Leitura de dados com readr

Prof. Walmes Zeviani

walmes@ufpr.br

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

Um overview do readr

Motivação

- O processo de análise de dados começa com a importação dos dados para o ambiente de manipulação.
- Existem várias meios para armazenar dados.
  - Arquivos de texto pleno (tsv, txt, csv, etc).
  - Planilhas eletrônicas.
  - Bancos de dados relacionais.
  - Etc.
- O readr tem recursos para importação de dados retangulares na forma de texto pleno.
- Documentação:
  - https://readr.tidyverse.org/.
  - https://r4ds.had.co.nz/data-import.html.
  - https://cran.r-project.org/package=readr

A ficha técnica

readr: Read Rectangular Text Data

The goal of 'readr' is to provide a fast and friendly way to read rectangular data (like 'csv', 'tsv', and 'fwf'). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes.

Version:	1.3.1
Depends:	R (≥ 3.1)
Imports:	Rcpp (≥ 0.12.0.5), tibble, hms (≥ 0.4.1), R6, clipr, crayon, methods
LinkingTo:	Rcpp, BH
Suggests:	curl, testthat, knitr, rmarkdown, stringi, covr, spelling
Published:	2018-12-21
Author:	Hadley Wickham [aut], Jim Hester [aut, cre], Romain Francois [aut], R Core Team [ctb] (Date time code adapted from R), RStudio [cph, fnd], Jukka Jylänki [ctb, cph] (grisu3 implementation), Mikkel Jørgensen [ctb, cph] (grisu3 implementation)
Maintainer:	Jim Hester <james.hester at rstudio.com>
BugReports:	https://github.com/tidyverse/readr/issues
License:	GPL-2 | GPL-3 | file LICENSE [expanded from: GPL (≥ 2) | file LICENSE]
URL:	http://readr.tidyverse.org, https://github.com/tidyverse/readr
NeedsCompilation:	yes
SystemRequirements:	GNU make
Language:	en-US
Materials:	README NEWS
CRAN checks:	readr results

Figura 1. Ficha técnica do readr.

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**)**

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()**)**

a,b,c
1,2,3
4,5,NA

A	B	C
1	2	3
4	5	NA

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")

a;b;c
1;2;3
4;5;NA

A	B	C
1	2	3
4	5	NA

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

a|b|c
1|2|3
4|5|NA

A	B	C
1	2	3
4	5	NA

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

a b c
1 2 3
4 5 NA

A	B	C
1	2	3
4	5	NA

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

a,b,c
1,2,3
4,5,NA

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

1	2	3
4	5	NA

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

A	B	C
1	2	3
4	5	NA

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

A	B	C
1	2	3

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

x	y	z
A	B	C
1	2	3
4	5	NA

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

A	B	C
NA	2	3
4	5	NA

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)

Figura 2. Cartão de referência de importação de dados com o readr.

Funções para importação

```
library(tidyverse)

ls("package:readr") %>% str_subset("^read_")
```

```
##  [1] "read_csv"           "read_csv2"          "read_csv2_chunked"
##  [4] "read_csv_chunked"   "read_delim"         "read_delim_chunked"
##  [7] "read_file"          "read_file_raw"      "read_fwf"
## [10] "read_lines"         "read_lines_chunked" "read_lines_raw"
## [13] "read_log"           "read_rds"           "read_table"
## [16] "read_table2"        "read_tsv"           "read_tsv_chunked"
```

Protótipo de argumentos da função

```
# Argumentos da `read_csv()`.
args(read_csv)
```

```
## function (file, col_names = TRUE, col_types = NULL, locale = default_locale
##     na = c("", "NA"), quoted_na = TRUE, quote = "\"", comment = "",
##     trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
##         n_max), progress = show_progress())
## NULL
```

```
# Argumentos da `locale()`.
args(locale)
```

```
## function (date_names = "en", date_format = "%AD", time_format = "%AT",
##     decimal_mark = ".", grouping_mark = ",", tz = "UTC", encoding = "UTF-8"
##     asciify = FALSE)
## NULL
```

