

Introdução ao R

Renato de Paula

2024-09-03

Contents

1	Prefácio	7
2	R e RStudio	9
2.1	Instalação e funcionalidades básicas	9
2.2	Navegando no RStudio	10
2.3	Atalhos	10
3	R como uma calculadora e Operações Aritméticas	13
3.1	O prompt	13
3.2	Objetos e variáveis	14
3.3	Operadores aritméticos em R	17
3.4	Funções <code>print()</code> , <code>readline()</code> , <code>paste()</code> e <code>cat()</code>	22
3.5	Operadores Lógicos e Relacionais	24
3.6	Exercícios	26
4	Estrutura de Dados Básicas	29
4.1	Vetor	29
4.2	Fatores	37
4.3	Matriz e array	38
4.4	Data-frame	45
4.5	Listas	55

5 Estruturas de Seleção	59
5.1 Condicional <code>if</code>	59
5.2 Condicional <code>if...else</code>	59
5.3 Condicional <code>if...else if...else</code>	60
5.4 A função <code>ifelse()</code>	61
5.5 Exemplos	61
5.6 Exercícios	63
6 Funções	65
6.1 Exercícios	69
7 Scripts	71
7.1 Exercícios	72
8 Leitura de dados	73
8.1 Leitura de dados da entrada do usuário	73
8.2 Diretório de trabalho	73
8.3 A Função <code>read.table()</code>	75
8.4 A função <code>read.csv()</code>	76
8.5 A função <code>read.csv2()</code>	76
8.6 A Função <code>read_excel()</code> do pacote <code>readxl</code>	76
8.7 Leitura de Dados Online	77
9 Pipe	79
9.1 O operador pipe	79
9.2 Exercícios	81
10 Loop while	83
10.1 Exercícios	85
11 Loop for	87
11.1 Exercícios	89

<i>CONTENTS</i>	5
12 Família <code>Xapply()</code>	93
12.1 Função <code>apply()</code>	94
12.2 Função <code>lapply()</code>	94
12.3 Função <code>sapply()</code>	95
12.4 Função <code>tapply()</code>	96
12.5 Exercícios	100
13 Gráficos (R base)	101
13.1 Gráfico de Barras	101
13.2 Gráfico circular (pizza)	103
13.3 Histograma	104
13.4 Box-plot	106
14 Manipulação de dados	109
14.1 Tibbles	109
14.2 O pacote <code>dplyr</code>	110
15 Visualização	131
16 O pacote <code>ggplot2</code>	133
17 Simulação	135
17.1 Geração de números pseudoaleatórios	137
17.2 A função <code>sample()</code>	139
17.3 Exercícios	140
18 Método da transformada inversa	143
18.1 Variável aleatória discreta	143
18.2 Variável aleatória contínua	150
19 Método da aceitação-rejeição	155

20 Distribuições univariadas no R	157
20.1 Função de distribuição empírica	161
20.2 Gerando uma variável aleatória com distribuição de Poisson . . .	169
20.3 Gerando uma variável aleatória com distribuição de Uniforme . .	175
20.4 Gerando uma variável aleatória com distribuição Exponencial . .	180
20.5 Gerando uma variável aleatória com distribuição Normal	185
20.6 Exercícios	190
21 Relatórios	197
21.1 Markdown	197
21.2 R Markdown	197
22 Referências	199
23 Respostas	201
23.1 O pacote dplyr	201

Chapter 1

Prefácio

Este material, “Introdução ao R”, foi desenvolvido com o objetivo de servir como um guia acessível e prático para os alunos do curso de Laboratório de Estatística I - Introdução à Simulação da Faculdade de Ciências da Universidade de Lisboa. Reconhecendo a importância cada vez maior da análise de dados na ciência moderna, o material aqui apresentado busca introduzir os conceitos fundamentais e as funcionalidades do R, uma ferramenta poderosa e amplamente utilizada na análise estatística.

Ao longo deste material, os leitores serão guiados através de uma série de tópicos essenciais, desde a instalação do software e a navegação no ambiente RStudio, até o manuseio de estruturas de dados complexas e a criação de gráficos sofisticados. Cada capítulo foi estruturado de forma a proporcionar uma compreensão sólida dos conceitos abordados, combinando explicações teóricas com exemplos práticos e exercícios que reforçam o aprendizado.

Este material foi elaborado para atender às necessidades dos alunos, tanto aqueles que estão iniciando seus estudos em Estatística quanto aqueles que já possuem alguma experiência prévia. Acreditamos que, ao concluir este curso, os alunos estarão bem preparados para aplicar técnicas estatísticas em diversas áreas do conhecimento, utilizando o R como uma ferramenta indispensável para suas análises.

Chapter 2

R e RStudio

O software de código aberto R foi desenvolvido como uma implementação livre da linguagem S, que foi projetada como uma linguagem para computação estatística, programação estatística e gráficos. A intenção principal era permitir aos usuários explorar os dados de uma forma fácil e interativa, apoiada em representações gráficas significativas. O software estatístico R foi originalmente criado por Ross Ihaka e Robert Gentleman (Universidade de Auckland, Nova Zelândia).

R é um conjunto integrado de recursos de software para manipulação de dados, cálculo e exibição gráfica. Ele inclui:

- Manuseio eficaz de dados e facilidade de armazenamento;
- Um conjunto de operadores para cálculos em arrays/matrizes;
- Uma coleção grande, coerente e integrada de ferramentas intermediárias para análise de dados;
- Recursos gráficos para análise e exibição de dados na tela ou em cópia impressa;
- Uma linguagem de programação bem desenvolvida, simples e eficaz que inclui condicionais, loops, funções recursivas definidas pelo usuário e recursos de entrada e saída.

2.1 Instalação e funcionalidades básicas

- A versão “base” do R, ou seja, o software com seus comandos mais relevantes, pode ser baixado em <https://www.r-project.org/>. Após instalar o R, é recomendável instalar também um editor. Um editor permite ao usuário salvar e exibir convenientemente o código R, enviar esse

código ao R Console e controlar as configurações e a saída. Uma escolha popular de editor é o RStudio (gratuito), que pode ser baixado em <https://www.rstudio.com/>.

- Muitos pacotes adicionais escritos pelo usuário estão disponíveis online e podem ser instalados no console R ou usando o menu R. Dentro do console, a função `install.packages("pacote para instalar")` pode ser usada. Observe que é necessária uma conexão com a Internet. Você pode ver todos os pacotes instalados usando a função `installed.packages()`.

2.2 Navegando no RStudio

Existem quatro painéis de trabalho no RStudio:

- **Editor/Scripts.** Este painel é onde os scripts são gravados/carregados e exibidos. Possui realce de sintaxe e preenchimento automático, além de permitir passar o código linha por linha.
- **Console.** É aqui que os comandos são executados e é essencialmente a aparência do console do R básico, só que melhor! O console possui realce de sintaxe, preenchimento de código e interface com outros painéis do RStudio.
- **Environment/Histórico.** A aba do espaço de trabalho exibe informações que normalmente ficam ocultas no R, como dados carregados, funções e outras variáveis. A aba histórico armazena todos os comandos (linhas de código) que foram analisados por meio do R.
- **Último painel.** Este painel inclui a aba de arquivos (lista todos os arquivos no diretório de trabalho atual), a aba de gráficos (quaisquer gráficos), a aba de pacotes (pacotes instalados) e a aba de ajuda (sistema de ajuda html embutido).

2.3 Atalhos

- **CTRL+ENTER:** compila a(s) linha(s) selecionada(s) no script.
- **ALT+-:** cria no script um sinal de atribuição (`<-`).
- **CTRL+SHIFT+M:** (`%>%`) operador pipe.
- **CTRL+1:** altera cursor para o script.
- **CTRL+2:** altera cursor para o console.
- **CTRL+ALT+I:** cria um chunk no R Markdown.

- **CTRL+SHIFT+K**: compila um arquivo no R Markdown.
- **ALT+SHIFT+K**: janela com todos os atalhos disponíveis.

No MacBook, os atalhos geralmente são os mesmos, substituindo o **CTRL** por **command** e o **ALT** por **option**.

Chapter 3

R como uma calculadora e Operações Aritméticas

A estatística tem uma relação estreita com a álgebra: os conjuntos de dados podem ser vistos como matrizes e as variáveis como vetores. O R faz uso dessas estruturas e é por isso que primeiro apresentamos funcionalidades de estrutura de dados antes de explicar alguns dos comandos estatísticos básicos mais relevantes.

3.1 O prompt

O R possui uma interface de linha de comando e aceitará comandos simples. Isso é marcado por um símbolo `>`, chamado **prompt**. Se você digitar um comando e pressionar Enter, o R irá avaliá-lo e imprimir o resultado para você.

```
print("Meu primeiro comando no R!")  
## [1] "Meu primeiro comando no R!"
```

Observe que nestas notas, caixas cinzas são usadas para mostrar o código R digitado no console R. O símbolo `##[1]` é usado para denotar o output do console R.

O caractere `#` marca o início de um comentário. Todos os caracteres até o final da linha são ignorados pelo R. Usamos `#` para comentar nosso código R.

```
# Meu primeiro comando no R!
```

Se soubermos o nome de um comando que gostaríamos de usar e quisermos aprender sobre a sua funcionalidade, basta digitar `?command` no prompt da linha de comando do R que ele exibe uma página de ajuda. Por exemplo

```
?sum
```

exibe uma página de ajuda para a função de soma.

- Usando

```
example(sum)
```

mostra exemplos de aplicação da respetiva função.

3.2 Objetos e variáveis

Um **objeto** em R é uma unidade de armazenamento que contém valores ou funções e pode ser referenciado por um nome. Esses valores podem ser números, caracteres, vetores, matrizes, data frames, listas, ou até mesmo funções. Objetos são criados e manipulados através de comandos e podem ser reutilizados em qualquer parte do código. Tudo o que é criado ou carregado na sessão de R, como dados ou funções, é considerado um objeto.

3.2.1 O que é uma variável?

Refere-se a um **nome** ou um identificador que é atribuído a um objeto. A variável armazena a referência ao objeto em si. Em outras palavras, uma variável é o nome que você usa para acessar os dados ou a função armazenada no objeto.

3.2.2 Atribuições

- A expressão `x<-10` cria uma variável `x` e atribui o valor 10 a `x`. Observe que a variável à esquerda é atribuída ao valor à direita. O lado esquerdo deve conter apenas um único nome de variável.
- Também se pode atribuir usando `=` (ou `->`). Porém, para evitar confusão, é comum usar `<-` para distinguir do operador de igualdade `=`.

```
# Atribuição correta
a <- 10
b <- a + 1
```

```
# Atribuição incorreta
10 = a
a + 2 = 10 # Uma atribuição não é uma equação
```

- O comando `c(1,2,3,4,5)` combina os números 1, 2, 3, 4 e 5 em um vetor.
- Os vetores podem ser atribuídos a um “objeto”. Por exemplo,

```
X <- c(2,12,22,32)
```

atribui um vetor numérico de comprimento 4 ao objeto `X`. Observe que o R diferencia maiúsculas de minúsculas, ou seja, `X` e `x` são dois nomes de variáveis diferentes.

À medida que definimos objetos no console, estamos na verdade alterando o espaço de trabalho. Você pode ver todas as variáveis salvas em seu espaço de trabalho digitando:

```
ls()
```

No RStudio a aba *Environment* mostra os valores.

3.2.3 Regras para definição de variáveis

Os nomes de variáveis em R devem começar com uma letra ou ponto final (seguido de uma letra) e podem conter letras, números, pontos e sublinhados.

- O nome da variável não pode conter espaços ou outro carácter especial (como `@`, `#`, `$`, `%`). Devemos usar apenas letras, números e sublinhados (`_`). Ex: `nome_cliente2`.
- Ao nomear variáveis, você não pode usar palavras reservadas do R. Palavras reservadas são termos que possuem significados específicos e não podem ser redefinidos (por exemplo, `if`, `else`, `for`, `while`, `class`, `FALSE`, `TRUE`, `exp`, `sum`).
- Como já mencionado, o R diferencia letras maiúsculas de minúsculas, o que significa que `fcu` e `Fcu` são tratados como duas variáveis diferentes. É uma convenção comum em R usar letras minúsculas para nomes de variáveis e separar palavras com sublinhados. Ex: `faculdade_de_ciencias`.
- Escolha nomes que descrevam claramente a finalidade da variável para que o código seja mais compreensível. Ex: `nome` em vez de `x`.

```
idade <- 20  
Idade <- 30
```

3.2.4 Tipos de dados

Variáveis em R podem armazenar vários tipos de dados, incluindo:

- **Numeric:** números. Ex: 42, 3.14
- **Character:** sequências de caracteres Ex: "Olá"
- **Logical:** valores booleanos. Ex: TRUE ou FALSE
- **Vectors:** coleções de elementos do mesmo tipo. Ex: c(1, 2, 3), c("a", "b", "c")
- **Data Frames:** estruturas de dados tabulares com linhas e colunas
- **Lists:** coleções de elementos de diferentes tipos
- **Factors:** dados categóricos

```
# Numeric
a <- 3.14

# Character
b <- "Programação R"

# Logical
c <- 3<2

# Vectors
d <- c(1,2,3)
```

3.2.5 Comandos importantes

```
ls() #exibe a lista de variáveis na memória

ls.str() #mostra a estrutura da lista de variáveis na memória

rm(a) #remove um objeto

rm(list=ls()) #remover todos os objetos

save.image('nome-do-arquivo.RData') #salvar
```


3.3 Operadores aritméticos em R

Operador	Descrição	Exemplo
+	adiciona dois valores	5 + 2 resulta em 7
-	subtrai dois valores	5 - 2 resulta em 3
*	multiplica dois valores	5 * 2 resulta em 10
/	divide dois valores (sem arredondamento)	5 / 2 resulta em 2.5
%%/%	realiza divisão inteira	5 %% 2 resulta em 2
%%	retorna o resto da divisão	5 %% 2 resulta em 1
^	realiza exponenciação	5 ^ 2 resulta em 25

Exemplos:

```
1+1
## [1] 2

5-2
## [1] 3

5*21
## [1] 105

sqrt(9)
## [1] 3

3^3
## [1] 27

3**3
## [1] 27

log(9)
## [1] 2.197225

log10(9)
## [1] 0.9542425

exp(1)
## [1] 2.718282

# prioridade de resolução
19 + 26 / 4 - 2 * 10
```

```
## [1] 5.5

((19 + 26) / (4 - 2)) * 10
## [1] 225
```

Ao contrário da função `ls()`, a maioria das funções requer um ou mais *argumentos*. Note que usamos acima as funções predefinidas do R `sqrt()`, `log()`, `log10()` e `exp()` com seus respectivos argumentos.

3.3.1 Quantidade de dígitos

```
exp(1)
## [1] 2.718282

options(digits = 20)
exp(1)
## [1] 2.7182818284590450908

options(digits = 3)
exp(1)
## [1] 2.72
```

3.3.2 Objetos predefinidos, Infinito, indefinido e valores ausentes

Existem vários conjuntos de dados incluídos para os usuários praticarem e testarem funções. Você pode ver todos os conjuntos de dados disponíveis digitando:

```
data()
```

Isso mostra o nome do objeto para cada um dos conjuntos de dados. Esses conjuntos de dados são objetos que podem ser usados simplesmente digitando o seu nome. Por exemplo, se você digitar:

```
co2
```

O R mostrará dados de concentração atmosférica de CO₂ de Mauna Loa.

Outros objetos predefinidos são quantidades matemáticas, como π e ∞ .

```
pi
## [1] 3.14

1/0
## [1] Inf

2*Inf
## [1] Inf

-1/0
## [1] -Inf

0/0
## [1] NaN

0*Inf
## [1] NaN

Inf - Inf
## [1] NaN

sqrt(-1)
## Warning in sqrt(-1): NaNs produced
## [1] NaN

c(1,2,3,NA,5)
## [1] 1 2 3 NA 5

mean(c(1,2,3,NA,5))
## [1] NA

mean(c(1,2,3,NA,5), na.rm = TRUE)
## [1] 2.75

x <- c(1, 2, NaN, 4, 5)
y <- c(1, 2, NA, 4, 5)

# Note que isso não funciona
y == NA
## [1] NA NA NA NA NA

# E isso também não
y == "NA"
## [1] FALSE FALSE NA FALSE FALSE
```

```

is.na(x)
## [1] FALSE FALSE TRUE FALSE FALSE

is.nan(x)
## [1] FALSE FALSE TRUE FALSE FALSE

is.na(y)
## [1] FALSE FALSE TRUE FALSE FALSE

is.nan(y)
## [1] FALSE FALSE FALSE FALSE FALSE

# Operações com NaN e NA
sum(x) # Exibe: NaN, porque a soma envolve um NaN
## [1] NaN

sum(y) # Exibe: NA, porque a soma envolve um NA
## [1] NA

sum(x, na.rm = TRUE) # Exibe: 12, ignora NaN na soma
## [1] 12

sum(y, na.rm = TRUE) # Exibe: 12, ignora NA na soma
## [1] 12

```

- **NaN** significa “**Not a Number**” e é usado para representar resultados indefinidos de operações matemáticas.
- **NA** significa “**Not Available**” e é usado para representar dados ausentes ou valores que não estão disponíveis em um conjunto de dados.

3.3.3 Escrita dinâmica

O R determina dinamicamente o tipo de uma variável com base no valor atribuído a ela.

```

x <- 5
class(x)
## [1] "numeric"

y <- "Cinco"
class(y)
## [1] "character"

```

```
z <- TRUE
class(z)
## [1] "logical"
```

- A função `class()` retorna a classe de um objeto em R. A classe de um objeto determina como ele será tratado pelas funções que operam sobre ele. Por exemplo, vetores, matrizes, data frames e listas são todas classes de objetos em R.
- A função `typeof()` em R é usada para retornar o tipo de armazenamento interno de um objeto. Ela fornece informações detalhadas sobre como os dados são representados na memória.

```
x <- 1:10
class(x)
## [1] "integer"

typeof(x)
## [1] "integer"

y <- c(1.1, 2.2, 3.3)
class(y)
## [1] "numeric"

typeof(y)
## [1] "double"

z <- data.frame(a = 1:3, b = c("A", "B", "C"))
class(z)
## [1] "data.frame"

typeof(z)
## [1] "list"

w <- list(a = 1, b = "text")
class(w)
## [1] "list"

typeof(w)
## [1] "list"
```

3.3.4 Conversão entre tipos de dados

```
# Convertendo inteiro em string
a <- 15
b <- as.character(15)
print(b)
## [1] "15"

# Convertendo float em inteiro
x <- 1.5
y <- as.integer(x)
print(y)
## [1] 1

# Convertendo string em float
z <- "10"
w <- as.numeric(z)
print(w)
## [1] 10
```

3.4 Funções print(), readline(), paste() e cat()

- A função `print()` é utilizada para exibir valores e resultados de expressões no console.
- A função `readline()` é usada para receber entradas do usuário por meio do teclado.
- A função `paste()` é utilizada para concatenar sequências de caracteres (strings) com um separador específico.
- A função `paste0()` é utilizada para concatenar strings sem nenhum separador específico.
- A função `cat()` é usada para concatenar e exibir uma ou mais strings ou valores de uma forma mais direta, sem estruturas de formatação adicionais.

Exemplo 1:

```
nome1 <- "faculdade"
nome2 <- "ciências"
print(paste(nome1, nome2))
## [1] "faculdade ciências"
```

Exemplo 2:

```
# Solicitar entrada do usuário
n <- readline(prompt = "Digite um número: ")

# Converta a entrada em um valor numérico
n <- as.integer(n)

# Imprima o valor no ecrã
print(n+1)
```

Exemplo 3:

```
# Solicitar entrada do usuário
nome <- readline(prompt = "Entre com o seu nome: ")

# Imprima uma mensagem de saudação
cat("Olá, ", nome, " !")
```

Exemplo 4:

```
# Solicitar ao usuário a entrada numérica
idade <- readline(prompt = "Digite a sua idade: ")

# Converta a entrada em um valor numérico
idade <- as.numeric(idade)

# Verifique se a entrada é numérica
if (is.na(idade)) {
  cat("Entrada inválida. Insira um valor numérico.\n")
} else {
  cat("Você tem ", idade, " anos.\n")
}
```

Concatenando duas palavras simples

```
result <- paste("Hello", "World")
print(result)
## [1] "Hello World"
```

Concatenando várias strings

```
result <- paste("Data", "Science", "with", "R")
print(result)
## [1] "Data Science with R"
```

Concatenando com um separador específico

```
result <- paste("2024", "04", "28", sep="-")
print(result)
## [1] "2024-04-28"
```

Concatenando vetor de strings

```
first_names <- c("Anna", "Bruno", "Carlos")
last_names <- c("Smith", "Oliveira", "Santos")
result <- paste(first_names, last_names)
print(result)
## [1] "Anna Smith"      "Bruno Oliveira" "Carlos Santos"
```

Concatene com cada elemento de um vetor

```
numbers <- 1:3
result <- paste("Number", numbers)
print(result)
## [1] "Number 1" "Number 2" "Number 3"
```

Usando paste0() para concatenar sem espaço

```
result <- paste0("Hello", "World")
print(result)
## [1] "HelloWorld"
```

Concatenando strings com números

```
age <- 25
result <- paste("I am", age, "years old")
print(result)
## [1] "I am 25 years old"
```

3.5 Operadores Lógicos e Relacionais

No R, operadores lógicos e relacionais são utilizados para realizar comparações e tomar decisões com base nos resultados dessas comparações. Estes operadores são fundamentais para a construção de estruturas de controle de fluxo, como instruções condicionais (`if`, `else`) e loops (`for`, `while`).

3.5.1 Operadores Lógicos

Os operadores lógicos são usados para combinar ou modificar condições lógicas.

- O operador `&` (E lógico) retorna `TRUE` se todas as expressões forem verdadeiras.
- O operador `|` (OU lógico) retorna `TRUE` se pelo menos uma das expressões for verdadeira.
- O operador `!` (Não lógico) inverte o valor de uma expressão booleana, transformando `TRUE` em `FALSE` e vice-versa.

Exemplos:

```
(5 > 3) & (4 > 2)
## [1] TRUE

(5 < 3) | (4 > 2)
## [1] TRUE

!(5 > 3)
## [1] FALSE
```

3.5.2 Operadores Relacionais

Os operadores relacionais são usados para comparar valores e retornam valores lógicos (`TRUE` ou `FALSE`) com base na comparação.

- `a == b` (“a” é igual a “b”)
- `a != b` (“a” é diferente de “b”)
- `a > b` (“a” é maior que “b”)
- `a < b` (“a” é menor que “b”)
- `a >= b` (“a” é maior ou igual a “b”)
- `a <= b` (“a” é menor ou igual a “b”)
- `is.na(a)` (“a” é missing - ausente/faltante)
- `is.null(a)` (“a” é nulo)

Exemplos:

```
# maior que
2 > 1
## [1] TRUE
```

```

1 > 2
## [1] FALSE

# menor que
1 < 2
## [1] TRUE

# maior ou igual a
0 >= (2+(-2))
## [1] TRUE

# menor ou igual a
1 <= 3
## [1] TRUE

# conjunção E
9 > 11 & 0 < 1
## [1] FALSE

# ou
6 < 5 | 0 > -1
## [1] TRUE

# igual a
1 == 2/2
## [1] TRUE

# diferente de
1 != 2
## [1] TRUE

```

3.6 Exercícios

1. Escreva um programa em R que leia dois inteiros e imprima sua soma, produto, a diferença do primeiro menos o segundo, a divisão do primeiro pelo segundo, o resto da divisão do primeiro pelo segundo, e o primeiro elevado à potência do segundo.
2. Escreva um programa em R que leia dois números de ponto flutuante e imprima sua soma, diferença, produto e o primeiro elevado à potência do segundo.
3. Escreva um programa em R que leia a quilometragem em milhas e a converta para quilômetros. A fórmula é $K = M * 1,609344$.
4. Escreva um programa em R que leia três inteiros correspondentes ao com-

primento, largura e altura de um paralelepípedo e imprima seu volume.

5. Escreva um programa em R que leia três inteiros e imprima sua média.

6. Escreva um programa em R que leia uma temperatura em graus Fahrenheit e imprima a temperatura correspondente em graus Celsius. A fórmula usada para a conversão de Fahrenheit para Celsius é: $C = \frac{F-32}{1,8}$.

7. Escreva um programa em R que leia uma hora no formato de 24 horas e imprima a hora correspondente no formato de 12 horas.

8. Você olha para um relógio e são exatamente 14h. Você definiu um alarme para tocar em 51 horas. A que horas o alarme tocará?

9. Escreva um programa em R que resolva a versão geral do problema acima. Peça ao usuário para inserir a hora atual (em horas) e o número de horas de espera antes que o alarme toque. Seu programa deve imprimir a hora em que o alarme tocará.

10. Escreva um programa que leia um número inteiro fornecido pelo utilizador e verifique se esse número é maior que 10. O programa deve imprimir **TRUE** se o número é maior que 10 ou **FALSE** caso contrário.

11. Escreva um programa que leia dois números fornecidos pelo utilizador e verifique se eles são iguais. O programa deve imprimir **TRUE** se os números são iguais ou **FALSE** caso contrário.

12. Escreva um programa que peça ao utilizador para inserir dois números e verifique se o primeiro número é maior ou igual ao segundo. O programa deve imprimir **TRUE** ou **FALSE**.

13. Escreva um programa que peça ao utilizador para inserir um número e verifique se esse número está entre 0 e 100, inclusive. O programa deve imprimir **TRUE** se o número está no intervalo e **FALSE** caso contrário.

14. Escreva um programa que leia três números fornecidos pelo utilizador e verifique se o primeiro número é menor que o segundo e se o segundo é menor que o terceiro. O programa deve imprimir uma mensagem indicando se a condição é verdadeira ou falsa.

Chapter 4

Estrutura de Dados Básicas

Em R temos objetos que são funções e objetos que são dados.

- Exemplos de funções:
 - `cos()`
 - `print()`
 - `plot()`
 - `integrate()`
- Exemplos de dados:
 - `23`
 - `"Hello"`
 - `TRUE`
 - `c(1,2,3)`
 - `data.frame(nome = c("Alice", "Bob"), idade = c(25, 30))`
 - `list(numero = 42, nome = "Alice", flag = TRUE)`
 - `factor(c("homem", "mulher", "mulher", "homem"))`

4.1 Vetor

Um vetor é uma estrutura de dados básica que pode armazenar uma sequência de objetos do mesmo tipo. Vetores podem conter dados numéricos, caracteres, valores lógicos (`TRUE/FALSE`), números complexos, entre outros.

- Todos os elementos de um vetor devem ser do mesmo tipo.
- Os elementos de um vetor são indexados a partir de 1.

- Vetores podem ser facilmente manipulados e transformados usando uma ampla gama de funções.
- Vetores podem ser criados usando a função `c()` (concatenate).

4.1.1 Tipos Comuns de Vetores

```
# vetor numérico
c(1.1, 2.2, 3.3)
## [1] 1.1 2.2 3.3

# vetor de caracteres
c("a", "b", "c")
## [1] "a" "b" "c"
#ou
c('a','b','c')
## [1] "a" "b" "c"

# vetor lógico
c(TRUE, 1==2)
## [1] TRUE FALSE

# Não podemos ter combinações...
c(3, 1==2, "a") ## Observe que o R simplesmente transformou tudo em characters!
## [1] "3"      "FALSE" "a"
```

4.1.2 Construindo vetores

```
# Inteiros de 1 a 10
x <- 1:10
x
## [1] 1 2 3 4 5 6 7 8 9 10

b <- 10:1
b
## [1] 10 9 8 7 6 5 4 3 2 1

# Sequência de 0 a 50 de 10 em 10
a <- seq(from = 0, to = 50, by=10)
a
## [1] 0 10 20 30 40 50
```

```

# Sequência de 15 números de 0 a 1
y <- seq(0,1, length=15)
y
## [1] 0.0000 0.0714 0.1429 0.2143 0.2857 0.3571 0.4286 0.5000 0.5714 0.6429
## [11] 0.7143 0.7857 0.8571 0.9286 1.0000

# O mesmo número ou o mesmo vetor várias vezes
z <- rep(1:3, times=4)
z
## [1] 1 2 3 1 2 3 1 2 3 1 2 3

# Cada elemento do vetor 4 vezes
t <- rep(1:3, each=4)
t
## [1] 1 1 1 1 2 2 2 2 3 3 3 3

# Combine números, vetores ou ambos em um novo vetor
w <- c(x,z,5)
w
## [1] 1 2 3 4 5 6 7 8 9 10 1 2 3 1 2 3 1 2 3 1 2 3 5

```

4.1.3 Acesso a Elementos de um Vetor

```

# Defina um vetor com inteiros de (-5) a 5 e extraia os números com valor absoluto menor que 3:
x <- (-5):5
x
## [1] -5 -4 -3 -2 -1 0 1 2 3 4 5

# pelo seu índice no vetor:
x[4:8]
## [1] -2 -1 0 1 2

# ou, por seleção negativa (coloque um sinal de menos na frente dos índices que não queremos):
x[-c(1:3,9:11)]
## [1] -2 -1 0 1 2

# todos menos o último
x[-length(x)]
## [1] -5 -4 -3 -2 -1 0 1 2 3 4

# Um vetor lógico pode ser definido por:
index <- abs(x)<3
index

```

```
## [1] FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE

# Agora este vetor pode ser usado para extrair os números desejados:
x[index]
## [1] -2 -1 0 1 2

# Que é a mesma coisa que...
x[abs(x) < 3]
## [1] -2 -1 0 1 2

letters[1:3]
## [1] "a" "b" "c"

letters[c(2,4,6)]
## [1] "b" "d" "f"

LETTERS[1:3]
## [1] "A" "B" "C"

y <- 1:10
y[ (y>5) ] # seleciona qualquer número > 5
## [1] 6 7 8 9 10

y[ (y%2==0) ] # números que são divisíveis por 2
## [1] 2 4 6 8 10

y[ (y%2==1) ] # números que não são divisíveis por 2
## [1] 1 3 5 7 9

y[5] <- NA
y[!is.na(y)] # todos y que não são NA
## [1] 1 2 3 4 6 7 8 9 10
```

4.1.4 Funções Comuns para Vetores

```
num_vector <- c(2.2, 1.1, 3.3)

# Obtém o comprimento de um vetor
length(num_vector)
## [1] 3

# Calcula o máximo
max(num_vector)
```



```
## [1] 3.3

# Calcula o mínimo
min(num_vector)
## [1] 1.1

# Calcula a soma dos elementos de um vetor
sum(num_vector)
## [1] 6.6

# Calcula a média dos elementos de um vetor
mean(num_vector)
## [1] 2.2

# Calcula a mediana
median(num_vector)
## [1] 2.2

# Vetor com mínimo e máximo
range(num_vector)
## [1] 1.1 3.3

# Variância amostral
var(num_vector)
## [1] 1.21

# Quantis
quantile(num_vector,type = 2)
##   0%  25%  50%  75% 100%
## 1.1  1.1  2.2  3.3  3.3

# Soma de todos os elementos até aquele ponto
cumsum(num_vector)
## [1] 2.2 3.3 6.6

# Produto de todos os elementos até aquele ponto
cumprod(num_vector)
## [1] 2.20 2.42 7.99

# Ordena os elementos de um vetor
sort(num_vector)
## [1] 1.1 2.2 3.3

sort(num_vector,decreasing = TRUE)
## [1] 3.3 2.2 1.1
```

```
# Remove elementos duplicados de um vetor
duplicate_vector <- c(1, 2, 2, 3, 3, 3)
unique(duplicate_vector)
## [1] 1 2 3
```

Existe uma função importante chamada `which` para encontrar índices dentro de vetores.

```
y <- c(8,3,5,7,6,6,8,9,2,3,9,4,10,4,11)
```

Suponha que queiramos saber quais elementos de `y` contêm valores maiores que 5. Digitamos:

```
which(y>5)
## [1] 1 4 5 6 7 8 11 13 15
```

Observe que a resposta a essa pergunta é um conjunto de índices. Não usamos índices dentro da função `which` em si. A função é aplicada a todo o array. Para ver os valores de `y` que são maiores que 5, simplesmente digitamos:

```
y[y>5]
## [1] 8 7 6 6 8 9 9 10 11
```

4.1.5 Operações com Vetores

```
# Adição
num_vector + 1
## 3.2 2.1 4.3

# Multiplicação
num_vector * 2
## 4.4 2.2 6.6

# Comparações
num_vector > 2
## TRUE FALSE TRUE

c(2,3,5,7)^2
## 4 9 25 49

c(2,3,5,7)^c(2,3)
```

```
## 4 27 25 343

c(1,2,3,4,5,6)^c(2,3,4)
## 1 8 81 16 125 1296

c(2,3,5,7)^c(2,3,4)
## 4 27 625 49
```

Os últimos quatro comandos mostram a “propriedade de reciclagem” do R. Ele tenta combinar os vetores em relação ao comprimento, se possível. Na verdade,

```
c(2,3,5,7)^c(2,3)
## [1] 4 27 25 343
```

é expandido para

```
c(2,3,5,7)^c(2,3,2,3)
## [1] 4 27 25 343
```

O último exemplo mostra que o R dá um aviso se o comprimento do vetor mais curto não puder ser expandido para o comprimento do vetor mais longo por uma simples multiplicação com um número natural (2, 3, 4,...). Aqui

```
c(2,3,5,7)^c(2,3,4)
## Warning in c(2, 3, 5, 7)^c(2, 3, 4): longer object length is not a multiple of
## shorter object length
## [1] 4 27 625 49
```

é expandido para

```
c(2,3,5,7)^c(2,3,4,2)
## [1] 4 27 625 49
```

de modo que nem todos os elementos de `c(2,3,4)` são “reciclados”.

4.1.6 Exercícios

1. Crie os vetores:

- (a) $(1, 2, 3, \dots, 19, 20)$
- (b) $(20, 19, \dots, 2, 1)$

- (c) $(1, 2, 3, \dots, 19, 20, 19, 18, \dots, 2, 1)$
- (d) $(10, 20, 30, \dots, 90, 10)$
- (e) $(1, 1, \dots, 1, 2, 2, \dots, 2, 3, 3, \dots, 3)$ onde existem 10 ocorrências do 1, 20 ocorrências do 2 e 30 ocorrências do 3.

2. Use a função `paste()` para criar o seguinte vetor de caracteres de tamanho 20:

("nome 1", "nome 2", ..., "nome 20")

3. Crie um vetor x_1 igual a "A" "A" "B" "B" "C" "C" "D" "D" "E" "E"

4. Crie um vetor x_2 igual a "a" "b" "c" "d" "e" "a" "b" "c" "d" "e"

5. Crie um vetor x_3 igual as palavras "uva" 10 vezes, "maçã" 9 vezes, "laranja" 6 vezes e "banana" 1 vez.

6. Crie um vetor de 15 números aleatórios entre 1 e 100 (use a função `sample()`). Ordene esse vetor em ordem crescente e depois em ordem decrescente. Encontre o menor e o maior valor no vetor.

7. Crie um vetor de 20 números aleatórios entre 1 e 50 (use a função `sample()`). Calcule a soma, a média, o desvio padrão e o produto de todos os elementos do vetor.

8. Crie um vetor de 10 números aleatórios entre 1 e 100 (use a função `sample()`). Extraia os elementos do vetor que são maiores que 50. Em seguida, substitua os valores menores que 30 por 0.

9. Crie um vetor de 10 números. Verifique quais elementos são maiores que 5 e quais são pares. Crie um novo vetor que contenha apenas os números que atendem a ambas as condições.

10. Crie um vetor com 10 números inteiros. Multiplique os elementos nas posições 2, 4 e 6 por 2. Substitua o último elemento por 100.

11. Calcule a média dos vetores:

(a) $x = (1, 0, NA, 5, 7)$

(b) $y = (-Inf, 0, 1, NA, 2, 7, Inf)$

12. Crie:

(a) um vetor com valores $e^x \sin(x)$ nos pontos $x = 2, 2.1, 2.2, \dots, 6$

(b) um vetor com valores $\left(3, \frac{3^2}{2}, \frac{3^3}{3}, \dots, \frac{3^{30}}{30}\right)$

13. Calcule:

(a)

$$\sum_{i=10}^{100} i^3 + 4j^2$$

(b)

$$\sum_{i=1}^{25} \frac{2^i}{i} + \frac{3^i}{i^2}$$

4.2 Fatores

Em R, um “factor” (ou “fator”, em português) é uma estrutura de dados usada para representar dados categóricos. Fatores são muito úteis em análises estatísticas e visualizações, pois permitem que você trate dados categóricos de forma eficiente e consistente.

