# Arrumação de dados com `tidyr`

Prof. Walmes Zeviani
walmes@ufpr.br

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

# Um overview do `tidyr`

# Motivação

- A maior parte das etapas de análise de dados assume que os dados estão arrumados:
  - Cada coluna é uma variável/atributo/campo.
  - Cada linha é uma observação/caso/instância/tupla.
  - Cada cédula é o registro de uma variável de uma observação.
- Situações que fogem a regra:
  - Disposição no formato longo ou amplo.
  - Colunas com valores concatenados.
  - Registros com valores ausentes.

# O `tidyr`

- O `tidyr` contém recursos para arrumação dos dados.
  - Mudança de disposição dos dados.
  - Substituição de missings.
  - Separação e união de campos.
- Documentação:
  - https://tidyr.tidyverse.org/.
  - https://r4ds.had.co.nz/tidy-data.html.
  - https://cran.r-project.org/package=tidyr

# A ficha técnica

**tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions**

An evolution of 'reshape2'. It's designed specifically for data tidying (not general reshaping or aggregating) and works well with 'dplyr' data pipelines.

| | |
|---|---|
| Version: | 0.8.3 |
| Depends: | R (≥ 3.1) |
| Imports: | dplyr (≥ 0.7.0), glue, magrittr, purrr, Rcpp, rlang, stringi, tibble, tidyselect (≥ 0.2.5), utils |
| LinkingTo: | Rcpp |
| Suggests: | covr, gapminder, knitr, rmarkdown, testthat |
| Published: | 2019-03-01 |
| Author: | Hadley Wickham [aut, cre], Lionel Henry [aut], RStudio [cph] |
| Maintainer: | Hadley Wickham <hadley at rstudio.com> |
| BugReports: | https://github.com/tidyverse/tidyr/issues |
| License: | MIT + file LICENSE |
| URL: | http://tidyr.tidyverse.org, https://github.com/tidyverse/tidyr |
| NeedsCompilation: | yes |
| Materials: | README NEWS |
| In views: | Databases |
| CRAN checks: | tidyr results |

Figura 1. Ficha técnica do `tidyr`.

# Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [ always returns a new tibble, [[ and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen

```
# A tibble: 234 × 6
   manufacturer      model displ
          <chr>      <chr> <dbl>
1          audi         a4   1.8
2          audi         a4   1.8
3          audi         a4   2.0
4          audi         a4   2.0
5          audi         a4   2.8
6          audi         a4   2.8
7          audi         a4   3.1
8          audi a4 quattro   1.8
9          audi a4 quattro   1.8
10         audi a4 quattro   2.0
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

tibble display

```
156 1999   6   auto(l4)
157 1999   6   auto(l4)
158 2008   8   auto(l4)
159 2008   8   auto(s4)
160 1999   4 manual(m5)
161 1999   4   auto(l4)
162 2008   4 manual(m5)
163 2008   4 manual(m5)
164 2008   4   auto(l4)
165 2008   4   auto(l4)
166 1999   4   auto(l4)
[ reached getOption("max.print")
-- omitted 68 rows ]
```

A large table to display

data frame display

- Control the default appearance with options:

  **options(**tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

**CONSTRUCT A TIBBLE IN TWO WAYS**

**tibble**(…)
Construct by columns.
*tibble(x = 1:3, y = c("a", "b", "c"))*

Both make this tibble

**tribble**(…)
Construct by rows.

