

Ecologia de Paisagens com R

Darren Norris: darren.norris@unifap.br

08 janeiro, 2024

Sumário

I Apresentação	5
Bem-vindos	5
Agradecimentos	5
Prefácio à primeira edição	6
Introdução	7
O que você vai aprender	7
Como este livro está organizado	7
O que você não vai aprender	7
Prerequisites	7
R	7
RStudio	8
O universo arrumado - tidyverse	8
Outros pacotes	8
Código no livro	8
Organização do código no livro	10
II Escala e padrões	11
1 Escala	12
1.1 Apresentação	12
1.2 Escala: breve definição	12
1.3 Pacotes e dados	14
1.3.1 Pacotes	14
1.3.2 Dados	15
1.4 Alterando a resolução	18
1.5 Escala espacial e desenho amostral	22
1.5.1 Obter e carregar dados (vectores)	22
1.5.2 Visualizar os arquivos (camadas vector)	23
1.5.3 Obter e carregar dados (raster)	24
1.5.4 Visualizar os arquivos (camadas raster e vector)	24
1.5.5 Reclassificação	26
1.6 Comparação multiescala	28
1.7 Próximos passos: repetindo para muitas amostras.	31
2 Métricas da paisagem	33
2.1 Apresentação	33

2.2	Métricas da paisagem e pacote “landscapemetrics”	34
2.2.1	Pacotes	34
2.3	Dados	36
2.3.1	Exibir dados raster e sobreposição com locais de amostragem	37
2.4	Calculo de métricas	40
2.4.1	Uso de pontos de amostragem na ecologia da paisagem	40
2.4.2	Ponto único, raio único, métrica única	41
2.4.3	Ponto único, distâncias variados, métrica única	43
2.4.4	Ponto único, distâncias variados, métricas variadas	53
2.5	Links úteis	57
3	Respostas multiescala de espécies	59
3.1	Apresentação	59
3.2	Escala de efeito - definição	60
3.2.1	Definições	60
3.2.2	Aplicação	60
3.2.3	Como identificar a escala de efeito?	61
3.3	Pacotes e dados	61
3.3.1	Pacotes	61
3.3.2	Dados necessarios	61
3.4	Buffers	65
3.4.1	Floresta	65
3.4.2	Floresta e a ocorrência de espécies	65
3.5	Kernels	78
3.6	Links úteis	78
III	Exemplos de caso	80
4	Garimpo do Lourenço	81
4.1	Apresentação	81
4.2	Pacotes necessarios:	82
4.3	Área de estudo	82
4.4	Dados	82
4.4.1	Ponto de referência (EPSG: 4326)	82
4.4.2	Ponto de referência (EPSG: 31976)	83
4.4.3	Verificar com mapa de base (OpenStreetMap)	83
4.4.4	Dados: MapBiomass cobertura da terra	84
4.5	Calculo de métricas	85
4.5.1	Métricas para a paisagem	86
4.5.2	Métricas para as classes	86
4.5.3	Métricas para as manchas	87
4.5.4	Quais métricas devo escolher?	88
4.5.5	Métricas por classe de mineração	88
4.5.6	Exportar as métricas	90
4.6	Preparando os resultados	90
4.7	Uma tabela versatil	93
4.8	Reorganização	93
4.9	Uma figura elegante	94
4.9.1	Gráfico de barra	94
4.9.2	Gráfico de boxplot	95
4.10	Comparação entre anos	96
4.11	Conclusões e próximos passos	100

IV R e RStudio: Código com certeza	101
5 Pré-requisitos	102
5.1 Introdução	102
5.2 Instalação do R	102
5.3 Instalação do RStudio	103
5.4 Versão do R	103
5.5 Pacotes	103
5.6 Dados	104
6 Introdução ao R	105
Pré-requisitos do capítulo	105
6.1 Contextualização	105
6.2 R e RStudio	105
6.3 Funcionamento da linguagem R	107
6.3.1 Console	108
6.3.2 Scripts	108
6.3.3 Operadores	109
6.3.4 Objetos	112
6.3.5 Funções	113
6.3.6 Pacotes	114
6.3.7 Ajuda (<i>Help</i>)	118
6.3.8 Ambiente (<i>Environment</i>)	119
6.3.9 Citações	120
6.3.10 Principais erros de iniciantes	121
6.4 Estrutura e manipulação de objetos	123
6.4.1 Atributo dos objetos	124
6.4.2 Manipulação de objetos unidimensionais	133
6.4.3 Manipulação de objetos multidimensionais	135
6.4.4 Valores faltantes e especiais	139
6.4.5 Diretório de trabalho	141
6.4.6 Importar dados	141
6.4.7 Conferência dos dados importados	142
6.4.8 Exportar dados	144
6.5 Para se aprofundar	144
6.5.1 Livros	144
6.5.2 Links	144
6.6 Exercícios	145
7 Tidyverse	146
Pré-requisitos do capítulo	146
7.1 Contextualização	146
7.2 <i>tidyverse</i>	147
7.3 <i>readr</i> , <i>readxl</i> e <i>writexl</i>	149
7.4 <i>tibble</i>	149
7.5 <i>pipe</i> : <i>base</i> (>) e <i>magrittr</i> (%>%)	150
7.6 <i>tidyR</i>	152
7.6.1 <i>palmerpenguins</i>	152
7.6.2 <i>glimpse()</i>	153
7.6.3 <i>unite()</i>	154
7.6.4 <i>separate()</i>	154
7.6.5 <i>drop_na()</i> e <i>replace_na()</i>	155
7.6.6 <i>pivot_longer()</i> e <i>pivot_wider()</i>	156
7.7 <i>dplyr</i>	157

7.7.1	Gramática	158
7.7.2	Sintaxe	158
7.7.3	palmerpenguins	158
7.7.4	relocate()	159
7.7.5	rename()	159
7.7.6	select()	160
7.7.7	pull()	161
7.7.8	mutate()	161
7.7.9	arrange()	162
7.7.10	filter()	163
7.7.11	slice()	165
7.7.12	distinct()	166
7.7.13	count()	167
7.7.14	group_by()	168
7.7.15	summarise()	168
7.7.16	bind_rows() e bind_cols()	169
7.7.17	*_join()	170
7.7.18	Operações de conjuntos e comparação de dados	171
7.8	stringr	172
7.9	forcats	174
7.10	lubridate	175
7.11	purrr	183
7.12	Para se aprofundar	186
7.12.1	Livros	186
7.12.2	Links	186
7.13	Exercícios	186
8	Primeiros passos com uma raster	188
8.0.1	Obter e carregar dados (raster)	188
8.0.2	Reclassificação	190
9	Primeiros passos com vector	192
9.1	Obter e carregar dados (vectores)	192
9.2	Visualizar os arquivos (camadas vector)	195
V	Encerramento	196
Bibliografia		197
Glossário		199
A	Annexo 1	207
A.1	Download mapbiomas cover	207
A.2	Crop mapbiomas cover	208
A.3	Plot with legend	210

Part I

Apresentação

Bem-vindos

Este é um trabalho em andamento do 1^a edição: “Ecologia de Paisagens com R”.

Versões:

- web: <https://darrenorris.github.io/epr/>
- pdf: <https://github.com/darrenorris/epr/blob/main/docs/epr.pdf>

Este material introdutório é direcionado principalmente a estudantes de graduação e pós-graduação em ecologia e áreas afins. Ele está sendo utilizado como interface para os materiais do curso de Ecologia da Paisagem, do Bacharelado em Ciências Ambientais da Universidade Federal do Amapá, Brasil.

O objetivo é de apresentar os capacidades e opções para desenvolver e integrar pesquisas na Ecologia da Paisagem no ambiente estatística de R.

Esperamos que ele seja utilizado tanto por quem quer se aprofundar em análises comumente utilizadas em Ecologia da Paisagem, mesmo por quem tem poucas habilidades quantitativas.

O conteúdo e organização dos capítulos estão separados em partes.

- O primeiro parte é “Apresentação”.
o objetivo dessa parte é incluir os aspectos mais gerais sobre os componentes e da estrutura do livro e seus objetivos.
- O segundo é “Escala e Padrões”.
Aqui, os capítulos incluem uma breve introdução usando R para explorar alguns componentes-chaves da Ecologia da Paisagem.
- A terceira é “Exemplos de caso”.
Aqui, os capítulos apresentam a aplicação de análises específicas e atualmente utilizadas em Ecologia da Paisagem.
- A quarta parte do livro é chamada de “R e RStudio”.
Esta parte trata o funcionamento da linguagem R. Aqui, nós aprendemos desde como instalar o R e o RStudio até a manipulação e visualização de dados geoespaciais no R.
- A parte final é “Encerramento”.
Aqui os capítulos apresentam a bibliografia e alguns apêndices (exemplos de código).

Agradecimentos

O material deste livro foi desenvolvido a partir de cursos de graduação ministrados pelos autores. Agradecemos a todos os alunos por fornecerem um suprimento ilimitado de perguntas que melhoraram o conteúdo. Este livro não é apenas o resultado dos autores. Mas é o resultado de muitas pessoas na comunidade R e Ecologia da Paisagem no Brasil. Agradecimentos especiais ao Dr. Alexandre Martensen, cujos exemplos inspiraram meu desenvolvimento de materiais de aprendizagem on-line e aos autores do livro [Análises Ecológicas no R](#) cuja dedicação me salvou anos de trabalho. Muito obrigado!

Prefácio à primeira edição

Há quem diga que a velocidade com que a tecnologia e a ciência avançam tende a tornar livros e manuais sobre métodos rapidamente obsoletos. A evolução dos computadores pessoais e a ampliação do acesso a estes e à Internet têm transformado o jeito como aprendemos e ensinamos. As pessoas estão cada vez mais familiarizados com a tecnologia e esperam que o ensino seja dinâmico e interativo.

Portanto, um livro aberto e disponível livremente, que as pessoas possam compartilhar e contribuir torna-se uma opção cada vez mais relevante. A intenção é para o livro seja utilizado como material de referência (fornecendo informações atualizadas); Como ferramenta de aprendizagem (apresentando diferentes ideias e abordagens) e como projeto de colaboração (permitindo que pessoas trabalhem juntas para criar um recurso educacional valioso).

Introdução

Ecologia da Paisagem é uma disciplina empolgante que permite transformar dados brutos em compreensão, insight e conhecimento.

O que você vai aprender

Ecologia da Paisagem é um campo vasto e não há como dominar tudo lendo um único livro. Este livro visa fornecer uma base sólida nas ferramentas mais importantes e conhecimento suficiente para encontrar os recursos para aprender mais quando necessário. Um modelo das etapas de um projeto típico de Ecologia da Paisagem se parece com.....

Six key steps for functional landscape analyses of habitat change <https://doi.org/10.1007/s10980-020-01048-y> - Acknowledge ecological theory and conceptual paradigms - Evaluate the fit of available data - Assess the three facets of the scale concept - Recognize different sampling designs - Use proper conceptual models - Measure meaningful raster characteristics

Como este livro está organizado

A descrição anterior das ferramentas da Ecologia da Paisagem é organizada aproximadamente de acordo com a ordem em que você as usa em uma análise (embora, é claro, você as itere várias vezes). Em nossa experiência, aprender os detalhes do linguagem R e programação primeiro não é o ideal, porque 80% do tempo é rotineiro e chato e, nos outros 20% do tempo, é estranho e frustrante. Esse é um péssimo lugar para começar a aprender um novo assunto! Em vez disso, os primeiros capítulos apresentam alguns dos conceitos centrais da Ecologia da Paisagem, apoiados em exemplos com R com dados que já foram importados e organizados. Dessa forma, quando você ingerir e organizar seus próprios dados, sua motivação permanecerá alta porque você sabe que a dor vale o esforço.

Dentro de cada capítulo, tentamos aderir a um padrão consistente: comece com alguns exemplos motivadores para que você possa ver o quadro geral e depois mergulhe nos detalhes. Cada seção do livro é combinada com exercícios para ajudá-lo a praticar o que aprendeu. Embora possa ser tentador pular os exercícios, não há melhor maneira de aprender do que praticar em problemas reais.

O que você não vai aprender

Base teórica pra trás os conceitos e cálculos. O foco é sobre a aplicação, e isso não preciso avanços teóricos. Incluímos referências para que é possível obter maior detalhes sobre os conceitos e teorias ecológicas e cálculos usados na Ecologia da Paisagem.

Prerequisites

Fizemos algumas suposições sobre o que você já sabe para aproveitar ao máximo este livro. Você deve ser geralmente alfabetizado numericamente, e com conhecimento previa de ecologia, geoprocessamento e uso de sistemas de informação geográfica.

Você precisa de quatro coisas para executar o código deste livro: R, RStudio, RTools e uma coleção de pacotes. Os pacotes são as unidades fundamentais do código R reproduzível. Eles incluem funções reutilizáveis, documentação que descreve como usá-los e dados de amostra.

R

Para fazer o download do R, acesse CRAN, a **comprehensive R archive network**, <https://cloud.r-project.org>. Uma nova versão principal do R é lançada uma vez por ano e há 2 a 3 versões secundárias a cada ano. É uma boa ideia atualizar regularmente. A atualização pode ser um pouco complicada, especialmente para as versões principais que exigem a reinstalação de todos os seus pacotes, mas adiar só piora as coisas. Para aqueles que são novos no uso do R, evite atualizações no meio do projeto ou quando há prazos se aproximando. Recomendamos R 4.2.0 ou posterior para este livro.

RStudio

RStudio é um ambiente de desenvolvimento integrado, ou IDE, para programação R, que você pode baixar em <https://posit.co/download/rstudio-desktop/>. O RStudio é atualizado algumas vezes por ano e avisa automaticamente quando uma nova versão é lançada, para que não haja necessidade de verificar novamente. É uma boa ideia atualizar regularmente para aproveitar os melhores e mais recentes recursos. Para este livro, certifique-se de ter pelo menos o RStudio 2022.02.0.

O universo arrumado - tidyverse

Você também precisará instalar alguns pacotes do R. Um **pacote** do R é uma coleção de funções, dados e documentação que estende os recursos do R base. O uso de pacotes é a chave para o uso bem-sucedido do R. A muitos dos pacotes que você aprenderá neste livro faz parte do chamado **tidyverse**: <https://www.tidyverse.org/>. Todos os pacotes no tidyverse compartilham uma filosofia comum de programação de dados e R e são projetados para trabalhar juntos. A análise de dados ecológicos tende a ser repetitiva e exige cuidado e atenção para garantir que os resultados sejam corretos, cientificamente válidos e possam ser verificados. os pacotes no tidyverse torna o trabalho com dados ecológicos mais rápido e fácil.

Você pode instalar o tidyverse completo com uma única linha de código:

```
install.packages("tidyverse")
```

No seu computador, digite essa linha de código no console e pressione enter para executá-lo. R irá baixar os pacotes do CRAN e instalá-los em seu computador.

Você não poderá usar as funções, objetos ou arquivos de ajuda em um pacote até carregá-lo com `library()`. Depois de instalar um pacote, você pode carregá-lo usando a função `library()`:

```
library(tidyverse)
```

Isso diz a você que o tidyverse carrega nove pacotes: dplyr,forcats, ggplot2, lubridate, purrr, readr, stringr, tibble, alignr. Eles são considerados o **núcleo** do tidyverse porque você os usará em quase todas as análises.

Os pacotes no tidyverse mudam com bastante frequência. Você pode ver se há atualizações disponíveis executando `tidyverse_update()`.

Outros pacotes

Existem muitos outros pacotes excelentes que não fazem parte do tidyverse porque resolvem problemas em um domínio diferente ou são projetados com um conjunto diferente de princípios subjacentes. Isso não os torna melhores ou piores, apenas diferentes. Em outras palavras, o complemento do tidyverse não é o universo bagunçado, mas muitos outros universos de pacotes inter-relacionados. Ao lidar com mais projetos de Ecologia da Paisagem com R, você aprenderá novos pacotes e novas formas de pensar sobre os dados.

Usaremos outras pacotes de fora do tidyverse neste livro. Por exemplo, usaremos os seguintes pacotes porque eles fornecem conjuntos de funções e dados interessantes para trabalharmos no processo de aprendizado de Ecologia da paisagem e de R:

```
install.packages(c("sp", "sf", "raster", "mapview", "tmap",
                   "terra", "kableExtra", "landscapemetrics", "siland"))
```

Código no livro

- Objetivo não é de apresentar detalhes sobre os cálculos/métodos estatísticas ou os funções no **R**. Existem diversos exemplos disponíveis “[Ciência de Dados com R–Introdução.....](#)”: e com google “r cran introdução tutorial”..... Alem disso, existem grupos de ajuda, como por exemplo: [R Brasil](#) e [Stack Overflow em Português](#)

- O objetivo é de apresentar um livro mostrando os capacidades e opções para desenvolver e integrar pesquisas na ecologia da paisagem no ambiente estatística de [R](#)

Obviamente, todas as tarefas de geoprocessamento podem ser desenvolvidas anteriormente em um SIG ([QGIS](#)). Então porque use R? R tem a capacidade (baseada em código) para alternar entre tarefas de processamento, modelagem e visualização de dados geográficos e não geográficos. Além disso, como é possível importar, modificar, analisar e visualizar dados espaciais no mesmo ambiente com script/código, o R permite fluxos de trabalho transparentes e reproduzíveis ([A Ciência Aberta](#)).

Aliás, atualmente a grande maioria dos artigos científicos publicados na revista [Landscape Ecology](#) incluir análises usando R.

Organização do código no livro

O capítulo está organizado em etapas de processamento, com blocos de código em caixas cinzas:
codigo de R para executar

Para seguir os passos, os blocos de código precisam ser executados em sequência. Se você pular uma etapa, ou rodar fora de sequência o próximo bloco de código provavelmente não funcionará.

As linhas de código de R dentro de cada caixa também precisam ser executadas em sequência. O símbolo `#` é usado para incluir comentários sobre os passos no código (ou seja, linhas começando com `#` não são código de execução).

```
# Passo 1  
codigo de R passo 1 # texto e números têm cores diferentes  
# Passo 2  
codigo de R passo 2  
# Passo 3  
codigo de R passo 3
```

Aém disso, os símbolos `#>` e/ou `[1]` no início de uma linha indicam o resultado que você verá no console de R.

```
# Passo 1  
1+1
```

```
[1] 2
```

```
# Passo 2  
x <- 1 + 1  
# Passo 3  
x
```

```
[1] 2
```

```
# Passo 4  
x + 1
```

```
[1] 3
```

Part II

Escala e padrões

1 Escala

1.1 Apresentação

Nesta capítulo vamos entender a importância de escala na ecologia da paisagem através cálculos com a proporção de floresta. Durante o capítulo você aprenderá a

1. Alterar escala (resolução e extensão espacial),
2. Calcular a área de uma classe de habitat,
3. Desenvolve uma comparação multiescala.

É muito importante ficar claro para você o que é escala (e o que não é!), e qual a importância desse conceito na elaboração do desenho amostral, na coleta de dados, nas análises e na tomada de decisão. Nesse tutorial usaremos conteúdo baseado no Capítulo 2 do livro [Spatial Ecology and Conservation Modeling](#) (Fletcher and Fortin 2018) e “[Tutorial Escala](#)” do Dr. Alexandre Martensen.

Para ajudar a acompanhar e entender os exemplos no capítulo, você deve ler os seguintes artigos:

- Best, J. 2019.
Anthropogenic stresses on the world's big rivers. *Nature Geoscience* 12, 7–21. <https://doi.org/10.1038/s41561-018-0262-x>
- Raffo DCD, Norris D, Hartz SM, Michalski F. 2022.
Anthropogenic influences on the distribution of a threatened apex-predator around sustainable-use reserves following hydropower dam installation. *PeerJ* 10:e14287 <https://doi.org/10.7717/peerj.14287>
- Bárcenas-García, A., Michalski, F., Gibbs, J. P., & Norris, D. (2022). Amazonian run-of-river dam reservoir impacts underestimated: Evidence from a before–after control–impact study of freshwater turtle nesting areas. *Aquatic Conservation: Marine and Freshwater Ecosystems*, 32(3), 508–522. <https://doi.org/10.1002/aqc.3775>
- Sá-Oliveira JC, Hawes JE, Isaac-Nahum VJ, Peres C 2015.
Upstream and downstream responses of fish assemblages to an eastern Amazonian hydroelectric dam. *Freshwater biology*. Oct;60(10):2037-50. <https://doi.org/10.1111/fwb.12628>

1.2 Escala: breve definição

Todos os processos e padrões ecológicos têm uma dimensão temporal e espacial. Assim sendo, o conceito de escala não somente representar essas dimensões, mas também, ajudar nos apresentá-los de uma forma que facilite o entendimento sobre os processos e padrões sendo estudados.

Na ecologia o termo escala refere-se à dimensão ou domínio espaço-temporal de um processo ou padrão. Na ecologia da paisagem, a escala é frequentemente descrita por sua componentes: resolução e extensão.

- **Resolução:** menor unidade espacial de medida para um padrão ou processo.
- **Extensão:** descreve o comprimento ou tamanho de área sob investigação.

Resolução e extensão tendem a covariar – estudos com maior extensão tendem a ter resolução maiores também. Parte dessa covariância é prática: é difícil trabalhar em grandes extensões com dados coletados em tamanhos de resolução finos. No entanto, parte dessa covariância também é conceitual: muitas vezes em grandes extensões, podemos esperar que processos operando em resolução muito fina forneçam somente “ruído” e não dados/informações relevantes sobre os sistemas. Como os desafios computacionais diminuíram e a disponibilidade de dados de alta resolução aumentou, a covariância entre resolução e extensão nas investigações diminuiu.

Lembrando, na primeira aula, vimos que a escala espacial pode ser interpretada com base em três dimensões:

- no fenômeno de interesse;

- na amostragem que ocorre;

e/ou

- na análise

Para que a Ecologia da Paisagem gere evidências científicas robustas e úteis, a escala nas três dimensões deve ser consistente e apropriada para o estudo. Incompatibilidade entre a escala de análise (definida por o pesquisador) e a escala de efeito (definida pelo verdadeiro relacionamento) representa uma versão do Problema de Unidade Areal Modificável (“Modifiable Areal Unit Problem” (Gehlke and Biehl 1934)), em que os resultados e a inferência torna-se um artefato do esquema de amostragem espacial em vez de um processo ecológico (Jelinski and Wu 1996; Wu and Li 2006).

Aqui nos concentramos na dimensão “análise”, e aprendemos como a escala espacial pode ser alterada e representada em modelos ecológicos.

1.3 Pacotes e dados

Em geral é necessário baixar alguns pacotes para que possamos fazer as nossas análises. Neste caso precisamos os seguintes pacotes, que deve estar instalado antes:

- [tidyverse](#),
- [sf](#),
- [terra](#),
- [mapview](#),
- [tmap](#).

1.3.1 Pacotes

No R, carregar os pacotes necessários com o código:

```
library(tidyverse)
library(sf)
library(terra)
library(mapview)
library(tmap)
library(eprdados)
```

Caso os pacotes não tenham sido instalados, o R vai avisar através de uma mensagem tipo: [Error in library\(nomepacote\) there is no package called nomepacote](#). Neste caso, para instalá-los consulte o capítulo aqui [Capítulo 4 instalação de pacotes](#) e aqui [Capítulo 4 pacotes](#).

1.3.2 Dados

Vamos olhar um exemplo do mundo real. Uma pequena amostra do Rio Araguari, perto de Porto Grande. O ponto central é de longitude: -51.406312 latitude: 0.726236. Para visualizar o ponto no Google Earth: <https://earthengine.google.com/timelapse?v=0.72154,-51.41543,11.8,latLng&t=2.24&ps=25&bt=19840101&et=20201231&startDwell=0&endDwell=0>.

Vamos trabalhar com os dados de [MapBiomas](#), que produz mapeamento anual da cobertura e uso da terra no Brasil desde 1985. Os dados de MapBiomas vêm no formato de raster, que tem uma classificação da terra feita a partir da classificação pixel a pixel de imagens das satélites Landsat. Todo processo é feito com algoritmos de aprendizagem de máquina (machine learning) através da plataforma Google Earth Engine, que oferece imensa capacidade de processamento na nuvem. Mais detalhes sobre a metodologia aqui: [Metodologia MapBiomas](#).

Para carregar um arquivo raster trabalhamos com o pacote [terra](#). O pacote tem vários funções para análise e modelagem de dados geográficos. Nós podemos ler os dados de cobertura da terra no arquivo “.tif” com a função [rast](#).

```
# arquivo no pacote "eprdados"
arquivo <- system.file("raster/amostra_mapbiomas_2020.tif",
                      package = "eprdados")
# carregar
ramostra <- rast(arquivo)
```

Plotar para verificar.

```
plot(ramostra)
```

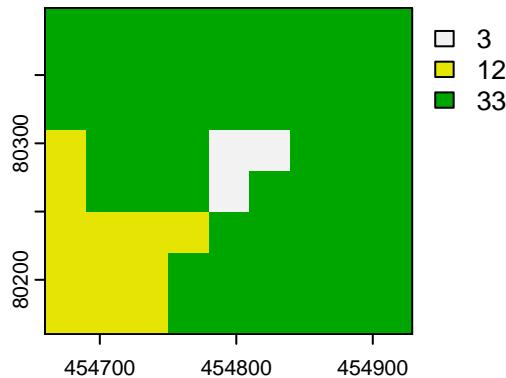


Figura 1.1: Mapbiomas 2020. Uma pequena amostra do Rio Araguari, perto de Porto Grande.

Podemos também verificar informações sobre o raster (metadados) - simplesmente rodar o nome do objeto: [ramostra](#)

```
## class      : SpatRaster
## dimensions : 8, 9, 1  (nrow, ncol, nlyr)
## resolution : 29.89281, 29.89281  (x, y)
## extent     : 454659.8, 454928.9, 80160.06, 80399.2  (xmin, xmax, ymin, ymax)
## coord. ref. : SIRGAS 2000 / UTM zone 22N (EPSG:31976)
## source     : amostra_mapbiomas_2020.tif
## name       : mapbiomas_2020
## min value  : 3
```

Isso nos mostra informações sobre os atributos da raster no objeto ramostra. As informações como escala espacial (resolução e extensão) e a sistema de coordenadas (SIRGAS 2000 / UTM zone 22N , [EPSG:31976](#)) são apresentados e representem os componentes conforme a proxima figura.

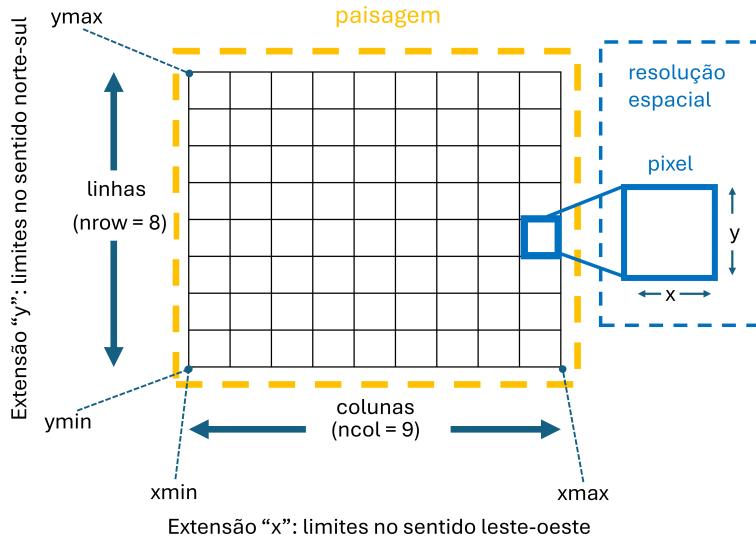


Figura 1.2: Componentes de uma raster e suas atrubutos no pacote terra.

Além disso é possível obter informações específicas através de funções específicas.

```
# Obter informações sobre escala espacial
# resolução, comprimento e largura do pixel em metros
res(ramostra)
# numero de colunas
ncol(ramostra)
# numero de linhas
nrow(ramostra)
```

1.3.2.1 Pergunta 1 Sobre o objeto ramostra. Com base nos resultados obtidos, qual o área do pixel em metros quadrados? Qual o área total da paisagem em hectares e quilometros quadrados?

Olhando a mapa (Figura 1.1), existem três classes com valores de 3, 12 e 33. O objetivo principal não é de fazer mapas, mas, a visualização dos dados é um passo importante para verificar e entender os padrões. Portanto, segue exemplo mostrando uma forma de visualizar o arquivo de raster como mapa.

Para entender o que os valores (3, 12, 33) representam no mundo real precisamos de uma referência (legenda). Para a MapBiomas Coleção 6, arquivo: [Cod_Class_legenda_Col6_MapBiomas_BR.pdf](#). Existe também arquivos para fazer as mapas com cores corretas em [QGIS](#) ou [ArcGIS](#).

Olhando a legenda ([Cod_Class_legenda_Col6_MapBiomas_BR.pdf](#)), sabemos que “3”, “12” e “33” representem cobertura de “Formação Florestal”, “Formação Campestre”, e “Rio, Lago e Oceano”. Então podemos fazer um mapa mostrando tais informações.

Daqui pra frente vamos aproveitar uma forma mais elegante de apresentar mapas e gráficos. Isso seria através funções em 2 pacotes:

- [tmap](#) é utilizado para gerar mapas temáticos
- [ggplot2](#), que faz parte do “tidyverse”, é utilizado para produção de gráficos, e pode representar dados geoespaciais.

Exemplos : [Pacotes ggplot2 e tmap](#)

Mais exemplos sobre o uso de [ggplot2](#) no R cookbook : <http://www.cookbook-r.com/Graphs/> .

E com mais exemplos de mapas e dados espaciais no R: [sf](#) e [ggplot2](#) : <https://www.r-spatial.org/r/2018/10/25/ggplot2-sf.html>

[Capítulo 9](#) no livro [Geocomputation with R](#) : <https://geocompr.robinlovelace.net/adv-map.html>

Primeiramente precisamos incluir as informações relevantes da legenda. Ou seja, incluir os nomes para cada valor de classe.

```
# legenda e cores na sequencia correta
classe_valor <- c(3, 12, 33)
classe_legenda <- c("Formação Florestal",
                     "Formação Campestre", "Rio, Lago e Oceano")
classe_cores <- c("#006400", "#B8AF4F", "#0000FF")
```

Agora podemos fazer o mapa com as classes e os cores seguindo o padrão recomendado pela MapBiomas para Coleção 6.

```
tm_shape(ramostra) +
  tm_raster(style = "cat",
             palette = c("3" = "#006400", "12" ="#B8AF4F",
                         "33"= "#0000FF"), legend.show = FALSE) +
  tm_grid(labels.format = list(big.mark = ""))
  tm_add_legend(type = "fill", labels = classe_legenda,
                 col = classe_cores, title = "Classe") +
  tm_compass(position = c("right", "bottom")) +
  tm_scale_bar(breaks = c(0, 0.05, 0.1), text.size = 1,
                text.color = "white", position=c("right", "bottom")) +
  tm_layout(legend.position = c("right","top"),legend.bg.color = "white")
```

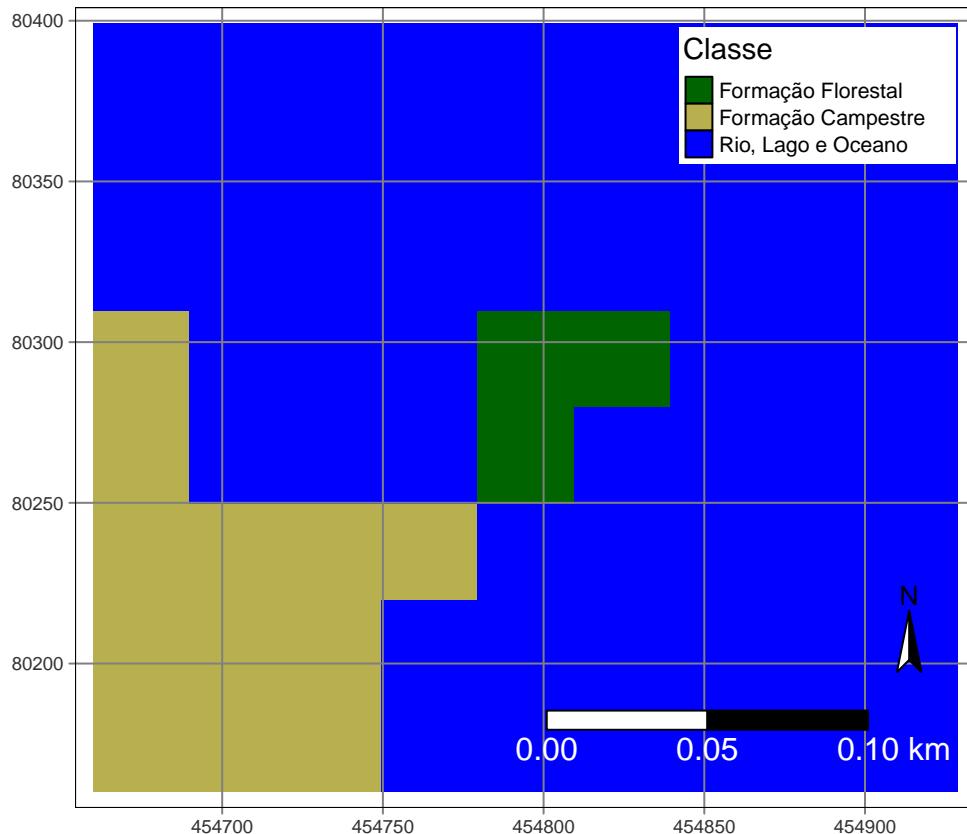


Figura 1.3: Paisagem com valores e classes de cobertura da terra. Mapbiomas 2020. Uma pequena amostra do Rio Araguari, perto de Porto Grande.

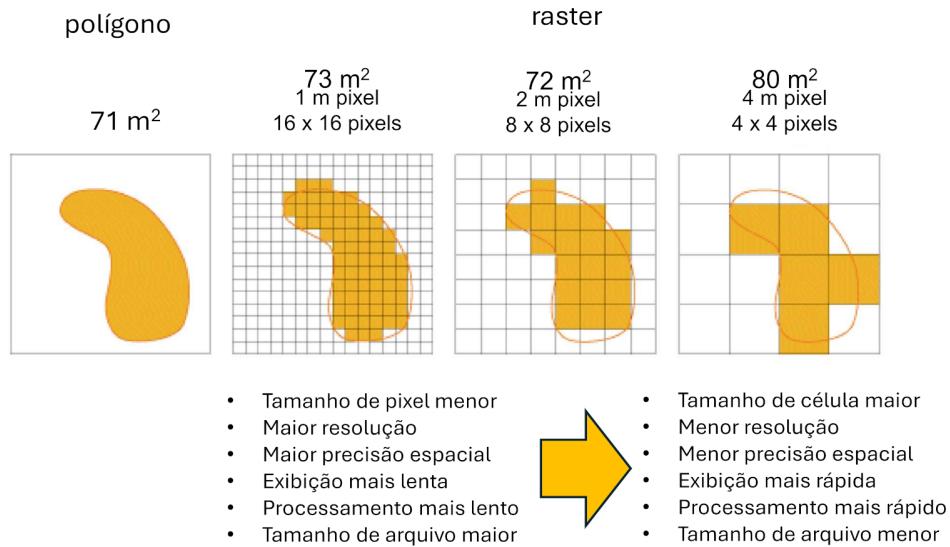
1.4 Alterando a resolução

Alterando a resolução serve como exemplo mostrando como os passos/etapas/cálculos mude dependendo o tipo de dados. Ou seja, é preciso adotar metodologias diferentes para dados categóricos (por exemplo classificação de cobertura da terra) e dados contínuos (por exemplo distância até rio).

Alterando a resolução às vezes seria necessário, por exemplo, quando preciso padronizar dados/imagens oriundos de fontes diferentes com resoluções diferentes e/ou para reduzir a complexidade da modelagem. Lembrando - em cada nível de resolução, são observáveis processos e padrões que não podem necessariamente ser inferidos daqueles abaixo ou acima.

A principal razão para tornar o mapa mais grosso é traduzi-lo para uma resolução de dados coletados em campo que estamos usando para fazer inferências. Por exemplo, se os dados foram coletados ao longo de dois transectos de 200 x 100 m dentro de manchas florestais. Se desejarmos fazer previsões das relações espécie-ambiente, podemos querer que o grão do nosso mapa reflita o grão da amostragem. Consequentemente, gostaríamos que o mapa tivesse um grão aproximado de 200 x 200 m.

Ao escolher um tamanho de pixel apropriado, equilibre a resolução espacial desejada – com base na unidade mínima de mapeamento das variáveis que você precisa analisar – com requisitos práticos para exibição rápida, tempo de processamento e armazenamento. Essencialmente, num SIG, os resultados são tão precisos quanto o conjunto de dados menos preciso. Se você estiver usando um conjunto de dados classificados derivado de imagens de satélite com resolução de 30 metros, o uso de um modelo digital de elevação ou outros dados auxiliares com resolução mais alta, como 10 metros, pode ser desnecessário. Quanto mais homogênea for uma área para variáveis críticas, como topografia e cobertura, maior poderá ser o tamanho da célula sem



Adaptado de: <https://pro.arcgis.com/en/pro-app/latest/help/data/imagery/pixel-size-of-image-and-raster-data-pro.htm>

Figura 1.4: Comparação de tamanhos de pixels pequenos e grandes. Mostrando diferenças causadas pela agregação de 4 pixels em 1.

afetar a precisão.

Agora iremos degradar a resolução desses dados, ou seja, iremos alterar o tamanho dos pixels. Como exemplo, iremos juntar (agregar) 3 pixels em um único pixel. Como você acha que podemos fazer isso? Quais valores esse pixel que vai substituir os 3 originais deve ter? Existem diversas maneiras de se fazer isso, como por exemplo através a média ou valor modal (valor mais comum). O valor mais comum da área, é particularmente adequado quando temos um mapa categórico, como por exemplo a classificação do MapBiomas. Segue exemplo de código para agregar com a média e o valor mais frequente (modal).

```
# Média
ramostra_media <- aggregate(ramostra, fact=3, fun="mean")
ramostra_media <- resample(ramostra, ramostra_media)

# Modal
ramostra_modal <- aggregate(ramostra, fact=3, fun="modal")
ramostra_modal <- resample(ramostra, ramostra_modal, method="near")
```

1.4.0.1 Pergunta 2 Utilizando as funções disponíveis no pacote `tmap`, crie mapas temáticos dos objetos `ramostra_media` e `ramostra_modal`. Inclua cópias do seu código e mapas na sua resposta. Você pode usar o printscreenshot para mostrar o RStudio com seu código e mapas.

Visualizar os resultados apresentados em Figura 1.4. Os valores calculados pela média não fazem sentido para uma classificação categórica. Os valores calculados pela modal são consistentes com o original e fazem sentido.

Em cada nível de resolução, são observáveis processos e padrões que não podem necessariamente ser inferidos daqueles abaixo ou acima. Aqui por exemplo, mudamos a proporção de cobertura florestal em nossa pequeno paisagem quando juntamos 3 pixels em um único: a proporção de floresta moudou de 4% (3/72) para 11% (1/9). Ou seja, com cada passo mudamos a representação do mundo.

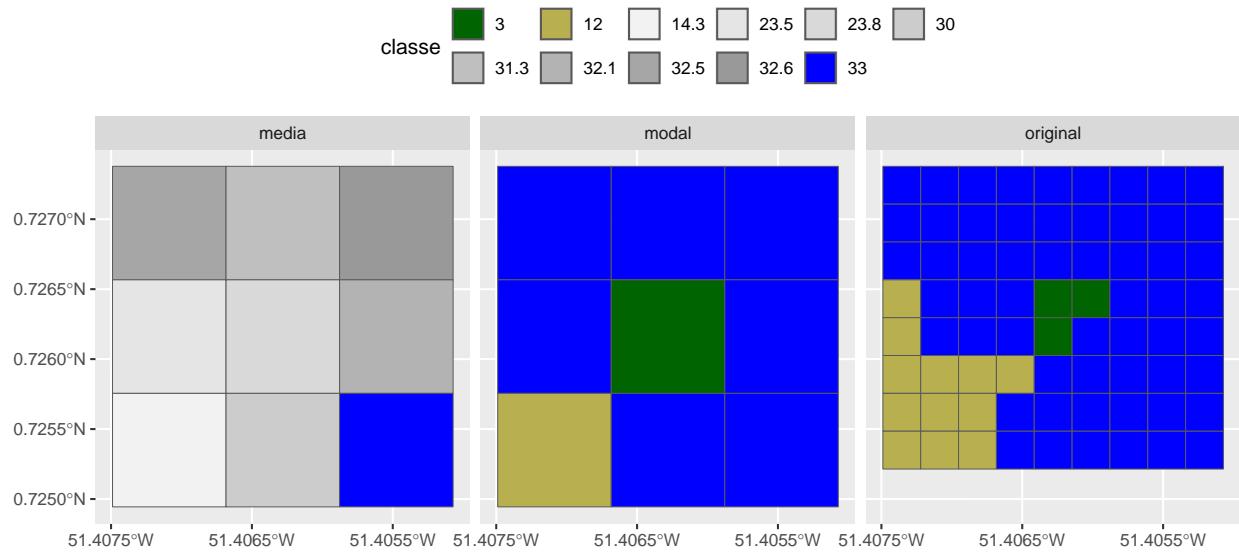


Figura 1.5: Mudanças causadas pela agregação.

1.4.0.2 Pergunta 3 Confira o código e os resultados obtidos anteriormente, quando mudamos a resolução da raster ramostra (por exemplo figura 1.4). Explique o que aconteceu. Como e porque moudou os valores em cada caso (média e modal)?

1.5 Escala espacial e desenho amostral

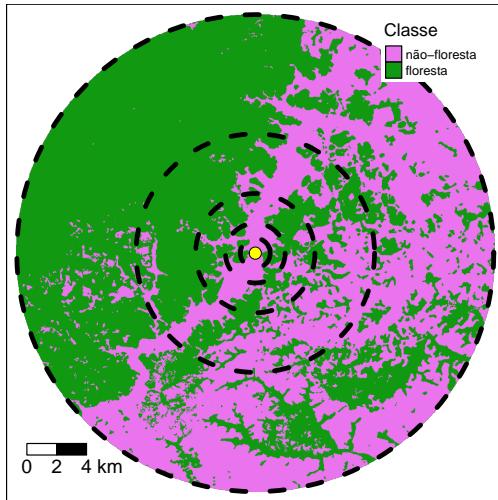


Figura 1.6: A cobertura florestal ao redor de um ponto de amostragem pode variar em escalas diferentes. Para entender essas variações, podemos criar buffers circulares de diferentes extensões ao redor dos pontos de amostragem. Esses buffers representam áreas de diferentes tamanhos ao redor de cada ponto. Quantificando a quantidade de floresta que ocorre em cada buffer, podemos obter uma visão geral da escala em que a cobertura florestal muda ao redor dos pontos de amostragem. Por exemplo, podemos descobrir que a cobertura florestal é mais alta dentro de um buffer de 5 km do que em um buffer de 10 km.

Dado o papel que a escala pode desempenhar em nossa compreensão dos padrões e processos ecológicos, como escala deve ser considerada no desenho do estudo? Claramente, a resposta a esta pergunta irá variar dependendo dos fenômenos de interesse, mas ecologistas e estatísticos têm fornecido algumas orientações importantes. As questões-chave incluem o tamanho da unidade de amostragem (resolução), o tipo de unidade de amostra e localizações da unidade de amostra, incluindo o espaçamento entre as amostras (distância entre as amostras) e o tamanho da área de estudo.

Com a disponibilidade de imagens de satélite é possível responder questões importantes relacionadas ao desenho do estudo antes de qualquer trabalho de campo. Uma técnica de geoprocessamento (zona tampão - [Buffers](#)) é um dos mais frequentemente adotados para quantificar escala espacial na ecologia da paisagem.

O objetivo é criar buffers circulares de diferentes extensões ao redor dos sitios de amostragem (pontos, pixels, manchas, transets lineares etc). Aqui, vamos entender a escala em que a cobertura de floresta muda ao redor dos rios. Para isso, quantificamos a quantidade de floresta que ocorre em várias distâncias em pontos ao longo dos rios a montante das hidrelétricas no Rio Araguari. Para ilustrar esta abordagem geral, usamos o banco de dados MapBiomas Coleção 6 de 2020, e vincule esses dados de cobertura da terra aos pontos de amostragem em rios.

1.5.1 Obter e carregar dados (vectores)

Antes de quantificar a quantidade de floresta, precisamos carregar os dados de rios e pontos de amostragem. O formato de vector é diferente de “tif” (raster), portanto o processo de importação é diferente. Aqui, nós só precisamos de duas dessas camadas, ambos do pacote ‘eprdados’: “rio_linhacentral” e “rio_pontos”. A primeira camada de dados contém o eixo central de 260 km de rios a montante da Barragem Cachoeira Caldeirão. Os dados foram obtidos a partir de registros de trajetória GPS durante levantamentos de barco. Os rios foram divididos em 52 seções, cada uma com aproximadamente 5 km de extensão. A segunda camada de dados, “rio_pontos”, contém 52 pontos espaçados regularmente ao longo dos rios. Cada ponto está localizado a aproximadamente 5 km de distância do ponto anterior.

