

R para Ciência de Dados: introdução à programação

Departamento de Estatística
Instituto de Matemática e Estatística
Universidade Federal da Bahia

Preparando o ambiente

Durante o curso

- Usaremos nas aulas: posit.cloud.
 - Recomendamos instalar e usar R com versão pelo menos 4.1: cran.r-project.org.
 - usaremos o *framework* [tidyverse](https://www.tidyverse.org):
 - Instalação: `install.packages("tidyverse")`
-

Na sua casa

- **IDE** recomendadas: [RStudio](https://www.rstudio.com) e [VSCode](https://code.visualstudio.com).
 - Caso você queira usar o [VSCode](https://code.visualstudio.com), instale a extensão da linguagem R: [REditorSupport](https://marketplace.visualstudio.com/items?itemName=ms-kvnlabs.r-editor-support).
- Outras linguagens interessantes: [python](https://www.python.org) e [julia](https://julialang.org).
 - [python](https://www.python.org): linguagem interpretada de propósito geral, contemporânea do R, simples e fácil de aprender.
 - [julia](https://julialang.org): linguagem interpretada para análise de dados, lançada em 2012, promete simplicidade e velocidade.

A linguagem R: uma introdução

O começo de tudo

O precursor do R: S.

- R é uma linguagem derivada do S.
- S foi desenvolvido em `fortran` por **John Chambers** em 1976 no **Bell Labs**.
- S foi desenvolvido para ser um ambiente de análise estatística.
- Filosofia do S: permitir que usuários possam analisar dados usando estatística com pouco conhecimento de programação.

História do R

- Em 1991, **Ross Ihaka** e **Robert Gentleman** criaram o R na **Nova Zelândia**.
- Em 1996, **Ross** e **Robert** liberam o R sob a licença “GNU General License”, o que tornou o R um software livre.
- Em 1997, **The Core Group** é criado para melhorar e controlar o código fonte do R.

Porque usar R

- Constante melhoramento e atualização.
- Portabilidade (roda em praticamente todos os sistemas operacionais).
- Grande comunidade de desenvolvedores que adicionam novas capacidades ao R através de pacotes.
- Gráficos de maneira relativamente simples.
- Interatividade.
- Um grande comunidade de usuários (especialmente útil para resolução de problemas).

Onde estudar fora de aula?

Livros

Recomendo principalmente o livro [R for Data Science](#).

- **Nível Iniciante:** [R Tutorial na W3Schools](#).
 - **Nível Iniciante:** [Hands-On Programming with R](#).
 - **Nível Iniciante:** [R for Data Science](#).
 - **Nível Intermediário:** [Advanced R](#).
-

Livros em português

- **Nível *cheguei agora aqui*:** [zen do R](#).
- **Nível Avançado:** [Advanced R](#).
- **Nível Iniciante:** [material.curso-r.com](#).
- **Nível Iniciante:** [ecoR](#).
- **Nível Iniciante:** [analises-ecologicas.com](#).

Plataformas de ensino on-line

- **Datacamp:** datacamp.com
- **Dataquest:** dataquest.io

O que você pode fazer quando estiver em apuros?

- consultar a documentação do R:

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

- Peça ajuda a um programador mais experiente.

```
sqrt("Gilberto")
```

- Pesquise por Como resolver o erro: `Error in sqrt("Gilberto"): non-numeric argument to mathematical function`, nas seguintes plataformas:
 - Consulte o erro no [chatgpt](#).
 - Consulte o erro [pt.stackoverflow.com](#).
 - Consulte [Rstudio community](#).
 - Use ferramentas de busca como o [google](#) e [duckduckgo.com](#).

Operações básicas

Soma

$$1 + 1$$

$$[1] \ 2$$

Subtração

$$2 - 1$$

$$[1] \ 1$$

Divisão

$$3 / 2$$

$$[1] \ 1.5$$

Potenciação

$$2^3$$

$$[1] \ 8$$

Operações básicas

Exercício

Qual o resultado das seguintes operações?

- 1 $5.32 + 7.99$
- 2 $5.55 - 10$
- 3 $3.33 * 5.12$
- 4 $1 / 4.55$
- 5 $5^{1.23}$

Funções na linguagem R

Função: é uma ação e tem os seguinte componentes na ordem:

- *nome da função*
- *parênteses*
- *argumentos posicionais*
- *argumentos nomeados*

nome da função *parênteses* *argumentos posicionais* *argumentos nomeados* *parênteses*
`nome_funcao (valor1, valor2, nome1 = valor3, nome2 = valor4)`

example:

```
read_excel('data/raw/casas.xlsx', sheet=1)
```

Funções na linguagem R

Exercício

- Obtenha ajuda para `mean` usando a função `help`.
- Calcule o logaritmo de 10 na base 3 usando a função `log`.
- Leia o conjunto de dados `amostra_enem_salvador.xlsx` usando a função `read_excel` do pacote `readxl`.

Estrutura de dados no R

- **Tipo de dados:** caracter (`character`), número real (`double`), número inteiro (`integer`), número complexo (`complex`) e lógico (`logical`).
- **Estrutura de dados:** atomic vector (a estrutura de dados mais básica no R), `matrix`, `array`, `list` e `data.frame` (`tibble` no `tidyverse`).
- **Estrutura de dados Homogênea:** `vector`, `matrix` e `array`.
- **Estrutura de dados Heterôgenea:** `list` e `data.frame` (`tibble` no `tidyverse`).

Tipo de dados no R

Número inteiro

```
class(1L)
```

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

Número real

```
class(1.2)
```

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

Número complexo

```
class(1 + 1i)
```

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

Tipo de dados no R

Número lógico ou valor booleano

```
class(TRUE)
```

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

Caracter ou *string*

```
class("Gilberto")
```

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

Estrutura de dados homogênea

Vetor

- Agrupamento de valores de mesmo tipo em um único objeto.
- Criação de vetor:
 - `c(...)`;
 - `vector('<tipo de dados>', <comprimento do vetor>)`;
 - `seq(from = a, to = b, by = c)`;
 - `seq_along(<vetor>)` - vetor de números inteiros com o mesmo trabalho de `<vetor>`;
 - `seq_len(<número inteiro>)` - vetor de números inteiros com o tamanho `<número inteiro>`;
 - `<número inicial>:<número final>` - sequência de números inteiros entre `<número inicial>` e `<número final>`
- Podemos checar o tipo de dados de um vetor com a função `class`.

Vetor de caracteres

```
nomes <- c("Gilberto", "Sassi")  
class(nomes)
```

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

```
nomes
```

```
[1] "Gilberto" "Sassi"
```

```
texto_vazio <- vector("character", 3)  
class(texto_vazio)
```

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

```
texto_vazio
```

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

Estrutura de dados homogênea

Vetor de números reais

```
vetor_real <- c(0.2, 1.35)  
class(vetor_real)
```

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

```
vetor_real
```

```
[1] 0.20 1.35
```

```
vetor_real <- vector("double", 3)  
vetor_real
```

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

```
vetor_real <- seq(from = 1, to = 3.5, by = 0.5)  
vetor_real
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5
```

Estrutura de dados homogênea

Vetor de números inteiros

```
vetor_inteiro <- c(1L, 2L)  
class(vetor_inteiro)
```

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

```
vetor_inteiro
```

```
[1] 1 2
```

```
vetor_inteiro <- vector("integer", 3)  
vetor_inteiro
```

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

```
vetor_inteiro <- 1:4  
vetor_inteiro
```

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

```
vetor_real <- seq_along(nomes)
class(vetor_real)
```

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

```
vetor_real
```

```
[1] 1 2
```

```
vetor_real <- seq_len(5)
class(vetor_real)
```

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

```
vetor_real
```

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

Estrutura de dados homogênea

Vetor lógico

```
vetor_logico <- c(TRUE, FALSE)  
class(vetor_logico)
```

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

```
vetor_logico
```

```
[1] TRUE FALSE
```

```
vetor_logico <- vector("logical", 3)  
vetor_logico
```

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

Estrutura de dados homogênea

Exercício

Crie os seguintes vetores:

- 1 (0,1 0,2 0,3 0,4 0,5)
- 2 (*TRUE TRUE FALSE*)
- 3 ("*Marx*" "*Engels*" "*Lênin*")
- 4 (1 2 3)

Estrutura de dados homogênea

Operações com vetores numéricos (double, integer e complex).

- Operações básicas (operação, subtração, multiplicação e divisão) realizada em cada elemento do vetor.
- *Slicing*: extrair parte de um vetor (não precisa ser vetor numérico).

