Dataframes e outros Objetos

Análise de Dados Financeiros e Econômicos com o R

Marcelo S. Perlin

EA-UFRGS

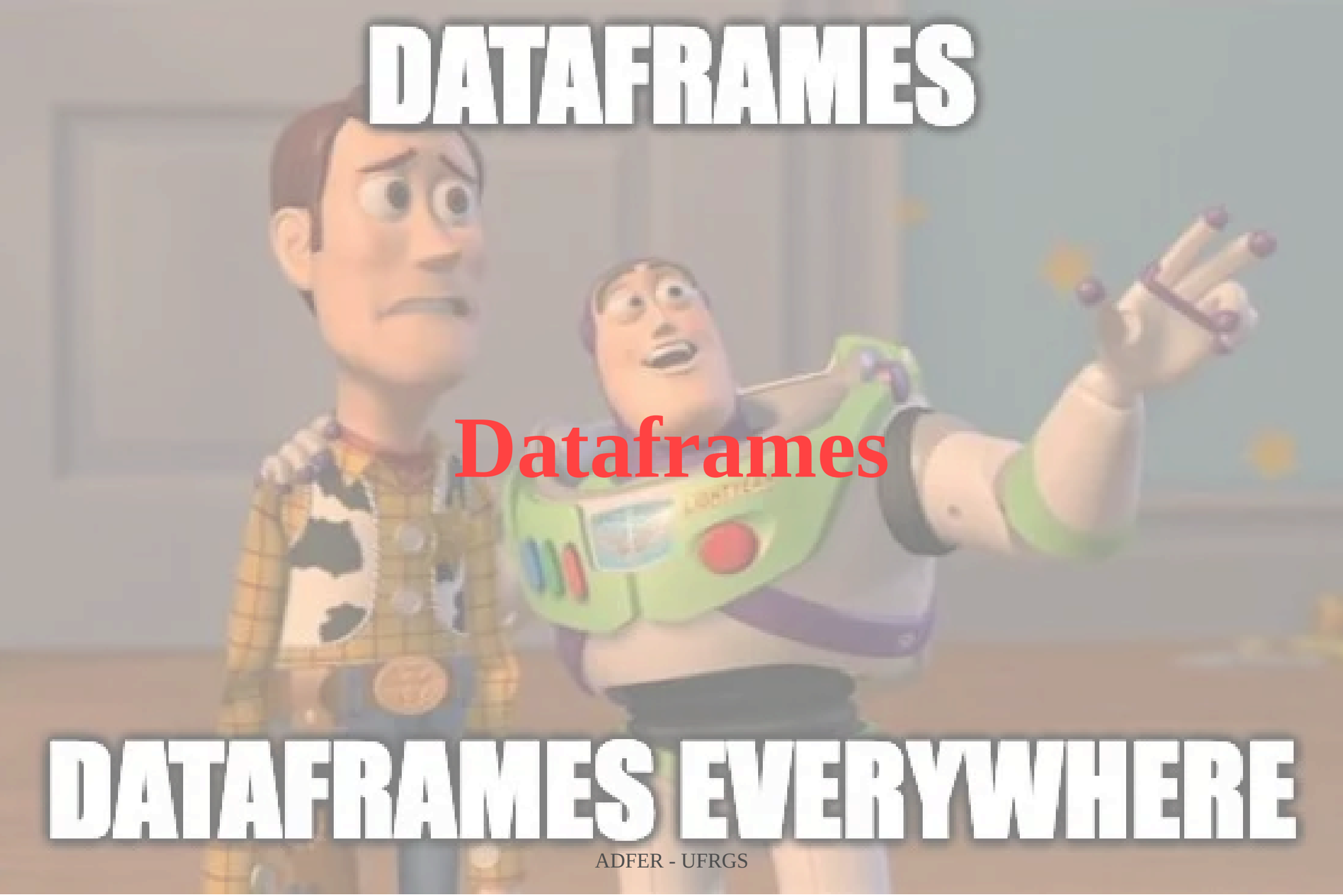
08/09/2026

Sumário

As Classes de Armazenamento no R

Introdução

- As classes básicas de objetos no R (números, texto, …) em vetores são difíceis de trabalhar com dados em larga escala (muitas observações, muitas colunas)
- Classes de armazenamento facilitam a agregação de objetos de forma estruturada
- `dataframes` e `lists` são os objetos mais usados no R



Dataframes

Introdução

> `dataframe` ou `tibble` significa “estrutura ou organização de dados”. Grosso modo, um objeto da classe dataframe nada mais é do que uma **tabela**, com linhas e colunas.