1.5.2 Visualizar os arquivos (camadas vector)

Visualizar para verificar. Mapa com ambos a linha central e pontos de rios em trechos de 5km.

```
ggplot(rio_linhacentral) +  
  geom_sf(aes(color=rio)) +  
  geom_sf(data = rio_pontos, shape=21, aes(fill=zone))
```

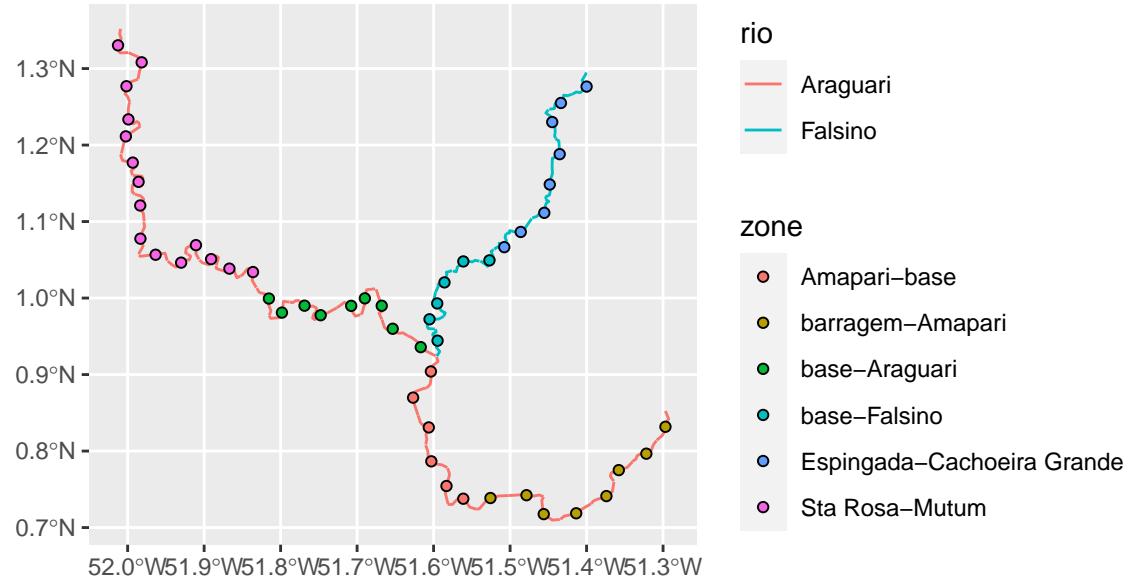


Figura 1.7: Pontos ao longo dos rios a montante das hidrelétricas no Rio Araguari.

Mapa interativo (funcione somente com internet) Mostrando agora com fundo de mapas “base” (OpenStreetMap/ESRI etc)

```
#  
mapview(rio_linhacentral, zcol = "rio")
```

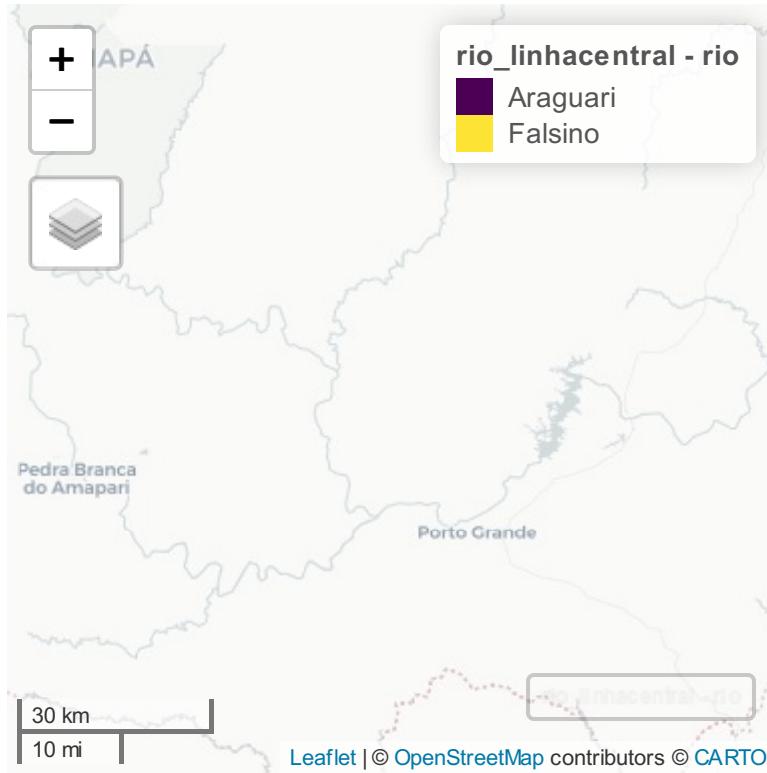


Figura 1.8: Linhas dos rios a montante das hidrelétricas no Rio Araguari.

1.5.3 Obter e carregar dados (raster)

Mais uma vez vamos aproveitar os dados de MapBiomas. Agora um arquivo raster com cobertura de terra no entorno dos rios em 2020, (formato “.tif”, tamanho 1.3 MB). O código abaixo vai carregar os dados e criar o objeto “mapbiomas_2020”:

```
meuSIGr <- system.file("raster/utm_cover_AP_rio_2020.tif",
                       package = "eprdados")
# carregar
mapbiomas_2020 <- rast(meuSIGr)
```

1.5.4 Visualizar os arquivos (camadas raster e vector)

Visualizar para verificar. É possível de visualizar varios camadas de raster e vetor juntos com funções no pacote tmap (<https://r-tmap.github.io/tmap-book/index.html>).

```
# Passo necessário para agilizar o processamento
mapbiomas_2020_modal <- aggregate(mapbiomas_2020,
                                      fact = 10,
                                      fun = "modal")

# Plot
tm_shape(mapbiomas_2020_modal) +
  tm_raster(title = "Classe", style = "cat", palette = "Set3") +
  tm_shape(rio_linhacentral) +
  tm_lines(col="blue") +
  tm_shape(rioPontos) +
  tm_dots(size = 0.2, col = "yellow") +
  tm_compass(position=c("left", "top")) +
```

```
tm_scale_bar(breaks = c(0, 25, 50), text.size = 1,  
            position=c("left", "bottom")) +  
tm_layout(legend.position = c("right", "top"), legend.bg.color="white")
```

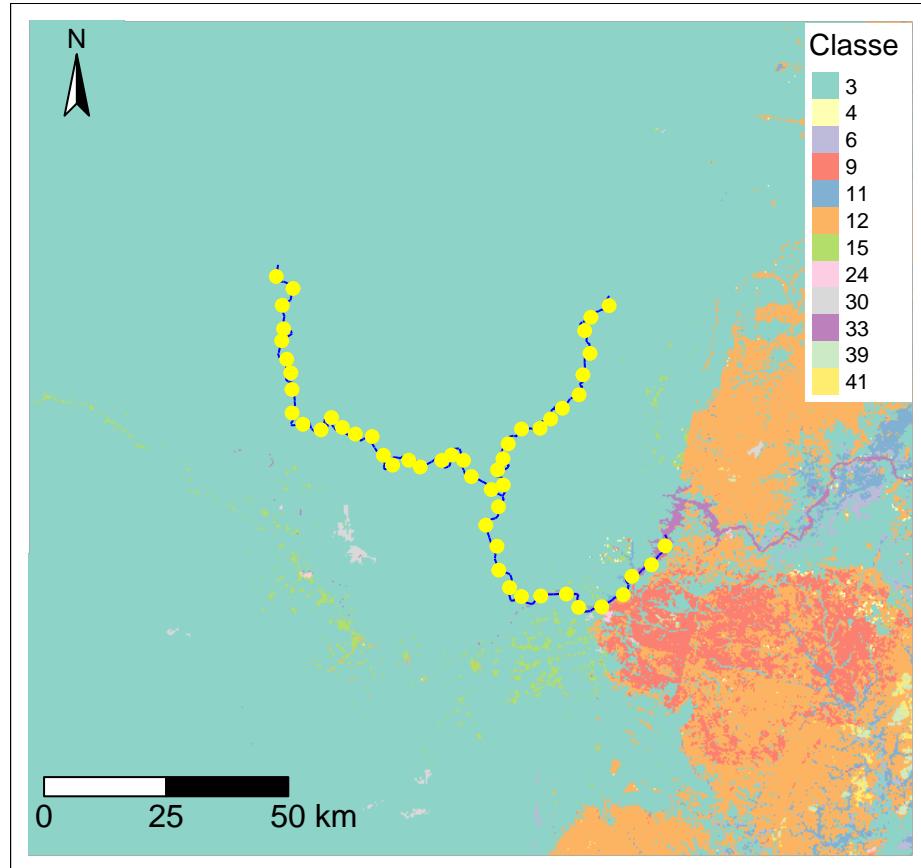


Figura 1.9: Cobertura da terra ao redor do Rio Araguari em 2020. Mostrando os pontos de amostragem (pontos amarelas) cada 5 quilômetros ao longo do rio.

1.5.5 Reclassificação

Para simplificar nossa avaliação de escala, reclassificamos a camada mapbiomas_2020 em uma camada binária de floresta/não-floresta. Essa tarefa de geoprocessamento pode ser realizada anteriormente usando SIG ([QGIS](#)). Aqui vamos reclassificar as categorias de cobertura da terra (agrupando diferentes áreas de cobertura florestal tipos) usando alguns comandos genéricos do R para criar uma nova camada com a cobertura de floresta em toda a região de estudo. Para isso, criamos um mapa do mesmo resolução e extensão, e então podemos redefinir os valores do mapa. Neste caso, queremos agrupar a cobertura da terra categorias 3 e 4 (Formação Florestal e Formação Savânica, respectivamente).

```
# criar uma nova camada de floresta
floresta_2020 <- mapbiomas_2020
# Com valor de 0
values(floresta_2020) <- 0
# Atualizar categorias florestais agrupados com valor de 1
floresta_2020[mapbiomas_2020 == 3 | mapbiomas_2020 == 4] <- 1
```

Vizualizar para verificar.

```
# Passo necessário para agilizar o processamento
floresta_2020_modal <- aggregate(floresta_2020,
                                    fact=10,
                                    fun="modal")

# Plot
tm_shape(floresta_2020_modal) +
  tm_raster(style = "cat",
             palette = c("0" = "#E974ED", "1" ="#129912"), legend.show = FALSE) +
  tm_add_legend(type = "fill", labels = c("não-floresta", "floresta"),
                col = c("#E974ED", "#129912"), title = "Classe") +
  tm_shape(rio_linhacentral) +
  tm_lines(col="blue") +
  tm_shape(rioPontos) +
  tm_dots(size = 0.2, col = "yellow") +
  tm_scale_bar(breaks = c(0, 25, 50), text.size = 1,
               text.color = "white", position=c("left", "bottom")) +
  tm_layout(legend.position = c("right","top"), legend.bg.color = "white")
```

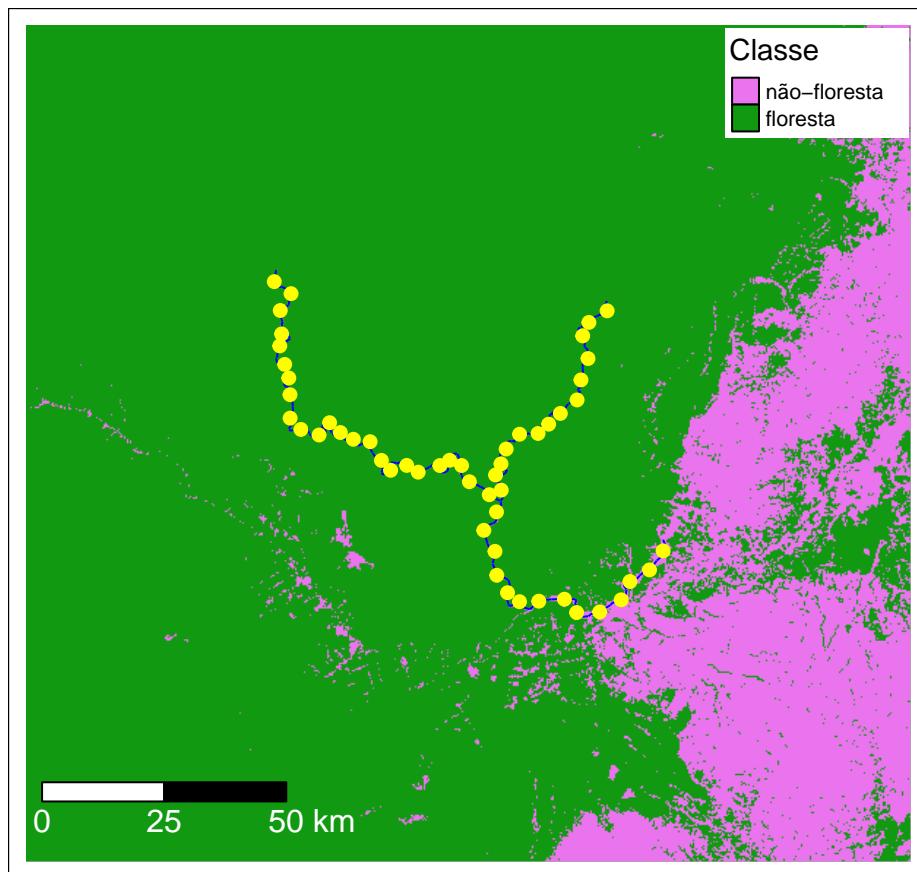


Figura 1.10: Floresta ao redor do Rio Araguari. MapBiomas 2020 reclassificado em floresta e não-floresta. Mostrando os pontos de amostragem (pontos amarelos) cada 5 quilômetros ao longo do rio.

1.6 Comparação multiescala

Em seguida, com as coordenadas dos pontos de amostragem, podemos calcular a quantidade de floresta que circunda cada local de amostragem em diferentes extensões. Primeiramente, vamos fazer só para um ponto, assim para entender o processo e os passos melhor.

```
rioPontos_31976 <- st_transform(rioPontos, 31976)
# Buffer
rioPontos_31976_b1000 <- st_buffer(rioPontos_31976[1, ], dist = 1000)

# Recorte com buffer de 1000 metros (mudando a extensão).
# Máscara (mask), assim os pixels fora do polígono sejam nulos.
buffer.forest1.1km <- terra::crop(floresta_2020, rioPontos_31976_b1000,
                                     snap="out", mask = TRUE)
names(buffer.forest1.1km) <- "forest_2020_1km"
```

Vizualizar para verificar.

```
# Plot
tm_shape(buffer.forest1.1km) +
  tm_raster(style = "cat",
             palette = c("0" = "#E974ED",
                         "1" ="#129912"), legend.show = FALSE) +
tm_shape(rio_pontos_31976[1, ]) +
  tm_symbols(shape =21, col = "yellow",
             border.col = "black", border.lwd = 0.2, size=0.5) +
tm_shape(rio_pontos_31976_b1000) +
  tm_borders(col = "black", lwd = 4, lty = "dashed") +
tm_add_legend(type = "fill", labels = c("não-floresta", "floresta"),
              col = c("#E974ED", "#129912"), title = "Classe") +
tm_compass(position=c("left", "top")) +
tm_scale_bar(breaks = c(0, 0.5, 1), text.size = 1,
             position=c("left", "bottom")) +
tm_layout(legend.position = c("right","top"), legend.bg.color = "white")
```

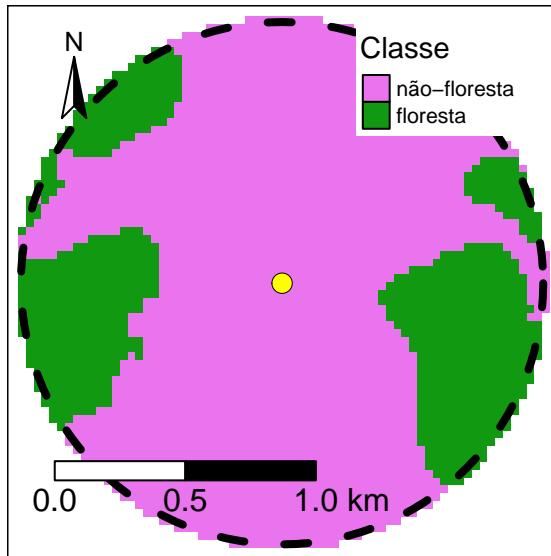


Figura 1.11: Ilustração da determinação da quantidade de habitat ao redor de um ponto. Para um determinada extensão, o habitat de interesse é isolado. Um buffer (linha tracejada) é colocado ao redor de um ponto (amarela) e o número de células (pixels) que contém o habitat é somado e multiplicado pela área de cada pixel.

1.6.0.1 Pergunta 4 Qual é a extensão em número de pixels desse recorte (buffer.forest1.1km)?

Temos valores de 0 (não-floresta) e 1 (floresta). Então, para saber a área de floresta podemos somar o número de células (pixels) que contém o habitat e multiplica pela área de cada pixel conforme o código:

```
# 1) Somatório.  
# No caso igual o numero de pixels de floresta.  
# Para todo a paisagem, somatorio "global".  
# Não deve incluir pixels nulos, então use "na.rm = TRUE".  
soma_floresta <- global(buffer.forest1.1km, "sum", na.rm = TRUE)  
soma_floresta  
  
## sum  
## forest_2020_1km 939  
  
# 2) Área de cada pixel.  
# Sabemos o sistema de coordenadas (EPSG = 31976).  
# EPSG 31976 é uma sistema projetado com unidade em metros.  
buffer.forest1.1km  
  
## class : SpatRaster  
## dimensions : 68, 68, 1 (nrow, ncol, nlyr)  
## resolution : 29.89281, 29.89281 (x, y)  
## extent : 465959.3, 467992, 90921.47, 92954.18 (xmin, xmax, ymin, ymax)  
## coord. ref. : SIRGAS 2000 / UTM zone 22N (EPSG:31976)  
## source(s) : memory  
## varname : utm_cover_AP_rio_2020  
## name : forest_2020_1km  
## min value : 0  
## max value : 1  
  
# Portanto, o tamanho de cada pixel é igual.  
area_pixel_m2 <- 29.89281 * 29.89281  
area_pixel_m2  
  
## [1] 893.5801  
  
# 3) Calculos de área.  
# Área de floresta m2  
area_floresta_m2 <- soma_floresta * area_pixel_m2  
area_floresta_m2  
  
## sum  
## forest_2020_1km 839071.7  
  
# Área de floresta hectares  
area_floresta_ha <- area_floresta_m2 / 10000  
area_floresta_ha  
  
## sum  
## forest_2020_1km 83.90717
```

Para uma comparação multiescala, vamos repetir o mesmo processo, mas agora com distâncias de 250, 500, 1000, 2000 e 4000 metros, dobrando a escala (extensão) em cada passo.

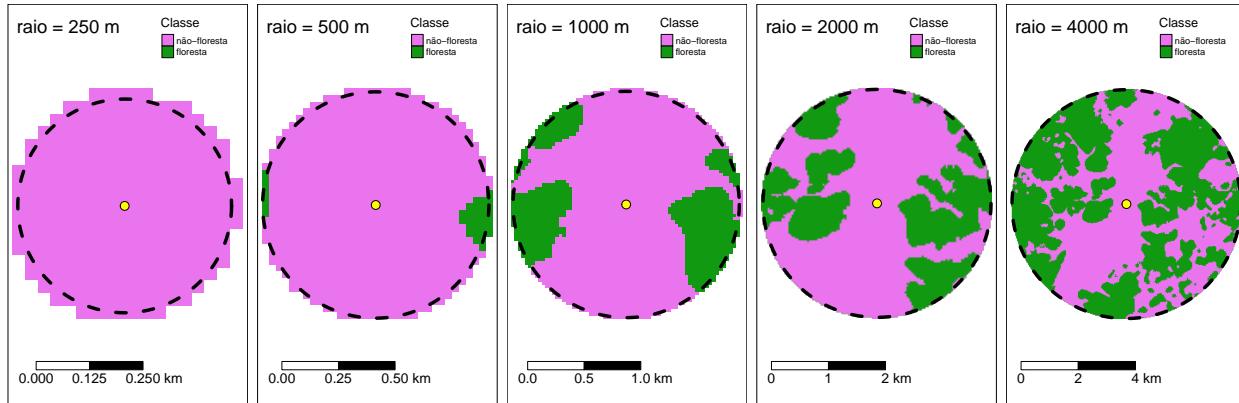


Figura 1.12: Cobertura florestal em extensões diferentes ao redor de um local de amostragem.

Aspectos quantitativos das paisagens mudam fundamentalmente com a escala. Por exemplo, nesse caso, parece que a proporção de floresta aumenta à medida que a extensão aumenta de 500 para 4000 metros. Esta percepção visual é confirmada pelos valores calculados, onde as áreas são:

- raio 250 m = 0 hectares de floresta
- raio 500 m = 6,3 hectares de floresta
- raio 1000 m = 84,3 hectares de floresta
- raio 2000 m = 502,6 hectares de floresta
- raio 4000 m = 3351,0 hectares de floresta

1.6.0.2 Pergunta 5 Usando os valores listadas acima de raio e área de floresta para os diferentes buffers circulares, calcule a proporção de floresta em cada uma das diferentes extensões de buffer. Apresente 1) os resultados incluindo cálculos. 2) um gráfico com valores de extensão no eixo x e proporção da floresta no eixo y. 3) Em menos de 200 palavras apresente a sua interpretação do gráfico.

1.6.0.3 Pergunta 6 A modelagem multiescala quantifica as condições do ambiente em múltiplas escalas alterando a resolução ou a extensão da análise e, em seguida, avaliando qual das escalas consideradas explica melhor um padrão ou processo. Escolha 1 espécie aquática e 1 espécie terrestre que ocorram na região a montante das hidrelétricas no Rio Araguari. Com base nas diferenças entre extensões (indicados no exemplo anterior) e as características funcionais das espécies (por exemplo área de vida), escolher as extensões mais adequadas para um estudo multiescala de cada espécie.

Soluções dos exercícios: https://darrenorris.github.io/eprsol/solutions01_escala.html

1.7 Próximos passos: repetindo para muitas amostras.

Neste exemplo comparamos a área de floresta em torno de um único ponto de amostragem. Para calcular o mesmo para todos os 52 pontos, seriam necessárias várias repetições (52 pontos x 5 extensões = 260 repetições).

Poderíamos escrever código para executar esse processo automaticamente. Felizmente, alguém já escreveu funções para fazer isso e muito mais. O próximo tutorial sobre métricas de paisagem mostrará exemplos usando o pacote “landscapemetrics” (<https://r-spatialecology.github.io/landscapemetrics/>).

2 Métricas da paisagem

2.1 Apresentação

As métricas da paisagem nos ajudam a entender as mudanças na paisagem de diferentes perspectivas (visual, ecológica, cultural).

Assim sendo, análises com métricas de paisagem é um atividade fundamental na ecologia da paisagem. Nesta capítulo aprenderemos sobre como analisar a cobertura da terra com métricas de paisagem em R. o objetivo principal é mostrar a importância de selecionar a métrica “certa” para o problema “certo” usando R. As tecnicas será ilustrada através cálculos usando a cobertura florestal ao redor do Rio Araguari, Amapá, Brasil.

Ao longo do caminho, revisaremos modelos lineares e não lineares, aprenderemos sobre manipulação de dados em R e aprenderemos como criar gráficos com o pacote `ggplot2`. No capítulo você aprenderá a:

- Importar e plotar dados raster em R e mapear locais de amostragem com os pacotes `terra`, `sf` e `tmap`.
- Calcular métricas de paisagem com o pacote `landscapemetrics`.
- Calcular métricas de paisagem em locais de amostragem e dentro de um buffer ao redor deles (com comparação multiescala).
- Construir gráficos com o pacote `ggplot2`.
- Comparação de padrões lineares e não-lineares.

Para ajudar a acompanhar e entender os exemplos no capítulo, use como base as aulas (“Métricas da paisagem: Modelo mancha-corredor-matriz” e “Índices de Paisagem e Análises de Padrões Espaciais”) e você deve ler os seguintes artigos:

- Tischendorf, L. 2001.
Can landscape indices predict ecological processes consistently?. *Landscape Ecology* 16, 235–254. <https://doi.org/10.1023/A:1011112719782>
- Casimiro PC. 2009.
Estrutura, composição e configuração da paisagem conceitos e princípios para a sua quantificação no âmbito da ecologia da paisagem. *Revista portuguesa de estudos regionais*. (20):75-99. <https://www.redalyc.org/pdf/5143/514351895006.pdf>
- Pereira JL, et. al. 2001.
Métricas da paisagem na caracterização da evolução da ocupação da Amazônia. *Geografia*. 2001:59-90. <https://www.periodicos.rc.biblioteca.unesp.br/index.php/ageteo/article/view/1907>
- McGarigal K. 2017.
Landscape metrics for categorical map patterns. Lecture notes. <https://opencourses.ionio.gr/modules/document/file.php/TFP122/%CE%98%CE%95%CE%A9%CE%A1%CE%99%CE%91/%CE%94%CE%95%CE%99%CE%9A%CE%A4%CE%95%CE%A3%20%CE%A4%CE%9F%CE%A0%CE%99%CE%9F%CE%A5/Landscape%20metrics.pdf>

Cópias dos artigos estão no Google Classroom e também podem ser baixadas aqui: https://drive.google.com/drive/folders/1n6KrIiqu8IPvKLdhcKqwpI5o3MB_4gEu?usp=sharing

2.2 Métricas da paisagem e pacote “landscapemetrics”

As métricas de paisagem são a forma que os ecólogos de paisagem usam para descrever os padrões espaciais de paisagens para depois avaliar a influência destes padrões espaciais nos padrões e processos ecológicos.

A paisagem objeto de análise é definida pelo usuário e pode representar qualquer fenômeno espacial. As métricas simplesmente quantifica a heterogeneidade espacial da paisagem; cabe ao usuário estabelecer uma base sólida para definir e dimensionar a paisagem em termos de conteúdo temático e resolução e extensão espacial.

É importante ressaltar que o resultado das métricas só é relevante se a paisagem definida for adequado em relação ao fenômeno em consideração.

O uso comum do termo “métricas de paisagem” refere-se exclusivamente a índices desenvolvidos para padrões de mapas categóricos. Métricas de paisagem são algoritmos que quantificam características espaciais específicas de manchas, classes de manchas ou mosaicos inteiros de paisagem. Uma infinidade de métricas foi desenvolvida para quantificar padrões de mapas categóricos. Estas métricas enquadram-se em duas categorias gerais: aquelas que quantificam a composição do mapa sem referência a atributos espaciais, e aquelas que quantificam a configuração espacial do mapa, necessitando de informação espacial para o seu cálculo [McGarigal & Marks 1995, https://doi.org/10.2737/PNW-GTR-351](https://doi.org/10.2737/PNW-GTR-351).

O pacote [landscapemetrics](#) tem funções para calcular métricas de paisagem em paisagens categóricos (onde tem uma classificação de cobertura de terra/habitat - modelo mancha-corredor-matriz), em um fluxo de trabalho organizado (Hesselbarth et al. 2019). Existem diversos tutoriais e exemplos no site do próprio pacote: <https://r-spatialecology.github.io/landscapemetrics/>.

O pacote [landscapemetrics](#) pode ser usado como um substituto do FRAGSTATS ([McGarigal & Marks 1995, https://doi.org/10.2737/PNW-GTR-351](https://doi.org/10.2737/PNW-GTR-351)), pois oferece um fluxo de trabalho reproduzível para análise de paisagem em um único ambiente. FRAGSTATS é um programa de análise de padrões espaciais para mapas categóricos disponível apenas para o sistema operacional Windows. Pode obter FRAGSTATS aqui: <https://fragstats.org/>. Além de métricas padrão [landscapemetrics](#) também permite cálculos de quatro métricas teóricas de complexidade da paisagem: entropia marginal, entropia condicional, entropia conjunta e informação mútua ([Nowosad e Stepinski 2019 https://doi.org/10.1007/s10980-019-00830-x](https://doi.org/10.1007/s10980-019-00830-x)).

2.2.1 Pacotes

Além do “[landscapemetrics](#)”, precisamos carregar alguns pacotes a mais para facilitar a organização e apresentação de dados espaciais (vector e raster) e os resultados.

Carregar pacotes (que deve estar instalado antes):

```
library(tidyverse)
library(sf)
library(raster)
library(terra)
library(tmap)
library(gridExtra)
library(kableExtra)
library(mgcv)
library(eprdados)
```

Caso os pacotes não tenham sido instalados, o R vai avisar através um mensagem tipo: **Error in library(nomepacote) there is no package called nomepacote**. Neste caso, para instalá-los consulte o capítulo aqui [Capítulo 4 instalação de pacotes](#) e aqui [Capítulo 4 pacotes](#).

2.2.1.1 [landscapemetrics](#) Agora, vamos para o pacote principal [landscapemetrics](#). Digite o código abaixo e veja o resultado. Leia com atenção e preste particular atenção na organização da página de ajuda.

```
library(landscapemetrics)
# Olhar a pagina de ajuda
?landscapemetrics
```

No final da página você vai encontrar a palavra “Index”. Clique nela e você verá todas as funções do pacote. Desça até as lsm_. . . e clique em algumas delas ali. Explorar! Para listar todas as métricas disponíveis, você pode usar a função `list_lsm()`. A função também permite mostrar métricas filtradas por nível, tipo ou nome da métrica. Para obter mais informações sobre as métricas, consulte os arquivos de ajuda correspondentes ou <https://r-spatialecology.github.io/landscapemetrics>.

Digite o código abaixo e veja o resultados, mostrando exemplos das métricas diferentes disponíveis no pacote.

```
# métricas de agregação, nível de fragmento
landscapemetrics::list_lsm(level = "patch", type = "aggregation metric")
# métricas de agregação, nível de classe
landscapemetrics::list_lsm(level = "class", type = "aggregation metric")
#
landscapemetrics::list_lsm(metric = "area")
# ajudar com opções da função
?landscapemetrics::list_lsm
```

Nesse pacote o formato geral para uma função é em três partes “lsm_nível_métrica”. A primeira parte é sempre lsm_ (“landscapemetric”), seguinda do “nível_” e por fim a “métrica”:

1. Ou seja, todas as funções que calculam métricas começam com lsm_
2. Daí você deve incluir o nível da análise: “p”, “c” ou “l”.
Sendo, “p” para patch (ou seja, para a mancha/fragmento), “c” para classe e “l” para landscape ou seja, métricas para a paisagem como um todo.
3. E daí existem inúmeras métricas.

Como por exemplo a `cpland`, que é o percentual de área central - “core area” na paisagem, como vimos na aula teórica. Assim sendo, a função `lsm_c_cpland` vai calcular a métrica porcentagem da área central em cada classe. Lembrando existem metricas que podem se calculados nos três niveis, e metricas que só pode se calculados somente para um nível específico. Ja sabendo o nome da função, podemos buscar ajudar para entender mais detalhes. Digite o código abaixo e veja o resultado, mostrando um exemplo para uma métrica.

```
# ajudar com opções para uma função específica
?landscapemetrics::lsm_c_cpland
```

2.2.1.2 Pergunta 1 Descreva brevemente 2 métricas de cada nível (patch, class, landscape) usando ajudar (usando ? e/ou list_lsm), aulas (Métricas da paisagem: Modelo mancha-corredor-matriz e ”Índices de Paisagem e Análises de Padrões Espaciais”) e/ou a leitura disponivel no Google Classroom (Base teórica 4 Dados, métricas, analyses). Incluindo na descrição - o nome, porque serve, unidades de medida, e relevância ecológica.

2.3 Dados

Vamos continuar as análises que começo no capítulo “Escala”, trabalhando com os mesmos bancos de dados. Então, primeiramente carregar os dados de cobertura da terra com a função rast. E, em seguida implementar uma reclassificação para gerar uma camada binária de floresta (valor de “1”) e não-floresta (valor de “0”).

```
# Carregar
mapbiomas_2020 <- rast(utm_cover_AP_rio_2020)

# Reclassificação -
# Criar uma nova camada de floresta (novo objeto de raster copiando mapbiomas_2020,
# assim para ter os mesmos coordenados, resolução e extensão)
floresta_2020 <- mapbiomas_2020
# Todos os pixels com valor de 0
values(floresta_2020) <- 0
# Atualizar com valor de 1 quando pixels originais são de floresta (classe 3 e 4)
floresta_2020[mapbiomas_2020==3 | mapbiomas_2020==4] <- 1
```

Plotar para verificar, incluindo nomes e os cores para classes de floresta (valor = 1) e não-floresta (valor = 0).

```
# Passo necessário para agilizar o processamento
floresta_2020_modal <- aggregate(floresta_2020,
                                    fact=10,
                                    fun="modal")

# Plot
tm_shape(floresta_2020_modal) +
  tm_raster(style = "cat",
             palette = c("0" = "#E974ED", "1" ="#129912"), legend.show = FALSE) +
  tm_add_legend(type = "fill", labels = c("não-floresta", "floresta"),
                col = c("#E974ED", "#129912"), title = "Classe") +
  tm_layout(legend.bg.color = "white")
```

Se esta todo certo, voces devem ter uma imagem assim:

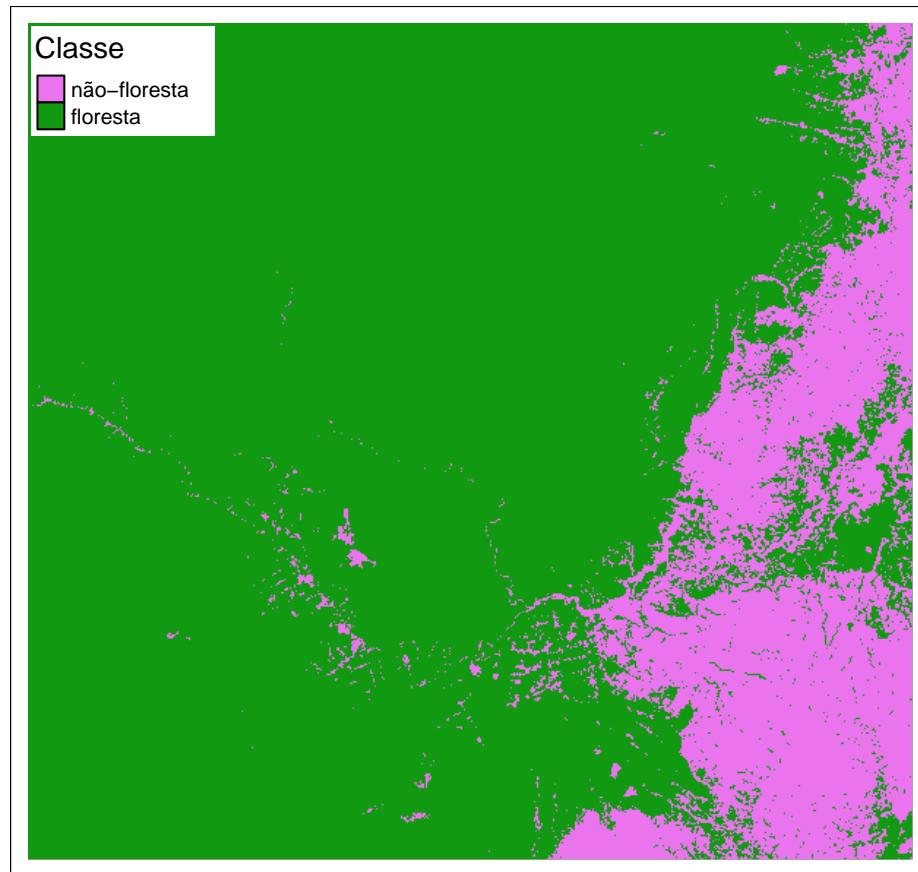


Figura 2.1: Floresta ao redor do Rio Araguari. MapBiomas 2020 reclassificado em floresta e não-floresta.

2.3.1 Exibir dados raster e sobreposição com locais de amostragem

Agora temos a paisagem, precisamos tambem os pontos de amostra. Por isso, precisamos carregar os dados de rios e pontos de amostragem que usamos no tutorial Escala. Vamos carregar as camadas, e no mesmo

tempo implementar uma reprojeção, assim para que as sistemas de coordenados ficam iguais para todas as camadas - tanto de vector quanto raster.

No código abaixo, usamos |>, que estabelece a ligação entre os passos do processo. Ou seja, |> passa o objeto anterior automaticamente para a próxima função como primeiro argumento. Primeiro, carregamos os dados. Em seguida, usamos a função `st_transform` para converter as coordenadas para o mesmo sistema de referência do arquivo raster. As duas etapas ficam ligados através a função |>.

```
# pontos cada 5 km
rioPontos_31976 <- rioPontos |>
  st_transform(31976)
# linha central de rios
rioLinhaCentral_31976 <- rioLinhaCentral |>
  st_transform(31976)
```

Visualizer para verificar.

```
# Passo necessário para agilizar o processamento
floresta_2020_modal <- aggregate(floresta_2020,
                                    fact=10,
                                    fun="modal")

# Mapa
tm_shape(floresta_2020_modal) +
  tm_raster(style = "cat",
             palette = c("0" = "#E974ED", "1" ="#129912"), legend.show = FALSE) +
  tm_add_legend(type = "fill", labels = c("não-floresta", "floresta"),
                col = c("#E974ED", "#129912"), title = "Classe") +
tm_shape(rio_linhacentral_31976) +
  tm_lines(col="blue") +
tm_shape(rioPontos_31976) +
  tm_dots(size = 0.2, col = "yellow") +
tm_layout(legend.bg.color="white")
```

Depois de executar (“run”) o código acima, você deverá ver a figura a seguir.

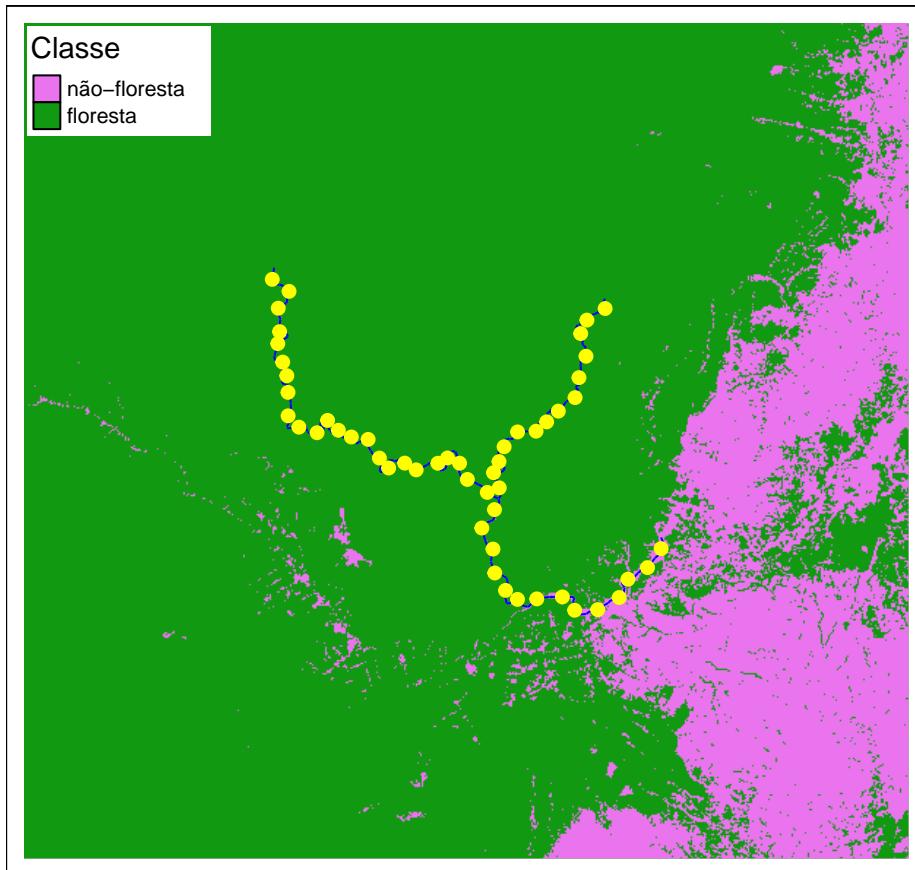


Figura 2.2: Cobertura da terra ao redor do Rio Araguari em 2020. Mostrando os pontos de amostragem (pontos amarelos) cada 5 quilômetros ao longo do rio (linha azul).

2.4 Calculo de métricas

Agora com todos as camadas organizados e verificados podemos calcular as métricas de paisagem. Para ilustrar como rodar as funções e cálculos com landscapemetrics, vamos calcular o percentual de área central na paisagem (`cpland`). Vamos estudar uma classe (floresta), portanto vamos incluir as métricas para nível de classe. No geral, as métricas de paisagem em nível de classe são mais eficazes na definição de processos ecológicos (Tischendorf, L. Can landscape indices predict ecological processes consistently?. *Landscape Ecology* 16, 235–254 (2001). <https://doi.org/10.1023/A:101112719782>). No entanto, por vezes, os índices de manchas podem ser importantes e informativos nas investigações ao nível da paisagem. Por exemplo, muitos vertebrados requerem manchas de habitat maiores do que um tamanho mínimo, por isso seria útil saber também o tamanho de cada mancha na paisagem.

2.4.1 Uso de pontos de amostragem na ecologia da paisagem

Na ecologia da paisagem, os pontos de amostragem são ferramentas cruciais para a coleta de dados e a compreensão dos padrões e processos espaciais que ocorrem em diferentes paisagens. Eles atuam como janelas para o complexo mosaico de ecossistemas, fornecendo informações valiosas sobre fatores como:

- Composição e distribuição de espécies:

Os pontos de amostragem podem ser usados para registrar a presença e abundância de diferentes espécies em locais específicos, permitindo aos pesquisadores mapear padrões de biodiversidade e identificar áreas com alto valor de conservação.

- Características do habitat:

Ao medir várias variáveis ambientais em pontos de amostragem, como cobertura vegetal, umidade do solo e características topográficas, os pesquisadores podem caracterizar os diferentes tipos de habitat dentro de uma paisagem e compreender como eles influenciam a distribuição de espécies e os processos ecológicos.

- Conectividade da paisagem:

Os pontos de amostragem podem ser usados para avaliar a conectividade entre diferentes manchas de habitat, o que é crucial para o movimento e dispersão de espécies. Essas informações podem informar estratégias de conservação e manejo destinadas a manter paisagens saudáveis e funcionais.

- Mudança da paisagem:

A amostragem repetida dos mesmos pontos ao longo do tempo pode fornecer dados valiosos sobre como as paisagens estão mudando. Essas informações podem ser usadas para monitorar os impactos das atividades humanas, perturbações naturais e mudanças climáticas nas comunidades ecológicas.

Os métodos específicos usados para amostragem de pontos em ecologia da paisagem variam dependendo da questão de pesquisa e do tipo de dados coletados. Alguns métodos comuns incluem:

- Amostragem aleatória:

Os pontos são distribuídos aleatoriamente por toda a paisagem para garantir uma representação imparcial de toda a área.

- Amostragem estratificada:

Os pontos são localizados em diferentes tipos de habitat ou outros estratos da paisagem para garantir que todos os componentes da paisagem sejam amostrados adequadamente.

- Amostragem sistemática:

Os pontos são localizados em intervalos regulares ao longo de transectos ou grades dispostas sobre a paisagem. Amostragens sistemáticas são adotados como padrão dentro do [Programa de Pesquisa em Biodiversidade: https://ppbio.inpa.gov.br/](https://ppbio.inpa.gov.br/).

A escolha do método de amostragem e do número de pontos de amostragem necessários dependerá do projeto específico do estudo e do nível de precisão desejado.

Neste capítulo, vamos calcular as métricas de paisagem dentro de um certo buffer em torno de pontos de amostra. Ou seja, “buffers” de diferentes tamanhos serão usado para medir a paisagem circundante aos locais

de amostragem. Usando o pacote `landscapemetrics`, podemos calcular as métricas de paisagem dentro de um certo buffer em torno de pontos de amostra, com a função `sample_lsm()`. Através da função `sample_lsm()` podemos calcular mais de 50 métricas da paisagem, dentro de buffers (raios/distâncias) diferentes.

Aqui vamos calcular as métricas em três exemplos de caso:

1. Ponto único, raio único, métrica única.
2. Ponto único, distâncias variados, métrica única.
3. Ponto único, distâncias variados, métricas variadas.

2.4.2 Ponto único, raio único, métrica única

Métricas de área central (“core area”) são consideradas medidas da qualidade de habitat, uma vez que indica quanto existe realmente de área efetiva de um fragmento/classe, após descontar-se o efeito de borda. Vamos calcular a percentual de área central (“core area”) no entorno de um ponto de amostragem. Isso seria, a percentual de áreas centrais (excluídas as bordas de 30 m) de cada classe em relação à área total da paisagem.

