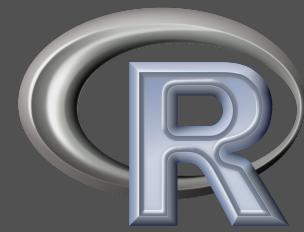




ME115 - Linguagem R

Parte 05

1º semestre de 2023



R Markdown

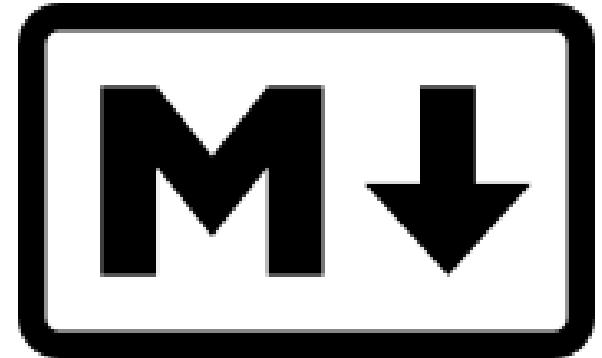
Markdown

Markdown é uma linguagem simples e leve usada para criar textos formatados, como HTML, usando uma codificação mínima, simples, fácil de ser lida e mais limpa.

Basicamente, alterações nos textos (subtítulos, negrito, itálico etc) são marcadas apenas com os símbolos do teclado, sem usar teclas de atalho, menus, selecionando o texto e sem aquele visual complexo de HTML.

Desenvolvido em 2004 por John Gruber e Aaron Swartz.

Amplamente usada para escrever páginas da web, blogs, mensagens de texto, fóruns, páginas de documentação, etc.



R Markdown

Assim como a linguagem Markdown, o R Markdown facilita a criação de diversos formatos de arquivo.

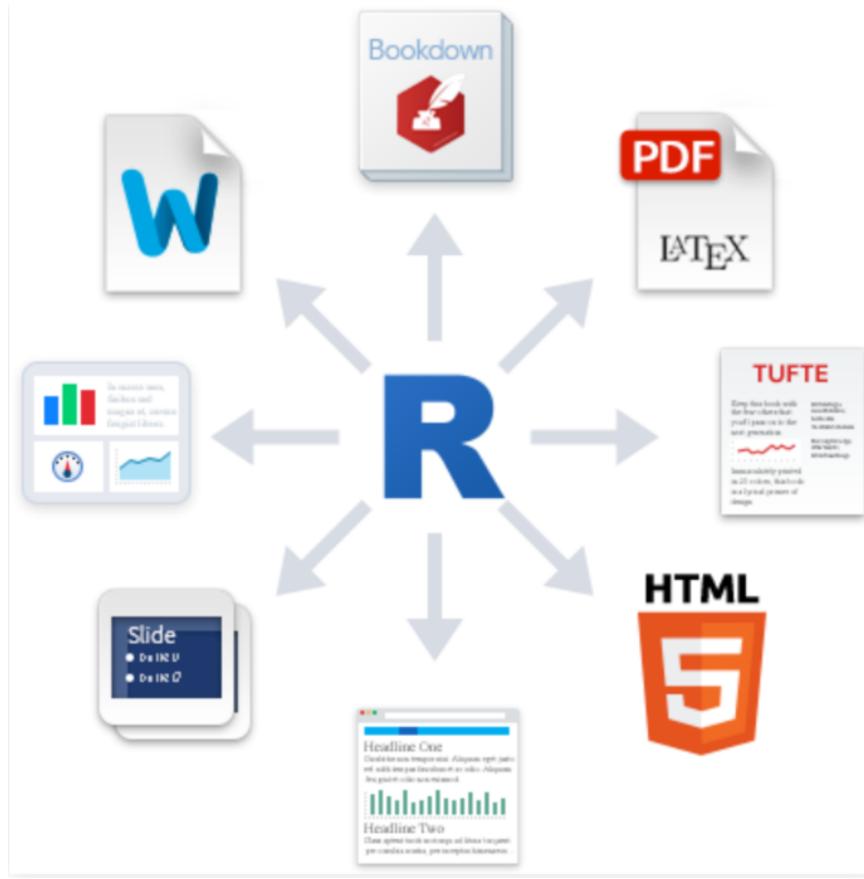
Usado para:

- Salvar e executar códigos (como um R Script);
- Gerar relatórios dinâmicos;
- Facilitar a vida do autor, pois combina texto e código em um único arquivo,



Você precisa do pacote `rmarkdown`, mas não é necessário instalar ou carregar esse pacote explicitamente, já que o RStudio faz isso automaticamente.

R Markdown



- Documentos: HTML, PDF, Word
- Slides: HTML (ioslides, slidify), PDF (Beamer), Power Point
- Books
- Handouts
- Dashboards
- Websites
- Shiny

Os slides dessa aula são preparados em R Markdown e são compilados usando **ioslides**.

R Markdown

Basicamente, um arquivo em R Markdown (formato .Rmd) é composto por três tipos de conteúdos:

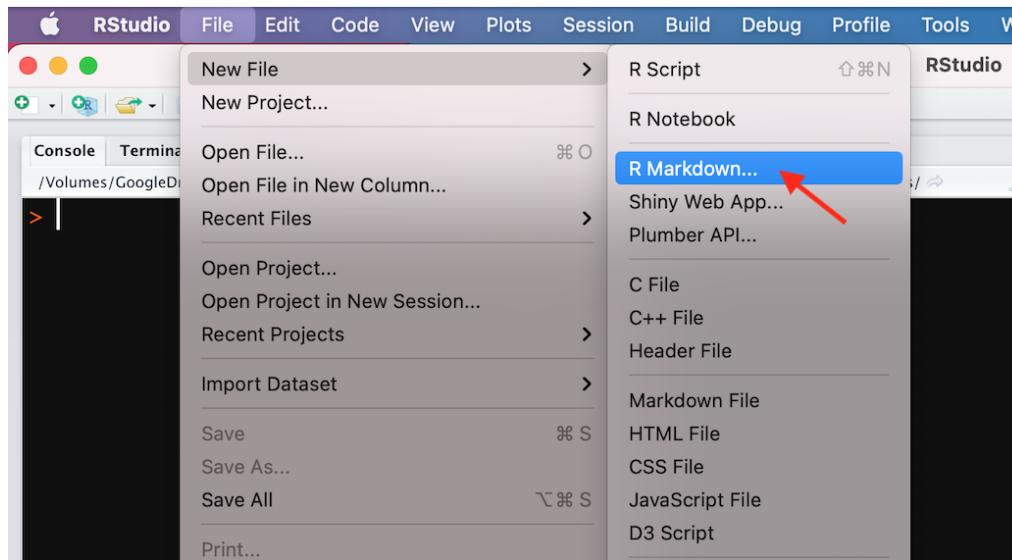
- Um cabeçalho (opcional) YAML delimitado por `---`
- Texto, tabelas, equações e imagens
- Códigos de R (*chunks*) delimitado por `````

The screenshot shows an RStudio interface with an 'Untitled1' document open. The code is highlighted in green and orange. A red box highlights the YAML header (lines 1-6), which is labeled 'Cabeçalho'. Another red box highlights the text block (lines 12-14), which is labeled 'Texto, tabelas, equações, figuras, etc...'. A third red box highlights the R code chunk (lines 18-20), which is labeled 'R Chunk (código)'.

