Introdução à linguagem R

Jean S. S. Resende, Jéssica M. Magno, João C. D. Muzzi, Mauro A. A. Castro

2023-06-04

Contents

Chapter 1

Prefácio

A linguagem R foi criada por professores do departamento de estatística da univer sidade de Auckland no ano 2000. A intenção destes professores era disponibilizar uma linguagem *open-source* para computação estatística. Com o avanço e popularização da linguagem, profissionais de diversas áreas passaram a utilizá-la em suas análises de dados.

Esta apostila contém uma introdução à linguagem de programação R. Aborda desde conteúdos teóricos quanto práticos, com exemplos didáticos a fim de facilitar o aprendizado. Esta apostila foi produzida principalmente pelos autores: Jean Silva de Souza Resende, Jéssica Maria Magno, João Carlos Degram Muzzi sob orientação de Mauro Antônio Alves Castro. Em versões anteriores, tivemos a colaboração dos autores: Sheyla Trefflich, Danrley R. Fernandes e Giuseppe Pasqualato Neto.

Chapter 2

Introdução

2.1 Contextualizando a linguagem de programação R

A linguagem R é uma linguagem de programação com o foco em computação estatística e manipulação de gráficos. Criada no início dos anos 90 por Geroge Ross Ihaka e Robert Clifford Gentleman, o R é usado mais utilizado por estatísticos, bioinformatas, analistas de dados e desenvolvedor de *software* estatístico. No entanto ele tem se destacado na comunidade científica. Em maio de 2023, o R ocupava a 16ª posição no índice TIOBE, uma medida de popularidade da linguagem de programação, sendo que em agosto de 2020 o R atingiu seu pico em ficando 8º lugar.

O R é um ambiente de *software* livre de código aberto, disponível sob a *GNU General Public License*. Seus executáveis pré-compilados são fornecidos para vários sistemas operacionais. Ele tem uma interface de linha de comando, mas também possui interfaces gráficas de usuário (GUI) de terceiros como o Rstudio - que será a IDE (*Integrated Development Envirenment*) que iremos utilizar na apostila.

2.2 Instalação do R

1. Acesse o repositório do R (clique aqui).
2. Acesse o link referente ao seu sitema operacional: Linux, macOS ou Windows. 2.1. Linux: escolha a distribuição linux (debian, fedora, redhat, suse ou ubuntu) e então prossiga com os comandos no terminal. 2.2. macOS: escolha o instalador conforme o modelo da sua máquina e execute-o. 2.3. Windows: acesse o link Base e então baixe o instalador e execute-o.

2.3 Instalação do RStudio

1. Clique aqui para acessar o repositório do RStudio.
2. Baixe o instalador conforme o sistema operacional da sua máquina (Linux/macOS/Windows).

2.4 Interface do RStudio

Por padrão o RStudio abre quatro janelas (pode ocorrer de uma estar oculta, mas observe o botão de minimizar/maximizar no canto superior direito de cada janela).

- **Editor de código** (*janela do canto superior esquerdo*): Nela você digita os comando a serem executados no RStudio. Para executá-los aperte as teclas 'CTRL' e 'ENTER' simultaneamente na linha ou bloco de código selecionado.
- **Console** (*janela do canto inferior esquerdo*): É visto as saídas dos comandos que são rodados. Também é possível digitar e rodar códigos diretamente nesta janela.
- **Histórico** (*janela do canto superior direito*): Nesta janela ficam salvos os objetos, históricos de comandos e conexões com outros aplicativos.
- **Visualização** (*janela do canto inferior direito*): Aqui você pode visualizar os gráficos no RStudio, navegar entre os arquivos do seu computador, visualizar os pacotes instalados e ver a ajuda de comandos e descrições de tabelas de dados e por fim navegar entre os arquivos html.

