Introdução à Estatística com R para Biólogos

Fernando Andrade

Sumário

Prefácio **4**

I Ferramentas Essenciais: Dominando o R **5**

1 Introdução **7**
1.1 O que são R e RStudio? 7
1.2 Instalação passo a passo 7
1.3 Navegando na interface do RStudio 8
1.4 O conceito de pacotes 9
1.4.1 Instalando e Carregando Pacotes Essenciais 9
1.5 Diretório de Trabalho e Projetos RStudio 10
1.6 R Básico 10
1.6.1 Operadores Matemáticos 10
1.6.2 Objetos e funções 11
1.6.3 Importanto dados 12
1.6.4 Vetores e *Data frames* 13
1.6.5 Fatores 14
1.6.6 Valores especiais 15
1.6.7 Pedindo ajuda 16

2 Tidyverse **17**
2.1 Manipulação de dados com o pacote `dplyr` 18
2.2 Visualização de Dados com `ggplot2` 30
2.2.1 Estatística descritiva 30
2.2.2 Tipos de gráficos 34
2.2.3 Técnicas avançadas de visualização e comunicação 40
2.2.4 Sub-gráficos com `facet_wrap()` 40

3 Exercícios **43**

II Fundamentos do Pensamento Estatístico **44**

4 Princípios de Probabilidade **45**

5 A Lógica da Inferência Estatística **46**

6 Delineamento de Experimentos **47**

III Modelagem Estatística de Dados Biológicos **48**

7 Modelos Lineares – A Base da Modelagem **49**

8 Entendendo os Modelos Mistos **50**

9 Modelos Lineares Mistos com `lme4` **51**

10 Modelos Lineares Generalizados Mistos **52**

11 Validação e Interpretação de Modelos Mistos **53**

IV Aplicações Práticas e Recursos **54**

Referências **55**

Prefácio

Essa apostila foi criada para ser um guia abrangente e conceitual para pesquisadores de mestrado e doutorado na área de Ciências Biológicas e Zootecnia, com um foco particular em estudos de aves. Reconhecendo que muitos biólogos possuem uma base sólida em suas áreas de especialidade mas podem não ter uma formação apropriada em programação e estatística teórica, este material busca preencher essa lacuna de maneira saudável. O objetivo não é apresentar um oceano de fórmulas matemáticas, mas sim construir uma compreensão intuitiva e rigorosa dos princípios estatísticos e de sua aplicação prática utilizando a linguagem de programação R, um forte e indispensável aliado nos tempos atuais para pesquisadores e cientistas de dados.

A estrutura da apostila foi projetada para seguir uma progressão lógica, começando com as habilidades fundamentais de programação e manipulação de dados em R, e avançando gradualmente para modelos estatísticos mais complexos que são frequentemente necessários para pesquisas biológicas, como os modelos lineares mistos e generalizados mistos. Cada capítulo combina explicações conceituais, exemplos práticos contextualizados na ornitologia e zootecnia, e blocos de códigos em R detalhadamente explicados.

O pressuposto é que a estatística não seja um obstáculo a ser superado, mas uma ferramenta poderosa para analisar criteriosamente os dados, planejar experimentos robustos e extrair conclusões válidas e significativas do trabalho de pesquisa. Ao final desta jornada, é esperado que o leitor estará equipado não apenas para analisar seus próprios dados com confiança, mas também para interpretar criticamente a literatura científica de sua área.

Zuur et al. (2007) e (ZUUR ET AL., 2007)

Parte I

Ferramentas Essenciais: Dominando o R

Nesta primeira parte, o foco residirá no entendimento das ferramentas computacionais essenciais que fundamentam qualquer análise de dados moderna. Antes de mergulhar nos conceitos estatísticos propriamente ditos, é necessário desenvolver uma fluência básica no ambiente de programação que será utilizado.

A abordagem que será adotada aqui prioriza a eficiência e a intuição, introduzindo o início do ecossistema `tidyverse`, um conjunto de pacotes R projetados para trabalhar em harmonia, tornando a manipulação, exploração e visualização de dados um processo mais humano, lógico e simples.

Ao dominar essa ferramenta, o leitor transformará tarefas árduas de preparação de dados em uma parte integrada e poderosa do processo de descoberta científica.

1 Introdução

Para começar essa jornada, o primeiro passo é configurar o ambiente de trabalho. Isso envolve a instalação de dois *softwares* distintos, mas que trabalham juntos: R e RStudio. Compreender o funcionamento de cada um e como eles se interagem é fundamental para as próximas etapas.

1.1 O que são R e RStudio?

É comum que iniciantes confundam R e RStudio, mas esta distinção é crucial para o processo.

- **R** é a linguagem de programação e o ambiente de software para computação estatística e gráficos. Pode-se pensar que é o "motor" que executa todos os cálculos, análises e gera os gráficos. Além de tudo, é um projeto de código aberto, gratuito e mantido por uma vasta comunidade de desenvolvedores e estatísticos ao redor do mundo.
- **RStudio** é um Ambiente de Desenvolvimento Integrado (IDE, do inglês *Integrated Development Environment*). Se o R é o motor do carro, o RStudio é o painel, o volante, e todo o interior que torna a condução do carro uma experiência agradável e gerenciável. O RStudio fornece uma interface gráfica e amigável que organiza o trabalho em R, facilitando a escrita de *scripts* (arquivos de códigos), a visualização de gráficos, o gerenciamento de pacotes (bibliotecas) e muito mais. Embora seja possível utilizar o R sem o RStudio, a utilização do RStudio é fortemente recomendada, pois deixa o processo de análise muito mais interativo e organizado.

1.2 Instalação passo a passo

A instalação adequada dos programas é um pré-requisito crucial. A ordem de instalação é importante: **R deve ser instalado antes do RStudio**.

1. Instalando o R:

- Acesse o site do *Comprehensive R Archive Network* (CRAN), que é o repositório oficial para o R e seus pacotes.
- Na página inicial, selecione o link de download para o seu sistema operacional (Linux, macOS ou Windows).
- Siga as instruções para baixar a versão mais recente ("base"). É crucial baixar a versão diretamente do CRAN, pois os gerenciadores de pacotes de alguns sistemas operacionais (como o `get-apt` do Ubuntu) podem fornecer versões desatualizadas.
- Execute o arquivo de instalação baixado e siga as instruções padrão, aceitando as configurações padrão.

2. Instalando o RStudio:

- Após a instalação do R, acesse o site da Posit e clique para baixar a versão gratuita do RStudio Desktop.
- Baixe o instalador apropriado para o seu sistema operacional.
- Execute o arquivo de instalação. O RStudio detectará automaticamente a instalação do R existente.

1.3 Navegando na interface do RStudio

Ao abrir o RStudio pela primeira vez, a interface se apresenta dividida em quatro painéis ou quadrantes principais, cada um com uma função específica:

1. **Editor de *scripts*** (Superior esquerdo): Este é o seu principal espaço de trabalho. Aqui, você escreverá e salvará seus *scripts* R (arquivos com extensão `.R`). Trabalhar em um *script*, em vez de digitar comandos diretamente no console, é a base da ciência reprodutível, pois permite salvar, comentar e reutilizar seu código.
2. **Console** (Inferior esquerdo): O console é o código R é efetivamente executado. Você pode digitar os comandos diretamente nele para testes rápidos ou executar linhas de códigos do seu *script* (utilizando o atalho `Ctrl+Enter`). A saída dos comandos também aparecerá aqui.
3. **Ambiente e Histórico** (Superior direito): A aba *Environment* mostra todos os objetos (como *datasets*, variáveis, etc.) que foram criadas na sessão atual do R. Já a aba *History* mantém um registro de todos os comandos utilizados.
4. **Arquivos, Gráficos, Pacotes e Ajuda** (Inferior direita): Este painel multifuncional permite navegar pelos arquivos do seu computador (*Files*) , visualizar gráficos gerados (*Plots*), gerenciar pacotes instalados (*Packages*), e acessar documentações de ajuda do R (*Help*).

É importante salientar que o RStudio permite customizações, como a alteração das posições dos painéis.

1.4 O conceito de pacotes

A grande força do R reside em seu ecossistema de pacotes. Um pacote é a coleção de funções, dados e documentação que estende as capacidades iniciais do R. Para qualquer tarefa estatística ou de manipulação de dados que se possa imaginar, provavelmente existe algum pacote que a facilita.

1.4.1 Instalando e Carregando Pacotes Essenciais

Existe uma distinção básica a ser realizada entre instalar e carregar um pacote.

- **Instalação**: É o ato de baixar o pacote do CRAN e instalá-lo no computador. Isso é realizado apenas uma vez para cada pacote.
- **Carregamento**: É o ato de carregar o pacote instalado em sua sessão do R de forma que as funções adicionais fiquem disponíveis para uso. Isso precisa ser feito toda vez que uma sessão no R é iniciada.

Para este material, os pacotes centrais são: `tidyverse`, `lme4`, `lmerTest` e `nlme`. Um dos métodos para instalar pacotes R no computador é por meio da função `install.packages()`:

```
# Instala o pacote tidyverse, que inclui dplyr, ggplot2 e outros
install.packages("tidyverse")

# Instala o pacote para modelos lineares mistos
install.packages("lme4")

# Instala outros pacotes para modelos mistos
install.packages("lmerTest")
install.packages("nlme")
```

Após a instalação, para usar as funções de um pacote, é preciso carregá-lo com a função `library()`:

```
library(tidyverse)
```

Cabe ressaltar que, ao longo do uso de diversos pacotes, podem ocorrer conflitos de funções com o mesmo nome. Nesses casos, a solução mais prática é utilizar a notação `pacote::funcao` para indicar explicitamente ao R de qual biblioteca desejamos chamar a função.

