Análise Estatística de Dados com R

PET Estatística UFC

Índice

Todo esse material foi feito com auxílio do livro "Data Science for R (2e)". Embora esteja em inglês, é um material para uso livre e que vai bem mais longe do que o conteúdo que a gente vai tratar aqui.

1 Algumas dicas

1.1 Alguns atalhos úteis do Rstudio

- Mostrar todos os atalhos do Rstudio: *Alt, shift, K*
- Usar a atribuição (->): *Alt, -*
- Usar o pipe (|>): *Ctrl, Shift, M*
- Mostrar a paleta de comandos: *Ctrl, Shift, P*
- Criar uma seção: *Ctrl, Shift, R*
- Salvar tudo: *Ctrl, Alt, S*
- Rodar uma linha do código: *Ctrl, Enter*
- Rodar todo o código: *Ctrl, Alt, R*
- Ir para o Script: *Ctrl, 1*
- Ir para o console: *Ctrl, 2*

- Transformar/Reverter comentário: *Ctrl, Shift, C*
- Mostrar pacotes (agiliza instalação): *Ctrl, 7*
- Sair do Script: *Ctrl, W*
- Sair de todos os Scripts: *Ctrl, Shift, W*
- Criar novo script: *Ctrl, shift, N*
- Finalizar seção: *Ctrl, Q*
- Reindentar código: *Ctrl, I*
- Andar entre seções: *Alt, Shift, J*
- Andar para próximo script: *Ctrl, Tab*

1.2 Ajude seu eu do futuro

1.2.1 Comentários

Use o símbolo # para fazer comenstários. Você, ao fazer comentários de um código, deve explicar o por que do código, não como ele funciona (coisa que o seu eu do futuro pode descobrir por si mesmo).

1.2.2 Nomeação de variáveis

Use o símbolo <- para guardar algum valor em uma variável. As variáveis devem seguir um padrão, recomendamos o snake_case, o qual contém letras minúsculas separadas por "_". Coloque nomes descritivos que falam o que é a variável, não se incomode com nomes um pouco grandes, você pode usar o *TAB* para procurar seu objeto. Seu eu do futuro vai economizar muito tempo com nomes que deixam claro o que é aquela variável.

1.2.3 Organização de código

Um código bem organizado é muito mais fácil de ler do que um que não segue um padrão. Siga as seguintes recomendações:

1.2.3.1 Use espaço nos lugares certos

- Use espaço nos dois lados de operadores (+, -, *, /, >, <, ==, &, |, etc);
- Não use espaços nos parênteses , nem em aspas e nem em expoentes ((,), ^, ', “);
- Use espaços na atribuição (|>, <-, =);
- Não tem problema adicionar espaços extras para melhorar o alinhamento, veja o exemplo seguinte:

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
library(nycflights13)

flights |>
  mutate(
    speed      = distance / air_time,
    dep_hour   = dep_time %/% 100,
    dep_minute = dep_time %%  100
    )
```

```
# A tibble: 336,776 x 22
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
```

```
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 14 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>, speed <dbl>, dep_hour <dbl>,
#   dep_minute <dbl>
```

1.2.3.2 Quebra de linhas com o pipe

Sempre use um espaço antes do pipe (|>) e após isso use a quebra de linhas. Funções com variáveis nomeadas também devem ter quebra de linha depois do parênteses de abertura da função e sempre que passar de uma variável nomeada para outra. Veja o exemplo.

```r
flights |>
  group_by(tailnum) |>
  summarize(
    delay = mean(arr_delay, na.rm = TRUE),
    n = n()
  )
```

```
# A tibble: 4,044 x 3
   tailnum  delay     n
   <chr>    <dbl> <int>
 1 D942DN  31.5       4
 2 N0EGMQ   9.98    371
 3 N10156  12.7     153
 4 N102UW   2.94     48
 5 N103US  -6.93     46
 6 N104UW   1.80     47
 7 N10575  20.7     289
 8 N105UW  -0.267    45
 9 N107US  -5.73     41
10 N108UW  -1.25     60
# i 4,034 more rows
```

1.3 Projetos no Rstudio

Os projetos no Rstudio são uma ótima forma de organizar os seus arquivos. Inicialmente, recomendamos que você remova a opção que lembra do seu último script e objetos. Para isso siga a instrução: Tools -> Global Options... -> General -> Restore .RData into workspace at startup. Além disso, marque como *Never*. Com isso, sempre que você entrar no Rstudio seu script estará limpo. Agora, crie um projeto em File -> New Project. Ao criar um projeto, o histórico de tudo o que você fazer com os arquivos nele contidos será salvo. Com isso seus projetos estarão mais organizados e com um fácil compartilhamento com outras pessoas.

1.4 Nomeação de arquivos

Os nomes de arquivos devem ser legíveis por máquinas, não podendo ter espaços, símbolos e caracteres especiais, e devem ser legíveis por humanos, com um nome que descreve o que há no arquivo. Caso seus arquivos tiverem uma sequência para serem abertos coloque um número no início do nome, se aproveitando na ordenação por classificação alfabética.

2 Importação de dados

2.1 Arquivos CSV

Existem várias funções para ler cada tipo de arquivo. Para ler arquivos CSV (valores separados por vírgula) você pode utilizar a seguinte função.

```
library(tidyverse)

students <- read_csv(
  'D:/projetos_2023.2/AED_R/base_de_dados/students.csv'
)
```

```
Rows: 6 Columns: 5
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (4): Full Name, favourite.food, mealPlan, AGE
dbl (1): Student ID

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
students
```

```
# A tibble: 6 x 5
  `Student ID` `Full Name`      favourite.food     mealPlan            AGE
         <dbl> <chr>            <chr>              <chr>               <chr>
1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
2            2 Barclay Lynn     French fries       Lunch only          5
3            3 Jayendra Lyne    N/A                Breakfast and lunch 7
4            4 Leon Rossini     Anchovies          Lunch only          <NA>
5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6            6 Güvenç Attila    Ice cream          Lunch only          6
```

A string colocada dentro da função read_csv é o caminho que leva até a base de dados dentro do computador. Dessa forma, provavelmente seu caminho será diferente do que estamos apresentando. Observe que, como tem o 'D:' no caminho, significa que é um caminho absoluto. Sempre use caminhos relativos, os absolutos são péssimos para compartilhar o seu projeto com outras pessoas.

Perceba que na coluna favourite.food temos uma string "N/A". O problema é que o R não vai reconhecer como valor faltante. Logo, nós temos que informar que, além de espaços vazios (o R reconhece espaços vazios, por padrão, como valores faltantes) o R deve reconhecer "N/A" como valor faltante. Veja como fica no código seguinte.

```
library(tidyverse)

students <- read_csv(
  'D:/projetos_2023.2/AED_R/base_de_dados/students.csv',
  na = c("N/A", "")
)
```

```
Rows: 6 Columns: 5
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (4): Full Name, favourite.food, mealPlan, AGE
dbl (1): Student ID

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
students
```

```
# A tibble: 6 x 5
  `Student ID` `Full Name`      favourite.food     mealPlan            AGE
         <dbl> <chr>            <chr>              <chr>               <chr>
1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
2            2 Barclay Lynn     French fries       Lunch only          5
3            3 Jayendra Lyne    <NA>               Breakfast and lunch 7
4            4 Leon Rossini     Anchovies          Lunch only          <NA>
5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6            6 Güvenç Attila    Ice cream          Lunch only          6
```

Observe que tem aspas nas duas primeiras variáveis. Isso ocorre por que elas tem um espaço como separador (recomendamos a barrinha '_'). Para consertá-las você pode usar o rename() que é uma função do Dplyr ou usar o clean_names() do pacote janitor que edita o nome de colunas automaticamente.

```r
# install.packages("janitor")
library(janitor)
```

```
Attaching package: 'janitor'

The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

```r
students |> clean_names()
```

```
# A tibble: 6 x 5
  student_id full_name        favourite_food     meal_plan           age
       <dbl> <chr>            <chr>              <chr>               <chr>
1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
2          2 Barclay Lynn     French fries       Lunch only          5
3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
4          4 Leon Rossini     Anchovies          Lunch only          <NA>
5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6          6 Güvenç Attila    Ice cream          Lunch only          6
```

Observe os tipos de variáveis. Veja que 'meal_plan', por ser uma variável categórica e ter um conjunto de valores possíveis , deve ser representado como fator. Além disso, veja que a variável 'age' é dita como caractere por ter um valor digitado como five ao invés de 5. Para resolver isso fazemos o seguinte código.

```r
students |>
  clean_names() |>
  mutate(
    meal_plan = factor(meal_plan),
    age = parse_number(if_else(age == "five", "5", age ))
  )
```

```
# A tibble: 6 x 5
  student_id full_name        favourite_food     meal_plan             age
       <dbl> <chr>            <chr>              <fct>               <dbl>
1          1 Sunil Huffmann   Strawberry yoghurt Lunch only              4
2          2 Barclay Lynn     French fries       Lunch only              5
3          3 Jayendra Lyne    <NA>               Breakfast and lunch     7
4          4 Leon Rossini     Anchovies          Lunch only             NA
5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch     5
6          6 Güvenç Attila    Ice cream          Lunch only              6
```

A função parse_number() pega o primeiro número que encontra em uma string e descarta todos os caracteres anteriores e posteriores. Dentro dessa função temos o if_else() que, dada a condição proposta, coloca 5 se for verdade e, se não, apenas repete o valor de age.

Também é possível criar manualmente um tibble com o read_csv. Enquanto mostramos isso, também colocaremos uns exemplos de usos de argumentos da função read_csv, os quais são intuitivos.

Pula/ignora linhas:

```
read_csv(
  "The first line of metadata
  The second line of metadata
  x,y,z
  1,2,3",
  skip = 2
)
```

```
Rows: 1 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): x, y, z

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# A tibble: 1 x 3
      x     y     z
  <dbl> <dbl> <dbl>
1     1     2     3
```

Expecifica o que é comentário:

```
read_csv(
  "# A comment I want to skip
  x,y,z
  1,2,3",
  comment = "#"
)
```

```
Rows: 1 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): x, y, z
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

# A tibble: 1 x 3
      x     y     z
  <dbl> <dbl> <dbl>
1     1     2     3
```

Diz que a base de dados não tem nomes de colunas:

```
read_csv(
  "1,2,3
  4,5,6",
  col_names = FALSE
)
```

```
Rows: 2 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): X1, X2, X3

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

# A tibble: 2 x 3
     X1    X2    X3
  <dbl> <dbl> <dbl>
1     1     2     3
2     4     5     6
```

Expecifica quem são as colunas:

```
read_csv(
  "1,2,3
  4,5,6",
  col_names = c("x", "y", "z")
)
```

```
Rows: 2 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (3): x, y, z
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# A tibble: 2 x 3
      x     y     z
  <dbl> <dbl> <dbl>
1     1     2     3
2     4     5     6
```

2.2 Outras funções de importação

As outras funções de importação seguem exatamente o modelo de read_csv(). Desse modo, basta saber seus nomes.

- read_csv2() lê arquivos separados por ";";
- read_tsv() lê arquivos delimitados por tabuação;
- read_delim() lê arquivos com qualquer delimitador;
- read_fwf() lê arquivos com larguras fixas. fwf_widths() e fwf_positions() são usados para especificar as laguras ou posições;
- read_table() lê arquivos com lagura fixa e separado por espaço em branco;
- read_log() lê arquivos log no estilo Apache.

2.3 Encontrando erros

O R usa a heurística para descobrir os tips de coluna. Observe o código seguinte e entenderá.

```
read_csv("
  logical,numeric,date,string
  TRUE,1,2021-01-15,abc
  false,4.5,2021-02-15,def
  T,Inf,2021-02-16,ghi
")
```

```
Rows: 3 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (1): string
dbl  (1): numeric
```

```
lgl  (1): logical
date (1): date

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

# A tibble: 3 x 4
  logical numeric date       string
  <lgl>     <dbl> <date>     <chr>
1 TRUE        1   2021-01-15 abc
2 FALSE       4.5 2021-02-15 def
3 TRUE      Inf   2021-02-16 ghi
```

Observe a seguinte base de dados que tem um errinho e que deveria ser numérica. Nós gostaríamos de descobrir exatamente onde ocorreu esse erro (finja ser uma base de dados gigantesca que tornasse inviável procurar visualmente onde ocorreu o erro).

```
simple_csv <- "
  x
  10
  .
  20
  30"
read_csv(simple_csv)
```

```
Rows: 4 Columns: 1
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): x

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

# A tibble: 4 x 1
  x
  <chr>
1 10
2 .
3 20
4 30
```

Uma forma de descobrir o erro é dizer ao R que essa coluna x deve ser numérica, o que vai gerar um erro por ser do tipo chr. Depois vamos peguntar onde está esse erro. Observe seguinte código.

```
df <- read_csv(
  simple_csv,
  col_types = list(x = col_double())
)
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
```

```
problems(df)
```

```
# A tibble: 1 x 5
    row   col expected actual file
  <int> <int> <chr>    <chr>  <chr>
1     3     1 a double .      C:/Users/Administrator/AppData/Local/Temp/RtmpKkS~
```

Com isso, encontramos que o erro está na linha 3 da primeira coluna.

