

Aprendendo R

Magno TF Severino

2023-12-23

Contents

Introdução	5
1 Básicos da linguagem R	7
1.1 Usando o R como uma calculadora	7
1.2 Atribuindo variáveis	9
1.3 Números especiais	9
1.4 Vetores, Matrizes e Dataframes	10
2 Controle de fluxo e repetição	17
2.1 Funções	20
2.2 Lista de exercicios	21
3 Importação e manipulação de bases de dados	23
3.1 Análise descritiva	28
3.2 Lista de exercicios	34
4 O pacote tidyverse	35
4.1 O operador pipe (%>%) da library magrittr	36
4.2 Transformação de Dados com dplyr	36
4.3 Filtrando linhas com filter()	37
4.4 Arranjando linhas com arrange()	42
4.5 Selecionando colunas com select()	43
4.6 Criando variáveis (colunas) com mutate()	46
4.7 summarise() colapsa a tabela toda em apenas uma linha	47

4.8	<code>group_by()</code> muda a “unidade de análise” de toda a tabela para os grupos definidos	47
5	ggplot	49
	References	51

Introdução

Este livro compila tutoriais da linguagem R que usei em diversas aulas que dei ao longo dos últimos anos e foi escrito baseado em Cotton (2013)

Chapter 1

Básicos da linguagem R

O conteúdo apresentado aqui foi usado na primeira aula prática do curso de “Aprendizagem Estatística com Aplicações em R”.

Para instalação,

- faça o download do R em <http://www.r-project.org>;
- sugestão: utilizar a IDE R Studio.

É muito importante saber como obter ajuda no R. Sempre que estiver em dúvidas quanto as características de alguma função, consulte a aba *help* do RStudio.

```
?mean #abre a página de ajuda da função 'mean'  
??plot #procura por tópicos contendo a palavra 'plot'
```

1.1 Usando o R como uma calculadora

O operador `+` realiza a adição entre dois elementos.

```
1 + 2
```

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

Um vetor é um conjunto ordenado de valores. O operador `:` cria uma sequência a partir de um número até outro. A função `c` concatena valores, criando um vetor.

```
1:5
```

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

```
c(1, 2, 3, 4, 5)
```

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

Além de adicionar dois números, o operador `+` pode ser usado para adicionar dois vetores.

```
1:5 + 6:10
```

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

```
c(1, 2, 3, 4, 5) + c(6, 7, 8, 9, 10)
```

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

```
1:5 + c(6, 7, 8, 9, 10)
```

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

Os próximos exemplos mostram subtração, multiplicação, exponenciação e divisão.

```
c(2, 3, 5, 7, 11, 13) - 2      #subtração
```

```
## [1] 0 1 3 5 9 11
```

```
-2:2 * -2:2      #multiplicação
```

```
## [1] 4 1 0 1 4
```

```
(1:10) ^ 2      #exponenciação
```

```
## [1] 1 4 9 16 25 36 49 64 81 100
```



```
1:10 / 3 #divisão
```

```
## [1] 0.3333333 0.6666667 1.0000000 1.3333333 1.6666667 2.0000000 2.3333333  
## [8] 2.6666667 3.0000000 3.3333333
```

1.2 Atribuindo variáveis

Fazer cálculos com o R é bem simples e útil. A maior parte das vezes queremos armazenar os resultados para uso posterior. Assim, podemos atribuir valor à uma variável, através do operador `<-`.

```
a <- 1  
b <- 5 * 3  
x <- 1:5  
y <- 6:10
```

Agora, podemos reutilizar esses valores para fazer outros cálculos.

```
a + 2 * b
```

```
## [1] 31
```

```
x + 2 * y - 3
```

```
## [1] 10 13 16 19 22
```

Observe que não temos que dizer ao R qual o tipo da variável, se era um número (as variáveis `a` e `b`) ou vetor (`x` e `y`).

1.3 Números especiais

Para facilitar operações aritméticas, R suporta quatro valores especiais de números: `Inf`, `-Inf`, `NaN` e `NA`. Os dois primeiros representam infinito positivo e negativo. `NaN` é um acrônimo inglês para “not a number”, ou seja, não é um número. Ele aparece quando um cálculo não faz sentido, ou não está definido. `NA` significa “not available”, ou seja, não disponível, e representa um valor faltante.

```
c(Inf + 1, Inf - 1, Inf - Inf, NA + 1)
```

```
## [1] Inf Inf NaN NA
```

```
c(0 / 0, Inf / Inf, 1 / Inf)
```

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

1.4 Vetores, Matrizes e Dataframes

Previamente, vimos alguns tipos de vetores para valores lógicos, caracteres e números. Nessa seção, utilizaremos técnicas de manipulação de vetores e introduziremos o caso multidimensional: matrizes e dataframes.

Abaixo relembramos as operações que já foram feitas com vetores

```
10:5 #sequência de números de 10 até 5
```

```
## [1] 10 9 8 7 6 5
```

```
c(1, 2:5, c(6, 7), 8) #valores concatenados em um único vetor
```

```
## [1] 1 2 3 4 5 6 7 8
```

1.4.1 Vetores

Existem funções para criar vetores de um tipo e com tamanho específicos. Todos os elementos deste vetor terá valor zero, FALSE, um caracter vazio, ou o equivalente à *nada/vazio* para aquele tipo. Veja abaixo duas maneiras de definir um vetor.

```
vector("numeric", 5) #cria um vetor numérico de 5 elementos
```

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

```
numeric(5) #equivalente ao comando acima
```

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

```
vector("logical", 5) #cria um vetor lógico de 5 elementos
```

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

```
logical(5) #equivalente ao comando acima
```

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

```
vector("character", 5) #cria um vetor de caracteres de 5 elementos
```

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

```
character(5) #equivalente ao comando acima
```

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

1.4.1.1 Sequências

Podemos criar sequências mais gerais que aquelas criadas com o operador `:`. A função `seq` te permite criar sequências em diferentes maneiras Veja abaixo.

```
seq(3, 12) #equivalente à 3:12
```

```
## [1] 3 4 5 6 7 8 9 10 11 12
```

```
seq(3, 12, 2) #o terceiro argumento indica a distância entre os elementos na lista.
```

```
## [1] 3 5 7 9 11
```

```
seq(0.1, 0.01, -0.01)
```

```
## [1] 0.10 0.09 0.08 0.07 0.06 0.05 0.04 0.03 0.02 0.01
```

1.4.1.2 Tamanhos

Todo vetor tem um tamanho, um número não negativo que representa a quantidade de elementos que o vetor contém. A função `length` retorna o tamanho de um dado vetor.

```
length(1:5)

## [1] 5

frase <- c("Observe", "o", "resultado", "dos", "comandos", "abaixo")
length(frase)

## [1] 6

nchar(frase)

## [1] 7 1 9 3 8 6
```

1.4.1.3 Indexando vetores

A indexação é útil quando queremos acessar elementos específicos de um vetor. Considere o vetor

```
x <- (1:5) ^ 2
```

Abaixo, três métodos de indexar os mesmos valores do vetor x.

```
x[c(1, 3, 5)]
x[c(-2, -4)]
x[c(TRUE, FALSE, TRUE, FALSE, TRUE)]
```

```
## [1] 1 9 25
```