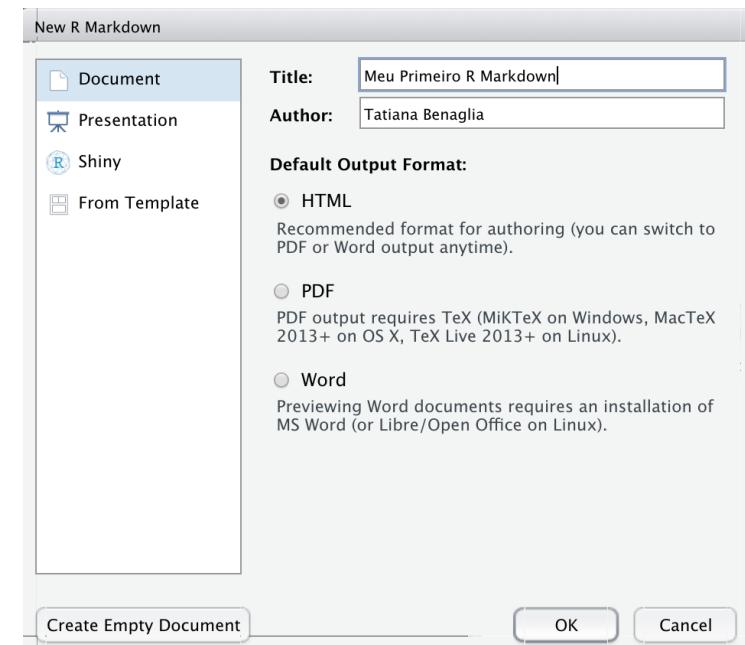
```
1 ---  
2 title: "R Markdown - Exemplo"  
3 author: "Tatiana Benaglia"  
4 date: "4/20/2021"  
5 output: html_document  
6 ---  
7  
8 ```{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10 ```  
11  
12 ## R Markdown  
13  
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.  
15  
16 When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:  
17  
18 ```{r cars}  
19 summary(cars)  
20 ```
```

Como criar um documento em R Markdown

No RStudio, clique em **File > New File > R Markdown**, como indica a figura:



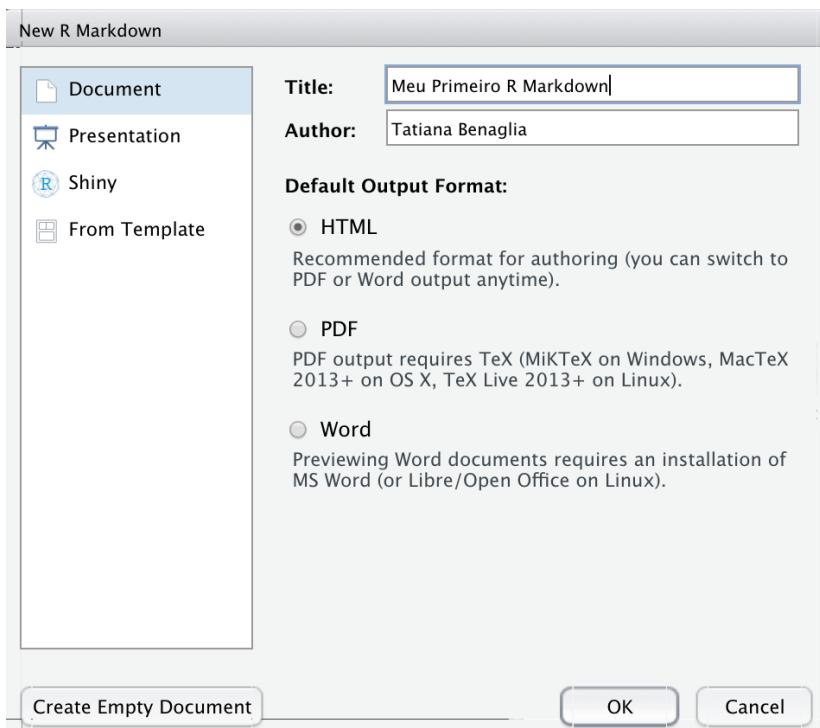
E isso abrirá a seguinte janela:



Preencha com o título e autor(es) do documento, mas sabendo que esses parâmetros podem ser modificados posteriormente também.

