

# R para Ciência de Dados 2

Dplyr 1.0 e Tidyr



Novembro de 2021

Dplyr 1.0

# O que já sabemos

Já vimos que com os principais verbos do `dplyr` já conseguimos fazer diversas operações de manipulação de bases de dados.

- Selecionar colunas: `select()`
- Ordenar linhas: `arrange()`
- Filtrar linhas: `filter()`
- Criar ou modificar colunas: `mutate()`
- Agrupar e sumarizar: `group_by()` + `summarise()`



# O novo dplyr

A versão 1.0 do pacote `dplyr` foi oficialmente lançada em junho de 2020 e contou com diversas novidades. Vamos falar das principais mudanças:

- A nova função `across()`, que facilita aplicar uma mesma operação em várias colunas.
- A repaginada função `rowwise()`, para fazer operações por linha.
- Novas funcionalidades das funções `select()` e `rename()` e a nova função `relocate()`.

# Motivação

Base de dados de venda de casas na cidade de Ames, nos Estados Unidos.

- 2930 linhas e 77 colunas.
- Cada linha corresponde a uma casa vendida e cada coluna a uma característica da casa ou da venda.
- Versão traduzida

```
install.packages("remotes")  
remotes::install_github("cienciadedatos/dados")
```

- Base original:

```
install.packages("AmesHousing")  
data(ames_raw, package = "AmesHousing")
```

# across()

A função `across()` substitui as famílias `*_all()`, `*_if` e `*_at()`. A ideia é facilitar a aplicação de uma operação a diversas colunas da base. Antigamente fazíamos:

```
casas <- dados::casas
casas %>%
  group_by(geral_qualidade) %>%
  summarise(
    lote_area_media = mean(lote_area, na.rm = TRUE),
    venda_valor_medio = mean(venda_valor, na.rm = TRUE)
  )
```

```
#> # A tibble: 10 × 3
#>   geral_qualidade lote_area_media venda_valor_medio
#>   <chr>           <dbl>           <dbl>
#> 1 abaixo da média      8464.           106485.
#> 2 acima da média      9788.           162130.
#> 3 boa                10309.          205026.
#> 4 excelente          12777.          368337.
#> 5 média              9996.           134753.
#> # ... with 5 more rows
```

# across()

Com a nova função `across()`, a sintaxe é parecida `summarise_at()`, não precisamos mais usar `vars()` nem `~nome_da_funcao(.x)` para definir a função aplicada nas colunas.

```
casas %>%
  group_by(geral_qualidade) %>%
  summarise(across(
    .cols = c(lote_area, venda_valor), # Variáveis
    .fns  = mean, na.rm = TRUE        # Função
  ))
```

```
#> # A tibble: 10 × 3
#>   geral_qualidade lote_area venda_valor
#>   <chr>          <dbl>      <dbl>
#> 1 abaixo da média    8464.    106485.
#> 2 acima da média    9788.    162130.
#> 3 boa              10309.    205026.
#> 4 excelente        12777.    368337.
#> 5 média            9996.    134753.
#> # ... with 5 more rows
```

# across()

Usando `across()`, podemos facilmente aplicar uma função em todas as colunas da nossa base. Abaixo, calculamos o número de valores distintos para todas as variáveis da base `casas`.

```
# Pegando apenas 5 colunas por questão de espaço
casas %>%
  summarise(across(.fns = n_distinct)) %>%
  select(1:5)
```

```
#> # A tibble: 1 × 5
#>   ordem  pid moradia_classe moradia_zoneamento lote_fachada
#>   <int> <int>          <int>          <int>          <int>
#> 1   2930  2930             16             7             129
```

O padrão do parâmetro `.cols` é `everything()`, que representa "todas as colunas". Anteriormente, teríamos que utilizar a função `summarise_all()`.



