

Oficina de Tidyverse

Luben Miguel

Universidade Federal de São Carlos

10/06/2022

O que é o Tidyverse?

- O `{tidyverse}` é um conjunto de pacotes para ciência de dados.
- Fazem parte do `{tidyverse}`: `{ggplot2}`, `{dplyr}`, `{tidyverse}`, `{purrr}`, `{readr}`, e assim vai...



- Pacotes com mesma sintaxe e estilo de programar, usados em conjunto
- Seguem o *manifesto tidy* e trabalham em torno de bases *tidy*

Manifesto tidy

1. Reutilizar estruturas de dados existentes
2. Organizar funções simples usando o pipe (%>%)
3. Aderir à programação funcional
4. Projetado para ser usado por seres humanos.

Base tidy

- Base onde é possível trabalhar com dados de uma maneira simples e eficiente
 - Cada coluna representa uma variável
 - Cada linha uma observação
 - Cada cédula um único valor

Resumão do que alguns pacotes fazem



{readr}: Ler dados em formato tabular



{purrr}: Programação funcional



{tidyr}: Arrumar e padronizar dados



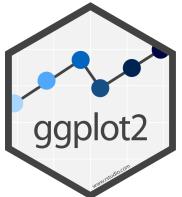
{stringr}: Lidar com strings



{dplyr}: transformar dados



{forcats}: Lidar com fatores



{ggplot2}: criar gráficos



{tibble}: Data frames repaginados

Pacotes preliminares

Tibbles

- Tibbles são praticamente data.frames com melhorias na visualização pelo console
- São usadas em diversas funções e pacotes do **tidyverse**, principalmente pelos pacotes **dplyr** e **tidyverse**
- Data-frame padrão
 - Tibble

```
mtcars[, 1:6]
```

```
as_tibble(mtcars[, 1:6])
```

```
##          mpg cyl  disp  hp drat    wt## # A tibble: 32 × 6## Mazda RX4      21.0   6 160.0 110 3.90 2.620##          mpg     cyl     disp      hp     drat      wt## Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875##      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>## Datsun 710     22.8   4 108.0  93 3.85 2.320##      1     21      6    160     110     3.9    2.62## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215##      2     21      6    160     110     3.9    2.88## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440##      3     22.8     4    108     93     3.85    2.32## Valiant        18.1   6 225.0 105 2.76 3.460##      4     21.4     6    258     110     3.08    3.22## Duster 360      14.3   8 360.0 245 3.21 3.570##      5     18.7     8    360     175     3.15    3.44## Merc 240D       24.4   4 146.7  62 3.69 3.190##      6     18.1     6    225     105     2.76    3.46## Merc 230        22.8   4 140.8  95 3.92 3.150##      7     14.3     8    360     245     3.21    3.57## Merc 280        19.2   6 167.6 123 3.92 3.440##      8     24.4     4    147.     62     3.69    3.19## Merc 280C       17.8   6 167.6 123 3.92 3.440##      9     22.8     4    141.     95     3.92    3.15## Merc 450SE      16.4   8 275.8 180 3.07 4.070##      10    19.2     6    168.    123     3.92    3.44## Merc 450SL      17.3   8 275.8 180 3.07 3.730## # ... with 22 more rows
```

Operador Pipe

- Torna a escrita e leitura de códigos mais intuitiva
- Funções do tidyverse foram projetadas para serem usadas com o pipe `%>%`
- Ideia do pipe: usar o valor resultante da expressão do lado esquerdo como primeiro argumento da função do lado direito.

```
# expressões equivalentes
f(x, y)
x %>% f(y)
```

Exemplinhos

- Vamos calcular a raiz quadrada da soma de um vetor $x = c(1, 2, 3, 4)$:

```
# sem pipe
x <- c(1, 2, 3, 4)
sqrt(sum(x))
```

```
## [1] 3.162278
```

```
# com pipe
x %>% sum() %>% sqrt()
```

```
## [1] 3.162278
```

Exemplinhos

- Ajustando um modelo linear no banco de dados **airquality**:

```
# resultado do lado esquerdo indo para outro argumento do lado direito que não o primeiro
airquality %>%
  na.omit() %>%
  lm(Ozone ~ Wind + Temp + Solar.R, data = .)
```

```
##
## Call:
## lm(formula = Ozone ~ Wind + Temp + Solar.R, data = .)
##
## Coefficients:
## (Intercept)      Wind       Temp      Solar.R
## -64.34208     -3.33359     1.65209     0.05982
```

Dplyr e tidyverse

Dplyr

- Pacote muito útil, fácil e versatil de se usar para transformação de dados
- Diversas funções que podem ser combinadas e encadeadas pelo pipe `%>%`
- Principais funções:
 - `select()`: seleciona colunas
 - `arrange()`: ordena a base de dados
 - `filter()`: filtra linhas
 - `mutate()`: cria/modifica colunas
 - `group_by()`: agrupa a base de dados de acordo com certa(s) variável(eis)
 - `summarise()`: sumariza a base de dados
- Características dessas funções:
 - O input o output são sempre uma tibble.
 - Tibble vai no primeiro argumento das funções e o que queremos fazer nos outros argumentos.
 - A utilização é facilitada com o emprego do operador `%>%`.

Dados

Vamos usar o banco de dados iris para ilustrar as funções do **dplyr**:

```
# transformando antes para tibble
iris <- as_tibble(iris) %>% rowid_to_column()
iris

## # A tibble: 150 × 6
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <int>      <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1     1         5.1         3.5         1.4         0.2 setosa
## 2     2         4.9         3.0         1.4         0.2 setosa
## 3     3         4.7         3.2         1.3         0.2 setosa
## 4     4         4.6         3.1         1.5         0.2 setosa
## 5     5         5.0         3.6         1.4         0.2 setosa
## 6     6         5.4         3.9         1.7         0.4 setosa
## 7     7         4.6         3.4         1.4         0.3 setosa
## 8     8         5.0         3.4         1.5         0.2 setosa
## 9     9         4.4         2.9         1.4         0.2 setosa
## 10   10        4.9         3.1         1.5         0.1 setosa
## # ... with 140 more rows
```

Selecionando colunas

- Usando a função `select()`
- Primeiro argumento é a base e os demais são as variáveis a serem selecionadas

Selecionando uma coluna:

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

```
## # A tibble: 150 × 1
##   Sepal.Length
##       <dbl>
## 1     5.1
## 2     4.9
## 3     4.7
## 4     4.6
## 5     5
## 6     5.4
## 7     4.6
## 8     5
## 9     4.4
## 10    4.9
## # ... with 140 more rows
```

Selecionando mais de uma:

```
# iris %>% select(1,2)
iris %>% select(Sepal.Length, Sepal.Width)
```

```
## # A tibble: 150 × 2
##   Sepal.Length Sepal.Width
##       <dbl>      <dbl>
## 1     5.1      3.5
## 2     4.9      3.0
## 3     4.7      3.2
## 4     4.6      3.1
## 5     5.0      3.6
## 6     5.4      3.9
## 7     4.6      3.4
## 8     5.0      3.4
## 9     4.4      2.9
## 10    4.9      3.1
## # ... with 140 more rows
```

Selecionando colunas

Selecionando um intervalo de variaveis

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

## # A tibble: 150 × 3
##   Sepal.Length Sepal.Width Petal.Length
##       <dbl>       <dbl>       <dbl>
## 1       5.1       3.5       1.4
## 2       4.9       3.0       1.4
## 3       4.7       3.2       1.3
## 4       4.6       3.1       1.5
## 5       5.0       3.6       1.4
## 6       5.4       3.9       1.7
## 7       4.6       3.4       1.4
## 8       5.0       3.4       1.5
## 9       4.4       2.9       1.4
## 10      4.9       3.1       1.5
## # ... with 140 more rows
```

Selecionando colunas

Também podemos remover variáveis:

```
iris %>% select(-Sepal.Length)

## # A tibble: 150 × 5
##   rowid Sepal.Width Petal.Length Petal.Width Species
##   <int>     <dbl>      <dbl>      <dbl> <fct>
## 1     1         3.5        1.4       0.2  setosa
## 2     2         3.0        1.4       0.2  setosa
## 3     3         3.2        1.3       0.2  setosa
## 4     4         3.1        1.5       0.2  setosa
## 5     5         3.6        1.4       0.2  setosa
## 6     6         3.9        1.7       0.4  setosa
## 7     7         3.4        1.4       0.3  setosa
## 8     8         3.4        1.5       0.2  setosa
## 9     9         2.9        1.4       0.2  setosa
## 10    10        3.1        1.5       0.1  setosa
## # ... with 140 more rows
```

Selecionando colunas

Algumas funções suplementares que ajudam na seleção de variáveis:

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

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

