Introdução à Programação e à Ciência de Dados

Prof. Dr. Helson Gomes de Souza

2025-07-07

Table of contents

Prefácio

Esse é um material construído para auxiliar os discentes do curso de Ciências Econômicas da Universidade Regional do Cariri no decorrer da disciplina de *Introdução à programação e à ciência de dados,* ministrada no terceiro período do curso. O material possui os instrumentos básicos para o acompanhamento do curso, incluindo explicações, comandos e exercícios. O material está organizado na ordem cronológica dos conteúdos repassados na disciplina de tal modo que essa ordem tem o intúito de conduzir o discente no aprendizado da programação computacional aplicada à economia em um cronograma planejado de acordo com as boas práticas do aprendizado da programação, evitando que o discente comece os estudos por conteúdos inadequados para os iniciantes nessa área.

O foco do material é a linguagem R, a mesma utilizada como objeto central da disciplina. Contudo, os métodos de aprendizado também podem ser expandidos para outras linguagens, desde que as devidas adaptações de vocabulário sejam consideradas. O material também possui um foco na análise de dados, desviando-se da programação computacional *"bruta"* devido a necessidade do curso e dos economistas em dominar os fundamentos básicos da utilização de dados.

1 Introdução

1.1 A lógica de programação

Imagine que o computador é uma empresa que realiza diferentes tarefas e que nessa empresa existem funcionários de várias partes do mundo. Cada funcionário realiza as mesmas tarefas, porém, falando diferentes idiomas. Assim, se você quer que um funcionário americano realize uma tarefa, terá que ordenar que ele a faça em inglês; se você quer que um funcionário italiano realize a mesma tarefa, terá de fazer o pedido para ele falando italiano, e assim sucessivamente. No computador, os funcionários são as linguagens de programação e o idioma é o vocabulário da linguagem (ou a maneira de escrever os comandos em cada linguagem). Resumidamente e exemplificadamente, você pode executar uma tarefa usando R ou pode executar essa mesma tarefa usando Python ou qualquer outra linguagem de programação pois a lógica de comunicação com o computador será a mesma, porém, os comandos para execuar a mesma tarefa podem ser diferentes em cada linguagem. Como consequência, o primeiro passo para dominar as linguagens de programação é desenvolver uma lógica de programação.

A lógica de programação é um raciocínio sobre como se comunicar com o computador para realizar tarefas. Seguindo o exemplo da empresa citado no parágrafo anterior, se você precisa de um relatório impresso, terá de falar o seguinte *"eu preciso que você imprima esse relatório"*. Porém, como os funcionários falam diferentes idiomas, você precisa fazer esse pedido em uma linguagem diferente a depender do funcionário responsável pela impressão. A maneira como você faz o pedido (*"eu preciso que você imprima esse relatório")* não vai mudar independente do funcionário que receberá a ordem, o que muda é apenas o idioma em que a ordem será feita. No computador, essa maneira como você faz o pedido é a lógica de programação, ela será a mesma para qualquer linguagem de programação, que no caso do exemplo corresponde ao idioma falado.

O primeiro passo para desenvolver uma lógica de programação é entender as operações fundamentais de uma **linguagem de programação**. Uma linguagem de programação é nada mais do que uma maneira de se comunicar com o computador para ordenar que ele execute tarefas automáticas. As linguagens são configuradas em sistemas binários (negação e afirmação). Cada linguagem de programação possui o seu **vocabulário**, porém, todas as linguagens partem dos mesmos princípios e das mesmas lógicas fundamentais. **Exemplos**: JavaScript, R, Python, Julia, Ruby, Scala...

1.2 Como usar a linguagem de programação

O primeiro paso para se comunicar com o computador por meio de uma linguagem de programação é informar à máquina a sua intenção de usar o vocabulário da linguagem. Em outras palavras, é preciso "instalar" a linguagem em sua máquina. Na analogia da empresa anteriormente mencionada, instalar a linguagem de programação seria como fazer um curso de idiomas e receber um manual de instruções de gramática para falar com os empregados.

Nesse material, utilizaremos a linguagem R. Seguindo o primeiro passo, a instalação da linguagem deve ser feita antes de qualquer outro procedimento. Os métodos de instalação variam de acordo com o sistema operacional do usuário. Nesse material, vamos supor que o aluno dispõe de uma máquina com sistema operacional windows. Nesse caso, o usuário deve baixar os componentes da linguagem nesse endereço Tendo feito isso, o usuário deve executar o arquivo baixado e instalar normalmente como qualquer instalação convencional no sistema windows.

O segundo passo é adotar um **ambiente de execução**. Esses ambientes são softwares também conhecidos como ***ambiente de desenvolvimento integrado*** *(IDE)* - Exemplos: Rstudio, Pycharm, Google Colaboratory, StataMP-. De forma geral, o IDE é uma plataforma onde você escreve as ordens que deseja que o computador execute em uma determinada linguagem. Especificando com outras palavras, o ambiente de execução é um software usado para escrever as ordens ao computador por meio do vocabulário da linguagem. Como vamos usar a linguagem R nesse material, temos que adotar um IDE que execute ordens nessa linguagem. É recomendado que o usuário utilize o Rstudio apesar de existirem diversas outras ferramentas com esse mesmo propósito. Para tanto, é preciso baixar o programa nesse endereço e proceder com com os procedimentos padrões de instalação de softwares no sistema windows.

É importante ressaltar que uma linguagem de programação não é um software, por exemplo, é errôneo se referir à linguagem R como *"software R"*, assim como é equivocado se referir à linguagem *stata* como *"software stata"*, e assim sucessivamente. Também é importante destacar que os IDEs não são linguagens de programação, eles apenas transferem para o computador uma ordem para ser executado em uma determinada linguagem. Com isso, é errôneo afirmar que uma dada tarefa foi *"executada pelo software Rstudio"*.

Ao instalar o Rstudio, o usuário irá se deparar com a tela ilustrada na imagem a seguir. Note que o IDE é formado por quatro painéis, cada um deles com a sua funcionalidade. O painel 1 é conhecido como *input* ou painel de entrada. Nesse painel o usuário irá abrir e manipular os arquivos de entrada como scripts e códigos de programação em R. Outros arquivos de entrada também são suportados como arquivos *html, markdown*, dentre outros. Mas essas extensões adicionais não são o foco do curso. Para gerar um arquivo de entrada onde serão escritos os comandos da linguagem R, pressione *Ctrl Shift n* ou vá na parte superior esquerda, na barra de tarefas do IDE em *file, new file, r script*.

O painel 2 é conhecido como *console*. Nesse painel serão expostos os resultados das tarefas executadas nos comandos escritos no painel 1. O usuário também pode escrever e executar os

comandos diretamente no console, com a diferença de que no *r script* os códigos podem ser salvos para o uso posterior ao contrário do console.

O painel 3 é conhecido como ambiente de trabalho ou *working environment*. Ele é dividido em quatro abas que podem ser acessadas separadamente, sendo: o *Environment* onde são expostos todos os objetos criados, o histórico *(history)* onde os últimos comandos ficam armazenados numa espécie de breve histórico de comandos, as conecções *(Connectione)* onde são expostas as ferramentas conectadas a linguagem R e ao Rstudio para executar tarefas, e a aba de tutoriais, onde o usuário pode acessar um breve guia sobre como usar o Rstudio. Versões mais recentes também disponibilizam abas adicionais sobre controle de versionamento no ambiente de trabalho, mas por enquanto esse não é o foco do curso.

O painel 4 é conhecido como *output* ou painel de saída. Nele o usuário pode visualizar os produtos gerados com os comandos e navegar pelo diretório de trabalho. O painel é dividido em cinco abas, sendo: a aba de arquivos *(files)*, onde o usuário pode navegar pela pasta que está usando; a aba *plots*, onde o usuário pode visualizar gráficos e figuras geradas com os comandos escritos no painel 1; a aba de bibliotecas *(packages)* onde o usuário pode acessar as bibliotecas instaladas na linguagem; a aba de ajuda *(Help)*, onde o usuário pode consultar manuais de instrução sobre diferentes comandos e bibliotecas; e por fim a aba de visualização *(View)*, onde o usuário pode visualizar objetos dinâmicos ou estáticos criados com alguma ferramenta adicional auxiliar à linguagem R como páginas web, arquivos pdf, etc.

1.3 Comandos

Os ambientes de execução podem operar em ordens diretas (cliques) ou comunicação agrupada *(comandos)*. Os ambientes de execução sugeridos nesta disciplina operam com *linhas de comando*. Nas linhas de comando, o operador (aluno) escreve uma ordem que o ambiente de execução entende como um comando para executar uma tarefa. Esses comandos são escritos individualmente em cada linha, isto é, cada linha escrita só agrupa um único comando. Ao escrever uma linha de comando no Rstudio, você poderá executar essse comando posicionando o cursor na linha (ou selecionando as linhas de interesse) e em seguida pressionando *"Ctrl Enter"*.

Alguns comandos já estão pré-programados no vocabulário das linguagens. Porém outros comandos precisam ser elaborados pelo próprio usuário usando os comandos pré-progamados e as demais ferramentas próprias do vocabulário da linguagem. Por exemplo, se você quer somar dois números quaisquer a e b, você pode fazer isso com uma operação de soma que já é automaticamente reconhecida pela linguagem R. Mas se você quer somar $a + b$ apenas se $a > b$, então você precisa usar a lógica de programação para programar esse comando.

1.4 Bibliotecas

O usuário pode ordenar que o computador execute uma tarefa por meio de comandos. Além disso, o usuário também pode unir vários comandos em uma função para executar uma tarefa de interesse. Muitas funções já estão disponíveis na própria linguagem nativa, por exemplo, caso o usuário queira gerar um gráfico ele pode usar o comando *plot()* que é uma função que usa vários comandos nativos da linguagem *R* para gerar uma figura. Assim como a função *plot*, muitas outras funções já estão disponíveis e prontas para o acesso no ato da instalação da linguagem. Porém, existem funções que são criadas por terceiros e que precisam ser instaladas para que possam ser usadas. Por exemplo, em vez de elaborar um gráfico com a função *plot*, o usuário pode usar a função *ggplot*. No entanto, essa função só pode ser usada caso a biblioteca *ggplo2* esteja instalada. O usuário pode verificar se uma determinada biblioteca está instalada navegando pelo painel inferior direito, na aba *Packages*, digitando o nome da biblioteca na guia de busca. Caso a biblioteca esteja instalada, então a busca retornará um indicativo com o nome da biblioteca, do contrário, o resultado da busca será vazio.