Se nomearmos os elementos do vetor, o método abaixo obtém os mesmos valores de x.

```
names(x) <- c("one", "four", "nine", "sixteen", "twenty five")
x[c("one", "nine", "twenty five")]
```

```
##          one          nine twenty five
##          1           9          25
```

Cuidado, acessar um elemento fora do tamanho do vetor não gera um erro no R, apenas NA.

```
x[6]
```

```
## <NA>  
## NA
```

1.4.2 Matrizes

Uma matriz é o equivalente à um vetor, porém em duas dimensões. Abaixo, um exemplo de definição de uma matriz com 4 linhas e 3 colunas (total de 12 elementos).

```
?matrix
```

```
uma_matriz <- matrix(  
  1:12,  
  nrow = 4, #ncol = 3 gera o mesmo resultado. Verifique!  
  dimnames = list(  
    c("L1", "L2", "L3", "L4"),  
    c("C1", "C2", "C3")  
  )  
)  
  
class(uma_matriz)
```

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

```
uma_matriz
```

```
##      C1 C2 C3  
## L1   1  5  9  
## L2   2  6 10  
## L3   3  7 11  
## L4   4  8 12
```

Por padrão, ao criar uma matrix, o vetor passado como primeiro argumento preenche a matrix por colunas. Para preencher a matrix por linhas, basta especificar o argumento `byrow=TRUE`

A função `dim` retorna um vetor de inteiros com as dimensões da variável.

```
dim(uma_matriz)
```

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

```
nrow(uma_matriz) #retorna o número de linhas da matriz
```

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

```
ncol(uma_matriz) #retorna o número de colunas da matriz
```

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

```
length(uma_matriz) #retorna o número de elementos da matriz
```

```
## [1] 12
```

1.4.2.1 Nomeando linhas, colunas e dimensões

Da mesma forma para vetores, podemos nomear (e obter os nomes de) linhas e colunas de matrizes.

```
rownames(uma_matriz)
```

```
## [1] "L1" "L2" "L3" "L4"
```

```
colnames(uma_matriz)
```

```
## [1] "C1" "C2" "C3"
```

```
dimnames(uma_matriz)
```

```
## [[1]]  
## [1] "L1" "L2" "L3" "L4"  
##  
## [[2]]  
## [1] "C1" "C2" "C3"
```

1.4.2.2 Indexação

A indexação de matrizes funciona de maneira similar à de vetores, com a diferença que agora precisam ser especificadas mais de uma dimensão.

```
uma_matriz[1, c("C2", "C3")] #elementos na primeira linha, segunda e terceira colunas
```

```
## C2 C3
## 5 9
```

```
uma_matriz[1, ] #todos elementos da primeira linha
```

```
## C1 C2 C3
## 1 5 9
```

```
uma_matriz[, c("C2", "C3")] #todos elementos da segunda e terceira colunas
```

```
## C2 C3
## L1 5 9
## L2 6 10
## L3 7 11
## L4 8 12
```

```
uma_matriz[, c(2, 3)] #todos elementos da segunda e terceira colunas
```

```
## C2 C3
## L1 5 9
## L2 6 10
## L3 7 11
## L4 8 12
```

1.4.2.3 Combinando matrizes

Considere a seguinte matriz.

```
outra_matriz <- matrix(
  seq(2, 24, 2),
  nrow = 4,
  dimnames = list(
    c("L5", "L6", "L7", "L8"),
    c("C5", "C6", "C7")
  )
)
```

A combinação de matrizes pode ser feita através das funções `cbind` e `rbind`, que combina matrizes por colunas e por linhas, respectivamente.

```
cbind(uma_matriz, outra_matriz)
```

```
##      C1 C2 C3 C5 C6 C7
## L1   1  5  9  2 10 18
## L2   2  6 10  4 12 20
## L3   3  7 11  6 14 22
## L4   4  8 12  8 16 24
```

```
rbind(uma_matriz, outra_matriz)
```

```
##      C1 C2 C3
## L1   1  5  9
## L2   2  6 10
## L3   3  7 11
## L4   4  8 12
## L5   2 10 18
## L6   4 12 20
## L7   6 14 22
## L8   8 16 24
```

1.4.2.4 Operações com matrizes

As operações básicas (+, -, *, /) funcionam de elemento a elemento em matrizes, da mesma forma como em vetores:

```
uma_matriz + outra_matriz
```

```
##      C1 C2 C3
## L1   3 15 27
## L2   6 18 30
## L3   9 21 33
## L4  12 24 36
```

```
uma_matriz * outra_matriz
```

```
##      C1 C2 C3
## L1   2 50 162
## L2   8 72 200
## L3  18 98 242
## L4  32 128 288
```

Cuidado: as matrizes e vetores devem ter tamanhos compatíveis!

Chapter 2

Controle de fluxo e repetição

Este capítulo foi escrito baseado em Cotton (2013) e o conteúdo apresentado aqui foi usado como base para o minicurso “R para não-programadores”.

Por algum motivo particular, você pode querer executar ou não um determinado trecho do seu código. Também, pode querer que um trecho seja executado repetidas vezes.

2.0.1 if e else

A maneira mais simples de controlar o fluxo de execução é adicionando um valor lógico e executando o comando `if`.

```
if(TRUE) message("Isso é verdadeiro!")
```

```
## Isso é verdadeiro!
```

```
if(FALSE) message("Isso não é verdadeiro!")  
if(is.na(NA)) message("O valor está faltando!")
```

```
## O valor está faltando!
```

```

if(runif(1) > 0.5) message("Essa mensagem ocorre com 50% de chance.")

x <- 3
if(x > 2) {
  y <- 2 * x
  z <- 3 * y
}

```

Para executar um comando caso a condição do `if` não seja verdadeira, utilize o `else`.

```

if(FALSE)
{
  message("Não será executado...")
} else #o else deve estar na mesma linha que fecha as chaves
{
  message("mas este será.")
}

```

```
## mas este será.
```

Como R é uma linguagem vetorizada, é possível vetorizar estruturas de controle de fluxo, através da função `ifelse`. Esta função necessita de três argumentos: o primeiro é um vetor de valor lógico, o segundo contém os valores que serão retornados quando a condição lógica é verdadeira (`TRUE`), o terceiro contém os valores que serão retornados quando a condição é falsa (`FALSE`).

```

set.seed(123)
numeros <- sample(1:100, 10)

maiores_50 <- ifelse(numeros > 50, "Maior que 50", "Menor ou igual a 50")

maiores_50 <- factor(maiores_50)
table(maiores_50)

```

```

## maiores_50
##          Maior que 50 Menor ou igual a 50
##                4                6

```

2.0.2 Estruturas de repetição

2.0.2.1 while

O `while` primeiramente checa a condição e, caso verdadeira, a executa. No exemplo abaixo, o `while` será executado somente se `acao` for diferente de “Des-

cansar”.

```
acoes <- c( "Aprender a programar em R",
            "Fazer um café",
            "Assistir um filme",
            "Descansar")
```

```
acao <- sample(acoes, 1)
```

```
print(acao)
```

```
## [1] "Fazer um café"
```

```
while(acao != "Descansar") {
  acao <- sample(acoes, 1)
  print(acao)
}
```

```
## [1] "Assistir um filme"
```

```
## [1] "Descansar"
```