Como criar um documento em R Markdown

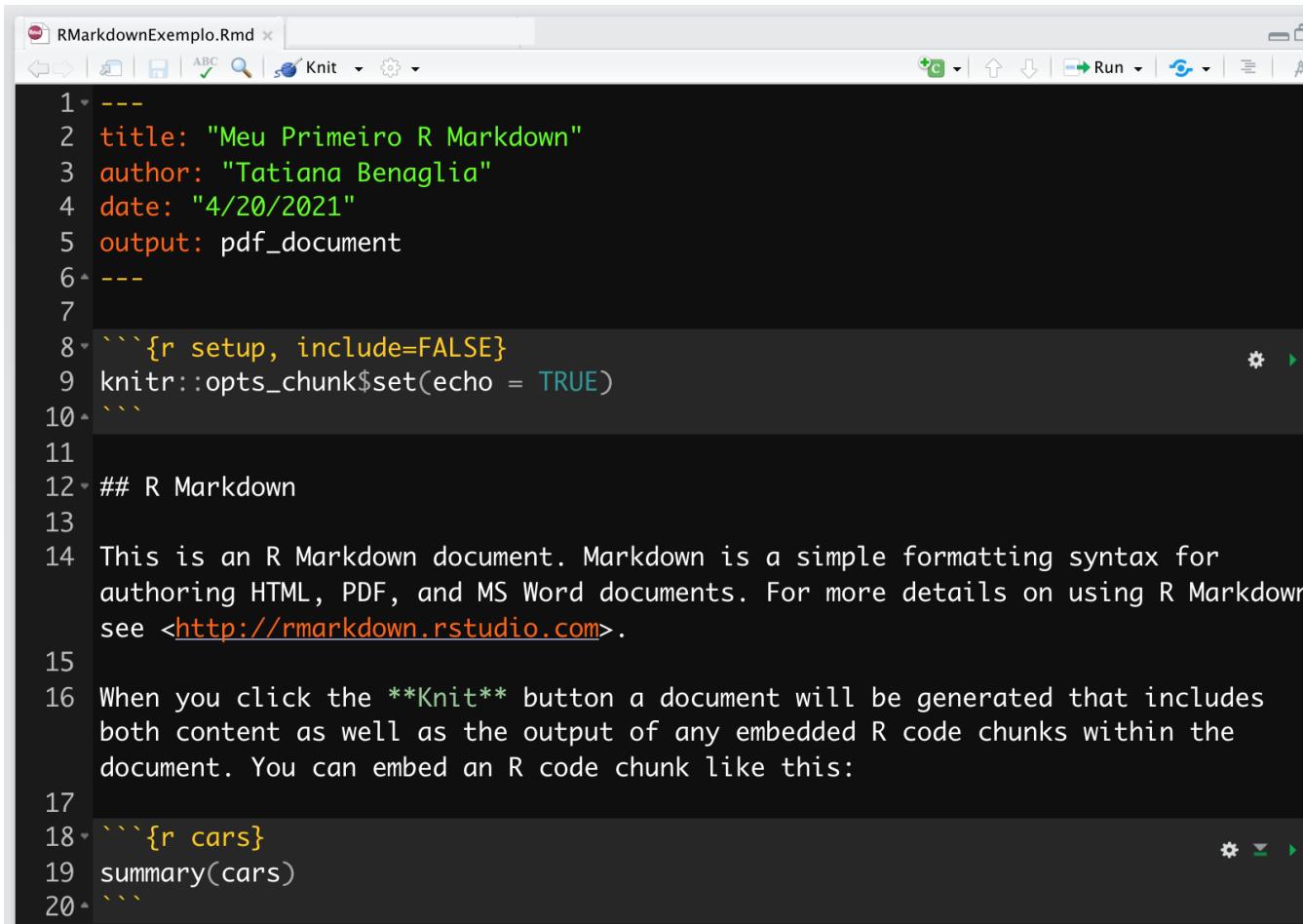
Nessa janela você seleciona também o tipo de saída:



- **html_document**: gera um arquivo em HTML (web page)
- **pdf_document**: gera um arquivo em PDF
- **word_document**: gera um arquivo Microsoft Word (.docx)
- **beamer_presentation**: gera uma apresentação em PDF (Beamer/LaTeX)
- **ioslides_presentation**: gera slides em HTML

Ao clicar em OK, será criado automaticamente um template do seu arquivo R Markdown. Experimente!

R Markdown - Exemplo



The screenshot shows the RStudio interface with the file 'RMarkdownExemplo.Rmd' open. The code editor displays the following R Markdown code:

```
1 ---  
2 title: "Meu Primeiro R Markdown"  
3 author: "Tatiana Benaglia"  
4 date: "4/20/2021"  
5 output: pdf_document  
6 ---  
7  
8 ```{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10 ````  
11  
12 ## R Markdown  
13  
14 This is an R Markdown document. Markdown is a simple formatting syntax for  
authoring HTML, PDF, and MS Word documents. For more details on using R Markdown  
see <http://rmarkdown.rstudio.com>.  
15  
16 When you click the **Knit** button a document will be generated that includes  
both content as well as the output of any embedded R code chunks within the  
document. You can embed an R code chunk like this:  
17  
18 ```{r cars}  
19 summary(cars)  
20 ````
```

Markdown Sintaxe

syntax

```
Plain text
End a line with two spaces to start a new paragraph.
*italics* and _italics_
**bold** and __bold__
superscript^2^
~~strikethrough~~
[link] (www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

###### Header 6

endash: --
emdash: ---
ellipsis: ...
inline equation: $A = \pi * r^2$
image: 

horizontal rule (or slide break):

***
```

becomes

Plain text
End a line with two spaces to start a new paragraph.

italics and *italics*
bold and **bold**
superscript²
strikethrough

[link](#)

Header 1

Header 2

Header 3

Header 4

Header 5

Header 6

endash: –
emdash: —
ellipsis: ...
inline equation: $A = \pi * r^2$



horizontal rule (or slide break):

Markdown Sintaxe (cont.)

syntax

```
* unordered list
* item 2
  + sub-item 1
  + sub-item 2

1. ordered list
2. item 2
  + sub-item 1
  + sub-item 2

Table Header | Second Header
----- | -----
Table Cell   | Cell 2
Cell 3       | Cell 4
```

becomes

```
• unordered list
• item 2
  ◦ sub-item 1
  ◦ sub-item 2

1. ordered list
2. item 2
  ◦ sub-item 1
  ◦ sub-item 2
```

Table Header	Second Header
Table Cell	Cell 2
Cell 3	Cell 4

Dependendo do formato escolhido para o seu documento, você pode editar seu Rmd misturando Markdown com HTML ou LaTeX se as saídas forem em HTML ou PDF, respectivamente.

R Markdown - Como Funciona

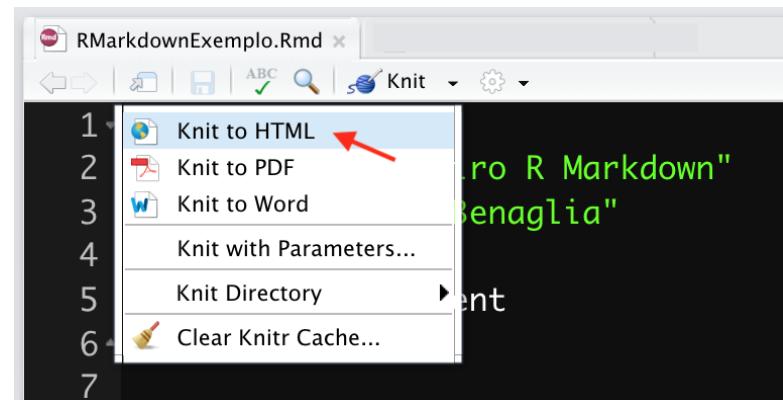
O arquivo .Rmd é compilado pelo pacote `knitr` usando a função `render()`

```
rmarkdown::render("RMarkdownExemplo.Rmd")
```

Veja o fluxo do que acontece:



Atalho: No RStudio, basta clicar no botão Knit e selecionar a opção desejada, como mostra a figura ao lado.



Pacote **knitr**

O pacote **knitr** é usado para criar relatórios dinâmicos, por possibilitar o uso dos chunks.

Ao compilar o documento .Rmd, os códigos contidos em *chunks* são executados criando ao final um documento independente do R no formato escolhido.