Para instalar uma biblioteca o usuário deve usar o comando *install.packages()* posicionando no parêntesis o nome da biblioteca entre aspas. Por exemplo, para instalar a biblioteca *ggplot2*, o usuário deve executar o comando *install.packages("ggplot2")*. Feito isso, a biblioteca estará disponível na lista de bibliotecas instaladas da aba *Packages* do painel inferior direito. Uma vez instalada, a instalação não precisa ser refeita, exceto em caso de atualização para uma nova versão.

No entanto, não basta instalar a biblioteca para usufruir de suas funções. Sempre que o Rstudio for reiniciado ou sempre que uma nova seção for iniciada no Rstudio é necessário ***liberar a biblioteca para o uso***. Isso é feito por meio do comando *library()* ou *require()* sempre posicionando entre parêntesis o nome da biblioteca desejada, dessa vez, sem aspas. Por exemplo, para liberar a biblioteca *ggplot2* para o uso, proceda conforme a seguir:

```
library(ggplot2)
```

1.5 Operações fundamentais em R

1.5.1 Soma

Para efetuar uma soma, você deve utilizar o operador de adição *"+"*. Por exemplo, para somar $1 + 1$ proceda como a seguir:

```
1+1
```

```
[1] 2
```

1.5.2 Subtração

Para efetuar uma subtração, você deve utilizar o traço simples "-" como operador de subtração. Por exemplo, para subtrair $1 - 1$ proceda como a seguir:

```
1-1
```

```
[1] 0
```

1.5.3 Multiplicação

Para efetuar uma multiplicação, você deve utilizar o asterisco "*" como operador de multiplicação. Por exemplo, para somar 1 x 1 proceda como a seguir:

```
1*1
```

```
[1] 1
```

1.5.4 Divisão

Para efetuar uma divisão, você deve utilizar a barra simples "/" como operador de divisão. Por exemplo, para dividr 4 por 2, proceda como a seguir:

```
4/2
```

```
[1] 2
```

1.5.5 Potência

Para efetuar uma potenciação, você deve utilizar o circunflexo "^" ou o duplo asterisco "**" como operador de multiplicação. Por exemplo, para calcular 2^3 proceda como a seguir:

```
2^3
```

```
[1] 8
```

O que também pode ser feito da seguinte maneira:

```
2**3
```

```
[1] 8
```

1.5.6 Raíz quadrada

Para efetuar uma radiciação, você deve utilizar o comando *sqrt* posisionando o número em prêntesis. Por exemplo, para calcular $\sqrt{16}$ proceda como a seguir:

```
sqrt(16)
```

```
[1] 4
```

1.5.7 Logaritmo

Para calcular o logaritmo de um número, você deve utilizar ocomando *log10* posisionando o número em prêntesis. Por exemplo, para calcular $log(2)$ proceda como a seguir:

```
log10(2)
```

```
[1] 0.30103
```

1.5.8 Logaritmo natural

Para calcular o logaritmo natural de um número, você deve utilizar ocomando *log* posisionando o número em prêntesis. Por exemplo, para calcular $ln(2)$ proceda como a seguir:

```
log(2)
```

```
[1] 0.6931472
```

1.5.9 Seno

Para calcular o seno de um número, você deve utilizar ocomando *sin* posisionando o número em prêntesis. Por exemplo, para calcular o seno de 90 proceda como a seguir:

```
sin(90)
```

```
[1] 0.8939967
```

1.5.10 Cosseno

Para calcular o cosseno de um número, você deve utilizar ocomando *cos* posisionando o número em prêntesis. Por exemplo, para calcular o cosseno de 90 proceda como a seguir:

```
cos(90)
```

```
[1] -0.4480736
```

1.5.11 Tangente

Para calcular a tangente de um número, você deve utilizar ocomando *tan* posisionando o número em prêntesis. Por exemplo, para calcular a tangente de 90 proceda como a seguir:

```
tan(90)
```

```
[1] -1.9952
```

1.5.12 Divisão inteira

Algumas divisões resultam em números não inteiros, de tal modo que o resultado é composto por um número inteiro aderido de um "resto". Para calcular a divisão sem o resto você deve usar o operador *%/%*. Por exemplo, uma divisão inteira de cinco por dois deve resultar em dois e pode ser feita da seguinte maneira:

```
5 %/% 2
```

```
[1] 2
```

1.5.13 Resto da divisão

Porém, se o usuário estiver interessado em obter apenas o resto da divisão, pode usar o operador *%%*. No caso do exemplo anterior, o resto da divisão é um e pode ser obtido da seguinte maneira:

```
5 %% 2
```

```
[1] 1
```

1.6 Operadores lógicos

Os operadores lógicos são usados para comparar valores. O principal operador lógico de uma linguagem de programação é o operador de afirmação ou negação. Sempre que houver uma afirmação, a lingaugem retornará um sinal de verdadeiro (TRUE) e sempre que houver uma negação, a lingaugem retornará um sinal de falso (FALSE). Nos comandos, o TRUE pode ser substituído pelo T, enquanto o FALSE pode ser substituído pelo F. A linguagem R atribui o valor zero para as negações e o valor um para as afirmações. Assim, *TRUE = 1* e *FALSE = 0* sempre ocorrerá.

1.6.1 Igualdade

Para checar uma condição de igualdade, você deve usar o operador "==". Por exemplo, para checar se dois é igual a três, você deve proceder como:

```
2 == 3
```

```
[1] FALSE
```

1.6.2 Desigualdade maior que

Para checar uma condição de desigualdade na forma de maior que, isto é, para verificar se um valor é maior que outro, você deve usar o operador ">". Por exemplo, para checar se dois é maior que três, você deve proceder como:

```
2 > 3
```

```
[1] FALSE
```

1.6.3 Desigualdade menor que

Para checar uma condição de desigualdade na forma de menor que, isto é, para verificar se um valor é menor que outro, você deve usar o operador "<". Por exemplo, para checar se dois é menor que três, você deve proceder como:

```
2 < 3
```

```
[1] TRUE
```

1.6.4 Desigualdade maior ou igual

Para checar uma condição de desigualdade na forma de maior ou igual a, isto é, para verificar se um valor é maior ou igual outro, você deve usar o operador ">=". Por exemplo, para checar se dois é maior ou igual a três, você deve proceder como:

```
2 >= 3
```

```
[1] FALSE
```

1.6.5 Desigualdade menor ou igual

Para checar uma condição de desigualdade na forma de menor ou igual a, isto é, para verificar se um valor é menor ou igual outro, você deve usar o operador "<=". Por exemplo, para checar se dois é menor ou igual a três, você deve proceder como:

```
2 <= 3
```

```
[1] TRUE
```

1.6.6 Diferente de

Para checar se um valor é diferente de outro, você deve usar o operador "!=". Por exemplo, para checar se dois é diferente de três, você deve proceder como:

```
2 != 3
```

```
[1] TRUE
```

1.7 Objetos

As linguagens de programação geralmente são identificadas ao objeto, isto é, é possível criar um objeto que representa algum ítem ou valor. Em R, os objetos devem ser criados om um indicativo de igualdade "=". Por exemplo, imagine que precisamos criar um objeto com o nome "idade" contendo a idade de uma pessoa em anos. Este procedimento é feito informando o comando "nome do objeto = valor do objeto" conforme demonstrado a seguir:

```
idade = 18
```

Ao executar esse comando, um novo objeto surgirá na aba *Environment* do painel 3. Esse objeto tem o nome "idade" e recebe um valor de 18. Para visualizar o valor do objeto, o usuário pode usar a função *print* que imprimirá no painel 2 (console) o valor referente ao objeto mencionado.

```
print(idade)
```

```
[1] 18
```

Tendo feito isso, e dado que o objeto de nome *idade* e valor 18 está no *Environment*, cada vez que esse objeto for mencionado a linguagem R reconhecerá que se trata do número 18. Para exemplificar, suponha que uma pessoa é considerada idosa a partir dos 60 anos e suponha que você precise descobrir quantos anos ainda restam para que essa pessoa com idade = 18 se torne idosa. Nesse caso, você deve proceder como:

```
60 - idade
```

```
[1] 42
```

1.8 Tipos de objetos

Os objetos são maneiras de armazenar informações em um dado arranjo. Em R, essas informações podem ser arranjadas em funções, vetores, matrizes, listas, arrays ou quadros de dados (*data frames*). Cada objeto tem a sua função específica e deve ser usado conforme a necessidade. Por exemplo, uma matriz é ideal para armazenar objetos com duas dimensões (linha e coluna) mas não é adequada para agrupar objetos com três dimensões, nesse caso melhor seria usar um array ou uma lista.