Para a função `sample_lsm()` funcionar, precisamos informar (i) a paisagem (arquivo de raster), (ii) ponto de amostragem (arquivo vector), (iii) raio do buffer desejada, (iv) forma do buffer (círculo ou quadrado) e por final (v) a métrica desejada. Cada opção tem especificações particulares, assim para que a função pode receber dados em formatos diferentes e produzir resultados conforme as necessidades de diversos casos.

Para este primeiro exemplo trabalharemos com apenas um ponto. Fazemos isso selecionando a primeira linha (primeiro ponto) do objeto com os 52 pontos de amostragem (“rio_pontos_31976”) usando colchetes: `[1,]`.

```
# com a paisagem "floresta_2020",
# fazer uma amostragem de um ponto "rio_pontos_31976[1, ]"
# dentro de uma buffer com raio de 1000 metros e forma circular
# e calcular a métrica "cpland"

minha_amostra_1000 <- sample_lsm(landscape = floresta_2020,
                                    y = rio_pontos_31976[1, ],
                                    size = 1000,
                                    shape = "circle",
                                    metric = "cpland",
                                    edge_depth = 1)
```

Depois que executar (“run”), podemos olhar os dados com o código a seguir. Rodando o nome do objeto, como no próximo bloco de código podemos verificar os resultados.

```
minha_amostra_1000
```

Os dados deve ter os valores (coluna “value”) da métrica (coluna “metric”) de cada classe (coluna “class”) conforme a próxima tabela:

layer	level	class	id	metric	value	plot_id	percentage_inside
1	class	0	NA	cpland	66.95102	1	103.4332
1	class	1	NA	cpland	19.59274	1	103.4332

2.4.2.1 Pergunta 2 O modelo mancha-corredor-matriz é frequentemente adotado na ecologia da paisagem. Com base nas aulas teóricas e usando os valores no objeto `minha_amostra_1000` apresentados na tabela acima, identificar qual classe representar a matriz na paisagem. Há alguma informação faltando que limita a sua capacidade de identificar qual classe representar a matriz? Se sim, o que precisa ser adicionado? Justifique as suas respostas de forma clara e concisa.

2.4.3 Ponto único, distâncias variados, métrica única

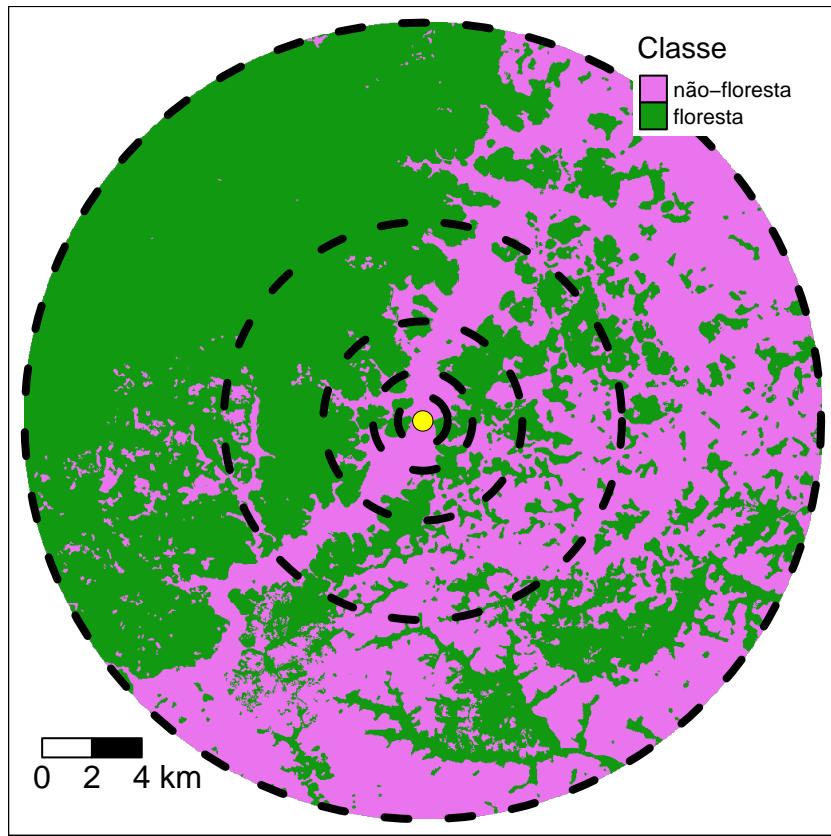


Figura 2.3: Cobertura florestal em extensões diferentes ao redor de um ponto de amostragem.

Para uma comparação multiescala, vamos calcular a mesma métrica, no mesmo ponto, mas agora com extensões diferentes. Continuando o exemplo no tutorial anterior (Escala), vamos repetir o mesmo processo, mas agora com raios de 250, 500, 1000, 2000, 4000, 8000 e 16000 metros, dobrando a escala (extensão) em cada passo.

Para obter resultados com extensões diferentes, precisamos primeiramente repetir o código, ajustando para cada extensão, e depois juntar os resultados. O código a seguir calculará a mesma métrica para as diferentes distâncias. No exemplo, usamos `|>`, que estabelece a ligação entre os passos do processo. Neste caso, usamos a função `mutate()` para incluir uma coluna nova (“raio”) para manter o valor das diferentes distâncias. `mutate()` é uma função muito versátil que pode ser usada para criar, modificar e excluir colunas em um quadro de dados (data frame). Mais exemplos com o uso de `mutate()` aqui: <https://analises-ecologicas.com/cap5#mutate>.

Uso de `mutate()` seguir com a sintaxe: o quadro de dados será sempre o primeiro argumento, seguido de um operador pipe (`|>`) e pelo nome da função `mutate()`. Isso permite o encadeamento de várias operações consecutivas mantendo a estrutura do dado original e acrescentando mudanças num encadeamento lógico.

Sendo assim, `mutate()` não precisa modificar necessariamente os dados originais, sendo que as operações de manipulações podem e devem ser atribuídas a um novo objeto.

```
# raio 250 metros
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 250, shape = "circle",
            metric = "cpland") |>
  mutate(raio = 250) -> minhaAmostra_250

# raio 500 metros
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 500, shape = "circle",
            metric = "cpland") |>
  mutate(raio = 500) -> minhaAmostra_500

# raio 1 km (1000 metros)
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 1000, shape = "circle",
            metric = "cpland") |>
  mutate(raio = 1000) -> minhaAmostra_1000

# raio 2 km
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 2000, shape = "circle",
            metric = "cpland") |>
  mutate(raio = 2000) -> minhaAmostra_2000

# raio 4 km
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 4000, shape = "circle",
            metric = "cpland") |>
  mutate(raio = 4000) -> minhaAmostra_4000

# raio 8 km
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 8000, shape = "circle",
            metric = "cpland") |>
  mutate(raio = 8000) -> minhaAmostra_8000

# raio 16 km
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 16000, shape = "circle",
            metric = "cpland") |>
  mutate(raio = 16000) -> minhaAmostra_16000
```

E agora, o código a seguir juntará os resultados das diferentes extensões. Para combinar as linhas de duas ou mais tabelas de dados podemos utilizar a função `bind_rows()`. Para esta função, precisamos incluir os nomes de todos os dados que queremos combinar, separados por vírgulas. Aqui combinamos as métricas de sete buffers separados e criamos um novo objeto “amostras_metrica”.

```
bind_rows(minha amostra_250,
          minha amostra_500,
          minha amostra_1000,
          minha amostra_2000,
          minha amostra_4000,
          minha amostra_8000,
          minha amostra_16000) -> amostras_metrica
```

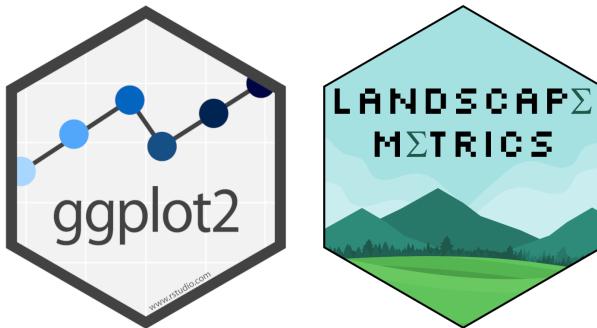
Depois que executar (“run”), podemos olhar os dados “amostras_metrica” com o código a seguir.

```
amostras_metrica
```

Os dados deve ter os valores (coluna value) da métrica (coluna metric) de cada classe (coluna class) para cada distância (coluna raio):

layer	level	class	id	metric	value	plot_id	percentage_inside	raio
1	class	0	NA	cpland	80.3	1	113	250
1	class	0	NA	cpland	86.4	1	106	500
1	class	1	NA	cpland	0.7	1	106	500
1	class	0	NA	cpland	67.0	1	103	1000
1	class	1	NA	cpland	19.6	1	103	1000
1	class	0	NA	cpland	59.1	1	102	2000
1	class	1	NA	cpland	27.9	1	102	2000
1	class	0	NA	cpland	41.9	1	101	4000
1	class	1	NA	cpland	44.2	1	101	4000
1	class	0	NA	cpland	39.6	1	100	8000
1	class	1	NA	cpland	48.5	1	100	8000
1	class	0	NA	cpland	39.2	1	100	16000
1	class	1	NA	cpland	50.7	1	100	16000

2.4.3.1 Faça um gráfico



Uma imagem vale mais que mil palavras. Portanto, gráficos/figuras/imagens são uma das mais importantes formas de comunicar a ciência. Os dados apresentados em uma tabela podem ser difíceis de entender. Portanto, a primeira pergunta que você deve se fazer é se você pode transformar aquela tabela (chata e feia) em algum tipo de gráfico. Lembrando, sempre pode incluir a tabela como anexo.

Aqui, vamos fazer um gráfico com os dados amostras_metricas, usando o pacote [ggplot2](#) para verificar como a paisagem muda com a escala.

O [ggplot2](#) faz parte do conjunto de pacotes [tidyverse](#), e é um pacote de visualização de dados. “gg” se refere a uma gramática de gráficos. A ideia principal é criar um gráfico como se fosse uma frase, onde cada elemento do gráfico seria uma palavra, organizados em uma sequencia lógica para construir uma frase completa (gráfico final). Você fornece os dados, informa ao [ggplot2](#) como mapear variáveis para estética, quais tipos/formatos gráficas usar e ele cuida dos detalhes.

Isto nos permite construir gráficos tão complexos quanto quisermos. Os gráficos criados com [ggplot2](#) são, em geral, mais elegantes do que os gráficos tradicionais do R. Para mais exemplos e tutoriais com mais detalhes veja os capítulos sobre [ggplot2](#) nos livros:

- [Ciência de Dados com R](#)
- [Análises Ecológicas no R](#)
- No livro em inglês [R Graphics Cookbook](#).
- E sempre pode buscar exemplos no Google, por exemplo digitando: ggplot2 gráfico de barra no Google, tem mais de 50 mil resultados com páginas de imagens, código pronto e exemplos no YouTube.

O [ggplot2](#) exige que os dados a serem plotados estejam em um “data frame” ([quadro de dados](#)). Ou seja, sempre teremos que transformar os dados para um data frame ou construir um data frame com os dados que possuímos. Data frame é um formato comum e fácil de trabalhar - parecido com uma planilha de dados com linhas e colunas. Todo que pode fazer com uma planilha de dados pode fazer com um data frame. Por exemplo, se você importar uma planilha de dados, o resultado seria como data frame (para mais detalhes veja [Estrutura e manipulação de objetos](#) e [lendo dados](#)).

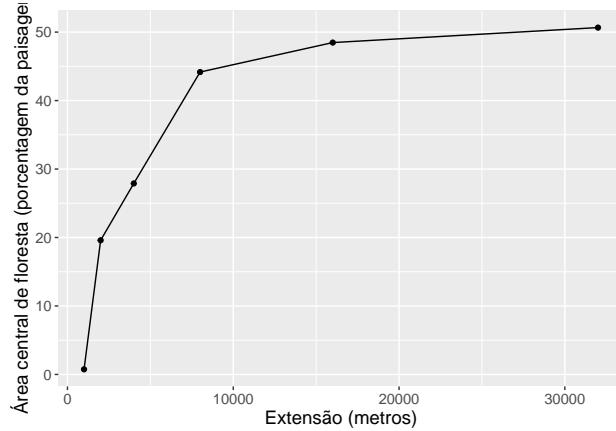
Além disso, o mais importante é que o resultado das funções de [landscapemetrics](#) é sempre um data frame, ou seja os resultados da função `sample_lsm()` são prontos para um gráfico elegante com [ggplot2](#).

O principal função a ser utilizado é `ggplot()`. Para `ggplot()`, precisamos os dados (data frame), e depois cria o “mapeamento” das variáveis, normalmente usando `aes()` (de aesthetics). Ou seja, você especifica quais são as variáveis dos eixos x e y dentro de `aes()`. Através dele vamos definir qual é a variável preditora/explanadora/independente (eixo x) e qual é a variável resposta/dependente (eixo y) em nosso conjunto de dados. Depois da função `ggplot()`, na sequencia no codigo nós especificamos qual tipo de gráfico com um “geom”. Por exemplo, `geom_point()` para plotar pontos, `geom_boxplot()` para um boxplot, etc. Para a lista completa de geoms e todas as outras opções do pacote, visite a página do projeto `ggplot2` <https://ggplot2.tidyverse.org/index.html>.

Aqui vamos fazer um gráfico com valores de extensão no eixo x (calculado a partir da coluna raio) e proporção da floresta central no eixo y (coluna value). Assim sendo, com o código a seguir, vamos informar (i) os dados, selecionando classe de floresta através de um filtro com a função `filter()` e acrescentando com `mutate()` uma coluna nova (“ext_m”) com a extensão em metros, (ii) as colunas para os eixos x e y, (iii) tipo de gráfico (gráfico de pontos - `geom_point()` e gráfico de linha - `geom_line()`), (iv) nomes para os eixos. No exemplo, usamos `|>`, que estabelece a ligação entre os passos do processo, ligando os dados (`amostras_metrica`) e o gráfico `ggplot`. Note que no código a seguir, adicionamos um geom com um “+”. No `ggplot2`, nós criamos gráficos em camadas, e adicionamos camada a camada com um “+”. Assim, é possível ajustar qualquer elemento do gráfico.

```
# arrumar os dados
amostras_metrica |>
  filter(class==1) |>
  mutate(ext_m = 2*raio) |>
# fazer o gráfico
ggplot(aes(x = ext_m, y = value)) +
  geom_point() +
  geom_line() +
  labs(x = "Extensão (metros)",
       y = "Área central de floresta (porcentagem da paisagem)")
```

Depois de executar (“run”) o código acima, você deverá ver o gráfico a seguir.



2.4.3.2 Pergunta 3 Em vez de extensão, você preciso incluir o tamanho (área do círculo) correspondente a cada raio. Incluir uma cópia do código ajustado para produzir uma figura com tamanho (área em quilômetros quadrados) no eixo x.

2.4.3.3 Faça um gráfico elegante Podemos ajustar qualquer elemento do gráfico com ggplot2. Agora, vamos mudar as unidades de metros para quilometros, aumentar o tamanho dos pontos, incluir uma linha reta para ilustrar a tendência geral usando a função `stat_smooth()`, colocar o titulo longo do eixo y em duas linhas, e aumentar o tamanho da fonte para o texto ficar mais claro. O objetivo principal de `stat_smooth()` é ajudar o olho a ver padrões nos dados plotados, ou seja, `stat_smooth()` sobrepõe uma curva ajustada aos dados. Se houver um agrupamento dos dados, curvas separadas poderão ser mostradas para cada grupo por exemplo por cor. Esta função pode traçar curvas de uma variedade de modelos diferentes, incluindo regressão linear (`method = "lm"`).

```
# arrumar os dados
amostras_metrica |>
  filter(class==1) |>
  mutate(ext_m = 2*raio,
        ext_km = (2*raio)/1000) |>
# fazer o gráfico
ggplot(aes(x = ext_km, y = value)) +
  geom_point(size = 4) +
  geom_line() +
  stat_smooth(method = "lm", se = FALSE, color = "green",
             linetype = "dashed") +
  labs(x = "Extensão (quilômetros)",
       y = "Área central de floresta\n(porcentagem da paisagem)") +
  theme(text = element_text(size = 18))

## `geom_smooth()` using formula = 'y ~ x'
```

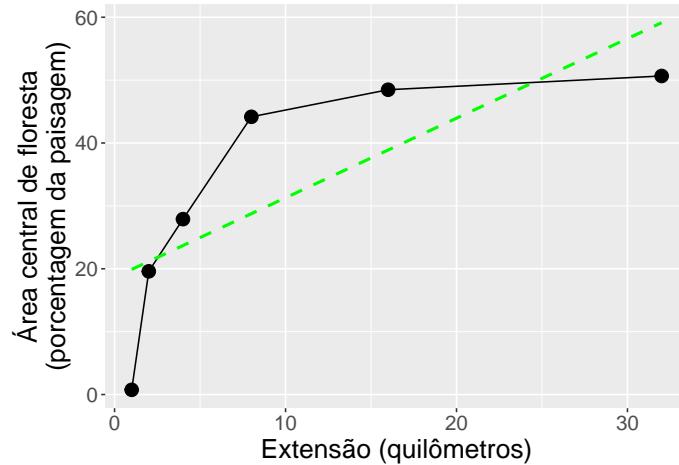


Figura 2.4: Comparação da área central de floresta em diferentes extensões.

2.4.3.4 Pergunta 4 Em menos de 200 palavras apresente a sua interpretação do gráfico em figura 5.2.

2.4.3.5 Modelos linear e não linear Um dos desafios mais frequentes é como melhor representar os dados observados para gerar evidências científicas robustas e informações confiáveis. Nós vimos que as mudanças na métrica porcentagem de área central de floresta não segue uma linha reta em relação de escala (extensão). Para ir além de uma descrição simplista dos padrões observados, na ecologia da paisagem uma variedade de modelos estatísticos são usados. Não vamos rodar modelos (ainda), mas é importante entender algumas das opções disponíveis ao interpretar os gráficos.

Por exemplo, modelos de regressão são amplamente usados em diversas aplicações para descrever a relação entre uma variável resposta Y e uma variável explicativa x. Os modelos lineares são uma generalização dos testes de hipótese clássicos mais simples (para mais detalhes veja [Modelos linear e Modelos lineares](#)). Uma regressão linear só pode ser aplicada para dados em que tanto a variável preditora quanto a resposta são contínuas, enquanto uma análise de variância é utilizada quando a variável preditora/explícata é categórica. Os modelos lineares generalizados não têm essa limitação, podemos usar variáveis contínuas ou categóricas indistintamente (para mais detalhes e exemplos ecológicos veja [Modelos Lineares Generalizados](#)).

Mas, no caso de padroes ecológicas, sera que um modelo linear é o melhor modelo para representar a relação que explica “y” em função de “x”? Um número crescente de pesquisadores compartilham o sentimento de que as relações entre variáveis biológicas/ecológicas são melhores descritas por funções não lineares. Os sistemas ecológicos são inherentemente complexos, com interações multifacetadas entre organismos e seu ambiente. Capturar essas complexidades muitas vezes requer ir além da simplicidade das relações lineares. É aí que os modelos não lineares se tornam ferramentas valiosas na pesquisa ecológica.

Ao contrário dos modelos lineares, que assumem uma relação linear entre as variáveis, os modelos não lineares abraçam a possibilidade de curvas, limiares e outros padrões intrincados. Eles alcançam isso por meio de funções matemáticas que permitem diversas interações entre variáveis. A principal vantagem dos modelos não lineares sobre os lineares é que eles são baseados em conhecimento prévio sobre a relação a ser modelada e geralmente apresentam interpretação prática para os parâmetros. Portanto, modelos não lineares captura relações complexas como:

- Efeitos de limiar:
Mudanças repentinas na resposta além de um ponto crítico, como o colapso das populações de plantas após exceder uma pressão de pastejo específica.
- Loops de feedback:
Interações positivas ou negativas entre variáveis, como ciclos predador-presa ou crescimento populacional exponencial.
- Respostas saturadas:
Diminuição ou nivelamento da resposta à medida que uma variável explicativa aumenta, como a absorção de nutrientes se aproximando de um limite máximo.

Em modelos não lineares, os dados observados de uma variável resposta são descritos por uma função que é não linear em seus parâmetros. Assim como nos modelos lineares, o objetivo é identificar e estabelecer a relação entre as variáveis explicativas e resposta. No entanto, enquanto os modelos lineares geralmente são utilizados para descrever relações empíricas, os modelos não lineares são geralmente motivados pelo conhecimento do tipo de relação entre as variáveis. Isso sendo, os modelos não lineares podem se ajustar melhor e prever padrões complexos nos dados ecológicos, levando a previsões mais precisas da dinâmica populacional, impactos ambientais e estratégias de manejo de recursos.

Como resultado, os modelos não lineares são amplamente utilizados em diversas áreas, como física, biologia, química, fisiologia, etc., onde as relações entre as variáveis são descritas por funções não lineares.

Como as mudanças na estrutura da paisagem caracterizam-se por serem não-lineares, para desenvolver análises estatísticas robustas pode (i) aplicar uma transformação (por exemplo, “log”) ou (ii) adotar modelos não-lineares.

Nesta análise, exploramos os dados `amostras_metrica` através das lentes de três modelos distintos:

1. Modelo Linear (Dados Originais):
Este modelo de linha de base assume uma relação linear entre as variáveis independentes e dependentes

nos dados originais não transformados. Podemos interpretar os coeficientes do modelo para compreender o impacto direto de cada preditor na variável resposta. No entanto, esta abordagem pode não capturar padrões não lineares ou heterocedasticidade (variância desigual) nos dados.

2. Modelo Linear (dados transformados em log):

Reconhecendo a potencial não linearidade, transformamos logaritmo da variável dependente. Isto pode estabilizar a variância e potencialmente linearizar a relação, melhorando o ajuste de um modelo linear. No entanto, a interpretação dos coeficientes torna-se menos simples, exigindo transformação reversa e exponenciação para retornar às unidades originais.

3. Modelo Não Linear (Dados Originais):

Para contabilizar a potencial não linearidade sem transformar os dados, empregamos um modelo não linear, como uma regressão polinomial. Isto permite relações mais flexíveis entre as variáveis mas pode levar a coeficientes menos interpretáveis.

Visualizando o ajuste:

Para comparar a eficácia destes modelos, empregamos ferramentas de diagnóstico visual:

- Valores previstos versus pontos de dados:
sobreponemos os valores previstos (linhas tracejadas) gerados por cada modelo nos pontos de dados originais (pontos pretos). Quanto mais próximas as linhas previstas estiverem dos pontos de dados, melhor o modelo captura as tendências subjacentes.
- Intervalos de confiança:
visualizamos os intervalos de confiança (sombreamento cinza) associados às previsões de cada modelo. Intervalos mais estreitos indicam maior precisão e menor incerteza nas estimativas. A comparação das larguras dos intervalos entre os modelos pode revelar qual deles fornece previsões mais confiáveis.

Podemos mostrar os intervalos de confiança no gráfico com a área central de floresta em diferentes extensões com o seguinte código:

```
# arrumar os dados
amostras_metrica |>
  filter(class==1) |>
  mutate(ext_m = 2*raio,
        ext_km = (2*raio)/1000) |>
# fazer o gráfico
ggplot(aes(x = ext_km, y = value)) +
  geom_point(size = 4) +
  geom_line() +
  stat_smooth(method = "lm",
              se = TRUE,
              color = "green",
              linetype = "dashed") +
  labs(x = "Extensão (quilômetros)",
       y = "Área central de floresta\n(porcentagem da paisagem)") +
  theme_bw() +
  theme(text = element_text(size = 18))

## `geom_smooth()` using formula = 'y ~ x'
```

Para fornecer evidências científicas robustas, precisaríamos ir além de uma avaliação visual subjetiva e adicionar análises estatísticas para apoiar as nossas conclusões. Ao analisar esses recursos visuais juntamente com métricas estatísticas como R-quadrado (por enquanto não precisamos saber os detalhes, apenas que existem técnicas), podemos tirar conclusões informadas sobre qual modelo melhor se ajusta aos dados amostras_metrica. Esta abordagem comparativa proporciona uma compreensão mais profunda das relações dentro dos dados e permite-nos escolher o modelo mais apropriado para análise e previsão subsequentes.

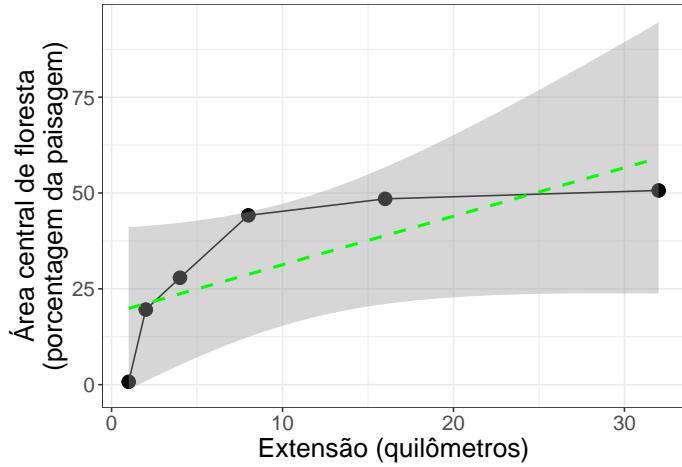


Figura 2.5: Comparação da área central de floresta em diferentes extensões. Mostrando os valores previstos do modelo linear (linha tracejada), pontos de dados originais (pontos pretos) e o intervalo de confiança do modelo (sombreamento cinza).

Vamos comparar os três modelos: modelo linear (dados Originais), modelo Linear (dados transformados em log) e modelo Não Linear (dados Originais).

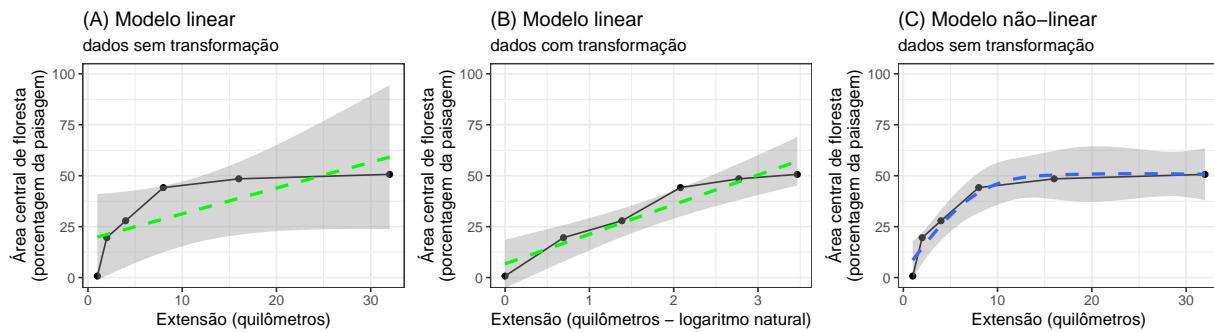


Figura 2.6: Comparação de padrões lineares e não-lineares. Comparamos visualmente o ajuste dos modelos comparando o quanto próximos os valores previstos (linhas tracejadas) estão dos pontos de dados originais (pontos pretos) e as diferenças nos intervalos de confiança do modelo (sombreamento cinza).

2.4.3.6 Pergunta 5 Comparar os resultados apresentados nas figuras com modelos lineares e não-lineares. Como podemos estabelecer qual seria o melhor modelo? Qual modelo seria mais adequado para identificar limiares no padrão de área central de floresta?

2.4.3.7 Pergunta 6 Aumentar o tamanho amostral na análise. Usando os exemplos de código anteriores, aumentar o número de buffers entre 125 m e 16 km. i) Calcular a métrica cpland para 12 distâncias de buffers e montar novos dados (isso é sete distâncias novas e as cinco distâncias originais) em um objeto com nome de "amostras_métrica_nova". ii) usando ggplot2 fazer um gráfico com os novos dados mostrando valores de extensão no eixo x e proporção da floresta central no eixo y. Ajustar o código para incluir intervalos de confiança junto com o modelo linear. Comparar os resultados apresentados nas figuras com modelos lineares sem transformação. O que aconteceu com o ajuste do modelo e os intervalos de confiança? Foi útil incluir mais distâncias? Justifique suas respostas de forma clara e concisa. Inclua cópias do

seu código e gráficos na sua resposta. Você pode usar o printscreen para mostrar o RStudio com seu código e gráficos.

2.4.4 Ponto único, distâncias variados, métricas variadas

No exemplo anterior comparamos uma métrica da paisagem em torno de um único ponto de amostragem. Mas sabemos que uma combinação de várias métricas é necessária para entender os padrões na paisagem. Aqui mostraremos como incluir cálculos de diferentes métricas de paisagem ao mesmo tempo.

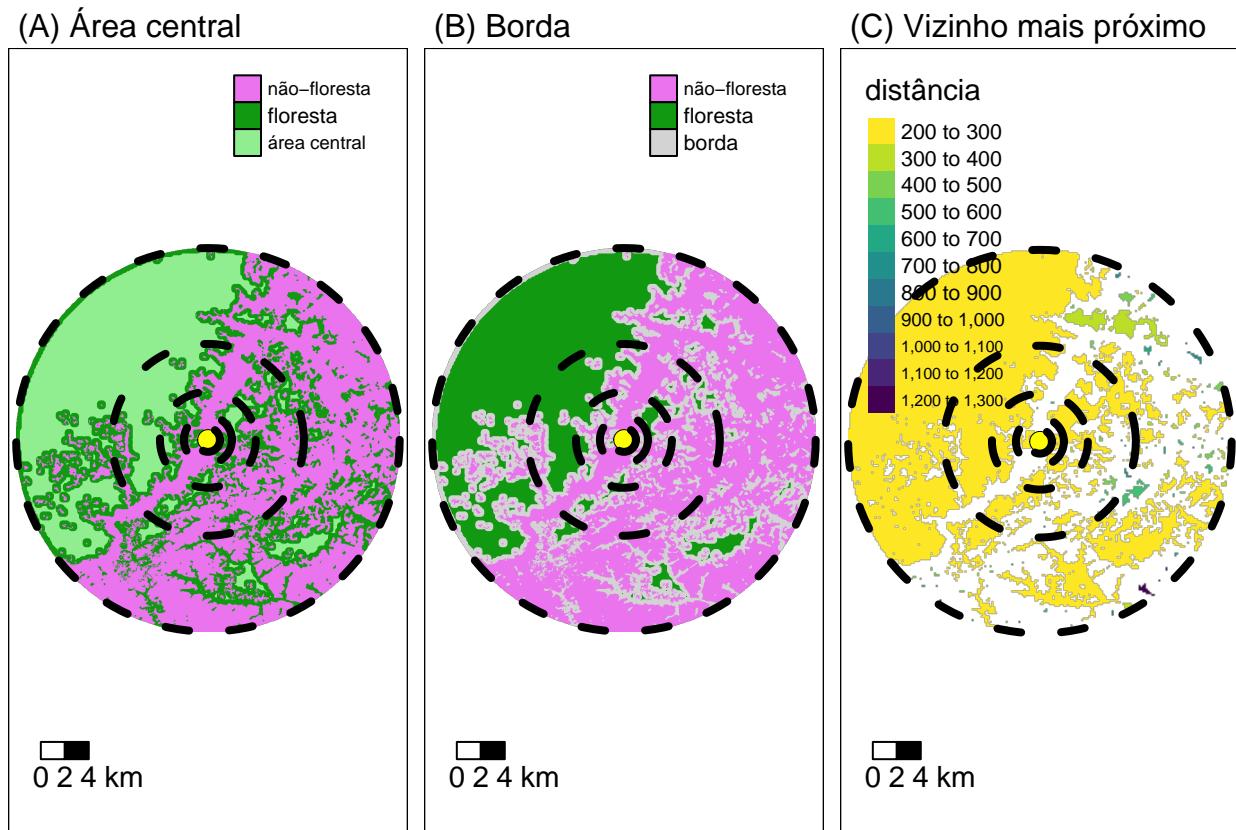


Figura 2.7: Ilustração da determinação de métricas da paisagem diferentes ao redor de um ponto. Exemplo com a estrutura da paisagem representado com três características (A) Área central, (B) Borda e (C) Vizinho mais próximo. O habitat de interesse (classe) é isolado. Um buffer (linha tracejada) é colocado ao redor de um ponto (amarela) e as métricas calculadas. E em seguida o processo é repetido em diferentes extensões.

Não deve calcular todas as métricas disponíveis, mas sim, escolher aquelas que podem ser realmente adequadas para sua pergunta de pesquisa.

Calculando todas as métricas se chama um “tiro no escuro”, algo cujo resultado se desconhece ou é imprevisível. Isso não é recomendado. Para fazer uma escolha melhor (mais robusta), seguindo princípios básicos da ciência, precisamos ler os estudos anteriores (artigos) para obter as métricas mais relevantes para nosso objetivo, pergunta e/ou a hipótese a ser testada.

A métrica mais simples e mais importante é o tamanho da mancha, que representa um atributo fundamental do caráter espacial de uma mancha. A maioria das métricas de paisagem incorpora diretamente informações sobre o tamanho da mancha ou é afetada pelo tamanho da mancha.

Aqui, como exemplo ilustrativa vamos calcular alguns das métricas mais comuns. Mas, isso não representa necessariamente as métricas mais adequados ou recomendadas. Lembre-se que algumas métricas possuem uma combinação de estatísticas (média, coeficiente de variação e desvio padrão). É importante incluir medidas de variação (coeficiente de variação e desvio padrão), bem como a média, para que possamos fornecer evidências científicas robustas. **Nunca apresente uma média sem uma medida da variação.**

- Métricas de área e borda (area and edge metrics). Quantificam a composição da paisagem:
 - `pland` = percentage of landscape. Percentagem da paisagem. Porcentagem de cobertura da classe na paisagem.
 - `ed` = edge density . Densidade de borda que é igual à soma dos comprimentos (m) de todos os segmentos de borda que envolvem o fragmento, dividida pela área total da paisagem (m^2), sendo posteriormente convertido em hectares.
 - `cpland` = core area percentage of landscape. Percentual de área central (“core”) na paisagem. Percentual de áreas centrais (excluídas as bordas de 30 m) em relação à área total da paisagem. O termo “Core area” foi traduzido como área central ou área núcleo. Aqui vamos adotar área central.
- Métricas de agregação. Quantificam a configuração da paisagem:
 - `enn` = euclidian nearest neighbour distance. Distância euclidiana do vizinho mais próximo.
 - `enn_cv` = Coefficient of variation of euclidean nearest-neighbor distance. Coeficiente de variação da distância euclidiana do vizinho mais próximo. A métrica resume cada classe como o Coeficiente de variação das distâncias euclidianas do vizinho mais próximo entre as manchas pertencentes à classe. O valor de `enn_cv` = 0 se a distância euclidiana do vizinho mais próximo for idêntica para todas as manchas. Aumenta, sem limite, à medida que a variação do ENN aumenta.
 - `enn_sd` = Standard deviation of euclidean nearest-neighbor distance. Desvio padrão da distância euclidiana do vizinho mais próximo.
 - `pd` = Patch density. Densidade das manchas.
 - `cohesion` = Cohesion index. Índice de coesão das manchas.

Para incluir cálculos de diferentes métricas de paisagem ao mesmo tempo, precisamos acrescentar somente uma nova linha de código. Uma nova linha, que cria um objeto com os nomes das funções para as métricas que queremos calcular..... Também precisamos usar a opção “what” na função para aceitar os nomes das funções .

```
# Objeto com os nomes das funções para calcular as métricas desejadas.  
# 6 métricas,  
# um (enn) com 3 estatísticas (mn = media,  
# sd = desvio padrão,  
# cv = coeficiente de variação)  
minhas_metricas <- c("lsm_c_pland", "lsm_c_ed", "lsm_c_cpland",  
"lsm_c_enn_mn", "lsm_c_enn_sd", "lsm_c_enn_cv",  
"lsm_c_pd", "lsm_c_cohesion")  
  
# 6 Métricas calculadas para cada extensão
```

```

# raio 250 metros
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 250, shape = "circle",
            what = minhas_metricas) |>
  mutate(raio = 250) -> metricasAmostra_250
# raio 500 metros
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 500, shape = "circle",
            what = minhas_metricas) |>
  mutate(raio = 500) -> metricasAmostra_500
# raio 1 km (1000 metros)
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 1000, shape = "circle",
            what = minhas_metricas) |>
  mutate(raio = 1000) -> metricasAmostra_1000
# raio 2 km (2000 metros)
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 2000, shape = "circle",
            what = minhas_metricas) |>
  mutate(raio = 2000) -> metricasAmostra_2000
# raio 4 km (4000 metros)
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 4000, shape = "circle",
            what = minhas_metricas) |>
  mutate(raio = 4000) -> metricasAmostra_4000
# raio 8 km (8000 metros)
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 8000, shape = "circle",
            what = minhas_metricas) |>
  mutate(raio = 8000) -> metricasAmostra_8000
# raio 16 km (16000 metros)
sample_lsm(floresta_2020, y = rioPontos_31976[1, ],
            size = 16000, shape = "circle",
            what = minhas_metricas) |>
  mutate(raio = 16000) -> metricasAmostra_16000

```

E agora, o código a seguir juntará os resultados das diferentes extensões.

```

bind_rows(metricasAmostra_250,
          metricasAmostra_500,
          metricasAmostra_1000,
          metricasAmostra_2000,
          metricasAmostra_4000,
          metricasAmostra_8000,
          metricasAmostra_16000) -> amostrasMetricas

```

Depois que executar (“run”), podemos olhar os dados com o código a seguir.

```
amostrasMetricas
```

Os dados deve ter os valores (coluna “value”) das métricas (coluna metric) de cada classe (coluna “class”) para cada distância (coluna “raio”):

Agora, vamos fazer um gráfico com os dados amostrasMetricas, usando o pacote ggplot2. Para ajudar na visualização incluímos quais métricas são para composição e configuração, e nomes que são mais fáceis de entender. A função `mutate()` é usado para incluir novas colunas.

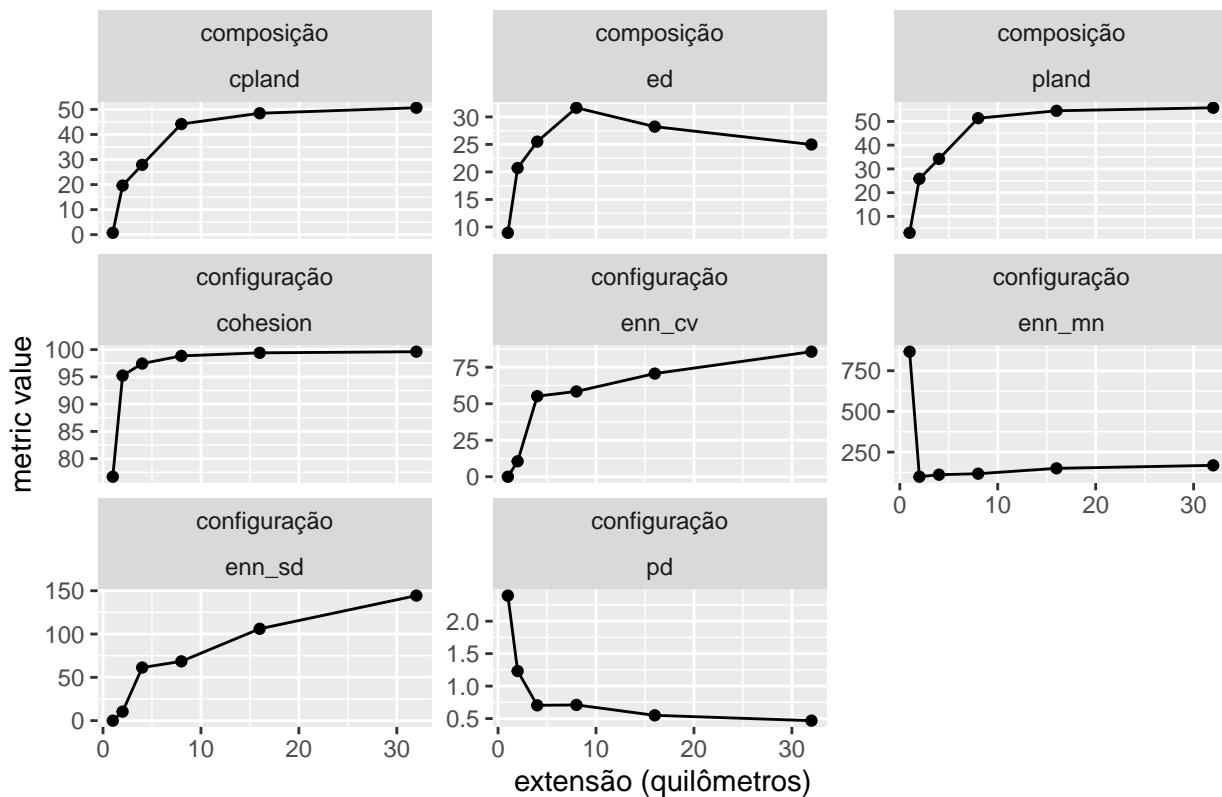
layer	level	class	id	metric	value	plot_id	percentage_inside	raio
1	class	0	NA	cohesion	100.00	1	113.4	250
1	class	0	NA	cpland	80.32	1	113.4	250
1	class	0	NA	ed	0.00	1	113.4	250
1	class	NA	NA	enn_cv	NA	1	113.4	250
1	class	NA	NA	enn_mn	NA	1	113.4	250
1	class	NA	NA	enn_sd	NA	1	113.4	250
1	class	0	NA	pd	4.49	1	113.4	250
1	class	0	NA	pland	100.00	1	113.4	250
1	class	0	NA	cohesion	99.95	1	106.5	500
1	class	1	NA	cohesion	76.69	1	106.5	500

```

metricas_composicao <- c("pland", "ed", "cpland")
# arrumar dados
amostras_metricas |>
  filter(class==1) |>
  mutate(ext_km = (2*raio)/1000,
        met_cat = if_else(metric %in% metricas_composicao,
                           "composição", "configuração")) |>
# fazer gráfico
ggplot(aes(x=ext_km, y=value)) +
  geom_point() +
  geom_line() +
  facet_wrap(met_cat~metric, scales = "free_y") +
  labs(title = "Comparação multiescala de várias métricas",
       x = "extensão (quilômetros)",
       y = "metric value")

```

Comparação multiescala de várias métricas



2.4.4.1 Pergunta 7 Com base nos resultados apresentados (figura e tabela) caracterizar as mudanças na paisagem em função de extensões diferentes. Olhando os gráficos prever como seria o padrão para extensões maiores (lembrando que valores são dobrados - por exemplo raio de 250 metros gerar uma extensão de 500 metros). Seria relevante repetir incluindo cálculos para extensões maiores (por exemplo 64 km e 128 km)? Justifique sua caracterização e previsões de forma clara e concisa, apoie sua escolha com exemplos da literatura científica.

2.4.4.2 Pergunta 8 Usando como base o conteúdo das aulas, leitura disponível no Google Classroom (Base teórica 4 Dados, métricas, análises), e/ou exemplos apresentados aqui no tutorial, selecione pelo menos oito métricas de nível classe para caracterizar a paisagem de estudo e objectivos da sua projeto em grupo. Justifique sua seleção de forma clara e concisa, apoie sua escolha com exemplos da literatura científica.

Essa foi a última pergunta.

Soluções dos exercícios: https://darrennorris.github.io/eprsol/solutions02_metrics.html

2.5 Links úteis

Spatial Data Science with R: <https://rspatial.org/>

McGarigal K., Cushman SA., & Ene, E. 2023.

Spatial Pattern Analysis. Concepts and applications: <https://www.fragstats.org/index.php/documentation>

Wagner, H. 2022.

Landscape Genetic Data Analysis with R: https://bookdown.org/hhwagner1/LandGenCourse_book/ He-

lene Wagner and Max Hesselbarth. Spatial data landscape analysis: [https://bookdown.org/hhwagner1/
LandGenCourse_book/WE_2.html](https://bookdown.org/hhwagner1/LandGenCourse_book/WE_2.html)

Lovelace, R. 2023.
Geocomputation with R: <https://geocompr.robinlovelace.net/index.html>

3 Respostas multiescala de espécies

3.1 Apresentação

Este capítulo pretende ilustrar a abordagem geral de identificação de escalas de efeitos nas relações ambientais usando R. Usaremos conteúdo baseado no Capítulo 2 do livro [Spatial Ecology and Conservation Modeling](#) (Fletcher and Fortin 2018) . Aqui interpretamos a escala em que as espécies podem responder ao habitat, como a cobertura florestal. Para fazer isso, quantificamos a quantidade de floresta que ocorre a várias distâncias dos pontos de amostragem e, em seguida, determinamos a escala em que a cobertura florestal melhor prevê a ocorrência de espécies.

Durante o capítulo você aprenderá:

1. Sobre conceitos de escalas características e escalas de efeito,
2. Quantificar e avaliar as correlações entre diferentes escalas,
3. Determinar a escala em que a floresta melhor prevê a ocorrência de espécies,
4. Compreender a relevância dos conceitos para a interpretação e aplicação de resultados dos estudos de ecologia da paisagem.

