Introdução ao tidyverse

3 tidyverse

xaringan [presentation ninja]

Maurício Vancine
25/04/2019


tidyverse
www.rstudio.com

3 tidyverse

Conteúdo

3 tidyverse

Script

`script_aula_03.R`


tidyverse
www.rstudio.com

3.1 Pacote tidyverse

O tidyverse é um **pacote** com a função de **instalar** e **carregar** outros pacotes

O **conjunto** desses pacotes forma o **tidyverse**

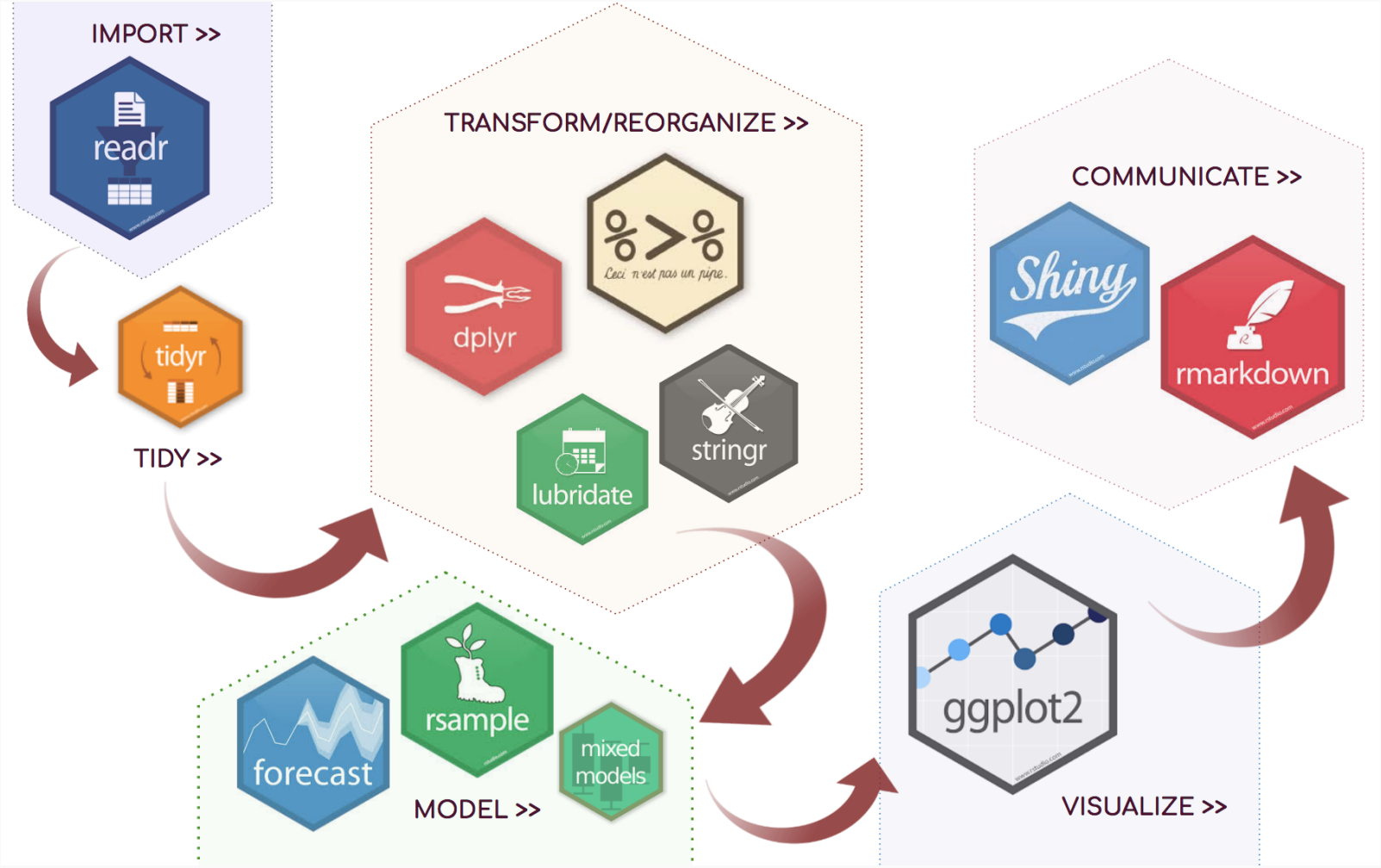
É considerado um "universo" à parte do R, pois todas as suas **ferramentas** possuem formas de uso consistentes e **funcionam** muito bem em conjunto

Seu uso é mais voltado para a **Ciência de Dados**

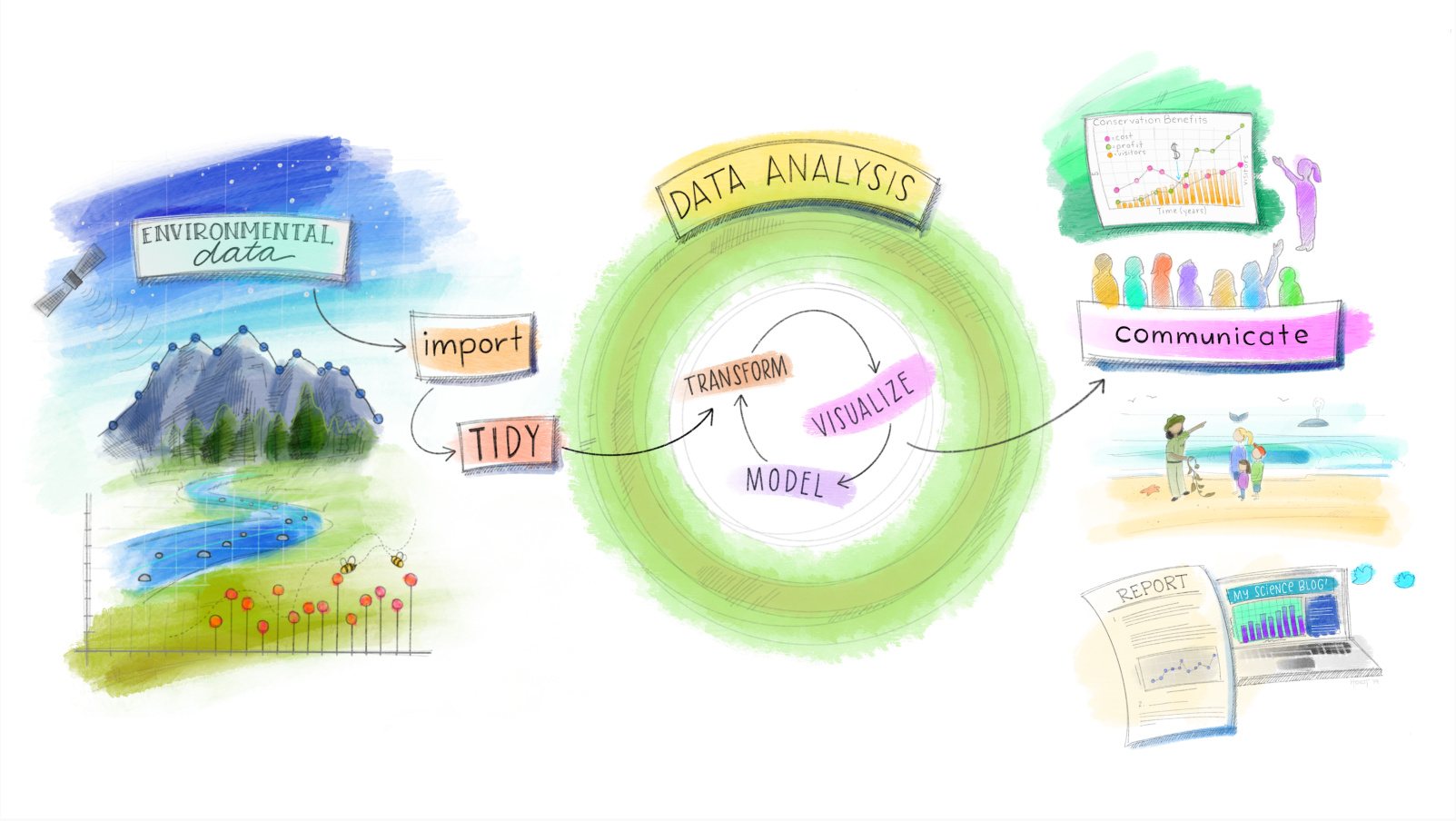
E depois que vocês **aprenderem**, nunca mais usaram o R de outra forma...

Iniciativa Vingadores do R

3.1 Pacote tidyverse



3.1 Pacote tidyverse



Fonte: @allison_horst

3.1 Pacote tidyverse

Fonte: @ViviFabrien

3.1 Pacote tidyverse

O idealizador foi o **Hadley Wickham** e atualmente **muitas pessoas** têm contribuído para sua expansão

[*] http://hadley.nz/

3.1 Pacote tidyverse

Tidy Data (2014) - *Journal of Statistics Software*

Hadley Wickham

Journal of Statistical Software
MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

Tidy Data

Hadley Wickham
RStudio

Abstract

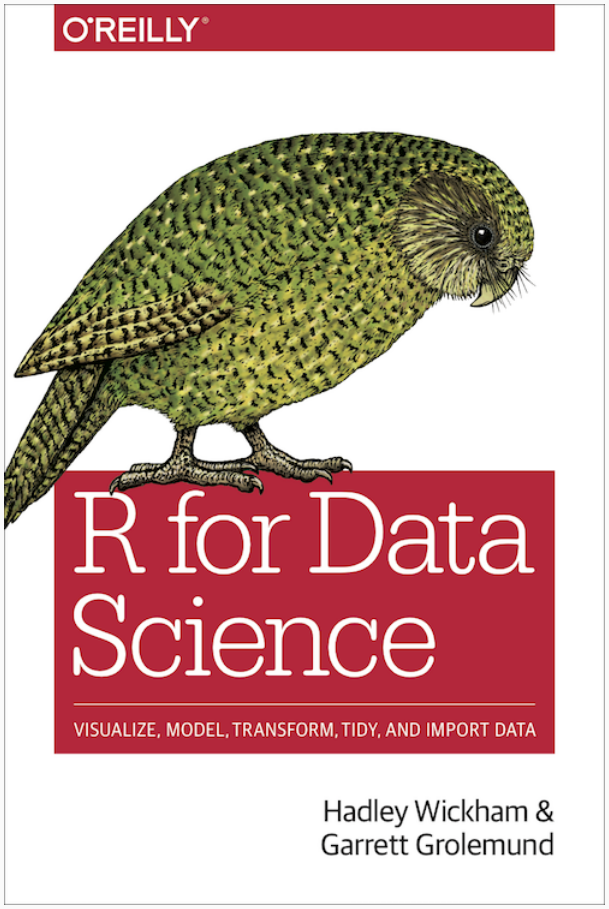
A huge amount of effort is spent cleaning data to get it ready for analysis, but there has been little research on how to make data cleaning as easy and effective as possible. This paper tackles a small, but important, component of data cleaning: data tidying. Tidy datasets are easy to manipulate, model and visualise, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table. This framework makes it easy to tidy messy datasets because only a small set of tools are needed to deal with a wide range of un-tidy datasets. This structure also makes it easier to develop tidy tools for data analysis, tools that both input and output tidy datasets. The advantages of a consistent data structure and matching tools are demonstrated with a case study free from mundane data manipulation chores.

Keywords: data cleaning, data tidying, relational databases, R.

[*] http://vita.had.co.nz/papers/tidy-data.pdf

3.1 Pacote tidyverse

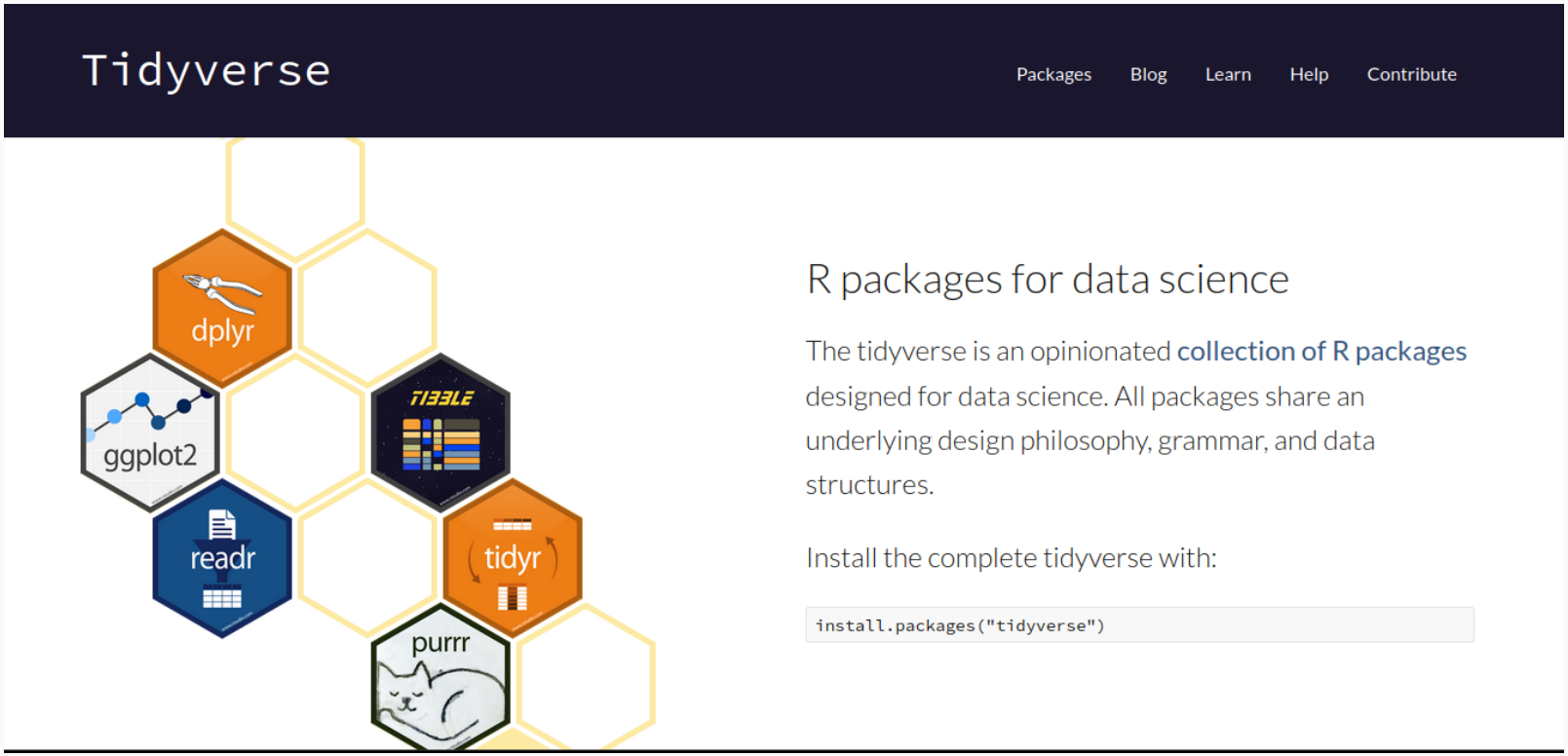
R for Data Science (2017)



[*] https://r4ds.had.co.nz/

3.1 Pacote tidyverse

Sites



[*] https://www.tidyverse.org/

3.1 Pacote tidyverse

Sites

RPubs brought to you by RStudio

R para data science

Manipulação e Visualização de dados utilizando os pacotes do tidyverse()

Uma breve discussão sobre os procedimentos padrões para se trabalhar com dados

Para uma melhor compreensão de como é feito, em geral, o trabalho em data science, vamos definir os procedimentos ou passos a serem considerados na análise. Os procedimentos que apresentamos decorrem das ideias descritas no livro (Grolemund,2017) e podem ser observados na figura abaixo.

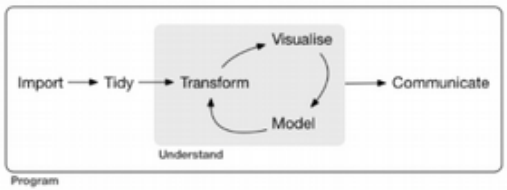


Diagrama de ferramentas para Data Science em (Grolemund,2017)

Pensando em um projeto com dados retangulares, devemos ter **as seguintes etapas**:

Passo 1: Formulação da pergunta de pesquisa

[*] https://rpubs.com/modelthinkingbr/dados

3.1 Pacote tidyverse

Para utilizar os pacotes do **tidyverse** é preciso instalar e carregar o pacote `tidyverse`

```
# instalar o pacote
install.packages("tidyverse")
```

```
# carregar o pacote
library(tidyverse)
```

3.1 Pacote tidyverse

IMPORTANTE!

Todas as funções dos pacotes atrelados ao **tidyverse** usam `_` para separar os nomes internos das funções (snake_code)

`read_csv()`

`read_xlsx()`

`as_tibble()`

`left_join()`

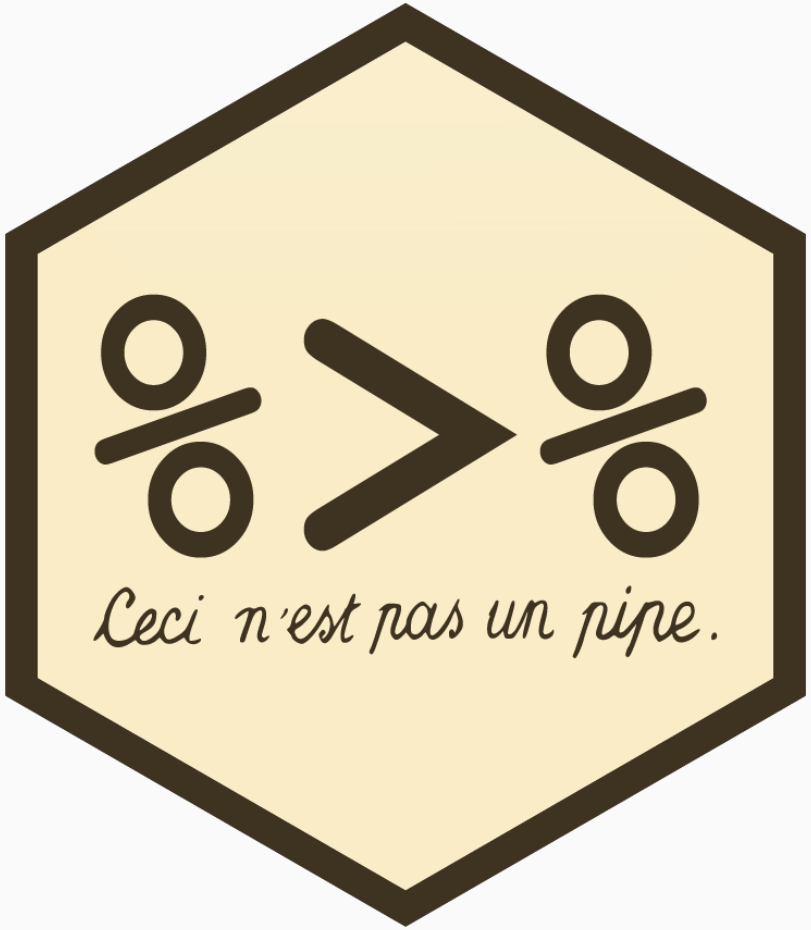
`group_by()`

3.1 Pacote tidyverse

Listar todos os pacotes do tidyverse

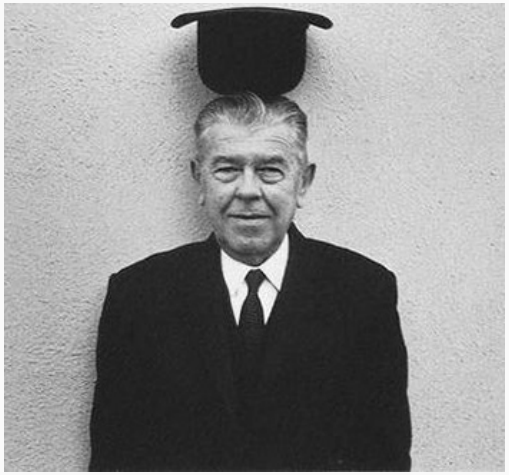
```
# list all packages in the tidyverse
tidyverse::tidyverse_packages(include_self = TRUE)
```

```
##  [1] "broom"      "cli"        "crayon"     "dbplyr"     "dplyr"      "forcats
## [10] "httr"       "jsonlite"   "lubridate"  "magrittr"   "modelr"     "pillar"
## [19] "reprex"     "rlang"      "rstudioapi" "rvest"      "stringr"    "tibble"
```


%>%
Ceci n'est pas un pipe.

