

MBA Business Analytics e Big Data

Análise Exploratória de Dados

Prof. Dr. João Rafael Dias

2º semestre - 2022

Manipulação de *strings* com `stringr`
Manipulação de datas com `lubridate`
Manipulação de *dataframes* com `dplyr`
Prática no RStudio

Técnicas de visualização
Tipos de gráficos
Análise univariada
Análise bivariada
Prática no RStudio

Apresentação do curso
Introdução ao R/Rstudio
Estrutura de dados
Comando básicos
Leitura e escrita de dados
Prática no RStudio

Tipos de variáveis
Medidas de Centralidade
Medidas de Dispersão
Medidas de Associação
Prática no RStudio

Trabalhos práticos
Aplicação do conteúdo

Manipulação de *strings*

- Em R, não há diferenciação entre *strings* e caracteres; um pedaço de texto é representado por uma sequência de caracteres (letras, números, e símbolos). Uma *string* é uma variável caractere que pode possuir um ou mais caracteres.
- Os usos mais comuns para *strings* em R são: nomes de arquivos e diretórios, nomes de elementos em objetos de dados, elementos de texto em figuras e gráficos.
- Em R as *strings* estão sempre entre pares de “ “ ou ‘ ‘
- O base R possui diversas funções, porém elas podem ser inconsistentes. Para isso usaremos o pacote *stringr* (documentação em: <https://cloud.r-project.org/web/packages/stringr/stringr.pdf>)

Acesso

```
# instalando o stringr
install.packages('stringr')

# carregand 'stringr'
library(stringr)
```



Se isso acontecer,
pressione a tecla Esc
e faça a correção,
fechando com as
aspas faltantes

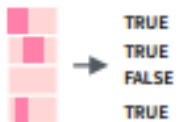
Exemplo

```
> string1 <- 'Esta é uma string!'
> string1
[1] "Esta é uma string!"
>
> string2 <- "Esta também é"
> string2
[1] "Esta também é"
>
> string3 <- 'Esta não é
+
+
+ HELP! :O
```

Detecção de correspondência

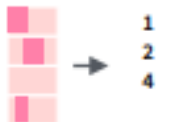
- Para determinar a correspondência de um padrão específico em uma *string*, existe uma variedade de comandos que detectam ou retornam a localização desses padrões.

Detect Matches



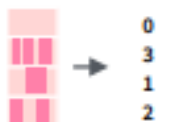
TRUE
TRUE
FALSE
TRUE

str_detect(string, **pattern**) Detect the presence of a pattern match in a string.
`str_detect(fruit, "a")`



1
2
4

str_which(string, **pattern**) Find the indexes of strings that contain a pattern match.
`str_which(fruit, "a")`



0
3
1
2

str_count(string, **pattern**) Count the number of matches in a string.
`str_count(fruit, "a")`



start end
2 4
4 7
NA NA
3 4

str_locate(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all**. `str_locate(fruit, "a")`

Exemplo

```
> string <- c('apple','banana','orange', 'kiwi')
> class(string)
[1] "character"
>
> str_detect(string, 'a')
[1] TRUE TRUE TRUE FALSE
> string[str_detect(string, 'a')] # subsetting do vetor
[1] "apple" "banana" "orange"
>
> str_which(string, 'e')
[1] 1 3
>
> str_count(string, 'w')
[1] 0 0 0 1
>
> str_locate(string, 'an')
      start end
[1,]    NA  NA
[2,]     2   3
[3,]     3   4
[4,]    NA  NA
```

- Para possibilitar a extração de uma parte específica da *string*, existem comandos específicos que podem usar tanto uma posição específica quanto a um padrão determinado.

Subset Strings



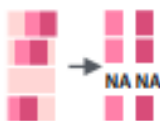
str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector.
`str_sub(fruit, 1, 3); str_sub(fruit, -2)`



str_subset(string, **pattern**) Return only the strings that contain a pattern match.
`str_subset(fruit, "b")`



str_extract(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. `str_extract(fruit, "[aeiou]")`



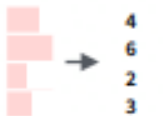
str_match(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each () group in pattern. Also **str_match_all**.
`str_match(sentences, "(a|the) ([^]+)")`

Exemplo

```
> string <- c('inferencia','estatistica','analise')
>
> str_sub(string,3,5) # posicao 3 à 5
[1] "fer" "tat" "ali"
>
> str_sub(string,-3) # ultimas 3 letras
[1] "cia" "ica" "ise"
>
> str_subset(string,'tis')
[1] "estatistica"
>
> str_extract(string,'er')
[1] "er" NA NA
>
> str_match(string,'er')
      [,1]
[1,] "er"
[2,] NA
[3,] NA
```

- Algumas funções permitem a contagem de caracteres de uma *string*, truncar o comprimento delas ou simplesmente remover espaços em branco.

Manage Lengths



str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*



str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*



str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*



str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

Exemplo

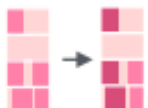
```
> string <- c('inferencia','estatística','analise')
>
> str_length(string)                                # comprimento
[1] 10 11 7
>
> str_pad(string,15)                                # mesmo tamanho
[1] "      inferencia" "      estatística" "      analise"
>
> str_pad(string,15, side = 'right') # mesmo tamanho
[1] "inferencia      " "estatística      " "analise          "
>
> str_trunc(string,6)
[1] "inf..." "est..." "ana..."
>
> string2 <- ' big data '                            # string c espaços
>
> str_trim(string2,side = 'left')
[1] "big data  "
>
> str_trim(string2,side = 'both')
[1] "big data"
```

- Podem-se ainda fazer alterações mais profundas nas *strings*.

Mutate Strings



str_sub() <- value. Replace substrings by identifying the substrings with `str_sub()` and assigning into the results.
`str_sub(fruit, 1, 3) <- "str"`



str_replace()(string, **pattern**, replacement) Replace the first matched pattern in each string. `str_replace(fruit, "a", "-")`



str_replace_all()(string, **pattern**, replacement) Replace all matched patterns in each string. `str_replace_all(fruit, "a", "-")`

A STRING
↓
a string

str_to_lower()(string, locale = "en")¹ Convert strings to lower case.
`str_to_lower(sentences)`

a string
↓
A STRING

str_to_upper()(string, locale = "en")¹ Convert strings to upper case.
`str_to_upper(sentences)`

a string
↓
A String

str_to_title()(string, locale = "en")¹ Convert strings to title case. `str_to_title(sentences)`

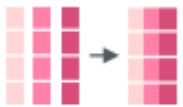
Exemplo

```
> string <- c('jsmith@domain.com')
>
> str_sub(string, 8, 13) <- 'fgv'
>
> string
[1] "jsmith@fgv.com"
>
> string <- str_replace(string, 'com', 'com.br')
>
> string
[1] "jsmith@fgv.com.br"
>
> string2 <- c('Analytics is cool!')
>
> str_to_lower(string2)      # tudo minusculo
[1] "analytics is cool!"
>
> str_to_upper(string2)     # tudo maiusculo
[1] "ANALYTICS IS COOL!"
>
> str_to_sentence(string2)  # apenas a primeira letra
[1] "Analytics is cool!"
```


Junções e quebras

- Existem também funções que permite concatenar, colapsar, preencher ou ainda particionar *strings*.
- Esses comandos são bastante usados em manipulação de partes de *strings*.