1.5 Diretório de Trabalho e Projetos RStudio

O diretório de trabalho é a pasta no seu computador onde o R irá procurar por arquivos para ler e onde, também, salvará os arquivos criados (como gráficos, *scripts* e *datasets* modificados). É possível identificar o diretório atual através do comando `getwd()` e, embora também seja possível defini-la manualmente com a função `setwd("caminho/para/sua/pasta")`, essa prática não é aconselhável, visto que o uso de caminhos de arquivos absolutos torna o código não portável; ou seja, ele não irá funcionar se você mover a pasta do projeto ou tentá-la executá-lo em outro computador.

A solução moderna e robusta para esse problema é a utilização de **Projetos RStudio**. Um projeto RStudio (extensão `.Rproj`) é um arquivo que você cria dentro de uma pasta do seu projeto de pesquisa. Ao abrir um projeto, o RStudio automaticamente define o diretório de trabalho para aquela pasta. Isso garante que todos os caminhos de arquivo do seu código possam ser relativos à raiz do projeto, tornando sua análise totalmente reprodutível e compartilhável de forma eficaz. Outra maneira de criar projetos é através do próprio RStudio, através das seguintes instruções `File > New Project > New Directory > New Project` e nesta última etapa, você escolherá um nome para o projeto e a pasta de sua pesquisa, finalizando em `Create Project`. A criação de um projeto para cada análise de pesquisa é uma prática fundamental para a organização e a reprodutibilidade científica.

1.6 R Básico

A leitura desta sessão é aconselhada para o leitor que nunca teve contato com o R. Os tópicos introduzidos são especiais para a compreensão do que é um *dataframe*, a estrutura dos *datasets* dentro do R, e quais operações estarão sendo realizadas quando estivermos efetuando filtragens e modificações de suas colunas. Também são importantes para a compreensão do que é uma função no R.

1.6.1 Operadores Matemáticos

Os operadores matemáticos, também conhecidos por operadores binários, dentro do ambiente R soam como familiares. A Tabela 1.1 exibe os operadores mais básicos utilizados.

Para exemplificar como efetuar cálculos de expressões matemáticas no R, suponha que desenhamos calcular o valor de:

$$2 \times 2 + \frac{4+4}{2}.$$

Para isso, escrevemos `2*2 + (4+4)/2` no console para determinarmos o resultado

```
2*2 + (4+4)/2
```

```
[1] 8
```

Tabela 1.1: Operadores matemáticos básicos.

Operadores	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
^	Exponenciação

1.6.2 Objetos e funções

O R permite guardar valores dentro de um **objeto**. Um objeto é simplesmente um nome que guarda uma determinada informação na memória do computador, que é criado por meio do operador `<-`. Veja que no código a seguir

```
x <- 10 # Salvando "10" em "x"
x       # Avaliando o objeto "x"
```

```
[1] 10
```

foi salvo que a informação que `x` carrega é o valor `10`. Portanto, toda vez que o objeto `x` for avaliado, o R irá devolver o valor `10`.

É importante ressaltar que há regras para a nomeação dos objetos, dentre elas, não começar com números. Assim, todos os seguintes exemplos são permitidos: `x <- 1`, `x1 <- 1`, `meu_objeto <- 1`, `meu.objeto <- 1`. Ainda, o R diferencia letras minúsculas de maiúsculas, então objetos como `y` e `Y` são diferentes.

Enquanto que os objetos são nomes que salvam informações de valores, **funções** são nomes que guardam informações de um código R, retornando algum resultado programado. A sintaxe básica de uma função é `nome_funcao(arg1, arg2, ...)`. Os valores dentro dos parênteses são chamados por **argumentos**, que são informações necessárias para o bom funcionamento de uma função. Às vezes, uma função não necessita do fornecimento de argumentos específicos.

Uma função simples, porém útil, é a `sum()`. Ela consiste em somar os valores passados em seu argumento. Suponha que desejamos somar `1+2+3+4+5`. Assim,

```r
sum(1,2,3,4,5)
```

```
[1] 15
```

é possível reparar que o resultado é 15.

A classe de um objeto é muito importante na programação em R. É a partir disso que as funções e operadores conseguem entender o que fazer com cada objeto. Há uma infinidade de classes, dentre as mais conhecidas são: `numeric`, `character`, `data.frame`, `logical` e `factor`. Para averiguar o tipo de classe, a função `class()` retorna exatamente a classe do objeto.

```r
class("a")
```

```
[1] "character"
```

```r
class(1)
```

```
[1] "numeric"
```

```r
class(mtcars)
```

```
[1] "data.frame"
```

```r
class(TRUE)
```

```
[1] "logical"
```

1.6.3 Importanto dados

Uma atividade importante para qualquer análise estatística que vier ser feita no R é importante importar os dados para o ambiente de trabalho, que ficarão guardados dentro de um objeto no projeto RStudio – afinal, como faríamos as análises sem os dados? No contexto da Biologia, isso costuma significar ler arquivos com medidas de peso, contagens de indivíduos, medidas de comprimento etc., geralmente armazenados em formatos de texto (`.csv` ou `.tsv`) ou planilhas (`.xlsx`). As principais funções para cada ocasião de arquivo são:

- CSV com cabeçalho:

```r
dados <- read.csv("dados.csv",
  header = TRUE, # indica que há cabeçalho
  sep    = ",",  # separador vírgula
  stringsAsFactors = FALSE # evita conversão automática em fatores
)
```

- TXT ou TSV com tabulação:

```r
dados <- read.delim("dadostsv",
  header = TRUE,
  sep    = "\t"
)
```

- Planilhas no Excel (arquivos `.xlsx`):

```r
dados <- readxl::read_excel("dados.xlsx",
  sheet = "Planilha1" # aqui você escolhe a planilha a ser lida
)
```

Ressaltamos, neste caso, a necessidade da utilização da biblioteca `readxl` para que seja possível lermos planilhas no R.

1.6.4 Vetores e *Data frames*

Vetores são uma estrutura fundamental dentro do R, em especial, é a partir deles que os *data frames* são construídos. Por definição, são conjuntos indexados de valores e para criá-los, basta utilizar a função `c()` com valores separados por vírgula (ex.: `c(1,2,4,10)`). Para acessar um valor dentro de um determinado vetor, utiliza-se os colchetes `[]`:

```r
vetor <- c("a", "b", "c")

# Acessando valor "b"
vetor[2]
```

```
[1] "b"
```

Um vetor só pode guardar um tipo de objeto e ele terá sempre a mesma classe dos objetos que guarda. Caso tentarmos misturar duas classes, o R vai apresentar o comportamento conhecido como **coerção**.

```r
class(c(1,2,3))
```

```
[1] "numeric"
```

```r
class(vetor)
```

```
[1] "character"
```

```r
class(c(1,2,"a","b"))
```

```
[1] "character"
```

Neste caso, todos os elementos do vetor se transformaram em texto.

Assim, também, *data frames* são de extrema importância no R, visto que são os objetos que guardam os dados e são equivalentes a uma planilha do Excel. A principal característica é possuir linha e colunas. Em geral, as colunas são vetores de mesmo tamanho (ou dimensão). Um valor específico de um *data frame* pode ser acessado, também, via colchetes `[]`:

```r
class(mtcars)
```

```
[1] "data.frame"
```

```r
mtcars[1,2]
```

```
[1] 6
```

`mtcars` é um conjunto de dados muito conhecido na comunidade R.

1.6.5 Fatores

Fatores são uma classe de objetos no R criada para representar variáveis categóricas numericamente. A característica que define essa classe é o atributo `levels`, que representam as possíveis categorias de uma variável categórica.

A título de exemplificação, considere o objeto `sexo` que contém as informações do sexo de uma pessoa. As possibilidades são: `F` (feminino) e `M` (masculino). Por padrão, o R interpreta essa variável como texto (*character*), no entanto, é possível transformá-la em fator por meio da função `as.factor()`.

```r
sexo <- c("F", "F", "M", "M", "F")
class(sexo)
```

```
[1] "character"
```

```r
# Transformando em fator
class(as.factor(sexo))
```

```
[1] "factor"
```

```r
as.factor(sexo)
```

```
[1] F F M M F
Levels: F M
```

Observa-se que a linha adicional `Levels: F M` indicam as categorias. Por padrão, o R ordena esses níveis em ordem alfabética. Para facilitar os cálculos e análises, o R interpreta os níveis categóricos como sendo números distintos, sendo assim, dentro do nosso exemplo `F` representaria o número `0` e `M` representaria o `1`.

1.6.6 Valores especiais

Valores como `NA`, `NaN`, `Inf` e `NULL` ocorrem frequentemente dentro do mundo da programação estatística no R. Em resumo:

- `NA` representa a Ausência de Informação. Suponha que o vetor `idades` que representa a idade de três pessoas. Uma situação que pode ocorrer é `idades <- c(10, NA, NA)`. Portanto, não é sabido a idade das pessoas 2 e 3.
- `NaN` representa indefinições matemáticas. Um exemplo típico é o valor $\log -1$, do qual $x = -1$ não pertence aos possíveis valores de saída da função logarítmica, gerando um `NaN` (*Not a number*).

```r
log(-1)
```

```
Warning in log(-1): NaNs produzidos
```

```
[1] NaN
```

- `Inf` representa um número muito grande ou um limite matemático. Exemplos:

```
# Número muito grande
10^510
```

```
[1] Inf
```

```
# Limite matemático
1/0
```

```
[1] Inf
```

- `NULL` representa a ausência de um objeto. Muitas vezes define-se um objeto como nulo para dizer ao R que não desejamos atribuir valores a ele.