3.2 magrittr (pipe - %>%)

René Magritte (1898-1967)



3.2 magrittr (pipe - %>%)

O operador pipe (*%>%*) permite o "encadeamento" de várias funções e **não é preciso de objetos** para armazenar resultados intermediários

Essa função torna os códigos em R **mais simples**, pois realizamos **múltiplas operações** em uma **única linha**

Ele captura o **resultado de uma declaração** e o **torna a entrada da próxima declaração**. Podemos pensar como *"EM SEGUIDA FAÇA"*

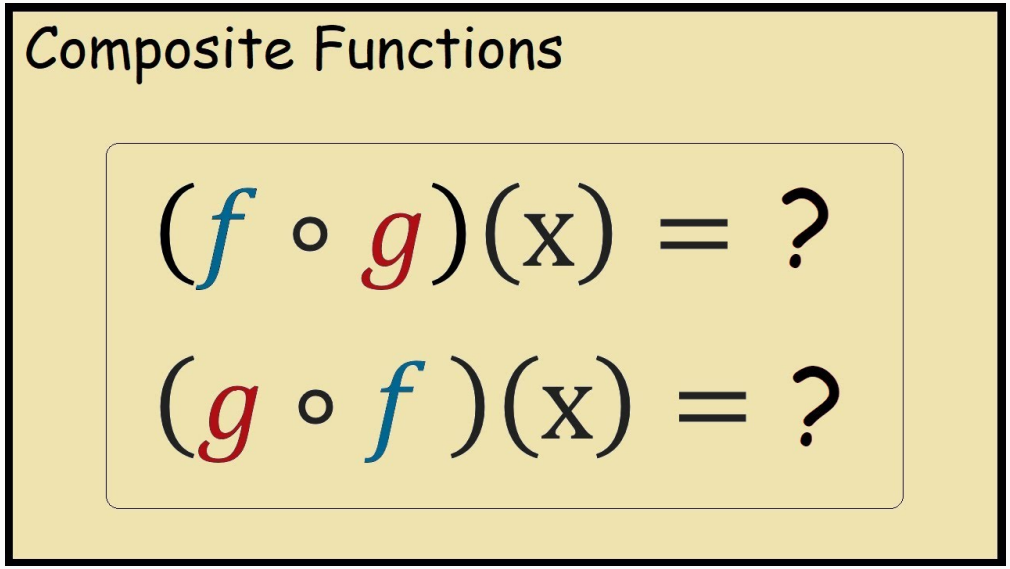
O operador pipe é **%>%**

3.2 magrittr (pipe - %>%)

Atalho: `crtl + shift + M`

```
# sem pipe
sqrt(sum(1:100))
```

```
## [1] 71.06335
```



3.2 magrittr (pipe - %>%)

Atalho: `crtl + shift + M`

```
# sem pipe
sqrt(sum(1:100))
```

```
## [1] 71.06335
```

```
# com pipe
1:100 %>%
  sum() %>%
  sqrt()
```

```
## [1] 71.06335
```

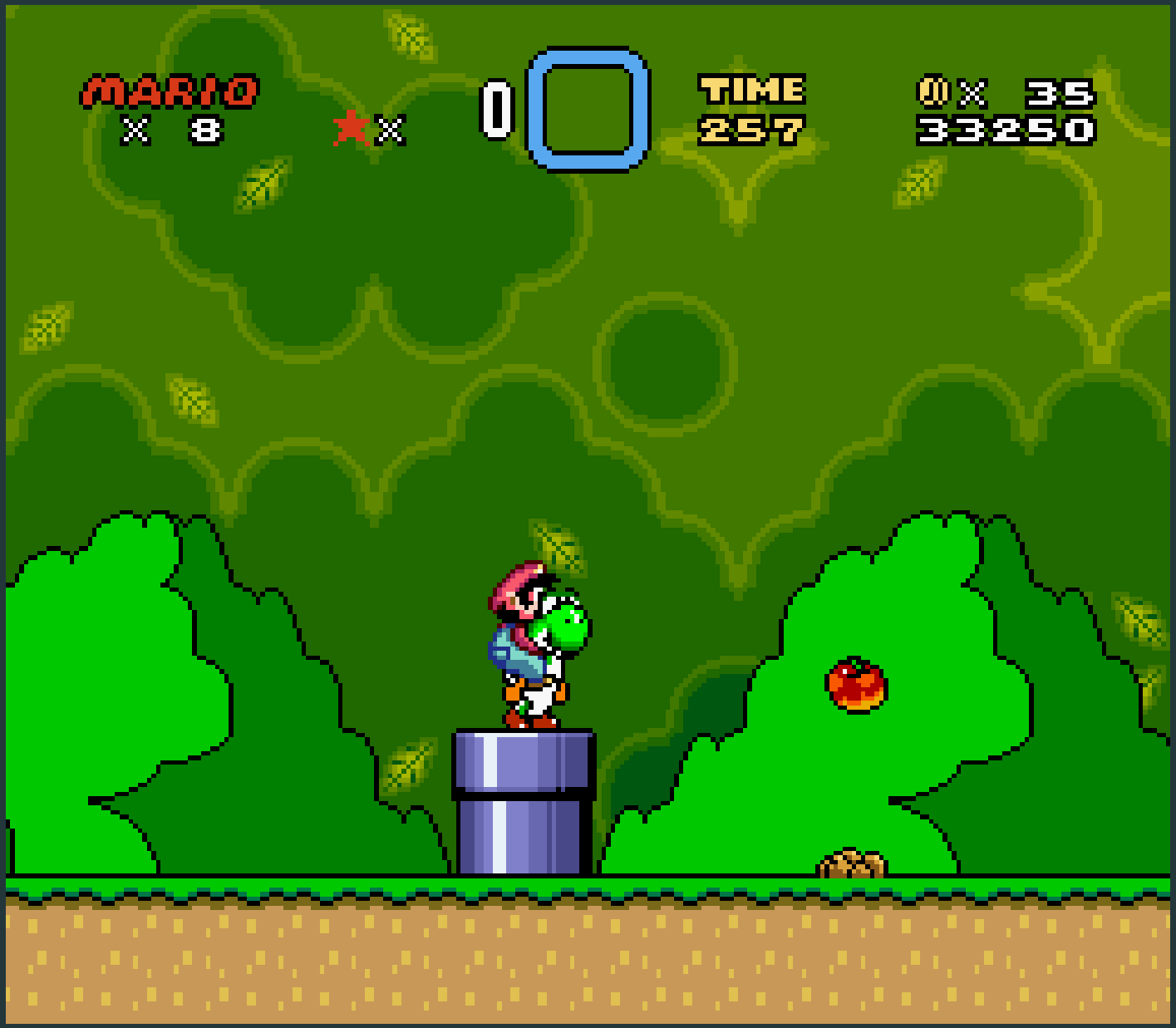
3.2 magrittr (pipe - %>%)

Atalho: `crtl + shift + M`

```
# fixar amostragem
set.seed(42)

# sem pipe
ve <- sum(sqrt(sort(log10(rpois(100, 10)))))
ve
```

```
## [1] 99.91426
```

```
# fixar amostragem
set.seed(42)

# com pipe
ve <- rpois(100, 10) %>%
  log10() %>%
  sort() %>%
  sqrt() %>%
  sum()
ve
```


MARIO
X 8
0
TIME
257
0X 35
33250

Exercícios

Exercício 09

Reescreva cada uma das operações utilizando pipes `%>%`:

```
log10(cumsum(1:100))
```

```
sum(sqrt(rnorm(100)))
```

```
prod(sort(sample(1:10, 10000, rep = TRUE)))
```

05:00

Exercício 09

Solução

```
# solucao
# 1.
log10(cumsum(1:100))

1:100 %>%
  cumsum %>%
  log10
```

Exercício 09

Solução

```
# 2.
sum(sort(rnorm(100)))

100 %>%
  rnorm %>%
  sort %>%
  sum
```

Exercício 09

Solução

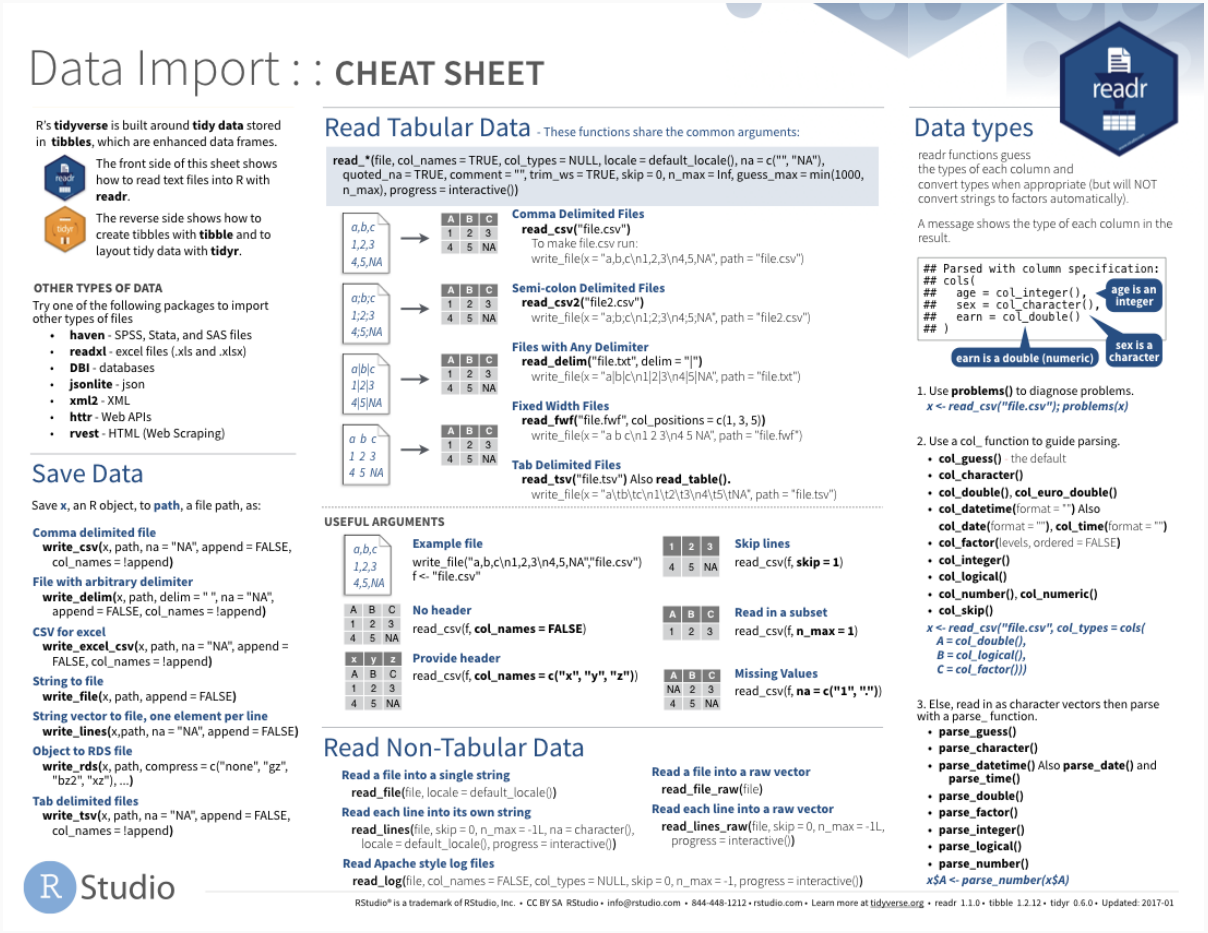
```
# 3.
scale(sample(1:10, 10000, rep = TRUE))

1:10 %>%
  sample(10000, rep = TRUE) %>%
  scale
```

Dúvidas?


readr
www.rstudio.com

3.3 readr

Data Import Cheatsheet



Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append)

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append)

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)

String to file
write_file(x, path, append = FALSE)

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE)

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...)

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)

RStudio

Read Tabular Data - These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive())

Comma Delimited Files
read_csv("file.csv")
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

Semi-colon Delimited Files
read_csv2("file2.csv")
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

Files with Any Delimiter
read_delim("file.txt", delim = "|")
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5))
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files
read_tsv("file.tsv") Also **read_table()**.
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

Skip lines
read_csv(f, **skip = 1**)

No header
read_csv(f, **col_names = FALSE**)

Read in a subset
read_csv(f, **n_max = 1**)

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

Missing Values
read_csv(f, **na = c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale())

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive())

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())

Read a file into a raw vector
read_file_raw(file)

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use **problems()** to diagnose problems.
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing.
- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime**(format = "") Also **col_date**(format = ""), **col_time**(format = "")
- **col_factor**(levels, ordered = FALSE)
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

x <- read_csv("file.csv", col_types = cols(
A = col_double(),
B = col_logical(),
C = col_factor()))

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

x$A <- parse_number(x$A)

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01

[*] https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf

3.3 readr

Carrega e salva grandes arquivos de forma **mais rápida**

As funções **read.csv()** e **read.csv2()** são substituídas pelas funções **read_csv()** e **read_csv2()**

Essas funções fornecem **medidores de progresso**

E também **classificam** automaticamente o **modo** dos dados de cada coluna

A classe do objeto atribuído é **tibble**

Para salvar arquivos no formato .csv: **write_csv()** e **write_csv2()**

3.3 readr

Formato .csv

```
# diretorio
setwd("/home/mude/data/github/minicurso-tidyverse/03_dados")
```

```
# import sites
si <- readr::read_csv("ATLANTIC_AMPHIBIANS_sites.csv")
si
```

```
## # A tibble: 1,163 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1001             19 ab     fo,ll           as
##  2 amp1…             1002             16 co     fo,la,ll        as
##  3 amp1…             1002             14 co     fo,la,ll        as
##  4 amp1…             1002             13 co     fo,la,ll        as
##  5 amp1…             1003             30 co     fo,ll,br        as
##  6 amp1…             1004             42 co     tp,pp,la,ll,is  <NA>
##  7 amp1…             1005             23 co     sp              as
##  8 amp1…             1005             19 co     sp,la,sw        as,sb,tr
```

3.3 readr

Formato .txt

```
# diretorio
setwd("/home/mude/data/github/minicurso-tidyverse/03_dados")
```

```
# import sites
si <- readr::read_tsv("ATLANTIC_AMPHIBIANS_sites.txt")
si
```

```
## # A tibble: 1,163 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1001             19 ab     fo,ll           as
##  2 amp1…             1002             16 co     fo,la,ll        as
##  3 amp1…             1002             14 co     fo,la,ll        as
##  4 amp1…             1002             13 co     fo,la,ll        as
##  5 amp1…             1003             30 co     fo,ll,br        as
##  6 amp1…             1004             42 co     tp,pp,la,ll,is  <NA>
##  7 amp1…             1005             23 co     sp              as
##  8 amp1…             1005             19 co     sp,la,sw        as,sb,tr
```


readxl
www.rstudio.com

3.4 readxl e writexl

São pacotes à parte do **tidyverse**

```
# import .xlsx
install.packages("readxl")
library("readxl")
```

```
# export .xlsx
install.packages("writexl")
library("writexl")
```

3.4 readxl e writexl

Carrega e salva grandes arquivos de forma **mais rápida** no formato **.xlsx**

Funções **read_xlsx()** e **read_xlsx2()**

Essas funções fornecem **medidores de progresso**

E também **classificam** automaticamente o **modo** dos dados de cada coluna

A classe do objeto atribuído é **tibble**

Para salvar arquivos no formato .xlsx: **write_xlsx()** e **write_xlsx2()**

3.4 readxl e writexl

Formato .xlsx

```
# diretorio
setwd("/home/mude/data/github/minicurso-tidyverse/03_dados")
```

```
# import sites
si <- readxl::read_xlsx("ATLANTIC_AMPHIBIANS_sites.xlsx")
si
```

```
## # A tibble: 1,163 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1001             19 ab     fo,ll           as
##  2 amp1…             1002             16 co     fo,la,ll        as
##  3 amp1…             1002             14 co     fo,la,ll        as
##  4 amp1…             1002             13 co     fo,la,ll        as
##  5 amp1…             1003             30 co     fo,ll,br        as
##  6 amp1…             1004             42 co     tp,pp,la,ll,is  <NA>
##  7 amp1…             1005             23 co     sp              as
##  8 amp1…             1005             19 co     sp,la,sw        as,sb,tl
```

Importar os dados

Formato .csv

Matriz de locais

```
# import sites
si <- readr::read_csv("ATLANTIC_AMPHIBIANS_sites.csv")
si
```

Matriz de espécies

```
# import species
sp <- readr::read_csv("ATLANTIC_AMPHIBIANS_species.csv")
sp
```


TIBBLE
www.rstudio.com

3.5 tibble

O tibble (classe *tbl_df*) é um **tipo especial de data frame**

É o **formato** aconselhado para que as funções do tidyverse **funcionem**

Converter **data frame** em **tibble** usa-se a função `as_tibble()`

Converter **tibble** em **data frame** usa-se a função `as_data_frame()`

Cada variável pode ser do tipo *numbers(int, dbl)*, *character(chr)*, *logical(lgl)* ou *factor(fctr)*

3.5 tibble

Descrição dos modos das colunas através da função `glimpse()` - "espiar os dados"

```
tibble::glimpse(si)
```

```
## Rows: 1,163
## Columns: 25
## $ id                    <chr> "amp1001", "amp1002", "amp1003", "amp1004", "amp
## $ reference_number      <dbl> 1001, 1002, 1002, 1002, 1003, 1004, 1005, 1005,
## $ species_number        <dbl> 19, 16, 14, 13, 30, 42, 23, 19, 13, 1, 1, 2, 4,
## $ record                <chr> "ab", "co", "co", "co", "co", "co", "co", "co",
## $ sampled_habitat       <chr> "fo,ll", "fo,la,ll", "fo,la,ll", "fo,la,ll", "fo
## $ active_methods        <chr> "as", "as", "as", "as", "as", NA, "as", "as,sb,t
## $ passive_methods       <chr> "pt", "pt", "pt", "pt", NA, NA, NA, NA, "pt", "p
## $ complementary_methods <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,
## $ period                <chr> "mo,da,tw,ni", "mo,da,tw,ni", "mo,da,tw,ni", "mo
## $ month_start           <dbl> 9, 12, 12, 12, 7, NA, 4, 4, 4, 5, 5, 5, 5, 5, 5,
## $ year_start            <dbl> 2000, 2007, 2007, 2007, 1988, NA, 2007, 2007, 20
## $ month_finish          <dbl> 1, 5, 5, 5, 8, NA, 4, 4, 4, 7, 7, 7, 7, 7, 7, 7,
```

3.5 tibble

tibble vs data.frame

1. Nunca converte um tipo **character como factor**

```
df <- data.frame(ch = c("a", "b"), nu = 1:2)
str(df)
```

```
## 'data.frame':    2 obs. of  2 variables:
##  $ ch: Factor w/ 2 levels "a","b": 1 2
##  $ nu: int  1 2
```

```
tb <- tibble::tibble(ch = c("a", "b"), nu = 1:2)
tibble::glimpse(tb)
```

```
## Rows: 2
## Columns: 2
## $ ch <chr> "a", "b"
## $ nu <int> 1, 2
```

tibble vs data.frame

2. A indexação com colchetes retorna um **tibble**

```
df_ch <- df[, 1]
class(df_ch)
```

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

```
tb_ch <- tb[, 1]
class(tb_ch)
```

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

```
# indexacao pelo nome devolve um vetor
tb_ch <- tb$ch
class(tb_ch)
```