Join and Split



str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string.
`str_c(letters, LETTERS)`



str_c(..., sep = "", collapse = "") Collapse a vector of strings into a single string.
`str_c(letters, collapse = "")`



str_dup(string, times) Repeat strings times times. `str_dup(fruit, times = 2)`



str_split_fixed(string, pattern, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings.
`str_split_fixed(fruit, " ", n=2)`



str_glue(..., .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. `str_glue("Pi is {pi}")`

Exemplo

```
> str_c('May','the','force','be','with','you', sep = '')
[1] "Maytheforcebewithyou"
>
> str_c('May','the','force','be','with','you', sep = '_')
[1] "May_the_force_be_with_you"
>
> string <- c('May','the','force','be','with','you')
>
> str_c(string, sep = "", collapse = '')
[1] "Maytheforcebewithyou"
>
> str_dup('hola',2) # duplicando
[1] "holahola"
>
> str_dup('hola',1:3) # sequencia x1, x2, x3
[1] "hola" "holahola" "holaholahola"
>
> fruits <- c('apple','banana','apricot','kiwi')
>
> str_split_fixed(fruits,'a',n = 3)
      [,1] [,2] [,3]
[1,] ""   "pple" ""
[2,] "b"  "n"  "na"
[3,] ""   "pricot" ""
[4,] "kiwi" "" ""
```

Manipulação de *strings*

Expressões regulares [extra]

- Regexps* é uma linguagem concisa para descrever padrões em uma *string*. É possível pegar um vetor e uma expressão regular e mostrar a correspondência entre elas.

Regexp

<https://www.rstudio.com/resources/cheatsheets/>

| | |
|---------------------|-------------|
| [:alnum:] | |
| [:digit:] | |
| 0 1 2 3 4 5 6 7 8 9 | |
| [:alpha:] | |
| [:lower:] | [:upper:] |
| a b c d e f | A B C D E F |
| g h i j k l | G H I J K L |
| m n o p q r | M N O P Q R |
| s t u v w x | S T U V W X |
| z | Z |

| | |
|------------|-------|
| [:space:] | |
| ↵ new line | |
| [:blank:] | |
| □ | space |
| □ | tab |

| | |
|--------------------------------|--|
| [:graph:] | |
| [:punct:] | |
| . , : ; ? ! \ / ` = * + - ^ | |
| _ ~ " ' [] { } () < > @ # \$ | |

Regular Expressions -

Regular expressions, or *regexps*, are a concise language for describing patterns in strings.

```
see <- function(rx) str_view_all("abc ABC 123\t.!?\n()\n", rx)
```

MATCH CHARACTERS

| string (type this) | regex (to mean this) | matches (which matches this) | example |
|--------------------|----------------------|---------------------------------------------|------------------|
| | a (etc.) | a (etc.) | see("a") |
| \\. | \\. (etc.) | . | see("\\.") |
| \\! | \\! | ! | see("\\!") |
| \\? | \\? | ? | see("\\?") |
| \\\\ | \\\\ | \\ | see("\\\\") |
| \\(| \\(| (| see("\\(") |
| \\) | \\) |) | see("\\)") |
| \\{ | \\{ | { | see("\\{") |
| \\} | \\} | } | see("\\}") |
| \\n | \\n | new line (return) | see("\\n") |
| \\t | \\t | tab | see("\\t") |
| \\s | \\s | any whitespace (\\S for non-whitespaces) | see("\\s") |
| \\d | \\d | any digit (\\D for non-digits) | see("\\d") |
| \\w | \\w | any word character (\\W for non-word chars) | see("\\w") |
| \\b | \\b | word boundaries | see("\\b") |
| | [:digit:] | digits | see("[:digit:]") |
| | [:alpha:] | letters | see("[:alpha:]") |
| | [:lower:] | lowercase letters | see("[:lower:]") |
| | [:upper:] | uppercase letters | see("[:upper:]") |
| | [:alnum:] | letters and numbers | see("[:alnum:]") |
| | [:punct:] | punctuation | see("[:punct:]") |
| | [:graph:] | letters, numbers, and punctuation | see("[:graph:]") |
| | [:space:] | space characters (i.e. \\s) | see("[:space:]") |
| | [:blank:] | space and tab (but not new line) | see("[:blank:]") |
| | . | every character except a new line | see(".") |

Manipulação de datas

- Em R é possível tratar e trabalhar dados que são relacionados a data e tempo.
- Bases de dados com variáveis do tipo data-tempo são muito comuns, e em muitas situações precisamos transformar e extrair datas, trabalhar com fuso horário e fazer operações aritmética com datas (intervalos de tempo, etc)
- Quando importamos uma base de dados no R, informação de datas e tempo são comumente armazenadas como caracteres ou *factors* dado ao fato de possuírem símbolos como "-", ":" ou "/"
- Para trabalhar com datas usaremos o pacote `lubridate` (documentação em: <https://cloud.r-project.org/web/packages/lubridate/lubridate.pdf>)

Exemplo

```
# instalando o lubridate
install.packages('lubridate')

# carregand 'lubridate'
library(lubridate)
```



Perceba que o R mostra ambos valores entre aspas porém apenas um é uma *string*

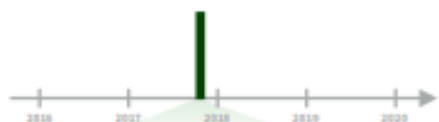
Exemplo

```
> data_string <- "21-10-2015"
> class(data_string)
[1] "character"
> data_string
[1] "21-10-2015"
>
> data_mdy <- dmy(data_string)
> class(data_mdy)
[1] "Date"
> data_mdy
[1] "2015-10-21"
```

Data - Tempo

- O dado quando convertido em data é armazenado como um número no R, relativo à uma data de referência (cada linguagem de programação tem a sua referência).
- Para formatar data, o pacote fornece algumas funções que são permutações das letra “y”, “m” e “d” para representar a ordem do ano, mês e dia.

Date-times



2017-11-28 12:00:00

2017-11-28 12:00:00

A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

```
dt <- as_datetime(1511870400)
## "2017-11-28 12:00:00 UTC"
```

2017-11-28

A **date** is a day stored as the number of days since 1970-01-01

```
d <- as_date(17498)
## "2017-11-28"
```

12:00:00

An **hms** is a **time** stored as the number of seconds since 00:00:00

```
t <- hms::as_hms(85)
## 00:01:25
```

Exemplo

```
> data <- '2020/04/26 10:03:20' # string
> data
[1] "2020/04/26 10:03:20"
> data <- as_datetime(data)      # data (UTC)
> data
[1] "2020-04-26 10:03:20 UTC"
>
> as_datetime(1587895400) # n segs desde 01-01-1970
[1] "2020-04-26 10:03:20 UTC"
>
> as_date(18378)           # n dias desde 01-01-1970
[1] "2020-04-26"
>
> hms::as_hms(35)         # n segs
00:00:35
```


Convertendo *strings* ou números em data-tempo

- Para realizar a conversão de forma correta, primeiro é necessário identificar a ordem dos elementos de ano (y), mês (m), dia (d), hora (h), minuto (m) e segundo (s) da base de dados.
- A partir da ordem identificada, utilize uma das funções abaixo que replique a ordem.

```
.....  
2017-11-28T14:02:00 ymd_hms(), ymd_hm(), ymd_h().  
                    ymd_hms("2017-11-28T14:02:00")  
  
2017-22-12 10:00:00 ydm_hms(), ydm_hm(), ydm_h().  
                    ydm_hms("2017-22-12 10:00:00")  
  
11/28/2017 1:02:03 mdy_hms(), mdy_hm(), mdy_h().  
                    mdy_hms("11/28/2017 1:02:03")  
  
1 Jan 2017 23:59:59 dmy_hms(), dmy_hm(), dmy_h().  
                    dmy_hms("1 Jan 2017 23:59:59")  
  
20170131 ymd(), ydm(). ymd(20170131)  
  
July 4th, 2000 mdy(), myd(). mdy("July 4th, 2000")  
  
4th of July '99 dmy(), dym(). dmy("4th of July '99")  
  
2001: Q3 yq() Q for quarter. yq("2001: Q3")  
  
2:01 hms::hms() Also lubridate::hms(),  
      hm() and ms(), which return  
      periods.* hms::hms(sec = 0, min= 1,  
      hours = 2)
```

Exemplo

```
> ymd_hms('2020/04/26 10:03:20') # yyyymmdd hhmmss  
[1] "2020-04-26 10:03:20 UTC"  
>  
> ymd_hm('2020/04/26 10:03')      # yyyymmdd hhmm  
[1] "2020-04-26 10:03:00 UTC"  
>  
> ymd_h('2020/04/26 10')          # yyyymmdd hh  
[1] "2020-04-26 10:00:00 UTC"  
>  
> ydm_hms('2020-26-04 10:03:20') # yyyyddmm hhmmss  
[1] "2020-04-26 10:03:20 UTC"  
>  
> mdy_hms('04/26/2020 10:03:20') # mmddyyyy hhmmss  
[1] "2020-04-26 10:03:20 UTC"  
>  
> ymd(20200426)                   # yyyymmdd  
[1] "2020-04-26"  
>  
> mdy('April 26th, 2020')         # Month Day, Year  
[1] "2020-04-26"  
>
```

Obter e configurar componentes

- Existem funções de acesso para obter componentes de data e tempo e alterá-las.

