Introdução à programação com R

Material complementar da Aula 2

curso-r

Tópicos desta aula

- Pacotes
- Importação de bases de dados
- Tabelas: Data.frames
- Operadores
- Mais sobre Data.frames

Pacotes

Pacotes

Um pacote no R é um conjunto de funções que visam resolver um problema em específico. O R já vem com alguns pacotes instalados. Geralmente chamamos esses pacotes de *base R*.

Mas a força do R está na gigantesca variedade de pacotes desenvolvidos pela comunidade, em especial, pelos criadores do `tidyverse`.

Instalando e carregando pacotes

Para instalar um pacote, usamos a função `install.packages`.

```
# Instalando um pacote
install.packages("tidyverse")

# Instalando vários pacotes de uma vez
install.packages(c("tidyverse", "rmarkdown", "devtools"))
```

Para usar as funções de um pacote, precisamos carregá-lo. Fazemos isso usando a função `library()`.

```
library(tidyverse)
```

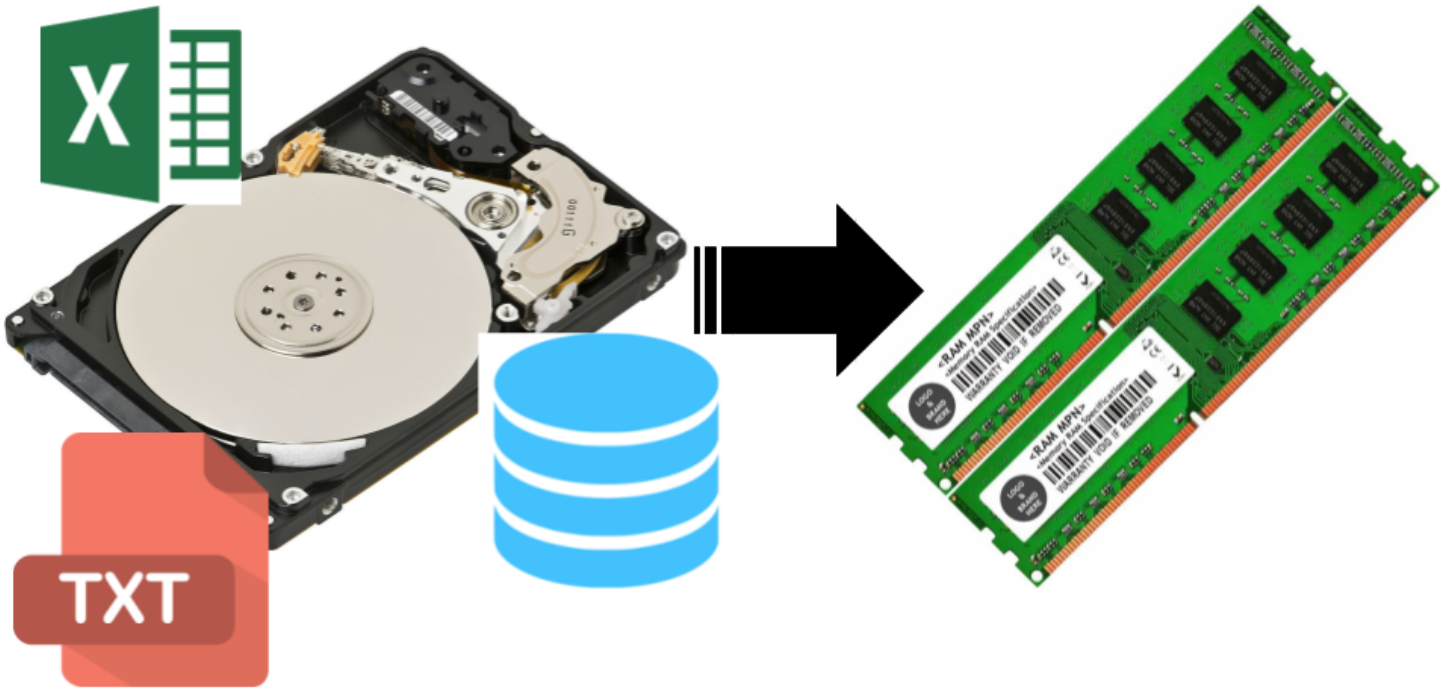
Instale uma vez, carregue várias vezes!