2.0.2.2 for

É usado quando se sabe exatamente quantas vezes o trecho de código deverá ser repetido. O `for` aceita uma variável iterativa bem como um vetor. O loop é repetido, dando ao iterador cada elemento do vetor por vez. O caso mais simples é um vetor contendo inteiros.

```
for(i in 1:5) {
  j <- i ^ 2
  print(paste("j = ", j))
}
```

```
## [1] "j = 1"
```

```
## [1] "j = 4"
```

```
## [1] "j = 9"
```

```
## [1] "j = 16"
```

```
## [1] "j = 25"
```

```
for(month in month.name)
{
  print(paste0("The month of ", month))
}
```

```
## [1] "The month of January"
## [1] "The month of February"
## [1] "The month of March"
## [1] "The month of April"
## [1] "The month of May"
## [1] "The month of June"
## [1] "The month of July"
## [1] "The month of August"
## [1] "The month of September"
## [1] "The month of October"
## [1] "The month of November"
## [1] "The month of December"
```

No exemplo abaixo, sorteamos 10 números entre 1 e 100 e, para cada número, se ele for par, calculamos o quadrado, se for ímpar, somamos 1.

```
set.seed(2)

numeros <- sample(1:100, 10)
resultado <- numeric(length(numeros))
for(i in 1:length(numeros)) {
  if(numeros[i] %% 2 == 0)
    resultado[i] <- numeros[i]^2
  else
    resultado[i] <- numeros[i]+1
}

rbind(numeros, resultado)
```

```
##           [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## numeros      85   79   70    6   32    8   17   93   81    76
## resultado    86   80 4900   36 1024   64   18   94   82  5776
```

Uma alternativa é usar a função `ifelse` para fazer a mesma operação acima.

```
ifelse(numeros %% 2 == 0, numeros^2, numeros+1)
```

```
## [1]    86    80 4900    36 1024    64    18    94    82 5776
```

2.1 Funções

Os tipos e estruturas de variáveis são importantes para armazenamento de dados, funções nos permitem processar os dados.

2.1.0.0.1 Criando e chamando funções Uma função é criada de maneira parecida com a criação de variáveis, atribuindo um nome à função.

```
hipotenusa <- function(x, y) {
  sqrt(x^2 + y^2)
}
```

Aqui, `hipotenusa` é o nome da função criada, `x` e `y` são os argumentos e o conteúdo entre chaves é chamado de corpo da função.

```
hipotenusa(3, 4) #sem nomear, os parametros sao considerados seguindo a ordem de definicao da fun
```

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

```
hipotenusa(y=24, x=7)
```

```
## [1] 25
```

Pode-se definir valores padrão para argumentos de uma função. Na função abaixo, que calcula a potenciação, caso `exp` não seja definido, a função calculará o quadrado do número dado.

```
potencia <- function(base, exp = 2) {
  base ^ exp
}
```

```
potencia(10)
```

```
## [1] 100
```

```
potencia(10, 3)
```

```
## [1] 1000
```

```
potencia(c(1:10, NA))
```

```
## [1] 1 4 9 16 25 36 49 64 81 100 NA
```

2.2 Lista de exercicios

1. O comando abaixo gera amostras aleatórias seguindo a distribuição binomial para simular o lançamento de 100 moeda.

```
set.seed(1)
lancamentos <- rbinom(100, 1, 0.5)
```

Considere que 0 represente “cara” e 1 represente “coroa.” Crie uma variável para armazenar os lançamentos como um vetor contendo os níveis “cara” e “coroa”.

2. Para fins de exemplo, neste exercício considere apenas uma amostra de 1000 linhas dos dados armazenados no dataframe `flights` do pacote `nycflights`.

```
library(nycflights13)
```

```
## Warning: package 'nycflights13' was built under R version 4.2.3
```

```
set.seed(1)
flights <- flights[sample(1:nrow(flights), 1000),]
```

- a) De acordo com a descrição do mesmo (veja `?flights`), a coluna `distance` está registrada em milhas. Crie uma função que receba como argumento um número em milhas e converta-o para quilômetros (para um resultado aproximado, multiplique o valor de comprimento por 1,609). Em seguida, crie um novo dataframe, `flights2` em que sua coluna `distance` esteja representada em km.
- b) Crie uma sequência de comandos utilizando a estrutura `for` para classificar cada uma das distâncias obtidas no item anterior em “curta distância” (até 500km), “média distância” (entre 500km e 2000km) e “longa distância” (mais que 2000km). Armazene o resultado obtido em uma nova coluna no dataframe `flights`. Transforme essa coluna em fator.

Chapter 3

Importação e manipulação de bases de dados

Este capítulo foi escrito baseado em Cotton (2013) e o conteúdo apresentado aqui foi usado como base para o minicurso “R para não-programadores”.

O tipo de dados mais comum de ser importado para o R são aqueles que são armazenadas em planilhas. Nesta aula, vamos trabalhar com os dados contidos na biblioteca `nycflights13`, porém, vamos considerar a versão dos dados salvos em arquivos `.csv`.

Para importar dados do tipo `.csv` (valores separados por vírgula), utilize o comando abaixo.

```
flights <- read.csv("flights.csv")
airports <- read.csv("airports.csv")

class(flights)
```

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

```
class(airports)
```

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

Veja a seguir os comandos para analisar a estrutura dos dados e obter uma descrição do tipo de cada coluna presente na base.

```
str(flights)
```

```
## 'data.frame':    336776 obs. of  20 variables:
## $ X          : int  1 2 3 4 5 6 7 8 9 10 ...
## $ year       : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
## $ month      : int  1 1 1 1 1 1 1 1 1 1 ...
## $ day        : int  1 1 1 1 1 1 1 1 1 1 ...
## $ dep_time   : int  517 533 542 544 554 554 555 557 557 558 ...
## $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
## $ dep_delay  : int  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
## $ arr_time   : int  830 850 923 1004 812 740 913 709 838 753 ...
## $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
## $ arr_delay  : int  11 20 33 -18 -25 12 19 -14 -8 8 ...
## $ carrier    : chr  "UA" "UA" "AA" "B6" ...
## $ flight     : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
## $ tailnum    : chr  "N14228" "N24211" "N619AA" "N804JB" ...
## $ origin     : chr  "EWR" "LGA" "JFK" "JFK" ...
## $ dest       : chr  "IAH" "IAH" "MIA" "BQN" ...
## $ air_time   : int  227 227 160 183 116 150 158 53 140 138 ...
## $ distance   : int  1400 1416 1089 1576 762 719 1065 229 944 733 ...
## $ hour       : int  5 5 5 5 6 5 6 6 6 6 ...
## $ minute     : int  15 29 40 45 0 58 0 0 0 0 ...
## $ time_hour  : chr  "2013-01-01 05:00:00" "2013-01-01 05:00:00" "2013-01-01 05:00:00"
```

```
str(airports)
```

```
## 'data.frame':    1458 obs. of  9 variables:
## $ X          : int  1 2 3 4 5 6 7 8 9 10 ...
## $ faa        : chr  "04G" "06A" "06C" "06N" ...
## $ name       : chr  "Lansdowne Airport" "Moton Field Municipal Airport" "Schaumburg Regional Airport" ...
## $ lat        : num  41.1 32.5 42 41.4 31.1 ...
## $ lon        : num  -80.6 -85.7 -88.1 -74.4 -81.4 ...
## $ alt        : int  1044 264 801 523 11 1593 730 492 1000 108 ...
## $ tz         : int  -5 -6 -6 -5 -5 -5 -5 -5 -5 -8 ...
## $ dst        : chr  "A" "A" "A" "A" ...
## $ tzone      : chr  "America/New_York" "America/Chicago" "America/Chicago" "America/New_York"
```

Frequentemente, você deverá alterar o tipo de alguma(s) coluna(s), como, por exemplo, para tratar dados do tipo fator, caracteres e datas.