- Fatores têm **níveis**, que são os valores distintos que a variável categórica pode assumir.
- Internamente, os fatores são armazenados como inteiros que correspondem aos níveis, mas são exibidos como rótulos (labels).
- Fatores podem ser ordenados (ordered factors) ou não ordenados (un-ordered factors).

```
# Vetor de dados categóricos
data <- c("low", "medium", "high", "medium", "low", "high")

# Criar um fator
factor_data <- factor(data)

print(factor_data)
## [1] low    medium high    medium low    high
## Levels: high low medium

# Especificar os níveis
factor_data <- factor(data, levels = c("low", "medium", "high"))
print(factor_data)
## [1] low    medium high    medium low    high
## Levels: low medium high

# Criar um fator ordenado
ordered_factor <- factor(data, levels = c("low", "medium", "high"), ordered = TRUE)
print(ordered_factor)
## [1] low    medium high    medium low    high
```

```
## Levels: low < medium < high

# Verificar Níveis
levels(factor_data)
## [1] "low"    "medium" "high"

# Modificar Níveis
levels(factor_data) <- c("Low", "Medium", "High")
print(factor_data)
## [1] Low    Medium High  Medium Low    High
## Levels: Low Medium High
```

A função `gl` ('generate levels') é útil quando você quer codificar vetores longos de níveis de fator.

```
gl(4,3)
## [1] 1 1 1 2 2 2 3 3 3 4 4 4
## Levels: 1 2 3 4

gl(2, 10, labels = c("mulher", "homem"))
## [1] mulher mulher mulher mulher mulher mulher mulher mulher mulher mulher
## [11] homem homem homem homem homem homem homem homem homem homem
## Levels: mulher homem

# Também podemos fazer
as.factor(c(rep("mulher", 10), rep("homem", 10)))
## [1] mulher mulher mulher mulher mulher mulher mulher mulher mulher mulher
## [11] homem homem homem homem homem homem homem homem homem homem
## Levels: homem mulher
```

4.3 Matriz e array

Uma **matriz** é uma coleção de objetos do mesmo tipo (numérico, lógico, etc.) organizada em um formato bidimensional, ou seja, em linhas e colunas.

- **nrow**: corresponde ao número de linhas;
- **ncol**: corresponde ao número de colunas.

```
matrix(c(1,2,3,4,5,6)+exp(1),nrow=2)
##      [,1] [,2] [,3]
## [1,] 3.72 5.72 7.72
## [2,] 4.72 6.72 8.72
```

```

matrix(c(1,2,3,4,5,6)+exp(1),nrow=2) > 6
##      [,1] [,2] [,3]
## [1,] FALSE FALSE TRUE
## [2,] FALSE  TRUE TRUE

# Também podemos criar matrizes de ordem superior
array(c(1:24), dim=c(4,3,2))
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   13   17   21
## [2,]   14   18   22
## [3,]   15   19   23
## [4,]   16   20   24

```

4.3.1 Construindo matrizes

- O comando `rbind` (row bind) é usado para combinar objetos por linhas. Isso significa que os vetores ou matrizes fornecidos serão empilhados verticalmente, criando novas linhas na estrutura de dados resultante.
- O comando `cbind` (column bind) é usado para combinar objetos por colunas. Isso significa que os vetores ou matrizes fornecidos serão combinados horizontalmente, criando novas colunas na estrutura de dados resultante.

Exemplo com vetores

```

# Criar dois vetores
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)

# Combinar os vetores por linhas
result <- rbind(vector1, vector2)
print(result)
##      [,1] [,2] [,3]
## vector1    1    2    3

```

```
## vector2      4      5      6

# Combinar os vetores por colunas
result <- cbind(vector1, vector2)
print(result)
##           vector1 vector2
## [1,]           1         4
## [2,]           2         5
## [3,]           3         6

# Combinando linhas em uma matriz
A <- rbind(1:3, c(1,1,2))
A
##           [,1] [,2] [,3]
## [1,]         1      2      3
## [2,]         1      1      2

# Combinando colunas em uma matriz
B <- cbind(1:3, c(1,1,2))
B
##           [,1] [,2]
## [1,]         1      1
## [2,]         2      1
## [3,]         3      2
```

Exemplo com matrizes

```
# Criar duas matrizes
matrix1 <- matrix(1:6, nrow = 2, ncol = 3)
matrix2 <- matrix(7:12, nrow = 2, ncol = 3)

# Combinar as matrizes por linhas
result <- rbind(matrix1, matrix2)
print(result)
##           [,1] [,2] [,3]
## [1,]         1      3      5
## [2,]         2      4      6
## [3,]         7      9     11
## [4,]         8     10     12

result <- cbind(matrix1, matrix2)
print(result)
##           [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]         1      3      5      7      9     11
## [2,]         2      4      6      8     10     12
```


4.3.2 Índice e índice lógico

```
A<-matrix((-4):5, nrow=2, ncol=5)
A
##           [,1] [,2] [,3] [,4] [,5]
## [1,]    -4   -2    0    2    4
## [2,]    -3   -1    1    3    5

# Acessando as entradas de uma matriz
A[1,2]
## [1] -2

# Valores negativos
A[A<0]
## [1] -4 -3 -2 -1

# Atribuições
A[A<0]<-0
A
##           [,1] [,2] [,3] [,4] [,5]
## [1,]      0    0    0    2    4
## [2,]      0    0    1    3    5

# Selecionando as linhas de uma matriz
A[2,]
## [1] 0 0 1 3 5

# Selecionando as colunas de uma matriz
A[,c(2,4)]
##           [,1] [,2]
## [1,]      0    2
## [2,]      0    3
```

4.3.3 Nomeando linhas e colunas numa matriz

```
x <- matrix(rnorm(12),nrow=4)
x
##           [,1] [,2] [,3]
## [1,] -0.508 -0.523 -0.0258
## [2,]  1.864  2.422  0.3408
## [3,] -0.230  0.314 -1.9076
## [4,] -0.571  0.478 -0.5948
```

```

colnames(x) <- paste("dados",1:3,sep="")
x
##      dados1 dados2 dados3
## [1,] -0.508 -0.523 -0.0258
## [2,]  1.864  2.422  0.3408
## [3,] -0.230  0.314 -1.9076
## [4,] -0.571  0.478 -0.5948

y <- matrix(rnorm(15),nrow=5)
y
##      [,1] [,2] [,3]
## [1,] -0.559 -1.705  0.672
## [2,] -0.796  1.176 -0.566
## [3,]  0.855 -1.474  0.804
## [4,] -0.630 -1.786 -0.973
## [5,]  1.261  0.447  0.285

colnames(y) <- LETTERS[1:ncol(y)]
rownames(y) <- letters[1:nrow(y)]

y
##      A      B      C
## a -0.559 -1.705  0.672
## b -0.796  1.176 -0.566
## c  0.855 -1.474  0.804
## d -0.630 -1.786 -0.973
## e  1.261  0.447  0.285

```

4.3.4 Multiplicação de matrizes

```

M<-matrix(rnorm(20),nrow=4,ncol=5)
N<-matrix(rnorm(15),nrow=5,ncol=3)

M%*%N
##      [,1] [,2] [,3]
## [1,] -1.67927  0.8103 -3.405
## [2,] -0.33112 -0.9712 -2.352
## [3,] -0.83679 -0.2961  0.140
## [4,] -0.00563 -0.0709 -0.113

```

4.3.5 Adicionando linhas e colunas a matriz

```
X <- matrix(c(1,2,3,4,5,6),nrow=2)
X
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6

# Coluna
cbind(X, c(7,8))
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8

# Linha
rbind(X, c(7,8,9))
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## [3,]    7    8    9
```

4.3.6 Algumas outras funções

Seja M uma matriz quadrada.

- dimensão de uma matriz $\rightarrow \text{dim}(M)$
- transposta de uma matriz $\rightarrow \text{t}(M)$
- determinante de uma matriz $\rightarrow \text{det}(M)$
- inversa de uma matriz $\rightarrow \text{solve}(M)$
- autovalores e autovetores $\rightarrow \text{eigen}(M)$
- soma dos elementos de uma matriz $\rightarrow \text{sum}(M)$
- média dos elementos de uma matriz $\rightarrow \text{mean}(M)$
- aplicar uma função a cada linha ou coluna $\rightarrow \text{apply}(M, 1, \text{sum})$ # soma de cada linha
- aplicar uma função a cada linha ou coluna $\rightarrow \text{apply}(M, 2, \text{mean})$ # média de cada coluna

4.3.7 Exercícios

1. Crie uma matriz 3×4 com os números de 1 a 12, preenchendo a matriz por colunas. Exiba a matriz e determine a soma dos elementos da segunda coluna.
2. Crie duas matrizes 2×3 chamadas A e B , cada uma preenchida com números aleatórios inteiros de 1 a 10. Em seguida, some as duas matrizes e multiplique-as elemento por elemento.
3. Crie uma matriz 3×3 chamada M com números aleatórios entre 1 e 9. Crie também um vetor de comprimento 3 chamado v . Realize a multiplicação entre a matriz M e o vetor v (ou seja, $M \times v$).
4. Crie uma matriz 4×2 chamada N com números sequenciais de 1 a 8. Transponha a matriz e calcule a soma dos elementos de cada linha da matriz transposta.
5. Crie uma matriz 3×3 chamada P com valores de sua escolha. Calcule o determinante da matriz P . Caso o determinante seja diferente de zero, calcule também a matriz inversa de P .
6. Crie uma matriz 5×5 chamada Q com números aleatórios de 1 a 25. Extraia a submatriz composta pelas 2ª, 3ª e 4ª linhas e colunas.
7. Crie uma matriz 3×3 com valores sequenciais de 1 a 9. Calcule a soma de todos os elementos da primeira linha e a soma de todos os elementos da terceira coluna.
8. Considere a matriz

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{bmatrix}$$

- (a) Verifique que $A^3 = 0$.
- (b) Troque a terceira coluna pela soma da coluna 1 e coluna 3.
- (c) Considere a matriz

$$M = \begin{bmatrix} 20 & 22 & 23 \\ 34 & 55 & 57 \\ 99 & 97 & 71 \\ 12 & 16 & 19 \\ 10 & 53 & 24 \\ 14 & 21 & 28 \end{bmatrix},$$

e troque os números pares por 0.

4.4 Data-frame

Um data frame em R é uma estrutura de dados bidimensional que é usada para armazenar dados tabulares. Cada coluna em um data frame pode conter valores de diferentes tipos (numéricos, caracteres, fatores, etc.), mas todos os elementos dentro de uma coluna devem ser do mesmo tipo. Um data frame é similar a uma tabela em um banco de dados ou uma planilha em um programa de planilhas como o Excel. Podemos criar data frames lendo dados de arquivos ou usando a função `as.data.frame()` em um conjunto de vetores.

4.4.1 Criando um data frame

```
df <- data.frame(  
  id = 1:4,  
  nome = c("Ana", "Bruno", "Carlos", "Diana"),  
  idade = c(23, 35, 31, 28),  
  salario = c(5000, 6000, 7000, 8000))  
df  
##   id  nome idade  salario  
## 1  1   Ana   23    5000  
## 2  2 Bruno   35    6000  
## 3  3 Carlos  31    7000  
## 4  4 Diana   28    8000  
  
# Comparando com uma matriz  
cbind(id = 1:4,  
  nome = c("Ana", "Bruno", "Carlos", "Diana"),  
  idade = c(23, 35, 31, 28),  
  salario = c(5000, 6000, 7000, 8000))  
##      id nome      idade  salario  
## [1,] "1" "Ana"    "23"   "5000"  
## [2,] "2" "Bruno"  "35"   "6000"  
## [3,] "3" "Carlos" "31"   "7000"  
## [4,] "4" "Diana"  "28"   "8000"
```

4.4.2 Acessando linhas e colunas

```
# Acessando a coluna id  
df[,1]  
## [1] 1 2 3 4
```

```

# Outra forma de acessar a coluna id
df$id
## [1] 1 2 3 4

# Outra forma de acessar a coluna id
df[["id"]]
## [1] 1 2 3 4

# Acessando linhas e colunas por índice
df[1, ] # Primeira linha
##   id nome idade salario
## 1  1  Ana   23   5000

# Segunda coluna
df[, 2]
## [1] "Ana"    "Bruno"  "Carlos" "Diana"

# Elemento na primeira linha, segunda coluna
df[1, 2]
## [1] "Ana"

# Subconjunto das primeiras duas linhas e colunas
df[1:2, 1:2]
##   id nome
## 1  1  Ana
## 2  2 Bruno

# Acessando linhas e colunas por nome
df[1, "nome"] # Elemento na primeira linha, coluna "nome"
## [1] "Ana"

# Colunas "nome" e "idade"
df[c("nome", "idade")]
##   nome idade
## 1  Ana   23
## 2 Bruno  35
## 3 Carlos 31
## 4 Diana  28

```

4.4.3 Adicionando e removendo colunas

```

# Adicionar novas colunas
df$novo_salario <- df$salario * 1.1

```

```
df
##   id  nome idade salario novo_salario
## 1  1   Ana   23   5000         5500
## 2  2 Bruno   35   6000         6600
## 3  3 Carlos  31   7000         7700
## 4  4 Diana   28   8000         8800

# Remover coluna
df$id <- NULL
df
##      nome idade salario novo_salario
## 1   Ana   23   5000         5500
## 2 Bruno   35   6000         6600
## 3 Carlos  31   7000         7700
## 4 Diana   28   8000         8800
```

4.4.4 Fundindo dados

```
df1 <- data.frame(curso=c("PE","LE","CAL"), horas=c(60,75,90))
df1
##   curso horas
## 1    PE    60
## 2    LE    75
## 3    CAL    90

df2 <- data.frame(curso=c("CAL","PE","LE"), creditos=c(8,6,7))
df2
##   curso creditos
## 1    CAL        8
## 2    PE         6
## 3    LE         7

df12 <- merge(df1, df2, by="curso")
df12
##   curso horas creditos
## 1    CAL    90        8
## 2    LE    75         7
## 3    PE    60         6
```

4.4.5 Dimensão, informações de colunas e outros

```

df <- iris
names(df)
## [1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"

class(df$Sepal.Length)
## [1] "numeric"

class(df$Species)
## [1] "factor"

dim(df)
## [1] 150 5

nrow(df)
## [1] 150

ncol(df)
## [1] 5

# Visão geral da estrutura do objeto
str(df)
## 'data.frame': 150 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 .

head(df, 3)
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1 5.1 3.5 1.4 0.2 setosa
## 2 4.9 3.0 1.4 0.2 setosa
## 3 4.7 3.2 1.3 0.2 setosa

tail(df, 5)
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 146 6.7 3.0 5.2 2.3 virginica
## 147 6.3 2.5 5.0 1.9 virginica
## 148 6.5 3.0 5.2 2.0 virginica
## 149 6.2 3.4 5.4 2.3 virginica
## 150 5.9 3.0 5.1 1.8 virginica

```


4.4.6 A função `subset()`

```
df1 <- df[df$Sepal.Width > 3, c("Petal.Width", "Species")]
head(df1)
##   Petal.Width Species
## 1          0.2  setosa
## 3          0.2  setosa
## 4          0.2  setosa
## 5          0.2  setosa
## 6          0.4  setosa
## 7          0.3  setosa

(df2 <- subset(df, Sepal.Width > 3, select = c(Petal.Width, Species)))
##   Petal.Width Species
## 1          0.2  setosa
## 3          0.2  setosa
## 4          0.2  setosa
## 5          0.2  setosa
## 6          0.4  setosa
## 7          0.3  setosa
## 8          0.2  setosa
## 10         0.1  setosa
## 11         0.2  setosa
## 12         0.2  setosa
## 15         0.2  setosa
## 16         0.4  setosa
## 17         0.4  setosa
## 18         0.3  setosa
## 19         0.3  setosa
## 20         0.3  setosa
## 21         0.2  setosa
## 22         0.4  setosa
## 23         0.2  setosa
## 24         0.5  setosa
## 25         0.2  setosa
## 27         0.4  setosa
## 28         0.2  setosa
## 29         0.2  setosa
## 30         0.2  setosa
## 31         0.2  setosa
## 32         0.4  setosa
## 33         0.1  setosa
## 34         0.2  setosa
## 35         0.2  setosa
## 36         0.2  setosa
```

```
## 37      0.2      setosa
## 38      0.1      setosa
## 40      0.2      setosa
## 41      0.3      setosa
## 43      0.2      setosa
## 44      0.6      setosa
## 45      0.4      setosa
## 47      0.2      setosa
## 48      0.2      setosa
## 49      0.2      setosa
## 50      0.2      setosa
## 51      1.4 versicolor
## 52      1.5 versicolor
## 53      1.5 versicolor
## 57      1.6 versicolor
## 66      1.4 versicolor
## 71      1.8 versicolor
## 86      1.6 versicolor
## 87      1.5 versicolor
## 101     2.5 virginica
## 110     2.5 virginica
## 111     2.0 virginica
## 116     2.3 virginica
## 118     2.2 virginica
## 121     2.3 virginica
## 125     2.1 virginica
## 126     1.8 virginica
## 132     2.0 virginica
## 137     2.4 virginica
## 138     1.8 virginica
## 140     2.1 virginica
## 141     2.4 virginica
## 142     2.3 virginica
## 144     2.3 virginica
## 145     2.5 virginica
## 149     2.3 virginica

(df3 <- subset(df, Petal.Width == 0.3, select = -Sepal.Width))
##      Sepal.Length Petal.Length Petal.Width Species
## 7              4.6           1.4         0.3  setosa
## 18             5.1           1.4         0.3  setosa
## 19             5.7           1.7         0.3  setosa
## 20             5.1           1.5         0.3  setosa
## 41             5.0           1.3         0.3  setosa
## 42             4.5           1.3         0.3  setosa
```

```
## 46          4.8          1.4          0.3  setosa

(df4 <- subset(df, select = Sepal.Width:Petal.Width))
##      Sepal.Width Petal.Length Petal.Width
## 1           3.5           1.4           0.2
## 2           3.0           1.4           0.2
## 3           3.2           1.3           0.2
## 4           3.1           1.5           0.2
## 5           3.6           1.4           0.2
## 6           3.9           1.7           0.4
## 7           3.4           1.4           0.3
## 8           3.4           1.5           0.2
## 9           2.9           1.4           0.2
## 10          3.1           1.5           0.1
## 11          3.7           1.5           0.2
## 12          3.4           1.6           0.2
## 13          3.0           1.4           0.1
## 14          3.0           1.1           0.1
## 15          4.0           1.2           0.2
## 16          4.4           1.5           0.4
## 17          3.9           1.3           0.4
## 18          3.5           1.4           0.3
## 19          3.8           1.7           0.3
## 20          3.8           1.5           0.3
## 21          3.4           1.7           0.2
## 22          3.7           1.5           0.4
## 23          3.6           1.0           0.2
## 24          3.3           1.7           0.5
## 25          3.4           1.9           0.2
## 26          3.0           1.6           0.2
## 27          3.4           1.6           0.4
## 28          3.5           1.5           0.2
## 29          3.4           1.4           0.2
## 30          3.2           1.6           0.2
## 31          3.1           1.6           0.2
## 32          3.4           1.5           0.4
## 33          4.1           1.5           0.1
## 34          4.2           1.4           0.2
## 35          3.1           1.5           0.2
## 36          3.2           1.2           0.2
## 37          3.5           1.3           0.2
## 38          3.6           1.4           0.1
## 39          3.0           1.3           0.2
## 40          3.4           1.5           0.2
## 41          3.5           1.3           0.3
```

## 42	2.3	1.3	0.3
## 43	3.2	1.3	0.2
## 44	3.5	1.6	0.6
## 45	3.8	1.9	0.4
## 46	3.0	1.4	0.3
## 47	3.8	1.6	0.2
## 48	3.2	1.4	0.2
## 49	3.7	1.5	0.2
## 50	3.3	1.4	0.2
## 51	3.2	4.7	1.4
## 52	3.2	4.5	1.5
## 53	3.1	4.9	1.5
## 54	2.3	4.0	1.3
## 55	2.8	4.6	1.5
## 56	2.8	4.5	1.3
## 57	3.3	4.7	1.6
## 58	2.4	3.3	1.0
## 59	2.9	4.6	1.3
## 60	2.7	3.9	1.4
## 61	2.0	3.5	1.0
## 62	3.0	4.2	1.5
## 63	2.2	4.0	1.0
## 64	2.9	4.7	1.4
## 65	2.9	3.6	1.3
## 66	3.1	4.4	1.4
## 67	3.0	4.5	1.5
## 68	2.7	4.1	1.0
## 69	2.2	4.5	1.5
## 70	2.5	3.9	1.1
## 71	3.2	4.8	1.8
## 72	2.8	4.0	1.3
## 73	2.5	4.9	1.5
## 74	2.8	4.7	1.2
## 75	2.9	4.3	1.3
## 76	3.0	4.4	1.4
## 77	2.8	4.8	1.4
## 78	3.0	5.0	1.7
## 79	2.9	4.5	1.5
## 80	2.6	3.5	1.0
## 81	2.4	3.8	1.1
## 82	2.4	3.7	1.0
## 83	2.7	3.9	1.2
## 84	2.7	5.1	1.6
## 85	3.0	4.5	1.5
## 86	3.4	4.5	1.6

## 87	3.1	4.7	1.5
## 88	2.3	4.4	1.3
## 89	3.0	4.1	1.3
## 90	2.5	4.0	1.3
## 91	2.6	4.4	1.2
## 92	3.0	4.6	1.4
## 93	2.6	4.0	1.2
## 94	2.3	3.3	1.0
## 95	2.7	4.2	1.3
## 96	3.0	4.2	1.2
## 97	2.9	4.2	1.3
## 98	2.9	4.3	1.3
## 99	2.5	3.0	1.1
## 100	2.8	4.1	1.3
## 101	3.3	6.0	2.5
## 102	2.7	5.1	1.9
## 103	3.0	5.9	2.1
## 104	2.9	5.6	1.8
## 105	3.0	5.8	2.2
## 106	3.0	6.6	2.1
## 107	2.5	4.5	1.7
## 108	2.9	6.3	1.8
## 109	2.5	5.8	1.8
## 110	3.6	6.1	2.5
## 111	3.2	5.1	2.0
## 112	2.7	5.3	1.9
## 113	3.0	5.5	2.1
## 114	2.5	5.0	2.0
## 115	2.8	5.1	2.4
## 116	3.2	5.3	2.3
## 117	3.0	5.5	1.8
## 118	3.8	6.7	2.2
## 119	2.6	6.9	2.3
## 120	2.2	5.0	1.5
## 121	3.2	5.7	2.3
## 122	2.8	4.9	2.0
## 123	2.8	6.7	2.0
## 124	2.7	4.9	1.8
## 125	3.3	5.7	2.1
## 126	3.2	6.0	1.8
## 127	2.8	4.8	1.8
## 128	3.0	4.9	1.8
## 129	2.8	5.6	2.1
## 130	3.0	5.8	1.6
## 131	2.8	6.1	1.9

```
## 132      3.8      6.4      2.0
## 133      2.8      5.6      2.2
## 134      2.8      5.1      1.5
## 135      2.6      5.6      1.4
## 136      3.0      6.1      2.3
## 137      3.4      5.6      2.4
## 138      3.1      5.5      1.8
## 139      3.0      4.8      1.8
## 140      3.1      5.4      2.1
## 141      3.1      5.6      2.4
## 142      3.1      5.1      2.3
## 143      2.7      5.1      1.9
## 144      3.2      5.9      2.3
## 145      3.3      5.7      2.5
## 146      3.0      5.2      2.3
## 147      2.5      5.0      1.9
## 148      3.0      5.2      2.0
## 149      3.4      5.4      2.3
## 150      3.0      5.1      1.8
```

4.4.7 A função `summary()`

A função `summary()` no R é usada para gerar resumos estatísticos de objetos. O comportamento da função varia dependendo do tipo de objeto que você passa para ela, mas geralmente fornece uma visão geral das características principais do objeto.

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
summary(x)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      1.00   3.25   5.50   5.50   7.75   10.00

summary(iris$Sepal.Length)
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      4.30   5.10   5.80   5.84   6.40   7.90

summary(iris)
##      Sepal.Length  Sepal.Width  Petal.Length  Petal.Width  Species
##      Min.      :4.30   Min.      :2.00   Min.      :1.00   Min.      :0.1   setosa      :50
##      1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3   versicolor:50
##      Median :5.80   Median :3.00   Median :4.35   Median :1.3   virginica  :50
##      Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
##      3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
##      Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5
```

4.4.8 Valores faltantes

4.4.9 Exercícios

1. Crie um data frame chamado `estudantes` com as colunas: Nome (caracteres), Idade (números inteiros), Curso (caracteres) e Nota (números decimais). Adicione os dados de cinco estudantes fictícios e exiba o data frame.
2. Utilize o data frame `estudantes` criado no exercício anterior. Acesse a coluna Nota e aumente todas as notas em 0.5. Substitua o valor da coluna Curso para “Estatística” para todos os estudantes com nota superior a 8.0.
3. Dado o data frame `mtcars` embutido no R, filtre as linhas onde o Number of cylinders (`cyl`) é igual a 6 e a potência (`hp`) é maior que 100. Crie um novo data frame com essas informações.
4. Adicione uma nova coluna ao data frame `mtcars` chamada `Peso_KG` que converta o peso dos carros (`wt`) de mil libras para quilogramas (multiplicando por 453.592).

4.5 Listas

Uma lista em R é uma estrutura de dados que permite armazenar elementos de diferentes tipos, como vetores, matrizes, data frames, funções e até outras listas. Essa flexibilidade distingue as listas de outras estruturas, como vetores, que são homogêneos e podem conter apenas elementos de um único tipo.

- **Indexação:** Os elementos de uma lista podem ser acessados usando colchetes duplos `[[]]` ou utilizando o operador `$` para acessar elementos nomeados. Além disso, o índice simples `[]` retorna uma sublista.

```
# Criando uma lista com diferentes tipos de elementos
minha_lista <- list(
  nome = "Estudante",
  idade = 21,
  notas = c(85, 90, 92),
  disciplinas = c("Matemática", "Estatística", "Computação"),
  matriz_exemplo = matrix(1:9, nrow = 3, byrow = TRUE),
  media = function(x) mean(x)
)

# Visualizando a lista
print(minha_lista)
## $nome
## [1] "Estudante"
```

```
##
## $idade
## [1] 21
##
## $notas
## [1] 85 90 92
##
## $disciplinas
## [1] "Matemática" "Estatística" "Computação"
##
## $matriz_exemplo
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
##
## $media
## function(x) mean(x)

# Acessando um elemento pelo nome usando $
print(minha_lista$nome)
## [1] "Estudante"

# Acessando um elemento pelo índice
print(minha_lista[[1]])
## [1] "Estudante"

# Acessando uma sublista
print(minha_lista[1:2])
## $nome
## [1] "Estudante"
##
## $idade
## [1] 21

# Acessando uma parte de um elemento, como o segundo valor do vetor "notas"
print(minha_lista$notas[2])
## [1] 90
```

4.5.1 Exercícios

1. Crie uma lista em R chamada `dados_estudante` que contenha as seguintes informações sobre um estudante:

- Nome: “João”
- Idade: 21
- Notas: Um vetor com as notas em Estatística (85), Matemática (90), e Computação (95).

Depois de criar a lista, acesse e imprima:

1. O nome do estudante.
 2. A idade do estudante.
 3. A nota em Computação.
2. Considere a lista `dados_estudante` criada no exercício anterior. Adicione um novo elemento à lista que contenha o status de aprovação do estudante, com valor “Aprovado”. Em seguida, substitua a nota de Estatística para 88. Por fim, imprima a lista completa.
3. Crie duas listas chamadas `estudante1` e `estudante2` com as mesmas estruturas da lista `dados_estudante`. Em `estudante1`, use os valores:

- Nome: “Maria”
- Idade: 22
- Notas: 78, 85, 90
- Status: “Aprovado”

Em `estudante2`, use os valores:

- Nome: “Carlos”
- Idade: 23
- Notas: 70, 75, 80
- Status: “Aprovado”

Agora, combine `estudante1` e `estudante2` em uma nova lista chamada `turma`, e imprima a lista `turma`.

4. Utilizando a lista `turma` criada no exercício anterior, faça as seguintes operações:

- (a) Extraia e imprima o nome do segundo estudante.
- (b) Calcule a média das notas do primeiro estudante.
- (c) Altere o status do segundo estudante para “Reprovado” e imprima a lista atualizada.

5. Crie uma lista chamada `estatistica_aplicada` que contenha duas listas internas: `turma1` e `turma2`. Cada uma dessas listas internas deve conter as informações de dois estudantes (com as mesmas estruturas utilizadas anteriormente). Por exemplo:

- `turma1`: Contendo `estudante1` e `estudante2`.
- `turma2`: Contendo dois novos estudantes de sua escolha.

Acesse e imprima:

- (a) O nome do primeiro estudante da `turma2`.
- (b) A média das notas do segundo estudante da `turma1`.
- (c) A lista completa `estatistica_aplicada`.

Chapter 5

Estruturas de Seleção

Em R, as estruturas de seleção ou decisão são usadas para controlar o fluxo de execução do código com base em condições específicas. Estas estruturas permitem executar diferentes blocos de código dependendo de valores ou condições lógicas. As estruturas de seleção mais comuns em R são `if`, `else`, `else if`.

5.1 Condicional `if`

A instrução `if` executa um bloco de código se uma condição for verdadeira.

```
# Sintaxe
if (condição) {
    # Código a ser executado se a condição for TRUE
}
```

Exemplo 1:

```
x <- 10

if (x > 5) {
    print("x é maior que 5")
}
## [1] "x é maior que 5"
```

5.2 Condicional `if...else`

A estrutura `if...else` permite executar um bloco de código quando a condição é verdadeira e outro bloco de código quando a condição é falsa.

```
# Sintaxe
if (condição) {
    # Código a ser executado se a condição for TRUE
} else {
    # Código a ser executado se a condição for FALSE
}
```

Exemplo 2:

```
x <- 3

if (x > 5) {
    print("x é maior que 5")
} else {
    print("x não é maior que 5")
}
## [1] "x não é maior que 5"
```

5.3 Condicional if...else if...else

A estrutura if...else if...else permite testar múltiplas condições em sequência. Executa o bloco de código do primeiro teste que resulta em verdadeiro.

```
# Sintaxe
if (condição1) {
    # Código se condição1 for TRUE
} else if (condição2) {
    # Código se condição2 for TRUE
} else {
    # Código se nenhuma condição anterior for TRUE
}
```

Exemplo 3:

```
x <- 7

if (x > 10) {
    print("x é maior que 10")
} else if (x > 5) {
    print("x é maior que 5, mas não maior que 10")
} else {
    print("x não é maior que 5")
}
## [1] "x é maior que 5, mas não maior que 10"
```

5.4 A função ifelse()

A função `ifelse` é uma versão vetorizada de `if...else` que retorna valores dependendo de uma condição. É muito útil para aplicar condições a vetores.

```
# Sintaxe  
resultado <- ifelse(condição, valor_se_true, valor_se_false)
```

Exemplo 4:

```
valores <- c(4, 6, 9, 3)  
resultado <- ifelse(valores > 5, "maior que 5", "não é maior que 5")  
print(resultado)  
## [1] "não é maior que 5" "maior que 5"      "maior que 5"  
## [4] "não é maior que 5"
```

5.5 Exemplos

Exemplo 5: Indique o(os) erro(s) no código abaixo

```
if (x%%2 = 0){  
  print("Par")  
} else {  
  print("Ímpar")  
}
```

Código correto

```
if (x%%2 == 0){  
  print("Par")  
} else {  
  print("Ímpar")  
}
```

Exemplo 6: Indique o(os) erro(s) no código abaixo

```
if (a>0) {  
  print("Positivo")  
  if (a%%5 = 0)  
    print("Divisível por 5")  
} else if (a==0)  
  print("Zero")
```

```
else if {  
    print("Negativo")  
}
```

Código correto

```
if (a>0) {  
    print("Positivo")  
    if (a%5 == 0) {  
        print("Divisível por 5")  
    }  
} else if (a==0) {  
    print("Zero")  
} else {  
    print("Negativo")  
}
```

Exemplo 7: Quais os valores de x e y no final da execução

```
x = 1  
y = 0  
if (x == 1){  
    y = y - 1  
}  
if (y == 1){  
    x = x + 1  
}
```

Exemplo 8:

- Se x=1 qual será o valor de x no final da execução?
- Qual teria de ser o valor de x para que no final da execução fosse -1?
- Há uma parte do programa que nunca é executada: qual é e porquê?

```
if (x == 1){  
    x = x + 1  
    if (x == 1){  
        x = x + 1  
    } else {  
        x = x - 1  
    }  
} else {  
    x = x - 1  
}
```

5.6 Exercícios

1. Escreva um programa em R que leia um número do usuário e exiba “O número é positivo” se for maior que zero.
2. Crie um programa em R que leia um número inteiro do usuário e imprima “Par” se o número for par e “Ímpar” caso contrário.
3. Escreva um programa em R que leia um número e exiba se ele está no intervalo $[0, 10]$, $[11, 20]$, ou maior que 20.
4. Escreva um programa em R que leia a idade de uma pessoa e classifique-a como “Criança”, “Adolescente”, “Adulto” ou “Idoso”. Considere:
 - Criança: 0-12 anos
 - Adolescente: 13-17 anos
 - Adulto: 18-64 anos
 - Idoso: 65 anos ou mais
5. Escreva um programa em R que leia o preço original de um produto e aplique um desconto com base no seguinte critério:
 - Desconto de 5% se o preço for inferior a \$100
 - Desconto de 10% se o preço estiver entre \$100 e \$500
 - Desconto de 15% se o preço for superior a \$500
6. Escreva um programa em R que leia as coordenadas (x, y) de um ponto e determine em qual quadrante o ponto está localizado.
7. Escreva um programa em R que leia um ano e determine se é um ano bissexto. Um ano é bissexto se:
 - É divisível por 4, mas não divisível por 100, exceto quando divisível por 400.
8. Escreva um programa que leia um número inteiro entre 0 e 999, e o descreva em termos do número de centenas, número de dezenas e número de unidades. Por exemplo, para 304 o resultado pretendido é “3 centenas 0 dezenas e 4 unidades”. Caso o número inteiro não pertença ao intervalo $[0;999]$ deverá imprimir um aviso “Número fora do intervalo”.
9. Escreva um programa em R que leia um número inteiro positivo (menor ou igual a 5) e escreva no ecrã a sua representação em numeração romana.
10. Escreva um programa em R que leia quatro números inteiros (um de cada vez) e escreva após cada interação qual o menor número lido até ao momento. Segue-se um exemplo da interação com o computador.

Introduza um numero inteiro: 4

`O menor numero introduzido até agora é 4.`

`Introduza um numero inteiro: 6`

`O menor numero introduzido até agora é 4.`

`Introduza um numero inteiro: 2`

`O menor numero introduzido até agora é 2.`

Chapter 6

Funções

- Uma **função** é um bloco de código que realiza tarefas específicas e que só é executado quando é chamada.
- São reutilizáveis e podem ser chamadas várias vezes dentro de um script.
- Podem ser passados dados para uma função, conhecidos como parâmetros ou argumentos.

