
- Valor de p > 0.05 significa que os dados **apresentam** distribuição normal ou homogênea

Teste de Shapiro-Wilk para normalidade dos resíduos.

```
## Teste de Shapiro-Wilk
residuos_modelo <- residuals(residuos)
shapiro.test(residuos_modelo)
#>
#>  Shapiro-Wilk normality test
#>
#> data:  residuos_modelo
#> W = 0.98307, p-value = 0.6746
```

Teste de Levene para homogeneidade de variância dos resíduos.

```
## Teste de homogeneidade de variância
leveneTest(CRC ~ as.factor(Estacao), data = CRC_PN_macho)
#> Levene's Test for Homogeneity of Variance (center = median)
#>       Df F value Pr(>F)
#> group  1  1.1677 0.2852
#>       49
```

Percebam que a distribuição dos resíduos foi normal e homogênea na inspeção gráfica, assim como nos testes de Shapiro e Levene, respectivamente. Agora podemos realizar a análise sabendo que os dados seguem as premissas requeridas pelo Teste T.

Vamos para os códigos da análise do Teste T para amostragens independentes e variâncias iguais.

```
## Análise Teste T
t.test(CRC ~ Estacao, data = CRC_PN_macho, var.equal = TRUE)
#>
#>  Two Sample t-test
#>
#> data:  CRC by Estacao
#> t = 4.1524, df = 49, p-value = 0.000131
#> alternative hypothesis: true difference in means between group Chuvosa and
group Seca is not equal to 0
#> 95 percent confidence interval:
#>  0.2242132 0.6447619
#> sample estimates:
#> mean in group Chuvosa    mean in group Seca
#>              3.695357              3.260870
```

Quatro valores devem ser apresentados aos leitores: i) estatística do teste - representada por **t = 4,15**, ii) valor de significância - representado por **p-value = 0,0001**, iii) graus de liberdade - representado por **df = 49**, e iv) diferença entre as médias. Veja abaixo como descrever os resultados no seu trabalho.

Visualizar os resultados em gráfico (Figura 7.3).

```
## Gráfico
ggplot(data = CRC_PN_macho, aes(x = Estacao, y = CRC, color = Estacao)) +
    labs(x = "Estações",
         y = expression(paste("CRC (mm) - ", italic("P. nattereri")))) +
```

```
  geom_boxplot(fill = c("darkorange", "cyan4"), color = "black",
               outlier.shape = NA) +
  geom_jitter(shape = 16, position = position_jitter(0.1),
              cex = 5, alpha = 0.7) +
  scale_color_manual(values = c("black", "black")) +
  tema_livro() +
  theme(legend.position = "none")
```

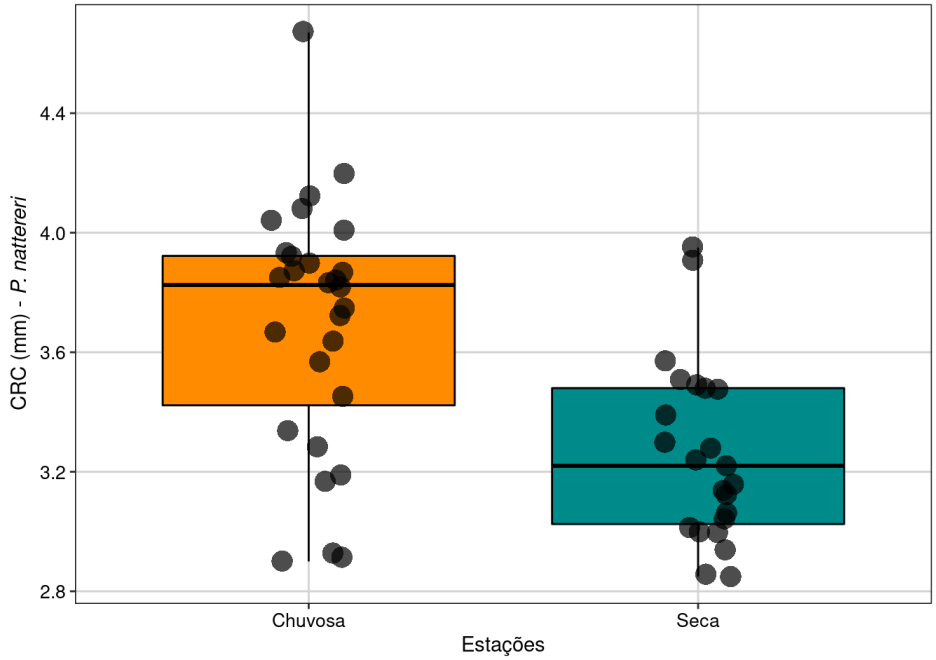


Figura 7.3: Boxplot da análise do Teste T para duas amostras com variâncias iguais.

Interpretação dos resultados

Neste exemplo, rejeitamos a hipótese nula de que as médias do CRC dos machos entre as estações seca e chuvosa são iguais. Os resultados mostram que os machos de *P. nattereri* coletados na estação chuvosa foram em média 0,43 mm maiores do que os machos coletados na estação seca (t_{49} = 4,15, P < 0,001).

Exemplo prático 2 - Teste T para duas amostras independentes com variâncias diferentes

Explicação dos dados

Neste exemplo, avaliaremos o comprimento rostro-cloacal (CRC - milímetros) de fêmeas de *Leptodactylus podicipinus* amostradas em diferentes estações do ano com armadilhas de interceptação e queda na Região Noroeste do Estado de São Paulo (da Silva & Rossa-Feres 2010).

> **Importante**
> Os dados foram alterados em relação a publicação original para se enquadrarem no exemplo de amostras com variâncias diferentes.

Pergunta

- O CRC das fêmeas de *L. podicipinus* é maior na estação chuvosa do que na estação seca?

Predições

- O CRC das fêmeas será maior na estação chuvosa porque há uma vantagem seletiva para os indivíduos maiores durante a atividade reprodutiva

Variáveis

- Data frame com os indivíduos (unidade amostral) nas linhas e CRC (mm - variável resposta contínua) e estação (variável preditora categórica) como colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variáveis preditoras e respostas nas colunas

Análise

Olhar os dados usando a função `head()`.

```
## Cabeçalho dos dados
head(CRC_LP_femea)
#>    CRC Estacao
#> 1 2.72 Chuvosa
#> 2 2.10 Chuvosa
#> 3 3.42 Chuvosa
#> 4 1.50 Chuvosa
#> 5 3.90 Chuvosa
#> 6 4.00 Chuvosa
```

Vamos avaliar as premissas do teste. Começando com o teste de normalidade (Figura 7.4).

```
## Teste de normalidade usando QQ-plot
residuos_LP <- lm(CRC ~ Estacao, data = CRC_LP_femea)
qqPlot(residuos_LP)
#> [1] 4 6
```

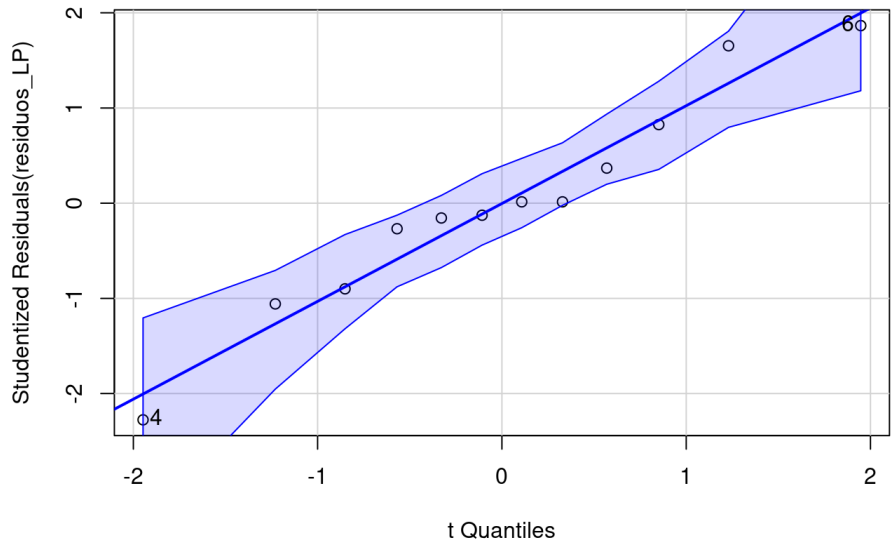


Figura 7.4: Normalidade dos resíduos usando o QQ-plot.

Os resíduos apresentam distribuição normal. Vamos testar também com o Shapiro-Wilk para normalidade dos resíduos.

```
## Teste de Shapiro-Wilk
residuos_modelo_LP <- residuals(residuos_LP)
shapiro.test(residuos_modelo_LP)
#>
#>  Shapiro-Wilk normality test
#>
#> data:  residuos_modelo_LP
#> W = 0.96272, p-value = 0.8219
```

Agora vamos avaliar a homogeneidade da variância.

```
## Teste de homogeneidade da variância
leveneTest(CRC ~ as.factor(Estacao), data = CRC_LP_femea)
#> Levene's Test for Homogeneity of Variance (center = median)
#>       Df F value  Pr(>F)
#> group  1  9.8527 0.01053 *
#>       10
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Os resíduos não apresentam homogeneidade da variância. Portanto, vamos realizar o Teste T com variâncias diferentes. Para isso, use o argumento `var.equal = FALSE`.

```
## Teste T
t.test(CRC ~ Estacao, data = CRC_LP_femea, var.equal = FALSE)
#>
#>  Welch Two Sample t-test
#>
#> data:  CRC by Estacao
#> t = -1.7633, df = 6.4998, p-value = 0.1245
```

```
#> alternative hypothesis: true difference in means between group Chuvosa and
group Seca is not equal to 0
#> 95 percent confidence interval:
#>  -1.5489301  0.2375016
#> sample estimates:
#> mean in group Chuvosa    mean in group Seca
#>              2.834286              3.490000
```

Neste exemplo, não rejeitamos a hipótese nula e consideramos que as médias do CRC das fêmeas entre as estações seca e chuvosa são iguais ($t_{6,49}$ = 1,76, P = 0,12).

Vamos visualizar os resultados em um gráfico (Figura 7.5).

```
## Gráfico
ggplot(data = CRC_LP_femea, aes(x = Estacao, y = CRC, color = Estacao)) +
    geom_boxplot(fill = c("darkorange", "cyan4"), width = 0.5,
                 color = "black", outlier.shape = NA, alpha = 0.7) +
    geom_jitter(shape = 20, position = position_jitter(0.2),
                color = "black", cex = 5) +
    scale_color_manual(values = c("darkorange", "cyan4")) +
    labs(x = "Estações",
         y = expression(paste("CRC (mm) - ", italic("L. podicipinus"))),
         size = 15) +
    tema_livro() +
    theme(legend.position = "none")
```

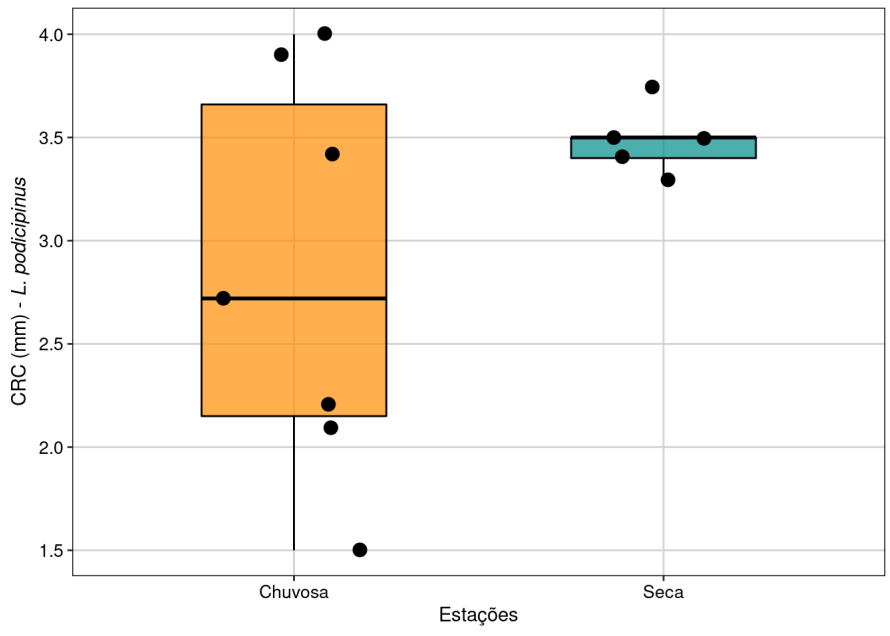


Figura 7.5: Boxplot da análise do Teste T para duas amostras independentes com variâncias diferentes.

Interpretação dos resultados

Os resultados mostram que as fêmeas de *L. podicipinus* coletadas na estação chuvosa não são maiores do que as fêmeas coletadas na estação seca, apesar de possuírem maior variância, o que pode ser biologicamente interessante.

7.2 Teste T para amostras pareadas

O Teste T Pareado é uma estatística que usa dados medidos duas vezes na mesma unidade amostral, resultando em pares de observações para cada amostra (amostras pareadas). Ele determina se a diferença da média entre duas observações é zero.

$$t = \frac{\acute{d}}{S_{\acute{d}}}$$

Onde:

- $\acute{d}$ = média da diferença das medidas pareadas. Observe que o teste não usa as medidas originais, e sim, a diferença para cada par
- $S\acute{d}$ = erro padrão da diferença das medidas pareadas

Premissas do Teste t para amostras pareadas

- As unidades amostrais são selecionadas aleatoriamente
- As observações **não** são independentes
- Distribuição normal (gaussiana) dos valores da diferença para cada par

Exemplo prático 1 - Teste T para amostras pareadas

Explicação dos dados

Neste exemplo, avaliaremos a diferença na riqueza de espécies de artrópodes registradas em 27 localidades. Todas as localidades foram amostradas duas vezes. A primeira amostragem foi realizada na localidade antes da perturbação e a segunda amostragem foi realizada após a localidade ter sofrido uma queimada. Portanto, existe uma dependência temporal, uma vez que amostramos a mesma localidade antes e depois da queimada.

Pergunta

- A riqueza de espécies de artrópodes é prejudicada pelas queimadas?

Predições

- A riqueza de espécies de artrópodes será maior antes da queimada devido a extinção local das espécies

Variáveis

- Data frame com as localidades nas linhas e riqueza de espécies (variável resposta contínua) e estado (Pré-queimada ou Pós-queimada - variável preditora categórica) da localidade nas colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variáveis preditoras e respostas nas colunas

Análise

Olhando os dados com a função `head()`.

```
## Cabeçalho dos dados
head(Pareado)
#>   Areas Riqueza       Estado
#> 1     1      92 Pre-Queimada
#> 2     2      74 Pre-Queimada
#> 3     3      96 Pre-Queimada
#> 4     4      89 Pre-Queimada
#> 5     5      76 Pre-Queimada
#> 6     6      80 Pre-Queimada
```

Cálculo do Teste T com amostras pareadas.

```
## Análise Teste T Pareado

t.test(Riqueza ~ Estado, paired = TRUE, data = Pareado)
#>
#>  Paired t-test
#>
#> data:  Riqueza by Estado
#> t = -7.5788, df = 26, p-value = 4.803e-08
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#>  -56.63994 -32.47117
#> sample estimates:
#> mean of the differences
#>               -44.55556
```

Neste exemplo, rejeitamos a hipótese nula de que a riqueza de espécies de artrópodes é igual antes e depois da queimada ($t_{26} = 7{,}57$, $P < 0{,}001$).

Podemos visualizar os resultados em gráfico (Figura 7.6).

```
## Gráfico
ggpaired(Pareado, x = "Estado", y = "Riqueza",
         color = "Estado", line.color = "gray", line.size = 0.8,
         palette = c("darkorange", "cyan4"), width = 0.5,
         point.size = 4, xlab = "Estado das localidades",
         ylab = "Riqueza de Espécies") +
    expand_limits(y = c(0, 150)) +
    tema_livro()
```

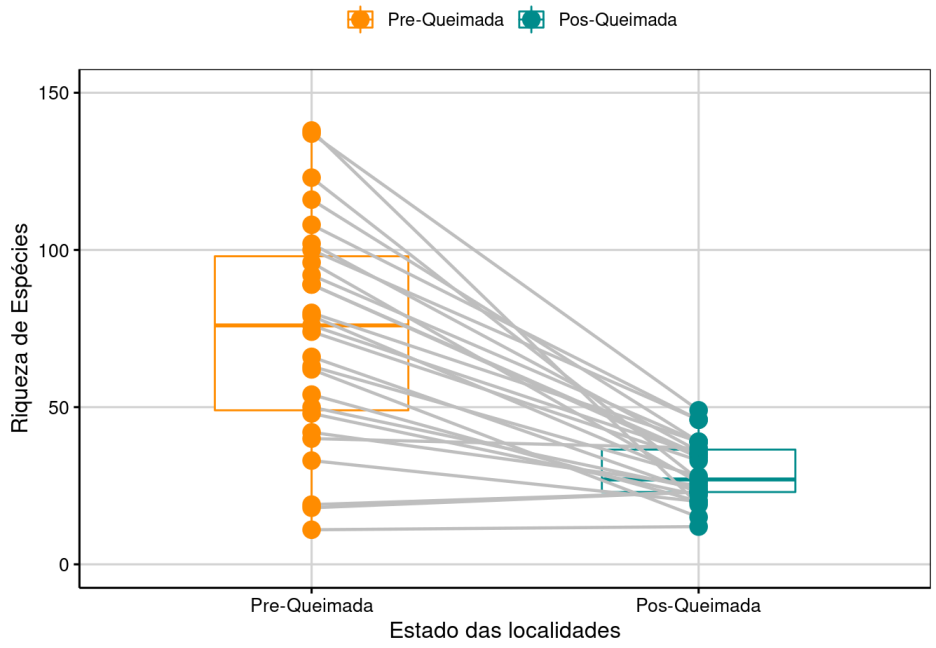


Figura 7.6: Boxplot da análise do Teste T para duas amostras pareadas.

Interpretação dos resultados

Os resultados mostram que as localidades após as queimadas apresentam em média 44,5 espécies de artrópodes a menos do que antes das queimadas.

7.3 Correlação de Pearson

É um teste que mede a força relativa da relação linear entre duas variáveis contínuas (X e Y). Importante ressaltar que a análise de correlação não assume que a variável X influencie a variável Y, ou que exista uma relação de causa e efeito entre elas (Zar 2010). A análise é definida em termos da variância de X, a variância de Y, e a covariância de X e Y (i.e., como elas variam juntas).

$$r = \frac{\sum XY - \frac{\sum X \sum Y}{n}}{\sqrt{\left(\sum X^2 - \frac{\sum X^2}{n}\right)\left(\sum Y^2 - \frac{\sum Y^2}{n}\right)}}$$

Onde:

- r = coeficiente de correlação que indica a força da relação linear entre as duas variáveis. Seu limite de valores está entre $-1 \leq r \leq 1$. A correlação positiva indica que o aumento no valor de uma das variáveis é acompanhado pelo aumento no valor da outra variável. A correlação negativa indica que o aumento no valor de uma das variáveis é acompanhado pela diminuição no valor da outra variável. Se *r* é igual a zero, não existe correlação entre as variáveis (Figura 7.7).

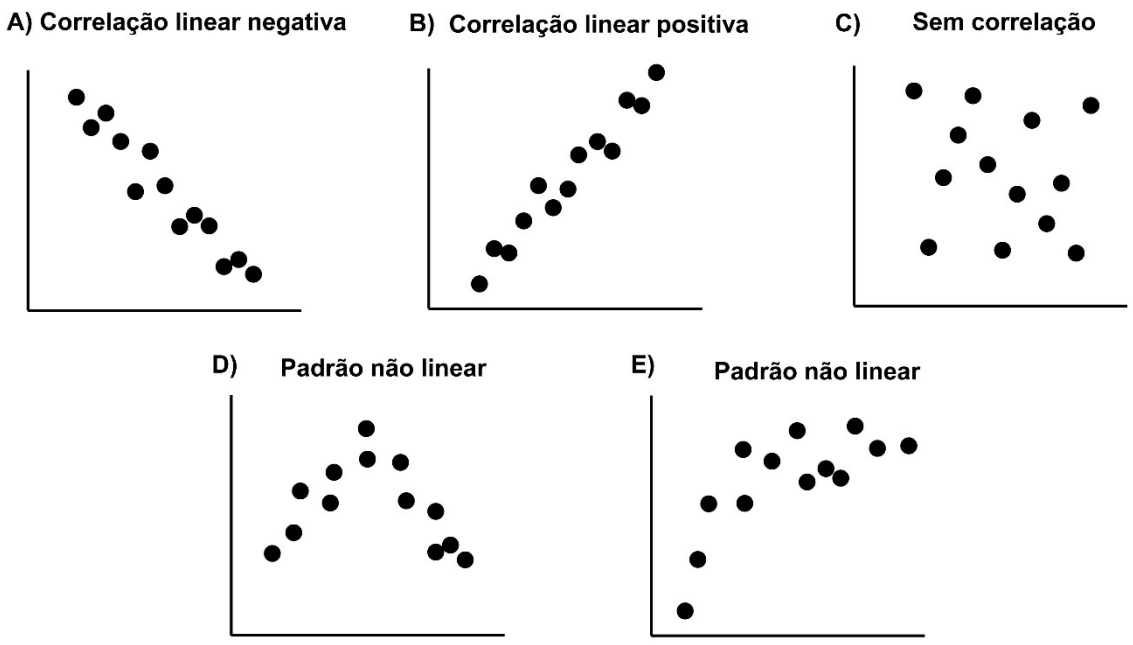


Figura 7.7: Exemplo de correlações negativa (A), positiva (B) e nula (C) e variáveis que não apresentam relações lineares entre si (D-E).

Premissas da Correlação de Person

- As amostras devem ser independentes e pareadas (i.e., as duas variáveis devem ser medidas na mesma unidade amostral)
- As unidades amostrais são selecionadas aleatoriamente
- A relação entre as variáveis tem que ser linear

Exemplo prático 1 - Correlação de Pearson

Explicação dos dados

Neste exemplo, avaliaremos a correlação entre a altura do tronco e o tamanho da raiz medidos em 35 indivíduos de uma espécie vegetal arbustiva.

Pergunta

- Existe correlação entre a altura do tronco e o tamanho da raiz dos arbustos?

Predições

- A altura do tronco é positivamente correlacionada com o tamanho da raiz

Variáveis

- Data frame com os indivíduos (unidade amostral) nas linhas e altura do tronco e tamanho da raiz (duas variáveis têm que ser contínuas) como colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variáveis preditoras e respostas nas colunas

Análise

Vamos olhar os dados com a função `head()`.

```
## Cabeçalho dos dados
head(correlacao_arbustos)
#>   Tamanho_raiz Tamanho_tronco
#> 1    10.177049       19.54383
#> 2     6.622634       17.13558
#> 3     7.773629       19.50681
#> 4    11.055257       21.57085
#> 5     4.487274       13.22763
#> 6    11.190216       21.62902
```

Cálculo do Teste da Correlação de Pearson. Para outros testes de correlação como Kendall ou Spearman é só alterar o argumento `method` e inserir o teste desejado.

```
## Correlação de Pearson
cor.test(correlacao_arbustos$Tamanho_raiz,
         correlacao_arbustos$Tamanho_tronco, method = "pearson")
#>
#>  Pearson's product-moment correlation
#>
#> data:  correlacao_arbustos$Tamanho_raiz and correlacao_arbustos$Tamanho_
tronco
#> t = 11.49, df = 33, p-value = 4.474e-13
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  0.7995083 0.9457816
#> sample estimates:
#>       cor
#> 0.8944449

## Alternativamente
cor.test(~ Tamanho_tronco + Tamanho_raiz, data = correlacao_arbustos,
         method = "pearson")
#>
#>  Pearson's product-moment correlation
#>
#> data:  Tamanho_tronco and Tamanho_raiz
#> t = 11.49, df = 33, p-value = 4.474e-13
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  0.7995083 0.9457816
#> sample estimates:
#>       cor
#> 0.8944449
```

Neste exemplo, rejeitamos a hipótese nula de que as variáveis não são correlacionadas (r = 0.89, P < 0,001).

Podemos visualizar os resultados em gráfico (Figura 7.8).

```
## Gráfico
ggplot(data = correlacao_arbustos, aes(x = Tamanho_raiz,
                                       y = Tamanho_tronco)) +
    labs(x = "Tamanho da raiz (m)", y = "Altura do tronco (m)") +
    geom_point(size = 4, shape = 21, fill = "darkorange", alpha = 0.7) +
    geom_text(x = 14, y = 14, label = "r = 0.89, P < 0.001",
              color = "black", size = 5) +
    geom_smooth(method = lm, se = FALSE, color = "black",
                linetype = "dashed") +
    tema_livro() +
    theme(legend.position = "none")
```

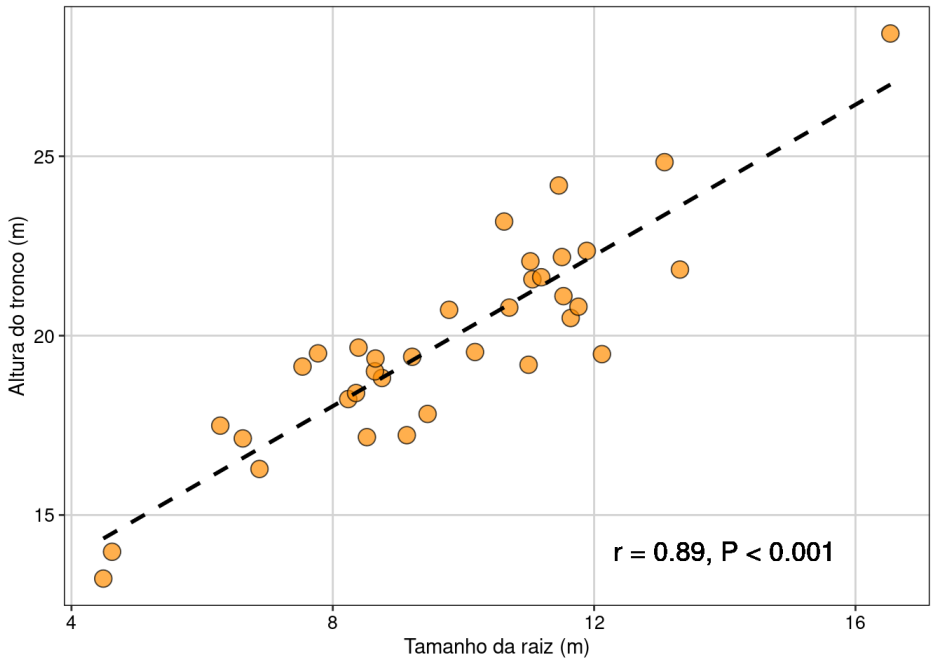


Figura 7.8: Gráfico mostrando a relação entre as variáveis e uma linha de tendência dos dados.

> **Importante**
> A linha de tendência tracejada no gráfico é apenas para ilustrar a relação positiva entre as variáveis. Ela não é gerada pela análise de correlação.

Interpretação dos resultados

Os resultados mostram que o aumento na altura dos arbustos é acompanhado pelo aumento no tamanho da raiz.

7.4 Regressão Linear Simples

A regressão linear simples é usada para analisar a relação entre uma variável preditora (plotada no eixo-X) e uma variável resposta (plotada no eixo-Y). As duas variáveis devem ser contínuas. Diferente das correlações, a regressão assume uma relação de causa e efeito entre as variáveis. O valor da variável preditora (X) causa, direta ou indiretamente, o valor da variável resposta (Y). Assim, Y é uma função linear de X:

$$Y = \beta_0 + \beta_1 X_i + \epsilon_i$$

Onde:

- β_0 = intercepto (*intercept*) que representa o valor da função quando X = 0
- β_1 = inclinação (*slope*) que mede a mudança na variável Y para cada mudança de unidade da variável X
- ϵ_1 = erro aleatório referente à variável Y que não pode ser explicado pela variável X

Premissas da Regressão Linear Simples

- As amostras devem ser independentes
- As unidades amostrais são selecionadas aleatoriamente
- Distribuição normal (gaussiana) dos resíduos
- Homogeneidade da variância dos resíduos

Exemplo prático 1 - Regressão Linear Simples

Explicação dos dados

Neste exemplo, avaliaremos a relação entre o gradiente de temperatura média anual (°C) e o tamanho médio do comprimento rostro-cloacal (CRC em mm) de populações de *Dendropsophus minutus* (Anura:Hylidae) amostradas em 109 localidades no Brasil (Boaratti & Da Silva 2015).

Pergunta

- A temperatura afeta o tamanho do CRC de populações de *Dendropsophus minutus*?

Predições

- Os CRC das populações serão menores em localidades mais quentes do que em localidades mais frias de acordo com a Hipótese do Balanço de Calor (Olalla-Tárraga & Rodríguez 2007)

Variáveis

- Data frame com as populações (unidade amostral) nas linhas e CRC (variável resposta) médio (mm) e temperatura média anual (variável preditora) como colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variáveis preditoras e respostas nas colunas

Análise

Olhando os dados com a função `head()`.

```
## Cabeçalho dos dados
head(dados_regressao)
#>      Municipio      CRC Temperatura Precipitacao
#> 1     Acorizal 22.98816    24.13000       1228.2
#> 2  Alpinopolis 22.91788    20.09417       1487.6
#> 3 Alto_Paraiso 21.97629    21.86167       1812.4
#> 4    Americana 23.32453    20.28333       1266.2
#> 5      Apiacas 22.83651    25.47333       2154.0
#> 6  Arianopolis 20.86989    20.12167       1269.2
```

Vamos calcular a regressão linear simples.

```
## regressão simples
modelo_regressao <- lm(CRC ~ Temperatura, data = dados_regressao)
```

Antes de vermos os resultados, vamos verificar a normalidade e homogeneidade das variâncias (Figura 7.9).

```
## Verificar as premissas do teste
par(mfrow = c(2, 2), oma = c(0, 0, 2, 0))
plot(modelo_regressao)
dev.off() # volta a configuração dos gráficos para o formato padrão
```

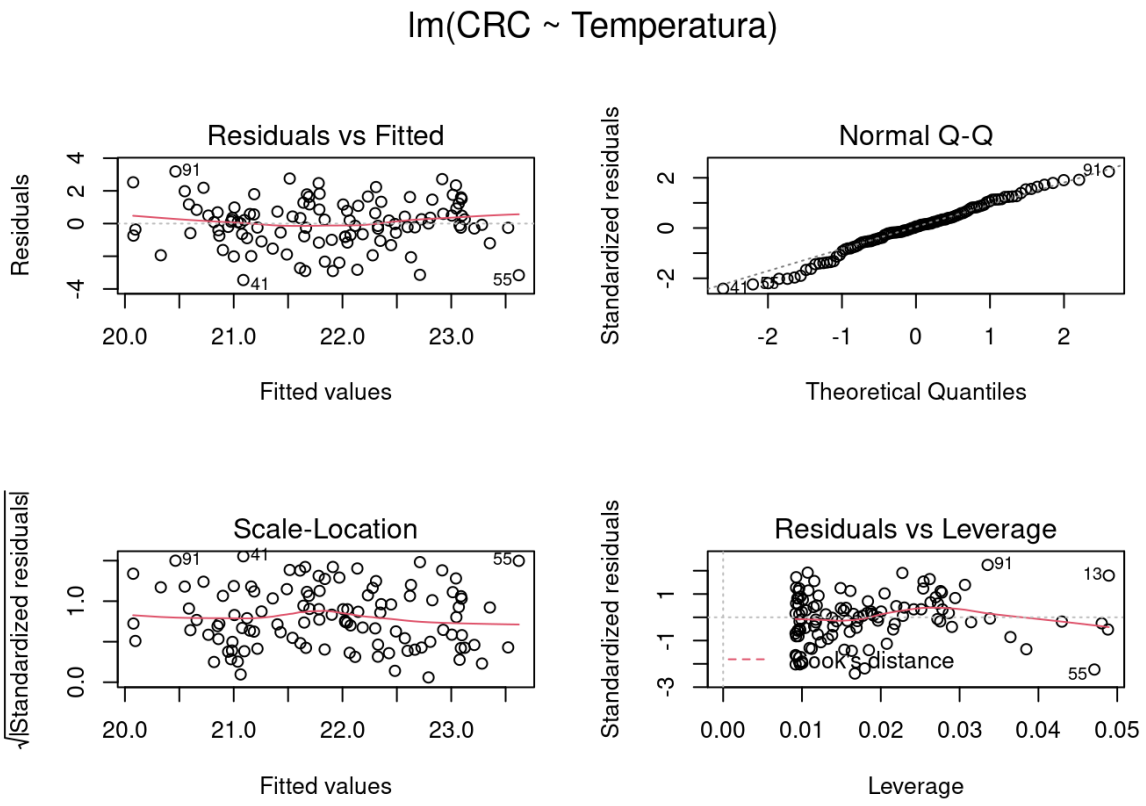


Figura 7.9: Gráficos mostrando as premissas da regressão linear simples.

Os gráficos *Residuals vs Fitted*, *Scale-Location*, e *Residual vs Leverage* estão relacionados com a homogeneidade da variância. Nestes gráficos, esperamos ver os pontos dispersos no espaço sem padrões com formatos em *U* ou funil. Neste caso, vemos que as linhas vermelhas (que indicam a tendência dos dados) estão praticamente retas, seguindo a linha pontilhada, sugerindo que não exista heterogeneidade de variância dos resíduos. O gráfico *Residual vs Leverage*, identifica os valores extremos que estejam a mais de uma unidade da distância de Cook (linha pontilhada vermelha). Quando muito discrepantes, esses valores podem influenciar os resultados dos testes estatísticos. Também não temos problemas com esse pressuposto do modelo aqui. O gráfico *Normal Q-Q (quantile-quantile plot)* mede desvios da normalidade. Neste caso, esperamos que os pontos sigam uma linha reta (i.e., fiquem muito próximos da linha pontilhada) e não apresentem padrões com formatos de *S* ou arco. Podemos observar que tanto a normalidade como a homogeneidade do resíduos estão dentro dos padrões esperados.

Vamos ver os resultados da regressão simples usando as funções `anova()` e `summary()`. A função `anova()` retorna uma tabela contendo o grau de liberdade (df), soma dos quadrados, valor de F e o valor de P.

```
## Resultados usando a função anova
anova(modelo_regressao)
#> Analysis of Variance Table
#>
#> Response: CRC
#>              Df  Sum Sq Mean Sq F value    Pr(>F)
#> Temperatura   1  80.931  80.931   38.92 9.011e-09 ***
#> Residuals   107 222.500   2.079
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A função `summary()` retorna uma tabela contendo o valor do intercepto, inclinação da reta (*slope*) e o coeficiente de determinação (R^2) que indica a proporção da variação na variável Y que pode ser atribuída à variação na variável X. Percebam que a parte final dos resultados apresentados no `summary()` são os mesmo apresentados pela função `anova()`.

```
## Resultados usando a função summary
summary(modelo_regressao)
#>
#> Call:
#> lm(formula = CRC ~ Temperatura, data = dados_regressao)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -3.4535 -0.7784  0.0888  0.9168  3.1868
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 16.23467    0.91368  17.768  < 2e-16 ***
#> Temperatura  0.26905    0.04313   6.239 9.01e-09 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#>
#> Residual standard error: 1.442 on 107 degrees of freedom
#> Multiple R-squared:  0.2667, Adjusted R-squared:  0.2599
#> F-statistic: 38.92 on 1 and 107 DF,  p-value: 9.011e-09
```

Vamos visualizar os resultados em gráfico (Figura 7.10).

```
## Gráfico
ggplot(data = dados_regressao, aes(x = Temperatura, y = CRC)) +
    labs(x = "Temperatura média anual (°C)",
         y = "Comprimento rostro-cloacal (mm)") +
    geom_point(size = 4, shape = 21, fill = "darkorange", alpha = 0.7) +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    tema_livro() +
    theme(legend.position = "none")
```

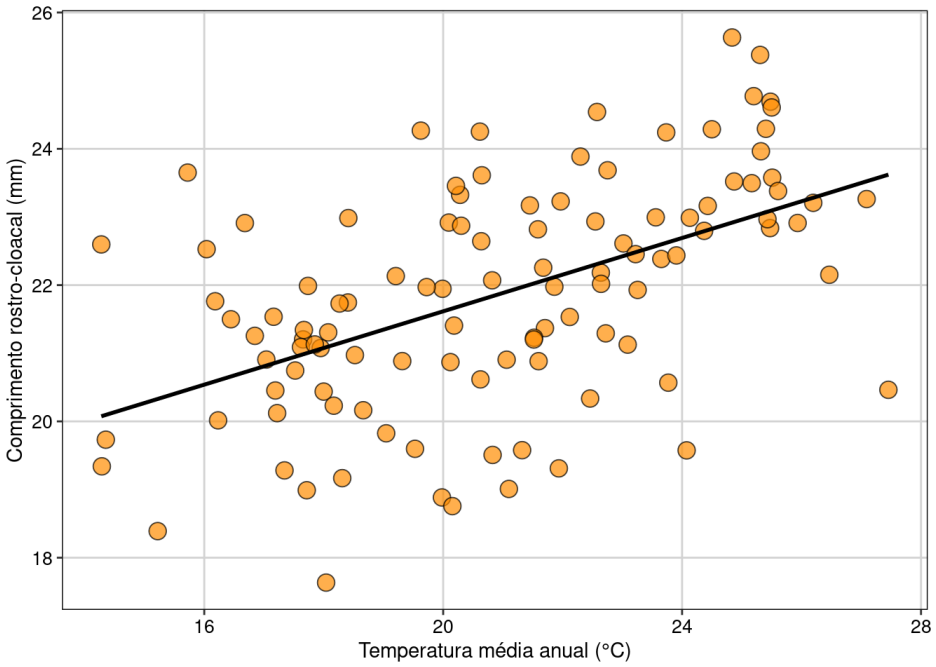


Figura 7.10: Gráfico mostrando a relação entre as variáveis e modelo linear simples, representado pela linha contínua.

Interpretação dos resultados

Neste exemplo, rejeitamos a hipótese nula de que não existe relação entre o tamanho do CRC das populações de *D. minutus* e a temperatura da localidade onde elas ocorrem ($F_{1,107}$ = 38,92, P < 0,001). Os resultados mostram que o tamanho do CRC das populações tem uma relação positiva com a temperatura das localidades. Assim, populações de *D. minutus* em localidades mais quentes apresentam maior CRC do que as populações em localidades mais frias. Podemos também usar os coeficientes da regressão para entender como a mudança na variável preditora (temperatura) afeta o tamanho corporal médio dos anuros. Neste caso, ao usar o comando `coef(modelo_regressao)`, obtemos os valores `16,23` e `0,27`, respectivamente os valores do intercepto (`β0`) e da temperatura (`β1`). O valor de `0,27` indica que a mudança de uma unidade na variável preditora (neste caso, graus), aumenta em 0,27 unidades (neste caso, centímetros) da variável dependente. Por exemplo, o modelo indica que o tamanho médio

dos indivíduos em locais com temperatura de 16º C é de 20,55 cm, ao passo que em locais com 26º C o tamanho aumenta para 23,25 cm, o que representa um ganho de 13%.

7.5 Regressão Linear Múltipla

A regressão linear múltipla é uma extensão da regressão linear simples. Ela é usada quando queremos determinar o valor da variável resposta (Y) com base nos valores de duas ou mais variáveis preditoras (X_1, X_2, X_n).

$$Y = \beta_0 + \beta_1 X_1 + \beta_n X_n + \epsilon_i$$

Onde:

- β_0 = intercepto (*intercept*) que representa o valor da função quando X = 0
- β_n = inclinação (*slope*) que mede a mudança na variável Y para cada mudança de unidade das variáveis X_n
- ϵ_1 = erro aleatório referente a variável Y que não pode ser explicado pelas variáveis preditoras

Premissas da Regressão Linear Múltipla

- As amostras devem ser independentes
- As unidades amostrais são selecionadas aleatoriamente
- Distribuição normal (gaussiana) dos resíduos
- Homogeneidade da variância dos resíduos

Exemplo prático 1 - Regressão Linear Múltipla

Explicação dos dados

Utilizaremos o mesmo exemplo da regressão linear simples. Contudo, além do gradiente de temperatura média anual (°C), incluiremos o gradiente de precipitação anual (mm) como outra variável preditora do tamanho médio do comprimento rostro-cloacal (CRC em mm) de populações de *Dendropsophus minutus* (Anura:Hylidae) amostradas em 109 localidades no Brasil (Boaratti & Da Silva 2015).

Pergunta

- O tamanho do CRC das populações de *D. minutus* é influenciado pela temperatura e precipitação das localidades onde os indivíduos ocorrem?

Predições

- Os CRC das populações serão menores em localidades com clima quente e chuvoso do que em localidades com clima frio e seco

Variáveis

- Data frame com as populações (unidade amostral) nas linhas e CRC (variável resposta) médio (mm) e temperatura e precipitação (variáveis preditoras) como colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variáveis preditoras e respostas nas colunas

Análise

Olhando os dados usando a função `head()`.

```
## Cabeçalho dos dados
head(dados_regressao_mul)
#>      Municipio      CRC Temperatura Precipitacao
#> 1     Acorizal 22.98816    24.13000       1228.2
#> 2  Alpinopolis 22.91788    20.09417       1487.6
#> 3 Alto_Paraiso 21.97629    21.86167       1812.4
#> 4    Americana 23.32453    20.28333       1266.2
#> 5      Apiacas 22.83651    25.47333       2154.0
#> 6  Arianopolis 20.86989    20.12167       1269.2
```

Códigos para ajustar o modelo de regressão múltipla.

```
## Regressão múltipla
modelo_regressao_mul <- lm(CRC ~ Temperatura + Precipitacao,
                           data = dados_regressao_mul)
```

> **Importante**
> Multicolinearidade ocorre quando as variáveis preditoras são correlacionadas. Essa correlação é um problema porque as variáveis preditoras deveriam ser independentes. Além disso, a multicolinearidade aumentam o erro padrão associado aos coeficientes produzindo resultados menos confiáveis. O Fator de Inflação da Variância (VIF) é um teste que quantifica quanto do erro padrão dos coeficientes estimados estão inflados devido à multicolinearidade. Na regressão múltipla, cada variável preditora tem um valor de VIF. Alguns autores consideram valores de VIF acima de 10 como fortemente correlacionadas, outros mais conservadores consideram o valor de 5, 3 ou até mesmo 2. Mais detalhes em Zuur et al. (2010) e Dormann et al. (2013).

Vamos analisar se as variáveis apresentam multicolinearidade.

```
# Multicolinearidade
vif(modelo_regressao_mul)
#>  Temperatura Precipitacao
#>     1.041265     1.041265
```

Os valores são menores que 3, indicando que não há multicolinearidade.

Agora vamos verificar as premissas de normalidade e homogeneidade das variâncias (Figura 7.11).

```
## Normalidade e homogeneidade das variâncias
plot_grid(plot_model(modelo_regressao_mul , type = "diag"))
```

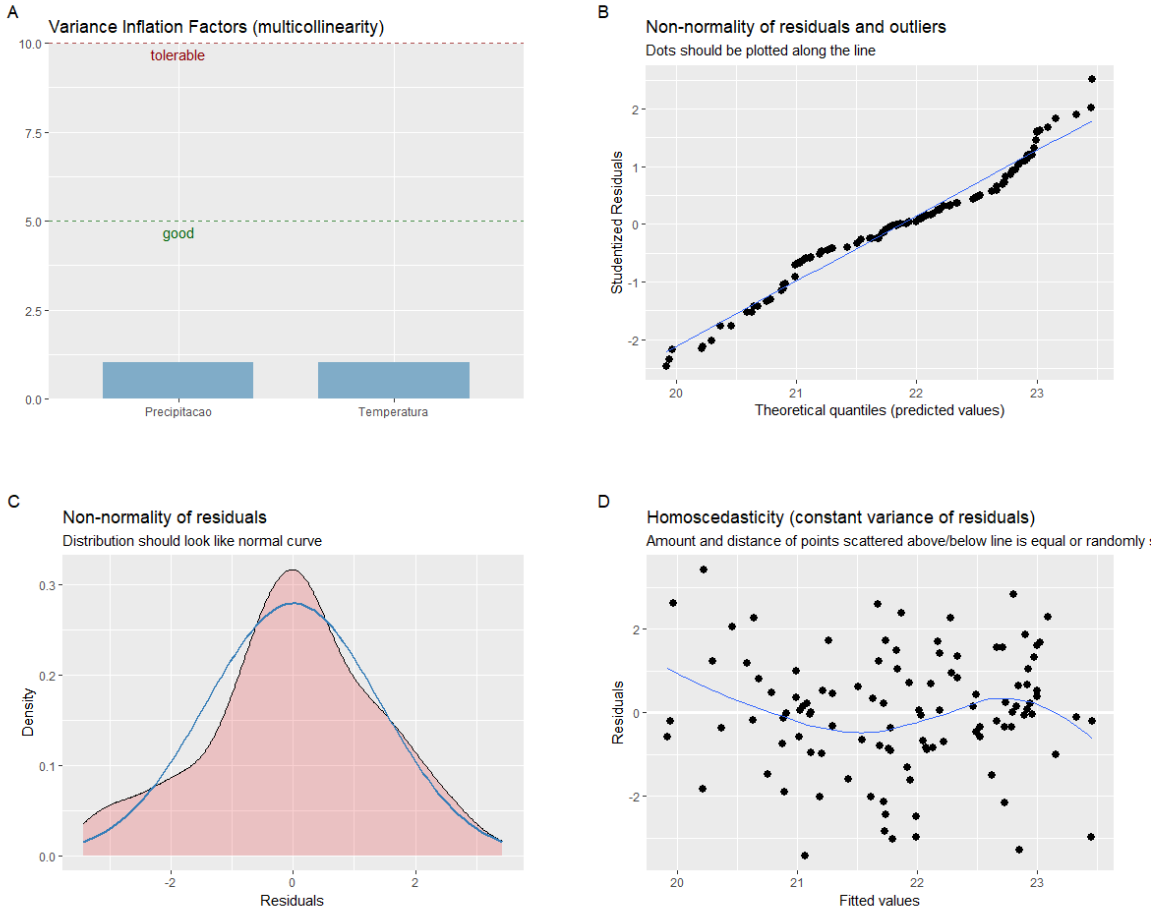


Figura 7.11: Gráficos mostrando as premissas da regressão linear múltipla.

Os resíduos apresentam distribuição normal e variâncias homogêneas.

Podemos ver os resultados da análise.

```
## regressão múltipla
summary(modelo_regressao_mul)
#>
#> Call:
#> lm(formula = CRC ~ Temperatura + Precipitacao, data = dados_regressao_mul)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -3.4351 -0.8026  0.0140  0.9420  3.4300
#>
#> Coefficients:
#>               Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  16.7162571  1.0108674  16.537  < 2e-16 ***
#> Temperatura   0.2787445  0.0439601   6.341 5.71e-09 ***
#> Precipitacao -0.0004270  0.0003852  -1.108     0.27
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#>
#> Residual standard error: 1.44 on 106 degrees of freedom
#> Multiple R-squared:  0.2751, Adjusted R-squared:  0.2614
#> F-statistic: 20.12 on 2 and 106 DF,  p-value: 3.927e-08
```

Percebam que a temperatura tem uma relação significativa e positiva com o tamanho do CRC das populações (P < 0.001), enquanto a precipitação não apresenta relação com o CRC (P = 0.27). Neste caso, é interessante saber se um modelo mais simples (e.g., contendo apenas temperatura) explicaria a distribuição tão bem ou melhor do que este modelo mais complexo considerando duas variáveis (temperatura e precipitação).

Para isso, podemos utilizar a *Likelihood-ratio test (LRT)* para comparar os modelos. A LRT compara dois modelos aninhados, testando se os parâmetros do modelo mais complexo diferem significativamente do modelo mais simples. Em outras palavras, ele testa se há necessidade de se incluir uma variável extra no modelo para explicar os dados.

```
## Criando os modelos aninhados
modelo_regressao_mul <- lm(CRC ~ Temperatura + Precipitacao,
                           data = dados_regressao_mul)
modelo_regressao <- lm(CRC ~ Temperatura, data = dados_regressao_mul)

## Likelihood-ratio test (LRT)
lrtest(modelo_regressao_mul, modelo_regressao)
#> Likelihood ratio test
#>
#> Model 1: CRC ~ Temperatura + Precipitacao
#> Model 2: CRC ~ Temperatura
#>   #Df  LogLik Df  Chisq Pr(>Chisq)
#> 1   4 -192.93
#> 2   3 -193.55 -1 1.2558     0.2624
```

> 👉 **Importante**
> A Hipótese Nula (H0) do teste *Likelihood-ratio test (LRT)* é de que o modelo mais simples é o melhor.
> - Valor de p < 0.05 **rejeita a hipótese nula** e o modelo mais complexo é o melhor
> - Valor de p > 0.05 **não rejeita a hipótese nula** e o modelo mais simples é o melhor

```
## Comparando com o modelo somente com o intercepto
# Criando um modelo sem variáveis, só o intercepto.
modelo_intercepto <- lm(CRC ~ 1, data = dados_regressao_mul)
lrtest(modelo_regressao, modelo_intercepto)
#> Likelihood ratio test
#>
#> Model 1: CRC ~ Temperatura
#> Model 2: CRC ~ 1
#>   #Df  LogLik Df  Chisq Pr(>Chisq)
#> 1   3 -193.55
```

```
#> 2   2 -210.46 -1 33.815  6.061e-09 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interpretação dos resultados

Neste exemplo, a precipitação não está associada com a variação no tamanho do CRC das populações de *D. minutus*. Por outro lado, a temperatura explicou 26% da variação do tamanho do CRC das populações.

7.6 Análises de Variância (ANOVA)

ANOVA refere-se a uma variedade de delineamentos experimentais nos quais a variável preditora é categórica e a variável resposta é contínua (Gotelli & Ellison 2012). Exemplos desses delineamentos experimentais são: ANOVA de um fator, ANOVA de dois fatores, ANOVA em blocos aleatorizados, ANOVA de medidas repetidas e ANOVA *split-splot*. De forma geral, a ANOVA é um teste estatístico usado para comparar a média entre grupos amostrados independentemente. Para isso, o teste leva em conta, além das médias dos grupos, a variação dos dados dentro e entre os grupos. Neste capítulo, iremos demonstrar os códigos para alguns dos principais delineamentos experimentais.

Premissas da ANOVA

- As amostras devem ser independentes
- As unidades amostrais são selecionadas aleatoriamente
- Distribuição normal (gaussiana) dos resíduos
- Homogeneidade da variância

> **Importante**
> ANOVA de medidas repetidas e ANOVA *split-plot* são análises com desenhos experimentais que apresentam dependência entre as amostras, mas controlam essa dependência nas suas formulações matemáticas.

7.6.1 ANOVA de um fator

Este teste considera delineamentos experimentais com apenas um fator (ou tratamento) que pode ser composto por três ou mais grupos (ou níveis).

Exemplo prático 1 - Anova de um fator

Explicação dos dados

Neste exemplo hipotético, avaliaremos se o adubo X-2020 disponibilizado recentemente no mercado melhora o crescimento dos indivíduos de *Coffea arabica* como divulgado pela empresa responsável pela venda do produto. Para isso, foi realizado um experimento com indivíduos de *C. arabica* cultivados em três grupos: i) grupo controle onde os indivíduos não receberam adubação, ii) grupo onde os

indivíduos receberam a adição do adubo tradicional mais utilizado pelos produtores de *C. arabica*, e iii) grupo onde os indivíduos receberam a adição do adubo X-2020.

Pergunta

- O crescimento dos indivíduos de *C. arabica* é melhorado pela adição do adubo X-2020?

Predições

- O crescimento dos indivíduos de *C. arabica* será maior no grupo que recebeu o adubo X-2020

Variáveis

- Data frame com as plantas (unidade amostral) nas linhas e o crescimento dos indivíduos de *C. arabica* (variável resposta) e os tratamentos (variável preditora) nas colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variável preditora e resposta nas colunas

Análise

Olhando os dados e criando o modelo para Anova de um fator.

```
## Cabeçalho dos dados
head(dados_anova_simples)
#>   Crescimento Tratamento
#> 1       7.190   Controle
#> 2       6.758   Controle
#> 3       6.101   Controle
#> 4       4.758   Controle
#> 5       6.542   Controle
#> 6       7.667   Controle

## Análise ANOVA de um fator
Modelo_anova <- aov(Crescimento ~ Tratamento, data = dados_anova_simples)
```

Vamos verificar a normalidade dos resíduos e homogeneidade da variância usando os testes de Shapiro-Wilk e Bartlett, respectivamente.

```
## Normalidade
shapiro.test(residuals(Modelo_anova))
#>
#>  Shapiro-Wilk normality test
#>
#> data:  residuals(Modelo_anova)
#> W = 0.94676, p-value = 0.08266

## Homogeneidade da variância
bartlett.test(Crescimento ~ Tratamento, data = dados_anova_simples)
#>
```

```
#>  Bartlett test of homogeneity of variances
#>
#> data:  Crescimento by Tratamento
#> Bartlett's K-squared = 0.61835, df = 2, p-value = 0.7341
```

Os resíduos apresentam distribuição normal e homogeneidade de variância.

Vamos ver os resultados da análise.

```
## Resultados da anova
anova(Modelo_anova)
#> Analysis of Variance Table
#>
#> Response: Crescimento
#>            Df Sum Sq Mean Sq F value    Pr(>F)
#> Tratamento  2 340.32 170.160  77.989 3.124e-13 ***
#> Residuals  33  72.00   2.182
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Percebam que o resultado da ANOVA (Pr(>F) < 0.001) indica que devemos rejeitar a hipótese nula que não há diferença entre as médias dos grupos. Contudo, os resultados não mostram quais são os grupos que apresentam diferenças. Para isso, temos que realizar testes de comparações múltiplas *post-hoc* para detectar os grupos que apresentam diferenças significativas entre as médias.

> **Importante**
> Os testes *post-hoc* só devem ser utilizados quando rejeitamos a hipótese nula (P < 0.05) no teste da ANOVA.

```
## Diferenças entre os tratamentos
# Teste de Tuckey's honest significant difference
TukeyHSD(Modelo_anova)
#>   Tukey multiple comparisons of means
#>     95% family-wise confidence level
#>
#> Fit: aov(formula = Crescimento ~ Tratamento, data = dados_anova_simples)
#>
#> $Tratamento
#>                                     diff       lwr       upr     p adj
#> Adubo_X-2020-Adubo_Tradicional  0.04991667 -1.429784  1.529617 0.9962299
#> Controle-Adubo_Tradicional     -6.49716667 -7.976867 -5.017466 0.0000000
#> Controle-Adubo_X-2020          -6.54708333 -8.026784 -5.067383 0.0000000
```

Visualizar os resultados em gráfico (Figura 7.12).

```
## Reorganizando a ordem que os grupos irão aparecer no gráfico
dados_anova_simples$Tratamento <- factor(dados_anova_simples$Tratamento,
                                         levels = c("Controle",
```

```
                                                "Adubo_Tradicional",
                                                "Adubo_X-2020"))

## Gráfico
ggplot(data = dados_anova_simples,
       aes(x = Tratamento, y = Crescimento, color = Tratamento)) +
    geom_boxplot(fill = c("darkorange", "darkorchid", "cyan4"),
                 color = "black", show.legend = FALSE, alpha = 0.4) +
    geom_jitter(shape = 16, position = position_jitter(0.1),
                cex = 4, alpha = 0.7) +
    scale_color_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    scale_y_continuous(limits = c(0, 20), breaks = c(0, 5, 10, 15, 20)) +
    geom_text(x = 1, y = 12, label = "ab", color = "black", size = 5) +
    geom_text(x = 2, y = 17, label = "a", color = "black", size = 5) +
    geom_text(x = 3, y = 17, label = "b", color = "black", size = 5) +
    scale_x_discrete(labels = c("Sem adubo", "Tradicional", "X-2020")) +
    labs(x = "Adubação", y = "Crescimento Coffea arabica (cm)",
         size = 20) +
    tema_livro() +
    theme(legend.position = "none")
```

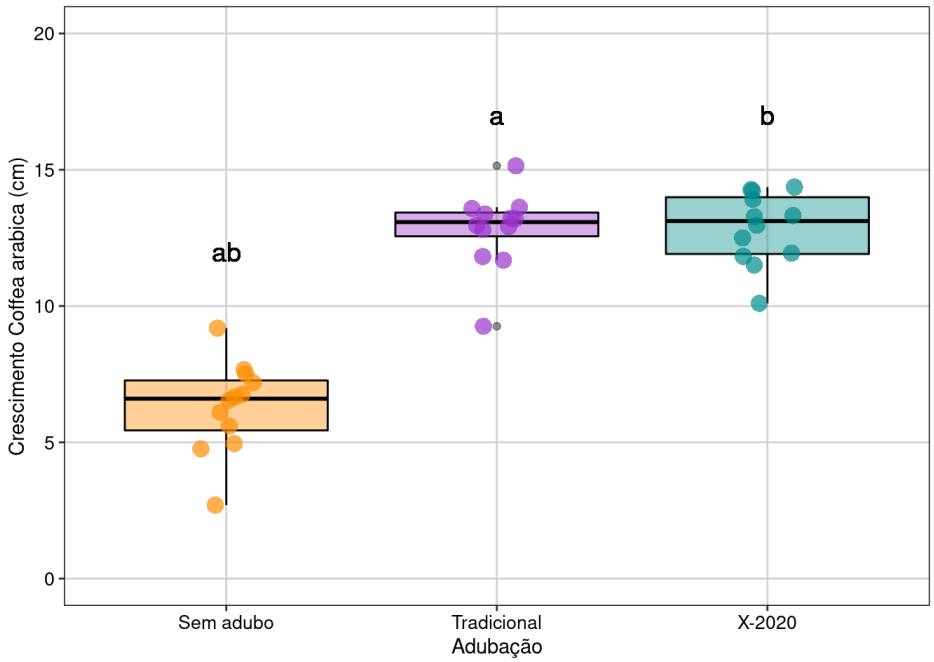


Figura 7.12: Gráfico de caixa mostrando o resultado da ANOVA de um fator.

Interpretação dos resultados

Neste exemplo, os indivíduos de *C. arabica* que receberam adubação (tradicional e X-2020) apresentaram maior crescimento do que os indivíduos que não receberam adubação. Contudo, diferente do que foi divulgado pela empresa, o adubo X-2020 não apresentou melhor desempenho que o adubo tradicional já utilizado pelos produtores.

7.6.2 ANOVA com dois fatores ou ANOVA fatorial

Este teste considera delineamentos amostrais com dois fatores (ou tratamentos) que podem ser compostos por dois ou mais grupos (ou níveis). Esta análise tem uma vantagem, pois permite avaliar o efeito da interação entre os fatores na variável resposta. Quando a interação está presente, o impacto de um fator depende do nível (ou grupo) do outro fator.

Exemplo prático 1 - ANOVA com dois fatores

Explicação dos dados

Neste exemplo hipotético, avaliaremos se o tempo que o corpo leva para eliminar uma droga utilizada em exames de ressonância magnética está relacionado com o sistema XY de determinação do sexo e/ou com a idade dos pacientes. Para isso, foi realizado um experimento com 40 pacientes distribuídos da seguinte maneira: i) 10 indivíduos XX - jovens, ii) 10 indivíduos XX - idosas, iii) 10 indivíduos XY - jovens, e iv) 10 indivíduos XY - idosos.

Pergunta

- O tempo de eliminação da droga é dependente do sistema XY de determinação do sexo e idade dos pacientes?

Predições

- O tempo de eliminação da droga vai ser mais rápido nos pacientes XX e jovens

Variáveis

- Data frame com os pacientes (unidade amostral) nas linhas e o tempo de eliminação da droga (variável resposta) e os tratamentos sexo e idade dos pacientes (variáveis preditoras) nas colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e as variáveis preditoras e respostas nas colunas

Análise

Verificando os dados usando a função `head()`.

```
## Cabeçalho dos dados
head(dados_dois_fatores)
#>    Tempo Pessoas Idade
#> 1 18.952      XX Jovem
#> 2 16.513      XX Jovem
#> 3 17.981      XX Jovem
#> 4 21.371      XX Jovem
#> 5 14.470      XX Jovem
#> 6 19.130      XX Jovem
```

Códigos para realizar a ANOVA com dois fatores.

```
## Análise Anova de dois fatores
# A interação entre os fatores é representada por *
```

```
Modelo1 <- aov(Tempo ~ Pessoas * Idade, data = dados_dois_fatores)

# Olhando os resultados
anova(Modelo1)
#> Analysis of Variance Table
#>
#> Response: Tempo
#>               Df  Sum Sq Mean Sq  F value    Pr(>F)
#> Pessoas        1  716.72  716.72 178.8538  1.56e-15 ***
#> Idade          1 1663.73 1663.73 415.1724 < 2.2e-16 ***
#> Pessoas:Idade  1    4.77    4.77   1.1903    0.2825
#> Residuals     36  144.26    4.01
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Percebam que a interação não apresenta um efeito significativo (P > 0.05). Assim, iremos retirar a interação e verificar, usando *Likelihood-ratio test*, se o modelo mais simples é melhor.

```
# Criando modelo sem interação.
Modelo2 <- aov(Tempo ~ Pessoas + Idade, data = dados_dois_fatores)

## LRT
lrtest(Modelo1, Modelo2)
#> Likelihood ratio test
#>
#> Model 1: Tempo ~ Pessoas * Idade
#> Model 2: Tempo ~ Pessoas + Idade
#>   #Df  LogLik Df  Chisq Pr(>Chisq)
#> 1   5 -82.413
#> 2   4 -83.063 -1 1.3012      0.254
```

Analisando o resultado do teste (P > 0.05), a interação não é importante. Então podemos seguir com o modelo mais simples. Vamos verificar a normalidade e homogeneidade da variância (Figura 7.13).

```
# Verificando as premissas do teste.
plot_grid(plot_model(Modelo2, type = "diag"))
```

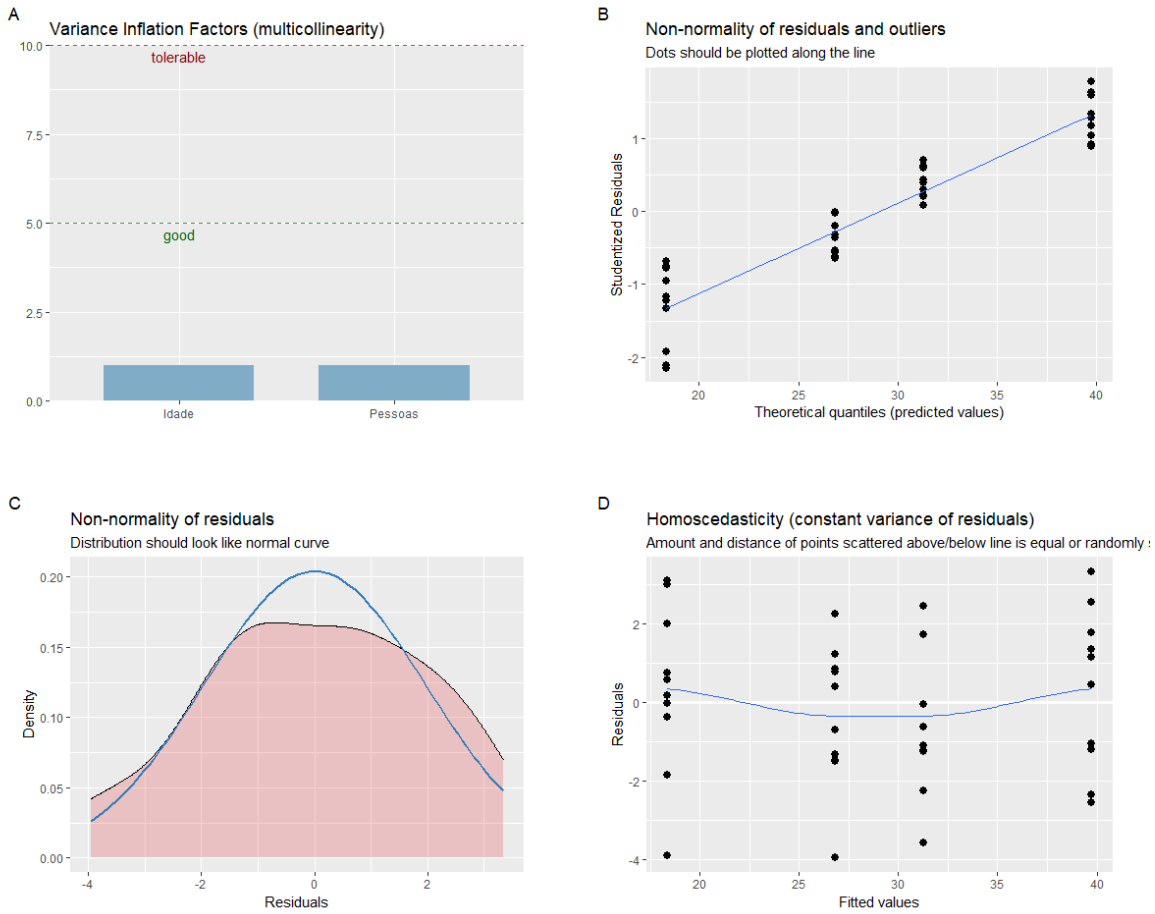


Figura 7.13: Gráficos mostrando as premissas da ANOVA fatorial.

Dois pontos estão fugindo da reta e chamam atenção sobre a normalidade da distribuição dos resíduos. A homogeneidade da variância está adequada. Por enquanto, vamos seguir a análise, mas veja o Capítulo 8 para entender como lidar com modelos que os resíduos não apresentam distribuição normal.

```
# Resultados do modelo
anova(Modelo2)
#> Analysis of Variance Table
#>
#> Response: Tempo
#>           Df  Sum Sq Mean Sq F value    Pr(>F)
#> Pessoas    1  716.72  716.72  177.94 1.041e-15 ***
#> Idade      1 1663.73 1663.73  413.05 < 2.2e-16 ***
#> Residuals 37  149.03    4.03
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Percebam que o resultado da ANOVA (Pr(>F) < 0.001) indica que devemos rejeitar a hipótese nula de que não há diferença entre as médias do sistema XY e idade dos pacientes. Neste caso, não precisamos realizar testes de comparações múltiplas *post-hoc* porque os fatores apresentam apenas dois níveis. Contudo, se no seu delineamento experimental um dos fatores apresentar três ou mais níveis, você deverá utilizar os testes de comparações *post-hoc* para determinar as diferenças entre os grupos.

Vamos visualizar os resultados em gráfico (Figura 7.14).

```
## Gráfico
ggplot(data = dados_dois_fatores_interacao,
       aes(y = Tempo, x = Pessoas, color = Idade)) +
    geom_boxplot() +
    stat_summary(fun = mean, geom ="point",
                 aes(group = Idade, x = Pessoas),
                 color = "black",
                 position = position_dodge(0.7), size  = 4) +
    geom_link(aes(x = 0.8, y = 31, xend = 1.8, yend = 40),
              color = "darkorange",
              lwd  = 1.3, linetype = 2) +
    geom_link(aes(x = 1.2, y = 19, xend = 2.2, yend = 26.5),
              color = "cyan4", lwd  = 1.3, linetype = 2) +
    labs(x = "Sistema XY de determinação do sexo",
         y = "Tempo (horas) para eliminar a droga") +
    scale_color_manual(values = c("darkorange", "cyan4",
                                  "darkorange", "cyan4")) +
    scale_y_continuous(limits = c(10, 50),
                       breaks = c(10, 20, 30, 40, 50)) +
    tema_livro()
```

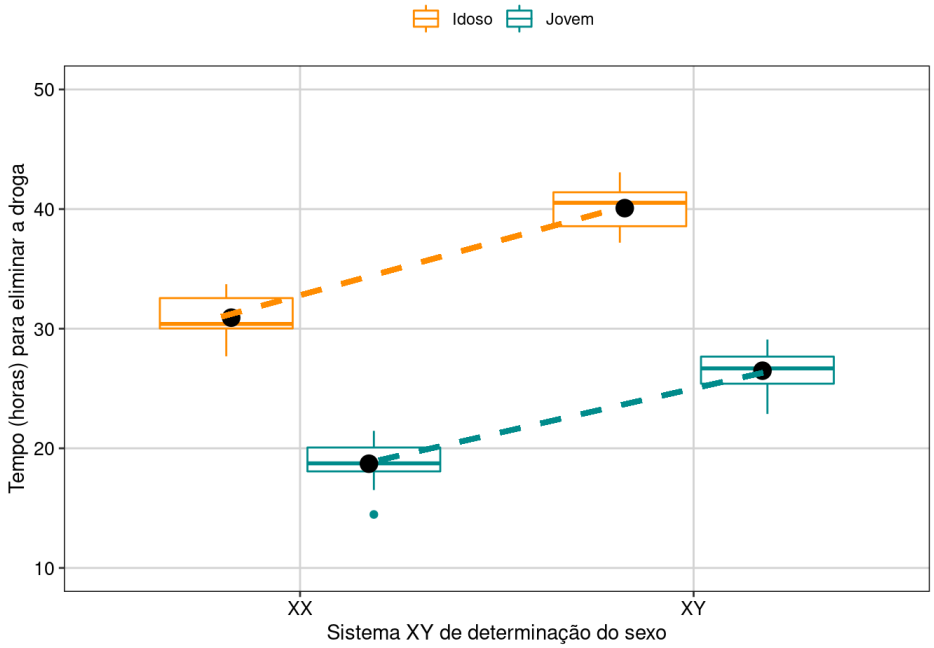


Figura 7.14: Gráfico de caixa mostrando o resultado da ANOVA fatorial.

Interpretação dos resultados

Neste exemplo, o sistema XY de determinação do sexo e a idade dos pacientes têm um efeito no tempo de eliminação da droga do organismo. Os pacientes XX e jovens apresentaram eliminação mais rápida da droga do que pacientes XY e idosos.

Exemplo prático 2 - ANOVA com dois fatores com efeito da interação

Explicação dos dados

Neste exemplo, novamente hipotético, usaremos os mesmos dados do exemplo anterior. Entretanto, alteramos os dados para que agora a interação seja significativa.

```
## Olhando os dados
head(dados_dois_fatores_interacao2)
#>    Tempo Pessoas Idade
#> 1 18.952      XX Jovem
#> 2 16.513      XX Jovem
#> 3 17.981      XX Jovem
#> 4 21.371      XX Jovem
#> 5 14.470      XX Jovem
#> 6 19.130      XX Jovem

## Análise anova de dois fatores
Modelo_interacao2 <- aov(Tempo ~ Pessoas * Idade,
                         data = dados_dois_fatores_interacao2)

## Olhando os resultados
anova(Modelo_interacao2)
#> Analysis of Variance Table
#>
#> Response: Tempo
#>               Df  Sum Sq Mean Sq  F value    Pr(>F)
#> Pessoas        1  716.72  716.72 178.8538  1.56e-15 ***
#> Idade          1    4.77    4.77   1.1903    0.2825
#> Pessoas:Idade  1 1663.73 1663.73 415.1724 < 2.2e-16 ***
#> Residuals     36  144.26    4.01
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Percebam que a interação é significativa ($P < 0.05$), mas a idade não é significativa. Nossa interpretação precisa ser baseada na interação entre os fatores. Vamos visualizar os resultados em gráfico (Figura 7.15).

```
## Gráfico
ggplot(data = dados_dois_fatores_interacao2,
       aes(y = Tempo, x = Pessoas, color = Idade)) +
    geom_boxplot() +
    stat_summary(fun = mean, geom ="point",
                 aes(group = Idade, x = Pessoas),
                 color = "black", position = position_dodge(0.7),
                 size  = 4) +
    geom_link(aes(x = 0.8, y = 31, xend = 1.8, yend = 27),
              color = "darkorange",
              lwd  = 1.3, linetype = 2) +
```

```
  geom_link(aes(x = 1.2, y = 19, xend = 2.2, yend = 41),
            color = "cyan4",
            lwd  = 1.3, linetype = 2) +
  labs(x = "Sistema XY de determinação do sexo",
       y = "Tempo (horas) para eliminar a droga") +
  scale_color_manual(values = c("darkorange", "cyan4",
                                "darkorange", "cyan4")) +
  scale_y_continuous(limits = c(10, 50),
                     breaks = c(10, 20, 30, 40, 50)) +
  tema_livro()
```

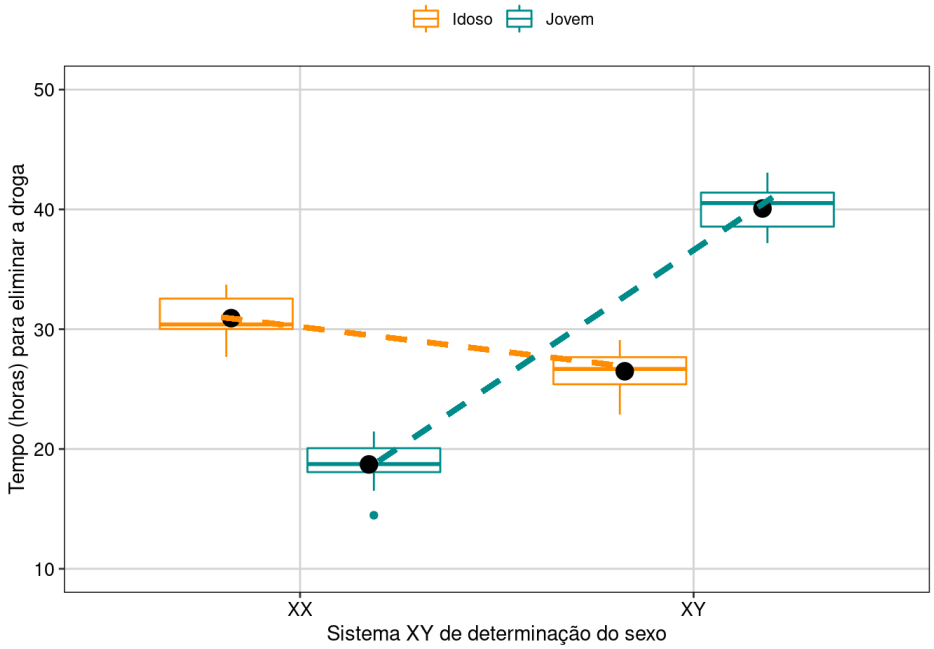


Figura 7.15: Gráfico de caixa mostrando o resultado da ANOVA fatorial com interação.

Interpretação dos resultados

Percebam que as linhas se cruzam. Esse é um exemplo clássico de interação. Novamente, para saber a resposta do fator idade (jovem ou idoso), você precisa saber com qual pessoa (XX ou XY) ele está associado. Jovens são mais rápidos para eliminarem a droga em pessoas XX, enquanto os idosos são mais rápidos para eliminarem a droga nas pessoas XY.

7.6.3 ANOVA em blocos aleatorizados

No delineamento experimental com blocos aleatorizados, cada fator é agrupado em blocos, com réplicas de cada nível do fator representado em cada bloco (Gotelli & Ellison 2012). O bloco é uma área ou período de tempo dentro do qual as condições ambientais são relativamente homogêneas. O objetivo do uso dos blocos é controlar fontes de variações indesejadas na variável dependente que não são de interesse do pesquisador. Desta maneira, podemos retirar dos resíduos os efeitos das variações indesejadas que não são do nosso interesse, e testar com maior poder estatístico os efeitos dos tratamentos de interesse.

> 👉 **Importante**
> Os blocos devem ser arranjados de forma que as condições ambientais sejam mais similares dentro dos blocos do que entre os blocos.

Exemplo prático 1 - ANOVA em blocos aleatorizados

Explicação dos dados

Neste exemplo, avaliaremos a riqueza de espécies de anuros amostradas em poças artificiais instaladas a diferentes distâncias de seis fragmentos florestais no sudeste do Brasil (da Silva et al. 2012). Os fragmentos florestais apresentam diferenças entre si que não são do interesse do pesquisador. Por isso, eles foram incluídos como blocos nas análises. As poças artificiais foram instaladas em todos os fragmentos florestais com base no seguinte delineamento experimental (da Silva et al. 2012): i) quatro poças no interior do fragmento a 100 m de distância da borda do fragmento; ii) quatro poças no interior no fragmento a 50 m de distância da borda do fragmento; iii) quatro poças na borda do fragmento; iv) quatro poças na matriz de pastagem a 50 m de distância da borda do fragmento; e v) quatro poças na matriz de pastagem a 100 m de distância da borda do fragmento. Percebam que todos os tratamentos foram instalados em todos os blocos.

> 👉 **Importante**
> Os valores da riqueza de espécies foram alterados em relação a publicação original (da Silva et al. 2012) para deixar o exemplo mais didático.

Pergunta

- A distância da poça artificial ao fragmento florestal influencia a riqueza de espécies anuros?

Predições

- Poças na borda do fragmento florestal apresentarão maior riqueza de espécies do que poças distantes da borda

Variáveis

- Data frame com as poças (unidade amostral) nas linhas e a riqueza de espécies (variável reposta), distância dos fragmentos florestais (variável preditora categórica) e fragmentos florestais (blocos) nas colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variáveis preditoras e respostas nas colunas

Análise

Olhando os dados usando a função `head()`.

```
## Cabeçalho dos dados
head(dados_bloco)
#>   Riqueza Blocos    Pocas
#> 1      90      A  Int-50m
#> 2      95      A Int-100m
#> 3     107      A    Borda
#> 4      92      A  Mat-50m
#> 5      89      A Mat-100m
#> 6      92      B  Int-50m
```

Análise da ANOVA em blocos.

```
## Análise Anova em blocos aleatorizados
model_bloco <- aov(Riqueza ~ Pocas + Error(Blocos), data = dados_bloco)
summary(model_bloco)
#>
#> Error: Blocos
#>           Df Sum Sq Mean Sq F value Pr(>F)
#> Residuals  5   1089   217.8
#>
#> Error: Within
#>           Df Sum Sq Mean Sq F value Pr(>F)
#> Pocas      4   1504   376.1   2.907 0.0478 *
#> Residuals 20   2588   129.4
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Lembre-se que nos delineamentos experimentais em bloco, o pesquisador não está interessado no efeito do bloco, mas sim em controlar a variação associada a ele.

O que não pode acontecer é ignorar o efeito do bloco que é incorporado pelos resíduos quando não informado no modelo. Veja abaixo a **forma errada** de analisar delineamento experimental com blocos.

```
## Forma errada de análisar Anova em blocos
modelo_errado <- aov(Riqueza ~ Pocas, data = dados_bloco)
anova(modelo_errado)
#> Analysis of Variance Table
#>
#> Response: Riqueza
#>           Df Sum Sq Mean Sq F value  Pr(>F)
#> Pocas      4 1504.5  376.12  2.5576 0.06359 .
#> Residuals 25 3676.5  147.06
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

O resultado da ANOVA (Pr(>F) < 0.001) indica que devemos rejeitar a hipótese nula que não há diferença entre as médias dos grupos. Contudo, os resultados não mostram quais são os grupos que apresentam diferenças. Para isso, temos que realizar testes de comparações múltiplas *post-hoc* para detectar os grupos que apresentam diferenças significativas entre as médias.

```
## Teste de Tuckey's honest significant difference
pairs(lsmeans(model_bloco, "Pocas"), adjust = "tukey")
#>  contrast                estimate   SE df t.ratio p.value
#>  Borda - (Int-100m)        16.000 6.57 20   2.436  0.1463
#>  Borda - (Int-50m)         19.833 6.57 20   3.020  0.0472
#>  Borda - (Mat-100m)        15.833 6.57 20   2.411  0.1531
#>  Borda - (Mat-50m)          8.167 6.57 20   1.244  0.7269
#>  (Int-100m) - (Int-50m)     3.833 6.57 20   0.584  0.9760
#>  (Int-100m) - (Mat-100m)   -0.167 6.57 20  -0.025  1.0000
#>  (Int-100m) - (Mat-50m)    -7.833 6.57 20  -1.193  0.7553
#>  (Int-50m) - (Mat-100m)    -4.000 6.57 20  -0.609  0.9720
#>  (Int-50m) - (Mat-50m)    -11.667 6.57 20  -1.777  0.4135
#>  (Mat-100m) - (Mat-50m)    -7.667 6.57 20  -1.167  0.7692
#>
#> P value adjustment: tukey method for comparing a family of 5 estimates
```

Visualizar os resultados em gráfico (Figura 7.16).

```
# Reordenando a ordem que os grupos irão aparecer no gráfico.
dados_bloco$Pocas <- factor(dados_bloco$Pocas,
                            levels = c("Int-100m", "Int-50m", "Borda",
                                       "Mat-50m", "Mat-100m"))

## Gráfico
ggplot(data = dados_bloco, aes(x = Pocas, y = Riqueza)) +
    labs(x = "Poças artificiais", y = "Riqueza de espécies de anuros") +
    geom_boxplot(color = "black", show.legend = FALSE, alpha = 0.4) +
    geom_jitter(shape = 16, position = position_jitter(0.1), cex = 4,
                alpha = 0.7) +
    scale_x_discrete(labels = c("-100m","-50m","Borda", "50m", "100m")) +
    tema_livro() +
    theme(legend.position = "none")
```

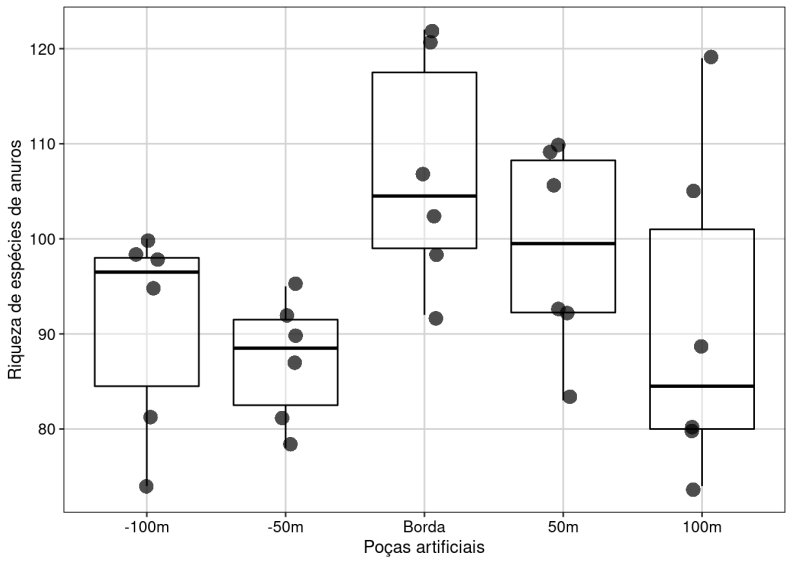


Figura 7.16: Gráfico de caixa mostrando o resultado da ANOVA de blocos aleatorizados.

Interpretação dos resultados

Neste exemplo, rejeitamos a hipótese nula de que a distância das poças artificiais até as bordas dos fragmentos florestais não influência a riqueza de espécies de anuros. As poças artificiais instaladas nas bordas dos fragmentos florestais apresentaram maior riqueza de espécies do que as poças distantes.

7.6.4 Análise de covariância (ANCOVA)

A ANCOVA pode ser compreendida como uma extensão da ANOVA com a adição de uma variável contínua (covariável) medida em todas as unidades amostrais (Gotelli & Ellison 2012). A ideia é que a covariável também afete os valores da variável resposta. Não incluir a covariável irá fazer com que a variação não explicada pelo modelo se concentre nos resíduos. Incluindo a covariável, o tamanho do resíduo é menor e o teste para avaliar as diferenças nos tratamentos, que é o interesse do pesquisador, terá mais poder estatístico.

Exemplo prático 1 - ANCOVA

Explicação dos dados

Neste exemplo hipotético, avaliaremos o efeito da herbivoria na biomassa dos frutos de uma espécie de árvore na Mata Atlântica. O delineamento experimental permitiu que alguns indivíduos sofressem herbivoria e outros não. Os pesquisadores também mediram o tamanho da raiz dos indivíduos para inseri-la como uma covariável no modelo.

Pergunta

- A herbivoria diminui a biomassa dos frutos?

Predições

- Os indivíduos que sofreram herbivoria irão produzir frutos com menor biomassa do que os indivíduos sem herbivoria

Variáveis

- Data frame com os indivíduos da espécie de planta (unidade amostral) nas linhas e a biomassa dos frutos (variável resposta), herbivoria (variável preditora categórica) e tamanho da raiz (covariável contínua) nas colunas

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas e variáveis preditoras e respostas nas colunas

Análise

Olhando os dados usando a função `head()`.

```
## Cabeçalho dos dados
head(dados_ancova)
#>    Raiz Biomassa Herbivoria
#> 1 6.225    59.77   Sem_herb
```

```
#> 2 6.487    60.98   Sem_herb
#> 3 4.919    14.73   Sem_herb
#> 4 5.130    19.28   Sem_herb
#> 5 5.417    34.25   Sem_herb
#> 6 5.359    35.53   Sem_herb
```

Códigos para o cálculo da ANCOVA (Figura 7.17).

```
## Ancova
modelo_ancova <- lm(Biomassa ~ Herbivoria * Raiz, data = dados_ancova)

# Verificando as premissas da Ancova
plot_grid(plot_model(modelo_ancova, type = "diag"))
```

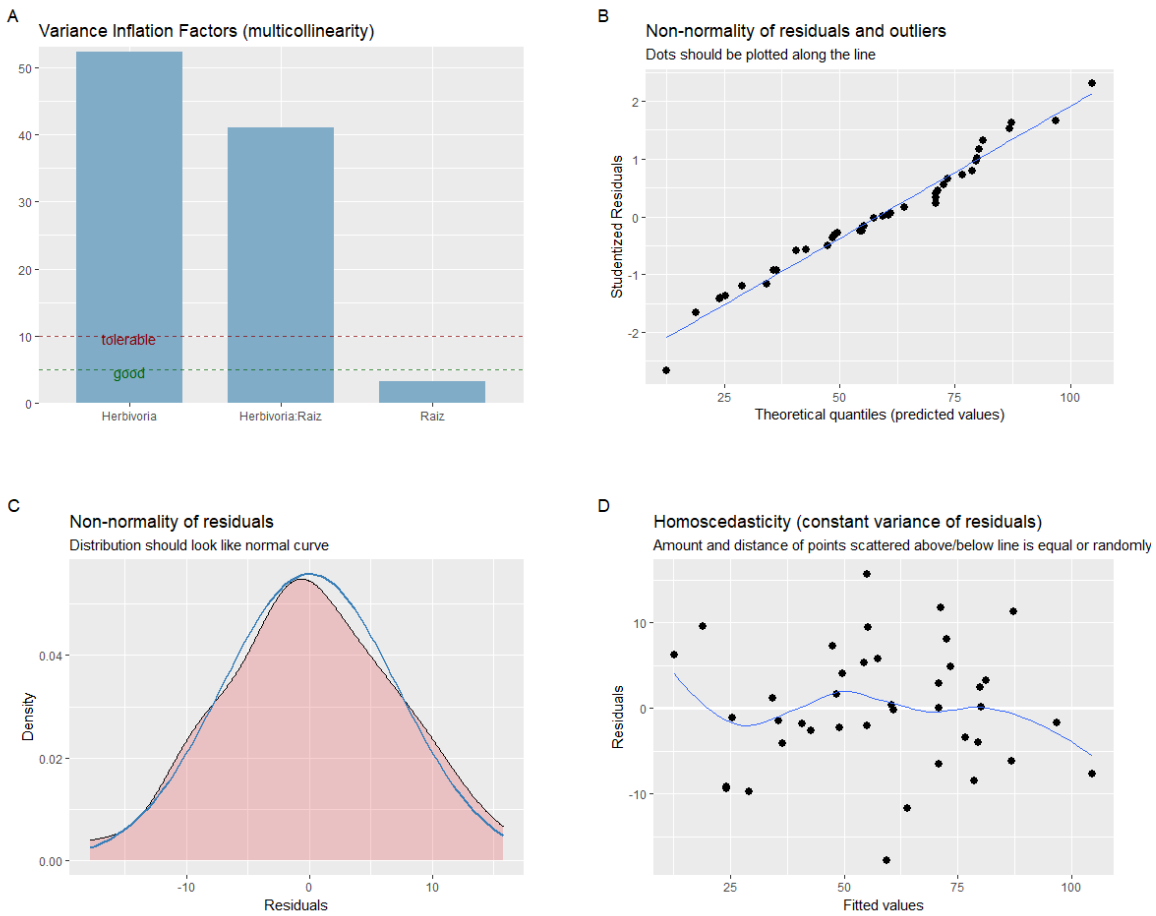


Figura 7.17: Gráficos de diagnóstico dos resíduos da ANCOVA.

As premissas da ANCOVA estão adequadas. Vamos olhar os resultados do modelo.

```
## Resultados do modelo
anova(modelo_ancova)
#> Analysis of Variance Table
#>
#> Response: Biomassa
```

```
#>                 Df  Sum Sq Mean Sq F value    Pr(>F)
#> Herbivoria       1  1941.9  1941.9  35.101 8.764e-07 ***
#> Raiz             1 17434.1 17434.1 315.124 < 2.2e-16 ***
#> Herbivoria:Raiz  1   136.7   136.7   2.471    0.1247
#> Residuals       36  1991.7    55.3
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Percebam que o resultado da ANCOVA (Pr(>F) < 0.001) indica que tanto a herbivoria como o tamanho da raiz (covariável) têm efeitos significativos na biomassa dos frutos. Contudo, a interação entre as variáveis não foi significativa. Vamos usar o *Likelihood-ratio test* para ver se podemos seguir com um modelo mais simples (sem interação).

```
## Criando modelo sem interação
modelo_ancova2 <- lm(Biomassa ~ Herbivoria + Raiz, data = dados_ancova)

## Likelihood-ratio test
lrtest(modelo_ancova, modelo_ancova2)
#> Likelihood ratio test
#>
#> Model 1: Biomassa ~ Herbivoria * Raiz
#> Model 2: Biomassa ~ Herbivoria + Raiz
#>   #Df  LogLik Df  Chisq Pr(>Chisq)
#> 1   5 -134.91
#> 2   4 -136.24 -1 2.6554     0.1032
```

A interação não é importante, pois P > 0.05. Seguiremos com o modelo mais simples.

Vamos fazer a visualizar os resultados em gráfico (Figura 7.18).

```
## Gráfico
ggplot(data = dados_ancova, aes(x = Raiz, y = Biomassa,
                                fill = Herbivoria)) +
    labs(x = "Tamanho da raiz (cm)", y = "Biomassa dos frutos (g)") +
    geom_point(size = 4, shape = 21, alpha = 0.7) +
    scale_colour_manual(values = c("darkorange", "cyan4")) +
    scale_fill_manual(values = c("darkorange", "cyan4"),
                      labels = c("Com herbivoria", "Sem herbivoria")) +
    geom_smooth(aes(color = Herbivoria), method = "lm",
                show.legend = FALSE) +
    tema_livro()
```

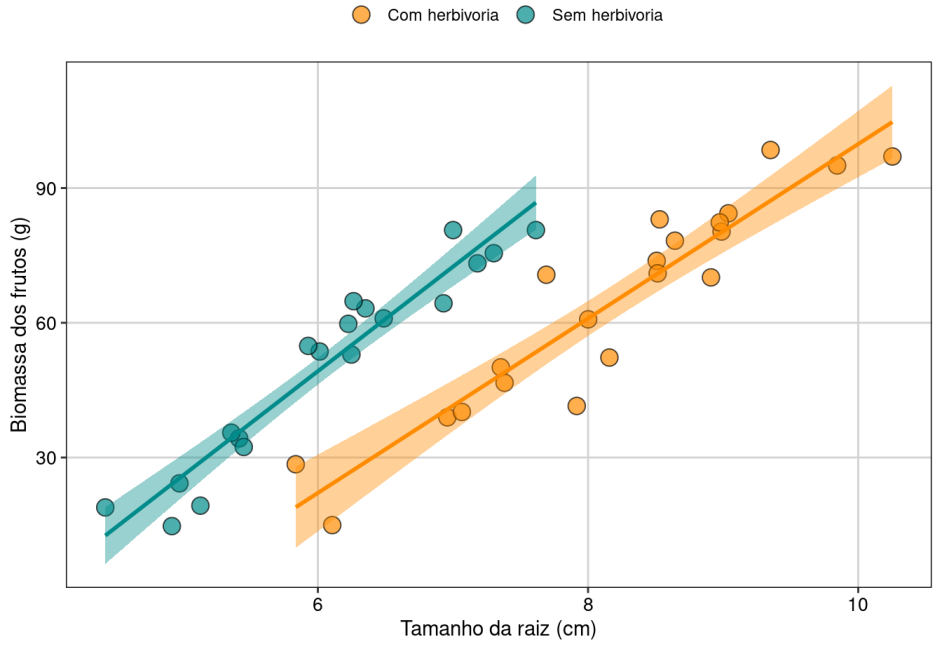


Figura 7.18: Gráfico de caixa mostrando o resultado da ANCOVA.

Interpretação dos resultados

Neste exemplo, o tamanho da raiz (covariável) tem uma relação positiva com a biomassa dos frutos. Quanto maior o tamanho da raiz, maior a biomassa dos frutos. Usando a ANCOVA e controlando o efeito da covariável, percebemos que a herbivoria também afeta a biomassa dos frutos. Os indivíduos com o mesmo tamanho de raiz que não sofreram herbivoria produziram frutos com maior biomassa do que os indivíduos com herbivoria.

7.7 Generalized Least Squares (GLS)

Em seu artigo clássico publicado em 1993, Pierre Legendre se pergunta se a autocorrelação espacial é um problema ou um novo paradigma (Legendre 1993). Segundo o autor, estudar estruturas espaciais é tanto uma necessidade, quanto um desafio para pesquisadores da ecologia e conservação que lidam com dados espacialmente distribuídos. Uma vez que todas as variáveis tipicamente utilizadas em estudos de biodiversidade (populações, condições climáticas, diversidade) possuem algum tipo de estrutura espacial, é fundamental compreender os motivos de como incluir esta informação nos modelos analíticos. De fato, a Primeira Lei da Geografia postula que "todas as coisas estão relacionadas com todas as outras, porém coisas próximas estão mais relacionadas do que coisas distantes". Como resultado, os valores observados em uma localidade (e.g., composição de espécies) serão mais afetados pelo conjunto de espécies que ocorre nas localidades vizinhas e, desse modo, alguns pontos de coleta podem não ser estatisticamente independentes. Como vimos anteriormente, um dos pressupostos dos modelos lineares é a independência das unidades amostrais. Assim, a presença de autocorrelação espacial nos resíduos viola este pressuposto e, consequentemente, aumenta a taxa de Erro do Tipo I (rejeitar a hipótese nula quando ela é verdadeira) nos modelos. Uma das soluções para incorporar a dependência espacial dos resíduos é usar o método de Mínimos Quadrados Generalizados (*Generalized Least Squares* - GLS). Diferente dos modelos apresentados anteriormente, este método ajusta explicitamente modelos heteroscedásticos e com resíduos correlacionados (Pinheiro & Bates 2000).

Para representar a estrutura espacial (e, assim, a dependência entre as observações) é necessário incluir variáveis espaciais (geralmente coordenadas geográficas, i.e., vetores bidimensionais: *sensu* Pinheiro & Bates (2000)) no argumento `corStruct` na função `gls()` do pacote `nlme`. Este modelo basicamente assume que a estrutura de covariância é uma função da distância entre as localidades (Littell et al. 2006). É importante ressaltar, todavia, que existem diferentes funções de covariância que são discutidas detalhadamente em Littell et al. (2006). Aqui, iremos nos concentrar nas seguintes funções.

- **Esférica**: `corSpher(form=\~lat+long)`
- **Exponencial**: `corExp(form=\~lat+long)`
- **Gaussiana**: `corGaus(form=\~lat+long)`
- **Linear**: `corLin(form=\~lat+long)`
- **Razão quadrática**: `corRatio(form=\~lat+long)`

Exemplo prático 1 - GLS

Explicação dos dados

Neste exemplo, utilizamos os dados de riqueza de espécies de ácaros (Oribatidae) em 70 amostras de musgo (gênero *Sphagnum*) (Borcard et al. 1992). Para cada amostra, além da riqueza de ácaros, os autores registraram a quantidade de água no substrato e as coordenadas geográficas. Os dados completos estão disponíveis no pacote `vegan`.

```
## Calcular a riqueza de espécies em cada comunidade
riqueza <- specnumber(mite)

## Selecionar a variável ambiental - quantidade de água no substrato
agua <- mite.env[,2]

## Criar um data.frame com riqueza, quantidade de água no substrato e
coordenadas geográficas
mite_dat <- data.frame(riqueza, agua, coords)
```

Modelo linear sem incorporar a estrutura espacial

Vamos inicialmente ajustar um modelo sem incorporar a estrutura espacial (Figura 7.19).

```
## Modelo
linear_model <- lm(riqueza ~ agua, mite_dat)

## Resíduos
par(mfrow = c(2, 2))
plot(linear_model, which = 1:4)

## Resultados do modelo
res_lm <- summary(linear_model)

## Coeficiente de determinação e coeficientes
res_lm$adj.r.squared
#> [1] 0.4632024
res_lm$coefficients
```

```
#>               Estimate  Std. Error  t value     Pr(>|t|)
#> (Intercept) 24.3040255 1.249094710 19.45731 1.687907e-29
#> agua        -0.0223793 0.002876239 -7.78075 5.481035e-11
```

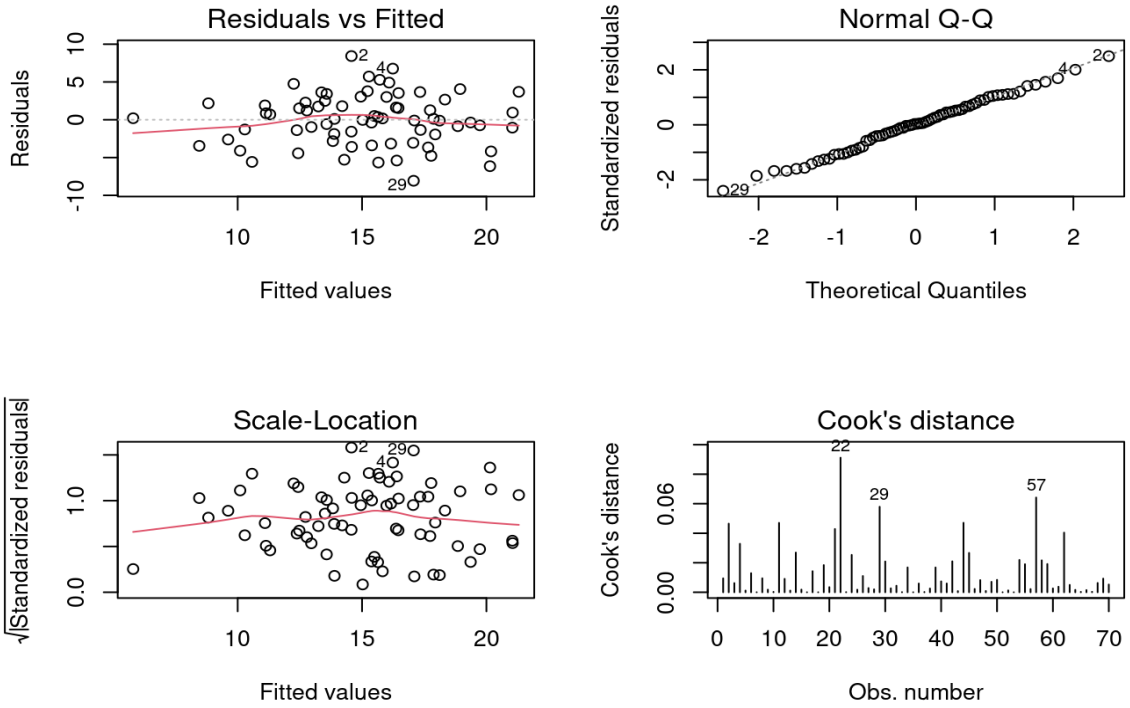


Figura 7.19: Gráficos mostrando as premissas da regressão linear simples.

Acessando a informação espacial com o GLS

Como dito, dependendo da estrutura espacial de suas variáveis (dependentes, independentes, covariáveis), o pressuposto de independência dos resíduos pode ser afetado e, desse modo, o modelo linear convencional terá maior chance de Erro do Tipo I. Abaixo, iremos comparar um modelo GLS sem incorporar estrutura espacial (o que é exatamente igual ao modelo criado acima *Modelo Linear*) com diferentes modelos que utilizam explicitamente resíduos correlacionados.

```
## Modelo gls sem estrutura espacial
no_spat_gls <- gls(riqueza ~ agua, mite_dat, method = "REML")
```

Uma maneira de identificar se os resíduos do modelo linear apresentam estrutura espacial é fazendo uma figura chamada **variograma**. O variograma possui três parâmetros: i) *nugget*, ii) *range* e iii) *sill* (Fortin & Dale 2005). O *nugget* é utilizado para quantificar a variabilidade observada nos valores menores (ou seja, em pequenas distâncias). O *range*, por sua vez, é usado para identificar a distância máxima em que a autocorrelação espacial está presente (Figura 7.20, pontos laranjas). Deste modo, os valores posicionados a partir do *range* (Figura 7.20, pontos verdes) representam pontos não correlacionados (Fortin & Dale 2005). A posição limiar que representa claramente a "pausa" no crescimento da curva (*range*) indica os pontos não correlacionados e representa o *sill* (Figura 7.20). No exemplo da Figura 7.20, o *sill* é constante. Porém, é possível que os valores de *sill* não sejam constantes (Chiles & Delfiner 1999). Um exemplo é o "efeito buraco" que é caracterizado por um ou mais picos (ou vales) no variograma que correspondem ao número de valores negativos na covariância. Esses valores

sugerem que valores altos podem estar rodeados de valores baixos (Chiles & Delfiner 1999). Porém, os detalhamentos desses comportamentos vão além do escopo deste livro.

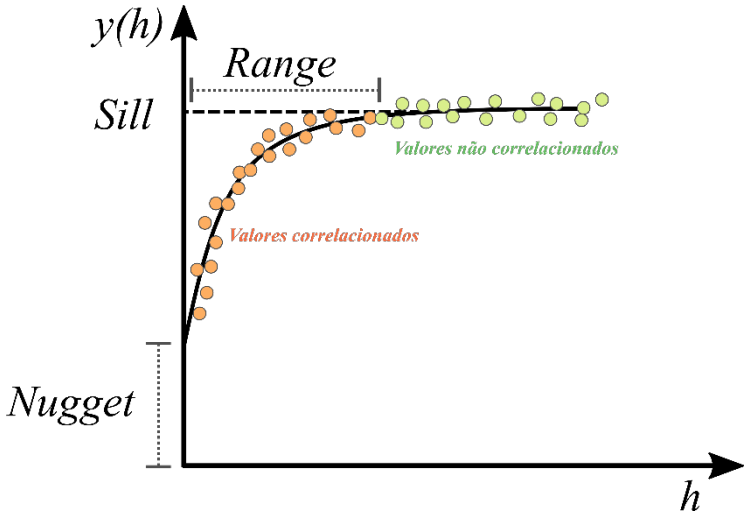


Figura 7.20: Variograma representando a semi-variância y(h) em função do intervalor espacial h. Cada ponto representa a distância entre localidades e a linha a variação teórica a função de covariância esférica (veja abaixo). Adaptado de Fortin & Dale (2005).

Abaixo, podemos analisar o variograma para os resíduos do modelo GLS ajustado (Figura 7.21).

```
## Variograma
variog_mod1 <- nlme::Variogram(no_spat_gls, form = ~lat+long,
                               resType = "normalized")

## Gráfico
plot(variog_mod1)

## Índice I de Moran

## Primeiro precisamos calcular uma matriz de distâncias geográficas entre as
comunidades
dat_dist <- pdist(coords) # matriz de distância

Moran.I(x = mite_dat$riqueza, w = dat_dist)
#> $observed
#> [1] -0.2398147
#>
#> $expected
#> [1] -0.01449275
#>
#> $sd
#> [1] 0.009971923
#>
#> $p.value
#> [1] 4.783614e-113
```

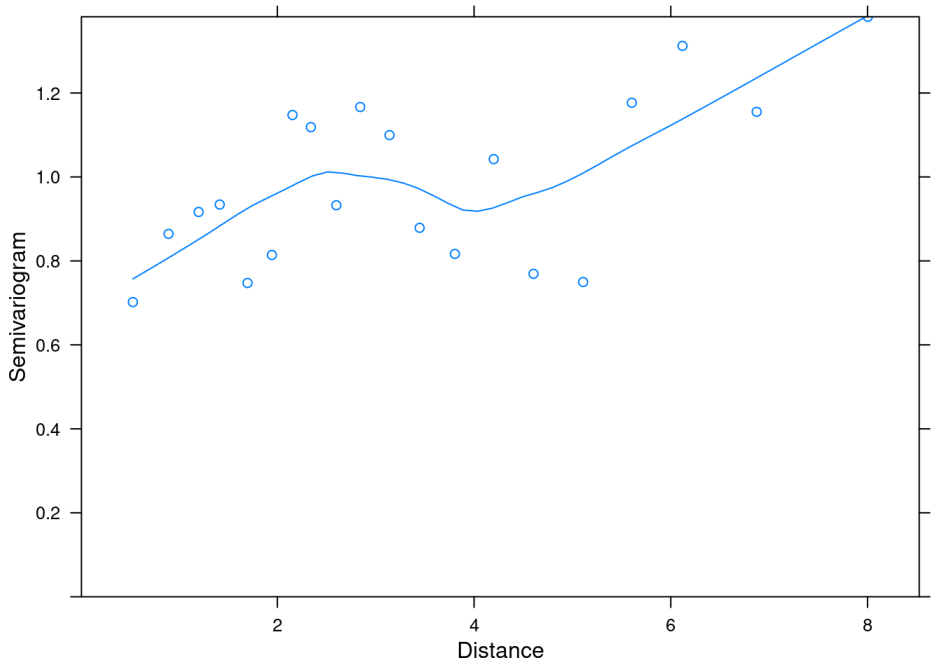


Figura 7.21: Variograma para os resíduos do modelo GLS ajustado.

O primeiro passo é utilizar diferentes variogramas teóricos para identificar o melhor modelo que representa a estrutura espacial dos seus dados.

```
## Covariância esférica
espher_model <- gls(riqueza ~ agua, mite_dat,
                    corSpher(form = ~lat+long, nugget = TRUE))

## Covariância exponencial
expon_model <- gls(riqueza ~ agua, mite_dat,
                   corExp(form = ~lat+long, nugget = TRUE))

## Covariância Gaussiana
gauss_model <- gls(riqueza ~ agua, mite_dat,
                   corGaus(form = ~lat+long, nugget = TRUE))

## Covariância linear
cor_linear_model <- gls(riqueza ~ agua, mite_dat,
                        corLin(form = ~lat+long, nugget = TRUE))

## Covariância razão quadrática
ratio_model <- gls(riqueza ~ agua, mite_dat,
                   corRatio(form = ~lat+long, nugget = TRUE))
```

Agora, depois de ajustar o modelo GLS com os diferentes variogramas é necessário comparar os modelos para escolher o mais "provável," utilizando a seleção de modelos pelo **Critério de Seleção de Akaike (AIC)** (Chiles & Delfiner 1999, Burnham & Anderson 2002, Fortin & Dale 2005) (Figura 7.22). AIC é um método estatístico que compara os modelos criados na sua pesquisa e seleciona o melhor entre eles. Diferente do LRT que compara apenas modelos aninhados, o AIC compara modelos com diferentes combinações de variáveis preditoras. Uma vez que a fórmula do AIC penaliza parâmetros

extras no modelo, se dois modelos explicam igualmente a variação dos dados, o modelo com menos parâmetros terá o menor valor de AIC e será selecionado como o melhor modelo.

> **Importante**
> O valor de AIC por si só não tem significado. Ele precisa ser comparado com outro modelo. Contudo, um modelo só será considerado superior a outro quando a diferença entre os seus valores de AIC (i.e, *delta*) forem maiores do que 2. (Burnham & Anderson 2002)

```
## Seleção de modelos
aic_fit <- AIC(no_spat_gls, espher_model, expon_model, gauss_model,
               cor_linear_model, ratio_model)
aic_fit %>% arrange(AIC)
#>   df      AIC
#> 1  5 373.1186
#> 2  5 373.2439
#> 3  3 383.8616
#> 4  5 387.8616
#> 5  5 387.8616
#> 6  5 387.8616

## Gráfico
plot(residuals(ratio_model, type = "normalized") ~ fitted(ratio_model))
```

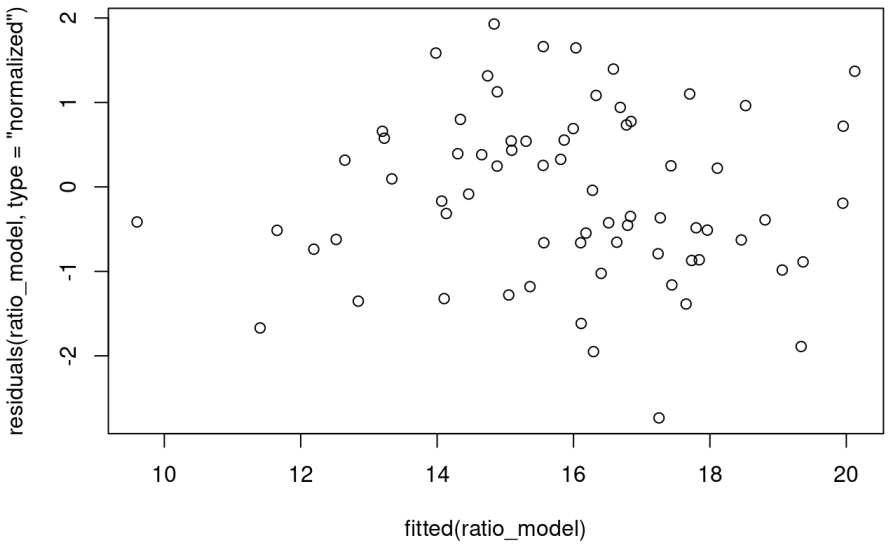


Figura 7.22: Gráfico dos resíduos em relação aos dados ajustados do modelo GLS Ratio.

O variograma ajustado pelo modelo razão quadrática (*ratio_model*) demonstra que o *range* não é crescente (indicando correlação espacial entre localidades próximas) e, desse modo, sugere que é mais apropriado usar o modelo GLS do que um modelo linear desconsiderando a estrutura espacial (Figura 7.23).

```
## Varigrama
ratio_variog <- Variogram(ratio_model, form = ~lat+long,
                                       resType = "normalized")

## Resumo dos modelos
summary(ratio_model)$tTable
#>                 Value   Std.Error   t-value      p-value
#> (Intercept) 22.1612850 2.444329887  9.066405 2.561826e-13
#> agua        -0.0151894 0.003505891 -4.332536 4.976757e-05
summary(no_spat_gls)$tTable
#>                 Value   Std.Error  t-value      p-value
#> (Intercept) 24.3040255 1.249094710 19.45731 1.687907e-29
#> agua        -0.0223793 0.002876239 -7.78075 5.481035e-11

## Gráficos
plot(ratio_variog, main = "Variograma como Modelo Ratio")
plot(variog_mod1, main = "Variograma Modelo Normal")
```

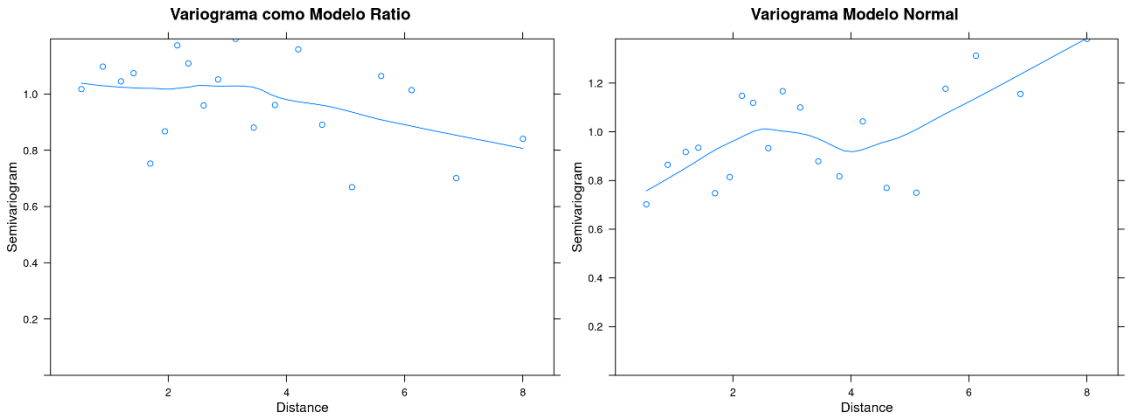


Figura 7.23: Variogramas para os resíduos do modelo GLS Normal e Ratio.

Interpretação dos resultados

Dessa forma, O valor mais apropriado da estatística t é de -4.33 (*ratio_model*) ao invés de -7.78 (*no_spat_gls*). Neste caso, a decisão (quantidade de água afetando a riqueza de ácaros) não foi afetada ($P < 0.05$ nos dois modelos), somente a estatística do teste.

7.8 Para se aprofundar

Listamos abaixo livros e links que recomendamos para seguir com sua aprendizagem em modelos lineares.

7.8.1 Livros

- Recomendamos aos interessados(as) os livros: i) Zar (2010) *Biostatiscal analysis*, ii) Gotelli & Ellison (2012) *A primer of ecological statistics*, iii) Quinn & Keough (2002) *Experimental*

design and data analysis for biologists, iv) Zuur e colaboradores (2007) *Analysing Ecological Data* e v) Touchon (2021) *Applied statistics with R: a practical guide for the life sciences*.

7.8.2 Links

- Recomendamos a página Curso de Estatística para Pesquisa Científica do pesquisador Thiago Rangel, Universidade Federal de Goiás. Nesta página, vocês irão encontrar exemplos e explicações detalhadas sobre as análises e a filosofia estatística.

7.9 Exercícios

7.1 Avalie se os indivíduos machos de uma espécie de aranha são maiores do que as fêmeas. Qual a sua interpretação sobre o dimorfismo sexual nesta espécie? Faça um gráfico boxplot usando também a função `geom_jitter()`. Use os dados `Cap7_exercicio1` disponível no pacote `ecodados`.

7.2 Avalie se o número de polinizadores visitando uma determinada espécie de planta é dependente da presença ou ausência de predadores. A mesma planta, em tempos diferentes, foi utilizada como unidade amostral para os tratamentos com e sem predadores. Qual a sua interpretação sobre os resultados? Faça um gráfico boxplot ligando os resultados da mesma planta com e sem a presença do predador. Use os dados `Cap7_exercicio2` disponível no pacote `ecodados`.

7.3 Avalie se existe correlação entre o número de filhotes nos ninhos de uma espécie de ave com o tamanho do fragmento florestal. Qual a sua interpretação dos resultados? Faça um gráfico mostrando a relação entre as variáveis. Use os dados `Cap7_exercicio3` disponível no pacote `ecodados`.

7.4 Avalie se a relação entre o tamanho da área de diferentes ilhas e a riqueza de espécies de lagartos. Qual a sua interpretação dos resultados? Faça um gráfico mostrando a relação predita pelo modelo. Use os dados `Cap7_exercicio4` disponível no pacote `ecodados`.

7.5 Avalie se existe relação entre a abundância de uma espécie de roedor com o tamanho da área dos fragmentos florestais e/ou a altitude. Faça uma regressão múltipla. Em seguida, crie diferentes modelos e selecione o mais parcimonioso com base nos valores do teste de *Likelihood-ratio test* e *Akaike information criterion*. Qual a sua interpretação? Use os dados `Cap7_exercicio5` disponível no pacote `ecodados`.

7.6 Avalie se o local que machos territoriais ocupam (pasto, cana, floresta) influência no peso dos indivíduos. Qual a sua interpretação dos resultados? Faça um gráfico com os resultados. Use os dados `Cap7_exercicio6` disponível no pacote `ecodados`.

7.7 Avalie se a abundância de formigas está relacionada com o fato das domácias estarem abertas ou fechadas e com a idade das domácias. Verifique a interação entre os fatores. Qual a sua interpretação dos resultados? Faça um gráfico com os resultados. Use os dados `Cap7_exercicio7` disponível no pacote `ecodados`.

7.8 Avalie se o número de parasitas está relacionado com o tamanho corporal de fêmeas de uma espécie de ave. Além disso, use a idade das aves como uma covariável explicando o número de parasitas. Qual a sua interpretação dos resultados? Faça um gráfico com os resultados. Use os dados `Cap7_exercicio8` disponível no `ecodados`.

7.9 Avalie se a presença ou ausência de predadores afeta a riqueza de macroinvertebrados em 10 lagos. Os tratamentos dos predadores foram realizados nos mesmos lagos. Qual a sua interpretação dos resultados? Faça um gráfico com os resultados. Use os dados `Cap7_exercicio9` disponível no pacote `ecodados`.

7.10 Avalie se a precipitação anual afeta a riqueza de espécies de anuros em 44 localidades na Mata Atlântica. Use as coordenadas geográficas para controlar o efeito da autocorrelação espacial. Qual a sua interpretação dos resultados das análises com e sem levar em consideração a autocorrelação espacial? Use os dados `anuros_ambientais` disponível no pacote `ecodados`.

Soluções dos exercícios.

Capítulo 8

Modelos Lineares Generalizados

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
library(ecodados)
library(visdat)
library(tidyverse)
library(lattice)
library(RVAideMemoire)
library(DHARMa)
library(performance)
library(MuMIn)
library(piecewiseSEM)
library(MASS)
library(ggExtra)
library(Rmisc)
library(emmeans)
library(sjPlot)
library(bbmle)
library(glmmTMB)
library(ordinal)
library(car)
library(ecolottery)
library(naniar)
library(vcd)
library(generalhoslem)

## Dados
lagartos <- ecodados::lagartos
parasitas <- ecodados::parasitas
fish <- ecodados::fish
fragmentos <- ecodados::fragmentos
uv_cells <- ecodados::uv_cells
cores <- ecodados::cores
```

8.1 Introdução

No Capítulo 7, descrevemos sobre os modelos lineares (também chamados de Modelos Lineares Gerais) que podem ser descritos pelo mesmo modelo matemático de uma equação da reta do tipo:

$$Y = \beta_0 + \beta_1 X_i + \epsilon_i$$

Nesse tipo de estrutura, o que difere uma regressão linear de uma análise de variância é a natureza do elemento x_i, variável contínua para a regressão linear e variável categórica no caso da ANOVA (que é codificada numa matriz *design* para desenhos mais complexos). Nesse sentido, o que todos esses métodos têm em comum é a variável resposta Y que é um vetor numérico contínuo. Outro elemento em comum desses métodos é a distribuição de frequência do erro. Se quiser mais detalhes sobre como

modelos lineares podem ser escritos na forma de matrizes, consulte a introdução de (Fox et al. 2015). Todos os modelos lineares assumem que a distribuição do erro seja Gaussiana (ou Normal). Isso de certa forma limita o tipo de dado que pode ser usado como variável resposta por estas análises. Por exemplo, dados de contagem (e.g., riqueza e/ou abundância de espécies), frequência (e.g., frequência de ocorrência, porcentagem de cobertura vegetal), incidência (e.g., presença ou ausência de uma espécie) ou proporção (e.g., números de animais infectados a cada 1000 animais) não são adequados para serem utilizados como variáveis resposta em modelos lineares. Uma prática comum quando nossos dados não são Normais é transformar por logaritmo ou raiz quadrada. No entanto, para dados de contagem isso não é recomendado - veja O'Hara & Kotze (2010), Ives (2015) e Warton (2018) para mais detalhes.

Nestes casos, devemos recorrer a um conjunto de modelos chamados Modelos Lineares Generalizados (GLM). Nestes modelos, o usuário especifica a distribuição de frequência que deseja utilizar para modelar a variável resposta. Esta distribuição de frequência deve pertencer à família exponencial, que inclui a distribuição de Poisson, Gaussiana, binomial, binomial negativa, Gamma, Bernoulli e Beta. Ainda é possível utilizar *Cumulative Link Models* para modelar dados ordinais (fatores cuja ordem dos elementos importa, tais como muito baixo, baixo, alto e muito alto). Abaixo vamos ver um pouco sobre como um GLM funciona e exemplos com cada uma destas distribuições.

8.2 Como um GLM funciona?

Diferentemente do modelo linear, um GLM estima os parâmetros por meio de Máxima Verossimilhança (ML) ao invés dos Mínimos Quadrados Comuns, também chamados de Mínimos Quadrados Ordinários (OLS).

Portanto, um GLM relaciona a **distribuição da variável resposta** aos **preditores lineares** por meio de uma **função de ligação**. Por exemplo, no caso da distribuição de Poisson, usa-se uma ligação logarítmica (também chamada de log link) que garante que o valores ajustados são sempre não negativos. Portanto, um GLM é composto por esses três componentes: i) função de distribuição, ii) preditor linear e iii) função de ligação. A função de distribuição é uma hipótese sobre a distribuição da variável resposta Y_i. Isso também define a média e a variância de Y_i. Já a função de ligação define a relação entre o valor médio de Y_i e da parte sistemática. Esta é também chamada de ligação entre a média e a parte sistemática do modelo. Existem três tipos de função de ligação:

- **Identity link**, que é definido por `g(μ) = μ`, e modela a média ou valor esperado de Y. Usado em modelos lineares padrão
- **Log link**, que é `g(μ) = log(μ)`, e modela o log da média. É usado para dados de contagem (que não podem assumir valores negativos) em modelos log-linear
- **Logit link**, que é `g(μ) = log[μ/(1-μ)]`, e é usado para dados binários e regressão logística

Logo, um modelo linear pode ser visto como um caso particular de um GLM em que utiliza distribuição Gaussiana, com *identity link*.

8.3 Como escolher a distribuição correta para seus dados?

8.3.1 Para dados contínuos

Se Y é uma variável contínua, a sua distribuição de probabilidade deve ser normal. Nesses casos as distribuições recomendadas são a **Gaussiana (Normal) ou Gamma**. Para essas distribuições, o parâmetro de dispersão é estimado separadamente da média e é às vezes, chamado de *nuisance parameter*. Uma particularidade da distribuição Gamma é que ela só aceita valores contínuos positivos.

Se Y é uma variável de proporção que varia continuamente entre 0 e 1 ou 0 e 100, mas não inclui 0 nem 1, a distribuição recomendada é a beta. Um exemplo é a área (Km^2) de floresta numa imagem de satélite, cuja área total é conhecida. Caso a variável inclua 0 e 1 (e.g., 100% da imagem é composta por floresta), deve-se realizar uma transformação nos dados (Smithson & Verkuilen 2006).

8.3.2 Para dados de contagem

Se Y é binário (e.g., vivo ou morto), a distribuição de probabilidade deve ser **binomial**.

Se Y é uma contagem (e.g., abundância ou riqueza de espécies), então a distribuição de probabilidade deve ser **Poisson ou binomial negativa**. Existem também correções dessas distribuições quando apresentam sobredispersão (*overdispersion*), tais como quasi-Poisson ou quasi-Negative binomial. Falaremos delas no momento certo.

Para distribuições tais como binomial e Poisson, a variância deve ser igual à média e o parâmetro de dispersão é sempre 1. Na maioria dos dados ecológicos esse pressuposto não é cumprido, veremos estratégias para lidar com isso logo à frente.

As funções `Ord_plot()` e `goodfit()` do pacote `vcd` podem auxiliar na escolha da distribuição para dados de contagem.

8.4 Dados de contagem: a distribuição de Poisson

Para casos em que estamos interessados em quantificar uma variável discreta, ou seja, uma variável positiva, representada sempre por números inteiros, contendo um número finito de possibilidades, devemos utilizar a **distribuição de Poisson**. Esta distribuição é peculiar por ser descrita apenas por um parâmetro livre (λ). Isso quer dizer que tanto a média quanto a variância dos dados são descritos por um único parâmetro, o que implica em dizer que a média e a variância têm de ser iguais.

Vamos ver um exemplo com dados reais.

Exemplo 1

Explicação dos dados

Neste exemplo, iremos utilizar dados de riqueza de anfíbios anuros coletados em 40 poças, açudes e brejos ao redor de fragmentos florestais no Noroeste Paulista (Prado & Rossa-Feres 2014). Os autores mediram seis variáveis em escala local e outras três em escala de paisagem.

Pergunta

A distância linear para o corpo d'água mais próximo influencia a abundância total de espécies de anuros?

Predições

Corpos d'água mais conectados permitem que indivíduos dispersem entre eles com maior facilidade, suportando melhor dinâmicas de metapopulações. Portanto, espero que poças que estejam mais conectadas entre si tenham maior riqueza total de anuros.

Variáveis

- Variável reposta: riqueza de anuros em 40 poças
- Variável preditora: distância da poça focal para a mais próxima na escala da paisagem

Checklist

- Verificar se o seu data frame está com as unidades amostrais nas linhas (neste caso poças) e variáveis nas colunas

Antes de começar com a análise, vamos primeiro explorar os dados.

```
## Explorar os dados
glimpse(fragmentos)
```

Percebam que o data frame contém 40 colunas. Neste conjunto de dados as variáveis preditoras já estão padronizadas com média 0 e desvio padrão 1. As variáveis com "2" indicam variáveis quadráticas (podem ser usadas para se testar relações não lineares). Também temos a riqueza observada e a estimada (`Riqueza_HB`) e as coordenadas geográficas (X e Y). Vamos agora explorar os dados e ver como é a relação entre riqueza e distância para a poça mais próxima. Sempre é recomendado visualizar os dados antes de efetivamente os modelar para se ter uma ideia da relação entre as variáveis (Figura 8.1).

```
## Gráfico
ggplot(fragmentos, aes(dfrag, Riqueza_obs)) +
    geom_point(size = 4, alpha = 0.7) +
    geom_smooth(method = "lm") +
     labs(x = "Distância para o fragmento mais próximo",
          y = "Riqueza observada") +
    tema_livro()
```

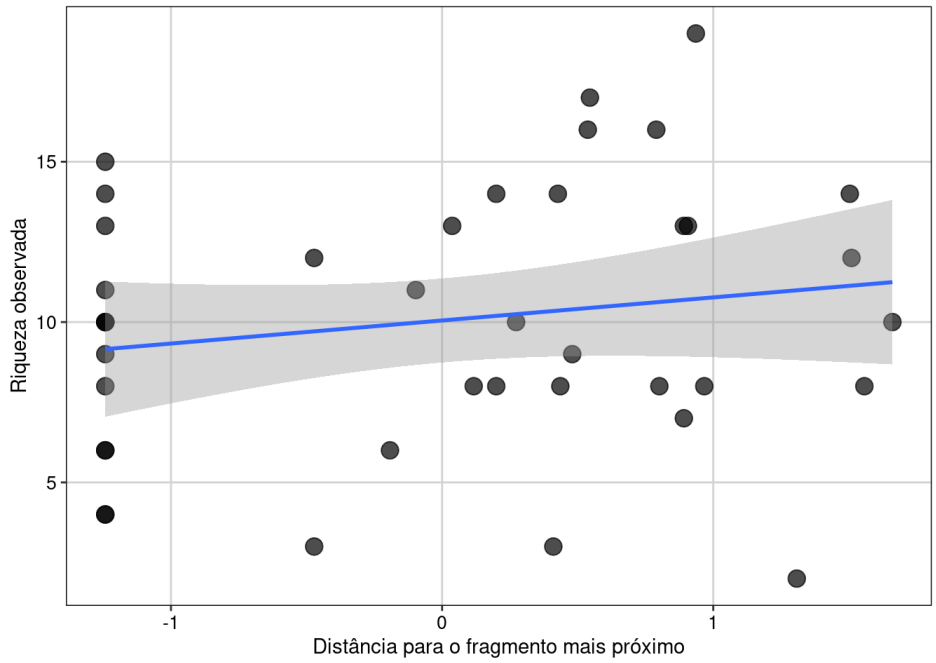


Figura 8.1: Gráfico de dispersão para explorar as variáveis.

Aqui vemos que há de fato uma relação linear positiva entre as duas variáveis.

- A partir de agora vamos sempre usar uma mesma estrutura para realizar nossos exercícios de modelagem:

1. Primeiro vamos especificar o modelo
2. Depois realizar a diagnose
3. Por último realizar inferência a partir do nosso modelo

Modelagem

O primeiro argumento da função `glm()` é uma fórmula, em que na parte esquerda temos a variável resposta seguida do símbolo `~` (lê-se: modelado em função de) seguido pelas variáveis preditoras. Aqui podemos usar uma ou mais variáveis e testar o seu efeito aditivo (usando o sinal de +) ou a interação entre elas (usando o sinal de *). Um bom resumo sobre como especificar o seu modelo pode ser encontrada aqui neste blog. Aqui optamos por um modelo bem simples modelando a riqueza de anfíbios apenas em função da distância para o fragmento mais próximo.

```
## Modelo
mod_pois <- glm(Riqueza_obs ~ dfrag, family = poisson(link = "log"),
                data = fragmentos)
```

Assim como modelos lineares que vimos no Capítulo 7, GLMs com distribuição de Poisson requerem que se realizem testes de pressupostos de normalidade e homogeneidade de variância dos resíduos, assim como sobredispersão e inflação de zeros.

Diagnose básica dos resíduos do modelo

Iremos realizar três diagnoses básicas dos GLMs, avaliando diferentes aspectos do modelo:

1. Homogeneidade da variância e normalidade dos resíduos
2. Sobredispersão (*Overdispersion*)

3. Inflação de zeros (*Zero-inflation*)

Vamos começar avaliando a homogeneidade da variância e a normalidade dos resíduos (Figura 8.2).

```
## Diagnose básica
par(mfrow = c(2, 2))
plot(mod_pois)
```

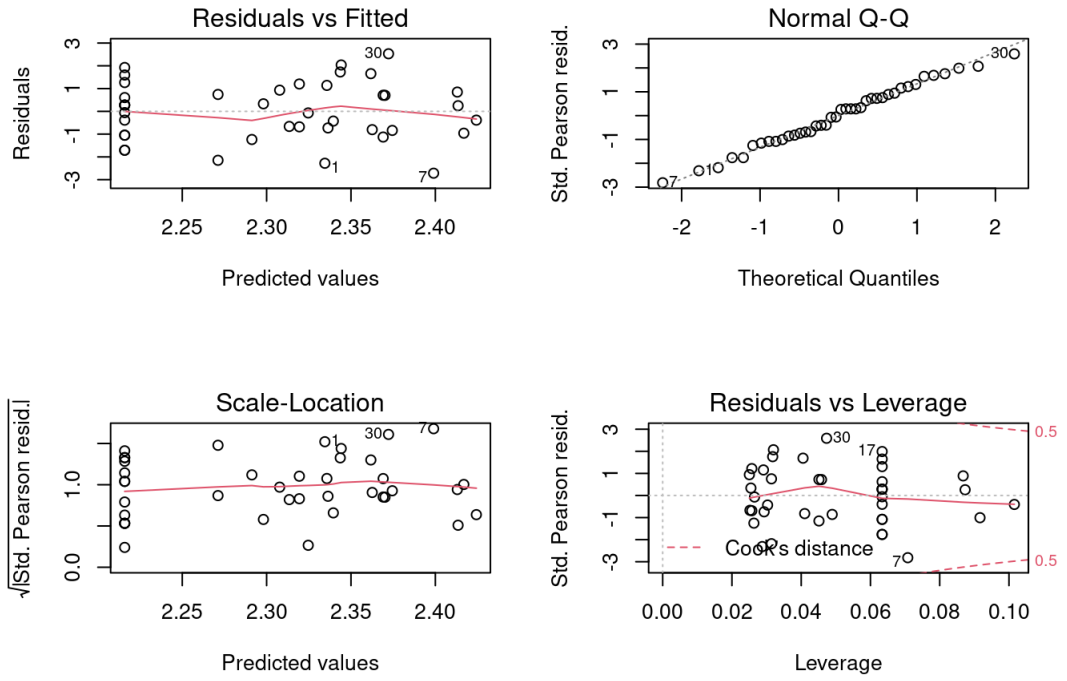


Figura 8.2: Diagnose básica dos resíduos do modelo GLM Poisson.

Vemos que as linhas vermelhas (que indicam a tendência dos dados) estão praticamente retas seguindo a linha pontilhada, sugerindo que existe homogeneidade de variância dos resíduos. Vemos também que nos quatro *plots* alguns dados, 1, 7 e 30 (referem-se às linhas do data frame) aparecem identificados, pois apresentam ligeiro desvio da normalidade e estão distantes da média. No entanto, não é algo para nos preocuparmos, pois não são valores muito extremos. Portanto, a diagnose básica indicou que o modelo com Poisson parece ser adequado para modelar estes dados, ao menos em termos de homogeneidade de variância.

Diagnose avançada dos resíduos do modelo

Alguns pacotes permitem calcular outros aspectos do modelo que facilitam a diagnose, ou seja, se podemos de fato confiar nos parâmetros estimados por eles, incluindo valores de significância. Um pressuposto importante dos modelos de contagem (incluindo Poisson) é a sobredispersão (*overdispersion*).

Vejamos como o pacote `DHARMa` funciona (Figura 8.3).

```
## Diagnose avançada
simulationOutput <- simulateResiduals(fittedModel = mod_pois,
                                      plot = TRUE)
```

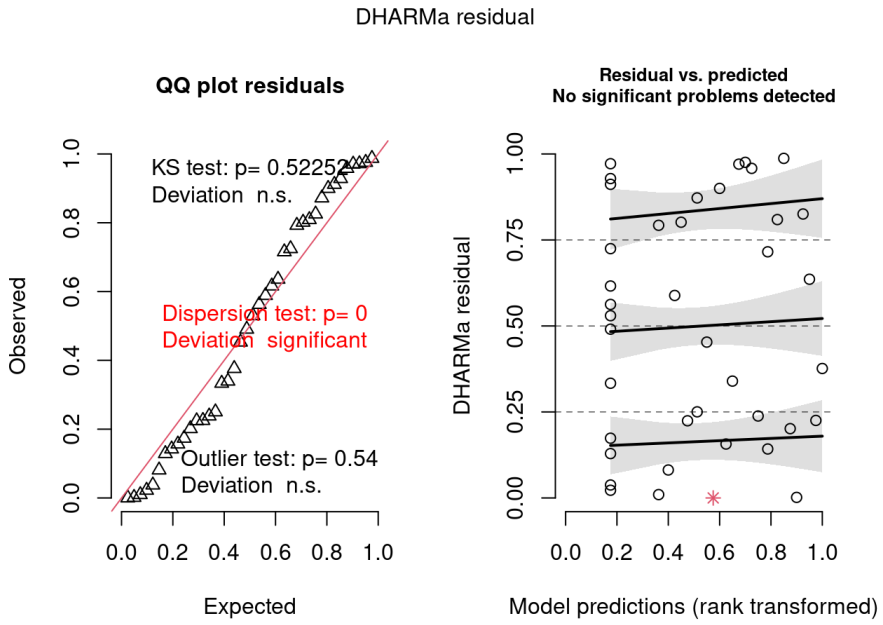


Figura 8.3: Diagnose avançada dos resíduos do modelo GLM Poisson.

O plot claramente indica que há problema com *overdispersion*, mas não em termos de desvios de normalidade (KS test) ou *outlier*, já que apenas o primeiro foi significativo (aparece em vermelho).

Detectando e lidando com *overdispersion*

O que é sobredispersão (*overdispersion*)? Ela ocorre quando a variância observada é muito maior do que aquela predita pelo modelo. Para modelos que utilizam a distribuição de Poisson, isso ocorre quando a variância aumenta com a média. Lembre-se de que esta distribuição tem apenas um único parâmetro para descrever tanto a média quanto a variância (λ). Portanto, a variância tem de ser igual à média. No entanto, se a variância nos dados observados for muito maior do que a média, dizemos que há sobredispersão nos dados.

Existem duas formas de diagnosticar *overdispersion* que estão implementadas na maioria dos pacotes. Aqui vamos demonstrá-las usando as funções `check_overdispersion()` e `testDispersion()` disponíveis nos pacotes `performance` e `DHARMa`, respectivamente.

A função `testDispersion()` do `DHARMa` utiliza um método de aleatorização dos resíduos para determinar se há *overdispersion* nos dados, cuja vantagem é que aborda diretamente a variação nos dados, ao invés de medir o ajuste do modelo em si, com outros testes (Figura 8.4).

```
## Overdispersion
par(mfrow = c(1, 1))
testDispersion(mod_pois) # modelo tem overdispersion
#>
#>  DHARMa nonparametric dispersion test via sd of residuals fitted vs.
simulated
#>
#> data:  simulationOutput
#> dispersion = 1.6489, p-value < 2.2e-16
#> alternative hypothesis: two.sided
```

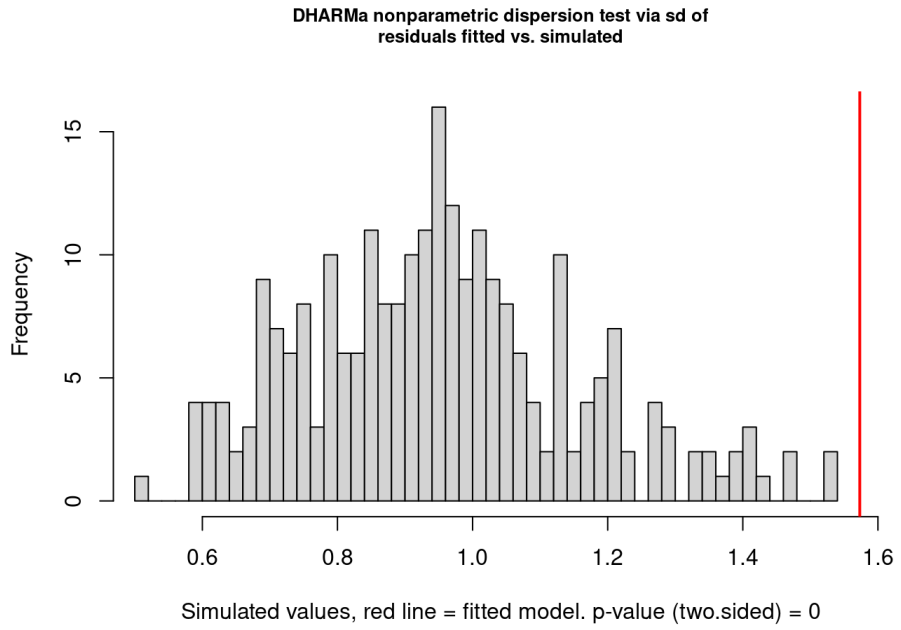


Figura 8.4: Teste de overdispersion do pacote `DHARMa`.

Aqui temos um gráfico e o resultado novamente do teste de *overdispersion* (que já aparecia no gráfico anterior) mostrando que de fato há *overdispersion*: perceba que o valor de *P* é significativo. O gráfico nos mostra em cinza a distribuição dos resíduos aleatorizados e a linha vermelha o valor observado da estatística. Já que a linha está bem à direita da distribuição, isso indica *overdispersion*, se estivesse à esquerda seria o caso de *underdispersion*.

Agora vamos utilizar a função `check_overdisperion()` que utiliza uma distribuição chi-quadrado e o valor de *dispersion ratio* para testar a presença de *overdispersion* no modelo. Esse teste também pode ser feito com a função acima ao se especificar o argumento `type="PearsonChisq"`.

```
## Testar a presença de overdispersion
check_overdispersion(mod_pois) # modelo tem overdispersion
#> # Overdispersion test
#>
#>        dispersion ratio =  1.657
#>   Pearson's Chi-Squared = 62.951
#>                 p-value =  0.007
```

Quando este resultado é significativo, como vimos na última linha acima, isso indica que há *overdispersion*.

Usando a função `summary()` podemos ter um resumo e descrição dos parâmetros do modelo.

```
## Resumo do modelo
summary(mod_pois)
#>
#> Call:
#> glm(formula = Riqueza_obs ~ dfrag, family = poisson(link = "log"),
#>     data = fragmentos)
#>
```

```
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -3.3467  -0.9110   0.0942   0.8336   2.2773
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   2.3051     0.0500  46.101   <2e-16 ***
#> dfrag         0.0718     0.0507   1.416    0.157
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for poisson family taken to be 1)
#>
#>     Null deviance: 70.868  on 39  degrees of freedom
#> Residual deviance: 68.856  on 38  degrees of freedom
#> AIC: 235.29
#>
#> Number of Fisher Scoring iterations: 4

## Dispersion parameter
deviance(mod_pois) / df.residual(mod_pois)
#> [1] 1.81199
```

Na parte de baixo do output da função `summary()` também podemos calcular o *dispersion parameter* dividindo o *residual deviance* pelos graus de liberdade dos resíduos. Esta é outra maneira fácil e rápida de detectar *overdispersion*. Neste exemplo, temos que *Dispersion parameter* = 1.8119903. Quando esse valor é próximo de 1, isso sugere que não há *overdispersion*. No entanto, se ele for maior que 1.5, isso sugere que o modelo sofre de *overdispersion* e que devemos usar outra distribuição, tal como a distribuição **binomial negativa**, por exemplo.

Além disso, uma outra forma de diagnosticar o modelo é calcular os resíduos de Pearson (resíduos normalizados), que é basicamente a raiz quadrada da variância da variável resposta.

Inflação de zeros

Qualquer das formas mostradas acima de diagnosticar *overdispersion* pode ser usada na maioria das vezes, com exceção de dados com muitos zeros (pouca variância). Por isso, devemos também testar se o nosso modelo sofre de inflação de zeros. Vejamos como isso funciona usando as funções `check_zeroinflation()` no pacote `performanace` e `testZeroInflation()` no pacote `DHARMa` (Figura 8.5).

```
## Inflação de zeros - performanace
check_zeroinflation(mod_pois) # para diagnosticar se o modelo sofre de zero
inflation
#> Model has no observed zeros in the response variable.
#> NULL

## Inflação de zeros - DHARMa
testZeroInflation(mod_pois) # para testar se existe zero inflation
```

```
#>
#>  DHARMa zero-inflation test via comparison to expected zeros with
simulation under H0 = fitted model
#>
#> data:  simulationOutput
#> ratioObsSim = NaN, p-value = 1
#> alternative hypothesis: two.sided
```

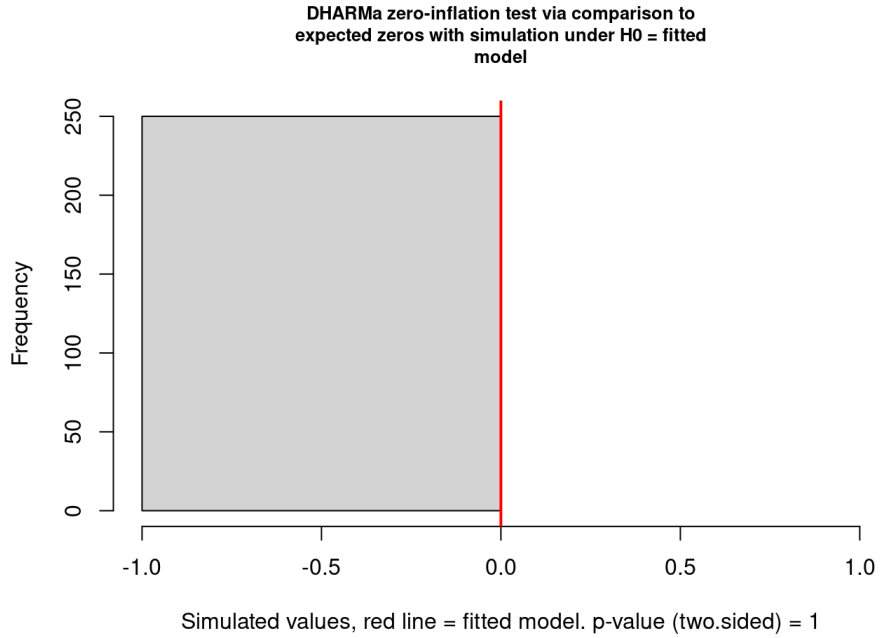


Figura 8.5: Teste de inflação de zeros do pacote `DHARMa`.

Tanto a função do `DHARMa` quanto do `performance` conseguiram detectar que o modelo tem problemas com *overdispersion*, mas isso não é causado pelo excesso de zeros. Como já dissemos acima, no caso da distribuição Poisson, tanto a média quanto a variância são modeladas pelo mesmo parâmetro (λ). Isso faz com que esta distribuição não seja muito útil para modelar dados de contagem em que haja muita variância em torno da média. Esse infelizmente é o caso da grande maioria dos dados ecológicos.

Por estes motivos, não podemos fazer inferência com este modelo porque os parâmetros estimados não são confiáveis. Mas vejamos como seria feita essa inferência *caso este modelo fosse adequado*.

Inferência

Aqui iremos apresentar várias funções para calcular o coeficiente de determinação (R^2). No caso de GLM(M)s, não há um consenso sobre como se calcula este coeficiente, havendo várias propostas que utilizam maneiras diferentes de estimar a homogeneidade de variância e covariância entre observações dos resíduos, veja Nakagawa et al. (2017) e Ives (2015) para maiores detalhes, assim como o *help* das respectivas funções.

```
## Coeficientes estimados pelo modelo
summary(mod_pois)
#>
```

```
#> Call:
#> glm(formula = Riqueza_obs ~ dfrag, family = poisson(link = "log"),
#>     data = fragmentos)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -3.3467  -0.9110   0.0942   0.8336   2.2773
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)   2.3051     0.0500  46.101   <2e-16 ***
#> dfrag         0.0718     0.0507   1.416    0.157
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for poisson family taken to be 1)
#>
#>     Null deviance: 70.868  on 39  degrees of freedom
#> Residual deviance: 68.856  on 38  degrees of freedom
#> AIC: 235.29
#>
#> Number of Fisher Scoring iterations: 4

## Calculando o R2 do modelo
r.squaredGLMM(mod_pois)
#>                  R2m        R2c
#> delta     0.04925800 0.04925800
#> lognormal 0.05154558 0.05154558
#> trigamma  0.04696308 0.04696308

rsquared(mod_pois)
#>      Response  family link     method  R.squared
#> 1 Riqueza_obs poisson  log nagelkerke 0.04919844

r2(mod_pois)
#> # R2 for Generalized Linear Regression
#>   Nagelkerke's R2: 0.059
```

Podemos ver que os valores de R^2 são bem baixos (em torno de 4 - 5%), independentemente do método que usamos pra calculá-lo.

Plot do modelo predito

Vamos realizar a visualização do ajuste do modelo Poisson (Figura 8.6).

```
a1 <- ggplot(fragmentos, aes(dfrag, Riqueza_obs)) +
    geom_point(cex = 4,alpha = 0.7) +
    geom_smooth(method = "glm", formula = y~x,
                method.args = list(family = "poisson"), se = TRUE) +
```

```
    labs(x = "Distância para o fragmento mais próximo",
         y = "Riqueza observada") +
    tema_livro()

ggMarginal(a1, fill = "red")
```

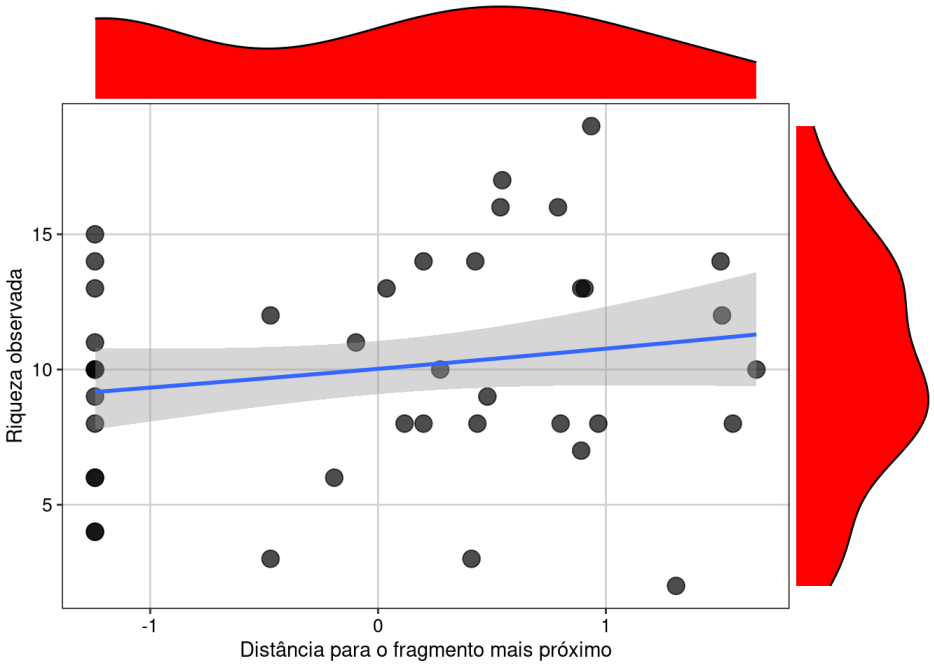


Figura 8.6: Gráfico do GLM Poisson.

Interpretação dos resultados

Aqui vemos que há uma leve tendência na relação positiva entre distância para o fragmento mais próximo e a riqueza de anfíbios observada. No entanto, há uma grande dispersão nos dados ao redor da reta do modelo, fazendo com que a relação não seja de fato significativa e tenhamos um R^2 bem baixo. Caso pudéssemos confiar nos parâmetros deste modelo, poderíamos dizer que existe uma leve tendência a um aumento da riqueza observada de anfíbios anuros à medida que aumenta a distância da poça para o fragmento mais próximo.

8.4.1 O que causa a *overdispersion*?

Existem dois conjuntos de causas: aparente ou real.

As causas aparentes são geradas pela má especificação do modelo, tais como:

1. não inclusão de covariáveis ou interações no modelo
2. presença de *outliers* na variável resposta
3. efeitos não lineares da covariável (X2, X3...)
4. escolha errada da função de ligação (*link function*)

As causas reais incluem:

1. variância maior que a média
2. muitos zeros
3. agregação de observações
4. correlação entre observações (não independência)

8.4.2 O que fazer se seu modelo tiver *overdispersion*?

Depois de tentar corrigir possíveis más especificações, como as listadas acima, existem duas alternativas:

1. usar outra distribuição, tal como binomial negativa, caso o *dispersion parameter* seja maior que 15 ou 20
2. Usar um modelo com correção de erro da sobredispersão, caso 1.5 < *dispersion* > 15

Geralmente, dados de contagem em estudos ecológicos não seguem uma distribuição Poisson, pois há muita dispersão (variância) nos dados. Logo, o pressuposto da distribuição Poisson, i.e., de que a média e variância são descritas por um mesmo parâmetro (λ) é quebrado.

Como vimos, *overdispersion* é um problema comum ao analisar dados ecológicos e deve necessariamente ser diagnosticado no modelo. Uma maneira de lidar com esse tipo de problema é utilizar uma outra distribuição diferente da Poisson. A binomial negativa pode ser entendida como uma mistura das distribuições Poisson e Gamma, ou seja, ela aceita dados de contagem que sejam positivos, mas sem zero. A grande vantagem desta distribuição é que, diferentemente da Poisson, ela tem um parâmetro para modelar a média (λ) e outro para modelar a variância (k). Logo, ela permite modelar dados em que a média é diferente da variância. Vejamos um exemplo.

Aqui vamos continuar com estes dados para ver como o modelo se comporta com essa nova distribuição especificada. Para isso vamos utilizar a função `glm.nb` do pacote `MASS`.

Modelagem

```
## Ajuste do modelo
mod_nb <- glm.nb(Riqueza_obs ~ dfrag, data = fragmentos)
```

Diagnose resíduos

Assim como fizemos com o modelo com Poisson, vamos agora diagnosticar os resíduos (Figura 8.7).

```
## Diagnose
par(mfrow = c(2, 2))
plot(mod_nb)
par(mfrow = c(1, 1))
(chat <- deviance(mod_nb) / df.residual(mod_nb)) # DISPERSION PARAMETER
#> [1] 1.126184
```

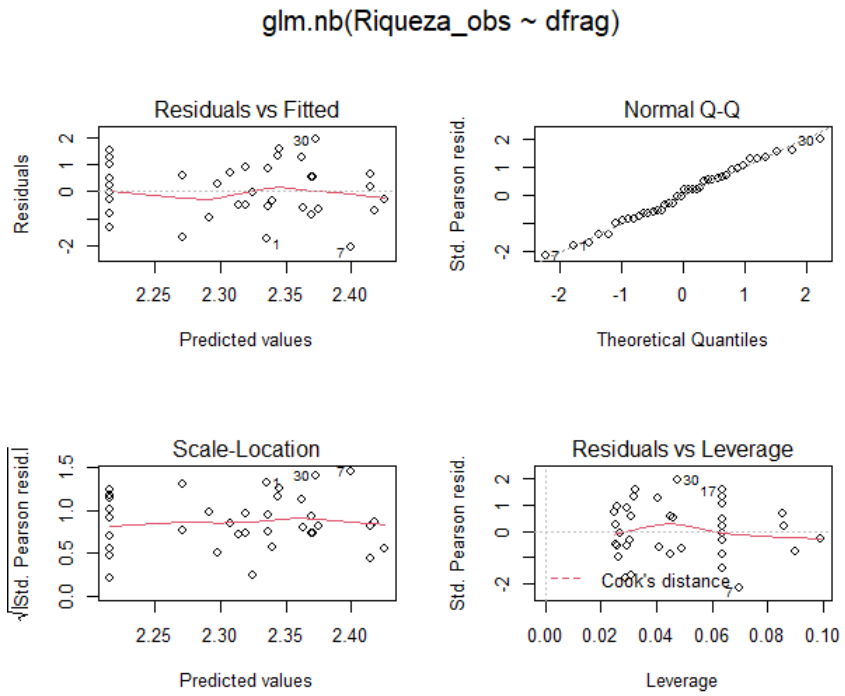


Figura 8.7: Diagnose básica dos resíduos do modelo GLM binomial Negativo.

Compare estes gráficos (Figura 8.7) com os do modelo anterior com distribuição Poisson (Figura 8.2). Eles são praticamente idênticos, ou seja, o modelo com Poisson já tinha homogeneidade de variância e não tinha problemas com normalidade dos resíduos. Agora vejamos se o problema com *overdispersion* foi resolvido (Figura 8.8).