Nos dados armazenados em `flights`, podemos converter as colunas `carrier`, `origin` e `dest` para fator.


```
flights$carrier <- as.factor(flights$carrier)
flights$origin <- as.factor(flights$origin)
flights$dest <- as.factor(flights$dest)
```

Na tabela airport, podemos converter a coluna tzone para o tipo fator.

```
airports$tzone <- as.factor(airports$tzone)
```

Note que a coluna flights\$time_hour representa a hora data e hora do voo. Com a função head() podemos visualizar as primeiras observações de uma coluna (e de um data frame também).

```
head(flights$time_hour)
```

```
## [1] "2013-01-01 05:00:00" "2013-01-01 05:00:00" "2013-01-01 05:00:00"
## [4] "2013-01-01 05:00:00" "2013-01-01 06:00:00" "2013-01-01 05:00:00"
```

```
class(flights$time_hour)
```

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

Entretanto, a classe de flights\$time_hour é character, o que nos impossibilita trabalhar com datas de maneira adequada. Uma alternativa, é utilizar a biblioteca lubridate. Para isso, execute o comando abaixo para instalá-la.

```
install.packages("lubridate")
```

Em seguida, basta carregar a biblioteca e utilizar as funções que fazem conversão de caracteres em data e manipulações relacionadas a este tipo de dado. Abaixo, adicionamos períodos de tempos à uma data (anos, meses, dias), obtemos a diferença entre duas datas e selecionamos a unidade de tempo para exibir essa diferença.

```
library(lubridate)
```

```
## Carregando pacotes exigidos: timechange
```

```
##
```

```
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##      date, intersect, setdiff, union
```

```
data <- "25/12/92"
class(data)
```

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

```
data <- dmy(data)
class(data)
```

```
## [1] "Date"
```

```
year(data)
```

```
## [1] 1992
```

```
month(data)
```

```
## [1] 12
```

```
day(data)
```

```
## [1] 25
```

```
# adicionar 10 anos a uma data
data + years(10)
```

```
## [1] "2002-12-25"
```

```
# adicionar 10 anos e 5 meses a uma data
data + years(10) + months(5)
```

```
## [1] "2003-05-25"
```

```
# adicionar 10 anos, 5 meses e 7 dias a uma data
data + years(10) + months(5) + days(7)
```

```
## [1] "2003-06-01"
```

```
# obter a diferença de duas datas em dias
interval(data, today()) / years(1)
```

```
## [1] 30.99452
```

```
# obter a diferença de duas datas em dias
interval(data, today()) / days(1)
```

```
## [1] 11320
```

Este pacote nos permite trabalhar com data e hora em uma mesma variável. Note que agora a data está representada no formato MM/DD/AAAA, além de hora, minutos e segundos.

```
data <- "12/25/92 10:07:35"
class(data)
```

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

```
data <- mdy_hms(data)
class(data)
```

```
## [1] "POSIXct" "POSIXt"
```

```
year(data)
```

```
## [1] 1992
```

```
hour(data)
```

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

```
minute(data)
```

```
## [1] 7
```

```
second(data)
```

```
## [1] 35
```

Com isso, podemos converter a coluna `flights$time_hour` para o tipo de dados `date`.

```
flights$time_hour <- ymd_hms(flights$time_hour)
```

Agora, todas as colunas de `flights` estão com os tipos de dados corretos.

```
str(flights)
```

```
## 'data.frame':    336776 obs. of  20 variables:
## $ X              : int  1 2 3 4 5 6 7 8 9 10 ...
## $ year           : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
## $ month          : int  1 1 1 1 1 1 1 1 1 1 ...
## $ day            : int  1 1 1 1 1 1 1 1 1 1 ...
## $ dep_time       : int  517 533 542 544 554 554 555 557 557 558 ...
## $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
## $ dep_delay      : int  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
## $ arr_time       : int  830 850 923 1004 812 740 913 709 838 753 ...
## $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
## $ arr_delay      : int  11 20 33 -18 -25 12 19 -14 -8 8 ...
## $ carrier        : Factor w/ 16 levels "9E","AA","AS",...: 12 12 2 4 5 12 4 6 4 2 ..
## $ flight         : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
## $ tailnum        : chr   "N14228" "N24211" "N619AA" "N804JB" ...
## $ origin         : Factor w/ 3 levels "EWR","JFK","LGA": 1 3 2 2 3 1 1 3 2 3 ...
## $ dest           : Factor w/ 105 levels "ABQ","ACK","ALB",...: 44 44 59 13 5 70 36 4...
## $ air_time       : int  227 227 160 183 116 150 158 53 140 138 ...
## $ distance       : int  1400 1416 1089 1576 762 719 1065 229 944 733 ...
## $ hour           : int  5 5 5 5 6 5 6 6 6 6 ...
## $ minute         : int  15 29 40 45 0 58 0 0 0 0 ...
## $ time_hour      : POSIXct, format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

3.1 Análise descritiva

Realizar uma análise descritiva é fundamental para compreender, de forma resumida, a informação contida nos dados.

Uma função que resume de maneira “inteligente” todas as colunas de um data frame é a `summary`. É “inteligente” porque, a forma como o resumo é feito depende do tipo de cada coluna da tabela. Em `flights` há colunas numéricas, fatores, caracter e data.

```
summary(flights)
```

```
##           X                year           month           day
## Min.      :      1   Min.    :2013   Min.      : 1.000   Min.    : 1.00
## 1st Qu.: 84195   1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00
```

```
## Median :168389   Median :2013   Median : 7.000   Median :16.00
## Mean :168389   Mean :2013   Mean : 6.549   Mean :15.71
## 3rd Qu.:252582   3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00
## Max. :336776   Max. :2013   Max. :12.000   Max. :31.00
##
##      dep_time      sched_dep_time    dep_delay      arr_time      sched_arr_time
## Min. : 1      Min. : 106      Min. : -43.00      Min. : 1      Min. : 1
## 1st Qu.: 907    1st Qu.: 906      1st Qu.: -5.00      1st Qu.:1104    1st Qu.:1124
## Median :1401    Median :1359      Median : -2.00      Median :1535    Median :1556
## Mean :1349      Mean :1344      Mean : 12.64      Mean :1502      Mean :1536
## 3rd Qu.:1744    3rd Qu.:1729      3rd Qu.: 11.00      3rd Qu.:1940    3rd Qu.:1945
## Max. :2400      Max. :2359      Max. :1301.00      Max. :2400      Max. :2359
## NA's :8255      NA's :8255      NA's :8713
##      arr_delay      carrier      flight      tailnum
## Min. : -86.000      UA :58665      Min. : 1      Length:336776
## 1st Qu.: -17.000      B6 :54635      1st Qu.: 553      Class :character
## Median : -5.000      EV :54173      Median :1496      Mode :character
## Mean : 6.895      DL :48110      Mean :1972
## 3rd Qu.: 14.000      AA :32729      3rd Qu.:3465
## Max. :1272.000      MQ :26397      Max. :8500
## NA's :9430      (Other):62067
##      origin      dest      air_time      distance      hour
## EWR:120835      ORD : 17283      Min. : 20.0      Min. : 17      Min. : 1.00
## JFK:111279      ATL : 17215      1st Qu.: 82.0      1st Qu.: 502      1st Qu.: 9.00
## LGA:104662      LAX : 16174      Median :129.0      Median : 872      Median :13.00
##      BOS : 15508      Mean :150.7      Mean :1040      Mean :13.18
##      MCO : 14082      3rd Qu.:192.0      3rd Qu.:1389      3rd Qu.:17.00
##      CLT : 14064      Max. :695.0      Max. :4983      Max. :23.00
##      (Other):242450      NA's :9430
##      minute      time_hour
## Min. : 0.00      Min. :2013-01-01 05:00:00.00
## 1st Qu.: 8.00      1st Qu.:2013-04-04 13:00:00.00
## Median :29.00      Median :2013-07-03 10:00:00.00
## Mean :26.23      Mean :2013-07-03 05:02:36.49
## 3rd Qu.:44.00      3rd Qu.:2013-10-01 07:00:00.00
## Max. :59.00      Max. :2013-12-31 23:00:00.00
##
```

Vamos obter a média de atrasos por aeroporto de origem. Antes, vamos verificar quantos aeroportos de origem existem na base.

