Introdução ao R

Renato de Paula

2024-08-15

Contents

Chapter 1

Prefácio

Material utilizado no curso **Laboratório de Estatística I** da Faculdade de Ciências da Universidade de Lisboa.

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**: roda 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

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.

3.3 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.

3.3.1 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 =.

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

```r
# 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,

```r
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:

```r
ls()
```

No RStudio a aba *Environment* mostra os valores.

3.4 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 caracter 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 `fcul` e `Fcul` 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
```

3.5 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)
```

3.6 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
```

3.7 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 respetivos argumentos.

3.8 Quantidade de digitos

```
exp(1)
```

```
## [1] 2.718282
```

```
options(digits = 20)

exp(1)
```

```
## [1] 2.7182818284590450908
```

```
options(digits = 3)

exp(1)
```

```
## [1] 2.72
```

3.9 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 esses conjuntos de dados. Esses conjuntos de dados são objetos que podem ser usados simplesmente digitando o nome. Por exemplo, se você digitar:

```
co2
```

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
```

```
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)

is.na(x)
```

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

```r
is.nan(x)
```

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

```r
is.na(y)
```

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

```r
is.nan(y)
```

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

```r
# Operações com NaN e NA
sum(x)  # Exibe: NaN, porque a soma envolve um NaN
```

```
## [1] NaN
```

```r
sum(y)  # Exibe: NA, porque a soma envolve um NA
```

```
## [1] NA
```

```r
sum(x, na.rm = TRUE)  # Exibe: 12, ignora NaN na soma
```

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

```r
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.

3.10 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"
```

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

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

```r
typeof(z)
```

```
## [1] "list"
```

```r
w <- list(a = 1, b = "text")
class(w)
```

```
## [1] "list"
```

```r
typeof(w)
```

```
## [1] "list"
```

3.11 Conversão entre tipos de dados

```r
# Convertendo inteiro em string
a <- 15
b <- as.character(15)
print(b)
```

```
## [1] "15"
```

```r
# Convertendo float em inteiro
x <- 1.5
y <- as.integer(x)
print(y)
```

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

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

```
## [1] 10
```

3.12 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.

Ex1:

```
nome1 <- "faculdade"
nome2 <- "ciências"
print(paste(nome1, nome2))
```

```
## [1] "faculdade ciências"
```

Ex2:

```
# 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)
```

Ex3:

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

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

Ex4:

```
# 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"
```

3.13 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`).

3.13.1 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
```

3.13.2 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
```

3.14 Exercícios

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 inteiro
c(1L, 2L, 3L)
```

```
## [1] 1 2 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

```r
# Inteiros de 1 a 10
x <- 1:10
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
# 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
```

```r
# Sequência de 11 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
```

```r
# 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
```

```r
t <- rep(1:3, each=4)
t
```

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

```r
# 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

```r
# Defina um vetor com inteiros de (-5) a 5 e extraia os números com valor absoluto men
x<- (-5):5
x
```

```
##  [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

```r
# pelo seu índice no vetor:
x[4:8]
```

```
## [1] -2 -1  0  1  2
```

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

```
## [1] -2 -1  0  1  2
```

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

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

```r
# Agora este vetor pode ser usado para extrair os números desejados:
x[index]
```

```
## [1] -2 -1  0  1  2
```

```r
# Que é a mesma coisa que...
x[abs(x) < 3]
```

```
## [1] -2 -1  0  1  2
```

```r
letters[1:3]
```

```
## [1] "a" "b" "c"
```

```r
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 a soma dos elementos de um vetor
sum(num_vector)
```

```
## [1] 6.6
```

```r
# Calcula a média dos elementos de um vetor
mean(num_vector)
```

```
## [1] 2.2
```

```r
# Ordena os elementos de um vetor
sort(num_vector)
```

```
## [1] 1.1 2.2 3.3
```

```r
sort(num_vector,decreasing = TRUE)
```

```
## [1] 3.3 2.2 1.1
```

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

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

4.1.5 Operações com Vetores

```r
# Adição
num_vector + 1
```

```
## [1] 3.2 2.1 4.3
```

```r
# Multiplicação
num_vector * 2
```

```
## [1] 4.4 2.2 6.6
```

```r
# Comparações
num_vector > 2
```

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

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

```
## [1]  4  9 25 49
```

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

```
## [1]   4  27  25 343
```

```
c(1,2,3,4,5,6)^c(2,3,4)
```

```
## [1]    1    8   81   16  125 1296
```

```
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
```

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

```r
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

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 (unordered factors).

```r
# 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
```

```r
# 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
```

```r
# 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
```

```r
# Verificar Níveis
levels(factor_data)
```

```
## [1] "low"    "medium" "high"
```

```r
# Modificar Níveis
levels(factor_data) <- c("Low", "Medium", "High")
print(factor_data)
```

```
## [1] Low    Medium High   Medium Low    High
## Levels: Low Medium High
```

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.

```r
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
```

```r
matrix(c(1,2,3,4,5,6)+exp(1),nrow=2) > 6
```

```
##       [,1]  [,2] [,3]
## [1,] FALSE FALSE TRUE
## [2,] FALSE  TRUE TRUE
```

```r
# 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

```r
# 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
```

```r
# Combinar os vetores por colunas
result <- cbind(vector1, vector2)
print(result)
```

```
##      vector1 vector2
## [1,]       1       4
## [2,]       2       5
## [3,]       3       6
```

```r
# 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
```

```r
# 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

```r
# 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,]  0.216  0.4300 -0.906
## [2,]  0.645 -0.0482 -0.547
## [3,] -1.413 -0.8728 -0.441
## [4,] -0.463 -0.4017 -0.195
```

```
colnames(x) <- paste("dados",1:3,sep="")
x
```

```
##      dados1  dados2 dados3
## [1,]  0.216  0.4300 -0.906
## [2,]  0.645 -0.0482 -0.547
## [3,] -1.413 -0.8728 -0.441
## [4,] -0.463 -0.4017 -0.195
```

```
y <- matrix(rnorm(15),nrow=5)
y
```

```
##         [,1]   [,2]    [,3]
## [1,]  0.4446  0.119  0.3338
## [2,]  1.5957  0.386 -0.1811
## [3,] -0.0182 -0.734 -1.1146
## [4,] -0.1067  0.416 -0.0533
## [5,]  0.3021 -0.117 -0.7563
```

```
colnames(y) <- LETTERS[1:ncol(y)]

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

y
```

```
##         A      B       C
## a  0.4446  0.119  0.3338
## b  1.5957  0.386 -0.1811
## c -0.0182 -0.734 -1.1146
## d -0.1067  0.416 -0.0533
## e  0.3021 -0.117 -0.7563
```

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,]  0.311 -0.0938 -1.49
## [2,]  2.836 -0.8051  2.13
## [3,]  0.873  4.1783  2.54
## [4,] -4.758  3.6254  1.40
```

4.3.5 Algumas outras funções

Seja M uma matriz quadrada.

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

4.3.6 Exercícios

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
```

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

```
## [1] "Ana"
```

```r
# 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

```r
# 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
```

```r
# 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"
```

```r
dim(df)
```

```
## [1] 150   5
```

```r
nrow(df)
```

```
## [1] 150
```

```r
ncol(df)
```

```
## [1] 5
```

```r
# 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 ...
```

```r
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
```

```r
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

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.

- **Heterogeneidade**: Uma lista pode conter elementos de diferentes tipos. Por exemplo, você pode ter um vetor numérico, um vetor de caracteres, uma matriz e um data frame, todos na mesma lista.
- **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.
- **Flexibilidade**: As listas podem ser usadas para armazenar saídas complexas de funções ou para estruturar dados que requerem uma organização mais flexível.

```r
# 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
```

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, `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. Veremos mais sobre isso após falarmos sobre 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(os) 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(os) 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

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 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

```r
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:

```r
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?

```r
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

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()`.

```r
# 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:

(a) 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.

(b) Um script `estatistica.R` que disponibiliza funções que permitem ordenar uma amostra e calcular a média, calcular a variância e o desvio padrão.

(c) 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 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.

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

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.

Exemplo:

8.2 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.

```r
# 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()`.

8.3 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)
```

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

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

```
library(readxl)
# Leitura de arquivo Excel
dados_excel <- read_excel("dados.xlsx", sheet = 1)
```

8.5 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

9.2 Exercícios

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

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)
}
```

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.

```r
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:"
```

```r
print(soma_linhas)
```

```
## [1] 12 15 18
```

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

```r
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:"
```

```r
print(medias)
```

```
## [1] 2 5 8
```

11.1 Exercícios

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**
`apply`	`apply(x, MARGIN, FUN)`	Aplica uma função às linhas ou colunas ou a ambas	Data frame ou matriz	vetor, lista, array
`lapply`	`lapply(x, FUN)`	Aplica uma função a todos os elementos da entrada	Lista, vetor ou data frame	lista
`sapply`	`sapply(x, FUN)`	Aplica uma função a todos os elementos da entrada	Lista, vetor ou data frame	vetor ou matriz
`tapply`	`tapply(x, INDEX, FUN)`	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 raíz 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.