.....

| | |
|---------------------|--------------------------------------------------------------------------------------------------------------------------|
| 2018-01-31 11:59:59 | date(x) Date component. <i>date(dt)</i> |
| 2018-01-31 11:59:59 | year(x) Year. <i>year(dt)</i> isoyear(x) The ISO 8601 year. epiyear(x) Epidemiological year. |
| 2018-01-31 11:59:59 | month(x, label, abbr) Month. <i>month(dt)</i> |
| 2018-01-31 11:59:59 | day(x) Day of month. <i>day(dt)</i> wday(x, label, abbr) Day of week. qday(x) Day of quarter. |
| 2018-01-31 11:59:59 | hour(x) Hour. <i>hour(dt)</i> |
| 2018-01-31 11:59:59 | minute(x) Minutes. <i>minute(dt)</i> |
| 2018-01-31 11:59:59 | second(x) Seconds. <i>second(dt)</i> |

Exemplo

```
> dt <- as_datetime('2020-04-26 10:03:20')
> dt
[1] "2020-04-26 10:03:20 UTC"
>
> year(dt)                                # apenas o ano
[1] 2020
>
> month(dt, label = TRUE) # mês com label
[1] abr
12 Levels: jan < fev < mar < abr < mai < jun < jul < ago < ... < dez
>
> wday(dt)                                # dia da semana numerico
[1] 1
>
> wday(dt, label = T, abbr = F, locale = 'English')
[1] Sunday
7 Levels: Sunday < Monday < Tuesday < Wednesday < ... < Saturday
>
> hour(dt)                                # hora
[1] 10
>
> minute(dt)                              # minuto
[1] 3
>
> year(dt) <- 2019                        # alterando o ano
> month(dt) <- 9                          # alterando o mês
> day(dt) <- 12                           # alterando o dia
```

Obter e configurar componentes

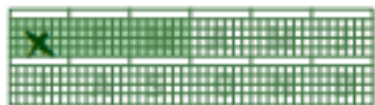
- Existem funções de acesso para obter componentes de data e tempo e alterá-las.



week(x) Week of the year. *week(dt)*

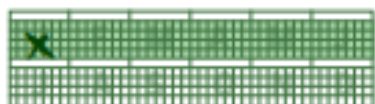
isoweek() ISO 8601 week.

epiweek() Epidemiological week.



quarter(x, with_year = FALSE)

Quarter. *quarter(dt)*



semester(x, with_year = FALSE)

Semester. *semester(dt)*



am(x) Is it in the am? *am(dt)*

pm(x) Is it in the pm? *pm(dt)*

dst(x) Is it daylight savings? *dst(d)*

leap_year(x) Is it a leap year?

leap_year(d)

update(object, ..., simple = FALSE)

update(dt, mday = 2, hour = 1)

Exemplo

```
> dt <- as_datetime('2020-04-26 10:03:20')
> dt
[1] "2020-04-26 10:03:20 UTC"
>
> week(dt)                                # no. da semana
[1] 17
>
> quarter(dt)                             # trimestre
[1] 2
> quarter(dt, with_year = T) # trimestre com ano
[1] 2020.2
>
> semester(dt)                            # semestre
[1] 1
> semester(dt, with_year = T) # semestre com ano
[1] 2020.1
>
> am(dt)                                  # ante meridiem < 12h?
[1] TRUE
> pm(dt)                                  # post meridiem > 12h?
[1] FALSE
>
> dst(dt)                                 # daylight saving?
[1] FALSE
>
> leap_year(dt)                           # ano bissexto?
[1] TRUE
```

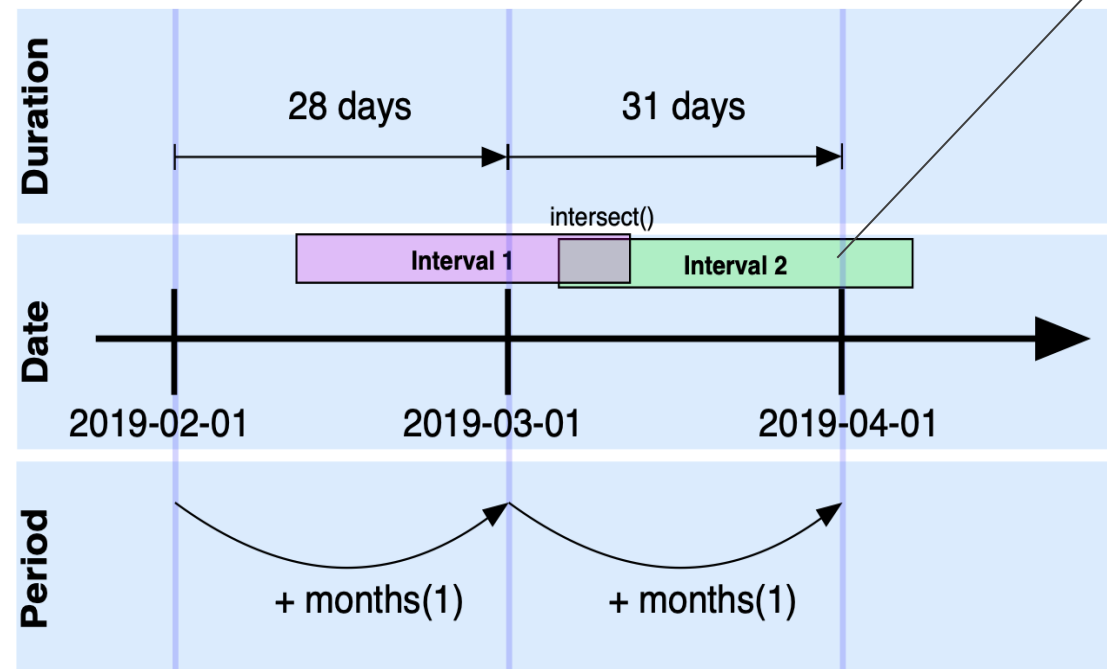

Manipulação de datas

Aritmética com datas-tempo

- Matemática com data e tempo baseiam-se em linha do tempo, a qual se comporta de forma inconsistente (um dia não possui 24 horas exatas, existem anos bissextos, etc.).
- O pacote `lubridate` possui três classes de intervalo de tempo para facilitar as operações: período, duração e intervalo.

Acompanha a passagem do tempo físico, o qual se difere do tempo do relógio quando irregularidades acontecem. Representa a duração do tempo medida em segundos.

Acompanha as mudanças nos horários, o qual ignora irregularidade na linha do tempo. Representa um período de tempo em termos de campos, por exemplo 3 anos 5 meses 2 dias e 7 horas. A duração em segundos depende do mês e se o ano é bissexto



Representa intervalos específicos na linha do tempo, limitados por um início e de fim. É o intervalo de tempo entre o instante de um segundo a outro instante.

<https://towardsdatascience.com/its-merely-a-matter-of-time-dr-watson-2fd74a648842>

Períodos

- Adicione ou subtraia duração para modelar eventos que acontecem em um tempo específico do relógio/calendário.

Make a period with the name of a time unit *pluralized*, e.g.

```
p <- months(3) + days(12)
p
"3m 12d 0H 0M 0S"
```

Number
of months

Number
of days

etc.

years(x = 1) x years.

months(x) x months.

weeks(x = 1) x weeks.

days(x = 1) x days.

hours(x = 1) x hours.

minutes(x = 1) x minutes.

