TIDYR & DPLYR

A dupla implacável

Nicole Luduvice

Março de 2021

Sobre mim

- [2017] Estudante de graduação de Estatística no IME-USP
- [2020] Estagiaria na Curso-R/R6 consultoria
- [2021] Terranova Jurimetria
- RLadies São Paulo <3

Contato:

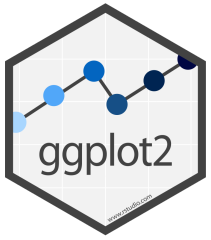
Linkedin: Nicole Luduvice

e-mail: nzluduvice@gmail.com

Tidyverse

- Coleção de pacotes para ciência de dados
- Estes pacotes tem filosofia, gramática e estruturas de dados em comum (criadas para um pacote trabalhar em conjunto com o outro além de que quando voce aprende a usar um pacote, aprender a usar os outros pacotes do tidyverse se torna muito mais tranquilo)
- Facilita MUITOOO o trabalho do cientista de dados :)

Principais Pacotes



ggplot2: criar gráficos

dplyr: manipular dados

tidyr: Arrumar (*tidy*) os dados

readr: Ler dados retangulares (csv,tsv e fwf)

purrr: fornecer kit completo de ferramentas para programação funcional

stringr: Lidar com strings

forcats: Lidar com fatores

tibble:

Tibble `tbl_df`

- Diversas funções do `dplyr` e `tidyr` recebem uma tibble e retornam uma tibble
- Tibbles são data frames com melhorias
- Principal diferença (data.frame vs. tibble): forma em que a data frame é apresentada no console

data.frame

```
##     Sepal.Length Sepal.Width Petal.Length
## 1            5.1         3.5          1.4
## 2            4.9         3.0          1.4
## 3            4.7         3.2          1.3
## 4            4.6         3.1          1.5
## 5            5.0         3.6          1.4
## 6            5.4         3.9          1.7
## 7            4.6         3.4          1.4
## 8            5.0         3.4          1.5
## 9            4.4         2.9          1.4
## 10           4.9         3.1          1.5
## 11           5.4         3.7          1.5
## 12           4.8         3.4          1.6
## 13           4.8         3.0          1.4
## 14           4.3         3.0          1.1
## 15           5.8         4.0          1.2
## 16           5.7         4.4          1.5
## 17           5.4         3.9          1.3
## 18           5.1         3.5          1.4
## 19           5.7         3.8          1.7
## 20           5.1         3.8          1.5
## 21           5.4         3.4          1.7
## 22           5.1         3.7          1.5
## 23           4.6         3.6          1.0
## 24           5.1         3.3          1.7
## 25           4.8         3.4          1.9
## 26           5.0         3.0          1.6
## 27           5.0         3.4          1.6
## 28           5.2         3.5          1.5
## 29           5.2         3.4          1.4
## 30           4.7         3.2          1.6
## 31           4.8         3.1          1.6
## 32           5.4         3.4          1.5
## 33           5.2         4.1          1.5
```

tibble

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length
##           <dbl>       <dbl>        <dbl>
##  1          5.1         3.5          1.4
##  2          4.9         3            1.4
##  3          4.7         3.2          1.3
##  4          4.6         3.1          1.5
##  5          5           3.6          1.4
##  6          5.4         3.9          1.7
##  7          4.6         3.4          1.4
##  8          5           3.4          1.5
##  9          4.4         2.9          1.4
## 10          4.9         3.1          1.5
## # … with 140 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

Pipe %>%

- Atalho: Control + Shift + M
- As funções do tidyverse foram projetadas para serem usadas com o operador %>%
- Avalia o código do lado esquerdo e passa o resultado como o primeiro argumento do código do lado direito

- As linhas abaixo são equivalentes

```
x <- c(1.234, 2.4536, 3.442, 3.24, 1.2223)

sum(round(mean(x),2),5)
```

```
## [1] 7.32
```

```
x %>% mean() %>% round(2) %>% sum(5)
```

```
## [1] 7.32
```

Dados

Iris

```
iris <- tibble::as_tibble(iris) %>% tibble::rowid_to_column()
iris
```

```
## # A tibble: 150 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 140 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

DPLYR

```
install.packages("dplyr")
library(dplyr)
```

dplyr()

- O pacote auxilia na manipulação de dados, realizando-a de forma eficiente
- Com essas funções temos uma flexibilidade incrível para transformar nossos dados!
- `funcao(base_dados, variaveis)`

select()

- Seleciona colunas