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

tibble vs data.frame

3. Não faz correspondência parcial, retorna **NULL** se a coluna não existe com o nome especificado

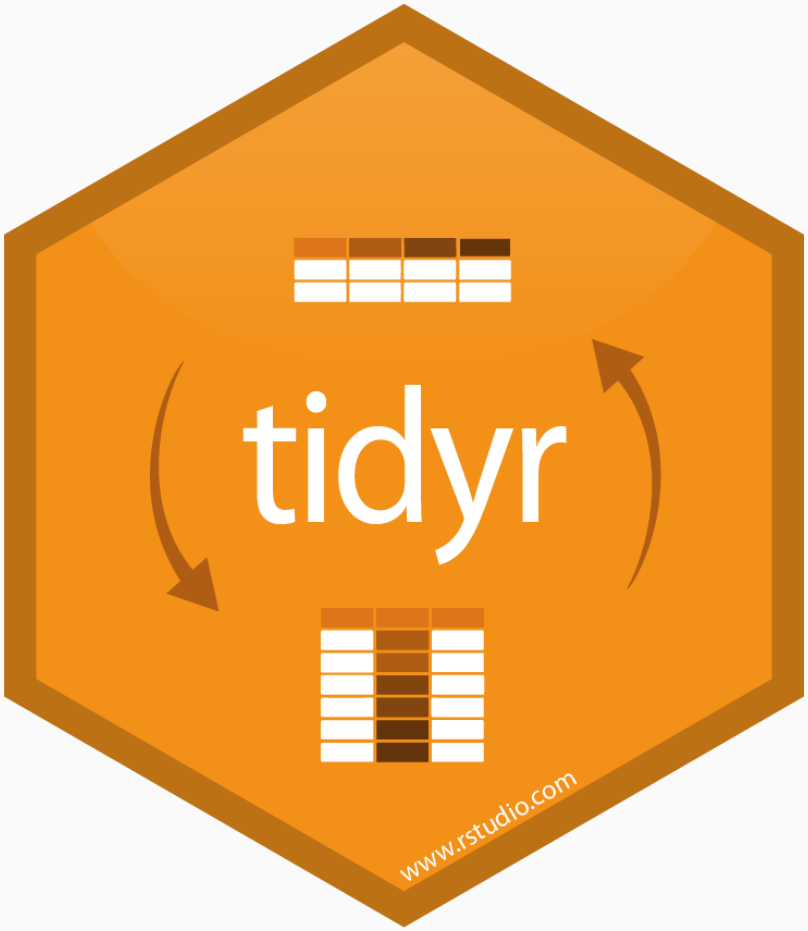
```
df$c
```

```
## [1] a b
## Levels: a b
```

```
tb$c
```

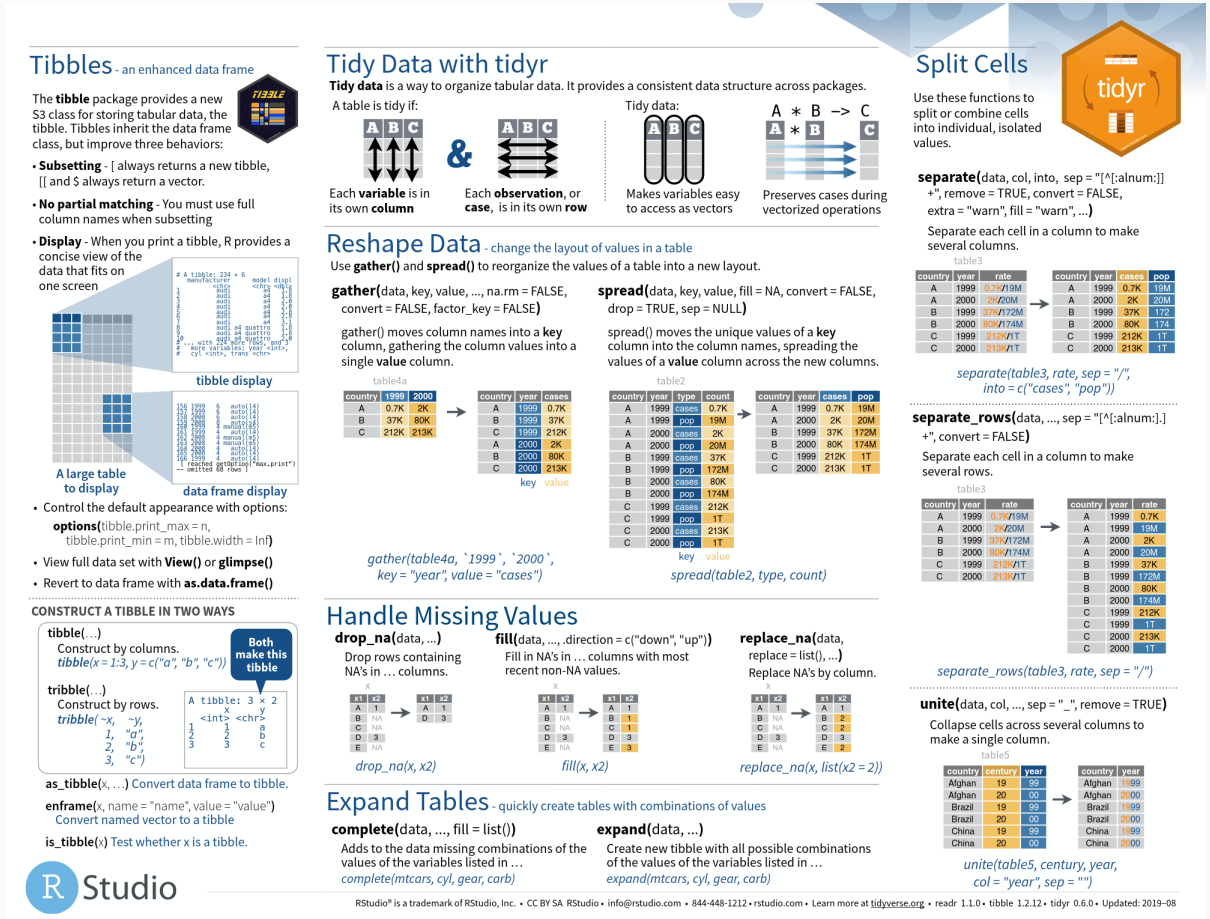
```
## NULL
```

Dúvidas?


tidyr
www.rstudio.com

3.6 tidyr

Data Import Cheatsheet



[*] https://github.com/rstudio/cheatsheets/raw/master/data-import.pdf

3.6 tidyr

Os conjuntos de dados **tidy** (organizados) são fáceis de manipular, modelar e visualizar

Um conjunto de dados está **arrumado ou não**, dependendo de como linhas, colunas e células são combinadas com observações, variáveis e valores

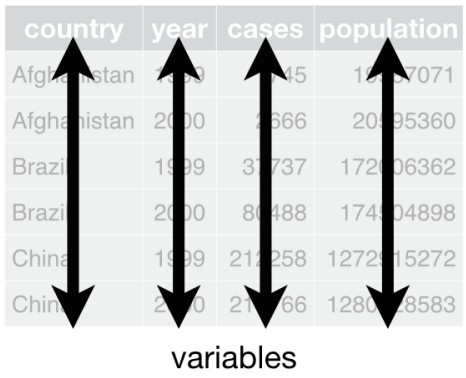
Nos dados tidy:

1 Cada variável em uma coluna

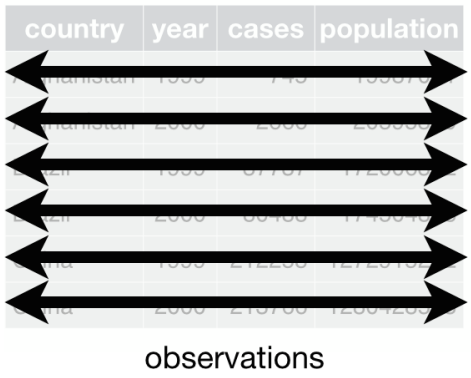
2 Cada observação em uma linha

3 Cada valor como uma célula

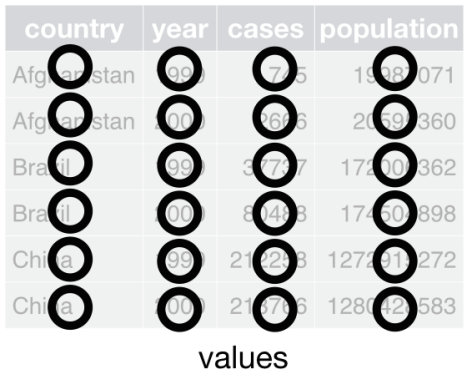
3.6 tidyr



variables



observations



values

3.6 tidyr

Funções

1 unite(): junta dados de múltiplas colunas em uma

2 separate(): separa caracteres em múlplica colunas

3 separate_rows(): separa caracteres em múlplica colunas e linhas

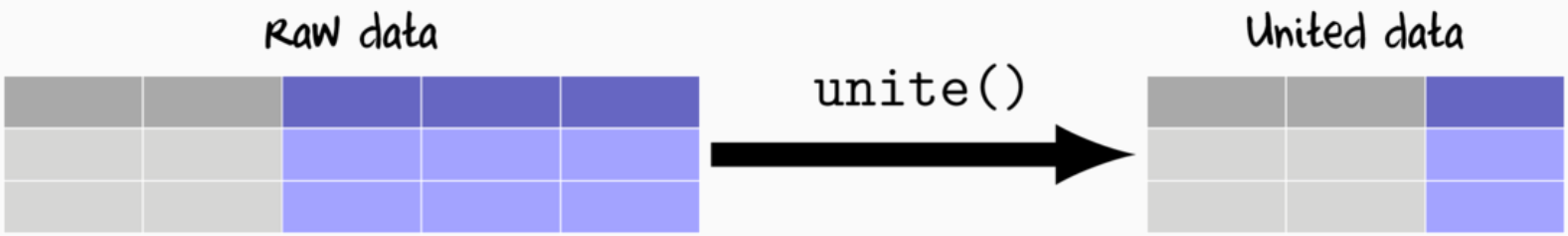
4 drop_na(): retira linhas com NA

5 replace_na(): substitui NA

6 spread() => pivot_wider(): long para wide

7 gather() => pivot_longer(): wide para long

3.6 tidyr



3.6 tidyr

1 unite

unir as colunas latirude e longitude separadas por uma vírgula

```
# sem pipes
si_unite <- tidyr::unite(si, "lat_lon", latitude:longitude, sep = ",")
si_unite$lat_lon
```

```
##    [1] "-8.68,-43.42194444"       "-3.545527,-38.857833"     "-3.574194,-38
##    [5] "-4.280555556,-38.91083333" "-9.229166667,-36.42805556" "-3.846111111,-
##    [9] "-3.8375,-40.91027778"     "-6.136944444,-35.22944444" "-6.173888889,-
##   [13] "-6.212222222,-35.21027778" "-6.234166667,-35.3125"    "-6.261666667,-
##   [17] "-6.256944444,-35.17222222" "-6.271944444,-35.17027778" "-6.253333333,-
##   [21] "-6.349722222,-35.21555556" "-6.3425,-35.19194444"     "-6.370277778,-
##   [25] "-6.382222222,-35.18305556" "-6.376666667,-35.18222222" "-6.367222222,-
##   [29] "-6.324722222,-35.14194444" "-6.335833333,-35.1175"    "-6.336388889,-
##   [33] "-6.994444444,-34.96277778" "-7.191388889,-34.85888889" "-7.064444444,-
##   [37] "-7.144166667,-34.85861111" "-7.153611111,-34.93361111" "-7.138611111,-
##   [41] "-7.148611111,-34.79666667" "-6.72425,-35.144722"      "-6.728472,-35
##   [45] "-6.743806,-35.155083"     "-6.740806,-35.171167"     "-6.718722,-35
```

3.6 tidyr

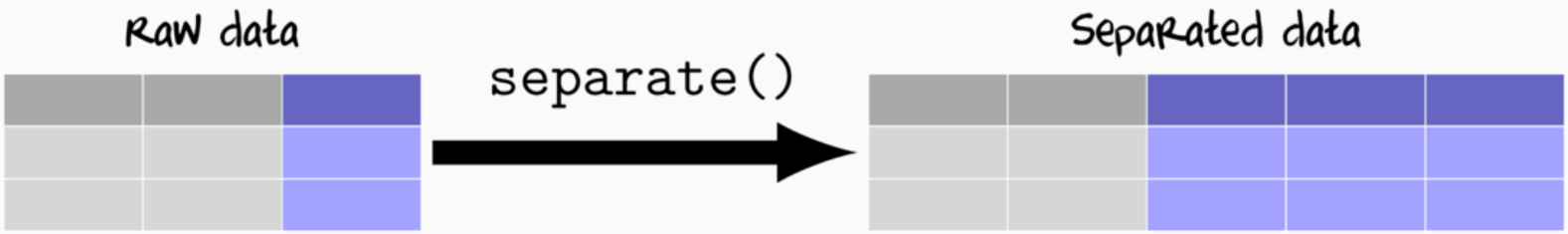
1 unite

unir as colunas latirude e longitude separadas por uma vírgula

```
# com pipes
si_unite <- si %>%
  tidyr::unite("lat_lon", latitude:longitude, sep = ",")
si_unite$lat_lon
```

```
##    [1] "-8.68,-43.42194444"       "-3.545527,-38.857833"     "-3.574194,-38
##    [5] "-4.280555556,-38.91083333" "-9.229166667,-36.42805556" "-3.846111111,-
##    [9] "-3.8375,-40.91027778"     "-6.136944444,-35.22944444" "-6.173888889,-
##   [13] "-6.212222222,-35.21027778" "-6.234166667,-35.3125"    "-6.261666667,-
##   [17] "-6.256944444,-35.17222222" "-6.271944444,-35.17027778" "-6.253333333,-
##   [21] "-6.349722222,-35.21555556" "-6.3425,-35.19194444"     "-6.370277778,-
##   [25] "-6.382222222,-35.18305556" "-6.376666667,-35.18222222" "-6.367222222,-
##   [29] "-6.324722222,-35.14194444" "-6.335833333,-35.1175"    "-6.336388889,-
##   [33] "-6.994444444,-34.96277778" "-7.191388889,-34.85888889" "-7.064444444,-
##   [37] "-7.144166667,-34.85861111" "-7.153611111,-34.93361111" "-7.138611111,-
##   [41] "-7.148611111,-34.79666667" "-6.72425,-35.144722"      "-6.728472,-35
```

3.6 tidyr



2 separate

separar os dados de "period" em quatro colunas dos seus valores

```
si_separate <- si %>%
  tidyr::separate("period", c("mo", "da", "tw", "ni"), remove = FALSE)
si_separate[, c(1, 9:13)]
```

```
## # A tibble: 1,163 x 6
##    id      period      mo    da    tw    ni
##    <chr>   <chr>       <chr> <chr> <chr> <chr>
##  1 amp1001 mo,da,tw,ni mo    da    tw    ni
##  2 amp1002 mo,da,tw,ni mo    da    tw    ni
##  3 amp1003 mo,da,tw,ni mo    da    tw    ni
##  4 amp1004 mo,da,tw,ni mo    da    tw    ni
##  5 amp1005 mo,da,ni    mo    da    ni    <NA>
##  6 amp1006 <NA>        <NA>  <NA>  <NA>  <NA>
##  7 amp1007 <NA>        <NA>  <NA>  <NA>  <NA>
##  8 amp1008 tw,ni       tw    ni    <NA>  <NA>
##  9 amp1009 mo,da,tw,ni mo    da    tw    ni
```

3 separate_rows()

separar os dados de "period" na mesma coluna e repetindo os valores

```
si_separate_row <- si %>%
  tidyr::separate_rows("period")
si_separate_row[, c(1, 9:13)]
```

```
## # A tibble: 2,650 x 6
##    id      period month_start year_start month_finish year_finish
##    <chr>   <chr>        <dbl>      <dbl>        <dbl>       <dbl>
##  1 amp1001 mo               9       2000            1        2002
##  2 amp1001 da               9       2000            1        2002
##  3 amp1001 tw               9       2000            1        2002
##  4 amp1001 ni               9       2000            1        2002
##  5 amp1002 mo              12       2007            5        2009
##  6 amp1002 da              12       2007            5        2009
##  7 amp1002 tw              12       2007            5        2009
```

4 drop_na()

remove as linhas com NA de todas as colunas

```
si_drop_na <- si %>%
  tidyr::drop_na()
si_drop_na
```

```
## # A tibble: 40 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1011             14 co     fo,tp,ll,is     as
##  2 amp1…             1028             29 co     fo              as
##  3 amp1…             1031             33 co     fo,sw           as
##  4 amp1…             1077             29 co     fo,pp,la,sw,is  as,sb
##  5 amp1…             1086              9 co     fo,la,is        as
##  6 amp1…             1086             18 co     fo,la,is        as
##  7 amp1…             1086             20 co     fo,la,is        as
##  8 amp1…             1086             18 co     fo,la,is        as
##  9 amp1…             1087             49 co     fo,tp,la,sw,is  as
```

4 drop_na()

remove as linhas com NA da coluna "active_methods"

```
si_drop_na <- si %>%
  tidyr::drop_na(active_methods)
si_drop_na
```

```
## # A tibble: 1,005 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1001             19 ab     fo,ll           as
##  2 amp1…             1002             16 co     fo,la,ll        as
##  3 amp1…             1002             14 co     fo,la,ll        as
##  4 amp1…             1002             13 co     fo,la,ll        as
##  5 amp1…             1003             30 co     fo,ll,br        as
##  6 amp1…             1005             23 co     sp              as
##  7 amp1…             1005             19 co     sp,la,sw        as,sb,tr
##  8 amp1…             1008              5 co     pp              as
##  9 amp1…             1008              2 co     pp              as
```

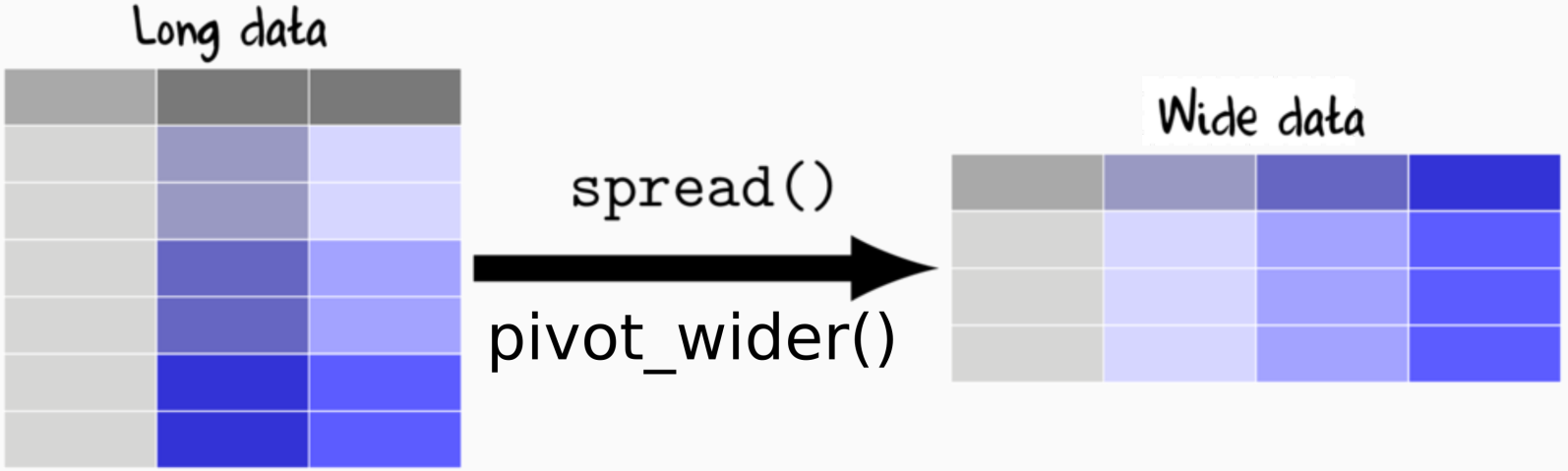
5 replace_na()

substituir os NAs da coluna "active_methods" por 0

```
si_replace_na <- si %>%
  tidyr::replace_na(list(active_methods = 0))
si_replace_na
```

```
## # A tibble: 1,163 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1001             19 ab     fo,ll           as
##  2 amp1…             1002             16 co     fo,la,ll        as
##  3 amp1…             1002             14 co     fo,la,ll        as
##  4 amp1…             1002             13 co     fo,la,ll        as
##  5 amp1…             1003             30 co     fo,ll,br        as
##  6 amp1…             1004             42 co     tp,pp,la,ll,is  0
##  7 amp1…             1005             23 co     sp              as
##  8 amp1…             1005             19 co     sp,la,sw        as,sb,tr
##  9 amp1…             1005             13 ab     fo              0
```

3.6 tidyr



6 spread()

Long para wide

1. **key**: variável categórica que irá definir os nomes das colunas
2. **value**: variável numérica que irá preencher os dados
3. **fill**: valor para preencher os NAs

```
si[, c("id", "record", "species_number")]
```

```
## # A tibble: 1,163 x 3
##    id      record species_number
##    <chr>   <chr>           <dbl>
##  1 amp1001 ab                 19
##  2 amp1002 co                 16
##  3 amp1003 co                 14
##  4 amp1004 co                 13
##  5 amp1005 co                 30
##  6 amp1006 co                 42
##  7 amp1007 co                 23
```

6 spread()

