Manipulação de dados com o dplyr

Prof. Walmes Zeviani
walmes@ufpr.br

Laboratório de Estatística e Geoinformação
Departamento de Estatística
Universidade Federal do Paraná

Um overview do dplyr

Motivação

- Depois dos dados arrumados, é a hora começar conhecê-los!
- Começa a fase de **análise exploratória de dados** (AED).
- Os dados são explorados para:
  - Conhecer as (propriedades das) variáveis.
  - Determinar medidas descritivas.
  - Comparar grupos.
  - Quantificar relações entre variáveis.
  - Extrair padrões.
  - Detectar ameaças e corrigir problemas.
- AED envolve inúmeras operações.
- É preciso conhecê-las e ser criativo para aplicar da melhor forma.

Detalhes do dplyr

- O dplyr é a **gramática** para manipulação de dados.
- Tem um conjunto **consistente** de verbos para atuar sobre tabelas.
  - 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.
  - E muito mais no cartão de referência: https://github.com/rstudio/cheatsheets/raw/master/data-transformation.pdf.
- Documentação:
  - https://dplyr.tidyverse.org/.
  - https://r4ds.had.co.nz/relational-data.html.
  - https://cran.r-project.org/package=dplyr

A ficha técnica

dplyr: A Grammar of Data Manipulation

A fast, consistent tool for working with data frame like objects, both in memory and out of memory.

Version:	0.8.0.1
Depends:	R (≥ 3.1.2)
Imports:	assertthat (≥ 0.2.0), glue (≥ 1.1.1), magrittr (≥ 1.5), methods, pkgconfig (≥ 2.0.1), R6 (≥ 2.2.2), Rcpp (≥ 1.0.0), rlang (≥ 0.3.0), tibble (≥ 2.0.0), tidyselect (≥ 0.2.5), utils
LinkingTo:	BH (≥ 1.58.0-1), plogr (≥ 0.1.10), Rcpp (≥ 1.0.0)
Suggests:	bit64 (≥ 0.9.7), callr (≥ 3.1.1), covr (≥ 3.0.1), DBI (≥ 0.7.14), dbplyr (≥ 1.2.0), dtplyr (≥ 0.0.2), ggplot2 (≥ 2.2.1), hms (≥ 0.4.1), knitr (≥ 1.19), Lahman (≥ 3.0-1), lubridate (≥ 1.7.4), MASS, mgcv (≥ 1.8.23), microbenchmark (≥ 1.4.4), nycflights13 (≥ 0.2.2), rmarkdown (≥ 1.8), RMySQL (≥ 0.10.13), RPostgreSQL (≥ 0.6.2), RSQLite (≥ 2.0), testthat (≥ 2.0.0), withr (≥ 2.1.1), broom (≥ 0.5.1), purrr (≥ 0.3.0), readr (≥ 1.3.1), crayon (≥ 1.3.4)
Published:	2019-02-15
Author:	Hadley Wickham [aut, cre], Romain François [aut], Lionel Henry [aut], Kirill Müller [aut], RStudio [cph, fnd]
Maintainer:	Hadley Wickham <hadley at rstudio.com>
BugReports:	https://github.com/tidyverse/dplyr/issues
License:	MIT + file LICENSE
URL:	http://dplyr.tidyverse.org, https://github.com/tidyverse/dplyr
NeedsCompilation:	yes
Materials:	README NEWS
In views:	Databases, ModelDeployment
CRAN checks:	dplyr results

Figura 1. Ficha técnica do `dplyr`.

Data Transformation with dplyr : : CHEAT SHEET

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

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)

Figura 2. Cartão de referência de operações em dados com tabulares com `dplyr`.