```
iris %>% select(Sepal.Length, Sepal.Width)

iris %>% select(1,2)
```

```
iris %>% select(Sepal.Length:Sepal.Width)
```

```
## # A tibble: 150 x 2
##    Sepal.Length Sepal.Width
##           <dbl>       <dbl>
##  1          5.1         3.5
##  2          4.9         3  
##  3          4.7         3.2
##  4          4.6         3.1
##  5          5           3.6
##  6          5.4         3.9
##  7          4.6         3.4
##  8          5           3.4
##  9          4.4         2.9
## 10          4.9         3.1
## # … with 140 more rows
```

Exisem funções no `dplyr` que ajudam na seleção das variáveis:

- `starts_with()`: colunas que começam com um prefixo
- `ends_with()`: colunas que terminam com um sufixo
- `contains()`: colunas que contêm uma string
- `last_col()`: última coluna

```
iris %>% select(starts_with("Petal"))
```

```
## # A tibble: 150 x 2
##    Petal.Length Petal.Width
##           <dbl>       <dbl>
##  1          1.4         0.2
##  2          1.4         0.2
##  3          1.3         0.2
##  4          1.5         0.2
##  5          1.4         0.2
##  6          1.7         0.4
##  7          1.4         0.3
##  8          1.5         0.2
##  9          1.4         0.2
## 10          1.5         0.1
## # … with 140 more rows
```

Esta função nos permite retirar colunas da base da seguinte forma

```
iris %>% select(-Species)
```

```
## # A tibble: 150 x 5
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 140 more rows, and 1 more variable:
## #   Petal.Width <dbl>
```

relocate()

- Muda as posições das colunas

Podemos colocar uma coluna antes de outra qualquer

```
iris %>% relocate(Species, .before = Petal.Length)
```

```
## # A tibble: 150 x 6
##    rowid Sepal.Length Sepal.Width Species
##    <int>        <dbl>       <dbl> <fct>
##  1     1          5.1         3.5 setosa
##  2     2          4.9         3   setosa
##  3     3          4.7         3.2 setosa
##  4     4          4.6         3.1 setosa
##  5     5          5           3.6 setosa
##  6     6          5.4         3.9 setosa
##  7     7          4.6         3.4 setosa
##  8     8          5           3.4 setosa
##  9     9          4.4         2.9 setosa
## 10    10          4.9         3.1 setosa
## # … with 140 more rows, and 2 more variables:
## #   Petal.Length <dbl>, Petal.Width <dbl>
```

Ou colocar uma coluna depois de outra qualquer

```
iris %>% relocate(Species, .after = Petal.Length)
```

```
## # A tibble: 150 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 140 more rows, and 2 more variables:
## #   Species <fct>, Petal.Width <dbl>
```

Obs: Se nenhum dos argumentos (`.before` ou `.after`) for fornecido, as colunas especificadas serão colocadas no começo da base

arrange()

- Orderna linhas
  - Se a variável for categórica ordena por ordem alfabética
  - Se a variável for numérica ordena do menor para o maior
  - Se a varíavel for um fator, ordena pelos níveis do fator

```
iris %>% arrange(Sepal.Length)
```

```
## # A tibble: 150 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1    14          4.3         3            1.1
##  2     9          4.4         2.9          1.4
##  3    39          4.4         3            1.3
##  4    43          4.4         3.2          1.3
##  5    42          4.5         2.3          1.3
##  6     4          4.6         3.1          1.5
##  7     7          4.6         3.4          1.4
##  8    23          4.6         3.6          1
##  9    48          4.6         3.2          1.4
## 10     3          4.7         3.2          1.3
## # … with 140 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

Podemos ordenar as linhas de forma decrescente usando a função `desc()`

```
iris %>% arrange(desc(Sepal.Length))
```

```
## # A tibble: 150 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1   132          7.9         3.8          6.4
##  2   118          7.7         3.8          6.7
##  3   119          7.7         2.6          6.9
##  4   123          7.7         2.8          6.7
##  5   136          7.7         3            6.1
##  6   106          7.6         3            6.6
##  7   131          7.4         2.8          6.1
##  8   108          7.3         2.9          6.3
##  9   110          7.2         3.6          6.1
## 10   126          7.2         3.2          6
## # … with 140 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

É também possível ordenar mais de uma coluna de uma vez

```
iris %>% arrange(Species, Sepal.Length)
```

```
## # A tibble: 150 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1    14          4.3         3            1.1
##  2     9          4.4         2.9          1.4
##  3    39          4.4         3            1.3
##  4    43          4.4         3.2          1.3
##  5    42          4.5         2.3          1.3
##  6     4          4.6         3.1          1.5
##  7     7          4.6         3.4          1.4
##  8    23          4.6         3.6          1
##  9    48          4.6         3.2          1.4
## 10     3          4.7         3.2          1.3
## # … with 140 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

filter()

- Filtra linhas

```
iris %>% filter(Petal.Length <= 2)
```

```
## # A tibble: 50 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 40 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

Podemos filtrar mais de duas colunas