# across()

Se quisermos selecionar as colunas a serem modificadas a partir de um teste lógico, utilizamos o ajudante `where()`. No exemplo abaixo, calculamos o número de valores distintos das colunas de categóricas:

```
# Pegando apenas 5 colunas por questão de espaço
casas %>%
  summarise(across(where(is.character), n_distinct)) %>%
  select(1:5)
```

```
#> # A tibble: 1 × 5
#>   pid moradia_classe moradia_zoneamento rua_tipo beco_tipo
#>   <int>          <int>          <int>    <int>    <int>
#> 1  2930             16             7         2         3
```

Todas as colunas da base resultante eram colunas com classe `character` na base `casas`. Anteriormente, teríamos que utilizar a função `summarise_if()`.

# across()

Você também pode combinar as ações do `summarise_if()` e `summarise_at()` em um único `across()`. Calculamos as áreas médias, garantindo que pegamos apenas variáveis numéricas.

```
# Pegando apenas 5 colunas por questão de espaço
casas %>%
  summarise(across(where(is.numeric) & contains("_area"), mean, na.rm = TRUE)) %>%
  select(1:5)
```

```
#> # A tibble: 1 × 5
```

```
#>   lote_area alvenaria_area porao_area_com_acabamento_1 porao_area_com_acabamento_2 porao_ar
#>   <dbl>      <dbl>                <dbl>                <dbl>
#> 1   10148.        102.                443.                49.7
```

# across()

Isso não era possível utilizando apenas as funções `summarise()`, `summarise_if()` e `summarise_at()`:

```
casas %>%
  group_by(fundacao_tipo) %>%
  summarise(
    across(contains("area"), mean, na.rm = TRUE),
    across(where(is.character), ~sum(is.na(.x))),
    n_obs = n(),
  ) %>%
  select(1:2, 19:20, n_obs)
```

```
#> # A tibble: 6 × 5
#>   fundacao_tipo      lote_area   pid moradia_classe n_obs
#>   <chr>          <dbl> <int>         <int> <int>
#> 1 bloco de concreto    10616.     0           0    1244
#> 2 concreto derrramado  10054.     0           0    1310
#> 3 laje                10250.     0           0     49
#> 4 madeira              9838.     0           0      5
#> 5 pedra                8659.     0           0     11
#> # ... with 1 more row
```

# across() outros verbos

Embora a nova sintaxe, usando `across()`, não seja muito diferente do que fazíamos antes, realizar sumarizações complexas não é a única vantagem desse novo *framework*.

O `across()` pode ser utilizado em todos os verbos do `dplyr` (com exceção do `select()` e `rename()`), já que ele não traz vantagens com relação ao que já podia ser feito) e isso unifica o modo que trabalhamos essas operações no `dplyr`. Em vez de termos uma família de funções para cada verbo, temos agora apenas o próprio verbo e o `across()`.

Vamos ver um exemplo para o `mutate()` e para o `filter()`.

# across()

O código abaixo transforma todas as variáveis que possuem "area" no nome, passando os valores de pés quadrados para metros quadrados.

```
casas %>%  
  mutate(across(  
    contains("area"),  
    ~.x / 10.764  
  ))
```

Já o código a seguir filtra apenas as casas que possuem varanda aberta, cerca e lareira.

```
casas %>%  
  filter(across(  
    c(varanda_aberta_area, cerca_qualidade, lareira_qualidade),  
    ~!is.na(.x)  
  ))
```

# select()

Não precisamos do `across()` na hora de selecionar colunas. A função `select()` já usa naturalmente o mecanismo de seleção de colunas que o `across()` proporciona.

```
# Pegando apenas 5 colunas por questão de espaço
casas %>%
  select(where(is.numeric)) %>%
  select(1:5)
```

```
#> # A tibble: 2,930 × 5
#>   ordem lote_fachada lote_area construcao_ano remodelacao_ano
#>   <int>         <int>      <int>          <int>          <int>
#> 1     1           141    31770          1960          1960
#> 2     2            80    11622          1961          1961
#> 3     3            81    14267          1958          1958
#> 4     4            93    11160          1968          1968
#> 5     5            74    13830          1997          1998
#> # ... with 2,925 more rows
```