Vector Functions

TO USE WITH MUTATE ()

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

vectorized function

OFFSETS

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

CUMULATIVE AGGREGATES

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

RANKINGS

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

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

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

summary function

COUNTS

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

LOCATION

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

LOGICALS

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

POSITION/ORDER

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

RANK

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

SPREAD

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

Row Names

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

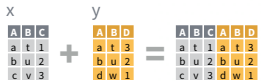
rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

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

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

Combine Tables

COMBINE VARIABLES



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

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

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

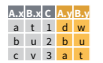
left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join matching values from y to x.

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

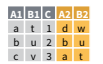
inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join data. Retain only rows with matches.

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

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

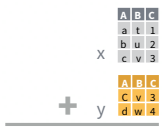


Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))



Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES



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

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

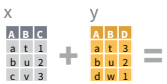
intersect(x, y, …)
Rows that appear in both x and y.

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

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

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS



Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, …)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

Figura 3. Cartão de referência de operações em dados com tabulares com `dplyr`.

Ordenação

Ordenação ascendente

```
library(tidyverse)

tb <- tibble(x = 1:4,
             y = c(4, 7, 1, 3),
             z = c(10, 10, 22, 22),
             k = c(TRUE, FALSE, FALSE, TRUE),
             u = c("A", "B", "A", "B"))

tb %>%
    arrange(z, y)
```

```
## # A tibble: 4 x 5
##       x     y     z k     u
##   <int> <dbl> <dbl> <lgl> <chr>
## 1     1     4    10 TRUE  A
## 2     2     7    10 FALSE B
## 3     3     1    22 FALSE A
## 4     4     3    22 TRUE  B
```

Ordenação descendente

```
tb %>%
    arrange(desc(z), -y)
```

```
## # A tibble: 4 x 5
##       x     y     z k     u
##   <int> <dbl> <dbl> <lgl> <chr>
## 1     4     3    22 TRUE  B
## 2     3     1    22 FALSE A
## 3     2     7    10 FALSE B
## 4     1     4    10 TRUE  A
```

Seleções

Filtro de linhas

```
tb %>%
    filter(x > 2 | u == "A")
```

```
## # A tibble: 3 x 5
##       x     y     z k     u
##   <int> <dbl> <dbl> <lgl> <chr>
## 1     1     4    10 TRUE  A
## 2     3     1    22 FALSE A
## 3     4     3    22 TRUE  B
```

Fatiar linhas

```
tb %>% slice(1:3)
```

1

```
## # A tibble: 3 x 5
##       x     y     z k     u
##   <int> <dbl> <dbl> <lgl> <chr>
## 1     1     4    10 TRUE  A
## 2     2     7    10 FALSE B
## 3     3     1    22 FALSE A
```

```
tb %>% slice(-(3:5))
```

1

```
## # A tibble: 2 x 5
##       x     y     z k     u
##   <int> <dbl> <dbl> <lgl> <chr>
## 1     1     4    10 TRUE  A
## 2     2     7    10 FALSE B
```

Sortear linhas

```
tb %>% sample_n(size = 3)
```

1

```
## # A tibble: 3 x 5
##       x     y     z k     u
##   <int> <dbl> <dbl> <lgl> <chr>
## 1     1     4    10 TRUE  A
## 2     4     3    22 TRUE  B
## 3     3     1    22 FALSE A
```

```
tb %>% sample_frac(size = 0.5)
```

1

```
## # A tibble: 2 x 5
##       x     y     z k     u
##   <int> <dbl> <dbl> <lgl> <chr>
## 1     4     3    22 TRUE  B
## 2     3     1    22 FALSE A
```

Seleção de variáveis por listagem

```
tb %>% select(x, y) # Inclusão.
```

1

```
## # A tibble: 4 x 2
##       x     y
##   <int> <dbl>
## 1     1     4
## 2     2     7
## 3     3     1
## 4     4     3
```

```
tb %>% select(-z)   # Exclusão.
```

1

```
## # A tibble: 4 x 4
##       x     y k     u
##   <int> <dbl> <lgl> <chr>
## 1     1     4 TRUE  A
## 2     2     7 FALSE B
## 3     3     1 FALSE A
## 4     4     3 TRUE  B
```

```
tb %>% select(y:k)  # Intervalo.
```