Para ajudar a acompanhar e entender os exemplos no capítulo, você deve ler os seguintes artigos:

- da Silva, C. R., et. al. (2013).
Mammals of Amapá State, Eastern Brazilian Amazonia: a revised taxonomic list with comments on species distributions. *Mammalia*, 77(4), 409-424. <https://doi.org/10.1515/mammalia-2012-0121>
- Ferreira, A. S., Peres, C. A., Dodonov, P., & Cassano, C. R. (2020).
Multi-scale mammal responses to agroforestry landscapes in the Brazilian Atlantic Forest: the conservation value of forest and traditional shade plantations. *Agroforestry Systems*, 94, 2331-2341. <https://doi.org/10.1007/s10457-020-00553-y>
- Lyra-Jorge, M. C., Ribeiro, M. C., Ciochetti, G., Tambosi, L. R., & Pivello, V. R. (2010).
Influence of multi-scale landscape structure on the occurrence of carnivorous mammals in a human-modified savanna, Brazil. *European Journal of Wildlife Research*, 56, 359-368. <https://doi.org/10.1007/s10344-009-0324-x>
- Michalski, F., Boulhosa, R. L. P., Nascimento, Y. N. D., & Norris, D. (2020).
Rural wage-earners' attitudes towards diverse wildlife groups differ between tropical ecoregions: implications for forest and savanna conservation in the Brazilian Amazon. *Tropical Conservation Science*, 13, <https://doi.org/10.1177/1940082920971747>
- Michalski, F., & Peres, C. A. (2007).
Disturbance-mediated mammal persistence and abundance-area relationships in Amazonian forest fragments. *Conservation Biology*, 21(6), 1626-1640. <https://doi.org/10.1111/j.1523-1739.2007.00797.x>
- Michalski, F., & Peres, C. A. (2005).
Anthropogenic determinants of primate and carnivore local extinctions in a fragmented forest landscape of southern Amazonia. *Biological Conservation*, 124(3), 383-396. <https://doi.org/10.1016/j.biocon.2005.01.045>

O código de R usado neste capítulo é um tanto aspiracional: se este livro for sua primeira introdução ao R, este capítulo provavelmente será uma desafio. Requer que você tenha ideias internalizadas sobre modelagem, estruturas de dados e iteração. Os exemplos incluem vários links para leituras adicionais que podem ajudar. Então, não se preocupe se você não entender completamente o código de R, mas tentar focar em entendendo mais sobre a aplicação de escalas características e escalas de efeito na Ecologia da Paisagem.

3.2 Escala de efeito - definição

Quantificando a escala de efeito é um problema cada vez mais considerado na ecologia aplicada (Jackson and Fahrig 2015; Dátilo et al. 2023; McGarigal et al. 2016). A escala de efeito refere-se à extensão espacial em que um elemento ou processo da paisagem influencia mais fortemente uma resposta ecológica (Fletcher and Fortin 2018). Em termos mais simples, é o “escala ideal”, onde a relação entre as características da paisagem e os padrões ecológicos é mais significativa. Este conceito é crucial para compreender como a heterogeneidade da paisagem molda os processos ecológicos e informa as estratégias de conservação.

3.2.1 Definições

Vários ecologistas definiram a escala do efeito:

- Martin and Fahrig (2012): “A extensão espacial, ou gama de extensões, dentro da qual uma variável da paisagem tem seu(s) efeito(s) mais forte(s) na resposta de uma determinada espécie.”
- Wu & Li (2006): “As escalas espaciais, temporais e hierárquicas relevantes para processos e fenômenos ecológicos.” No mesmo trabalho Wu and Li (2006) identificou duas conceitos de escala distintas, mas inter-relacionadas: escalas características e escalas de efeito.

1. escalas características

O conceito de escala característica implica que muitos, se não a maioria, dos fenômenos naturais têm suas próprias escalas distintas (ou faixas de escalas) que caracterizam seu comportamento (por exemplo, extensão espacial típica ou frequência do evento). As escalas características são intrínsecas aos fenômenos de interesse, mas as escalas características detectadas com o envolvimento do observador podem ser matizadas com subjetividade. Os padrões e processos ecológicos têm escalas características distintas, nas quais sua dinâmica pode ser estudada de forma mais eficaz. Assim sendo, identificando escalas características fornece uma chave para uma compreensão profunda na Ecologia da Paisagem (Wu 2004).

2. escalas de efeito

As escalas de efeito geralmente se referem às mudanças no resultado de um estudo devido a uma mudança na escala em que o estudo é conduzido. Os efeitos da mudança de escala no desenho experimental, na amostragem, nas análises estatísticas e na modelagem foram bem documentados em ecologia. As escalas de efeito podem ser explicados em termos de multiplicidade de escala, escalas características e hierarquia. Mas escalas de efeito também podem ser artefatos devido a erros na amostragem e medições, distorções na reamostragem de dados e falhas na análise estatística e modelagem (Wu 2004). As escalas características e as escalas de efeito estão inherentemente relacionados à questão da escala. Embora as escalas características forneçam uma base conceitual e diretrizes práticas para o dimensionamento, as descrições quantitativas das escalas de efeito podem levar diretamente a relações de escala (Wu 2004).

Estas definições destacam a natureza dinâmica e dependente do contexto da escala de efeito. Ou seja - o que é mais importante para uma espécie ou processo pode ser irrelevante numa escala diferente ou para outro organismo.

3.2.2 Aplicação

Existem muitas exemplos de uso onde a escala de efeito foi analisado como:

- Ocorrência de espécies de aves:
Estudos que investigam a relação entre a cobertura florestal e a diversidade de aves frequentemente encontram diferentes escalas de efeito para diferentes espécies. Por exemplo, alguns especialistas florestais podem apresentar a resposta mais forte num raio de 1 km de manchas florestais, enquanto os generalistas podem ser influenciados pela cobertura florestal em escalas maiores (por exemplo, 5 km).
- Dispersão das sementes:
A distância percorrida pelas sementes antes da germinação pode variar dependendo do agente dispersor,

variáveis de resposta e preditor de paisagens (San-José et al. 2019). Por exemplo, as sementes dispersas pelo vento podem ter efeitos em escalas maiores, enquanto as sementes dispersas pelas formigas podem ser limitadas a paisagens menores.

- Dinâmica metapopulacional:

A persistência de populações pequenas e fragmentadas pode depender da conectividade de manchas de habitat adequadas dentro de uma determinada escala de paisagem. Compreender a escala do efeito da dispersão entre manchas é crucial para a concepção de estratégias de conservação eficazes.

3.2.3 Como identificar a escala de efeito?

Consideramos como podemos relacionar diferenças de escala (extensão) com a ocorrência de espécies para ajudar a identificar a escala característica (ou escala de efeito) da cobertura florestal na ocorrência de espécies. Existem várias maneiras de quantificar a escala de efeito. Uma abordagem é usar zonas tampão (daqui adiante “buffers”) de diferentes tamanhos para medir a paisagem circundante aos locais de amostragem. Essa abordagem foi popularizada por Pearson (1993) e é amplamente utilizada por outros pesquisadores (Jackson and Fahrig 2015; Galán-Acedo et al. 2018; Moll et al. 2020).

Um método alternativo emprega uma função de suavização ponderada pela distância - Kernels espaciais. Os kernels podem ser usados para ponderar os dados com base na distância do local de amostragem. Isso ajuda a capturar melhor os efeitos da vizinhança, ponderando mais os locais próximos do que os distantes. Chandler e Hepinstall-Cymerman (2016) mostraram como os kernels espaciais podem ser usados para selecionar a escala de efeito sem recorrer a binning a priori (como por exemplo, os buffers de 1 km no capítulo “Métricas”).

Neste capítulo, primeiramente ilustramos o uso de buffers de diferentes distâncias e depois ilustramos o uso de kernels espaciais.

3.3 Pacotes e dados

3.3.1 Pacotes

Carregar pacotes (que deve estar instalado antes):

```
library(mapview)
library(eprdados)
library(tidyverse)
library(tidymodels)
library(GGally)
library(siland)
```

Caso os pacotes não tenham sido instalados, o R vai avisar através um mensagem tipo: **Error in library(nomepacote) there is no package called nomepacote**. Neste caso, para instalá-los consulte o capítulos aqui [Capítulo 4 instalação de pacotes](#) e aqui [Capítulo 4 pacotes](#).

3.3.2 Dados necessários

Aqui usaremos os conjuntos de dados entrevistas e carnivoros. Os dados estão no pacote epardados. Carregar os dados com o código seguinte:

```
entrevistas <- epardados::entrevistas
carnivoros <- epardados::carnivoros
```

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 “Funções para verificação e resumo de dados multidimensionais.” no Capítulo 6.

Exemplos de alguns funções e os resultados são no código seguinte:

```

## tipo de objeto
class(carnivoros)

## Primeiras linhas
head(carnivoros)

## Número de linhas e colunas
nrow(carnivoros)
#> [1] 2544
ncol(carnivoros)
#> [1] 13
dim(carnivoros)
#> [1] 2544   13

## Estrutura dos dados
str(carnivoros)

## Ajudar para dados que vem de uma pacote
?eprdados::carnivoros

```

3.3.2.1 Pergunta 1 Verificar os dados entrevistas e carnivoros, usando funções "class()"; "str()" e ajudar "?". Descreva o conteúdo e compare as diferenças nesses dois conjuntos de dados.

Os dados entrevistas tem a localização de 106 propriedades rurais no Amapá.

Os dados carnivoros consistem em:

- Abundâncias e ausência-presença de três espécies de carnívoros simuladas no entorno de 106 propriedades rurais no Amapá, Brasil (Michalski et al. 2020). A distribuição das espécies foi simulada usando o pacote [virtualspecies](#).
- Uma métrica de paisagem cpland foi calculada no entorno das propriedades seguindo a metodologia no capítulo “Métricas” usando o pacote [landscapemetrics](#).

As três espécies têm respostas contrastantes à cobertura florestal. Os padrões de resposta seguem aqueles conhecidos para espécies de carnívoros (Michalski and Peres 2005, 2007).

Os dados incluem valores de uma métrica de paisagem (cpland) calculada para uma única classe de cobertura do solo - cobertura florestal ([MapBiomas Collection 8](#)). A cobertura florestal foi reclassificada das classes originais e inclui formações florestais e savânicas. A métrica cpland é o percentual da área central da classe em relação à área total da paisagem. A métrica cpland foi calculada em torno de cada propriedade rural em buffers com raios de 0,125, 0,25, 0,5, 1, 2, 4, 8 e 16 km. Para reduzir a influência da qualidade da imagem de satélite, os valores foram calculados como medianas de três anos: o ano de estudo, o ano anterior e o ano posterior.

Para compreender o contexto da paisagem de estudo, examinaremos primeiro a área de estudo e a localização das propriedades rurais. Para visualizar as mudanças na área de estudo (onde foi feito as entrevistas) nos últimos decadas podemos olhar com Google Earth Timelapse (com coordenadas graus decimais 0.50979, -52.12023): <https://earthengine.google.com/timelapse?v=0.50979,-52.12023,7.512,latLng&t=0.49&ps=50&bt=19840101&et=20221231>.

Alem disso, no R podemos olhar os dados através de uma mapa interativa com o pacote [mapview](#) : <https://r-spatial.github.io/mapview/>. [mapview](#) serve para visualização exploratória de dados espaciais — gerando mapas úteis com muito pouco código. Uma captura de tela do mapa padrão é mostrada abaixo, incluindo

uma tabela pop-up que aparece se você clicar em um feijão (ponto, linha etc) e o texto que aparece se você passar o mouse sobre uma camada.

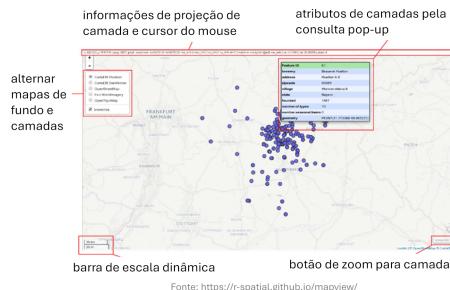


Figura 3.1: mapview: Componentes e funções disponíveis no mapa interativo padrão.

A maneira mais fácil de criar um mapa no `mapview` é usar a função `mapview()` e passar um conjunto de dados para ele. Podemos visualizar os locais das entrevistas com o seguinte código, que tem a mesma estrutura de sempre “pacote::função(dados)”:

```
mapview::mapview(entrevistas)
```

Com o código `mapview::mapview(entrevistas)` você deverá ver um mapa semelhante ao mapa da próxima figura. Com este mapa interativo você pode aproximar diferentes pontos e alterar o mapa base para outros, incluindo OpenStreetMap (com cidades, ruas etc) e ESRI WorldView (com imagens de satélite). Essas funcionalidades permitem ver padrões e heterogeneidade na cobertura da paisagem em diferentes escalas.



3.3.2.2 Pergunta 2 Na aula "Organização de paisagens" vimos que as paisagens contemporâneas são uma resultado de i) variabilidade em condições abióticos; ii) interações bióticas; iii) distúrbios naturais e sucessão; e iv) padrões antrópicos no uso da terra. Usando Google Earth Timelapse e a forma interativa da mapa, verificar a paisagem acerca das propriedades. Com base na sua interpretação das mapas, as aulas, e literatura citado até agora (como Michalski et. al. 2020, Michalski & Peres 2005, Michalski & Peres 2007), apresentar três fatores que poderia explicar a distribuição espacial de espécies de mamíferos na paisagem. Para cada fator inclui uma hipótese válida (e não trivial) junto com uma previsão que pode se testado com um estudo de Ecologia da Paisagem.

3.4 Buffers

Agora investigamos a escala em que as espécies podem responder ao habitat, como a cobertura florestal. Para fazer isso, quantificamos a quantidade de floresta que ocorre em várias distâncias e, em seguida, determinamos a escala em que a cobertura florestal melhor explica a ocorrência das espécies.

3.4.1 Floresta

Primeiramente, olhamos a proporção de cobertura florestal em diferentes escalas (distâncias de buffer) para todos os pontos (locais de 106 entrevistas). Esses dados já foram calculados seguindo os passos no capítulo “Métricas”. Para nos auxiliar à ter uma visão geral do conjunto de dados e de suas interrelações podemos apresentar como um gráfico pareado. Esse gráfico também é chamado de “pairs plot” ou “correlograma”. 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. Mais detalhes sobre a função `ggpairs()` no Capítulo ??.

Ao fazê-lo, descobrimos que a percentagem de cobertura florestal em diferentes escalas tende a ser altamente correlacionada. Isto não é surpreendente, dado que os cálculos com um tamanho de buffer maior incluem a área em tamanhos de buffer menores.

3.4.2 Floresta e a ocorrência de espécies

Para entender como a cobertura florestal afeta a ocorrência de espécies, usamos um modelo de regressão logística. Esse modelo é usado para prever a probabilidade de um evento ocorrer, como a presença de uma espécie. No caso da regressão logística, o evento é a presença de uma espécie (1) ou sua ausência (0).

A regressão logística é um dos métodos estatísticos mais comumente usados em ecologia. É aplicado a uma ampla gama de questões, incluindo:

- Distribuição de espécies e adequação de habitat: Modelagem da probabilidade de uma espécie ocorrer em um habitat específico com base em fatores ambientais.
- Dinâmica populacional: Análise de fatores que afetam o crescimento ou declínio populacional.
- Biologia da conservação: Prevendo o impacto das atividades humanas nas espécies ameaçadas.
- Ecologia comunitária: Investigando as relações entre as espécies e seu ambiente.

A regressão logística funciona de forma semelhante à regressão linear, mas é adaptada para dados binários (0 e 1). Na regressão linear, o modelo prevê um valor contínuo, como o tamanho do corpo de um animal. Na regressão logística, o modelo prevê a probabilidade de um evento ocorrer, como a presença de uma espécie. O modelo de regressão logística que usamos é simples, e não leva em conta todos os fatores que podem afetar a ocorrência de espécies. No entanto, é útil para ilustrar como efeitos de escala na cobertura florestal pode afetar nossa entendimento sobre a ocorrência de espécies.

Compararemos diferentes modelos com base em medições de cobertura florestal em diferentes escalas - extensões locais (distâncias de buffer). Para comparar os modelos com buffers diferentes, podemos usar medidas, como medidas de ajuste (verossimilhança), a variação explicada e/ou o sucesso preditivo. Uma avaliação de modelo completa incorpora vários critérios para garantir que sua regressão logística forneça insights confiáveis sobre seus dados ecológicos.

Primeiramente vamos rodar as análises para uma espécie. Isso seria um exemplo mostrando os passos.

3.4.2.1 terminologia Um detalhe importante sobre a regressão logística é que este modelo se enquadra na classe de modelos lineares generalizados (Generalized Linear Models - GLM). Logo, no R, este modelo pode ser estimado a partir da função `glm`, escolhendo a família binomial no argumento `family`. Para obter

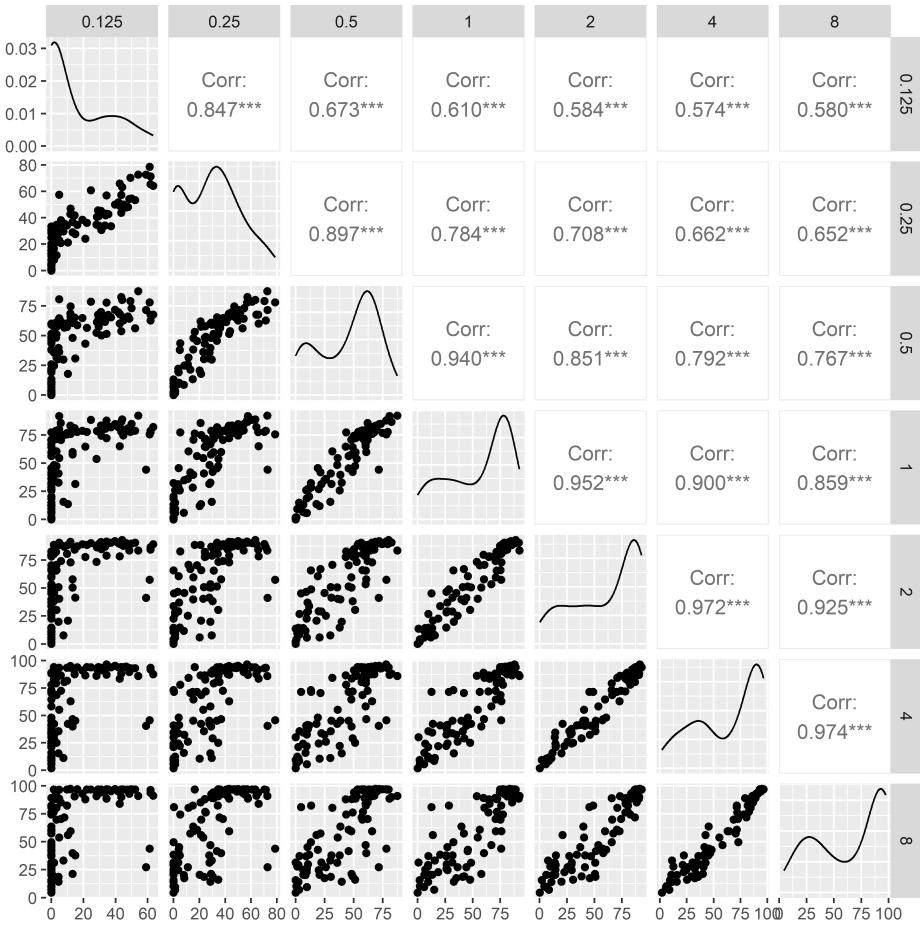


Figura 3.2: Cobertura florestal em torno dos locais de amostragem, calculada em diferentes escalas. São mostrados gráficos de dispersão da porcentagem de cobertura florestal para cada combinação de escalas em pares. Observe o alto grau de correlação entre as escalas similares (125 e 250 metros por exemplo) e escalas maiores (acima de 1 km).

mais detalhes e exemplos mais aprofundados de modelos lineares e modelos lineares generalizados em R, consulte:

- Ciência de Dados com R - Modelos: <http://sillasgonzaga.com/material/cdr/modelos>
- Análises Ecológicas no R - Modelos Lineares Generalizados: <https://analises-ecologicas.com/cap8>

Usando os dados carnivoros, vamos nos referir a:

- A coluna `sp_pa` como variável dependente/resposta, com o valor observado (1 ou 0) denominado respostas.

As outras variáveis são referidas como variáveis independentes, mas classificaremos adicionalmente:

- A coluna `buff_dist_km` como variável de desenho (isto foi controlada pelo experimentador).
- A coluna `value_median` como variável independente/explanatoria; um covariável específica do ponto de amostragem (propriedade rural) e distância de buffer.

A forma estrutural do modelo de regressão logística é:

$$\text{logit}(E[Y_i|X_i]) = \text{logit}(p_i) = \ln\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 X_{1i} + \epsilon$$

Typical notation of the linear regression include:

- Y_i denotes the outcome (or dependent) variable for subject i ; this is a binary variable
- X_{1i} denotes the predictor of interest or the independent variable (X_1) for subject i
- β_0 denotes the Y-intercept when X is zero; this is not informative for logistic regression models
- β_1 denotes the slope or the change in Y with a 1-unit change in X
- p_i denotes the probability of the event occurring
- ϵ denotes the errors

Para comparar o ajuste de modelos diferentes, usamos a log-verossimilhança, que tem um forte base filosófica em estatística. log-verossimilhanças são baseadas no conceito de máxima verossimilhança, que é uma técnica estatística para estimar os parâmetros de um modelo a partir dos dados. Neste estudo, comparamos modelos que possuem o mesmo número de parâmetros. Isso é importante porque permite que os resultados sejam interpretados com mais confiança. Se os modelos tivessem diferentes números de parâmetros, seria difícil dizer se as diferenças nos resultados eram devido às diferenças nas escalas espaciais ou ao número de parâmetros. Assim sendo, neste caso, o uso de alternativos como critérios de seleção de modelos, como o Critério de Informação de Akaike, que penaliza a log-verossimilhança com base no número de parâmetros (Burnham and Anderson 2002), forneceria resultados idênticos.

3.4.2.2 Construção de modelos Todos os modelos estão errados, alguns são úteis (“All models are wrong, some are useful.” - [George E. P. Box : https://en.wikipedia.org/wiki/All_models_are_wrong](https://en.wikipedia.org/wiki/All_models_are_wrong)).

Devemos ter cuidado durante o processo utilizado para construir modelos, assim, não somente para fornecer uma representação robusta e útil do mundo real, mas também para representar fielmente os erros/incertezas associadas. Cada escolha feita mudará a representação dos padrões espaciais e dos processos ecológicos. Portanto, para gerar evidências científicas robustas, todas as etapas devem ser descritas detalhadamente. Um modelo é uma simplificação ou aproximação da realidade e, portanto, não refletirá toda a realidade. Embora um modelo nunca possa ser “verdadeiro”, um modelo pode ser classificado de muito útil, a útil, a algo útil e, finalmente, essencialmente inútil (Burnham and Anderson 2002). Uma avaliação de modelo completa incorpora vários critérios para garantir que sua regressão logística forneça insights confiáveis sobre seus dados ecológicos.

Neste exemplo, analisamos a ocorrência da espécie “sppD” em relação à cobertura florestal. Antes de construir e executar os modelos, é importante executar uma análise exploratória de dados. Aqui usamos boxplots para uma análise exploratória preliminar. Existem muitos tipos de análise exploratória de dados, que variam dependendo dos dados e do tipo de modelo. No caso de modelos de regressão logística, para as variáveis independentes contínuas como proporção de floresta (`value_median`), podemos tentar gráficos de caixa (boxplots) de cada variável independente no eixo “y” com a variável dependente (presença-ausencia) tratada como um fator de agrupamento no eixo “x”.

Os boxplots mostram que, onde a espécie está presente, a cobertura florestal é, em média, mais elevada do que onde a espécie está ausente. No entanto, há uma variação substancial na cobertura florestal nos locais onde a espécie está ausente. Essa variação sugere que, embora a cobertura florestal seja um fator importante que influencia a ocorrência da espécie, outros fatores também podem desempenhar um papel. Por exemplo, a espécie pode estar sujeita a outros fatores de pressão, que podem incluir a caça, limitação de recursos (comida, abrigo, etc.) e/ou a competição com outras espécies.

A previsão da ausência de espécies na ecologia apresenta vários desafios que a tornam mais difícil do que a previsão da sua presença. Aqui estão alguns motivos principais:

1. Detecção imperfeita:

Detectar de forma conclusiva a ausência de uma espécie é muitas vezes mais difícil do que confirmar a sua presença. A falta de detecção pode ser devido a:

- Baixas taxas de encontro: Algumas espécies podem ser naturalmente raras ou secretas, tornando-as difíceis de encontrar mesmo quando estão presentes.

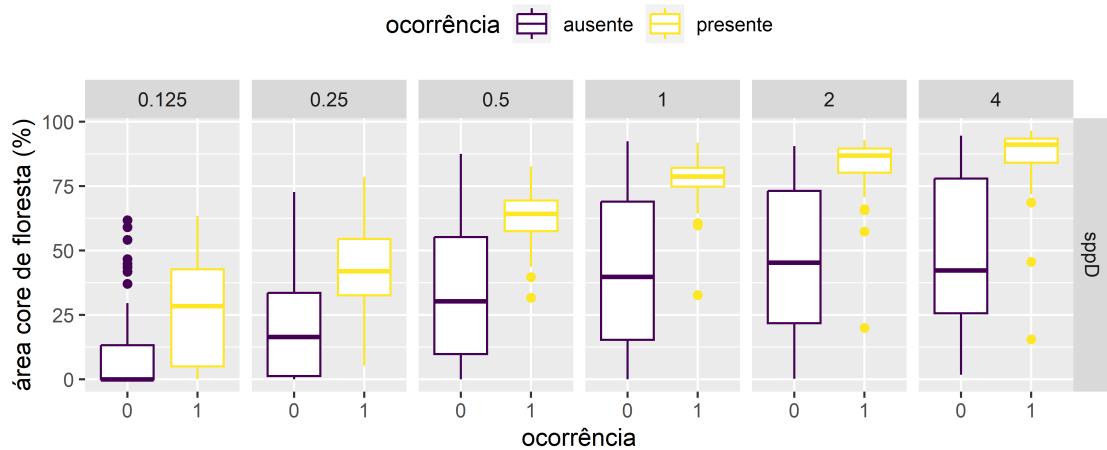


Figura 3.3: Gráfico para explorar as relações entre as variáveis. Boxplot comparando área core de floresta entre a presença-ausença de uma espécie em escalas diferentes (distâncias de 0.125 a 4 km).

- Vieses de detecção: Os métodos de amostragem podem ser tendenciosos em relação a determinadas espécies ou habitats, levando a falsos negativos para outros.
 - Fatores ambientais: As condições ambientais, como o clima ou a cobertura vegetal, podem influenciar a detectabilidade, tornando mais difícil inferir a verdadeira ausência.
2. Nichos ecológicos:
Os nichos ecológicos são complexos e multifacetados. Prever a ausência requer a compreensão de toda a gama de condições e recursos ambientais que uma espécie pode tolerar, o que pode ser um desafio devido a:
 - Conhecimento limitado: A nossa compreensão das necessidades ecológicas de muitas espécies ainda é incompleta, especialmente para táxons menos estudados.
 - Variabilidade espacial e temporal: A adequação do habitat pode variar no espaço e no tempo, tornando difícil prever a ausência de uma espécie em todos os locais potenciais.
 - Preferências de microhabitat: Algumas espécies podem ter preferências específicas de microhabitat em áreas aparentemente adequadas, tornando a sua ausência mais difícil de identificar.
 3. Efeitos de limiar:
As respostas das espécies aos factores ambientais envolvem frequentemente limiares, onde estão presentes acima de um determinado valor, mas ausentes abaixo dele. Identificar esses limites com precisão pode ser difícil, levando a incertezas na previsão de áreas de ausência.

3.4.2.3 Pergunta 3 Identifique a espécie "sppD". Com base nos resultados dos boxplots e os artigos (Michalski & Peres 2005; Michalski & Peres 2007), sugerem três espécies de carnívoros que poderiam apresentar os mesmos padrões do sppD? Que informações/resultados adicionais o ajudariam a identificar a espécie sppD com maior certeza? Justifique suas escolhas de forma clara e concisa.

Os boxplots são úteis para análises exploratórias, mas não fornecem muitas informações para a compreensão da escala do efeito. O modelo de regressão logística é um modelo preditivo para dados binários. Conseqüentemente, o modelo de regressão logística pode gerar probabilidades de que uma amostra terá o resultado discreto dada(s) uma(s) variável(ões) de entrada. O modelo de regressão logística usa estimativa de máxima verossimilhança (Maximum Likelihood Estimation - MLE), que é uma probabilidade condicional que classifica o resultado se um determinado limite for atingido (por exemplo, $> 0,50$). Portanto, o intervalo de probabilidade de um modelo de regressão logística está entre 0 e 1.

Agora plotamos os dados, ajustando uma regressão logística para obter uma representação visual. É útil executar isso (representação visual) antes de executar os modelos, para que tenhamos uma ideia dos padrões que existem nos dados.

Os graficos mostram a mesma padrao que os boxplots, mas com informações adicionais. Como vimos nos boxplots a probabilidade de ocorrência aumenta com a cobertura florestal. Agora podemos ver mais claramente como a forma e a força da relação com a cobertura florestal mudam em diferentes escalas. Por exemplo, observe as diferenças dos valores da cobertura florestal com 50% de probabilidade (linhas horizontais tracejadas).

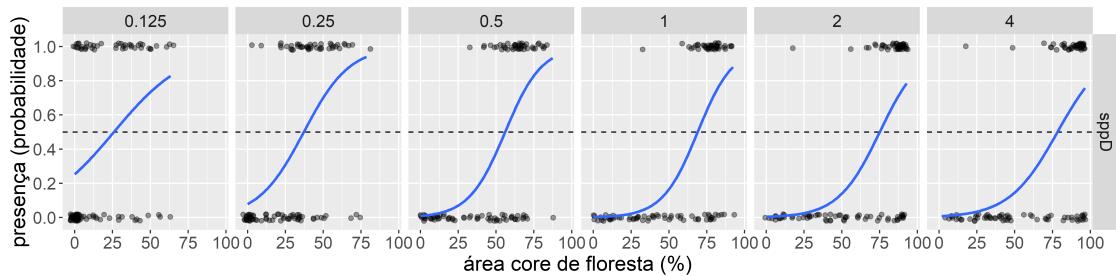


Figura 3.4: Gráfico para explorar as relações entre as variáveis. Comparando a resposta de presença-ausença de uma espécie em escalas diferentes (distâncias de 0.125 a 4 km). Pontos mostram presença-ausência da espécie em 106 propriedades rurais. A linha azul é o ajuste de uma regressão logística.

Uma impressão visual é útil. Mas devemos utilizar modelos estatísticos para fornecer evidências mais robustas e objetivas. Usamos código que executará modelos simultaneamente para todas as distâncias de buffer. Podemos fazer a mesma tarefa com modelos e resultados separadamente para cada distância de buffer. Mas, seria pelo menos 50 linhas de código para rodar os modelos separadamente e obter os resultados que precisamos. Rodando os modelos separadamente seria viável se precisássemos de apenas um ou dois modelos. Mas torna-se muito ineficiente e muito propenso a gerar erros quando precisamos executar vários modelos diferentes, por exemplo, em distâncias diferentes e para espécies diferentes. Aqui usamos funções oriundas do `tidyverse` e `tidymodels` para rodar os modelos e obter os resumos que precisamos em menos de 10 linhas - fazendo múltiplos modelos de regressão simultaneamente no R. Vamos fazer múltiplos modelos simultaneamente usando funções de:

- listas aninhadas `tidy::nest`,
- iteração `purr::map` e
- extração de resultados arrumados `broom::glance` e `broom::tidy`.

Embora possa parecer complicado, este é um processo poderoso que pode ser aplicado a grandes conjuntos de dados e funciona para muitos dos diferentes tipos de modelos que você pode executar em R (incluindo aprendizado de máquina). O processo inclui dois princípios básicos:

1. Usando colunas de lista para armazenar estruturas de dados em um data frame. Por exemplo, isso permitirá que você tenha uma coluna que contenha modelos lineares generalizados como uma regressão logística.
 2. Usando o pacote `broom`, para transformar modelos em dados organizados.
- Essa é uma técnica poderosa para trabalhar com um grande número de modelos porque, depois de ter dados organizados, você poderá aplicar muitas opções para análise e visualizações rapidamente e sem erros.

Em menos de 10 linhas, incluímos funções para obter um conjunto de 8 modelos diferentes (um modelo para cada distância de buffer) e resultados associados dentro do objeto final “modelos_sppD”. Para isso o processo tem quatro etapas conectadas com o operador pipe `|>`. Começando com o banco de dados carnivoros, na sequencia:

1. aplicar um filtro para selecionar o especie “sppD” `dplyr::filter`

2. selecionar as colunas que precisamos no modelos `dplyr::select`
3. divida os dados em partes que serão modeladas separadamente (listas aninhadas para 8 modelos, um para cada distância do buffer) `tidyr::nest`
4. aplique os modelos e salve os resultados `dplyr::mutate`.

A última etapa inclui a criação de três novos itens usando a função `map`:

- fit: o modelo de regressão logística a ser aplicado `glm`
- glanced: resumo do modelo (informações e ajustes dos modelos)
- tidied: resumo das variáveis (estimativas, intervalos de confiança etc)

```
# Modelos para sppD e 8 distâncias de buffer
modelos_sppD <- carnivoros |>
# 1. filtro a especie desejada
  dplyr::filter(sp_name == "sppD") |>
# 2. selecionar as colunas que faz parte dos modelos
  dplyr::select(sp_name, buff_dist_km, sp_pa, value_median) |>
# 3. agrupar para rodar modelos diferentes para especie
# e escala (distância de buffer)
  tidyr::nest(.by = c(sp_name, buff_dist_km)) |>
# 4. especificar modelo e os resumos desejados
  dplyr::mutate(
    fit = purrr::map(data, ~ glm(sp_pa ~ value_median,
                                   data = .x, family = "binomial"))
    ),
  glanced = purrr::map(fit, broom::glance),
  tidied = purrr::map(fit, broom::tidy, conf.int = TRUE, conf.level = 0.95)
)
```

Olhar os resultados. Se tudo correr bem, haverá um novo objeto de dados “modelos_sppD” no ambiente. Existem 8 linhas (um modelo para cada distância do buffer) e 6 colunas. As colunas incluem os dados (“data”) e resumos para cada modelo (“glanced”) e para cada variável em cada modelo (“tidied”)

```
modelos_sppD
```

```
## # A tibble: 8 x 6
##   sp_name buff_dist_km data             fit     glanced      tidied
##   <chr>        <dbl> <list>          <list> <list>       <list>
## 1 sppD         0.125 <tibble [106 x 2]> <glm>  <tibble [1 x 8]> <tibble [2 x 7]>
## 2 sppD         0.25  <tibble [106 x 2]> <glm>  <tibble [1 x 8]> <tibble [2 x 7]>
## 3 sppD         0.5   <tibble [106 x 2]> <glm>  <tibble [1 x 8]> <tibble [2 x 7]>
## 4 sppD         1     <tibble [106 x 2]> <glm>  <tibble [1 x 8]> <tibble [2 x 7]>
## 5 sppD         16    <tibble [106 x 2]> <glm>  <tibble [1 x 8]> <tibble [2 x 7]>
## 6 sppD         2     <tibble [106 x 2]> <glm>  <tibble [1 x 8]> <tibble [2 x 7]>
## 7 sppD         4     <tibble [106 x 2]> <glm>  <tibble [1 x 8]> <tibble [2 x 7]>
## 8 sppD         8     <tibble [106 x 2]> <glm>  <tibble [1 x 8]> <tibble [2 x 7]>
```

Olhar resumos dos modelos. Para verificar o conteúdo de uma coluna com lista aninhada feita com `tidyr::nest`, precisa desembrulhar com `tidyr::unnest`. O próximo código mostrará os valores de resumo para cada modelo. Isso inclui as colunas “logLik” (log-verossimilhança) e “AIC” (Critério de Informação de Akaike) que usaremos para comparar e identifique a escala do efeito.

```
modelos_sppD |>
  tidyr::unnest(glanced)

## # A tibble: 8 x 13
##   sp_name buff_dist_km data      fit null.deviance df.null logLik   AIC   BIC deviance
##   <chr>        <dbl> <list>    <lis>      <dbl>    <int>   <dbl>   <dbl>   <dbl>    <dbl>
```

```

## 1 sppD      0.125 <tibble> <glm>      143.    105  -63.8 132. 137. 128.
## 2 sppD      0.25   <tibble> <glm>      143.    105  -54.6 113. 119. 109.
## 3 sppD      0.5    <tibble> <glm>      143.    105  -47.5 99.1 104. 95.1
## 4 sppD      1     <tibble> <glm>      143.    105  -44.6 93.2 98.5 89.2
## 5 sppD      16    <tibble> <glm>      143.    105  -50.2 104. 110. 100.
## 6 sppD      2     <tibble> <glm>      143.    105  -46.8 97.7 103. 93.7
## 7 sppD      4     <tibble> <glm>      143.    105  -49.1 102. 108. 98.2
## 8 sppD      8     <tibble> <glm>      143.    105  -48.8 102. 107. 97.5
## # i 3 more variables: df.residual <int>, nobs <int>, tidied <list>

```

Olhar resumos das variáveis, ou seja valores que resume as “descobertas” estatísticas do modelo, como as estimativas, erro padrão, intervalos de confiança, e valores de p. A coluna “estimate” (termos beta) tem os valores quantificando a relação entre a ocorrência de espécie sppD e a cobertura florestal.

```

modelos_sppD |>
  tidyrr::unnest(tidied)

## # A tibble: 16 x 12
##   sp_name buff_dist_km data    fit    glanced term      estimate std.error statistic
##   <chr>        <dbl> <list>  <list> <list> <chr>       <dbl>     <dbl>     <dbl>
## 1 sppD         0.125 <tibble> <glm>  <tibble> (Interce~ -1.08      0.285    -3.81
## 2 sppD         0.125 <tibble> <glm>  <tibble> value_me~  0.0419     0.0114    3.68
## 3 sppD         0.25   <tibble> <glm>  <tibble> (Interce~ -2.46      0.514    -4.79
## 4 sppD         0.25   <tibble> <glm>  <tibble> value_me~  0.0662     0.0139    4.76
## 5 sppD         0.5    <tibble> <glm>  <tibble> (Interce~ -4.66      1.03     -4.51
## 6 sppD         0.5    <tibble> <glm>  <tibble> value_me~  0.0832     0.0177    4.70
## 7 sppD         1     <tibble> <glm>  <tibble> (Interce~ -5.89      1.35     -4.36
## 8 sppD         1     <tibble> <glm>  <tibble> value_me~  0.0853     0.0185    4.62
## 9 sppD         16    <tibble> <glm>  <tibble> (Interce~ -4.29      0.873    -4.91
## 10 sppD        16   <tibble> <glm>  <tibble> value_me~  0.0543     0.0106    5.13
## 11 sppD        2     <tibble> <glm>  <tibble> (Interce~ -5.45      1.22     -4.48
## 12 sppD        2     <tibble> <glm>  <tibble> value_me~  0.0726     0.0154    4.72
## 13 sppD        4     <tibble> <glm>  <tibble> (Interce~ -4.88      1.07     -4.58
## 14 sppD        4     <tibble> <glm>  <tibble> value_me~  0.0624     0.0130    4.80
## 15 sppD        8     <tibble> <glm>  <tibble> (Interce~ -4.55      0.958    -4.74
## 16 sppD        8     <tibble> <glm>  <tibble> value_me~  0.0575     0.0115    4.99
## # i 3 more variables: p.value <dbl>, conf.low <dbl>, conf.high <dbl>

```

3.4.2.4 Compare e identifique a escala do efeito Agora vamos usar os resultados em “modelos_sppD” para responder duas perguntas:

1. Qual escala melhor se ajusta aos dados de presença-ausencia de sppD?
2. A relação entre a ocorrência de espécie sppD e a cobertura florestal é a mesma em diferentes escalas?

Qual escala melhor se ajusta aos dados de presença-ausencia de sppD?.

Ajustamos modelos de regressão logística aos dados, usando a cobertura florestal como variável independente. Para identificar a escala do efeito da cobertura florestal na ocorrência de sppD, plotamos as log-verossimilhanças, desvio explicado e AIC de diferentes modelos de regressão logística ajustados aos dados em função da cobertura florestal calculada em diferentes escalas, variando de 0.125 m a 16 km.

- Log-verossimilhança: uma medida da probabilidade dos dados serem observados sob o modelo. A log-verossimilhança mede quanto bem as probabilidades previstas do modelo correspondem aos dados observados. Valores mais altos de log-verossimilhança indicam um melhor ajuste. Interpretação:
 - Não é diretamente interpretável por si só, mas é usado para calcular outras medidas, como desvio e AIC.

- Comparar log-verossimilhanças de diferentes modelos (aninhados ou com o mesmo número de parâmetros) pode indicar qual deles se ajusta melhor aos dados.
- Desvio explicado: uma medida da proporção da variação na ocorrência que é explicada pelo modelo. O desvio explicado é a proporção do desvio total (desvio nulo) que é explicado pelo modelo. Interpretação:
 - Representa a capacidade do modelo de capturar a variabilidade na variável resposta.
 - Os valores variam de 0 a 1, com valores mais altos indicando melhor ajuste.
 - Pode ser interpretado de forma semelhante ao R-quadrado na regressão linear.
 - Não existe um único valor “bom” para o desvio explicado em uma regressão logística em ecologia, pois depende de vários fatores, incluindo o tamanho da amostra e complexidade do modelo:
 - Tamanho da amostra: Amostras maiores: Geralmente, com amostras maiores, você verá valores explicados de desvio mais altos devido ao aumento do poder estatístico para detectar efeitos menores. Um valor acima de 0,5 pode ser considerado bom em amostras grandes. Amostras menores: Valores em torno de 0,2-0,3 podem ser considerados decentes para amostras menores devido às limitações na detecção de efeitos sutis.
 - Complexidade do modelo: Modelos mais simples: com menos variáveis explicativas, você provavelmente verá valores explicados de desvio mais baixos, mesmo que o modelo capture os padrões principais. Valores acima de 0,3 podem ser bons para esses modelos. Modelos mais complexos: Esses modelos tendem a ter valores explicados de desvio mais altos, mas isso não significa necessariamente que sejam melhores. Valores altos também podem indicar ajuste excessivo aos dados específicos. Valores próximos a 0,8 ou acima podem exigir um exame cuidadoso para overfitting.
- AIC: índice de informação de Akaike.
O AIC equilibra a qualidade do ajuste (medida pela Log-verossimilhança) com a complexidade do modelo (número de parâmetros). Penaliza os modelos pela sua complexidade, recompensando modelos mais simples que se ajustam quase tão bem aos dados. Isso significa que um modelo com uma valor de AIC mais baixa é geralmente considerado mais preferível, pois provavelmente representa bem os dados, evitando o sobreajuste. Interpretação:
 - Valores mais baixos de AIC indicam um melhor equilíbrio entre ajuste e parcimônia.
 - Usado para seleção de modelo, sendo geralmente preferido o modelo com o AIC mais baixo.

Esperaríamos que a escala que melhor se ajustasse aos dados fosse a que apresentasse os maiores valores de log-verossimilhança, desvio explicado e valor de AIC mais baixo. Nesse caso, descobrimos que a cobertura florestal dentro de 1 km é a que melhor se ajusta aos dados.

A relação entre a ocorrência de espécie sppD e a cobertura florestal é a mesma em diferentes escalas?

Vimos nos gráficos comparando a resposta de presença-ausença de sppD em escalas diferentes, que existem diferenças no padrão (50% de probabilidade aumenta de 25% para 75% de cobertura florestal de 0,125 para 1 km). Ou seja, em escalas maiores, são necessárias maiores proporções de floresta para que a sppD ocorra. Mas existe algum suporte estatístico para mostrar que as diferenças entre escalas são relevantes?

Para identificar se há diferenças na relação entre a ocorrência de espécie sppD e a cobertura florestal, plotamos os valores estimados (coluna “estimate”) para a variável independente (value_median). Incluímos intervalos de confiança de diferentes modelos de regressão logística ajustados aos dados em função da cobertura florestal calculada em diferentes escalas, variando de 0,125 m a 16 km. Ao representar graficamente os termos beta em diferentes escalas, descobrimos que as relações são quase idênticas (há uma grande sobreposição nos intervalos de confiança) quando a cobertura florestal é medida em 500 metros ou em tamanhos de buffer maiores. Nesta caso, apenas a distâncias menores (125 metros) vemos uma relação relativamente fraca com a cobertura florestal.

Conclusão: Há evidências consistentes de que a ocorrência de sppD aumenta com a cobertura florestal na paisagem (com base na cobertura medida em 1 km e em escalas maiores).