2.5 Pacotes

Por ser *Open Source*, o R permite que qualquer usuário disponibilize funções e bancos de dados a comunidade. As funções/bancos de dados são disponibilizados através de pacotes. A instalação de um pacote depende do repositório que ele está armazenado: máquina local, CRAN, GitHub, Bioconductor, entre outros. O repostitório CRAN contêm muitos pacotes e não é direcionado à uma área específica (como é o caso do Bioconductor que se destina a pacotes voltados para área de biotecnologia). A instalação de um pacote do repositório CRAN é feita pelo menu "Tools > Install Packages" ou simplesmente utilizando o seguinte comando:

```
installed.packages("nomeDoPacote")
```

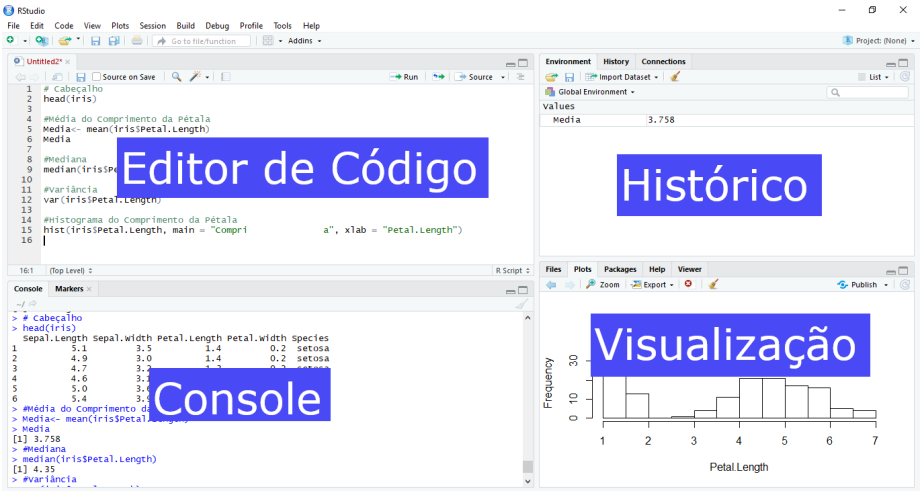

Figure 2.1: Janelas do RStudio. Ref: https://www.est.ufmg.br/~cristianocs/Pacotes2021/Intro.html#6

Para que você possa utilizar as funções do pacote que instalou, você deve usar um dos dois comandos a seguir, para de fato carregar as funções do pacote para o ambiente R:

```
library(nomeDoPacote)
require(nomeDoPacote)
```

A função `library()` é utilizada normalmente no corpo do script, enquanto que a função `require()` é utilizada dentro de outras funções.

2.6 Começando de fato a programar em R

No console (janela do canto inferior esquerdo) digite o comando a seguir e tecle ENTER:

```
print("Hello World")
```

Agora digite o mesmo comando no editor de código (janela do canto superior esquerdo) e com o cursor na mesma linha do comando, tecle CTRL e ENTER simultaneamente:

```
print("Hello World")
```

A diferença é que quando executamos os comandos no editor de código, o comando continua no editor para ser executado, ou seja svocê está construindo um script. Mas você executa comando diretamente no console, eles não ficam gravados em um editor.

O dado de saída da função, foi um print do que estava dentro da função. Mas como você saber o que usar dentro de uma determinada função, como `print()`? Você precisa acessar o manual desta função.

2.7 Acessando o manual da função

Esta é uma etapa muito importante que antecede a sua caminhada no aprendizado do R. Você pode visualizar o manual da função executando um comando onde um ponto de interregoção (?) precisa anteceder a função:

```
?print
```

Mas se você deseja encontrar funções que realizam uma determinada ação, basta inserir dois pontos de interrogação antecedendo a ação desejada:

```
??priting
```

O comando acima realizará uma busca por tópicos que contenham a palavra *plotting*. Outra opção alternativa ao **?** é o uso da função `help()` e `help.search()` para **??**

```
help("print")
help.search("priting")
```

Algumas funções possuem exemplos de sua execução. Se você quer saber como utilizar uma determinada função através de exemplos, execute a função `example()`.

```
example("print")
```

2.8 Comentando códigos no R

A maioria das linguagens de programação e até linguagem de marcação, possuem uma forma de inserção de textos que não serão executados pela linguagem. Esse procedimento é denominado de comentário. Você pode comentar os seus códigos. Isso é algo essencial para todos os programadores, indiferente da linguagem. Pois, códigos comentados facilitam a interpretação do mesmo por

outros programadores e até mesmo pelo autor, devido a um período de tempo que se passou desde a criação daquele código.