1.6.7 Pedindo ajuda

Uma das coisas que intimidam novos programadores, independente da linguagem utilizada, é a ocorrência de erros. Neste sentido, o R pode ser um grande aliado, pois ele relata mensagens, erros e avisos sobre o código no console, como se fosse uma espécie de resposta e/ou comunicação. As situações são:

- `Error`: em situações de erro legítimo aparecerá mensagens do tipo `Error in ...` e tentará explicar o que há de errado. Nestas situações o código, geralmente, não é executado. Por exemplo: `Error in ggplot(...) : could not find function "ggplot"`.
- `Warning`: em situações de avisos, o R exibirá uma mensagem do tipo `Warning: ...` e tentará explicar o motivo do aviso. Geralmente, o código será executado, mas com algumas ressalvas. Por exemplo: `Warning: Removed 2 rows containing missing values (geom_point)`.
- `Message`: quando o texto exibido não se enquadra nas duas opções anteriores, dizemos que é apenas uma mensagem. Pense, nessa situação, que tudo está acontecendo como o esperado e está tudo bem.

Quando surgir qualquer uma dessas saídas, não estaremos perdidos, pois o R oferece mecanismos para encontrarmos respostas. Afinal, nem todo mundo decorou todas as funções ou argumentos. Os principais mecanismos são:

- `?função` ou `help(função)` para consultar a documentação oficial.
- `??termo` e `help.search("termo")` para buscas por palavras-chave.

Além disso, o RStudio oferece alguns *Cheatsheets* (resumo de códigos) que podem ajudar com determinados pacotes. E, por fim, existem grandes comunidade online, tais como: Stack Overflow e RStudio Community dos quais também podem serem úteis.

2 Tidyverse

O `tidyverse` (WICKHAM ET AL., 2019) é um ecossistema de pacotes R que reúne as tarefas essenciais de qualquer fluxo de trabalho em ciência de dados: importação, organização, manipulação, visualização e programação. Seu principal objetivo é criar uma sintaxe consistente e legível, facilitando a comunicação entre quem escreve o código e quem o executa. Note-se que, embora o tidyverse cubra grande parte do fluxo de trabalho, ele não inclui ferramentas específicas de modelagem estatística.

Para facilitar essa integração, o `tidyverse` utiliza intensamente do operador pipe[1] (%>%) , que passa o resultado de uma etapa diretamente para a próxima, evitando aninhamentos confusos. Ao carregar o pacote, diversos módulos são automaticamente disponibilizados:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------ tidyverse 2.0.0 --
 dplyr     1.1.4      readr     2.1.5
 forcats   1.0.0      stringr   1.5.1
 ggplot2   3.5.2      tibble    3.2.1
 lubridate 1.9.4      tidyr     1.3.1
 purrr     1.0.4
-- Conflicts ------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (http://conflicted.r-lib.org/)
  to force all conflicts to become errors
```

Entre os principais estão:

- `ggplot2` (visualização de dados);
- `dplyr` (manipulação de dados);
- `tidyr` (formatação "*long*"/"*wide*");

[1] A partir da versão 4.1 do R, existe também o operador pipe nativo |>. No entanto, nesta apostila manteremos o uso de %>%, amplamente adotado no contexto do tidyverse.

- `readr` (leitura eficiente de arquivos de texto);
- `tibble` (versão moderna do `data.frame`);
- `purrr` (programação funcional);
- `stringr`, `forcats` e outros.

Como dito, muitos pacotes definem funções com nomes idênticos, sendo costatumum que o console exiba nomes como:

```
The following objects are masked from 'package:stats':
    filter, lag
```

Um pilar do `tidyverse` é a adoção do princípio `tidy` (WICKHAM, 2014), em que:

- Cada variável ocupa uma coluna;
- Cada observação ocupa uma linha;
- Cada tipo de entidade observacional fica em sua própria tabela.

Nesse contexto, a **entidade observacional** é o conceito central que define o que uma linha representa. Pode ser um paciente em um estudo clínico, um país em dados econômicos ou, como nos exemplos a seguir:

- **Aves**: Cada linha corresponde a uma única ave, registrando suas características (peso, envergadura, espécie, etc.).
- **Plantas**: Cada linha representa um vaso de planta em um experimento (altura, número de folhas, tipo de solo, etc.).

A estrutura de dados que implementa essa filosofia no `tidyverse` é o `tibble`. Ele é a versão moderna do `data.frame`, projetado para ser mais prático e informativo, exibindo resumos concisos dos dados e fornecendo diagnósticos mais úteis.

Uma vez apresentada a filosofia e a estrutura de dados do `tidyverse`, o foco se volta para a aplicação prática. A seguir, a concentração do material residirá nos dois pacotes centrais do `tidyverse`: o `dplyr`, para manipulação de dados, e o `ggplot2`, para a criação de gráficos.

2.1 Manipulação de dados com o pacote `dplyr`

O `dplyr` é um pacote do `tidyverse` que fornece um conjunto de ferramentas robustas e intuitivas para manipulação de dados. Os comandos oferecidos soam um tanto quanto intuitivos, correspondendo ações comuns na área de análise de dados. Para explorar as principais funções

será utilizado o *dataset* `penguins`, focando em processos de filtragem, organização, transformação e resumos dos dados, permitindo responder a perguntas básicas sobre a biologia e ecologia dos pinguins.

O primeiro passo a ser feito é instalar a biblioteca `palmerpenguins` e, em seguida, carregá-la no ambiente de trabalho, para que possamos realizar uma inspeção inicial na estrutura dos dados.

```
install.packages("palmerpenguins") # Realizar apenas uma única vez
```

```
library(palmerpenguins)
```

Para carregarmos os dados sobre pinguins no ambiente de trabalho, podemos utilizar a função `data()`:

```
data("penguins", package = "palmerpenguins")
```

Podemos observar que no painel **Environment** do RStudio, aparece o objeto `penguins`, isso significa que o conjunto de dados está carregado no ambiente de trabalho e podemos dar início nas inspeções. O primeiro comando que será visto é o `glimpse()`. Ele exibe, de maneira prática e rápida, a estrutura do *dataset* como: dimensão (número de linhas e colunas), o nome de cada coluna, o tipo de dado de cada coluna e as primeiras observações.

```
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex               <fct> male, female, female, NA, female, male, female, male~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

A saída deste comando revela que existem 344 observações e 8 variáveis, sendo elas `species`, `island`, `bill_length_mm`, `flipper_length_mm`, `body_mass_g`, `sex` e `year`, com seus respectivos tipos, como `factor` para `species` e `numeric` para `bill_length_mm`. Além disso, é possível observar dados ausentes em algumas variáveis, representados por `NA`. Em geral,

nos *datasets* disponíveis em pacotes R, é possível utilizar o comando `help(penguins)` para buscar informações sobre o conjunto de dados que será trabalhado.

Executando o comando de ajuda, são obtidas as seguintes informações sobre as variáveis:

- `species`: um fator que denota a espécie do pinguim (Adélie, Chinstrap ou Gentoo).
- `island`: um fator que denota ilhas no Arquipélago Palmer na Antártica (Biscoe, Dream ou Torgersen).
- `bill_length_mm`: um número que representa o comprimento do bico (em milímetros).
- `bill_depth_mm`: um número que representa a profundidade do bico (em milímetros).
- `flipper_length_mm`: um número que representa o comprimento da nadadeira (em milímetros).
- `body_mass_g`: um número inteiro que representa a massa do animal (em gramas).
- `sex`: um fator que representa o sexo do animal (feminino ou masculino).
- `year`: um número inteiro que denota o ano de estudo (2007, 2008 ou 2009).

Adicionalmente, também é informado que os dados foram originalmente publicados no estudo de Gorman et al. (2014) e que essa pesquisa fez parte do programa *Palmer Station Long-Term Ecological Research* (LTER). Isso significa que o conjunto de dados que está sendo utilizado possui uma origem científica real, ligada a questões sobre como o ambiente e as diferenças entre sexos afetam a vida dessas aves.

A segunda função que será vista é o `select()`. Frequentemente, um conjunto de dados contém mais informações do que o necessário para uma análise específica. Com isso em mente, a função `select()` permite-nos selecionar colunas de interesse. Em geral, os argumentos são os nomes das colunas.

```
penguins %>%
  select(species, island, sex)
```

```
# A tibble: 344 x 3
   species island    sex
   <fct>   <fct>     <fct>
 1 Adelie  Torgersen male
 2 Adelie  Torgersen female
 3 Adelie  Torgersen female
 4 Adelie  Torgersen <NA>
 5 Adelie  Torgersen female
 6 Adelie  Torgersen male
 7 Adelie  Torgersen female
```

```
 8 Adelie  Torgersen male
 9 Adelie  Torgersen <NA>
10 Adelie  Torgersen <NA>
# i 334 more rows
```

O `dplyr` também oferece "seletores auxiliares" que tornam a seleção mais poderosa e flexível. Por exemplo, caso desejarmos selecionar todas as medidas biométricas contidas no *dataset* que terminam com `_mm`, é possível usar a função-argumento `ends_with()` dentro de `select()`:

```
penguins %>%
  select(
    body_mass_g, ends_with("_mm")
  )
```

```
# A tibble: 344 x 4
   body_mass_g bill_length_mm bill_depth_mm flipper_length_mm
         <int>          <dbl>         <dbl>              <int>
 1        3750           39.1          18.7                181
 2        3800           39.5          17.4                186
 3        3250           40.3          18                  195
 4          NA           NA            NA                   NA
 5        3450           36.7          19.3                193
 6        3650           39.3          20.6                190
 7        3625           38.9          17.8                181
 8        4675           39.2          19.6                195
 9        3475           34.1          18.1                193
10        4250           42            20.2                190
# i 334 more rows
```

Outros seletores úteis incluem `starts_with()` e `contains()`. Para remover colunas, utiliza-se o sinal de menos (`-`). Por exemplo, deseja-se remover as colunas `ano` e `island`:

```
penguins %>%
  select(-year, -island)
```

```
# A tibble: 344 x 6
   species bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex
   <fct>            <dbl>         <dbl>              <int>       <int> <fct>
 1 Adelie            39.1          18.7                181        3750 male
```

```
 2 Adelie            39.5          17.4               186        3800 female
 3 Adelie            40.3          18                 195        3250 female
 4 Adelie            NA            NA                  NA          NA <NA>
 5 Adelie            36.7          19.3               193        3450 female
 6 Adelie            39.3          20.6               190        3650 male
 7 Adelie            38.9          17.8               181        3625 female
 8 Adelie            39.2          19.6               195        4675 male
 9 Adelie            34.1          18.1               193        3475 <NA>
10 Adelie            42            20.2               190        4250 <NA>
# i 334 more rows
```

Antes prosseguirmos para a próxima função, vale destacar que o conjunto de dados `penguins` é um objeto `tibble` dentro do R e, portanto, por mais que existam 344 observações, o `tibble` enxuga a visualização para somente 10, além de indicar quantas linhas ainda existem.

