R para Ciência de Dados 2

Manipulação de dados avançada

Iniciar gravação!

tidyr

Motivação

Um dos conceitos mais importantes do tidyverse é o de **dados tidy** ("dados arrumados"). Na prática, uma tabela tidy tem três propriedades importantes:

- Cada coluna é uma variável.
- Cada linha é uma observação.
- Cada célula é um único valor.

Essa definição garante uma maneira consistente de se referir a variáveis (nomes das colunas) e observações (índices das linhas). Além disso, o tidyverse foi construído pensando em tabelas tidy; na prática, uma base tidy fica mais fácil de manipular, visualizar, modelar, e por aí vai.

Mas esse conceito não parece óbvio? As nossas colunas não são *sempre* variáveis? As nossas linhas não são *sempre* observações? A realidade é bem mais complicada do que parece...

Bases bagunçadas

> *Tabelas arrumadas são todas parecidas, mas cada tabela bagunçada é bagunçada do seu próprio jeito.* — Hadley Wickham

Tabelas normalmente ficam bagunçadas por causa de processos manuais de imputação. O Excel é o culpado número 1.

O exemplo abaixo traz as notas de 3 cachorros em um treino de comportamento. Como o treino é repetido toda semana, basta criar mais uma coluna `NotaSemanaX` para registrar as notas de um novo treino.

```
#> # A tibble: 3 × 5
#>   Cachorro NotaSemana1 NotaSemana2 NotaSemana3 NotaSemana4
#>   <chr>          <dbl>       <dbl>       <dbl>       <dbl>
#> 1 Bacon             10           4           8           9
#> 2 Dexter            10           4          10           8
#> 3 Zip                0           0          10           9
```

Ilustração por @allison_horst

Arrumando a base

Não tem nada de errado com aquela tabela, mas perceba que cada coluna `NotaSemana` não é uma variável diferente! Na verdade, o próprio nome `NotaSemana` já indica que são duas variáveis: `Nota` e `Semana`.

```
#> # A tibble: 12 × 3
#>   Cachorro Semana  Nota
#>   <chr>     <dbl> <dbl>
#> 1 Bacon         1    10
#> 2 Bacon         2     4
#> 3 Bacon         3     8
#> 4 Bacon         4     9
#> 5 Dexter        1    10
#> # ℹ 7 more rows
```

Neste novo formato, cada linha é uma observação e cada coluna é uma variável.

Pivotagem

O conceito de pivotagem no tidyverse se refere a essa mudança da estrutura da base, geralmente para alcançar o formato tidy. Ela é similar à tabela dinâmica do Excel, mas um pouco mais poderosa.

O ato de pivotar envolve passar uma tabela da sua forma **larga** para a sua forma **longa** (ou fazer o caminho contrário). No exemplo dos cachorros, a base original era a versão larga e a base modificada era a versão longa.

Não existe um absoluto. Nunca dizemos que uma base é longa ou larga, nós só estamos tentando deixar ela *mais* longa ou *mais* larga. Também não é certo dizer que a base tidy é sempre melhor; apesar de o tidyverse geralmente funcionar melhor com tabelas tidy, existem ocasiões em que a tabela bagunçada é de fato o que queremos.

O pacote tidyr

O pacote que nos permite transformar uma base bagunçada em uma base tidy é o tidyr. Ele também nos ajuda a bagunçar um pouquinho a nossa base quando isso for necessário.

Apesar de o tidyr ser um pacote bem amplo, hoje vamos aprender sobre as duas funções mais importantes que ele traz: `pivot_longer()` e `pivot_wider()`. Para os exemplos, usaremos as mais tocadas da *Billboard* e os dados de renda do censo americano.

```
# Já carrega o tidyr e o dplyr
library(tidyverse)

# Tabelas que vamos usar
musicas <- select(billboard, track, wk1:wk6)
renda <- select(us_rent_income, NAME, variable, estimate)
```

Larga para longa

A tabela abaixo é muito parecida com a dos cachorros. Cada linha é uma música e cada coluna `wkX` é a posição da mesma no Top 100 durante aquela semana.

```
musicas
```

```
#> # A tibble: 317 × 7
#>   track                   wk1   wk2   wk3   wk4   wk5   wk6
#>   <chr>                 <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 Baby Don't Cry (Keep...    87    82    72    77    87    94
#> 2 The Hardest Part Of ...    91    87    92    NA    NA    NA
#> 3 Kryptonite                 81    70    68    67    66    57
#> 4 Loser                      76    76    72    69    67    65
#> 5 Wobble Wobble              57    34    25    17    17    31
#> # ℹ 312 more rows
```