```
## # A tibble: 150 × 2
##       Petal.Length Petal.Width
##             <dbl>       <dbl>
## 1             1.4         0.2
## 2             1.4         0.2
## 3             1.3         0.2
## 4             1.5         0.2
## 5             1.4         0.2
## 6             1.7         0.4
## 7             1.4         0.3
## 8             1.5         0.2
```

Ordenando a base

- Com `arrange()` ordenamos as linhas de acordo com alguma variável de interesse:
 - Variável categórica: ordena por ordem alfabética
 - Variável numérica: ordena do menor para o maior
 - Varíavel fator: ordena pelos níveis do fator

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

## # A tibble: 150 × 6
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <int>      <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1     1        4.3        3.0       1.1       0.1 setosa
## 2     2        4.4        3.4       1.4       0.2 setosa
## 3     3        4.4        3.0       1.3       0.2 setosa
## 4     4        4.4        3.2       1.3       0.2 setosa
## 5     5        4.5        2.3       1.3       0.3 setosa
## 6     6        4.6        3.1       1.5       0.2 setosa
## 7     7        4.6        3.4       1.4       0.3 setosa
## 8     8        4.6        3.6       1.0       0.2 setosa
## 9     9        4.6        3.2       1.4       0.2 setosa
## 10   10       4.7        3.2       1.3       0.2 setosa
```

Ordenando a base

Ordenando de forma decrescente

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

## # A tibble: 150 × 6
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <int>      <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1     1       7.9       3.8       6.4       2.0 virginica
## 2     2       7.7       3.8       6.7       2.2 virginica
## 3     3       7.7       2.6       6.9       2.3 virginica
## 4     4       7.7       2.8       6.7       2.0 virginica
## 5     5       7.7       3.0       6.1       2.3 virginica
## 6     6       7.6       3.0       6.6       2.1 virginica
## 7     7       7.4       2.8       6.1       1.9 virginica
## 8     8       7.3       2.9       6.3       1.8 virginica
## 9     9       7.2       3.6       6.1       2.5 virginica
## 10   10      7.2       3.2       6.0       1.8 virginica
## # ... with 140 more rows
```

Ordenando a base

Ordenando por mais de uma variável

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

## # A tibble: 150 × 6
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <int>      <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1     1        4.3        3         1.1        0.1 setosa
## 2     9        4.4        2.9         1.4        0.2 setosa
## 3    39        4.4        3          1.3        0.2 setosa
## 4    43        4.4        3.2         1.3        0.2 setosa
## 5    42        4.5        2.3         1.3        0.3 setosa
## 6     4        4.6        3.1         1.5        0.2 setosa
## 7     7        4.6        3.4         1.4        0.3 setosa
## 8    23        4.6        3.6          1        0.2 setosa
## 9    48        4.6        3.2         1.4        0.2 setosa
## 10    3        4.7        3.2         1.3        0.2 setosa
## # ... with 140 more rows
```

Filtrando linhas

- Usamos `filter()` para selecionar as linhas de acordo com certa condição (filtro)

```
iris %>% filter(Sepal.Length > 6)

## # A tibble: 61 × 6
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <int>      <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1     1         7         3.2        4.7       1.4 versicolor
## 2     2        6.4        3.2        4.5       1.5 versicolor
## 3     3        6.9        3.1        4.9       1.5 versicolor
## 4     4        6.5        2.8        4.6       1.5 versicolor
## 5     5        6.3        3.3        4.7       1.6 versicolor
## 6     6        6.6        2.9        4.6       1.3 versicolor
## 7     7        6.1        2.9        4.7       1.4 versicolor
## 8     8        6.7        3.1        4.4       1.4 versicolor
## 9     9        6.2        2.2        4.5       1.5 versicolor
## 10   10       7.2        6.1        2.8        4       1.3 versicolor
## # ... with 51 more rows
```

Filtrando linhas

- Combinando filter, select e arrange para analisar os individuos que tem tamanho de sépala maior que 6:

```
iris %>% filter(Sepal.Length > 6) %>%  
  select(rowid, Sepal.Length) %>%  
  arrange(desc(Sepal.Length))
```

```
## # A tibble: 61 × 2  
##   rowid  Sepal.Length  
##   <int>     <dbl>  
## 1 132      7.9  
## 2 118      7.7  
## 3 119      7.7  
## 4 123      7.7  
## 5 136      7.7  
## 6 106      7.6  
## 7 131      7.4  
## 8 108      7.3  
## 9 110      7.2  
## 10 126     7.2  
## # ... with 51 more rows
```

Filtrando linhas

- Filtrando por mais de uma condição

```
iris %>% filter(Sepal.Length > 6, Species == "versicolor")
```

```
## # A tibble: 20 × 6
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <int>      <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1     1         7         3.2        4.7       1.4 versicolor
## 2     2        6.4        3.2        4.5       1.5 versicolor
## 3     3        6.9        3.1        4.9       1.5 versicolor
## 4     4        6.5        2.8        4.6       1.5 versicolor
## 5     5        6.3        3.3        4.7       1.6 versicolor
## 6     6        6.6        2.9        4.6       1.3 versicolor
## 7     7        6.1        2.9        4.7       1.4 versicolor
## 8     8        6.7        3.1        4.4       1.4 versicolor
## 9     9        6.2        2.2        4.5       1.5 versicolor
## 10   10       7.2        6.1        2.8        4       1.3 versicolor
## 11   11       7.3        6.3        2.5        4.9       1.5 versicolor
## 12   12       7.4        6.1        2.8        4.7       1.2 versicolor
## 13   13       7.5        6.4        2.9        4.3       1.3 versicolor
## 14   14       7.6        6.6        3         4.4       1.4 versicolor
## 15   15       7.7        6.8        2.8        4.8       1.4 versicolor
## 16   16       7.8        6.7        3         5        1.7 versicolor
```

Filtrando linhas

- Filtrando por mais de uma categoria

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

## # A tibble: 100 × 6
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <int>      <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1     1          5.1        3.5        1.4        0.2 setosa
## 2     2          4.9        3          1.4        0.2 setosa
## 3     3          4.7        3.2        1.3        0.2 setosa
## 4     4          4.6        3.1        1.5        0.2 setosa
## 5     5          5          3.6        1.4        0.2 setosa
## 6     6          5.4        3.9        1.7        0.4 setosa
## 7     7          4.6        3.4        1.4        0.3 setosa
## 8     8          5          3.4        1.5        0.2 setosa
## 9     9          4.4        2.9        1.4        0.2 setosa
## 10   10          4.9        3.1        1.5        0.1 setosa
## # ... with 90 more rows
```

Modificando colunas

- Função `mutate()` para criar e modificar colunas
- Primeiro modificando:

```
iris %>%  
  mutate(  
    Sepal.Length = Sepal.Length*10,  
    Petal.Length = Petal.Length*10  
)  
  
## # A tibble: 150 × 6  
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
##   <int>      <dbl>      <dbl>       <dbl>      <dbl> <fct>  
## 1     1          51        3.5         14        0.2 setosa  
## 2     2          49        3           14        0.2 setosa  
## 3     3          47        3.2         13        0.2 setosa  
## 4     4          46        3.1         15        0.2 setosa  
## 5     5          50        3.6         14        0.2 setosa  
## 6     6          54        3.9         17        0.4 setosa  
## 7     7          46        3.4         14        0.3 setosa  
## 8     8          50        3.4         15        0.2 setosa  
## 9     9          44        2.9         14        0.2 setosa  
## 10   10          49        3.1         15        0.1 setosa
```

Modificando colunas

- Criando colunas:

```
iris %>%  
  mutate(  
    Sepal.Length_mm = Sepal.Length*10,  
    Petal.Length_mm = Petal.Length*10  
)  
  
## # A tibble: 150 × 8  
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
##   <int>      <dbl>      <dbl>       <dbl>      <dbl> <fct>  
## 1     1         5.1        3.5        1.4       0.2  setosa  
## 2     2         4.9        3.0        1.4       0.2  setosa  
## 3     3         4.7        3.2        1.3       0.2  setosa  
## 4     4         4.6        3.1        1.5       0.2  setosa  
## 5     5         5.0        3.6        1.4       0.2  setosa  
## 6     6         5.4        3.9        1.7       0.4  setosa  
## 7     7         4.6        3.4        1.4       0.3  setosa  
## 8     8         5.0        3.4        1.5       0.2  setosa  
## 9     9         4.4        2.9        1.4       0.2  setosa  
## 10   10        4.9        3.1        1.5       0.1  setosa  
## # ... with 140 more rows, and 2 more variables: Sepal.Length_mm <dbl>,  
## #   Petal.Length_mm <dbl>
```

Sumarizando a base

- Pela função `summarise()` podemos resumir nossa base para alguma métrica de interesse
- Média, mediana, desvio padrão, etc.
- Podemos fazer isso para um ou mais colunas de interesse

```
# media e desvio padrao do Sepal.Length
iris %>%
  summarise(
    media_sepal_lenght = mean(Sepal.Length),
    sd_sepal_lenght = sd(Sepal.Length),
    media_sepal_width = mean(Sepal.Width),
    sd_sepal_width = sd(Sepal.Width),
    n = n()
  )

## # A tibble: 1 × 5
##   media_sepal_lenght sd_sepal_lenght media_sepal_width sd_sepal_width     n
##   <dbl>              <dbl>              <dbl>              <dbl> <int>
## 1      5.84             0.828             3.06             0.436    150
```

Sumarizando a base

- Podemos juntar o `summarise()` com o `group_by()` para tratar cada grupo como se fosse uma base de dados diferente.
- Isso também é válido para o `mutate()`.