```
## Diagnose avançada
simulationOutput <- simulateResiduals(fittedModel = mod_nb, plot = TRUE)
```

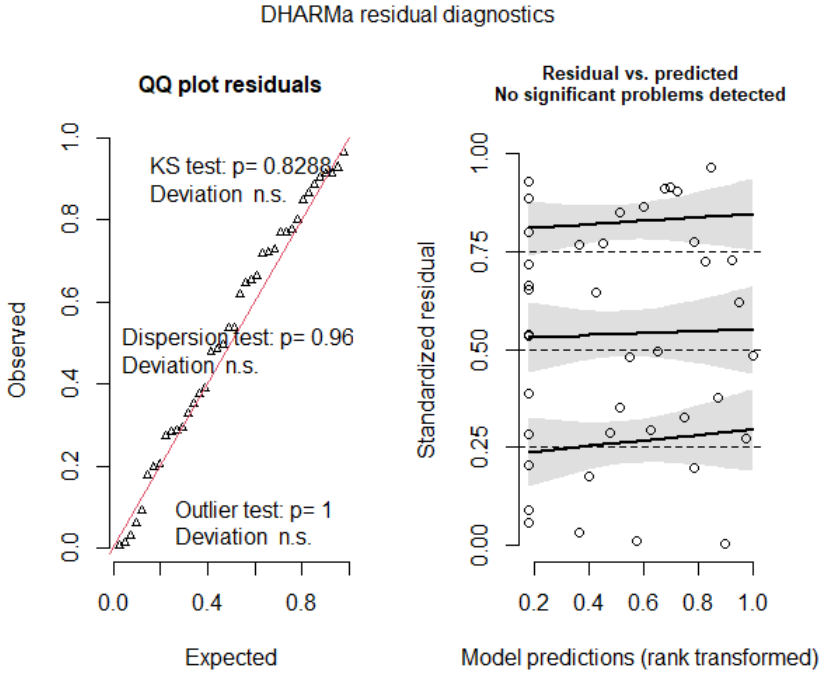


Figura 8.8: Diagnose avançada dos resíduos do modelo GLM binomial Negativo.

Na diagnose do modelo pelo `DHARMa`, vemos que bastou mudar a distribuição de probabilidade que o problema de *overdispersion* foi resolvido (nenhum teste foi significativo no quadro da esquerda). Como já sabíamos, não há problemas com homogeneidade de variância (plot da direita mostrando a tendência entre o predito e resíduos pra cada *quantil*), nem de *outliers*. O *dispersion parameter* é mais próximo de 1 do que no modelo com Poisson. Agora sim podemos levar em conta o R^2.

Inferência

```
## Coeficiente de determinação
rsquared(mod_nb)
#>      Response                    family link     method  R.squared
#> 1 Riqueza_obs Negative Binomial(14.7068)  log nagelkerke 0.02935674
```

Mas esse valor parece ser um pouco menor do que anteriormente. Perceba que aqui utilizamos somente uma das funções apresentadas anteriormente, já que se trata de um modelo GLM com binomial negativa, calculamos o R^2 pelo método de Nagelkerke.

Usando a função `summary()` podemos ter um resumo e descrição dos parâmetros do modelo.

```
## Coeficientes estimados pelo modelo
summary(mod_nb)
#>
#> Call:
#> glm.nb(formula = Riqueza_obs ~ dfrag, data = fragmentos, init.theta =
14.70679964,
#>     link = log)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -2.7569  -0.7068   0.0694   0.6194   1.6546
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  2.30504    0.06481  35.567   <2e-16 ***
#> dfrag        0.07248    0.06571   1.103     0.27
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for Negative Binomial(14.7068) family taken to be 1)
#>
#>     Null deviance: 44.002  on 39  degrees of freedom
#> Residual deviance: 42.795  on 38  degrees of freedom
#> AIC: 231.68
#>
#> Number of Fisher Scoring iterations: 1
#>
#>
#>               Theta:  14.71
#>           Std. Err.:  8.62
```

```
#>
#>  2 x log-likelihood:  -225.68
```

Interpretação dos resultados

Aqui vemos que a interpretação do resultado em termos de valor de *P* não mudou, ou seja, a distância para o fragmento mais próximo não foi significativa. Mas vejam que o coeficiente (*slope*) mudou um pouco, antes era `0.0718` (`SE=0.0507`) e com binomial negativa passa a ser `0.07248` (`SE=0.06571`).

Plot do modelo predito

Vamos realizar a visualização do ajuste do modelo binomial Negativo (Figura 8.9).

```
## Gráfico
ggplot(fragmentos, aes(dfrag, Riqueza_obs)) +
    geom_point(size = 4, alpha = 0.7) +
    geom_smooth(method = "glm.nb", formula = y~x, se = TRUE) +
    labs(x = "Distância para o fragmento mais próximo",
         y = "Riqueza observada") +
    tema_livro()
```

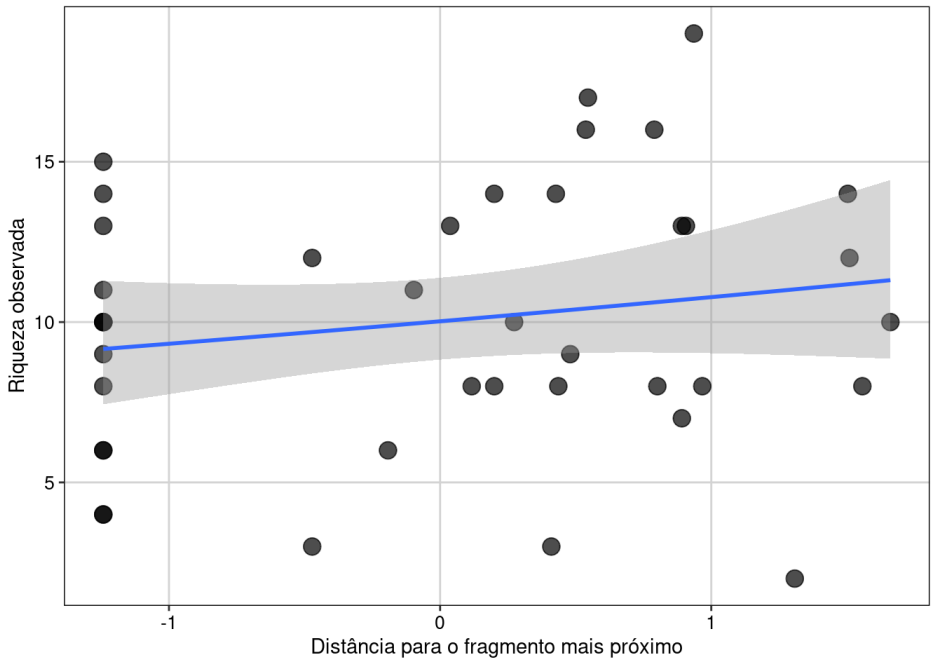


Figura 8.9: Gráfico do GLM binomial Negativo.

Aqui vemos que a reta predita pelo modelo é muito similar ao que tivemos com o modelo Poisson. No entanto, agora que sabemos que este modelo com binomial negativa foi corretamente especificado e podemos confiar nos parâmetros estimados.

8.5 Dados de contagem: modelos quasi-likelihood

Como dissemos acima, uma outra alternativa para ajustar modelos GLM a dados de contagem são os chamados "quasi-likelihood," tais como quasi-Poisson e quasi-binomial. Dependendo do valor do *dispersion parameter*, pode ser útil escolher este tipo de modelo. No entanto, eles vêm com uma desvantagem: não é possível calcular o valor de *Akaike Information Criterion* (AIC) porque estes modelos não retornam um valor de *likelihood* (verosimilhança). Este parâmetro é comumente utilizado em abordagens estatísticas de teoria da informação para selecionar o melhor modelo que se ajusta aos dados. Neste caso, precisamos utilizar outras funções disponíveis nos pacotes `MuMIn`, `AICcmodavg` e `bbmle` para calcular o QAIC. Para mais detalhes sobre esses modelos, veja o vignette sobre o assunto do pacote `bbmle`.

Análise

Aqui vamos apenas exemplificar como um modelo com distribuição quasi-poisson pode ser especificado.

```
## Modelo
mod_quasipois <- glm(Riqueza_obs ~ dfrag,
                     family = quasipoisson(link = "log"),
                     data = fragmentos)
```

Diagnose dos resíduos

A função `resid` não leva em conta a sobredispersão e temos de calcular manualmente o parâmetro de dispersão e inclui-lo no plot. Portanto, não podemos realizar a diagnose de modelos quasi-Poisson apenas com a função `plot` como fazíamos até agora. Então, calculamos primeiramente os resíduos de Pearson e depois dividindo-o pela raiz quadrada do parâmetro de dispersão (Figura 8.10).

```
## Diagnose dos resíduos
EP <- resid(mod_quasipois, type = "pearson")
ED <- resid(mod_quasipois, type = "deviance")
mu <- predict(mod_quasipois, type = "response")
E <- fragmentos$Riqueza_obs - mu
EP2 <- E / sqrt(1.65662 * mu) # dispersion parameter da quasipoisson
op <- par(mfrow = c(2, 2))
plot(x = mu, y = E, main = "Response residuals")
plot(x = mu, y = EP, main = "Pearson residuals")
plot(x = mu, y = EP2, main = "Pearson residuals scaled")
plot(x = mu, y = ED, main = "Deviance residuals")
par(op)
par(mfrow = c(1, 1))
```

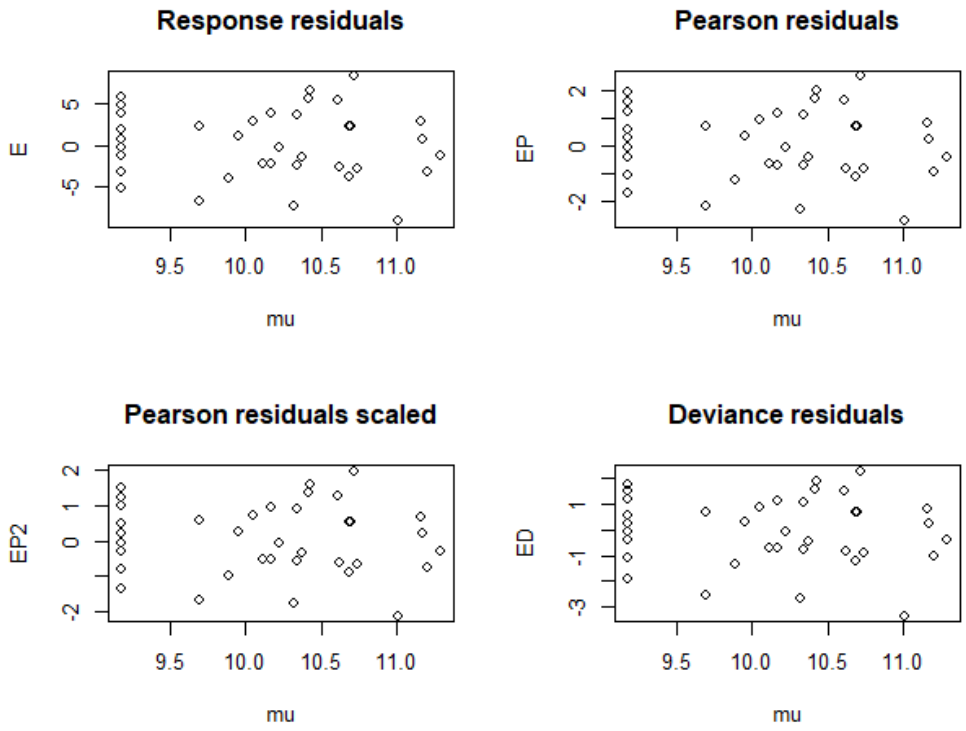


Figura 8.10: Diagnose dos resíduos do modelo GLM Quasi-Poisson.

Aqui vemos que não existe um padrão claro nos resíduos, muito similar ao que tínhamos anteriormente. Devido às limitações de distribuições "quasi" e dado que já temos um modelo adequado com binomial negativa, sugerimos interpretar apenas o modelo anterior com binomial negativa.

8.6 Dados de contagem: a distribuição binomial

Quando temos dados de proporção (e.g., número de doentes por 1000 habitantes) ou incidência (i.e., presença ou ausência), a distribuição mais adequada para modelar os dados é a distribuição binomial. No entanto, temos que especificar o modelo de acordo com o tipo dos dados no argumento `formula`. Vejamos dois exemplos.

8.6.1 Análise com dados de proporção

Neste exemplo, vamos ver como podemos modelar a proporção de células sanguíneas em função do tipo de tratamento.

Explicação dos dados

Este conjunto de dados foi coletado por (Franco-Belussi et al. 2018). Os autores utilizaram um desenho experimental típico de uma 2x5 ANOVA fatorial (ou *two-way* ANOVA) em que temos dois tratamentos (fatores): pigmentação do girino com dois níveis (Yes e No) e Tempo de exposição com cinco níveis (controle sem UV, 6 h, 12 h, 18 h e 24 h de exposição à UV).

Pergunta

A melanina protege girinos contra os efeitos da radiação ultravioleta?

Predições

Como a melanina participa do sistema imune inato, ela desempenharia um papel na resposta do organismo à radiação UV, auxiliando as células imunes a combater os seus efeitos deletérios.

Variáveis

- Variável resposta: Contagem diferencial de eosinófilos
- data frame com 10 girinos em cada tratamento, totalizando 50 girinos

```
## Dados
head(uv_cells)
#>     UV Pigmentation Total_Cell Lymphocyte Neutrophil Basophil Monocyte
Eosinophil
#> 1 1.CT          Yes        100         80         18        0        0
2
#> 2 1.CT          Yes        100         74         17        6        0
3
#> 3 1.CT          Yes        100         78         22        0        0
0
#> 4 1.CT          Yes        100         87         13        0        0
0
#> 5 1.CT          Yes        100         74         21        1        0
4
#> 6 2.6h          Yes        100         95          4        0        0
1

## Traduzir nomes das colunas e níveis de pigmentação
colnames(uv_cells) <- c("UV", "Pigmentacao", "n_celulas", "linfocito",
                        "neutrofilo", "basofilo", "monocito",
                        "eosinofilo")
uv_cells$Pigmentacao[uv_cells$Pigmentacao=="Yes"] <- "sim"
uv_cells$Pigmentacao[uv_cells$Pigmentacao=="No"] <- "nao"
```

Vamos explorar os dados para tentar entender como são as relações (Figura 8.11).

```
## Gráfico

# Calcular média e intervalo de confiança
eosinofilo <- summarySE(uv_cells,
                        measurevar = "eosinofilo",
                        groupvars = c("UV", "Pigmentacao"))

# Definir posição de linhas e pontos no gráfico
pd <- position_dodge(0.1)

eosinofilo %>%
  ggplot(aes(x = UV, y = eosinofilo, colour = Pigmentacao,
             group = Pigmentacao, fill = Pigmentacao)) +
  geom_errorbar(aes(ymin=eosinofilo-se, ymax=eosinofilo +se),
```

```
              width=.1, size = 1.1, position=pd) +
  geom_line(position=pd, size = 1.1) +
  geom_point(pch = 21, colour = "black", position=pd, size=3.5) +
  scale_colour_manual(values = c("darkorange", "cyan4")) +
  scale_fill_manual(values = c("darkorange", "cyan4")) +
  labs(x = "UV", y = "Eosinófilo", fill="Pigmentação",
       colour="Pigmentação")+
  tema_livro()
```

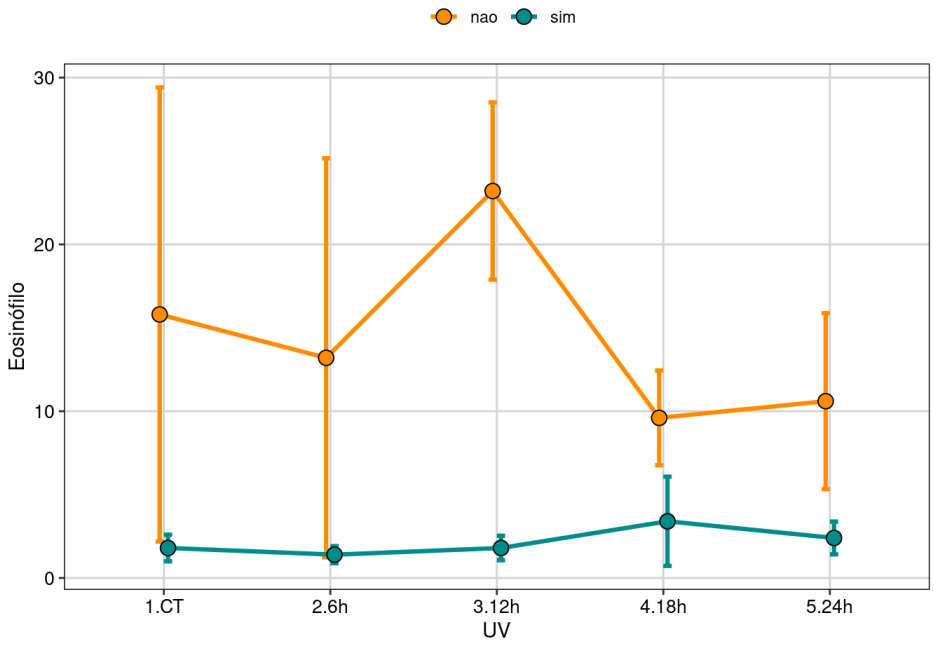


Figura 8.11: Gráfico para explorar as relações entre as variáveis.

Aqui vemos que a quantidade de eosinófilos é muito maior nos girinos sem pigmentação ("albinos"). Já que estes animais não têm pigmentação melânica, as células brancas do sangue são a única ferramenta de combate aos efeitos deletérios da UV.

Modelagem

Aqui vamos usar o `cbind` no argumento `formula` para dizer que queremos modelar a contagem de eosinófilos em relação ao número total de células, ou seja, sua proporção. Aqui temos a contagem do número de eosinófilos (um tipo de célula da série branca do sangue) em lâminas histológicas de girinos da rã-touro (*Lithobates catesbeianus*) num total de 1000 células.

```
## Modelo
mod1 <- glm(cbind(eosinofilo, n_celulas) ~ UV * Pigmentacao,
            family = binomial, data = uv_cells)
```

Diagnose básica dos resíduos do modelo

Vamos fazer a diagnose básica dos resíduos do modelo (Figura 8.12).

```
## Diagnose dos resíduos
par(mfrow = c(2, 2))
```

```
plot(mod1)
par(mfrow = c(1, 1))
```

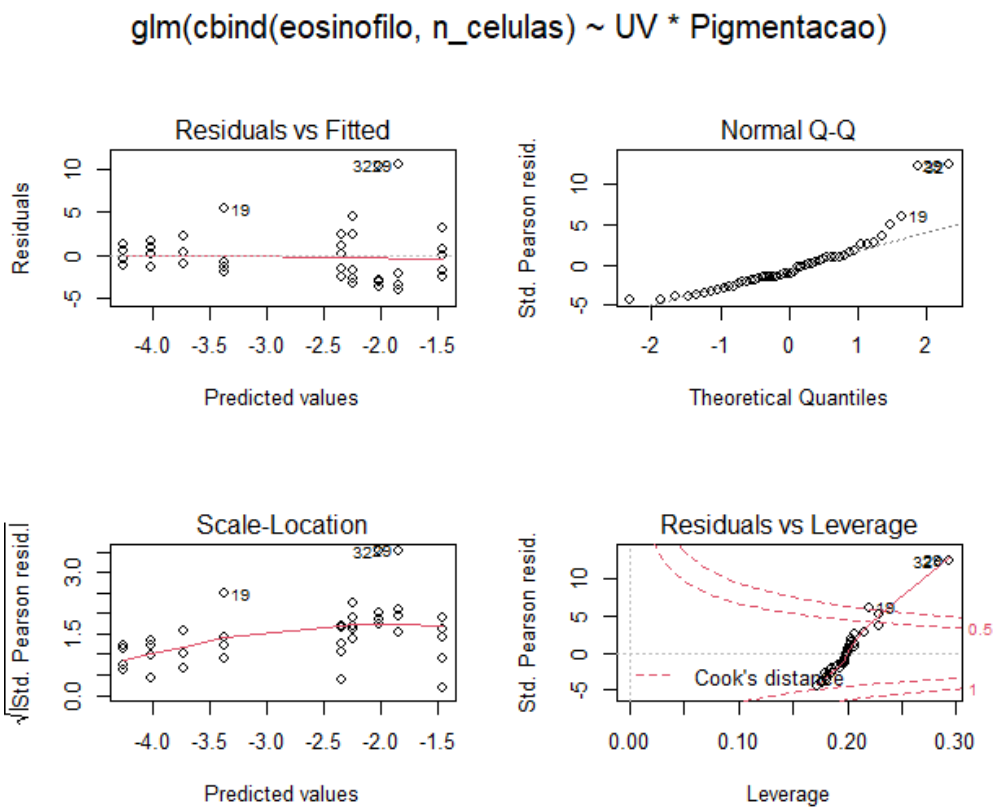


Figura 8.12: Diagnose básica dos resíduos do modelo GLM binomial.

Parece que os resíduos não sofrem de heterogeneidade de variância (linha vermelha está reta), mas parece haver um pequeno desvio da normalidade (veja pontos 19, 29 e 32 destacados no plot de *quantis* e no de *outliers*). Vejamos o que o `DHARMa` nos diz (Figura 8.13).

```
## Diagnose avançada
simulationBion <- simulateResiduals(fittedModel = mod1, plot = TRUE)
```

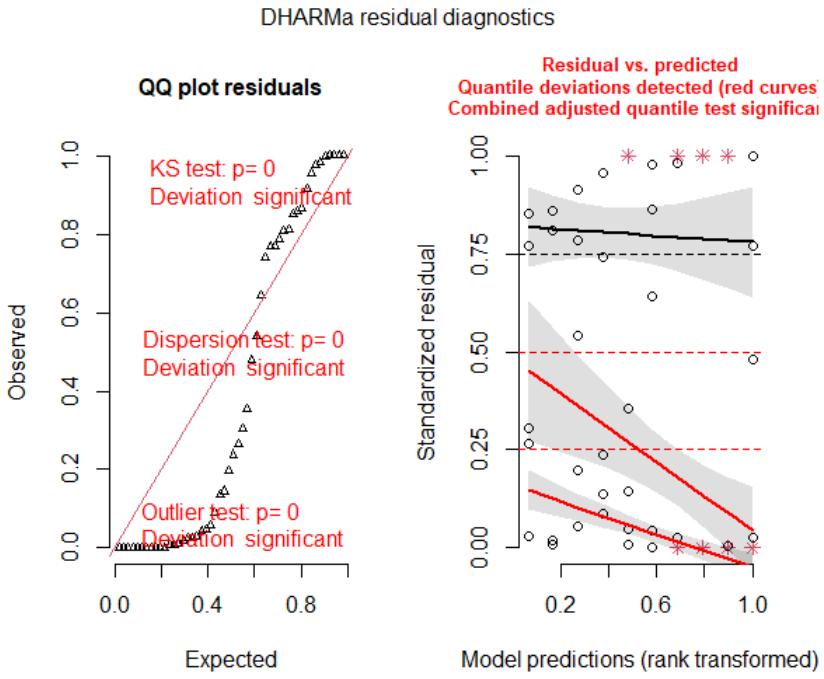


Figura 8.13: Diagnose avançada dos resíduos do modelo GLM binomial.

Agora vamos checar a qualidade do modelo com base na distribuição dos resíduos (Figura 8.14)

```
## Diagnose avançada
binned_residuals(mod1)
#> Warning: Probably bad model fit. Only about 29% of the residuals are
inside the error bounds.
```

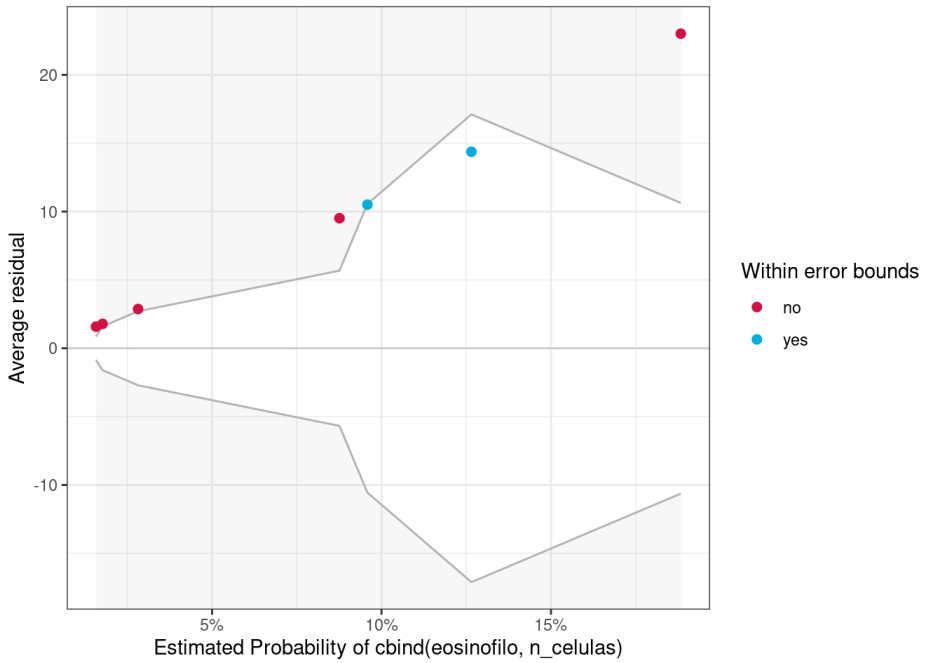


Figura 8.14: Diagnose avançada dos resíduos do modelo GLM binomial.

Aqui já não resta dúvidas de que os resíduos deste modelo sofrem tanto com heterogeneidade de variância, quanto *overdispersion* e problemas com *outliers*. Provavelmente o problema com *outliers* ocorreu por conta do pequeno tamanho amostral.

Inferência

Sabemos que o modelo não parece ser adequado para os dados, mas vamos interpretá-lo mesmo assim para que possamos entender a saída da função `summary()` e os contrastes entre os níveis dos fatores.

```
## Coeficientes estimados pelo modelo
summary(mod1)
#>
#> Call:
#> glm(formula = cbind(eosinofilo, n_celulas) ~ UV * Pigmentacao,
#>     family = binomial, data = uv_cells)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -5.4165  -2.5266  -1.0148   0.8068   8.8233
#>
#> Coefficients:
#>                       Estimate Std. Error z value Pr(>|z|)
#> (Intercept)           -1.84516    0.12107 -15.241  < 2e-16 ***
#> UV2.6h                -0.17979    0.17835  -1.008   0.3134
#> UV3.12h                0.38414    0.15899   2.416   0.0157 *
#> UV4.18h               -0.49825    0.19363  -2.573   0.0101 *
#> UV5.24h               -0.39916    0.18848  -2.118   0.0342 *
#> Pigmentacaosim        -2.17222    0.35745  -6.077 1.22e-09 ***
#> UV2.6h:Pigmentacaosim -0.07152    0.53831  -0.133   0.8943
#> UV3.12h:Pigmentacaosim -0.38414    0.50150  -0.766   0.4437
#> UV4.18h:Pigmentacaosim  1.13424    0.45981   2.467   0.0136 *
#> UV5.24h:Pigmentacaosim  0.68684    0.48370   1.420   0.1556
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 737.36  on 49  degrees of freedom
#> Residual deviance: 460.85  on 40  degrees of freedom
#> AIC: 610.35
#>
#> Number of Fisher Scoring iterations: 5

anova(mod1)
#> Analysis of Deviance Table
#>
#> Model: binomial, link: logit
#>
```

```
#> Response: cbind(eosinofilo, n_celulas)
#>
#> Terms added sequentially (first to last)
#>
#>
#>                Df Deviance Resid. Df Resid. Dev
#> NULL                           49     737.36
#> UV              4   26.034        45     711.32
#> Pigmentacao     1  235.682        44     475.64
#> UV:Pigmentacao  4   14.789        40     460.85
```

Aqui temos tanto a tabela com os resultados por níveis dos fatores (`summary()`), quanto a tabela com a *Deviance* que mostra os fatores e suas interações (`anova()`). Vemos que nenhum fator foi significativo. Caso houvesse algum fator significativo, poderíamos testar a significância de cada nível dos fatores usando contrastes, desta forma.

```
## Parâmetros
pairs(emmeans(mod1, ~ UV|Pigmentacao))
#> Pigmentacao = nao:
#>  contrast      estimate    SE  df z.ratio p.value
#>  1.CT - 2.6h     0.1798 0.178 Inf   1.008  0.8518
#>  1.CT - 3.12h   -0.3841 0.159 Inf  -2.416  0.1109
#>  1.CT - 4.18h    0.4982 0.194 Inf   2.573  0.0753
#>  1.CT - 5.24h    0.3992 0.188 Inf   2.118  0.2124
#>  2.6h - 3.12h   -0.5639 0.167 Inf  -3.384  0.0064
#>  2.6h - 4.18h    0.3185 0.200 Inf   1.593  0.5021
#>  2.6h - 5.24h    0.2194 0.195 Inf   1.125  0.7933
#>  3.12h - 4.18h   0.8824 0.183 Inf   4.824  <.0001
#>  3.12h - 5.24h   0.7833 0.177 Inf   4.414  0.0001
#>  4.18h - 5.24h  -0.0991 0.209 Inf  -0.474  0.9897
#>
#> Pigmentacao = sim:
#>  contrast      estimate    SE  df z.ratio p.value
#>  1.CT - 2.6h     0.2513 0.508 Inf   0.495  0.9879
#>  1.CT - 3.12h    0.0000 0.476 Inf   0.000  1.0000
#>  1.CT - 4.18h   -0.6360 0.417 Inf  -1.525  0.5461
#>  1.CT - 5.24h   -0.2877 0.445 Inf  -0.646  0.9675
#>  2.6h - 3.12h   -0.2513 0.508 Inf  -0.495  0.9879
#>  2.6h - 4.18h   -0.8873 0.454 Inf  -1.957  0.2876
#>  2.6h - 5.24h   -0.5390 0.480 Inf  -1.123  0.7942
#>  3.12h - 4.18h  -0.6360 0.417 Inf  -1.525  0.5461
#>  3.12h - 5.24h  -0.2877 0.445 Inf  -0.646  0.9675
#>  4.18h - 5.24h   0.3483 0.382 Inf   0.911  0.8928
#>
#> Results are given on the log odds ratio (not the response) scale.
#> P value adjustment: tukey method for comparing a family of 5 estimates
```

Aqui temos o valor de cada combinação de níveis dos fatores, com seu respectivo valor de contraste e o valor de *P*. Vemos que para girinos sem pigmentação, apenas três contrastes foram significativos.

Plot do modelo predito

Vamos realizar a visualização do ajuste do modelo binomial (Figura 8.15).

```
ggplot(uv_cells, aes(UV, eosinofilo)) +
    geom_violin(aes(color = Pigmentacao)) +
    geom_jitter(shape = 16, position = position_jitter(0.1),
                cex = 4, alpha = 0.7) +
    scale_colour_manual(values = c("darkorange", "cyan4")) +
    tema_livro()
```

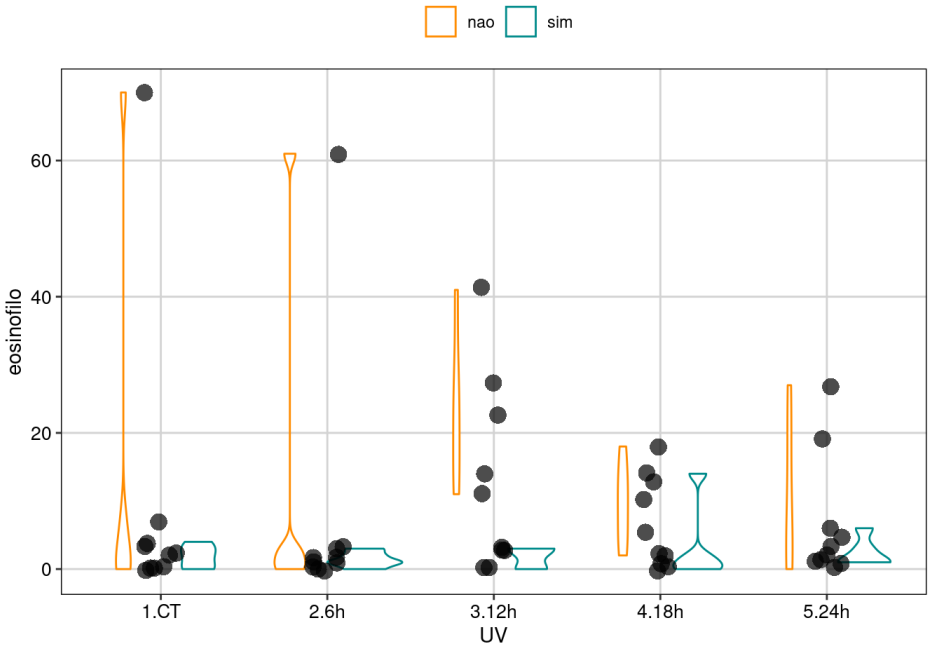


Figura 8.15: Gráfico do GLM binomial.

Usando o `geom_violin()`, podemos perceber que existe uma dispersão maior nos tratamentos que utilizaram girinos sem pigmentação do que nos tratamentos com girinos pigmentados.

8.7 Análise com dados de incidência

Uma outra aplicação da distribuição binomial é quando temos dados de incidência, ou seja, presença ou ausência, de alguma variável. Por exemplo, presença ou ausência de uma espécie ou indivíduo num local. Neste caso a `formula` é diferente e o modelo é similar a uma regressão logística, vejamos.

Aqui vamos utilizar os dados sobre autotomia da cauda de lagartos da espécie *Coleodactylus meridionalis* observados em fragmentos florestais da Mata Atlântica no estado de Pernambuco (Oliveira et al. 2020).

Pergunta

A probabilidade de lagartos da espécie *Coleodactylus meridionalis* perderem (autotomizarem) a cauda aumenta com o tamanho do corpo e de acordo com o sexo dos lagartos?

Predições

Quanto maior o lagarto, maior a probabilidade de autotomia da cauda e que esta resposta poderia também diferir entre sexos devido ao dimorfismo sexual.

Variáveis

- Variável resposta: Presença ou ausência de cauda autotomizada em lagartos encontrados por busca ativa

Exploração dos dados

Este conjunto de dados possui muitas entradas faltantes (codificadas como `NA`). Primeiro vamos visualizar o conjunto de dados e depois precisamos remover as linhas que contêm dados faltantes. Aqui podemos usar a função interna do `ggplot2::remove_missing()` para remover linhas cujas variáveis informadas no argumento estejam faltando, vejamos (Figuras 8.16 e 8.17).

```
## Traduzir nomes das colunas e níveis de pigmentação
head(lagartos)
#>   Numero    Sex   SVL Intact_tail_length Autotomized_tail_length Tail_state
#> 1      2   Male 20.70                 NA                   12.88          0
#> 2      3   Male 21.10                 NA                   13.07          0
#> 3      6 Female 23.72                 NA                   17.56          0
#> 4      9   Male 18.84              17.38                      NA          1
#> 5     21   Male 22.20                 NA                   16.50          0
#> 6     22   <NA> 20.59                 NA                   12.46          0
colnames(lagartos) <- c("numero", "sexo", "SVL", "comprimento_cauda",
                        "cauda_autotomizada", "estado_cauda")

## Dados faltantes
vis_miss(lagartos, cluster = TRUE)
```

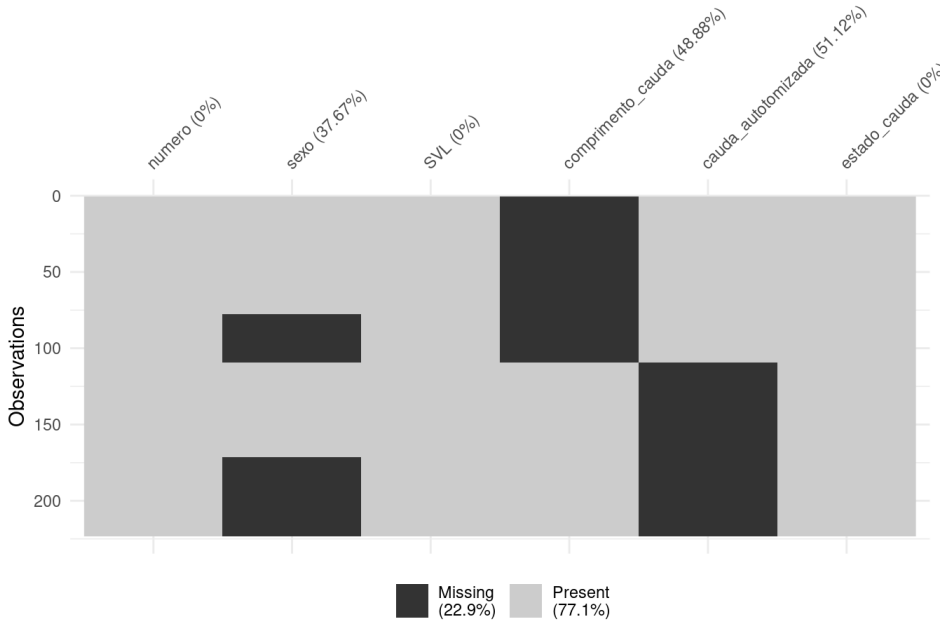


Figura 8.16: Gráfico para explorar os dados faltantes.

Reparem que estão faltando 22.9% dos dados. Vamos excluir as linhas com os dados faltantes para a variável `sex` (Figura 8.17).

```
## Removendo dados faltantes
dados_semNA <- remove_missing(lagartos, vars = "sexo")

## Visualizar
vis_miss(dados_semNA)
```

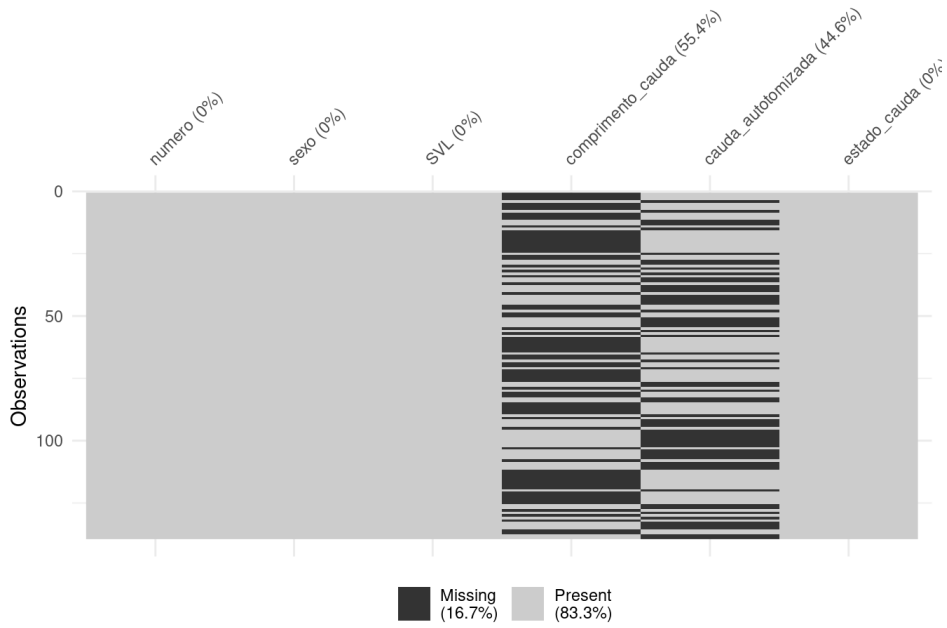


Figura 8.17: Gráficos para explorar os dados faltantes.

Agora, seguindo o que já estamos acostumados a fazer, vamos visualizar os dados com a nossa hipótese (Figura 8.18).

```
## Gráfico
ggplot(dados_semNA, aes(SVL, estado_cauda)) +
    geom_point(aes(shape = sexo, colour = sexo), size = 4, alpha = 0.4) +
    geom_smooth(method = "glm",
                method.args = list(family = "binomial")) +
    labs(y = "Estado da Cauda", x = "Comprimento Rostro-Cloacal (mm)",
         shape = "Sexo", colour = "Sexo") +
    tema_livro()
```

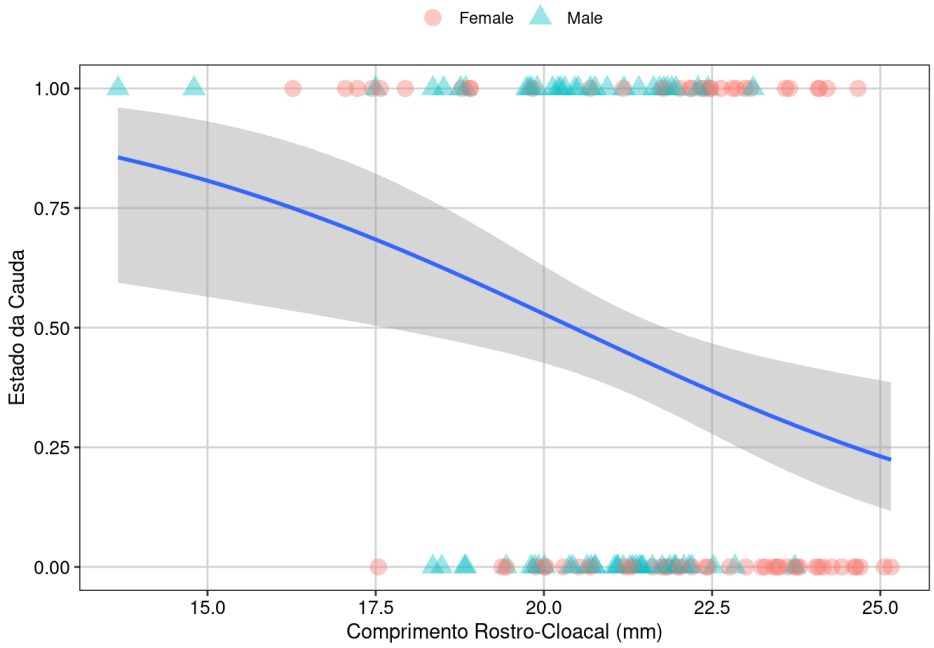


Figura 8.18: Gráfico para explorar as relações entre as variáveis.

Modelagem

Aqui vamos construir dois modelos com a mesma distribuição binomial, mas com dois *links functions*: logit e probit. A função `logit` possui caudas um pouco mais achatadas, isto é, a curva `probit` se aproxima dos eixos mais rapidamente que a logit. Geralmente não há muita diferença entre elas. Como não temos nenhuma expectativa de qual *link function* é o melhor, podemos fazer uma seleção de modelos.

```
## Modelos
mod_log <- glm(estado_cauda ~ SVL * sexo, data = dados_semNA,
               family = binomial(link = "logit"))
mod_pro <- glm(estado_cauda ~ SVL * sexo, data = dados_semNA,
               family = binomial(link = "probit"))

# Seleção de modelos
AICctab(mod_log, mod_pro, nobs = 139)
#>         dAICc df
#> mod_pro 0.0   4
#> mod_log 0.1   4
```

Existe pouca diferença entre o modelo `probit` e `logit`. Como o modelo logit é mais simples vamos interpretá-lo apenas.

Diagnose dos resíduos do modelo

Vamos fazer a diagnose dos resíduos do modelo (Figura 8.19).

```
## Diagnóse avançada
simulationBion <- simulateResiduals(fittedModel = mod_log, plot = T)
```

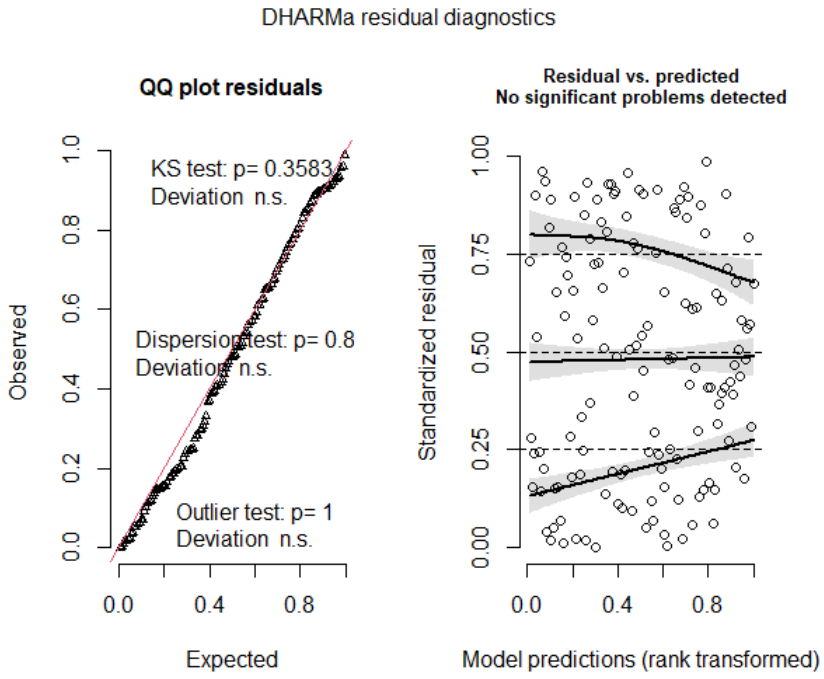


Figura 8.19: Gráfico de diagnóstico avançado do modelo

Vamos avaliar a qualidade do modelo (Figura 8.20).

```
## Diagnóse avançada
binned_residuals(mod_log)
#> Warning: About 92% of the residuals are inside the error bounds (~95% or
higher would be good).
```

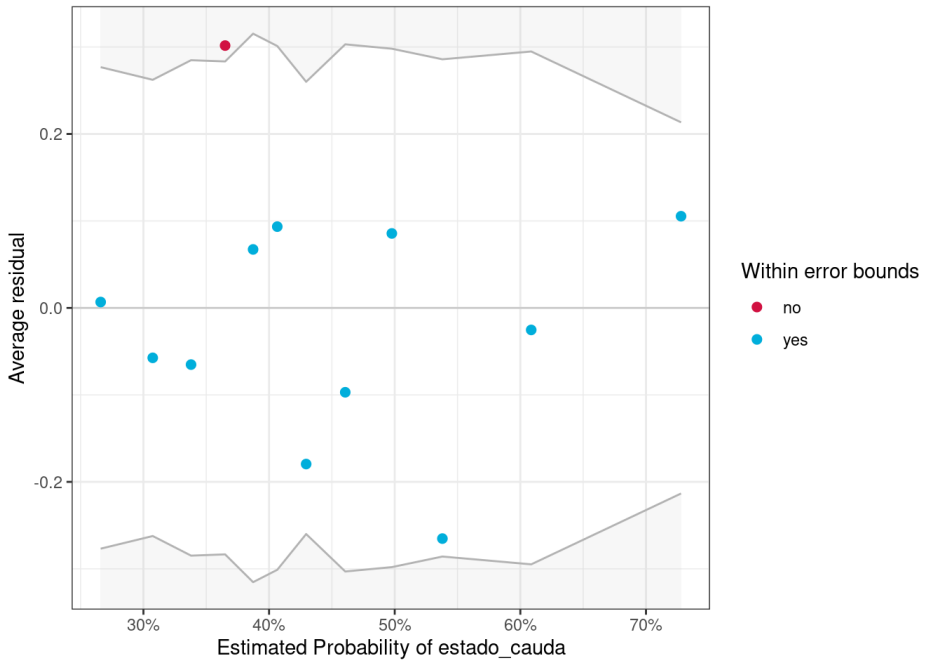


Figura 8.20: Diagnose avançada dos resíduos do modelo GLM binomial.

Inferência

Para modelos com parâmetro de dispersão conhecida (e.g., binomial e Poisson), o chi-quadrado é a estatística mais apropriada.

```
## Coeficientes estimados pelo modelo
summary(mod_log)
#>
#> Call:
#> glm(formula = estado_cauda ~ SVL * sexo, family = binomial(link =
"logit"),
#>     data = dados_semNA)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -1.5694  -1.0449  -0.8196   1.2181   1.6310
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    5.5834     2.5909   2.155   0.0312 *
#> SVL           -0.2678     0.1178  -2.274   0.0230 *
#> sexoMale       0.6977     4.4055   0.158   0.8742
#> SVL:sexoMale  -0.0442     0.2085  -0.212   0.8321
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 191.07  on 138  degrees of freedom
#> Residual deviance: 181.38  on 135  degrees of freedom
#> AIC: 189.38
#>
#> Number of Fisher Scoring iterations: 4

anova(mod_log, test = "Chisq" )
#> Analysis of Deviance Table
#>
#> Model: binomial, link: logit
#>
#> Response: estado_cauda
#>
#> Terms added sequentially (first to last)
#>
#>
#>          Df Deviance Resid. Df Resid. Dev Pr(>Chi)
#> NULL                       138     191.07
#> SVL       1   9.2563       137     181.82 0.002347 **
#> sexo      1   0.3920       136     181.43 0.531262
```

```
#> SVL:sexo  1   0.0454       135     181.38 0.831292
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interpretação dos resultados

A interpretação dos resultados é que o tamanho de corpo (SVL) afeta negativamente a probabilidade da cauda estar intacta, i.e., com o aumento do tamanho, a probabilidade da cauda permanecer intacta diminui. A interação não foi significativa, então o efeito é independente do sexo dos lagartos.

8.8 Dados de contagem com excesso de zeros

Quando se analisa a abundância ou riqueza de espécies é comum que tenhamos dados com muitos zeros. Esse fenômeno pode ser causado por vários processos ecológicos, tais como locais fora do nicho da espécie, falha na detecção, amostras feitas fora do habitat ou em locais onde não se espera encontrar a espécie (Blasco-Moreno et al. 2019). Esse tipo de dado é problemático porque rompe com os pressupostos da distribuição Poisson e binomial negativa, podendo inclusive ser uma das causas da *overdispersion*.

Nesses casos, temos de ajustar modelos que levam em conta esse excesso de zeros nos dados. Esses modelos são chamados de **zero-inflated** e **hurdle models** (também chamados de *zero-altered models*), dependendo de como o processo que causou os zeros é modelado.

Hurdle models (ou *zero-altered models*) modelam os dados dividindo-os em dois subconjuntos: um no qual reduzimos os dados à presença-ausência, ou seja, todos os dados maiores que 1 são transformados em 1 e usamos, por exemplo, uma distribuição binomial; e uma outra parte que só considera os valores positivos sem zero, utilizando uma Poisson ou binomial negativa truncadas. Ao fazer isso, a distribuição truncada assume que os zeros são gerados tanto por processos ecológicos, quanto erros de amostragem (ou seja, é impossível distinguir entre essas duas fontes). Portanto, esses zeros são excluídos da distribuição com dados de contagem. Por exemplo, se uma distribuição binomial negativa for usada para modelar a parte quantitativa, chamamos o modelo de *Zero-altered Negative binomial*. A interpretação dos modelos deve ser feita de forma conjunta.

Modelos com zero inflados funcionam de maneira similar, mas permitem que a distribuição Poisson contenha zeros, ou seja, *não é utilizada uma distribuição truncada*. Ao fazer isso, está distribuição de Poisson pressupõe que os zeros foram gerados por um processo ecológico real, tal como, ausência de habitat adequado.

Para ilustrar como podemos lidar com conjuntos de dados complexos, vamos utilizar os dados do parasita *Raillietiella mottae* infectando duas espécies de lagartos que ocorrem no Nordeste Brasileiro (Lima et al. 2018). Ao todo, 63 indivíduos de *Hemidactylus agrius* e 132 de *Phyllopezus pollicaris* foram amostrados.

Pergunta

Quais atributos de história de vida dos lagartos são relacionados com o volume (*load*) de infecção, tais como tamanho e sexo?

Predições

Quanto maior o lagarto, maior o número de parasitas encontrados, esta resposta poderia também diferir entre sexos devido ao dimorfismo sexual.

Variáveis

- Variável resposta: número do parasita *Raillietiella mottae*, que é um crustáceo parasita, infectando o aparelho respiratório e intestinal de lagartos.

```
## Cabeçalho dos dados
head(parasitas)
#>                    Especie Sexo CRC Raillietiella_mottae
#> W124 Phyllopezus_pollicaris    F  61                   3
#> W125 Phyllopezus_pollicaris    F  56                   0
#> W127 Phyllopezus_pollicaris    M  61                   0
#> W128 Phyllopezus_pollicaris    M  48                   0
#> W129 Phyllopezus_pollicaris    F  40                   0
#> W130 Phyllopezus_pollicaris    M  62                   0
```

Vamos explorar os dados (Figuras 8.21 e 8.22).

```
## Gráficos
ggplot(parasitas, aes(Raillietiella_mottae, fill = Especie)) +
    geom_density(alpha = 0.4) +
    facet_grid(Especie ~ Sexo) +
    scale_fill_manual(values = c("darkorange", "cyan4")) +
    tema_livro() +
    theme(legend.position = "none")

ggplot(parasitas, aes(CRC, Raillietiella_mottae, fill = Especie)) +
    geom_point(size = 4, alpha = 0.4, shape = 21) +
    facet_grid(Sexo ~ Especie) +
    scale_fill_manual(values = c("darkorange", "cyan4")) +
    theme(legend.position = "none") +
    labs(x = "Comprimento Rostro-Cloacal",
         y = expression(italic("Raillietiella mottae")))+
    tema_livro()
```

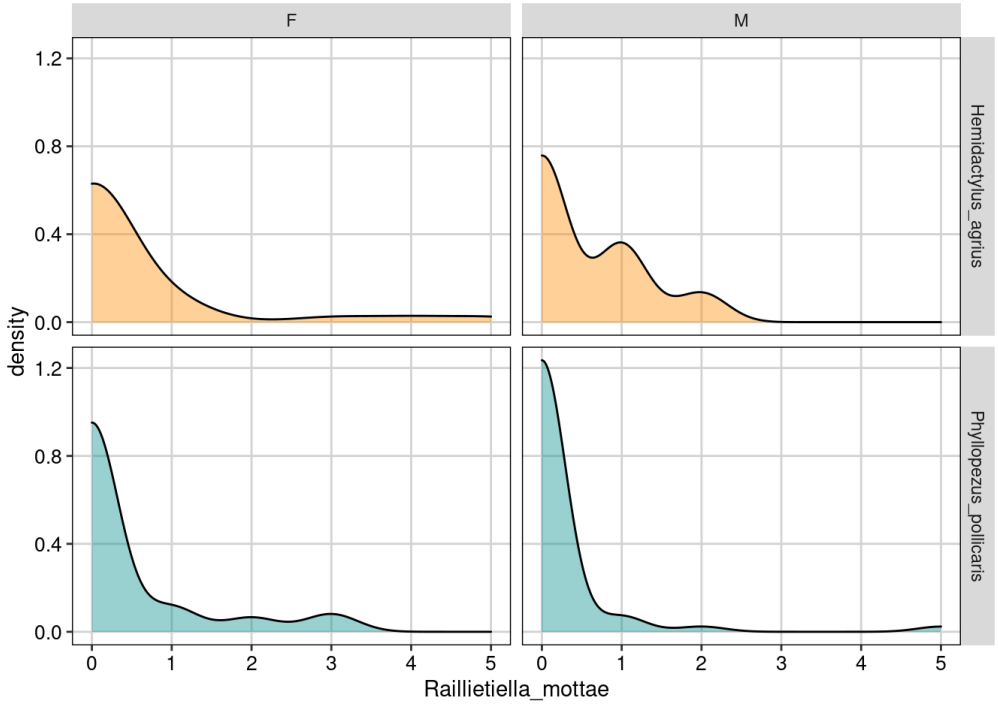


Figura 8.21: Gráfico para explorar as relações entre as variáveis.

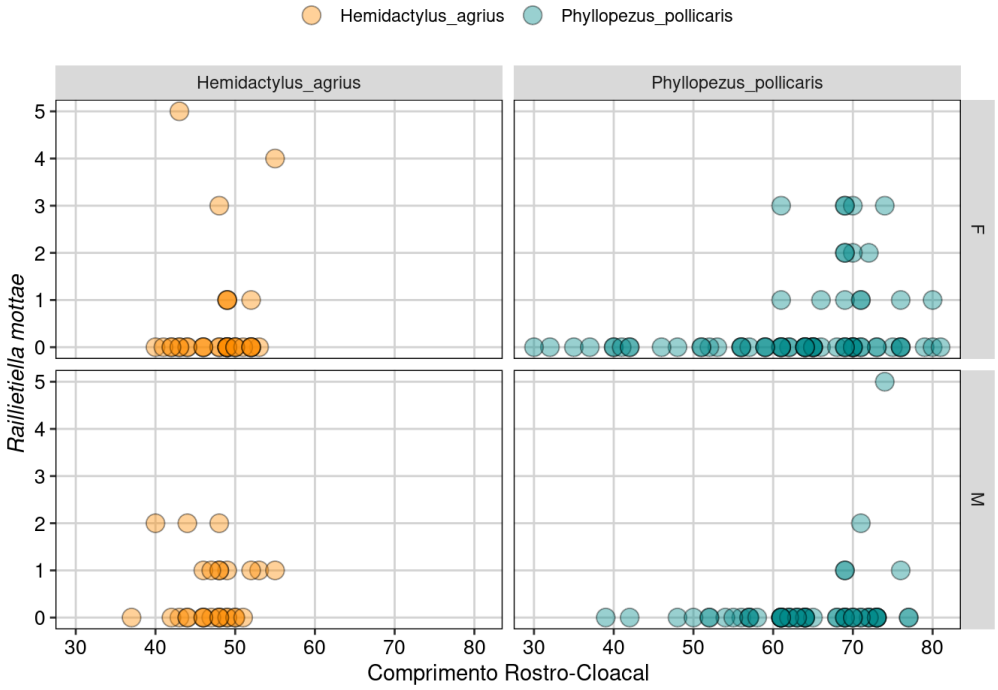


Figura 8.22: Gráfico para explorar as relações entre as variáveis.

Os gráficos acima mostram a contagem do parasita *Raillietiella mottae* nos dois sexos (F e M para fêmea e macho) nas duas espécies de lagartos (*Hemidactylus agrius* e *Phyllopezus pollicaris*), tanto na forma de uma distribuição de densidade, quanto de gráfico de dispersão. Aqui podemos ver que de fato existe um excesso de zeros principalmente em *P. pollicaris*.

Quando nos deparamos com dados complexos assim, a estratégia é sempre começar com um modelo simples e depois adicionar mais parâmetros. Portanto, vamos iniciar com um modelo Poisson, mesmo sabendo que ele muito provavelmente não será adequado para modelar estes dados.

Modelagem

Ajuste do modelo Poisson.

```
## Modelo
pois_plain <- glm(Raillietiella_mottae ~ CRC + Sexo * Especie,
                  data = parasitas, family = "poisson")
```

Diagnose

Aqui vamos utilizar as funções do pacote `performance` para o GLM Poisson.

```
## Diagnose avançada
# Verificar zero inflation
check_zeroinflation(pois_plain)
#> # Check for zero-inflation
#>
#>    Observed zeros: 156
#>   Predicted zeros: 140
#>             Ratio: 0.90

# Verificar overdispersion
check_overdispersion(pois_plain)
#> # Overdispersion test
#>
#>        dispersion ratio =   1.932
#>   Pearson's Chi-Squared = 367.133
#>                 p-value = < 0.001
```

A diagnose não só nos disse que o modelo possui *overdispersion*, como também de *zero-inflation*, como já esperávamos. Vejamos então como melhorar o nosso modelo para lidar com esses dois problemas. Especificamente, vamos utilizar um modelo *Hurdle* com binomial negativa truncada (ou seja, desconsiderando os zeros) e dois outros modelos com *zero-inflated* usando distribuição binomial negativa e Poisson. Aqui vamos utilizar o pacote `glmmTMB`.

```
## Modelos
# Hurdle model
hur_NB <- glmmTMB(Raillietiella_mottae ~ CRC + Sexo * Especie,
                  zi = ~., data = parasitas, family = truncated_nbinom2)

# zero-inflated Poisson
ziNB_mod2 <- glmmTMB(Raillietiella_mottae ~ CRC + Sexo * Especie,
                     zi = ~., data = parasitas, family = nbinom2)

# zero-inflated Negative binomial
ziP_mod2 <- glmmTMB(Raillietiella_mottae ~ CRC + Sexo * Especie,
                    zi = ~., data = parasitas, family = poisson)
```

Diagnose

Vamos fazer a diagnose desses três modelos ajustados anteriormente.

```
## Diagnose de inflação de zeros
check_zeroinflation(hur_NB)
#> # Check for zero-inflation
#>
#>    Observed zeros: 156
#>   Predicted zeros: 157
#>             Ratio: 1.01

check_zeroinflation(ziP_mod2)
#> # Check for zero-inflation
#>
#>    Observed zeros: 156
#>   Predicted zeros: 140
#>             Ratio: 0.90

check_zeroinflation(ziNB_mod2)
#> # Check for zero-inflation
#>
#>    Observed zeros: 156
#>   Predicted zeros: 142
#>             Ratio: 0.91
```

Aqui vemos que o modelo *zero-altered* (*Hurdle Model*) conseguiu predizer exatamente a quantidade de zeros observada, fazendo com que o modelo seja suficiente para usarmos com esses dados.

```
## Seleção de modelos
ICtab(pois_plain, hur_NB, ziP_mod2, ziNB_mod2, type = c("AICc"),
        weights = TRUE)
#>            dAICc df weight
#> ziP_mod2    0.0  10 0.62
#> ziNB_mod2   1.6  11 0.28
#> hur_NB      3.6  11 0.10
#> pois_plain 44.6  5  <0.001
```

Mas quando comparamos o AICc entre modelos, os modelos *zero-inflated* (tanto Poisson, quanto binomial negativa) que tem menos parâmetros, são ranqueados ligeiramente melhor do que o modelo binomial negativa *zero-altered* (ou *hurdle*). Não podemos distinguir entre os dois modelos com *zero-inflated* porque o dAICc < 2, ou seja, o ajuste deles aos dados são praticamente iguais. Vejam que a diferença de *Akaike Weights* entre os dois primeiros modelos e o *hurdle* é bastante substancial (0.34 e 0.52). Além disso, vemos que os modelos que levam em conta o excesso de zeros se ajustam bem melhor aos dados do que o modelo simples com distribuição Poisson. Vamos ver como os modelos se saem em relação aos outros pressupostos.

Modelo *hurdle* (Figuras 8.23).

```
## Diagnoses
simulationOutput <- simulateResiduals(fittedModel = hur_NB, plot = T)
```

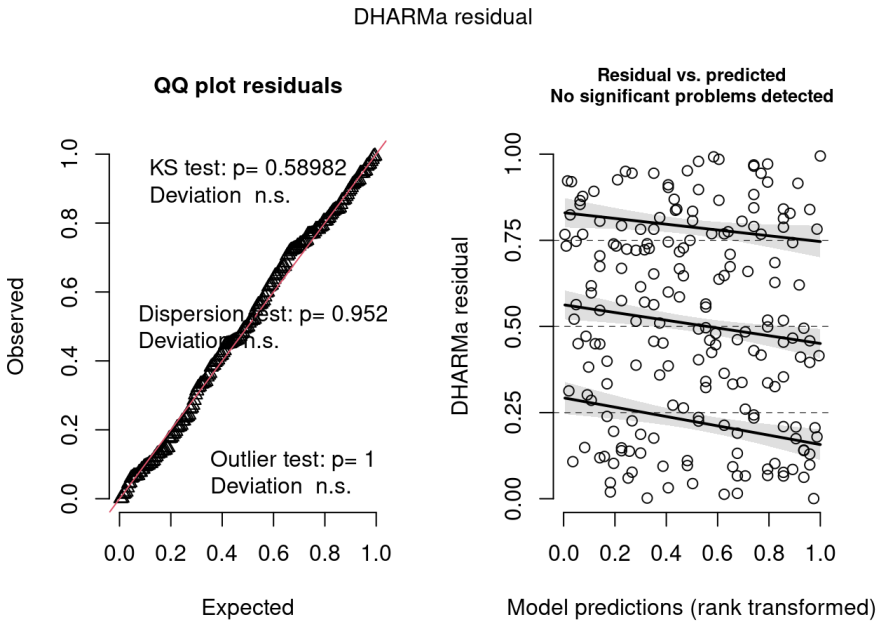


Figura 8.23: Diagnose avançada dos resíduos do modelo GLM Hurdle.

Modelo *zero-inflated* Poisson (Figuras 8.24).

```
## Diagnoses
simulationOutput <- simulateResiduals(fittedModel = ziP_mod2, plot = T)
```

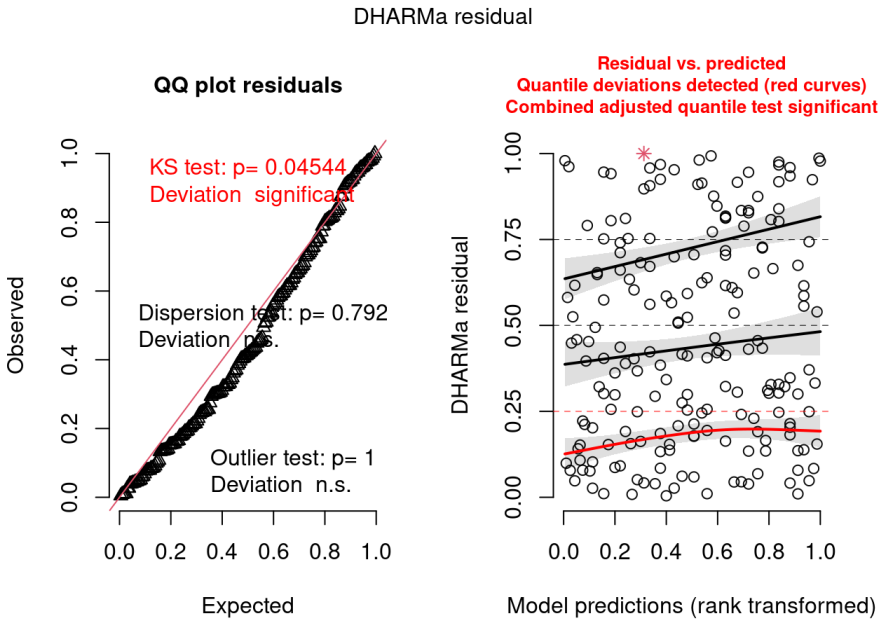


Figura 8.24: Diagnose avançada dos resíduos do modelo GLM zero-inflated Poisson.

Modelo *zero-inflated negative binomial* (Figuras 8.25).

```
## Diagnoses
simulationOutput <- simulateResiduals(fittedModel = ziNB_mod2, plot = T)
```

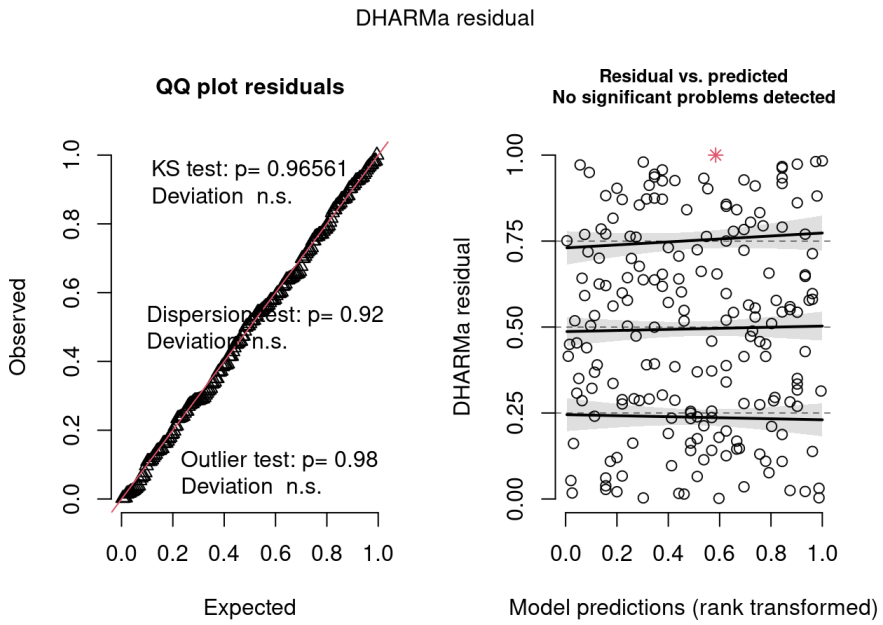


Figura 8.25: Diagnose avançada dos resíduos do modelo GLM zero-inflated negative binomial.

Os gráficos de diagnose do `DHARMa` são outra evidência de que tanto o modelo *hurdle* quanto o *zero-inflated* Poisson são adequados para os dados, em termos de homogeneidade de variância, *outliers* e *overdispersion*.

Interpretação dos resultados

Apesar de não ter um ajuste tão bom aos dados, o modelo *hurdle* prediz melhor a quantidade de zeros. Portanto, vamos interpretar os coeficientes apenas deste modelo.

```
## Coeficientes estimados pelo modelo
summary(hur_NB)
#>  Family: truncated_nbinom2  ( log )
#> Formula:          Raillietiella_mottae ~ CRC + Sexo * Especie
#> Zero inflation:                      ~.
#> Data: parasitas
#>
#>      AIC      BIC   logLik deviance df.resid
#>    277.8    313.8   -127.9    255.8      184
#>
#>
#> Dispersion parameter for truncated_nbinom2 family (): 4.64
#>
#> Conditional model:
#>                                 Estimate Std. Error z value Pr(>|z|)
#> (Intercept)                      3.03428    2.36511   1.283   0.1995
```

```
#> CRC                                  -0.05041    0.04861  -1.037   0.2997
#> SexoM                                -1.49505    0.71440  -2.093   0.0364 *
#> EspeciePhyllopezus_pollicaris         0.68945    1.09380   0.630   0.5285
#> SexoM:EspeciePhyllopezus_pollicaris  1.75281    0.94217   1.860   0.0628 .
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Zero-inflation model:
#>                                     Estimate Std. Error z value Pr(>|z|)
#> (Intercept)                           7.6283     1.8529   4.117 3.84e-05
***
#> CRC                                  -0.1291     0.0369  -3.499 0.000468
***
#> SexoM                                -1.0893     0.5867  -1.856 0.063386 .
#> EspeciePhyllopezus_pollicaris         2.2701     0.9140   2.484 0.013003 *
#> SexoM:EspeciePhyllopezus_pollicaris   2.2002     0.8192   2.686 0.007239
**
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Para maiores detalhes na interpretação deste tipo de modelo, sugerimos fortemente consultar p. 382-3 de Brooks et al. (2017). Para fatores com mais de um nível, o `summary()` mostra os resultados usando contraste, para isto toma como referência um dos níveis do fator (o primeiro em ordem alfabética) e o compara com os outros. Note que na parte com excesso de zeros, o contraste é positivo para Espécie. Ou seja, o *P. pollicaris* tem maior chance de ter ausência de parasitas que *H. agrius*. O contraste para espécie continua sendo positivo na parte condicional do modelo, mas o valor do parâmetro não é tão alto. Isso quer dizer que *P. pollicaris* tem abundância de parasitas em média ligeiramente maior que *H. agrius*. Vemos que a interação é significativa entre sexo e espécie na parte do modelo com excesso de zeros, mas apenas marginalmente significativa na parte condicional. Portanto, a influência do sexo na incidência, mas não na abundância do parasita depende conjuntamente da espécie. No entanto, o CRC só passa a ser significativo na parte de excesso de zeros, ou seja, quando modelamos apenas a incidência (presença-ausência) do parasita. Portanto, o *CRC determina se o lagarto vai ou não ser infectado, mas não o quanto vai receber de parasitas*. Já tanto o sexo quanto a espécie foram significativas em ambas as partes do modelo, ou seja, esses fatores não influenciam diferentemente a infecção e a quantidade de parasitas. Agora vejamos como podemos plotar as predições deste modelo (Figura 8.26).

```
## Gráfico
parasitas$phat <- predict(hur_NB, type = "response")
parasitas <- parasitas[with(parasitas, order(Sexo, Especie)), ]

ggplot(parasitas, aes(x = CRC, y = phat, colour = Especie,
                      shape = Sexo, linetype = Sexo)) +
    geom_point(aes(y = Raillietiella_mottae), size = 4,
               alpha = .7, position = position_jitter(h = .2)) +
    geom_line(size = 1) +
    scale_fill_manual(values = c("darkorange", "cyan4")) +
    scale_colour_manual(values = c("darkorange", "cyan4")) +
```

```
      labs(x = "Comprimento Rostro-Cloacal",
           y = expression(paste("Abundância de ",
           italic("Raillietiella mottae")))) +
  tema_livro()
```

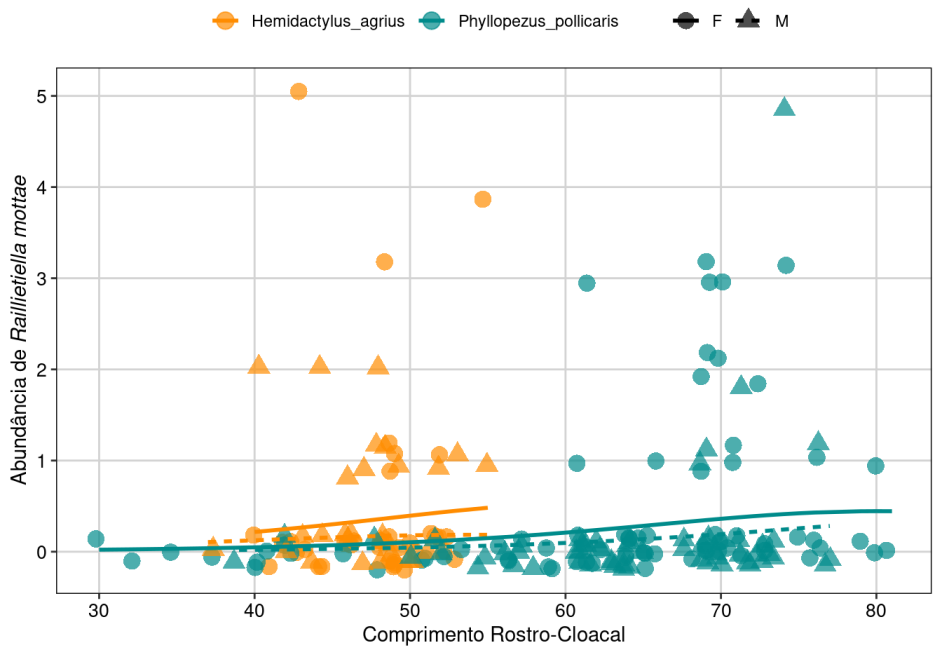


Figura 8.26: Gráfico do GLM Hurdle.

8.9 Dados ordinais: os modelos cumulative link

Uma outra maneira de codificarmos os dados é utilizando categorias ordenadas, tais como ranques. Exemplos incluem a escala de Likert, scores e intervalos (e.g., de idade).

Para este exemplo, iremos utilizar um outro conjunto de dados do artigo de (Franco-Belussi et al. 2018) que manipulou *in vitro* a concentração do hormônio noradrenalina nos olhos de peixes esganagato (*Gasterosteus aculeatus*) e avaliaram a expressão de várias cores conferidas por tipos de células (cromatóforos). Aqui vamos usar os dados do efeito da noradrenalina na cor vermelha em machos. Neste experimento, os autores realizaram medidas repetidas no mesmo animal ao longo do tempo (que é um dos fatores deste experimento). Portanto, para não incorrer no risco de pseudoréplicas deveríamos incluir esta informação no modelo. A maneira mais simples de fazê-lo é criar um modelo de efeito misto em que teríamos uma parte fixa (o que de fato estamos interessados em testar) e outra aleatória (variáveis que precisamos controlar). Portanto, `Animal` será incluído como um fator aleatório, tendo um intercepto estimado separadamente para cada animal no modelo. Não iremos entrar em detalhes sobre modelos de efeito misto porque necessitaríamos de um outro capítulo para isso, dada a complexidade do assunto. No entanto, recomendamos aos leitores dois artigos introdutórios muito bons - Harisson (2018) e Bolker et al. (2009).

Pergunta

A noradrenalina causa uma diminuição da coloração vermelha, via agregação dos pigmentos?

Predições

A presença de noradrenalina causa a agregação dos pigmentos, permitindo que os hormônios reprodutivos atuem.

Variáveis

- Variável resposta: escala de intensidade de cor. Para mais detalhes veja o artigo original (Franco-Belussi et al. 2018).

```
## Importar os dados
head(cores)
#>   Animal Treatment Time Sex Black Red
#> 1      1        CT   0h   M     5   5
#> 2      1        CT   1h   M     5   5
#> 3      1        CT   2h   M     5   5
#> 4      1        CT   3h   M     5   5
#> 5      2        CT   0h   M     5   4
#> 6      2        CT   1h   M     5   4

## Tradução dos nomes das colunas
colnames(cores) <- c("animal", "tratamento", "tempo", "sexo",
                     "preto", "vermelho")

## Filtrando dados - macho vermelho
macho_verm <- filter(cores, sexo == "M")
head(macho_verm)
#>   animal tratamento tempo sexo preto vermelho
#> 1      1         CT    0h    M     5        5
#> 2      1         CT    1h    M     5        5
#> 3      1         CT    2h    M     5        5
#> 4      1         CT    3h    M     5        5
#> 5      2         CT    0h    M     5        4
#> 6      2         CT    1h    M     5        4
```

Esses dados, no entanto, têm de ser codificados como um fator ordenado antes de entrarmos com eles no modelo.

```
## Fator
macho_verm$animal <- factor(macho_verm$animal)
macho_verm$vermelho_ord <- factor(macho_verm$vermelho,
                                  levels = c("1", "2", "3", "4", "5"),
                                  ordered = TRUE)
str(macho_verm)
#> 'data.frame':    40 obs. of  7 variables:
#>  $ animal      : Factor w/ 5 levels "1","2","3","4",..: 1 1 1 1 2 2 2 2 3
3 ...
#>  $ tratamento  : chr  "CT" "CT" "CT" "CT" ...
#>  $ tempo       : chr  "0h" "1h" "2h" "3h" ...
#>  $ sexo        : chr  "M" "M" "M" "M" ...
```

```
#>  $ preto       : int  5 5 5 5 5 5 5 5 4 4 ...
#>  $ vermelho    : int  5 5 5 5 4 4 4 4 4 4 ...
#>  $ vermelho_ord: Ord.factor w/ 5 levels "1"<"2"<"3"<"4"<..: 5 5 5 5 4 4 4
4 4 4 ...
```

Repare que a classe do objeto muda e temos agora que Red é um `Ordered factor`.

Modelagem

Vamos ajustar um modelo *cumulative link*, utilizando a função `clmm()` do pacote `ordinal`.

```
## Modelo
mod3 <- clmm(vermelho_ord ~ tratamento + tempo + (1|animal),
             data = macho_verm, threshold = "equidistant")
```

Diagnose

Infelizmente, o pacote `ordinal` não fornece métodos para lidar com modelos mistos, como o nosso. Então, montamos um modelo fixo apenas para entrar nas duas funções de diagnose. Essas duas funções `scale_test()` e `nominal_test()` testam a qualidade do ajuste (*goodness-of-fit*) do modelo, similar aos *likelihood-ratio test*, só que para dados ordinais.

```
## Diagnose
assumption3 <- clm(vermelho_ord ~ tratamento + tempo,
                   data = macho_verm, threshold = "equidistant")

scale_test(assumption3)
#> Tests of scale effects
#>
#> formula: vermelho_ord ~ tratamento + tempo
#>            Df  logLik    AIC      LRT Pr(>Chi)
#> <none>        -24.301 60.602
#> tratamento  1 -24.293 62.586 0.015248   0.9017
#> tempo

nominal_test(assumption3)
#> Tests of nominal effects
#>
#> formula: vermelho_ord ~ tratamento + tempo
#>            Df  logLik    AIC    LRT Pr(>Chi)
#> <none>        -24.301 60.602
#> tratamento  1 -19.749 53.499 9.1031 0.002552 **
#> tempo       3 -22.803 63.606 2.9953 0.392356
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Parece que não há problemas com o efeito de escala do dado ordinal, mas a diagnose sugere que possa haver evidência de rompimento do pressuposto de probabilidades proporcionais em relação ao tratamento. Esse é um pressuposto importante de modelos ordinais, os quais assumem que os efeitos de qualquer uma das variáveis explicativas são consistentes (proporcionais) ao longo de diferentes *thresholds* (que são as quebras entre cada par de categorias da variável resposta ordinal).

Isto provavelmente se deve ao baixo tamanho amostral. Por questão de brevidade vamos apenas ignorar este aspecto e interpretar o resultado do modelo mesmo assim. Mas se o seu modelo apresentar este problema, a solução deve ser realizar regressões logísticas separadamente.

Inferência

Vamos analisar os parâmetros do modelo.

```
## Coeficientes estimados pelo modelo
summary(mod3)
#> Cumulative Link Mixed Model fitted with the Laplace approximation
#>
#> formula: vermelho_ord ~ tratamento + tempo + (1 | animal)
#> data:    macho_verm
#>
#>  link  threshold   nobs logLik AIC   niter    max.grad cond.H
#>  logit equidistant 40   -22.89 59.77 226(681) 1.04e-05 4.1e+01
#>
#> Random effects:
#>  Groups Name        Variance Std.Dev.
#>  animal (Intercept) 1.438    1.199
#> Number of groups:  animal 5
#>
#> Coefficients:
#>                  Estimate Std. Error z value Pr(>|z|)
#> tratamentoNA10uM   -4.602      1.228  -3.748 0.000178 ***
#> tempo1h            -3.602      1.377  -2.616 0.008894 **
#> tempo2h            -3.602      1.377  -2.616 0.008894 **
#> tempo3h            -3.602      1.377  -2.616 0.008894 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Threshold coefficients:
#>             Estimate Std. Error z value
#> threshold.1   -6.198      1.722   -3.60
#> spacing        4.978      1.254    3.97

anova(assumption3)
#> Type I Analysis of Deviance Table with Wald chi-square tests
#>
#>            Df   Chisq Pr(>Chisq)
#> tratamento  1 15.3616  8.877e-05 ***
#> tempo       3  9.1992    0.02676 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

pairs(emmeans(mod3, ~ tratamento|tempo, adjust = "tukey"))
#> tempo = 0h:
#>  contrast    estimate   SE  df z.ratio p.value
```

```
#>  CT - NA10uM      4.6 1.23 Inf   3.748  0.0002
#>
#> tempo = 1h:
#>  contrast    estimate   SE  df z.ratio p.value
#>  CT - NA10uM      4.6 1.23 Inf   3.748  0.0002
#>
#> tempo = 2h:
#>  contrast    estimate   SE  df z.ratio p.value
#>  CT - NA10uM      4.6 1.23 Inf   3.748  0.0002
#>
#> tempo = 3h:
#>  contrast    estimate   SE  df z.ratio p.value
#>  CT - NA10uM      4.6 1.23 Inf   3.748  0.0002
```

Aqui vemos que tanto o tratamento, quanto o tempo de exposição foram significativos.

Interpretação dos resultados

Vamos analisar a predição do modelo (Figura 8.27).

```
## Gráfico

# Calcular média e erro padrão
macho_verm_res <- summarySE(macho_verm,
                        measurevar = "vermelho",
                        groupvars = c("tempo", "tratamento"))

# Definir posição de linhas e pontos no gráfico
pd <- position_dodge(0.1)

macho_verm_res %>%
  ggplot(aes(x = tempo, y = vermelho, colour = tratamento,
             group = tratamento, fill = tratamento)) +
  geom_errorbar(aes(ymin=vermelho-se, ymax=vermelho +se),
                width=.1, size = 1.1, position=pd) +
  geom_line(position=pd, size = 1.1) +
  geom_point(pch = 21, colour = "black", position=pd, size=3.5) +
  scale_colour_manual(values = c("darkorange", "cyan4")) +
  scale_fill_manual(values = c("darkorange", "cyan4")) +
  xlab("Tempo de exposição (horas)") +
  ylab("Índice de eritróforos") +
  tema_livro()
```

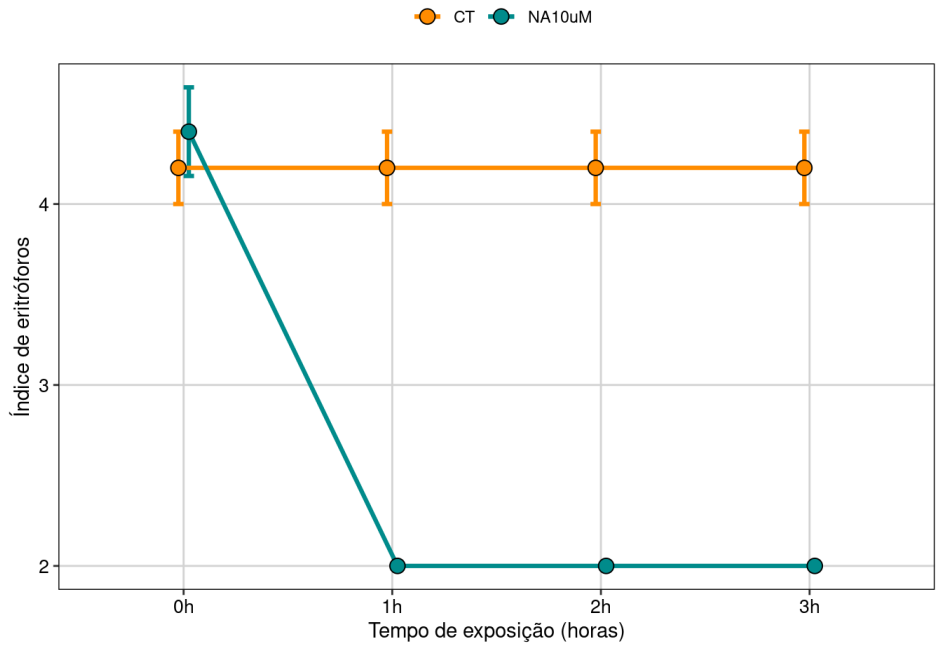


Figura 8.27: Gráfico do modelo cumulative link.

Neste gráfico vemos que o grupo tratado com o hormônio começa com um índice de eritróforo ligeiramente maior do que o controle, mas logo tem uma brusca redução com uma hora de exposição, passando a ter índice 2, demonstrando que houve uma mudança de cor induzida pela dispersão dos cromatóforos. Já no controle, o índice não muda em relação ao tempo de exposição. Logo, podemos ver que há uma interação entre os fatores devido a esta queda do índice no grupo tratado e a ausência de efeito no controle.

8.10 Dados contínuos: distribuição beta

Aqui vamos utilizar como exemplo os dados do artigo de Franco-Belussi et al. (2018). Os pesquisadores fizeram um experimento *in vivo* com peixes esgana-gato (*Gasterosteus aculeatus*) para testar como a coloração dos animais respondem ao fármaco ioimbina (YOH), que bloqueia a coloração típica que os machos exibem na época de acasalamento, e o tempo de exposição ao mesmo (além de um controle), num desenho de ANOVA fatorial. Como as medidas foram feitas repetidamente no mesmo animal, iremos incluir o `Animal` como um fator aleatório no modelo.

```
head(fish)
#>   Animal Treatment Time Sex Darkness Redness
#> 1      1        CT   0h   M    56.42  131.17
#> 2      1        CT   1h   M    49.53  133.30
#> 3      1        CT   2h   M    54.94  132.69
#> 4      1        CT   3h   M    42.19  135.37
#> 5      2        CT   0h   M    58.93  133.35
#> 6      2        CT   1h   M    52.45  133.65

## Tradução dos nomes das colunas
colnames(fish) <- c("animal", "tratamento", "tempo", "sexo",
                    "preto", "vermelho")
```

Esses dados contêm as variáveis resposta medidas no experimento: a quantidade de vermelho e preto. Além dos fatores manipulados: Tratamento (controle e presença de YOH) e tempo de exposição.

Pergunta

A YOH aumenta a coloração escura no olho e mandíbula dos peixes via dispersão dos pigmentos?

Predições

A YOH promoverá um escurecimento do corpo do animal, já que ela inibe a ação NorAdrenalia (NA).

Variáveis

- Variável resposta: a intensidade de coloração escura em peixes machos. Esses dados são expressos em termos de porcentagem e variam continuamente de 0 a 100%. Para facilitar a modelagem e nos adequarmos à maneira com que a função requer os dados, vamos simplesmente dividir por 100 para que os dados variem entre 0 e 1

Para modelar os dados vamos utilizar a função `glmmTMB`. Antes disso, vamos analisar graficamente os dados (Figura 8.28). Vamos usar apenas os dados dos machos para este exemplo.

```
## Filtrando os dados
fish$animal <- factor(fish$animal)
fish$sexo <- factor(fish$sexo)
macho_preto <- dplyr::filter(fish, sexo == "M")

## Gráfico
ggplot(macho_preto, aes(preto/100)) +
    geom_density(colour = "cyan4", fill = "cyan4", alpha = 0.4) +
    theme(legend.position = "none") +
    labs(x = "Índice de escuridão do corpo")+
    tema_livro()
```

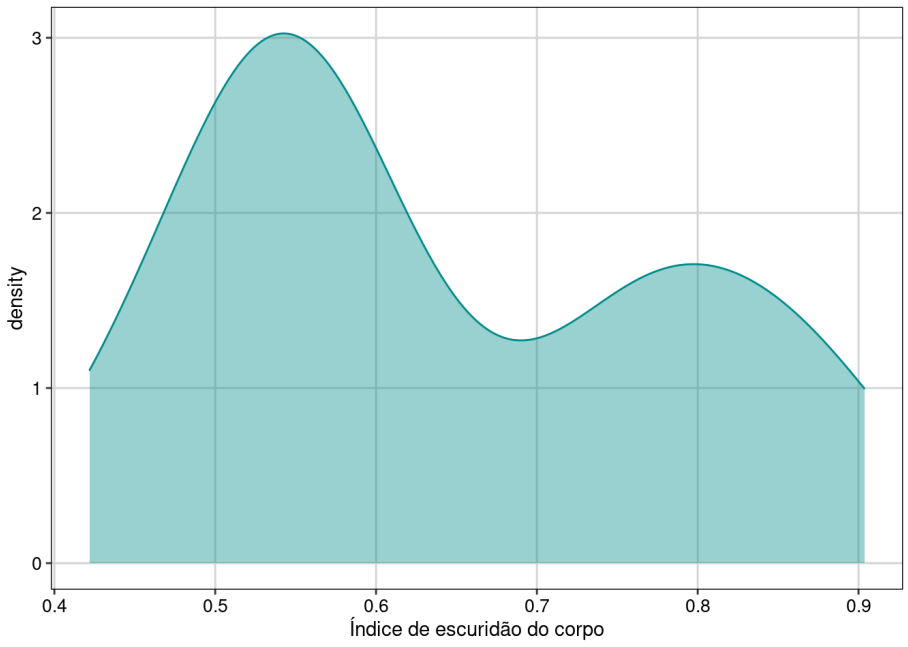


Figura 8.28: Gráfico para explorar a distribuição da variável resposta.

No histograma podemos ver que os dados de fato variam continuamente no intervalo entre 0 e 1, tendo uma distribuição notadamente bimodal.

Modelagem

Vamos ajustar um GLM Beta.

```
## Modelo
mod2 <- glmmTMB(preto/100 ~ tratamento * tempo + (1|animal),
                family = beta_family, data = macho_preto)
```

Diagnose

Aqui utilizaremos o mesmo pacote `DHARMa` para realizar a diagnose do modelo (Figura 8.29).

```
## Diagnose
simulationOutput <- simulateResiduals(fittedModel = mod2, plot = TRUE)
```

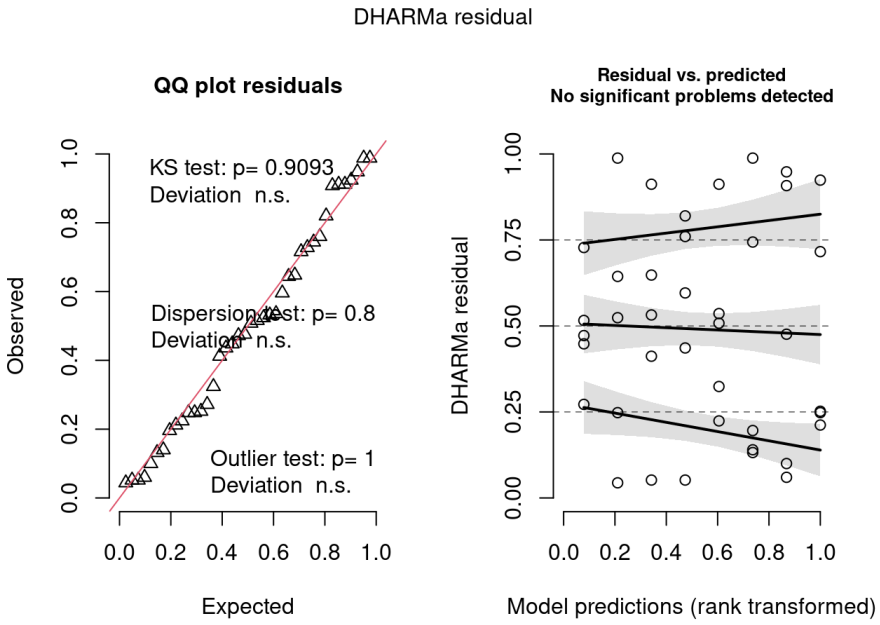


Figura 8.29: Diagnose avançada dos resíduos do modelo GLM Beta.

Podemos ver que o modelo não sofre de heterogeneidade de dispersão, *overdispersion*, nem problemas com *outlier*.

Interpretação dos resultados

Agora que podemos interpretar a saída do modelo ajustado com confiança, vamos obter a tabela de anova em que teremos os testes de cada fator do modelo.

```
## Coeficientes estimados pelo modelo
Anova(mod2)
#> Analysis of Deviance Table (Type II Wald chisquare tests)
#>
#> Response: preto/100
#>                    Chisq Df Pr(>Chisq)
```

```
#> tratamento       105.546  1  < 2.2e-16 ***
#> tempo             40.719  3  7.499e-09 ***
#> tratamento:tempo  49.262  3  1.147e-10 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Aqui vemos que a interação é significativa. Portanto, temos de interpretar os níveis do fator da combinação, fazemos isso no pacote `emmeans` colocando a barra |.

```
## níveis do fator da combinação
pairs(emmeans(mod2, ~ tratamento|tempo))
#> tempo = 0h:
#>  contrast estimate    SE df t.ratio p.value
#>  CT - YOH   0.0283 0.160 30   0.177  0.8609
#>
#> tempo = 1h:
#>  contrast estimate    SE df t.ratio p.value
#>  CT - YOH  -1.3068 0.181 30  -7.210  <.0001
#>
#> tempo = 2h:
#>  contrast estimate    SE df t.ratio p.value
#>  CT - YOH  -1.2286 0.182 30  -6.763  <.0001
#>
#> tempo = 3h:
#>  contrast estimate    SE df t.ratio p.value
#>  CT - YOH  -1.4025 0.185 30  -7.582  <.0001
#>
#> Results are given on the log odds ratio (not the response) scale.
```

Agora podemos perceber que a diferença entre o controle e o tratado só passa a ser significativa depois de 1 h de exposição.

Isso fica mais evidente quando plotamos os dados (Figura 8.30).

```
## Gráfico
escuridao <- summarySE(macho_preto,
                       measurevar = "preto",
                       groupvars = c("tempo", "tratamento"))

# Definir posição de linhas e pontos no gráfico
pd <- position_dodge(0.1)

escuridao %>%
  ggplot(aes(x = tempo, y = preto, colour = tratamento,
             group = tratamento, fill = tratamento)) +
  geom_errorbar(aes(ymin=preto-se, ymax=preto +se),
                width=.1, size = 1.1, position=pd) +
  geom_line(position=pd, size = 1.1) +
  geom_point(pch = 21, colour = "black", position=pd, size=3.5) +
```

```
  scale_colour_manual(values = c("darkorange", "cyan4")) +
  scale_fill_manual(values = c("darkorange", "cyan4")) +
  xlab("Tempo de experimento (horas)") +
  ylab("Índice de escuridão do corpo") +
  tema_livro()
```

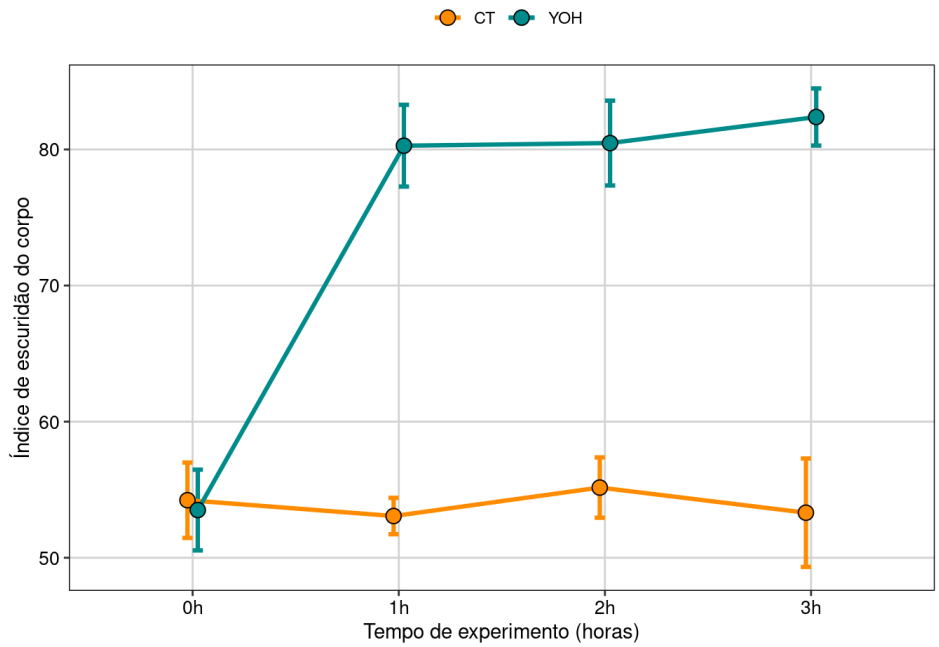


Figura 8.30: Gráfico do modelo GLM beta.

8.11 Para se aprofundar

Neste capítulo apenas fizemos uma breve introdução aos modelos lineares generalizados. Para conhecer um pouco mais a fundo todos os detalhes, recomendamos a consulta dos livros - Zurr et al. (2009) *Mixed effects models and extensions in ecology with R* e Pinheiro & Bates (2000) *Mixed-Effects Models in S and S-PLUS* - que são as referências clássicas sobre GLM com aplicações em Ecologia. Para dados ordinais, sugerimos os livros: i) Agresti (2010) *Analysis of ordinal categorical data* e ii) Agresti (2012) *Categorical Data Analysis*.

8.12 Exercícios

8.1 Baixe estes dados que foram coletados numa pesquisa de opinião sobre uso de vídeo games por 91 estudantes de graduação no outono de 1994. Utilizando estes dados, construa um modelo para: 1) Predizer a frequência com que uma pessoa joga vídeo game em função da idade; 2) Predizer a nota do estudante em função do tempo que jogou na semana antes da entrevista. Dica: esses são dados ordinais!

8.2 Uma pesquisadora interessada em entender o efeito de diferentes usos de solo sobre a abundância de morcegos em paisagens agroecológicas desenvolveu uma simulação para gerar dados parecidos

com o que irá coletar em breve no campo. Ela utilizou uma abordagem que simula a abundância de uma espécie sob deriva ecológica segundo a Teoria Neutra da Biodiversidade. A simulação gerou os seguintes dados:

```
set.seed(42)
J <- 5000 # número de indivíduos na comunidade local
theta <- 50 #número fundamental da biodiversidade
m <- 0.05 #taxa de dispersão
comm1a <- coalesc(J, m, theta) #simulação
abund1a <- abund(comm1a) #extração de dados de abundância local
```

Além disso, a pesquisadora também simulou dados que representam a porcentagem do solo destinado à agricultura:

```
#simulação uso do solo para agricultura
porc_solo <- rbeta(length(abund1a$com$ab), 10, 90)

dados_finais <- data.frame(abund = abund1a$com$ab, solo = porc_solo)
head(dados_finais)
#>   abund       solo
#> 1   161 0.11063484
#> 2    93 0.11498853
#> 3   185 0.05718937
#> 4    86 0.06134667
#> 5    35 0.18910965
#> 6    69 0.12660113
```

Agora com esses dados ela pode ter uma ideia do que esperar quando for pra campo. Ajude a pesquisadora a construir um modelo linear generalizado que seja adequado para modelar a abundância desta espécie de morcego em função da porcentagem de agricultura. Lembre-se de que ela vai precisar diagnosticar o modelo antes de utilizá-lo para fazer uma inferência. Por fim, interprete os resultados e sugira uma possível interpretação para a pesquisadora.

Soluções dos exercícios.

Capítulo 9

Análises Multidimensionais

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(ade4)
library(ecodados)
library(tidyverse)
library(vegan)
library(pvclust)
library(BiodiversityR)
library(labdsv)
library(ggplot2)
library(gridExtra)
library(ape)
library(FactoMineR)
library(factoextra)
library(FD)
library(palmerpenguins)
library(GGally)
library(fields)
library(ade4)
library(ggord)
library(udunits2)
library(adespatial)
library(spdep)
library(mvabund)
library(reshape)

## Dados
sp_compos        <- ecodados::bocaina
species          <- ecodados::com_birds
env              <- ecodados::env_birds
xy               <- ecodados::birds.xy
bocaina.env      <- ecodados::bocaina.env
bocaina.xy       <- ecodados::bocaina.xy
anuros_permanova <- ecodados::anuros_permanova
macroinv         <- ecodados::macroinv
fish_comm        <- ecodados::fish_comm
data(mite)
data(doubs)
data(mite.env)

## Traduzir nomes para português
colnames(penguins) <- c("especies", "ilha", "comprimento_bico",
                        "profundidade_bico", "comprimento_nadadeira",
                        "massa_corporal", "sexo", "ano")
```

9.1 Aspectos teóricos

Análises multivariadas avaliam hipóteses cuja variável resposta é definida por múltiplas variáveis ao mesmo tempo, comumente expressa na forma de uma matriz quadrada ou de distância (dissimilaridade).

Em geral, análises multivariadas têm três principais utilidades: i) reduzir a dimensionalidade dos dados e encontrar a principal direção de variação dos mesmos, ii) testar relações entre matrizes, ou ainda, iii) encontrar diferenças entre grupos. Análises multivariadas podem ser utilizadas como análises exploratórias e/ou para descrever padrões em estudos ecológicos. No entanto, mesmo quando se deseja apenas explorar o conjunto de dados para encontrar possíveis padrões, a necessidade de se ter hipóteses, ou ao menos expectativas *a priori*, não pode ser ignorada. Antes de entrar de cabeça nas análises multivariadas, também sugerimos fortemente o estudo de métodos de amostragem e como fazer boas perguntas (Capítulo 2).

Análises multivariadas podem ser divididas, grosseiramente, em dois tipos: agrupamento e ordenação. **Análises de agrupamento**, em geral, tentam agrupar objetos (observações) ou descritores em grupos de maneira que objetos do mesmo grupo sejam mais semelhantes entre si do que objetos de outros grupos (Legendre & Legendre 2012). Por exemplo, os objetos podem ser localidades como "parcelas," "riachos" ou "florestas," enquanto os descritores são as diferentes variáveis coletadas para esses objetos (e.g., espécies, variáveis ambientais). **A análise de ordenação**, por sua vez, é uma operação pela qual os objetos (ou descritores) são posicionados num espaço que contém menos dimensões que o conjunto de dados original; a posição dos objetos ou descritores em relação aos outros também pode ser usada para agrupá-los.

9.1.1 Coeficientes de associação

Assim chamados genericamente, os coeficientes de associação medem o quão parecidos objetos ou descritores são entre si. Objetos estão nas linhas da matriz, enquanto descritores estão nas colunas. Geralmente objetos são as nossas unidades amostrais, enquanto os descritores são as variáveis. Quando analisamos a relação entre objetos fazemos uma análise no **modo Q**, ao passo que o **modo R** é quando analisamos a relação entre descritores. Coeficientes de associação do modo Q são medidas de (dis)similaridade ou distância, enquanto para o modo R utilizamos covariância ou correlação. Como já tratamos neste livro sobre covariância e correlação (ver Capítulo 7), neste tópico vamos falar sobre índices de distância e similaridade. Mas qual a definição destas duas quantidades?

- Similaridade são máximas ($S=1$) quando dois objetos são idênticos
- Distâncias são o contrário da similaridade ($D=1-S$) e não têm limites superiores (dependem da unidade de medida)

Existem ao menos 26 índices de similaridade que podem ser agrupados de acordo com o tipo de dado (qualitativos ou quantitativos) ou a maneira com que lidam com duplos zeros (simétricos ou assimétricos) (Legendre & Legendre 2012). Do seu lado, as distâncias só se aplicam a dados quantitativos e têm como características serem métricas, semi-métricas ou não-métricas. Vejamos agora os principais índices de similaridade e distância de cada tipo.

9.1.2 Métricas de distância

O principal coeficiente de distância usado em ecologia é a distância euclidiana. Além disso, temos ainda *Canberra* (variação da Distância Euclidiana), *Mahalanobis* (calcula a distância entre dois pontos num espaço não ortogonal, levando em consideração a covariância entre descritores), *Manhattan* (variação da Distância Euclidiana), *Chord* (elimina diferenças entre abundância total de espécies), $\chi 2$ (dá peso maior para espécies raras) e *Hellinger* (não dá peso para espécies raras). Essas distâncias são recomendadas nos casos em que as variáveis de estudo forem contínuas, como por exemplo, **variáveis morfométricas ou descritores ambientais**.

Uma característica comum de conjuntos de dados ecológicos são os vários zeros encontrados em matrizes de composição. Eles surgem porque não encontramos nenhum indivíduo de uma determinada espécie num local, seja porque aquele local não tem as condições ambientais adequadas a ela, falha na detectabilidade, ou dinâmicas demográficas estocásticas de colonização-extinção (Blasco-Moreno et al. 2019). Logo, quando dois locais compartilham ausência de espécies, não é possível atribuir uma única razão da dupla ausência. Como essas medidas de distância apresentadas acima assumem que os dados são quantitativos e não de contagem, elas não são adequadas para lidar com dados de abundância ou incidência de espécies, porque atribuem um grau de parecença a pares de locais que compartilham zeros (Legendre & Legendre 2012). Por esse motivo, precisamos de coeficientes que desconsiderem os duplos zeros. Eles são chamados de *assimétricos*.

Coeficientes assimétricos binários para objetos

Esses coeficientes (ou índices) são apropriados para dados de incidência de espécies (presença-ausência) e desconsideram as duplas ausências. Os índices deste tipo mais comuns utilizados em ecologia são *Jaccard*, *Sørensen* e *Ochiai*.

O coeficiente de *Jaccard* é dado por:

$$\beta_j = a/(a+b+c)$$

onde

- a = número de espécies compartilhadas
- b = número de espécies exclusivas da comunidade 1
- c = número de espécies exclusivas da comunidade 2

A diferença entre os índices de *Jaccard* e *Sørensen* é que o índice de *Sørensen* dá peso dobrado para duplas presenças. Por conta dessas características, estes índices são adequados para quantificar diversidade beta (Anderson et al. 2011, Legendre & De Cáceres 2013). Esses índices variam entre 0 (nenhuma espécie é compartilhada entre o par de locais) a 1 (todas as espécies são compartilhadas entre o par de locais).

O coeficiente de Sørensen é dado por:

$$\beta_s = 2\,a/(2a+b+c)$$

onde

- a = número de espécies compartilhadas
- b = número de espécies exclusivas da comunidade 1
- c = número de espécies exclusivas da comunidade 2

Coeficientes binários para descritores (R mode)

Se o objetivo for calcular a similaridade entre descritores binários (e.g., presença ou ausência de características ambientais) de pares de locais, geralmente o coeficiente recomendado é o de Sokal & Michener. Este índice está implementado na função `dist.binary()` do pacote `ade4`.

Coeficientes quantitativos para objetos

Estes são os coeficientes utilizados para dados de contagem (e.g., abundância) e quantitativos (e.g., frequência, biomassa, porcentagem de cobertura). Diferentemente das distâncias, estes coeficientes são assimétricos, ou seja, não consideram duplas ausências e, portanto, são adequados para analisar dados de composição de espécies. Além disso, uma outra característica deles é serem semi-métricos. Os índices mais comuns deste tipo são *Bray-Curtis* (conhecido como *percentage difference*, em inglês), *Chord*, *log-Chord*, *Hellinger*, chi-quadrado e *Morisita-Horn*.

Todos os índices discutidos até aqui estão implementados nas funções `ade4::dist.ktab()`, `adespatial::dist.ldc()` e `vegan::vegdist()`.

Coeficientes para descritores (R mode) que incluem mistura de tipos de dados

É comum em análises de diversidade funcional que tenhamos um conjunto de atributos (*traits*) de espécies que são formados por vários tipos de dados: quantitativos (e.g., tamanho de corpo), binários (presença/ausência de uma dada característica), *fuzzy* (um atributo multiestado codificado em várias colunas), ordinais e circulares (e.g., distribuição de uma fenofase ao longo de um ano). O índice que lida com todos esses dados é o Gower. A versão estendida do índice de Gower pode ser encontrada na função `ade4::dist.ktab().`

O capítulo 7 de Legendre & Legendre (2012) fornece uma chave dicotômica para escolha do índice mais adequado.

Padronizações e transformações

É comum coletarmos múltiplas variáveis ambientais cujas unidades sejam diferentes. Por exemplo, temperatura (ºC), distância da margem (m), área (m^2), etc. Para diminuir a taxa de Erro do Tipo I das análises (rejeitar a hipótese nula quando ela é verdadeira), é recomendado que padronizemos os dados utilizando distribuição *Z*, assim todas as variáveis passam a ter média 0 e desvio padrão 1. Essa operação garante que todas as variáveis tenham o mesmo peso nas análises que avaliam o padrão dos objetos considerando os múltiplos descritores. Essa padronização pode ser implementada na função `vegan::decostand()`.

Outro problema comum de matrizes de dados de composição de espécies é o alto número de zeros, enquanto outras espécies podem ter altas abundâncias. Isso gera problemas em ordenações. Para diminuir essa discrepância, podemos **transformar** os dados, por exemplo, utilizando a distância de *Hellinger* ou *Chord*. Para dados contínuos pode ser que transformação log ou raiz quadrada ajude quando há valores muito discrepantes (*leverages*) que podem influenciar em demasiado a relação entre objetos ou descritores. Isso pode ser feito na função `vegan::decostand()`.

9.2 Análises de agrupamento

O objetivo da análise de agrupamento é agrupar objetos admitindo que haja um grau de similaridade entre eles. Esta análise pode ser utilizada ainda para classificar uma população em grupos homogêneos de acordo com uma característica de interesse. A grosso modo, uma análise de agrupamento tenta resumir uma grande quantidade de dados e apresentá-la de maneira fácil de visualizar e entender (em geral, na forma de um dendrograma). No entanto, os resultados da análise podem não refletir necessariamente toda a informação originalmente contida na matriz de dados. Para avaliar o quão bem uma análise de agrupamento representa os dados originais existe uma métrica — o coeficiente de correlação cofenético — o qual discutiremos em detalhes mais adiante.

Antes de considerar algum método de agrupamento, pense porque você esperaria que houvesse uma descontinuidade nos dados; ou ainda, considere se existe algum ganho prático em dividir uma nuvem de objetos contínuos em grupos. O padrão apresentado pelo dendrograma depende do protocolo utilizado (método de agrupamento e índice de dissimilaridade); os grupos formados dependem do nível de corte escolhido.

A matriz deve conter os objetos a serem agrupados (e.g., espécies) nas linhas e as variáveis (e.g., locais de coleta ou medidas morfológicas) nas colunas. A escolha do método de agrupamento é crítica para a escolha de um coeficiente de associação. É importante compreender as propriedades dos métodos de agrupamento para interpretar corretamente a estrutura ecológica que eles evidenciam (Legendre & Legendre 2012). De acordo com a classificação de Sneath & Sokal (1973), existem cinco tipos de métodos: i) sequenciais ou simultâneos, ii) aglomerativo ou divisivo, iii) monotéticos ou politéticos, iv) hierárquico ou não hierárquicos e v) probabilístico. Sugerimos a leitura do livro citado anteriormente para aprofundar seus conhecimentos sobre os diferentes métodos.

9.2.1 Agrupamento hierárquico

Métodos hierárquicos podem ser divididos naqueles que consideram o centroide ou a média aritmética entre os grupos. O principal método hierárquico que utiliza a média aritmética é o UPGMA (Agrupamento pelas médias aritméticas não ponderadas), e o principal método que utiliza centroides é a Distância mínima de Ward.

O UPGMA funciona da seguinte forma: a maior similaridade (ou menor distância) identifica os próximos agrupamentos a serem formados. Após esse evento, o método calcula a média aritmética das similaridades ou distâncias entre um objeto e cada um dos membros do grupo ou, no caso de um grupo previamente formado, entre todos os membros dos dois grupos. Todos os objetos recebem pesos iguais no cálculo.

O método de Ward é baseado no critério de quadrados mínimos (OLS), o mesmo utilizado para ajustar um modelo linear (Capítulo 7). O objetivo é definir os grupos de maneira que a soma de quadrados (i.e., similar ao erro quadrado da ANOVA) dentro dos grupos seja minimizada (Borcard et al. 2018).

No entanto, para interpretar os resultados precisamos antes definir um nível de corte, que vai nos dizer quantos grupos existem. Há vários métodos para definir grupos, desde os heurísticos aos que utilizam reamostragem (*bootstrap*). Se quisermos interpretar este dendrograma, podemos, por exemplo, estabelecer um nível de corte de 50% de distância (ou seja, grupos cujos objetos tenham ao menos 50% de similaridade entre si).

Checklist

- Verifique se não há espaço nos nomes das colunas e linhas
- Se os dados forem de abundância, recomenda-se realizar a transformação de *Hellinger* (Legendre & Gallagher 2001). Esta transformação é necessária porque a matriz de comunidades (em especial, com a presença de muitas espécies raras) pode causar distorções nos métodos de ordenação baseados em distância Euclidiana (Legendre & Gallagher 2001)
- Se a matriz original contiver muitos valores discrepantes (e.g., uma espécie muito mais ou muito menos abundante que outras) é necessário transformar os dados usando `log1p()`. No entanto, deve-se fazer ou a transformação de *Hellinger* ou logarítmica e nunca as duas ao mesmo tempo
- Se as variáveis forem medidas tomadas em diferentes escalas (metros, graus celsius etc.), é necessário padronizar cada variável para ter a média 0 e desvio padrão 1. Isso pode ser feito utilizando a função `decostand()` do pacote `vegan`

Exemplo 1

Neste exemplo, vamos utilizar um conjunto de dados que contém girinos de espécies de anuros coletados em 14 poças com diferentes coberturas de dossel (Provete et al. 2014).

Pergunta

- Existem grupos de espécies de anfíbios anuros com padrões de ocorrência similar ao longo das poças?

Predições

- Iremos encontrar ao menos dois grupos de espécies: aquelas que ocorrem em poças dentro de floresta (i.e., maior cobertura de dossel) *versus* aquelas que ocorrem em poças de áreas abertas (menor cobertura de dossel)

Variáveis

- Variáveis preditoras: a matriz de dados contém a abundância das espécies nas linhas e locais (poças) nas colunas

Análises

Para começar, vamos primeiro importar os dados e depois calcular a matriz de distância que seja adequada para o tipo de dado que temos (abundância de espécies - dados de contagem) (Figura 9.1).

```
## Composição de espécies (seis primeiras localidades)
head(sp_compos)
#>         BP4  PP4 PP3  AP1 AP2 PP1 PP2 BP9  PT1 PT2 PT3 BP2 PT5
#> Aper      0    3   0    0   2   0   0   0    0   0   0 181   0
#> Bahe    859   14  14    0  87 312 624 641    0   0   0  14   0
#> Rict   1772 1517 207  573 796   0   0   0    0   0   0   0   0
#> Cleuco    0    0   0    0   0   0   0   0    0  29 369   0  84
#> Dmic      0    0   6   60   4   0   0   0 2758 319  25   0 329
#> Dmin      0   84 344 1045  90   0   0   0    8   0   0   0   0

## Matriz de similaridade com o coeficiente de Morisita-Horn
```

```
distBocaina <- vegdist(x = sp_compos, method = "horn")

## Agrupamento com a função hclust e o método UPGMA
dendro <- hclust(d = distBocaina, method = "average")

## Visualizar os resultados
plot(dendro, main = "Dendrograma",
     ylab = "Similaridade (índice de Horn)",
     xlab="", sub="")
```

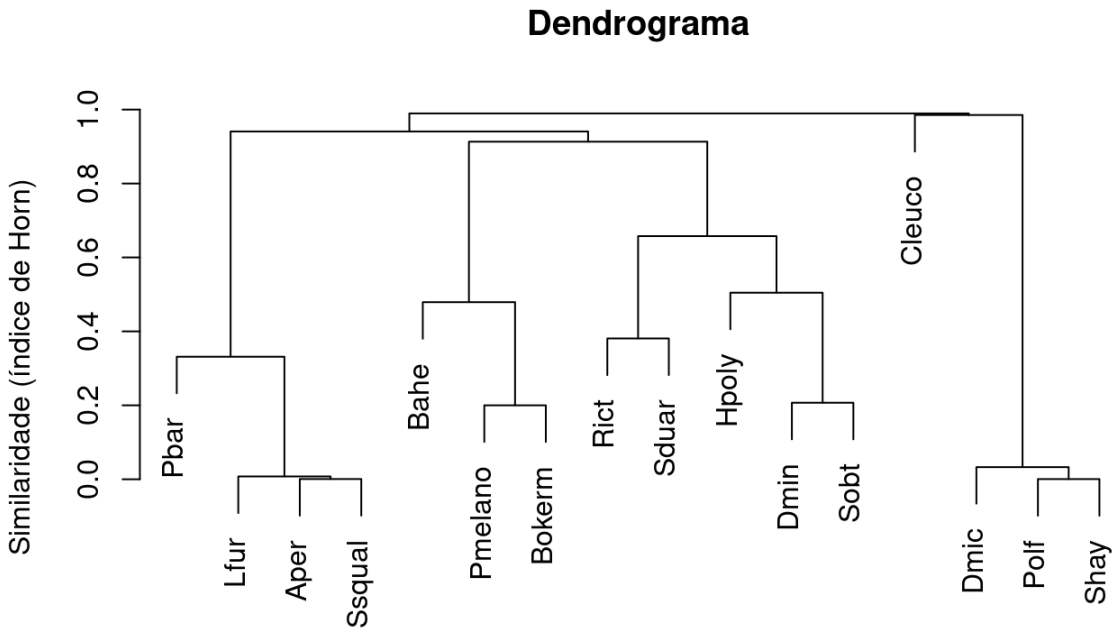


Figura 9.1: Dendrograma mostrando uma análise de agrupamento de anuros.

Avaliando a qualidade do dendrograma

Precisamos verificar se o agrupamento reduziu a dimensionalidade da matriz de forma eficiente, de maneira a não distorcer a informação. Fazemos isso calculando o **Coeficiente de Correlação Cofenética** que é uma medida que nos indica quão bem o resultado do agrupamento corresponde às (dis)similaridades originais.

```
## Coeficiente de correlação cofenética
cofresult <- cophenetic(dendro)
cor(cofresult, distBocaina)
#> [1] 0.9455221
```

Um coeficiente de correlação cofenética > .7 indica uma boa representação. Portanto, o nosso resultado de `0.9455221` é alto, garantindo que o dendrograma é adequado (Figura 9.2). Note que testar a "significância" deste coeficiente não é adequado, já que seria algo tautológico (circular). Claro que definir o que seria um valor "alto" ou "baixo" da correlação é um pouco arbitrário, mas pode-se usar como "regra do polegar."

```
## Gráfico
plot(dendro, main = "Dendrograma",
     ylab = "Similaridade (índice de Horn)",
     xlab="", sub="")
k <- 4
n <- ncol(sp_compos)
MidPoint <- (dendro$height[n-k] + dendro$height[n-k+1]) / 2
abline(h = MidPoint, lty=2)
```

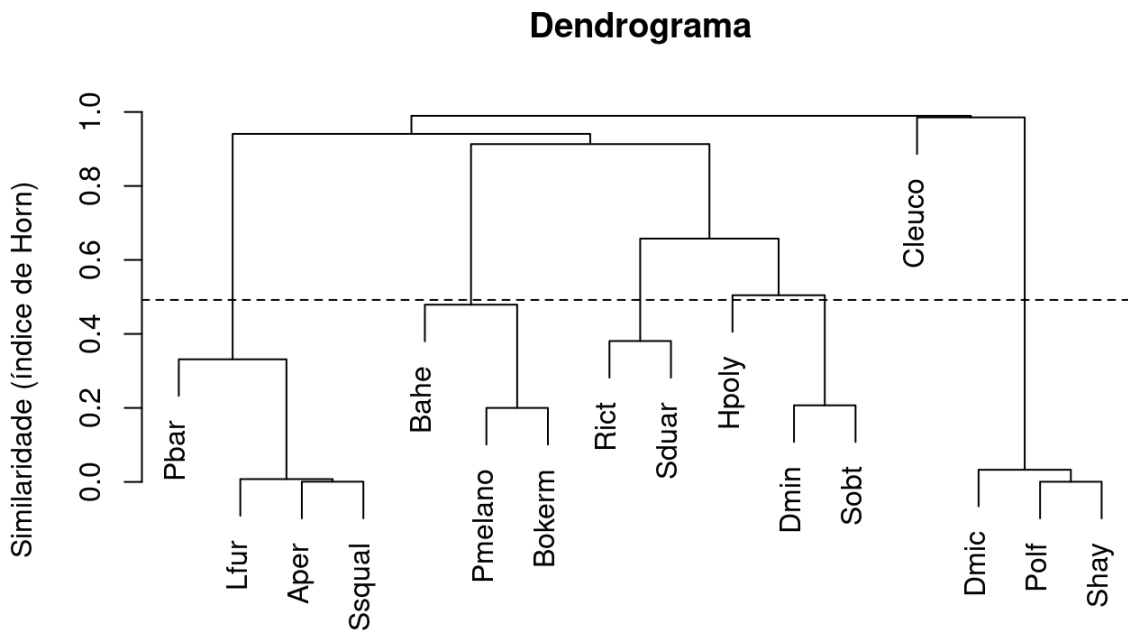


Figura 9.2: Dendrograma mostrando uma análise de agrupamento de anuros com uma linha de corte formando cinco grupos.

Nesse caso teremos a formação de cinco grupos, representados pelos nós que estão abaixo da linha de corte. Portanto, o resultado não suporta a nossa hipótese *a priori* que predizia a formação de apenas dois grupos de espécies.

Exemplo 2

No exemplo anterior, vimos que é difícil interpretar os grupos baseado num nível de corte. A seguir, vamos utilizar o pacote `pvclust` que calcula automaticamente o nível de corte de similaridade baseado no *Bootstrap* de cada nó. Uma desvantagem deste método é que ele somente aceita índices de similaridade da função `dist()`, que possui apenas a distância *Euclidiana*, *Manhattan* e *Canberra*. Uma maneira de contornarmos essa limitação é utilizar transformações dos dados disponíveis na função `disttransform()` no pacote `BiodiversityR` ou a função `decostand()` do pacote `vegan`. Também é possível utilizar a transformação de Box-Cox para dados multivariados, disponível no material suplementar de Legendre & Borcard (2018). Esta transformação é geralmente utilizada para tornar a distribuição dos dados mais simétrica (menos enviesada para valores extremos: reduzir o *skewness* dos dados).

Análises

Vamos utilizar o mesmo conjunto de dados do Exemplo 1 para responder à mesma pergunta. Aqui vamos utilizar a distância de *Chord* (que é indicada para dados de composição de espécies) para calcular a matriz de distância. Se transformarmos uma matriz usando a transformação *Chord* e depois calcularmos a distância Euclidiana, isso equivale a calcular diretamente a distância de Chord (Figura 9.3).

```
## Dados
head(t(sp_compos))
#>     Aper Bahe Rict Cleuco Dmic Dmin Hpoly Lfur Pbar Polf Pmelano Sduar
Shay Sobt Ssqual Bokerm
#> BP4    0  859 1772      0    0    0    61    3  387    0       0     0
0    0      0      0
#> PP4    3   14 1517      0    0   84   275    0  187    0       0  1150
6    0      0      1
#> PP3    0   14  207      0    6  344   388    0    0    0       0   428
0    0      0      0
#> AP1    0    0  573      0   60 1045  1054    0    0    0       0   476
92   13      0      0
#> AP2    2   87  796      0    4   90  3002    0    0    2       0     7
0    5      0      0
#> PP1    0  312    0      0    0    0   329    0    0    0       0     0
0    0      0      0

## Passo 1: transformar para distância de Chord
bocaina_transf <- disttransform(t(sp_compos), "chord")

## Passo 2: realizar pvclust com método average e distância euclidiana
analise <- pvclust(bocaina_transf, method.hclust = "average",
                   method.dist = "euclidean", quiet = TRUE)

## Passo 3: dendrograma
plot(analise, hang=-1, main = "Dendrograma com valores de P",
     ylab = "Distância Euclideana",
     xlab="", sub="")
pvrect(analise)
```

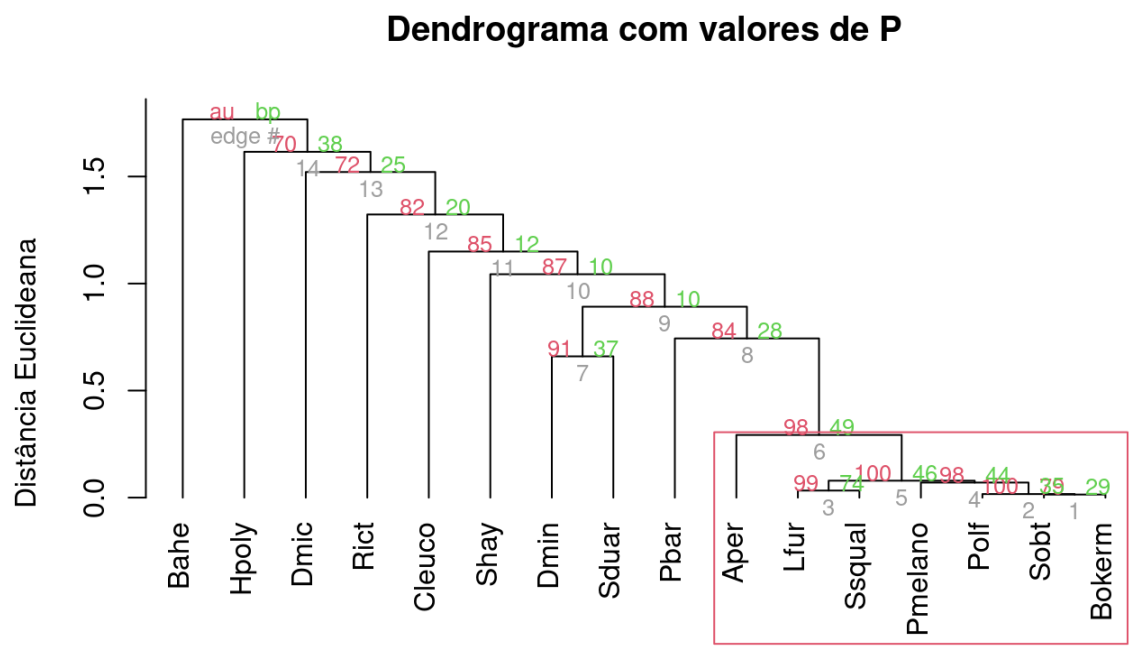


Figura 9.3: Dendrograma mostrando uma análise de agrupamento de anuros com uma linha de corte criada por bootstrap e usando distância de Chord.

É possível notar que existe um único grupo com BS > %95. Agora vamos tentar usar a distância de *Hellinger*, que é recomendada (junto com a distância de *Chord*) para transformar dados de composição de espécies (Legendre & Gallagher 2001) (Figura 9.4).

```
## Passo 1: transformar dados com Hellinger
bocaina_transf2 <- disttransform(t(bocaina), "hellinger")

## Passo 2: realizar pvclust com método average e distância euclidiana
analise2 <- pvclust(bocaina_transf2, method.hclust="average",
                    method.dist="euclidean", quiet = TRUE)

## Passo 3: dendrograma
plot(analise2, hang=-1, main = "Dendrograma com valores de P",
     ylab = "Distância Euclideana",
     xlab="", sub="")
k <- 4
n <- ncol(sp_compos)
MidPoint <- (dendro$height[n-k] + dendro$height[n-k+1]) / 2
abline(h = MidPoint, lty=2)
pvrect(analise2)
```

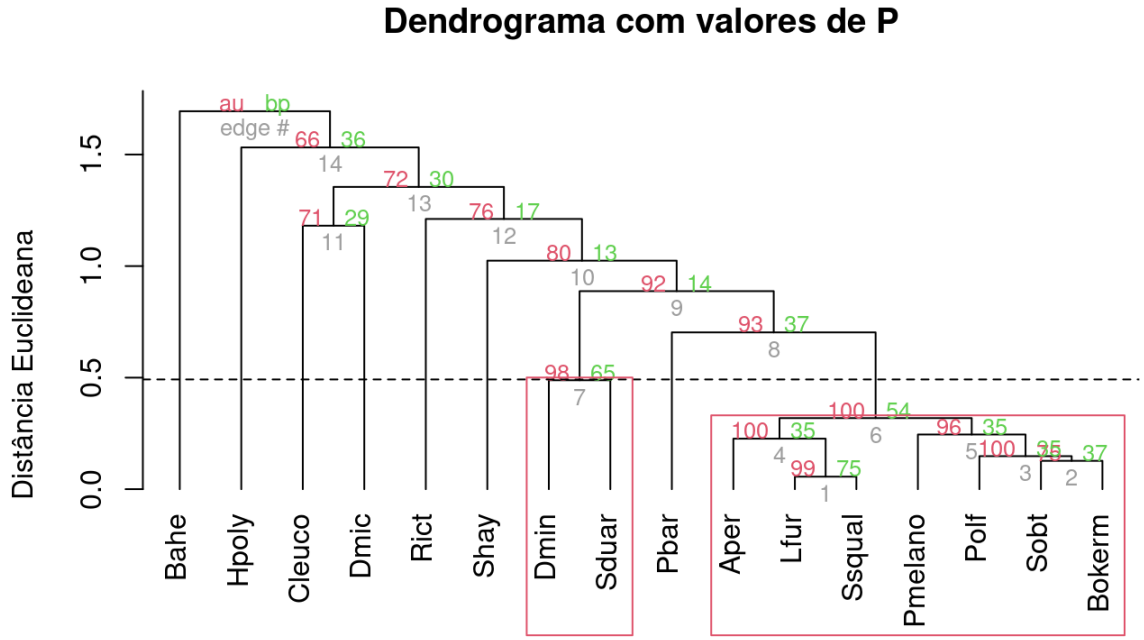


Figura 9.4: Dendrograma mostrando uma análise de agrupamento de anuros com uma linha de corte criada por bootstrap e usando distância de Hellinger.

Interpretação dos resultados

Notem que se mudarmos o coeficiente de associação, o resultado também muda. Agora temos um grupo a mais, composto por *Dendropsophus minutus* e *Scinax duartei* que não apareciam antes. Isso se deve ao fato de que a distância de *Hellinger* dá menos peso para espécies raras do que a *Chord*.

Neste sentido, os dados não suportam a nossa hipótese inicial da formação de dois grupos, independentemente do coeficiente de associação utilizado e do cálculo automático do nível de corte baseado na reamostragem.

9.2.2 Agrupamento não-hierárquico (*K-means*)

Ao contrário do dendrograma, o *K-means* é um agrupamento não-hierárquico e, desse modo, não é otimizado para buscar grupos menores aninhados em grupos maiores. Resumidamente, podemos calcular o *K-means* a partir de uma matriz quadrada ou de distância. Essa técnica procura particionar os objetos em *k* grupos de maneira a minimizar a soma de quadrados entre grupos e maximizá-la dentro dos grupos. Um critério similar ao de uma ANOVA (Capítulo 7). Um diferencial do *K-means* em relação aos agrupamentos hierárquicos é que o usuário pode escolher antecipadamente o número de grupos que deseja formar.

Exemplo 1

Para este exemplo, iremos utilizar um conjunto de dados disponível no pacote `ade4` que contém dados de 27 espécies de peixes coletados em 30 pontos ao longo do Rio Doubs, na fronteira entre a França e Suíça.

Pergunta

- Qual é o número de grupos que melhor sumariza o padrão de ocorrência de espécies de peixes ao longo de um riacho?

> **Importante**
> Neste caso, estamos realizando uma análise exploratória e não temos uma predição.

Variáveis

- Variáveis resposta: composição de espécies de peixes

Checklist

- Vamos normalizar os dados de abundância antes de entrar na análise propriamente, já que existem muitos zeros na matriz

Análises

Vamos iniciar selecionando e padronizando os dados.

```
## Mostrar somente seis primeiras espécies de seis localidades
head(doubs$fish)[,1:6]
#>   Cogo Satr Phph Neba Thth Teso
#> 1    0    3    0    0    0    0
#> 2    0    5    4    3    0    0
#> 3    0    5    5    5    0    0
#> 4    0    4    5    5    0    0
#> 5    0    2    3    2    0    0
#> 6    0    3    4    5    0    0

## Verificar se existem localidades sem nenhuma ocorrência
rowSums(doubs$fish)
#>  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30
#>  3 12 16 21 34 21 16  0 14 14 11 18 19 28 33 40 44 42 46 56 62 72  4 15 11
43 63 70 87 89

## Retirar a linha 8 (rio sem nenhuma ocorrência de peixe)
spe <- doubs$fish[-8,]

## Função do pacote vegan para normalizar os dados
spe.norm <- decostand(x = spe, method = "normalize")
```

O argumento `centers` na função `kmeans()` indica o número de grupos que se quer formar. Neste exemplo, estamos utilizando `centers = 4`.

```
## K-Means
spe.kmeans <- kmeans(x = spe.norm, centers = 4, nstart = 100)
spe.kmeans
```

O objeto que fornece o resultado contém: i) o tamanho (número de objetos) em cada um dos 4 grupos, ii) o centroide de cada grupo e o pertencimento de cada espécie a cada grupo, e iii) o quanto da Soma de Quadrados dos dados é explicada por esta conformação de grupos.

No entanto, não é possível saber *a priori* qual o número "ideal" de grupos. Para descobrir isso, repetimos o *k-means* com uma série de valores de **K**. Isso pode ser feito na função `cascadeKM()`.

```
## Repetindo o K-Means
spe.KM.cascade <- cascadeKM(spe.norm, inf.gr = 2, sup.gr = 10,
                            iter = 100, criterion = "ssi")
```

Tanto **calinski** quando **ssi** são bons critérios para encontrar o número ideal de grupos. Quanto maior o valor de **ssi**, melhor (veja `?cascadeKM()` mais detalhes). Os valores de ssi é o critério utilizado pelo algoritmo para achar o agrupamento ótimo dos objetos (Figura 9.5).

```
## Resumo dos resultados
spe.KM.cascade$results
#>      2 groups  3 groups  4 groups  5 groups   6 groups  7 groups  8 groups
9 groups 10 groups
#> SSE 8.2149405 6.4768108 5.0719796 4.3015573 3.58561200 2.9523667 2.4840549
2.0521888 1.7599292
#> ssi 0.1312111 0.1685126 0.1409061 0.1299662 0.08693436 0.1481826 0.1267918
0.1134307 0.1226392

## Gráfico
plot(spe.KM.cascade, sortg = TRUE)
```

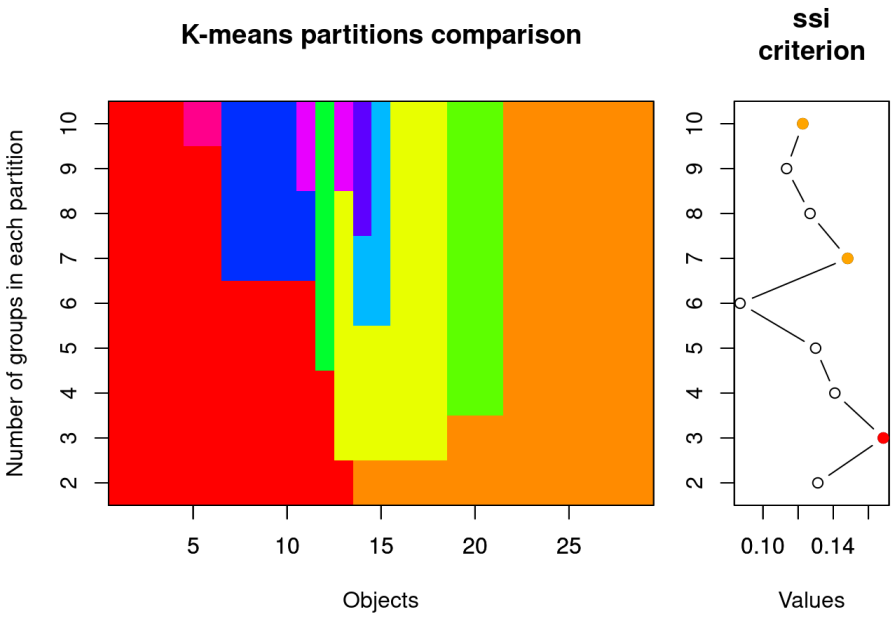


Figura 9.5: Gráficos mostrando os resultados da análise de K-Means.

Interpretação dos resultados

Diferentemente da nossa predição inicial, o resultado da análise mostra que o número ideal de grupos para explicar a variância no padrão de ocorrência de espécies é 3. Notem que o SSI máximo é alcançado neste número de grupos `0.1685126` (também indicado pela bola vermelha no plot).

9.2.3 Espécies indicadoras

Uma pergunta normalmente feita por ecólogos é: qual espécie pode ser indicadora de uma determinada condição ambiental (e.g., poluição)?

O índice IndVal mede dois aspectos das espécies: fidelidade e especificidade. Uma alta fidelidade significa que espécies ocorrem em todos os locais do grupo, e uma alta especificidade significa que as espécies ocorrem somente naquele grupo. Uma boa espécie indicadora é aquela na qual todos os indivíduos ocorrem em todas as amostras referentes a um grupo específico. A especificidade é dada pela divisão da abundância média da espécie no grupo pela somatória das abundâncias médias dos grupos. Fidelidade é igual ao número de lugares no grupo onde a espécie está presente dividido pelo número total de lugares do grupo (Dufrêne & Legendre 1997).

Espécies raras podem receber o mesmo valor de IndVal das espécies indicadoras, porém são chamadas de indicadoras assimétricas, uma vez que contribuem com a especificidade do habitat, mas não servem para predizer grupos. Ao contrário, as espécies indicadoras são verdadeiros indicadores simétricos e podem ser usadas para predizer grupos.

A análise procede da seguinte forma:

1. Uma matriz de distância é construída e as unidades amostrais são classificadas com alguma análise de agrupamento, hierárquico ou não
2. A variável ambiental para a qual se deseja classificar os grupos é inserida
3. As espécies indicadoras de cada grupo são formadas através do cálculo da especificidade e fidelidade, obtendo-se o valor de IndVal para cada espécie
4. Por fim, o conjunto de dados originais é comparado para ver se a análise faz sentido

O cálculo da significância do índice de IndVal é feito por aleatorização de Monte Carlo. Os métodos de Monte Carlo utilizam números aleatórios de dados reais para simular certos padrões esperados na ausência de um processo ecológico específico (Legendre & Legendre 2012). Assim, o valor do índice é aleatorizado 999 vezes (ou o número de vezes que você optar) dentro dos tratamentos e o valor de *P* é dado pelo número de vezes em que o índice observado foi igual ou maior que os valores aleatorizados. Portanto, o IndVal fornece um conjunto de espécies que são indicadoras de um grupo de locais (e.g., muito poluídos), que por sua vez precisam ser definidos utilizando alguma técnica de agrupamento, como as que vimos anteriormente.

Exemplo 1

Para este exemplo, vamos usar o mesmo conjunto de dados utilizado acima com abundância de 16 espécies de girinos coletados em 14 poças com diferentes graus de cobertura de dossel na Serra da Bocaina (Provete et al. 2014).

Pergunta

- Podemos utilizar as espécies de girinos como indicadoras da fitofisionomia?

Predições

- Espécies terrestres serão indicadoras de área aberta, enquanto espécies arborícolas serão indicadoras de áreas florestais

Variáveis

- Variáveis resposta: mesma matriz já utilizada contendo a abundância de girinos ao longo de poças na Serra da Bocaina

Análises

O IndVal está disponível tanto no pacote `indicspecies`, quando no `labdsv`. Para este exemplo, iremos usar o `labdsv`. Primeiro, vamos agrupar as unidades amostrais (poças) que informa os grupos de fitofisionomias onde as poças se localizam e para os quais deseja-se encontrar espécies indicadoras.

```
## Dados
head(bocaina)
#>         BP4  PP4 PP3  AP1 AP2 PP1 PP2 BP9  PT1 PT2 PT3 BP2 PT5
#> Aper      0    3   0    0   2   0   0   0    0   0   0 181   0
#> Bahe    859   14  14    0  87 312 624 641    0   0   0  14   0
#> Rict   1772 1517 207  573 796   0   0   0    0   0   0   0   0
#> Cleuco    0    0   0    0   0   0   0   0    0  29 369   0  84
#> Dmic      0    0   6   60   4   0   0   0 2758 319  25   0 329
#> Dmin      0   84 344 1045  90   0   0   0    8   0   0   0   0
fitofis <- c(rep(1, 4), rep(2, 4), rep(3, 4), rep(4, 4), rep(5, 4))

## Análise de espécies indicadoras
res_indval <- indval(t(sp_compos), fitofis)

# A função summary só exibe o resultado para as espécies indicadoras
summary(res_indval)
#>       cluster indicator_value probability
#> Rict        1          0.8364       0.011
#> Sduar       1          0.7475       0.034
#> Bahe        2          0.6487       0.048
#>
#> Sum of probabilities                =  7.984
#>
#> Sum of Indicator Values             =  7.3
#>
#> Sum of Significant Indicator Values  =  2.23
#>
#> Number of Significant Indicators     =  3
#>
#> Significant Indicator Distribution
#>
#> 1 2
#> 2 1
```

Para apresentar uma tabela dos resultados para todas as espécies temos de processar os dados.

```
## Resultados
tab_indval <- cbind.data.frame(maxcls = res_indval$maxcls,
                               ind.value = res_indval$indcls,
                               P = res_indval$pval)
tab_indval
#>         maxcls ind.value     P
#> Aper         3 0.2432796 1.000
#> Bahe         2 0.6487329 0.048
#> Rict         1 0.8363823 0.011
#> Cleuco       3 0.4128631 0.385
#> Dmic         3 0.6645244 0.195
#> Dmin         1 0.7032145 0.101
#> Hpoly        2 0.6208711 0.259
#> Lfur         3 0.2279412 1.000
#> Pbar         1 0.2813725 0.624
#> Polf         3 0.2437500 1.000
#> Pmelano      2 0.2500000 1.000
#> Sduar        1 0.7474527 0.034
#> Shay         3 0.4930269 0.416
#> Sobt         2 0.2222222 0.683
#> Ssqual       3 0.2500000 1.000
#> Bokerm       2 0.4583333 0.228

## Espécies
tab_indval[tab_indval$P < 0.05, ]
#>       maxcls ind.value     P
#> Bahe       2 0.6487329 0.048
#> Rict       1 0.8363823 0.011
#> Sduar      1 0.7474527 0.034
```

Interpretação dos resultados

No resultado apresentado, podemos ver que temos duas espécies indicadoras da fitofisionimia 1: *Rhinella icterica* (Rict) e *Scinax duartei* (Sduar). Nenhuma espécie foi indicadora dos outros grupos neste exemplo.

9.3 Análises de Ordenação

As análises de ordenação representam um conjunto de métodos e técnicas multivariadas que organizam objetos (e.g., localidades, indivíduos) em alguma ordem considerando o conjunto de descritores que podem estar mais ou menos relacionados entre si. Se os descritores estiverem bem relacionados, eles são redundantes e organizam os objetos de forma similar. Esse fenômeno é observado, por exemplo, quando queremos organizar um conjunto de unidades amostrais baseado em variáveis que indicam um mesmo processo: ver o padrão de similaridade geral de lagos usando várias variáveis relacionadas com a produtividade do sistema: clorofila-a, concentração de fósforo, nitrogênio, entre outros. Por exemplo, tais métodos permitem identificar se existem grupo de

espécies que ocorrem exclusivamente em um determinado hábitat. Ao buscar esta *ordem*, as técnicas de ordenação possuem três principais utilidades: i) reduzir a dimensionalidade e revelar padrões, ii) separar as variáveis mais e menos importantes em combinações complexas e iii) separar relações mais e menos fortes ao comparar variáveis preditoras e dependentes.

Em geral, os métodos são divididos em ordenações irrestritas (ou análise de gradiente indireto) e restritas (ou análise de gradiente direto). **As ordenações irrestritas** organizam os objetos (e.g., espécies) de acordo com sua estrutura de covariância (ou correlação), o que demonstra que a proximidade (ou distância) dentro do espaço multidimensional representa semelhança (ou diferença) dos objetos. Por outro lado, **as ordenações restritas** posicionam os objetos (e.g., espécies) de acordo com sua relação linear com outras variáveis coletadas nas mesmas unidades amostrais (e.g., temperatura e precipitação). Ao passo que as ordenações irrestritas dependem somente de uma matriz (e.g., espécies por localidades), as ordenações restritas utilizam no mínimo duas matrizes (e.g., espécies por localidades e variáveis climáticas por localidade). Desse modo, fica claro esta diferença entre os dados utilizados que as análises irrestritas são mais exploratórias, enquanto análises restritas são ideais para testar hipóteses com dados multidimensionais. A tabela a seguir apresenta as principais análises utilizadas em ecologia.

Método	Tipo de variável	Função R
Ordenação irrestrita		
PCA	Variáveis contínuas (distância euclidiana)	`PCA()`, `rda()`, `dudi.pca()`
PCoA	Aceita qualquer tipo de variável, mas depende da escolha apropriada de uma medida de distância	`pcoa()`, `dudi.pco()`
nMDS	Aceita qualquer tipo de variável, mas depende da escolha apropriada de uma medida de distância	`metaMDS()`, `nmds()`
CA		`dudi.coa()`
Hill-Smith	Aceita qualquer tipo de variável	`dudi.hillsmith()`
Ordenação restrita		
RDA	Variáveis preditoras de qualquer tipo e variáveis dependentes contínuas (ou presença e ausência)	`rda()`
RDA parcial	Variáveis preditoras de qualquer tipo e variáveis dependentes contínuas (ou presença e ausência)	`rda()`
dbRDA	Variáveis preditoras de qualquer tipo e matriz de distância obtida a partir das variáveis dependentes	`capscale()`, `dbrda()`
CCA	Variáveis preditoras de qualquer tipo e variáveis dependentes contínuas (ou presença e ausência)	`rda()`
PERMANOVA	Variáveis preditoras de qualquer tipo e matriz de distância obtida a partir das variáveis dependentes	`adonis()`, `adonis2()`
PCR	Variável dependente necessariamente representada por escores da PCA ou PCoA e variáveis preditoras de qualquer tipo	`pca()`, `pcoa()`, `lm()`, `glm()`

9.4 Ordenação irrestrita

Ordenações irrestritas, ou análise de gradiente indireto ou ainda análises de fator, são um conjunto de métodos multivariados que lidam com uma única matriz quadrada. Esta matriz pode ou não ter pesos nas linhas ou colunas. Geralmente, o objetivo deste tipo de análise é resumir a informação contida na matriz de maneira gráfica, por meio de um diagrama de ordenação. Quanto maior e mais complexa for a matriz, mais eficiente é este tipo de análise. Os tipos de análise irão diferir de acordo com o tipo de dado contido nesta matriz, se contínuo ou contagem, etc. De maneira geral, essas ordenações irrestritas calculam combinações lineares, cuja formulação irá diferir ligeiramente entre os métodos. Da mesma forma, essas combinações lineares irão preservar um tipo de distância. Por exemplo, a Análise de Componentes Principais preserva a distância Euclidiana, enquanto a Análise de Correspondência preserva a distância de chi-quadrado.

9.4.1 Análise de Componentes Principais (PCA)

A Análise de Componentes Principais (*Principal Component Analysis* - PCA) é uma das ordenações mais utilizadas em diversas áreas do conhecimento. Em Ecologia, ela se popularizou por facilitar a visualização de dados complexos como de distribuição de espécies em diferentes localidades e de potenciais variáveis explicativas. Ao mesmo tempo que ganhou tamanha popularidade, a PCA tem sido empregada de maneira incorreta, uma vez que muitos estudos utilizam a visualização gráfica da ordenação (o *biplot*) para interpretar "relações" entre variáveis preditoras (ambientais) e dependentes (espécies). Porém, como informado anteriormente, as ordenações irrestritas utilizam a estrutura de covariância dos objetos para organizar suas relações de similaridade.

Antes de explicar a análise, imagine que vamos usar uma matriz com cinco espécies de aranhas que foram encontradas em oito cidades diferentes. A quantidade de indivíduos de cada espécie coletada em cada cidade será o valor de preenchimento desta matriz. Sendo assim, a matriz possui oito objetos (cidades, representando unidades amostrais) e cinco descritores (espécies).

O primeiro passo da PCA é obter uma matriz centralizada, onde cada valor é subtraído da média da coluna que aquele valor pertence. Esta centralização pode ser calculada com a função `scale()`.

```
## Dados
aranhas <- data.frame(
    sp1 = c(5, 7, 2, 0, 0, 0, 0, 0),
    sp2 = c(0, 6, 3, 4, 0, 0, 0, 0),
    sp3 = c(0, 0, 0, 9, 12, 3, 0, 0),
    sp4 = c(0, 0, 0, 0, 4, 10, 8, 0),
    sp5 = c(0, 0, 0, 0, 0, 6, 9, 12),
    row.names = paste0("cidade", 1:8))

## Centralização
aranha.cent <- as.data.frame(base::scale(aranhas, center = TRUE,
                                scale=FALSE))
```

O segundo passo é calcular uma matriz de covariância (ou matriz de dispersão) e, a partir desta matriz, obter os autovalores e autovetores. **Os autovalores** representam a porcentagem de explicação de cada eixo e podem ser calculados dividindo a soma do autovalor de cada eixo pela soma de todos os

autovalores. No exemplo que apresentamos, os dois primeiros eixos representam 47,20% e 35,01% de toda variação, respectivamente. **Os autovetores**, por sua vez, representam os valores que multiplicam as variáveis originais e, desse modo, indicam a direção desses valores. Por fim, os componentes principais (Matriz F) são obtidos multiplicando os autovetores com os valores da matriz centralizada.

```
## Matriz de covaiância
matriz_cov <- cov(aranha.cent)

## Autovalores e autovetores
eigen_aranhas <- eigen(matriz_cov)
autovalores <- eigen_aranhas$values
autovetores <- as.data.frame(eigen_aranhas$vectors)
autovalores # eigenvalue
#> [1] 36.733031 27.243824  9.443805  2.962749  1.438020

colnames(autovetores) <- paste("PC", 1:5, sep="")
rownames(autovetores) <- colnames(aranhas)
autovetores
#>            PC1         PC2         PC3         PC4        PC5
#> sp1 -0.2144766  0.38855265  0.29239380 -0.02330706  0.8467522
#> sp2 -0.2442026  0.17463316  0.01756743  0.94587037 -0.1220204
#> sp3 -0.3558368 -0.80222917 -0.27591770  0.10991178  0.3762942
#> sp4  0.4159852 -0.41786654  0.78820962  0.17374202  0.0297183
#> sp5  0.7711688  0.01860152 -0.46560957  0.25003826  0.3544591

## Componentes principais
matriz_F <- as.data.frame(as.matrix(aranha.cent) %*%
                          as.matrix(autovetores))
matriz_F
#>               PC1        PC2        PC3        PC4        PC5
#> cidade1 -2.979363  4.4720575  1.1533417 -3.2641923  0.5433206
#> cidade2 -4.873532  6.2969618  1.8435339  2.3644158  1.5047024
#> cidade3 -3.068541  3.8302991  0.3288626 -0.3566600 -2.3629973
#> cidade4 -6.086322 -3.9922356 -2.7216169  1.6250305 -0.7918743
#> cidade5 -4.513082 -8.7689219 -0.4668012 -1.1337476  0.9439633
#> cidade6  5.812374 -3.9444494  3.9520584  0.4197281 -0.1376205
#> cidade7  8.361421 -0.6462243  1.8065636  0.4926235 -0.2625625
#> cidade8  7.347046  2.7525126 -5.8959421 -0.1471979  0.5630683

## Porcentagem de explicação de cada eixo
100 * (autovalores/sum(autovalores))
#> [1] 47.201691 35.008126 12.135225  3.807112  1.847846
```

Agora, é possível visualizar a relação entre as cidades e similaridade nas espécies de aranhas que vivem em cada uma delas (Figura 9.6).

```
## Gráfico
ggplot(matriz_F, aes(x = PC1, y = PC2, label = rownames(matriz_F))) +
    geom_label() +
```

```
  geom_hline(yintercept = 0, linetype=2) +
  geom_vline(xintercept = 0, linetype=2) +
  xlim(-10, 10) +
  tema_livro()
```

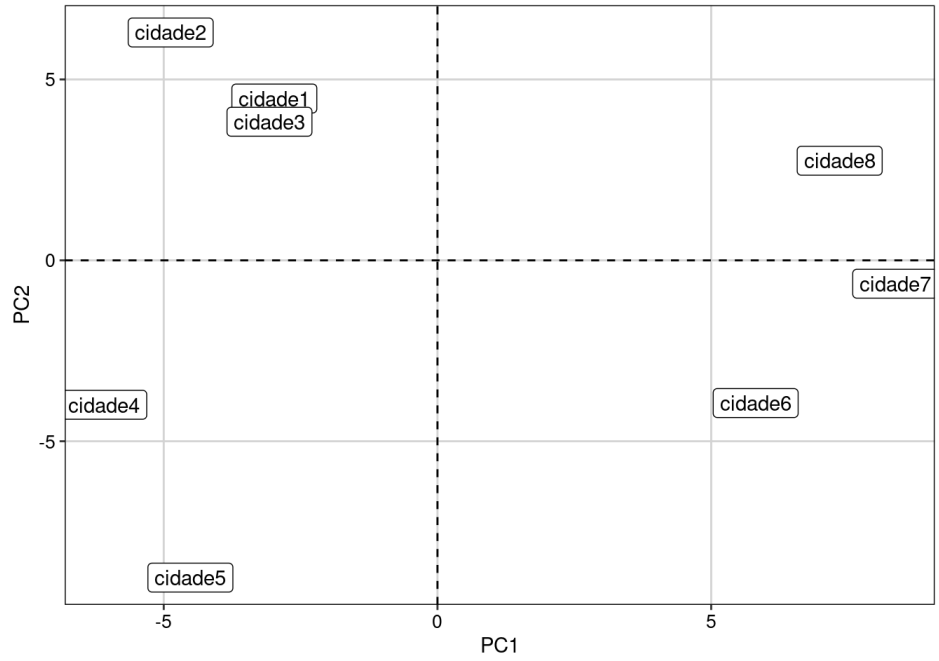


Figura 9.6: Biplot da PCA ordenando as cidades pela composição das espécies de aranhas.