Para deixar ela tidy, precisamos usar a função `pivot_longer()`.

pivot_longer()

```
musicas |>
  pivot_longer(
    cols = starts_with("wk"), # Colunas que começam com "wk"
    names_to = "semana",      # Nomes das colunas viram "semana"
    values_to = "posicao"     # Valores das colunas viram "posicao"
  )
```

```
#> # A tibble: 1,902 × 3
#>   track                   semana posicao
#>   <chr>                   <chr>    <dbl>
#> 1 Baby Don't Cry (Keep... wk1         87
#> 2 Baby Don't Cry (Keep... wk2         82
#> 3 Baby Don't Cry (Keep... wk3         72
#> 4 Baby Don't Cry (Keep... wk4         77
#> 5 Baby Don't Cry (Keep... wk5         87
#> # ℹ 1,897 more rows
```

Longa para larga

Na tabela abaixo temos o problema inverso da anterior: uma única coluna (`variable`) contém duas variáveis (`income` e `rent`).

```
renda
```

```
#> # A tibble: 104 × 3
#>   NAME    variable estimate
#>   <chr>   <chr>       <dbl>
#> 1 Alabama income      24476
#> 2 Alabama rent          747
#> 3 Alaska  income      32940
#> 4 Alaska  rent         1200
#> 5 Arizona income      27517
#> # ℹ 99 more rows
```

Para deixar ela tidy, precisamos usar a função `pivot_wider()`.

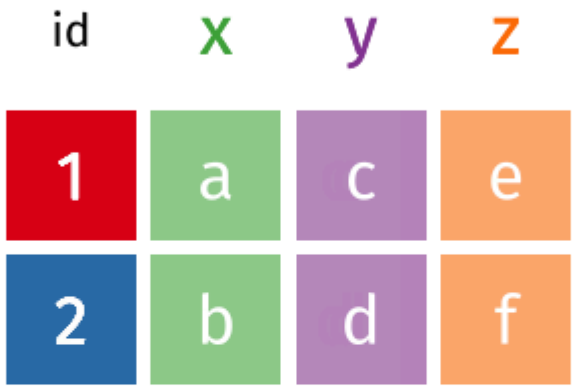
pivot_wider()

```
renda |>
  pivot_wider(
    names_from = variable, # Nomes das colunas vêm de `variable`
    values_from = estimate # Valores das colunas vêm de `estimate`
  )
```

```
#> # A tibble: 52 × 3
#>   NAME       income  rent
#>   <chr>       <dbl> <dbl>
#> 1 Alabama     24476   747
#> 2 Alaska      32940  1200
#> 3 Arizona     27517   972
#> 4 Arkansas    23789   709
#> 5 California  29454  1358
#> # ℹ 47 more rows
```

Animação por Garrick Aden-Buie

wide



dplyr

Motivação

No *R para Ciência de Dados I*, já aprendemos os verbos mais importantes do dplyr:

- `select()`: selecionar colunas.
- `arrange()`: ordenar linhas.
- `filter()`: filtrar linhas.
- `mutate()`: modificar e criar colunas.
- `group_by()`: agrupar linhas.
- `summarise()`: sumarizar colunas.
- `left_join()`: juntar tabelas.

O que mais falta aprender se essas funções já resolvem praticamente todos os problemas de manipulação? Talvez possamos usá-las *melhor*...

Ilustração por @allison_horst

Tarefas repetitivas

É bastante comum lidar com tabelas que têm muitas colunas parecidas e nas quais precisamos fazer o mesmo tratamento. Por causa de uma falha no Excel, todas as colunas da tabela abaixo foram lidas como texto (`<chr>`) .

```
#> # A tibble: 3 × 10
#>   Aluno Nota1 Nota2 Nota3 Nota4 Nota5 Nota6 Nota7 Nota8 Nota9
#>   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
#> 1 Ana   7.0   1.6   7.7   3.9   9.5   10.0  0.9   7.9   3.6
#> 2 Bruno 9.2   1.5   1.3   5.1   5.8   7.1   5.5   8.7   3.7
#> 3 Caio  5.7   7.4   1.9   0.8   2.3   6.3   3.3   1.3   1.5
```

Para transformar todas as `NotaX` juntas, poderíamos usar `pivot_longer()` e aplicar a transformação na coluna resultante. O problema é que nem sempre queremos reestruturar a tabela inteira para fazer uma transformação simples.