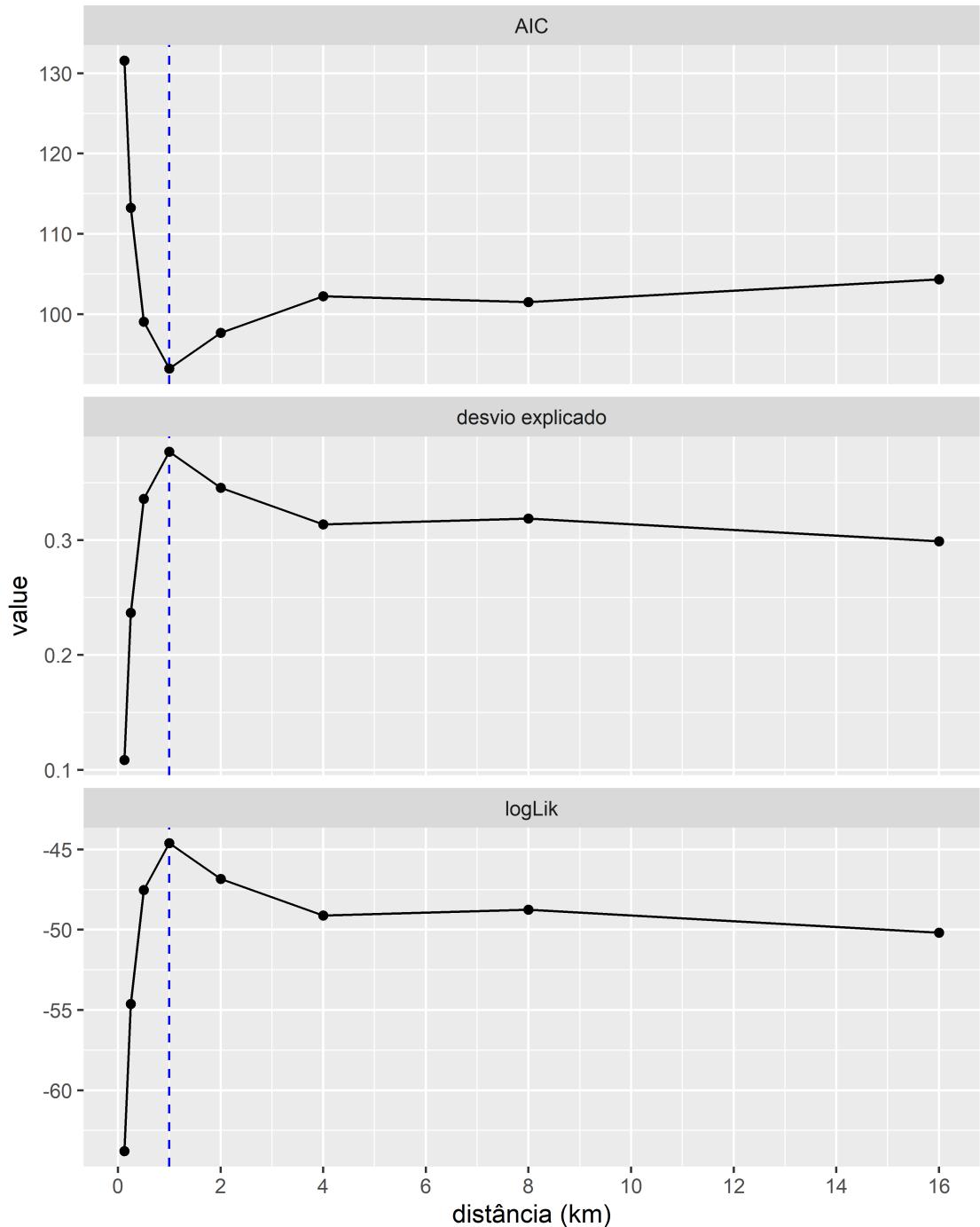


Figura 3.5: a fazer

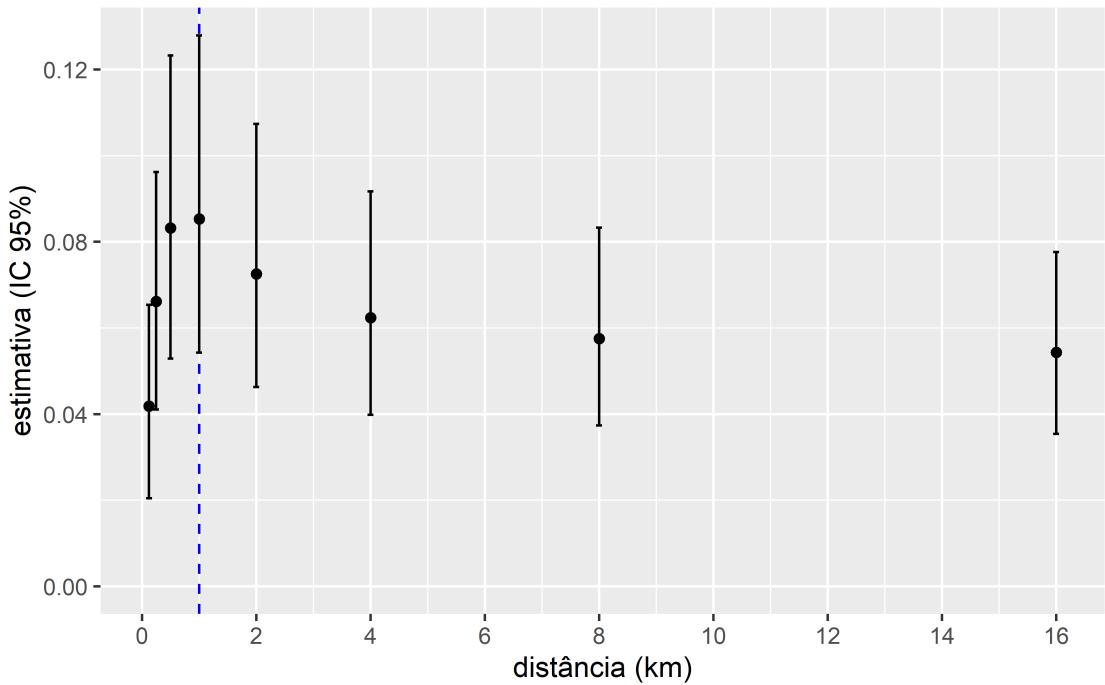


Figura 3.6: As estimativas do parâmetro para o efeito da cobertura florestal (termo beta) na probabilidade de ocorrência de sppD usando buffers de diferentes tamanhos.

3.4.2.5 Pergunta 4 O que sabemos sobre a espécie "sppD"?. Com base nos resultados obtidos: boxplots, graficos com a relação entre occorencia e proporção de floresta, resultados sobre escala do efeito e os artigos (Michalski & Peres 2005; Michalski & Peres 2007), forneça uma caracterização da espécie (por exemplo, especialista/generalista). Justifique a sua caracterização e interpretações com resultados específicos, destacando os resultados com maior relevância. Que informações/resultados adicionais o ajudariam a caracterizar a espécie sppD com maior certeza? Justifique suas escolhas de forma clara e concisa.

3.4.2.6 Múltiplas espécies Vamos agora repetir o processo para todas as três espécies nos dados carnívoros. Indo na mesma sequência: boxplots, gráficos de regressão logística e, em seguida, escala de efeito pela plotagem de resumos de modelos de regressão logística e termos beta.

Primeiramente os boxplots:

Agora plotamos os dados, ajustando uma regressão logística para obter uma representação visual. É útil executar isso (representação visual) antes de executar os modelos, para que tenhamos uma ideia dos padrões que existem nos dados.

```
carnivoros |>
  ggplot(aes(x=value_median, y = sp_pa)) +
    geom_jitter(width=2, height=0.05, alpha=0.4) +
    geom_hline(yintercept = 0.5, linetype="dashed") +
    geom_smooth(method="glm",
                method.args = list(family = "binomial"),
                se = FALSE) +
    facet_grid(sp_name~buff_dist_km) +
```

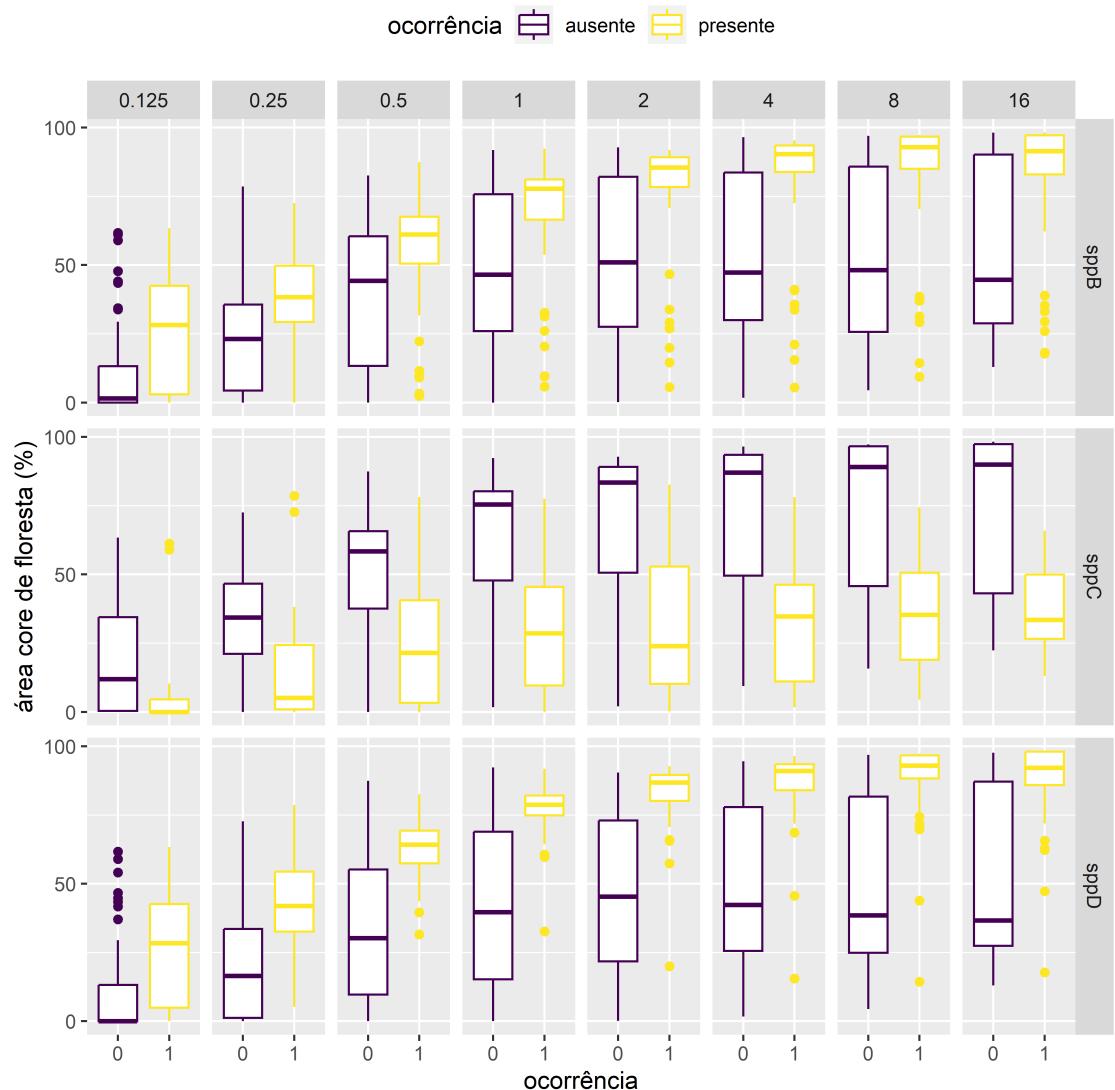


Figura 3.7: Boxplot comparando área core de floresta entre a presença-ausença de espécies em escalas diferentes (distâncias de 0.125 a 16 km).

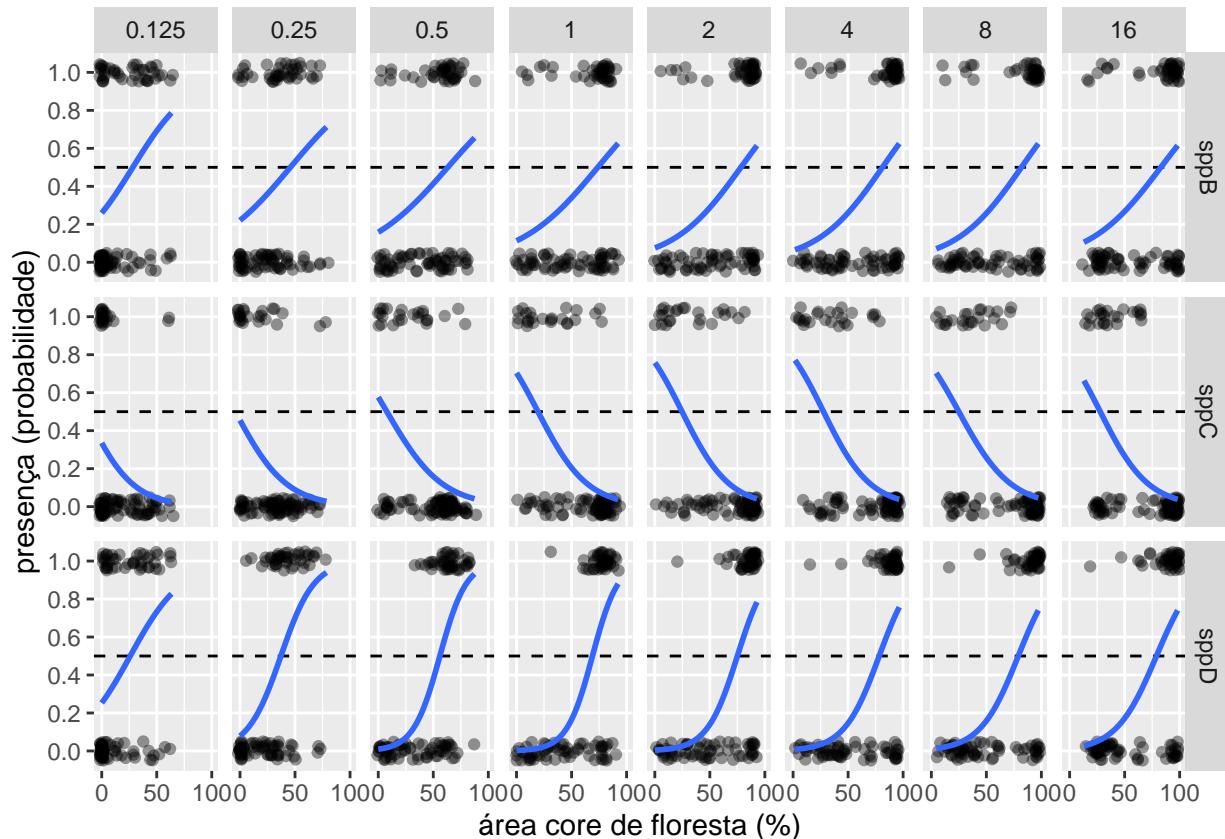
```

scale_x_continuous(breaks = c(0,50, 100)) +
scale_y_continuous("presença (probabilidade)",
c(0,1), breaks = c(0,0.2,0.4,0.6,0.8,1.0)) +
labs(x = "área core de floresta (%)") -> fig_spp

```

fig_spp

```
## `geom_smooth()` using formula = 'y ~ x'
```



Para identificar a escala do efeito da cobertura florestal na ocorrência de espécies, plotamos as log-verossimilhanças, desvio explicado e AIC de diferentes modelos de regressão logística ajustados aos dados em função da cobertura florestal calculada em diferentes escalas, variando de 0.125 m a 16 km. Esperaríamos que a escala que melhor se ajustasse aos dados fosse a que apresentasse os maiores valores de log-verossimilhança, desvio explicado e valor de AIC mais baixo:

Para identificar se há diferenças na relação entre a ocorrência de espécies e a cobertura florestal, plotamos os valores estimados (coluna “estimate”) para a variável independente (value_median). Incluímos intervalos de confiança de diferentes modelos de regressão logística ajustados aos dados em função da cobertura florestal calculada em diferentes escalas, variando de 0.125 m a 16 km:

3.4.2.7 Pergunta 5 O que sabemos sobre as três espécies de carnívoros? Com base nos resultados obtidos: boxplots, graficos com a relação entre occorencia e proporção de floresta, resultados sobre escala do efeito e os artigos (Michalski & Peres 2005; Michalski & Peres 2007), forneça uma identificação e caracterização de cada espécie (por exemplo, especialista/generalista/sensível/resiliente). Justifique a sua identificação, caracterização e interpretações com resultados específicos, destacando os resultados com maior relevância. Que

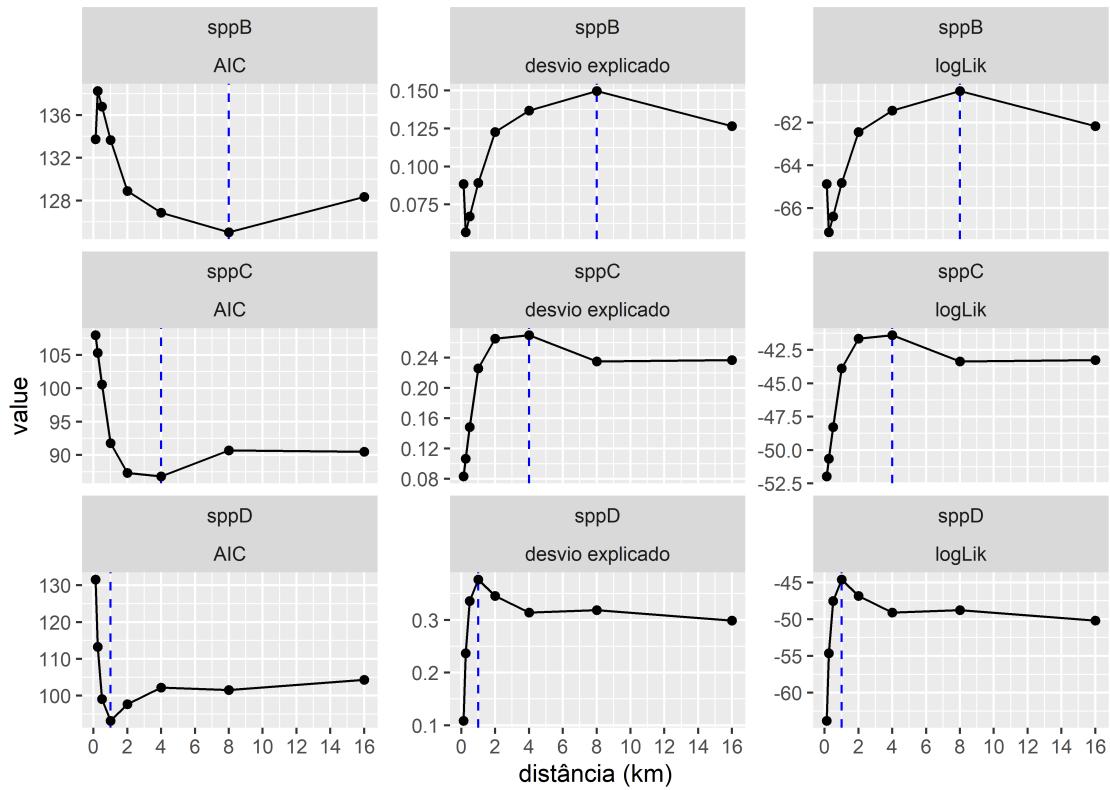


Figura 3.8: a fazer

informações/resultados adicionais o ajudariam a caracterizar as espécies com maior certeza? Justifique suas escolhas de forma clara e concisa.

3.4.2.8 Pergunta 6 Vimos como a incerteza quanto à escala do efeito é provavelmente, pelo menos em parte, uma consequência da correlação entre escalas. Também é possível que, além da cobertura florestal, existam outros fatores que afetem a ocorrência das três espécies na paisagem estudada. Usando como base o conteúdo das aulas, leitura disponível no Google Classroom (Base teórica 4 Dados, métricas, análises), e/ou exemplos apresentados aqui no capítulo, apresentam algumas hipóteses alternativas que poderiam explicar a incerteza nos modelos para as três espécies. Forneça recomendações para variáveis adicionais que poderiam ser incluídas nos modelos para testar suas hipóteses. Justifique sua seleção de forma clara e concisa, apoie sua escolha com exemplos da literatura científica.

Essa foi a última pergunta. O que se segue é um exemplo que mostra uma abordagem alternativa.

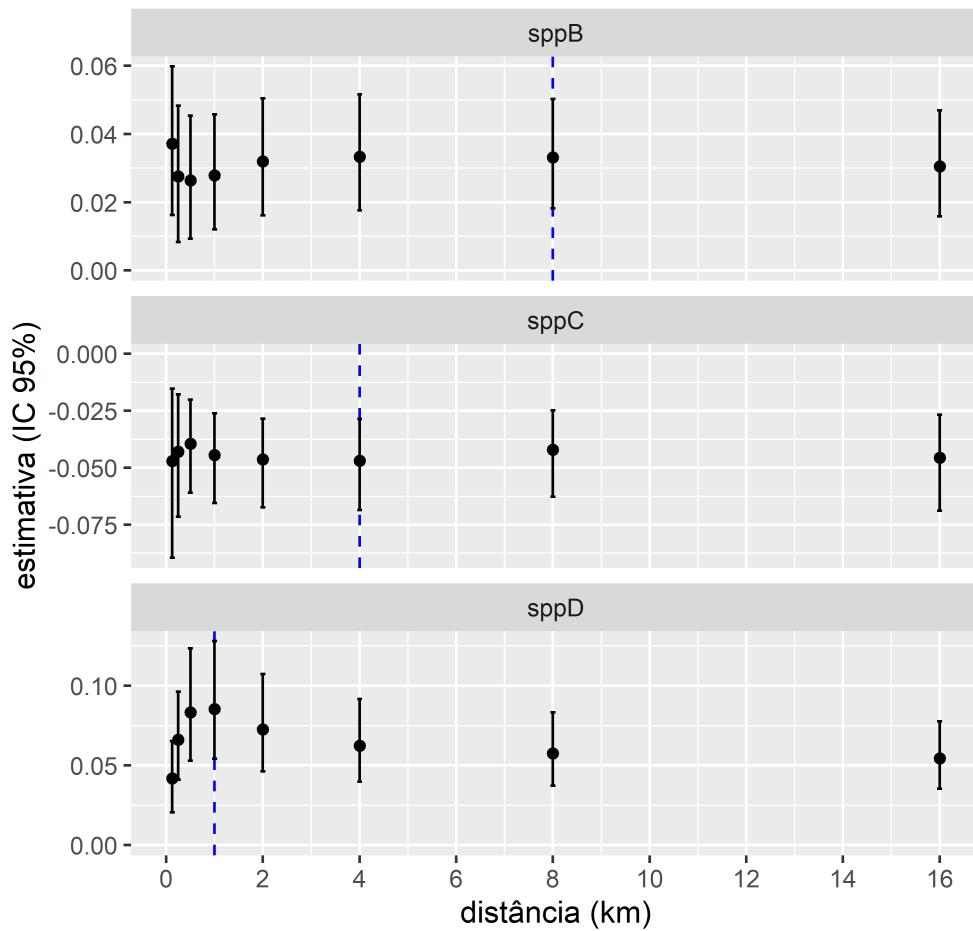


Figura 3.9: Estimativas para o efeito da cobertura florestal (termo beta) na ocorrência de três espécies usando buffers de diferentes tamanhos.

3.5 Kernels

Nesta seção usaremos funções no pacote `siland` para executar uma análise de escala de efeito usando Kernels espaciais.

A abordagem de Kernels fornece uma maneira rigorosa de estimar as escalas nas quais as variáveis da paisagem afetam respostas ecológicas, e pode ser incorporado na maioria das classes de modelos estatísticos (Chandler and Hepinstall-Cymerman 2016). Esta abordagem pode reduzir o viés do modelo e levar a interpretações mais precisas das relações espécie-paisagem (Aue et al. 2012; Chandler and Hepinstall-Cymerman 2016; Miguet, Fahrig, and Lavigne 2017). A função de suavização Kernel assume que o efeito de uma variável da paisagem é mais forte perto do local focal e decai em função da distância. Assim fornecendo uma representação mais realista de muitos processos ecológicos como por exemplo, dispersão de sementes (Bullock et al. 2017) e movimentos de forrageadoras de lugar central (Orians e Pearson 1979).

3.6 Links úteis

Software R curso avançado: <https://smolski.github.io/livroavancado/>

Capítulo 7 Regressão Logística: <https://smolski.github.io/livroavancado/reglog.html>

Análises Ecológicas no R Modelos Lineares Generalizados: <https://analises-ecologicas.com/cap8>

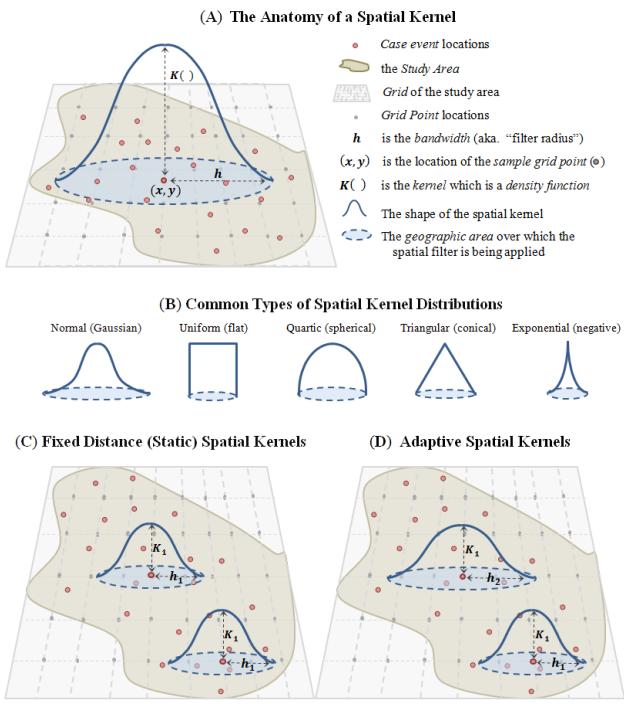


Figura 3.10: Esquema ilustrativo demonstrando diferentes tipos de kernels espaciais: (A) diagrama representando a estrutura geral de um filtro espacial baseado em kernel; (B) os cinco tipos mais comuns de distribuições espaciais de kernel; (C) núcleos espaciais de largura de banda de distância fixa (estática); e (D) núcleos espaciais de distância adaptativa. Observe que a figura é baseada nas seguintes fontes: Figura 3.4 na página 86 de Bailey e Gatrell (1995); Fotheringham et al. (2002) Figuras 2.11 e 2.13 nas páginas 45 e 47 respectivamente; Figura 3.2 na página 37 de Wang (2006); páginas 67-68 de Smith e Bruce (2008); e Figura 4-47 na página 177 de de Smith et al. (2009).

Estatística com R: <https://storopoli.io/Estatistica/>

Regressão Logística : https://storopoli.io/Estatistica/7-Regressao_Logistica.html

Tidy Modeling with R: <https://www.tmwr.org/>

models: <https://www.tmwr.org/models> workflow: <https://www.tmwr.org/workflows>

R for Data Science Many Models: <https://r4ds.had.co.nz/many-models.html>

Astonishingly easy mapping in R with mapview: <https://www.infoworld.com/article/3644848/astonishingly-easy-mapping-in-r-with-mapview.html>

Video - Fazer múltiplos modelos de regressão simultaneamente no R: https://www.youtube.com/watch?v=Wukf7v3_u4I

Part III

Exemplos de caso

4 Garimpo do Lourenço

4.1 Apresentação

Mudanças na paisagem ao redor do Garimpo do Lourenço. Changes in the landscape surrounding the Lourenço gold mine.

Código de R e dados para calcular métricas de paisagem associadas com a exploração de recursos minerários.

O objetivo é calcular métricas de paisagem e descrever a composição e a configuração da paisagem no entorno do Garimpo do Lourenço.

As métricas de paisagem são a forma que os ecólogos de paisagem usam para descrever os padrões espaciais de paisagens para depois avaliar a influência destes padrões espaciais nos padrões e processos ecológicos.

Nesta exemplo (<https://rpubs.com/darren75/lourenco>) aprenderemos sobre como analisar a cobertura da terra com métricas de paisagem em R.

Este exemplo tem como base teórica o modelo “mancha-corredor-matriz” - uma representação da paisagem em manchas de habitat (fragmentos).

4.2 Pacotes necessarios:

```
library(tidyverse)
library(readxl)
library(terra)
library(sf)
library(landscapemetrics)
library(mapview)
library(knitr)
library(gridExtra)
```

4.3 Área de estudo

Para alcançar o objetivo de caracterizar a paisagem no entorno do Garimpo do Lourenço, precisamos estabelecer a extensão da área de estudo. Isso seria estabelicida com base nos objetivos e estudos anteriores. Sabemos que atividades associados com a mineração pode aumentar a perda da floresta até 70 km além dos limites do processo de mineração: Sonter et. al. 2017. Mining drives extensive deforestation in the Brazilian Amazon <https://www.nature.com/articles/s41467-017-00557-w>

Para visualizar um exemplo com a Extração de bauxita na Flona Saracá-Taquera: <https://earthengine.google.com/timelapse/#v=-1.70085,-56.45017,8.939,latLng&t=2.70>

E aqui com o Garimpo do Lourenço: <https://earthengine.google.com/timelapse#v=2.2994,-51.68423,11.382,latLng&t=0.03>

4.4 Dados

4.4.1 Ponto de referência (EPSG: 4326)

Aqui vamos incluir um raio de 20 km além do ponto de acesso para o Garimpo do Lourenço em 1985. Isso representa uma área quadrada de 40 x 40 km (1600 km²).

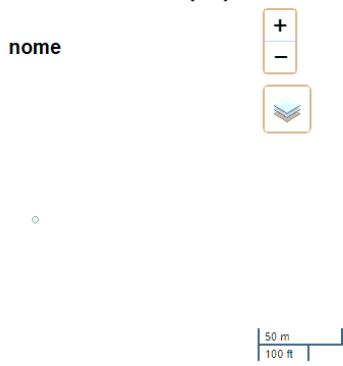
```
# Tabela de dados com coordenados de acesso em 1985.
acesso <- data.frame(nome = "garimpo do Lourenço",
                      coord_x = -51.630871,
                      coord_y = 2.318514)

# Converter para objeto espacial, com sistema de coordenados geográfica.
sf_acesso <- st_as_sf(acesso,
                      coords = c("coord_x", "coord_y"),
                      crs = 4326)
```

Visualizar para verificar.

```
# 
plot(sf_acesso) # teste basica
mapview(sf_acesso) #verificar com mapa de base (OpenStreetMap)
```

(A) basica



(B) com mapa de base



4.4.2 Ponto de referência (EPSG: 31976)

As análises da paisagem com o modelo “mancha-corredor-matriz” depende de uma classificação categórica. Portanto, deve optar para uma sistema de coordenados projetados, com pixels de área igual e com unidade em metros. Temos um raio de 20 km, que é um area geográfica onde o retângulo envolvente é menor que um fuso UTM. Assim sendo, vamos adotar a sistema de coordenados projetados de datum SIRGAS 2000, especificamente EPSG:31976 (SIRGAS 2000/UTM zone 22N).

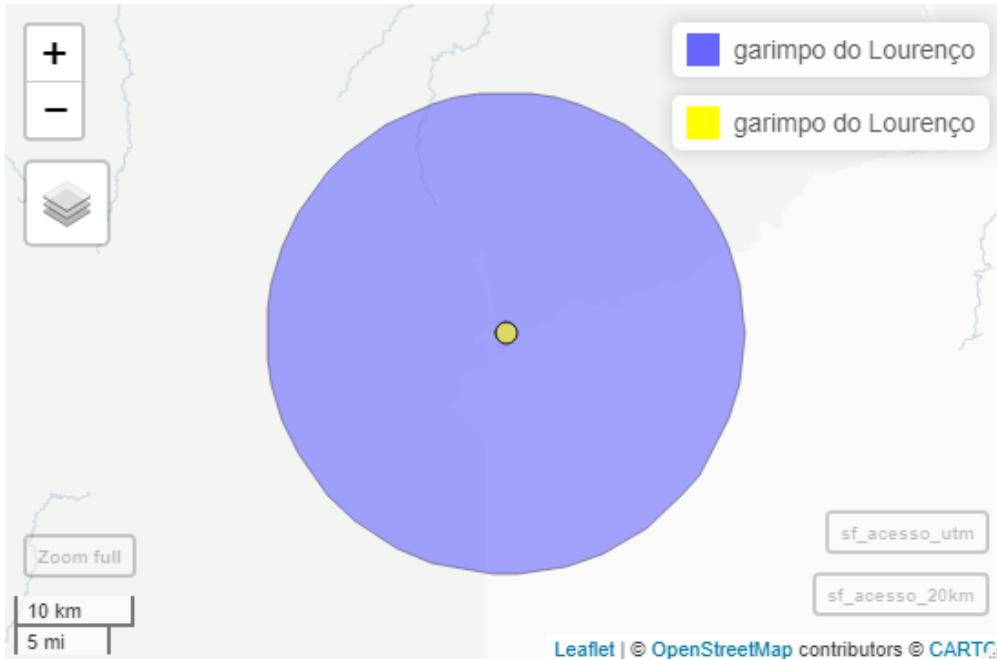
Precisamos então reprojetar o objeto original (em coordenadas geográficas) para a sistema de coordenados projetados. Em seguida, vamos produzir um polígono com raio de 20 km no entorno do ponto.

```
# Reprojetar o ponto.
sf_acesso_utm <- st_transform(sf_acesso, crs = 31976)
# Polígono com raio de 500 metros no entorno do ponto.
sf_acesso_500m <- st_buffer(sf_acesso_utm, dist=500) %>%
  mutate(raio_km = 0.5)
# Polígono com raio de 1 km no entorno do ponto.
sf_acesso_1km <- st_buffer(sf_acesso_utm, dist=1000) %>%
  mutate(raio_km = 1)
# Polígono com raio de 2 km no entorno do ponto.
sf_acesso_2km <- st_buffer(sf_acesso_utm, dist=2000) %>%
  mutate(raio_km = 2)
# Polígono com raio de 4 km no entorno do ponto.
sf_acesso_4km <- st_buffer(sf_acesso_utm, dist=4000) %>%
  mutate(raio_km = 4)
# Polígono com raio de 20 km no entorno do ponto.
sf_acesso_20km <- st_buffer(sf_acesso_utm, dist=20000)

acesso_buffers <- bind_rows(sf_acesso_500m, sf_acesso_1km,
                           sf_acesso_2km, sf_acesso_4km)
```

4.4.3 Verificar com mapa de base (OpenStreetMap).

```
# 
mapview(sf_acesso_20km) +
  mapview(sf_acesso_utm, color = "black", col.regions = "yellow")
```



4.4.4 Dados: MapBiomas cobertura da terra

Agora vamos olhar cobertura e uso da terra no espaço que preciso (área de estudo). Para isso, vamos utilizar um arquivo de raster do projeto [MapBiomas](#) com cobertura de terra ao redor do Garimpo do Lourenço em 1985. Este arquivo no formato raster, tem apenas valores inteiros, em que cada célula/pixel representa uma área considerada homogênea, como uso do solo ou tipo de vegetação. Arquivo “.tif” disponível aqui: [utm_cover_AP_lorenco_1985.tif](#)

Não vamos construir mapas, portanto os cores nas visualizações não corresponde ao mundo real (por exemplo, verde não é floresta). Para visualizar em QGIS preciso baixar um arquivo com a legenda e cores para Coleção⁶ (<https://mapbiomas.org/codigos-de-legenda>) e segue tutoriais: <https://www.youtube.com/watch?v=WtyotodHK8E>.

Este vez, a entrada de dados espaciais seria através a importação de um raster (arquivo de .tif). Lembre-se, para facilitar, os arquivos deve ficar no mesmo diretório do seu código (verifique com `getwd()`). Como nós já sabemos a sistema de coordenadas desejadas, o geoprocessamento da raster foi concluído antes de começar com as análises da paisagem.

```
r1985 <- rast("utm_cover_AP_lorenco_1985.tif")
r1985

#class      : SpatRaster
#dimensions : 1341, 1341, 1  (nrow, ncol, nlyr)
#resolution : 29.87713, 29.87713  (x, y)
#extent     : 409829.5, 449894.7, 236241.1, 276306.3  (xmin, xmax, ymin, ymax)
#coord. ref.: SIRGAS 2000 / UTM zone 22N (EPSG:31976)
#source     : utm_cover_AP_lorenco_1985.tif
#name       : classification_1985
#min value  : 
#max value  : 
```

Ou use o função `file.choose()`, que faz a busca para arquivos.

```
r1985 <- rast(file.choose())
r1985
```

```

##           used   (Mb) gc trigger   (Mb) max used   (Mb)
## Ncells 7981538 426.3 14900711 795.8 14900711 795.8
## Vcells 86626336 661.0 211545401 1614.0 211545401 1614.0

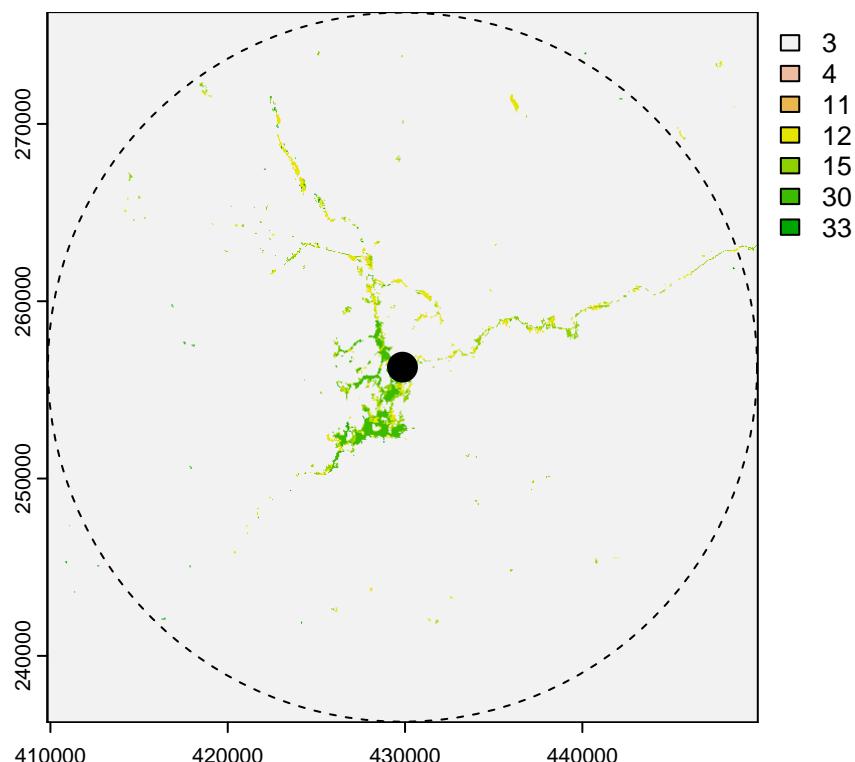
```

Agora que o arquivo foi importado, podemos visualizá-lo.

```

# Visualizar para verificar
# Gradiente de cores padrão não corresponde
# ao mundo real (por exemplo verde não é floresta)
plot(r1985, type="classes")
plot(sf_acesso_20km, add = TRUE, lty ="dashed", color = "black")
plot(sf_acesso_utm, add = TRUE, cex = 2, pch = 19, color = "black")

```



4.5 Calculo de métricas

Vamos olhar alguns exemplos de métricas para cada nível da análise:

- landscape (métricas para a paisagem como um todo).
- class (métricas por classe ou tipo de habitat).
- patch (para a mancha ou fragmento).

Primeiro, precisamos verificar se o raster está no formato correto.

```
check_landscape(r1985)
```

```

##   layer      crs units   class n_classes OK
## 1    1 projected   m integer          7  v

```

```
# layer crs    units   class n_classes OK
# 1 projected  m    integer        7 v
```

Tudo certo (veja a coluna do “OK”)!

4.5.1 Métricas para a paisagem

Vamos começar avaliando a área total da paisagem (área) de estudo.

```
area.total <- lsm_l_ta(r1985)
area.total #160264 Hectares
```

```
## # A tibble: 1 x 6
##   layer level      class     id metric   value
##   <int> <chr>      <int> <int> <chr>    <dbl>
## 1     1 landscape    NA     NA ta      160264.
```

Agora vamos ver a distância total de borda (te= “total edge”).

```
##           used   (Mb) gc trigger   (Mb) max used   (Mb)
## Ncells  7978759 426.2  14900711  795.8  14900711  795.8
## Vcells 86624465 660.9 211545401 1614.0 211545401 1614.0
te <- lsm_l_te(r1985)
te # 547140 metros
```

```
## # A tibble: 1 x 6
##   layer level      class     id metric   value
##   <int> <chr>      <int> <int> <chr>    <dbl>
## 1     1 landscape    NA     NA te      547140.
```

Total de borda mede a configuração da paisagem porque uma paisagem altamente fragmentada terá muitas bordas. No entanto, a borda total é uma medida absoluta, dificultando comparações entre paisagens com áreas totais diferentes. Mas pode ser aplicado para comparar a configuração na mesma paisagem em anos diferentes.

Agora vamos ver a densidade de Borda (“Edge Density”). Densidade de Borda mede a configuração da paisagem porque uma paisagem altamente fragmentada terá valores mais altos. “Densidade” é uma medida adequada para comparações de paisagens com áreas totais diferentes.

```
ed <- lsm_l_ed(r1985)
ed #3.41 metros por hectare
```

```
## # A tibble: 1 x 6
##   layer level      class     id metric   value
##   <int> <chr>      <int> <int> <chr>    <dbl>
## 1     1 landscape    NA     NA ed      3.41
```

4.5.2 Métricas para as classes

Área de cada classe em hectares.

```
lsm_c_ca(r1985)
```

```
## # A tibble: 7 x 6
##   layer level class     id metric   value
##   <int> <chr> <int> <int> <chr>    <dbl>
## 1     1 class    3     NA ca      158582.
## 2     1 class    4     NA ca       1.70
```

```

## 3    1 class    11    NA ca      1.79
## 4    1 class    12    NA ca      548.
## 5    1 class    15    NA ca      563.
## 6    1 class    30    NA ca      526.
## 7    1 class    33    NA ca      41.4

```

Como tem varios classes é dificil de interpretar os resultados porque os numeros (3, 4, 11....) não tem uma referencia do mundo real. Para entender os resultados, precisamos acrescentar nomes para os valores. Ou seja incluir uma coluna de legenda com os nomes para cada classe. Para isso precisamos outro arquivo com os nomes.

Baixar o arquivo de legenda: [mapbiomas_6_legend.xlsx](#).

Agora carregar o arquivo com o código a seguir.

```
class_nomes <- read_excel(file.choose())
```

Agora rodar de novo, com os resultados juntos com a legenda de cada classe. Nos resultados acima, os valores na coluna “class” são as mesmas que tem na coluna “aid” no objeto “class_nomes”, onde também tem os nomes. Assim, podemos repetir, mas agora incluindo os nomes para cada valor de class, com base na ligação (join) entre as colunas.

```

# Área de cada classe em hectares, incluindo os nomes para cada classe
lsm_c_ca(r1985) %>%
  left_join(class_nomes, by = c("class" = "aid"))

# Número de fragmentos (manchas)
lsm_c_np(r1985) %>%
  left_join(class_nomes, by = c("class" = "aid"))

# Maior número de manchas em classes de cobertura classificadas como
# pasto (pasture) e formação campestre (grassland).

#   layer level class    id metric value class_description group_description
# 1 1 class    3    NA np      28 Forest Formation Natural forest
# 2 1 class    4    NA np      2 Savanna Formation Natural forest
# 3 1 class   11    NA np      7 Wetlands Natural non fore
# 4 1 class   12    NA np     246 Grassland Natural non fore.
# 5 1 class   15    NA np     262 Pasture Farming
# 6 1 class   30    NA np      35 Mining Non vegetated
# 7 1 class   33    NA np     50 River,Lake and Ocean Water

```

4.5.3 Métricas para as manchas

Vamos calcular o tamanho de cada mancha agora.

```
mancha_area <- lsm_p_area(r1985) # 630 manchas
mancha_area
```

Agora queremos saber o tamanho da maior mancha em cada class, e portanto o tamanho da maior mancha de mineração.

```

mancha_area %>%
  group_by(class) %>%
  summarise(max_ha = max(value))
# 30.8 hectares (class 15 = mineração)

```

4.5.4 Quais métricas devo escolher?

A decisão deve ser tomada com base em uma combinação de fatores. Incluindo tais fatores como: base teórica, considerações estatísticas, relevância para o objetivo/hipótese e a escala e heterogeneidade na paisagem de estudo.

Queremos caracterizar áreas de mineração na paisagem, e aqui vamos olhar somente uma paisagem, em um momento do tempo. Então as métricas para a paisagem como todo não tem relevância.