```
iris %>%  
  group_by(Species) %>%  
  summarise(  
    media_sepal_lenght = mean(Sepal.Length),  
    sd_sepal_lenght = sd(Sepal.Length),  
    media_sepal_width = mean(Sepal.Width),  
    sd_sepal_width = sd(Sepal.Width),  
    n = n()  
)  
  
## # A tibble: 3 × 6  
##   Species  media_sepal_lenght  sd_sepal_lenght  media_sepal_width  sd_sepal_width     n  
##   <fct>        <dbl>            <dbl>            <dbl>            <dbl> <int>  
## 1 setosa       5.01            0.352            3.43            0.379     50  
## 2 versic...     5.94            0.516            2.77            0.314     50  
## 3 virgin...    6.59            0.636            2.97            0.322     50
```

Outras funções do dplyr

- `across()`: facilita aplicar uma mesma operação em várias colunas.
- `rowwise()`: operações por linha.
- `relocate()`: reposicionar colunas

Tidyr

Tidyr

- Bom pacote para transformar bases bagunçadas em bases *tidy*
- Também usado para modificar a estrutura dos dados (pivotagem)
- Principais funções:
 - `separate()` e `unite()`: para separar variáveis concatenadas em uma única coluna ou uni-las.
 - `pivot_wider()` e `pivot_longer()`: para pivotar a base.

separate() e unite()

- Função **separate()**: separa duas ou mais variáveis que estão concatenadas em uma mesma coluna:

```
dados %>%
  separate(
    col = coluna_velha,
    into = c("colunas", "novas"),
    sep = "separador"
  )
```

- Função **unite()**: concatena os valores de várias variáveis em uma única coluna:

```
dados %>%
  unite(
    col = coluna_nova,
    colunas_para_juntar,
    sep = "separador"
  )
```

Pivotagem

- `pivot_longer()`: alonga/empilha os dados:

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

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

Pivotagem

- `pivot_wider()`: alarga/espalha os dados:

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

```
## # A tibble: 150 × 5  
##   rowid Sepal.Length Sepal.Width Petal.Length Petal.Width  
##   <int>     <dbl>     <dbl>      <dbl>      <dbl>  
## 1     1         5.1       3.5        1.4       0.2  
## 2     2         4.9       3.0        1.4       0.2  
## 3     3         4.7       3.2        1.3       0.2  
## 4     4         4.6       3.1        1.5       0.2  
## 5     5         5.0       3.6        1.4       0.2  
## 6     6         5.4       3.9        1.7       0.4  
## 7     7         4.6       3.4        1.4       0.3  
## 8     8         5.0       3.4        1.5       0.2  
## 9     9         4.4       2.9        1.4       0.2  
## 10   10        4.9       3.1        1.5       0.1  
## # ... with 140 more rows
```

Ggplot

Ggplot2

- Fruto da tese de doutorado de Hadley Wickham
- Gráfico estatístico
- Elementos de um gráfico são as suas camadas
- Construção de um gráfico se dá pela sobreposição dessas camadas.
- `{ggplot2}`: Construir um gráfico camada por camada

Vantagens sobre o R-base

- Gráficos mais bonitos
- Fácil personalização
- Aprendizado intuitivo
- Diferença no código entre tipos de gráficos diferentes é pequena

Primeira camada

- A primeira camada é dado pela função `ggplot()`

```
ggplot(data = iris)
```

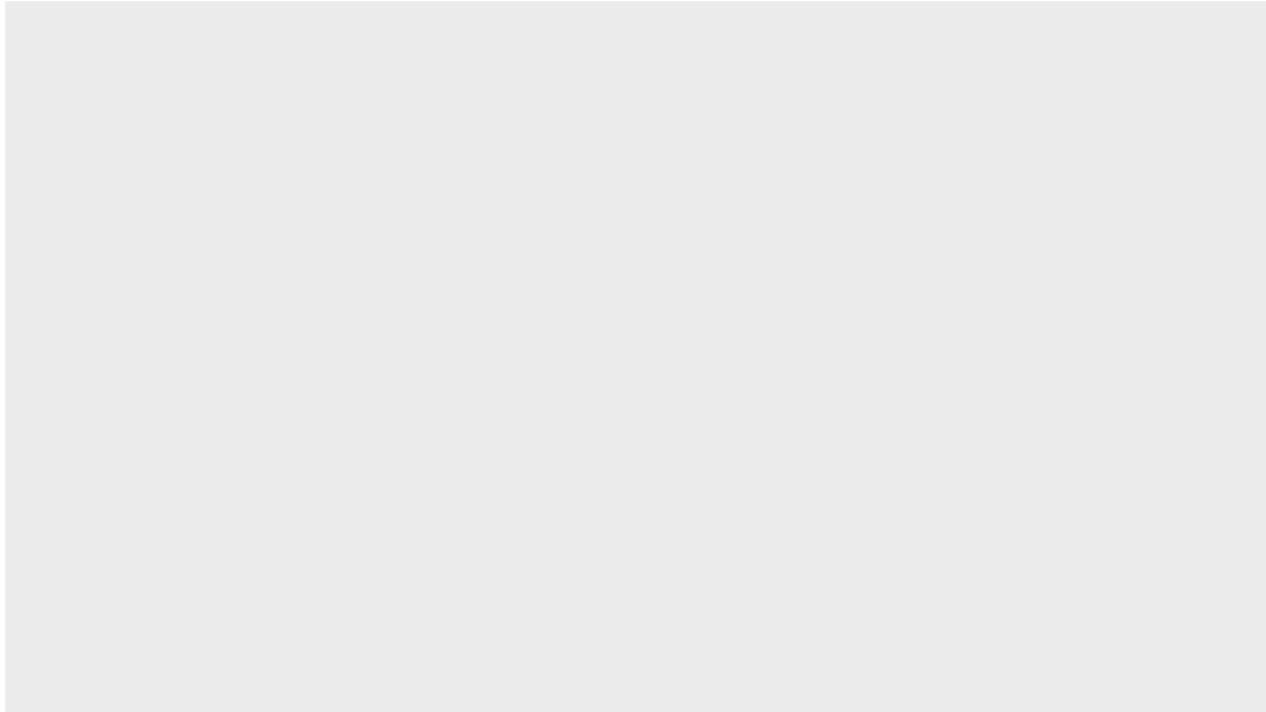


Gráfico de pontos

- Precisamos especificar como as observações serão mapeadas no gráfico
- Especificar quais formas geométricas serão utilizadas

```
ggplot(iris) +  
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width))
```

Gráfico de pontos

- Adicionando uma reta $y = -6 + 1.5x$