A terceira função é o `filter()`. Enquanto `select()` trabalha nas colunas, o `filter()` trabalha nas linhas, permitindo-nos manter apenas as observações que satisfazem certas condições. É aqui que é possível responder perguntas investigadas com relação aos dados. Por exemplo, para encontrar todos os pinguins da espécie Adelie que vivem na ilha Torgersen:

```
penguins %>%
  filter(
      species == "Adelie", island == "Torgersen"
    )
```

```
# A tibble: 52 x 8
   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
 7 Adelie  Torgersen           38.9          17.8               181        3625
 8 Adelie  Torgersen           39.2          19.6               195        4675
 9 Adelie  Torgersen           34.1          18.1               193        3475
10 Adelie  Torgersen           42            20.2               190        4250
# i 42 more rows
# i 2 more variables: sex <fct>, year <int>
```

Neste exemplo, as condições separadas por vírgula são unidas por um "E" lógico. Também é possível utilizar o "OU" lógico para determinar pinguins mais pesados (acima de 6000g) ou com bicos muito longos (mais de 55mm) através do conectivo `|`:

```
penguins %>%
  filter(
    body_mass_g > 6000 | bill_length_mm > 55
  )
```

```
# A tibble: 6 x 8
  species   island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
  <fct>     <fct>           <dbl>         <dbl>             <int>       <int>
1 Gentoo    Biscoe           49.2          15.2               221        6300
2 Gentoo    Biscoe           59.6          17                 230        6050
3 Gentoo    Biscoe           55.9          17                 228        5600
4 Gentoo    Biscoe           55.1          16                 230        5850
5 Chinstrap Dream            58            17.8               181        3700
6 Chinstrap Dream            55.8          19.8               207        4000
# i 2 more variables: sex <fct>, year <int>
```

O `filter()` também permite encontrar valores ausentes (`NA`s) em conjunto da função `is.na()`. Por exemplo, deseja-se verificar quais pinguins não tiveram seu sexo registrado:

```
penguins %>%
  filter(is.na(sex))
```

```
# A tibble: 11 x 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
 1 Adelie  Torgersen           NA            NA                  NA          NA
 2 Adelie  Torgersen           34.1          18.1               193        3475
 3 Adelie  Torgersen           42            20.2               190        4250
 4 Adelie  Torgersen           37.8          17.1               186        3300
 5 Adelie  Torgersen           37.8          17.3               180        3700
 6 Adelie  Dream               37.5          18.9               179        2975
 7 Gentoo  Biscoe              44.5          14.3               216        4100
 8 Gentoo  Biscoe              46.2          14.4               214        4650
 9 Gentoo  Biscoe              47.3          13.8               216        4725
10 Gentoo  Biscoe              44.5          15.7               217        4875
11 Gentoo  Biscoe              NA            NA                  NA          NA
# i 2 more variables: sex <fct>, year <int>
```

A interpretação do `NA` é relativa ao contexto dos dados. No caso das observações sobre os pinguins, os valores ausentes na variável `sex` permite identificar pinguins que não tiveram o sexo avaliado, tornando um provável erro frustrante de coleta de dados para um objeto de investigação. O pacote `tidyr`, também do `tidyverse`, oferece a função `drop_na()`, que remove quaisquer linhas que contenham `NA`s, permitindo a criação de um *dataset* auxiliar:

```
penguins_completos <- penguins %>%
  drop_na()
```

A quarta função que será apresentada é `arrange()`, que permite reordenar as linhas do dataframe com base nos valores de uma ou mais colunas. Isso é útil para encontrar extremos ou simplesmente para organizar a saída de uma forma mais lógica. Para encontrar os pinguins mais leves, ordenamos pela massa corporal em ordem crescente (o padrão):

```
penguins %>%
  arrange(body_mass_g)
```

```
# A tibble: 344 x 8
   species   island   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>     <fct>             <dbl>         <dbl>             <int>       <int>
 1 Chinstrap Dream              46.9          16.6               192        2700
 2 Adelie    Biscoe             36.5          16.6               181        2850
 3 Adelie    Biscoe             36.4          17.1               184        2850
 4 Adelie    Biscoe             34.5          18.1               187        2900
 5 Adelie    Dream              33.1          16.1               178        2900
 6 Adelie    Torgers~           38.6          17                 188        2900
 7 Chinstrap Dream              43.2          16.6               187        2900
 8 Adelie    Biscoe             37.9          18.6               193        2925
 9 Adelie    Dream              37.5          18.9               179        2975
10 Adelie    Dream              37            16.9               185        3000
# i 334 more rows
# i 2 more variables: sex <fct>, year <int>
```

Para ordenar os valores em ordem decrescente (do maior para o menor), utilizamos a função auxiliar `desc()`, desta maneira, encontramos os pinguins mais pesados:

```
penguins %>%
  arrange(desc(body_mass_g))
```

```
# A tibble: 344 x 8
   species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
 1 Gentoo  Biscoe           49.2          15.2               221        6300
 2 Gentoo  Biscoe           59.6          17                 230        6050
 3 Gentoo  Biscoe           51.1          16.3               220        6000
 4 Gentoo  Biscoe           48.8          16.2               222        6000
 5 Gentoo  Biscoe           45.2          16.4               223        5950
 6 Gentoo  Biscoe           49.8          15.9               229        5950
 7 Gentoo  Biscoe           48.4          14.6               213        5850
 8 Gentoo  Biscoe           49.3          15.7               217        5850
 9 Gentoo  Biscoe           55.1          16                 230        5850
10 Gentoo  Biscoe           49.5          16.2               229        5800
# i 334 more rows
# i 2 more variables: sex <fct>, year <int>
```

Também é possível ordenar múltiplas colunas. Por exemplo, para encontrar o pinguim mais pesado dentro de cada espécie:

```
penguins %>%
  arrange(
    species, # Primeiro por espécie
    desc(body_mass_g) # Depois por massa decrescente
  )
```

```
# A tibble: 344 x 8
   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
 1 Adelie  Biscoe              43.2          19                 197        4775
 2 Adelie  Biscoe              41            20                 203        4725
 3 Adelie  Torgersen           42.9          17.6               196        4700
 4 Adelie  Torgersen           39.2          19.6               195        4675
 5 Adelie  Dream               39.8          19.1               184        4650
 6 Adelie  Dream               39.6          18.8               190        4600
 7 Adelie  Biscoe              45.6          20.3               191        4600
 8 Adelie  Torgersen           42.5          20.7               197        4500
 9 Adelie  Dream               37.5          18.5               199        4475
10 Adelie  Torgersen           41.8          19.4               198        4450
# i 334 more rows
# i 2 more variables: sex <fct>, year <int>
```

A quinta função e, com certeza, uma das mais funcionais é a `mutate()`. Ela permite criar novas colunas (variáveis) que são funções de colunas já existentes, sem modificar as originais. Por exemplo, suponha que desejamos mostrar somente as espécies e massas de pinguins em quilogramas (kg):

```
penguins %>%
  mutate(body_mass_kg = body_mass_g/1000) %>%
  select(species, body_mass_kg)
```

```
# A tibble: 344 x 2
   species body_mass_kg
   <fct>          <dbl>
 1 Adelie          3.75
 2 Adelie          3.8
 3 Adelie          3.25
 4 Adelie         NA
 5 Adelie          3.45
 6 Adelie          3.65
 7 Adelie          3.62
 8 Adelie          4.68
 9 Adelie          3.48
10 Adelie          4.25
# i 334 more rows
```

Podemos usar `mutate()` para criar categorias. A função `case_when()` é extremamente útil para criar classificações baseadas em condições lógicas., Suponha que desejamos criar uma categoria de tamanho baseada na massa corporal:

```
penguins %>%
  mutate(
    size_category = case_when(
      body_mass_g > 4750 ~ "Grande",
      body_mass_g < 3500 ~ "Pequeno",
      TRUE ~ "Médio"
    )
  ) %>%
  select(
    species, body_mass_g, size_category
  )
```

```
# A tibble: 344 x 3
   species body_mass_g size_category
   <fct>         <int> <chr>
 1 Adelie         3750 Médio
 2 Adelie         3800 Médio
 3 Adelie         3250 Pequeno
 4 Adelie           NA Médio
 5 Adelie         3450 Pequeno
 6 Adelie         3650 Médio
 7 Adelie         3625 Médio
 8 Adelie         4675 Médio
 9 Adelie         3475 Pequeno
10 Adelie         4250 Médio
# i 334 more rows
```

As funções `group_by()` e `summarise()` formam uma dupla formidável para agrupar e resumir os dados, pertencendo ao coração da análise de dados. A função `summarise()` erve para calcular estatísticas resumidas (como média, total, mínimo etc.) e, quando usada em conjunto com `group_by()` permite gerar resumos por grupo.