Checklist

- Verifique se todas as variáveis utilizadas são contínuas. Caso contrário, considere utilizar PCoA (veja mais no próximo tópico)
- Apesar do exemplo acima ter apresentado a ocorrência de espécies de aranhas em diferentes cidades, é fundamental saber que utilizar a PCA com esses dados pode ser problemático. Assim, tenha cuidado em usar dados de composição de espécies, especialmente abundância, com PCA, uma vez que 'duplos zeros' podem gerar distorções na ordenação (Legendre & Legendre 2012). Como alternativa, é possível utilizar PCA com dados padronizados com o método de *Hellinger* (Legendre & Gallagher 2001).

Exemplo 1

Neste exemplo vamos utilizar um conjunto de dados morfológicos de pinguins do arquipélago Palmer (Península Antártica) disponíveis no pacote `palmerpenguins`. Os dados representam medidas do comprimento e largura do bico (mm), comprimento da nadadeira (mm) e massa corporal (gramas) de três espécies: Adélie, Chinstrap e Gentoo. Como descrito acima, a PCA deve ser utilizada para exploração de dados ou para testes *a posteriori* (e.g., Regressão de Componentes Principais - PCR, tópico explorado mais a frente nesse capítulo). Neste exemplo, iremos usar a estrutura de perguntas e predições para manter a proposta do livro.

Pergunta

- Existe diferenças nas características morfológicas das espécies de pinguins do arquipélago Palmer?

Predições

- Pinguins com dieta diferente possuem diferentes características morfológicas

Variáveis

- Preditora: espécie (categórica com três níveis)
- Dependentes: variáveis morfológicas (contínua)

Análises

Antes de começar, é necessário remover dados ausentes (se houver) e editar nomes das variáveis (ponto importante para determinar como devem aparecer no gráfico).

```
## Verificar se existem NAs nos dados
sum(is.na(penguins))
#> [1] 19

## Remover dados ausentes (NA), quando houver
penguins <- na.omit(penguins)

## Manter somentes dados contínuos que pretende aplicar a PCA
penguins_trait <- penguins[, 3:6]
```

Agora sim, os dados estão prontos para fazer a PCA. Um argumento é essencial na análise, o `scale.unit`. Se você utilizar dentro deste argumento a seleção `TRUE`, a função padroniza automaticamente as variáveis para terem a média 0 e variância 1. Esta padronização é essencial quando as variáveis estão em escalas muito diferentes. No exemplo selecionado, temos variáveis como comprimento do bico (em milímetros) e massa corporal (em gramas).

```
## Compare com este código a variância das variáveis
penguins_trait %>%
    dplyr::summarise(across(where(is.numeric),
                            ~var(.x, na.rm = TRUE)))
#> # A tibble: 1 × 4
#>   comprimento_bico profundidade_bico comprimento_nadadeira massa_corporal
#>              <dbl>             <dbl>                 <dbl>          <dbl>
#> 1             29.9              3.88                  196.        648372.

## Agora, veja o mesmo cálculo se fizer a padronização (scale.unit da função
PCA)
penguins_pad <- decostand(x = penguins_trait, method = "standardize")
penguins_pad %>%
    dplyr::summarise(across(where(is.numeric),
                            ~var(.x, na.rm = TRUE)))
#>   comprimento_bico profundidade_bico comprimento_nadadeira massa_corporal
#> 1                1                 1                     1              1

## PCA
pca.p <- PCA(X = penguins_trait, scale.unit = TRUE, graph = FALSE)
```

Apesar da simplicidade do código para executar a PCA, o objeto resultante da análise possui diversas informações que são essenciais para sua plena interpretação. Dentre elas, se destacam os autovalores, escores e cargas (*loadings*). Os autovalores representam a porcentagem de explicação de cada eixo. Os escores representam as coordenadas (posições no espaço multidimensional) representando os objetos (geralmente localidades ou indivíduos) e descritores (geralmente espécies ou variáveis ambientais e espaciais). Os *loadings*, por sua vez, representam a combinação linear entre os escores (nova posição do valor do descritor no espaço ordenado) e os valores originais dos descritores (Figura 9.7).

```
## Autovalores: porcentagem de explicação para usar no gráfico
pca.p$eig
#>        eigenvalue percentage of variance cumulative percentage of variance
#> comp 1  2.7453557             68.633893                          68.63389
#> comp 2  0.7781172             19.452929                          88.08682
#> comp 3  0.3686425              9.216063                          97.30289
#> comp 4  0.1078846              2.697115                         100.00000

## Visualização da porcentagem de explicação de cada eixo
# nota: é necessário ficar atento ao valor máximo do eixo 1 da análise para
determinar o valor do ylim (neste caso, colocamos que o eixo varia de 0 a
70).
fviz_screeplot(pca.p, addlabels = TRUE, ylim = c(0, 70), main = "",
               xlab = "Dimensões",
               ylab = "Porcentagem de variância explicada")
## Outros valores importantes
var_env <- get_pca_var(pca.p)

## Escores (posição) das variáveis em cada eixo
var_env$coord
#>                           Dim.1      Dim.2      Dim.3       Dim.4
#> comprimento_bico       0.7518288 0.52943763 -0.3900969 -0.04768208
#> profundidade_bico     -0.6611860 0.70230869  0.2585287  0.05252186
#> comprimento_nadadeira  0.9557480 0.00510580  0.1433474  0.25684871
#> massa_corporal         0.9107624 0.06744932  0.3592789 -0.19204478

## Contribuição (%) das variáveis para cada eixo
var_env$contrib
#>                           Dim.1        Dim.2     Dim.3     Dim.4
#> comprimento_bico      20.58919 36.023392267 41.279994  2.107420
#> profundidade_bico     15.92387 63.388588337 18.130600  2.556942
#> comprimento_nadadeira 33.27271  0.003350291  5.574092 61.149849
#> massa_corporal        30.21423  0.584669105 35.015313 34.185789

## Loadings - correlação das variáveis com os eixos
var_env$cor
#>                           Dim.1      Dim.2      Dim.3       Dim.4
#> comprimento_bico       0.7518288 0.52943763 -0.3900969 -0.04768208
#> profundidade_bico     -0.6611860 0.70230869  0.2585287  0.05252186
```

```
#> comprimento_nadadeira  0.9557480 0.00510580  0.1433474  0.25684871
#> massa_corporal         0.9107624 0.06744932  0.3592789 -0.19204478

## Qualidade da representação da variável. Esse valor é obtido multiplicado
var_env$coord por var_env$coord
var_env$cos2
#>                           Dim.1        Dim.2      Dim.3       Dim.4
#> comprimento_bico      0.5652466 2.803042e-01 0.15217561 0.002273581
#> profundidade_bico     0.4371669 4.932375e-01 0.06683710 0.002758546
#> comprimento_nadadeira 0.9134542 2.606919e-05 0.02054847 0.065971260
#> massa_corporal        0.8294881 4.549411e-03 0.12908133 0.036881196

## Escores (posição) das localidades ("site scores") em cada eixo
ind_env <- get_pca_ind(pca.p)
```

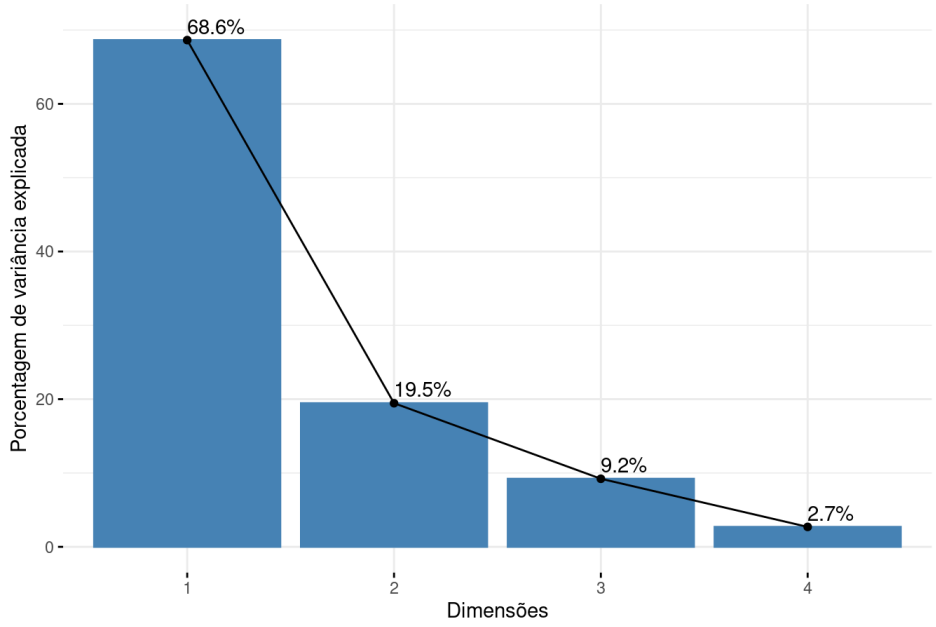


Figura 9.7: Scree plot mostrando a porcentagem de contribuição de cada eixo para a ordenação dos dados de penguins.

O pacote `FactoMineR` criou uma função (`dimdesc()`) que seleciona as melhores variáveis (aquelas mais explicativas) para cada eixo através de uma análise fatorial. No exemplo com pinguins, o primeiro eixo (objeto `pca.p$eig`) explica ~69% da variação morfológica. A função `dimdesc()` mostra que as quatro variáveis morfológicas estão fortemente associadas com o eixo 1. Porém, enquanto comprimento da nadadeira, massa corporal e comprimento do bico estão positivamente associados com o eixo 1 (correlação positiva), a largura do bico tem relação negativa. O eixo 2, por sua vez, explica ~20% da variação, sendo relacionado somente com largura e comprimento do bico.

```
## Variáveis mais importantes para o Eixo 1
dimdesc(pca.p)$Dim.1
#> $quanti
#>                       correlation       p.value
#> comprimento_nadadeira   0.9557480 5.962756e-178
```

```
#> massa_corporal          0.9107624 3.447018e-129
#> comprimento_bico        0.7518288  7.830597e-62
#> profundidade_bico      -0.6611860  3.217695e-43
#>
#> attr(,"class")
#> [1] "condes" "list"

## Variáveis mais importantes para o Eixo 2
dimdesc(pca.p)$Dim.2
#> $quanti
#>                   correlation      p.value
#> profundidade_bico   0.7023087 8.689230e-51
#> comprimento_bico    0.5294376 1.873918e-25
#>
#> attr(,"class")
#> [1] "condes" "list"
```

Agora podemos utilizar o famoso **biplot** para representar a comparação morfológica dos pinguins dentro e entre espécies (Figura 9.8).

```
fviz_pca_biplot(X = pca.p,
                geom.ind = "point",
                fill.ind = penguins$especies,
                col.ind = "black",
                alpha.ind = 0.7,
                pointshape = 21,
                pointsize = 4,
                palette = c("darkorange", "darkorchid", "cyan4"),
                col.var = "black",
                invisible = "quali",
                title = NULL) +
    labs(x = "PC1 (68.63%)", y = "PC2 (19.45%)") +
    xlim(c(-4, 5)) +
    ylim(c(-3, 3)) +
    tema_livro()
```

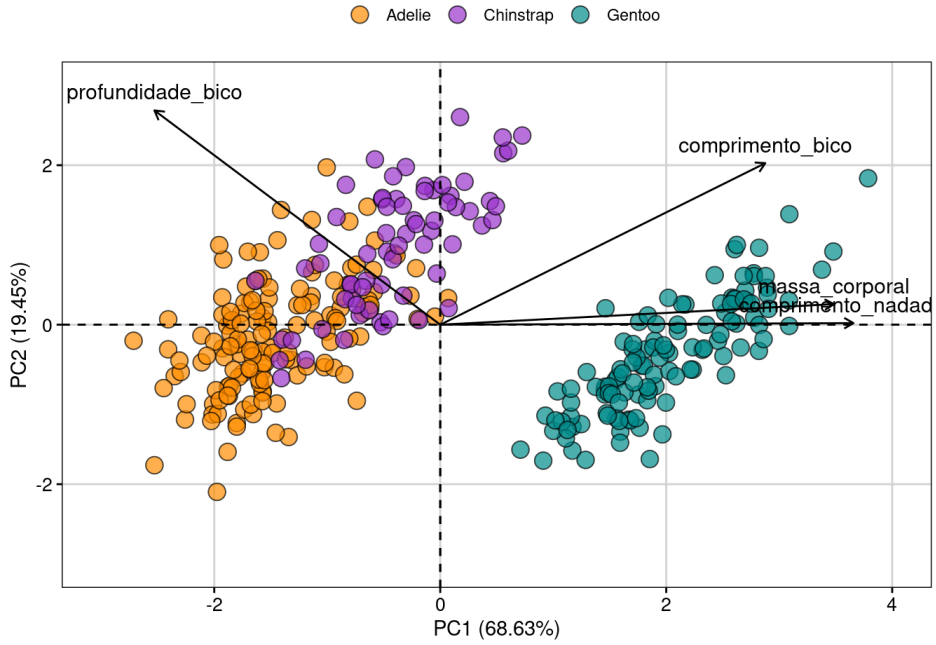


Figura 9.8: Biplot da PCA ordenando os dados morfológicos de penguins.

9.4.2 Análises de Coordenadas Principais (PCoA)

Diferentemente da PCA, a Análises de Coordenadas Principais (*Principal Coordinate Analysis* - PCoA) é uma análise de ordenação irrestrita que aceita dados de diferentes tipos, como contínuos, categóricos, ordinais, binários, entre outros. Assim, a PCoA é aplicada para casos em que a distância euclidiana não é aplicada (como na PCA). Desse modo, o primeiro passo da análise é calcular uma matriz de similaridade ou de distância (discutido acima). Depois, os passos para obter autovalores e autovetores são bastante parecidos com a PCA. Da mesma forma, os eixos da PCoA e os valores ou posições dos objetos nesses eixos representam a relação de semelhança (ou diferença) baseada nos descritores desses objetos. A diferença, neste caso, é que a PCoA representa um espaço não-euclidiano, que irá ser afetado pela escolha do método de similaridade.

As utilizações mais comuns da PCoA são a ordenação: i) da matriz de composição de espécies usando a distância apropriada (*Jaccard*, *Sorensen*, *Bray-Curtis*), ii) da matriz de variáveis ambientais com mistos (contínuos, categóricos, circulares, etc.), e iii) da matriz filogenética (método PVR Diniz-Filho et al. 1998). Abaixo, exemplificamos a ordenação da matriz de composição de espécies.

Checklist

- Compare as dimensões das matrizes utilizadas para a PCoA. Com bastante frequência, a tentativa de combinar dados categóricos (algum descritor dos objetos) com os valores obtidos com a PCoA gera erros para plotar a figura ou para executar a análise. Verifique, então, se as linhas são as mesmas (nome das localidades ou indivíduos e quantidade)
- É fundamental conhecer o tipo de dados que está usando para selecionar a medida de distância apropriada. Essa escolha vai afetar a qualidade da ordenação e sua habilidade para interpretar a relação de semelhança entre os objetos comparados
- Diferente da PCA, a PCoA aceita dados ausentes se a medida de distância escolhida também não tiver esta limitação. Por exemplo, a distância de Gower produz matrizes de similaridade mesmo com dados ausentes em determinados objetos

- Em alguns casos, autovalores negativos são produzidos na ordenação com PCoA. Veja as principais causas desses valores em Legendre & Legendre (2012). Apesar deste problema, os autovalores mais importantes (eixos iniciais) não são afetados e, deste modo, a qualidade da representação dos objetos no espaço multidimensional não é afetada. Alguns autores sugerem utilizar correções métodos de correção, como Lingoes ou Cailliez (Legendre & Legendre 2012).

Exemplo 1

Neste exemplo, vamos utilizar a composição de ácaros Oribatidae em 70 manchas de musgo coletados por Borcard et al. (1992).

Pergunta

- A composição de espécies de ácaros muda entre diferentes topografias?

Predições

- Iremos encontrar ao menos dois grupos de espécies: aquelas que ocorrem em poças dentro de floresta *versus* aquelas que ocorrem em poças de áreas abertas

Variáveis

- Preditora: topografia (categórica com dois níveis)
- Dependentes: composição de espécies de ácaro

Análises

Vamos primeiramente padronizar dos dados, calcular uma matriz de distância com método *Bray-Curtis* e depois calcular a PCoA.

```
## Padronização dos dados com Hellinger
mite.hel <- decostand(x = mite, method = "hellinger")

## Cálculo da matriz de distância com método Bray-Curtis
sps.dis <- vegdist(x = mite.hel, method = "bray")

## PCoA
pcoa.sps <- pcoa(D = sps.dis, correction = "cailliez")
```

Assim como na PCA, a porcentagem de explicação dos eixos é uma das informações mais importantes pois descrevem a efetividade da redução da dimensionalidade dos dados.

```
## Porcentagem de explicação do Eixo 1
100 * (pcoa.sps$values[, 1]/pcoa.sps$trace)[1]
#> [1] 49.10564

## Porcentagem de explicação dos Eixo 2
100 * (pcoa.sps$values[, 1]/pcoa.sps$trace)[2]
#> [1] 14.30308

## Porcentagem de explicação acumulada dos dois primeiros eixos
sum(100 * (pcoa.sps$values[, 1]/pcoa.sps$trace)[1:2])
```

```
#> [1] 63.40872

## Selecionar os dois primeiros eixos
eixos <- pcoa.sps$vectors[, 1:2]

## Juntar com algum dado categórico de interesse para fazer a figura
pcoa.dat <- data.frame(topografia = mite.env$Topo, eixos)
```

Para visualizar os resultados da PCoA, vamos exportar os escores dos eixos para usar no pacote `ggplot2` (Figura 9.9).

```
## Escores dos dois primeiros eixos
eixos <- pcoa.sps$vectors[, 1:2]

## Combinar dados dos escores com um dado categórico de interesse para nossa
pergunta
pcoa.dat <- data.frame(topografia = mite.env$Topo, eixos)

### Gráfico biplot da PCoA
ggplot(pcoa.dat, aes(x = Axis.1, y = Axis.2, fill = topografia,
                     color = topografia, shape = topografia)) +
    geom_point(size = 4, alpha = 0.7) +
    scale_shape_manual(values = c(21, 22)) +
    scale_color_manual(values = c("black", "black")) +
    scale_fill_manual(values = c("darkorange", "cyan4")) +
    labs(x = "PCO 1 (49.11%)", y = "PCO 2 (14.30%)") +
    geom_hline(yintercept = 0, linetype = 2) +
    geom_vline(xintercept = 0, linetype = 2) +
    tema_livro()
```

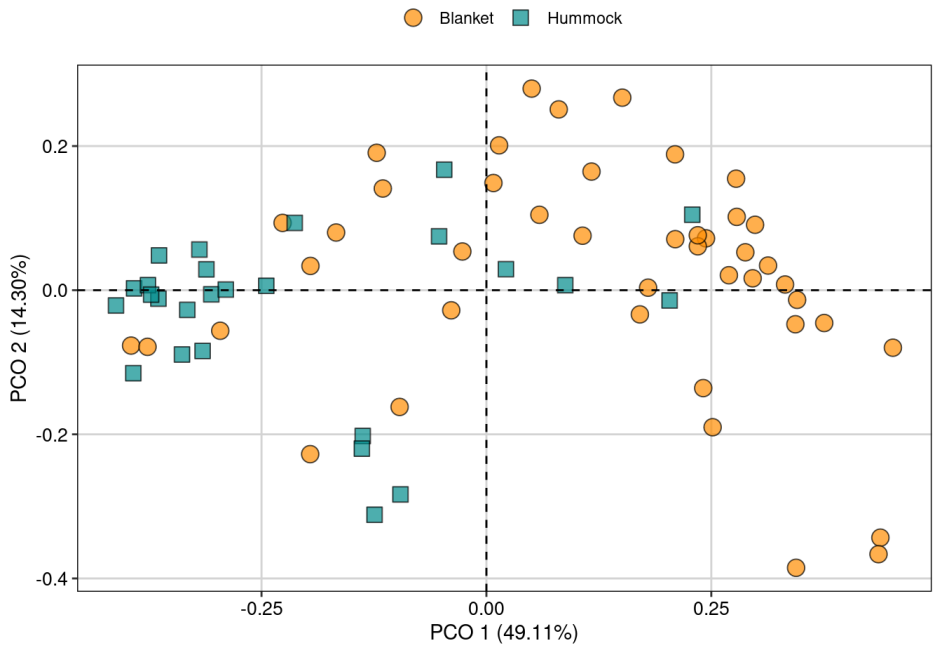


Figura 9.9: Biplot da PCoA ordenando as espécies de ácaros entre diferentes topografias.

Limitações importantes das ordenações irrestritas

Com frequência, pesquisadores utilizam análises como PCA e PCoA para "testar" diferenças na composição de espécies entre determinados fatores relevantes (altitude, clima, etc.). Porém, como falado acima, as análises de ordenação irrestritas não são utilizadas para testar qualquer hipótese. Ao invés disso, essas análises representam uma poderosa ferramenta para explorar padrões em variáveis dependentes ou independentes para ajudar na interpretação ou mesmo para testar hipóteses em análises combinadas com as ordenações restritas.

9.4.3 Regressão de Componentes Principais (PCR)

Uma maneira de testar hipóteses utilizando ordenações irrestritas é utilizando os resultados da ordenação (escores) como variáveis preditoras ou dependentes como, por exemplo, em modelos lineares (e.g., regressão múltipla Capítulo 7). O primeiro passo é utilizar uma ordenação, como a PCA, para gerar os "novos" dados que serão usados na análise. A utilização desses novos dados (que representam as coordenadas principais ou escores da PCA) vai depender da pergunta em questão. Por exemplo, pode ser que esses valores representem gradientes climáticos e, por este motivo, serão utilizados como variáveis preditoras em um modelo linear (e.g., regressão múltipla). Por outro lado, esses valores podem representar o espaço morfológicos de espécies de peixes e, como consequência, serão utilizados como variáveis dependentes para entender o efeito da presença de predador sobre a morfologia. É importante ressaltar que existem algumas limitações importantes na PCR como, por exemplo, as coordenadas principais (escores da PCA) utilizadas como variáveis preditoras podem não representar, biologicamente, as mais importantes para explicar a variação na variável resposta (Hadi & Ling 1998).

Checklist

- Compare as dimensões das matrizes utilizadas para a PCR. Com bastante frequência, a tentativa de combinar dados categóricos (algum descritor dos objetos) com os valores obtidos com a PCoA gera erros para plotar a figura ou para executar a análise. Verifique, então, se as linhas são as mesmas (nome das localidades ou indivíduos e quantidade)
- Estudos recentes têm criticado a utilização de PCR para testar hipóteses ecológicas pelo fato dos escores não representarem, necessariamente, a variação total das variáveis originais, bem como a relação entre a variável preditora e a dependente

Exemplo 1

Neste exemplo, vamos utilizar a composição de espécies de aves em 23 regiões dos alpes franceses. Os dados ambientais (env) representam variáveis climáticas (temperatura e chuva) e altitude.

Pergunta

- Gradientes climáticos afetam a riqueza de aves?

Predições

- O aumento da umidade e redução da temperatura aumentam o número de espécies de aves

Variáveis

- Preditora: temperatura e chuva (contínuas) e altitude (categórica com três níveis)
- Dependentes: riqueza de espécies de aves

```
## Dados
env_cont <- env[,-8]
env.pca <- PCA(env_cont, scale.unit = TRUE, graph = FALSE)
var_env <- get_pca_var(env.pca)

## Contribuição (%) das variáveis para cada eixo
var_env$contrib
#>             Dim.1      Dim.2      Dim.3      Dim.4       Dim.5
#> mini.jan 10.93489 22.2975487 16.1607726  7.6025527  0.01782438
#> maxi.jan 20.18065  3.2890767  2.1814486  4.2756350 41.05646526
#> mini.jul 11.87396 21.1379132  0.3428843  0.7750666 44.70209396
#> maxi.jul 18.47244  0.9159957 56.5369988  9.4368661  2.59283074
#> rain.jan  9.95206 21.5387403  6.5737927 53.7375738  4.44283706
#> rain.jul 16.14997 11.2368132  7.2608047 19.6972097  0.71454880
#> rain.tot 12.43603 19.5839121 10.9432983  4.4750959  6.47339980

## Loadings - correlação das variáveis com os eixos
var_env$cor
#>               Dim.1     Dim.2       Dim.3       Dim.4        Dim.5
#> mini.jan  0.6830371 0.6766524 -0.21924927  0.12298817 -0.004517369
#> maxi.jan  0.9279073 0.2598807 -0.08055260  0.09223249  0.216804944
#> mini.jul  0.7117620 0.6588220  0.03193603 -0.03926930 -0.226225907
#> maxi.jul  0.8877675 0.1371462  0.41008461 -0.13702428  0.054483561
#> rain.jan -0.6516187 0.6650391 -0.13983474 -0.32698110  0.071319550
#> rain.jul -0.8300858 0.4803509  0.14696011  0.19796389 -0.028601865
#> rain.tot -0.7284135 0.6341424  0.18041856  0.09435932  0.086088397

ind_env <- get_pca_ind(env.pca)
env.pca$eig
#>        eigenvalue percentage of variance cumulative percentage of variance
#> comp 1 4.26652359             60.9503370                           60.95034
#> comp 2 2.05340251             29.3343216                           90.28466
#> comp 3 0.29745014              4.2492878                           94.53395
#> comp 4 0.19896067              2.8422953                           97.37624
#> comp 5 0.11448717              1.6355310                           99.01177
#> comp 6 0.04312874              0.6161248                           99.62790
#> comp 7 0.02604718              0.3721025                          100.00000
```

O objeto `env.pca$eig` demonstra que os três primeiros eixos explicam 94.54% da variação total dos dados climáticos. O intuito da PCR é reduzir a dimensionalidade, ou seja, o número de variáveis preditoras ou dependentes para facilitar a interpretação e garantir que as variáveis não sejam correlacionadas. O próximo passo então é obter os valores dos escores que representam os valores convertidos para serem usados em uma determinada análise, como a regressão múltipla.

```
## Passo 1: obter os primeiros eixos
pred.env <- ind_env$coord[, 1:3]

## Passo 2: calcular a riqueza de espécies
```

```
riqueza <- specnumber(species)

## Passo 3: combinar os dois valores em um único data.frame
dat <- data.frame(pred.env, riqueza)
```

Agora que os dados foram combinados em um único data frame, podemos utilizar os códigos apresentados no Capítulo 7 para testar nossa hipótese (Figura 9.10).

```
## Regressão múltipla
mod1 <- lm(riqueza ~ Dim.1 + Dim.2 + Dim.3, data = dat)
par(mfrow = c(2, 2))
plot(mod1) # verificar pressupostos dos modelos lineares
summary(mod1) # resultados do  teste
#>
#> Call:
#> lm(formula = riqueza ~ Dim.1 + Dim.2 + Dim.3, data = dat)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -3.4008 -1.1729  0.4356  1.2072  2.4571
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 13.30435    0.37639  35.347  < 2e-16 ***
#> Dim.1        0.68591    0.18222   3.764  0.00131 **
#> Dim.2       -0.09961    0.26267  -0.379  0.70874
#> Dim.3       -0.21708    0.69014  -0.315  0.75654
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.805 on 19 degrees of freedom
#> Multiple R-squared:  0.4313, Adjusted R-squared:  0.3415
#> F-statistic: 4.804 on 3 and 19 DF,  p-value: 0.01179

dimdesc(env.pca)$Dim.1
#> $quanti
#>          correlation      p.value
#> maxi.jan   0.9279073 1.846790e-10
#> maxi.jul   0.8877675 1.607390e-08
#> mini.jul   0.7117620 1.396338e-04
#> mini.jan   0.6830371 3.282701e-04
#> rain.jan  -0.6516187 7.559358e-04
#> rain.tot  -0.7284135 8.112903e-05
#> rain.jul  -0.8300858 9.588034e-07
#>
#> attr(,"class")
#> [1] "condes" "list"
```

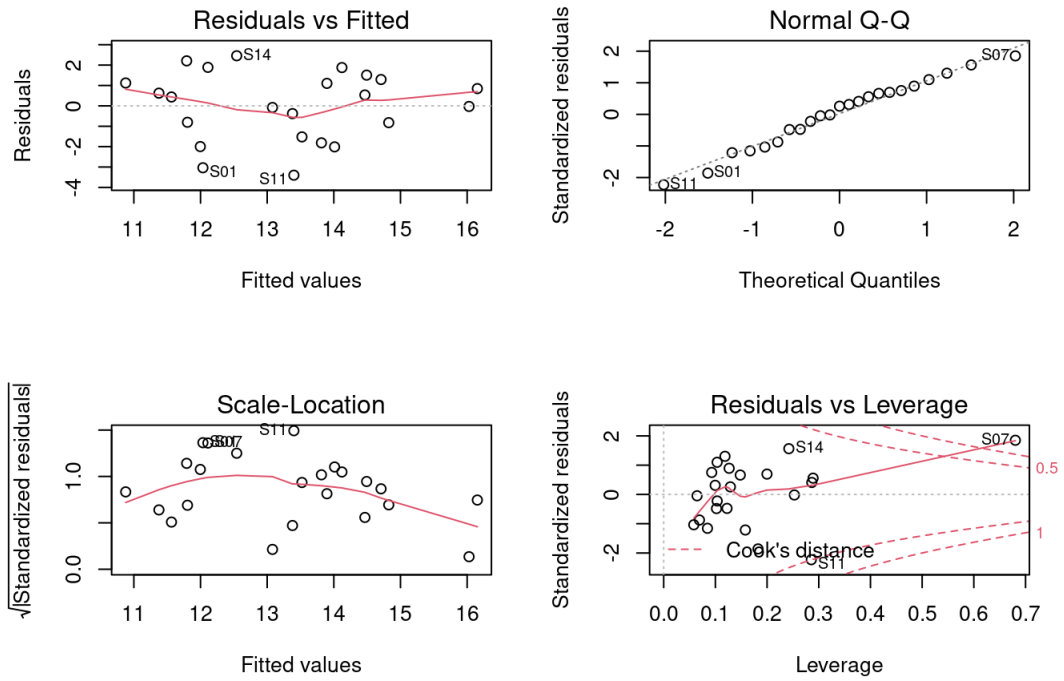


Figura 9.10: Diagnósticos do modelo PCR para aves.

Como percebemos, a Dim.1 foi o único gradiente ambiental que afetou a riqueza de espécies. Para interpretar esta dimensão (e outras importantes), podemos usar a função `dimdesc()` para verificar as variáveis mais importantes. Neste caso, os valores mais extremos de correlação (maior que 0.8) indicam que a temperatura do mês de janeiro e julho bem como a chuva do mês de julho foram as variáveis mais importantes para determinar o gradiente ambiental expresso na dimensão 1. Assim, podemos fazer um gráfico para representar a relação entre Eixo 1 (gradiente chuva-temperatura) e a riqueza de espécies de aves. Valores negativos do eixo 1 (Gradiente ambiental - PC1) representam localidades com mais chuva, ao passo que valores positivos indicam localidades com temperaturas maiores (Figura 9.11).

```
ggplot(dat, aes(x = Dim.1, y = riqueza)) +
    geom_smooth(method = lm, fill = "#525252", color = "black") +
    geom_point(size = 4, shape = 21, alpha = 0.7, color = "#1a1a1a",
         fill = "cyan4") +
    labs(x = "Gradiente ambiental (PC1)", y = "Riqueza de aves") +
    tema_livro()
```

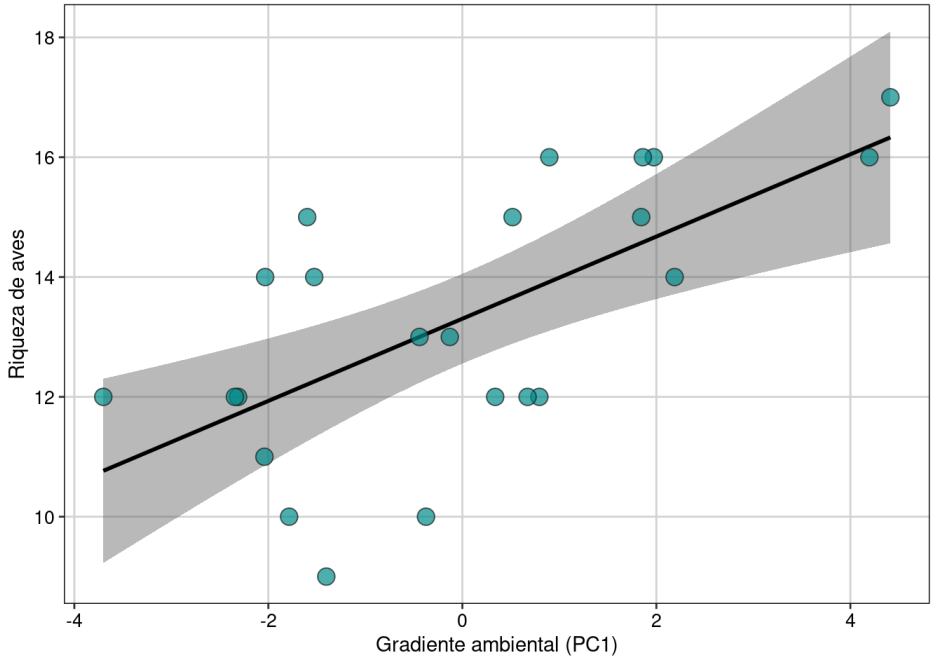


Figura 9.11: Modelo linear representando a relação entre Eixo 1 (gradiente chuva-temperatura) e a riqueza de espécies de aves.

Exemplo 2

É possível que os dados utilizados em seu estudo sejam mistos, ou seja, incluem tanto variáveis categóricas quanto contínuas. Como falado acima, nesses casos a análise indicada é a PCoA. Assim como na PCA, podemos extrair os escores da PCoA para utilizar em análises univariadas e multivariadas.

Pergunta

- Variáveis climáticas, vegetacionais e topográficas afetam a riqueza de ácaros?

Predições

- A densidade da vegetação e disponibilidade de água aumentam a riqueza de espécies de ácaros

Variáveis

- Preditoras: densidade de substrato e disponibilidade de água (contínuas), tipo de substrato (categórica com 7 níveis), densidade arbusto (ordinal com 3 níveis), e topografia (categórica com 2 níveis)
- Dependentes: riqueza de espécies de ácaros

O primeiro passo então é utilizar um método de distância apropriado para o seu conjunto de dados. Em nosso exemplo, utilizaremos a distância de Gower, que é usada para dados mistos (veja Capítulo 14).

```
## Matriz de distância
env.dist <- gowdis(mite.env)

## PCoA
env.mite.pco <- pcoa(env.dist, correction = "cailliez")
```

```
## Porcentagem de explicação do Eixo 1
100 * (env.mite.pco$values[, 1]/env.mite.pco$trace)[1]
#> [1] 61.49635

## Porcentagem de explicação dos Eixo 2
100 * (env.mite.pco$values[, 1]/env.mite.pco$trace)[2]
#> [1] 32.15486
```

O próximo passo é exportar os escores para as análises *a posteriori* (Figura 9.12).

```
## Selecionar os dois primeiros eixos
pred.scores.mite <- env.mite.pco$vectors[, 1:2]

## Juntar com os dados da área para fazer a figura
mite.riqueza <- specnumber(mite)
pred.vars <- data.frame(riqueza = mite.riqueza, pred.scores.mite)

### Regressão múltipla
mod.mite <- lm(riqueza ~ Axis.1 + Axis.2, data = pred.vars)
par(mfrow = c(2, 2))
plot(mod.mite)
summary(mod.mite)
#>
#> Call:
#> lm(formula = riqueza ~ Axis.1 + Axis.2, data = pred.vars)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -10.6874  -2.3960  -0.1378   2.5032   8.6873
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)  15.1143     0.4523  33.415  < 2e-16 ***
#> Axis.1      -11.4303     2.0013  -5.711  2.8e-07 ***
#> Axis.2        5.6832     2.7677   2.053   0.0439 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.784 on 67 degrees of freedom
#> Multiple R-squared:  0.3548, Adjusted R-squared:  0.3355
#> F-statistic: 18.42 on 2 and 67 DF,  p-value: 4.225e-07
```

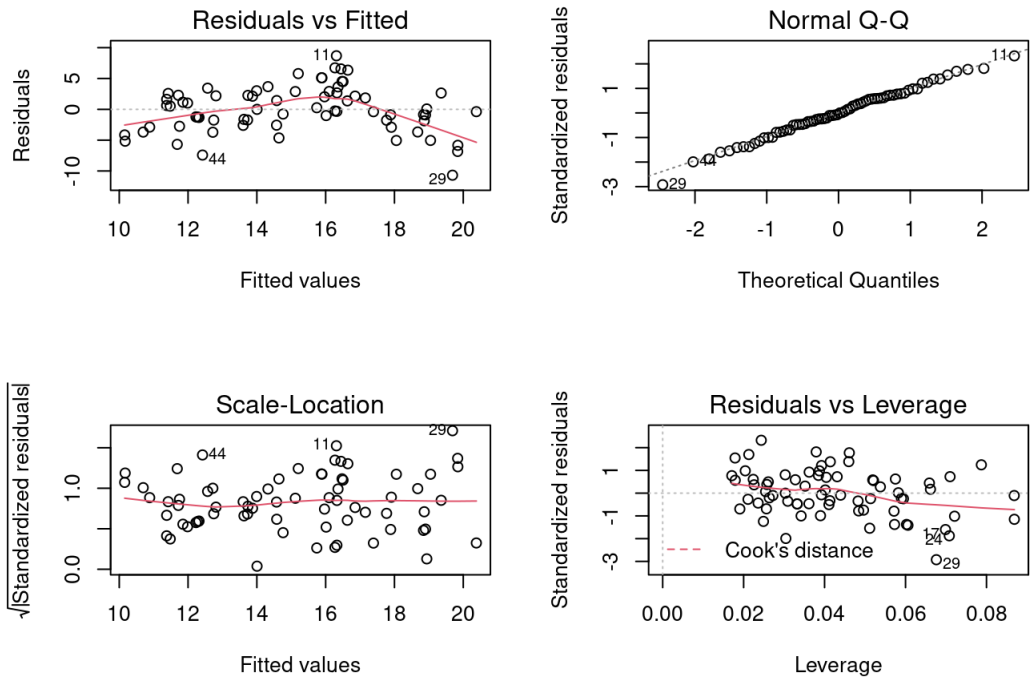


Figura 9.12: Diagnósticos do modelo PCR para ácaros.

Finalmente, após interpretar os resultados do modelo, podemos fazer a figura com as variáveis (eixos) importantes (Figura 9.13).

```
g_acari_axi1 <- ggplot(pred.vars, aes(x = Axis.1, y = riqueza)) +
    geom_smooth(method = lm, fill = "#525252", color = "black") +
    geom_point(size = 4, shape = 21, alpha = 0.7, color = "#1a1a1a",
               fill="cyan4") +
    labs(x = "Gradiente ambiental (PC1)", y = "Riqueza de ácaros") +
    tema_livro()

g_acari_axi2 <- ggplot(pred.vars, aes(x = Axis.2, y = riqueza)) +
    geom_smooth(method = lm, fill = "#525252", color = "black") +
    geom_point(size = 4, shape = 21, alpha = 0.7, color = "#1a1a1a",
               fill = "darkorange") +
    labs(x = "Gradiente ambiental (PC2)", y = "Riqueza de ácaros") +
    tema_livro()

## Função para combinar os dois plots em uma única janela
grid.arrange(g_acari_axi1, g_acari_axi2, nrow = 1)
```

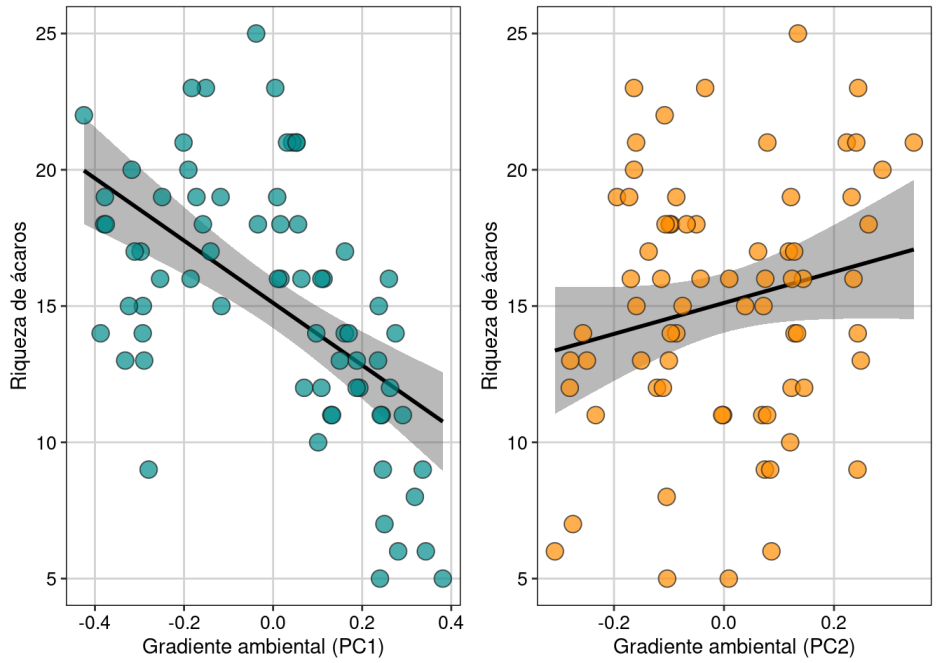


Figura 9.13: Modelo linear representando a relação entre Eixo 1 e Eixo 2 e a riqueza de espécies de ácaros.

9.5 Ordenação restrita

A ordenação restrita ou análise de gradiente direto, organiza os objetos de acordo com suas relações com outras variáveis (preditoras) coletadas nas mesmas unidades amostrais. O exemplo mais comum na ecologia é de investigar a relação entre diversas variáveis ambientais (matriz **X**) coletadas em *n* localidades e a abundância (ou presença ausência) de *y* espécies coletadas nas mesmas localidades (matrix **Y**). Com frequência, outros dados são utilizados como as coordenadas geográficas das unidades amostrais (matriz **W**), os atributos funcionais das espécies coletadas (matriz **T**) e a relação filogenética dessas espécies (matriz **P**). Diversos métodos são utilizados para combinar duas ou mais matrizes, mas neste capítulo iremos apresentar a Análise de Redundância (RDA), Análise de Redundância parcial (RDAp) e métodos espaciais para incluir a matriz **W** nas análises de gradiente direto.

9.5.1 Análise de Redundância (RDA)

A RDA é uma análise semelhante à regressão múltipla (Capítulo 7), mas que usa dados multivariados como variável dependente. As duas matrizes comuns, matriz X (*n* unidades amostrais e *m* variáveis) e matriz Y (*n* unidades amostrais e *p* descritores - geralmente, espécies). O primeiro passo da RDA é centralizar (assim como na PCA, exemplo acima) as matrizes X e Y. Após a centralização, realiza-se regressões lineares entre X e Y para obter os valores preditos de Y (ou seja, os valores de Y que representação uma combinação linear com X). O passo seguinte é realizar uma PCA dos valores preditos de Y. Este último procedimento gera os autovalores, autovetores e os eixos canônicos que correspondem às coordenadas dos objetos (unidades amostrais), variáveis preditoras e das variáveis resposta. A diferença da ordenação do valor de Y predito e da ordenação somente de Y (como na PCA implementada acima) é que a segunda mostra a posição prevista pela relação linear entre X e Y. Logo, esse é exatamente o motivo da ordenação ser conhecida como restrita, pois a variação em Y

é restrita (linearmente) pela variação de X. Assim como na regressão múltipla, a estatística da RDA é representada pelo valor de R^2 e F. O valor de R^2 indica a força da relação linear entre X e Y e o valor do F representa o teste global de significância. Além disso, é possível testar a significância de cada um dos eixos da ordenação (e a presença de pelo menos um eixo significativo é pré-requisito para que exista a relação linear entre X e Y) e de cada uma das variáveis preditoras da matriz X.

Checklist

- Variáveis preditoras: importante verificar se: i) a estrutura de correlação das variáveis ambientais, e a ii) presença de autocorrelação espacial
- Composição de espécies como matriz Y: fundamental observar se os valores utilizados representam abundância ou presença-ausência e qual a necessidade de padronização (e.g., *Hellinger*)
- Assim como em modelos de regressão linear simples e múltipla, os valores de R^2 ajustado devem ser selecionados ao invés do valor de R^2

Exemplo 1

Espécies de aves que ocorrem em localidades com diferentes altitudes.

Pergunta

- O clima e a altitude modificam a composição de espécies de aves?

Predições

- Diferenças climáticas (temperatura e chuva) e altitudinais alteram a composição de espécies de aves

Variáveis

- Preditoras: Temperatura e precipitação (contínuas) e altitude (categórica com três níveis)
- Dependente: composição de espécies de aves

Análises

```
## Passo 1: transformação de hellinger da matriz de espécies
# caso tenha dados de abundância.
species.hel <- decostand(x = species, method = "hellinger")

## Passo 2: selecionar variáveis importantes
# Para isso, é necessário remover a variável categórica.
env.contin <- env[, -8]

## Evite usar variáveis muito correlacionadas
sel.vars <- forward.sel(species.hel, env.contin)
#> Testing variable 1
#> Testing variable 2
#> Testing variable 3
#> Procedure stopped (alpha criteria): pvalue for variable 3 is 0.212000 (>
0.050000)
sel.vars$variables
#> [1] "rain.jul" "maxi.jul"
```

```
env.sel <- env[,sel.vars$variables]

## Passo 3: padronizar matriz ambiental (somente variáveis contínuas)
env.pad <- decostand(x = env.sel, method = "standardize")

## Matriz final com variáveis preditoras
env.pad.cat <- data.frame(env.pad, altitude = env$altitude)
```

Depois de selecionar um subconjunto dos dados com o método *Forward Selection* e padronizá-los (média 0 e desvio padrão 1), o modelo da RDA é construído como modelos lineares (Figura 9.14). A Ordenação Multi-Escala (MSO) também é ajustada para analisar a autocorrelação espacial.

```
## RDA com dados selecionados e padronizados
rda.bird <- rda(species.hel ~ rain.jul + maxi.jul + altitude,
                data = env.pad.cat)

## Para interpretar, é necessário saber a significância dos eixos para
representar a relação entre as variáveis preditoras e a composição de
espécies
res.axis <- anova.cca(rda.bird, by = "axis")
res.axis
#> Permutation test for rda under reduced model
#> Forward tests for axes
#> Permutation: free
#> Number of permutations: 999
#>
#> Model: rda(formula = species.hel ~ rain.jul + maxi.jul + altitude, data =
env.pad.cat)
#>          Df Variance       F Pr(>F)
#> RDA1      1 0.045759 12.0225  0.001 ***
#> RDA2      1 0.009992  2.6252  0.061 .
#> RDA3      1 0.007518  1.9752  0.136
#> RDA4      1 0.003582  0.9410  0.470
#> Residual 18 0.068510
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

## Em seguida, é possível identificar quais são as variáveis que contribuem
ou que mais contribuem para a variação na composição de espécies
res.var <- anova.cca(rda.bird, by = "term") ## Qual variável?
res.var
#> Permutation test for rda under reduced model
#> Terms added sequentially (first to last)
#> Permutation: free
#> Number of permutations: 999
#>
#> Model: rda(formula = species.hel ~ rain.jul + maxi.jul + altitude, data =
env.pad.cat)
```

```
#>          Df Variance      F Pr(>F)
#> rain.jul  1 0.036514 9.5936  0.001 ***
#> maxi.jul  1 0.011264 2.9596  0.016 *
#> altitude  2 0.019071 2.5053  0.011 *
#> Residual 18 0.068510
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

## Além disso, é possível obter o valor do R2 do modelo
r_quadr <- RsquareAdj(rda.bird)
r_quadr
#> $r.squared
#> [1] 0.4938685
#>
#> $adj.r.squared
#> [1] 0.3813949

## Ordenação multi-escala (MSO) para entender os resultados da ordenação em
relação à distância geográfica
bird.rda <- mso(rda.bird, xy, grain = 1, permutations = 99)
msoplot(bird.rda)
```

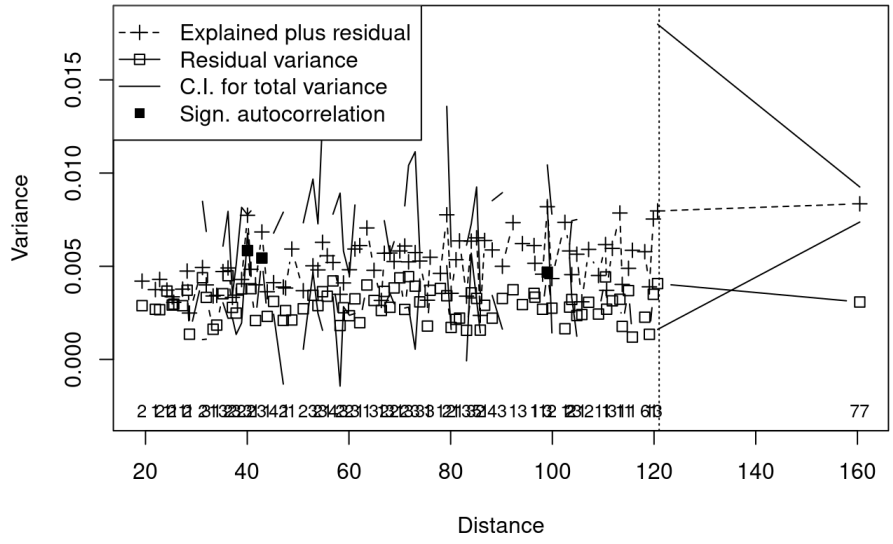


Figura 9.14: Ordenação multi-escala (MSO) para entender os resultados da ordenação em relação à distância geográfica.

Da mesma forma que nas ordenações irrestritas, podemos fazer um gráfico para visualizar as ordenações, mas agora esse gráfico é denominado "triplot," pois possui os dados das duas ordenações (Figura 9.15).

```
## Triplot da RDA
ggord(rda.bird, ptslab = TRUE, size = 1, addsize = 3, repel = TRUE) +
    geom_hline(yintercept = 0, linetype = 2) +
```

```
    geom_vline(xintercept = 0, linetype = 2) +
    tema_livro()
```

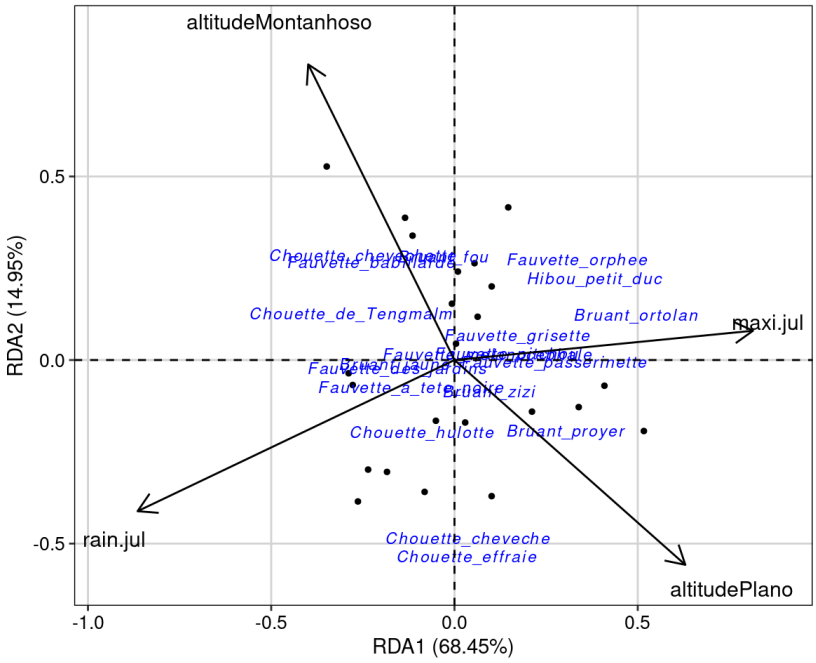


Figura 9.15: Triplot da RDA.

Interpretação dos resultados

Os objetos `res.axis`, `res.var` e `r_quadr` mostram, respectivamente, i) as dimensões (RDA1, RDA2, etc.) que possuem variação na composição de espécies, ii) as variáveis preditoras que explicam esta variação, e iii) o valor do R^2 ajustado. Neste exemplo, podemos observar que somente a dimensão 1 (RDA1) representa uma variação significativa da composição de espécies (**P** = 0,001). As variáveis *rain.jul*, *maxi.jul* e *altitude* foram todas preditoras importantes da composição de espécies, mas *rain.jul* se destacada com maior valor de **F**. Além disso, o valor do R^2 ajustado de 0.381 indica forte contribuição dessas variáveis preditoras. Porém, uma das limitações desta análise é não considerar que tanto espécies quanto variáveis preditoras podem estar estruturadas espacialmente. Como resultado, os resíduos das análises podem apresentar autocorrelação espacial que, por sua vez, aumenta o Erro do Tipo I (Legendre & Legendre 2012). A figura obtida com o código `msoplot(bird.rda)` demonstra que existe autocorrelação espacial em algumas distâncias da análise. Veja abaixo algumas alternativas para resíduos com autocorrelação espacial.

9.5.2 Análise de Redundância parcial (RDAp)

Um dos problemas da abordagem anterior é que tanto a composição de espécies como as variáveis ambientais estão estruturadas espacialmente. Talvez mais importantes, para que os valores de probabilidade da RDA sejam interpretados corretamente (e para evitar Erro do Tipo I), os resíduos do modelo não devem estar correlacionados espacialmente, como demonstrado com a análise MSO. Uma alternativa é incluir a matriz de dados espaciais (matrix W) como valor condicional dentro da RDA. Esta análise é conhecida como RDA parcial.

Porém, a obtenção dos dados espaciais da matriz W é mais complexo do que simplesmente incluir dados de localização geográfica (latitude e longitude), como feito em alguns modelos lineares (e.g., *Generalized Least Squares* - GLS Capítulos 7 e 10). Existem diversas ferramentas que descrevem e incorporam o componente espacial em métodos multidimensionais, mas os Mapas de Autovetores de Moran (MEM) são certamente os mais utilizados (Dray et al. 2012). A análise MEM consiste na ordenação (PCoA) de uma matriz truncada obtida através da localização geográfica das localidades utilizando distância euclidiana, matriz de conectividade e matriz espacial ponderada. Os autovalores obtidos no MEM são idênticos aos coeficientes de correlação espacial de Moran *I*. Um procedimento chave desta análise é a definição de um limiar de trucamento (do inglês *truncate threshold*). Este limiar é calculado a partir de uma "árvore de espaço mínimo" (*Minimum Spanning Tree* - MST) que conecta todos os pontos de coleta. Na prática, pontos com valores menores do que o limiar definido pela MST indicam que eles estão conectados e, assim possuem correlação positiva. Outro ponto importante desta análise é a obtenção da matriz espacial ponderada (*Spatial Weighting Matrix* - SWM). A seleção da matriz SWM é parte essencial do cálculo dos MEM e não deve ser feita arbitrariamente (Bauman, Drouet, Fortin, et al. 2018). Por este motivo, a análise recebe este nome (Legendre & Legendre 2012). Finalmente, o método produz autovetores que representam preditores espaciais que podem ser utilizados na RDA parcial (e em outras análises). É importante ressaltar que o critério de seleção do número de autovetores é bastante debatido na literatura e, para isso, sugerimos a leitura dos seguintes artigos: Diniz-Filho et al. (2012), Bauman, Drouet, Dray, et al. (2018) e Bauman, Drouet, Fortin, et al. (2018).

Então, o primeiro passo para realizar uma RDA parcial é de gerar os autovetores espaciais (MEMs).

```
## Dados
# Matriz padronizada de composição de espécies.
head(species.hel)[, 1:6]

## Latitude e longitude
head(xy)
#>       x   y
#> S01 156 252
#> S02 141 217
#> S03 171 233
#> S04 178 215
#> S05 123 189
#> S06 154 195

## dados ambientais padronizados e altitude
head(env.pad.cat)
#>     rain.jul   maxi.jul      altitude
#> S01 1.333646  0.1462557    Montanhoso
#> S02 1.468827 -0.6848206 Intermediário
#> S03 1.505694 -0.2099199    Montanhoso
#> S04 1.296778 -2.0699476    Montanhoso
#> S05 1.075572 -0.3682201         Plano
#> S06 1.100151 -0.6056705 Intermediário

## Passo 1: Gerar um arquivo LIST W: list binária de vizinhança
mat_knn <- knearneigh(as.matrix(xy), k = 2, longlat = FALSE)
```

```
mat_nb <- knn2nb(mat_knn, sym = TRUE)
mat_listw <- nb2listw(mat_nb, style = "W")
mat_listw
#> Characteristics of weights list object:
#> Neighbour list object:
#> Number of regions: 23
#> Number of nonzero links: 58
#> Percentage nonzero weights: 10.96408
#> Average number of links: 2.521739
#>
#> Weights style: W
#> Weights constants summary:
#>    n  nn S0       S1       S2
#> W 23 529 23 18.84444 96.01111

## Passo 2: Listar os métodos "candidatos" para obter a matriz SWM
MEM_mat <- scores.listw(mat_listw, MEM.autocor = "positive")
candidates <- listw.candidates(xy, nb = c("gab", "mst", "dnear"),
                               weights = c("binary", "flin"))

## Passo 3: Selecionar a melhor matriz SWM e executar o MEM
W_sel_mat <- listw.select(species.hel, candidates,
                    MEM.autocor = "positive",
                          p.adjust = TRUE, method = "FWD")
#> Procedure stopped (alpha criteria): pvalue for variable 5 is 0.112000 (> 0.050000)
#> Procedure stopped (alpha criteria): pvalue for variable 3 is 0.059000 (> 0.050000)
#> Procedure stopped (alpha criteria): pvalue for variable 3 is 0.060000 (> 0.050000)
#> Procedure stopped (alpha criteria): pvalue for variable 4 is 0.157000 (> 0.050000)

## Passo 4: Matriz dos preditores espaciais escolhidos (MEMs)
spatial.pred <- as.data.frame(W_sel_mat$best$MEM.select)

## É necessário atribuir os nomes das linhas
rownames(spatial.pred) <- rownames(xy)
```

Depois de gerar os valores dos autovetores espaciais (MEM), é possível executar a RDA parcial utilizando esses valores no argumento `Conditional`.

```
## Combinar variáveis ambientais e espaciais em um único data.frame
pred.vars <- data.frame(env.pad.cat, spatial.pred)

## RDA parcial
rda.p <- rda(species.hel ~
                rain.jul + maxi.jul + altitude + # Var. Ambientais
                Condition(MEM1 + MEM2 + MEM4 + MEM5), # Var. Espaciais
                data = pred.vars)

## Interpretação
```

```
# Para interpretar, é necessário saber a significância dos eixos para
representar a relação entre as variáveis preditoras e a composição de
espécies.
res.p.axis <- anova.cca(rda.p, by = "axis")
res.p.axis
#> Permutation test for rda under reduced model
#> Forward tests for axes
#> Permutation: free
#> Number of permutations: 999
#>
#> Model: rda(formula = species.hel ~ rain.jul + maxi.jul + altitude +
Condition(MEM1 + MEM2 + MEM4 + MEM5), data = pred.vars)
#>          Df Variance      F Pr(>F)
#> RDA1      1 0.008471 2.1376  0.316
#> RDA2      1 0.004830 1.2189  0.779
#> RDA3      1 0.003240 0.8176  0.890
#> RDA4      1 0.001891 0.4773  0.895
#> Residual 14 0.055477

## Contribuição
# Em seguida, é possível identificar quais são as variáveis que contribuem ou
que mais contribuem para a variação na composição de espécies.
res.p.var <- anova.cca(rda.p, by = "term") ## Qual variável?
res.p.var
#> Permutation test for rda under reduced model
#> Terms added sequentially (first to last)
#> Permutation: free
#> Number of permutations: 999
#>
#> Model: rda(formula = species.hel ~ rain.jul + maxi.jul + altitude +
Condition(MEM1 + MEM2 + MEM4 + MEM5), data = pred.vars)
#>          Df Variance      F Pr(>F)
#> rain.jul  1 0.004406 1.1119  0.349
#> maxi.jul  1 0.004446 1.1220  0.338
#> altitude  2 0.009579 1.2087  0.241
#> Residual 14 0.055477
RsquareAdj(rda.p)
#> $r.squared
#> [1] 0.1361661
#>
#> $adj.r.squared
#> [1] 0.02330319
```

Se você comparar os resultados do objeto res.p.var (RDA parcial) com res.var (RDA simples) é possível perceber como a estrutura espacial nos resíduos aumenta a probabilidade de cometer Erro do Tipo I. O modelo da RDA parcial mostra que não existem qualquer efeito direto das variáveis ambientais sobre a composição de espécies (conclusão com a RDA simples). Na verdade, tanto a composição de espécies

quanto as variáveis climáticas estão fortemente estruturadas no espaço, como demonstramos a seguir.

```
## Padrão espacial na composição de espécies
pca.comp <- dudi.pca(species.hel, scale = FALSE, scannf = FALSE)
moran.comp <- moran.mc(pca.comp$li[, 1], mat_listw, 999)

## Padrão espacial das variáveis ambientais
env$altitude <- as.factor(env$altitude)
ca.env <- dudi.hillsmith(env, scannf = FALSE)
moran.env <- moran.mc(ca.env$li[, 1], mat_listw, 999)

## Estrutura espacial na composição de espécies?
moran.comp
#>
#>  Monte-Carlo simulation of Moran I
#>
#> data:  pca.comp$li[, 1]
#> weights: mat_listw
#> number of simulations + 1: 1000
#>
#> statistic = 0.62815, observed rank = 1000, p-value = 0.001
#> alternative hypothesis: greater

## Estrutura espacial na variação ambiental?
moran.env
#>
#>  Monte-Carlo simulation of Moran I
#>
#> data:  ca.env$li[, 1]
#> weights: mat_listw
#> number of simulations + 1: 1000
#>
#> statistic = 0.72714, observed rank = 1000, p-value = 0.001
#> alternative hypothesis: greater
```

Como resultado, é possível que a variação ambiental espacialmente estruturada seja o principal efeito sobre a composição de espécies. Uma maneira de visualizar a contribuição relativa de diferentes matrizes (ambiental e espacial, por exemplo) é utilizar o método de partição de variância. O resultado deste modelo indica que, de fato, não existe efeito direto das variáveis ambientais e sim do componente representado pela autocorrelação espacial dessas variáveis (Figura 9.16).

```
## Triplot da RDA
# Partição de variância.
pv.birds <- varpart(species.hel, env.pad.cat, spatial.pred)
plot(pv.birds, cutoff = -Inf)
mtext("Diagrama de Venn: partição de variância")
```

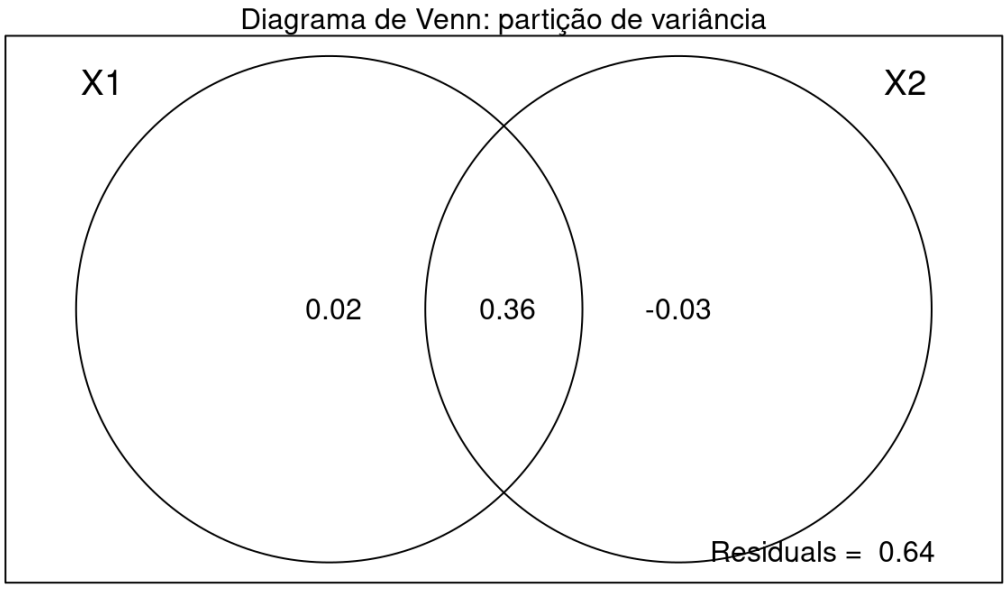


Figura 9.16: Diagrama de Venn mostrando a partição de variância da RDA.

9.5.3 Análise de Redundância baseada em distância (db-RDA)

Acima, apresentamos a análise RDA que permite comparar, por exemplo, se variáveis ambientais determinam a variação na composição de espécies. A RDA utiliza os dados brutos de composição de espécies e é baseada em uma PCA (i.e., requer distância euclidiana). Porém, em algumas situações é necessário utilizar outras medidas de distância tais como *Bray-Curtis* ou Gower. Esta análise é bastante útil também quando a matriz resposta já é uma matriz de distância, tais como as que são geradas por decomposição da diversidade beta. Neste caso, Legendre & Anderson (1999) propuseram a análise db-RDA (RDA baseada em distância). O primeiro passo da análise é gerar uma matriz de distância a partir da matriz bruta (e.g., composição de espécies). Depois, a técnica executa uma PCoA (corrigindo potenciais autovetores com autovalores negativos; ver discussão acima) para comparar com os termos do modelo (matriz **X**, como na RDA). Para exemplificar como esta análise pode ser implementada no R, vamos usar o mesmo exemplo demonstrado na análise RDA (ver acima). Porém, vamos fazer um ajuste sutil na pergunta "o clima e a altitude modificam a dissimilaridade (diversidade beta) de espécies de aves?" Veja que neste caso, estamos perguntando diretamente sobre uma medida de diversidade beta, que será calculada com o método *Bray-Curtis*. As variáveis preditoras serão as mesmas obtidas no exemplo da RDA: `env.pad.cat`.

Análise

```
## Passo 1: transformação de hellinger da matriz de espécies
# caso tenha dados de abundância.
species.hel <- decostand(species, "hellinger")

## Passo 2: gerar matriz de distância (Diversidade beta: Bray-Curtis)
dbeta <- vegdist(species.hel, "bray")

## Passo 2: dbRDA
dbrda.bird <- capscale (dbeta~rain.jul+maxi.jul+altitude,
```

```
                        data=env.pad.cat)

# Para interpretar, é necessário saber a significância dos eixos para
representar a relação entre as variáveis preditoras e a composição de
espécies
db.res.axis <- anova.cca(dbrda.bird, by = "axis")
db.res.axis
#> Permutation test for capscale under reduced model
#> Forward tests for axes
#> Permutation: free
#> Number of permutations: 999
#>
#> Model: capscale(formula = dbeta ~ rain.jul + maxi.jul + altitude, data =
env.pad.cat)
#>          Df SumOfSqs       F Pr(>F)
#> CAP1      1 0.274186 16.9355  0.001 ***
#> CAP2      1 0.044965  2.7773  0.110
#> CAP3      1 0.018141  1.1205  0.656
#> CAP4      1 0.009500  0.5868  0.788
#> Residual 18 0.291420
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Em seguida, é possível identificar quais são as variáveis que contribuem ou
que mais contribuem para a variação na composição de espécies
db.res.var <- anova.cca(dbrda.bird, by = "term") ## Qual variável?
db.res.var
#> Permutation test for rda under reduced model
#> Terms added sequentially (first to last)
#> Permutation: free
#> Number of permutations: 999
#>
#> Model: rda(formula = species.hel ~ rain.jul + maxi.jul + altitude, data =
env.pad.cat)
#>          Df Variance      F Pr(>F)
#> rain.jul  1 0.036514 9.5936  0.001 ***
#> maxi.jul  1 0.011264 2.9596  0.015 *
#> altitude  2 0.019071 2.5053  0.011 *
#> Residual 18 0.068510
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Além disso, é possível obter o valor do R2 do modelo
db_r_quadr <- RsquareAdj(dbrda.bird)
db_r_quadr
#> $r.squared
#> [1] 0.6116758
```

```
#>
#> $adj.r.squared
#> [1] 0.5253816
```

Interpretação dos resultados

Assim como apresentado na RDA, os resultados obtidos em `db.res.var` sugerem que as variáveis `rain.jul`, `maxi.jul` e `altitude` afetam a diversidade beta de aves. O valor do R^2 ajustado (0.495) sugere forte relação entre o gradiente ambiental e a diversidade beta. A comparação entre o valor do `db_r_quadr` e `r_quadr` indica que a db-RDA teve melhor ajuste para comparar o mesmo conjunto de dados.

9.6 PERMANOVA

A PERMANOVA é uma sigla, em inglês, de *Permutational Multivariate Analysis of Variance*, análise proposta por Marti Anderson (Anderson 2001). A PERMANOVA é usada para testar hipóteses multivariadas que comparam a abundância de diferentes espécies em resposta a diferentes tratamentos ou gradientes ambientais. Essa análise foi desenvolvida como forma de solucionar algumas limitações da tradicional ANOVA multivariada (MANOVA). Em especial, o pressuposto da MANOVA de distribuição normal multivariada é raramente encontrado em dados ecológicos.

O primeiro passo da PERMANOVA é selecionar uma medida de distância apropriada aos dados e, além disso, verificar a necessidade de padronização ou transformação dos dados. Em seguida, as distâncias são comparadas entre os grupos de interesse (por exemplo, tratamento versus controle) usando a estatística *F* de maneira muito parecida com uma ANOVA (Capítulo 7), chamada de *pseudo-F*:

$$F_pseudo = (SSa/SSr) * [(N - g)/(g - 1)]$$

Nessa equação

- *SSa* representa a soma dos quadrados entre grupos
- *SSr* a soma de quadrados dentro do grupo (residual)
- *N* o número de unidades amostrais
- *g* os grupos (ou níveis da variável categórica)

Esta fórmula do *pseudo-F* é específica para desenho experimental com um fator. Outros desenhos mais complexos são apresentados em Anderson (2001, 2017). O cálculo do valor de probabilidade é realizado por métodos de permutação que são discutidos em Anderson & Ter Braak (2003).

Exemplo 1

Espécies de aves que ocorrem em localidades com diferentes altitudes.

Pergunta

- O clima e a altitude modificam a composição de espécies de aves?

Predições

- Diferenças climáticas (temperatura e chuva) e altitudinais alteram a composição de espécies de aves

Variáveis

- Preditoras: Temperatura e chuva (contínuas) e altitude (categórica com três níveis)
- Dependente: composição de espécies de aves

```
## Composição de espécies padronizar com método de Hellinger
species.hel <- decostand(x = species, method = "hellinger")

## Matriz de distância com método Bray-Curtis
sps.dis <- vegdist(x = species.hel, method = "bray")
```

Para reduzir o número de variáveis no modelo, você pode considerar duas abordagens. A primeira, e mais importante delas, é manter somente variáveis preditoras que você tenha razão biológica para mantê-la e, além disso, que esteja relacionada com suas hipóteses. Assim, uma vez que você já removeu variáveis que não possuem relevância biológica, você deve usar diferentes métodos para remover as variáveis muito correlacionadas (*forward selection*, *Variance Inflation Factor (VIF)*, entre outros). Neste exemplo, vamos simplesmente fazer uma correlação múltipla e remover as variáveis com correlação maior do que 0.9 ou -0.9. A função `ggpairs()` mostra um gráfico bem didático para representar a relação entre todas as variáveis e o valor (`r`) desta correlação (mais em Capítulo 7) (Figura 9.17).

```
## Verifica correlação entre as variáveis
ggpairs(env) +
    tema_livro()

## Após verificar a estrutura de correlação, vamos manter somente três
variáveis
env2 <- env[, c("mini.jan", "rain.tot", "altitude")]
```

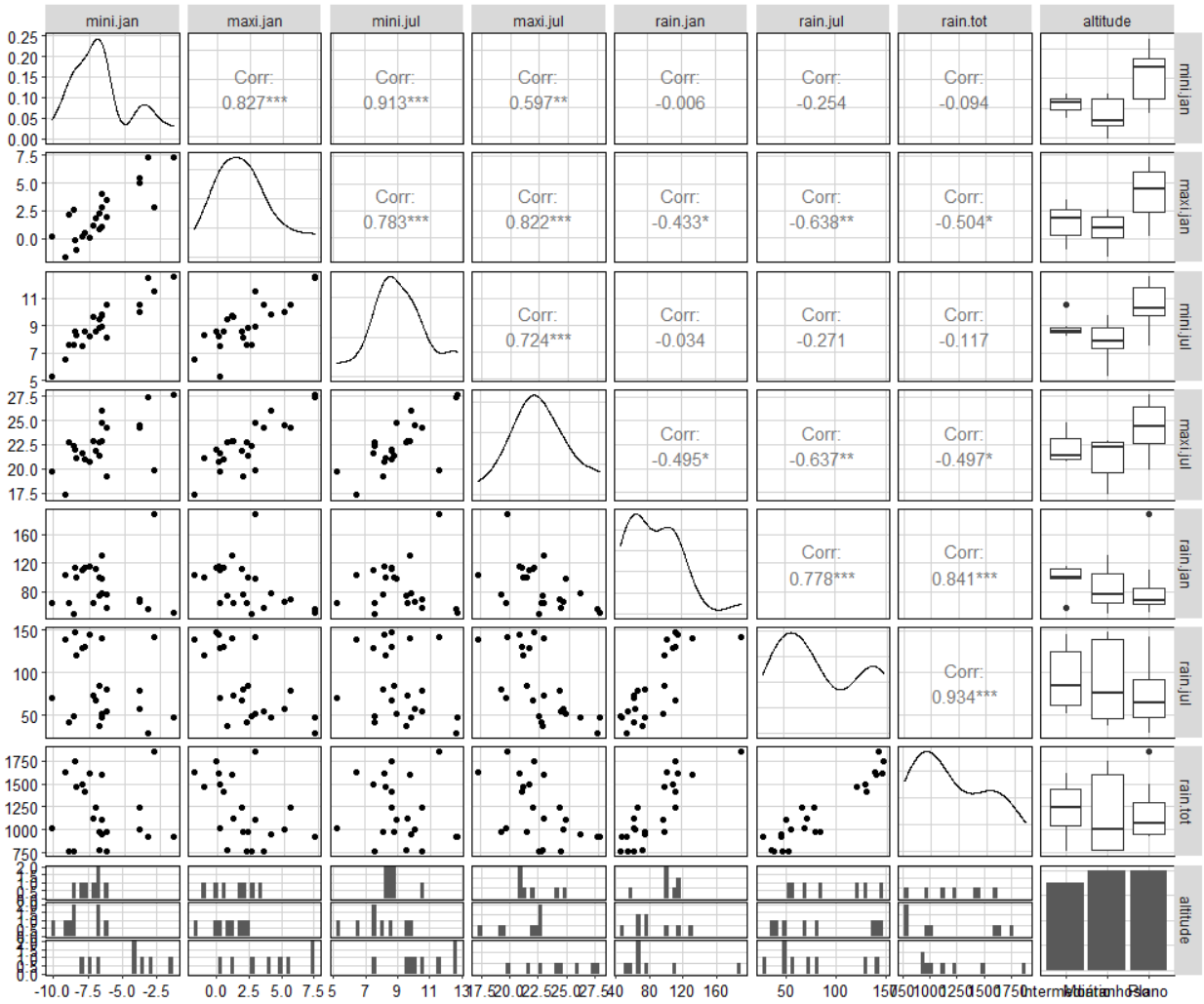


Figura 9.17: Correlograma mostrando a correlação entre as variáveis.

Após selecionar as variáveis do modelo, vamos executar a PERMANOVA e entender as principais etapas para interpretar corretamente o teste. A função `adonis()` do pacote `vegan` é uma boa opção. Porém, é importante referir o programa PRIMER e PERMANOVA+ como ótima opção para implementar a PERMANOVA e ter maior controle em desenhos complexos (Anderson et al. 2008). Assim como nos modelos lineares apresentados no Capítulo 7, os argumentos seguem o mesmo formato, com variável dependente separada por um "~" das variáveis preditoras. Porém, alguns autores demonstraram que a PERMANOVA (assim como Mantel e ANOSIM) não pode identificar se diferenças significativas do teste (usando a estatística *pseudo-F*) devem-se a diferenças na posição, na dispersão ou ambos. Ou seja, ao comparar grupos não é possível identificar se existe mudanças de composição (posição) ou se a variação da composição de espécies dentro de um grupo (dispersão) é maior do que a variação dentro do outro grupo (Anderson & Walsh 2013). Para solucionar este problema, é possível combinar a PERMANOVA com a análise PERMDISP (ou BETADISPER, como chamado no pacote `vegan` implementado através da função `betadisper()`). Esta análise permite comparar se existe heterogeneidade nas variâncias entre grupos. Deste modo, com a presença de heterogeneidade de variâncias (valor do BETADISPER significativo), é possível saber que as diferenças entre os grupos ocorrem principalmente por diferenças na dispersão e não, necessariamente, de posição. Mais detalhes sobre a relevância de combinar essas duas análises estão disponíveis em Anderson & Walsh (2013).

```
## PERMANOVA
perm.aves <- adonis2(sps.dis ~ mini.jan + rain.tot + altitude,
                     data = env2)
perm.aves
#> Permutation test for adonis under reduced model
#> Terms added sequentially (first to last)
#> Permutation: free
#> Number of permutations: 999
#>
#> adonis2(formula = sps.dis ~ mini.jan + rain.tot + altitude, data = env2)
#>          Df SumOfSqs      R2      F Pr(>F)
#> mini.jan  1  0.09069 0.15997 6.3307  0.001 ***
#> rain.tot  1  0.12910 0.22771 9.0118  0.001 ***
#> altitude  2  0.08929 0.15749 3.1163  0.011 *
#> Residual 18  0.25787 0.45483
#> Total    22  0.56695 1.00000
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

## BETADISPER
betad.aves <- betadisper(sps.dis, env2$altitude)
permutest(betad.aves)
#>
#> Permutation test for homogeneity of multivariate dispersions
#> Permutation: free
#> Number of permutations: 999
#>
#> Response: Distances
#>           Df    Sum Sq   Mean Sq      F N.Perm Pr(>F)
#> Groups     2 0.0042643 0.0021322 1.4672    999  0.265
#> Residuals 20 0.0290636 0.0014532
```

Em nosso exemplo, a temperatura, precipitação e altitude afetaram a variação na composição de espécies. Porém, para identificar se há diferenças de composição entre os níveis da variável altitude, é necessário interpretar os resultados da análise BETADISPER. O código `permutest(betad.aves)` mostra que o valor de probabilidade da análise foi de 0.253, ou seja, a hipótese nula de que a variância entre grupos é homogênea é aceita. Assim, não existe diferenças na dispersão entre grupos, sugerindo que a diferença encontrada na PERMANOVA (objeto `perm.aves`) se deve, em parte, à mudança na composição de espécies de aves entre diferentes altitudes (R^2 = 0.135). Além disso, a precipitação (R^2 = 0.183) e temperatura (R^2 = 0.127) foram fatores importantes na variação da composição de espécies.

Como falado anteriormente, as análises de ordenação irrestritas (PCA, PCoA, nMDS) são utilizadas para explorar dados. Uma maneira poderosa de usá-las é combinando com análises que testam hipóteses, como PERMANOVA e RDA. A literatura ecológica tem usado a Análise de Escalonamento não-Métrico (*non-Metric Multidimensional Scaling* - nMDS) combinado com análises multidimensionais de variância (como a PERMANOVA) para visualização da similaridade na composição de espécies dentro e entre grupos. A seguir, implementamos o nMDS na matriz de composição de espécies de ácaros (Figura 9.18).

```
## Matriz de distância representando a variação na composição de espécies
(método Bray-Curtis)
as.matrix(sps.dis)[1:6, 1:6]
#>           S01        S02        S03       S04       S05       S06
#> S01 0.0000000 0.15133734 0.16720405 0.2559122 0.2559882 0.2588892
#> S02 0.1513373 0.00000000 0.04114702 0.1190172 0.1289682 0.1391056
#> S03 0.1672040 0.04114702 0.00000000 0.1420233 0.1358127 0.1410867
#> S04 0.2559122 0.11901720 0.14202325 0.0000000 0.1140283 0.1199803
#> S05 0.2559882 0.12896823 0.13581271 0.1140283 0.0000000 0.2129054
#> S06 0.2588892 0.13910558 0.14108668 0.1199803 0.2129054 0.0000000

## É preciso calcular uma primeira "melhor" solução do nMDS
sol1 <- metaMDS(sps.dis)
nmds.beta <- metaMDS(sps.dis, previous.best = sol1)

## Stress
# O stress é o valor mais importante para interpretar a qualidade da
ordenação.
nmds.beta$stress # valor ideal entre 0 e 0.2
#> [1] 0.1272417

## Exportar os valores para fazer gráfico
dat.graf <- data.frame(nmds.beta$points,
                       altitude = env2$altitude)

## Definir os grupos ("HULL") para serem categorizados no gráfico
grp.mon <- dat.graf[dat.graf$altitude == "Montanhoso", ]
                   [chull(dat.graf[dat.graf$altitude == "Montanhoso",
                   c("MDS1", "MDS2")]), ]

grp.int <- dat.graf[dat.graf$altitude == "Intermediário", ]
                   [chull(dat.graf[dat.graf$altitude == "Intermediário",
                   c("MDS1", "MDS2")]), ]

grp.pla <- dat.graf[dat.graf$altitude == "Plano", ]
                   [chull(dat.graf[dat.graf$altitude == "Plano",
                   c("MDS1", "MDS2")]), ]

## Combinar dados dos grupos para cada Convex Hull
hull.data <- rbind(grp.mon, grp.int, grp.pla)
head(hull.data)
#>           NMDS1       NMDS2      altitude
#> S04 -0.10578586 -0.10682766    Montanhoso
#> S01 -0.25332406  0.04199004    Montanhoso
#> S11 -0.12504415  0.14477005    Montanhoso
#> S15  0.09166059  0.09857114    Montanhoso
#> S18  0.01968127 -0.12417579 Intermediário
```

```
#> S06 -0.16053825 -0.08924030 Intermediário

## Gráfico
ggplot(dat.graf, aes(x = MDS1, y = MDS2, color = altitude, shape = altitude))
+
    geom_point(size = 4, alpha = 0.7) +
    geom_polygon(data = hull.data,
                 aes(fill = altitude, group = altitude), alpha = 0.3) +
    scale_color_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    labs(x = "MDS1", y = "MDS2") +
    tema_livro()
```

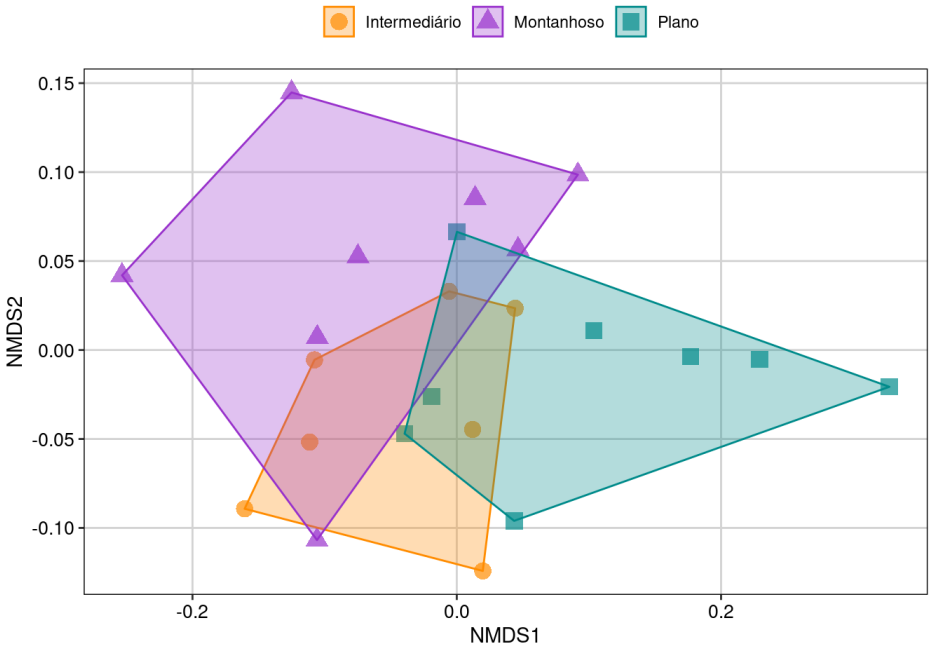


Figura 9.18: Biplot mostrando o resultado do nMDS.

9.7 Mantel e Mantel parcial

O teste de Mantel se aplica quando temos duas ou mais matrizes de distância (triangulares). Em estudos ecológicos, essas matrizes normalmente descrevem dados de distância ou (dis)similaridade na composição de espécies, distância geográfica, e similaridade ambiental. Mas este teste também é amplamente aplicado na área de genética de paisagem para testar hipóteses sobre como a composição genética de uma população varia em função do ambiente e/ou espaço geográfico (Legendre & Fortin 2010). O teste nada mais é do que uma generalização do coeficiente de correlação de Pearson (Capítulo 7) para lidar com matrizes de distância. Como essas matrizes descrevem a relação entre vários pares de locais, temos a repetição do mesmo local em outros pares. Neste caso é preciso um procedimento semelhante ao de correção para múltiplos testes.

Já o Mantel parcial pretende testar a correlação entre duas matrizes controlando para o efeito de uma terceira. Por exemplo, testar a associação entre a dissimilaridade na composição de espécies e dissimilaridade ambiental, controlando para a distância geográfica entre os locais. Isso nos permite levar em conta a possível autocorrelação espacial nas variáveis ambientais, ou seja, estaríamos testando de fato o efeito da dissimilaridade ambiental "pura" na dissimilaridade na composição de espécies, evitando o efeito do espaço em si. Portanto, note que sempre estamos raciocinando em termos de (dis)similaridade e não dos dados brutos em si, seja composição de espécies ou variáveis ambientais. É importante ter isto em mente na hora de interpretar os resultados.

Exemplo 1

Neste exemplo, vamos utilizar um conjunto de dados que contém girinos de espécies de anfíbios anuros coletados em 14 poças d'água e os respectivos dados de variáveis ambientais destas mesmas poças (Provete et al. 2014).

Pergunta

- Existe correlação entre a dissimilaridade na composição de espécies de girinos e a dissimilaridade ambiental?

Predições

- Assumindo que processos baseados no nicho sejam predominantes, quanto mais diferentes as poças forem maior será a diferença na composição de espécies

Variáveis

- Variáveis: a matriz de distância (Bray-Curtis) da composição das espécies e outra matriz de distância (Euclideana) das variáveis ambientais

Análises

Aqui vamos utilizar além do conjunto de dados `bocaina` já conhecido, outras duas matrizes disponíveis no pacote `ecodados`: `bocaina.xy` e `bocaina.env`, que contêm as coordenadas geográficas e variáveis ambientais das poças, respectivamente.

```
## Dados
head(bocaina.env)
#>       pH  tur   DO  temp veg  dep canopy conduc hydro area
#> PP3 7.07 0.66 2.26 18.93  30 0.35   30.3 0.0100   per  139
#> PP4 6.85 0.33 2.78 20.53   0 0.33   32.1 0.0100   per  223
#> AP1 6.64 0.40 4.09 15.26   5 3.50   42.0 0.0105   per  681
#> AP2 6.13 0.80 3.47 15.40  70 2.50   82.0 0.0300   per  187
#> PP1 5.80 0.75 3.22 16.75  70 0.70   91.4 0.0100   per   83
#> PP2 5.32 4.00 2.10 15.90  95 0.27   75.3 0.0100   per  108

head(bocaina.xy)
#>         Long      Lat
#> PT5 -44.6239 -22.7243
#> PP3 -44.6340 -22.7228
#> PP4 -44.6334 -22.7228
#> BP4 -44.6212 -22.7262
```

```
#> BP2 -44.6155 -22.7289
#> PT1 -44.6170 -22.7350

## Matriz de distância geográfica
# matriz de distância geográfica (geodésica) considerando a curvatura da
Terra
Dist.km <- as.dist(rdist.earth(bocaina.xy, miles=F))

## Padronizações
comp.bo.pad <- decostand(t(bocaina), "hellinger")
env.bocaina <- decostand(bocaina.env[,-9], "standardize")

## Dissimilaridades
dissimil.bocaina <- vegdist(comp.bo.pad, "bray")
dissimil.env <- vegdist(env.bocaina, "euclidian")
```

Vamos verificar a correlação entre a distância geográfica em linha reta e a dissimilaridade ambiental (Figura 9.19)

```
## Mantel
# Espaço vs. ambiente
mantel(Dist.km, dissimil.env)
#>
#> Mantel statistic based on Pearson's product-moment correlation
#>
#> Call:
#> mantel(xdis = Dist.km, ydis = dissimil.env)
#>
#> Mantel statistic r: 0.4848
#>       Significance: 0.005
#>
#> Upper quantiles of permutations (null model):
#>   90%   95% 97.5%   99%
#> 0.250 0.330 0.393 0.447
#> Permutation: free
#> Number of permutations: 999

## Preparar os dados para o gráfico
matrix.dist.env <- data.frame(x = melt(as.matrix(Dist.km))$value,
                              y = melt(as.matrix(dissimil.env))$value)

## Gráfico
ggplot(matrix.dist.env , aes(x, y)) +
    geom_point(size = 4, shape = 21, fill = "darkorange", alpha = 0.7) +
    labs(x = "Distância geográfica (km)",
         y = "Dissimilaridade Ambiental") +
    tema_livro()
```

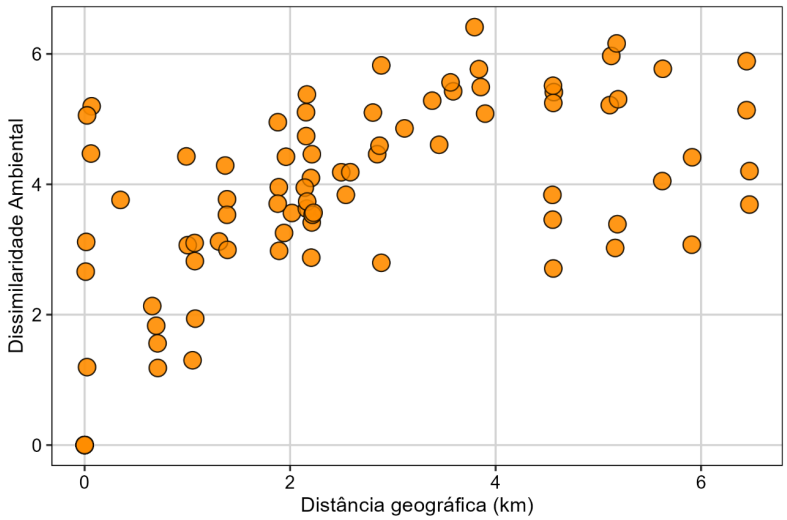


Figura 9.19: Relação das matrizes de (dis)similaridade dos dados ambientais com a distância geográfica em linha reta

Vamos verificar a correlação entre a distância geográfica em linha reta e a dissimilaridade na composição de espécies (Figura 9.20)

```
## Mantel
# Espaço vs. composição
mantel(Dist.km, dissimil.bocaina)
#>
#> Mantel statistic based on Pearson's product-moment correlation
#>
#> Call:
#> mantel(xdis = Dist.km, ydis = dissimil.bocaina)
#>
#> Mantel statistic r: 0.3745
#>       Significance: 0.007
#>
#> Upper quantiles of permutations (null model):
#>   90%   95% 97.5%   99%
#> 0.149 0.215 0.279 0.335
#> Permutation: free
#> Number of permutations: 999

## Preparar os dados para o gráfico
matrix.dist.bocaina <- data.frame(x = melt(as.matrix(Dist.km))$value,
                                  y = melt(as.matrix(dissimil.bocaina))$value)

## Gráfico
ggplot(matrix.dist.bocaina , aes(x, y)) +
    geom_point(size = 4, shape = 21, fill = "darkorange", alpha = 0.7) +
    labs(x = "Distância geográfica (km)",
         y = "Dissimilaridade (Bray-Curtis)") +
    tema_livro()
```

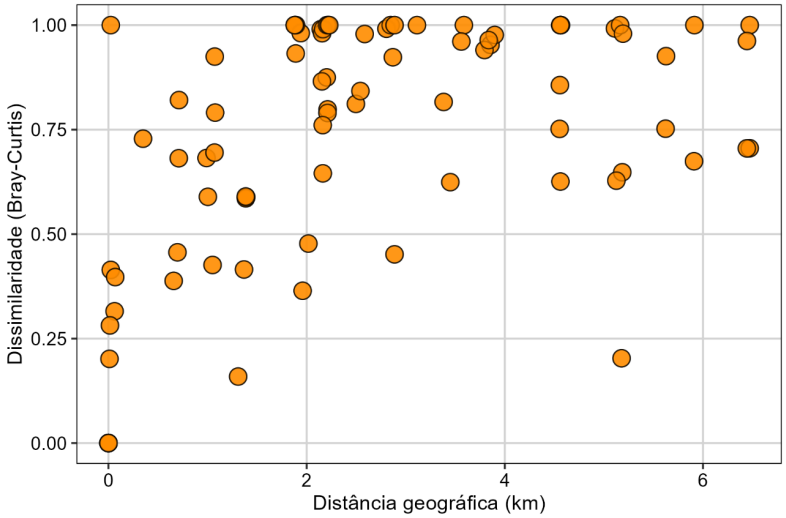


Figura 9.20: Relação das matrizes de (dis)similaridade da composição de espécies com a distância geográfica em linha reta

Vamos verificar a correlação entre a dissimilaridade ambiental e a dissimilaridade na composição de espécies (Figura 9.21).

```
## Mantel
# Ambiente vs. composição
mantel(dissimil.env, dissimil.bocaina)
#>
#> Mantel statistic based on Pearson's product-moment correlation
#>
#> Call:
#> mantel(xdis = dissimil.env, ydis = dissimil.bocaina)
#>
#> Mantel statistic r: 0.2898
#>       Significance: 0.02
#>
#> Upper quantiles of permutations (null model):
#>   90%   95% 97.5%   99%
#> 0.171 0.227 0.266 0.322
#> Permutation: free
#> Number of permutations: 999

## Preparar os dados para o gráfico
matrix.bocaina.env <- data.frame(
                            x = melt(as.matrix(dissimil.bocaina))$value,
                            y = melt(as.matrix(dissimil.env))$value)

## Gráfico
ggplot(matrix.bocaina.env, aes(x, y)) +
    geom_point(size = 4, shape = 21, fill = "darkorange", alpha = 0.7) +
    labs(x = "Similaridade Bocaina",
```

```
        y = "Similaridade Ambiental") +
    tema_livro()
```

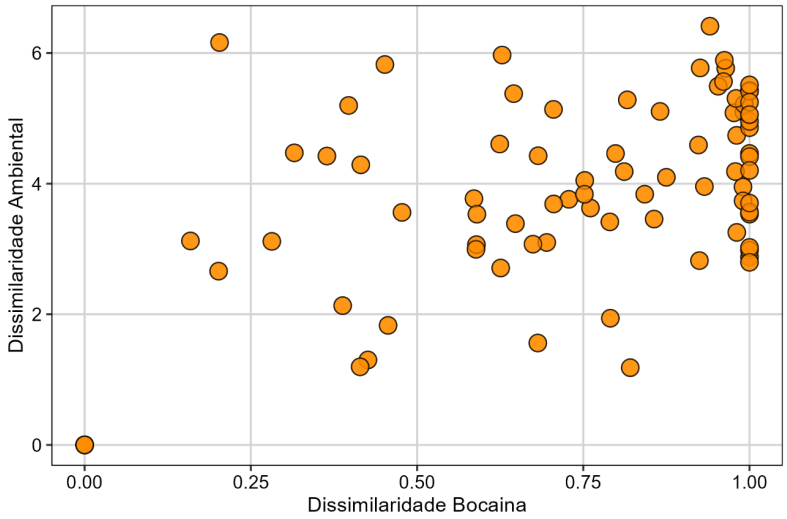


Figura 9.21: Relação das matrizes de (dis)similaridade da composição de espécies e variáveis ambientais

Interpretação dos resultados

Os resultados mostraram que existe uma relação positiva, embora fraca, entre a dissimilaridade ambiental e a dissimilaridade na composição de espécies. Este resultado está de acordo com a nossa hipótese inicial. No entanto, vimos também que existe uma relação positiva e significativa entre a distância em linha reta entre as poças e a dissimilaridade ambiental. Logo, para de fato testar a relação entre o ambiente e a composição de espécies temos de levar em conta o espaço.

Para isto temos de realizar um teste de Mantel Parcial.

```
## Mantel Parcial
#comp. vs. ambiente controlando o espaço
mantel.partial(dissimil.bocaina, dissimil.env, Dist.km)
#>
#> Partial Mantel statistic based on Pearson's product-moment correlation
#>
#> Call:
#> mantel.partial(xdis = dissimil.bocaina, ydis = dissimil.env,      zdis =
Dist.km)
#>
#> Mantel statistic r: 0.1334
#>       Significance: 0.123
#>
#> Upper quantiles of permutations (null model):
#>   90%   95% 97.5%   99%
#> 0.148 0.200 0.229 0.278
#> Permutation: free
#> Number of permutations: 999
```

Agora sim, vimos que a relação entre ambiente e espécies se mantém, mas que a força dessa relação diminuiu bastante (veja o valor de r) quando controlamos o espaço.

9.8 Mantel espacial com modelo nulo restrito considerando autocorrelação espacial

Embora seja um tipo de análise bastante popular, o Mantel e a sua versão parcial parecem não ser muito adequadas para lidar com autocorrelação espacial. Os autores dos artigos abaixo sugerem inclusive que esse tipo de análise não é adequado para testar a maioria das hipóteses em Ecologia, incluindo diversidade beta. Esse é um tema bastante discutido na literatura e não vamos nos alongar muito sobre isso, mas referimos o(a) leitor(a) para os seguintes artigos chave: Legendre et al. 2015 (2015), Legendre & Fortin 2010 (2010), e Legendre (2000).

No entanto, Crabot et al. (2019) propuseram uma modificação no teste que leva em conta autocorrelação espacial ao gerar o modelo nulo usado para o teste de hipótese. É utilizado um procedimento chamado *Moran Spectral Randomization* porque preserva a autocorrelação global medida por meio do I de Moran global. A grande vantagem é que este teste utiliza toda a flexibilidade dos *Moran Eigenvector Maps* (MEMs), que é apresentado na parte de RDA, permitindo analisar relações entre as matrizes em múltiplas escalas espaciais. Vejamos como o teste funciona.

Primeiro precisamos construir um objeto com as duas matrizes que desejamos testar, no nosso caso composição de espécies e espaço. Depois precisamos construir explicitamente a nossa hipótese de relação espacial entre os locais por meio de uma rede de vizinhança. Esta rede é uma estrutura que nos dirá como (e eventualmente com qual intensidade) se há relação entre os locais. Aqui, entende-se que nos referimos à probabilidade de fluxo de indivíduos entre os locais.

```
## Mantel
compos_espac <- mantel.randtest(sqrt(dissimil.bocaina), Dist.km)
compos_espac
#> Monte-Carlo test
#> Call: mantel.randtest(m1 = sqrt(dissimil.bocaina), m2 = Dist.km)
#>
#> Observation: 0.3738076
#>
#> Based on 999 replicates
#> Simulated p-value: 0.005
#> Alternative hypothesis: greater
#>
#>     Std.Obs Expectation    Variance
#> 3.265589171 0.002421976 0.012933827
```

Note que até aqui não há nada de novo, apenas utilizamos uma função diferente para realizar o teste de Mantel, agora no pacote `ade4` ao invés do `vegan`. Agora temos de construir a nossa rede de vizinhança. Optamos neste exemplo por uma rede bastante simples, que é a *Minimum Spanning Tree*. Essa é a rede espacial mínima que mantém todos os pontos ligados entre si (Figura 9.22).

```
## Minimum Spanning Tree
nb.boc <- mst.nb(Dist.km)
plot(nb.boc, bocaina.xy)
```

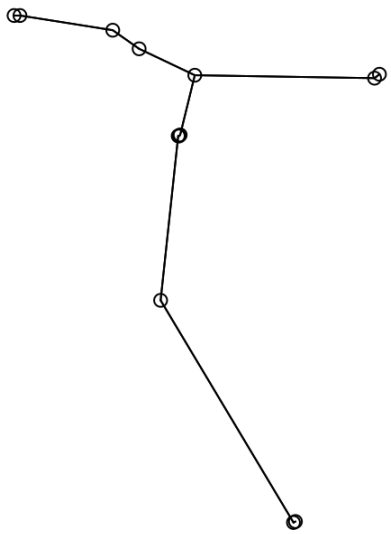


Figura 9.22: Rede de vizinhança com a Minimum Spanning Tree

No gráfico, podemos ver a "cara" da rede. Cada ponto representa um local (poça) e as linhas representam a ligação hipotética entre eles, plotamos no espaço geográfico, ou seja, o Norte aponta pra cima. Agora temos apenas que converter o formato `nb` para `listw` e finalmente entrar com esses objetos na função que realizará a análise.

```
## Conversão
lw <- nb2listw(nb.boc)

## Mantel espacial
msr(compos_espac, lw, method = "pair")
#> Monte-Carlo test
#> Call: msr.mantelrtest(x = compos_espac, listwORorthobasis = lw, method =
"pair")
#>
#> Observation: 0.3738076
#>
#> Based on 999 replicates
#> Simulated p-value: 0.127
#> Alternative hypothesis: greater
#>
#>     Std.Obs Expectation    Variance
#>  1.21335033  0.17433665  0.02702631
```

Interpretação dos resultados

Vamos comparar com o Mantel comum.

```
## Mantel comum
compos_espac
#> Monte-Carlo test
#> Call: mantel.randtest(m1 = sqrt(dissimil.bocaina), m2 = Dist.km)
#>
#> Observation: 0.3738076
#>
#> Based on 999 replicates
#> Simulated p-value: 0.005
#> Alternative hypothesis: greater
#>
#>     Std.Obs Expectation    Variance
#> 3.265589171 0.002421976 0.012933827
```

Note que o valor da estatística do teste, ou seja, o r de Mantel é exatamente o mesmo. No entanto, o valor de *P* que indicava que o teste havia sido significativo anteriormente mudou e passou a ser não significativo. Isso ocorreu porque, embora a maneira de calcular a estatística do teste seja a mesma, a forma de construir a distribuição nula, a partir da qual será calculado o valor de *P* mudou drasticamente, resultando num teste não significativo.

9.9 PROCRUSTES e PROTEST

Uma alternativa ao Mantel para comparar o grau de associação entre conjuntos multidimensionais de dados (e.g., matriz de espécies e matriz ambiental) é a análise conhecida como Procrustes (Gower 1971, Jackson 1995). Esta análise compara duas matrizes (objetos nas linhas e variáveis nas colunas) minimizando a soma dos desvios quadrados entre os valores através das seguintes técnicas: translação, escalonamento e rotação. É importante notar que os objetos devem ser os mesmos e, desse modo, o número de linhas é obrigatoriamente igual, ao passo que o número de colunas pode variar. A posição do objeto X_i (i = 1, ...n) é comparada com o objeto Y_i da matriz correspondente, movimentando os pontos via translação, reflexão, rotação e dilatação. Assim, os objetos da matriz X resultam na matriz X' que é representada pelo melhor ajuste (menor valor residual da soma dos quadrados: estatística m_{12}) entre a matriz X e Y. A estatística m_{12} mede, então, o grau de concordância entre as duas matrizes. Os valores de m_{12} variam de 1 (sem concordância) a 0 (máxima concordância) (Peres-Neto & Jackson 2001, Lisboa et al. 2014). Para testar a significância do valor observado de m_{12}, Jackson (1995) sugeriu um teste de aleatorização chamado PROTEST.

As possibilidades de aplicação do Procrustes para ecologia são diversas. Dentre elas, se destacam a: i) morfometria geométrica e ii) ecologia de comunidades (concordância de comunidades). Uma característica interessante do Procrustes é a possibilidade de usar matriz bruta (e.g., variáveis ambientais) ou matriz de distância, o que amplia as possibilidades analíticas. Para comparações com matrizes brutas, análises de ordenação irrestrita (PCA, CA) devem ser realizadas antes do Procrustes. Por outro lado, para comparações com matrizes de distância (e.g., Euclidiana, Jaccard, Bray-Curtis), análises como PCoA e nMDS devem preceder à análise de Procrustes (veja Figura 4 em Peres-Neto

& Jackson 2001). Nos dois casos, a ordenação é importante para gerar matrizes com a mesma dimensionalidade o que, por sua vez, permite comparar matrizes com número de colunas diferentes.

Exemplo 1

Neste exemplo, vamos utilizar dois conjuntos de dados simulados de peixes e macroinvertebrados aquáticos coletados em riachos. Neste exemplo hipotético, existe uma rede de 12 riachos com diferentes graus de poluição. Em cada riacho, foi produzida uma lista de espécies de peixes e de macroinvertebrados (com dados de abundância).

Pergunta

- Existe concordância na distribuição na composição de espécies de peixes e macroinvertebrados?

Predições

- Assumindo que alguns poluentes, tais como metais pesados, podem atuar como filtro ambiental e limitar a ocorrência de determinados organismos aquáticos, espera-se encontrar alta concordância entre as duas matrizes (peixes e macroinvertebrados). Ou seja, a comunidade tanto de peixes quanto de macroinvertebrados responderão de forma similar ao gradiente ambiental

Variáveis

- Variáveis: composição das espécies de peixes e macroinverebrados aquáticos

Análises

No exemplo escolhido, as duas matrizes representam a composição de espécies em diferentes riachos. Desse modo, o ideal é transformar esta matriz bruta em matriz de distância com o método Bray-Curtis (`vegdist()`). O próximo passo, então, é realizar uma `Análise de Coordenadas Principais (PCoA)` (veja abaixo) para gerar duas matrizes de ordenação que serão comparadas com a função `procrustes()`. Por fim, utilizamos a função `protest()` para testar a significância da concordância entre as matrizes (Figura 9.23).

```
## Dados
head(fish_comm)

head(macroinv)


## Fixar a amostragem
set.seed(1001)

## Matrizes de distância de PCoA
d_macro <- vegdist(macroinv, "bray")
pcoa_macro <- cmdscale(d_macro)

d_fish <- vegdist(fish_comm, "bray")
pcoa_fish <- cmdscale(d_fish)
```

```
## PROCRUSTES
concord <- procrustes(pcoa_macro, pcoa_fish)
protest(pcoa_macro, pcoa_fish)
#>
#> Call:
#> protest(X = pcoa_macro, Y = pcoa_fish)
#>
#> Procrustes Sum of Squares (m12 squared):        0.6185
#> Correlation in a symmetric Procrustes rotation: 0.6176
#> Significance:  0.019
#>
#> Permutation: free
#> Number of permutations: 999

## Gráfico
plot(concord, main = "",
     xlab = "Dimensão 1",
     ylab = "Dimensão 2")
```

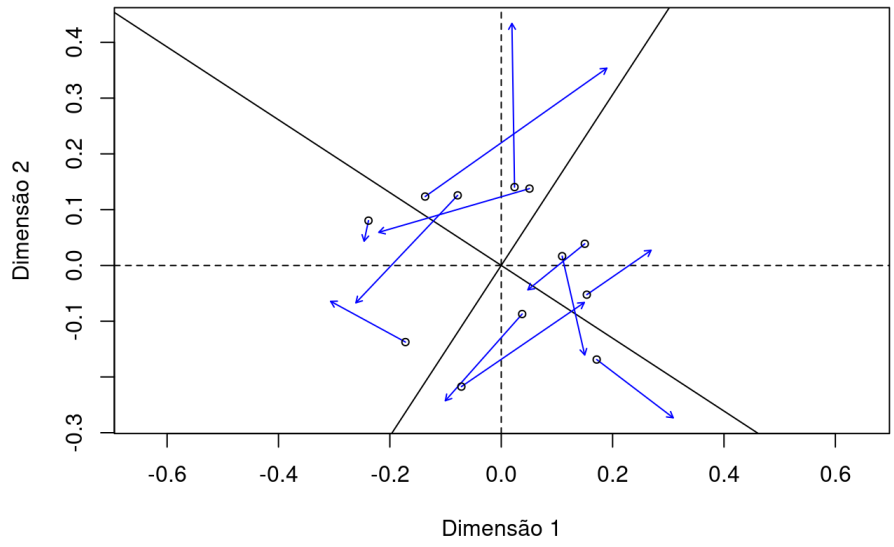


Figura 9.23: Biplot mostrando o resultado da análise de PROCRUSTES.

Interpretação dos resultados

A função `protest()` apresenta dois resultados importantes: i) a estatística m_{12} e ii) o valor de significância (`p`). No exemplo, o valor de m_{12} é igual a `0.6185` e o valor de `p` é `0,019`. Desse modo, podemos concluir que existe concordância entre a composição de espécies de peixes e macroinvertebrados aquáticos, sugerindo que o nível de poluição pode gerar padrões previsíveis de distribuição de diferentes organismos aquáticos. A figura produzida pelo código `plot(concord)` mostra um gráfico típico do Procrustes, onde a base da seta representa a matriz **X** e a ponta da seta a matriz alvo (**Y**) após a rotação para comparar o grau de concordância entre **X** e **Y**. Cada ponto representa um objeto (linha) das matrizes **X** e **Y**. Quanto menor for o tamanho das setas, mais concordantes são as observações em cada objeto.

9.10 Métodos multivariados baseados em modelos

Mais recentemente, alguns autores têm proposto métodos multivariados baseados em modelos. Estes métodos têm se diversificado hoje em dia e existem pacotes que realizam praticamente todas as análises que vimos acima, com a diferença que utilizam distribuições de probabilidade dado um modelo (e.g., Poisson) ao invés de coeficientes de dissimilaridades. Vamos exemplificar o uso de um desses métodos, mas não vamos fazer uma revisão extensiva. Caso o(a) leitor(a) deseje conhecer mais, recomendamos consultar principalmente os pacotes `gllvm`, `ecoCopula` e `Hmsc`.

Um dos primeiros métodos foi proposto em 2012 pelo grupo de David Warton (Warton et al. 2012), implementado no pacote `mvabund`. Anteriormente neste mesmo ano foi publicado um artigo pelo mesmo grupo (Wang et al. 2012) em que os autores demonstraram que métodos baseados em dissimilaridade, especialmente PERMANOVA, não conseguem modelar adequadamente dados multivariados de contagem (e.g., abundância) por não levarem em conta a relação monotônica entre média e variância. Vejamos o que isso quer dizer utilizando os dados do artigo de da Silva et al. (2017) (Figura 9.24).

```
## Dados com seis primeiras localidades e espécies
head(anuros_permanova)[1:6]

#retirando a coluna que contém o fator e deixando apenas dados de abundância
anuros_abund <- as.data.frame(anuros_permanova[,-28])

#incluindo apenas o fator a ser testado no modelo
grupos <- factor(anuros_permanova[,28])

## Média-variância
# criando um objeto da classe mvabund com os dados de abundância
abund_tr <- mvabund(anuros_abund)
meanvar.plot(abund_tr)
```

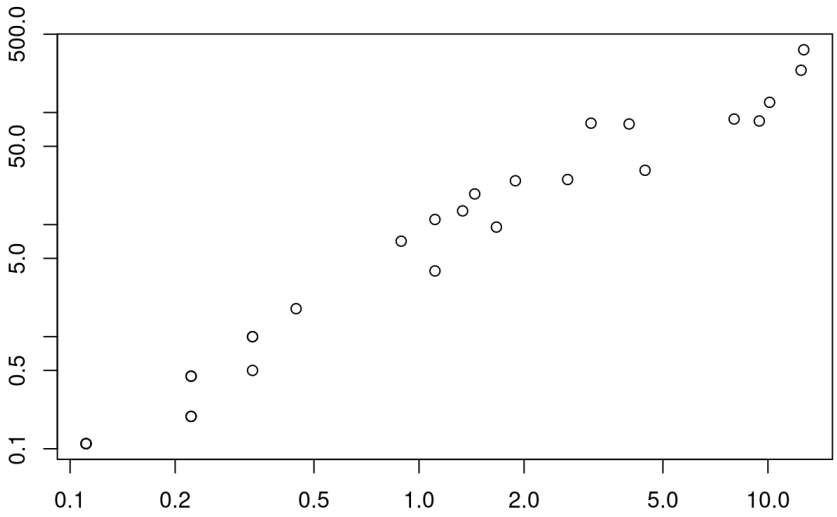


Figura 9.24: Gráfico mostrando a média-variância para dados de abundância multivariada.

Este conjunto de dados contém abundâncias de anfíbios coletados num conjunto de cidades do interior de São Paulo e também sua presença em museus. Aqui, vemos que há uma clara relação monotônica entre a média e a variância da abundância das espécies na matriz. Ou seja, à medida que espécies aumentam sua média de abundância, elas também aumentam quase que proporcionalmente a sua variância. Esta é uma propriedade bastante comum de dados multivariados de contagem, mas que não é muito bem modelada por métodos baseados em dissimilaridade, já que nesse caso apenas as espécies mais abundantes têm de fato um peso na análise.

A proposta implementada no pacote `mvabund` é baseada em Modelos Lineares Generalizados (GLMs) que você já conhece do Capítulo 8. Ou seja, este método permite que sejam modeladas as abundâncias das espécies num contexto multivariado utilizando explicitamente distribuições de probabilidade, tais como aquelas que utilizamos (e.g., Poisson, Binomial negativa, etc.). Isso faz com que demos peso semelhante às espécies. Vejamos um exemplo de como o modelo é ajustado e como é feito teste de hipótese.

Exemplo 1

A principal pergunta do artigo citado acima foi testar se a composição de espécies que obtemos em campo e de coleção científicas é diferente. Isso tem implicações para levantamentos de fauna, já que podemos complementar um tipo de dado com outro ou saber se eles são redundantes. Aqui, `grupos` é apenas um fator que indica se aquela linha da matriz apresenta dados obtidos em campo ou coleção. Vamos testar a hipótese nula de que não há diferença na composição entre as duas fontes de dados ajustando um modelo com distribuição binomial negativa (Figura 9.25).

```
## Gráfico
plot(abund_tr ~ grupos, cex.axis = 0.8, cex = 0.8)
#>
#>  PIPING TO 2nd MVFACTOR

## Modelo
modelo1 <- manyglm(abund_tr ~ grupos, family = "negative.binomial")
```

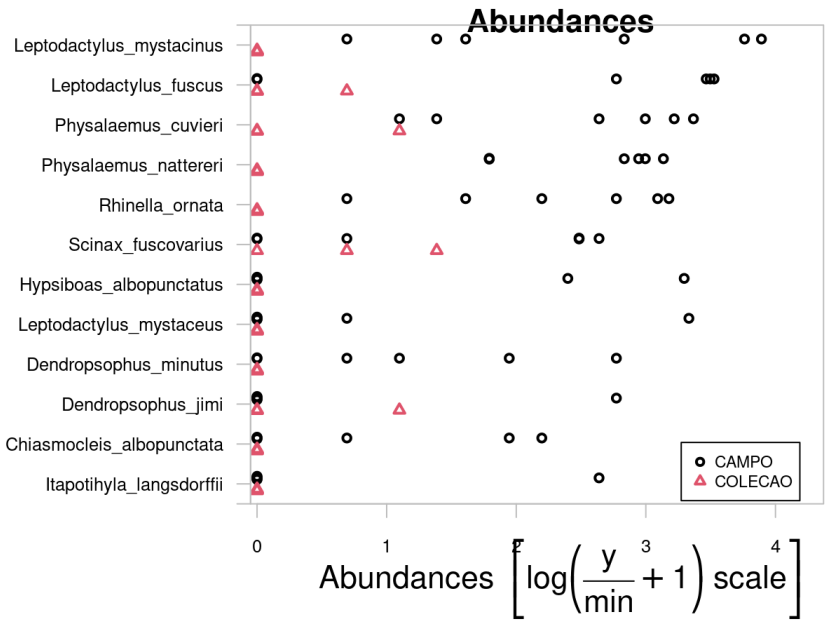


Figura 9.25: Gráfico mostrando as abundâncias para os dois grupos.

O gráfico mostrando os dados possui log da abundância das espécies e o fator (campo vs. coleção). Com essa análise exploratória de dados já podemos identificar se há diferença ou não na abundância das espécies em cada local. Vemos que algumas espécies são bem mais abundantes no campo do que em coleções, tais como *Leptodactylus mystacinus* e *Rhinella ornata*.

Uma grande vantagem desse tipo de método baseado em modelos é que, assim como um GLM típico, também podemos realizar diagnose dos resíduos utilizando plots de resíduos versus predito. Vejamos como isso funciona (Figura 9.26).

```
## Diagnose
plot(modelo1)
```

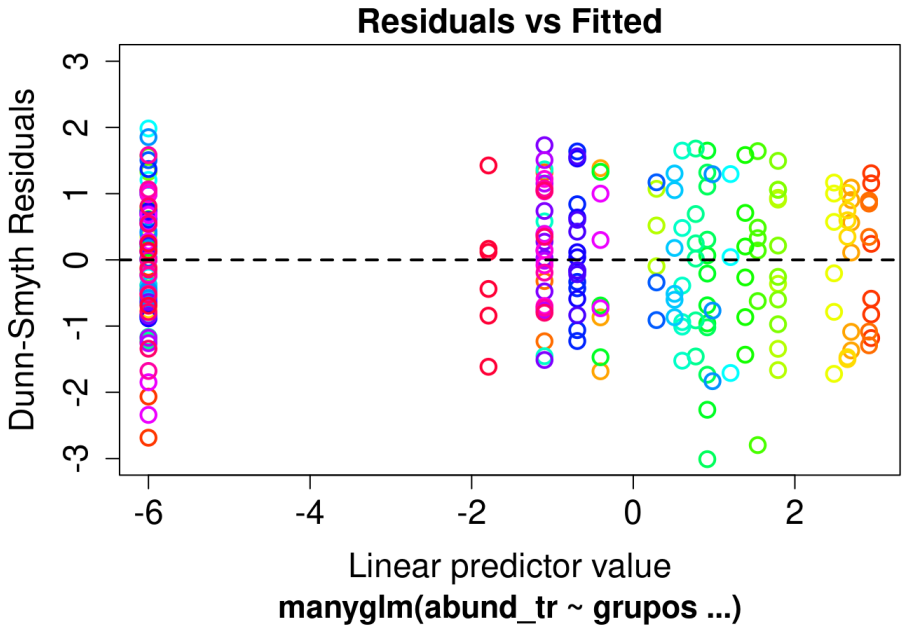


Figura 9.26: Diagnose dos resíduos do modelo ajustado.

Aqui vemos que os dados se distribuem de maneira mais ou menos homogênea e sem um padrão claro ao redor do zero. Isso indica que a distribuição Binomial negativa foi adequada para modelar estes dados. E agora sim podemos interpretar os resultados com mais segurança.

Interpretação dos resultados

```
## Resultados
summary(modelo1)
#>
#> Test statistics:
#>               wald value Pr(>wald)
#> (Intercept)       18.947      0.001 ***
#> gruposCOLECAO      4.727      0.004 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Test statistic:  4.727, p-value: 0.004
#> Arguments:
```

```
#>  Test statistics calculated assuming response assumed to be uncorrelated
#>  P-value calculated using 999 resampling iterations via pit.trap
resampling (to account for correlation in testing).
```

Aqui vemos que há sim uma diferença significativa na abundância relativa das espécies entre o campo e coleção. Mas até então estamos levando em conta toda a matriz numa abordagem multivariada. No entanto, não sabemos qual(is) espécie(s) são responsáveis por este padrão. Para investigar isso, podemos realizar uma Análise de *Deviance* separada para cada espécie, isso é o que a linha abaixo faz.

```
## ANOVA
anova(modelo1, p.uni = "adjusted")
```

Aqui vemos que algumas espécies já identificadas no plot exploratório de fato são importantes para determinar o padrão, tais como: *Leptodactylus mystacinus*, *Physalaemus nattereri* e *Rhinella ornata*. Essas foram espécies cuja Deviance foi significativa.

9.11 Para se aprofundar

9.11.1 Livros

Listamos a seguir livros e artigos com material que recomendamos para seguir com sua aprendizagem em análises multivariadas em R: i) Legendre & Legendre (2012) *Numerical Ecology*, ii) Borcard e colaboradores (2018) *Numerical Ecology with R*, iii) Thioulouse e colaboradores (2018) *Multivariate Analysis of Ecological Data with ade4*, iv) Ovaskainen & Abrego (2020) *Joint Species Distribution Modelling*, v) James & McCulloch (1990) *Multivariate Analysis in Ecology and Systematics: Panacea or Pandora's Box?* e vi) Dunstan e colaboradores (2011) *Model based grouping of species across environmental gradients*.

9.11.2 Links

Existem alguns tutoriais online bem interessantes, mas todos em inglês. Veja lista abaixo:

- Análises de ordenação - 1
- Análises de ordenação - 2
- Classificação numérica - agrupamento e kmeans

9.12 Exercícios

9.1 Utilize os dados `"mite"` do pacote `vegan` para testar o efeito de variáveis ambientais sobre a composição de espécies de ácaros utilizando as seguintes análises: RDA, RDAp (combinada com MEM), dbRDA e PERMANOVA. Após realizar as cinco análises, responda às seguintes perguntas?

A. Quais são as variáveis ambientais (`mite.env`) mais importantes para a composição de ácaros em cada uma das análises? B. Os vetores espaciais obtidos com a análise MEM explicam a variação na

composição de espécies? Eles são mais ou menos importantes do que as variáveis ambientais? C. Discuta as diferenças de interpretação entre a RDA, RDAp, dbRDA e PERMANOVA.

Dados necessários: `data(mite), data(mite.env) e data(mite.xy)`

9.2 Efetue uma análise de agrupamento pela função `hclust()`. Lembre-se de dar nome ao objeto para poder plotar o dendrograma depois. Utilize a ajuda para encontrar como entrar com os argumentos da função.

A) utilizando o método UPGMA e o índice de Bray-Curtis.
B) Faça agora o dendrograma com outro índice de dissimilaridade e compare os resultados. São diferentes? No que eles influenciariam a interpretação do resultado?

9.3 Na perspectiva de metacomunidades (Leibold et al., 2004), a dispersão dos organismos tem um papel proeminente para entender como as espécies estão distribuídas na natureza. Com o objetivo de testar se a dispersão influencia a composição de espécies de cladóceros e copépodos e, portanto, a estrutura da metacomunidade, um pesquisador selecionou dois conjuntos de lagos: em um deles todos os lagos são isolados e no outro os lagos são conectados.

A) Importe o conjunto de dados lagos do pacote `ecodados` e responda à pergunta se o fato de os lagos estarem conectados ou não influencia a composição de espécies desses microcrustáceos. Utilize métodos baseados em modelos que você aprendeu ao longo do capítulo para modelar a abundância multivariada.
B) Faça um plot mostrando a abundância relativa das espécies com maior abundância e veja se elas são diferentes entre os tipos lagos. Combine este resultado com o do item anterior para interpretar o resultado final.

9.4 Carregue o pacote `MASS` para utilizar os dados `crabs`. Este conjunto traz medidas morfológicas de dois morfo-tipos da espécie de caranguejo *Leptograpsus variegatus* coletada em Fremantle, Austrália. Calcule uma PCA e veja se existe uma semelhança morfológica entre os dois morfo-tipos. Lembre-se de dar nome ao objeto e use a função `biplot()` para plotar o resultado do teste. Dica: a projeção de um objeto perpendicular à seta do descritor fornece a posição aproximada do objeto ao longo desse descritor. A distância dos objetos no espaço cartesiano reflete a distância euclidiana entre eles.

Soluções dos exercícios.

Capítulo 10

Rarefação

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(iNEXT)
library(ecodados)
library(ggplot2)
library(vegan)
library(nlme)
library(dplyr)
library(piecewiseSEM)

## Dados
data("mite")
data("mite.xy")
coord <- mite.xy
colnames(coord) <- c("long", "lat")
data("mite.env")
agua <- mite.env[, 2]
dados_rarefacao <- ecodados::rarefacao_morcegos
rarefacao_repteis <- ecodados::rarefacao_repteis
rarefacao_anuros <- ecodados::rarefacao_anuros
dados_amostras <- ecodados::morcegos_rarefacao_amostras
```

10.1 Aspectos teóricos

Uma das grandes dificuldades na comparação da riqueza de espécies (número de espécies) entre comunidades é decorrente da diferença no esforço amostral (e.g., diferença no número de indivíduos, discrepância na quantidade de unidades amostrais ou área amostrada) que inevitavelmente influenciará no número de espécies observadas (Gotelli & Ellison 2012, Roswell et al. 2021). O método de rarefação nos permite comparar o número de espécies entre comunidades quando o tamanho da amostra (e.g., número de unidades amostrais), o esforço amostral (e.g., tempo de amostragem) ou a número de indivíduos não são iguais. A rarefação calcula o número esperado de espécies em cada comunidade tendo como base comparativa um valor em que todas as amostras atinjam um tamanho padrão. Gotelli & Colwell (2001) descrevem dois tipos de curvas de rarefação: i) baseada em indivíduos (*individual-based*) - as comparações são feitas considerando a abundância da comunidade padronizada pelo menor número de indivíduos, e ii) baseada na amostra (*sampled-based*) - as comparações são padronizadas pela comunidade com menor número de amostragens.

O método foi formulado considerando a seguinte pergunta: Se considerarmos *n* indivíduos ou amostras ($n < N$) para cada comunidade, quantas espécies registraríamos nas comunidades considerando o mesmo número de indivíduos ou amostras?

Gotelli & Colwell (2001) descrevem este método e discutem em detalhes as restrições sobre seu uso na ecologia.

- As amostras a serem comparadas devem ser consistentes do ponto de vista taxonômico, ou seja, todos os indivíduos devem pertencer ao mesmo grupo taxonômico
- As comparações devem ser realizadas somente entre amostras com as mesmas técnicas de coleta. Por exemplo, não é recomendado comparar amostras onde a riqueza de espécies de anuros de uma amostra foi estimada utilizando armadilhas de interceptação e queda e a outra foi estimada por vocalizações em sítios de reprodução
- Os tipos de hábitat onde as amostras são obtidas devem ser semelhantes
- É um método para estimar a riqueza de espécies em uma amostra menor – não pode ser usado para extrapolar a riqueza para amostras maiores

Importante
Esta última restrição foi superada por Colwell et al. (2012) e Chao & Jost (2012), que desenvolveram uma nova abordagem onde os dados podem ser interpolados (rarefeito) para amostras menores e extrapolados para amostras maiores. Além disso, Chao & Jost (2012) propõem a curva de rarefação *coverage-based* que padroniza as amostras pela cobertura ou totalidade (*completeness*) da amostra ao invés do tamanho. As rarefações tradicionais apresentam limitações matemáticas que são superadas por essa nova abordagem (Chao & Jost 2012).

10.2 Curva de rarefação baseada no indivíduo (*individual-based*)

Exemplo prático 1 - Morcegos

Explicação dos dados

Usaremos os dados de espécies de morcegos amostradas em três fragmentos florestais (Breviglieri 2008): i) Mata Ciliar do Córrego Talhadinho com 12 hectares, ii) Mata Ciliar do Córrego dos Tenentes com 10 hectares, e iii) Fazenda Experimental de Pindorama com 128 hectares.

Pergunta

- A riqueza de espécies de morcegos é maior na Fazenda Experimental do que nos fragmentos florestais menores?

Predições

- O número de espécies será maior em fragmentos florestais maiores

Variáveis

- Matriz ou data frame com as abundâncias das espécies de morcegos (variável resposta) registradas nos três fragmentos florestais (variável preditora)

Checklist

- Verificar se a sua matriz ou data frame estão com as espécies nas linhas e os fragmentos florestais nas colunas

Análise

Vamos olhar os dados usando a função `head()`.

```
## Cabeçalho dos dados
head(dados_rarefacao)
#>                       MC_Tenentes MC_Talhadinho FF_Experimental
#> Chrotopterus_auritus            0             1               1
#> Phyllostomus_hastatus           0             1               0
#> Phyllostomus_discolor           0             2               2
#> Artibeus_lituratus             17            26              26
#> Artibeus_obscurus               1             4               6
#> Artibeus_planirostris          34            72              52

## Número de indivíduos por local
colSums(dados_rarefacao)
#>     MC_Tenentes   MC_Talhadinho FF_Experimental
#>             166             413             223
```

Usaremos as funções do pacote `iNEXT` (iNterpolation e EXTrapolation) para o cálculo da rarefação rarefação (Hsieh et al. 2016). Esta função permite estimar a riqueza de espécies utilizando a família *Hill-numbers* (Hill 1973; explicação dos conceitos da família *Hill-numbers* está detalhada no Capítulo 12. O argumento `q` refere-se a família Hill-numbers onde: `0` = riqueza de espécies, `1` = diversidade de Shannon e `2` = diversidade de Simpson. No exemplo abaixo, utilizamos somente `q = 0`.

```
## Rarefação
# Datatype refere-se ao tipo de dados que você vai analisar (e.g. abundância,
incidência).
# Endpoint refere-se ao valor máximo que você determina para a extrapolação.
resultados_morcegos <- iNEXT(dados_rarefacao, q = 0,
                             datatype = "abundance", endpoint = 800)
```

Vamos visualizar os resultados (Figura 10.1).

```
## Gráfico
# type define o tipo de curva de rarefação
# 1 = curva de rarefação baseada no indivíduo ou amostra
# 2 = curva de representatividade da amostra
# 3 = curva de rarefação baseada na representatividade (coverage-based)

ggiNEXT(resultados_morcegos, type = 1) +
    geom_vline(xintercept = 166, lty = 2) +
    scale_linetype_discrete(labels = c("Interpolado", "Extrapolado")) +
    scale_colour_manual(values = c("darkorange", "darkorchid",
                                   "cyan4")) +
    scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
    labs(x = "Número de indivíduos", y = " Riqueza de espécies") +
    tema_livro()
```

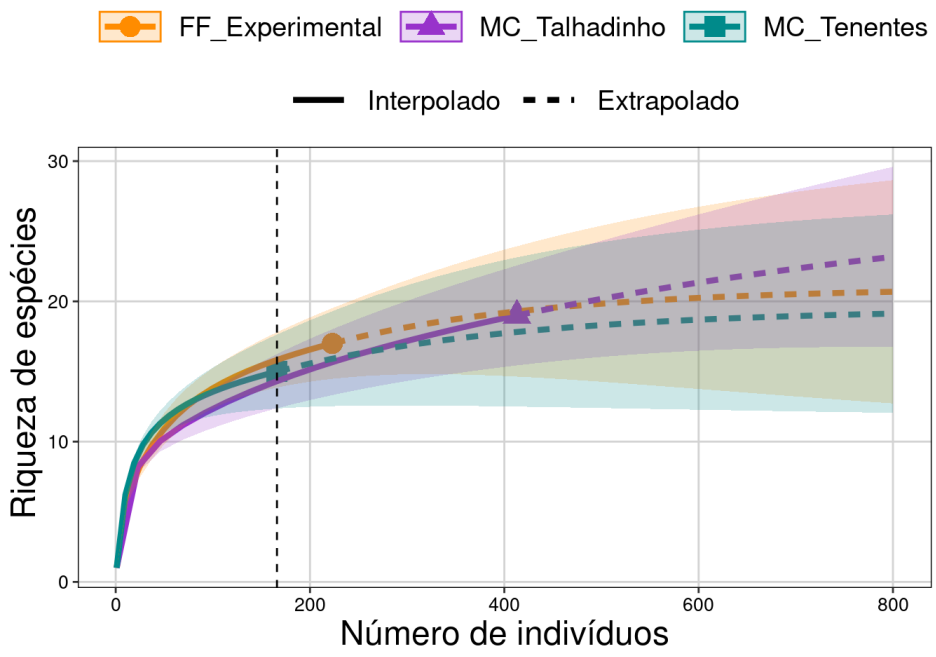


Figura 10.1: Curvas de rarefação baseada nos indivíduos de morcegos.

Interpretação dos resultados

Foram registrados 166 indivíduos na MC_Tenentes, 413 na MC_Talhadinho e 223 na FF_Experimental. Lembrando, você não pode comparar a riqueza de espécies observada diretamente: 15 espécies na MC_Tenentes, 19 espécies na MC_Talhadinho, e 17 espécies no FF_Experimental. A comparação da riqueza de espécies entre as comunidades deve ser feita com base na riqueza de espécies rarefeita, que é calculada com base no número de indivíduos da comunidade com menor abundância (166 indivíduos - linha preta tracejada). Olhando o gráfico é possível perceber que a riqueza de espécies de morcegos rarefeita não é diferente entre os três fragmentos florestais quando corrigimos o problema da diferença na abundância pela rarefação. A interpretação é feita com base no intervalo de confiança de 95%. As curvas serão diferentes quando os intervalos de confiança não se sobreporem (Chao et al. 2014). Percebam que esta abordagem, além da interpolação (rarefação), também realiza extrapolações que podem ser usadas para estimar o número de espécies caso o esforço de coleta fosse maior. Este é o assunto do Capítulo 11.

Exemplo prático 2 - Anuros e Répteis

Explicação dos dados

Neste exemplo, iremos comparar o número de espécies de anuros e répteis (serpentes e lagartos) usando informações dos indivíduos depositados em coleções científicas e coletas de campo (da Silva et al. 2017).

Pergunta

- A riqueza de espécies estimada para uma mesma região é maior usando informações de coleções científicas do que informações de coletas de campo?

Predições

- O número de espécies será maior em coleções científicas devido ao maior esforço amostral (i.e., maior variação temporal para depositar os indivíduos e maior número de pessoas contribuindo com coletas esporádicas)

Variáveis

- Matriz ou data frame com as abundâncias das espécies de anuros e répteis (variável resposta) registradas em coleções científicas e coletas de campo (variável preditora)

Checklist

- Verificar se a sua matriz ou data frame estão com as espécies nas linhas e a fonte dos dados nas colunas

Análise

Olhando os dados dos répteis.

```
## Cabeçalho
head(rarefacao_repteis)
#>                          Coleta.Campo Colecoes.Cientificas
#> Ameiva_ameiva                       1                    0
#> Amphisbaena_mertensii               1                    0
#> Apostolepis_dimidiata               0                    1
#> Bothrops__itapetiningae             0                    2
#> Bothrops__pauloensis                0                    1
#> Bothrops_alternatus                 0                    1
```

Análise usando o pacote `iNEXT` (Figura 10.2).

```
## Análise
resultados_repteis <- iNEXT(rarefacao_repteis, q = 0,
                            datatype = "abundance",
                            endpoint = 200)

## Visualizar os resultados
ggiNEXT(resultados_repteis, type = 1) +
    geom_vline(xintercept = 48, lty = 2) +
    scale_linetype_discrete(labels = c("Interpolado", "Extrapolado")) +
    scale_colour_manual(values = c("darkorange", "cyan4")) +
    scale_fill_manual(values = c("darkorange", "cyan4")) +
    labs(x = "Número de indivíduos", y = " Riqueza de espécies") +
    tema_livro()
```

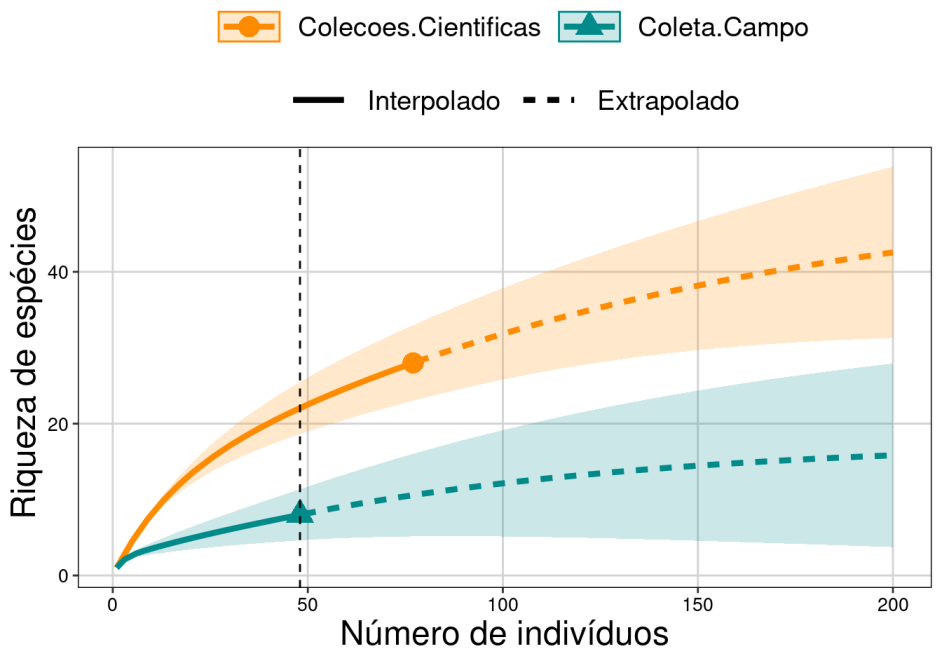


Figura 10.2: Curvas de rarefação baseada nos indivíduos de répteis.

Interpretação dos resultados

Foram registradas oito espécies de répteis nas coletas de campo (48 indivíduos) e 28 espécies nas coleções científicas (77 indivíduos). Com base na rarefação, concluímos que a riqueza de espécies de répteis obtida nas coleções científicas (20,5) é 2,5 vezes maior do que a obtida em coletas de campo.

Olhando os dados dos anuros.

```
## Cabeçalho
head(rarefacao_anuros)
#>                          Coleta.Campo Colecoes.Cientificas
#> Chiasmocleis_albopunctata          15                    0
#> Dendropsophus_elianae              11                    1
#> Dendropsophus_jimi                 15                    2
#> Dendropsophus_nanus                 0                    1
#> Dendropsophus_minutus              24                    0
#> Dendropsophus_sanborni              0                    1
```

Análise e visualização do gráfico (Figura 10.3).

```
## Análise
resultados_anuros <- iNEXT(rarefacao_anuros, q = 0,
                           datatype = "abundance", endpoint = 800)

## Visualizar os resultados
ggiNEXT(resultados_anuros, type = 1) +
    geom_vline(xintercept = 37, lty = 2) +
    scale_linetype_discrete(labels = c("Interpolado", "Extrapolado")) +
    scale_colour_manual(values = c("darkorange", "cyan4")) +
    scale_fill_manual(values = c("darkorange", "cyan4")) +
```

```
  labs(x = "Número de indivíduos", y = " Riqueza de espécies") +
  tema_livro()
```

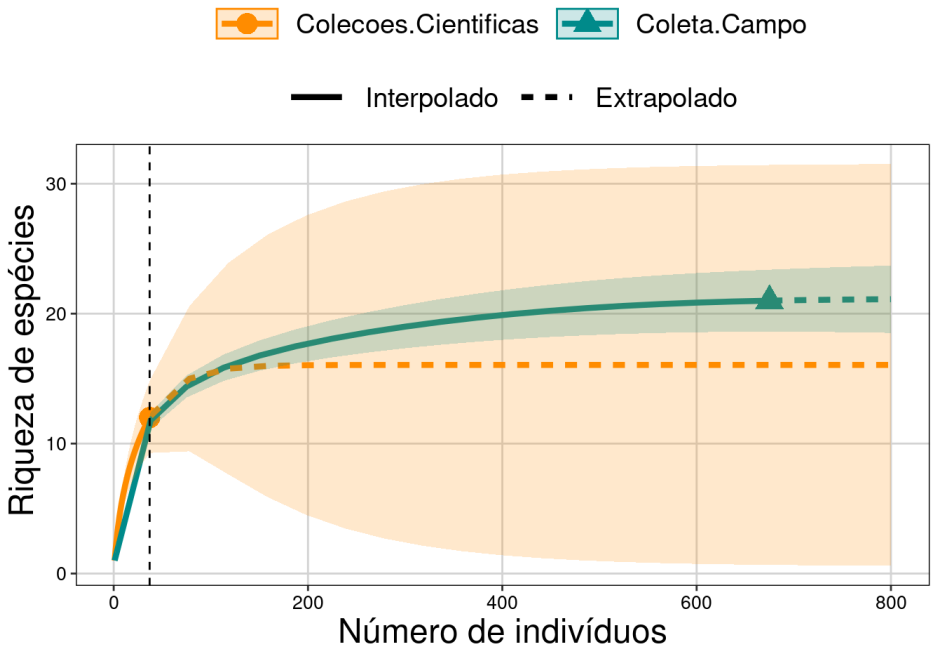


Figura 10.3: Curvas de rarefação baseada nos indivíduos de anuros.

Interpretação dos resultados - anuros

Foram registradas 21 espécies de anuros nas coletas de campo (675 indivíduos) e 12 espécies nas coleções científicas (37 indivíduos). Com base na rarefação, concluímos que não há diferença entre a riqueza de espécies de anuros obtida em coletas de campo e coleções científicas.

10.3 Curva de rarefação baseada em amostras (*sample-based*)

Exemplo prático 3 - Morcegos

Explicação dos dados

Usaremos os mesmos dados de espécies de morcegos amostradas em três fragmentos florestais (Breviglieri 2008). Contudo, ao invés de padronizarmos a riqueza de espécies pela abundância, iremos padronizar pelo número de amostras.

Variáveis

- Lista de vetores. Cada vetor deve conter como primeira informação o número total de amostras (variável preditora), seguido da frequência de ocorrência das espécies (i.e., número de amostras em que cada espécie foi registrada - variável resposta)

Checklist

- Verificar se sua lista está com o número total de amostras e a frequência de ocorrência das espécies

Análise

Vamos olhar os dados.

```
## Cabeçalho
head(dados_amostras)
#>          MC_Tenentes MC_Talhadinho FF_Experimental
#> amostras          12            20              12
#> sp1               12            20              12
#> sp2               12            19              10
#> sp3               10            15               8
#> sp4                8            10               8
#> sp5                6             7               7
```

Vamos criar uma lista com as amostragens de cada comunidade e os códigos da análise.

```
## Dados
# Usamos [,] para excluir os NAs. Lembrando que valores antes da
# vírgula representam as linhas e os posteriores representam as colunas.
lista_rarefacao <- list(Tenentes = dados_amostras[1:18, 1],
                        Talhadinho = dados_amostras[, 2],
                        Experimental = dados_amostras[1:16, 3])

## Análise
res_rarefacao_amostras <- iNEXT(lista_rarefacao, q = 0,
                                datatype = "incidence_freq")
```

Visualizar os resultados (Figura 10.4).

```
## Gráfico
ggiNEXT(res_rarefacao_amostras , type = 1, color.var = "site") +
   geom_vline(xintercept = 12, lty = 2) +
   scale_linetype_discrete(name = "Método",
                           labels = c("Interpolado", "Extrapolado")) +
   scale_colour_manual(values = c("darkorange", "darkorchid", "cyan4")) +
   scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
   labs(x = "Número de amostras", y = " Riqueza de espécies") +
   tema_livro()
```

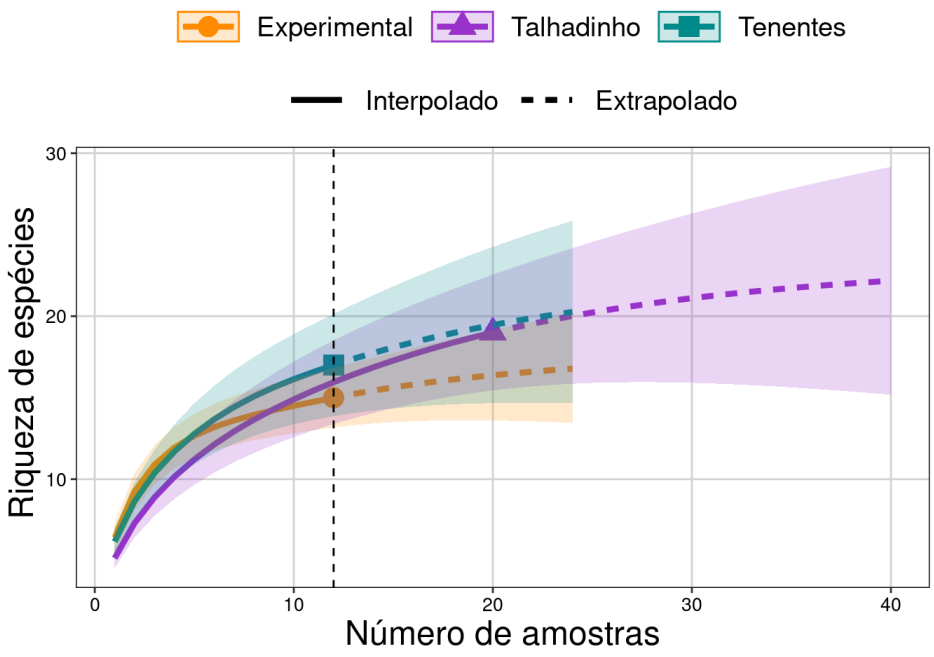


Figura 10.4: Curvas de rarefação baseada em amostras de morcegos.

Interpretação dos resultados

Olhando o gráfico é possível perceber que a riqueza de espécies de morcegos rarefeita não é diferente entre os três fragmentos florestais, quando corrigimos o problema da diferença no número de amostras.

10.4 Curva de rarefação *coverage-based*

Exemplo prático 4 - Morcegos

Explicação dos dados

Neste exemplo, usaremos os mesmos dados de espécies de morcegos amostradas em três fragmentos florestais (Breviglieri 2008).

Análise

Os códigos para a realização da rarefação *coverage-based* são idênticos aos utilizados para o cálculo das curvas de rarefações baseadas nas abundâncias e amostras. Portanto, não repetiremos as linhas de código aqui e utilizaremos os resultados já calculados para a visualização dos gráficos. Para isso, digitamos `type` = 3 que representa a curva de rarefação *coverage-based* (Figura 10.5).

```
## Gráfico
# Visualizar os resultados da rarefação *coverage-based*.
ggiNEXT(res_rarefacao_amostras, type = 3, color.var = "site") +
 geom_vline(xintercept = 0.937, lty = 2) +
 scale_linetype_discrete(labels = c("Interpolado", "Extrapolado")) +
 scale_colour_manual(values = c("darkorange", "darkorchid", "cyan4")) +
 scale_fill_manual(values = c("darkorange", "darkorchid", "cyan4")) +
```

```
labs(x = "Representatividade nas amostras", y = "Riqueza de espécies") +
tema_livro()
```

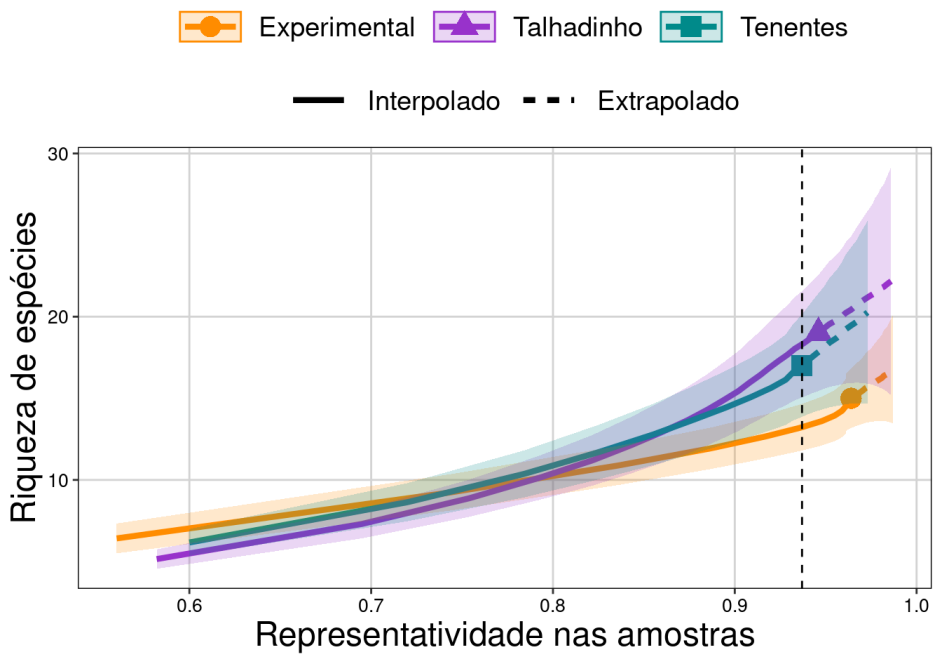


Figura 10.5: Curvas de rarefação baseada coverage-based de morcegos.

Interpretação dos resultados

Coverage é uma medida que determina a proporção de amostras (*sampled-based*) ou do número de indivíduos (*abundance-based*) da comunidade que representa as espécies presentes na amostra. Um valor de *coverage* = 0,85 representa a riqueza estimada com base em 85% das amostragens ou da abundância da comunidade. No nosso exemplo, os valores de *coverage* foram acima de 0,93, indicando que precisamos de praticamente todas as amostras para estimar a riqueza observada em cada comunidade. Comparando as comunidades considerando o mesmo valor de *coverage* 0,937 na comunidade *Tenentes*, identificamos que a riqueza de espécies de morcegos estimada na comunidade *Experimental* é menor do que a estimada para a comunidade de *Talhadinho* (não há sobreposição do intervalo de confiança). Percebam que usando a curva de rarefação *coverage-based*, a interpretação dos resultados foi diferente das observadas usando as curvas baseadas nos indivíduos ou amostras. Veja Chao & Jost (2012) e Roswell et al. (2021) para explicações mais detalhadas sobre esta metodologia.

Exemplo prático 5 - Generalized Least Squares (GLS)

Explicação dos dados

Neste exemplo, iremos refazer o exercício do Capítulo 7 onde usamos *Generalized Least Squares* (GLS) para testar a relação da riqueza de ácaros com a quantidade de água no substrato. Contudo, ao invés de considerar a riqueza de espécies de ácaros observada como variável resposta, iremos utilizar a riqueza rarefeita para controlar o efeito da amostragem (i.e., diferentes abundâncias entre as comunidades). Os dados que usaremos estão disponíveis no pacote `vegan` e representam a composição de espécies de ácaros amostradas em 70 amostras.

Pergunta

- A riqueza rarefeita de espécies de ácaros é maior em comunidades localizadas em áreas com substratos secos?

Predições

- O número de espécies rarefeita será maior em substratos secos, uma vez que as limitações fisiológicas impostas pela umidade limitam a ocorrência de várias espécies de ácaros

Variáveis

- Matriz ou data frame com as abundâncias das espécies de ácaros (variável resposta) registradas em 70 comunidades (variável preditora)

Checklist

- Verificar se a sua matriz ou data frame estão com as espécies nas linhas e as comunidades nas colunas

Análise

Vamos calcular a riqueza rarefeita com base na comunidade com menor abundância.