```
ggplot(iris) +  
  geom_abline(intercept = -6, slope = 1.5, color = "red") +  
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width))
```

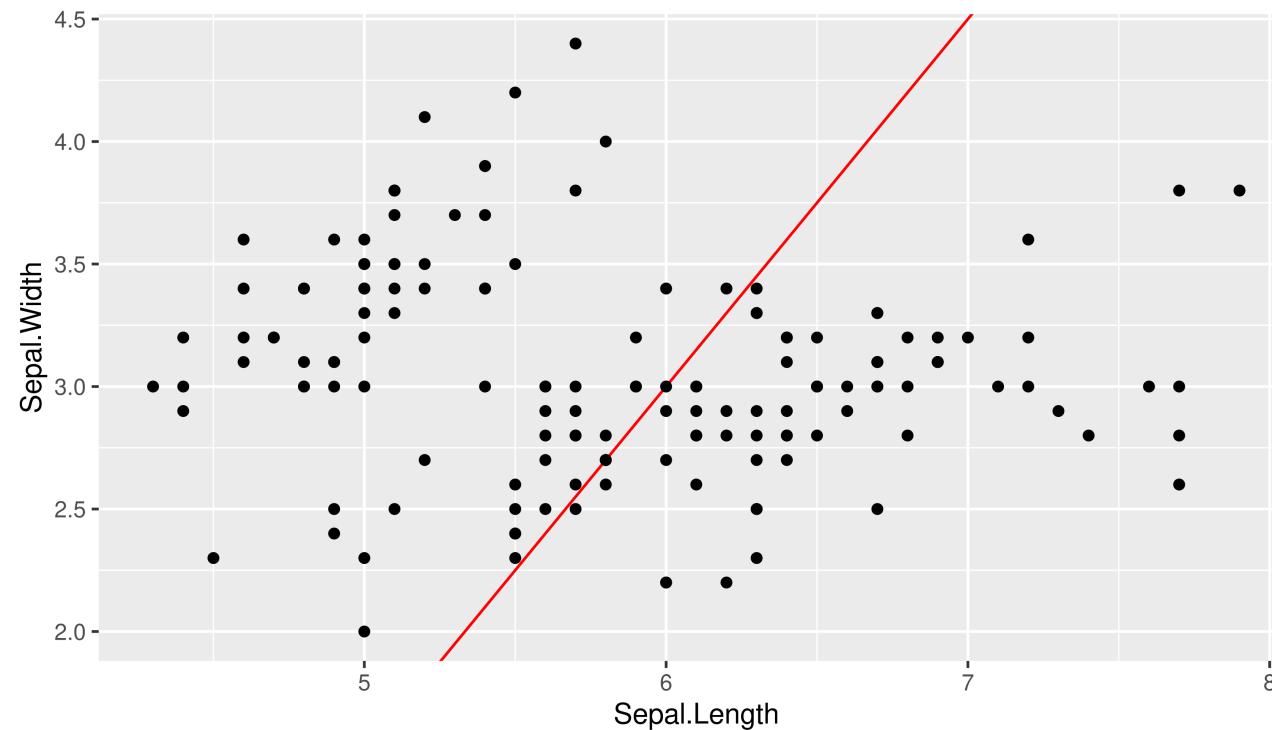


Gráfico de pontos

- Mapear a cor dos pontos de acordo com outra variável

```
# variavel continua
iris %>%
  ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width, color = Petal.Width))
```

Gráfico de pontos

- Para fator/variável categórica

```
# fator
iris %>%
ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width, color = Species))
```

Gráfico de pontos

- Única cor para tudo

```
# fator
iris %>%
ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width), color = "blue")
```

Gráfico de barras

- Criando nova categoria para ser plotada

```
# frequencia de categorias de comprimento de sepalias e petalias
iris_novo <- iris %>%
  mutate(Sepal_cat = case_when(Sepal.Length < 5 ~ "Menor que 5",
                               Sepal.Length >= 5 & Sepal.Length < 6 ~ "Maior que 5, menor que 6",
                               Sepal.Length >= 6 ~ "Maior/igual a 6"))
```

Gráfico de barras

- Devemos mudar a forma geométrica que queremos representar

```
iris_novo %>%
  count(Sepal_cat) %>%
  ggplot() +
  geom_col(aes(x = Sepal_cat, y = n))
```

Gráfico de barras

- Preencher barras de acordo com a categoria

```
iris_novo %>%  
  count(Sepal_cat) %>%  
  ggplot() +  
  geom_col(aes(x = Sepal_cat, y = n, fill = Sepal_cat), show.legend = FALSE)
```

Gráfico de barras

- Inverter gráfico de barras

```
iris_novo %>%  
  count(Sepal_cat) %>%  
  ggplot() +  
  geom_col(aes(y = Sepal_cat, x = n, fill = Sepal_cat), show.legend = FALSE)
```

Gráfico de barras

- Colocando número da frequência em cada barra

```
iris_novo %>%  
  count(Sepal_cat) %>% ggplot() +  
  geom_col(aes(x = Sepal_cat, y = n, fill = Sepal_cat), show.legend = FALSE) +  
  geom_label(aes(x = Sepal_cat, y = n/2, label = n)) +  
  coord_flip()
```

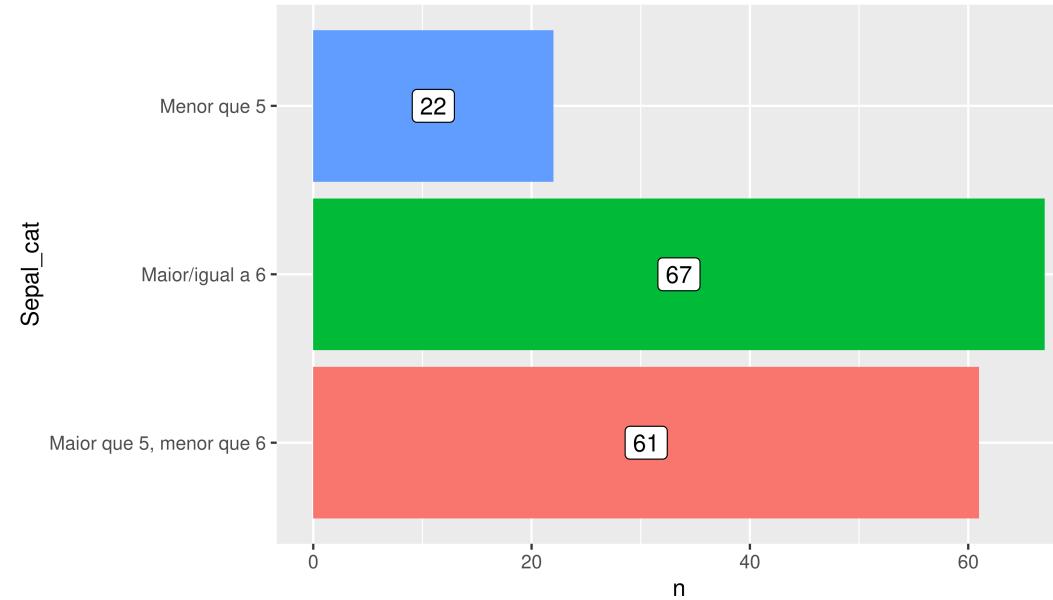
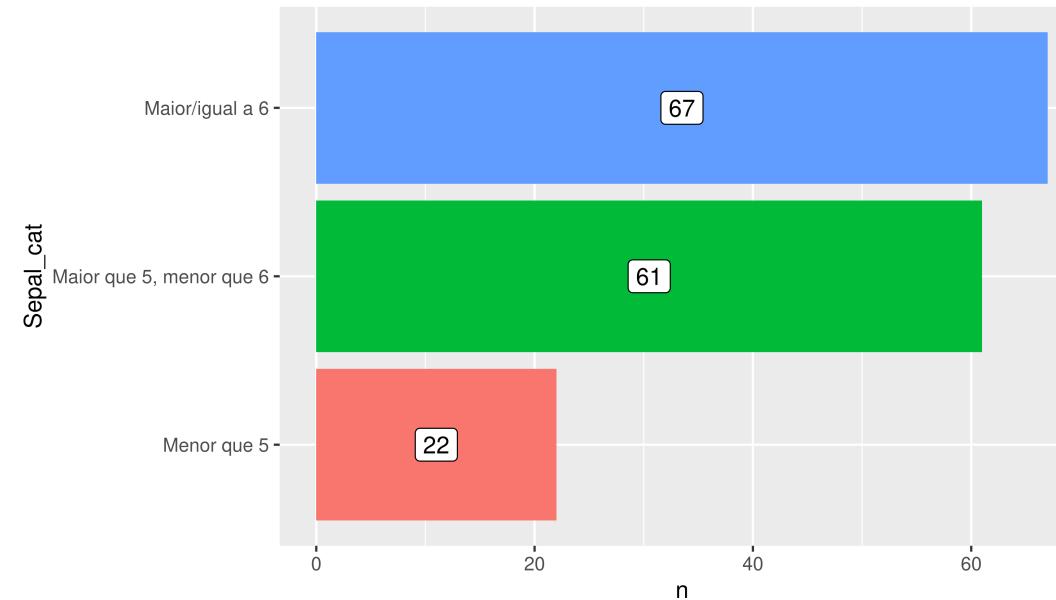


Gráfico de barras

- Reordenando as barras

