# Linguagens de Programação: R

**Aula 03:** Tidyverse - Manipulação de Dados & Visualização de Dados

Profa. Deisy Morselli Gysi, Ph.D.



# Agenda

- Tidy data
- Visualização de dados



# Tidyverse

- O Tidyverse é um conjunto de pacotes R que compartilham uma filosofia comum: eles foram projetados para tornar a análise de dados mais fácil e eficiente.
- Apresenta uma sintaxe consistente e intuitiva, permitindo que você se concentre no pensamento sobre seus dados, e não sobre a linguagem de programação.

```
Dados %>%
  filter(condicao) %>%
  select(variaveis) %>%
  group_by(variaveis) %>%
  summarise(media = mean(variavel)) %>%
  arrange(desc(media))
```

- As principais funções do `dplyr` são:
  - `filter()`: filtra linhas de um data frame.
  - `select()`: seleciona colunas de um data frame.
  - `mutate()`: cria ou modifica colunas de um data frame.
  - `arrange()`: ordena um data frame com base em uma ou mais colunas.
  - `group_by()`: agrupa um data frame por uma ou mais colunas.
  - `summarise()`: resume um data frame em um único valor.



# Sumarizando Dados com `summarise()`

- Para sumarizar dados, podemos utilizar a função `summarise()`.
- Vamos sumarizar o número total de automóveis envolvidos em acidentes no banco de dados `car_crash`.

```
library(data.table)
library(tidyverse)
car_crash <- fread("data/Brazil Total highway crashes 2010 - 2023.csv.gz")

car_crash1 = car_crash %>%
  summarise(total_automoveis = sum(automovel, na.rm = TRUE))
car_crash1
```

```
  total_automoveis
1           789971
```

- Podemos sumarizar mais de uma variável. Vamos sumarizar o número total de automóveis envolvidos em acidentes e o número total de mortos.

```
car_crash2 = car_crash %>%
  summarise(total_automoveis = sum(automovel, na.rm = TRUE),
            total_mortos = sum(mortos, na.rm = TRUE))
car_crash2
```

```
  total_automoveis total_mortos
1           789971        19430
```



# Agrupando Dados com `group_by()`

- Para agrupar dados, podemos utilizar a função `group_by()`.
- Vamos agrupar o banco de dados `car_crash` pela variável `ano`.
- Primeiro, vamos criar a variável `ano` a partir da variável `data`.

```
car_crash3 = car_crash %>%
  mutate(ano = year(dmy(data))) %>%
  group_by(ano)
glimpse(car_crash3)
```

```
Rows: 864,561
Columns: 25
Groups: ano [14]
$ data                  <chr> "01/01/2010", "01/01/2010", "01/01/2010…
$ horario               <chr> "04:21:00", "02:13:00", "03:35:00", "07…
$ n_da_ocorrencia       <chr> "18", "20", "000024/2010", "000038/2010…
$ tipo_de_ocorrencia    <chr> "sem vítima", "sem vítima", "sem vítima…
$ km                    <chr> "167", "269,5", "77", "52", "33", "24",…
$ trecho                <chr> "BR-393/RJ", "BR-116/PR", "BR-290/RS", …
$ sentido               <chr> "Norte", "Sul", "Norte", "Norte", "Nort…
$ lugar_acidente        <chr> "Rodovia do Aço", "Autopista Regis Bitt…
$ tipo_de_acidente      <chr> "Derrapagem", "Colisão Traseira", "COLI…
$ automovel             <int> 1, 2, 2, 0, 0, 1, 1, 1, 2, 1, NA, 1, 1,…
```



# Sumarizando Dados com `summarise()`

- Agora vamos sumarizar o número total de automóveis envolvidos em acidentes e o número total de mortos por ano.

```
car_crash4 = car_crash %>%
  mutate(ano = year(dmy(data))) %>%
  group_by(ano) %>%
  summarise(total_automoveis = sum(automovel, na.rm = TRUE),
            total_mortos = sum(mortos, na.rm = TRUE))
head(car_crash4)
```

```
# A tibble: 6 × 3
    ano total_automoveis total_mortos
  <dbl>            <int>        <int>
1  2010            51223         1472
2  2011            57531         1514
3  2012            59020         1468
4  2013            59855         1502
5  2014            67626         1684
6  2015            72166         1786
```



# Encadeando Funções

- Podemos encadear funções utilizando o operador `%>%`.
- Vamos filtrar as observações cujo tipo de ocorrência é `com vítima` e sumarizar o número total de automóveis envolvidos em acidentes e o número total de mortos.

```
car_crash5 = car_crash %>%
  filter(tipo_de_ocorrencia == "com vítima") %>%
  summarise(total_automoveis = sum(automovel, na.rm = TRUE),
            total_mortos = sum(mortos, na.rm = TRUE))
car_crash5
```

```
  total_automoveis total_mortos
1           152409        13356
```



# Exercícios

1. Utilizando o banco de dados `starwars` faça o que se pede:

- Qual é o número total de espécies únicas presentes? Qual a frequência de indivíduos por espécie?
- Calcule a altura média de personagens masculinos e femininos.
- Qual é o peso médio dos personagens de cada espécie para personagens masculinos?
- Para cada espécie presente na base de dados, identifique o personagem mais pesado e seu peso correspondente.