# rename()

O mesmo vale para o `rename()`. Se quisermos renomear várias colunas, a partir de uma função, utilizamos o `rename_with()`.

```
# Pegando apenas 5 colunas por questão de espaço
casas %>%
  rename_with(toupper, contains("venda")) %>%
  select(contains("VENDA"))
```

```
#> # A tibble: 2,930 × 5
#>   VENDA_MES VENDA_ANO VENDA_TIPO VENDA_CONDICAO VENDA_VALOR
#>   <int>      <int> <chr>      <chr>          <int>
#> 1         5      2010 "WD "        venda normal    215000
#> 2         6      2010 "WD "        venda normal    105000
#> 3         6      2010 "WD "        venda normal    172000
#> 4         4      2010 "WD "        venda normal    244000
#> 5         3      2010 "WD "        venda normal    189900
#> # ... with 2,925 more rows
```

# relocate()

O dplyr possui agora uma função própria para reorganizar colunas: `relocate()`. Por padrão, ela coloca uma ou mais colunas no começo da base.

```
# Pegando apenas 5 colunas por questão de espaço
casas %>%
  relocate(venda_valor, venda_tipo) %>%
  select(1:5)
```

```
#> # A tibble: 2,930 × 5
#>   venda_valor venda_tipo ordem pid      moradia_classe
#>   <int> <chr>      <int> <chr>      <chr>
#> 1     215000 "WD "           1 0526301100 020
#> 2     105000 "WD "           2 0526350040 020
#> 3     172000 "WD "           3 0526351010 020
#> 4     244000 "WD "           4 0526353030 020
#> 5      189900 "WD "           5 0527105010 060
#> # ... with 2,925 more rows
```



# relocate()

Podemos usar os argumentos `.after` e `.before` para fazer mudanças mais complexas.

O código baixo coloca a coluna `venda_ano` depois da coluna `construcao_ano`.

```
casas %>%  
  relocate(venda_ano, .after = construcao_ano)
```

O código baixo coloca a coluna `venda_ano` antes da coluna `construcao_ano`.

```
casas %>%  
  relocate(venda_ano, .before = construcao_ano)
```

# rowwise()

Por fim, vamos discutir operações feitas por linha. Tome como exemplo a tabela abaixo:

```
tab_notas <- tibble(  
  student_id = 1:5,  
  prova1 = sample(0:10, 5),  
  prova2 = sample(0:10, 5),  
  prova3 = sample(0:10, 5),  
  prova4 = sample(0:10, 5)  
)  
  
tab_notas
```

```
#> # A tibble: 5 × 5  
#>   student_id prova1 prova2 prova3 prova4  
#>       <int>  <int>  <int>  <int>  <int>  
#> 1         1      3      3     10      1  
#> 2         2      6      4      1      3  
#> 3         3      0     10      4      4  
#> 4         4     10      8      8      5  
#> 5         5      7      6      7      2
```

# rowwise()

Se quisermos gerar uma coluna com a nota média de cada aluno nas quatro provas, não poderíamos usar o `mutate()` diretamente.

```
tab_notas %>% mutate(media = mean(c(prova1, prova2, prova3, prova4)))
```

```
#> # A tibble: 5 × 6
#>   student_id prova1 prova2 prova3 prova4 media
#>   <int>    <int>   <int>   <int>   <int> <dbl>
#> 1         1         3     3      10     1  5.1
#> 2         2         6     4       1     3  5.1
#> 3         3         0    10       4     4  5.1
#> 4         4        10     8       8     5  5.1
#> 5         5         7     6       7     2  5.1
```

Neste caso, todas as colunas estão sendo empilhadas e gerando uma única média, passada a todas as linhas da coluna `media`.

# rowwise()

Para fazermos a conta para cada aluno, podemos agrupar por aluno. Agora sim a média é calculada apenas nas notas de cada estudante.

```
tab_notas %>%  
  group_by(student_id) %>%  
  mutate(media = mean(c(prova1, prova2, prova3, prova4)))
```

```
#> # A tibble: 5 × 6  
#> # Groups:   student_id [5]  
#>   student_id prova1 prova2 prova3 prova4 media  
#>       <int>   <int>   <int>   <int>   <int> <dbl>  
#> 1         1       3       3      10       1  4.25  
#> 2         2       6       4       1       3  3.5  
#> 3         3       0      10       4       4  4.5  
#> 4         4      10       8       8       5  7.75  
#> 5         5       7       6       7       2  5.5
```