- **principal objeto utilizado no trabalho com o R**, sendo usado como tipo de entrada para diferentes funções
- `dataframe` é um tipo especial de lista, onde cada coluna é um vetor atômico com o mesmo número de elementos, porém com sua própria classe.
- organiza os dados, forçando o pareamento de variáveis
- altamente escalável para dados complexos

Criando `dataframes`

Exemplo simples

```
df <- tibble::tibble(
  x = 1:10,
  y = sample(letters, 10)
)

dplyr::glimpse(df)
```

```
Rows: 10
Columns: 2
$ x <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
$ y <chr> "y", "a", "q", "u", "j", "e", "r", "k", "m",
"p"
```

Exemplo menos simples

```
# set tickers
ticker <- c(rep('ABEV3',4),
```

```
             rep('BBAS3', 4),
             rep('BBDC3', 4))

# set dates
ref_date <- as.Date(rep(c('2010-01-01', '2010-01-04',
                          '2010-01-05', '2010-01-06'),
                        3) )

# set prices
price <- c(736.67, 764.14, 768.63, 776.47,
           59.4  , 59.8  , 59.2  , 59.28,
```

```
# A tibble: 12 × 3
   ticker ref_date   price
   <chr>  <date>     <dbl>
 1 ABEV3  2010-01-01 737.
 2 ABEV3  2010-01-04 764.
 3 ABEV3  2010-01-05 769.
 4 ABEV3  2010-01-06 776.
 5 BBAS3  2010-01-01  59.4
```

Inspecionando um `dataframe`

> **Atenção!**
>
> Inspecionar tabelas é uma das artes em análise de dados, exigindo intimidades com os dados e o problema em questão.

Estamos interessados em:

- Número de linhas e colunas
- Nomes das colunas
  - sem caracteres latinos, espaços ou até mesmo caixa alta
  - preferencialmente em inglês
- **Classes das colunas**
- **Existência de dados omissos (`NA`)**
- Verificar se dados **fazem sentido**

Função dplyr::glimpse()

```
1 library(dplyr)
2
3 # check content of my_df
4 glimpse(my_df)
```

```
Rows: 12
Columns: 3
$ ticker   <chr> "ABEV3", "ABEV3", "ABEV3", "ABEV3",
"BBAS3", "BBAS3", "BBAS3"…
$ ref_date <date> 2010-01-01, 2010-01-04, 2010-01-05,
2010-01-06, 2010-01-01, …
$ price    <dbl> 736.67, 764.14, 768.63, 776.47, 59.40,
59.80, 59.20, 59.28, 2…
```

Função `base::summary()`

```
1 # check variation my_df
2 summary(my_df)
```

```
      ticker       ref_date                price       
 Length   :12   Min.   :2010-01-01   Min.   : 29.81  
 N.unique : 3   1st Qu.:2010-01-03   1st Qu.: 30.71  
 N.blank  : 0   Median :2010-01-04   Median : 59.34  
 Min.nchar: 5   Mean   :2010-01-04   Mean   :283.73  
 Max.nchar: 5   3rd Qu.:2010-01-05   3rd Qu.:743.54  
                Max.   :2010-01-06   Max.   :776.47  
```

Função `skimr::skim()`

```
1  skimr::skim(my_df)
```

Data summary

Name	my_df
Number of rows	12
Number of columns	3
_____	
Column type frequency:	
character	1
Date	1
numeric	1
_____	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	wl
ticker	0	1	5	5	0	3	

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
ref_date	0	1	2010-01-01	2010-01-06	2010-01-04	4

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p
price	0	1	283.73	353.17	29.81	30.71	59.

Exemplo

Avalie os seguintes dados:

Data	id.candidato	Nome do candidato	Profissão	age	ano_casamento
08/09/2026	1	Y	professor	8	1999
08/09/2026	1	D	professor	-1	2010
08/09/2026	1	G	professor	9	1995
08/09/2026	1	A	casado	-4	2004
08/09/2026	1	B	advogado	4	2001
08/09/2026	1	K	casado	3	1998
08/09/2026	1	N	advogado	-5	1996
08/09/2026	1	R	advogado	-2	2008
08/09/2026	1	W	advogado	-3	2005
08/09/2026	1	J	advogado	0	1997

Problemas

- coluna `Data` em formato local (dd/mm/yyyy)
- coluna `id.candidato` tem somente valor 1
- coluna `Nome do Candidato` tem espaço no nome da coluna, e valores que não fazem sentido
- coluna `Profissão` tem caracter latino e valor “casado”, o qual não faz sentido
- coluna `age` está em inglês
- **ninguém com menos de 10 anos pode estar casado!!!!**

Acessando Elementos de um `dataframe`

- use sempre nomes para acessar colunas (via posição é possível, mas não recomendado)
- toda coluna de um *dataframe* vira um vetor atômico
- todo `dataframe` possui notação matricial para elementos indiviudais ou blocos