```
iris_novo %>%  
  count(Sepal_cat) %>%  
  mutate(Sepal_cat = forcats::fct_reorder(Sepal_cat, n)) %>% ggplot() +  
  geom_col(aes(x = Sepal_cat, y = n, fill = Sepal_cat), show.legend = FALSE) +  
  geom_label(aes(x = Sepal_cat, y = n/2, label = n)) +  
  coord_flip()
```



Histogramas e boxplots

- Histograma rápido

```
iris %>%
  filter(Species == "setosa") %>%
  ggplot() +
  geom_histogram(aes(x = Sepal.Length))
```

Histogramas e boxplots

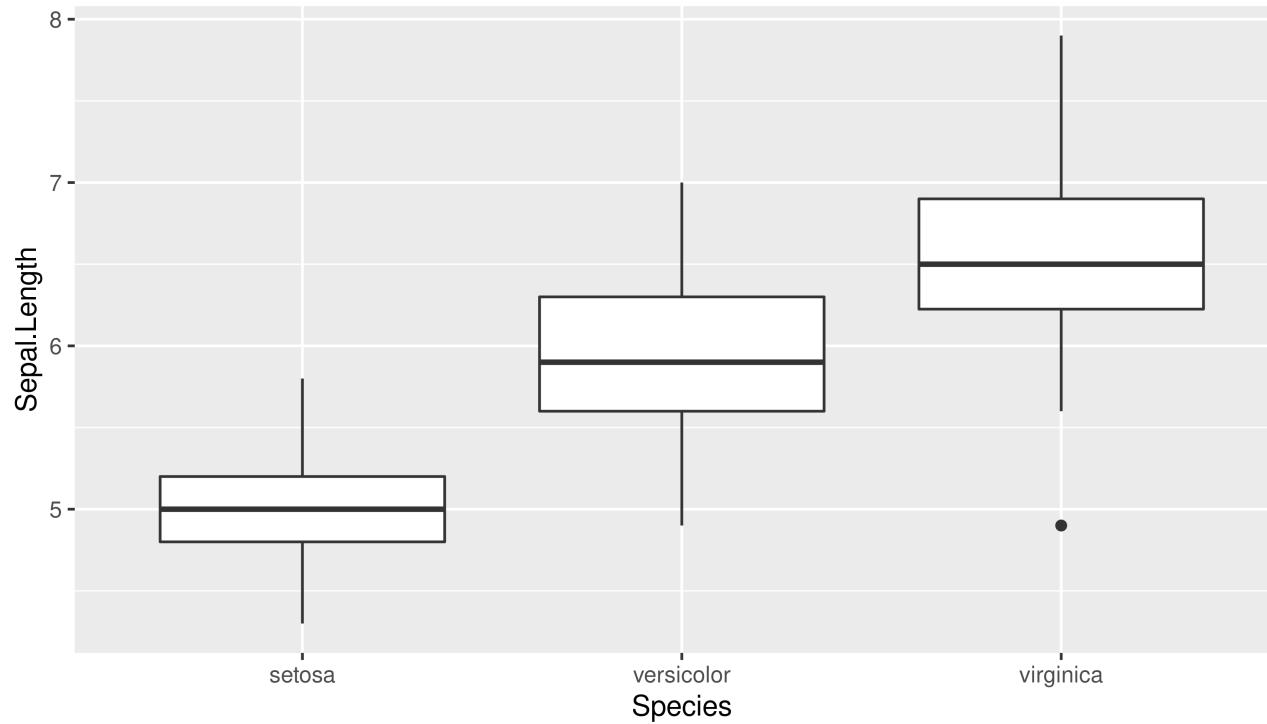
- Fixando largura da banda

```
iris %>%
  filter(Species == "setosa") %>%
  ggplot() +
  geom_histogram(aes(x = Sepal.Length), binwidth = 0.25, color = "white")
```

Histogramas e boxplots

- Boxplots por Species

```
iris %>%
  ggplot() +
  geom_boxplot(aes(x = Species, y = Sepal.Length))
```



Histogramas e boxplots

- Invertido

```
iris %>%
  ggplot() +
  geom_boxplot(aes(x = Species, y = Sepal.Length)) +
  coord_flip()
```

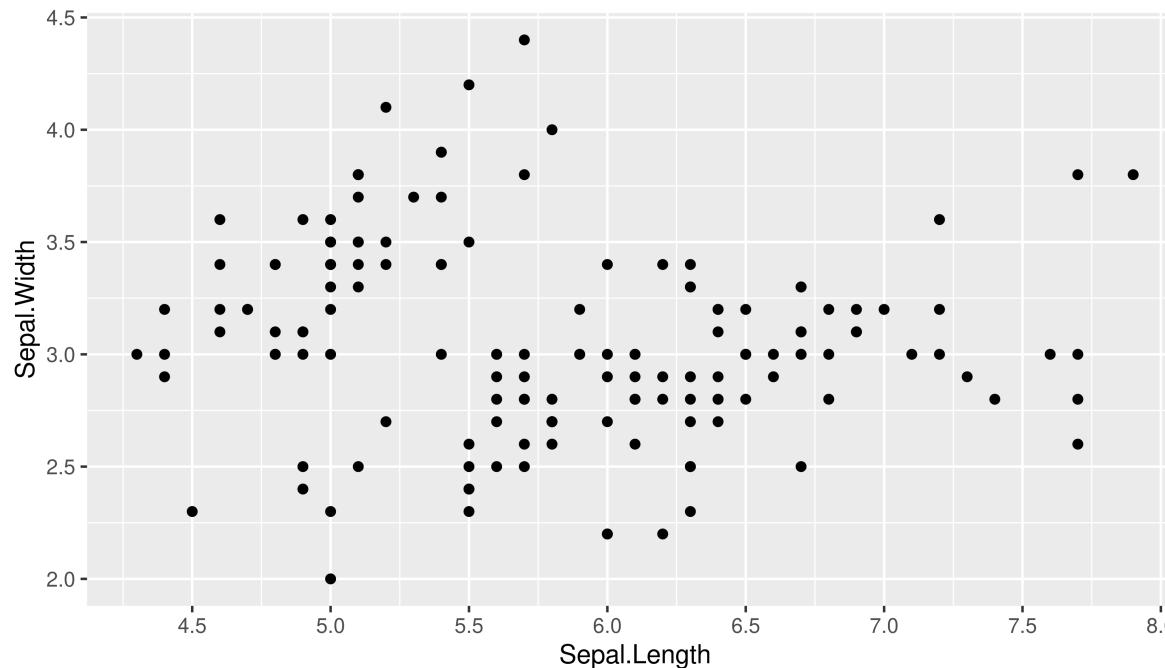
Títulos e labels