As funções de escrita

```
ls("package:readr") %>% str_subset("^write_")
```

```
## [1] "write_csv"       "write_delim"     "write_excel_csv"
## [4] "write_file"      "write_lines"     "write_rds"
## [7] "write_tsv"
```

```
args(write_delim)
```

```
## function (x, path, delim = " ", na = "NA", append = FALSE, col_names = !app
## NULL
```

As funções de parsing

- Uma das maiores vantagens do `readr` é a maior flexibilidade para determinar o tipo de valor dos campos.
- As funções para o *parsing* (exame) são usadas para atribuir o tipo de valor apropriado durante a importação.
- Isso é economia de tempo.

```
ls("package:readr") %>% str_subset("^parse_")
```

1

```
##  [1] "parse_character" "parse_date"      "parse_datetime"
##  [4] "parse_double"    "parse_factor"    "parse_guess"
##  [7] "parse_integer"   "parse_logical"   "parse_number"
## [10] "parse_time"      "parse_vector"
```

As funções de parsing

```
# Conversão para data e data-tempo.
parse_date("2018/12/25")
parse_datetime("20181225")
parse_datetime("2018-12-25T12:10:00")

# Número com separador de milhar.
guess_parser("1,234,566")
parse_guess("1,234,566")

# Datas.
guess_parser(c("2010-10-10"))
parse_guess(c("2010-10-10"))
```

Importação de dados com readr

- O argumento obrigatório é o caminho para o arquivo.
- Argumentos opcionais existem pra um controle detalhado das opções de importação.

```
url <- "http://leg.ufpr.br/~walmes/data/anovareg.txt"

# Default.
tb <- read_tsv(file = url)

# Tipo de valores para cada variável.
tb <- read_tsv(file = url, col_types = "cicd")

# Renomeia os campos.
tb <- read_tsv(file = url, col_names = c("clt", "ntr", "blc", "index"))
```

Importação de dados com readr

- A lista de especificações de tipo de valor garante que variáveis com certos nomes serão importadas com o tipo de valor definido.
- Ou seja, em todo arquivo que houver `bloco`, ele será importado como fator.
- Não precisa estar na ordem e nem conter todas as variáveis.

```
# Lista de especificações.
specs <- cols(
    cultivar = col_factor(levels = NULL),
    bloco = col_factor(levels = NULL)
    # dose = col_integer(),
    # indice = col_integer()
)

# Usando a lista.
tb <- read_tsv(file = url, col_types = specs)
str(tb)
```

Importação de dados com readr

- Funções da readr produzem tibbles.
- Não importam string como fator, apenas por especificação.

```
attr(tb, "spec") <- NULL
str(tb)
```