```
iris %>% filter(Species == "setosa", Sepal.Length <= 5)
```

```
## # A tibble: 28 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     2          4.9         3            1.4
##  2     3          4.7         3.2          1.3
##  3     4          4.6         3.1          1.5
##  4     5          5           3.6          1.4
##  5     7          4.6         3.4          1.4
##  6     8          5           3.4          1.5
##  7     9          4.4         2.9          1.4
##  8    10          4.9         3.1          1.5
##  9    12          4.8         3.4          1.6
## 10    13          4.8         3            1.4
## # … with 18 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

Usualmente queremos selecionar mais de uma categoria dentro do filter

```
iris %>% filter(Species %in% c("setosa", "versicolor"))
```

```
## # A tibble: 100 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 90 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

Qualquer tipo de operação pode ser feita dentro do filter, contanto que esta operação retorne TRUE/FALSE

```
c(0,1,2,3,4) > 2
```

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

Principais testes lógicos

- **<** menor que
- **>** maior que
- **<=** menor igual a
- **>** maior igual a
- **==** igual a
- **%in%** faz parte de um grupo
- **is.na()** é NA

- **&** e
- **|** ou
- **!** diferente de

mutate()

- Modifica e cria colunas

Modificando...

```
iris %>%
  mutate(
    Sepal.Length = Sepal.Length*10, Sepal.Width = Sepal.Width*10,
    Petal.Length = Petal.Length*10, Petal.Width = Petal.Width*10
  )
```

```
## # A tibble: 150 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1           51          35           14
##  2     2           49          30           14
##  3     3           47          32           13
##  4     4           46          31           15
##  5     5           50          36           14
##  6     6           54          39           17
##  7     7           46          34           14
##  8     8           50          34           15
##  9     9           44          29           14
## 10    10           49          31           15
## # … with 140 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

Criando...

```
iris %>%
  mutate(
    Sepal.Length_mm = Sepal.Length*10, Sepal.Width_mm = Sepal.Width*10,
    Petal.Length_mm = Petal.Length*10, Petal.Width_mm = Petal.Width*10
  )
```

```
## # A tibble: 150 x 10
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 140 more rows, and 6 more variables:
## #   Petal.Width <dbl>, Species <fct>,
## #   Sepal.Length_mm <dbl>, Sepal.Width_mm <dbl>,
## #   Petal.Length_mm <dbl>, Petal.Width_mm <dbl>
```

Podemos fazer qualquer operação contanto que o seu resultado retorne um vetor o mesmo comprimento das linhas da base ou de comprimento 1

```
iris %>%
  mutate(unidade_medida = "cm") %>%
  relocate(unidade_medida)
```

```
## # A tibble: 150 x 7
##    unidade_medida rowid Sepal.Length Sepal.Width
##    <chr>          <int>        <dbl>       <dbl>
##  1 cm                 1          5.1         3.5
##  2 cm                 2          4.9         3
##  3 cm                 3          4.7         3.2
##  4 cm                 4          4.6         3.1
##  5 cm                 5          5           3.6
##  6 cm                 6          5.4         3.9
##  7 cm                 7          4.6         3.4
##  8 cm                 8          5           3.4
##  9 cm                 9          4.4         2.9
## 10 cm                10          4.9         3.1
## # … with 140 more rows, and 3 more variables:
## #   Petal.Length <dbl>, Petal.Width <dbl>,
## #   Species <fct>
```

Funções do dplyr úteis:

```
df <- tibble::tibble(
  var = c(1, 2, 3, 2, 4, 5, -1)
)
```

- if_else: retorna valor dependendo de um teste lógico

```
df %>%
  mutate(
    var2 = if_else(var >= 0, "yay", ":(")
  )
```

```
## # A tibble: 7 x 2
##     var var2
##   <dbl> <chr>
## 1     1 yay
## 2     2 yay
## 3     3 yay
## 4     2 yay
## 5     4 yay
## 6     5 yay
## 7    -1 :(
```

- case_when: generalização do `if_else()`

```
df %>%
  mutate(
    var2 = case_when(
      var < 0 ~ ":(",
      var < 4 ~ "yay",
      TRUE ~ "ok.."
    )
  )
```

```
## # A tibble: 7 x 2
##     var var2
##   <dbl> <chr>
## 1     1 yay
## 2     2 yay
## 3     3 yay
## 4     2 yay
## 5     4 ok..
## 6     5 ok..
## 7    -1 :(
```

Obs1: A ordem das condições é importante (a segunda condição verifica se `0 <= var < 4`)

Obs2: O `TRUE` significa que caso as observações não entrem nas condições anteriores a função retorna o valor do lado direito

- lag: devolve valor defasado
- lead: devolve o valor futuro

```
df %>%
  mutate(
    var2 = lag(var),
    var3 = lead(var)
  )
```

```
## # A tibble: 7 x 3
##     var  var2  var3
##   <dbl> <dbl> <dbl>
## 1     1    NA     2
## 2     2     1     3
## 3     3     2     2
## 4     2     3     4
## 5     4     2     5
## 6     5     4    -1
## 7    -1     5    NA
```