Acessando colunas inteiras

Existem duas principais maneiras de acessar uma coluna de um *dataframe* via nome:

```
# isolate columns of df
my_ticker <- my_df$ticker
my_prices <- my_df[['price']]

# print contents
print(my_ticker)
```

```
 [1] "ABEV3" "ABEV3" "ABEV3" "ABEV3" "BBAS3" "BBAS3"
"BBAS3" "BBAS3" "BBDC3"
[10] "BBDC3" "BBDC3" "BBDC3"
```

```
print(my_prices)
```

```
 [1] 736.67 764.14 768.63 776.47  59.40  59.80  59.20
59.28  29.81  30.82
[11]  30.38  30.20
```

Acessando valores específicos

```
1 # first element
2 my_price <- my_df$price[1]
3
4 print(my_price)
```

```
[1] 736.67
```

```
1 # first 10 elements
2 my_prices <- my_df$price[1:10]
3
4 print(my_prices)
```

```
 [1] 736.67 764.14 768.63 776.47  59.40  59.80  59.20
59.28  29.81  30.82
```

```
1 # first 10 rows of first and second column (returns a
2 my_prices <- my_df[1:10, 1:2]
3
4 print(my_prices)
```

```
# A tibble: 10 × 2
   ticker ref_date
   <chr>  <date>
 1 ABEV3  2010-01-01
 2 ABEV3  2010-01-04
 3 ABEV3  2010-01-05
 4 ABEV3  2010-01-06
 5 BBAS3  2010-01-01
 6 BBAS3  2010-01-04
 7 BBAS3  2010-01-05
 8 BBAS3  2010-01-06
```

Atenção!

Ao usar notação de matriz ([row, column]) em um *dataframe*, o resultado é sempre outro *dataframe*.

Operador de *pipeline*

Introdução

> O operador de *pipeline* é usado com o símbolos `|>` (nativo com R > 4.1) e `%>%` (pacote `magritt`) e permite que dataframes sejam criados em etapas consecutivas

Imagine o seguinte código, onde a ideia é repassar um *dataframe* entre diferentes e consecutivas funções:

```
1 df1 <- fct1(df0)
2 df2 <- fct2(df1)
3 df3 <- fct3(df2)
```

Uma maneira equivalente e mais elegante:

```
1 my_tab <- my_df |>
2   fct1() |>
3   fct2() |>
4   fct3()
```

Um exemplo prático

```r
df <- tibble::tibble(
  x = 1:10,
  y = runif(10)
)

fct1 <- function(df_in) {
  df_in$col1 <- 1
  return(df_in)
}

fct2 <- function(df_in) {
  df_in$col2 <- 2
  return(df_in)
```

```
Rows: 10
Columns: 5
$ x    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

```
$ y    <dbl> 0.33239467, 0.65087047, 0.25801678,
0.47854525, 0.76631067, 0.084…
$ col1 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
$ col2 <dbl> 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
$ col3 <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3
```

Modificando um `dataframe`

Para criar novas colunas em um `dataframe`, basta utilizar a função `dplyr::mutate`

```
1 library(dplyr)
2
3 # add columns with mutate
4 my_df <- my_df |>
5   mutate(ret = price/dplyr::lag(price) -1,
6          my_seq1 = 1:nrow(my_df),
7          my_seq2 =  my_seq1 +9) |>
8   glimpse()
```

```
Rows: 12
Columns: 6
$ ticker   <chr> "ABEV3", "ABEV3", "ABEV3", "ABEV3",
"BBAS3", "BBAS3", "BBAS3"…
$ ref_date <date> 2010-01-01, 2010-01-04, 2010-01-05,
2010-01-06, 2010-01-01, …
$ price    <dbl> 736.67, 764.14, 768.63, 776.47, 59.40,
```

```
59.80, 59.20, 59.28, 2…
$ ret       <dbl> NA, 0.037289424, 0.005875887,
0.010199966, -0.923499942, 0.00…
```

A maneira mais tradicional, e comumente encontrada em código, para criar novas colunas é utilizar o símbolo `$`:

```r
# add new column with base R
my_df$my_seq3 <- 1:nrow(my_df)

# check it
glimpse(my_df)
```

```
Rows: 12
Columns: 7
$ ticker   <chr> "ABEV3", "ABEV3", "ABEV3", "ABEV3",
"BBAS3", "BBAS3", "BBAS3"…
$ ref_date <date> 2010-01-01, 2010-01-04, 2010-01-05,
2010-01-06, 2010-01-01, …
$ price    <dbl> 736.67, 764.14, 768.63, 776.47, 59.40,
59.80, 59.20, 59.28, 2…
```