Slicing

```
vetor <- c("a", "b", "c", "d", "e", "f", "g", "h", "i")  
# selecionado todos os elementos entre o primeiro e o quinta  
vetor[1:5]
```

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

Adição (vetores numéricos)

```
vetor_1 <- 1:5  
vetor_2 <- 6:10  
vetor_1 + vetor_2
```

```
[1] 7 9 11 13 15
```

Estrutura de dados homogênea

Subtração (vetores numéricos)

```
vetor_1 <- 1:5  
vetor_2 <- 6:10  
vetor_2 - vetor_1
```

```
[1] 5 5 5 5 5
```

Multiplicação (vetores numéricos)

```
vetor_1 <- 1:5  
vetor_2 <- 6:10  
vetor_2 * vetor_1
```

```
[1] 6 14 24 36 50
```

Divisão (vetores numéricos)

```
vetor_1 <- 1:5  
vetor_2 <- 6:10  
vetor_2 / vetor_1
```

```
[1] 6.000000 3.500000 2.666667 2.250000 2.000000
```

Estrutura de dados homogênea

Exercício

Realize as seguintes operações envolvendo vetores:

① $(1 \ 2 \ 3) + (0,1 \ 0,05 \ 0,33)$

② $(1 \ 2 \ 3) - (0,1 \ 0,05 \ 0,33)$

③ $(1 \ 2 \ 3) * (0,1 \ 0,05 \ 0,33)$

④ $(1 \ 2 \ 3) / (0,1 \ 0,05 \ 0,33)$

Fatores (factor)

Podemos fixar o conjunto de valores possíveis de uma variável qualitativa (e especificar uma ordem implícita) usando `factor`.

Principais vantagens:

- Evita os erros de digitação.
- Introduz uma ordenação que pode ser útil para construir gráficos e tabelas.
- Necessário para funções de modelagem estatística (que não veremos neste curso).

Vamos usar o pacote `forcats`.

Fatores

Função `fct` do pacote `forcats`: transforma uma variável qualitativa (`chr`) em fator (`fct`).

- `x`: primeiro vetor de texto;
- `levels`: valores possíveis da variável qualitativa, onde a ordem de inputação é a ordem implícita. Se não fornecida, `fct` usará a ordem de aparição.

Fatores

Vamos transformar a variável especie em fator.

```
dados_iris <- read_xlsx("dados/brutos/iris.xlsx")

niveis <- c("setosa", "versicolor", "virginica")
dados_iris <- mutate(
  dados_iris,
  especies = fct(especies, levels = niveis)
)
glimpse(dados_iris)
```

Rows: 150

Columns: 5

```
$ comprimento_sepala <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.
$ largura_sepala      <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.
$ comprimento_petala <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.
$ largura_petala      <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.
$ especies            <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, setosa
```

Fatores

Exercício

Para o conjunto de dados `amostra_enem_salvador.xlsx`, transforme as variáveis `tp_escola` e `tp_cor_raca` em fatores usando a função `fct`.

Estrutura de dados homogênea

Matriz

- Agrupamento de valores de mesmo tipo em um único objeto de dimensão 2.
- Criação de matriz:
 - `matrix(..., nrow = <integer>, ncol = <integer>, byrow = TRUE)` - preenche a matriz a partir das linhas se `byrow = TRUE`;
 - `diag(<vector>)` - diagonal principal igual a `<vetor>` e outros elementos zero;
 - `rbind()` - especificação das linhas da matriz;
 - `cbind()` - especificação das colunas da matriz.

Matriz de caracteres

```
matriz_texto <- rbind(c("a", "b"), c("c", "d"))  
matriz_texto
```

```
      [,1] [,2]  
[1,] "a"  "b"  
[2,] "c"  "d"
```

Matriz de números reais

```
matriz_real <- matrix(seq(from = 0, to = 1.5, by = 0.5),  
                      nrow = 2, byrow = TRUE)  
matriz_real
```

```
      [,1] [,2]  
[1,]    0  0.5  
[2,]    1  1.5
```

Estrutura de dados homogênea

Matriz de inteiros

```
matriz_inteiro <- cbind(c(1L, 2L), c(3L, 4L))  
matriz_inteiro
```

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

Matriz de valores lógicos

```
matriz_logico <- matrix(c(TRUE, F, F, T), nrow = 2)  
matriz_logico
```

```
      [,1] [,2]  
[1,] TRUE FALSE  
[2,] FALSE TRUE
```

Estrutura de dados homogênea

Array

- Agrupamento de valores de mesmo tipo em um único objeto em duas ou mais dimensões.
- Criação de array: `array(..., dim = <vector of integers>)`.

```
dados_matriz_1 <- 10:13
dados_matriz_2  <- 14:17
resultado <- array(c(dados_matriz_1, dados_matriz_2),
                  dim = c(2, 2, 2))
resultado
```

, , 1

	[,1]	[,2]
[1,]	10	12
[2,]	11	13

, , 2

	[,1]	[,2]
[1,]	14	16
[2,]	15	17

Estrutura de dados homogênea

Operações com matrizes numéricas (double, integer e complex).

- Operações básicas (operação, subtração, multiplicação e divisão) realizada em cada elemento das matrizes.
- Outras operações:
 - Multiplicação de matrizes;
 - Inversão de matrizes;
 - Matriz transposta;
 - Determinante;
 - Solução de sistema de equações lineares.

Operações com matrizes

Matrizes

```
matriz_a <- rbind(c(1, 2), c(0, 3))  
matriz_b <- matrix(runif(4), ncol = 2)
```

Soma

```
matriz_soma <- matriz_a + matriz_b  
matriz_soma
```

```
      [,1]      [,2]  
[1,] 1.8423974 2.283715  
[2,] 0.5442284 3.162655
```

Subtração

```
matriz_menos <- matriz_a - matriz_b  
matriz_menos
```

```
      [,1]      [,2]  
[1,] 0.1576026 1.716285  
[2,] -0.5442284 2.837345
```

Operações com matrizes

Produto de Hadamard

- Multiplicação de matrizes, elemento por elemento.
- Para detalhes consulte [produto de Hadamard](#).

```
matriz_hadamard <- matriz_a * matriz_b  
matriz_hadamard
```

```
      [,1]      [,2]  
[1,] 0.8423974 0.5674292  
[2,] 0.0000000 0.4879637
```

Multiplicação de matrizes

```
matriz_multiplicacao <- matriz_a %*% matriz_b  
matriz_multiplicacao
```

```
      [,1]      [,2]  
[1,] 1.930854 0.6090237  
[2,] 1.632685 0.4879637
```

Operações com matrizes

Matriz inversa

```
matriz_inversa <- solve(matriz_a)
matriz_inversa
```

```
      [,1]      [,2]
[1,]    1 -0.6666667
[2,]    0  0.3333333
```

```
matriz_a %*% matriz_inversa
```

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

Matriz transposta

```
matriz_transposta <- t(matriz_a)
matriz_transposta
```

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

Operações com matrizes

Determinante

```
det(matriz_a)
```

```
[1] 3
```

Solução de sistema de equações lineares

```
b <- c(1, 2)  
solve(matriz_a, b)
```

```
[1] -0.3333333  0.6666667
```

Matriz inversa generalizada

G é a matriz inversa generalizada de A se $A \cdot G \cdot A = A$. Para detalhes vide [matriz inversa generalizada](#).

```
library(MASS) # ginv é uma função do pacote MASS  
ginv(matriz_a)
```

```
           [,1]      [,2]  
[1,] 1.000000e+00 -0.6666667  
[2,] -2.775558e-17  0.3333333
```

Operações com matrizes

Outras operações com matrizes.

Operador ou função	Descrição
$A \% \% B$	produto diádico $A \cdot B^T$
<code>crossprod(A, B)</code>	$A \cdot B^T$
<code>crossprod(A)</code>	$A \cdot A^T$
<code>diag(x)</code>	retorna uma matrix diagonal com diagonal igual a x
<code>diag(A)</code>	retorna um vetor com a diagona de A
<code>diag(k)</code>	retorna uma matrix diagona de ordem k

Estrutura de dados homogênea

Exercício

Realize as seguinte operações envolvendo as matrizes:

1 $\begin{pmatrix} 1 & 0 \\ 2 & 0,5 \end{pmatrix} + \begin{pmatrix} 0,1 & 0 \\ 0 & 0,5 \end{pmatrix}$

2 $\begin{pmatrix} 1 & 0 \\ 2 & 0,5 \end{pmatrix} - \begin{pmatrix} 0,1 & 0 \\ 0 & 0,5 \end{pmatrix}$

3 Multiplicação de matriz: $\begin{pmatrix} 1 & 0 \\ 2 & 0,5 \end{pmatrix} \cdot \begin{pmatrix} 0,1 & 0 \\ 0 & 0,5 \end{pmatrix}$

4 Divisão elemento a elemento: $\begin{pmatrix} 1 & 0 \\ 2 & 0,5 \end{pmatrix} / \begin{pmatrix} 0,1 & 0 \\ 0 & 0,5 \end{pmatrix}$