Montando frases com mutate()

Nós sabemos transformar uma tabela (`mutate()`), selecionar as colunas desejadas (`select(Nota1:Nota9)`) e converter textos para números (`as.numeric()`). Só ainda não aprendemos uma maneira de repetir esse processo para muitas colunas sem precisar escrever a mesma coisa várias vezes.

```
alunos |>
  mutate(
    Nota1 = as.numeric(Nota1),
    Nota2 = as.numeric(Nota2),
    # ... O que fazer se tivéssemos 500 colunas?
  )
```

Vamos pensar em uma frase que descreva a operação que queremos fazer:

> *Aplicar uma transformação **ao longo** das colunas `Nota1` a `Nota9` usando a função `as.numeric()`.*

across()

```
alunos |>
  mutate(                    # Aplicar uma transformação...
    across(                  # ...ao longo...
      .cols = Nota1:Nota9, # ...das colunas `Nota1` a `Nota9`...
      .fns = as.numeric      # ...usando a função `as.numeric()`.
    )
  )
```

```
#> # A tibble: 3 × 10
#>   Aluno Nota1 Nota2 Nota3 Nota4 Nota5 Nota6 Nota7 Nota8 Nota9
#>   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 Ana     7     1.6   7.7   3.9   9.5  10     0.9   7.9   3.6
#> 2 Bruno   9.2   1.5   1.3   5.1   5.8   7.1   5.5   8.7   3.7
#> 3 Caio    5.7   7.4   1.9   0.8   2.3   6.3   3.3   1.3   1.5
```

Argumentos da across()

`.cols`, o primeiro argumento da `across()`, é uma seleção de variáveis idêntica à que usaríamos na `select()`; `.fns`, o segundo, é um **nome** de uma função, ou seja, a função sem os parênteses na frente. Não precisamos explicitar eles sempre.

```
# Expressões equivalentes à do slide anterior
alunos |>
  mutate(across(Nota1:Nota9, as.numeric))

alunos |>
  mutate(across(-Aluno, as.numeric))

alunos |>
  mutate(across(starts_with("Nota"), as.numeric))

alunos |>
  mutate(across(c(2, 3, 4, 5, 6, 7, 8, 9, 10), as.numeric)) # c()!
```

Um universo de possibilidades

A `across()` já seria muito boa se ela só fizesse o que vimos até agora, mas ela é capaz de muito mais. Vamos usar de exemplo a tabela a seguir: dados sobre vários personagens do universo Star Wars.

```
estrelas <- select(starwars, 1:6)
estrelas
```

```
#> # A tibble: 87 × 6
#>   name           height  mass hair_color skin_color  eye_color
#>   <chr>           <int> <dbl> <chr>      <chr>       <chr>
#> 1 Luke Skywalker    172    77 blond      fair        blue
#> 2 C-3PO             167    75 <NA>       gold        yellow
#> 3 R2-D2              96    32 <NA>       white, blue red
#> 4 Darth Vader       202   136 none       white       yellow
#> 5 Leia Organa       150    49 brown      light       brown
#> # ℹ 82 more rows
```

where()

A função `where()` é amiga íntima da `across()`. Com ela, nós podemos selecionar colunas baseadas nas suas características ao invés de seus nomes. Abaixo vamos ver como aplicar a `toupper()` em todas as colunas de texto:

> *Aplicar uma transformação ao longo das colunas* ***onde*** *`is.character()` é verdadeira usando a função `toupper()`.*

```
estrelas |>
  mutate(                    # Aplicar uma transformação...
    across(                  # ...ao longo das colunas...
      where(is.character),   # ...onde `is.character()` é verdadeira...
      toupper                # ...usando a função `toupper()`.
    )
  )
```

P.S.: inclusive podemos usar a `where()` dentro da `select()`!