1.8.1 Vetores

Os vetores são objetos que servem para guardar informações unidimensionais, isto é, informações que podem ser escritas em uma única linha ou coluna. Por exemplo, imagine que você trabalhou cinco dias em um emprego e notou em uma planilha o seu salário de cada dia. Suponha que os seus ganhos dia após dia em reais foram 50.00, 52.00, 55.00, 48.00, 60.00. Se você anotou essas informações em uma linha de uma planilha, então você tem um vetor linha. Analogamente, se as informações foram anotadas em uma coluna de uma planilha, tem-se um vetor coluna. Se você chamou essa planilha de "salario", então isso é o mesmo que:

$$salario = [50.00, 52.00, 55.00, 48.00, 60.00]$$

Para digitar esse vetor em *R* deve-se usar o operador de vetores *c()*, sempre colocando os valores dentro do parêntesis separando cada valor por uma vírgula. Lembre-se que o separador decimal da linguagem *R* é o ponto e que a vírgula é um separador de valores. Com isso, o vetor anterior deve ser escrito como:

```
salario = c(50.00, 52.00, 55.00, 48.00, 60.00)
```

Feito isso, um objeto de nome *salario* irá aparecer no ambiente de trabalho. Note que o nome do objeto é sucedido do termo *num [1:5]*, isso indica que se trata de um vetor numérico com cinco elementos. Um detalhe importante a ser mencionado é o fato de que valores não numéricos também podem ser armazenados em vetores, por exemplo:

```
nomes = c("João", "Maria", "José")
```

Note que **os valores não numéricos sempre devem estar entre aspas**. Para checar se um dado objeto é um vetor, o usuário pode usar a função *is.vector()* indicando o nome do objeto entre parêntesis. Por exemplo, para checar se o objeto *salario* é um vetor, proceda conforme a seguir:

```
is.vector(salario)
```

```
[1] TRUE
```

Caso o elemento de fato seja um vetor, o output obtido será *TRUE*, do contrário o output será *FALSE*. Para transformar um determinado objeto em um vetor, o usuário pode usar a função *as.vector()*, indicando o nome do objeto entre parêntesis. Por exemplo, para transformar uma sequência de 1 a 10 em um vetor, proceda conforme a seguir:

```r
sq = as.vector(1:10)
```

1.8.2 Matrizes

As matrizes são objetos que servem para guardar informações bidimensionais, isto é, informações que podem ser escritas em um múltiplas linhas e múltiplas colunas, desde que o usuário precise realizar operações algébricas com esses valores. Para exemplificar, considere o exemplo anterior do salário. Considere agora que você trabalhou cinco dias da semana não em um mas em dois empregos. Agora você vai atribuir um diasda semana para cada linha e vai anotar os ganhos de cada emprego em colunas diferentes. Suponha agora que os ganhos do emprego 2 foram de 140.00, 160.00, 165.00, 150.00 e 155.00. Isso equivale a:

$$salario = \left[\begin{array}{cc} Emprego1 & Emprego2 \\ \hline 50.00 & 140.00 \\ 52.00 & 160.00 \\ 55.00 & 165.00 \\ 48.00 & 150.00 \\ 60.00 & 155.00 \end{array}\right]$$

Agora a planilha de ganhos possui dois vetores coluna de cinco elementos cada ou cinco vetores linha de dois elementos cada. Para informar essa planilha como matriz no *R*, o usuário deve usar a função *matrix*. Nessa função o usuário deve digitar os elementos da planilha linha por linha em um único vetor e indicar isso com o parâmetro *by.row = TRUE*. Ou se preferir o usuário pode digitar os elementos da planilha coluna por coluna em um único vetor e indicar isso com o parâmetro *by.row = FALSE*. O usuário também deve informar o número de linhas da matriz com o parâmetro *nrow* e o número de colunas com o parâmetro *ncol*. Para repassar a planilha anterior em *R* na forma de matriz, proceda conforme a seguir:

```r
salario = matrix(
  c(50,140,52,160,55,165,48,150,60,155),
  byrow = TRUE,
  ncol = 2,
  nrow = 5
)

print(salario)
```

```
     [,1] [,2]
[1,]   50  140
[2,]   52  160
```

```
[3,]   55  165
[4,]   48  150
[5,]   60  155
```

Isso é o mesmo que fazer:

```r
salario = matrix(
  c(50,52,55,48,60,140,160,165,150,155),
  byrow = FALSE, # agrupamento por coluna
  ncol = 2,
  nrow = 5
)

print(salario)
```

```
     [,1] [,2]
[1,]   50  140
[2,]   52  160
[3,]   55  165
[4,]   48  150
[5,]   60  155
```

Para dar nomes às linhas de uma matriz, use a função *rownames(),* indicando o nome da matriz entre o parêntesis e informando os nomes das linhas em um vetor. Por exemplo:

```r
rownames(salario) = c("Seg", "Ter", "Quar", "Qui", "Sex")
print(salario)
```

```
     [,1] [,2]
Seg    50  140
Ter    52  160
Quar   55  165
Qui    48  150
Sex    60  155
```

Para dar nomes às colunas de uma matriz, use a função *colnames(),* indicando o nome da matriz entre o parêntesis e informando os nomes das colunas em um vetor. Por exemplo:

```r
colnames(salario) = c("Emprego 1","Emprego 2")
print(salario)
```

```
     Emprego 1 Emprego 2
Seg         50        140
Ter         52        160
Quar        55        165
Qui         48        150
Sex         60        155
```

Para verificar se um determinado objeto é uma matriz, use a função *is.matrix()*, indicando o nome do objeto entre o parêntesis, por exemplo:

```
is.matrix(salario)
```

```
[1] TRUE
```

Caso o elemento de fato seja uma matriz, o output obtido será *TRUE*, do contrário o output será *FALSE*. Para transformar um determinado objeto em uma matriz, o usuário pode usar a função *as.matrix()*, indicando o nome do objeto entre parêntesis. Por exemplo, para transformar uma sequência de 1 a 10 em uma matriz, proceda conforme a seguir:

```
sq = as.matrix(1:10)
print(sq)
```

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

1.8.2.1 Operações com matrizes

1.8.2.1.1 Soma de matrizes

A soma de matrizes em *R* não apresenta diferenças das operações convencionais de soma, ou seja, é feita usando o operador de soma "+". Para exemplificar, considere as dias matrizes a seguir:

$$matriz1 = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix} \qquad matiz2 = \begin{bmatrix} 5 & 3 \\ 7 & 0 \end{bmatrix}$$

Para gerar uma nova matriz de nome *matriz3* contendo a soma da matriz 1 com a matriz 2, basta proceder conforme a seguir:

```
matriz1 = matrix(c(0,2,3,1), nrow = 2, ncol = 2, byrow = TRUE)
matriz2 = matrix(c(5,3,7,0), nrow = 2, ncol = 2, byrow = TRUE)
matriz3 = matriz1 + matriz2
print(matriz3)
```

```
     [,1] [,2]
[1,]    5    5
[2,]   10    1
```

1.8.2.1.2 Subtração de matrizes

De maneira análoga à operação de soma, a subtração de matrizes é feita usando o operador de subtração "-". Nesse caso, os elementos de cada posição das matries são subtraídos. Para exemplificar, considere subtrair a matriz 1 da matriz 2, procedendo de acordo com o código a seguir:

```
matriz3 = matriz1 - matriz2
print(matriz3)
```

```
     [,1] [,2]
[1,]   -5   -1
[2,]   -4    1
```

É importante ressaltar que a soma e a subtração de matrizes só podem ser feitas com matrizes de mesma dimensão.

1.8.2.1.3 Multiplicação de matrizes

Na multiplicação, não é correto utilizar o asterisco como operador de multiplicação dado que multiplicar os termos de mesma posição de duas matrizes não é a maneira correta de efetuar o produto de matrizes. Para essa tarefa, o operador de multiplicação agora é $\%*\%$. Para exemplificar, considere multiplicarr a matriz 1 pela matriz 2.

```r
matriz3 = matriz1 %*% matriz2
print(matriz3)
```

```
     [,1] [,2]
[1,]   14    0
[2,]   22    9
```

1.8.2.1.4 Matriz transposta

A transposta de uma matriz nada mais é do que reorganizar as linhas como colunas e as colunas como linhas. Para obter a transposta de uma matriz em R basta usar a função *t()*, indicando entre parêntesis o nome da matriz que se deseja transpor. Para exemplificar, considere transpor a matriz 1. Nesse caso, deve-se proceder conforme a seguir:

```r
t(matriz1)
```

```
     [,1] [,2]
[1,]    0    3
[2,]    2    1
```

1.8.2.1.5 Matriz inversa

Uma matriz M pode ser invertida em R para obter M^{-1} usando o comando *solve()* e indicando dentro do parêntesis a matriz que se deseja inverter. Para exemplificar, considere obter a inversa da matriz 1. Nesse caso, deve-se proceder conforme a seguir:

```r
solve(matriz1)
```

```
           [,1]      [,2]
[1,] -0.1666667 0.3333333
[2,]  0.5000000 0.0000000
```

1.8.2.1.6 Exemplo de operações com matrizes: O estimador de mínimos quadrados ordinários

Na estatística, os coeficientes lineares de uma equação linear com múltiplos argumentos podem ser calculados por meio do método de mínimos quadrados ordinários. Se essa equação é:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Um termo de erro ε é adicionado e a relação anterior pode ser escrita matricialmente como:

$$\mathbf{y} = \mathbf{x}\beta + \varepsilon$$

Mininiando a soma dos erros quadráticos, o vetor β de parâmetros estimados é:

$$\beta = (\mathbf{x}'\mathbf{x})^{-1}\mathbf{x}'\mathbf{y}$$

Para exemplificar, considere usar a base natova sobre automóveis *mtcars* e suponha que estejamos interessados em saber a relação entre o consumo do automóvel (mpg) e as variáveis peso (wt) e número de cilindros (cyl). Nesse caso, a matriz y são os valores da coluna mpg e a matriz x é composta pelas colunas wt e cyl. O vetor de parâmetros estimados pode ser calculado de acordo com o seguinte processo:

```
y = matrix(mtcars$mpg, ncol = 1) # matriz y
x = matrix(c(rep(1,nrow(mtcars)), mtcars$wt, mtcars$cyl), ncol = 3) # matriz x
tx = t(x) # transposta da matriz x
beta = solve(tx%*%x)%*%tx%*%y
print(beta)
```

```
          [,1]
[1,] 39.686261
[2,] -3.190972
[3,] -1.507795
```

Isso é o mesmo que fazer:

```
summary(lm(mtcars$mpg ~ 1 + mtcars$wt + mtcars$cyl))
```

```

Call:
lm(formula = mtcars$mpg ~ 1 + mtcars$wt + mtcars$cyl)

Residuals:
    Min      1Q  Median      3Q     Max
-4.2893 -1.5512 -0.4684  1.5743  6.1004

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  39.6863     1.7150  23.141  < 2e-16 ***
```

```
mtcars$wt    -3.1910     0.7569  -4.216 0.000222 ***
mtcars$cyl   -1.5078     0.4147  -3.636 0.001064 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.568 on 29 degrees of freedom
Multiple R-squared:  0.8302,    Adjusted R-squared:  0.8185
F-statistic: 70.91 on 2 and 29 DF,  p-value: 6.809e-12
```