Long para wide

1. **key**: variável categórica que irá definir os nomes das colunas
2. **value**: variável numérica que irá preencher os dados
3. **fill**: valor para preencher os NAs

```
si_spread <- si[, c("id", "state_abbreviation", "species_number")] %>%
  tidyr::spread(key = state_abbreviation, value = species_number, fill = 0)
si_spread
```

```
## # A tibble: 1,163 x 24
##    id    `AR-N` `BR-AL` `BR-BA` `BR-CE` `BR-ES` `BR-GO` `BR-MG` `BR-MS` `BR-PE
##    <chr>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl
##  1 amp1…      0       0       0       0       0       0       0       0
##  2 amp1…      0       0       0      16       0       0       0       0
##  3 amp1…      0       0       0      14       0       0       0       0
##  4 amp1…      0       0       0      13       0       0       0       0
##  5 amp1…      0       0       0      30       0       0       0       0
```

6 spread()

Long para wide

1. **key**: variável categórica que irá definir os nomes das colunas
2. **value**: variável numérica que irá preencher os dados
3. **fill**: valor para preencher os NAs

```
sp[1:1000, c("id", "species", "individuals")]
```

```
## # A tibble: 1,000 x 3
##    id      species                    individuals
##    <chr>   <chr>                            <dbl>
##  1 amp1001 Adenomera sp. n.                    25
##  2 amp1001 Corythomantis greeningi             11
##  3 amp1001 Dendropsophus soaresi                1
##  4 amp1001 Dermatonotus muelleri               20
##  5 amp1001 Leptodactylus aff. syphax           10
##  6 amp1001 Leptodactylus fuscus                 2
##  7 amp1001 Leptodactylus macrosternum           3
```

6 spread()

Long para wide

1. **key**: variável categórica que irá definir os nomes das colunas
2. **value**: variável numérica que irá preencher os dados
3. **fill**: valor para preencher os NAs

```
sp_spread <- sp[1:1000, c("id", "species", "individuals")] %>%
  tidyr::replace_na(list(individuals = 0)) %>%
  tidyr::spread(key = species, value = individuals, fill = 0)
sp_spread
```

```
## # A tibble: 91 x 179
##    id    `Adelophryne ba… `Adenomera cf. … `Adenomera hyla… `Adenomera marm…
##    <chr>             <dbl>            <dbl>            <dbl>            <dbl>
##  1 amp1…                 0                0                0                0
##  2 amp1…                 0                0                0                0
##  3 amp1…                 0                0                0                0
##  4 amp1…                 0                0                0                0
```

6 pivot_wider()

Long para wide

1. **id_cols**: variável id
2. **names_from**: variável categórica que irá definir os nomes das colunas
3. **values_from**: variável numérica que irá preencher os dados
4. **values_fill**: valor para preencher os NAs

```
si_wide <- si %>%
  tidyr::pivot_wider(id_cols = id,
                     names_from = state_abbreviation,
                     values_from = species_number,
                     values_fill = list(species_number = 0))
si_wide
```

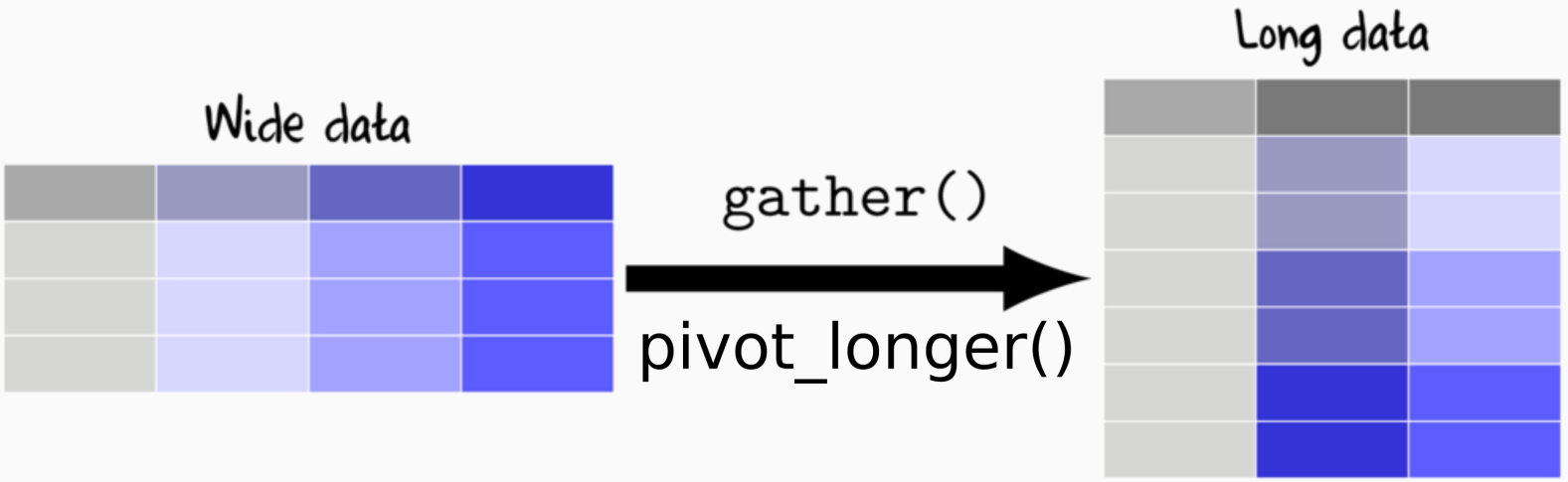
6 pivot_wider()

Long para wide

1. **id_cols**: variável id
2. **names_from**: variável categórica que irá definir os nomes das colunas
3. **values_from**: variável numérica que irá preencher os dados
4. **values_fill**: valor para preencher os NAs

```
sp_wide <- sp[1:1000, ] %>%
  tidyr::replace_na(list(individuals = 0)) %>%
  tidyr::pivot_wider(id_cols = id,
                     names_from = species,
                     values_from = individuals,
                     values_fill = list(individuals = 0))
sp_wide
```

3.6 tidyr



7 gather()

wide para long

1. **key**: variável categórica que irá definir os nomes das colunas
2. **value**: variável numérica que irá preencher os dados

```
si_gather <- si_spread %>%
  tidyr::gather(key = record, value = species_number, -id)
si_gather
```

```
## # A tibble: 26,749 x 3
##    id      record species_number
##    <chr>   <chr>           <dbl>
##  1 amp1001 AR-N                0
##  2 amp1002 AR-N                0
##  3 amp1003 AR-N                0
##  4 amp1004 AR-N                0
##  5 amp1005 AR-N                0
##  6 amp1006 AR-N                0
```

7 pivot_longer()

wide para long

1. **cols**: coluna do id
2. **names_to**: nome da coluna que receberá os nomes
3. **values_to**: nome da coluna que receberá os valores

```
si_long <- si_wide %>%
  tidyr::pivot_longer(cols = -id,
                      names_to = "record",
                      values_to = "species_number")
si_long
```

```
## # A tibble: 26,749 x 3
##    id      record species_number
##    <chr>   <chr>           <dbl>
##  1 amp1001 BR-PI              19
##  2 amp1001 BR-CE               0
##  3 amp1001 BR-RN               0
```

Exercícios

Exercício 10

Combine as colunas `country`, `state`, `state_abbreviation`, `municipality`, `site`, em uma coluna chamada `local_total` separadas por `, (vírgula + espaço)`, atribuindo o resultado a um novo objeto, utilizando o formato tidyverse

01:00

Exercício 10

Solução

```
si_local <- si %>%
  tidyr::unite("local_total",
               c(country, state, state_abbreviation, municipality, site),
               sep = ", ")
si_local$local_total
```

```
##    [1] "Brazil, Piauí, BR-PI, Canto do Buriti, Parque Nacional Serra das Confu
##    [2] "Brazil, Ceará, BR-CE, São Gonçalo do Amarante, Dunas"
##    [3] "Brazil, Ceará, BR-CE, São Gonçalo do Amarante, Jardim Botânico Mu
##    [4] "Brazil, Ceará, BR-CE, São Gonçalo do Amarante, Taíba"
##    [5] "Brazil, Ceará, BR-CE, Baturité, Serra de Baturité"
##    [6] "Brazil, Ceará, BR-CE, Quebrangulo, Reserva Biológica de Pedra Talhada'
##    [7] "Brazil, Ceará, BR-CE, Ubajara, Planalto da Ibiapaba"
##    [8] "Brazil, Ceará, BR-CE, Ubajara, Planalto da Ibiapaba"
##    [9] "Brazil, Ceará, BR-CE, Ubajara, Planalto da Ibiapaba"
##   [10] "Brazil, Rio Grande do Norte, BR-RN, São José de Mipibu, Patch A"
##   [11] "Brazil, Rio Grande do Norte, BR-RN, Arês, Patch B"
##   [12] "Brazil, Rio Grande do Norte, BR-RN, Arês, Patch C"
```

Exercício 11

Quanto indivíduos por há por família nos locais amostrados? Apenas para as 1000 primeiras linhas da tabela `sp`

05:00

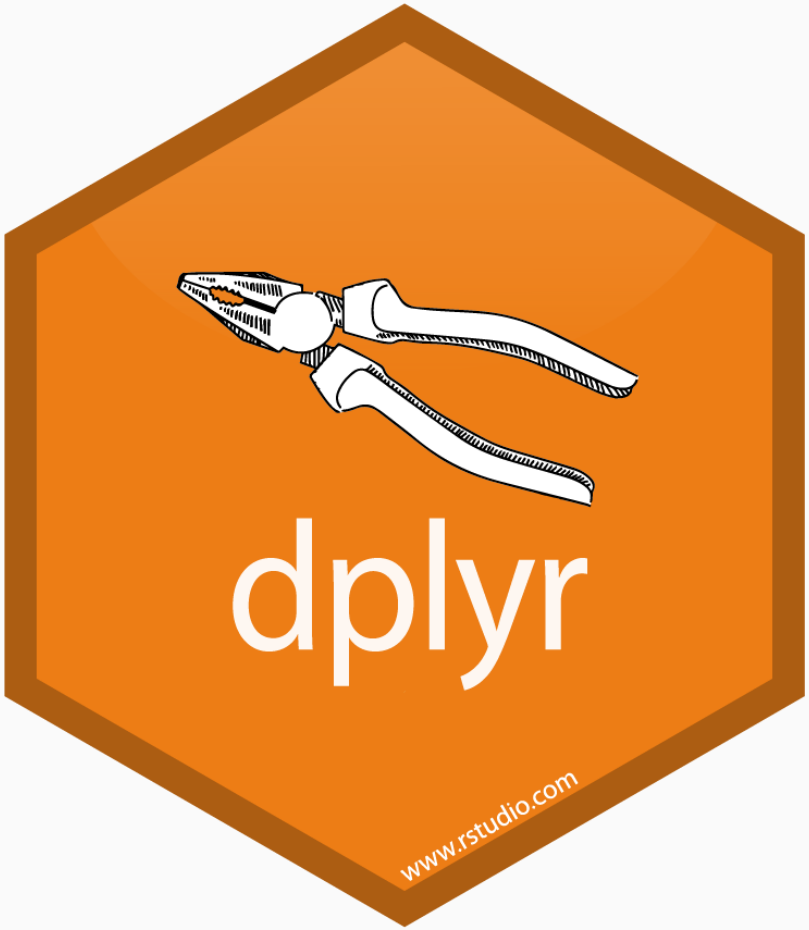
Exercício 11

Solução

```
family_wide <- sp[1:1000, ] %>%
  tidyr::replace_na(list(individuals = 0)) %>%
  tidyr::drop_na(family) %>%
  tidyr::pivot_wider(id_cols = id,
                     names_from = family,
                     values_from = individuals,
                     values_fn = list(individuals = sum),
                     values_fill = list(individuals = 0))
family_wide
```

```
## # A tibble: 91 x 12
##    id    Hylidae Microhylidae Leptodactylidae Phyllomedusidae Odontophrynidae
##    <chr>   <dbl>        <dbl>            <dbl>            <dbl>           <dbl>
##  1 amp1…      12           20              116               11              19
##  2 amp1…       0            0                0                0               0
##  3 amp1…       0            0                0                0               0
##  4 amp1…       0            0                0                0               0
##  5 amp1…       0            0                0                0               0
```

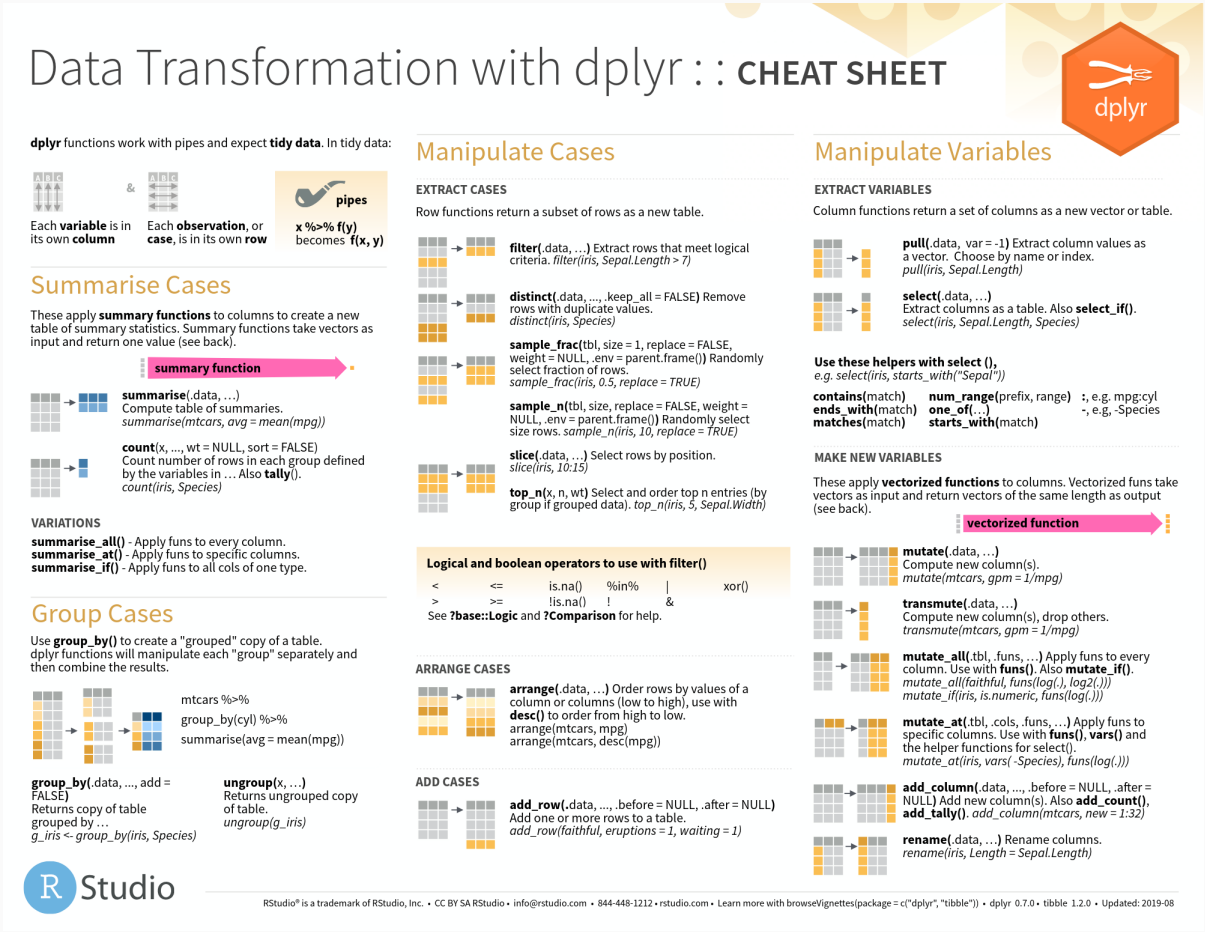
Dúvidas?


dplyr
www.rstudio.com

3.7 dplyr

Fonte: @allison_horst

Data Transformation Cheatsheet



Data Transformation with dplyr : : CHEAT SHEET

dplyr

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes **f(x, y)**

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).

summary function

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.

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)

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)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

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)

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)

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs 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 funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), 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)

RStudio

[*] https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf

3.7 dplyr

O **dplyr** é um pacote que **facilita** o trabalho com dados, com uma **gramática de manipulação** de dados **simples e flexível** (filtragem, reordenamento, seleção, entre outras)

Ele foi construído com o intuito de obter uma forma **mais rápida** e **expressiva** de tratar os dados

O **tibble** é a **versão de data frame** mais **conveniente** para **se usar** com dplyr

3.7 dplyr

Sua gramática simples contém **funções verbais** para manipulação de dados

Funções

- Verbos: mutate(), select(), filter(), arrange(), summarise(), slice(), rename(), etc.
- Sufixos: _at(), _if(), _all(), etc.
- Agrupamento: group_by() e ungroup()
- Junções: inner_join(), full_join(), left_join() e right_join()
- Funções resumo: n(), n_distinct(), first(), last(), nth(), etc.

3.7 dplyr

Sua gramática simples contém **funções verbais** para manipulação de dados

1. select(): seleciona colunas pelo nome gerando um tibble
2. pull(): seleciona uma coluna como vetor
3. rename(): muda o nome das colunas
4. mutate(): adiciona novas colunas ou adiciona resultados em colunas existentes
5. arrange(): reordenar as linhas com base nos valores de colunas
6. filter(): seleciona linhas com base em valores
7. distinc(): remove linhas com valores repetidos com base nos valores de colunas
8. slice(): seleciona linhas pelos números
9. sample_n(): amostragem aleatória de linhas
10. summarise(): agrega ou resume os dados através de funções, podendo considerar valores das colunas
11. *_join(): junta dados de duas tabelas através de uma coluna chave

3.7 dplyr

O **tibble** é sempre o **primeiro argumento** das funções verbais

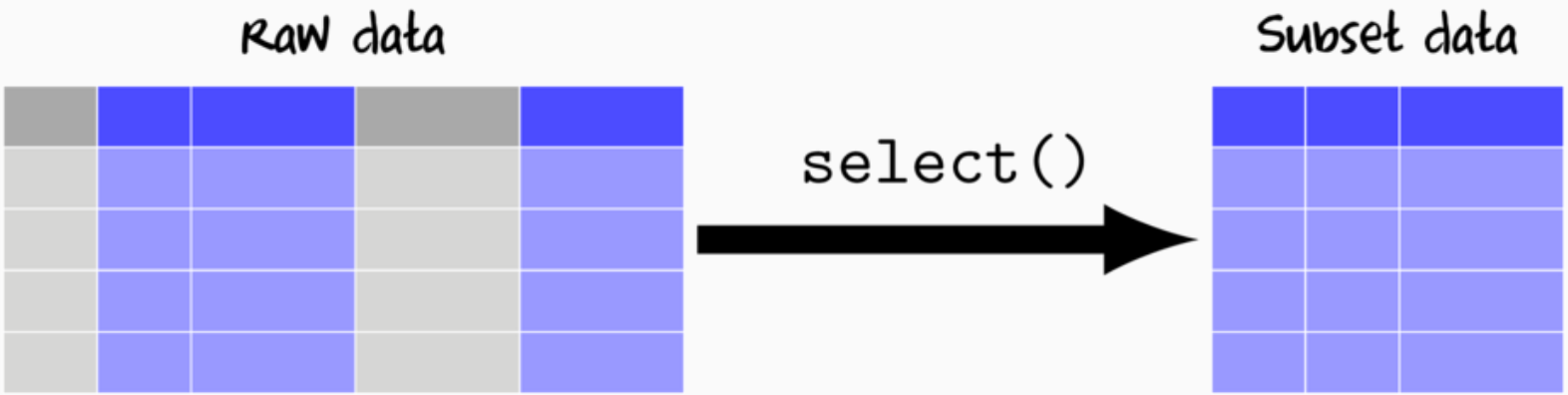
Todas seguem a mesma sintaxe:

1. tibble
2. operador pipe
3. nome da função verbal com os argumentos entre parênteses

As funções verbais **não modificam** o tibble original

```
tb_dplyr <- tb %>%
  funcao_verbal(argumento1, argumento2)
```

1 select



1 select

Seleciona colunas pelo nome

```
si_select <- si %>%
  dplyr::select(id, longitude, latitude)
si_select
```

```
## # A tibble: 1,163 x 3
##    id      longitude latitude
##    <chr>       <dbl>    <dbl>
##  1 amp1001     -43.4    -8.68
##  2 amp1002     -38.9    -3.55
##  3 amp1003     -38.9    -3.57
##  4 amp1004     -38.9    -3.52
##  5 amp1005     -38.9    -4.28
##  6 amp1006     -36.4    -9.23
##  7 amp1007     -40.9    -3.85
##  8 amp1008     -40.9    -3.83
##  9 amp1009     -40.9    -3.84
```