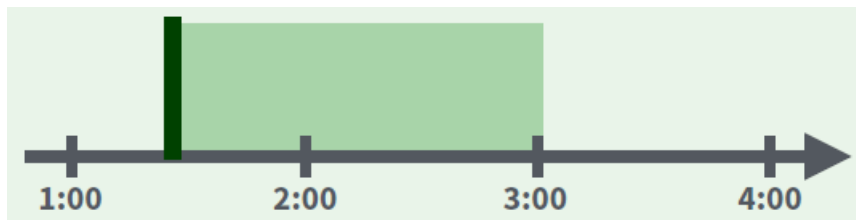
seconds(x = 1) x seconds.

milliseconds(x = 1) x milliseconds.

microseconds(x = 1) x microseconds.

nanoseconds(x = 1) x nanoseconds.

plcoseconds(x = 1) x picoseconds.



Exemplo

```
> p <- months(5) + days(1) # somar 5 meses com 1 dia
> p
[1] "5m 1d 0H 0M 0S"
> p*10 # resultado x 10
[1] "50m 10d 0H 0M 0S"
>
> dt <- as_datetime('2020/04/26 10:03:20')
> dt
[1] "2020-04-26 10:03:20 UTC"
>
> dt + days(1) # date time + 1 dia
[1] "2020-04-27 10:03:20 UTC"
>
> dt + months(20) # date time + 20 meses
[1] "2021-12-26 10:03:20 UTC"
>
> dt + weeks(3) + hours(2) # + 3 semanas + 2 horas
[1] "2020-05-17 12:03:20 UTC"
>
> months(1:4) # vetor de qtd de meses
[1] "1m 0d 0H 0M 0S" "2m 0d 0H 0M 0S" "3m 0d 0H 0M 0S" "4m 0d 0H 0M 0S"
>
> hours(c(12, 24)) # vetor de qtde de horas
[1] "12H 0M 0S" "24H 0M 0S"
```

Duração

- Adicione ou subtraia durações para modelar processos físicos. São armazenados em segundos, que é a única unidade consistente.

Make a duration with the name of a period prefixed with a *d*, e.g.

```
dd <- ddays(14)
dd
"1209600s (~2 weeks)"
```

Exact
length in
seconds

Equivalent
in common
units

dyears(x = 1) 31536000x seconds.

dweeks(x = 1) 604800x seconds.

ddays(x = 1) 86400x seconds.

dhours(x = 1) 3600x seconds.

dminutes(x = 1) 60x seconds.

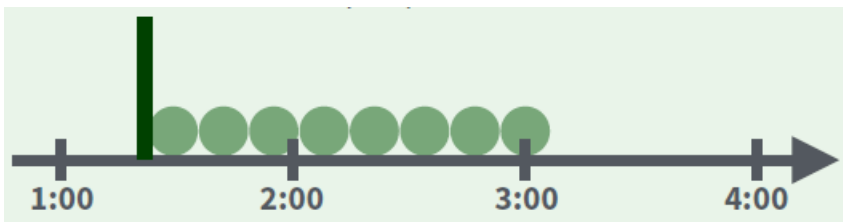
dseconds(x = 1) x seconds.

dmilliseconds(x = 1) $x \times 10^{-3}$ seconds.

dmicroseconds(x = 1) $x \times 10^{-6}$ seconds.

dnanoseconds(x = 1) $x \times 10^{-9}$ seconds.

dpicoseconds(x = 1) $x \times 10^{-12}$ seconds.



Exemplo

```
> p <- dyears(1)           # duracao de um ano segs
> p
[1] "31557600s (~1 years)"
>
> p <- p + dweeks(12)      # somando 12 semanas (seg)
> p
[1] "38815200s (~1.23 years)"
>
> a <- 10*ddays(1)         # 1 dia - 86400 segs
> a                        # x10 ?
[1] "864000s (~1.43 weeks)"
>
> dminutes(1:4)           # duracaoem segs 1,2,3,4 mins
[1] "60s (~1 minutes)"  "120s (~2 minutes)" "180s (~3 minutes)"
[4] "240s (~4 minutes)"
>
> duration(1)              # duracao 1 seg em segundos
[1] "1s"
> duration(1, units = 'months') # duracao 1 mês em segundos
[1] "2629800s (~4.35 weeks)"
> duration(1, units = 'years')  # duracao 1 ano em segundos
[1] "31557600s (~1 years)"
```

Intervalo

- Divida um intervalo por uma duração para determinar seu comprimento físico, divida um intervalo por um período para determinar seu comprimento no tempo do relógio.

Make an interval with **Interval()** or **%--%**, e.g.

```
i <- interval(ymd("2017-01-01"), d)    ## 2017-01-01 UTC--2017-11-28 UTC
j <- d %--% ymd("2017-12-31")         ## 2017-11-28 UTC--2017-12-31 UTC
```



a %within% b Does interval or date-time *a* fall within interval *b*? *now()* %within% *i*



Int_start(int) Access/set the start date-time of an interval. Also **Int_end()**. *int_start(i) <- now(); int_start(i)*



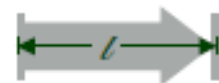
Int_aligns(int1, int2) Do two intervals share a boundary? Also **Int_overlaps()**. *int_aligns(i, j)*



Int_diff(times) Make the intervals that occur between the date-times in a vector. *v <- c(dt, dt + 100, dt + 1000); int_diff(v)*



Int_flip(int) Reverse the direction of an interval. Also **Int_standardize()**. *int_flip(i)*



Int_length(int) Length in seconds. *int_length(i)*



Int_shift(int, by) Shifts an interval up or down the timeline by a timespan. *int_shift(i, days(-1))*

Exemplo

```
> inicio <- dmy('01-02-2003')
> fim <- dmy('02-03-2005')
>
> inter1 <- interval(inicio, fim) # intervalo 1
> inter2 <- mdy('05-04-2004') %--% mdy('03/12/2012') # intervalo 2
> inter1; inter2 # duas formas diferentes de criar intervalo
[1] 2003-02-01 UTC--2005-03-02 UTC
[1] 2004-05-04 UTC--2012-03-12 UTC
>
> inter1 %within% inter2 # inter 1 está contido em inter 2?
[1] FALSE
> int_overlaps(inter1, inter2) # inter 1 e inter 2 tem interseccao?
[1] TRUE
> int_length(inter2) # comprimento em segundos
[1] 247881600
> int_aligns(inter1, inter2) # intervalos compartilham fronteira?
[1] FALSE
>
> inter1/ddays(1) # intervalo em duracao em dias
[1] 760
> as.duration(inter1)/ddays(1) # idem
[1] 760
>
> inter1/dweeks(1) # intervalo em duracao de semanas
[1] 108.5714
> inter1/dyears(1) # intervalo em duracao de anos
[1] 2.080767
```

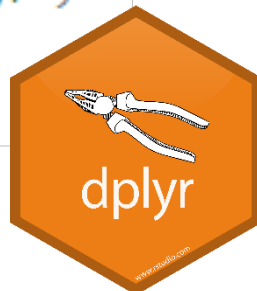
Manipulação de *data frames*

- Em *analytics* usualmente começamos o estudo a partir de bases de dados “cruas”.
- Esse dado “cru” precisa passar por processos de tratamento, manipulação, agregação e criação de informação até chegarmos na base de dados analítica (a qual é necessariamente estruturada).
- O pacote `dplyr` representa uma “gramática” fornecendo uma solução simples e eficiente para poder manipular bases de dados (documentação em: <https://cloud.r-project.org/web/packages/dplyr/dplyr.pdf>).
- Ele baseia-se em 5 verbos que são as tarefas mais básicas de manipulação: `select()`, `filter()`, `arrange()`, `mutate()` e `summarise()` [+ `group_by()`].

