

Ciência de dados em R

Curso-R

2020-02-20

Contents

Sobre	5
1 Instalação	7
1.1 Instalação do R	7
1.2 Instalação do RStudio	11
1.3 Instalação de pacotes	21
2 RStudio	23
2.1 Telas	23
2.2 Atalhos	25
2.3 Projetos	25
2.4 Cheatsheets	27
3 R Básico	29
3.1 Pedindo Ajuda	29
3.2 R como calculadora	33
3.3 Objetos e Classes	35
3.4 Valores especiais	43
3.5 Controles de Fluxo	43
3.6 Fórmulas	46
3.7 Gráficos (base)	47
3.8 Exercícios	54
3.9 Respostas	55
3.10 Observe que não há problema em usar os mesmos nomes para os argumentos. Isso se deve aos <i>environments</i>	60
4 Pipe	63
4.1 O operador pipe	63
4.2 Outros operadores	65
4.3 Exercícios	67
4.4 Respostas	67
5 Importação	71
5.1 readr	71

5.2	readxl	74
5.3	haven	76
6	Manipulação	79
6.1	Trabalhando com tibbles	81
6.2	dplyr	82
6.3	tidyr	93

Sobre

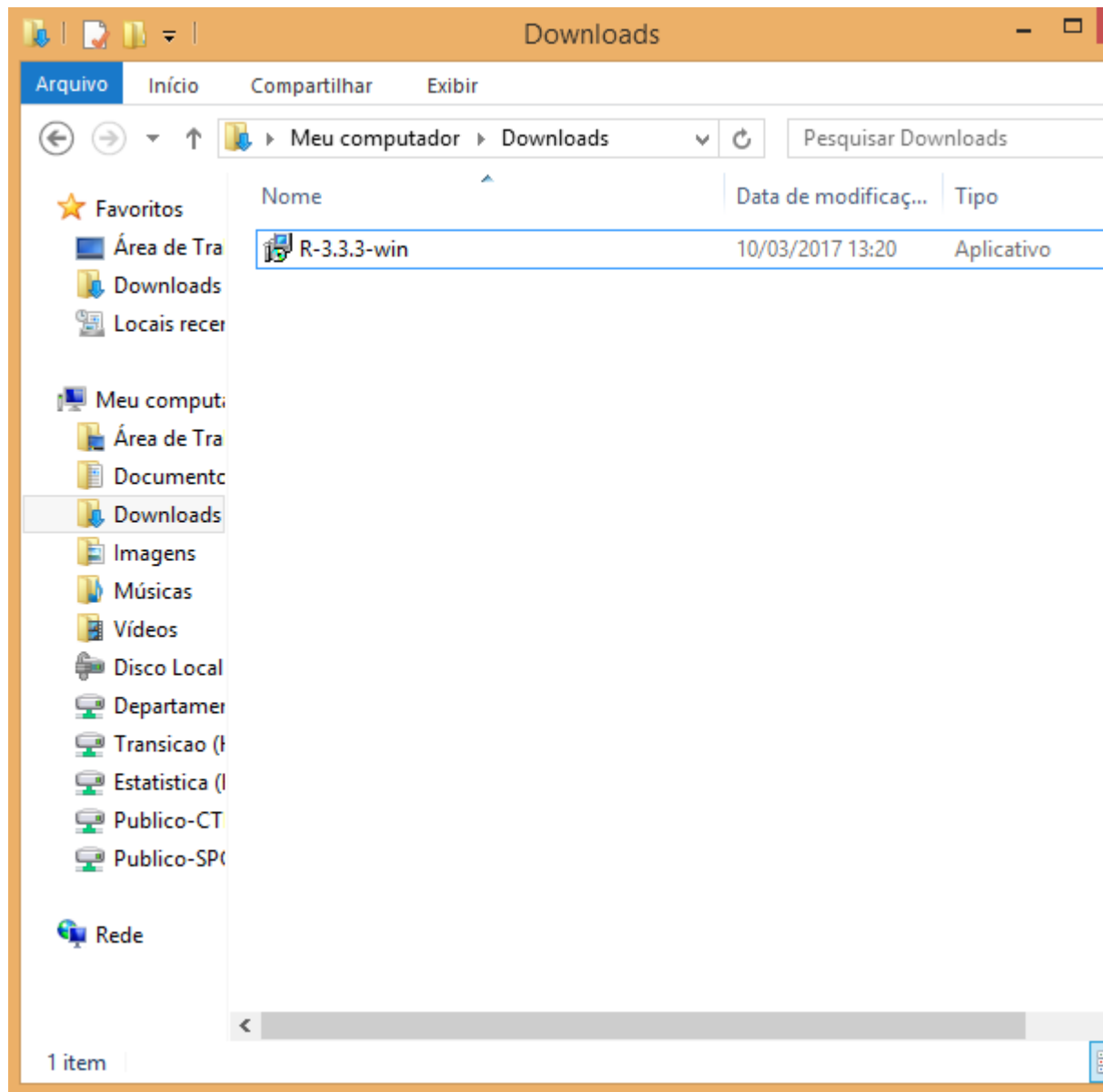
Chapter 1

Instalação

1.1 Instalação do R

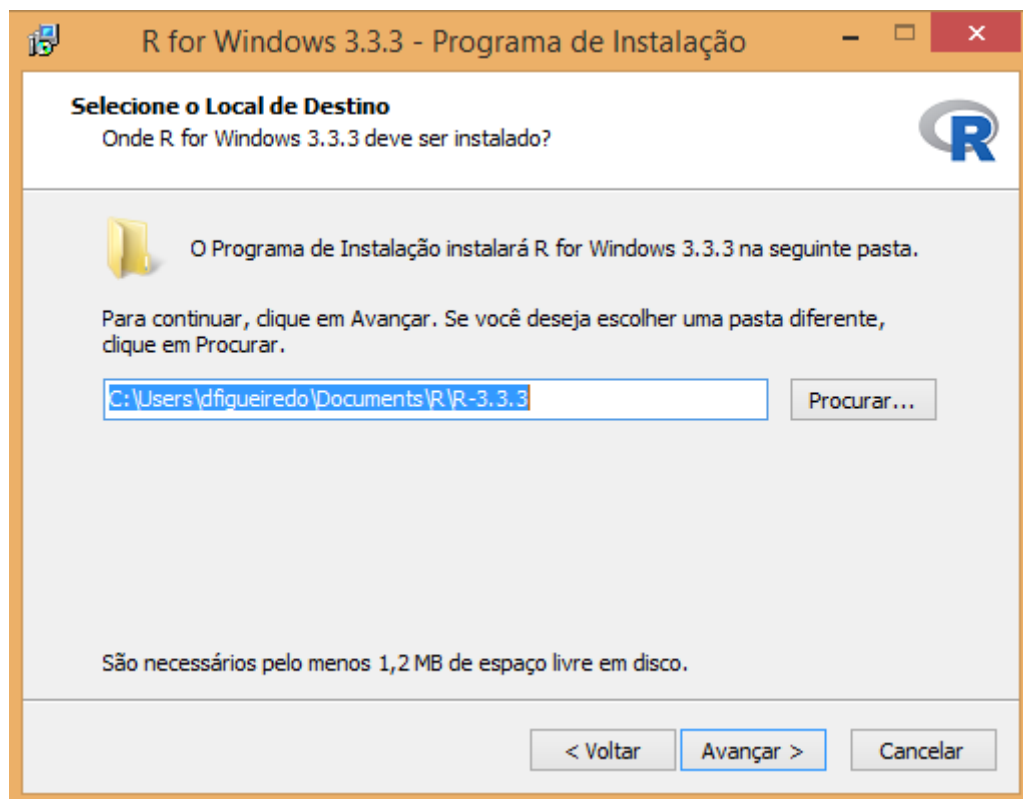
1.1.1 No Windows

1. Entre neste link e clique em “Download R x.x.x for Windows”, sendo x.x.x o número da versão mais recente disponível.
2. Salve o arquivo em qualquer pasta do seu computador. Aqui, salvamos na pasta “Downloads”.

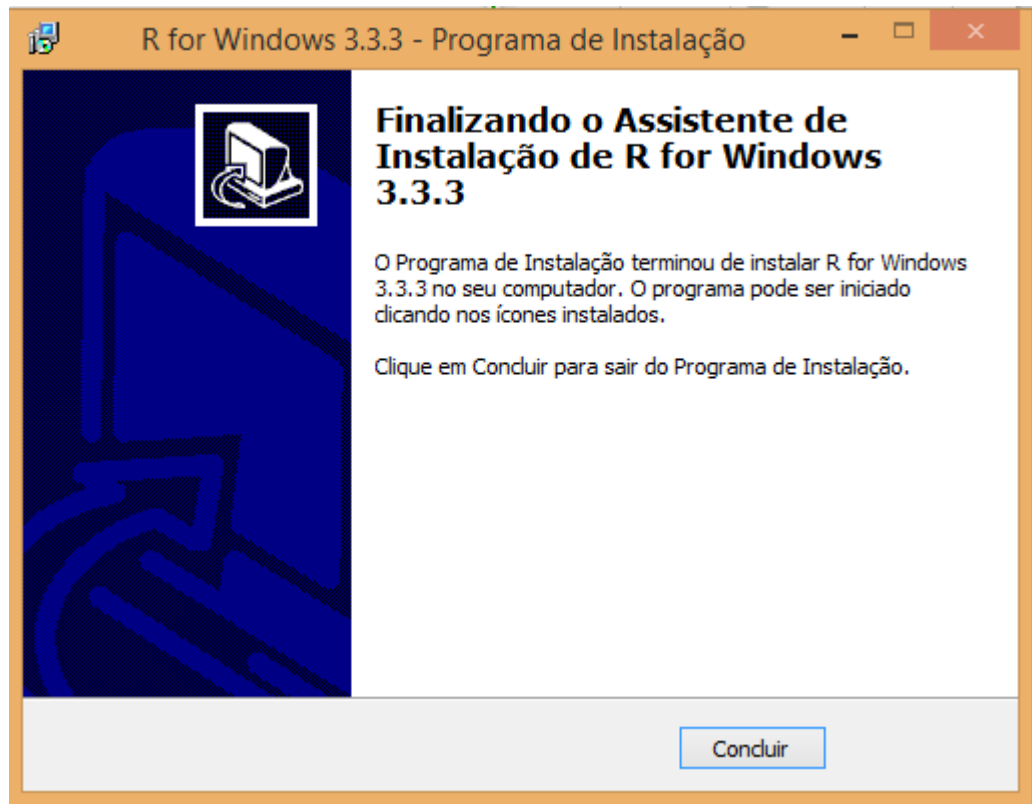


3. Clique no arquivo duas vezes com o botão esquerdo. Ele pedirá para você selecionar a linguagem da instalação. Escolha um idioma e clique em “OK”.
4. Clique em “Avançar” até chegar na tela da imagem abaixo. Nessa etapa, você precisará escolher a pasta de instalação. Se você escolher um local que não está dentro da pasta do seu usuário, você precisará de acesso de administrador. Se escolher uma pasta dentro do seu usuário (como na

imagem abaixo), não precisará.



5. Continue clicando em “Avançar” e, ao fim da instalação, em “Concluir”.



Pronto! O R está instalado no seu computador!

1.1.2 No Linux

Como a instalação no Linux depende da distribuição utilizada e, em geral, usuário de Linux são mais experientes, vamos informar apenas as coordenadas até as instruções/arquivos de instalação para cada distribuição. Se você tiver alguma dificuldade durante o processo, por favor envie a sua dúvida para ato@curso-r.com. Faremos o possível para ajudar.

1. Acesse este link e clique em *Download R for Linux*.
2. Em seguida, clique no link referente à distribuição que você utiliza.
3. Siga as instruções contidas na página para instalar o R.

1.2 Instalação do RStudio

1.2.1 No Windows











Agora vamos instalar o RStudio, a IDE que utilizaremos para editar e executar códigos em R.

1. Entre nesta página.






All Installers

Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system.

RStudio 1.2 requires a 64-bit operating system. If you are on a 32 bit system, you can use an [older version of RStudio](#).

OS	Download	Size
Windows 10/8/7	 RStudio-1.2.5033.exe	149.83 MB
macOS 10.12+	 RStudio-1.2.5033.dmg	126.89 MB
Ubuntu 14/Debian 8	 rstudio-1.2.5033-amd64.deb	96.18 MB
Ubuntu 16	 rstudio-1.2.5033-amd64.deb	104.14 MB
Ubuntu 18/Debian 10	 rstudio-1.2.5033-amd64.deb	105.21 MB
Fedora 19/Red Hat 7	 rstudio-1.2.5033-x86_64.rpm	120.23 MB
Fedora 28/Red Hat 8	 rstudio-1.2.5033-x86_64.rpm	120.87 MB
Debian 9	 rstudio-1.2.5033-amd64.deb	105.45 MB
SLES/OpenSUSE 12	 rstudio-1.2.5033-x86_64.rpm	98.87 MB
OpenSUSE 15	 rstudio-1.2.5033-x86_64.rpm	106.91 MB

Zip/Tarballs

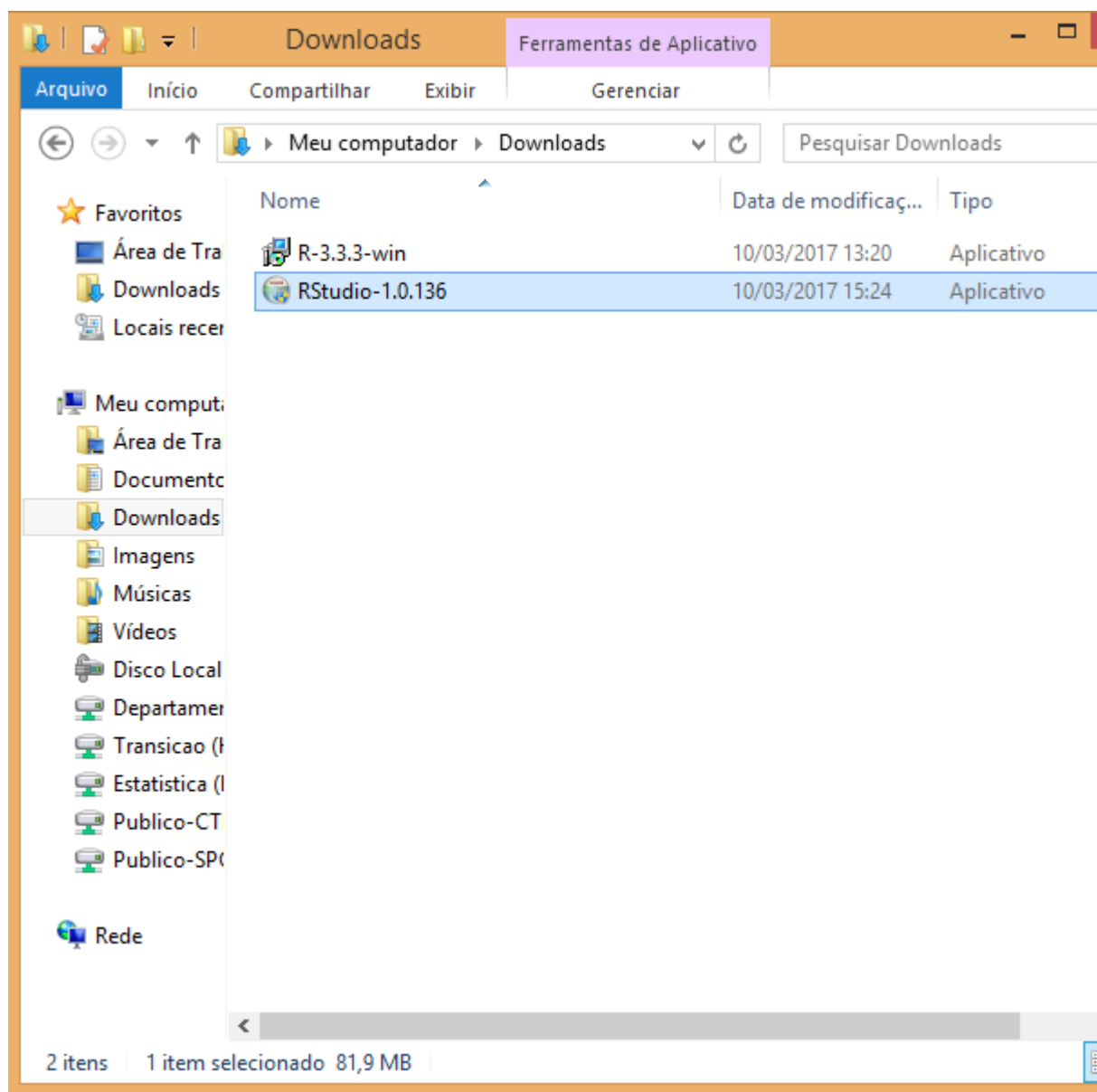
OS	Zip/tar	Size
Windows 10/8/7	 RStudio-1.2.5033.zip	219.03 MB
Ubuntu 14/Debian 8	 rstudio-1.2.5033-amd64-debian.tar.gz	144.32 MB
Ubuntu 18/Debian 10	 rstudio-1.2.5033-amd64-debian.tar.gz	155.83 MB
Fedora 19/Red Hat 7	 rstudio-1.2.5033-x86_64-fedora.tar.gz	155.25 MB
Debian 9	 rstudio-1.2.5033-amd64-debian.tar.gz	156.11 MB

Você terá duas opções:

- a) Se você tiver acesso administrador, baixe a versão que está na lista de *All Installers*. Em seguida a instalação será bem simples: fazer o download, abrir o instalador e seguir as instruções, clicando no botão “Avançar”.
- b) Se você não tiver acesso de administrador, faça o download da versão que está na lista *Zip/Tarballs*. (veja imagem acima)

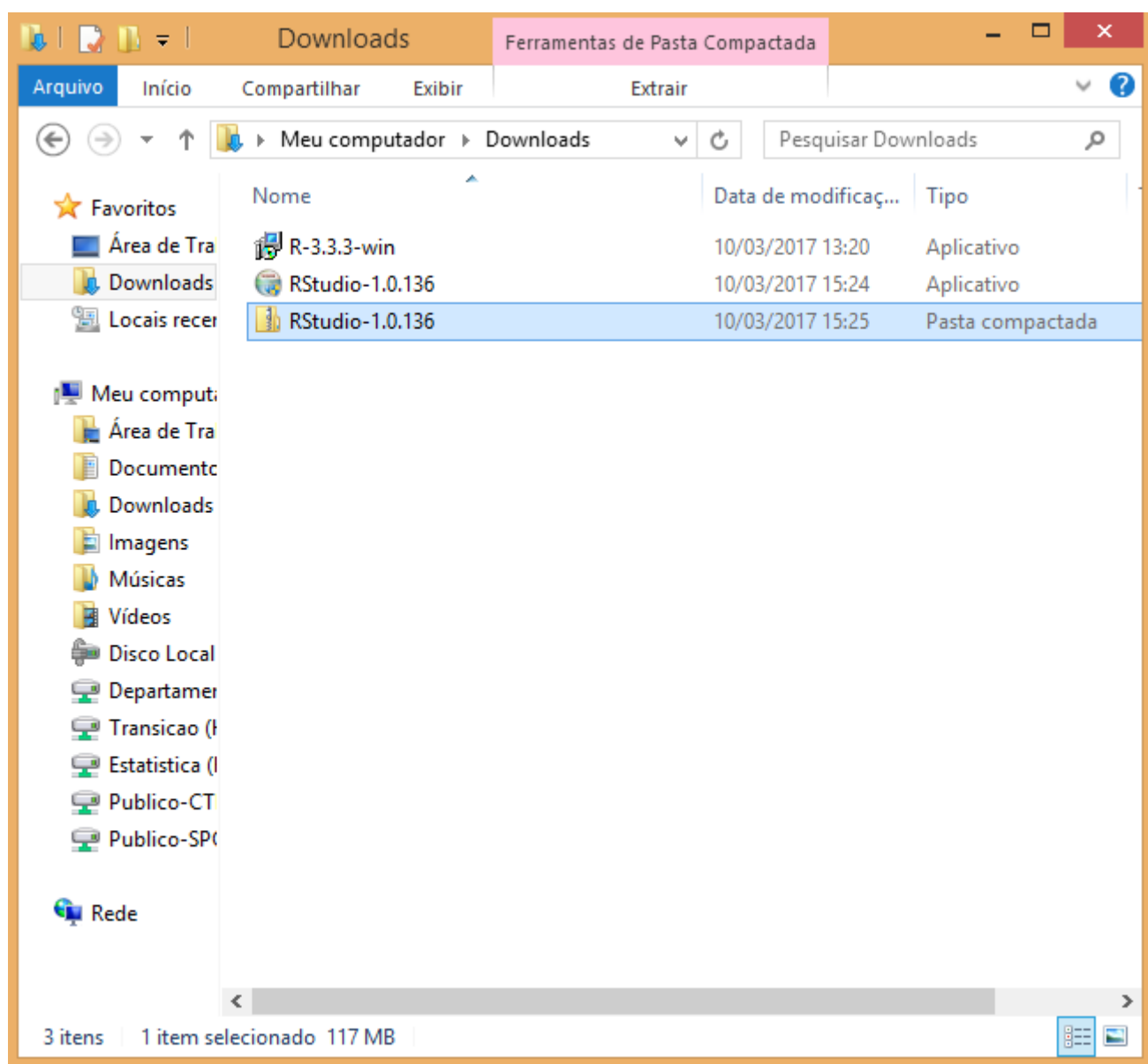
1.2.1.1 Instalando se você for administrador

2a. Clique duas vezes no arquivo que você baixou da página do RStudio (ver imagem abaixo) e siga as instruções de instalação.