```
iris %>%
  ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  labs(x = "Comprimento de Sépala", y = "Largura de Sépala",
       color = "Espécies", title = "Gráfico de dispersão",
       subtitle = "Largura x Comprimento de Sépala")
```

Escalas

- Quebras (ticks) dos eixos x e y

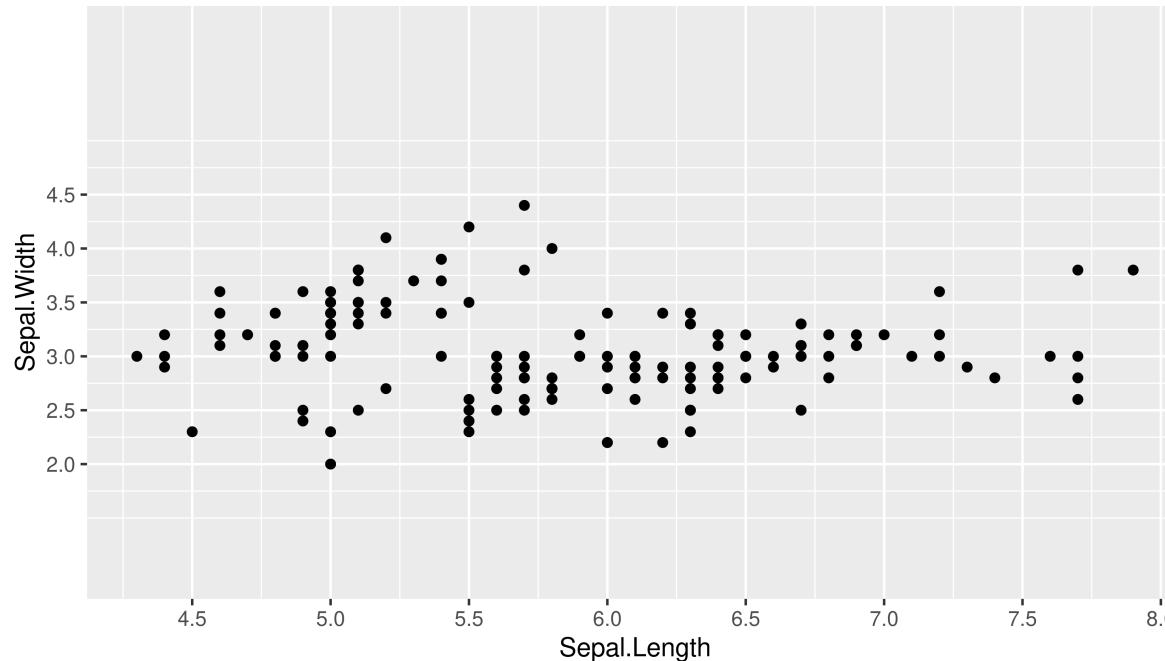
```
iris %>%
  ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  scale_x_continuous(breaks = seq(4, 8, 0.5)) +
  scale_y_continuous(breaks = seq(2, 4.5, 0.5))
```



Escalas

- Alterando limites

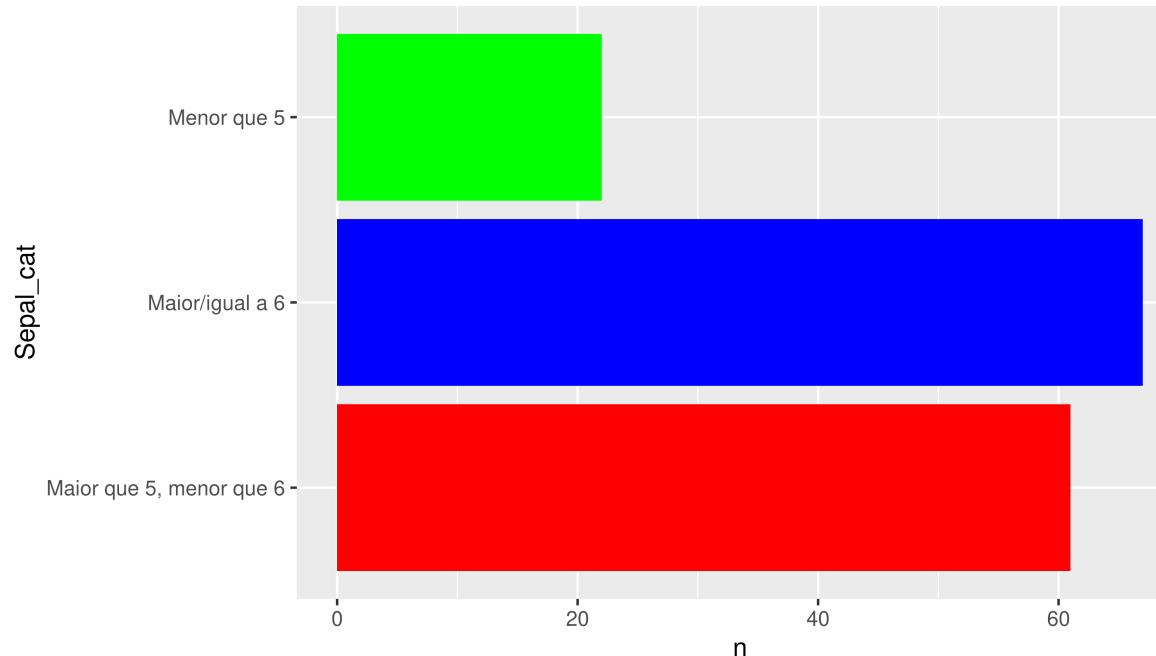
```
iris %>%
  ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  scale_x_continuous(breaks = seq(4, 8, 0.5)) +
  scale_y_continuous(breaks = seq(2, 4.5, 0.5)) + coord_cartesian(ylim = c(1, 6))
```



Escalas

- Escalas de cores

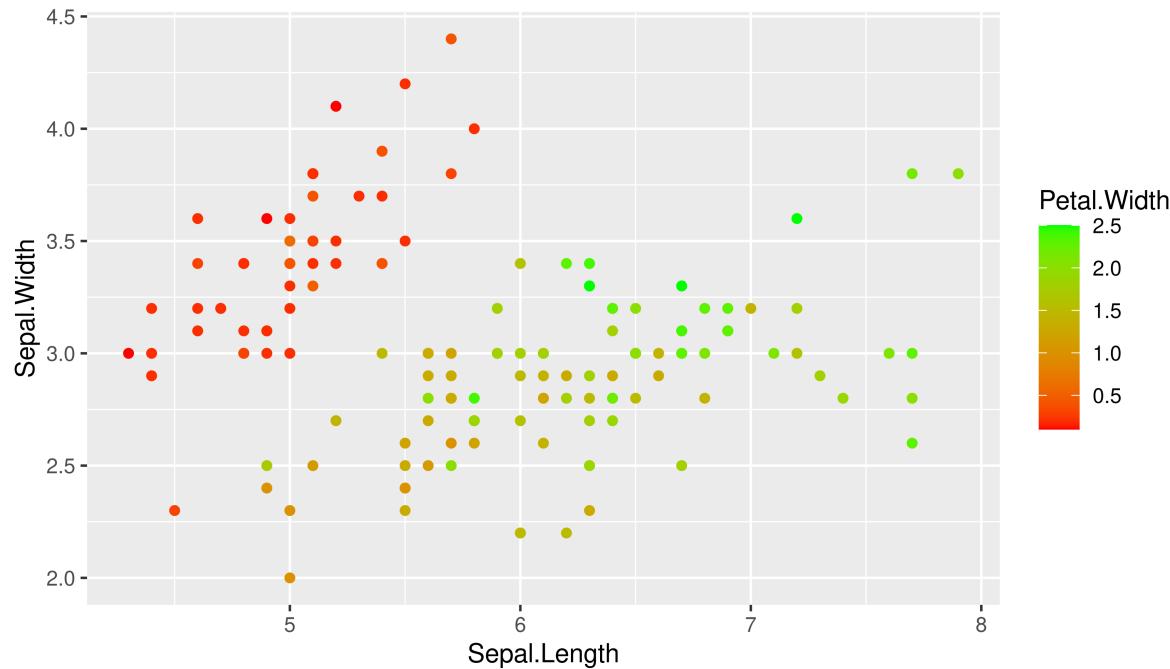
```
iris_novo %>%  
  count(Sepal_cat) %>%  
  ggplot() +  
  geom_col(aes(y = Sepal_cat, x = n, fill = Sepal_cat), show.legend = FALSE) +  