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

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

Exemplo 1:

```r
nomes <- c("ANA", "JOAO", "PAULO", "FILIPA")
(nomes_minusc <- lapply(nomes, tolower))
```

```
## [[1]]
## [1] "ana"
##
## [[2]]
## [1] "joao"
##
## [[3]]
## [1] "paulo"
##
## [[4]]
## [1] "filipa"
```

```r
str(nomes_minusc)
```

```
## List of 4
##  $ : chr "ana"
##  $ : chr "joao"
##  $ : chr "paulo"
##  $ : chr "filipa"
```

Exemplo 2:

```r
# 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

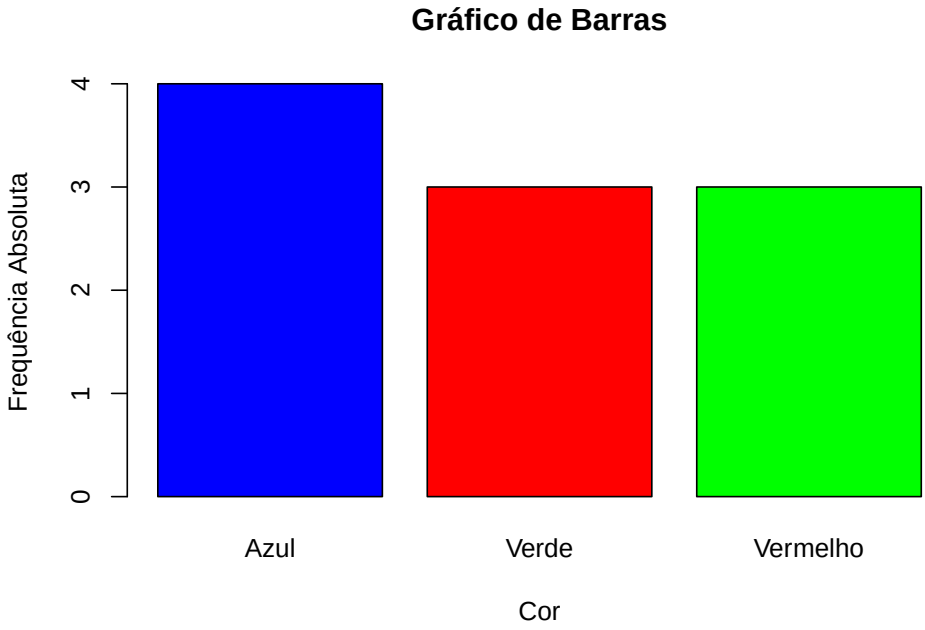
Chapter 13

Gráficos (R base)

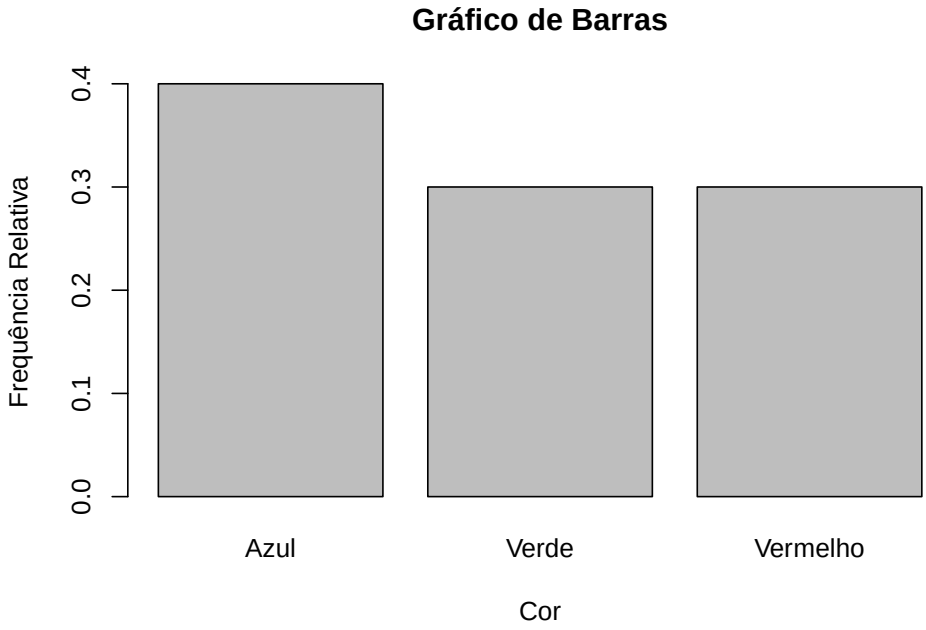
13.1 Gráfico de Barras

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

```r
# 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"))
```



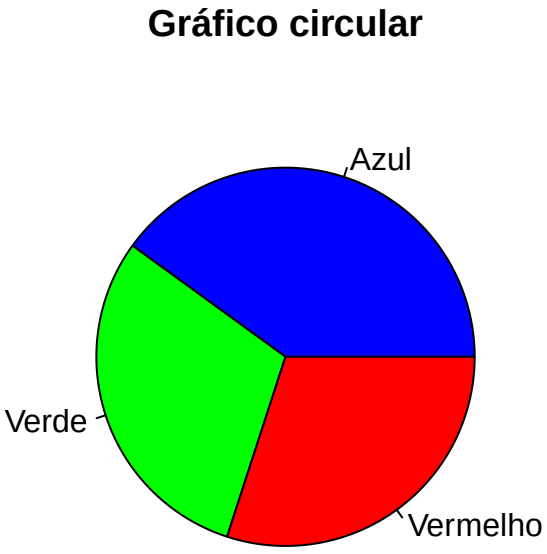

```r
# Calcular as frequências relativas
frequencia_relativa <- frequencia_absoluta / length(cores)

# 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 percentagens 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"))
```



13.3 Histograma

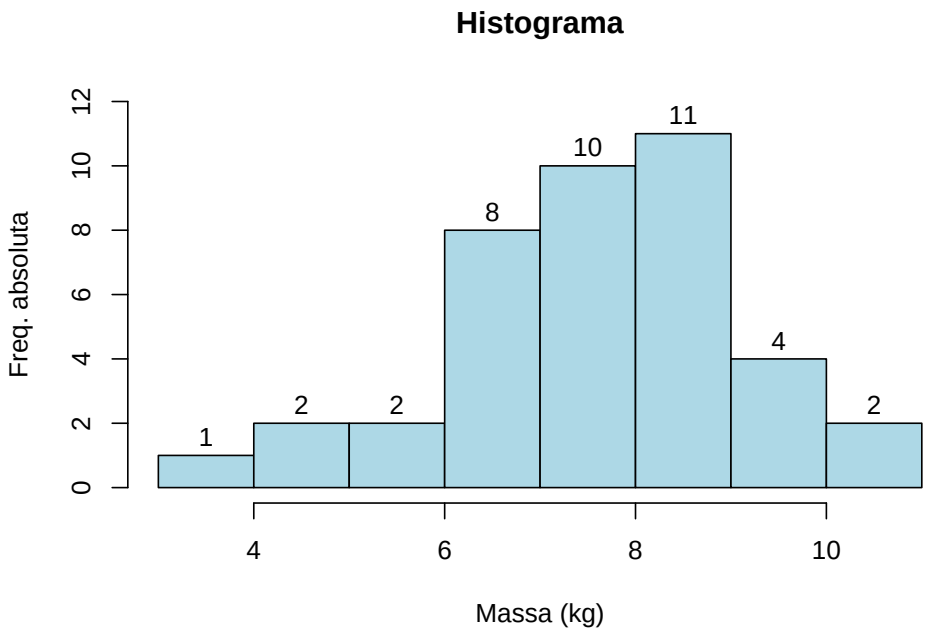
Histograma é uma representação gráfica dos dados em que se marcam as classes (intervalos) no eixo horizontal e as frequências (absuluta ou relativa) no eixo vertical.

- 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")
```



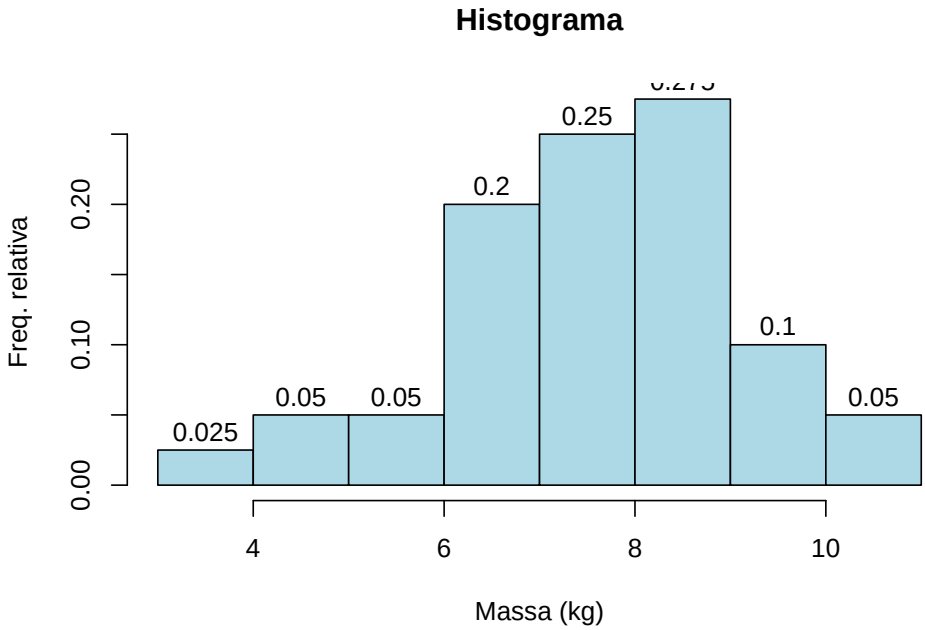

```
# Pontos limites das classes
h$breaks
```

```
## [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
```