Veja alguns "col's" que podem ser usados além de col_double(): col_logical(), col_integer(), col_character(), col_factor(), col_date(), col_number(), col_skip(). Além dessas, tem o argumento .default que muda o tipo de variável para a escolhida. O cols_only() permite especificar apenas as colunas que você quer ler.

2.4 Criando base de dados na mão

Também é possível criar uma base de dados com o tibble() e com o tribble(). A primeira cria a partir de vetores e a segunda cria de uma forma parecida com read_csv.

```
tibble(
  x = c(1, 2, 5),
  y = c("h", "m", "g"),
  z = c(0.08, 0.83, 0.60)
)
```

```
# A tibble: 3 x 3
      x y         z
  <dbl> <chr> <dbl>
1     1 h      0.08
2     2 m      0.83
3     5 g      0.6
```

```r
tribble(
  ~x, ~y, ~z,
  1, "h", 0.08,
  2, "m", 0.83,
  5, "g", 0.60
)
```

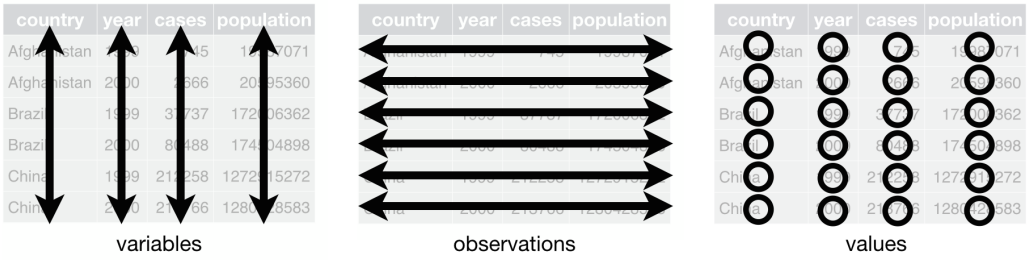
```
# A tibble: 3 x 3
      x y         z
  <dbl> <chr> <dbl>
1     1 h      0.08
2     2 m      0.83
3     5 g      0.6
```

3 Tidyr

O tidyr tem como objetivo a organizadoro de bancos de dados desorganizados.

Os dados organizados seguem um padrão:

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



Com isso em mente, o primeiro passo é instalar e carregar o pacote “tidyverse”.

```r
# install.packages('tidyverse')
library(tidyverse)
```

3.1 Tibble: billboard

Vamos usar algumas bases de dados presentes no próprio pacote "tidyverse" para contemplar alguns casos expecíficos de tratamento de dados. A primeira tem nome billboard.

```
glimpse(billboard)
```

```
Rows: 317
Columns: 79
$ artist       <chr> "2 Pac", "2Ge+her", "3 Doors Down", "3 Doors Down", "504 ~
$ track        <chr> "Baby Don't Cry (Keep...", "The Hardest Part Of ...", "Kr~
$ date.entered <date> 2000-02-26, 2000-09-02, 2000-04-08, 2000-10-21, 2000-04-~
$ wk1          <dbl> 87, 91, 81, 76, 57, 51, 97, 84, 59, 76, 84, 57, 50, 71, 7~
$ wk2          <dbl> 82, 87, 70, 76, 34, 39, 97, 62, 53, 76, 84, 47, 39, 51, 6~
$ wk3          <dbl> 72, 92, 68, 72, 25, 34, 96, 51, 38, 74, 75, 45, 30, 28, 5~
$ wk4          <dbl> 77, NA, 67, 69, 17, 26, 95, 41, 28, 69, 73, 29, 28, 18, 4~
$ wk5          <dbl> 87, NA, 66, 67, 17, 26, 100, 38, 21, 68, 73, 23, 21, 13, ~
$ wk6          <dbl> 94, NA, 57, 65, 31, 19, NA, 35, 18, 67, 69, 18, 19, 13, 3~
$ wk7          <dbl> 99, NA, 54, 55, 36, 2, NA, 35, 16, 61, 68, 11, 20, 11, 34~
$ wk8          <dbl> NA, NA, 53, 59, 49, 2, NA, 38, 14, 58, 65, 9, 17, 1, 29, ~
$ wk9          <dbl> NA, NA, 51, 62, 53, 3, NA, 38, 12, 57, 73, 9, 17, 1, 27, ~
$ wk10         <dbl> NA, NA, 51, 61, 57, 6, NA, 36, 10, 59, 83, 11, 17, 2, 30,~
$ wk11         <dbl> NA, NA, 51, 61, 64, 7, NA, 37, 9, 66, 92, 1, 17, 2, 36, N~
$ wk12         <dbl> NA, NA, 51, 59, 70, 22, NA, 37, 8, 68, NA, 1, 3, 3, 37, N~
$ wk13         <dbl> NA, NA, 47, 61, 75, 29, NA, 38, 6, 61, NA, 1, 3, 3, 39, N~
$ wk14         <dbl> NA, NA, 44, 66, 76, 36, NA, 49, 1, 67, NA, 1, 7, 4, 49, N~
$ wk15         <dbl> NA, NA, 38, 72, 78, 47, NA, 61, 2, 59, NA, 4, 10, 12, 57,~
$ wk16         <dbl> NA, NA, 28, 76, 85, 67, NA, 63, 2, 63, NA, 8, 17, 11, 63,~
$ wk17         <dbl> NA, NA, 22, 75, 92, 66, NA, 62, 2, 67, NA, 12, 25, 13, 65~
$ wk18         <dbl> NA, NA, 18, 67, 96, 84, NA, 67, 2, 71, NA, 22, 29, 15, 68~
$ wk19         <dbl> NA, NA, 18, 73, NA, 93, NA, 83, 3, 79, NA, 23, 29, 18, 79~
$ wk20         <dbl> NA, NA, 14, 70, NA, 94, NA, 86, 4, 89, NA, 43, 40, 20, 86~
$ wk21         <dbl> NA, NA, 12, NA, NA, NA, NA, NA, 5, NA, NA, 44, 43, 30, NA~
$ wk22         <dbl> NA, NA, 7, NA, NA, NA, NA, NA, 5, NA, NA, NA, 50, 40, NA,~
$ wk23         <dbl> NA, NA, 6, NA, NA, NA, NA, NA, 6, NA, NA, NA, NA, 39, NA,~
$ wk24         <dbl> NA, NA, 6, NA, NA, NA, NA, NA, 9, NA, NA, NA, NA, 44, NA,~
$ wk25         <dbl> NA, NA, 6, NA, NA, NA, NA, NA, 13, NA, NA, NA, NA, NA, NA~
$ wk26         <dbl> NA, NA, 5, NA, NA, NA, NA, NA, 14, NA, NA, NA, NA, NA, NA~
$ wk27         <dbl> NA, NA, 5, NA, NA, NA, NA, NA, 16, NA, NA, NA, NA, NA, NA~
$ wk28         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, 23, NA, NA, NA, NA, NA, NA~
$ wk29         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, 22, NA, NA, NA, NA, NA, NA~
$ wk30         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, 33, NA, NA, NA, NA, NA, NA~
$ wk31         <dbl> NA, NA, 4, NA, NA, NA, NA, NA, 36, NA, NA, NA, NA, NA, NA~
$ wk32         <dbl> NA, NA, 3, NA, NA, NA, NA, NA, 43, NA, NA, NA, NA, NA, NA~
```

```
$ wk33        <dbl> NA, NA, 3, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ wk34        <dbl> NA, NA, 3, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ wk35        <dbl> NA, NA, 4, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ wk36        <dbl> NA, NA, 5, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ wk37        <dbl> NA, NA, 5, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ wk38        <dbl> NA, NA, 9, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ wk39        <dbl> NA, NA, 9, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ wk40        <dbl> NA, NA, 15, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk41        <dbl> NA, NA, 14, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk42        <dbl> NA, NA, 13, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk43        <dbl> NA, NA, 14, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk44        <dbl> NA, NA, 16, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk45        <dbl> NA, NA, 17, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk46        <dbl> NA, NA, 21, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk47        <dbl> NA, NA, 22, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk48        <dbl> NA, NA, 24, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk49        <dbl> NA, NA, 28, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk50        <dbl> NA, NA, 33, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk51        <dbl> NA, NA, 42, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk52        <dbl> NA, NA, 42, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk53        <dbl> NA, NA, 49, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk54        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk55        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk56        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk57        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk58        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk59        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk60        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk61        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk62        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk63        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk64        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk65        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk66        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk67        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk68        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk69        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk70        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk71        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk72        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk73        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk74        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk75        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ wk76        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
```

Observe que todas essas colunas com 'wk' podem ser guardadas em uma única com nome week. Alem disso, os valores presentes em cada 'wk' podem ser guardados numa variavel chamada rank.

3.2 Valores de uma variável no nome das colunas

O comando 'pivot_longer' permite fazer operacões com variáveis que deveriam ser dados. O argumento 'cols =' seleciona as variáveis que serão armazenadas em uma única coluna. O 'names_to =' nomeia a coluna que vai receber as variáveis. O 'values_to =' nomeia a coluna que vai receber os dados que estavam em todas as antigas variáveis. O 'values_drop_na = TRUE' retira todos os valores com NA.

```
billboard |>
  pivot_longer(
    cols = starts_with('wk'),
    names_to = 'week',
    values_to = 'rank',
    values_drop_na = TRUE
  )
```

```
# A tibble: 5,307 x 5
   artist  track                   date.entered week   rank
   <chr>   <chr>                   <date>       <chr> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk1      87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk2      82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk3      72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk4      77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk5      87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk6      94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26   wk7      99
 8 2Ge+her The Hardest Part Of ... 2000-09-02   wk1      91
 9 2Ge+her The Hardest Part Of ... 2000-09-02   wk2      87
10 2Ge+her The Hardest Part Of ... 2000-09-02   wk3      92
# i 5,297 more rows
```

Para facilitar cálculos pode-se tranformar os 'wk' em números com a função do dplyr com nome 'mutate'. Alem disso, o parse_number extrai o primeiro número de uma string, o que é útil nesse caso.

```
library(readr)

billboard_longer <- billboard |>
  pivot_longer(
```

```
    cols = starts_with('wk'),
    names_to = 'week',
    values_to = 'rank',
    values_drop_na = TRUE
  ) |>
  mutate(
    week = parse_number(week)
)
billboard_longer
```

```
# A tibble: 5,307 x 5
   artist  track                   date.entered  week  rank
   <chr>   <chr>                   <date>       <dbl> <dbl>
 1 2 Pac   Baby Don't Cry (Keep... 2000-02-26       1    87
 2 2 Pac   Baby Don't Cry (Keep... 2000-02-26       2    82
 3 2 Pac   Baby Don't Cry (Keep... 2000-02-26       3    72
 4 2 Pac   Baby Don't Cry (Keep... 2000-02-26       4    77
 5 2 Pac   Baby Don't Cry (Keep... 2000-02-26       5    87
 6 2 Pac   Baby Don't Cry (Keep... 2000-02-26       6    94
 7 2 Pac   Baby Don't Cry (Keep... 2000-02-26       7    99
 8 2Ge+her The Hardest Part Of ... 2000-09-02       1    91
 9 2Ge+her The Hardest Part Of ... 2000-09-02       2    87
10 2Ge+her The Hardest Part Of ... 2000-09-02       3    92
# i 5,297 more rows
```

3.3 Tibble: who2

A segunda base de dados é o who2.

```
glimpse(who2)
```

```
Rows: 7,240
Columns: 58
$ country    <chr> "Afghanistan", "Afghanistan", "Afghanistan", "Afghanistan",~
$ year       <dbl> 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989,~
$ sp_m_014   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_m_1524  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_m_2534  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_m_3544  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_m_4554  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_m_5564  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_m_65    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_f_014   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
```

```
$ sp_f_1524  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_f_2534  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_f_3544  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_f_4554  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_f_5564  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sp_f_65    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_m_014   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_m_1524  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_m_2534  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_m_3544  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_m_4554  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_m_5564  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_m_65    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_f_014   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_f_1524  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_f_2534  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_f_3544  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_f_4554  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_f_5564  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sn_f_65    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_m_014   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_m_1524  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_m_2534  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_m_3544  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_m_4554  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_m_5564  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_m_65    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_f_014   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_f_1524  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_f_2534  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_f_3544  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_f_4554  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_f_5564  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ ep_f_65    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_m_014  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_m_1524 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_m_2534 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_m_3544 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_m_4554 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_m_5564 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_m_65   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_f_014  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_f_1524 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_f_2534 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_f_3544 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
```

```
$ rel_f_4554 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_f_5564 <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ rel_f_65   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
```

Veja que 'year' e 'country' já estão na forma que desejamos. Essa base de dados é sobre diagnósticos de tuberculose recolhidos pela OMS. Cada uma das outras colunas tem um padrão: três informacões separadas por '_'. A primeira parte descreve o método usado para o diagnóstico. A segunda descreve o gênero masculino ou feminino. E a terceira descreve intervalos de idade (por exemplo: 3544 significa de 35 a 44 anos).