```
unique(flights$origin)
```

```
## [1] EWR LGA JFK
## Levels: EWR JFK LGA
```

Há quantos voos registrados saindo de cada aeroporto?

```
table(flights$origin)
```

```
##
##      EWR      JFK      LGA
## 120835 111279 104662
```

Exercício: qual o nome dos aeroportos cujos códigos FAA estão listados acima? (Dica: fazer merge com `airports`).

Agora, vamos calcular a média de atraso geral para voos que partiram de “JFK”.

```
mean(flights$dep_delay[flights$origin == "JFK"])
```

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

Note que existem dados faltantes (do tipo `NA`) na coluna `dep_delay`. Podemos desconsiderar esses dados no cálculo da média.

```
mean(flights$dep_delay[flights$origin == "JFK"], na.rm = TRUE)
```

```
## [1] 12.11216
```

Podemos utilizar a função `tapply` para obter o resultado de uma função aplicada de acordo com certos grupos. Por exemplo, para obter a média de atraso dos voos de cada um dos aeroportos de origem, use o comando abaixo.

```
tapply(flights$dep_delay, flights$origin, mean, na.rm = TRUE)
```

```
##      EWR      JFK      LGA
## 15.10795 12.11216 10.34688
```

Para arredondar números, podemos usar a função `round`, informando quantas casas decimais devem ser consideradas.

```
round(tapply(flights$dep_delay, flights$origin, mean, na.rm = TRUE), 2)
```

```
##      EWR      JFK      LGA
## 15.11 12.11 10.35
```

Podemos fazer o mesmo considerando os aeroportos de destino.

```
unique(flights$dest)
```

```
##      [1] IAH MIA BQN ATL ORD FLL IAD MCO PBI TPA LAX SFO DFW BOS LAS MSP DTW RSW
##     [19] SJU PHX BWI CLT BUF DEN SNA MSY SLC XNA MKE SEA ROC SYR SRQ RDU CMH JAX
##     [37] CHS MEM PIT SAN DCA CLE STL MYR JAC MDW HNL BNA AUS BTW PHL STT EGE AVL
##     [55] PWM IND SAV CAK HOU LGB DAY ALB BDL MHT MSN GSO CVG BUR RIC GSP GRR MCI
##     [73] ORF SAT SDF PDX SJC OMA CRW OAK SMF TUL TYS OKC PVD DSM PSE BHM CAE HDN
##     [91] BZN MTJ EYW PSP ACK BGR ABQ ILM MVY SBN LEX CHO TVC ANC LGA
## 105 Levels: ABQ ACK ALB ANC ATL AUS AVL BDL BGR BHM BNA BOS BQN BTW BUF ... XNA
```

```
round(tapply(flights$arr_delay, flights$origin, mean, na.rm = TRUE), 2)
```

```
##      EWR      JFK      LGA
##  9.11  5.55  5.78
```

Saber somente a média de uma medida não é suficiente, devemos obter outras estatísticas, como a mediana, desvio padrão e variância, por exemplo.

```
tapply(flights$dep_delay, flights$origin, median, na.rm = TRUE)
```

```
##      EWR      JFK      LGA
##    -1    -1    -3
```

```
tapply(flights$dep_delay, flights$origin, sd, na.rm = TRUE)
```

```
##           EWR           JFK           LGA
## 41.32370 39.03507 39.99302
```

```
tapply(flights$dep_delay, flights$origin, var, na.rm = TRUE)
```

```
##           EWR           JFK           LGA
## 1707.649 1523.737 1599.442
```

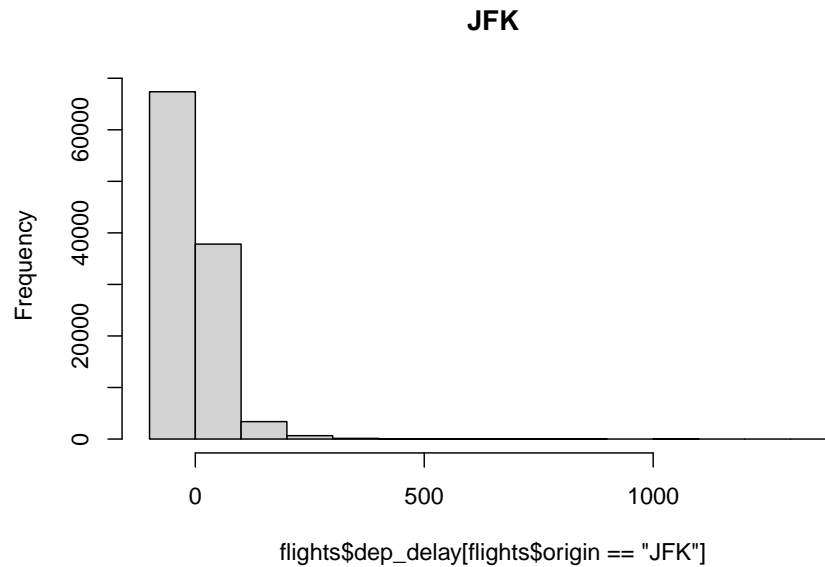
Podemos combinar todos esses resultados acima em uma única tabela, usando o comando `cbind`. Abaixo, acrescentamos ainda os valores mínimo e máximo de atraso na partida observado em cada aeroporto.