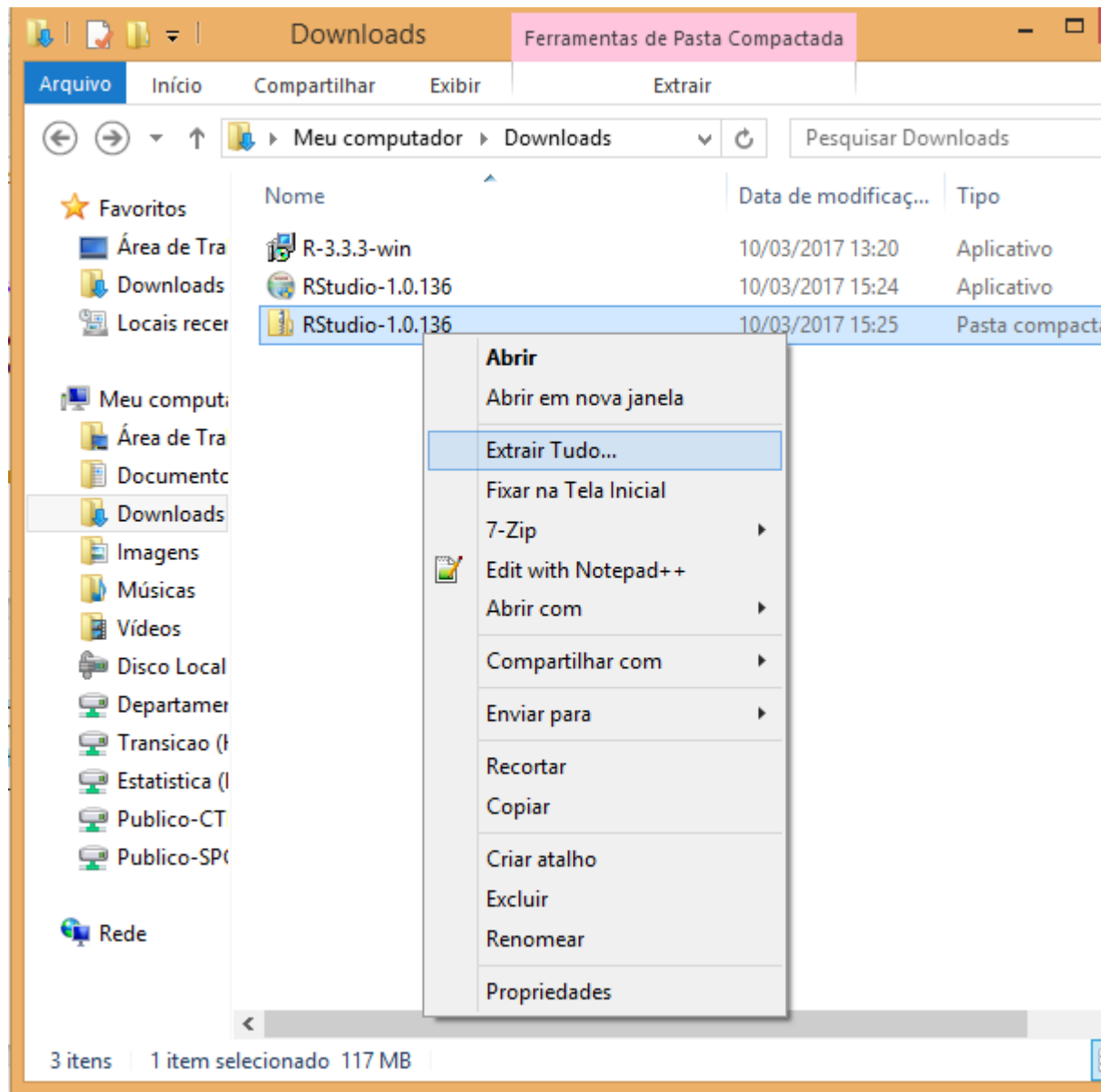


1.2.1.2 Instalação se você não for administrador

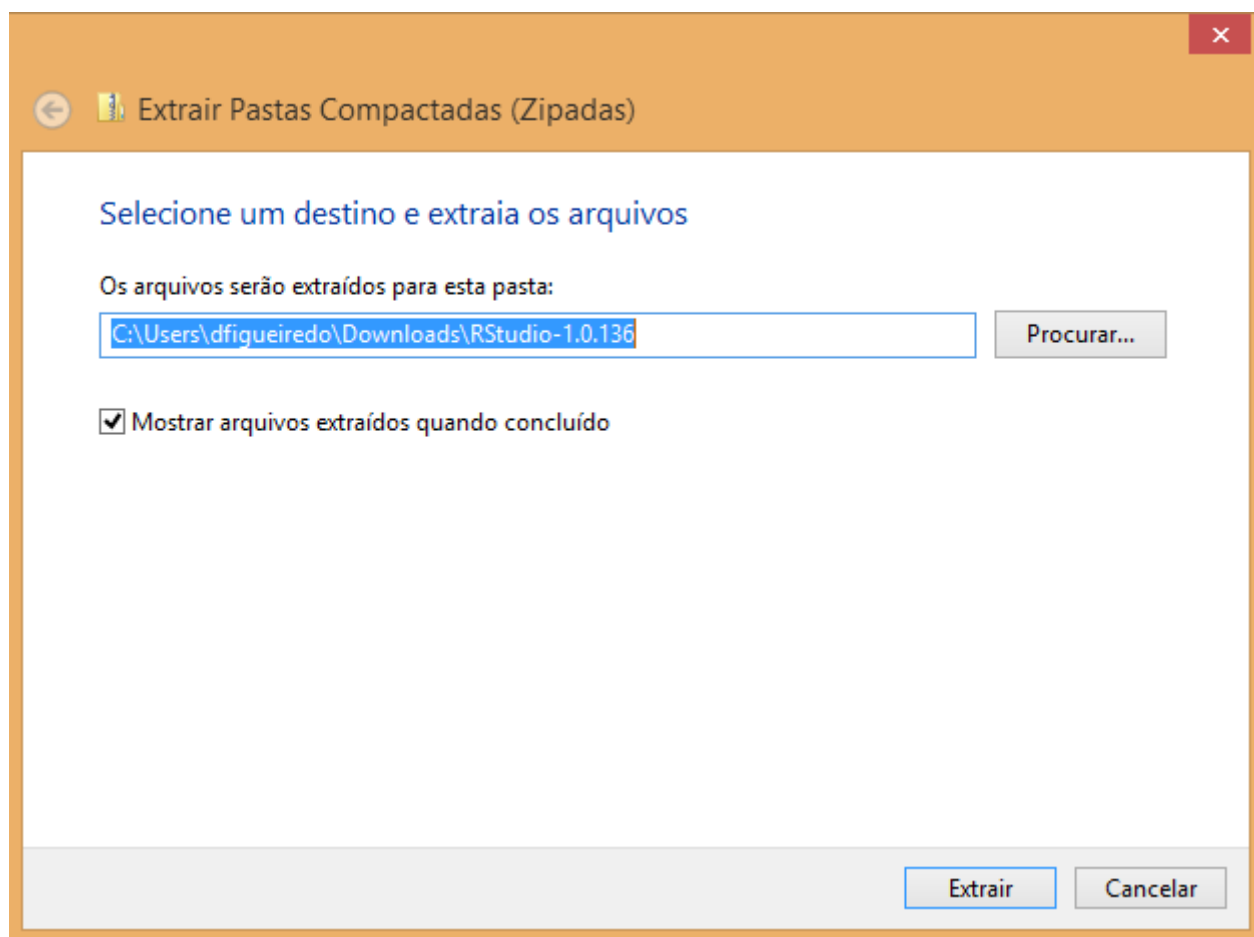
2b. Se você não for administrador, você deve ter feito o download de um arquivo do tipo *.zip*, que contém o código do RStudio. É o arquivo selecionado na imagem abaixo.



3. Clique com o botão direito neste arquivo e depois em *Extrair Tudo* conforme a imagem.

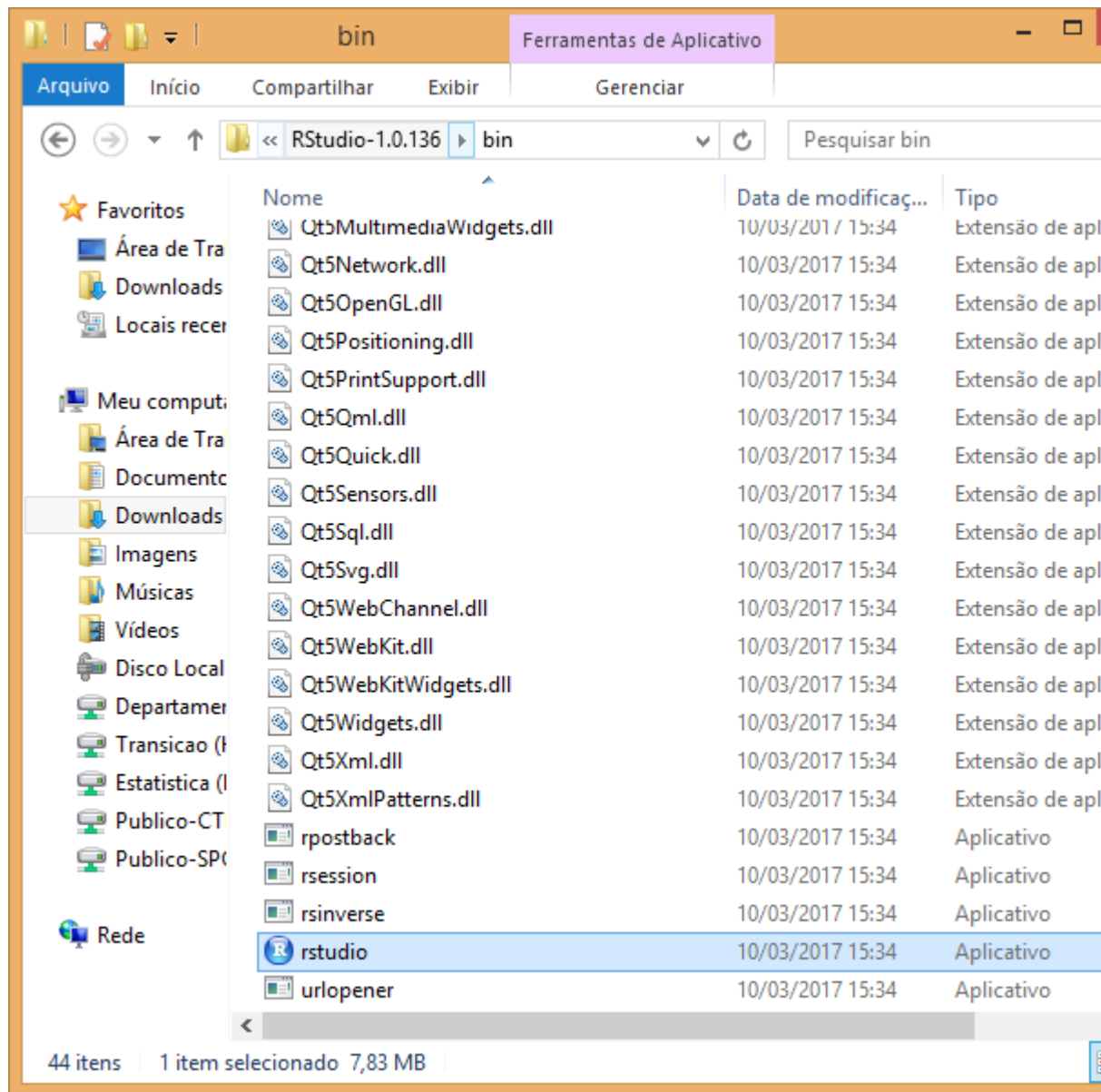


4. Você verá uma tela como a imagem a seguir. Não mude nada e clique em extrair. Espere o Windows completar a extração.



5. Agora, na pasta *Downloads*, a pasta que deixamos como local de extração (imagem acima), você terá uma pasta chamada: **RStudio-x.x.x**, em que x.x.x é o número da versão mais recente disponível. Abra essa pasta e entre na subpasta com nome **bin**.
6. Procure pelo arquivo chamado **rstudio** e clique duas vezes. Isso abrirá o RStudio. Recomendo fixar o programa na barra de tarefas para não ter que ficar procurando nessa pasta sempre que quiser abri-lo.

Observação: se você excluir a pasta que extraímos, o RStudio irá parar de funcionar.













1.2.2 No Linux

1. Acesse este link.
2. Clique no link referente à distribuição que você utiliza para fazer o download do arquivo de instalação.

All Installers

Linux users may need to [import RStudio's public code-signing key](#) prior to installing.

RStudio 1.2 requires a 64-bit operating system. If you are on a 32 bit system, you can use RStudio 1.1.

OS	Download
Windows 10/8/7	 RStudio-1.2.5033.exe
macOS 10.12+	 RStudio-1.2.5033.dmg
Ubuntu 14/Debian 8	 rstudio-1.2.5033-amd64.deb
Ubuntu 16	 rstudio-1.2.5033-amd64.deb
Ubuntu 18/Debian 10	 rstudio-1.2.5033-amd64.deb
Fedora 19/Red Hat 7	 rstudio-1.2.5033-x86_64.rpm
Fedora 28/Red Hat 8	 rstudio-1.2.5033-x86_64.rpm
Debian 9	 rstudio-1.2.5033-amd64.deb
SLES/OpenSUSE 12	 rstudio-1.2.5033-x86_64.rpm
OpenSUSE 15	 rstudio-1.2.5033-x86_64.rpm

1.3 Instalação de pacotes

Um pacote é um conjunto de funções que têm como objetivo resolver um problema específico. São eles que deixam o R poderoso, capaz de enfrentar qualquer tarefa de análise de dados. Assim, fique bastante à vontade para instalar e atualizar muitos e muitos pacotes ao longo da sua experiência com o R.

O legal é que qualquer pessoa pode fazer um novo pacote e disponibilizar para a comunidade, o que acelera bastante o desenvolvimento da ferramenta. Difícilmente você vai fazer uma análise apenas com as funções básicas do R e dificilmente não vai existir um pacote com as funções que você precisa.

Existem três principais maneiras de instalar pacotes. Em ordem de frequência, são:

- Via CRAN (Comprehensive R Archive Network): `install.packages("nome-do-pacote")`.
- Via Github: `devtools::install_github("nome-do-repo/nome-do-pacote")`.
- Via arquivo .zip/.tar.gz: `install.packages("C:/caminho/nome-do-pacote.zip", repos = NULL)`.

Se você é usuário do linux, para conseguir instalar alguns pacotes, talvez você precise instalar dependências do sistema no terminal do computador manualmente. Por exemplo, se você quer instalar o pacote `devtools` no R, será necessário ter as dependências `curl`, `openssl`, `httr` e `git2r`.

Para saber quais dependências instalar basta usar a função `install.packages("nome_do_pacote")` e ler as mensagens de erro que aparecerão.

1.3.1 Via CRAN

Instale pacotes que não estão na sua biblioteca usando a função `install.packages("nome_do_pacote")`. Por exemplo:

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

E, de agora em diante, não precisa mais instalar. Basta carregar o pacote com `library(magrittr)`.

Escreva `nome_do_pacote::nome_da_funcao()` se quiser usar apenas uma função de um determinado pacote. O operador `::` serve para isso. Essa forma também é útil quando se tem duas funções com o mesmo nome e precisamos garantir que o código vá usar a função do pacote correto.

1.3.2 Via Github

Desenvolvedores costumam disponibilizar a última versão de seus pacotes no Github, e alguns deles sequer estão no CRAN. Mesmo assim ainda é possível utilizá-los instalando diretamente pelo github. O comando é igualmente simples:

```
devtools::install_github("rstudio/shiny")
```

Apenas será necessário o username e o nome do repositório (que geralmente tem o mesmo nome do pacote). No exemplo, o username foi “rstudio” e o repositório foi “shiny”.

Se você não é familiar com o github, não se preocupe! Os pacotes disponibilizados na plataforma geralmente têm um README cuja primeira instrução é sobre a instalação. Se não tiver, provavelmente este pacote não te merece! =)

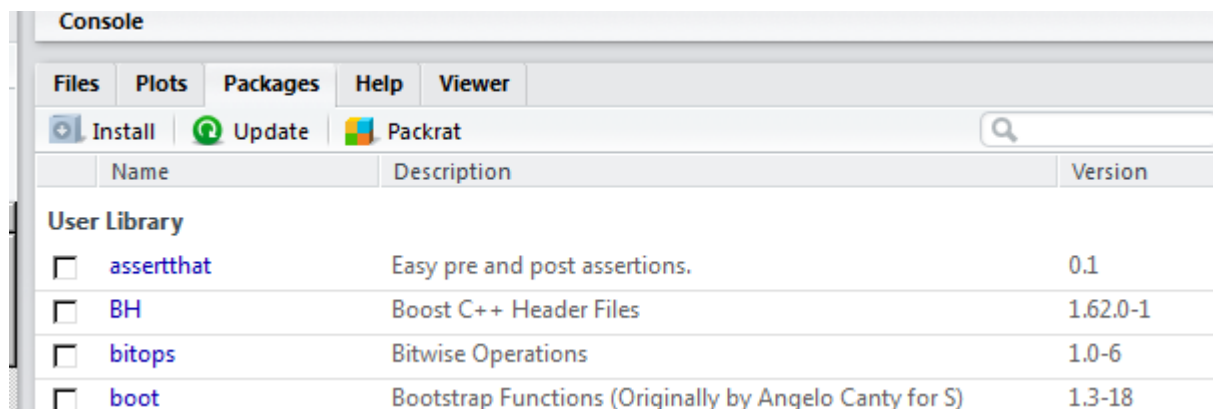
1.3.3 Via arquivo .zip ou .tar.gz

Se você precisar instalar um pacote que está zipado no seu computador (ou em algum servidor), utilize o seguinte comando:

```
install.packages("C:/caminho/para/o/arquivo/zipado/nome-do-pacote.zip", repos = NULL)
```

É semelhante a instalar pacotes via CRAN, com a diferença que agora o nome do pacote é o caminho inteiro até o arquivo. O parâmetro `repos = NULL` informa que estamos instalando a partir da máquina local.