1.8.3 Arrays

O conceito de array generaliza a idéia de matrix. Enquanto em uma matrix os elementos são organizados em duas dimensões (linhas e colunas), em um array os elementos podem ser organizados em um número arbitrário de dimensões. Em *R* um array é definido utilizando a função array(). O usuário deve informar dentro do parêntesis as informações sempre ordenadas coluna a coluna e indicar as dimensões do array com o parâmetro *dim*. Por exemplo, imagine que queiramos armazenar o consumo de dois bens por duas famílias em três anos, neste caso, teremos que criar um array de dimensão 2 x 2 x 3. Para exemplificar, imagine que esse consumo seja o demonstrado a seguir:

$$Ano1 = \left[\begin{array}{c|cc} & Bem1 & Bem2 \\ \hline Familia1 & 0 & 5 \\ Familia2 & 2 & 3 \end{array}\right] \quad Ano2 = \left[\begin{array}{c|cc} & Bem1 & Bem2 \\ \hline Familia1 & 1 & 4 \\ Familia2 & 3 & 2 \end{array}\right]$$

$$Ano3 = \left[\begin{array}{c|cc} & Bem1 & Bem2 \\ \hline Familia1 & 2 & 3 \\ Familia2 & 2 & 3 \end{array}\right]$$

Em *R* isso equivale a:

```r
consumo = array(
  c(
    0,2,5,3, # Ano 1
    1,3,4,2, # Ano 2
    2,2,3,3 # ano 3
  ),
  dim = c(2,2,3)
)
print(consumo)
```

```
, , 1
```

```
     [,1] [,2]
[1,]    0    5
[2,]    2    3

, , 2

     [,1] [,2]
[1,]    1    4
[2,]    3    2

, , 3

     [,1] [,2]
[1,]    2    3
[2,]    2    3
```

Para checar se um objeto é um array, use a função *is.array()*. Caso o objeto de fato seja um array, o output obtido será *TRUE*, do contrário o output será *FALSE*.

```
is.array(consumo)
```

```
[1] TRUE
```

1.8.4 Listas

As listas são objetos ideais para guardar informações em múltiplas dimensões, isto é, informações que possuam múltiplas linhas, múltiplas colunas e múltiplas planilhas. As listas podem armazenar diversos outros objetos, incluindo outras listas. Por exemplo, uma lista pode conter um vetor e uma matriz ou múltiplos vetores e múltiplas matrizes. Para criar uma lista, o usuário deve usar a função *list()* indicando dentro do parêntesis os objetos que irão compor a lista. Por exemplo, vamos criar uma lista contendo os as matrizes de consumo do array anterior e vamos chamar essa lista de *lista1*.

```
ano1 = matrix(
  c(0,5,2,3), byrow = TRUE, nrow = 2, ncol = 2
)
ano2 = matrix(
  c(1,4,3,2), byrow = TRUE, nrow = 2, ncol = 2
)
ano3 = matrix(
```

```
  c(2,3,2,3), byrow = TRUE, nrow = 2, ncol = 2
)

lista1 = list(ano1, ano2, ano3)
print(lista1)
```

```
[[1]]
     [,1] [,2]
[1,]    0    5
[2,]    2    3

[[2]]
     [,1] [,2]
[1,]    1    4
[2,]    3    2

[[3]]
     [,1] [,2]
[1,]    2    3
[2,]    2    3
```

Para checar se um objeto é uma lista, use a função *is.list()*. Caso o elemento de fato seja uma lista, o output obtido será *TRUE*, do contrário o output será *FALSE*.

```
is.list(lista1)
```

```
[1] TRUE
```

1.8.5 Data frames

Os data frames - ou quadro de dados - são objetos que possuem utilidade semelhante às matrizes, isto é, são ideais para armazenar informações bidimensionais. No entanto, nos data frames não é possível realizar operações algébricas como nas matrizes. Os data frames são exatamente iguais às planilhas do excel, onde cada coluna é uma variável com valores distribuídos entre as linhas. Essa estrutura de dados é inserida em *R* usando a função *data.frame()*, de tal modo que o usuário precisa indicar o nome de cada coluna precedida pelos seus valores em um vetor. Por exemplo, se usarmos o exemplo anterior do salário de dois empregos e quisermos colocar as informações dos ganhos em um data frame de nome *salario_diario*, devemos proceder conforme a seguir:

```r
salario_diario = data.frame(
  emprego1 = c(50,52,55,48,60),
  emprego2 = c(140,160,165,150,155)
)

print(salario_diario)
```

```
  emprego1 emprego2
1       50      140
2       52      160
3       55      165
4       48      150
5       60      155
```

Você pode mudar os nomes das colunas dos data frames usando a função *colnames()*. Por exemplo, se quisermos alterar o nome das colunas para *trabalho1* e *trabalho2*, devemos proceder conforme a seguir:

```r
colnames(salario_diario) = c("trabalho1", "trabalho2")
print(salario_diario)
```

```
  trabalho1 trabalho2
1        50       140
2        52       160
3        55       165
4        48       150
5        60       155
```

Já os nomes das linhas podem ser alterados por meio da função *rownames()*.

```r
rownames(salario_diario) = c("Seg", "Ter", "Quar", "Qui", "Sex")
print(salario_diario)
```

```
     trabalho1 trabalho2
Seg         50       140
Ter         52       160
Quar        55       165
Qui         48       150
Sex         60       155
```

Para checar se um objeto é um data frame, o usuário deve usar a função *is.data.frame()* indicando o nome do objeto dentro do parêntesis. Caso o objeto de fato seja um data frame, o output obtido será *TRUE*, do contrário o output será *FALSE*.

```
is.data.frame(salario_diario)
```

```
[1] TRUE
```

Assim como em uma planilha excel, as colunas dos data frames podem conter valores numéricos e não numéricos. Maiores detalhes sobre essas possibilidades de valores serão vistos posteriormente quando abordarmos as classes dos elementos.

1.9 Exercício 1

(1) Crie um objeto de nome *valor1* com a operação $\left(\frac{5^5}{100}\right)^{0.5}$.

(2) Crie um objeto de nome *valor2* com a operação $\left(\frac{3^8}{100}\right)^{0.5}$.

(3) Cheque se $\left(\frac{5^5}{100}\right)^{0.5} \neq \sqrt{\left(\frac{5^5}{100}\right)}$.

(4) Cheque se $\left(\frac{3^8}{100}\right)^{0.5} \neq \sqrt{\left(\frac{3^8}{100}\right)}$.

(5) Verifique se $\left(\frac{5^5}{100}\right)^{0.5} >= \left(\frac{3^8}{100}\right)^{0.5}$.

(6) Divida *valor1* por *valor2* e verifique se o resultado é maior que 1.

1.10 Exercício 2

Considere as seguintes planilhas de dados representando os preços das ações das empresas Petrobras, Vale e Itau em uma semana de negociações na bolsa de valores:

$$petrobras = \left[\begin{array}{c|ccc} Dia & Preco & Maximo & Minimo \\ \hline Seg & 34.5 & 34.75 & 33.8 \\ Ter & 34.7 & 35.05 & 34.2 \\ Quar & 34.9 & 35.5 & 34.6 \\ Qui & 34.55 & 34.9 & 34.3 \\ Sex & 34 & 34.55 & 33.6 \end{array}\right]$$

$$vale = \left[\begin{array}{c|ccc} Dia & Preco & Maximo & Minimo \\ \hline Seg & 55.5 & 55.75 & 55.1 \\ Ter & 56 & 56.6 & 55.5 \\ Quar & 56.5 & 56.75 & 56 \\ Qui & 55.8 & 56 & 55.2 \\ Sex & 55.2 & 55.8 & 55 \end{array}\right]$$

$$itau = \left[\begin{array}{c|ccc} Dia & Preco & Maximo & Minimo \\ \hline Seg & 28.5 & 28.75 & 28.3 \\ Ter & 28.7 & 29.05 & 28.5 \\ Quar & 28.9 & 29.2 & 28.7 \\ Qui & 28.6 & 28.9 & 28.5 \\ Sex & 28.3 & 28.6 & 28.1 \end{array}\right]$$

(1) Repasse essas três planilhas para o R na forma de matrizes, nomeando-as de *petrobras*, *vale* e *itau*, assim como esboçado na representação das planilhas. Dê nomes às linhas e às colunas.

(2) Agora repasse as planilhas para o R na forma de data frames, nomeando-os de *df_petrobras*, *df_vale* e *df_itau*. Dê nomes às linhas e às colunas.

(3) Transforme as matrizes *petrobras*, *vale* e *itau* em data frames, nomeando-os de *df_petrobras_2*, *df_vale_2* e *df_itau_2*.

(4) Transforme os data frames *df_petrobras*, *df_vale* e *df_itau* em matrizes, nomeando-as de *m_petrobras*, *m_vale* e *m_itau*.

(5) Cheque se os objetos *petrobras*, *vale* e *itau* são matrizes.

(6) Cheque se os objetos *df_petrobras*, *df_vale* e *df_itau* são data frames.

(7) Repasse as três planilhas para o R em um array nomeando-o de *preco_acoes*.

(8) Cheque se o objeto criado na questão anterior é um array.

2 Indexação e operações indexadas

Nesse módulo estão expressos os detalhes básicos acerca de operações fundamentais com diferentes tipos de objetos. O módulo ainda apresenta os conceitos e as normas básicas sobre posições de elementos em um objeto (indexação) e também mostra as operações que usam essa indexação e que podem ser úteis na análise de dados.