# rowwise()

E se não tivermos uma coluna ID? Então basta usarmos a `rowwise()`, uma função que automaticamente agrupa pelo identificador das linhas.

```
tab_notas %>%  
  rowwise() %>%  
  mutate(media = mean(c(prova1, prova2, prova3, prova4)))
```

```
#> # A tibble: 5 × 6  
#> # Rowwise:  
#>   student_id prova1 prova2 prova3 prova4 media  
#>       <int>  <int>  <int>  <int>  <int> <dbl>  
#> 1         1      3      3     10      1  4.25  
#> 2         2      6      4      1      3  3.5  
#> 3         3      0     10      4      4  4.5  
#> 4         4     10      8      8      5  7.75  
#> 5         5      7      6      7      2  5.5
```

# c\_across()

Também podemos nos aproveitar da sintaxe do `across()` neste caso. Para isso, precisamos substituir a função `c()` pela função `c_across()`.

```
tab_notas %>%  
  rowwise() %>%  
  mutate(media = mean(c_across(starts_with("prova"))))
```

```
#> # A tibble: 5 × 6  
#> # Rowwise:  
#>   student_id prova1 prova2 prova3 prova4 media  
#>   <int>    <int>   <int>   <int>   <int> <dbl>  
#> 1         1         3     3     10     1  4.25  
#> 2         2         6     4      1     3  3.5  
#> 3         3         0    10     4     4  4.5  
#> 4         4        10     8     8     5  7.75  
#> 5         5         7     6     7     2  5.5
```

# Miscelânea de funções úteis

Para aquecer, vamos listar uma miscelânea de funções muito úteis, mas menos conhecidas do `dplyr`.

- `bind_rows()`: para empilhar duas bases.
- `case_when()`: generalização da `ifelse()` para várias condições.
- `first()`, `last()`: para pegar o primeiro ou último valor de um vetor/coluna.
- `na_if()`: para transformar um determinado valor de um vetor/coluna em NA.
- `coalesce()`: para substituir os NAs de uma coluna pelos valores equivalentes de uma segunda coluna.
- `lag()`, `lead()`: para gerar colunas defasadas.
- `pull()`: para transformar uma coluna da base em um vetor.
- `slice_sample`: para gerar amostras da base.

Tidyr



# Dados arrumados

Dentro do `tidyverse`, uma base *tidy* é uma base fácil de se trabalhar, isto é, fácil de se fazer manipulação de dados, fácil de se criar visualizações, fácil de se ajustar modelos e por aí vai.

Na prática, uma base *tidy* é aquela que se encaixa bem no *framework* do `tidyverse`, pois os pacotes como o `dplyr` e o `ggplot2` foram desenvolvidos para funcionar bem com bases *tidy*. E assim como esses pacotes motivaram o uso de bases *tidy*, o conceito *tidy* motiva o surgimento de novos *frameworks*, como o `tidymodels` para modelagem.

As duas propriedades mais importantes de uma base *tidy* são:

- Cada coluna é uma variável;
- Cada linha é uma observação.

Essa definição proporciona uma maneira consistente de se referir a variáveis (nomes de colunas) e observações (índices das linhas).

# O pacote tidyr

O pacote `tidyr` possui funções que nos ajudam a deixar uma base bagunçada em uma base *tidy*. Ou então, nos ajudam a bagunçar um pouquinho a nossa base quando isso nos ajudar a produzir o resultados que queremos.

Vamos ver aqui algumas de suas principais funções:

- `separate()` e `unite()`: para separar variáveis concatenadas em uma única coluna ou uni-las.
- `pivot_wider()` e `pivot_longer()`: para pivotar a base.