  scale_fill_manual(values = c("red", "blue", "green"))
```



Escalas

- Gradientes

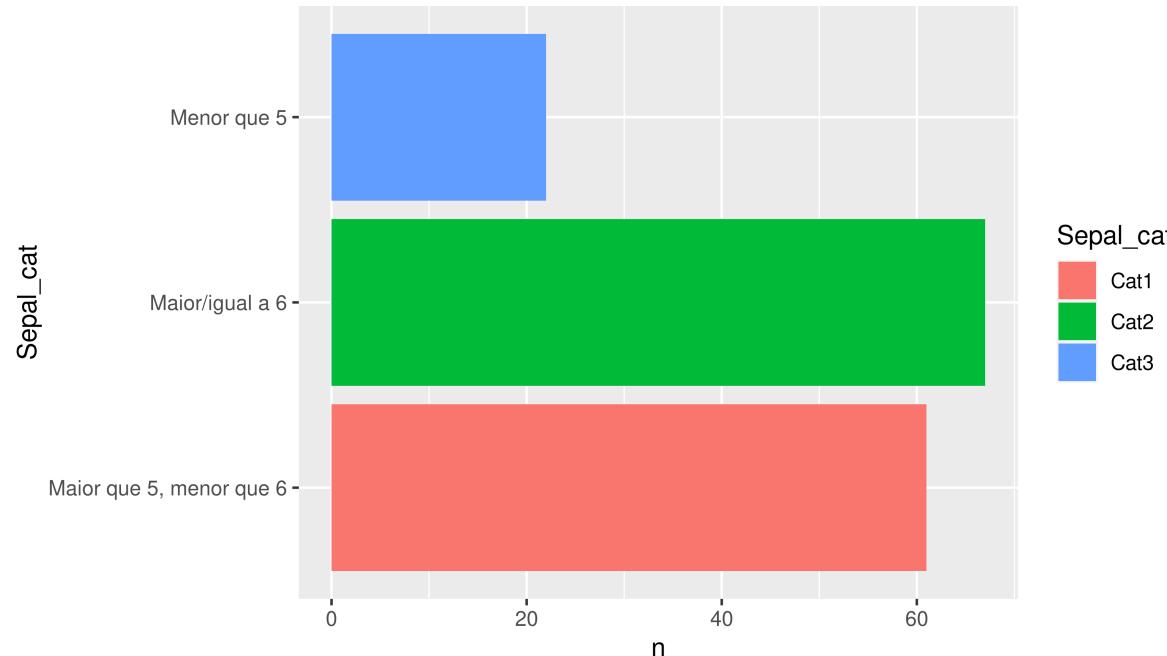
```
iris %>%
  ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width, color = Petal.Width)) +
  scale_color_gradient(low = "red", high = "green")
```



Escalas

- Mudando nome de categorias

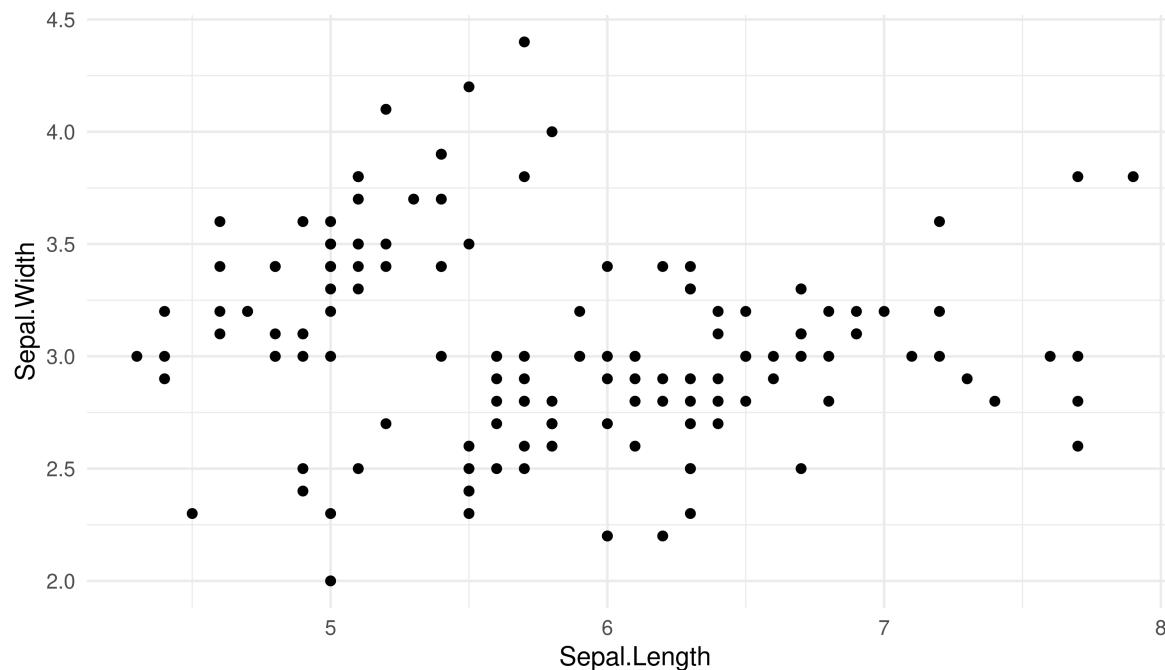
```
iris_novo %>%  
  count(Sepal_cat) %>%  
  ggplot() +  
  geom_col(aes(y = Sepal_cat, x = n, fill = Sepal_cat))+  
  scale_fill_discrete(labels = c("Cat1", "Cat2", "Cat3"))
```



Temas

- O tema usado até agora é o tema padrão do ggplot
- Podemos mudar isso, usando por exemplo `theme_minimal()`

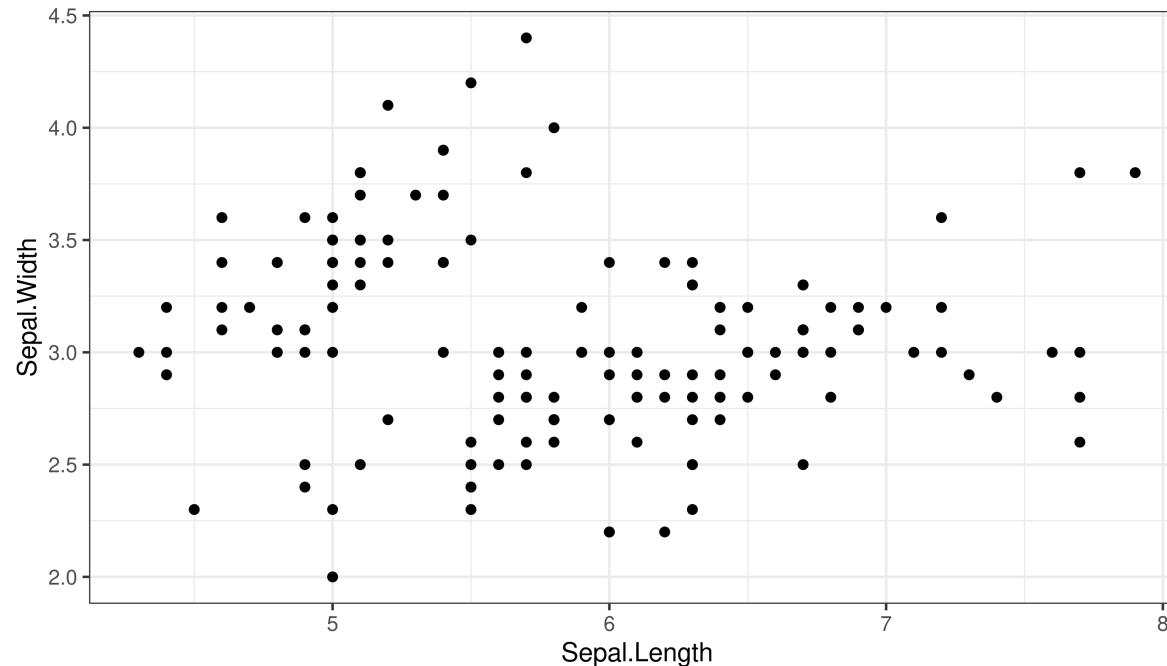
```
iris %>%  
  ggplot() +  
    geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +  
    theme_minimal()
```



Temas

- Tema em preto e branco `theme_bw()`

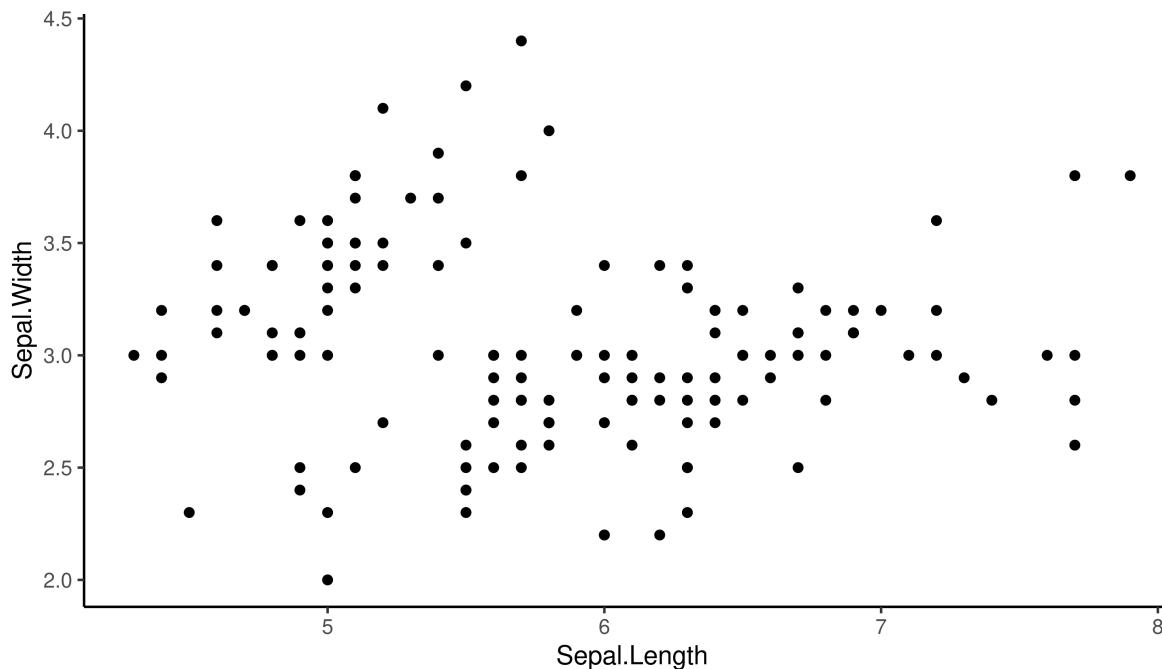
```
iris %>%  
  ggplot() +  
    geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +  
    theme_bw()
```



Temas

- Tema do R base `theme_classic()`

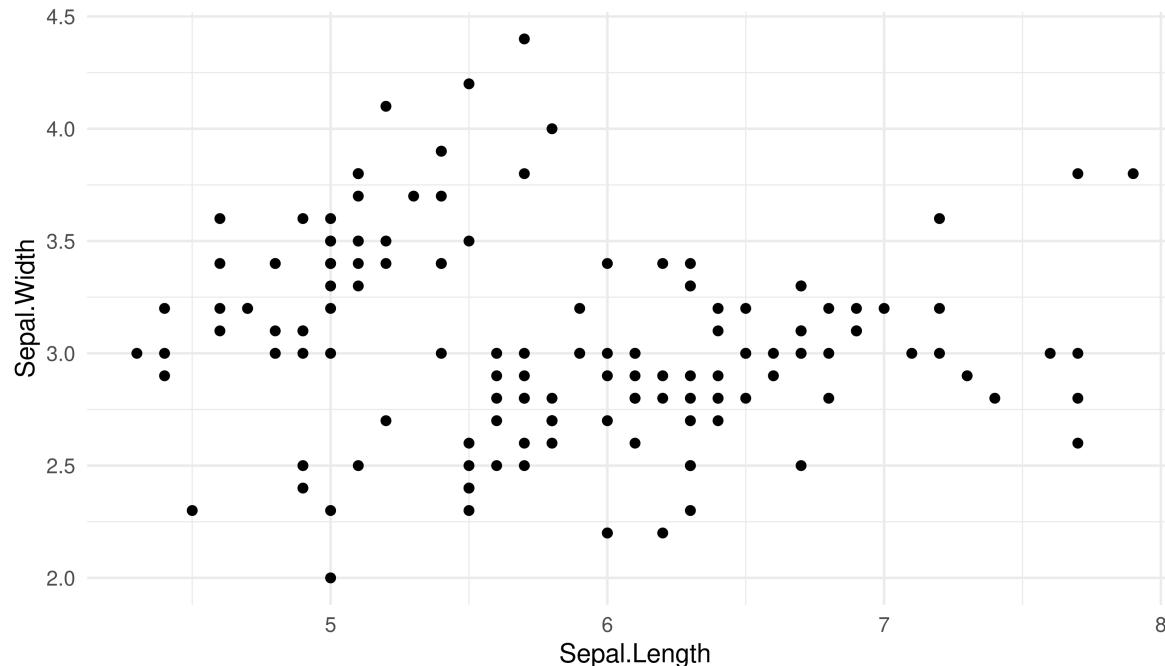
```
iris %>%  
  ggplot() +  
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +  
  theme_classic()
```



Temas

- Tema escuro `theme_dark()`

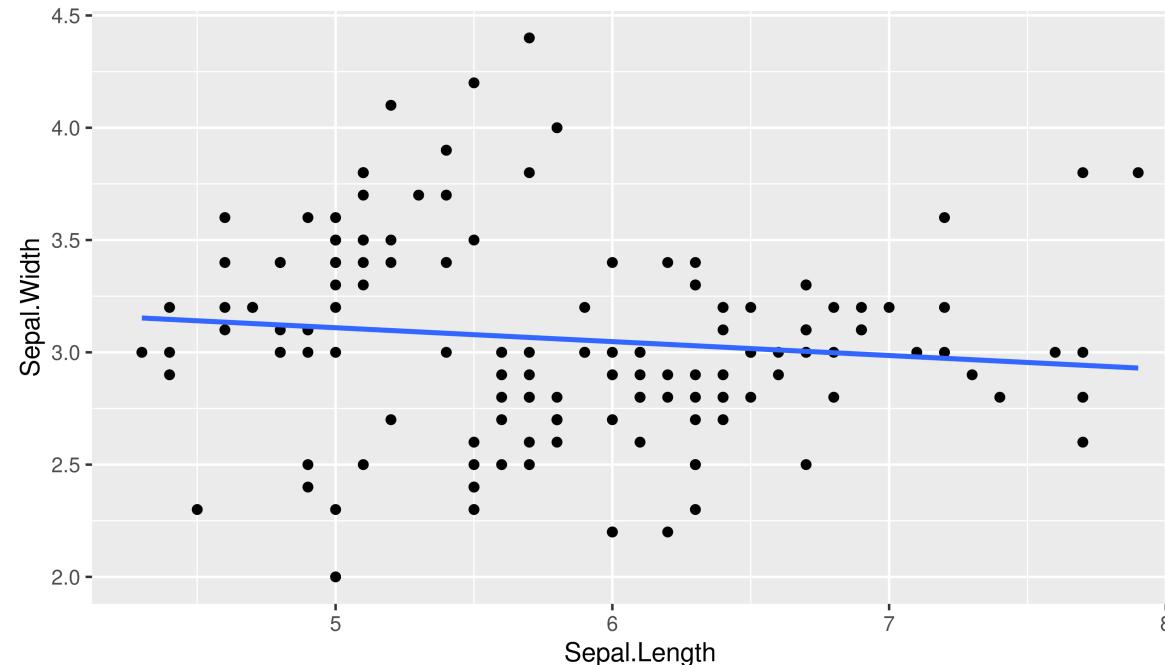
```
iris %>%
  ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  theme_minimal()
```



Juntando gráficos

- Podemos adicionar varios **geoms** em um mesmo gráfico

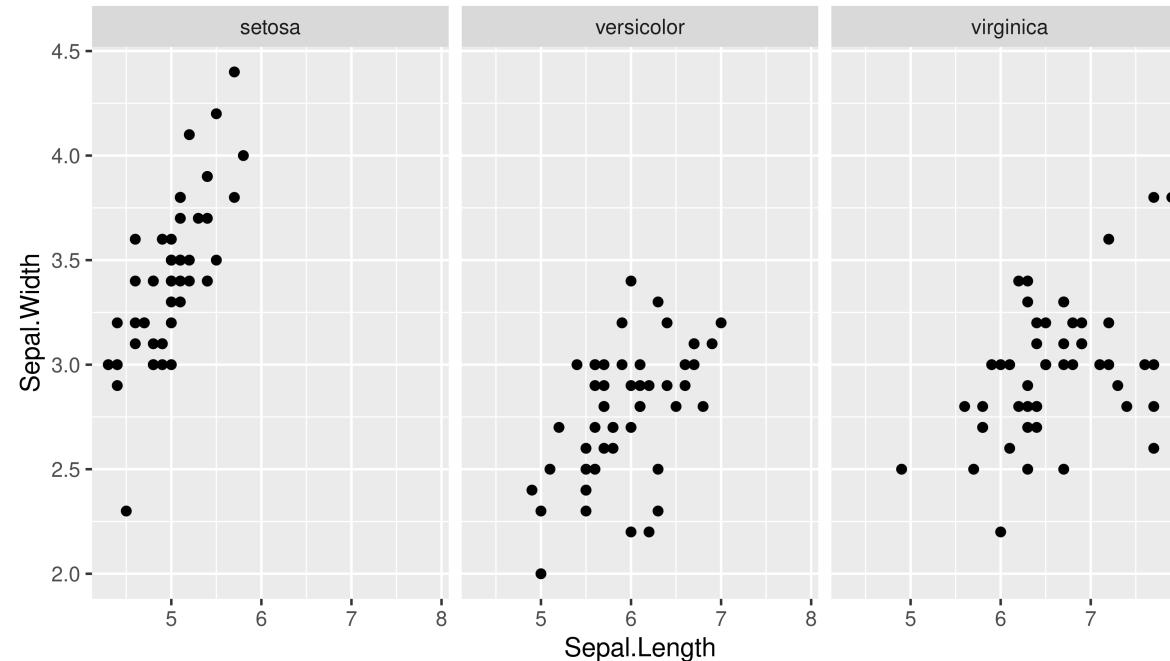
```
iris %>%  
  ggplot(aes(x = Sepal.Length, y = Sepal.Width)) +  
    geom_point() +  
    geom_smooth(se = FALSE, method = "lm")
```



Juntando gráficos

- Paineis de acordo com variáveis categóricas

```
iris %>%
  ggplot() +
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
  facet_wrap(~Species, ncol = 3)
```



Muito obrigado!

Referências

- Livro da Curso-R
- A tidyverse Cookbook
- R for data science