Inicialmente, vamos utilizar o `summarise()` no *dataset* completo para obter estatísticas globais. Não obstante, é bom frisar a utilização do argumento `na.rm = TRUE` para instruir a remoção dos valores `NA`.

```
penguins %>%
  summarise(
    massa_media = mean(body_mass_g, na.rm = TRUE),
    nadadeira_max = max(flipper_length_mm, na.rm = TRUE)
  )
```

```
# A tibble: 1 x 2
  massa_media nadadeira_max
        <dbl>         <int>
1       4202.           231
```

No entanto, essas métricas não fornecem informações com relação as espécies de pinguins. Para resolver isso e possibilitar que mais perguntas sejam respondidas, a função `group_by()` permite que o R faça operações em subconjuntos. Por exemplo, suponha que desejamos determinar qual é a massa corporal por espécie:

```r
penguins %>%
  group_by(species) %>%
  summarise(
    massa_media_g = mean(body_mass_g, na.rm = TRUE)
  )
```

```
# A tibble: 3 x 2
  species   massa_media_g
  <fct>             <dbl>
1 Adelie            3701.
2 Chinstrap         3733.
3 Gentoo            5076.
```

Podemos fazer agrupamentos por múltiplas variáveis para investigações mais profundas. Por exemplo, considere que um pesquisador deseja explorar o dimorfismo sexual. Para isso, estatísticas por espécie e sexo serão calculadas.

```r
tabela_resumo <- penguins %>%
  drop_na(sex) %>%
  group_by(species, sex) %>%
  summarise(
    contagem = n(),
    massa_media_g = mean(body_mass_g),
    massa_dp_g = sd(body_mass_g),
    comp_bico_medio_mm = mean(bill_length_mm),
    .groups = "drop"
  )
tabela_resumo
```

Tabela 2.1: Estatísticas descritivas de características biométricas de pinguins, agrupadas por espécie e sexo.

Espécies	Sexo	Contagem	Massa média (g)	Massa Desvio-padrão (g)	Comprimento médio do bico (mm)
Adelie	female	73	3368.84	269.38	37.26
Adelie	male	73	4043.49	346.81	40.39
Chinstrap	female	34	3527.21	285.33	46.57
Chinstrap	male	34	3938.97	362.14	51.09
Gentoo	female	58	4679.74	281.58	45.56
Gentoo	male	61	5484.84	313.16	49.47

Vale reforçar que a Tabela 2.1 foi gerada usando o `dplyr`, com as funções auxiliares `n()` para realizar a contagem de observações em cada grupo e `drop_na(sex)` para remover as observações onde o sexo é desconhecido, permitindo avaliar dimorfismo sexual em todas as três espécies, especialmente na massa corporal. O grande potencial dessa tabela é obter respostas como:

- Os pinguins Gentoo são, em média, os mais pesados.
- Dentro de cada espécie, os machos são consistentemente mais pesados e têm bicos mais longo que as fêmeas.

Esses resultados permitem tirar conclusões sobre algumas hipóteses biológicas.

Por fim, a última função que será abordada é a `recode()`. Muitas vezes, os nomes das categorias nos conjuntos de dados não são ideais para a análise ou apresentação em gráficos. Podem ser longos demais, estarem em outro idioma ou simplesmente não serem claros. Para isso, a função `recode()` permite renomear valores de uma variável categórica de forma simples e direta. Por exemplo, suponha que desejamos traduzir os termos da variável `sex` da Tabela 2.1 para o português:

```
tabela_resumo %>%
  mutate(
    sex = recode(sex,
                 "female" = "Fêmea",
                 "male" = "Macho")
  )
```

Tabela 2.2: Tradução da variável `sexo` da Tabela 2.1.

Espécies	Sexo	Contagem	Massa média (g)	Massa Desvio-padrão (g)	Comprimento médio do bico (mm)
Adelie	Fêmea	73	3368.84	269.38	37.26
Adelie	Macho	73	4043.49	346.81	40.39
Chinstrap	Fêmea	34	3527.21	285.33	46.57
Chinstrap	Macho	34	3938.97	362.14	51.09
Gentoo	Fêmea	58	4679.74	281.58	45.56
Gentoo	Macho	61	5484.84	313.16	49.47

As principais funções do pacote `dplyr` que foram vistas estão resumidas e descritas na Tabela 2.3 e agora que aprendemos como manipular os dados com o `dplyr`, podemos avançar para a construção de gráficos com o pacote `ggplot2`.

Tabela 2.3: Descrição das principais funções do `dplyr`.

Função	Descrição
glimpse()	Inspecionar conjuntos de dados.
select()	Seleciona colunas pelo nome.
filter()	Filtra linhas com base em seus valores.
arrange()	Reordena as linhas.
mutate()	Cria novas colunas (variáveis).
group_by()	Agrupa os dados por uma ou mais variáveis.
summarise()	Reduz múltiplos valores a um único resumo.
recode()	Renomeia categorias de variáveis.
n()	Conta o número de observações.

2.2 Visualização de Dados com `ggplot2`

Se o `dplyr` é a gramática da manipulação de dados, possuindo funções essenciais para esse trabalho, o `ggplot2` (WICKHAM, 2016) é gramática dos gráficos, permitindo construir gráficos por meio de camadas e oferecendo um sistema robusto e flexível para visualização os dados. Nesta seção, continuaremos utilizando os dados dos pinguins para explorar alguns *insights* visuais, desde gráficos mais simples até os mais elaborados.

Todo gráfico no `ggplot2` é constituído por três camadas essenciais:

1. Dados (`data`): O *dataframe* que contém as informações a serem plotadas.
2. Mapeamento Estéticos (`aes`): A função `aes()` (de *aesthetics*) descreve como as variáveis do nosso *dataframe* são mapeadas para as propriedades visuais do gráfico. As estéticas mais comuns são `x` e `y` (os eixos), mas também incluem `color` (cor), `shape` (forma), `size` (tamanho) e `alpha` (transparência/opacidade).
3. Objetos geométricos (`geom`): Os `geoms` definem como os dados são representados visualmente. Por exemplo, `geom_point()` cria um gráfico de dispersão, `geom_bar()` cria um gráfico de barras, `geom_line()` cria um gráfico de linhas, e assim por diante.

2.2.1 Estatística descritiva

Antes de explorar as relações gráficas, é útil enfatizar e entender alguns conceitos essenciais da Estatística Descritiva como os tipos de variáveis, normas para tabelas e as definições de frequências.

Em geral, pode-se dizer que existem duas categorias de variáveis dentro da estatística:

1. **Variáveis Qualitativas**: Também chamadas por variáveis categóricas e como o próprio nome diz, expressam qualidade e indicam categoria ou classificação a qual o objeto pertence. Se existir uma ordem entre as possíveis categorias, a variável é dita **qualitativa ordinal**. Caso contrário, é dita ser **qualitativa nominal**.
2. **Variáveis Quantitativas**: São variáveis que tomam valores numéricos e expressam quantidade. Podem ser especificadas por **Variáveis Discretas**, quando assumem valores dentro de um conjunto enumerável (quando é possível contá-las) ou por **Variáveis Continuas**, quando podem assumir infinitos valores de um intervalo não inumerável[2] (não é possível contar o número de valores dentro de um intervalo).

Assim, para explorar e apresentar as informações contidas num conjunto de dados, precisamos resumir essas informações de forma que seja possível enxergá-las rapidamente e adquirir conhecimento sobre o assunto.

O resumo pode ser feito por meio de tabelas, gráficos e cálculo de algumas quantidades representativas. O primeiro passo é identificar o tipo de cada variável para aplicarmos a técnica apropriada.

As normas gerais para construção de uma tabela envolvem:

- Devem ser auto-explicativas.
- Devem conter um título, que precisa ser simples e claro, indicando informações sobre os dados (do que, onde e quando foram coletados – se forem relevantes).

Além das normas gerais, existem duas convenções importantes a serem seguidas com relação ao título de tabelas e gráficos:

- Em **tabelas** os títulos vem primeiro, em cima da tabela.
- Em **gráficos** os títulos vem por último, embaixo do gráfico.

Se necessário, notas e fontes vêm embaixo, em ambos os casos. Uma tabela começa e termina com um traço horizontal e traços na vertical devem ser evitados, conforme visto na Tabela 2.1 por exemplo.

Uma forma adequada para resumir informações sobre uma variável numa tabela é através da construção de uma **tabela de frequências**, que informa quais valores ou categorias a variável pode tomar, com suas respectivas frequências. Quando a variável é qualitativa, as frequências vão

[2] Tente contar os números do conjunto $A = \{1, 2, 3, 4\}$ e, também, de $B = [1, 4]$. Observe que no conjunto A há 4 elementos, enquanto em B há infinitos valores de 1 a 4.

revelar se temos categorias mais comuns (típicas) e categorias raras ou se a distribuição é uniforme/homogênea. Já quando a variável é quantitativa, as frequências revelarão os valores típicos e/ou a distribuição dos valores é simétricas ou assimétrica.

Existem alguns tipos de frequência que podem ser utilizados para resumir as informações, dentre eles:

- Frequência absoluta ou simplesmente frequência (f): é a contagem do número de vezes que um valor ou categoria aparece.
- Frequência relativa (fr $= f/n$): quando esse valor é multiplicado por 100, informa a porcentagem do aparecimento de uma determinada categoria sobre o número total de contagem.
- Frequência acumulada (fa): é a frequência acumulada até um valor específico.
- Frequência acumulada relativa (far $=$ fa$/n$): quando esse valor é multiplicado por 100, informa a porcentagem acumulada do aparecimento de uma determinada categoria sobre o número total de contagem.

A Tabela 2.4 contém as informações sobre as frequências de cada espécie de pinguim existente no conjunto de dados *penguins*. As informações básicas que podem ser extraídas é que entre as espécies dentro do estudo sobre pinguins 44.2% são *Adelie*, 36% são *Gentoo* e 19.8% são *Chinstrap*.

Tabela 2.4: Distribuição de frequências para as espécies de Pinguins.

Espécies	Frequência	Frequência absoluta	Porcentagem
Adelie	152	0.442	44.2%
Gentoo	124	0.360	36%
Chinstrap	68	0.198	19.8%
Total	344	1.000	100%

Quando a variável é quantitativa e assume muitos valores distintos, para resumir e capturar o padrão da distribuição, esses valores devem ser agrupados em intervalos. A quantidade de intervalos é arbitrário, no entanto, não pode ser nem muito baixo e nem muito alto. O próximo passo é especificar os intervalos, contando quantos valores aparecem dentro de cada um deles.