- n: retorna o tamanho do grupo
- n_distinct: retorna o número de valores distintos

```
df %>%
  mutate(
    var2 = n(),
    var3 = n_distinct(var)
  )
```

```
## # A tibble: 7 x 3
##     var  var2  var3
##   <dbl> <int> <int>
## 1     1     7     6
## 2     2     7     6
## 3     3     7     6
## 4     2     7     6
## 5     4     7     6
## 6     5     7     6
## 7    -1     7     6
```

- na_if: converte um valor específico em NA

```
df %>%
  mutate(
    var2 = na_if(var, -1)
  )
```

```
## # A tibble: 7 x 2
##     var  var2
##   <dbl> <dbl>
## 1     1     1
## 2     2     2
## 3     3     3
## 4     2     2
## 5     4     4
## 6     5     5
## 7    -1    NA
```

`transmute()`

- Modifica e cria colunas
- Seleciona apenas as colunas modificadas

Os códigos abaixo são equivalentes

```
iris %>% mutate(
    Sepal.Length_mm = Sepal.Length*10,
    Sepal.Width_mm = Sepal.Width*10,
    Petal.Length_mm = Petal.Length*10,
    Petal.Width_mm = Petal.Width*10
  ) %>%
  select(
    Sepal.Length_mm, Sepal.Width_mm, Petal.Length_mm, Petal.Width_mm
  )
```

```
iris %>% transmute(
    Sepal.Length_mm = Sepal.Length*10,
    Sepal.Width_mm = Sepal.Width*10,
    Petal.Length_mm = Petal.Length*10,
    Petal.Width_mm = Petal.Width*10
  )
```

```
## # A tibble: 150 x 4
##    Sepal.Length_mm Sepal.Width_mm Petal.Length_mm
##              <dbl>          <dbl>            <dbl>
##  1              51             35               14
##  2              49             30               14
##  3              47             32               13
##  4              46             31               15
##  5              50             36               14
##  6              54             39               17
##  7              46             34               14
##  8              50             34               15
##  9              44             29               14
## 10              49             31               15
## # … with 140 more rows, and 1 more variable:
## #   Petal.Width_mm <dbl>
```

summarise()

- Calcula medidas resumo como:
  - médias
  - soma
  - desvios padrões
  - frequências absolutas

```
iris %>%
  summarise(
    media_sepal_lenght = mean(Sepal.Length),
    sd_sepal_lenght = sd(Sepal.Length),
    media_sepal_width = mean(Sepal.Width),
    sd_sepal_width = sd(Sepal.Width),
    n = n()
  )
```

```
## # A tibble: 1 x 5
##   media_sepal_lenght sd_sepal_lenght
##                <dbl>           <dbl>
## 1               5.84           0.828
## # … with 3 more variables:
## #   media_sepal_width <dbl>,
## #   sd_sepal_width <dbl>, n <int>
```

group_by()

- O `group_by()` em si não faz nenhuma alteração notável na base

```
iris %>% group_by(Species)
```

```
## # A tibble: 150 x 6
## # Groups:   Species [3]
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 140 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

- Junto com outras funções como `mutate()` ou `summarise()`, o `group_by()` muda o escopo da função
  - trata cada grupo como se fossem bases distintas
  - executa o código separadamente para cada grupo

Obs: É importante lembrar de usar a função `ungroup()` para desagrupar os dados se quisermos continuar fazendo mudanças na base

group_by() + sumarise()

- Calcula medidas resumo agrupadas pelas categorias de uma outra coluna
- As medidas resumo serão aplicadas por grupo

```
iris %>%
  group_by(Species) %>%
  summarise(
    media_sepal_lenght = mean(Sepal.Length),
    sd_sepal_lenght = sd(Sepal.Length),
    media_sepal_width = mean(Sepal.Width),
    sd_sepal_width = sd(Sepal.Width),
    n = n()
  ) %>%
  ungroup()
```

```
## # A tibble: 3 x 6
##   Species    media_sepal_lenght sd_sepal_lenght
##   <fct>                   <dbl>           <dbl>
## 1 setosa                   5.01           0.352
## 2 versicolor               5.94           0.516
## 3 virginica                6.59           0.636
## # … with 3 more variables:
## #   media_sepal_width <dbl>,
## #   sd_sepal_width <dbl>, n <int>
```

across()

- Substitui a familia `*_all()`, `*_if()` e `*_at()`
- Facilita fazer operações em várias colunas
- `across(colunas, função, argumentos_adicionais, nomes_das_colunas)`

```
iris %>%
  summarise(
    across(
      .cols = c(Sepal.Width, Sepal.Length),
      .fns = sum,
      na.rm = TRUE,
      .names = "soma_{.col}"
    )
  )
```