```
$ ret      <dbl> NA, 0.037289424, 0.005875887,
0.010199966, -0.923499942, 0.00…
```

Removendo colunas

Para remover colunas de um `dataframe`, basta usar `dplyr::select` com operador negativo para o nome das colunas indesejadas:

```
# removing columns
my_df_temp <- my_df |>
  select(-my_seq1, -my_seq2, -my_seq3) |>
  glimpse()
```

```
Rows: 12
Columns: 4
$ ticker   <chr> "ABEV3", "ABEV3", "ABEV3", "ABEV3",
"BBAS3", "BBAS3", "BBAS3"…
$ ref_date <date> 2010-01-01, 2010-01-04, 2010-01-05,
2010-01-06, 2010-01-01, …
$ price    <dbl> 736.67, 764.14, 768.63, 776.47, 59.40,
59.80, 59.20, 59.28, 2…
$ ret      <dbl> NA, 0.037289424, 0.005875887,
0.010199966, -0.923499942, 0.00…
```

No uso de funções nativas do R, a maneira tradicional de remover colunas é alocar o valor nulo (`NULL`):

```
# set temp df
temp_df <- my_df

# remove cols
temp_df$price <- NULL
temp_df$ref_date  <- NULL

# check it
glimpse(temp_df)
```

```
Rows: 12
Columns: 5
$ ticker  <chr> "ABEV3", "ABEV3", "ABEV3", "ABEV3",
"BBAS3", "BBAS3", "BBAS3",…
$ ret     <dbl> NA, 0.037289424, 0.005875887,
0.010199966, -0.923499942, 0.006…
$ my_seq1 <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
```

```
$ my_seq2 <dbl> 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21
$ my_seq3 <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
```

Filtrando linhas de um `dataframe`

- selecione linhas de um tabela com `dplyr::filter()`:

```
1 library(dplyr)
2
3 # filter df for single stock
4 df_temp1 <- my_df |>
5   filter(ticker == 'ABEV3') |>
6   glimpse()
```

```
Rows: 4
Columns: 7
$ ticker   <chr> "ABEV3", "ABEV3", "ABEV3", "ABEV3"
$ ref_date <date> 2010-01-01, 2010-01-04, 2010-01-05,
2010-01-06
$ price    <dbl> 736.67, 764.14, 768.63, 776.47
$ ret      <dbl> NA, 0.037289424, 0.005875887,
0.010199966
```

```
$ my_seq1  <int> 1, 2, 3, 4
$ my_seq2  <dbl> 10, 11, 12, 13
```

A função também aceita mais de uma condição. Veja a seguir onde filtramos os dados para `'ABEV3'` em datas após ou igual a `'2010-01-05'`:

```
1 library(dplyr)
2 # filter df for single stock and date
3 my_df_temp <- my_df |>
4   filter(ticker == 'ABEV3',
5          ref_date >= as.Date('2010-01-05')) |>
6   glimpse()
```

```
Rows: 2
Columns: 7
$ ticker   <chr> "ABEV3", "ABEV3"
$ ref_date <date> 2010-01-05, 2010-01-06
$ price    <dbl> 768.63, 776.47
$ ret      <dbl> 0.005875887, 0.010199966
$ my_seq1  <int> 3, 4
```

Ordenando um `dataframe`

No `tidyverse`, a forma de ordenar `dataframes` é pelo uso da função `arrange`. No caso de ordenamento decrescente, encapsulamos o nome das colunas com `desc`:

```
 1 library(dplyr)
 2
 3 # set df
 4 my_df <- tibble(col1 = c(4,1,2),
 5                 col2 = c(1,1,3),
 6                 col3 = c('a','b','c'))
 7
 8 # sort ascending, by col1 and col2
 9 my_df <- my_df |>
10   arrange(col1, col2) |>
11   print()
```

```
# A tibble: 3 × 3
   col1  col2 col3
  <dbl> <dbl> <chr>
```

```
1     1     1 b
2     2     3 c
3     4     1 a
```

```r
# sort descending, col1 and col2
my_df <- my_df |>
  arrange(desc(col1), desc(col2)) |>
  print()
```