Estamos olhando uma classe (mineração), portanto vamos incluir as métricas para classes. Além disso, as métricas de paisagem em nível de classe são mais eficazes na definição de processos ecológicos (Tischendorf, L. Can landscape indices predict ecological processes consistently?. *Landscape Ecology* 16, 235–254 (2001). <https://doi.org/10.1023/A:1011112719782>.).

```
# métricas de composição para a paisagem por classes
list_lsm(level = "class", type = "area and edge metric")

# métricas de configuração para a paisagem por classes
list_lsm(level = "class", type = "aggregation metric")
```

4.5.5 Métricas por classe de mineração

Aqui vamos calcular todos as métricas por classe (função calculate_lsm()).

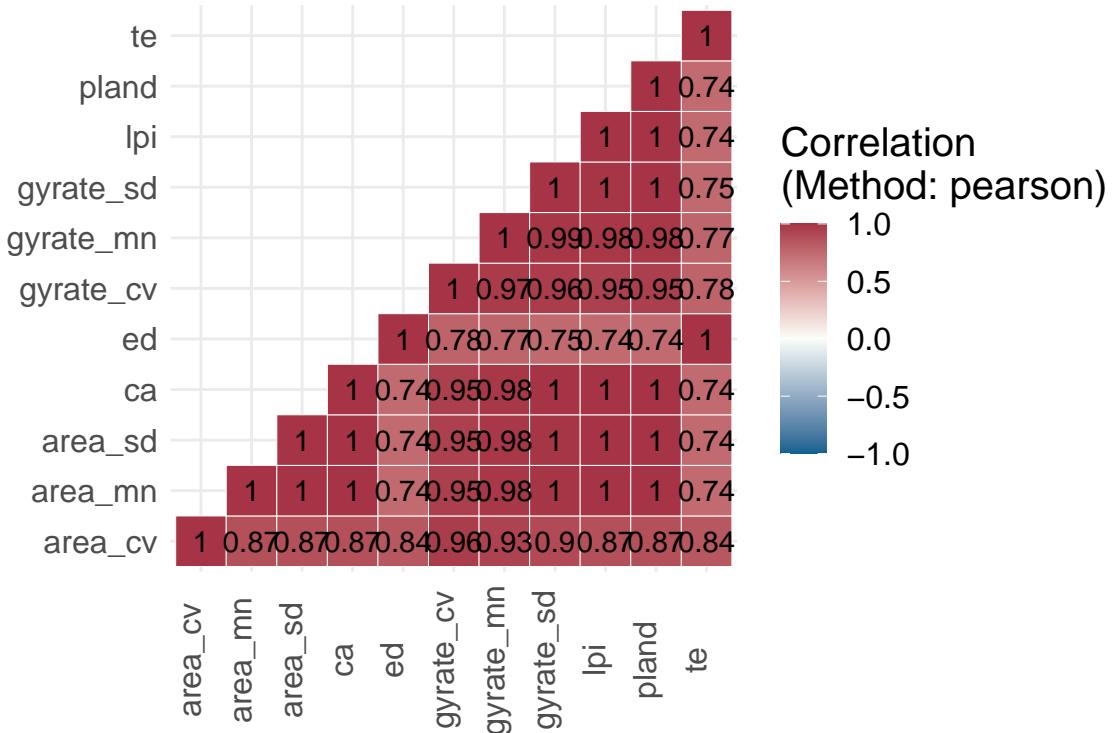
```
# métricas de composição para a paisagem por classes
metrics_comp <- calculate_lsm(r1985, level = "class", type = "area and edge metric")

# métricas de configuração para a paisagem por classes
metrics_config <- calculate_lsm(r1985, level = "class", type = "aggregation metric")
```

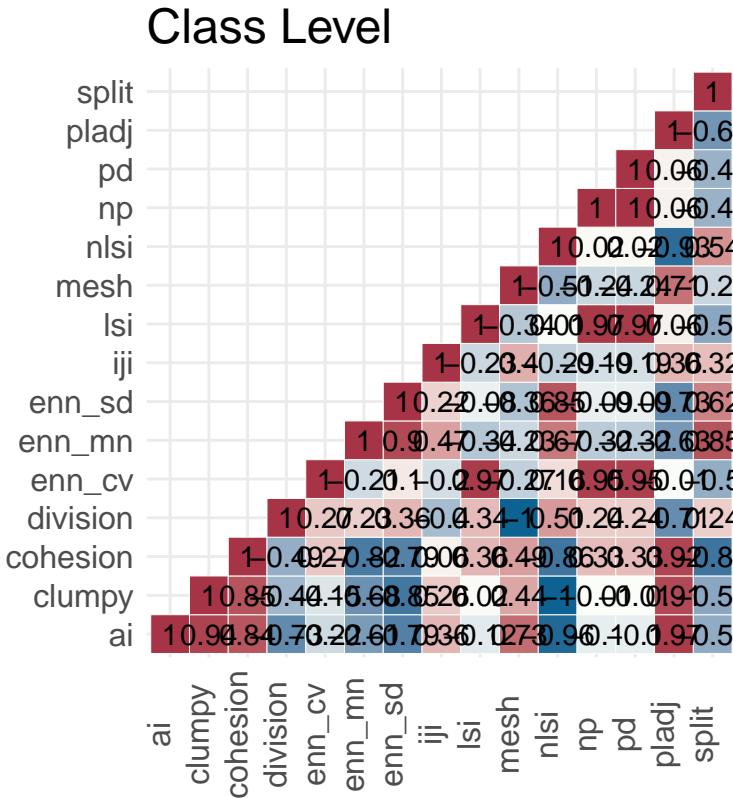
E aqui, calcular correlações entre todos as métricas por classe (função show_correlation()).

```
show_correlation(data = metrics_comp, method = "pearson", labels = TRUE)
```

Class Level



```
show_correlation(data = metrics_config, method = "pearson", labels = TRUE)
```



Temos muitos valores e muitas métricas. Este se chama um “tiro no escuro”, algo cujo resultado se desconhece ou é imprevisível. Isso não é recomendado. Para fazer uma escolha melhor (mais robusta), seguindo princípios básicos da ciência, precisamos ler os estudos anteriores (artigos) para obter as métricas mais relevantes para nosso objetivo e a hipótese a ser testada. Com base em os estudos anteriores e os objetivos vamos incluir 8 métricas nos resultados.

4.5.6 Exportar as métricas

O próximo passo é comunicar os resultados obtidos. Para isso precisamos resumir e apresentar as métricas selecionadas em tabelas e figuras. Agora já fizemos os cálculos, as tabelas e figuras podem ser feitas no R ([figuras](#)), tanto quanto em aplicativos diferentes (por exemplo tabelas através [“tabelas dinâmicas”] no [Microsoft Excel](#) ou [LibreOffice calc](#)). Mas por isso, primeiramente precisamos exportar os resultados (veja mais exemplos aqui: [Introdução ao R import-export](#)).

O arquivo vai sair no mesmo diretório do seu código (verifique com `getwd()`).

```
bind_rows(metrics_comp, metrics_config) -> metricas_1985
write.csv2(metricas_1985, "metricas_lourenco_1985.csv", row.names=FALSE)
```

4.6 Preparando os resultados

A entrada de dados seria com as métricas da paisagem calculados anteriormente.

Vocês devem baixar o arquivo de Excel [metricas_lourenco_1985.xlsx](#). Lembre-se, para facilitar, os dados deve ficar no mesmo diretório do seu código (verifique com `getwd()`).

No caso de um arquivo de Excel simples, a importação poderia ser feita através menu de “Import Dataset” na janela/panel “Environment” de Rstudio. Ou com linhas de código:

```

metricas_1985 <- read_excel("metricas_lourenco_1985.xlsx")
metricas_1985

#   layer level class id      metric      value
#   <dbl> <chr> <dbl> <chr> <chr>      <dbl>
#     1 class     3 NA    area_cv  529.
#     1 class     4 NA    area_cv  22.3
#     1 class    11 NA   area_cv  71.2

```

Ou use o função file.choose(), que faz a busca para arquivos.

```

metricas_1985 <- read_excel(file.choose())
metricas_1985

```

```

## # A tibble: 182 x 6
##   layer level class id      metric      value
##   <dbl> <chr> <dbl> <chr> <chr>      <dbl>
##     1     1 class     3 NA    area_cv  529.
##     2     1 class     4 NA    area_cv  22.3
##     3     1 class    11 NA   area_cv  71.2
##     4     1 class    12 NA   area_cv  169.
##     5     1 class    15 NA   area_cv  175.
##     6     1 class    30 NA   area_cv  276.
##     7     1 class    33 NA   area_cv  74.2
##     8     1 class     3 NA    area_mn 5664.
##     9     1 class     4 NA    area_mn  0.848
##    10     1 class    11 NA   area_mn  0.255
## # i 172 more rows

```

Os dados são padronizados (“tidy”), mas ainda não parece adequados para apresentação em tabelas ou figuras. Temos muitos valores e muitas métricas (listadas na coluna “metric”). Com base em os estudos anteriores e os objetivos vamos incluir 8 métricas (4 de composição e 4 de configuração).

Métricas de composição:

- mean patch area (lsm_c_area_mn) Área médio das manchas por classe.
- SD patch area (lsm_c_area_sd) Desvio padrão das áreas das manchas por classe.
- class area percentage of landscape (lsm_c_pland) Porcentagem de área na paisagem por classe.
- largest patch index (lsm_c_lpi) Índice de maior mancha (proporção da paisagem).

Métricas de configuração:

- aggregation index (lsm_c_ai) Índice de agregação.
- patch cohesion index (lsm_c_cohesion) Índice de coesão das manchas.
- number of patches (lsm_c_np) Número de manchas.
- patch density (lsm_c_pd) Densidade de manchas.

Escolheremos (atraves um filtro) as métricas que queremos para obter uma tabela de dados. Mantendo os dados originais, assim sendo para apresentar mais métricas nos resultados, preciso somente acrescentar mais no código.

```

# Arquivo com os nomes das classes
class_in <- "C:\\\\Users\\\\user\\\\Documents\\\\Articles\\\\gis_layers\\\\gisdata\\\\inst\\\\raster\\\\mapbiomas_cover_1"
class_nomes <- read_excel(class_in)

# Especificar métricas desejados
met_comp <- c("pland", "lpi", "area_mn", "area_sd")
met_conf <- c("ai", "cohesion", "np", "pd")

```

```
met.todos <- c(met_comp, met_conf)

# Escolher métricas desejados do conjunto completo
metricas_1985 %>%
filter(metric %in% met.todos) %>%
left_join(class_nomes, by = c("class" = "aid")) -> metricas_nomes
```

4.7 Uma tabela versatil

Mas, ainda não tem uma coluna com os nomes das métricas. Portanto, solução simples é de exportar no formato de .csv e finalizar/editar no Excel / calc.

Outra opção que pode facilitar, particularmente quando pode há mudanças e revisões, é produzir a tabela no R. Aqui vamos repetir no R os passos que vocês conhecem com as ferramentas de Excel (arraste e solte, copiar-colar, filtro, tabela dinâmica).

4.8 Reorganização

Escolhendo as colunas desejadas (select), reorganizando para as métricas ficam nas colunas (pivot_wider) e colocando as colunas novas na sequência desejada (select).

```
metricas_nomes %>%
# Escolher métricas desejados do conjunto completo de métricas.
dplyr::select(c(type_class, classe_descricao, hexadecimal_code,
metric, value)) %>%
# reorganizando
pivot_wider(names_from = metric, values_from = value) -> metricas_tab
```

4.9 Uma figura elegante

Uma imagem vale mais que mil palavras. Portanto, gráficos/figuras/imagens são uma das mais importantes formas de comunicar a ciência.

Como exemplo ilustrativo, aqui vamos produzir gráficos comparando métricas de composição e configuração da paisagem ao redor do Garimpo do Lourenço.

É uma boa ideia gastar bastante tempo para tornar figuras científicas as mais informativas e atraentes possíveis. Escusado será dizer que a precisão empírica é primordial. E por isso, o que fica excluído/omitido é tão importante quanto o que foi incluído. Para ajudar, você deve se perguntar o seguinte ao criar uma figura: eu apresentaria essa figura em uma apresentação para um grupo de colegas? Eu o apresentaria a um público de não especialistas? Eu gostaria que essa figura aparecesse em um artigo de notícias sobre meu trabalho? É claro que todos esses locais exigem diferentes graus de precisão, complexidade e estética, mas uma boa figura deve servir para educar simultaneamente públicos muito diferentes.

Tabelas versus gráficos — A primeira pergunta que você deve se fazer é se você pode transformar aquela tabela (chata e feia) em algum tipo de gráfico. Você realmente precisa dessa tabela no texto principal? Você não pode simplesmente traduzir as entradas das células em um gráfico de barras/columnas/xy? Se você pode, você deve. Quando uma tabela não pode ser facilmente traduzida em uma figura, na maioria das vezes a provavelmente pertence às Informações Suplementares/Anexos/Apêndices.

4.9.1 Gráfico de barra

Primeiramente, vamos produzir uma gráfico de barra comparando a proporção que cada classe representa na paisagem.

```
# Incluindo cores conforme legenda da Mapbiomas Coleção 6
# Legenda nomes ordem alfabetica
classe_cores <- c("Campo Alagado e Área Pantanosa" = "#45C2A5",
"Formação Campestre" = "#B8AF4F",
"Formação Florestal" = "#006400",
"Formação Savântica" = "#00ff00",
"Mineração" = "#af2a2a",
"Pastagem" = "#FFD966",
"Rio, Lago e Oceano" = "#0000FF")
```

E agora o grafico.....

```
# Grafico de barra basica
metricas_tab %>%
  mutate(class_prop = pland) %>%
  ggplot(aes(x = classe_descricao, y = class_prop)) +
  geom_col()

# Agora com ajustes
# Agrupando por tipo (natural e antropico)
# Com cores conforme legenda da Mapbiomas Coleção 6
# Corrigindo texto dos eixos.
# Mudar posição da legenda para o texto com nomes longas encaixar.
metricas_tab %>%
  mutate(class_prop = pland) %>%
  ggplot(aes(x = type_class, y = class_prop,
  fill = classe_descricao)) +
  scale_fill_manual("classe", values = classe_cores) +
  geom_col(position = position_dodge2(width = 1)) +
```

```

coord_flip() +
labs(title = "MapBiomas cobertura da terra",
subtitle = "Entorno do Garimpo do Lorenço 1985",
y = "Proporção da paisagem (%)",
x = "") +
theme(legend.position="bottom") +
guides(fill = guide_legend(nrow = 4))

```

Uma imagem vale mais que mil palavras:

Mas existe uma separação grande na faixa de valores e ainda é difícil de ver todas as classes. Temos uma distribuição com valores muito mais altos comparada com os outros. extremos. Uma solução seria uma transformação (por exemplo “log”), assim os valores ficarem mais próximos.

4.9.2 Gráfico de boxplot

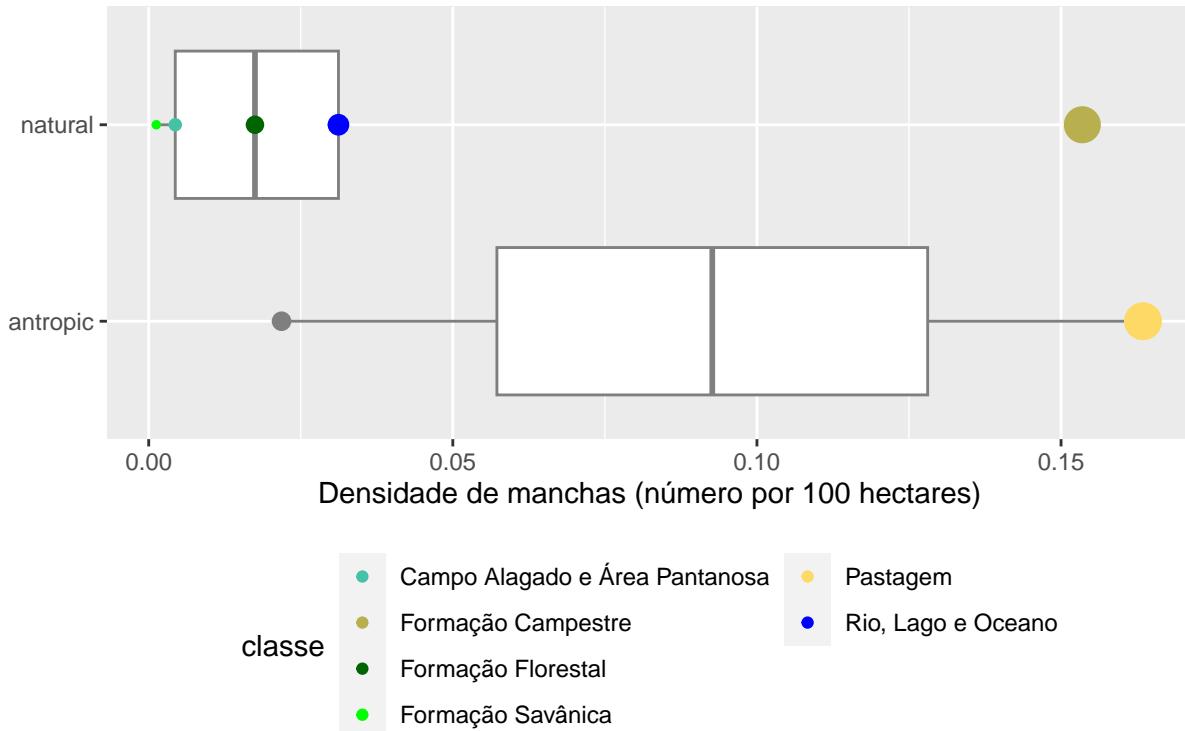
Agora com uma métrica de configuração:

```

# Métrica de configuração: Densidade de manchas (coluna "pd").
# Agrupando por tipo (natural e antrópico)
# Incluindo boxplot indicando tendência central (mediano)
# Com cores conforme legenda da Mapbiomas Coleção 6
# Tamanho dos pontos proporcional o numero de manchas
# Corrigindo texto dos eixos.
# Mudar posição da legenda para o texto com nomes longas encaixar.
metricas_tab %>%
ggplot(aes(x = type_class, y = pd)) +
geom_boxplot(colour = "grey50") +
geom_point(aes(size = np, colour = classe_descricao)) +
scale_color_manual("classe", values = classe_cores) +
scale_size(guide = "none") +
coord_flip() +
labs(title = "MapBiomas cobertura da terra",
subtitle = "Entorno do Garimpo do Lorenço 1985",
y = "Densidade de manchas (número por 100 hectares)",
x = "") +
theme(legend.position="bottom") +
guides(col = guide_legend(nrow = 4))

```

MapBiomas cobertura da terra Entorno do Garimpo do Lorenço 1985



4.10 Comparação entre anos

Calcular as métricas para 3 anos.

```
# metricas desejados
what_metricas <- c("lsm_c_pland", "lsm_c_lpi", "lsm_c_area_mn", "lsm_c_area_sd",
                     "lsm_c_ai", "lsm_c_cohesion", "lsm_c_np", "lsm_c_pd")
# rodar
metricas_anos <- sample_lsm(landscape = mapbiomas_85a20,
                             y = acesso_buffers,
                             plot_id = data.frame(acesso_buffers)[, 'raio_km'],
                             what = what_metricas,
                             edge_depth = 1)

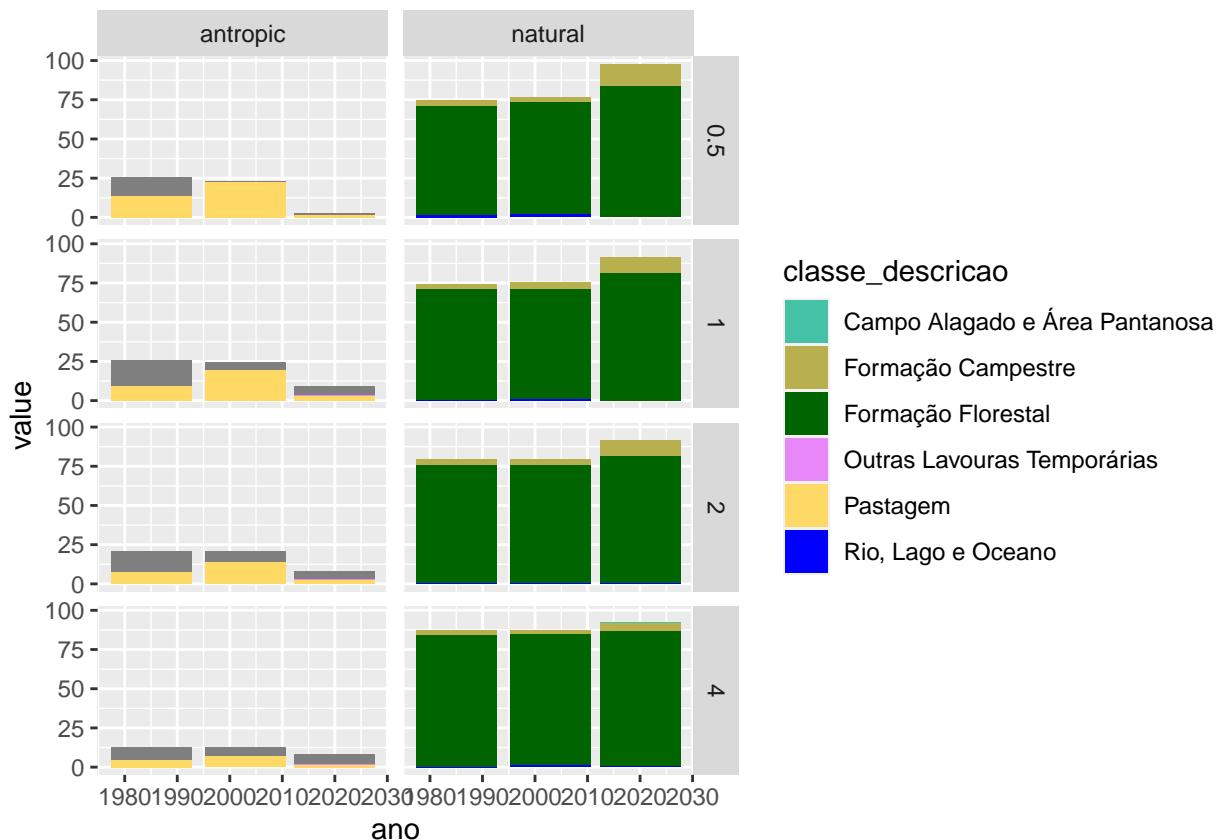
# Organizar dados
# Dados referentes os buffers
resultados_anos <- acesso_buffers %>%
  left_join(metricas_anos %>%
    dplyr::mutate(value = round(value, 2),
    ano = case_when(layer==1 ~1985,
                    layer==2~2003,
                    layer==3~2020)) %>%
    dplyr::select(ano, plot_id, class, metric, value),
    by=c("raio_km"="plot_id"))
# Dados referentes os classes
resultados_anos <- resultados_anos %>%
```

```
left_join(class_nomes, by = c("class" = "aid"))
```

Agora grafico de barra com varios anos

```
# It's recommended to use a named vector
# Legenda nomes ordem alfabetica
classe_cores <- c("Campo Alagado e Área Pantanosa" = "#45C2A5",
"Formação Campestre" = "#B8AF4F",
"Formação Florestal" = "#006400",
"Formação Savânicas" = "#00ff00",
"Mineração" = "#af2a2a",
"Pastagem" = "#FFD966",
"Rio, Lago e Oceano" = "#0000FF",
"Outras Lavouras Temporárias" = "#e787f8")

resultados_anos %>%
  mutate(rcor = paste("#", hexdecimal_code, sep="")) %>%
  filter(metric=="pland") %>%
  ggplot(aes(x=ano, y=value)) +
  geom_col(position="stack", aes(fill=classe_descricao)) +
  scale_fill_manual(values = classe_cores) +
  facet_grid(raio_km~type_class)
```



Agora com “pland” e densidade de manchas juntos.

```
resultados_anos %>%
  mutate(rcor = paste("#", hexdecimal_code, sep="")) %>%
```

```

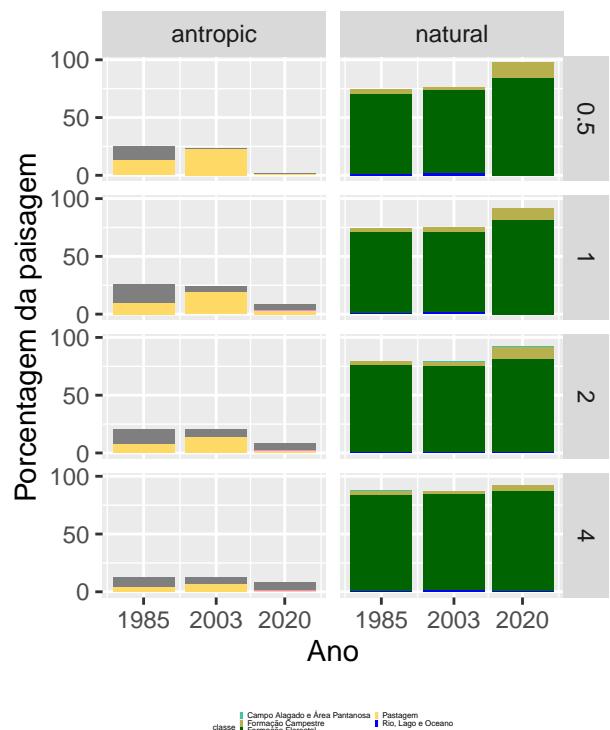
filter(metric=="pland") %>%
ggplot(aes(x=ano, y=value)) +
geom_col(position="stack", aes(fill=classe_descricao)) +
scale_fill_manual("classe", values = classe_cores) +
scale_y_continuous(breaks=c(0,50,100)) +
scale_x_continuous(breaks=c(1985,2003, 2020)) +
facet_grid(raio_km~type_class) +
labs(title = "MapBiomas cobertura da terra",
subtitle = "Entorno do Garimpo do Lorenço 1985-2020",
y = "Porcentagem da paisagem",
x = "Ano") +
theme(legend.position="bottom",
      legend.title = element_text(size = 3),
      legend.text = element_text(size = 3),
      legend.key.size = unit(0.1, "lines")) +
guides(fill = guide_legend(nrow = 4)) -> fig_pland

# Densidade de manchas
resultados_anos %>%
  mutate(rcor = paste("#", hexadecimal_code, sep="")) %>%
  filter(metric=="pd") %>%
ggplot(aes(x = factor(ano), y = value)) +
geom_boxplot(colour = "grey50") +
geom_point(aes(colour = classe_descricao)) +
scale_color_manual("classe", values = classe_cores) +
scale_size(guide = "none") +
facet_grid(raio_km~type_class) +
labs(title = "MapBiomas cobertura da terra",
subtitle = "Entorno do Garimpo do Lorenço 1985-2020",
y = "Densidade de manchas (número por 100 hectares)",
x = "Ano") +
theme(legend.position="bottom",
      legend.title = element_text(size = 3),
      legend.text = element_text(size = 3),
      legend.key.size = unit(0.1, "lines")) +
guides(fill = guide_legend(nrow = 4)) -> fig_pd

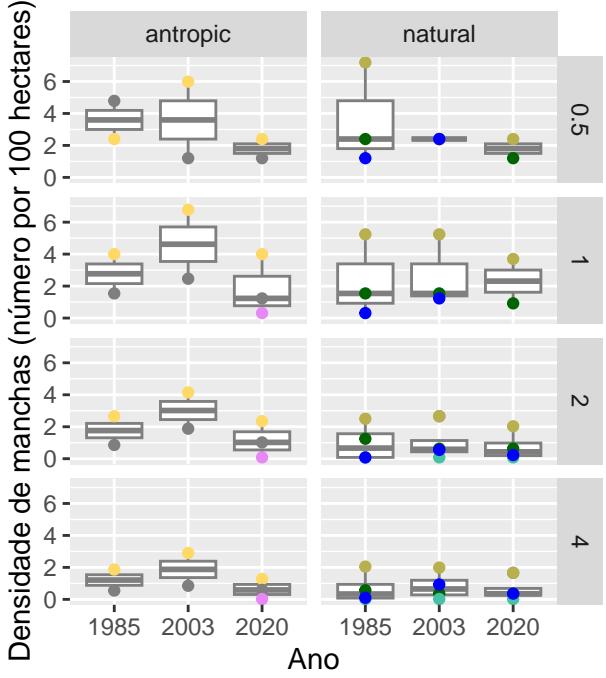
grid.arrange(fig_pland, fig_pd, nrow=1)

```

MapBiomas cobertura da terra
Entorno do Garimpo do Lorenço 1985–2020



MapBiomas cobertura da terra
Entorno do Garimpo do Lorenço 1985–2020



4.11 Conclusões e próximos passos

Os resultados apresentados na figura anterior não segam os resultados esperados que a cobertura de classes antrópicos ia aumentar ao longo do tempo. Para entender melhor os padões, precisamos:

- Verificar padrões nas outras metricas calculados
- Verificar padrões com mais anos
- Verificar padrões usando poligonos/pontos dos processos de mineração (dados no SIGMINE <https://www.gov.br/anm/pt-br> e https://app.anm.gov.br/dadosabertos/SIGMINE/PROCESOS_MINERARIOS/)

Alem disso, seria relevante buscar complementar os dados de MapBiomas com uma classificação supervisionado usando imagens de Sentinel-2 (exemplo com QGIS Semi-Automatic Classification plugin aqui: <https://fromgistsors.blogspot.com/2016/09/basic-tutorial-2.html>). Assim para aumentar a precisão dos resultados.

Uma forma alternativa para visualização as mudanças entre anos seria um diagrama “Sankey”/“Alluvial”. Como exemplo, veja figura 3 no artigo “Rapid land use conversion in the Cerrado has affected water transparency in a hotspot of ecotourism, Bonito, Brazil” <https://doi.org/10.1177/19400829221127087>.

Prata River Basin LULCC

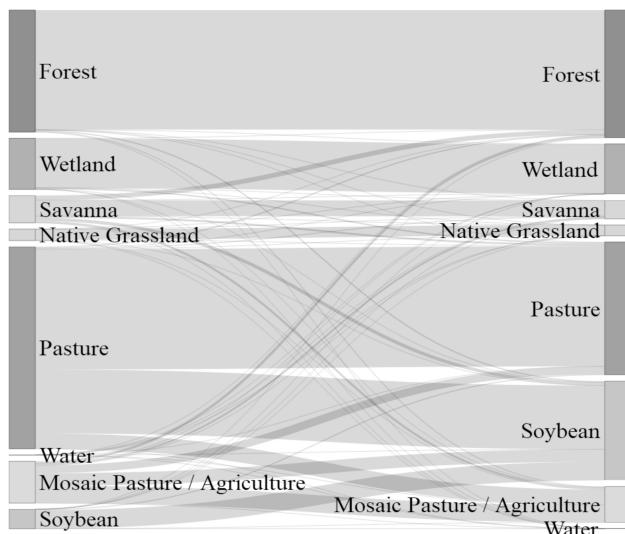


Figura 4.1: Diagrama Sankey mostrando a mudança de uso da terra na bacia do rio Prata entre 2010 (lado esquerdo) e 2020 (lado direito). Fonte: Figura 3. Chiaravalloti et. al. 2022. Tropical Conservation Science. doi:10.1177/19400829221127087

No R pode fazer com o pacote “networkD3” segue tutoriais:

- <https://www.displayr.com/sankey-diagrams-r/>
- <https://epirhandbook.com/en/diagrams-and-charts.html#alluvialsankey-diagrams>
- <https://rpubs.com/droach/CPP526-codethrough>

Part IV

R e RStudio: Código com certeza

5 Pré-requisitos

Conteúdo copiado, com pequenas alterações e atualizações de [Capítulo 3 Pré-requisitos](#), do livro [Análises Ecológicas no R](#). O livro está licenciado sob a licença [BY-NC-ND 4.0](#) : <https://creativecommons.org/licenses/by-nc-nd/4.0/>.

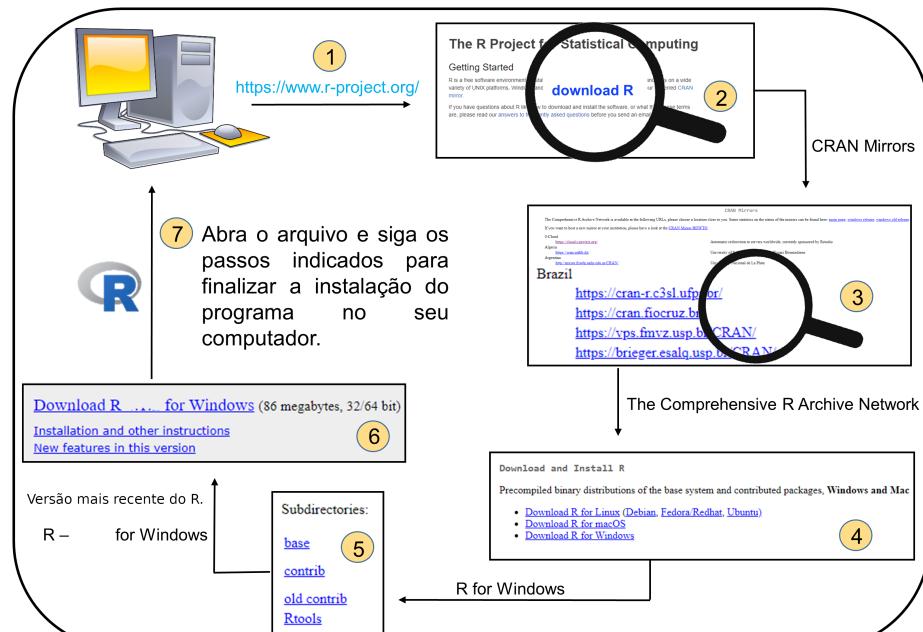
5.1 Introdução

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.

5.2 Instalação do R

Abaixo descrevemos os sete passos necessários para a instalação do programa R no seu computador. Para Windows, use link <https://cran.r-project.org/bin/windows/base/> e vai para passo 6 :

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



Fonte das figuras:
imagem computador https://pt.wikipedia.org/wiki/Computador_pessoal
imagem da lupa <https://openclipart.org/detail/185356/magnifier>

Figura 5.1: Esquema ilustrativo demonstrando os passos necessários para instalação do programa R no computador.

Importante

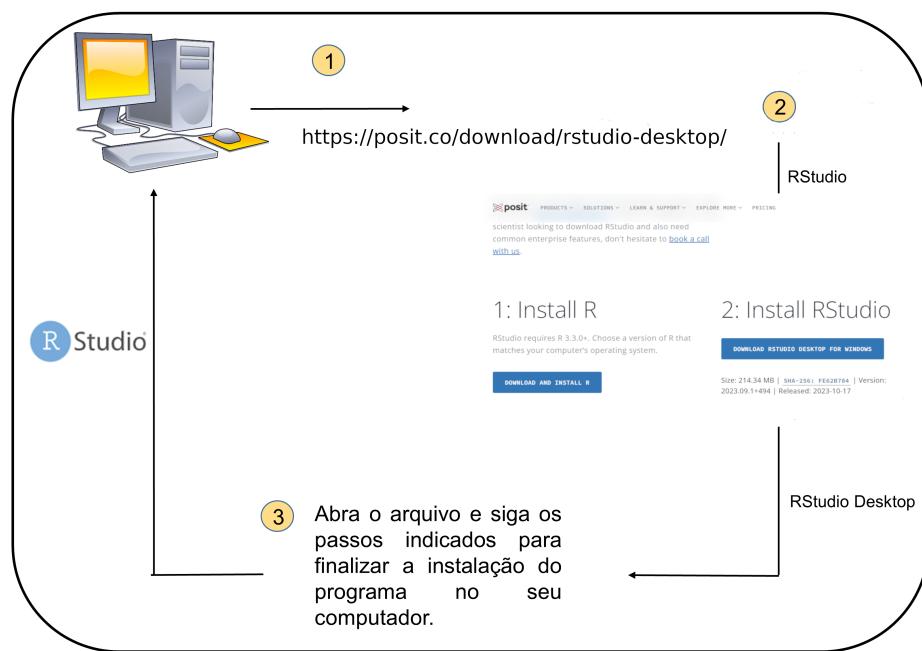
Para o Sistema Operacional (SO) Windows, alguns pacotes são dependentes da instalação separada do **Rtools40**. Da mesma forma, GNU/Linux e MacOS também possuem dependências de outras bibliotecas para pacotes específicos, mas que não abordaremos aqui. Essas informações de dependência geralmente são retornadas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

5.3 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:

1. Para fazer o download do RStudio, acessamos o site <https://posit.co/download/rstudio-desktop/>,
2. Clique em **download...**,
3. Abra o arquivo que foi baixado no seu computador e siga os passos indicados para finalizar a instalação do programa RStudio.



Fonte das figuras: imagem computador https://pt.wikipedia.org/wiki/Computador_pessoal

Figura 5.2: Esquema ilustrativo demonstrando os passos necessários para instalação do programa RStudio no computador.

5.4 Versão do R

Todas os códigos, pacotes e análises disponibilizados no livro foram realizados no Programa R versão 4.3.2 (10-12-2023).

5.5 Pacotes

Descrevemos no Capítulo 6 o que são e como instalar os pacotes para realizar as análises estatísticas no R.

Importante

Criamos o pacote **eprdados** que contém todas as informações e dados utilizados neste livro. Assim, recomen-

damos que você instale e carregue este pacote no início de cada capítulo para ter acesso aos dados necessários para executar as funções no R.

Para a instalação do pacote `eprdados` no macOS, você precisará ter instalado o programa XCode que pode ser baixado [aqui](#). Este programa é disponibilizado gratuitamente pela Apple e é necessário para compilar quaisquer programas distribuídos em código fonte (ou seja, sem um binário). Após instalar esse programa e o pacote `devtools`, você poderá instalar o `eprdados` utilizando as instruções abaixo.

Abaixo, listamos todos os pacotes utilizados no livro. Você pode instalar os pacotes agora ou esperar para instalá-los quando ler o Capítulo 6 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", "d
```

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("darrennorris/eprdados")
install_github("ropensci/rnaturalearthhires")
```

5.6 Dados

A maioria dos exemplos do livro utilizam dados reais extraídos de artigos científicos que já foram publicados ou dados que foram coletados por um dos autores deste livro. Todos os dados, publicados ou simulados, estão disponíveis no pacote `eprdados`. Além disso, em cada capítulo fazemos uma breve descrição dos dados para facilitar a compreensão sobre como essas variáveis estão relacionadas com as perguntas do exemplo.

6 Introdução ao R

Conteúdo copiado, com pequenas alterações e atualizações de [Capítulo 4 Introdução ao R](#), do livro [Análises Ecológicas no R](#). O livro está licenciado sob a licença [BY-NC-ND 4.0](#) : <https://creativecommons.org/licenses/by-nc-nd/4.0/>.

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(palmerpenguins)

## Dados necessários
pinguins <- palmerpenguins::penguins
```

6.1 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.

6.2 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. 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). 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](#). 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 5. Reserve um tempo para explorar esta página do R-Project

(<https://www.r-project.org/>). 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. 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 ou no próprio terminal para usuários Linux e MacOS, existem alguns IDEs específicos para facilitar nosso uso dessa linguagem.

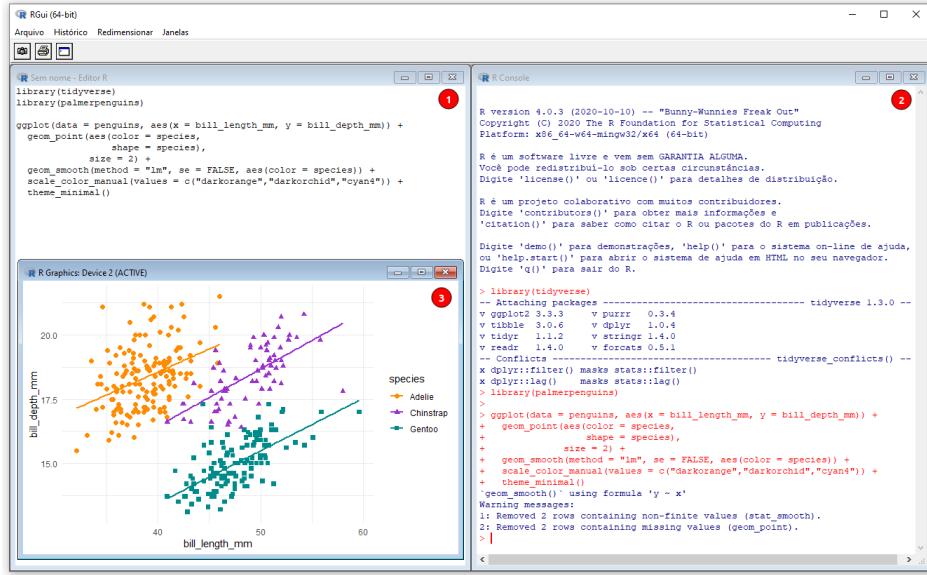


Figura 6.1: Interface do RGui. Os números indicam: (1) R Script, (2) R Console, e (3) R Graphics.

Dessa forma, nós 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 “fugar” 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 Margins 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

Importante

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

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 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).

6.3 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.

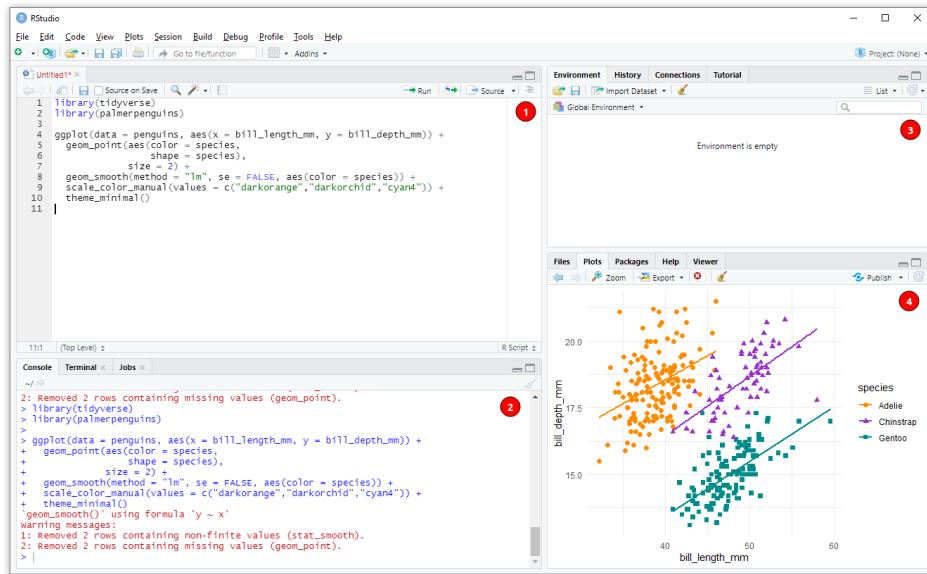


Figura 6.2: Interface do RStudio. Os números indicam: (1) janela com abas de Script, R Markdown, dentre outras; (2) janela com abas de Console, Terminal e Jobs; (3) janela com abas de Environment, History, Conections e Tutorial; e (4) janela com abas de Files, Plots, Packages, Help e Viewer.

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

6.3.1 Console

O console é onde a versão da linguagem R instalada é carregada para executar os códigos da linguagem R (Figura 6.2 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, [37].

```
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] 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
```

Podemos ver o histórico dos códigos executados no console na aba **History** (Figura 6.2 janela 3).

6.3.2 Scripts

Scripts são arquivos de texto simples, criados com a extensão (terminação) .R (Figura 6.2 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 fina 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 lidas 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 uni
## [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](#). Para simplificar a vida temos o pacote `styler` (<https://styler.r-lib.org/>), que serve para adequar o código.

Ainda em relação aos scripts, temos os *Code Snippets* (Fragments 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 -----
```

6.3.3 Operadores

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

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 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
```

6.3.4 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** . 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**, **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
```

```
val1 <- 10
```

```
val2 <- 2
```

```
## Operações com objetos e atribuição
```

```
adi <- val1 + val2
```

```
adi
```

```
## [1] 12
```

6.3.5 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  
## [42] 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [42] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 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.

```
## Atribuição 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  
## [42] 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

6.3.6 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 estamos 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: 20192 em 15/12/2023
nrow(available.packages())
```

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. Para isso, podemos ir em **Tools > Install Packages...**, ou ir na aba **Packages**, 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 instalação dos pacotes
.libPaths()
```

```
## [1] "C:/Users/user/AppData/Local/R/win-library/4.3"
## [2] "C:/Program Files/R/R-4.3.2/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** 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 como por exemplo: [GitHub](#), [GitLab](#) e [Bioconductor](#). Para instalar pacotes do GitHub, por exemplo, precisamos

instalar e carregar o pacote `devtools`. Para funcionar, deve instalar “RTools” antes (<https://cran.r-project.org/bin/windows/Rtools/>) .

```
## 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 `eprdados` do repositório do GitHub `darrennorris/eprdados` e depois utilizar a função `library()` para carregá-lo. Para funcionar, deve instalar “RTools” antes (<https://cran.r-project.org/bin/windows/Rtools/>) .