5 Resolva o seguinte sistema de equações:
$$\begin{cases} x + 2y = 21 \\ x - 2y = 1 \end{cases}$$

6 Encontre a matriz inversa de $\begin{pmatrix} 1 & 2 \\ 1 & -2 \end{pmatrix}$.

Estrutura de Dados Heterogênea

Lista

- Agrupamento de valores de tipos diversos e estrutura de dados
- Criação de listas: `list(...)` e `vector("list", <comprimento da lista>)`

```
lista_info <- list(pedido_id = 8001406,  
                 nome = "Fulano",  
                 sobrenome = "de Tal",  
                 cpf = "12345678900",  
                 itens = list(list(descricao = "Ferrari",  
                                   frete = 0,  
                                   valor = 500000),  
                               list(descricao = "Dolly", frete = 1.5,  
                                   valor = 3.90)))  
  
lista_info
```

```
$pedido_id  
[1] 8001406  
  
$nome  
[1] "Fulano"  
  
$sobrenome  
[1] "de Tal"  
  
$cpf  
[1] "12345678900"
```

```
$itens  
$itens[[1]]  
$itens[[1]]$descricao  
[1] "Ferrari"
```

```
$itens[[1]]$frete  
[1] 0
```

```
$itens[[1]]$valor  
[1] 5e+05
```

```
$itens[[2]]  
$itens[[2]]$descricao  
[1] "Dolly"
```

```
$itens[[2]]$frete  
[1] 1.5
```

```
$itens[[2]]$valor  
[1] 3.9
```

Estrutura de dados heterogênea

Exercício

Crie uma lista, chamada `informacoes_pessoais` com os seguintes campos:

- `nome`: seu nome
- `idade`: sua idade
- `informacao_profissional`: uma lista com os seguintes campos:
 - `matricula`: escolaridade
 - `origem`: variável qualitativa com a sua cidade de origem.
- `matriz`: inclua uma matriz de números reais de dimensão 2×2

Operação com listas

- *slicing* - [] - extrai parte da lista (valor retornado é uma lista).
- Acessando *k*-ésimo valor da lista: `lista[[k]]`.
- Acessando um valor da lista pela chave (nome do campo):
`lista$cpf`.
- Concatenação de listas: `c()`.

Slicing

```
lista_info[c(2, 4)]
```

```
$nome
```

```
[1] "Fulano"
```

```
$cpf
```

```
[1] "12345678900"
```

Acessando elemento pela posição

```
lista_info[[2]]
```

```
[1] "Fulano"
```



Acessando elemento pela chave

```
lista_info$nome
```

```
[1] "Fulano"
```

Concatenação de listas

```
lista_1 <- list(1, 2)
lista_2 <- list("Gilberto", "Sassi")
lista_concatenada <- c(lista_1, lista_2)
lista_concatenada
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] 2
```

```
[[3]]
```

```
[1] "Gilberto"
```

```
[[4]]
```

```
[1] "Sassi"
```

Estrutura de dados heterogênea

Exercício

Recupe e imprima as seguintes informações da lista `informacoes_pessoais`:

- os três primeiros campos de `informacoes_pessoais`
- os nomes dos campos de `informacoes_pessoais`
- campo nome de `informacoes_pessoais`
- o terceiro campo de `informacoes_pessoais`

Estrutura de Dados Heterogênea

Tidy data

- Dados em formato de tabela.
 - Cada coluna é uma variável e cada linha é uma observação.
-

`tibble` (data frame)

- Estrutura de dados tabular.
- Assumimos que os dados estão **tidy**.
- Criação de `tibble`: `tibble(...)` e `tribble(...)`.
- `glimpse` mostra as informações do `tibble`.

```
library(tidyverse) # carregando o framework tidyverse
data_frame <- tibble(
  nome = c("Marx", "Engels", "Rosa", "Lênin", "Olga Benário"),
  idade = c(22, 23, 21, 24, 30)
)
glimpse(data_frame)
```

Rows: 5

Columns: 2

\$ nome <chr> "Marx", "Engels", "Rosa", "Lênin", "Olga Benário"

\$ idade <dbl> 22, 23, 21, 24, 30

Valores especiais em R

Valores especiais	Descrição	Função para identificar
NA	Valor faltante.	<code>is.na()</code>
NaN	Resultado do cálculo indefinido.	<code>is.nan()</code>
Inf	Valor que excede o valor máximo que sua máquina aguenta.	<code>is.inf()</code>
NULL	Valor indefinido de expressões e funções (diferente de NaN e NA)	<code>is.null()</code>

Operações básicas em um tibble

Função	Descrição
<code>head()</code>	Mostra as primeiras linhas de um tibble
<code>tail()</code>	Mostra as últimas linhas de um tibble
<code>glimpse()</code>	Impressão de informações básicas dos dados
<code>add_case()</code>	Adiciona uma nova observação
<code>add_row()</code>	Adiciona uma nova observação

```
head(data_frame, n=2)
```

```
# A tibble: 2 x 2
  nome      idade
  <chr>    <dbl>
1 Marx      22
2 Engels    23
```

```
tail(data_frame, n=2)
```

```
# A tibble: 2 x 2
  nome          idade
  <chr>        <dbl>
1 Lênin        24
2 Olga Benário 30
```

Estrutura de dados heterogênea

Exercício

Realize as seguintes operações no *dataset* *iris* (disponível no R):

- imprima um resumo sobre o *dataset* *iris*.
- pegue as 5 primeiras linhas de *iris*.
- pegue as 5 últimas linhas de *iris*.
- crie *na mão* o seguinte conjunto de dados:

nomes	origem
Fidel Castro	Cuba
Ernesto 'Che' Guevara	Cuba
Célia Sánchez	Cuba

Organização é fundamental

Guia de estilo no R

O nome de um objeto precisa ter um *significado*.

O nome deve indicar e deixar claro o que este objeto é ou faz.

- Use a convenção do R:
 - Use apenas letras minúsculas, números e *underscore* (comece sempre com letras minúsculas).
 - Nomes de objetos precisam ser substantivos e precisam descrever o que este objeto é ou faz (seja conciso, direto e significativo).
 - Evite ao máximo os nomes que já são usados (*buit-in*) do R. Por exemplo: `c`.
 - Coloque espaço depois da vírgula.
 - Não coloque espaço antes nem depois de parênteses. Exceção: Coloque um espaço () antes e depois de `if`, `for` ou `while`, e coloque um espaço depois de ().
 - Coloque espaço entre operadores básicos: `+`, `-`, `*`, `==` e outros. Exceção: `^`.

Estrutura de diretórios

Mantenha uma estrutura (organização) consistente de diretórios em seus projetos.

- Sugestão de estrutura:
 - dados: diretório para armazenar seus conjuntos de dados.
 - brutos: dados brutos.
 - processados: dados processados.
 - scripts: código fonte do seu projeto.
 - figuras: figuras criadas no seu projeto.
 - output: outros arquivos que não são figuras.
 - legado: arquivos da versão anterior do projeto.
 - notas: notas de reuniões e afins.
 - relatorio (ou artigos): documento final de seu projeto.
 - documentos: livros, artigos e qualquer coisa que são referências em seu projeto.

Para mais detalhes, consulte esse guia do [curso-r: diretórios e .Rproj](#).

Importação e exportação de dados

Lendo dados no R

Leitura de arquivos no formato `xlsx` ou `xls`

- **Pacote:** `readxl`
- Parâmetros das funções `read_xls` (arquivos `.xls`) e `read_xlsx` (arquivos `.xlsx`):
 - `path`: caminho até o arquivo.
 - `sheet`: especifica a planilha do arquivo que será lida.
 - `range`: especifica uma área de uma planilha para leitura. Por exemplo: `B3:E15`.
 - `col_names`: Argumento lógico com valor padrão igual a `TRUE`. Indica se a primeira linha tem o nome das variáveis.

Para mais detalhes, consulte a documentação: [documentação de `read_xl`](#).

Lendo dados no R

Leitura de arquivos no formato xlsx ou xls

```
library(tidyverse)
library(readxl)
dados_iris <- read_xlsx("dados/brutos/iris.xlsx")
dados_iris <- clean_names(dados_iris)

glimpse(dados_iris)
```

Rows: 150

Columns: 5

```
$ comprimento_sepala <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4,
$ largura_sepala      <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9,
$ comprimento_petala <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4,
$ largura_petala     <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2,
$ especies           <chr> "setosa", "setosa", "setosa", "setosa", "set
```

Lendo dados no R

Exercício

Leia o *dataset* `dados_leitura.xlsx` usando o pacote `readxl`.