3.4 Valores de mais de uma variável no nome das colunas

Para a base de dados who2 que concentra três informacões em diversas colunas, podemos utilizar '!' para identificar que não queremos alterar as colunas 'country' e 'year'. No 'names_to', será onde colocaremos quais serão as novas colunas que armazenarão as três variáveis nos nomes das antigas colunas. Para o R identificar qual é cada variável podemos dizer que as informacões estão separadas, por meio de 'names_sep', com o símbolo '_'. Por último, basta usarmos 'values_to' para criar a coluna que irá armazenar os valores de cada célula.

```r
who2 |>
  pivot_longer(
    cols = !(country:year),
    names_to = c("diagnosis", "gender", "age"),
    names_sep = "_",
    values_to = "count",
    values_drop_na = TRUE
  )
```

```
# A tibble: 76,046 x 6
   country      year diagnosis gender age   count
   <chr>       <dbl> <chr>     <chr>  <chr> <dbl>
 1 Afghanistan  1997 sp        m      014       0
 2 Afghanistan  1997 sp        m      1524     10
 3 Afghanistan  1997 sp        m      2534      6
 4 Afghanistan  1997 sp        m      3544      3
 5 Afghanistan  1997 sp        m      4554      5
 6 Afghanistan  1997 sp        m      5564      2
 7 Afghanistan  1997 sp        m      65        0
 8 Afghanistan  1997 sp        f      014       5
 9 Afghanistan  1997 sp        f      1524     38
10 Afghanistan  1997 sp        f      2534     36
# i 76,036 more rows
```

3.5 Tibble: household

A terceira base de dados é a household.

```
glimpse(household)
```

```
Rows: 5
Columns: 5
$ family      <int> 1, 2, 3, 4, 5
$ dob_child1  <date> 1998-11-26, 1996-06-22, 2002-07-11, 2004-10-10, 2000-12-05
$ dob_child2  <date> 2000-01-29, NA, 2004-04-05, 2009-08-27, 2005-02-28
$ name_child1 <chr> "Susan", "Mark", "Sam", "Craig", "Parker"
$ name_child2 <chr> "Jose", NA, "Seth", "Khai", "Gracie"
```

Observe que, dessa vez, as colunas tem duas variáveis. As primeiras são 'dob' e 'names', e a segunda é 'child'. A coluna 'family' já está organizada.

3.6 Caso especial para uso de .value

Para organizar a base de dados household podemos usar '!' para informar que não queremos modificar a coluna 'family'. Além disso, no 'names_to', colocamos o argumento ".value" para dizer que temos mais de uma informacão na posicão do inicio do nome das colunas, antes do separador. Após isso, como tem apenas 'child' no final do nome das colunas basta que coloquemos 'child'. Por fim, informamos qual o separador e, como têm valores faltantes, usamos o argumento 'values_drop_na = TRUE'.

```
household |>
  pivot_longer(
    cols = !family,
    names_to = c('.value', 'child'),
    names_sep = "_",
    values_drop_na = TRUE
  )
```

```
# A tibble: 9 x 4
  family child  dob        name
   <int> <chr>  <date>     <chr>
1      1 child1 1998-11-26 Susan
2      1 child2 2000-01-29 Jose
3      2 child1 1996-06-22 Mark
4      3 child1 2002-07-11 Sam
5      3 child2 2004-04-05 Seth
6      4 child1 2004-10-10 Craig
```

```
7      4 child2 2009-08-27 Khai
8      5 child1 2000-12-05 Parker
9      5 child2 2005-02-28 Gracie
```

3.7 Tibble: cms_patient_experience

A quarta base de dados é a cms_patient_experience.

```
glimpse(cms_patient_experience)
```

```
Rows: 500
Columns: 5
$ org_pac_id    <chr> "0446157747", "0446157747", "0446157747", "0446157747", ~
$ org_nm        <chr> "USC CARE MEDICAL GROUP INC", "USC CARE MEDICAL GROUP IN~
$ measure_cd    <chr> "CAHPS_GRP_1", "CAHPS_GRP_2", "CAHPS_GRP_3", "CAHPS_GRP_~
$ measure_title <chr> "CAHPS for MIPS SSM: Getting Timely Care, Appointments, ~
$ prf_rate      <dbl> 63, 87, 86, 57, 85, 24, 59, 85, 83, 63, 88, 22, 49, NA, ~
```

Essa é uma base de dados dos serviços dos Centros de Medicare e Medicaid que coleta dados sobre experiências dos pacientes.

Observe a coluna 'measure_cd', a qual tem o nome 'CAHPS_GRP_n', com n pertencente a {1 ,2 ,3 ,5 ,8 ,12}. Cada 'CAHPS_GRP_n' representa uma experiência diferente dos pacientes. Observe que o 'CAHPS_GRP_n' é repetido para cada organizacão na coluna 'org_nm', o que faz cada organizacão ser repetida 6 vezes. Isso nos inclina a querer transformar esses 'CAHPS_GRP_n' em 6 colunas e excluir a coluna 'measure_title' que tem a descrição de cada experiência (o que para fins de cálculo no R não fazem falta).

Com a função distinct() fica mais fácil de ver que cada 'CAHPS_GRP' representa uma experiência diferente dos pacientes.

```
cms_patient_experience |>
  distinct(measure_cd, measure_title)
```

```
# A tibble: 6 x 2
  measure_cd   measure_title
  <chr>        <chr>
1 CAHPS_GRP_1  CAHPS for MIPS SSM: Getting Timely Care, Appointments, and Infor~
2 CAHPS_GRP_2  CAHPS for MIPS SSM: How Well Providers Communicate
3 CAHPS_GRP_3  CAHPS for MIPS SSM: Patient's Rating of Provider
4 CAHPS_GRP_5  CAHPS for MIPS SSM: Health Promotion and Education
5 CAHPS_GRP_8  CAHPS for MIPS SSM: Courteous and Helpful Office Staff
6 CAHPS_GRP_12 CAHPS for MIPS SSM: Stewardship of Patient Resources
```

3.8 Variáveis nas linhas

Para organizar a base de dados cms_patient_experience podemos usar 'pivot_wider' que faz o contrário de 'pivot_longer'. Nós, primeiramente, colocamos no argumento 'id_cols' a string 'org' para o R tirar os valores repetidos dentro das colunas 'org_pac_id' e 'org_nm'. Após isso, colocamos no argumento 'names_from' para identificar de onde serão criadas as novas colunas. Por fim, usamos o argumento 'values_from' para identificar de onde será tirado as células das novas colunas.

```
cms_patient_experience |>
  pivot_wider(
    id_cols = starts_with("org"),
    names_from = measure_cd,
    values_from = prf_rate
  )
```

```
# A tibble: 95 x 8
   org_pac_id org_nm CAHPS_GRP_1 CAHPS_GRP_2 CAHPS_GRP_3 CAHPS_GRP_5 CAHPS_GRP_8
   <chr>      <chr>        <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
 1 0446157747 USC C~          63          87          86          57          85
 2 0446162697 ASSOC~          59          85          83          63          88
 3 0547164295 BEAVE~          49          NA          75          44          73
 4 0749333730 CAPE ~          67          84          85          65          82
 5 0840104360 ALLIA~          66          87          87          64          87
 6 0840109864 REX H~          73          87          84          67          91
 7 0840513552 SCL H~          58          83          76          58          78
 8 0941545784 GRITM~          46          86          81          54          NA
 9 1052612785 COMMU~          65          84          80          58          87
10 1254237779 OUR L~          61          NA          NA          65          NA
# i 85 more rows
# i 1 more variable: CAHPS_GRP_12 <dbl>
```

4 Dplyr

O dplyr permite a transformação de dados por meio de linhas, colunas ou grupos.

4.1 Tibble: flights

A base de dados que vamos usar para aplicar o dplyr é a flights, a qual já está organizada.

```r
# install.packages('nycflights13')
# install.packages('tidyverse')

library(tidyverse)
library(nycflights13)
glimpse(flights)
```

```
Rows: 336,776
Columns: 19
$ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
$ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
$ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
$ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
$ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
$ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
$ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
$ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
$ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
$ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
$ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
$ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
$ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
$ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
```

Observe os tipos de cada variável:

- int <- número inteiro;
- dbl <- número real;
- chr <- caracteres;
- dttm <- data e hora;

4.2 Linhas

4.2.1 filter()

Esse argumento filtra a base de dados de acordo com os valores nas linhas atreladas à uma determinada coluna. Por exemplo, o seguinte código pega todas as linhas cuja coluna

'dep_delay' apresenta valores maiores do que 120.

```
flights |>
  filter(dep_delay > 120)
```

```
# A tibble: 9,723 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      848           1835       853     1001           1950
 2  2013     1     1      957            733       144     1056            853
 3  2013     1     1     1114            900       134     1447           1222
 4  2013     1     1     1540           1338       122     2020           1825
 5  2013     1     1     1815           1325       290     2120           1542
 6  2013     1     1     1842           1422       260     1958           1535
 7  2013     1     1     1856           1645       131     2212           2005
 8  2013     1     1     1934           1725       129     2126           1855
 9  2013     1     1     1938           1703       155     2109           1823
10  2013     1     1     1942           1705       157     2124           1830
# i 9,713 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Lembrando os operadores lógicos:

- > maior; >= maior ou igual;
- < menor; <= menor ou igual;
- == igual; != diferente;
- '&' e; '|' ou;
- '&&' e; '||' ou; (retornando somente o primeiro resultado)

Outros exemplos (tente imaginar a saída do código):

```
flights |>
  filter(month == 1 & day == 1)
```

```
# A tibble: 842 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
```

```
 4  2013     1     1      544            545        -1     1004            1022
 5  2013     1     1      554            600        -6      812             837
 6  2013     1     1      554            558        -4      740             728
 7  2013     1     1      555            600        -5      913             854
 8  2013     1     1      557            600        -3      709             723
 9  2013     1     1      557            600        -3      838             846
10  2013     1     1      558            600        -2      753             745
# i 832 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights |>
  filter(month == 1 | month == 2)
```

```
# A tibble: 51,955 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 51,945 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Outra forma usando o %in% no código:

```
flights|>
  filter(month %in% c(1,2))
```

```
# A tibble: 51,955 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
```

```
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 51,945 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

perceba que o %in% fala: pegue os meses que são iguais a 1 ou iguais a 2. É uma forma equivalente ao último exemplo.

Observação: o filter apenas filtra a base de dados e imprime. Desse modo, nada que é da base de dados original será modificado. Caso queira guardar use o <- como no exemplo abaixo.

```
jan1 <- flights |>
  filter(month == 1 & day == 1)
```

4.2.2 arrange()

O argumento arrange ordena a base de dados, por padrão, do menor para o maior. Ele prioriza as primeiras variáveis que colocamos. O exemplo abaixo ordena os anos do menor para o maior, depois os meses são ordenados sem prejudicar a ordenação dos anos, depois os dias são ordenados sem prejudicar as ordenações anteriores, etc.

```
flights |>
  arrange(year, month, day, dep_time)
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
```

```
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

O argumento desc() muda o padrão da função arrange() para ordenar de forma decrescente, ou seja, do maior para o menor.

```
flights |>
  arrange(desc(dep_delay))
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     9      641            900      1301     1242           1530
 2  2013     6    15     1432           1935      1137     1607           2120
 3  2013     1    10     1121           1635      1126     1239           1810
 4  2013     9    20     1139           1845      1014     1457           2210
 5  2013     7    22      845           1600      1005     1044           1815
 6  2013     4    10     1100           1900       960     1342           2211
 7  2013     3    17     2321            810       911      135           1020
 8  2013     6    27      959           1900       899     1236           2226
 9  2013     7    22     2257            759       898      121           1026
10  2013    12     5      756           1700       896     1058           2020
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

4.2.3 distinct()

A função distinct() pega todas as primeiras ocorrências de cada linha única, ou seja ela oculta todas as linhas repetidas. Por exemplo, a função abaixo pega todas as linhas distintas de toda a base de dados. Perceba que, como todas as linhas da base de dados são distintas, nenhuma linha foi ocultada.