```
# Sintaxe
nome_da_funcao <- function(argumentos) {
  # Código da função
  resultado <- ... # Cálculos ou operações
  return(resultado) # Retorno do valor
}
```

- **Nome da Função:** Identificador da função.
- **Argumentos:** Valores de entrada para a função.
- **Corpo da Função:** Bloco de código que realiza operações.
- **Return:** Valor que a função devolve.

As funções têm como objetivo principal a modularização (dividir o código em partes menores e gerenciáveis) e a reutilização de código, facilitando a organização e a legibilidade dos scripts. Ao encapsular um bloco de código em uma função, podemos executá-lo múltiplas vezes com diferentes parâmetros, reduzindo a redundância e o tempo de desenvolvimento.

Exemplo: Função para calcular a área de um objeto retangular

```
calcula_area <- function(largura, altura) {
  area <- largura * altura
  return(area)
}

largura_obj <- as.numeric(readline("Insira a largura (em cm): "))
altura_obj <- as.numeric(readline("Insira a altura (em cm): "))

# Chamada da função
area_obj <- calcula_area(largura_obj, altura_obj)

print(area_obj)
```

- **Variáveis locais:** As variáveis `largura` e `altura` são locais. Estas variáveis só existem quando a função está a ser executada. Quando a execução da função termina, as variáveis locais são destruídas.
- **Variáveis globais:** As variáveis `largura_obj` e `altura_obj` são variáveis globais. Estas variáveis são acessíveis a todo o script e representam a largura e altura do objeto inserido pelo utilizador.
- As variáveis locais e globais devem ter nomes **diferentes** para que o código seja mais legível.

Passagem de argumento: valores de argumentos por omissão (default)

```
calcula_area <- function(largura, altura=2) {
  area <- largura * altura
  return(area)
}

# Chamada da função

area_obj <- calcula_area(4)

print(area_obj)
## [1] 8
```

- Caso a altura seja omitida é considerada por definição o valor 2.
- Como a altura foi omitida o cálculo da área será 4×2

```
calcula_area <- function(largura, altura=2) {
  area <- largura * altura
  return(area)
}
```

```
# Chamada da função
area_obj <- calcula_area(altura=4,largura=3)

print(area_obj)
## [1] 12
```

- Se os argumentos forem passados por palavra chave, a ordem dos argumentos pode ser trocada.

Exemplo:

```
f <- function(x) {
  if (x < 0) {
    stop("Erro: x não pode ser negativo") # Interrompe a função com uma mensagem de erro
  }
  return(sqrt(x))
}

f(-2)
```

- O `stop` é usado para interromper a execução de uma função ou de um script, gerando um erro.
- Ele pode ser usado em qualquer lugar do código, dentro ou fora de loops, para gerar um erro e parar a execução do código.
- Quando `stop` é chamado, ele pode exibir uma mensagem de erro personalizada, e a execução do script ou função é completamente interrompida.

Exercício 1: Qual será o output do script abaixo?

```
x <- 10

minha_funcao <- function() {
  x <- 5
  return(x)
}

print(minha_funcao())
print(x)
```

Exercício 2: Qual é o resultado da chamada da função `dados_estudante`?

```
dados_estudante <- function(nome, altura=167){  
  print(paste("O(A) estudante",nome,"tem",altura,"centímetros de altura."))  
}  
  
dados_estudante("Joana",160)
```

Exercício 3: Qual é a sintaxe correta para definir uma função em R que soma dois números?

- (a) `sum <- function(x, y) {return(x + y)}`
- (b) `function sum(x, y) {return(x + y)}`
- (c) `def sum(x, y) {return(x + y)}`
- (d) `sum(x, y) = function {return(x + y)}`

Exercício 4: Qual das seguintes chamadas à função estão corretas?

```
dados_estudante <- function(nome, altura=167) {  
  print(paste("O(A) estudante",nome,"tem",altura,"centímetros de altura."))  
}  
  
dados_estudante("Joana",160)  
dados_estudante(altura=160, nome="Joana")  
dados_estudante(nome = "Joana", 160)  
dados_estudante(altura=160, "Joana")  
dados_estudante(160)
```

`dados_estudante(160)` - Esta chamada está errada porque 160 será interpretado como nome, e altura usará seu valor padrão, 167. Isso resultará na impressão: "O(A) estudante 160 tem 167 centímetros de altura." A chamada está tecnicamente correta no sentido de sintaxe, mas o resultado não faz sentido lógico, já que 160 não é um nome válido para um estudante.

Exercício 5: Qual o resultado do seguinte programa?

```
adi <- function(a,b) {  
  return(c(a+5, b+5))  
}  
  
resultado <- adi(3,2)
```

6.1 Exercícios

1. Crie uma função chamada `quadrado()` que recebe um único argumento numérico `x` e retorna o quadrado de `x`. Em seguida, teste a função com os valores 3, 5 e 7.
2. Implemente uma função chamada `hipotenusa()` que calcula a hipotenusa de um triângulo retângulo dado os comprimentos dos dois catetos, `a` e `b`. A função deve retornar o comprimento da hipotenusa usando o Teorema de Pitágoras. Teste a função com os catetos de comprimentos 3 e 4.
3. Escreva uma função chamada `divisao_segura()` que aceita dois argumentos numéricos, numerador e denominador, e retorna o resultado da divisão. A função deve verificar se o denominador é zero e retornar uma mensagem de erro apropriada em vez de tentar realizar a divisão. Teste a função com os valores 10 e 2, e novamente com 10 e 0.
4. Escreva uma função que receba dois inteiros e devolva o maior deles. Teste a função escrevendo um programa que recebe dois números inteiros do utilizador e imprime o resultado de chamada à função. Como teria de fazer para determinar o menor de dois números com uma segunda função que tirasse partido de chamar a primeira?
5. Escreva uma função que devolva o maior de três números inteiros, recorrendo à função implementada no exercício anterior. Teste a função escrevendo um programa que recebe três números inteiros do utilizador e imprime o resultado de chamada à função.
6. Crie uma função chamada `analise_vetor()` que recebe um vetor numérico e retorna outro vetor contendo a soma, a média, o desvio padrão e o valor máximo do vetor. Use a função para analisar o vetor `c(4, 8, 15, 16, 23, 42)`. Funções auxiliares `sum()`, `mean()`, `sd()`, `max()`.
7. Implemente uma função chamada `classificar_numero()` que aceita um número inteiro e retorna “positivo”, “negativo”, ou “zero” com base no valor do número fornecido. Teste a função com os valores -5, 0, e 7.
8. Escreva uma função que calcule o IMC (Índice de Massa Corporal) com base no peso (kg) e altura (m) fornecidos como argumentos. Teste a função e escreva um programa que receba a altura e o peso do utilizador e imprima a classificação de acordo com a seguinte tabela:

IMC (kg/m ²)	Classificação
Menor que 18.5	Baixo peso
De 18.5 a 24.9	Peso normal
De 25 a 29.9	Sobrepeso
De 30 a 34.9	Obesidade grau I
De 35 a 39.9	Obesidade grau II

IMC (kg/m ²)	Classificação
Igual ou maior que 40	Obesidade grau III

Lembre que $IMC = \frac{\text{Peso}}{\text{Altura}^2}$.

9. Crie funções em R para converter valores entre diferentes moedas com base em taxas de câmbio. Implemente e teste as funções, e escreva um programa que solicite ao utilizador um valor em euros e imprima a conversão desse valor para algumas das principais moedas, como dólar, libra, iene e real.

Chapter 7

Scripts

Um **script** é um ficheiro que contém um conjunto de definições (variáveis, funções e blocos de código) que podem ser reutilizadas noutros programas R.

Criação do scrip: Para criar um script, basta guardar o seu código num ficheiro R com a extensão “.R”

Exemplo: Guarde o seguinte código no ficheiro `meu_script.R`

```
produto function(x,y){  
  return(x*y)  
}
```

Utilização do script

Para executar o script, use a função `source()` no console do R ou dentro de outro script:

```
source("meu_script.R")
```

Se o ficheiro `meu_script.R` não estiver no diretório de trabalho atual, você pode fornecer o caminho completo:

```
source("/caminho/para/seu/script/meu_script.R")
```

Resultado:

Quando você executa o comando `source()`, o R lê e executa todas as linhas do script, e os resultados (por exemplo, impressões de mensagens, funções...) serão exibidos no console. Qualquer função ou variável definida no script ficará disponível no ambiente de trabalho após a execução do `source()`.

```
# Faça agora  
produto(2,3)
```

Observações

- **Diretório de trabalho:** Para verificar ou alterar o diretório de trabalho em R, você pode usar as funções `getwd()` para ver o diretório atual e `setwd("caminho/do/diretorio")` para definir um novo diretório de trabalho.

7.1 Exercícios

1. Construa o seguinte:

- Um script `quadrado.R` que disponibiliza funções que permitem calcular o perímetro e a área do quadrado dado o comprimento do lado. Use `quadrado.R` num outro script qualquer.
- Um script `estatistica.R` que disponibiliza funções que permitem ordenar uma amostra e calcular a média, calcular a variância e o desvio padrão.
- Um programa que recorrendo ao script anterior, calcula a mediana, variância e o desvio padrão da amostra `amostra <- c(1,2,3,4,5,6,7)`

Chapter 8

Leitura de dados

R é uma linguagem poderosa para análise de dados e oferece várias funções para importar dados de diferentes formatos. Independentemente do formato, o processo básico de leitura de dados em R consiste em:

- Especificar o caminho do arquivo.
- Indicar as características do arquivo (como delimitador, presença de cabeçalhos, etc.).
- Ler os dados e armazená-los em um objeto (geralmente um data frame).

8.1 Leitura de dados da entrada do usuário

A função `scan()` lê dados de entrada de um arquivo ou da entrada padrão e retorna um vetor. Um uso básico do `scan()` é para entrada de dados manual:

```
# Solicita ao usuário para digitar números  
numeros <- scan()
```

Após digitar `scan()`, o usuário pode inserir uma série de números separados por espaços e pressionar Enter para concluir.

8.2 Diretório de trabalho

O working directory (diretório de trabalho) em R é o diretório atual onde o R está configurado para ler e gravar arquivos. Quando você importa dados de um arquivo ou salva um gráfico, por exemplo, o R usa o diretório de trabalho como o ponto de referência para encontrar ou salvar esses arquivos. Ter um diretório

de trabalho configurado corretamente é crucial para garantir que você consiga acessar arquivos de dados e salvar saídas de forma eficiente.

Funções principais: `getwd()` e `setwd()`

- `getwd()`: Retorna o caminho do diretório atual de trabalho.
- `setwd()`: Define o diretório de trabalho para um diretório especificado.

Para saber qual é o diretório de trabalho atual, você pode usar a função `getwd()`:

```
getwd()
```

A saída esperada (por exemplo, em um sistema Windows (Mac)) seria:

```
# Windows
## [1] "C:/Users/Usuario/Documents"

# Mac
## [1] "/Users/Usuario/Documents"
```

Suponha que você queira alterar o diretório de trabalho para uma pasta chamada "ProjetosR" localizada em "C:/Users/Usuario/Documents/ProjetosR". Você pode usar a função `setwd()` para fazer isso:

```
# Define o diretório de trabalho para "C:/Users/Usuario/Documents/ProjetosR"
setwd("C:/Users/Usuario/Documents/ProjetosR")

# Verifica se o diretório de trabalho foi alterado
print(getwd())

# Saída Esperada
## [1] "C:/Users/Usuario/Documents/ProjetosR"
```

Ao definir corretamente o diretório de trabalho, você pode importar arquivos sem precisar especificar o caminho completo. Por exemplo, se você tiver um arquivo "dados.csv" na pasta de "ProjetosR", você pode lê-lo diretamente:

```
# Lê o arquivo "dados.csv" no diretório de trabalho atual
dados <- read.csv("dados.csv")

# Exibe as primeiras linhas do conjunto de dados
head(dados)
```

8.3 A Função `read.table()`

`read.table()` é uma das funções mais versáteis em R para leitura de arquivos de texto. Esta função permite importar arquivos tabulares e configurá-los de acordo com as características do arquivo. Faça o download do arquivo `dados.txt` em [link](#)

```
# Leitura de arquivo txt com read.table()
dados <- read.table(file = "dados.txt", header = TRUE, sep = "")
print(dados)
```

- `file = "dados.txt"`: Estamos especificando o arquivo "data.txt" para leitura.
- `header = TRUE`: Indicamos que a primeira linha do arquivo é o cabeçalho, contendo os nomes das colunas.

`sep = ""`: Como os dados são separados por espaço, definimos o separador como "".

A função `read.table()` tem várias opções de argumentos.

```
args(read.table)
```

```
## function (file, header = FALSE, sep = "", quote = "\"'", dec = ".",
##     numerals = c("allow.loss", "warn.loss", "no.loss"), row.names,
##     col.names, as.is = !stringsAsFactors, tryLogical = TRUE,
##     na.strings = "NA", colClasses = NA, nrows = -1, skip = 0,
##     check.names = TRUE, fill = !blank.lines.skip, strip.white = FALSE,
##     blank.lines.skip = TRUE, comment.char = "#", allowEscapes = FALSE,
##     flush = FALSE, stringsAsFactors = FALSE, fileEncoding = "",
##     encoding = "unknown", text, skipNul = FALSE)
## NULL
```

Alguns importantes são:

- **header**: Especifica se a primeira linha do arquivo contém nomes de coluna (cabeçalho). Se `header = TRUE`, a primeira linha é considerada como cabeçalho e os nomes das colunas são extraídos dessa linha. Se `header = FALSE`, a primeira linha é tratada como dados.
- **sep**: Define o caractere usado para separar os campos (colunas) no arquivo. Por padrão, é uma vírgula (,), mas você pode especificar outros caracteres, como ponto e vírgula (;).

- **dec**: Define o caractere usado para representar o separador decimal nos valores numéricos. Por exemplo, em alguns países, usa-se a vírgula (,), enquanto em outros, o ponto (.).
- **nrows**: Permite especificar o número máximo de linhas a serem lidas do arquivo. Útil quando você deseja ler apenas uma parte do arquivo.
- **na.strings**: Define os valores que devem ser tratados como NA (valores ausentes). Por exemplo, se você tiver “N/A” ou “NA” no arquivo, pode especificá-los aqui.
- **skip**: Indica quantas linhas devem ser ignoradas no início do arquivo antes de começar a leitura. Útil para pular cabeçalhos ou linhas de comentário.
- **comment.char**: Define o caractere usado para indicar comentários no arquivo. Linhas começando com esse caractere serão ignoradas.

Se o arquivo fosse um CSV (valores separados por vírgula), poderíamos usar:

```
dados_csv <- read.table("dados.csv", header = TRUE, sep = ",")
```

8.4 A função `read.csv()`

A função `read.csv()` é otimizada para a leitura de arquivos CSV (Comma-Separated Values). A principal diferença entre `read.table()` e `read.csv()` é que esta última tem o separador padrão como vírgula.

```
# Leitura de arquivo CSV com read.csv
dados_csv <- read.csv("dados.csv", header = TRUE)
```

Os argumentos adicionais são semelhantes aos da função `read.table()`.

8.5 A função `read.csv2()`

A função `read.csv2()` é semelhante à `read.csv()`, mas o separador padrão é um ponto e vírgula.

```
# Leitura de CSV com separador ponto e vírgula
dados_csv2 <- read.csv2("dados.csv2", header = TRUE)
```

8.6 A Função `read_excel()` do pacote `readxl`

Para ler arquivos Excel (.xls e .xlsx), utilizamos a função `read_excel()` do pacote `readxl`.

```
library(readxl)
# Sintaxe básica
read_excel(path, sheet = NULL, range = NULL, col_names = TRUE,
            col_types = NULL, na = "", trim_ws = TRUE, skip = 0, n_max = Inf,
            guess_max = min(1000, n_max), progress = readxl_progress())
```

Principais parâmetros

- **path**: O caminho do arquivo Excel a ser lido. Este é um argumento obrigatório. Pode ser um caminho local ou uma URL.
- **sheet**: O nome ou o índice da aba (planilha) do Excel a ser lida. Se não especificado, a função lerá a primeira aba.
- **range**: Um intervalo específico a ser lido, como "A1:D10". Se NULL (padrão), a função lê toda a aba.
- **col_names**: Um argumento lógico (TRUE ou FALSE) que indica se a primeira linha da planilha deve ser usada como nomes das colunas no data frame. O padrão é TRUE.
- **col_types**: Um vetor de tipos de colunas que podem ser "blank", "numeric", "date", "text", ou "guess". O padrão é NULL, o que permite que a função adivinhe os tipos de coluna automaticamente.
- **na**: Um vetor de strings que devem ser interpretadas como valores ausentes (NA). O padrão é "" (string vazia).
- **trim_ws**: Um argumento lógico (TRUE ou FALSE) que indica se os espaços em branco devem ser removidos do início e do fim dos valores de texto. O padrão é TRUE.
- **skip**: Número de linhas a serem ignoradas antes de começar a leitura. O padrão é 0.
- **n_max**: Número máximo de linhas a serem lidas. O padrão é Inf, o que significa que todas as linhas são lidas.

Vamos considerar um exemplo em que temos um arquivo Excel chamado "instagram.xlsx" com três planilhas: "Sheet1", "Sheet2" e "Sheet3". Este arquivo pode ser baixado em link. Vamos ler a primeira planilha (Sheet1):

```
dados <- read_excel(path = "instagram.xlsx",
                    sheet = "Sheet1",
                    col_names = TRUE,
                    trim_ws = TRUE)
```

8.7 Leitura de Dados Online

É possível ler diretamente dados hospedados em URLs usando funções como `read.table()`.

```
# Leitura de dados online com read.table  
url <- "https://example.com/data.csv"  
dados_online <- read.table(url, header = TRUE, sep = ",")
```

Chapter 9

Pipe

9.1 O operador pipe

O operador pipe (`%>%`) em R é uma ferramenta essencial para escrever código de maneira mais limpa e legível, permitindo que os resultados de uma função sejam passados diretamente como entrada para a próxima função, sem a necessidade de criar variáveis intermediárias.

O operador pipe foi popularizado pelo pacote `magrittr`, desenvolvido por Stefan Milton Bache e lançado em 2014. O nome “magrittr” é uma referência ao artista surrealista René Magritte, famoso pela pintura “Ceci n’est pas une pipe” (Isto não é um cachimbo), que inspirou a ideia de que o operador pipe permite que dados fluam através de uma sequência de operações de maneira intuitiva e sem a necessidade de variáveis temporárias.

Para começar a utilizar o pipe, instale e carregue o pacote `magrittr`.

```
install.packages("magrittr")
library(magrittr)
```

O operador pipe `%>%` permite que o valor de uma expressão à esquerda seja passado como o primeiro argumento para a função à direita. Isto permite que o código seja lido de cima para baixo, em vez de dentro para fora, facilitando a compreensão e o fluxo de dados. Por exemplo:

```
library(magrittr)
x <- c(1, 2, 3, 4)
sqrt(sum(x))
## [1] 3.16
```

```
# Com o pipe
x %>% sum() %>% sqrt()
## [1] 3.16
```

Outro exemplo

```
resultado <- sqrt(sum(log(abs(x))))
```

Com o operador pipe, o mesmo código pode ser reescrito de forma mais clara:

```
resultado <- x %>%
  abs() %>%
  log() %>%
  sum() %>%
  sqrt()
```

Às vezes, desejamos que o resultado do lado esquerdo do operador pipe seja inserido em um argumento diferente do primeiro na função do lado direito. Nesses casos, usamos um ponto (.) como um marcador para indicar onde o valor deve ser colocado.

```
# Queremos que o dataset seja recebido pelo segundo argumento (data=) da função "lm".

airquality %>%
  na.omit() %>%
  lm(Ozone ~ Wind + Temp + Solar.R, data = .) %>%
  summary()
##
## Call:
## lm(formula = Ozone ~ Wind + Temp + Solar.R, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -40.48 -14.22  -3.55   10.10   95.62
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -64.3421    23.0547  -2.79   0.0062 **
## Wind           -3.3336     0.6544  -5.09  1.5e-06 ***
## Temp            1.6521     0.2535   6.52  2.4e-09 ***
## Solar.R         0.0598     0.0232   2.58   0.0112 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```



```
## Residual standard error: 21.2 on 107 degrees of freedom
## Multiple R-squared:  0.606, Adjusted R-squared:  0.595
## F-statistic: 54.8 on 3 and 107 DF,  p-value: <2e-16
```

9.2 Exercícios

1. Reescreva a expressão abaixo utilizando o %>%,

```
round(mean(sum(1:10)/3), digits = 1)
```

Dica: utilize a função `magrittr::divide_by()`, Veja o help da função para mais informações.

```
1:10 %>%
  sum() %>%
  magrittr::divide_by(3) %>%
  mean() %>%
  round(digits=1)
## [1] 18.3
```

2. Reescreva o código abaixo utilizando o %>%.

```
x <- rnorm(100)

x.pos <- x[x>0]

media <- mean(x.pos)

saida <- round(media, 2)
```

Dica: utilize a função `magrittr::extract()`. Veja o help da função para mais informações.

```
set.seed(123)
rnorm(100) %>%
  magrittr::extract(>0) %>%
  mean() %>%
  round(digits = 2)
## [1] 0.79
```

3. Sem rodar, diga qual a saída do código abaixo. Consulte o help das funções caso precise.

```
2 %>%  
  add(2) %>%  
  c(6, NA) %>%  
  mean(na.rm = T) %>%  
  equals(5)
```

- Primeiro, somamos 2 com 2, gerando o valor 4.
- Então colocamos esse valor em um vetor com os valores 6 e NA.
- Em seguida, tiramos a média desse vetor, desconsiderando o NA, obtendo o valor 5.
- Por fim, testemos se o valor é igual a 5, obtendo o valor TRUE.

Chapter 10

Loop while

A instrução `while` em R é uma estrutura de controle de fluxo que permite executar um bloco de código repetidamente, enquanto uma condição especificada for verdadeira. É particularmente útil para situações em que o número de repetições não é conhecido antecipadamente, mas depende de alguma condição lógica.

```
# Sintaxe  
while (condição) {  
  # Bloco de código a ser executado  
}
```

- **condição:** Uma expressão lógica que é avaliada antes de cada iteração do loop. Enquanto essa condição for `TRUE`, o bloco de código dentro do `while` será executado.
- **Bloco de código:** As instruções que devem ser repetidamente executadas enquanto a condição for verdadeira.

Como funciona o while

- **Avaliação da Condição:** Antes de cada execução do bloco de código, a condição é avaliada.
- **Execução do Bloco de Código:** Se a condição for `TRUE`, o bloco de código dentro do `while` é executado.
- **Reavaliação:** Após a execução do bloco de código, a condição é reavaliada. Se continuar a ser `TRUE`, o ciclo se repete. Se a condição for `FALSE`, o loop termina e o controle do programa continua com a próxima instrução após o `while`.

Exemplo 1: Somando números até um limite.

```
limite <- 10
soma <- 0
contador <- 1

while (contador <= limite) {
  soma <- soma + contador
  contador <- contador + 1
}

print(paste("A soma dos números de 1 a", limite, "é:", soma))
## [1] "A soma dos números de 1 a 10 é: 55"
```

Exemplo 2: Escrevendo a tabuada de um número inteiro

```
n <- as.numeric(readline("Digite um número inteiro: "))

print(paste("Tabuada do",n, ":"))
i=1
while (i <= 10){
  print(paste(n,"x",i, "=", n*i))
  i + 1
}
```

Explique porque o programa acima não termina. Qual o erro no nosso código?

Exemplo 3:

```
limite <- 10
soma <- 0
contador <- 1

while (contador <= limite) {
  soma <- soma + contador
  print(contador)
  if (contador == 3){
    break
  }
  contador <- contador + 1
}

## [1] 1
## [1] 2
## [1] 3
```

- **break** é uma instrução utilizada em ciclos para interromper a sua execução (sair de um ciclo antes de ter sido percorrido completamente). Quando o **break** é chamado, o loop é imediatamente interrompido, e o fluxo de execução continua na próxima linha de código após o loop.

Considerações importantes sobre o uso do `while()`

- **Condição de Parada:** É crucial garantir que a condição do `while` se torne `FALSE` em algum ponto para evitar loops infinitos que podem fazer o programa parar de responder.
- **Incremento/Decremento:** Certifique-se de que a variável que controla a condição seja atualizada adequadamente dentro do loop para evitar loops infinitos.
- **Desempenho:** Loops `while` podem ser menos eficientes do que loops vetorizados em R, portanto, para grandes conjuntos de dados, considere outras abordagens, como aplicar funções vetorizadas (`apply`, `lapply`, etc.).

10.1 Exercícios

1. Escreva um programa em R que peça ao utilizador um número inteiro n e escreva no ecrã os números inteiros de 0 a n , um por linha.
2. Escreva um programa em R que peça ao utilizador um número inteiro e escreva no ecrã o seu quadrado, perguntando em seguida se o utilizador quer terminar a utilização do programa.
3. Escreva um programa em R que peça ao utilizador um número inteiro n e escreva no ecrã o resultado do somatório dos primeiros n inteiros.
4. Escreva um programa em R que peça ao utilizador um número inteiro n e escreva no ecrã o número de pares no intervalo de 0 a n .
5. Usar um loop `while` para somar números inteiros positivos até que um número negativo seja inserido.

Escreva um programa em R que leia números do utilizador e some-os. O loop deve terminar quando o usuário inserir um número negativo. O programa deve então imprimir a soma dos números positivos inseridos.

6. Usar um loop `while` para implementar uma contagem regressiva.

Crie um programa em R que peça ao utilizador para inserir um número positivo e, em seguida, conte regressivamente até zero, imprimindo cada número no console.

7. Crie um programa em R que peça ao utilizador para inserir um número negativo e, em seguida, conte progressivamente até zero, imprimindo cada número no console.
8. Crie um programa em R que peça ao utilizador para inserir um número e, em seguida, contar regressivamente até zero, se o número for positivo e progressivamente se o número for negativo, imprimindo cada número no console.
9. Escreva um programa em R que leia um número inteiro não negativo do utilizador e calcule seu fatorial usando um loop `while`. Lembre que $0! = 1$ e que $4! = 4 \cdot 3 \cdot 2 \cdot 1$.
10. Escreva um programa em R que leia um número inteiro positivo e calcule a soma de seus dígitos usando um loop `while`. Por exemplo, $23 \rightarrow 2 + 3 = 5$.
11. Crie um programa em R que gere e imprima os números da sequência de Fibonacci até um valor máximo especificado pelo utilizador. 1 1 2 3 5 8 13...

Chapter 11

Loop for

A instrução `for` em R é uma estrutura de controle de fluxo que permite executar repetidamente um bloco de código para cada elemento em um conjunto de elementos. É especialmente útil para situações em que se conhece o número de iterações a serem realizadas com antecedência. A instrução `for` é amplamente utilizada em R para iterar sobre vetores, listas, data frames e outras estruturas de dados.

```
# Sintaxe
for (variável in sequência) {
  # Bloco de código a ser executado
}
```

- **variável:** Uma variável que assume o valor de cada elemento na sequência em cada iteração do loop.
- **sequência:** Um vetor, lista ou qualquer estrutura de dados sobre a qual se deseja iterar.
- **bloco de código:** O conjunto de instruções que serão executadas para cada elemento da sequência.

Como funciona o `for()`

- **Inicialização:** Antes do loop começar, a variável de controle é inicializada com o primeiro elemento da sequência.
- **Iteração:** Em cada iteração do loop, a variável de controle assume o próximo valor da sequência.
- **Execução do Bloco de Código:** O bloco de código dentro do loop é executado uma vez para cada elemento da sequência.

- **Finalização:** O loop termina quando todos os elementos da sequência forem processados.

Exemplo 1: Imprima os números de 0 a 10 no ecrã.

```
for (i in 0:10) {  
  print(i)  
}  
## [1] 0  
## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7  
## [1] 8  
## [1] 9  
## [1] 10
```

Exemplo 2: Soma dos elementos de um vetor

```
numeros <- c(1, 2, 3, 4, 5)  
soma <- 0  
  
for (num in numeros) {  
  soma <- soma + num  
}  
  
print(paste("A soma dos números é:", soma))  
## [1] "A soma dos números é: 15"
```

Exemplo 3: Uso do `for` com índices. Você também pode usar o loop `for` para iterar sobre índices de vetores ou listas, o que pode ser útil quando se deseja acessar ou modificar elementos em posições específicas. Multiplique por 2 os elementos do vetor.

```
numeros <- c(10, 20, 30, 40, 50)  
  
for (i in 1:length(numeros)) {  
  numeros[i] <- numeros[i] * 2  
}  
  
print("Elementos do vetor multiplicados por 2:")  
## [1] "Elementos do vetor multiplicados por 2:"
```



```
print(numeros)
## [1] 20 40 60 80 100
```

Exemplo 4: Exemplo com matrizes. O loop for também pode ser usado para iterar sobre elementos de uma matriz, seja por linha ou por coluna.

```
matriz <- matrix(1:9, nrow=3, ncol=3)
soma_linhas <- numeric(nrow(matriz))

for (i in 1:nrow(matriz)) {
  soma_linhas[i] <- sum(matriz[i, ])
}

print("Soma dos elementos de cada linha:")
## [1] "Soma dos elementos de cada linha:"

print(soma_linhas)
## [1] 12 15 18
```

Exemplo 5: Cálculo de Médias de Colunas em um Data Frame

```
dados <- data.frame(
  A = c(1, 2, 3),
  B = c(4, 5, 6),
  C = c(7, 8, 9)
)

medias <- numeric(ncol(dados))

for (col in 1:ncol(dados)) {
  medias[col] <- mean(dados[, col])
}

print("Médias das colunas do data frame:")
## [1] "Médias das colunas do data frame:"

print(medias)
## [1] 2 5 8
```

11.1 Exercícios

1. Escreva um programa em R que use um loop for para imprimir todos os números de 1 a 10.

2. Escreva um programa em R que use um loop `for` para imprimir o quadrado de cada número de 1 a 5.
3. Escreva um programa em R que some todos os elementos do vetor `numeros <- c(2, 4, 6, 8, 10)` usando um loop `for`.
4. Escreva um programa em R que calcule a média das notas no vetor `notas <- c(85, 90, 78, 92, 88)` usando um loop `for`.
5. Escreva um programa em R que imprima todos os números ímpares do vetor `valores <- c(1, 3, 4, 6, 9, 11, 14)`. Utilize um loop `for`.
6. Escreva um programa em R que leia um número inteiro e use um loop `for` para imprimir sua tabuada de 1 a 10.
7. Escreva um programa em R que leia um número inteiro e use um loop `for` para dizer se o número é perfeito ou não. Um número perfeito é aquele cuja soma de seus divisores próprios é igual ao próprio número. Exemplo: 6 tem 1,2,3 como divisores próprios e $1+2+3=6$.
8. Escreva um programa em R que use um loop `for` para encontrar e imprimir todos os números perfeitos entre 1 e 1000.
9. Escreva um programa em R que leia um inteiro n e um inteiro m e escreva os números entre 0 e n , inclusive, de m em m . Utilize um ciclo `for` para o efeito.
10. Escreva um programa que calcule a média de um conjunto de notas fornecidas pelo utilizador. O número de notas que irão ser inseridas são previamente indicadas pelo utilizador. Utilize um loop `for` para o efeito. Segue-se um exemplo da interação com o computador.

Digite o número de notas que serão inseridas: 4

Digite a nota: 12

Digite a nota: 13

Digite a nota: 15.5

Digite a nota: 16

A média das notas é 14.125

11. Crie uma matriz 4×4 com números de 1 a 16. Utilizando o comando de repetição `for` calcule:

- (a) a média da terceira coluna da matriz.
- (b) a média da terceira linha da matriz.
- (c) a média de cada coluna da matriz.
- (d) a média de cada linha da matriz.
- (e) a média dos números pares de cada coluna da matriz.

- (f) a média dos números ímpares de cada linha da matriz.

Obs: Use comandos de decisão (`if...else...`) sempre que necessário.

Chapter 12

Família Xapply()

A família Xapply() no R refere-se a um conjunto de funções que são usadas para iterar sobre objetos de forma eficiente, substituindo a necessidade de ciclos explícitos como `for`. Essas funções são muito úteis para realizar operações repetitivas em listas, vetores, matrizes, data frames e outros objetos, de maneira concisa e muitas vezes mais rápida.

Função	Argumentos	Objetivo	Input	Output
<code>apply</code>	<code>apply(x, MARGIN, FUN)</code>	Aplica uma função às linhas ou colunas ou a ambas	Data frame ou matriz	vetor, lista, array
<code>lapply</code>	<code>lapply(x, FUN)</code>	Aplica uma função a todos os elementos da entrada	Lista, vetor ou data frame	lista
<code>sapply</code>	<code>sapply(x, FUN)</code>	Aplica uma função a todos os elementos da entrada	Lista, vetor ou data frame	vetor ou matriz
<code>tapply</code>	<code>tapply(x, INDEX, FUN)</code>	Aplica uma função a cada fator	Vetor ou data frame	array

12.1 Função apply()

Aplica uma função a margens (linhas ou colunas) de uma matriz ou data frame e fornece saída em vetor, lista ou array. É usada para evitar loops (ciclos).

```
# Sintaxe  
apply(X, MARGIN, FUN)
```

- X: A matriz ou data frame.
- MARGIN: Indica se a função deve ser aplicada a linha (1) ou coluna (2).
- FUN: A função a ser aplicada.

Exemplo: Calcular a soma, a média e a raiz quadrada de cada coluna de uma matriz.

```
matriz <- matrix(1:9, nrow = 3)  
  
apply(matriz, 2, sum)  
## [1] 6 15 24  
  
apply(matriz, 2, mean)  
## [1] 2 5 8  
  
f <- function(x) sqrt(x)  
apply(matriz, 2, f)  
##      [,1] [,2] [,3]  
## [1,] 1.00 2.00 2.65  
## [2,] 1.41 2.24 2.83  
## [3,] 1.73 2.45 3.00
```

12.2 Função lapply()

Aplica uma função a cada elemento de uma lista ou vetor e retorna uma lista. É útil quando você precisa manter a estrutura de saída como uma lista.

```
# Sintaxe  
lapply(X, FUN, ...)
```

- X: A lista ou vetor.
- FUN: A função a ser aplicada.

Exemplo 1:

```

nomes <- c("ANA", "JOAO", "PAULO", "FILIPA")
(nomes_minusc <- lapply(nomes, tolower))
## [[1]]
## [1] "ana"
##
## [[2]]
## [1] "joao"
##
## [[3]]
## [1] "paulo"
##
## [[4]]
## [1] "filipa"

str(nomes_minusc)
## List of 4
## $ : chr "ana"
## $ : chr "joao"
## $ : chr "paulo"
## $ : chr "filipa"

```

Exemplo 2:

```

# Aplicar a função sqrt a cada elemento de uma lista
vetor_dados <- list(a = 1:4, b = 5:8)
lapply(vetor_dados, sqrt)
## $a
## [1] 1.00 1.41 1.73 2.00
##
## $b
## [1] 2.24 2.45 2.65 2.83

```

12.3 Função sapply()

Similar ao `lapply()`, aplica uma função a cada elemento de uma lista, vetor ou data frame, mas tenta simplificar o resultado(saída) para um vetor ou matriz.

```

# Sintaxe
sapply(X, FUN, ...)