A aba **Packages** do RStudio também ajuda a administrar os seus pacotes.



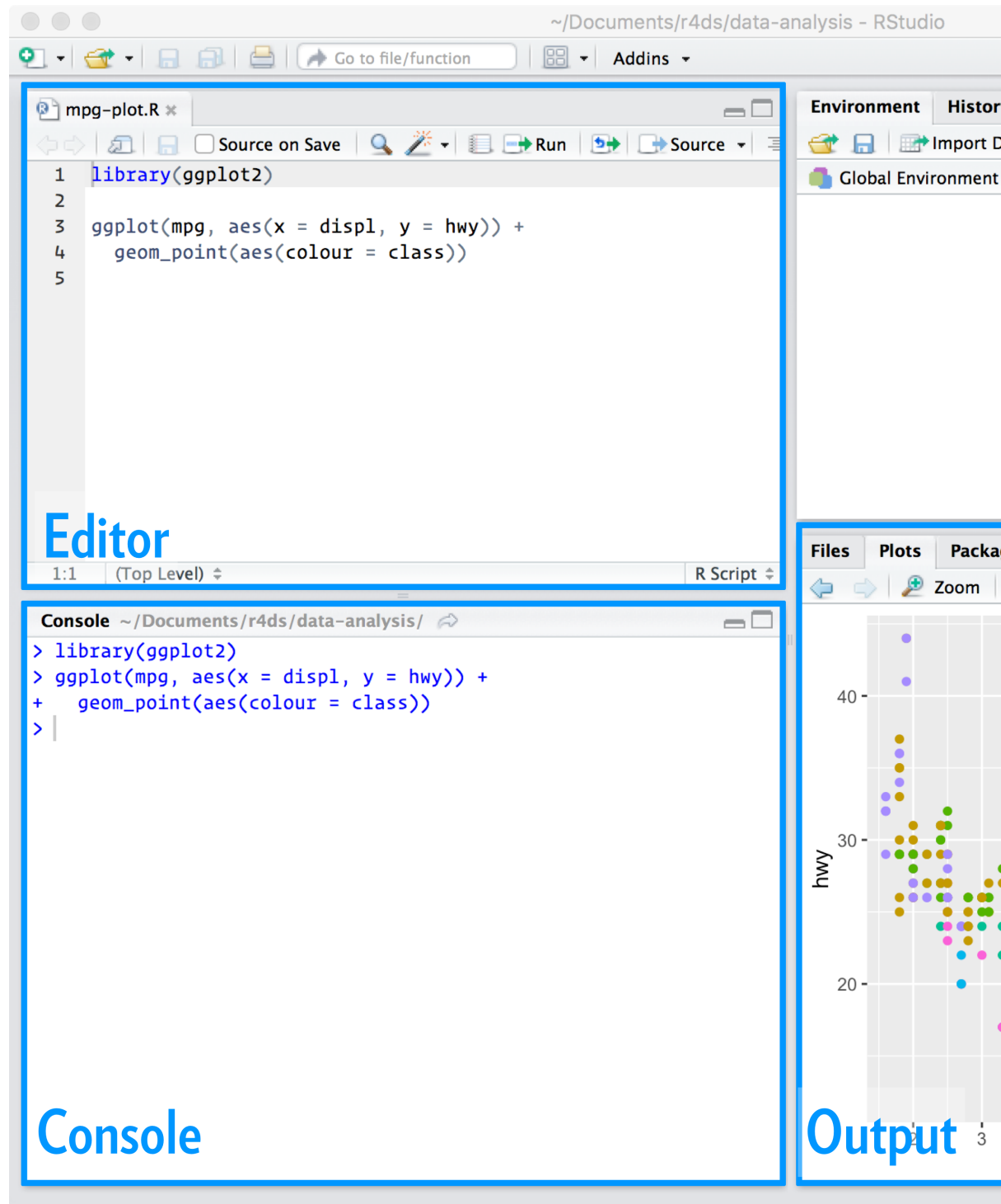
Chapter 2

RStudio

O RStudio é o melhor ambiente disponível para programação em R. Nesta seção, apresentaremos as suas principais funcionalidades, que te deixarão muito mais eficiente na hora de programar em R.

2.1 Telas

Ao abrir o RStudio, você verá 4 quadrantes. Observe a figura abaixo.



Esses quadrantes representam o **editor**, o **console**, o **environment** e o **output**. Eles vêm nesta ordem, e depois você pode organizá-los da forma que preferir.

Listamos abaixo as funções dos principais painéis:

- **Editor/Scripts:** é onde escrevemos nossos códigos.
- **Console:** é onde rodamos o código e recebemos as saídas. O R vive aqui!
- **Environment:** painel com todos os objetos criados na sessão.
- **Files:** mostra os arquivos no diretório de trabalho. É possível navegar entre diretórios.
- **Plots:** painel onde os gráficos serão apresentados.
- **Help:** janela onde a documentação das funções serão apresentadas.
- **History:** painel com um histórico dos comandos rodados.

2.2 Atalhos

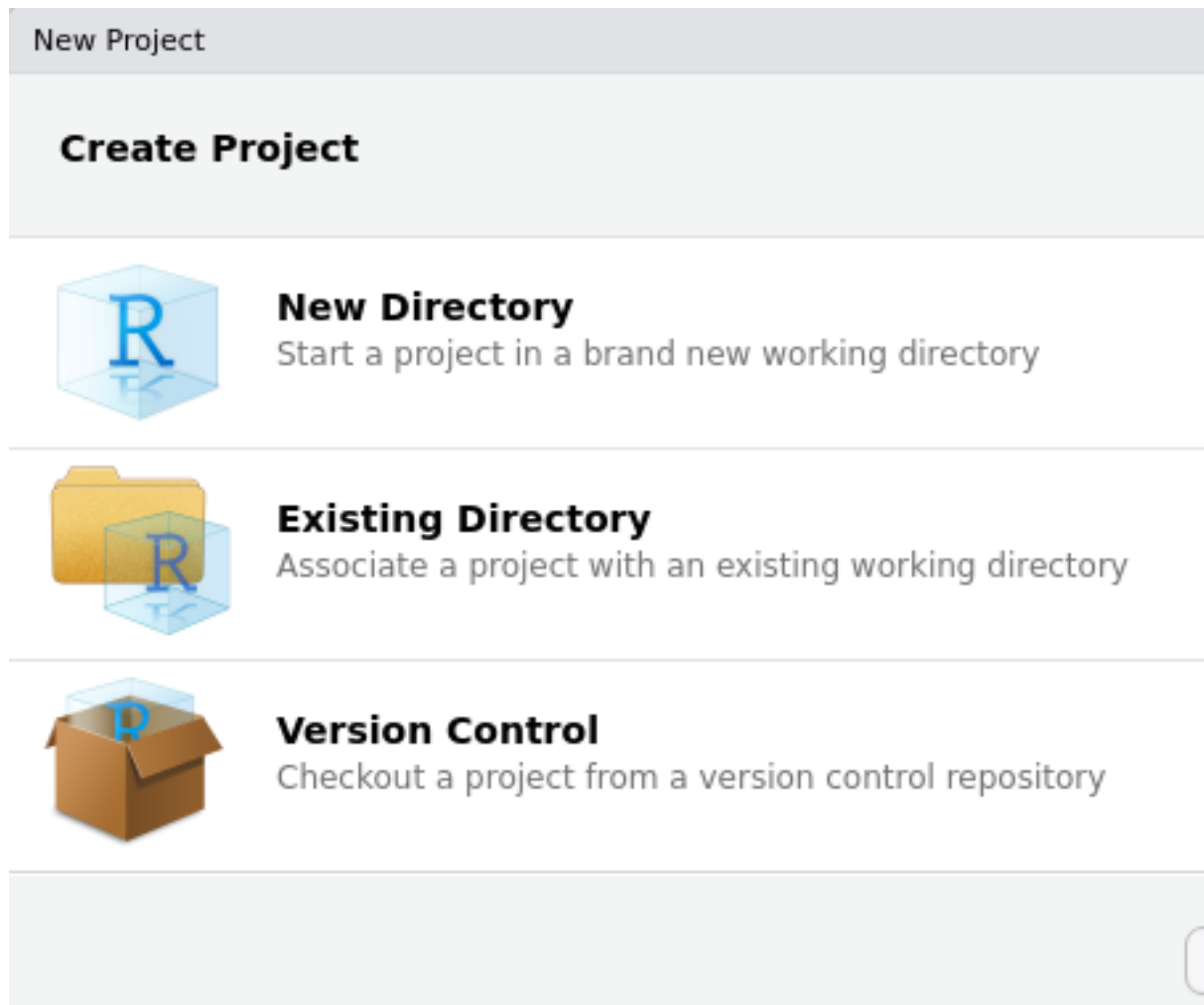
Conhecer atalhos ajuda bastante quando estamos programando no RStudio. Veja os principais:

- **CTRL+ENTER:** roda a linha selecionada no script. O atalho mais utilizado.
- **ALT+-:** (<-) sinal de atribuição. Você usará o tempo todo.
- **CTRL+SHIFT+M:** (%>%) operador *pipe*. Guarde esse atalho, você usará bastante.
- **CTRL+1:** altera cursor para o script.
- **CTRL+2:** altera cursor para o console.
- **CTRL+ALT+I:** cria um chunk no R Markdown.
- **ALT+SHIFT+K:** janela com todos os atalhos disponíveis.

2.3 Projetos

Uma funcionalidade importante é a criação de projetos, permitindo dividir o trabalho em múltiplos ambientes, cada um com o seu diretório, documentos e *workspace*.

Para criar um projeto, clique em **New Project...** no Menu **File**. Na caixa de diálogo que aparecerá, clique em **New Directory** para criar o projeto em uma nova pasta ou **Existing Directory** para criar em uma pasta existente. Se você tiver o **Git** instalado, você também pode usar projetos para conectar com repositórios do Github e outras plataformas de desenvolvimento. Para isso, basta clicar em **Version Control**.



A seguir, apresentamos algumas estruturas para a organização de um projeto.

Estrutura 1. Por extensão de arquivo.

```
nome_do_projeto/  
- .Rprofile      # códigos para rodar assim que abrir o projeto  
- R/             # Código R, organizado com a-carrega.R, b-prepara bd.R, c-vis.R, d-mod  
- RData/         # Dados em formato .RData  
- csv/           # Dados em .csv  
- png/           # gráficos em PNG  
- nome_do_projeto.Rproj
```

Estrutura 2. Típico projeto de análise estatística.

```
project/
- README.Rmd      # Descrição do pacote
- set-up.R        # Pacotes etc
- R/              # Código R, organizado com 0-load.R, 1-tidy.R, 2-vis.R, ...
- data/           # Dados (estruturados ou não)
- figures/        # gráficos (pode ficar dentro de output/)
- output/         # Relatórios em .Rmd, .tex etc
- project.Rproj
```

Estrutura 3. Pacote do R (avançado).

```
project/
- README.md       # Descrição do pacote
- DESCRIPTION     # Metadados estruturados do pacote e dependências
- NAMESPACE      # importações e exportações do pacote
- vignettes/     # Relatórios em .Rmd
- R/             # Funções do R
- data/          # Dados estruturados (tidy data)
- data-raw/      # Dados não estruturados e arqs 0-load.R, 1-tidy.R, 2-vis.R, ...
- project.Rproj
```

2.4 Cheatsheets

Se você clicar no Menu **Help** e então em Cheatsheets, você verá algumas opções de *folhas de cola*, um guia de consulta rápido para diversos pacotes e para o próprio RStudio.

RStudio IDE : : CHEAT SHEET

Documents and Apps

Open Shiny, R Markdown, knitr, Sweave, LaTeX, Rd files and more in Source Pane

Check spelling, Render output, Choose output format, Choose output location, Insert code chunk

Jump to previous chunk, Jump to next chunk, Run selected lines, Publish to server, Show file outline

Access markdown guide at **Help > Markdown Quick Reference**

Jump to chunk, Set knitr options, Run this and all previous code chunks, Run this code chunk

RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app

Run app, Choose location to shinyapps.io view app, Publish to shinyapps.io or server, Manage publish accounts

Write Code

Navigate tabs, Open in new window, Save, Find and replace, Compile as notebook, Run selected code

Re-run previous code, Source with or without Echo, Show file outline

Multiple cursors/column selection with **Alt + mouse drag**

Code diagnostics that appear in the margin. Hover over diagnostic symbols for details.

Syntax highlighting based on your file's extension

Tab completion to finish function names, file paths, arguments, and more

Multi-language code snippets to quickly use common blocks of code

Jump to function in file, Change file type

Working Directory, Maximize, minimize panes, Press **↑** to see command history, Drag pane boundaries

R Support

Import data with wizard, History of past commands to run/copy

Load workspace, Save workspace, Delete workspace

Choose environment to display list of parent environments

Displays saved objects by type with short description

Create folder, Upload file, Delete file, Rename file

Path to displayed directory

A File browser keyed to your work. Click on file or directory name to

Debug Mode

Open with **debug()**, **browser()**, or a breakpoint. RStudio will open the debugger mode when it encounters a breakpoint while executing code.

Launch debugger mode from origin of error

Open traceback to examine the functions that R called before the error occurred

Click next to line number to add/remove a breakpoint

Highlighted line shows where execution has paused

Run commands in environment where execution has paused

Examine variables in executing environment

Select function in traceback to debug

Step through code one line at a time

Step into and out of functions to run

Resume execution mode

Quit debug

Version Control

Turn on at **Tools > Project > Version Control**

Stage files, Show file diff, Commit

Added, Deleted, Modified, Renamed, Untracked

Open shell to type commands

Package Writing

File > New Project > New Directory > R Project

Turn project into package, Enable roxygen documentation on **Tools > Project Options > Build**

Roxygen guide at **Help > Roxygen Quick Reference**



Chapter 3

R Básico

3.1 Pedindo Ajuda

A linguagem R é bem intuitiva. É possível fazer bastante coisa à base da tentativa e erro. Além disso, grande parte do conhecimento é escalável, isto é, aprender a utilizar uma função é meio caminho andado para aprender todas as outras funções que operam de forma semelhante. Essa ideia é um dos princípios por trás do `tidyverse`.

No entanto, a intuição não infalível, e recorrentemente vamos precisar de ajuda para rodar alguma função ou descobrir como fazer alguma tarefa no R. Felizmente, a comunidade R é bem ativa e existem vários lugares para buscar respostas. Nesta seção, vamos apresentar as principais maneiras algumas dessas maneiras.

No R, há quatro principais entidades para se pedir ajuda:

- Help/documentação do R (comandos `help(nome_da_funcao)` ou `?nome_da_funcao`)
- Google
- Stack Overflow
- Coleguinha

A busca por ajuda é feita preferencialmente, mas não necessariamente, na ordem acima.

3.1.1 Documentação do R

A documentação do R serve para você aprender a usar uma determinada função.

```
?mean  
help(mean)
```

Cinco dicas:

1. Os exemplos no final são particularmente úteis.
2. Leia a seção **Usage** para ter noção de como usar a função.
3. Os parâmetros da função estão descritos em **Arguments**.
4. Caso essa função não atenda às suas necessidades, a seção **See Also** sugere funções relacionadas.
5. Alguns pacotes possuem tutorias de uso mais completos. Esses textos são chamados de **vignettes** e podem ser acessados com a função `vignette(package = 'nomeDoPacote')`. Por exemplo, `vignette(package = 'dplyr')`. Depois de ver a lista de artigos, escolha um nome e rode `vignette(topic = 'nome', package = 'nomeDoPacote')`. Por exemplo, `vignette(topic = 'introduction', package = 'dplyr')`.
6. Bases de dados presentes em pacotes também têm documentação, e geralmente é possível encontrar o significado de cada variável nela.

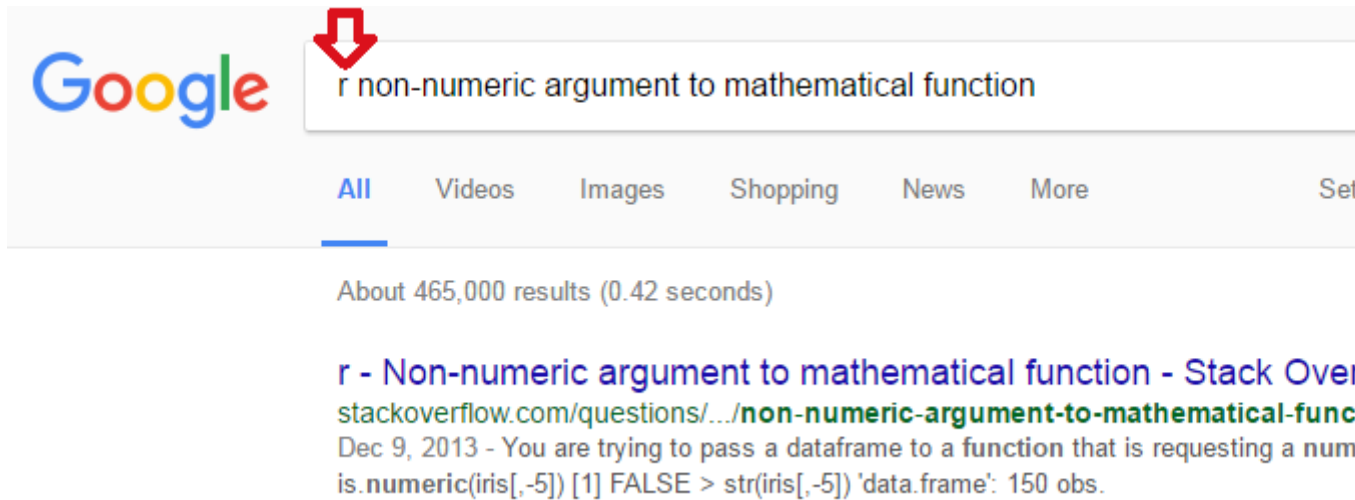
3.1.2 Google

Há uma comunidade gigantesca de usuários de R gerando diariamente uma infinidade de conteúdos e discussões. Não raramente, você irá encontrar discussões sobre o seu problema simplesmente jogando o seu erro no Google. Essa deve ser sua primeira tentativa! Pesquisas em inglês aumentam consideravelmente a chance de encontrar uma resposta.

Exemplo (repare no ‘r’ adicionado na busca, isso ajuda bastante):

```
log("5")
```

```
## Error in log("5"): non-numeric argument to mathematical function
```



3.1.3 Stack Overflow

O Stack Overflow e o Stack Overflow em Português são sites de Pergunta e Resposta amplamente utilizados por todas as linguagens de programação, e o R é uma delas. Nos EUA, chegam até a usar a reputação dos usuários como diferencial no currículo!

Provavelmente o Google lhe indicará uma página deles quando você estiver procurando ajuda. E quando todas as fontes possíveis de ajuda falharem, o Stack Overflow lhe dará o espaço para **criar sua própria pergunta**.

Um ponto importante: como fazer uma **boa** pergunta no Stack Overflow?

No site, existe um tutorial com uma lista de boas práticas, que se encontra aqui. Resumindo, as principais dicas são

- ser conciso;
- ser específico;
- ter mente aberta; e
- ser gentil.

Porém, no caso do R, há outro requisito que vai aumentar muito sua chance de ter uma boa resposta: **exemplinho minimal e reprodutível**.

- Ser **minimal**: usar bancos de dados menores e utilizar pedaços de códigos apenas suficientes para apresentar o seu problema. Não precisa de banco de dados de um milhão de linhas e nem colocar o seu código inteiro para descrever a sua dúvida.

- Ser **reprodutível**: o seu código deve rodar fora da sua máquina. Se você não fornecer uma versão do seu problema que rode (ou que imite seu erro), as pessoas vão logo desistir de te ajudar. Por isso, nunca coloque bancos de dados que só você tem acesso. Use bancos de dados que já vem no R ou disponibilize um exemplo (possivelmente anonimizado) em `.csv` na web para baixar. E se precisar utilizar funções diferentes, coloque as `library`'s correspondentes.