Exemplo

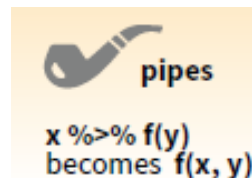
```
# instalando o dplyr
install.packages('dplyr')

# carregando o dplyr
library(dplyr)
```



Comando *pipe*

%>%



É um comando importado do pacote *magrittr* que permite que conectar a saída de uma função para a entrada de outra (ao invés de aninhar uma função dentro da outra)

Os script em R que fazem uso dos verbos do *dplyr* e o operador *pipe* tendem a ficar mais legíveis e organizados sem perder a velocidade de execução. Pode ser lido como ‘então’

Manipulação de *data frames*

select()

- Usado para escolher um subconjunto de colunas de um *data frame*.

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)



select(.data, ...)
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with **select ()**,
e.g. *select(iris, starts_with("Sepal"))*

contains(match) **num_range(prefix, range)** :, e.g. mpg:cyl
ends_with(match) **one_of(...)** -, e.g. -Species
matches(match) **starts_with(match)**

Exemplo

```
> data <- data.frame(gender = c("M", "M", "F"),  
+                   age     = c(20, 60, 30),  
+                   height  = c(180, 200, 150))
```

```
> data  
  gender age height  
1      M  20   180  
2      M  60   200  
3      F  30   150
```

```
> data %>% select(age)
```

```
  age  
1  20  
2  60  
3  30
```

```
> data %>% select(-age)
```

```
  gender height  
1      M   180  
2      M   200  
3      F   150
```

| gender | age | height |
|--------|-----|--------|
| M | 20 | 180 |
| M | 60 | 200 |
| F | 30 | 150 |



| age |
|-----|
| 20 |
| 60 |
| 30 |

- Usado para escolher um subconjunto de linhas de um *data frame*.

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.



filter(.data, ...) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*



distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values. *distinct(iris, Species)*



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows. *sample_frac(iris, 0.5, replace = TRUE)*



sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, ...) Select rows by position. *slice(iris, 10:15)*

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Exemplo

```
> data <- data.frame(gender = c("M", "M", "F"),  
+                    age     = c(20, 60, 30),  
+                    height  = c(180, 200, 150))
```

```
> data  
  gender age height  
1      M  20   180  
2      M  60   200  
3      F  30   150
```

```
> data %>% filter(height > 160)
```

```
  gender age height  
1      M  20   180  
2      M  60   200
```

```
> data %>% filter(height > 160 & age > 30)
```

```
  gender age height  
1      M  60   200
```

| gender | age | height |
|--------|-----|--------|
| M | 20 | 180 |
| M | 60 | 200 |
| F | 30 | 150 |



| gender | age | height |
|--------|-----|--------|
| M | 20 | 180 |
| M | 60 | 200 |

Manipulação de *data frames*

arrange()

- Usado para ordenar colunas de um *data frame*.

Manipulate Cases

ARRANGE CASES



arrange(.data, ...) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
`arrange(mtcars, mpg)`
`arrange(mtcars, desc(mpg))`

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL)
Add one or more rows to a table.
`add_row(faithful, eruptions = 1, waiting = 1)`

Exemplo

```
> data <- data.frame(gender = c("M", "M", "F"),  
+                    age     = c(20, 60, 30),  
+                    height  = c(180, 200, 150))
```

```
> data  
  gender age height  
1      M  20   180  
2      M  60   200  
3      F  30   150
```

```
> data %>% arrange(height)
```

```
  gender age height  
1      F  30   150  
2      M  20   180  
3      M  60   200
```

```
> data %>% arrange(desc(height))
```

```
  gender age height  
1      M  60   200  
2      M  20   180  
3      F  30   150
```

| gender | age | height |
|--------|-----|--------|
| M | 20 | 180 |
| M | 60 | 200 |
| F | 30 | 150 |



| gender | age | height |
|--------|-----|--------|
| F | 30 | 150 |
| M | 20 | 180 |
| M | 60 | 200 |

Manipulação de *data frames*

mutate()

- Usado para adicionar novas colunas (i.e. novas variáveis) a um *data frame*.

Manipulate Variables

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funks take vectors as input and return vectors of the same length as output (see back).

vectorized function



mutate(.data, ...)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)



transmute(.data, ...)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)



mutate_all(.tbl, .funs, ...) Apply funks to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))



add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*



rename(.data, ...) Rename columns.
rename(iris, Length = Sepal.Length)

Exemplo

```
> data <- data.frame(gender = c("M", "M", "F"),  
+                   age     = c(20, 60, 30),  
+                   height  = c(180, 200, 150))  
> data  
  gender age height  
1     M  20   180  
2     M  60   200  
3     F  30   150  
>  
> data %>% mutate(height2 = height/100)  
  gender age height height2  
1     M  20   180     1.8  
2     M  60   200     2.0  
3     F  30   150     1.5  
>  
> data %>% mutate(var = if_else(age < 30, 'A', 'B',  
+                               missing = 'C'))  
  gender age height var  
1     M  20   180    A  
2     M  60   200    B  
3     F  30   150    B
```

| gender | age | height |
|--------|-----|--------|
| M | 20 | 180 |
| M | 60 | 200 |
| F | 30 | 150 |



| gender | age | height | height2 |
|--------|-----|--------|---------|
| M | 20 | 180 | 1.8 |
| M | 60 | 200 | 2.0 |
| F | 30 | 150 | 1.5 |

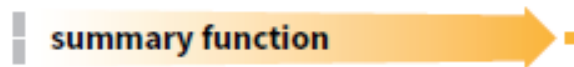
Manipulação de *data frames*

summarise()

- Usado para agregar valores de um *data frame*

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).



summarise(.data, ...)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))



count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in ... Also **tally()**.
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.



Existe uma infinidade de funções que pode ser usadas na sumarização de *data frames* (veja seção extra de funções)

Exemplo

```
> data <- data.frame(gender = c("M", "M", "F"),  
+                   age     = c(20, 60, 30),  
+                   height  = c(180, 200, 150))  
> data  
  gender age height  
1      M  20   180  
2      M  60   200  
3      F  30   150  
>  
>  
> data %>% summarise(average_height = mean(height))  
  average_height  
1       176.6667  
>  
> data %>% count(gender)  
# A tibble: 2 x 2  
  gender      n  
  <fct> <int>  
1 F         1  
2 M         2
```

| gender | age | height |
|--------|-----|--------|
| M | 20 | 180 |
| M | 60 | 200 |
| F | 30 | 150 |



| average_height |
|----------------|
| 177 |

Manipulação de *data frames*

group_by()

- Usado para mudar o dado para a forma agrupada de forma a aplicar funções separadas em cada grupo

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.



```
mtcars %>%  
group_by(cyl) %>%  
summarise(avg = mean(mpg))
```

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by ...
`g_iris <- group_by(iris, Species)`

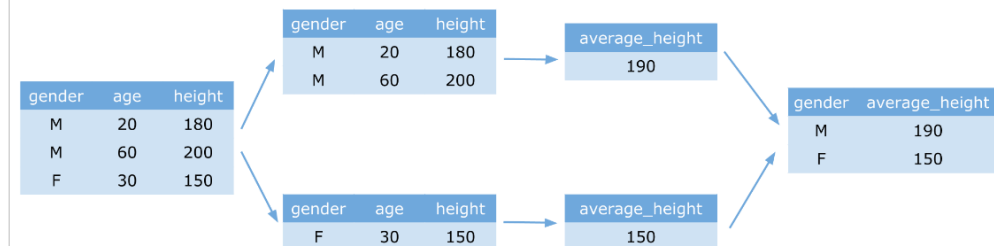
ungroup(x, ...)
Returns ungrouped copy of table.
`ungroup(g_iris)`



Existe uma infinidade de funções que pode ser usadas na sumarização de *data frames* (veja seção extra de funções)