Importação de bases de dados

Importação de bases de dados

O que é?

Importar uma base de dados para o R significa levar a informação contida no disco rígido (HD) para a mé moria RAM.



curso-r

Lendo tabelas

Para ler tabelas, como arquivos `.csv`, utilizaremos funções do pacote `readr`.

Para isso, utilizamos a função `read_csv()` ou `read_csv2()`. Se o arquivo estiver bem formatado, a função só precisa do caminho até o arquivo para funcionar.

A mensagem devolvida pela função indica qual classe foi atribuída para cada coluna da base.

```
## ℹ Using ',' as decimal and '.' as grouping mark. Use `read_delim()` for more control.

##
## ── Column specification ────────────────────────────────────────────────────────
## cols(
##   ano = col_double(),
##   mes = col_double(),
##   dia = col_double(),
##   horario_saida = col_double(),
##   saida_programada = col_double(),
##   atraso_saida = col_double(),
##   horario_chegada = col_double(),
##   chegada_prevista = col_double(),
##   atraso_chegada = col_double(),
##   companhia_aerea = col_character(),
##   voo = col_double(),
##   cauda = col_character(),
##   origem = col_character(),
##   destino = col_character(),
##   tempo_voo = col_double(),
##   distancia = col_double(),
##   hora = col_double(),
##   minuto = col_double(),
##   data_hora = col_datetime(format = "")
## )
```

- Em alguns países, como o Brasil, as vírgulas são utilizadas para separar as casas decimais dos números, inviabilizando os arquivos `.csv`. Nesses casos, os arquivos `.csv` são na verdade separados por ponto-e-vírgula. Para ler bases separadas por ponto-e-vírgula no R, utilize a função `read_csv2()`.

```
voos_csv <- readr::read_csv2("../dados/voos_de_janeiro.csv")
```

- Arquivos `.txt` podem ser lidos com a função `read_delim()`. Além do caminho até o arquivo, você também precisa indicar qual é o caractere utilizado para separar as colunas da base. Um arquivo separado por tabulação, por exemplo, pode ser lido utilizando a o código abaixo. O código `\t` é uma forma textual de representar a tecla TAB.

- Para ler planilhas do Excel (arquivos `.xlsx` ou `.xls`), basta utilizarmos a função `read_excel()` do pacote `readxl`.

Tabelas no R: Data frames

Tabelas no r: Data frames

O objeto mais importante para o cientista de dados é, claro, a base de dados. No R, uma base de dados é representa por objetos chamados de *data frames*. Eles são equivalentes a uma tabela do SQL ou uma planilha do Excel.

A principal característica de um *data frame* é possuir linhas e colunas:

```
mtcars
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

O `mtcars` é um *data frame* nativo do R que contém informações sobre diversos modelos de carros. Ele possui 32 linhas e 11 colunas (só estamos vendo as primeiras 10 linhas no slide anterior).

A primeira "coluna" representa apenas o *nome* das linhas (modelo do carro), não é uma coluna da base. Repare que ela não possui um nome, como as outras. Essa estrutura de nome de linha é própria de *data frames* no R. Se exportássemos essa base para o Excel, por exemplo, essa coluna não apareceria.

Se você quiser saber mais sobre o `mtcars`, veja a documentação dele rodando `?mtcars` no **Console**.

Para entender melhor sobre *data frames*, precisamos estudar um pouco sobre classes, vetores e testes lógicos.

Classes

A classe de um objeto é muito importante dentro do R. É a partir dela que as funções e operadores conseguem saber exatamente o que fazer com um objeto.

Por exemplo, podemos somar dois números, mas não conseguimos somar duas letras (texto):

```
1 + 1
```

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

```
"a" + "b"
```

```
## Error in "a" + "b": argumento não-numérico para operador binário
```

O operador `+` verifica que `"a"` e `"b"` não são números (ou que a classe deles não é numérica) e devolve uma mensagem de erro informando isso.

Texto

Observe que para criar texto no R, colocamos os caracteres entre aspas. As aspas servem para diferenciar *nomes* (objetos, funções, pacotes) de *textos* (letras e palavras). Os textos são muito comuns em variáveis categóricas.