3.7 dplyr

1 select

Remove as colunas pelo nome

```
si_select <- si %>%
  dplyr::select(-c(id, longitude, latitude))
si_select
```

```
## # A tibble: 1,163 x 22
##    reference_number species_number record sampled_habitat active_methods passi
##               <dbl>          <dbl> <chr>  <chr>           <chr>          <chr>
##  1             1001             19 ab     fo,ll           as             pt
##  2             1002             16 co     fo,la,ll        as             pt
##  3             1002             14 co     fo,la,ll        as             pt
##  4             1002             13 co     fo,la,ll        as             pt
##  5             1003             30 co     fo,ll,br        as             <NA>
##  6             1004             42 co     tp,pp,la,ll,is  <NA>           <NA>
##  7             1005             23 co     sp              as             <NA>
##  8             1005             19 co     sp,la,sw        as,sb,tr       <NA>
##  9             1005             13 ab     fo              <NA>           pt
```

1 select

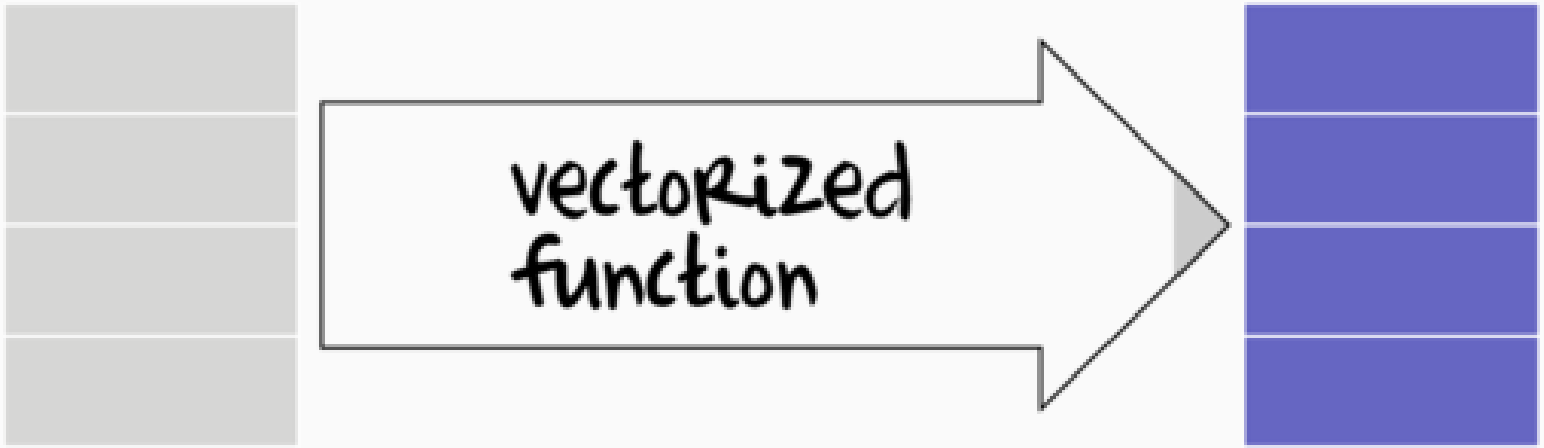
Seleciona colunas por um padrão

```
#  starts_with(), ends_with() e contains()
si_select <- si %>%
  dplyr::select(contains("sp"))
si_select
```

```
## # A tibble: 1,163 x 1
##    species_number
##             <dbl>
##  1             19
##  2             16
##  3             14
##  4             13
##  5             30
##  6             42
##  7             23
##  8             19
```

2 pull

Seleciona uma coluna como vetor



2 pull

Seleciona uma coluna como vetor

```
# coluna para vetor
si_pull <- si %>%
  dplyr::pull(id)
si_pull
```

```
##    [1] "amp1001" "amp1002" "amp1003" "amp1004" "amp1005" "amp1006" "amp1007" "
##   [12] "amp1012" "amp1013" "amp1014" "amp1015" "amp1016" "amp1017" "amp1018" "
##   [23] "amp1023" "amp1024" "amp1025" "amp1026" "amp1027" "amp1028" "amp1029" "
##   [34] "amp1034" "amp1035" "amp1036" "amp1037" "amp1038" "amp1039" "amp1040" "
##   [45] "amp1045" "amp1046" "amp1047" "amp1048" "amp1049" "amp1050" "amp1051" "
##   [56] "amp1056" "amp1057" "amp1058" "amp1059" "amp1060" "amp1061" "amp1062" "
##   [67] "amp1067" "amp1068" "amp1069" "amp1070" "amp1071" "amp1072" "amp1073" "
##   [78] "amp1078" "amp1079" "amp1080" "amp1081" "amp1082" "amp1083" "amp1084" "
##   [89] "amp1089" "amp1090" "amp1091" "amp1092" "amp1093" "amp1094" "amp1095" "
##  [100] "amp1100" "amp1101" "amp1102" "amp1103" "amp1104" "amp1105" "amp1106" "
##  [111] "amp1111" "amp1112" "amp1113" "amp1114" "amp1115" "amp1116" "amp1117" "
```

2 pull

Seleciona uma coluna como vetor

```
# coluna para vetor
si_pull <- si %>%
  dplyr::pull(species_number)
si_pull
```

```
##    [1] 19 16 14 13 30 42 23 19 13  1  1  2  4  4  6  5  8  2  5  1  2  2  1  2
##   [40]  2  3  5  2  1  1  2  8  8  7  8  9  6  5 21 24 19 21 14 14  8 12 25 23
##   [79]  6 16  9 21 27 19 20 26 10  6 30 35 28 27 32 29  5  4 23 33 18 21 26 14
##  [118]  8  8  2  2  1  1 13  2  6  3  2  2  2  1  2  9  3  1  1 21  1 11  9  5
##  [157] 11  9 15  9  6 12  8  5 32 18 23 19 32 24 20 17 18 16 24 19 21 19 23 39
##  [196] 15  6  9  4 14 15 34 44 33 43 18  5  5  5  5 11 40 35 37  8 15 12 15 14
##  [235] 14 22 14 32 15 18 10  4 18 16 18 26  9 18 20 18 21 49 34  9  5  3  7  3
##  [274]  8 16 14  9 36  8 10  7 41 11 30 19  5  7 20  7  6  5  2  3  3  9  5  5
##  [313] 22 22 29 12 22 14 40 41 13 19 22  4 14 11  2  5  3 34 41  2  7 13 50 53
##  [352] 20 76 15 32 26 13  7  4  5  1  6 25 26 12 41 32 36 36 14 11 12 19 69 32
##  [391]  5  2 25 16  6 50 23 24 40  9 16 13 31  9 16 22 10  8 30 27 14 13 58 24
```

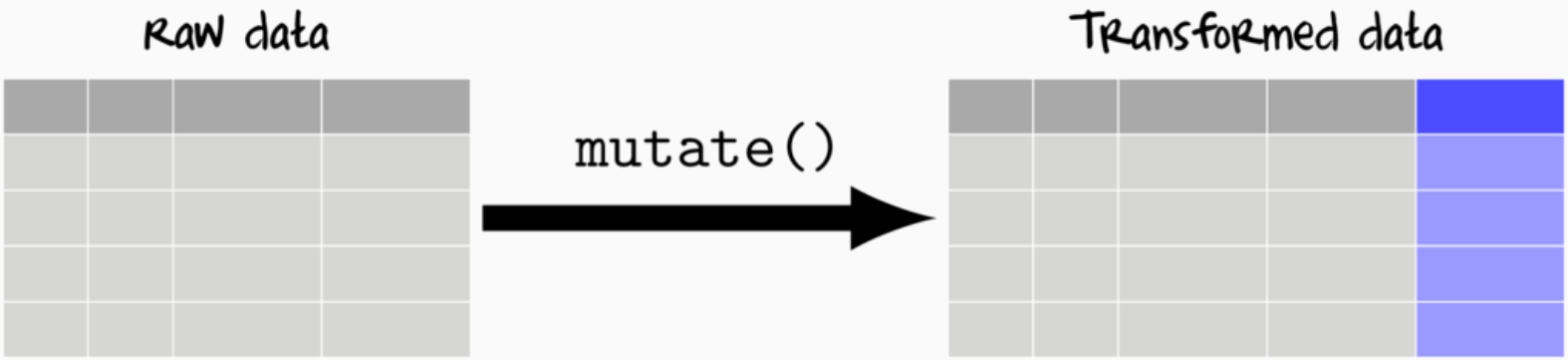
3 rename

Renomeia colunas

```
# rename sp column
si_rename <- si %>%
  dplyr::rename(sp = species_number)
si_rename
```

```
## # A tibble: 1,163 x 25
##    id    reference_number    sp record sampled_habitat active_methods passive_
##    <chr>            <dbl> <dbl> <chr>  <chr>           <chr>          <chr>
##  1 amp1…             1001    19 ab     fo,ll           as             pt
##  2 amp1…             1002    16 co     fo,la,ll        as             pt
##  3 amp1…             1002    14 co     fo,la,ll        as             pt
##  4 amp1…             1002    13 co     fo,la,ll        as             pt
##  5 amp1…             1003    30 co     fo,ll,br        as             <NA>
##  6 amp1…             1004    42 co     tp,pp,la,ll,is  <NA>           <NA>
##  7 amp1…             1005    23 co     sp              as             <NA>
##  8 amp1…             1005    19 co     sp,la,sw        as,sb,tr       <NA>
```

4 mutate



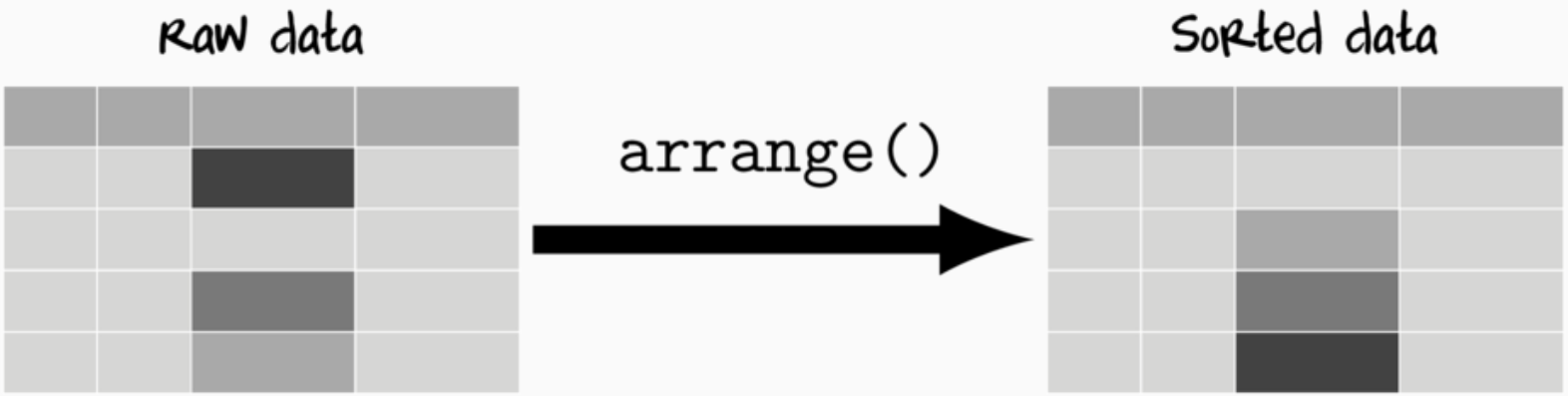
4 mutate

Adiciona colunas novas ou vindas de operações

```
# add column record as factor
si_mutate <- si %>%
  dplyr::mutate(record_factor = as.factor(record))
si_mutate$record_factor
```

```
##    [1] ab co co co co co co co ab ab ab ab ab ab ab ab ab ab ab ab ab ab ab ab
##   [40] co co co co co co co co co co co co co co co co co co co co co co ab ab ab ab
##   [79] co co co co co co co co co co co ab co co co ab co ab ab ab co co ab ab ab
##  [118] co co co co co co co co co co co ab ab ab ab ab ab ab ab ab ab ab ab ab ab
##  [157] ab ab co co co co co co co co co co co co co co co co co co co co co co co co
##  [196] co co co co co co co co co co co co ab ab ab ab co co co co ab co co co co
##  [235] co co co co co ab ab ab co co co co co co co co co co co co co co co co co co
##  [274] co co co co co ab ab ab co co co co co co co co co co co co co co co co co
##  [313] co co ab co co co co co ab ab ab ab ab ab ab ab co co co co co ab co
##  [352] co co co co co co co co co co co co co co co co co ab co ab co ab ab ab co co
##  [391] co co co ab ab co ab co co ab ab ab ab ab ab co co ab co ab ab co co
```

5 arrange



5 arrange

Ordenar de forma crescente pela coluna **species_number**

```
si_arrange <- si %>%
  dplyr::arrange(species_number)
si_arrange
```

```
## # A tibble: 1,163 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1006              1 ab     fo              <NA>
##  2 amp1…             1006              1 ab     fo              <NA>
##  3 amp1…             1006              1 ab     fo              <NA>
##  4 amp1…             1006              1 ab     fo              <NA>
##  5 amp1…             1006              1 ab     fo              <NA>
##  6 amp1…             1008              1 co     pp              as
##  7 amp1…             1008              1 co     pp              as
##  8 amp1…             1041              1 co     re,du,br        as,qs
##  9 amp1…             1041              1 co     re,du,br        as,qs
```

5 arrange

Ordenar de forma decrescente pela coluna **species_number**

```
si_arrange <- si %>%
  dplyr::arrange(desc(species_number))
si_arrange
```

```
## # A tibble: 1,163 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1037             80 co     fo,tp,pp,is     as,tr
##  2 amp1…             1132             76 co     <NA>            as,tr,qs
##  3 amp1…             1145             69 co     <NA>            <NA>
##  4 amp1…             1241             66 ab     fo,tp,la,sw,ll… as
##  5 amp1…             1230             65 co     fo,tp,pp,la,ll… as
##  6 amp2…             1359             55 co     <NA>            <NA>
##  7 amp1…             1227             54 co     <NA>            <NA>
##  8 amp2…             1320             54 co     fo,pp,ll,is,br  as,tr
##  9 amp1…             1118             53 co     pp,sw,is        as,tr
```

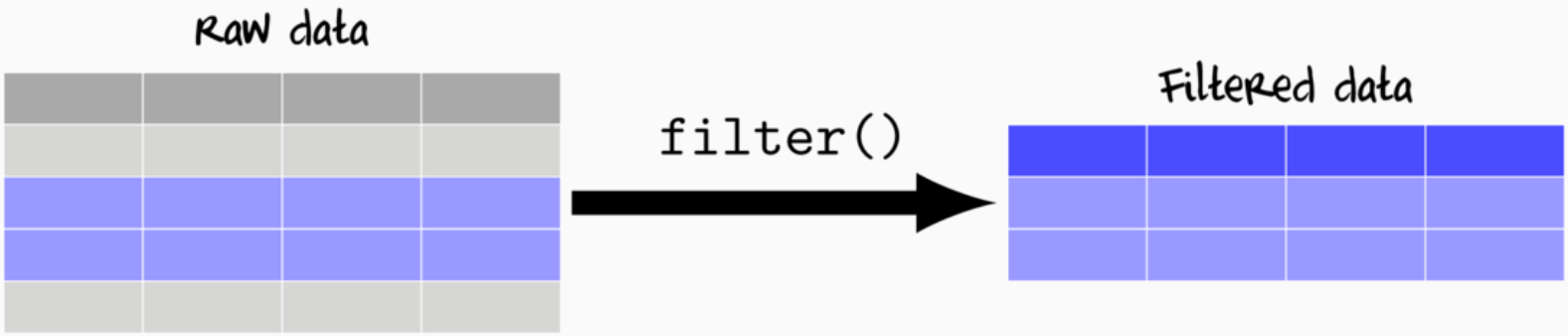
5 arrange

Ordenar de forma decrescente pela coluna **species_number**

```
si_arrange <- si %>%
  dplyr::arrange(-species_number)
si_arrange
```

```
## # A tibble: 1,163 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1037             80 co     fo,tp,pp,is     as,tr
##  2 amp1…             1132             76 co     <NA>            as,tr,qs
##  3 amp1…             1145             69 co     <NA>            <NA>
##  4 amp1…             1241             66 ab     fo,tp,la,sw,ll… as
##  5 amp1…             1230             65 co     fo,tp,pp,la,ll… as
##  6 amp2…             1359             55 co     <NA>            <NA>
##  7 amp1…             1227             54 co     <NA>            <NA>
##  8 amp2…             1320             54 co     fo,pp,ll,is,br  as,tr
##  9 amp1…             1118             53 co     pp,sw,is        as,tr
```

6 filter



6 filter

operadores: >, >=, <, <=, ==, !=, is.na, !is.na, %in%

Filtrar para locais com mais de 5 espécies

```
si_filter <- si %>%
  dplyr::filter(species_number > 5)
si_filter
```

Filtrar para locais centre 1 e 5 espécies

```
si_filter <- si %>%
  dplyr::filter(between(species_number, 1, 5))
si_filter
```

6 filter

Filtrar para locais com NA no métodos passivos

```
si_filter <- si %>%
  dplyr::filter(is.na(passive_methods))
si_filter
```

Filtrar para locais sem NA no métodos passivos

```
si_filter <- si %>%
  dplyr::filter(!is.na(passive_methods))
si_filter
```

6 filter

Filtrar para locais sem NA no métodos ativos **E** passivos

```
si_filter <- si %>%
  dplyr::filter(!is.na(active_methods) & !is.na(passive_methods))
si_filter
```

6 filter

Filtrar para locais amostrados com mais de 5 espécies **E** em São Paulo

```
si_filter <- si %>%
  dplyr::filter(species_number > 5 & state_abbreviation == "BR-SP")
si_filter
```

Filtrar para locais amostrados com mais de 5 espécies **OU** em São Paulo

```
si_filter <- si %>%
  dplyr::filter(species_number > 5 | state_abbreviation == "BR-SP")
si_filter
```

7 distinct

Remove linhas duplicadas considerando a coluna número de espécies

```
si_distinct <- si %>%
  dplyr::distinct(species_number)
si_distinct
```

```
## # A tibble: 58 x 1
##    species_number
##             <dbl>
##  1             19
##  2             16
##  3             14
##  4             13
##  5             30
##  6             42
##  7             23
```

7 distinct

Remove linhas duplicadas considerando a coluna número de espécies e mantém as demais

```
si_distinct <- si %>%
  dplyr::distinct(species_number, .keep_all = TRUE)
si_distinct
```

```
## # A tibble: 58 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1001             19 ab     fo,ll           as
##  2 amp1…             1002             16 co     fo,la,ll        as
##  3 amp1…             1002             14 co     fo,la,ll        as
##  4 amp1…             1002             13 co     fo,la,ll        as
##  5 amp1…             1003             30 co     fo,ll,br        as
##  6 amp1…             1004             42 co     tp,pp,la,ll,is  <NA>
##  7 amp1…             1005             23 co     sp              as
```

8 slice

Selecionar linhas 1 a 10

```
si_slice <- si %>%
  dplyr::slice(1:10)
si_slice
```

```
## # A tibble: 10 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1001             19 ab     fo,ll           as
##  2 amp1…             1002             16 co     fo,la,ll        as
##  3 amp1…             1002             14 co     fo,la,ll        as
##  4 amp1…             1002             13 co     fo,la,ll        as
##  5 amp1…             1003             30 co     fo,ll,br        as
##  6 amp1…             1004             42 co     tp,pp,la,ll,is  <NA>
##  7 amp1…             1005             23 co     sp              as
##  8 amp1…             1005             19 co     sp,la,sw        as,sb,tr
##  9 amp1…             1005             13 ab     fo              <NA>
```