```
# A tibble: 3 × 3
   col1  col2 col3
  <dbl> <dbl> <chr>
1     4     1 a
2     2     3 c
3     1     1 b
```

Combinando e Agregando `dataframes`

- podemos combinar tabelas por linhas (orientação vertical): `dplyr::bind_rows()`
- podemos combinar tabelas por colunas (orientação horizontal): `dlyr::bind_cols()`
- **agregar** tabelas de acordo com colunas: `dplyr::inner_join`, `dplyr::left_join`, `dplyr::right_join`, `dplyr::full_join`

Função `dplyr::bind_rows()`

Name	City	Country
Lenna	San Francisco	US
Malcom	New York	US
Akiko	Tokyo	Japan

+

Name	City	Country
Lenna	San Francisco	US
Thomas	London	UK
Diane	Chicago	US

Name	City	Country
Lenna	San Francisco	US
Malcom	New York	US
Akiko	Tokyo	Japan
Thomas	London	UK
Diane	Chicago	US

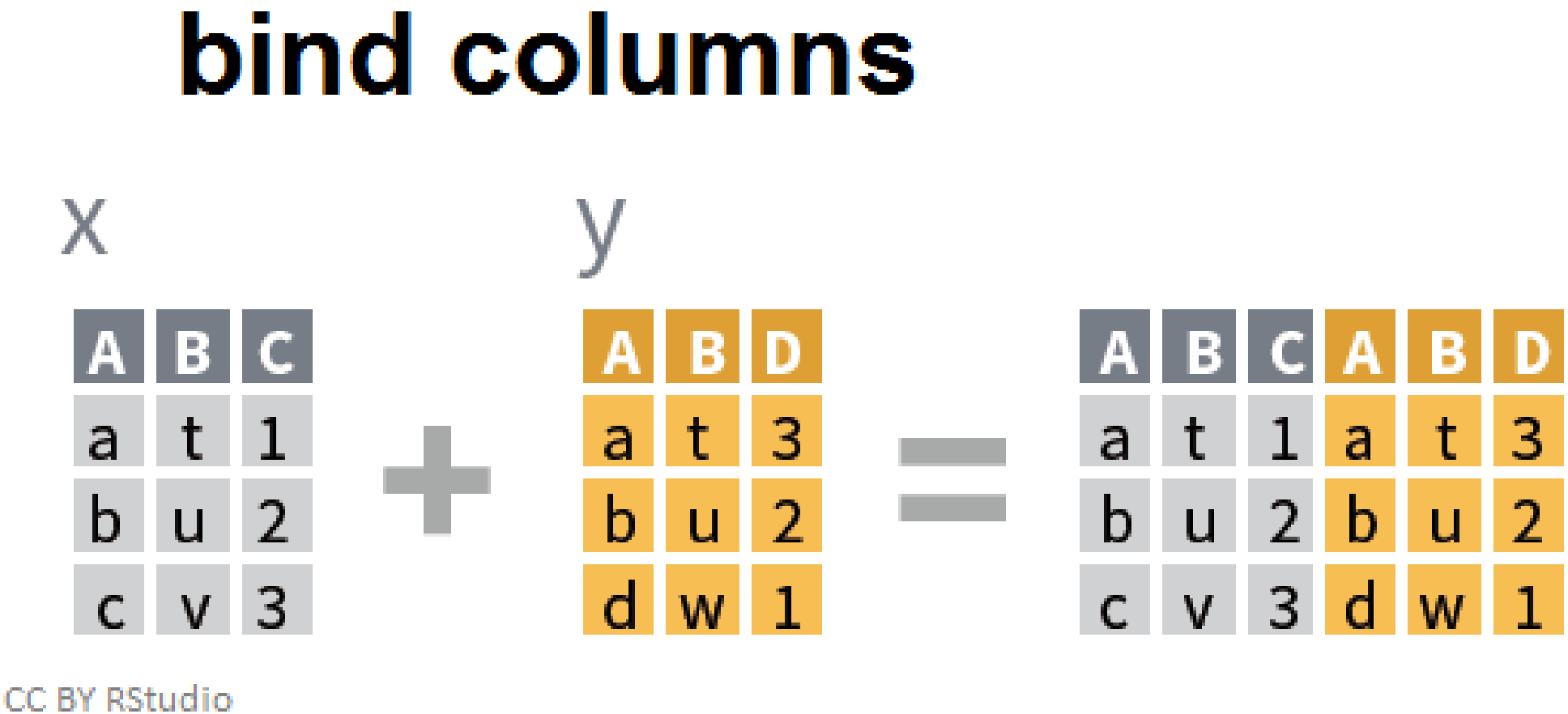
Exemplo de uso

```r
library(dplyr)

# set dfs
my_df_1 <- tibble(col1 = 1:5,
                  col2 = rep('a', 5))

my_df_2 <- tibble(col1 = 6:10,
                  col2 = rep('b', 5),
                  col3 = rep('c', 5))

# bind by row
my_df <- bind_rows(my_df_1, my_df_2) |>
  glimpse()
```

```
Rows: 10
Columns: 3
$ col1 <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

```
$ col2 <chr> "a", "a", "a", "a", "a", "b", "b", "b", "b",
"b"
$ col3 <chr> NA, NA, NA, NA, NA, "c", "c", "c", "c", "c"
```