```
a <- 10
# O objeto `a`, sem aspas
a
```

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

```
# A letra (texto) `a`, com aspas
"a"
```

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

A classe de um objeto

Para saber a classe de um objeto, basta rodarmos `class(nome-do-objeto)`.

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

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

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

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

```
class(mtcars)
```

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

Operadores

Operações lógicas

Uma operação lógica é um teste que retorna **verdadeiro** ou **falso**. No R (e em outras linguagens de programação), esses valores dois valores recebem uma classe especial: `logical`.

O verdadeiro no R vai ser representado pelo valor `TRUE` e o falso pelo valor `FALSE`. Esses nomes no R são **reservados**, isto é, você não pode chamar nenhum objeto de TRUE ou FALSE.

```
TRUE <- 1
## Error in TRUE <- 1 : invalid (do_set) left-hand side to assignment
```

Valores lógicos

Checando a classe desses valores, vemos que são lógicos (também conhecidos como valores binários ou booleanos). Eles são os únicos possíveis valores dessa classe.

```
class(TRUE)
```

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

```
class(FALSE)
```

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

Agora que conhecemos o `TRUE` e `FALSE`, podemos explorar os teste lógicos.

Igualdades

Começando pela igualdade: vamos testar se um valor é igual ao outro. Para isso, usamos o operador `==`.

```
# Testes com resultado verdadeiro
1 == 1
```

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

```
"a" == "a"
```

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

```
# Testes com resultado falso
1 == 2
```

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

```
"a" == "b"
```

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

Diferenças

Também podemos testar se dois valores são diferentes. Para isso, usamos o operador `!=`.

```
# Testes com resultado falso
1 != 1
```

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

```
"a" != "a"
```

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

```
# Testes com resultado verdadeiro
1 != 2
```

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

```
"a" != "b"
```

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

Desigualdades

Para comparar se um valor é maior que outro, temos à disposição 4 operadores:

```
# Maior
3 > 3
```

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

```
3 > 2
```

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

```
# Maior ou igual
3 > 4
```

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

```
3 >= 3
```

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

```r
# Menor
3 < 3
```

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

```r
3 < 4
```

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

```r
# Menor ou igual
3 < 2
```

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

```r
3 <= 3
```

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

Pertence

Um outro operador muito útil é o `%in%`. Com ele, podemos verificar se um valor está dentro de um conjunto de valores (vetor).

```
3 %in% c(1, 2, 3)
```

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

```
"a" %in% c("b", "c")
```

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

Filtros

Os testes lógicos fazem parte de uma operação muito comum na manipulação de base de dados: os **filtros**. No Excel, por exemplo, quando você filtra uma planilha, o que está sendo feito por trás é um teste lógico.

Falamos anteriormente que cada coluna das nossas bases de dados será representada dentro do R como um vetor. O comportamento que explica a importância dos testes lógicos na hora de filtrar uma base está ilustrado abaixo:

```
minha_coluna <- c(1, 3, 0, 10, -1, 5, 20)
minha_coluna > 3
```

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

```
minha_coluna[minha_coluna > 3]
```

```
## [1] 10  5 20
```

Muitas coisas aconteceram no código anterior, vamos por partes.

Primeiro, na operação `minha_coluna > 3` o R fez um excelente uso do comportamento de reciclagem. No fundo, o que ele fez foi transformar (reciclar) o valor `3` no vetor `c(3, 3, 3, 3, 3, 3, 3)` e testar se `c(1, 3, 0, 10, -1, 5, 20) > c(3, 3, 3, 3, 3, 3, 3)`.

Como os operadores lógicos também são vetorizados (fazem operações elemento a elemento), os testes realizados foram `1 > 3`, `3 > 3`, `0 > 3`, `10 > 3`, `-1 > 3`, `5 > 3` e, finalmente, `20 > 3`. Cada um desses testes tem o seu próprio resultado. Por isso a saída de `minha_coluna > 3` é um vetor de verdadeiros e falsos, respectivos a cada um desses 7 testes.

A segunda operação traz a grande novidade aqui: podemos usar os valores `TRUE` e `FALSE` para selecionar elementos de um vetor!

A regra é a seguinte: **retornar** as posições que receberem `TRUE`, **não retornar** as posições que receberem `FALSE`.

Portanto, a segunda operação é equivalente a:

```
minha_coluna[c(FALSE, FALSE, FALSE, TRUE, FALSE, TRUE, TRUE)]
```

```
## [1] 10  5 20
```

O vetor lógico filtra o vetor `minha_coluna`, retornando apenas os valores maiores que 3, já que foi esse o teste lógico que fizemos.

Essa é a *mágica* que acontece por trás de filtros no R. Na prática, não precisaremos usar colchetes, não lembraremos da reciclagem e nem veremos a cara dos `TRUE` e `FALSE`. Mas conhecer esse processo é muito importante, principalmente para encontrar problemas de código ou de base.

Mais sobre data frames

Chegou a hora de usarmos tudo o que aprendemos na seção anterior para exploramos ao máximo o nosso objeto favorito: o *data frame*.

Para isso, continuaremos a usar o `mtcars`.

```
mtcars
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

Acessando as colunas

Lembrando que cada coluna de um *data frame* é um vetor, podemos usar o operador `$` para acessar cada uma de suas colunas.

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4
## [17] 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
```

```
mtcars$cyl
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
```

```
mtcars$wt
```

```
##  [1] 2.620 2.875 2.320 3.215 3.440 3.460 3.570 3.190 3.150 3.440 3.440 4.070 3.730
## [14] 3.780 5.250 5.424 5.345 2.200 1.615 1.835 2.465 3.520 3.435 3.840 3.845 1.935
## [27] 2.140 1.513 3.170 2.770 3.570 2.780
```

Dimensões

A classe *data frame* possui uma característica especial: seus objetos possuem duas **dimensões**.

```
class(mtcars)
```

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

```
dim(mtcars)
```

```
## [1] 32 11
```

O resultado do código `dim(mtcars)` nos diz que a primeira dimensão tem comprimento 32 e a segunda dimensão tem comprimento 11. Em outras palavras: a base `mtcars` tem 32 linhas e 11 colunas.

Subsetting

Ter duas dimensões significa que devemos usar dois índices para acessar os valores de um *data frame* (fazer *subsetting*). Para isso, ainda usamos o colchete, mas agora com dois argumentos: `[linha, coluna]`.

```
mtcars[2, 3]
```

```
## [1] 160
```

O código acima está nos devolvendo o valor presente na segunda linha da terceira coluna da base `mtcars`.

Também podemos pegar todos as linhas de uma coluna ou todas as colunas de uma linha deixando um dos argumentos vazio:

```
# Todas as linhas da coluna 1
mtcars[,1]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4
## [17] 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
```

```
# Todas as colunas da linha 1
mtcars[1,]
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

Selecionando colunas

Podemos usar o *subsetting* para selecionar colunas:

```
mtcars[, c(1, 2)]
```

```
##                    mpg cyl
## Mazda RX4         21.0   6
## Mazda RX4 Wag     21.0   6
## Datsun 710        22.8   4
## Hornet 4 Drive    21.4   6
## Hornet Sportabout 18.7   8
## Valiant           18.1   6
## Duster 360        14.3   8
## Merc 240D         24.4   4
## Merc 230          22.8   4
## Merc 280          19.2   6
```

```r
mtcars[, c("mpg", "am")]
```

```
##                    mpg am
## Mazda RX4         21.0  1
## Mazda RX4 Wag     21.0  1
## Datsun 710        22.8  1
## Hornet 4 Drive    21.4  0
## Hornet Sportabout 18.7  0
## Valiant           18.1  0
## Duster 360        14.3  0
## Merc 240D         24.4  0
## Merc 230          22.8  0
## Merc 280          19.2  0
```

Nos dois exemplos, exibimos apenas as 5 primeiras linhas do *data frame*.

Filtrando linhas

Também podemos usar o *subsetting* para filtrar linhas:

```
mtcars$cyl == 4
```

```
##  [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
## [14] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE
## [27]  TRUE  TRUE FALSE FALSE FALSE  TRUE
```

```
mtcars[mtcars$cyl == 4, ]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

O código `mtcars$cyl == 4` nos diz em quais linhas estão os carros com 4 cilindros. Quando usamos o vetor de `TRUE` e `FALSE` resultante dentro do *subsetting* das linhas em `mtcars[mtcars$cyl == 4, ]`, o R nos devolve todos as colunas dos carros com 4 cilindros. A regra é a seguinte: linha com `TRUE` é retornada, linha com `FALSE` não.

Outro exemplo:

```
mtcars[mtcars$mpg > 25, ]
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```