```
## # A tibble: 1 x 2
##   soma_Sepal.Width soma_Sepal.Length
##              <dbl>             <dbl>
## 1             459.              876.
```

Obs: O `across()` pode ser utilizado em todos os verbos do `dplyr` menos `select()` e `rename()`

Se o argumento `.cols`não for fornecido, todas as colunas serão selecionadas

```
iris %>% summarise(
  across(.fns = n_distinct)
)
```

```
## # A tibble: 1 x 6
##   rowid Sepal.Length Sepal.Width Petal.Length
##   <int>        <int>       <int>        <int>
## 1   150           35          23           43
## # … with 2 more variables: Petal.Width <int>,
## #   Species <int>
```

Podemos colocar mais de uma função para ser aplicada nas colunas

```
iris %>%
  summarise(
    media_sepal_lenght = mean(Sepal.Length),
    sd_sepal_lenght = sd(Sepal.Length),
    media_sepal_width = mean(Sepal.Width),
    sd_sepal_width = sd(Sepal.Width),
    n = n()
  )
```

```
iris %>%
  summarise(
    across(
      starts_with("Sepal"),
      .fns = list(mean = mean, sd = sd)
    )
  )
```

```
## # A tibble: 1 x 4
##   Sepal.Length_mean Sepal.Length_sd
##               <dbl>           <dbl>
## 1              5.84           0.828
## # … with 2 more variables:
## #   Sepal.Width_mean <dbl>, Sepal.Width_sd <dbl>
```

Também é possível selecionar colunas a partir de testes lógico. Para isso usamos a função `where()`

```
iris %>%
  mutate(
    across(where(is.numeric) & starts_with("Sepal"), as.character)
  )
```

```
## # A tibble: 150 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int> <chr>        <chr>               <dbl>
##  1     1 5.1          3.5                   1.4
##  2     2 4.9          3                     1.4
##  3     3 4.7          3.2                   1.3
##  4     4 4.6          3.1                   1.5
##  5     5 5            3.6                   1.4
##  6     6 5.4          3.9                   1.7
##  7     7 4.6          3.4                   1.4
##  8     8 5            3.4                   1.5
##  9     9 4.4          2.9                   1.4
## 10    10 4.9          3.1                   1.5
## # … with 140 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <fct>
```

bind_rows()

- A função `bind_rows()` junta as linhas de duas ou mais bases

```
nova_df <- tibble::tibble( Sepal.Width = c(3.5, 3, 2.9), Sepal.Length = c(5, 4.3, 4.9), Petal.Length = c(1.2, 1.
  Petal.Width = c(0.3, 0.2, 0.1), Species = c("setosa", "versicolor", "virginica"))

iris %>% bind_rows(nova_df)
```

```
## # A tibble: 153 x 6
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 143 more rows, and 2 more variables:
## #   Petal.Width <dbl>, Species <chr>
```

Obs: A ordem das colunas das bases não precisam ser iguais

bind_cols()

- A função `bind_cols()` junta as colunas de duas ou mais bases

```
dados_pais
```

```
## # A tibble: 150 x 1
##    pais
##    <chr>
##  1 Maine, USA
##  2 Maine, USA
##  3 Maine, USA
##  4 Maine, USA
##  5 Maine, USA
##  6 Maine, USA
##  7 Maine, USA
##  8 Maine, USA
##  9 Maine, USA
## 10 Maine, USA
## # … with 140 more rows
```