Verifique se esse pacote está instalado no seu RStudio. Se não estiver, instale-o:

```
install.packages("knitr")
```



R chunk

Utilize a sintaxe do pacote `knitr` para inserir o código do R no relatório.

Para inserir um novo *chunk*: Code > Insert Chunk (Ctrl+Alt+I)

O R executa o código e inclui os resultados no arquivo de saída. Veja um exemplo:



Opções do chunk

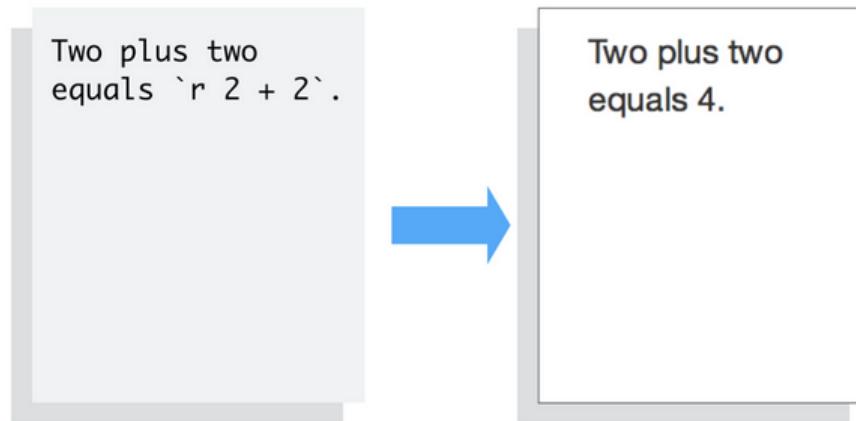
option	default	effect
eval	TRUE	Whether to evaluate the code and include its results
echo	TRUE	Whether to display code along with its results
warning	TRUE	Whether to display warnings
error	FALSE	Whether to display errors
message	TRUE	Whether to display messages
tidy	FALSE	Whether to reformat code in a tidy way when displaying it
results	"markup"	"markup", "asis", "hold", or "hide"
cache	FALSE	Whether to cache results for future renders
comment	"##"	Comment character to preface results with
fig.width	7	Width in inches for plots created in chunk
fig.height	7	Height in inches for plots created in chunk

Se quiser aplicar alguma opção em TODOS os *chunks* do seu arquivo, use a opção global:

```
knitr::opts_chunk$set(message = FALSE)
```

Inserir código diretamente no texto

Resultados de códigos podem ser inseridos diretamente no texto usando `r` e serão indistinguíveis do texto ao redor.



R Markdown irá sempre:

- mostrar os resultados do código digitado no texto, mas não o código em si;
- aplicar formatação de texto relevante aos resultados.

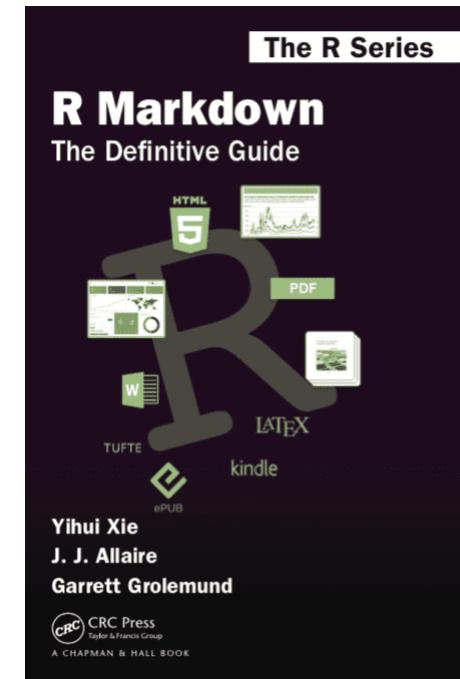
R Markdown: The Definite Guide

Você pode usar o R Markdown para fazer suas atividades de ME115, ou listas de exercícios/trabalhos de muitos outros cursos.

É impossível apresentar toda a potencialidade do R Markdown em uma única aula.

O melhor é pesquisar e buscar aprender os recursos à medida que surgirem as necessidades.

Recomendo que vocês leiam o livro: [R Markdown: The Definitive Guide](#)



R Markdown Cheat Sheet

rmarkdown :: CHEAT SHEET

What is rmarkdown?

.Rmd files • Develop your code and ideas side-by-side in a single document. Run code as individual chunks or as an entire document.
Dynamic Documents • Knit together plots, tables, and results with narrative text. Render to a variety of formats like HTML, PDF, MS Word, or MS PowerPoint.
Reproducible Research • Upload, link to, or attach your report to share. Anyone can read or run your code to reproduce your work.

Workflow

- 1 Open a new .Rmd file in the RStudio IDE by going to File > New File > R Markdown.
- 2 Embed code in chunks. Run code by line, by chunk, or all at once.
- 3 Write text and add tables, figures, images, and citations. Format with Markdown syntax or the RStudio Visual Markdown Editor.
- 4 Set output format(s) and options in the YAML header. Customize themes or add parameters to execute or add interactivity with shiny.
- 5 Save and render the whole document. Knit periodically to preview your work as you write.
- 6 Share your work!

Embed Code with knitr

CODE CHUNKS

Surround code chunks with `{{`r}}` and `{{` or use the Insert Code Chunk button. Add a chunk label and/or chunk options inside the curly braces after r.

```
```{r chunk-label, include=FALSE}
summary(mtcars)
```
```