Função `dplyr::bind_cols()`



Exemplo de uso

```r
# set dfs
my_df_1 <- tibble(col1 = 1:5, col2 = rep('a', 5))
my_df_2 <- tibble(col3 = 6:10, col4 = rep('b', 5))

# bind by column
my_df <- bind_cols(my_df_1, my_df_2) |>
  glimpse()
```

```
Rows: 5
Columns: 4
$ col1 <int> 1, 2, 3, 4, 5
$ col2 <chr> "a", "a", "a", "a", "a"
$ col3 <int> 6, 7, 8, 9, 10
$ col4 <chr> "b", "b", "b", "b", "b"
```

Funções da família `join`

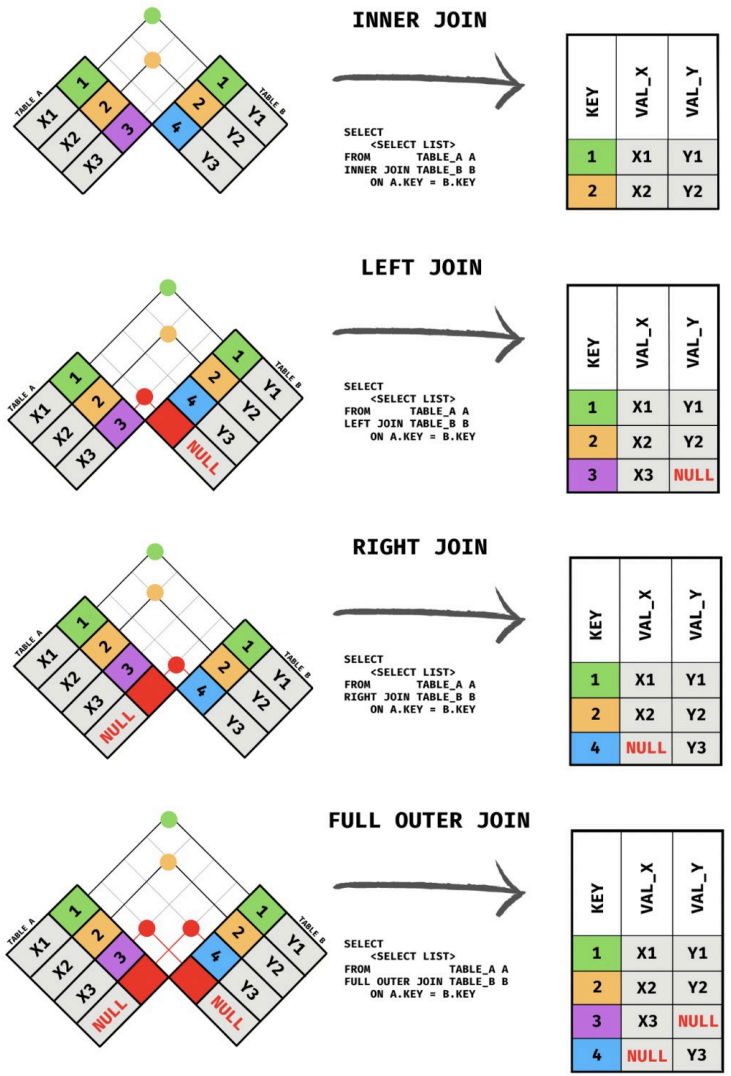


Image by author, inspired by R for Data Science

left_join()

Table 1

ID	y
A	5
B	5
C	8
D	0
F	9

Table 2

ID	z
A	30
B	21
C	22
D	25
E	29

Table 3

ID	y	z
A	5	30
B	5	21
C	8	22
D	0	25
F	9	NA

right_join()

Table 1

ID	y
A	5
B	5
C	8
D	0
F	9

Table 2

ID	z
A	30
B	21
C	22
D	25
E	29

Table 3

ID	y	z
A	5	30
B	5	21
C	8	22
D	0	25
E	NA	29

inner_join()

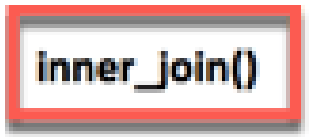


Table 1

ID	y
A	5
B	5
C	8
D	0
F	9

Table 2

ID	z
A	30
B	21
C	22
D	25
E	29

Table 3

ID	y	z
A	5	30
B	5	21
C	8	22
D	0	25

Exemplo `dplyr::inner_join()`

```r
# set df
my_df_1 <- tibble(date = as.Date('2016-01-01')+0:10,
                  x = 1:11)

my_df_2 <- tibble(date = as.Date('2016-01-05')+0:10,
                  y = seq(20,30, length.out = 11))

# aggregate tables
my_df <- inner_join(my_df_1, my_df_2)

glimpse(my_df)
```

```
Rows: 7
Columns: 3
$ date <date> 2016-01-05, 2016-01-06, 2016-01-07, 2016-
01-08, 2016-01-09, 2016…
```

Listas

Introdução

> Uma lista (`list`) é uma classe de objeto extremamente flexível. Ao contrário de vetores atômicos, a lista não apresenta restrição alguma em relação aos tipos de elementos nela contidos.