2.1 Indexação de objetos

Os objetos são compostos por elementos e esses elementos ocupam uma posição dentro do objeto. Por exemplo, imagine um vetor $x = [Maria, Paulo, Pedro, Ana]$. Esse vetor é composto por quatro elementos, maria na posição 1, Paulo na posição 2, Pedro na posição 3 e Ana na posição 4. Agora considere que essas mesmas informações estejam dispostas em uma matriz.

$$\begin{bmatrix} Maria & Paulo \\ Pedro & Ana \end{bmatrix}$$

Agora a posição deve ser visualizada como "linha por coluna". Maria está na linha 1 e coluna 1 (posição *[1, 1])*, Paulo está na linha 1 e coluna 2 (posição *[1, 2])*, Pedro está na linha 2 e coluna 1 (posição *[2, 1])* e Ana está na linha 2 e coluna 2 (posição *[2, 2])*. A essa posição dá-se o nome de indexação. A indexação nada mais é do que a posição de um elemento em um conjunto, que nesse caso são os objetos no ambiente de trabalho.

Para verificar qual elemento está em uma determinada posição do objeto, deve-se informar o nome do objeto precedido da posição entre colchetes, isto é, *nome do objeto[posição na linha, posição na coluna]*. Por exemplo, na matriz anterior, caso queiramos consultar quem está na linha 1 da coluna 1, devemos proceder conforme a seguir:

```
nomes = matrix(c("Maria", "Paulo", "Pedro", "Ana"), byrow = TRUE, nrow = 2, ncol = 2)
nomes[1,1]
```

```
[1] "Maria"
```

A lógica é a mesma em um data frame,

```r
df = data.frame(
  coluna1 = c("Maria", "Pedro"),
  coluna2 = c("Paulo", "Ana")
)
df[1,1]
```

```
[1] "Maria"
```

Se quisermos nos referir apenas às linhas, então a posição da coluna deve ficar vazia, isto é, *nome do objeto[posição na linha,]*. Seguindo o exemplo anterior, caso queiramos checar quem está na linha 1 da matriz, devemos proceder conforme a seguir:

```r
nomes[1,]
```

```
[1] "Maria" "Paulo"
```

A mesma lógica se aplica aos data frames.

```r
df[1,]
```

```
  coluna1 coluna2
1   Maria   Paulo
```

Se quisermos nos referir apenas às colunas, então a posição da linha deve ficar vazia, isto é, *nome do objeto[, posição na coluna]*. Seguindo o exemplo anterior, caso queiramos checar quem está na coluna 1 da matriz, devemos proceder conforme a seguir:

```r
nomes[,1]
```

```
[1] "Maria" "Pedro"
```

A mesma lógica se aplica aos data frames.

```r
df[,1]
```

```
[1] "Maria" "Pedro"
```

A posição das linhas e colunas também pode ser referenciada de acordo com os nomes. Nesse caso, o procedimento é *nome do objeto["nome da linha", "nome da coluna"]*. Para exemplificar, vamos dar nomes às linhas e as colunas da matriz e do data frame usados nos exemplos anteriores.

```
rownames(nomes) = c("linha 1", "linha 2")
colnames(nomes) = c("coluna 1", "coluna 2")
print(nomes)
```

```
        coluna 1 coluna 2
linha 1 "Maria"  "Paulo"
linha 2 "Pedro"  "Ana"
```

```
rownames(df) = c("linha 1", "linha 2")
print(nomes)
```

```
        coluna 1 coluna 2
linha 1 "Maria"  "Paulo"
linha 2 "Pedro"  "Ana"
```

Para checar quem está na linha 1, proceda conforme a seguir:

```
nomes["linha 1",]
```

```
coluna 1 coluna 2
 "Maria"  "Paulo"
```

E no data frame:

```
df["linha 1", ]
```

```
        coluna1 coluna2
linha 1   Maria   Paulo
```

Para checar quem está na coluna 1, proceda conforme a seguir:

```
nomes[, "coluna 1"]
```

```
linha 1 linha 2
"Maria" "Pedro"
```

E no data frame:

```
df[, "coluna1"]
```

```
[1] "Maria" "Pedro"
```

Nos data frames, a indexação das colunas pode ser mais simplificada devido a possibilidade de referenciar as colunas plo nome usando o cifrão "$". Nesse caso, o procedimento a se fazer é *nome do objeto$nome da coluna*. Por exemplo, para se referir a coluna 1 do data frame anterior, deve-se proceder conforme a seguir:

```
df$coluna1
```

```
[1] "Maria" "Pedro"
```

2.2 Adicionando linhas a um data frame

Tendo comopreendido como funciona a indexação, o próximo passo é aprender como essa indexação pode ser usada para adicionar, excluir ou alterar elementos de um objeto. Caso a intenção seja adicionar uma linha em um data frame, então o procedimento a se fazer é *nome do objeto[número de linhas do objeto + 1,]*. Para exemplificar, vamos incluir uma linha na data frame de nomes contendo os valores *[João, Clara]*. Esse procedimento deve ser feito conforme especificado a seguir:

```
df[3,] = c("João", "Clara")
print(df)
```

```
        coluna1 coluna2
linha 1   Maria   Paulo
linha 2   Pedro     Ana
3          João   Clara
```

2.3 Adicionando linhas em uma matriz

Para adicionar linhas em uma matriz é preciso usar a função *rbind()*. Essa função também funciona perfeitamente para data frames. Nesse caso, o procedimento a se fazer é *rbind(nome do objeto, vetor com os valores da linha adicionada)*. Por exemplo, para adicionar uma linha com os valores *[João, Clara]* no data frame de nomes, deve-se proceder conforme a seguir:

```
nomes = rbind(nomes, c("João", "Clara"))
print(nomes)
```

```
        coluna 1 coluna 2
linha 1 "Maria"  "Paulo"
linha 2 "Pedro"  "Ana"
        "João"   "Clara"
```

A mesma lógica se aplica aos data frames:

```
rbind(df, c("João", "Clara"))
```

```
        coluna1 coluna2
linha 1   Maria   Paulo
linha 2   Pedro     Ana
3          João   Clara
4          João   Clara
```

2.4 Removendo linhas de um objeto

A remoção de linhas de um objeto é feita adicionando um sinal de menos antes do número da linha indicado em colchetes. Por exemplo, caso queiramos remover a linha 3 da matriz de nomes, devemos proceder conforme a seguir:

```
nomes[-3,]
```

```
        coluna 1 coluna 2
linha 1 "Maria"  "Paulo"
linha 2 "Pedro"  "Ana"
```

A mesma lógica se aplica aos data frames:

```
df[-3,]
```

```
        coluna1 coluna2
linha 1   Maria   Paulo
linha 2   Pedro     Ana
```

Caso a intenção seja remover múltiplas linhas, então o número das linhas que serão removidas deve ser indicado em um vetor precedido do sinal de menos dentro do colchetes. Por exemplo, caso queiramos remover as linhas 2 e 3 da matriz de nomes, devemos proceder conforme a seguir:

```
nomes[-c(2,3),]
```

```
coluna 1 coluna 2
 "Maria"  "Paulo"
```

A mesma lógica se aplica aos data frames:

```
df[-c(2,3),]
```

```
        coluna1 coluna2
linha 1   Maria   Paulo
```

2.5 Adicionando colunas em um data frame

Existem várias maneiras de adicionar colunas em um data frame. O usuário pode indicar a posição da coluna entre colchetes e em seguida indicar os valores, pode indicar o nome da coluna em colchetes e em seguida indicar os valores, pode usar o sifrão para indicar o nome da coluna criada, ...

Para exemplificar, suponha que queiramos adicionar uma coluna com os valores *[pessoa 1, pessoa 2, pessoa 3] no dtaframe df. Essa tarefa pode ser feita das seguintes maneiras:

- Indicando a posição em colchetes

```
df[,3] = c("pessoa 1", "pessoa 2", "pessoa 3")
print(df)
```

```
        coluna1 coluna2       V3
linha 1   Maria   Paulo pessoa 1
linha 2   Pedro     Ana pessoa 2
3          João   Clara pessoa 3
```

- Indicando o nome da coluna no colchetes.

```
df[, "coluna3"] = c("pessoa 1", "pessoa 2", "pessoa 3")
print(df)
```

```
        coluna1 coluna2  coluna3
linha 1   Maria   Paulo pessoa 1
linha 2   Pedro     Ana pessoa 2
3          João   Clara pessoa 3
```

```
df=df[,-3]
```

- Usando o cifrão.

```
df$coluna3 = c("pessoa 1", "pessoa 2", "pessoa 3")
print(df)
```

```
        coluna1 coluna2  coluna3
linha 1   Maria   Paulo pessoa 1
linha 2   Pedro     Ana pessoa 2
3          João   Clara pessoa 3
```

2.6 Adicionando colunas em uma matriz

Para adicionar colunas em uma matriz é preciso usar a função *cbind()*. Essa função tem a mesma lógica de uso da função *rbind()* apresentada anteriormente, com a diferença de que a função *cbind()* posiciona os novos elementos em uma coluna em vez de uma linha. Para exemplificar, vamos adicionar a mesma coluna incluída no data frame *df* na matriz de nomes.

```
nomes = cbind(nomes, c("pessoa 1", "pessoa 2", "pessoa 3"))
print(nomes)
```

```
        coluna 1 coluna 2
linha 1 "Maria"  "Paulo"  "pessoa 1"
linha 2 "Pedro"  "Ana"    "pessoa 2"
        "João"   "Clara"  "pessoa 3"
```

A mesma lógica pode ser aplicada aos data frames fazendo *df = cbind(df, c("pessoa 1", "pessoa 2", "pessoa 3"))*.