3.1.4 R Markdown

O R Markdown é um tipo de documento especial que contém tanto textos em *markdown* quanto *chunks* de códigos em R, tudo escrito no mesmo lugar.

O *markdown* nada mais é do que um documento de texto com alguns padrões básicos de formatação, como negrito, itálico, títulos, subtítulos, itens e referências cruzadas. Já os *chunks* são pedaços de códigos em R encapsulados por três crases (“”). Os códigos são executados sempre que o documento é processado.

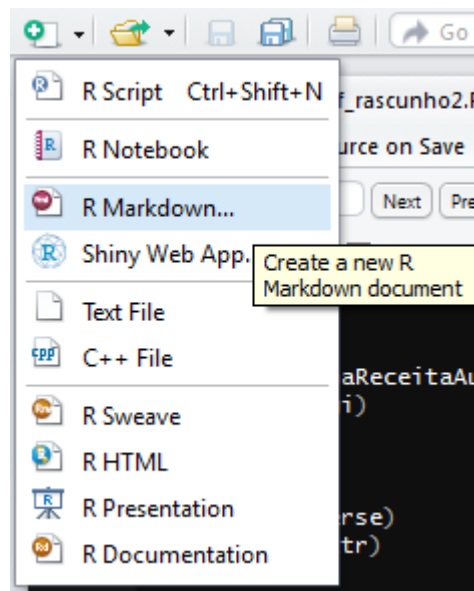
```
## ```{r}
## isto é um chunk.
## ```
```

Este site foi escrito em R Markdown. Toda vez que aparecer exemplos de código de R, havia um chunk no `.Rmd` original.

Para produção de relatórios, o R Markdown possui algumas vantagens, como:

1. **Simplicidade e foco.** Permite ao usuário o foco na análise e não na formatação do documento.
2. **Versátil.** Pode ser utilizado para gerar documentos em **LaTeX**, **Word**, **HTML** e apresentações em **beamer**, **pptx** e **HTML**. Pode ainda gerar sites, livros, dissertações de mestrado e até mesmo dashboards interativos.
3. **Reprodutível.** O R Markdown nada mais é que um arquivo de texto. Além disso, ele tenta te obrigar a fazer o documento mais autocontido possível. Assim, um documento `.Rmd` é fácil de compartilhar e de ser utilizado pelo receptor. Lembre-se, o receptor pode ser o futuro você! Vale enfatizar que a reprodutibilidade é considerada como um dos princípios fundamentais da ciência. Então, só de usar R Markdown, você já está colaborando com o método científico. :)
4. **Flexível.** É possível configurar e criar *templates* de análises para quaisquer tipos de aplicações e clientes. Os textos podem ser parametrizados por números que variam de versão para versão, mensalmente, por exemplo, tudo escrito somente em R.

Criar um R Markdown novo no RStudio é fácil. Clique no botão de criar arquivo e selecione R Markdown.



Para detalhes sobre como utilizar o R Markdown, leia esta seção do R4DS e o tutorial do RStudio.

3.2 R como calculadora

Pelo console, é possível executar qualquer comando do R.

```
1:30
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25  
## [26] 26 27 28 29 30
```

Esse comando é uma forma simplificada de criar um vetor de inteiros de 1 a 30. Os números que aparecem entre colchetes ([1] e [24]) indicam o índice do primeiro elemento impresso em cada linha.

Quando compilamos? Quem vem de linguagens como o C ou Java espera que seja necessário compilar o código em texto para o código das máquinas (geralmente um código binário). No R, isso não é necessário. O R é uma linguagem de programação dinâmica que interpreta o seu código enquanto você o executa.

Tente jogar no console: $2 * 2 - (4 + 4) / 2$.

Pronto! Com essa simples expressão você já é capaz de pedir ao R para fazer qualquer uma das quatro operações aritméticas básicas. A seguir, apresentamos uma lista resumindo como fazer as principais operações no R.

```
# adição
1 + 1

## [1] 2

# subtração
4 - 2

## [1] 2

# multiplicação
2 * 3

## [1] 6

# divisão
5 / 3

## [1] 1.666667

# potência
4 ^ 2

## [1] 16

# resto da divisão de 5 por 3
5 %% 3

## [1] 2

# parte inteira da divisão de 5 por 3
5 %/% 3

## [1] 1
```

Além do mais, as operações e suas precedências são mantidas como na matemática, ou seja, divisão e multiplicação são calculadas antes da adição e subtração. E os parênteses nunca são demais!

Uma outra forma de executar uma expressão é escrever o código no **script** e teclar **Ctrl + Enter** ou **Ctrl + R**. Assim, o comando é enviado para o **console**, onde é diretamente executado.

Se você digitar um comando incompleto, como `5 +`, e apertar **Enter**, o R mostrará um `+`, o que não tem nada a ver com somar alguma coisa. Isso significa que o R está esperando que você complete o seu comando. Termine o seu comando ou aperte **Esc** para recomeçar.

```
> 5 -
+
+ 5
[1] 0
```

Se você digitar um comando que o R não reconhece, ele retornará uma mensagem de erro.

NÃO ENTRE EM PÂNICO!

Ele só está avisando que não conseguiu interpretar o comando. Você pode digitar outro comando normalmente em seguida.

```
> 5 % 5
Error: unexpected input in "5 % 5"
> 5 - 5
[1] 0
```

3.3 Objetos e Classes

O R te permite salvar dados dentro de um objeto. Para isso, utilizamos o operador `<-`.

No exemplo abaixo, salvamos o valor 1 em `a`. Sempre que o R encontrar o símbolo `a`, ele vai substituí-lo por 1.

```
a <- 1
a
```

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

Atenção! O R diferencia letras maiúsculas e minúsculas, isto é, “a” é considerado um objeto diferente de “A”.

3.3.1 Objetos atômicos

Existem cinco classes básicas ou “atômicas” no R:

- character
- numeric
- integer
- complex
- logical

Veja alguns exemplos:

```
# characters
"a"
"1"
"positivo"
>Error: objeto x não encontrado
```

```
# numeric

1
0.10
0.95
pi

# integer

1L
5L
10L

# complex (raramente utilizado para análise de dados)

2 + 5i

# logical

TRUE
FALSE
```

3.3.2 Vetores

Vetores no R são os objetos mais simples que podem guardar objetos atômicos.

```
vetor1 <- c(1, 2, 3, 4)
vetor2 <- c("a", "b", "c")

vetor1
```

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

```
vetor2
```

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

De forma bastante intuitiva, você pode fazer operações com vetores.

```
vetor1 - 1
```

```
## [1] 0 1 2 3
```

Quando você faz `vetor1 - 1`, o R subtrai 1 de cada um dos elementos do vetor. O mesmo acontece quando você faz qualquer operação aritmética com vetores no R.

```
vetor1 / 2  
vetor1 * 10
```

Você também pode fazer operações que envolvem mais de um vetor:

```
vetor1 * vetor1
```

```
## [1] 1 4 9 16
```

Neste caso, o R irá alinhar os dois vetores e multiplicar elemento por elemento. Isso pode ficar um pouco confuso quando os dois vetores não possuem o mesmo tamanho:

```
vetor2 <- 1:3  
vetor1 * vetor2
```

```
## Warning in vetor1 * vetor2: comprimento do objeto maior não é múltiplo do  
## comprimento do objeto menor
```

```
## [1] 1 4 9 4
```

O R alinhou os dois vetores e, como eles não possuíam o mesmo tamanho, foi repetindo o vetor menor até completar o vetor maior. Esse comportamento é chamado de **reciclagem** e é útil para fazer operações elemento por elemento (vetorizadamente), mas às vezes pode ser confuso. Com o tempo, você aprenderá a se aproveitar dele.

3.3.3 Classes

Para saber a classe de um objeto, você pode usar a função `class()`.

```
x <- 1  
class(x)
```

```
## [1] "numeric"
```

```
y <- "a"  
class(y)
```

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

```
z <- TRUE  
class(z)
```

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

Um vetor tem sempre a mesma classe dos objetos que guarda.

```
class(vetor1)
```

```
## [1] "numeric"
```

```
class(vetor2)

## [1] "integer"
```

3.3.4 Coerção

3.3.4.1 Misturando objetos

Vetores são homogêneos. Os elementos de um vetor são sempre da mesma classe. Ou todos são numéricos, ou são todos character, ou todos são lógicos etc. Não dá para ter um número e um character no mesmo vetor, por exemplo.

Se colocarmos duas ou mais classes diferentes dentro de um mesmo vetor, o R vai forçar que todos os elementos passem a pertencer à mesma classe. O número 1.7 viraria "1.7" se fosse colocado ao lado de um "a".

```
y <- c(1.7, "a") # character
y <- c(TRUE, 2)  # numeric
y <- c(TRUE, "a") # character
```

A ordem de precedência é:

DOMINANTE — character > complex > numeric > integer > logical — **RECESSIVO**

3.3.4.2 Forçando classes explicitamente

Você pode coagir um objeto a ser de uma classe específica com as funções `as.character()`, `as.numeric()`, `as.integer()` e `as.logical()`. É equivalente à função `convert()` do SQL.

```
x <- 0:4
class(x)
```

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

```
as.numeric(x)
```

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

```
as.logical(x)
```

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

```
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4"
```

Se o R não entender como coagir uma classe na outra, ele soltará um **warning** informado que colocou NA no lugar.

```
x <- c("a", "b", "c")
as.numeric(x)
```

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

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

Observação. O NA tem o mesmo papel que o null do SQL. Porém, há um NULL no R também, com diferenças sutis que vamos abordar mais adiante.

3.3.5 data.frame

Um `data.frame` é o mesmo que uma tabela do SQL ou um spreadsheet do Excel, por isso são objetos muito importantes. Usualmente, seus dados serão importados para um objeto `data.frame`. Em grande parte do curso, eles serão o principal objeto de estudo.

Os `data.frame`'s são listas especiais em que todos os elementos possuem o **mesmo comprimento**. Cada elemento dessa lista pode ser pensado como uma coluna da tabela. Seu comprimento representa o número de linhas.

Já que são listas, essas colunas podem ser de classes diferentes. Essa é a grande diferença entre `data.frame`'s e matrizes. Algumas funções úteis para trabalhar com `data.frame`'s :

- `head()` - Mostra as primeiras 6 linhas.
- `tail()` - Mostra as últimas 6 linhas.
- `dim()` - Número de linhas e de colunas.
- `names()` - Os nomes das colunas (variáveis).
- `str()` - Estrutura do `data.frame`. Mostra, entre outras coisas, as classes de cada coluna.
- `cbind()` - Acopla duas tabelas lado a lado.
- `rbind()` - Empilha duas tabelas.

O exemplo abaixo mostra que uma lista pode virar `data.frame` apenas se todos os elementos tiverem o mesmo comprimento.

```
minha_lista <- list(x = c(1, 2, 3), y = c("a", "b"))
as.data.frame(minha_lista)
```

```
## Error in (function (... , row.names = NULL, check.rows = FALSE, check.names = TRUE, : arguments
```

```
minha_lista <- list(x = c(1, 2, 3), y = c("a", "b", "c"))
as.data.frame(minha_lista)
```

```
##      x y
```

```
## 1 1 a
## 2 2 b
## 3 3 c
```

3.3.6 Matrizes

Matrizes são vetores com duas dimensões (e por isso só possuem elementos de uma mesma classe).

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m
```

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

```
dim(m) # função dim() retorna a dimensão do objeto.
```

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

Repare que os números de 1 a 6 foram dispostos na matriz coluna por coluna (*column-wise*), ou seja, preenchendo de cima para baixo e depois da esquerda para a direita.

Operações úteis

```
m[3, ] # seleciona a terceira linha
m[, 2] # seleciona a segunda coluna
m[1, 2] # seleciona o primeiro elemento da segunda coluna
t(m)    # matriz transposta
m %*% n  # multiplicação matricial
solve(m) # matriz inversa de m
```

3.3.7 Fatores

Fatores podem ser vistos como vetores de inteiros que possuem rótulos (levels).

```
sexo <- c("M", "H", "H", "H", "M", "M", "H")
fator <- as.factor(sexo)
fator
```

```
## [1] M H H H M M H
## Levels: H M
```

```
as.numeric(fator)
```

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


Eles são úteis para representar uma variável categórica (nominal e ordinal). Na modelagem, eles serão tratados de maneira especial em funções como `lm()` e `glm()`.

A função `levels()` retorna os rótulos do fator:

```
levels(fator)
```

```
## [1] "H" "M"
```

A ordem das categorias de um fator pode importar. Como exemplo, temos as caselas de referência de modelos estatísticos e a ordem das barras de um gráfico. Para ajudar nesta tarefa, consulte o pacote `forcats`.

Um erro comum e desastroso. Quando um vetor de números está como `factor`, ao tentar transformá-lo em `numeric`, você receberá um vetor de inteiros que não tem nada a ver com os valores originais!

```
numeros <- factor(c("10", "55", "55", "12", "10", "-5", "-90"))
as.numeric(numeros)
```

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

```
#Por essa eu não esperava!
```

Para evitar isso, use `as.character()` antes de transformar para número.

```
as.numeric(as.character(numeros))
```

```
## [1] 10 55 55 12 10 -5 -90
```

```
# Agora está OK!
```

3.3.8 Listas

Listas são um tipo especial de vetor que aceita elementos de classes diferentes.

```
x <- list(1:5, "Z", TRUE, c("a", "b"))
x
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "Z"
##
## [[3]]
## [1] TRUE
##
## [[4]]
```

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

É um dos objetos mais importantes para armazenar dados e vale a pena saber manuseá-los bem. Existem muitas funções que fazem das listas objetos incrivelmente úteis.

Criamos uma lista com a função `list()`, que aceita um número arbitrário de elementos. Listas aceitam QUALQUER tipo de objeto. Podemos ter listas dentro de listas, por exemplo. Como para quase todas as classes de objetos no R, as funções `is.list()` e `as.list()` também existem.

Na lista `pedido` abaixo, temos `numeric`, `Date`, `character`, vetor de `character` e `list` contida em uma lista:

```
pedido <- list(pedido_id = 8001406,
              pedido_registro = as.Date("2017-05-25"),
              nome = "Athos",
              sobrenome = "Petri Damiani",
              cpf = "12345678900",
              email = "athos.damiani@gmail.com",
              qualidades = c("incrível", "impressionante"),
              itens = list(
                list(descricao = "Ferrari",
                     frete = 0,
                     valor = 500000),
                list(descricao = "Dolly",
                     frete = 1.5,
                     valor = 3.90)
              ),
              endereco = list(entrega = list(logradouro = "Rua da Glória",
                                              numero = "123",
                                              complemento = "apto 71"),
                              cobranca = list(logradouro = "Rua Jose de Oliveira Cout.",
                                              numero = "151",
                                              complemento = "5o andar")
              )
)
```

Operações úteis

```
pedido$cpf      # elemento chamado 'cpf'
pedido[1]       # nova lista com apenas o primeiro elemento
pedido[[2]]     # segundo elemento
pedido["nome"]  # nova lista com apenas o elemento chamado 'nome'
```

Certamente você se deparará com listas quando for fazer análise de dados com o R. Nos tópicos mais aplicados, iremos aprofundar sobre o tema. O pacote `purrr` contribui com funcionalidades incríveis para listas.

3.4 Valores especiais

Existem valores reservados para representar dados faltantes, infinitos, e indefinições matemáticas.

- **NA** (Not Available) significa dado faltante/indisponível. É o `null` do SQL ou o `.` do SAS. O **NA** tem uma classe, ou seja, podemos ter **NA** numeric, **NA** character etc.
- **NaN** (Not a Number) representa indefinições matemáticas, como $0/0$ e $\log(-1)$. Um **NaN** é um **NA**, mas a recíproca não é verdadeira.
- **Inf** (Infinito) é um número muito grande ou o limite matemático, por exemplo, $1/0$ e 10^{310} . Aceita sinal negativo **-Inf**.
- **NULL** representa a ausência de informação. Conceitualmente, a diferença entre **NA** e **NULL** é sutil, mas, no R, o **NA** está mais alinhado com os conceitos de estatística (ou como gostaríamos que os dados faltantes se comportassem em análise de dados) e o **NULL** está em sintonia com comportamentos de lógica de programação.
- Use as funções `is.na()`, `is.nan()`, `is.infinite()` e `is.null()` para testar se um objeto é um desses valores.

```
x <- c(NaN, Inf, 1, 2, 3, NA)
is.na(x)

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

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

3.5 Controles de Fluxo

Como toda boa linguagem de programação, o R possui estruturas de `if`'s, `else`'s, `for`'s, `while`'s etc. Esses **controles de fluxo** são importantes na hora de programar.

3.5.1 IF e ELSE

O seguinte trecho de código só será executado se o objeto `x` for igual a 1. Repare que a condição de igualdade é representada por dois iguais `==`.

```
x <- 2

if(x == 1) {
  Sys.time()      # Devolve a data/hora no momento da execução.
}
```

```
x <- 1
```

```
if(x == 1) {
  Sys.time()
}
```

```
## [1] "2020-02-20 10:25:42 -03"
```

O R só vai executar o que está na expressão dentro das chaves {} se o que estiver dentro dos parênteses () retornar TRUE.

A sintaxe com o else e o if else é

```
if(x < 0) {
  sinal <- "negativo"
} else if(x == 0) {
  sinal <- "neutro"
} else if(x > 0) {
  sinal <- "positivo"
}

sinal
```

```
## [1] "positivo"
```

Diferença entre SQL e R nas comparações lógicas

Igualdade: no SQL é só um sinal de igual: $<2 = 1$. No R são dois: $2 == 1$. Diferença: teste de diferente no R é $!=$ em vez de $<>$. Negação: em vez de usar a palavra “not” igual ao SQL, usamos $!$. Por exemplo, “entidade_id not in (‘100515’)” fica “!entidade_id %in% c(‘100515’)”.

3.5.2 for

Vamos usar o for para somar todos os elementos de um vetor.

```
x <- 1:10 # Cria um vetor com a sequência 1, 2, ..., 10.
soma <- 0

for(i in 1:10) {
  soma <- soma + x[i]
}
```

```
soma
```

```
## [1] 55
```

De forma equivalente, podemos usar diretamente a função `sum()`.

```
sum(x)
```

```
## [1] 55
```

Agora, vamos imprimir na tela o resultado da divisão de cada elemento de um vetor por dois. Para isso, utilizaremos a função `print()`.

```
vetor <- 30:35
indices <- seq_along(vetor) # cria o vetor de índices segundo o tamanho
                             # do objeto vetor.
for(i in indices) {
  print(vetor[1:i] / 2)
}
```

```
## [1] 15
## [1] 15.0 15.5
## [1] 15.0 15.5 16.0
## [1] 15.0 15.5 16.0 16.5
## [1] 15.0 15.5 16.0 16.5 17.0
## [1] 15.0 15.5 16.0 16.5 17.0 17.5
```

No trecho de código acima, preste atenção no resultado individual de cada uma das operações para entender como o R funciona.

3.5.3 while

O código a seguir irá imprimir na tela o valor de `i` enquanto este objeto for menor que 3. No momento em que a condição dentro das chaves `{}` não for mais respeitada, o processo será interrompido.

```
i <- 1
while(i < 3){
  print(i)
  i = i + 1
}
```

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

É importante que o valor de `i` seja atualizado em cada interação, caso contrário a função entrará em um loop infinito.

Vamos usar o `while` para encontrar uma aproximação da solução de $\sqrt{x} = x$. Este método é conhecido como Interação do ponto fixo e pode ser usado no cálculo aproximado de soluções de equações de uma variável real.

```
x <- 4
erro <- 100
while (abs(erro) > 0.1) {
  erro <- (sqrt(x)) - x
  x <- sqrt(x)
}
print(x)
```

```
## [1] 1.090508
```

Quando a diferença entre \sqrt{x} e x ficou menor que 0.1, o loop foi interrompido. Vemos que solução aproximada da equação $\sqrt{x} = x$ dada pelo algoritmo não difere muito da solução real $x = 1$.