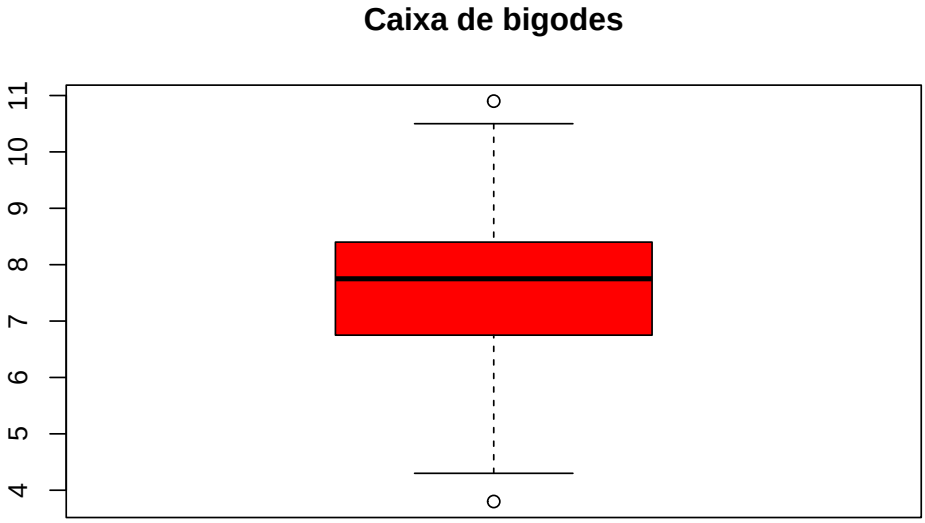
```
## [1]  1  2  2  8 10 11  4  2
```

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

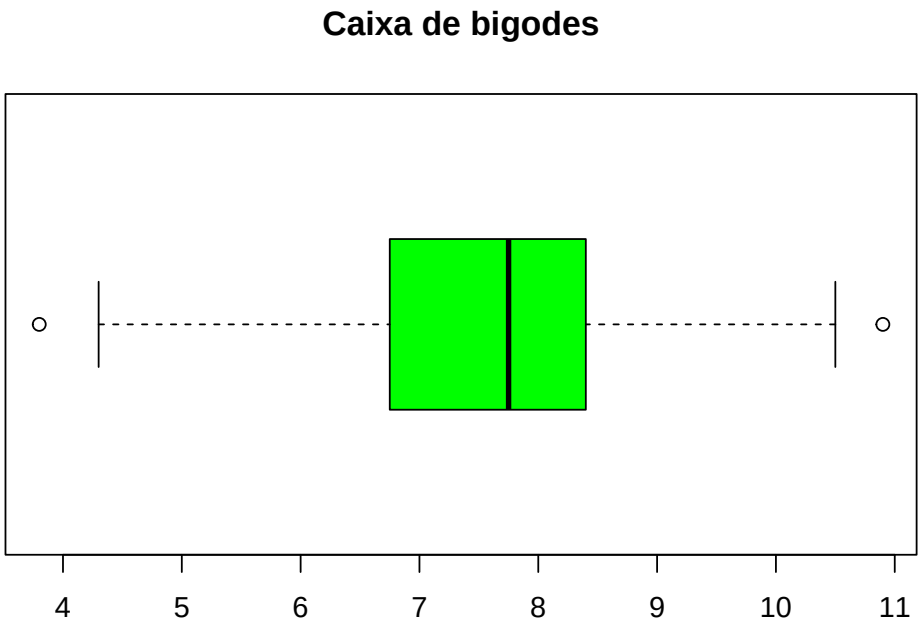


13.4 Box-plot

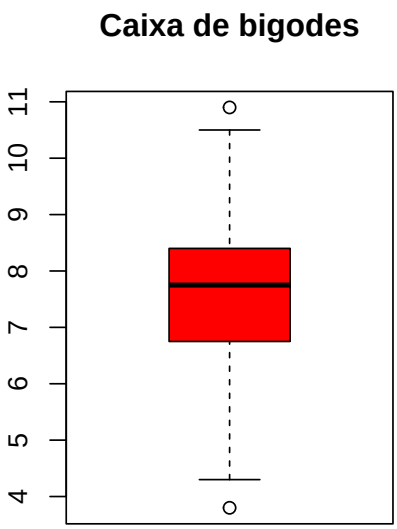
```
# Caixa de bigodes vertical
boxplot(bicicletas, main = "Caixa de bigodes", col="red")
```

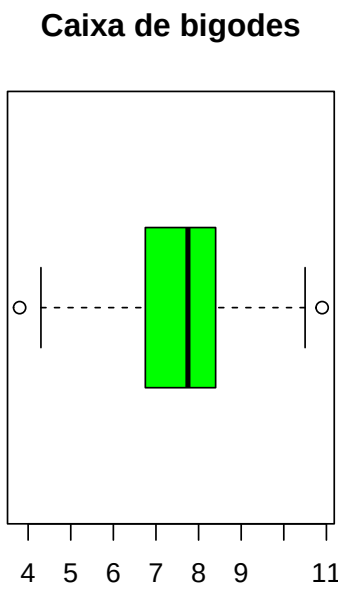


```
# Caixa de bigodes horizontal
boxplot(bicicletas, main = "Caixa de bigodes", col="green", horizontal = TRUE)
```




```
# Caixa de bigodes 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)
```






```
dev.off()
```

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

Chapter 14

Manipulando dados

Chapter 15

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.

15.1 Exercícios

Chapter 16

Simulação

16.1 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.

16.2 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.

```r
# 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]  6  5 10  4  3
```

Exemplo 2: Amostragem com Reposição.

```r
# Amostra com reposição
amostra_repos <- sample(pop, size = 5, replace = TRUE)
print(amostra_repos)
```

```
## [1] 8 5 1 8 6
```

Exemplo 3: Amostragem com Probabilidades Diferentes.

```r
# 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] 9 6 5 8 7
```

16.3 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 17

Distribuições univariadas no R

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

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

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

17.1 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

$$F_n(x) = \begin{cases} 0, & x < x_{(1)} \\ \frac{i}{n}, & x_{(i)} \leq x < x_{(i+1)}, \quad i = 1, \dots, n-1 \\ 1, & x \geq x_{(n)} \end{cases}$$

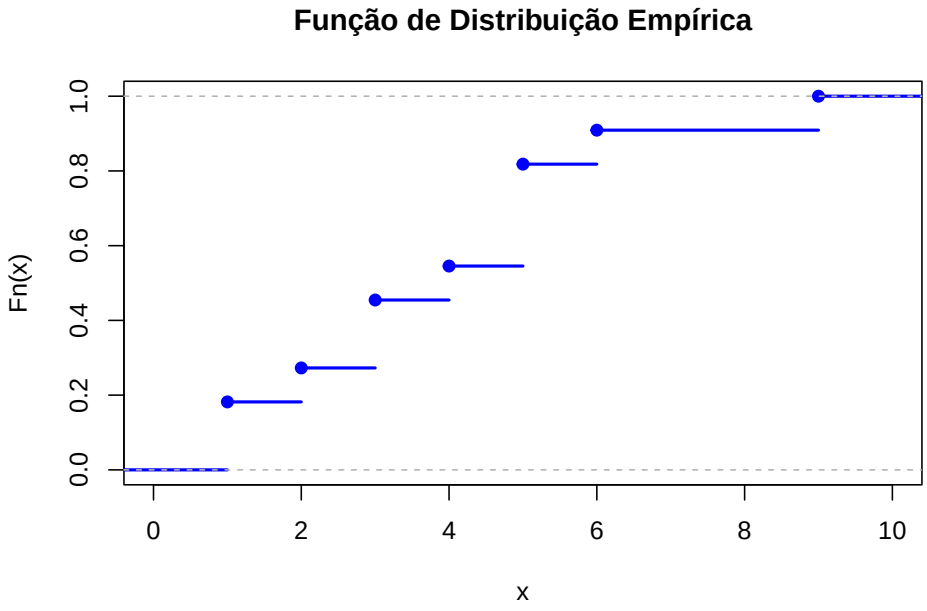
- Passo a passo para a construção da função

  - 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 $\frac{1}{n}$.

17.1.1 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 = "
```



17.2 Gerando uma variável aleatória com distribuição binomial

17.2.1 Cálculo de probabilidades

Seja $X \sim \text{Binomial}(n = 20, p = 0.1)$.

$P(X = 4) \rightarrow$ `dbinom(4,20,0.1)` $= 0.08978$

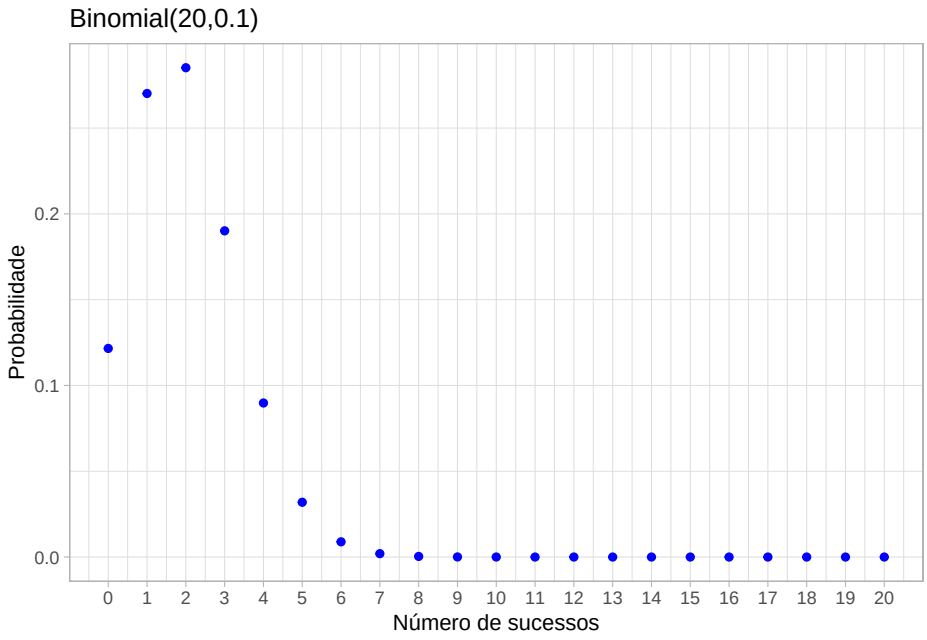
$P(X \leq 4) \rightarrow$ `pbinom(4,20,0.1)` $= 0.9568$

$P(X > 4) \rightarrow$ `pbinom(4,20,0.1,lower.tail=FALSE)` $= 0.04317$

17.2.2 Função massa de probabilidade (teórica)

```r
# 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))
```

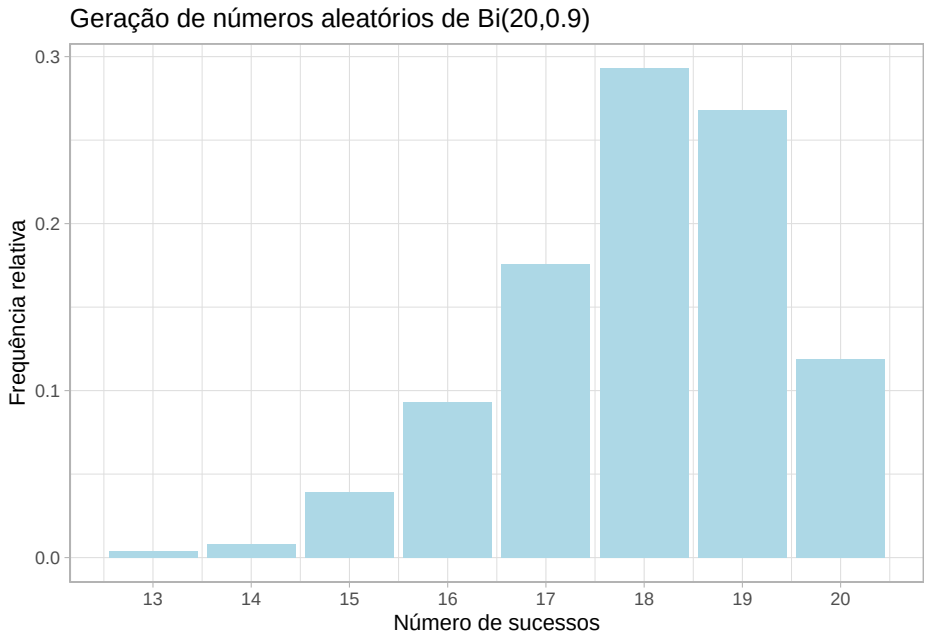
```r
# Carregue o pacote ggplot2
library(ggplot2)

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()
```



17.2.3 Função massa de probabilidade (simulação)