```
## Instalar pacote do github
devtools::install_github("darrennorris/eprdados")

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

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

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

```
## Package: vegan
## Title: Community Ecology Package
## Version: 2.6-4
## Authors@R: c(person("Jari", "Oksanen", role=c("aut", "cre"),
##                 email="jhoksane@gmail.com"), person("Gavin L.", "Simpson",
##                 role="aut", email="ucfagls@gmail.com"), person("F. Guillaume",
##                 "Blanchet", role="aut"), person("Roeland", "Kindt", role="aut"),
##                 person("Pierre", "Legendre", role="aut"), person("Peter R.",
##                 "Minchin", role="aut"), person("R.B.", "O'Hara", role="aut"),
##                 person("Peter", "Solymos", role="aut"), person("M. Henry H.",
##                 "Stevens", role="aut"), person("Eduard", "Szoecs", role="aut"),
##                 person("Helene", "Wagner", role="aut"), person("Matt", "Barbour",
##                 role="aut"), person("Michael", "Bedward", role="aut"), person("Ben",
##                 "Bolker", role="aut"), person("Daniel", "Borcard", role="aut"),
##                 person("Gustavo", "Carvalho", role="aut"), person("Michael",
##                 "Chirico", role="aut"), person("Miquel", "De Caceres", role="aut"),
##                 person("Sebastien", "Durand", role="aut"), person("Heloisa Beatriz
##                 Antoniazi", "Evangelista", role="aut"), person("Rich", "FitzJohn",
##                 role="aut"), person("Michael", "Friendly", role="aut"),
##                 person("Brendan", "Furneaux", role="aut"), person("Geoffrey",
##                 "Hannigan", role="aut"), person("Mark O.", "Hill", role="aut"),
##                 person("Leo", "Lahti", role="aut"), person("Dan", "McGlinn",
##                 role="aut"), person("Marie-Helene", "Ouellette", role="aut"),
##                 person("Eduardo", "Ribeiro Cunha", role="aut"), person("Tyler",
##                 "Smith", role="aut"), person("Adrian", "Stier", role="aut"),
##                 person("Cajo J.F.", "Ter Braak", role="aut"), person("James",
##                 "Weedon", role="aut"))
## Depends: permute (>= 0.9-0), lattice, R (>= 3.4.0)
## Suggests: parallel, tcltk, knitr, markdown
## Imports: MASS, cluster, mgcv
```

```

## VignetteBuilder: utils, knitr
## Description: Ordination methods, diversity analysis and other functions for
##               community and vegetation ecologists.
## License: GPL-2
## BugReports: https://github.com/vegandevs/vegan/issues
## URL: https://github.com/vegandevs/vegan
## NeedsCompilation: yes
## Packaged: 2022-10-11 08:36:07 UTC; jarioksa
## Author: Jari Oksanen [aut, cre], Gavin L. Simpson [aut], F. Guillaume
##         Blanchet [aut], Roeland Kindt [aut], Pierre Legendre [aut], Peter R.
##         Minchin [aut], R.B. O'Hara [aut], Peter Solymos [aut], M. Henry H.
##         Stevens [aut], Eduard Szoecs [aut], Helene Wagner [aut], Matt
##         Barbour [aut], Michael Bedward [aut], Ben Bolker [aut], Daniel
##         Borcard [aut], Gustavo Carvalho [aut], Michael Chirico [aut], Miquel
##         De Caceres [aut], Sebastien Durand [aut], Heloisa Beatriz Antoniazi
##         Evangelista [aut], Rich FitzJohn [aut], Michael Friendly [aut],
##         Brendan Furneaux [aut], Geoffrey Hannigan [aut], Mark O. Hill [aut],
##         Leo Lahti [aut], Dan McGlinn [aut], Marie-Helene Ouellette [aut],
##         Eduardo Ribeiro Cunha [aut], Tyler Smith [aut], Adrian Stier [aut],
##         Cajo J.F. Ter Braak [aut], James Weedon [aut]
## Maintainer: Jari Oksanen <jhoksane@gmail.com>
## Repository: CRAN
## Date/Publication: 2022-10-11 12:40:02 UTC
## Built: R 4.3.2; x86_64-w64-mingw32; 2023-11-02 03:07:22 UTC; windows
## Archs: x64
##
## -- File: C:/Users/user/AppData/Local/R/win-library/4.3/vegan/Meta/package.rds

```

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	Alopogeni	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

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** 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)
```

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()
```

6.3.7 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. O arquivo de ajuda do R possui 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.

```
## 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 emails, manuais, páginas na internet e livros sobre o programa.

6.3.8 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")
```

6.3.9 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 (2023). _R: A Language and Environment for Statistical
##   Computing_. R Foundation for Statistical Computing, Vienna, Austria.
##   <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 = {2023},
##   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")
```

```
## To cite package 'vegan' in publications use:
##
##   Oksanen J, Simpson G, Blanchet F, Kindt R, Legendre P, Minchin P, O'Hara R,
##   Solymos P, Stevens M, Szoecs E, Wagner H, Barbour M, Bedward M, Bolker B,
```

```

## Borcard D, Carvalho G, Chirico M, De Caceres M, Durand S, Evangelista H,
## FitzJohn R, Friendly M, Furneaux B, Hannigan G, Hill M, Lahti L, McGlinn D,
## Ouellette M, Ribeiro Cunha E, Smith T, Stier A, Ter Braak C, Weedon J
## (2022). _vegan: Community Ecology Package_. R package version 2.6-4,
## <https://CRAN.R-project.org/package=vegan>.
##
## A BibTeX entry for LaTeX users is
##
## @Manual{,
##   title = {vegan: Community Ecology Package},
##   author = {Jari Oksanen and Gavin L. Simpson and F. Guillaume Blanchet and Roeland Kindt and Pier
## year = {2022},
## note = {R package version 2.6-4},
## url = {https://CRAN.R-project.org/package=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")

```

6.3.10 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, descance, 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)
```

```
## Warning in data(dune): data set 'dune' not found
```

```
## 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)
```

```
## This is vegan 2.6-4
##
## Attaching package: 'vegan'
##
## The following object is masked from 'package:parsnip':
##   nullmodel
```

```

## Carregar dados
data(dune)

## Função do pacote vegan
decostand(dune[1:6, 1:6], "hell")

## Achimill Agrostol Airaprae Alopogeni 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()`.

```

## Soma das colunas
colsums(dune)

```

```
## Error in colsums(dune): could not find function "colsums"
```

```

## Soma das colunas
colSums(dune)

```

```

## Achimill Agrostol Airaprae Alopogeni Anthodor Bellpere Bromhord Chenalbu Cirsarve
##      16     48      5     36     21     13     15      1      2
## Comapalu Eleopalu Elymrepe Empenigr Hyporadi Juncarti Juncbufo Lolipere Planlanc
##      4     25     26      2      9     18     13     58     26
## Poaprat Poatriv Ranuflam Rumeacet Sagiproc Salirepe Scorautu Trifprat Trifrepe
##      48     63     14     18     20     11     54      9     47
## Vicalath Bracruta Callcusp
##      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.

6.4 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 7.

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.

6.4.1 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" "Allopogeni" "Anthodor" "Bellpere" "Bromhord"
## [8] "Chenalbu" "Cirsarve" "Comapalu" "Eleopalu" "Elymrepe" "Empenigr" "Hyporadi"
## [15] "Juncarti" "Juncbufo" "Lolipere" "Planlanc" "Poaprat" "Poatriv" "Ranuflam"
## [22] "Rumeacet" "Sagiproc" "Salirepe" "Scorautu" "Trifprat" "Trifrepe" "Vicilath"
## [29] "Bracruta" "Callcusp"
##
## $row.names
## [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"  "13"  "14"  "15"  "16"
## [17] "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" # atenção 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.character(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. 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*)

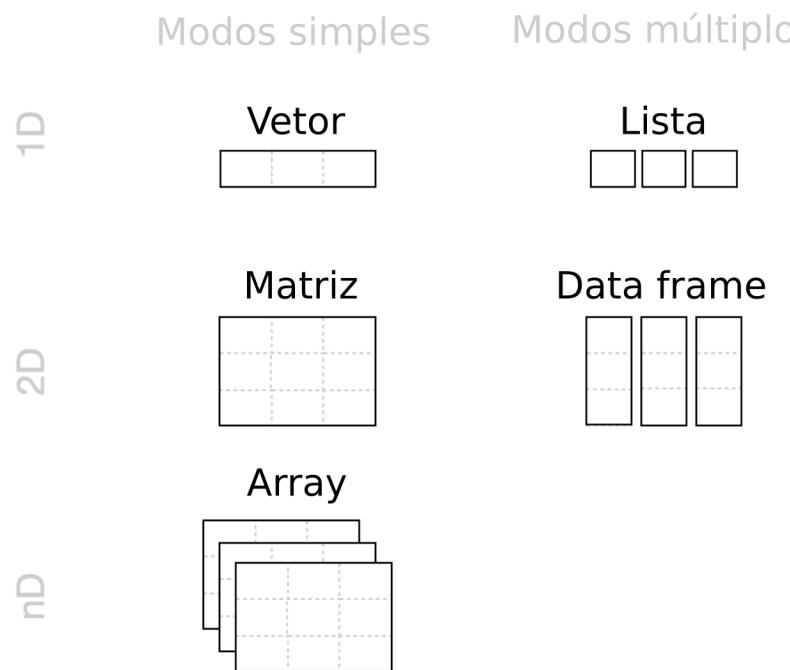


Figura 6.3: Estruturas de dados mais comuns no R: vetores, matrizes, arrays, listas e data frames. Adaptado de: R for Data Science (2e) .

Vetor

Vetores representam o encadeamento de elementos numa sequência unidimensional. Dessa forma, no R, 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"
```

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" "90"
## [7] "11"         "19"         "92"         "85"         "36"         "42"
```

```

## [13] "86"      "89"      "54"
## 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 7 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 floresta pastagem cerrado floresta pastagem cerrado floresta
## [10] pastagem pastagem cerrado cerrado pastagem floresta cerrado pastagem floresta
## [19] floresta floresta
## 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] media alta media alta media alta alta baixa media baixa alta media baixa
## [14] alta media baixa baixa baixa media media
## Levels: baixa < media < alta

## Conversão
fator <- as.factor(x = sample(x = c("floresta", "pastagem", "cerrado"),
                               size = 20, replace = TRUE))
fator

## [1] floresta cerrado floresta cerrado pastagem cerrado floresta pastagem pastagem
## [10] pastagem pastagem cerrado floresta cerrado floresta cerrado floresta pastagem
## [19] pastagem cerrado
## Levels: cerrado floresta pastagem

```

Matriz

A matriz representa dados no formato de tabela, com linhas e colunas. As linhas geralmente representam unidades amostrais (loais, 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 segunda 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,]     0     1     0     1     0
## [2,]     1     0     0     1     1
## [3,]     1     0     1     0     0
## [4,]     0     0     0     0     1
## [5,]     1     1     1     1     1
##
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]     0     1     0     1     0
## [2,]     0     1     1     1     1
## [3,]     1     1     0     1     1
## [4,]     0     1     0     0     0
## [5,]     0     1     0     1     1
##
## , , 3
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]     1     1     0     0     1
## [2,]     0     0     0     0     1
## [3,]     0     1     1     0     0
## [4,]     0     0     1     1     0
## [5,]     1     1     1     0     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
##
## $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 a 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. Nossa blocos 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
```

6.4.2 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.

```
## 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.

## 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.

Table 1: Funções para verificação e resumo de dados unidimensionais.

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

6.4.3 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
## [10] floresta
## Levels: campo floresta

## $ funciona apenas para data frame
df$sp

## [1] "sp1"  "sp2"  "sp3"  "sp4"  "sp5"  "sp6"  "sp7"  "sp8"  "sp9"  "sp10"
## Operação de vetores
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 seguinte:

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

6.4.4 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)

## Warning in log(-1): NaNs produced
## [1] NaN
## Limite matemático
1/0

## [1] Inf
## Número grande
10^310

## [1] Inf
## Objeto nulo
nulo <- NULL
nulo

```

```
## NULL
```

6.4.5 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()
```

6.4.6 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** 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 pinguins no Palmer Station, Antarctica [Palmer Penguins](#). 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”. Em seguida, 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 .csv e importar a tabela de dados.

```

## Download
download.file(url = "https://github.com/allisonhorst/palmerpenguins/blob/main/inst/extdata/penguins_raw.csv",
              destfile = "penguins_raw.csv", method = "curl")

```

Agora podemos importar a tabela de dados com a função `read.csv()`, atribuindo ao objeto `penguins_raw`.

```

## Importar a tabela
penguins_raw <- read.csv("dados/tabelas/penguins_raw.csv")

```

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 ;, usariammos a função `read.csv2()`.

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

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

```

## Importar os dados pelo pacote palmerpenguins
penguins_raw <- palmerpenguins::penguins_raw
head(penguins_raw)

```

```

## # A tibble: 6 x 17
##   studyName `Sample Number` Species           Region Island Stage `Individual ID`
##   <chr>      <dbl> <chr>             <chr> <chr> <chr> <chr>
## 1 PAL0708     1 Adelie Penguin (Pygoscelis adeliae) Anvers Torgeby Adul~ N1A1
## 2 PAL0708     2 Adelie Penguin (Pygoscelis adeliae) Anvers Torgeby Adul~ N1A2
## 3 PAL0708     3 Adelie Penguin (Pygoscelis adeliae) Anvers Torgeby Adul~ N2A1
## 4 PAL0708     4 Adelie Penguin (Pygoscelis adeliae) Anvers Torgeby Adul~ N2A2
## 5 PAL0708     5 Adelie Penguin (Pygoscelis adeliae) Anvers Torgeby Adul~ N3A1
## 6 PAL0708     6 Adelie Penguin (Pygoscelis adeliae) Anvers Torgeby Adul~ N3A2
## # i 10 more variables: `Clutch Completion` <chr>, `Date Egg` <date>,
## #   `Culmen Length (mm)` <dbl>, `Culmen Depth (mm)` <dbl>,
## #   `Flipper Length (mm)` <dbl>, `Body Mass (g)` <dbl>, Sex <chr>,
## #   `Delta 15 N (o/oo)` <dbl>, `Delta 13 C (o/oo)` <dbl>, Comments <chr>

```

6.4.7 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 conferêcia desses dados. Dentre todas 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:

```

penguins <- palmerpenguins::penguins
## Primeiras linhas
head(penguins)

```

```

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex     year
##   <fct>    <fct>       <dbl>          <dbl>            <dbl>      <int> <fct> <int>
## 1 Adelie   Anvers        30.0           18.0            188.0      3750   Adul~ 2008
## 2 Adelie   Anvers        36.0           23.0            233.0      5100   Adul~ 2008
## 3 Adelie   Anvers        36.0           24.0            235.0      5100   Adul~ 2008
## 4 Adelie   Anvers        39.0           24.0            245.0      5100   Adul~ 2008
## 5 Adelie   Anvers        39.0           23.0            228.0      5100   Adul~ 2008
## 6 Adelie   Anvers        43.0           28.0            265.0      5100   Adul~ 2008

```

```

## 1 Adelie Torge-      39.1      18.7      181      3750 male   2007
## 2 Adelie Torge-      39.5      17.4      186      3800 fema~  2007
## 3 Adelie Torge-      40.3       18      195      3250 fema~  2007
## 4 Adelie Torge-      NA        NA       NA      NA <NA>  2007
## 5 Adelie Torge-      36.7      19.3      193      3450 fema~  2007
## 6 Adelie Torge-      39.3      20.6      190      3650 male   2007

## Últimas linhas
tail(pinguins)

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex     year
##   <fct>   <fct>      <dbl>          <dbl>            <dbl>        <int> <fct>    <int>
## 1 Chinst~ Dream       45.7           17             195        3650 fema~  2009
## 2 Chinst~ Dream       55.8           19.8            207        4000 male   2009
## 3 Chinst~ Dream       43.5           18.1            202        3400 fema~  2009
## 4 Chinst~ Dream       49.6           18.2            193        3775 male   2009
## 5 Chinst~ Dream       50.8           19              210        4100 male   2009
## 6 Chinst~ Dream       50.2           18.7            198        3775 fema~  2009

## Número de linhas e colunas
nrow(pinguins)

## [1] 344

ncol(pinguins)

## [1] 8

dim(pinguins)

## [1] 344   8

## Nome das linhas e colunas
rownames(pinguins)
colnames(pinguins)

## Estrutura dos dados
str(pinguins)

## Resumo dos dados
summary(pinguins)

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

## Remover as linhas com NAs
pinguins_na <- na.omit(pinguins)

```

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](#).

6.4.8 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(pinguins_na, "pinguins_na.csv",
           row.names = FALSE, quote = FALSE)

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

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

6.5 Para se aprofundar

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

6.5.1 Livros

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

6.5.2 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 \(very\) short introduction 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](#)

6.6 Exercícios

4.1 Use o R para verificar o resultado da operação $7 + 7 \div 7 + 7 \times 7 - 7$.

4.2 Verifique através do R se $3x2^3$ é maior que $2x3^2$.

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 30 locais de amostragem, 15 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 300 valores aleatórios entre 0 e 10. A matriz deve conter 30 linhas e ser disposta por colunas.

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

- `id: 1:30`
- `sp: sp01, sp02, ..., sp29, sp30`
- `ab: 30 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`. Use a função `set.seed()` para fixar a amostragem.

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.](#)

7 Tidyverse

Conteúdo copiado, com pequenas alterações e atualizações de Capítulo 5 Tidyverse, do livro Análises Ecológicas no R . O livro está licenciado sob a licença BY-NC-ND 4.0 : <https://creativecommons.org/licenses/by-nc-nd/4.0/>.

Pré-requisitos do capítulo

Pacotes e dados que serão utilizados neste capítulo.

```
## Pacotes
library(tidyverse)
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
```

7.1 Contextualização

Como todo idioma, a linguagem R tem “dialetos” e “sotaques” diversos. Ademais, 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 linguagens de programação interpretadas como como R R como as principais linguagens frente a esses novos desafios. Linguagens de programação de código aberto são uma ótima escolha para diversos propósitos, pois são gratuitas, possuem grandes comunidades de desenvolvedores, e são relativamente fáceis de aprender e aplicar.

As 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 pacote *ggplot2* (Wickham 2016) para a composição de gráficos no R, baseado na gramática de gráficos (Wilkinson and 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.

A partir dessas ideias, o dialeto *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) . O principal objetivo do *tidyverse* é aproximar a “linguagem R” 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 and Grolemund 2017), além do [Tidyverse](#) que possui muito mais informações.

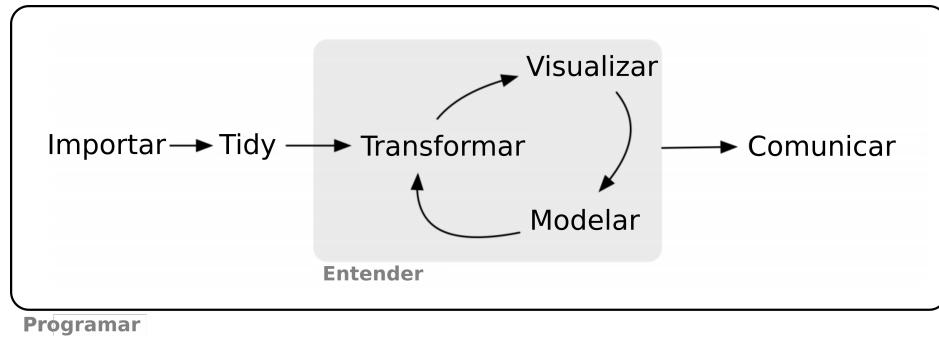


Figura 7.1: Modelo das ferramentas necessárias em um projeto típico de ciência de dados: importar, organizar, entender (transformar, visualizar, modelar) e comunicar, envolto à essas ferramentas está a programação.

7.2 *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 exportar)
- `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"          "conflicted"     "cli"            "dbplyr"         "dplyr"
## [6] "dplyr"          "forcats"        "ggplot2"        "googledrive"   "googlesheets4"
## [11] "haven"          "hms"            "httr"           "jsonlite"      "lubridate"
## [16] "magrittr"       "modelr"         "pillar"         "purrr"         "ragg"
## [21] "readr"          "readxl"         "reprex"         "rlang"          "rstudioapi"
## [26] "rvest"          "stringr"        "tibble"         "tidyverse"      "xml2"
## [31] "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 forte-

mente 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 performance de ajustes. Por razões de espaço e especificidade, não entraremos em detalhes desses pacotes.

Um projeto de ciência de dados envolve uma sequência de passos:

1. Importar os dados
2. Organizar os dados
3. Entender os dados (transformar, visualizar, modelar)
4. Comunicar os resultados Nas próximas seções, veremos como cada um desses passos pode ser realizado com funções de pacotes específicos.

7.3 `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 a função `readxl::read_excel()`, para importar dados, atentado para o fato de podermos indicar a aba/planilha 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 se aprofundar no tema, recomendamos a leitura do Capítulo 11 [Data import](#) de Wickham & Grolemund (2017).

7.4 `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 depois de serem importados. 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 6, 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
```

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).

7.5 pipe: base (`|>`) e magrittr (`%>%`)

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.

Aqui usamos a versão nativo (`|>`), que vem na instalação, que é mais “simples”. O operador pipe original `%>%` vem do pacote `magrittr`. R 4.1.0 introduziu um operador de pipe nativo “`|>`”. O comportamento do pipe nativo é em geral o mesmo do pipe `%>%` fornecido pelo pacote `magrittr`. Ambos os operadores (`|>` e `%>%`) permitem “canalizar” um objeto para uma função ou expressão de chamada, permitindo assim expressar uma sequência de operações que transformam um objeto.

Felizmente, não há necessidade de se comprometer inteiramente com um pipe ou outro - você pode usar o pipe nativo na maioria dos casos, e usar o pipe `magrittr` quando realmente precisar de seus recursos especiais. O pipe 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.

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).

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

## com pipe
1:100 |>
  sum() |>
  sqrt()

## [1] 71.06335

## R Base - sem pipe
ve <- sum(sqrt(sin(log10(rpois(100, 10)))))

## Fixar amostragem
set.seed(42)

## com pipe
ve <- rpois(100, 10) |>
  log10() |>
  sin() |>
  sqrt() |>
  sum()

## [1] 91.27018

## Fixar amostragem
set.seed(42)

## com pipe
ve <- rpois(100, 10) |>
  log10() |>
  sin() |>
  sqrt() |>
  sum()

## [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. 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+.

7.6 tidy

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 7.2).

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

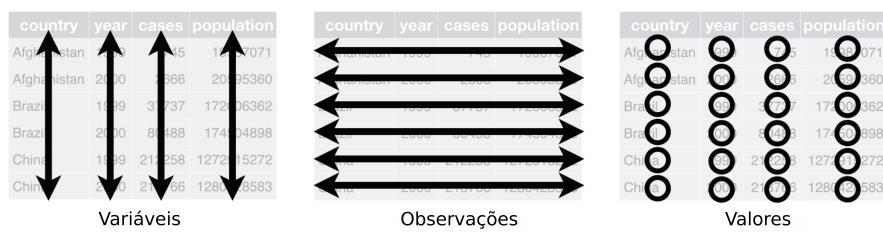


Figura 7.2: As três regras que tornam um conjunto de dados *tidy*.

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*)

7.6.1 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
```

7.6.2 glimpse()

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

```
## Visualizar os dados
penguins_raw
```

```
## # A tibble: 344 x 17
##   studyName `Sample Number` Species           Region Island Stage `Individual ID`
##   <chr>          <dbl> <chr>             <chr>  <chr> <chr> <chr>
## 1 PAL0708          1 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N1A1
## 2 PAL0708          2 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N1A2
## 3 PAL0708          3 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N2A1
## 4 PAL0708          4 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N2A2
## 5 PAL0708          5 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N3A1
## 6 PAL0708          6 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N3A2
## 7 PAL0708          7 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N4A1
## 8 PAL0708          8 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N4A2
## 9 PAL0708          9 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N5A1
## 10 PAL0708         10 Adelie Penguin (Pygoscelis adeliae) Anvers Torge~ Adul~ N5A2
## # i 334 more rows
## # i 10 more variables: `Clutch Completion` <chr>, `Date Egg` <date>,
## #   `Culmen Length (mm)` <dbl>, `Culmen Depth (mm)` <dbl>,
## #   `Flipper Length (mm)` <dbl>, `Body Mass (g)` <dbl>, Sex <chr>,
## #   `Delta 15 N (o/oo)` <dbl>, `Delta 13 C (o/oo)` <dbl>, Comments <chr>
## Espiar os dados
dplyr::glimpse(penguins_raw)
```

```
## Rows: 344
## Columns: 17
## $ studyName          <chr> "PAL0708", "PAL0708", "PAL0708", "PAL0708", ~
## $ `Sample Number`    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ~
## $ Species            <chr> "Adelie Penguin (Pygoscelis adeliae)", "Adelie Penguin ~
## $ Region             <chr> "Anvers", "Anvers", "Anvers", "Anvers", "Anver~
## $ Island              <chr> "Torgersen", "Torgersen", "Torgersen", "Torgersen", "To~
## $ Stage               <chr> "Adult", "Adult", "Adult", "Adult", "Adult", "Adult", "Adult", "Ad~
## $ `Individual ID`    <chr> "N1A1", "N1A2", "N2A1", "N2A2", "N3A1", "N3A2", "N4A1", ~
## $ `Clutch Completion` <chr> "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "Yes", "No", "No", ~
## $ `Date Egg`          <date> 2007-11-11, 2007-11-11, 2007-11-16, 2007-11-16, 2007-1~
## $ `Culmen Length (mm)` <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, 42.~
## $ `Culmen Depth (mm)` <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, 20.~
## $ `Flipper Length (mm)` <dbl> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186, 1~
## $ `Body Mass (g)`     <dbl> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, 425~
## $ Sex                 <chr> "MALE", "FEMALE", "FEMALE", NA, "FEMALE", "MALE", "FEMA~
## $ `Delta 15 N (o/oo)` <dbl> NA, 8.94956, 8.36821, NA, 8.76651, 8.66496, 9.18718, 9.~
```

```
## $ `Delta 13 C (o/oo)`    <dbl> NA, -24.69454, -25.33302, NA, -25.32426, -25.29805, -25~  
## $ Comments                <chr> "Not enough blood for isotopes.", NA, NA, "Adult not sa~
```

7.6.3 unite()

Primeiramente, vamos exemplificar como juntar e separar colunas. Vamos utilizar a função `tidy::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 <- tidy::unite(data = penguins_raw,  
                                    col = "region_island",  
                                    Region:Island,  
                                    sep = ", ",  
                                    remove = FALSE)  
head(penguins_raw_unir[, c("Region", "Island", "region_island")])
```

```
## # A tibble: 6 x 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
```

7.6.4 separate()

De forma contrária, podemos utilizar as funções `tidy::separate()` e `tidy::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 <- tidy::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 x 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 <- tidyverse::separate_rows(data = penguins_raw,
  Stage,
  sep = " ", )
head(penguins_raw_separar_linhas[, c("studyName", "Sample Number", "Species",
  "Region", "Island", "Stage")])

## # A tibble: 6 x 6
##   studyName `Sample Number` Species      Region Island     Stage
##   <chr>          <dbl> <chr>          <chr> <chr>     <chr>
## 1 PAL0708           1 Adelie Penguin (Pygoscelis adeliae) Anvers Torgersen Adult
## 2 PAL0708           1 Adelie Penguin (Pygoscelis adeliae) Anvers Torgersen 1 Egg-
## 3 PAL0708           2 Adelie Penguin (Pygoscelis adeliae) Anvers Torgersen Adult
## 4 PAL0708           2 Adelie Penguin (Pygoscelis adeliae) Anvers Torgersen 1 Egg-
## 5 PAL0708           3 Adelie Penguin (Pygoscelis adeliae) Anvers Torgersen Adult
## 6 PAL0708           3 Adelie Penguin (Pygoscelis adeliae) Anvers Torgersen 1 Egg-

```

7.6.5 drop_na() e replace_na()

Valor *faltante* (NA) é um tipo especial de elemento que são discutidos no Capítulo 6 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 `tidyverse::drop_na()` e `tidyverse::replace_na()`, para retirar linhas e substituir esses valores, respectivamente.

```

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

## # A tibble: 6 x 17
##   studyName `Sample Number` Species      Region Island Stage `Individual ID`
##   <chr>          <dbl> <chr>          <chr> <chr> <chr> <chr>
## 1 PAL0708           7 Adelie Penguin (Pygos~ Anvers Torge~ Adul~ N4A1
## 2 PAL0708           8 Adelie Penguin (Pygos~ Anvers Torge~ Adul~ N4A2
## 3 PAL0708          29 Adelie Penguin (Pygos~ Anvers Biscoe Adul~ N18A1
## 4 PAL0708          30 Adelie Penguin (Pygos~ Anvers Biscoe Adul~ N18A2
## 5 PAL0708          39 Adelie Penguin (Pygos~ Anvers Dream Adul~ N25A1
## 6 PAL0809          69 Adelie Penguin (Pygos~ Anvers Torge~ Adul~ N32A1
## # i 10 more variables: `Clutch Completion` <chr>, `Date Egg` <date>,
## #   `Culmen Length (mm)` <dbl>, `Culmen Depth (mm)` <dbl>,
## #   `Flipper Length (mm)` <dbl>, `Body Mass (g)` <dbl>, Sex <chr>,
## #   `Delta 15 N (o/oo)` <dbl>, `Delta 13 C (o/oo)` <dbl>, Comments <chr>
## Remover linhas de colunas específicas com NAs
penguins_raw_colunas_na <- tidyverse::drop_na(data = penguins_raw,
  any_of("Comments"))
head(penguins_raw_colunas_na[, "Comments"])

## # A tibble: 6 x 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 <- tidyverse::replace_na(data = penguins_raw,
                                             list(Comments = "Unknown"))
head(penguins_raw_subs_na[, "Comments"])

## # A tibble: 6 x 1
##   Comments
##   <chr>
## 1 Not enough blood for isotopes.
## 2 Unknown
## 3 Unknown
## 4 Adult not sampled.
## 5 Unknown
## 6 Unknown

```

7.6.6 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 `tidyverse::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 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 x 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

## Pivatar para largo
penguins_raw_pivot_wider <- tidyverse::pivot_wider(data = penguins_raw_sel_col,
                                                     names_from = Species,
                                                     values_from = `Body Mass (g)`)

head(penguins_raw_pivot_wider)

## # A tibble: 6 x 4
##   `Sample Number` Adelie Penguin (Pygo~1 Gentoo penguin (Pygo~2 Chinstrap penguin (P~3
##   <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
## # i abbreviated names: 1: `Adelie Penguin (Pygoscelis adeliae)`, 
## #   2: `Gentoo penguin (Pygoscelis papua)`, 
## #   3: `Chinstrap penguin (Pygoscelis antarctica)` 

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 tidy::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)

## # A tibble: 6 x 6
##   `Sample Number` Species     `Culmen Length (mm)` `Culmen Depth (mm)`
##   <dbl> <chr>             <dbl>                  <dbl>
## 1 1     Adelie Penguin (Pygoscelis ~      39.1       18.7
## 2 2     Adelie Penguin (Pygoscelis ~      39.5       17.4
## 3 3     Adelie Penguin (Pygoscelis ~      40.3        18
## 4 4     Adelie Penguin (Pygoscelis ~      NA          NA
## 5 5     Adelie Penguin (Pygoscelis ~      36.7       19.3
## 6 6     Adelie Penguin (Pygoscelis ~      39.3       20.6
## # i 2 more variables: `Flipper Length (mm)` <dbl>, `Body Mass (g)` <dbl>

## Pivatar para largo
penguins_raw_pivot_longer <- tidy::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)

## # A tibble: 6 x 4
##   `Sample Number` Species     medidas      valores
##   <dbl> <chr>       <chr>        <dbl>
## 1 1     Adelie Penguin (Pygoscelis adeliae) Culmen Length (mm) 39.1
## 2 1     Adelie Penguin (Pygoscelis adeliae) Culmen Depth (mm) 18.7
## 3 1     Adelie Penguin (Pygoscelis adeliae) Flipper Length (mm) 181
## 4 1     Adelie Penguin (Pygoscelis adeliae) Body Mass (g)    3750
## 5 2     Adelie Penguin (Pygoscelis adeliae) Culmen Length (mm) 39.5
## 6 2     Adelie Penguin (Pygoscelis adeliae) Culmen Depth (mm) 17.4

```

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

7.7 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.

7.7.1 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

7.7.2 Sintaxe

As funções do `dplyr` podem seguir uma mesma sintaxe: o `tibble` será sempre o primeiro argumento dessas funções, seguido de um operador pipe (`|>`ou `%>%`) 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

7.7.3 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)
```

7.7.4 `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)

## # A tibble: 6 x 8
##   species island sex     year bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct> <fct> <int>      <dbl>        <dbl>          <int>       <int>
## 1 Adelie   Torgo- male    2007      39.1         18.7          181        3750
## 2 Adelie   Torgo- fema-   2007      39.5         17.4          186        3800
## 3 Adelie   Torgo- fema-   2007      40.3         18            195        3250
## 4 Adelie   Torgo- <NA>    2007       NA           NA            NA         NA
## 5 Adelie   Torgo- fema-   2007      36.7         19.3          193        3450
## 6 Adelie   Torgo- male    2007      39.3         20.6          190        3650

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

## # A tibble: 6 x 8
##   species island sex     year bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct> <fct> <int>      <dbl>        <dbl>          <int>       <int>
## 1 Adelie   Torgo- male    2007      39.1         18.7          181        3750
## 2 Adelie   Torgo- fema-   2007      39.5         17.4          186        3800
## 3 Adelie   Torgo- fema-   2007      40.3         18            195        3250
## 4 Adelie   Torgo- <NA>    2007       NA           NA            NA         NA
## 5 Adelie   Torgo- fema-   2007      36.7         19.3          193        3450
## 6 Adelie   Torgo- male    2007      39.3         20.6          190        3650
```

7.7.5 `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 x 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)

## # A tibble: 6 x 8
##   SPECIES ISLAND BILL_LENGTH_MM BILL_DEPTH_MM FLIPPER_LENGTH_MM BODY_MASS_G SEX   YEAR
##   <fct>   <fct>     <dbl>      <dbl>        <int>      <int> <fct> <int>
## 1 Adelie  Torge-    39.1       18.7         181      3750 male  2007
## 2 Adelie  Torge-    39.5       17.4         186      3800 fema- 2007
## 3 Adelie  Torge-    40.3       18           195      3250 fema- 2007
## 4 Adelie  Torge-    NA          NA           NA       NA <NA> 2007
## 5 Adelie  Torge-    36.7       19.3         193      3450 fema- 2007
## 6 Adelie  Torge-    39.3       20.6         190      3650 male  2007

```

7.7.6 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 (3a até 6a)
penguins_select_position <- penguins |>
  dplyr::select(3:6)
head(penguins_select_position)

## # A tibble: 6 x 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 x 2
##   bill_length_mm body_mass_g
##   <dbl>      <int>
## 1 39.1        3750
## 2 39.5        3800
## 3 40.3        3250
## 4 NA          NA
## 5 36.7        3450
## 6 39.3        3650

```

```

## 1      39.1      3750
## 2      39.5      3800
## 3      40.3      3250
## 4       NA        NA
## 5      36.7      3450
## 6      39.3      3650

## Selecionar colunas por padrão
penguins_select_contains <- penguins |>
  dplyr::select(contains("_mm"))
head(penguins_select_contains)

## # A tibble: 6 x 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

```

7.7.7 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
```

7.7.8 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)

## # A tibble: 6 x 9
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>          <dbl>          <dbl>            <int>      <int>
## 1 Adelie  Torgersen     39.1          18.7            181      3750
## 2 Adelie  Torgersen     39.5          17.4            186      3800
## 3 Adelie  Torgersen     40.3           18              195      3250
## 4 Adelie  Torgersen      NA            NA              NA        NA
## 5 Adelie  Torgersen     36.7          19.3            193      3450

```

```

## # 6 Adelie Torgersen      39.3       20.6        190      3650
## # i 3 more variables: body_mass_kg <dbl>, sex <fct>, year <int>
## # Modificar várias colunas
penguins_mutate_across <- penguins |>
  dplyr::mutate(across(where(is.factor), as.character))
head(penguins_mutate_across)

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <chr>    <chr>        <dbl>        <dbl>        <int>      <int> <chr> <int>
## 1 Adelie  Torge-         39.1        18.7        181      3750 male  2007
## 2 Adelie  Torge-         39.5        17.4        186      3800 fema~ 2007
## 3 Adelie  Torge-         40.3        18          195      3250 fema~ 2007
## 4 Adelie  Torge-         NA          NA          NA        NA <NA>  2007
## 5 Adelie  Torge-         36.7        19.3        193      3450 fema~ 2007
## 6 Adelie  Torge-         39.3        20.6        190      3650 male  2007

```

7.7.9 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)

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>    <fct>        <dbl>        <dbl>        <int>      <int> <fct> <int>
## 1 Chinst~ Dream          46.9        16.6        192      2700 fema~ 2008
## 2 Adelie  Biscoe         36.5        16.6        181      2850 fema~ 2008
## 3 Adelie  Biscoe         36.4        17.1        184      2850 fema~ 2008
## 4 Adelie  Biscoe         34.5        18.1        187      2900 fema~ 2008
## 5 Adelie  Dream          33.1        16.1        178      2900 fema~ 2008
## 6 Adelie  Torge-         38.6        17          188      2900 fema~ 2009

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>    <fct>        <dbl>        <dbl>        <int>      <int> <fct> <int>
## 1 Gentoo  Biscoe         49.2        15.2        221      6300 male  2007
## 2 Gentoo  Biscoe         59.6        17          230      6050 male  2007
## 3 Gentoo  Biscoe         51.1        16.3        220      6000 male  2008
## 4 Gentoo  Biscoe         48.8        16.2        222      6000 male  2009
## 5 Gentoo  Biscoe         45.2        16.4        223      5950 male  2008
## 6 Gentoo  Biscoe         49.8        15.9        229      5950 male  2009

```

```

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Gentoo  Biscoe     49.2         15.2           221       6300 male  2007
## 2 Gentoo  Biscoe     59.6         17              230       6050 male  2007
## 3 Gentoo  Biscoe     51.1         16.3            220       6000 male  2008
## 4 Gentoo  Biscoe     48.8         16.2            222       6000 male  2009
## 5 Gentoo  Biscoe     45.2         16.4            223       5950 male  2008
## 6 Gentoo  Biscoe     49.8         15.9            229       5950 male  2009

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie   Dream     32.1         15.5           188       3050 fema~ 2009
## 2 Adelie   Dream     33.1         16.1           178       2900 fema~ 2008
## 3 Adelie   Torge~    33.5          19              190       3600 fema~ 2008
## 4 Adelie   Dream     34              17.1           185       3400 fema~ 2008
## 5 Adelie   Torge~    34.1         18.1            193       3475 <NA>  2007
## 6 Adelie   Torge~    34.4         18.4            184       3325 fema~ 2007

```

7.7.10 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 no Capítulo 6, 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)

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie   Torge~    39.1         18.7           181       3750 male  2007
## 2 Adelie   Torge~    39.5         17.4           186       3800 fema~ 2007
## 3 Adelie   Torge~    40.3          18              195       3250 fema~ 2007
## 4 Adelie   Torge~     NA            NA             NA        NA <NA>  2007
## 5 Adelie   Torge~    36.7         19.3           193       3450 fema~ 2007
## 6 Adelie   Torge~    39.3         20.6           190       3650 male  2007

```

```

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie   Torgo-    39.5         17.4           186       3800 fema~  2007
## 2 Adelie   Torgo-    40.3         18             195       3250 fema~  2007
## 3 Adelie   Torgo-    36.7         19.3           193       3450 fema~  2007
## 4 Adelie   Torgo-    38.9         17.8           181       3625 fema~  2007
## 5 Adelie   Torgo-    41.1         17.6           182       3200 fema~  2007
## 6 Adelie   Torgo-    36.6         17.8           185       3700 fema~  2007

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie   Torgo-    39.5         17.4           186       3800 fema~  2007
## 2 Adelie   Torgo-    40.3         18             195       3250 fema~  2007
## 3 Adelie   Torgo-    36.7         19.3           193       3450 fema~  2007
## 4 Adelie   Torgo-    38.9         17.8           181       3625 fema~  2007
## 5 Adelie   Torgo-    41.1         17.6           182       3200 fema~  2007
## 6 Adelie   Torgo-    36.6         17.8           185       3700 fema~  2007

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie   Torgo-    39.1         18.7           181       3750 male   2007
## 2 Adelie   Torgo-    39.5         17.4           186       3800 fema~  2007
## 3 Adelie   Torgo-    40.3         18             195       3250 fema~  2007
## 4 Adelie   Torgo-    36.7         19.3           193       3450 fema~  2007
## 5 Adelie   Torgo-    39.3         20.6           190       3650 male   2007
## 6 Adelie   Torgo-    38.9         17.8           181       3625 fema~  2007

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie   Torgo-    39.1         18.7           181       3750 male   2007
## 2 Adelie   Torgo-    39.5         17.4           186       3800 fema~  2007

```

```

## 3 Adelie Torge-        40.3      18      195    3250 fema~  2007
## 4 Adelie Torge-        36.7      19.3     193    3450 fema~  2007
## 5 Adelie Torge-        39.3      20.6     190    3650 male   2007
## 6 Adelie Torge-        38.9      17.8     181    3625 fema~  2007

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>       <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie  Dream        35.7         18            202      3550 fema~  2008
## 2 Adelie  Dream        41.1         18.1           205      4300 male   2008
## 3 Adelie  Dream        40.8         18.9           208      4300 male   2008
## 4 Adelie  Biscoe       41            20             203      4725 male   2009
## 5 Adelie  Torge-        41.4         18.5           202      3875 male   2009
## 6 Adelie  Torge-        44.1         18             210      4000 male   2009

```

7.7.11 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 300 até final
penguins_slice <- penguins |>
  dplyr::slice(300:n())
head(penguins_slice)

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>       <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Chinst~ Dream        50.6         19.4           193      3800 male   2007
## 2 Chinst~ Dream        46.7         17.9           195      3300 fema~  2007
## 3 Chinst~ Dream        52            19             197      4150 male   2007
## 4 Chinst~ Dream        50.5         18.4           200      3400 fema~  2008
## 5 Chinst~ Dream        49.5         19             200      3800 male   2008
## 6 Chinst~ Dream        46.4         17.8           191      3700 fema~  2008

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

## # A tibble: 5 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>       <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie  Torge-        39.1         18.7           181      3750 male   2007
## 2 Adelie  Torge-        39.5         17.4           186      3800 fema~  2007
## 3 Adelie  Torge-        40.3         18             195      3250 fema~  2007
## 4 Adelie  Torge-        NA           NA              NA      NA <NA>   2007
## 5 Adelie  Torge-        36.7         19.3           193      3450 fema~  2007

```

```

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Gentoo  Biscoe     49.2         15.2           221       6300 male  2007
## 2 Gentoo  Biscoe     59.6         17              230       6050 male  2007
## 3 Gentoo  Biscoe     51.1         16.3           220       6000 male  2008
## 4 Gentoo  Biscoe     48.8         16.2           222       6000 male  2009
## 5 Gentoo  Biscoe     45.2         16.4           223       5950 male  2008
## 6 Gentoo  Biscoe     49.8         15.9           229       5950 male  2009

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

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie   Biscoe     41.3         21.1           195       4400 male  2008
## 2 Gentoo   Biscoe     44.5         15.7           217       4875 <NA>  2009
## 3 Adelie   Torge-     41.4         18.5           202       3875 male  2009
## 4 Adelie   Biscoe     37.6         17              185       3600 fema~ 2008
## 5 Adelie   Dream      36            17.9           190       3450 fema~ 2007
## 6 Adelie   Biscoe     35.7         16.9           185       3150 fema~ 2008

```

7.7.12 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 x 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)

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie  Torgo-       39.1         18.7          181      3750 male  2007
## 2 Adelie  Torgo-       39.5         17.4          186      3800 fema~ 2007
## 3 Adelie  Torgo-       40.3         18            195      3250 fema~ 2007
## 4 Adelie  Torgo-        NA           NA             NA      NA <NA>  2007
## 5 Adelie  Torgo-       36.7         19.3          193      3450 fema~ 2007
## 6 Adelie  Torgo-       39.3         20.6          190      3650 male  2007

## 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)

## # A tibble: 6 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <fct>     <dbl>        <dbl>          <int>      <int> <fct> <int>
## 1 Adelie  Torgo-       39.1         18.7          181      3750 male  2007
## 2 Adelie  Torgo-       39.5         17.4          186      3800 fema~ 2007
## 3 Adelie  Torgo-       40.3         18            195      3250 fema~ 2007
## 4 Adelie  Torgo-        NA           NA             NA      NA <NA>  2007
## 5 Adelie  Torgo-       36.7         19.3          193      3450 fema~ 2007
## 6 Adelie  Torgo-       39.3         20.6          190      3650 male  2007

```

7.7.13 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 x 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 x 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

```

7.7.14 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)

## # A tibble: 6 x 8
## # Groups:   species [1]
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex     year
##   <fct>   <fct>        <dbl>        <dbl>        <int>      <int> <fct> <int>
## 1 Adelie   Torge-       39.1        18.7        181       3750 male   2007
## 2 Adelie   Torge-       39.5        17.4        186       3800 fema~  2007
## 3 Adelie   Torge-       40.3        18          195       3250 fema~  2007
## 4 Adelie   Torge-       NA          NA          NA         NA <NA>  2007
## 5 Adelie   Torge-       36.7        19.3        193       3450 fema~  2007
## 6 Adelie   Torge-       39.3        20.6        190       3650 male   2007

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

## # A tibble: 6 x 8
## # Groups:   species, island, sex [3]
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex     year
##   <fct>   <fct>        <dbl>        <dbl>        <int>      <int> <fct> <int>
## 1 Adelie   Torge-       39.1        18.7        181       3750 male   2007
## 2 Adelie   Torge-       39.5        17.4        186       3800 fema~  2007
## 3 Adelie   Torge-       40.3        18          195       3250 fema~  2007
## 4 Adelie   Torge-       NA          NA          NA         NA <NA>  2007
## 5 Adelie   Torge-       36.7        19.3        193       3450 fema~  2007
## 6 Adelie   Torge-       39.3        20.6        190       3650 male   2007

```

7.7.15 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 x 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 x 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.