Para finalizar, listamos na tabela abaixo os principais operadores lógicos.

Operador	Descrição
<code>x < y</code>	x menor que y?
<code>x <= y</code>	x menor ou igual a y?
<code>x > y</code>	x maior que y?
<code>x >= y</code>	x maior ou igual a y?
<code>x == y</code>	x igual a y?
<code>x != y</code>	x diferente de y?
<code>!x</code>	Negativa de x
<code>x y</code>	x ou y são verdadeiros?
<code>x & y</code>	x e y são verdadeiros?
<code>xor(x, y)</code>	x ou y são verdadeiros (apenas um deles)?

3.6 Fórmulas

Fórmulas são objetos do tipo `y ~ x`. Em geral, elas representam associações entre objetos, como em um modelo de regressão. As funções as usam de diversas maneiras, mas o exemplo mais emblemático vem da modelagem estatística.

```
formula <- y ~ x1 + x2
class(formula)
```

```
## [1] "formula"
```

A função `lm()` é a que ajusta um modelo linear no R, e `lm(y ~ x)` lê-se “regressão linear de y explicada por x”.

```
minha_formula <- Sepal.Width ~ Petal.Length + Petal.Width
class(minha_formula)
```

```
## [1] "formula"
lm(minha_formula, data = iris)

##
## Call:
## lm(formula = minha_formula, data = iris)
##
## Coefficients:
## (Intercept)  Petal.Length  Petal.Width
##          3.5870         -0.2571          0.3640
```

No caso específico dos modelos lineares, são nas fórmulas que conseguimos descrever as variáveis explicativas e suas interações. A fórmula $y \sim x1 * x2$ significa “y regredido por x1, x2 e a interação entre x1 e x2”. Fórmulas aparecem frequentemente em tarefas de modelagem.

Demais usos de fórmulas aparecerão em outras funções, como as do pacote `ggplot2`, com outros significados, e a documentação nos dirá como usá-las.

3.7 Gráficos (base)

O R já vem com funções básicas que fazem gráficos estatísticos de todas as naturezas.

- Vantagens: são rápidas e simples.
- Desvantagens: são feias e difíceis para gerar gráficos complexos.

Nesta seção, mostraremos como construir alguns tipos de gráficos usando as funções base do R, mas o nosso foco em visualização de dados está nas funções do pacote `ggplot2`.

3.7.1 Gráfico de dispersão

Para construir um gráfico de dispersão, utilizamos a função `plot()`. Seus principais parâmetros são:

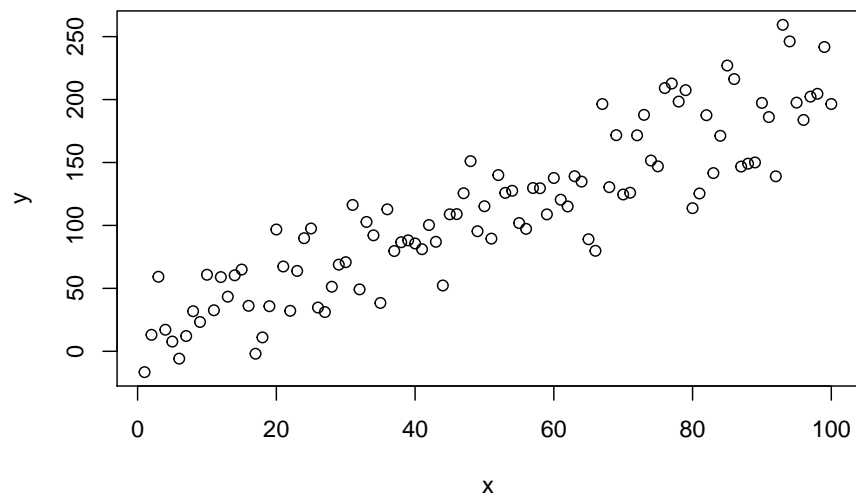
- `x`, `y` - Vetores para representarem os eixos x e y.
- `type` - Tipo de gráfico. Pode ser pontos, linhas, escada, entre outros.

Para mais detalhes sobre os argumentos, ver `help(plot)`.

Outras formas de utilizar a função `plot()`

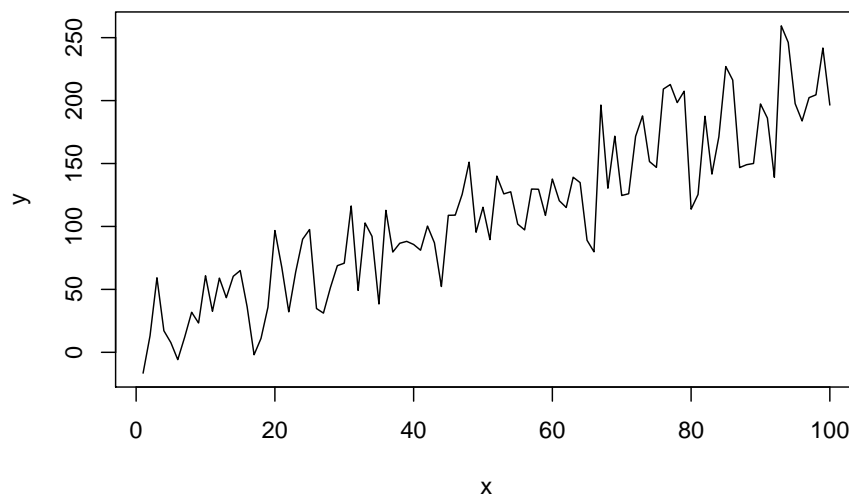
Além de gerar gráficos de dispersão, tentar chamar a função `plot(objeto_diferentao)` para qualquer tipo de objeto do R geralmente gera um gráfico interessante! Sempre tente fazer isso, a menos que seu objeto seja um `data.frame` com milhares de colunas!

```
N <- 100  
x <- 1:N  
y <- 5 + 2 * x + rnorm(N, sd = 30)  
plot(x, y)
```



O parâmetro `type = "l"` indica que queremos que os pontos sejam interligados por linhas.

```
plot(x, y, type = "l")
```

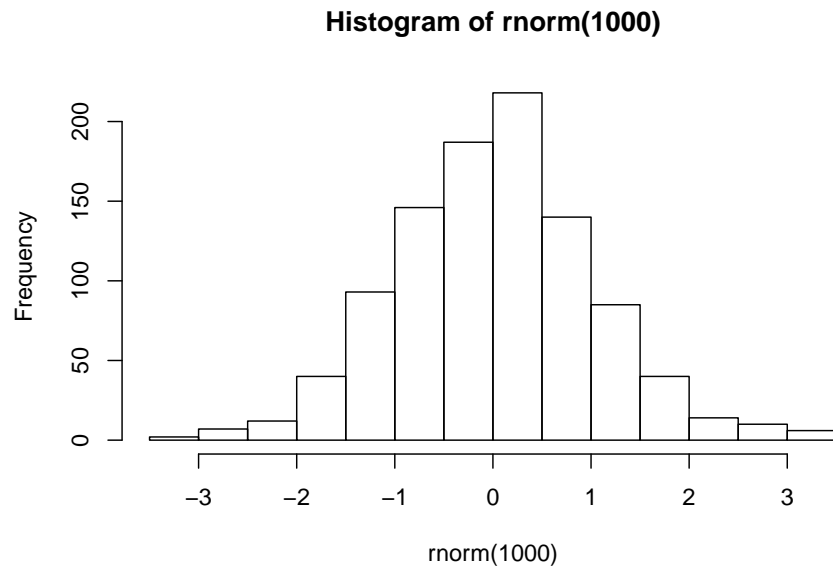



3.7.2 Histograma

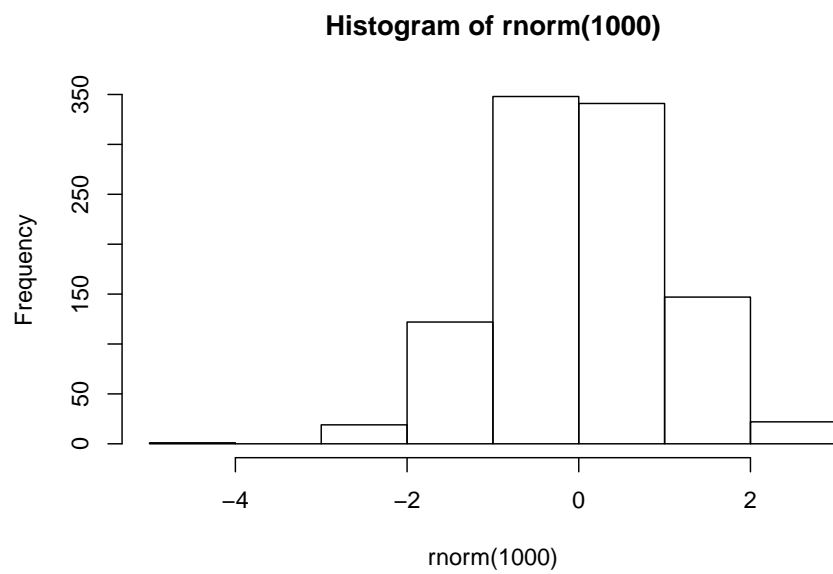
Para construir histogramas, utilizamos a função `hist()`. Os principais parâmetros são:

- `x` - O vetor numérico para o qual o histograma será construído.
- `breaks` - O número (aproximado) de retângulos.

```
hist(rnorm(1000))
```



```
hist(rnorm(1000), breaks = 6)
```

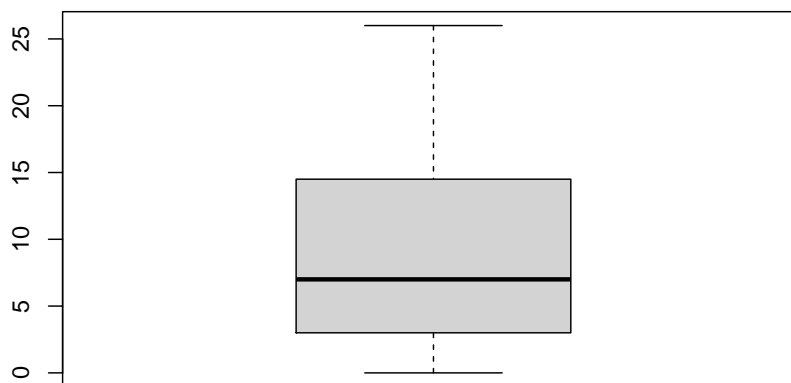


3.7.3 Boxplot

Para construir histogramas, utilizamos a função `boxplot()`. Os principais parâmetros são:

- `x` - O vetor numérico para o qual o boxplot será construído.

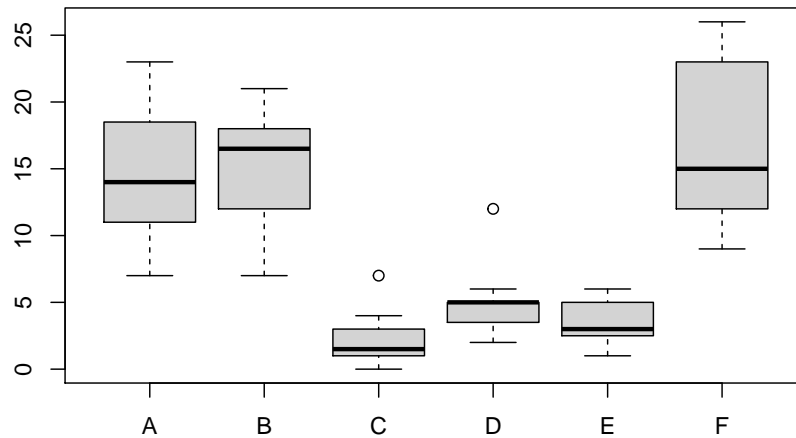
```
boxplot(InsectSprays$count, col = "lightgray")
```



Observe que o argumento `col=` muda a cor da caixa do boxplot.

Para mapear duas variáveis ao gráfico, utilizamos um objeto da classe `formula` e o argumento `data=`.

```
boxplot(count ~ spray, data = InsectSprays, col = "lightgray")
```



3.7.4 Gráfico de barras

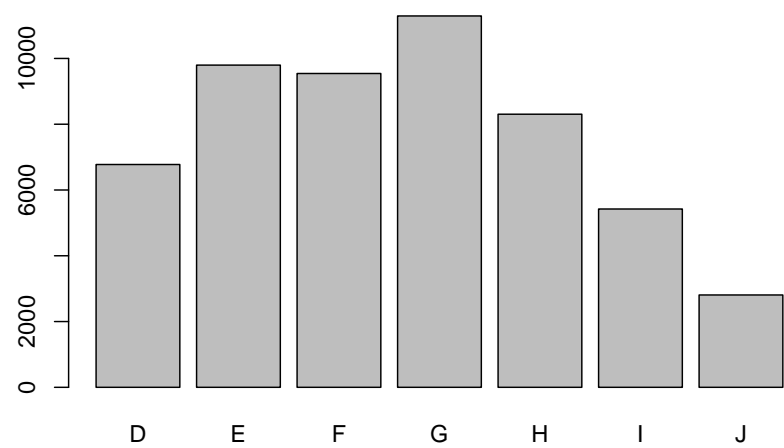
Para construir gráficos de barras, precisamos combinar as funções `table()` e `barplot()`.

No gráfico abaixo, primeiro criamos uma tabela de frequências com a função `table()` e, em seguida, construímos o gráfico com a função `barplot()`. A função `data()` carrega bases de dados de pacotes instalados. Veja `help(data)` para mais detalhes.

```
data(diamonds, package = "ggplot2")
tabela <- table(diamonds$color)
tabela
```

```
##
##      D      E      F      G      H      I      J
## 6775 9797 9542 11292 8304 5422 2808
```

```
barplot(tabela)
```

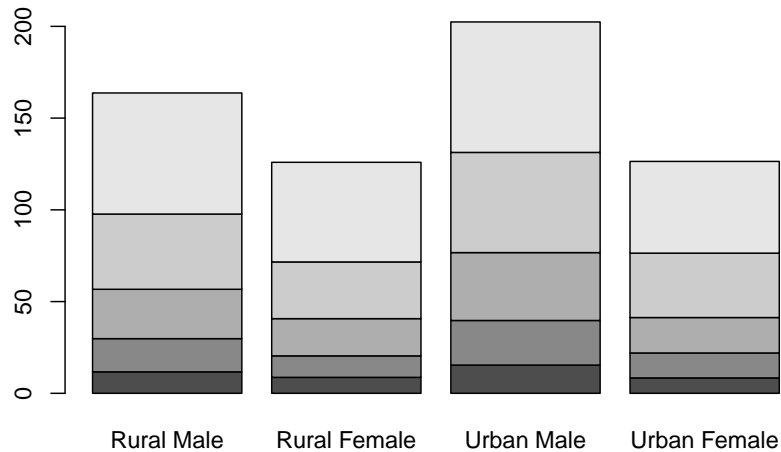


Também podemos mapear duas variáveis a um gráfico de barras utilizando tabelas de dupla entrada.

```
VADeaths
```

##	Rural	Male	Rural	Female	Urban	Male	Urban	Female
## 50-54		11.7		8.7		15.4		8.4
## 55-59		18.1		11.7		24.3		13.6
## 60-64		26.9		20.3		37.0		19.3
## 65-69		41.0		30.9		54.6		35.1
## 70-74		66.0		54.3		71.1		50.0

```
barplot(VADeaths)
```



3.8 Exercícios

Sugestão: resolva os exercícios em arquivo R Markdown, aproveitando para fazer anotações e registrar suas dúvidas ao longo do caminho.

1. Calcule o número de ouro no R.

Dica: o número de ouro é dado pela expressão $\frac{1+\sqrt{5}}{2}$.

2. Qual o resultado da divisão de 1 por 0 no R? E de -1 por 0?

3. Quais as diferenças entre `NaN`, `NULL`, `NA` e `Inf`? Digite expressões que retornam cada um desses resultados.

4. Sem rodar o código, calcule o que a expressão `5 + 3 * 10 %/% 3 == 15` vai resultar no R. Em seguida, apenas utilizando parênteses, faça a expressão retornar o valor contrário (i.e., se originariamente for `TRUE`, faça retornar `FALSE`).

5. Por que o código abaixo retorna erro? Arrume o código para retornar o valor `TRUE`.

```
x <- 4
if(x = 4) {
  TRUE
}
```

6. Usando `if` e `else`, escreva um código que retorne a string “número” caso o valor seja da classe `numeric` ou `integer`; a string “palavra” caso o valor seja da classe `character`; e `NULL` caso contrário.

7. Use o `for` para retornar o valor mínimo do seguinte vetor: `vetor <- c(4, 2, 1, 5, 3)`. Modifique o seu código para receber vetores de qualquer tamanho.

8. Usando apenas `for` e a função `length()`, construa uma função que calcule a média de um vetor número qualquer. Construa uma condição para a função retornar `NULL` caso o vetor não seja numérico.

9. Rode `help(runif)` para descobrir o que a função `runif()` faz. Em seguida, use-a para escrever uma função que retorne um número aleatório inteiro entre 0 e 10 (0 e 10 inclusive).

10. Rode `help(sample)` para descobrir o que a função `sample()` faz. Em seguida, use-a para escrever uma função que escolha uma linha aleatoriamente de uma matriz e devolva os seus valores.

11. Rode `help(paste)` e `help(names)` para descobrir o que as funções `paste()` e `names()` fazem. Em seguida, use-as para escrever um código para gerar a fórmula `mpg ~ cyl + disp + hp + drat + wt + qsec + vs + am + gear + carb` a partir do data frame `mtcars`.

3.9 Respostas

1. Calculo o número de ouro no R.

Dica: o número de ouro é dado pela expressão $\frac{1+\sqrt{5}}{2}$.

Resposta:

```
(1 + sqrt(5))/2
```

2. Qual o resultado da divisão de 1 por 0 no R? E de -1 por 0?

Resposta:

Infinito e -Infinito.

```
1/0
```

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

```
-1/0
```

```
## [1] -Inf
```

3. Quais as diferenças entre NaN, NULL, NA e Inf? Digite expressões que retornam cada um desses resultados.

Resposta:

```
# NaN é o resultado de uma operação matemática inválida.  
# Significa Not A Number.
```

```
0/0
```

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

```
# NULL é o vazio do R. É como se o objeto não existisse.
```

```
NULL  
a = NULL
```

```
# veja que um vetor, mesmo sem elementos não é NULL
```

```
is.null(integer(length = 0))
```

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

```
# NA é uma constante lógica do R. Significa Not Available.  
# NA pode ser convertido para quase todos os tipos de vetores do R.  
# É usado principalmente para indicar valores faltantes.
```

```
NA
```

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

```
as.numeric(c("1", "2", "a"))
```

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



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

```
# Inf é significa infinito. É o resultado de operações matemáticas  
# cujo limite é infinito.
```

```
1/0
```

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

```
1/Inf
```

```
## [1] 0
```

4. Sem rodar o código, calcule o que a expressão `5 + 3 * 10 %% 3 == 15` vai resultar no R. Em seguida, apenas utilizando parênteses, faça a expressão retornar o valor contrário (i.e., se originariamente for `TRUE`, faça retornar `FALSE`).

Resposta:

O resultado da parte esquerda é 14, por isso a expressão retornará `FALSE`. Para fazê-la retornar `TRUE`, basta colocar parênteses em volta de `3 * 10`.

```
5 + (3 * 10) %% 3 == 15
```

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

5. Por que o código abaixo retorna erro? Arrume o código para retornar o valor `TRUE`.

```
x <- 4  
if(x = 4) {  
  TRUE  
}
```

Resposta:

A expressão `x = 4` está tentando atribuir o valor 4 ao objeto `x` dentro do `if`, o que não é permitido pois o controlador `if` só aceita valores lógicos. Para corrigir o código e fazê-lo retornar `TRUE`, basta trocar `=` por `==`.

```
x <- 4  
if(x == 4) {  
  TRUE  
}
```

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

6. Usando `if` e `else`, escreva um código que retorne a string “número” caso o objeto `x` seja da classe `numeric` ou `integer`; a string “palavra” caso o objeto seja da classe `character`; e `NULL` caso contrário.