```
## Menor abundância
# Os dados estão com as comunidades nas colunas e as espécies nas linhas.
# Para as análises teremos que transpor a planilha.
composicao_acaros <- t(mite)

# Verificar qual é a menor abundância registrada nas comunidades.
abun_min <- min(colSums(composicao_acaros))
```

Vamos calcular a riqueza rarefeita de espécies para todas as comunidades considerando a menor abundância.

Para padronizar e facilitar a extração dos resultados, definimos os argumentos `knots` (i.e., representa o intervalo igualmente espaçado que a função irá utilizar para determinar a riqueza estimada) e `endpoint` (i.e., o valor final de amostras ou abundância extrapolados) com o valor de abundância = 8.

```
## Riqueza rarefeita
resultados_rarefacao <- iNEXT(composicao_acaros, q = 0,
                              datatype = "abundance",
                              knots = abun_min,
                              endpoint = abun_min)
```

Lembrando, estamos interessados no valor rarefeito considerando a abundância de 8 indivíduos. Esta informação está armazenada na linha 8 e na coluna 4 dos data frames salvos no objeto `resultados_rarefação$iNextEst`. Assim, para obtermos o valor rarefeito de interesse, vamos criar um `loop for` para facilitar a extração da riqueza rarefeita para as 70 comunidades.

```
## Riqueza rarefeita para cada comunidade
riqueza_rarefeita <- c()
```

```
for (i in 1:70) {
  resultados_comunidades <- resultados_rarefacao$iNextEst[[i]]
  subset_res <- resultados_comunidades %>% dplyr::filter(m==abun_min)
  riqueza_rarefeita[i] <- subset_res$qD
}
```

Vamos juntar esses resultados com os dados geográficos e ambientais.

```
## Dados finais
# Agrupando os dados em um data frame final.
dados_combinado <- data.frame(riqueza_rarefeita, agua, coord)
```

Agora, seguindo os passos descritos no Capítulo 7, vamos identificar o melhor modelo que representa a estrutura espacial dos dados da riqueza rarefeita.

```
## Criando diferentes modelos usando a função gls
## Sem estrutura espacial
no_spat_gls <- gls(riqueza_rarefeita ~ agua, data = dados_combinado,
                   method = "REML")

## Covariância esférica
espher_model <- gls(riqueza_rarefeita ~ agua, data = dados_combinado,
                    corSpher(form = ~lat + long, nugget = TRUE))

## Covariância exponencial
expon_model <- gls(riqueza_rarefeita ~ agua, data = dados_combinado,
                   corExp(form = ~lat + long, nugget = TRUE))

## Covariância Gaussiana
gauss_model <- gls(riqueza_rarefeita ~ agua, data = dados_combinado,
                   corGaus(form = ~lat + long, nugget = TRUE))

## Covariância razão quadrática
ratio_model <- gls(riqueza_rarefeita ~ agua, data = dados_combinado,
                   corRatio(form = ~lat + long, nugget = TRUE))
```

Agora vamos usar a seleção de modelo por AIC para selecionar o modelo mais "provável" explicando a distribuição da riqueza rarefeita das espécies de ácaros (Figura 10.6).

```
## Seleção dos modelos
aic_fit <- AIC(no_spat_gls, espher_model, expon_model,
               gauss_model, ratio_model)
aic_fit %>% arrange(AIC)
#>   df      AIC
#> 1  5 164.5840
#> 2  5 165.7649
#> 3  5 165.8698
#> 4  3 166.7530
#> 5  5 169.0242
```

```
## Visualizando os resíduos do modelo selecionado
plot(gauss_model)
```

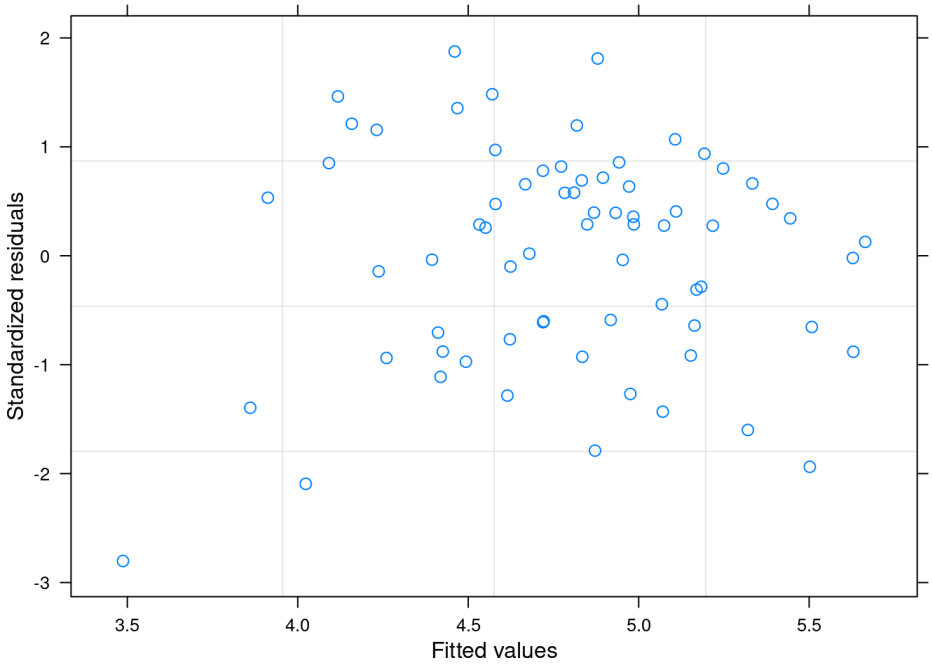


Figura 10.6: Visualização dos resíduos do modelo selecionado.

Percebam que os pontos estão dispersos no gráfico e não apresentam padrões que indiquem heterogeneidade de variância (Figura 10.7).

```
## Visualizando os resultados
summary(gauss_model)$tTable
#>                    Value    Std.Error   t-value      p-value
#> (Intercept)  6.086125990 0.2927633293 20.788553 3.550849e-31
#> agua        -0.003142615 0.0006670097 -4.711498 1.258304e-05

## Calculando o R-squared
rsquared(gauss_model)
#>            Response   family     link method R.squared
#> 1 riqueza_rarefeita gaussian identity   none 0.2991059

## Obtendo os valores preditos pelo modelo
predito <- predict(gauss_model)

## Plotando os resultados no gráfico
ggplot(data = dados_combinado, aes(x= agua, y= riqueza_rarefeita)) +
    geom_point(size = 4, shape = 21, fill = "gray", alpha = 0.7) +
    geom_line(aes(y = predito), size = 1) +
    labs(x = "Concentração de água no substrato",
         y = "Riqueza rarefeita \ndas espécies de ácaros") +
    tema_livro()
```

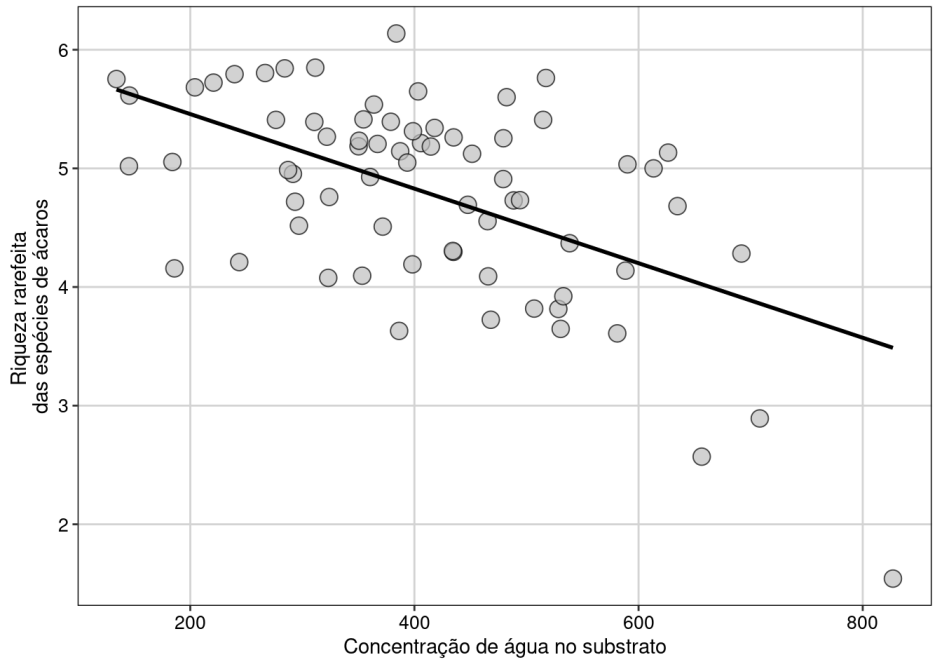


Figura 10.7: Gráfico do modelo GLS selecionado.

Interpretação dos resultados

A concentração de água no substrato explica 29,9% da variação na riqueza rarefeita das espécies de ácaros. Como predito, a riqueza de espécies de ácaros foi maior em comunidades localizadas em áreas com substratos secos do que em áreas com substratos úmidos (t = -4.71, df = 68, P < 0.01).

10.5 Para se aprofundar

10.5.1 Livros

- Recomendamos o livro *Biological Diversity Frontiers in Measurement and Assessment* (Magurran & McGill 2011)

10.5.2 Links

- Recomendamos também que acessem a página do EstimateS software e baixem o manual do usuário que contém informações detalhadas sobre os índices de rarefação. Este site foi criado e é mantido pelo Dr. Robert K. Colwell, um dos maiores especialistas do mundo em estimativas da biodiversidade
- Recomendamos a página pessoal da pesquisadora Anne Chao que é uma das responsáveis pelo desenvolvimento da metodologia e do pacote `iNEXT`. Nesta página, vocês irão encontrar exemplos e explicações detalhadas sobre as análises

10.6 Exercícios

10.1 Usando os dados `Cap10_exercicio1` disponível no pacote `ecodados`, avalie se diferentes tipos de uso da terra (fragmento florestal, borda de mata, área de pastagem e cana de açúcar) apresentam diferentes riquezas de espécies? Qual a sua interpretação? Faça um gráfico com os resultados.

10.2 O estudo é o mesmo do exercício anterior. Contudo, ao invés da rarefação baseada na abundância, faça rarefações baseadas no número de amostras. Qual a sua interpretação considerando os resultados do exercício 1? Faça um gráfico com os resultados.

10.3 Use os dados dos exercícios anteriores e calcule a rarefação baseada na cobertura de amostragem (*coverage-based*). Qual a sua interpretação considerando os resultados anteriores? Faça um gráfico com os resultados.

Soluções dos exercícios.

Capítulo 11

Estimadores de riqueza

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(iNEXT)
library(devtools)
library(ecodados)
library(ggplot2)
library(vegan)
library(nlme)
library(dplyr)
library(piecewiseSEM)

## Dados
dados_coleta <- poca_anuros
data(mite)
data(mite.xy)
coord <- mite.xy
colnames(coord) <- c("long", "lat") # altera o nome das colunas
data(mite.env)
agua <- mite.env[, 2] # seleciona a variável de interesse
```

11.1 Aspectos teóricos

Uma vez que determinar o número total de espécies numa área é praticamente impossível, principalmente em regiões com alta riqueza de espécies, os estimadores são úteis para extrapolar a riqueza observada e tentar estimar a riqueza total através de uma amostra incompleta de uma comunidade biológica (Walther & Moore 2005). Neste capítulo, serão considerados os estimadores não paramétricos que usam informações da frequência de espécies raras na comunidade (Gotelli & Chao 2013). Isto porque tanto os testes paramétricos que tentam determinar os parâmetros de uma curva usando o formato da curva de acumulação de espécies (e.g., equação logística, Michaelis-Menten), quanto os testes que usam a frequência do número de indivíduos para enquadrá-las em uma das distribuições de abundância das espécies (e.g., distribuições log-séries, log-normal) não funcionam muito bem com dados empíricos (Gotelli & Chao 2013). Para mais detalhes sobre os testes paramétricos veja Magurran & McGill (2011) e Colwell & Coddington (1994).

Para que estimador de riqueza seja considera bom, ele precisa atender a quatro características (Chazdon et al. 1998, Hortal et al. 2006).

1. Independência do tamanho da amostra (quantidade de esforço amostral realizado)
2. Insensibilidade a diferentes padrões de distribuições (e.g., agrupado, disperso ou aleatório)
3. Insensibilidade em relação à ordem das amostragens
4. Insensibilidade à heterogeneidade entre as amostras usadas entre os estudos

11.2 Estimadores baseados na abundância das espécies

11.2.1 CHAO 1

Estimador simples do número absoluto de espécies em uma comunidade. É baseado no número de espécies raras dentro de uma amostra (Chao 1984, Chao 1987).

$$Chao_1 = S_{obs} + \left(\frac{n-1}{n}\right)\frac{F_1(F_1-1)}{2(F_2+1)}$$

onde:

- S_{obs} = número de espécies observadas na comunidade
- n = número de amostras
- F_1 = número de espécies observadas com abundância de um indivíduo (espécies *singleton*)
- F_2 = número de espécies observadas com abundância de dois indivíduos (espécies *doubletons*)

O valor de Chao 1 é máximo quando todas as espécies menos uma são únicas (*singleton*). Neste caso, a riqueza estimada é aproximadamente o dobro da riqueza observada.

Exemplo prático

Explicação dos dados

Usaremos os dados hipotéticos de 17 espécies de anuros amostradas em 14 dias de coletas de campo em um habitat reprodutivo localizado na região noroeste do estado de São Paulo, Brasil.

Pergunta

- Quantas espécies a mais poderiam ser amostradas caso aumentássemos o esforço amostral até o infinito?

Predições

- O número de espécies estimadas é similar ao número de espécies observadas
- O número de espécies estimadas é maior do que o número de espécies observadas

Variáveis

- Data frame com as abundâncias das espécies de anuros (variável resposta) registradas em 14 dias de amostragens (variável preditora) em um habitat reprodutivo

Checklist

- Verificar se a sua matriz está com as espécies nas colunas e as amostragens nas linhas
- Verificar se os dados são de abundância e não de incidência (presença e ausência)

Análise

Vamos olhar os dados.

```
## Cabeçalho
head(dados_coleta)
```

Cálculo do estimador de riqueza - Chao 1.

```
## Análise
est_chao1 <- estaccumR(dados_coleta, permutations = 100)
summary(est_chao1, display = "chao")
#> $chao
#>         N      Chao   2.5%    97.5%  Std.Dev
#> Dia_13  1  7.161667  3.000 12.33333 2.786019
#> Dia_7   2 10.255429  6.000 18.81250 3.465467
#> Dia_12  3 11.614500  7.475 19.52500 3.104127
#> Dia_5   4 13.078167  9.000 20.00000 3.037566
#> Dia_6   5 14.093333  9.000 22.00000 3.259457
#> Dia_4   6 14.621667 10.000 22.00000 3.129084
#> Dia_10  7 15.308333 10.475 22.00000 3.048741
#> Dia_1   8 16.140000 12.000 22.00000 2.863106
#> Dia_11  9 16.773333 12.000 22.00000 2.777931
#> Dia_8  10 17.606667 13.000 22.00000 2.754093
#> Dia_3  11 18.205000 13.000 22.00000 2.453708
#> Dia_9  12 18.815000 14.975 22.00000 2.034618
#> Dia_14 13 19.585000 15.500 22.00000 1.494189
#> Dia_2  14 20.000000 20.000 20.00000 0.000000
#>
#> attr(,"class")
#> [1] "summary.poolaccum"
```

Percebam que a função retorna: `N` = número de amostragens, `Chao` = valor médio da estimativa do índice de Chao, `2.5%` e `97.5%` = intervalo de confiança de 95%, e `Std.Dev` = desvio padrão. Esses dados são obtidos usando permutações, sem reposição, que alteram a ordem das amostragens. Neste exemplo, usamos 100 permutações.

Vamos visualizar os resultados com intervalo de confiança de 95% (Figura 11.1).

```
## Preparando os dados para fazer o gráfico
resultados <- summary(est_chao1, display = c("S", "chao"))
res_chao <- cbind(resultados$chao[, 1:4], resultados$S[, 2:4])
res_chao <- as.data.frame(res_chao)
colnames(res_chao) <- c("Amostras", "Chao", "C_inferior", "C_superior",
                        "Riqueza", "R_inferior", "R_superior")

## Gráfico
ggplot(res_chao, aes(y = Riqueza, x = Amostras)) +
    geom_point(aes(y = Chao, x = Amostras + 0.1), size = 4,
               color = "darkorange", alpha = 0.7) +
    geom_point(aes(y = Riqueza, x = Amostras), size = 4,
               color = "cyan4", alpha = 0.7) +
    geom_point(y = 7.5, x = 9, size = 4, color = "darkorange",
               alpha = 0.7) +
    geom_point(y = 5.9, x = 9, size = 4, color = "cyan4", alpha = 0.7) +
    geom_label(y = 7.5, x = 12, label = "Riqueza estimada - Chao 1") +
```

```
  geom_label(y = 5.9, x = 11.3, label = "Riqueza observada") +
  geom_line(aes(y = Chao, x = Amostras), color = "darkorange") +
  geom_line(aes(y = Riqueza, x = Amostras), color = "cyan4") +
  geom_linerange(aes(ymin = C_inferior, ymax = C_superior,
                     x = Amostras + 0.1), color = "darkorange") +
  geom_linerange(aes(ymin = R_inferior, ymax = R_superior,
                     x = Amostras), color = "cyan4") +
  scale_x_continuous(limits = c(1, 15), breaks = seq(1, 15, 1)) +
  labs (x = "Número de amostras", y = "Riqueza estimada - Chao 1") +
  tema_livro()
```

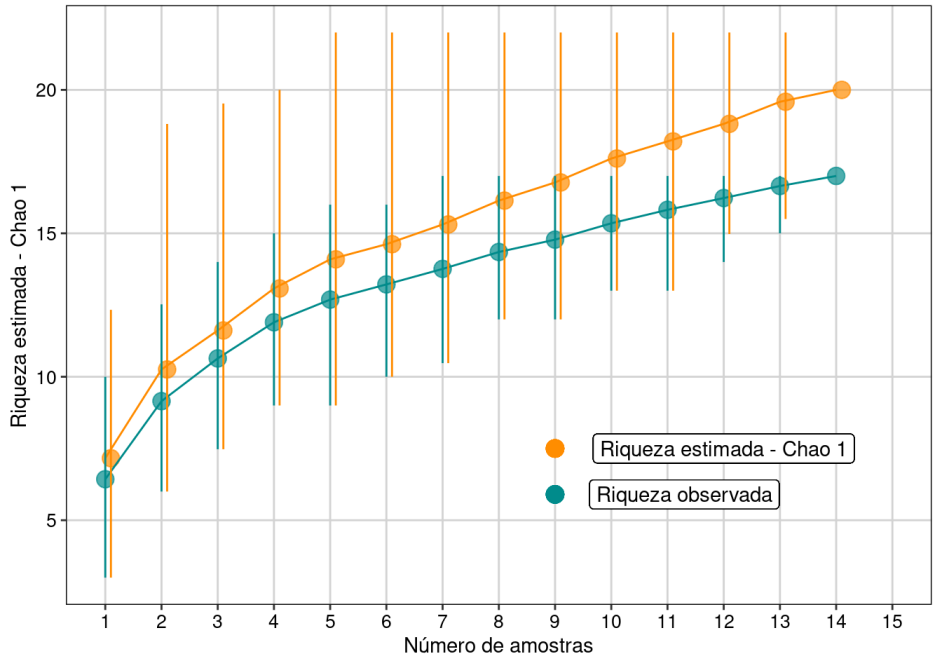


Figura 11.1: Resultados com intervalo de confiança de 95% para o estimador Chao 1.

Interpretação dos resultados

Com base no número de espécies raras (*singletons* e *doubletons*), o estimador Chao 1 indicou a possibilidade de encontrarmos mais três espécies, caso o esforço amostral fosse maior e não estima tendência de estabilização da curva em uma assíntota.

11.2.2 ACE - *Abundance-based Coverage Estimator*

Este método trabalha com a abundância das espécies raras (i.e., abundância baixa) (Chao & Lee 1992, Chao et al. 2000). Entretanto, diferente do estimador anterior, esse método permite ao pesquisador determinar os limites para os quais uma espécie seja considerada rara. Em geral, são consideradas raras espécies com abundância entre 1 e 10 indivíduos. A riqueza estimada pode variar conforme se aumente ou diminua o limiar de abundância, e infelizmente não existem critérios biológicos definidos para a escolha do melhor intervalo.

$$ACE = S_{abund} + \frac{S_{rare}}{C_{ace}} + \frac{F_1}{C_{ace}} Y_{ace}^2$$

onde:

$$Y_{ace}^{2} = max\left[\frac{S_{rare}}{C_{ace}}\frac{\sum_{i=i}^{10} i\,(i-1)F1}{(N_{rare})(N_{rare}-1)} - 1{,}0\right]$$

$$C_{ace} = 1 - \frac{F1}{N_{rare}}$$

$$N_{rare} = \sum_{i=1}^{10} i\,F_i$$

Exemplo prático

Explicação dos dados

Usaremos os mesmos dados hipotéticos de 17 espécies de anuros amostradas em 14 dias de coletas de campo em um habitat reprodutivo localizado na região Noroeste do Estado de São Paulo, Brasil.

Análise

Cálculo do estimador de riqueza - ACE.

```
## Análise
est_ace <- estaccumR(dados_coleta, permutations = 100)
summary(est_ace, display = "ace")
#> $ace
#>          N       ACE      2.5%    97.5%  Std.Dev
#> Dia_11   1  7.123899  3.545190 13.71429 2.768212
#> Dia_3    2  9.832864  6.000000 18.42880 2.876526
#> Dia_2    3 11.395043  7.619618 18.01220 2.668935
#> Dia_14   4 12.442264  8.000000 17.13587 2.398428
#> Dia_7    5 13.512461  9.328990 19.24111 2.482220
#> Dia_8    6 14.249301 10.179603 19.70014 2.608287
#> Dia_6    7 15.272604 10.712067 21.68808 2.950251
#> Dia_5    8 16.269161 11.419992 22.61582 3.000033
#> Dia_12   9 17.584889 12.635634 24.20307 3.149600
#> Dia_1   10 19.491955 13.410767 25.28994 3.732346
#> Dia_13  11 21.058884 13.923335 25.72368 3.607014
#> Dia_9   12 22.452802 15.911357 25.72368 3.249493
#> Dia_4   13 23.796512 17.676471 25.72368 2.243847
#> Dia_10  14 24.703704 24.703704 24.70370 0.000000
#>
#> attr(,"class")
#> [1] "summary.poolaccum"
```

Visualizar os resultados com intervalo de confiança de 95% (Figura 11.2).

```
## Preparando os dados para fazer o gráfico
resultados_ace <- summary(est_ace, display = c("S", "ace"))
res_ace <- cbind(resultados_ace$ace[, 1:4], resultados_ace$S[, 2:4])
res_ace <- as.data.frame(res_ace)
colnames(res_ace) <- c("Amostras", "ACE", "ACE_inferior", "ACE_superior",
```

```
                    "Riqueza", "R_inferior", "R_superior")

## Gráfico
ggplot(res_ace, aes(y = Riqueza, x = Amostras)) +
    geom_point(aes(y = ACE, x = Amostras + 0.1), size = 4,
               color = "darkorange", alpha = 0.7) +
    geom_point(aes(y = Riqueza, x = Amostras), size = 4,
               color = "cyan4", alpha = 0.7) +
    geom_point(y = 7.5, x = 9, size = 4, color = "darkorange",
               alpha = 0.7) +
    geom_point(y = 5.9, x = 9, size = 4, color = "cyan4", alpha = 0.7) +
    geom_label(y = 7.5, x = 11.7, label = "Riqueza estimada - ACE") +
    geom_label(y = 5.9, x = 11.3, label = "Riqueza observada") +
    geom_line(aes(y = ACE, x = Amostras), color = "darkorange") +
    geom_line(aes(y = Riqueza, x = Amostras), color = "cyan4") +
    geom_linerange(aes(ymin = ACE_inferior, ymax = ACE_superior,
                       x = Amostras + 0.1), color = "darkorange") +
    geom_linerange(aes(ymin = R_inferior, ymax = R_superior,
                       x = Amostras), color = "cyan4") +
    scale_x_continuous(limits = c(1, 15), breaks = seq(1, 15, 1)) +
    labs(x = "Número de amostras", y = "Riqueza estimada - ACE") +
    tema_livro()
```

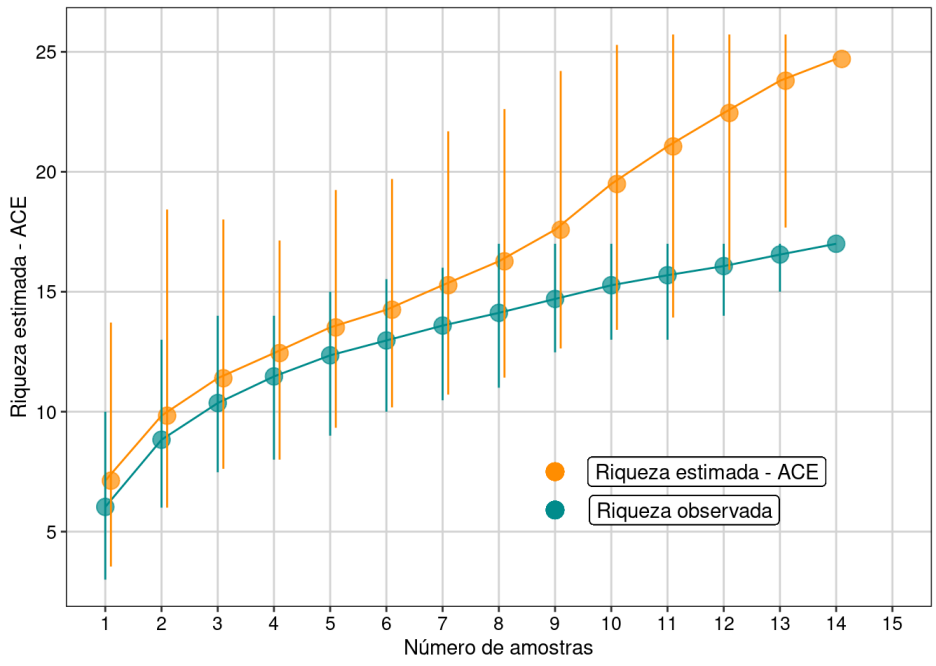


Figura 11.2: Resultados com intervalo de confiança de 95% para o estimador ACE.

Interpretação dos resultados

Com base no número de espécies raras (abundância menor que 10 indivíduos - *default*), o estimador ACE indica a possibilidade de encontrarmos mais sete espécies, caso o esforço amostral fosse maior e não estimou tendência de estabilização da curva em uma assíntota.

11.3 Estimadores baseados na incidência das espécies

11.3.1 CHAO 2

De acordo com Anne Chao, o estimador Chao 1 pode ser modificado para uso com dados de presença/ausência levando em conta a distribuição das espécies entre amostras (Chao 1987). Neste caso é necessário somente conhecer o número de espécies encontradas em somente uma amostra e o número de espécies encontradas exatamente em duas amostras. Essa variação ficou denominada como Chao 2.

$$Chao_2 = S_{obs} + \left(\frac{m-1}{m}\right)$$

onde:

- S_{obs} = o número de espécies observada na comunidade
- m = número de amostras
- Q_1 = número de espécies observadas em uma amostra (espécies *uniques*)
- Q_2 = número de espécies observadas em duas amostras (espécies *duplicates*)

O valor de Chao2 é máximo quando as espécies menos uma são únicas (*uniques*). Neste caso, a riqueza estimada é aproximadamente o dobro da riqueza observada.

Colwell & Coddington (1994) encontraram que o valor de Chao 2 mostrou ser o estimador menos enviesado para amostras com tamanho pequeno.

> **Importante**
> Você perceberá que ao longo do capítulo as recomendações sobre qual é o melhor índice varia entre estudos (e.g., Palmer 1990, Walther & Moore 2005).

Exemplo prático

Explicação dos dados

Usaremos novamente os mesmos dados hipotéticos de 17 espécies de anuros amostradas em 14 dias de coletas de campo em um habitat reprodutivo localizado na região Noroeste do Estado de São Paulo, Brasil.

Análise

Cálculo do estimador de riqueza - Chao 2.

```
## Análise
est_chao2 <- poolaccum(dados_coleta, permutations = 100)
summary(est_chao2, display = "chao")
#> $chao
#>         N     Chao      2.5%    97.5%  Std.Dev
#>  [1,]   3 14.31571  9.211111 24.35000 3.909186
```

```
#>  [2,]  4 15.12125  8.796875 26.50000 4.676977
#>  [3,]  5 17.18113 10.485000 34.12500 5.116710
#>  [4,]  6 18.40042 11.336806 34.50625 5.802968
#>  [5,]  7 20.06214 12.142857 34.00000 6.145670
#>  [6,]  8 21.28187 12.391927 38.11562 6.923969
#>  [7,]  9 22.60556 12.538889 36.82500 6.845149
#>  [8,] 10 25.94025 14.327500 42.20000 7.446691
#>  [9,] 11 27.58773 15.401136 39.27273 6.931535
#> [10,] 12 29.55229 19.933333 39.45833 6.111500
#> [11,] 13 31.51769 22.384615 39.61538 4.586452
#> [12,] 14 33.71429 33.714286 33.71429 0.000000
#>
#> attr(,"class")
#> [1] "summary.poolaccum"
```

Visualizar os resultados com intervalo de confiança de 95% (Figura 11.3).

```
## Preparando os dados para fazer o gráfico
resultados_chao2 <- summary(est_chao2, display = c("S", "chao"))
res_chao2 <- cbind(resultados_chao2$chao[, 1:4],
                   resultados_chao2$S[, 2:4])
res_chao2 <- as.data.frame(res_chao2)
colnames(res_chao2) <- c("Amostras", "Chao2", "C_inferior", "C_superior",
                         "Riqueza", "R_inferior", "R_superior")

## Gráfico
ggplot(res_chao2, aes(y = Riqueza, x = Amostras)) +
    geom_point(aes(y = Chao2, x = Amostras + 0.1), size = 4,
               color = "darkorange", alpha = 0.7) +
    geom_point(aes(y = Riqueza, x = Amostras), size = 4,
               color = "cyan4", alpha = 0.7) +
    geom_point(y = 9.8, x = 10, size = 4, color = "darkorange",
               alpha = 0.7) +
    geom_point(y = 7.7, x = 10, size = 4, color = "cyan4", alpha = 0.7) +
    geom_label(y = 9.8, x = 12.95, label = "Riqueza estimada - Chao 2") +
    geom_label(y = 7.7, x = 12.3, label = "Riqueza observada") +
    geom_line(aes(y = Chao2, x = Amostras), color = "darkorange") +
    geom_line(aes(y = Riqueza, x = Amostras), color = "cyan4") +
    geom_linerange(aes(ymin = C_inferior, ymax = C_superior,
                       x = Amostras + 0.1), color = "darkorange") +
    geom_linerange(aes(ymin = R_inferior, ymax = R_superior,
                       x = Amostras), color = "cyan4") +
    scale_x_continuous(limits = c(1, 15), breaks = seq(1, 15, 1)) +
    labs(x = "Número de amostras", y = "Riqueza estimada - Chao 2") +
    tema_livro()
```

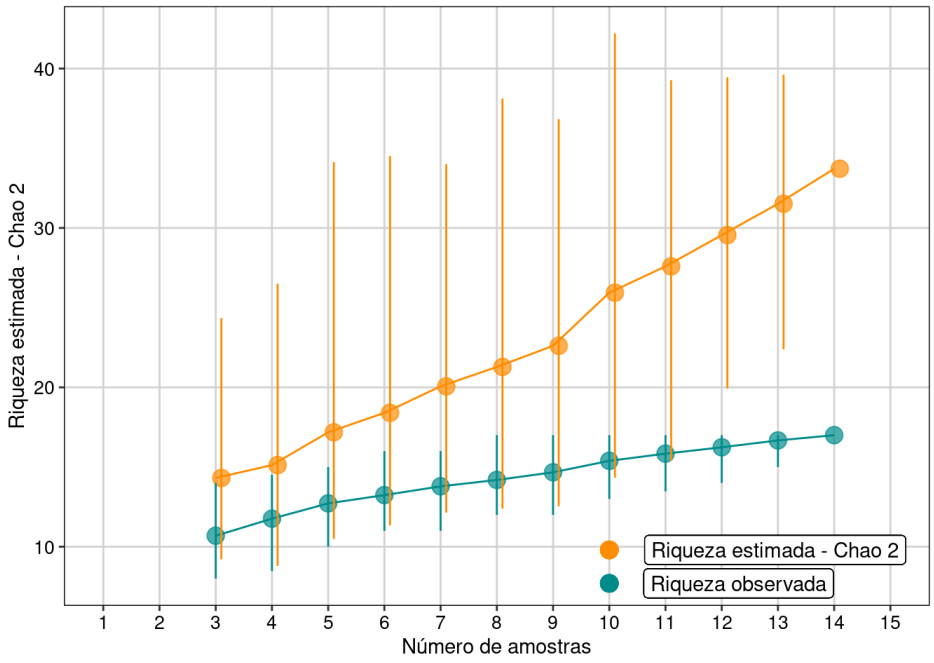


Figura 11.3: Resultados com intervalo de confiança de 95% para o estimador Chao 2.

Interpretação dos resultados

Com base no número de espécies raras (*uniques* e *duplicates*), o estimador Chao 2 encontrou a possibilidade de acharmos mais dezesseis espécies, caso o esforço amostral fosse maior e não estimou tendência de estabilização da curva em uma assíntota.

11.3.2 JACKKNIFE 1

Este estimador baseia-se no número de espécies que ocorrem em somente uma amostra (Q_1)(Burnham & Overton 1978, Burnham & Overton 1979).

$$S_{jack1} = S_{obs} + Q_1\left(\frac{m-1}{m}\right)$$

onde:

- S_{obs} = o número de espécies observadas na comunidade
- Q_1 = número de espécies observadas em uma amostra (espécies *uniques*)
- m = número de amostras

Jackknife é um método de reamostragem (sem repetição) não paramétrico usado para estimar a riqueza de espécies e a variância associada com a estimativa. Para isso, o método: i) exclui uma amostra e contabiliza o valor da riqueza estimada usando a fórmula acima, ii) repete este processo *n* vezes até que todas as amostras tenham sido excluídas, e iii) estima a média e a variância da riqueza de espécie (Smith & van Belle 1984).

Palmer (1990) verificou que Jackknife 1 foi o estimador mais preciso e menos enviesado comparado a outros métodos de extrapolação.

Exemplo prático

Explicação dos dados

Usaremos os mesmos dados hipotéticos de 17 espécies de anuros amostradas em 14 dias de coletas de campo em um habitat reprodutivo localizado na região Noroeste do Estado de São Paulo, Brasil.

Análise

Cálculo do estimador de riqueza - Jackknife 1.

```
## Análise
est_jack1 <- poolaccum(dados_coleta, permutations = 100)
summary(est_jack1, display = "jack1")
#> $jack1
#>        N Jackknife 1      2.5%    97.5%  Std.Dev
#>  [1,]  3    13.64667  8.333333 19.17500 2.798035
#>  [2,]  4    14.75000  9.750000 20.01250 2.788188
#>  [3,]  5    15.32800  9.800000 20.42000 2.780091
#>  [4,]  6    16.01500 11.229167 21.96250 2.802425
#>  [5,]  7    17.07714 12.782143 22.93214 2.575868
#>  [6,]  8    18.11625 14.165625 23.12500 2.536306
#>  [7,]  9    18.77778 14.719444 23.22222 2.609373
#>  [8,] 10    19.56100 14.800000 23.77250 2.586909
#>  [9,] 11    20.42455 16.727273 23.36364 2.190412
#> [10,] 12    21.16000 17.185417 23.41667 1.863867
#> [11,] 13    21.92846 18.692308 23.46154 1.335047
#> [12,] 14    22.57143 22.571429 22.57143 0.000000
#>
#> attr(,"class")
#> [1] "summary.poolaccum"
```

Visualizar os resultados com 95% intervalo de confiança (Figura 11.4).

```
## Preparando os dados para fazer o gráfico
resultados_jack1 <- summary(est_jack1, display = c("S", "jack1"))
res_jack1 <- cbind(resultados_jack1$jack1[, 1:4],
                   resultados_jack1$S[, 2:4])
res_jack1 <- as.data.frame(res_jack1)
colnames(res_jack1) <- c("Amostras", "JACK1", "JACK1_inferior",
                         "JACK1_superior", "Riqueza", "R_inferior",
                        "R_superior")

## Gráfico
ggplot(res_jack1, aes(y = Riqueza, x = Amostras)) +
    geom_point(aes(y = JACK1, x = Amostras + 0.1), size = 4,
               color = "darkorange", alpha = 0.7) +
    geom_point(aes(y = Riqueza, x = Amostras), size = 4,
               color = "cyan4", alpha = 0.7) +
    geom_point(y = 9.9, x = 9, size = 4, color = "darkorange",
```

```
            alpha = 0.7) +
  geom_point(y = 8.6, x = 9, size = 4, color = "cyan4", alpha = 0.7) +
  geom_label(y = 9.9, x = 12.5,
             label = "Riqueza estimada - Jackknife 1") +
  geom_label(y = 8.6, x = 11.5, label = "Riqueza observada") +
  geom_line(aes(y = JACK1, x = Amostras), color = "darkorange") +
  geom_line(aes(y = Riqueza, x = Amostras), color = "cyan4") +
  geom_linerange(aes(ymin = JACK1_inferior, ymax = JACK1_superior,
                     x = Amostras + 0.1), color = "darkorange") +
  geom_linerange(aes(ymin = R_inferior, ymax = R_superior,
                     x = Amostras), color = "cyan4") +
  scale_x_continuous(limits = c(1, 15), breaks = seq(1, 15, 1)) +
  labs(x = "Número de amostras",
       y = "Riqueza estimada - Jackknife 1") +
  tema_livro()
```

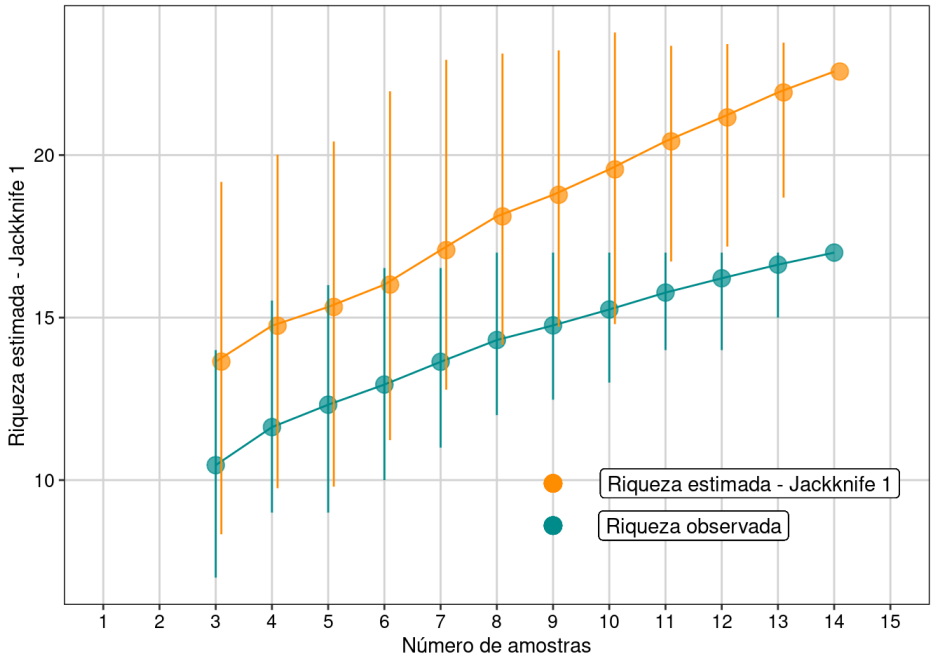


Figura 11.4: Resultados com intervalo de confiança de 95% para o estimador Jackknife 1.

Interpretação dos resultados

Com base no número de espécies raras, o estimador Jackknife 1 estimou possibilidade de encontrarmos mais seis espécies, caso o esforço amostral fosse maior e não estimou tendência de estabilização da curva em uma assíntota.

11.3.3 JACKKNIFE 2

Este método baseia-se no número de espécies que ocorrem em apenas uma amostra e no número de espécies que ocorrem em exatamente duas amostras (Burnham & Overton 1978, Burnham & Overton 1979, Palmer 1991).

$$S_{jack2} = S_{obs} + \left[\frac{Q_1(2m-3)}{m} - \frac{Q_2(m-2)^2}{m(m-1)}\right]$$

onde:

- S_{obs} = o número de espécies observadas na comunidade
- m = número de amostras
- Q_1 = número de espécies observadas em uma amostra (espécies *uniques*)
- Q_2 = número de espécies observadas em duas amostras (espécies *duplicates*)

Exemplo prático

Explicação dos dados

Usaremos os mesmos dados hipotéticos de 17 espécies de anuros amostradas em 14 dias de coletas de campo em um habitat reprodutivo localizado na região Noroeste do Estado de São Paulo, Brasil.

Análise

Cálculo do estimador de riqueza - Jackknife 2.

```
## Análise
est_jack2 <- poolaccum(dados_coleta, permutations = 100)
summary(est_jack2, display = "jack2")
#> $jack2
#>        N Jackknife 2      2.5%    97.5%  Std.Dev
#>  [1,]  3    14.27667  7.904167 21.50000 3.750886
#>  [2,]  4    15.73500  8.566667 23.77292 4.127584
#>  [3,]  5    16.36650  9.150000 24.68625 4.262113
#>  [4,]  6    18.16900 10.631667 27.46667 4.274110
#>  [5,]  7    19.57786 12.339881 27.21250 3.875295
#>  [6,]  8    20.78679 14.133929 28.58527 3.778223
#>  [7,]  9    21.82028 13.972222 27.98611 3.960469
#>  [8,] 10    22.84444 14.977778 28.18889 3.770196
#>  [9,] 11    24.04227 17.445455 28.35455 3.123930
#> [10,] 12    24.99455 20.242424 28.49242 2.618660
#> [11,] 13    26.09481 21.301282 28.60897 1.953804
#> [12,] 14    26.92308 26.923077 26.92308 0.000000
#>
#> attr(,"class")
#> [1] "summary.poolaccum"
```

Visualizar os resultados com intervalo de confiança de 95% (Figura 11.5).

```
## Preparando os dados para fazer o gráfico
resultados_jack2 <- summary(est_jack2, display = c("S", "jack2"))
res_jack2 <- cbind(resultados_jack2$jack2[, 1:4],
                   resultados_jack2$S[, 2:4])
res_jack2 <- as.data.frame(res_jack2)
colnames(res_jack2) <- c("Amostras", "JACK2", "JACK2_inferior",
                         "JACK2_superior", "Riqueza", "R_inferior",
```

```
                          "R_superior")

## Gráfico
ggplot(res_jack2, aes(y = Riqueza, x = Amostras)) +
    geom_point(aes(y = JACK2, x = Amostras + 0.1), size = 4,
               color = "darkorange", alpha = 0.7) +
    geom_point(aes(y = Riqueza, x = Amostras), size = 4,
               color = "cyan4", alpha = 0.7) +
    geom_point(y = 9.9, x = 9, size = 4, color = "darkorange",
               alpha = 0.7) +
    geom_point(y = 8.2, x = 9, size = 4, color = "cyan4", alpha = 0.7) +
    geom_label(y = 9.9, x = 12.5,
               label = "Riqueza estimada - Jackknife 2") +
    geom_label(y = 8.2, x = 11.5, label = "Riqueza observada") +
    geom_line(aes(y = JACK2, x = Amostras), color = "darkorange") +
    geom_line(aes(y = Riqueza, x = Amostras), color = "cyan4") +
    geom_linerange(aes(ymin = JACK2_inferior, ymax = JACK2_superior,
                       x = Amostras + 0.1), color = "darkorange") +
    geom_linerange(aes(ymin = R_inferior, ymax = R_superior,
                       x = Amostras), color = "cyan4") +
    scale_x_continuous(limits = c(1, 15), breaks = seq(1, 15, 1)) +
    labs(x = "Número de amostras",
         y = "Riqueza estimada - Jackknife 2") +
    tema_livro()
```

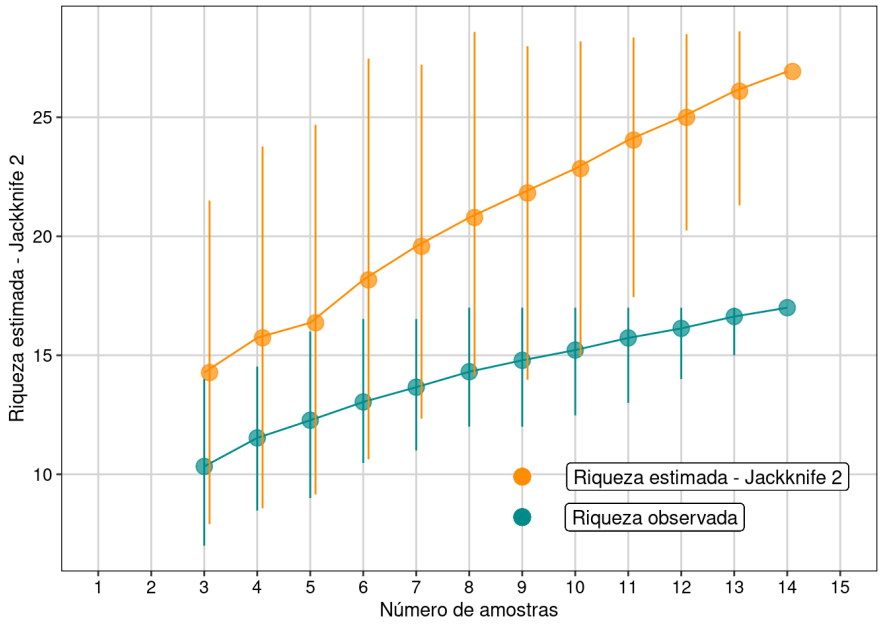


Figura 11.5: Resultados com intervalo de confiança de 95% para o estimador Jackknife 2.

Interpretação dos resultados

Com base no número de espécies raras, o estimador Jackknife 2 estimou a possibilidade de encontrarmos mais dez espécies, caso o esforço amostral fosse maior e não estimou tendência estabilização da curva em uma assíntota.

11.3.4 BOOTSTRAP

Este método difere dos demais por utilizar dados de todas as espécies coletadas para estimar a riqueza total, não se restringindo às espécies raras (Smith & van Belle 1984). Ele requer somente dados de incidência. A estimativa pelo bootstrap é calculada somando-se a riqueza observada à soma do inverso da proporção de amostras em que cada espécie ocorre.

$$S_{boot} = S_{obs} + \sum_{k=1}^{S_{obs}} (1 - P_k)^m$$

onde:

- S_{obs} = o número de espécies observadas na comunidade
- m = número de amostragens
- P_k = proporção do número de amostras em que cada espécie foi registrada

Bootstrap é um método não paramétrico usado para estimar os parâmetros de uma população por reamostragem. A premissa é que as reamostragens podem ser entendidas como pseudo-populações, com características similares as da população original. Para isso, o método: i) seleciona ao acaso um conjunto de amostras (no nosso exemplo 14 amostras) **com reposição**, ii) repete este processo *n* vezes, e iii) estima a média e a variância da riqueza de espécie (Smith & van Belle 1984).

Exemplo prático

Explicação dos dados

Usaremos os mesmos dados hipotéticos de 17 espécies de anuros amostradas em 14 dias de coletas de campo em um habitat reprodutivo localizado na região Noroeste do Estado de São Paulo, Brasil.

Análise

Cálculo do estimador de riqueza - Bootstrap.

```
## Análise
est_boot <- poolaccum(dados_coleta, permutations = 100)
summary(est_boot, display = "boot")
#> $boot
#>        N Bootstrap      2.5%    97.5%  Std.Dev
#>  [1,]  3  11.87407  8.425000 15.51852 2.011130
#>  [2,]  4  13.14934  9.771387 16.77627 2.001598
#>  [3,]  5  13.83898 10.822584 17.43899 1.964793
#>  [4,]  6  14.66163 10.705144 18.37757 1.977243
#>  [5,]  7  15.47596 11.981076 18.68644 1.879223
#>  [6,]  8  16.18653 12.599602 19.68927 1.830489
#>  [7,]  9  16.61378 13.042764 19.61772 1.779771
#>  [8,] 10  17.19346 13.077245 19.70960 1.788670
#>  [9,] 11  17.82230 14.315556 19.81162 1.558281
```

```
#> [10,] 12  18.35152 15.374464 19.58721 1.270158
#> [11,] 13  18.75381 16.570376 19.59107 1.027014
#> [12,] 14  19.27832 19.278321 19.27832 0.000000
#>
#> attr(,"class")
#> [1] "summary.poolaccum"
```

Visualizar os resultados com intervalo de confiança de 95% (Figura 11.6).

```
## Preparando os dados para fazer o gráfico
resultados_boot <- summary(est_boot, display = c("S", "boot"))
res_boot <- cbind(resultados_boot$boot[,1:4], resultados_boot$S[,2:4])
res_boot <- as.data.frame(res_boot)
colnames(res_boot) <- c("Amostras", "BOOT", "BOOT_inferior",
                        "BOOT_superior", "Riqueza", "R_inferior",
                        "R_superior")

## Gráfico
ggplot(res_boot, aes(y = Riqueza, x = Amostras)) +
    geom_point(aes(y = BOOT, x = Amostras + 0.1), size = 4,
               color = "darkorange", alpha = 0.7) +
    geom_point(aes(y = Riqueza, x = Amostras), size = 4,
               color = "cyan4", alpha = 0.7) +
    geom_point(y = 10.4, x = 9, size = 4, color = "darkorange",
               alpha = 0.7) +
    geom_point(y = 9.3, x = 9, size = 4, color = "cyan4", alpha = 0.7) +
    geom_label(y = 10.4, x = 12.3,
               label = "Riqueza estimada - Bootstrap") +
    geom_label(y = 9.3, x = 11.5, label = "Riqueza observada") +
    geom_line(aes(y = BOOT, x = Amostras), color = "darkorange") +
    geom_line(aes(y = Riqueza, x = Amostras), color = "cyan4") +
    geom_linerange(aes(ymin = BOOT_inferior, ymax = BOOT_superior,
                       x = Amostras + 0.1), color = "darkorange") +
    geom_linerange(aes(ymin = R_inferior, ymax = R_superior,
                       x = Amostras), color = "cyan4") +
    scale_x_continuous(limits = c(1, 15), breaks = seq(1, 15, 1)) +
    labs(x = "Número de amostras", y = "Riqueza estimada - Bootstrap") +
    tema_livro()
```

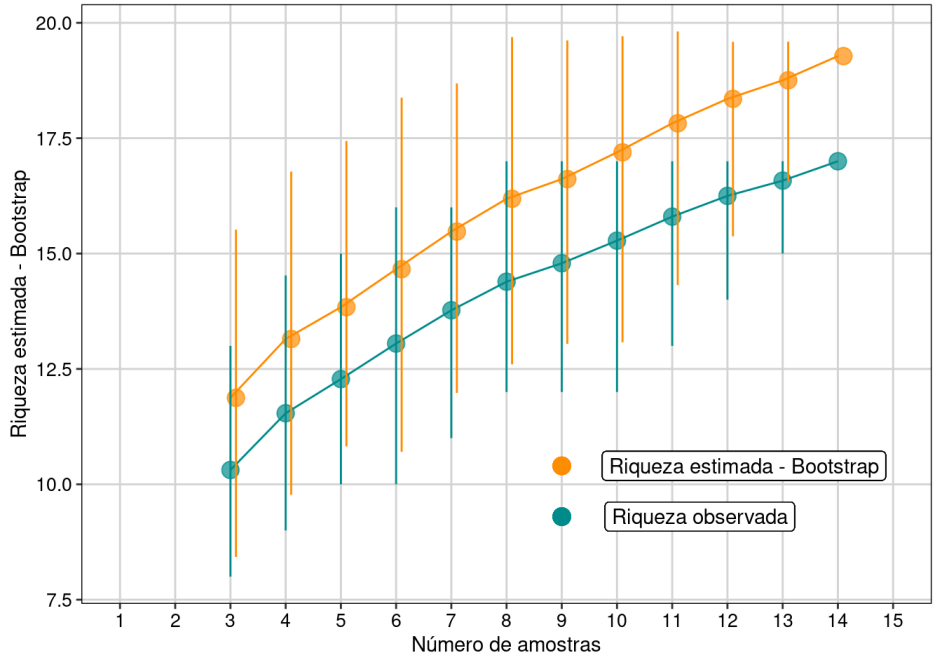


Figura 11.6: Resultados com intervalo de confiança de 95% para o estimador Bootstrap.

Interpretação dos resultados

Com base na frequência de ocorrência das espécies, o estimador bootstrap estimou a possibilidade de encontrarmos mais duas espécies, caso o esforço amostral fosse maior e não estimou tendência de estabilização da curva em uma assíntota.

11.4 Interpolação e extrapolação baseadas em rarefação usando amostragens de incidência ou abundância

Este método utiliza teoria de amostragem (e.g., modelos multinomial, Poisson e Bernoulli) para conectar rarefação (interpolação) e predição (extrapolação) com base no tamanho da amostra (Chao & Jost 2012, Colwell et al. 2012). Ele utiliza uma abordagem com bootstrap para calcular o intervalo de confiança de 95%.

> **Importante**
> A extrapolação torna-se altamente incerta quando estendida para o dobro ou mais do tamanho da amostragem.

Exemplo prático 1

Explicação dos dados

Usaremos os mesmos dados hipotéticos de 17 espécies de anuros amostradas em 14 dias de coletas de campo em um habitat reprodutivo localizado na região Noroeste do Estado de São Paulo, Brasil.

Análise

Cálculo da extrapolação da riqueza com base no número de indivíduos (Figura 11.7).

```
## Preparando os dados para análises considerando a abundância
dados_inext_abu <- colSums(dados_coleta)
resultados_abundancia <- iNEXT(dados_inext_abu, q = 0,
                               datatype = "abundance",
                               endpoint = 600)

## Gráfico
ggiNEXT(resultados_abundancia, type = 1) +
    scale_linetype_discrete(labels = c("Interpolado", "Extrapolado")) +
    scale_colour_manual(values = "darkorange") +
    scale_fill_manual(values = "darkorange") +
    labs(x = "Número de indivíduos", y = " Riqueza de espécies") +
    tema_livro()
```

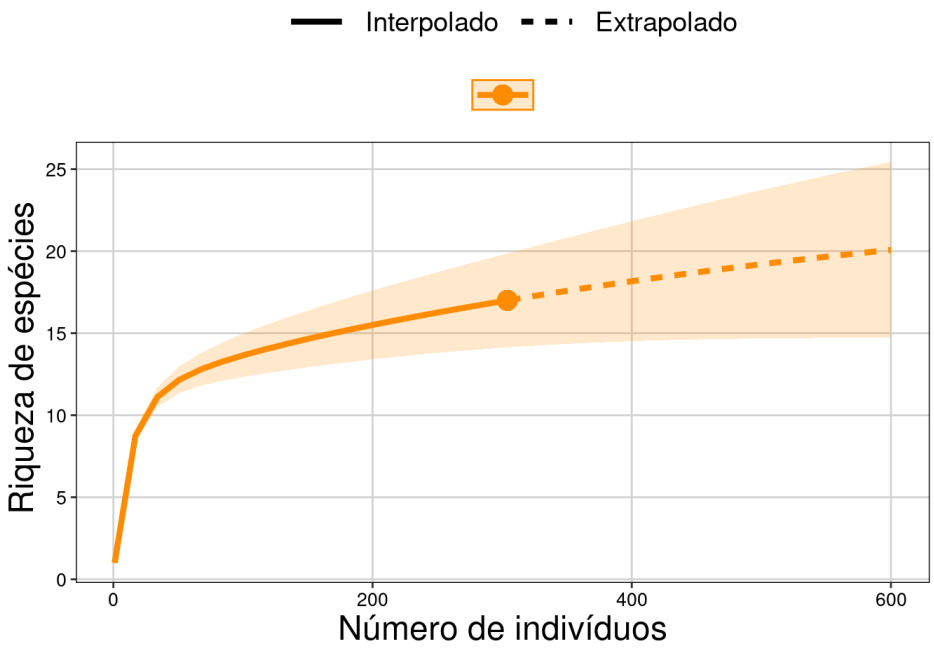


Figura 11.7: Resultados com intervalo de confiança de 95% para a extrapolação.

Interpretação dos resultados

Veja que o ponto no final da linha contínua representa as 17 espécies de anuros (eixo Y) observadas entre os 304 indivíduos (eixo X). A extrapolação máxima (600 indivíduos no nosso exemplo) estima um aumento de até oito espécies (intervalo de confiança) caso amostrássemos mais 296 indivíduos.

Cálculo da extrapolação da riqueza com base no número de amostras (Figura 11.8).

```
## Preparando os dados para análises considerando a incidência
# Precisa transpor o data frame.
dados_inext <- as.incfreq(t(dados_coleta))
resultados_incidencia <- iNEXT(dados_inext, q = 0,
```

```
                                datatype = "incidence_freq",
                                endpoint = 28)

## Gráfico
ggiNEXT(resultados_incidencia, type = 1) +
    scale_linetype_discrete(labels = c("Interpolado", "Extrapolado")) +
    scale_colour_manual(values = "darkorange") +
    scale_fill_manual(values = "darkorange") +
    labs(x = "Número de amostras", y = " Riqueza de espécies") +
    tema_livro()
```

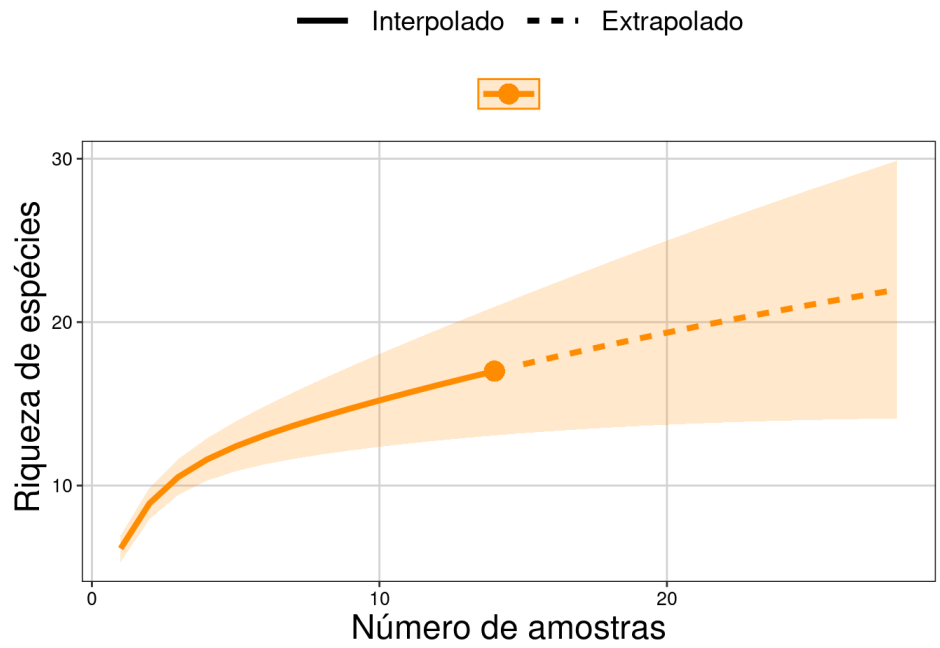


Figura 11.8: Resultados com intervalo de confiança de 95% para extrapolação da riqueza.

Interpretação dos resultados

Veja que o ponto no final da linha contínua representa as 17 espécies de anuros (eixo Y) observadas nos 14 dias de coleta (eixo X - amostras). A extrapolação máxima (28 dias de coleta no nosso exemplo) estima um aumento de até 12 espécies (intervalo de confiança) caso amostrássemos mais 14 dias.

Exemplo prático 2

Explicação dos dados

Neste exemplo, iremos refazer o exemplo do Capítulo 7 que usa *Generalized Least Squares* (GLS) para testar a relação da riqueza de ácaros com a quantidade de água no substrato. Contudo, ao invés de considerar a riqueza de espécies de ácaros observada como variável resposta, iremos utilizar a riqueza extrapolada. Os dados que usaremos estão disponíveis no pacote `vegan` e representam a composição de espécies de ácaros amostradas em 70 comunidades/amostras.

Pergunta

- A riqueza extrapolada de espécies de ácaros é maior em comunidades localizadas em áreas com substratos secos?

Predições

- O número de espécies extrapolada será maior em substratos secos uma vez que as limitações fisiológicas impostas pela umidade limitam a ocorrência de várias espécies de ácaros

Variáveis

- Matriz ou data frame com as abundâncias das espécies de ácaros (variável resposta) registradas em 70 comunidades/amostras (variável preditora)

Checklist

- Verificar se a sua matriz ou data frame estão com as espécies nas linhas e as comunidades nas colunas

Análise

Vamos iniciar calculando a riqueza extrapolada com base na comunidade com maior abundância.

```
## Transposição
# Os dados estão com as comunidades nas colunas e as espécies nas linhas.
# Para as análises teremos que transpor a planilha.
composicao_acaros <- t(mite)

## Dados
# A comunidade com maior abundância tem 781 indivíduos.
max(colSums(composicao_acaros))
#> [1] 781

# Calcular a riqueza extrapolada de espécies para todas as comunidades
# considerando a maior abundância.
resultados_extrapolacao <- iNEXT(composicao_acaros, q = 0,
                                 datatype = "abundance",
                                 endpoint = 781)
```

Lembrando, estamos interessados no valor rarefeito considerando a abundância total de 781 indivíduos. Esta informação está armazenada na linha 40 e na coluna 4 dos data frames salvos no objeto `resultados_extrapolacao$iNextEst`. Assim, para obtermos o valor rarefeito de interesse, vamos criar um `loop for` para facilitar a extração da riqueza rarefeita para as 70 comunidades.

```
## Dados
# Criando um data.frame vazio para salvar os dados
resultados_comunidades_ext <- data.frame()

# Criando um vetor vazio para salvar os resultados
riqueza_extrapolada <- c()
```

```
## Loop
# Loop repetindo as análises para as 70 comunidades
# O objetivo é extrair a riqueza estimada extrapolada para 781 indivíduos
```

```
for (i in 1:70){
    resultados_comunidades_ext <- data.frame(
                                    resultados_extrapolacao$iNextEst[i])
    riqueza_extrapolada[i] <- resultados_comunidades_ext[40, 4]
}
```

Agora, seguindo os passos descritos no Capítulo 7, vamos identificar o melhor modelo que representa a estrutura espacial dos dados extrapolados.

```
## Dados
# Criando data frame com todas as variáveis
dados_combinado_ext <- data.frame(riqueza_extrapolada, agua, coord)

## Modelo gls sem estrutura espacial
no_spat_gls <- gls(riqueza_extrapolada ~ agua,
                   data = dados_combinado_ext,
                   method = "REML")

## Covariância esférica
espher_model <- gls(riqueza_extrapolada ~ agua,
                    data = dados_combinado_ext,
                    corSpher(form = ~lat + long, nugget = TRUE))

## Covariância exponencial
expon_model <- gls(riqueza_extrapolada ~ agua,
                   data = dados_combinado_ext,
                   corExp(form = ~lat + long, nugget = TRUE))

## Covariância Gaussiana
gauss_model <- gls(riqueza_extrapolada ~ agua,
                   data = dados_combinado_ext,
                   corGaus(form = ~lat + long, nugget = TRUE))

## Covariância linear
cor_linear_model <- gls(riqueza_extrapolada ~ agua,
                        data = dados_combinado_ext,
                        corLin(form = ~lat + long, nugget = TRUE))

## Covariância razão quadrática
ratio_model <- gls(riqueza_extrapolada ~ agua,
                   data = dados_combinado_ext,
                   corRatio(form = ~lat + long, nugget = TRUE))
```

Vamos usar a seleção de modelos por AIC para selecionar o modelo mais "provável" explicando a distribuição da riqueza extrapolada das espécies de ácaros (Figura 11.9).

```
## Seleção dos modelos
aic_fit_ext <- AIC(no_spat_gls, espher_model,
                   cor_linear_model, expon_model,
                   gauss_model, ratio_model)
aic_fit_ext %>% arrange(AIC)
#>   df      AIC
#> 1  5 467.9349
#> 2  3 469.3103
#> 3  5 473.2373
#> 4  5 473.2815
#> 5  5 473.3086
#> 6  5 473.3103

## Visualizando os resíduos do modelo com menor valor de AIC (veja Capítulo
7)
plot(ratio_model)
```

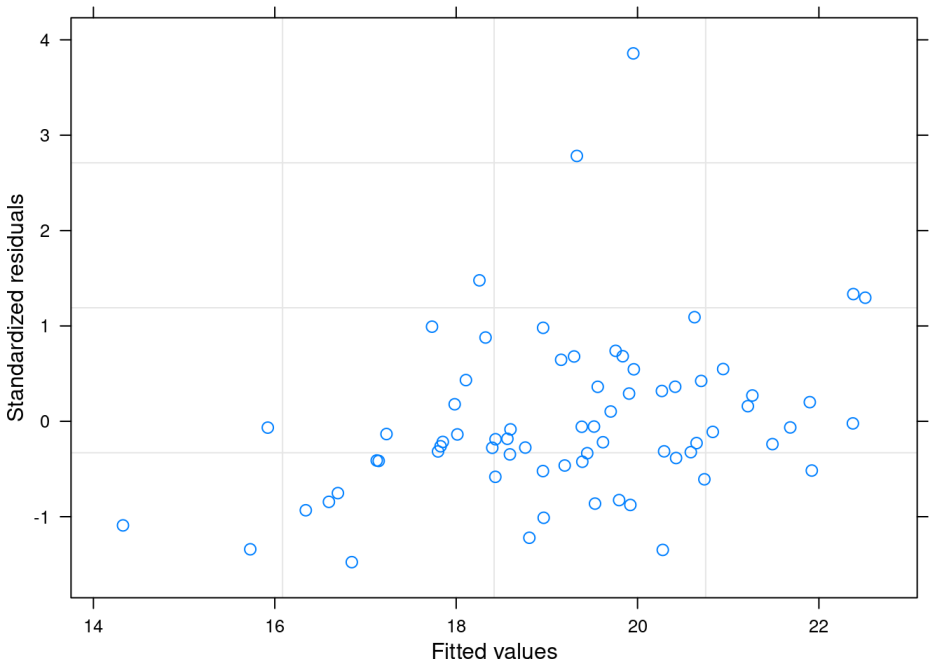


Figura 11.9: Visualização dos resíduos do modelo selecionado.

De forma geral, a distribuição dos resíduos está adequada com apenas dois pontos fugindo da nuvem. Contudo, eles podem influenciar os resultados (Figura 11.10).

```
## Visualizando os resultados e calculando pseudo-R-squared.
summary(ratio_model)$tTable
#>                  Value   Std.Error   t-value      p-value
#> (Intercept) 24.09577588 4.816461582  5.002796 4.227862e-06
#> agua        -0.01181425 0.006977381 -1.693221 9.499017e-02
rsquared(ratio_model)
#>             Response   family     link method  R.squared
#> 1 riqueza_extrapolada gaussian identity   none 0.05977552
```

```
## Gráfico
predito_ext <- predict(ratio_model)

ggplot(data = dados_combinado_ext,
       aes(x= agua, y= riqueza_extrapolada)) +
    geom_point(size = 4, shape = 21, fill = "darkorange", alpha = 0.7) +
    geom_line(aes(y = predito_ext), size = 1) +
    labs(x = "Concentração de água no substrato",
         y = "Riqueza extrapolada \ndas espécies de ácaros") +
    tema_livro()
```

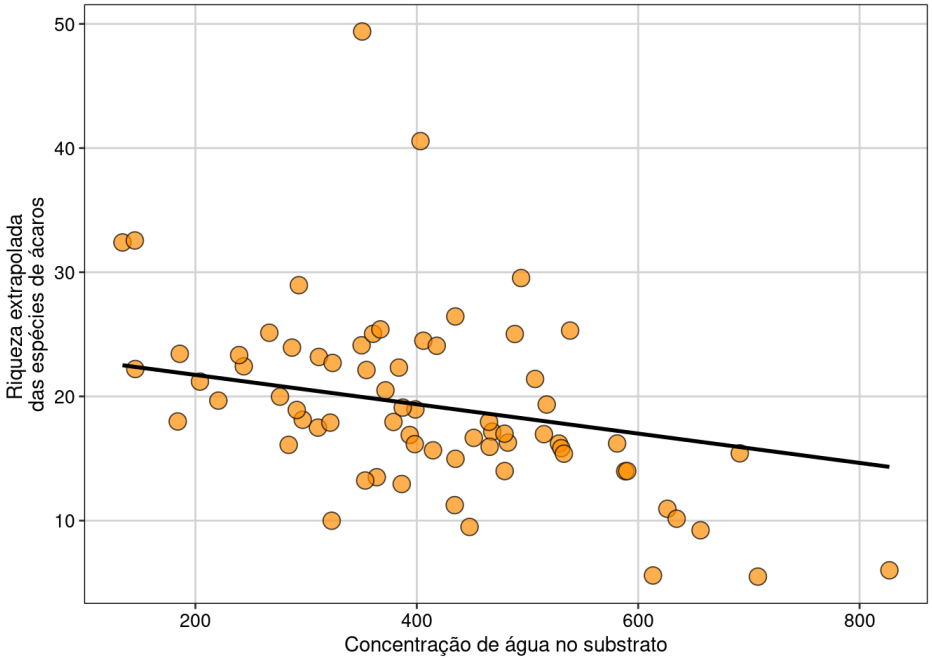


Figura 11.10: Gráfico do modelo GLS selecionado.

Interpretação dos resultados

A riqueza extrapolada das espécies de ácaros foi maior em comunidades localizadas em áreas com substratos secos do que em áreas com substratos úmidos. Contudo, apesar do modelo apresentar uma relação significativa entre as variáveis, a concentração de água explica apenas 5,9% da variação da riqueza extrapolada das espécies de ácaros. O padrão observado, valor de $P < 0.05$ e o baixo valor de R^2, provavelmente está relacionado com as duas comunidades com altos valores de riqueza extrapolada (e.g., *outilers*). Refaça as análises sem os dois pontos e vejam o padrão dos novos resultados.

11.5 Para se aprofundar

11.5.1 Livros

- Recomendamos o livro *Biological Diversity Frontiers in Measurement and Assessment* (Magurran & McGill 2011)

11.5.2 Links

- Recomendamos também que acessem a página do EstimateS software e baixem o manual do usuário que contém informações detalhadas sobre os índices de rarefação e estimadores de riqueza. Este site foi criado e é mantido pelo Dr. Robert K. Colwell, um dos maiores especialistas do mundo em estimativas da biodiversidade
- Recomendamos a página pessoal da pesquisadora Anne Chao que é uma das responsáveis pelo desenvolvimento da metodologia e do pacote iNEXT. Nesta página, vocês irão encontrar exemplos e explicações detalhadas sobre as análises

11.6 Exercícios

11.1 Carregue os dados - Cap11_exercicio1 - que está no pacote `ecodados`. Este conjunto de dados representa a abundância de 50 espécies de besouros coletados em 30 amostras. Calcule os estimadores de riqueza - Chao1 e ACE - e faça um gráfico contendo a riqueza observada e os dois estimadores de riqueza. Qual a sua interpretação sobre o esforço amostral?

11.2 Utilize o mesmo conjunto de dados do exercício anterior. Calcule os estimadores de riqueza - Jackknife 1 e bootstrap. Faça um gráfico contendo a riqueza observada e os dois estimadores de riqueza. Qual a sua interpretação sobre o esforço amostral? Compare com os resultados do exercício anterior que utilizam estimadores baseados na abundância das espécies.

11.3 Vamos refazer o exercício 10 do Capítulo 7 que usa *Generalized Least Squares* (GLS) para testar a relação da riqueza de anuros em 44 localidades na Mata Atlântica com a precipitação anual. Contudo, ao invés de considerar a riqueza de espécies de anuros observada como variável resposta, iremos utilizar a riqueza extrapolada. Utilize os dados `anuros_composicao` para estimar a riqueza extrapolada e os dados `anuros_ambientais` para acessar os dados de `precipitação anual` e `coordenadas geográficas`. Qual a sua interpretação dos resultados utilizando a riqueza observada e extrapolada?

Soluções dos exercícios.

Capítulo 12

Diversidade Taxonômica

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(devtools)
library(ecodados)
library(vegan)
library(ggplot2)
library(BiodiversityR)
library(hillR)
library(betapart)

## Dados
composicao_especies <- ecodados::composicao_anuros_div_taxonomica
precipitacao        <- ecodados::precipitacao_div_taxonomica
```

12.1 Aspectos teóricos

A diversidade biológica é um conceito multifacetado que pode ser definido e analisado de diferentes maneiras (e.g., diversidade genética, taxonômica, funcional, filogenética, ecossistêmica, etc.) (Magurran & McGill 2011, Gotelli & Chao 2013). Whittaker (1960, 1972) particionou a diversidade em três componentes: i) **diversidade alfa** que é caracterizada pela diversidade dentro do habitat ou unidade amostral, ii) **diversidade beta** que é caracterizada pela variação na diversidade entre habitats ou unidades amostrais, e iii) **diversidade gama** que é caracterizada pela combinação da diversidade alfa e beta ou definida como a diversidade regional englobando todos os habitat ou unidades amostrais. Portanto, não existe um método que quantifique todos os parâmetros associados à diversidade biológica. Consequentemente, a escolha da métrica de diversidade dependerá: i) do objetivo do estudo, e ii) das informações disponíveis para o pesquisador.

Neste capítulo, iremos abordar a diversidade taxonômica que ignora a relação de parentesco entre as espécies (e.g., diversidade filogenética - Capítulo 13) e as diferentes funções que as espécies realizam no ecossistema (e.g., diversidade funcional - Capítulo 14). Na diversidade taxonômica, pesquisadores estão interessados na riqueza de espécies (e.g., número de espécies), na distribuição de abundância das espécies (e.g., fato que algumas espécies são comuns e outras raras) e/ou diversidade de espécies (e.g., índices que descrevem a relação entre a riqueza e a distribuição da abundância relativa das espécies) nas localidades.

12.2 Diversidade alfa

12.2.1 Riqueza de espécies ou número de espécies

Riqueza de espécies é uma métrica intuitiva e de fácil compreensão, uma vez que se refere ao número de espécies observadas em uma localidade. É importante ter em mente que a riqueza de espécies é

influenciada pelo esforço amostral e sua estimativa real é um imenso desafio (Magurran & McGill 2011). Comparações entre comunidades com diferenças no número de amostragens ou abundância das espécies devem ser realizadas por meio de rarefações (veja Capítulo 10), enquanto que o número de espécies não detectadas pode ser estimado pelos estimadores de riqueza (veja Capítulo 11). Embora raramente usados como alternativas à rarefação, existem alguns índices que calculam a riqueza de espécies ponderando a abundância total (i.e., tamanho da amostra) dentro de cada comunidade.

Esses índices são:

1. Índice de Margalef

$$D_{Mg} = \frac{S - 1}{ln(N)}$$

onde:

- S = o número de espécies na comunidade
- ln = logaritmo natural
- N = número total de indivíduos na comunidade
- D_{Mg} não tem um valor máximo e sua interpretação é comparativa, com valores maiores indicando maior riqueza de espécies

2. Índice de Menhinick

$$D_{Mn} = \frac{S}{\sqrt{N}}$$

onde:

- S = o número de espécies na comunidade
- N = número total de indivíduos na comunidade
- D_{Mn} não tem um valor máximo e sua interpretação é comparativa, com valores maiores indicando maior riqueza de espécies

Exemplo prático 1

Explicação dos dados

Neste exemplo, avaliaremos a riqueza de espécies de 10 comunidades. Os dados de ocorrência das espécies nas comunidades foram simulados para demonstrar as propriedades das métricas de diversidade taxonômicas. Utilizaremos este conjunto de dados para todos os exemplos deste capítulo.

Pergunta

- A variação espacial na riqueza de espécies nas comunidades está associada com a variação na precipitação?

Predições

- Os valores de riqueza de espécies serão maiores nas comunidades localizadas em regiões que recebem grande volume de precipitação do que em regiões mais secas

Variáveis

- Data frame com as comunidades (unidade amostral) nas linhas e as espécies (variável resposta) nas colunas
- Data frame com as comunidades (unidade amostral) nas linhas e precipitação anual (variável preditora) na coluna

Checklist

- Verificar se os data frames de composição de espécies e variáveis ambientais estão com as unidades amostrais nas linhas e variáveis preditores nas colunas
- Verificar se as comunidades nos data frames de composição de espécies e variáveis ambientais estão distribuídos na mesma sequência/ordem nos dois arquivos

Análise

Abaixo, demonstramos os códigos no R para determinar a riqueza de espécies para cada comunidade a partir dos dados de composição de espécies. Os dados estão disponíveis no pacote `ecodados`.

```
## Ver os dados das comunidades
head(composicao_especies)
#>       sp1 sp2 sp3 sp4 sp5 sp6 sp7 sp8 sp9 sp10
#> Com_1  10  10  10  10  10  10  10  10  10   10
#> Com_2  91   1   1   1   1   1   1   1   1    1
#> Com_3   1   3   6  25   1   0   0   0   0    0
#> Com_4   0   0   0   0   0  15  15  18  17   16
#> Com_5   0   9   0   6   0  11   0   2  12    0
#> Com_6   3   0   5   0  12   1   0  13  12    0
```

Vamos ver a riqueza de espécies para cada comunidade.

```
## Calculando a riqueza observada de espécies para cada comunidade
riqueza_sp <- specnumber(composicao_especies)
riqueza_sp
#>  Com_1  Com_2  Com_3  Com_4  Com_5  Com_6  Com_7  Com_8  Com_9 Com_10
#>     10     10      5      5      5      6      2      4      6      4
```

Vamos ver a abundância total de cada comunidade.

```
## Calculamos a abundância total para cada comunidade
abundancia <- apply(composicao_especies, 1, sum)
abundancia
#>  Com_1  Com_2  Com_3  Com_4  Com_5  Com_6  Com_7  Com_8  Com_9 Com_10
#>    100    100     36     81     40     46      4     20     15     11
```

Calculando o Índice de Margalef.

```
## Índice de Margalef
# A função round é para limitar o resultado para duas casas decimais.
Margalef <- round((riqueza_sp - 1)/log(abundancia), 2)
Margalef
```

```
#>  Com_1  Com_2  Com_3  Com_4  Com_5  Com_6  Com_7  Com_8  Com_9 Com_10
#>   1.95   1.95   1.12   0.91   1.08   1.31   0.72   1.00   1.85   1.25
```

Calculando o Índice de Menhinick.

```
## Índice de Menhinick
Menhinick <- round(riqueza_sp/sqrt(abundancia), 2)
Menhinick
#>  Com_1  Com_2  Com_3  Com_4  Com_5  Com_6  Com_7  Com_8  Com_9 Com_10
#>   1.00   1.00   0.83   0.56   0.79   0.88   1.00   0.89   1.55   1.21
```

Agora vamos analisar a relação entre a riqueza de espécies e a precipitação anual.

```
## Juntando todos os dados em um único data frame
dados <- data.frame(precipitacao$prec, riqueza_sp, Margalef, Menhinick)

## Renomenado as colunas
colnames(dados) <- c("Precipitacao", "Riqueza", "Margalef", "Menhinick")

## ANOVA
anova_riq <- lm(Riqueza ~ Precipitacao, data = dados)
anova(anova_riq)
#> Analysis of Variance Table
#>
#> Response: Riqueza
#>              Df Sum Sq Mean Sq F value  Pr(>F)
#> Precipitacao  1 30.622 30.6224  8.9156 0.01744 *
#> Residuals     8 27.478  3.4347
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Há uma relação positiva entre a riqueza de espécies e a precipitação anual ($F_{1,8}$ = 8,91, P = 0,01).

Analisar a relação entre o Índice de Margalef e a precipitação anual.

```
## ANOVA
anova_marg <- lm(Margalef ~ Precipitacao, data = dados)
anova(anova_marg)
#> Analysis of Variance Table
#>
#> Response: Margalef
#>              Df  Sum Sq Mean Sq F value Pr(>F)
#> Precipitacao  1 0.37865 0.37865  2.1201 0.1835
#> Residuals     8 1.42879 0.17860
```

Não há uma relação positiva entre o índice de Margalef e a precipitação anual ($F_{1,8}$ = 2,12, P = 0,18).

Agora vamos analisar a relação entre o índice de Menhinick e a precipitação anual.

```
## ANOVA
anova_menh <- lm(Menhinick ~ Precipitacao, data = dados)
anova(anova_menh)
```

```
#> Analysis of Variance Table
#>
#> Response: Menhinick
#>              Df  Sum Sq  Mean Sq F value Pr(>F)
#> Precipitacao  1 0.07626 0.076262  1.0992 0.3251
#> Residuals     8 0.55503 0.069378
```

Não há uma relação positiva entre o índice de Menhinick e a precipitação anual ($F_{1,8}$ = 1,09, P = 0,32).

Com base nos resultados, vamos plotar apenas o gráfico com os resultados da riqueza de espécies ao longo do gradiente de precipitação anual (Figura 12.1).

```
## Gráfico
ggplot(data = dados, aes(x= Precipitacao, y= Riqueza)) +
    labs(x = "Precipitação anual (mm)", y = "Riqueza de espécies") +
    geom_point(size = 4, shape = 21, fill = "darkorange", alpha = 0.7) +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    tema_livro()
```

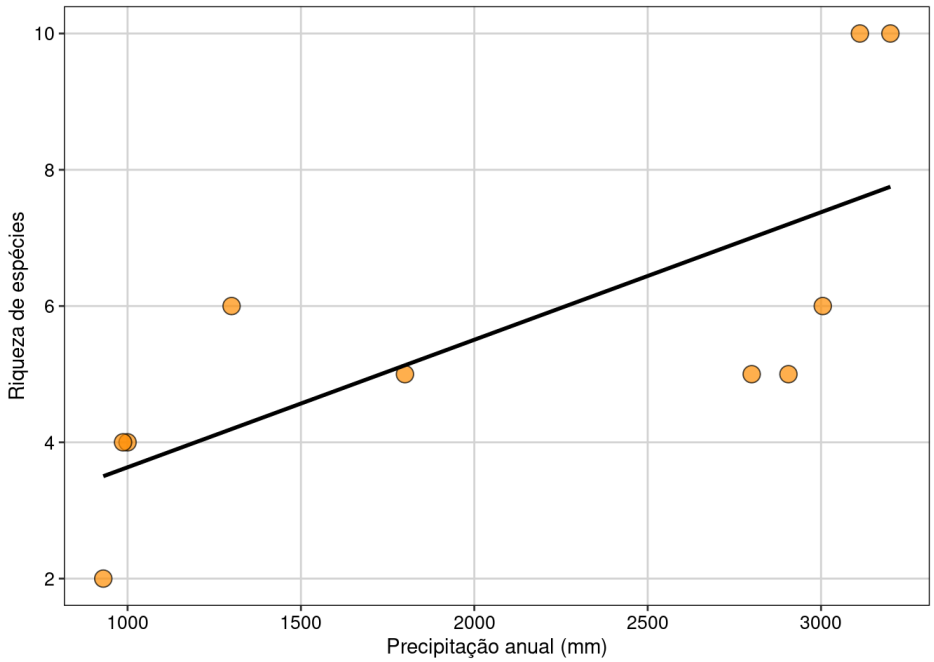


Figura 12.1: Gráfico do modelo linear da relação entre riqueza e precipitação.

Interpretação dos resultados

O número de espécies é maior em comunidades com maior precipitação. Contudo, quando ponderamos pela abundância (índices de Margalef ou Menhinick), a relação com a precipitação não é significativa.

> **Importante**
> Percebam que ponderar a riqueza de espécies pela abundância altera a interpretação dos resultados.

12.2.2 Diversidade de espécies

Diferente dos índices de riqueza de espécies que não levam em consideração a abundância relativa das espécies (i.e., todas as espécies tem o mesmo peso), os índices de diversidade avaliam além da riqueza, a dominância ou raridade das espécies nas comunidades. Assim, quando comparamos duas comunidades com a mesma riqueza de espécies, e uma das comunidades é dominada por uma única espécie e a outra comunidade apresenta espécies com abundâncias parecidas, consideramos a segunda comunidade mais diversa. Os índices de diversidade variam porque eles dão pesos diferentes para a riqueza e equitabilidade das espécies. Assim, um determinado índice de diversidade pode indicar que uma comunidade X é mais diversa que uma comunidade Y, enquanto outro índice indica o oposto (Melo 2008). Portanto, uma maneira de determinar qual índice de diversidade usar é saber se você quer dar maior peso para riqueza ou equitabilidade das espécies nas comunidades.

> **Importante**
> Ressaltamos que há várias críticas em relação ao uso dos índices de diversidade que são abstratos e difíceis de se interpretar (Hurlbert 1971). Por exemplo, dizer que o valor X estimado por índices de diversidade é alto ou baixo é irrelevante se não tivermos uma base comparativa (para mais detalhes veja Melo (2008)).

Os dois índices de diversidade mais usados em Ecologia são:

1. Índice de Shannon-Wiener

Quantifica a incerteza associada em predizer a identidade de uma espécie dado o número de espécies e a distribuição de abundância para cada espécie. Este índice é mais sensível às mudanças nas espécies raras da comunidade.

$$H' = -\sum_{i=1}^{S} p_i * lnp_i$$

onde:

- p_i = abundância relativa de cada espécie, calculada pela proporção dos indivíduos de uma espécie pelo número total dos indivíduos na comunidade
- ln = logaritmo natural, mas outras bases logarítmicas podem ser utilizadas
- H' = não tem um valor máximo e sua interpretação é comparativa, com valores maiores indicando maior diversidade

2. Índice de Simpson

Quantifica a probabilidade de dois indivíduos retirados ao acaso da comunidade pertencerem à mesma espécie. Este índice é na verdade uma medida de dominância. Assim como a probabilidade dos indivíduos serem da mesma espécie diminui com o aumento da riqueza de espécies, o índice de Simpson também diminui com a riqueza.

$$D = \sum_{i=1}^{S} p_i^2$$

onde:

- P_i = abundância relativa de cada espécie, calculada pela proporção dos indivíduos de uma espécie pelo número total dos indivíduos na comunidade
- D = varia de 0 a 1, com valores próximos de 1 indicando menor diversidade enquanto valores próximos de 0 indicam maior diversidade. Para evitar confusão nas interpretações, normalmente o índice de Simpson é expressado como o valor inverso (1 - D) para que os maiores valores representem maior diversidade. Neste caso, o valor inverso é conhecido na literatura como índice Gini-Simpson. Para o índice Gini-Simpson estamos avaliando a probabilidade de dois indivíduos retirados ao acaso da comunidade sejam de espécies diferentes.

Exemplo prático 2

Explicação dos dados

Usaremos os mesmos dados simulados do exemplo prático 1.

Pergunta

- A variação espacial na diversidade de espécies das comunidades está associada com o gradiente de precipitação?

Predições

- Os valores de diversidade de espécies serão maiores nas comunidades localizadas em regiões maior volume de precipitação do que em regiões mais secas

Análise

Abaixo demonstramos os códigos no R para determinar a diversidade de espécies para cada comunidade a partir da planilha de composição de espécies.

```
## Índice de Shannon
# MARGIN = 1 significa que a função irá calcular o índice considerando
# as linhas do data.frame (comunidades).
shannon_res <- diversity(composicao_especies, index = "shannon",
                         MARGIN = 1)
shannon_res
#>     Com_1     Com_2     Com_3     Com_4     Com_5     Com_6     Com_7
Com_8     Com_9    Com_10
#> 2.3025851 0.5002880 0.9580109 1.6068659 1.4861894 1.5607038 0.6931472
1.1058899 1.7140875 1.2636544
```

O argumento `index = "simpson"`, calcula o índice Gini-Simpson (1-D).

```
## Índice de Simpson
simpson_res <- diversity(composicao_especies, index = "simpson",
                         MARGIN = 1)
simpson_res
#>     Com_1     Com_2     Com_3     Com_4     Com_5     Com_6     Com_7
Com_8     Com_9    Com_10
```

```
#> 0.9000000 0.1710000 0.4814815 0.7989636 0.7587500 0.7674858 0.5000000
0.5850000 0.8088889 0.6942149
```

Interpretação dos resultados

A comunidade 1 foi a comunidade que apresentou a maior diversidade de espécies (Shannon-Wiener = `2,3` e Gini-Simpson = `0.9`), enquanto a comunidade 2 foi a comunidade que apresentou a menor diversidade (Shannon-Wiener = `0,5` e Gini-Simpson = `0,17`). Gostaríamos de chamar a atenção para a importância da distribuição da abundância relativa das espécies dentro das comunidades. Percebam que tanto a comunidade 1 quanto a comunidade 2 abrigam o mesmo número de espécies (10 espécies) e abundância total (100 indivíduos), mas os padrões de distribuição da abundância relativa entre as espécies dentro das comunidades são bem discrepantes. Na comunidade 1 as espécies apresentam abundâncias semelhantes entre elas (i.e., alta equitabilidade), enquanto na comunidade 2 uma espécie é dominante e as outras raras (i.e., baixa equitabilidade). Essa diferença na distribuição da abundância relativa entre as comunidades é um fator muito importante para os índices de diversidade.

Dentro desta perspectiva, alguns índices fornecem uma estimativa sobre a equitabilidade da distribuição da abundância nas comunidades. Entre eles, o mais conhecido foi proposto por Pielou (1966).

Índice de Equabilidade (ou Equitabilidade) de Pielou

É uma métrica derivada do índice de Shannon-Wiener que descreve o padrão de distribuição da abundância relativa das espécies na comunidade.

$$J = \frac{H'}{Hmax} = \frac{H'}{ln(S)}$$

onde:

- H' = índice de Shannon-Wiener
- H_{max} = todas as espécies teriam a mesma abundância relativa
- H_{max} é calculado aplicando o logaritmo natural (ln) para a riqueza de espécies (S)
- Se todas as espécies apresentam a mesma abundância relativa, então J = 1. Se uma espécie apresenta forte dominância, J aproxima-se de zero

Não há uma função no R que calcule o índice de Pielou, mas ele pode facilmente ser calculado usando os valores de diversidade de Shannon-Wiener e o logaritmo da riqueza de espécies de cada comunidade.

```
## Índice de Pielou
Pielou <- shannon_res/log(specnumber(composicao_especies))
```

Agora que temos uma ideia de como a riqueza de espécies e a distribuição da abundância relativa são importantes para quantificar os valores dos índices de diversidade, vamos testar se há alguma relação entre os índices de diversidade e precipitação anual nas comunidades.

```
## Juntando todos os dados em um único data frame
dados_div <- data.frame(precipitacao$prec, shannon_res,
                        simpson_res, Pielou)

## Renomeando as colunas
colnames(dados_div) <- c("Precipitacao", "Shannon", "Simpson", "Pielou")
```

Vamos realizar uma regressão simples para verificar a relação entre o índice de Shannon-Wiener e a precipitação anual nas comunidades.

```
## ANOVA
anova_shan <- lm(Shannon ~ Precipitacao, data = dados_div)
anova(anova_shan)
#> Analysis of Variance Table
#>
#> Response: Shannon
#>              Df  Sum Sq Mean Sq F value Pr(>F)
#> Precipitacao  1 0.10989 0.10989  0.3627 0.5637
#> Residuals     8 2.42366 0.30296
```

Agora, vamos realizar uma regressão simples para verificar a relação entre o índice de Simpson e a precipitação anual nas comunidades.

```
## ANOVA
anova_simp <- lm(Simpson ~ Precipitacao, data = dados_div)
anova(anova_simp)
#> Analysis of Variance Table
#>
#> Response: Simpson
#>              Df  Sum Sq  Mean Sq F value Pr(>F)
#> Precipitacao  1 0.00132 0.001325  0.0252 0.8778
#> Residuals     8 0.42064 0.052580
```

Por fim, vamos fazer regressão simples para verificar a relação entre o índice de Pielou e a precipitação anual nas comunidades.

```
## ANOVA
anova_piel <- lm(Pielou ~ Precipitacao, data = dados_div)
anova(anova_piel)
#> Analysis of Variance Table
#>
#> Response: Pielou
#>              Df  Sum Sq  Mean Sq F value Pr(>F)
#> Precipitacao  1 0.09080 0.090798  1.5792 0.2443
#> Residuals     8 0.45997 0.057496
```

Importante

As análises acima são apenas ilustrativas. Não estamos avaliando as premissas de normalidade e homogeneidade da variância dos resíduos (veja Capítulo 7). Além disso, é importante estar ciente das críticas e limitações de usar índices de Shannon e Simpson como nesses exemplos (Jost 2007).

Interpretação dos resultados

A variação espacial na diversidade de espécies, obtida através dos índices de Shannon-Wiener e Simpson, e a equitabilidade de Pielou não foram associados com a variação na precipitação anual entre as áreas (P > 0,05).

12.2.3 Diagramas de Whittaker ou Curva de Dominância

Embora os índices de diversidade de espécies englobem os componentes de riqueza e abundância relativa das espécies nas suas estimativas, não é possível conhecer o número de espécies ou quais são as espécies dominantes ou raras dentro das comunidades. Por exemplo, duas comunidades podem ter o mesmo valor de diversidade e ainda assim apresentarem diferenças na riqueza e equitabilidade (Melo 2008). O **Diagrama de Whittaker** é um método que lida com essas questões utilizando informações visuais do número de espécies e abundância relativa de cada espécie nas comunidades. Este método plota as espécies ranqueadas no eixo X da mais abundante para a menos abundante, enquanto no eixo Y as abundâncias relativas das espécies são plotadas em escala logarítmica ($\log_{10}$). Este gráfico permite ao leitor reconhecer: i) a riqueza de espécies observando o eixo X, ii) a equitabilidade da abundância relativa das espécies pela inclinação da reta, e iii) quais são as espécies dominantes, intermediárias e raras nas comunidades através da observação em relação ao eixo Y. A partir destas curvas, vários autores propuseram modelos matemáticos para explicar a distribuição de abundância das espécies gerando diferentes modelos teóricos (e.g., série geométrica, *broken-stick*, log-series e log-normal). Cada modelo possui predições distintas: o modelo geométrico prediz distribuição de abundâncias desiguais, *broken-stick* prediz distribuição de abundâncias uniformes, enquanto log-normal e log-series são intermediárias com predições distintas sobre as proporções de espécies raras - alta em log-series, baixa em log-normal (veja McGill et al. (2007) para revisão).

Para análises exploratórias onde você tem interesse em visualizar o padrão da distribuição relativa das espécies por comunidade, a função `rankabundance` do pacote `BiodiversityR` é uma opção interessante (Figura 12.2).

```
## Cálculo da curva para as comunidades 2 e 3
rank_com2 <- rankabundance(composicao_especies[2,
                           composicao_especies[2,] > 0])
rank_com3 <- rankabundance(composicao_especies[3,
                           composicao_especies[3,] > 0])

## Gráfico
# Veja a ajuda da função rankabundplot para outros exemplos de gráficos.
rankabunplot(rank_com2, scale = "logabun", specnames = c(1),
             pch = 19, col = "darkorange")
rankabunplot(rank_com3, scale = "logabun", specnames = c(1), pch = 19,
             xlim = c(0,10), addit = TRUE, col = "cyan4" , legend = TRUE)
legend(5, 40, legend = c("Comunidade 2", "Comunidade 3"),
       col = c("darkorange", "cyan4"), lty = 1, cex = 0.8, box.lty = 0)
```

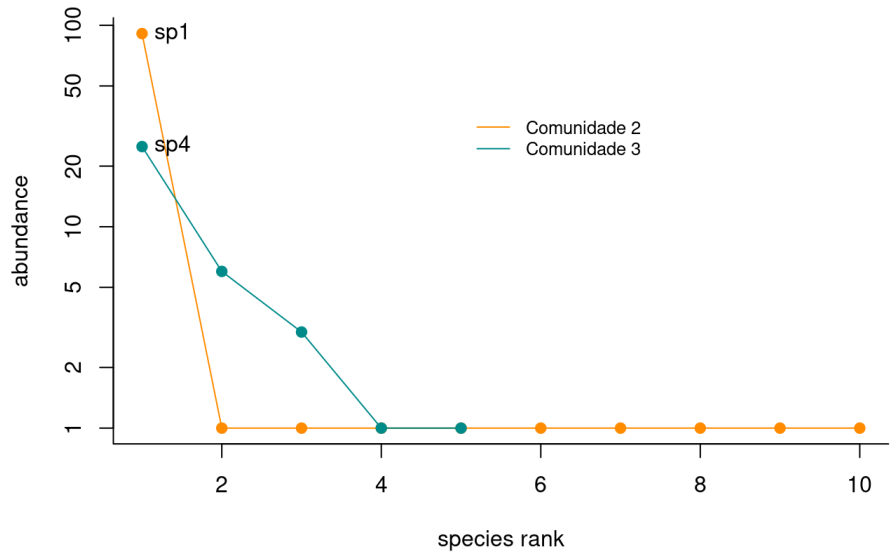


Figura 12.2: Diagramas de Whittaker para duas comunidades.

Interpretação dos resultados

Percebam que olhando os eixos do gráfico conseguimos determinar que a Comunidade 2 (círculo laranja) abriga 10 espécies no total (i.e., comprimento do eixo X), com a espécie sp1 apresentando alta dominância e as outras espécies apresentando abundâncias muito baixas. A Comunidade 3 (círculo ciano) abriga cinco espécies no total, sendo que a espécie `sp4` apresenta alta dominância, duas espécies apresentam abundâncias intermediárias e outras duas abundâncias baixas.

12.2.4 Curvas de distribuição de abundâncias

Caso o interesse seja avaliar qual dos modelos teóricos melhor explica a distribuição das abundâncias das espécies, a função `radift()` do pacote `vegan` é a melhor opção.

A função `radfit` avalia cinco modelos teóricos para determinar qual deles melhor se ajustam aos dados. Os modelos teóricos avaliados na função são:

- Null = modelo broken-stick
- preemption = série geométrica
- log-normal
- Zipf
- Zipf-Mandelbrot

Você pode realizar as análises separadamente para cada comunidade ou para todas as comunidades ao mesmo tempo.

Vamos começar avaliando separadamente a Comunidade 2.

```
## Teste das curvas de distribuição de abundâncias
curvas_dominancia_com2 <- radfit(composicao_especies[2,])
curvas_dominancia_com2
#>
#> RAD models, family poisson
#> No. of species 10, total abundance 100
```

```
#>
#>             par1     par2    par3        Deviance AIC     BIC
#> Null                                    175.242  199.592 199.592
#> Preemption  0.68962                      79.560  105.910 106.213
#> Lognormal  -0.65366  3.2485              47.350   75.701  76.306
#> Zipf        0.83829 -3.0254              26.612   54.963  55.568
#> Mandelbrot  0.83829 -3.0254  1.6448e-07  26.612   56.963  57.871
```

Agora vamos fazer um gráfico com as predições dos modelos (Figura 12.3).

```
plot(curvas_dominancia_com2,
     ylab = "Abundância",
     xlab = "Ranqueamento das espécies")
```

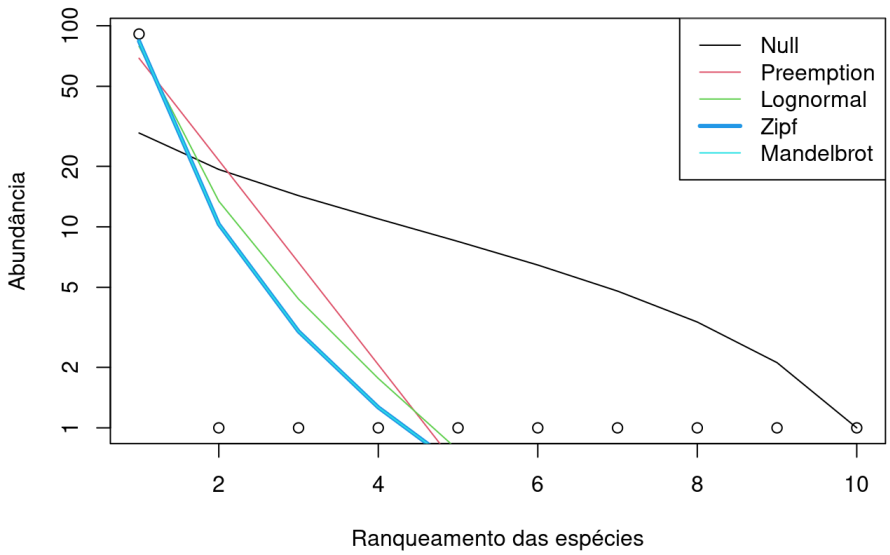


Figura 12.3: Teste de cinco Diagramas de Whittaker para a Comunidade 2.

Interpretação dos resultados

Os pontos brancos representam as espécies ranqueadas de acordo com a abundância e as linhas representam as predições dos modelos matemáticos. Com base nos valores de AIC (veja Capítulo 7), o modelo **Zipf** é o melhor modelo que explica a distribuição da abundância relativa das espécies na Comunidade 2.

Agora vamos analisar os dados considerando todas as comunidades (Figura 12.4).

```
## Teste das curvas de distribuição de abundâncias
curvas_dominancia_todas <- radfit(composicao_especies)
curvas_dominancia_todas

# Vamos fazer um gráfico para cada comunidade
plot(curvas_dominancia_todas, log = "y")
```

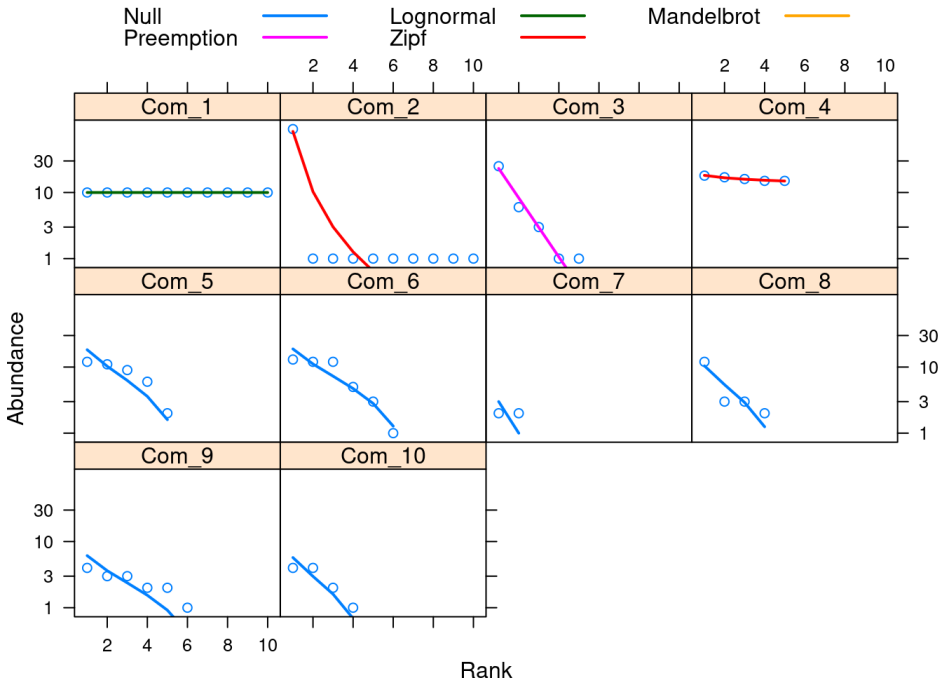


Figura 12.4: Testes de cinco Diagramas de Whittaker para todas as comunidades.

Interpretação dos resultados

A Comunidade 1 foi associada com o modelo **log-normal**, as Comunidades 2 e 4 com o modelo **Zipf**, a Comunidade 3 com o modelo **série geométrica** e as outras comunidades com o **modelo nulo**. Para explorar a explicação biológica por trás destes modelos veja os artigos (Wilson 1991, McGill et al. 2007, Magurran & McGill 2011).

> **Importante**
> A ligação entre o modelo matemático e a explicação biológica precisa ser interpretada com cuidado porque diferentes modelos matemáticos podem levar ao mesmo padrão de distribuição de abundância.

12.2.5 Números de Hill ou Série de Hill

Embora os índices de Shannon-Wiener e Gini-Simpson sejam amplamente usados em estudos ecológicos e de conservação, eles sofrem de propriedades matemáticas e não representam a diversidade propriamente dita (Jost 2006). Portanto, quando o objetivo é avaliar a diversidade, os índices de Shannon-Wiener e Gini-Simpson não deveriam ser utilizados na sua forma padrão, mas transformados em números efetivos de espécies ou diversidade verdadeira (Jost 2006). O número efetivo de espécies é o número de espécies igualmente abundantes (i.e., todas as espécies com a mesma abundância) necessárias para produzir o valor observado para um determinado índice. Por exemplo, uma comunidade com índice de Shannon-Wiener estimado de 4,5 teria um número efetivo de 90 espécies igualmente abundantes. Jost et al. (2006) usam o seguinte exemplo para explicar o conceito do número efetivo de espécies - uma comunidade com 16 espécies igualmente abundantes é duas vezes mais diversa do que uma comunidade com 8 espécies igualmente abundantes. Neste caso, a diversidade deveria ser proporcional ao número de espécies. Contudo, quando aplicamos os índices

de diversidade para estas comunidades com 16 e 8 espécies (cada espécie com 5 indivíduos), o índice de Shannon-Wiener é 2,772 e 2,079, respectivamente, e o índice de Gini-Simpson é 0,937 e 0,875, respectivamente. Claramente, os valores estimados pelos índices de diversidade não representam a diferença entre as comunidades porque eles carecem de uma particularidade matemática conhecida como propriedade de duplicação.

O próximo exemplo (modificado do website de Lou Jost; http://www.loujost.com/), demostra a importância da transformação dos índices de diversidade em números efetivos de espécies. Imagine que você foi contratado para avaliar a diversidade de peixes em um riacho antes e depois da instalação de uma usina hidrelétrica. Suponha que os valores estimados pelo índice de Gini-Simpson foi de 0,99 antes da instalação e de 0,97 depois da instalação. A princípio, você poderia concluir que a diversidade diminuiu somente 2% e que a instalação da hidrelétrica não afetou a diversidade de peixes no riacho. Contudo, transformando os valores do índice de diversidade em números efetivos, percebemos que antes da instalação a diversidade do riacho equivale a 100 espécies igualmente abundantes, enquanto após a instalação, a diversidade equivale a 33 espécies igualmente abundantes. Portanto, a queda da diversidade foi de 66% e não 2%.

Hill (1973) derivou uma equação geral para o cálculo do número efetivo de espécies ou diversidade verdadeira que depende apenas do valor de q e da abundância relativa das espécies.

$$ {}^{q}D = \left(\sum_{i=1}^{S} p_i^q \right)^{1/(1-q)} $$

Onde:

- q = é um parâmetro conhecido como ordem da diversidade e é usado para dar peso às espécies comuns ou raras. `q = 0` não considera a frequência das espécies e representa a riqueza observada de espécies, `q = 1` equivale a transformação do índice de Shannon-Wiener (i.e., `exp(H')`) e atribui pesos às espécies com base na proporção das suas frequências e, `q = 2` equivale à transformação do índice de Gini-Simpson (i.e., `1/(1-D)`) e atribui peso às espécies mais comuns. Valores de `q <1` favorecem espécies raras, enquanto valores de `q > 1` favorecem espécies comuns.
- p_i = abundância relativa de cada espécie, calculada pela proporção dos indivíduos de uma espécie pelo número total dos indivíduos na comunidade

Vamos calcular o Número de Hill para as comunidades do nosso exemplo.

Calculando o Número de Hill com `q = 0`.

```
## Número de Hill para q = 0
hill_res_q_0 <- hill_taxa(composicao_especies, q  = 0)
hill_res_q_0
#>  Com_1  Com_2  Com_3  Com_4  Com_5  Com_6  Com_7  Com_8  Com_9 Com_10
#>     10     10      5      5      5      6      2      4      6      4
```

Calculando o Número de Hill com `q = 1`.

```
## Número de Hill para q = 1
hill_res_q_1 <- hill_taxa(composicao_especies, q  = 1)
hill_res_q_1
#>     Com_1     Com_2     Com_3     Com_4     Com_5     Com_6     Com_7
```

```
Com_8     Com_9    Com_10
#> 10.000000  1.649196  2.606507  4.987156  4.420220  4.762172  2.000000
3.021912  5.551608  3.538328
```

Calculando o Número de Hill com `q = 2`.

```
## Número de Hill para q = 2
hill_res_q_2 <- hill_taxa(composicao_especies, q  = 2)
hill_res_q_2
#>     Com_1     Com_2     Com_3     Com_4     Com_5     Com_6     Com_7
Com_8     Com_9    Com_10
#> 10.000000  1.206273  1.928571  4.974223  4.145078  4.300813  2.000000
2.409639  5.232558  3.270270
```

Criando um data frame com os três resultados anteriores.

```
## Resultados
res_hill <- data.frame(hill_res_q_0, hill_res_q_1, hill_res_q_2)
colnames(res_hill) <- c("q=0", "q=1", "q=2")
head(res_hill)
#>       q=0       q=1       q=2
#> Com_1  10 10.000000 10.000000
#> Com_2  10  1.649196  1.206273
#> Com_3   5  2.606507  1.928571
#> Com_4   5  4.987156  4.974223
#> Com_5   5  4.420220  4.145078
#> Com_6   6  4.762172  4.300813
```

Interpretação dos resultados

Como na comunidade 1 todas as espécies são igualmente abundantes, alterar os valores de `q` não altera o número efetivo de espécies que permanece sempre 10. Contudo, na comunidade 2, que apresenta alta dominância de uma espécie, alterar os valores de `q` diminui consideravelmente a estimativa da diversidade. A vantagem dos Números de Hill é que eles são de fácil interpretação e comparação entre as comunidades. Fator ausente para os índices de diversidade.

> **Importante**
> Neste ponto, esperamos que tenha ficado claro que mais do que a riqueza de espécies, a abundância relativa das espécies (e.g., comuns ou raras) tem um papel fundamental na estimativa da diversidade de espécies.

12.3 Diversidade beta

O termo diversidade beta foi proposto por Whittker (1960) e foi definido como a razão entre a diversidade gama e a diversidade alfa (i.e., diversidade beta multiplicativa), quantificando não só a relação entre a diversidade regional e local, mas também o grau de diferenciação entre as comunidades. Para

demonstrar como a diversidade beta varia entre comunidades locais dentro de uma região usaremos a explicação do Baselga. Imagine três comunidades, cada comunidade abrigando as mesmas cinco espécies. Neste caso, a média da diversidade alfa = 5, a diversidade gama = 5 e a razão entre elas (gama/alfa) indica uma diversidade beta = 1. Isso significa que na região existe apenas uma unidade distinta de composição. Quando a composição de espécies das três comunidades é completamente diferente (i.e., diferenciação máxima), temos que a média da diversidade alfa = 5, a diversidade gama = 15 e a razão entre elas indica uma diversidade beta = 3. Neste caso, existem três unidades distintas dentro da região. Assim, a diversidade beta multiplicativa varia de 1 até o número de comunidades dentro da região.

A maioria dos índices de (dis)similaridade utilizadas na ecologia (e.g., índices de *Jaccard* e *Sørensen*) são índices que padronizam a diversidade beta e geram valores independentes do número de comunidades. Eles podem ser calculados para dados de incidência (presença e ausência) ou abundância (Legendre & Legendre 2012) e considerando comparações par-a-par entre as comunidades ou comparação entre múltiplas comunidades (i.e., multiple-site). Por muito tempo, os valores de (dis)similaridade foram interpretados como sinônimo de substituição de espécies (*turnover*) entre comunidades. Contudo, índices de (dis)similaridade como *Jaccard* e *Sørensen* geram valores de (dis)similaridade para comunidades que não apresentam diferenças na composição de espécies, mas apresentam diferenças na riqueza de espécies (i.e., comunidades aninhadas). Pensando nestes fatores, Baselga (2010, 2012) propôs uma abordagem que particiona a diversidade beta total (β_{jac}) em dois componentes: o componente resultante da substituição de espécies (*turnover* - β_{tur}) e o componente resultante do aninhamento (*nestedness*, i.e., diferença na riqueza de espécies - β_{nes}). Vejam a Figura 12.5 onde temos 3 comunidades (X, Y e Z). No primeiro exemplo, temos apenas diferença no número de espécies entre as comunidades. Neste caso, o componente subsituição de espécies (β_{tur}) é zero porque as espécies na comunidade Z são um sub-grupo das espécies nas comunidades X e Y. O mesmo para as espécies na comunidade Y que são um sub-grupo das espécies da comunidade X. No segundo exemplo, temos o cenário oposto com as comunidades abrigando a mesma riqueza de espécies e assim, o componente resultante do aninhamento (β_{nes}) é zero, temos um valor máximo para o β_{tur}. Percebam que somando β_{tur} com β_{nes} temos o valor da diversidade beta total (β_{jac}). Baselga (2013) também propôs a partição da diversidade beta para índices de dissimilaridade que lidam com dados de abundância. Neste caso os componentes da diversidade beta são chamados de variação balanceada na abundância (similar ao componente substituição de espécies) e gradiente de abundância (similar ao componente aninhamento). Reconhecer estes componentes da diversidade beta é importante porque eles apresentam padrões distintos (substituição de espécies *versus* perda ordenada de espécies), que provavelmente estão sendo gerados por processos ecológicos diferentes (Baselga 2010, Baselga 2012, Baselga 2013).

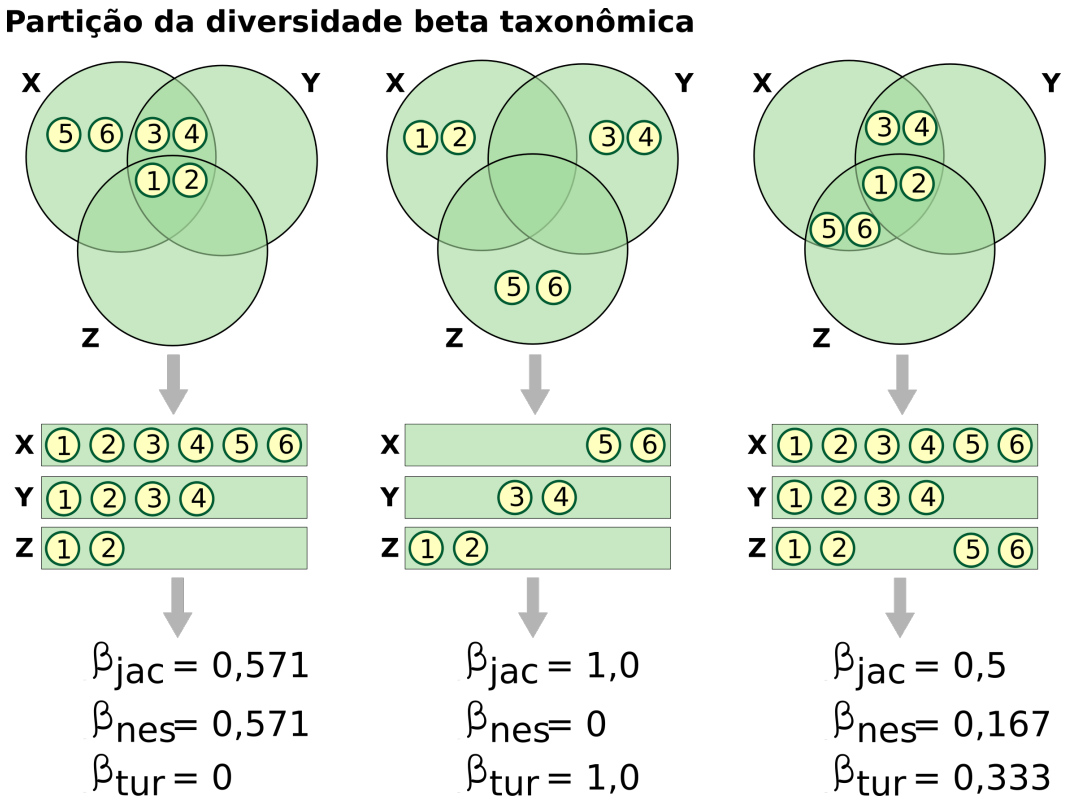


Figura 12.5: Partição da diversidade beta taxonômica. Os três cenários apresentados representam a diversidade beta explicada somente por substituição, aninhamento e uma combinação dos dois.

Aqui, vamos demonstrar alguns exemplos de como calcular a partição da diversidade beta para os dados deste capítulo.

Para isso, primeiro vamos transformar nossos dados de abundância em presença e ausência.

```
## Transformando dados em presencia e ausência.
composicao_PA <- decostand(composicao_especies, method = "pa")
```

Calculando a diversidade beta par a par usando os dados de presença e ausência.

```
## Diversidade beta
resultado_PA <- beta.pair(composicao_PA, index.family = "sorensen")
```

> **Importante**
> A função `beta.pair()` gera três listas com matrizes triangulares:
> - **Diversidade beta total** = índice de *Sørensen* (beta.sor)
> - **Componente de substituição** = índice de *Simpson* (beta.sim)
> - **Componente de aninhamento** = diferença na riqueza (beta.sne)

Vamos olhar os resultados da diversidade beta total.

```
## Resultados
resultado_PA$beta.sor
#>            Com_1     Com_2     Com_3     Com_4     Com_5     Com_6
```

```
Com_7     Com_8     Com_9
#> Com_2  0.0000000
#> Com_3  0.3333333 0.3333333
#> Com_4  0.3333333 0.3333333 1.0000000
#> Com_5  0.3333333 0.3333333 0.6000000 0.4000000
#> Com_6  0.2500000 0.2500000 0.4545455 0.4545455 0.4545455
#> Com_7  0.6666667 0.6666667 0.7142857 0.7142857 1.0000000 0.7500000
#> Com_8  0.4285714 0.4285714 0.7777778 0.3333333 0.3333333 0.2000000
1.0000000
#> Com_9  0.2500000 0.2500000 0.4545455 0.4545455 0.2727273 0.5000000
0.7500000 0.4000000
#> Com_10 0.4285714 0.4285714 0.3333333 0.7777778 0.5555556 0.4000000
0.6666667 0.7500000 0.6000000
```

Vamos montar um data frame com os resultados

```
## Data frame com os resultados
data.frame_PA <- data.frame(round(as.numeric(resultado_PA$beta.sor), 2),
                            round(as.numeric(resultado_PA$beta.sim), 2),
                            round(as.numeric(resultado_PA$beta.sne), 2))
colnames(data.frame_PA) <- c("Sorensen", "Simpson", "Aninhamento")
head(data.frame_PA)
#>   Sorensen Simpson Aninhamento
#> 1     0.00       0        0.00
#> 2     0.33       0        0.33
#> 3     0.33       0        0.33
#> 4     0.33       0        0.33
#> 5     0.25       0        0.25
#> 6     0.67       0        0.67
```

> **Importante**
> Percebam que a primeira linha e primeira coluna do data frame (i.e., 0.00) representa a dissimilaridade de *Sørensen* entre a Com1 e Com2 (compare com os valores da matriz triangular acima). As linhas subsequentes representam a dissimilaridade da Com1 com todas as outras comunidades, depois da Com2 com todas as comunidades e assim sucessivamente. Lembrem-se que os componentes, substituição (*Simpson*) e aninhamento, são um desdobramento da diversidade beta total (*Sørensen*). Assim, a soma da dissimilaridade de *Simpson* e aninhamento é igual ao valor de dissimilaridade de *Sørensen* (Baselga 2010, Baselga 2012)

Vamos calcular a dissimilaridade entre a precipitação anual das comunidades usando o índice de distância euclidiana. Vejam a ajuda da função `vegdist()` que calcula 17 índices diferentes de dissimilaridade.

```
## Dissimilaridade
prec_dis <- vegdist(precipitacao, method = "euclidian")
dados_prec <- as.numeric(prec_dis)
```

Agora vamos juntar os resultados.

> 👉 **Importante**
> As comunidades devem estar dispostas na mesma ordem nas duas planilhas (composição de espécies e precipitação) para que os resultados representem as dissimilaridades par a par para as mesmas comunidades no data frame.

Criando data frame.

```
## Data frame
dados_dis <- data.frame(dados_prec, data.frame_PA)
head(dados_dis)
#>   dados_prec Sorensen Simpson Aninhamento
#> 1         88     0.00       0        0.00
#> 2        400     0.33       0        0.33
#> 3       1400     0.33       0        0.33
#> 4        294     0.33       0        0.33
#> 5        195     0.25       0        0.25
#> 6       2270     0.67       0        0.67
```

Vamos testar a relação entre as diferenças na composição de espécies e precipitação nas comunidades.

```
## ANOVA
# Avaliar a relação entre os valores de diversidade beta total (Sørensen) e
precipitação.
anova_sore <-lm(Sorensen ~ dados_prec, data = dados_dis)
anova(anova_sore)
#> Analysis of Variance Table
#>
#> Response: Sorensen
#>            Df  Sum Sq  Mean Sq F value Pr(>F)
#> dados_prec  1 0.00188 0.001877  0.0358 0.8508
#> Residuals  43 2.25264 0.052387

# Avaliar a relação entre os valores do componente substituição (Simpson) e
precipitação
anova_simp <-lm(Simpson ~ dados_prec, data = dados_dis)
anova(anova_simp)
#> Analysis of Variance Table
#>
#> Response: Simpson
#>            Df Sum Sq  Mean Sq F value Pr(>F)
#> dados_prec  1 0.1403 0.140342  1.4905 0.2288
#> Residuals  43 4.0488 0.094157

# Avaliar a relação entre os valores do componente aninhamento e precipitação
```

```
anova_anin <-lm(Aninhamento ~ dados_prec, data = dados_dis)
anova(anova_anin)
#> Analysis of Variance Table
#>
#> Response: Aninhamento
#>            Df  Sum Sq Mean Sq F value  Pr(>F)
#> dados_prec  1 0.17467 0.17467  6.4006 0.01515 *
#> Residuals  43 1.17349 0.02729
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interpretação dos resultados

Há uma relação positiva entre o componente aninhado da diversidade beta e a diferença na precipitação entre as comunidades ($F_{1,43}$ = 6,4, P = 0,01). Contudo, não há relação entre a diversidade beta total (*Sørensen*) e o componente substituição de espécies (*Simpson*) com a precipitação (P > 0,05).

Agora vamos fazer um gráfico com o componente aninhamento da diversidade beta (Figura 12.6).

```
## Gráfico
ggplot(data = dados_dis, aes(x = dados_prec, y = Aninhamento)) +
    geom_point(size = 4, shape = 21, fill = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Diferença precipitação (mm)",
         y = "Componente aninhamento da\n diversidade beta") +
    tema_livro()
```

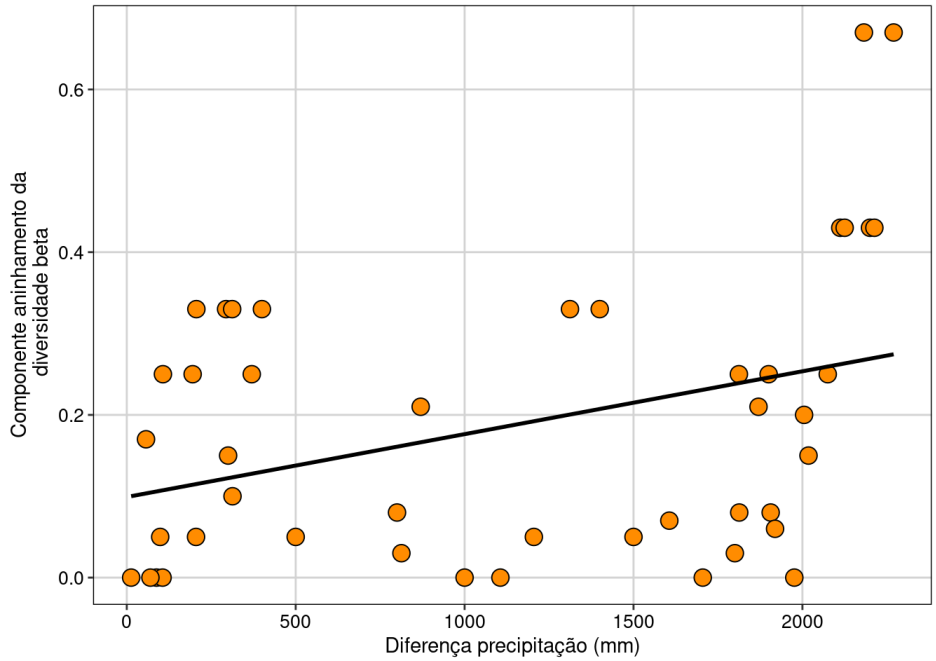


Figura 12.6: Relação entre o componente de aninhamento da diversidade beta e a diferença da precipitação.

Interpretação dos resultados

As comunidades localizadas em locais com baixa precipitação anual apresentam espécies que são um subgrupo das espécies presentes nas comunidades com alta precipitação anual.

Agora vamos fazer um exemplo considerando os dados de abundância das espécies.

A função `beta.pair.abund()` gera três listas com matrizes triangulares:

- **Diversidade beta total** = índice de *Bray-Curtis* (beta.bray)
- **Componente variação balanceada (beta.bray.bal)**
- **Componente gradiente de abundância (beta.bray.gra)**

Análise.

```
## Diversidade beta para abundância
resultado_AB <- beta.pair.abund(composicao_especies,
                                index.family = "bray")
```

Cria um data frame com os resultados.

```
## Data frame
# Vamos montar um data.frame com os resultados.
data.frame_AB <- data.frame(round(as.numeric(resultado_AB$beta.bray), 2),
                        round(as.numeric(resultado_AB$beta.bray.bal), 2),
                        round(as.numeric(resultado_AB$beta.bray.gra), 2))
colnames(data.frame_AB) <- c("Bray", "Balanceada", "Gradiente")
head(data.frame_AB)
#>   Bray Balanceada Gradiente
#> 1 0.81       0.81      0.00
#> 2 0.69       0.42      0.27
#> 3 0.45       0.38      0.06
#> 4 0.47       0.07      0.40
#> 5 0.47       0.15      0.31
#> 6 0.92       0.00      0.92

## Agora vamos juntar os resultados com a precipitação
dados_dis_AB <- data.frame(dados_prec, data.frame_AB)
```

Testar a relação da dissimilaridade considerando a abundância com a diferença na precipitação entre as comunidades.

```
## ANOVA
# Avaliar a relação entre os valores de diversidade beta total e precipitação
anova_dis_AB <- lm(Bray ~ dados_prec, data = dados_dis_AB)
anova(anova_dis_AB)
#> Analysis of Variance Table
#>
#> Response: Bray
```

```
#>            Df  Sum Sq  Mean Sq F value Pr(>F)
#> dados_prec  1 0.01782 0.017815  0.8441 0.3634
#> Residuals  43 0.90755 0.021106

# Avaliar a relação entre os valores do componente balanceada e precipitação
anova_balan <- lm(Balanceada ~ dados_prec, data = dados_dis_AB)
anova(anova_balan)
#> Analysis of Variance Table
#>
#> Response: Balanceada
#>            Df  Sum Sq Mean Sq F value  Pr(>F)
#> dados_prec  1 0.48761 0.48761  7.0742 0.01094 *
#> Residuals  43 2.96391 0.06893
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Avaliar a relação entre os valores do componente gradiente e precipitação
anova_grad <- lm(Gradiente ~ dados_prec, data = dados_dis_AB)
anova(anova_grad)
#> Analysis of Variance Table
#>
#> Response: Gradiente
#>            Df  Sum Sq Mean Sq F value    Pr(>F)
#> dados_prec  1 0.68981 0.68981  18.705 8.903e-05 ***
#> Residuals  43 1.58575 0.03688
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Interpretação dos resultados

Há uma relação positiva entre os componentes variação balanceada ($F_{1,43}$ = 7,07, P = 0,01) e gradiente ($F_{1,43}$ = 18,7, P < 0,001) de abundância da diversidade beta com a diferença na precipitação entre as comunidades. Contudo, não há relação entre a diversidade beta total (Bray) com a precipitação ($F_{1,43}$ = 0,84, P = 0,36).

Vamos fazer um gráfico para a variação balanceada da diversidade beta (Figura 12.7).

```
ggplot(data = dados_dis_AB, aes(x = dados_prec, y = Balanceada)) +
    geom_point(size = 4, shape = 21, fill = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Diferença precipitação (mm)",
         y = "Componente variação balanceada\n da diversidade beta") +
    tema_livro()
```

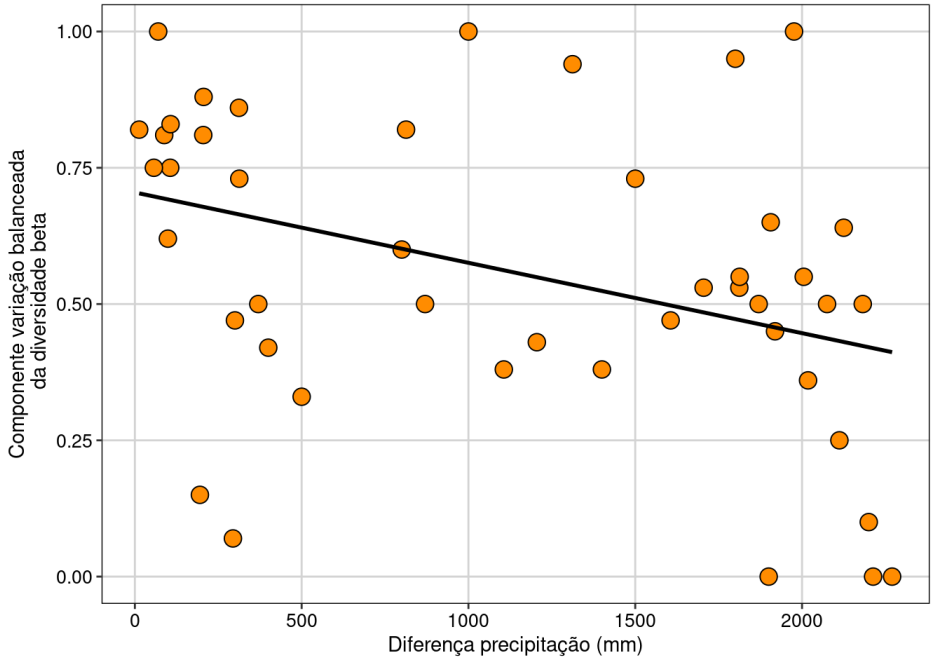


Figura 12.7: Relação entre o componente variação balanceada da diversidade beta e a diferença da precipitação.

Interpretação dos resultados

Olhando o início do eixo X onde as comunidades apresentam precipitação anual similares (i.e., baixa diferença na precipitação), o componente variação balanceada indica que há uma tendência das espécies com maiores abundâncias não serem as mesmas quando comparamos duas comunidades (i.e., maiores valores de dissimilaridade). Por outro lado, quando a diferença na precipitação entre duas comunidades é alta, o componente variação balanceada é baixo, indicando que as mesmas espécies estão dominando a abundância nas comunidades comparadas.

Vamos fazer agora um gráfico para a variação gradiente da diversidade beta (Figura 12.8).

```
ggplot(data = dados_dis_AB, aes(x = dados_prec, y = Gradiente)) +
    geom_point(size = 4, shape = 21, fill = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Diferença precipitação anual (mm)",
         y = "Componente gradiente de abundância\n da diversidade beta") +
    tema_livro()
```

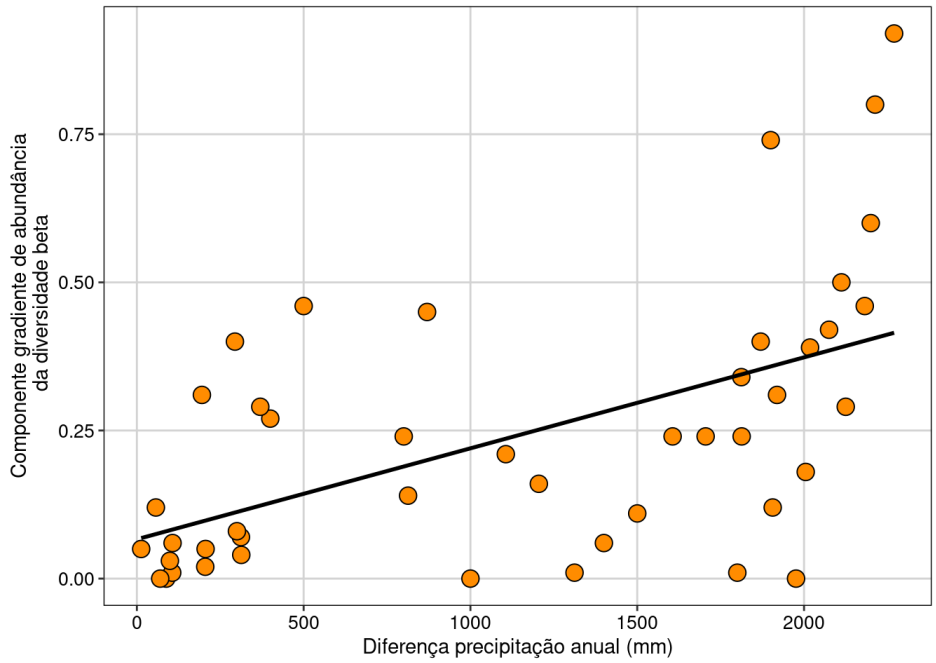


Figura 12.8: Relação entre o componente variação gradiente da diversidade beta e a diferença da precipitação.

Interpretação dos resultados

Olhando o início do eixo X onde as comunidades apresentam precipitação anual similares (i.e., baixa diferença na precipitação), o componente gradiente indica que há uma tendência das espécies apresentarem abundâncias parecidas (i.e., menor valor de dissimilaridade). Por outro lado, quando a diferença na precipitação entre duas comunidades é alta, o componente gradiente é alto, indicando que as mesmas espécies têm valores discrepantes de abundâncias entre as comunidades.

12.4 Para se aprofundar

12.4.1 Livros

- Recomendamos a leitura dos artigos citados no capítulo e os livros de Magurran & McGill (2011) - *Biological Diversity Frontiers in Measurement and Assessment* e Legendre & Legendre (2012) - *Numerical Ecology*.

12.4.2 Links

Recomendamos a página pessoal do pesquisador Lou Jost que apresenta e discute diversas medidas de diversidade taxonômica:

- Medidas de diversidade e similaridade
- Medindo a diversidade de uma única localidade
- Comparando a diversidade entre duas comunidades
- Número efetivo de espécies

12.5 Exercícios

12.1 Carregue os dados - `anuros_composicao` - que está no pacote `ecodados`. Este conjunto de dados representa a abundância de 211 espécies de anuros coletados em 44 localidades na Mata Atlântica. Calcule a riqueza de espécies para cada comunidade e os índices de Margalef, Menhinich, Shannon-Wiener, Gini-Simpson e Equitabilidade de Pielou. Salve todos os resultados em novo `data frame`. Faça um gráfico usando a função `ggpairs` para ver a correlação entre as métricas. Qual a sua interpretação sobre os resultados?

12.2 Usando os resultados anteriores, selecione as duas comunidades com os maiores e menores valores de Shannon-Wiener. Em seguida, faça um Diagrama de Whittaker. Por fim, interprete as curvas considerando as curvas teóricas (i.e., geométrica, broken-stick, etc.) descritas nos livros de ecologia.

12.3 Usando os dados - `anuros_composicao` - calcule a partição da diversidade beta considerando os dados de abundância e presença e ausência. a) Faça um gráfico boxplot com os resultados. Discuta se os resultados usando abundância ou presença e ausência são congruentes ou discrepantes. b) Calcule a distância geográfica (use a planilha `anuros_ambientais`) entre as localidades (use a Distância Euclidiana). Em seguida, faça uma análise de regressão para verificar se as localidades que estão próximas apresentam maior similaridade na composição de espécies (use componente turnover - Bsim) do que as comunidades que estão distantes (e.g., Decaimento da similaridade).

Soluções dos exercícios.

Capítulo 13

Diversidade Filogenética

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(devtools)
library(ecodados)
library(V.PhyloMaker)
library(vegan)
library(ggplot2)
library(GGally)
library(ggpubr)
library(picante)
library(phytools)
library(ape)
library(geiger)
library(phyloregion)
library(pez)
library(reshape2)
library(betapart)

## Dados
minha_arvore <- ecodados::filogenia_aves
especies_plantas <- ecodados::sp_list
comunidade <- ecodados::comm
composicao_especies <- ecodados::composicao_aves_filogenetica
filogenia_aves <- ecodados::filogenia_aves
precipitacao <- precipitacao_filogenetica
```

13.1 Aspectos teóricos

A diversidade filogenética captura a ancestralidade compartilhada entre as espécies em termos de quantidade da história evolutiva e o grau de parentesco entre as espécies. Pesquisadores têm utilizado diferentes métricas de diversidade filogenética em duas linhas de investigações principais: i) incorporar a história evolutiva das espécies na seleção das áreas prioritárias para conservação visando minimizar a perda da diversidade evolutiva (Vane-Wright et al. 1991, Faith 1992, Véron et al. 2019), e ii) produzir explicações sobre os processos atuando na montagem das comunidades (Webb et al. 2002, Helmus et al. 2007). A quantidade de artigos abordando ecologia, macroecologia e conservação com diversidade filogenética cresceram exponencialmente nas últimas décadas (Véron et al. 2019). Seguindo esta tendência, o número de métricas de diversidade filogenética propostas não param de aumentar. Tucker et al. (2017) revisaram 70 métricas de diversidade filogenética e classificaram estas métricas em três dimensões: i) **riqueza** - representa a soma da diferença filogenética acumulada entre táxons, ii) **divergência** - representa o padrão de diferença filogenética entre táxons de uma assembleia, e iii) **regularidade** - representa o grau de variação das diferenças filogenéticas entre táxons em uma assembleia. Outros autores utilizaram diferentes classificações (Pavoine & Bonsall

2011, Vellend et al. 2011, Garamszegi 2014). Neste capítulo, iremos seguir a classificação de Tucker et al. (2017) e mostrar algumas das principais métricas dentro de cada uma dessas dimensões.

> **Importante**
> Alguns autores recomendam que os pesquisadores não foquem em apenas uma dimensão, mas comparem métricas de diferentes dimensões (Tucker et al. 2017).

13.2 Manipulação de filogenias

Nesta seção, iremos descrever os códigos em R para carregar, plotar, acessar os dados, e excluir e adicionar espécies em filogenias. Estes são códigos introdutórios e necessários para realizarmos as análises de diversidade filogenética. Não iremos descrever os comandos necessários para construir uma filogenia. Estamos assumindo que já existe uma filogenia disponível para os organismos de interesse.

Mas antes, vamos entender as principais terminologias de uma filogenia e analisá-las graficamente (Figura 13.1).

- **Árvore filogenética**: são hipóteses que representam a relação de parentesco entre as espécies (pode ser também indivíduos, genes, etc.) com informações sobre quais espécies compartilham um ancestral comum e a distância (tempo, genética, ou diferenças nos caracteres) que as separam
- **Nó**: o ponto onde uma linhagem dá origem a duas ou mais linhagens descendentes
- **Politomia**: três ou mais linhagens descendendo de um único nó
- **Ramo**: uma linha orientada ao longo de um eixo terminais-raiz que conecta os nós na filogenia
- **Terminal (do inglês *tip*)**: o final do ramo representando uma espécie atual ou extinta (pode também representar gêneros, indivíduos, genes, etc.)
- **Raiz**: representa o ancestral comum de todas as espécies na filogenia
- **Clado**: um grupo de espécies aparentadas descendendo de um único nó na filogenia
- **Ultramétrica**: a distância de todos os terminais até a raiz são idênticas. Característica requerida pela maioria dos índices de diversidade filogenética

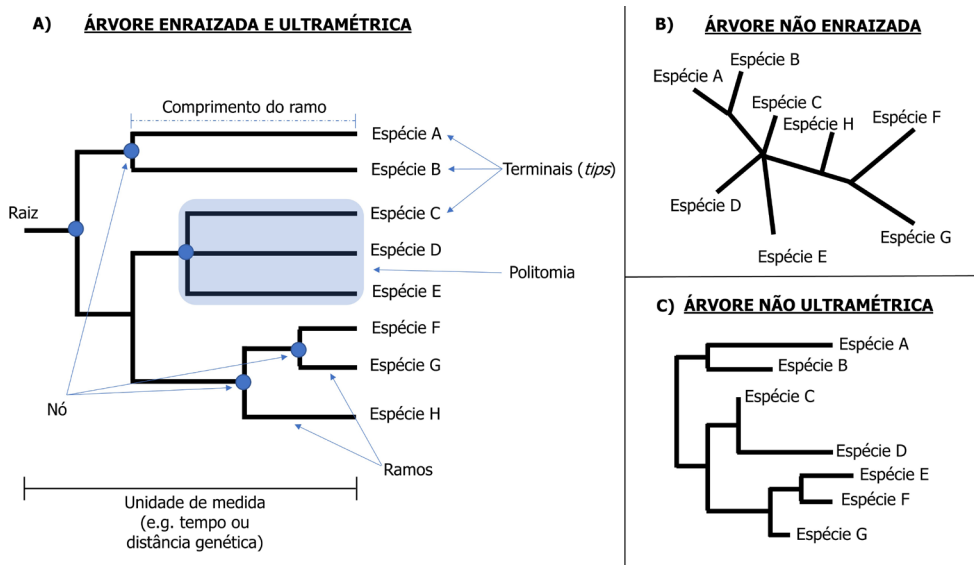


Figura 13.1: Ilustrações de diferentes árvores filogenéticas. A) Árvore enraizada e ultramétrica indicando a raiz da árvore, nós, ramos, comprimento do ramo, politomias e terminais. B) Árvore não enraizada que mostra as relações entre as espécies, mas não define a história evolutiva. C) Árvore não ultramétrica onde as espécies apresentam diferentes distâncias até a raiz.

Agora vamos plotar a filogenia para visualizar as relações entre as 37 espécies de aves endêmicas da Mata Atlântica. Essa filogenia foi extraída de Jetz et al. (2012). Os dados estão disponíveis no pacote `ecodados` (Figura 13.2).

```
## Gráfico
plot.phylo (minha_arvore, type = "phylogram", show.tip.label = TRUE,
          show.node.label = TRUE, edge.color = "black", edge.width = 1.5,
          tip.color = "black", cex = 0.45, label.offset = 2)
```

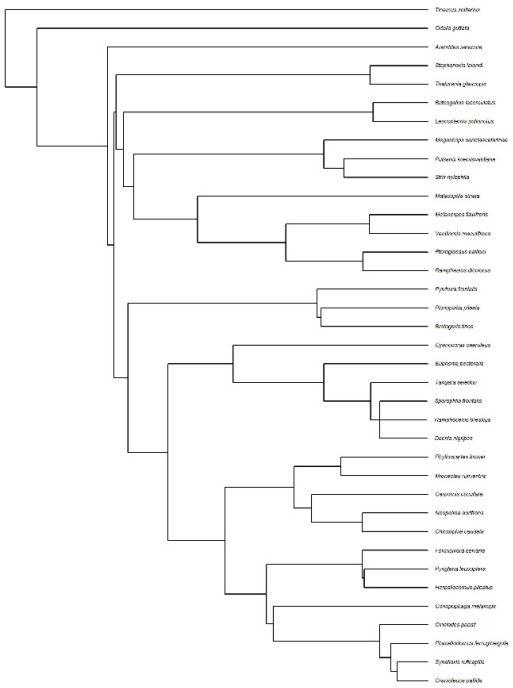


Figura 13.2: Filogenia de 37 espécies de aves endêmicas da Mata Atlântica.

Podemos alterar o formato de apresentação da filogenia usando o argumento `type` e a cor dos ramos usando o argumento `edge.color` (Figura 13.3).

```
## Gráfico
plot.phylo (minha_arvore, type = "fan", show.tip.label = TRUE,
            show.node.label = TRUE, edge.color = "blue", edge.width = 1.5,
            tip.color = "black", cex = 0.45, label.offset = 2)
```

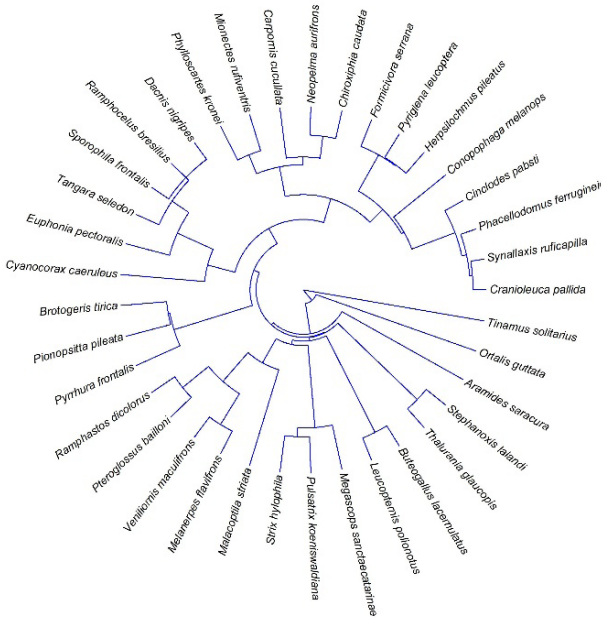


Figura 13.3: Filogenia de 37 espécies de aves endêmicas da Mata Atlântica, com alterações de parâmetros de visualização.

Percebam que existem vários argumentos para modificar a largura e cor dos ramos, tamanho da fonte, distância entre a filogenia e os nomes das espécies e muito mais. Uma sugestão é visitar o blog do professor Liam Revell (http://blog.phytools.org/) que é o criador e mantenedor do pacote `phytools` no R.

Acessar informações da filogenia

Uma das características mais interessantes do R é que podemos acessar as informações do objeto que contém a filogenia. Neste caso, o nosso objeto é a filogenia e, muitas vezes, temos interesse nas informações que estão inseridas dentro da filogenia. Para sabermos quais são as informações que podemos acessar na filogenia, vamos usar a função `names()`.

```
## Nomes
names(minha_arvore)
#> [1] "edge"        "edge.length" "Nnode"       "tip.label"
```

Temos acesso a quatro componentes da filogenia: i) ramo (`edge`), ii) comprimento do ramo (`edge.length`), iii) número de nós (`Nnode`), e iv) nome das espécies (`tip.label`). Podemos usar o operador `$` para acessar estes componentes. Veja abaixo como acessar o nome das 37 espécies de aves na filogenia ou o comprimento de cada um dos ramos da filogenia.

```
## Nome das espécies
minha_arvore$tip.label
```

```
## Comprimento dos ramos
minha_arvore$edge.length
```

Remover espécies da filogenia

Nas análises de diversidade filogenética, as espécies que estarão presentes na filogenia normalmente são aquelas que foram amostradas no seu estudo. Contudo, muitas vezes utilizamos filogenias contendo espécies que não estão presentes no nosso estudo. Neste caso, precisamos excluir essas espécies da filogenia. A função `drop.tip()` faz essa tarefa.

```
## Remover espécies da filogenia
# Vamos criar um novo nome para o objeto e excluir as espécies Leucopternis
polionotus e Aramides saracura da filogenia
filogenia_cortada <- drop.tip(minha_arvore,
                              c("Leucopternis_polionotus",
                                "Aramides_saracura"))
filogenia_cortada
#>
#> Phylogenetic tree with 35 tips and 33 internal nodes.
#>
#> Tip labels:
#>   Cranioleuca_pallida, Synallaxis_ruficapilla, Phacellodomus_
ferrugineigula, Cinclodes_pabsti, Conopophaga_melanops, Herpsilochmus_
pileatus, ...
#>
#> Rooted; includes branch lengths.
```

Vejam que agora a filogenia tem 35 espécies de aves. As duas espécies que selecionamos foram excluídas da filogenia.

Adicionar espécies na filogenia

Outra situação bem comum é quando precisamos inserir espécies que foram amostradas no nosso estudo, mas não estão presente na filogenia. Para isso, vamos usar a função `add.species.to.genus()`. A função `force.ultrametric()` é usada para que a filogenia continue sendo ultramétrica (sem essa função a árvore perde os comprimentos dos ramos)

> **Importante**
> O comprimento do ramo que a espécie irá receber dependerá de onde você indicar a inserção da espécie.
>
> As opções são:
>
> - `root`: insere a espécie no ancestral comum mais recente (MRCA) de todas as espécies do gênero (*default*)
> - `random`: insere a espécie aleatoriamente dentro do clado do MRCA contendo todos as espécies do gênero

```
## Adicionar espécies à filogenia
# Vamos inserir as espécies Megascops_sp1, Carponis_sp, Strix_sp1, Strix_sp2 e
Strix_sp3 na filogenia
Megascops <- c("Megascops_sp1")
Carpornis <- c("Carpornis_sp1")
Strix <- c("Strix_sp1", "Strix_sp2", "Strix_sp3")

# Inserindo espécies como politomias
filogenia_nova <- add.species.to.genus(force.ultrametric(minha_arvore,
                                        message = FALSE), Megascops)
filogenia_nova <- add.species.to.genus(force.ultrametric(filogenia_nova,
                                        message = FALSE), Carpornis)
```

Agora vamos inserir várias espécies dentro do mesmo gênero.

```
## Adicionar várias espécies à filogenia
# Para inserir mais de uma espécie dentro do gênero, vamos utilizar um loop.
for(i in 1:length(Strix))
    filogenia_nova <- add.species.to.genus(force.ultrametric(
                                            filogenia_nova,
                                            message = FALSE),
                                            Strix[i], where = "root")
```

Vamos plotar essa nova filogenia (Figura 13.4).

```
## Gráfico
plot(filogenia_nova, cex = 0.5, no.margin = TRUE)
```

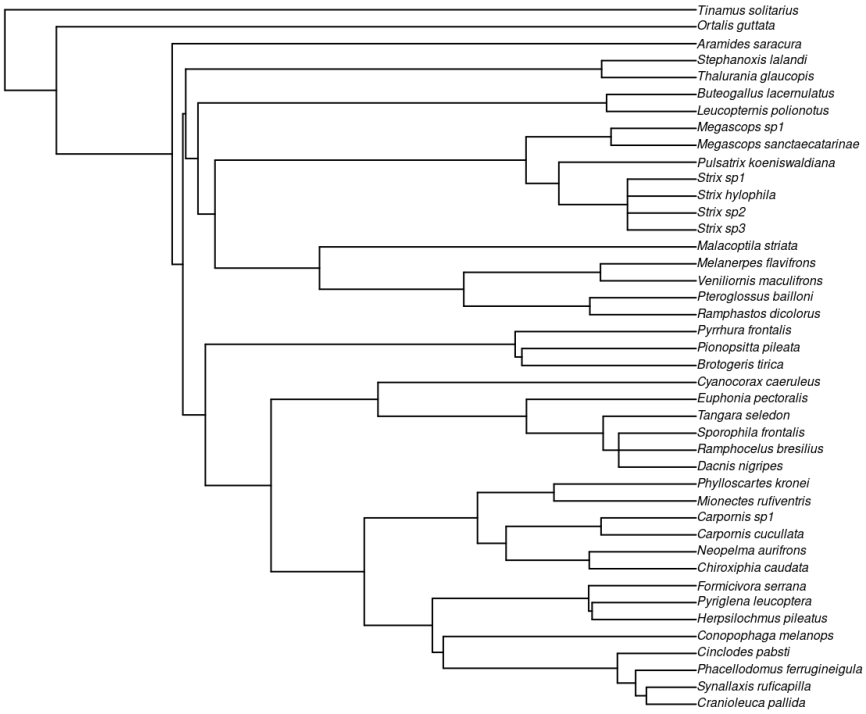


Figura 13.4: Filogenia de espécies de aves endêmicas da Mata Atlântica, com adição de espécies.

Vamos fazer outro exemplo usando a função `phylo.maker()` do pacote `V.PhyloMaker` que adiciona as espécies nos gêneros ou os gêneros nas famílias usando uma filogênia *backbone*.

Essa função permite a adição dos gêneros ou espécies considerando três cenários diferentes:

- **Cenário 1**: adiciona gêneros ou espécies como politomias basais dentro das famílias ou gêneros da filogenia respectivamente
- **Cenário 2**: adiciona gêneros e espécies aleatoriamente nas famílias ou gêneros da filogenia respectivamente
- **Cenário 3**: adiciona gêneros e espécies nas famílias ou gêneros da filogenia respectivamente usando as abordagens implementadas no *Phylomatic* e *BLADJ*

```
## phylo.maker
# A função phylo.maker usa uma filogenia default de plantas (i.e. GBOTB.
extended).
# Caso você queira utilizar outra filogenia, é só alterar o argumento tree
novas_filogenias <- phylo.maker(especies_plantas,
                                tree = GBOTB.extended,
                                scenarios = c("S1","S2","S3"))
#> [1] "Note: 2 taxa fail to be binded to the tree,"
#> [1] "Genus7_sp1" "Genus8_sp1"
```

Vamos essa filogenia criada pelo pacote `V.PhyloMaker` (Figura 13.5).