```
cbind(Media = tapply(flights$dep_delay, flights$origin, mean, na.rm = TRUE),
      Mediana = tapply(flights$dep_delay, flights$origin, median, na.rm = TRUE),
      Minimo = tapply(flights$dep_delay, flights$origin, min, na.rm = TRUE),
      Maximo = tapply(flights$dep_delay, flights$origin, max, na.rm = TRUE),
      Desvio_padrao = tapply(flights$dep_delay, flights$origin, sd, na.rm = TRUE),
      Variancia = tapply(flights$dep_delay, flights$origin, var, na.rm = TRUE))
```

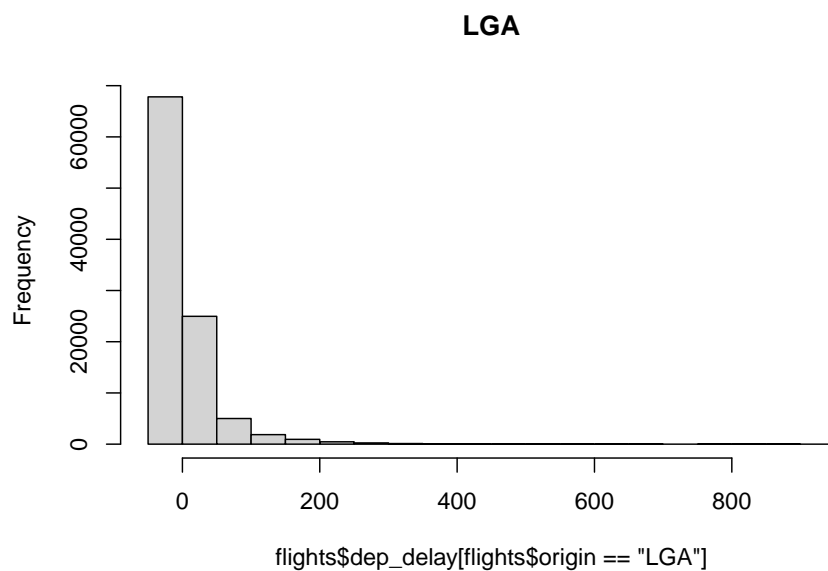
##		Media	Mediana	Minimo	Maximo	Desvio_padrao	Variancia
##	EWB	15.10795	-1	-25	1126	41.32370	1707.649
##	JFK	12.11216	-1	-43	1301	39.03507	1523.737
##	LGA	10.34688	-3	-33	911	39.99302	1599.442

Além disso, podemos construir gráficos, para analisar os dados de forma visual. Abaixo, criamos histogramas do atraso na partida de cada um dos três aeroportos de Nova York.

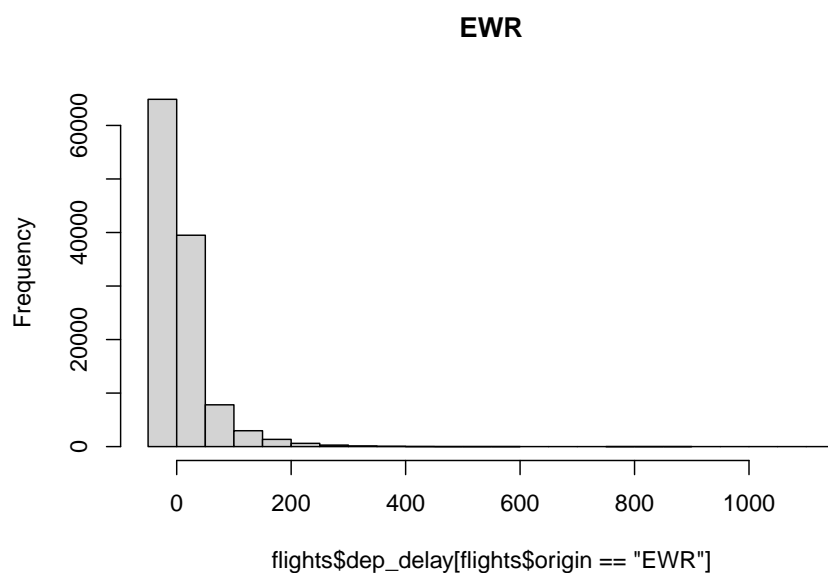
```
hist(flights$dep_delay[flights$origin == "JFK"], main="JFK")
```



```
hist(flights$dep_delay[flights$origin == "LGA"], main="LGA")
```

```
hist(flights$dep_delay[flights$origin == "EWR"], main="EWR")
```



Os três gráficos podem ser exibidos de uma única vez se usarmos a opção `mfrow`, que permite definir o número de linhas e colunas do grid para plotar os gráficos.

```

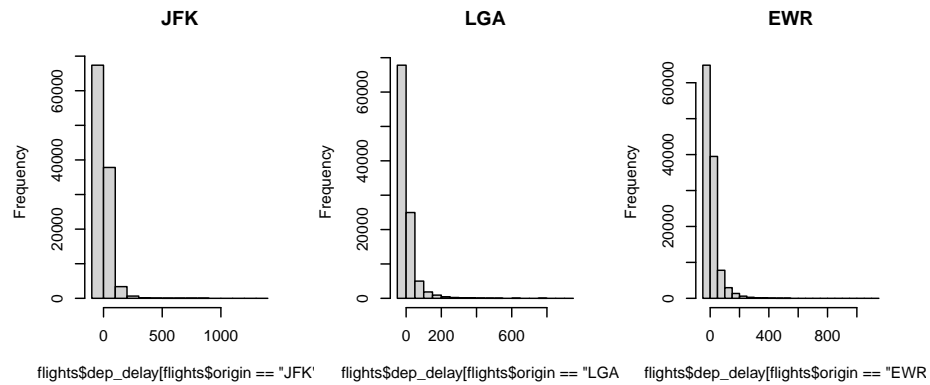
par(mfrow=c(1,3))

hist(flights$dep_delay[flights$origin == "JFK"], main="JFK")

hist(flights$dep_delay[flights$origin == "LGA"], main="LGA")

hist(flights$dep_delay[flights$origin == "EWR"], main="EWR")

```



3.2 Lista de exercicios

1. Importe no R a planilha de dados correspondente ao seu projeto aplicado.
2. Verifique se alguma coluna deve ser transformada para um tipo de dados mais adequado.
3. Obtenha estatísticas descritivas dos dados.

Chapter 4

O pacote tidyverse

Este capítulo foi baseado no livro R for Data Science, que atualmente está sendo traduzido para o português brasileiro através de um grupo de pessoas voluntárias (mais detalhes aqui).

tidyverse é uma coleção de bibliotecas criadas para o universo de data science. Todos os pacotes ‘tidyverse’ possuem a mesma gramática, estrutura de dados e filosofia:

Veja todos os pacotes disponíveis em: <https://www.tidyverse.org/>

Vamos instalar o pacote **tidyverse**:

```
install.packages("tidyverse")
```

E carregar a biblioteca:

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.3.2 --
## v ggplot2 3.4.2      v purrr   0.3.5
## v tibble  3.2.1      v dplyr  1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2

## Warning: package 'ggplot2' was built under R version 4.2.3

## Warning: package 'tibble' was built under R version 4.2.3
```

```
## -- Conflicts ----- tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()      masks base::date()
## x dplyr::filter()        masks stats::filter()
## x lubridate::intersect() masks base::intersect()
## x dplyr::lag()            masks stats::lag()
## x lubridate::setdiff()    masks base::setdiff()
## x lubridate::union()      masks base::union()
```

4.1 O operador pipe (%>%) da library magrittr

Atalho no teclado: Ctrl + Shift + M

Passa o objeto do lado esquerdo como primeiro argumento (ou `.argumento`) da função do lado direito:

- `x %>% f(y)` é equivalente a `f(x,y)`
- `y %>% f(x,.,z)` é equivalente a `f(x,y,z)`

Na prática, vamos supor que queremos somar todos os elementos do **vetor** e em seguida tirar a raiz quadrada desta soma:

```
vetor <- c(20,40,60,80,200)

#raiz da soma
sqrt(sum(vetor))
```

```
## [1] 20
```