2.7 Renomeando colunas e linhas específicas

Como visto no capítulo anterior, a função *colnames()* pode ser usada para renomear colunas de uma matriz ou data frame. Porém, caso o objetivo seja renomear uma única coluna específica é possível usar a indexação para realizar essa tarefa. Por exemplo, considere renomear apenas a coluna 3 do data frame de nomes atribuindo a essa coluna o nome *"coluna3"*.

```
colnames(nomes)[3] = "Coluna3"
print(nomes)
```

```
        coluna 1 coluna 2 Coluna3
linha 1 "Maria"  "Paulo"  "pessoa 1"
linha 2 "Pedro"  "Ana"    "pessoa 2"
        "João"   "Clara"  "pessoa 3"
```

O mesmo pode ser feito para as linhas. Por exemplo, caso queiramos renomear apenas a linha 3 do data frame de nomes, podemos proceder conforme a seguir:

```
rownames(nomes)[3] = "linha 3"
print(nomes)
```

```
        coluna 1 coluna 2 Coluna3
linha 1 "Maria"  "Paulo"  "pessoa 1"
linha 2 "Pedro"  "Ana"    "pessoa 2"
linha 3 "João"   "Clara"  "pessoa 3"
```

2.8 Operações com colunas

Em um data frame novas colunas podem ser geradas por meio de operações com colunas existentes. Para exemplificar, considere um dataframe com informações sobre medidas de alunos de uma academia.

```
alunos = data.frame(
nome = c("Aluno 1", "Aluno 2", "Aluno 3"),
peso = c(65, 70, 90),
altura = c(1.60, 1.70, 1.78)
)
print(alunos)
```

```
     nome peso altura
1 Aluno 1   65   1.60
2 Aluno 2   70   1.70
3 Aluno 3   90   1.78
```

Imagine que seja necessário adicionar nesse data frame uma nova coluna com o índice de massa corporal dos alunos (IMC). Sabe-se que:

$$IMC = \frac{Peso}{Altura^2}$$

A nova coluna pode ser adicionada indexando a sua posição:

```
alunos[,4] = alunos[,2]/(alunos[,3]^2)
print(alunos)
```

```
     nome peso altura       V4
1 Aluno 1   65   1.60 25.39062
2 Aluno 2   70   1.70 24.22145
3 Aluno 3   90   1.78 28.40550
```

Indexando o nome da coluna:

```
alunos["imc"] = alunos["peso"]/(alunos["altura"]^2)
print(alunos)
```

```
     nome peso altura      imc
1 Aluno 1   65   1.60 25.39062
2 Aluno 2   70   1.70 24.22145
3 Aluno 3   90   1.78 28.40550
```

Ou utilizando o cifrão:

```
alunos$imc = alunos$peso/(alunos$altura^2)
print(alunos)
```

```
     nome peso altura      imc
1 Aluno 1   65   1.60 25.39062
2 Aluno 2   70   1.70 24.22145
3 Aluno 3   90   1.78 28.40550
```

2.8.1 Criando uma nova coluna como recorte de uma coluna existente

Imagine o caso em que seja preciso criar uma coluna como um recorte de valores de uma coluna existente em um data frame. Falando com outras palavras, imagine que seja preciso adicionar uma coluna contendo uma parte dos valores contidos em outra coluna. Por exemplo, no data frame de alunos da academia, imagine que seja preciso mostrar apenas o número do aluno em uma nova coluna, isto é, em vez de mostrar o termo "Aluno 1" suponha que seja preciso mostrar apenas o número "1".

Esse procedimento pode ser feito usando a função *substr*. Essa função é nativa da linguagem *R* e serve para desmembrar valores de acordo com a posição dos caracteres desses valores. Exemplificando, o termo "Aluno 1" tem sete caracteres que correspondem a seis letras, um espaço e um número. O número ocupa a sétima posição no termo. Na função substr temos que indicar a posição inicial e a posição final do conjunto de caracteres que queremos desmembrar do valor objetivo. A maneira correta de usar essa função é fazendo *substr(nome do elemento, posição inicial, posição final)*. Por exemplo, se fizermos *substr("Aluno 1", 1, 2)* o resultado será *"Al"* que corresponde aos dois primeiros caracteres do termo utilizado.

No nosso exemplo como o número do aluno inicia e termina na sétima posição, então o correto a se fazer é:

```
alunos$num_aluno = substr(alunos$nome, 7, 7)
print(alunos)
```

```
     nome peso altura      imc num_aluno
1 Aluno 1   65   1.60 25.39062         1
2 Aluno 2   70   1.70 24.22145         2
3 Aluno 3   90   1.78 28.40550         3
```

2.8.2 Mesclando colunas

Em vez de separar valores de uma coluna, imagine o caso em que seja preciso juntar valores. Em *R* isso pode ser facilmente feito usando a função *paste()*. Essa função junta múltiplos valores separando-os com um searador indicado pelo usuário. Para usar a função, basta indicar entre parênteses os valores que serão unificados e indicar o separador com o parâmetro *"sep"*. Por exemplo:

```
nome = "João"
sobrenome = "Silva"
nome_completo = paste(nome, sobrenome, sep = " ")
print(nome_completo)
```

```
[1] "João Silva"
```

Essa função pode ser usada para unir colunas. Para exemplificar, considere o novo data frame com as características dos alunos de uma academia indicado a seguir:

```
alunos2 = alunos
alunos2$nome = c("João", "Maria", "João")
```

E imagine que seja preciso criar um código do aluno, unindo o nome e o número do aluno separando o nome e um número por um traço. Esse procedimento pode ser feito conforme indicado a seguir:

```
alunos2$cod_aluno = paste(alunos2$nome, alunos2$num_aluno, sep = "-")
print(alunos2)
```

```
   nome peso altura      imc num_aluno cod_aluno
1  João   65   1.60 25.39062         1    João-1
2 Maria   70   1.70 24.22145         2   Maria-2
3  João   90   1.78 28.40550         3    João-3
```

2.9 Reposicionando linhas e colunas

Conhecendo a posição de cada linha e cada coluna, é possível reordená-las de acordo com as necessidades ou preferências do usuário usando operações indexadas. Por exemplo, suponha que seja preciso reordenar as linhas do data frame de alunos para posicionar o aluno 3 na primeira linha, o aluno 2 na segunda linha e o aluno 1 na terceira linha. Esse procedimento pode ser feito conforme a seguir:

```r
alunos = alunos[c(3,2,1),]
print(alunos)
```

```
     nome peso altura      imc num_aluno
3 Aluno 3   90   1.78 28.40550         3
2 Aluno 2   70   1.70 24.22145         2
1 Aluno 1   65   1.60 25.39062         1
```

Agora suponha que seja necessário posicionar o nome do aluno na primeira coluna, a altura na segunda coluna, o peso na terceira coluna e o imc na última coluna, isto é, suponha que seja preciso inverter a posição do peso e da altura no data frame. Esse procedimento pode ser feito conforme indicado a seguir:

```r
alunos = alunos[, c(1,3,2,4)]
print(alunos)
```

```
     nome altura peso      imc
3 Aluno 3   1.78   90 28.40550
2 Aluno 2   1.70   70 24.22145
1 Aluno 1   1.60   65 25.39062
```

2.10 Operações básicas

Novas linhas e colunas podem ser adicionadas em um data frame usando operações básicas como soma, produto, ou estatísticas básicas. Para demonstrar, considere um data frame contendo a idade de cinco pessoas conforme a seguir:

```r
df = data.frame(
  pessoa = c("pessoa1", "pessoa2", "pessoa3", "pessoa4", "pessoa"),
  idade = c(25, 50, 68,45,NA)
)
print(df)
```

```
   pessoa idade
1 pessoa1    25
2 pessoa2    50
3 pessoa3    68
4 pessoa4    45
5  pessoa    NA
```

Note que há um valor faltante, indicado pelo termo *NA* que representa a sigla do "não disponível" (do inglês *not available*). Qualquer operação contendo essa coluna deve indicar que esse valor ausente deve ser ignorado. Isso pode ser feito adicionando a opção *na.rm = TRUE*.

2.10.1 Soma total

Suponha que queiramos encontrar o somatório da idade de todas as pessoas contidas no data frame. Isso pode ser facilmente executado usando a função *sum()*, conforme indicado a seguir:

```
sum(df$idade, na.rm = TRUE)
```

```
[1] 188
```

2.10.2 Produto total

Caso o objetivo seja encontrar o produto de todos os valores de uma dada coluna, então o ideal é usar a função *prod()*, conforme demonstrado a seguir:

```
prod(df$idade, na.rm = TRUE)
```

```
[1] 3825000
```

2.10.3 Média

Já a média pode ser obtida com a função *mean()*, conforme demonstrado a seguir:

```
mean(df$idade, na.rm = TRUE)
```

```
[1] 47
```

2.10.4 Mínimo

O valor mínimo de um dado objeto pode ser computado por meio da função *min()*, conforme demonstrado a seguir:

```r
min(df$idade, na.rm = TRUE)
```

```
[1] 25
```

2.10.5 Máximo

O valor máximo de um dado objeto pode ser computado por meio da função *max()*, conforme demonstrado a seguir:

```r
max(df$idade, na.rm = TRUE)
```

```
[1] 68
```

2.10.6 Desvio padrão

O desvio padrão de um dado objeto pode ser computado por meio da função *sd()*, conforme demonstrado a seguir:

```r
sd(df$idade, na.rm = TRUE)
```

```
[1] 17.68238
```

2.10.7 Número de observações

O número de observações - ou comprimento - de um dado objeto pode ser verificado com o uso da função *length()*, conforme demonstrado a seguir:

```r
length(df$idade)
```

```
[1] 5
```

Note que nesse caso é preciso despresar a omissão dos valores *NA*.