```
## Gráfico
par(mfrow = c(1, 2))
plot.phylo(novas_filogenias$scenario.1, cex = 0.5, main = "Cenário 1")
plot.phylo(novas_filogenias$scenario.3, cex = 0.5, main = "Cenário 3")
dev.off()
```

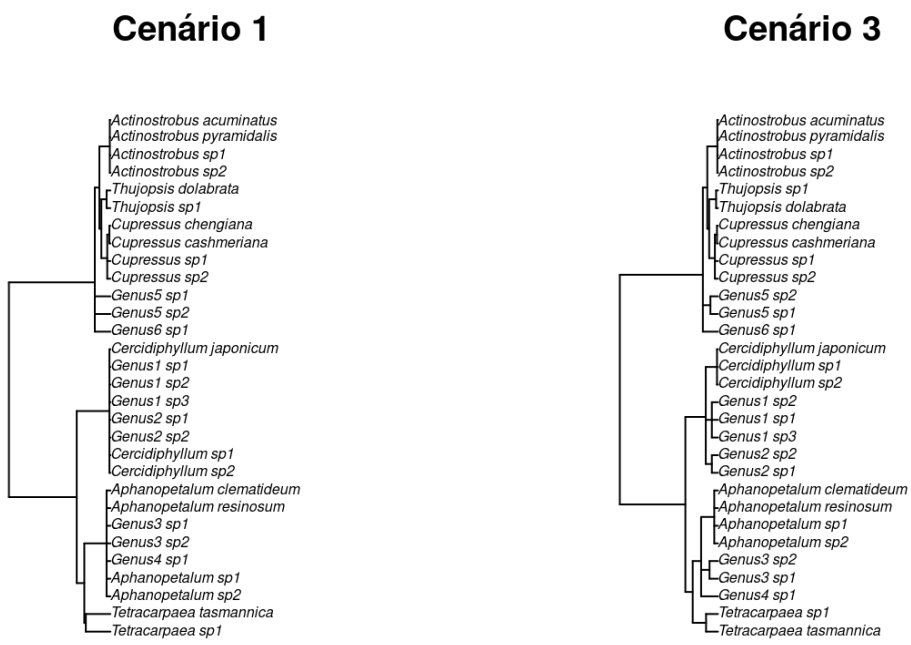


Figura 13.5: Filogenia de espécies de plantas criada pelo pacote `V.PhyloMaker`.

13.3 Métricas de diversidade alfa filogenética

Métricas de diversidade alfa utilizam os dados de incidência (presença e ausência) ou abundância das espécies para determinar um valor de diversidade para cada comunidade ou sítio de interesse.

Exemplo prático 1

Explicação dos dados

Avaliaremos a diversidade filogenética de 10 comunidades de aves amostradas ao longo de um gradiente de precipitação. Utilizaremos este conjunto de dados para todos os exemplos deste capítulo.

Pergunta

- A variação na distribuição espacial dos valores de diversidade filogenética das comunidades está associada com o gradiente de precipitação?

Predições

- Os valores de diversidade filogenética serão maiores nas comunidades localizadas em regiões com altas precipitações do que em regiões mais secas

Variáveis

- Data frame com as comunidades (unidade amostral) nas linhas e as espécies de aves nas colunas (variável resposta)
- Data frame com as comunidades (unidade amostral) nas linhas e a variável precipitação anual na coluna (variável preditora)
- Arquivo com a filogenia das 37 espécies de aves (variável resposta)

Checklist

- Verificar se os data frames de composição de espécies e variáveis ambientais estão com as unidades amostrais nas linhas e variáveis preditoras nas colunas
- Verificar se as comunidades nos data frames de composição de espécies e variáveis ambientais estão distribuídos na mesma sequência/ordem nos dois arquivos.
- Verificar se o nome das espécies de aves no data frame de composição de espécies é idêntico ao nome das espécies na filogenia.

13.3.1 Riqueza da diversidade alfa filogenética

As métricas de riqueza somam a quantidade da diferença filogenética presente em uma comunidade (Tucker et al. 2017).

Phylogenetic diversity (PD)

Esta métrica é definida pela soma do comprimento dos ramos conectando todas as espécies na comunidade. É a métrica mais conhecida e usada nos estudos de conservação e comunidade (Faith 1992).

Vamos conferir se os nomes das espécies de aves no data frame de composição são os mesmos da filogenia. O resultado `OK` indica que os nomes estão corretos. Caso contrário, você deve verificar e arrumar.

```
## Conferir os nomes das espécies
name.check(filogenia_aves, t(composicao_especies))
#> [1] "OK"
```

Agora vamos colocar os nomes das espécies do data frame na mesma ordem que os nomes das espécies aparecem na filogenia. Isso é obrigatório para algumas funções.

```
## Colocar os nomes das espécies do data frame na mesma ordem que aparecem na filogenia
composicao_especies_P <- match.phylo.comm(phy = filogenia_aves,
                                          comm = composicao_especies)$comm
```

Abaixo, demonstramos os códigos no R para o cálculo de PD para as comunidades de aves.

```
## Phylogenetic diversity (PD)
# Calculando a métrica de diversidade filogenética proposta por Faith (1992).
resultados_PD <- pd(composicao_especies_P, filogenia_aves)

# Mostra o valor de PD e riqueza de espécies para cada comunidade.
resultados_PD
#>                PD SR
#> Com_1  1259.3151 27
#> Com_2  1293.1521 26
#> Com_3  1222.3102 25
#> Com_4  1254.5410 25
#> Com_5  1021.9670 22
#> Com_6   856.7810 18
#> Com_7   930.6452 15
#> Com_8   678.9394 12
#> Com_9   673.6288 13
#> Com_10  599.6924  9
```

A comunidade 2 abriga a maior diversidade filogenética com a composição de espécies contemplando `1293,15` milhões de anos (i.e., soma do comprimento dos ramos ligando todas as espécies da comunidade). Por outro lado, a comunidade 10 abriga a menor diversidade filogenética contemplando `599,69` milhões de anos.

> **Importante**
> Este índice é correlacionado com a riqueza de espécies. Discutiremos essa questão na seção de modelos nulos.

Phylogenetic Species Richness (PSR)

Esta métrica é calculada multiplicando a riqueza de espécies registrada na comunidade pela *Phylogenetic Species Variability* (PSV) da comunidade (Helmus et al. 2007). PSR é diretamente comparável ao número de espécies na comunidade, mas inclui o parentesco filogenético entre as espécies.

Abaixo, demonstramos os códigos no R para o cálculo do PSR utilizando os dados das comunidades de aves.

```
## Phylogenetic Species Richness (PSR)
# Análise com dados de composição das espécies nas comunidades.
resultados_PSR <- psr(composicao_especies_P,filogenia_aves)

# Mostra os valores de PSR para cada comunidade.
resultados_PSR
#>              PSR SR       vars
#> Com_1  18.084236 27 0.04537904
#> Com_2  18.167183 26 0.04881734
#> Com_3  16.230938 25 0.05205832
#> Com_4  17.153972 25 0.05205832
#> Com_5  13.981597 22 0.06060866
#> Com_6  11.287030 18 0.06933707
#> Com_7  10.279983 15 0.07398666
#> Com_8   7.538134 12 0.07721118
#> Com_9   8.060933 13 0.07627517
#> Com_10  5.720063  9 0.07948474
```

A comunidade 2 abriga o maior valor de PSR enquanto a comunidade 10 abriga o menor valor. Vejam que PSR é fortemente correlacionado com número de espécies nas comunidades ($r = 0.99$, $p < 0.0001$). Contudo, existe uma variabilidade residual no PSR em relação ao número de espécies que afeta o ranqueamento das comunidades quando utilizando PSR ou número de espécies. Consequentemente, a escolha da métrica pode gerar diferentes delineamentos de áreas prioritárias para conservação (Helmus et al. 2007).

Phylogenetic Endemism (PE)

Esta métrica calcula a fração dos ramos restritas a regiões específicas. PE identifica áreas ou comunidades que abrigam componentes restritos da diversidade filogenética. PE é uma métrica proposta para auxiliar estudos de conservação estabelecendo critérios para priorizar regiões a serem conservadas com base na importância evolutiva (i.e., partes da filogênia com distribuição espacial limitada) das espécies que ocorrem nestes locais (Rosauer et al. 2009).

Abaixo, demonstramos os códigos no R para o cálculo do PE utilizando os dados das comunidades de aves.

```
## Phylogenetic Endemism (PE)
# Transformando data.frame em matriz.
dados_matriz <- as.matrix(composicao_especies_P)

# Análise.
resultados_PE <- phylo_endemism(dados_matriz, filogenia_aves,
                                weighted = TRUE)

# Mostra os valores de PE para cada comunidade.
resultados_PE
#>     Com_1     Com_2     Com_3     Com_4     Com_5     Com_6     Com_7
```

```
Com_8     Com_9    Com_10
#> 232.09145 272.60106 210.22647 218.89037 146.99281 135.06423 148.65234
79.22402  77.95458  68.50266
```

O índice PE considera as 10 comunidades como o range espacial máximo. Se todas as espécies ocorressem nas 10 comunidades, o valor de PE seria 1, indicando baixo endemismo filogenético. A comunidade 2 abriga um conjunto de espécies cujo os ramos com distribuição espacial restrita contemplam `272,6` milhões de anos. Por outro lado, a comunidade 10 abriga um conjunto de espécies cujo os ramos com distribuição espacial restrita contemplam `68,5` milhões de anos. Assim, as comunidades 1, 2 e 4 são as áreas que abrigam os maiores endemismo filogenéticos.

Species Evolutionary Distinctiveness (ED)

Esta métrica calcula qual é a fração da árvore filogenética que é atribuída para uma espécie. ED reflete quão evolutivamente isolada uma espécie é comparada com as outras espécies na filogenia (Redding and Mooers 2006). ED é uma métrica proposta para auxiliar estudos de conservação estabelecendo critérios para priorizar as espécies a serem conservadas com base na sua importância evolutiva (exclusividade do comprimento do ramo) que não é compartilhada com outras espécies. Portanto, apenas as informações da filogenia são utilizadas para o cálculo de ED.

Abaixo, demonstramos os códigos no R para o cálculo do ED utilizando os dados das comunidades de aves.

```
## Species Evolutionary Distinctiveness (ED)
# Análise.
resultados_ED <- evol.distinct(filogenia_aves)

# Mostra os valores de ED para cada espécie.
head(resultados_ED)
#>                         Species        w
#> 1          Cranioleuca_pallida 14.07447
#> 2       Synallaxis_ruficapilla 14.07447
#> 3 Phacellodomus_ferrugineigula 20.05793
#> 4             Cinclodes_pabsti 30.27020
#> 5         Conopophaga_melanops 47.72685
#> 6       Herpsilochmus_pileatus 26.40947
```

Com base na filogenia estudada, *Tinamus solitarius* (`112,58` milhões de anos), *Ortalis guttata* (`108,39` m.a.) e *Aramides saracura* (`96,86` m.a.) são as espécies com maior distinção evolutiva devido a elevada fração dos ramos não compartilhado com as outras espécies.

13.3.2 Divergência da diversidade alfa filogenética

As métricas de divergência utilizam a média da distribuição das unidades extraídas da árvore filogenética (Tucker et al. 2017).

Mean Pairwise Distance (MPD)

Esta métrica utiliza a matriz de distância filogenética para quantificar a distância média do parentesco entre pares de espécies em uma comunidade. Este índice pode ser calculado considerando dados de

incidência ou considerando dados de abundância das espécies. Importante, o MPD é uma métrica que pesa a estrutura interna da filogenia (e.g., relações entre espécies de famílias diferentes) (Webb et al. 2002).

Abaixo, demonstramos os códigos no R para o cálculo do MPD utilizando os dados das comunidades de aves.

Vamos iniciar com dados de incidência (presença e ausência) das espécies nas comunidades. A função `cophenetic()` gera uma matriz com as distâncias par a par entre as espécies. Essas distâncias são utilizadas para computar a distância média do parentesco das espécies dentro das comunidades.

```
## Mean Pairwise Distance (MPD)
# Análise com dados de incidência das espécies nas comunidades.
resultados_MPD_PA <- mpd(composicao_especies_P,
                         cophenetic(filogenia_aves),
                         abundance.weighted = FALSE)

# Mostra os valores de MPD para cada comunidade.
resultados_MPD_PA
#>  [1] 150.7914 157.3158 146.1622 154.5005 143.0727 141.1926 154.3145
141.4292 139.6198 143.0862
```

A comunidade 9 abriga a composição de espécies mais aparentada (i.e., menor diversidade filogenética) com distância média entre as espécies de `139,62` milhões de anos. Por outro lado, a comunidade 2 abriga a composição de espécies menos aparentada (i.e., maior diversidade filogenética) com distância média de `157,31` milhões anos.

Vamos refazer a análise do MPD, mas desta vez, considerando a abundância das espécies de aves nas comunidades. Para isso, alteramos o argumento `abundance.weighted = TRUE`.

```
## Mean Pairwise Distance (MPD)
# Análise com dados de abundância das espécies nas comunidades.
resultados_MPD_AB <- mpd(composicao_especies_P,
                         cophenetic(filogenia_aves),
                         abundance.weighted = TRUE)

# Mostra os valores de MPD para cada comunidade.
resultados_MPD_AB
#>  [1] 135.0704 143.3156 129.1940 142.8127 131.4027 128.7733 134.0380
132.6389 133.4041 117.8787
```

Percebam que pesando o comprimento do ramo pela abundância das espécies altera-se os valores do índice de diversidade filogenética. Neste caso, a comunidade 10 passa a ser a comunidade que abriga a composição de espécies mais aparentada (i.e., menor diversidade filogenética) com distância média entre as espécies de `117,88` milhões de anos.

Mean Nearest Taxon Distance (MNTD)

Esta métrica utiliza a matriz de distância filogenética para quantificar a média dos valores mínimos de parentesco entre pares de espécies em uma comunidade. Ou seja, qual o valor médio da distância para o vizinho mais próximo. Este índice pode ser calculado considerando dados de incidência

(presença e ausência) ou considerando dados de abundância das espécies. Diferente do MPD, o MNTD é uma métrica terminal que pesa as relações nas pontas da filogenia (e.g., espécies dentro do mesmo gênero) (Webb et al. 2002).

Abaixo, demonstramos os códigos no R para o cálculo do MNTD utilizando os dados das comunidades de aves.

```
## Mean Nearest Taxon Distance (MNTD)
# Análise com dados de presença e ausência das espécies nas comunidades.
resultados_MNTD_PA <- mntd(composicao_especies_P,
                           cophenetic(filogenia_aves),
                           abundance.weighted = FALSE)

# Mostra os valores de MPD para cada comunidade.
resultados_MNTD_PA
#>  [1]  63.89727  66.15828  72.96912  67.67170  64.93477  63.72337  93.54980
78.24876  62.34565 112.23127
```

A comunidade 9 abriga a composição de espécies com distância média do vizinho mais próximo de `62,34` milhões de anos. Esse resultado indica que as espécies terminais são mais aparentadas (e.g., espécies do mesmo gênero) do que a composição de espécies da comunidade 10 onde a distância média do vizinho mais próximo é `112,23` milhões de anos (e.g., espécies de gêneros diferentes).

Vamos refazer a análise do MNTD, mas desta vez, considerando a abundância das espécies de aves nas comunidades.

```
# Análise com dados de abundância das espécies nas comunidades.
resultados_MNTD_AB <- mntd(composicao_especies_P,
                           cophenetic(filogenia_aves),
                           abundance.weighted = TRUE)

# Mostra os valores de MPD para cada comunidade.
resultados_MNTD_AB
#>  [1]  57.11745  53.02212  70.47864  59.12049  61.23225  60.26180 110.13043
97.35404  82.12099 127.70084
```

Como nos resultados do MPD, pesar o comprimento do ramo pela abundância das espécies altera os valores do MNTD. Neste caso, ao invés da comunidade 9, a comunidade 2 passa a ser a comunidade que abriga a composição de espécies com a menor distância média do vizinho mais próximo (`53,02` milhões de anos).

> 👉 **Importante**
> Perceba que ao determinar as análises com base na incidência ou abundância das espécies, você pode também alterar a interpretação dos padrões encontrados.

Phylogenetic Species Variability (PSV)

Esta métrica estima a quantidade relativa dos comprimentos dos ramos não compartilhados entre as comunidades. Quando todas as espécies em uma amostra não são aparentadas (i.e., filogenia em estrela), o valor do PSV é 1 (um), indicando máxima variabilidade. Quando as espécies se tornam mais aparentadas, o valor de PSV aproxima-se de 0 (zero), indicando reduzida variabilidade. Os valores esperados de PSV são estatisticamente independentes da riqueza de espécies (Helmus et al. 2007).

> **Importante**
> Os valores de PSV são idênticos ao MPD quando a filogenia é ultramétrica.

Abaixo, demonstramos os códigos no R para o cálculo do PSV utilizando os dados das comunidades de aves.

```
## Phylogenetic Species Variability (PSV)
# Análise com dados de presença e ausência das espécies nas comunidades.
resultados_PSV <- psv(composicao_especies_P,filogenia_aves)

# Mostra os valores de PSV para cada comunidade.
resultados_PSV
#>             PSVs SR         vars
#> Com_1  0.6697865 27 6.224834e-05
#> Com_2  0.6987378 26 7.221499e-05
#> Com_3  0.6492375 25 8.329332e-05
#> Com_4  0.6861589 25 8.329332e-05
#> Com_5  0.6355271 22 1.252245e-04
#> Com_6  0.6270572 18 2.140033e-04
#> Com_7  0.6853322 15 3.288296e-04
#> Com_8  0.6281778 12 5.361887e-04
#> Com_9  0.6200717 13 4.513324e-04
#> Com_10 0.6355626  9 9.812931e-04
```

A comunidade 2 abriga a maior variabilidade filogenética (`0,69`) enquanto a comunidade 9 abriga a menor variabilidade (`0,62`). Perceba que os valores de PSV não são correlacionados com número de espécies nas comunidades (r = 0,59, p = 0,07).

13.3.3 Regularidade da diversidade alfa filogenética

As métricas de regularidade caracterizam a variação das distâncias entre as espécies em uma comunidade (Tucker et al. 2017).

Variance of Pairwise Distance (VPD)

Esta métrica utiliza a matriz de distância filogenética para quantificar a variância do parentesco entre pares de espécies em uma comunidade (Clarke & Warwick 2001).

Abaixo, demonstramos os códigos no R para o cálculo do VPD utilizando os dados das comunidades de aves.

```
## Variance of Pairwise Distance (VPD)
# Transformando data frame em matriz.
dados_matriz <- as.matrix(composicao_especies_P)

# Transformar os dados para o formato requerido pelo pacote pez.
dados <- comparative.comm(filogenia_aves, dados_matriz)

# Análise.
resultados_VPD <- .vpd(dados, cophenetic(filogenia_aves))

# Mostra os valores de VPD para cada comunidade.
resultados_VPD
#>     Com_1    Com_10     Com_2     Com_3     Com_4     Com_5     Com_6
Com_7     Com_8     Com_9
#> 1619.4697 1031.8887 1828.1930 1630.4026 1317.9919 1465.1728 1519.6115
825.5349 1278.0076 1508.0495
```

A comunidade 2 abriga a maior variância na distância filogenética entre pares de espécies dentro da comunidade (`1828,19` milhões de anos) enquanto a comunidade 7 abriga a menor variância entre os pares de espécies (`825,53` m.a.).

13.3.4 Correlação entre as métricas de diversidade alfa filogenética

Vamos avaliar a correlação entre os valores das métricas de diversidade alfa filogenética. Vamos criar um data frame com os resultados das métricas separados para as dimensões de riqueza e divergência. Não iremos fazer para regularidade, pois só apresentamos uma métrica de diversidade filogenética nesta dimensão (Figura 13.6).

```
## Data frame
# Vamos criar um data.frame com os resultados das métricas da dimensão
riqueza.
metricas_riqueza <- data.frame(riqueza = resultados_PD$SR,
                               PD = resultados_PD$PD,
                               PSR = resultados_PSR$PSR,
                               PE = resultados_PE)

## Gráfico
# Gráfico mostrando na parte:
# i) inferior a distribuição dos pontos considerando as métricas pareadas
# ii) superior o valor da correlação de pearson
# iii) diagonal a curva de densidade
ggpairs(metricas_riqueza, upper = list(
                                  continuous = wrap("cor", size = 4))) +
                                  tema_livro()
```

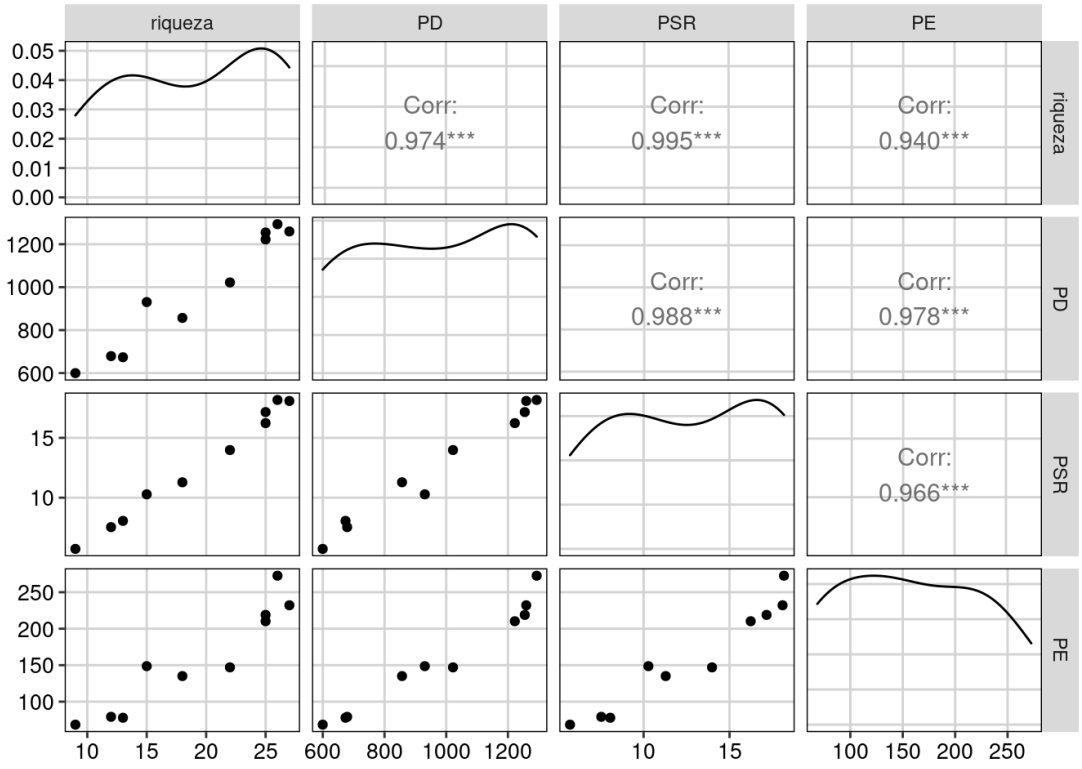


Figura 13.6: Correlação entre as métricas de riqueza da diversidade alfa filogenética.

Percebam que as três métricas apresentam correlações pareadas acima de 94%. Isso indica que as métricas são redundantes. Portanto, não há necessidade de calcular mais de uma métrica dentro da dimensão da riqueza filogenética. Além disso, as três métricas de diversidade alfa filogenética também apresentam alta correlação com a riqueza de espécies. Veja abaixo na seção de modelos nulos como controlar o efeito da riqueza de espécies nas métricas de diversidade filogenética.

Vamos avaliar a correlação entre os valores das métricas de diversidade alfa filogenética para a dimensão divergência (Figura 13.7).

```
## Data frame
# Vamos criar um data.frame com os resultados das métricas da dimensão
divergência.
metricas_divergencia <- data.frame(riqueza = resultados_PD$SR,
                                   MPD = resultados_MPD_PA,
                                   MPD_AB = resultados_MPD_AB,
                                   MNTD = resultados_MNTD_PA,
                                   MNTD_AB = resultados_MNTD_AB,
                                   PSV = resultados_PSV$PSVs)

## Gráfico
ggpairs(metricas_divergencia, upper = list(
                                  continuous = wrap("cor", size = 4))) +
                                  tema_livro()
```

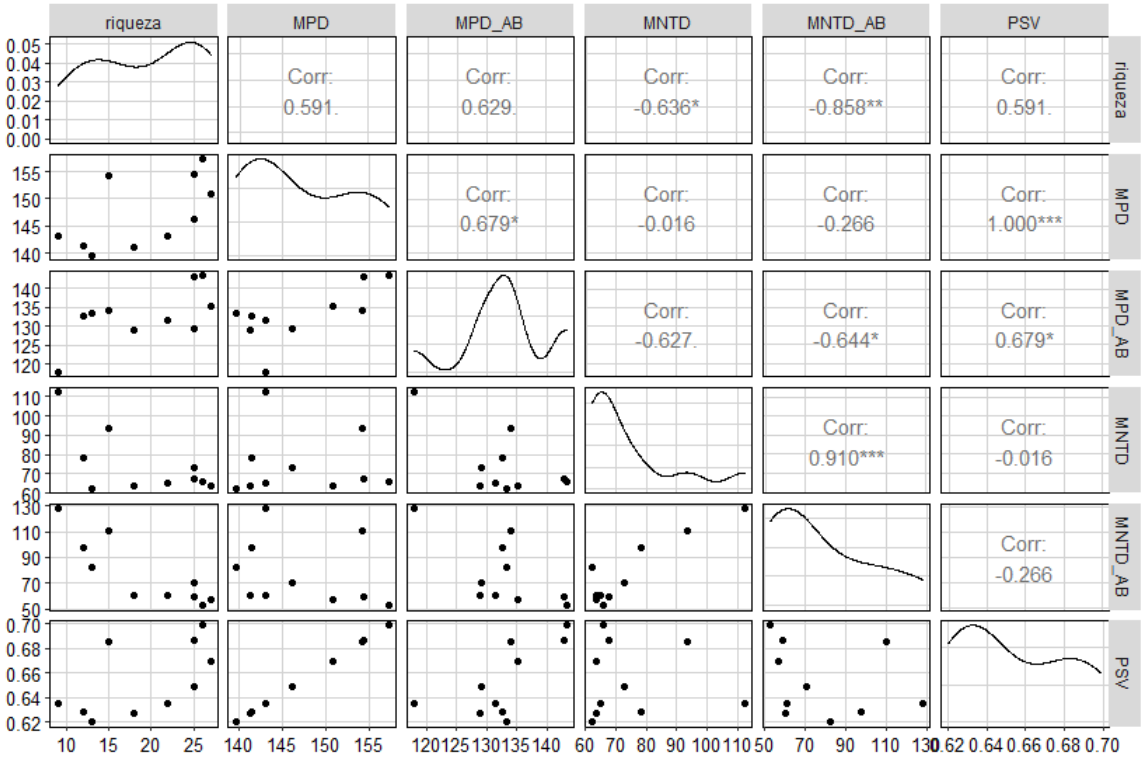


Figura 13.7: Correlação entre as métricas de divergência da diversidade alfa filogenética.

Como mencionado, as métricas MPD e PSV são idênticas quando usamos uma filogenia ultramétrica. Contudo, as métricas de divergência não apresentam correlações tão altas como as métricas da dimensão riqueza, com exceção do MNTD usando dados de incidência e abundância que foram fortemente correlacionados (`r = 0,9`). Além disso, estas métricas não são tão afetadas pela riqueza de espécies das comunidades como as métricas da dimensão riqueza.

13.3.5 Associação entre a diversidade alfa filogenética e o ambiente

Vamos avaliar e plotar a relação entre os valores de algumas métrica de diversidade alfa filogenética (variável resposta) e os valores de precipitação (variável preditora) (Figura 13.8).

```
## Dados
# Vamos inserir os dados de precipitação na planilha metrica_divergencia.
metricas_divergencia$precipitacao <- precipitacao_filogenetica$prec

## Gráficos
MPD_PA_plot <- ggplot(metricas_divergencia, aes(precipitacao, MPD)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    labs(x = "Precipitação (mm)",
         y = "Mean Pairwise Distance\n (MPD - Ausência e Presença)") +
    tema_livro()

MPD_AB_plot <- ggplot(metricas_divergencia, aes(precipitacao, MPD_AB)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    labs(x = "Precipitação (mm)",
         y = "Mean Pairwise Distance\n (MPD - Abundância)", size = 8) +
```

```
    tema_livro()

MNTD_AP_plot <- ggplot(metricas_divergencia, aes(precipitacao, MNTD)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    labs(x = "Precipitação (mm)",
        y = "Mean Nearest Taxon Distance\n (MNTD - Ausência e Presença)",
        size = 8) +
    tema_livro()

MNTD_AB_plot <- ggplot(metricas_divergencia,
                       aes(precipitacao, MNTD_AB)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Precipitação (mm)",
         y = "Mean Nearest Taxon Distance\n (MNTD - Abundância)",
         size = 8) +
    tema_livro()

ggarrange(MPD_PA_plot, MPD_AB_plot, MNTD_AP_plot, MNTD_AB_plot,
          ncol = 2, nrow = 2)
```

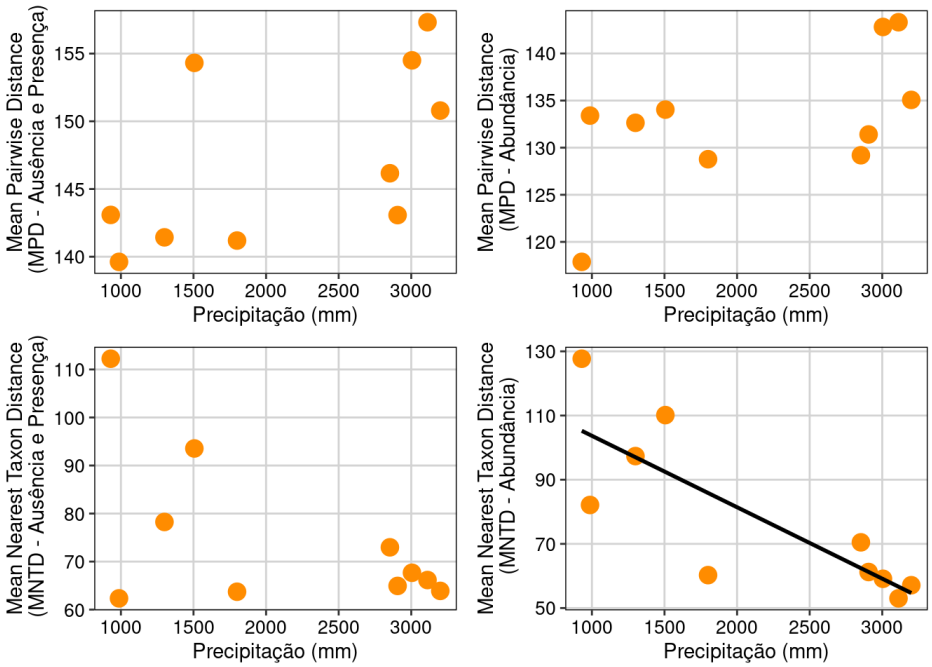


Figura 13.8: Relação de algumas métrica de diversidade alfa filogenética e valores de precipitação.

O MPD, que avalia as relações de parentesco mais internas da filogenia (i.e., relações entre espécies de famílias diferentes) não apresentou associação com o gradiente de precipitação. Por outro lado, o MNTD que avalia as relações mais terminais da filogenia (i.e., espécies dentro do mesmo gênero) apresentou uma relação negativa com o gradiente de precipitação. Interessante que a associação só foi significativa quando pesamos a análise pela abundância das espécies nas comunidades. Esses resultados demonstram a importância da seleção das métricas de diversidade filogenética e tipos

de dados (e.g., incidência ou abundância) utilizados na interpretação dos padrões observados na natureza.

Vamos ver os gráficos das métricas da dimensão riqueza da diversidade alfa filogenética (Figura 13.9).

```
## Dados
# Vamos inserir os dados de precipitação na planilha metrica_riqueza.
metricas_riqueza$precipitacao <- precipitacao$prec

## Gráficos
Riqueza_plot <- ggplot(metricas_riqueza, aes(precipitacao, riqueza)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Precipitação (mm)", y = "Riqueza de espécies") +
    tema_livro()

PD_plot <- ggplot(metricas_riqueza, aes(precipitacao, PD)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Precipitação (mm)",
         y = "Diversidade Filogenética\n (Faith)", size = 8) +
    tema_livro()

PSR_plot <- ggplot(metricas_riqueza, aes(precipitacao, PSR)) +
    geom_point(size = 4, shape = 19, col = "darkorange")  +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Precipitação (mm)",
         y = "Phylogenetic Species Richness\n (PSR)",
         size = 8) +
    tema_livro()

PE_plot <- ggplot(metricas_riqueza, aes(precipitacao, PE)) +
    geom_point(size = 4, shape = 19, col = "darkorange")  +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Precipitação (mm)",
         y = "Phylogenetic Endemism\n (PE)",
         size = 8) +
    tema_livro()

ggarrange(Riqueza_plot, PD_plot, PSR_plot, PE_plot, ncol = 2, nrow = 2)
```

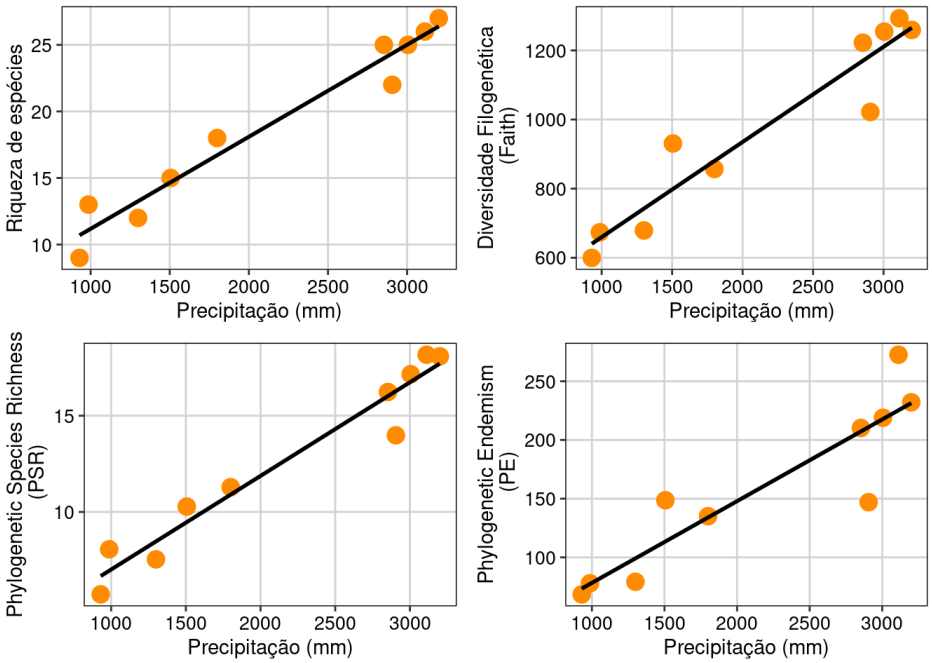


Figura 13.9: Relação de algumas métrica de diversidade alfa filogenética e valores de precipitação.

As três métricas de diversidade filogenética foram relacionadas com o gradiente de precipitação. Esse resultado indica que comunidades localizadas em áreas com maior precipitação anual abrigaram maior diversidade filogenética do que comunidades localizadas em áreas mais secas. Contudo, estas métricas são dependentes da riqueza de espécies nas comunidades. Veja abaixo a seção de modelos nulos para entender como lidar com essa dependência.

13.4 Métricas de diversidade beta filogenética

Métricas de diversidade beta filogenética utilizam dados de presença e ausência ou abundância das espécies para determinar um valor que representa a diferença entre comunidades em relação a história evolutiva das linhagens (para detalhes de diversidade beta, consulte o Capítulo 12).

13.4.1 Divergência da diversidade beta filogenética

Community Mean Pairwise Distance (COMDIST)

Esta métrica é uma extensão do MPD. COMDIST calcula a média da distância filogenética entre as espécies de duas comunidades (Webb et al. 2008). COMDIST pode ser calculada usando dados de incidência (presença e ausência) ou abundância das espécies. Esta extensão do MPD também é conhecida na literatura como Dpw (Swenson 2011, Swenson 2014).

Abaixo, demonstramos os códigos no R para o cálculo do COMDIST utilizando os dados das comunidades de aves.

```
## Community Mean Pairwise Distance (COMDIST)
# Análise com dados de presença e ausência das espécies nas comunidades.
resultados_Comdist_PA <- comdist(composicao_especies_P,
                                 cophenetic(filogenia_aves),
```

```
                                  abundance.weighted = FALSE)
resultados_Comdist_PA
#>           Com_1    Com_2    Com_3    Com_4    Com_5    Com_6    Com_7
Com_8    Com_9
#> Com_2  150.5242
#> Com_3  144.3300 148.7436
#> Com_4  149.4782 153.2992 147.5838
#> Com_5  141.8435 146.9829 139.9606 145.8746
#> Com_6  142.6019 148.3160 140.0376 143.1340 137.9527
#> Com_7  147.7189 150.2248 147.1703 150.3418 144.1481 145.2975
#> Com_8  141.3083 145.6684 138.5875 145.3897 137.0034 137.7062 144.4818
#> Com_9  141.6018 146.4697 138.3515 144.3480 136.9036 136.3453 144.3257
130.7628
#> Com_10 140.8810 145.9333 136.9978 145.1400 136.5394 137.2350 144.3660
130.0691 130.2321
```

As comunidades 8 e 10 apresentaram a menor média na distância filogenética (`130,06` m.a. - espécies de linhagens mais próximas) entre as espécies presente em cada comunidade, enquanto as comunidades 2 e 4 apresentaram a maior média na distância filogenética (`153.29` m.a. - espécies de linhagens mais distintas).

Vamos refazer a análise do COMDIST, mas desta vez, considerando a abundância das espécies de aves nas comunidades.

```
## Community Mean Pairwise Distance (COMDIST)
# Análise com dados de abundância das espécies nas comunidades.
resultados_Comdist_AB <- comdist(composicao_especies_P,
                                 cophenetic(filogenia_aves),
                                 abundance.weighted = TRUE)
```

Como no caso do MPD, pensar a abundância das espécies altera o padrão de distribuição dos valores de COMDIST. Neste caso, ao invés das comunidades 2 e 4, as comunidades 2 e 10 apresentam a maior média na distância filogenética (`155,85` m.a.).

Community Mean Nearest Taxon Distance (COMDISTNT)

Esta métrica é uma extensão do MNTD. COMDISTNT calcula a média da distância filogenética entre o táxon mais próximo das espécies de duas comunidades (Webb et al. 2008). COMDISTNT pode ser calculada usando dados de incidência ou abundância das espécies. Esta extensão do MNTD também é conhecida na literatura como Dnn (Swenson 2011).

Abaixo, demonstramos os códigos no R para o cálculo do COMDISTNT utilizando os dados das comunidades de aves.

```
## Community Mean Nearest Taxon Distance (COMDISTNT)
# Análise com dados de presença e ausência das espécies nas comunidades.
resultados_Comdistnt_PA <- comdistnt(composicao_especies_P,
                                     cophenetic(filogenia_aves),
                                     abundance.weighted = FALSE)
resultados_Comdistnt_PA
```

As comunidades 8 e 9 apresentaram a menor média na distância do vizinho mais próximo (`10,69` m.a. - espécies do mesmo gênero ou gêneros irmãos) entre as espécies presente em cada comunidade, enquanto as comunidades 7 e 10 apresentaram a maior média na distância entre vizinhos (`60,24` m.a. - espécies de linhagens distintas).

Vamos refazer a análise do COMDISTNT, mas desta vez, considerando a abundância das espécies de aves nas comunidades.

```
## Community Mean Nearest Taxon Distance (COMDISTNT)
# Análise com dados de abundância das espécies nas comunidades.
resultados_Comdistnt_AB <- comdistnt(composicao_especies_P,
                                     cophenetic(filogenia_aves),
                                     abundance.weighted = TRUE)
```

As comunidades 8 e 10 apresentaram a menor média na distância do vizinho mais próximo (`5,64` m.a.) entre as espécies presente em cada comunidade, enquanto as comunidades 6 e 10 apresentaram a maior média na distância entre vizinhos (`82,62` m.a.).

13.4.2 Correlação entre as métricas de diversidade beta filogenética

Vamos avaliar a correlação entre os valores das métricas da diversidade beta filogenética para a dimensão divergência (Figura 13.10).

```
## Dados
# Vamos criar um data frame com os resultados das métricas da dimensão
divergência.
metricas_divergencia_beta <- data.frame(
    COMDIST_PA = as.numeric(resultados_Comdist_PA),
    COMDIST_AB = as.numeric(resultados_Comdist_AB),
    COMDISTNT_PA = as.numeric(resultados_Comdistnt_PA),
    COMDISTNT_AB = as.numeric(resultados_Comdistnt_AB))

## Gráfico
ggpairs(metricas_divergencia_beta,
        upper = list(continuous = wrap("cor", size = 4))) +
    tema_livro()
```

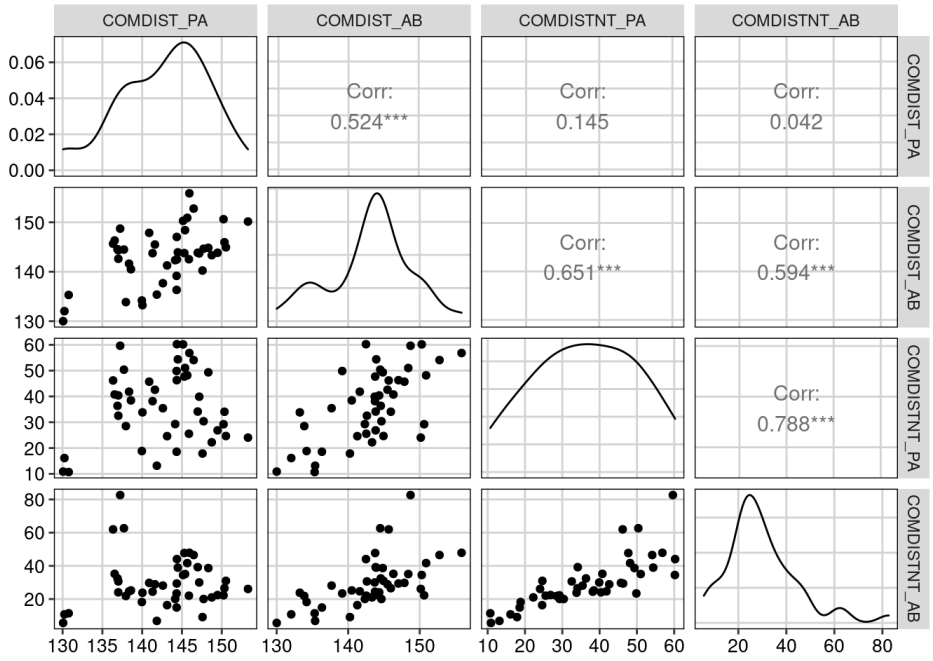


Figura 13.10: Correlação entre as métricas de divergência da diversidade beta filogenética.

Os valores das métricas de divergência filogenética beta apresentam correlações mais baixas do que as métricas da dimensão riqueza. Lembrem-se que COMDIST e COMDISTNT dão pesos diferentes para as relações de parentesco. COMDIST pesa as relações mais basais e internas da filogenia, enquanto COMDISTNT pesa as relações nas partes terminais da filogenia. Portanto, elas podem trazer informações complementares.

13.4.3 Associação entre a divergência da diversidade beta filogenética e o ambiente

Vamos avaliar e plotar a relação entre os valores de algumas métricas de divergência da diversidade beta filogenética (variável resposta) e os valores de precipitação (variável preditora) (Figura 13.11).

```
## Dados
# Precisamos calcular a dissimilaridade par a par da precipitação entre as
comunidades.
dis_prec <- vegdist(precipitacao, "euclidian")

# Vamos inserir estes dados na planilha metrica_divergencia_beta.
metricas_divergencia_beta$dis_prec <- as.numeric(dis_prec)

# Gráficos.
COMDIST_PA_plot <- ggplot(metricas_divergencia_beta,
                          aes(dis_prec, COMDIST_PA)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    labs(x = "Diferença na precipitação (mm)",
         y = "COMDIST\n (Presença e Ausência)") +
    tema_livro()
```

```
COMDIST_AB_plot <- ggplot(metricas_divergencia_beta,
                          aes(dis_prec, COMDIST_AB)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Diferença na precipitação (mm)",
         y = "COMDIST\n (Abundância)", size = 8) +
    tema_livro()

COMDISTNT_PA_plot <- ggplot(metricas_divergencia_beta,
                            aes(dis_prec, COMDISTNT_PA)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Diferença na precipitação (mm)",
         y = "COMDISTNT\n (Ausência e Presença)",
         size = 8) +
    tema_livro()

COMDISTNT_AB_plot <- ggplot(metricas_divergencia_beta,
                            aes(dis_prec, COMDISTNT_AB)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    labs(x = "Diferença na precipitação (mm)",
         y = " COMDISTNT\n (Abundância)",
         size = 8) +
    tema_livro()

ggarrange(COMDIST_PA_plot, COMDIST_AB_plot, COMDISTNT_PA_plot,
          COMDISTNT_AB_plot, ncol = 2, nrow = 2)
```

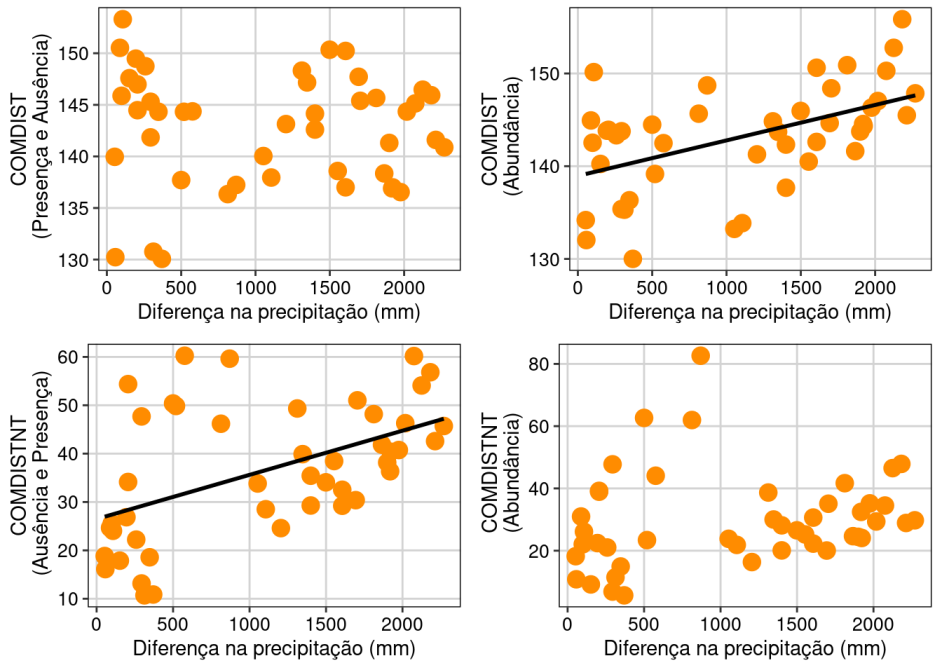


Figura 13.11: Relação de algumas métrica de divergência diversidade beta filogenética e valores de precipitação.

O COMDIST que avalia as relações de parentesco mais internas da filogenia (i.e., relações entre espécies de famílias diferentes) apresentou associação com o gradiente de precipitação quando avaliado pesado pela abundância das espécies. Por outro lado, o COMDISTNT que avalia as relações mais terminais da filogenia (i.e., espécies dentro do mesmo gênero) apresentou uma relação negativa com o gradiente de precipitação quando avaliado usando a incidência das espécies.

13.4.4 Riqueza da diversidade beta filogenética

Phylogenetic index of beta diversity (Phylosor)

Phylosor é uma métrica de similaridade e determina o comprimento total dos ramos da filogenia que é compartilhado entre pares de comunidades (Bryant et al. 2008).

Abaixo, demonstramos os códigos no R para o cálculo do Phylosor utilizando os dados das comunidades de aves.

```
## Phylogenetic index of beta diversity (Phylosor)
# Análise com dados de presença e ausência das espécies nas comunidades.
resultados_Phylosor <- phylosor(composicao_especies_P, filogenia_aves)

# Mostra uma matriz triangular com a similaridade entre a fração dos ramos
compartilhados entre duas comunidades
resultados_Phylosor
```

As espécies presentes nas comunidades 6 e 10 compartilham a menor porção do comprimento dos ramos da filogenia (53% - menor similaridade entre os pares de comunidades), enquanto as espécies presentes nas comunidades 8 e 10 compartilham grande parte dos comprimentos dos ramos da filogenia (91% - maior similaridade entre os pares de comunidades).

Unique Fraction metric (UniFrac)

UniFrac é uma métrica de dissimilaridade e determina a fração única da filogenia contida em cada uma das duas comunidades (Lozupone & Knight 2005).

Abaixo, demonstramos os códigos no R para o cálculo da UniFrac utilizando os dados das comunidades de aves.

```
## Unique Fraction metric (UniFrac)
# Análise com dados de presença e ausência das espécies nas comunidades.
resultados_UniFrac <- unifrac(composicao_especies_P, filogenia_aves)
```

As espécies presentes nas comunidades 6 e 10 apresentam a menor dissimilaridade (16,4 % - maior fração única da filogenia em cada comunidade), enquanto as espécies presentes nas comunidades 8 e 10 apresentam a maior dissimilaridade (63,36 % - maior compartilhamento de ramos da filogenia entre os pares de comunidades).

13.4.5 Correlação entre Phylosor e Unifrac

Vamos calcula a correlação entre as duas métricas de Riqueza da diversidade beta filogenética: Phylosor e Unifrac (Figura 13.12).

```
## Dados
# Vamos criar um data.frame com os resultados das métricas separados
# para as dimensões de riqueza e divergência.
metricas_riqueza_beta <- data.frame(
                            Phylosor = as.numeric(resultados_Phylosor),
                            UniFrac = as.numeric(resultados_UniFrac))

## Gráfico
ggpairs(metricas_riqueza_beta, upper=list(
                                continuous = wrap("cor", size = 4))) +
                                tema_livro()
```

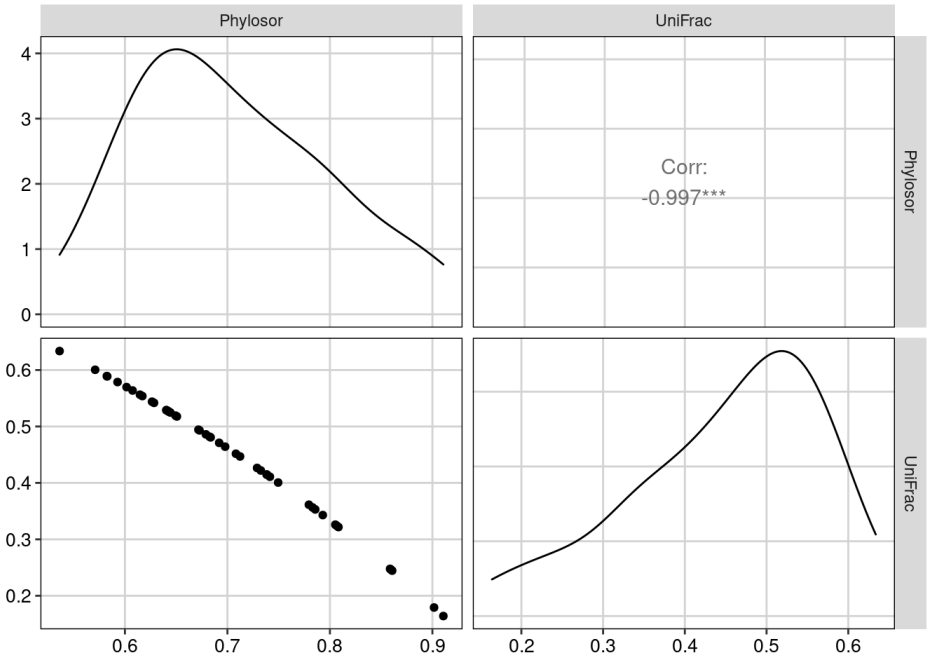


Figura 13.12: Correlação entre as métricas de riqueza da diversidade beta filogenética.

Importante
Os valores de Phylosor e UniFrac apresenta 99% de correlação entre eles. Portanto, essas duas métricas identificam padrões idênticos e não devem ser utilizadas simultaneamente.

13.4.6 Associação entre a riqueza da diversidade beta filogenética e o ambiente

Vamos avaliar e plotar a relação entre os valores de algumas métricas de riqueza da diversidade beta filogenética (variável resposta) e os valores de precipitação (variável preditora) (Figura 13.13).

```
## Dados
# Vamos inserir os dados de precipitação na planilha metrica_riqueza_beta.
metricas_riqueza_beta$dis_prec <- as.numeric(dis_prec)
```

```
## Gráficos
# Phylosor.
plot_phylosor <- ggplot(metricas_riqueza_beta, aes(dis_prec, Phylosor)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    scale_y_continuous(limits = c(0, 1.0)) +
    labs(x = "Diferença na precipitação (mm)",
         y = "Phylosor", size = 8) +
    tema_livro()

# Unifrag.
plot_unifrac <- ggplot(metricas_riqueza_beta, aes(dis_prec, UniFrac)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    scale_y_continuous(limits = c(0, 1.0)) +
    labs(x = "Diferença na precipitação (mm)",
         y = "UniFrac", size = 8) +
    tema_livro()

ggarrange(plot_phylosor, plot_unifrac, ncol = 2)
```

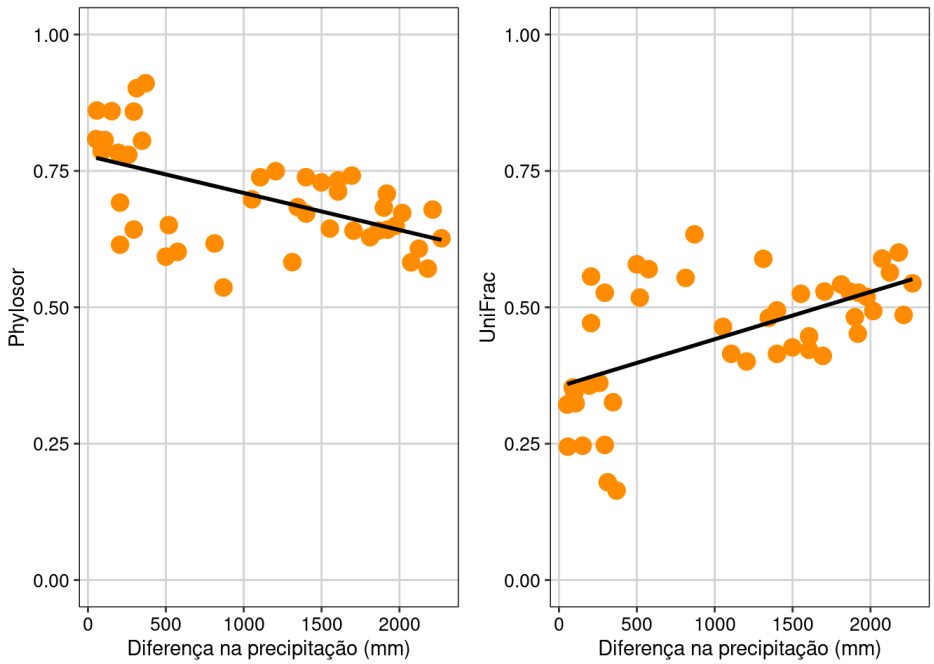


Figura 13.13: Relação de algumas métrica de riqueza diversidade beta filogenética e valores de precipitação.

Phylosor (similaridade) e UniFrac (dissimilaridade) foram relacionadas com o gradiente de precipitação. Comunidades com quantidade de precipitação parecidas abrigaram linhagens similares enquanto comunidades que recebem quantidade de precipitação diferentes abrigam linhagens mais distintas.

13.4.7 Partição da diversidade beta filogenética

As métricas, Phylosor e UniFrac, podem ser particionadas em dois componentes (Baselga 2010, Leprieur et al. 2012): i) substituição (do inglês *turnover*) de espécies entre as comunidades; e ii) componente de aninhamento (do inglês *nestedness*) que representa a perda ou ganho de espécies entre comunidades atribuídos a diferença na riqueza de espécies. A partição da diversidade beta nestes componentes permite avaliar diferentes hipóteses sobre os processos e mecanismos atuando na montagem de comunidades.

Abaixo, demonstramos os códigos no R para o cálculo da partição da diversidade beta filogenética utilizando os dados das comunidades de aves.

```
## Partição
# Temos que transformar os dados para presença e ausência das espécies nas
comunidades.
dados_PA <- decostand(composicao_especies_P, "pa")

# Partição dos componentes do Phylosor.
resultados_Phylosor_particao <- phylo.beta.pair(dados_PA,
                                                filogenia_aves,
                                                index.family = "sorensen")
resultados_Phylosor_particao
# Resultado tem três matrizes:
# i) dissimilaridade total (phylo.beta.sor);
# ii) componente substituição de espécies (phylo.beta.sim); e
# iii) componente aninhamento (phylo.beta.sne).
```

Vamos refazer a análise para UniFrac.

```
# Partição dos componentes do UniFrac.
resultados_UniFrac_particao <- phylo.beta.pair(dados_PA,
                                                filogenia_aves,
                                                index.family = "jaccard")
resultados_UniFrac_particao
# Resultado tem três matrizes:
# i) dissimilaridade total (phylo.beta.jac);
# ii) componente substituição de espécies (phylo.beta.jtu); e
# iii) componente aninhamento (phylo.beta.jne).
```

Gráfico com os resultados dos componentes substituição e aninhamento da diversidade beta filogenética - Phylosor (Figura 13.14).

```
## Dados
# Vamos preparar os dados para o gráfico.
particao_phylosor <- data.frame(
  substituicao = as.numeric(resultados_Phylosor_particao$phylo.beta.sim),
  aninhamento = as.numeric(resultados_Phylosor_particao$phylo.beta.sne),
  sorensen = as.numeric(resultados_Phylosor_particao$phylo.beta.sor),
  dis_prec = as.numeric(dis_prec))
```

```
## Gráficos
sorensen_plot <- ggplot(particao_phylosor,
                        aes(dis_prec, sorensen)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "", y = "Sorensen") +
    tema_livro()

subst_plot <- ggplot(particao_phylosor,
                     aes(dis_prec, substituicao)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "Diferença na precipitação\n (mm)",
         y = "Componente Substituição", size = 8) +
    tema_livro()

aninha_plot <- ggplot(particao_phylosor,
                      aes(dis_prec, aninhamento)) +
    geom_point(size = 4, shape = 19, col = "darkorange") +
    geom_smooth(method = lm, se = FALSE, color = "black") +
    labs(x = "", y = "Componente aninhamento", size = 8) +
    tema_livro()

ggarrange(sorensen_plot, subst_plot, aninha_plot,
          ncol = 3, nrow = 1)
```

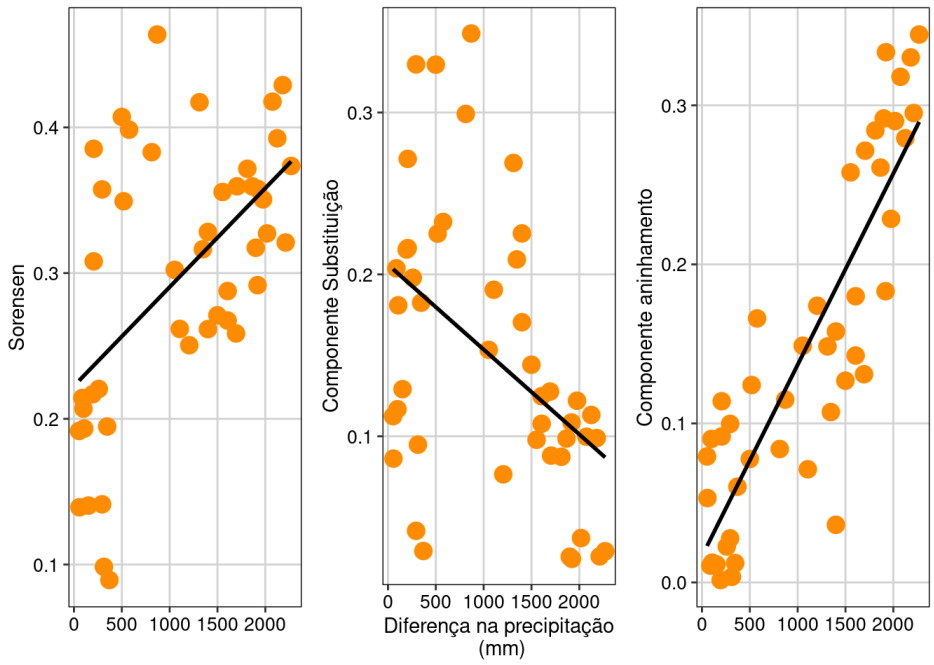


Figura 13.14: Relação dos componentes substituição e aninhamento da diversidade beta filogenética - Phylosor e valores de precipitação.

Percebam que o componente substituição é maior entre comunidades que apresentam diferenças altas na quantidade de precipitação, enquanto o componente aninhamento é maior entre as comunidades que apresentam quantidade similar de precipitação.

13.5 Modelos Nulos

Em muitos casos, os valores de diversidade filogenética são correlacionados com a riqueza de espécies nas comunidades. Por exemplo, se um pesquisador relata que duas comunidades apresentam diferentes valores de PD, é impossível saber se esta diferença é simplesmente porque elas têm diferentes valores de riqueza de espécies ou se há algum fator fundamental sobre a informação filogenética que é importante. Outra questão abordada nos estudos de montagem das comunidades é saber se os valores observados para as métricas (e.g., MPD ou MNTD) relacionadas com a estrutura filogenética das comunidades seriam diferentes se a colonização das espécies do *pool* regional fosse aleatória? Os modelos nulos respondem a estas perguntas. Contudo, a definição do *pool* regional não é uma tarefa trivial (Lessard et al. 2012, Carstensen et al. 2013).

Os modelos nulos são construídos considerando processos ecológicos ou evolutivos de interesse. Eles geram padrões que são baseados na aleatorização dos dados ecológicos ou amostragens aleatórias de uma distribuição conhecida ou hipotética (Gotelli & Graves 1996). Neste caso, alguns elementos dos dados (como colunas ou linhas) são mantidos constantes, e outros são permitidos variar aleatoriamente para criar novos padrões. O principal motivo para a construção de modelos nulos é produzir um padrão que seria esperado na ausência de um mecanismo ecológico específico (Gotelli & Graves 1996). Contudo, ressaltamos que os modelos nulos podem revelar padrões não comuns, mas eles não podem determinar os mecanismos responsáveis por gerar estes padrões (Gotelli & Graves 1996).

Os modelos nulos empregados para contrapor os padrões observados pelas métricas de diversidade filogenética utilizam a aleatorização dos dados de duas formas principais: i) aleatorizando o nome das espécies na árvore filogenética mantendo a estrutura e composição da matriz de co-ocorrência das espécies e o comprimento dos ramos da árvore inalterados; e ii) aleatorizando as linhas e/ou colunas da matriz de co-ocorrência das espécies (Gotelli 2000, Ulrich & Gotelli 2010). De forma geral, nas análises de diversidade filogenética as aleatorizações são repetidas 999 vezes (pode ser mais ou menos, a critério do pesquisador) e calcula-se a média e o desvio padrão dos valores gerados pelos modelos. Com estes dados, calcula-se o tamanho do efeito padronizado (do inglês *Standardized Effect Size* - SES) utilizando a seguinte fórmula:

- SES = (valor observado - média dos valores gerados na aleatorização)/ desvio padrão dos valores gerados na aleatorização

Os valores de SES são utilizados para rejeitar ou não a hipótese nula de que o padrão observado difere do esperado ao acaso. Contudo, tenha em mente que a definição do esquema de aleatorização dos modelos nulos não é meramente uma questão técnica (Götzenberger et al. 2012). A definição do esquema de aleatorização irá determinar quais os mecanismos ecológicos são permitidos ou excluídos no modelo nulo (Götzenberger et al. 2012). Consequentemente, ele estará avaliando diferentes hipóteses nulas.

Abaixo, demonstramos os códigos no R para calcular os modelos nulos para as métricas de diversidade filogenética.

Nearest Relative Index (NRI) ou Standardized Effect Size of MPD

Esta métrica calcula o tamanho do efeito padronizado para a métrica MPD. Contudo, NRI é calculado multiplicando os resultados do SES por -1. Valores positivos de NRI indicam agrupamento filogenético e valores negativos de NRI indicam dispersão filogenética (Webb et al. 2008).

Veja a ajuda desta função usando `?ses.mpd()` para ver todas as possibilidades de modelos nulos disponíveis.

```
## NRI ou SES_MPD
resultados_SES_MPD <- ses.mpd(composicao_especies_P,
                              cophenetic(filogenia_aves),
                              null.model = "taxa.labels",
                              abundance.weighted = FALSE,
                              runs = 999)

# Mostra a riqueza de espéices, MPD observado, média e desvio padrão dos
# valores de MPD das aleatorizações, SES e o valor de p.
head(resultados_SES_MPD)
#>       ntaxa  mpd.obs mpd.rand.mean mpd.rand.sd mpd.obs.rank  mpd.obs.z mpd.obs.p runs
#> Com_1    27 150.7914      153.9953    3.891718          215 -0.8232597     0.215  999
#> Com_2    26 157.3158      153.5833    4.367399          770  0.8546215     0.770  999
#> Com_3    25 146.1622      153.9931    4.476428           52 -1.7493791     0.052  999
#> Com_4    25 154.5005      153.5113    4.720348          551  0.2095591     0.551  999
#> Com_5    22 143.0727      153.7490    5.294432           24 -2.0165115     0.024  999
#> Com_6    18 141.1926      153.9709    6.951990           34 -1.8380846     0.034  999
```

Somente as comunidades 5 e 6 apresentaram valores de $p < 0.05$ indicando que os resultados observados de MPD são menores que o esperado ao acaso (i.e., valores simulados). Neste caso, a composição de espécies presentes nessas duas comunidades apresenta agrupamento filogenético. Por outro lado, os valores de MPD observados para as outras comunidades são similares aos valores obtidos para comunidades simuladas com a redistribuição dos nomes das espécies na filogenia.

Nearest Taxon Index (NTI) ou Standardized Effect Size of MNTD

Esta métrica calcula o tamanho do efeito padronizado para a métrica MNTD. Contudo, NTI é calculado multiplicando os resultados do SES por -1. Valores positivos de NTI indicam agrupamento filogenético e valores negativos de NTI indicam dispersão filogenética (Webb et al. 2008).

```
## NTI ou SES_MNTD
resultados_SES_MNTD <- ses.mntd(composicao_especies_P,
                                cophenetic(filogenia_aves),
                                null.model = "taxa.labels",
                                abundance.weighted = FALSE,
                                runs = 999)

# Mostra a riqueza de espéices,MNTD observado, média e desvio padrão dos
# valores de MNTD das aleatorizações, SES e o valor de p.
head(resultados_SES_MNTD)
#>       ntaxa mntd.obs mntd.rand.mean mntd.rand.sd mntd.obs.rank  mntd.obs.z mntd.obs.p runs
```

```
#> Com_1    27 63.89727       63.30504     6.864478          518  0.08627467      0.518  999
#> Com_2    26 66.15828       64.81499     7.217753          575  0.18610860      0.575  999
#> Com_3    25 72.96912       65.76333     7.754651          811  0.92922217      0.811  999
#> Com_4    25 67.67170       65.70886     7.752673          600  0.25318258      0.600  999
#> Com_5    22 64.93477       69.46545     9.133114          305 -0.49607138      0.305  999
#> Com_6    18 63.72337       76.12099    11.819687          162 -1.04889562      0.162  999
```

Somente a comunidade 9 apresentou valor de p < 0.05 indicando que o resultado observado de MNTD foi menor que o esperado ao acaso (i.e., valores simulados). Neste caso, a composição de espécies presente nessa comunidade apresenta agrupamento filogenético. Por outro lado, os valores de MNTD observados para as outras comunidades são similares aos valores obtidos para comunidades simuladas com a redistribuição dos nomes das espécies na filogenia.

Standardized Effect Size of PD

Esta métrica calcula o tamanho do efeito padronizado para a métrica PD (Webb et al. 2008).

```
## SES_PD
resultados_SES_PD <- ses.pd(composicao_especies_P, filogenia_aves,
                            null.model = "independentswap",
                            runs = 999)

# Mostra a riqueza de espéices,MNTD observado, média e desvio padrão dos
# valores de PD das aleatorizações, SES e o valor de p.
head(resultados_SES_PD)
#>       ntaxa   pd.obs pd.rand.mean pd.rand.sd pd.obs.rank   pd.obs.z pd.obs.p runs
#> Com_1    27 1259.315     1272.480   66.74674         429 -0.1972372    0.429  999
#> Com_2    26 1293.152     1238.690   68.04238         799  0.8004172    0.799  999
#> Com_3    25 1222.310     1201.471   65.51971         624  0.3180526    0.624  999
#> Com_4    25 1254.541     1205.879   68.34858         750  0.7119674    0.750  999
#> Com_5    22 1021.967     1096.024   67.73466         129 -1.0933409    0.129  999
#> Com_6    18  856.781      950.239   64.54864          76 -1.4478696    0.076  999
```

Nenhuma comunidade apresentou valor de p < 0.05. Neste caso, os valores observados de PD são similares aos valores obtidos para comunidades simuladas com a redistribuição dos nomes das espécies na filogenia.

Standardized effect size do Phylosor

Não há pacotes que calculam o SES para a métrica Phylosor. Assim, iremos usar a função `phylosor.rnd()` para criar modelos nulos para o Physolor, e em seguida, iremos usar uma função criada por Pedro Braga & Katherine Hébert para calcular os valores de SES e os valores de P.

```
## Standardized effect size do Phylosor
# Modelo nulo que rearranja o nome das espécies na filogenia.
modelos_nulo <- phylosor.rnd(composicao_especies_P, filogenia_aves,
                             null.model = "taxa.labels", runs = 9)

# Função para calcular o SES eo valor de P.
ses.physo <- function(obs, nulo_phylosor){
    nulo_phylosor <- t(as.data.frame(lapply
```

```
                                        (nulo_phylosor, as.vector)))
    physo.obs <- as.numeric(obs)
    physo.mean <- apply(nulo_phylosor, MARGIN = 2,
                        FUN = mean, na.rm = TRUE)
    physo.sd <- apply(nulo_phylosor, MARGIN = 2,
                      FUN = sd, na.rm = TRUE)
    physo.ses <- (physo.obs - physo.mean)/physo.sd
    physo.obs.rank <- apply(X = rbind(physo.obs,
                                      nulo_phylosor), MARGIN = 2,
                            FUN = rank)[1, ]
    physo.obs.rank <- ifelse(is.na(physo.mean), NA,
                             physo.obs.rank)
    data.frame(physo.obs, physo.mean, physo.sd,
               physo.obs.rank, physo.ses,
               physo.obs.p = physo.obs.rank/
                   (dim(nulo_phylosor)[1] + 1))
}

## Resultados
resultados <- ses.physo (resultados_Phylosor, modelos_nulo)
head(resultados)
#>   physo.obs physo.mean   physo.sd physo.obs.rank  physo.ses physo.obs.p
#> 1 0.7856828  0.7915561 0.04011508             6 -0.1464099         0.6
#> 2 0.8052839  0.8813422 0.03356047             1 -2.2663056         0.1
#> 3 0.7831520  0.8174306 0.04545519             2 -0.7541175         0.2
#> 4 0.8586780  0.8687616 0.03659588             5 -0.2755416         0.5
#> 5 0.6717414  0.7460991 0.02806215             1 -2.6497526         0.1
#> 6 0.7414284  0.6904007 0.04842748             9  1.0536941         0.9
```

Nenhum valor de similaridade entre pares comunidade apresentou valor de $p < 0.05$. Neste caso, os valores de Phylosor observados são similares aos valores obtidos para as comunidades simuladas com a redistribuição dos nomes das espécies na filogenia.

13.6 Para se aprofundar

13.6.1 Livros

- Recomendamos aos interessados(as) os livros: i) Swenson (2014) *Functional and Phylogenetic Ecology in R*; ii) Paradis (2012) *Analysis of Phylogenetics and Evolution in R*; iii) Cadotte & Davies (2016) *Phylogenies in Ecology*, iv) Gotelli & Graves (1996) *Null Models in Ecology*; e v) Magurran & McGill (2011) *Biological Diversity Frontiers in Measurement and Assessment*.

13.6.2 Links

O blog Ferramentas filogenéticas para biologia comparada do pesquisador Liam Revell é uma ferramenta excelente para obter informações e aplicações das análises filogenéticas diretamente no R.

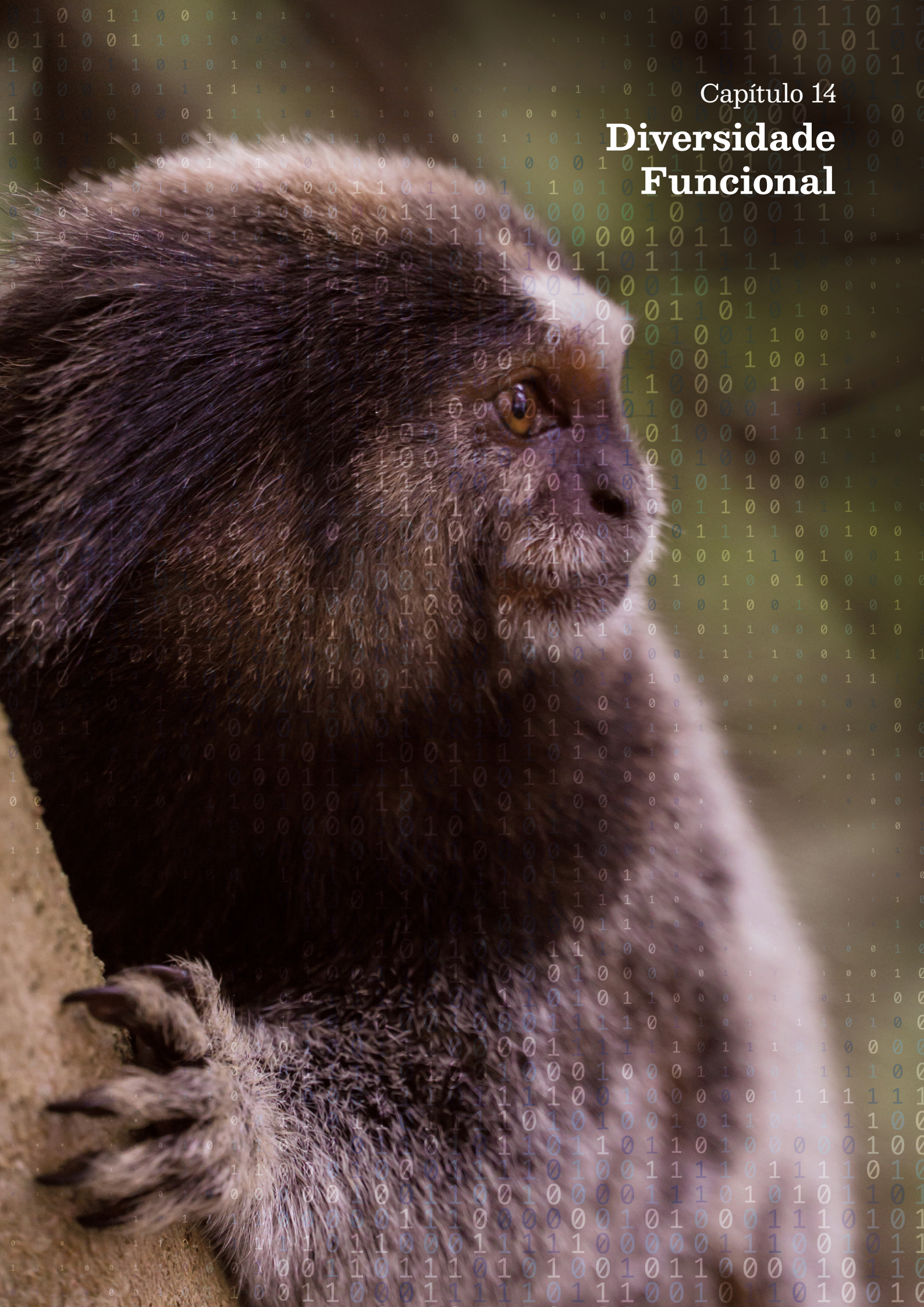
13.7 Exercícios

13.1 Carregue os dados - `anuros_composicao` (i.e., 211 espécies de anuros coletados em 44 localidades na Mata Atlântica), `anuros_ambientais` (i.e., variáveis climáticas, topográficas e coordenadas geográficas) e `filogenia_anuros` (filogenia das 211 espécies) - que estão no pacote `ecodados`. Use a função `varpart()` do pacote `vegan` para testar a importância relativa dos efeitos da precipitação anual, range altitudinal e temperatura anual na distribuição espacial da diversidade filogenética (PD) e Endemismo filogenético (PE). Calcule o SES para verificar se os resultados da diversidade filogenética (PD) diferem do esperado ao acaso devido ao número de espécies em cada comunidade. Qual a sua interpretação sobre os resultados?

13.2 Carregue os dados - `anuros_composicao` (i.e., 211 espécies de anuros coletados em 44 localidades na Mata Atlântica), `anuros_ambientais` (i.e., variáveis climáticas, topográficas e coordenadas geográficas) e `filogenia_anuros` (filogenia das 211 espécies) - que estão no pacote ecodados. Use a função `varpart()` do pacote `vegan` para testar a importância relativa dos efeitos da precipitação anual, range altitudinal e temperatura anual na distribuição espacial do NRI e NTI. Qual a sua interpretação sobre os resultados?

13.3 Carregue os dados - `anuros_composicao` (i.e., 211 espécies de anuros coletados em 44 localidades na Mata Atlântica), `anuros_ambientais` (i.e., variáveis climáticas, topográficas e coordenadas geográficas) e `filogenia_anuros` (filogenia das 211 espécies) - que estão no pacote ecodados. Use a função `varpart()` do pacote `vegan` para testar a importância relativa dos efeitos da precipitação anual, range altitudinal e distância geográfica na distribuição espacial dos diferentes componentes da diversidade beta filogenética (Phylosor). Qual a sua interpretação sobre os resultados?

Soluções dos exercícios.



Capítulo 14

Diversidade Funcional

Pré-requisitos do capítulo

Pacotes, dados e funções que serão utilizados neste capítulo.

```
## Pacotes
library(FD)
library(ade4)
library(ecodados)
library(gridExtra)
library(ggplot2)
library(ggrepel)
library(tidyverse)
library(picante)
library(vegan)
library(SYNCSA)
library(GGally)
library(FD)
library(betapart)
library(nlme)
library(ape)
library(TPD)
library(cati)
library(kableExtra)

## Dados e funções
comun_fren_dat <- ecodados::fundiv_frenette2012a_comu
ambie_fren_dat <- ecodados::fundiv_frenette2012a_amb
trait_fren_dat <- ecodados::fundiv_frenette2012a_trait
trait_dat      <- ecodados::fundiv_barbaro2009a_trait
comun_dat      <- ecodados::fundiv_barbaro2009a_comu
ambie_dat      <- ecodados::fundiv_barbaro2009a_amb
trait_baselga  <- ecodados::trait_baselga
comm_baselga   <- ecodados::comm_baselga
anuros_comm    <- ecodados::anuros_comm
traits         <- ecodados::traits
env            <- ecodados::env
# ecodados::wITV # funtion: wITV
```

14.1 Aspectos teóricos

Até a década de 1990, a teoria ecológica investigava basicamente quais processos determinavam a abundância e riqueza de espécies no espaço e tempo. As décadas de 1980 e 1990 foram marcadas por intensos debates sobre as regras de montagem de comunidades e como interações e filtros ambientais determinavam a coexistência de espécies (Strong et al. 1984). Porém, a década de 2000 foi marcada pelo uso mais explícito da características das espécies como uma variável fundamental tanto para explicar como a distribuição dos organismos seria afetada pelo ambiente, quanto para entender como

tais espécies afetariam o ecossistema (Díaz & Cabido 2001, McGill et al. 2006). O primeiro estudo que utilizou o termo *Diversidade Funcional* foi publicado por Williams (1967), que comparou espécies de naúplios filogeneticamente relacionadas e demonstrou que elas possuem alta plasticidade funcional que favorecem ampla variação de comportamentos e, desse modo, permitem que sejam espécies generalistas em ambientes em constante mudança. A unidade básica desses estudos, o atributo funcional (do inglês "*functional trait*"), é definido como uma propriedade mensurável dos organismos (geralmente em nível individual) que represente características morfológicas, fisiológicas ou fenológicas que afetam a aptidão alterando aspectos do crescimento, reprodução e sobrevivência (Violle et al. 2007). Mais especificamente, o atributo funcional pode ser divido em **atributo efeito** (i.e., atributos do organismo que afetam condições ambientais ou propriedades do ecossistema) e **atributo resposta** (i.e., atributos do organismo que variam em resposta a condições ambientais) (Violle et al. 2007).

Dessa forma, as medidas de diversidade passaram a ser representadas não somente por diferenças no número e na quantidade de espécies, mas pelas diferenças e/ou semelhanças dos atributos funcionais das espécies dentro e entre localidades. Assim, a variação no grau de expressão de diferentes atributos funcionais entre diferentes populações, comunidades ou ecossistemas é definida como *Diversidade Funcional* (*sensu* Garnier et al. 2015). Porém, a diversidade funcional não deve ser usada como medida única, uma vez que tais diferenças entre os atributos funcionais podem ser medida a partir da abundância relativa, riqueza e variação dos atributos funcionais. Desse modo, podemos dividir a diversidade funcional em três diferentes medidas: i) riqueza funcional, ii) divergência funcional, e iii) regularidade funcional (Villéger et al. 2008). Existem dezenas de métricas que calculam cada uma dessas *dimensões* da diversidade funcional, mas se destacam aquelas baseadas em dendrograma (e.g., FD: Petchey and Gaston 2002) ou em medidas de distância (e.g., Villéger et al. 2008). Assim como a diversidade taxonômica (Capítulo 12), a diversidade funcional pode ser medida em componentes alfa e beta. A seguir, apresentamos diferentes maneiras de calcular a distância entre localidades tendo como base os atributos funcionais das espécies e, além disso, demonstramos como calcular algumas das métricas de diversidade (alfa e beta) funcional mais usadas em Ecologia. A parte final deste capítulo apresenta dois exemplos de como podemos testar hipóteses ecológicas comparando a diversidade funcional alfa e beta.

14.2 Definindo a dis(similaridade) entre espécies

Definir o quão diferente ou semelhante são duas espécies que ocorrem em uma determinada localidade é a base para calcular a diversidade alfa e beta funcional. Para isso, é fundamental ter em mente que os atributos funcionais podem ser de vários tipos como, por exemplo, contínuos (e.g., tamanho corporal em centímetro), categóricos (e.g., guilda: frugívoro, detritívoro, etc.), ordinais (e.g., 1 para organismo até 5 cm, 2 para organismos entre 5 e 30 cm, e 3 para organismos maiores do que 30 cm), binários (e.g., presença ou ausência de espinho), entre outros [veja a Figura 2.1 no Capítulo 2]. Por este motivo, a decisão do método de distância só será possível após o reconhecimento dos tipos de atributos funcionais escolhidos. Em linhas gerais, para variáveis contínuas a distância euclidiana é a melhor opção, enquanto para os outros tipos de variáveis ou para conjuntos de atributos com mais de um tipo de variável, a distância de Gower geralmente deve ser a melhor opção (Pavoine et al. 2009).

Exemplo prático

Exemplo1: variáveis contínuas

Vamos utilizar um conjunto de dados com atributos contínuos (e.g., área foliar específica e massa foliar seca) de 34 espécies de plantas em um gradiente de aridez (Frenette-Dussault et al. 2012). Diversas análises funcionais podem ser afetadas por valores extremos ou pela diferença de unidade/escala entre as variáveis utilizadas. Por este motivo, é importante padronizar a matriz de atributos com média 0 e desvio padrão 1. Esta padronização é necessária tanto para fazer uma PCA como para PCoA (veja Capítulo 9).

Pergunta

Quais são as espécies de plantas mais semelhantes? (Neste caso, sem predição, pois representa uma avaliação exploratória com as características funcionais das espécies).

Variáveis

- Dependentes: atributos funcionais (matriz de atributos contínuos por espécie: `trait_fren_dat`)

Análises

Aqui realizamos os passos para uma ordenação usando os atributos funcionais contínuos (Figura 14.1).

```
## PCoA dos atributos contínuos

# 1. Padronização dos dados
trait_pad <- decostand(trait_fren_dat, "standardize")
euclid_dis <- vegdist(trait_pad, "euclidean")

# 2. PCoA
# Resultados são idênticos aos resultados de uma PCA.
pcoa_traits_cont <- pcoa(euclid_dis, correction = "cailliez")

# 3. Exportandos dados para gráfico
# Ao usar '[,1:2]' você irá selecionar os dois primeiros eixos.
eixos_cont <- as.data.frame(pcoa_traits_cont$vectors[, 1:2])

# 4. Gráfico de ordenação
plot_trait_cont <- ggplot(eixos_cont, aes(x = Axis.1, y = Axis.2)) +
    geom_point(pch = 21, size = 4, color = "black", alpha = 0.7,
               fill = "red2") +
    geom_text_repel(aes(Axis.1, Axis.2, label = rownames(eixos_cont))) +
    geom_hline(yintercept = 0, linetype = 2) +
    geom_vline(xintercept = 0, linetype = 2) +
    labs(x = "PCO 1", y = "PCO 2", title = "Dados contínuos") +
    tema_livro()
plot_trait_cont
```

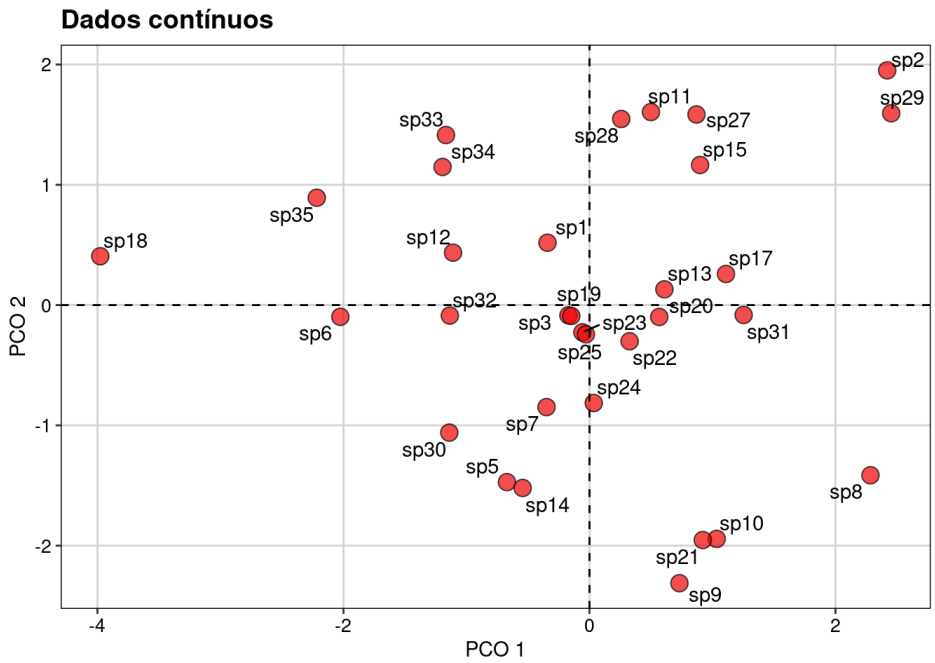


Figura 14.1: Ordenação usando os atributos funcionais contínuos.

Exemplo 2: variáveis categóricas

No próximo exemplo, utilizamos atributos funcionais de besouros distribuídos na Europa (Barbaro & Van Halder 2009). Esses dados são categóricos e incluem atributos como período de atividade (noturno, diurno, dioturno), tendência da população na Europa (estável, aumentando, diminuindo) entre outros.

> **Importante**
> Ao contrário dos dados contínuos, para dados categóricos não é possível utilizar PCA.

Pergunta

Quais são as espécies de besouros mais semelhantes? (Neste caso, sem predição, pois representa uma avaliação exploratória com as características funcionais das espécies).

Variáveis

- Dependentes: atributos funcionais (matriz de atributos categóricos por espécie: `trait_dat`)

Análises

Aqui realizamos os passos para uma ordenação usando os atributos funcionais categóricos (Figura 14.2).

```
## PCoA dos atributos categóricos

# 1. Selecionar somente os atributos categóricos
trait_cat <- trait_dat %>%
    dplyr::select_if(is.character)
```

```
# 2. Calcular a distância de Gower
dist_categ <- gowdis(trait_cat)

# 3. PCoA da matriz de distância funcional (Gower)
pcoa_traits_cat <- pcoa(dist_categ, correction = "cailliez")

# 4. Exportar dados (escores) para ggplot
eixos_cat <- as.data.frame(pcoa_traits_cat$vectors[,1:2]) # Selecionar os
dois primeiros eixos

# 5. Gráfico de ordenação
plot_trait_cat <- ggplot(eixos_cat, aes(x = Axis.1, y = Axis.2)) +
    geom_point(pch = 21, size = 4, alpha = 0.7, color = "black",
               fill = "cyan4") +
    geom_text_repel(aes(Axis.1, Axis.2, label = rownames(eixos_cat))) +
    geom_hline(yintercept = 0, linetype = 2) +
    geom_vline(xintercept = 0, linetype = 2) +
    labs(x = "PCO 1", y = "PCO 2", title = "Dados categóricos") +
    tema_livro()
plot_trait_cat
```

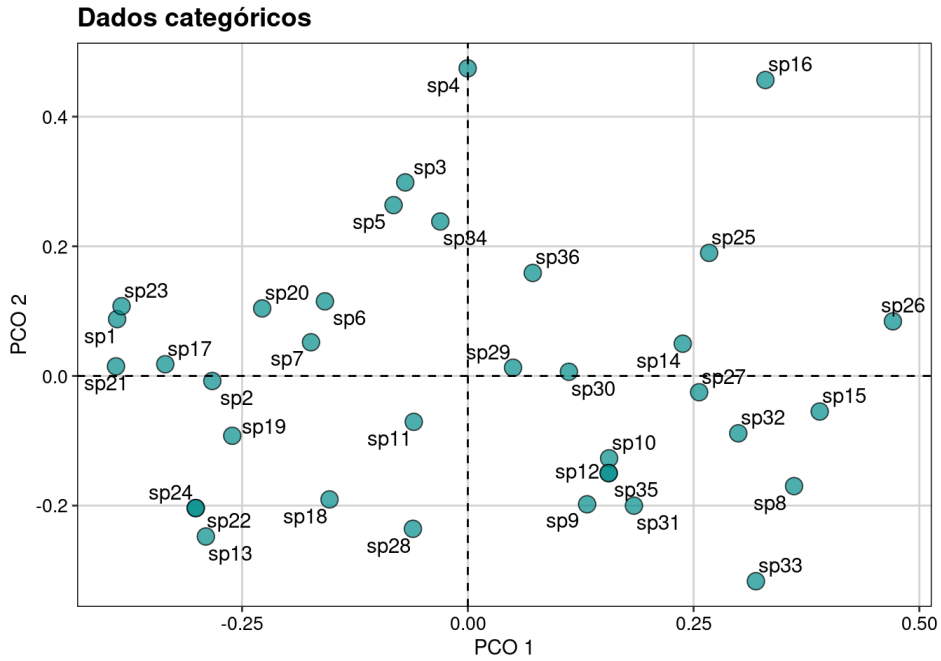


Figura 14.2: Ordenação usando os atributos funcionais categóricos.

Exemplo 3: variáveis mistas

Em casos mais complexos, a pesquisa inclui diversos atributos funcionais com naturezas diferentes, como atributos contínuos, categóricos, ordinais, circulares, entre outros. Desse modo, é possível utilizar medidas como Gower (`gowdis()`). Porém, existe uma alternativa mais apropriada que generalizou o coeficiente de Gower para tratar cada conjunto de variáveis de acordo com sua natureza

(Pavoine et al. 2009). Vamos usar o mesmo conjunto de dados que foram considerados no exemplo anterior. Porém, ao invés de utilizar somente as variáveis categóricas, usaremos todas elas. O primeiro passo é identificar para o programa as classes apropriadas para cada tipo de variável e, além disso, preparar os dados para a função `dist.ktab()`.

Pergunta

Quais são as espécies de besouros mais semelhantes? (Neste caso, sem predição, pois representa uma avaliação exploratória com as características funcionais das espécies)

Variáveis

- Dependentes: atributos funcionais (matriz de atributos contínuos e categóricos por espécie: `trait_dat`)

Análises

Aqui realizamos os passos para uma ordenação usando os atributos funcionais de diversas naturezas (Figura 14.3).

```
## PCoA dos atributos mistos

## 1. Verifique a classe de todos os traits e veja se estão de acordo com sua
expectativa
trait_dat %>%
    dplyr::summarise_all(class) %>%
    tidyr::gather(variable, class)
#>    variable     class
#> 1     trend character
#> 2   redlist character
#> 3     regio   integer
#> 4      biog character
#> 5     activ character
#> 6      diet character
#> 7    winter character
#> 8     color character
#> 9     breed character
#> 10     body   integer
#> 11     wing character
#> 12   period character

## 2. Neste exemplo, algumas variáveis que são ordinais (regio e body)
# foram reconhecidas como numéricas ou categóricas.
trait_dat$regio <- as.ordered(trait_dat$regio)
trait_dat$body <- as.ordered(trait_dat$body)

## 3. Combinar cada conjunto de atributos de acordo com sua natureza em um
# data.frame separado.
# 3.1. Categóricos.
trait_categ <- cbind.data.frame(
```

```
    trend = trait_dat$trend,
    redlist = trait_dat$redlist,
    biog = trait_dat$biog,
    activ = trait_dat$activ,
    diet = trait_dat$diet,
    winter = trait_dat$winter,
    color = trait_dat$color,
    breed = trait_dat$breed,
    wing = trait_dat$wing,
    period = trait_dat$period)

# 3.2 Ordinais.
trait_ord <- cbind.data.frame(regio = trait_dat$regio,
                              body = trait_dat$body)
rownames(trait_categ) <- rownames(trait_dat)
rownames(trait_ord) <- rownames(trait_dat)

# Agora, combinar os dois data.frames em uma lista chamada "ktab".
ktab_list <- ktab.list.df(list(trait_categ, trait_ord))

# Por fim, calcular a distância funcional entre as espécies.
# Em "type", a letra "N" indica variável categórica (ou nominal),
# enquanto a letra "O" indica variável ordinal.
dist_mist <- dist.ktab(ktab_list, type = c("N", "O"))

## Visualize os dados com uma PCoA
pcoa_traits_mist <- pcoa(dist_mist, correction = "cailliez")
eixos_mist <- as.data.frame(pcoa_traits_mist$vectors[,1:2])

plot_trait_mist <- ggplot(eixos_mist, aes(x = Axis.1, y = Axis.2)) +
    geom_point(pch = 21, size = 4, alpha = 0.7,
               color = "black", fill = "darkorange") +
    geom_text_repel(aes(Axis.1, Axis.2, label = rownames(eixos_mist)))+
    geom_hline(yintercept = 0, linetype = 2) +
    geom_vline(xintercept = 0, linetype = 2) +
    labs(x = "PCO 1", y = "PCO 2", title = "Dados mistos") +
    tema_livro()
plot_trait_mist
```

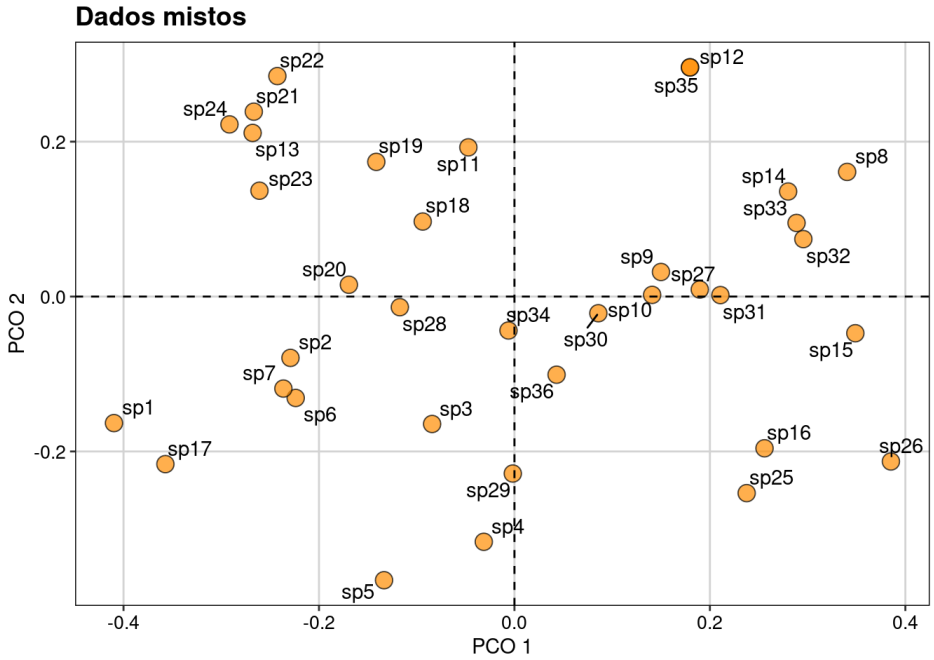


Figura 14.3: Ordenação usando os atributos funcionais de dados mistos.

Podemos combinar os dois gráficos (baseado em variáveis categóricas e em variáveis mistas) para comparar as duas medidas de distância, uma somente com dados categóricos (`gower()`) e uma com dados categóricos e ordinais (`dist.ktab()`) (Figura 14.4).

```
## Gráficos
grid.arrange(plot_trait_cat, plot_trait_mist, ncol = 2)
```

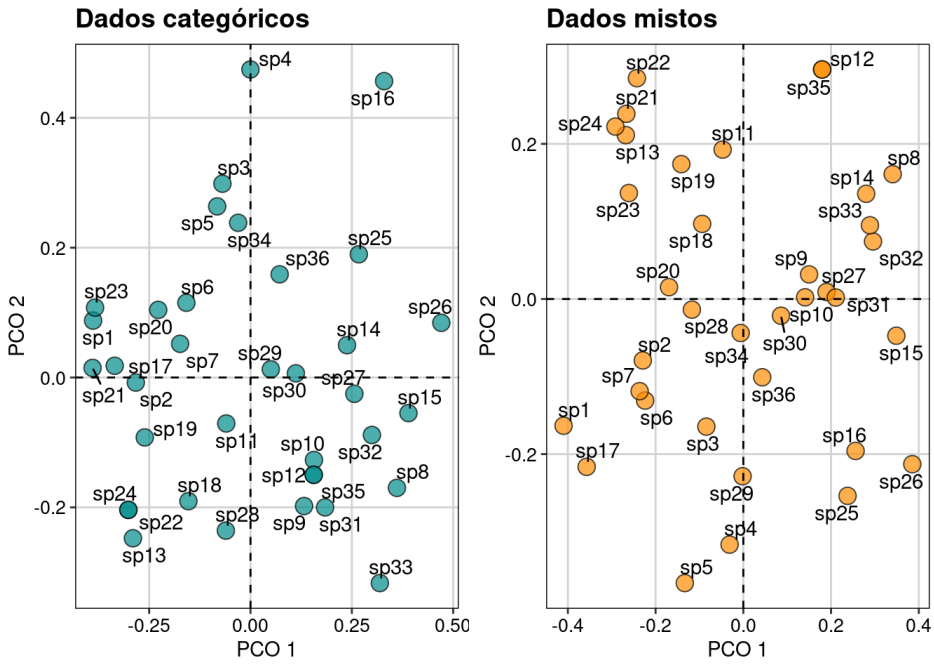


Figura 14.4: Ordenações usando os atributos funcionais categóricos e de dados mistos.

14.3 Métricas de diversidade funcional (alfa)

14.3.1 Riqueza funcional

A riqueza funcional mede a quantidade de espaço funcional preenchido pelas espécies de uma comunidade (Mason & Mouillot 2013). A estimativa desse espaço pode ser calculada usando dendrogramas (Petchey & Gaston 2002) ou através do método **Convex Hull** (Cornwell et al. 2006) que dão origem, respectivamente, às duas métricas mais usadas: i) Diversidade Funcional (FD) e ii) Riqueza Funcional (FRic). Os índices de riqueza funcional geralmente são usados como indicadores do espaço de nicho que é potencialmente usado ou não (Schleuter et al. 2010).

Exemplo prático

Explicação dos dados

Os dados utilizados neste exemplo são os mesmos do exemplo com dados mistos, i.e., categóricos e contínuos (objeto `dist_mist`).

Pergunta

Qual a relação entre riqueza de espécies e diversidade funcional? Todos os índices são correlacionados com a riqueza?

Variáveis

- Dependentes: atributos funcionais e composição de espécies para cálculo da diversidade funcional e riqueza em cada parcela

Análises

Vamos começar olhando os dados de dissimilaridade das espécies (`dist_mist`) e os dados de composição (`comun_dat`)

```
## Estrutura dos dados
# matriz de distância: distância entre as seis primeiras espécies
as.matrix(dist_mist)[1:6, 1:6]
#>           sp1       sp2       sp3       sp4       sp5       sp6
#> sp1 0.0000000 0.5000000 0.7107801 0.7771900 0.6107116 0.5041691
#> sp2 0.5000000 0.0000000 0.6808389 0.8538292 0.7345988 0.6487320
#> sp3 0.7107801 0.6808389 0.0000000 0.7179711 0.7381353 0.6527339
#> sp4 0.7771900 0.8538292 0.7179711 0.0000000 0.5106682 0.6522593
#> sp5 0.6107116 0.7345988 0.7381353 0.5106682 0.0000000 0.5177440
#> sp6 0.5041691 0.6487320 0.6527339 0.6522593 0.5177440 0.0000000

# composição de espécies: seis primeiras espécies nas seis primeiras
localidades
head(comun_dat)[1:6, 1:6]
#>    sp1 sp2 sp3 sp4 sp5 sp6
#> 3    0  19   2   0   0   0
#> 4    0   4   0   0   0   0
```

```
#> 6    1  58   2   0   0   0
#> 7    1   0   0   0   0   0
#> 9    3   0   0   0   0   0
#> 10   3  15   0   0   0   0
```

Agora vamos calcular a riqueza de espécies por comunidade

```
## Riqueza de espécies
richness <- dbFD(dist_mist, comun_dat)$nbsp
head(richness)
#>  3  4  6  7  9 10
#> 12  3  7  7  4  7
```

Vamos calcular a Riqueza (`FRic`) e Diversidade funcional usando o pacote `dbFD`.

```
# Functional Richness
fric <- dbFD(dist_mist, comun_dat)$FRic
head(fric)
#>           3           4           6           7           9          10
#> 0.226236923 0.009033539 0.158760885 0.158529234 0.014290140 0.200075112

## Functional Diversity
# Passo 1: análise de agrupamento para criar o dendrograma.
dend <- hclust(dist_mist, "average")

# Passo 2: transformar o dendrograma em um arquivo da classe phylo.
tree_dend <- as.phylo(dend)

# Passo 3: calcular o valor da diversidade funcional.
FD <- pd(comun_dat, tree_dend)$PD
head(FD)
#> [1] 3.590053 1.115574 2.255337 2.356478 1.472314 2.430329
```

> 👉 **Importante**
> O índice *Functional Richness* só funciona para comunidades com 3 ou mais espécies. Caso você tenha comunidades com 1 ou 2 espécies, o valor será `NA`

Os valores da Riqueza Funcional (`FRic`) variam entre 0 e +∞, sendo o valor máximo limitado pelo número de espécies em uma comunidade. Desse modo, os valores de FRic basicamente são uma representação da riqueza de espécies (e seus atributos funcionais) de uma comunidade (Villéger et al. 2008). Ou seja, quanto maior o número de espécies, maior será o espaço funcional ocupado por essas espécies. No pacote dbFD o valor de FRic é dividido pelo valor de FRic global (ou seja, pela riqueza funcional de todas as comunidades combinadas). Como resultado, ele limita a variação entre 0 e 1, onde valores próximos a 1 indicam que uma determinada comunidade tem riqueza funcional tão alta quanto a riqueza funcional de todas as comunidades juntas. Neste exemplo, as localidades 21 e 89 possuem, respectivamente, a maior e menor Riqueza Funcional.

14.3.2 Divergência funcional

A divergência funcional é uma medida que descreve a irregularidade na distribuição dos valores dos atributos no volume do espaço funcional ocupado por todas as espécies de uma certa comunidade (Garnier et al. 2015). Para obter os valores de divergência, o espaço funcional é calculado através do método *Convex Hull* (*Functional Divergence*) ou do espaço multidimensional calculado com um PCoA (*Functional Dispersion*). Nos dois casos, o valor da métrica representa a distância média das espécies para o centro de gravidade ou centroide do espaço funcional, ponderado pela abundância relativa das espécies (Villéger et al. 2008, Laliberté & Legendre 2010). Desse modo, a divergência funcional é uma medida que calcula o grau de diferenciação em que a distribuição da abundância maximiza a divergência entre os atributos funcionais (Mason & Mouillot 2013). Em geral, estudos que usam esses índices buscam entender o grau de diferenciação de recursos de espécies que coexistem em uma comunidade (Garnier et al. 2015).

Vamos calcular a divergência funcional (`FDiv`) e dispersão funcional (`FDis`) usando o pacote `dbFD`. Para isso, usaremos a matriz de distância obtida dos dados `trait_dat` (variáveis categóricas e ordinais) e nomeada como `dist_mist`

```
## Functional Divergence
fdiv <- dbFD(dist_mist, comun_dat)$FDiv
head(fdiv)
#>         3         4         6         7         9        10
#> 0.9692023 0.8838557 0.4082808 0.9147644 0.9010790 0.6982640

# Functional Dispersion
fdis <- dbFD(dist_mist, comun_dat)$FDis
head(fdis)
#>         3         4         6         7         9        10
#> 0.2977975 0.3203602 0.2218237 0.3261248 0.3683898 0.3910530
```

> 👉 **Importante**
> - O índice *Functional Divergence* só funciona para comunidades com 3 ou mais espécies. Caso você tenha comunidades com 1 ou 2 espécies, o valor será `NA`
> - O índice *Functional Dispersion* só funciona para comunidades com 3 ou mais espécies. Caso você tenha comunidades com 1 ou 2 espécies, o valor será zero

Os valores da Divergência Funcional (`FDiv`) variam entre 0 e 1. Valores que se aproximam de zero indicam que a espécie mais abundante está muito próxima do valor do atributo médio da comunidade, ao passo que valores próximos a 1 indicam que a espécie mais abundante está muito distante (ou seja, é muito diferente) do valor médio da comunidade (Villéger et al. 2008). Neste exemplo, as localidades 159 e 6 possuem, respectivamente, a maior e menor Divergência Funcional.

14.3.3 Regularidade funcional

A regularidade funcional (do inglês *Functional Evenness* - `FEve`) mede o quão regular é a distribuição da abundância dos valores dos atributos funcionais no espaço funcional. Diferente dos outros métodos, a versão multidimensional deste índice utiliza um método chamado *Minimum Spanning Tree* (MST) para conectar todas espécies no espaço funcional. A distância par apar das espécies na MST é ponderada pela abundância relativa das espécies e, desse modo, o valor final da regularidade funcional (FEve) vai variar de 0 (máxima irregularidade da distribuição da abundância ou distância funcional das espécies) a 1 (máxima regularidade).

Vamos calcular a Regularidade Funcional (`FEve`) usando o pacote `dbFD`. Para isso, usaremos a matriz de distância obtida dos dados `trait_dat` (variáveis categóricas e ordinais) e nomeada como `dist_mist`

```
## Functional evenness
feve <- dbFD(dist_mist, comun_dat)$FEve
head(feve)
#>         3         4         6         7         9        10
#> 0.4054808 0.5587917 0.5406140 0.6974712 0.9575697 0.6297941
```

> **Importante**
> O índice *Functional evenness* só funciona para comunidades com 3 ou mais espécies. Caso você tenha comunidades com 1 ou 2 espécies, o valor será `NA`

Os valores da Regularidade Funcional (FEve) variam entre 0 e 1 (máxima regularidade ou regularidade perfeita). A diminuição do valor de FEve em direção a zero indica que uma redução da regularidade da distribuição da abundância ou distância funcional entre as espécies (Villéger et al. 2008). Neste exemplo, as localidades 197 e 175 possuem, respectivamente, a maior e menor Regularidade Funcional.

14.3.4 Correlação entre as métricas de diversidade funcional (alfa)

Aqui vamos fazer a correlação entre as métricas de diversidade funcional (alfa) (Figura 14.5).

```
## Gráficos
## Você pode criar uma tabela com os resultados de todas as métricas
metricas <- data.frame(richness = richness,
                       FD_gp = FD,
                       fric = fric,
                       fdiv = fdiv,
                       fdis = fdis,
                       feve = feve)
head(metricas)
#>    richness    FD_gp        fric      fdiv      fdis      feve
#> 3        12 3.590053 0.226236923 0.9692023 0.2977975 0.4054808
#> 4         3 1.115574 0.009033539 0.8838557 0.3203602 0.5587917
#> 6         7 2.255337 0.158760885 0.4082808 0.2218237 0.5406140
```

```
#> 7          7 2.356478 0.158529234 0.9147644 0.3261248 0.6974712
#> 9          4 1.472314 0.014290140 0.9010790 0.3683898 0.9575697
#> 10         7 2.430329 0.200075112 0.6982640 0.3910530 0.6297941

## Gráfico para comparar o comportamento das métricas
ggpairs(metricas) + tema_livro()
```

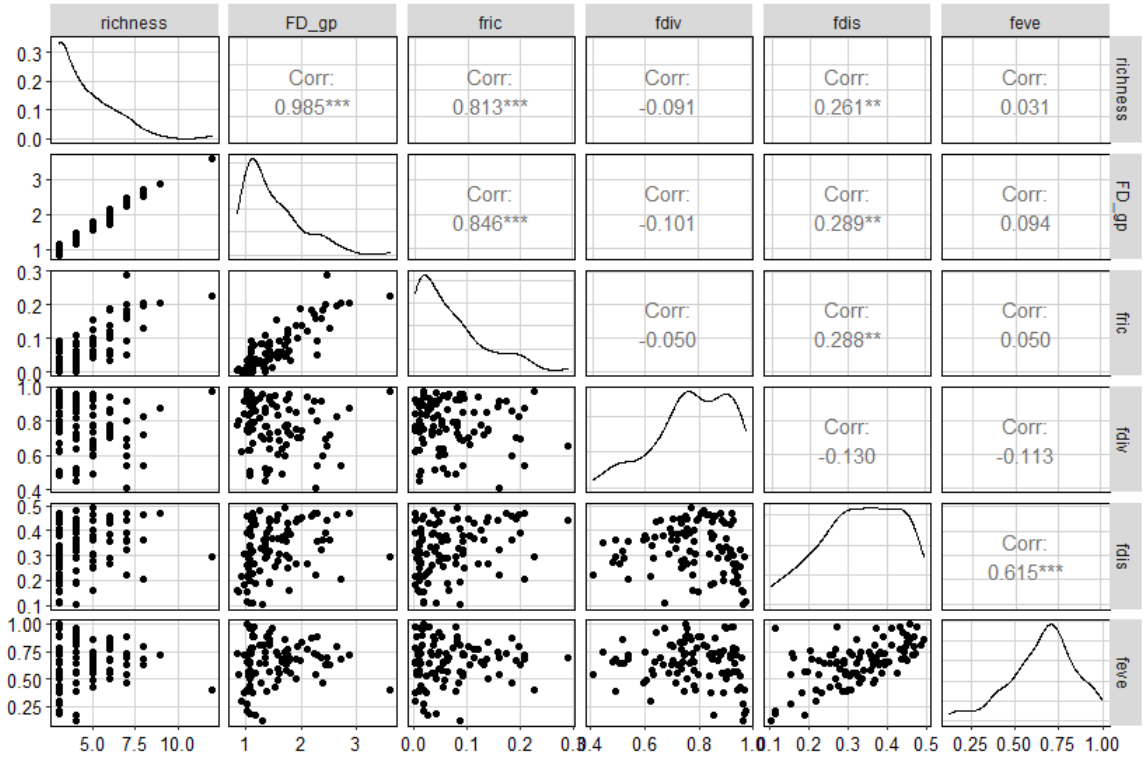


Figura 14.5: Correlação entre as métricas de diversidade funcional (alfa).

Os resultados indicam que a Diversidade Funcional de Petchey & Gaston (2002) (`r = 0.985`) e a riqueza funcional (`r = 0.813`) são altamente correlacionadas com a riqueza de espécies. Porém, a divergência funcional, regularidade funcional e dispersão funcional não estão correlacionadas com a riqueza de espécies.

A figura obtida com o código `ggpairs(metricas)` representa uma matriz de correlação comparando cada par de variáveis (neste caso, os índices de diversidade). No lado esquerdo da figura (abaixo da diagonal) são representados *scatter plots* (veja Capítulo 6), a no lado direito (acima da diagonal) pode-se encontrar os valores das correlações (`r`) entre os pares comparados. No caso das correlações, quanto mais próximo de +1 ou -1, mais forte é a relação entre essas variáveis do par comparado. O gráfico de linhas na diagonal demonstra a densidade de cada variáveis individualmente (veja Capítulo 7).

14.4 Métricas de diversidade funcional (beta)

Assim como na diversidade beta taxonômica (Capítulo 12) e na diversidade beta filogenética (Capítulo 13), a diversidade beta funcional é uma medida que compara a composição (e a variação na composição) de atributos funcionais das espécies entre duas ou mais localidades. Porém, assim

como na medida tradicional taxonômica (como Jaccard ou Sørensen), diferenças na diversidade beta podem ser geradas pela **mudança na identidade das espécies (ou do atributo)** ou na **riqueza de espécies (ou de atributos)** entre duas localidades (Figura 14.6). Desse modo, é possível particionar a diversidade beta funcional em aninhamento (do inglês *nestedness*) e substituição (do inglês *turnover*) (veja Capítulo 12). Além disso, os cálculos da diversidade beta funcional podem ser realizados par a par (`functional.beta.pair()`) ou para a comparações de múltiplas localidades (`funcional.beta.multi()`).

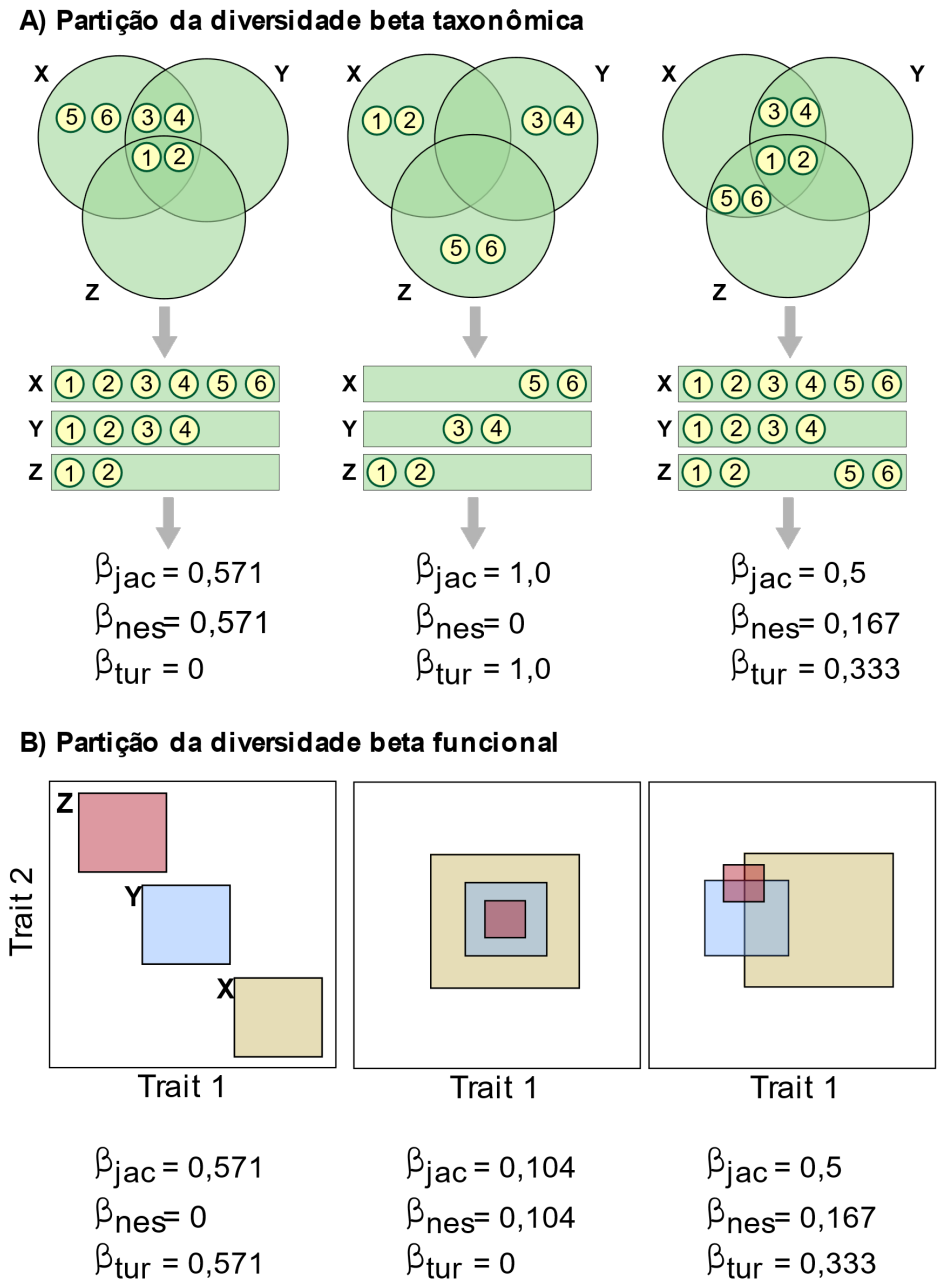


Figura 14.6: Partição da diversidade beta taxonômica (A) e funcional (B). Os três cenários apresentados tanto para a diversidade beta taxonômica como funcional representam, respectivamente, diversidade beta explicada somente por substituição, aninhamento e uma combinação dos dois.

Exemplo 4

Os dados no exemplo a seguir utilizam somente a informação de presença (1) ou ausência (0) das espécies nas localidades. Neste exemplo hipotético criado por Baselga et al. (2021), foram amostradas 11 espécies (sp1-sp11) em quatro localidades (A-D). Para cada espécie, criamos dois atributos contínuos hipotéticos (trait1 e trait2).

Pergunta

Qual a contribuição relativa do aninhamento e substituição para a diversidade beta?

Variáveis

- Dependentes: atributos funcionais e composição de espécies.

Análises

Vamos fazer a análise da partição da diversidade beta funcional e comparar seus componentes (Figura 14.7).

```
## Partição da Diversidade beta (Método Baselga)
fun_beta_multi <- functional.beta.multi(x = comm_baselga,
                                        trait = trait_baselga,
                                        index = "jaccard")
fun_beta_multi
#> $funct.beta.JTU
#> [1] 0.7101449
#>
#> $funct.beta.JNE
#> [1] 0.1509662
#>
#> $funct.beta.JAC
#> [1] 0.8611111

## Partição da Diversidade beta (Método Baselga)
fun_beta <- functional.beta.pair(x = comm_baselga,
                                 trait = trait_baselga,
                                 index = "jaccard")

# Os códigos abaixo permitem extrair a matriz de distância (par a par) com a
partição em substituição e nestedness
fun_turnover <- fun_beta$funct.beta.jne
fun_nestedness <- fun_beta$funct.beta.jtu
fun_jaccard <- fun_beta$funct.beta.jac

## Gráfico de comparação do substituição e aninhamento
dat_betapart <- data.frame(turnover = as.numeric(fun_turnover),
                           nested = as.numeric(fun_nestedness))

## Gráfico
plot_betapart <- ggplot(dat_betapart, aes(x = turnover, y = nested)) +
```

```
    geom_point(pch = 21, size = 4, alpha = 0.7, color = "black",
               fill = "#525252") +
    labs(x = "Beta Diveristy (Substituição)",
         y = "Beta Diveristy (Aninhamento)") +
    tema_livro()
plot_betapart
```

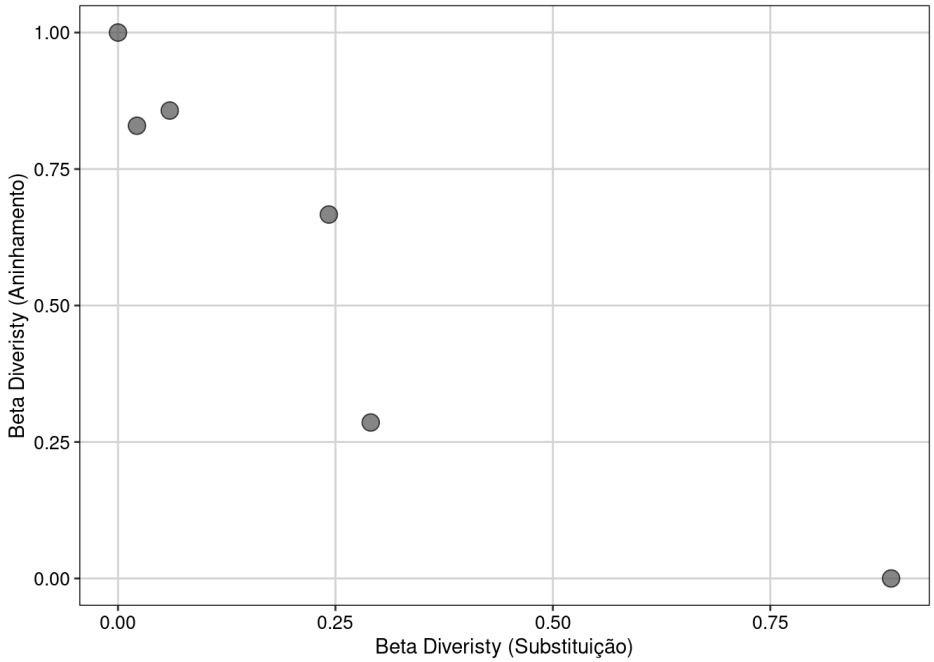


Figura 14.7: Relação entre os componentes partição da diversidade funcional (beta).

Os resultados da análise de partição (`fun_beta_multi()`) indicam que 82,5% (`0,710`/`0,861`) da variação na diversidade beta é explicada pelo componente substituição, enquanto 17,5% (`0,151`/`0,861`) pelo componente aninhamento. As matrizes de distância obtidas na análise par a par podem ser utilizadas para testar, *a posteriori*, a relação entre gradientes ambientais e diversidade beta funcional (mais detalhes abaixo).

14.5 Composição Funcional (*Community Wegihed Means* - CWM)

As medidas de diversidade beta funcional apresentadas acima fornecem matrizes de distância com comparações par a par de localidades em termos da composição de atributos funcionais. Porém, muitas vezes o pesquisador quer medir o "atributo médio" da comunidade para investigar, por exemplo, se um determinado gradiente ambiental afeta a expressão (em termos de abundância ou densidade) de dado atributo funcional. Em geral, a medida utilizada é o CWM (do inglês *Community Wegihed Means*). O CWM é basicamente uma média ponderada de um determinado atributo (coluna *m* na matriz T) em relação a abundância de todas as espécies que ocorrem na localidade (linha *n* na matriz X). O cálculo no R é feito pela função `functcomp()` e usa somente as duas matrizes (T e X). Os leitores que pretendem usar essas métricas devem ler críticas em Peres-Neto et al. (2017).

```
## Matriz T
head(trait_baselga)
#>     Trait.1 Trait.2
#> sp1       1       1
#> sp2       1       2
#> sp3       1       4
#> sp4       2       1
#> sp5       2       2
#> sp6       3       3

## Matriz X
head(comm_baselga)
#>   sp1 sp2 sp3 sp4 sp5 sp6 sp7 sp8 sp9 sp10 sp11
#> A   1   1   0   1   1   0   0   0   0    0    0
#> B   1   0   1   0   0   0   0   1   1    0    0
#> C   0   0   0   0   0   1   1   0   0    1    1
#> D   0   1   0   1   0   0   1   0   1    0    0

## Função functcomp calcula o cwm para combinar as matrizes T e X
cwm_ex <- functcomp(trait_baselga, as.matrix(comm_baselga))
cwm_ex
#>   Trait.1 Trait.2
#> A     1.5     1.5
#> B     2.5     2.5
#> C     4.0     4.0
#> D     2.5     3.0
```

A matriz resultante `cwm_es` é formada pelas localidades (linhas) e os atributos "médios" (colunas) nestas localidades. Essa matriz pode ser utilizada em diversas análises como dbRDA, RDA ou RDA parcial (veja Capítulo 9). Na sequência, vamos utilizar testes de hipóteses para entender como podemos calcular as diversidades funcional alfa e beta com outros testes estatísticos apresentados neste livro.

Exemplo 5

Neste exemplo, usaremos novamente os dados de 34 espécies de plantas (Frenette-Dussault et al. 2012), mas agora vamos testar o efeito de um gradiente de aridez sobre a diversidade alfa funcional.

Pergunta

O gradiente de aridez influencia a divergência e regularidade funcional de plantas?

Predições

- *Predição 1*: locais mais áridos possuem menor divergência funcional de plantas (métrica escolhida: FDis)
- *Predição 2*: locais mais úmidos possuem menor regularidade funcional de plantas (métrica escolhida: FEve)

Variáveis

- Preditora: gradiente de aridez (matriz de variáveis ambientais por localidade: `ambie_fren_dat`)
- Dependentes: composição de espécies (matriz de espécies por localidade: `comun_fren_dat`) e atributos funcionais (matriz de atributos contínuos por espécie: `trait_fren_dat`)

Análises

Vamos calcular primeiramente as matrizes de Divergência funcional (FDis) e Regularidade Funcional (FEve), depois ajustar modelos lineares, fazer o diagnóstico dos mesmo e por fim plotar os resultados (Figura 14.8).

```
## Passo 1: calcular a distância funcional
trait_pad <- decostand(trait_fren_dat, "standardize")
euclid_dis <- vegdist(trait_pad, "euclidean")

## Passo 2: calcular a Divergência funcional (FDis) e Regularidade Funcional
(FEve)
fdis <- dbFD(euclid_dis, comun_fren_dat)$FDis # Fdis=0 em locais com somente
uma espécie
feve <- dbFD(euclid_dis, comun_fren_dat)$FEve

## Passo 3: Utilizar um modelo linear para comparar o efeito da aridez sobre
FDis (predição 1) e FEve (predição 2)
# Combinar dados em um data.frame.
lm_dat <- data.frame(aridez = ambie_fren_dat$Aridity,
                     fdis = fdis, feve = feve)

# Modelo 1
mod1 <- lm(fdis ~ aridez, data = lm_dat)

# Conclusão: a aridez não tem efeito sobre a divergência funcional
anova(mod1)
#> Analysis of Variance Table
#>
#> Response: fdis
#>           Df Sum Sq Mean Sq F value Pr(>F)
#> aridez     1 0.2083 0.20834  0.9945 0.3241
#> Residuals 44 9.2179 0.20950

# Modelo 2
mod2 <- lm(feve ~ aridez, data = lm_dat)

# Conclusão: a aridez não tem efeito sobre a regularidade funcional
anova(mod2)
#> Analysis of Variance Table
#>
#> Response: feve
```

```
#>           Df  Sum Sq  Mean Sq F value Pr(>F)
#> aridez     1 0.02098 0.020979  1.0447 0.3123
#> Residuals 44 0.88353 0.020080

## Passo 4: gráfico para visualizar os dois resultados
# Gráfico modelo 1.
plot_pred1 <- ggplot(lm_dat, aes(x = aridez, y = fdis)) +
    geom_point(pch = 21, size = 4, alpha = 0.7, color = "black",
               fill="darkorange") +
    labs(x = "Aridez", y = "Divergência Funcional (FDis)") +
    tema_livro()

# Gráfico modelo 2.
plot_pred2 <- ggplot(lm_dat, aes(x = aridez, y = feve)) +
    geom_point(pch=21, size=4, alpha = 0.7, color = "black",
               fill="cyan4") +
    labs(x = "Aridez", y = "Regularidade Funcional (FEve)") +
    tema_livro()

## Visualização dos dois gráficos em um única janela
grid.arrange(plot_pred1, plot_pred2, ncol = 2)
```

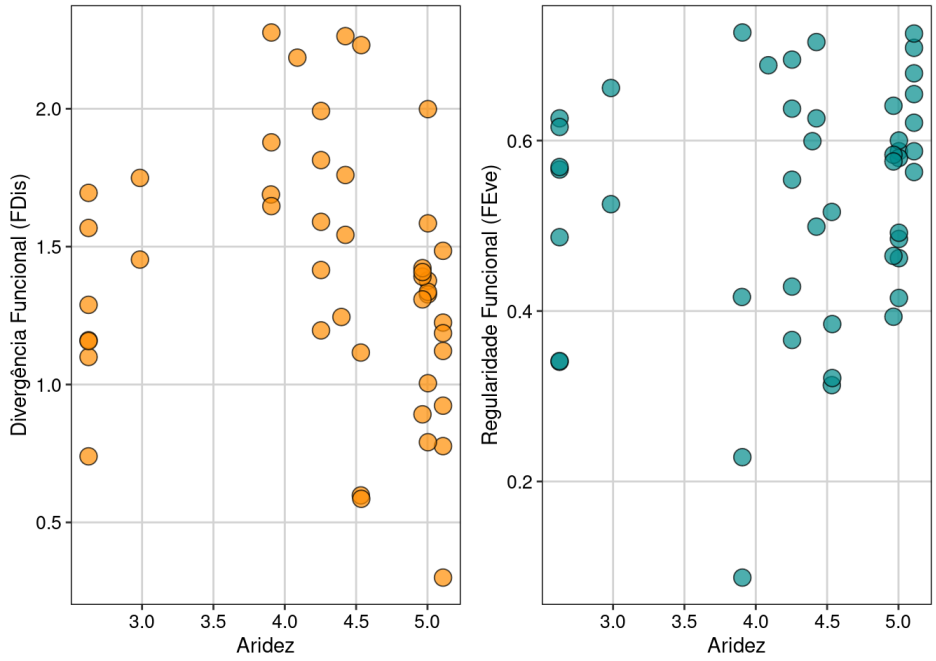


Figura 14.8: Relação entre a composição funcional e o gradiente de aridez, ajustado por modelos lineares com seus diagnósticos.

Os resultados dos modelos `anova(mod1)` e `anova(mod2)` indicam que o gradiente de aridez não afeta a dispersão e regularidade funcional. Os detalhes para conferir os pressupostos das análises foram descritos no Capítulo 7.

Exemplo 6

Agora, vamos utilizar novamente os dados de 34 espécies de plantas (Frenette-Dussault et al. 2012), mas agora para testar o efeito do pastejo sobre a diversidade beta funcional.

Pergunta

O pastejo determina a ocorrência de espécies de plantas com diferentes atributos funcionais?

Predição

- A composição funcional de plantas é diferente entre áreas com e sem pastejo?

Variáveis

- Preditora: áreas com e sem pastejo de gado (variável categórica com dois níveis: *grazed* e *ungrazed*: `ambie_fren_dat`)
- Dependentes: composição de espécies (matriz de espécies por localidade: `comun_fren_dat`) e atributos funcionais (matriz de atributos contínuos por espécie: `trait_fren_dat`)

Análises

Nesse exemplo vamos calcular a CWM e depois usar uma PERMANOVA (Capítulo 9) para comparar o efeito das áreas com e sem pastejo sobre a diversidade funcional (Figura 14.9).

```
## Passo 1: CWM
cwm_fren <- functcomp(trait_pad, as.matrix(comun_fren_dat))
head(cwm_fren)
#>           LA        SLA        LDMC       LN15        LCC
#> 1 -0.2411700 -0.3485515  0.19745200  0.1874003 -0.5367368
#> 2 -0.3977371  0.2326622 -0.09093270 -0.2859777  0.1643190
#> 3 -0.1857134  0.2010756 -0.39877265 -0.1250643 -0.4304617
#> 4 -0.2284064  0.1604101  0.80496307 -0.3704253  0.7193853
#> 5 -0.1664790  0.3486956  0.02232213 -0.2041931  0.2051391
#> 6 -0.3258821  0.3664583  0.04996829 -0.3352572  0.4713089

## Passo 2: calcular a distância funcional
cwm_dis <- vegdist(cwm_fren, "euclidean")

## Passo 3: testar se a composição funcional varia entre as áreas com uma
PERMANOVA
perman_fren <- adonis(cwm_fren ~ Grazing, data = ambie_fren_dat)
perman_fren
#>
#> Call:
#> adonis(formula = cwm_fren ~ Grazing, data = ambie_fren_dat)
#>
#> Permutation: free
#> Number of permutations: 999
#>
#> Terms added sequentially (first to last)
#>
```

```
#>           Df SumsOfSqs MeanSqs  F.Model      R2 Pr(>F)
#> Grazing    1    -174.5 -174.46 -0.61591 -0.0142  0.547
#> Residuals 44   12463.0  283.25           1.0142
#> Total     45   12288.5                   1.0000

## Passo 4: comparar a variação dentro de cada grupo com Betadisper
betad_fren <- betadisper(cwm_dis, ambie_fren_dat$Grazing)
permutest(betad_fren)
#>
#> Permutation test for homogeneity of multivariate dispersions
#> Permutation: free
#> Number of permutations: 999
#>
#> Response: Distances
#>           Df  Sum Sq  Mean Sq      F N.Perm Pr(>F)
#> Groups     1  0.0539 0.053858 0.1946    999  0.669
#> Residuals 44 12.1763 0.276735

## Passo 5: PCoA
cwm_pcoa <- pcoa(D = cwm_dis, correction = "cailliez")
pcoa_eixos <- cwm_pcoa$vectors[, 1:2]
pcoa_dat <- data.frame(pastagem = ambie_fren_dat$Grazing, pcoa_eixos)

## Passo 6: definir os grupos ("HULL") para serem categorizados no gráfico
grp.Grazed <- pcoa_dat[pcoa_dat$pastagem == "Grazed", ]
                      [chull(pcoa_dat[pcoa_dat$pastagem == "Grazed",
                       c("Axis.1", "Axis.2")]), ]
grp.Ungrazed <- pcoa_dat[pcoa_dat$pastagem == "Ungrazed", ]
                        [chull(pcoa_dat[pcoa_dat$pastagem == "Ungrazed",
                       c("Axis.1", "Axis.2")]), ]
hull_cwm <- rbind(grp.Grazed, grp.Ungrazed)

## Passo 7: Gráfico biplot
# % de explicação do eixo 1
100 * (cwm_pcoa$values[, 1]/cwm_pcoa$trace)[1] #> [1] 53.5471

# % de explicação do eixo 2
100 * (cwm_pcoa$values[, 1]/cwm_pcoa$trace)[2] #> [1] 24.56026

ggplot(pcoa_dat, aes(x = Axis.1, y = Axis.2, color = pastagem,
                     shape = pastagem)) +
  geom_point(size = 4, alpha = 0.7) +
  geom_polygon(data = hull_cwm, aes(fill = pastagem, group = pastagem),
                                    alpha = 0.3) +
  scale_color_manual(values = c("darkorange", "cyan4")) +
  scale_fill_manual(values = c("darkorange", "cyan4")) +
```

```
  labs(x = "PCO 1 (53.6%)", y = "PCO 2 (24.6%)") +
  tema_livro()
```

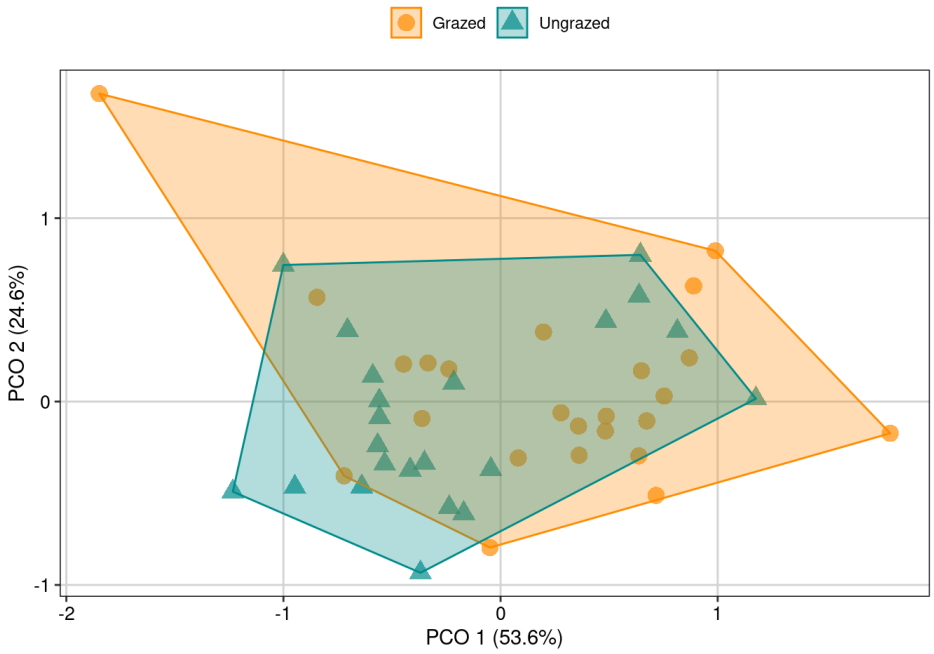


Figura 14.9: Relação do pastejo sobre a composição funcional.

Neste exemplo, os resultados `perman_fren` demonstram que a composição funcional de plantas não é afetada pelo pastejo (P > 0,05) e que a dispersão da composição (uma medida potencial de diversidade beta: Anderson et al. 2006) de espécies também não muda entre áreas com ou sem pastejo (`permutest(betad_fren)`). A função `betadisper()` deve ser sempre utilizada em conjunto com a PERMANOVA (`adonis()`) para poder interpretar quais as fontes de variação na composição de espécies. Sendo assim, esta análise representa um método fundamental para comparar se o potencial efeito (quando houver) é fruto de diferença na composição de espécies (i.e., diferença na posição dos centroides entre dois ou mais grupos) ou na variação da composição de espécies entre os grupos (i.e., diferença na dispersão dos dados em relação aos centroides, ver mais no Capítulo 9). Esta última informação, a dispersão, é geralmente interpretada como uma analogia a homogeneidade de variâncias de uma ANOVA (i.e., Teste de Levene). A hipótese nula do `betadisper()` é que a dispersão dos grupos é homogênea (ou seja, o valor de probabilidade nos casos que existem dispersões homogêneas será maior do que 0,05). Porém, se esse valor de p for menor do que 0,05, você deve rejeitar a hipótese nula e concluir que as dispersões são heterogêneas.

Os gráficos de PCoA são uma ferramenta poderosa para interpretar os resultados da PERMANOVA + Betadisper. Quanto mais diferente a composição de espécies entre dois ou mais grupos, mais distante devem ser os centroides desses grupos. Além disso, se as áreas dos polígonos que conectam todas as réplicas de cada grupo forem diferentes em tamanho (hipótese que será testada com o Betadisper), é possível também visualizar esta diferença. Em conclusão, para testar se diferenças de composição funcional existem entre dois ou mais grupos, será fundamental: i) comparar a variação da posição dos centroides (função `adonis()`) e ii) a variação da dispersão da composição funcional entre os grupos (função `betadisper()`).

14.6 Variação Intraespecífica

Os métodos discutidos anteriormente utilizam valores médios dos atributos das espécies para descrever a estrutura funcional da comunidade e interpretar as relações entre determinadas variáveis preditoras (como clima, por exemplo) com a diversidade funcional. Porém, ao utilizar atributos médios estamos desconsiderando que a variação deste atributo dentro da espécie seja determinante para a resposta da espécie ao ambiente ou seu efeito sobre o ecossistema (Bolnick et al. 2011, Violle et al. 2012). Os estudos que usam dados médios para testar hipóteses em ecologia funcional argumentam que a variação dentro da espécie é menor do que a variação entre espécies e, desse modo, o ruído causado ao desconsiderar a variância do atributo dentro das espécies é desprezível (Siefert et al. 2015). Porém, diversos estudos têm mostrado que esse argumento é frágil e que a inclusão da variação intraespecífica melhora nossa capacidade preditiva em ecologia funcional (Violle et al. 2012, Siefert et al. 2015). Uma abordagem geralmente utilizada é a decomposição da variância do atributo em diferentes níveis de organização: i) variação dentro da população da mesma espécie em uma mesma unidade amostral, ii) variação dentro das populações (independente da espécie) de uma comunidade em uma mesma unidade amostral, e iii) variação entre populações. Conhecida como estatística T, esta abordagem permite entender as fontes (intra ou interespecífica) de variação nos atributos em diferentes escalas (Violle et al. 2012). Outro método que quantifica a variância explicada pela variabilidade intraespecífica, interespecífica e a covariância entre elas foi proposto por Leps et al. (2011). Este método permite calcular a contribuição da variação intraespecífica dentro e entre comunidades. Agora, vamos entender a contribuição da variação de um atributo dentro da espécie comparada à variação entre espécies.

Exemplo 7

Vamos utilizar os dados de 11 espécies de anuros associados com 26 poças no Parque Nacional Lagoa do Peixe (Dalmolin et al. 2020). Atributos morfológicos foram coletados em todos os indivíduos coletados em cada poça. Desse modo, é possível comparar a variação morfológica entre indivíduos da mesma espécie e entre espécies diferentes. Além disso, é possível quantificar a contribuição da variação dentro e entre diferentes poças. No exemplo abaixo, criamos cinco atributos com nomes diferentes daqueles usados no artigo de Dalmolin et al. (2020). Em cada poça, os autores coletaram os seguintes dados das poças: i) profundidade, ii) area, iii) distância entre poças, e iv) distância da poça para a floresta mais próxima.

Pergunta 1

Qual a contribuição da variação intraespecífica para a variação total dos atributos morfológicos de anuros?

Predição

- A alta plasticidade fenotípica de anuros indica alta contribuição da variação intraespecífica comparada a interespecífica

Variáveis

- Preditora: espécies (categórica)
- Dependente: variação dos atributos morfológicos

Análises

Aqui vamos realizar o passo a passo das análises para comparar a contribuição da variação intraespecífica para a variação total dos atributos morfológicos.

```
## Dados necessários
# Matriz de traits.
head(traits)
#>   pond Species body_size biomass eye_size leg_size flatness
#> 1  DN1     Sp2     2.405   2.291    3.104    0.450    0.794
#> 2  DN1     Sp3     1.882   2.039    2.926    0.345    1.063
#> 3  DN1     Sp4     0.699   0.342    0.782    0.104    3.055
#> 4  DN1     Sp4     0.725   0.598    1.120    0.136    2.759
#> 5  DN1     Sp4     0.448   0.385    0.844    0.107    3.557
#> 6  DN1     Sp4     0.640   0.470    0.861    0.093    3.420

## Partição da variação intra e interespecífica
## Passo 1: Tamanho corporal
mod_body_size <- aov(body_size~Species, data = traits)
summary(mod_body_size)
#>              Df Sum Sq Mean Sq F value Pr(>F)
#> Species      10  95.91   9.591    25.5 <2e-16 ***
#> Residuals   195  73.35   0.376
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Contribuição da variação intra-específica para o tamanho corporal.
itv_BS <- 100 * (73.35/(95.92 + 73.35))
itv_BS
#> [1] 43.33314

## Passo 2: Biomassa
mod_biomass <- aov(biomass~Species, data = traits)
summary(mod_biomass)
#>              Df Sum Sq Mean Sq F value Pr(>F)
#> Species      10 118.17  11.817   23.95 <2e-16 ***
#> Residuals   195  96.22   0.493
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Contribuição da variação intra-específica para a biomassa.
itv_biomass <- 100 * (96.22/(118.17 + 96.22))
itv_biomass
#> [1] 44.88082

## Passo 3: Tamanho do olho
mod_eye_size <- aov(eye_size ~ Species, data = traits)
summary(mod_eye_size)
```

```
#>              Df Sum Sq Mean Sq F value Pr(>F)
#> Species      10  203.1  20.309   50.51 <2e-16 ***
#> Residuals   195   78.4   0.402
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Contribuição da variação intra-específica para o tamanho do olho.
itv_eye_size <- 100 * (78.39/(203.09 + 78.39))
itv_eye_size
#> [1] 27.84923

## Passo 4: Achatamento dorso-ventral
mod_flatness <- aov(flatness~Species, data = traits)
summary(mod_flatness)
#>              Df Sum Sq Mean Sq F value Pr(>F)
#> Species      10 104.47  10.447   92.07 <2e-16 ***
#> Residuals   195  22.13   0.113
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# Contribuição da variação intra-específica para o achatamento dorso-ventral.
itv_flatness <- 100 * (22.13/(104.48 + 22.13))
itv_flatness
#> [1] 17.47887

## Passo 5: Combinar os valores de cada trait em um vetor
valores <- c(itv_BS, itv_biomass, itv_eye_size, itv_flatness)

# Passo 6: Combinar valores e traits em um data.frame.
itv_results <- data.frame(trait = c("body_size", "biomass", "eye_size",
                                    "flatness"), itv_explic = valores)

## Tabela com resultados da explicação atribuida para a variação
intraespecífica
itv_results %>%
    mutate("explained intraspecific variance" = round(itv_explic, 2)) %>%
    dplyr::select(trait, "explained intraspecific variance")
#>       trait explained intraspecific variance
#> 1 body_size                            43.33
#> 2   biomass                            44.88
#> 3  eye_size                            27.85
#> 4  flatness                            17.48
```

Pergunta 2

Qual a contribuição da variação entre poças para a variação total dos atributos morfológicos de anuros?

Predição

- A variação morfológica de anuros é afetada por mudanças dentro das espécies e entre diferentes espécies de poças distintas

Variáveis

- Preditoras: poças e espécies (ambas categóricas)
- Dependente: variação dos atributos morfológicos

Análises

Aqui vamos realizar o teste para comparar a contribuição da variação entre poças para a variação total dos atributos morfológicos (Figura 14.10).

```
## Dados necessários
# Matriz de traits sem nomes de espécies ou localidades
trait_m <- traits[, c("body_size", "biomass", "eye_size",
                      "leg_size", "flatness")]
head(trait_m)
#>   body_size biomass eye_size leg_size flatness
#> 1     2.405   2.291    3.104    0.450    0.794
#> 2     1.882   2.039    2.926    0.345    1.063
#> 3     0.699   0.342    0.782    0.104    3.055
#> 4     0.725   0.598    1.120    0.136    2.759
#> 5     0.448   0.385    0.844    0.107    3.557
#> 6     0.640   0.470    0.861    0.093    3.420

trait_decomp <- decompCTRE(traits = trait_m, sp = traits$Species,
                           ind.plot = traits$pond, print = FALSE)
barplot.decompCTRE(trait_decomp)
```

Figura 14.10: Gráfico de barras mostrando a contribuição da variação entre poças para a variação total dos atributos morfológicos.

Pergunta 3

Características ambientais das poças afetam a variação intraespecífica?

Predição

- A profundidade e área da poça aumentam a contribuição da variação intraespecífica em relação a variação interespecífica.

Variáveis

- Preditora: características das poças
- Dependentes: variação dos atributos morfológicos e contribuição da variação intraespecífica

Análises

Para calcular a contribuição relativa da variação intraespecífica em relação à variação interespecífica dentro de uma comunidade, por exemplo, Siefert et al. (2015) sugeriram uma métrica chamada de wITV (*within-community Intraspecific Trait Variation*). A wITV representa a razão da variação intraespecífica em relação a variação total dentro de uma comunidade (e.g., parcela, poça) que inclui: i) a abundância relativa de cada espécie *j* ocorrendo na comunidade *i*, ii) o valor médio do atributo da espécie *j* na comunidade *i*, e iii) o valor do atributo *k* de cada indivíduo da espécie *j* que ocorre na comunidade *i*. Como esta medida é feita por unidade amostral (ou seja, sua comunidade de interesse), é possível testar hipóteses ecológicas que tentem explicar processos que aumentem ou diminuam a variação de um determinado atributo dentro ou entre espécies diferentes. A função `wITV` foi adaptada para a linguagem R por de Bello et al. (2021). Para facilitar o cálculo do wITV para cada comunidade, de Bello et al. (2021) executaram os códigos com a função `for()` que repete iterativamente a análise para gerar os valores de todas as comunidades em uma forma dinâmica. Após executar as análises com o `for()`, a função salva os resultados dentro do objeto `wITVResults`. Após obter esses resultados, é possível utilizar modelos lineares para testar quais variáveis preditoras (em nosso exemplo, características das poças) afetam o aumento ou diminuição da contribuição relativa da variação intraespecífica (Figura 14.10).