```

Exemplo

```
dados <- 1:5
f <- function(x) x^2

lapply(dados, f)
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 9
##
## [[4]]
## [1] 16
##
## [[5]]
## [1] 25

sapply(dados, f)
## [1] 1 4 9 16 25
```

12.4 Função tapply()

Aplica uma função a grupos de valores em um vetor. É ideal para operações em subconjuntos de dados categorizados.

```
# Sintaxe
tapply(X, INDEX, FUN, ...)
```

- X: O vetor de dados
- INDEX: Um fator ou lista de fatores que definem os grupos.
- FUN: A função a ser aplicada

Exemplo: O dataset `iris` no R é um dos conjuntos de dados mais conhecidos e frequentemente utilizados para exemplificar análises estatísticas e técnicas de aprendizado de máquina. Foi introduzido por Ronald A. Fisher em 1936 em seu artigo sobre a utilização de modelos estatísticos para discriminação de espécies de plantas. O objetivo deste conjunto de dados é prever a classe de cada uma das três espécies de flores (fatores): Setosa, Versicolor, Virginica. O conjunto de dados coleta informações para cada espécie sobre seu comprimento e largura.


```
iris
##      Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5         1.4         0.2    setosa
## 2           4.9         3.0         1.4         0.2    setosa
## 3           4.7         3.2         1.3         0.2    setosa
## 4           4.6         3.1         1.5         0.2    setosa
## 5           5.0         3.6         1.4         0.2    setosa
## 6           5.4         3.9         1.7         0.4    setosa
## 7           4.6         3.4         1.4         0.3    setosa
## 8           5.0         3.4         1.5         0.2    setosa
## 9           4.4         2.9         1.4         0.2    setosa
## 10          4.9         3.1         1.5         0.1    setosa
## 11          5.4         3.7         1.5         0.2    setosa
## 12          4.8         3.4         1.6         0.2    setosa
## 13          4.8         3.0         1.4         0.1    setosa
## 14          4.3         3.0         1.1         0.1    setosa
## 15          5.8         4.0         1.2         0.2    setosa
## 16          5.7         4.4         1.5         0.4    setosa
## 17          5.4         3.9         1.3         0.4    setosa
## 18          5.1         3.5         1.4         0.3    setosa
## 19          5.7         3.8         1.7         0.3    setosa
## 20          5.1         3.8         1.5         0.3    setosa
## 21          5.4         3.4         1.7         0.2    setosa
## 22          5.1         3.7         1.5         0.4    setosa
## 23          4.6         3.6         1.0         0.2    setosa
## 24          5.1         3.3         1.7         0.5    setosa
## 25          4.8         3.4         1.9         0.2    setosa
## 26          5.0         3.0         1.6         0.2    setosa
## 27          5.0         3.4         1.6         0.4    setosa
## 28          5.2         3.5         1.5         0.2    setosa
## 29          5.2         3.4         1.4         0.2    setosa
## 30          4.7         3.2         1.6         0.2    setosa
## 31          4.8         3.1         1.6         0.2    setosa
## 32          5.4         3.4         1.5         0.4    setosa
## 33          5.2         4.1         1.5         0.1    setosa
## 34          5.5         4.2         1.4         0.2    setosa
## 35          4.9         3.1         1.5         0.2    setosa
## 36          5.0         3.2         1.2         0.2    setosa
## 37          5.5         3.5         1.3         0.2    setosa
## 38          4.9         3.6         1.4         0.1    setosa
## 39          4.4         3.0         1.3         0.2    setosa
## 40          5.1         3.4         1.5         0.2    setosa
## 41          5.0         3.5         1.3         0.3    setosa
## 42          4.5         2.3         1.3         0.3    setosa
## 43          4.4         3.2         1.3         0.2    setosa
```

```
## 44      5.0      3.5      1.6      0.6      setosa
## 45      5.1      3.8      1.9      0.4      setosa
## 46      4.8      3.0      1.4      0.3      setosa
## 47      5.1      3.8      1.6      0.2      setosa
## 48      4.6      3.2      1.4      0.2      setosa
## 49      5.3      3.7      1.5      0.2      setosa
## 50      5.0      3.3      1.4      0.2      setosa
## 51      7.0      3.2      4.7      1.4 versicolor
## 52      6.4      3.2      4.5      1.5 versicolor
## 53      6.9      3.1      4.9      1.5 versicolor
## 54      5.5      2.3      4.0      1.3 versicolor
## 55      6.5      2.8      4.6      1.5 versicolor
## 56      5.7      2.8      4.5      1.3 versicolor
## 57      6.3      3.3      4.7      1.6 versicolor
## 58      4.9      2.4      3.3      1.0 versicolor
## 59      6.6      2.9      4.6      1.3 versicolor
## 60      5.2      2.7      3.9      1.4 versicolor
## 61      5.0      2.0      3.5      1.0 versicolor
## 62      5.9      3.0      4.2      1.5 versicolor
## 63      6.0      2.2      4.0      1.0 versicolor
## 64      6.1      2.9      4.7      1.4 versicolor
## 65      5.6      2.9      3.6      1.3 versicolor
## 66      6.7      3.1      4.4      1.4 versicolor
## 67      5.6      3.0      4.5      1.5 versicolor
## 68      5.8      2.7      4.1      1.0 versicolor
## 69      6.2      2.2      4.5      1.5 versicolor
## 70      5.6      2.5      3.9      1.1 versicolor
## 71      5.9      3.2      4.8      1.8 versicolor
## 72      6.1      2.8      4.0      1.3 versicolor
## 73      6.3      2.5      4.9      1.5 versicolor
## 74      6.1      2.8      4.7      1.2 versicolor
## 75      6.4      2.9      4.3      1.3 versicolor
## 76      6.6      3.0      4.4      1.4 versicolor
## 77      6.8      2.8      4.8      1.4 versicolor
## 78      6.7      3.0      5.0      1.7 versicolor
## 79      6.0      2.9      4.5      1.5 versicolor
## 80      5.7      2.6      3.5      1.0 versicolor
## 81      5.5      2.4      3.8      1.1 versicolor
## 82      5.5      2.4      3.7      1.0 versicolor
## 83      5.8      2.7      3.9      1.2 versicolor
## 84      6.0      2.7      5.1      1.6 versicolor
## 85      5.4      3.0      4.5      1.5 versicolor
## 86      6.0      3.4      4.5      1.6 versicolor
## 87      6.7      3.1      4.7      1.5 versicolor
## 88      6.3      2.3      4.4      1.3 versicolor
```

## 89	5.6	3.0	4.1	1.3 versicolor
## 90	5.5	2.5	4.0	1.3 versicolor
## 91	5.5	2.6	4.4	1.2 versicolor
## 92	6.1	3.0	4.6	1.4 versicolor
## 93	5.8	2.6	4.0	1.2 versicolor
## 94	5.0	2.3	3.3	1.0 versicolor
## 95	5.6	2.7	4.2	1.3 versicolor
## 96	5.7	3.0	4.2	1.2 versicolor
## 97	5.7	2.9	4.2	1.3 versicolor
## 98	6.2	2.9	4.3	1.3 versicolor
## 99	5.1	2.5	3.0	1.1 versicolor
## 100	5.7	2.8	4.1	1.3 versicolor
## 101	6.3	3.3	6.0	2.5 virginica
## 102	5.8	2.7	5.1	1.9 virginica
## 103	7.1	3.0	5.9	2.1 virginica
## 104	6.3	2.9	5.6	1.8 virginica
## 105	6.5	3.0	5.8	2.2 virginica
## 106	7.6	3.0	6.6	2.1 virginica
## 107	4.9	2.5	4.5	1.7 virginica
## 108	7.3	2.9	6.3	1.8 virginica
## 109	6.7	2.5	5.8	1.8 virginica
## 110	7.2	3.6	6.1	2.5 virginica
## 111	6.5	3.2	5.1	2.0 virginica
## 112	6.4	2.7	5.3	1.9 virginica
## 113	6.8	3.0	5.5	2.1 virginica
## 114	5.7	2.5	5.0	2.0 virginica
## 115	5.8	2.8	5.1	2.4 virginica
## 116	6.4	3.2	5.3	2.3 virginica
## 117	6.5	3.0	5.5	1.8 virginica
## 118	7.7	3.8	6.7	2.2 virginica
## 119	7.7	2.6	6.9	2.3 virginica
## 120	6.0	2.2	5.0	1.5 virginica
## 121	6.9	3.2	5.7	2.3 virginica
## 122	5.6	2.8	4.9	2.0 virginica
## 123	7.7	2.8	6.7	2.0 virginica
## 124	6.3	2.7	4.9	1.8 virginica
## 125	6.7	3.3	5.7	2.1 virginica
## 126	7.2	3.2	6.0	1.8 virginica
## 127	6.2	2.8	4.8	1.8 virginica
## 128	6.1	3.0	4.9	1.8 virginica
## 129	6.4	2.8	5.6	2.1 virginica
## 130	7.2	3.0	5.8	1.6 virginica
## 131	7.4	2.8	6.1	1.9 virginica
## 132	7.9	3.8	6.4	2.0 virginica
## 133	6.4	2.8	5.6	2.2 virginica

```
## 134      6.3      2.8      5.1      1.5 virginica
## 135      6.1      2.6      5.6      1.4 virginica
## 136      7.7      3.0      6.1      2.3 virginica
## 137      6.3      3.4      5.6      2.4 virginica
## 138      6.4      3.1      5.5      1.8 virginica
## 139      6.0      3.0      4.8      1.8 virginica
## 140      6.9      3.1      5.4      2.1 virginica
## 141      6.7      3.1      5.6      2.4 virginica
## 142      6.9      3.1      5.1      2.3 virginica
## 143      5.8      2.7      5.1      1.9 virginica
## 144      6.8      3.2      5.9      2.3 virginica
## 145      6.7      3.3      5.7      2.5 virginica
## 146      6.7      3.0      5.2      2.3 virginica
## 147      6.3      2.5      5.0      1.9 virginica
## 148      6.5      3.0      5.2      2.0 virginica
## 149      6.2      3.4      5.4      2.3 virginica
## 150      5.9      3.0      5.1      1.8 virginica

tapply(iris$Petal.Length, iris$Species, mean)
##      setosa versicolor virginica
##      1.46      4.26      5.55
```

12.5 Exercícios

Chapter 13

Gráficos (R base)

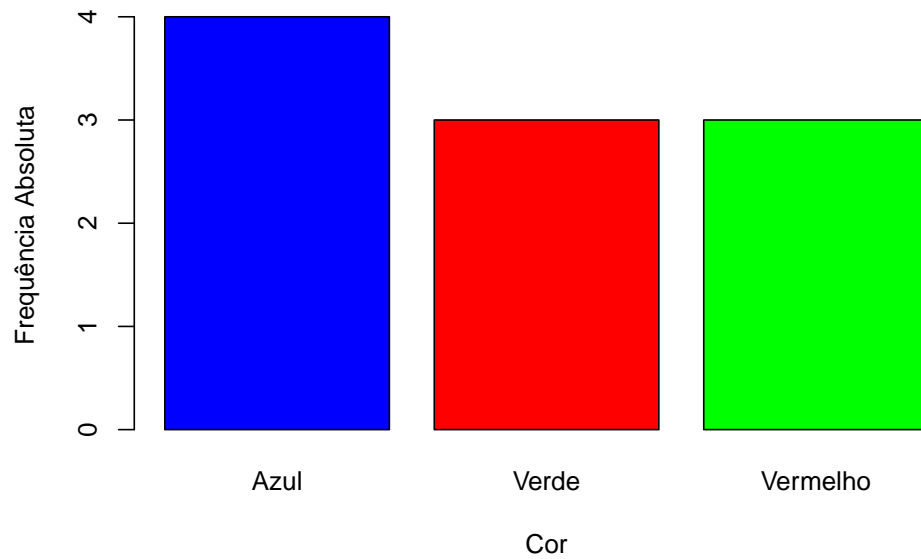
13.1 Gráfico de Barras

Gráfico de barras: Conjunto de barras verticais ou horizontais. Cada barra representa uma categoria, e a altura da barra mostra a frequência absoluta ou relativa dessa categoria. A largura das barras não tem significado.

```
# Dados de exemplo: cores favoritas
cores <- c("Azul", "Vermelho", "Verde", "Azul", "Verde",
"Vermelho", "Azul", "Verde", "Azul", "Vermelho")

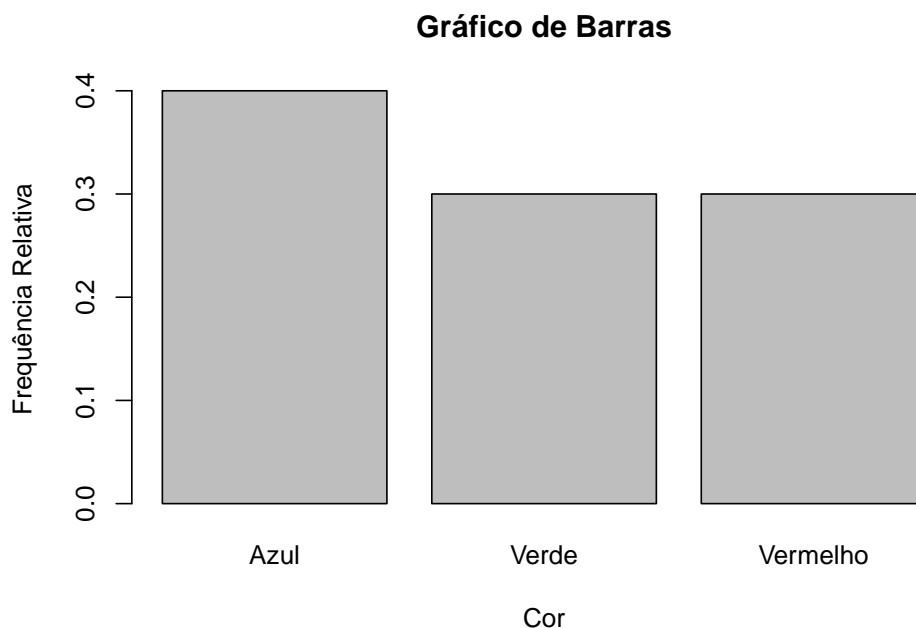
# Calcular as frequências absolutas
frequencia_absoluta <- table(cores)

# Criar o gráfico de barras com frequências absolutas
barplot(frequencia_absoluta,
  main = "Gráfico de Barras",
  xlab = "Cor",
  ylab = "Frequência Absoluta",
  col = c("blue", "red", "green"))
```

Gráfico de Barras

```
# Calcular as frequências relativas
frequencia_relativa <- frequencia_absoluta / length(cores)

# Criar o gráfico de barras com frequências relativas
barplot(frequencia_relativa,
  main = "Gráfico de Barras",
  xlab = "Cor",
  ylab = "Frequência Relativa")
```

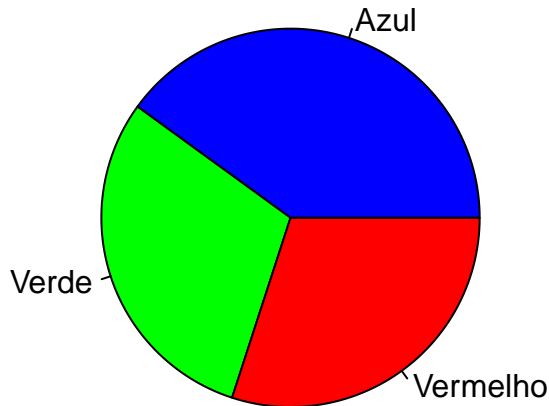


13.2 Gráfico circular (pizza)

Gráfico circular: Exibe as proporções ou porcentagens de diferentes categorias de dados em relação a um todo. Cada categoria é representada como uma “fatia” do círculo, e o tamanho de cada fatia é proporcional à sua contribuição para o total.

```
# Criar gráfico circular
pie(frequencia_relativa, main="Gráfico circular",
    col=c("blue", "green", "red"))
```

Gráfico circular



13.3 Histograma

Histograma é uma representação gráfica dos dados em que se marcam as classes (intervalos) no eixo horizontal e as frequências (absoluta ou relativa) no eixo vertical.

- Cada retângulo corresponde a uma classe.
- A largura de cada retângulo é igual à amplitude da classe
- Se as classes tiverem todas a mesma amplitude, a altura do retângulo é proporcional à frequência.

Por default, o R utiliza a frequência absoluta para construir o histograma. Se tiver interesse em representar as frequências relativas, utilize a opção `freq=FALSE` nos argumentos da função `hist()`. O padrão de intervalo de classe no R é $(a, b]$.

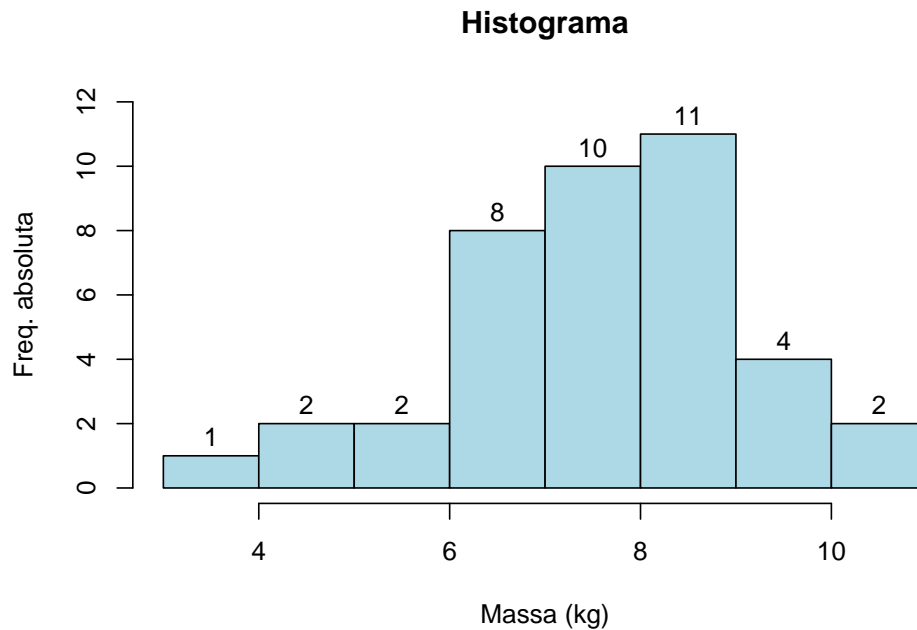
```
# Considere os dados referentes à massa (em kg) de 40 bicicletas
```

```
bicicletas <- c(4.3,6.8,9.2,7.2,8.7,8.6,6.6,5.2,8.1,10.9,7.4,4.5,3.8,7.6,6.8,7.8,8.4,7
```

```
h <- hist(bicicletas,
  main = "Histograma",
  xlab = "Massa (kg)",
  ylab = "Freq. absoluta",
```



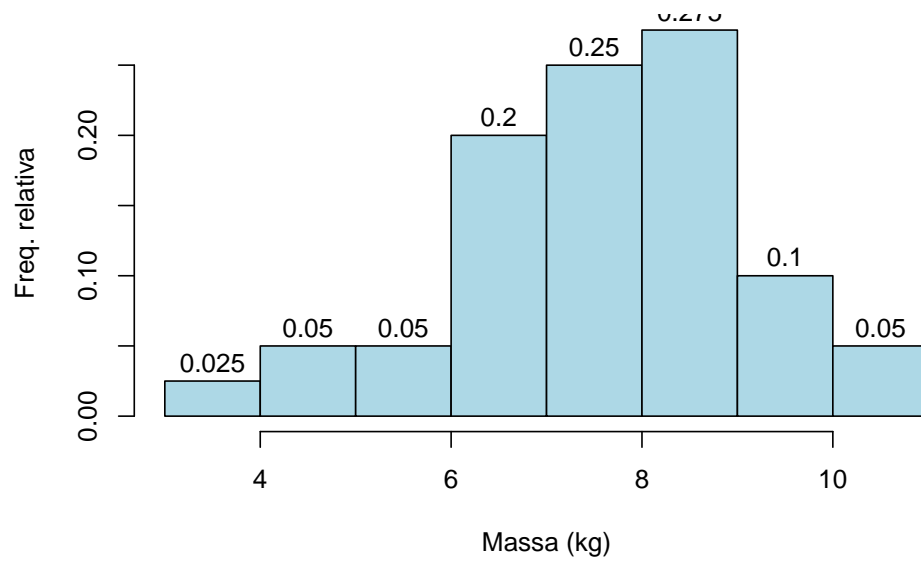
```
ylim = c(0,12),  
labels = TRUE,  
col = "lightblue")
```



```
# Pontos limites das classes  
h$breaks  
## [1] 3 4 5 6 7 8 9 10 11  
  
# O comando h$counts retorna um vetor com as frequências absolutas dentro de cada classe  
h$counts  
## [1] 1 2 2 8 10 11 4 2
```

```
# Histograma com frequência relativa  
hist(bicicletas,  
main = "Histograma",  
xlab = "Massa (kg)",  
ylab = "Freq. relativa",  
freq = FALSE,  
labels = TRUE,  
col = "lightblue")
```

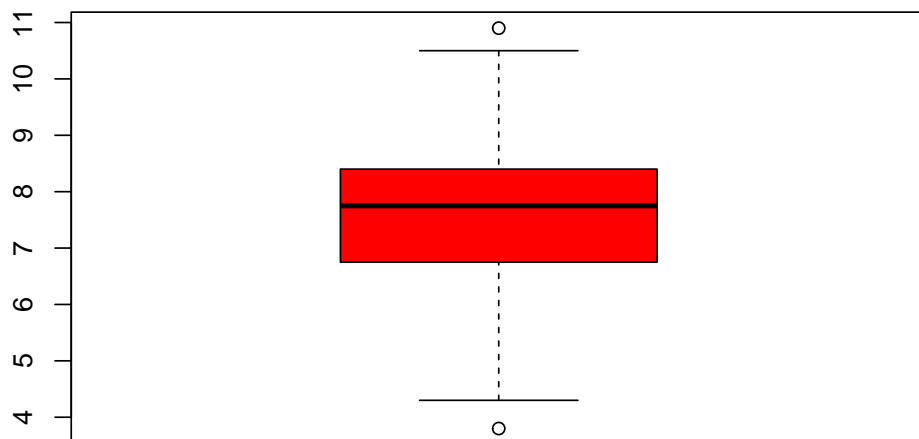
Histograma



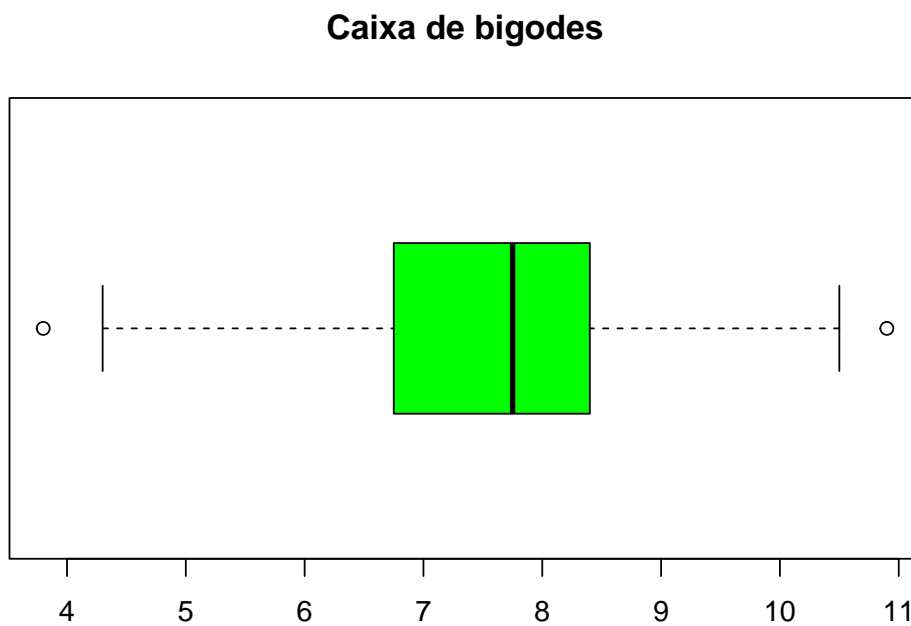
13.4 Box-plot

```
# Caixa de bigodes vertical
boxplot(bicicletas, main = "Caixa de bigodes", col="red")
```

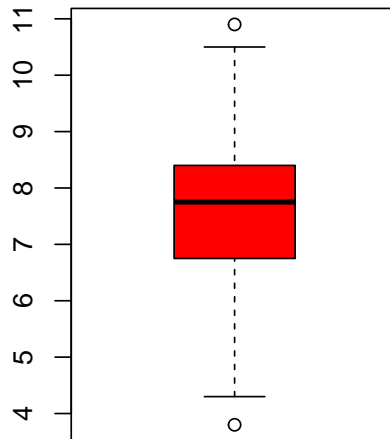
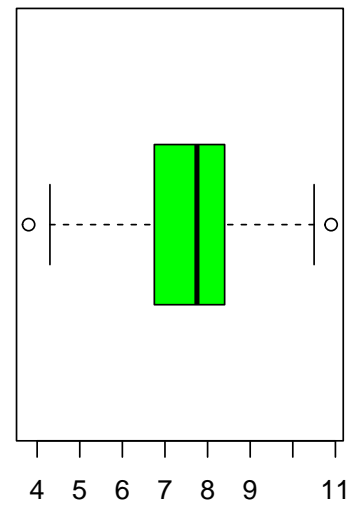
Caixa de bigodes



```
# Caixa de bigodes horizontal  
boxplot(bicicletas, main = "Caixa de bigodes", col="green", horizontal = TRUE)
```



```
# Caixa de bigodes lado a lado  
par(mfrow=c(1,2))  
  
# Caixa de bigodes vertical  
boxplot(bicicletas, main = "Caixa de bigodes", col = "red")  
  
# Caixa de bigodes horizontal  
boxplot(bicicletas, main = "Caixa de bigodes", col = "green", horizontal = TRUE)
```

Caixa de bigodes**Caixa de bigodes**

```
dev.off()
```

```
## null device  
##          1
```

Chapter 14

Manipulação de dados

14.1 Tibbles

Tibbles são uma versão aprimorada dos data frames no R, introduzida pelo pacote `tibble`, que faz parte do `tidyverse`, um conjunto de pacotes projetados para a ciência de dados. Um tibble fornece uma estrutura de dados mais moderna e amigável, melhorando a experiência de manipulação de dados no R.

Diferenças Principais entre Tibble e Data Frame

Característica	Data Frame	Tibble
Impressão no Console	Exibe todos os dados	Exibe um resumo com 10 linhas e colunas visíveis
Conversão de Strings	Converte strings para fatores por padrão	Mantém strings como caracteres
Manuseio de Colunas	Apenas vetores, listas podem ser problemáticas	Permite listas, funções, outros tibbles
Nomes de Colunas	Nomes devem ser únicos e sem espaços	Nomes podem ter espaços, ser duplicados, etc.
Retorno de Subsetting (<code>[]</code>)	Pode retornar um vetor ou data frame	Sempre retorna um tibble
Compatibilidade e Integração	Totalmente compatível com base R	Compatível, mas otimizado para uso com <code>tidyverse</code>
Manipulação de Dados	Pode ser menos intuitivo para algumas operações	Mais amigável e intuitivo, especialmente com pipes

```
library(tibble)

# Criando um tibble
tb <- tibble(
  x = 1:5,
  y = c("A", "B", "C", "D", "E"),
  z = x * 2
)

print(tb)
## # A tibble: 5 × 3
##       x y       z
##   <int> <chr> <dbl>
## 1     1 A         2
## 2     2 B         4
## 3     3 C         6
## 4     4 D         8
## 5     5 E        10
```

Conversão de data frame pra tibble:

```
# Criando um data frame
df <- data.frame(
  x = 1:5,
  y = c("A", "B", "C", "D", "E")
)

# Convertendo para tibble
tb <- as_tibble(df)

print(tb)
## # A tibble: 5 × 2
##       x y
##   <int> <chr>
## 1     1 A
## 2     2 B
## 3     3 C
## 4     4 D
## 5     5 E
```

14.2 O pacote dplyr

O `dplyr` é um dos pacotes mais populares e amplamente utilizados no R para manipulação e transformação de dados. Ele faz parte do conjunto de pacotes

“tidyverse,” que são projetados para simplificar o trabalho com dados no R. O `dplyr` oferece uma interface intuitiva e de fácil uso para realizar operações comuns em data frames, como seleção de colunas, filtragem de linhas, ordenação, resumo de dados e junção de data frames.

As principais funções do `dplyr` são:

- `select()` - seleciona colunas
- `arrange()` - ordena a base
- `filter()` - filtra linhas
- `mutate()` - cria/modifica colunas
- `group_by()` - agrupa a base
- `summarise()` - sumariza a base

Neste capítulo, vamos trabalhar com uma base de dados do Star Wars. Essa base está disponível dentro do pacote `dplyr` no R e pode ser acessada através do comando `starwars`.

Assim, utilizaremos o objeto `sw` para acessar os dados.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

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

## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union
```

```
sw <- starwars
sw
```

```
## # A tibble: 87 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Luke Sk~    172    77 blond      fair        blue         19  male  mascu~
## 2 C-3PO      167    75 <NA>      gold        yellow       112  none  mascu~
## 3 R2-D2       96    32 <NA>      white, bl~  red         33  none  mascu~
## 4 Darth V~   202   136 none      white       yellow      41.9  male  mascu~
## 5 Leia Or~   150    49 brown     light       brown        19  fema~  femin~
```

```
## 6 Owen La~      178    120 brown, gr~ light      blue      52    male mascu~
## 7 Beru Wh~      165     75 brown      light      blue      47    fema~ femin~
## 8 R5-D4         97     32 <NA>      white, red red      NA     none mascu~
## 9 Biggs D~     183     84 black      light      brown      24    male mascu~
## 10 Obi-Wan~    182     77 auburn, w~ fair      blue-gray  57    male mascu~
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

14.2.1 Selecionando colunas

Para selecionar colunas, utilizamos a função `select()`. Repare que não precisamos colocar o nome da coluna entre aspas.

```
select(sw, name)
```

```
## # A tibble: 87 x 1
##   name
##   <chr>
## 1 Luke Skywalker
## 2 C-3P0
## 3 R2-D2
## 4 Darth Vader
## 5 Leia Organa
## 6 Owen Lars
## 7 Beru Whitesun Lars
## 8 R5-D4
## 9 Biggs Darklighter
## 10 Obi-Wan Kenobi
## # i 77 more rows
```

Também podemos selecionar várias colunas

```
select(sw, name, mass, hair_color)
```

```
## # A tibble: 87 x 3
##   name      mass hair_color
##   <chr>    <dbl> <chr>
## 1 Luke Skywalker    77 blond
## 2 C-3P0             75 <NA>
## 3 R2-D2             32 <NA>
## 4 Darth Vader     136 none
## 5 Leia Organa      49 brown
```



```
## 6 Owen Lars          120 brown, grey
## 7 Beru Whitesun Lars  75 brown
## 8 R5-D4              32 <NA>
## 9 Biggs Darklighter  84 black
## 10 Obi-Wan Kenobi     77 auburn, white
## # i 77 more rows
```

Podemos usar o operador `:` para selecionar colunas consecutivas.

```
select(sw, name:hair_color)
```

```
## # A tibble: 87 x 4
##   name          height mass hair_color
##   <chr>         <int> <dbl> <chr>
## 1 Luke Skywalker   172    77 blond
## 2 C-3PO            167    75 <NA>
## 3 R2-D2            96    32 <NA>
## 4 Darth Vader      202   136 none
## 5 Leia Organa      150    49 brown
## 6 Owen Lars        178   120 brown, grey
## 7 Beru Whitesun Lars 165    75 brown
## 8 R5-D4            97    32 <NA>
## 9 Biggs Darklighter 183    84 black
## 10 Obi-Wan Kenobi   182    77 auburn, white
## # i 77 more rows
```

O `dplyr` possui um conjunto de funções auxiliares muito úteis para seleção de colunas. As principais são:

- `starts_with()`: para colunas que começam com um texto padrão
- `ends_with()`: para colunas que terminam com um texto padrão
- `contains()`: para colunas que contêm um texto padrão

```
select(sw, ends_with("color"))
```

```
## # A tibble: 87 x 3
##   hair_color skin_color eye_color
##   <chr>      <chr>      <chr>
## 1 blond     fair         blue
## 2 <NA>      gold         yellow
## 3 <NA>      white, blue red
## 4 none     white         yellow
## 5 brown    light         brown
```

```
## 6 brown, grey light blue
## 7 brown light blue
## 8 <NA> white, red red
## 9 black light brown
## 10 auburn, white fair blue-gray
## # i 77 more rows
```

Para remover colunas basta acrescentar um - antes da seleção.

```
select(sw, -name, -hair_color)
```

```
## # A tibble: 87 x 12
##   height mass skin_color eye_color birth_year sex gender homeworld species
##   <int> <dbl> <chr> <chr> <dbl> <chr> <chr> <chr> <chr>
## 1 172 77 fair blue 19 male mascu~ Tatooine Human
## 2 167 75 gold yellow 112 none mascu~ Tatooine Droid
## 3 96 32 white, blue red 33 none mascu~ Naboo Droid
## 4 202 136 white yellow 41.9 male mascu~ Tatooine Human
## 5 150 49 light brown 19 female femin~ Alderaan Human
## 6 178 120 light blue 52 male mascu~ Tatooine Human
## 7 165 75 light blue 47 female femin~ Tatooine Human
## 8 97 32 white, red red NA none mascu~ Tatooine Droid
## 9 183 84 light brown 24 male mascu~ Tatooine Human
## 10 182 77 fair blue-gray 57 male mascu~ Stewjon Human
## # i 77 more rows
## # i 3 more variables: films <list>, vehicles <list>, starships <list>
```

```
select(sw, -ends_with("color"))
```

```
## # A tibble: 87 x 11
##   name height mass birth_year sex gender homeworld species films vehicles
##   <chr> <int> <dbl> <dbl> <chr> <chr> <chr> <chr> <lis> <list>
## 1 Luke S~ 172 77 19 male mascu~ Tatooine Human <chr> <chr>
## 2 C-3P0 167 75 112 none mascu~ Tatooine Droid <chr> <chr>
## 3 R2-D2 96 32 33 none mascu~ Naboo Droid <chr> <chr>
## 4 Darth ~ 202 136 41.9 male mascu~ Tatooine Human <chr> <chr>
## 5 Leia O~ 150 49 19 fema~ femin~ Alderaan Human <chr> <chr>
## 6 Owen L~ 178 120 52 male mascu~ Tatooine Human <chr> <chr>
## 7 Beru W~ 165 75 47 fema~ femin~ Tatooine Human <chr> <chr>
## 8 R5-D4 97 32 NA none mascu~ Tatooine Droid <chr> <chr>
## 9 Biggs ~ 183 84 24 male mascu~ Tatooine Human <chr> <chr>
## 10 Obi-Wa~ 182 77 57 male mascu~ Stewjon Human <chr> <chr>
## # i 77 more rows
## # i 1 more variable: starships <list>
```

14.2.2 Exercícios

Utilize a base `sw` nos exercícios a seguir.

1. Teste aplicar a função `glimpse()` do pacote ‘dplyr’ à base `sw`. O que ela faz?
2. Crie uma tabela com apenas as colunas `name`, `gender`, e `films`. Salve em um objeto chamado `sw_simples`.
3. Selecione apenas as colunas `hair_color`, `skin_color` e `eye_color` usando a função auxiliar `contains()`.
4. Usando a função `select()` (e suas funções auxiliares), escreva códigos que retornem a base `sw` sem as colunas `hair_color`, `skin_color` e `eye_color`. Escreva todas as soluções diferentes que você conseguir pensar.

14.2.3 Ordenando a base

Para ordenar linhas, utilizamos a função `arrange()`. O primeiro argumento é a base de dados. Os demais argumentos são as colunas pelas quais queremos ordenar as linhas.

```
arrange(sw, mass)
```

```
## # A tibble: 87 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Ratts T~    79    15 none      grey, blue unknown      NA male mascul~
## 2 Yoda        66    17 white     green      brown      896 male mascul~
## 3 Wicket ~    88    20 brown     brown      brown       8 male mascul~
## 4 R2-D2       96    32 <NA>      white, bl~ red       33 none mascul~
## 5 R5-D4       97    32 <NA>      white, red red       NA none mascul~
## 6 Sebulba    112    40 none      grey, red  orange     NA male mascul~
## 7 Padmé A~   185    45 brown     light      brown      46 fema~ femin~
## 8 Dud Bolt    94    45 none      blue, grey yellow     NA male mascul~
## 9 Wat Tam~   193    48 none      green, gr~ unknown    NA male mascul~
## 10 Sly Moo~  178    48 none      pale       white      NA <NA> <NA>
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

Também podemos ordenar de forma decrescente usando a função `desc()`.

```
arrange(sw, desc(mass))
```

```
## # A tibble: 87 x 14
##   name      height mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Jabba D~    175  1358 <NA>      green-tan~ orange        600 herm~ mascu~
## 2 Grievous    216   159 none      brown, wh~ green, y~    NA  male mascu~
## 3 IG-88       200   140 none      metal      red          15  none mascu~
## 4 Darth V~    202   136 none      white      yellow       41.9 male mascu~
## 5 Tarfful     234   136 brown     brown      blue         NA  male mascu~
## 6 Owen La~    178   120 brown, gr~ light      blue         52  male mascu~
## 7 Bossk       190   113 none      green      red          53  male mascu~
## 8 Chewbac~    228   112 brown     unknown    blue         200 male mascu~
## 9 Jek Ton~    180   110 brown     fair       blue         NA  <NA> <NA>
## 10 Dexter ~   198   102 none      brown      yellow        NA  male mascu~
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

Ordenar segundo duas ou mais colunas.

```
arrange(sw, desc(height), desc(mass))
```

```
## # A tibble: 87 x 14
##   name      height mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Yarael ~    264    NA none      white      yellow        NA  male mascu~
## 2 Tarfful     234   136 brown     brown      blue         NA  male mascu~
## 3 Lama Su     229    88 none      grey       black         NA  male mascu~
## 4 Chewbac~    228   112 brown     unknown    blue         200 male mascu~
## 5 Roos Ta~    224    82 none      grey       orange        NA  male mascu~
## 6 Grievous    216   159 none      brown, wh~ green, y~    NA  male mascu~
## 7 Taun We     213    NA none      grey       black         NA  fema~ femin~
## 8 Tion Me~    206    80 none      grey       black         NA  male mascu~
## 9 Rugor N~    206    NA none      green      orange        NA  male mascu~
## 10 Darth V~    202   136 none      white      yellow       41.9 male mascu~
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

14.2.4 Exercícios

1. Ordene `mass` em ordem crescente e `birth_year` em ordem decrescente e salve em um objeto chamado `sw_ordenados`.

```
sw_ordenados <- arrange(sw, mass, desc(birth_year))
sw_ordenados
```

2. Selecione apenas as colunas `name` e `birth_year` e então ordene de forma decrescente pelo `birth_year`.