Para exemplificar, utilizaremos a variável `body_mass_g` de nosso conjunto de dados sobre os pinguins. Em geral, segue-se os passos:

1. Calcular a amplitude (A) dos dados, que é a diferença entre o maior e menor valor da variável.

```r
penguins %>%
  summarise(
    maior = max(body_mass_g, na.rm = TRUE),
    menor = min(body_mass_g, na.rm = TRUE),
    amplitude = max(body_mass_g, na.rm = TRUE) - min(body_mass_g, na.rm = TRUE))
```

```
# A tibble: 1 x 3
  maior menor amplitude
  <int> <int>     <int>
1  6300  2700      3600
```

```r
# Ou ainda
A <- penguins %>%
  summarise(
    amplitude = diff(range(body_mass_g, na.rm = TRUE))
  )
```

2. Encontrar o comprimento aproximado de cada intervalo, dividindo A pelo número de intervalos desejados.

```r
A/6
```

```
  amplitude
1       600
```

A partir da Tabela 2.5 e pela coluna das frequências relativas, observa-se que a distribuição do peso é assimétrica. Em breve, isso será observado graficamente.

Tabela 2.5: Distribuição de frequências para a massa corporal (g) dos pinguins.

Massa corporal (g)	Frequência	f.r. (%)	f.a.r. (%)
2700 ⊢ 3300	34	9.94%	9.94%
3300 ⊢ 3900	110	32.16%	42.11%
3900 ⊢ 4500	80	23.39%	65.5%
4500 ⊢ 5100	60	17.54%	83.04%
5100 ⊢ 5700	41	11.99%	95.03%
5700 ⊢ 6300	17	4.97%	100%

2.2.2 Tipos de gráficos

As distribuições de frequências podem ser representadas em gráficos, que facilitam a interpretação visual do comportamento dos dados. Os gráficos mais comuns, segundo o tipo de variável, são:

- Barras ou colunas: apropriado para variáveis qualitativas e quantitativas discretas.
- Setores: qualitativas nominais com poucas categorias.
- Histograma: qualitativas contínuas.
- Box-plot: quantitativas.
- Diagrama de dispersão: relaciona duas quantitativas.
- Gráfico de linhas: evolução de quantitativa ao longo do tempo ou espaço.

Com esse conhecimento em mente podemos prosseguir para as construções dos gráficos utilizando o `ggplot2`

2.2.2.1 Visualizando uma única variável

Para visualizar a distribuição de uma variável contínua como a massa corporal, utiliza-se o histograma com a função `geom_histogram()`. Como já foi visto, a escolha do número de colunas é arbitrário e pode afetar significativamente a aparência e a interpretação do gráfico.

```
penguins %>%
  ggplot(mapping = aes(x = body_mass_g))+
    geom_histogram(color = "white", fill = "steelblue",
                   breaks = seq(2700, 6300, by = 600),
                   closed = "left")+
    scale_x_continuous(
      breaks = seq(2700, 6300, by = 600),
      labels = seq(2700, 6300, by = 600),
      limits = c(2700, 6300)
    )+
    labs(
      x = "Massa corporal (g)",
      y = "Contagem"
    )+
    ggthemes::theme_clean()
```

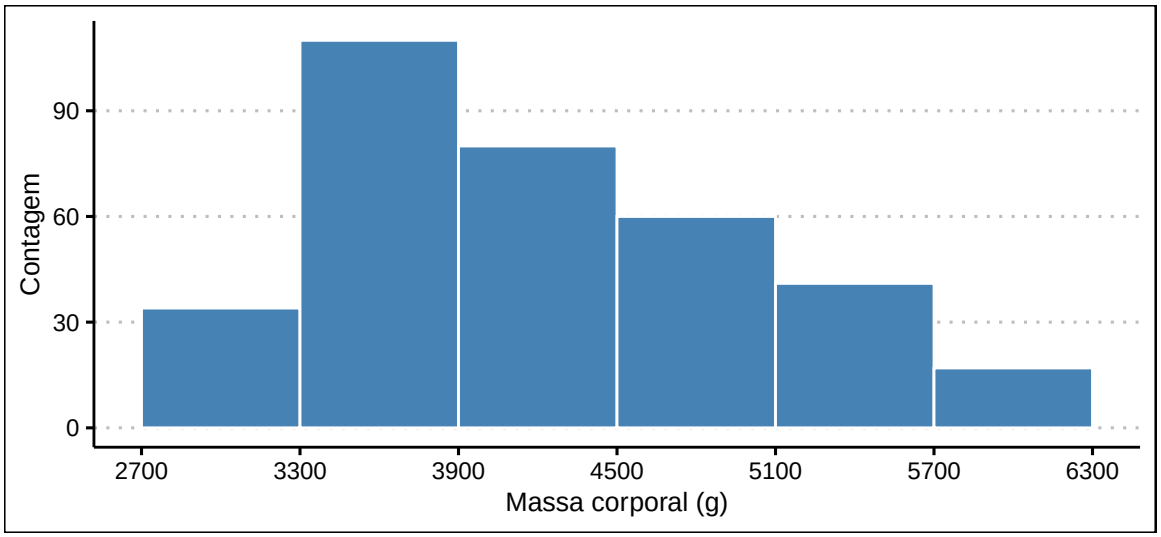


Figura 2.1: Distribuição da massa corporal (g) dos pinguins.

Observe que a biblioteca `ggthemes` foi utilizada para melhorar o aspecto estético do gráfico, fornecendo temas adicionais. Portanto, é recomendável instalá-la e carregá-la no espaço de trabalho.

```
# install.packages("ggthemes")
library(ggthemes)
```

Para variáveis categóricas, como `species`, usamos o `geom_bar()` para criar um gráfico de barras que mostra a contagem de observações em cada categoria.

```
penguins %>%
  ggplot(mapping = aes(x = species, fill = species))+
    geom_bar()+
    labs(
      x = "Espécie",
      y = "Número de Indivíduos"
    )+
    labs(
        x = "Massa corporal (g)",
        y = "Contagem",
        fill = "Espécies"
      )+
    theme_clean()+
    theme(legend.position = "bottom")
```

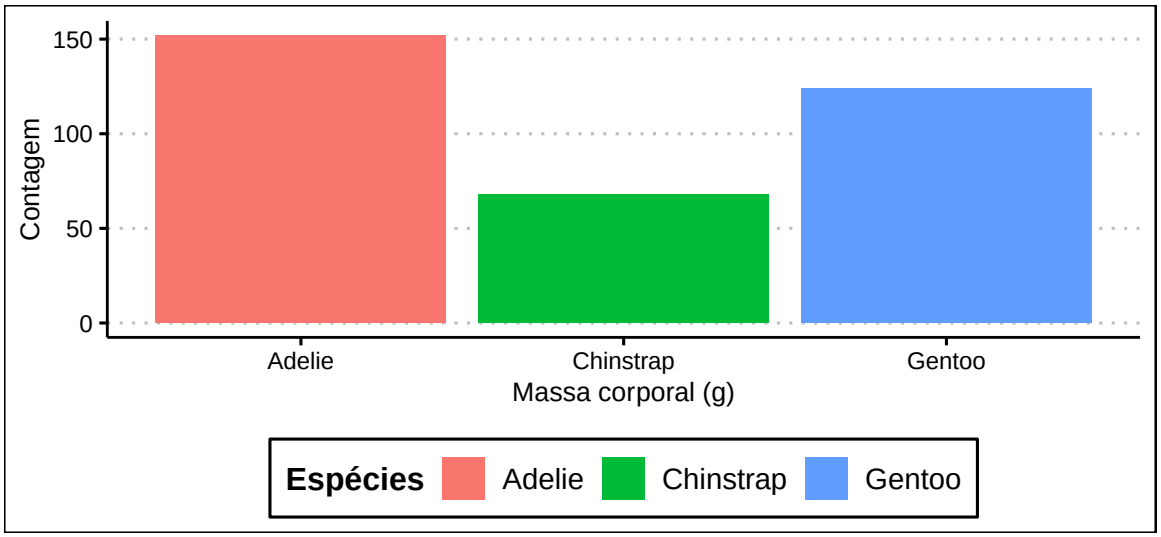


Figura 2.2: Distribuição de Pinguins por Espécie.

2.2.2.2 Relações entre variáveis

O gráfico de dispersão é a ferramenta clássica para explorar relações entre duas variáveis numéricas. Investigaremos a relação existente entre o comprimento da nadadeira e a massa corporal. A hipótese é que pinguins com nadadeiras maiores também serão mais pesados, uma relação positiva e intuitiva que serve como um excelente parâmetro de partida.

```
penguins %>%
  ggplot(aes(x = flipper_length_mm, y = body_mass_g))+
    geom_point()+
    labs(
      x = "Comprimento da Nadadeira (mm)",
      y = "Mass corporal (g)"
    )+
  theme_clean()
```

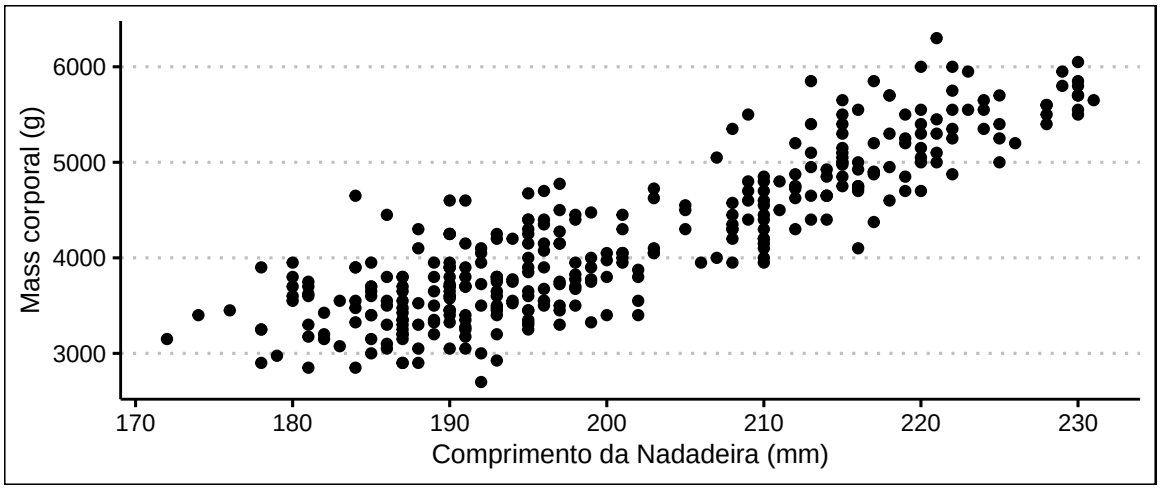


Figura 2.3: Relação entre o tamanho da nadadeira (mm) e o peso corporal (g) dos pinguins.