```
## Dados necessários
# Matriz de traits.
head(traits)
#>   pond Species body_size biomass eye_size leg_size flatness
#> 1  DN1     Sp2     2.405   2.291    3.104    0.450    0.794
#> 2  DN1     Sp3     1.882   2.039    2.926    0.345    1.063
#> 3  DN1     Sp4     0.699   0.342    0.782    0.104    3.055
#> 4  DN1     Sp4     0.725   0.598    1.120    0.136    2.759
#> 5  DN1     Sp4     0.448   0.385    0.844    0.107    3.557
#> 6  DN1     Sp4     0.640   0.470    0.861    0.093    3.420

# Matriz de comunidades e padronização para abundância relativa
head(anuros_comm)
#>      Sp10 Sp11 Sp2 Sp3 Sp4 Sp6 Sp7 Sp8 Sp1 Sp9 Sp5
#> DN1     1    8   1   1   6   8   4   0   0   0   0
#> DN2     0    0   0   0   1   1   4   2   0   0   0
#> DN3     1    0   0   0   0   0   0   0   0   0   0
```

```
#> DN4     0    0   2   2   1   0   0   0   6   0   0
#> DN5     0    0   3   0   1   0   4   1   0   0   0
#> FIG1    0    0   1   0   0   0   1   0   0   0   0

anuros_comm_rel <- decostand(anuros_comm, "total")

# Variáveis ambientais.
head(env)
#>      depth  area dits_bt_pond dist_for
#> DN1   0.50  3800          115     2650
#> DN2   0.60 54600          250     2500
#> DN3   0.80 29110          150     1800
#> DN4   1.00  1386          410      195
#> DN5   1.00   590          770      100
#> FIG1  0.15    30           25      135

## Prearação da matriz para receber os resultados do `for`
wITVResults <- data.frame(ITV = matrix(ncol = 1,
                                       nrow = length(unique(traits$pond))))
rownames(wITVResults) <- unique(traits$pond)

for(i in 1:length(unique(traits$pond))){
    commAux <- subset(traits, traits$pond == unique(traits$pond)[i])
    commAux$Species <- droplevels(factor(commAux$Species))
    spNames <- unique(commAux$Species)
    relAbund <- anuros_comm_rel[i, as.character(spNames)]
    traitsVector <- commAux$body_size
    spVector <- commAux$Species
    wITVResults[i, 1] <-  wITV(spIDs = spVector, traitVals = traitsVector,
relAbund = relAbund)
}

wITVResults$ITV
env$wITV <- wITVResults$ITV # NaN = locais com uma única espécie

## Remover NAs para executar o modelo linear
env2 <- na.omit(env)
head(env2)
#>      depth  area dits_bt_pond dist_for       wITV
#> DN1   0.50  3800          115     2650 0.82517670
#> DN2   0.60 54600          250     2500 0.23326457
#> DN4   1.00  1386          410      195 0.15341806
#> DN5   1.00   590          770      100 0.10298952
#> FIG1  0.15    30           25      135 0.00000000
#> FIG2  0.15    30           25      135 0.02338235
```

```
## Modelo linear
mod_itv <- lm(wITV ~ depth + area + dits_bt_pond + dist_for, data = env)

## Testar pressuposto da análise
par(mfrow = c(2, 2))
plot(mod_itv)

## Resultado
summary(mod_itv)
#>
#> Call:
#> lm(formula = wITV ~ depth + area + dits_bt_pond + dist_for, data = env)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -0.6298 -0.3937  0.0448  0.3387  0.5266
#>
#> Coefficients:
#>                Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   4.555e-01  2.113e-01   2.156   0.0449 *
#> depth         2.767e-01  3.789e-01   0.730   0.4746
#> area         -3.116e-06  9.276e-06  -0.336   0.7409
#> dits_bt_pond -3.653e-04  6.193e-04  -0.590   0.5626
#> dist_for      5.698e-05  1.419e-04   0.401   0.6928
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.4103 on 18 degrees of freedom
#>   (3 observations deleted due to missingness)
#> Multiple R-squared:  0.03155,    Adjusted R-squared:  -0.1837
#> F-statistic: 0.1466 on 4 and 18 DF,  p-value: 0.9621
```

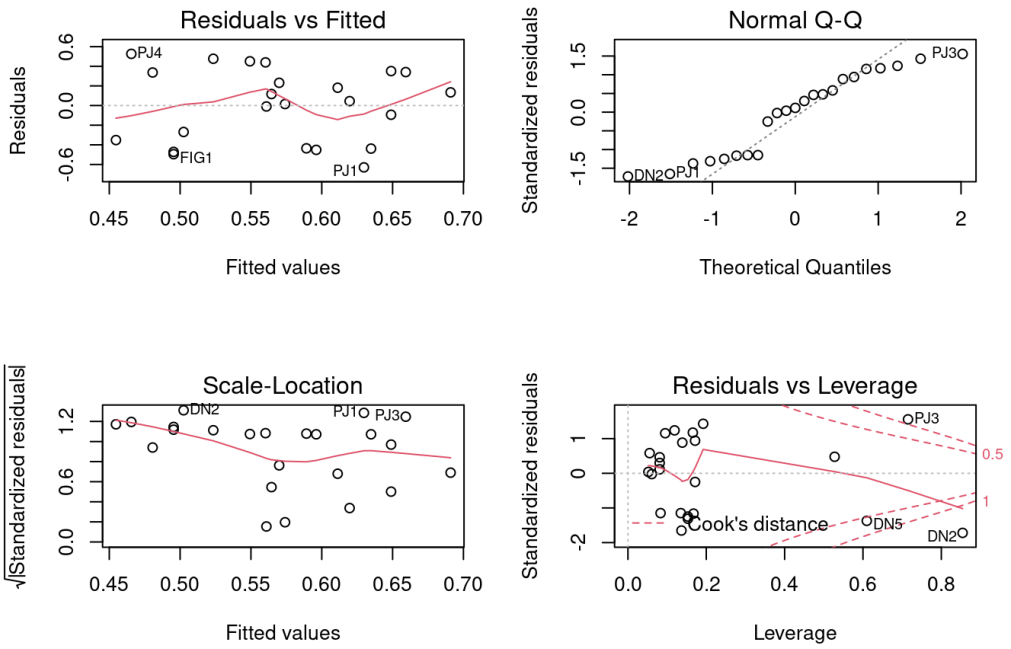


Figura 14.11: Gráficos diagnósticos do modelo linear ajustado para iWTV em função de características das poças.

Combinando os resultados das três análises é possível compreender que existem diferenças morfológicas entre as espécies de poças diferentes (componente substituição). Porém, é evidente que a variação dentro da espécie é bastante relevante para compreender a diversidade funcional de anuros. Na primeira análise, os resultados dessas quatro análises indicaram que a variação intraespecífica explica de 17% a 45% da variação morfológica nas metacomunidades de anuros. A segunda, por sua vez, demonstra que a variação morfológica entre espécies de poças diferentes representa o principal componente de variação, mas que a variação intraespecífica não pode ser ignorada. Por fim, ao combinar a métrica wITV com modelos lineares, percebe-se que as características das poças não determinam a contribuição da variação intraespecífica. Além disso, existe uma variação muito grande entre poças. Ao passo que em algumas poças a variação intraespecífica não contribui para a variação total (wITV = 0), em outras, este componente representou 100% da variação (wITV = 1). Os resultados obtidos nas análises das perguntas 1 a 3 indicam que utilizar somente a média dos atributos morfológicos pode refletir em interpretações incorretas em estudos que compararam a diversidade funcional no espaço/tempo (veja discussão em Dalmolin et al. 2020).

14.7 Para se aprofundar

14.7.1 Livros

Recomendamos a leitura de dois livros: i) Garnier et al. (2015) *Plant Functional Diversity: Organism Traits, Community Structure, and Ecosystem Properties*, e ii) de Bello et al. (2021) *Handbook of Trait-Based Ecology*. Os dois livros oferecem excelente oportunidade para se aprofundar no campo teórico da diversidade funcional e, além disso, o livro liderado pelo pesquisador Francesco de Bello fornece

diversas aplicações analíticas na linguagem R. Outro recurso excelente foi publicado por Mammola et al. (2021) e serve para se aprofundar nas diferentes medidas da diversidade funcional.

14.7.2 Links

Um passo importante nos estudos de Diversidade Funcional é encontrar atributos funcionais dos organismos estudados. Abaixo indicamos uma lista de potenciais bases de dados:

- COMADRE - Animal Matrix Database
- COMPADRE - Plant Matrix Database
- Elton Traits 1.0
- TetraDENSITY - Density estimates in terrestrial vertebrates
- TRY - Plant Trait Database
- World Spider Trait

14.8 Exercícios

14.1 Utilize os dados `aviurba` do pacote `ade4` para testar o efeito de variáveis ambientais na dispersão (FDis) e regularidade funcional (FEve). Utilize modelos lineares (veja Capítulo 7) para testar quais variáveis ambientais são as mais importantes para a dispersão e regularidade funcional. Além disso, faça um boxplot (veja Capítulo 6) comparando os valores de FDis e FEve entre as categorias das variáveis ambientais mais relevantes.

14.2 Utilize os dados `mafragh` do pacote `ade4` para teste o efeito da das variáveis `conductivity`, `silt` e `K2O` na diversidade funcional (método de Petchey & Gaston). Utilize modelos lineares (regressão múltipla, veja Capítulo 7) para testar a relação entre essas variáveis e discuta: (a) qual variável mais importante (se houver) e (b) se as conclusões são coerentes tendo como base os pressupostos dos modelos lineares. Além disso, caso exista alguma relação significativa, faça um gráfico (*scatterplot*, veja Capítulo 6) da relação da variável mais importante e a diversidade funcional.

14.3 Utilize os dados `mafragh` do pacote `ade4` para comparar a composição filogenética e funcional em áreas com alta e baixa concentração de potássio. Para fazer esta comparação será necessário transformar a matriz de atributos funcionais e a árvore filogenética em matrizes de distância e, depois, utilizar o CWM para criar uma matriz de localidades por composição funcional ou filogenética. Depois, você poderá usar a matriz CWM para testar potenciais diferenças entre concentrações com PERMANOVA e para visualizar com PCoA.

Soluções dos exercícios.

Capítulo 15

Dados geoespaciais

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(ecodados)
library(here)
library(tidyverse)
library(sf)
library(raster)
library(rgdal)
library(spData)
library(rnaturalearth)
library(geobr)
library(ggplot2)
library(ggspatial)
library(tmap)
library(tmaptools)
library(grid)
library(mapview)
library(leaflet)
library(viridis)
library(knitr)
library(sidrar)
library(landscapetools)

## Dados
world <- world
volcano <- volcano
geo_anfibios_locais <- ecodados::geo_anfibios_locais
geo_anfibios_especies <- ecodados::geo_anfibios_especies
geo_vetor_nascentes <- ecodados::geo_vetor_nascentes
geo_vetor_hidrografia <- ecodados::geo_vetor_hidrografia
geo_vetor_cobertura <- ecodados::geo_vetor_cobertura
geo_vetor_rio_claro <- ecodados::geo_vetor_rio_claro
geo_vetor_brasil <- ecodados::geo_vetor_brasil
geo_vetor_brasil_anos <- ecodados::geo_vetor_brasil_anos
geo_vetor_am_sul <- ecodados::geo_vetor_am_sul
geo_vetor_biomas <- ecodados::geo_vetor_biomas
geo_vetor_mata_atlantica <- ecodados::geo_vetor_mata_atlantica
geo_raster_srtm <- ecodados::geo_raster_srtm
geo_raster_bioclim <- ecodados::geo_raster_bioclim
geo_raster_globcover_mata_atlantica <- ecodados::geo_raster_globcover_mata_
atlantica
```

15.1 Contextualização

Nesta seção, vamos fazer uma breve introdução aos principais conceitos sobre a manipulação e visualização de dados geoespaciais no R. Iremos abordar temas de forma teórica e prática, utilizando a linguagem R, focando em: i) formatos de dados vetoriais e dados raster, ii) Sistemas de Referências de Coordenadas e unidades (geográficas e projetadas), iii) fontes de dados, iv) importar e exportar dados, v) descrição de objetos geoespaciais e vi) principais operações (atributos, espaciais e geométricas). Num segundo momento, criaremos mapas com seus principais elementos como mapas principal e secundário, título, legenda, barra de escala, indicador de orientação (Norte), gride de coordenadas, descrição do Sistema de Referência de Coordenadas e informações de origem dos dados. Por fim, apresentaremos exemplos de aplicações de análises geoespaciais para dados ecológicos, focadas em: i) agregar informações sobre a biodiversidade, ii) preparar dados para compor variáveis preditoras, e iii) como fazer predições espaciais de distribuição de uma espécie e riqueza de espécies.

Esse capítulo segue parte da estrutura organizada por Lovelace et al. (2019), principalmente os Capítulos 2 a 8, sendo adaptado para atender aos principais requisitos que julgamos necessários a estudos ecológicos. Entretanto, não foi possível cobrir todos os assuntos sobre o uso de dados geoespaciais no R, sendo um tema muito extenso que requer a leitura de livros especializados na área como: i) Mas et al. (2019) Análise espacial com R, ii) Wegmann, Leutner & Dech (2016) *Remote Sensing and GIS for Ecologists: Using Open Source Software*, iii) Wegmann, Schwalb-Willmann & Dech (2020) *An Introduction to Spatial Data Analysis Remote Sensing and GIS with Open Source Software*, e iv) Fletcher & Fortin (2018) *Spatial ecology and conservation modeling: Applications with R*. Outros livros sobre a análise geoespacial no R podem ser consultados no Capítulo 11 - *Geospatial* do *Big Book of R*.

15.2 Vetor

Dados vetoriais são usados para mapear fenômenos ou objetos espacialmente explícitos que possuem localização ou dimensões bem definidas, representado a partir de formas geométricas (como pontos, linhas e polígonos) e possuem a possibilidade de ter associado a eles informações tabulares. A tabela de atributos é uma tabela que inclui dados geoespaciais e dados alfanuméricos. Os dados geoespaciais são representados por feições geolocalizadas espacialmente (ponto, linha ou polígono), e os dados alfanuméricos (tabela de dados). Dessa forma, a tabela de atributos reúne informações sobre cada feição e pode ser utilizada para realizar filtros ou agregações dos dados de cada feição (Figura 15.1).

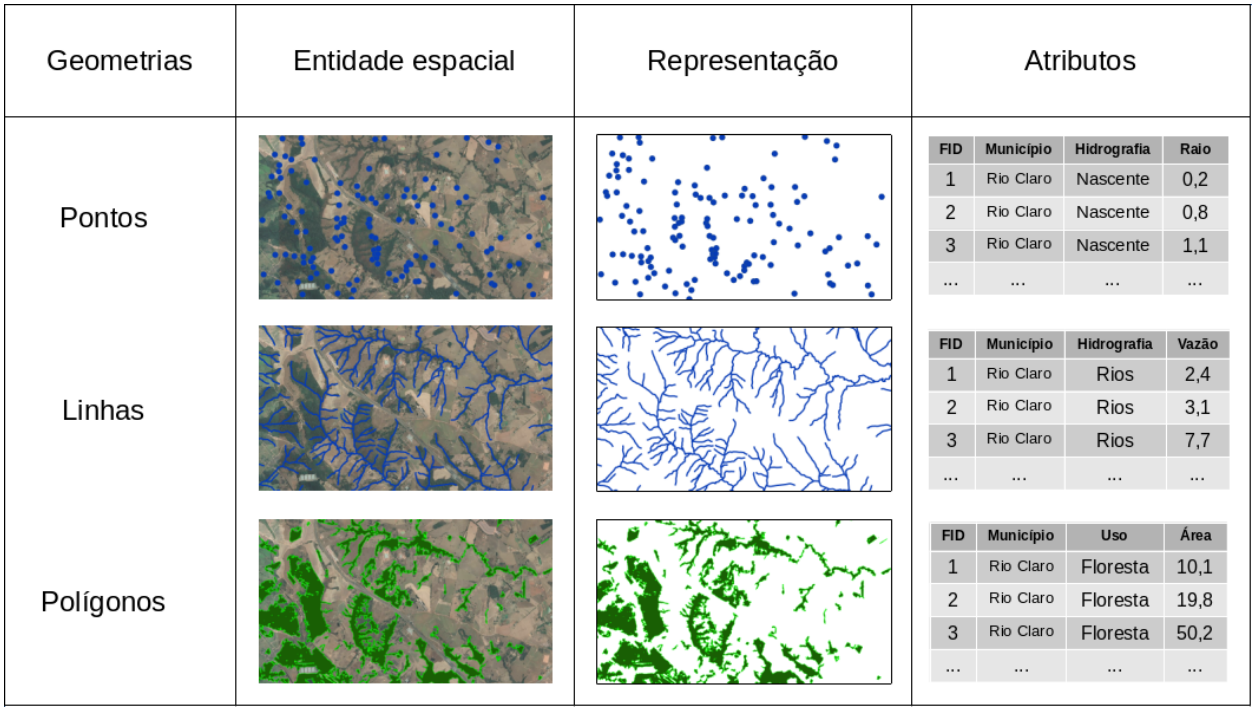


Figura 15.1: Representação das geometrias de ponto, linha e polígono e atributos. Adaptado de Olaya (2020).

15.2.1 sf: principal pacote no R para dados vetoriais

Atualmente o principal pacote para trabalhar com dados vetoriais no R é o `sf`, que implementou o *Simple Feature* (Pebesma 2018). Entretanto, outro pacote pode ser tão versátil quanto o `sf`, no caso o `terra`, com algumas mudanças na sintaxe que não abordaremos nesse livro por questões de redução de espaço.

Os tipos de geometrias apresentadas são representados por diferentes classes: `POINT`, `LINESTRING` e `POLYGON` para apenas uma feição de cada tipo de geometria; `MULTIPOINT`, `MULTILINESTRING` e `MULTIPOLYGON` para várias feições de cada tipo de geometria e; `GEOMETRYCOLLECTION` para várias feições e tipos de geometrias e classes.

Ao olharmos as informações de um objeto da classe `sf`, podemos notar diversas informações que descrevem o mesmo, numa espécie de cabeçalho:

- **resumo do vetor**: indica o número de feições (linhas) e campos (colunas)
- **tipo da geometria**: umas das sete classes (ou mais outras) listadas anteriormente
- **dimensão**: número de dimensões, geralmente duas (XY)
- **bbox (bordas)**: coordenadas mínimas e máximas da longitude e latitude
- **informação do CRS**: `epsg` ou `proj4string` indicando o CRS (*Coordinate Reference System*)
- **tibble**: tabela de atributos, com destaque para a coluna `geom` ou `geometry` que representa cada feição ou geometria

```
## Dados vetoriais de polígonos do mundo
data(world)
world
```

Podemos fazer um mapa simples utilizando a função `plot()` desse objeto. Para facilitar, escolheremos apenas a primeira coluna `[1]` (Figura 15.2). Caso não escolhermos apenas uma coluna, um mapa para cada coluna será plotado.

> 👉 **Importante**
> Faremos mapas mais elaborados na seção de visualização de dados geoespaciais deste capítulo.

```
## Plot dos polígonos do mundo
plot(world[1], col = viridis::viridis(100), main = "Mapa do mundo")
```

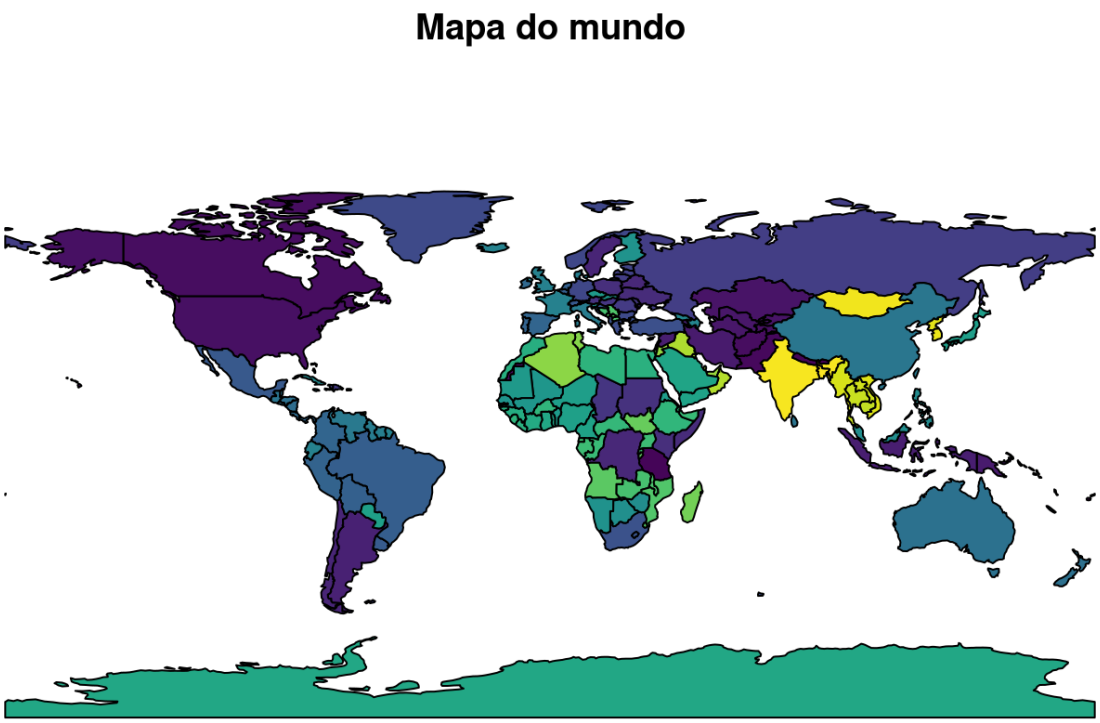


Figura 15.2: Mapa vetorial do mundo.

15.3 Raster

Os dados no formato raster consistem em uma matriz (com linhas e colunas) em que os elementos representam células, geralmente igualmente espaçadas (pixels; Figura 15.3). As células dos dados raster possuem duas informações: i) identificação das células (IDs das células) para especificar sua posição na matriz (Figura 15.3 A) e; ii) valores das células (Figura 15.3 B), que geralmente são coloridos para facilitar a interpretação da variação dos valores no espaço (Figura 15.3 C). Além disso, valores ausentes ou não amostrados são representados por `NA`, ou seja, *not available* (Figura 15.3 B e C).

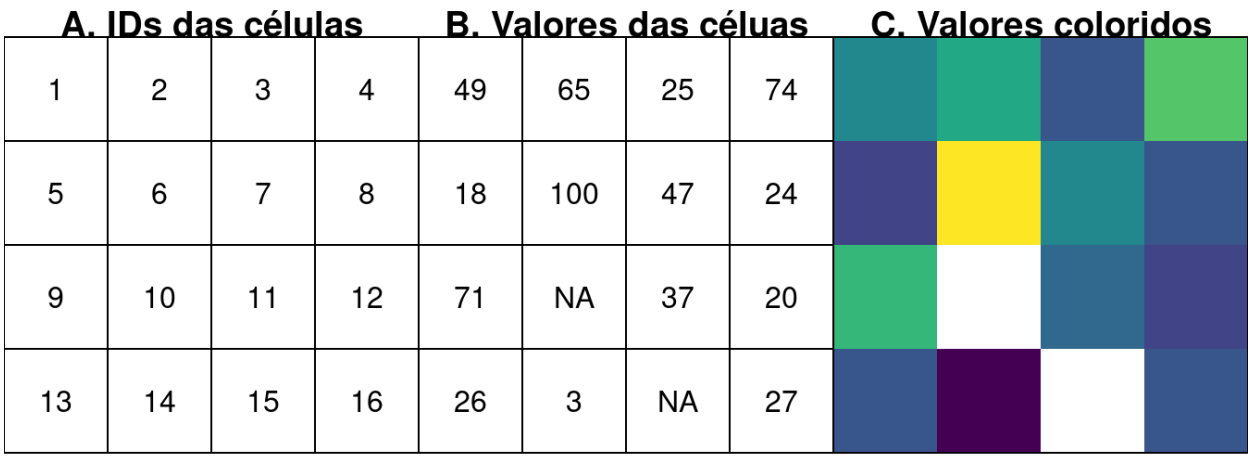


Figura 15.3: Raster: (A) IDs das células, (B) valores das células, (C) células coloridas. Adaptado de Lovelace et al. (2019).

Podemos ainda fazer uma comparação com as representações de dados vetoriais vistos na Figura 15.1, mas agora no formato raster (Figura 15.4).

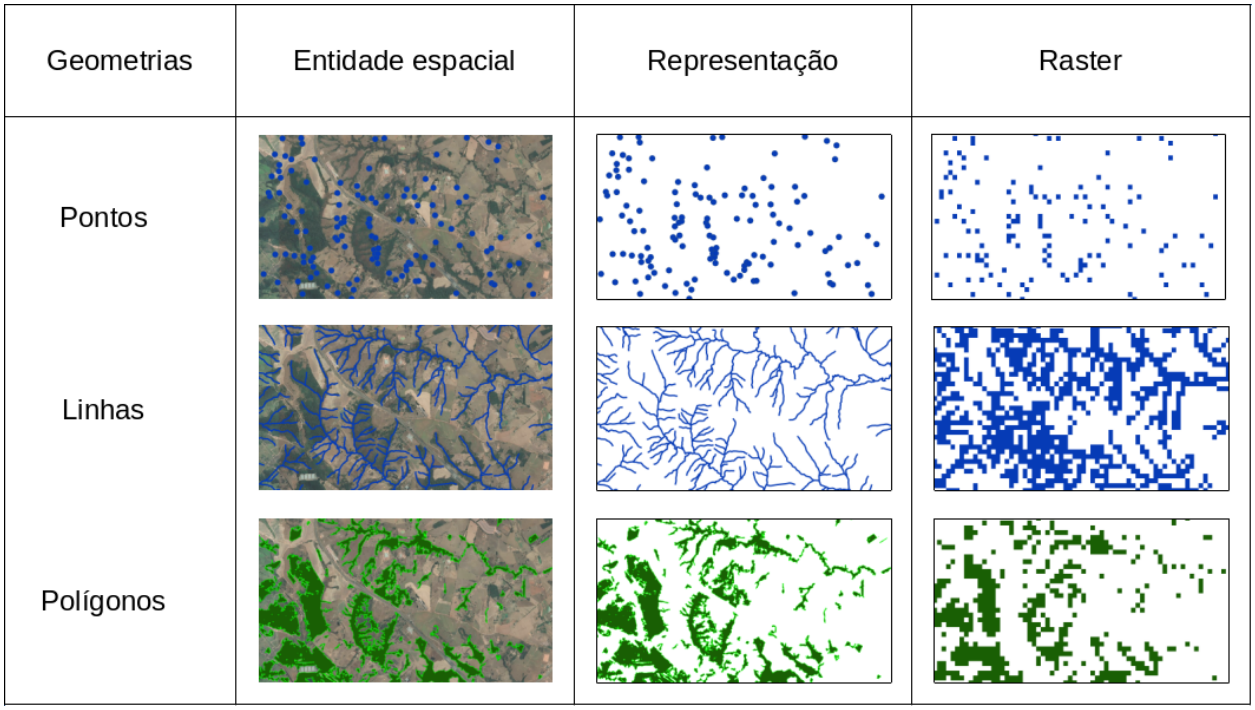


Figura 15.4: Representação das geometrias de ponto, linha e polígono no formato raster. Adaptado de Olaya (2020).

15.3.1 raster: principal pacote no R para dados raster

Atualmente, o principal pacote para trabalhar com dados raster é o *raster*, apesar de existir outros dois: *terra* e *stars*, com algumas mudanças na sintaxe que não abordaremos neste livro.

O pacote `raster` fornece uma ampla gama de funções para criar, importar, exportar, manipular e processar dados raster no R. O objeto raster criado à partir do pacote `raster` pode assumir três classes: `RasterLayer`, `RasterStack` e `RasterBrick`.

A classe `RasterLayer` representa apenas uma camada raster. Para criar ou importar um raster no R podemos utilizar a função `raster::raster()`. Observando essa classe, podemos notar as seguintes informações:

- **class**: classe raster do objeto `raster`
- **dimensions**: número de linhas, colunas e células
- **resolution**: largura e altura da célula
- **extent**: coordenadas mínimas e máximas da longitude e latitude
- **crs**: Sistema de Referência de Coordenadas (*CRS*)
- **source**: fonte dos dados (memória ou disco)
- **names**: nome das camadas
- **values**: valores máximos e mínimos das células

Vamos utilizar os dados `volcano`, que possui informações topográficas (elevação) do vulcão *Maunga Whau* de Auckland na Nova Zelândia.

```
## Dados de altitude de um vulcão
volcano[1:5, 1:5]
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]  100  100  101  101  101
#> [2,]  101  101  102  102  102
#> [3,]  102  102  103  103  103
#> [4,]  103  103  104  104  104
#> [5,]  104  104  105  105  105
```

Vamos transformar essa matriz de dados em um raster com a função `raster::raster()`.

```
## Rasterlayer
raster_layer <- raster::raster(volcano)
raster_layer
#> class      : RasterLayer
#> dimensions : 87, 61, 5307  (nrow, ncol, ncell)
#> resolution : 0.01639344, 0.01149425  (x, y)
#> extent     : 0, 1, 0, 1  (xmin, xmax, ymin, ymax)
#> crs        : NA
#> source     : memory
#> names      : layer
#> values     : 94, 195  (min, max)
```

Um mapa simples do objeto raster pode ser obtido utilizando a função `plot()`, do próprio pacote `raster` (Figura 15.5).

```
## Plot raster layers
plot(raster_layer, col = viridis::viridis(n = 100))
```

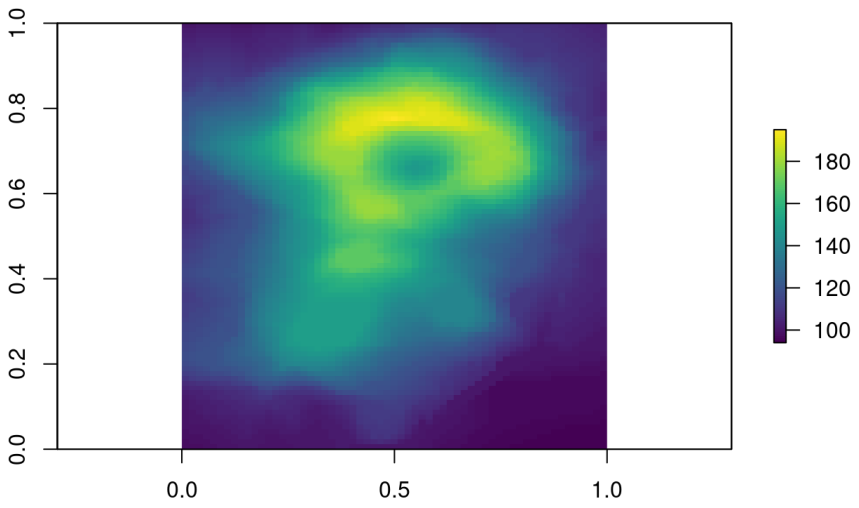


Figura 15.5: Mapa simples de um `RasterLayer`.

Além da classe `RasterLayer`, há mais duas classes que trabalham com múltiplas camadas: `RasterBrick` e `RasterStack`. Elas diferem em relação ao formato dos arquivos suportados, tipo de representação interna e velocidade de processamento.

A classe `RasterBrick` geralmente corresponde à importação de um único arquivo de imagem de satélite multiespectral (multicamadas) ou a um único objeto com várias camadas na memória. A função `raster::brick()` cria um objeto `RasterBrick`.

```
## Raster layers
raster_layer1 <- raster_layer
raster_layer2 <- raster_layer * raster_layer
raster_layer3 <- sqrt(raster_layer)
raster_layer4 <- log10(raster_layer)

## Raster brick
raster_brick <- raster::brick(raster_layer1, raster_layer2,
                              raster_layer3, raster_layer4)
raster_brick
#> class      : RasterBrick
#> dimensions : 87, 61, 5307, 4  (nrow, ncol, ncell, nlayers)
#> resolution : 0.01639344, 0.01149425  (x, y)
#> extent     : 0, 1, 0, 1  (xmin, xmax, ymin, ymax)
#> crs        : NA
#> source     : memory
#> names      :      layer.1,      layer.2,      layer.3,      layer.4
#> min values :    94.000000,  8836.000000,     9.695360,     1.973128
#> max values :   195.000000, 38025.000000,    13.964240,     2.290035
```

Ao utilizarmos a função `plot()` do pacote `raster`, podemos visualizar os raster contidos no objeto `RasterBrick` (Figura 15.6).

```
## Plot raster brick
plot(raster_brick, col = viridis::viridis(n = 25), main = "")
```

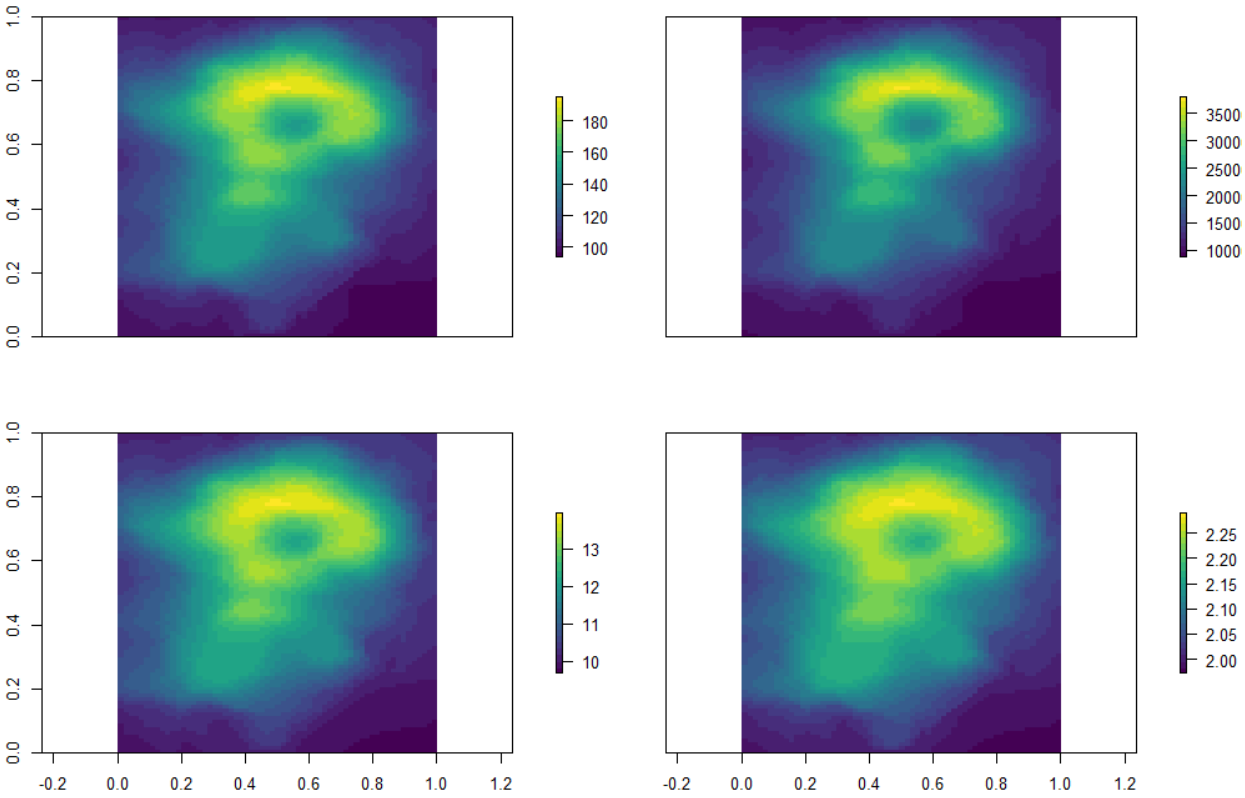


Figura 15.6: Mapas simples de um raster `RasterBrick`.

Já a classe `RasterStack` permite conectar vários objetos raster armazenados em arquivos diferentes ou vários objetos no ambiente do R. Um `RasterStack` é uma lista de objetos `RasterLayer` com a mesma extensão, resolução e CRS. Uma maneira de criá-lo é com a junção de vários objetos geoespaciais já existentes no ambiente do R ou listar vários arquivos raster em um diretório armazenado no disco. A função `raster::stack()` cria um objeto `RasterStack`.

Outra diferença é que o tempo de processamento, para objetos `RasterBrick` geralmente é menor do que para objetos `RasterStack`. A decisão sobre qual classe `Raster` deve ser usada depende principalmente do caráter dos dados de entrada.

```
## Raster layers
raster_layer1 <- raster_layer
raster_layer2 <- raster_layer * raster_layer
raster_layer3 <- sqrt(raster_layer)
raster_layer4 <- log10(raster_layer)

## Raster stack
raster_stack <- raster::stack(raster_layer1, raster_layer2,
                              raster_layer3, raster_layer4)
raster_stack
#> class      : RasterStack
#> dimensions : 87, 61, 5307, 4  (nrow, ncol, ncell, nlayers)
#> resolution : 0.01639344, 0.01149425  (x, y)
#> extent     : 0, 1, 0, 1  (xmin, xmax, ymin, ymax)
#> crs        : NA
#> names      :      layer.1,      layer.2,      layer.3,      layer.4
```

```
#> min values :    94.000000,  8836.000000,     9.695360,     1.973128
#> max values :   195.000000, 38025.000000,    13.964240,     2.290035
```

Da mesma forma, ao utilizar a função `plot()` do pacote `raster`, podemos visualizar os raster contidos no objeto `RasterStack` (Figura 15.7).

```
## Plot raster stack
plot(raster_stack, col = viridis::viridis(n = 25), main = "")
```

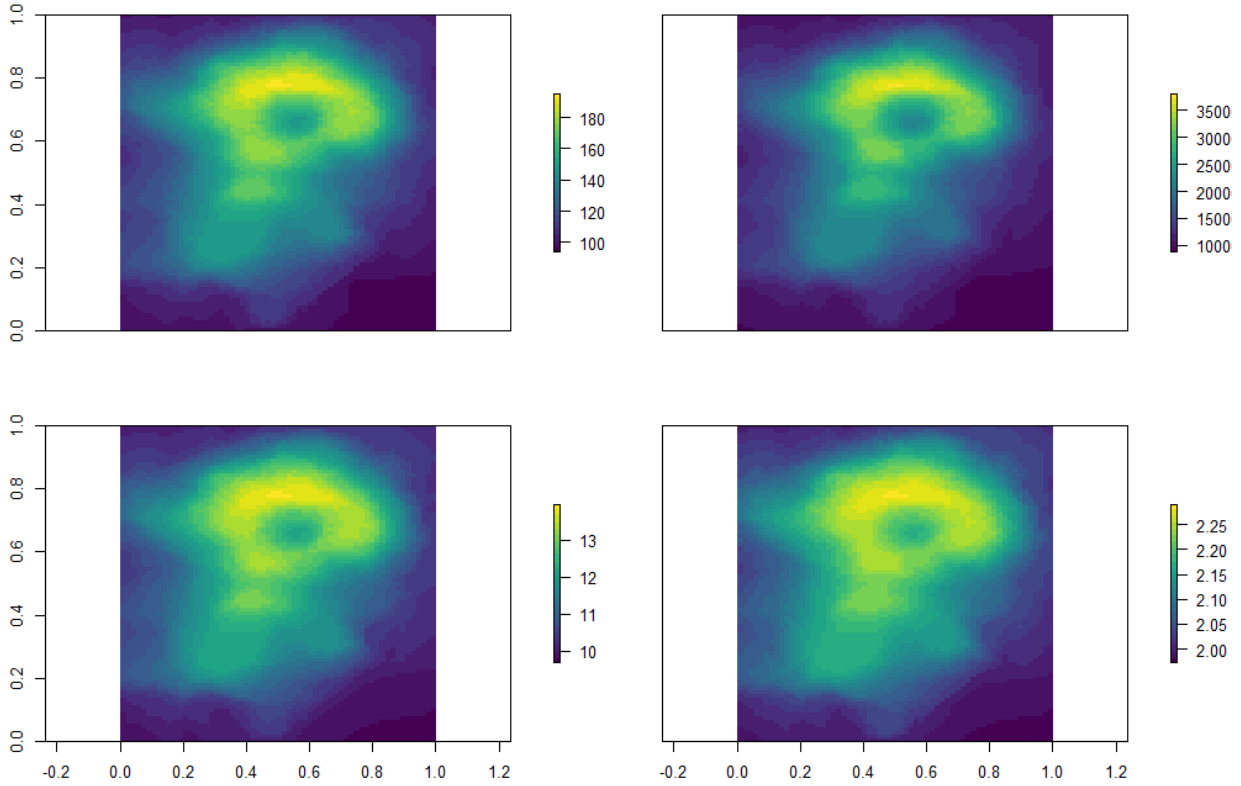


Figura 15.7: Mapas simples de um raster `RasterStack`.

15.4 Sistema de Referência de Coordenadas e Unidades

Os dados geoespaciais (vetor e raster) possuem ainda um outro componente fundamental que é o Sistema de Referência de Coordenadas, ou do inglês *Coordinate Reference System (CRS)*. Esse componente define a referência espacial dos elementos geoespaciais (vetor e raster) na superfície da Terra. Ele é composto por dois principais conceitos: primeiro, que tipos de unidades estão sendo utilizadas para a representação geográfica, podendo assumir dois tipos - ângulos ou metros, que definem o Sistema de Coordenadas Geográficas e o Sistema de Coordenadas Projetadas, respectivamente. O segundo componente é o *datum*, que é a relação do sistema de coordenadas (geográfica ou projetada) com a superfície da Terra. Esse último componente faz parte de uma área da Cartografia denominada Geodésia que estuda a forma e dimensões da Terra, campo gravitacional e a localização de pontos fixos e sistemas de coordenadas. O livro de Lapaine & Usery (2017) é um excelente material para se aprofundar nesse assunto.

15.4.1 Sistema de Coordenadas Geográficas

O Sistema de Coordenadas Geográficas utiliza ângulos (graus) para representar feições na superfície da Terra através de dois valores: longitude e latitude. A longitude representa o eixo Leste-Oeste e a latitude o eixo Norte-Sul. Nesse sistema, a superfície da Terra é representada geralmente por uma superfície elipsoidal, pois a Terra é ligeiramente achatada nos polos devido ao seu movimento de rotação.

15.4.2 Sistema de Coordenadas Projetadas

O Sistema de Coordenadas Projetadas utiliza um Sistema Cartesiano de Coordenadas em uma superfície plana. Dessa forma, a partir de uma origem traçam-se eixos X e Y e uma unidade linear é utilizada, como o metro. Todos as projeções feitas de sistemas geoespaciais convertem uma superfície tridimensional em uma superfície plana bidimensional. Sendo assim, essa conversão traz consigo algum tipo de distorção em relação à porção real, podendo ser distorções em: i) formas locais, ii) áreas, iii) distâncias, iv) flexão ou curvatura, v) assimetria ou vi) lacunas de continuidade. Dessa forma, um sistema de coordenadas projetadas pode preservar somente uma ou duas dessas propriedades.

Existem três grandes grupos de projeções: i) cilíndricos, ii) cônicos e iii) planares. Na projeção cilíndrica, a superfície da Terra é mapeada em um cilindro, criada tocando a superfície da Terra ao longo de uma ou duas linhas de tangência, sendo utilizada com mais frequência para mapear todo o globo tendo como exemplo mais conhecido a Projeção Universal Transversa de Mercator (UTM). Na projeção cônica, a superfície da Terra é projetada em um cone ao longo de uma linha ou duas linhas de tangência, de modo que as distorções são minimizadas ao longo das linhas e aumentam com a distância das mesmas sendo, portanto, mais adequada para mapear áreas de latitudes médias, tendo como exemplo mais conhecido a Projeção Cônica Equivalente de Albers e a Projeção Cônica Conforme de Lambert. E na projeção plana, também denominada Projeção Azimutal, o mapeamento toca o globo em um ponto ou ao longo de uma linha de tangência, sendo normalmente utilizado no mapeamento de regiões polares, sendo a mais comum a Projeção Azimutal Equidistante, a mesma utilizada na bandeira da ONU.

15.4.3 Datum

Como dito anteriormente, o *datum* é a relação do sistema de coordenadas com a superfície da Terra. Ele representa o ponto de intersecção do elipsoide de referência com a superfície da Terra (geoide, a forma verdadeira da Terra), compensando as diferenças do campo gravitacional da Terra. Existem dois tipos de datum: i) local e ii) geocêntrico. Em um *datum* local, como o SAD69 - *South American Datum 1969*, o elipsoide de referência é deslocado para se alinhar com a superfície em um determinado local, por exemplo, na América do Sul. Já em um *datum* geocêntrico, como WGS84 - *World Geodetic System 1984*, o centro do elipsoide é o centro de gravidade da Terra e a precisão das projeções não é otimizada para um local específico do globo.

No Brasil, desde 2015, o Instituto Brasileiro de Geografia e Estatística (IBGE) ajudou a desenvolver e reafirmou o uso do *datum* SIRGAS2000 - *Sistema de Referencia Geocéntrico para las Américas 2000* para todos os mapeamentos realizados no Brasil, um esforço conjunto para adotar o mesmo *datum* em toda a América. Mais sobre esse *datum* pode ser lido aqui: SIRGAS2000.

15.4.4 Sistema de Referência de Coordenadas (CRS) no R

No R, há duas formas principais de representar um Sistema de Referência de Coordenadas: i) código `epsg` e ii) `proj4string`. O código EPSG (*European Petroleum Survey Group*) é uma sequência de números curta, referindo-se apenas a um CRS. O site epsg.io permite consultar diversas informações como procurar por um código, representação de mapas e fazer transformações de CRS.

Já o `proj4string` permite mais flexibilidade para especificar diferentes parâmetros, como o tipo de projeção, datum e elipsoide. Dessa forma, é possível especificar muitas projeções, ou mesmo modificar as projeções existentes, tornando a representação `proj4string` mais complexa e flexível.

Além disso, ainda é possível consultar uma extensa lista de CRSs no site spatialreference.org, que fornece descrições em diversos formatos, baseados em GDAL e Proj.4. Essa abordagem permite consultar uma URL que pode produzir uma referência espacial em um formato que seu software SIG ou o R pode utilizar como referência.

Os pacotes (geo)espaciais no R suportam uma ampla variedade de CRSs e usam a biblioteca PROJ. A função `rgdal::make_EPSG()` retorna um `data frame` das projeções disponíveis, com informações dos códigos `epsg` e `proj4string` numa mesma tabela, facilitando a busca e uso de CRSs (Tabela 15.1).

```
## Listagem dos Sistemas de Referências de Coordenadas no R
crs_data <- rgdal::make_EPSG()
head(crs_data)
```

Tabela 15.1: Listagem de Sistemas de Referências de Coordenadas disponíveis no R, com informações dos códigos `epsg` *e* `proj4string`

Code	note	prj4	prj_method
3819	HD1909	+proj=longlat +ellps=bessel +no_defs +type=crs	(null)
3821	TWD67	+proj=longlat +ellps=aust_SA +no_defs +type=crs	(null)
3822	TWD97	+proj=geocent +ellps=GRS80 +units=m +no_defs +type=crs	(null)
3823	TWD97	+proj=longlat +ellps=GRS80 +no_defs +type=crs	(null)
3824	TWD97	+proj=longlat +ellps=GRS80 +no_defs +type=crs	(null)
3887	IGRS	+proj=geocent +ellps=GRS80 +units=m +no_defs +type=crs	(null)

15.5 Principais fontes de dados geoespaciais

Existem diversas fontes de dados geoespaciais em diferentes bases de dados disponíveis gratuitamente. Geralmente essas bases de dados são disponibilizadas separadamente em apenas dados vetoriais e dados raster. Para dados vetoriais, grande parte dos dados disponibilizados são utilizados em mapas como limites políticos, limites de biomas ou distribuição de espécies para polígonos; estradas e rios para dados lineares, ou ainda pontos de ocorrência de espécies ou comunidades, ou medidas tomadas em campo sobre condições naturais como clima ou relevo, como pontos. Entretanto, é sempre recomendado o uso de bases oficiais, principalmente em relação a dados vetoriais de limites políticos. Para tanto, é fundamental buscar as bases oficiais de cada país, entretanto, há bases que podem ser utilizadas globalmente, como veremos.

Sobre as bases de dados raster, há uma infinidade de dados para diferentes objetivos, mas grande parte deles são relativos a condições ambientais, representando uma variável de interesse de forma contínua no espaço, como temperatura, precipitação, elevação, etc.

Há uma compilação de dados geoespaciais vetoriais e raster feita por Marcus Vinícius Alves de Carvalho e Angelica Carvalho Di Maio, chamada GeoLISTA. Entretanto, como as bases de dados tendem a ser muito dinâmicas, é possível que muitas bases tenham surgido e desaparecido desde a listagem realizada.

Além das bases de dados, há pacotes específicos no R que fazem o download de dados vetoriais e rasters, facilitando a aquisição e reprodutibilidade. Para conferir uma listagem completa de pacotes para diversas análises espaciais, veja CRAN Task View: Analysis of Spatial Data.

15.5.1 Vetor

Dentre as bases vetoriais, destacamos as seguintes na Tabela 15.2.

Tabela 15.2. Principais bases de dados vetoriais para o Brasil e o mundo.

Bases de dados	Descrição
IBGE	Limites territoriais e censitários do Brasil
FBDS	Uso da terra, APP e hidrografia - Mata Atlântica e Cerrado
GeoBank	Dados geológicos do Brasil
Pastagem.org	Dados de pastagens e gado para o Brasil
CanaSat	Dados de cana-de-açúcar para o Brasil
CSR Maps	Diversos dados vetoriais e raster para o Brasil
Ecoregions	Dados de biorregiões e biomas do mundo
UN Biodiversity Lab	Diversas bases de dados para o mundo
Biodiversity Hotspots	Dados dos limites dos *Hotspots* de Biodiversidade
IUCN Red List of Threatened Species	Dados dos limites das distribuições das espécies para o mundo
Map of Life (MOL)	Dados da distribuição de espécies e outros dados para o mundo
Key Biodiversity Areas	Dados dos limites das *Key Biodiversity Areas*
HydroSHEDS	Informações hidrológicas do mundo
Global Roads Inventory Project (GRIP)	Dados de estradas do mundo todo
Database of Global Administrative Areas (GADM)	Limites de áreas administrativas do mundo
Natural Earth	Diversos limites para o mundo
Protected Planet	Limites de áreas protegidas para o mundo
Global Biological Information Facility (GBIF)	Dados de ocorrências de espécies para o mundo
Species Link	Dados de ocorrências de espécies para o Brasil

Bases de dados	Descrição
Global Invasive Species Information Network (GISIN)	Dados de ocorrências de espécies invasoras para o Mundo

15.5.2 Raster

Dentre as bases raster, destacamos as seguintes na Tabela 15.3.

Tabela 15.3. Principais bases de dados raster para o Brasil e o mundo.

Bases de dados	Descrição
MapBiomas	Uso e cobertura da terra para o Brasil, Panamazonia Legal Chaco e Mata Atântica de 1985 a 2020
Bahlu	Distribuições históricas de terras agrícolas e pastagens para todo o Brasil de 1940 a 2012
USGS	Dados de diversos satélites livres para o mundo
SRTM	Dados de elevação para o mundo
Geoservice Maps	Dados de elevação e florestas para o mundo
Global Forest Watch	Dados de florestas para o mundo
GlobCover	Dados de uso e cobertura da terra para todo o planeta
Landcover	Dados de uso e cobertura da terra para todo o planeta
Global Human Footprint	Dados de pegada ecológica para o mundo
GHSL - Global Human Settlement Layer	Dados e ferramentas abertos e gratuitos para avaliar a presença humana no planeta
Land-Use Harmonization (LUH2)	Dados atuais e previsões de uso da terra
ESA Climate Change Initiative	Arquivos globais de observação da Terra nos últimos 30 anos da Agência Espacial Europeia (ESA)
WorldClim	Dados climáticos para o mundo
CHELSA	Dados climáticos para o mundo
EarthEnv	Dados de cobertura da terra, nuvens, relevo e hidrografia
SoilGrids	Dados de solo para o mundo
Global Wetlands	Dados de áreas úmidas para o mundo
Global Surface Water Explorer	Dados de águas superficiais para o mundo
MARSPEC	Dados de condições do oceano para o mundo
Bio-ORACLE	Dados de condições do oceano para o mundo

15.5.3 Pacotes do R

Dentre os pacotes no R para download de dados geoespaciais, destacamos os seguintes na Tabela 15.4.

Tabela 15.4. Principais pacotes no R para download de dados vetoriais e raster.

Pacotes	Descrição
`geobr`	Carrega *shapefiles* de Conjuntos de Dados Espaciais Oficiais do Brasil
`rnaturalearth`	Dados do mapa mundial da *Natural Earth*
`rworldmap`	Visualização de dados globais
`spData`	Conjuntos de dados para análise espacial
`OpenStreetMap`	Acesso para abrir imagens raster de mapas de ruas
`osmdata`	Baixar e importar dados do *OpenStreetMap*
`geonames`	Interface para o serviço da Web de consulta espacial '*Geonames*'
`rgbif`	Interface para o *Global 'Biodiversity' Information Facility API*
`maptools`	Ferramentas para lidar com objetos geoespaciais
`marmap`	Importar, traçar e analisar dados batimétricos e topográficos
`oce`	Fonte e processamento de dados oceanográficos
`envirem`	Geração de variáveis ENVIREM
`sdmpredictors`	Conjuntos de dados preditor de modelagem de distribuição de espécies
`metScanR`	Encontrar, mapear e coletar dados e metadados ambientais
`ClimDown`	Biblioteca de redução de escala do clima para a produção diária do modelo climático
`rWBclimate`	Acessar dados climáticos do Banco Mundial
`rnoaa`	Dados meteorológicos 'NOAA' de R
`RNCEP`	Obter, organizar e visualizar dados meteorológicos NCEP
`smapr`	Aquisição e processamento de dados ativos-passivos (SMAP) de umidade do solo da NASA

15.6 Importar e exportar dados geoespaciais

Agora que sabemos o que são dados geoespaciais e em quais bases de dados podemos buscar e baixar esses dados, veremos seus principais formatos e como importá-los e exportá-los do R.

15.6.1 Principais formatos de arquivos geoespaciais

Há diversos formatos de arquivos geoespaciais, alguns específicos para dados vetoriais e raster, e outros no formato de banco de dados geoespaciais, como PostGIS, que podem armazenar ambos os formatos.

Entretanto, todos os formatos para serem importados para o R usam o GDAL (*Geospatial Data Abstraction Library*), uma interface unificada para leitura e escrita de diversos formatos de arquivos geoespaciais, sendo utilizado também por uma série de softwares de GIS como QGIS, GRASS GIS e ArcGIS.

Dentre esses formatos, destacamos os seguintes na Tabela 15.5.

Tabela 15.5. Principais formatos de arquivos geoespaciais. Adaptado de: Lovelace et al. (2019).

Nome	extensão	Descrição	Tipo	Modelo
ESRI Shapefile	.shp (arquivo principal)	Formato popular que consiste em pelo menos quatro arquivos: .shp (feição), .dbf (tabela de atributos), .shx (ligação entre .shp e .dbf) e .prj (projeção)	Vetor	Parcialmente aberto
GeoJSON	.geojson	Estende o formato de troca JSON incluindo um subconjunto da representação de recurso simples	Vetor	Aberto
KML	.kml	Formato baseado em XML para visualização espacial, desenvolvido para uso com o *Google Earth*. O arquivo KML compactado forma o formato KMZ	Vetor	Aberto
GPX	.gpx	Esquema XML criado para troca de dados de GPS	Vetor	Aberto
GeoTIFF	.tif/.tiff	Formato raster popular. Um arquivo TIFF contendo metadados espaciais adicionais.	Raster	Aberto
Arc ASCII	.asc	Formato de texto em que as primeiras seis linhas representam o cabeçalho raster, seguido pelos valores das células raster organizadas em linhas e colunas	Raster	Aberto
NetCDF	.nc	NetCDF (*Network Common Data Form*) é um conjunto de bibliotecas de software e formatos de dados independentes que suportam a criação, acesso e compartilhamento de dados científicos orientados a arrays	Raster	Aberto
BIL	.bil/.hdr	BIL (Banda intercalada por linha) são métodos comuns de organização para imagens multibanda, geralmente acompanhados por um arquivo .hdr, descrevendo atributos específicos da imagem	Raster	Aberto
R-raster	.gri/ .grd	Formato raster nativo do raster do pacote R	Raster	Aberto
SQLite/ SpatiaLite	.sqlite	Banco de dados relacional autônomo	Vetor e raster	Aberto
ESRI FileGDB	.gdb	objetos geoespaciais e não espaciais criados pelo ArcGIS. Permite: várias classes de recursos; topologia	Vetor e raster	Proprietário
GeoPackage	.gpkg	Contêiner de banco de dados leve baseado em SQLite permitindo uma troca fácil e independente de plataforma de geodados	Vetor e raster	Aberto

O formato mais comum para arquivos vetoriais é o ESRI Shapefile; para arquivos raster é o GeoTIFF; e para dados climáticos em múltiplas camadas, geralmente há a disponibilização de dados no formato

NetCDF. Entretanto, recentemente tivemos o surgimento do GeoPackage, que possui diversas vantagens em relação aos formatos anteriores, podendo armazenar em apenas um arquivo, dados no formato vetorial, raster e também dados não-espaciais (e.g., tabelas), além de possuir uma grande integração com diversos softwares e bancos de dados.

15.6.2 Importar dados

As principais funções para importar dados no R são: i) para vetores a função `sf::st_read()`, e ii) para raster a função `raster::raster()` e suas variações `raster::brick()` e `raster::stack()` para múltiplas camadas. Essas funções atribuem objetos ao seu espaço de trabalho, armazenando-os na memória RAM disponível em seu hardware, sendo essa a maior limitação para trabalhar com dados geoespaciais no R. Por exemplo, se um arquivo raster possui mais de 8 Gb de tamanho, e seu computador possui exatamente 8 Gb de RAM, é muito provável que ele não seja importado ou mesmo criado como um objeto dentro do ambiente R. Existem soluções para esses problemas, mas não as abordaremos neste capítulo.

Vetor

Como vimos, os arquivos vetoriais são disponibilizados em diversos formatos. Para sabermos se um determinado formato pode ser importado ou exportado utilizando o pacote `sf`, podemos utilizar a função `sf::st_drivers()`. Uma amostra desses formatos é apresentado na Tabela 15.6.

```
## Formatos vetoriais importados e exportados pelo pacote sf
head(sf::st_drivers())
```

Tabela 15.6. Alguns formatos vetoriais importados e exportados pelo pacote `sf`.

name	long_name	write	copy	is_raster	is_vector	vsi
ESRIC	Esri Compact Cache	FALSE	FALSE	TRUE	TRUE	TRUE
FITS	Flexible Image Transport System	TRUE	FALSE	TRUE	TRUE	FALSE
PCIDSK	PCIDSK Database File	TRUE	FALSE	TRUE	TRUE	TRUE
netCDF	Network Common Data Format	TRUE	TRUE	TRUE	TRUE	TRUE
PDS4	NASA Planetary Data System 4	TRUE	TRUE	TRUE	TRUE	TRUE
VICAR	MIPL VICAR file	TRUE	TRUE	TRUE	TRUE	TRUE

Importar dados vetoriais existentes

Para importar vetores existentes para o R, utilizaremos a função `sf::st_read()`. A estrutura é semelhante para todos os formatos descritos na Tabela 15.6, de modo que sempre preencheremos o argumento `dsn` (*data source name*) com o nome do arquivo a ser importado. Entretanto, para banco de dados, como *GeoPackage*, pode ser necessário especificar a camada que se tem interesse com um segundo argumento chamado `layer`, com o nome da camada.

Para quase todas as operações vetoriais nesse capítulo, usaremos os dados disponíveis para o município de Rio Claro/SP. Primeiramente, baixaremos esses dadosda FBDS (Fundação Brasileira para o Desenvolvimento Sustentável), através desse repositório de dados. Em 2013, a FBDS deu início ao Projeto de Mapeamento em Alta Resolução dos Biomas Brasileiros, mapeando a cobertura da terra, hidrografia (nascentes, rios e lagos) e Áreas de Preservação Permanente (APPs). O mapeamento foi concluído para os municípios dos Biomas Mata Atlântica e Cerrado, e mais recentemente para os

outros biomas. Para fazer o download dos arquivos de interesse, utilizaremos o R, através da função `download.file()`.

Primeiramente, criaremos um diretório com a função `dir.create()`, usando a função `here::here()` para indicar o repositório (ver o Capítulo 5).

```
## Criar diretório
dir.create(here::here("dados"))
dir.create(here::here("dados", "vetor"))
```

Em seguida, vamos fazer o download de pontos de nascentes, linhas de hidrografia e polígonos de cobertura da terra para o município de Rio Claro/SP.

```
## Aumentar o tempo de download
options(timeout = 1e3)

## Download
for(i in c(".dbf", ".prj", ".shp", ".shx")){

    # Pontos de nascentes
    download.file(
        url = paste0("http://geo.fbds.org.br/SP/RIO_CLARO/HIDROGRAFIA/
SP_3543907_NASCENTES", i),
        destfile = here::here("dados", "vetor",
                              paste0("SP_3543907_NASCENTES", i)),
                              mode = "wb")

    # Linhas de hidrografia
    download.file(
        url = paste0("http://geo.fbds.org.br/SP/RIO_CLARO/HIDROGRAFIA/
SP_3543907_RIOS_SIMPLES", i),
        destfile = here::here("dados", "vetor",
                              paste0("SP_3543907_RIOS_SIMPLES", i)),
                              mode = "wb")

    # Polígonos de cobertura da terra
    download.file(
        url = paste0("http://geo.fbds.org.br/SP/RIO_CLARO/USO/SP_3543907_
USO", i),
        destfile = here::here("dados", "vetor",
                              paste0("SP_3543907_USO", i)),
                              mode = "wb")
}
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_vetor_nascentes
```

```
ecodados::geo_vetor_hidrografia
ecodados::geo_vetor_cobertura
```

Agora podemos importar esses dados para o R. Primeiro vamos importar as nascentes (Figura 15.8).

```
## Importar nascentes
geo_vetor_nascentes <- sf::st_read(
  here::here("dados", "vetor", "SP_3543907_NASCENTES.shp"), quiet = TRUE)
## Plot
plot(geo_vetor_nascentes[1], pch = 20, col = "blue", main = NA,
     axes = TRUE, graticule = TRUE)
```

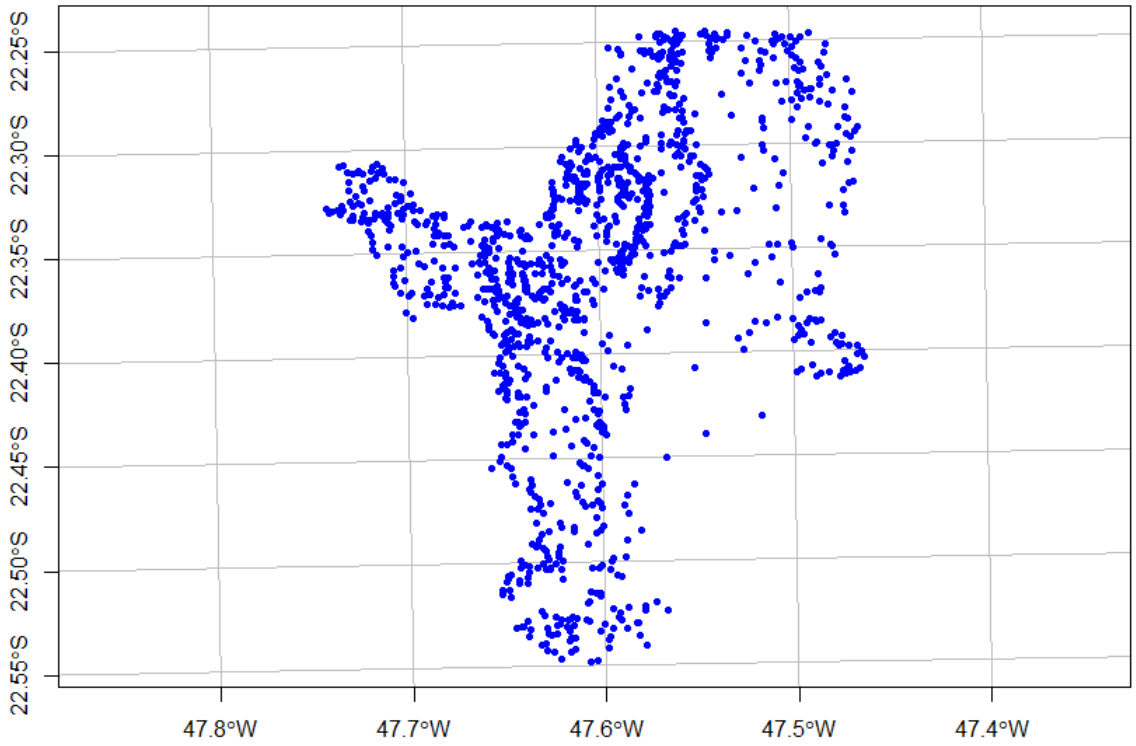


Figura 15.8: Mapa de nascentes de Rio Claro/SP.

Agora vamos importar a hidrografia (Figura 15.9).

```
## Importar hidrografia
geo_vetor_hidrografia <- sf::st_read(
    here::here("dados", "vetor", "SP_3543907_RIOS_SIMPLES.shp"),
               quiet = TRUE)
## Plot
plot(geo_vetor_hidrografia[1], col = "steelblue", main = NA,
     axes = TRUE, graticule = TRUE)
```

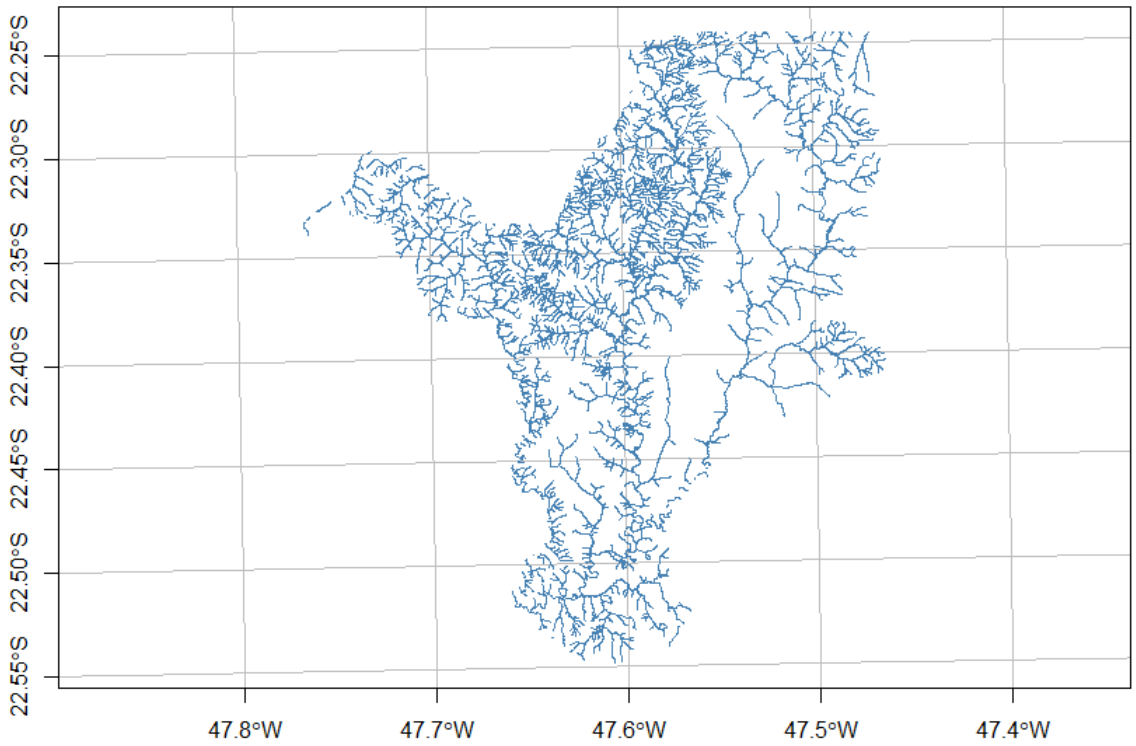


Figura 15.9: Mapa da hidrografia de Rio Claro/SP.

E por fim, vamos importar a cobertura da terra (Figura 15.10).

```
## Importar cobertura da terra
geo_vetor_cobertura <- sf::st_read(
    here::here("dados", "vetor", "SP_3543907_USO.shp"), quiet = TRUE)
## Plot
plot(geo_vetor_cobertura[5],
     col = c("blue", "orange", "gray30", "forestgreen", "green"),
     main = NA, axes = TRUE, graticule = TRUE)
legend(x = .1, y = .3, pch = 15, cex = .7, pt.cex = 2.5,
       legend = (geo_vetor_cobertura$CLASSE_USO),
       col = c("blue", "orange", "gray30", "forestgreen", "green"))
```

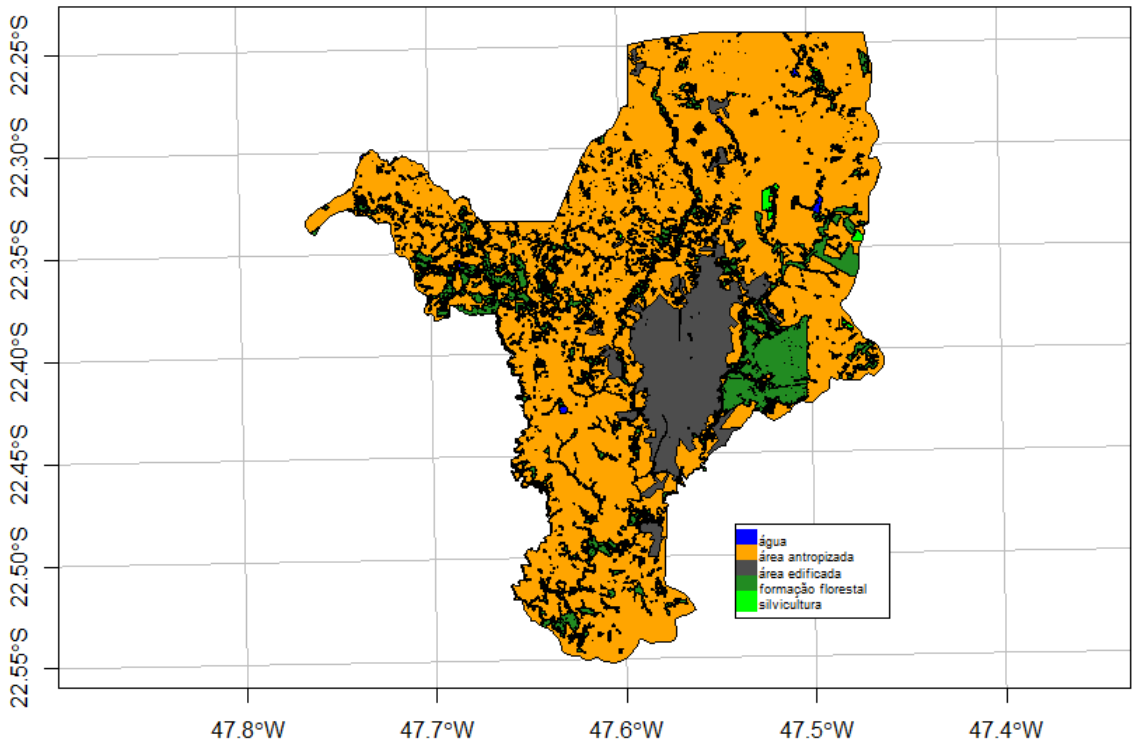


Figura 15.10: Mapa de cobertura da terra de Rio Claro/SP.

Importar utilizando pacotes

Além de dados existentes, podemos importar dados vetoriais de pacotes, como listado anteriormente na Tabela 15.4. Para o Brasil, o pacote mais interessante trata-se do `geobr`, do Instituto de Pesquisa Econômica Aplicada (IPEA), que possui dados oficiais do Instituto Brasileiro de Geografia e Estatística (IBGE).

É possível listar todos os dados disponíveis no pacote através da função `geobr::list_geobr()`. Na Tabela 15.7 é possível ver alguns desses dados.

```
## Listar todos os dados do geobr
geobr::list_geobr()
```

Tabela 15.7. Alguns dados disponíveis no pacote `geobr`

function	geography	years	source
read_country	Country	1872, 1900, 1911, 1920, 1933, 1940, 1950, 1960, 1970, 1980, 1991, 2000, 2001, 2010, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020	IBGE
read_region	Region	2000, 2001, 2010, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020	IBGE
read_state	States	1872, 1900, 1911, 1920, 1933, 1940, 1950, 1960, 1970, 1980, 1991, 2000, 2001, 2010, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020	IBGE
read_meso_region	Meso region	2000, 2001, 2010, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020	IBGE
read_micro_region	Micro region	2000, 2001, 2010, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020	IBGE

function	geography	years	source
read_intermediate_region	Intermediate region	2017, 2019, 2020	IBGE

Como exemplo, vamos fazer o download o limite do município de Rio Claro/SP, utilizando o código do município (3543907) (Figura 15.11).

> **Importante**
> Para saber todos os códigos dos municípios do Brasil, recomendamos a verificação no site do IBGE.

```
## Polígono do limite do município de Rio Claro
geo_vetor_rio_claro <- geobr::read_municipality(code_muni = 3543907,
                                                year = 2020, showProgress = FALSE)
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_vetor_rio_claro
## Plot
plot(geo_vetor_rio_claro[1], col = "gray", main = NA, axes = TRUE,
     graticule = TRUE)
```

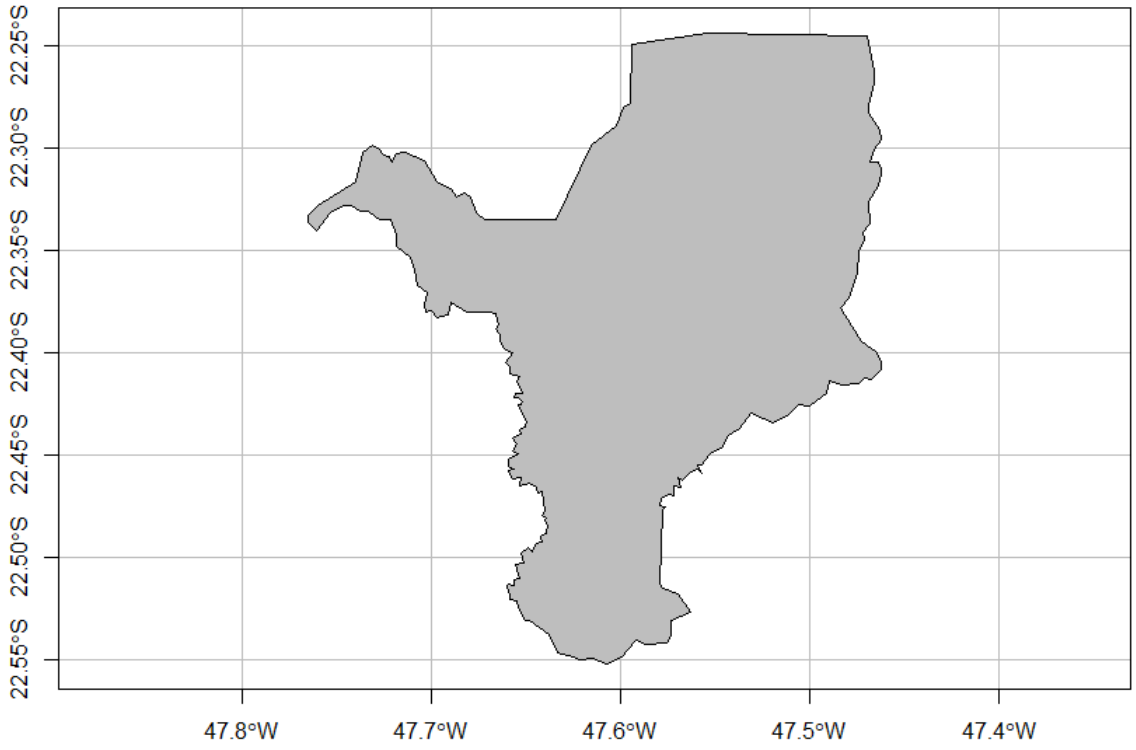


Figura 15.11: Limite do município de Rio Claro/SP.

Já para o mundo, o pacote mais interessante trata-se do `rnaturalearth`, que faz o download de dados do Natural Earth. Vamos fazer o download do limite do Brasil (Figura 15.12).

```
## Polígono do limite do Brasil
geo_vetor_brasil <- rnaturalearth::ne_countries(scale = "large",
                                    country = "Brazil", returnclass = "sf")
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_vetor_brasil
## Plot
plot(geo_vetor_brasil[1], col = "gray", main = NA, axes = TRUE,
     graticule = TRUE)
```

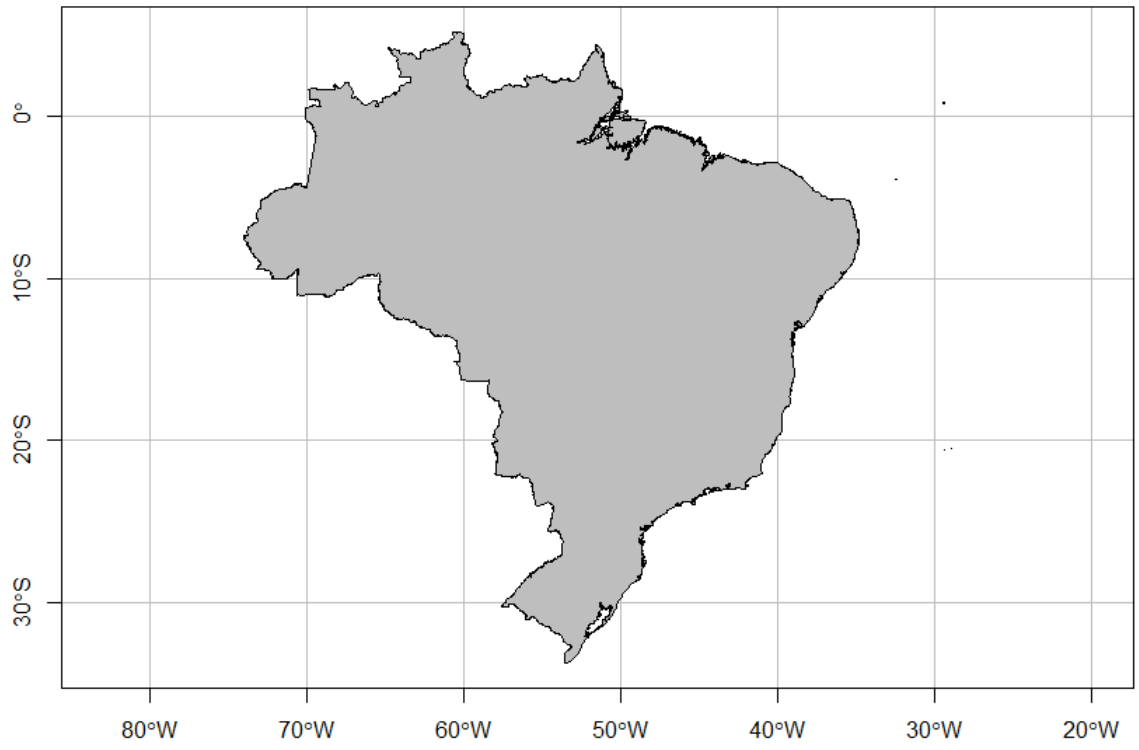


Figura 15.12: Limite do Brasil.

Criar um objeto espacial de uma tabela de coordenadas

É muito comum em coletas de campo ou bases de dados, ter coordenadas de locais de estudo ou de ocorrências de espécies organizadas em tabelas. Essas tabelas devem possuir duas colunas: longitude e latitude, ou X e Y para dados UTM, por exemplo. Ao importá-las para o R, o formato que assumem pode ser de uma das classes: `matrix`, `data frame` ou `tibble`, ou seja, ainda não são da classe vetorial `sf`. Nesta seção iremos ver como fazer essa conversão.

Para tanto, vamos usar os dados de comunidades de anfíbios da Mata Atlântica (Atlantic Amphibians, Vancine et al. (2018)). Faremos o download diretamente do site da fonte dos dados. Antes vamos criar um diretório.

```
## Criar diretório
dir.create(here::here("dados", "tabelas"))
```

Em seguida, vamos fazer o download de um arquivo `.zip` e vamos extrair usando a função `unzip()` nesse mesmo diretório.

```
## Download
download.file(url = "https://esajournals.onlinelibrary.wiley.com/action/downl
oadSupplement?doi=10.1002%2Fecy.2392&file=ecy2392-sup-0001-DataS1.zip",
              destfile = here::here("dados", "tabelas",
                                    "atlantic_amphibians.zip"),
                                     mode = "wb")

## Unzip
unzip(zipfile = here::here("dados", "tabelas",
                           "atlantic_amphibians.zip"),
      exdir = here::here("dados", "tabelas"))
```

Agora podemos importar a tabela de dados com a função `readr::read_csv()`.

```
## Importar tabela de locais
geo_anfibios_locais <- readr::read_csv(
    here::here("dados", "tabelas", "ATLANTIC_AMPHIBIANS_sites.csv"),
    locale = readr::locale(encoding = "latin1"))
geo_anfibios_locais
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_anfibios_locais
```

Por fim, podemos facilmente criar um objeto espacial do tipo `MULTIPOINT` utilizando a função `sf::st_as_sf()`. Podemos ver essas coordenadas plotadas no mapa simples da Figura 15.13 (Vancine et al. 2018).

É necessário antes se ater ao argumento `coords` que deve indicar as colunas de longitude e latitude, nessa ordem; e também ao argumento `crs` para indicar o CRS correspondente dessas coordenadas, que aqui sabemos se tratar de coordenadas geográficas e *datum* WGS84. Então podemos facilmente utilizar o código EPSG 4326. Entretanto, se as coordenadas estiverem em metros, por exemplo, teremos de nos ater a qual CRS as mesmas foram coletadas, ou seja, se forem coordenadas de GPS, é preciso saber como o GPS estava configurado (projeção e *datum*).

```
## Converter dados tabulares para sf
geo_anfibios_locais_vetor <- geo_anfibios_locais %>%
    sf::st_as_sf(coords = c("longitude", "latitude"), crs = 4326)
geo_anfibios_locais_vetor

## Plot
plot(geo_anfibios_locais_vetor[1], pch = 20, col = "black",
     main = NA, axes = TRUE, graticule = TRUE)
```

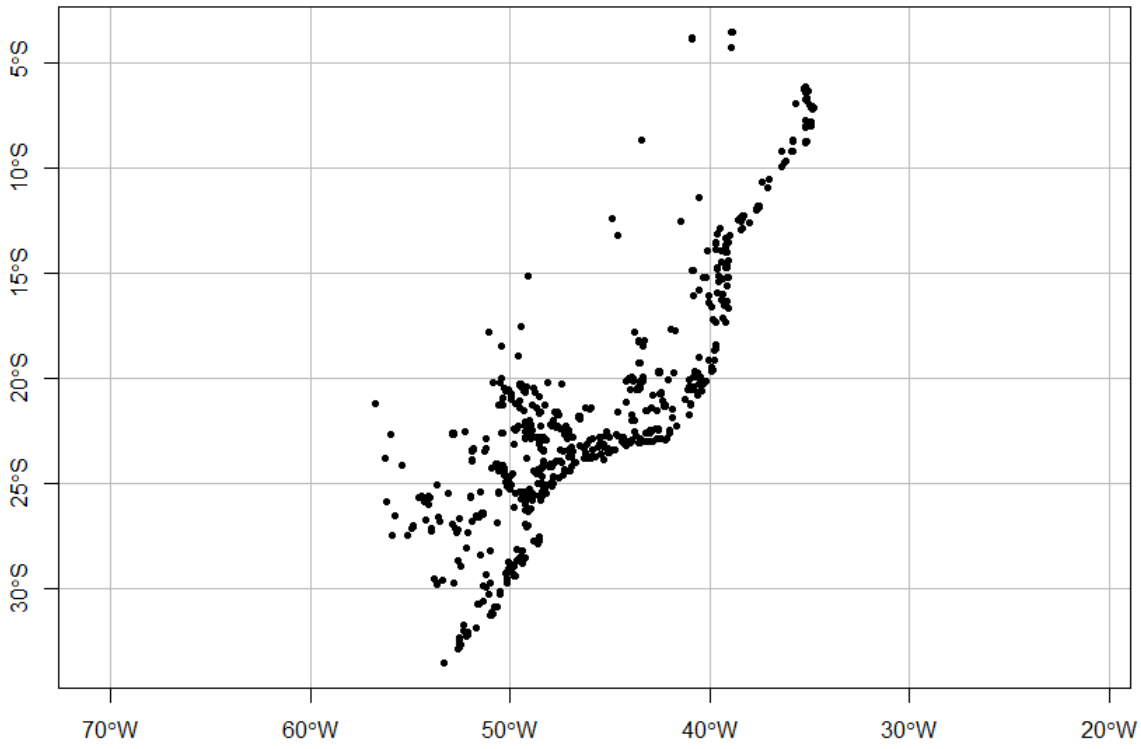


Figura 15.13: Coordenadas das comunidades do Atlantic Amphibians.

Converter dados espaciais sp para sf

O pacote `sf` é mais recente e mais fácil de manipular objetos vetoriais no R. Seu predecessor, o pacote `sp` possui uma classe própria e homônima. Entretanto, muitos pacotes de análises espaciais ainda utilizam essa classe em suas funções, apesar dessa migração ter ocorrido rapidamente recentemente. Dessa forma, a conversão entre essas classes pode ser necessária em alguns momentos.

Abaixo, veremos como podemos fazer essa conversão facilmente. Primeiramente, vamos importar dados `sp`.

```
## Polígonos países sp
co110_sp <- rnaturalearth::countries110
class(co110_sp)
#> [1] "SpatialPolygonsDataFrame"
#> attr(,"package")
#> [1] "sp"
```

Agora, podemos converter facilmente com a função `sf::st_as_sf()`.

```
## Polígonos países sf
co110_sf <- sf::st_as_sf(co110_sp)
class(co110_sf)
#> [1] "sf"         "data.frame"
```

Podemos facilmente converter esse objeto novamente para a classe `sp` com a função `sf::as_Spatial()`.

```
## Polígonos países sp
co110_sp <- sf::as_Spatial(co110_sf)
```

```
class(co110_sp)
#> [1] "SpatialPolygonsDataFrame"
#> attr(,"package")
#> [1] "sp"
```

Raster

Para importar dados raster no R, utilizaremos a função `raster::raster()`, `raster::brick()` ou `raster::stack()`. Para apenas uma camada raster, usaremos a função `raster::raster()`, com o argumento `x` sendo o nome do arquivo. Já para mais camadas, usaremos `raster::brick()` para um arquivo que possua múltiplas camadas, ou ainda a função `raster::stack()` para vários arquivos em diferentes camadas também no argumento `x`, sendo necessário listar os arquivos no diretório, geralmente utilizando a função `dir()` ou `list.files()`. Entretanto, para especificar uma camada, podemos utilizar o argumento `band` ou `layer` e o nome dessa camada.

Raster Layer

Primeiramente, vamos criar um diretório para os dados raster que fazeremos o download.

```
## Criar diretório
dir.create(here::here("dados", "raster"))
```

Em seguida, vamos fazer o download de dados de elevação, na verdade dados de Modelo Digital de Elevação (*Digital Elevation Model* - DEM), localizados também para o município de Rio Claro. Utilizaremos os dados do Shuttle Radar Topography Mission - SRTM. Para saber mais sobre esses dados, recomendamos a leitura do artigo de Farr et al. (2007).

```
## Aumentar o tempo de download
options(timeout = 1e3)

## Download
download.file(url = "https://srtm.csi.cgiar.org/wp-content/uploads/files/
srtm_5x5/TIFF/srtm_27_17.zip",
              destfile = here::here("dados", "raster", "srtm_27_17.zip"),
mode = "wb")

## Unzip
unzip(zipfile = here::here("dados", "raster", "srtm_27_17.zip"),
      exdir = here::here("dados", "raster"))
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_raster_srtm
```

Agora podemos importar essa camada para o R, e visualizá-la em relação ao limite do município de Rio Claro/SP (Figura 15.14).

```
## Importar raster de altitude
geo_raster_srtm <- raster::raster(here::here("dados", "raster", "srtm_27_17.
tif"))
```

```
geo_raster_srtm
#> class       : RasterLayer
#> dimensions  : 6000, 6000, 3.6e+07  (nrow, ncol, ncell)
#> resolution  : 0.0008333333, 0.0008333333  (x, y)
#> extent      : -50, -45, -25, -20  (xmin, xmax, ymin, ymax)
#> crs         : +proj=longlat +datum=WGS84 +no_defs
#> source      : srtm_27_17.tif
#> names       : srtm_27_17
#> values      : -32768, 32767  (min, max)
## Plot
plot(geo_raster_srtm, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

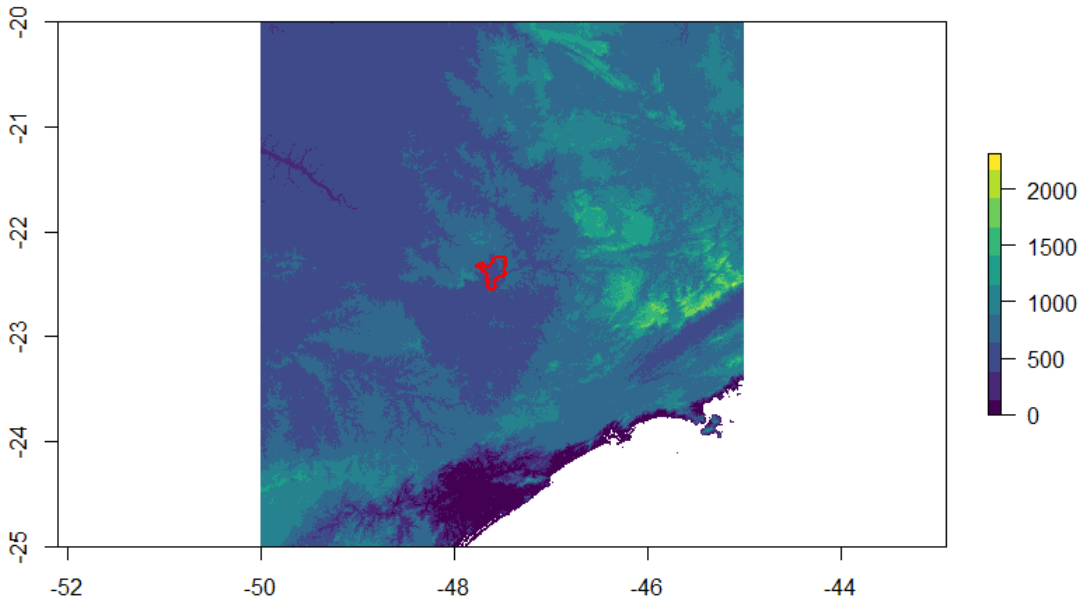


Figura 15.14: Camada raster do DEM em relação ao limite do município de Rio Claro/SP.

Raster Stack

Além dos dados de elevação, dados de temperatura e precipitação podem ser obtidos do WorldClim. Para saber mais sobre esses dados, recomendamos a leitura do artigo Fick & Hijmans (2017).

```
## Aumentar o tempo de download
options(timeout = 1e3)

## Download
download.file(url = "https://biogeo.ucdavis.edu/data/worldclim/v2.1/base/
wc2.1_10m_bio.zip",
              destfile = here::here("dados", "raster",
                                    "wc2.0_10m_bio.zip"), mode = "wb")

## Unzip
```

```
unzip(zipfile = here::here("dados", "raster", "wc2.0_10m_bio.zip"),
      exdir = here::here("dados", "raster"))
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_raster_bioclim
```

Para importar essa série de camadas, primeiramente listaremos os arquivos e depois importaremos no formato `RasterStack` (Figura 15.15).

```
## Listar arquivos
arquivos_raster <- dir(path = here::here("dados", "raster"),
                       pattern = "wc") %>%
                       grep(".tif", ., value = TRUE)
arquivos_raster

## Importar vários rasters como stack
geo_raster_bioclim <- raster::stack(here::here("dados", "raster",
                                               arquivos_raster))
geo_raster_bioclim

## Plot
plot(geo_raster_bioclim[[c(1, 4)]], col = viridis::viridis(10))
```

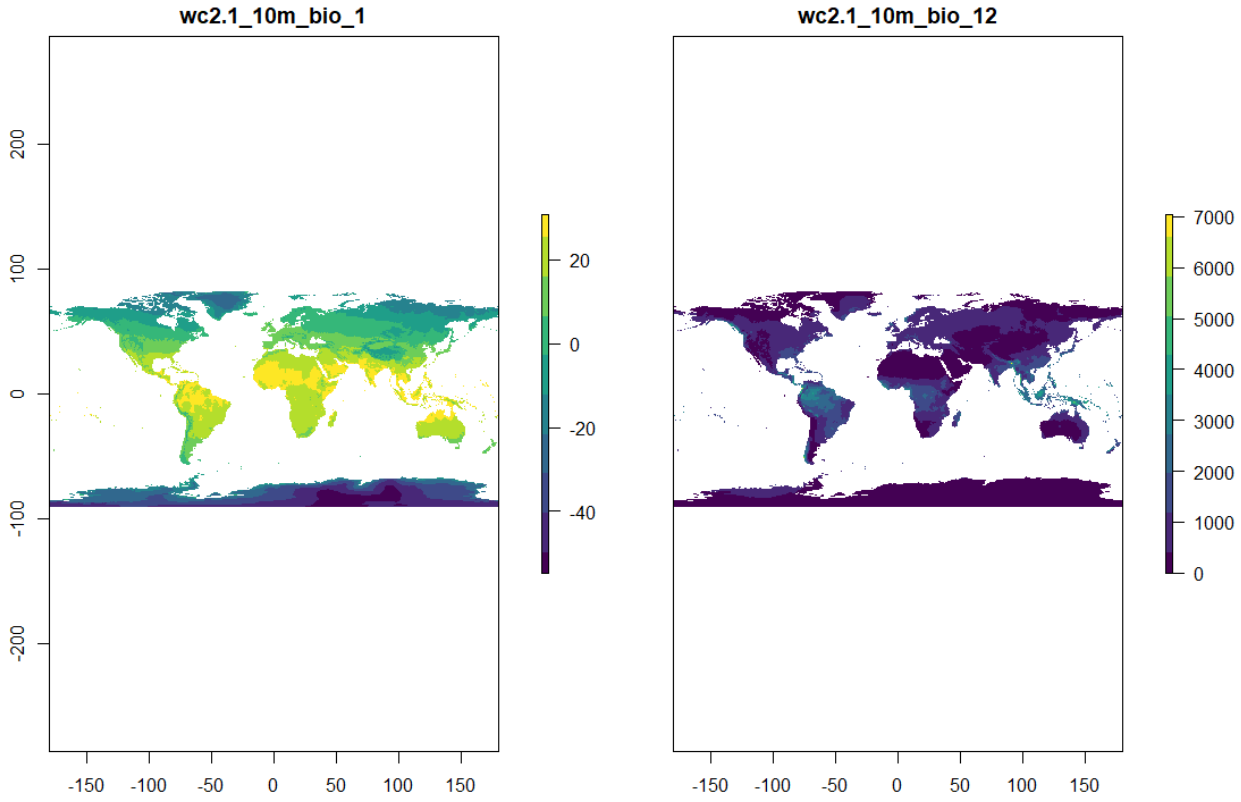


Figura 15.15: Camadas rasters do WorldClim (BIO01 e BIO12) para o mundo.

15.6.3 Exportar dados

Saber a melhor forma de exportar dados geoespaciais de objetos recém-criados no R é fundamental, principalmente porque essa ação dependerá do tipo de dado (vetor ou raster), classe do objeto (por exemplo, `MULTIPOINT` ou `RasterLayer`) e tipo e quantidade de informações armazenadas (por exemplo, tamanho do objeto, intervalo de valores, etc.).

Vetor

Para dados vetoriais, a principal função utilizada é a `sf::st_write()`. Essa função permite gravar objetos `sf` em vários formatos de arquivos vetoriais, como `.shp`, `.gpkg` ou `.geojson`. O formato a ser exportado vai influenciar na velocidade do processo de gravação.

Os argumentos dessa função será o `obj` que é o objeto `sf` criado no ambiente R, e o `dsn` (*data source name*), ou seja, o nome que o arquivo terá ao ser exportado do R, de modo que o complemento `.shp` no nome de saída, por exemplo, definirá que o arquivo terá a extensão `ESRI Shapefile`. Entretanto, essa extensão pode ser definida também utilizando o argumento `driver`, com as possibilidades listadas nesse site.

```
## Exportar o polígono de Rio Claro na extensão ESRI Shapefile
sf::st_write(obj = geo_vetor_rio_claro,
             dsn = here::here("dados", "vetor", "geo_vetor_rio_claro.shp"))
```

Ou podemos ainda exportar o objeto vetorial na extensão `GeoPackage`. Entretanto, aqui é interessante acrescentar um argumento chamado `layer` para definir o nome das camadas a serem exportadas no mesmo arquivo `GeoPackage`, por exemplo.

```
## Exportar o polígono de Rio Claro na extensão Geopackage
sf::st_write(obj = geo_vetor_rio_claro,
             dsn = here::here("dados", "vetor", "vetores.gpkg"),
             layer = "rio_claro")
```

Ainda sobre o formato `GeoPackage`, há algo muito interessante que podemos fazer: podemos acrescentar outros arquivos vetoriais ao mesmo arquivo já criado. Como exemplo, exportaremos o limite do Brasil para o mesmo arquivo.

```
## Exportar o polígono do Brasil na extensão Geopackage
sf::st_write(obj = geo_vetor_brasil,
             dsn = here::here("dados", "vetor", "vetores.gpkg"),
             layer = "brasil")
```

Raster

Para exportar dados raster utilizamos geralmente a função `raster::writeRaster()`. Exportar dados raster é um pouco mais complexo que exportar dados vetoriais. Teremos de definir se exportaremos arquivos em uma ou várias camadas, quantidade de informações por pixel, e ainda diferentes extensões de saída.

> **Importante**
> Arquivos raster escritos em discos geralmente ocupam bastante espaço, e dessa forma, há parâmetros específicos para certos tipos de dados, que detalharemos a seguir para contornar esse problema e comprimir os arquivos.

Na função `raster::writeRaster()`, o argumento `x` diz respeito ao objeto raster no ambiente R. O argumento `filename` é nome do arquivo que será exportado do R, podendo ou não possuir a extensão que se pretende que o arquivo tenha. O argumento `format` é o formato do arquivo, sendo as principais possibilidades resumidas na Tabela 15.8, e para saber das possibilidades suportadas, use a função `raster::writeFormats()`. O argumento `bylayer` diz se múltiplas camadas serão exportadas em arquivos diferentes ou em apenas um arquivo.

Tabela 15.8. Principais formatos de arquivos raster exportados do R.

Tipo de arquivo	Nome longo	Extensão	Suporte a múltiplas camadas
raster	Formato pacote raster	.grd	Sim
ascii	ESRI Ascii	.asc	Não
SAGA	SAGA GIS	.sdat	Não
IDRISI	IDRISI	.rst	Não
CDF	netCDF (requer ncdf4)	.nc	Sim
GTiff	GeoTiff (requer rgdal)	.tif	Sim
ENVI	ENVI .hdr	.envi	Sim
EHdr	ESRI .hdr	.bil	Sim
HFA	Erdas imagem (.img)	.img	Sim

Dentre os argumentos adicionais, temos ainda o `datatype`, que faz referência a um dos nove tipos de formato de dados detalhados na Tabela 15.9, sendo que o tipo de dado determina a representação em bits (quantidade de informação) na célula do objeto raster exportado e depende da faixa de valores do objeto raster em cada pixel. Quanto mais valores um tipo de dado puder representar, maior será o arquivo exportado no disco. Dessa forma, é interessante utilizar um tipo de dado que diminua o tamanho do arquivo a ser exportado, dependendo do tipo de dado em cada pixel. Para a função `raster::writeRaster()`, o default é `FLT4S`, o que pode ocupar mais espaço em disco do que o necessário.

Tabela 15.9. Tipos de dados suportados pelo pacote `raster`

Tipo de dado	Valor mínimo	Valor máximo
LOG1S	FALSE (0)	TRUE (1)
INT1S	-127	127
INT1U	0	255
INT2S	-32.767	32.767
INT2U	0	65534

Tipo de dado	Valor mínimo	Valor máximo
INT4S	-2.147.483.647	2.147.483.647
INT4U	0	42.94.967.296
FLT4S	-3,4e+38	3,4e+38
FLT8S	-1,7e+308	1,7e+308

Outros argumentos de suporte são: `overwrite` para sobrescrever um arquivo que já exista, `progress` para mostrar uma barra de progresso da exportação como "text" ou "window," e `options` que permite opções do GDAL. Para esse último, quando exportar especificamente na extensão `GeoTIFF`, podemos utilizar `options = c("COMPRESS=DEFLATE", "TFW=YES")` para que haja compressão do arquivo, diminuindo consideravelmente seu tamanho (cerca de um terço), aliado à criação de um arquivo auxiliar `.tfw`, para ser carregado em softwares específicos de SIG, como o ArcGIS.

Para exportar apenas uma camada `RasterLayer`, podemos utilizar a função `raster::writeRaster()` em um formato mais simples.

```
## Criar diretório
dir.create(here::here("dados", "raster", "exportados"))

## Exportar raster layer
raster::writeRaster(geo_raster_srtm,
                    filename = here::here("dados", "raster",
                                          "exportados", "elevation"),
                    format = "GTiff",
                    datatype = "INT2S",
                    options = c("COMPRESS=DEFLATE", "TFW=YES"),
                    progress = "text",
                    overwrite = TRUE)
```

Para mais de uma camada `RasterBrick` ou `RasterStack`, podemos utilizar a função `raster::writeRaster()` com o `bylayer = TRUE`.

```
## Exportar raster stack
raster::writeRaster(x = geo_raster_bioclim,
                    filename = here::here("dados", "raster",
                                          "exportados",
                                          names(geo_raster_bioclim)),
                    bylayer = TRUE,
                    format = "GTiff",
                    datatype = "INT2S",
                    options = c("COMPRESS=DEFLATE", "TFW=YES"),
                    progress = "text",
                    overwrite = TRUE)
```

15.7 Descrição de objetos geoespaciais

Muitas vezes precisaremos verificar as informações dos objetos geoespaciais importados para o R. Apesar de chamar o objeto trazer grande parte das informações que precisamos consultar, existem funções específicas que nos auxiliam nesse processo de descrição dos objetos.

15.7.1 Vetor

Podemos acessar as informações geoespaciais e a tabela de atributos de um objeto importado como vetor simplesmente chamando o nome do objeto no R.

```
## Município de Rio Claro
geo_vetor_rio_claro
```

Mas também podemos acessar informações geoespaciais com funções específicas, como tipo de geometria, limites geoespaciais do vetor (extensão), sistema de referência de coordenadas (CRS), e a tabela de atributos.

```
## Tipo de geometria
sf::st_geometry_type(geo_vetor_rio_claro)

## Extensão
sf::st_bbox(geo_vetor_rio_claro)
#>      xmin      ymin      xmax      ymax
#> -47.76521 -22.55203 -47.46188 -22.24368

## CRS
sf::st_crs(geo_vetor_rio_claro)

## Acessar a tabela de atributos
geo_vetor_rio_claro_tab <- sf::st_drop_geometry(geo_vetor_rio_claro)
geo_vetor_rio_claro_tab
```

15.7.2 Raster

Da mesma forma, podemos acessar as informações objetos raster chamando o nome do objeto.

```
## Raster layer
geo_raster_srtm
#> class      : RasterLayer
#> dimensions : 6000, 6000, 3.6e+07  (nrow, ncol, ncell)
#> resolution : 0.0008333333, 0.0008333333  (x, y)
#> extent     : -50, -45, -25, -20  (xmin, xmax, ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : srtm_27_17.tif
#> names      : srtm_27_17
#> values     : -32768, 32767  (min, max)
```

Além disso, podemos selecionar informações desse objeto com funções específicas, tanto para `RasterLayer`, quanto para `RasterBrick` ou `RasterStack` como: classe, dimensões (número de linhas, colunas e camadas), número de camadas, número de linhas, número de colunas, número de células, resolução (largura e altura do tamanho do pixel), extensão (limites geoespaciais), sistema de referência de coordenadas (CRS), nome das camadas e extrair os valores de todos os pixels.

```
## Classe
class(geo_raster_srtm)
#> [1] "RasterLayer"
#> attr(,"package")
#> [1] "raster"

## Dimensões
dim(geo_raster_srtm)
#> [1] 6000 6000    1

## Número de camadas
nlayers(geo_raster_srtm)
#> [1] 1

## Número de linhas
nrow(geo_raster_srtm)
#> [1] 6000

## Número de colunas
ncol(geo_raster_srtm)
#> [1] 6000

## Número de células
ncell(geo_raster_srtm)
#> [1] 3.6e+07

## Resolução
res(geo_raster_srtm)
#> [1] 0.0008333333 0.0008333333

## Extensão
extent(geo_raster_srtm)
#> class      : Extent
#> xmin       : -50
#> xmax       : -45
#> ymin       : -25
#> ymax       : -20

## Projeção ou CRS
projection(geo_raster_srtm)
#> [1] "+proj=longlat +datum=WGS84 +no_defs"
```

```
## Nomes
names(geo_raster_srtm)
#> [1] "srtm_27_17"

## Valores
```

```
getValues(geo_raster_srtm) %>% head
#> [1] 382 379 379 379 379 383
values(geo_raster_srtm) %>% head
#> [1] 382 379 379 379 379 383
geo_raster_srtm[] %>% head
#> [1] 382 379 379 379 379 383
```

15.8 Reprojeção de dados geoespaciais

Em algumas situações é necessário alterar o CRS de um objeto espacial para um novo CRS. A reprojeção é justamente a transformação de coordenadas de um CRS para outro: geoespaciais ('lon/lat,' com unidades em graus de longitude e latitude) e projetados (normalmente com unidades de metros a partir de um *datum*).

Geralmente precisaremos fazer essa operação para transformar camadas vetoriais ou rasters para o mesmo CRS, de modo que possam ser exibidas conjuntamente, ou ainda que as camadas possuem CRS projetado para realizar alguma operação espacial entre camadas, ou quando precisamos calcular áreas, formatos ou distâncias, como métricas de paisagem, por exemplo. Existe uma infinidade de projeções e um excelente material de consulta é o livro de Lapaine & Usery (2017).

Podemos verificar o CRS de uma camada através da função `sf::st_crs()` ou `raster::projection()` e `raster::crs()`, ou ainda, saber se a mesma possui um CRS geográfico ou não, com a função `sf::st_is_longlat()`.

Já para reprojetar um objeto `sf` usamos a função `sf::st_transform()` e para um objeto `raster` usamos a função `raster::projectRaster()`.

```
## Projeção de vetores
sf::st_crs(geo_vetor_rio_claro)
## Projeção de raster
raster::projection(geo_raster_srtm)
raster::crs(geo_raster_srtm)

## Verificar se o CRS é geográfico
sf::st_is_longlat(geo_vetor_rio_claro)
#> [1] TRUE
```

As funções `sf::st_transform()` e `raster::projectRaster()` possuem dois argumentos importantes: `x` que é o objeto a ser reprojetado e o `crs` que é o CRS alvo. O argumento `crs` pode ser especificado de quatro maneiras: i) código EPSG (por exemplo, 4326), ii) string PROJ4 (por exemplo, `+proj = longlat + datum = WGS84 + no_defs`), iii) string WKT, ou iv) objeto `crs` de outra camada,

conforme retornado por `sf::st_crs()` ou `raster::crs()`. Essas informações de EPSG, PROJ4 e WKT podem ser acessadas nas bases: epsg.io e spatialreference.org.

Dentre os possíveis CRSs a serem utilizados, alguns são mais comuns para CRSs geoespaciais e projetados. Para CRSs geoespaciais, o mais comum para o mundo é o *World Geodetic System 1984* (WGS84), ou seja, geográfico com *datum* WGS84. Para o Brasil, o CRS adotado é o Sistema de Referencia Geocéntrico para las Américas 2000 (SIRGAS 2000), ou seja, geográfico com *datum* SIRGAS2000.

Para CRSs projetados, essa escolha vai depender da extensão e localização da área de interesse no globo terrestre. Aqui destacaremos os principais, para três escalas: global, regional e local. Para a escala global, geralmente usa-se umas dessas projeções, dependendo do objetivo: i) Projeção de Mollweide, ii) Projeção de Winkel Tripel, iii) Projeção de Eckert IV, iv) Projeção Azimutal de Lambert. Para a escala regional, como um hemisfério, geralmente usa-se a Projeção Cônica de Albers. Por fim, para a escala local, usa-se geralmente a Projeção Universal Transverse Mercator (UTM), um conjunto de CRSs que divide a Terra em 60 cunhas longitudinais e 20 segmentos latitudinais, como pode ser visto neste link.

Os principais CRSs são descr itos na Tabela 15.10.

Tabela 15.10. Principais CRSs utilizados.

CRS	Tipo de CRS	Descrição	epsg.io	spatialreference.org
World Geodetic System 1984 (WGS84)	Geográfico	CRS geográfico mais comum para o mundo	EPSG:4326	EPSG:4326
Sistema de Referencia Geocéntrico para las Américas 2000 (SIRGAS 2000)	Geográfico	CRS geográfico oficial para o Brasil	EPSG:4674	EPSG:4674
Projeção de Mollweide	Projetado	CRS projetado que preserva as relações de área	ESRI:54009	SR-ORG:7099
Projeção de Winkel Tripel	Projetado	CRS projetado com mínimo de distorção para área, direção e distância	NA	SR-ORG:7291
Projeção de Eckert IV	Projetado	CRS projetado que preserva a área e com meridianos elípticos	EPSG:54012	ESRI:54012
Projeção Azimutal de Lambert	Projetado	CRS projetado que preserva os tamanhos relativos e senso de direção a partir do centro	NA	NA
Projeção Cônica de Albers	Projetado	CRS projetado para escala regional, mantendo a área constante em toda sua superfície	NA	SR-ORG:7823
Projeção Universal Transverse Mercator (UTM)	Projetado	CRS projetado para escala local, distorcendo áreas e distâncias com gravidade crescente com a distância do centro da zona UTM	EPSG:31983	EPSG:31983

15.8.1 Vetor

Como dissemos, para reprojetar um vetor, utilizamos a função `sf::st_transform()`, observando os argumentos `x` que é a camada a ser reprojetada, e o `crs` que é o CRS alvo.

Vamos reprojetar o limite do município de Rio Claro/SP do CRS SIRGAS2000/geográfico para o CRS projetado SIRGAS2000/UTM23S, com os efeitos da transformação podendo ser notados na Figura 15.16.

```
## Converter CRS
geo_vetor_rio_claro_sirgas2000_utm23s <- sf::st_transform(
                                    x = geo_vetor_rio_claro, crs = 31983)
```

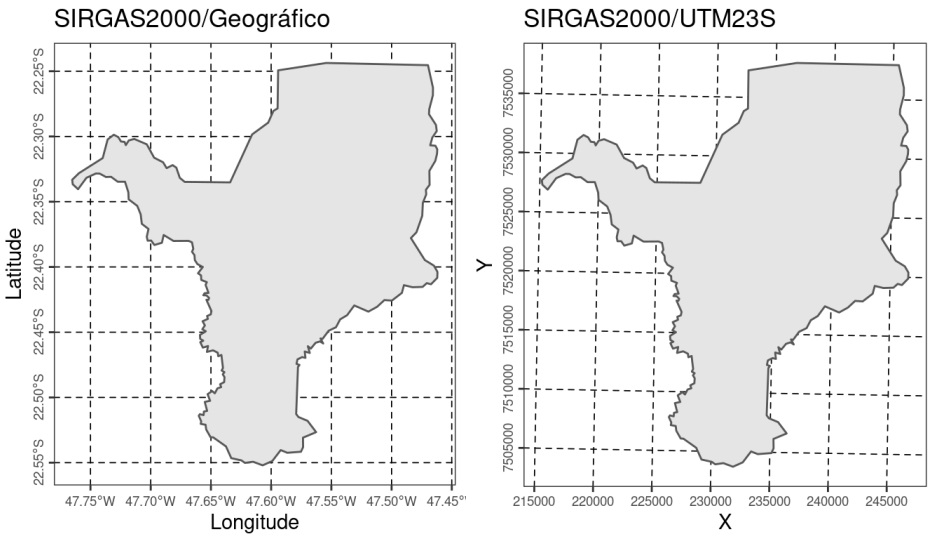


Figura 15.16: Limites do município de Rio Claro/SP com CRS SIRGAS2000/ geográfico e com CRS SIRGAS2000/UTM23S.

Podemos ainda utilizar o formato `proj4string` no argumento `crs` para fazer a transformação. Vamos primeiramente plotar o mundo em WGS84/Geográfico (Figura 15.17).

```
## Plot
plot(co110_sf[1], col = "gray",  main = "WGS84/Geográfio", graticule = TRUE)
```

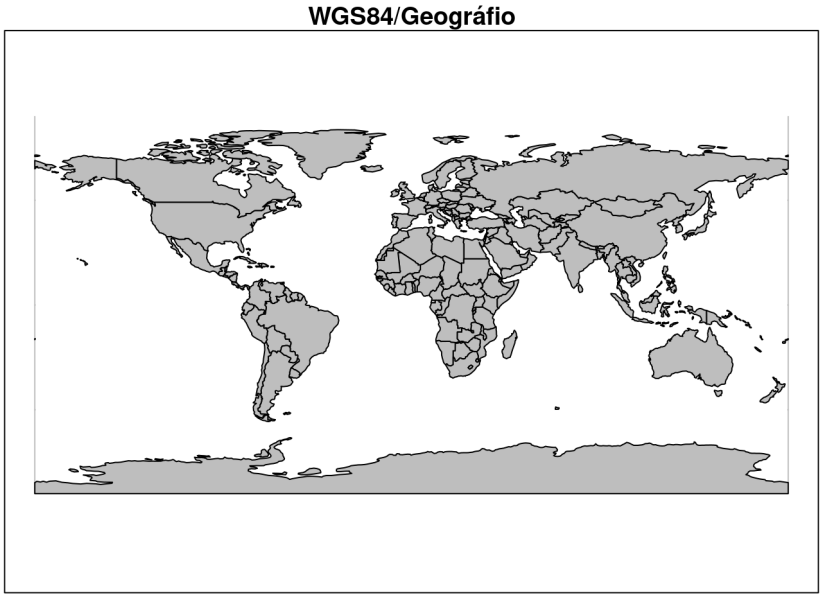


Figura 15.17: Limite dos países do mundo com CRS geográfico e datum WGS84.

Agora, reprojetaremos utilizando a Projeção de Mollweide (Figura 15.18).

```
## Projeção de Mollweide
co110_sf_moll <- sf::st_transform(x = co110_sf, crs = "+proj=moll")
## Plot
plot(co110_sf_moll[1], col = "gray", main = "Projeção de Mollweide",
     graticule = TRUE)
```

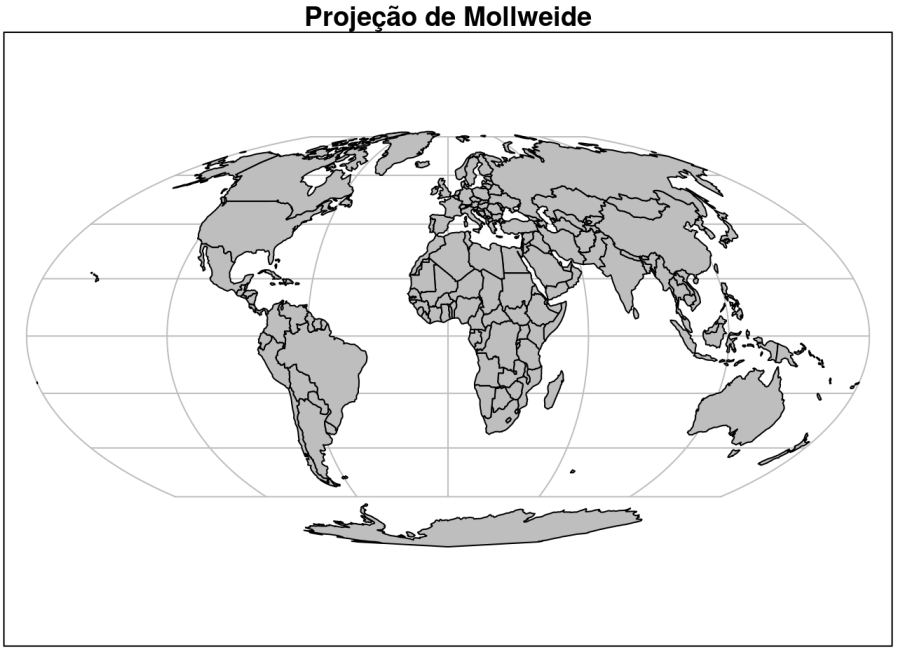


Figura 15.18: Limite dos países do mundo com CRS Projeção de Mollweide.

Ou ainda podemos utilizar a Projeção Azimutal de Lambert com alguns parâmetros ajustados para centralizar a projeção no Brasil (15.19).

```
## Projeção Azimutal de Lambert
co110_sf_laea <- sf::st_transform(x = co110_sf,
                    crs = "+proj=laea +x_0=0 +y_0=0 +lon_0=-50 +lat_0=0")
## Plot
plot(co110_sf_laea[1], col = "gray",
      main = "Projeção Azimutal de Lambert", graticule = TRUE)
```

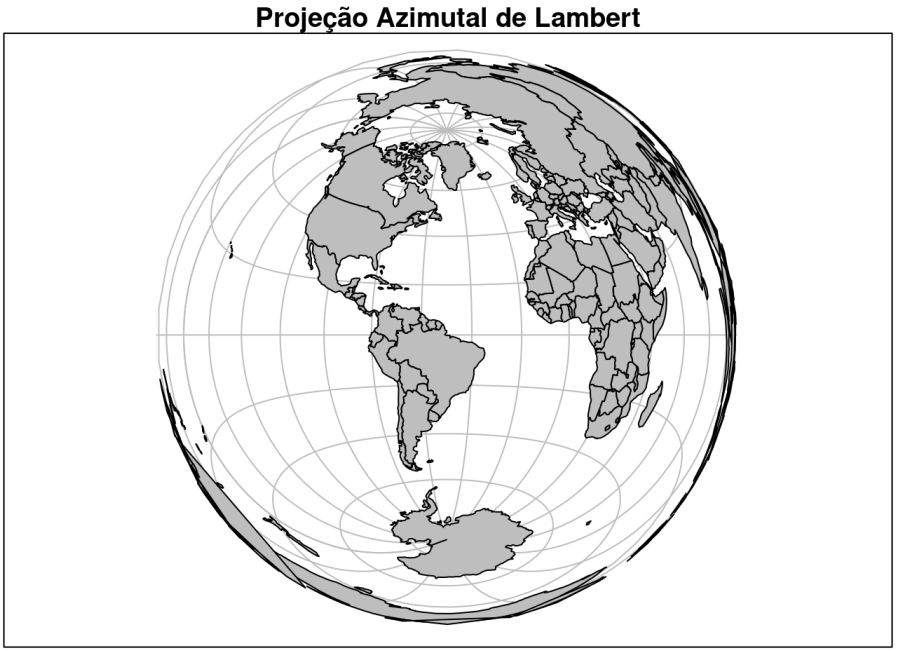


Figura 15.19: Limite dos países do mundo com CRS Projeção Azimutal de Lambert centrado no Brasil.

15.8.2 Raster

A reprojeção de objetos raster não é uma tarefa tão simples quanto a reprojeção de vetores. Em vetores, a reprojeção altera as coordenadas de cada vértice. Entretanto, como rasters são compostos de células retangulares do mesmo tamanho, a reprojeção do raster envolve a criação de um novo objeto raster, com duas operações espaciais separadas: i) reprojeção vetorial dos centroides celulares para outro CRS (i.e., muda a posição e tamanho do pixel) e, ii) cálculo de novos valores do pixel por meio de reamostragem (i.e., muda o valor do pixel).

A função `raster::projectRaster()` possui alguns parâmetros que necessitam de algumas especificações. O argumento `from` é o objeto raster de entrada que sofre a reprojeção. O argumento `to` é um objeto raster do qual todas as propriedades dos CRSs, como extensão e resolução serão associadas ao objeto raster indicado no argumento `from`. O argumento `res` permite ajustar a resolução do pixel de saída do objeto raster reprojetado.

O argumento `crs` aceita apenas as definições de `proj4string` extensas de um CRS em vez de códigos EPSG concisos. Contudo, é possível usar um código `EPSG` em uma definição de `proj4string` com `+init=epsg:EPSG`. Por exemplo, pode-se usar a definição `+init=epsg:4326` para definir CRS para WGS84 (código EPSG de 4326). A biblioteca `PROJ` adiciona automaticamente o resto dos parâmetros e os converte em `+init=epsg:4326 +proj=longlat +datum=WGS84 + no_defs + ellps=WGS84 + towgs84=0,0,0`.

O argumento `method` permite escolher entre os métodos `ngb` (vizinho mais próximo) ou `biliniar` (interpolação bilinear), sendo o primeiro mais indicado para reprojeção de rasters categóricos, pois os valores estimados devem ser iguais aos do raster original. O método `ngb` define cada novo valor de célula para o valor da célula mais próxima (centro) do raster de entrada. Já o método `biliniar` é indicado para raster contínuos e calcula o valor da célula de saída com base nas quatro células mais próximas no raster original, sendo a média ponderada da distância dos valores dessas quatro células. Existem outras formas de interpolação, mas não as abordaremos aqui.

Aqui, vamos reprojetar os dados de elevação para Rio Claro/SP. Para que esse processo seja mais rápido, iremos antes ajustar a extensão do raster para o limite do município usando a função `raster::crop()` (Figura 15.20). Essa função é melhor explicada na seção de cortes e máscaras, mais adiante.

```
## Ajuste do limite
geo_raster_srtm_rio_claro <- raster::crop(x = geo_raster_srtm,
                                          y = geo_vetor_rio_claro)
geo_raster_srtm_rio_claro
#> class      : RasterLayer
#> dimensions : 370, 364, 134680  (nrow, ncol, ncell)
#> resolution : 0.0008333333, 0.0008333333  (x, y)
#> extent     : -47.765, -47.46167, -22.55167, -22.24333  (xmin, xmax, ymin,
ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : srtm_27_17
#> values     : 491, 985  (min, max)
## Plot
plot(geo_raster_srtm_rio_claro, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

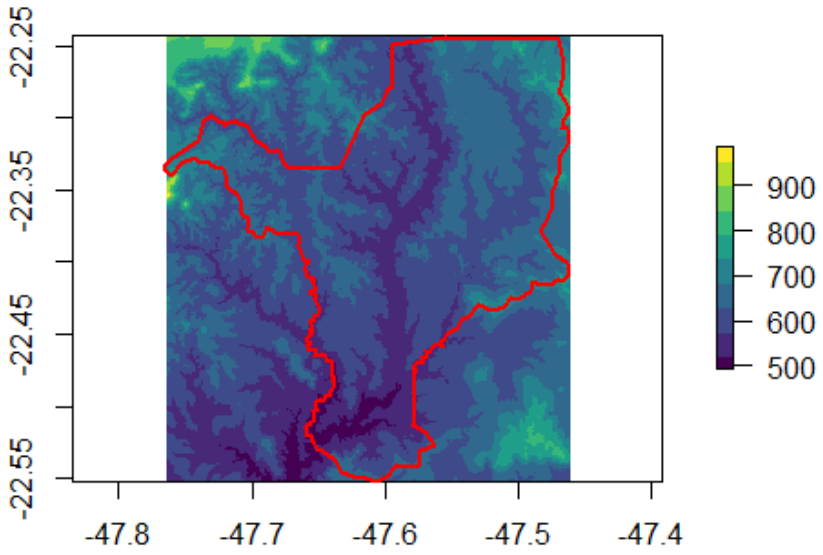


Figura 15.20: Ajuste da extensão do raster de elevação para o município de Rio Claro/SP.

Primeiramente, vamos reprojetar indicando uma projeção e sem especificar o tamanho da célula. Note que o tamanho da célula vai se ajustar para valores diferentes sendo, portanto, pixels retangulares e não quadrados.

```
## Reprojeção
geo_raster_srtm_rio_claro_sirgas2000_utm23s <- raster::projectRaster(
    from = geo_raster_srtm_rio_claro,
    crs = "+init=epsg:31983",
    method = "bilinear")
geo_raster_srtm_rio_claro_sirgas2000_utm23s
#> class      : RasterLayer
#> dimensions : 386, 381, 147066  (nrow, ncol, ncell)
#> resolution : 85.8, 92.3  (x, y)
#> extent     : 214575.4, 247265.2, 7503009, 7538637  (xmin, xmax, ymin,
ymax)
#> crs        : +proj=utm +zone=23 +south +ellps=GRS80 +units=m +no_defs
#> source     : memory
#> names      : srtm_27_17
#> values     : 491.6033, 980.4151  (min, max)
```

Agora vamos reprojetar especificando o tamanho da célula (Figura 15.21). Dessa forma, todas as células terão o mesmo, i.e., quadrados de 90 metros.

```
## Reprojeção
geo_raster_srtm_rio_claro_sirgas2000_utm23s <- raster::projectRaster(
    from = geo_raster_srtm_rio_claro,
    crs = "+init=epsg:31983",
    method = "bilinear",
    res = 90)
geo_raster_srtm_rio_claro_sirgas2000_utm23s
#> class      : RasterLayer
#> dimensions : 396, 364, 144144  (nrow, ncol, ncell)
#> resolution : 90, 90  (x, y)
#> extent     : 214554.4, 247314.4, 7502985, 7538625  (xmin, xmax, ymin,
ymax)
#> crs        : +proj=utm +zone=23 +south +ellps=GRS80 +units=m +no_defs
#> source     : memory
#> names      : srtm_27_17
#> values     : 493.2395, 986.686  (min, max)
## Plot
plot(geo_raster_srtm_rio_claro_sirgas2000_utm23s,
     col = viridis::viridis(10))
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom,
     col = NA, border = "red", lwd = 2, add = TRUE)
```

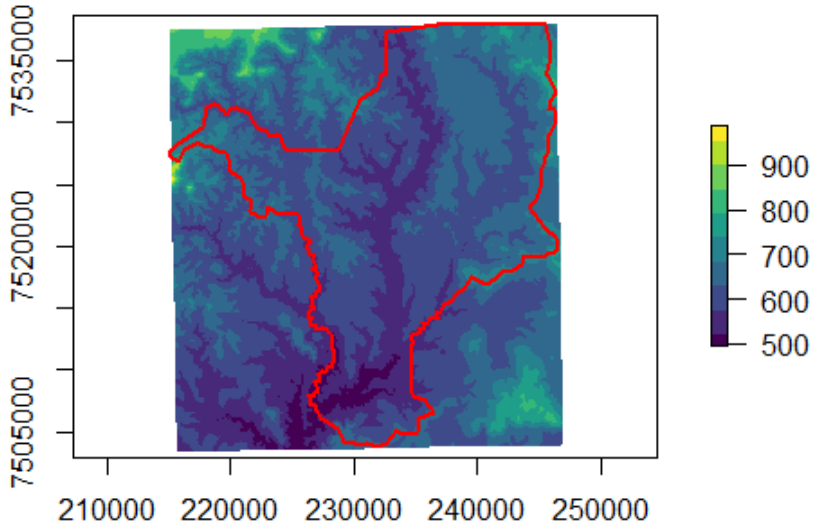


Figura 15.21: Reprojeção do raster de elevação para SIRGAS2000/UTM23S especificado por um objeto e informando o tamanho da célula.

Vamos também reprojetar uma camada mundial da média de temperatura anual (BIO01), indicando o tamanho da célula para 25.000 m (Figura 15.22).

```
## Reprojeção
geo_raster_bioclim_moll <- raster::projectRaster(
    from = geo_raster_bioclim[[1]],
    crs = "+proj=moll",
    res = 25000,
    method = "bilinear")
geo_raster_bioclim_moll
#> class      : RasterLayer
#> dimensions : 732, 1453, 1063596  (nrow, ncol, ncell)
#> resolution : 25000, 25000  (x, y)
#> extent     : -18159905, 18165095, -9154952, 9145048  (xmin, xmax, ymin,
ymax)
#> crs        : +proj=moll +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_
defs
#> source     : memory
#> names      : wc2.1_10m_bio_1
#> values     : -54.66752, 30.71805  (min, max)
## Plot
plot(geo_raster_bioclim_moll, col = viridis::viridis(10))
plot(co110_sf_moll[1], col = NA, add = TRUE)
```

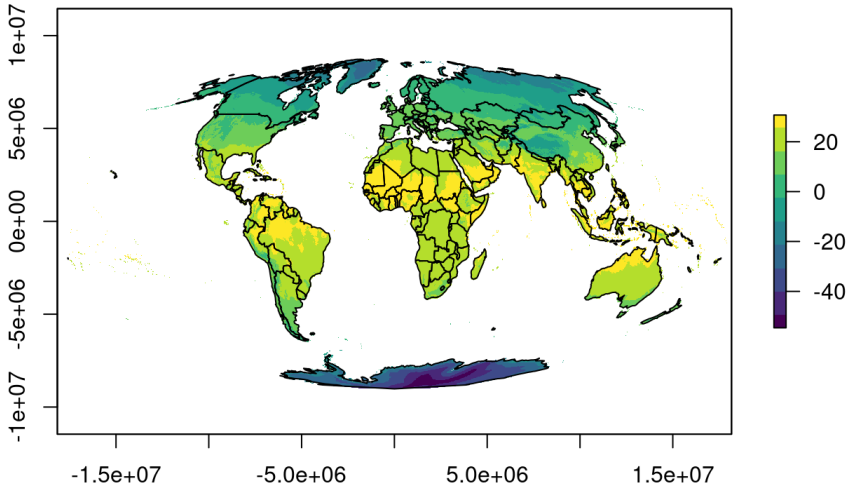


Figura 15.22: Reprojeção do raster de média de temperatura anual (BIO01) para Projeção de Mollweide informando o tamanho da célula.

15.9 Principais operações com dados geoespaciais

Nesta seção veremos as principais funções para realizar operações com dados geoespaciais. Essas operações são separadas conforme Lovelace et al. (2019) em: **Operações de atributos**, **Operações espaciais**, e **Operações geométricas**.

15.9.1 Operações de atributos

São modificação de objetos geoespaciais baseado em informações não espaciais, como a tabela de atributos ou valores das células e nome das camadas dos rasters.

Vetor

As principais operações de atributos vetoriais são com respeito à tabela de atributos, sendo as principais: i) filtro, ii) junção, iii) agregação e iv) manipulação da tabela de atributos. A lista de possíveis operações é longa, dessa forma, apresentaremos algumas operações utilizando as principais funções e listamos as demais funções e suas operações, que dependerão de objetivos específicos.

Quase todas as operações serão as mesmas realizadas pelo pacote `dplyr` em uma tabela de dados (ver o Capítulo 5), sendo algumas operações específicas para alterar apenas campos da tabela de atributos e outras que refletem operações nas feições, ou seja, alterarão através da tabela de atributos as características das feições. Essas funções e suas operações são descritas com detalhes na Tabela 15.11.

Tabela 15.11. Principais funções para realizar operações de atributos e suas descrições.

Funções	Onde atua	Descrição
`filter()`	Feições	Selecionar feições por valores
`slice()`	Feições	Selecionar feições pela posição na tabela de atributos
`n_sample()`	Feições	Amostrar feições na tabela de atributos
`group_by()`	Feições	Agrupar feições por valores da tabela de atributos
`summarise()`	Feições	Operações com valores das feições na tabela de atributos, que acabam por dissolver as feições
`select()`	Atributos	Selecionar colunas da tabela de atributos
`pull()`	Atributos	Selecionar uma coluna da tabela de atributos como vetor
`rename()`	Atributos	Renomear uma coluna da tabela de atributos
`mutate()`	Atributos	Criar uma coluna ou alterar os valores da tabela de atributos
`*_join()`	Atributos	Diversas funções para juntar dados de outras tabelas de dados à tabela de atributos

Para exemplificar as operações de atributos, vamos utilizar os dados de nascentes, hidrologia e cobertura da terra para o município de Rio Claro/SP.

Filtro

Vamos iniciar as operações fazendo o filtro de feições pela tabela de atributos, que permite selecionar feições pelos seus valores atribuídos, utilizando a função `dplyr::filter()`. Aqui vamos selecionar as feições de floresta do mapa de cobertura da terra para Rio Claro/SP (Figura 15.23).

```
## Filtro
geo_vetor_cobertura_floresta <- geo_vetor_cobertura %>%
    dplyr::filter(CLASSE_USO == "formação florestal")
```

```
## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_cobertura_floresta$geometry, col = "forestgreen",
     add = TRUE)
```

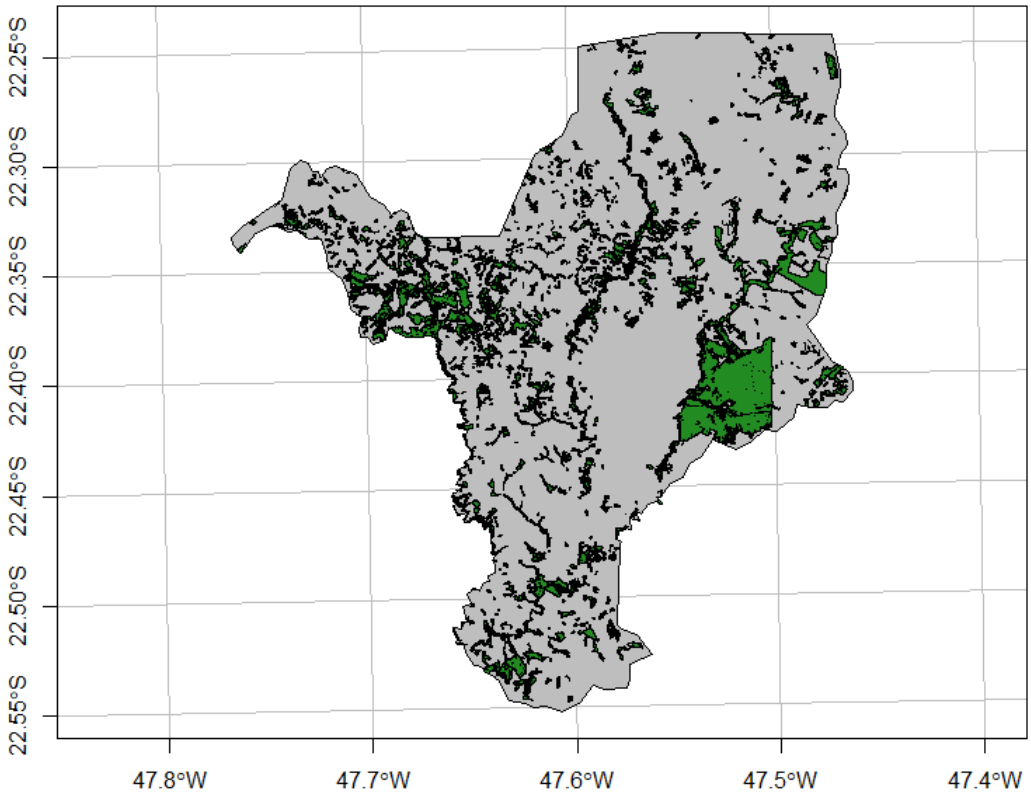


Figura 15.23: Filtro da classe floresta para o mapeamento de cobertura da terra para o município de Rio Claro/SP.

Junção

Uma das funções mais úteis de operações de atributos é a junção, referida em inglês como *join*, realizada através das funções `dplyr::*_join()` (ver detalhes do Capítulo 5). Nela, usamos uma coluna identificadora para atribuir dados de outra tabela de dados. Como exemplo, vamos criar uma tabela de dados com novos nomes das classes de cobertura da terra e atribuir esses novos nomes à tabela de atributos do objeto vetorial. É fundamental destacar que para que essa função funcione, precisamos de uma coluna identificadora dos valores para que a junção seja possível.

```
## Dados
dados_classes <- tibble::tibble(
    CLASSE_USO = geo_vetor_cobertura$CLASSE_USO,
    classe = c("agua", "antropico", "edificado", "floresta",
               "silvicultura"))
dados_classes
#> # A tibble: 5 × 2
#>   CLASSE_USO         classe
#>   <chr>              <chr>
#> 1 água               agua
#> 2 área antropizada   antropico
#> 3 área edificada     edificado
#> 4 formação florestal floresta
#> 5 silvicultura       silvicultura
```

```
## Junção
geo_vetor_cobertura_classes <- dplyr::left_join(
    x = geo_vetor_cobertura,
    y = dados_classes,
    by = "CLASSE_USO") %>%
    sf::st_drop_geometry()
geo_vetor_cobertura_classes
#>   GEOCODIGO MUNICIPIO UF CD_UF         CLASSE_USO   AREA_HA       classe
#> 1   3543907 RIO CLARO SP    35              água   357.027         agua
#> 2   3543907 RIO CLARO SP    35  área antropizada 37297.800    antropico
#> 3   3543907 RIO CLARO SP    35     área edificada  5078.330    edificado
#> 4   3543907 RIO CLARO SP    35 formação florestal  7017.990     floresta
#> 5   3543907 RIO CLARO SP    35       silvicultura   138.173 silvicultura
```

Agregação

Outra função bastante útil é a agregação de atributos. Apesar de existir uma função que realiza a união de feições que veremos na próxima seção, o uso conjunto das funções `dplyr::group_by()` e `dplyr::summarise()` realizam uma tarefa semelhante. Aqui vamos agregar as nascentes para Rio Claro/SP, i.e., juntar cada ponto que estava numa linha da tabela de atributos de modo que todos fiquem numa mesma linha, com o valor da quantidade de nascentes (Figura 15.24).

```
## Agregar
geo_vetor_nascentes_n <- geo_vetor_nascentes %>%
    dplyr::group_by(MUNICIPIO, HIDRO) %>%
    dplyr::summarise(n = n())
geo_vetor_nascentes_n

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom,
     col = "gray", main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_nascentes_n$geometry, pch = 20,
     col = "blue", add = TRUE)
```

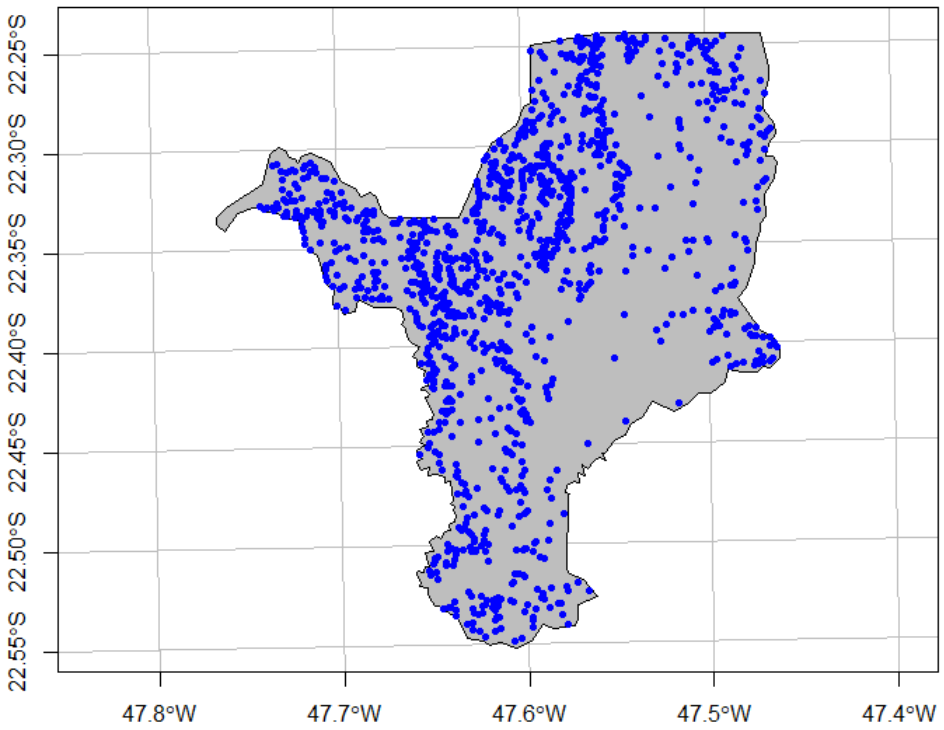


Figura 15.24: Agregação e contagem das nascentes para o município de Rio Claro/SP.

Manipulação da tabela de atributos

Por fim, é muito comum em análises de softwares SIG a criação ou atualização dos valores de colunas na tabela de atributos. Aqui, podemos utilizar a função `dplyr::mutate()` para criar essas novas colunas, assim como atualizar os valores de colunas existentes. Em nosso exemplo, faremos uma composição das colunas `CLASSE_USO` e `AREA_HA` numa nova coluna chamada `classe_area`.

```
## Criar coluna
geo_vetor_cobertura_cob_col_area <- geo_vetor_cobertura %>%
    dplyr::mutate(classe_area = paste0(CLASSE_USO,
                  " (", AREA_HA, " ha)")) %>%
    sf::st_drop_geometry()
geo_vetor_cobertura_cob_col_area
```

Duas funções são bastante interessantes de serem integradas junto à manipulação de tabelas de atributos. Elas calculam propriedades geométricas numéricas dos vetores de linhas (comprimento) e polígonos (área): `sf::st_length()` e `sf::st_area()`. Essas funções calculam essas propriedades em metros para comprimento e em metros quadrados para área, independentemente do CRS. Para tanto, vamos utilizar as linhas de hidrografia e os polígonos de cobertura da terra para Rio Claro/SP, e atribuir esses valores à tabela de atributos de ambos os objetos geoespaciais, utilizando em conjunto a função `dplyr::mutate()`.

```
## Comprimento das linhas
geo_vetor_hidrografia_comp <- geo_vetor_hidrografia %>%
    dplyr::mutate(comprimento = sf::st_length(.))
geo_vetor_hidrografia_comp
## Área dos polígonos
geo_vetor_cobertura_area <- geo_vetor_cobertura %>%
```

```
    dplyr::mutate(area_m2 = sf::st_area(.))
geo_vetor_cobertura_area
```

Raster

Devido à estrutura espacial do raster ser formada por uma ou mais superfícies contínuas, as manipulações como subconjunto e outras operações em objetos raster funcionam de uma maneira diferente do que em objetos vetoriais. Veremos aqui as três principais: i) subconjunto de células usando o operador `[]` ou subconjunto de camadas `RasterStack` ou `RasterBrick` utilizando os operadores `[[]]` e `$`, ii) renomear nomes das camadas, e iii) resumir informações de todos os pixels.

Subconjunto

Podemos fazer um subconjunto de células utilizando dentro dos operadores `[]` valores para indicar a posição da linha e da coluna de um raster, ou ainda a posição de uma célula utilizando apenas um número. Essas operações resultarão em valores diferentes para `RasterLayer` e `RasterBrick` ou `RasterStack`.

```
## Raster - linha 1 e columna 1
geo_raster_srtm[1, 1]
#>
#> 382

## Raster - célula 1
geo_raster_srtm[1]
#>
#> 382

## Stack - linha 1 e columna 1
geo_raster_bioclim[1, 1]

## Stack - célula 1
geo_raster_bioclim[1]
```

Para selecionar uma camada de um `RasterBrick` ou `RasterStack`, podemos utilizar as funções `raster::subset()` ou `raster::raster()` com o argumento `layer` indicando a ordem ou o nome da camada, além dos operadores `[[]]` e `$` (Figura 15.25).

```
## Seleção de camada num objeto stack utilizando a função subset
geo_raster_bioclim_bio01 <- raster::subset(geo_raster_bioclim, "wc2.1_10m_
bio_1")
geo_raster_bioclim_bio01

## Seleção de camada num objeto stack utilizando a função raster
geo_raster_bioclim_bio01 <- raster::raster(geo_raster_bioclim, layer = 1)
geo_raster_bioclim_bio01
## Seleção de camada num objeto stack utilizando os operadores [[]] e o nome
geo_raster_bioclim_bio01 <- geo_raster_bioclim[["wc2.1_10m_bio_1"]]
geo_raster_bioclim_bio01
```

```
## Seleção de camada num objeto stack utilizando os operadores [[]] e a
posicao
geo_raster_bioclim_bio01 <- geo_raster_bioclim[[1]]
geo_raster_bioclim_bio01

## Seleção de camada num objeto stack utilizando o operador $
geo_raster_bioclim_bio01 <- geo_raster_bioclim$wc2.1_10m_bio_1
geo_raster_bioclim_bio01

# Plot
plot(geo_raster_bioclim_bio01, col = viridis::viridis(10))
```

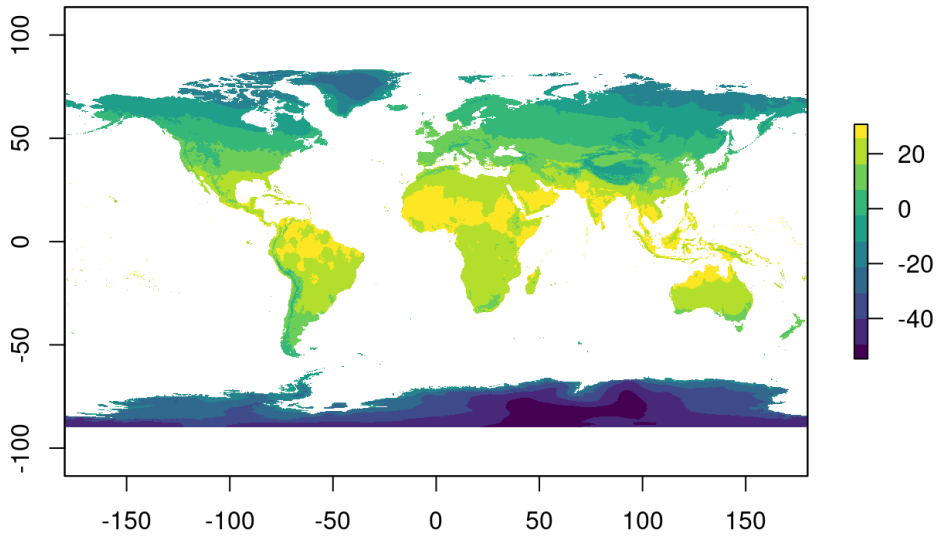


Figura 15.25: Camada BIO01 selecionada pelas operações de subconjunto.

Renomear

Podemos ainda renomear camadas dos raster `RasterLayer` utilizando a função `names()`.

```
## Raster - nomes
names(geo_raster_srtm_rio_claro)
#> [1] "srtm_27_17"

## Raster - renomear
names(geo_raster_srtm_rio_claro) <- "elevacao"

## Raster - nomes
names(geo_raster_srtm_rio_claro)
#> [1] "elevacao"
```

E essa operação também funciona para `RasterBrick` e `RasterStack`.

```
## Stack - nomes
names(geo_raster_bioclim)
```

```
#>  [1] "wc2.1_10m_bio_1"  "wc2.1_10m_bio_10" "wc2.1_10m_bio_11" "wc2.1_10m_
bio_12" "wc2.1_10m_bio_13" "wc2.1_10m_bio_14"
#>  [7] "wc2.1_10m_bio_15" "wc2.1_10m_bio_16" "wc2.1_10m_bio_17" "wc2.1_10m_
bio_18" "wc2.1_10m_bio_19" "wc2.1_10m_bio_2"
#> [13] "wc2.1_10m_bio_3"  "wc2.1_10m_bio_4"  "wc2.1_10m_bio_5"  "wc2.1_10m_
bio_6"  "wc2.1_10m_bio_7"  "wc2.1_10m_bio_8"
#> [19] "wc2.1_10m_bio_9"

## Stack - renomear
names(geo_raster_bioclim) <- c("bio01", paste0("bio", 10:19), paste0("bio0",
2:9))

## Stack - nomes
names(geo_raster_bioclim)
#>  [1] "bio01" "bio10" "bio11" "bio12" "bio13" "bio14" "bio15" "bio16"
"bio17" "bio18" "bio19" "bio02" "bio03" "bio04" "bio05"
#> [16] "bio06" "bio07" "bio08" "bio09"
```

Resumir

Muitas vezes queremos fazer cálculos para todos as células de um raster. Podemos resumir informações de todos os pixels fazendo cálculos simples com todos os pixels de cada camada com a função `raster::cellStats()`, sendo `x` o argumento do objeto raster e `stat` o nome da função resumo, como "mean" ou "sum."

```
## Raster - média de todas as células de altitude
raster::cellStats(x = geo_raster_srtm_rio_claro, stat = mean)
#> [1] 625.8273

## Stack - média de todas as células de cada camada bioclimática
raster::cellStats(x = geo_raster_bioclim, stat = mean)
```

Ou ainda, podemos analisar a frequência com que cada valor dos pixels ocorre, utilizando a função `raster::freq()`.

```
## Raster - frequência das células
raster::freq(x = geo_raster_srtm_rio_claro) %>% head()
#>      value count
#> [1,]   491     1
#> [2,]   492     4
#> [3,]   493     9
#> [4,]   494    19
#> [5,]   495    32
#> [6,]   496    44

## Stack - frequência das células
raster::freq(x = geo_raster_bioclim[[1]]) %>% head()
#>      value count
#> [1,]   -55   319
#> [2,]   -54  4529
```

```
#> [3,]   -53  5778
#> [4,]   -52  6128
#> [5,]   -51  6090
#> [6,]   -50  7892
```

15.9.2 Operações espaciais

As operações espaciais são modificações de objetos geoespaciais baseado em informações espaciais, como localização e formato. Seria quase impossível abordar todas as operações realizáveis nesse capítulo, então demonstraremos as principais para dados vetoriais e raster.

Vetor

As principais operações espaciais para dados vetoriais são: i) filtro espacial, ii) junção espacial, iii) agregação espacial e iv) distância espacial. Apresentaremos essas operações utilizando as principais funções utilizando os dados de nascentes, hidrologia e cobertura da terra para o município de Rio Claro/SP.

Filtro espacial

Filtros espaciais são operações que realizam seleção de feições espaciais entre dois objetos geoespaciais (x e y). Existe uma grande quantidade de funções para realizar filtros espaciais no R, como podemos ver na Tabela 15.12. Essas funções verificam se cada feição em x mantém sua relação em y. Ao especificar o parâmetro `sparse = FALSE`, as funções retornam uma matriz lógica (composta por `TRUE` e `FALSE`).

Tabela 15.12. Principais pacotes para composição de mapas no R.

Função	Descrição	Função inversa
`sf::st_contains()`	Nenhum dos pontos de x está fora de y	`st_within`
`sf::st_contains_properly()`	x contém y, e y não tem pontos em comum com a fronteira de x	NA
`sf::st_covers()`	Nenhum ponto de y se encontra no exterior de x	`st_covered_by`
`sf::st_covered_by()`	Inverso de `sf::st_covers()`	NA
`sf::st_crosses()`	x e y têm alguns, mas não todos os pontos internos em comum	NA
`sf::st_disjoint()`	x e y não têm pontos em comum	`st_intersects`
`sf::st_equals()`	x e y são geometricamente iguais; o número de pedido dos nós pode ser diferente	NA
`sf::st_equals_exact()`	x e y são geometricamente iguais e têm ordem de nó idêntica	NA
`sf::st_intersects()`	x e y não são separados	`st_disjoint`
`sf::st_is_within_distance()`	x está mais perto de y do que uma determinada distância	NA
`sf::st_within()`	Nenhum dos pontos de y está fora de x	`st_contains`

Função	Descrição	Função inversa
`sf::st_touches()`	x e y têm pelo menos um ponto limite em comum, mas nenhum ponto interno	NA
`sf::st_overlaps()`	x e y têm alguns pontos em comum; a dimensão destes é idêntica à de x e y	NA
`sf::st_relate()`	Dado um padrão, retorna se x e y aderem a este padrão	NA

Em nosso exemplo, utilizaremos a função `sf::intersects()` para filtrar as nascentes dentro de floresta para Rio Claro/SP. Essa função vai retornar a resposta binária se as nascentes estão (1) ou não (*empty*) dentro dos polígonos de floresta.

```
## Filtro espacial
sf::st_intersects(x = geo_vetor_nascentes,
                  y = geo_vetor_cobertura_floresta)
#> Sparse geometry binary predicate list of length 1220, where the predicate
was `intersects'
#> first 10 elements:
#>  1: 1
#>  2: 1
#>  3: (empty)
#>  4: 1
#>  5: (empty)
#>  6: (empty)
#>  7: (empty)
#>  8: (empty)
#>  9: 1
#>  10: (empty)
```

Podemos usar essa mesma função em conjunto com a função `dplyr::filter()` para filtrar as nascentes dentro de florestas, mas agora com o argumento `sparse = FALSE` para valores lógicos funcionarem com o filtro.

```
## Filtro espacial - interno
geo_vetor_nascentes_floresta_int <- geo_vetor_nascentes %>%
    dplyr::filter(sf::st_intersects(x = ., y = geo_vetor_cobertura_floresta,
sparse = FALSE))
```

Ou ainda podemos utilizar o operador `[]` para realizar esse filtro, como podemos notar na Figura 15.26.

```
## Filtro espacial com [] - interno
geo_vetor_nascentes_floresta_int <- geo_vetor_nascentes[
                                    geo_vetor_cobertura_floresta, ]
## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_cobertura_floresta$geometry, col = "forestgreen",
     add = TRUE)
```

```
plot(geo_vetor_nascentes_floresta_int$geometry, col = "blue",
     pch = 20, cex = 1, add = TRUE)
```

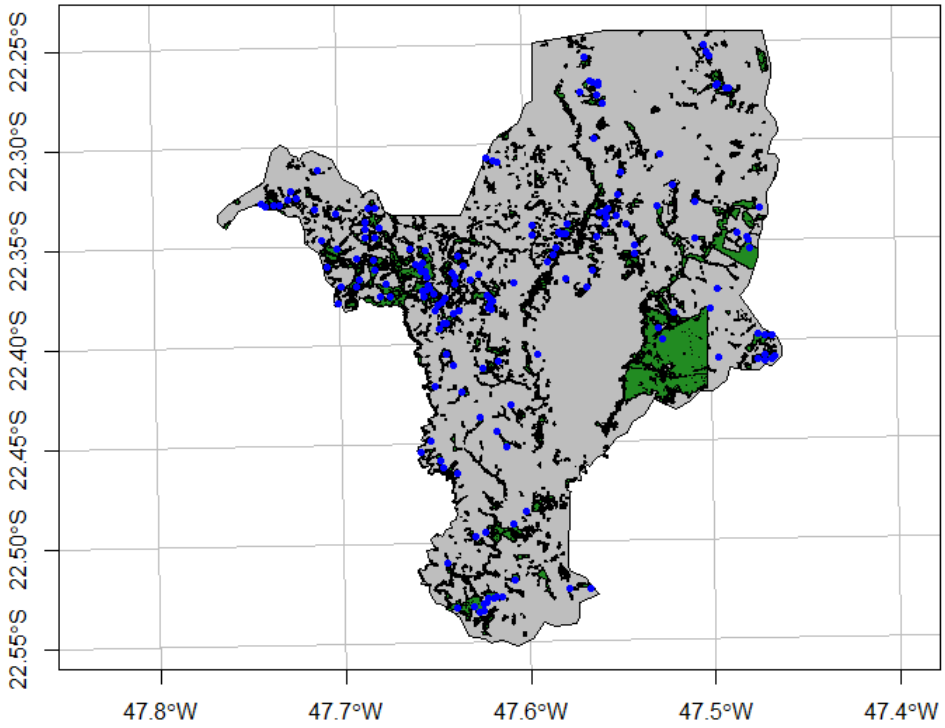


Figura 15.26: Nascentes dentro de florestas no município de Rio Claro/SP.

Entretanto, muitas vezes queremos fazer o filtro de feições que estão fora de feições de outro objeto espacial. Para isso, podemos usar a função `sf::st_disjoint()` ou ainda utilizando o operador `[]`, mas com o argumento `op`, nesse caso utilizando a mesma função `sf::st_disjoint()` como operação (Figura 15.27). Atentar o segundo argumento vazio nessa operação.

```
## Filtro espacial - externo
geo_vetor_nascentes_floresta_ext <- geo_vetor_nascentes %>%
    dplyr::filter(sf::st_disjoint(x = .,
                                  y = geo_vetor_cobertura_floresta,
                                  sparse = FALSE))

## Filtro espacial com [] - externo
geo_vetor_nascentes_floresta_ext <- geo_vetor_nascentes[
                                    geo_vetor_cobertura_floresta, ,
                                    op = st_disjoint]

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_cobertura_floresta$geometry, col = "forestgreen",
     add = TRUE)
```

```
plot(geo_vetor_nascentes_floresta_ext$geometry, col = "steelblue",
     pch = 20, cex = 1, add = TRUE)
```

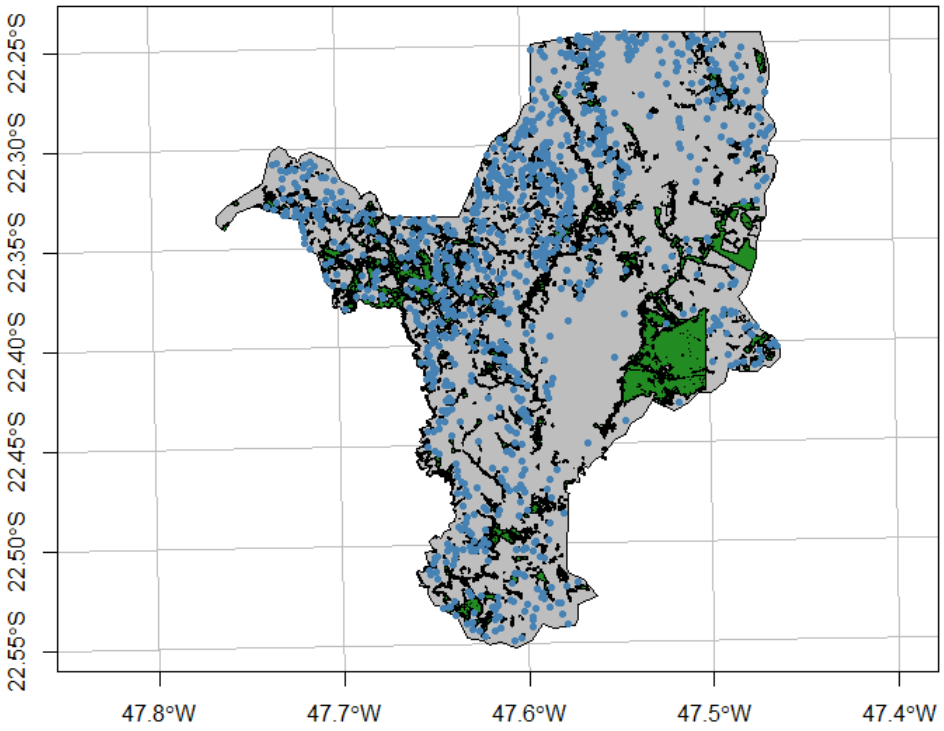


Figura 15.27: Nascentes fora de florestas no município de Rio Claro/SP.