Argumento da where()

O único argumento da `where()` precisa ser o nome de um **predicado**, ou seja, de uma função que retorne `TRUE` para as colunas que você quer e `FALSE` caso contrário.

```
# Equivalente ao slide anterior
estrelas |>
  mutate(across(where(is.character), toupper))
```

```
#> # A tibble: 87 × 6
#>   name           height  mass hair_color skin_color  eye_color
#>   <chr>           <int> <dbl> <chr>      <chr>       <chr>
#> 1 LUKE SKYWALKER    172    77 BLOND      FAIR        BLUE
#> 2 C-3PO             167    75 <NA>       GOLD        YELLOW
#> 3 R2-D2              96    32 <NA>       WHITE, BLUE RED
#> 4 DARTH VADER       202   136 NONE       WHITE       YELLOW
#> 5 LEIA ORGANA       150    49 BROWN      LIGHT       BROWN
#> # ℹ 82 more rows
```

Combinando seleções

Não estamos limitados a apenas um seletor por `across()`: no exemplo abaixo, estamos aplicando `toupper()` às colunas textuais **exceto** `name`. A sintaxe continua igual à da `select()`.

```
estrelas |>
  mutate(across(c(where(is.character), -name), toupper)) # c()!
```

```
#> # A tibble: 87 × 6
#>   name           height  mass hair_color skin_color  eye_color
#>   <chr>           <int> <dbl> <chr>      <chr>       <chr>
#> 1 Luke Skywalker    172    77 BLOND      FAIR        BLUE
#> 2 C-3PO             167    75 <NA>       GOLD        YELLOW
#> 3 R2-D2              96    32 <NA>       WHITE, BLUE RED
#> 4 Darth Vader       202   136 NONE       WHITE       YELLOW
#> 5 Leia Organa       150    49 BROWN      LIGHT       BROWN
#> # ℹ 82 more rows
```

Outros verbos e across()

A `across()` funciona com outros verbos do dplyr além da `mutate()`: no exemplo abaixo, estamos tirando a média de todas as colunas numéricas. Note que agora precisamos usar `where(is.numeric)`.

```
estrelas |>
  summarise(across(where(is.numeric), mean))
```

```
#> # A tibble: 1 × 2
#>   height  mass
#>    <dbl> <dbl>
#> 1     NA    NA
```

Não teve erro nenhum na execução! O problema foi que temos personagens com altura ou peso desconhecidos na nossa tabela e, no R, qualquer operação matemática que tenha um `NA` no meio retorna `NA` também.

Funções próprias na across()

Podemos usar qualquer função no argumento `.fns` da `across()`, com a condição de que ela receba *apenas um* argumento. No caso do slide anterior, precisaríamos passar a coluna e `na.rm = TRUE`. A solução é criar nossa própria função!

```
media_limpa <- function(x) {
  mean(x, na.rm = TRUE)
}

estrelas |>
  summarise(across(where(is.numeric), media_limpa))
```

```
#> # A tibble: 1 × 2
#>   height  mass
#>    <dbl> <dbl>
#> 1   174.  97.3
```

Várias across() de uma vez

Agora que conseguimos sumarizar as as colunas numéricas com `across()`, podemos tentar sumarizar as colunas textuais também... De uma vez só. A `n_distinct()` retorna o número de elementos distintos em uma coluna.

```
estrelas |>
  summarise(
    across(where(is.numeric), media_limpa),
    across(where(is.character), n_distinct)
  )
```

```
#> # A tibble: 1 × 6
#>   height  mass  name hair_color skin_color eye_color
#>    <dbl> <dbl> <int>      <int>      <int>     <int>
#> 1   174.  97.3    87         13         31        15
```

Várias funções em uma across()

Se passarmos uma lista nomeada para a `across()`, podemos aplicar mais de uma função ao mesmo tempo. Falaremos mais sobre listas na aula de purrr.

```
estrelas |>
  summarise(
    across(
      .cols = where(is.numeric),
      .fns = list("media" = media_limpa, "distintos" = n_distinct)
    )
  )
```

```
#> # A tibble: 1 × 4
#>   height_media height_distintos mass_media mass_distintos
#>          <dbl>            <int>      <dbl>          <int>
#> 1         174.               46       97.3             39
```

Rodada bônus!

O tidyr tem mais uma função que pode vir a calhar: `unite()`.

```
estrelas |>
  pivot_longer(4:6, names_to = "variaveis", values_to = "cor") |>
  unite("cor_orgao", c(cor, variaveis), sep = "_")
```

```
#> # A tibble: 261 × 4
#>   name            height  mass cor_orgao
#>   <chr>            <int> <dbl> <chr>
#> 1 Luke Skywalker     172    77 blond_hair_color
#> 2 Luke Skywalker     172    77 fair_skin_color
#> 3 Luke Skywalker     172    77 blue_eye_color
#> 4 C-3PO              167    75 NA_hair_color
#> 5 C-3PO              167    75 gold_skin_color
#> # ℹ 256 more rows
```

Fim