Usando o pipe:

```
vetor %>% sum() %>% sqrt()
```

```
## [1] 20
```

4.2 Transformação de Dados com dplyr

As cinco principais funções do `dplyr` são:

- filter()
- arrange()
- select()
- mutate()
- summarize()

Todos os verbos funcionam de maneira similar:

1. O primeiro argumento é um data frame
2. Os próximos argumentos descrevem o que fazer com o data frame
3. O resultado é um novo data frame

4.3 Filtrando linhas com filter()

Vamos voltar a utilizar a base de dados flights:

```
flights <- read.csv("flights.csv")
```

Sem pipe:

```
filter(flights, month == 1, day == 1)
```

```
## # A tibble: 842 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1  2013     1     1     517             515           2     830             819
## 2  2013     1     1     533             529           4     850             830
## 3  2013     1     1     542             540           2     923             850
## 4  2013     1     1     544             545          -1    1004            1022
## 5  2013     1     1     554             600          -6     812             837
## 6  2013     1     1     554             558          -4     740             728
## 7  2013     1     1     555             600          -5     913             854
## 8  2013     1     1     557             600          -3     709             723
## 9  2013     1     1     557             600          -3     838             846
## 10 2013     1     1     558             600          -2     753             745
## # i 832 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

Com pipe:

```
flights %>% filter(month == 1, day == 1)
```

```
## # A tibble: 842 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1  2013     1     1     517           515           2     830           819
## 2  2013     1     1     533           529           4     850           830
## 3  2013     1     1     542           540           2     923           850
## 4  2013     1     1     544           545          -1    1004          1022
## 5  2013     1     1     554           600          -6     812           837
## 6  2013     1     1     554           558          -4     740           728
## 7  2013     1     1     555           600          -5     913           854
## 8  2013     1     1     557           600          -3     709           723
## 9  2013     1     1     557           600          -3     838           846
## 10 2013     1     1     558           600          -2     753           745
## # i 832 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

Atribuindo e imprimindo:

```
# Para atribuir e imprimir de uma só vez, coloque
# parênteses em volta da atribuição (os dois espaços
# depois dos parênteses abaixo não são necessários).

( jan1 <- flights %>% filter(month == 1, day == 1) )
```

```
## # A tibble: 842 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1  2013     1     1     517           515           2     830           819
## 2  2013     1     1     533           529           4     850           830
## 3  2013     1     1     542           540           2     923           850
## 4  2013     1     1     544           545          -1    1004          1022
## 5  2013     1     1     554           600          -6     812           837
## 6  2013     1     1     554           558          -4     740           728
## 7  2013     1     1     555           600          -5     913           854
## 8  2013     1     1     557           600          -3     709           723
## 9  2013     1     1     557           600          -3     838           846
## 10 2013     1     1     558           600          -2     753           745
## # i 832 more rows
```

```
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,  
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,  
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

Podemos utilizar os operadores lógicos aprendidos nas primeiras aulas para filtrar aqui também.

Por exemplo, vamos filtrar somente as observações que **não** são NA na coluna `arr_delay`:

```
flights %>% filter(!is.na(arr_delay))
```

```
## # A tibble: 327,346 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1  2013     1     1     517             515           2       830           819
## 2  2013     1     1     533             529           4       850           830
## 3  2013     1     1     542             540           2       923           850
## 4  2013     1     1     544             545          -1      1004          1022
## 5  2013     1     1     554             600          -6       812           837
## 6  2013     1     1     554             558          -4       740           728
## 7  2013     1     1     555             600          -5       913           854
## 8  2013     1     1     557             600          -3       709           723
## 9  2013     1     1     557             600          -3       838           846
## 10 2013     1     1     558             600          -2       753           745
## # i 327,336 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

Origem = Kennedy ou Newark, Destino = Los Angeles:

```
flights %>% filter(origin %in% c("JFK", "EWR"), dest == "LAX")
```

```
## # A tibble: 16,174 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1  2013     1     1     558             600          -2       924           917
## 2  2013     1     1     628             630          -2      1016           947
## 3  2013     1     1     658             700          -2      1027          1025
## 4  2013     1     1     702             700           2      1058          1014
## 5  2013     1     1     743             730          13      1107          1100
## 6  2013     1     1     828             823           5      1150          1143
## 7  2013     1     1     829             830          -1      1152          1200
## 8  2013     1     1     856             900          -4      1226          1220
## 9  2013     1     1     859             900          -1      1223          1225
## 10 2013     1     1     921             900          21      1237          1227
## # i 16,164 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
```



```
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,  
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

4.4 Arranjando linhas com `arrange()`

Organizando os dados segundo a ordem crescente da coluna `dep_delay`:

```
flights %>% arrange(dep_delay)
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1  2013    12     7    2040           2123         -43     40           2352
## 2  2013     2     3    2022           2055         -33    2240           2338
## 3  2013    11    10    1408           1440         -32    1549           1559
## 4  2013     1    11    1900           1930         -30    2233           2243
## 5  2013     1    29    1703           1730         -27    1947           1957
## 6  2013     8     9     729            755         -26    1002            955
## 7  2013    10    23    1907           1932         -25    2143           2143
## 8  2013     3    30    2030           2055         -25    2213           2250
## 9  2013     3     2    1431           1455         -24    1601           1631
## 10 2013     5     5     934            958         -24    1225           1309
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

Agora usando a ordem decrescente:

```
flights %>% arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1  2013     1     9     641            900        1301    1242           1530
## 2  2013     6    15    1432           1935        1137    1607           2120
## 3  2013     1    10    1121           1635        1126    1239           1810
## 4  2013     9    20    1139           1845        1014    1457           2210
## 5  2013     7    22     845           1600        1005    1044           1815
## 6  2013     4    10    1100           1900         960    1342           2211
## 7  2013     3    17    2321            810         911     135           1020
## 8  2013     6    27     959           1900         899    1236           2226
## 9  2013     7    22    2257            759         898     121           1026
## 10 2013    12     5     756           1700         896    1058           2020
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

Se você fornecer mais de uma coluna, as demais colunas serão usadas sucessivamente para decidir os empates:

```
flights %>% arrange(desc(month), day)
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1  2013    12     1      13           2359          14     446           445
## 2  2013    12     1      17           2359          18     443           437
## 3  2013    12     1     453           500          -7     636           651
## 4  2013    12     1     520           515           5     749           808
## 5  2013    12     1     536           540          -4     845           850
## 6  2013    12     1     540           550         -10    1005          1027
## 7  2013    12     1     541           545          -4     734           755
## 8  2013    12     1     546           545           1     826           835
## 9  2013    12     1     549           600         -11     648           659
## 10 2013    12     1     550           600         -10     825           854
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

4.5 Selecionando colunas com select()

Vamos supor que eu só queira utilizar colunas específicas da minha base: carrier, year, month e day

```
flights %>% select(carrier, year, month, day)
```

```
## # A tibble: 336,776 x 4
##   carrier year month   day
##   <chr>   <int> <int> <int>
## 1 UA      2013     1     1
## 2 UA      2013     1     1
## 3 AA      2013     1     1
## 4 B6      2013     1     1
## 5 DL      2013     1     1
## 6 UA      2013     1     1
## 7 B6      2013     1     1
## 8 EV      2013     1     1
## 9 B6      2013     1     1
## 10 AA     2013     1     1
## # i 336,766 more rows
```