```r
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) +
```

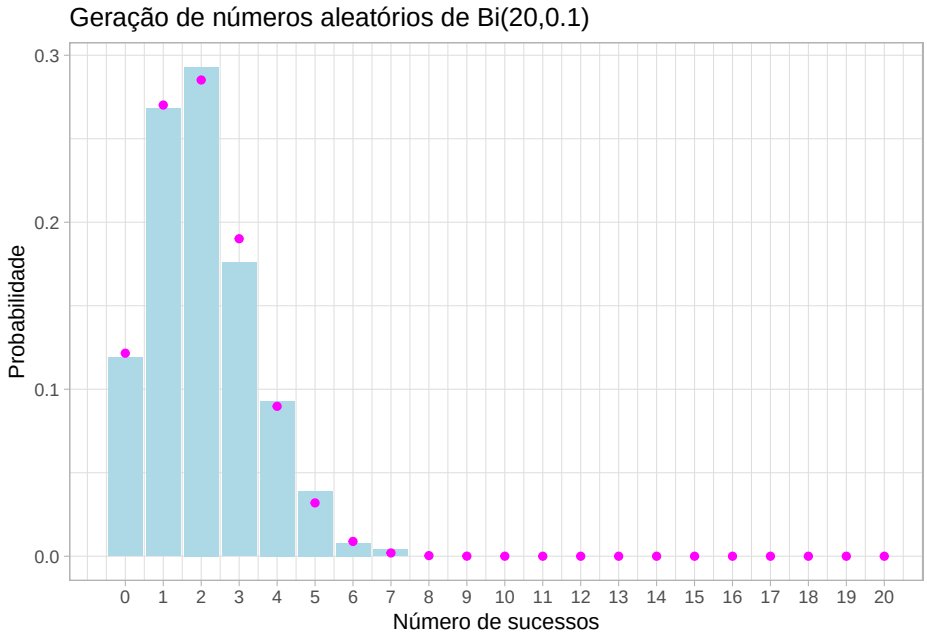
```
geom_bar(aes(x=X, y=after_stat(prop)), fill = "lightblue") +
  scale_x_continuous(breaks = 0:n) +
  labs(title = "Geração de números aleatórios de Bi(20,0.9)", x="Número de sucessos",
  y="Frequência relativa") +
  theme_light()
```



17.2.4 Comparação

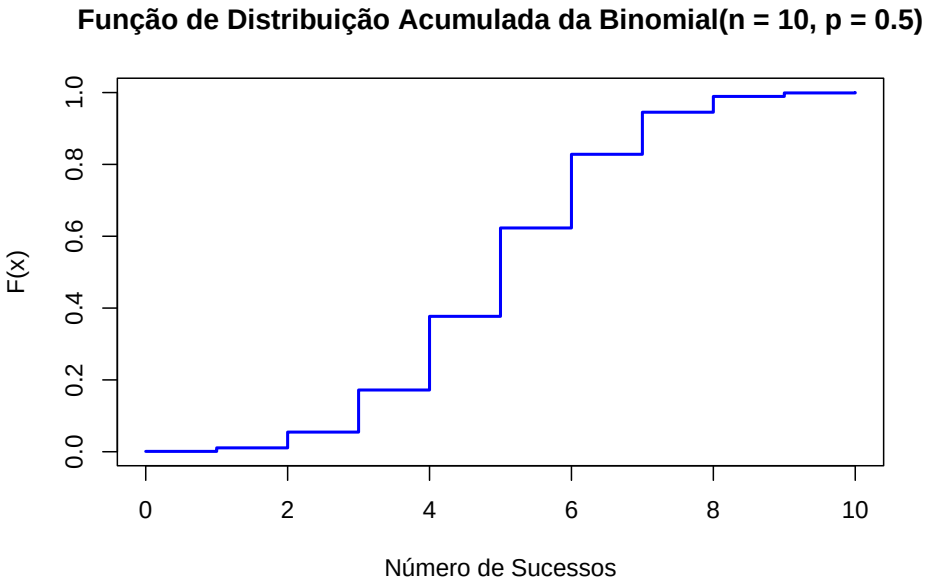
```
set.seed(1234)

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()
```



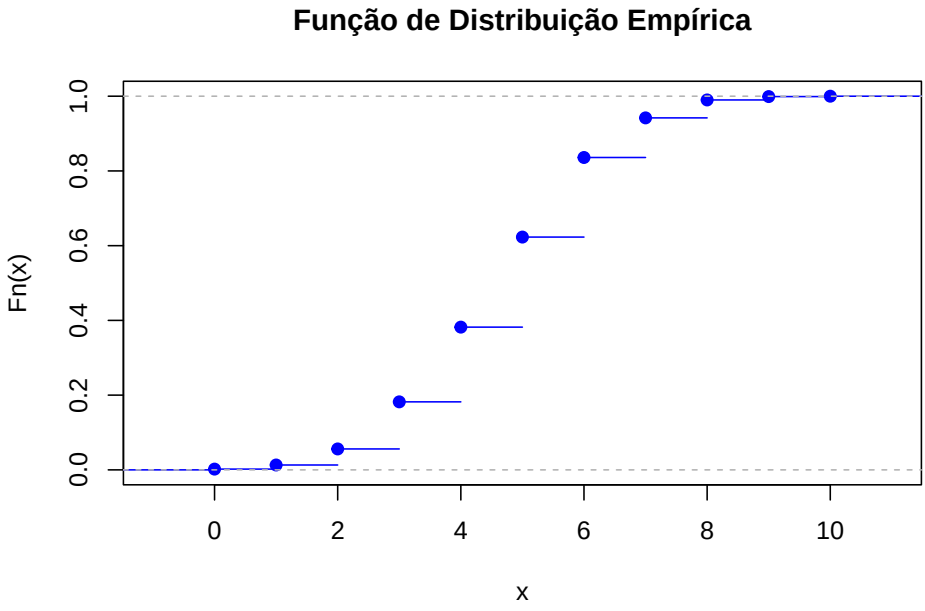
17.2.5 Função de distribuição

```
# 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)")
```

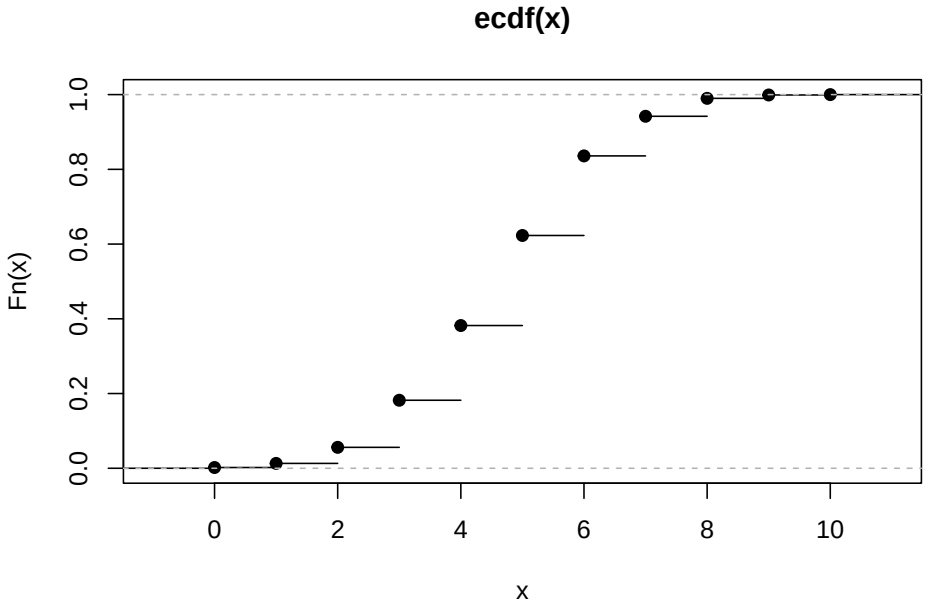


17.2.6 Função de distribuição empírica

```r
# 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")
```




```
# OU
plot.ecdf(amostra)
```



Cálculo de probabilidade: Seja $X \sim \text{Binomial}(n = 10, p = 0.5)$.

$P(X \leq 4) =$ `pbinom(4,10,0.5)` $= 0.377$

$P(X \leq 4) \approx$ `Fn(4)` $= 0.382$

17.3 Gerando uma variável aleatória com distribuição de Poisson

17.3.1 Cálculo de probabilidades

Seja $X \sim \text{Poisson}(\lambda = 5)$.

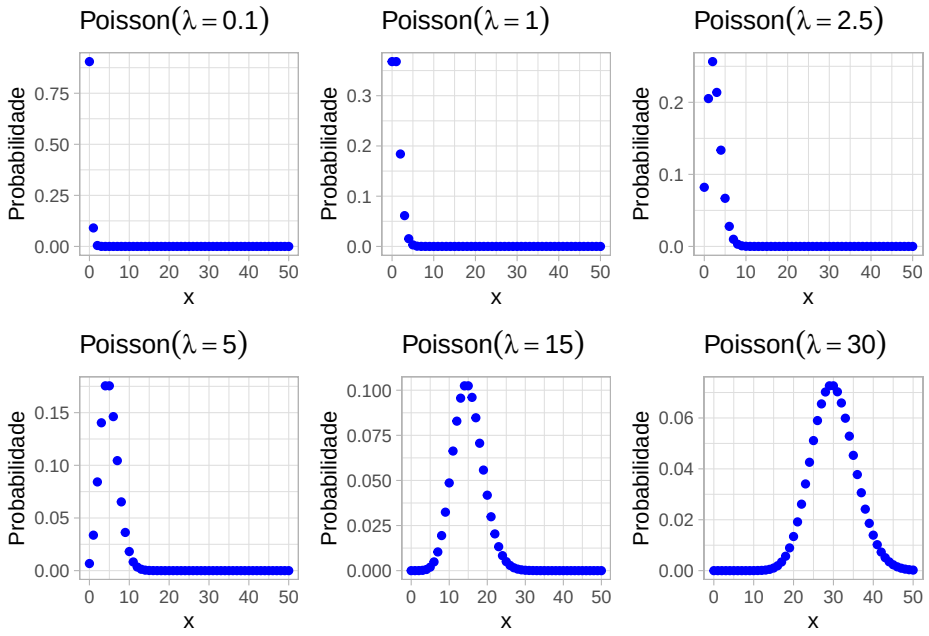
$P(X = 4) \rightarrow$ `dpois(4,5)` $= 0.1755$

$P(X \leq 4) \rightarrow$ `ppois(4,5)` $= 0.4405$

$P(X > 4) \rightarrow$ `ppois(4,5,lower.tail=FALSE)`$= 0.5595$

17.3.2 Função massa de probabilidade (teórica)

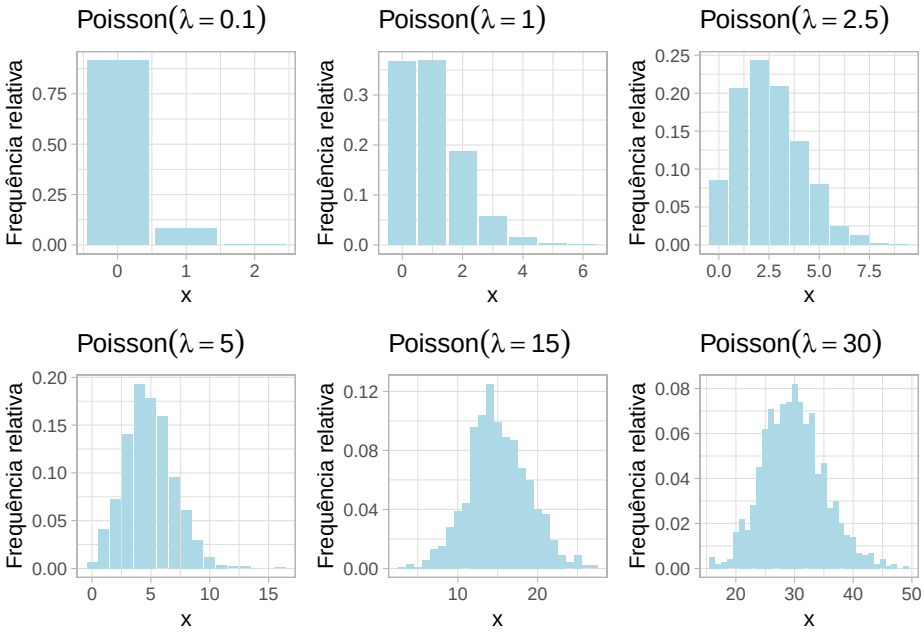
```r
# Definir os valores de lambda e x
p <- c(0.1, 1, 2.5, 5, 15, 30)
x <- 0:50