Lendo dados no R

As formatações dos arquivos csv

- *csv*: *comma separated values* (valores separados por coluna). O *separador* varia em diferentes sistemas de medidas.
-

- No sistema métrico:
 - As casas decimais são separadas por ,
 - O agrupamento de milhar é marcada por .
 - As colunas dos arquivos de texto são separadas por ;
-

- No sistema imperial inglês (UK e USA):
 - As casas decimais são separadas por .
 - O agrupamento de milhar é marcada por ,
 - As colunas dos arquivos de texto são separadas por ;

Preste atenção em como o seus dados estão armazenados!

Leitura de arquivos no formato csv

- **Pacote:** `readr` do *tidyverse* (instale com o comando `install.packages('readr')`).
- Parâmetros das funções `read_csv` (sistema imperial inglês) e `read_csv2` (sistema métrico):
 - `path`: caminho até o arquivo.

Para mais detalhes, consulte a documentação oficial do *tidyverse*:
[documentação de `read_r`](#).

Lendo dados no R

Leitura de arquivos no formato csv

```
dados_mtcarrros <- read_csv2("dados/brutos/mtcarrros.csv")
dados_mtcarrros <- clean_names(dados_mtcarrros)
glimpse(dados_mtcarrros)
```

Rows: 32

Columns: 11

```
$ milhas_por_galao <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, ~
$ cilindros <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, ~
$ cilindrada <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.~
$ cavalos_forca <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 1~
$ eixo <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, ~
$ peso <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.19~
$ velocidade <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.0~
$ forma <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, ~
$ transmissao <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, ~
$ marchas <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, ~
$ carburadores <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, ~
```

Lendo dados no R

Exercício

Leia o *dataset* `dados_leitura.csv` usando o pacote `readr`.

Lendo dados no R

Leitura de arquivos no formato ods

- **Pacote:** `readODS` (instale com o comando `install.packages('readODS')`).
- Parâmetros das funções `read_ods`:
- `path`: caminho até o arquivo.
 - `sheet`: especifica a planilha do arquivo que será lida.
 - `range`: especifica uma área de uma planilha para leitura. Por exemplo: `B3:E15`.
 - `col_names`: Argumento lógico com valor padrão igual a `TRUE`. Indica se a primeira linha tem o nome das variáveis.

Para mais detalhes, consulte a documentação do *readODS*: [documentação de readODS](#).

Lendo dados no R

Leitura de arquivos no formato ods

```
library(readODS)
dados_dentes <- read_ods("dados/brutos/crescimento_dentes.ods")
dados_dentes <- clean_names(dados_dentes)

glimpse(dados_dentes)
```

Rows: 60

Columns: 3

```
$ comprimento <dbl> 4.2, 11.5, 7.3, 5.8, 6.4, 10.0, 11.2, 11.2,
$ suplemento <chr> "Vitamina C", "Vitamina C", "Vitamina C", "V
$ dose <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
```

Lendo dados no R

Exercício

Leia o *dataset* `dados_leitura.ods` usando o pacote `readODS`.

Exportando dados no R

Salvar no formato .csv (sistema métrico)

`write_csv2` é parte do pacote `readr`.

```
write_csv2(dados_dentes, file = "dados/processados/nome.csv")
```

Salvar no formato .xlsx

`write_xlsx` é parte do pacote `writexl`.

```
write_xlsx(dados_dentes, path = "dados/processados/nome.xlsx")
```

Salvar no formato ods

`write_ods` é parte do pacote `readODS`.

```
write_ods(dados_toothgrowth, path = "dados/processados/nome.ods")
```

Salvando dados no R

Exercício

- 1 Salve o objeto `milhas` do pacote `dados` como `milhas.ods` na pasta `output` do seu projeto.
- 2 Salve o objeto `diamante` do pacote `dados` como `diamante.csv` na pasta `output` do seu projeto.
- 3 Salve o objeto `velho_fiel` do pacote `dados` como `velho_fiel.xlsx` na pasta `output` do seu projeto.

O operador pipe

|>



O valor resultante da expressão do lado esquerdo vira primeiro argumento da função do lado direito.

Principal vantagem: simplifica a leitura e a documentação de funções compostas.

Executar

$f(x, y)$

é exatamente a mesma coisa que executar

$x \mid > f(y)$

```
log(sqrt(sum(x^2)))
```

é exatamente a mesma coisa que executar

```
x^2 |> sum() |> sqrt() |> log()
```

|>

Fazendo um bolo

Exemplo adaptado de [6.1 O operador pipe](#).

Para cozinhar o bolo precisamos usar as seguintes funções:

- `acrescente(lugar, algo)`
- `misture(algo)`
- `asse(algo)`



Fazendo um bolo

- Passo 1:

```
acrescente(  
  "tigela vazia",  
  "farinha"  
)
```

- Passo2:

```
acrescente(  
  acrescete(  
    "tigela vazia",  
    "farinha"  
  ),  
  "ovos"  
)
```

- Passo3:

```
acrescente(  
  acrescete(  
    acrescete(  
      "tigela vazia",  
      "farinha"  
    ),  
    "ovos"  
  ),  
  "leite"  
)
```

- Passo4:

```
acrescente(  
  acrescete(  
    acrescete(  
      acrescete(  
        "tigela vazia",  
        "farinha"  
      ),  
      "ovos"  
    ),  
    "leite"  
  ),  
  "fermento"  
)
```

- Passo 5:

```
misture(  
  acrescente(  
    acrescente(  
      acrescente(  
        "tigela vazia",  
        "farinha"  
      ),  
      "ovos"  
    ),  
    "leite"  
  ),  
  "fermento"  
)  
)
```

- Passo 6:

```
asse(  
  misture(  
    acrescente(  
      acrescente(  
        acrescente(  
          acrescente(  
            "tigela vazia",  
            "farinha"  
          ),  
          "ovos"  
        ),  
        "leite"  
      ),  
      "fermento"  
    )  
  )  
)
```

Usando o operador |>.

```
acrescente("tigela vazia", "farinha") |>  
  acrescente("ovos") |>  
  acrescente("leite") |>  
  acrescente("fermento") |>  
  misture() |>  
  asse()
```

Função

Função

- Evita repetição do mesmo código na análise de dados.
- Se você um mesmo pedaço de código, fica mais simples a atualização deste pedaço de código.
- Evita erros ao fazermos `ctrl+c` e `ctrl+v`.
- Facilita a reutilização de código em outros projetos.

Se você fizer `ctrl+c` e `ctrl+v` duas ou mais vezes, escreva uma função!

Esta seção foi adaptada do livro [R for Data Science](#).

Função motivação

Considere a medida de assimetria descrita por

$$\text{assimetria} = \frac{(\text{quantile}(x, 0.95) - 2 * \text{median}(x) + \text{quantile}(x, 0.05))}{(\text{quantile}(x, 0.95) - \text{quantile}(x, 0.05))}$$

- $\text{assimetria} < 0$ se e somente se x tem assimetria à esquerda ou negativa;
- $\text{assimetria} > 0$ se e somente se x tem assimetria à direita ou positiva;
- $\text{assimetria} == 0$ se e somente se x tem simetria.

Vamos calcular essa medida para cada variável quantitativa do conjunto de dados `iris2.xlsx`.

```

dados_iris <- read_csv2("dados/brutos/iris2.csv")

df_assimetria <- dados_iris |>
  summarise(
    ass_larg_sep = (quantile(larg_sep, 0.95) - 2 * median(larg_sep) +
                    quantile(larg_sep, 0.05)) /
                    (quantile(larg_sep, 0.95) - quantile(larg_sep, 0.05))),
    ass_comp_sep = (quantile(comp_sep, 0.95) - 2 * median(comp_sep) +
                    quantile(comp_sep, 0.05)) /
                    (quantile(comp_sep, 0.95) - quantile(comp_sep, 0.05))),
    ass_larg_pet = (quantile(larg_pet, 0.95) - 2 * median(larg_pet) +
                    quantile(larg_pet, 0.05)) /
                    (quantile(larg_pet, 0.95) - quantile(larg_pet, 0.05))),
    ass_comp_pet = (quantile(comp_pet, 0.95) - 2 * median(comp_pet) +
                    quantile(comp_pet, 0.05)) /
                    (quantile(comp_pet, 0.95) - quantile(comp_pet, 0.05))
  )

```

```
df_assimetria
```

```
# A tibble: 1 x 4
```

```
  ass_larg_sep ass_comp_sep ass_larg_pet ass_comp_pet  
    <dbl>         <dbl>         <dbl>         <dbl>  
1    0.0960       0.0997       -0.271       -0.0476
```

Largura de pétala tem assimetria à esquerda ou negativa.

Repetimos quatro vezes o cálculo da medida de assimetria.

precisamos criar uma função!

Função

Uma função é composta por três partes:

- nome: nome para o pedaço de código;
- argumentos: informações necessárias para execução do pedaço de código;
- corpo: pedaço de código.

Em corpo, geralmente incluímos a função `return(<objeto>)`.

Se a função não tiver `return(<objeto>)`, valor da última linha será retornado.