SET GLOBAL OPTIONS

Set options for the entire document in the first chunk.

```
```{r include=FALSE}
knitr::opts_chunk$set(message = FALSE)
```

#### INLINE CODE

Insert `{{`r`}}

Code is evaluated at render and results appear as text.

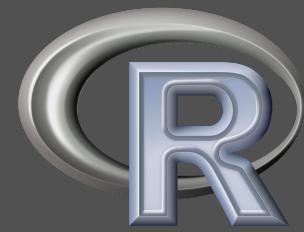
```
"Built with `r getRversion()`" -> "Built with 4.1.0"
```



The screenshot displays the R Markdown Cheat Sheet with several sections:

- SOURCE EDITOR:** Shows the RStudio interface with the source editor open. It highlights steps 1 through 6: 1. New File, 2. Embed Code, 3. Write Text, 4. Set Output Format(s) and Options, 5. Save and Render, and 6. Share.
- RENDERED OUTPUT:** Shows the rendered HTML output of the R Markdown document. It includes sections for Document Title, R Markdown, and summary(cars). It also shows the RStudio interface with the rendered file open.
- Write with Markdown:** A section titled "Write with Markdown" containing a logo and text about the syntax rendering. It lists plain text, headings (Header 1, Header 2, Header 6), lists (unordered and ordered), URLs, and captions.
- Insert Citations:** A section titled "Insert Citations" with instructions for creating citations from a bibliography file, a Zotero library, or DOI references.
- BUILD YOUR BIBLIOGRAPHY:** A section titled "BUILD YOUR BIBLIOGRAPHY" with instructions for adding BibTeX or CSL bibliographies to the YAML header.
- Insert Tables:** A section titled "Insert Tables" with instructions for outputting data frames as tables using `knitr::kable`.
- Results:** A section titled "Results" showing examples of plots and tables.

Fonte: [R Markdown Cheat Sheet by RStudio](#)



Python com R

# **reticulate** package



O pacote **reticulate** fornece um conjunto abrangente de ferramentas para interoperabilidade entre o Python e o R. O pacote inclui recursos para:

- Chamar o Python do R de várias maneiras, incluindo o R Markdown.
- Tradução entre objetos R e Python.
- Ligação flexível para diferentes versões do Python, incluindo ambientes virtuais e ambientes Conda.

O **reticulate** incorpora uma sessão Python em sua sessão R, permitindo interoperabilidade perfeita e de alto desempenho. Se o desenvolvedor de R que usa Python para algum de seus trabalhos ou um membro da equipe de ciência de dados que usa os dois idiomas, o **reticulate** pode simplificar drasticamente o fluxo de trabalho.



Fonte: <https://rstudio.github.io/reticulate/>

# Python com R Cheat Sheet

## Use Python with R with reticulate :: CHEAT SHEET



The `reticulate` package lets you use Python and R together seamlessly in R code, in R Markdown documents, and in the RStudio IDE.

### Python in R Markdown

(Optional) Build Python env to use.

Add `knitr::knit_engines$set(python = reticulate::eng_python)` to the setup chunk to set up the reticulate Python engine (not required for `knitr >= 1.18`).

Suggest the Python environment to use, in your setup chunk.

Begin Python chunks with ````{python}`. Chunk options like `echo`, `include`, etc. all work as expected.

Use the `r` object to access objects created in Python chunks from R chunks.

Python chunks all execute within a single Python session so you have access to all objects created in previous chunks.

Use the `r` object to access objects created in R chunks from Python chunks.

Output displays below chunk, including matplotlib plots.