1

```
## # A tibble: 4 x 3
##       y     z k
##   <dbl> <dbl> <lgl>
## 1     4    10 TRUE
## 2     7    10 FALSE
```

Seleção por posição

```
tb %>% select(3, 2, 1)
```

1

```
## # A tibble: 4 x 3
##       z     y     x
##   <dbl> <dbl> <int>
## 1    10     4     1
## 2    10     7     2
## 3    22     1     3
## 4    22     3     4
```

```
tb %>% select(-(1:2))
```

1

```
## # A tibble: 4 x 3
##       z k     u
##   <dbl> <lgl> <chr>
## 1    10 TRUE  A
## 2    10 FALSE B
## 3    22 FALSE A
## 4    22 TRUE  B
```

Seleção de variáveis por condição

```
tb %>% select_if(is.numeric)
```

```
## # A tibble: 4 x 3
##       x     y     z
##   <int> <dbl> <dbl>
## 1     1     4    10
## 2     2     7    10
## 3     3     1    22
## 4     4     3    22
```

```
tb %>% select_if(negate(is.numeric))
```

```
## # A tibble: 4 x 2
##   k     u
##   <lgl> <chr>
## 1 TRUE  A
## 2 FALSE B
## 3 FALSE A
## 4 TRUE  B
```

Transformações

Modificar uma variável

```
tb %>%
    mutate(x = x * 2,
           u = as_factor(u))
```

```
## # A tibble: 4 x 5
##       x     y     z k     u
##   <dbl> <dbl> <dbl> <lgl> <fct>
## 1     2     4    10 TRUE  A
## 2     4     7    10 FALSE B
## 3     6     1    22 FALSE A
## 4     8     3    22 TRUE  B
```

Criar variáveis uma variável

```
tb %>%
    mutate(v = y * z^(x/4))
```

```
## # A tibble: 4 x 6
##       x     y     z k     u         v
##   <int> <dbl> <dbl> <lgl> <chr> <dbl>
## 1     1     4    10 TRUE  A      7.11
## 2     2     7    10 FALSE B     22.1
## 3     3     1    22 FALSE A     10.2
## 4     4     3    22 TRUE  B     66
```

Estatísticas resumo

Medidas resumo gerais

```
tb %>%
    summarise(x_mean = mean(x),
              y_median = median(y))
```

```
## # A tibble: 1 x 2
##   x_mean y_median
##    <dbl>    <dbl>
## 1    2.5      3.5
```

```
tb %>%
    summarise_if(.predicate = is.numeric, .funs = mean)
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1   2.5  3.75    16
```

```
# tb %>% summarise_at(.vars = c("x", "y"), .funs = mean)
# tb %>% summarise_at(.vars = 1:3, .funs = mean)
tb %>%
    summarise_at(.vars = vars(x:z), .funs = mean)
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1   2.5  3.75    16
```

Medidas resumo vetoriais

```
# Funções que retornam vetores.
# range(tb$x)
# fivenum(tb$x)

tb %>%
    summarise(x = list(range(x)),
              y = list(range(y))) %>%
    unnest()
```

```
## # A tibble: 2 x 2
##       x     y
##   <int> <dbl>
## 1     1     1
## 2     4     7
```

```
tb %>%
    do(data_frame(x = range(.$x),
                  y = range(.$y)))
```

```
## # A tibble: 2 x 2
##       x     y
##   <int> <dbl>
## 1     1     1
## 2     4     7
```

```
tb %>%
    summarise_if(is.numeric, function(s) list(range(s))) %>%
```

Descrição global

https://dabblingwithdata.wordpress.com/2018/01/02/my-favourite-r-package-for-summarising-data/

```
Hmisc::describe(iris)                1
psych::describe(iris)                2
skimr::skim(iris)                    3
```

```
brotools::describe(iris)             1
summarytools::descr(iris)            2
summarytools::dfSummary(iris)        3
```