# Carregar os pacotes necessários
library(ggplot2)
library(latex2exp)
library(gridExtra)

# 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)
```



17.3.3 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()
}
```

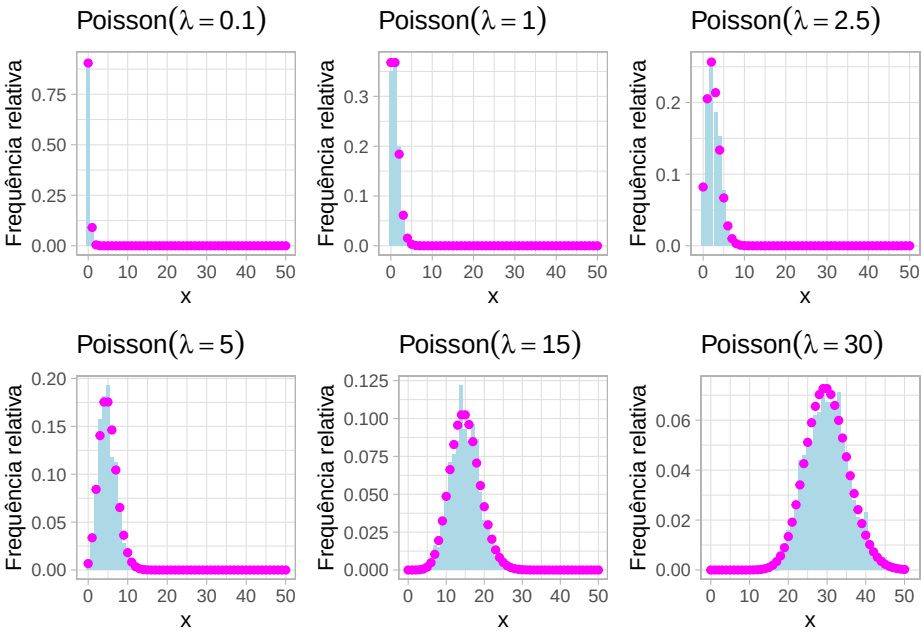
```
# Dispor os gráficos em uma grade 2x3
grid.arrange(grobs = plots, nrow = 2, ncol = 3)
```



17.3.4 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) +
```

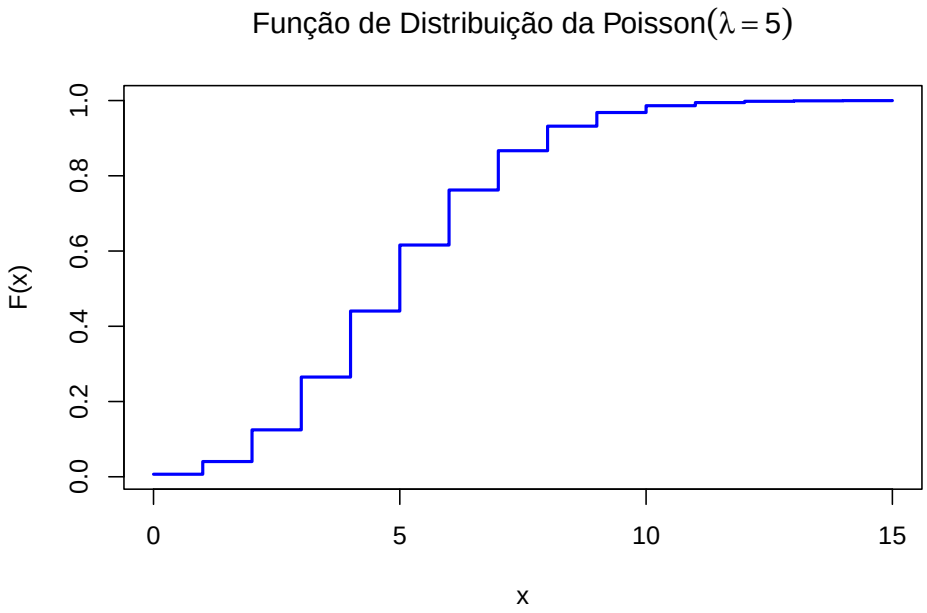
```
    geom_bar(aes(x = X, y =after_stat(prop)), fill="lightblue") +
    geom_point(data = teorico, aes(x, y), color = "magenta") +
    scale_x_continuous(breaks = seq(0, 50, by = 10)) +
    labs(title=TeX(paste("$Poisson(lambda=", p[i], ")$")),
    x = "x", y = "Frequência relativa") +
    theme_light()
}

# Dispor os gráficos em uma grade 2x3
grid.arrange(grobs = plots, nrow = 2, ncol = 3)
```



17.3.5 Função de distribuição

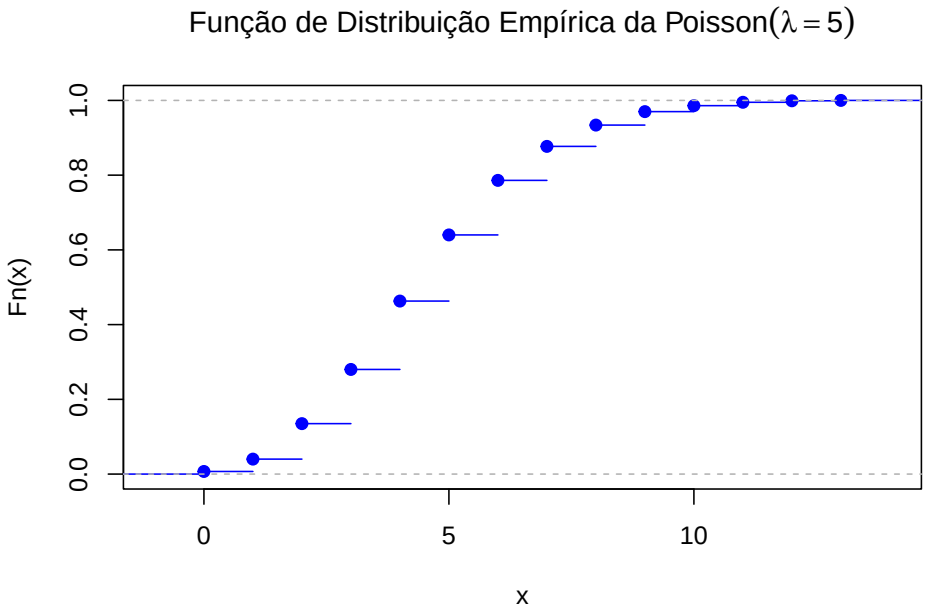
```
lambda <- 5  # Parâmetro da Poisson
x <- 0:15    # Valores de x para plotar a distribuição

# 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)")
```

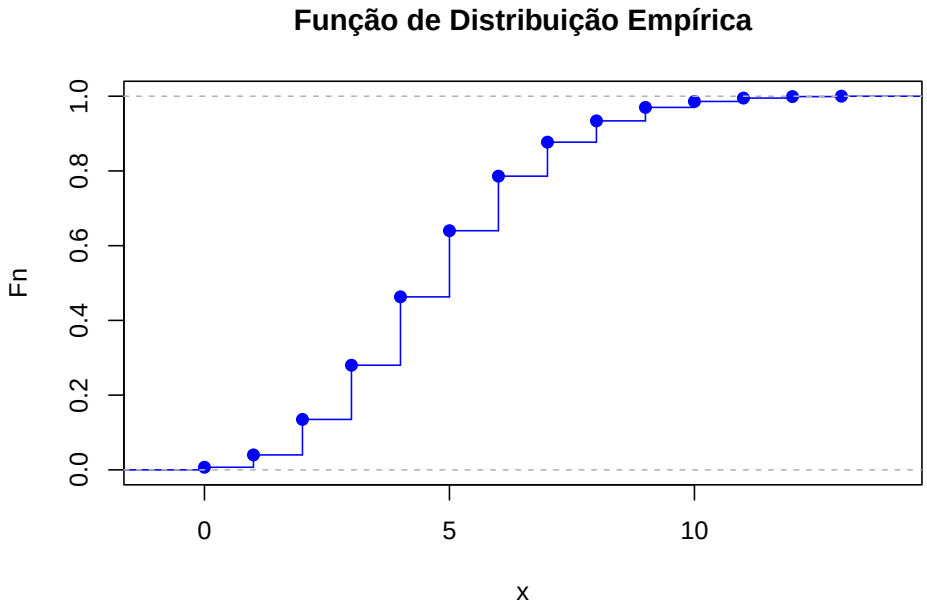


17.3.6 Função de distribuição empírica