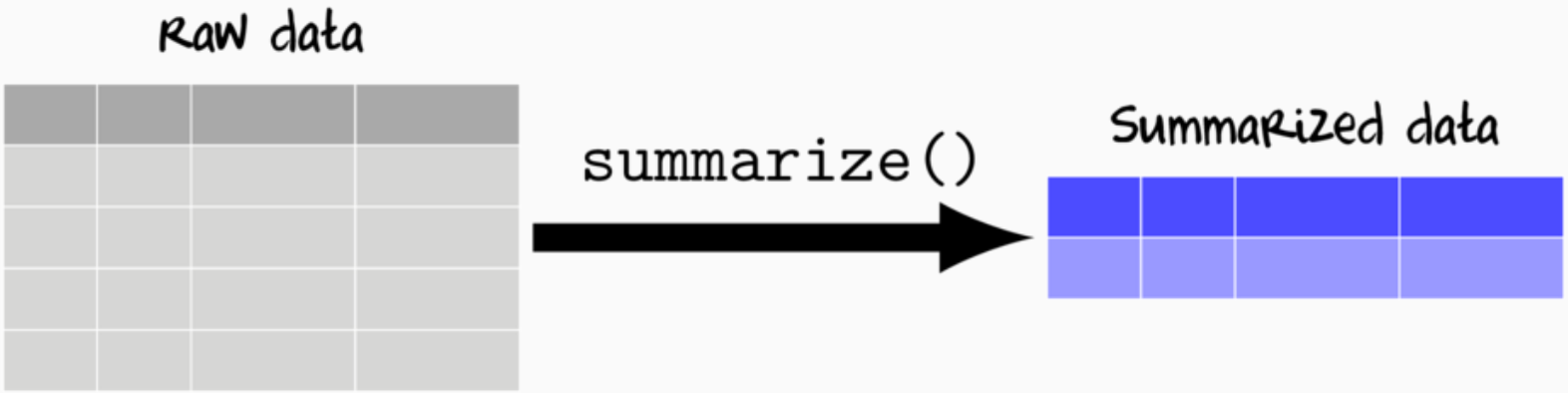
9 n_sample

Amostrar 100 linhas aleatoriamente

```
si_sample_n <- si %>%
  dplyr::sample_n(100)
si_sample_n
```

```
## # A tibble: 100 x 25
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1181              3 co     fo              <NA>
##  2 amp1…             1316              3 ab     tp              as
##  3 amp1…             1073             28 co     fo,tp,sp,pp,la… as
##  4 amp2…             1367              7 co     pp              as
##  5 amp1…             1050             20 co     fo,la,is        as
##  6 amp1…             1181              6 co     sw,ll,br        as
##  7 amp1…             1311              5 co     fo,sw,ll,is     as,tr
##  8 amp1…             1240             22 ab     tp,pp,sw        as,sb
##  9 amp2…             1323             12 co     sw              as,sb
```

10 summarise



10 summarise

Média e desvio padrão do número de espécies total

```
si_summarise <- si %>%
  dplyr::summarise(mean_sp = mean(species_number),
                   sd_sp = sd(species_number))
si_summarise
```

```
## # A tibble: 1 x 2
##   mean_sp sd_sp
##     <dbl> <dbl>
## 1    15.2  11.2
```

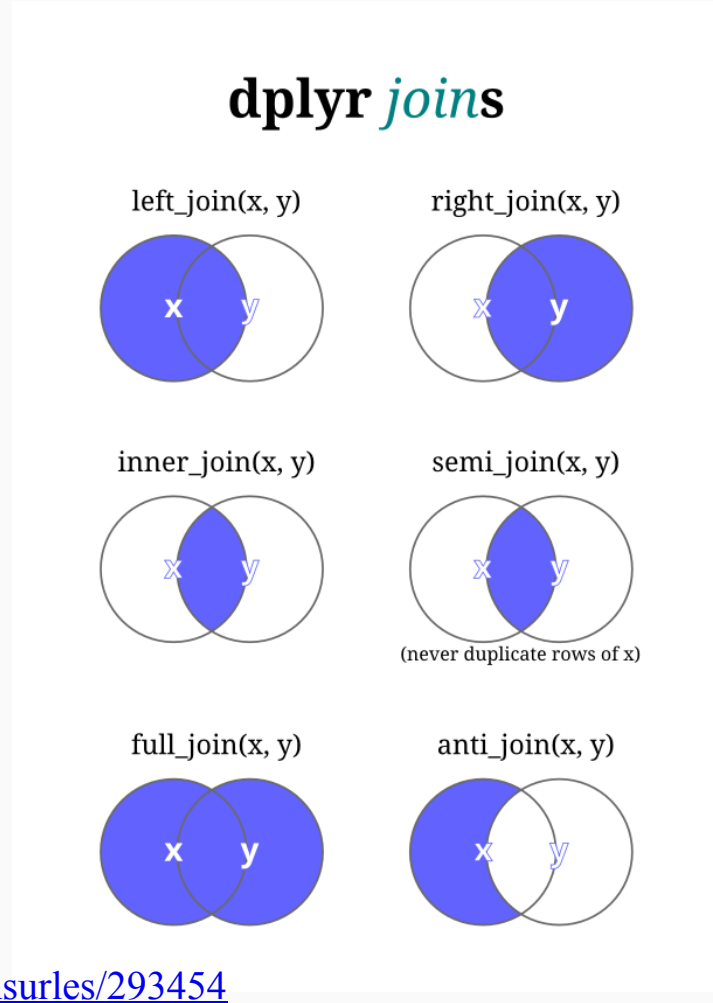
10 summarise

Média e desvio padrão do número de espécies por país

```
si_summarise_group <- si %>%
  dplyr::group_by(country) %>%
  dplyr::summarise(mean_sp = mean(species_number),
                   sd_sp = sd(species_number))
si_summarise_group
```

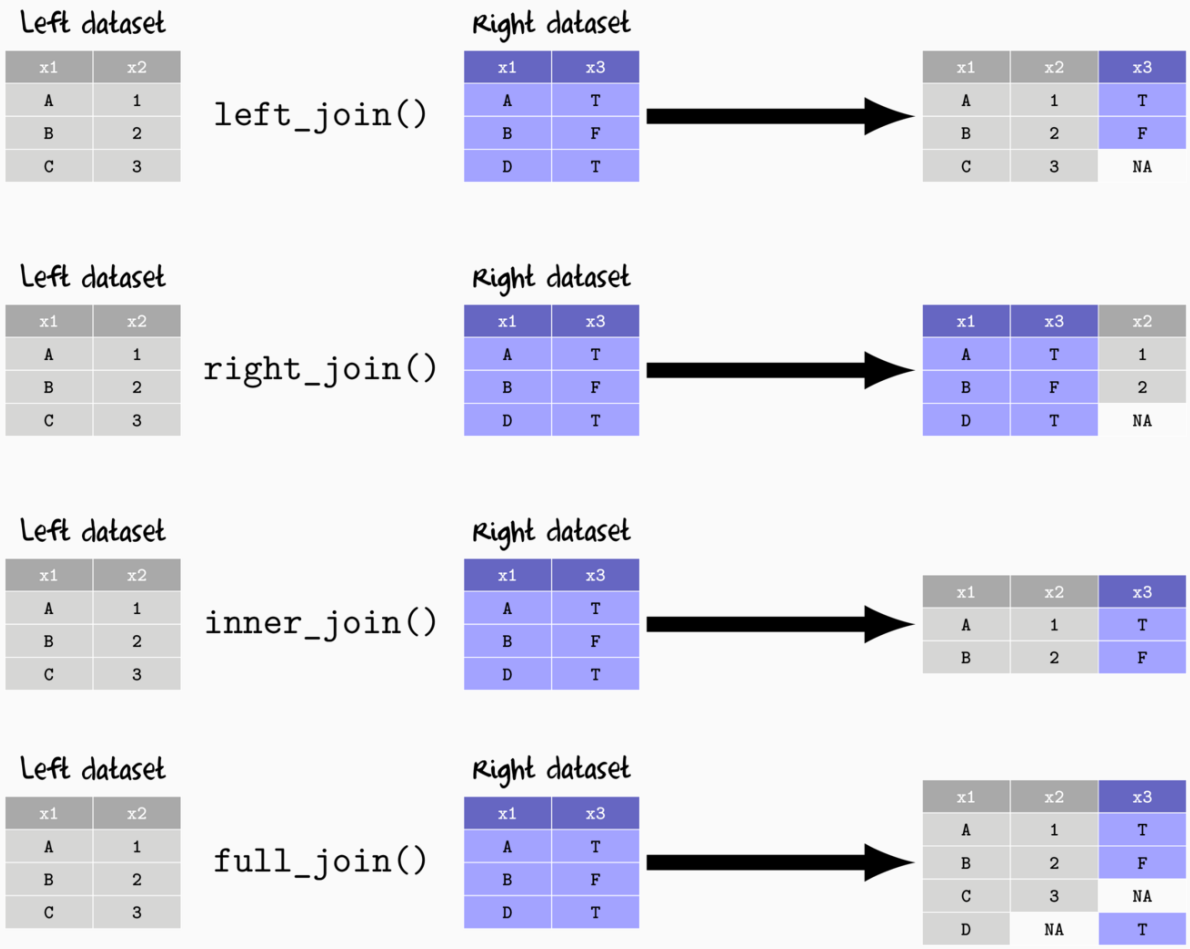
```
## # A tibble: 3 x 3
##   country   mean_sp sd_sp
##   <chr>       <dbl> <dbl>
## 1 Argentina    8.36  7.73
## 2 Brazil      15.2  11.3
## 3 Paraguay    28     2.92
```

11 *_join

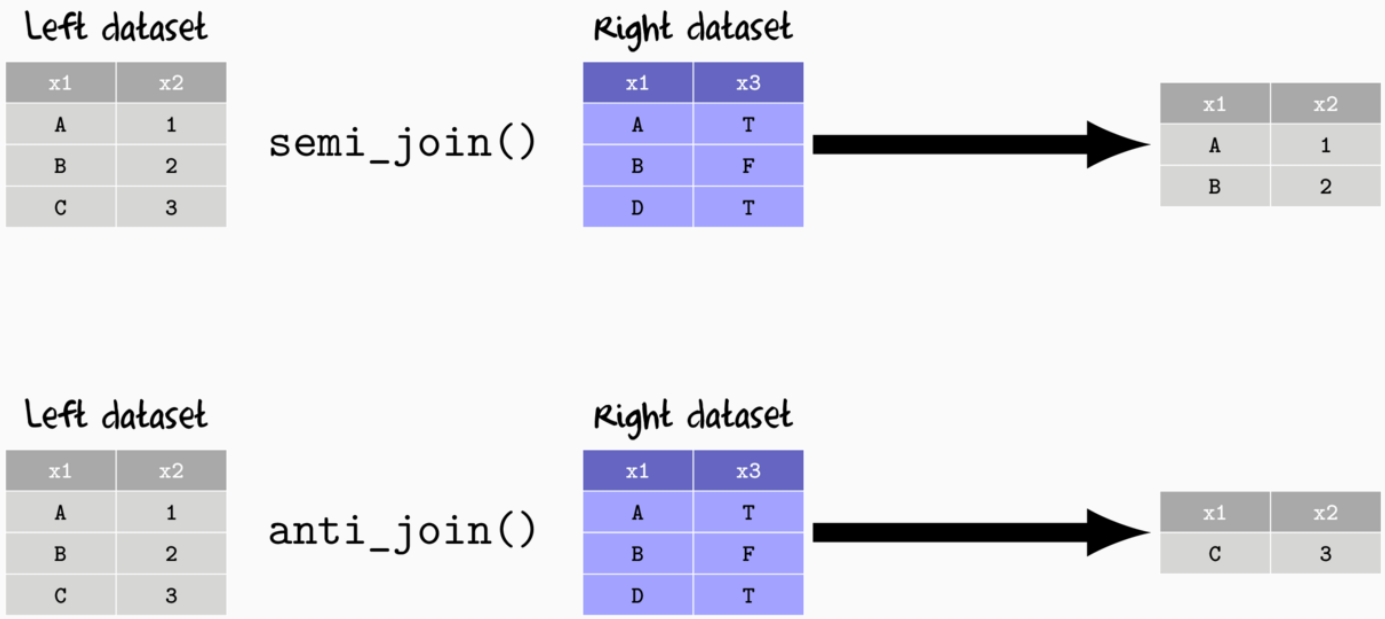


[*] https://rpubs.com/williamsurles/293454

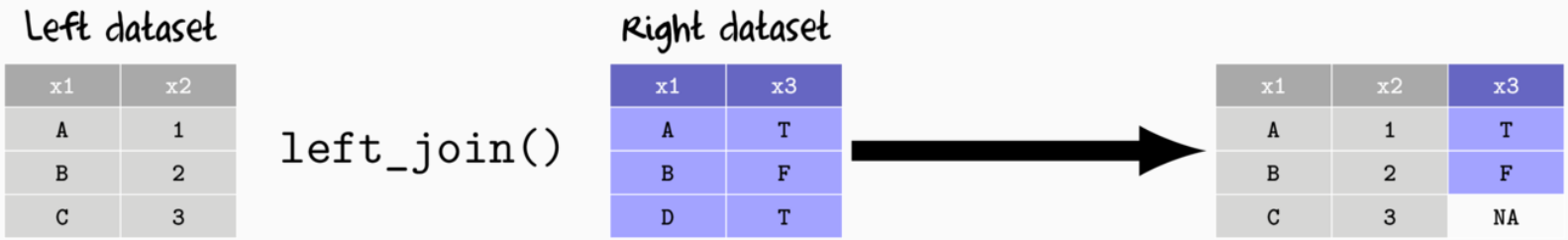
11 *_join


Left dataset
x1 x2
A 1
B 2
C 3
left_join()
Right dataset
x1 x3
A T
B F
D T
x1 x2 x3
A 1 T
B 2 F
C 3 NA
Left dataset
x1 x2
A 1
B 2
C 3
right_join()
Right dataset
x1 x3
A T
B F
D T
x1 x3 x2
A T 1
B F 2
D T NA
Left dataset
x1 x2
A 1
B 2
C 3
inner_join()
Right dataset
x1 x3
A T
B F
D T
x1 x2 x3
A 1 T
B 2 F
Left dataset
x1 x2
A 1
B 2
C 3
full_join()
Right dataset
x1 x3
A T
B F
D T
x1 x2 x3
A 1 T
B 2 F
C 3 NA
D NA T

11 *_join


Left dataset
x1 x2
A 1
B 2
C 3
semi_join()
Right dataset
x1 x3
A T
B F
D T
x1 x2
A 1
B 2
Left dataset
x1 x2
A 1
B 2
C 3
anti_join()
Right dataset
x1 x3
A T
B F
D T
x1 x2
C 3

11 *_join


Left dataset
x1 x2
A 1
B 2
C 3
left_join()
Right dataset
x1 x3
A T
B F
D T
x1 x2 x3
A 1 T
B 2 F
C 3 NA

11 *_join

Adicionar longitute e latitude na tabela de espécies

Selecionar as colunas que preciso da tabela **si**

```
si_coord <- si %>%
  select(id, longitude, latitude)
si_coord
```

```
## # A tibble: 1,163 x 3
##    id      longitude latitude
##    <chr>       <dbl>    <dbl>
##  1 amp1001     -43.4    -8.68
##  2 amp1002     -38.9    -3.55
##  3 amp1003     -38.9    -3.57
##  4 amp1004     -38.9    -3.52
##  5 amp1005     -38.9    -4.28
##  6 amp1006     -36.4    -9.23
```

11 *_join

Adicionar longitute e latitude na tabela de espécies

Join com as tabelas pela coluna "id"

```
# join dos dados
sp_join <- sp %>%
  left_join(si_coord, by = "id")
sp_join
```

```
## # A tibble: 17,619 x 12
##    id     order superfamily family    subfamily   genus    species      valid_
##    <chr>  <chr> <chr>       <chr>     <chr>       <chr>    <chr>        <chr>
##  1 amp10… <NA>  <NA>        <NA>      <NA>        <NA>     Adenomera s… <NA>
##  2 amp10… Anura <NA>        Hylidae   Lophyohyli… Corytho… Corythomant… Coryth
##  3 amp10… Anura <NA>        Hylidae   Dendropsop… Dendrop… Dendropsoph… Dendro
##  4 amp10… Anura <NA>        Microhyl… Gastrophry… Dermato… Dermatonotu… Derma
##  5 amp10… <NA>  <NA>        <NA>      <NA>        <NA>     Leptodactyl… <NA>
```

11 *_join

Adicionar longitute e latitude na tabela de espécies

Nome das colunas

```
colnames(sp_join)
```

```
##  [1] "id"          "order"       "superfamily" "family"      "subfamily"   "ge
##  [9] "individuals" "endemic"     "longitude"   "latitude"
```

E não termina nunca...

3.7 dplyr

Sufixos: _at(), _if(), _all()

Realiza operações dependente de confições

```
sp_wide_rename <- sp_wide %>%
  dplyr::rename_at(vars(contains(" ")),
                   list(~stringr::str_replace_all(., " ", "_"))) %>%
  dplyr::rename_all(list(~stringr::str_to_lower(.)))
sp_wide_rename
```

```
## # A tibble: 91 x 179
##    id    adenomera_sp._n. corythomantis_g… dendropsophus_s… dermatonotus_mu… l
##    <chr>            <dbl>            <dbl>            <dbl>            <dbl>
##  1 amp1…               25               11                1               20
##  2 amp1…                0                0                0                0
##  3 amp1…                0                0                0                0
##  4 amp1…                0                0                0                0
##  5 amp1…                0                0                0                0
##  6 amp1…                0                0                0                0
##  7 amp1…                0                0                0                0
```

Mas o mais importante é a ideia de

encadeamento das funções

3.7 dplyr

Permite manipular os dados de forma simples

```
da <- si %>%
  dplyr::select(id, state_abbreviation, species_number)
da
```

```
## # A tibble: 1,163 x 3
##    id      state_abbreviation species_number
##    <chr>   <chr>                       <dbl>
##  1 amp1001 BR-PI                          19
##  2 amp1002 BR-CE                          16
##  3 amp1003 BR-CE                          14
##  4 amp1004 BR-CE                          13
##  5 amp1005 BR-CE                          30
##  6 amp1006 BR-CE                          42
##  7 amp1007 BR-CE                          23
##  8 amp1008 BR-CE                          19
##  9 amp1009 BR-CE                          13
## 10 amp1010 BR-RN                           1
## # … with 1,153 more rows
```

3.7 dplyr

Permite manipular os dados de forma simples

```
da <- si %>%
  dplyr::select(id, state_abbreviation, species_number) %>%
  dplyr::filter(species_number > 5)
da
```

```
## # A tibble: 934 x 3
##    id      state_abbreviation species_number
##    <chr>   <chr>                       <dbl>
##  1 amp1001 BR-PI                          19
##  2 amp1002 BR-CE                          16
##  3 amp1003 BR-CE                          14
##  4 amp1004 BR-CE                          13
##  5 amp1005 BR-CE                          30
##  6 amp1006 BR-CE                          42
##  7 amp1007 BR-CE                          23
##  8 amp1008 BR-CE                          19
##  9 amp1009 BR-CE                          13
## 10 amp1015 BR-RN                           6
```

3.7 dplyr

Permite manipular os dados de forma simples

```
da <- si %>%
  dplyr::select(id, state_abbreviation, species_number) %>%
  dplyr::filter(species_number > 5) %>%
  dplyr::group_by(state_abbreviation) %>%
  dplyr::summarise(nsp_state_abb = n())
da
```

```
## # A tibble: 23 x 2
##    state_abbreviation nsp_state_abb
##    <chr>                      <int>
##  1 AR-N                           7
##  2 BR-AL                          3
##  3 BR-BA                         63
##  4 BR-CE                          8
##  5 BR-ES                         33
##  6 BR-GO                          4
##  7 BR-MG                         91
##  8 BR-MS                          1
```

3.7 dplyr

Permite manipular os dados de forma simples

```
da <- si %>%
  dplyr::select(id, state_abbreviation, species_number) %>%
  dplyr::filter(species_number > 5) %>%
  dplyr::group_by(state_abbreviation) %>%
  dplyr::summarise(nsp_state_abb = n()) %>%
  dplyr::arrange(nsp_state_abb)
da
```

```
## # A tibble: 23 x 2
##    state_abbreviation nsp_state_abb
##    <chr>                      <int>
##  1 BR-MS                          1
##  2 BR-PI                          1
##  3 PY-13                          1
##  4 PY-14                          1
##  5 PY-2                           1
##  6 PY-4                           1
##  7 PY-7                           1
```

Exercícios

Exercício 12

Adicione essas novas colunas `alt_log, tem_log e pre_log`, que são a operação `log10` das colunas `altitude, temperature e precipitation` e atribua ao mesmo objeto `si` utilizando o formato tidyverse