Agregações

Agrupando por variáveis estratificadoras

```
tb %>%
    count(u)
```

```
## # A tibble: 2 x 2
##   u         n
##   <chr> <int>
## 1 A         2
## 2 B         2
```

```
tb %>%
    group_by(u) %>%
    summarise(x_mean = mean(x),
              y_range = max(y) - min(y),
              z_desv = sd(z))
```

```
## # A tibble: 2 x 4
##   u     x_mean y_range z_desv
##   <chr>  <dbl>   <dbl>  <dbl>
## 1 A          2       3   8.49
## 2 B          3       4   8.49
```

```
tb %>%
    group_by(u) %>%
    summarise_if(is.numeric, mean)
```

```
## # A tibble: 2 x 4
##   u         x     y     z
```

Agrupando usando funções resumo vetoriais

```
tb %>%
    group_by(u) %>%
    summarise_if(is.numeric, funs(min, median, max))
```

```
## # A tibble: 2 x 10
##   u     x_min y_min z_min x_median y_median z_median x_max y_max z_max
##   <chr> <dbl> <dbl> <dbl>    <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl>
## 1 A         1     1    10        2      2.5       16     3     4    22
## 2 B         2     3    10        3      5         16     4     7    22
```

```
tb %>%
    group_by(u) %>%
    do(data_frame(stat = c("min", "q1", "median", "q3", "max"),
                  x = fivenum(.$x),
                  y = fivenum(.$y)))
```

```
## # A tibble: 10 x 4
## # Groups:   u [2]
##    u     stat       x     y
##    <chr> <chr>  <dbl> <dbl>
##  1 A     min        1   1
##  2 A     q1         1   1
##  3 A     median     2   2.5
##  4 A     q3         3   4
##  5 A     max        3   4
##  6 B     min        2   3
##  7 B     q1         2   3
##  8 B     median     3   5
```

Junções

Interseção

```
tb1 <- tibble(aluno = c("Marcos", "Pedro", "Cintia", "Larissa", "Carla"),
              nota = c(10, 3, 6, 9, 10),
              faltas = c(3, 5, 6, 3, 0))
tb2 <- tibble(paciente = c("Pedro", "Larissa", "Marcos", "Lucas"),
              consulta = c("2017-02-23", "2019-01-30", "2018-12-10", "2018-11-
              exame = c("sangue", "urina", NA, NA))

inner_join(tb1, tb2, by = c("aluno" = "paciente"))
```

```
## # A tibble: 3 x 5
##   aluno    nota faltas consulta   exame
##   <chr>   <dbl>  <dbl> <chr>      <chr>
## 1 Marcos     10      3 2018-12-10 <NA>
## 2 Pedro       3      5 2017-02-23 sangue
## 3 Larissa     9      3 2019-01-30 urina
```

União

```
full_join(tb1, tb2, by = c("aluno" = "paciente"))
```

1

```
## # A tibble: 6 x 5
##   aluno    nota faltas consulta   exame
##   <chr>   <dbl>  <dbl> <chr>      <chr>
## 1 Marcos     10      3 2018-12-10 <NA>
## 2 Pedro       3      5 2017-02-23 sangue
## 3 Cintia      6      6 <NA>       <NA>
## 4 Larissa     9      3 2019-01-30 urina
## 5 Carla      10      0 <NA>       <NA>
## 6 Lucas      NA     NA 2018-11-20 <NA>
```