Para comentar linhas no R você precisa inserir o # antes do que seria o comentário:

```
# isto é um comentário
```

Exemplo aplicado:

```
print("Hello World") # imprimindo na tela Hello World
```

Perceba que o conteúdo após o # não é interpretado no R, ou seja, este conteúdo é um comentário.

Chapter 3

Fundamentos básicos da programação

3.1 Variáveis

Utilizamos a variável para armazenar um valor qualquer em um local da memória RAM do computador. Deste modo, é possível reutilizar esse valor, usando o nome da sua variável.

3.1.1 Declaração e atribuição de variáveis

Em R declaramos uma variável atribuindo a ela um valor em três formas diferente: **símbolo de atribuição <-**, **símbolo de atribuição =** e **função assign()**.

```
nome.var <- valor # atribuicao: menor e traco
nome.var = valor # atribuicao: igual
assign("nome_var",valor) # funcao: assign
```

3.1.2 Dicas para nomear variáveis

As variáveis podem ser nomeadas com o uso letras, números, ponto (.) e underline (_), no entanto é necessário se atentar para algumas dicas de como nomear as variáveis:

1. O nome da variável deve sempre começar com uma letra ou um ponto, ou seja, não pode iniciar com números ou símbolos. Se iniciar com ponto o próximo caracter não pode ser um número.

2. O nome da variável que contêm mais de uma palavra é recomendado o uso do underline (_) para separa-la.
3. O nome da variável não pode ser palavras reservadas da linguagem como TRUE, if, while, entre outras.
4. O nome da variável não pode conter espaços.
5. O nome da variável deve ser condizente com o seu valor.

3.1.3 Tipos de dados das variáveis

Em R o tipo de dado da variável é obtido a partir do valor atribuído à ela. Isto faz da linguagem R: **Linguagem dinamicamente tipada**. Pois, o tipo de dado de uma variável pode ser alterado dinamicamente enquanto o programa/script é executado.

As variáveis em R podem ser do tipo: inteiro (interger), ponto flutuante (double), complexo (complex), caracteres (character/string) e lógico (logical).

```
var_int <- 2L      # var integer
var_db1 <- 1.5     # var double
var_db2 <- 2       # var double
var_comp <- 2 + 3i # var complex
var_str <- "a_01"  # var string/character
var_log <- TRUE    # var logical
```

Podemos verificar o tipo das variáveis criadas no *chunk* anterior através da função `typeof()`.

```
typeof(var_int)
```

```
## [1] "integer"
```

```
typeof(var_db1)
```

```
## [1] "double"
```

```
typeof(var_db2)
```

```
## [1] "double"
```

```
typeof(var_comp)
```

```
## [1] "complex"
```

```
typeof(var_str)
```

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

```
typeof(var_log)
```

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

Para verificar quais variáveis o R está usando *workspace* usando a função `ls()`.

```
ls()
```

```
## [1] "var_comp" "var_db1"  "var_db2"  "var_int"  "var_log"  "var_str"
```

Para excluir variáveis, ou seja, desalocar determinada variável da memória RAM, basta usar a função `rm()`.

```
rm(var_str)     # desaloca a variavel var_str
rm(list = ls()) # desaloca todas as variaveis
```

3.2 Operações em R

Podemos executar operações matemáticas, lógicas e comparações em R. Para isso o R faz uso de **operadores**. Os operadores são divididos em: aritmético, relacional e lógico.