2.10.8 Exemplo: Criando uma tabela de estatísticas descritivas

Uma tabela de estatísticas descritivas mostra o perfil básico de um banco de dados e geralmente expressa o número de observações, o valor médio, o desvio padrão, o valor máximo e o valor mínimo de cada variável. Para representar, considere usar o banco de dados nativo do *R* sobre características das flores (*iris)*. Esse banco de dados possui cinco colunas (*Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species*) que mostram o comprimento e a largura da pétala e da sépala de cada espécie de flor. Elaborar uma tabela de estatísticas descritivas dessa base de dados seria o mesmo que preencher a seguinte tabela:

	Observações	Média	Desvio Padrão	Mínimo	Máximo
Sepal.Length					
Sepal.Width					
Petal.Length					
Petal.Width					

Para esse propósito, vamos usar a indexação aos nomes das linhas e colunas e vamos usar as estatísticas básicas mostradas anteriormente. O primeiro passo para tal é criar um data frame vazio com os mesmos padrões da tabela anterior:

```
est_desc = data.frame(
  Observacoes = c(rep(NA, 4)),
  Media = c(rep(NA, 4)),
  Desvio_Padrao = c(rep(NA, 4)),
  Minimo = c(rep(NA, 4)),
  Maximo  = c(rep(NA, 4))
)
rownames(est_desc) = c("Sepal.Length", "Sepal.Width", "Petal.Length", 'Petal.Width')
print(est_desc)
```

```
             Observacoes Media Desvio_Padrao Minimo Maximo
Sepal.Length          NA    NA            NA     NA     NA
Sepal.Width           NA    NA            NA     NA     NA
Petal.Length          NA    NA            NA     NA     NA
Petal.Width           NA    NA            NA     NA     NA
```

O próximo passo é preencher cada célula do data frame com as estatísticas correspondentes. Inicialmente, vamos preencher a coluna referente ao número de observações:

```
est_desc["Sepal.Length", "Observacoes"] = length(iris$Sepal.Length)
est_desc["Sepal.Width", "Observacoes"] = length(iris$Sepal.Width)
est_desc["Petal.Length", "Observacoes"] = length(iris$Petal.Length)
est_desc["Petal.Width", "Observacoes"] = length(iris$Petal.Width)
```

Agora vamos preencher a coluna da média:

```
est_desc["Sepal.Length", "Media"] = mean(iris$Sepal.Length)
est_desc["Sepal.Width", "Media"] = mean(iris$Sepal.Width)
est_desc["Petal.Length", "Media"] = mean(iris$Petal.Length)
est_desc["Petal.Width", "Media"] = mean(iris$Petal.Width)
```

Agora vamos preencher a coluna do desvio padrão:

```
est_desc["Sepal.Length", "Desvio_Padrao"] = sd(iris$Sepal.Length)
est_desc["Sepal.Width", "Desvio_Padrao"] = sd(iris$Sepal.Width)
est_desc["Petal.Length", "Desvio_Padrao"] = sd(iris$Petal.Length)
est_desc["Petal.Width", "Desvio_Padrao"] = sd(iris$Petal.Width)
```

Agora vamos fazer o mesmo para a coluna do valor mínimo:

```
est_desc["Sepal.Length", "Minimo"] = min(iris$Sepal.Length)
est_desc["Sepal.Width", "Minimo"] = min(iris$Sepal.Width)
est_desc["Petal.Length", "Minimo"] = min(iris$Petal.Length)
est_desc["Petal.Width", "Minimo"] = min(iris$Petal.Width)
```

Por fim, vamos preencher a coluna do valor máximo:

```
est_desc["Sepal.Length", "Maximo"] = max(iris$Sepal.Length)
est_desc["Sepal.Width", "Maximo"] = max(iris$Sepal.Width)
est_desc["Petal.Length", "Maximo"] = max(iris$Petal.Length)
est_desc["Petal.Width", "Maximo"] = max(iris$Petal.Width)
```

O resultado desse procedimento é o data frame a seguir:

```
print(est_desc, digits = 4)
```

```
             Observacoes Media Desvio_Padrao Minimo Maximo
Sepal.Length         150 5.843        0.8281    4.3    7.9
Sepal.Width          150 3.057        0.4359    2.0    4.4
Petal.Length         150 3.758        1.7653    1.0    6.9
Petal.Width          150 1.199        0.7622    0.1    2.5
```

Existem maneiras mais rápidas e mais eficientes de preparar uma tabela de estatísticas descritivas, porém, isso exige artifícios que só serão vistos em capítulos posteriores.

2.11 Exercício

Considere usar a base de dados nativa sobre carros *mtcars*.

```
df = mtcars
```

(1) Remova o carro Fiat X1-9.

(2) Usando o nome das linhas e das colunas, encontre o consumo *(mpg)* do Toyota Corolla.

(3) Crie uma nova coluna de nome *difCorolla* mostrando a diferença entre o consumo médio de cada carro e o consumo médio do Toyota Corolla.

(4) Crie uma nova coluna de nome *consumo_peso* mostrando o consumo *(mpg)* dos carros para cada tonelada de peso *(wt)*.

(5) Apague os carros de câmbio manual (carros com $am = 0$).

3 Classes de elementos

Os dados podem assumir diversos formatos, por exemplo, números, textos, imagens, etc. A depender da ocasião, o usuário precisará lidar com um tipo de dado específico. Por exemplo, um economista prevendo preços no mercado de ações utilizará objetos numéricos para representar o preço dos ativos. Um técnico em radiologia coleta informações em imagem de um paciente. Um especialista em análise de sentimento em redes sociais coleta os textos das postagens para extrair delas informações úteis. Cada tipo de informação tem uma utilidade e um propósito e cada tipo de informação possui propriedades que o analista de dados precisa conhecer.

Um objeto criado possui múltiplos elementos e esses elementos podem ser de diferentes tipos. Esses *"tipos"* de elementos recebem o nome de *classe*. Existem operações que são aplicadas adequadamente a uma classe de elementos específica, por exemplo, é possível somar dois elementos numéricos mas é estranho pensar em somar duas palavras. Nesse capítulo vamos conhecer as principais classes de dados, suas características e as principais operações que podem ser realizadas com cada classe.

3.1 Elementos textuais: (*strings*)

Os elementos textuais sempre devem ser declarados entre aspas, sejam aspas simples ou duplas, do contrário, a linguagem *R* não reconhecerá esse elemento como texto. A linguagem *R* não suporta operações matemáticas com elementos textuais, ao contrário de outras linguagens como *Python* onde operações matemáticas básicas podem ser aplicadas a esses elementos.

Imagine que seja necessário criar um data frame com o nome dos estados e os nomes das capitais da região Sul do Brasil. Nesse caso, os elemento são textuais e cada nome de estado e capital deve ser informado entre aspas conforme demonstrado a seguir:

```
sul = data.frame(
  estados = c("Paraná", "Santa Cararina", "Rio Grande do Sul"),
  capital = c("Curitiba", "Florianópolis", "Porto Alegre")
)
print("Sul")
```

```
[1] "Sul"
```

Caso os elementos não sejam declarados entre aspas, a execução do comando retornará um erro.

É possível checar qual a classe de um dado elemento ou de um dado objeto por meio do comando *class()*, informando o nome do objeto entre parêntesis. Por exemplo, é possível checar se o objeto criado anteriormente com o nome *sul* é um data frame, um vetor, um array, etc.

```
class(sul)
```

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

O comando *class()* também pode ser usado para verificar se um objeto possui ou não uma classe textual. Caso se trate de um objeto textual, então o comando *class()* retornará o output *"character"*, indicando que se trata de um caracter. Por exemplo, vamos verificar a classe da coluna de nome *estados* do objeto de nome *sul*:

```
class(sul$estados)
```

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

Elementos númericos ou outros tipos de elementos muitas vezes podem ser reconhecidos, declarados ou transformados em caracteres. Nesse caso, cabe ao usuário saber identificar a classe dos elementos e transformá-lo na classe desejada. Para exemplificar, considere incluir no data frame anterior o código de unidade federativa do ibge:

```
sul$codigo_uf = c(41, 42, 43)
print(sul)
```

```
            estados       capital codigo_uf
1            Paraná      Curitiba        41
2    Santa Cararina Florianópolis        42
3 Rio Grande do Sul  Porto Alegre        43
```

Note que os números não estão em aspas, indicando que não são caracteres. Abrindo o data frame, o usuário verá facilmente que as duas primeiras colunas estão com os valores alinhados à esquerda, enquanto a última coluna está alinhada à direita. Nesse caso, apenas visualizando esse alinhamento é possível afirmar que as duas primeiras colunas têm elementos textuais. No entanto, o usuário pode checar se as colunas são de fato *strings* usando o comando *is.character()*, indicando o nome do objeto ou elemento no parênteses. Por exemplo, para checar se o código do estado é um elemento textual, proceda conforme a seguir:

```
is.character(sul$codigo_uf)
```

```
[1] FALSE
```

Note que o output foi *FALSE*, indicando que não se trata de um elemento textual. Caso o usuário precise dessa coluna como um elemento de texto, então ele pode usar o comando *as.character()*, indicando o nome do objeto ou elemento no parênteses. Nesse caso, o objeto ou elemento indicado será transformado forçadamente em um elemento textual. Para exemplificar, vamos transformar o código do estado em um caracter:

```
sul$codigo_uf = as.character(sul$codigo_uf)
```

Checando agora a classe da coluna, nota-se que se trata de um elemento textual:

```
class(sul$codigo_uf)
```

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

Existem operações no âmbito da análise de dados que são usadas com esse tipo de elemento. Por exemplo, as análises de sentimento baseiam-se fundamentalmente em coletas de dados de texto. Um exemplo factível é a mensuração da sensibilidade do mercado financeiro em relação a intensidade do texto da ata do banco central. Esse procedimento é feito pela contagem de palavras pré-definidas na ata. Para esse tipo de operação existem bibliotecas que auxiliam o usuário e que serão vistas posteriormente quando estivermos abordando operações com *strings*.

3.2 Elementos numéricos

Os elementos numéricos são todos os elementos representados na forma de números, sejam eles de quaisquer um dos conjuntos numéricos. Por exemplo, 2 é um elemento numérico do conjunto dos naturais inteiros, ao passo que 2.5 é um elemento numérico do conjunto dos números racionais. É possível checar se um elemento é numérico usando o comando *class()*, no entanto, é mais adequado usar o *is.numeric()*, indicando o nome do elemento em parênteses. Para exemplificar, vamos checar se a coluna *codigo_uf* do data frame de nome *sul* é numérico.

```
is.numeric(sul$codigo_uf)
```

```
[1] FALSE
```

Note que o resultado é *FALSE*, indicando que não se trata de um elemento numérico. Caso o usuário precise transformar essa coluna em um elemento numérico, então ele pode usar o comando *as.numeric()*, indicando o nome do elemento em parêntesis:

```
sul$codigo_uf = as.numeric(sul$codigo_uf)
```

Agora podemos verificar novamente se a coluna é ou não numérica.

```
is.numeric(sul$codigo_uf)
```

```
[1] TRUE
```

Em um data frame, as colunas numéricas sempre estarão alinhadas à direita e o usuário pode checar a classe dessa coluna posicionando o cursor sobre o seu nome no data frame.