Exemplo

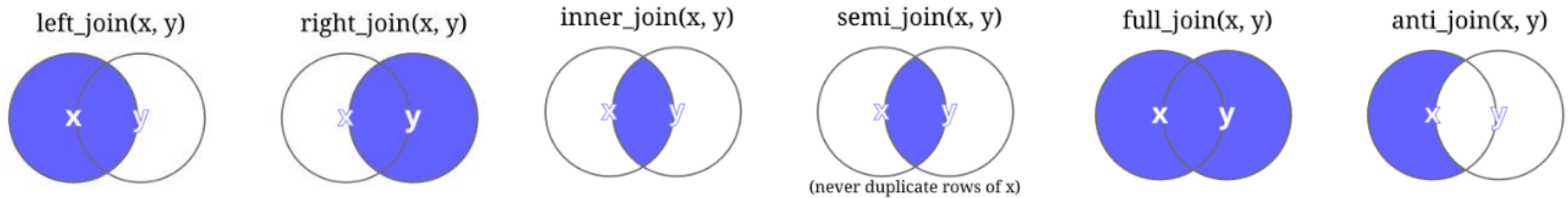
```
> data <- data.frame(gender = c("M", "M", "F"),  
+                   age     = c(20, 60, 30),  
+                   height  = c(180, 200, 150))  
> data  
  gender age height  
1     M  20   180  
2     M  60   200  
3     F  30   150  
>  
> data %>% group_by(gender) %>%  
+   summarise(average_height = mean(height))  
`summarise()` ungrouping output (override with `.groups` argument)  
# A tibble: 2 x 2  
  gender average_height  
  <chr>         <dbl>  
1 F             150  
2 M             190
```



Junção de tabelas

- É raro que a informação esteja contida apenas uma única base de dados. Tipicamente são combinados dados diversos que possuem relação entre si para que seja possível responder às questões de negócio.
- Além de permitir a manipulação dos dados, o pacote `dplyr` possui funcionalidades que permitem o cruzamento de informações oriundas de tabelas distintas (similar ao SQL).
- Aqui as junções podem ser feitas concatenando colunas ou linhas, ou fazendo a correspondência com linhas que contenham o mesmo valor (*keys*).
- Chave (*key*) é uma coluna (ou um conjunto de colunas) que define unicamente uma observação (linha).

Tipos de *joins*



https://coletl.github.io/tidy_intro/lessons/dplyr_join/dplyr_join.html

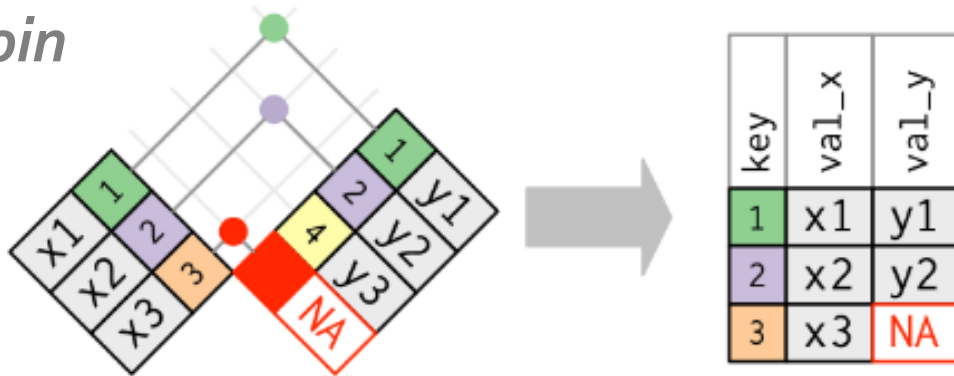
Manipulação de *data frames*

Junção de tabelas

- Intuição de como funciona o cruzamento de tabelas de dados x e y

| x | y |
|------|------|
| 1 x1 | 1 y1 |
| 2 x2 | 2 y2 |
| 3 x3 | 4 y3 |

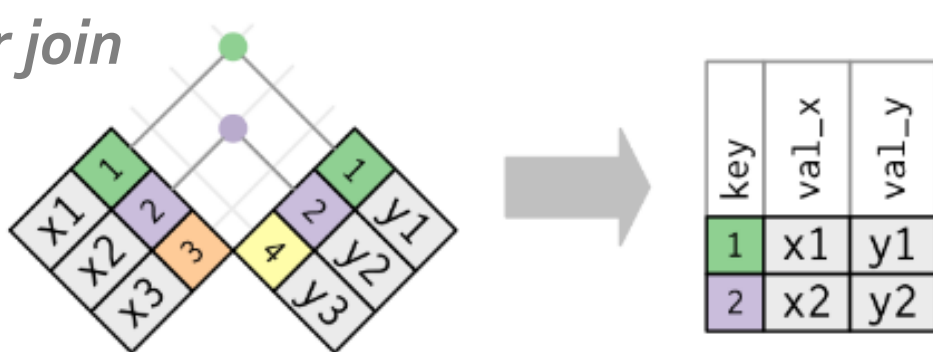
left join



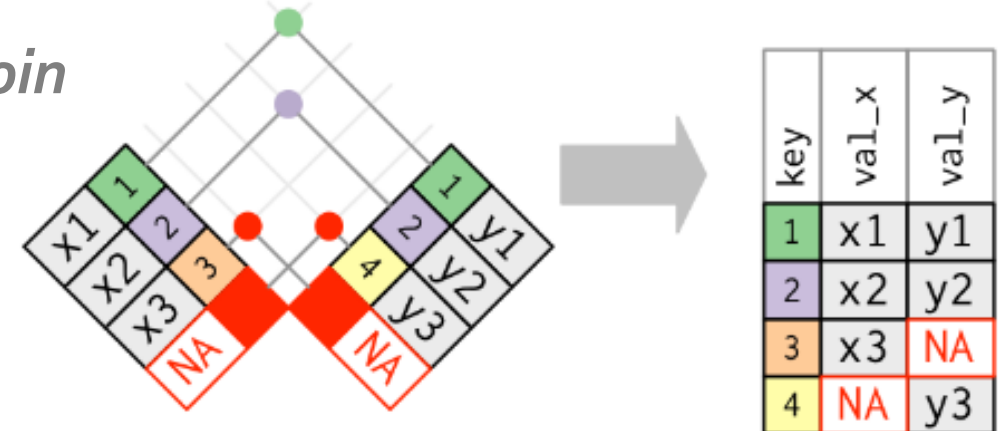
right join



inner join



full join



Manipulação de *data frames*

left join

- Retorna todas as linhas x e todas as colunas de x. Linhas de x sem correspondência em y terão valores NA nas novas colunas. Se houver múltiplas correspondências entre x e y, todas elas serão retornadas.

| A | B | C | D |
|---|---|---|----|
| a | t | 1 | 3 |
| b | u | 2 | 2 |
| c | v | 3 | NA |

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),...)
Join matching values from y to x.

| A | B | C | B.y | D |
|---|---|---|-----|----|
| a | t | 1 | t | 3 |
| b | u | 2 | u | 2 |
| c | v | 3 | NA | NA |

Use **by = c("col1", "col2")** to specify the column(s) to match on.
`left_join(x, y, by = "A")`

| A.x | B.x | C | A.y | B.y |
|-----|-----|---|-----|-----|
| a | t | 1 | d | w |
| b | u | 2 | b | u |
| c | v | 3 | a | t |

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
`left_join(x, y, by = c("C" = "D"))`

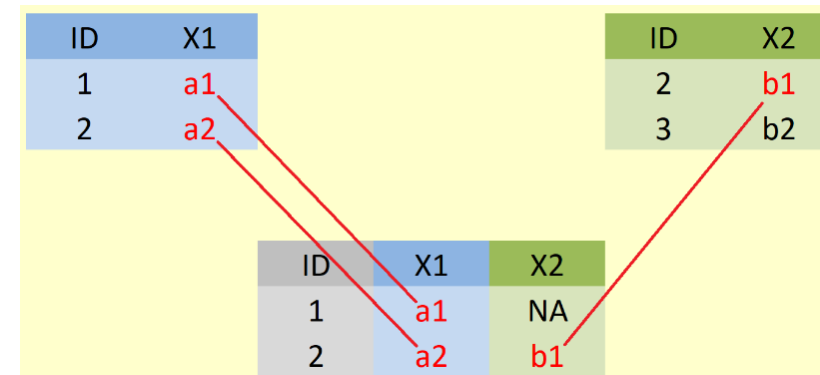
| A1 | B1 | C | A2 | B2 |
|----|----|---|----|----|
| a | t | 1 | d | w |
| b | u | 2 | b | u |
| c | v | 3 | a | t |