- um dos objetos mais utilizados no R
- acomoda informações complexas de forma fácil

Criando Listas

Uma lista pode ser criada através do comando `base::list`, seguido por seus elementos separados por vírgula:

```
 1 library(dplyr)
 2
 3 # create list
 4 my_l <- list(c(1, 2, 3),
 5               c('a', 'b'),
 6               factor('A', 'B', 'C'),
 7               tibble(col1 = 1:5))
 8
 9 # use base::print
10 print(my_l)
```

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


[[2]]
```

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

[[3]]
[1] <NA>
Levels: C
```

```
1 # use dplyr::glimpse
2 glimpse(my_l)
```

```
List of 4
 $ : num [1:3] 1 2 3
 $ : chr [1:2] "a" "b"
 $ : Factor w/ 1 level "C": NA
 $ : tibble [5 × 1] (S3: tbl_df/tbl/data.frame)
  ..$ col1: int [1:5] 1 2 3 4 5
```

Também podemos nomear os elementos:

```
1 # set named list
2 my_named_l <- list(ticker = 'TICK4',
3                    market = 'Bovespa',
```

```
                    df_prices = tibble(P = c(1,1.5,2,2.
                                       ref_date = Sys.D

# check content
glimpse(my_named_l)
```

```
List of 3
 $ ticker   : chr "TICK4"
 $ market   : chr "Bovespa"
 $ df_prices: tibble [4 × 2] (S3: tbl_df/tbl/data.frame)
  ..$ P        : num [1:4] 1 1.5 2 2.3
  ..$ ref_date: Date[1:4], format: "2026-09-08" "2026-09-
09" ...
```

Acessando os Elementos de uma Lista

Os elementos de uma lista podem ser acessados através do uso de duplo colchete ([[]]) ou $, tal como em:

```
1 # accessing elements from list
2 print(my_named_l[[2]]) # using position
```

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

```
1 print(my_named_l[["df_prices"]]) # using name
```

```
# A tibble: 4 × 2
      P ref_date
  <dbl> <date>
1   1   2026-09-08
2   1.5 2026-09-09
3   2   2026-09-10
4   2.3 2026-09-11
```

```
1 print(my_named_l$ticker) # using $
```

Adicionando e Removendo Elementos de uma Lista

Para adicionar ou substituir, basta definir um novo objeto na posição desejada da lista:

```
# set list
my_l <- list(slot1 = 'a',
             slot2 = 1:10)

# add new elements to list
my_l$slot3 <- c(1:5)
my_l$slot4 <- rep("b", 15)

# print result
glimpse(my_l)
```

```
List of 4
 $ slot1: chr "a"
```

Removendo elementos de uma lista

Para remover elementos de uma lista, basta definir o elemento para o símbolo reservado `NULL` (nulo):

```
# set list
my_l <- list(text = 'b',
             num1 = 2,
             num2 = 4)

# remove elements
my_l$num1 <- NULL
my_l$num2 <- NULL

glimpse(my_l)
```

```
List of 1
 $ text: chr "b"
```

Processando os Elementos de uma Lista

> Toda lista pode ser processado de forma programática através de funções específicas.

- processar elemento por elemento de uma lista

Por exemplo, imagine uma lista com vetores numéricos de diferentes tamanhos, tal como a seguir:

```
1 # set list
2 my_l_num <- list(c(1, 2, 3),
3                  seq(1:50),
4                  seq(-5, 5, by = 0.5))
```

Caso quiséssemos calcular a média de cada elemento de `my_l_num` e apresentar o resultado na tela como um vetor, poderíamos fazer isso através de um procedimento simples, processando cada elemento individualmente:

```r
# calculate mean of vectors
mean_1 <- mean(my_l_num[[1]])
mean_2 <- mean(my_l_num[[2]])
mean_3 <- mean(my_l_num[[3]])

# print it
print(c(mean_1, mean_2, mean_3))
```

```
[1]  2.0 25.5  0.0
```

O código anterior funciona, porém não é recomendado devido sua falta de escabilidade.