```
tribble( ~x,   ~y,
         1,   "a",
         2,   "b",
         3,   "c")
```

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

**as_tibble**(x, …) Convert data frame to tibble.

**enframe**(x, name = "name", value = "value")
Convert named vector to a tibble

**is_tibble**(x) Test whether x is a tibble.

RStudio

# Tidy Data with tidyr

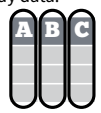
**Tidy data** is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

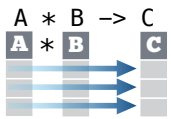

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Tidy data:




Makes variables easy to access as vectors




Preserves cases during vectorized operations

# Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

**gather(**data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

| country | 1999 | 2000 |
|---|---|---|
| A | 0.7K | 2K |
| B | 37K | 80K |
| C | 212K | 213K |

→

| country | year | cases |
|---|---|---|
| A | 1999 | 0.7K |
| B | 1999 | 37K |
| C | 1999 | 212K |
| A | 2000 | 2K |
| B | 2000 | 80K |
| C | 2000 | 213K |

key value

*gather(table4a, `1999`, `2000`, key = "year", value = "cases")*

**spread(**data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

| country | year | type | count |
|---|---|---|---|
| A | 1999 | cases | 0.7K |
| A | 1999 | pop | 19M |
| A | 2000 | cases | 2K |
| A | 2000 | pop | 20M |
| B | 1999 | cases | 37K |
| B | 1999 | pop | 172M |
| B | 2000 | cases | 80K |
| B | 2000 | pop | 174M |
| C | 1999 | cases | 212K |
| C | 1999 | pop | 1T |
| C | 2000 | cases | 213K |
| C | 2000 | pop | 1T |

→

| country | year | cases | pop |
|---|---|---|---|
| A | 1999 | 0.7K | 19M |
| A | 2000 | 2K | 20M |
| B | 1999 | 37K | 172M |
| B | 2000 | 80K | 174M |
| C | 1999 | 212K | 1T |
| C | 2000 | 213K | 1T |

key value

*spread(table2, type, count)*

# Handle Missing Values

**drop_na(**data, ...**)**
Drop rows containing NA's in … columns.

x

| x1 | x2 |
|---|---|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

→

| x1 | x2 |
|---|---|
| A | 1 |
| D | 3 |

*drop_na(x, x2)*

**fill(**data, ..., .direction = c("down", "up")**)**
Fill in NA's in … columns with most recent non-NA values.

x

| x1 | x2 |
|---|---|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

→

| x1 | x2 |
|---|---|
| A | 1 |
| B | 1 |
| C | 1 |
| D | 3 |
| E | 3 |

*fill(x, x2)*

**replace_na(**data, replace = list(), ...**)**
Replace NA's by column.

x

| x1 | x2 |
|---|---|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

→

| x1 | x2 |
|---|---|
| A | 1 |
| B | 2 |
| C | 2 |
| D | 3 |
| E | 2 |

*replace_na(x, list(x2 = 2))*

# Expand Tables - quickly create tables with combinations of values

**complete(**data, ..., fill = list()**)**
Adds to the data missing combinations of the values of the variables listed in …
*complete(mtcars, cyl, gear, carb)*

**expand(**data, ...**)**
Create new tibble with all possible combinations of the values of the variables listed in …
*expand(mtcars, cyl, gear, carb)*

# Split Cells

Use these functions to split or combine cells into individual, isolated values.

**separate(**data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...**)**

Separate each cell in a column to make several columns.

table3

| country | year | rate |
|---|---|---|
| A | 1999 | 0.7K/19M |
| A | 2000 | 2K/20M |
| B | 1999 | 37K/172M |
| B | 2000 | 80K/174M |
| C | 1999 | 212K/1T |
| C | 2000 | 213K/1T |

→

| country | year | cases | pop |
|---|---|---|---|
| A | 1999 | 0.7K | 19M |
| A | 2000 | 2K | 20M |
| B | 1999 | 37K | 172 |
| B | 2000 | 80K | 174 |
| C | 1999 | 212K | 1T |
| C | 2000 | 213K | 1T |

*separate(table3, rate, into = c("cases", "pop"))*

**separate_rows(**data, ..., sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows. Also **separate_rows_()**.

table3

| country | year | rate |
|---|---|---|
| A | 1999 | 0.7K/19M |
| A | 2000 | 2K/20M |
| B | 1999 | 37K/172M |
| B | 2000 | 80K/174M |
| C | 1999 | 212K/1T |
| C | 2000 | 213K/1T |

→

| country | year | rate |
|---|---|---|
| A | 1999 | 0.7K |
| A | 1999 | 19M |
| A | 2000 | 2K |
| A | 2000 | 20M |
| B | 1999 | 37K |
| B | 1999 | 172M |
| B | 2000 | 80K |
| B | 2000 | 174M |
| C | 1999 | 212K |
| C | 1999 | 1T |
| C | 2000 | 213K |
| C | 2000 | 1T |

*separate_rows(table3, rate)*

**unite(**data, col, ..., sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

table5

| country | century | year |
|---|---|---|
| Afghan | 19 | 99 |
| Afghan | 20 | 0 |
| Brazil | 19 | 99 |
| Brazil | 20 | 0 |
| China | 19 | 99 |
| China | 20 | 0 |

→

| country | year |
|---|---|
| Afghan | 1999 |
| Afghan | 2000 |
| Brazil | 1999 |
| Brazil | 2000 |
| China | 1999 |
| China | 2000 |

*unite(table5, century, year, col = "year", sep = "")*



Figura 2. Cartão de referência arrumação de dados com `tidyr`.

# Funções do pacote