```
flights |>
  distinct()
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Perceba nesse outro exemplo que, dessa vez, ele pegou todas as primeiras linhas distintas presentes nas duas colunas representadas pelas variáveis 'origin' e 'dest'. De outro modo, ele ocultou todas as linhas repetidas nas variáveis 'origin' e 'dest'.

```
flights |>
  distinct(origin, dest)
```

```
# A tibble: 224 x 2
   origin dest
   <chr>  <chr>
 1 EWR    IAH
 2 LGA    IAH
 3 JFK    MIA
 4 JFK    BQN
 5 LGA    ATL
 6 EWR    ORD
 7 EWR    FLL
 8 LGA    IAD
 9 JFK    MCO
10 LGA    ORD
# i 214 more rows
```

Se quiser manter as outras colunas mas, ainda assim, ocultando as primeiras linhas repetidas de acordo com as duas variáveis é possível usar o argumento '.keep_all = TRUE'.

```r
flights |>
  distinct(origin, dest, .keep_all = TRUE)
```

```
# A tibble: 224 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 214 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

4.2.4 count()

Caso o seu objetivo seja apenas contar o número de ocorrências de cada variável pode-se utilizar a função count() com o argumento 'sort = TRUE' que permite ordenar da maior para a menor fequência.

```r
flights |>
  count(origin, dest, sort = TRUE)
```

```
# A tibble: 224 x 3
   origin dest      n
   <chr>  <chr> <int>
 1 JFK    LAX   11262
 2 LGA    ATL   10263
 3 LGA    ORD    8857
 4 JFK    SFO    8204
 5 LGA    CLT    6168
 6 EWR    ORD    6100
 7 JFK    BOS    5898
 8 LGA    MIA    5781
 9 JFK    MCO    5464
10 EWR    BOS    5327
# i 214 more rows
```

4.3 Colunas

4.3.1 mutate()

A função mutate() adiciona novas colunas na base de dados a partir das existentes. Por exemplo, pode-se adicionar a coluna 'gain' ao final da base de dados a partir da diferença de outras duas colunas existentes.

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed = distance / air_time * 60
  )
```

```
# A tibble: 336,776 x 21
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 13 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>, gain <dbl>, speed <dbl>
```

Para adicionar a nova coluna no inicio da base de dados (o que ajuda na hora de visualizar o que está acontecendo) pode-se utilizar o argumento '.before = 1'.

```
flights |>
  mutate(gain = dep_delay - arr_delay,
         speed = distance / air_time * 60,
         .before = 1
  )
```

```
# A tibble: 336,776 x 21
    gain speed  year month   day dep_time sched_dep_time dep_delay arr_time
```

```
   <dbl> <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1    -9  370.  2013     1     1      517            515         2      830
 2   -16  374.  2013     1     1      533            529         4      850
 3   -31  408.  2013     1     1      542            540         2      923
 4    17  517.  2013     1     1      544            545        -1     1004
 5    19  394.  2013     1     1      554            600        -6      812
 6   -16  288.  2013     1     1      554            558        -4      740
 7   -24  404.  2013     1     1      555            600        -5      913
 8    11  259.  2013     1     1      557            600        -3      709
 9     5  405.  2013     1     1      557            600        -3      838
10   -10  319.  2013     1     1      558            600        -2      753
# i 336,766 more rows
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Tambem é possivel adicionar a nova coluna depois de uma coluna já existente com o argumento '.after = variavel_existente'.

```
flights |>
  mutate(gain = dep_delay - arr_delay,
         speed = distance / air_time * 60,
         .after = dep_time
  )
```

```
# A tibble: 336,776 x 21
    year month   day dep_time  gain speed sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int> <dbl> <dbl>          <int>     <dbl>    <int>
 1  2013     1     1      517    -9  370.            515         2      830
 2  2013     1     1      533   -16  374.            529         4      850
 3  2013     1     1      542   -31  408.            540         2      923
 4  2013     1     1      544    17  517.            545        -1     1004
 5  2013     1     1      554    19  394.            600        -6      812
 6  2013     1     1      554   -16  288.            558        -4      740
 7  2013     1     1      555   -24  404.            600        -5      913
 8  2013     1     1      557    11  259.            600        -3      709
 9  2013     1     1      557     5  405.            600        -3      838
10  2013     1     1      558   -10  319.            600        -2      753
# i 336,766 more rows
# i 12 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

O argumento '.keep = "used" ' mantém na base de dados apenas as variáveis existentes que foram envolvidas em alguma operação do mutate() e às variáveis que foram criadas.

```
flights |>
  mutate(gain = dep_delay - arr_delay,
         speed = distance / air_time * 60,
         gain_per_hour = gain/hour,
         .keep = "used"
  )
```

```
# A tibble: 336,776 x 8
   dep_delay arr_delay air_time distance  hour  gain speed gain_per_hour
       <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl> <dbl>         <dbl>
 1         2        11      227     1400     5    -9  370.         -1.8
 2         4        20      227     1416     5   -16  374.         -3.2
 3         2        33      160     1089     5   -31  408.         -6.2
 4        -1       -18      183     1576     5    17  517.          3.4
 5        -6       -25      116      762     6    19  394.          3.17
 6        -4        12      150      719     5   -16  288.         -3.2
 7        -5        19      158     1065     6   -24  404.         -4
 8        -3       -14       53      229     6    11  259.          1.83
 9        -3        -8      140      944     6     5  405.          0.833
10        -2         8      138      733     6   -10  319.         -1.67
# i 336,766 more rows
```

4.3.2 select()

A função select() permite selecionar determinadas colunas de várias formas.

Exemplos:

1. Selecionar colunas por nome:

```
flights |>
  select(year, month, day)
```

```
# A tibble: 336,776 x 3
    year month   day
   <int> <int> <int>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
```

```
10  2013     1     1
# i 336,766 more rows
```

2. Selecionar colunas entre (inicio:final):

```
flights |>
  select(year:day)
```

```
# A tibble: 336,776 x 3
    year month   day
   <int> <int> <int>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
10  2013     1     1
# i 336,766 more rows
```

3. Selecionar todas as colunas diferentes das (!inicio:final):

```
flights |>
  select(!year:month)
```

```
# A tibble: 336,776 x 17
     day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1     1      517            515         2      830            819        11
 2     1      533            529         4      850            830        20
 3     1      542            540         2      923            850        33
 4     1      544            545        -1     1004           1022       -18
 5     1      554            600        -6      812            837       -25
 6     1      554            558        -4      740            728        12
 7     1      555            600        -5      913            854        19
 8     1      557            600        -3      709            723       -14
 9     1      557            600        -3      838            846        -8
10     1      558            600        -2      753            745         8
# i 336,766 more rows
# i 10 more variables: carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

```
flights |>
  select(! year, month, day)
```

```
# A tibble: 336,776 x 18
   month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1     1     1      517            515         2      830            819
 2     1     1      533            529         4      850            830
 3     1     1      542            540         2      923            850
 4     1     1      544            545        -1     1004           1022
 5     1     1      554            600        -6      812            837
 6     1     1      554            558        -4      740            728
 7     1     1      555            600        -5      913            854
 8     1     1      557            600        -3      709            723
 9     1     1      557            600        -3      838            846
10     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

4. Selecionar todas as colunas que são caracteres

```
flights |>
  select(where(is.character))
```

```
# A tibble: 336,776 x 4
   carrier tailnum origin dest
   <chr>   <chr>   <chr>  <chr>
 1 UA      N14228  EWR    IAH
 2 UA      N24211  LGA    IAH
 3 AA      N619AA  JFK    MIA
 4 B6      N804JB  JFK    BQN
 5 DL      N668DN  LGA    ATL
 6 UA      N39463  EWR    ORD
 7 B6      N516JB  EWR    FLL
 8 EV      N829AS  LGA    IAD
 9 B6      N593JB  JFK    MCO
10 AA      N3ALAA  LGA    ORD
# i 336,766 more rows
```

5. Selecionar todas as colunas que começam com 'd'

```
flights |>
  select(starts_with('d'))
```

```
# A tibble: 336,776 x 5
     day dep_time dep_delay dest  distance
   <int>    <int>     <dbl> <chr>    <dbl>
 1     1      517         2 IAH       1400
 2     1      533         4 IAH       1416
 3     1      542         2 MIA       1089
 4     1      544        -1 BQN       1576
 5     1      554        -6 ATL        762
 6     1      554        -4 ORD        719
 7     1      555        -5 FLL       1065
 8     1      557        -3 IAD        229
 9     1      557        -3 MCO        944
10     1      558        -2 ORD        733
# i 336,766 more rows
```

6. Selecionar todas as colunas que terminam com 'e'

```
flights |>
  select(ends_with('e'))
```

```
# A tibble: 336,776 x 7
   dep_time sched_dep_time arr_time sched_arr_time air_time distance minute
      <int>          <int>    <int>          <int>    <dbl>    <dbl>  <dbl>
 1      517            515      830            819      227     1400     15
 2      533            529      850            830      227     1416     29
 3      542            540      923            850      160     1089     40
 4      544            545     1004           1022      183     1576     45
 5      554            600      812            837      116      762      0
 6      554            558      740            728      150      719     58
 7      555            600      913            854      158     1065      0
 8      557            600      709            723       53      229      0
 9      557            600      838            846      140      944      0
10      558            600      753            745      138      733      0
# i 336,766 more rows
```

7. Selecionar todas as colunas que contém 'o'

```
flights |>
  select(contains('o'))
```

```
# A tibble: 336,776 x 4
   month origin  hour time_hour
   <int> <chr>  <dbl> <dttm>
 1     1 EWR        5 2013-01-01 05:00:00
 2     1 LGA        5 2013-01-01 05:00:00
 3     1 JFK        5 2013-01-01 05:00:00
 4     1 JFK        5 2013-01-01 05:00:00
```

```
 5     1 LGA        6 2013-01-01 06:00:00
 6     1 EWR        5 2013-01-01 05:00:00
 7     1 EWR        6 2013-01-01 06:00:00
 8     1 LGA        6 2013-01-01 06:00:00
 9     1 JFK        6 2013-01-01 06:00:00
10     1 LGA        6 2013-01-01 06:00:00
# i 336,766 more rows
```

8. Selecionar todas as colunas que tem x1, x2 e x3 (não vai retornar nada por que não tem nenhuma coluna que tem as características solicitadas).

```
flights |>
  select(num_range('x', 1:3))
```

```
# A tibble: 336,776 x 0
```

9. Selecionar determinada coluna e renomeá-la

```
flights |>
  select(lalala = dep_time)
```

```
# A tibble: 336,776 x 1
   lalala
    <int>
 1    517
 2    533
 3    542
 4    544
 5    554
 6    554
 7    555
 8    557
 9    557
10    558
# i 336,766 more rows
```

4.3.3 rename()

O rename permite renomear colunas mantendo todas as existentes.

```
flights |>
  rename(dia = day)
```

```
# A tibble: 336,776 x 19
    year month   dia dep_time sched_dep_time dep_delay arr_time sched_arr_time
```

```
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

4.3.4 relocate()

O relocate desloca as variáveis selecionadas para o inicio da base de dados.

```
flights |>
  relocate(dest, hour)
```

```
# A tibble: 336,776 x 19
   dest   hour  year month   day dep_time sched_dep_time dep_delay arr_time
   <chr> <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1 IAH       5  2013     1     1      517            515         2      830
 2 IAH       5  2013     1     1      533            529         4      850
 3 MIA       5  2013     1     1      542            540         2      923
 4 BQN       5  2013     1     1      544            545        -1     1004
 5 ATL       6  2013     1     1      554            600        -6      812
 6 ORD       5  2013     1     1      554            558        -4      740
 7 FLL       6  2013     1     1      555            600        -5      913
 8 IAD       6  2013     1     1      557            600        -3      709
 9 MCO       6  2013     1     1      557            600        -3      838
10 ORD       6  2013     1     1      558            600        -2      753
# i 336,766 more rows
# i 10 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, air_time <dbl>, distance <dbl>,
#   minute <dbl>, time_hour <dttm>
```

```
glimpse(flights)
```

```
Rows: 336,776
Columns: 19
$ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
$ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
$ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
$ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
$ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
$ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
$ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
$ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
$ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
$ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
$ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
$ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
$ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
$ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
```

Também é possivel especificar onde as colunas serão colocadas com os argumentos '.after' para depois de algo e '.before' para antes de algo.