# Motivação

Como motivação para utilizar essas funções, vamos utilizar a nossa boa e velha base imdb.

```
imdb <- readr::read_rds("../data/imdb.rds")
```

```
imdb
```

```
#> # A tibble: 3,713 × 15
```

```
#>   titulo
```

```
#>   <chr>
```

```
#> 1 Avatar
```

```
#> 2 Pirates of the Caribbean: At World's End
```

```
#> 3 The Dark Knight Rises
```

```
#> 4 John Carter
```

```
#> 5 Spider-Man 3
```

```
#> # ... with 3,708 more rows
```

ano	diretor	duracao	cor	generos	pais	clas
<int>	<chr>	<int>	<chr>	<chr>	<chr>	<chr>
2009	James C...	178	Color	Action ...	USA	A pa
2007	Gore Ve...	169	Color	Action ...	USA	A pa
2012	Christo...	164	Color	Action ...	USA	A pa
2012	Andrew ...	132	Color	Action ...	USA	A pa
2007	Sam Rai...	156	Color	Action ...	USA	A pa

# separate()

A função `separate()` separa duas ou mais variáveis que estão concatenadas em uma mesma coluna. Como exemplo, vamos transformar a coluna `generos` da base IMDB em três colunas, cada uma com um dos gêneros do filme. Lembrando que os valores da coluna `generos` estão no seguinte formato:

```
imdb %>% pull(generos) %>% head()
```

```
#> [1] "Action|Adventure|Fantasy|Sci-Fi"  
#> [3] "Action|Thriller"  
#> [5] "Action|Adventure|Romance"
```

```
"Action|Adventure|Fantasy"  
"Action|Adventure|Sci-Fi"  
"Adventure|Animation|Comedy"
```

# separate()

Veja que agora, temos 3 colunas de gênero. Filmes com menos de 3 gêneros recebem NA na coluna genero2 e/ou genero3. Os gêneros sobressalentes são descartados, assim como a coluna generos original.

```
imdb %>%  
  separate(col = generos, into = c("genero1", "genero2", "genero3"), sep = "\\|")
```

```
#> # A tibble: 3,713 × 17  
#>   titulo      ano diretor duracao cor genero1 genero2 genero3 pais classificac  
#>   <chr>      <int> <chr>    <int> <chr> <chr>    <chr>    <chr>    <chr>    <chr>  
#> 1 Avatar      2009 James Cam...   178 Color Action Adventu... Fantasy USA    A partir de  
#> 2 Pirates of the... 2007 Gore Verb...   169 Color Action Adventu... Fantasy USA    A partir de  
#> 3 The Dark Knigh... 2012 Christoph...   164 Color Action Thriller <NA>    USA    A partir de  
#> 4 John Carter      2012 Andrew St...   132 Color Action Adventu... Sci-Fi  USA    A partir de  
#> 5 Spider-Man 3      2007 Sam Raimi     156 Color Action Adventu... Romance USA    A partir de  
#> # ... with 3,708 more rows
```

# unite()

A função `unite()` realiza a operação inversa da função `separate()`. Como exemplo, vamos agora transformar as colunas `ator1`, `ator2` e `ator3` em uma única coluna `atores`. Lembrando que essas colunas estão no formato abaixo.

```
imdb %>% select(starts_with("ator"))
```

```
#> # A tibble: 3,713 × 3
#>   ator_1      ator_2      ator_3
#>   <chr>      <chr>      <chr>
#> 1 CCH Pounder Joel David Moore Wes Studi
#> 2 Johnny Depp Orlando Bloom  Jack Davenport
#> 3 Tom Hardy   Christian Bale  Joseph Gordon-Levitt
#> 4 Daryl Sabara Samantha Morton Polly Walker
#> 5 J.K. Simmons James Franco   Kirsten Dunst
#> # ... with 3,708 more rows
```

# unite()

Veja que agora a coluna `elenco` possui os 3 atores/atrizes concatenados. Se a ordem das colunas `ator1`, `ator2` e `ator3` nos trazia a informação de protagonismo, essa informação passa a ficar implícita nesse novo formato. As 3 colunas originais são removidas da base resultante.

```
imdb %>%  
  unite(col = "elenco", starts_with("ator"), sep = " - ") %>%  
  select(elenco)
```

```
#> # A tibble: 3,713 × 1  
#>   elenco  
#>   <chr>  
#> 1 CCH Pounder - Joel David Moore - Wes Studi  
#> 2 Johnny Depp - Orlando Bloom - Jack Davenport  
#> 3 Tom Hardy - Christian Bale - Joseph Gordon-Levitt  
#> 4 Daryl Sabara - Samantha Morton - Polly Walker  
#> 5 J.K. Simmons - James Franco - Kirsten Dunst  
#> # ... with 3,708 more rows
```

# Pivotagem

O conceito de pivotagem no *tidyverse* se refere a mudança da estrutura da base, geralmente para alcançar o formato *tidy*.