Junção espacial

Outra operação muito usada dentro de análises espaciais é a junção espacial ou do inglês *spatial join*. A ideia base é muito semelhante com a junção baseada em atributos, mas aqui atribuiremos o valor da tabela de atributos das feições de um objeto espacial y às feições que fazem intersecção com um objeto espacial x, de modo que esses valores sejam armazenados na tabela de atributos do primeiro objeto espacial.

Para exemplificar, vamos atribuir os valores dos polígonos de cobertura da terra aos pontos de nascentes para Rio Claro/SP, fazendo um agrupamento pela tabela de atributos para permitir criar o mapa da Figura 15.28.

```
## Junção espacial
geo_vetor_nascentes_cob_jun <- geo_vetor_nascentes %>%
    sf::st_join(x = ., y = geo_vetor_cobertura) %>%
    dplyr::group_by(CLASSE_USO) %>%
    dplyr::summarise(n = n())
## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_nascentes_cob_jun[1], col = c("blue", "orange", "gray30",
                                             "forestgreen", "green"),
                                     pch = 20, add = TRUE)
```

```
legend(x = 209000, y = 7520000, pch = 15, cex = .7, pt.cex = 2.5,
       legend = (geo_vetor_nascentes_cob_jun$CLASSE_USO),
       col = c("blue", "orange", "gray30", "forestgreen", "green"))
```

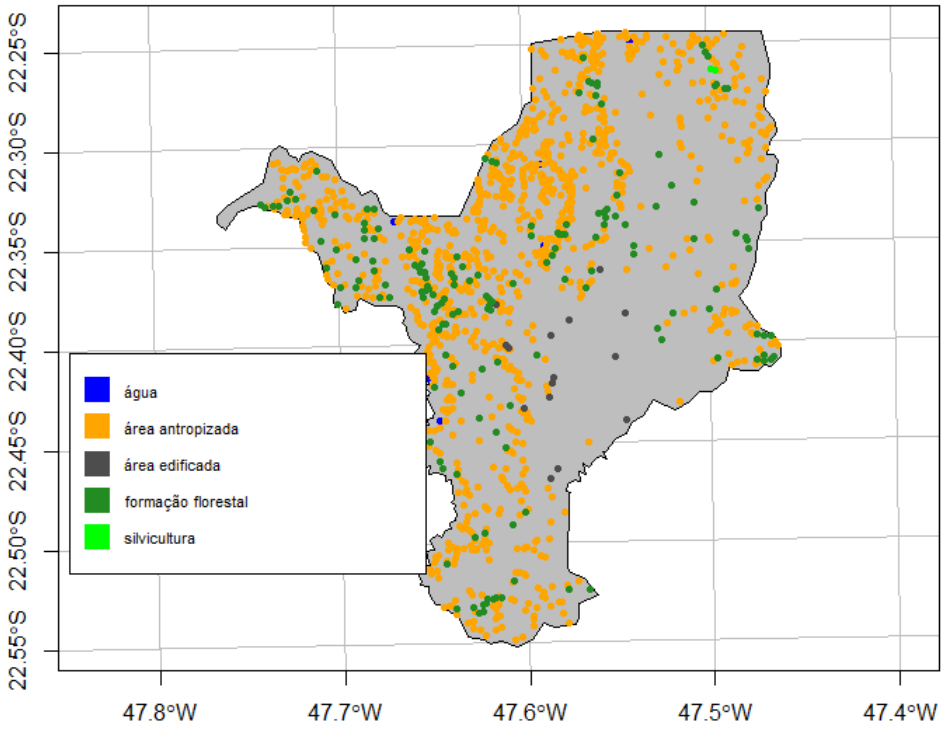


Figura 15.28: Junção espacial da cobertura da terra para as nascentes no município de Rio Claro/SP.

Agregação espacial

Muitas vezes queremos contabilizar quantas feições ou agregar valores de feições para polígonos. Podemos realizar essa operação usando as funções `dplyr::group_by()` e `dplyr::summarise`, ou utilizar a função `aggregate()`. Nesse exemplo, vamos contabilizar quantas nascentes há por polígono de cobertura da terra para o município de Rio Claro/SP (Figura 15.29).

```
## Agregação espacial
geo_vetor_cobertura_nas_agre <- geo_vetor_nascentes %>%
    aggregate(x = ., by = geo_vetor_cobertura, FUN = length)
## Plot
plot(geo_vetor_cobertura_nas_agre[1], axes = TRUE, graticule = TRUE,
     main = NA)
```

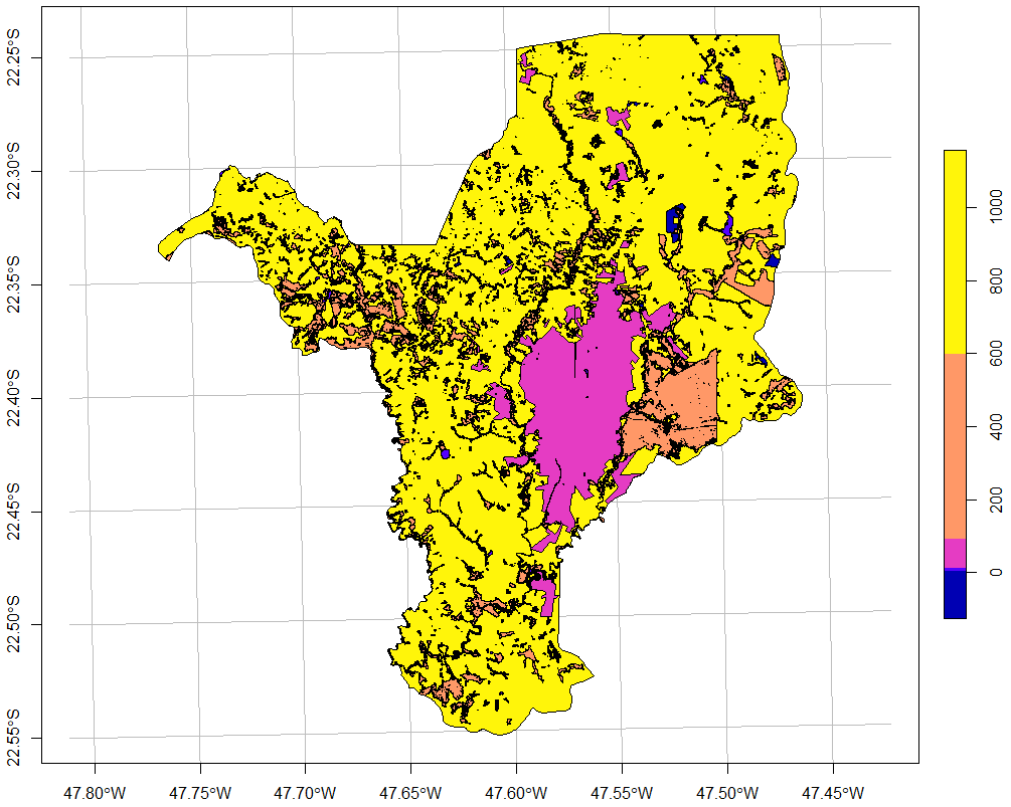


Figura 15.29: Agregação espacial contabilizando o número de nascentes para cada classe de cobertura da terra no município de Rio Claro/SP.

Distância espacial

A distância espacial é a distância calculada em duas dimensões (2D) entre um objeto espacial x e y baseado no CRS e para cada feição dos objetos geoespaciais. Para realizar esse cálculo, utilizamos a função `sf::st_distance()`. Em nosso exemplo, vamos calcular a distância das nascentes até a floresta mais próxima, e adicionar essa informação para cada ponto na tabela de atributos com a função `dplyr::mutate()`, para o município de Rio Claro/SP (Figura 15.30).

```
## Distância espacial
geo_vetor_nascentes_dist_flo <- geo_vetor_nascentes %>%
    dplyr::mutate(dist_flo = sf::st_distance(geo_vetor_nascentes,
                                             geo_vetor_cobertura_floresta))

## Plot
plot(geo_vetor_nascentes_dist_flo[7], pch = 20, axes = TRUE,
     graticule = TRUE, main = NA)
```

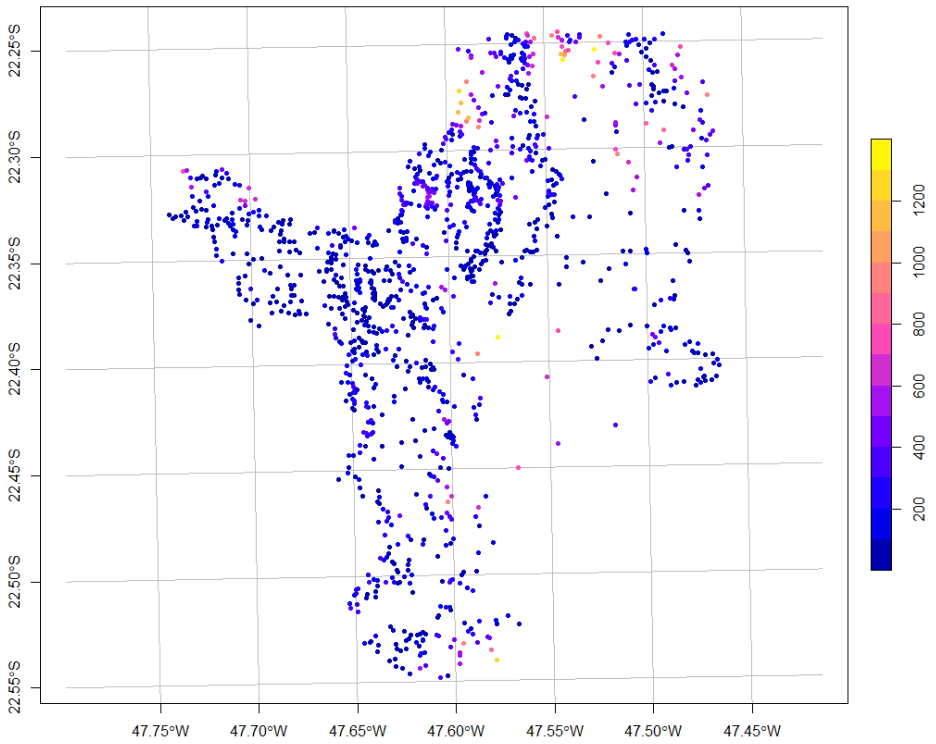


Figura 15.30: Distância espacial das nascentes até o fragmento de floresta mais próxima no município de Rio Claro/SP.

Raster

As principais operações espaciais para dados raster podem ser classificas, segundo Lovelace et al. (2019), em: i) operações locais (por célula), ii) operações focais (por bloco de múltiplas células regulares - e.g., 3x3), iii) operações zonais (por bloco de múltiplas células irregulares) e iv) operações globais (por um ou vários rasters inteiros). Cada uma delas é aplicada para objetivos e escalas espaciais específicas. Para os exemplos desta seção, utilizaremos o dado raster de elevação para o município de Rio Claro/SP.

Operações locais

As operações locais contemplam todas as operações realizadas célula a célula em uma ou várias camadas de um objeto raster. A álgebra de raster é uma das mais comuns, simples e poderosas operações no R envolvendo rasters. Com ela podemos fazer operações simples através de operadores aritméticos (soma, subtração, multiplicação, divisão ou potenciação) entre dois ou mais objetos raster, ou utilizar funções para alterar todos os valores dos pixels como, por exemplo, as funções `log10()` ou `sqrt()`, ou ainda a função `raster::scale()` para padronizar ou centralizar os valores dos rasters (Figura 15.31).

```
## Soma
geo_raster_srtm_rio_claro2 <- geo_raster_srtm_rio_claro +
                              geo_raster_srtm_rio_claro

## Log10
geo_raster_srtm_rio_claro_log10 <- log10(geo_raster_srtm_rio_claro)
```

```
## Plot
old_par <- par(mfrow = c(1, 2))
plot(geo_raster_srtm_rio_claro2, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)

plot(geo_raster_srtm_rio_claro_log10, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
par(old_par)
```

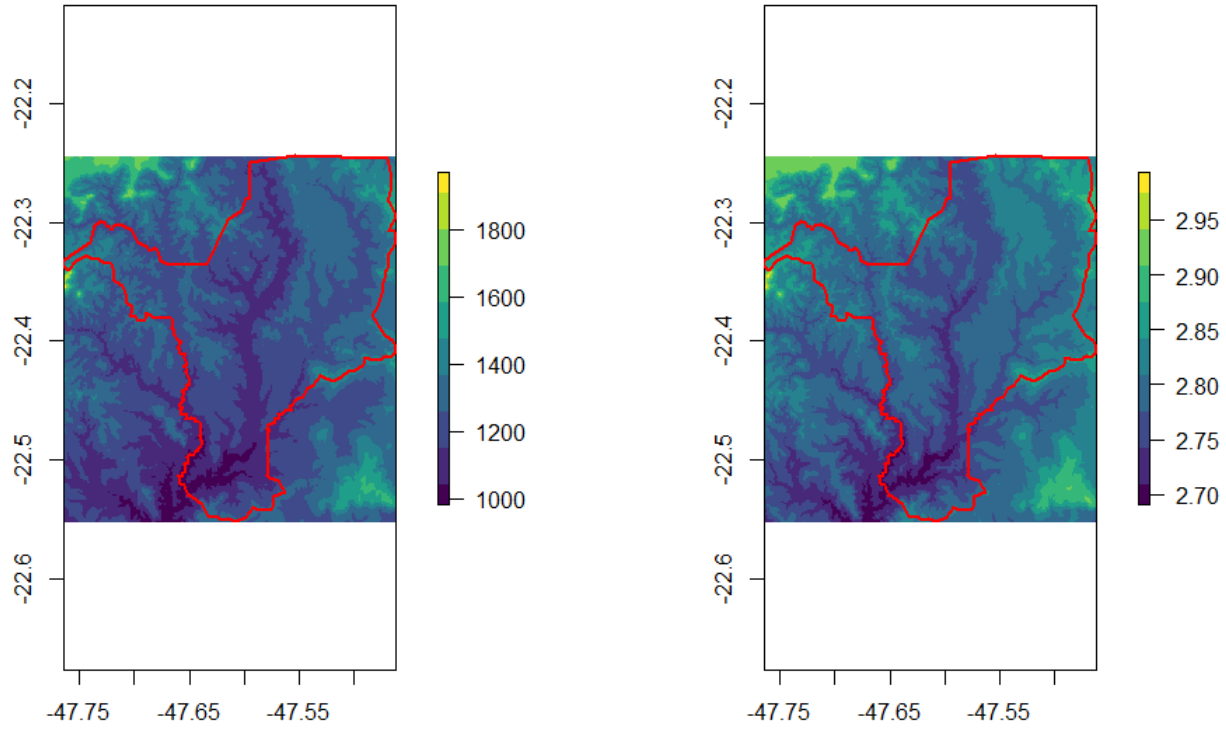


Figura 15.31: Rasters de soma e log10 do mapa de elevação para Rio Claro/SP.

Além das operações aritméticas, a álgebra de rasters também permite operações lógicas, como criar um raster binário (composto por 1 quando a operação lógica é verdadeira, e 0 quanto é falsa). Em nosso caso, buscamos todos os pixels acima de 600 metros para o raster de elevação de Rio Claro/SP (Figura 15.32).

```
## Acima de 600
geo_raster_srtm_rio_claro_acima_600 <- geo_raster_srtm_rio_claro > 600
## Plot
plot(geo_raster_srtm_rio_claro_acima_600, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

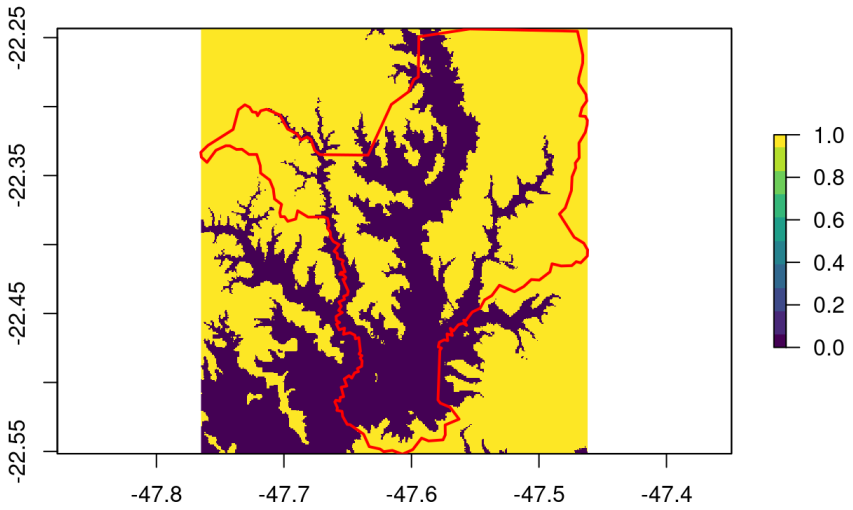


Figura 15.32: Operação local lógica mostrando todos os pixels acima de 600 metros de elevação para Rio Claro/SP.

Além dos operadores aritméticos, também podemos usar as funções `raster::calc()` (uma camada) e `raster::overlay()` (duas ou mais camadas) para realizar operações em todas as células. Elas funcionam com a criação de uma função específica através da função `function()` (Capítulo 4), para que esta seja aplicada em todas as células do raster. Essas funções são muito eficientes, portanto, são preferíveis para grandes conjuntos de dados raster. Exemplificaremos essa operação calculando o produto de todos os pixels por eles mesmos do raster de elevação de Rio Claro/SP (Figura 15.33).

```
## Produto dos pixel - calc
geo_raster_srtm_rio_claro_prod <- raster::calc(
                 x = geo_raster_srtm_rio_claro, fun = function(x){x * x})
geo_raster_srtm_rio_claro_prod
#> class      : RasterLayer
#> dimensions : 370, 364, 134680  (nrow, ncol, ncell)
#> resolution : 0.0008333333, 0.0008333333  (x, y)
#> extent     : -47.765, -47.46167, -22.55167, -22.24333  (xmin, xmax, ymin,
ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : layer
#> values     : 241081, 970225  (min, max)
## Plot
plot(geo_raster_srtm_rio_claro_prod, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
 add = TRUE)
```

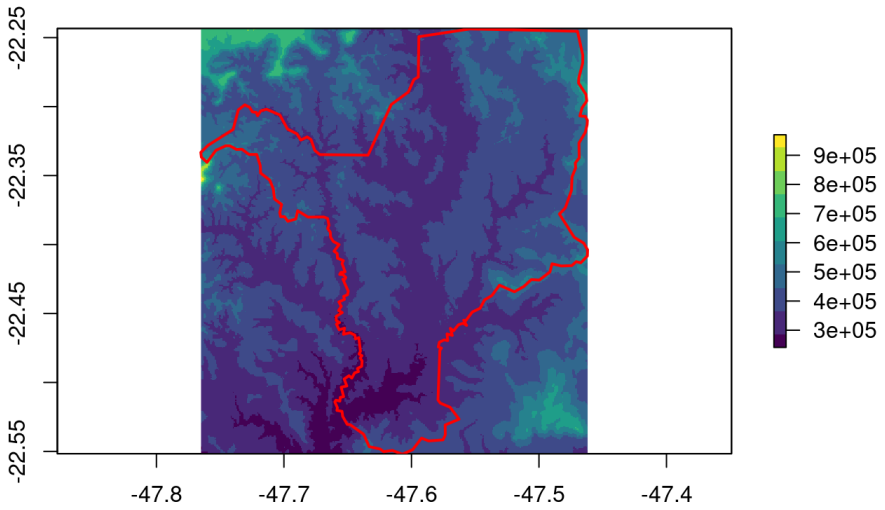


Figura 15.33: Operação local de multiplicação de todos os pixels por eles mesmos do raster de elevação para Rio Claro/SP.

A predição de objetos raster (utilizando a função `raster::predict()`) é outra aplicação extremamente útil em operações locais, nós a veremos mais adiante neste capítulo. Essa função possui basicamente dois argumentos: `object` que é os rasters preditores e `model` com o modelo ajustado para o qual os valores serão preditos com base nos valores dos rasters. A partir da relação entre variáveis respostas (e.g, pontos no espaço, como ocorrência ou riqueza de espécies), e variáveis preditoras (rasters contínuos de elevação, pH, precipitação, temperatura, cobertura da terra ou classe de solo), criamos modelos usando funções como `lm()`, `glm()`, `gam()` ou uma técnica de aprendizado de máquina, e fazemos predições espaciais aplicando os coeficientes estimados aos valores dos raster preditores (consulte os Capítulos 7 e 8).

Por fim, a reclassificação de rasters é outra operação muito comum quando trabalhamos com rasters. Nela é realizada a classificação de intervalos de valores numéricos em grupos, e.g., agrupar um modelo digital de elevação em classes de valores. A função que faz essa operação é a `raster::reclassify()`. Ela possui dois argumentos: `x` que é o raster a ser reclassificado, e o segundo `rcl` para o qual devemos construir uma matriz de reclassificação, onde a primeira coluna é a extremidade inferior, a segunda coluna é a extremidade superior, e a terceira coluna representa o novo valor para os intervalos das colunas um e dois. Vamos reclassificar o raster de elevação de Rio Claro/SP para os intervalos 400–600, 600–800 e 800–1000 que são reclassificados para os valores 1, 2 e 3, respectivamente (Figura 15.34).

```
## Matriz de reclassificação
rcl  <- matrix(c(400,600,1,
                 600,800,2,
                 800,1000,3),
               ncol = 3, byrow = TRUE)

## Reclassificação
geo_raster_srtm_rio_claro_rcl <- raster::reclassify(
                                 x = geo_raster_srtm_rio_claro, rcl = rcl)
## Plot
plot(geo_raster_srtm_rio_claro_rcl, col = viridis::viridis(3))
```

```
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

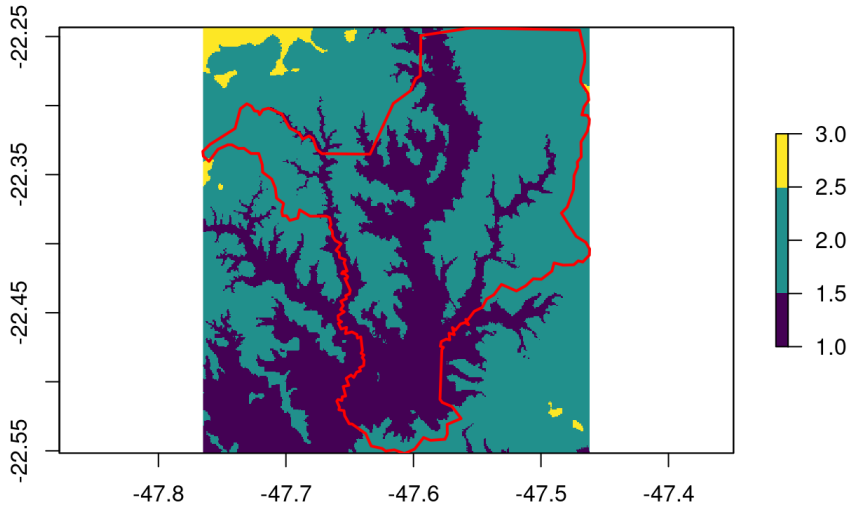


Figura 15.34: Operação local de reclassificação para três classes de elevação para Rio Claro/SP.

Operações focais

As operações focais levam em consideração uma célula central e seus vizinhos. A vizinhança (também chamada de janela móvel - *moving window*) tipicamente é composta de células de 3 por 3 (célula central e seus oito vizinhos), mas pode assumir outra forma. A operação focal aplica uma função de agregação a todas as células dentro da vizinhança especificada, e usa a saída correspondente como o novo valor para a célula central, e segue para a próxima célula central e seus vizinhos. Essa operação é realizada através da função `raster::focal()`. O parâmetro `x` especifica o raster de entrada, o parâmetro `w` define a janela móvel por uma matriz cujos valores correspondem a pesos, e por fim, o parâmetro `fun` especifica a função que desejamos aplicar às células, como `min()`, `max()`, `sum()`, `mean()`, `sd()` ou `var()`. Existem diversas aplicações dessa operação para dados raster, como no processamento de imagens de satélite (ver mais em Wegmann et al. (2016)). Outra utilidade é para o cálculo de características topográficas, como declividade, aspecto e direções de fluxo. Para calcular essas métricas específicas, podemos utilizar a função `raster::terrain()`.

Para nosso exemplo, vamos realizar o cálculo do desvio padrão da elevação e a métrica de aspecto (orientação da vertente) para o raster de elevação em Rio Claro/SP (Figura 15.35).

```
## Janela móvel - moving window
geo_raster_srtm_rio_claro_focal_sd <- raster::focal(
    x = geo_raster_srtm_rio_claro,
    w = matrix(data = 1, nrow = 3, ncol = 3),
    fun = sd)

## Declividade
geo_raster_srtm_rio_claro_asp <- raster::terrain(
                          x = geo_raster_srtm_rio_claro, opt = "aspect")

## Plot
old_par <- par(mfrow = c(1, 2))
```

```
plot(geo_raster_srtm_rio_claro_focal_sd, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)

plot(geo_raster_srtm_rio_claro_asp, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
par(old_par)
```

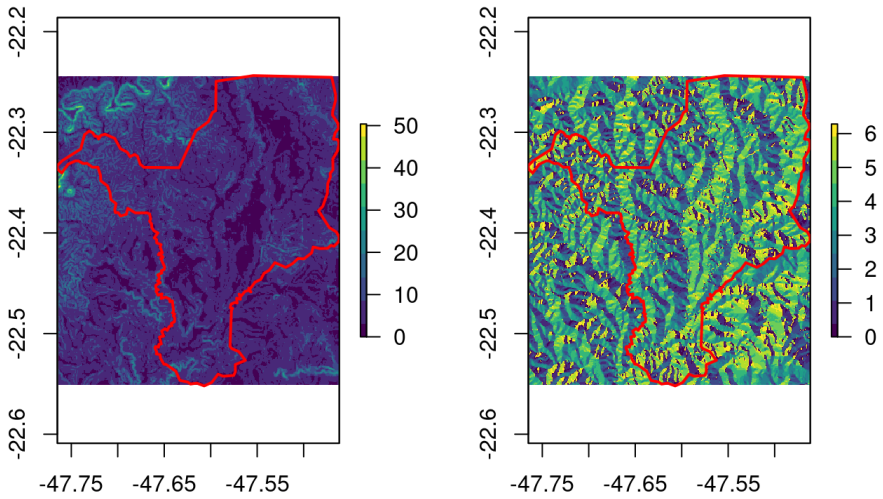


Figura 15.35: Cálculo do desvio padrão da elevação para uma janela de 3x3 e do aspecto para Rio Claro/SP.

Operações zonais

As operações zonais aplicam uma função de agregação para várias células de um raster. Geralmente usa-se um segundo raster categórico para definir as zonas, de modo que as células raster que definem a zona não precisam ser vizinhas, como na operação focal. O resultado de uma operação zonal é uma tabela de resumo agrupada por zona, explicando porque essa operação também é conhecida como estatística zonal. Isso é um contraste com as operações focais que retornam um objeto raster.

A operação zonal é realizada através da função `raster::zonal()`, que recebe de entrada no `x` o raster contínuo, em `z` o raster categórico, e em `fun` a função que resumirá as células. Em nosso exemplo, vamos calcular diversas medidas resumo da elevação com a função `summary()` para cada classe de elevação que criamos anteriormente.

```
## Estatística zonal
geo_raster_srtm_rio_claro_zonal <- data.frame(
                                raster::zonal(geo_raster_srtm_rio_claro,
                                geo_raster_srtm_rio_claro_rcl,
                                fun = "summary"))

colnames(geo_raster_srtm_rio_claro_zonal) <- c("zona", "min",
                                               "1qt", "mediana",
                                               "media", "3qt", "max")
geo_raster_srtm_rio_claro_zonal
#>   zona min 1qt mediana    media 3qt max
```

```
#> 1    1 491 552   574.0 567.5995 589 600
#> 2    2 601 620   640.0 650.6829 670 800
#> 3    3 801 817   832.5 834.2732 846 985
```

Operações globais

As operações globais usam todo o conjunto de dados raster representando uma única zona. As operações globais mais comuns são estatísticas descritivas para todos os pixels do raster, utilizando a função `raster::cellStats()` ou `raster::freq()`, já vistas. Além das estatísticas descritivas, podemos gerar rasters de distância, que calcula a distância de cada célula a uma ou um grupo células-alvo específica, utilizando a função `raster::distance()`.

Em nosso exemplo, vamos selecionar os pixels abaixo de 500 m do raster de elevação e calcular a Distância Euclidiana (Figura 15.36).

```
## Distância euclidiana
geo_raster_srtm_rio_claro_abaixo_500 <- raster::calc(
     x = geo_raster_srtm_rio_claro,
     fun = function(x) ifelse(x < 500, 1, NA))

geo_raster_srtm_rio_claro_global_dist <- raster::distance(
                                    geo_raster_srtm_rio_claro_abaixo_500)
## Plot
plot(geo_raster_srtm_rio_claro_global_dist, col = viridis::viridis(10))
plot(geo_raster_srtm_rio_claro_abaixo_500, add = TRUE, col = "white",
     legend = FALSE)
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

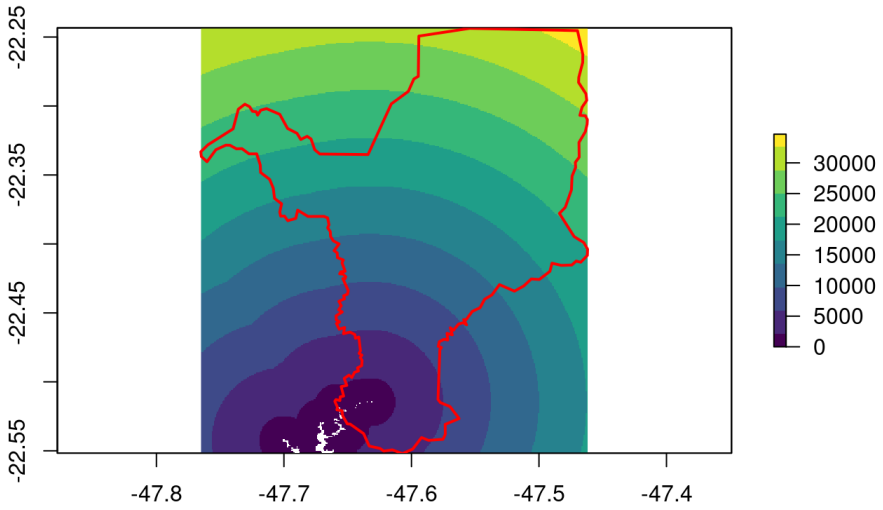


Figura 15.36: Raster de distância Euclidiana dos pixels abaixo de 500 m de elevação para Rio Claro/SP.

15.9.3 Operações geométricas

As operações geométricas realizam modificações em objetos geoespaciais baseado na geometria do vetor ou do raster e na interação e conversão entre vetor-raster. As operações geométricas vetoriais podem ser unárias (funcionam em uma única geometria) ou binárias (modificam uma geometria com base na forma de outra geometria). Ainda podemos fazer transformações para alterar os tipos vetores, que refletirá se as feições são únicas ou múltiplas, inclusive na tabela de atributos. As operações geométricas em rasters envolvem mudar a posição, tamanho e número dos pixels subjacentes e atribuir-lhes novos valores. Por fim, podemos ainda fazer operações de interações e conversões entre raster-vetor para ajustar rasters a vetores, assim como converter um objeto espacial vetorial para raster e vice-versa.

Vetor

Como dissemos, as operações geométricas em vetores criam ou alteram a geometria de objetos da classe `sf`, podendo fazer alterações em única geometria (unárias): i) simplificação, ii) centroides, iii) pontos aleatórios, iv) buffers, v) polígono convexo, vi) polígonos de Voronoi, vii) quadrículas e hexágonos; ou modificar uma geometria com base na forma de outra geometria (binárias), viii) união e ix) recortes; ou ainda fazer transformações de tipo de geometrias.

Para exemplificar as operações geométricas com vetores, vamos utilizar os dados do limite, nascentes, hidrologia e cobertura da terra para o município de Rio Claro/SP.

Simplificação

A simplificação possui o intuito de generalizar linhas ou polígonos, diminuindo assim suas complexidades em relação ao número de vértices. É utilizada para representação em mapas menores ou mapas interativos ou ainda quando um objeto vetorial é muito grande. A função utilizada é a `sf::st_simplify()`, que usa o argumento `x` para uma geometria de entrada e `dTolerance` para controlar o nível de generalização nas unidades do mapa. Em nosso exemplo, simplificaremos a hidrografia de Rio Claro/SP (Figura 15.37).

```
## Simplificação
geo_vetor_hidrografia_simplificado <- sf::st_simplify(
                            x = geo_vetor_hidrografia, dTolerance = 1000)

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
      main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_hidrografia$geometry, col = "steelblue", lwd = 2,
      add = TRUE)
plot(geo_vetor_hidrografia_simplificado$geometry,
     col = adjustcolor("black", .7), add = TRUE)
```

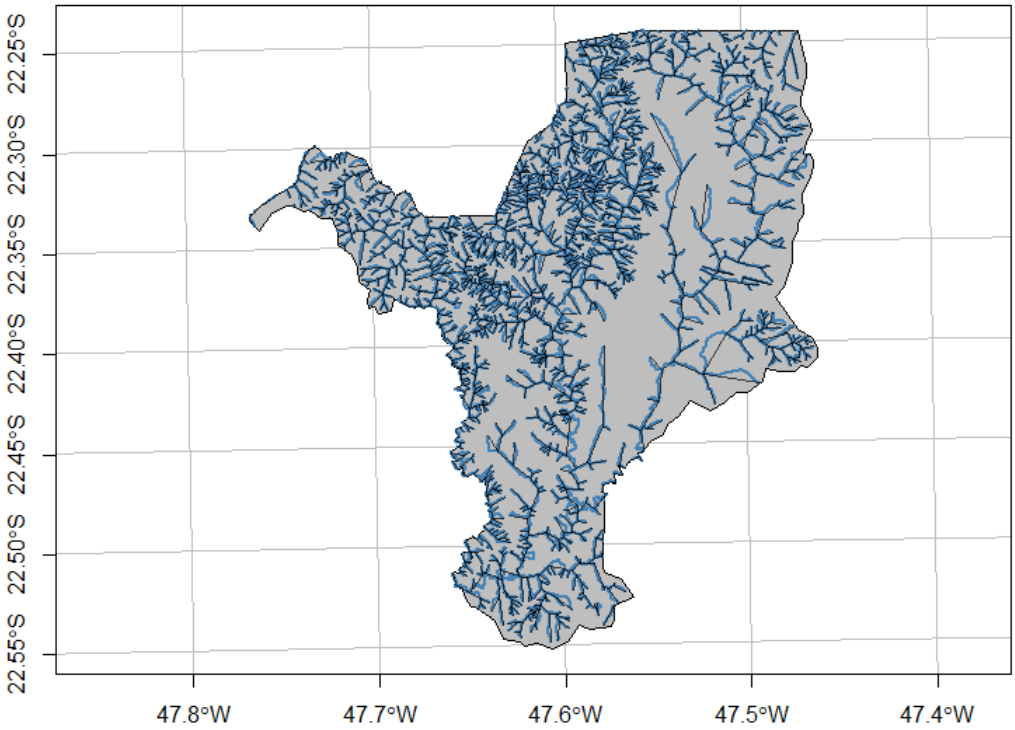


Figura 15.37: Simplificação da hidrografia para Rio Claro/SP. As linhas em azul é a hidrografia original e as linhas em preto mostram a simplificação.

Centroides

A operação de centroides identifica o centro de objetos geoespaciais, geralmente o centro de massa das feições. É utilizado para gerar um ponto simples para representações complexas ou para estimar a distância entre polígonos utilizando esse centroide. Podemos calculá-los com a função `sf::st_centroids()` ou com a função `sf::st_point_on_surface()` para garantir que esses centroides caiam dentro das geometrias. Aqui calcularemos o centroide do município de Rio Claro/SP (Figura 15.38).

```
## Centroides
geo_vetor_rio_claro_sirgas2000_utm23s_cent <- sf::st_centroid(
                                  geo_vetor_rio_claro_sirgas2000_utm23s)

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_cent$geom, cex = 3,
     pch = 20, add = TRUE)
```

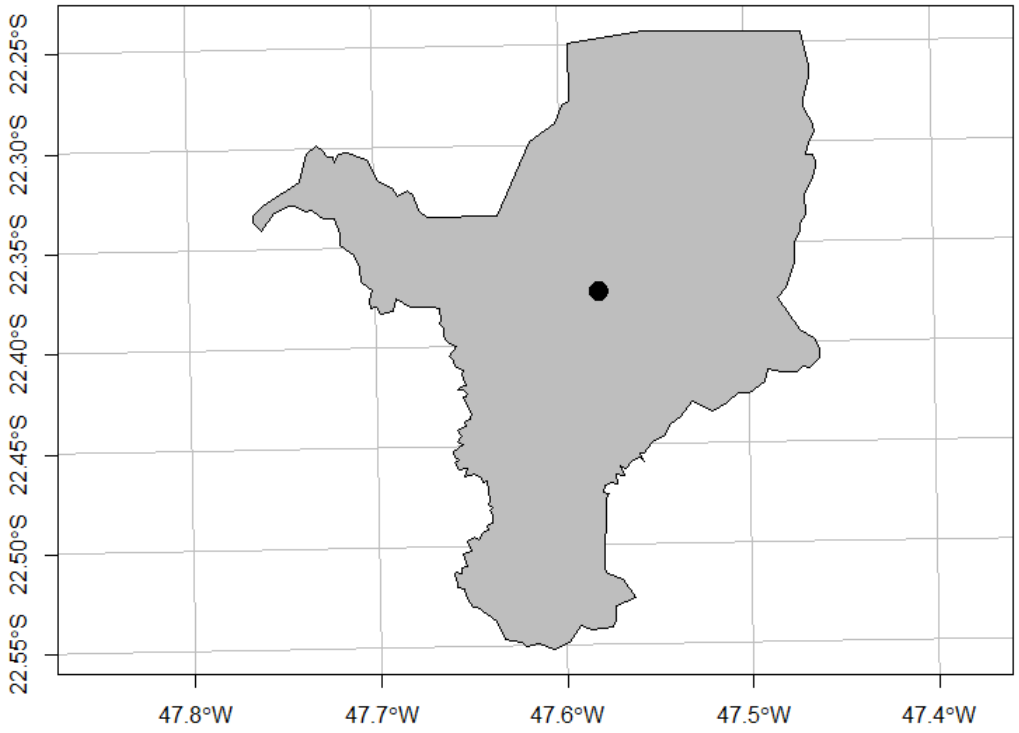


Figura 15.38: Centroide do limite do município de Rio Claro/SP.

Pontos aleatórios

Por vezes precisamos criar algum padrão aleatório dentro de um contexto espacial. Isso pode ser realizado de diversas formas. Uma delas é a criação de pontos aleatórios dentro de um polígono. Podemos realizar essa operação com a função `sf::st_sample()`. Para essa função, dois argumentos são utilizados: `x` uma geometria de entrada e o `size` indicando o número de pontos à seres criados. Outro argumento bastante interessante é o `type`, indicando o tipo de amostragem espacial (aleatório, regular ou triangular). Para nosso exemplo, vamos fixar a amostragem utilizando a função `set.seed()` e sortear 30 pontos para o limite do município de Rio Claro/SP (Figura 15.39). Para mais detalhes da função `set.seed()`, consultar o Capítulo 4.

```
## Fixar amostragem
set.seed(42)

## Pontos aleatórios
geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios <-
                  sf::st_sample(geo_vetor_rio_claro_sirgas2000_utm23s,
                                size = 30)

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios,
     pch = 20, add = TRUE)
```

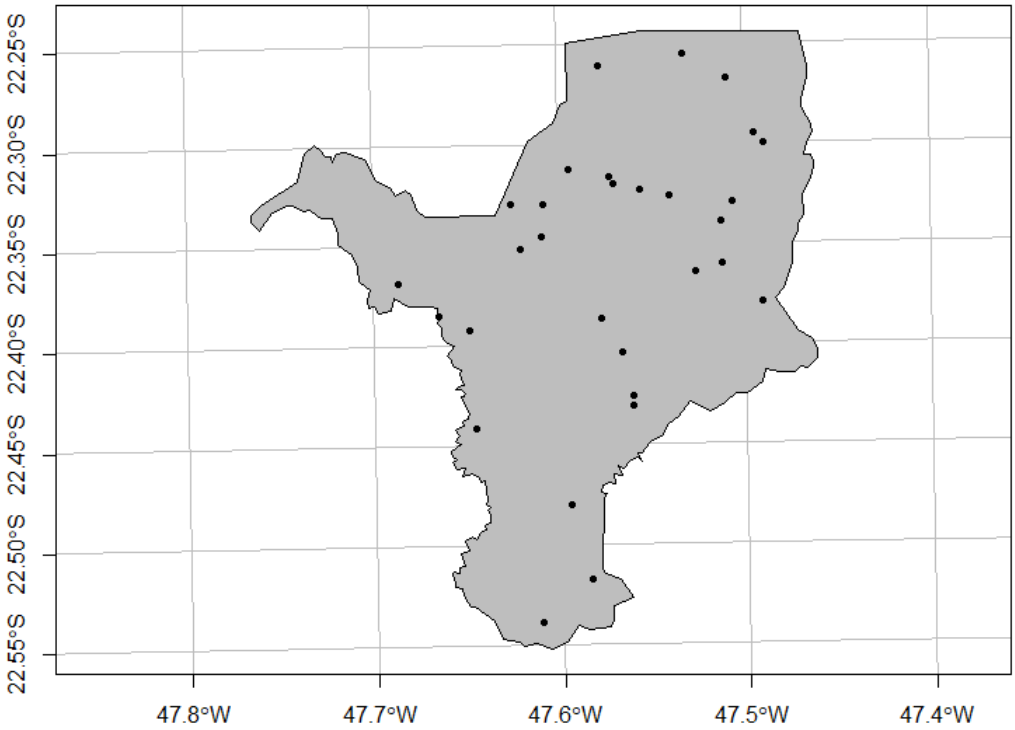


Figura 15.39: Sorteio de 30 pontos aleatório para Rio Claro/SP.

Buffer

Buffers são polígonos que representam a área dentro de uma determinada distância de um elemento geométrico, independentemente de ser um ponto, linha ou polígono. O buffer é comumente utilizado para análise de dados geoespaciais, geralmente sendo entendido como uma unidade amostral, delimitando uma porção no entorno de algum elemento ou evento, como as condições climáticas ou da estrutura da paisagem para uma amostragem, ou as características de cobertura da terra ao longo de um corpo d'água, e.g., Áreas de Preservação Permanente (APPs).

A função utilizada para criar buffers é a `sf::st_buffer()`, que requer pelo menos dois argumentos: `x` uma geometria de entrada e o `dist` uma distância para o buffer, fornecido nas unidades do CRS da geometria de entrada. Em nosso exemplo, vamos criar buffers circulares de 1000 metros para os 30 pontos aleatórios criados anteriormente para o município de Rio Claro/SP (Figura 15.40).

```
## Buffer
geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer <-
            sf::st_buffer(
            x = geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios,
            dist = 1000)

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer,
     col = NA, lwd = 2, border = "red", add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios,
     pch = 20, cex = 1, add = TRUE)
```

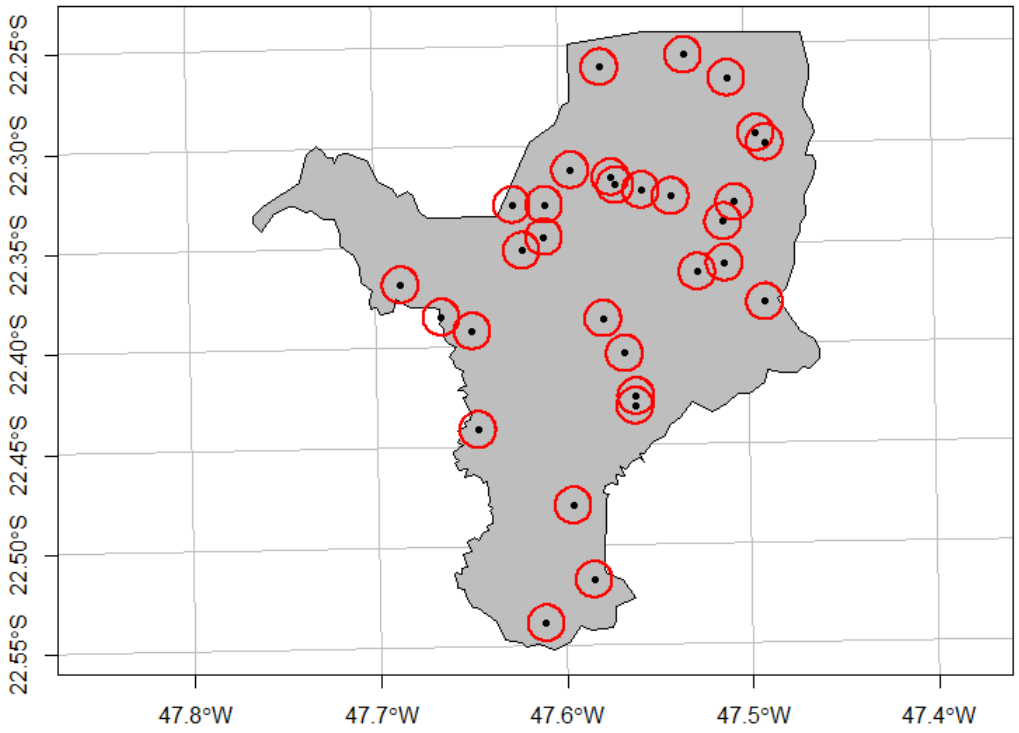


Figura 15.40: Buffers de 1000 metros para os 30 pontos aleatórios no município de Rio Claro/SP.

Podemos ainda criar buffers quadrados acrescentando o argumento `endCapStyle = "SQUARE"` (Figura 15.41).

```
## Buffer
geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer_quad <-
    sf::st_buffer(
    x = geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios,
    dist = 1000, endCapStyle = "SQUARE")

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer_quad,
     col = NA, lwd = 2, border = "red", add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios, pch = 20,
     cex = 1, add = TRUE)
```

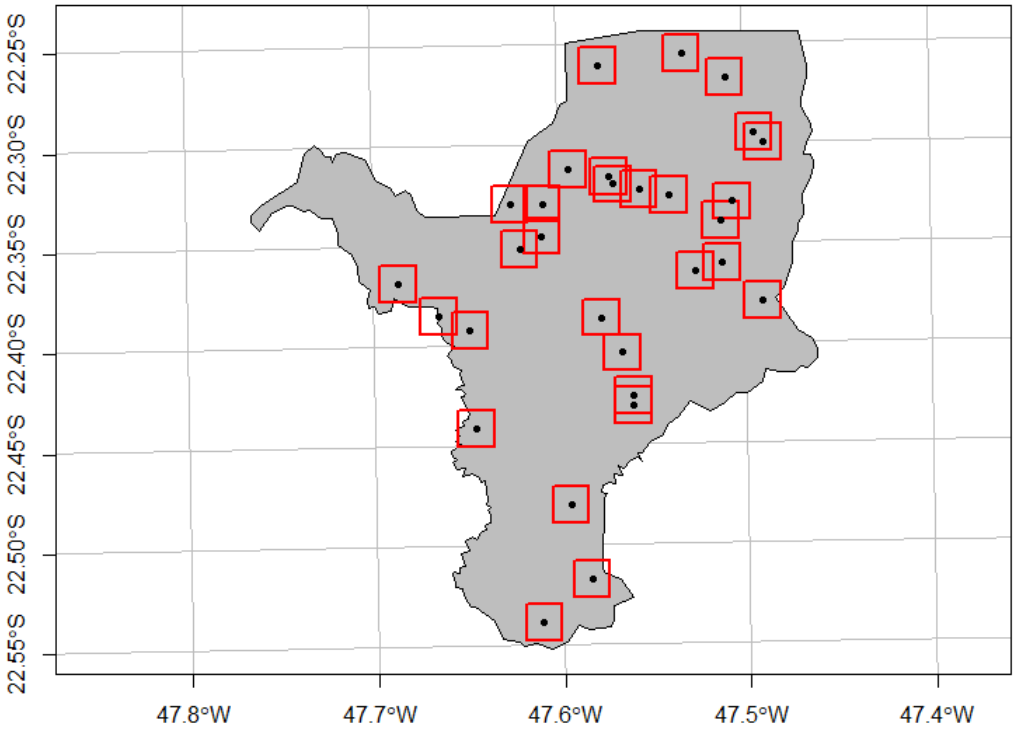


Figura 15.41: Buffers quadrados de 1000 metros para os 30 pontos aleatórios no município de Rio Claro/SP.

Polígono convexo

Uma análise bastante comum, principalmente realizada pela IUCN, é a criação de polígonos convexos, para definir a extensão de ocorrência de uma espécie (*Extent of occurrence* - EOO). Nesse sentido, essa operação liga os pontos externos de um conjunto de pontos e cria um polígono a partir deles. Podemos criar esse polígono com a função `sf::st_convex_hull()`. Um único passo que precisamos adiantar é utilizar a função `sf::st_union()` para unir todos os pontos e criar um objeto `sf` do tipo `MULTIPOINT`, já iremos explicar com mais detalhes adiante. Vamos utilizar os pontos aleatórios que criamos anteriormente para criar o polígono convexo (Figura 15.42).

```
## Polígono convexo
geo_vetor_rio_claro_sirgas2000_utm23s_convexo <- geo_vetor_rio_claro_
sirgas2000_utm23s_pontos_aleatorios %>%
    sf::st_union() %>%
    sf::st_convex_hull()

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray",
     main = NA, axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_convexo, col = NA,
     lwd = 2, border = "red", add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios,
     pch = 20, cex = 1, add = TRUE)
```

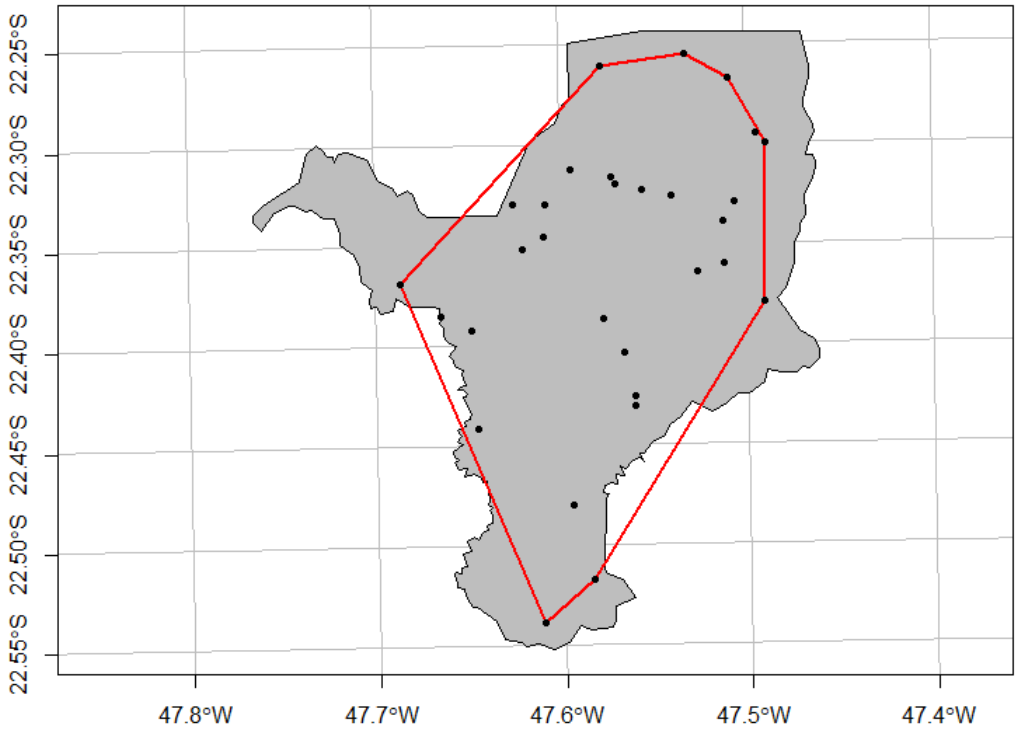


Figura 15.42: Polígono convexo para os 30 pontos criados aleatoriamente para Rio Claro/SP.

Polígonos de Voronoi

Uma outra forma de criar polígonos para resumir dados espaciais é através dos Polígonos de Voronoi ou Diagrama de Voronoi. Nele, polígonos irregulares são criados a partir da proximidade de pontos, de modo a estimar uma área de abrangência no entorno dos mesmos (Okabe 2000). Esses polígonos podem ser criados com a função `sf::st_voronoi()`, mas precisamos novamente utilizar a função `sf::st_union()` para unir todos os pontos e criar um objeto `sf` do tipo `MULTIPOINT`. Vamos utilizar os pontos aleatórios que criamos anteriormente para criar o polígono de Voronoi (Figura 15.43).

```
## Polígonos de Voronoi
geo_vetor_rio_claro_sirgas2000_utm23s_voronoi <- geo_vetor_rio_claro_
sirgas2000_utm23s_pontos_aleatorios %>%
    sf::st_union() %>%
    sf::st_voronoi()

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray", main = NA,
     axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_voronoi, col = NA, lwd = 2,
     border = "red", add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios, pch = 20,
     cex = 1, add = TRUE)
```

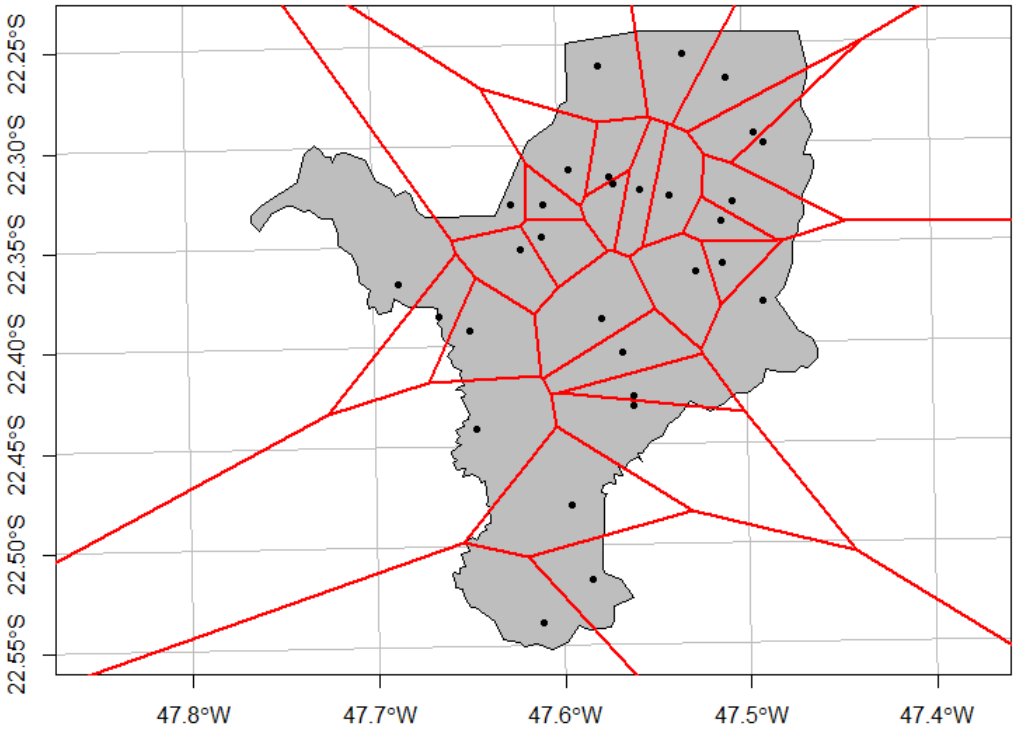


Figura 15.43: Polígonos de Voronoi para os 30 pontos criados aleatoriamente para Rio Claro/SP.

Quadrículas e hexágonos

Muitas vezes precisamos criar unidades espaciais idênticas e igualmente espaçadas para resumir informações dispersas por toda a nossa área de estudo. Uma prática muito comum é a criação de um gride de pontos ou quadrículas em toda a área de estudo, e depois utilizar essas geometrias para associar ou resumir informações espacializadas, como a IUCN utiliza para a análise de área de ocupação (*Area of occupancy* - AOO). Além das quadrículas, uma outra geometria que se tornou bastante comum para as finalidades descritas, é a criação de hexágonos, que além de serem mais esteticamente atraentes, possuem uma explicação matemática de sua melhor funcionalidade para análises espaciais em Ecologia (Birch et al. 2007).

A função utilizada para criar esses grides é a `sf::st_make_grid()`, que requer pelo menos dois argumentos: `x` uma geometria de entrada e o `cellsize` indicando o tamanho do gride a ser criado, fornecido nas unidades do CRS da geometria de entrada. Há diversos outros argumentos, mas os mais importantes são o `square` que definirá se o gride será de quadriculas ou de hexágonos, e o `what` que definirá se geraremos polígonos, cantos ou centroides.

Em nosso exemplo, vamos criar quadrículas e hexágonos de 2000 metros (i.e., 4000000 metros quadrados) para o município de Rio Claro/SP (Figura 15.44 e Figura 15.45). Podemos ainda utilizar as funções de filtros espaciais (Tabela 15.12) para definir como selecionaremos esses elementos para a área de estudo. Aqui utilizamos a função `sf::st_intersects()`.

```
## Quadrículas
geo_vetor_rio_claro_sirgas2000_utm23s_grid <- sf::st_make_grid(
    x = geo_vetor_rio_claro_sirgas2000_utm23s, cellsize = 2000,
    what = "polygons") %>%
    sf::st_as_sf() %>%
    dplyr::filter(sf::st_intersects(x = .,
```

```
                    y = geo_vetor_rio_claro_sirgas2000_utm23s,
                    sparse = FALSE))

## Centroides das quadrículas
geo_vetor_rio_claro_sirgas2000_utm23s_grid_cent <- geo_vetor_rio_claro_
sirgas2000_utm23s %>%
    sf::st_make_grid(cellsize = 2000, what = "centers") %>%
    sf::st_as_sf() %>%
    dplyr::filter(sf::st_intersects(x = .,
           y = sf::st_union(geo_vetor_rio_claro_sirgas2000_utm23s_grid),
           sparse = FALSE))

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray", main = NA,
     axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_grid, col = NA,
     border = "red", lwd = 2, add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_grid_cent, pch = 20,
     add = TRUE)
```

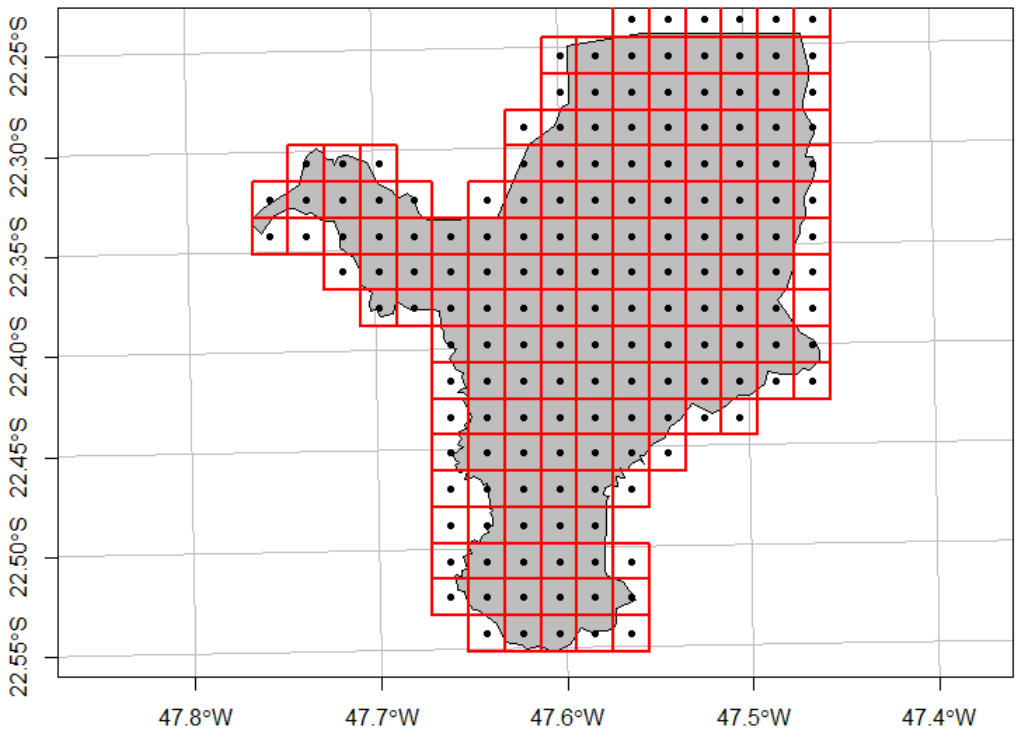


Figura 15.44: Quadrículas de 2000 metros de arestas e centroides para Rio Claro/SP.

```
## Hexágonos
geo_vetor_rio_claro_sirgas2000_utm23s_hex <-
                geo_vetor_rio_claro_sirgas2000_utm23s %>%
                sf::st_make_grid(cellsize = 2000, square = FALSE) %>%
                sf::st_as_sf() %>%
                dplyr::filter(sf::st_intersects(x = .,
```

```
                                y = geo_vetor_rio_claro_sirgas2000_utm23s,
                                sparse = FALSE))

## Centroides de hexágonos
geo_vetor_rio_claro_sirgas2000_utm23s_hex_cent <-
    geo_vetor_rio_claro_sirgas2000_utm23s %>%
    sf::st_make_grid(cellsize = 2000, square = FALSE,
    what = "centers") %>%
    sf::st_as_sf() %>%
    dplyr::filter(sf::st_intersects(x = .,
             y = sf::st_union(geo_vetor_rio_claro_sirgas2000_utm23s_hex),
             sparse = FALSE))

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray", main = NA,
     axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_hex, col = NA, border = "red",
     lwd = 2, add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_hex_cent, pch = 20,
     add = TRUE)
```

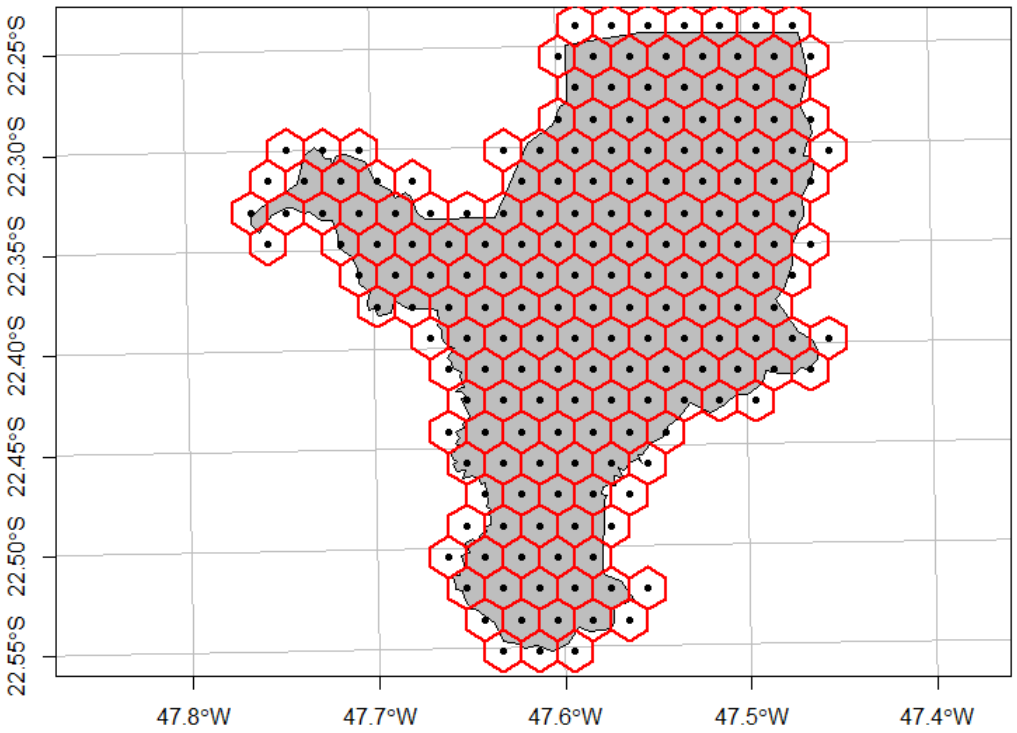


Figura 15.45: Hexágonos equivalentes a quadrículas de 2000 metros de arestas e centroides para Rio Claro/SP.

União ("dissolver")

Como vimos, na agregação por atributos podemos dissolver as geometrias de polígonos do mesmo grupo pelos valores da tabela de atributos, onde, naquele exemplo, contabilizamos quantas nascentes haviam por polígono de cobertura da terra para o município de Rio Claro/SP (Figura 15.29).

Nesta seção, vamos utilizar a função `sf::st_union()` para unir diversas feições em uma só, dissolvendo os limites entre elas. Vamos utilizar de exemplo os buffers que criamos a partir dos 30 pontos aleatórios (Figura 15.46).

```
## União
geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer_uniao <-
sf::st_union(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer)

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray", main = NA,
      axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer_uniao
      , col = adjustcolor("blue", .1), add = TRUE)
```

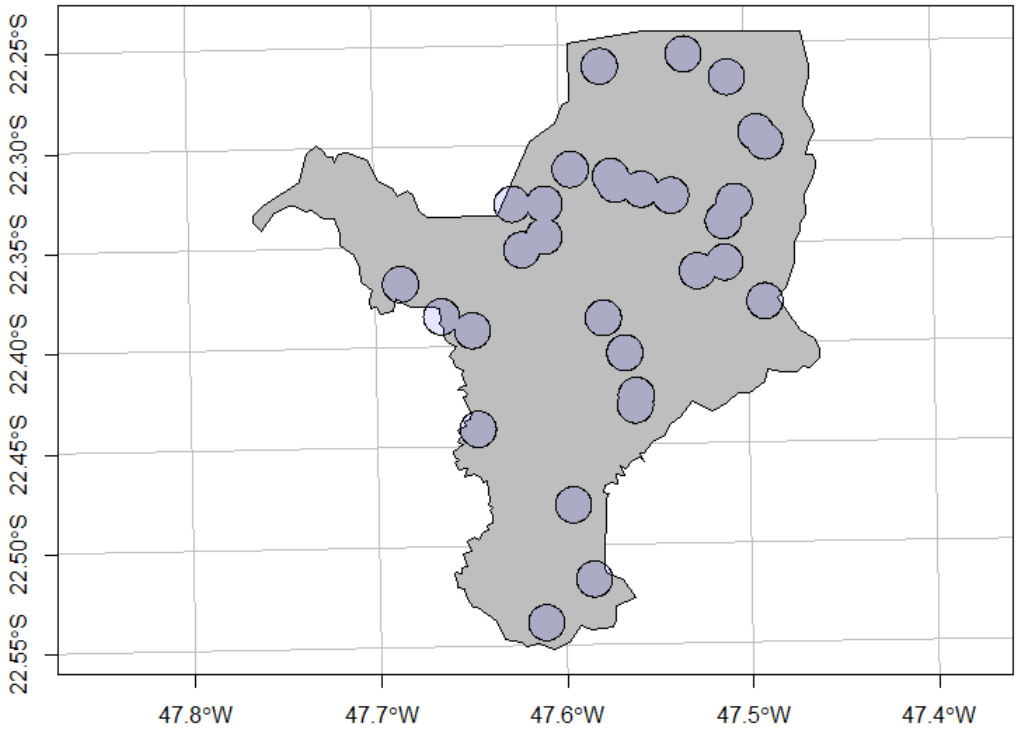


Figura 15.46: União - dissolução - dos buffers criados a partir dos 30 pontos aleatórios para Rio Claro/SP.

Recortar e apagar ("clip" e "erase")

O recorte realiza um subconjunto espacial envolvendo dois objetos geoespaciais. O recorte é aplicado somente a linhas e polígonos, ou seja, usaremos linhas e polígonos para recortar linhas ou polígonos. Esse recorte pode ser realizado de três formas: i) intersecção (subconjunto das geometrias sobrepostas entre os dois objetos), ii) diferença (subconjunto das geometrias do primeiro objeto sem sobreposição com o segundo objeto), e iii) diferença simétrica (apenas as geometrias não sobrepostas entre os dois objetos). Respectivamente para cada uma dessas operações temos funções específicas: `sf::st_intersection()`, `sf::st_difference()` e `sf::st_sym_difference()`.

Para nosso exemplo, faremos o recorte da hidrografia em relação aos buffers criados e unidos para os 30 pontos aleatórios em Rio Claro/SP. Primeiramente, faremos o recorte para dentro dos buffers com a função `sf::st_intersection()` (Figura 15.47).

```
## Recorte - intersecção
geo_vetor_hidrografia_interseccao <- sf::st_intersection(
                     x = geo_vetor_hidrografia,
y = geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer_uniao)
## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray", main = NA,
     axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer_uniao
     , col = adjustcolor("blue", .1), add = TRUE)
plot(geo_vetor_hidrografia_interseccao$geometry, col = "blue",
     add = TRUE)
```

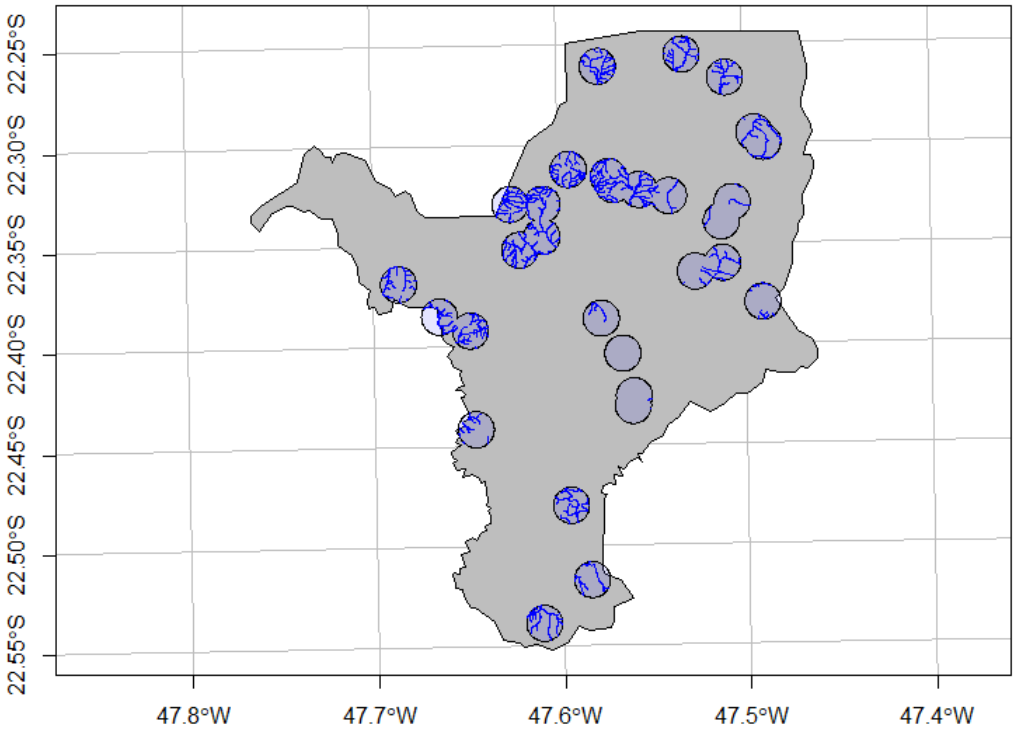


Figura 15.47: Recorte da hidrografia para dentro dos buffers dos 30 pontos aleatórios para Rio Claro/SP.

Para nosso segundo exemplo, realizamos o recorte da hidrografia em relação aos buffers, mas agora para fora dos buffers utilizando a função `sf::st_difference()` (Figura 15.48), que seria semelhando a operação de apagar ("erase").

```
## Recorte - diferença
geo_vetor_hidrografia_diferenca <- sf::st_difference(
 x = geo_vetor_hidrografia,
y = geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer_uniao)

## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray", main = NA,
     axes = TRUE, graticule = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s_pontos_aleatorios_buffer_uniao
     , col = adjustcolor("blue", .1), add = TRUE)
plot(geo_vetor_hidrografia_diferenca$geometry, col = "blue", add = TRUE)
```

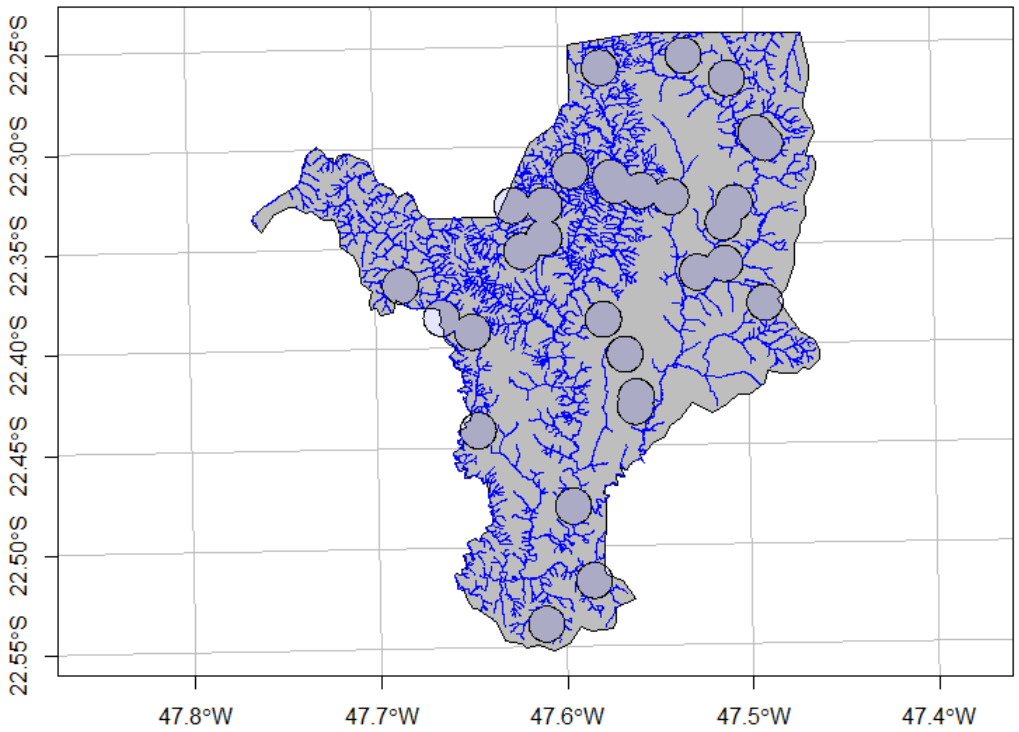


Figura 15.48: Recorte da hidrografia para fora dos buffers dos 30 pontos aleatórios para Rio Claro/SP.

Transformações de tipo

Esse tópico possui muitas funcionalidades, que são exploradas no tópico "5.2.7 *Type transformations*" de Lovelace et al. (2019). Aqui, nosso interesse principal é em relação à transformação dos tipos de objetos geoespaciais da classe `sf`: `MULTIPOINT`, `MULTILINESTRING` e `MULTIPOLYGON`, para `POINT`, `LINESTRING` e `POLYGON`. Muitas vezes as feições de nossos objetos, i.e., as linhas da tabela de atributos estão agrupadas em apenas uma linha da tabela. Quando o objeto espacial está nesse formato, geralmente em alguma classe dessas (`MULTIPOINT`, `MULTILINESTRING` e `MULTIPOLYGON`), não temos como realizar operações espaciais ou geométricas para cada feição, e precisamos separá-las em linhas diferentes para que operações como o cálculo de comprimento ou área seja possível para cada feição.

Dessa forma, podemos utilizar a função `sf::st_cast()` para fazer essas transformações e atribuir cada feição a uma linha da tabela de atributos. Como exemplo, vamos separar os fragmentos de floresta e calcular a área para cada feição em hectares (Figura 15.49).

```
## Transformação de tipo
geo_vetor_cobertura_floresta_polygon <- geo_vetor_cobertura_floresta %>%
    sf::st_cast("POLYGON") %>%
    dplyr::mutate(area_ha = sf::st_area(.)/1e4 %>% round(2))
## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = "gray", main = NA,
     axes = TRUE, graticule = TRUE)
plot(geo_vetor_cobertura_floresta_polygon["area_ha"],
     col = viridis::viridis(100),  add = TRUE)
```

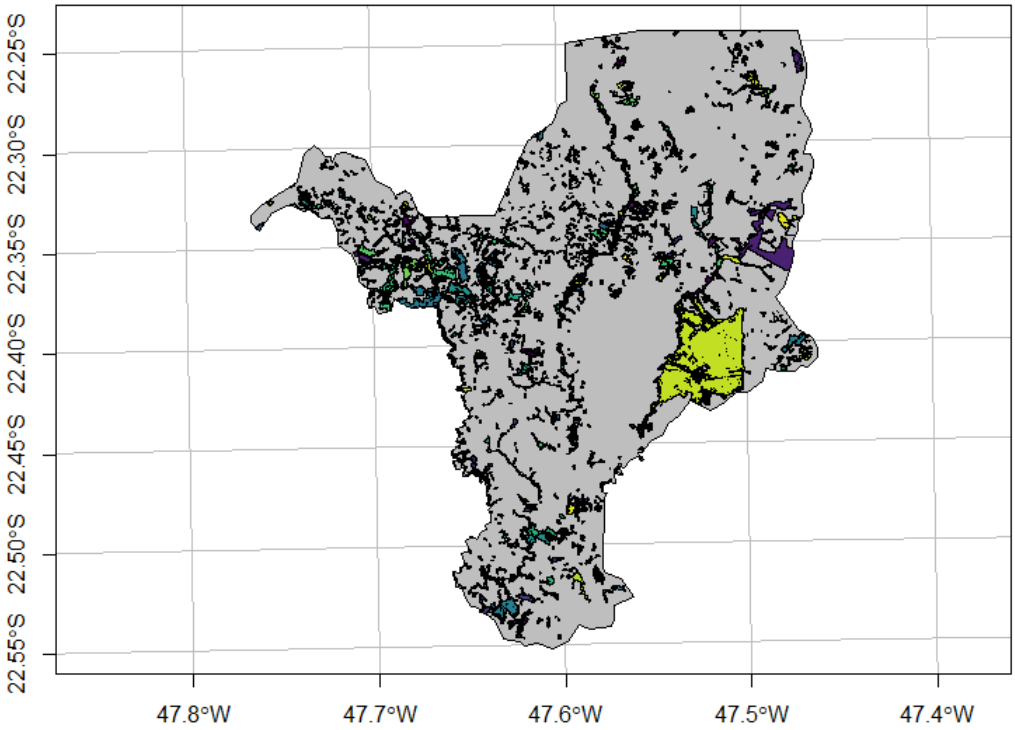


Figura 15.49: Transformação do vetor de florestas em `POLYGON` e cálculo da área para cada feição para Rio Claro/SP.

Raster

As operações geométricas em rasters envolvem mudar a posição, tamanho e número dos pixels e atribuir novos valores, geralmente aumentando ou diminuindo o tamanho desses pixels. Essas operações permitem alinhar rasters de diversas fontes, fazendo com que compartilhem uma correspondência entre seus pixels, permitindo que eles sejam processados todos juntos, ou simplesmente permite a realização de análises que demorariam muito, caso os rasters possuam um tamanho de pixel muito pequeno.

> **Importante**
> Essas operações funcionam para as três classes dos objetos raster: `RasterLayer`, `RasterBrick` e `RasterStack`.

Para exemplificar as operações geométricas com rasters, vamos utilizar os dados de elevação para o município de Rio Claro/SP e bioclimáticos para o mundo.

Agregação

Na agregação de rasters, aumentamos o tamanho dos pixels (diminuindo a resolução), agregando os valores dos pixels em um pixel maior. Podemos realizar essa operação com a função `raster::aggregate()`, que possui três argumentos: `x` corresponde ao objeto raster de entrada, `fact` é o fator de agregação e corresponde ao número que definirá o novo tamanho do pixel (e.g., se um raster tem resolução de 90 m, um fator de agregação de 10 fará com o novo raster tenha a resolução de 900 m), e `fun` é a função utilizada para realizar a agregação dos pixels (Figura 15.50).

Em nosso exemplo, vamos aumentar o tamanho dos pixels para 900 metros do raster de elevação para Rio Claro/SP.

```
## Agregação - aumentar o tamanho do pixel
geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media <- raster::aggre
            gate(x = geo_raster_srtm_rio_claro_sirgas2000_utm23s,
            fact = 10, fun = "mean")
geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media
#> class      : RasterLayer
#> dimensions : 40, 37, 1480  (nrow, ncol, ncell)
#> resolution : 900, 900  (x, y)
#> extent     : 214554.4, 247854.4, 7502625, 7538625  (xmin, xmax, ymin,
ymax)
#> crs        : +proj=utm +zone=23 +south +ellps=GRS80 +units=m +no_defs
#> source     : memory
#> names      : srtm_27_17
#> values     : 506.0024, 922.8709  (min, max)


## Plot
plot(geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media,
     col = viridis::viridis(10))
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = NA,
     border = "red", lwd = 2, add = TRUE)
```

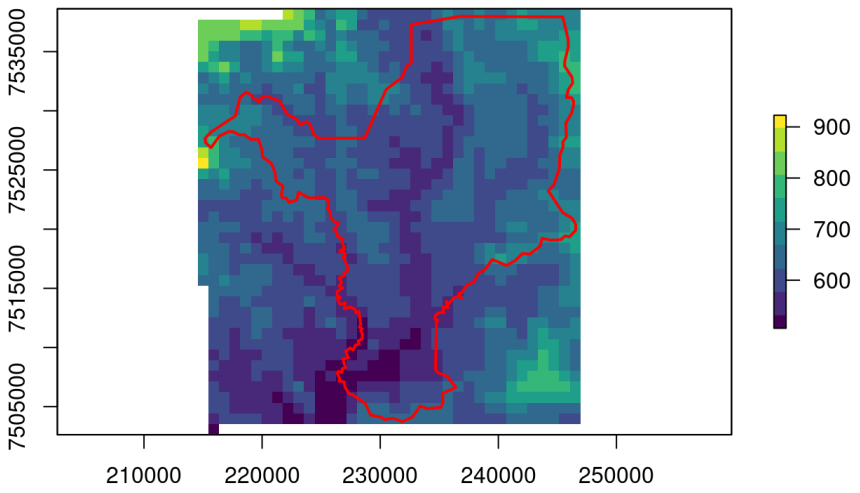


Figura 15.50: Agregação (aumento do pixel para 900 metros) utilizando a média para o raster de elevação para Rio Claro/SP.

Desagregação

De modo contrário, na desagregação de rasters, diminuímos o tamanho dos pixels (aumentando a resolução), preenchendo com novos valores. Podemos realizar essa operação com a função `raster::desaggregate()`, que assim como a função anterior, possui três argumentos: `x` corresponde ao objeto raster de entrada, `fact` é o fator de desagregação e corresponde ao número que definirá o novo tamanho do pixel (e.g., se um raster tem resolução de 90 m, um fator de desagregação

de 2 fará com que o novo raster tenha a resolução de 45 m), e `method` é a função utilizada para realizar a desagregação dos pixels (Figura 15.51).

Nesse exemplo, vamos diminuir o tamanho dos pixels para 45 metros do raster de elevação para Rio Claro/SP.

```
## Desagregação - diminuir o tamanho do pixel
geo_raster_srtm_rio_claro_desg_bil <- raster::disaggregate(
       x = geo_raster_srtm_rio_claro_sirgas2000_utm23s,
       fact = 2, method = "bilinear")
geo_raster_srtm_rio_claro_desg_bil
#> class      : RasterLayer
#> dimensions : 792, 728, 576576  (nrow, ncol, ncell)
#> resolution : 45, 45  (x, y)
#> extent     : 214554.4, 247314.4, 7502985, 7538625  (xmin, xmax, ymin,
ymax)
#> crs        : +proj=utm +zone=23 +south +ellps=GRS80 +units=m +no_defs
#> source     : memory
#> names      : srtm_27_17
#> values     : 493.7046, 986.5187  (min, max)
## Plot
plot(geo_raster_srtm_rio_claro_desg_bil, col = viridis::viridis(10))
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = NA,
     border = "red", lwd = 2, add = TRUE)
```

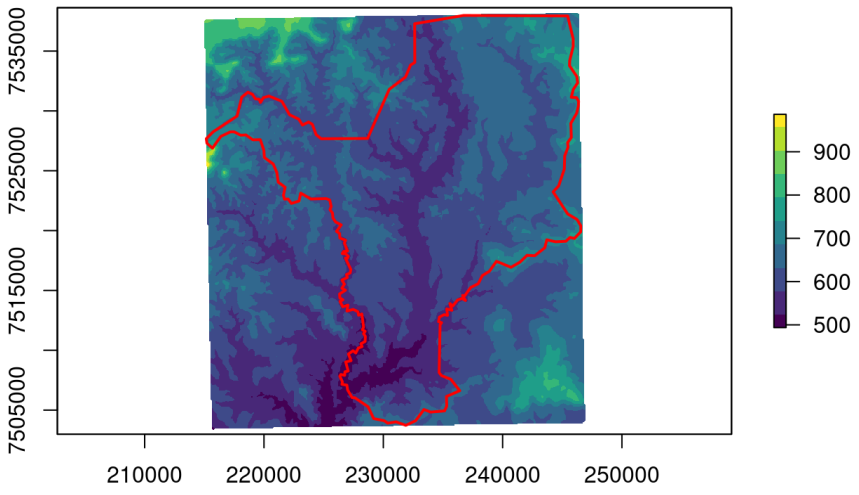


Figura 15.51: Desagregação (diminuição do pixel para 45 metros) utilizando o método bilinear para o raster de elevação para Rio Claro/SP.

Alinhamento de rasters

Muitas vezes queremos ir além de ajustar o tamanho do pixel, ajustando também a extensão, número e origem dos pixels para várias camadas rasters, principalmente se precisamos criar objetos das classes `RasterBrick` ou `RasterStack`. Dessa forma, podemos utilizar a função `raster::compareRaster()` para comparar os rasters em relação a extensão, número de linhas e colunas, projeção, resolução e origem (ou um subconjunto dessas comparações).

Podemos utilizar a função `raster::resample()` para fazer esse alinhamento, ou ainda a função `gdalUtils::align_rasters()`. Para a primeira função, os argumentos são `x` para o raster de entrada, `y` para o raster de alinhamento e `method` para o método utilizado no alinhamento. Para nosso exemplo, vamos ajustar uma camada bioclimática (`BIO01`) à camada de elevação para Rio Claro/SP (Figura 15.52).

```
## Reamostragem
geo_raster_bioclim_rc <- raster::resample(x = geo_raster_bioclim$bio01,
                                          y = geo_raster_srtm_rio_claro,
                                          method = "bilinear")
geo_raster_bioclim_rc
#> class      : RasterLayer
#> dimensions : 370, 364, 134680  (nrow, ncol, ncell)
#> resolution : 0.0008333333, 0.0008333333  (x, y)
#> extent     : -47.765, -47.46167, -22.55167, -22.24333  (xmin, xmax, ymin,
ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : bio01
#> values     : 19.8383, 20.60492  (min, max)

## Plot
plot(geo_raster_bioclim_rc$bio01, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

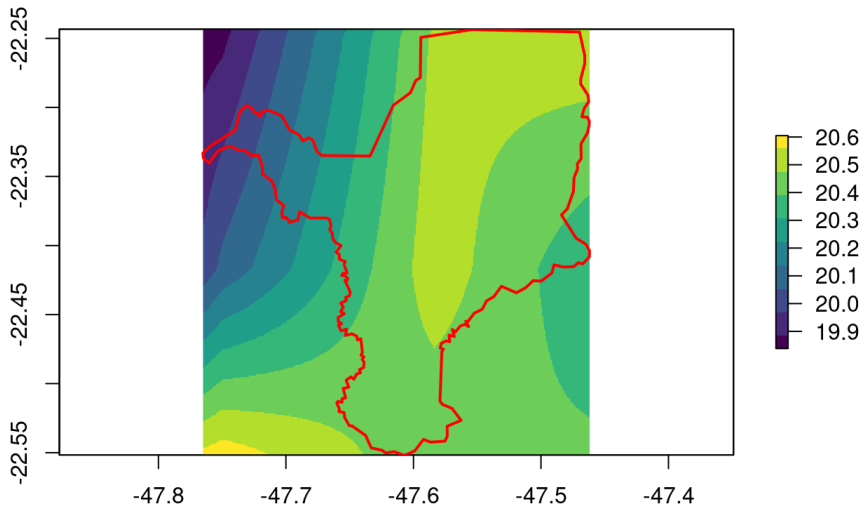


Figura 15.52: Reamostragem (alinhamento dos rasters) utilizando o método bilinear para alinhar o raster bioclimático (BIO01) ao de elevação para Rio Claro/SP.

Interações raster-vetor

Podemos fazer operações da interação entre objetos vetoriais e raster, como ajustes da extensão e limite do raster para vetores (corte e máscara), extração dos valores dos pixels para vetores (pontos,

linhas e polígonos), e estatísticas zonais dos valores dos pixels dos raster para um vetor (linhas e polígonos).

Cortes e máscaras

Muitas vezes precisamos ajustar o tamanho de um objeto raster a uma área menor de interesse, geralmente definido por um objeto vetorial. Para realizar essa operação, dispomos de duas funções: `raster::crop()` e `raster::mask()`.

> **Importante**
> É fundamental que ambos os objetos raster a ser reduzido e o vetor como molde precisam estar no mesmo CRS.

A função `raster::crop()` ajusta o raster à extensão do vetor. Como exemplo, vamos retomar o raster de elevação original baixado e importado anteriormente (Figura 15.14). Primeiramente, vamos usar a função `raster::crop()` para ajustar esse raster à extensão do limite do município de Rio Claro/SP (Figura 15.53).

```
## Crop - adjuste da extensão
geo_raster_srtm_rio_claro_crop <- raster::crop(geo_raster_srtm,
                                               geo_vetor_rio_claro)

## Plot
plot(geo_raster_srtm_rio_claro_crop, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

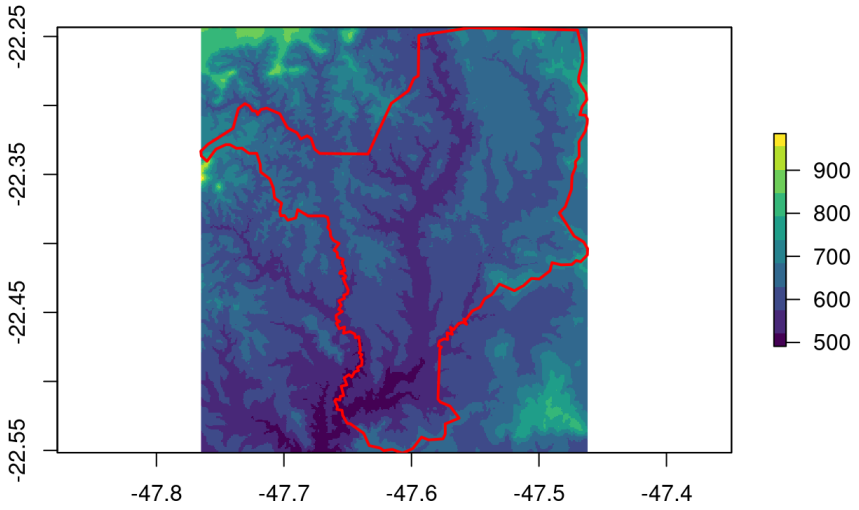


Figura 15.53: Ajuste da extensão do raster de elevação para a extensão de Rio Claro/SP.

Para ajustar o raster ao limite do município de Rio Claro/SP, vamos usar a função `raster::mask()`. É importante notar que essa função preenche com `NAs` os pixels que estão fora do limite do polígono e não ajusta a extensão (Figura 15.53).

```
## Mask - adjuste ao limite
geo_raster_srtm_rio_claro_mask <- raster::mask(geo_raster_srtm,
                                               geo_vetor_rio_claro)

## Plot
plot(geo_raster_srtm_rio_claro_mask, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

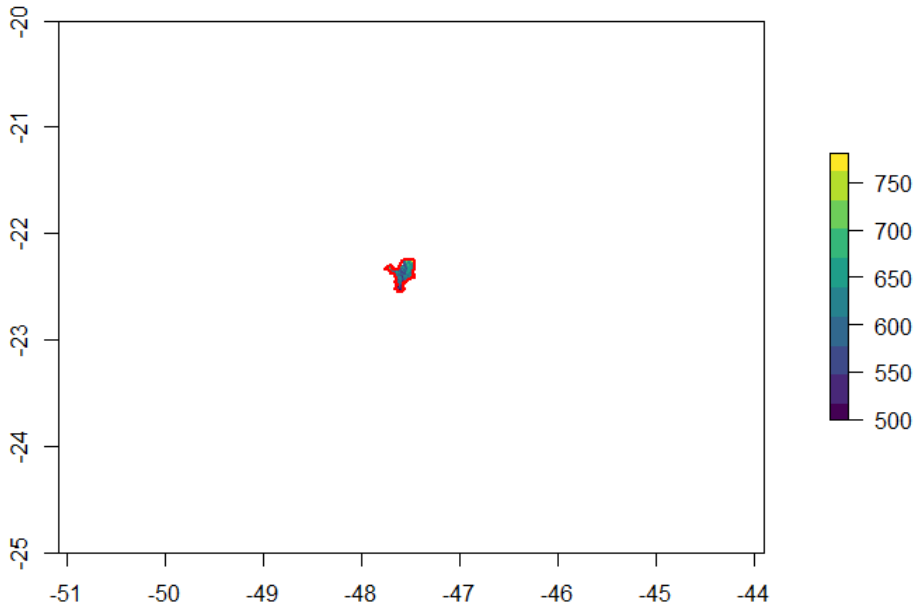


Figura 15.54: Ajuste do raster de elevação para o limite de Rio Claro/SP.

Para ajustar o raster à extensão e ao limite do município de Rio Claro/SP, precisamos utilizar conjuntamente as funções `raster::crop()` e `raster::mask()` (Figura 15.55).

```
## Crop e mask - ajuste da extensão e do limite
geo_raster_srtm_rio_claro_crop_mask <- geo_raster_srtm %>%
    raster::crop(geo_vetor_rio_claro) %>%
    raster::mask(geo_vetor_rio_claro)

## Plot
plot(geo_raster_srtm_rio_claro_crop_mask, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

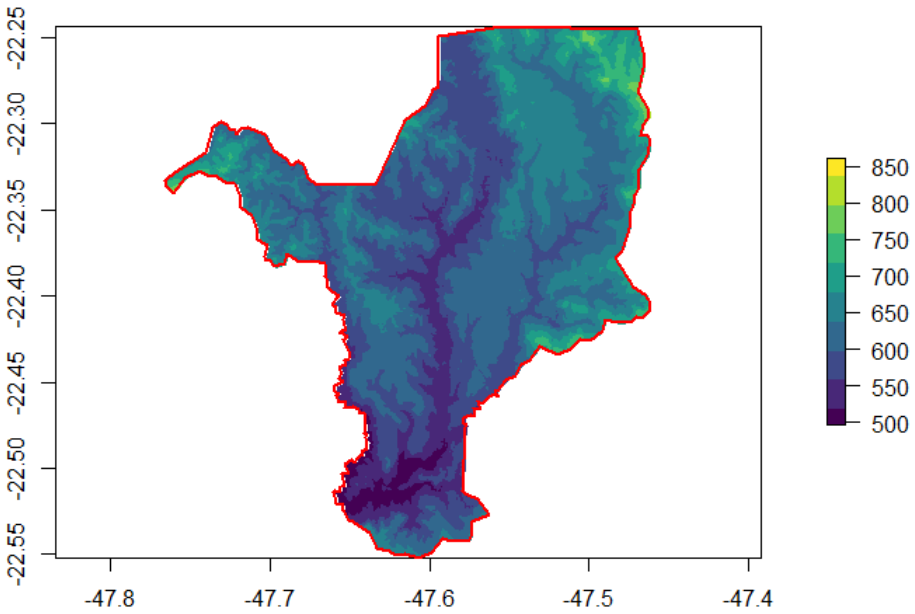


Figura 15.55: Ajuste da extensão e do limite do raster de elevação para Rio Claro/SP.

A função `raster::mask()` possui ainda um argumento chamado `inverse`, que cria uma máscara inversa ao limite, preenchendo com `NA` o pixels internos ao limite do polígono, como podemos ver para o raster de elevação e o limite de Rio Claro/SP (Figura 15.56).

```
## Crop e mask inversa - ajuste da extensão e do limite inverso
geo_raster_srtm_rio_claro_crop_mask_inv <- geo_raster_srtm %>%
    raster::crop(geo_vetor_rio_claro) %>%
    raster::mask(geo_vetor_rio_claro, inverse = TRUE)

## Plot
plot(geo_raster_srtm_rio_claro_crop_mask_inv, col = viridis::viridis(10))
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

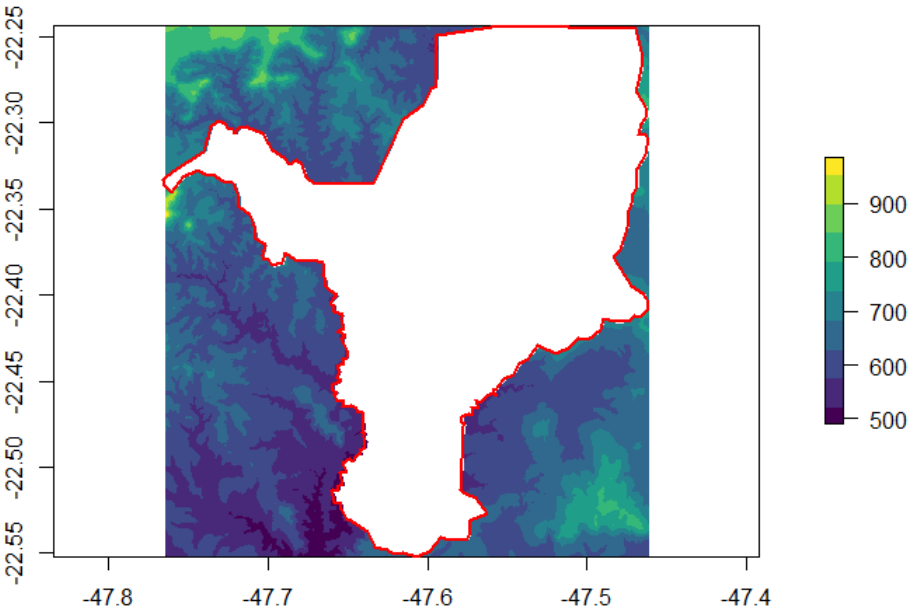


Figura 15.56: Ajuste da extensão e do limite externo do raster de elevação para Rio Claro/SP.

Extração

A interação entre raster-vetor de extração é o processo que identifica e retorna valores associados de pixels de um raster com base em um objeto vetorial. É uma operação extremamente comum em análises espaciais, principalmente para associar valores de raster ambientais (contínuos ou categóricos) a pontos de ocorrência ou amostragem. Os valores retornados dependerão do tipo vetor (pontos, linhas ou polígonos) e de argumentos da função `raster::extract()` que alteram o funcionamento da extração.

Em nosso exemplo, vamos extrair os valores do raster de elevação para as nascentes do município de Rio Claro/SP (Figura 15.57).

```
## Extração
geo_vetor_nascentes_ele <- geo_vetor_nascentes %>%
    dplyr::mutate(elev = raster::extract(
    x = geo_raster_srtm_rio_claro_sirgas2000_utm23s, y = .))
```

```
## Plot
plot(geo_vetor_nascentes_ele["elev"],
     pch = 20, main = NA, axes = TRUE, graticule = TRUE)
```

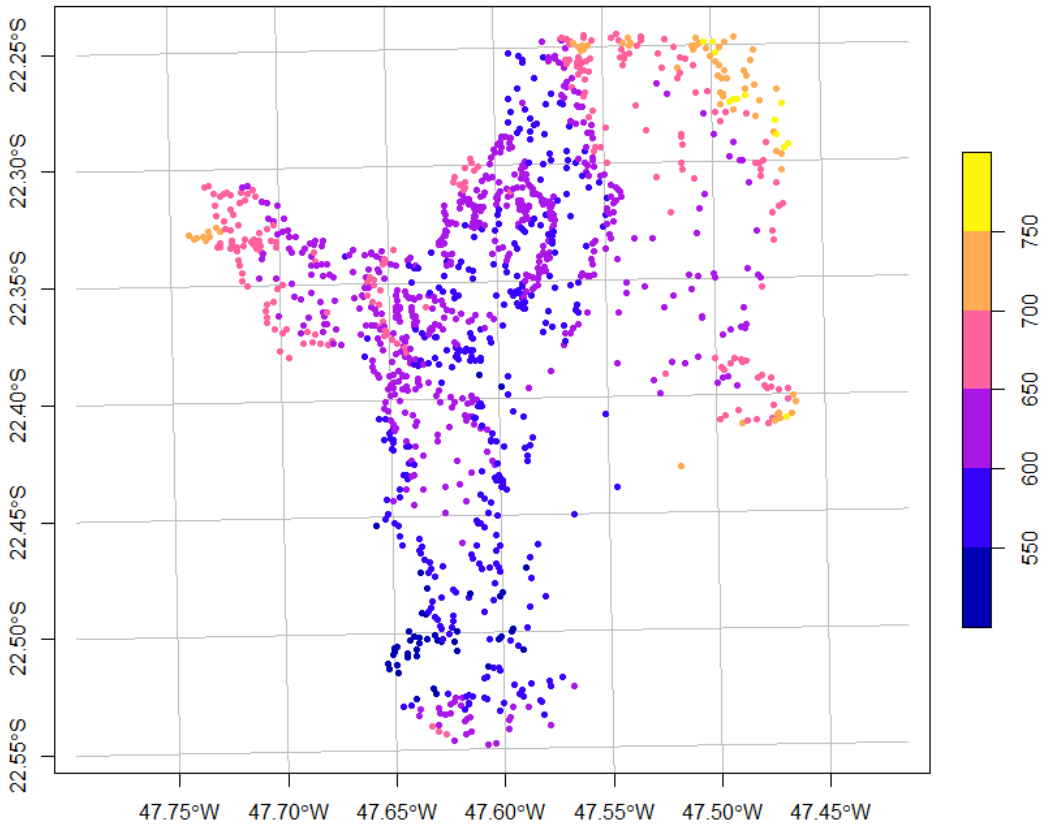


Figura 15.57: Extração dos valores de elevação para as nascentes de Rio Claro/SP.

Além da extração dos valores totais, podemos resumir os valores dos pixels com a mesma operação de extração, utilizando ainda a função `raster::extract()`, mas utilizando uma outra função para resumir os valores dos pixels para um polígono, operação também denominada de estatística zonal (agora para vetores). Já vimos que ela pode ser realizada entre rasters na seção de operações zonais, mas aqui a realizaremos para rasters e vetores.

Para o exemplo, vamos calcular a elevação média dos valores para os hexágonos que criamos para o limite de Rio Claro/SP(Figura 15.58).

```
## Extração - estatística por zonas
geo_vetor_rio_claro_sirgas2000_utm23s_hex_alt <-
    geo_vetor_rio_claro_sirgas2000_utm23s_hex %>%
    dplyr::mutate(elev_mean = raster::extract(
                  x = geo_raster_srtm_rio_claro_sirgas2000_utm23s,
                  y = geo_vetor_rio_claro_sirgas2000_utm23s_hex,
                  fun = mean, na.rm = TRUE))
## Plot
plot(geo_vetor_rio_claro_sirgas2000_utm23s_hex_alt["elev_mean"],
     pch = 20, main = NA, axes = TRUE, graticule = TRUE)
```

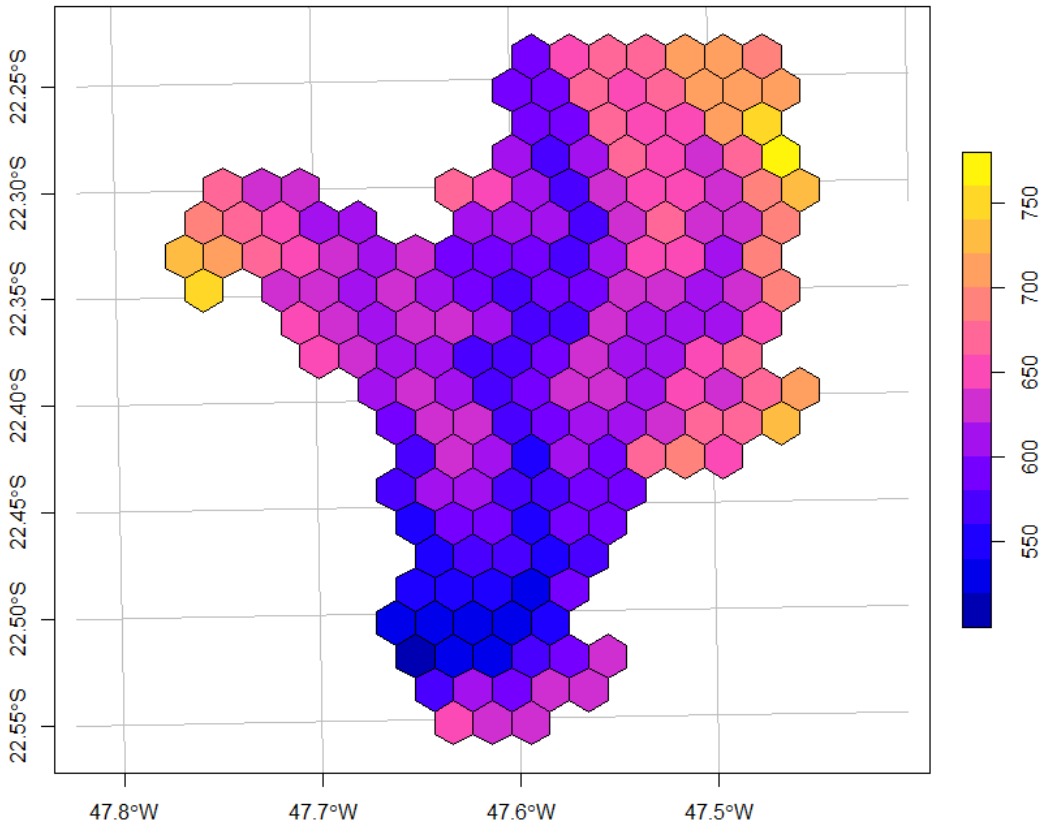


Figura 15.58: Extração dos valores de elevação e resumo pela média para os hexágonos de Rio Claro/SP.

Conversões raster-vetor

Por fim, podemos ainda fazer operações de conversão entre objetos vetoriais para raster e vice-versa. Nessas operações, podemos resumir ou transformar objetos vetoriais (pontos, linhas ou polígonos) para rasters, escolhendo um raster previamente existente, processo denominado rasterização. Também podemos realizar o processo inverso, i.e., transformar o raster em um vetor, podendo esse vetor ser pontos, linhas ou polígonos, operação chamada de vetorização.

Rasterização

A conversão de vetor para raster pode ser realizada de pontos, linhas ou polígonos para rasters. Nesse processo, podemos utilizar uma função para resumir os dados pontuais para os pixels do raster que criaremos. Para essa operação, podemos utilizar a função `raster::rasterize()`, com o argumento `x` sendo o vetor de entrada, `y` o raster base, `field` a coluna ou campo da tabela de atributos do objeto vetorial para os quais os valores serão utilizados e `fun` a função utilizada para agregação dos dados.

Aqui, vamos contabilizar a quantidade de nascentes por pixel, utilizando como base o raster para o qual mudamos a resolução para 900 metros (Figura 15.59).

```
## Rasterizar pontos
geo_vetor_nascentes_rasterizacao <- raster::rasterize(
          x = geo_vetor_nascentes,
          y = geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media,
          field = 1, fun = "count")
```

```
## Plot
plot(geo_vetor_nascentes_rasterizacao, col = viridis::viridis(10))
plot(geo_vetor_nascentes$geometry, pch = 20, cex = .5,
     col = adjustcolor("gray", .5), add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = NA,
     border = "red", lwd = 2, add = TRUE)
```

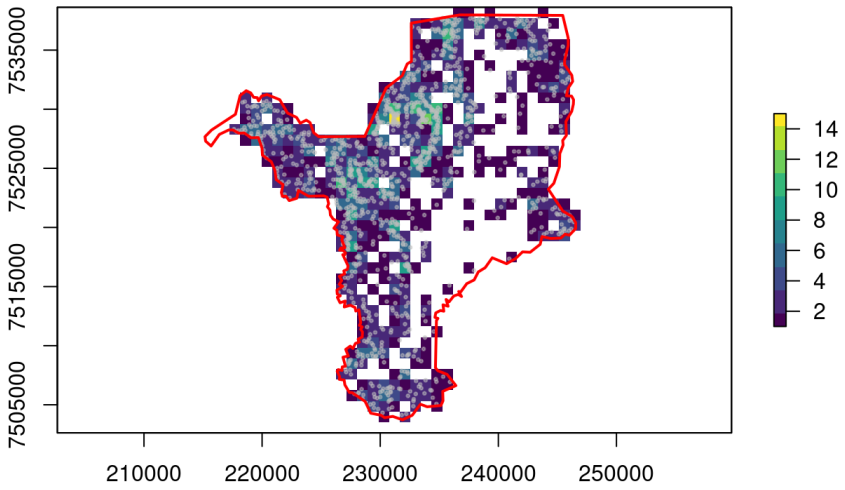


Figura 15.59: Rasterização das nascentes, com a operação de contabilização de pontos para Rio Claro/SP.

Além de pontos, podemos também rasterizar linhas. Aqui vamos contabilizar as linhas da hidrografia simplificada para Rio Claro/SP (Figura 15.60).

```
## Rasterizar linhas
geo_vetor_hidrografia_rasterizacao <- raster::rasterize(
    x = geo_vetor_hidrografia_simplificado,
    y = geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media,
    field = 1, fun = "count")

## Plot
plot(geo_vetor_hidrografia_rasterizacao, col = viridis::viridis(10))
plot(geo_vetor_hidrografia_simplificado$geom, col = "gray", add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = NA,
     border = "red", lwd = 2, add = TRUE)
```

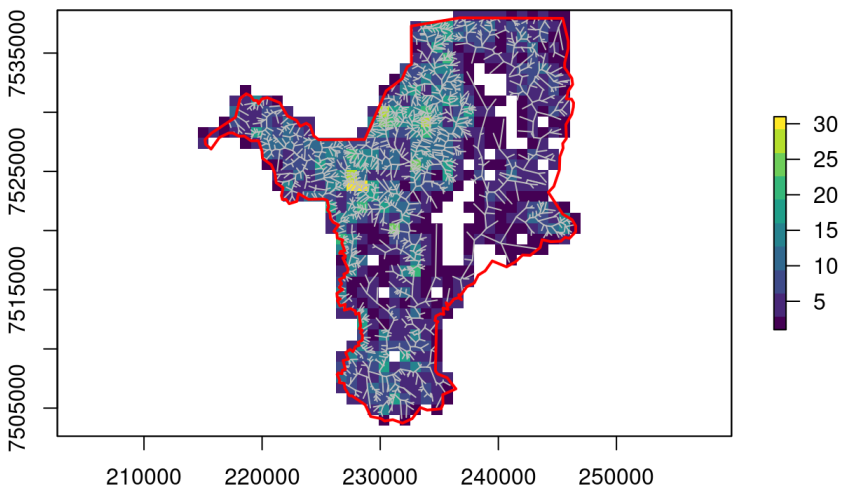


Figura 15.60: Rasterização da hidrografia, com a operação de contabilização de linhas para Rio Claro/SP.

Podemos ainda rasterizar polígonos, de modo que cada pixel do raster a ser criado receberá o valor da tabela de atributos, ou uma análise pelo vizinho mais próximo no caso de um campo categórico, como a cobertura da terra, que também vai depender da resolução do raster base e do tamanho da feição do polígono. Para nosso exemplo, antes de criar o raster vamos transforma a coluna de classe de cobertura da terra em `factor` (Figura 15.61). Entretanto, essa operação de rasterização tende a demorar muito no caso de polígonos muito detalhados ou um raster com pixels muito pequenos, sendo que dois pacotes aceleram esse processamento (`fasterize` e `gdalUtils`), com suas respectivas funções: `fasterize::fasterize()` e `gdalUtils::gdal_rasterize()`.

```
## Rasterizar polígonos
geo_vetor_cobertura_rasterizacao <- geo_vetor_cobertura %>%
    dplyr::mutate(classe = as.factor(CLASSE_USO)) %>%
    raster::rasterize(x = .,
            y = geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media,
            field = "classe")

## Plot
plot(geo_vetor_cobertura_rasterizacao, col = viridis::viridis(10))
plot(geo_vetor_cobertura$geom, add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = NA,
     border = "red", lwd = 2, add = TRUE)
```

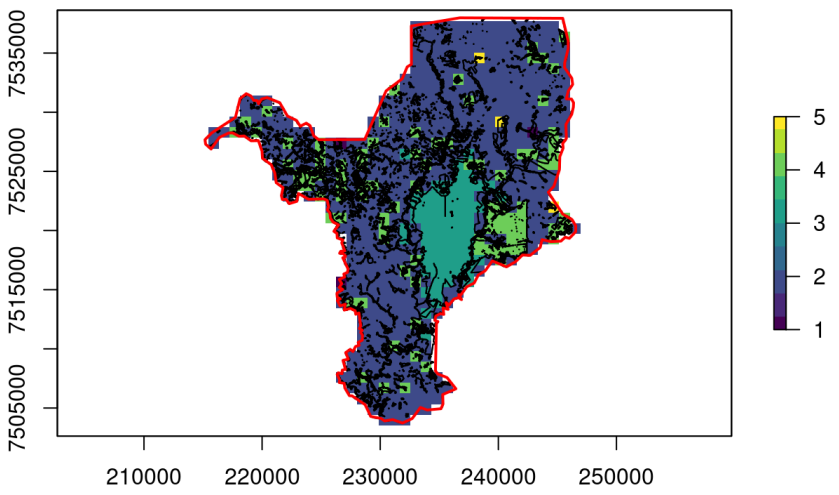


Figura 15.61: Rasterização da cobertura da terra para Rio Claro/SP.

Vetorização

A operação inversa à rasterização é a vetorização, na qual convertemos um raster em um vetor, sendo que esse vetor receberá os valores dos pixels. O vetor em questão pode ser pontos (geralmente um gride de pontos), linhas (geralmente isolinhas ou linhas de contorno), ou polígonos (podendo esses polígonos ser ou não dissolvidos pelos valores dos pixels). Existem funções específicas para cada uma dessas conversões, sendo elas: `raster::rasterToPoints()`, `raster::rasterToContour()` e `raster::rasterToPolygons()`, respectivamente. Para a última função, ainda dispomos de uma alternativa mais veloz `spex::polygonize()`.

Em nosso exemplo, vamos vetorizar o raster de elevação para Rio Claro/SP, criando um gride de pontos, sendo os pontos os centroides de cada pixels (Figura 15.62).

```
## Vetorização de pontos
geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media_pontos <-
           raster::rasterToPoints(
           geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media,
           spatial = TRUE) %>%
           sf::st_as_sf()

## Plot
plot(geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media,
     col = viridis::viridis(10, alpha = .8))
plot(geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media_pontos,
     pch = 20, cex = .7, main = FALSE, add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom, col = NA,
     border = "red", lwd = 2, add = TRUE)
```

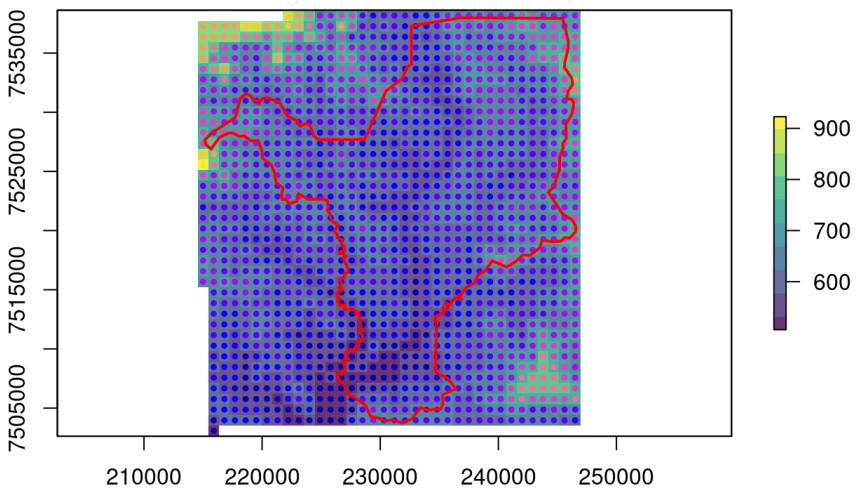


Figura 15.62: Vetorização do raster de elevação criando um gride de pontos para Rio Claro/SP.

Neste outro exemplo, vamos vetorizar o raster de elevação para Rio Claro/SP novamente, mas agora criando isolinhas (Figura 15.63).

```
## Vetorização de linhas
geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media_linhas <-
    raster::rasterToContour(
    x = geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media) %>%
    sf::st_as_sf()

## Plot
plot(geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media,
     col = viridis::viridis(10, alpha = .8))
contour(geo_raster_srtm_rio_claro_sirgas2000_utm23s_agre_media,
        labcex = 1, main = FALSE, add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom,
     col = NA, border = "red", lwd = 2, add = TRUE)
```

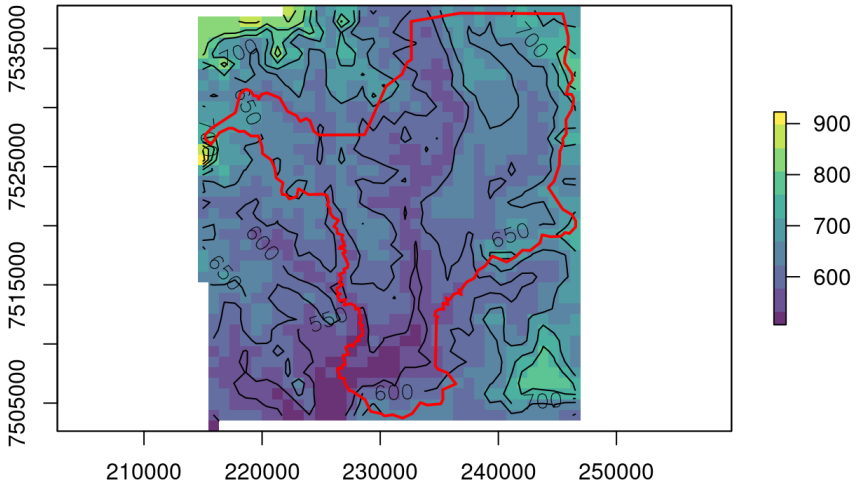


Figura 15.63: Vetorização do raster de elevação criando isolinhas para Rio Claro/SP.

Por fim, vamos vetorizar o raster de cobertura da terra criado anteriormente para Rio Claro/SP, criando polígonos não dissolvidos e dissolvidos (Figura 15.64).

```
## Vetorização de polígonos
geo_vetor_cobertura_rasterizacao_poligonos <-
    raster::rasterToPolygons(geo_vetor_cobertura_rasterizacao) %>%
    sf::st_as_sf()

## Vetorização de polígonos dissolvendo
geo_vetor_cobertura_rasterizacao_poligonos_dissolvidos <-
    raster::rasterToPolygons(geo_vetor_cobertura_rasterizacao,
                             dissolve = TRUE) %>%
    sf::st_as_sf()

## Plot
old_par <- par(mfrow = c(1, 2))
plot(geo_vetor_cobertura_rasterizacao, col = viridis::viridis(10))
plot(geo_vetor_cobertura_rasterizacao_poligonos$geometry,
     col = NA, border = "gray", lwd = 1, main = FALSE, add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom,
     col = NA, border = "red", lwd = 2, add = TRUE)

plot(geo_vetor_cobertura_rasterizacao, col = viridis::viridis(10))
plot(geo_vetor_cobertura_rasterizacao_poligonos_dissolvidos$geometry,
     col = NA, border = "gray", lwd = 1, main = FALSE, add = TRUE)
plot(geo_vetor_rio_claro_sirgas2000_utm23s$geom,
     col = NA, border = "red", lwd = 2, add = TRUE)
par(old_par)
```

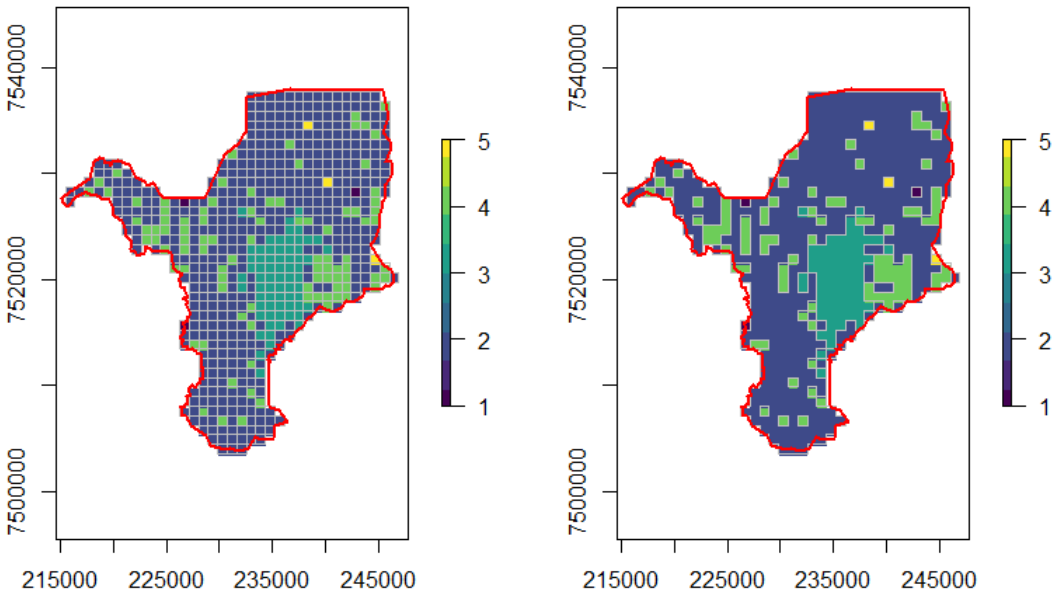


Figura 15.64: Vetorização do raster de cobertura da terra para Rio Claro/SP, não dissolvendo e dissolvendos os polígonos gerados.

15.10 Visualização de dados geoespaciais

Um dos pontos finais de toda a análise envolvendo a manipulação de dados geoespaciais será a apresentação de um mapa com as informações de interesse geoespacializadas. Mas antes, é necessário ter conhecimento de alguns dos elementos principais para a composição de um mapa relativamente bem informativo. Além disso, o R nos permite criar tipos diferentes de mapas: estáticos, animados e interativos. Os mais comuns são os estáticos, mas podemos por vezes melhorar a apresentação dos dados geoespaciais criando mapas animados e/ou interativos, com o auxílio de páginas web. Por fim, veremos as melhores formas de exportar mapas para diferentes formatos.

15.10.1 Principais elementos de um mapa

Um mapa pode ser composto de vários elementos, tendo estes o intuito de auxiliar a visualização e entendimento de seu conteúdo. Apesar disso, nem todos os elementos necessitam estar presentes em todos os *layouts* de mapas, sendo que os mesmos devem atendem à necessidade das representações, podendo ser muitas vezes omitidos ou outros podem ser acrescentados.

Os principais elementos de um mapa geralmente são compostos por:

1. Mapa principal (ocupando quase toda a área da figura)
2. Mapa secundário (geralmente muito menor que o mapa principal e com o intuito de mostrar a localização do mapa principal num contexto mais amplo, como país ou continente)
3. Título (para resumir o intuito do mapa)
4. Legenda (apresentando as informações detalhadas das classes ou escala de valores, geralmente identificando as cores e/ou texturas),
5. Barra de escala (representando a relação entre unidades do mapa e do mundo real)
6. Indicador de orientação (Norte) (indicando o norte geográfico, podendo ser representado por uma flecha, bússola ou compasso)
7. Gride de coordenadas (coordenadas presentes nas laterais)
8. Descrição do CRS (indicando qual o Sistema de Referência de Coordenadas)
9. Informações de origem (informações sobre a fonte dos dados representados no mapa)
10. Outros elementos auxiliares (como elementos textuais e figuras extras)

Podemos visualizar todos esses elementos resumidos na Figura 15.65.

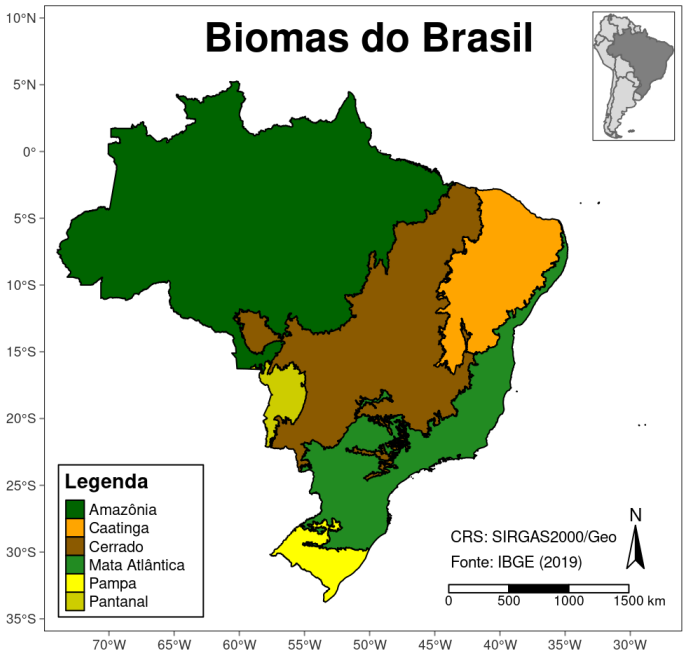


Figura 15.65: Principais elementos de um mapa.

15.10.2 Principais pacotes para a composição de mapas

Há uma grande quantidade de pacotes para a composição de mapas no R. Aqui listamos os principais (Tabela 15.13).

Tabela 15.13: Principais pacotes para composição de mapas no R.

Pacote	Descrição
`ggplot2`	Cria visualizações de dados elegantes usando a gramática de gráficos
`ggspatial`	Estrutura de dados espaciais para `ggplot2`
`ggmap`	Visualização espacial com `ggplot2`
`tmap`	Mapas temáticos
`leaflet`	Cria mapas da web interativos com a biblioteca JavaScript 'Leaflet'
`plotly`	Cria gráficos interativos da Web por meio de 'plotly.js'
`cartography`	Cartografia temática
`googleway`	Cartografia temática
`mapview`	Acessa APIs do Google Maps para recuperar dados e mapas de plotagem
`rasterVis`	Visualização interativa de dados espaciais em R
`cartogram`	Métodos de visualização para dados raster
`mapsf`	Crie cartogramas com R
`geogrid`	Transforme polígonos geoespaciais em grades regulares ou hexagonais
`geofacet`	`ggplot2` Utilitários de facetação para dados geoespaciais
`globe`	Mapas de visualização 2D e 3D da Terra, incluindo as linhas da costa
`linemap`	Mapas de linhas

15.10.3 Mapas estáticos

Mapas estáticos são mapas simples e fixos para visualização de dados, sendo o tipo mais comum de saída visual. No início da composição de mapas no R, esse era o único tipo de mapa que a linguagem permitia produzir, principalmente utilizando o pacote `sp` (Pebesma & Bivand 2005). No entanto, com o advento de ferramentas de visualização dinâmicas no R, como componentes HTML, os mapas puderam ser compostos de forma dinâmica (animados e interativos).

Neste tópico abordaremos funções simples para composição de mapas estáticos, como o `plot()`, além de pacotes para composição de mapas mais elaborados, como os pacotes `ggplot2` (Wickham et al. 2020) e `tmap` (Tennekes 2021).

Função `plot()`

A função genérica `plot()` é a maneira mais rápida de compor mapas estáticos utilizando objetos geoespaciais vetoriais e raster, funcionando para ambos os pacotes que apresentamos anteriormente (`sf` e `raster`). Apesar da simplicidade, essa função geralmente tende a criar mapas com relativa velocidade, nos auxiliando principalmente em fases iniciais de desenvolvimento de um projeto. Essa função oferece dezenas de argumentos em R Base, permitindo alguns ajustes limitados, com resultados bastante interessantes.

Como dito anteriormente, a função `plot()` vai funcionar diferentemente dependendo da classe do objeto geoespacial. Para objetos geoespaciais `sf`, a função vai plotar um mapa para cada coluna da tabela de atributos. Vamos usar de exemplo nosso mapa de biomas mostrado com os principais elementos de um mapa, podendo inclusive selecionar apenas a coluna de características geoespaciais (`geom`).

Primeiramente, vamos fazer o download dos dados de limites de biomas, retirando os sistemas costeiros, usando o pacote `geobr` (Pereira & Goncalves 2021).

```
## Download de polígonos dos geo_vetor_biomas Brasileiros
geo_vetor_biomas <- geobr::read_biomes(showProgress = FALSE) %>%
    dplyr::filter(name_biome != "Sistema Costeiro") %>%
    dplyr::rename(nome_bioma = name_biome,
                  codigo_bioma = code_biome,
                  ano = year)
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_vetor_biomas
```

Agora, quando utilizamos a função `plot()` para um objeto da classe `sf`, temos os três mapas, cada um indicando uma coluna da tabela de atritos (Figura 15.66).

```
## Plot
plot(geo_vetor_biomas)
```

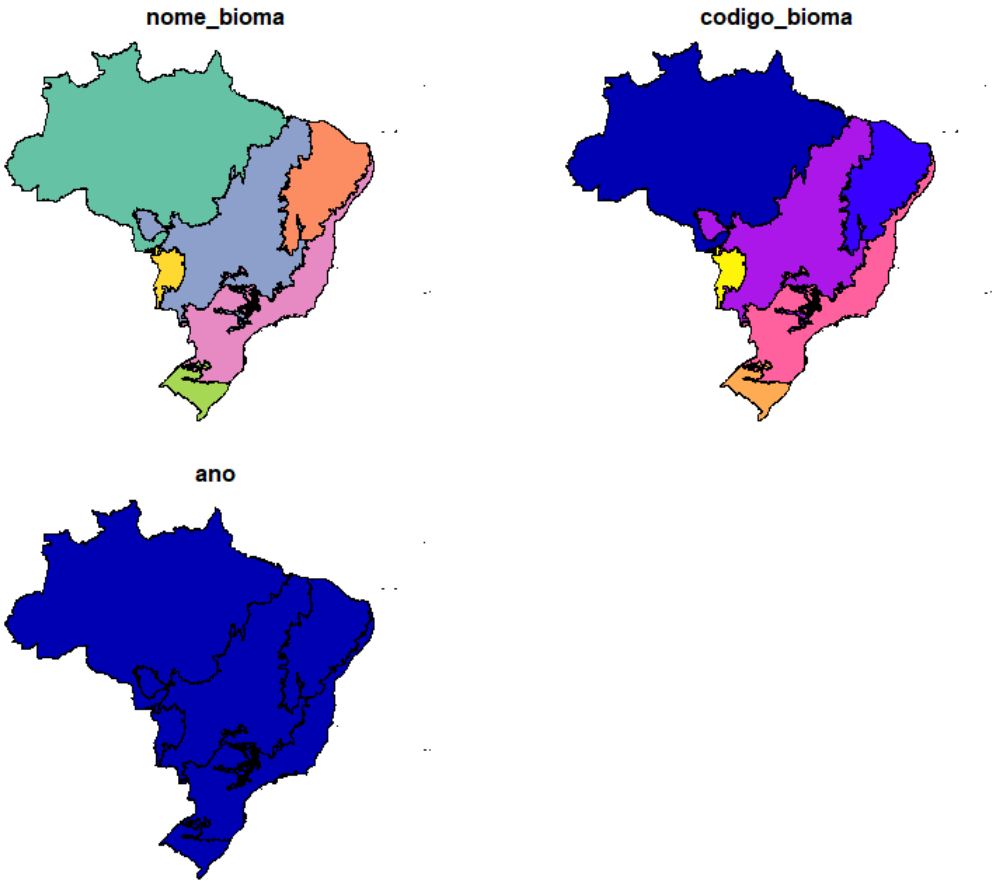


Figura 15.66: Mapa feito com a função `plot()` de um objeto `sf` para os Biomas do Brasil.

Selecionando as colunas desse objeto, podemos escolher a informação que queremos plotar, por exemplo, apenas a geometria `geom`. Além disso, podemos acrescentar os argumentos `col` para colorir e `main` para o título, além dos argumentos `axes` e `graticule` para adicionar as coordenadas e quadrículas, respectivamente. A legenda pode ser adicionada com a função `legend()` (Figura 15.67).

```
## Plot
plot(geo_vetor_biomas$geom,
     col = c("darkgreen", "orange", "orange4", "forestgreen",
             "yellow", "yellow3"),
     main = "Biomas do Brasil", axes = TRUE, graticule = TRUE)
legend(x = -75, y = -20, pch = 15, cex = .7, pt.cex = 2.5,
       legend = geo_vetor_biomas$nome_bioma,
       col = c("darkgreen", "orange", "orange4", "forestgreen",
               "yellow", "yellow3"))
```

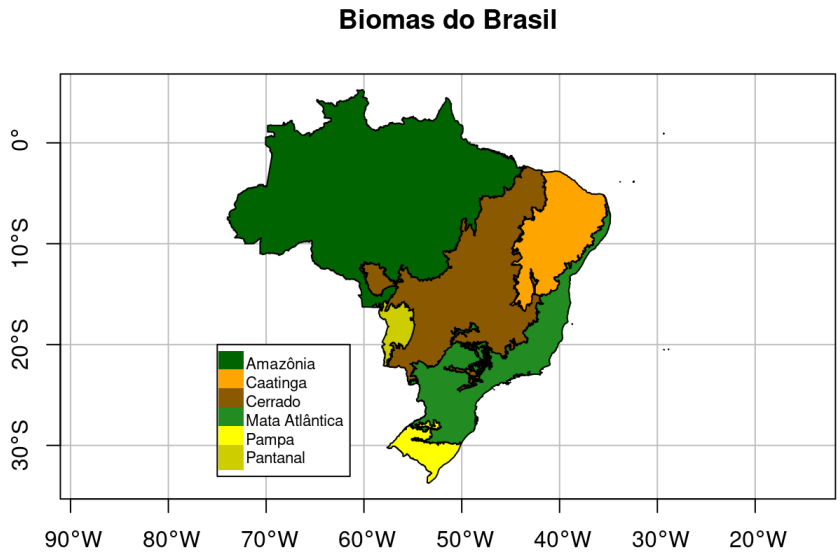


Figura 15.67: Mapa feito com a função `plot()` de um objeto vetor.

Para a classe dos objetos geoespaciais raster, a função `plot()` vai plotar um mapa para o tipo `RasterLayer` e quantos mapas houverem no objeto e couberem no espaço de plot para `RasterBrick` e `RasterStack`. Além disso, para essas classes do objeto raster, essa função provê também uma legenda e uma escala de cores automática (`terrain`). Vamos fazer o mapa da camada raster de elevação para o limite do município de Rio Claro/SP (Figura 15.68).

```
## Plot
plot(geo_raster_srtm_rio_claro)
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
```

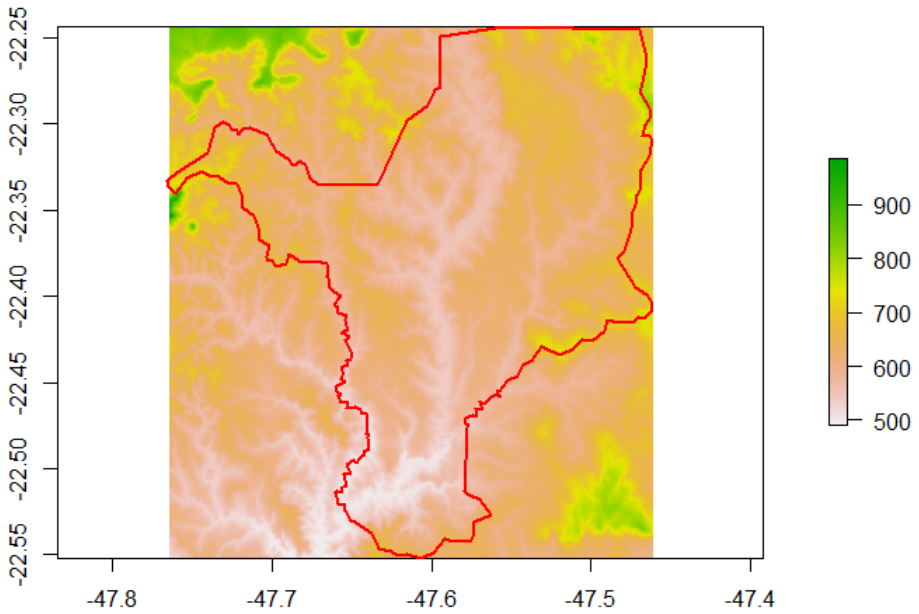


Figura 15.68: Mapa feito com a função `plot()` de um objeto raster com uma camada.

Agora vamos plotar objetos da classe `RasterStack`, alterando a cor para `viridis`, usando a função `viridis::viridis()` do pacote homônimo. Vamos fazer o mapa de duas camadas raster bioclimáticas para o mundo (Figura 15.69).

```
## Plot
plot(geo_raster_bioclim[[1:2]], col = viridis::viridis(10))
```

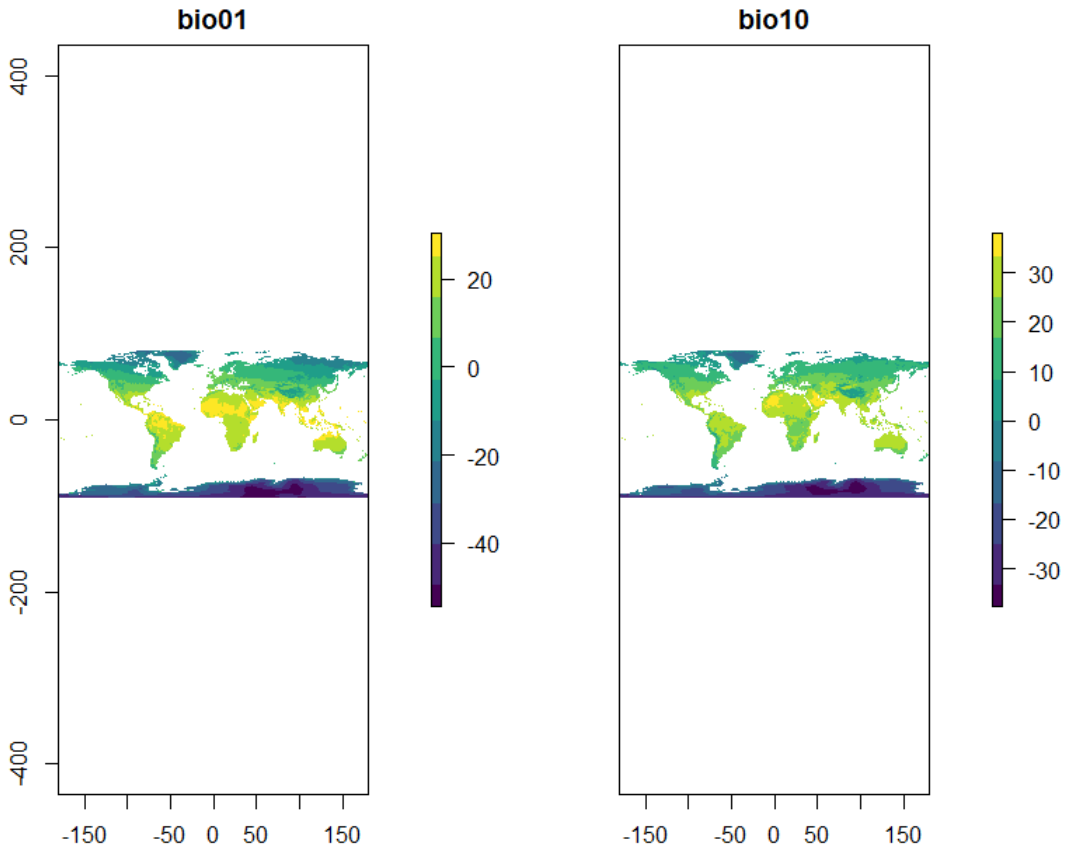


Figura 15.69: Mapa feito com a função `plot()` de um objeto raster com várias camadas.

Para exportar esses mapas podemos utilizar as funções `png()` ou `pdf()`, indicando os argumentos para ter as configurações que desejamos, e finalizando com a função `dev.off()`. Vamos exportar, a título de exemplo, a última figura (mais detalhes no Capítulo 6).

```
## Criar diretório
dir.create(here::here("dados", "mapas"))

## Exportar mapa
png(filename = here::here("dados", "mapas", "elev_rc.png"),
    width = 20, height = 20, units = "cm", res = 300)
plot(geo_raster_srtm_rio_claro)
plot(geo_vetor_rio_claro$geom, col = NA, border = "red", lwd = 2,
     add = TRUE)
dev.off()
```

Pacotes `ggplot2` e `ggspatial`

Como discutimos no Capítulo 6 sobre gráficos, o pacote `ggplot2` utiliza a gramática de gráficos para composição de figuras no R (Wilkinson & Wills 2005, Wickham 2016). Para cada classe de objeto geográfico há funções específicas para os dados: para objetos sf `geom_sf()` e para objetos raster `geom_raster()` e `geom_tile()`.

Além do pacote `ggplot2`, podemos utilizar o pacote `ggspatial` para acrescentar elementos geoespaciais como a barra de escala e o indicador de orientação (Norte), através das funções `annotation_scale()` e `annotation_north_arrow()`, respectivamente, além de outras funções específicas que não abordaremos nesta seção.

A estrutura de composição das funções do pacote `ggplot2` vai funcionar parecido com a estruturação de gráficos já vista no Capítulo 6, de modo que a cada função iremos utilizando o sinal de `+` para acrescentar outra camada. Indicaremos os dados com a função `ggplot()` e a coluna da tabela de atributos que queremos representar com a função `aes()`. Em seguida, utilizamos a função `geom_sf()` para indicar que trata-se de um objeto `sf`.

Além dessas funções, podemos ainda fazer alterações nos mapas através das funções: `scale_*()` que vai alterar as características indicadas em `aes()`, `coord_*()` que vai alterar construção do mapa em relação às coordenadas, `facet_*()` que altera a disposição de vários mapas, e `theme_*()` e `theme()` que alterarão características relacionadas ao tema, como cores de fundo, fontes e legenda. Podemos ainda utilizar as funções `annotate()` para adicionar textos e `labs()` para alterar o texto do título, legenda e eixos.

Vamos demonstrar esse funcionamento para compor o mapa de biomas, apresentado no início desta seção (Figura 15.70).

```
## Plot
mapa_vetor_biomas_ggplot2 <- ggplot(data = geo_vetor_biomas) +
    aes(fill = nome_bioma) +
    geom_sf(color = "black") +
    scale_fill_manual(values = c("darkgreen", "orange", "orange4",
                                 "forestgreen", "yellow", "yellow3")) +
    annotation_scale(location = "br") +
    annotation_north_arrow(location = "br", which_north = "true",
                           pad_x = unit(0, "cm"), pad_y = unit(.5, "cm"),
                           style = north_arrow_fancy_orienteering) +
    annotate(geom = "text", label = "CRS: SIRGAS2000/Geo",
             x = -38, y = -31, size = 2.5) +
    annotate(geom = "text", label = "Fonte: IBGE (2019)",
             x = -39, y = -32.5, size = 2.5) +
    labs(title = "Biomas do Brasil", fill = "Legenda", x = "Longitude",
         y = "Latitude") +
    theme_bw() +
    theme(title = element_text(size = 15, face = "bold"),
          legend.title = element_text(size = 10, face = "bold"),
          legend.position = c(.15, .25),
          legend.background = element_rect(colour = "black"),
```

```
          axis.title = element_text(size = 10, face = "plain"))
mapa_vetor_biomas_ggplot2
```

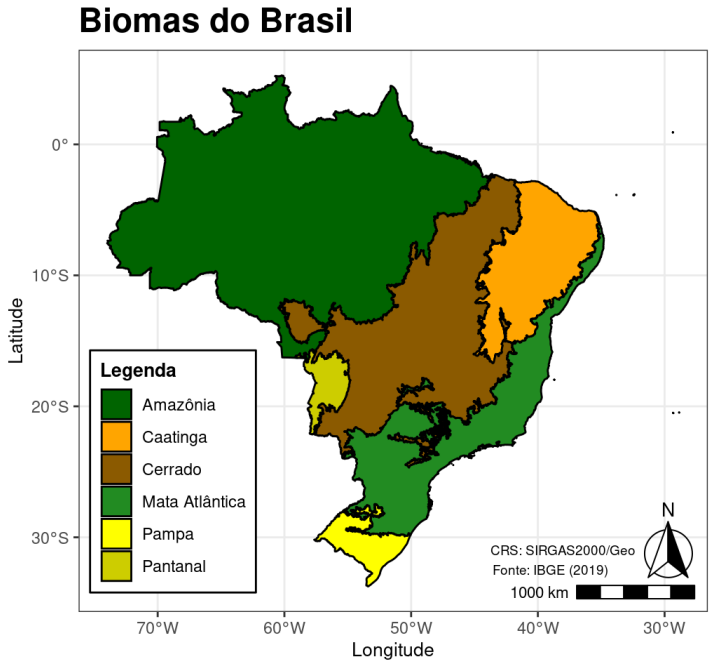


Figura 15.70: Mapa de Biomas do Brasil com o pacote `ggplot2`.

Para objetos raster, o uso do pacote `ggplot2` para compor mapas requer um passo preliminar. Primeiramente, vamos criar um `data frame` com os dados do raster, com as linhas sendo os pixels e as colunas sendo as coordenadas centrais da longitude e latitude, além dos valores de cada camada em cada coluna. Esse passo pode ser realizado com a função `raster::rasterToPoints()` ou `raster::as.data.frame()`.

Uma vez que temos esses dados organizados, podemos utilizar as funções `ggplot()` para indicar o `data frame`, e as colunas com a função `aes()`. Em seguida, utilizamos a função `geom_raster()` para indicar que trata-se de um objeto raster. Além dessas funções, podemos ainda utilizar as demais funções para alterar as características do mapa, como comentamos acima. Entretanto, devemos nos atentar para a função `coord_*()` e escolher aquela que vai fazer a construção do mapa em relação à coordenadas e resolução das células.

Como exemplo, vamos compor o mapa de elevação para Rio Claro/SP, adicionando também o limite do município (Figura 15.71).

```
## Dados
geo_raster_srtm_rio_claro_dados <- raster::rasterToPoints(
    geo_raster_srtm_rio_claro) %>%
    tibble::as_tibble()
head(geo_raster_srtm_rio_claro_dados)
#> # A tibble: 6 × 3
#>       x     y elevacao
#>   <dbl> <dbl>    <dbl>
#> 1 -47.8 -22.2      859
#> 2 -47.8 -22.2      856
```

```
#> 3 -47.8 -22.2      856
#> 4 -47.8 -22.2      856
#> 5 -47.8 -22.2      853
#> 6 -47.8 -22.2      852

## Mapa
mapa_srtm_rio_claro_ggplot2 <- ggplot() +
    geom_raster(data = geo_raster_srtm_rio_claro_dados,
                aes(x = x, y = y, fill = elevacao)) +
    geom_sf(data = geo_vetor_rio_claro, color = "red", fill = NA,
            size = 1.3) +
    scale_fill_viridis_c() +
    coord_sf() +
    annotation_scale(location = "br",
                     pad_x = unit(.5, "cm"), pad_y = unit(.7, "cm"),) +
    annotation_north_arrow(location = "br", which_north = "true",
                           pad_x = unit(.4, "cm"),
                           pad_y = unit(1.3, "cm"),
                           style = north_arrow_fancy_orienteering) +
    annotate(geom = "text", label = "CRS: WGS84/Geo",
             x = -47.51, y = -22.53, size = 3) +
    labs(title = "Elevação de Rio Claro/SP", fill = "Elevação (m)",
         x = "Longitude", y = "Latitude") +
    theme_bw() +
    theme(title = element_text(size = 15, face = "bold"),
          legend.title = element_text(size = 10, face = "bold"),
          legend.position = c(.2, .25),
          legend.background = element_rect(colour = "black"),
          axis.title = element_text(size = 10, face = "plain"),
          axis.text.y = element_text(angle = 90, hjust = .4))
mapa_srtm_rio_claro_ggplot2
```

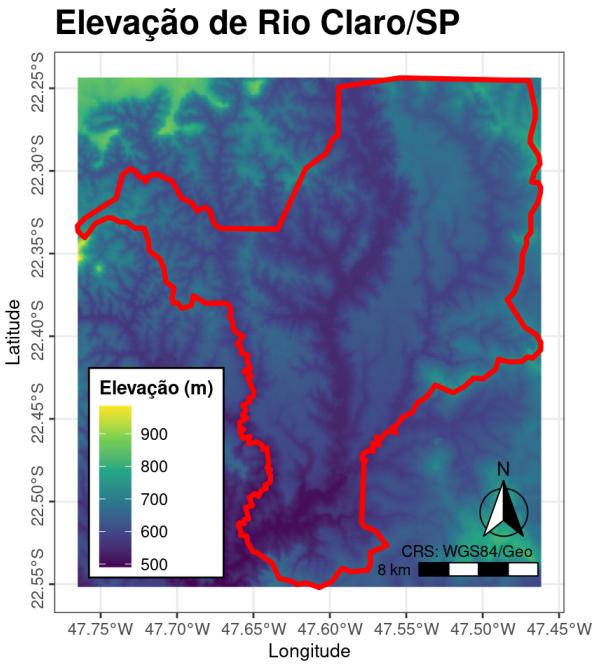


Figura 15.71: Mapa raster com o pacote `ggplot2`.

Para exportar mapas criados com o pacote `ggplot2`, podemos utilizar a função `ggplot2::ggsave()`, indicando os argumentos para ter as configurações que desejamos. Vamos exportar, a título de exemplo, a última figura.

```
## Exportar mapa ggplot2
ggsave(filename = here::here("dados", "mapas",
                             "srtm_rio_claro_ggplot2.png"),
       plot = mapa_srtm_rio_claro_ggplot2, width = 20, height = 20,
       units = "cm", dpi = 300)
```

Pacote tmap

O pacote `tmap` é um pacote direcionado à criação de mapas temáticos, com uma sintaxe concisa que permite a criação de mapas com o mínimo de códigos, mas muito similar à sintaxe do pacote `ggplot2` (Tennekes 2018). Ele também pode gerar mapas estáticos ou interativos usando o mesmo código, apenas mudando a forma de visualização com a função `tmap_mode()`, com o argumento `mode` igual a "plot" para estático e "view" para interativo. Por fim, o pacote `tmap` aceita diversas classes espaciais, incluindo objetos raster, de forma bastante simples. Mais sobre o pacote pode ser lido aqui.

> **Importante**
> Atentar para a instalação extra em Sistemas Operacionais GNU/Linux e MacOS.