Escrevemos uma função da seguinte forma:

```
nome <- function(arg1, arg2, arg3 = <valor padrão>, arg4 = <valor padrão>) {  
  corpo  
}
```

Função

Vamos reescrever o código do cálculo de assimetria.

```
dados_iris <- read_csv2("dados/brutos/iris2.csv")

ass <- function(x) {
  q_lower <- quantile(x, 0.05)
  q_upper <- quantile(x, 0.95)
  m <- median(x)

  (q_upper - 2 * m + q_lower) / (q_upper - q_lower)
}

df_assimetria <- dados_iris |>
  summarise(
    ass_larg_sep = ass(larg_sep), ass_comp_sep = ass(comp_sep),
    ass_larg_pet = ass(larg_pet), ass_comp_pet = ass(comp_pet)
  )
df_assimetria
```

```
# A tibble: 1 x 4
```

	ass_larg_sep	ass_comp_sep	ass_larg_pet	ass_comp_pet
	<dbl>	<dbl>	<dbl>	<dbl>
1	0.0960	0.0997	-0.271	-0.0476

Função

Se mudássemos o coeficiente de assimetria para:

$$\text{assimetria} = \frac{(\text{quantile}(x, 0.99) - 2 * \text{median}(x) + \text{quantile}(x, 0.01))}{(\text{quantile}(x, 0.99) - \text{quantile}(x, 0.01))}$$

```
dados_iris <- read_csv2("dados/brutos/iris2.csv")

ass <- function(x) {
  q_lower <- quantile(x, 0.01)
  q_upper <- quantile(x, 0.99)
  m <- median(x)

  (q_upper - 2 * m + q_lower) / (q_upper - q_lower)
}

df_assimetria <- dados_iris |>
  summarise(
    ass_larg_sep = ass(larg_sep), ass_comp_sep = ass(comp_sep),
    ass_larg_pet = ass(larg_pet), ass_comp_pet = ass(comp_pet)
  )
df_assimetria
```

```
# A tibble: 1 x 4
  ass_larg_sep ass_comp_sep ass_larg_pet ass_comp_pet
  <dbl>         <dbl>         <dbl>         <dbl>
1      0.152      0.180         -0.153        -3.47e-17
```

Função Exercício

Para amostra_enem_salvador.xlsx, use o seguinte código

```
((x - min(x)) * 10) / (max(x) - min(x))
```

para repadronizar (de 0 a 10) as seguintes variáveis: nu_nota_cn, nu_nota_ch, nu_nota_lc, nu_nota_mt, nu_nota_comp1, nu_nota_comp2, nu_nota_comp3, nu_nota_comp4, nu_nota_comp5, e nu_nota_redacao.

Controle de fluxo

Controle de fluxo

Controle de fluxo para vetores, listas e matrizes.

- `if else;`
- `for;`
- `while;`
- `case_when;`
- `sapply, apply` e `vapply;`
- família de funções `map_*`, onde `*` é `chr`, `dbl`, `int`, `df` e `lgl`.

Controle de fluxo

if else e for

if else

```
if (<expressao>) {  
    <bloco de código>  
} else if (<expressao>) {  
    <bloco de código>  
} else if (<expressao>) {  
    <bloco de código>  
} else {  
    <bloco de código>  
}
```

for

```
for (k in <vetor or lista>) {  
    <bloco de código>  
}
```



Controle de fluxo if else e for

Considere a variável aleatória discreta com função de probabilidade dada por:

x	$f(x)$
1	0,4
3	0,25
10	0,35

Gere uma amostra aleatória com 1000 observações de X .

```
amostra_x <- vector("integer", 1000)
for (k in 1:1000) {
  valor_aleatorio <- runif(1)
  if (valor_aleatorio <= 0.4) {
    amostra_x[k] <- 1
  } else if (0.4 < valor_aleatorio & valor_aleatorio <= 0.65) {
    amostra_x[k] <- 3
  } else {
    amostra_x[k] <- 10
  }
}

table(amostra_x) / 1000
```

```
amostra_x
      1      3     10
0.420 0.259 0.321
```

Controle de fluxo

if else e while

while

```
while (<expressão lógica>) {  
    <bloco de código>  
}
```

-
- Para sair de um laço for, usamos break.
 - Para o restante do bloco de código e ir para o próximo passo laço, usamos continue.

Controle de fluxo if else e while

Considere $X \sim N(2, 4)$ truncado com valor 0 e 4: $X \mid 0 \leq X \leq 4$.

Gere uma amostra aleatória com 1000 observações de X .

```
amostra_x <- vector("double", 1000)
for (k in seq_len(1000)) {
  while (TRUE) {
    valor_aleatorio <- rnorm(1, 2, 2)
    if (between(valor_aleatorio, 0, 4)) {
      amostra_x[k] <- valor_aleatorio
      break
    }
  }
}
summary(amostra_x)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.000762	1.122630	1.943720	1.977426	2.882528	3.996917

Controle de fluxo

sapply, lapply e vapply

Versão mais rápida e mais simples de ler do seguinte código:

```
resultado <- vector("double", length(sequencia))
for (k in sequencia) {
  resultado[k] <- funcao(k)
}
```

sapply

- sapply cria um vetor se funcao retorna um escalar;
- sapply cria uma matriz se funcao retorna um vetor.

```
resultado <- sapply(sequencia, funcao)
```

lapply

lapply cria uma lista.

```
resultado <- lapply(sequencia, funcao)
```

vapply

Precisamos apresentar um modelo de valor retornado de funcao. Este modelo pode um escalar, um vetor ou uma matriz (neste caso resultado será um array).

```
resultado <- vapply(sequencia, funcao, modelo)
```

Função anônima

É comum criamos uma função *simples* para usar apenas uma vez com `sapply`, `lapply` e `vapply`.

Neste caso podemos criar uma função anônima:

```
\(x) <código>
```

ou

```
\(x) {  
  <bloco de código>  
}
```

Controle de fluxo

sapply, lapply e vapply

Exemplo

Vamos padronizar as variáveis do conjunto de dados `mtcarros.xlsx`.

```
dados_carros <- read_csv2("dados/brutos/mtcarros.csv")
m_carros <- dados_carros |>
  select(where(is.numeric)) |>
  sapply(\(x) (x - mean(x)) / sd(x))
class(m_carros)
```

```
[1] "matrix" "array"
```

```
dim(m_carros)
```

```
[1] 32 11
```

Exemplo

Suponha que queremos calcular a média, a mediana e o desvio padrão para as variáveis `milhas_por_galao`, `cilindros` e `velocidade` do conjunto de dados `mtcarros.xlsx`

```
sumario <- dados_carros |>
  select(milhas_por_galao, cilindros, velocidade) |>
  vapply(
    \(x) c("Média" = mean(x), "Mediana" = median(x), "DP" = sd(x)),
    c(0,0,0)
  )
class(sumario)
```

```
[1] "matrix" "array"
```

```
sumario
```

	<code>milhas_por_galao</code>	<code>cilindros</code>	<code>velocidade</code>
Média	20.090625	6.187500	17.848750
Mediana	19.200000	6.000000	17.710000
DP	6.026948	1.785922	1.786943

Controle de fluxo

apply, lapply e vapply

Exemplo

```
medias_carros <- dados_carros |>  
  select(milhas_por_galao, cilindros, velocidade) |>  
  lapply(\(coluna) mean(coluna))  
class(medias_carros)
```

```
[1] "list"
```

```
medias_carros
```

```
$milhas_por_galao
```

```
[1] 20.09062
```

```
$cilindros
```

```
[1] 6.1875
```

```
$velocidade
```

```
[1] 17.84875
```

Controle de fluxo

sapply, lapply e vapply

Exercício

- Calcule a média, a mediana, e o desvio padrão para as notas do conjunto de dados `amostra_enem_salvador.xlsx` (variáveis que começam com `nu_nota_`). Use `sapply`.
- Crie um array com a média, a mediana, mínimo e o máximo para largura e comprimento das pétalas e sépalas do conjunto de dados `iris.xlsx`. Use `vapply`.
- Crie uma lista com a moda das variáveis `tpsexo`, `tpcor_raca` e `tpescola` do conjunto de dados `amostra_enem_salvador.xlsx`. Use `lapply`.

Controle de fluxo

map_*

Equivalente das funções `sapply`, `lapply` e `vapply` do *framework* tidyverse.

Principais melhorias: consistências na família `map_` e documentação (explícita o valor retornado).

- `map(sequencia, funcao)`: equivalente a `lapply`;
- `map_chr(sequencia, funcao)`: produz um vetor de *strings*:
 - equivalente a `vapply(sequencia, funcao, 'texto')`;
 - `funcao` retorna um escalar do tipo `character`;
- `map_dbl(sequencia, funcao)`: produz um vetor de números reais:
 - equivalente a `vapply(sequencia, funcao, 1.11)`;
 - `funcao` retorna um escalar do tipo `numeric`;

- `map_int(sequencia, funcao)`: produz um vetor de números inteiros:
 - equivalente a `vapply(sequencia, funcao, 1.11)`;
 - `funcao` retorna um escalar do tipo `integer`;
- `map_lgl(sequencia, funcao)`: produz um vetor de valores lógicos:
 - equivalente a `vapply(sequencia, funcao, TRUE)`;
 - `funcao` retorna um escalar do tipo `logical`;
- `map_df(tibble, funcao)`: produz um `tibble`:
 - `funcao` retorna um vetor com o tamanho `nrow(tibble)`.

Controle de fluxo

map_*

Exemplo

Vamos calcular o quantil de ordem 60% para as variáveis do conjunto de dados `mtcarros.xlsx`