Geralmente realizamos pivotagem quando nossas linhas não são unidades observacionais ou nossas colunas não são variáveis. Ela é similar à pivotagem do Excel, mas um pouco mais complexa.

O ato de pivotar resulta em transformar uma base de dados *long* em *wide* e vice-versa.

Uma base no formato *long* possui mais linhas e pode ter menos colunas, enquanto no formato *wide* possui menos linhas e pode ter mais colunas

Esses formatos são sempre relativos às colunas que estão sendo pivotadas, sendo que uma base *tidy* pode estar tanto no formato *long* quanto *wide*.



# Pivotagem

Antigamente, utilizávamos as funções `gather()` e `spread()` para fazer as operações de pivotagem. A fonte da imagem é [este site](#).

# pivot\_longer()

Agora, no lugar de `gather()`, utilizamos a função `pivot_longer()`. Abaixo, transformamos as colunas `ator1`, `ator2` e `ator3` em duas colunas: `ator_atriz` e `protagonismo`.

```
imdb %>%
  pivot_longer(
    cols = starts_with("ator"),
    names_to = "protagonismo",
    values_to = "ator_atriz"
  ) %>%
  select(titulo, ator_atriz, protagonismo)
```

```
#> # A tibble: 11,139 × 3
#>   titulo                                ator_atriz    protagonismo
#>   <chr>                                <chr>         <chr>
#> 1 Avatar                               CCH Pounder   ator_1
#> 2 Avatar                               Joel David Moore ator_2
#> 3 Avatar                               Wes Studi     ator_3
#> 4 Pirates of the Caribbean: At World's End Johnny Depp   ator_1
#> 5 Pirates of the Caribbean: At World's End Orlando Bloom ator_2
#> # ... with 11,134 more rows
```

# pivot\_longer()

Se considerarmos que na análise da base IMDB cada observação deve ser um filme, então essa nova base já não mais *tidy*, pois agora cada filme aparece em três linhas diferentes, uma vez para cada um de seus atores.

Nesse sentido, embora possa parecer que a variável `ator_protagonismo` estava implícita na base original, ela não é uma variável de fato. Todos filmes tem um `ator_1`, um `ator_2` e um `ator_3`. Não existe nenhuma informação sobre o filme que podemos tirar da coluna `ator_protagonismo`, pois ela qualifica apenas os atores, não o filme em si.

# pivot\_wider()

A função `pivot_wider()` faz a operação inversa da `pivot_longer()`. Sem aplicarmos as duas funções em sequência, voltamos para a base original.

```
imdb_long %>%  
  pivot_wider(  
    names_from = "protagonismo",  
    values_from = "ator_atriz"  
  ) %>%  
  select(1:3, starts_with("ator"))
```

```
#> # A tibble: 3,713 × 6
```

```
#>   titulo
```

```
#>   <chr>
```

```
#> 1 Avatar
```

```
#> 2 Pirates of the Caribbean: At World's End
```

```
#> 3 The Dark Knight Rises
```

```
#> 4 John Carter
```

```
#> 5 Spider-Man 3
```

```
#> # ... with 3,708 more rows
```

	ano	diretor	ator_1	ator_2
	<int>	<chr>	<chr>	<chr>
	2009	James Cameron	CCH Pounder	Joel David
	2007	Gore Verbinski	Johnny Depp	Orlando Blo
	2012	Christopher Nolan	Tom Hardy	Christian B
	2012	Andrew Stanton	Daryl Sabara	Samantha Mo
	2007	Sam Raimi	J.K. Simmons	James Franc

# Referências

- Material de tidyverse da UFPR
- Livro da Curso-R
- Apresentação Garret Grolemund
- Excelente blog post sobre manipulação de bases