```
# Aninhando funções
arrange(select(sw, name, birth_year), desc(birth_year))

# Criando um objeto intermediário
sw_aux <- select(sw, name, birth_year)
arrange(sw_aux, desc(birth_year))

# Pipe
sw %>%
  select(name, birth_year) %>%
  arrange(desc(birth_year))
```

14.2.5 Filtrando linhas

Para filtrar valores de uma coluna da base, utilizamos a função `filter()`.

```
# filter(sw, height > 170)
# Ou
sw %>% filter(height > 170)
```

```
## # A tibble: 55 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Luke Sk~    172    77 blond      fair       blue        19   male mascul~
## 2 Darth V~    202   136 none       white      yellow      41.9 male mascul~
## 3 Owen La~    178   120 brown, gr~ light      blue        52   male mascul~
## 4 Biggs D~    183    84 black      light      brown       24   male mascul~
## 5 Obi-Wan~    182    77 auburn, w~ fair       blue-gray   57   male mascul~
## 6 Anakin ~    188    84 blond      fair       blue       41.9 male mascul~
## 7 Wilhuff~    180   NA auburn, g~ fair       blue       64   male mascul~
## 8 Chewbac~    228   112 brown      unknown    blue      200   male mascul~
## 9 Han Solo    180    80 brown      fair       brown       29   male mascul~
## 10 Greedo     173    74 <NA>      green      black       44   male mascul~
## # i 45 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

Podemos seleccionar apenas as colunas `name` e `height` para visualizarmos as alturas:

```
sw %>%
  filter(height > 170) %>%
  select(name, height)
```

```
## # A tibble: 55 x 2
##   name          height
##   <chr>         <int>
## 1 Luke Skywalker    172
## 2 Darth Vader       202
## 3 Owen Lars        178
## 4 Biggs Darklighter 183
## 5 Obi-Wan Kenobi    182
## 6 Anakin Skywalker  188
## 7 Wilhuff Tarkin    180
## 8 Chewbacca         228
## 9 Han Solo          180
## 10 Greedo           173
## # i 45 more rows
```

Podemos estender o filtro para duas ou mais colunas.

```
filter(sw, height>170, mass>80)
```

```
## # A tibble: 21 x 14
##   name      height mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>    <int> <dbl> <chr>    <chr>    <chr>    <dbl> <chr> <chr>
## 1 Darth V~    202   136 none     white     yellow     41.9 male masculi~
## 2 Owen La~    178   120 brown, gr~ light     blue       52  male masculi~
## 3 Biggs D~    183    84 black    light     brown      24  male masculi~
## 4 Anakin ~    188    84 blond   fair      blue      41.9 male masculi~
## 5 Chewbac~    228   112 brown   unknown   blue      200  male masculi~
## 6 Jabba D~    175  1358 <NA>     green-tan~ orange     600  herm~ masculi~
## 7 Jek Ton~    180   110 brown   fair      blue      NA   <NA> <NA>
## 8 IG-88      200   140 none     metal     red       15  none masculi~
## 9 Bossk      190   113 none     green     red       53  male masculi~
## 10 Ackbar    180    83 none     brown mot~ orange     41  male masculi~
## # i 11 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

```
sw %>% filter(height > 170, mass > 80)
```

```
## # A tibble: 21 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Darth V~    202   136 none      white      yellow      41.9 male  mascu~
## 2 Owen La~    178   120 brown, gr~ light      blue        52   male  mascu~
## 3 Biggs D~    183    84 black     light      brown       24   male  mascu~
## 4 Anakin ~    188    84 blond     fair       blue       41.9 male  mascu~
## 5 Chewbac~    228   112 brown     unknown    blue       200   male  mascu~
## 6 Jabba D~    175  1358 <NA>      green-tan~ orange     600   herm~ mascu~
## 7 Jek Ton~    180   110 brown     fair       blue       NA    <NA> <NA>
## 8 IG-88      200   140 none      metal      red        15   none  mascu~
## 9 Bossk      190   113 none      green      red        53   male  mascu~
## 10 Ackbar    180    83 none      brown mot~ orange     41   male  mascu~
## # i 11 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

Podemos filtrar colunas categóricas. O exemplo abaixo retorna uma tabela apenas com os personagens com cabelo preto ou castanho.

```
filter(sw, hair_color == "black" | hair_color == "brown")
```

```
## # A tibble: 31 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Leia Or~    150   49   brown     light      brown       19   fema~ femin~
## 2 Beru Wh~    165   75   brown     light      blue        47   fema~ femin~
## 3 Biggs D~    183   84   black     light      brown       24   male  mascu~
## 4 Chewbac~    228  112   brown     unknown    blue       200   male  mascu~
## 5 Han Solo    180   80   brown     fair       brown       29   male  mascu~
## 6 Wedge A~    170   77   brown     fair       hazel       21   male  mascu~
## 7 Jek Ton~    180  110   brown     fair       blue       NA    <NA> <NA>
## 8 Boba Fe~    183  78.2 black     fair       brown     31.5 male  mascu~
## 9 Lando C~    177   79   black     dark      brown       31   male  mascu~
## 10 Arvel C~    NA    NA   brown     fair       brown       NA    male  mascu~
## # i 21 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

```
sw %>% filter(hair_color %in% c("black", "brown"))
```



```
library(stringr)
sw %>% filter(str_detect(hair_color, "grey"))
```

```
## # A tibble: 3 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Owen Lars    178    120 brown, gr~ light      blue          52 male masculi~
## 2 Wilhuff T.   180     NA auburn, g~ fair       blue          64 male masculi~
## 3 Palpatine    170     75 grey      pale       yellow        82 male masculi~
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

14.2.6 Exercícios

Utilize a base `sw` nos exercícios a seguir.

1. Crie um objeto chamado `humanos` apenas com personagens que sejam humanos.
2. Crie um objeto chamado `altos_fortes` com personagens que tenham mais de 200 cm de altura e peso maior que 100 kg.
3. Retorne tabelas (tibbles) apenas com:
 - a. Personagens humanos que nasceram antes de 100 anos antes da batalha de Yavin (`birth_year < 100`).
 - b. Personagens com cor `light` ou `red`.
 - c. Personagens com massa maior que 100 kg, ordenados de forma decrescente por altura, mostrando apenas as colunas `name`, `mass` e `height`.
 - d. Personagens que sejam “Humano” ou “Droid”, e tenham uma altura maior que 170 cm.
 - e. Personagens que não possuem informação tanto de altura (`height`) quanto de massa (`mass`), ou seja, possuem NA em ambas as colunas.

14.2.7 Modificando e criando novas colunas

Para modificar uma coluna existente ou criar uma nova coluna, utilizamos a função `mutate()`. O código abaixo divide os valores da coluna `height` por 100, mudando a unidade de medida dessa variável de centímetros para metros.

```
sw %>% mutate(height = height/100)
```

```
## # A tibble: 87 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <dbl> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Luke Sk~    1.72    77 blond      fair        blue        19    male  mascu~
## 2 C-3PO      1.67    75 <NA>      gold        yellow      112   none  mascu~
## 3 R2-D2      0.96    32 <NA>      white, bl~  red        33    none  mascu~
## 4 Darth V~   2.02   136 none      white       yellow     41.9  male  mascu~
## 5 Leia Or~   1.5     49 brown     light       brown      19    fema~  femin~
## 6 Owen La~   1.78   120 brown, gr~ light       blue       52    male  mascu~
## 7 Beru Wh~   1.65    75 brown     light       blue       47    fema~  femin~
## 8 R5-D4      0.97    32 <NA>      white, red  red        NA     none  mascu~
## 9 Biggs D~   1.83    84 black     light       brown      24    male  mascu~
## 10 Obi-Wan~  1.82    77 auburn, w~ fair        blue-gray   57    male  mascu~
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

Também poderíamos ter criado essa variável em uma nova coluna. Repare que a nova coluna `height_meters` é colocada no final da tabela.

```
sw %>% mutate(height_meters = height/100)
```

```
## # A tibble: 87 x 15
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Luke Sk~    172    77 blond      fair        blue        19    male  mascu~
## 2 C-3PO      167    75 <NA>      gold        yellow      112   none  mascu~
## 3 R2-D2      96    32 <NA>      white, bl~  red        33    none  mascu~
## 4 Darth V~   202   136 none      white       yellow     41.9  male  mascu~
## 5 Leia Or~   150    49 brown     light       brown      19    fema~  femin~
## 6 Owen La~   178   120 brown, gr~ light       blue       52    male  mascu~
## 7 Beru Wh~   165    75 brown     light       blue       47    fema~  femin~
## 8 R5-D4      97    32 <NA>      white, red  red        NA     none  mascu~
## 9 Biggs D~   183    84 black     light       brown      24    male  mascu~
## 10 Obi-Wan~  182    77 auburn, w~ fair        blue-gray   57    male  mascu~
## # i 77 more rows
## # i 6 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>, height_meters <dbl>
```

Podemos fazer qualquer operação com uma ou mais colunas. Abaixo vamos criar um tibble que contenha as colunas `name`, `height`, `mass`, e uma nova coluna BMI, que calcule o Índice de Massa Corporal (IMC) de cada personagem, usando a fórmula $\text{mass} / (\text{height}/100)^2$. Caso `height` ou `mass` seja NA, a coluna BMI deve ser NA.

```
sw %>%
  mutate(BMI = ifelse(!is.na(height) & !is.na(mass), mass / (height/100)^2, NA)) %>%
  select(name, height, mass, BMI)
```

```
## # A tibble: 87 x 4
##   name          height mass  BMI
##   <chr>         <int> <dbl> <dbl>
## 1 Luke Skywalker    172    77  26.0
## 2 C-3P0             167    75  26.9
## 3 R2-D2              96    32  34.7
## 4 Darth Vader      202   136  33.3
## 5 Leia Organa       150    49  21.8
## 6 Owen Lars         178   120  37.9
## 7 Beru Whitesun Lars 165    75  27.5
## 8 R5-D4              97    32  34.0
## 9 Biggs Darklighter 183    84  25.1
## 10 Obi-Wan Kenobi   182    77  23.2
## # i 77 more rows
```

14.2.8 Exercícios

1. Crie uma coluna chamada `dif_peso_altura` (diferença entre altura e peso) e salve a nova tabela em um objeto chamado `starwars_dif`. Em seguida, filtre apenas os personagens que têm altura maior que o peso e ordene a tabela por ordem crescente de `dif_peso_altura`.

a. `indice_massa_altura = mass / height`

b. `indice_massa_medio = mean(mass, na.rm = TRUE)`

c. `indice_relativo = (indice_massa_altura - indice_massa_medio) / indice_massa_medio`

d. `acima_media = ifelse(indice_massa_altura > indice_massa_medio, "sim", "não")`

3. Crie uma nova coluna que classifique o personagem em “recente” (nascido após 100 anos antes da batalha de Yavin) e “antigo” (nascido há 100 anos ou mais).

14.2.9 Sumarizando a base

A função `summarize()` no R é utilizada para criar resumos estatísticos de dados dentro de um data frame ou tibble. Ela é frequentemente usada para calcular estatísticas agregadas, como médias, somas, contagens, entre outras.

O código abaixo resume a coluna `mass` pela sua média.

```
sw %>% summarize(media_massa = mean(mass, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   media_massa
##         <dbl>
## 1         97.3
```

Podemos calcular ao mesmo tempo sumarizações diferentes.

```
sw %>% summarize(
  media_massa = mean(mass, na.rm = TRUE),
  mediana_massa = median(mass, na.rm = TRUE),
  variancia_massa = var(mass, na.rm = TRUE)
)
```

```
## # A tibble: 1 x 3
##   media_massa mediana_massa variancia_massa
##         <dbl>         <dbl>         <dbl>
## 1         97.3           79         28716.
```

Podemos também sumarizar diversas colunas.

```
sw %>% summarize(
  media_massa = mean(mass, na.rm = TRUE),
  media_altura = mean(height, na.rm = TRUE),
  media_ano = mean(birth_year, na.rm = TRUE)
)
```

```
## # A tibble: 1 x 3
##   media_massa media_altura media_ano
##         <dbl>         <dbl>         <dbl>
## 1         97.3          175.         87.6
```

Para sumarizar uma coluna agrupada pelas categorias de uma segunda coluna usamos além do `summarize()` a função `group_by()`.

O código a abaixo calcula a altura média dos personagens para cada categoria da coluna `hair_color`.

```
sw %>%
  filter(!is.na(hair_color), !is.na(height)) %>%
  group_by(hair_color) %>%
  summarize(media_altura = mean(height, na.rm = TRUE))
```

```
## # A tibble: 11 x 2
##   hair_color    media_altura
##   <chr>         <dbl>
## 1 auburn         150
## 2 auburn, grey   180
## 3 auburn, white  182
## 4 black         174.
## 5 blond         177.
## 6 blonde        168
## 7 brown         177.
## 8 brown, grey   178
## 9 grey          170
## 10 none         181.
## 11 white        156
```

14.2.10 Exercícios

Utilize a base `sw` nos exercícios a seguir.

1. Calcule a altura média e mediana dos personagens.
2. Calcule a massa média dos personagens cuja altura é maior que 175 cm.
3. Apresente na mesma tabela a massa média dos personagens com altura menor que 175 cm e a massa média dos personagens com altura maior ou igual a 175 cm.
4. Retorne tabelas (`tibbles`) apenas com:
 - a. A altura média dos personagens por espécie.
 - b. A massa média e mediana dos personagens por espécie.
 - c. Apenas o nome dos personagens que participaram de mais de 2 filmes.

14.2.11 Juntando duas bases

Podemos juntar duas tabelas (data frame ou tibble) a partir de uma coluna utilizando as funções `left_join()`, `right_join()` ou `full_join()`.

- `left_join()`: Mantém todas as linhas da tabela à esquerda e adiciona colunas da tabela à direita onde existe uma correspondência.
- `right_join()`: Mantém todas as linhas da tabela à direita e adiciona colunas da tabela à esquerda onde existe uma correspondência.
- `full_join()`: Mantém todas as linhas de ambas as tabelas e adiciona NA onde não existe correspondência.

Para ilustrar o uso das funções de junção, vamos criar dois subconjuntos de dados da base `sw`:

1. Tabela de personagens altos (`personagens_altos`) - personagens com altura superior a 180 cm.
2. Tabela de personagens humanos (`personagens_humanos`).

```
# Criar subconjunto de personagens altos
personagens_altos <- sw %>%
  filter(height > 180) %>%
  select(name, height)

# Criar subconjunto de personagens humanos
personagens_humanos <- sw %>%
  filter(species == "Human") %>%
  select(name, species)
```

Agora, com essas duas tabelas, podemos usar as funções `left_join()`, `right_join()`, e `full_join()`.

A função `left_join()` junta duas tabelas, mantendo todas as linhas da tabela à esquerda (primeira tabela) e adicionando colunas da tabela à direita (segunda tabela) para as quais existe uma correspondência. Valores sem correspondência entre as bases receberão NA na nova base.

```
# Usar left_join para combinar os personagens altos com os humanos
humanos_altos_left_join <- left_join(personagens_altos, personagens_humanos, by = "name")

# Visualizar o resultado
print(humanos_altos_left_join)
```

```
## # A tibble: 39 x 3
##   name          height species
##   <chr>          <int> <chr>
## 1 Darth Vader      202 Human
## 2 Biggs Darklighter 183 Human
## 3 Obi-Wan Kenobi   182 Human
## 4 Anakin Skywalker 188 Human
## 5 Chewbacca        228 <NA>
## 6 Boba Fett        183 Human
## 7 IG-88            200 <NA>
## 8 Bossk            190 <NA>
## 9 Qui-Gon Jinn     193 Human
## 10 Nute Gunray     191 <NA>
## # i 29 more rows
```

Neste exemplo, a tabela resultante `humanos_altos_left_join` mantém todas as linhas de `personagens_altos` e adiciona informações da tabela

`personagens_humanos` quando há uma correspondência pelo nome do personagem.

A função `right_join()` faz o oposto de `left_join()`: mantém todas as linhas da tabela à direita e adiciona colunas da tabela à esquerda para as quais existe uma correspondência.

```
# Usar right_join para combinar humanos com personagens altos
humanos_altos_right_join <- right_join(personagens_altos, personagens_humanos, by = "name")

# Visualizar o resultado
print(humanos_altos_right_join)
```

```
## # A tibble: 35 x 3
##   name          height species
##   <chr>         <int> <chr>
## 1 Darth Vader      202 Human
## 2 Biggs Darklighter 183 Human
## 3 Obi-Wan Kenobi    182 Human
## 4 Anakin Skywalker  188 Human
## 5 Boba Fett         183 Human
## 6 Qui-Gon Jinn      193 Human
## 7 Padmé Amidala     185 Human
## 8 Ric Olié          183 Human
## 9 Quarsh Panaka     183 Human
## 10 Mace Windu       188 Human
## # i 25 more rows
```

Aqui, a tabela resultante `humanos_altos_right_join` mantém todas as linhas de `personagens_humanos` e adiciona informações de `personagens_altos` quando há uma correspondência pelo nome.

A função `full_join()` junta as duas tabelas, mantendo todas as linhas de ambas as tabelas e adicionando NA (valores ausentes) para correspondências inexistentes.

```
# Usar full_join para combinar todas as informações de personagens altos e humanos
humanos_altos_full_join <- full_join(personagens_altos, personagens_humanos, by = "name")

# Visualizar o resultado
print(humanos_altos_full_join)
```

```
## # A tibble: 59 x 3
##   name          height species
##   <chr>         <int> <chr>
```

```
## 1 Darth Vader          202 Human
## 2 Biggs Darklighter    183 Human
## 3 Obi-Wan Kenobi       182 Human
## 4 Anakin Skywalker     188 Human
## 5 Chewbacca            228 <NA>
## 6 Boba Fett            183 Human
## 7 IG-88                200 <NA>
## 8 Bossk                190 <NA>
## 9 Qui-Gon Jinn         193 Human
## 10 Nute Gunray         191 <NA>
## # i 49 more rows
```

A tabela resultante `humanos_altos_full_join` contém todas as linhas de ambas as tabelas `personagens_altos` e `personagens_humanos`. Se um personagem é encontrado em apenas uma das tabelas, as colunas da tabela ausente serão preenchidas com `NA`.

14.2.12 Exercícios

1. Crie uma tabela `personagens_altos` que contenha apenas personagens com altura superior a 180 cm e uma outra tabela `personagens_leves` que contenha apenas personagens com massa inferior a 75 kg. Use `left_join()` para combinar as duas tabelas com base no nome do personagem.

```
personagens_altos <- sw %>%
  filter(height > 180) %>%
  select(name, height)

personagens_leves <- sw %>%
  filter(mass < 75) %>%
  select(name, mass)

left_join(personagens_altos, personagens_leves, by = "name")
```

2. Crie uma tabela `personagens_humanos` que contenha apenas personagens humanos e uma outra tabela `cor_cabelo` que contenha apenas personagens com cor de cabelo diferente de `NA`. Use `right_join()` para combinar `personagens_humanos` e `cor_cabelo` com base no nome do personagem.

```
personagens_humanos <- sw %>%
  filter(species == "Human") %>%
  select(name, species)

cor_cabelo <- sw %>%
```



```
filter(!is.na(hair_color)) %>%  
select(name, hair_color,)  
  
right_join(personagens_humanos, cor_cabelo, by = "name")
```

3. Crie uma tabela `especies_personagens` que contenha apenas personagens de espécies conhecidas (não NA) e uma outra tabela `cor_olhos` que contenha apenas personagens com cor de olho conhecida (não NA). Use `full_join()` para combinar as duas tabelas com base no nome do personagem.

```
especies_personagens <- sw %>%  
  filter(!is.na(species)) %>%  
  select(name, species)  
  
cor_olhos <- sw %>%  
  filter(!is.na(eye_color)) %>%  
  select(name, eye_color)  
  
full_join(especies_personagens, cor_olhos, by = "name")
```


Chapter 15

Visualização

Chapter 16

O pacote ggplot2

O `ggplot2` é um dos pacotes mais populares do R para criar gráficos. Ele implementa o conceito de *Grammar of Graphics*, que oferece uma maneira sistemática de descrever e construir gráficos. Este conceito está apresentado no livro *The Grammar of graphics*.

Para este capítulo vamos seguir o material disponível em [link](#). O ficheiro com a base de filmes IMDB está disponível no [fénix](#)

Chapter 17

Simulação

A simulação é uma poderosa ferramenta que aproveita a capacidade dos computadores modernos para realizar cálculos que, de outra forma, seriam difíceis ou até impossíveis de serem feitos analiticamente. A Lei dos Grandes Números nos assegura que, ao observarmos uma grande amostra de variáveis aleatórias independentes e identicamente distribuídas (i.i.d.) com média finita, a média dessas observações tende a se aproximar da média real da distribuição. Em vez de nos esforçarmos para encontrar essa média através de métodos analíticos complexos, podemos utilizar o poder de processamento de um computador para gerar uma amostra suficientemente grande dessas variáveis aleatórias. A partir dessa amostra, calculamos a média observada, que serve como uma estimativa confiável da média verdadeira. No entanto, a eficácia desse método depende de três fatores cruciais: identificar corretamente os tipos de variáveis aleatórias necessárias para o problema em questão, garantir que o computador seja capaz de gerar essas variáveis de forma precisa, e determinar o tamanho adequado da amostra para que possamos confiar nos resultados obtidos. A simulação, portanto, não só simplifica o processo de resolução de problemas complexos, como também oferece uma abordagem prática e eficiente para explorar cenários onde o cálculo analítico tradicional é impraticável.

Iniciamos com alguns exemplos básicos de simulação para resolver questões cujas respostas já conhecemos de forma analítica, apenas para demonstrar que a simulação cumpre o que promete. Além disso, esses exemplos introdutórios ajudarão a destacar alguns pontos importantes que precisam ser considerados ao tentar resolver problemas mais complexos utilizando simulação.

Exemplo 1 (A média de uma distribuição): A média da distribuição uniforme no intervalo $[0,1]$ é conhecida por ser $1/2$. Se tivéssemos disponível um grande número de variáveis aleatórias i.i.d. uniformes no intervalo $[0,1]$, digamos, X_1, \dots, X_n , a Lei dos Grandes Números nos diz que $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$ deve estar próximo da média $1/2$. A Tabela abaixo fornece as médias de várias

amostras simuladas diferentes de tamanho n a partir da distribuição uniforme em $[0,1]$ para vários valores diferentes de n . Não é difícil ver que as médias estão próximas de 0.5 na maioria dos casos, mas há bastante variação, especialmente para $n = 100$. Parece haver menos variação para $n = 1000$, e ainda menos para os dois maiores valores de n .

n	Replicações da Simulação				
100	0.485	0.481	0.484	0.569	0.441
1.000	0.497	0.506	0.480	0.498	0.499
10.000	0.502	0.501	0.499	0.498	0.498
100.000	0.502	0.499	0.500	0.498	0.499

A maneira que obtemos a amostra aleatória uniforme foi usando a função `runif`

```
runif(100, 0, 1)
```

Como mencionamos anteriormente, não há necessidade de simulação no exemplo acima. Esta foi apenas para ilustrar que a simulação pode fazer o que afirma. Porém, é preciso estar ciente de que, por maior que seja a amostra simulada, a média de uma amostra de variáveis aleatórias i.i.d. não será necessariamente igual à sua média. É preciso ser capaz de levar em conta a variabilidade.

Exemplo onde a simulação pode ajudar

A seguir, apresentamos um exemplo em que as questões básicas são relativamente simples de descrever, mas a solução analítica seria, na melhor das hipóteses, entediante.

Esperando por uma pausa. Dois atendentes, A e B, em um restaurante fast-food começam a servir clientes ao mesmo tempo. Eles concordam em se encontrar para um intervalo depois que cada um deles atender 10 clientes. Presumivelmente, um deles terminará antes do outro e terá que esperar. Quanto tempo, em média, um dos atendentes terá que esperar pelo outro?

Suponha que modelemos todos os tempos de serviço, independentemente do atendente, como variáveis aleatórias i.i.d. tendo distribuição exponencial com parâmetro 0.3 clientes por minuto. Então, o tempo que um atendente leva para atender 10 clientes tem a distribuição gama com parâmetros 10 e 0.3. Seja X o tempo que A leva para atender 10 clientes e seja Y o tempo que B leva para atender 10 clientes. Somos solicitados a calcular a média de $|X - Y|$. A maneira mais direta de encontrar esta média analiticamente exigiria um integral bidimensional sobre a união de duas regiões não retangulares.

Por outro lado, suponha que um computador possa nos fornecer tantas variáveis aleatórias gama independentes quantas desejarmos. Podemos então obter um par (X, Y) e calcular $Z = |X - Y|$. Em seguida, repetimos esse processo

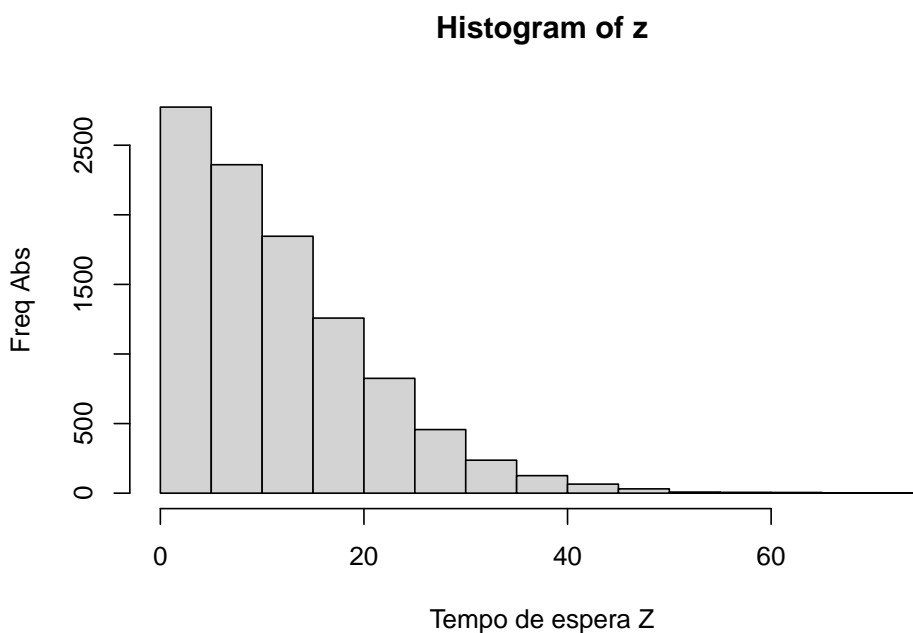
independentemente quantas vezes quisermos e calculamos a média de todos os valores Z observados. A média deve ficar próxima da média de Z .

```
set.seed(123)
x <- rgamma(10000, 10, 0.3)
y <- rgamma(10000, 10, 0.3)
z <- abs(x-y)
mean(z)
```

```
## [1] 11.75882
```

Temos então um tempo médio de 11.75 minutos.

```
hist(z, xlab = "Tempo de espera Z", ylab = "Freq Abs")
```



17.1 Geração de números pseudoaleatórios

Números Aleatórios

Números aleatórios são valores que são gerados de forma imprevisível e não seguem nenhum padrão determinado. Em outras palavras, cada número em uma sequência de números aleatórios é escolhido de maneira independente dos outros, sem qualquer correlação entre eles. Na prática, os números aleatórios

são usados em diversas áreas, como criptografia, simulações, estatísticas, jogos de azar, entre outros, onde é crucial que os números não possam ser antecipados.

A verdadeira aleatoriedade é geralmente derivada de processos físicos que são inerentemente imprevisíveis, como a radiação cósmica, ruído térmico em circuitos eletrônicos, ou o decaimento radioativo. Em computação, no entanto, obter números verdadeiramente aleatórios é difícil e muitas vezes desnecessário.

Números Pseudoaleatórios

Números pseudoaleatórios, por outro lado, são números que são gerados por algoritmos que produzem sequências que parecem aleatórias, mas são, na verdade, determinadas por um valor inicial chamado semente (ou “seed” em inglês). Se o algoritmo é iniciado com a mesma semente, ele produzirá exatamente a mesma sequência de números.

Embora sejam determinísticos, os números pseudoaleatórios são amplamente utilizados porque podem ser gerados rapidamente e, para muitas aplicações, eles são suficientemente aleatórios. A principal vantagem é que, ao usar a mesma semente, é possível replicar experimentos ou simulações, o que é útil em pesquisas e depurações.

Uma das aproximações mais comuns para gerar números pseudoaleatórios é o método *congruencial multiplicativo*:

- Considere um valor inicial x_0 , chamado semente;
- Recursivamente calcule os valores sucessivos x_n , $n \geq 1$, usando:

$$x_n = ax_{n-1} \bmod m,$$

onde a e m são inteiros positivos dados. Ou seja, x_n é o resto da divisão inteira de ax_{n-1} por m ;

- A quantidade x_n/m é chamada um número pseudoaleatório, ou seja, é uma aproximação para o valor de uma variável aleatória uniforme.

As constantes a e m a serem escolhidas devem satisfazer três critérios:

- Para qualquer semente inicial, a sequência resultante deve ter a “aparência” de uma sequência de variáveis aleatórias uniformes $(0,1)$ independentes.
- Para qualquer semente inicial, o número de variáveis que podem ser geradas antes da repetição ocorrer deve ser grande.
- Os valores podem ser calculados eficientemente em um computador.

17.2 A função `sample()`

A função `sample()` em R é utilizada para gerar uma amostra aleatória a partir de um conjunto de dados ou uma sequência de números. Ela é extremamente flexível, permitindo que você defina o tamanho da amostra, se a amostragem é feita com ou sem reposição, e também se os elementos têm probabilidades diferentes de serem selecionados.

Sintaxe

```
sample(x, size, replace = FALSE, prob = NULL)
```

- **x**: Vetor de elementos a serem amostrados.
- **size**: Tamanho da amostra.
- **replace**: Indica se a amostragem é com reposição (`TRUE`) ou sem reposição (`FALSE`).
- **prob**: Um vetor de probabilidades associadas a cada elemento em

x.

Exemplo 1: Amostragem Simples sem Reposição.

```
# Suponha que temos uma população de 1 a 10
pop <- 1:10

# Queremos uma amostra de 5 elementos
amostra <- sample(pop, size = 5, replace = FALSE)
print(amostra)
## [1] 5 2 4 10 8
```

Exemplo 2: Amostragem com Reposição.

```
# Amostra com reposição
amostra_repos <- sample(pop, size = 5, replace = TRUE)
print(amostra_repos)
## [1] 6 8 9 9 6
```

Exemplo 3: Amostragem com Probabilidades Diferentes.

```
# Probabilidades associadas a cada elemento
prob <- c(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.05, 0.05)

# Amostra com probabilidades diferentes
amostra_prob <- sample(pop, size = 5, prob = prob)
print(amostra_prob)
## [1] 2 4 6 5 3
```

17.3 Exercícios

1. Crie um vetor com os números de 1 a 20. Utilize a função `sample()` para selecionar uma amostra aleatória de 5 elementos desse vetor. A amostragem deve ser feita sem reposição.
2. Suponha que você tem uma população representada pelos números de 1 a 10. Utilize a função `sample()` para selecionar uma amostra de 10 elementos com reposição.
3. Crie um vetor com as letras A, B, C, D, E. Aplique a função `sample()` para selecionar uma amostra de 3 letras, onde a probabilidade de cada letra ser selecionada é dada pelo vetor `c(0.1, 0.2, 0.3, 0.25, 0.15)`.
4. Crie um vetor com os números de 1 a 10. Utilize a função `sample()` para reordenar aleatoriamente os elementos desse vetor.
5. Crie um vetor com os nomes de cinco frutas: “Maçã”, “Banana”, “Laranja”, “Uva”, “Pera”. Utilizando a função `sample()`, selecione aleatoriamente uma fruta desse vetor. Em seguida, selecione uma amostra de 3 frutas.
6. Você é responsável por realizar um teste de qualidade em uma fábrica. Há 1000 produtos fabricados, numerados de 1 a 1000. Selecione uma amostra aleatória de 50 produtos para inspeção, garantindo que não haja reposição na seleção.
7. Simule o lançamento de dois dados justos 10000 vezes e registre as somas das faces resultantes. Utilize a função `sample()` para realizar a simulação. Em seguida, crie um histograma das somas obtidas.
8. Você possui um vetor de 200 estudantes classificados em três turmas: A, B, e C. As turmas têm tamanhos diferentes (50, 100, e 50 alunos, respectivamente). Usando `sample()`, selecione uma amostra de 20 alunos, mantendo a proporção original das turmas.
9. Um cartão de Bingo contém 24 números aleatórios entre 1 e 75 (excluindo o número central “free”). Crie 5 cartões de Bingo únicos usando a função `sample()`.
10. Em um estudo clínico, 30 pacientes devem ser randomizados em dois grupos: tratamento e controle. O grupo de tratamento deve conter 20 pacientes e o grupo

de controle 10. Usando `sample()`, faça a randomização dos pacientes. Dica: use a função `setdiff()`.

Chapter 18

Método da transformada inversa

18.1 Variável aleatória discreta

Suponha que queremos gerar o valor de uma variável aleatória discreta X com função massa de probabilidade $P(X = x_i) = p_i$, $i = 0, 1, \dots$, $\sum_i p_i = 1$. Para isso, basta gerar um número aleatório $U \sim U(0, 1)$ e considerar:

$$X = \begin{cases} x_0, & \text{se } U < p_0 \\ x_1, & \text{se } p_0 \leq U < p_0 + p_1 \\ \vdots & \\ x_i, & \text{se } \sum_{j=0}^{i-1} p_j \leq U < \sum_{j=0}^i p_j \\ \vdots & \end{cases}$$

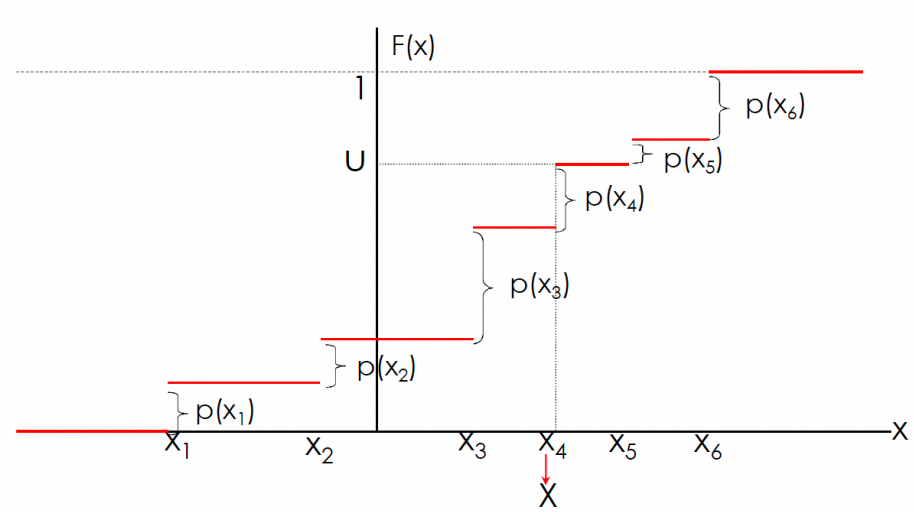
Como, para $0 < a < b < 1$, $P(a \leq U < b) = b - a$, temos que

$$P(X = x_i) = P\left(\sum_{j=0}^{i-1} p_j \leq U < \sum_{j=0}^i p_j\right) = p_i.$$

Se os x_i , $i \geq 0$, estão ordenados $x_0 < x_1 < \dots$ e se denotarmos por F a função de distribuição de X , então $F(x_k) = \sum_{i=0}^k p_i$ e assim

$$X = x_i \quad \text{se} \quad F(x_{i-1}) \leq U < F(x_i)$$

Em outras palavras, depois de gerar um número aleatório U nós determinamos o valor de X encontrando o intervalo $[F(x_{i-1}), F(x_i)]$ no qual U pertence (ou, equivalentemente, encontrando a inversa de $F(U)$).



Exemplo 1: Seja X uma variável aleatória discreta tal que $p_1 = 0.20$, $p_2 = 0.15$, $p_3 = 0.25$, $p_4 = 0.40$ onde $p_j = P(X = j)$. Gere 1000 valores dessa variável aleatória.

Para a variável aleatória X , a função de distribuição acumulada é dada pela soma cumulativa das probabilidades:

$$F(x) = \begin{cases} 0, & \text{se } x < 1 \\ p_1, & \text{se } 1 \leq x < 2 \\ p_1 + p_2, & \text{se } 2 \leq x < 3 \\ p_1 + p_2 + p_3, & \text{se } 3 \leq x < 4 \\ 1, & \text{se } x \geq 4 \end{cases}$$

Com os valores fornecidos:

$$F(x) = \begin{cases} 0, & \text{se } x < 1 \\ 0.20, & \text{se } 1 \leq x < 2 \\ 0.35, & \text{se } 2 \leq x < 3 \\ 0.60, & \text{se } 3 \leq x < 4 \\ 1, & \text{se } x \geq 4 \end{cases}$$

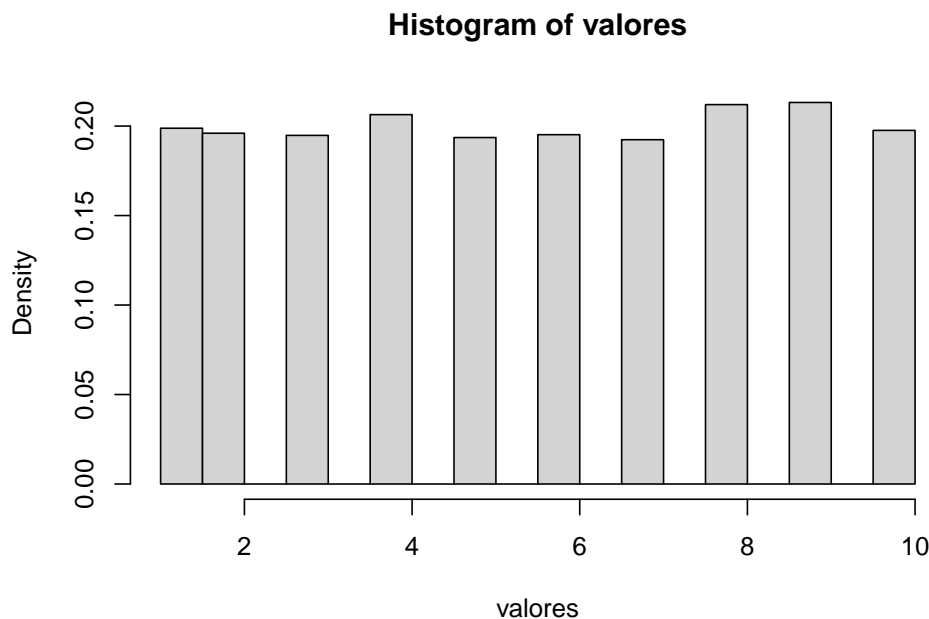
Gerar um número aleatório uniforme U no intervalo $[0,1]$. Para determinar o valor de X correspondente a U :

- Se $U < 0.20$, então $X = 1$
- Se $0.20 \leq U < 0.35$, então $X = 2$
- Se $0.35 \leq U < 0.60$, então $X = 3$

- Se $0.60 \leq U \leq 1$, então $X = 4$

Exemplo 2: Seja X uma variável aleatória discreta assumindo os valores: $1, 2, \dots, 10$ com probabilidade $1/10$ para $x = 1, 2, \dots, 10$. Gerar 5000 valores dessa variável aleatória. Representar graficamente e determinar: média, desvio padrão e mediana.