```
dados_carros <- read_excel("dados/brutos/mtcarros.xlsx")
medias_carros <- dados_carros |>
  select(where(is.numeric)) |>
  map_dbl(\(variavel) quantile(variavel, 0.60))
class(medias_carros)
```

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

```
medias_carros
```

<code>milhas_por_galao</code>	<code>cilindros</code>	<code>cilindrada</code>	<code>cavalos_forca</code>
21.0	8.0	275.0	165.0
<code>forma</code>	<code>transmissao</code>	<code>marchas</code>	<code>carburadores</code>
1.0	0.6	4.0	3.0

Controle de fluxo

map_*

Exemplo

Padronize as variáveis quantitativas do conjunto de dados `mtcarros.xlsx`.

```
dados_carros <- read_excel("dados/brutos/mtcarros.xlsx")
df_carros_padronizada <- dados_carros |>
  select(where(is.numeric)) |>
  map_df(\(variavel) (variavel - mean(variavel)) / sd(variavel))
glimpse(df_carros_padronizada)
```

Rows: 32

Columns: 8

```
$ milhas_por_galao <dbl> 0.2188603, 0.2188603, 0.3856111, 0.2188
$ cilindros <dbl> -0.1049878, -0.1049878, -1.2248578, -0.
$ cilindrada <dbl> -0.56836598, -0.56836598, -0.98758628,
$ cavalos_forca <dbl> -0.53509284, -0.53509284, -0.78304046,
$ forma <dbl> -0.8680278, -0.8680278, 1.1160357, 1.11
$ transmissao <dbl> 1.1899014, 1.1899014, 1.1899014, -0.814
$ marchas <dbl> 0.4235542, 0.4235542, 0.4235542, -0.931
$ carburadores <dbl> 0.7352031, 0.7352031, -1.1221521, -1.12
```

Transformações nos dados

Transformações nos dados

Objetivo: Transformações em um objeto tibble.

Transformações que veremos neste curso:

- 1 seleção de observações (linhas);
- 2 seleção de variáveis (colunas);
- 3 criação de novas colunas;
- 4 ordenação de linhas;
- 5 transposição de linhas para colunas;
- 6 transposição de colunas para linhas.

Vamos usar o conjunto de dados `voos.xlsx` que todos os voos que decolaram no horários na cidade de Nova Iorque nos EUA.

No *framework* tidyverse (especialmente no pacote dplyr), temos que

- 1 O primeiro argumento é sempre um data frame (objeto da classe tibble);
- 2 Os outros argumentos tipicamente descrevem quais colunas para realizar as operações, e essas colunas são indicadas pelo nome sem aspas;
- 3 O resultado sempre é sempre um data frame (objeto da classe tibble);
- 4 Chama-se de *verbo* funções que agem sobre base de dados;
- 5 Para resolver problemas reais, combinamos vários *verbos* usando operador |>.
- 6 Os *verbos* são classificados em quatro categorias: **linhas**, **colunas**, **grupos** e **tabelas**.

Transformações nos dados

linhas

Principais *verbos*:

- `filter()`: seleciona linhas baseadas de condições envolvendo colunas;
- `arrange()`: ordena linhas de acordo com alguma(s) coluna(s);
- `distinct()`: remove linhas com repetições segundo algumas(s) coluna(s).

Transformações nos dados

filter()

Usamos operadores lógicos para selecionar linhas segundo alguma condições envolvendo colunas:

- desigualdades: >, <, <=, >=, ==, !=;
- combinações de expressões lógicas: & (e), | (ou);
- checka se valor está em um vetor: %in%. Exemplo, 'Gilberto' %in% equipe retorna TRUE;

Vamos selecionar os voos que saíram de Nova York no feriado de ano novo.

```
df_voos <- read_excel('dados/brutos/voos.xlsx')
```

```
df_voos_ano_novo <- df_voos |>  
  filter(mes == 1 & dia == 1)  
glimpse(df_voos_ano_novo)
```

Rows: 842

Columns: 19

\$ ano	<dbl>	2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013
\$ mes	<dbl>	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
\$ dia	<dbl>	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
\$ horario_saida	<dbl>	517, 533, 542, 544, 554, 554, 555, 557, 557, 5
\$ saida_programada	<dbl>	515, 529, 540, 545, 600, 558, 600, 600, 600, 6
\$ atraso_saida	<dbl>	2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -
\$ horario_chegada	<dbl>	830, 850, 923, 1004, 812, 740, 913, 709, 838,
\$ chegada_prevista	<dbl>	819, 830, 850, 1022, 837, 728, 854, 723, 846,
\$ atraso_chegada	<dbl>	11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2,
\$ companhia_aerea	<chr>	"UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV"
\$ voo	<dbl>	1545, 1714, 1141, 725, 461, 1696, 507, 5708, 7
\$ cauda	<chr>	"N14228", "N24211", "N619AA", "N804JB", "N668D
\$ origem	<chr>	"EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR"
\$ destino	<chr>	"IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL"
\$ tempo_voo	<dbl>	227, 227, 160, 183, 116, 150, 158, 53, 140, 13
\$ distancia	<dbl>	1400, 1416, 1089, 1576, 762, 719, 1065, 229, 9
\$ hora	<dbl>	5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5
\$ minuto	<dbl>	15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0,
\$ data_hora	<dtm>	2013-01-01 10:00:00, 2013-01-01 10:00:00, 201

Transformações nos dados

arrange()

- Se mais de uma coluna é usada, as colunas adicionais são usadas para desempatar a ordenação.
- Para ordenar em ordem decrescente (maior ao menor), use a função desc().

```
df_voos_ordenado <- df_voos |>
  arrange(mes, desc(dia), desc(horario_saida))
head(df_voos_ordenado, 5)
```

```

# A tibble: 5 x 19
  ano   mes   dia horario_saida saida_programada atraso_saida
  <dbl> <dbl> <dbl>         <dbl>             <dbl>         <dbl>
1  2013     1    31           2354             2055           179
2  2013     1    31           2330             2159            91
3  2013     1    31           2324             1905           259
4  2013     1    31           2312             2159            73
5  2013     1    31           2310             2105           125
# i 12 more variables: chegada_prevista <dbl>, atraso_chegada <dbl>,
#   companhia_aerea <chr>, voo <dbl>, cauda <chr>, origem <chr>,
#   tempo_voo <dbl>, distancia <dbl>, hora <dbl>, minuto <dbl>,
#   data_hora <dtm>

```

Transformações nos dados

distinct()

- Remove linhas repetidas.
- A primeira ocorrência das linhas repetidas será mantida, e todas as outras ocorrências serão descartadas.

Exemplo:

```
coordenadores <- tribble(  
  ~nome, ~email, ~funcao,  
  "Gilberto", "gilberto.sassi@ufba.br", "coordenador",  
  "Gilberto", "gilberto.sassi@ufba.br", "professor",  
  "Gilberto", "gilberto.sassi@ufba.br", "professor",  
  "Carolina", "carolina.paraiba@ufba.br", "coordenadora",  
  "Carolina", "carolina.paraiba@ufba.br", "professora",  
  "Carolina", "carolina.paraiba@ufba.br", "professora"  
)
```



Tiramos a repetição das linhas.

```
distinct(coordenadores)
```

```
# A tibble: 4 x 3
```

	nome	email	funcao
	<chr>	<chr>	<chr>
1	Gilberto	gilberto.sassi@ufba.br	coordenador
2	Gilberto	gilberto.sassi@ufba.br	professor
3	Carolina	carolina.paraiba@ufba.br	coordenadora
4	Carolina	carolina.paraiba@ufba.br	professora

Tiramos a repetição por e-mail.