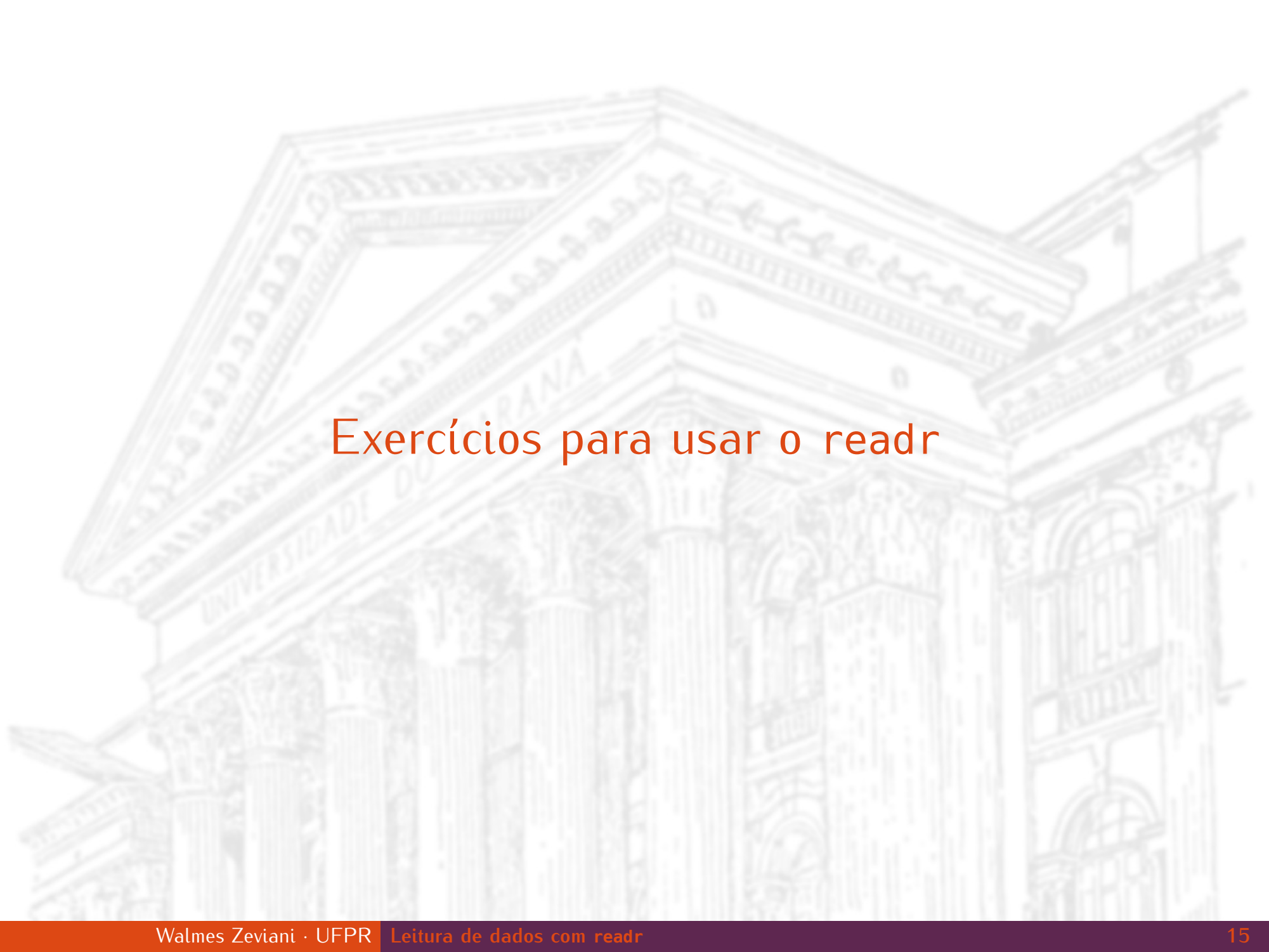
```
## Classes 'tbl_df', 'tbl' and 'data.frame':    72 obs. of  4 variables:
##  $ cultivar: Factor w/ 3 levels "Ag-1002","BR-300",..: 1 1 1 1 1 1 1 1 1 1
##  $ dose    : int  0 0 0 0 60 60 60 60 120 120 ...
##  $ bloco   : Factor w/ 4 levels "I","II","III",..: 1 2 3 4 1 2 3 4 1 2 ...
##  $ indice  : int  46 48 44 46 48 47 49 48 52 50 ...
```

Exportação de dados com readr

```
ls("package:readr") %>% str_subset("^write_")
```

```
## [1] "write_csv"       "write_delim"     "write_excel_csv"
## [4] "write_file"      "write_lines"     "write_rds"
## [7] "write_tsv"
```

```
write_csv(iris, path = "iris_dataset.csv")
```



Exercícios para usar o readr

No diretório de arquivos web http://leg.ufpr.br/~walmes/data/, importar os dados dos arquivos:

1. `irmpa.csv`.
2. `frango_comportamento.txt`.
3. `soja.txt`.
4. `compingest.txt`.
5. `carros_venda_webmotors_270314.txt`.
6. `metereologia.txt`.
7. `aval_carros_fwf.txt`. São 16 campos. Exceto o primeiro, 15 últimos campos são de 2 dígitos.