```
iris %>%
  bind_cols(dados_pais) %>%
  relocate(pais)
```

```
## # A tibble: 150 x 7
##    pais       rowid Sepal.Length Sepal.Width
##    <chr>      <int>        <dbl>       <dbl>
##  1 Maine, USA     1          5.1         3.5
##  2 Maine, USA     2          4.9         3
##  3 Maine, USA     3          4.7         3.2
##  4 Maine, USA     4          4.6         3.1
##  5 Maine, USA     5          5           3.6
##  6 Maine, USA     6          5.4         3.9
##  7 Maine, USA     7          4.6         3.4
##  8 Maine, USA     8          5           3.4
##  9 Maine, USA     9          4.4         2.9
## 10 Maine, USA    10          4.9         3.1
## # … with 140 more rows, and 3 more variables:
## #   Petal.Length <dbl>, Petal.Width <dbl>,
## #   Species <fct>
```

Obs: Essas colunas devem sempre ter a mesma quantidade de linhas e (pra fazer sentido) elas devem ter as linhas de um data frame correspondendo com as linhas do outro data frame

Joins

- Junta duas bases a partir dos valores de uma ou mais colunas em comum
- Principais tipos de join:
  - Inner Join
  - Left Join
  - Right Join
  - Full Join
  - Anti Join

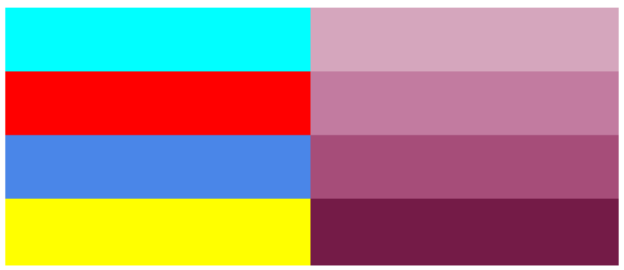
`inner_join()`

A

id	value1

B

id	value2



semi_join(A, B, by = "id")

id	value1

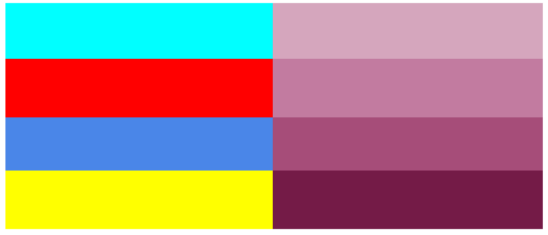
left_join()

A

id value1

B

id value2



left_join(A, B, by = "id")

id value1 value2

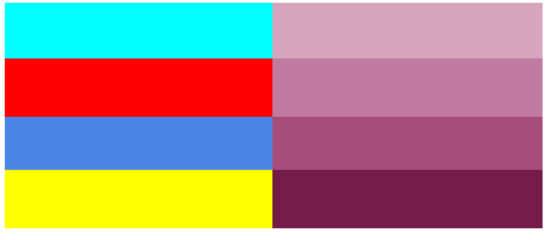
right_join()

A

id value1

B

id value2



right_join(A, B, by = "id")

id value2 value1

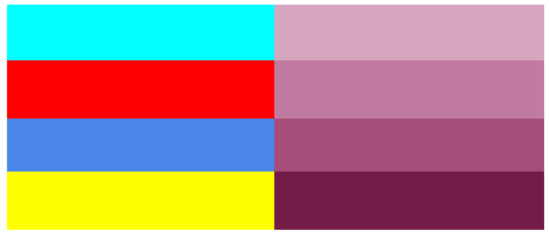
full_join()

A

id value1

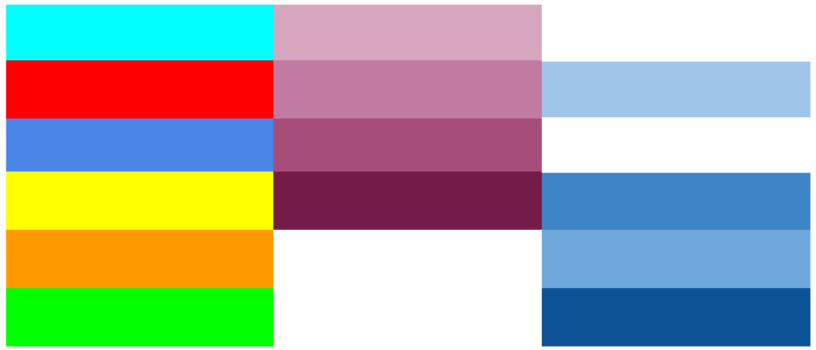
B

id value2



full_join(A, B, by = "id")

id value2 value1



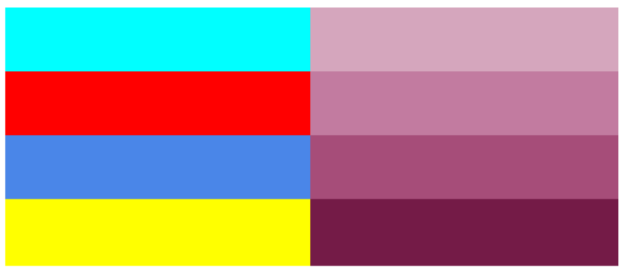
anti_join()

A

id value1

B

id value2



anti_join(A, B, by = "id")

id value1

```
dicio_iris
```

```
## # A tibble: 3 x 2
##   Species    pais
##   <chr>      <chr>
## 1 setosa     Maine, USA
## 2 versicolor Fortaleza, Brasil
## 3 virginica  Ottawa, Canada
```

```
iris_pais <- iris %>%
  left_join(dicio_iris, by = "Species")
iris_pais
```

```
## # A tibble: 150 x 7
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 140 more rows, and 3 more variables:
## #   Petal.Width <dbl>, Species <chr>, pais <chr>
```

Exemplo "prático" - pipe