```
distinct(coordenadores, email, .keep_all = TRUE)
```

```
# A tibble: 2 x 3
```

	nome	email	funcao
	<chr>	<chr>	<chr>
1	Gilberto	gilberto.sassi@ufba.br	coordenador
2	Carolina	carolina.paraiba@ufba.br	coordenadora

Transformações nos dados

Exercício

Para o conjunto de dados `amostra_enem_salvador.xlsx`, realiza as seguintes operações:

- selecione todas/os candidatas/os que são pardas, pretas e indígenas (`tp_cor_raca`);
- ordene as/os candidatas/os pela nota em matemática (`nu_not_a_mt`).

Transformações nos dados

colunas

Principais *verbos*:

- `mutate()`: adiciona uma nova coluna ao `data.frame`;
- `select()`: selecionar colunas de um `data.frame`;
- `rename()`: atualize os nomes das colunas de um `data.frame`;
- `relocate()`: mude a ordem das colunas.

Transformações nos dados

mutate()

`mutate` adiciona a coluna a direita da última coluna. Podemos usar os seguintes argumentos nomeados:

- `.before`: número inteiro positivo (indicado a posição da coluna) ou o nome da coluna. A nova coluna será incluído a esquerda.
- `.after`: número inteiro positivo (indicado a posição da coluna) ou o nome da coluna. A nova coluna será incluído a direita.

```
df_voos <- df_voos |>  
  mutate(velocidade = distancia / tempo_voo, .before = "ano")
```

```
head(df_voos, n=5)
```

```
# A tibble: 5 x 20
```

```
  velocidade  ano  mes  dia horario_saida saida_programada at
  <dbl> <dbl> <dbl> <dbl>          <dbl>          <dbl>
1     6.17  2013     1     1             517             515
2     6.24  2013     1     1             533             529
3     6.81  2013     1     1             542             540
4     8.61  2013     1     1             544             545
5     6.57  2013     1     1             554             600
# i 13 more variables: horario_chegada <dbl>, chegada_prevista <dbl>,
# atraso_chegada <dbl>, companhia_aerea <chr>, voo <dbl>, caud
# origem <chr>, destino <chr>, tempo_voo <dbl>, distancia <dbl>
# minuto <dbl>, data_hora <dtm>
```

Transformações nos dados

select()

Seleção de colunas pelo nome e/ou por *atalhos*.

Seleção por nomes das colunas.

```
df_select <- df_voos |>
  select(ano, mes, dia)
glimpse(df_select)
```

Rows: 336,776

Columns: 3

\$ ano <dbl> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013

\$ mes <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

\$ dia <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

Seleção de todas as colunas segundo uma função.

```
df_select <- df_voos |>
  select(where(is.character))
glimpse(df_select)
```

Rows: 336,776

Columns: 4

```
$ companhia_aerea <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6"
$ cauda           <chr> "N14228", "N24211", "N619AA", "N804JB",
$ origem         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR"
$ destino        <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD"
```

Seleção de todas as colunas que começam com alguma *string*.

```
df_select <- dados_iris |>
  select(starts_with("largura"))
glimpse(df_select)
```

Rows: 150

Columns: 0

Seleção de todas as colunas que terminam com alguma *string*.

```
df_select <- df_voos |>
  select(ends_with("_chegada"))
glimpse(df_select)
```

Rows: 336,776

Columns: 2

```
$ horario_chegada <dbl> 830, 850, 923, 1004, 812, 740, 913, 709,
$ atraso_chegada <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8
```

Seleção de todas as colunas que contém alguma *string*.

```
df_select <- df_voos |>
  select(contains("chegada"))
glimpse(df_select)
```

Rows: 336,776

Columns: 3

\$ horario_chegada <dbl> 830, 850, 923, 1004, 812, 740, 913, 709

\$ chegada_prevista <dbl> 819, 830, 850, 1022, 837, 728, 854, 723

\$ atraso_chegada <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8,

Seleção de todas as colunas em sequencia.

```
dados_billboard <- read_excel("dados/brutos/billboard_eua_2000.x  
df_select <- dados_billboard |>  
  select(num_range("semana_", 1:6))  
glimpse(df_select)
```

Rows: 317

Columns: 6

```
$ semana_1 <dbl> 87, 91, 81, 76, 57, 51, 97, 84, 59, 76, 84, 57,  
$ semana_2 <dbl> 82, 87, 70, 76, 34, 39, 97, 62, 53, 76, 84, 47,  
$ semana_3 <dbl> 72, 92, 68, 72, 25, 34, 96, 51, 38, 74, 75, 45,  
$ semana_4 <dbl> 77, NA, 67, 69, 17, 26, 95, 41, 28, 69, 73, 29,  
$ semana_5 <dbl> 87, NA, 66, 67, 17, 26, 100, 38, 21, 68, 73, 23  
$ semana_6 <dbl> 94, NA, 57, 65, 31, 19, NA, 35, 18, 67, 69, 18,
```

Transformações nos dados

rename()

Renome as colunas de um `data.frame`.

O padrão de renomeação é: `<nome novo> = <nome velho>`.

```
df_iris <- dados_iris |>
  rename(
    "sepala_comprimento" = "comprimento_sepala",
    "sepala_largura" = "largura_sepala",
    "petala_comprimento" = "comprimento_petala",
    "petala_largura" = "largura_petala"
  )
glimpse(df_iris)
```

Rows: 150

Columns: 5

```
$ sepala_comprimento <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.  
$ sepala_largura      <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.  
$ petala_comprimento <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.  
$ petala_largura     <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.  
$ especies           <chr> "setosa", "setosa", "setosa", "setosa"
```

Transformações nos dados

relocate()

Move as colunas pelo `data.frame`. Por padrão, as colunas listas vão para a extrema esquerda do `data.frame`, mas pode usar os argumentos `.before` e `.after`:

- `.before`: número inteiro positivo (indicado a posição da coluna) ou o nome da coluna. As colunas serão incluídas a esquerda.
- `.after`: número inteiro positivo (indicado a posição da coluna) ou o nome da coluna. As colunas serão incluídas a direita.

```
dados_mtcarrros <- dados_mtcarrros |>
  relocate(marchas, carburadores, .before = "cilindros")
glimpse(dados_mtcarrros)
```

Rows: 32

Columns: 11

```
$ milhas_por_galao <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.
$ marchas          <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3,
$ carburadores     <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3,
$ cilindros       <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8,
$ cilindrada      <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.
$ cavalos_forca   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 9
$ eixo            <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.2
$ peso            <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.46
$ velocidade     <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.2
$ forma          <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
$ transmissao    <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Transformações nos dados

Exercício

Para o conjunto de dados `amostra_enem_salvador.xlsx`, realize as seguintes operações:

- transforme a variável `tp_cor_raca` em fator, e agrupe os valores parda e preta em negra;
- selecione as colunas: `tp_cor_raca`, `tp_escola` e `nu_nota_mt`;
- renomeie as colunas da seguinte forma:
 - `tp_cor_raca` por `raca`;
 - `tp_escola` por `tipo_escola`;
 - `nu_nota_mt` por `nota_matematica`;
- coloque `tipo_escola` como a primeira coluna.

Transformações nos dados grupos

- `group_by`: transforma um tibble em um tibble agrupado por uma ou mais colunas. Todas as operações serão realizadas por este agrupamento;
- `slice_*` (* pode ser `min`, `max`, `sample`, `tail`, e `head`): extrai uma ou algumas linhas de cada grupo;
- `summarize`: calcula medidas de resumo por grupos;
- `ungroup`: remove o agrupamento.

Transformações nos dados

group_by()

Prepara um conjunto de dados para realizar operações em grupos (usando a função summarise).

```
dados_mtcarros_grupos <- group_by(dados_mtcarros, marchas, carburadores)
glimpse(dados_mtcarros_grupos)
```

Rows: 32

Columns: 11

Groups: marchas, carburadores [11]

```
$ milhas_por_galao <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, ~
$ marchas <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, ~
$ carburadores <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, ~
$ cilindros <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, ~
$ cilindrada <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.~
$ cavalos_forca <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 1~
$ eixo <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, ~
$ peso <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.19~
$ velocidade <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.0~
$ forma <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, ~
$ transmissao <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
```

Transformações nos dados

`slice_*`()

Pega parte do `data.frame` respeitando o agrupamento estabelecido por `group_by`.

- `df |> slice_head(n = 1)` pega as `n` primeiras linhas de cada grupo.
- `df |> slice_tail(n = 1)` pega as `n` últimas linhas de cada grupo.
- `df |> slice_min(x, n = 1)` pega as linhas com os menores valores de `x` em cada grupo.
- `df |> slice_max(x, n = 1)` pega as linhas com os maiores valores de `x` em cada grupo.
- `df |> slice_sample(n = 1)` pega aleatoriamente `n` linhas de cada grupo.

```
dados_iris |>  
  group_by(especies) |>  
  slice_sample(n = 2)
```

```
# A tibble: 6 x 5
```

```
# Groups:   especies [3]
```

	comprimento_sepala	largura_sepala	comprimento_petala	largura_p
	<dbl>	<dbl>	<dbl>	
1	5.4	3.7	1.5	
2	5.1	3.4	1.5	
3	5	2	3.5	
4	6.3	2.3	4.4	
5	6.2	3.4	5.4	
6	7.2	3	5.8	

Transformações nos dados

summarise()

Calcula medidas de resumo por cada grupo.