```
library(tidyverse)

ls("package:tidyr")
```

```
##  [1] "%>%"           "complete"      "complete_"
##  [4] "crossing"      "crossing_"     "drop_na"
##  [7] "drop_na_"      "expand"        "expand_"
## [10] "extract"       "extract_"      "extract_numeric"
## [13] "fill"          "fill_"         "full_seq"
## [16] "gather"        "gather_"       "nest"
## [19] "nest_"         "nesting"       "nesting_"
## [22] "population"    "replace_na"    "separate"
## [25] "separate_"     "separate_rows" "separate_rows_"
## [28] "smiths"        "spread"        "spread_"
## [31] "table1"        "table2"        "table3"
## [34] "table4a"       "table4b"       "table5"
## [37] "uncount"       "unite"         "unite_"
## [40] "unnest"        "unnest_"       "who"
```

# Empilhar variáveis

- Situação comum quando:
  - são feitas medidas repetidas no tempo.
  - dados de painel e/ou questionário.

```
n <- 3
tb1 <- tibble("trat" = LETTERS[1:n],
              aval1 = rpois(n, 4),
              aval2 = rpois(n, 4),
              aval3 = rpois(n, 4))
tb1
```

```
## # A tibble: 3 x 4
##   trat  aval1 aval2 aval3
##   <chr> <int> <int> <int>
## 1 A         5     2     2
## 2 B         4     6     2
## 3 C         4     2     3
```

# Empilhar variáveis

```
tb2 <- tb1 %>%
    gather(key = "aval",
           value = "insetos",
           aval1:aval3)
tb2
```

```
## # A tibble: 9 x 3
##   trat  aval  insetos
##   <chr> <chr>   <int>
## 1 A     aval1       5
## 2 B     aval1       4
## 3 C     aval1       4
## 4 A     aval2       2
## 5 B     aval2       6
## 6 C     aval2       2
## 7 A     aval3       2
## 8 B     aval3       2
## 9 C     aval3       3
```

# Desempilhar variável

- É a operação inversa de empilhar.
- Dados nessa disposição são menos comuns.

```
tb2 %>%
    spread(key = "aval",
           value = "insetos")
```

```
## # A tibble: 3 x 4
##   trat  aval1 aval2 aval3
##   <chr> <int> <int> <int>
## 1 A         5     2     2
## 2 B         4     6     2
## 3 C         4     2     3
```

# Separar variável

- Muito comum quando um campo de texto é a união de várias informações.
- Ex: datas, horas, endereços, etc.

```
tb <- tibble(veiculo = c("Celta", "Gol", "Uno"),
             ano_mod = c("2011/2012", "2012/2012", "2015/2016"),
             local = c("Curitiba-PR", "Santos-SP", "Viçosa-MG"))

tb %>%
    separate(col = "ano_mod",
             into = c("ano", "modelo"),
             sep = "/",
             convert = TRUE) %>%
    separate(col = "local",
             into = c("cidade", "estado"),
             sep = "-")
```

```
## # A tibble: 3 x 5
##   veiculo   ano modelo cidade   estado
##   <chr>   <int>  <int> <chr>    <chr>
## 1 Celta    2011   2012 Curitiba PR
## 2 Gol      2012   2012 Santos   SP
## 3 Uno      2015   2016 Viçosa   MG
```

# Unir variáveis

- Quando vários campos precisam ser combinados para gerar uma informação.
- Ex: datas, horas, endereços, nomes.

```
tb <- tibble(dia = c(1, 5, 23, 16),
             mes = c(3, 6, 2, 9),
             ano = 2018)
tb %>%
    unite(col = "data", ano, mes, dia, sep = "-", remove = FALSE) %>%
    mutate(data = parse_date(data, format = "%Y-%m-%d"))
```

```
## # A tibble: 4 x 4
##   data         dia   mes   ano
##   <date>     <dbl> <dbl> <dbl>
## 1 2018-03-01     1     3  2018
## 2 2018-06-05     5     6  2018
## 3 2018-02-23    23     2  2018
## 4 2018-09-16    16     9  2018
```

# Manuseio de valores ausentes