Resposta:

```
x <- 1
# x <- 1L
# x <- "1"

if(is.numeric(x)) {
  "número"
} else if(is.character(x)) {
  "palavra"
} else {
  NULL
}
```

```
## [1] "número"
```

Note que a função `is.numeric()` retorna `TRUE` para as classes `integer` e `numeric`.

7. Use o `for` para retornar o valor mínimo do seguinte vetor: `vetor <- c(4, 2, 1, 5, 3)`. Modifique o seu código para receber vetores de qualquer tamanho.

Resposta:

```
vetor <- c(4, 2, 1, 5, 3)
minimo <- Inf

for(i in 1:5) {

  if(minimo > vetor[i]) {
    minimo <- vetor[i]
  }

}

minimo
```

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

Lembrete: o R já possui a função `min()` para calcular o mínimo de um conjunto de valores.

8. Usando apenas `for` e a função `length()`, construa uma função que calcule

a média de um vetor número qualquer. Construa uma condição para a função retornar NULL caso o vetor não seja numérico.

Resposta:

```
media <- function(x) {  
  
  i <- 1  
  tamanho <- length(x)  
  soma <- 0  
  
  for(i in 1:tamanho){  
    soma <- soma + x[i]  
  }  
  
  return(soma/tamanho)  
}  
  
media(1:3)  
  
## [1] 2
```

9. Rode `help(runif)` para descobrir o que a função `runif()` faz. Em seguida, use-a para escrever uma função que retorne um número aleatório inteiro entre 0 e 10 (0 e 10 inclusive).

Resposta:

A função `runif()` gera números reais aleatórios entre um valor mínimo e um valor máximo.

```
alea <- function() {  
  
  x <- runif(n = 1, min = 0, max = 10)  
  x <- round(x)  
  
  return(x)  
}  
  
alea()  
  
## [1] 4
```

Veja que construímos uma função sem argumentos. Podemos generalizá-la incluindo os argumentos da função `runif()`.

```
alea <- function(n, min, max) {  
  
  x <- runif(n = n, min = min, max = max)
```

```

x <- round(x)

return(x)
}

alea(2, 2, 5)

## [1] 4 3
alea(5, 100, 105)

## [1] 104 101 102 103 104

```

3.10 Observe que não há problema em usar os mesmos nomes para os argumentos. Isso se deve aos *environments*.

10. Rode `help(sample)` para descobrir o que a função `sample()` faz. Em seguida, use-a para escrever uma função que escolha uma linha aleatoriamente de uma matriz e devolva os seus valores.

Resposta:

```

matriz <- matrix(runif(20), nrow = 5, ncol = 4)

linha_alea <- function(matriz) {

  x <- 1:nrow(matriz)

  linha <- sample(x, size = 1)

  return(matriz[linha,])
}

matriz

##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.3103665 0.4302944 0.32463224 0.978550865
## [2,] 0.5691939 0.2314452 0.49573388 0.424211303
## [3,] 0.3664093 0.5934599 0.40798828 0.009468612
## [4,] 0.7712976 0.7420212 0.05534789 0.450321388
## [5,] 0.8085146 0.7000902 0.53750627 0.329305838

linha_alea(matriz)

## [1] 0.77129764 0.74202116 0.05534789 0.45032139

```

3.10. OBSERVE QUE NÃO HÁ PROBLEMA EM USAR OS MESMOS NOMES PARA OS ARGUMENTOS. ISSO S

11. Rode `help(paste)` e `help(names)` para descobrir o que as funções `paste()` e `names()` fazem. Em seguida, use-as para escrever um código para gerar a fórmula `mpg ~ cyl + disp + hp + drat + wt + qsec + vs + am + gear + carb` a partir do data frame `mtcars`.

Resposta:

```
variaveis <- names(mtcars)

esq <- "mpg ~ "
dir <- paste(variaveis[-1], collapse = " + ")

formula <- paste0(esq, dir)
as.formula(formula)

## mpg ~ cyl + disp + hp + drat + wt + qsec + vs + am + gear + carb
```

Observe que a função `paste0()` é equivalente à função `paste()` com o argumento `sep = ""`.

Chapter 4

Pipe

4.1 O operador pipe

O operador `%>%` (*pipe*) foi uma das grandes revoluções recentes do R, tornando a leitura de códigos mais lógica, fácil e compreensível. Ele foi introduzido por Stefan Milton Bache no pacote `magrittr` e já existem diversos pacotes construídos para facilitar a sua utilização.

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

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

A ideia do operador `%>%` (*pipe*) é bem simples: usar o valor resultante da expressão do lado esquerdo como primeiro argumento da função do lado direito.

As duas linhas abaixo são equivalentes.

```
f(x, y)  
x %>% f(y)
```

Nos casos mais simples, o *pipe* parece não trazer grandes vantagens. Agora, veja como fica um caso com mais etapas.

Vamos calcular a raiz quadrada da soma dos valores de 1 a 4. Primeiro, sem o pipe.

```
x <- c(1, 2, 3, 4)  
sqrt(sum(x))
```

```
## [1] 3.162278
```

```
# Agora com o pipe.

x %>% sum() %>% sqrt()
```

```
## [1] 3.162278
```

O caminho que o código `x %>% sum() %>% sqrt()` seguiu foi enviar o objeto `x` como argumento da função `sum()` e, em seguida, enviar a saída da expressão `sum(x)` como argumento da função `sqrt()`. Observe que escrevemos o código na mesma ordem das operações. A utilização de parênteses após o nome das funções não é necessário, mas recomendável.

Se você ainda não está convencido com o poder do *pipe*, fica que vai ter bolo!

No exemplo abaixo, vamos ilustrar um caso em que temos um grande número de funções aninhadas. Veja como a utilização do *pipe* transforma um código confuso e difícil de ser lido em algo simples e intuitivo.

```
# Receita de bolo sem pipe. Tente entender o que é preciso fazer.
```

```
esfrie(asse(coloque(bata(acrescente(recipiente(rep("farinha", 2), "água", "fermento",
```

```
# Veja como o código acima pode ser reescrito utilizando-se o pipe. Agora realmente se
```

```
recipiente(rep("farinha", 2), "água", "fermento", "leite", "óleo") %>%
  acrescente("farinha", até = "macio") %>%
  bata(duração = "3min") %>%
  coloque(lugar = "forma", tipo = "grande", untada = TRUE) %>%
  asse(duração = "50min") %>%
  esfrie("geladeira", "20min")
```

Às vezes, queremos que o resultado do lado esquerdo vá para outro argumento do lado direito que não o primeiro. Para isso, utilizamos um `.` como marcador.

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

```
airquality %>%
  na.omit %>%
  lm(Ozone ~ Wind + Temp + Solar.R, data = .) %>%
  summary
```

```
##
## Call:
## lm(formula = Ozone ~ Wind + Temp + Solar.R, data = .)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```



```
## -40.485 -14.219 -3.551 10.097 95.619
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -64.34208   23.05472  -2.791  0.00623 **
## Wind        -3.33359    0.65441  -5.094 1.52e-06 ***
## Temp         1.65209    0.25353   6.516 2.42e-09 ***
## Solar.R      0.05982    0.02319   2.580 0.01124 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.18 on 107 degrees of freedom
## Multiple R-squared:  0.6059, Adjusted R-squared:  0.5948
## F-statistic: 54.83 on 3 and 107 DF,  p-value: < 2.2e-16
```

Também é possível definir funções na sua *pipeline*.

```
c(1,2,3) %>%
  (function(x){
    sum(x)
  })
```

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

O *pipe* é a força da gravidade dentro do *tidyverse*. Veremos nas próximas seções como as funções de diferentes pacotes interagem perfeitamente graças a esse operador.

4.2 Outros operadores

Existem outros operadores do mesmo pacote, que apesar de menos usados, também são úteis.

São eles:

- Assignment operator %<>%
- Operador tee %T>%
- Exposition operator %\$%

4.2.1 Operador de atribuição (Assignment operator)

Quando queremos sobrescrever um objeto, é comum utilizarmos o operador <-. Por exemplo, se queremos somar 10 a cada valor do vetor *x*, podemos fazer:

```
x <- c(1,2,3,4)
x <- x %>% add(10)
x
```

```
## [1] 11 12 13 14
```

Com o operador de atribuição, o código acima se reduz a

```
x %<>% add(10)
```

Este operador pode ser usado sempre que desejamos fazer algo da forma `objeto <- objeto %>% função`

4.2.2 Operador tee

O operador tee retorna o valor do comando anterior a `%T>%`, não o resultado do lado direito dele como o pipe faz. O seguinte exemplo vai imprimir na tela os valores de 1 a 10. Se usássemos o pipe, o código retornaria a soma dos dez números.

```
1:10 %T>% sum() %>% cat()
```

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

Neste caso, o operador não parece fazer sentido e apenas deixa o código mais complicado, mas se desejamos usar funções como `cat()` ou `plot()` que não retornam nada, o operador se torna muito útil.

Vamos imprimir na tela os valores de 1 a 10 e depois soma-los.

```
1:10 %T>% cat() %>% sum()
```

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

```
## [1] 55
```

4.2.3 Exposition operator

Usamos o operador `%%` para salvar o valor resultante da expressão do lado esquerdo, podendo usar como quiser do lado direito.

Por exemplo, para obter o primeiro elemento de um vetor, podemos fazer:

```
data.frame(z=1:10)%% z[1]
```

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

Para mais informações sobre o `pipe`, outros operadores relacionados e exemplos de utilização, visite a página [Ceci n'est pas un pipe](#).

4.3 Exercícios

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

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

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

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

```
x <- rnorm(100)
x.pos <- x[x>0]
media <- mean(x.pos)
saida <- round(media, 1)
```

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

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

-
4. Leia o capítulo sobre *pipes* do R for data science. É curto e vale muito a pena.

-
5. Pegue algum script que você já tenha programado em R e o reescreva utilizando o operador *pipe*. Se você não tiver nenhum, não se preocupe. Utilizaremos **bastante** o *pipe* daqui pra frente.

4.4 Respostas

Não há apenas uma maneira de resolver os exercícios. Você pode encontrar soluções diferentes das nossas, algumas vezes mais eficientes, outras vezes menos.

Quando estiver fazendo suas análises, tente buscar o equilíbrio entre eficiência e praticidade. Economizar 1 hora com a execução do código pode não valer a pena se você demorou 2 horas a mais para programá-lo.

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

```
round(mean(sum(1:10)/3), digits = 1)
## [1] 18.3

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

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

```
# Setamos a semente que gera números aleatórios para deixar o resultado reprodutível

set.seed(137)

x <- rnorm(100)
x.pos <- x[x>0]
media <- mean(x.pos)
saida <- round(media, 2)
saida
## [1] 0.78

set.seed(137)

rnorm(100) %>%
  magrittr::extract(. > 0) %>%
  mean %>%
  round(digits = 2)
## [1] 0.78
```

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

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

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

Chapter 5

Importação

Nesta seção, vamos introduzir os principais pacotes para importar dados para o R. Mostraremos como importar dados de arquivos de texto, planilhas do excel e extensões de outros programas estatísticos (SAS e SPSS, por exemplo).

Antes de começarmos, vale a pena tocarmos num ponto importante. As funções de importação do tidyverse carregam os dados em **tibbles**, que diferem da classe **data.frames** usual em dois pontos importantes:

- imprime os dados na tela de maneira muito mais organizada, resumida e legível; e
- permite a utilização de *list-columns*.

Se você não estiver familiarizado com o conceito de *list-columns*, não se preocupe. Trataremos melhor do assunto no tópico sobre funcionais.

5.1 readr

O pacote **readr** do tidyverse é utilizado para importar arquivos de texto, como **.txt** ou **.csv**, para o R.

O **readr** transforma 7 tipos de arquivos de textos em **tibbles** usando as funções:

- **read_csv()**: arquivos separados por vírgula.
- **read_tsv()**: arquivos separados por tab.
- **read_delim()**: arquivos separados por um delimitador qualquer. O argumento **delim=** indica qual caracter separa cada coluna no arquivo de texto.
- **read_fwf()**: arquivos compactos que devem ter sua largura especificada. Existem várias funções para especificar a largura ou posição.

- `read_table()`: arquivos de texto tabular com suas colunas separadas por espaço.
- `read_log()`: arquivos log do Apache.

Como exemplo, utilizaremos uma base de filmes do IMDB, gravada em diversos formatos. Os arquivos podem ser encontrados neste link.

```
library(readr)

imdb_csv <- read_csv(file = "data/imdb.csv")
```

Repare que o argumento `file=` representa o caminho até o arquivo. Se o arquivo a ser lido não estiver no diretório de trabalho da sua sessão, você precisa especificar o caminho até o arquivo.

Exercício relâmpago! Descubra qual a diferença entre as funções `read_csv()` e `read_csv2()`?

Para a maioria das funções `read_`, existe uma respectiva função `write_`. Essas funções servem para salvar bases em um formato específico de arquivo. Além do nome do arquivo a ser criado, você também precisa passar o objeto que será gravado. Repare nos exemplos abaixo que você precisa especificar a extensão do arquivo corretamente.

```
write_csv(x = mtcars, path = "data/mtcars.csv")
write_delim(x = mtcars, delim = " ", path = "data/mtcars.txt")
```

Com exceção do arquivo `.fwf`, a forma de importar e escrever arquivos são muito semelhantes. Para a função `read_fwf`, além da necessidade de especificar o caminho até o arquivo, também é necessário especificar a posição da coluna, `col_position =`. Para isso usamos funções como `fwf_empty()` ou `fwf_widths()`. Este formato não tem uma função `write_` específica no pacote, mas podemos salvar bases em `fwf` com a função `write.fwf()` do pacote `gdata`.

```
library(gdata)
write.fwf(mtcars[,1:4], 'data/mtcars.fwf', colnames = F)

read_fwf('data/mtcars.fwf', col_positions = fwf_empty('data/mtcars.fwf', col_names = c(
```

Aqui, pegamos as 4 primeiras colunas de `mtcars` e criamos um arquivo `.fwf`, depois o transformamos em tibble. A função `fwf_empty` especifica a posição das colunas com base nas colunas que estão vazias. Para entender melhor o que cada função `fwf_` faz, entre em `help(read_fwf)`.

Observe que quando chamamos a função `read_`, uma mensagem é impressa na tela.

```
imdb_tsv <- read_tsv('data/imdb.tsv')
```

```
## Parsed with column specification:
```



```
## cols(
##   titulo = col_character(),
##   ano = col_double(),
##   diretor = col_character(),
##   duracao = col_double(),
##   cor = col_character(),
##   generos = col_character(),
##   pais = col_character(),
##   classificacao = col_character(),
##   orcamento = col_double(),
##   receita = col_double(),
##   nota_imdb = col_double(),
##   likes_facebook = col_double(),
##   ator_1 = col_character(),
##   ator_2 = col_character(),
##   ator_3 = col_character()
## )
```

Essa mensagem está definindo a classe de cada variável. Se as variáveis não foram especificadas de maneira correta, você pode copiar esta mensagem e reescrevê-la da forma desejada.

No exemplo, suponha que você gostaria que a variável `ano` fosse um inteiro. Podemos resolver isso da seguinte forma:

```
imdb_tsv <- read_tsv('data/imdb.tsv', col_types =
  cols(
    titulo = col_character(),
    #mudando de double para integer
    ano = col_integer(),
    diretor = col_character(),
    duracao = col_double(),
    cor = col_character(),
    generos = col_character(),
    pais = col_character(),
    classificacao = col_character(),
    orcamento = col_double(),
    receita = col_double(),
    nota_imdb = col_double(),
    likes_facebook = col_double(),
    ator_1 = col_character(),
    ator_2 = col_character(),
    ator_3 = col_character()
  ))
```

Para conhecer as especificações das colunas disponíveis clique [aqui](#).

Também é possível salvar objetos, como `data.frames` em um tipo especial de

arquivos, o `.rds`. A vantagem dessa extensão é guardar a estrutura dos dados salvos, como a classe das colunas de um `data.frame`. Além disso, é uma boa alternativa para lidar com grandes bancos de dados, já que arquivos `.rds` serão bem mais compactos do que arquivos Excel.

```
imdb_rds <- read_rds(path = "data/imdb.rds")
write_rds(mtcars, path = "data/mtcars.rds")
```

5.2 readxl

O pacote `readxl` do `tidyverse` contém funções para importação com os formatos `.xls` e `.xlsx`.

```
readxl::read_xls(path = "data/imdb.xls")
readxl::read_xlsx(path = "data/imdb.xlsx")
```

A função `read_excel()` auto detecta a extensão do arquivo.

```
read_excel(path = "data/imdb.xls")
read_excel(path = "data/imdb.xlsx")
```

O pacote disponibiliza 5 exemplos de arquivos com formato `.xls` e `.xlsx`.

```
readxl_example()
```

```
## [1] "clippy.xls"      "clippy.xlsx"     "datasets.xls"    "datasets.xlsx"
## [5] "deaths.xls"     "deaths.xlsx"     "geometry.xls"    "geometry.xlsx"
## [9] "type-me.xls"    "type-me.xlsx"
```

Vamos acessar o arquivo `datasets.xls`.

```
datasets <- readxl_example("datasets.xls")
read_xls(datasets)
```

```
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <dbl>         <dbl>         <dbl>         <dbl> <chr>
## 1         5.1         3.5         1.4         0.2 setosa
## 2         4.9         3         1.4         0.2 setosa
## 3         4.7         3.2         1.3         0.2 setosa
## 4         4.6         3.1         1.5         0.2 setosa
## 5         5         3.6         1.4         0.2 setosa
## 6         5.4         3.9         1.7         0.4 setosa
## 7         4.6         3.4         1.4         0.3 setosa
## 8         5         3.4         1.5         0.2 setosa
## 9         4.4         2.9         1.4         0.2 setosa
## 10        4.9         3.1         1.5         0.1 setosa
## # ... with 140 more rows
```

No Excel, um arquivo pode ter várias planilhas. Você pode ver quais planilhas fazem parte do arquivo:

```
excel_sheets(datasets)
```

```
## [1] "iris"      "mtcars"    "chickwts" "quakes"
```

Observe que quando usamos a função `read_xls(datasets)`, o R transformou em tibble apenas a primeira planilha do arquivo. Caso essa não for a tabela que você deseja acessar, não se preocupe! Podemos resolver seu problema de forma simples.

```
read_xls(datasets,sheet = 'chickwts')
```

```
## # A tibble: 71 x 2
##   weight feed
##   <dbl> <chr>
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
## 7    108 horsebean
## 8    124 horsebean
## 9    143 horsebean
## 10   140 horsebean
## # ... with 61 more rows
```

Também é possível fazer uma seleção das células da planilha que você deseja importar usando o argumento `range` = da função `read_excel`. Podemos indicar quais colunas ou linhas desejamos com as funções `cell_cols()` e `cell_rows()` respectivamente. Ou podemos definir a dimensão dos dados a partir de uma célula com a função `anchored()`. Veja todas as funções disponíveis neste manual.

```
read_xls(datasets,sheet = 'chickwts',range = anchored('A3',dim = c(5,2)),col_names = F)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2

## # A tibble: 5 x 2
##   ...1 ...2
##   <dbl> <chr>
## 1    160 horsebean
## 2    136 horsebean
## 3    227 horsebean
## 4    217 horsebean
## 5    168 horsebean
```

5.3 haven

Para ler arquivos gerados por outros softwares, como SPSS, SAS e STATA, você pode usar as funções do pacote **haven**. Este pacote faz parte do **tidyverse** e é um wrapper da biblioteca ReadStat, escrita em C.

```
library(haven)

imdb_sas <- read_sas("data/imdb.sas7bdat")
imdb_spss <- read_spss("data/imdb.sav")
imdb_dta <- read_dta("data/imdb.dta")
```

É possível salvar ou escrever bases em SAS e STATA com as funções `write_sas` e `write_dta`.

```
write_dta(mtcars, 'data/matcars.dta')
```

Quando importamos arquivos gerados pelo SAS SPSS ou STATA para o R, os rótulos de uma variável podem não ser importados de forma correta. O pacote **haven** tem uma solução para este problema.

```
x <- labelled(c(1,1,2,3,2,2,1,2), c(Ruim = 1, Bom = 2, Otimo = 3))
```

`labelled()` adiciona rótulos à valores de uma variável. Para verificar quais são estes rótulos, podemos usar a função `print_labels()`.

```
print_labels(x)
```

```
##
## Labels:
## value label
##      1 Ruim
##      2 Bom
##      3 Otimo
```

Existe uma função similar a `labelled()`, exclusiva para o SPSS, que além de rotular as variáveis, também defini quais símbolos representam valores faltantes, dado que em SPSS pode haver mais de um tipo de *missing*.

```
x1 <- labelled_spss(c(1,3,0,2,2,1,0,2,4), c(Ruim = 1, Bom = 2, Otimo = 3), na_values = c(0,4))
is.na(x1)
```

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

Agora que já definimos os *missings* “especiais”, podemos transformá-los no *missing* padrão do R, representado pelo símbolo *NA*.

```
x1 <- zap_missing(x1)
x1
```

```
## <Labelled double>
## [1] 1 3 NA 2 2 1 NA 2 NA
##
## Labels:
## value label
##      1 Ruim
##      2 Bom
##      3 Otimo
```

Existem outras funções `zap_` interessantes no pacote.