```
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")
```




```
# OU
#plot.ecdf(dados)

plot(Fn, main="Função de Distribuição Empírica",
     xlab="x",
     ylab="Fn",
     col="blue",
     verticals = TRUE)
```



Cálculo de probabilidades: Seja $X \sim \text{Poisson}(\lambda = 5)$.

$P(X \leq 4) \rightarrow$ `ppois(4,5)` $= 0.4405$

$P(X \leq 4) \rightarrow$ `Fn(4)` $= 0.433$

17.4 Gerando uma variável aleatória com distribuição de Uniforme

17.4.1 Cálculo de probabilidades

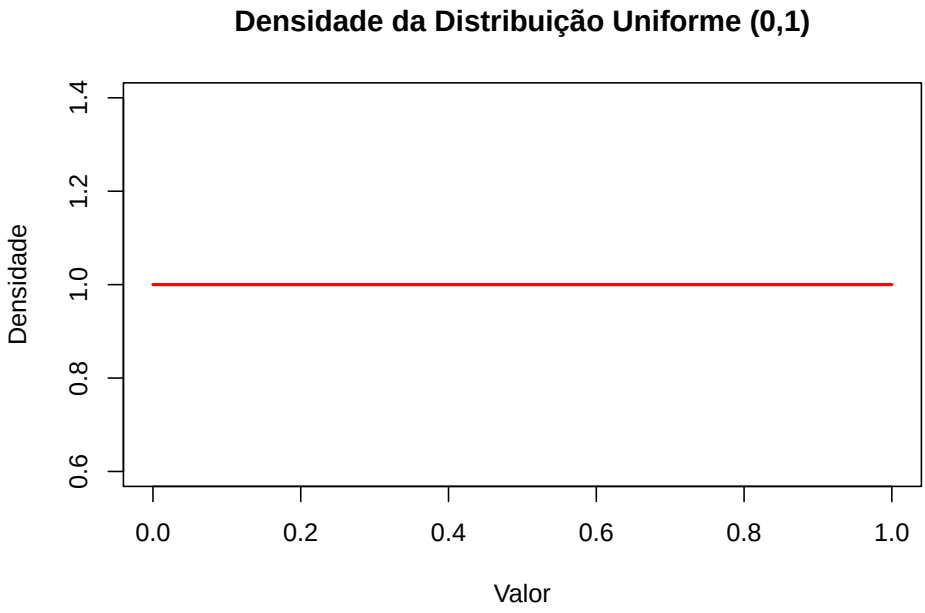
Seja $X \sim \text{Uniforme}(0, 1)$

- $P(X \leq 0.5) \rightarrow$ `punif(0.5, min = 0, max = 1)` $= 0.5$
- $P(X > 0.5) \rightarrow$ `punif(0.5, min = 0, max = 1, lower.tail = FALSE)` $= 0.5$

17.4.2 Função densidade de probabilidade

```r
# Gerar os valores x para a densidade teórica
x_vals <- seq(0, 1, length.out = 100)
```

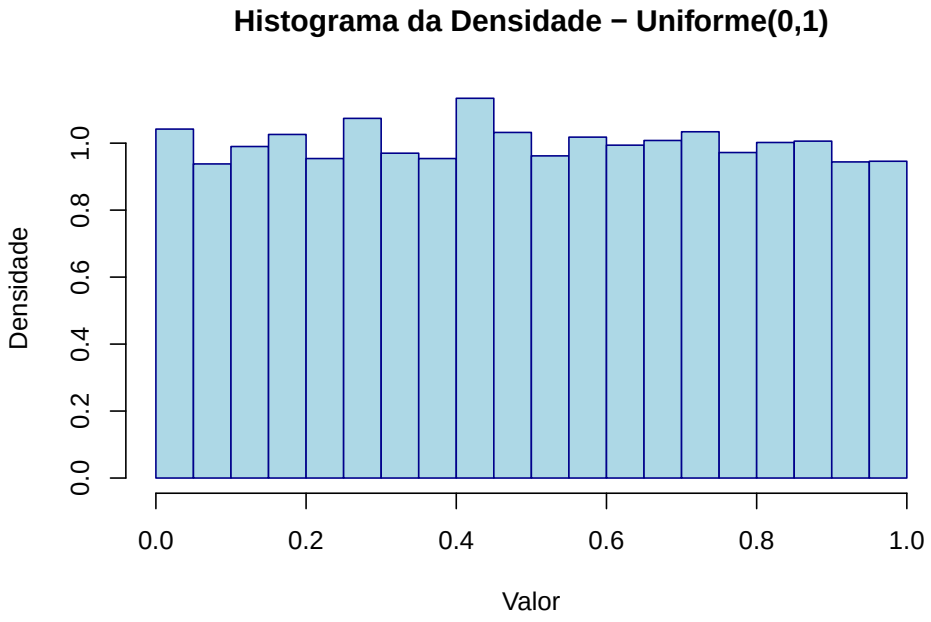
```r
# Calcular a densidade teórica para os valores x
y_vals <- dunif(x_vals, min = 0, max = 1)

# 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")
```



17.4.3 Função densidade de probabilidade (simulação)

```r
# 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)",
```

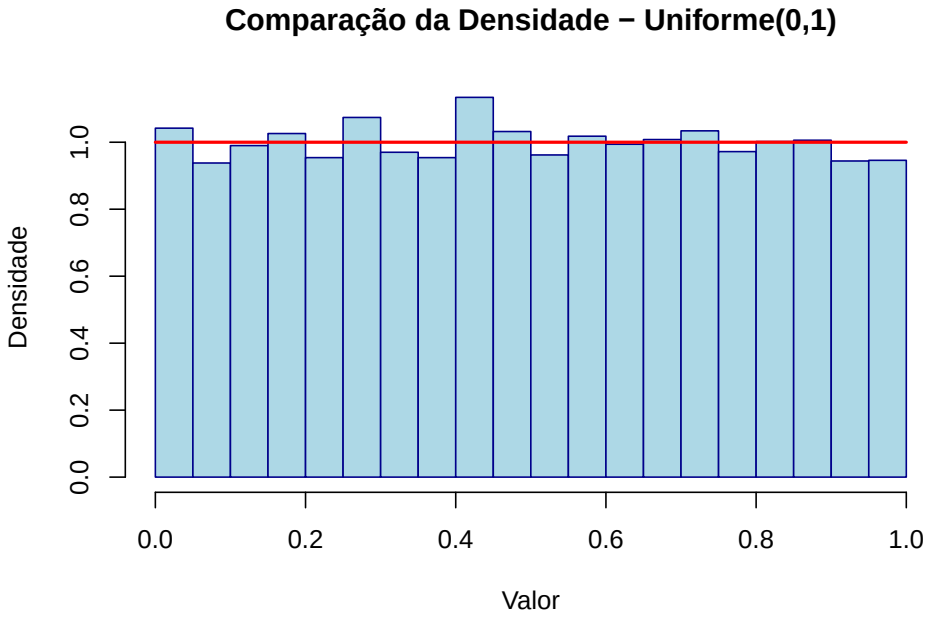
```
     xlab = "Valor",
     ylab = "Densidade",
     col = "lightblue",
     border = "darkblue")
```



17.4.4 Comparação

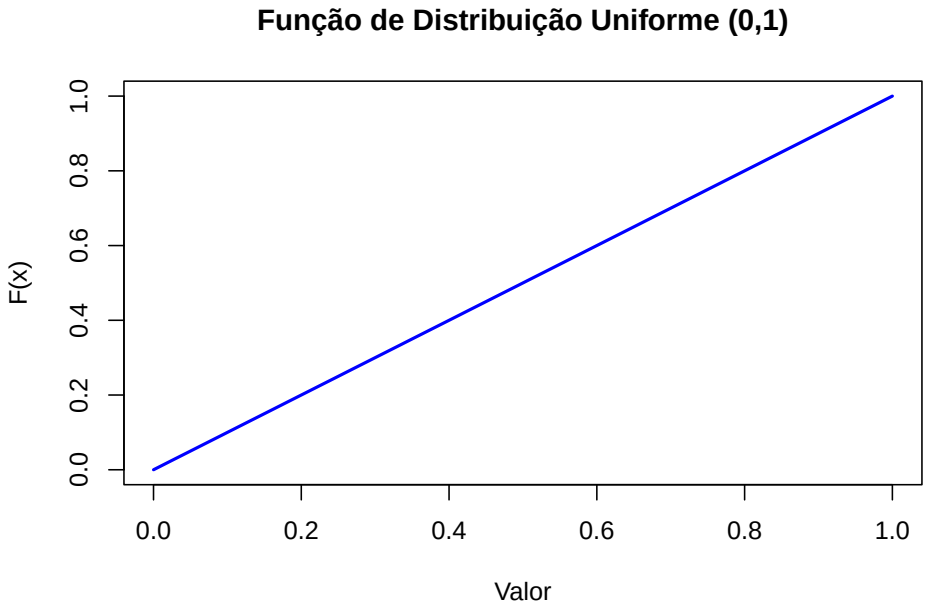
```
# Definir o tamanho da amostra
n <- 10000

# 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")
```

```r
# Adicionar a curva da densidade teórica
curve(dunif(x, min = 0, max = 1),
      add = TRUE,
      col = "red",
      lwd = 2)
```



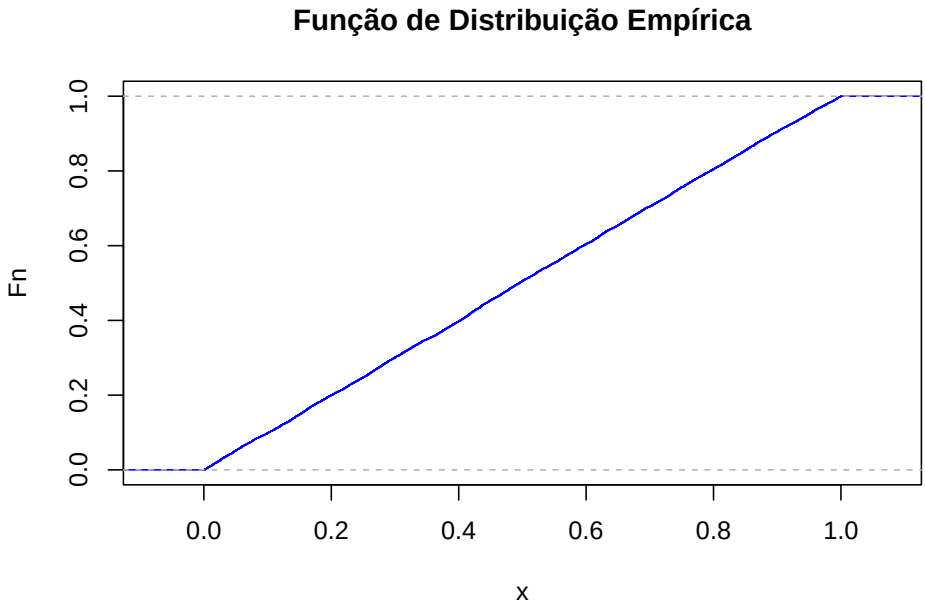
17.4.5 Função de distribuição