01:30

Exercício 12

Solução

```
# solucao
si <- si %>%
  dplyr::mutate(alt_log = log10(altitude),
                tem_log = log10(temperature),
                pre_log = log10(precipitation))
si[, 25:28]
```

```
## # A tibble: 1,163 x 4
##    precipitation alt_log tem_log pre_log
##            <dbl>   <dbl>   <dbl>   <dbl>
##  1           853    2.73    1.40    2.93
##  2          1318    1.18    1.42    3.12
##  3          1248    1.46    1.42    3.10
##  4          1376    1.40    1.42    3.14
##  5          1689    2.88    1.33    3.23
##  6          1249    2.87    1.31    3.10
##  7          1520    2.94    1.33    3.18
##  8          1474    2.94    1.33    3.17
```

Exercício 13

Ordene os dados em forma decrescente pela coluna `altitude`, atribuindo o resultado a um novo objeto utilizando o formato tidyverse

01:30

Exercício 13

Solução

```
# solucao
si_ar <- si %>%
  dplyr::arrange(-altitude)
si_ar
```

```
## # A tibble: 1,163 x 28
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1102             25 co     fo,sw,ll,is,br  as
##  2 amp1…             1142             14 co     fo,ll,is,br     as
##  3 amp1…             1090             18 co     la              as
##  4 amp1…             1120             31 co     fo,tp,pp,sw,is… as
##  5 amp1…             1273             19 co     tp,pp,sw,ll,is… as
##  6 amp1…             1068             15 co     tp              as
##  7 amp1…             1059             43 co     sp,sw,is        as
##  8 amp1…             1219             46 co     fo,pp,sw,ll,is… as
##  9 amp1…             1109             40 co     fo,pp,la,is,br  as
## 10 amp1…             1283             15 co     fo,tp,pp,ll,is  as
```

Exercício 14

Filtre as linhas com `altitude` maior que 1000 mm, `temperature` menor que 15 ºC e `precipitation` maior que 1000 mm, atribuindo o resultado a um novo objeto utilizando o formato tidyverse

01:30

Exercício 14

Solução

```
# solucao
si_fi <- si %>%
  dplyr::filter(altitude > 1000,
                temperature < 15,
                precipitation > 1000)
si_fi
```

```
## # A tibble: 25 x 28
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1090             18 co     la              as
##  2 amp1…             1120             31 co     fo,tp,pp,sw,is… as
##  3 amp1…             1102             25 co     fo,sw,ll,is,br  as
##  4 amp1…             1142             14 co     fo,ll,is,br     as
##  5 amp1…             1167              9 ab     <NA>            as
##  6 amp1…             1219             46 co     fo,pp,sw,ll,is… as
##  7 amp1…             1224             17 ab     tp,la,is        as
##  8 amp1…             1273             19 co     tp,pp,sw,ll,is… as
```

Exercício 15

Amostre 200 linhas aleatoriamente com número de espécies maior que 15 espécies, atribuindo o resultado a um novo objeto utilizando o formato tidyverse

01:30

Exercício 15

Solução

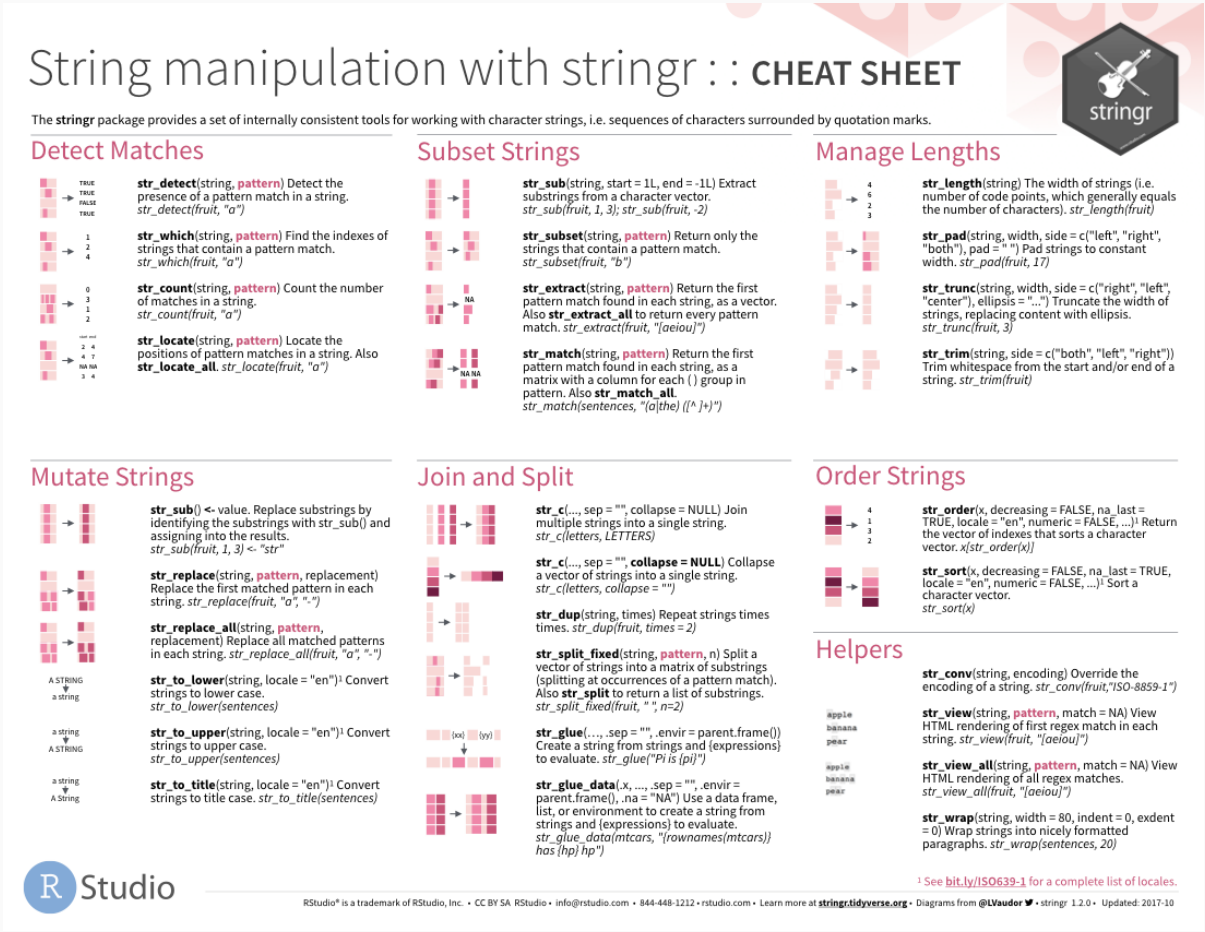
```
# solucao
si_sa <- si %>%
  dplyr::filter(species_number > 15) %>%
  dplyr::sample_n(200)
si_sa
```

```
## # A tibble: 200 x 28
##    id    reference_number species_number record sampled_habitat active_methods
##    <chr>            <dbl>          <dbl> <chr>  <chr>           <chr>
##  1 amp1…             1310             27 co     <NA>            <NA>
##  2 amp1…             1153             45 co     <NA>            as,tr,qs
##  3 amp1…             1073             30 co     fo,tp,sp,pp,la… as
##  4 amp1…             1203             23 ab     fo,ll           qs
##  5 amp1…             1283             21 co     fo,tp,pp,ll,is  as
##  6 amp1…             1193             16 co     sp              as,sb
##  7 amp1…             1087             34 co     fo,tp,is        as
##  8 amp1…             1098             20 co     fo,tp,pp,la,is  as
##  9 amp1…             1086             18 co     fo,la,is        as
```


stringr
www.rstudio.com

3.8 stringr

Work with Strings Cheatsheet



String manipulation with stringr : : **CHEAT SHEET**

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches

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

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

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

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

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) ([^]+)")*

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)*

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", "-")*

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

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

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

Join and Split

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

str_c(..., sep = "", **collapse = NULL**) 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}")*

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*

str_view(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

str_view_all(string, **pattern**, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See bit.ly/ISO639-1 for a complete list of locales.

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **stringr.tidyverse.org** • Diagrams from **@LVaudor** • stringr 1.2.0 • Updated: 2017-10

[*] https://github.com/rstudio/cheatsheets/raw/master/strings.pdf

3.8 stringr

Pacote para a manipulação de strings

Comprimento

```
str_length("abc")
```

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

Substituição

```
str_sub("abc", 3)
```

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

3.8 stringr

Inserir espaço em branco

```
str_pad("abc", width = 4, side = "left")
```

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

Inserir espaço em branco

```
str_pad("abc", width = 4, side = "right")
```

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

Remover espaço em branco do começo, final ou ambos

```
str_trim(" abc ")
```

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

3.8 stringr

Minúsculas e maiúsculas

```
str_to_upper("abc")
```

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

```
str_to_title("abc")
```

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

```
str_to_title("aBc")
```

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

3.8 stringr

Ordenarção

```
le <- sample(letters, 26, rep = TRUE)
le
```

```
##  [1] "s" "g" "i" "s" "u" "k" "k" "l" "q" "b" "t" "g" "q" "d" "m" "s" "a" "q" "
```

```
str_sort(le)
```

```
##  [1] "a" "b" "b" "c" "d" "d" "g" "g" "h" "i" "k" "k" "l" "m" "q" "q" "q" "s" "
```

```
str_sort(le, dec = TRUE)
```

```
##  [1] "z" "y" "y" "u" "u" "t" "s" "s" "s" "q" "q" "q" "m" "l" "k" "k" "i" "h" "
```

3.8 stringr

Extrair

```
str_extract("abc", "b")
```

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

Substituir

```
str_replace("abc", "a", "y")
```

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

Separação

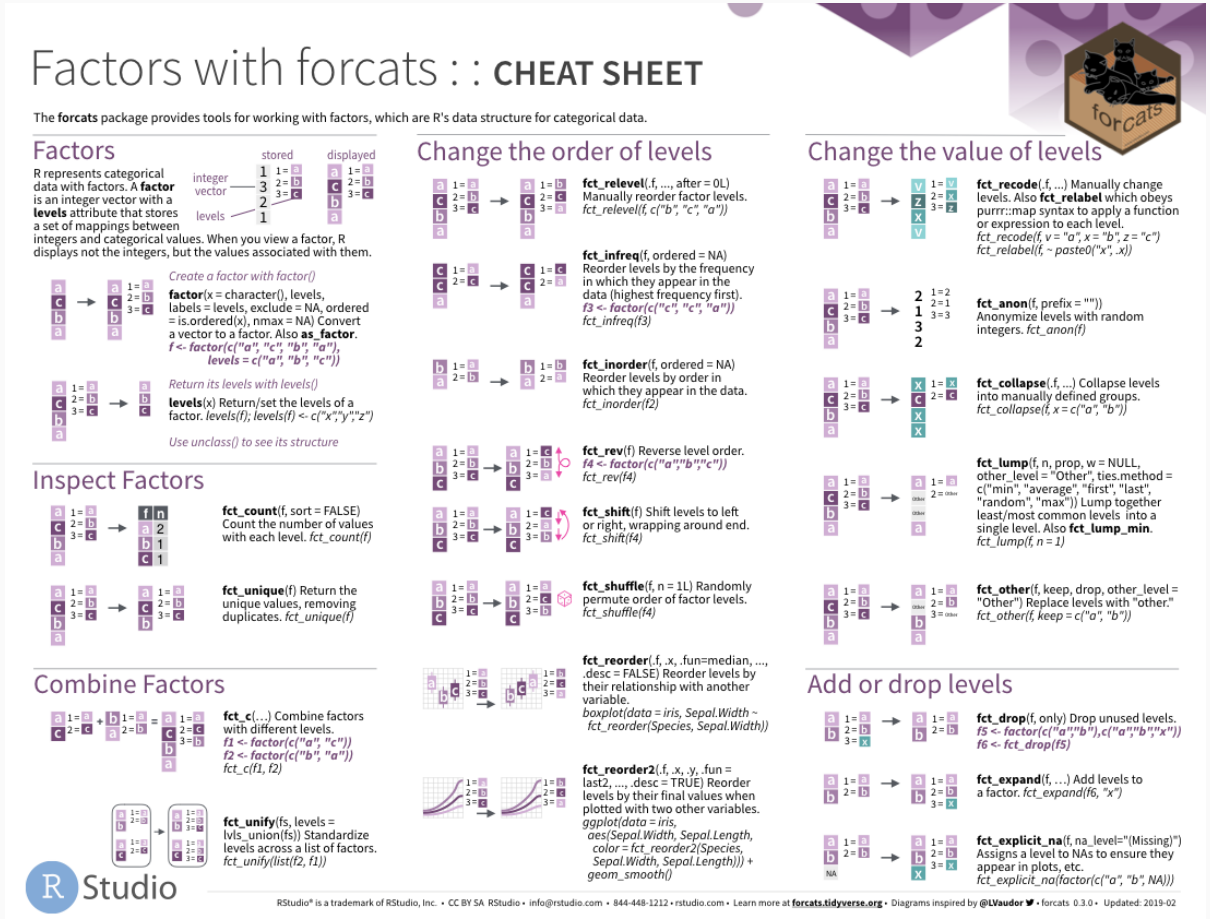
```
str_split("a-b-c", "-")
```

```
## [[1]]
## [1] "a" "b" "c"
```


forcats
www.rstudio.com

3.9 forcats

Factors with forcats Cheatsheet



[*] https://github.com/rstudio/cheatsheets/raw/master/factors.pdf

3.9 forcats

as_factor(): cria fatores

```
# fixar amostragem
set.seed(1)

# cria um fator
fa <- sample(c("alto", "medio", "baixo"), 30, rep = TRUE) %>%
  forcats::as_factor()
fa
```

```
##  [1] alto  baixo alto  medio alto  baixo baixo medio medio baixo baixo alto  a
## [21] baixo alto  alto  alto  alto  medio alto  alto  medio medio
## Levels: alto baixo medio
```

3.9 forcats

fct_recode(): muda o nome dos níveis

```
# muda o nome dos niveis
fa_recode <- fa %>%
  forcats::fct_recode(a = "alto", m = "medio", b = "baixo")
fa_recode
```

```
##  [1] a b a m a b b m m b b a a a m m m m b a b a a a a m a a m m
## Levels: a b m
```

3.9 forcats

fct_rev(): inverte os níveis

```
# inverte os niveis
fa_rev <- fa_recode %>%
  forcats::fct_rev()
fa_rev
```

```
##  [1] a b a m a b b m m b b a a a m m m m b a b a a a a m a a m m
## Levels: m b a
```

3.9 forcats

fct_relevel(): especifica a classificação de um ou mais níveis

```
# especifica a classificacao de um nivel
fa_relevel <- fa_recode %>%
  forcats::fct_relevel(c("a", "m", "b"))
fa_relevel
```

```
##  [1] a b a m a b b m m b b a a a m m m m b a b a a a a m a a m m
## Levels: a m b
```

3.9 forcats

fct_inorder(): ordem em que aparece

```
# ordem em que aparece
fa_inorder <- fa_recode %>%
  forcats::fct_inorder()
fa_inorder
```

```
##  [1] a b a m a b b m m b b a a a m m m m b a b a a a a m a a m m
## Levels: a b m
```

3.9 forcats

fct_infreq(): ordem (decrescente) de frequência

```
# ordem (decrescente) de frequencia
fa_infreq <- fa_recode %>%
  forcats::fct_infreq()
fa_infreq
```

```
##  [1] a b a m a b b m m b b a a a m m m m b a b a a a a m a a m m
## Levels: a m b
```

3.9 forcats

fct_lump(): agregação de níveis raros em um nível

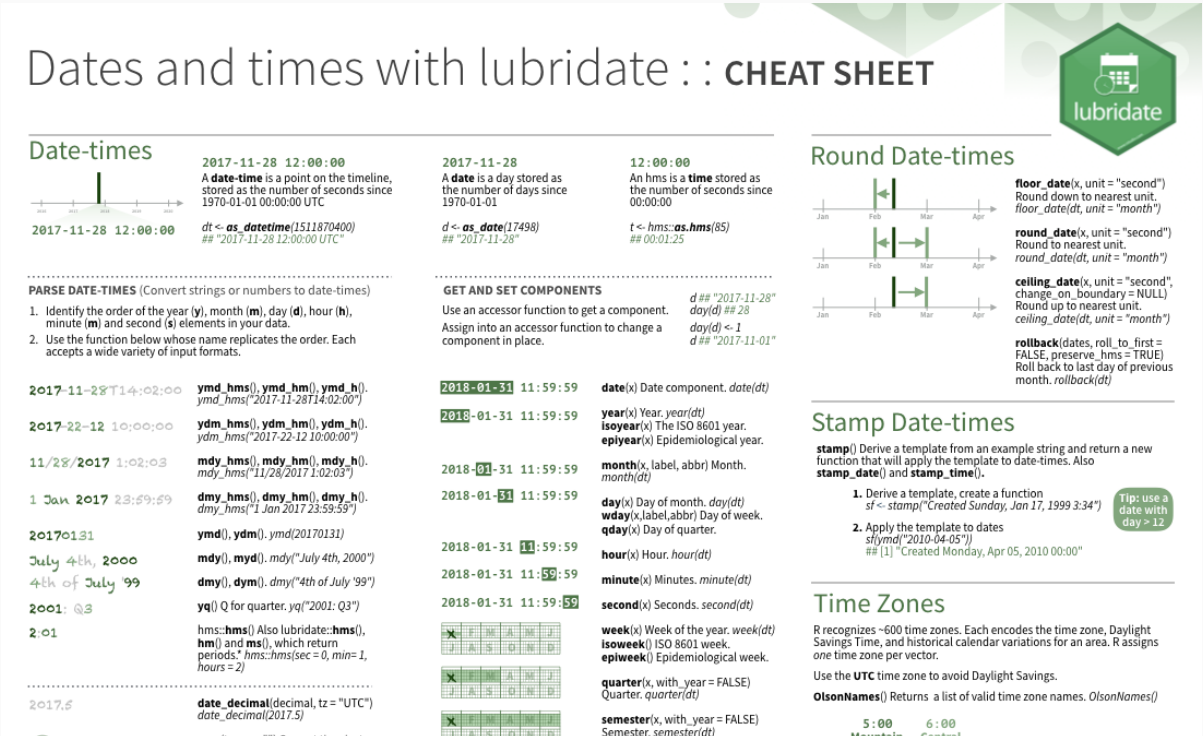
```
# agregacao de niveis raros em um nivel
fa_lump <- fa_recode %>%
  forcats::fct_lump()
fa_lump
```

```
##  [1] a     Other a     m     a     Other Other m     m     Other Other a     a
## [21] Other a     a     a     a     m     a     a     m     m
## Levels: a m Other
```


lubridate
www.rstudio.com

3.10 lubridate

Dates and Times Cheatsheet



Dates and times with lubridate : : **CHEAT SHEET**

lubridate

Date-times

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

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

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)*
2017.5	**date_decimal**(decimal, tz = "UTC") *date_decimal(2017.5)*
	now(tzone = "") Current time in tz (defaults to system tz). *now()*
	today(tzone = "") Current date in a tz (defaults to system tz). *today()*
	fast_strptime() Faster strptime. *fast_strptime('9/1/01', '%y/%m/%d')*
	parse_date_time() Easier strptime. *parse_date_time("9/1/01", "ymd")*

GET AND SET COMPONENTS

Use an accessor function to get a component.
d ## "2017-11-28"
day(d) ## 28

Assign into an accessor function to change a component in place.
day(d) <- 1
d ## "2017-11-01"

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

Round Date-times

floor_date(x, unit = "second") Round down to nearest unit. *floor_date(dt, unit = "month")*

round_date(x, unit = "second") Round to nearest unit. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit. *ceiling_date(dt, unit = "month")*

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. *rollback(dt)*

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date**() and **stamp_time**().