```
tb <- tibble(jogador = 1:5,
             jogos = c(0, 1, 3, 1, 2),
             gols = c(NA, 0, 0, 2, 1),
             faltas = c(NA, 1, 1, 0, 0))

# tb %>%
#     drop_na()

tb %>%
    replace_na(list(gols = 0, faltas = 0))
```

```
## # A tibble: 5 x 4
##   jogador jogos  gols faltas
##     <int> <dbl> <dbl>  <dbl>
## 1       1     0     0      0
## 2       2     1     0      1
## 3       3     3     0      1
## 4       4     1     2      0
## 5       5     2     1      0
```

# Cédulas com objetos complexos

```
tb <- warpbreaks %>%
    as_tibble() %>%
    nest(breaks)
tb
```

```
## # A tibble: 6 x 3
##   wool  tension data
##   <fct> <fct>   <list>
## 1 A     L       <tibble [9 x 1]>
## 2 A     M       <tibble [9 x 1]>
## 3 A     H       <tibble [9 x 1]>
## 4 B     L       <tibble [9 x 1]>
## 5 B     M       <tibble [9 x 1]>
## 6 B     H       <tibble [9 x 1]>
```

```
tb <- iris %>%
    as_tibble() %>%
    nest(-Species)
tb
```

```
## # A tibble: 3 x 2
##   Species    data
##   <fct>      <list>
## 1 setosa     <tibble [50 x 4]>
## 2 versicolor <tibble [50 x 4]>
## 3 virginica  <tibble [50 x 4]>
```

# Expandir combinações

- Expandir combinações entre variáveis é necessário:
  - Para criar o desenho experimental de expermentos fatoriais.
  - Criar o grid para predição da resposta combinando todas as variáveis de entrada.

```
tb <- crossing(blc = as.character(as.roman(1:2)),
               trt = LETTERS[1:3])
tb$prod <- rexp(tb$trt)
tb
```

```
## # A tibble: 6 x 3
##   blc   trt     prod
##   <chr> <chr>  <dbl>
## 1 I     A     0.0632
## 2 I     B     0.519
## 3 I     C     0.290
## 4 II    A     0.286
## 5 II    B     2.24
## 6 II    C     1.02
```

# Expandir combinações

```
tb <- tb[c(-1, -5, -6), ]
complete(tb, blc, trt)
```

```
## # A tibble: 6 x 3
##   blc   trt     prod
##   <chr> <chr>  <dbl>
## 1 I     A     NA
## 2 I     B      0.519
## 3 I     C      0.290
## 4 II    A      0.286
## 5 II    B     NA
## 6 II    C     NA
```

```
expand(tb, blc, trt)
```

```
## # A tibble: 6 x 2
##   blc   trt
##   <chr> <chr>
## 1 I     A
## 2 I     B
## 3 I     C
## 4 II    A
## 5 II    B
## 6 II    C
```

# Exercícios para usar o `tidyr`

1. Ninfas em soja.
    1.1 Ler os dados em http://leg.ufpr.br/~walmes/data/ninfas.txt.
    1.2 Empilhar nos terços da planta.
    1.3 Desempilhar nas datas.
2. Óleos essenciais.
    2.1 Ler o dados em http://leg.ufpr.br/~walmes/data/oleos.txt.
    2.2 Criar uma variável indicadora do registro: `1:nrow(tb)`.
    2.3 Empilhar nas avaliações, i.e, os 5 campos começados com a.
    2.4 Desempilhar na variável `forma` de aplicação.
3. Futebol.
    3.1 Ler os dados em http://leg.ufpr.br/~walmes/data/euro_football_players.txt.
    3.2 Substituir os missings em `goal`, `red` e `yel` por 0.

4. Avaliação de veículos.
    - 4.1 Ler os dados em http://leg.ufpr.br/~walmes/data/aval_carros_nota.txt.
    - 4.2 Desempilhar na variável `item` os valores de `nota`.
5. Carros à venda.
    - 5.1 Ler os dados em http://leg.ufpr.br/~walmes/data/duster_venda_260314.txt.
    - 5.2 Separar os campos ano e modelo na variável ano. Ex: `2012/2013`.
    - 5.3 Substituir o `NA` em `km` percorrido pela média de km percorrido: `mean(..., na.rm = TRUE)`.
6. Condição domíliar dos municípios.
    - 6.1 Ler os dados em http://leg.ufpr.br/~walmes/data/ipea_habitacao.csv.
    - 6.2 Contatenar o nome do município com o nome do estado.