```
python.Rmd:1-21.4
1: ``{r setup, include = FALSE}
2: library(reticulate)
3: virtualenv_create("fmri-proj")
4: py_install("seaborn", envname = "fmri-proj")
5: use_virtualenv("fmri-proj")
6:
7:
8: ```{python, echo = FALSE}
9: import seaborn as sns
10: fmri = sns.load_dataset("fmri")
11:
12:
13: # {r}
14: f1 <- subset(fmri, region == "parietal")
15:
16:
17: # {python}
18: import matplotlib as plt
19: sns.lmplot("timepoint", "signal", data=f1)
20: plt.pyplot.show()
21:
```

R Console: A small window showing a plot of signal vs timepoint.

R Markdown: A larger window showing the R code and the resulting plot.

R Script: Another window showing the R code.

### Object Conversion

Tip: To index Python objects begin at 0, use integers, e.g. `0L`

Reticulate provides automatic built-in conversion between Python and R for many Python types.

R	↔	Python
Single-element vector		Scalar
Multi-element vector		List
List of multiple types		Tuple
Named list		Dict
Matrix/Array		NumPy ndarray
Data Frame		Pandas DataFrame
Function		Python function
NULL, TRUE, FALSE		None, True, False

Or, if you like, you can convert manually with

`py_to_r(x)` Convert a Python object to an R object. Also `r_to_py()`, `py_to_r()`

`tuple(..., convert = FALSE)` Create a Python tuple. `tuple("a", "b", "c")`



dict(..., convert = FALSE) Create a Python dictionary object. Also `py_dict()` make a dictionary that uses Python objects as keys. `dictfoo = "bar", index = 42L`

`np_array(data, dtype = NULL, order = "C")` Create NumPy arrays. `np_array(c(1:8), dtype = "float16")`

`array_reshape(x, dim, order = c("C", "F"))` Reshape a Python array. `x <- 1:4; array_reshape(x, c(2, 2))`

`py_func(f)` Wrap an R function in a Python function with the same signature. `py_func(x)`

`py_main_thread_func(f)` Create a function that will always be called on the main thread.

`iterate(i, f = base::identity, simplify = TRUE)` Apply an R function to each value of a Python iterator or return the values as an R vector, drawing the iterator as you go. Also `iter_next()` and `as_iterator()`. `iterate(iterator, print)`

`py_iterator(fn, completed = NULL)` Create a Python iterator from an R function. `seq_gen <- function(x){n <- x; function() {h <- n + 1: n}; py_iterator(seq_gen)}`

`with(data, expr, as = NULL, ...)` Evaluate an expression within a Python context manager.

`py<- import_builtins(); with(py$open("output.txt", "w")) %as% file,`

`{filewrite("Hello, there!")})`

### Helpers

`py_capture_output(expr, type = c("stdout", "stderr"))` Capture and return Python output. Also `py_suppress_warnings()`. `py_capture_output("x")`

`py_get_attr(x, name, silent = FALSE)` Get an attribute of a Python object. Also `py_set_attr()`, `py_has_attr()`, and `py_list_attributes()`. `py_get_attr(x)`

`py_help(object)` Open the documentation page for a Python object. `py_help(sns)`

`py_last_error()` Get the last Python error encountered. Also `py_clear_last_error()` to clear the last error. `py_last_error()`

`py_save(object, filename, pickle = "pickle", ...)` Save and load Python objects with pickle. Also `py_load(object)`. `py_save(object, "x.pickle")`

`with(data, expr, as = NULL, ...)` Evaluate an expression within a Python context manager.

`py<- import_builtins();`

`with(py$open("output.txt", "w")) %as% file,`

`{filewrite("Hello, there!")})`

### Python in R

Call Python from R code in three ways:

#### IMPORT PYTHON MODULES

Use `import()` to import any Python module. Access the attributes of a module with `$`.

- `import(module, as = NULL, convert = FALSE, delay_load = FALSE)` Import a Python module. If `as = TRUE`, Python objects are converted to their equivalent R types. Also `import_from_path()`. `import("pandas")`

- `import_main(convert = TRUE)` Import the main module, where Python executes code by default. `import_main()`