```
iris %>%
  filter(Petal.Length < 5) %>%
  left_join(dicio_iris, by = "Species") %>%
  mutate(
    across(contains("l."), ~.x*10, .names = "{col}_mm"),
    pais =factor(pais, levels = c("Fortaleza, Brasil", "Ottawa, Canada", "Maine, USA"))
  ) %>%
  group_by(Species) %>%
  summarise(
    across(ends_with("_mm"), list(mean = mean, sd = sd), na.rm = TRUE),
    n = n(),
    diferentes_paises = n_distinct(pais)
  )
```

Exercicios

TIDYR

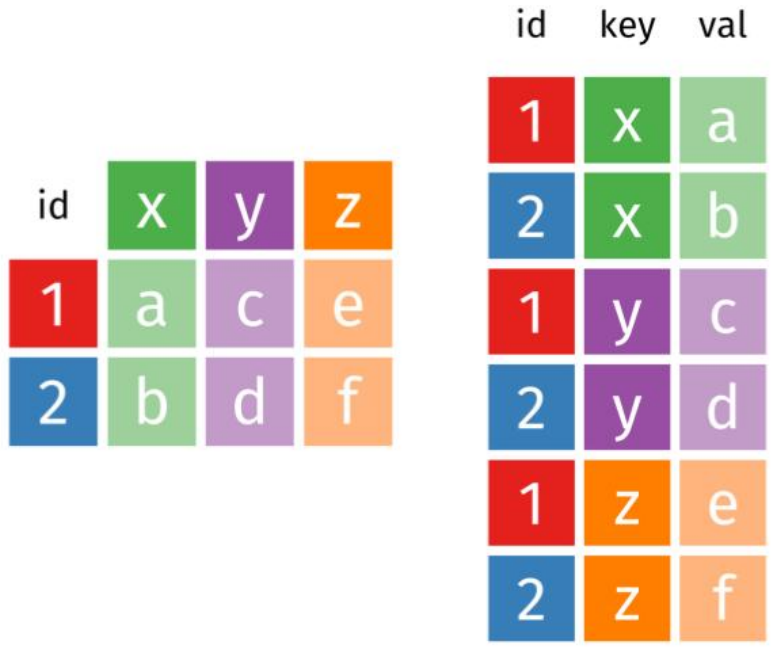
```
install.packages("tidyr")
library(tidyr)
```

`tidyr`

- Os pacotes do tidyverse foram criados para usar e retornar dados arrumados (*tidy*) sempre que apropriado.
- O pacote `tidyr`possui funções para deixar os dados no formato *tidy*
- Dados *tidy* estão organizados de forma que:
    - cada coluna é uma variável;
    - cada linha é uma observação;
    - cada célula representa um único valor.

`pivot_longer()`

- Em casos mais simples equivale ao `gather()`
- Alonga/empilha os dados



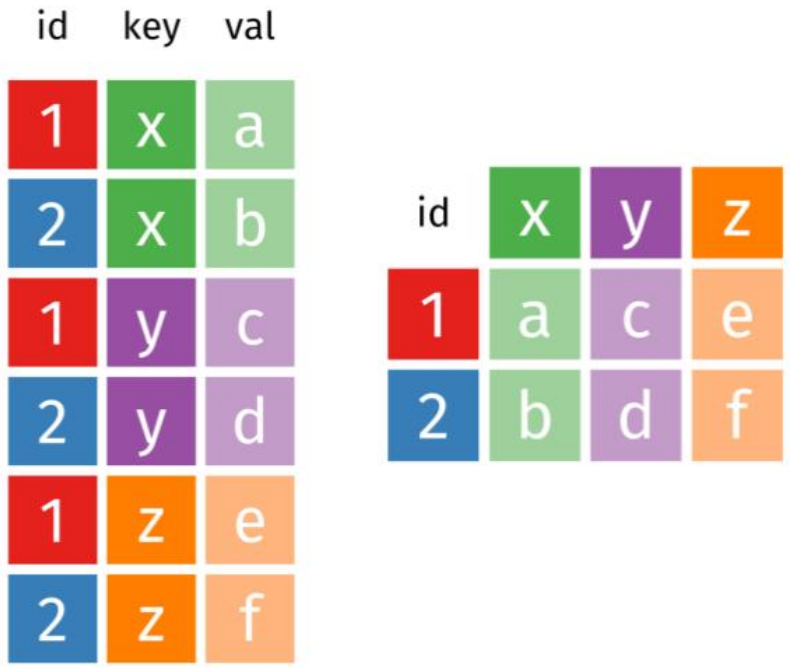
```
iris_longer <- iris_pais %>%
  pivot_longer(c(-Species, -pais, -rowid), names_to = "partes", values_to = "medidas")

iris_longer
```

```
## # A tibble: 600 x 5
##    rowid Species pais       partes       medidas
##    <int> <chr>   <chr>      <chr>          <dbl>
##  1     1 setosa  Maine, USA Sepal.Length     5.1
##  2     1 setosa  Maine, USA Sepal.Width      3.5
##  3     1 setosa  Maine, USA Petal.Length     1.4
##  4     1 setosa  Maine, USA Petal.Width      0.2
##  5     2 setosa  Maine, USA Sepal.Length     4.9
##  6     2 setosa  Maine, USA Sepal.Width      3
##  7     2 setosa  Maine, USA Petal.Length     1.4
##  8     2 setosa  Maine, USA Petal.Width      0.2
##  9     3 setosa  Maine, USA Sepal.Length     4.7
## 10     3 setosa  Maine, USA Sepal.Width      3.2
## # … with 590 more rows
```

`pivot_wider`

- Em casos mais simples equivale à antiga função `spread()`
- Alarga/espalha os dados