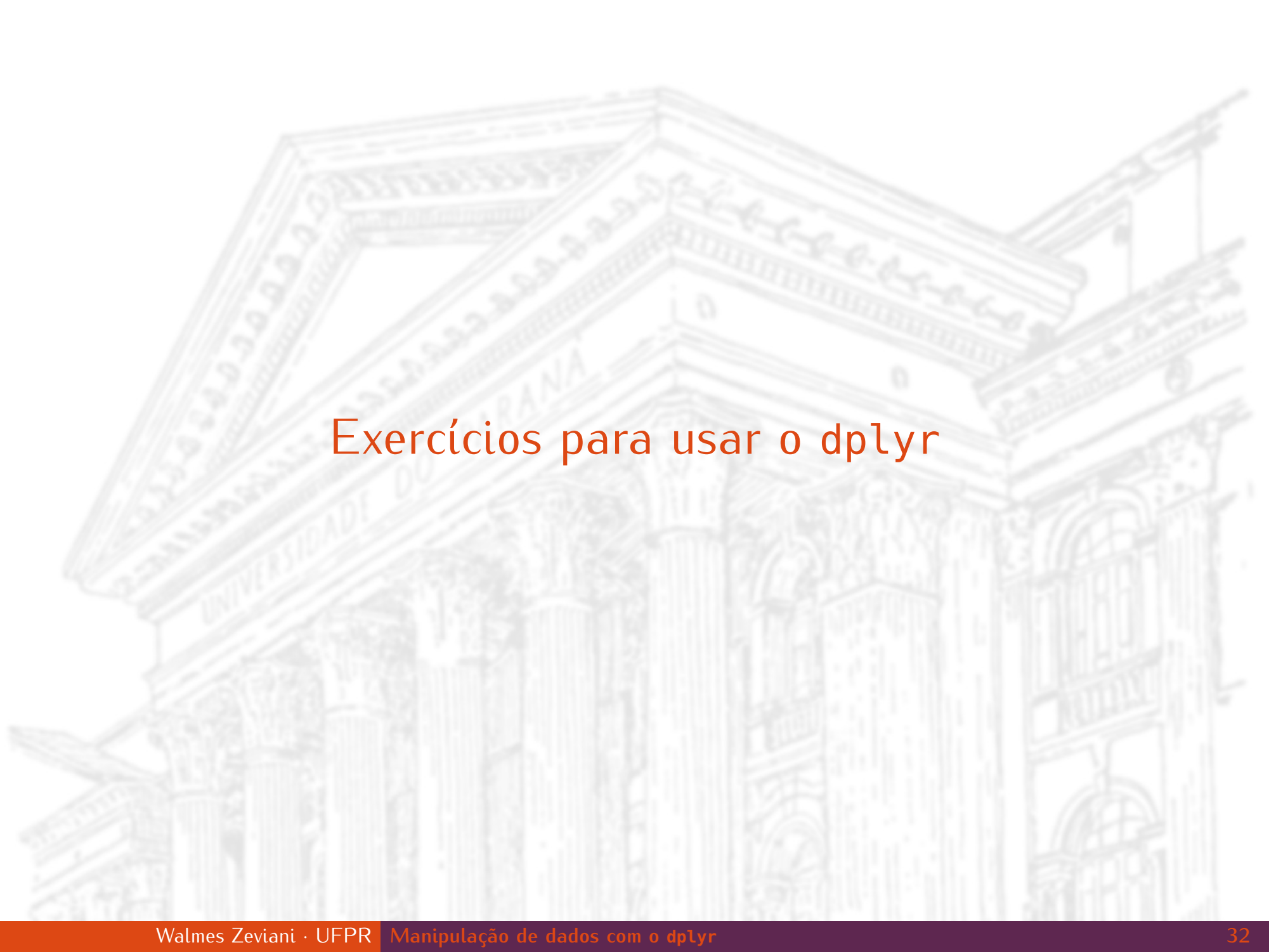
Junções parciais

```
right_join(tb1, tb2, by = c("aluno" = "paciente"))
```

```
## # A tibble: 4 x 5
##   aluno    nota faltas consulta   exame
##   <chr>   <dbl>  <dbl> <chr>      <chr>
## 1 Pedro       3      5 2017-02-23 sangue
## 2 Larissa     9      3 2019-01-30 urina
## 3 Marcos     10      3 2018-12-10 <NA>
## 4 Lucas      NA     NA 2018-11-20 <NA>
```

```
left_join(tb1, tb2, by = c("aluno" = "paciente"))
```

```
## # A tibble: 5 x 5
##   aluno    nota faltas consulta   exame
##   <chr>   <dbl>  <dbl> <chr>      <chr>
## 1 Marcos     10      3 2018-12-10 <NA>
## 2 Pedro       3      5 2017-02-23 sangue
## 3 Cintia      6      6 <NA>       <NA>
## 4 Larissa     9      3 2019-01-30 urina
## 5 Carla      10      0 <NA>       <NA>
```