A Figura 2.3 informa uma clara tendência positiva entre a nadadeira e o peso corporal dos pinguins. Mas será que esse tendência é a mesma para todas as espécies? Para responder essa pergunta, através da função `aes()`, é possível adicionar estéticas adicionais como `shape`, `color` ou `size` para distinguir as espécies no gráfico. O argumento a ser utilizado dependerá onde a imagem será utilizada. Por exemplo, em uma revista científica, que solicita gráficos em preto e branco, é aconselhável utilizar `shape` ou `size`.

```
penguins %>%
  ggplot(aes(x = flipper_length_mm, y = body_mass_g, color = species))+
    geom_point()+
    labs(
      x = "Comprimento da Nadadeira (mm)",
      y = "Massa corporal (g)",
      color = "Espécies"
    )+
    theme_clean()+
    theme(legend.position = "bottom")
```

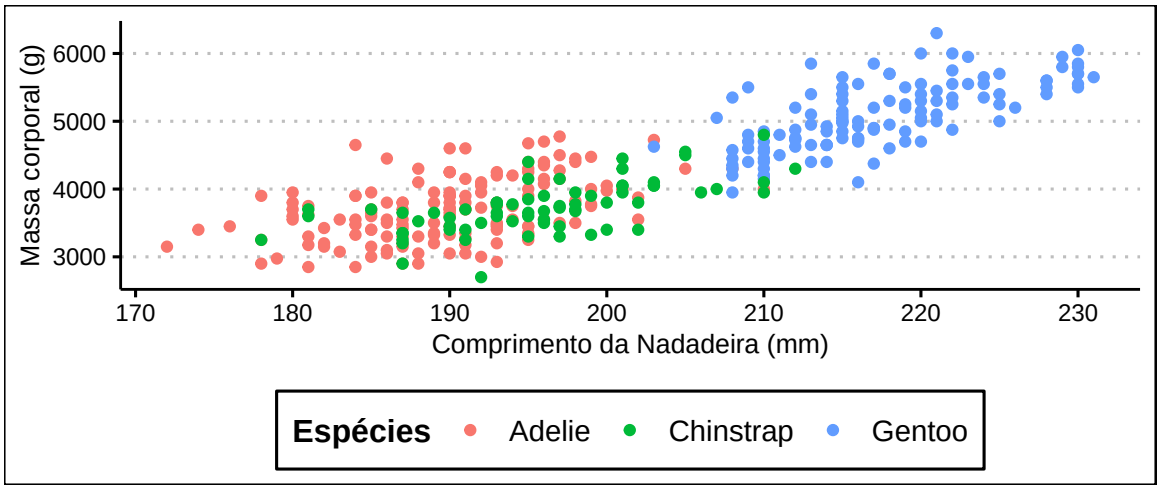


Figura 2.4: Relação entre o tamanho da nadadeira (mm) e o peso corporal (g) por espécie dos pinguins.

A Figura 2.4 revela detalhes mais pertinentes, mostrando que a relação entre massa corporal e comprimento da nadadeira mantêm-se positiva (nível de grupo).

Para comparar a distribuição de uma variável numérica entre diferentes categorias, o boxplot[3] (`geom_boxplot()`) é uma excelente ferramenta. Para isso, vamos comparar a distribuição da massa corporal entre as três espécies.

```
penguins %>%
  ggplot(aes(x = species, y = body_mass_g, fill = species))+
    geom_boxplot()+
    labs(
      x = "Espécies",
      y = "Massa corporal (g)",
      fill = "Espécies"
    )+
    theme_clean()+
    theme(legend.position = "none")
```

[3]Também chamado por gráfico de caixas.

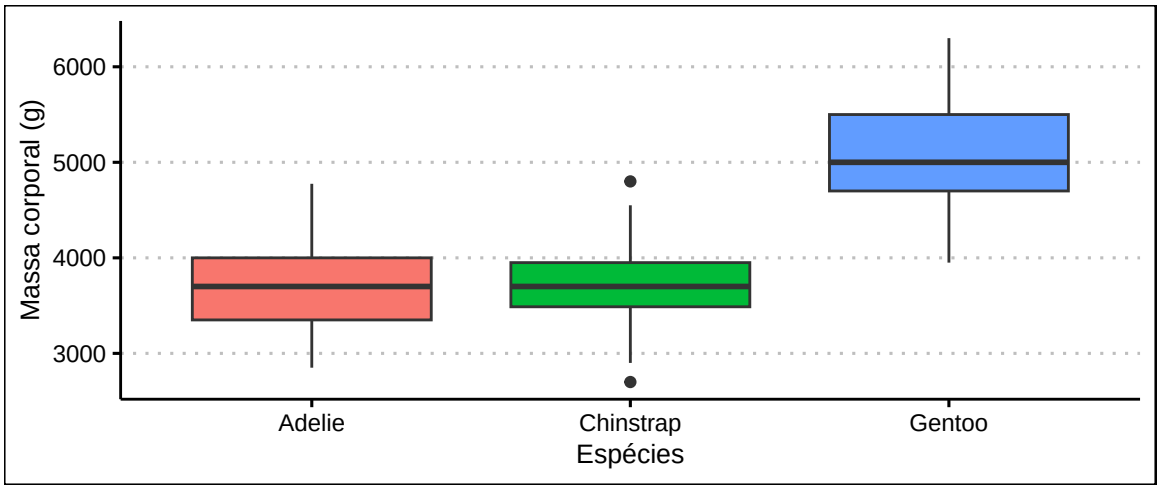
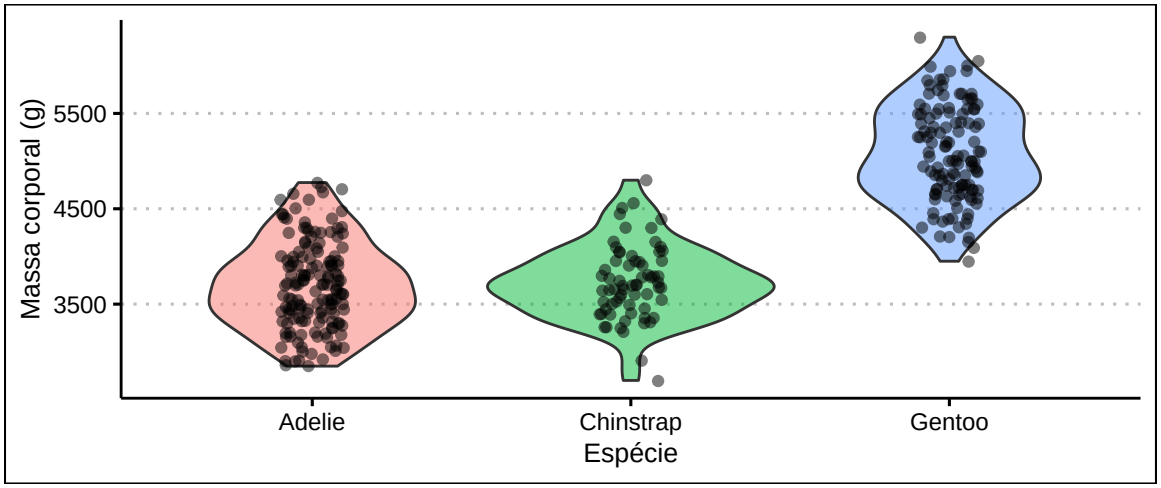


Figura 2.5: Boxplot para a massa corporal (g) por espécies de pinguins.

Uma alternativa ao `geom_boxplot()` é o `geom_violin()`, que traz um detalhamento sobre os dados de uma maneira mais simples. Adicionalmente, podemos utilizar o `geom_jitter()` para evitar a sobreposição dos dados, enriquecendo o visual do gráfico.

```
penguins %>%
  ggplot(aes(x = species, y = body_mass_g, fill = species))+
    geom_violin(alpha = 0.5)+
    geom_jitter(width = 0.1, alpha = 0.5)+
    labs(
      x = "Espécie",
      y = "Massa corporal (g)"
    )+
    theme_clean()+
    theme(legend.position = "none")
```



2.2.3 Técnicas avançadas de visualização e comunicação

Com a base da construção de gráficos vista anteriormente, é possível explorar técnicas para criar gráficos mais ricos e informativos, complementando informações descobertas de forma eficaz.