```r
# 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)")
```



17.4.6 Função de distribuição empírica

```r
# 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")
```




```
# OU
#plot.ecdf(uniform_data)
```

17.5 Gerando uma variável aleatória com distribuição Exponencial

17.5.1 Cálculo de probabilidades

Seja $X \sim \text{Exponencial}(\lambda = 1)$.

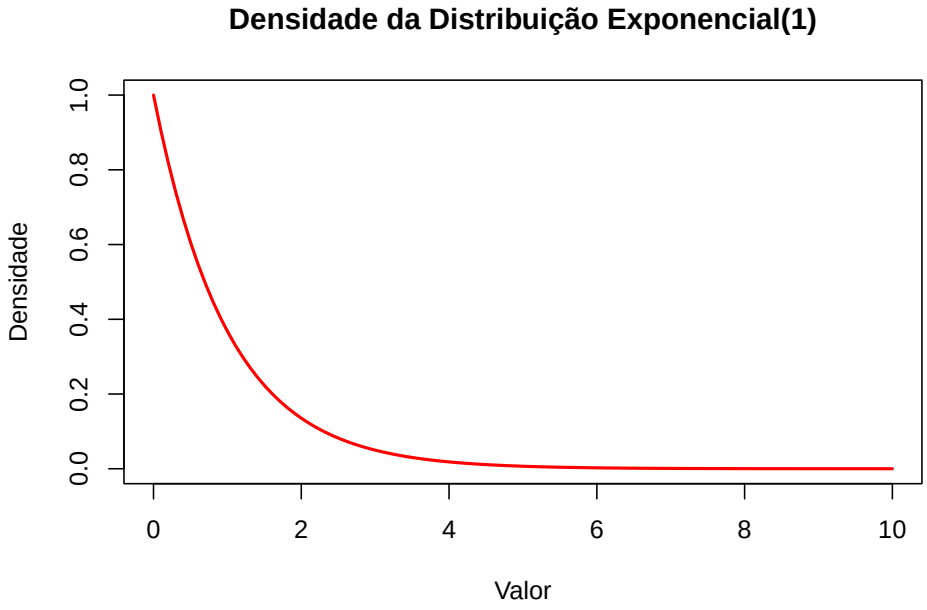
$P(X \leq 0.5) \rightarrow$ `pexp(0.5,rate=1)`$=0.3935$

$P(X > 0.5) \rightarrow$ `pexp(0.5,rate=1,lower.tail=FALSE)`$=0.6065$

17.5.2 Função densidade de probabilidade (teórica)

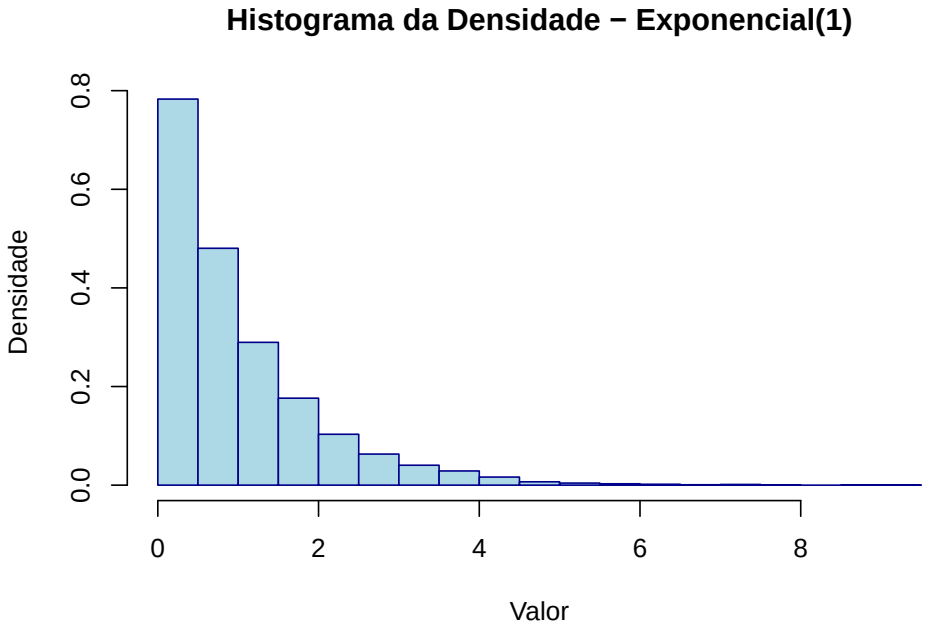
```
# Gerar os valores x para a densidade teórica
x_vals <- seq(0, 10, length.out = 100)

# Calcular a densidade teórica para os valores x
y_vals <- dexp(x_vals, rate=1)
```

```r
# 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")
```



17.5.3 Função densidade de probabilidade (simulação)

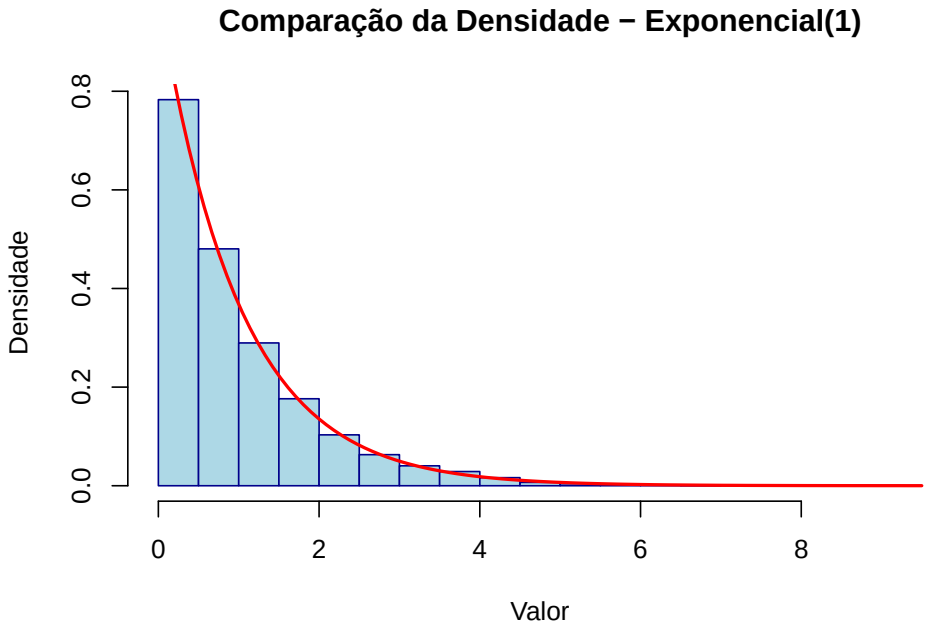
```r
# Definir o tamanho da amostra
n <- 10000

# 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")
```



17.5.4 Comparação

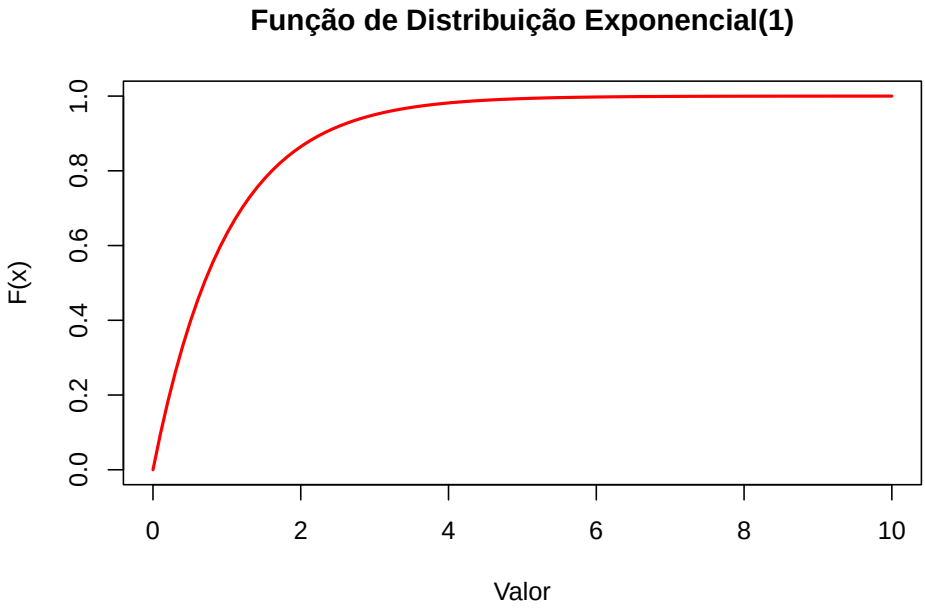
```
# Definir o tamanho da amostra
n <- 10000

# 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)
```



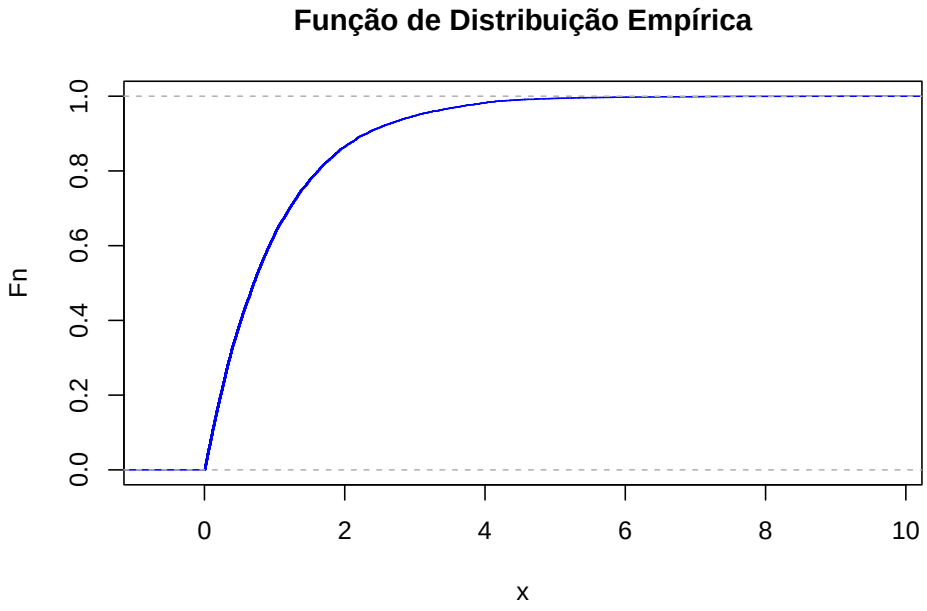
17.5.5 Função de distribuição

```
# 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)")
```



17.5.6 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 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")
```



17.6 Gerando uma variável aleatória com distribuição Normal

17.6.1 Cálculo de probabilidades

Seja $X \sim \text{Normal}(0, 1)$.