Use **suffix** to specify suffix to give to duplicate column names.
`left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))`

Exemplo

```
> data1 <- data.frame(ID = 1:2,                # Df 1
+                     X1 = c("a1", "a2"),
+                     stringsAsFactors = FALSE)
> data2 <- data.frame(ID = 2:3,                # Df 2
+                     X2 = c("b1", "b2"),
+                     stringsAsFactors = FALSE)
>
>
> left_join(data1, data2, by = 'ID')
```

| ID | X1 | X2 |
|----|----|------|
| 1 | a1 | <NA> |
| 2 | a2 | b1 |



<https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti>

right join

- Retorna todas as linhas de y e todas as colunas de x e y. Linhas em y sem correspondência com x terão valores NA nas novas colunas. Se há múltiplas correspondências entre x e y, todas as combinação serão retornadas.

| A | B | C | D |
|---|---|----|---|
| a | t | 1 | 3 |
| b | u | 2 | 2 |
| d | w | NA | 1 |

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x", ".y"),...)
Join matching values from x to y.

| A | B | C | B.y | D |
|---|---|---|-----|----|
| a | t | 1 | t | 3 |
| b | u | 2 | u | 2 |
| c | v | 3 | NA | NA |

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

| A.x | B.x | C | A.y | B.y |
|-----|-----|---|-----|-----|
| a | t | 1 | d | w |
| b | u | 2 | b | u |
| c | v | 3 | a | t |

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

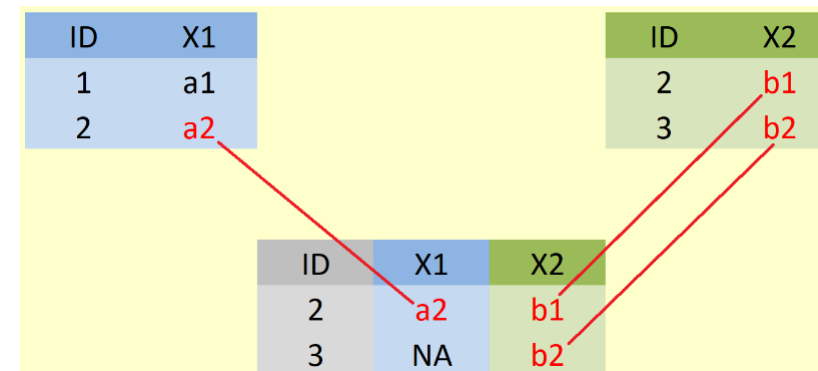
| A1 | B1 | C | A2 | B2 |
|----|----|---|----|----|
| a | t | 1 | d | w |
| b | u | 2 | b | u |
| c | v | 3 | a | t |

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

Exemplo

```
> data1 <- data.frame(ID = 1:2,                # Df 1
+                     X1 = c("a1", "a2"),
+                     stringsAsFactors = FALSE)
> data2 <- data.frame(ID = 2:3,                # Df 2
+                     X2 = c("b1", "b2"),
+                     stringsAsFactors = FALSE)
>
>
> right_join(data1, data2, by = 'ID')
```

| ID | X1 | X2 |
|----|----|---------|
| 1 | 2 | a2 b1 |
| 2 | 3 | <NA> b2 |



<https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti>

Manipulação de *data frames*

inner join

- Retorna todas as linhas x onde há correspondência de y e todas as colunas de x e y. Se há múltiplas correspondências entre x e y, todas as combinações serão retornadas.

| A | B | C | D |
|---|---|---|---|
| a | t | 1 | 3 |
| b | u | 2 | 2 |

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),...)
Join data. Retain only rows with matches.

| A | B | C | B.y | D |
|---|---|---|-----|----|
| a | t | 1 | t | 3 |
| b | u | 2 | u | 2 |
| c | v | 3 | NA | NA |

Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

| A.x | B.x | C | A.y | B.y |
|-----|-----|---|-----|-----|
| a | t | 1 | d | w |
| b | u | 2 | b | u |
| c | v | 3 | a | t |

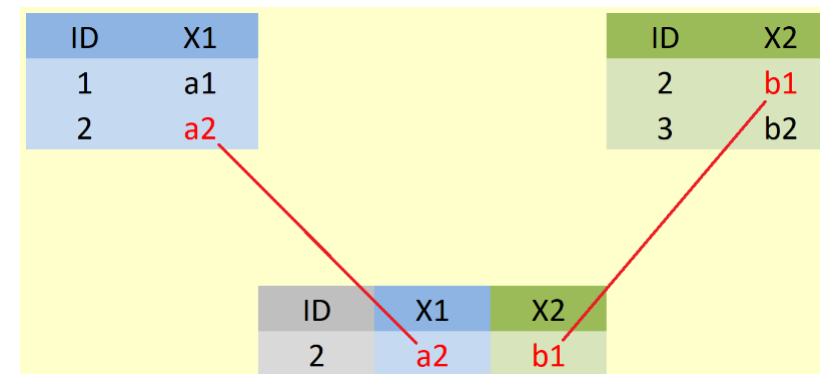
Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

| A1 | B1 | C | A2 | B2 |
|----|----|---|----|----|
| a | t | 1 | d | w |
| b | u | 2 | b | u |
| c | v | 3 | a | t |

Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

Exemplo

```
> data1 <- data.frame(ID = 1:2,                # Df 1
+                     X1 = c("a1", "a2"),
+                     stringsAsFactors = FALSE)
> data2 <- data.frame(ID = 2:3,                # Df 2
+                     X2 = c("b1", "b2"),
+                     stringsAsFactors = FALSE)
>
>
> inner_join(data1, data2, by = 'ID')
  ID X1 X2
1  2 a2 b1
```



<https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti>

Manipulação de *data frames*

full join

- Retorna todas as linhas e todas as colunas de ambos x e y. Quando não há correspondência, retorna NA para os valores omissos.

| A | B | C | D |
|---|---|----|----|
| a | t | 1 | 3 |
| b | u | 2 | 2 |
| c | v | 3 | NA |
| d | w | NA | 1 |

full_join(x, y, by = NULL,
copy=FALSE, suffix=c(".x",".y"),...)
Join data. Retain all values, all rows.

| A | B | C | B.y | D |
|---|---|---|-----|----|
| a | t | 1 | t | 3 |
| b | u | 2 | u | 2 |
| c | v | 3 | NA | NA |

Use **by = c("col1", "col2")** to
specify the column(s) to match on.
left_join(x, y, by = "A")

| A.x | B.x | C | A.y | B.y |
|-----|-----|---|-----|-----|
| a | t | 1 | d | w |
| b | u | 2 | b | u |
| c | v | 3 | a | t |

Use a named vector, **by = c("col1" =
"col2")**, to match on columns with
different names in each data set.
left_join(x, y, by = c("C" = "D"))

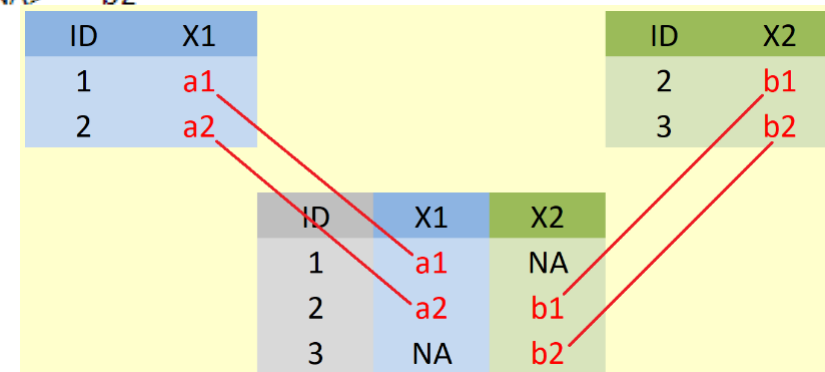
| A1 | B1 | C | A2 | B2 |
|----|----|---|----|----|
| a | t | 1 | d | w |
| b | u | 2 | b | u |
| c | v | 3 | a | t |

Use **suffix** to specify suffix to give to
duplicate column names.
**left_join(x, y, by = c("C" = "D"), suffix =
c("1", "2"))**

Exemplo

```
> data1 <- data.frame(ID = 1:2,           # Df 1
+                     X1 = c("a1", "a2"),
+                     stringsAsFactors = FALSE)
> data2 <- data.frame(ID = 2:3,           # Df 2
+                     X2 = c("b1", "b2"),
+                     stringsAsFactors = FALSE)
>
>
> full_join(data1, data2, by = 'ID')
```

| ID | X1 | X2 |
|----|------|------|
| 1 | a1 | <NA> |
| 2 | a2 | b1 |
| 3 | <NA> | b2 |



<https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti>

Manipulação de *data frames*

binding [extra]

- É possível ligar uma tabela a outra via coluna (i.e. agregando novas variáveis a tabela final) ou ligando via linha (i.e. acrescentando novos registros).