```
flights |>
  relocate(distance:minute, .before = day)
```

```
# A tibble: 336,776 x 19
    year month distance  hour minute   day dep_time sched_dep_time dep_delay
   <int> <int>    <dbl> <dbl>  <dbl> <int>    <int>          <int>     <dbl>
 1  2013     1     1400     5     15     1      517            515         2
 2  2013     1     1416     5     29     1      533            529         4
 3  2013     1     1089     5     40     1      542            540         2
 4  2013     1     1576     5     45     1      544            545        -1
 5  2013     1      762     6      0     1      554            600        -6
 6  2013     1      719     5     58     1      554            558        -4
 7  2013     1     1065     6      0     1      555            600        -5
 8  2013     1      229     6      0     1      557            600        -3
 9  2013     1      944     6      0     1      557            600        -3
10  2013     1      733     6      0     1      558            600        -2
# i 336,766 more rows
# i 10 more variables: arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, time_hour <dttm>
```

```
flights |>
  relocate(distance:minute, .after = day)
```

```
# A tibble: 336,776 x 19
    year month   day distance  hour minute dep_time sched_dep_time dep_delay
   <int> <int> <int>    <dbl> <dbl>  <dbl>    <int>          <int>     <dbl>
 1  2013     1     1     1400     5     15      517            515         2
 2  2013     1     1     1416     5     29      533            529         4
 3  2013     1     1     1089     5     40      542            540         2
 4  2013     1     1     1576     5     45      544            545        -1
 5  2013     1     1      762     6      0      554            600        -6
 6  2013     1     1      719     5     58      554            558        -4
 7  2013     1     1     1065     6      0      555            600        -5
 8  2013     1     1      229     6      0      557            600        -3
 9  2013     1     1      944     6      0      557            600        -3
10  2013     1     1      733     6      0      558            600        -2
# i 336,766 more rows
# i 10 more variables: arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, time_hour <dttm>
```

4.4 O verdadeiro poder do pipe

O pipe é o operador que pode ser representado por '|>' ou por '%>%'. Para utilizar o '|>' é necessario fazer alguns ajustes no Rstudio. Eu recomendo o '|>' por ser mais limpo e por fazer parte dos comandos base do R, não precisando carregar o tidyverse. Ajuste no Rstudio: tools -> Global Options -> Code –> Use native operator. O comando para escrever o pipe: (ctrl + shift + M).

Como exemplo, podemos encontrar voos rápidos para o aerporto IAH de Houston.

```
flights |>
  filter(dest == 'IAH') |>
  mutate(speed = distance / air_time * 60) |>
  select(year:day, dep_time, carrier, flight, speed) |>
  arrange(desc(speed))
```

```
# A tibble: 7,198 x 7
    year month   day dep_time carrier flight speed
   <int> <int> <int>    <int> <chr>    <int> <dbl>
 1  2013     7     9      707 UA         226  522.
 2  2013     8    27     1850 UA        1128  521.
 3  2013     8    28      902 UA        1711  519.
```

```
 4  2013     8    28     2122 UA        1022  519.
 5  2013     6    11     1628 UA        1178  515.
 6  2013     8    27     1017 UA         333  515.
 7  2013     8    27     1205 UA        1421  515.
 8  2013     8    27     1758 UA         302  515.
 9  2013     9    27      521 UA         252  515.
10  2013     8    28      625 UA         559  515.
# i 7,188 more rows
```

4.5 Grupos

4.5.1 group_by()

A função group_by() permite realizar operações sobre grupos presentes em uma ou mais variáveis de uma base de dados.

Observe o exemplo abaixo que agrupa por mês, mas como não foi realizado nenhuma operação então o código retornou apenas a base de dados original.

```
flights |>
  group_by(month)
```

```
# A tibble: 336,776 x 19
# Groups:   month [12]
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

4.5.2 summarize()

Essa é uma das operações agrupadas mais importantes a qual permite criar um resumo.

```
flights |>
  group_by(month) |>
  summarize(delay = mean(dep_delay))
```

```
# A tibble: 12 x 2
   month delay
   <int> <dbl>
 1     1    NA
 2     2    NA
 3     3    NA
 4     4    NA
 5     5    NA
 6     6    NA
 7     7    NA
 8     8    NA
 9     9    NA
10    10    NA
11    11    NA
12    12    NA
```

Observe que, por ter linhas com dados faltantes, o código retornou NA. Para contornar isso pode-se usar 'na.rm = TRUE' na função mean().

```
flights |>
  group_by(month) |>
  summarize(delay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 12 x 2
   month delay
   <int> <dbl>
 1     1 10.0
 2     2 10.8
 3     3 13.2
 4     4 13.9
 5     5 13.0
 6     6 20.8
 7     7 21.7
 8     8 12.6
 9     9  6.72
```

```
10    10  6.24
11    11  5.44
12    12 16.6
```

Um resumo útil é o n() que conta o número de linhas de cada grupo.

```
flights |>
  group_by(month) |>
  summarize(delay = mean(dep_delay, na.rm = TRUE),
            n = n())
```

```
# A tibble: 12 x 3
   month delay     n
   <int> <dbl> <int>
 1     1 10.0  27004
 2     2 10.8  24951
 3     3 13.2  28834
 4     4 13.9  28330
 5     5 13.0  28796
 6     6 20.8  28243
 7     7 21.7  29425
 8     8 12.6  29327
 9     9  6.72 27574
10    10  6.24 28889
11    11  5.44 27268
12    12 16.6  28135
```

É possÍvel agrupar mais de uma variável, porém cada resumo se destaca do último grupo.

```
diario <- flights |>
  group_by(year, month, day)
print(diario)
```

```
# A tibble: 336,776 x 19
# Groups:   year, month, day [365]
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
```

```
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Observe que o dplyr exibe uma mensagem informando como você pode alterar esse comportamento.

```
voos_diarios <- diario |>
  summarise(n = n())
```

```
`summarise()` has grouped output by 'year', 'month'. You can override using the
`.groups` argument.
```

Você pode suprimir a mensagem, caso esteja satisfeito, com '.groups = "drop_last"'. Se quiser eliminar todos os agrupamentos coloque '.groups = "drop"' e se quiser preservar os agrupamentos coloque '.groups = "keep"'.

```
voos_diarios <- diario |>
  summarise(n = n(),
            .groups = 'drop_last')
```

4.5.3 ungroup()

A função ungroup() remove o agrupamento de uma base de dados sem utilizar o summarise.

```
diario |>
  ungroup() |>
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    flights = n()
  )
```

```
# A tibble: 1 x 2
  delay flights
  <dbl>   <int>
1  12.6  336776
```

Observe que o resultado foi em uma única linha. Isso ocorre pois o dplyr trata todas as linhas de uma base de dados desagrupada como um único grupo.

4.5.4 .by

Caso nao queira usar o group_by() existe o argumento .by que simplifica a escrita.

```
flights |>
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n(),
    .by = month
  )
```

```
# A tibble: 12 x 3
   month delay     n
   <int> <dbl> <int>
 1     1 10.0  27004
 2    10  6.24 28889
 3    11  5.44 27268
 4    12 16.6  28135
 5     2 10.8  24951
 6     3 13.2  28834
 7     4 13.9  28330
 8     5 13.0  28796
 9     6 20.8  28243
10     7 21.7  29425
11     8 12.6  29327
12     9  6.72 27574
```

```
flights |>
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n(),
    .by = c(origin, dest)
  )
```

```
# A tibble: 224 x 4
   origin dest  delay     n
   <chr>  <chr> <dbl> <int>
 1 EWR    IAH   11.8   3973
 2 LGA    IAH    9.06  2951
 3 JFK    MIA    9.34  3314
 4 JFK    BQN    6.67   599
 5 LGA    ATL   11.4  10263
 6 EWR    ORD   14.6   6100
 7 EWR    FLL   13.5   3793
```

```
 8 LGA    IAD   16.7   1803
 9 JFK    MCO   10.6   5464
10 LGA    ORD   10.7   8857
# i 214 more rows
```

4.6 Slice_Functions

As funções slice permitem pegar linhas específicas de uma base de dados.

- slice_head() pega as n primeiras linhas.

```
flights |>
  slice_head(n = 1)
```

```
# A tibble: 1 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- slice_tail() pega as n últimas linhas.

```
flights |>
  slice_tail(n = 5)
```

```
# A tibble: 5 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     9    30       NA           1455        NA       NA           1634
2  2013     9    30       NA           2200        NA       NA           2312
3  2013     9    30       NA           1210        NA       NA           1330
4  2013     9    30       NA           1159        NA       NA           1344
5  2013     9    30       NA            840        NA       NA           1020
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- slice_min() pega as n menores linhas de uma coluna.

```
flights |>
  slice_min(flight,n = 1)
```

```
# A tibble: 701 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      856            900        -4     1226           1220
 2  2013     1     1     1153           1123        30     1454           1425
 3  2013     1     2      855            900        -5     1225           1220
 4  2013     1     2     1134           1123        11     1449           1425
 5  2013     1     3      855            900        -5     1144           1220
 6  2013     1     3     1201           1123        38     1510           1425
 7  2013     1     4      858            900        -2     1210           1220
 8  2013     1     4     1130           1123         7     1427           1425
 9  2013     1     4     2030           2030         0     2313           2338
10  2013     1     5      851            900        -9     1206           1220
# i 691 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- slice_max() pega as n maiores linhas de uma coluna.

```
flights |>
  slice_max(flight,n = 1)
```

```
# A tibble: 1 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1    30     1222           1115        67     1402           1215
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- slice_sample() pega n linhas aleatórias.

```
flights |>
  slice_sample(n = 1)
```

```
# A tibble: 1 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     2     3     1444           1420        24     1725           1713
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Ao invés de usar 'n =' pode-se utilizar 'prop ='.

Observe o exemplo que pega os últimos 40% do total de linhas.

```
flights |>
  slice_tail(prop = 0.4)
```

```
# A tibble: 134,710 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     5    10      817            820        -3      922            930
 2  2013     5    10      818            820        -2     1047           1110
 3  2013     5    10      819            819         0     1200           1210
 4  2013     5    10      820            829        -9     1008           1034
 5  2013     5    10      823            825        -2     1021           1023
 6  2013     5    10      823            829        -6      953           1024
 7  2013     5    10      823            825        -2     1015           1026
 8  2013     5    10      824            836       -12      930            947
 9  2013     5    10      824            830        -6      957           1015
10  2013     5    10      826            834        -8      944           1004
# i 134,700 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Observe o exemplo que pega o voo mais atrasado de cada destino.

```
flights |>
  group_by(dest) |>
  slice_max(arr_delay, n = 1) |>
  relocate(dest)
```

```
# A tibble: 108 x 19
# Groups:   dest [105]
   dest   year month   day dep_time sched_dep_time dep_delay arr_time
   <chr> <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1 ABQ    2013     7    22     2145           2007        98      132
 2 ACK    2013     7    23     1139            800       219     1250
 3 ALB    2013     1    25      123           2000       323      229
 4 ANC    2013     8    17     1740           1625        75     2042
 5 ATL    2013     7    22     2257            759       898      121
 6 AUS    2013     7    10     2056           1505       351     2347
 7 AVL    2013     8    13     1156            832       204     1417
 8 BDL    2013     2    21     1728           1316       252     1839
```

```
 9 BGR    2013    12     1     1504           1056       248     1628
10 BHM    2013     4    10       25           1900       325      136
# i 98 more rows
# i 11 more variables: sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Perceba que temos 108 linhas como resultado embora encontram-se apenas 105 destinos. Isso ocorre por que são pegos todas as linhas com o valor mais alto. Caso se queira apenas uma linha por grupo use o argumento 'with_ties = FALSE'.

5 Ggplot2

O ggplot2 permite criar gráficos. Ele tem um padrão de código o qual você perceberá que é parecido com uma idéia de ir adicionando camadas.