Após rotular os valores do vetor, podemos convertê-los, por exemplo, em fator. Para isso, usamos uma função do pacote `haven`. A função base `as.factor()` também poderia ser usada, mas quando a usamos, os rótulos não são considerados.

```
x_base <- base::as.factor(x)
levels(x_base)

## [1] "1" "2" "3"

x_factor <- haven::as_factor(x)
levels(x_factor)

## [1] "Ruim" "Bom" "Otimo"
```


Chapter 6

Manipulação

“(...) The fact that data science exists as a field is a colossal failure of statistics. To me, what I do is what statistics is all about. It is gaining insight from data using modelling and visualization. Data munging and manipulation is hard and statistics has just said that’s not our domain.” - Hadley Wickham

Esta seção trata do tema *manipulação de dados*. Trata-se de uma tarefa dolorosa e demorada, tomando muitas vezes a maior parte do tempo de uma análise estatística. Essa etapa é essencial em qualquer análise de dados e, apesar de negligenciada pela academia, é decisiva para o sucesso de estudos aplicados.

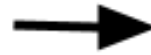
Usualmente, o cientista de dados parte de uma base “crua” e a transforma até obter uma base de dados analítica, que, a menos de transformações simples, está preparada para passar por análises estatísticas.

A figura abaixo mostra a fase de “disputa” com os dados (*data wrangling*) para deixá-los no formato analítico.

Importar



Arrumar
(Armazenar os dados
consistentemente)



Transformar
(Criar novas variáveis
agregações)

Comunicar



Um conceito importante para obtenção de uma base analítica é o *data tidying*, ou arrumação de dados. Uma base é considerada *tidy* se

1. Cada linha da base representa uma observação.
2. Cada coluna da base representa uma variável.

A base de dados analítica é estruturada de tal forma que pode ser colocada diretamente em ambientes de modelagem estatística ou de visualização. Nem sempre uma base de dados analítica está no formato *tidy*, mas usualmente são necessários poucos passos para migrar de uma para outra. A filosofia *tidy* é a base do tidyverse.

Os principais pacotes encarregados da tarefa de estruturar os dados são o `dplyr` e o `tidyr`. Eles serão o tema desse tópico. Instale e carregue os pacotes utilizando:

```
install.packages("dplyr")
install.packages("tidyr")

library(dplyr)
library(tidyr)
```

Mas antes de apresentar as principais funções do `dplyr` e do `tidyr`, precisamos falar sobre o conceito de `tibbles`.

6.1 Trabalhando com tibbles

Uma `tibble` nada mais é do que um `data.frame`, mas com um método de impressão mais adequado.

As `tibbles` são parte do pacote `tibble`. Assim, para começar a usá-las, instale e carregue o pacote.

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

Mais informações sobre `tibbles` podem ser encontradas neste link.

Nessa seção, vamos trabalhar com uma base de filmes do IMDB. Essa base pode ser baixada clicando aqui.

```
imdb <- readr::read_rds("data/imdb.rds")
```

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

```
imdb

## # A tibble: 3,807 x 15
##   titulo    ano diretor duracao cor   generos pais classificacao orcamento
##   <chr>    <int> <chr>      <int> <chr> <chr>   <chr> <chr>          <int>
## 1 Avata~  2009 James ~      178 Color Action~ USA    A partir de ~ 237000000
```

```
## 2 Pirat~ 2007 Gore V~ 169 Color Action~ USA A partir de ~ 300000000
## 3 The D~ 2012 Christ~ 164 Color Action~ USA A partir de ~ 250000000
## 4 John ~ 2012 Andrew~ 132 Color Action~ USA A partir de ~ 263700000
## 5 Spide~ 2007 Sam Ra~ 156 Color Action~ USA A partir de ~ 258000000
## 6 Tangl~ 2010 Nathan~ 100 Color Advent~ USA Livre 260000000
## 7 Aveng~ 2015 Joss W~ 141 Color Action~ USA A partir de ~ 250000000
## 8 Batma~ 2016 Zack S~ 183 Color Action~ USA A partir de ~ 250000000
## 9 Super~ 2006 Bryan ~ 169 Color Action~ USA A partir de ~ 209000000
## 10 Pirat~ 2006 Gore V~ 151 Color Action~ USA A partir de ~ 225000000
## # ... with 3,797 more rows, and 6 more variables: receita <int>,
## # nota_imdb <dbl>, likes_facebook <int>, ator_1 <chr>, ator_2 <chr>,
## # ator_3 <chr>
```

Veja que, por padrão, apenas as dez primeiras linhas da `tibble` são impressas na tela. Além disso, as colunas que não couberem na largura do console serão omitidas. Também são apresentadas a dimensão da tabela e as classes de cada coluna.

6.2 dplyr

O `dplyr` é o pacote mais útil para realizar transformação de dados, aliando simplicidade e eficiência de uma forma elegante. Os scripts em R que fazem uso inteligente dos verbos `dplyr` e as facilidades do operador *pipe* tendem a ficar mais legíveis e organizados sem perder velocidade de execução.

As principais funções do `dplyr` são:

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

Todas essas funções seguem as mesmas características:

- O *input* é sempre uma `tibble` e o *output* é sempre um `tibble`.
- Colocamos o `tibble` no primeiro argumento e o que queremos fazer nos outros argumentos.
- A utilização é facilitada com o emprego do operador `%>%`.
- O pacote faz uso extensivo de NSE (*non standard evaluation*).

As principais vantagens de se usar o `dplyr` em detrimento das funções do R base são:

- Manipular dados se torna uma tarefa muito mais simples.
- O código fica mais intuitivo de ser escrito e mais simples de ser lido.

- O pacote `dplyr` utiliza C e C++ por trás da maioria das funções, o que geralmente torna o código mais eficiente.
- É possível trabalhar com diferentes fontes de dados, como bases relacionais (SQL) e `data.table`.

Agora, vamos avaliar com mais detalhes as principais funções do pacote `dplyr`.

6.2.1 `filter()`

A função `filter()` filtra linhas. Ela é semelhante à função `subset()`, do R base. O código abaixo retorna apenas filmes com nota maior que nove.

```
imdb %>%
  filter(nota_imdb > 9)

## # A tibble: 3 x 15
##   titulo    ano diretor duracao cor   generos pais classificacao orcamento
##   <chr>    <int> <chr>      <int> <chr> <chr>   <chr>   <chr>          <int>
## 1 The S~   1994 Frank ~    142 Color Crime|~ USA   A partir de ~ 25000000
## 2 The G~   1972 Franci~    175 Color Crime|~ USA   A partir de ~ 6000000
## 3 Kickb~   2016 John S~     90 <NA> Action  USA   Outros          17000000
## # ... with 6 more variables: receita <int>, nota_imdb <dbl>,
## #   likes_facebook <int>, ator_1 <chr>, ator_2 <chr>, ator_3 <chr>
```

Para fazer várias condições, use os operadores lógicos `&` e `|` ou separe filtros entre vírgulas.

```
imdb %>%
  filter(ano > 2010 & nota_imdb > 8.5)

## # A tibble: 5 x 15
##   titulo    ano diretor duracao cor   generos pais classificacao orcamento
##   <chr>    <int> <chr>      <int> <chr> <chr>   <chr>   <chr>          <int>
## 1 Inter~   2014 Christ~    169 Color Advent~ USA   A partir de ~ 165000000
## 2 Runni~   2015 Mike M~     88 Color Family  USA   Outros          5000000
## 3 A Beg~   2016 Mitche~     87 Color Comedy~ USA   Outros           NA
## 4 Kickb~   2016 John S~     90 <NA> Action  USA   Outros          17000000
## 5 Butte~   2014 Cary B~     78 Color Docume~ USA   Outros          180000
## # ... with 6 more variables: receita <int>, nota_imdb <dbl>,
## #   likes_facebook <int>, ator_1 <chr>, ator_2 <chr>, ator_3 <chr>
```

```
imdb %>%
  filter(receita > orcamento | nota_imdb > 9)

## # A tibble: 1,762 x 15
##   titulo    ano diretor duracao cor   generos pais classificacao orcamento
##   <chr>    <int> <chr>      <int> <chr> <chr>   <chr>   <chr>          <int>
```

```
## 1 Avata~ 2009 James ~ 178 Color Action~ USA A partir de ~ 237000000
## 2 Pirat~ 2007 Gore V~ 169 Color Action~ USA A partir de ~ 300000000
## 3 The D~ 2012 Christ~ 164 Color Action~ USA A partir de ~ 250000000
## 4 Spide~ 2007 Sam Ra~ 156 Color Action~ USA A partir de ~ 258000000
## 5 Aveng~ 2015 Joss W~ 141 Color Action~ USA A partir de ~ 250000000
## 6 Batma~ 2016 Zack S~ 183 Color Action~ USA A partir de ~ 250000000
## 7 Pirat~ 2006 Gore V~ 151 Color Action~ USA A partir de ~ 225000000
## 8 Man o~ 2013 Zack S~ 143 Color Action~ USA A partir de ~ 225000000
## 9 The A~ 2012 Joss W~ 173 Color Action~ USA A partir de ~ 220000000
## 10 The A~ 2012 Marc W~ 153 Color Action~ USA A partir de ~ 230000000
## # ... with 1,752 more rows, and 6 more variables: receita <int>,
## # nota_imdb <dbl>, likes_facebook <int>, ator_1 <chr>, ator_2 <chr>,
## # ator_3 <chr>
```

O operador `%in%` é muito útil na hora de criar filtros. O resultado das operações com `%in%` é um vetor lógico o tamanho do vetor do elemento da esquerda, identificando quais elementos da esquerda batem com algum elemento da direita.

```
letters %in% c("a", "e", "z")
```

```
## [1] TRUE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE TRUE
```

```
imdb %>%
  filter(diretor %in% c("Steven Spielberg", "Quentin Tarantino"))
```

```
## # A tibble: 32 x 15
##   titulo    ano diretor duracao cor generos pais classificacao orcamento
##   <chr> <int> <chr> <int> <chr> <chr> <chr> <chr> <int>
## 1 India~ 2008 Steven~ 122 Color Action~ USA A partir de ~ 185000000
## 2 War o~ 2005 Steven~ 116 Color Advent~ USA A partir de ~ 132000000
## 3 The A~ 2011 Steven~ 107 Color Action~ USA Livre 135000000
## 4 Minor~ 2002 Steven~ 145 Color Action~ USA A partir de ~ 102000000
## 5 Djang~ 2012 Quenti~ 165 Color Drama|~ USA A partir de ~ 100000000
## 6 A.I. ~ 2001 Steven~ 146 Color Advent~ USA A partir de ~ 100000000
## 7 The L~ 1997 Steven~ 129 Color Action~ USA A partir de ~ 73000000
## 8 The T~ 2004 Steven~ 128 Color Comedy~ USA A partir de ~ 60000000
## 9 Inglo~ 2009 Quenti~ 153 Color Advent~ USA A partir de ~ 75000000
## 10 Hook 1991 Steven~ 142 Color Advent~ USA Livre 70000000
## # ... with 22 more rows, and 6 more variables: receita <int>, nota_imdb <dbl>,
## # likes_facebook <int>, ator_1 <chr>, ator_2 <chr>, ator_3 <chr>
```

Também podemos usar funções que retornam valores lógicos, como a `str_detect()`, do pacote `stringr`. Esse pacote possui funções para a manipulação de strings, e será abordado com mais detalhes quando falarmos sobre `stringr`.

```
library(stringr)

imdb %>%
  filter(str_detect(generos, "Action"))
```

```
## # A tibble: 861 x 15
##   titulo    ano diretor duracao cor  generos pais classificacao orcamento
##   <chr>    <int> <chr>      <int> <chr> <chr>   <chr> <chr>          <int>
## 1 Avata~   2009 James ~    178 Color Action~ USA   A partir de ~ 237000000
## 2 Pirat~   2007 Gore V~    169 Color Action~ USA   A partir de ~ 300000000
## 3 The D~   2012 Christ~    164 Color Action~ USA   A partir de ~ 250000000
## 4 John ~   2012 Andrew~    132 Color Action~ USA   A partir de ~ 263700000
## 5 Spide~   2007 Sam Ra~    156 Color Action~ USA   A partir de ~ 258000000
## 6 Aveng~   2015 Joss W~    141 Color Action~ USA   A partir de ~ 250000000
## 7 Batma~   2016 Zack S~    183 Color Action~ USA   A partir de ~ 250000000
## 8 Super~   2006 Bryan ~    169 Color Action~ USA   A partir de ~ 209000000
## 9 Pirat~   2006 Gore V~    151 Color Action~ USA   A partir de ~ 225000000
## 10 The L~  2013 Gore V~    150 Color Action~ USA   A partir de ~ 215000000
## # ... with 851 more rows, and 6 more variables: receita <int>, nota_imdb <dbl>,
## #   likes_facebook <int>, ator_1 <chr>, ator_2 <chr>, ator_3 <chr>
```

6.2.2 select()

A função `select()` seleciona colunas (variáveis). É possível utilizar nomes, índices, intervalos de variáveis ou utilizar as funções `starts_with(x)`, `contains(x)`, `matches(x)`, `one_of(x)` para selecionar as variáveis.

```
imdb %>%
  select(titulo, ano, orcamento)
```

```
## # A tibble: 3,807 x 3
##   titulo                                ano orcamento
##   <chr>                                <int>    <int>
## 1 Avatar                                2009 237000000
## 2 Pirates of the Caribbean: At World's End 2007 300000000
## 3 The Dark Knight Rises                  2012 250000000
## 4 John Carter                           2012 263700000
## 5 Spider-Man 3                           2007 258000000
## 6 Tangled                               2010 260000000
## 7 Avengers: Age of Ultron                 2015 250000000
## 8 Batman v Superman: Dawn of Justice       2016 250000000
## 9 Superman Returns                       2006 209000000
## 10 Pirates of the Caribbean: Dead Man's Chest 2006 225000000
## # ... with 3,797 more rows
```

```
imdb %>%
  select(starts_with("ator"))
```

```
## # A tibble: 3,807 x 3
##   ator_1      ator_2      ator_3
##   <chr>      <chr>      <chr>
## 1 CCH Pounder Joel David Moore Wes Studi
## 2 Johnny Depp Orlando Bloom Jack Davenport
## 3 Tom Hardy Christian Bale Joseph Gordon-Levitt
## 4 Daryl Sabara Samantha Morton Polly Walker
## 5 J.K. Simmons James Franco Kirsten Dunst
## 6 Brad Garrett Donna Murphy M.C. Gainey
## 7 Chris Hemsworth Robert Downey Jr. Scarlett Johansson
## 8 Henry Cavill Lauren Cohan Alan D. Purwin
## 9 Kevin Spacey Marlon Brando Frank Langella
## 10 Johnny Depp Orlando Bloom Jack Davenport
## # ... with 3,797 more rows
```

O operador `:` pode ser usado para selecionar intervalos de colunas.

```
imdb %>%
  select(titulo, ator_1:ator_3)
```

```
## # A tibble: 3,807 x 4
##   titulo      ator_1      ator_2      ator_3
##   <chr>      <chr>      <chr>      <chr>
## 1 Avatar      CCH Pounder Joel David M- Wes Studi
## 2 Pirates of the Caribbean: At Wor~ Johnny Depp Orlando Bloom Jack Davenport
## 3 The Dark Knight Rises      Tom Hardy Christian Ba~ Joseph Gordon~
## 4 John Carter      Daryl Sabara Samantha Mor~ Polly Walker
## 5 Spider-Man 3      J.K. Simmons James Franco Kirsten Dunst
## 6 Tangled      Brad Garrett Donna Murphy M.C. Gainey
## 7 Avengers: Age of Ultron      Chris Hemswo~ Robert Downe~ Scarlett Johan~
## 8 Batman v Superman: Dawn of Justi~ Henry Cavill Lauren Cohan Alan D. Purwin
## 9 Superman Returns      Kevin Spacey Marlon Brando Frank Langella
## 10 Pirates of the Caribbean: Dead M~ Johnny Depp Orlando Bloom Jack Davenport
## # ... with 3,797 more rows
```

Para retirar colunas da base, basta acrescentar um `-` antes da seleção.

```
imdb %>%
  select(-ano, - diretor)
```

```
## # A tibble: 3,807 x 13
##   titulo duracao cor generos pais classificacao orcamento receita nota_imdb
##   <chr> <int> <chr> <chr> <chr> <chr> <int> <int> <dbl>
## 1 Avata~ 178 Color Action~ USA A partir de ~ 237000000 7.61e8 7.9
## 2 Pirat~ 169 Color Action~ USA A partir de ~ 300000000 3.09e8 7.1
```

```
## 3 The D~      164 Color Action~ USA  A partir de ~ 250000000 4.48e8      8.5
## 4 John ~      132 Color Action~ USA  A partir de ~ 263700000 7.31e7      6.6
## 5 Spide~      156 Color Action~ USA  A partir de ~ 258000000 3.37e8      6.2
## 6 Tangl~      100 Color Advent~ USA  Livre           260000000 2.01e8      7.8
## 7 Aveng~      141 Color Action~ USA  A partir de ~ 250000000 4.59e8      7.5
## 8 Batma~      183 Color Action~ USA  A partir de ~ 250000000 3.30e8      6.9
## 9 Super~      169 Color Action~ USA  A partir de ~ 209000000 2.00e8      6.1
## 10 Pirat~     151 Color Action~ USA  A partir de ~ 225000000 4.23e8      7.3
## # ... with 3,797 more rows, and 4 more variables: likes_facebook <int>,
## #   ator_1 <chr>, ator_2 <chr>, ator_3 <chr>
```

```
imdb %>%
  select(-starts_with("ator"))
```

```
## # A tibble: 3,807 x 12
##   titulo    ano diretor duracao cor   generos pais classificacao orcamento
##   <chr>    <int> <chr>      <int> <chr> <chr>   <chr> <chr>          <int>
## 1 Avata~   2009 James ~    178 Color Action~ USA  A partir de ~ 237000000
## 2 Pirat~   2007 Gore V~    169 Color Action~ USA  A partir de ~ 300000000
## 3 The D~   2012 Christ~    164 Color Action~ USA  A partir de ~ 250000000
## 4 John ~   2012 Andrew~    132 Color Action~ USA  A partir de ~ 263700000
## 5 Spide~   2007 Sam Ra~    156 Color Action~ USA  A partir de ~ 258000000
## 6 Tangl~   2010 Nathan~    100 Color Advent~ USA  Livre           260000000
## 7 Aveng~   2015 Joss W~    141 Color Action~ USA  A partir de ~ 250000000
## 8 Batma~   2016 Zack S~    183 Color Action~ USA  A partir de ~ 250000000
## 9 Super~   2006 Bryan ~    169 Color Action~ USA  A partir de ~ 209000000
## 10 Pirat~  2006 Gore V~    151 Color Action~ USA  A partir de ~ 225000000
## # ... with 3,797 more rows, and 3 more variables: receita <int>,
## #   nota_imdb <dbl>, likes_facebook <int>
```