Combine Tables

COMBINE VARIABLES

| x | | y | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <table><tr><th>A</th><th>B</th><th>C</th></tr><tr><td>a</td><td>t</td><td>1</td></tr><tr><td>b</td><td>u</td><td>2</td></tr><tr><td>c</td><td>v</td><td>3</td></tr></table> | A | B | C | a | t | 1 | b | u | 2 | c | v | 3 | + | <table><tr><th>A</th><th>B</th><th>D</th></tr><tr><td>a</td><td>t</td><td>3</td></tr><tr><td>b</td><td>u</td><td>2</td></tr><tr><td>d</td><td>w</td><td>1</td></tr></table> | A | B | D | a | t | 3 | b | u | 2 | d | w | 1 | = |
| A | B | C | | | | | | | | | | | | | | | | | | | | | | | | | |
| a | t | 1 | | | | | | | | | | | | | | | | | | | | | | | | | |
| b | u | 2 | | | | | | | | | | | | | | | | | | | | | | | | | |
| c | v | 3 | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | B | D | | | | | | | | | | | | | | | | | | | | | | | | | |
| a | t | 3 | | | | | | | | | | | | | | | | | | | | | | | | | |
| b | u | 2 | | | | | | | | | | | | | | | | | | | | | | | | | |
| d | w | 1 | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | <table><tr><th>A</th><th>B</th><th>C</th><th>A</th><th>B</th><th>D</th></tr><tr><td>a</td><td>t</td><td>1</td><td>a</td><td>t</td><td>3</td></tr><tr><td>b</td><td>u</td><td>2</td><td>b</td><td>u</td><td>2</td></tr><tr><td>c</td><td>v</td><td>3</td><td>d</td><td>w</td><td>1</td></tr></table> | A | B | C | A | B | D | a | t | 1 | a | t | 3 | b | u | 2 | b | u | 2 | c | v | 3 | d | w | 1 | |
| A | B | C | A | B | D | | | | | | | | | | | | | | | | | | | | | | |
| a | t | 1 | a | t | 3 | | | | | | | | | | | | | | | | | | | | | | |
| b | u | 2 | b | u | 2 | | | | | | | | | | | | | | | | | | | | | | |
| c | v | 3 | d | w | 1 | | | | | | | | | | | | | | | | | | | | | | |

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

COMBINE CASES

| x | y | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <table><thead><tr><th>A</th><th>B</th><th>C</th></tr></thead><tbody><tr><td>a</td><td>t</td><td>1</td></tr><tr><td>b</td><td>u</td><td>2</td></tr><tr><td>c</td><td>v</td><td>3</td></tr></tbody></table> | A | B | C | a | t | 1 | b | u | 2 | c | v | 3 | <table><thead><tr><th>A</th><th>B</th><th>C</th></tr></thead><tbody><tr><td>a</td><td>t</td><td>3</td></tr><tr><td>b</td><td>u</td><td>2</td></tr><tr><td>d</td><td>w</td><td>4</td></tr></tbody></table> | A | B | C | a | t | 3 | b | u | 2 | d | w | 4 | <table><thead><tr><th>A</th><th>B</th><th>C</th></tr></thead><tbody><tr><td>a</td><td>t</td><td>1</td></tr><tr><td>b</td><td>u</td><td>2</td></tr><tr><td>c</td><td>v</td><td>3</td></tr><tr><td>a</td><td>t</td><td>3</td></tr><tr><td>b</td><td>u</td><td>2</td></tr><tr><td>d</td><td>w</td><td>4</td></tr></tbody></table> | A | B | C | a | t | 1 | b | u | 2 | c | v | 3 | a | t | 3 | b | u | 2 | d | w | 4 |
| A | B | C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a | t | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b | u | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c | v | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | B | C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a | t | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b | u | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| d | w | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | B | C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a | t | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b | u | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c | v | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a | t | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b | u | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| d | w | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Use **bind_rows()** to paste tables below each other as they are.

| DF | A | B | C |
|----|---|---|---|
| x | a | t | 1 |
| x | b | u | 2 |
| x | c | v | 3 |
| y | a | t | 3 |
| y | b | u | 2 |
| y | d | w | 4 |

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set **.id** to a column name to add a column of the original table names (as pictured)

| A | B | C |
|---|---|---|
| c | v | 3 |

intersect(x, y, ...)
Rows that appear in both x and y.

| A | B | C |
|---|---|---|
| a | t | 1 |
| b | u | 2 |

setdiff(x, y, ...)
Rows that appear in x but not y.

| A | B | C |
|---|---|---|
| a | t | 1 |
| b | u | 2 |
| c | v | 3 |
| d | w | 4 |

union(x, y, ...)
Rows that appear in x or y.
(Duplicates removed). **union_all()** retains duplicates.

Exemplo

```
> data1 <- data.frame(ID = 1:2,  
+                     X1 = c("a1", "a2"),  
+                     stringsAsFactors = FALSE) # df1  
> data2 <- data.frame(ID = 1:2,  
+                     X2 = c("b1", "b2"),  
+                     stringsAsFactors = FALSE) # df2  
> data3 <- data.frame(ID = 3:4,  
+                     X1 = c("a3", "a4"),  
+                     stringsAsFactors = FALSE) # df3  
>
```

```
> bind_cols(data1, data2)  
  ID X1 ID1 X2  
1  1 a1  1 b1  
2  2 a2  2 b2  
>
```

```
> bind_rows(data1, data3)  
  ID X1  
1  1 a1  
2  2 a2  
3  3 a3  
4  4 a4  
>
```

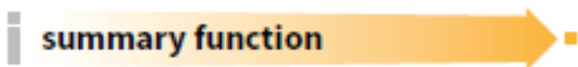

Manipulação de *data frames*

Funções úteis [extra]

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

 summary function

COUNTS

dplyr::n() - number of values/rows
dplyr::n_distinct() - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::first() - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK


quantile() - nth quantile
min() - minimum value
max() - maximum value


SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.


rownames_to_column()
Move row names into col.
`a <- rownames_to_column(iris, var = "C")`


column_to_rownames()
Move col in row names.
`column_to_rownames(a, var = "C")`

Also **has_rownames()**, **remove_rownames()**

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

 vectorized function

OFFSETS

dplyr::lag() - Offset elements by 1
dplyr::lead() - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::cumall() - Cumulative all()
dplyr::cumany() - Cumulative any()
cummax() - Cumulative max()
dplyr::cummean() - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::cume_dist() - Proportion of all values <=
dplyr::dense_rank() - rank with ties = min, no gaps
dplyr::min_rank() - rank with ties = min
dplyr::ntile() - bins into n bins
dplyr::percent_rank() - min_rank scaled to [0,1]
dplyr::row_number() - rank with ties = "first"

Prática no RStudio

...foco de hoje

- **CASE 2.1: Pagamento de faturas de cartão**

Comandos de leitura, trabalhando com datas e intervalos de tempo, sumarização de *data frames*

- **CASE 2.2: Série de vendas de supermercados da rede Walmart nos EUA (pt. 1)**

Manipulação e cruzamento de diferentes *dataframes*, agregação de variáveis, manipulação de datas