5.1 Tibble: penguins

Para carregar a base de dados penguins é necessário carregar o pacote 'palmerpenguins'.

```
# install.packages('palmerpenguins')
library(palmerpenguins)
```

Com isso, basta observarmos a base de dados, a qual já está organizada, para podermos aplicar o ggplot2.

```
# View(penguins)
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex               <fct> male, female, female, NA, female, male, female, male~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

Como você já deve ter notado, a função glimpse() permite observar todas as variáveis e as primeiras observações. Já a função View() permite observar a base de dados inteira, permitindo uma melhor visualização.

O pacote 'ggthemes' tem uma paleta de cores segura para daltônicos, o que recomendamos fortemente para uso nos gráficos. Faremos uso desse recurso mais a frente quando apresentarmos bem a função ggplot().

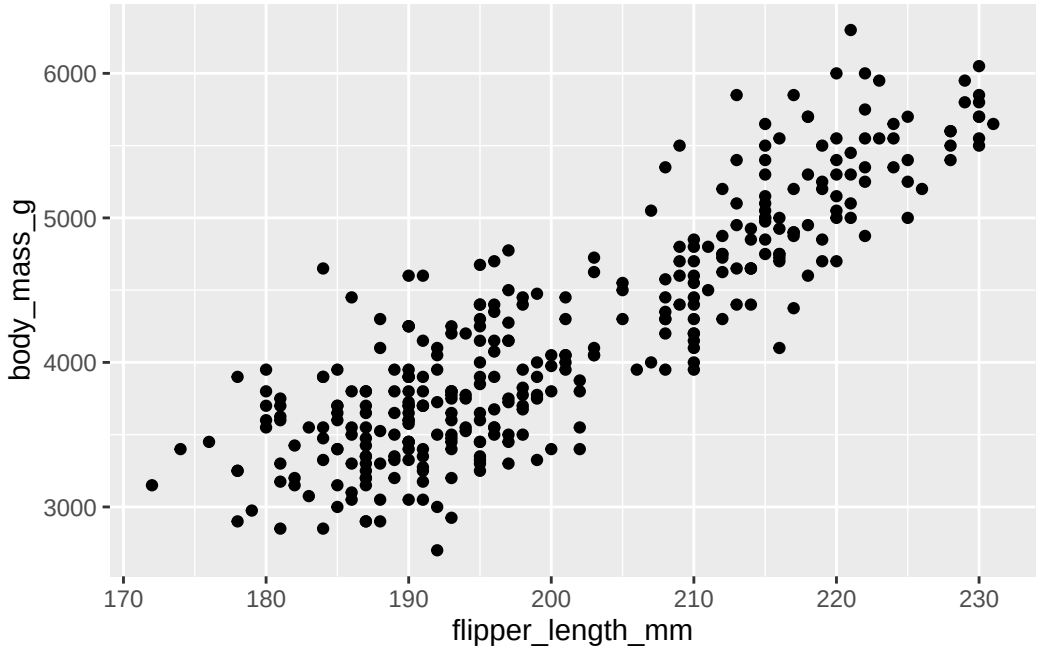
```
library(ggthemes)
```

5.2 Gráfico de Pontos

Vamos começar fazendo um diagrama de disperção com as variáveis flipper_length_mm e body_mass_g.

```
penguins |>
  ggplot(mapping = aes(x = flipper_length_mm, y = body_mass_g))+
  geom_point()
```

```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```

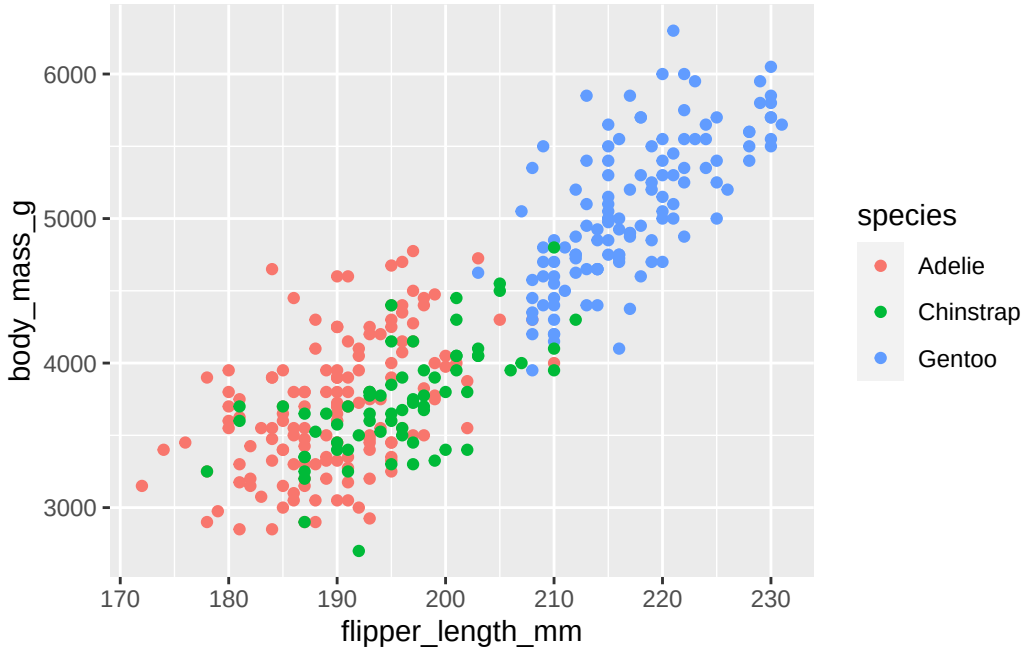


O 'mapping' define como as variáveis são mapeadas para propriedades visuais. O 'mapping' é sempre definido na função 'aes' (lembra de aesthetics). O símbolo '+' serve para adicionar

outras coisas no gráfico (por isso os gráficos do ggplot2 seguem uma ideia de adicionar camadas ao gráfico). Os 'geom' são objetos que o gráfico usa para representar os dados. Geralmente são eles que definem o tipo de gráfico (Ex.: Gráfico de pontos). Além disso, Observe a mensagem: Removed 2 rows containing missing values ('geom_point()'). Isso significa que há 2 observações de 'flipper_length' ou de 'body_mass' faltantes. No caso, essas duas observações foram removidas, e a mensagem é um alerta.

```
penguins |>
  ggplot(mapping = aes(x = flipper_length_mm,
                       y = body_mass_g, color = species))+
  geom_point()
```

```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```

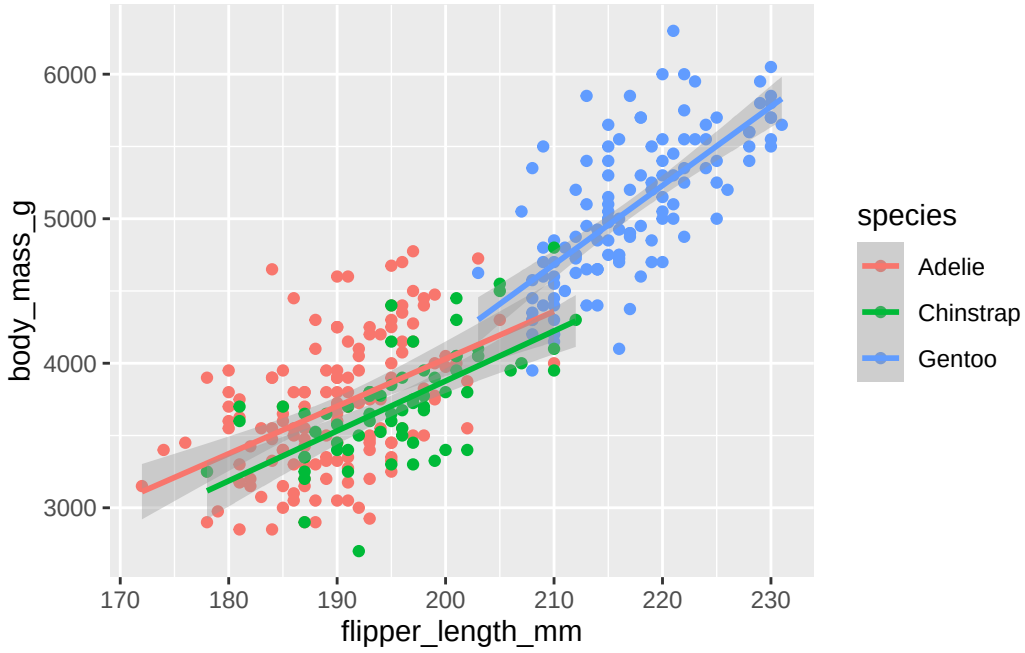


Ao adicionar 'color' na variável 'species' é dado um detalhamento. Uma cor para cada categoria da variável 'species'. Também é feito uma legenda automática com as cores usadas em cada categoria.

Observe a diferença no código dos dois próximos gráficos:

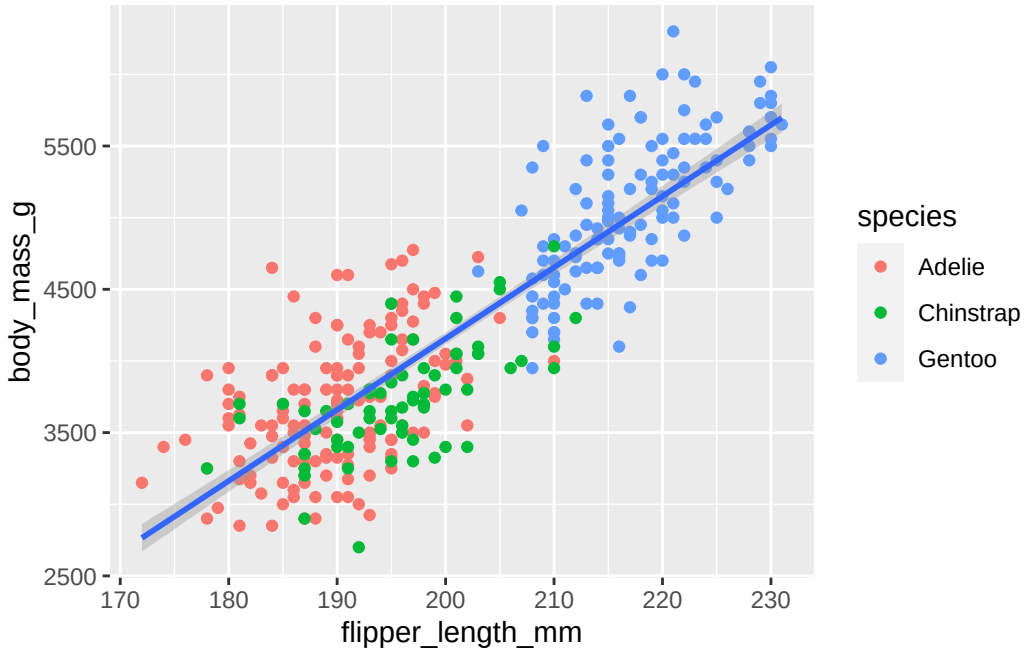
```
penguins |>
  ggplot(mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)) +
       geom_point() +
       geom_smooth(method = 'lm')
```

```
`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).

Warning: Removed 2 rows containing missing values (`geom_point()`).
```




```
ggplot(data = penguins,
       mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point(mapping = aes(color = species)) +
  geom_smooth(method = 'lm')
```

```
`geom_smooth()` using formula = 'y ~ x'

Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
Removed 2 rows containing missing values (`geom_point()`).
```



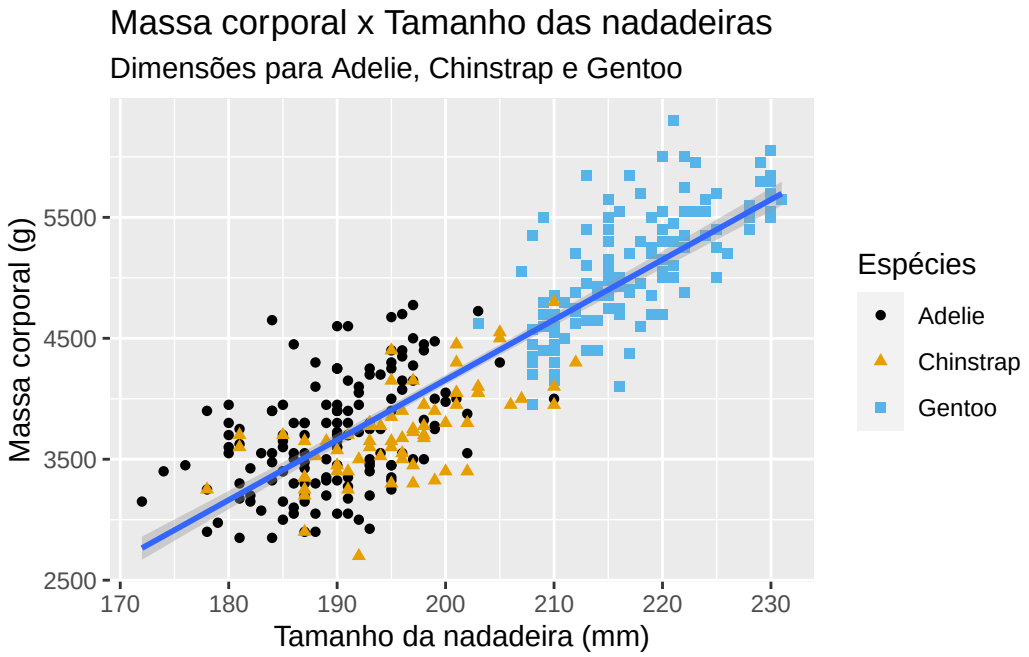
O 'geom_smooth' adiciona retas ao gráfico. O 'lm' indica que quer traçar retas de melhor ajuste com base em um modelo interno. Os mapeamentos estéticos definidos em ggplot() são de nível global, ou seja, são transmitidos para cada uma das camadas 'geom'. Porém, o mapping também pode ser colocado nos 'geom' e, desta vez, eles são em nível local.