- `import_builtins(convert = TRUE)` Import Python's built-in functions. `import_builtins()`

#### SOURCE PYTHON FILES

Use `source_python()` to source a Python script and make the Python functions and objects it creates available in the calling R environment.

- `source_python(file, envir = parent.frame(), convert = TRUE)` Run a Python script, assigning objects to a specified R environment. `source_python("file.py")`

#### RUN PYTHON CODE

Execute Python code into the `main` Python module with `py_run_file()` or `py_run_string()`.

- `py_run_string(code, local = FALSE, convert = TRUE)` Run Python code (passed as a string) in the `main` module. `py_run_string("x = 10"); py$x`

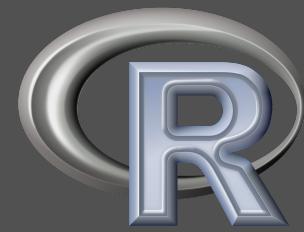
- `py_run_file(file, local = FALSE, convert = TRUE)` Run Python file in the `main` module. `py_run_file("script.py")`

- `py_eval(code, convert = TRUE)` Run a Python expression, return the result. Also `py_call()`. `py_eval("1 + 1")`

Access the results, and anything else in Python's `main` module, with `py`.

- `py` An R object that contains the Python `main` module and the results stored there. `py$x`

Fonte: [R Markdown Cheat Sheet by RStudio](#)



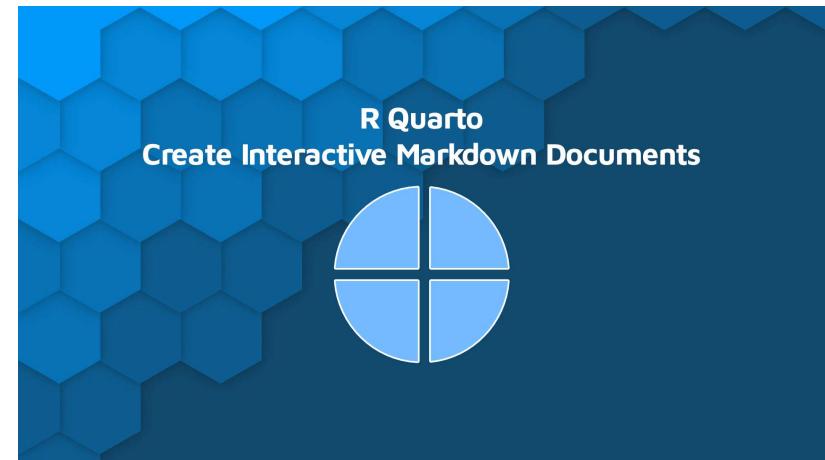
Quarto

# Quarto

R Quarto é uma versão de última geração do R Markdown.

O Quarto não se limita à linguagem de programação R. Também está disponível em Python, Julia e Observable.

Assim como o R Markdown com o Quarto você pode criar facilmente artigos, relatórios, apresentações, PDFs, livros, documentos do Word, ePubs e até sites inteiros de alta qualidade.





Mais detalhes: <https://quarto.org/>

The screenshot shows the Quarto website homepage. At the top, there is a navigation bar with links for Overview, Get Started, Guide, Extensions, Reference, Gallery, Blog, and Help. To the right of the navigation bar are social media icons for Twitter, GitHub, RSS, and a search icon. The main title "Welcome to Quarto" is displayed in a large, bold, dark blue font. Below the title, a subtitle reads "An open-source scientific and technical publishing system". A bulleted list details the features of Quarto:

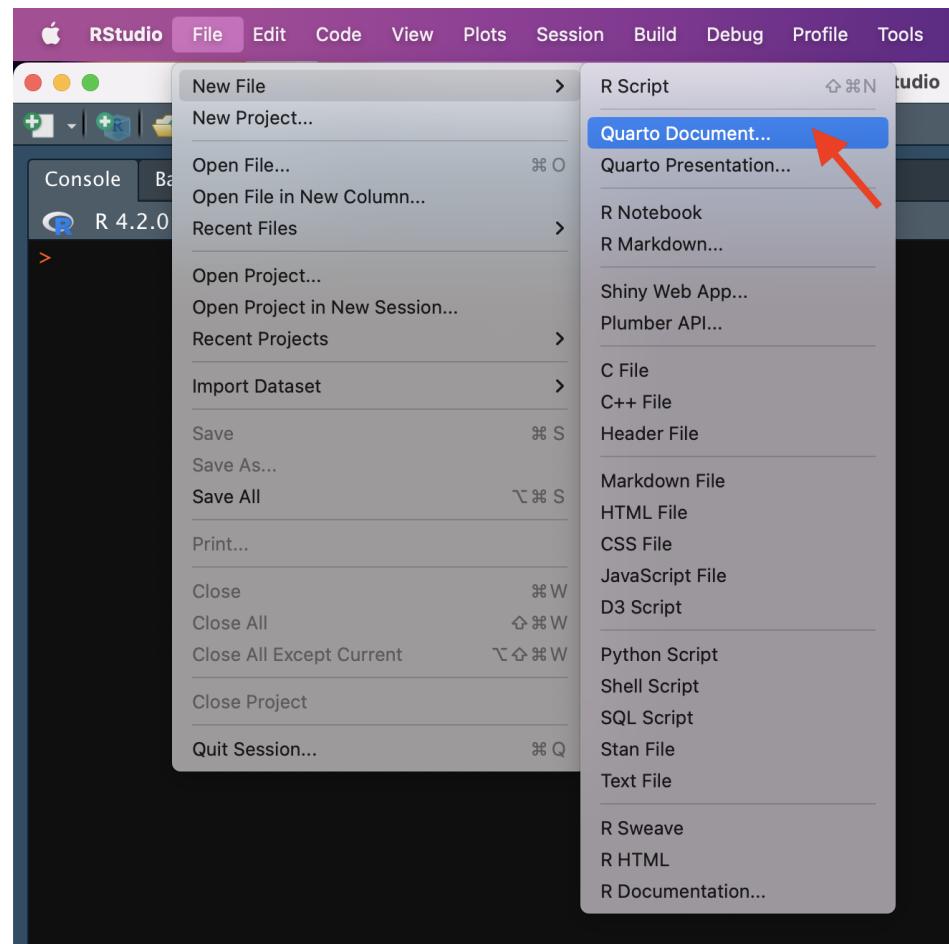
- Author using Jupyter notebooks or with plain text markdown in your favorite editor.
- Create dynamic content with Python, R, Julia, and Observable.
- Publish reproducible, production quality articles, presentations, websites, blogs, and books in HTML, PDF, MS Word, ePub, and more.
- Share knowledge and insights organization-wide by publishing to Posit Connect, Confluence, or other publishing systems.
- Write using Pandoc markdown, including equations, citations, crossrefs, figure panels, callouts, advanced layout, and more.

Below the list, a section titled "Analyze. Share. Reproduce. You have a story to tell with data—tell it with Quarto." is shown. At the bottom of the page are two buttons: "Get Started" (in a blue rounded rectangle) and "Guide" (in a grey rounded rectangle). To the right of the main content area, there is a sidebar with icons and labels for different types of publications:

	Websites / Blogs
	Articles
	Presentations
	Books
	Knowledge Repos

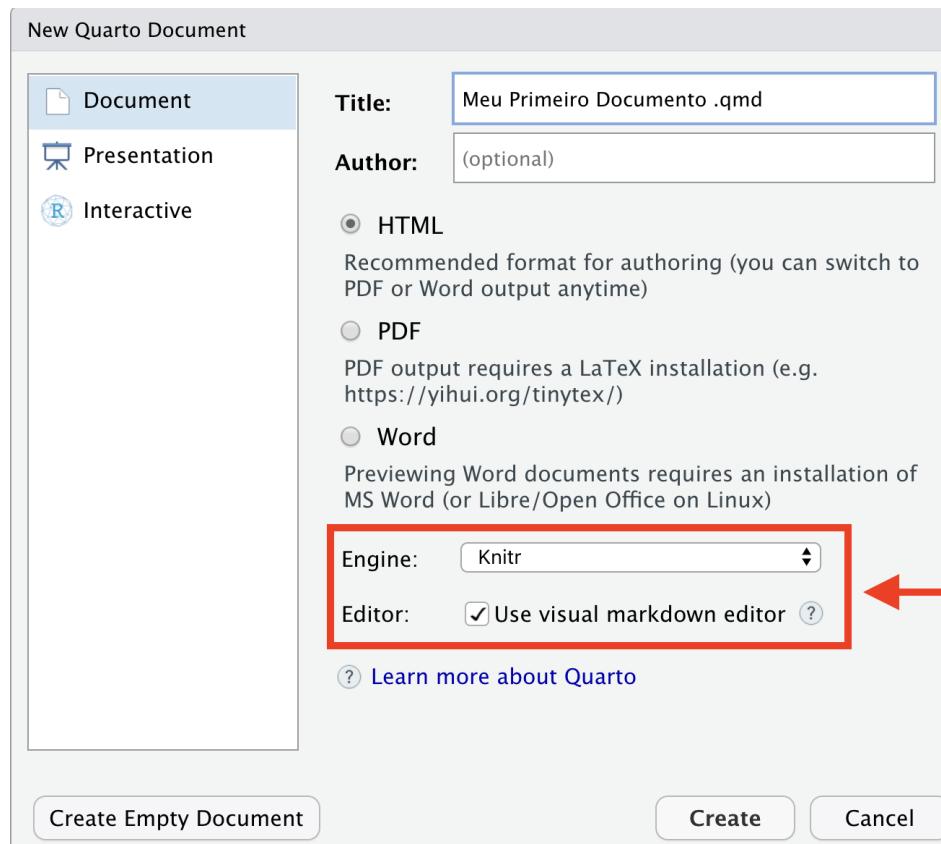
# Como criar um documento em Quarto

Assim como para R Markdown, no RStudio, clique em File > New File > Quarto Document, como indica a figura:

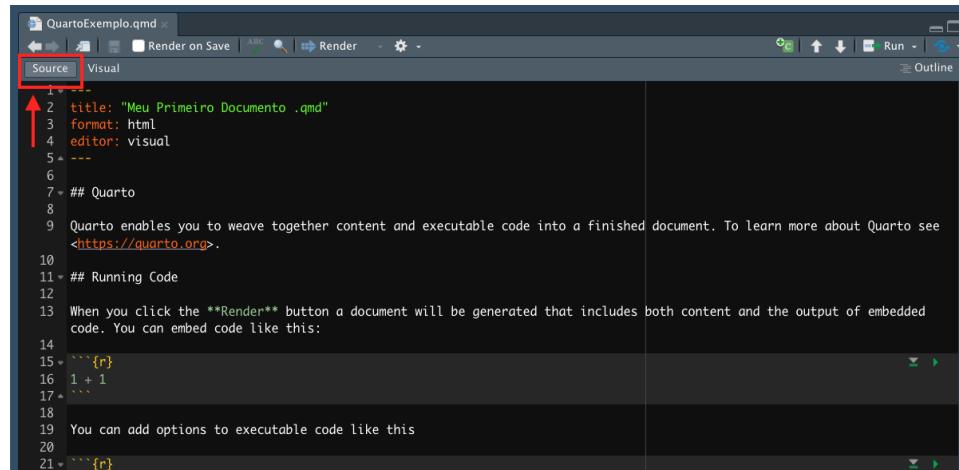


# Como criar um documento em Quarto

Nessa janela você seleciona também o tipo de saída. Além disso, você também seleciona o mecanismo e o editor.



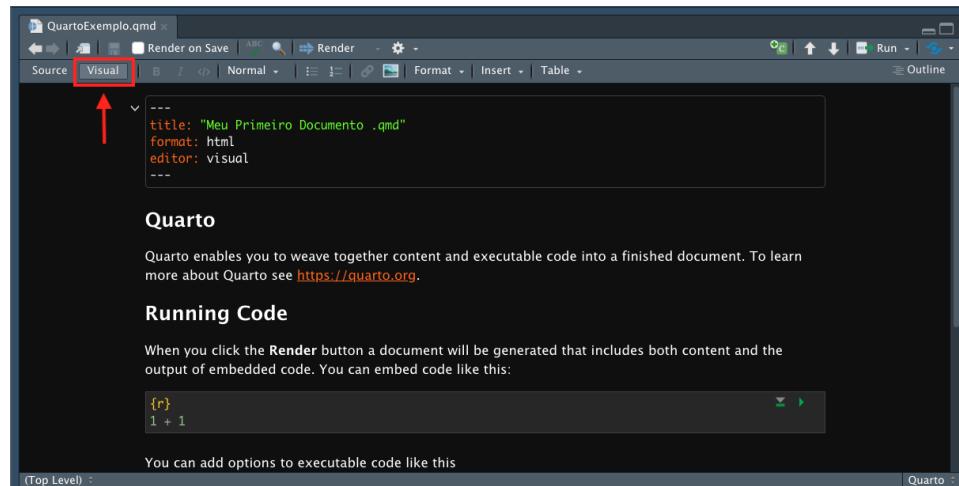
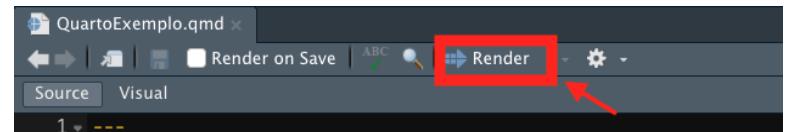
# Quarto - Exemplo



A screenshot of the Quarto interface showing the Source tab selected. The code editor displays a QMD file named "QuartoExemplo.qmd". The code includes a title block and a section titled "Quarto" containing a paragraph about weaving content and executable code.