6.2.3 arrange()

A função `arrange()` ordena a base segundo uma ou mais colunas. O argumento `desc=` pode ser utilizado para gerar uma ordem decrescente.

```
imdb %>%
  arrange(orcamento) %>%
  select(orcamento, everything())
```

```
## # A tibble: 3,807 x 15
##   orcamento titulo    ano diretor duracao cor   generos pais classificacao
##   <int> <chr>    <int> <chr>      <int> <chr> <chr>   <chr> <chr>
## 1      218 Tarna~   2003 Jonath~    88 Color Biogra~ USA  Outros
## 2     1100 My Da~   2004 Jon Gu~    90 Color Docume~ USA  Livre
## 3     1400 A Pla~   2013 Benjam~    76 Color Drama|~ USA  Outros
```

```
## 4      3250 The M~ 2005 Anthon~      84 Color Crime|~ USA  A partir de ~
## 5      7000 Prime~ 2004 Shane ~      77 Color Drama|~ USA  A partir de ~
## 6      7000 El Ma~ 1992 Robert~      81 Color Action~ USA  A partir de ~
## 7      9000 Newly~ 2011 Edward~      95 Color Comedy~ USA  Outros
## 8     10000 Pink ~ 1972 John W~     108 Color Comedy~ USA  A partir de ~
## 9     13000 The T~ 2007 Jane C~       7 Color Romanc~ USA  Outros
## 10     15000 Paran~ 2007 Oren P~      84 Color Horror~ USA  A partir de ~
## # ... with 3,797 more rows, and 6 more variables: receita <int>,
## #   nota_imdb <dbl>, likes_facebook <int>, ator_1 <chr>, ator_2 <chr>,
## #   ator_3 <chr>
```

```
imdb %>%
  arrange(desc(orcamento)) %>%
  select(orcamento, everything())
```

```
## # A tibble: 3,807 x 15
##   orcamento titulo    ano diretor duracao cor   generos pais classificacao
##   <int> <chr> <int> <chr>    <int> <chr> <chr> <chr> <chr>
## 1 300000000 Pirat~ 2007 Gore V~    169 Color Action~ USA  A partir de ~
## 2 263700000 John ~ 2012 Andrew~    132 Color Action~ USA  A partir de ~
## 3 260000000 Tangl~ 2010 Nathan~    100 Color Advent~ USA  Livre
## 4 258000000 Spide~ 2007 Sam Ra~    156 Color Action~ USA  A partir de ~
## 5 258000000 Spide~ 2007 Sam Ra~    156 Color Action~ USA  A partir de ~
## 6 250000000 The D~ 2012 Christ~    164 Color Action~ USA  A partir de ~
## 7 250000000 Aveng~ 2015 Joss W~    141 Color Action~ USA  A partir de ~
## 8 250000000 Batma~ 2016 Zack S~    183 Color Action~ USA  A partir de ~
## 9 250000000 Pirat~ 2011 Rob Ma~    136 Color Action~ USA  A partir de ~
## 10 250000000 Capta~ 2016 Anthon~    147 Color Action~ USA  A partir de ~
## # ... with 3,797 more rows, and 6 more variables: receita <int>,
## #   nota_imdb <dbl>, likes_facebook <int>, ator_1 <chr>, ator_2 <chr>,
## #   ator_3 <chr>
```

```
imdb %>%
  arrange(desc(ano), titulo) %>%
  select(titulo, ano, everything())
```

```
## # A tibble: 3,807 x 15
##   titulo    ano diretor duracao cor   generos pais classificacao orcamento
##   <chr> <int> <chr>    <int> <chr> <chr> <chr> <chr> <int>
## 1 10 Cl~ 2016 Dan Tr~    104 Color Drama|~ USA  A partir de ~ 15000000
## 2 13 Ho~ 2016 Michae~    144 Color Action~ USA  A partir de ~ 50000000
## 3 A Beg~ 2016 Mitche~     87 Color Comedy~ USA  Outros NA
## 4 Alice~ 2016 James ~    113 Color Advent~ USA  Livre 170000000
## 5 Alleg~ 2016 Robert~    120 Color Action~ USA  A partir de ~ 110000000
## 6 Allel~ 2016 Darren~     97 Color Horror~ USA  Outros 500000
## 7 Antib~ 2016 Danny ~     94 Color Horror~ USA  Outros 3500000
## 8 Bad M~ 2016 Jon Lu~    100 Color Comedy~ USA  A partir de ~ 20000000
```



```
## 9 Bad M~ 2016 Jon Lu~      100 Color Comedy USA A partir de ~ 20000000
## 10 Batma~ 2016 Zack S~     183 Color Action~ USA A partir de ~ 250000000
## # ... with 3,797 more rows, and 6 more variables: receita <int>,
## #   nota_imdb <dbl>, likes_facebook <int>, ator_1 <chr>, ator_2 <chr>,
## #   ator_3 <chr>
```

6.2.4 mutate()

A função `mutate()` cria ou modifica colunas. Ela é equivalente à função `transform()`, mas aceita várias novas colunas iterativamente. Novas variáveis devem ter o mesmo número de linhas da base original (ou comprimento 1).

```
# A coluna "duracao" é sobrescrita
imdb %>%
  mutate(duracao = duracao/60) %>%
  select(duracao)
```

```
## # A tibble: 3,807 x 1
##   duracao
##   <dbl>
## 1    2.97
## 2    2.82
## 3    2.73
## 4    2.2
## 5    2.6
## 6    1.67
## 7    2.35
## 8    3.05
## 9    2.82
## 10   2.52
## # ... with 3,797 more rows
```

```
# Criamos uma nova coluna na base
imdb %>%
  mutate(duracao_horas = duracao/60) %>%
  select(duracao, duracao_horas)
```

```
## # A tibble: 3,807 x 2
##   duracao duracao_horas
##   <int>      <dbl>
## 1    178         2.97
## 2    169         2.82
## 3    164         2.73
## 4    132         2.2
## 5    156         2.6
```

```
## 6      100      1.67
## 7      141      2.35
## 8      183      3.05
## 9      169      2.82
## 10     151      2.52
## # ... with 3,797 more rows

# Podemos fazer diversas operações em um mesmo mutate.
imdb %>%
  mutate(
    lucro = receita - orcamento,
    resultado = ifelse(lucro < 0, "prejuizo", "lucro")
  ) %>%
  select(lucro, resultado)

## # A tibble: 3,807 x 2
##       lucro resultado
##       <int> <chr>
## 1  523505847 lucro
## 2   9404152 lucro
## 3  198130642 lucro
## 4 -190641321 prejuizo
## 5   78530303 lucro
## 6  -59192738 prejuizo
## 7  208991599 lucro
## 8   80249062 lucro
## 9   -8930592 prejuizo
## 10 198032628 lucro
## # ... with 3,797 more rows
```

6.2.5 summarise()

A função `summarise()` sumariza a base. Ela aplica uma função às variáveis, retornando um vetor de tamanho 1. Ela é utilizada em conjunto da função `group_by()`. A função `n()` costuma ser bastante utilizada com a função `summarise()`.

```
imdb %>%
  summarise(media_orcamento = mean(orcamento, na.rm = TRUE))

## # A tibble: 1 x 1
##   media_orcamento
##       <dbl>
## 1      35755986.
```

```
imdb %>%
  summarise(
    media_orcamento = mean(orcamento, na.rm = TRUE),
    mediana_orcamento = median(orcamento, na.rm = TRUE),
    qtd = n(),
    qtd_diretores = n_distinct(diretor)
  )

## # A tibble: 1 x 4
##   media_orcamento mediana_orcamento   qtd qtd_diretores
##         <dbl>             <int> <int>         <int>
## 1      35755986.             20000000 3807             1813
```

```
imdb %>%
  group_by(ano) %>%
  summarise(qtd_filmes = n())

## # A tibble: 91 x 2
##   ano qtd_filmes
##   <int>     <int>
## 1  1916         1
## 2  1920         1
## 3  1925         1
## 4  1929         1
## 5  1930         1
## 6  1932         1
## 7  1933         2
## 8  1934         1
## 9  1935         1
## 10 1936         2
## # ... with 81 more rows
```

```
imdb %>%
  group_by(diretor) %>%
  summarise(orcamento_medio = mean(orcamento, na.rm = TRUE))

## # A tibble: 1,813 x 2
##   diretor          orcamento_medio
##   <chr>             <dbl>
## 1 A. Raven Cruz      1000000
## 2 Aaron Hann         NaN
## 3 Aaron Schneider    7500000
## 4 Aaron Seltzer      20000000
## 5 Abel Ferrara       12500000
## 6 Adam Carolla        1500000
## 7 Adam Goldberg      1650000
## 8 Adam Green         1500000
```

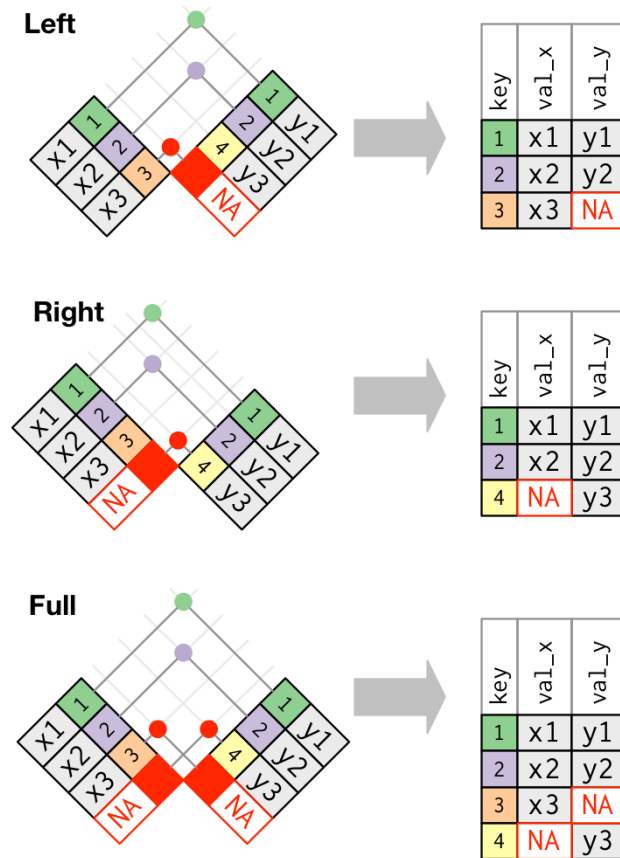
```
## 9 Adam Jay Epstein      NaN
## 10 Adam Marcus          2500000
## # ... with 1,803 more rows
```

6.2.6 `_join()`

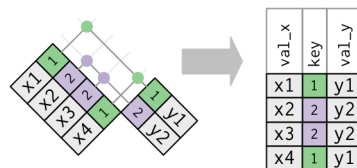
Para juntar duas tabelas de dados, podemos utilizar a família de funções `_join()` do `dplyr`. Essas funções geralmente recebem três argumentos: uma base esquerda (`x=`), uma base direita (`y=`) e uma chave `by=`. As principais funções `_join()` são:

- `left_join(x, y)`: retorna todas as linhas da base `x` e todas as colunas das bases `x` e `y`. Linhas de `x` sem correspondentes em `y` receberão `NA` na nova base.
- `right_join()`: retorna todas as linhas da base `y` e todas as colunas das bases `x` e `y`. Linhas de `y` sem correspondentes em `x` receberão `NA` na nova base.
- `full_join()`: retorna todas as linhas e colunas de `x` e `y`. Valores sem correspondência entre as bases receberão `NA` na nova base.

A figura a seguir esquematiza as operações dessas funções:



A figura a seguir mostra o que acontece quando temos chaves duplicadas em um `left_join()`. A ideia é equivalente para as outras funções.



6.3 tidyr

O pacote `tidyr` dispõe de funções úteis para deixar os seus dados no formato que você precisa para a análise. Na maioria das vezes, utilizamos para deixá-los *tidy*. Outras, precisamos “bagunçá-los” um pouco para poder aplicar alguma função específica.

As principais funções deste pacote são a `gather()` e a `spread()`

6.3.1 gather()

A função `gather()` “empilha” o banco de dados. Ela é utilizada principalmente quando as colunas da base não representam nomes de variáveis, mas sim seus valores.

```
library(tidyr)

imdb <- readr::read_rds("data/imdb.rds")

imdb_gather <- imdb %>%
  mutate(id = 1:n()) %>%
  gather(
    key = "importancia_ator",
    value = "nome_ator",
    ator_1, ator_2, ator_3
  ) %>%
  select(nome_ator, importancia_ator, everything())

imdb_gather
```

```
## # A tibble: 11,421 x 15
##   nome_ator importancia_ator titulo   ano diretor duracao cor   generos pais
##   <chr>      <chr>          <chr> <int> <chr>      <int> <chr> <chr> <chr>
## 1 CCH Poun~ ator_1        Avata~ 2009 James ~    178 Color Action~ USA
## 2 Johnny D~ ator_1        Pirat~ 2007 Gore V~    169 Color Action~ USA
## 3 Tom Hardy ator_1        The D~ 2012 Christ~   164 Color Action~ USA
## 4 Daryl Sa~ ator_1        John ~ 2012 Andrew~   132 Color Action~ USA
## 5 J.K. Sim~ ator_1        Spide~ 2007 Sam Ra~   156 Color Action~ USA
## 6 Brad Gar~ ator_1        Tangl~ 2010 Nathan~   100 Color Advent~ USA
## 7 Chris He~ ator_1        Aveng~ 2015 Joss W~   141 Color Action~ USA
## 8 Henry Ca~ ator_1        Batma~ 2016 Zack S~   183 Color Action~ USA
## 9 Kevin Sp~ ator_1        Super~ 2006 Bryan ~   169 Color Action~ USA
## 10 Johnny D~ ator_1        Pirat~ 2006 Gore V~   151 Color Action~ USA
## # ... with 11,411 more rows, and 6 more variables: classificacao <chr>,
## #   orcamento <int>, receita <int>, nota_imdb <dbl>, likes_facebook <int>,
## #   id <int>
```

6.3.2 spread()

A função `spread()` é essencialmente o inverso da `gather()`. Ela espalha uma variável nas colunas.

```
imdb_spread <- imdb_gather %>%
  spread(
    key = importancia_ator,
    value = nome_ator
  )

imdb_spread
```

```
## # A tibble: 3,807 x 16
##   titulo  ano diretor duracao cor generos pais classificacao orcamento
##   <chr>  <int> <chr>    <int> <chr> <chr>  <chr> <chr>          <int>
## 1 Avata~  2009 James ~    178 Color Action~ USA  A partir de ~ 237000000
## 2 Pirat~  2007 Gore V~    169 Color Action~ USA  A partir de ~ 300000000
## 3 The D~  2012 Christ~    164 Color Action~ USA  A partir de ~ 250000000
## 4 John ~  2012 Andrew~    132 Color Action~ USA  A partir de ~ 263700000
## 5 Spide~  2007 Sam Ra~    156 Color Action~ USA  A partir de ~ 258000000
## 6 Tangl~  2010 Nathan~    100 Color Advent~ USA  Livre        260000000
## 7 Aveng~  2015 Joss W~    141 Color Action~ USA  A partir de ~ 250000000
## 8 Batma~  2016 Zack S~    183 Color Action~ USA  A partir de ~ 250000000
## 9 Super~  2006 Bryan ~    169 Color Action~ USA  A partir de ~ 209000000
## 10 Pirat~  2006 Gore V~    151 Color Action~ USA  A partir de ~ 225000000
## # ... with 3,797 more rows, and 7 more variables: receita <int>,
## #   nota_imdb <dbl>, likes_facebook <int>, id <int>, ator_1 <chr>,
## #   ator_2 <chr>, ator_3 <chr>
```

6.3.3 Outras funções do tidyR

- A função `unite()` junta duas ou mais colunas usando algum separador (`_`, por exemplo).
- A função `separate()` faz o inverso de `unite()`: transforma uma coluna em várias usando um separador.

```
imdb %>%
  unite(
    col = "titulo_diretor",
    titulo, diretor,
    sep = " - "
  )
```

```
## # A tibble: 3,807 x 14
##   titulo_diretor  ano duracao cor  generos pais classificacao orcamento
##   <chr>          <int>  <int> <chr> <chr>  <chr> <chr>          <int>
## 1 Avatar - Jam~ 2009      178 Color Action~ USA   A partir de ~ 237000000
## 2 Pirates of th~ 2007      169 Color Action~ USA   A partir de ~ 300000000
## 3 The Dark Knig~ 2012      164 Color Action~ USA   A partir de ~ 250000000
## 4 John Carter ~ 2012      132 Color Action~ USA   A partir de ~ 263700000
## 5 Spider-Man 3 ~ 2007      156 Color Action~ USA   A partir de ~ 258000000
## 6 Tangled - Na~ 2010      100 Color Advent~ USA   Livre          260000000
## 7 Avengers: Age~ 2015      141 Color Action~ USA   A partir de ~ 250000000
## 8 Batman v Supe~ 2016      183 Color Action~ USA   A partir de ~ 250000000
## 9 Superman Retu~ 2006      169 Color Action~ USA   A partir de ~ 209000000
## 10 Pirates of th~ 2006      151 Color Action~ USA   A partir de ~ 225000000
## # ... with 3,797 more rows, and 6 more variables: receita <int>,
## #   nota_imdb <dbl>, likes_facebook <int>, ator_1 <chr>, ator_2 <chr>,
## #   ator_3 <chr>
```

```
imdb %>%
  separate(
    col = generos,
    into = c("genero_1", "genero_2", "genero_3"),
    sep = "\\|",
    extra = "drop"
  )
```

```
## # A tibble: 3,807 x 17
##   titulo  ano diretor duracao cor  genero_1 genero_2 genero_3 pais
##   <chr>  <int> <chr>    <int> <chr> <chr>    <chr>    <chr>    <chr>
## 1 Avata~ 2009 James ~    178 Color Action  Adventu~ Fantasy  USA
## 2 Pirat~ 2007 Gore V~    169 Color Action  Adventu~ Fantasy  USA
## 3 The D~ 2012 Christ~    164 Color Action  Thriller <NA>  USA
## 4 John ~ 2012 Andrew~    132 Color Action  Adventu~ Sci-Fi   USA
## 5 Spide~ 2007 Sam Ra~    156 Color Action  Adventu~ Romance  USA
## 6 Tangl~ 2010 Nathan~    100 Color Adventu~ Animati~ Comedy   USA
## 7 Aveng~ 2015 Joss W~    141 Color Action  Adventu~ Sci-Fi   USA
## 8 Batma~ 2016 Zack S~    183 Color Action  Adventu~ Sci-Fi   USA
## 9 Super~ 2006 Bryan ~    169 Color Action  Adventu~ Sci-Fi   USA
## 10 Pirat~ 2006 Gore V~    151 Color Action  Adventu~ Fantasy  USA
## # ... with 3,797 more rows, and 8 more variables: classificacao <chr>,
## #   orcamento <int>, receita <int>, nota_imdb <dbl>, likes_facebook <int>,
## #   ator_1 <chr>, ator_2 <chr>, ator_3 <chr>
```