Todas as funções do pacote `tmap` iniciam-se com `tm_*`, facilitando seu uso. A cada função iremos utilizar o sinal de `+` para acrescentar outra camada, da mesma forma que o pacote `ggplot2`. A principal função, em que todos os objetos geoespaciais são dados de entrada, é `tm_shape()`. A partir dela, podemos seguir com funções específicas para visualização de objetos `sf`, como `tm_polygons()`, `tm_borders()`, `tm_fill()`, `tm_lines()`, `tm_dots()` ou `tm_bubbles()`, ou com funções para

objetos raster como `tm_raster()`. Ainda há funções como `tm_text()` para representação de textos das colunas da tabela de atributos, e `tm_scale_bar()`, `tm_compass()` e `tm_graticules()`, para adicionar barra de escala, indicador de orientação (Norte) e gride de coordenadas, respectivamente. Por fim, a função `tm_credits()` adiciona um texto descritivo e a função `tm_layout()` faz diversas mudanças nos detalhes e apresentação do mapa.

Uma funcionalidade muito interessante do pacote `tmap` é o uso da função `tmaptools::palette_explorer()` para escolher as paletas de cores disponíveis. Essa função requer que os pacotes `shiny` e `shinyjs` estejam instalados, e quando executada, retorna uma aba onde é possível editar e escolher algumas paletas de cores nativas do `tmap`.

Diversos parâmetros podem ser acrescentados às funções de composição do `tmap`, mas não as detalharemos aqui, pois todas são descritas nos vignettes do pacote: tmap: get started! e tmap: version changes.

Vamos seguir com a composição do mapa de biomas para o Brasil apresentado no início dessa seção (Figura 15.72).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
mapa_vetor_biomas_tmap <- tm_shape(geo_vetor_biomas, bbox = c(-74, -35, -27,
10)) +
    tm_polygons(col = "nome_bioma",
                pal = c("darkgreen", "orange", "orange4",
                        "forestgreen", "yellow", "yellow3"),
                border.col = "black",
                title = "Legenda") +
    tm_compass() +
    tm_scale_bar(text.size = .6) +
    tm_graticules(lines = FALSE) +
    tm_credits("CRS: SIRGAS2000/Geo", position = c(.63, .13)) +
    tm_credits("Fonte: IBGE (2019)", position = c(.63, .09)) +
    tm_layout(title = "Biomas do Brasil",
              title.position = c(.25, .95),
              title.size = 1.8,
              title.fontface = "bold",
              legend.frame = TRUE,
              legend.position = c("left", "bottom"),
              legend.title.fontface = "bold")
mapa_vetor_biomas_tmap
```

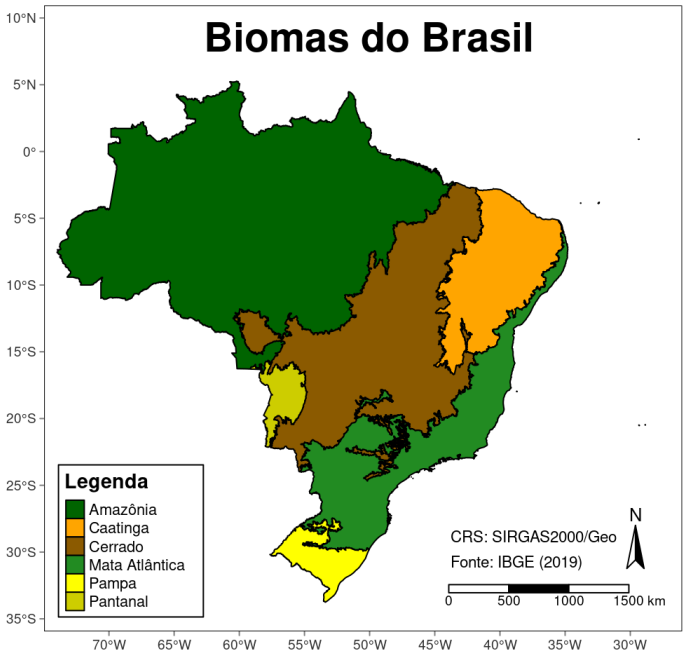


Figura 15.72: Mapa de Biomas do Brasil com o pacote `tmap`.

Além disso, o pacote `tmap` nos permite adicionar de forma simples um mapa secundário, provendo uma localização regional de interesse (Figura 15.73).

```
## Dados
geo_vetor_am_sul <- rnaturalearth::ne_countries(continent = "South America")
geo_vetor_brasil <- rnaturalearth::ne_countries(country = "Brazil")
geo_vetor_biomas <- geobr::read_biomes(showProgress = FALSE) %>%
    dplyr::filter(name_biome != "Sistema Costeiro")
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_vetor_am_sul
ecodados::geo_vetor_brasil
ecodados::geo_vetor_biomas

## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa secundário
mapa_am_sul <- tm_shape(geo_vetor_am_sul) +
    tm_polygons() +
    tm_shape(geo_vetor_brasil) +
    tm_polygons(col = "gray50")

## Juntando os mapas
mapa_vetor_biomas_tmap
print(mapa_am_sul, vp = grid::viewport(.815, .875, wi = .2, he = .2))
```

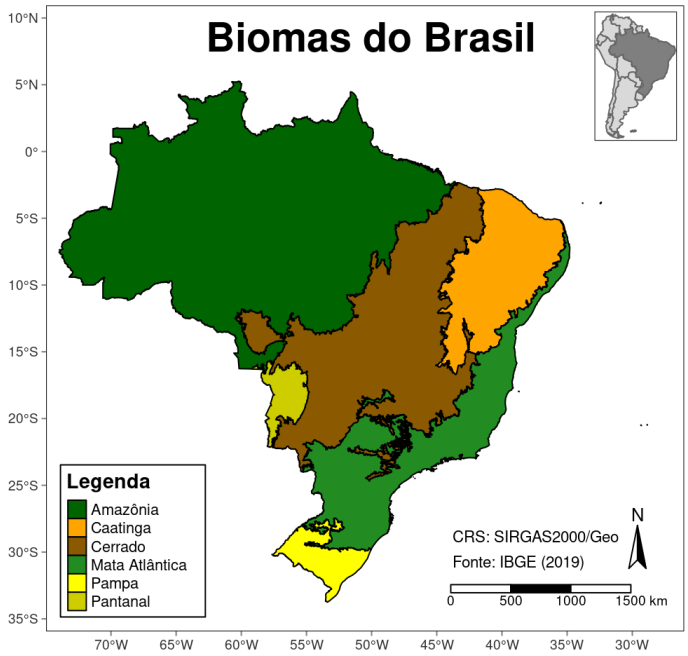


Figura 15.73: Mapa vetorial primário e secundário com o pacote `tmap`.

Como exemplo de mapa raster, vamos compor novamente o mapa de elevação para Rio Claro/SP, adicionando também o limite do município (Figura 15.74).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
mapa_srtm_rio_claro_tmap <- tm_shape(geo_raster_srtm_rio_claro) +
    tm_raster(pal = "viridis", title = "Elevação (m)") +
    tm_shape(geo_vetor_rio_claro) +
    tm_borders(col = "red", lwd = 2) +
    tm_compass(position = c(.9, .08)) +
    tm_scale_bar(text.size = .6, position = c(.67, 0)) +
    tm_graticules(lines = FALSE) +
    tm_credits("CRS: WGS84/Geo", position = c(.67, .06)) +
    tm_layout(title = "Elevação Rio Claro/SP",
              title.size = 1,
              title.fontface = "bold",
              legend.title.size = .7,
              legend.text.size = .6,
              legend.frame = TRUE,
              legend.position = c(.01, .01),
              legend.title.fontface = "bold")
mapa_srtm_rio_claro_tmap
```

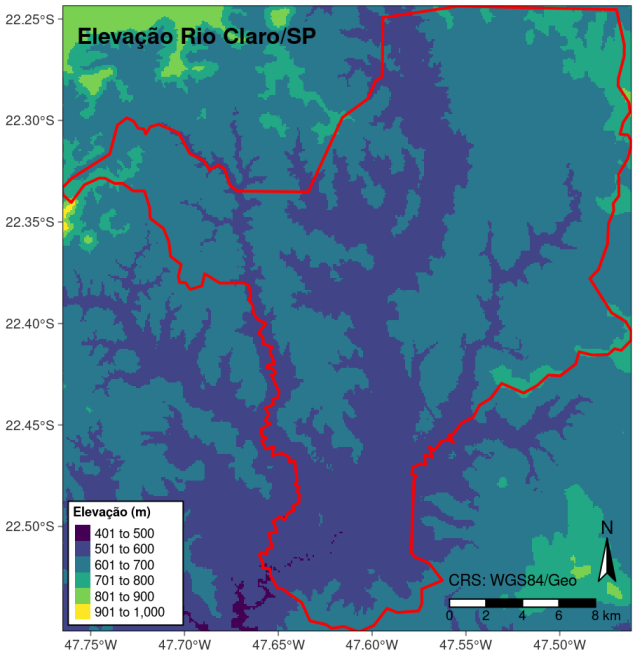


Figura 15.74: Mapa raster de elevação com o pacote `tmap`.

Para exportar mapas criados com o pacote `tmap` podemos utilizar a função `tmap::tmap_save()`, indicando os argumentos para ter as configurações que desejamos. Vamos exportar, a título de exemplo, a última figura.

```
## Exportar mapa tmap
tmap::tmap_save(tm = mapa_srtm_rio_claro_tmap,
                filename = here::here("dados", "mapas",
                                      "srtm_rio_claro_tmap.png"),
                width = 20, height = 20, units = "cm", dpi = 300)
```

15.10.4 Mapas animados

Podemos montar mapas facetados para mostrar como padrões espaciais variam ao longo do tempo, como por exemplo, os limites do Brasil ao longo dos anos (Figura 15.75). Entretanto, essa abordagem possui algumas desvantagens, de modo que as facetas podem ficar muito pequenas quando há muitas delas.

```
## Dados
geo_vetor_brasil_anos <- NULL
for(i in c(1872, 1900, 1911, 1920, 1933, 1940, 1950, 1960, 1970,
           1980, 1991, 2001, 2010, 2019)){
    geo_vetor_brasil_anos <- geobr::read_state(code_state = "all",
                                               year = i,
                                               showProgress = FALSE) %>%
        sf::st_geometry() %>%
        sf::st_as_sf() %>%
        dplyr::mutate(year = i) %>%
        dplyr::bind_rows(geo_vetor_brasil_anos, .)
}
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_vetor_brasil_anos

## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa facetado
mapa_brasil_tmap <- tm_shape(geo_vetor_brasil_anos) +
    tm_polygons() +
    tm_facets(by = "year", nrow = 4)
mapa_brasil_tmap
```

Figura 15.75: Mapa vetor facetado dos estados brasileiros ao longo do tempo com o pacote `tmap`.

Uma solução é a composição de mapas animados. Apesar de dependerem da publicação digital, os mapas animados podem aprimorar relatórios físicos à medida que o vínculo a uma página da web contendo a versão animada torna-se simples. Existem várias maneiras de gerar animações em R, por exemplo, com o pacote `gganimate`. Entretanto, aqui veremos a criação de mapas animados com `tmap`.

Podemos criar mapas animados alterando dois argumentos da função `tm_facets()`:

- trocando o `by = year` por `along = year`
- indicando o `free.coords = FALSE`

Por fim, podemos exportar o mapa animado no formato de `.gif` utilizando a função `tmap::tmap_animation()`, indicando a taxa de atualização com o argumento `delay` (Figura 15.76). Para

visualização do .gif animado, acessar o livro online. Alguns pacotes extras são requeridos dependendo do sistema operacional utilizado.

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa animado
mapa_brasil_tmap_ani <- tm_shape(geo_vetor_brasil_anos) +
    tm_polygons() +
    tm_facets(along = "year", free.coords = FALSE)

## Exportar mapa tmap animado
tmap::tmap_animation(tm = mapa_brasil_tmap_ani,
                     filename = here::here("dados", "mapas",
                                           "srtm_rio_claro_tmap_ani.gif"),
                     delay = 30)
```

Figura 15.76: Mapa vetorial animado mostrando os estados brasileiros ao longo do tempo com o pacote `tmap`.

15.10.5 Mapas interativos

Mapas interativos podem assumir muitas formas, sendo que a mais comum e útil é a capacidade de deslocar e ampliar qualquer parte de um conjunto de dados geoespaciais sobreposto em um "mapa

da web." Diversos pacotes nos permitem criar esse tipo de mapa, sendo os mais comuns o `tmap`, `mapview` e `leaflet`. Para visualização dos mapas interativos, acessar o livro online.

> **Importante**
> Destacamos que esses mapas irão ser compostos numa janela especial de "Viewer."

pacote tmap

Um recurso exclusivo do `tmap` é sua capacidade de criar mapas estáticos e interativos usando o mesmo código. Os mapas podem ser visualizados interativamente em qualquer ponto mudando para o modo de visualização, usando a função `tmap::tmap_mode(mode = "view")` (Figura 15.77).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "view")

## Atribuir novo mapa interativo
mapa_srtm_rio_claro_tmap_int <- mapa_srtm_rio_claro_tmap
```

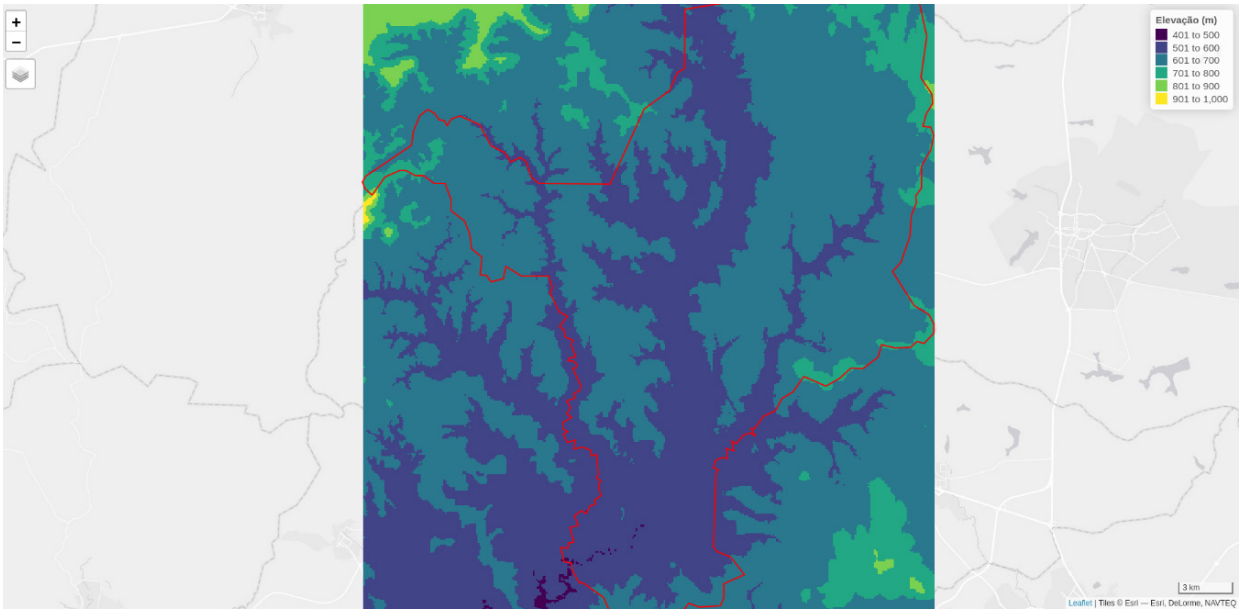


Figura 15.77: Mapa vetorial interativo com o pacote `tmap`.

Para exportar mapas interativos criados com o pacote `tmap`, podemos utilizar novamente a função `tmap::tmap_save()`, indicando a extensão como `.html`.

```
## Exportar mapa tmap interativo
tmap::tmap_save(tm = mapa_srtm_rio_claro_tmap_int,
 filename = here::here("dados", "mapas", "srtm_rio_claro_tmap_int.html"))
```

Pacote mapview

O pacote `mapview` cria rapidamente mapas interativos simples com a função `mapvew::mapview()` (Figura 15.78). Entretanto, outras características podem ser mudadas para criar mapas mais elaborados, como pode ser visto através do site do pacote.

```
## Mapa
mapa_srtm_rio_claro_mapview_int <- mapview::mapview(
    geo_raster_srtm_rio_claro, col.regions = viridis::viridis(100))
```

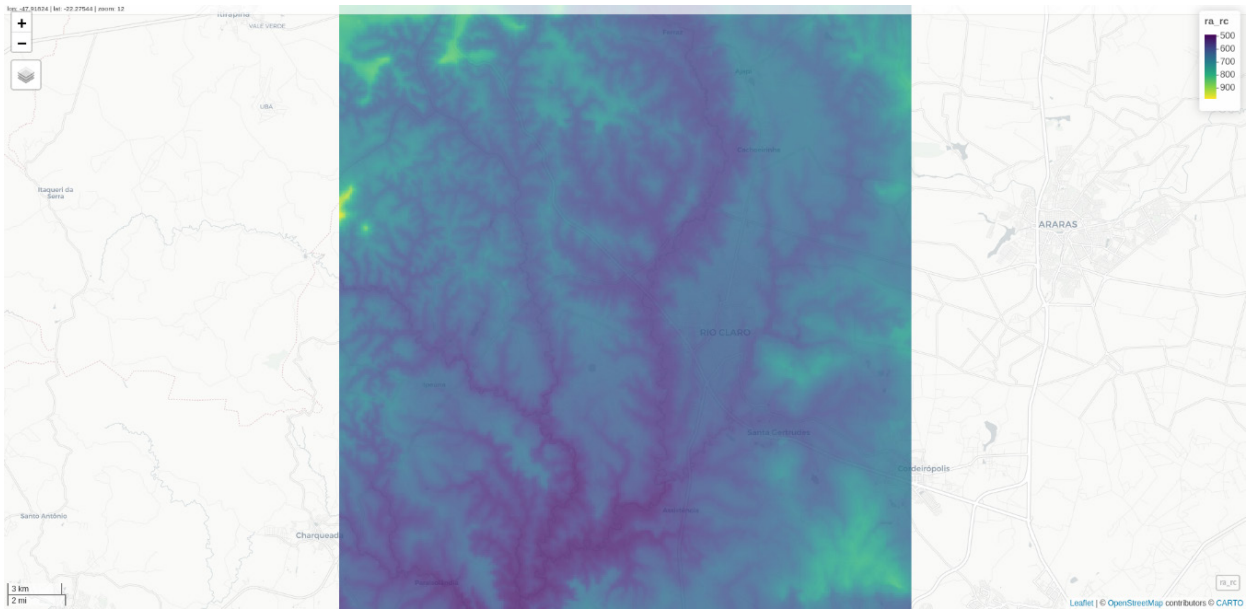


Figura 15.78: Mapa vetorial interativo com o pacote `mapview`.

Para exportar mapas interativos criados com o pacote `mapview`, podemos utilizar a função `mapivew::mapshot()`, indicando a extensão como `.html`.

```
## Exportar mapa mapview interativo
mapview::mapshot(x = mapa_srtm_rio_claro_mapview_int,
   url = here::here("dados", "mapas", "srtm_rio_claro_mapview_int.html"))
```

Pacote leaflet

O `leaflet` é um dos pacotes de mapeamento interativo mais utilizados e completos em R. Esse pacote fornece uma interface utilizando a biblioteca *JavaScript* e muitos argumentos podem ser compreendidos lendo a documentação da biblioteca original.

Mapas interativos usando esse pacote são criados utilizando a função `leaflet::leaflet()`. O resultado dessa função é um objeto da classe leaflet, que pode ser alterado por outras funções deste mesmo pacote, permitindo que várias camadas e configurações de controle sejam adicionadas interativamente (Figura 15.79).

Mais sobre o pacote `leaflet` pode ser consultado em seu site e CheatSheet.

```
## Paleta de cores
pal <- colorNumeric(viridis::viridis(10),
                    raster::values(geo_raster_srtm_rio_claro))
```

```
## Mapa
mapa_srtm_rio_claro_leaflet_int <- leaflet() %>%
    addProviderTiles("CartoDB.Positron") %>%
    addRasterImage(geo_raster_srtm_rio_claro, colors = pal,
                   opacity = .8) %>%
    addLegend(pal = pal,
              values = raster::values(geo_raster_srtm_rio_claro),
              title = "Elevação (m)") %>%
    addPolygons(data = geo_vetor_rio_claro, col = "red", fill = NA)
```

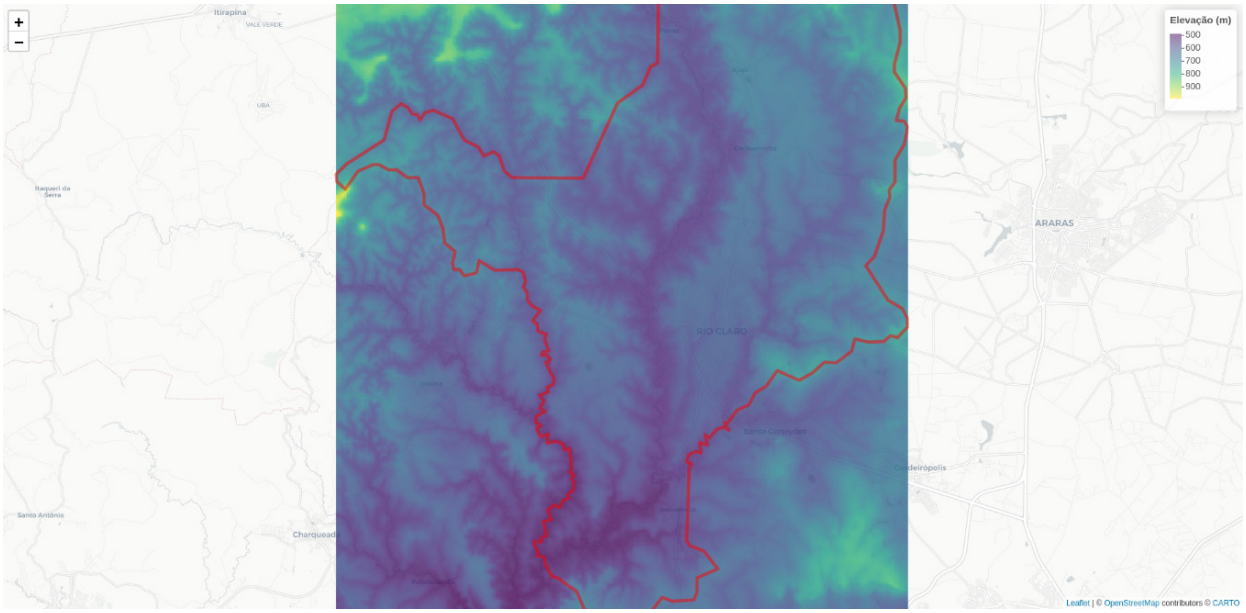


Figura 15.79: Mapa vetorial interativo com o pacote `leaflet`.

Para exportar mapas interativos criados com o pacote `leaflet`, podemos utilizar novamente a função `mapivew::mapshot()`, indicando a extensão como `.html`.

```
## Exportar mapa leaflet interativo
mapview::mapshot(x = mapa_srtm_rio_claro_leaflet_int,
   url = here::here("dados", "mapas", "srtm_rio_claro_leaflet_int.html"))
```

15.11 Exemplos de aplicações de análises geoespaciais para dados ecológicos

Agora que vimos os principais conceitos e aplicações do manejo e visualização de dados geoespaciais, podemos avançar para realizar quatro exemplos de aplicações para dados ecológicos. Para isso, usaremos novamente os dados de comunidades de anfíbios da Mata Atlântica (Vancine et al. 2018). Primeiramente, veremos como resumir informações de biodiversidade (número de ocorrências e riqueza) para hexágonos. Num segundo momento, veremos como associar dados ambientais a coordenadas de espécies ou comunidades. Depois, como resumir dados de rasters para buffers. Por

fim, realizaremos predições espaciais contínuas de adequabilidade de habitat para uma espécie e do número de espécies.

15.11.1 Resumir informações de biodiversidade para unidades espaciais

Resumir informações para unidades espaciais é um passo muito frequente em análises de Macroecologia, Biogeografia ou Ecologia da Paisagem. Nesta seção, contabilizaremos o número de ocorrências e riqueza de anfíbios para hexágonos na Mata Atlântica.

Primeiramente, vamos importar e preparar os dados de biodiversidade que usaremos nesses exemplos. Vamos começar importando os locais de amostragens de anfíbios na Mata Atlântica e selecionando apenas as colunas de interesse.

```
## Importar locais
geo_anfibios_locais <- readr::read_csv(
    here::here("dados", "tabelas", "ATLANTIC_AMPHIBIANS_sites.csv"),
    col_types = cols()) %>%
    dplyr::select(id, longitude, latitude, species_number)
```

Agora vamos importar as espécies das comunidades, selecionando apenas as espécies com nomes válidos e transformando a coluna de indivíduos para 1, para compor posteriormente uma matriz de comunidade de espécies.

```
## Importar espécies
geo_anfibios_especies <- readr::read_csv(
    here::here("dados", "tabelas", "ATLANTIC_AMPHIBIANS_species.csv"),
    col_types = cols()) %>%
    tidyr::drop_na(valid_name) %>%
    dplyr::select(id, valid_name, individuals) %>%
    dplyr::distinct(id, valid_name, .keep_all = TRUE) %>%
    dplyr::mutate(individuals = tidyr::replace_na(individuals, 1),
                  individuals = ifelse(individuals > 0, 1, 1))
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_anfibios_locais
ecodados::geo_anfibios_especies
```

Podemos agora juntar a tabela de locais, que possui as coordenadas à tabela de espécies. Em seguida convertemos essa única tabela na classe vetor `sf`.

```
## Junção das coordenadas e conversão para classe sf
geo_anfibios_especies_locais_vetor <- geo_anfibios_especies %>%
    dplyr::left_join(geo_anfibios_locais, by = "id") %>%
    dplyr::relocate(longitude, latitude, .after = 1) %>%
    dplyr::mutate(lon = longitude, lat = latitude) %>%
    sf::st_as_sf(coords = c("lon", "lat"), crs = 4326)
```

Agora vamos baixar o limite do Bioma da Mata Atlântica para o Brasil, converter o CRS para WGS84/Geo e ajustar sua extensão para remover as ilhas no Oceano Atlântico.

```
## Download do Bioma da Mata Atlântica
geo_vetor_mata_atlantica <- geobr::read_biomes(year = 2019,
                                               showProgress = FALSE) %>%
    dplyr::filter(name_biome == "Mata Atlântica") %>%
    sf::st_transform(crs = 4326) %>%
    sf::st_crop(xmin = -55, ymin = -30, xmax = -34, ymax = -5)
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_vetor_mata_atlantica
```

Podemos verificar se as coordenadas e o limite do bioma estão todos corretos compondo um mapa preliminar, usando o pacote `tmap` (Figura 15.80).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
tm_shape(geo_vetor_mata_atlantica,
         bbox = geo_anfibios_especies_locais_vetor) +
    tm_polygons() +
    tm_shape(geo_anfibios_especies_locais_vetor) +
    tm_dots(size = .1, col = "forestgreen")
```

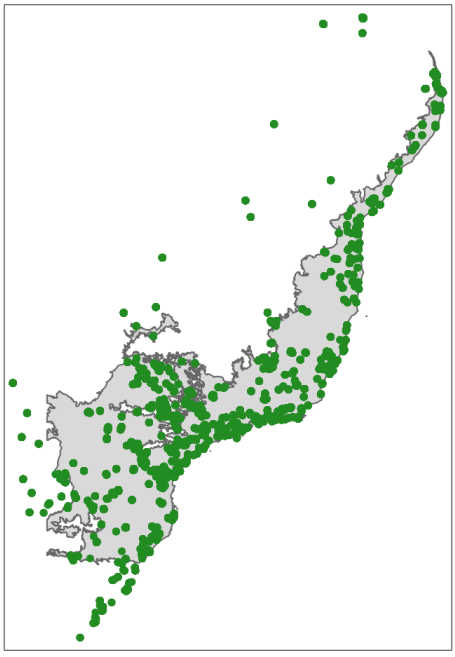


Figura 15.80: Mapa dos locais do Atlantic Amphibians e do limite da Mata Atlântica.

Como o limite utilizado para reunir informações das comunidades de anfíbios foi o mais abrangente possível (Muylaert et al. 2018, Vancine et al. 2018), selecionaremos apenas os locais que caem dentro do limite da Mata Atlântica que estamos utilizando aqui.

```
## Selecionar os locais dentro do limite
geo_anfibios_especies_locais_vetor_mata_atlantica <-
           geo_anfibios_especies_locais_vetor[geo_vetor_mata_atlantica, ]
```

Podemos refazer o mapa mostrando as coordenadas retiradas em vermelho e as que ficaram em verde (Figura 15.81).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
tm_shape(geo_vetor_mata_atlantica,
         bbox = geo_anfibios_especies_locais_vetor) +
    tm_polygons() +
    tm_shape(geo_anfibios_especies_locais_vetor) +
    tm_bubbles(size = .1, col = "red") +
    tm_shape(geo_anfibios_especies_locais_vetor_mata_atlantica) +
    tm_bubbles(size = .1, col = "forestgreen")
```

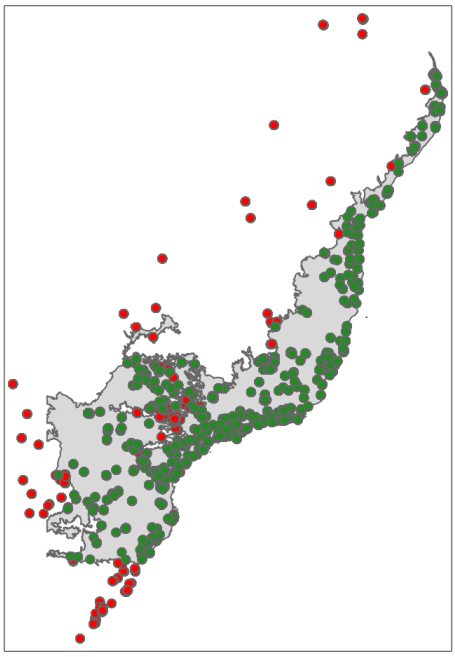


Figura 15.81: Mapa dos locais do Atlantic Amphibians que caem dentro do limite da Mata Atlântica.

O próximo passo é criar um gride de hexágonos para o Bioma da Mata Atlântica. Usaremos a função `sf::st_make_grid()` que pode criar quadrículas ou hexágonos. Esses hexágonos terão a área equivalente às quadrículas de 1º de tamanho (aproximadamente 10000 km^2). Usaremos a função `sf::st_area()` para calcular as áreas dos hexágonos e a função `tibble::rowid_to_column()` para criar uma identificação para cada feição.

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Criar hexágonos de ~110 km
geo_vetor_mata_atlantica_hex <- sf::st_make_grid(
    x = geo_vetor_mata_atlantica, cellsize = 1, square = FALSE) %>%
    sf::st_as_sf() %>%
    dplyr::mutate(areakm2 = sf::st_area(.)/1e6) %>%
    tibble::rowid_to_column("id_hex")

## Selecionar os hexágonos dentro do limite da Mata Atlântica
geo_vetor_mata_atlantica_hex <-
                  geo_vetor_mata_atlantica_hex[geo_vetor_mata_atlantica, ]
```

Podemos conferir os hexágonos criados fazendo um mapa preliminar (Figura 15.82).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
tm_shape(geo_vetor_mata_atlantica,
         bbox = geo_vetor_mata_atlantica_hex) +
    tm_polygons() +
    tm_shape(geo_vetor_mata_atlantica_hex) +
    tm_borders()
```

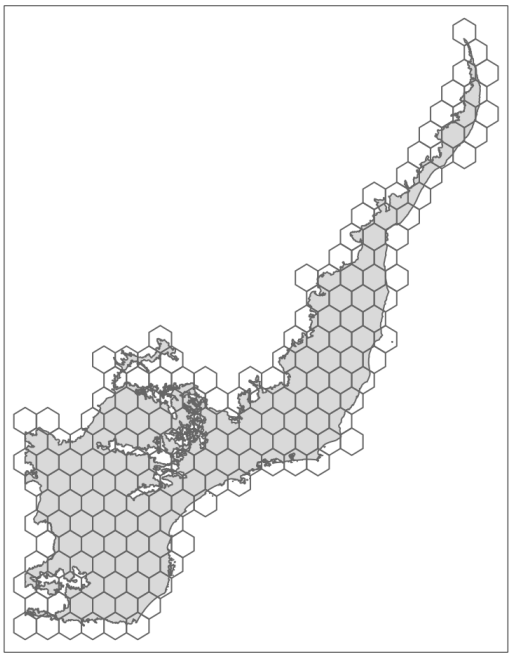


Figura 15.82: Mapa dos hexágonos para o limite da Mata Atlântica.

Podemos ser mais restritos e selecionar apenas os hexágonos dentro do limite do Bioma da Mata Atlântica utilizando o operador `st_within()` (Figura 15.83).

```
## Selecionar os hexágonos totalmente dentro do limite da Mata Atlântica
geo_vetor_mata_atlantica_hex_total <-
              geo_vetor_mata_atlantica_hex[geo_vetor_mata_atlantica, ,
              op = st_within]

## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
tm_shape(geo_vetor_mata_atlantica,
         bbox = geo_vetor_mata_atlantica_hex) +
    tm_polygons() +
    tm_shape(geo_vetor_mata_atlantica_hex_total) +
    tm_borders()
```

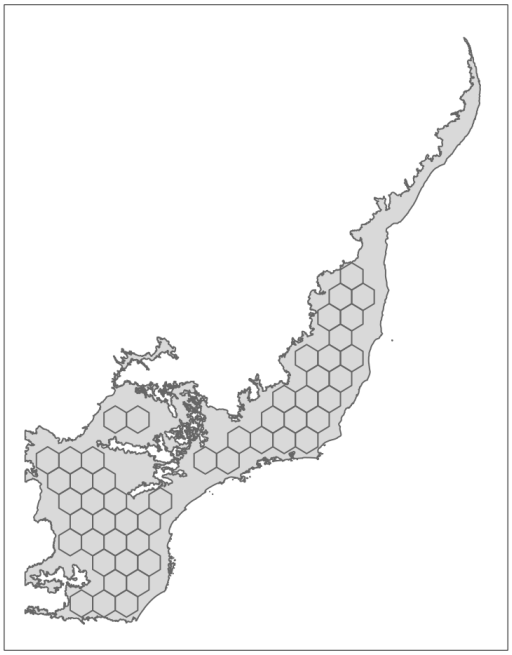


Figura 15.83: Mapa dos hexágonos totalmente dentro do limite da Mata Atlântica.

Podemos agora associar as espécies aos hexágonos fazendo um "join" espacial, utilizando a função `sf::st_join()`.

```
## Junção espacial dos locais com os hexágonos
geo_vetor_mata_atlantica_hex_especies <- sf::st_join(
    x = geo_vetor_mata_atlantica_hex,
    y = geo_anfibios_especies_locais_vetor_mata_atlantica,
    left = TRUE)
```

Por fim, podemos agregar os dados para ter o número de ocorrências e de espécies por hexágono.

```
## Agregar dados de ocorrências e número de espécies por hexágono
geo_vetor_mata_atlantica_hex_especies_oco_riq <-
    geo_vetor_mata_atlantica_hex_especies %>%
```

```
    dplyr::group_by(id_hex) %>%
    dplyr::summarise(ocorrencias =
                     length(valid_name[!is.na(valid_name)]),
                     riqueza = n_distinct(valid_name, na.rm = TRUE))
```

Finalmente podemos compor os mapas finais, mostrando os hexágonos com cores e valores do número de ocorrências e de espécies (15.84).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa de ocorrências
mapa_oco <- tm_shape(geo_vetor_mata_atlantica_hex_especies_oco_riq) +
    tm_polygons(title = "Ocorrência de anfíbios", col = "ocorrencias",
                pal = "viridis", style = "pretty") +
    tm_text("ocorrencias", size = .4) +
    tm_graticules(lines = FALSE) +
    tm_compass() +
    tm_scale_bar() +
    tm_layout(legend.title.size = 2,
              legend.title.fontface = "bold",
              legend.position = c("left", "top"))

## Mapa de riqueza
mapa_riq <- tm_shape(geo_vetor_mata_atlantica_hex_especies_oco_riq) +
    tm_polygons(title = "Riqueza de anfíbios", col = "riqueza",
                pal = "viridis", style = "pretty") +
    tm_text("riqueza", size = .4) +
    tm_graticules(lines = FALSE) +
    tm_compass() +
    tm_scale_bar() +
    tm_layout(legend.title.size = 2,
              legend.title.fontface = "bold",
              legend.position = c("left", "top"))

## União dos mapas
tmap_arrange(mapa_oco, mapa_riq)
```

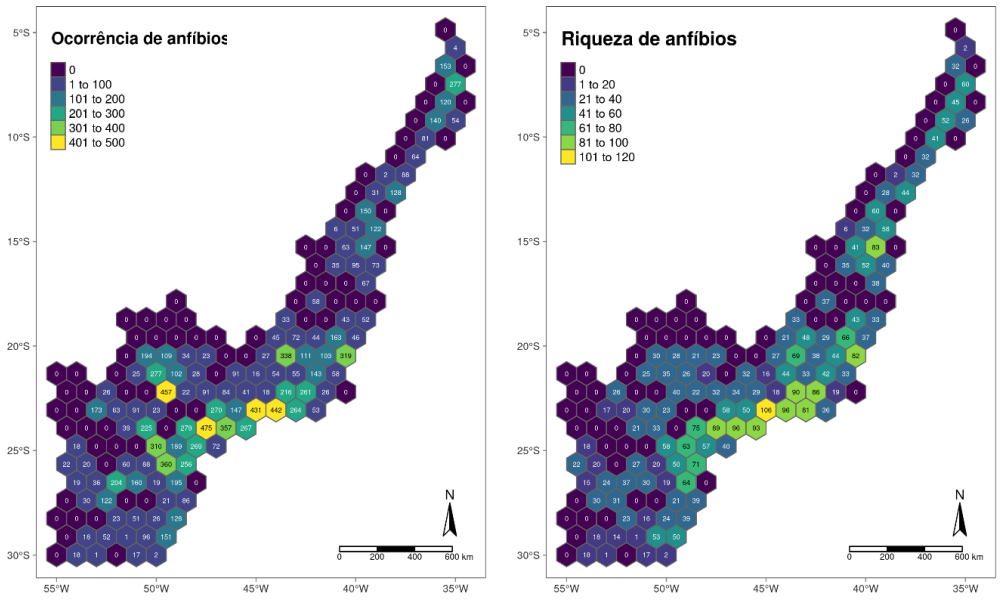


Figura 15.84: Mapa com o número de ocorrências e riqueza de anfíbios para hexágonos no limite da Mata Atlântica.

15.11.2 Extrair dados raster para pontos

Atribuir informações ambientais a ocorrências é um passo fundamental para diversas análises. Nesta seção, atribuiremos os valores das variáveis bioclimáticas aos locais de amostragem de anfíbios na Mata Atlântica.

Já realizamos o download das variáveis bioclimáticas na seção de raster. Vamos importar novamente esses dados, primeiramente listando as camadas e depois importando com a função `raster:stack()`.

```
## Listar arquivos
arquivos_raster <- dir(path = here::here("dados", "raster"),
                       pattern = "wc") %>%
                       grep(".tif", ., value = TRUE)

## Importar rasters
geo_raster_bioclim <- raster::stack(here::here("dados",
                                               "raster",
                                                arquivos_raster))

## Renomear rasters
names(geo_raster_bioclim) <- c("bio01", paste0("bio", 10:19),
                               paste0("bio0", 2:9))
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_raster_bioclim
```

Da seção anterior, já temos o objeto com a tabela de coordenadas dos locais de amostragem das comunidades de anfíbios. Vamos agora criar um objeto vetorial das coordenadas e em seguida selecionar os locais dentro do limite do bioma da Mata Atlântica.

```
## Importar locais e converter em sf
geo_anfibios_locais_vetor <- geo_anfibios_locais %>%
    dplyr::mutate(lon = longitude, lat = latitude) %>%
    sf::st_as_sf(coords = c("lon", "lat"), crs = 4326)
geo_anfibios_locais_vetor
#> Simple feature collection with 1163 features and 4 fields
#> Geometry type: POINT
#> Dimension:     XY
#> Bounding box:  xmin: -56.74194 ymin: -33.51083 xmax: -34.79667 ymax:
-3.51525
#> Geodetic CRS:  WGS 84
#> # A tibble: 1,163 × 5
#>    id      longitude latitude species_number             geometry
#>  * <chr>       <dbl>    <dbl>          <dbl>          <POINT [°]>
#>  1 amp1001     -43.4    -8.68             19     (-43.42194 -8.68)
#>  2 amp1002     -38.9    -3.55             16 (-38.85783 -3.545527)
#>  3 amp1003     -38.9    -3.57             14 (-38.88869 -3.574194)
#>  4 amp1004     -38.9    -3.52             13   (-38.9188 -3.51525)
#>  5 amp1005     -38.9    -4.28             30 (-38.91083 -4.280556)
#>  6 amp1006     -36.4    -9.23             42 (-36.42806 -9.229167)
#>  7 amp1007     -40.9    -3.85             23 (-40.89444 -3.846111)
#>  8 amp1008     -40.9    -3.83             19 (-40.91944 -3.825833)
#>  9 amp1009     -40.9    -3.84             13   (-40.91028 -3.8375)
#> 10 amp1010     -35.2    -6.14              1 (-35.22944 -6.136944)
#> # … with 1,153 more rows
```

Usaremos agora a função `raster::extract()` para extrair e associar os valores das variáveis bioclimáticas para os locais de amostragem.

```
## Extrair valores das variáveis para os locais
geo_anfibios_locais_vetor_bioclim <- geo_anfibios_locais_vetor %>%
    dplyr::mutate(raster::extract(geo_raster_bioclim, ., df = TRUE)) %>%
    dplyr::select(-ID) %>%
    dplyr::relocate(bio02:bio09, .after = bio01)
```

Podemos ver esses dados na Tabela 15.14.

Tabela 15.14: Dados extraídos e atribuídos aos locais de amostragens de comunidade de anfíbios na Mata Atlântica.

id	long	lat	Species number	bio01	bio02	bio03	bio04	bio05
amp1001	-43.421	-8.680	19	24.88	12.84	76.28	84.54	33.17
amp1002	-38.857	-3.5455	16	26.43	7.80	78.70	59.72	31.37
amp1003	-38.888	-3.574	14	26.43	7.80	78.70	59.72	31.37

id	long	lat	Species number	bio01	bio02	bio03	bio04	bio05
amp1004	-38.918	-3.515	13	26.43	7.80	78.70	59.72	31.37
amp1005	-38.910	-4.280	30	22.52	8.43	73.76	64.62	28.08
amp1006	-36.428	-9.229	42	21.61	8.11	67.74	147.67	27.98

Podemos ainda fazer alguns mapas para espacializar essas variáveis (Figura 15.85).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
geo_anfibios_locais_vetor_bioclim %>%
    dplyr::select(bio01:bio06) %>%
    tidyr::gather(var, val, -geometry) %>%
    tm_shape() +
    tm_bubbles(size = .1, col = "val", pal = "viridis") +
    tm_facets("var", free.scales = TRUE) +
    tm_layout(legend.outside = FALSE)
```

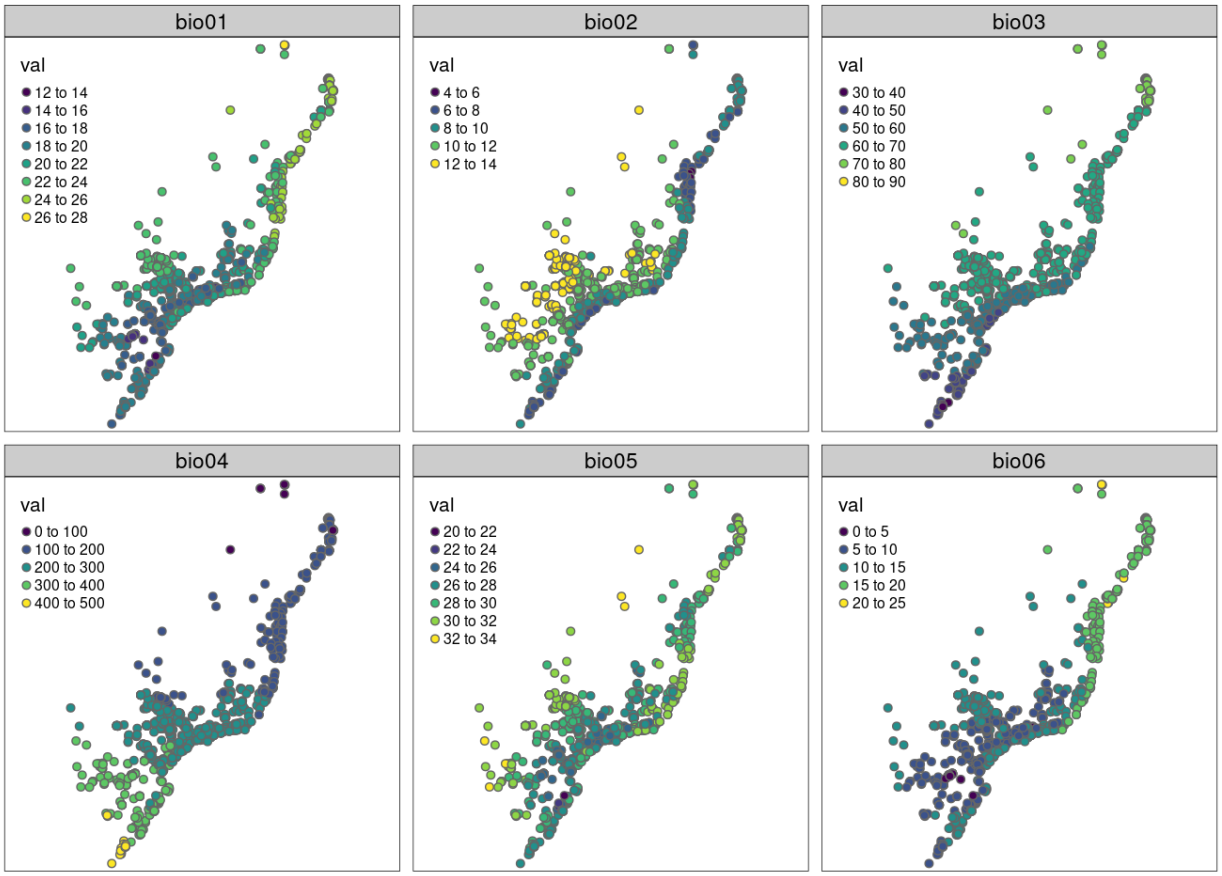


Figura 15.85: Mapa mostrando os valores das variáveis bioclimáticas (BIO01:BIO06) para os locais amostrados de comunidades de anfíbios para o limite da Mata Atlântica.

15.11.3 Resumir dados de rasters para buffers

Muitas análises requerem que façamos um resumo da composição da paisagem para buffers, sendo o buffer uma unidade de análise espacial no entorno de um ponto de amostragem.

Aqui, usaremos os dados do GlobCover v.2.3 de 2009 (Arino et al. 2012) como raster de cobertura da terra. O arquivo é grande (~400 Mb) e pode demorar muito, dependendo da velocidade da sua internet.

```
## Aumentar o tempo de download
options(timeout = 1e5)

## Download dos dados do GlobCover
download.file(
  url = "http://due.esrin.esa.int/files/Globcover2009_V2.3_Global_.zip",
  destfile = here::here("dados", "raster",
                        "Globcover2009_V2.3_Global.zip"),
                        mode = "wb", extra = "c")

## Unzip
unzip(zipfile = here::here("dados", "raster",
                           "Globcover2009_V2.3_Global.zip"),
                           exdir = here::here("dados", "raster"))
```

Depois de fazer o download, vamos importar e ajustar esse raster para o limite da Mata Atlântica (Figura 15.86).

```
## Importar raster do GlobCover
geo_raster_globcover <- raster::raster(
    here::here("dados", "raster", "GLOBCOVER_L4_200901_200912_V2.3.tif"))

## Ajustar para o limite do bioma da Mata Atlântica
geo_raster_globcover_mata_atlantica <- geo_raster_globcover %>%
    raster::crop(geo_vetor_mata_atlantica) %>%
    raster::mask(geo_vetor_mata_atlantica)
geo_raster_globcover_mata_atlantica
#> class      : RasterLayer
#> dimensions : 8940, 7274, 65029560  (nrow, ncol, ncell)
#> resolution : 0.002777778, 0.002777778  (x, y)
#> extent     : -54.99861, -34.79306, -29.98194, -5.148611  (xmin, xmax,
ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : GLOBCOVER_L4_200901_200912_V2.3
#> values     : 14, 220  (min, max)
```

Caso o download não funcione ou haja problemas com a importação, disponibilizamos os dados também no pacote `ecodados`.

```
## Importar os dados pelo pacote ecodados
ecodados::geo_raster_globcover_mata_atlantica
```

```
## Plot
plot(geo_raster_globcover_mata_atlantica,
     col = viridis::viridis(n = 200))
```

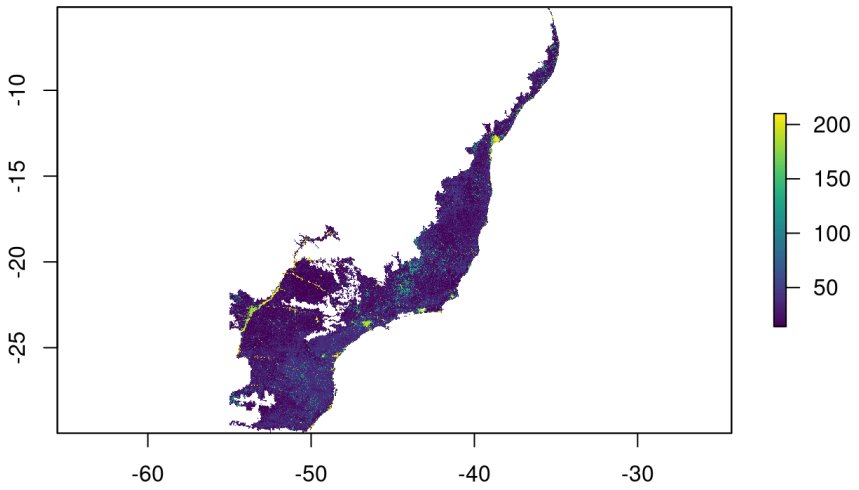


Figura 15.86: Camada raster do GlobCover 2.3 para o Bioma da Mata Atlântica.

Vamos agora transformar a tabela de locais em vetor, selecionar aleatoriamente 50 amostragens das comunidades de anfíbios e criar buffers de ~10 km.

```
## Fixar a amostragem
set.seed(42)

## Pontos
geo_anfibios_especies_locais_vetor_mata_atlantica <-
    geo_anfibios_locais %>%
    sf::st_as_sf(coords = c("longitude", "latitude"), crs = 4326) %>%
    dplyr::filter(sf::st_intersects(x = .,
                  y = geo_vetor_mata_atlantica, sparse = FALSE)) %>%
    dplyr::slice_sample(n = 50)

## Buffers de ~10 km
geo_anfibios_especies_locais_vetor_mata_atlantica_buffer10km <-
    sf::st_buffer(geo_anfibios_especies_locais_vetor_mata_atlantica,
                  dist = 10000)
```

Podemos conferir no mapa da Figura 15.87, atente para o aumento artificial do tamanho dos buffers para permitir a visualização.

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
tm_shape(geo_vetor_mata_atlantica) +
 tm_polygons() +
 tm_shape(geo_anfibios_especies_locais_vetor_mata_atlantica_buffer10km) +
```

```
    tm_bubbles(size = .3, border.col = "red", alpha = 0) +
  tm_shape(geo_anfibios_especies_locais_vetor_mata_atlantica) +
  tm_dots(size = .01, col = "forestgreen")
```

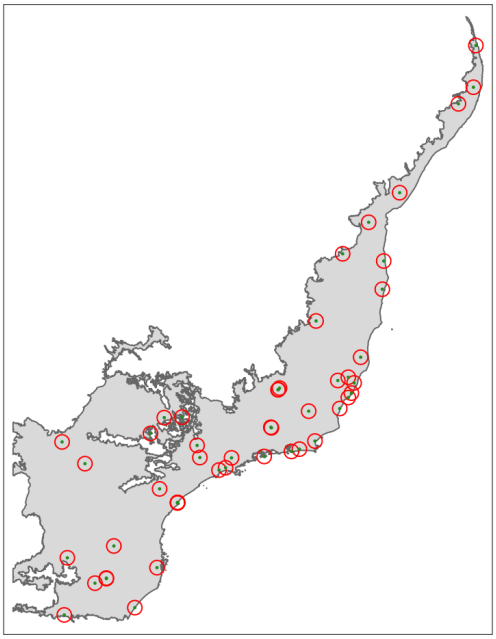


Figura 15.87: Distribuição de 50 localidades aleatórios e buffers de ~1 km (fora de escala).

Agora podemos utilizar a função `raster::extract()` para fazer a contabilização, já em porcentagem, de pixels de cada classe para cada buffer.

```
## Estatística zonal
geo_anfibios_locais_vetor_ma_buffer10km_ext <- raster::extract(
    x = geo_raster_globcover_mata_atlantica,
    y = geo_anfibios_especies_locais_vetor_mata_atlantica_buffer10km,
    df = TRUE, na.rm = TRUE) %>%
    dplyr::rename(id = ID, classe = 2) %>%
    tidyr::drop_na(classe) %>%
    dplyr::mutate(classe = paste0("classe_", classe)) %>%
    dplyr::group_by(id, classe) %>%
    dplyr::summarise(n = n()) %>%
    tidyr::pivot_wider(id_cols = id, names_from = classe,
                       values_from = n) %>%
    dplyr::mutate(across(everything(), ~replace_na(.x, 0))) %>%
    janitor::adorn_totals("col") %>%
    janitor::adorn_percentages("row") %>%
    janitor::adorn_pct_formatting(rounding = "half up", digits = 1)
head(geo_anfibios_locais_vetor_ma_buffer10km_ext)
```

Agora podemos combinar esses dados aos dados dos buffers.

```
## Combinação
geo_anfibios_locais_vetor_ma_buffer10km_ext_bind  <- dplyr::bind_cols(
```

```
    x = geo_anfibios_especies_locais_vetor_mata_atlantica_buffer10km,
    y = geo_anfibios_locais_vetor_ma_buffer10km_ext[, -1]) %>%
    sf::st_drop_geometry()
geo_anfibios_locais_vetor_ma_buffer10km_ext_bind
```

15.11.4 Predições espaciais de objetos raster

O pacote `raster` além de permitir realizar a manipulação e visualização de dados raster no R, também permite a extrapolação do ajuste de análises, como `LMs`, `GLMs`, `GAMs` dentre outras (Capítulos 7 e 8). Aqui, faremos uma pequena demonstração utilizando a função `raster::predict()`, predizendo o resultado de dois ajustes de GLMs para a presença/ausência de uma espécie de anuro e a extrapolação do número de espécies de anfíbios para o Bioma da Mata Atlântica.

Para ajustar um GLM para dados de presença/ausência, podemos usar o conjunto de dados já criado anteriormente, com as espécies e as coordenadas, e fazer uma junção com a última tabela que criamos com os dados bioclimáticos.

```
## Junção dos dados ambientais aos dados de espécies
geo_anfibios_locais_especies_vetor_bioclim <- geo_anfibios_especies %>%
    dplyr::left_join(.,
    sf::st_drop_geometry(geo_anfibios_locais_vetor_bioclim), by = "id")
```

Agora, vamos selecionar ocorrências da espécie *Haddadus binotatus*, atribuindo **1** quando ela ocorre e **0** quando ela não ocorre. Essa espécie é relativamente comum na serrapilheira de fragmentos florestais da Mata Atlântica, e recebe esse nome em homenagem a um grande pesquisador de anfíbios da Mata Atlântica, o **Prof. Célio Fernando Baptista Haddad**.

```
## Seleção da espécie Haddadus binotatus
geo_anfibios_locais_especies_vetor_bioclim_hb <-
    geo_anfibios_locais_especies_vetor_bioclim %>%
    dplyr::mutate(pa = ifelse(valid_name == "Haddadus binotatus", 1, 0),
                  .after = individuals) %>%
    dplyr::distinct(id, .keep_all = TRUE)
```

Vamos utilizar apenas as variáveis não correlacionadas para o índice de correlação de Pearson para r < 0,7 (Capítulo 7).

```
## Correlação entre as variáveis
corr <- geo_anfibios_locais_especies_vetor_bioclim_hb %>%
    dplyr::select(bio01:bio19) %>%
    cor() %>%
    caret::findCorrelation(.7, names = TRUE)

## Seleção das variáveis não correlacionadas
geo_anfibios_locais_especies_vetor_bioclim_hb_cor <- geo_anfibios_locais_
especies_vetor_bioclim_hb %>%
    dplyr::select(pa, bio01:bio19) %>%
    dplyr::select(-c(corr))
```

Agora sim, podemos ajustar um modelo simples da presença e ausência dessa espécie, utilizando as variáveis não correlacionadas, através de um GLM para a família binomial. Novamente, nosso intuito não é analisar se o modelo atende a todos os pressupostos, e sim exemplificar a predição espacial; para esses detalhes, consulte o Capítulo 8.

```
## Ajustar um modelo GLM binomial
modelo_pa <- glm(formula = pa ~ .,
                 data = geo_anfibios_locais_especies_vetor_bioclim_hb_cor,
                 family = binomial("logit"))
```

Antes de fazermos a predição da distribuição potencial da espécie é fundamental que o objeto raster esteja ajustado para o limite da Mata Atlântica. Para isso vamos utilizar as funções `raster::crop()` e `raster::mask()` para fazer esse ajuste (Figura 15.88).

```
## Ajuste da extensão e limite
geo_raster_bioclim_mata_atlantica <- geo_raster_bioclim %>%
    raster::crop(geo_vetor_mata_atlantica) %>%
    raster::mask(geo_vetor_mata_atlantica)

## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Mapa
tm_shape(geo_raster_bioclim_mata_atlantica[[c(1, 4)]]) +
    tm_raster(pal = "viridis", title = c("bio01", "bio12")) +
    tm_facets(free.scales.raster = TRUE)
```

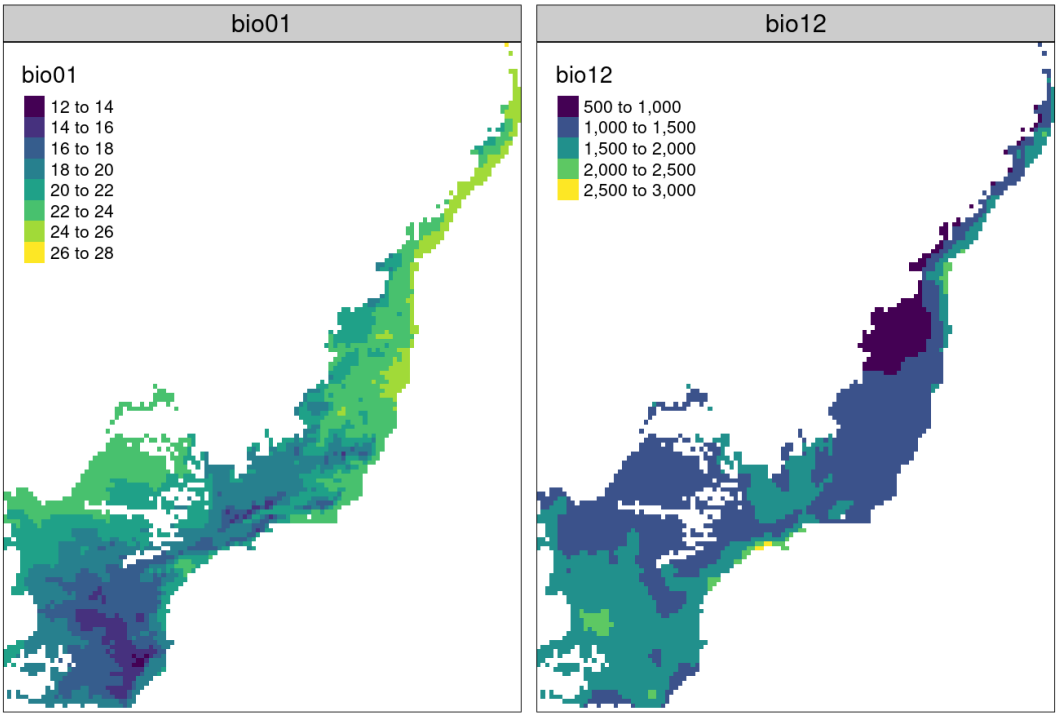


Figura 15.88: Mapa de dois rasters (BIO01 e BIO12) ajustados ao limite da Mata Atlântica.

Agora podemos fazer a predição desse modelo para todo o Bioma da Mata Atlântica. Essa função vai utilizar os coeficientes do modelo ajustado para gerar um raster de predição para todos os pixels da Mata Atlântica. Vamos usar o argumento `type = "response"` para que os valores da predição sejam ajustados a 0 a 1.

> **Importante**
> Para que a predição funcione, os nomes das camadas raster `geo_raster_bioclim_mata_atlantica` devem possuir o mesmo nome das colunas das variáveis preditoras ajustadas no modelo `modelo_pa`.

```
## Predições
modelo_pa_pred <- raster::predict(
    object = geo_raster_bioclim_mata_atlantica,
    model = modelo_pa,
    type = "response")
modelo_pa_pred
#> class      : RasterLayer
#> dimensions : 149, 121, 18029  (nrow, ncol, ncell)
#> resolution : 0.1666667, 0.1666667  (x, y)
#> extent     : -55, -34.83333, -30, -5.166667  (xmin, xmax, ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : layer
#> values     : 1.255833e-12, 0.1983049  (min, max)
```

No último passo podemos tornar esse modelo binário, ou seja, apenas com valores 0 ou 1. Para isso vamos adotar arbitrariamente o valor de 0,01 como ponto de corte. A partir desse valor consideraremos os pixels acima como 1 e abaixo como 0.

```
## Seleção dos pixels de presença/ausência potencial
modelo_pa_pred_corte <- modelo_pa_pred >= .01
```

Por fim, vamos produzir dois mapas mostrando os valores das predições e o mapa binário da espécie (Figura 15.89).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")

## Combinar os dois raster
modelo_pa_pred <- raster::stack(modelo_pa_pred, modelo_pa_pred_corte)
names(modelo_pa_pred) <- c("Contínuo", "Binário")

## Mapas de predição contínua e binária
tm_shape(modelo_pa_pred) +
    tm_raster(pal = "viridis", title = c("Predição", "Predição"),
              style = "fisher") +
    tm_facets(free.scales.raster = TRUE)
```

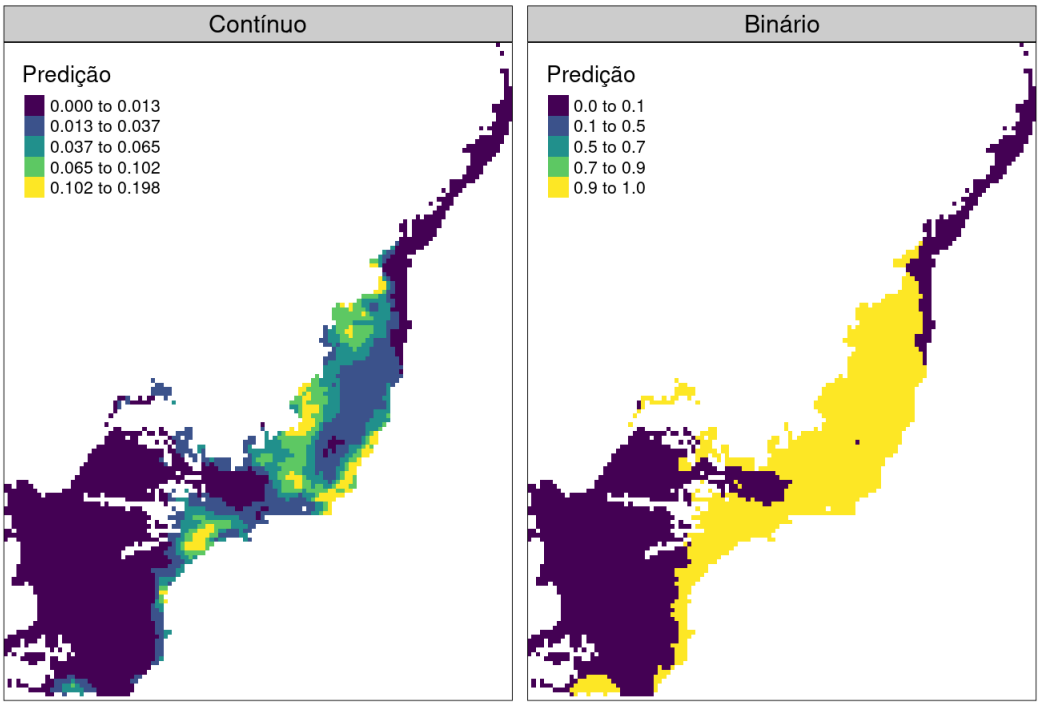


Figura 15.89: Mapa da predição contínua e binária do modelo ajustado para a presença/ausência da espécie Haddadus binotatus na Mata Atlântica.

> **Importante**
> Essa análise foi realizada com o intuito de exemplificar o funcionamento da função `raster::predict()`, para mais detalhes, consultar livros específicos da área de Modelagem de Distribuição de Espécies ou Modelagem de Nicho Ecológico Fletcher & Fortin (2018).

Em nossa segunda análise, vamos predizer os dados de riqueza de anfíbios (i.e. número de espécies) para todo o bioma da Mata Atlântica. Para isso, temos de retirar novamente as variáveis correlacionadas.

```
## Correlação
corr <- geo_anfibios_locais_vetor_bioclim %>%
    sf::st_drop_geometry() %>%
    dplyr::select(bio01:bio19) %>%
    cor() %>%
    caret::findCorrelation(.7, names = TRUE)

## Seleção das variáveis não correlacionadas
geo_anfibios_locais_bioclim_cor <- geo_anfibios_locais_vetor_bioclim %>%
    sf::st_drop_geometry() %>%
    dplyr::select(species_number, bio01:bio19) %>%
    dplyr::select(-c(corr))
```

Agora sim, podemos criar os GLMs com famílias de distribuição apropriadas para dados de contagem como Poisson e Binomial Negativa. Novamente, nosso intuito não é analisar se o modelo atende à todos os pressupostos, e sim exemplificar a predição espacial, para esses detalhes, consulte o Capítulo 8.

```
## Modelo Poisson
modelo_riq_pois <- glm(
    formula = species_number ~ .,
    data = geo_anfibios_locais_bioclim_cor,
    family = poisson)

## Modelo Binomial Negativo
modelo_riq_nb <- MASS::glm.nb(
    formula = species_number ~ .,
    data = geo_anfibios_locais_bioclim_cor)
```

Com os modelos ajustados, podemos fazer as predições utilizando os objetos raster com as variáveis ambientais.

```
## Predição do modelo Poisson
modelo_riq_pois_pred <- raster::predict(
    object = geo_raster_bioclim_mata_atlantica,
    model = modelo_riq_pois,
    type = "response")
modelo_riq_pois_pred
#> class      : RasterLayer
#> dimensions : 149, 121, 18029  (nrow, ncol, ncell)
#> resolution : 0.1666667, 0.1666667  (x, y)
#> extent     : -55, -34.83333, -30, -5.166667  (xmin, xmax, ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : layer
#> values     : 8.249131, 24.76082  (min, max)

## Predição do modelo Binomial Negativo
modelo_riq_nb_pred <- raster::predict(
    object = geo_raster_bioclim_mata_atlantica,
    model = modelo_riq_nb,
    type = "response")
modelo_riq_nb_pred
#> class      : RasterLayer
#> dimensions : 149, 121, 18029  (nrow, ncol, ncell)
#> resolution : 0.1666667, 0.1666667  (x, y)
#> extent     : -55, -34.83333, -30, -5.166667  (xmin, xmax, ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : memory
#> names      : layer
#> values     : 9.262095, 24.45297  (min, max)
```

Por fim, podemos compor os dois mapas de predições (Figura 15.90).

```
## Mudar o modo de exibição do tmap
tmap::tmap_mode(mode = "plot")
```

```
## Combinar os dois raster
modelo_riq_pred <- raster::stack(modelo_riq_pois_pred,
                                 modelo_riq_nb_pred)
names(modelo_riq_pred) <- c("Poisson", "Binomial Negativa")

## Mapas da predição Poisson e Binomial Negativa
tm_shape(modelo_riq_pred) +
    tm_raster(pal = "viridis",
              title = c("Número de espécies", "Número de espécies"),
              style = "fisher") +
    tm_facets(free.scales.raster = TRUE)
```

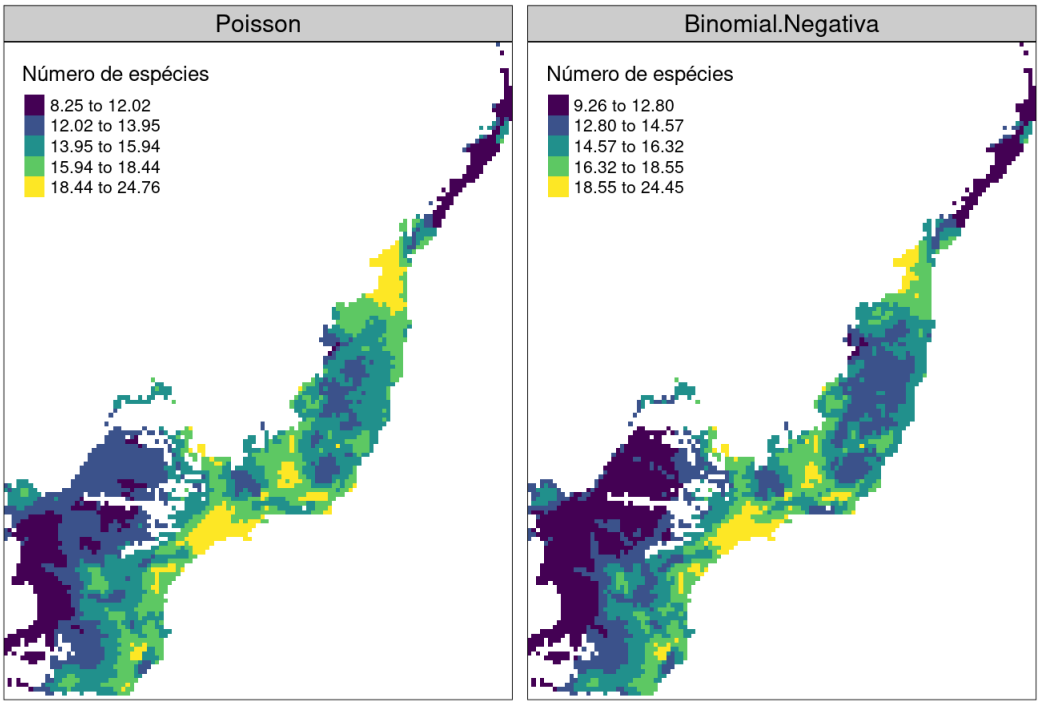


Figura 15.90: Mapa da predição de riqueza utilizando o modelo Poisson e Binomial Negativa para a Mata Atlântica.

Importante

Essa análise foi realizada com o intuito de exemplificar o funcionamento da função `raster::predict()`, para mais detalhes, consultar livros específicos da área de Ecologia Espacial (Fletcher & Fortin 2018).

15.12 Para se aprofundar

15.12.1 Livros

Listamos aqui as principais referências sobre manipulação, visualização de dados geoespaciais e análises geoespaciais no R. Recomendamos aos interesssados(as) os livros: i) Lovelace, Nowosad & Muenchow (2019) Geocomputation with R, ii) Mas et al. (2019) Análise espacial com R, iii) Olaya (2020) Sistemas de Información Geográfica, iv) Moraga (2019) Geospatial Health Data: Modeling and Visualization with R-INLA and Shiny, v) Brunsdon & Comber (2015) An Introduction to Spatial Analysis and Mapping in R, vi) Wegmann, Leutner & Dech (2016) *Remote Sensing and GIS for Ecologists: Using Open Source Software*, vii) Wegmann, Schwalb-Willmann & Dech (2020) *An Introduction to Spatial Data Analysis Remote Sensing and GIS with Open Source Software*, viii) Fletcher & Fortin (2018) *Spatial ecology and conservation modeling: Applications with R*, ix) Lapaine, Miljenko, Usery, E. Lynn (2017) *Choosing a Map Projection*.

15.12.2 Links

- Awesome Geospatial
- Spatial Data Science
- Intro to GIS and Spatial Analysis
- Introduction to Spatial Data Programming with R
- R for Geographic Data Science
- Spatial Data Science with R
- Spatial Modelling for Data Scientists
- Predictive Soil Mapping with R

15.13 Exercícios

15.1 Importe o limite dos estados brasileiros no formato `sf` com o nome `br`. Para isso, use a função `ne_states` do pacote `rnaturalearth`. Crie um mapa simples cinza utilizando a função `plot()`, selecionando a coluna `geometry` com o operador `$` e com os argumentos `axes` e `graticule` verdadeiros.

15.2 Dados vetoriais podem ser criados com diversos erros de topologia, e.g., sobreposição de linhas ou polígonos ou buracos. Algumas funções exigem que os objetos vetoriais aos quais são atribuídos esses dados não possuam esses erros para que o algoritmo funcione. Para verificar se há erros, podemos usar a função `st_is_valid()` do pacote `sf`. Há diversas forma de correções desses erros, mas vamos usar uma correção simples do R, com a função `st_make_valid()`. Vamos fazer essa correção para o `br` importado anteriormente e atribuindo ao objeto `br_valid`. Podemos conferir para saber se há erros e fazer um plot.

15.3 Crie um objeto `RasterLayer` vazio chamado `ra` com resolução de 5º (~600 km). Atribua um sistema de referência de coordenadas com o código `4326`. Atribua valores aleatórios de uma distribuição normal e plote o mesmo.

15.4 Reprojete o limite dos estados brasileiros do exercício anterior para o CRS SIRGAS 2000/Brazil Polyconic, utilizando o código `EPSG:5880` e chamando de `br_poly`. Faça um mapa simples como no exercício 1. Atente para as curvaturas das linhas.

15.5 Utilizando a função `st_centroid` do pacote `sf`, crie um vetor chamado `br_valid_cen` que armazenará o centroide de cada estado brasileiro do objeto `br_valid` do exercício 2 e plot o resultado.

15.6 Ajuste o limite e máscara do objeto raster criado no exercício 3 para o limite do Brasil, atribuindo ao objeto `ra_br`. Depois reprojete esse raster para a mesma projeção utilizada no exercício 4 com o nome `ra_br_poly` e plote o mapa resultante.

15.7 Extraia os valores de cada pixel do raster criado no exercício 6 para os centroides dos estados do Brasil criado no exercício 5, atribuindo à coluna `val` do objeto espacial chamado `br_valid_poly_cent_ra`.

15.8 Crie um mapa final usando os resultados dos exercícios 4, 5 e 6. Utilize o pacote `tmap` e inclua todos os principais elementos de um mapa.

Soluções dos exercícios.

Glossário

Neste glossário, resumimos os principais conceitos e definições dos termos utilizados no texto principal que possam ser desconhecidos ou pouco claros para alguns leitores(as).

A

Abundance-based Coverage Estimator (ACE): estimador de riqueza de espécies baseado na abundância de espécies raras

Agregação (raster): aumentar o tamanho dos pixels (diminuindo a resolução) de um raster, agregando os valores dos pixels em um pixel maior

Agrupamento filogenético: espécies coexistindo nas comunidades são mais aparentadas do que esperado pelo acaso

Alinhamento (raster): ajusta o tamanho do pixel, extensão, número e origem dos pixels para várias camadas rasters

Ambiente ou *Environment* (RStudio): porção onde os objetos criados são armazenados

Amostra: subconjunto da população, selecionados por um processo adequado de amostragem, utilizado para estimar características de toda a população

Análise de covariância (ANCOVA): é uma extensão da ANOVA com a adição de uma covariável medida em todas as unidades amostrais

Análise de variância (ANOVA): é um teste estatístico que avalia se há diferenças entre as médias de três ou mais grupos independentes. Existem uma variedade de delineamentos experimentais como - ANOVA de um fator, ANOVA de dois fatores, ANOVA em blocos aleatorizados, ANOVA de medidas repetidas e ANOVA *split-splot* - que diferem na maneira que o teste estatístico e os graus de liberdade são calculados

Análise paramétrica: assume que os dados foram amostrados de uma distribuição de forma conhecida (e.g., gaussiana, poisson, etc) e estima os parâmetros (e.g., média, desvio padrão, etc) da distribuição a partir dos dados

Análise não paramétrica: não pressupõe que os dados foram amostrados de uma distribuição de forma conhecida (e.g., gaussiana, poisson, etc)

Array: classe de objetos que representa elementos de um único modo no formato de combinação de tabelas, com linhas, colunas e dimensões

Árvore filogenética: são hipóteses que representam a relação de parentesco entre as espécies (pode ser também indivíduos, genes, etc.) com informações sobre quais espécies compartilham um ancestral comum e a distância (tempo, genética, ou diferenças nos caracteres) que as separam

Atributo funcional: uma propriedade mensurável dos organismos (geralmente em nível individual) que representa características morfológicas, fisiológicas ou fenológicas que afetam a aptidão alterando aspectos do crescimento, reprodução e sobrevivência

Atributos dos objetos: são o modo (natureza) e a estrutura (organização) dos elementos nos objetos

Autovalor(Eigenvalue): número inteiro (escalar) que multiplica um vetor, sendo portanto múltiplo deste. Esses valores representam a variância dos eixos e, se convertidos em valores relativos, medem a porcentagem de variância contida em cada eixo.

Autovetor (Eigenvector): vetor não nulo que muda somente quando é multiplicado por um escalar. O autovetor de uma matriz é encontrado pelo resultado da multiplicação de um vetor pela matriz, que é igual a *lambda* vezes o vetor. Esse vetor do resultado passa a ser o autovetor, e o *lambda* o seu respectivo autovalor

B

Bloco (ANOVA): é uma área ou período de tempo dentro do qual as condições ambientais são relativamente homogêneas. O objetivo do uso dos blocos é controlar fontes de variações indesejadas na variável dependente que não são de interesse do pesquisador

Bootstrap: estimador de riqueza de espécies que estima os parâmetros de uma população por reamostragens

Buffer (vetor): polígono que representa a área dentro de uma determinada distância de um dado vetorial, podendo ser a partir de um ponto, linha ou polígono

C

Camada (vetor ou raster): termo geral, mas geralmente associada à diferentes rasters reunidos num mesmo arquivo

Centroide (vetor ou raster): ponto central de uma feição vetorial ou o centro do pixel do raster

Centróide (multivariado): média ponderada de um conjunto multivariado, a menor distância média de todos os objetos num espaço multivariado

Chao 1: estimador de riqueza de espécies baseado na abundância das espécies *singleton* e *doubletons* dentro de uma amostra

Chao 2: estimador de riqueza de espécies baseado na incidência (presença e ausência) das espécies *singleton* e *doubletons* dentro de uma amostra

Clado: um grupo de espécies aparentadas descendendo de um único nó na filogenia

Coeficiente de correlação: indica a força da relação linear entre as duas variáveis

Coerção (linguagem R): transformação dos modos dos elementos seguindo uma hierarquia: character > double > integer > logical

Combinação linear: Combinação de várias variáveis (vetores) que são multiplicadas por constantes e adicionadas a outras variáveis. Por exemplo: combinação linear de x, y, z pode ser escrita como ax+by+cz.

Community Mean Nearest Taxon Distance (COMDISTNT): métrica de diversidade beta que calcula a média da distância filogenética/funcional entre o táxon mais próximo das espécies de duas comunidades

Community Mean Pairwise Distance (COMDIST): métrica de diversidade beta que calcula a média da distância filogenética/funcional entre as espécies de duas comunidades

Console (RStudio): onde a versão da linguagem R instalada é carregada para executar os códigos no RStudio

Conversão (linguagem R): transformação dos modos ou estrutura dos elementos de um objeto a partir de funções específicas

Conversões raster-vetor: transformar dados vetoriais em rasters (rasterização) e transformar rasters em vetores (vetorização)

Convex Hull: medida multivariada derivada da computação geométrica que calcula o espaço dos atributos de uma espécie ou de várias espécies em uma comunidade

Correlação: é um teste que mede a força relativa da relação linear entre duas variáveis contínuas. A análise de correlação não assume que a variável X influencie a variável Y, ou que exista uma relação de causa e efeito entre elas

Cortes e máscaras (raster): ajusta o tamanho de um dado raster a uma área menor de interesse, geralmente definido por um dado vetorial

Covariável: variável contínua que potencialmente afeta a variável resposta, mas não é necessariamente controlada ou manipulada pelo pesquisador

Critério de Informação de Akaike (*Akaike Information Criterion* – AIC): é uma métrica para seleção de modelos. Ele é usado para determinar qual entre os múltiplos modelos é o mais provável de prever os dados observados, ponderando pelo número de parâmetros dos modelos. Os menores valores de AIC representam modelos com melhores ajustes aos dados

Curva de dominância: veja Diagrama de Whittaker

D

Dados geoespaciais: são dados georreferenciados que expressão informações espaciais, podendo ser no formato vetorial ou matricial (raster)

Dados matriciais (raster): consistem em uma matriz (com linhas e colunas) em que os elementos representam células, geralmente igualmente espaçadas (pixels)

Dados tidy: dados organizados, com foco na limpeza e organização dos mesmos, de modo que os dados estão *tidy* quando: i) variáveis estão nas colunas, ii) observações estão nas linhas e iii) valores estão nas células, sendo que para esse último, não deve haver mais de um valor por célula

Dados vetoriais: são representações geométricas (pontos, linhas e polígonos) usadas para mapear fenômenos ou objetos espacialmente explícitos que possuem localização ou dimensões bem definidas, aos quais são atribuídas informações tabulares

Data frame: classe de objetos que representa dados no formato de tabela, com linhas e colunas, mas comportam mais de um modo em suas colunas

Data Science: nova área de conhecimento que vem se moldando a partir do desenvolvimento da sociedade em torno da era digital e da grande quantidade de dados gerados e disponíveis pela internet

Datum: simplificadamente é a relação do sistema de coordenadas (geográfica ou projetada) com a superfície da Terra

Dendrograma: diagrama representando uma árvore que organiza elementos, objetos e variáveis por suas semelhanças de maneira hierárquica ascendente. Desse modo, objetos mais próximos compartilham maior semelhança do que objetos distantes no dendrograma.

Desagregação (raster): diminui o tamanho dos pixels (aumentando a resolução) de um raster, preenchendo com novos valores que depende do tipo de função de preenchimento utilizada

Deviance: é um termo estatístico que mede o ajuste do modelo (*goodness of fit*). Quanto menor o valor de *deviance* melhor o modelo

Diagrama de Whittaker: método que utiliza informações visuais ao plotar as espécies ranqueadas no eixo X da mais abundante para a menos abundante, enquanto no eixo Y as abundâncias relativas das espécies são plotadas em escala logarítmica

Diretório de trabalho: endereço da pasta (ou diretório) de onde o R importará ou exportar dados

Dispersão filogenética: espécies coexistindo nas comunidades são menos aparentadas do que esperado pelo acaso

Distribuição de probabilidade: é uma função estatística que descreve todos os valores e probabilidades possíveis que uma variável aleatória pode assumir dentro de um determinado intervalo

Distribuição Gaussiana: veja distribuição normal

Distribuição normal: é uma distribuição de probabilidade em formato de sino que é simétrica em torno da média, mostrando que dados próximos à média são mais frequentes que dados distantes da média.

Diversidade alfa: é um conceito caracterizado pela diversidade dentro do habitat ou unidade amostral

Diversidade beta: é um conceito caracterizado pela variação na diversidade entre habitats ou unidades amostrais

Diversidade beta filogenética: engloba métricas que utilizam dados de presença e ausência ou abundância das espécies para determinar um valor que representa a diferença entre comunidades em relação a história evolutiva das linhagens

Diversidade de espécies: é um conceito que representa o número de espécies e a distribuição de abundância destas espécies em uma comunidade

Diversidade filogenética: engloba métricas que capturam a ancestralidade compartilhada entre as espécies em termos de quantidade da história evolutiva e o grau de parentesco entre as espécies

Diversidade funcional: é um conceito que captura a variação no grau de expressão de diferentes atributos funcionais entre diferentes populações, comunidades ou ecossistemas

Diversidade gama: é um conceito caracterizado pela combinação da diversidade alfa e beta ou definido como a diversidade regional englobando todos os habitat ou unidades amostrais

Doubletons: número de espécies observadas com abundância de dois indivíduos

Duplicate: número de espécies observadas em apenas duas amostras

E

Equitabilidade de Pielou: é uma métrica que descreve o padrão de distribuição da abundância relativa das espécies na comunidade

Erro do Tipo I: rejeitar a hipótese nula de um teste estatístico quando ela é verdadeira

Erro do Tipo II: aceitar a hipótese nula de um teste estatístico quando ela é falsa

Escalar: número inteiro com o qual geralmente se faz operações com matrizes (e.g., multiplicação ou adição)

Escores: posição das unidades amostrais ao longo de um eixo de ordenação. Pode se referir tanto a objetos quanto à variáveis. Escores são fornecidos pela substituição dos valores assumidos pelas variáveis originais nas combinações lineares. São utilizados para ordenar as unidades amostrais em um diagrama uni, bi ou tridimensional

ESRI Shapefile (vetor): principal formato de dados vetoriais, composto por pelo menos quatro arquivos: .shp (feição), .dbf (tabela de atributos), .shx (ligação entre .shp e .dbf) e .prj (projeção)

Estatística frequentista: são análises paramétricas que estimam probabilidades das frequências observadas dos eventos e usam essas probabilidades como base para inferências.

Estrutura dos objetos: diz respeito à organização dos elementos, com relação aos modos e dimensionalidade da disposição desses elementos. De modo bem simples, os elementos podem ser estruturados em seis tipos: i) vetor, ii) fator, iii) matriz, iv) array, v) data frame e vi) listas

Extensão (vetor e raster): limites geográficos de dados geoespaciais (vetor e raster), composto por dois pares de coordenadas de longitude e de latitude

Extração (vetor e raster): identifica e retorna valores associados de pixels de um raster com base em um objeto vetorial (ponto, linha e polígono)

Extrapolação: é o processo de estimar, além do intervalo de observação original, o valor de uma variável com base em sua relação com outra variável

F

Fator: classe de objetos que representa o encadeamento de elementos de um único modo (integer) numa sequência unidimensional, representando medidas de uma variável categórica, podendo ser nominal ou ordinal

Fator aleatório: pesquisador amostra aleatoriamente os níveis de um fator na população

Fator fixo: pesquisador controla todos os níveis do fator sobre os quais as inferências devem ser feitas

Fator de inflação da variância (VIF): é um teste que quantifica quanto do erro padrão dos coeficientes estimados estão inflados devido à multicolinearidade

Feição (vetor): um elemento do dado vetorial associado à cada linha da tabela de atributos

Fenômeno: Um evento, entidade ou relação observável

Funções: classe de objetos que possui códigos preparados para realizar uma tarefa específica de modo simples, realizando operações em argumentos

G

Generalized Least Squares (GLS): é um teste estatístico utilizado para estimar coeficientes desconhecidos de um modelo de regressão linear quando a variável independente é correlacionada com os resíduos

GeoPackage (vetor e raster): banco de dados geoespacial que armazena em apenas um arquivo, dados no formato vetorial, raster e também dados não-espaciais (e.g., tabelas)

GeoTIFF (raster): principal formato para dados raster, composto geralmente de um arquivo TIFF contendo metadados geoespaciais adicionais

GitHub: é um repositório de hospedagem de código-fonte e arquivos com controle de versões de projetos abertos

Goodness of fit: refere-se a um teste estatístico que determina quão bem os dados da amostra se ajustam a uma distribuição de uma população

Grau de liberdade: é o número de observações nos dados que são livres para variar quando estimamos os valores dos parâmetros populacionais desconhecidos

H

Hipótese: Afirmação testável derivada ou representando vários componentes de uma teoria

Hipótese alternativa (H1): é um conceito estatístico que sugere que diferenças entre grupos ou fenômenos medidos são maiores do que o esperado devido a variações aleatórias. A H1 é o oposto da hipótese nula.

Hipótese nula (H0): é um conceito estatístico que sugere que diferenças entre grupos ou fenômenos medidos não são maiores do que o esperado devido a variações aleatórias. Presume-se que a hipótese nula é verdadeira até que evidências indiquem o contrário.

Homocedasticidade: é um dos pressupostos dos testes paramétricos como ANOVA, Teste T e regressão linear simples que a variância da variável dependente deve ser constante para os valores das variáveis preditoras. Sinônimo de homogeneidade da variância

Homogeneidade da variância: veja homocedasticidade

I

Independência estatística: dois eventos são independentes se a ocorrência de um evento não influenciar a probabilidade que o outro evento irá ou não ocorrer

Indexação (linguagem R): acessa elementos de objetos por sua posição utilizando os operadores `[]` e `[[]]` ou por seu nome com o operador `$` depois do nome do objeto

Índice de diversidade: métricas que calculam a riqueza de espécies e a distribuição de abundância para cada espécie dentro das comunidades.

Índice de Gini-Simpson: métrica que quantifica a probabilidade de dois indivíduos retirados ao acaso da comunidade pertencerem a espécies diferentes. É o inverso do índice de Simpson

Índice de Margalef: métrica que calcula a riqueza de espécies ponderando a abundância total dentro de cada comunidade

Índice de Menhinick: métrica que calcula a riqueza de espécies ponderando a abundância total dentro de cada comunidade

Índice de Shannon-Wiener: métrica de diversidade de espécies que quantifica a incerteza associada em predizer a identidade de uma espécie dado o número de espécies e a distribuição de abundância para cada espécie

Índice de Simpson: métrica que quantifica a probabilidade de dois indivíduos retirados ao acaso da comunidade pertencerem à mesma espécie

Inf (Infinito): é um número muito grande ou um limite matemático

Interação raster-vetor: operações derivadas da interação entre raster-vetor, como ajuste do tamanho do raster ou extração dos valores dos pixels para dados vetoriais (pontos, linhas e polígonos)

Interpolação: é o processo de estimar, dentro um domínio de valores conhecidos, um valor desconhecido com base em sua relação com outra variável

J

Jackknife 1: estimador de riqueza de espécies baseado no número de espécies que ocorrem em somente uma amostra (*uniques*)

Jackknife 2: estimador de riqueza de espécies baseado no número de espécies que ocorrem em somente uma amostra (*uniques*) e no número de espécies ocorrem em exatamente duas amostras (*duplicates*)

Junção de tabelas: combinação de pares de conjunto de dados tabulares por uma ou mais colunas chaves

L

Likelihood-ratio test (LRT): é um teste estatístico que mede o grau do ajuste (*goodness-of-fit*) entre dois modelos aninhados. Um modelo relativamente mais complexo é comparado a um modelo mais simples para ver se ele se ajusta significativamente melhor a um determinado conjunto de dados. Ele testa se há necessidade de se incluir uma variável extra no modelo para explicar os dados

Linguagem R: ambiente de software livre para computação estatística e criação de gráficos

Lista: classe de objetos que é um tipo especial de vetor que aceita objetos como elementos

Loading: Correlações de Pearson entre cada variável original com cada eixo. Os valores serão maiores (positiva ou negativamente) para aquelas variáveis que forem mais importantes na formação de um dado eixo. Desse modo, representam o peso de uma variável para a construção de um eixo e variam de -1 a 1.

Loop for: função em que um bloco de códigos é repetido mudando um contador de uma lista de possibilidades

M

Mapa: representação bidimensional de elementos geoespaciais em uma proporção menor, podendo representar dados vetoriais ou raster, com diversos elementos visuais e textuais que facilitam a interpretação dos elementos geoespaciais mapeados

Mapas animados: mapas que incorporam animações para expressar mudanças nos padrões espaciais ou ao longo do tempo

Mapas estáticos: mapas simples e fixos para visualização de dados, sendo o tipo mais comum de saída visual

Mapas interativos: mapas que incorporam a capacidade de deslocar e ampliar qualquer parte de um conjunto de dados geoespaciais sobreposto em um "mapa da web"

Matriz: classe de objetos que representa elementos de um único modo no formato de tabela, com linhas e colunas

Máxima Verossimilhança: é um método que determina valores para os parâmetros de um modelo. Os valores dos parâmetros são encontrados de tal forma que maximizam a probabilidade de que o processo descrito pelo modelo produza os dados que foram realmente observados

Mean Pairwise Distance (MPD): é uma métrica que utiliza a matriz de distância filogenética para quantificar a distância média do parentesco entre pares de espécies em uma comunidade

Mean Nearest Taxon Distance (MNTD): é uma métrica que utiliza a matriz de distância filogenética para quantificar a média dos valores mínimos de parentesco entre pares de espécies em uma comunidade. Ou seja, qual o valor médio da distância para o vizinho mais próximo

Mecanismo: Interação direta de uma relação causal que resulta em um fenômeno

Modo dos objetos: diz respeito à natureza dos elementos que compõem os dados e que foram atribuídos aos objetos. Os modos geralmente são: numérico do tipo inteiro (integer), numérico do tipo flutuante (double), texto (character), lógico (logical) ou complexo (complex)

Modelo misto: pelo menos um fator no experimento é fixo, e pelo menos um fator é aleatório.

Modelo nulo: é um procedimento estatístico que usa aleatorizações para gerar distribuições de valores para uma determinada variável de interesse na ausência do processo causal em questão

Multicolinearidade: é um conceito estatístico onde variáveis independentes em um modelo são correlacionadas

N

NA (Not Available): significa dado faltante ou indisponível

NaN (Not a Number): representa indefinições matemáticas

Nearest Relative Index (NRI): métrica que calcula o tamanho do efeito padronizado para a métrica *Mean Pairwise Distance*.Valores positivos de NRI indicam agrupamento filogenético enquanto valores negativos de NRI indicam dispersão filogenética

Nearest Taxon Index (NTI): métrica que calcula o tamanho do efeito padronizado para a métrica *Mean Nearest Taxon Distance*.Valores positivos de NTI indicam agrupamento filogenético enquanto valores negativos de NTI indicam dispersão filogenética

NetCDF (raster): *Network Common Data Form* é um conjunto de bibliotecas de software e formatos de dados independentes que suportam a criação, acesso e compartilhamento de dados científicos orientados a arrays

Nó: o ponto onde uma linhagem dá origem a duas ou mais linhagens descendentes

Normalidade dos resíduos: é um dos pressupostos dos testes paramétricos como ANOVA, Teste T e regressão linear simples que dependem que os resíduos do modelo apresentem distribuição normal ou gaussiana.

NULL (Nulo): representa um objeto nulo, sendo útil para preenchimento em aplicações de programação

Números de Hill: é uma métrica que transforma a riqueza e a distribuição da abundância das espécies em números efetivos de espécies ou diversidade verdadeira

Número efetivo de espécies: é o número de espécies igualmente abundantes (i.e., todas as espécies com a mesma abundância) necessárias para produzir o valor observado para um determinado índice

O

Objetos: palavras às quais são atribuídos dados através da atribuição. A criação de objetos possibilita a manipulação de dados ou armazenar os resultados de análises. Em outras linguagens de programação são denominados variáveis

Operações geoespaciais: são operações para acessar ou alterar as propriedades não-espaciais, espaciais e geométricas dos dados geoespaciais, divididas em: operações de atributos, operações espaciais e operações geométricas

Operações geoespaciais de atributos: modificação de objetos geoespaciais baseado em informações não espaciais associadas a dados geoespaciais, como a tabela de atributos ou valores das células e nome das camadas dos rasters

Operações geoespaciais espaciais: modificações de objetos geoespaciais baseado em informações espaciais, como localização e formato

Operações geoespaciais geométricas: modificações em objetos geoespaciais baseado na geometria do vetor ou do raster e na interação e conversão entre vetor-raster

Operadores: conjuntos de caracteres que realiza operações, agrupados em cinco tipos principais: aritméticos, relacionais, lógicos, atribuição e diversos

P

Pacotes (linguagem R): conjuntos extras de funções para executar tarefas específicas

Padrão: Eventos repetidos, entidades recorrentes ou relações replicadas no tempo ou no espaço

Phylogenetic Diversity (PD): é uma métrica definida pela soma do comprimento dos ramos conectando todas as espécies na comunidade

Phylogenetic Endemism (PE): é uma métrica que calcula a fração dos ramos restritos a regiões específicas

Phylogenetic index of beta diversity (Phylosor): métrica de similaridade que determina o comprimento total dos ramos da filogenia que é compartilhado entre pares de comunidades

Phylogenetic Species Richness (PSR): é uma métrica diretamente comparável ao número de espécies na comunidade, mas inclui o parentesco filogenético entre as espécies

Phylogenetic Species Variability (PSV): é uma métrica que estima a quantidade relativa dos comprimentos dos ramos não compartilhados entre as comunidades

Pipe (%>%): operador implementado por uma função que faz com o que o resultado de uma função seja o primeiro argumento da função seguinte, permitindo o encadeamento de várias funções eliminando a necessidade de criar objetos para armazenar resultados intermediários

Pivotagem de dados: transporte de dados que estão em linhas para colunas e vice-versa, fazendo a referência cruzada ou rotacionando os dados. Partirmos de dados no formato longo (*long*, muitas linhas e poucas colunas) e criamos dados no formato largo (*wide*, poucas linhas e muitas colunas) e vice-versa

Pixel (raster): também chamado célula, é a unidade geoespacial do raster, representando o elemento da matriz de dados

Polígono convexo: operação que liga os pontos externos de um conjunto de pontos e cria um polígono a partir deles

Polígono de Voronoi: polígonos irregulares são criados a partir da proximidade dos pontos, de modo a estimar uma área de abrangência no entorno dos mesmos

Politomia: três ou mais linhagens descendendo de um único nó

População: é um conjunto de indivíduos ou elementos semelhantes que interessa para alguma pergunta ou hipótese

População amostral: é um conjunto de indivíduos ou elementos semelhantes que estão de fato acessível para serem amostrados

Predição espacial: utiliza os coeficientes de um modelo ajustado para gerar um raster de predição para todos os pixels considerando os dados preditores de entrada do modelo, extrapolando a predição da resposta

Predição: uma declaração de expectativa deduzida da estrutura lógica ou derivada da estrutura causal de uma teoria

Pressuposto: condições necessárias para sustentar uma hipótese ou construção da teoria

Principais elementos de um mapa: um mapa possui diversos elementos que facilitam sua interpretação, dentre eles: i) mapa principal, ii) mapa secundário iii) título, iv) legenda (apresentando as informações detalhadas das classes ou escala de valores, v) barra de escala, vi) indicador de orientação (Norte), vii) gride de coordenadas e viii) descrição do CRS

Processo: um subconjunto de fenômenos em que os eventos seguem uns aos outros no tempo ou espaço, que podem ou não serem causalmente conectados. É a causa, mecanismo ou limitação explicando um padrão

Programação Funcional: organização do código como funções e variáveis que trabalham de forma unificada para a resolução de um problema

Projeto do RStudio: arquivo no formato `.Rproj` que facilita o trabalho com o RStudio, pois define o diretório automaticamente e permite o controle de versão

Pull request: é um método de submeter contribuições que serão revisadas pelos responsáveis de um projeto de desenvolvimento aberto

R

Raiz: representa o ancestral comum de todas as espécies na filogenia

Ramo: uma linha orientada ao longo de um eixo terminais-raiz que conecta os nós na filogenia

Rarefação: é uma métrica usada para calcular o número esperado de espécies em cada comunidade tendo como base comparativa um valor em que todas as amostras ou número de indivíduos atinjam um tamanho padrão entre as comunidades

Rarefação baseada nas amostras (*Sampled-based*): as comparações são padronizadas pela comunidade com menor número de amostragens

Rarefação baseada na cobertura (*Coverage-based*): é uma medida que determina a proporção de amostras ou do número de indivíduos da comunidade que representa as espécies amostradas

Rarefação baseada nos indivíduos (*Individual-based*): as comparações são feitas considerando a abundância da comunidade padronizada pelo menor número de indivíduos

Rasterização: conversão de dados vetoriais para raster realizada de pontos, linhas ou polígonos para rasters

Regressão: é um teste usado para analisar a relação entre uma ou mais variáveis preditoras (Xn) e uma variável resposta (Y). A regressão assume uma relação de causa e efeito entre as variáveis

Reprojeção: transformação do Sistema de Referência de Coordenadas (CRS) de um dado geoespacial, alterando o CRS original para outro. Na reprojeção alteramos tanto a unidade do dado geoespacial (unidades em 'longitude/latitude' ou unidades de metros), quando o *Datum*

Resíduo: é a diferença entre os valores preditos e observados dos dados. São também chamados de erro

Resolução (raster): termo amplo que pode ser usado em diversos contextos. Para dados raster, é geralmente associado ao tamanho (dimensão) do pixel (i.e., altura e largura)

RStudio: ambiente de desenvolvimento integrado (IDE) para R. Inclui um console, editor de realce de sintaxe que suporta execução direta de código, bem como ferramentas para plotagem, histórico, depuração e gerenciamento de espaço de trabalho

S

Script (RStudio): arquivos de texto simples, criados com a extensão (terminação) `.R` onde os códigos são escritos e salvos

Seleção de modelos: é um processo usado para comparar o valor relativo de diferentes modelos estatísticos e determinar qual deles é o mais adequado para os dados observados

Série de Hill: veja Número de Hill

Singleton: número de espécies observadas com abundância de um indivíduo

Sistema de Coordenadas: composto por dois sistemas, o Sistema de Coordenadas Geográficas e o Sistema de Coordenadas Projetadas, que utilizam ângulos ou metros, respectivamente

Sistema de Referência de Coordenadas (CRS): também chamada projeção, define a referência geoespacial dos dados vetor e raster na superfície da Terra, composto pelo Sistema de Coordenadas e *Datum*

T

Tabela de atributos: dado tabular que inclui dados geoespaciais e dados alfanuméricos, geralmente associada a dados vetoriais

Terminal (do inglês tip): o final do ramo representando uma espécie atual ou extinta (pode também representar gêneros, indivíduos, genes, etc.)

Teste de Levene: é um teste estatístico utilizado para verificar a homogeneidade da variância entre dois ou mais grupos

Teste de Shapiro-Wilk: é um teste estatístico utilizado para verificar se os dados apresentam distribuição normal

Teste T (de Student): é um teste estatístico que segue uma distribuição t de *Student* para rejeitar ou não uma hipótese nula de médias iguais entre dois grupos

Teste T pareado: é um teste estatístico que usa dados medidos duas vezes na mesma unidade amostral, resultando em pares de observações para cada amostra (amostras pareadas). Ele determina se a diferença da média entre duas observações é zero

Tibble: classe de objetos que é uma versão aprimorada do data frame. É a classe aconselhada para que as funções do *tidyverse* funcionem melhor sobre conjuntos de dados tabulares

tidyverse: "dialeto novo" para a linguagem R, onde *tidy* quer dizer organizado, arrumado, ordenado, e *verse* é universo. Operacionalizado no R através de uma coleção de pacotes que atuam no fluxo de trabalho comum da ciência de dados: importação, manipulação, exploração, visualização, análise e comunicação de dados e análises. O principal objetivo do *tidyverse* é aproximar a linguagem para melhorar a interação entre ser humano e computador sobre dados, de modo que os pacotes

compartilham uma filosofia de design de alto nível e gramática, além da estrutura de dados de baixo nível

U

Ultramétrica: a distância de todos os terminais até a raiz são idênticas. Característica requerida pela maioria dos índices de diversidade filogenética

Unidade amostral: é o indivíduo (ou elemento) da população amostral sobre o qual a medida de interesse será observada

Unique: número de espécies observadas em apenas uma amostra

Unique Fraction metric (UniFrac): métrica de dissimilaridade que determina a fração única da filogenia contida em cada uma das duas comunidades

V

Valor de P: probabilidade de um teste estatístico ser igual ou maior que o observado, dado que a hipótese nula é verdadeira

Valores faltantes e especiais: valores reservados que representam dados faltantes, indefinições matemáticas, infinitos e objetos nulos

Variance of Pairwise Distance (VPD): é uma métrica que utiliza a matriz de distância filogenética para quantificar a variância do parentesco entre pares de espécies em uma comunidade

Variável categórica: são variáveis que não possuem valores quantitativos e são definidas por categorias ou grupos distintos

Variável contínua: são variáveis numéricas que têm um número infinito de valores entre dois valores quaisquer. Neste caso, valores fracionais fazem sentido

Variável dependente: é uma variável mensurada ou observada de interesse do pesquisador que depende do valor de outra variável

Variável discreta: é uma variável numérica que têm um número contável de valores inteiros

Variável explicativa: ver variável independente

Variável independente: variável mensurada ou observada pelo pesquisador que prevê ou afeta a variável dependente

Variável nominal: são variáveis categóricas que não apresentam ordenação dentre as categorias (e.g., preto, branco e rosa)

Variável ordinal: são variáveis categóricas com ordenação entre as categorias (e.g., pequeno, médio e grande)

Variável preditora: ver variável independente

Variável resposta: ver variável dependente

Variograma: é uma descrição da continuidade espacial dos dados. Ele mede a variabilidade entre pares de pontos em várias distâncias

Vetor: classe de objetos que representa o encadeamento de elementos de um único modo numa sequência unidimensional

Vetor (Multivariada): coluna de uma matriz

Vetorização (vetor): conversão de dados rasters para vetor, sendo que esse vetor receberá os valores dos pixels do raster, podendo ser convertido em pontos, isolinhas ou polígonos

Referências

Agresti A. 2010: Analysis of ordinal categorical data. Hoboken, NJ, USA, John Wiley & Sons, Inc.

Agresti A. 2012: Categorical Data Analysis. 3ª edição. Hoboken, NJ, Wiley.

Albuquerque UP. 2013. How to improve the quality of scientific publications in ethnobiology. Ethnobiology and Conservation 2: 1-5.

Albuquerque UP, Ferreira Júnior WS. 2017. What Do we study in evolutionary ethnobiology? Defining the theoretical basis for a research program. Evolutionary Biology 44: 206-215.

Albuquerque UP, Hanazaki N. 2009. Five problems in current ethnobotanical research — and some suggestions for strengthening them. Human Ecology 37: 653-661.

Altieri MA, Nicholls CI. 2017. The adaptation and mitigation potential of traditional agriculture in a changing climate. Climatic Change 140: 33-45.

Anderson DT. 1967. Larval development and segment formation in the branchipod crustaceans Limnadia stanleyana King (Conchostraca) and Artemia salina (L.) (Anostraca). Australian Journal of Zoology 15: 47-91.

Anderson MJ. 2001. A new method for non-parametric multivariate analysis of variance.: Austral Ecology 26: 32-46.

Anderson MJ. 2017. Permutational Multivariate Analysis of Variance (PERMANOVA). Wiley StatsRef: Statistics Reference Online: 1–15.

Anderson MJ, Crist TO, Chase JM, Vellend M, Inouye BD, Freestone AL, Sanders NJ, Cornell HV, Comita LS, Davies KF, Harrison SP, Kraft NJB, Stegen JC, Swenson NG. 2011. Navigating the multiple meanings of diversity: A roadmap for the practicing ecologist. Ecology Letters 14: 19-28.

Anderson MJ, Ellingsen KE, McArdle BH. 2006. Multivariate dispersion as a measure of beta diversity. Ecology Letters 9: 683-693.

Anderson MJ, Gorley RN, Clarke KR. 2008: PERMANOVA+ for PRIMER: Guide to Software and Statistical Methods. UK, Primer-E.

Anderson MJ, Walsh DCI. 2013. PERMANOVA, ANOSIM, and the Mantel test in the face of heterogeneous dispersions: What null hypothesis are you testing? Ecological Monographs 83: 557-574.

Anderson MJ, Ter Braak C. 2003. Permutation tests for multi-factorial analysis of variance. Journal of Statistical Computation and Simulation 73: 85-113.

Arino O, Ramos Perez JJ, Kalogirou V, Bontemps S, Defourny P, Van Bogaert E. 2012. Global land cover map for 2009. European Space Agency (ESA) & Université Catholique de Louvain (UCL), PANGAEA.

Arlidge SM, Thanukos A, Bean JR. 2017. Using the understanding science flowchart to illustrate and bring students' science stories to life. The Bulletin of the Ecological Society of America 98: 211-226.

Barbaro L, Van Halder I. 2009. Linking bird, carabid beetle and butterfly life-history traits to habitat fragmentation in mosaic landscapes. Ecography 32: 321-333.

Barber JJ, Ogle K. 2014. To P or not to P? Ecology 95: 621-626.

Baselga A. 2010. Partitioning the turnover and nestedness components of beta diversity. Global Ecology and Biogeography 19: 134-143.

Baselga A. 2013. Separating the two components of abundance-based dissimilarity: balanced changes in abundance vs. abundance gradients. Methods in Ecology and Evolution 4: 552-557.

Baselga A. 2012. The relationship between species replacement, dissimilarity derived from nestedness, and nestedness. Global Ecology and Biogeography 21: 1223-1232.

Baselga A, Orme D, Villeger S, De Bortoli J, Leprieur F, Logez M. 2021. Betapart: Partitioning beta diversity into turnover and nestedness components. R package version 1.5.4

Bauman D, Drouet T, Dray S, Vleminckx J. 2018. Disentangling good from bad practices in the selection of spatial or phylogenetic eigenvectors. Ecography 41: 1638-1649.

Bauman D, Drouet T, Fortin MJ, Dray S. 2018. Optimizing the choice of a spatial weighting matrix in eigenvector-based methods. Ecology 99: 2159-2166.

Bello F, Carmona CP, Dias ATC, Götzenberger L, Moretti M, Berg MP. 2021. Handbook of Trait-Based Ecology. Cambridge University Press.

Benjamin MB, Mollie EB, Connie JC, Shane WG, John RP, Henry MHS, Jada-Simone SW. 2009. Generalized linear mixed models: A practical guide for ecology and evolution.: Trends in Ecology and Evolution 24: 127-135.

Birch CPD, Oom SP, Beecham JA. 2007. Rectangular and hexagonal grids used for observation, experiment and simulation in ecology. Ecological Modelling 206: 347-359.

Blanchard JL, Watson RA, Fulton EA, Cottrell RS, Nash KL, Bryndum-Buchholz A, Büchner M, Carozza DA, Cheung WWL, Elliott J, Davidson LNK, Dulvy NK, Dunne JP, Eddy TD, Galbraith E, Lotze HK, Maury O, Müller C, Tittensor DP, Jennings S. 2017. Linked sustainability challenges and trade-offs among fisheries, aquaculture and agriculture. Nature Ecology and Evolution 1: 1240-1249.

Blasco-Moreno A, Pérez-Casany M, Puig P, Morante M, Castells E. 2019. What does a zero mean? Understanding false, random and structural zeros in ecology. Methods in Ecology and Evolution 10: 949-959.

Boaratti AZ, Da Silva FR. 2015: Relationships between environmental gradients and geographic variation in the intraspecific body size of three species of frogs (Anura). Austral Ecology 40: 869-876.

Bolnick DI, Amarasekare P, Araújo MS, Bürger R, Levine JM, Novak M, Rudolf VHW, Schreiber SJ, Urban MC, Vasseur DA. 2011. Why intraspecific trait variation matters in community ecology. Trends in Ecology and Evolution 26: 183-192.

Borcard D, Gillet F, Legendre P. 2018. Numerical Ecology with R. Springer International Publishing.

Borcard D, Legendre P, Drapeau P. 1992. Partialling out the spatial component of ecological variation.: Ecology 73: 1045-1055.

Breviglieri CPB. 2008: Diversidade de morcegos (Chiroptera; Mammalia) em três áreas do noroeste paulista, com ênfase nas relações tróficas em Phyllostomidae. PhD Thesis. State University of São Paulo, São José do Rio Preto.

Brooks ME, Kristensen K, Benthem KJV, Magnusson A, Berg CW, Nielsen A, Skaug HJ, Mächler M, Bolker BM. 2017. glmmTMB Balances speed and flexibility among packages for zero-inflated generalized linear mixed modeling. The R Journal 9: 378.

Brunsdon C, Comber L. 2015. An introduction to R for spatial analysis & mapping. Los Angeles, SAGE.

Bryant JA, Lamanna C, Morlon H, Kerkhoff AJ, Enquist BJ, Green JL. 2008. Microbes on mountainsides: Contrasting elevational patterns of bacterial and plant diversity. Proceedings of the National Academy of Sciences 105: 11505-11511.

Burnham KP, Anderson DR. 2002. Model Selection and Multimodel Inference. Springer.

Burnham KP, Anderson DR. 2014. P values are only an index to evidence: 20th- vs. 21st-century statistical science. Ecology 95: 627-630.

Burnham KP, Overton WS. 1978. Estimation of the size of a closed population when capture probabilities vary among animals. Biometrika 65: 625-633.

Burnham KP, Overton WS. 1979. Robust estimation of population size when capture probabilities vary among animals. Ecology 60: 927-936.

Cadotte MW, Davies TJ. 2016. Phylogenies in Ecology. Princeton University Press.

Carstensen DW, Lessard JP, Holt BG, Borregaard MK, Rahbek C. 2013. Introducing the biogeographic species pool. Ecography 36: 1310-1318.

Chaffin BC, Garmestani AS, Angeler DG, Herrmann DL, Stow CA, Nyström M, Sendzimir J, Hopton ME, Kolasa J, Allen CR. 2016. Biological invasions, ecological resilience and adaptive governance.: Journal of Environmental Management 183: 399-407.

Chang W. 2018: R graphics cookbook: Practical recipes for visualizing data. 2nd. edn. Beijing, Boston, O'Reilly.

Chao A. 1984. Nonparametric estimation of the number of classes in a population. Scandinavian Journal of Statistics 11: 265-270.

Chao A. 1987. Estimating the population size for capture-recapture data with unequal catchability. Biometrics 43: 783-791.

Chao A, Gotelli NJ, Hsieh TC, Sander EL, Ma KH, Colwell RK, Ellison AM. 2014. Rarefaction and extrapolation with Hill numbers: a framework for sampling and estimation in species diversity studies. Ecological Monographs 84: 45-67.

Chao A, Hwang WH, Chen YC, Kuo CY. 2000. Estimating the number of shared species in two communities. Statistica Sinica 10: 227-246.

Chao A, Jost L. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93: 2533-2547.

Chao A, Lee SM. 1992. Estimating the number of classes via sample coverage.: Journal of the American Statistical Association 87: 210-217.

Chazdon RL, Colwell RK, Denslow JS, Guariguata MR. 1998. Statistical methods for estimating species richness

of woody regeneration in primary and secondary rain forest of northeastern Costa Rica. In: Dallmeier F, Comiskey JA (eds.). Forest biodiversity, research. Monitoring and modeling: Conceptual background and old world case studies. Paris; New York, UNESCO; The Parthenon Publishing Group. p. 285-309

Chiles JP, Delfiner P. 1999. Geostatistics: Modeling spatial uncertainty. Wiley.

Clarke KR, Warwick RM. 2001. Change in marine communities: an approach to statistical analysis and interpretation. 2nd. edn. Primer-E, Plymouth Marine Laboratory, Plymouth.

Colwell RK, Chao A, Gotelli NJ, Lin SY, Mao CX, Chazdon RL, Longino JT. 2012. Models and estimators linking individual-based and sample-based rarefaction, extrapolation and comparison of assemblages. Journal of Plant Ecology 5: 3-21.

Colwell RK, Coddington JA. 1994. Estimating terrestrial biodiversity through extrapolation.: Philosophical Transactions of the Royal Society of London B 345: 101-118.

Cornwell WK, Schwilk DW, Ackerly DD. 2006. A trait-based test for habitat filtering: convex hull volume. Ecology 87: 1465-1471.

Crabot J, Clappe S, Dray S, Datry T. 2019. Testing the mantel statistic with a spatially-constrained permutation procedure. Methods in Ecology and Evolution 10: 532-540.

Crawley MJ. 2012. The R Book. Chichester, UK, John Wiley & Sons, Ltd.

da Silva FR, Oliveira TAL, Gibbs JP, Rossa-Feres DC. 2012. An experimental assessment of landscape configuration effects on frog and toad abundance and diversity in tropical agro-savannah landscapes of southeastern Brazil. Landscape Ecology 27: 87-96.

da Silva FR, Provete DB, Gerassi LK, Bovo RP. 2017. What do data from fieldwork and scientific collections tell us about species richness and composition of amphibians and reptiles? South American Journal of Herpetology 12: 99-106.

da Silva FR, Rossa-Feres DC. 2010. Seasonal variation in body size of tropical anuran amphibians. Herpetology Notes 3: 205-209.

Dalmolin DA, Tozetti AM, Pereira MJR. 2020. Turnover or intraspecific trait variation: explaining functional variability in a neotropical anuran metacommunity. Aquatic Sciences 82: 62.

Davies TM. 2016. The book of R: A first course in programming and statistics. San Francisco, No Starch Press.

Dawson W, Moser D, Kleunen MV, Kreft H, Pergl J, Pyšek P, Weigelt P, Winter M, Lenzner B, Blackburn TM, Dyer EE, Cassey P, Scrivens SL, Economo EP, Guénard B, Capinha C, Seebens H, García-Díaz P, Nentwig W, García-Berthou E, Casal C, Mandrak NE, Fuller P, Meyer C, Essl F. 2017. Global hotspots and correlates of alien species richness across taxonomic groups. Nature Ecology & Evolution 1: 0186.

Diniz-Filho JAF, de Sant'Ana CER, Bini LM. 1998. An eigenvector method for estimating phylogenetic inertia.: Evolution 52: 1247-1262.

Diniz-Filho JAF, Siqueira T, Padial AA, Rangel TF, Landeiro VL, Bini LM. 2012. Spatial autocorrelation analysis allows disentangling the balance between neutral and niche processes in metacommunities. Oikos 121: 201-210.

Díaz S, Cabido M. 2001. Vive la différence: plant functional diversity matters to ecosystem processes. Trends in Ecology & Evolution 16: 646-655.

Dormann CF, Elith J, Bacher S, Buchmann C, Carl G, Carré G, Marquéz JRG, Gruber B, Lafourcade B, Leitão PJ, Münkemüller T, McClean C, Osborne PE, Reineking B, Schröder B, Skidmore AK, Zurell D, Lautenbach S. 2013. Collinearity: A review of methods to deal with it and a simulation study evaluating their performance. Ecography 36: 27-46.

Dray S, Pélissier R, Couteron P, Fortin MJ, Legendre P, Peres-Neto PR, Bellier E, Bivand R, Blanchet FG, De Cáceres M, Dufour AB, Heegaard E, Jombart T, Munoz F, Oksanen J, Thioulouse J, Wagner HH. 2012. Community ecology in the age of multivariate multiscale spatial analysis. Ecological Monographs 82: 257-275.

Dufrêne M, Legendre P. 1997. Species assemblages and indicator species the need for a flexible asymmetrical approach. Ecological Monographs 67: 345-366.

Dunstan PK, Foster SD, Darnell R. 2011. Model based grouping of species across environmental gradients. Ecological Modelling 222: 955-963.

Ellison AM, Gotelli NJ, Inouye BD, Strong DR. 2014. P values, hypothesis testing, and model selection: it's déjà vu all over again. Ecology 95: 609-610.

Emerson JW, Green WA, Schloerke B, Crowley J, Cook D, Hofmann H, Wickham H. 2013. The Generalized Pairs Plot. Journal of Computational and Graphical Statistics 22: 79-91.

Ethnobiology Working Group. 2003: Intellectual imperatives in Ethnobiology. St. Louis, Missouri Botanical Garden Press.

Faith DP. 1992. Conservation evaluation and phylogenetic diversity. Biological Conservation 61: 1-10.

Farr TG, Rosen PA, Caro E, Crippen R, Duren R, Hensley S, Kobrick M, Paller M, Rodriguez E, Roth L, Seal D, Shaffer S, Shimada J, Umland J, Werner M, Oskin M, Burbank D, Alsdorf D. 2007. The shuttle radar topography mission. Reviews of Geophysics 45: RG2004.

Fick SE, Hijmans RJ. 2017. WorldClim 2: new 1-km spatial resolution climate surfaces for global land areas. International Journal of Climatology 37: 4302-4315.

Fletcher R, Fortin MJ. 2018. Spatial ecology and conservation modeling: applications with r. Cham, Springer International Publishing.

Ford ED. 2000. Scientific method for ecological research. Cambridge, UK, Cambridge University Press.

Fortin MJ, Dale MRT. 2005. Spatial analysis: A guide for ecologists. Cambridge University Press.

Fox GA, Negrete-Yankelevich S, Sosa VJ. (eds.) 2015. Ecological statistics: Contemporary theory and application. Oxford, Oxford University Press.

Franco-Belussi L, De Oliveira C, Sköld HN. 2018. Regulation of eye and jaw colouration in three-spined stickleback Gasterosteus aculeatus. Journal of Fish Biology 92: 1788-1804.

Franklin J, Miller JA. 2009. Mapping species distributions: Spatial inference and prediction. Cambridge; New York, Cambridge University Press.

Frenette-Dussault C, Shipley B, Léger JF, Meziane D, Hingrat Y. 2012. Functional structure of an arid steppe plant community reveals similarities with Grime's C-S-R theory. Journal of Vegetation Science 23: 208-222.

Garamszegi LZ (ed.). 2014. Modern phylogenetic comparative methods and their application in evolutionary biology. Berlin Heidelberg.

Garnier E, Navas ML, Grigulis K. 2015. Plant functional diversity: organism traits, community structure, and ecosystem properties. 1st edn. Oxford, United Kingdom, Oxford University Press.

Gillespie C, Lovelace R. 2017. Efficient R programming: A practical guide to smarter programming. O'Really.

Goldenberg SU, Nagelkerken I, Ferreira CM, Ullah H, Connell SD. 2017. Boosted food web productivity through ocean acidification collapses under warming. Global Change Biology 23: 4177-4184.

Gonçalves PHS, Albuquerque UP, Medeiros PM. 2016. The most commonly available woody plant species are the most useful for human populations: a meta-analysis. Ecological Applications 26: 2238-2253.

Gonçalves-Souza T, Garey MV, da Silva FR, Albuquerque UP, Provete DB. 2019. Multidimensional analyses for testing ecological, ethnobiological, and conservation hypotheses. In: Albuquerque UP, de Lucena RFP, da Cunha LVF, Alves RRN (eds.). Methods and techniques in ethnobiology and ethnoecology. Humana Press, New York, NY. p. 87-110.

Gonçalves-Souza T, Provete DB, Garey MV, da Silva FR, Albuquerque UP. 2019. Going back to basics: How to master the art of making scientifically sound questions. In: Albuquerque UP, de Lucena RFP, da Cunha LVF, Alves RRN (eds.). Methods and techniques in ethnobiology and ethnoecology. Humana Press, New York, NY. p. 71-86.

Goodness J, Andersson E, Anderson PML, Elmqvist T. 2016. Exploring the links between functional traits and cultural ecosystem services to enhance urban ecosystem management. Ecological Indicators 70: 597-605.

Gotelli NJ. 2000. Null model analysis of species co-occurrence patterns. Ecology 81: 2606-2621.

Gotelli NJ, Chao A. 2013. Measuring and estimating species richness, species diversity, and biotic similarity from sampling data. In: Levin SA (ed.). Encyclopedia of Biodiversity. 2nd. edn. Waltham, MA: Academic Press. p. 195-211.

Gotelli NJ, Colwell RK. 2001. Quantifying biodiversity: procedures and pitfalls in the measurement and comparison of species richness. Ecology Letters 4: 379-391.

Gotelli NJ, Ellison AM. 2012. A primer of ecological statistics. 2nd edn. Sunderland, Massachusetts, Sinauer Associates, Inc., Publishers.

Gotelli NJ, Graves GR. 1996. Null models in ecology. Smithsonian Institution Press.

Götzenberger L, Bello F, Bråthen KA, Davison J, Dubuis A, Guisan A, Lepš J, Lindborg R, Moora M, Pärtel M, Pellissier L, Pottier J, Vittoz P, Zobel K, Zobel M. 2012. Ecological assembly rules in plant communities-approaches, patterns and prospects. Biological Reviews 87: 111-127.

Gower JC. 1971. Statistical methods of comparing different multivariate analyses of the same data. In: Hodson FR, Kendall DG, Tautu P (eds.). Mathematics in the archaeological and historical science. Edinburgh, Edinburgh University Press. p. 138-149.

Grolemund G. 2014. Hands-On programming with R: write your own functions and simulations. O'Really.

Guisan A, Thuiller W, Zimmermann NE. 2017. Habitat Suitability and Distribution Models: With Applications in R. Cambridge, Cambridge University Press.

Hadi AS, Ling RF. 1998. Some Cautionary Notes on the Use of Principal Components Regression. The American Statistician 52: 15-19.

Halsey LG. 2019. The reign of the p -value is over: what alternative analyses could we employ to fill the power vacuum? Biology Letters 15: 20190174.

Harrison XA, Donaldson L, Correa-Cano ME, Evans J, Fisher DN, Goodwin CED, Robinson BS, Hodgson DJ, Inger R. 2018. A brief introduction to mixed effects modelling and multi-model inference in ecology. PeerJ 6: e4794.

Healy K 2018. Data visualization: A practical introduction. Princeton, Princeton University Press.

Helmus MR, Bland TJ, Williams CK, Ives AR. 2007. Phylogenetic measures of biodiversity. The American Naturalist 169: E68-E83.

Hill MO. 1973. Diversity and evenness: A unifying notation and its consequences. Ecology 54: 427-432.

Holmes S, Huber W. 2019. Modern statistics for modern biology. Cambridge, United Kingdom, Cambridge University Press.

Horgan GW, Elston DA, Franklin MF, Glasbey CA, Hunter EA, Talbot M, Kempton RA, McNicol JW, Wright F. 1999. Teaching statistics to biological research scientists. Journal of the Royal Statistical Society D 48: 393-400.

Hortal J, Borges PAV, Gaspar C. 2006. Evaluating the performance of species richness estimators: sensitivity to sample grain size. Journal of Animal Ecology 75: 274-287.

Hsieh TC, Ma KH, Chao A. 2016. iNEXT: An R package for rarefaction and extrapolation of species diversity (Hill numbers). Methods in Ecology and Evolution 7: 1451-1456.

Hurlbert SH. 1971. The nonconcept of species diversity: A critique and alternative parameters. Ecology 52: 577-586.

Irizarry RA. 2017. Data analysis for the life sciences with R. Boca Raton, CRC Press, Taylor & Francis Group.

Irizarry RA. 2019. Introduction to Data Science: Data Analysis and Prediction Algorithms with R. 1st. edn. Chapman, Hall/CRC.

Ismay C, Kim AYS. 2020. Statistical inference via data science: A ModernDive into R and the Tidyverse. Boca Raton, CRC Press / Taylor & Francis Group.

Ives AR. 2015. For testing the significance of regression coefficients, go ahead and log-transform count data. Methods in Ecology and Evolution 6: 828-835.

Jackson DA. 1995. PROTEST: A PROcrustean randomization TEST of community environment concordance. Ecoscience 2: 297-303.

James FC, McCulloch CE. 1990. Multivariate Analysis in Ecology and Systematics: Panacea or Pandora's Box? Annual Review of Ecology and Systematics 21: 129-166.

James G, Witten D, Hastie T, Tibshirani R. 2013. An introduction to statistical learning. Springer New York.

Jetz W, Thomas GH, Joy JB, Hartmann K, Mooers AO. 2012. The global diversity of birds in space and time. Nature 491: 444-448.

Jost L. 2006. Entropy and diversity. Oikos 113: 363-375.

Jost L. 2007. Partitioning diversity into independent alpha and beta components. Ecology 88: 2427-2439.

Kabacoff R. 2011. R in action: Data analysis and graphics with R. Shelter Island, NY, Manning.

Lai J, Lortie CJ, Muenchen RA, Yang J, Ma K. 2019. Evaluating the popularity of R in ecology. Ecosphere 10: e02567.

Laliberté E, Legendre P. 2010. A distance-based framework for measuring functional diversity from multiple traits. Ecology 91: 299-305.

Lapaine M, Usery EL. 2017. Choosing a map projection. Springer.

Legendre P. 1993. Spatial Autocorrelation: Trouble or New Paradigm? Ecology 74: 1659-1673.

Legendre P. 2000. Comparison of permutation methods for the partial correlation and partial mantel tests. Journal of Statistical Computation and Simulation 67: 37-73.

Legendre P, Anderson MJ. 1999. Distance-based redundancy analysis: Testing multispecies responses in multifactorial ecological experiments. Ecological monographs 69: 1-24.

Legendre P, Borcard D. 2018. Box-Cox-chord transformations for community composition data prior to beta diversity analysis. Ecography 41: 1820-1824.

Legendre P, De Cáceres M. 2013. Beta diversity as the variance of community data: dissimilarity coefficients and partitioning. Ecology Letters 16: 951-963.

Legendre P, Fortin MJ. 2010. Comparison of the mantel test and alternative approaches for detecting complex multivariate relationships in the spatial analysis of genetic data. Molecular Ecology Resources 10: 831-844.

Legendre P, Gallagher ED. 2001. Ecologically meaningful transformations for ordination of species data. Oecologia 129: 271-280.

Legendre P, Legendre L. 2012. Numerical Ecology. 3rd. edn. Elsevier.

Legendre P, Fortin MJ, Borcard, D. 2015. Should the mantel test be used in spatial analysis? Methods in Ecology and Evolution 6: 1239-1247.

Leprieur F, Albouy C, De Bortoli J, Cowman PF, Bellwood DR, Mouillot D. 2012. Quantifying phylogenetic beta

diversity: Distinguishing between 'true' turnover of lineages and phylogenetic diversity gradients. PLoS ONE 7: e42760.

Lepš J, de Bello F, Šmilauer P, Doležal J. 2011. Community trait response to environment: Disentangling species turnover vs intraspecific trait variability effects. Ecography 34: 856-863.

Lessard JP, Belmaker J, Myers JA, Chase JM, Rahbek C. 2012. Inferring local ecological processes amid species pool influences. Trends in Ecology & Evolution 27: 600-607.

Lima VF, Brito SV, Araujo Filho JA, Teles DA, Ribeiro SC, Teixeira AAM, Pereira AMA, Almeida WO. 2018. Raillietiella mottae (Pentastomida: Raillietiellidae) parasitizing four species of Gekkota lizards (Gekkonidae and Phyllodactylidae) in the Brazilian Caatinga. Helminthologia 55: 140-145.

Lindenmayer DB, Sato C. 2018. Hidden collapse is driven by fire and logging in a socioecological forest ecosystem. Proceedings of the National Academy of Sciences 115: 5181-5186.

Lisboa FJG, Peres-Neto PR, Chaer GM, Jesus EC, Mitchell RJ, Chapman SJ, Berbara RLL. 2014. Much beyond mantel: Bringing procrustes association metric to the plant and soil ecologist's toolbox. PLoS One 9: e101238.

Littell RC, Milliken GA, Stroup WW, Wolfinger RD, Schabenberger O. 2006. SAS for mixed models. 2nd. edn. SAS Institute.

Long JD, Teetor P. 2019. R cookbook: Proven recipes for data analysis, statistics, and graphics. 2nd. edn. Beijing Boston Farnham Sebastopol Tokyo, O'Reilly.

Lovelace R, Nowosad J, Münchow J. 2019. Geocomputation with R. Boca Raton, CRC Press, Taylor; Francis Group.

Lozupone C, Knight R. 2005. UniFrac: a new phylogenetic method for comparing microbial communities. Applied and Environmental Microbiology 71: 8228-8235.

MacDougall AS, Gilbert B, Levine JM. 2009. Plant invasions and the niche. Journal of Ecology 97: 609-615.

Magnusson WE, Mourão G. 2004. Estatística sem Matemática. A Ligação Entre as Questões e a Análise. Editora Planta.

Magurran AE, McGill BJ. 2011. Biological diversity: Frontiers in measurement and assessment. Oxford, Oxford University Press.

Mammola S, Carmona CP, Guillerme T, Cardoso P. 2021: Concepts and applications in functional diversity. Functional Ecology 35: 1869-1885.

Manly BFJ. 1991. Randomization and Monte Carlo methods in biology. Chapman & Hall.

Martínez-Abraín A. 2008. Statistical significance and biological relevance: A call for a more cautious interpretation of results in ecology. Acta Oecologica 34: 9-11.

Mas JF, Horta MB, de Vasconcelos RN, Cambui ECB. 2019. Análise espacial com R. UEFS Editora.

Mason NWH, Mouillot D. 2013. Functional diversity measures. In: Levin SA (ed.). Encyclopedia of Biodiversity. 2nd. edn. Waltham, MA: Academic Press. p. 597-608.

Matejka J, Fitzmaurice G. 2017. Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing. Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems: 1290–1294.

Matloff NS. 2011. The art of R programming: Tour of statistical software design. San Francisco, No Starch Press.

McGill BJ, Enquist BJ, Weiher E, Westoby M. 2006. Rebuilding community ecology from functional traits. Trends in Ecology & Evolution 21: 178-185.

McGill BJ, Etienne RS, Gray JS, Alonso D, Anderson MJ, Benecha HK, Dornelas M, Enquist BJ, Green JL, He F, Hurlbert AH, Magurran AE, Marquet PA, Maurer BA, Ostling A, Soykan CU, Ugland KI, White EP. 2007. Species abundance distributions: moving beyond single prediction theories to integration within an ecological framework. Ecology Letters 10: 995-1015.

McIntosh A, Pontius J. 2017. Science and the global environment: case studies for integrating science and the global environment. Elsevier.

Melo AS. 2008. O que ganhamos 'confundindo' riqueza de espécies e equabilidade em um índice de diversidade? Biota Neotropica 8: 21-27.

Metz AM. 2008. Teaching statistics in biology: Using inquiry-based learning to strengthen understanding of statistical analysis in biology laboratory courses. CBE—Life Sciences Education 7: 317-326.

Moraga P. 2019. Geospatial Health Data: Modeling and Visualization with R-INLA and Shiny. 1st. edn. Chapman; Hall/CRC.

Muff S, Nilsem EB, O'Hara RB, Nater CR. 2021. Rewriting results sections in the language of evidence. Trends in Ecology & Evolution 37: 203-210.

Murtaugh PA. 2014. In defense of P values. Ecology 95: 611-617.

Muylaert RL, Vancine MH, Bernardo R, Oshima JEF, Sobral-Souza T, Tonetti VR, Niebuhr BB, Ribeiro MC. 2018. Uma nota sobre os limites territoriais da mata atlântica. Oecologia Australis 22: 302-311.

Nakagawa S, Johnson PCD, Schielzeth H. 2017. The coefficient of determination R2 and intra-class correlation coefficient from generalized linear mixed-effects models revisited and expanded. Journal of The Royal Society Interface 14: 20170213.

Newbold T, Hudson LN, Hill SLL, Contu S, Lysenko I, Senior RA, Börger L, Bennett DJ, Choimes A, Collen B, Day J, De Palma A, Díaz S, Echeverria-Londoño S, Edgar MJ, Feldman A, Garon M, Harrison MLK, Alhusseini T, Ingram DJ, Itescu Y, Kattge J, Kemp V, Kirkpatrick L, Kleyer M, Correia DLP, Martin CD, Meiri S, Novosolov M, Pan Y, Phillips HRP, Purves DW, Robinson A, Simpson J, Tuck SL, Weiher E, White HJ, Ewers RM, Mace GM, Scharlemann JPW, Purvis A. 2015. Global effects of land use on local terrestrial biodiversity. Nature 520: 45-50.

Neyman J, Pearson ES. 1933. IX. On the problem of the most efficient tests of statistical hypotheses. Philosophical Transactions of the Royal Society of London A 231: 289-337.

Niggli U, Fliessbach A, Hepperly P, Scialabba N. 2009. Low greenhouse gas agriculture: Mitigation and adaptation potential of sustainable farming systems. Food; Agriculture Organization of the United Nations (FAO).

O'Hara, RB, Kotze DJ. 2010. Do not log-transform count data. Methods in Ecology and Evolution 1: 118-122.

Okabe A. 2000. Spatial tessellations: concepts and applications of Voronoi diagrams 2nd. edn. Chichester; New York, Wiley.

Olalla-Tárraga MA, Rodríguez MA. 2007. Energy and interspecific body size patterns of amphibian faunas in Europe and North America: anurans follow Bergmann's rule, urodeles its converse. Global Ecology and Biogeography 16: 606-617.

Olaya Ferrero V. 2020. Sistemas de información geográfica. España: Bubok Publishing. E-book.

Oliveira CN, Campos IHMP, Provete DB, Guarnieri MC, Ribeiro SC. 2020. Defensive behaviour and tail autotomy in Coleodactylus meridionalis (Squamata: Sphaerodactylidae). Journal of Natural History 54: 2209-2218.

Oliveira PF, Guerra S, McDonnell R. 2018. Ciência de Dados com R - Introdução. https://cdr.ibpad.com.br/cdr-intro.pdf. 17 Feb. 2022.

Ovaskainen O, Abrego N. 2020. Joint Species Distribution Modelling. Cambridge University Press.

Palmer MW. 1990. The estimation of species richness by extrapolation. Ecology 71: 1195-1198.

Palmer MW. 1991. Estimating species richness: The second-order Jackknife reconsidered. Ecology 72: 1512-1513.

Paradis E. 2012. Analysis of phylogenetics and evolution with R. New York, NY, Springer New York.

Pavoine S, Bonsall MB. 2011. Measuring biodiversity to explain community assembly: A unified approach. Biological Reviews 86: 792-812.

Pavoine S, Vallet J, Dufour AB, Gachet S, Daniel H. 2009. On the challenge of treating various types of variables: application for improving the measurement of functional diversity. Oikos 118: 391-402.

Pebesma EJ. 2018. Simple features for R: Standardized support for spatial vector data.: The R Journal 10: 439-446.

Pebesma EJ, Bivand RS. 2005. Classes and methods for spatial data in R. R News 5: 9-13.

Pereira RHM, Goncalves CN. 2021. geobr: Download official spatial data sets of Brazil. GitHub repository - https://github.com/ipeaGIT/geobr.

Peres-Neto PR, Dray S, Braak CJF ter. 2017. Linking trait variation to the environment: critical issues with community-weighted mean correlation resolved by the fourth-corner approach. Ecography 40: 806-816.

Peres-Neto PR, Jackson DA. 2001. How well do multivariate data sets match? The advantages of a procrustean superimposition approach over the mantel test. Oecologia 129: 169-178.

Petchey OL, Gaston KJ. 2002. Functional diversity (FD), species richness and community composition. Ecology Letters 5: 402-411.

Peterson AT, Soberón J, Pearson RG, Anderson RP, Martínez-Meyer E, Nakamura M, Araújo MB. 2011. Ecological niches and geographic distributions. Princeton, Princeton University Press.

Phillips O, Gentry AH. 1993. The useful plants of Tambopata, Peru: II. Additional hypothesis testing in quantitative ethnobotany. Economic Botany 47: 33-43.

Pickett ST, Kolasa J, Jones CG. 2007. Ecological understanding: the nature of theory and the theory of nature. 2nd. edn. Boston, Elsevier/Academic Press.

Pielou EC. 1966. The measurement of diversity in different types of biological collections. Journal of Theoretical Biology 13: 131-144.

Pinheiro JC, Bates DM. 2000. Mixed-Effects Models in S and S-PLUS. Springer-Verlag.

Platt JR. 1964. Strong Inference: Certain systematic methods of scientific thinking may produce much more rapid progress than others. Science 146: 347-353.

Popper KR. 1959. The logic of scientific discovery. London; New York, Routledge.

Prado VHM, Rossa-Feres DC. 2014. Multiple determinants of anuran richness and occurrence in an agricultural region in south-eastern Brazil. Environmental Management 53: 823-837.

Provete DB, Gonçalves-Souza T, Garey MV, Martins IA, Rossa-Feres DC. 2014. Broad-scale spatial patterns of canopy cover and pond morphology affect the structure of a Neotropical amphibian metacommunity. Hydrobiologia 734: 69-79.

Quinn GP, Keough MJ. 2002. Experimental design and data analysis for biologists. Cambridge University Press.

Rahlf T. 2019. Data visualisation with R: 111 examples. 2nd. edn. Cham, Springer.

Redding DW, Mooers AØ. 2006. Incorporating evolutionary measures into conservation prioritization. Conservation Biology 20: 1670-1678.

Reed J, Vianen JV, Barlow J, Sunderland T. 2017. Have integrated landscape approaches reconciled societal and environmental issues in the tropics? Land Use Policy 63: 481-492.

Ripple WJ, Estes JA, Beschta RL, Wilmers CC, Ritchie EG, Hebblewhite M, Berger J, Elmhagen B, Letnic M, Nelson MP, Schmitz OJ, Smith DW, Wallach AD, Wirsing AJ. 2014. Status and ecological effects of the World's largest carnivores. Science 343, 6167.

Rosauer D, Laffan SW, Crisp MD, Donnellan SC, Cook LG. 2009. Phylogenetic endemism: a new approach for identifying geographical concentrations of evolutionary history. Molecular Ecology 18: 4061-4072.

Roswell M, Dushoff J, Winfree R. 2021. A conceptual guide to measuring species diversity. Oikos 130: 321-338.

Ruxton GD, Colegrave N. 2016. Experimental design for the life sciences. 3rd. edn. Oxford, New York, Oxford University Press.

Saslis-Lagoudakis CH, Clarke AC. 2013. Ethnobiology: The missing link in ecology and evolution. Trends in Ecology & Evolution 28: 67-68.

Saslis-Lagoudakis CH, Hawkins JA, Greenhill SJ, Pendry CA, Watson MF, Tuladhar-Douglas W, Baral SR, Savolainen V. 2014. The evolution of traditional knowledge: environment shapes medicinal plant use in Nepal. Proceedings of the Royal Society B 281: 20132768.

Saslis-Lagoudakis CH, Savolainen V, Williamson EM, Forest F, Wagstaff SJ, Baral SR, Watson MF, Pendry CA, Hawkins JA. 2012. Phylogenies reveal predictive power of traditional medicine in bioprospecting. Proceedings of the National Academy of Sciences 109: 15835-15840.

Saul WC, Jeschke JM. 2015. Eco-evolutionary experience in novel species interactions. Ecology Letters 18: 236-245.

Scheiner SM, Gurevitch J. 2001. Design and analysis of ecological experiments. Oxford University Press.

Schleuter D, Daufresne M, Massol F, Argillier C. 2010. A user's guide to functional diversity indices. Ecological Monographs 80: 469-484.

Siefert A, Violle C, Chalmandrier L, Albert CH, Taudiere A, Fajardo A, Aarssen LW, Baraloto C, Carlucci MB, Cianciaruso MV, Dantas LV, Bello F, Duarte LDS, Fonseca CR, Freschet GT, Gaucherand S, Gross N, Hikosaka K, Jackson B, Jung V, Kamiyama C, Katabuchi M, Kembel SW, Kichenin E, Kraft NJB, Lagerström A, Bagousse-Pinguet YL, Li Y, Mason N, Messier J, Nakashizuka T, Overton JM, Peltzer DA, Pérez-Ramos IM, Pillar VD, Prentice HC, Richardson S, Sasaki T, Schamp BS, Schöb C, Shipley B, Sundqvist M, Sykes MT, Vandewalle M, Wardle DA. 2015. A global meta-analysis of the relative extent of intraspecific trait variation in plant communities. Ecology Letters 18: 1406-1419.

Sievert C. 2020. Interactive web-based data visualization with R, plotly, and shiny. Boca Raton, FL, CRC Press, Taylor; Francis Group.

Smith EP, van Belle G. 1984. Nonparametric estimation of species richness. Biometrics 40: 119-129.

Sneath PHA, Sokal RR. 1973. Numerical taxonomy: the principles and practice of numerical classification. San Francisco, W. H. Freeman.

Solar RRC, Barlow J, Ferreira J, Berenguer E, Lees AC, Thomson JR, Louzada J, Maués M, Moura NG, Oliveira VHF, Chaul JCM, Schoereder JH, Vieira ICG, Mac Nally R, Gardner TA. 2015. How pervasive is biotic homogenization in human-modified tropical forest landscapes? Ecology Letters 18: 1108-1118.

Stepp JR. 2005. Advances in ethnobiological field methods. Field Methods 17: 211-218.

Strong DR, Simberloff D, Abele LG, Thistle AB (eds.). 1984. Ecological communities. Princeton University Press.

Sutherland WJ, Freckleton RP, Godfray HCJ, Beissinger SR, Benton T, Cameron DD, Carmel Y, Coomes DA, Coulson T, Emmerson MC, Hails RS, Hays GC, Hodgson DJ, Hutchings MJ, Johnson D, Jones JPG, Keeling MJ, Kokko H, Kunin WE, Lambin X, Lewis OT, Malhi Y, Mieszkowska N, Milner-Gulland EJ, Norris K, Phillimore AB, Purves DW, Reid JM, Reuman DC, Thompson K, Travis JMJ, Turnbull LA, Wardle DA, Wiegand T. 2013. Identification of 100 fundamental ecological questions. Journal of Ecology 101: 58-67.

Swenson NG. 2011. Phylogenetic beta diversity metrics, trait evolution and inferring the functional beta diversity of communities. PLoS ONE 6: e21264.

Swenson NG. 2014. Functional and Phylogenetic Ecology in R. New York, NY, Springer New York.

Tennekes M. 2018. tmap: Thematic Maps in R. Journal of Statistical Software 84: 1-39.

Thioulouse J, Dray S, Dufour AB, Siberchicot A, Jombart T, Pavoine S. 2018. Multivariate Analysis of Ecological Data with ade4. New York, NY, Springer New York.

Touchon JC. 2021. Applied statistics with R: A practical guide for the life sciences. New York, AC Science: ACSCI.

Tucker CM, Cadotte MW, Carvalho SB, Davies TJ, Ferrier S, Fritz SA, Grenyer R, Helmus MR, Jin LS, Mooers AO, Pavoine S, Purschke O, Redding DW, Rosauer DF, Winter M, Mazel F. 2017. A guide to phylogenetic metrics for conservation, community ecology and macroecology. Biological Reviews 92: 698-715.

Ulrich W, Gotelli NJ. 2010. Null model analysis of species associations using abundance data. Ecology 91: 3384-3397.

Underwood AJ. 1997. Experiments in ecology: their logical design and interpretation using analysis of variance. Cambridge, Cambridge University Press.

Vancine MH, Duarte KS, de Souza YS, Giovanelli JGR, Martins-Sobrinho PM, López A, Bovo RP, Maffei F, Lion MBI, Ribeiro Júnior JW, Brassaloti R, da Costa COR, Sawakuchi HO, Forti LR, Cacciali P, Bertoluci J, Haddad CFB, Ribeiro MC. 2018. ATLANTIC AMPHIBIANS: a data set of amphibian communities from the Atlantic Forests of South America. Ecology 99: 1692-1692.

Vane-Wright RI, Humphries CJ, Williams PH. 1991. What to protect? Systematics and the agony of choice. Biological Conservation 55: 235-254.

Vellend M. 2016. The theory of ecological communities. Princeton, Princeton University Press.

Vellend M, Cornwell WK, Magnuson-Ford K, Mooers A. 2011. Measuring phylogenetic biodiversity. In: Magurran AE, McGill BJ (eds.). Biological diversity: Frontiers in measurement and assessment. Oxford University Press. p. 194–207.

Venables WN, Ripley BD. 2002. Modern applied statistics with S. Springer-Verlag.

Véron S, Saito V, Padilla-García N, Forest F, Bertheau Y. 2019. The Use of phylogenetic diversity in conservation biology and community ecology: A common base but different approaches. The Quarterly Review of Biology 94: 123-148.

Villéger S, Mason NWH, Mouillot D. 2008. New multidimensional functional diversity indices for a multifaceted framework in functional ecology. Ecology 89: 2290-2301.

Violle C, Enquist BJ, McGill BJ, Jiang L, Albert CH, Hulshof C, Jung V, Messier J. 2012. The return of the variance: Intraspecific variability in community ecology. Trends in Ecology and Evolution 27: 244-252.

Violle C, Navas ML, Vile D, Kazakou E, Fortunel C, Hummel I, Garnier E. 2007. Let the concept of trait be functional! Oikos 116: 882-892.

Walther BA, Moore JL. 2005. The concepts of bias, precision and accuracy, and their use in testing the performance of species richness estimators, with a literature review of estimator performance. Ecography 28: 815-829.

Wang Y, Naumann U, Wright ST, Warton DI. 2012. mvabund: an r package for model-based analysis of multivariate abundance data. Methods in Ecology and Evolution 3: 471-474.

Warton DI. 2018. Why you cannot transform your way out of trouble for small counts. Biometrics 74: 362-368.

Warton DI, Wright ST, Wang Y. 2012. Distance-based multivariate analyses confound location and dispersion effects. Methods in Ecology and Evolution 3: 89-101.

Webb CO, Ackerly DD, Kembel SW. 2008. Phylocom: software for the analysis of phylogenetic community structure and trait evolution. Bioinformatics 24: 2098-2100.

Webb CO, Ackerly DD, McPeek MA, Donoghue MJ. 2002. Phylogenies and community ecology. Annual Review of Ecology and Systematics 33: 475-505.

Wegmann M, Leutner B, Dech S. (eds.). 2016. Remote sensing and GIS for ecologists: using open source software. Exeter, Pelagic Publishing.

Wegmann M, Schwalb-Willmann J, Dech S. 2020. An introduction to spatial data analysis: Remote sensing and GIS with open source software. Pelagic Publishing Ltd.

White JW, Rassweiler A, Samhouri JF, Stier AC, White C. 2014. Ecologists should not use statistical significance tests to interpret simulation model results. Oikos 123: 385-388.

Whitlock M, Schluter D. 2015. The analysis of biological data. 2nd. edn. Roberts; Company Publishers.

Whittaker RH. 1972. Evolution and measurement of species diversity. TAXON 21: 213-251.

Whittaker RH. 1960. Vegetation of the Siskiyou Mountains, Oregon and California.: Ecological Monographs 30: 279-338.

Wickham H. 2013. R and S.: Encyclopedia of Environmetrics.

Wickham H. 2014. Tidy Data. Journal of Statistical Software 59: 1-23.

Wickham H. 2016. ggplot2: Elegant graphics for data analysis. Springer-Verlag New York.

Wickham H. 2019. Advanced R. 2nd. edn. Boca Raton, CRC Press/Taylor; Francis Group.

Wickham H, Averick M, Bryan J, Chang W, McGowan L, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen T, Miller E, Bache S, Müller K, Ooms J, Robinson D, Seidel D, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H. 2019. Welcome to the Tidyverse. Journal of Open Source Software 4: 1686.

Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke C, Woo K, Yutani H, Dunnington D. 2020. ggplot2: Create elegant data visualisations using the grammar of graphics,.

Wickham H, Grolemund G. 2017. R for data science: import, tidy, transform, visualize, and model data. 1st. edn. Sebastopol, CA, O'Reilly.

Wilke C. 2019. Fundamentals of data visualization: A primer on making informative and compelling figures. 1st. edn. Sebastopol, CA, O'Reilly Media.

Wilkinson L, Wills G. 2005. The grammar of graphics. 2nd edn. New York, Springer.

Williams GJ. 2018. The essentials of data science: Knowledge discovery using R. Boca Raton, Taylor & Francis, CRC Press.

Wilson JB. 1991. Methods for fitting dominance/diversity curves. Journal of Vegetation Science 2: 35-46.

Xu W, Xiao Y, Zhang J, Yang W, Zhang L, Hull V, Wang Z, Zheng H, Liu J, Polasky S, Jiang L, Xiao Y, Shi X, Rao E, Lu F, Wang X, Daily GC, Ouyang Z. 2017. Strengthening protected areas for biodiversity and ecosystem services in China. Proceedings of the National Academy of Sciences 114: 1601-1606.

Zar JH. 2010. Biostatistical analysis. Pearson.

Zumel N, Mount J. 2014. Practical data science with R. Shelter Island, NY, Manning Publications Co.

Zuur AF, Ieno EN, Elphick CS. 2010. A protocol for data exploration to avoid common statistical problems. Methods in Ecology and Evolution 1: 3-14.

Zuur AF, Ieno EN, Smith GM. 2007. Analysing Ecological Data. Gail M, Krickeberg K, Sarnet J, Tsiatis A, Wong W (eds.). New York, NY, Springer New York.

Zuur AF, Ieno EN, Walker N, Saveliev AA, Smith GM. 2009. Mixed effects models and extensions in ecology with R. New York, NY, Springer New York.

O livro **Análises Ecológicas no R** é uma contribuição para o contínuo avanço do ensino de métodos computacionais, com um foco específico em análise de dados ecológicos através da linguagem R. O livro descreve como os códigos devem ser consequências das perguntas que a pesquisa pretende responder. Essa visão tem como consequência um livro que do começo ao fim conecta teoria ecológica, métodos científicos, análises quantitativas e programação. Isso é feito de modo explícito através de exemplos claros e didáticos que apresentam contexto e dados reais, um ou mais exemplos de perguntas que poderiam ser feitas, predições relacionadas às perguntas e a teoria em questão, além das variáveis que poderiam ser utilizadas nas análises. O texto que descreve essas partes é intercalado com pedaços organizados e claros de código e gráficos, o que torna a leitura dos capítulos bastante fluida e dinâmica, principalmente para quem gosta de executar os códigos no seu computador conforme lê os capítulos. É como uma aula prática guiada.

Tadeu Siqueira - UNESP

Apoio

Instituto Nacional de Ciência e Tecnologia
ETNOBIOLOGIA, BIOPROSPECÇÃO E CONSERVAÇÃO DA NATUREZA

NUPEEA

canal6 editora