```
gerar_va_inversa <- function(){  
  # Gerar número aleatório entre 0 e 1  
  u <- runif(1,0,1)  
  p <- 1/10 # primeira probabilidade P(X=1)  
  F <- p # inicializar a função de distribuição acumulada  
  X <- 1 # inicializar o valor da va X  
  
  while(u > F){  
    X <- X+1  
    F <- F+p  
  }  
  return(X)  
}  
valores <- replicate(5000,gerar_va_inversa())  
hist(valores, freq = FALSE)
```



```
mean(valores)
```

```
## [1] 5.5388
```

```
sd(valores)
```

```
## [1] 2.878985
```

```
median(valores)
```

```
## [1] 6
```

Exemplo 3: Geração de uma variável aleatória com distribuição de Bernoulli. A variável aleatória X é de Bernoulli com parâmetro p se

$$P(X = x) = \begin{cases} 1 - p, & \text{se } x = 0 \\ p, & \text{se } x = 1 \end{cases}$$

Para gerar uma Bernoulli(p) podemos usar o seguinte algoritmo que é equivalente ao método da transformada inversa

1. Gerar um número aleatório U ;
2. Se $U \leq p$ então $X = 1$ senão $X = 0$.

```
# Gerando uma variável aleatória com distribuição de Bernoulli(p)
gerar_bernoulli_inversa <- function(p){
  U <- runif(1)
  if (U <= p){
    X <- 1
  } else {
    X <- 0
  }
  return(X)
}

valores <- replicate(100,gerar_bernoulli_inversa(0.8))
sum(valores)/100
```

```
## [1] 0.73
```

Exemplo 4: Gerar uma variável aleatória com distribuição Binomial(n, p). Aqui podemos usar o facto de que se X_1, X_2, \dots, X_n são Bernoullis i.i.d., então

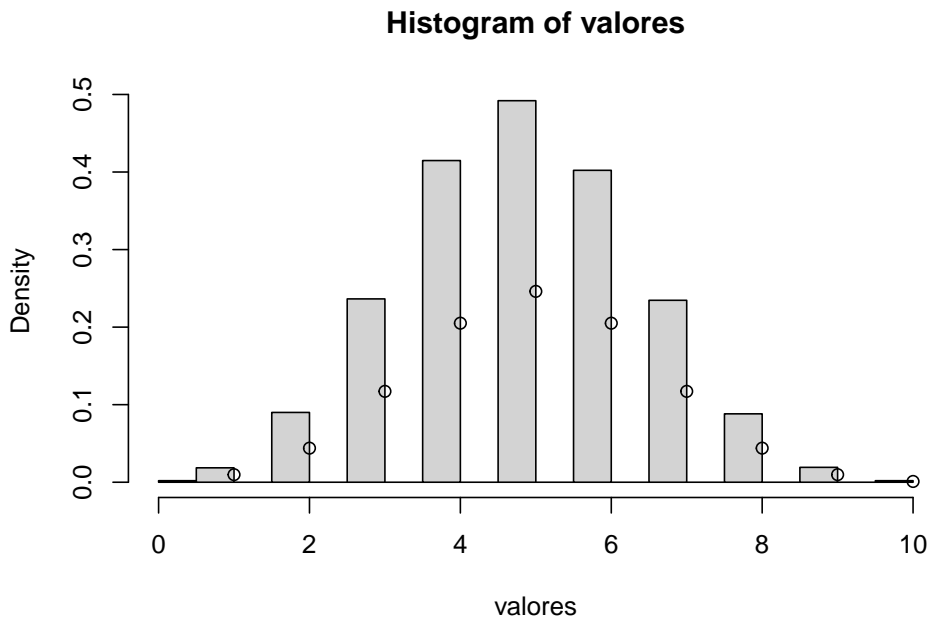
$$X = X_1 + X_2 + \dots + X_n$$

é uma Binomial(n, p).

```
# Gerando uma variável aleatória com distribuição Binomial(n,p)

gerar_binomial_inversa <- function(n,p){
  X <- sum(replicate(n,gerar_bernoulli_inversa(p)))
  return(X)
}

valores <- replicate(10000,gerar_binomial_inversa(10,0.5))
hist(valores, freq = FALSE)
points(1:10, dbinom(1:10,10,0.5))
```



Exemplo 5: Geração de uma variável aleatória com distribuição Geométrica(p). Seja $X \sim Geometrica(p)$. Lembre que

$$P(X = x) = p(1 - p)^{x-1}$$

e que

$$F(x) = P(X \leq x) = \begin{cases} 0, & \text{se } x < 1 \\ 1 - (1 - p)^x, & \text{se } x \geq 1 \end{cases}$$

O seguinte algoritmo é equivalente ao método da transformada inversa:

1. Gerar um número aleatório U ;
2. Fazer $X = \lfloor \ln(U)/\ln(1-p) \rfloor$

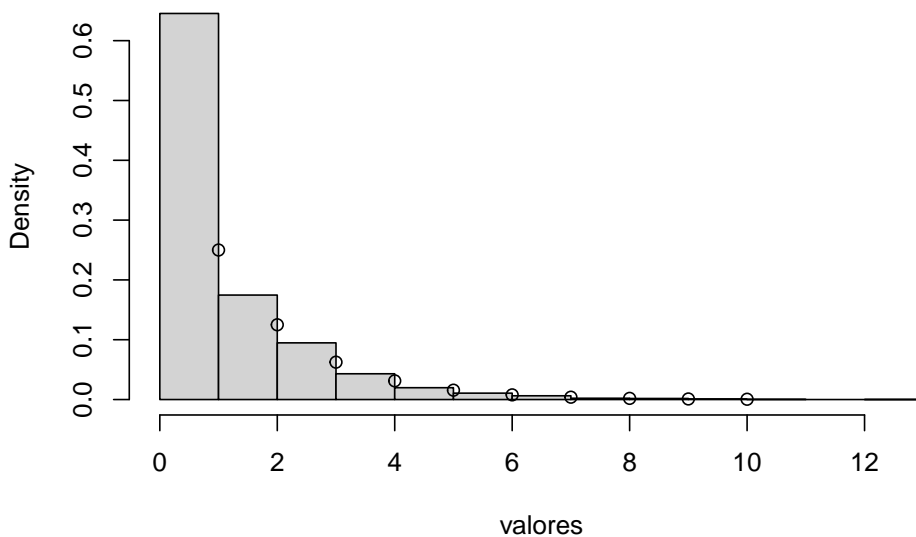
onde $\lfloor \cdot \rfloor$ = maior inteiro.

```
# Gerar uma variável aleatória com distribuição Geométrica(p)

gerar_geometrica_inversa <- function(p){
  U <- runif(1)
  X <- round(log(U)/log(1-p))
  return(X)
}

valores <- replicate(10000, gerar_geometrica_inversa(0.5))
hist(valores, freq = FALSE)
points(1:10, dgeom(1:10,0.5))
```

Histogram of valores



Exemplo 6: Geração de uma variável aleatória com distribuição de Poisson. A variável aleatória X é de Poisson com média λ se

$$p_i = P(X = i) = \frac{e^{-\lambda} \lambda^i}{i!}, \quad i = 0, 1, \dots$$

A chave para usar o método da transformada inversa para gerar uma tal variável aleatória é dada pela seguinte identidade:

$$p_{i+1} = \frac{\lambda}{i+1} p_i, \quad i \geq 0.$$

Ao utilizar a recursão acima para calcular as probabilidades de Poisson quando elas são necessárias, o algoritmo da transformada inversa para gerar uma variável aleatória de Poisson com média λ pode ser expresso como segue.

```
# Gerando uma va com distribuição de Poisson

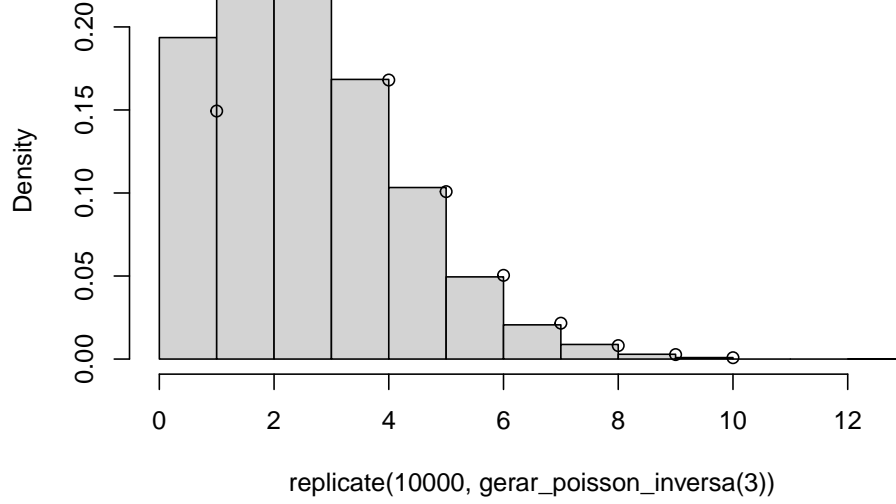
lambda <- 3 # exemplo com lambda = 3

# Função para gerar uma variável aleatória de Poisson usando o método da transformada inversa
gerar_poisson_inversa <- function(lambda) {
  U <- runif(1) # Gerar um número aleatório uniforme entre 0 e 1
  p <- exp(-lambda) # Inicializar a primeira probabilidade P(X=0)
  F <- p # Inicializar a função de distribuição acumulada (CDF)
  X <- 0 # Inicializar o valor da variável aleatória

  # Acumular probabilidades até que a CDF exceda U
  while (U > F) {
    X <- X + 1
    p <- p * lambda / X # Atualizar a probabilidade P(X=k)
    F <- F + p # Atualizar a CDF
  }

  return(X)
}

hist(replicate(10000,gerar_poisson_inversa(3)),freq = FALSE)
points(1:10,dpois(1:10,3))
```

Histogram of replicate(10000, gerar_poisson_inversa(3))

18.2 Variável aleatória contínua

Uma variável aleatória X tem densidade $f(x) = 2x$, para $0 < x < 1$, e 0, caso contrário. Suponha que queremos simular observações de X . Nesta secção, apresentaremos um método simples e flexível para simulação de uma distribuição contínua.

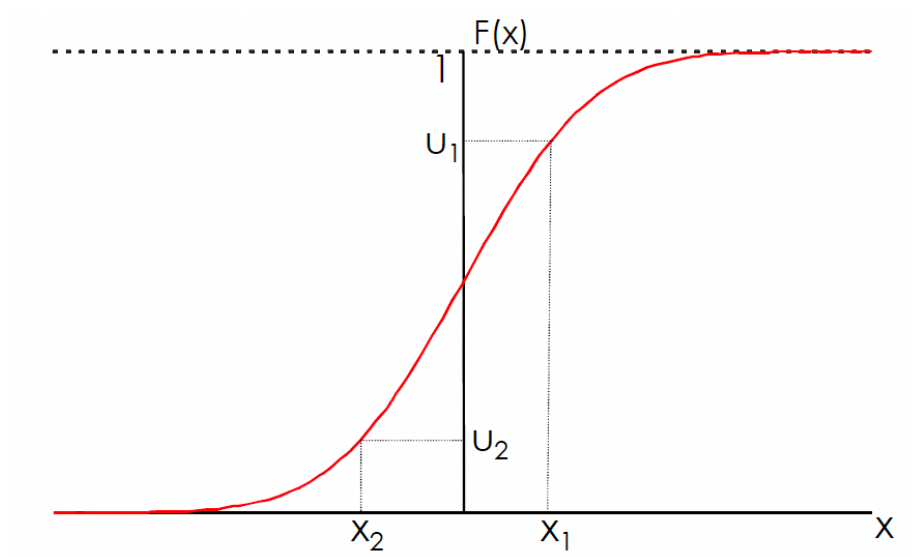
Proposição: Suponha que X é uma variável aleatória com função de distribuição F , onde F é invertível com função inversa F^{-1} . Seja U uma variável aleatória uniforme $(0, 1)$. Então a distribuição de $F^{-1}(U)$ é igual a distribuição de X , ou seja, a variável aleatória X definida por $X = F^{-1}(U)$ tem distribuição F .

A prova desta proposição é fácil e rápida. Precisamos mostrar que $F^{-1}(U)$ tem a mesma distribuição que X . Assim,

$$\begin{aligned}
 P(X \leq x) &= P(F^{-1}(U) \leq x) = P(FF^{-1}(U) \leq F(x)) \\
 &= P(U \leq F(x)) = P(0 \leq U \leq F(x)) \\
 &= F(x) - 0 \\
 &= F(x).
 \end{aligned}$$

A última igualdade segue do facto de que $U \sim U(0, 1)$ e $0 \leq F(x) \leq 1$.

Essa proposição mostra que pode-se gerar uma variável aleatória X de uma função de distribuição contínua F gerando um número aleatório U e tomando $X = F^{-1}(U)$.

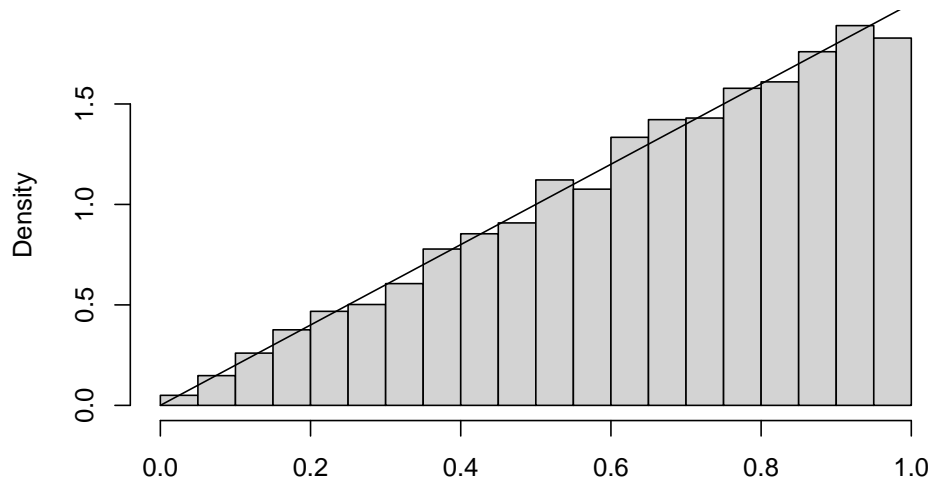


Exemplo 1: Considere nossa variável aleatória X com densidade $f(x) = 2x$. A função de distribuição de X é

$$F(x) = P(X \leq x) = \int_0^x 2t \, dt = x^2, \quad \text{para } 0 < x < 1.$$

A função $F(x) = x^2$ é invertível no intervalo $(0, 1)$ e $F^{-1}(x) = \sqrt{x}$. O método da transformada inversa diz que se $U \sim U(0, 1)$, então $F^{-1}(U) = \sqrt{U}$ tem a mesma distribuição que X . Portanto para simular X , basta gerar \sqrt{U} .

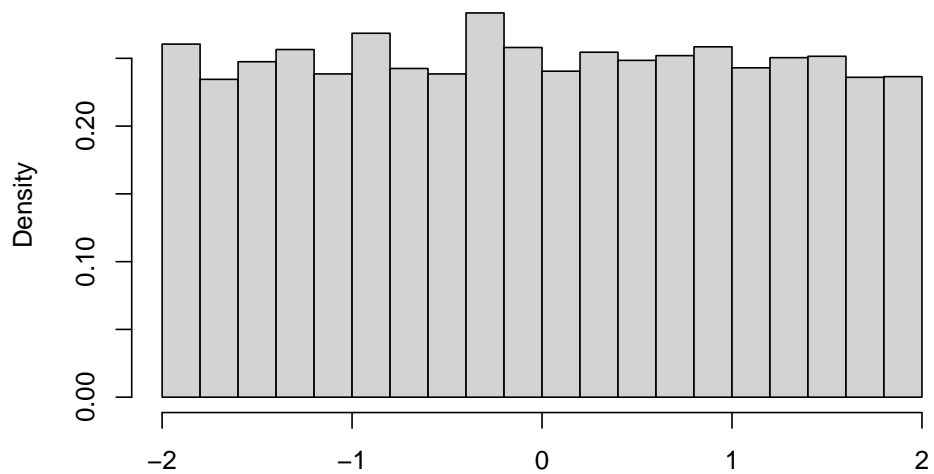
```
n <- 10000
set.seed(123)
simlist <- sqrt(runif(n))
hist(simlist, prob=T, main="", xlab="")
curve(2*x, 0,1, add=T)
```



Exemplo 2: Geração de uma variável aleatória uniforme(a,b). A geração é feita através de

$$X = a + (b - a)U.$$

```
# Geração de uma va uniforme(-2,2)
a <- -2
b <- 2
n <- 10000
set.seed(123)
simlist <- a+(b-a)*runif(n)
hist(simlist, prob=T, main="", xlab="")
```



Exemplo 3: Geração de uma variável aleatória exponencial. Seja X uma variável aleatória exponencial com taxa 1, então sua função de distribuição é

dada por

$$F(x) = 1 - e^{-x}.$$

Como $0 \leq F(x) \leq 1$, tomando $F(x) = u$, onde $u \sim U(0, 1)$ tem-se:

$$u = F(x) = 1 - e^{-x}$$

ou

$$1 - u = e^{-x}$$

ou, aplicando o logaritmo

$$x = -\ln(1 - u).$$

Daí, pode-se gerar uma exponencial com parâmetro 1 gerando um número aleatório U e em seguida fazendo

$$X = F^{-1}(U) = -\ln(1 - U).$$

Uma pequena economia de tempo pode ser obtida notando que $1 - U$ também é uniforme em $(0, 1)$ e assim, $-\ln(1 - U)$ tem a mesma distribuição que $-\ln(U)$. Isto é, o logaritmo negativo de um número aleatório é exponencialmente distribuído com taxa 1.

Além disso, note que se X é uma exponencial com média 1, então para qualquer constante c , cX é uma exponencial com média c . Assim, uma variável aleatória exponencial X com taxa λ (média $\frac{1}{\lambda}$) pode ser gerada através da geração de um número aleatório U e fazendo

$$X = -\frac{1}{\lambda} \ln(U).$$

```
# Definir a sequência de valores x
x <- seq(0,3, by = 0.02)

# Definir o parâmetro lambda da distribuição exponencial
lambda <- 3

# Número de simulação
n <- 10000

# Simular valores de uma distribuição exponencial
set.seed(123)
simlist <- -log(runif(n))/lambda

# Plotar o histograma da simulação com a densidade de probabilidade
hist(simlist, probability = TRUE, main = "Comparação da Distribuição Exponencial Simulada e Teórica",
      xlab = "Valores", ylab = "Densidade", col = "lightblue", border = "black")

# Adicionar a curva de densidade teórica
```

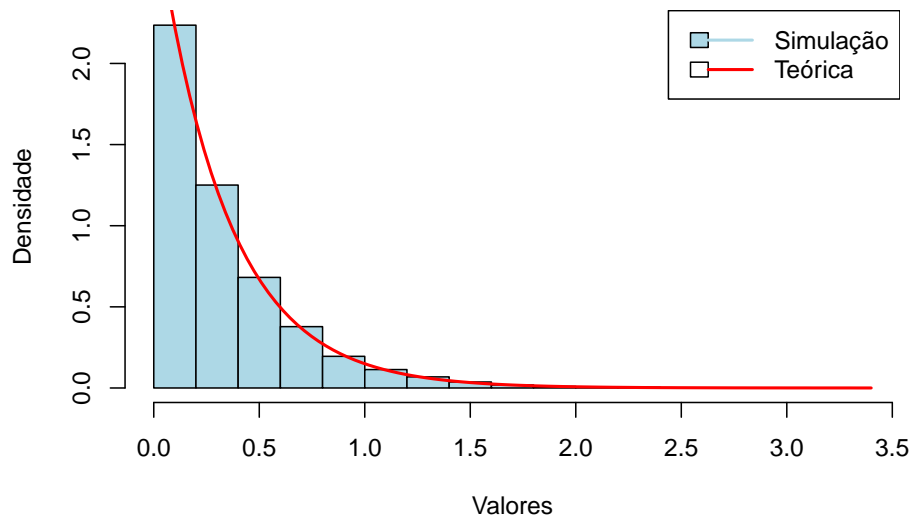
```

curve(dexp(x, rate = lambda), add = TRUE, col = "red", lwd = 2)

# Adicionar uma legenda
legend("topright", legend = c("Simulação", "Teórica"), col = c("lightblue", "red"), lwd = 2)

```

Comparação da Distribuição Exponencial Simulada e Teórica



Exercício: Seja X uma variável aleatória com distribuição $W(\alpha, \beta)$. Assim a fdp de X é

$$f(x) = \begin{cases} \alpha \beta^{-\alpha} x^{\alpha-1} e^{-(x/\beta)^\alpha}, & \text{se } x > 0 \\ 0, & \text{se } x \leq 0 \end{cases}$$

A função de distribuição de X é:

$$F(x) = \int_0^x f(u) du = \begin{cases} 1 - e^{-(x/\beta)^\alpha}, & \text{se } x > 0 \\ 0, & \text{se } x \leq 0 \end{cases}$$

Mostre que $X = \beta[-\ln(U)]^{1/\alpha}$. Gere 10000 valores de uma $W(2, 3)$. Represente graficamente a distribuição.

Chapter 19

Método da aceitação-rejeição

Chapter 20

Distribuições univariadas no R

No R temos acesso as mais comuns distribuições univariadas. Todas as funções tem as seguintes formas:

Função	Descrição
<code>pnome(...)</code>	função de distribuição
<code>dnome(...)</code>	função de probabilidade ou densidade de probabilidade
<code>qnome(...)</code>	inversa da função de distribuição
<code>rnome(...)</code>	geração de números aleatórios com a distribuição especificada

o **nome** é uma abreviatura do nome usual da distribuição (`binom`, `geom`, `pois`, `unif`, `exp`, `norm`, ...).

Exemplo 1: Simule o lançamento de três moedas honestas e a contagem do número de caras X .

(a) Use a sua simulação para estimar $P(X = 1)$ e $E(X)$.

(b) Modifique a alínea anterior para permitir uma moeda viciada onde $P(\text{cara}) = 3/4$.

```
set.seed(123)
n <- 10000
sim1 <- numeric(n)
sim2 <- numeric(n)
for (i in 1:n) {
```

```

moedas <- sample(0:1,3,replace=T)
sim1[i] <- if (sum(moedas)==1) 1 else 0
sim2[i] <- sum(moedas)
}
#  $P(X=1)$ 
mean(sim1)

```

```
## [1] 0.3821
```

```

#  $E(X)$ 
mean(sim2)

```

```
## [1] 1.4928
```

```

set.seed(123)
n <- 10000
sim1 <- numeric(n)
sim2 <- numeric(n)
for (i in 1:n) {
  moedas <- sample(c(0,1),3,prob=c(1/4,3/4),replace=T)
  sim1[i] <- if (sum(moedas)==1) 1 else 0
  sim2[i] <- sum(moedas)
}
#  $P(X=1)$ 
mean(sim1)

```

```
## [1] 0.1384
```

```

#  $E(X)$ 
mean(sim2)

```

```
## [1] 2.2503
```

Sabemos também que X – número de caras no lançamento de três moedas honestas tem distribuição $Binomial(n = 3, p = 0.5)$. Assim, podemos resolver a questão da seguinte maneira

```

set.seed(123)
valores <- rbinom(10000,3,0.5)
#  $P(X=1)$ 
sum(valores == 1)/length(valores)

```

```
## [1] 0.383
```

```
# E(X)
sum(valores)/length(valores)
```

```
## [1] 1.4897
```

```
mean(valores)
```

```
## [1] 1.4897
```

No segundo caso teremos $X \sim \text{Binomial}(n = 3, p = 3/4)$.

```
set.seed(123)
valores <- rbinom(10000, 3, 3/4)
# P(X=1)
sum(valores == 1)/length(valores)
```

```
## [1] 0.1365
```

```
# E(X)
sum(valores)/length(valores)
```

```
## [1] 2.2558
```

```
mean(valores)
```

```
## [1] 2.2558
```

Exemplo 2: O tempo até a chegada de um autocarro tem uma distribuição exponencial com média de 30 minutos.

(a) Use o comando `rexp()` para simular a probabilidade do autocarro chegar nos primeiros 20 minutos.

(b) Use o comando `pexp()` para comparar com a probabilidade exata.

```
set.seed(123)
valores <- rexp(10000, 1/30)
# Probabilidade P(X <= 20)
sum(valores < 20)/length(valores)
```

```
## [1] 0.4832
```

```
# Probabilidade exata
pexp(20, 1/30)
```

```
## [1] 0.4865829
```

Exemplo 3: As cartas são retiradas de um baralho padrão, com reposição, até que um ás apareça. Simule a média e a variância do número de cartas necessárias.

```
set.seed(123)
n <- 10000
# Denote os ases por 1,2,3,4
simlist <- numeric(n)

for (i in 1:n) {
  ct <- 0
  as <- 0
  while (as == 0) {
    carta <- sample(1:52,1,replace=T)
    ct <- ct + 1
    if (carta <= 4){
      as <- 1
    }
  }
  simlist[i] <- ct
}
mean(simlist)
```

```
## [1] 12.8081
```

```
var(simlist)
```

```
## [1] 147.5318
```

Podemos notar aqui também que X — número de provas de Bernoulli até o primeiro sucesso (aparecer um ás), que tem distribuição *Geomtrica*($p = 4/52$). Lembre que o R trabalha com a geométrica como sendo X — número de insucessos até o primeiro sucesso.

```
set.seed(123)

valores <- rgeom(10000, 4/52) + 1

# Média e variância
mean(valores)
```


[1] 13.0108

`var(valores)`

[1] 152.0335

20.1 Função de distribuição empírica

A função de distribuição empírica é uma função de distribuição acumulada que descreve a proporção ou contagem de observações em um conjunto de dados que são menores ou iguais a um determinado valor. É uma ferramenta útil para visualizar a distribuição de dados observados e comparar distribuições amostrais.

- É uma função definida para todo número real x e que para cada x dá a proporção de elementos da amostra menores ou iguais a x :

$$F_n(x) = \frac{\# \text{ observações} \leq x}{n}$$

- Para construir a função de distribuição empírica precisamos primeiramente ordenar os dados em ordem crescente: $(x_{(1)}, \dots, x_{(n)})$
- A definição da função de distribuição empírica é

$$F_n(x) = \begin{cases} 0, & x < x_{(1)} \\ \frac{i}{n}, & x_{(i)} \leq x < x_{(i+1)}, \quad i = 1, \dots, n-1 \\ 1, & x \geq x_{(n)} \end{cases}$$

- Passo a passo para a construção da função
 - Inicie desenhando a função do valor mais à esquerda para o mais à direita.
 - Atribua o valor 0 para todos os valores menores que o menor valor da amostra, $x_{(1)}$.
 - Atribua o valor $\frac{1}{n}$ para o intervalo entre $x_{(1)}$ e $x_{(2)}$, o valor $\frac{2}{n}$ para o intervalo entre $x_{(2)}$ e $x_{(3)}$, e assim por diante, até atingir todos os valores da amostra.
 - Para valores iguais ou superiores ao maior valor da amostra, $x_{(n)}$, a função tomará o valor 1.
 - Se um valor na amostra se repetir k vezes, o salto da função para esse ponto será $\frac{k}{n}$, em vez de $\frac{1}{n}$.

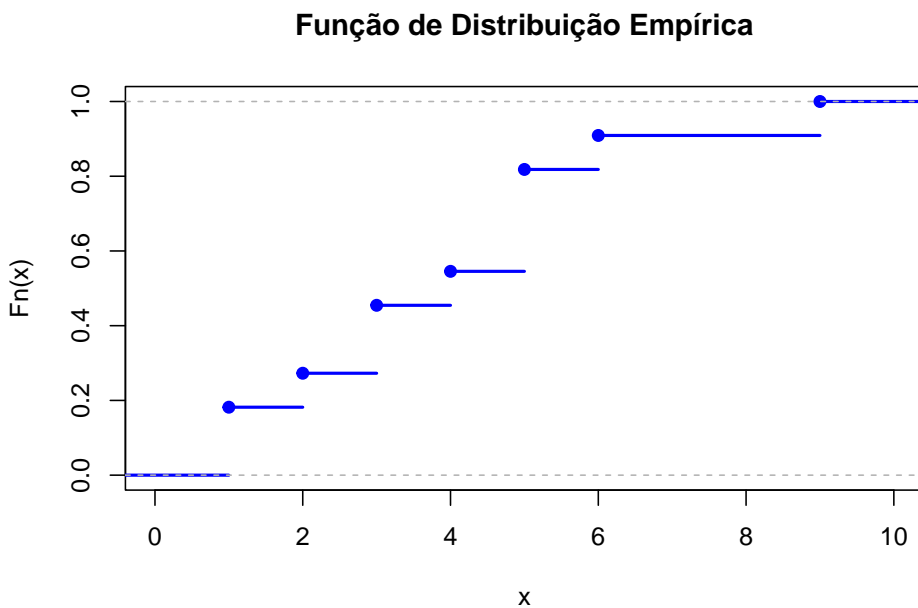
20.1.1 Função de distribuição empírica no R, função `ecdf()`

A função `ecdf()` no R é usada para calcular a função de distribuição empírica (Empirical Cumulative Distribution Function - ECDF) de um conjunto de dados.

```
# Conjunto de dados
dados <- c(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5)

# Calcular a ECDF usando a função ecdf()
Fn <- ecdf(dados)

# Plotar a ECDF usando a função ecdf()
plot(Fn, main = "Função de Distribuição Empírica", xlab = "x", ylab = "Fn(x)", col = "blue")
```



Exemplo 1: Resolva o exemplo 1 usando a função de distribuição empírica.