Os operadores aritméticos como o nome já diz são usados em operações aritméticas e são eles:

- Adição: +
- Subtração: -
- Multiplicação: *
- Divisão: /
- Resto de divisão: %%
- Divisão inteira: %/%
- Potenciação: ^

```
2+2   # soma
5-2   # subtracao
2*5   # multiplicacao
5/2   # divisao
5%%2  # resto de divisao
5%/%2 # divisao inteira
2^5   # potenciacao
```

Já os operadores relacionais, tratam da relação de um valor com o outro e são eles:

- Menor: <
- Maior: >
- Menor ou igual: <=
- Maior ou igual: >=
- Igual: ==
- Diferente: !=

```
2<5 # menor
2>5 # maior
2<=2 # menor ou igual
2>=5 # maior ou igual
5==5 # igual
2!=2 # diferente
```

Por fim, os operadores lógicos são:

- *logical NOT*: !
- *logical AND*: &
- *logical OR*: |

```
!TRUE # NOT = qual e o contrario de TRUE?
TRUE | FALSE # OR = um dos dois ou os dois é ou são verdadeiros?
TRUE & FALSE # AND = os dois são verdadeiros?
```

3.3 Condições e loops

Existem dois passos que são trilhados por toda linguagem de programação, e alguns progrmadores dizem que se uma linguagem de programação não permite a execução destes dois passos, ela não é bem considerada uma linguagem de programação. Um exemplo é a linguagem HTML, essa linguagem é dita como

linguagem de marcação sua finalidade é trabalhar com estruturação de textos. Não iremos utilizá-la para cálculos ou procedimentos que demandam de uma rotina computacional com base em cálculos e nos dois passos. Mas quais são estes dois passos? R: condições e *loops*.

3.3.1 Condições

Se alguma coisa for verdadeira (TRUE) o R vai agir de uma maneira, caso seja mentira (FALSE) ele vai agir de outra maneira. Você pode estabelecer algumas condições para que seja feita uma função.

3.3.1.1 Condição: if `if()`

Determinado código será executado somente se a condição for verdadeira, abaixo é apresentado a estrura do `if`.

```
# -- estrutura

# if(condicao){
#   comandos a serem executados
# }
```

Vamos agora fazer uma aplicação: se o número dois for maior que o número um, então imprima na tela a frase: dois é maior que um. Caso contrário não faça nada.

```
# -- aplicacao
## -- verdadeiro
if(2>1){
  print("dois é maior que um")
}
```

```
## [1] "dois é maior que um"
```

No exemplo abaixo a condição é falsa, logo o comando dentro de if não é executado.

```
## -- falso
if(2<1){
  print("dois é menor que um")
}
```

3.3.1.2 Condição: if else `if() else()`

Podemos querer que um comando seja executado se condição for verdadeira e outro comando seja executado se a condição for falsa. Faremos da seguinte forma:

```
if(TRUE){
  print("comando dentro do if")
}else{
  print("comando dentro do else")
}
```

```
## [1] "comando dentro do if"
```

Por ser verdadeira a condição dentro do `if`, foi executado o primeiro comando.

```
if(FALSE){
  print("comando dentro do if")
}else{
  print("comando dentro do else")
}
```

```
## [1] "comando dentro do else"
```

A condição dentro do `if` é falsa então foi executado o comando dentro do `else`

Outra forma de aplicar a condição if else é usando a função `ifelse()`.

```
ifelse(2 > 1, 2*1, 1/2) # condicao verdadeira
```

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

```
ifelse(2 < 1, 2*1, 1/2) # condicao falsa
```

```
## [1] 0.5
```

3.3.2 *Loops*

É muito trabalhoso reescrever código a fim de obeter repetições, sem mencionar o tempo gasto nesta reescrita. Sendo assi, o R possui algumas funções de repetições são elas: `for()`, `while()` e `repeat()`.

A função `for()` repete o código para o comprimento da sequência indicada à ela.

```
for(variavel in sequencia){
  comandos a serem repetidos
}
```

No exemplo abaixo a variável i vai assumir um valor da sequência numérica 1, 2, 3, 4 e 5, e então executar a função `print()` em i para cada valor da sequência adotada por i.

```
# : cria uma sequencia Ex.: sequencia do 1 ao 5 = 1:5
for(i in 1:5){
  print(i)
}
```

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

Outro exemplo do uso do for: Vamos printar na tela as cindo primeiras letras do alfabeto.

```
for(letra in letters[1:5]){
  print(letra)
}
```