```
1 ---
2 title: "Meu Primeiro Documento .qmd"
3 format: html
4 editor: visual
5 ---
6
7 ## Quarto
8
9 Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see
https://quarto.org.
10
11 ## Running Code
12
13 When you click the **Render** button a document will be generated that includes both content and the output of embedded
code. You can embed code like this:
14
15 ``{r}
16 1 + 1
17 ``
18
19 You can add options to executable code like this
20 ``{r}
21
```

Compilando o arquivo .qmd:



A screenshot of the Quarto interface showing the Visual tab selected. The rendered content is displayed in the main area, including the title "Quarto" and the "Running Code" section with its explanatory text and code block. A red box highlights the "Visual" tab in the toolbar, and a red arrow points from the Source tab in the first screenshot to the Visual tab here.

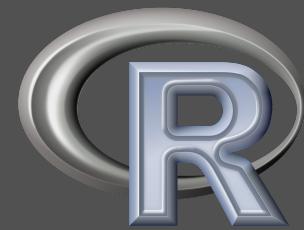
Quarto  
Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see <https://quarto.org>.

**Running Code**

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
{r}
1 + 1
```

You can add options to executable code like this



# Boas Práticas ao Escrever seu Código

# Boas Práticas

Ao escrever seu código em R, adote algumas (boas) práticas que deixem seu código mais fácil de ler, verificar e compartilhar.

Não existe nada oficial, mas existem algumas convenções bastante utilizadas e divulgadas na comunidade R.

Ter um bom estilo é importante porque, embora seu código possa ter apenas um autor, ele poderá ter vários leitores (Hadley Wickham - Stat405 - Style Guide).

“Good coding style is like using correct punctuation. You can manage without it, but it sure makes things easier to read.”

— Hadley Wickham

# Boas Práticas

Estilo de programação: conjunto de preferências do desenvolvedor.

**Using = instead of <- for assignment**



Se você está programando para você, siga seu estilo e seja consistente.

Se estiver trabalhando num time, façam um acordo quanto ao estilo a ser seguido.

As dicas apontadas aqui são práticas que eu utilizo ao escrever códigos em R.

# Nomeando Variáveis e Funções

Listamos aqui alguns estilos bastante utilizados:

- **alllowercase**: todas as letras minúsculas e sem separador de palavras como em `searchpaths()`, `srcfilecopy()` ou `mean()`.
- **period.separated**: todas as letras minúsculas e palavras são separadas por ponto, como em `as.numeric()` ou `read.table()`.
- **underscore\_separated**: todas as letras minúsculas e palavras são separadas por *underscore* como em `seq_along()` ou `package_version()`.
- **lowerCamelCase**: primeira letra de cada palavra é maiúscula, exceto pela primeira palavra, como em `colMeans()` e `rowSums()`.
- **UpperCamelCase**: todas as primeiras letras das palavras são maiúsculas, como em `vectorize()` ou `NextMethod()`.

# Nomes de Arquivos

- Nomes de arquivos devem sempre terminar em `.R`
- Escolha nomes com um significado
- Evite caracteres especiais e espaços

```
GOOD
fit_models.R
utility_functions.R
```

```
BAD
fit models.R
foo.r
Programa1.r
```

# Nomes de Variáveis

- Escolhe nomes curtos e que indiquem o significado daquela variável
- Nomes de variáveis normalmente são em letras minúsculas
- NUNCA use nomes de funções ou variáveis já existentes

```
GOOD
fit_rt
event
event_window

BAD
fit_regression_tree
foo
Event
T <- 10 # T é um atalho de TRUE em R
c <- "constant"
```

# Espaçamento

Coloque espaços ao redor de todos os operadores (=, +, \*, ==, &&, <-, %\*%, etc.).

```
GOOD
x == y
a <- a ^ 2 + 1
```

```
BAD
x==y
a<-a^2+1
```

Não use espaços nos seguintes casos:

```
GOOD
car$cyl
dplyr::select
1:10
```

# Espaçamento

Use um espaço depois da vírgula e nenhum espaço antes da vírgula:

```
GOOD
mtcars[, "cyl"]
mtcars[1,]
mean(x = c(1, NA, 2), na.rm = TRUE)
mean(x=c(1, NA, 2), na.rm=TRUE)

BAD
mtcars[, "cyl"]
mtcars[1 ,]
mean(x = c(1, NA, 2),na.rm = TRUE)
```

O espaço ao redor do = para argumentos de função é opcional.

# Indentação

Sempre faça a indentação do seu código. Veja a diferença:

```
BAD
for(i in 1:10){
 if(i%%2==0)
 print(paste(i,"is even"))
}

GOOD
for (i in 1:10) {
 if (i %% 2 == 0)
 print(paste(i, "is even"))
}
```

Atalho Mágico: Command+I (Ctrl+I para Windows/Linux) irá fazer a indentação.

# Quebra de Linha

É comum, ao definir ou chamar uma função, que os argumentos não caibam em uma única linha. Então, quebramos em duas alinhas, alinhando na abertura dos parêntesis:

```
long_function_name <- function(arg1, arg2, arg3, arg4,
 long_argument_name1 = TRUE)
```

Se for necessário mais que duas linhas, então cada argumento pode ser colocado numa linha separada:

```
long_function_name <- function(long_argument_name1 = c("value1", "value2"),
 long_argument_name2 = TRUE,
 long_argument_name3 = NULL,
 long_argument_name4 = FALSE)
```

Nota: Prefira TRUE e FALSE em vez de T e F.

# Controles de Fluxo

As chaves {} definem a hierarquia mais importante de código em R:

- { deve ser o último caracter na linha;
- } deve ser o primeiro caracter na linha;
- o conteúdo deve estar indentado.

```
GOOD
if (x > 0) {
 log(x)
} else {
 message("x is negative or zero")
}

BAD
if(x > 0){log(x)
} else {message("x is negative or zero")}
```

# Comentários

- Sempre comente o seu código
- Comentários devem explicar o por quê e não o “o que”
- Comentários curtos podem ser colocados na mesma linha do código

```
GOOD
define iterator
i <- 1

BAD
set i to 1
i <- 1

plot(price, weight) # plot a scatter chart of price and weight
```

# Referências

Algumas referências utilizadas para a construção desse material:

- [R Markdown Basics](#)
- [R Markdown: The Definitive Guide](#)
- [R Markdown Cheat Sheet](#)
- [R Markdown Reference Guide](#)
- [R Coding Style](#)
- [The Tidyverse Style Guide](#)



Slides produzidos pela profa. Tatiana Benaglia.