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

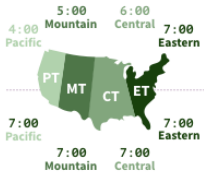
Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. *OlsonNames()*



with_tz(time, tzone = "") Get the **same date-time** in a new time zone (a new clock time). *with_tz(dt, "US/Pacific")*

force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). *force_tz(dt, "US/Pacific")*

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **lubridate.tidyverse.org** • lubridate 1.6.0 • Updated: 2017-12

[*] https://github.com/rstudio/cheatsheets/raw/master/lubridate.pdf

3.10 lubridate

Pacote à parte do tidyverse

Instalar

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

Carregar

```
# load
library(lubridate)
```

3.10 lubridate

As funções `date()` e `as_date()` assumem que a ordem segue o padrão da língua inglesa: ano-mês-dia (ymd)

date: classe para datas

```
# string
data_string <- "2020-04-24"
data_string
```

```
## [1] "2020-04-24"
```

```
class(data_string)
```

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

3.10 lubridate

As funções `date()` e `as_date()` assumem que a ordem segue o padrão da língua inglesa: ano-mês-dia (ymd)

date: classe para datas

```
# criar um objeto com a classe data
data_date <- lubridate::date(data_string)
data_date
```

```
## [1] "2020-04-24"
```

```
class(data_date)
```

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

3.10 lubridate

As funções `date()` e `as_date()` assumem que a ordem segue o padrão da língua inglesa: ano-mês-dia (ymd)

date: classe para datas

```
# criar um objeto com a classe data
data_date <- lubridate::as_date(data_string)
data_date
```

```
## [1] "2020-04-24"
```

```
class(data_date)
```

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

3.10 lubridate

Datas no padrão da língua portuguesa: dia-mês-ano (dmy)

date: classe para datas

```
# string
data_string <- "24-04-2020"
data_string
```

```
## [1] "24-04-2020"
```

```
class(data_string)
```

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

3.10 lubridate

Datas no padrão da língua portuguesa: dia-mês-ano (dmy)

date: classe para datas

```
# criar um objeto com a classe data
data_date <- lubridate::dmy(data_string)
data_date
```

```
## [1] "2020-04-24"
```

```
class(data_date)
```

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

3.10 lubridate

Além de outros diversos formatos

```
# formatos
lubridate::dmy(24042020)
```

```
## [1] "2020-04-24"
```

```
lubridate::dmy("24042020")
```

```
## [1] "2020-04-24"
```

```
lubridate::dmy("24/04/2020")
```

```
## [1] "2020-04-24"
```

```
lubridate::dmy("24.04.2020")
```

```
## [1] "2020-04-24"
```

3.10 lubridate

Especificar horários

```
# especificar horarios
lubridate::dmy_h(2404202013)
```

```
## [1] "2020-04-24 13:00:00 UTC"
```

```
lubridate::dmy_hm(2404202013 35)
```

```
## [1] "2020-04-24 13:35:00 UTC"
```

```
lubridate::dmy_hms(240420201335 35)
```

```
## [1] "2020-04-24 13:35:35 UTC"
```

3.10 lubridate

Funções úteis

- second() - extrai os segundos
- minute() - extrai os minutos
- hour() - extrai a hora
- wday() - extrai o dia da semana
- mday() - extrai o dia do mês
- month() - extrai o mês
- year() - extrai o ano

[*] http://material.curso-r.com/lubridate/

3.10 lubridate

Extração

```
# criar
data <- lubridate::dmy_hms(240420201335 35)
data
```

```
## [1] "2020-04-24 13:35:35 UTC"
```

```
# extrair
lubridate::second(data)
```

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

```
lubridate::day(data)
```

```
## [1] 24
```

```
lubridate::month(data)
```

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

3.10 lubridate

Extração

```
# criar
data <- lubridate::dmy_hms(240420201335 35)
data
```

```
## [1] "2020-04-24 13:35:35 UTC"
```

```
# extrair
lubridate::wday(data)
```

```
## [1] 6
```

```
lubridate::wday(data, label = TRUE)
```

```
## [1] Fri
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

3.10 lubridate

Inclusão

```
# criar
data <- lubridate::dmy(24042020)
data
```

```
## [1] "2020-04-24"
```

```
# inlcuir
lubridate::hour(data) <- 13
data
```

```
## [1] "2020-04-24 13:00:00 UTC"
```

3.10 lubridate

Extrair a data no instante da execução

```
# extrair a data no instante da execucao
lubridate::today()
```

```
## [1] "2020-04-28"
```

```
# extrair a data e horario no instante da execucao
lubridate::now()
```

```
## [1] "2020-04-28 01:04:28 -03"
```

quando eu preparava essa aula...

3.10 lubridate

Fusos horários

```
# agora
agora <- lubridate::ymd_hms(lubridate::now(), tz = "America/Sao_Paulo")
agora
```

```
## [1] "2020-04-28 01:04:28 -03"
```

```
# verificar os tz
OlsonNames()
```

```
##   [1] "Africa/Abidjan"                "Africa/Accra"                  "A
##   [4] "Africa/Algiers"                "Africa/Asmara"                 "A
##   [7] "Africa/Bamako"                 "Africa/Bangui"                 "A
##  [10] "Africa/Bissau"                 "Africa/Blantyre"               "A
##  [13] "Africa/Bujumbura"              "Africa/Cairo"                  "A
##  [16] "Africa/Ceuta"                  "Africa/Conakry"                "A
##  [19] "Africa/Dar_es_Salaam"          "Africa/Djibouti"               "A
##  [22] "Africa/El_Aaiun"               "Africa/Freetown"               "A
##  [25] "Africa/Harare"                 "Africa/Johannesburg"           "A
```

3.10 lubridate

Fusos horários

```
# que horas sao em...
lubridate::with_tz(agora, tzone = "GMT")
```

```
## [1] "2020-04-28 04:04:28 GMT"
```

```
lubridate::with_tz(agora, tzone = "Europe/Stockholm")
```

```
## [1] "2020-04-28 06:04:28 CEST"
```

```
# altera o fuso sem mudar a hora
lubridate::force_tz(agora, tzone = "GMT")
```

```
## [1] "2020-04-28 01:04:28 GMT"
```

3.10 lubridate

Operações com datas

Intervalos

Intervalos podem ser salvos em objetos com classe interval

```
# datas
inicio_r <- lubridate::dmy("30-11-2011")
hoje_r <- lubridate::today()

# intervalo
r_interval <- lubridate::interval(inicio_r, hoje_r)
r_interval
```

```
## [1] 2011-11-30 UTC--2020-04-28 UTC
```

```
class(r_interval)
```

```
## [1] "Interval"
## attr(,"package")
```

3.10 lubridate

Operações com datas

Intervalos

Intersecção de intervalor com a função `int_overlaps()`

```
# outra forma de definir um intervalo: o operador %--%
r_interval <- lubridate::dmy("30-11-2011") %--% lubridate::today()
namoro_interval <- lubridate::dmy("25-06-2008") %--% lubridate::today()
```

```
# verificar sobreposicao
lubridate::int_overlaps(r_interval, namoro_interval)
```

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

3.10 lubridate

Operações com datas

Aritmética com datas

```
# somando datas
inicio_r + lubridate::ddays(1)
```

```
## [1] "2011-12-01"
```

```
inicio_r + lubridate::dyears(1)
```

```
## [1] "2012-11-29 06:00:00 UTC"
```

3.10 lubridate

Operações com datas

Aritmética com datas

```
# criando datas recorrentes
reunioes <- lubridate::today() + lubridate::weeks(0:10)
reunioes
```

```
##  [1] "2020-04-28" "2020-05-05" "2020-05-12" "2020-05-19" "2020-05-26" "2020-06
## [10] "2020-06-30" "2020-07-07"
```

3.10 lubridate

Operações com datas

Aritmética com datas

```
# duracao de um intervalo
r_interval <- inicio_r %--% lubridate::today()
r_interval
```

```
## [1] 2011-11-30 UTC--2020-04-28 UTC
```

```
# transformacoes
r_interval / lubridate::dyears(1)
```

```
## [1] 8.410678
```

```
r_interval / lubridate::ddays(1)
```

```
## [1] 3072
```

Operações com datas

Aritmética com datas

```
# total do periodo estudando r
lubridate::as.period(r_interval)
```

```
## [1] "8y 4m 29d 0H 0M 0S"
```

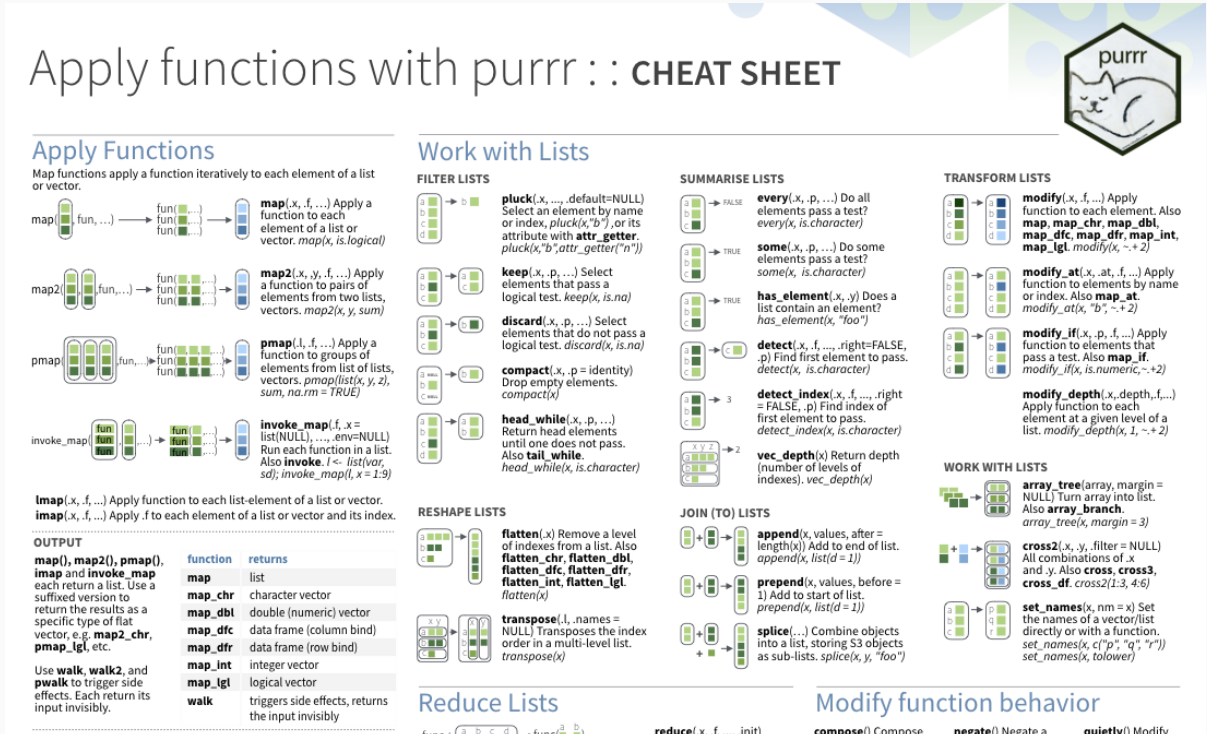
```
# tempo de namoro
lubridate::as.period(namoro_interval)
```

```
## [1] "11y 10m 3d 0H 0M 0S"
```


purrr
www.rstudio.com

3.11 purrr

Data Import Cheatsheet



Apply functions with purrr : : **CHEAT SHEET**

Apply Functions

Map functions apply a function iteratively to each element of a list or vector.

map(.x, .f, …) Apply a function to each element of a list or vector. *map(x, is.logical)*

map2(.x, .y, .f, …) Apply a function to pairs of elements from two lists, vectors. *map2(x, y, sum)*

pmap(.l, .f, …) Apply a function to groups of elements from list of lists, vectors. *pmap(list(x, y, z), sum, na.rm = TRUE)*

invoke_map(.f, .x = list(NULL), …, .env=NULL) Run each function in a list. Also **invoke**. *l <- list(var, sd); invoke_map(l, x = 1:9)*

lmap(.x, .f, ...) Apply function to each list-element of a list or vector.
imap(.x, .f, ...) Apply .f to each element of a list or vector and its index.

OUTPUT

map(), map2(), pmap(), imap and **invoke_map** each return a list. Use a suffixed version to return the results as a specific type of flat vector, e.g. **map2_chr**, **pmap_lgl**, etc.

Use **walk**, **walk2**, and **pwalk** to trigger side effects. Each return its input invisibly.

function	returns
map	list
map_chr	character vector
map_dbl	double (numeric) vector
map_dfc	data frame (column bind)
map_dfr	data frame (row bind)
map_int	integer vector
map_lgl	logical vector
walk	triggers side effects, returns the input invisibly

SHORTCUTS - within a purrr function:

"name" becomes **function(x) x[["name"]]**, e.g. *map(l, "a")* extracts *a* from each element of *l*

~ .x becomes **function(x) x**, e.g. *map(l, ~ 2 +.x)* becomes *map(l, function(x) 2 + x)*

~ .x .y becomes **function(.x, .y) .x .y**, e.g. *map2(l, p, ~ .x +.y)* becomes *map2(l, p, function(l, p) l + p)*

~ ..1 ..2 etc becomes **function(..1, ..2,** etc) **..1 ..2** etc, e.g. *pmap(list(a, b, c), ~ ..3 + ..1 - ..2)* becomes *pmap(list(a, b, c), function(a, b, c) c + a - b)*

Work with Lists

FILTER LISTS

pluck(.x, ..., .default=NULL) Select an element by name or index, *pluck(x,"b")* , or its attribute with **attr_getter**. *pluck(x,"b",attr_getter("n"))*

keep(.x, .p, …) Select elements that pass a logical test. *keep(x, is.na)*

discard(.x, .p, …) Select elements that do not pass a logical test. *discard(x, is.na)*

compact(.x, .p = identity) Drop empty elements. *compact(x)*

head_while(.x, .p, …) Return head elements until one does not pass. Also **tail_while**. *head_while(x, is.character)*

RESHAPE LISTS

flatten(.x) Remove a level of indexes from a list. Also **flatten_chr**, **flatten_dbl**, **flatten_dfc**, **flatten_dfr**, **flatten_int**, **flatten_lgl**. *flatten(x)*

transpose(.l, .names = NULL) Transposes the index order in a multi-level list. *transpose(x)*

SUMMARISE LISTS

every(.x, .p, …) Do all elements pass a test? *every(x, is.character)*

some(.x, .p, …) Do some elements pass a test? *some(x, is.character)*

has_element(.x, .y) Does a list contain an element? *has_element(x, "foo")*

detect(.x, .f, ..., .right=FALSE, .p) Find first element to pass. *detect(x, is.character)*

detect_index(.x, .f, ..., .right = FALSE, .p) Find index of first element to pass. *detect_index(x, is.character)*

vec_depth(x) Return depth (number of levels of indexes). *vec_depth(x)*

JOIN (TO) LISTS

append(x, values, after = length(x)) Add to end of list. *append(x, list(d = 1))*

prepend(x, values, before = 1) Add to start of list. *prepend(x, list(d = 1))*

splice(…) Combine objects into a list, storing S3 objects as sub-lists. *splice(x, y, "foo")*

TRANSFORM LISTS

modify(.x, .f, ...) Apply function to each element. Also **map**, **map_chr**, **map_dbl**, **map_dfc**, **map_dfr**, **map_int**, **map_lgl**. *modify(x, ~.+2)*

modify_at(.x, .at, .f, ...) Apply function to elements by name or index. Also **map_at**. *modify_at(x, "b", ~.+2)*

modify_if(.x, .p, .f, ...) Apply function to elements that pass a test. Also **map_if**. *modify_if(x, is.numeric, ~.+2)*

modify_depth(.x, depth, .f,...) Apply function to each element at a given level of a list. *modify_depth(x, 1, ~.+2)*

WORK WITH LISTS

array_tree(array, margin = NULL) Turn array into list. Also **array_branch**. *array_tree(x, margin = 3)*

cross2(.x, .y, .filter = NULL) All combinations of .x and .y. Also **cross**, **cross3**, **cross_df**. *cross2(1:3, 4:6)*

set_names(x, nm = x) Set the names of a vector/list directly or with a function. *set_names(x, c("p", "q", "r"))* *set_names(x, tolower)*

Reduce Lists

reduce(.x, .f, ..., .init) Apply function recursively to each element of a list or vector. Also **reduce_right**, **reduce2**, **reduce2_right**. *reduce(x, sum)*

accumulate(.x, .f, ..., .init) Reduce, but also return intermediate results. Also **accumulate_right**. *accumulate(x, sum)*

Modify function behavior

compose() Compose multiple functions.

lift() Change the type of input a function takes. Also **lift_dl**, **lift_dv**, **lift_ld**, **lift_lv**, **lift_vd**, **lift_vl**.

rerun() Rerun expression n times.

negate() Negate a predicate function (a pipe friendly !)

partial() Create a version of a function that has some args preset to values.

safely() Modify func to return list of results and errors.

quietly() Modify function to return list of results, output, messages, warnings.

possibly() Modify function to return default value whenever an error occurs (instead of error).

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **purrr.tidyverse.org** • purrr 0.2.3 • Updated: 2017-09

[*] https://github.com/rstudio/cheatsheets/raw/master/purrr.pdf

3.11 purrr

Pacote que implementa **Programação Funcional**

Uma função chama outra função para ser aplicada repetidamente percorrendo elementos de um objeto (vetor, lista ou data frame)

Exemplos:

- Análise de experimento em vários locais
- Análise de todas as respostas de um experimento
- Análise dos dados com diferentes transformações de variáveis
- Simulação computacional com diferentes delineamentos
- Importação de todos os datasets de um diretório

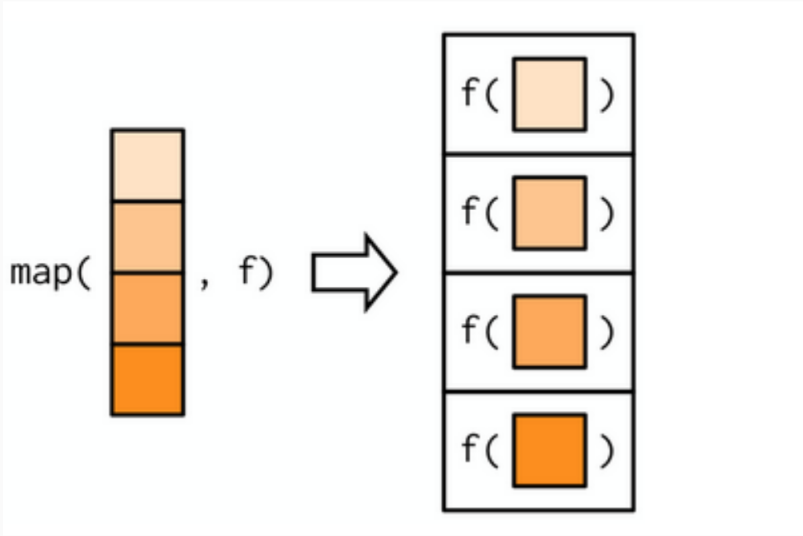
3.11 purrr

A principal função é a `map*()` e sua "família"

map(.x, .f)

.x: uma vetor, lista ou data frame

.f: uma função



3.11 purrr

Aplicando uma função em série

```
li <- list(1:5, c(4, 5, 7), c(98, 34,-10), c(2, 2, 2, 2, 2))
li
```

```
purrr::map(x, sum)
```

```
## [[1]]
## [1] 15
##
## [[2]]
## [1] 16
##
## [[3]]
## [1] 122
##
## [[4]]
## [1] 10
```

3.11 purrr

Tipos de retorno

map	returns
`map()`	list
`map_chr()`	character vector
`map_dbl()`	double vector (numeric)
`map_int()`	integer vector
`map_lgl()`	logical vector
`map_dfc()`	data frame (by column)
`map_dfr()`	data frame (by row)

3.11 purrr

Retorna double vector (numeric)

```
purrr::map_dbl(x, sum)
```

```
## [1]  15  16 122  10
```

Retorna strings

```
purrr::map_chr(x, paste, collapse = " ")
```

```
## [1] "1 2 3 4 5" "4 5 7"     "98 34 -10" "2 2 2 2 2"
```

3.11 purrr

Duas listas em paralelo

```
x <- list(3, 5, 0, 1)
y <- list(3, 5, 0, 1)

purrr::map2_dbl(x, y, prod)
```

```
## [1]  9 25  0  1
```

Várias listas aninhadas

```
x <- list(3, 5, 0, 1)
y <- list(3, 5, 0, 1)
z <- list(3, 5, 0, 1)

purrr::pmap_dbl(list(x, y, z), prod)
```

```
## [1]  27 125   0   1
```

3.11 purrr

Calcular a média para várias colunas

```
mean_var <- si %>%
  dplyr::select(species_number, altitude) %>%
  purrr::map_dbl(mean)
mean_var
```

```
## species_number       altitude
##       15.16509      545.64144
```

Calcular o desvio padrão para várias colunas

```
sd_var <- si %>%
  dplyr::select(species_number, altitude) %>%
  purrr::map_dbl(sd)
sd_var
```

```
## species_number       altitude
##       11.24582      392.40221
```

3.11 purrr

Material

Webinar de purrr avançado - Curso-R

purrr tutorial - Jennifer (Jenny) Bryan

Maurício Vancine

Contatos:

mauricio.vancine@gmail.com
mauriciovancine
mauriciovancine.netlify.com

Slides criados via pacote xaringan e tema Metropolis