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

Já a função `while()` executa os comandos enquanto a condição informada a ela for verdadeira.

```
while(condição){
  comandos a serem repetidos
}
```

Por exemplo: vamos construir um temporizador que determina um espaço de tempo de cinco segundos.

```
contador <- 1
while(contador <= 5){
  print(contador)
  contador = contador + 1
}
```

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

Perceba que não foi exatamente um espaço de tempo de cinco segundos, foi mais rápido. Vamos inserir um comando ao R dizendo a ele para aguardar um segundo após a execução anterior.

```
contador <- 1
while(contador <= 5){
  print(contador)
  contador = contador + 1
  Sys.sleep(1)
}
```

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

A função `repeat()` é usada quando queremos repetir um código sem a avaliação de uma condição. Atenção: vamos precisar utilizazr a função `break()` para dizer ao programa o momento em que deve parar a execução, ou seja a repetição. Também utilizaremos a função `if()` para avaliar a condição e então chamar o **break**.

```
contador <- 10
repeat{
  print(contador)
  contador <- contador + 10
  if(contador > 100) break()
}
```

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

```
## [1] 20
## [1] 30
## [1] 40
## [1] 50
## [1] 60
## [1] 70
## [1] 80
## [1] 90
## [1] 100
```

3.4 Lista de exercícios

1. Declare três variáveis atribuindo valores numéricos e apresente o resultado da multiplicação das suas combinações dois a dois destas três variáveis (cada variável com um número). Ex.: variáveis A, B, e C mostre AxB, AxC e BxC com atribuição dos valores as variáveis.

2. Converta (no R) a temperatura Fahrenheit 78 °F para Centígrados. Fórmula: C = (F-32) x (5/9).

3. Calcule (no R):

- o resto da divisão de 7 por 9
- 2 elevado ao cubo
- raiz quadrada de 64

4. Elabore um algoritmo que:

- crie um vetor com uma sequência numérica de 5 números
- faça um loop para calcular a soma destes números

5. Elabore um algoritmo que:

- crie um vetor com uma sequência numérica de 7 números
- faça um loop para calcular a média destes números

6. Elabore um algoritmo que:

- crie um vetor com uma sequência numérica de 12 números
- faça a soma dos números pares

Chapter 4

Fundamentos do R

Chapter 5

Processamento de dados

Chapter 6

Blocks

6.1 Equations

Here is an equation.

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{6.1}$$

You may refer to using `\@ref(eq:binom)`, like see Equation (6.1).

6.2 Theorems and proofs

Labeled theorems can be referenced in text using `\@ref(thm:tri)`, for example, check out this smart theorem 6.1.

Theorem 6.1. *For a right triangle, if c denotes the* length *of the hypotenuse and a and b denote the lengths of the* ***other*** *two sides, we have*

$$a^2 + b^2 = c^2$$

Read more here https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html.

6.3 Callout blocks

The R Markdown Cookbook provides more help on how to use custom blocks to design your own callouts: https://bookdown.org/yihui/rmarkdown-cookbook/custom-blocks.html

Chapter 7

Sharing your book

7.1 Publishing

HTML books can be published online, see: https://bookdown.org/yihui/bookdown/publishing.html

7.2 404 pages

By default, users will be directed to a 404 page if they try to access a webpage that cannot be found. If you'd like to customize your 404 page instead of using the default, you may add either a `_404.Rmd` or `_404.md` file to your project root and use code and/or Markdown syntax.

7.3 Metadata for sharing

Bookdown HTML books will provide HTML metadata for social sharing on platforms like Twitter, Facebook, and LinkedIn, using information you provide in the `index.Rmd` YAML. To setup, set the `url` for your book and the path to your `cover-image` file. Your book's `title` and `description` are also used.

This `gitbook` uses the same social sharing data across all chapters in your book- all links shared will look the same.

Specify your book's source repository on GitHub using the `edit` key under the configuration options in the `_output.yml` file, which allows users to suggest an edit by linking to a chapter's source file.

Read more about the features of this output format here:

https://pkgs.rstudio.com/bookdown/reference/gitbook.html

Or use:

```
?bookdown::gitbook
```