2.2.4 Sub-gráficos com `facet_wrap()`

As facetas permitem criar uma matriz de gráficos, dividindo os dados com uma base em uma ou mais variáveis categóricas. Isso é extremamente útil para comparações. Para exemplificar, vamos utilizar o gráfico de dispersão e segmentá-lo para a variável `sex`.

```
penguins %>%
  filter(!is.na(sex)) %>%
  mutate(
  sex = recode(sex,
         "female" = "Fêmea",
         "male" = "Macho")
  ) %>%
  ggplot(aes(x = flipper_length_mm, y = body_mass_g, color = species))+
    geom_point()+
    facet_wrap(~ sex)+
    labs(
      x = "Comprimento da nadadeira (mm)",
      y = "Massa corporal (g)",
      color = "Espécies"
    )+
    theme_clean()+
    theme(legend.position = "bottom")
```

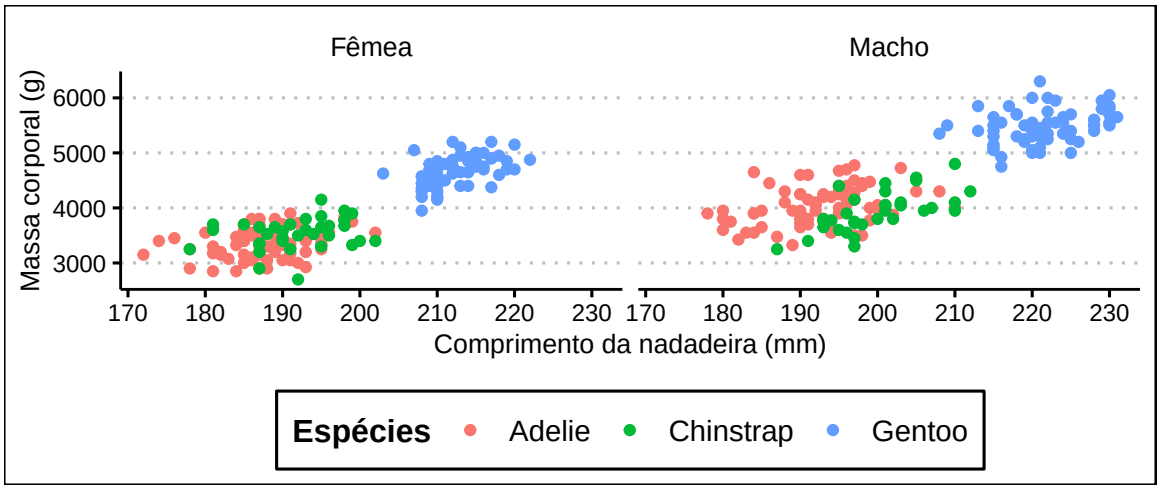


Figura 2.6: Distribuição de massa corporal (g) por espécie.

Neste gráfico, fica evidente que os pinguins fêmeas possuem menos massa corporal que os machos e, quanto as espécies, `Gentoo` é que concentra a maior massa. Contudo, pode ser do interesse do pesquisador além de verificar a massa corporal por sexo, também incluir a variável `island`.

```
penguins %>%
  filter(!is.na(sex)) %>%
  mutate(
  sex = recode(sex,
         "female" = "Fêmea",
         "male" = "Macho")
  ) %>%
  ggplot(aes(x = flipper_length_mm, y = body_mass_g, color = species))+
    geom_point()+
    facet_wrap(island ~ sex)+
    labs(
      x = "Comprimento da nadadeira (mm)",
      y = "Massa corporal (g)",
      color = "Espécies"
    )+
    theme_clean()+
    theme(legend.position = "bottom")
```

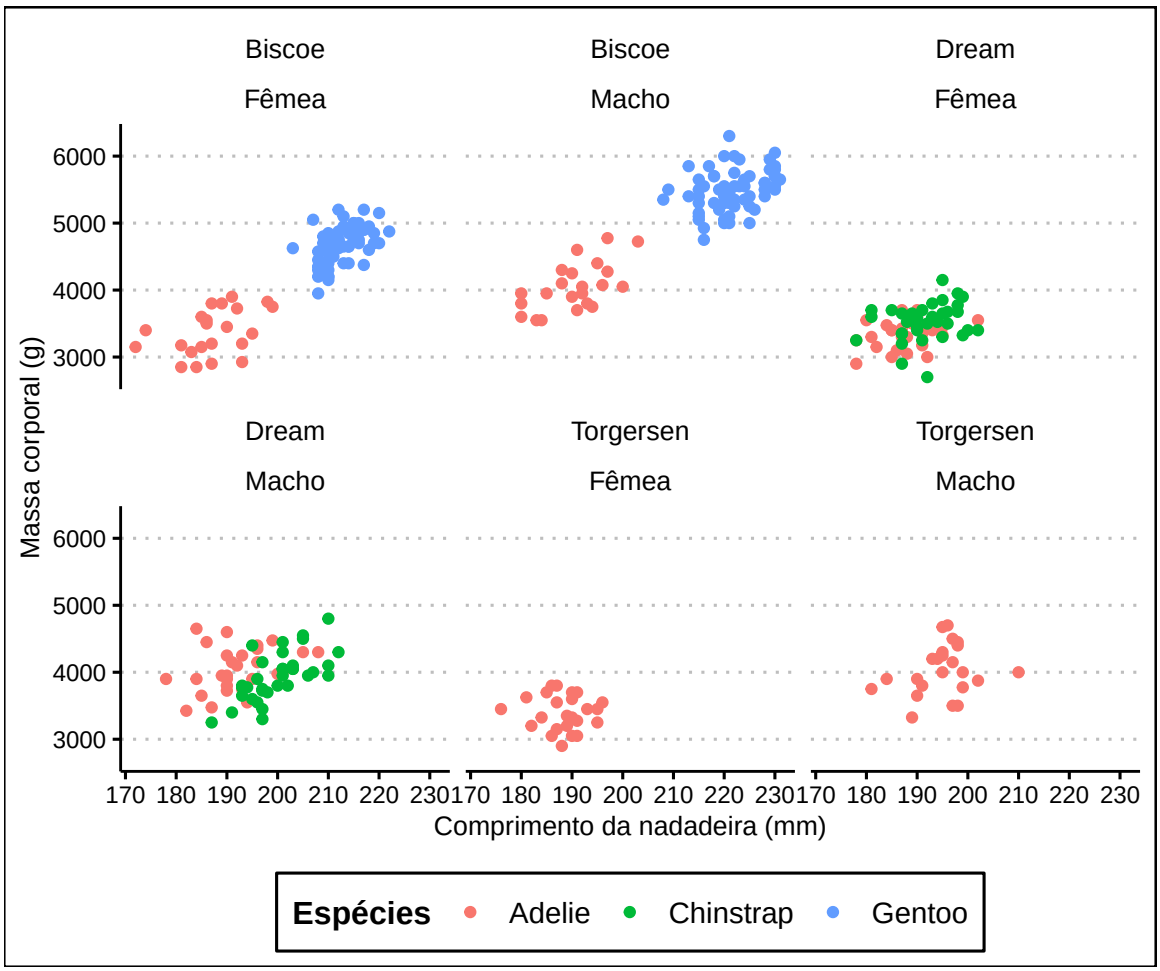


Figura 2.7: Distribuição de massa corporal (g) nas ilhas por espécie.

A partir da Figura 2.7 é possível reparar que nem todas as espécies estão presentes nas três ilhas simultaneamente. Além disso, a ilha de `Biscoe` é a que apresenta o maior percentual de massa corporal dos pinguins, isto é, a espécie `Gentoo` é a predominante.

3 Exercícios

Questão 1 (Estudo de caso)**.** Este estudo de caso é o climax do capítulo. Ele demonstra como a manipulação de dados e a visualização de dados não é apenas para apresentar resultados, mas uma ferramenta de investigação que pode relevar verdades contraintuitívas e protejer de conclusões precipitadas. O Paradoxo de Simpson ocorre quando uma tendência aparece em vários grupos de dados, mas desaparece ou se inverte quando esses grupos são combinados.

Um gráfico para comunicação precisa ser claro, informativo e esteticamente agradável. O ‘ggplot2‘ oferece total controle sobre cada elemento.

- `labs()`: Use para adicionar títulos, subtítulos, legendas e para renomear os eixos de forma clara e concisa.
- `theme()`: Altera a aparência geral do gráfico.
- `scale_*`: Controla como as estéticas são mapeadas. Por exemplos `scale_*` permite a escolha de cores personalizadas.

Faça o que se pede:

(a) Faça o plot do comprimento do bico *versus* a produndidade do bico para todos os pinguins e adicione uma linha de tendência com `geom_smooth(method = 'lm')`.

(b) Refaça o gráfico anterior, mas com o mapeamento com a variávle `species` à estética `color`. O `ggplot2` é inteligente o suficiente para, ao adicionarmos `geom_smooth()`, criar uma linha de tendência separada por cor (ou seja, para cada espécie)

Parte II

Fundamentos do Pensamento Estatístico

4 Princípios de Probabilidade

texto teste $f(x) = ax^2$.

5 A Lógica da Inferência Estatística

Sejam os testes de hipóteses H_0 *vs.* H_A.

6 Delineamento de Experimentos

Um delineamento é…

Parte III

Modelagem Estatística de Dados Biológicos

7 Modelos Lineares – A Base da Modelagem

Um modelo linear é definido por:

$$Y = X\beta + \epsilon$$

8 Entendendo os Modelos Mistos

Os efeitos aleatórios são…

9 Modelos Lineares Mistos com `lme4`

O pacote `lme4` nos permite modelar…

10 Modelos Lineares Generalizados Mistos

Um GLMM é definino como sendo…

11 Validação e Interpretação de Modelos Mistos

Após ajustar os modelos, é boa prática validá-los através de técnicas…

Parte IV

Aplicações Práticas e Recursos

Referências

GORMAN, Kristen B. *et al.* Ecological Sexual Dimorphism and Environmental Variability within a Community of Antarctic Penguins (Genus Pygoscelis). **PLOS ONE**, v. 9, n. 3, p. e90081, 2014.

WICKHAM, Hadley. Tidy Data. **Journal of Statistical Software**, v. 59, p. 1–23, set. 2014.

WICKHAM, Hadley. **Ggplot2**. Cham: Springer International Publishing, 2016.

WICKHAM, Hadley *et al.* Welcome to the Tidyverse. **Journal of Open Source Software**, v. 4, n. 43, p. 1686, nov. 2019.

ZUUR, Alain F. *et al.* **Analysing Ecological Data**. New York, NY: Springer, 2007.