```
valores <- rexp(10000, 1/30)
# Função de distribuição empírica
Fn <- ecdf(valores)
# Probabilidade P(X<=20)
Fn(20)
```

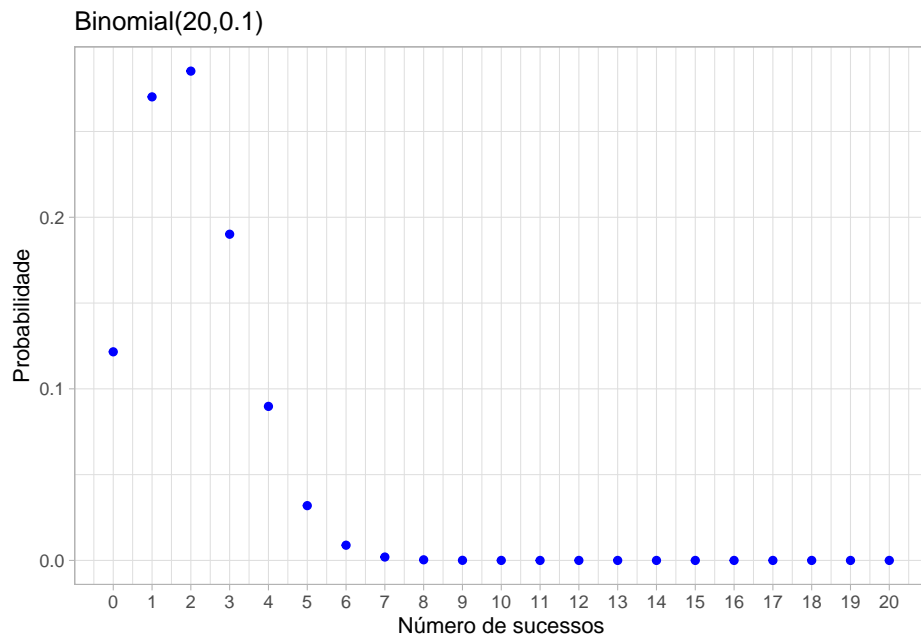
```
## [1] 0.4874
```

```
# Probabilidade exata  
pexp(20, 1/30)
```

```
## [1] 0.4865829
```

20.1.2 Função massa de probabilidade (teórica)

```
# Simulação de Variáveis aleatórias  
  
# Função massa de probabilidade Binomial(n,p)  
n <- 20  
p <- 0.1  
x <- 0:20  
  
teorico <- data.frame(x = x, y=dbinom(x, size = n, prob = p))  
  
# Carregue o pacote ggplot2  
library(ggplot2)  
  
ggplot(teorico) +  
  geom_point(aes(x = x, y=y), color = "blue") +  
  scale_x_continuous(breaks = 0:n) +  
  labs(title = "Binomial(20,0.1)", x = "Número de sucessos", y = "Probabilidade") +  
  theme_light()
```



20.1.3 Função massa de probabilidade (simulação)

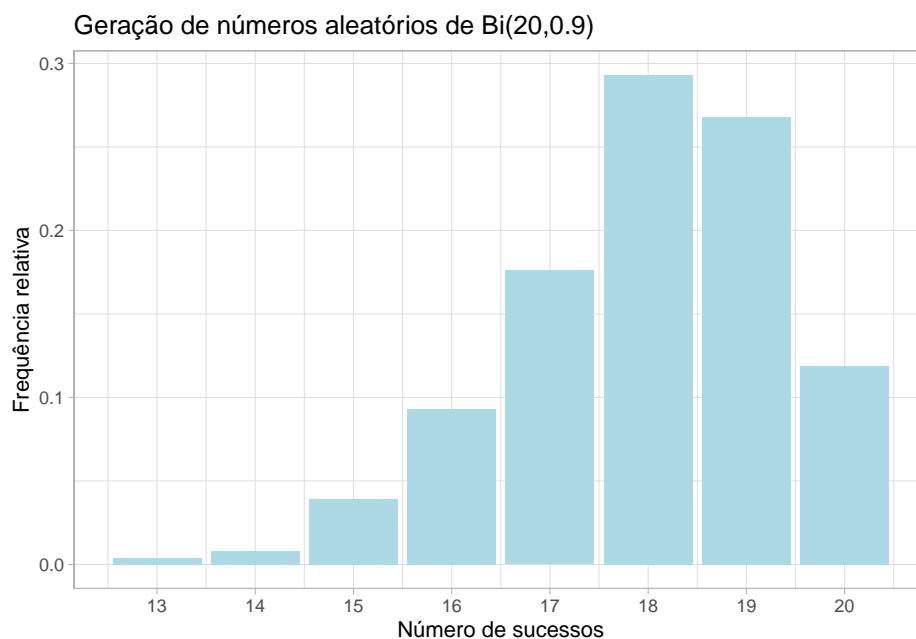
```
set.seed(1234)

n <- 20
p <- 0.9
k <- 1000 # número de simulações

dados <- data.frame(X = rbinom(k, size = n, prob = p))

# Carregue o pacote ggplot2library(ggplot2)

ggplot(dados) +
  geom_bar(aes(x=X, y=after_stat(prop)), fill = "lightblue") +
  scale_x_continuous(breaks = 0:n) +
  labs(title = "Geração de números aleatórios de Bi(20,0.9)", x="Número de sucessos",
y="Frequência relativa") +
  theme_light()
```



20.1.4 Comparação

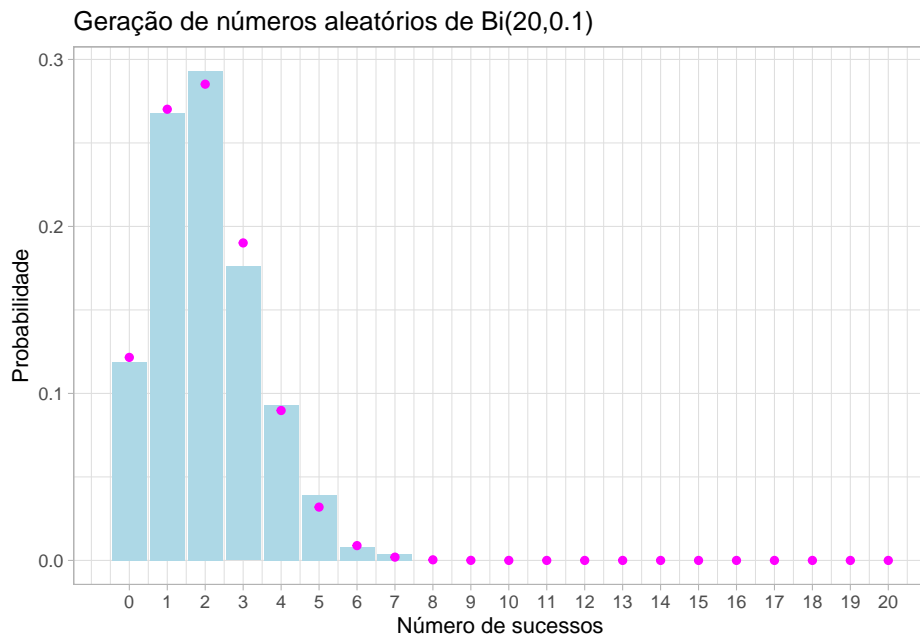
```
set.seed(1234)

n <- 20
p <- 0.1
k <- 1000 # número de simulações

dados <- data.frame(X = rbinom(k, size = n, prob = p))
teorico <- data.frame(x = 0:n, y=dbinom(0:n, size = n, prob = p))

# Carregue o pacote ggplot2
library(ggplot2)

ggplot(dados) +
  geom_bar(aes(x = X, y = after_stat(prop)), fill = "lightblue") +
  geom_point(data = teorico, aes(x, y), color = "magenta") +
  scale_x_continuous(breaks = 0:n) +
  labs(title = "Geração de números aleatórios de Bi(20,0.1)", x = "Número de sucessos",
       y = "Probabilidade") +
  theme_light()
```



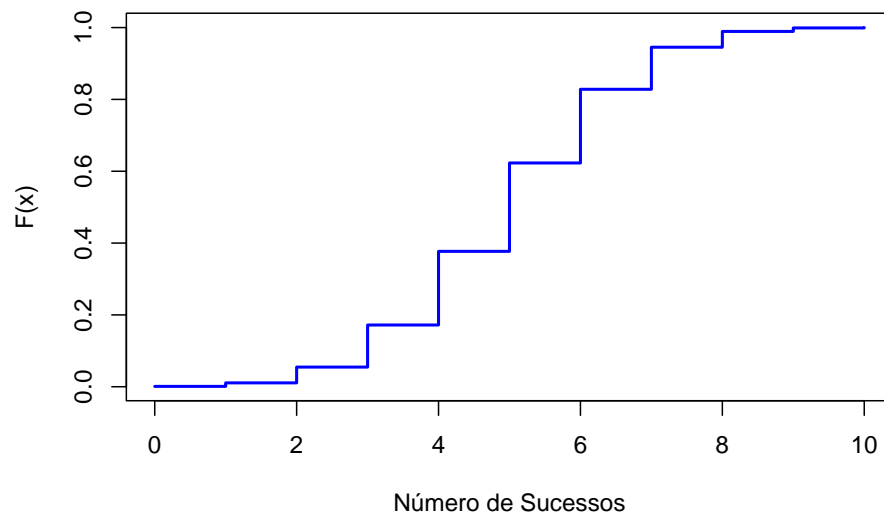
20.1.5 Função de distribuição

```
# Definir os parâmetros da distribuição binomial
n <- 10 # Número de tentativas
p <- 0.5 # Probabilidade de sucesso

# Valores possíveis de sucessos (0 a n)
x <- 0:n

# Calcular a FD
cdf_values <- pbinom(x, size = n, prob = p)

# Plotar a FD
plot(x, cdf_values, type = "s", lwd = 2, col = "blue",
     xlab = "Número de Sucessos", ylab = "F(x)",
     main = "Função de Distribuição Acumulada da Binomial(n = 10, p = 0.5)")
```

Função de Distribuição Acumulada da Binomial($n = 10$, $p = 0.5$)**20.1.6 Função de distribuição empírica**

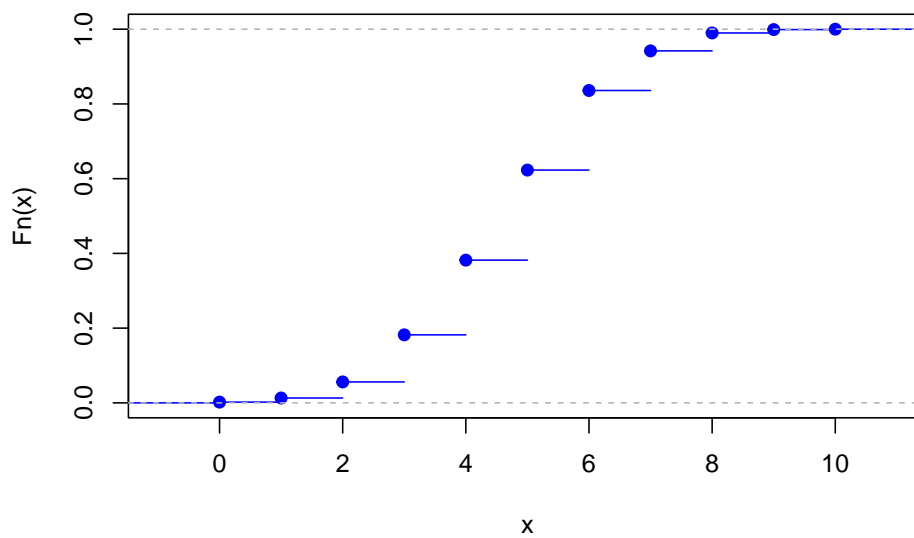
```
# Definir os parâmetros da distribuição binomial
n <- 10 # Número de tentativas
p <- 0.5 # Probabilidade de sucesso

set.seed(123)
# Amostra aleatória de dimensão 1000
amostra <- rbinom(1000, size = n, prob = p)

# Distribuição empírica
Fn <- ecdf(amostra)

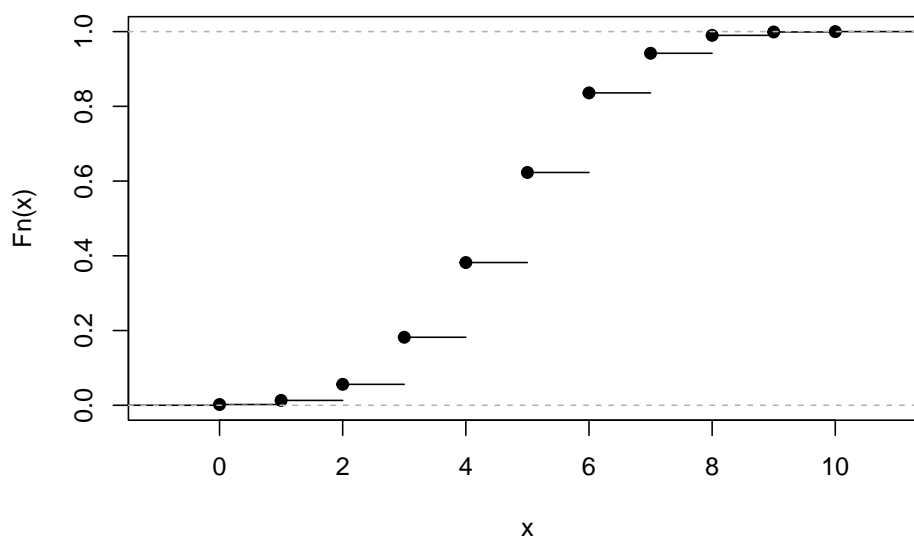
# Plotar CDF
plot(Fn, main = "Função de Distribuição Empírica", xlab = "x",
     ylab = "Fn(x)", col = "blue")
```

Função de Distribuição Empírica



```
# OU
plot.ecdf(amostra)
```

ecdf(x)



Cálculo de probabilidade: Seja $X \sim \text{Binomial}(n = 10, p = 0.5)$.

$$P(X \leq 4) = \text{pbinom}(4, 10, 0.5) = 0.377$$

$$P(X \leq 4) \approx \text{Fn}(4) = 0.382$$

20.2 Gerando uma variável aleatória com distribuição de Poisson

20.2.1 Cálculo de probabilidades

Seja $X \sim \text{Poisson}(\lambda = 5)$.

$P(X = 4) \rightarrow \text{dpois}(4, 5) = 0.1755$

$P(X \leq 4) \rightarrow \text{ppois}(4, 5) = 0.4405$

$P(X > 4) \rightarrow \text{ppois}(4, 5, \text{lower.tail}=\text{FALSE}) = 0.5595$

20.2.2 Função massa de probabilidade (teórica)

```
# Definir os valores de lambda e x
p <- c(0.1, 1, 2.5, 5, 15, 30)
x <- 0:50

# Carregar os pacotes necessários
library(ggplot2)
library(latex2exp)
library(gridExtra)

##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##      combine

# Inicializar uma lista para armazenar os gráficos
plots <- list()

# Loop para criar os data frames e gráficos
for (i in 1:length(p)) {
  teorico <- data.frame(x = x, y = dpois(x, lambda = p[i]))

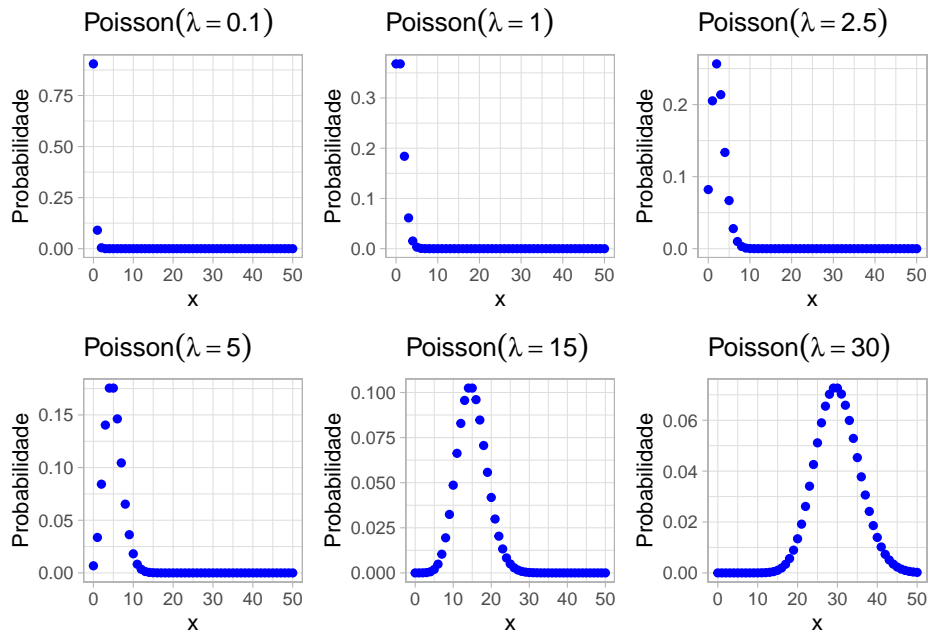
  plots[[i]] <- ggplot(teorico) +
    geom_point(aes(x = x, y = y), color = "blue") +
    scale_x_continuous(breaks = seq(0, 50, by = 10)) +
    labs(title = TeX(paste0("$Poisson(lambda=", p[i], ")$")), x="x", y="Probabilidade") +
    theme_light()
```

```

}

# Dispor os gráficos em uma grade 2x3
grid.arrange(grobs = plots, nrow = 2, ncol = 3)

```



20.2.3 Função massa de probabilidade (simulação)

```

p <- c(0.1, 1, 2.5, 5, 15, 30)
n <- 1000

# Carregar os pacotes necessários
library(ggplot2)
library(latex2exp)
library(gridExtra)

# Inicializar uma lista para armazenar os gráficos
plots <- list()

# Loop para criar os data frames e gráficos
for (i in 1:length(p)) {
  dados <- data.frame(X = rpois(n, lambda = p[i]))

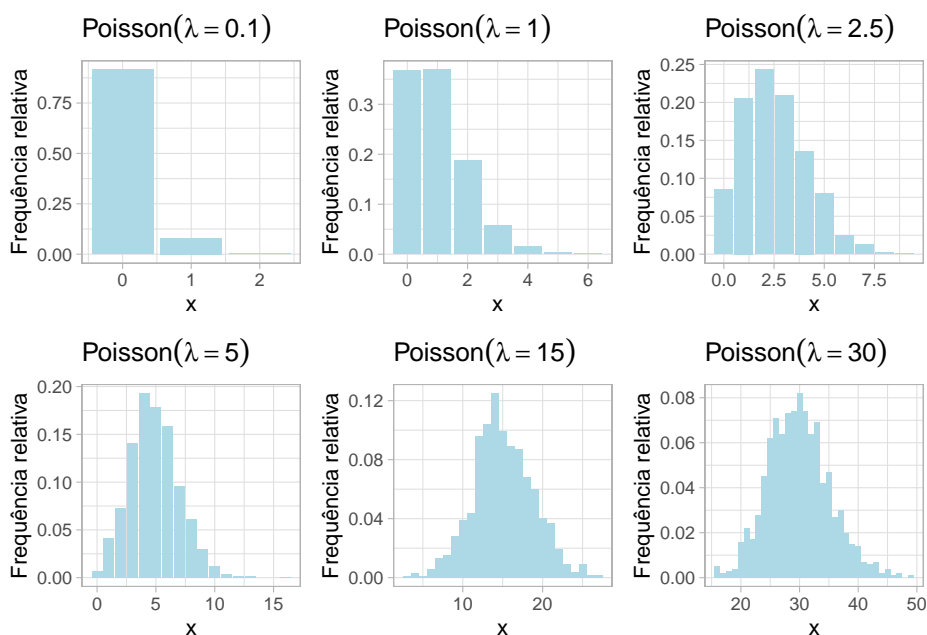
```

```

plots[[i]] <- ggplot(dados) +
  geom_bar(aes(x = X, y = after_stat(prop)), fill = "lightblue") +
  labs(title = TeX(paste("$Poisson(\\lambda=", p[i], ")$")),
       x = "x", y = "Frequência relativa") +
  theme_light()
}

# Dispor os gráficos em uma grade 2x3
grid.arrange(grobs = plots, nrow = 2, ncol = 3)

```



20.2.4 Comparação

```

p <- c(0.1, 1, 2.5, 5, 15, 30)
n <- 1000

# Carregar os pacotes necessários
library(ggplot2)
library(latex2exp)
library(gridExtra)

# Inicializar uma lista para armazenar os gráficos
plots <- list()

```

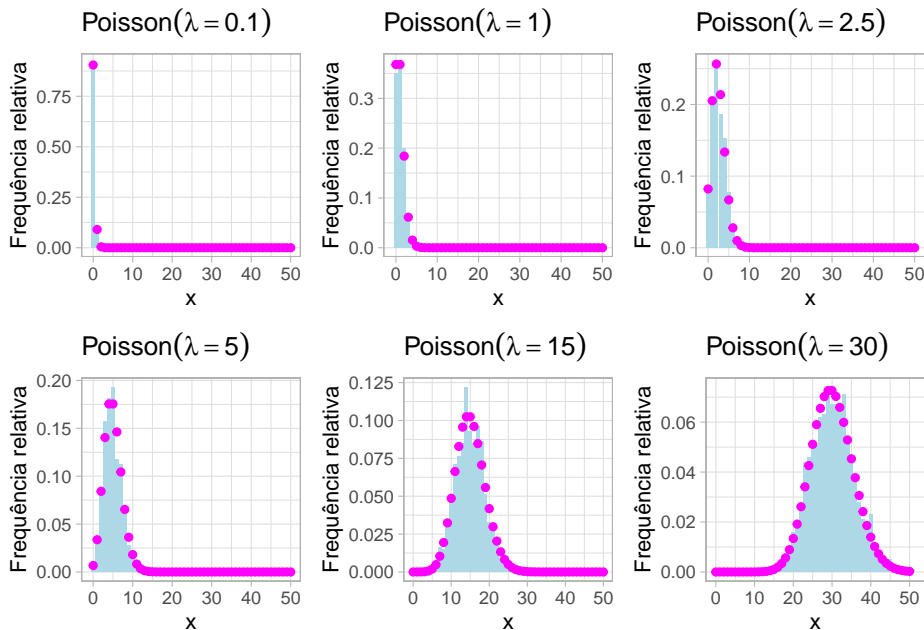
```

# Loop para criar os data frames e gráficos
for (i in 1:length(p)) {
  dados <- data.frame(X = rpois(n, lambda = p[i]))
  teorico <- data.frame(x=0:50, y=dpois(0:50,p[i]))

  plots[[i]] <- ggplot(dados) +
    geom_bar(aes(x = X, y =after_stat(prop)), fill="lightblue") +
    geom_point(data = teorico, aes(x, y), color = "magenta") +
    scale_x_continuous(breaks = seq(0, 50, by = 10)) +
    labs(title=TeX(paste("$Poisson(lambda=", p[i], ")$")),
         x = "x", y = "Frequência relativa") +
    theme_light()
}

# Dispor os gráficos em uma grade 2x3
grid.arrange(grobs = plots, nrow = 2, ncol = 3)

```



20.2.5 Função de distribuição

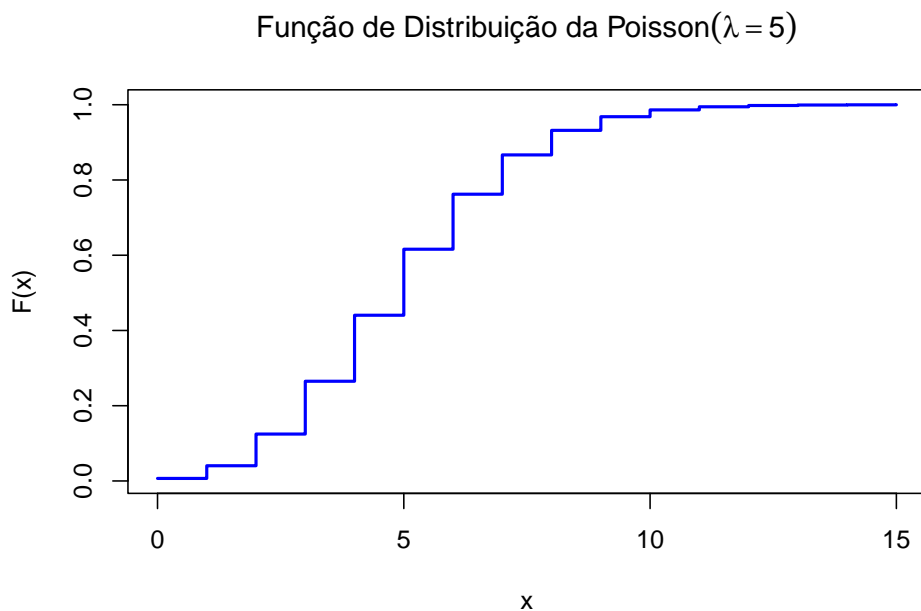
```

lambda <- 5 # Parâmetro da Poisson
x <- 0:15   # Valores de x para plotar a distribuição

```

```
# Calcular a FD
y <- ppois(x, lambda = lambda)

# Plotar a FD
plot(x,y, type="s", lwd=2, col="blue",
      main=TeX(paste("Função de Distribuição da $Poisson (lambda =", lambda, ")$")),
      xlab = "x",
      ylab = "F(x)")
```

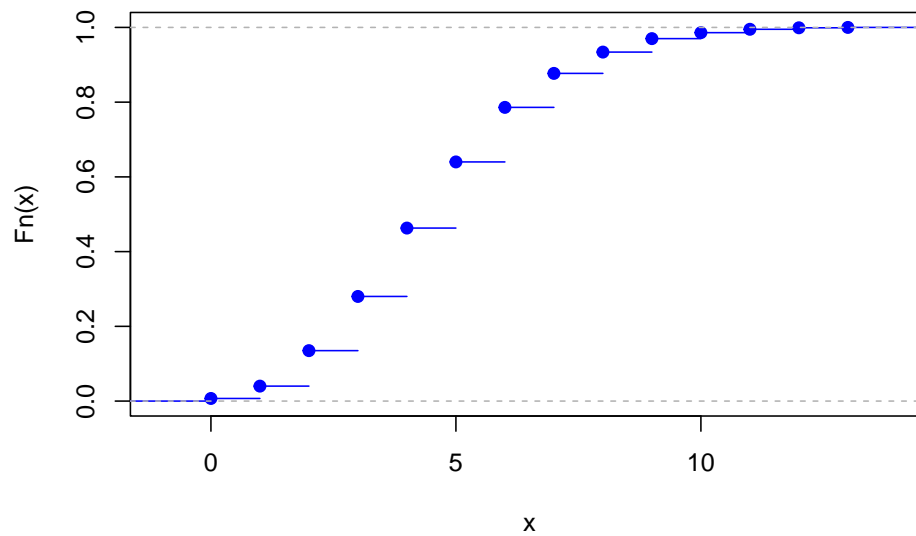


20.2.6 Função de distribuição empírica

```
library(latex2exp)
# Definir os parâmetros da distribuição de Poisson
lambda <- 5

dados <- rpois(1000, lambda = lambda)
Fn <- ecdf(dados)

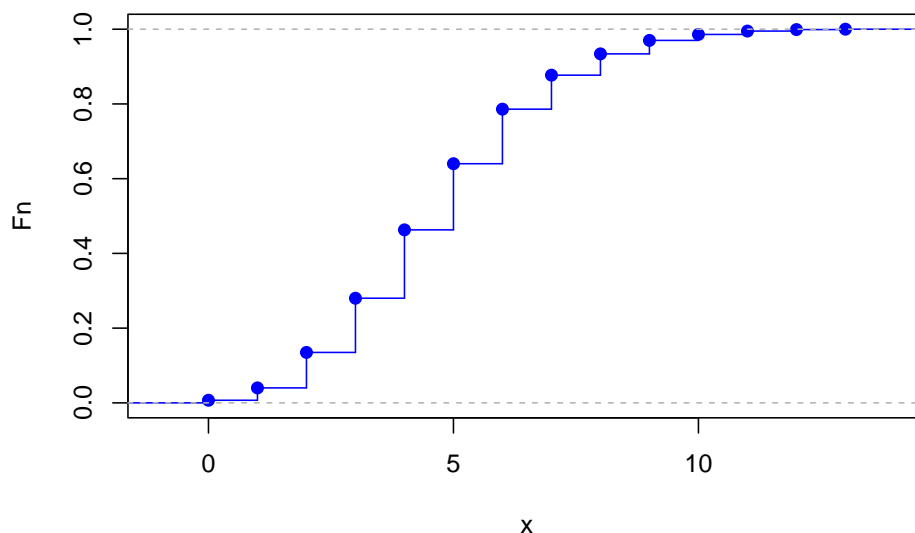
# Plotar CDF
plot(Fn, main=TeX("Função de Distribuição Empírica da $Poisson(lambda = 5)$"),
      xlab = "x",
      ylab = "Fn(x)",
      col = "blue")
```

Função de Distribuição Empírica da Poisson($\lambda = 5$)

```
# OU
#plot.ecdf(dados)

plot(Fn, main="Função de Distribuição Empírica",
     xlab="x",
     ylab="Fn",
     col="blue",
     verticals = TRUE)
```

Função de Distribuição Empírica



Cálculo de probabilidades: Seja $X \sim \text{Poisson}(\lambda = 5)$.

$$P(X \leq 4) \rightarrow \text{ppois}(4, 5) = 0.4405$$

$$P(X \leq 4) \rightarrow F_n(4) = 0.433$$

20.3 Gerando uma variável aleatória com distribuição de Uniforme

20.3.1 Cálculo de probabilidades

Seja $X \sim \text{Uniforme}(0, 1)$

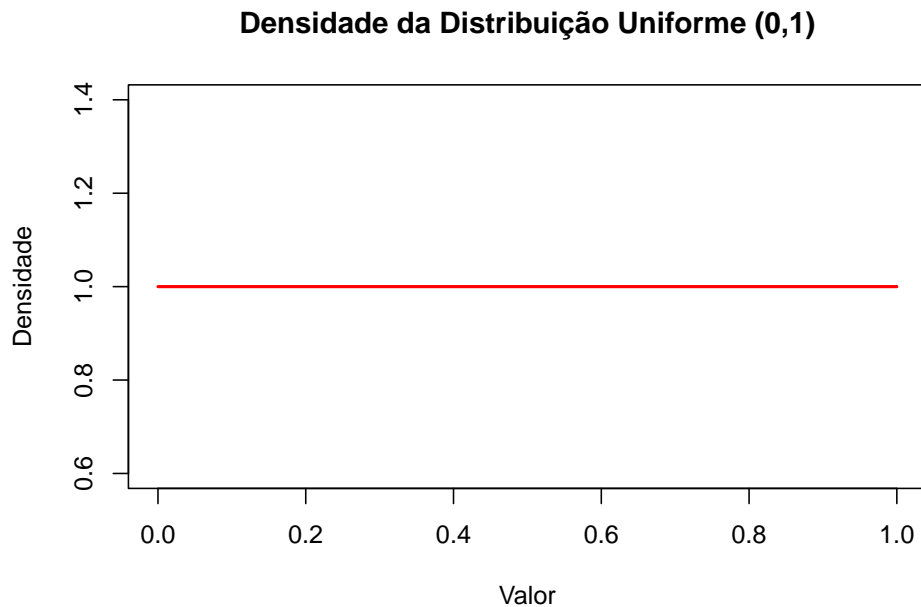
- $P(X \leq 0.5) \rightarrow \text{punif}(0.5, \text{min} = 0, \text{max} = 1) = 0.5$
- $P(X > 0.5) \rightarrow \text{punif}(0.5, \text{min} = 0, \text{max} = 1, \text{lower.tail} = \text{FALSE}) = 0.5$

20.3.2 Função densidade de probabilidade

```
# Gerar os valores x para a densidade teórica
x_vals <- seq(0, 1, length.out = 100)
```

```
# Calcular a densidade teórica para os valores x
y_vals <- dunif(x_vals, min = 0, max = 1)

# Desenhar o gráfico da função densidade de probabilidade
plot(x_vals, y_vals, type = "l",
     col = "red", lwd = 2,
     main = "Densidade da Distribuição Uniforme (0,1)",
     xlab = "Valor", ylab = "Densidade")
```



20.3.3 Função densidade de probabilidade (simulação)

```
# Definir o tamanho da amostra
n <- 10000

# Fixar a semente para reprodutibilidade
set.seed(123)

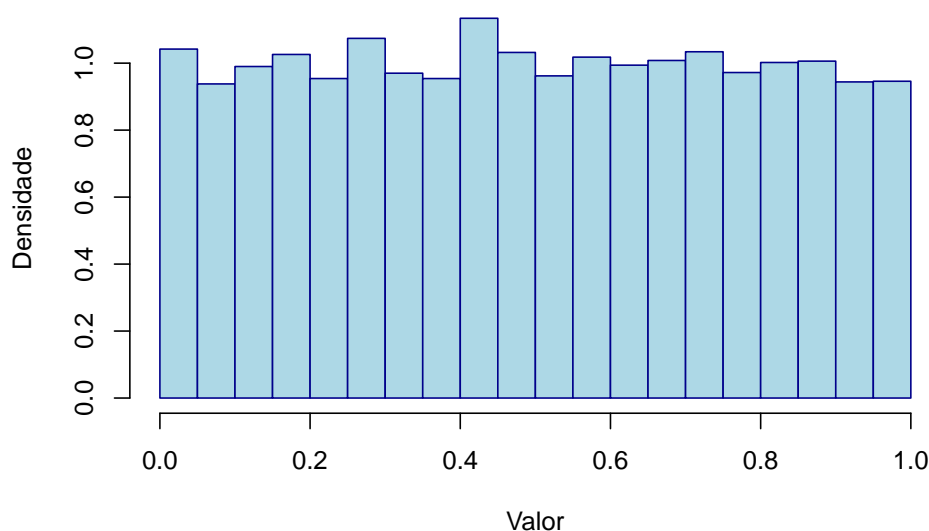
# Gerar a variável aleatória com distribuição uniforme (0,1)
uniform_data <- runif(n, min = 0, max = 1)

# Criar um histograma da amostra
hist(uniform_data, probability = TRUE,
     main = "Histograma da Densidade - Uniforme(0,1)",
```



```
xlab = "Valor",  
ylab = "Densidade",  
col = "lightblue",  
border = "darkblue")
```

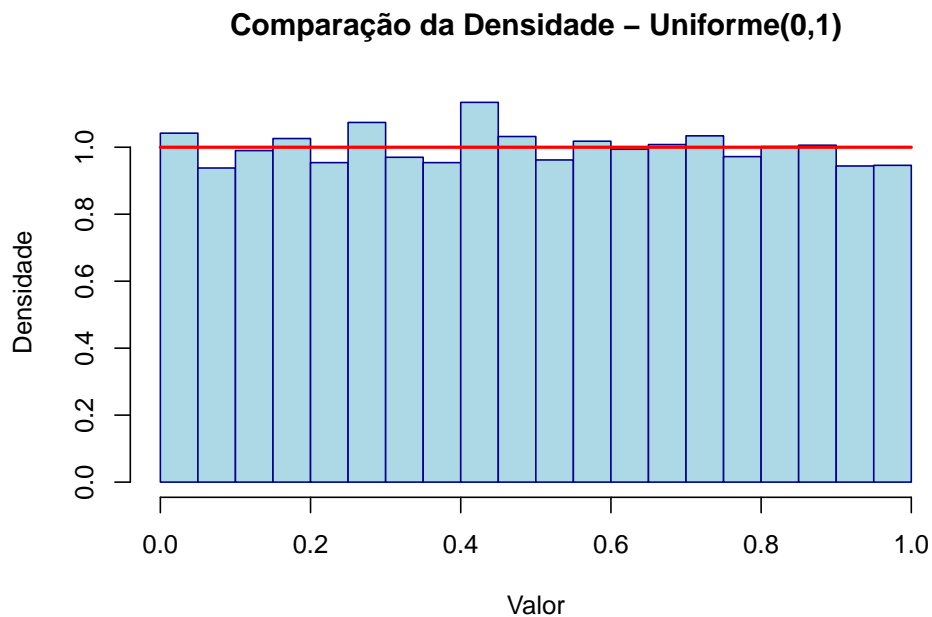
Histograma da Densidade – Uniforme(0,1)



20.3.4 Comparação

```
# Definir o tamanho da amostra  
n <- 10000  
  
# Fixar a semente para reprodutibilidade  
set.seed(123)  
  
# Gerar a variável aleatória com distribuição uniforme (0,1)  
uniform_data <- runif(n, min = 0, max = 1)  
  
# Criar um histograma da amostra com densidade  
hist(uniform_data, probability = TRUE,  
     main = "Comparação da Densidade - Uniforme(0,1)",  
     xlab = "Valor",  
     ylab = "Densidade",  
     col = "lightblue",  
     border = "darkblue")
```

```
# Adicionar a curva da densidade teórica
curve(dunif(x, min = 0, max = 1),
      add = TRUE,
      col = "red",
      lwd = 2)
```



20.3.5 Função de distribuição

```
# Gerar os valores x para a FD teórica
x_vals <- seq(0, 1, length.out = 100)

# Calcular a FD teórica para os valores x
y_vals <- punif(x_vals, min = 0, max = 1)

# Desenhar o gráfico da função de distribuição acumulada
plot(x_vals, y_vals, type = "l",
     col = "blue", lwd = 2,
     main = "Função de Distribuição Uniforme (0,1)",
     xlab = "Valor", ylab = "F(x)")
```

Função de Distribuição Uniforme (0,1)



20.3.6 Função de distribuição empírica

```
# Definir o tamanho da amostra
n <- 10000

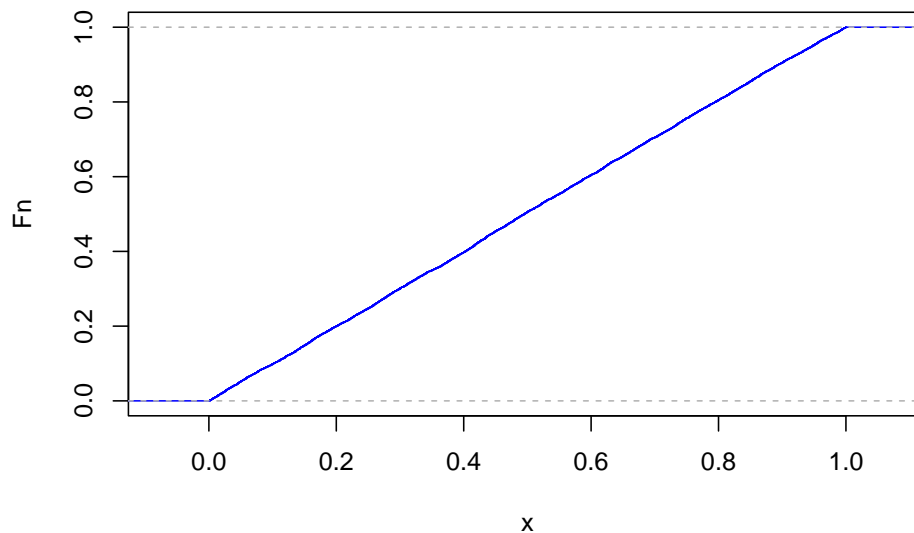
# Fixar a semente para reprodutibilidade
set.seed(123)

# Gerar a variável aleatória com distribuição uniforme (0,1)
uniform_data <- runif(n, min = 0, max = 1)

# Função de distribuição empírica
Fn <- ecdf(uniform_data)

plot(Fn, main="Função de Distribuição Empírica",
     xlab="x",
     ylab="Fn",
     col="blue")
```

Função de Distribuição Empírica