```
iris_longer %>%
  pivot_wider(id_cols = rowid, names_from = partes, values_from = medidas)
```

```
## # A tibble: 150 x 5
##    rowid Sepal.Length Sepal.Width Petal.Length
##    <int>        <dbl>       <dbl>        <dbl>
##  1     1          5.1         3.5          1.4
##  2     2          4.9         3            1.4
##  3     3          4.7         3.2          1.3
##  4     4          4.6         3.1          1.5
##  5     5          5           3.6          1.4
##  6     6          5.4         3.9          1.7
##  7     7          4.6         3.4          1.4
##  8     8          5           3.4          1.5
##  9     9          4.4         2.9          1.4
## 10    10          4.9         3.1          1.5
## # … with 140 more rows, and 1 more variable:
## #   Petal.Width <dbl>
```

separate()

- Separa uma coluna em varias usando um separador

```
iris_sep <- iris_pais %>%
  separate(pais, into = c("cidade", "pais"), sep = ", ") %>%
  relocate(cidade, pais, .after = rowid)

iris_sep
```

```
## # A tibble: 150 x 8
##    rowid cidade pais  Sepal.Length Sepal.Width
##    <int> <chr>  <chr>        <dbl>       <dbl>
##  1     1 Maine  USA            5.1         3.5
##  2     2 Maine  USA            4.9         3
##  3     3 Maine  USA            4.7         3.2
##  4     4 Maine  USA            4.6         3.1
##  5     5 Maine  USA            5           3.6
##  6     6 Maine  USA            5.4         3.9
##  7     7 Maine  USA            4.6         3.4
##  8     8 Maine  USA            5           3.4
##  9     9 Maine  USA            4.4         2.9
## 10    10 Maine  USA            4.9         3.1
## # … with 140 more rows, and 3 more variables:
## #   Petal.Length <dbl>, Petal.Width <dbl>,
## #   Species <chr>
```

unite

- Junta várias colunas em uma usando um separador

```
iris_sep %>%
  unite(col = "pais", cidade, pais, sep = ", ")
```

```
## # A tibble: 150 x 7
##    rowid pais       Sepal.Length Sepal.Width
##    <int> <chr>             <dbl>       <dbl>
##  1     1 Maine, USA          5.1         3.5
##  2     2 Maine, USA          4.9         3
##  3     3 Maine, USA          4.7         3.2
##  4     4 Maine, USA          4.6         3.1
##  5     5 Maine, USA          5           3.6
##  6     6 Maine, USA          5.4         3.9
##  7     7 Maine, USA          4.6         3.4
##  8     8 Maine, USA          5           3.4
##  9     9 Maine, USA          4.4         2.9
## 10    10 Maine, USA          4.9         3.1
## # … with 140 more rows, and 3 more variables:
## #   Petal.Length <dbl>, Petal.Width <dbl>,
## #   Species <chr>
```

nest()

- Cria uma coluna de lista

```
iris_nest <- iris %>% nest(-Species)
```

Ou podemos usar a função `group_by()`

```
iris %>% group_by(Species) %>% nest()
```

```
## # A tibble: 3 x 2
## # Groups:   Species [3]
##   Species    data
##   <fct>      <list>
## 1 setosa     <tibble [50 × 5]>
## 2 versicolor <tibble [50 × 5]>
## 3 virginica  <tibble [50 × 5]>
```

unnest()

- Desfaz a coluna de listas

```
iris_nest %>%
  unnest(data)
```

```
## # A tibble: 150 x 6
##    Species rowid Sepal.Length Sepal.Width
##    <fct>   <int>        <dbl>       <dbl>
##  1 setosa      1          5.1         3.5
##  2 setosa      2          4.9         3
##  3 setosa      3          4.7         3.2
##  4 setosa      4          4.6         3.1
##  5 setosa      5          5           3.6
##  6 setosa      6          5.4         3.9
##  7 setosa      7          4.6         3.4
##  8 setosa      8          5           3.4
##  9 setosa      9          4.4         2.9
## 10 setosa     10          4.9         3.1
## # … with 140 more rows, and 2 more variables:
## #   Petal.Length <dbl>, Petal.Width <dbl>
```

Exercicios

Referências

- Livro da Curso-R
- A tidyverse Cookbook
- R for data science
- Join com dplyr
- slide topp
- Fotos pivot

Obrigada!