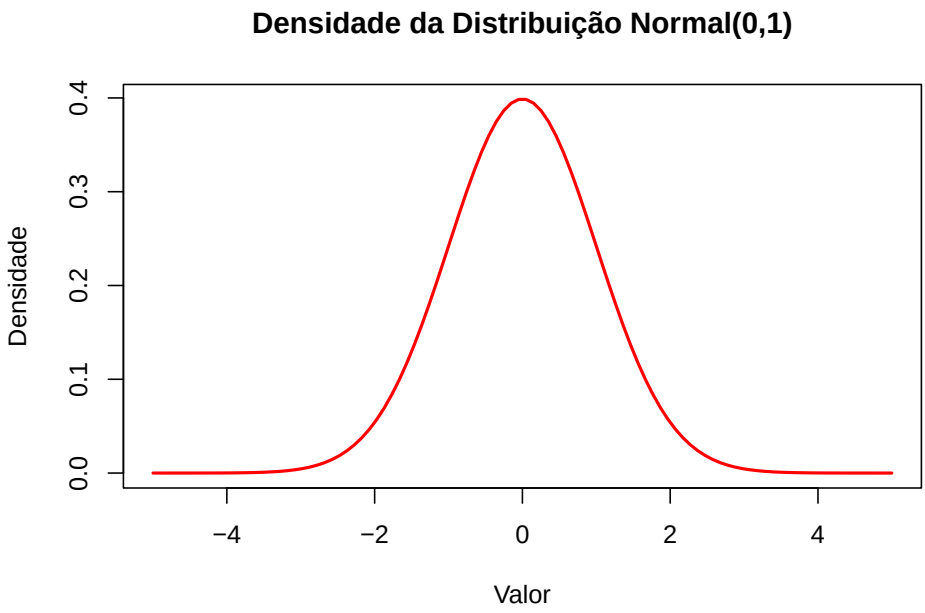
$P(X \leq 0.5) \rightarrow$ `pnorm(0.5,mean=0,sd=1)`$=0.6915$

$P(X > 0.5) \rightarrow$ `pnorm(0.5,mean=0,sd=1,lower.tail=FALSE)`$=0.3085$

17.6.2 Função densidade de probabilidade (teórica)

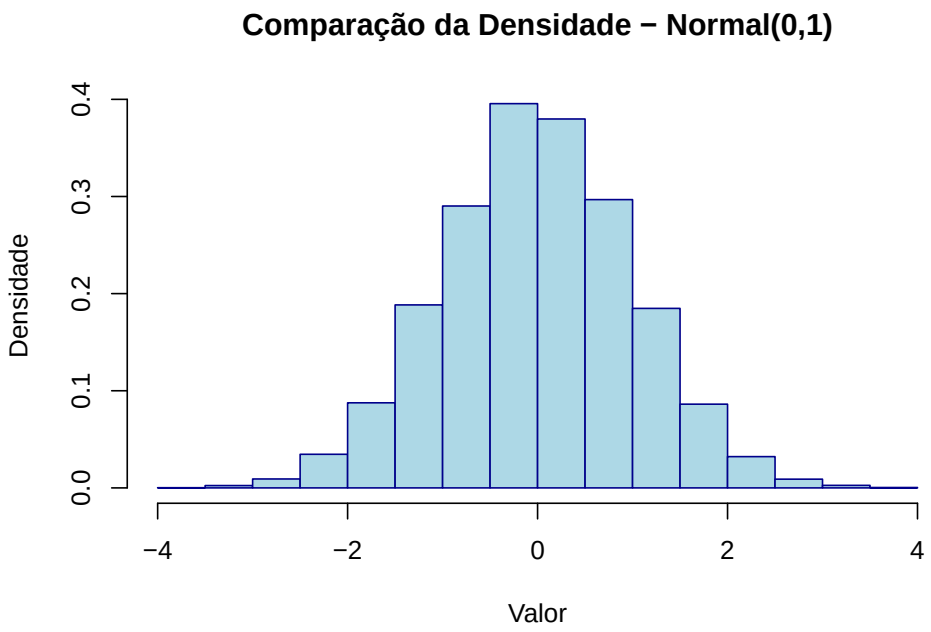
```r
# Gerar os valores x para a densidade teórica
x_vals <- seq(-5, 5, length.out = 100)

# 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")
```



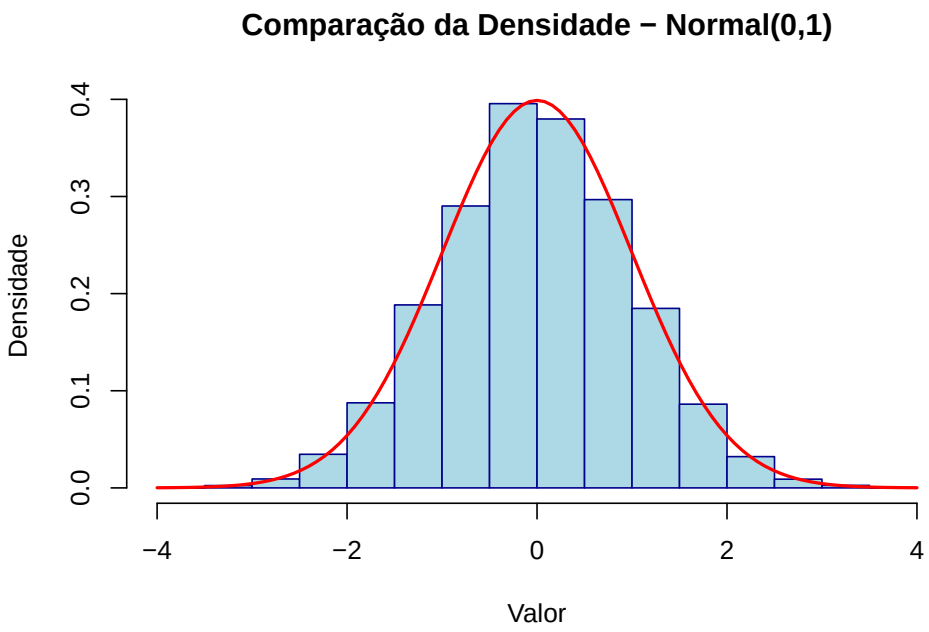
17.6.3 Função densidade de probabilidade (simulação)

```
# 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")
```



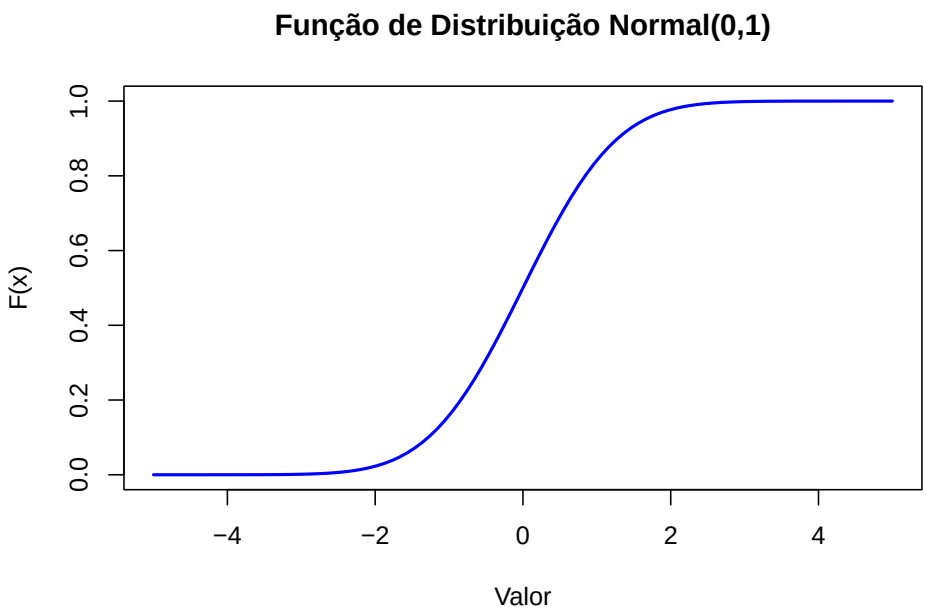
17.6.4 Comparação

```r
# 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)
```



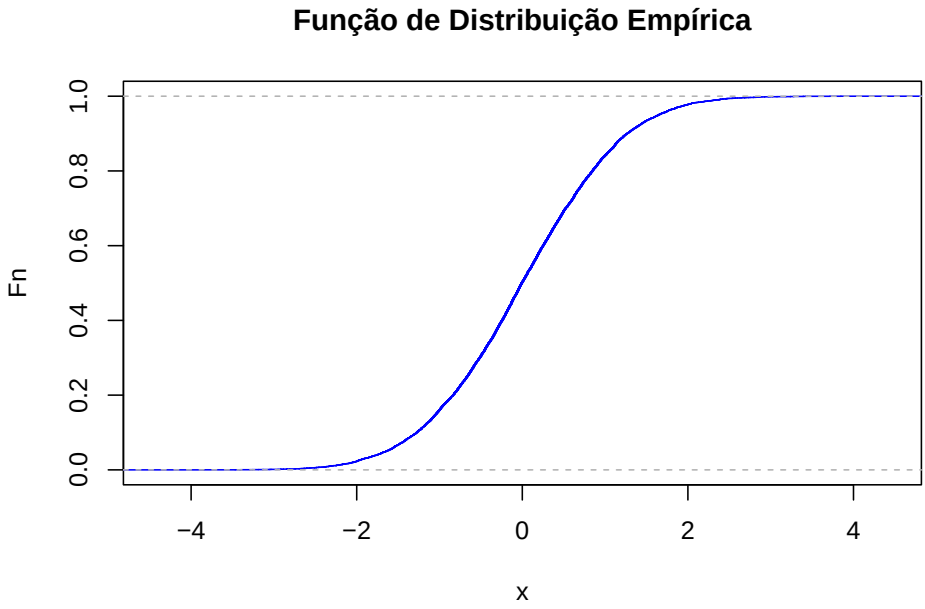
17.6.5 Função de distribuição

```r
# 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)")
```



17.6.6 Função de distribuição empírica

```r
# 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")
```



17.7 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 distruibuição 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 18

Relatórios

18.1 Markdown

18.2 R Markdown

Chapter 19

Referências

- https://cemapre.iseg.ulisboa.pt/~nbrites/CTA/index.html
- https://livro.curso-r.com/