```
# OU  
#plot.ecdf(uniform_data)
```

20.4 Gerando uma variável aleatória com distribuição Exponencial

20.4.1 Cálculo de probabilidades

Seja $X \sim \text{Exponencial}(\lambda = 1)$.

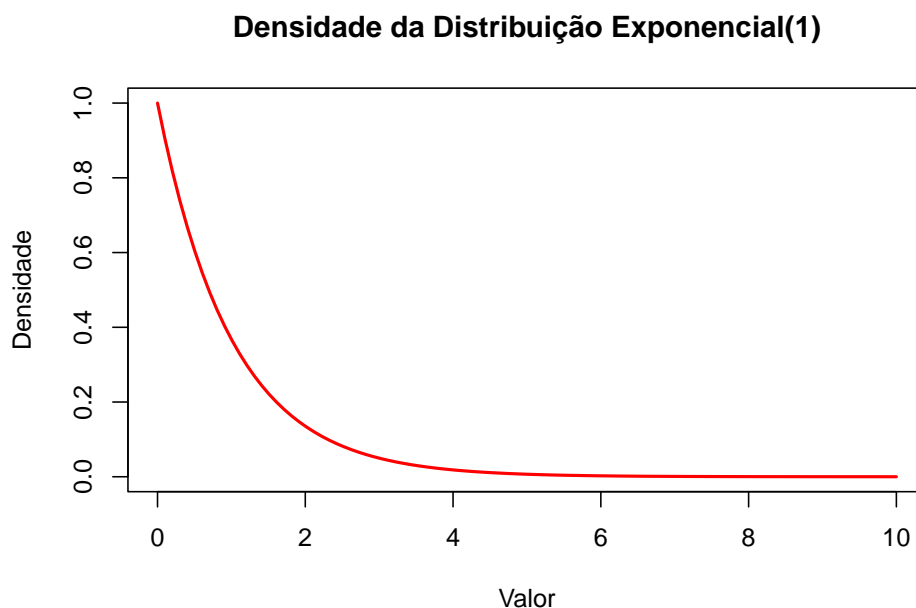
$P(X \leq 0.5) \rightarrow \text{pexp}(0.5, \text{rate}=1) = 0.3935$

$P(X > 0.5) \rightarrow \text{pexp}(0.5, \text{rate}=1, \text{lower.tail}=\text{FALSE}) = 0.6065$

20.4.2 Função densidade de probabilidade (teórica)

```
# Gerar os valores x para a densidade teórica  
x_vals <- seq(0, 10, length.out = 100)  
  
# Calcular a densidade teórica para os valores x  
y_vals <- dexp(x_vals, rate=1)
```

```
# Desenhar o gráfico da função densidade de probabilidade
plot(x_vals, y_vals, type = "l",
     col = "red", lwd = 2,
     main = "Densidade da Distribuição Exponencial(1)",
     xlab = "Valor", ylab = "Densidade")
```



20.4.3 Função densidade de probabilidade (simulação)

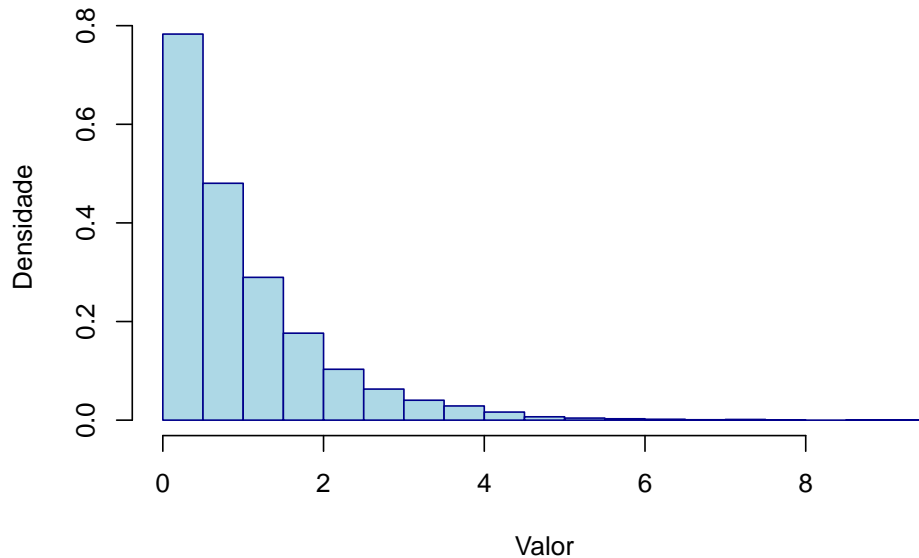
```
# Definir o tamanho da amostra
n <- 10000

# Fixar a semente para reprodutibilidade
set.seed(123)

# Gerar a variável aleatória com distribuição exponencial(1)
expo_data <- rexp(n, rate=1)

# Criar um histograma da amostra
hist(expo_data, probability = TRUE,
     main = "Histograma da Densidade - Exponencial(1)",
     xlab = "Valor",
     ylab = "Densidade",
     col = "lightblue",
     border = "darkblue")
```

Histograma da Densidade – Exponencial(1)



20.4.4 Comparação

```
# Definir o tamanho da amostra
n <- 10000

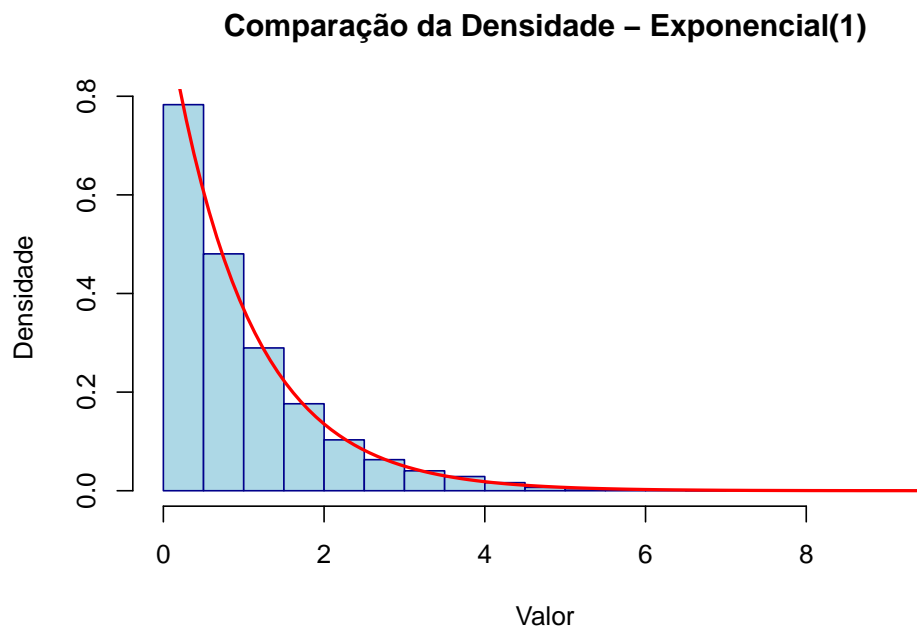
# Fixar a semente para reprodutibilidade
set.seed(123)

# Gerar a variável aleatória com distribuição exponencial(1)
expo_data <- rexp(n, rate=1)

# Criar um histograma da amostra
hist(expo_data, probability = TRUE,
     main = "Comparação da Densidade - Exponencial(1)",
     xlab = "Valor",
     ylab = "Densidade",
     col = "lightblue",
     border = "darkblue")

# Adicionar curva da densidade teórica
curve(dexp(x, rate=1),
      add=TRUE,
      col="red",
```

```
lwd=2)
```



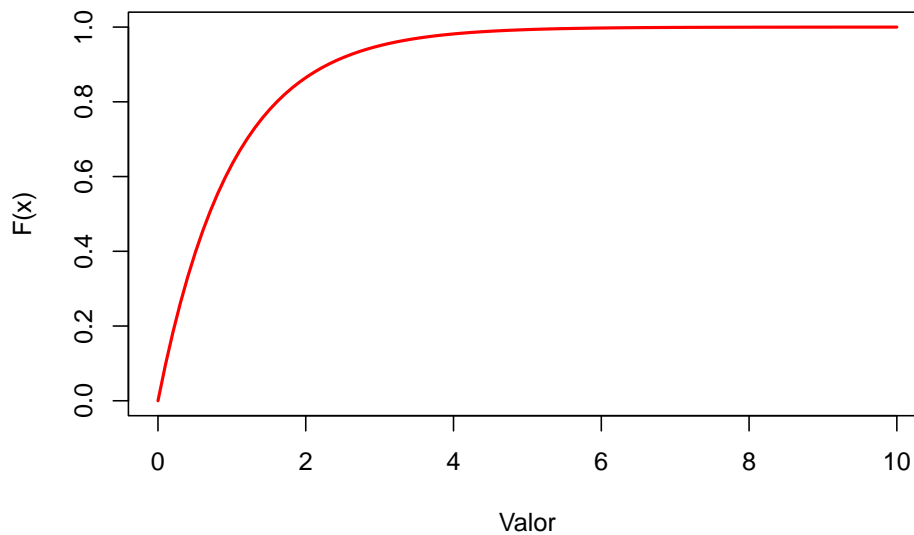
20.4.5 Função de distribuição

```
# Gerar os valores x para a FD teórica
x_vals <- seq(0, 10, length.out = 100)

# Calcular a FD teórica para os valores x
y_vals <- pexp(x_vals, rate=1)

# Desenhar o gráfico da FD
plot(x_vals, y_vals, type = "l",
     col = "red", lwd = 2,
     main = "Função de Distribuição Exponencial(1)",
     xlab = "Valor", ylab = "F(x)")
```

Função de Distribuição Exponencial(1)



20.4.6 Função de distribuição empírica

```
# Definir o tamanho da amostra
n <- 10000

# Fixar a semente para reprodutibilidade
set.seed(123)

# Gerar a variável aleatória com distribuição exponencial(1)
expo_data <- rexp(n, rate=1)

# Função de distribuição empírica
Fn <- ecdf(expo_data)

plot(Fn, main="Função de Distribuição Empírica",
     xlab="x",
     ylab="Fn",
     col="blue")
```


Função de Distribuição Empírica



20.5 Gerando uma variável aleatória com distribuição Normal

20.5.1 Cálculo de probabilidades

Seja $X \sim \text{Normal}(0, 1)$.

$P(X \leq 0.5) \rightarrow \text{pnorm}(0.5, \text{mean}=0, \text{sd}=1) = 0.6915$

$P(X > 0.5) \rightarrow \text{pnorm}(0.5, \text{mean}=0, \text{sd}=1, \text{lower.tail}=\text{FALSE}) = 0.3085$

20.5.2 Função densidade de probabilidade (teórica)

```
# Gerar os valores x para a densidade teórica
x_vals <- seq(-5, 5, length.out = 100)

# Calcular a densidade teórica para os valores x
y_vals <- dnorm(x_vals, mean = 0, sd = 1)

# Desenhar o gráfico da função densidade de probabilidade
plot(x_vals, y_vals, type = "l",
     col = "red", lwd = 2,
```

```
main = "Densidade da Distribuição Normal(0,1)",  
xlab = "Valor", ylab = "Densidade")
```



20.5.3 Função densidade de probabilidade (simulação)

```
# Definir o tamanho da amostra  
n <- 10000  
  
# Fixar a semente para reprodutibilidade  
set.seed(123)  
  
# Gerar a variável aleatória com distribuição Normal(0,1)  
normal_data <- rnorm(n, mean = 0, sd = 1)  
  
# Criar um histograma da amostra com densidade  
hist(normal_data, probability = TRUE,  
      main = "Comparação da Densidade - Normal(0,1)",  
      xlab = "Valor",  
      ylab = "Densidade",  
      col = "lightblue",  
      border = "darkblue")
```

Comparação da Densidade – Normal(0,1)



20.5.4 Comparação

```
# Definir o tamanho da amostra
n <- 10000

# Fixar a semente para reprodutibilidade
set.seed(123)

# Gerar a variável aleatória com distribuição Normal(0,1)
normal_data <- rnorm(n, mean = 0, sd = 1)

# Criar um histograma da amostra com densidade
hist(normal_data, probability = TRUE,
      main = "Comparação da Densidade - Normal(0,1)",
      xlab = "Valor",
      ylab = "Densidade",
      col = "lightblue",
      border = "darkblue")

# Adicionar a curva da densidade teórica
curve(dnorm(x, mean = 0, sd = 1),
      add = TRUE,
      col = "red",
      lwd = 2)
```

Comparação da Densidade – Normal(0,1)**20.5.5 Função de distribuição**

```
# Gerar os valores x para a FD teórica
x_vals <- seq(-5, 5, length.out = 100)

# Calcular a FD teórica para os valores x
y_vals <- pnorm(x_vals, mean = 0, sd = 1)

# Desenhar o gráfico da função de distribuição
plot(x_vals, y_vals, type = "l",
     col = "blue", lwd = 2,
     main = "Função de Distribuição Normal(0,1)",
     xlab = "Valor", ylab = "F(x)")
```

Função de Distribuição Normal(0,1)**20.5.6 Função de distribuição empírica**

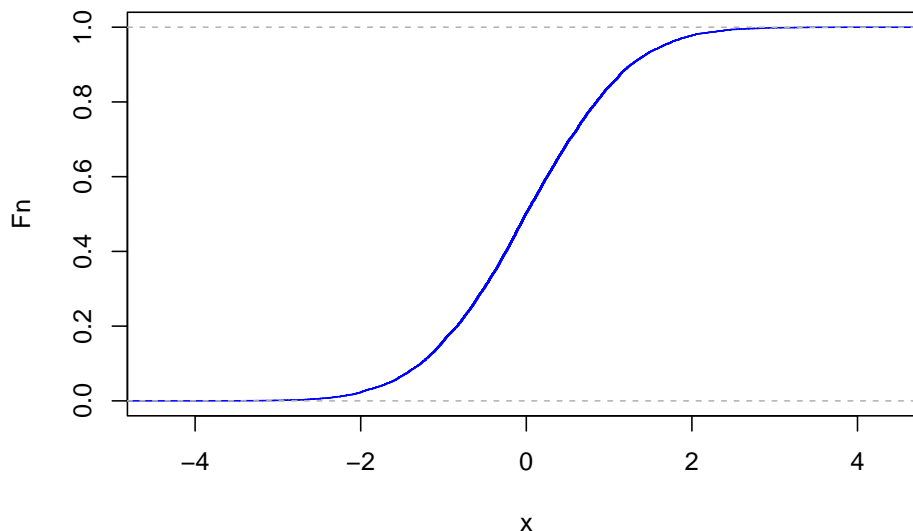
```
# Definir o tamanho da amostra
n <- 10000

# Fixar a semente para reprodutibilidade
set.seed(123)

# Gerar a variável aleatória com distribuição Normal(0,1)
normal_data <- rnorm(n, mean = 0, sd = 1)

# Função de distribuição empírica
Fn <- ecdf(normal_data)

plot(Fn, main="Função de Distribuição Empírica",
     xlab="x",
     ylab="Fn",
     col="blue")
```

Função de Distribuição Empírica**20.6 Exercícios**

1. Usando o R e fixando a semente em 123, simule 1000 lançamentos de uma moeda com probabilidade de 0.5 de sair cara. Conte o número de caras em cada lançamento e plote um histograma dos resultados.
2. Usando o R e fixando a semente em 123, gere uma amostra aleatória de 5000 observações de uma variável aleatória binomial com parâmetros $n = 10$ e $p = 0.3$. Calcule a média e a variância das observações geradas.
3. Usando o R e fixando a semente em 123, gere uma amostra aleatória de 2300 observações de uma variável aleatória de Poisson com parâmetro $\lambda = 4$. Calcule a média e o desvio padrão das observações geradas.
4. Em um processo de qualidade, considere uma variável aleatória X que representa o número de produtos defeituosos em um lote de 50 produtos, onde a probabilidade de um produto ser defeituoso é 0.1. Usando o R e fixando a semente em 123 gere uma amostra aleatória de 10000 observações de X . Conte a frequência de lotes com exatamente 5 produtos defeituosos. Calcule a proporção de lotes com exatamente 5 produtos defeituosos e compare o valor obtido com a probabilidade $P(X = 5)$, onde $X \sim \text{Binomial}(50, 0.1)$.
5. Usando o R e fixando a semente em 123, gere uma amostra aleatória de 5000 observações de uma variável aleatória X binomial com parâmetros $n = 20$ e $p = 0.7$.

(a) Faça um histograma de frequência relativa associado aos valores amostrais. Sobreponha no gráfico a distribuição de probabilidade de X .

(b) Use a função de distribuição empírica para estimar $P(X \leq 10)$ e compare com o valor teórico.

6. Usando o R e fixando a semente em 543, gere uma amostra aleatória de 2400 observações de uma variável aleatória Y de Poisson com parâmetro $\lambda = 6$.

(a) Faça um histograma de frequência relativa associado aos valores amostrais. Sobreponha no gráfico a distribuição de probabilidade de X .

(b) Use a função de distribuição empírica para estimar $P(Y > 5)$ e compare com o valor teórico.

7. Usando o R e fixando a semente em 345, gere uma amostra aleatória de 3450 observações de uma variável aleatória Z uniforme no intervalo $[0, 1]$. Use a função de distribuição empírica para estimar $P(Z \leq 0.5)$ e compare com o valor teórico.

8. Usando o R e fixando a semente em 123, gere uma amostra aleatória de 3467 observações de uma variável aleatória W normal com média $\mu = 0$ e desvio padrão $\sigma = 1$.

(a) Faça um histograma de frequência relativa associado aos valores amostrais. Sobreponha no gráfico a distribuição de X .

(b) Use a função de distribuição empírica para estimar $P(W > 1)$ e compare com o valor teórico.

9. Usando o R e fixando a semente em 123, gere uma amostra aleatória de 1234 observações de uma variável aleatória V exponencial com parâmetro $\lambda = 0.5$.

(a) Faça um histograma de frequência relativa associado aos valores amostrais. Sobreponha no gráfico a distribuição de probabilidade de X .

(b) Use a função de distribuição empírica para estimar $P(V > 2)$ e compare com o valor teórico.

10. O número de acertos num alvo em 30 tentativas onde a probabilidade de acerto é 0.4, é modelado por uma variável aleatória X com distribuição Binomial de parâmetros $n = 30$ e $p = 0.4$. Usando o R e fixando a semente em 123, gere uma amostra de dimensão $n = 700$ dessa variável. Para essa amostra:

(a) Faça um histograma de frequência relativa associado aos valores amostrais. Sobreponha no gráfico a distribuição de probabilidade de X .

(b) Calcule a função de distribuição empírica e com base nessa função estime a probabilidade do número de acertos no alvo, em 30 tentativas, ser maior que 15. Calcule ainda o valor teórico dessa probabilidade.

11. Usando o R e fixando a semente em 123, gere amostras de tamanho crescente $n = 100, 1000, 10000, 100000$ de uma variável aleatória X com distribuição

de Poisson com parâmetro $\lambda = 3$. Para cada tamanho de amostra, calcule a média amostral e compare-a com o valor esperado teórico. Observe e comente a convergência das médias amostrais.

12. Usando o R e fixando a semente em 123, gere amostras de tamanho crescente $n = 100, 1000, 10000, 100000$ de uma variável aleatória W com distribuição uniforme no intervalo $[0, 1]$. Para cada tamanho de amostra, calcule a média amostral e compare-a com o valor esperado teórico. Observe e comente a convergência das médias amostrais.

13. Um grupo de estudantes de Estatística está realizando uma pesquisa para avaliar o grau de satisfação dos alunos com um novo curso oferecido pela universidade. Cada estudante responde a uma pergunta onde pode indicar se está satisfeito ou insatisfeito com o curso. A probabilidade de um estudante estar satisfeito é de 0.75.

- Usando o R e fixando a semente em 42, simule amostras de tamanho crescente $n = 100, 500, 1000, 5000, 10000$ de uma variável aleatória X com distribuição binomial, onde X representa o número de estudantes satisfeitos. Para cada tamanho de amostra, calcule a proporção de estudantes satisfeitos e compare-a com a probabilidade teórica de satisfação (0.75).

14. Usando o R e fixando a semente em 1058, gere 9060 amostras de dimensão 9 de uma população, $X \sim \text{Binomial}(41, 0.81)$. Calcule a média de cada uma dessas amostras, obtendo uma amostra de médias. Calcule ainda o valor esperado da distribuição teórica de X e compare com a média da amostra de médias.

15. Em um hospital, o tempo de atendimento de pacientes segue uma distribuição exponencial com média de 30 minutos. Um pesquisador deseja estimar o tempo médio de atendimento coletando amostras de diferentes tamanhos.

- Usando o R e fixando a semente em 456, simule 1000 amostras de tamanho 50, 100 e 1000 do tempo de atendimento. Para cada tamanho de amostra, calcule a média de cada amostra e plote o histograma das médias amostrais para cada tamanho. Compare essas distribuições com a distribuição normal com média $E(X)$ e desvio padrão $\sqrt{V(X)n}$ e comente sobre a aplicação do Teorema do Limite Central.

16. O tempo de espera (em minutos) para o atendimento no setor de informações de um banco é modelado por uma variável aleatória X com distribuição Uniforme($a = 5, b = 20$). Usando o R e fixando a semente em 1430, gere 8000 amostras de dimensão $n = 100$ dessa variável. Para essas amostras:

(a) Calcule a soma de cada uma das amostras obtendo assim valores da distribuição da soma $S_n = \sum_{i=1}^n X_n$.

(b) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da soma e sobreponha no gráfico uma curva com distribuição normal de valor esperado $nE(X)$ e desvio padrão $\sqrt{V(X)n}$.

(c) Calcule a média de cada uma das amostras obtendo assim valores da distribuição da média \bar{X}_n .

(d) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da média \bar{X}_n . Sobreponha no gráfico uma curva com distribuição normal com valor esperado $E(X)$ e desvio padrão $\sqrt{V(x)/n}$.

17. O tempo de atendimento (em minutos), de doentes graves num determinado hospital, é modelado por uma variável aleatória X com distribuição Exponencial($\lambda = 0.21$). Usando o R e fixando a semente em 1580, gere 1234 amostras de dimensão $n = 50$ dessa variável. Para essas amostras:

(a) Calcule a soma de cada uma das amostras obtendo assim valores da distribuição da soma $S_n = \sum_{i=1}^n X_n$.

(b) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da soma e sobreponha no gráfico uma curva com distribuição normal de valor esperado $nE(X)$ e desvio padrão $\sqrt{V(X)n}$.

(c) Calcule agora a soma padronizada

$$\frac{S_n - E(S_n)}{\sqrt{V(S_n)}}$$

e faça um histograma de frequência relativa associado aos valores obtidos da distribuição da soma padronizada. Sobreponha no gráfico uma curva com distribuição normal de valor esperado 0 e desvio padrão 1.

(d) Calcule a média de cada uma das amostras obtendo assim valores da distribuição da média \bar{X}_n .

(e) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da média \bar{X}_n . Sobreponha no gráfico uma curva com distribuição normal com valor esperado $E(X)$ e desvio padrão $\sqrt{V(x)/n}$.

18. A altura (em centímetros) dos alunos de uma escola é modelada por uma variável aleatória X com distribuição Normal($\mu = 170, \sigma = 10$). Usando o R e fixando a semente em 678, gere 9876 amostras de dimensão $n = 80$ dessa variável. Para essas amostras:

(a) Calcule a soma de cada uma das amostras obtendo assim valores da distribuição da soma $S_n = \sum_{i=1}^n X_n$.

(b) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da soma e sobreponha no gráfico uma curva com distribuição normal de valor esperado $nE(X)$ e desvio padrão $\sqrt{V(X)n}$.

(c) Calcule agora a soma padronizada

$$\frac{S_n - E(S_n)}{\sqrt{V(S_n)}}$$

e faça um histograma de frequência relativa associado aos valores obtidos da distribuição da soma padronizada. Sobreponha no gráfico uma curva com distribuição normal de valor esperado 0 e desvio padrão 1.

(d) Calcule a média de cada uma das amostras obtendo assim valores da distribuição da média \bar{X}_n .

(e) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da média \bar{X}_n . Sobreponha no gráfico uma curva com distribuição normal com valor esperado $E(X)$ e desvio padrão $\sqrt{V(x)/n}$.

(f) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da média padronizada

$$\frac{\bar{X}_n - E(\bar{X}_n)}{\sqrt{V(\bar{X}_n)}}$$

e sobreponha no gráfico com uma curva com distribuição Normal com valor esperado 0 e desvio padrão 1.

19. A chegada de clientes em uma loja durante 1 hora, assumindo uma taxa média de 20 clientes por hora pode ser modelada por uma variável aleatória X com distribuição de Poisson($\lambda = 20$). Usando o R e fixando a semente em 1222, gere 8050 amostras de dimensão 30 de X .

(a) Calcule a soma de cada uma das amostras obtendo assim valores da distribuição da soma $S_n = \sum_{i=1}^n X_n$.

(b) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da soma e sobreponha no gráfico uma curva com distribuição normal de valor esperado $nE(X)$ e desvio padrão $\sqrt{V(X)n}$.

(c) Calcule agora a soma padronizada

$$\frac{S_n - E(S_n)}{\sqrt{V(S_n)}}$$

e faça um histograma de frequência relativa associado aos valores obtidos da distribuição da soma padronizada. Sobreponha no gráfico uma curva com distribuição normal de valor esperado 0 e desvio padrão 1.

(d) Calcule a média de cada uma das amostras obtendo assim valores da distribuição da média \bar{X}_n .

(e) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da média \bar{X}_n . Sobreponha no gráfico uma curva com distribuição normal com valor esperado $E(X)$ e desvio padrão $\sqrt{V(x)/n}$.

(f) Faça um histograma de frequência relativa associado aos valores obtidos da distribuição da média padronizada

$$\frac{\bar{X}_n - E(\bar{X}_n)}{\sqrt{V(\bar{X}_n)}}$$

e sobreponha no gráfico com uma curva com distribuição Normal com valor esperado 0 e desvio padrão 1.

Chapter 21

Relatórios

21.1 Markdown

21.2 R Markdown

Chapter 22

Referências

- <https://cemapre.iseg.ulisboa.pt/~nbrites/CTA/index.html>
- <https://livro.curso-r.com/>

Chapter 23

Respostas

23.1 O pacote dplyr

23.1.1 Selecionando colunas

1. Teste aplicar a função `glimpse()` do pacote `dplyr` à base `sw`. O que ela faz?

```
glimpse(sw)
```

```
## Rows: 87
## Columns: 14
## $ name      <chr> "Luke Skywalker", "C-3P0", "R2-D2", "Darth Vader", "Leia Or~
## $ height    <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, 182, 188, 180, 2~
## $ mass      <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, 32.0, 84.0, 77.~
## $ hair_color <chr> "blond", NA, NA, "none", "brown", "brown, grey", "brown", N~
## $ skin_color <chr> "fair", "gold", "white, blue", "white", "light", "light", "~
## $ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", "blue", "blue",~
## $ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, NA, 24.0, 57.0, ~
## $ sex        <chr> "male", "none", "none", "male", "female", "male", "female",~
## $ gender     <chr> "masculine", "masculine", "masculine", "masculine", "femini~
## $ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine", "Alderaan", "T~
## $ species    <chr> "Human", "Droid", "Droid", "Human", "Human", "Human", "Huma~
## $ films      <list> <"A New Hope", "The Empire Strikes Back", "Return of the J~
## $ vehicles   <list> <"Snowspeeder", "Imperial Speeder Bike">, <>, <>, <>, "Imp~
## $ starships  <list> <"X-wing", "Imperial shuttle">, <>, <>, "TIE Advanced x1",~
```

Mostra os nomes das variáveis, os tipos de dados e os primeiros valores de cada coluna em uma única visualização, tudo de forma horizontal.

2. Crie uma tabela com apenas as colunas `name`, `gender`, e `films`. Salve em um objeto chamado `sw_simples`.

```
sw_simples <- select(sw, name, gender, films)
sw_simples
```

```
## # A tibble: 87 x 3
##   name                gender  films
##   <chr>              <chr>   <list>
## 1 Luke Skywalker    masculine <chr [5]>
## 2 C-3P0             masculine <chr [6]>
## 3 R2-D2             masculine <chr [7]>
## 4 Darth Vader       masculine <chr [4]>
## 5 Leia Organa       feminine <chr [5]>
## 6 Owen Lars         masculine <chr [3]>
## 7 Beru Whitesun Lars feminine <chr [3]>
## 8 R5-D4             masculine <chr [1]>
## 9 Biggs Darklighter masculine <chr [1]>
## 10 Obi-Wan Kenobi    masculine <chr [6]>
## # i 77 more rows
```

3. Selecione apenas as colunas `hair_color`, `skin_color` e `eye_color` usando a função auxiliar `contains()`.

```
select(sw, contains("color"))
```

```
## # A tibble: 87 x 3
##   hair_color  skin_color eye_color
##   <chr>      <chr>    <chr>
## 1 blond     fair      blue
## 2 <NA>      gold      yellow
## 3 <NA>      white, blue red
## 4 none      white     yellow
## 5 brown     light     brown
## 6 brown, grey light     blue
## 7 brown     light     blue
## 8 <NA>      white, red red
## 9 black     light     brown
## 10 auburn, white fair      blue-gray
## # i 77 more rows
```

4. Usando a função `select()` (e suas funções auxiliares), escreva códigos que retornem a base `sw` sem as colunas `hair_color`, `skin_color` e `eye_color`. Escreva todas as soluções diferentes que você conseguir pensar.

```
select(sw, -hair_color, -skin_color, -eye_color)

select(sw, -contains("color"))

select(sw, -ends_with("color"))

select(sw, name:mass, birth_year:starships)
```

23.1.2 Ordenando a base

1. Ordene `mass` em ordem crescente e `birth_year` em ordem decrescente e salve em um objeto chamado `sw_ordenados`.

```
sw_ordenados <- arrange(sw, mass, desc(birth_year))
sw_ordenados
```

```
## # A tibble: 87 x 14
##   name      height  mass hair_color skin_color eye_color birth_year sex  gender
##   <chr>      <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr> <chr>
## 1 Ratts T~    79    15 none      grey, blue unknown      NA male  mascu~
## 2 Yoda        66    17 white     green      brown      896 male  mascu~
## 3 Wicket ~    88    20 brown     brown      brown      8 male  mascu~
## 4 R2-D2       96    32 <NA>      white, bl~ red       33 none  mascu~
## 5 R5-D4       97    32 <NA>      white, red red       NA none  mascu~
## 6 Sebulba    112    40 none      grey, red  orange     NA male  mascu~
## 7 Padmé A~   185    45 brown     light      brown     46 fema~ femin~
## 8 Dud Bolt    94    45 none      blue, grey yellow     NA male  mascu~
## 9 Wat Tam~   193    48 none      green, gr~ unknown    NA male  mascu~
## 10 Sly Moo~  178    48 none      pale       white     NA <NA>  <NA>
## # i 77 more rows
## # i 5 more variables: homeworld <chr>, species <chr>, films <list>,
## #   vehicles <list>, starships <list>
```

2. Selecione apenas as colunas `name` e `birth_year` e então ordene de forma decrescente pelo `birth_year`.

```
# Aninhando funções
arrange(select(sw, name, birth_year), desc(birth_year))

# Criando um objeto intermediário
sw_aux <- select(sw, name, birth_year)
arrange(sw_aux, desc(birth_year))
```

```
# Usando pipe
sw %>%
  select(name, birth_year) %>%
  arrange(desc(birth_year))
```

23.1.3 Filtrando linhas

Utilize a base `sw` nos exercícios a seguir.

1. Crie um objeto chamado `humanos` apenas com personagens que sejam humanos.

```
humanos <- filter(sw, species == "Human")

# O pipe
humanos <- sw %>%
  filter(species == "Human")
```

2. Crie um objeto chamado `altos_fortes` com personagens que tenham mais de 200 cm de altura e peso maior que 100 kg.

```
altos_fortes <- filter(sw, height > 200, mass > 100)
```

3. Retorne tabelas (`tibbles`) apenas com:

- a. Personagens humanos que nasceram antes de 100 anos antes da batalha de Yavin (`birth_year < 100`).

```
filter(sw, species == "Human", birth_year < 100)
```

- b. Personagens com cor `light` ou `red`.

```
filter(sw, skin_color == "light" | skin_color == "red")
```

- c. Personagens com massa maior que 100 kg, ordenados de forma decrescente por altura, mostrando apenas as colunas `name`, `mass` e `height`.

```
select(arrange(filter(sw, mass > 100), desc(height)), name, mass, height)

# usando o pipe
sw %>%
  filter(mass > 100) %>%
  arrange(desc(height)) %>%
  select(name, mass, height)
```

d. Personagens que sejam “Humano” ou “Droid”, e tenham uma altura maior que 170 cm.

```
filter(sw, species == "Human" | species == "Droid", height > 170)

# usando o pipe
sw %>%
  filter(species %in% c("Human", "Droid"), height > 170)
```

e. Personagens que não possuem informação tanto de altura quanto de massa, ou seja, possuem NA em ambas as colunas.

```
filter(sw, is.na(height), is.na(mass))
```

23.1.4 Modificando e criando novas colunas

1. Crie uma coluna chamada `dif_peso_altura` (diferença entre altura e peso) e salve a nova tabela em um objeto chamado `sw_dif`. Em seguida, filtre apenas os personagens que têm altura maior que o peso e ordene a tabela por ordem crescente de `dif_peso_altura`.

```
sw_dif <- mutate(sw, dif_peso_altura = height - mass)

arrange(filter(sw_dif, height > mass), dif_peso_altura)

# usando o pipe
sw_dif <- sw %>%
  mutate(dif_peso_altura = height - mass)

sw_dif %>%
  filter(height > mass) %>%
  arrange(dif_peso_altura)
```

2. Fazendo apenas uma chamada da função `mutate()`, crie as seguintes colunas novas na base `sw`:

- a. `indice_massa_altura = mass / height`
- b. `indice_massa_medio = mean(mass, na.rm = TRUE)`
- c. `indice_relativo = (indice_massa_altura - indice_massa_medio) / indice_massa_medio`
- d. `acima_media = ifelse(indice_massa_altura > indice_massa_medio, "sim", "não")`

```
mutate(sw,
  indice_massa_altura = mass/height,
  indice_massa_medio = mean(mass, na.rm = TRUE),
  indice_relativo = (indice_massa_altura - indice_massa_medio) / indice_massa_medio,
  acima_media = ifelse(indice_massa_altura > indice_massa_medio, "sim", "não"))
```

23.1.5 Sumarizando a base

Utilize a base `sw` nos exercícios a seguir.

1. Calcule a altura média e mediana dos personagens.

```
summarize(sw,
  media_altura = mean(height, na.rm=TRUE),
  mediana_altura = median(height, na.rm = TRUE))
```

```
## # A tibble: 1 x 2
##   media_altura mediana_altura
##         <dbl>         <int>
## 1         175.           180
```

2. Calcule a massa média dos personagens cuja altura é maior que 175 cm.

```
sw %>%
  filter(height > 175) %>%
  summarize(media_massa = mean(mass, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   media_massa
##         <dbl>
## 1         87.2
```

3. Apresente na mesma tabela a massa média dos personagens com altura menor que 175 cm e a massa média dos personagens com altura maior ou igual a 175 cm.

```
sw %>%
  mutate(alturas = ifelse(height < 175, "menor 175", "maior 175")) %>%
  filter(!is.na(height)) %>%
  group_by(alturas) %>%
  summarize(altura_media = mean(height, na.rm=TRUE))
)
```

```
## # A tibble: 2 x 2
##   alturas  altura_media
##   <chr>      <dbl>
## 1 maior 175      193.
## 2 menor 175      142.
```

4. Retorne tabelas (tibbles) apenas com:

a. A altura média dos personagens por espécie.

```
sw %>%
  group_by(species) %>%
  summarize(altura_media = mean(height, na.rm = TRUE))
```

```
## # A tibble: 38 x 2
##   species  altura_media
##   <chr>      <dbl>
## 1 Aleena      79
## 2 Besalisk    198
## 3 Cerean     198
## 4 Chagrian    196
## 5 Clawdite    168
## 6 Droid      131.
## 7 Dug        112
## 8 Ewok         88
## 9 Geonosian   183
## 10 Gungan     209.
## # i 28 more rows
```

b. A massa média e mediana dos personagens por espécie.

```
sw %>%
  filter(!is.na(mass)) %>%
  group_by(species) %>%
  summarize(massa_media = mean(mass, na.rm = TRUE),
            massa_mediana = median(mass, na.rm = TRUE))
```

```
## # A tibble: 32 x 3
##   species  massa_media massa_mediana
##   <chr>      <dbl>      <dbl>
## 1 Aleena      15         15
## 2 Besalisk   102        102
## 3 Cerean      82         82
## 4 Clawdite    55         55
```

```
## 5 Droid          69.8      53.5
## 6 Dug            40       40
## 7 Ewok           20       20
## 8 Geonosian      80       80
## 9 Gungan        74       74
## 10 Human         81.3     79
## # i 22 more rows
```

c. Apenas o nome dos personagens que participaram de mais de 2 filmes.

```
sw %>%
  filter(length(films) > 2) %>%
  select(name)
```

```
## # A tibble: 87 x 1
##   name
##   <chr>
## 1 Luke Skywalker
## 2 C-3P0
## 3 R2-D2
## 4 Darth Vader
## 5 Leia Organa
## 6 Owen Lars
## 7 Beru Whitesun Lars
## 8 R5-D4
## 9 Biggs Darklighter
## 10 Obi-Wan Kenobi
## # i 77 more rows
```