Existe, contudo, um problema associado ao uso do comando *as.numeric()*, dado que a transformação de um elemento não numérico para um elemento numérico só funciona perfeitamente caso o objeto não numérico seja composto por números declarados como caracteres, como é o caso da coluna *codigo_uf*. Do contrário, o resultado pode ser composto por *NAs*. Para exemplificar, vamos tentar transformar o nome dos estados em um objeto numérico:

```
as.numeric(sul$estados)
```

```
Warning: NAs introduzidos por coerção
```

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

Por esse motivo, antes de efetuar qualquer operação com um elemento numérico, é fundamental checar a sua classe para garantir que o resultado esteja de acordo com o esperado. Por exemplo, imagine o caso em que seja preciso calcular o pib per capita de um estado, sendo que o pib seja numérico e a população seja um caracter. Nesse caso, se o usuário não conhece a classe da população e não a transforma em um elemento numérico, o cálculo do pib per capita será feito dividindo um número por uma palavra, o que é algo inviável, resultando em um elemento do tipo *NA*.

3.2.1 Subclasses dos numéricos: inteiros *(integer)*

Como mencionado anteriormente, os elementos numéricos podem estar contidos em quaisquer um dos conjuntos numéricos. Uma subclasse bastante comum na economia são é a subclasse dos inteiros. Essa subclasse abriga os números inteiros positivos ou negativos e se caracteriza por abrigar um *L* após o número. Por exemplo, 10 é um elemento da classe numérica, mas 10L é um elemento da subclasse dos inteiros pertencente à classe dos numéricos.

Para checar se um elemento é inteiro, use o comando *is.integer()*, indicando o elemento no parêntesis. Por exemplo, vamos checar se 10 é inteiro:

```
is.integer(10)
```

```
[1] FALSE
```

Note que o resultado é *FALSE*, indicando que 10 não é inteiro. Isso ocore porque os números inteiros devem obrigatoriamente ser sucedidos da letra *L*. Informando corretamente tem-se:

```
is.integer(10L)
```

```
[1] TRUE
```

Para transformar um elemento numérico na subclasse dos inteiros, basta usar a função *as.integer()*, informando o elemento no parênteses. Por exemplo, vamos transformar o código do estado no data frame de nome *sul* em inteiro.

```
sul$codigo_uf = as.integer(sul$codigo_uf)
print(sul)
```

```
            estados       capital codigo_uf
1            Paraná      Curitiba        41
2    Santa Cararina Florianópolis        42
3 Rio Grande do Sul  Porto Alegre        43
```

Note que visivelmente não há mudanças nas propriedades da coluna modificada, porém agora quando consultarmos se essa coluna pertence ao conjunto dos inteiros, o output é verdadeiro:

```
is.integer(sul$codigo_uf)
```

```
[1] TRUE
```

3.2.2 Subclasses dos numéricos: racionais *(doubles)*

Os números racionais são declarados como *doubles* na linguagem *R*. Esse conjunto abriga também os inteiros e os naturais, isto é, um inteiro sempre será um *double,* assim como um natural sempre será um *double*. Para checar se um número pertence a essa categoria, use a função *is.double()*, indicando o elemento de interesse no parênteses.

```
is.double(10.5555)
```

```
[1] TRUE
```

De maneira análoga, um elemento numérico declarado como texto pode ser trannsformado em racional usando a função *as.double()*, indicando o elemento de interesse no parênteses.

```
as.double("10.5")
```

```
[1] 10.5
```

Em alguns casos, os números racionais possuem múltiplas casas decimais e é preciso reduzir essas casas decimais arredondando a última casa. Isso pode ser facilmente resolvido usando a função *round()*, que possui o seguinte modo de uso: *round(nome do objeto ou elemento, número de casas decimais)*. Por exemeplo, imagine que queiramos reduzir o número 5.56413 para apenas uma casa decimal. Nesse caso, devemos proceder conforme a seguir:

```
round(5.564132, 1)
```

```
[1] 5.6
```

3.3 Elementos lógicos: *(TRUE e FALSE)*

Os elementos lógicos podem assumir dois valores, verdadeiro *(TRUE)* ou falso *(FALSE)*. Sempre que o elemento lógico for verdadeiro, o *R* atribui valor 1 a este elemento, ao passo que sempre que o elemento lógico for falso, o *R* atribui valor 0 a este elemento. Assim, é possível aplicar operações matemáticas aos elementos pertencentes a essa classe. Isso advém da premissa de que as linguagens de programação são baseados em sistemas binários de afirmação e negação comentada no capítulo 1.

Essa classe de elementos é bastante utilizada na economia para representar variáveis binárias onde a categoria de interesse recebe o valor unitário. Por exemplo, imagine uma pesquisa com

foco no diferencial de salário por sexo. Nesse tipo de pesquisa é ideal saber se o indivíduo é homem ou mulher. Se a categoria de interesse for o sexo masculino, então os homens recebem valor *TRUE* e as mulheres recebem vaor *FALSE*. Isso equivale a atribuir 1 para os homens e 0 para as mulheres.

Para verificar se um elemento pertence a essa categoria, use a função *is.logical()*, indicando o nome do elemento entre o parênteses. Para exemplificar, vamos criar uma nova coluna no data frame de estados da regiao sul com o nome *parana* que identifica se o estado em questão é ou não o estado do Paraná.

```
sul$parana = c(1,0,0)
print(sul)
```

```
            estados       capital codigo_uf parana
1            Paraná      Curitiba        41      1
2    Santa Cararina Florianópolis        42      0
3 Rio Grande do Sul  Porto Alegre        43      0
```

Agora vamos checar se essa coluna é um elemento lógico.

```
is.logical(sul$parana)
```

```
[1] FALSE
```

Note que o output é *FALSE*, indicando que não se trata de um elemento lógico, o que é esperado dado que se trata de um elemento numérico. Para transformar essa coluna em um elemento lógico, basta usar a função *as.logical()*, indicando o nome do elemento de interesse no parênteses.

```
sul$parana = as.logical(sul$parana)
print(sul)
```

```
            estados       capital codigo_uf parana
1            Paraná      Curitiba        41   TRUE
2    Santa Cararina Florianópolis        42  FALSE
3 Rio Grande do Sul  Porto Alegre        43  FALSE
```

Agora vamos checar novamente se essa coluna é um elemento lógico.

```r
is.logical(sul$parana)
```

```
[1] TRUE
```

Note que o output é *TRUE*, indicando que se trata de um elemento lógico.

3.4 Valores multicategóricos: Fatores (*factors*)

Os *factors* são elementos usados para representar valores multicategóricos. Por exemplo, imagine uma variável que expressa a situação do empregado no mercado de trabalho. Ele pode estar (1) apenas trabalhando, (2) apenas estudando, (3) trabalhando e estudando, (4) nem trabalhando nem estudando porém procurando emprego, ou (5) nem trabalhando nem estudando nem procurando emprego. Note que são cinco possibilidades que agora não podem ser representadas pelos elementos lógicos.

Para declarar um elemento multicategórico é necessário usar o comando *factor()* que tem a seguinte forma de uso: *factor(x = elemento, levels = níveis das categorias)*. Para exemplificar, vamos criar um objeto de nome *emprego* com as possibilidades indicadas no parágrafo anterior e os seus respectivos valores.

```r
emprego = factor(
  x = c(
    "apenas trabalhando",
    "apenas estudando",
    "trabalhando e estudando",
    "nem trabalhando nem estudando porém procurando emprego",
    "nem trabalhando nem estudando nem procurando emprego"
  ),
  levels = c(
    "apenas trabalhando",
    "apenas estudando",
    "trabalhando e estudando",
    "nem trabalhando nem estudando porém procurando emprego",
    "nem trabalhando nem estudando nem procurando emprego"
  )
)

print(emprego)
```

```
[1] apenas trabalhando
[2] apenas estudando
[3] trabalhando e estudando
[4] nem trabalhando nem estudando porém procurando emprego
[5] nem trabalhando nem estudando nem procurando emprego
5 Levels: apenas trabalhando apenas estudando ... nem trabalhando nem estudando nem procuran
```

Será atribuído valor 1 para a primeira categoria indicada no vetor de níveis *(levels)*, 2 para a segunda categoria e assim sucessivamente.

Para checar se um elemento é multicategórico, devs-se usar a função *is.factor()*, indicando o nome do elemento no parênteses.

```
is.factor(emprego)
```

```
[1] TRUE
```

3.5 Exercício 1

Considere o seguinte data frame:

```
set.seed(10)
dados = data.frame(
  pessoa = 1:30,
  idade = sample(8:85,30, replace = T),
  sexo = sample(c("M", "F"), 30, replace = T),
  estado_civil = sample(c("Solteiro", "Casado", "Viúvo", "Divorciado"), 30, replace = T),
  salario = rnorm(30, mean = 1200, sd = 300)
)
```

(1) Crie uma variável de nome *sexo2* transformando a variável *sexo* em um elemento lógico atribuindo o valor unitário para as mulheres.

(2) Crie uma variável de nome *fase_vida* atribuindo os nomes *infância* para as pessoas com menos de 12 anos, *adolescência* para as pessoas com idade entre 12 e 18 anos, *adulta* para as pessoas com idade entre 18 e 65 anos e *velhice* para as pessoas com mais de 65 anos.

(3) Crie uma nova variável de nome *fase_vida2* transformando a variável *fase_vida* em um factor ordenando as categorias de acordo com a fase da vida em ordem crescente.

3.6 Exercício 2

Considere a base de dados sobre carros *mtcars*:

```
carros = mtcars
```

(1) Crie uma coluna de nome *automatico* transformando a coluna *am* em um elemento lógico.

(2) Transforme a coluna *cyl* em um factor onde o atributo *x* recebe os valores *"4 cilindros", "6 cilindore"* e *"8 cilindros*" ordenados na forma crescente.

(3) Crie uma nova coluna de nome *carro* contendo o nome dos carros indicados nos nomes das linhas do data frame.

(4) Verifique a casse da coluna criada na questão.

(5) Verifique se a coluna criada nas questões 1 e 2 são *factors*.

(6) Verifique se a coluna *mpg* é inteiro e caso não seja transforme-a em inteiro.