Caso 1: apenas uma variável

Retorna um `data.frame` sem agrupamento.

```
df_voos |>
  group_by(mes) |>
  summarise(freq = n())
```

```
# A tibble: 12 x 2
```

```
  mes  freq
```

```
<dbl> <int>
```

```
1     1 27004
2     2 24951
3     3 28834
4     4 28330
5     5 28796
6     6 28243
7     7 29425
8     8 29327
9     9 27574
10    10 28889
11    11 27268
12    12 28135
```

Caso 2: duas ou mais variáveis

Neste caso, `summarise` retorna um `data.frame` com agrupamentos que são controlados pelo argumento `.groups`:

- se `.groups = "drop_last"`: o agrupamento da última variável é retirado. Este é o comportamento padrão (para compatibilidade com versões antigas do tidyverse).
- se `.groups = "keep"`: os agrupamentos de todas as variáveis são mantidas.
- se `.groups = "drop"`: os agrupamentos de todas as variáveis são retiradas. (Geralmente, usamos este).
- se `.groups = "rowwise"`: os agrupamentos são mantidos para que linha tenha o seu próprio grupo.

Usando `.groups = "drop_last"`.

```
dados_mtcarrros |>
  group_by(marchas, forma) |>
  summarise(peso_medio = mean(peso, na.rm = TRUE))
```

```
# A tibble: 6 x 3
# Groups:   marchas [3]
  marchas forma peso_medio
  <dbl> <dbl> <dbl>
1     3     0     4.10
2     3     1     3.05
3     4     0     2.75
4     4     1     2.59
5     5     0     2.91
6     5     1     1.51
```

Usando `.groups = "keep"`.

```
dados_mtcarrros |>
  group_by(marchas, forma) |>
  summarise(peso_medio = mean(peso, na.rm = TRUE),
            .groups = "keep")
```

A tibble: 6 x 3

Groups: marchas, forma [6]

	marchas	forma	peso_medio
	<dbl>	<dbl>	<dbl>
1	3	0	4.10
2	3	1	3.05
3	4	0	2.75
4	4	1	2.59
5	5	0	2.91
6	5	1	1.51

Usando `.groups = "drop"`.

```
dados_mtcarrros |>
  group_by(marchas, forma) |>
  summarise(peso_medio = mean(peso, na.rm = TRUE),
            .groups = "drop")
```

A tibble: 6 x 3

	marchas	forma	peso_medio
	<dbl>	<dbl>	<dbl>
1	3	0	4.10
2	3	1	3.05
3	4	0	2.75
4	4	1	2.59
5	5	0	2.91
6	5	1	1.51

Transformações nos dados

Exercício

Para o conjunto de dados `amostra_enem_salvador.xlsx`, realize as seguintes operações:

- selecione as colunas `tp_cor_raca`, `tp_escola` e `nu_nota_mt`;
- agrupe o conjunto de dados por `tp_cor_raca` e `tp_escola`;
- pegue dois elementos de cada grupo;
- conte o número de pessoas em cada grupo e calcule a mediana.

Transformações nos dados tabelas

União de **conjunto de dados**.

- chaves (primárias e estrangeiras);
- adicionando novas colunas de outros conjuntos de dados:
`left_join()`, `inner_join()`, `right_join()`, `full_join()`,
`semi_join()`, e `anti_join()`;
- filtragem: `semi_join()` e `anti_join()`.

Transformações nos dados

chaves

chave primária : variável ou conjunto de variáveis (colunas) determinam unicamente uma observação (uma linha);

chave estrangeira : variável ou conjunto de variáveis (colunas) que corresponde a uma chave primária de outro conjunto de dados;

Geralmente **chaves primárias** e **chaves estrangeiras** têm o mesmo nome (para facilitar nossa vida);

Verifique se cada valor da chave identifica no máximo uma observação.

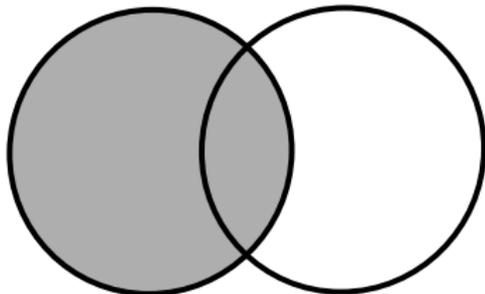
Transformações nos dados

*_join()

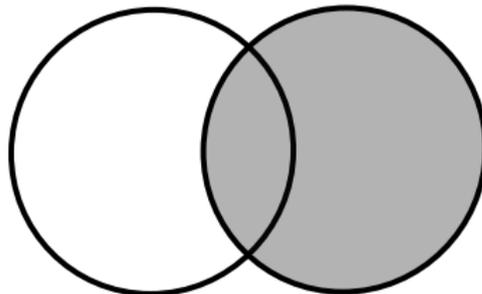
- `left_join(x, y, join_by(key_x == key_y))`: mantêm todas as linhas de x (com as colunas de y);
- `inner_join(x, y, join_by(key_x == key_y))`: mantêm todas as linhas que estão em x e y (com as linhas de y);
- `right_join(x, y, join_by(key_x == key_y))`: mantêm todas as linhas de y (com as colunas de x);
- `full_join(x, y, join_by(key_x == key_y))`: mantêm todas as linhas de x e y (com as colunas de x e y);
- `semi_join(x, y, join_by(key_x == key_y))`: mantêm as linhas de x que estão em y (mantem as apenas as colunas de x);
- `anti_join(x, y, join_by(key_x == key_y))`: mantêm as linhas de x que **não** estão em y (mantem as apenas as colunas de x).

Figura 1: Ilustração de `left_join`, `right_join`, `inner_join` e `full_join`.

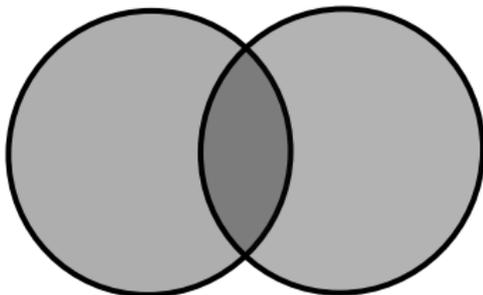
`left_join(x, y)`



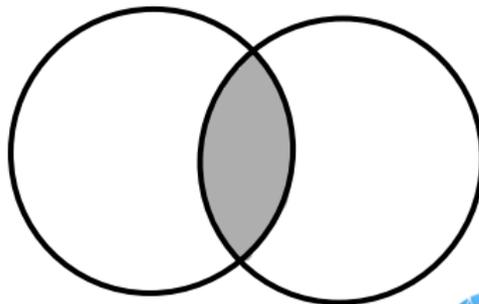
`right_join(x, y)`



`full_join(x, y)`



`inner_join(x, y)`



Transformações nos dados

*_join()

Em voos.xlsx temos o código da companhia aérea, mas não tem o nome. O nome da companhia aérea está na coluna nome em companhia_aerea.xlsx.

```
df_companhias_aereas <- read_excel("dados/brutos/companhias_aereas.xlsx")
df_voos2 <- df_voos |>
  left_join(df_companhias_aereas, join_by(companhia_aerea == companhia_aerea))
glimpse(df_voos2)
```

Rows: 336,776

Columns: 21

```
$ velocidade      <dbl> 6.167401, 6.237885, 6.806250, 8.612022, 6.568966, 4.7~
$ ano             <dbl> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013,~
$ mes            <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
$ dia            <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
$ horario_saida  <dbl> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558~
$ saida_programada <dbl> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600~
$ atraso_saida   <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, ~
$ horario_chegada <dbl> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 84~
$ chegada_prevista <dbl> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 85~
$ atraso_chegada <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, ~
$ companhia_aerea <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6",~
$ voo            <dbl> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301,~
$ cauda          <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N3~
$ origem         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA~
$ destino        <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD~
$ tempo_voo      <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149,~
$ distancia      <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733~
$ hora           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6,~
$ minuto         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 59,~
$ data_hora      <dtm> 2013-01-01 10:00:00, 2013-01-01 10:00:00, 2013-01-01~
$ nome           <chr> "United Air Lines Inc.", "United Air Lines Inc.", "Am~
```

Transformações nos dados

Exercício

Adicione as condições climáticas (`clima.csv`) a cada voo no conjunto de dados `voos.xlsx`. Use como chave `data_hora` e `origem`.

(Dica: use `left_join`).

Depois selecione os voos de algumas companhias aéreas que estão em `companhias_especiais.csv`. Use como chave `companhia_aerea`.

(Dica: use `semi_join`).