```

7.7.16 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 6. 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)

## # A tibble: 6 x 9
##   id   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
##   <chr> <fct>   <fct>        <dbl>        <dbl>          <dbl>        <int> <fct>
## 1 1    Adelie  Torgo~     39.1        18.7           181        3750 male 
## 2 1    Adelie  Torgo~     39.5        17.4           186        3800 fema~
## 3 1    Adelie  Torgo~     40.3        18             195        3250 fema~
## 4 1    Adelie  Torgo~      NA          NA            NA          NA <NA>
## 5 1    Adelie  Torgo~     36.7        19.3           193        3450 fema~
## 6 2    Adelie  Biscoe     39.6        17.7           186        3500 fema~

## # i 1 more variable: year <int>

## Combinar as colunas
penguins_bind_cols <- dplyr::bind_cols(penguins_01, penguins_02, .name_repair = "unique")

```

```

## New names:
## * `species` -> `species...1`
## * `island` -> `island...2`
## * `bill_length_mm` -> `bill_length_mm...3`
## * `bill_depth_mm` -> `bill_depth_mm...4`
## * `flipper_length_mm` -> `flipper_length_mm...5`
## * `body_mass_g` -> `body_mass_g...6`
## * `sex` -> `sex...7`
## * `year` -> `year...8`
## * `species` -> `species...9`
## * `island` -> `island...10`
## * `bill_length_mm` -> `bill_length_mm...11`
## * `bill_depth_mm` -> `bill_depth_mm...12`
## * `flipper_length_mm` -> `flipper_length_mm...13`
## * `body_mass_g` -> `body_mass_g...14`
## * `sex` -> `sex...15`
## * `year` -> `year...16`

head(penguins_bind_cols)

## # A tibble: 5 x 16
##   species...1 island...2 bill_length_mm...3 bill_depth_mm...4 flipper_length_mm...5
##   <fct>      <fct>          <dbl>           <dbl>           <int>
## 1 Adelie     Torgersen       39.1            18.7            181
## 2 Adelie     Torgersen       39.5            17.4            186
## 3 Adelie     Torgersen       40.3             18              195
## 4 Adelie     Torgersen       NA               NA              NA
## 5 Adelie     Torgersen       36.7            19.3            193
## # i 11 more variables: body_mass_g...6 <int>, sex...7 <fct>, year...8 <int>,
## #   species...9 <fct>, island...10 <fct>, bill_length_mm...11 <dbl>,
## #   bill_depth_mm...12 <dbl>, flipper_length_mm...13 <int>, body_mass_g...14 <int>,
## #   sex...15 <fct>, year...16 <int>

```

7.7.17 *`_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 7.3.

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

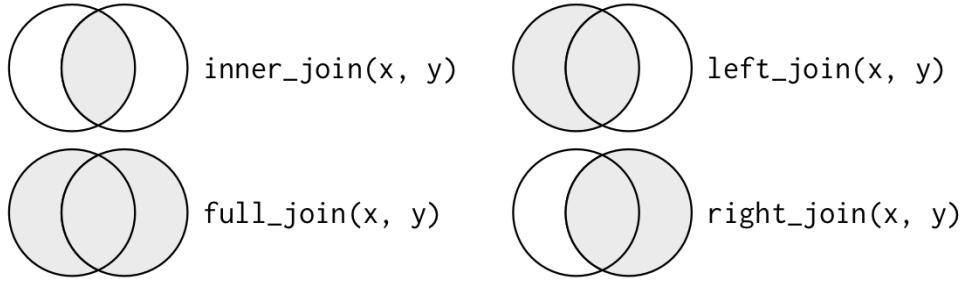


Figura 7.3: Diferentes tipos de joins, representados com um diagrama de Venn.

```

## 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)

## # A tibble: 6 x 10
##   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex   year
##   <fct>   <chr>      <dbl>        <dbl>          <int>       <int> <fct> <int>
## 1 Adelie   Torge-      39.1         18.7           181        3750 male   2007
## 2 Adelie   Torge-      39.5         17.4           186        3800 fema~  2007
## 3 Adelie   Torge-      40.3          18            195        3250 fema~  2007
## 4 Adelie   Torge-       NA            NA             NA        NA <NA>  2007
## 5 Adelie   Torge-      36.7         19.3           193        3450 fema~  2007
## 6 Adelie   Torge-      39.3         20.6           190        3650 male   2007
## # i 2 more variables: longitude <dbl>, latitude <dbl>

```

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.

7.7.18 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
- `intersect(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).

7.8 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 espaços 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"
## [21] "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"
## [21] "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).

7.9forcats

O pacote `forcats` fornece um conjunto de ferramentas úteis para facilitar a manipulação de fatores. Como dito no Capítulo 6, 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()
```

```
## [1] Adelie Penguin (Pygoscelis adeliae) Adelie Penguin (Pygoscelis adeliae)  
## [3] Adelie Penguin (Pygoscelis adeliae) Adelie Penguin (Pygoscelis adeliae)  
## [5] Adelie Penguin (Pygoscelis adeliae) Adelie Penguin (Pygoscelis adeliae)  
## 3 Levels: Adelie Penguin (Pygoscelis adeliae) ... Chinstrap penguin (Pygoscelis antarctica)
```

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  
## 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  
## 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  
## 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  
## Levels: Adelie Gentoo Chinstrap
```

```
## Ordem (decrescente) de frequência  
forcats::fct_infreq(penguins$species) |> head()
```

```
## [1] 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  
## 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).

7.10 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: `year` (ano), `month` (mês), `weak` (semana), `day` (dia), `hour` (hora), `minute` (minuto), e `second` (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] "2024-01-08"
```

```
## Extrair a data e tempo nesse instante
lubridate::now()
```

```
## [1] "2024-01-08 13:01:59 -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 x 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] "2024-01-08 UTC"

## Data-horário para data
lubridate::as_date(now())

```

```
## [1] "2024-01-08"
```

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
agora <- lubridate::now()
lubridate::year(agora)

```

```
## [1] 2024
lubridate::month(agora)
```

```
## [1] 1
lubridate::day(agora)
```

```
## [1] 8
lubridate::wday(agora)
```

```
## [1] 2
lubridate::second(agora)
```

```
## [1] 59.87338
```

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] "2024-01-09"

## Subtraindo durações de datas
lubridate::today() - lubridate::dyears(1)

## [1] "2023-01-07 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] "2024-03-18"

## Subtraindo datas
lubridate::today() - lubridate::weeks(10)

## [1] "2023-10-30"

## Criando datas recorrentes
lubridate::today() + lubridate::weeks(0:10)

## [1] "2024-01-08" "2024-01-15" "2024-01-22" "2024-01-29" "2024-02-05" "2024-02-12"
## [7] "2024-02-19" "2024-02-26" "2024-03-04" "2024-03-11" "2024-03-18"

```

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] "2024-01-08 UTC"
lubridate::int_length(r_intervalo)

## [1] 382060800
lubridate::int_flip(r_intervalo)

## [1] 2024-01-08 UTC--2011-11-30 UTC
lubridate::int_shift(r_intervalo, duration(days = 30))

## [1] 2011-12-30 UTC--2024-02-07 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] 12.10656
r_intervalo %/% lubridate::years()

## [1] 12
```

```
## Períodos dentro de um intervalo - dias e semanas  
filho_intervalo / lubridate::days()
```

```
## [1] 3756  
filho_intervalo / lubridate::weeks()
```

```
## [1] 536.5714
```

Ainda podemos fazer transformações dos dados para períodos e ter todas as unidades de data e tempo que o intervalo comprehende.

```
## Tempo total estudando R  
lubridate::as.period(r_intervalo)
```

```
## [1] "12y 1m 9d OH OM OS"  
## Idade do meu filho  
lubridate::as.period(filho_intervalo)
```

```
## [1] "10y 3m 13d OH OM OS"
```

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/Cayenne"
```

No R há uma listagem dos nomes dos fusos horários que podemos utilizar no argumento `tzone` para diferentes fusos horários.

```
## Verificar os fusos horários  
length(OlsonNames())
```

```
## [1] 596  
head(OlsonNames())
```

```
## [1] "Africa/Abidjan"      "Africa/Accra"        "Africa/Addis_Ababa"  "Africa/Algiers"  
## [5] "Africa/Asmara"       "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 = "GMT")
```

```
## [1] "2024-01-08 16:02:00 GMT"  
lubridate::with_tz(lubridate::now(), tzone = "Europe/Berlin")
```

```
## [1] "2024-01-08 17:02:00 CET"  
## Altera o fuso sem mudar a hora  
lubridate::force_tz(lubridate::now(), tzone = "GMT")
```

```
## [1] "2024-01-08 13:02:00 GMT"
```

Para se aprofundar no tema, recomendamos a leitura do Capítulo 16 Dates and times de Wickham & Grolemund (2017).

7.11 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))
purrr::map(x, sum)

```

```

## [[1]]
## [1] 15
##
## [[2]]
## [1] 16
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 10

```

Há diferentes tipos de retornos da família `purrr::map()`.

- `map()`: retorna uma lista
- `map_chr()`: retorna um vetor de strings
- `map_db1()`: 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"
```

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
##       43.92193        17.15117       200.91520      4201.75439
##   year
##       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).

7.12 Para se aprofundar

Listamos a seguir livros que recomendamos para seguir com sua aprendizagem em R e tidyverse.

7.12.1 Livros

Recomendamos aos (às) interessados(as) os livros: i) Oliveira e colaboradores (2018) [Ciéncia 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](#).

7.12.2 Links

[Ciéncia 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](#)

7.13 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 `tidyverse` 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.](#)

8 Primeiros passos com uma raster

Organize following: <https://jakubnowosad.com/posts/2020-05-26-intro-to-landscape-ecology/>

<https://youtu.be/dCTOGDnDuvM> at 28 minutes Attribute data operations - Vector attribute subsetting, aggregation and joins - Creating new vector attributes - Raster subsetting - Summarizing raster objects

Spatial data operations - Spatial subsetting - Topological relations - Spatial joining - zonal raster operations

Geometry operations - on vector - on raster - Vector-raster interactions

Coordinate reference systems - Understanding map projections - Reprojecting spatial data - Modifying map projections

Uma raster é um matriz de valores com coordenadas geográficos. Cada pixel de uma raster representa uma região geográfica, e o valor do pixel representa uma característica dessa região (mais sobre [dados raster](#)).

Em geral é necessário baixar alguns pacotes para que possamos fazer as nossas análises. Precisamos os seguintes pacotes, que deve estar instalado antes:

- [tidyverse](#),
- [terra](#),
- [sf](#),
- [mapview](#),
- [tmap](#).

Carregar pacotes:

```
library(tidyverse)
library(terra)
library(sf)
library(mapview)
library(tmap)
```

Inicialmente iremos gerar uma raster representando uma paisagem bem simples, de 6 por 6 pixels. Você já deve saber que pixel é a unidade básica de uma imagem (lembra da camera do seu celular, 10Mb ou algo assim?!). Vocês devem ter visto sobre pixels e resolução no mesmo em aulas de geoprocessamento. Aqui podemos tratar o pixel como a resolução. Vamos dizer que temos um pixel de 10 metros (res=10 no bloco de código), ou seja, uma quadrado de 10 por 10m, sendo essa, a menor unidade mapeável. Assim sendo, a resolução também tem ligação com escala cartográfica!

Vamos gerar e plotar uma paisagem simples em 4 passos. Primeiramente, a função `rast` cria um objeto do tipo raster. E depois a função `values` atribui valores, e na sequência vamos visualizar os valores com `plot` e `text`.

```
#Função "rast" gera a paisagem virtual (paisagem simulado)
pai_sim <- rast(ncols=6, nrows=6,
                 xmin=1, xmax=60,
                 ymin=1, ymax=60,
                 res=10)
#E essa atribui valores ("values") para os pixels criados acima
values(pai_sim) <- 1
plot(pai_sim) #Essa plota
text(pai_sim) #Essa coloca os valores dos pixels
```

8.0.1 Obter e carregar dados (raster)

Mais uma vez vamos aproveitar os dados de MapBiomas. Agora baixar arquivo raster com cobertura de terra no entorno dos rios em 2020, (formato “.tif”, tamanho 3.3 MB). Link: <https://github.com/darrenorris/>

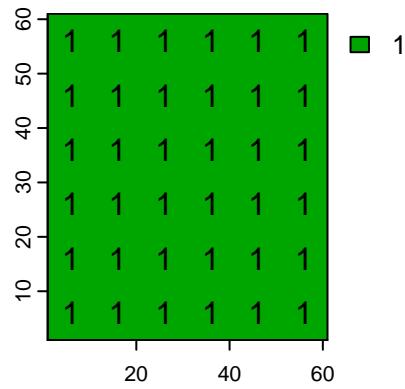


Figura 8.1: Paisagem simples de 36 pixels.

`gisdata/blob/master/inst/raster/mapbiomas_AP_utm_rio/utm_cover_AP_rio_2020.tif`. Lembrando-se de salvar o arquivo (“utm_cover_AP_rio_2020.tif”) em um local conhecido no seu computador. Agora avisar R sobre onde ficar o arquivo. O código abaixo vai abrir uma nova janela, e você deve buscar e selecionar o arquivo “utm_cover_AP_rio_2020.tif”:

```
meuSIGr <- file.choose()
```

O código abaixo vai carregar os dados e criar o objeto “mapbiomas_2020”.

```
mapbiomas_2020 <- rast(meuSIGr)
```

8.0.2 Reclassificação

Para simplificar nossa avaliação de escala, reclassificamos a camada mapbiomas_2020 em uma camada binária de floresta/não-floresta. Essa tarefa de geoprocessamento pode ser realizada anteriormente usando SIG ([QGIS](#)). Aqui vamos reclassificar as categorias de cobertura da terra (agrupando diferentes áreas de cobertura florestal tipos) usando alguns comandos genéricos do R para criar uma nova camada com a cobertura de floresta em toda a região de estudo. Para isso, criamos um mapa do mesmo resolução e extensão, e então podemos redefinir os valores do mapa. Neste caso, queremos agrupar a cobertura da terra categorias 3 e 4 (Formação Florestal e Formação Savânica, respectivamente).

```
# criar uma nova camada de floresta
floresta_2020 <- mapbiomas_2020
# Com valor de 0
values(floresta_2020) <- 0
# Atualizar categorias florestais agrupados com valor de 1
floresta_2020[mapbiomas_2020==3 | mapbiomas_2020==4] <- 1
## | ----- | ----- | ----- | ----- | ======
```

Vizualizar para verificar.

```
# Passo necessário para agilizar o processamento
floresta_2020_modal<-aggregate(floresta_2020, fact=10, fun="modal")
# Plot
tm_shape(floresta_2020_modal) +
  tm_raster(style = "cat",
             palette = c("0" = "#E974ED", "1" ="#129912"),
             legend.show = FALSE) +
  tm_add_legend(type = "fill", labels = c("não-floresta", "floresta"),
                col = c("#E974ED", "#129912"), title = "Classe") +
  tm_scale_bar(breaks = c(0, 25, 50), text.size = 1,
               text.color = "white", position=c("left", "bottom")) +
  tm_layout(legend.position = c("right","top"),legend.bg.color = "white")
```

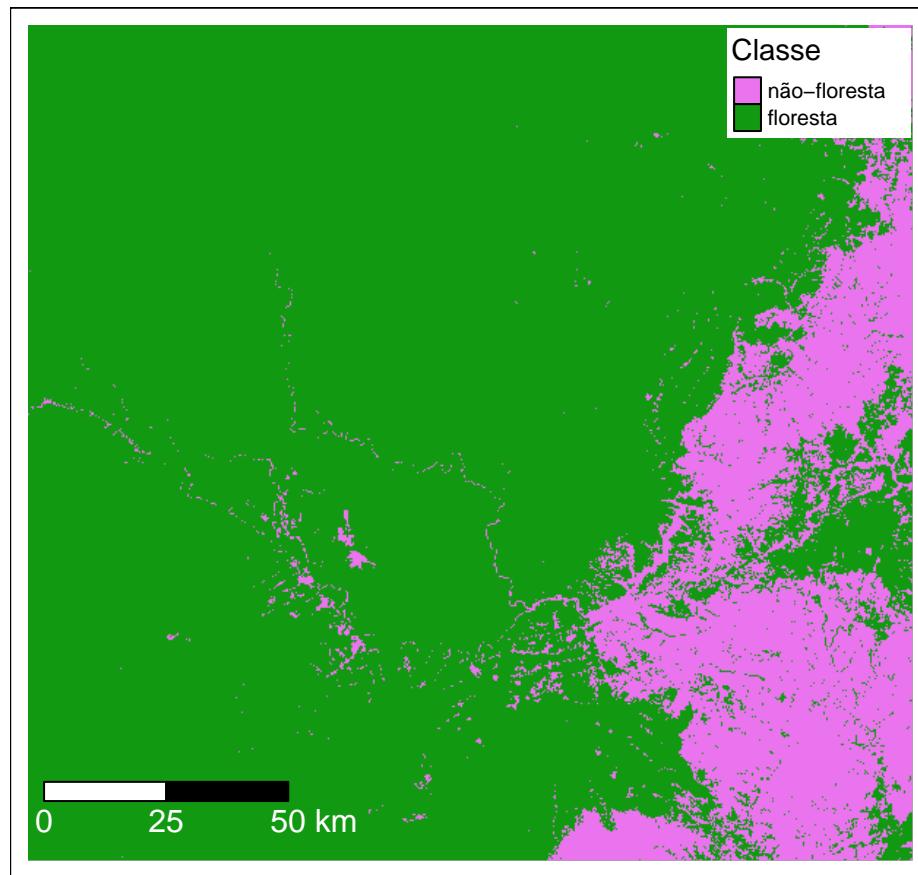


Figura 8.2: Floresta ao redor do Rio Araguari. MapBiomas 2020 reclassificado em floresta e não-floresta. Mostrando os pontos de amostragem (pontos amarelas) cada 5 quilômetros ao longo do rio.

9 Primeiros passos com vector

Em geral é necessário baixar alguns pacotes para que possamos fazer as nossas análises. Precisamos os seguintes pacotes, que deve estar instalado antes:

- `tidyverse`,
- `sf`,
- `mapview`,
- `tmap`.

Carregar pacotes:

```
library(tidyverse)
library(sf)
library(mapview)
library(tmap)
```

9.1 Obter e carregar dados (vectores)

Precisamos carregar os dados para rios e pontos de amostragem. Baixar arquivo (vector) com os dados (formato “GPKG”, tamanho 54.9 MB). Mais sobre [dados vetoriais](#). O formato aberto [GeoPackage](#) é um contêiner que permite armazenar dados SIG (feições/camadas) em um único arquivo. Por exemplo, um arquivo GeoPackage pode conter vários dados (dados vetoriais e raster) em diferentes sistemas de coordenadas. Todos esses recursos permitem que você compartilhe dados facilmente e evite a duplicação de arquivos.

Baixar o arquivo Link: <https://github.com/darrennorris/gisdata/blob/master/inst/vector/rivers.gpkg> . Lembrando-se de salvar o arquivo (“rivers.gpkg”) em um local conhecido no seu computador.

O formato “GPKG” é diferente de “tif” (raster), o processo de importação é, portanto, diferente. Primeira, avisar R sobre onde ficar o arquivo. O código abaixo vai abrir uma nova janela, e você deve buscar e selecionar o arquivo “rivers.GPKG”:

```
meuSIG <- file.choose()
```

Agora vamos olhar o que tem no arquivo. Depois que vocês rodar o código `st_layers(meuSIG)`, o resultado mostra que o arquivo rivers.GPKG inclui camadas diferentes com pontos (“Point”), linhas (“Line String”) e polígonos (“Polygon”). Além disso, a coluna “crs_name” mostrar que a sistema de coordenadas é geográfica (WGS84, (EPSG: 4326)

<https://epsg.io/4326>

, e é diferente do arquivo raster:

```
sf::st_layers(meuSIG)
```

```
## Driver: GPKG
## Available layers:
##   layer_name geometry_type features fields crs_name
## 1 centerline   Line String      52     15 WGS 84
## 2 forestloss    Point          276086    12 WGS 84
## 3 canalpoly    Polygon         3       6 WGS 84
## 4 extentpoly50km Polygon         1       0 WGS 84
## 5 midpoints    Point          52      17 WGS 84
## 6 midpoints_hansen Point          52      37 WGS 84
## 7 cachoeira_caldeirao Point         1       2 WGS 84
## 8 porto_grande Point          1       1 WGS 84
## 9 icmbio_base   Point          1       1 WGS 84
## 10 direct_affect Polygon         1       2 WGS 84
## 11 midpoints_hansen_distances Point         52      43 WGS 84
```

```
## 12      midpoints_hansen_ffr      Point      52      82      WGS 84
## 13      midpoints_hansen_ffril    Point      52      91      WGS 84
## 14      direct_affect_line     Line String      1       2      WGS 84
```

Nós só precisamos de duas dessas camadas. O código abaixo vai carregar as camadas que precisamos e criar os objetos “rsm” e “rsl”. Assim, agora temos dados com: pontos cada 5 km ao longo os rios (“rsm”) e a linha central de rios (“rsl”).

```
# pontos cada 5 km
rsm <- sf::st_read(meuSIG, layer = "midpoints")

## Reading layer `midpoints` from data source
##   `C:\Users\user\Documents\Articles\gis_layers\gisdata\inst\vector\rivers.gpkg'
##   using driver `GPKG'
## Simple feature collection with 52 features and 17 fields
## Geometry type: POINT
## Dimension:      XY
## Bounding box:  xmin: -52.01259 ymin: 0.7175827 xmax: -51.29688 ymax: 1.330365
## Geodetic CRS:  WGS 84

# linha central de rios
rsl <- sf::st_read(meuSIG, layer = "centerline")

## Reading layer `centerline` from data source
##   `C:\Users\user\Documents\Articles\gis_layers\gisdata\inst\vector\rivers.gpkg'
##   using driver `GPKG'
## Simple feature collection with 52 features and 15 fields
## Geometry type: LINESTRING
## Dimension:      XY
## Bounding box:  xmin: -52.01443 ymin: 0.7094595 xmax: -51.2924 ymax: 1.352094
## Geodetic CRS:  WGS 84
```

9.2 Visualizar os arquivos (camadas vector)

Visualizar para verificar. Mapa com linha central e pontos de rios em trechos de 5km.

```
ggplot(rsl) +  
  geom_sf(aes(color=rio)) +  
  geom_sf(data = rsm, shape=21, aes(fill=zone))
```

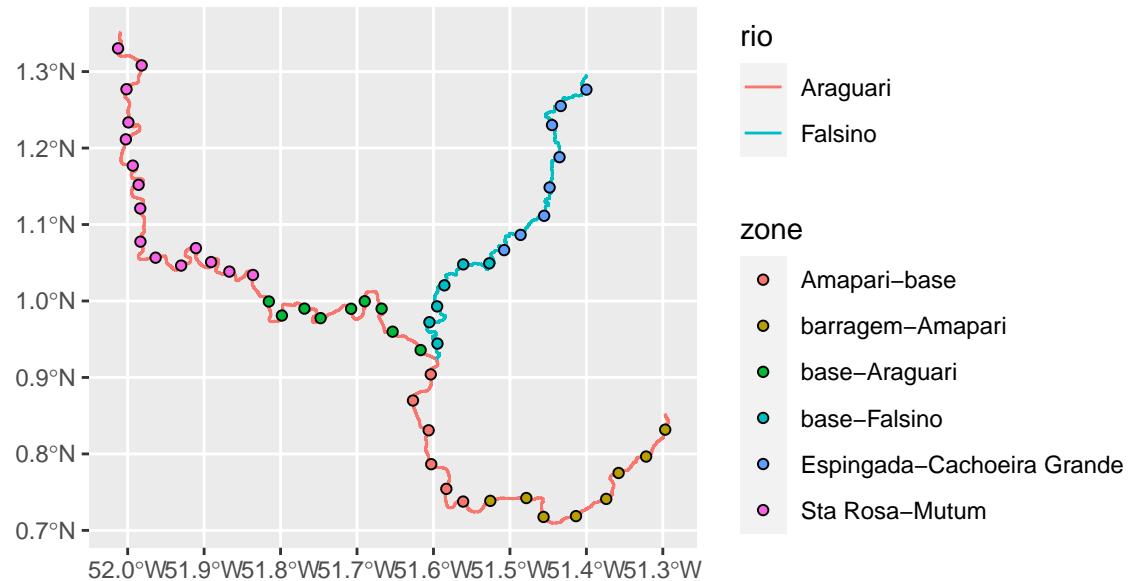


Figura 9.1: Pontos ao longo dos rios a montante das hidrelétricas no Rio Araguari.

Mapa interativo (funcione somente com internet). Mostrando agora com fundo de mapas “base” (OpenStreetMap/ESRI etc)

```
#  
mapview(rsl, zcol = "rio")
```

Em andamento

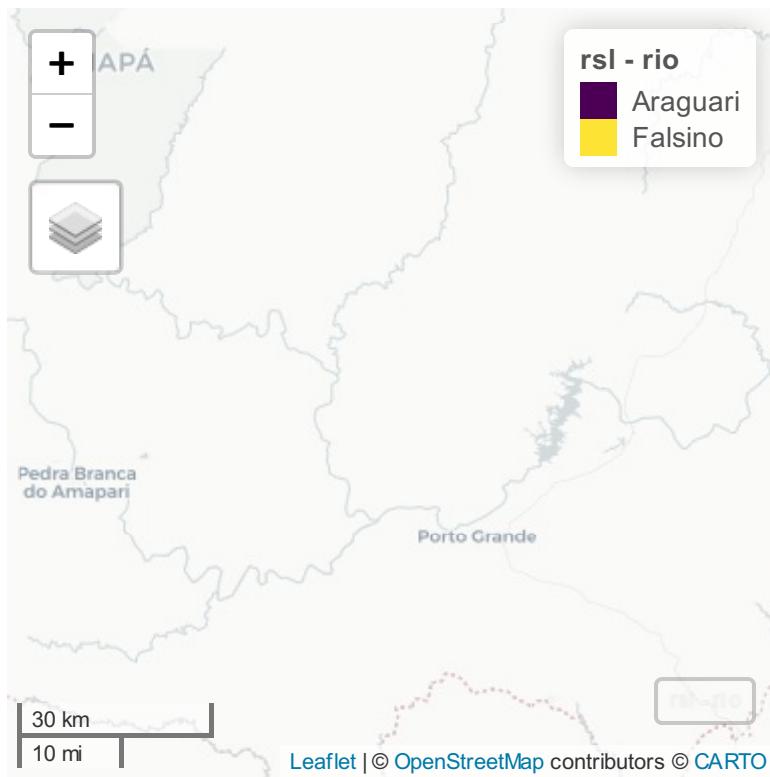


Figura 9.2: Linhas dos rios a montante das hidrelétricas no Rio Araguari.

Part V Encerramento

Bibliografia

- Aue, Birgit, Klemens Ekschmitt, Stefan Hotes, and Volkmar Wolters. 2012. "Distance Weighting Avoids Erroneous Scale Effects in Species-Habitat Models." *Methods in Ecology and Evolution* 3 (1): 102–11. <https://doi.org/10.1111/j.2041-210X.2011.00130.x>.
- Bullock, James M., Laura Mallada González, Riin Tamme, Lars Götzenberger, Steven M. White, Meelis Pärtel, and Danny A. P. Hooftman. 2017. "A Synthesis of Empirical Plant Dispersal Kernels." *Journal of Ecology* 105 (1): 6–19. <https://doi.org/10.1111/1365-2745.12666>.
- Burnham, K. A., and D. R. Anderson, eds. 2002. *Model Selection and Multimodel Inference*. Springer. <https://doi.org/https://doi.org/10.1007/b97636>.
- Chandler, Richard, and Jeffrey Hepinstall-Cymerman. 2016. "Estimating the Spatial Scales of Landscape Effects on Abundance." *Landscape Ecology* 31: 1383–94. <https://doi.org/10.1007/s10980-016-0380-z>.
- Dátillo, Wesley, André Luis Regolin, Fernanda Baena-Díaz, and Danilo Boscolo. 2023. "Spatial Scaling Involving the Complexity of Biotic Interactions: Integrating Concepts, Current Status, and Future Perspectives." *Current Landscape Ecology Reports*, 1–12. <https://doi.org/10.1007/s40823-023-00090-1>.
- Fletcher, R., and M.-J. Fortin. 2018. *Spatial ecology and conservation modeling: applications with r*. Cham: Springer International Publishing. <https://doi.org/10.1007/978-3-030-01989-1>.
- Galán-Acedo, Carmen, Víctor Arroyo-Rodríguez, Alejandro Estrada, and Gabriel Ramos-Fernández. 2018. "Drivers of the Spatial Scale That Best Predict Primate Responses to Landscape Structure." *Ecography* 41 (12): 2027–37. <https://doi.org/10.1111/ecog.03632>.
- Gehlke, Charles E, and Katherine Biehl. 1934. "Certain Effects of Grouping Upon the Size of the Correlation Coefficient in Census Tract Material." *Journal of the American Statistical Association* 29 (185A): 169–70.
- Hesselbarth, Maximilian H. K., Marco Sciaiani, Kimberly A. With, Kerstin Wiegand, and Jakub Nowosad. 2019. "LandscapeMetrics: An Open-Source r Tool to Calculate Landscape Metrics." *Ecography* 42: 1648–57. <https://doi.org/10.1111/ecog.04617>.
- Holmes, S., and W. Huber. 2019. *Modern Statistics for Modern Biology*. Cambridge, United Kingdom: Cambridge university press.
- Irizarry, R. A. 2017. *Data Analysis for the Life Sciences with R*. Boca Raton: CRC Press, Taylor & Francis Group.
- Irizarry, Rafael A. 2019. *Introduction to Data Science: Data Analysis and Prediction Algorithms with R*. 1^a edição. Chapman; Hall/CRC.
- Ismay, C., and A. Y.-S. Kim. 2020. *Statistical Inference via Data Science: A ModernDive into R and the Tidyverse*. Chapman & Hall/CRC the R Series. Boca Raton: CRC Press / Taylor & Francis Group.
- Jackson, Heather Bird, and Lenore Fahrig. 2015. "Are Ecologists Conducting Research at the Optimal Scale?" *Global Ecology and Biogeography* 24 (1): 52–63. <https://doi.org/10.1111/geb.12233>.
- Jelinski, Dennis E, and Jianguo Wu. 1996. "The Modifiable Areal Unit Problem and Implications for Landscape Ecology." *Landscape Ecology* 11: 129–40. <https://doi.org/10.1007/BF02447512>.
- Martin, Amanda E., and Lenore Fahrig. 2012. "Measuring and Selecting Scales of Effect for Landscape Predictors in Species–Habitat Models." *Ecological Applications* 22 (8): 2277–92. <https://doi.org/10.1890/11-2224.1>.
- McGarigal, Kevin, Ho Yi Wan, Kathy A Zeller, Brad C Timm, and Samuel A Cushman. 2016. "Multi-Scale Habitat Selection Modeling: A Review and Outlook." *Landscape Ecology* 31: 1161–75. <https://doi.org/10.1007/s10980-016-0374-x>.
- Michalski, Fernanda, Ricardo Luiz Pires Boulhosa, Yuri Nascimento do Nascimento, and Darren Norris. 2020. "Rural Wage-Earners' Attitudes Towards Diverse Wildlife Groups Differ Between Tropical Ecoregions: Implications for Forest and Savanna Conservation in the Brazilian Amazon." *Tropical Conservation Science* 13: 1940082920971747. <https://doi.org/10.1177/1940082920971747>.
- Michalski, Fernanda, and Carlos A. Peres. 2005. "Anthropogenic Determinants of Primate and Carnivore Local Extinctions in a Fragmented Forest Landscape of Southern Amazonia." *Biological Conservation* 124 (3): 383–96. <https://doi.org/10.1016/j.biocon.2005.01.045>.
- . 2007. "Disturbance-Mediated Mammal Persistence and Abundance-Area Relationships in Amazonian Forest Fragments." *Conservation Biology* 21 (6): 1626–40. <https://doi.org/10.1111/j.1523-1739.2007.00797.x>.
- Miguet, Paul, Lenore Fahrig, and Claire Lavigne. 2017. "How to Quantify a Distance-Dependent Landscape Effect on a Biological Response." *Methods in Ecology and Evolution* 8 (12): 1717–24. <https://doi.org/10.1111/mec.13227>.

1111/2041-210X.12830.

- Moll, Remington J., Jonathon D. Cepek, Patrick D. Lorch, Patricia M. Dennis, Terry Robison, and Robert A. Montgomery. 2020. "At What Spatial Scale(s) Do Mammals Respond to Urbanization?" *Ecography* 43 (2): 171–83. <https://doi.org/10.1111/ecog.04762>.
- Oliveira, P. F., S. Guerra, and R. McDonnell. 2018. *Ciência de Dados Com R - Introdução*.
- Pearson, Scott M. 1993. "The Spatial Extent and Relative Influence of Landscape-Level Factors on Wintering Bird Populations." *Landscape Ecology* 8 (1): 3–18. <https://doi.org/10.1007/BF00129863>.
- San-José, Miriam, Victor Arroyo-Rodríguez, Pedro Jordano, Jorge A Meave, and Miguel Martínez-Ramos. 2019. "The Scale of Landscape Effect on Seed Dispersal Depends on Both Response Variables and Landscape Predictor." *Landscape Ecology* 34: 1069–80. <https://doi.org/10.1007/s10980-019-00821-y>.
- Wickham, H. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10). <https://doi.org/10.18637/jss.v059.i10>.
- . 2016. *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.
- Wickham, H., M. Averick, J. Bryan, W. Chang, L. McGowan, R. François, G. Grolemund, et al. 2019. "Welcome to the Tidyverse." *Journal of Open Source Software* 4 (43): 1686. <https://doi.org/10.21105/joss.01686>.
- Wickham, H., and G. Grolemund. 2017. *R for data science: import, tidy, transform, visualize, and model data*. First edit. Sebastopol, CA: O'Reilly.
- Wilkinson, L., and G. Wills. 2005. *The grammar of graphics*. 2nd ed. Statistics and Computing. New York: Springer.
- Williams, G. J. 2018. *The Essentials of Data Science: Knowledge Discovery Using R*. Boca Raton: Taylor & Francis, CRC Press.
- Wu, Jianguo. 2004. "Effects of Changing Scale on Landscape Pattern Analysis: Scaling Relations." *Landscape Ecology* 19: 125–38. <https://doi.org/10.1023/B:LAND.0000021711.40074.ae>.
- Wu, Jianguo, and Harbin Li. 2006. "Concepts of Scale and Scaling." In *Scaling and Uncertainty Analysis in Ecology*, edited by Jianguo Wu, K. Bruce Jones, Harbin Li, and Orie L. Loucks, 3–15. Dordrecht: Springer Netherlands. https://doi.org/10.1007/1-4020-4663-4_1.
- Zumel, N., and J. Mount. 2014. *Practical Data Science with R*. Shelter Island, NY: Manning Publications Co.

Glossário

Conteúdo copiado, com pequenas alterações e atualizações de [Glossário](#), do livro [Análises Ecológicas no R](#).

- **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, agragando 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
- **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
- **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
- **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
- **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 de quatro arquivos: .shp (feição), .dbf (tabela de atributos), .shx (ligação entre .shp e .dbf) e .prj (projecção)
- **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
- **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
- **Forrageadoras de lugar central**: forrageiro viajando de uma base inicial (“casa”/“abrigos) para um local distante, em vez de simplesmente passar por uma área ou viajar aleatoriamente.
- **Funções**: classe de objetos que possui códigos preparados para realizar uma tarefa específica de modo simples, realizando operações em argumentos
- **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
- **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
- **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
- **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
- **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
- **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 cor-

relacionadas

- **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
- **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
- **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 (|> e %>%)**: 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) grade 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
- **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 (X_n) 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), quanto o Datum *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
- **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 ângulos e 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*
- **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
- **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
- **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

A Annexo 1

Exemplos de código R usado para - baixar Mapbiomas, - recortar para uma região de interesse - plotar com legenda

```
library(plyr)
library(tidyverse)
library(terra)
library(sf)
library(readxl)
library(tmap)
```

A.1 Download mapbiomas cover

Site: <https://brasil.mapbiomas.org/colecoes-mapbiomas/>. Any zeros are NA (NODATA). <http://forum.mapbiomas.ecostage.com.br/t/pixels-com-valor-zero/170/5> . And “27” “Não observado”. No exemplo os arquivos vai ficar no working directory.

```
#Get mapbiomas data
#
url_text <- "https://storage.googleapis.com/mapbiomas-public/initiatives/brasil/collection_8/lclu/cover"
layer_names <- paste("brasil_coverage_", 1985:2022, sep="")
file_type <- ".tif"
filenames_mapbioma_8 <- paste(layer_names, file_type, sep="")
urlnames_mapbioma_8 <- paste(url_text, layer_names, file_type, sep="")

#set timeout to 20 minutes as big files and teeny tiny internet connection
options(timeout = max((60*20), getOption("timeout")))
# test with one year - 1985
res <- utils::download.file(url=urlnames_mapbioma_8[1],
                             destfile=filenames_mapbioma_8[1],
                             method="auto", quiet = FALSE, mode = "wb",
                             cacheOK = TRUE)
#make list
dflist <- data.frame(layer_names = layer_names,
                      urls = urlnames_mapbioma_8,
```

```

        filenames = filenames_mapbioma_8)
alist <- split(dflist, f = dflist$layer_names)

#make function to automatically process list elements
get_mapbiomas <- function(x){
res <- utils::download.file(url=x$url,
                           destfile=x$filenames,
                           method="auto", quiet = FALSE, mode = "wb",
                           cacheOK = TRUE)
res
}

#run function in blocks
lapply(alist[2], get_mapbiomas)
lapply(alist[3:5], get_mapbiomas)
lapply(alist[6:8], get_mapbiomas)
lapply(alist[9:20], get_mapbiomas)
lapply(alist[21:30], get_mapbiomas)
lapply(alist[31:37], get_mapbiomas)

```

A.2 Crop mapbiomas cover

Identify all tif files in a folder.

```

#works one folder at a time
get_files <- function(folder_name = NA) {
  library(tidyverse)
  folder_location <- folder_name
  in_files <- list.files(folder_location,
                         pattern = ".tif", full.names = TRUE)
  data.frame(folder_id = folder_location, file_id = in_files) |>
    group_by(folder_id, file_id) |>
    dplyr::summarise(file_count = dplyr::n()) |>
    ungroup() -> df_muni_tif
  return(df_muni_tif)
}

infolder <- "E:/mapbiomas"
df_muni_tif <- get_files(folder_name = infolder)
#update
df_muni_tif |>
  dplyr::mutate(state_code = str_sub(folder_id, -2, -1),
                ) -> df_muni_tif
df_muni_tif$state_code <- "AP"

```

Area of interest - format extent for use by terra.

```

#extent from 50 km arouund river upstream of cachoeira caldeirao
meuSIG <- file.choose()
#"C:\\Users\\Darren\\Documents\\gisdata\\vector\\rivers.GPKG"
# "C:\\Users\\user\\Documents\\Articles\\gis_layers\\gisdata\\inst\\vector\\rivers.gpkg"
rsl <- st_read(meuSIG, layer = "centerline")
rsl_50km <- st_union(st_buffer(rsl, dist=50000))
myextent <- ext(vect(rsl_50km))

```

Now to crop and project to desired coordinate system (epsg:31976).

```
# This function reads in file, crops and projects to new coordinate system.
# Then exports result as Geotiff (.tif).
crop_proj <- function(x, state_id = NA,
                      sf_state = NA) {
  library(plyr)
  library(tidyverse)
  library(terra)
  library(sf)
  library(stringi)

  state_sigla = x$state_code
  rin <- x$file_id
  rbig <- terra::rast(rin)
  layer_name <- names(rbig)
  layer_year <- stri_sub(layer_name,-4,-1)
  out_name <- paste("mapbiomas_cover", layer_year, sep="_")

  e2 <- myextent

  #Crop
  rtest_mask <- crop(rbig, e2, snap="out")
  #rtest_mask <- rbig
  rm("rbig")

  #Project
  new_crs_utm <- "epsg:31976"
  rtest_mask <- project(rtest_mask, new_crs_utm, method = "near")
  names(rtest_mask) <- out_name

  #Export
  #folder <- paste(state_sigla, "_", "equalarea", sep = "")
  #folder_utm <- paste(state_sigla, "_", "utm", sep = "")

  #folder_utm_muninorte <- paste(state_sigla, "_", "utm_muni_norte", sep = "")
  #folder_utm_munimaza <- paste(state_sigla, "_", "utm_munis_maza_santana", sep = "")
  #folder_utm_cutias <- paste(state_sigla, "_", "utm_muni_cutias", sep = "")
  #folder_path <- paste("C:/Users/Darren/Documents/2022_Norris_gdp_deforestation/AmazonConservation/mapb
  #outfile <- paste("ea_cover_", state_sigla, "_",
  #                  layer_year, ".tif", sep = "")
  folder_utm_rio <- paste(state_sigla, "_", "utm_rio", sep = "")
  outfile_utm <- paste("utm_cover_AP_rio_", layer_year, ".tif", sep = "")
  f <- file.path(folder_utm_rio, outfile_utm)
  writeRaster(rtest_mask, f, datatype = "INT1U", NAflag=27,
             gdal=c("COMPRESS=DEFLATE"), overwrite = TRUE)

  #clear temporary files
  tmpFiles(current =TRUE, remove = TRUE)

  endtime <- Sys.time()
  textout <- paste(outfile_utm, ":", endtime, sep="")
  print(textout)
}
```

```
#run
plyr:::a_ply(df_muni_tif, .margins = 1,
             .fun = crop_proj)
```

A.3 Plot with legend

Combine all years into multi layer raster.

```
infolder_rio <- "E:/mapbiomas/AP_utm_rio"
df_tif_rio <- get_files(folder_name = infolder_rio)
r85a22 <- rast(df_tif_rio$file_id)
writeRaster(r85a22, "utm_cover_AP_rio_85a22.tif", datatype = "INT1U", NAflag=27,
            gdal=c("COMPRESS=DEFLATE"), overwrite = TRUE)
```

Legend.

```
mapbiomas8_legenda <- read.csv2("legend/Codigos-da-legenda-colecao-8.csv")
```

Plot.

```
class_vals <- mapbiomas8_legenda$Class_ID
class_color <- mapbiomas8_legenda$Color
names(class_color) <- mapbiomas8_legenda$Class_ID
#labels
my_label <- mapbiomas8_legenda$Descricao
names(my_label) <- mapbiomas8_legenda$Class_ID

# Plot one year to check
tm_shape(subset(r85a22n_ag, c(1, 2, 37,38))) +
  tm_raster(style = "cat", palette = class_color, labels = my_label, title="") +
  tm_scale_bar(breaks = c(0, 10), text.size = 1,
               position=c("right", "bottom"))

# Now all years together.
tm_shape(r85a22n_ag) +
  tm_raster(style = "cat", palette = class_color, labels = my_label, title="") +
  tm_scale_bar(breaks = c(0, 10), text.size = 1,
               position=c("right", "bottom")) +
  tm_facets(ncol=5) +
  tm_layout(legend.bg.color="white", legend.outside = TRUE,
            legend.outside.position = "right",
            panel.labels = c('1985', '1986', '1987', '1988', '1989', '1990',
                            '1991', '1992', '1993', '1994', '1995', '1996',
                            '1997', '1998',
                            '1999', '2000', '2001', '2002', '2003',
                            '2004', '2005', '2006', '2007',
                            '2008', '2009', '2010', '2011', '2012',
                            '2013', '2014', '2015',
                            '2016', '2017', '2018', '2019',
                            '2020', '2021', '2022')) -> figmap

png("figmap.png", width=6, height=9,
    units="in", res = 600)
figmap
invisible(dev.off())
```