Uma maneira mais fácil, elegante e inteligente seria utilizar a função `sapply`:

```r
# using sapply
my_mean <- sapply(my_l_num, mean)

# print result
print(my_mean)
```

```
[1]  2.0 25.5  0.0
```

Matrizes

Introdução

> uma matriz é uma representação bidimensional de diversos valores, arranjados em linhas e colunas. O uso de matrizes é uma poderosa maneira de representar dados numéricos em duas dimensões e, em certos casos, funções matriciais podem simplificar operações matemáticas complexas.

Um claro exemplo do uso de matrizes em Finanças seria a representação dos preços de diferentes ações ao longo do tempo.

Data	ABEV3	BBAS3	BBDC3
2010-01-01	736.67	59.4	29.81
2010-01-04	764.14	59.8	30.82
2010-01-05	768.63	59.2	30.38
2010-01-06	776.47	59.28	30.20

A matriz anterior poderia ser criada da seguinte forma no R:

```r
# create matrix
data <- c(736.67, 764.14, 768.63, 776.47,
          59.4, 59.8, 59.2, 59.28,
          29.81, 30.82, 30.38, 30.20)

my_mat <- matrix(data, nrow = 4, ncol = 3)

# set names of cols and rows
colnames(my_mat) <- c('ABEV3', 'BBAS3', 'BBDC3')
rownames(my_mat) <- c('2010-01-01', '2010-01-04',
                      '2010-01-05', '2010-01-06')

print(my_mat)
```

```
            ABEV3 BBAS3 BBDC3
2010-01-01 736.67 59.40 29.81
2010-01-04 764.14 59.80 30.82
2010-01-05 768.63 59.20 30.38
2010-01-06 776.47 59.28 30.20
```

Selecionando Valores de uma Matriz

- Os elementos de uma matriz podem ser acessados pela notação `[i,j]`, onde *i* representa a linha e *j* a coluna. Veja o exemplo a seguir:

```
1 my_mat <- matrix(1:9, nrow = 3)
2 print(my_mat)
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
1 print(my_mat[1,2])
```

```
[1] 4
```

O uso de testes lógicos para selecionar valores de matrizes também é possível. Veja a seguir:

```
1 my_mat <- matrix(1:9, nrow = 3)
```

```
2 print(my_mat >5)
```

```
      [,1]  [,2] [,3]
[1,] FALSE FALSE TRUE
[2,] FALSE FALSE TRUE
[3,] FALSE  TRUE TRUE
```

Tarefa final de aula

Tarefa 03

1. Crie um dataframe chamado `my_df` com uma coluna chamada `x` contendo uma sequência de 1 a 100 e outra coluna chamada `y` com o valor da coluna x adicionada de 5. Usando código, qual a quantidade de valores em `x` maiores que 10 e menores que 25?
2. Use função `dplyr::mutate` para criar uma nova coluna em `my_df` chamada `cumsum_x`, contendo a soma cumulativa de `x` (função `base::cumsum()`). Desta nova coluna, quantos valores são maiores que 50?
3. Com função `dplyr::filter()`, filtre o dataframe `my_df` para manter apenas as linhas onde o valor da coluna `x` é maior que 25.
4. Utilize pacote `yfR` para baixar dados da ação do META/Facebook (META) desde a data `'2010-01-01'` até hoje. Filtre os dados para mentar apenas dados onde o dia da semana era segunda feira.
5. Importe os dados disponíveis no arquivo `CH11_grunfeld.csv`, disponível no arquivo com os dados do livro. Utilize funções `dplyr::glimpse`, `summary` e `skimr::skim` para conhecer os dados importados.
6. Crie um objeto do tipo lista com três dataframes em seu conteúdo, `df1`, `df2` e `df3`. O conteúdo e tamanho dos dataframes é livre. Utilize função `sapply` para descobrir o número de linhas e colunas em cada dataframe.

o número de linhas e colunas em cada dataframe.

7. Usando pacote `yfR`, baixe os dados da Petrobras (PETR4.SA) e do índice ibovespa (^BVSP) em dois dataframes diferentes (duas chamadas da função `yfR::yf_get`). Empilhe os dados das duas ações com o comando `dplyr::bind_rows`.

8. Usando os mesmos dados do exercício anterior, crie uma nova coluna na tabela da Petrobras com os retornos do índice ibovespa. A tabela resultante deve ser como a apresentada a seguir:

ref_date	ret_petr3	ret_ibov
2026-09-09	100.8398	100.3254
2026-09-10	100.8112	101.0424
2026-09-11	101.7023	101.2851
2026-09-12	102.5667	101.9963
2026-09-13	102.9567	102.1179
2026-09-14	103.7340	102.3634
2026-09-15	104.6946	102.5067
2026-09-16	105.1293	102.7464
2026-09-17	105.8418	102.8053
2026-09-18	106.2418	103.4476

Note que precisarás:

- renomear e selecionar colunas da tabela da Petrobras

Referências