```
penguins |>
  ggplot(mapping = aes(x = flipper_length_mm,
                       y = body_mass_g)) +
  geom_point(mapping = aes(color = species, shape = species)) +
  geom_smooth(method = 'lm') +
  labs(title = 'Massa corporal x Tamanho das nadadeiras',
       subtitle = 'Dimensões para Adelie, Chinstrap e Gentoo',
       x = 'Tamanho da nadadeira (mm)', y = 'Massa corporal (g)',
       color = 'Espécies', shape = 'Espécies') +
  scale_color_colorblind()
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 2 rows containing non-finite values (`stat_smooth()`).
```

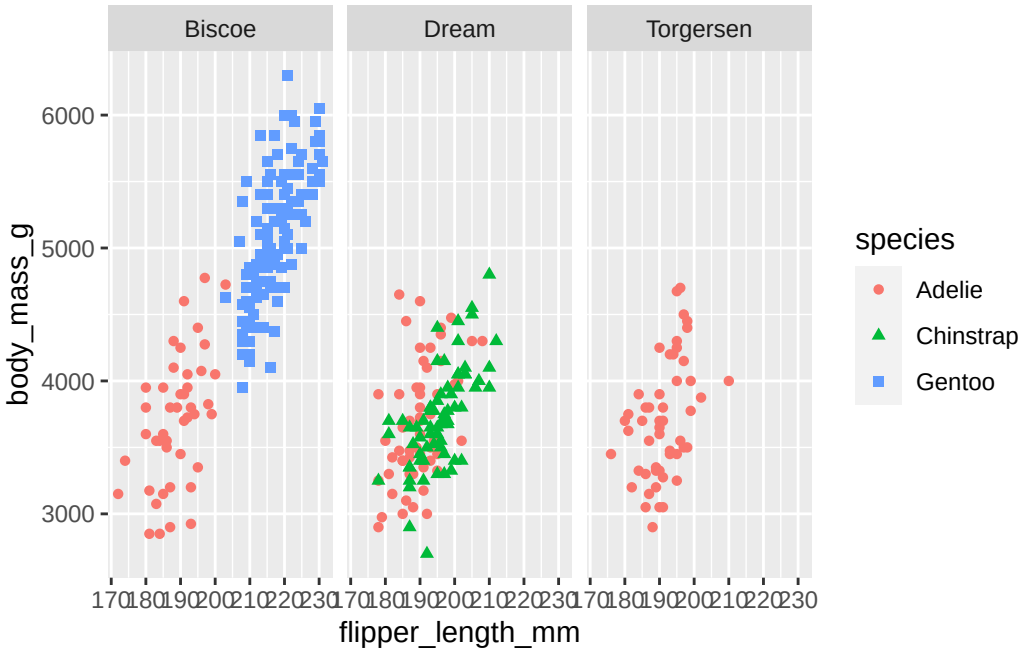
```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



O labs() adiciona legendas ao gráfico. Título: 'title'; Subtítulo: 'subtitle'; Legenda do eixo x: 'x'; Legenda do eixo y: 'y'; Legenda da legenda de cores/shape: 'color' e 'shape'. Se tirar uma dessas duas legendas (color, shape) aí fica uma automática a mais desnecessária, ou se tirar as duas fica apenas uma automática. O ultimo argumento coloca cores seguras para daltônicos.

```
penguins |>
  ggplot(aes(x = flipper_length_mm, y = body_mass_g))+
  geom_point(aes(color = species, shape = species))+
  facet_wrap(~island)
```

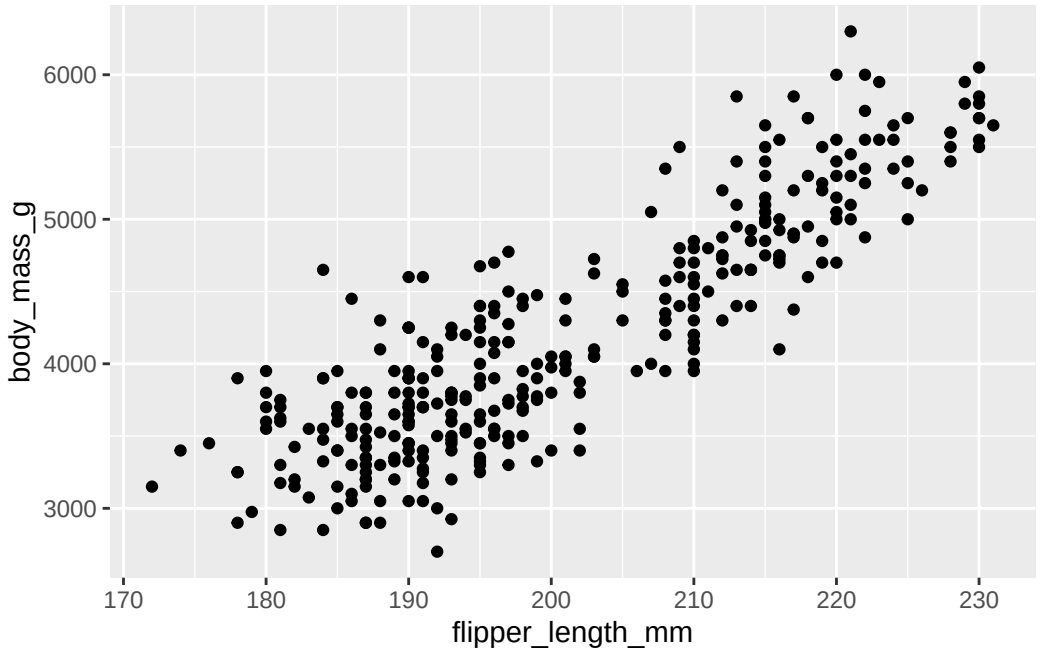
```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



O argumento 'facet_wrap' faz vários gráficos em uma janela. Ele faceta de acordo com a variável depois de '~'(categórica).

```
penguins |>
  ggplot(aes(x = flipper_length_mm, y = body_mass_g))+
  geom_point() # +
```

```
Warning: Removed 2 rows containing missing values (`geom_point()`).
```



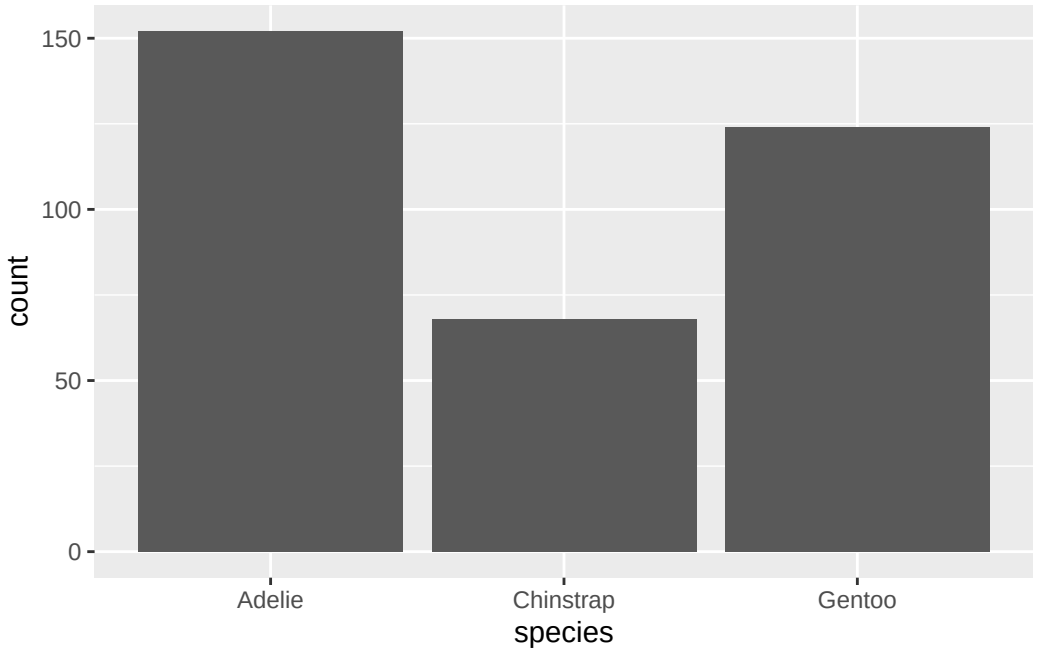

```
  #ggsave(filename = 'grafico_pontos_penguins.png')
```

O ggsave salva o gráfico no seu diretório de trabalho.

5.3 Gráfico de Barras

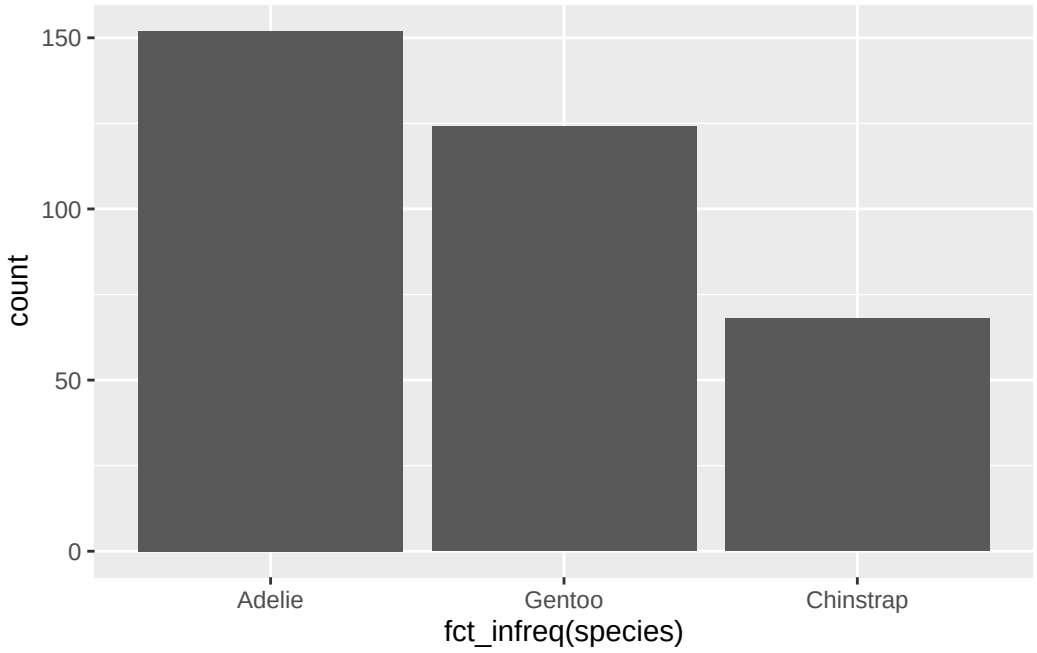
Para fazer um gráfico de barras basta mudarmos o 'geom'.

```
penguins |> ggplot(aes(x = species)) +
  geom_bar()
```



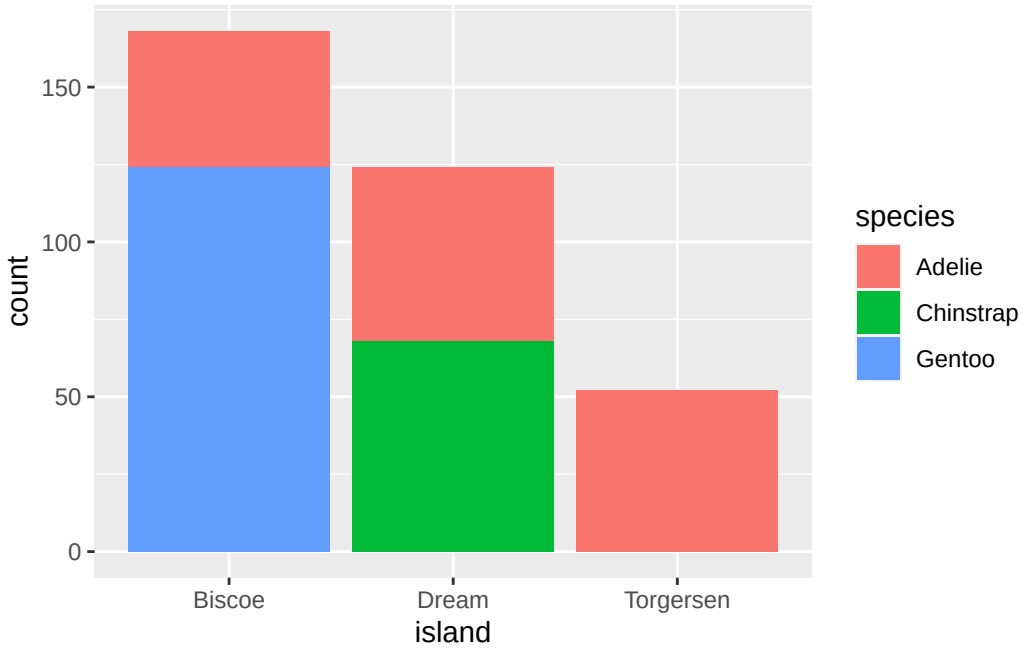
O geom bar adiciona barras ao gráfico. A altura representa a frequência.