Selecionando todas as colunas, MENOS a coluna `year`:

```
flights %>% select(-year)
```

```
## # A tibble: 336,776 x 18
##   month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int>   <int>         <int>         <dbl>   <int>         <int>
## 1     1     1     517             515           2     830           819
## 2     1     1     533             529           4     850           830
## 3     1     1     542             540           2     923           850
## 4     1     1     544             545          -1    1004          1022
## 5     1     1     554             600          -6     812           837
## 6     1     1     554             558          -4     740           728
## 7     1     1     555             600          -5     913           854
## 8     1     1     557             600          -3     709           723
## 9     1     1     557             600          -3     838           846
## 10    1     1     558             600          -2     753           745
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dtm>
```

`everything()` é útil para mover as colunas de lugar:

```
flights %>% select(carrier, origin, everything())
```

```
## # A tibble: 336,776 x 19
##   carrier origin  year month   day dep_time sched_dep_time dep_delay arr_time
##   <chr>   <chr> <int> <int> <int>   <int>         <int>         <dbl>   <int>
## 1 UA      EWR   2013     1     1     517             515           2     830
## 2 UA      LGA   2013     1     1     533             529           4     850
## 3 AA      JFK   2013     1     1     542             540           2     923
## 4 B6      JFK   2013     1     1     544             545          -1    1004
## 5 DL      LGA   2013     1     1     554             600          -6     812
## 6 UA      EWR   2013     1     1     554             558          -4     740
## 7 B6      EWR   2013     1     1     555             600          -5     913
## 8 EV      LGA   2013     1     1     557             600          -3     709
## 9 B6      JFK   2013     1     1     557             600          -3     838
## 10 AA     LGA   2013     1     1     558             600          -2     753
## # i 336,766 more rows
## # i 10 more variables: sched_arr_time <int>, arr_delay <dbl>, flight <int>,
## #   tailnum <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dtm>
```

Também podemos renomear uma coluna dentro do `select`:

#renomeando a coluna year para "ano":

```
flights %>% select("ano" = year, month, day)
```

```
## # A tibble: 336,776 x 3
##   ano month   day
##   <int> <int> <int>
## 1  2013     1     1
## 2  2013     1     1
## 3  2013     1     1
## 4  2013     1     1
## 5  2013     1     1
## 6  2013     1     1
## 7  2013     1     1
## 8  2013     1     1
## 9  2013     1     1
## 10 2013     1     1
## # i 336,766 more rows
```

Combinando `filter()`, `select()` e `arrange()`:

Suponha que nosso objetivo seja verificar qual a companhia aérea que mais atrasa nos vôos entre JFK e LAX:

```
flights %>%
  filter(origin == "JFK", dest == "LAX") %>%
  select(carrier, dep_delay) %>%
  arrange(desc(dep_delay))
```

```
## # A tibble: 11,262 x 2
##   carrier dep_delay
##   <chr>      <dbl>
## 1 DL          800
## 2 VX          634
## 3 VX          434
## 4 VX          413
## 5 VX          392
## 6 UA          364
## 7 AA          345
## 8 AA          334
## 9 VX          322
## 10 AA         321
## # i 11,252 more rows
```

4.6 Criando variáveis (colunas) com mutate()

Queremos criar uma variável que mostre a velocidade de cada voo:

```
flights %>%
  select(year:day, flight, distance, air_time) %>%
  mutate(speed = distance / (air_time / 60))
```

```
## # A tibble: 336,776 x 7
##   year month   day flight distance air_time speed
##   <int> <int> <int> <int>    <dbl>    <dbl> <dbl>
## 1  2013     1     1   1545    1400    227   370.
## 2  2013     1     1   1714    1416    227   374.
## 3  2013     1     1   1141    1089    160   408.
## 4  2013     1     1    725    1576    183   517.
## 5  2013     1     1    461     762    116   394.
## 6  2013     1     1   1696     719    150   288.
## 7  2013     1     1    507    1065    158   404.
## 8  2013     1     1   5708     229     53   259.
## 9  2013     1     1     79     944    140   405.
## 10 2013     1     1    301     733    138   319.
## # i 336,766 more rows
```

Note que você também pode se referir às variáveis que criou:

```
flights %>%
  select(year:day, flight, distance, air_time) %>%
  mutate(hours = air_time / 60,
         speed = distance / hours)
```

```
## # A tibble: 336,776 x 8
##   year month   day flight distance air_time hours speed
##   <int> <int> <int> <int>    <dbl>    <dbl> <dbl> <dbl>
## 1  2013     1     1   1545    1400    227  3.78   370.
## 2  2013     1     1   1714    1416    227  3.78   374.
## 3  2013     1     1   1141    1089    160  2.67   408.
## 4  2013     1     1    725    1576    183  3.05   517.
## 5  2013     1     1    461     762    116  1.93   394.
## 6  2013     1     1   1696     719    150  2.5    288.
## 7  2013     1     1    507    1065    158  2.63   404.
## 8  2013     1     1   5708     229     53  0.883  259.
## 9  2013     1     1     79     944    140  2.33   405.
## 10 2013     1     1    301     733    138  2.3    319.
## # i 336,766 more rows
```

4.7. SUMMARISE() COLAPSA A TABELA TODA EM APENAS UMA LINHA

4.7 summarise() colapsa a tabela toda em apenas uma linha

Vamos calcular o atraso médio de decolagem e a quantidade total de voos:

```
flights %>%  
  filter(!is.na(dep_delay)) %>%  
  summarise(mean_delay = mean(dep_delay), number_of_flights = n())
```

```
## # A tibble: 1 x 2  
##   mean_delay number_of_flights  
##   <dbl>         <int>  
## 1      12.6         328521
```

4.8 group_by() muda a “unidade de análise” de toda a tabela para os grupos definidos

Vamos calcular o atraso médio de decolagem e a quantidade total de voos, mas agora analisando por companhia aérea:

```
flights %>%  
  filter(!is.na(dep_delay)) %>%  
  group_by(carrier) %>%  
  summarise(mean_delay = mean(dep_delay), number_of_flights = n())
```

```
## # A tibble: 16 x 3  
##   carrier mean_delay number_of_flights  
##   <chr>         <dbl>         <int>  
## 1 9E           16.7           17416  
## 2 AA            8.59           32093  
## 3 AS            5.80             712  
## 4 B6           13.0           54169  
## 5 DL            9.26           47761  
## 6 EV           20.0           51356  
## 7 F9           20.2             682  
## 8 FL           18.7            3187  
## 9 HA            4.90             342  
## 10 MQ           10.6           25163  
## 11 OO           12.6             29  
## 12 UA           12.1           57979  
## 13 US            3.78           19873  
## 14 VX           12.9            5131
```

## 15 WN	17.7	12083
## 16 YV	19.0	545

4.8.1 DESAFIO:

Para cada destino, calcule: a (1) distância média dos voos e (2) o tempo de atraso médio na decolagem e (3) o número de voos na base e mostre tudo numa planilha só.

Chapter 5

ggplot

Em breve!

References

- Cotton, R. (2013). Learning R: a step-by-step function guide to data analysis. " O'Reilly Media, Inc."
- Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). R for Data Science. " O'Reilly Media, Inc."