Exercícios para usar o dplyr

Ninfas em soja

1. Ler os dados em http://leg.ufpr.br/~walmes/data/ninfas.txt.
2. Ordenação.
   - 2.1 Ordenar pelo valor do terço `superior`.
   - 2.2 Ordenar pelo valor do terço `medio` de forma descrescente.
   - 2.3 Ordenar pelas datas > variedade > bloco.
3. Filtros.
   - 3.1 Filtrar só para a variedade `BRS 245 RR`.
   - 3.2 Filtrar só para a variedade `BRS 245 RR` e `EMBRAPA 48`.
   - 3.3 Filtrar só para variedades diferentes de `EMBRAPA 48`.
   - 3.4 Filtrar quando `superior` com mais de 30 e inferior com mais de 20.
   - 3.5 Filtrar para `medio` entre 20 e 50.
   - 3.6 Filtrar para avaliações entre 2009-12-24 e 2010-01-11.
   - 3.7 Filtrar para a soma dos terços maior que 100.

4. Fatias.
    - 4.1 As linhas 34, 74, 23 e 41.
    - 4.2 As 10 primeiras linhas.
    - 4.3 Da linha 50 até a 63.
    - 4.4 As últimas 10 linhas.
    - 4.5 Apenas a linhas com posição múltipla de 3.
    - 4.6 Remover as 100 primeiras linhas.
5. Amostragem.
    - 5.1 Uma amostra de 30 linhas.
    - 5.2 Uma amostra de 30 linhas com reposição.
    - 5.3 Uma amostra de 10% das linhas.
    - 5.4 Pegar 20 registros com os maior número de ninfas no `superior`.

6. Seleção de variáveis.
    - 6.1 Selecionar apenas os terços.
    - 6.2 Remover a variável bloco.
    - 6.3 Mudar a ordem das colunas finais para `inferior`, `medio` e `superior`.
    - 6.4 Manter as variáveis com nome terminado em `rior`.
7. Modificação/criação de variáveis.
    - 7.1 Criar a variável total somando os terços.
    - 7.2 Criar a diferença entre o terço superior e inferior.
    - 7.3 Converter bloco e variedade para fator.
    - 7.4 Criar a raiz quadrada do número de ninfas em cada terço.

8. Renomear.
    - 8.1 Renomear variedade para tratamento.
    - 8.2 Renomear os terços para versões abreviadas com 3 digitos.
    - 8.3 Passar todas as variáveis para caixa alta.
    - 8.4 Abreviar todas as variáveis para nomes com 3 digitos.
9. Medidas descritivas gerais.
    - 9.1 Total de ninfas no terço superior.
    - 9.2 Total de ninfas em cada um dos terços.
    - 9.3 Média e desvio-padrão de ninfas em cada terço.
    - 9.4 Correlação do número de ninfas entre os terços (duas a duas).

10. Medidas descritivas por extrato.

10.1 Total de registros por variedade.
10.2 Total de registros por data.
10.3 Total de registros por variedade e data.
10.4 Total de ninfas no terço superior por data.
10.5 Total de ninfas nos 3 terços juntos por data.
10.6 Total de ninfas nos 3 terços juntos por variedade, ordene no final.
10.7 Total de ninfas nos 3 terços juntos por data e variedade. Guardar em objeto para usar a seguir.
10.8 A variedade com mais ninfas em cada data.
10.9 A data com mais ninfas em cada variedade.