```
ggplot(penguins, aes(x = fct_infreq(species))) +
  geom_bar()
```



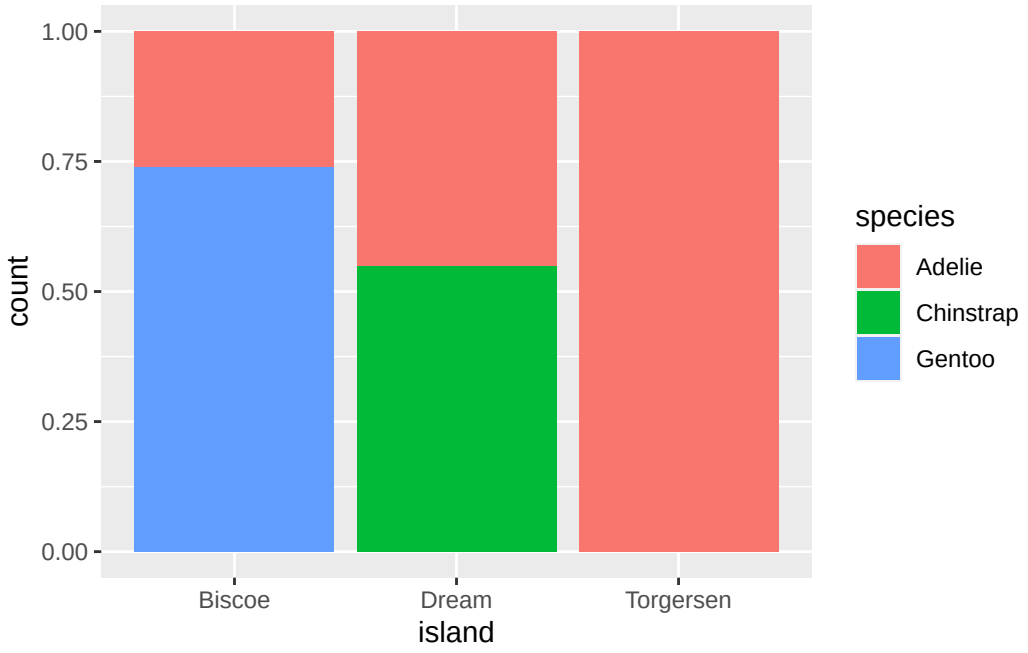
fct_infreq() transforma a variável 'species' em fator, dando maior nível para maior frequência.

```
penguins |>
  ggplot(aes(x = island, fill = species))+
  geom_bar()
```



O gráfico de barras contou a variável 'species' em cada local da variável 'island'. Perceba que embora Torgersen tenha apenas pinguins Adelie, ele ficou com uma menor altura devido a diferenta de quantidade.

```
penguins |>
  ggplot(aes(x = island, fill = species))+
  geom_bar(position = 'fill')
```



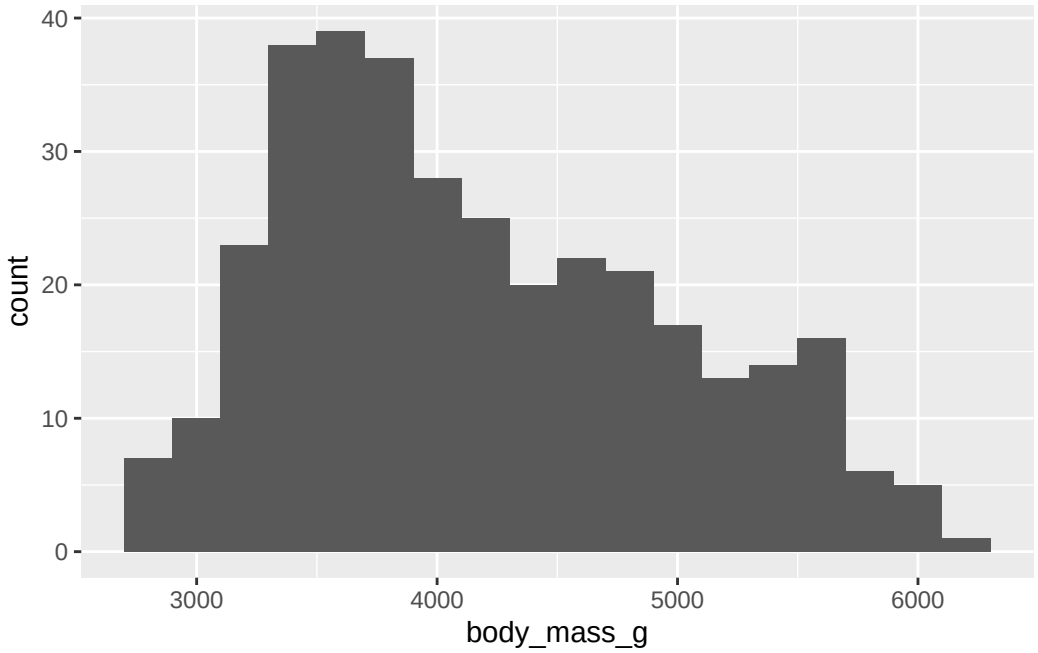
O argumento 'position' transformou o gráfico de barras com frequências absolutas em um gráfico com frequências relativas. Desse modo, Torgesen tem 100% de pinguins Adelie.

5.4 Histograma

Igual os anteriores, basta mudar o 'geom' para mudar o tipo de gráfico para histograma.

```
penguins |>
  ggplot(aes(x = body_mass_g)) +
  geom_histogram(binwidth = 200)
```

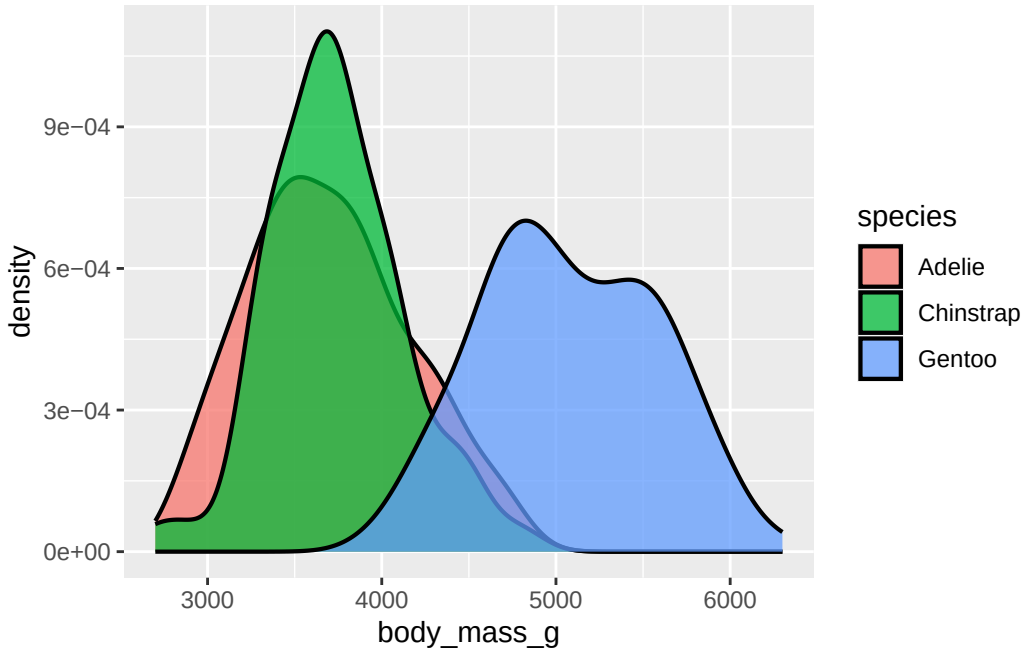
```
Warning: Removed 2 rows containing non-finite values (`stat_bin()`).
```



O geom_histogram() faz um histograma no gráfico. O argumento 'binwidth' define a largura das barras de acordo com o eixo x.

```
penguins |>
  ggplot(aes(x = body_mass_g, fill = species)) +
  geom_density(linewidth = 0.75, alpha = 0.75)
```

```
Warning: Removed 2 rows containing non-finite values (`stat_density()`).
```



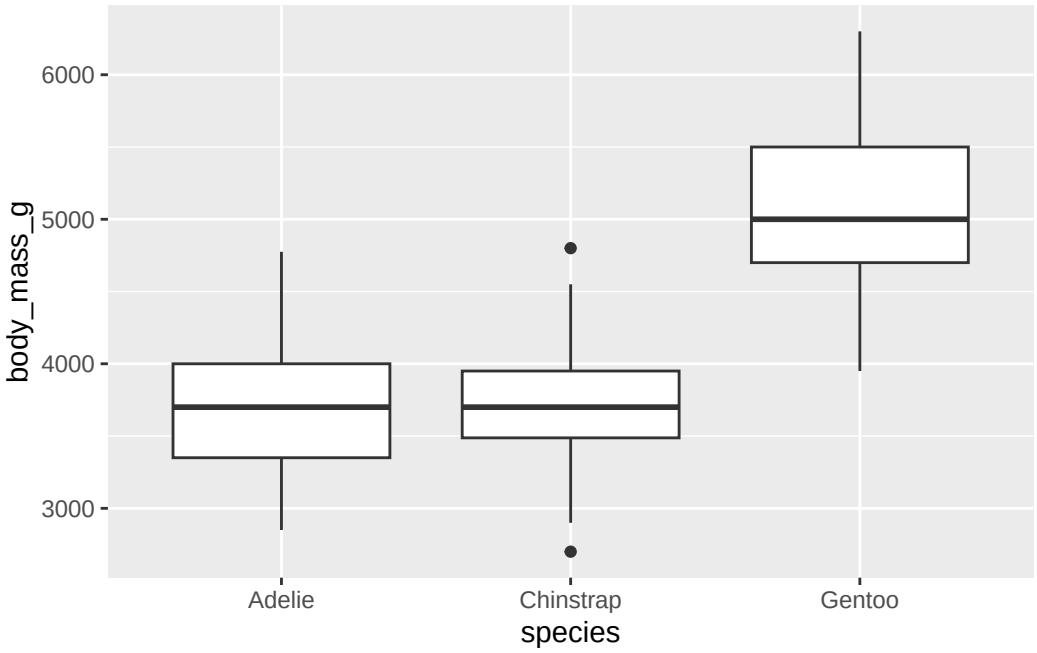
geom_density() faz uma curva de densidade. É uma versão de histograma sem barras e com uma cursa que representa a densidade. O argumento 'linewidth' define a espessura da linha. O argumento 'fill' preenche o gráfico com cores de acordo com a variável 'species'. O argumento 'alpha' deixa o preenchimento do fill transparente.

5.5 Boxplot

Para fazer um boxplot seguimos a mesma ideia (no geral esse é o padrão). Caso você deseje fazer qualquer gráfico diferente pense em mudar o 'geom'.

```
penguins |>
  ggplot(aes(x = species, y = body_mass_g))+
  geom_boxplot()
```

```
Warning: Removed 2 rows containing non-finite values (`stat_boxplot()`).
```



5.6 Esquisse

Esse pacote poderosíssimo e bem divertido permite criar gráficos manualmente de forma muito mais simples e bem bonita.

Para instalar e carregar o pacote realize o seguinte código.

```
# install.packages('esquisse')
library(esquisse)
```

Após isso basta se deliciar com o seguinte comando.

```
# esquisser(# Coloque aqui sua base de dados.)
```

O comando esquisser() é bem intuitivo e, sinseramente, é mais fácil você aprender testando! Use alguma das bases de dados organizadas que foram apresentadas anteriormente.

5.7 Gráficos Interativos

Para finalizar, vamos apresentar um pacote que contém o comando que permite transformar seus gráficos do ggplot2 em gráficos interativos. Ele também é bem simples e, infelizmente, não conseguimos mostrar como fica um gráfico interativo por meio de PDF. Logo, esperamos que você aplique no seu computador.

```r
# install.packages('plotly')
library(plotly)
```

```
Attaching package: 'plotly'

The following object is masked from 'package:ggplot2':

    last_plot

The following object is masked from 'package:stats':

    filter

The following object is masked from 'package:graphics':

    layout
```

```r
# ggplotly(
#   Coloque aqui um gráfico que você fez no ggplot().
#   Recomendamos guardar seu gráfico em uma variável
# )
```

[1]

Referências

[1] Hadley Wickham, Mine Çetinkaya-Rundel e Garrett Grolemund. *R for data science*. " O'Reilly Media, Inc.", 2023.

[1] Wickham, Çetinkaya-Rundel e Grolemund [1]