# Solução

```r
# Qual é o número total de espécies únicas presentes? Qual a frequência de indivídu
starwars %>%
  summarise(n_especies = n_distinct(species))

starwars %>%
  group_by(species) %>%
  summarise(freq_especies = n()) %>%
  arrange(desc(freq_especies))
```

```r
# Calcule a altura média de personagens masculinos e femininos.
starwars %>%
  filter(sex %in% c("female", "male")) %>%
  group_by(sex) %>%
  summarise(media_altura = mean(height, na.rm = TRUE))
```

```r
# Qual é a média de peso dos personagens de cada espécie para personagens masculino
starwars %>%
  filter(sex == "male") %>%
  group_by(species) %>%
  summarise(media_peso = mean(mass, na.rm = TRUE))
```

```r
# Para cada espécie presente na base de dados, identifique o personagem mais pesado

starwars %>%
  group_by(species) %>%
  filter(mass == max(mass, na.rm = TRUE)) %>%
  select(species, name, mass)
```



# Manipulação de datas

- Quando importamos datas em R (dentro de um data frame), elas são importadas como strings.
- Precisamos, portanto, transformar essas strings em objetos do tipo `date` para podermos manipulá-las, como por exemplo, extrair o ano, o mês, o dia, etc.
- Podemos extrair o ano, o mês e o dia de uma data utilizando as funções `year()`, `month()` e `day()`.

```
car_crash %>%
  mutate(data = dmy(data)) %>%
  select(data) %>%
  head(n =2)
```

```
         data
       <Date>
1: 2010-01-01
2: 2010-01-01
```

```
car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(ano = year(data),
         mes = month(data),
         dia = day(data)) %>%
  select(data, ano, mes, dia) %>%
  head()
```

```
         data   ano   mes   dia
       <Date> <num> <num> <int>
1: 2010-01-01  2010     1     1
2: 2010-01-01  2010     1     1
3: 2010-01-01  2010     1     1
4: 2010-01-01  2010     1     1
5: 2010-01-01  2010     1     1
6: 2010-01-01  2010     1     1
```



# Manipulação de datas

- Podemos calcular a diferença entre duas datas utilizando a função `difftime()`.

```
car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(dias_desde_acidente = difftime(Sys.Date(), data, units = "days")) %>%
  select(data, dias_desde_acidente) %>%
  head()
```

```
        data dias_desde_acidente
      <Date>          <difftime>
1: 2010-01-01           5271 days
2: 2010-01-01           5271 days
3: 2010-01-01           5271 days
4: 2010-01-01           5271 days
5: 2010-01-01           5271 days
6: 2010-01-01           5271 days
```



# Manipulação de datas

- Podemos somar ou subtrair dias, semanas e anos de uma data utilizando as funções `lubridate::days()`, `lubridate::weeks()`, `lubridate::years()`.

```
car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(data_mais_10_dias = data + lubridate::days(10)) %>%
  select(data, data_mais_10_dias) %>%
  head()
```

```
         data data_mais_10_dias
       <Date>            <Date>
1: 2010-01-01        2010-01-11
2: 2010-01-01        2010-01-11
3: 2010-01-01        2010-01-11
4: 2010-01-01        2010-01-11
5: 2010-01-01        2010-01-11
6: 2010-01-01        2010-01-11
```



# Manipulação de datas - Hora, minutos e segundos

- Podemos extrair a hora, os minutos e os segundos de uma data utilizando as funções `hour()`, `minute()` e `second()`.

```
data <- ymd_hms("2023-08-21 15:30:45")
ano <- year(data)
mes <- month(data)
dia <- day(data)
hora <- hour(data)
minuto <- minute(data)
segundo <- second(data)
```

```
print(ano)
```

```
[1] 2023
```

```
print(mes)
```

```
[1] 8
```

```
print(dia)
```

```
[1] 21
```

```
print(hora)
```

```
[1] 15
```

```
print(minuto)
```

```
[1] 30
```

```
print(segundo)
```

```
[1] 45
```



# Conversão de fuso horário

- Podemos converter o fuso horário de uma data utilizando a função `with_tz()`.

```
# Data original no fuso horário de Nova Iorque
data_ny <- ymd_hms("2023-08-21 12:00:00", tz = "America/New_York")

# Converter para o fuso horário de Londres
data_london <- with_tz(data_ny, tz = "Europe/London")

print(data_ny)
```

```
[1] "2023-08-21 12:00:00 EDT"
```

```
print(data_london)
```

```
[1] "2023-08-21 17:00:00 BST"
```



# Exercícios

1. Utilizando o banco de dados `car_crash` faça o que se pede:

- Quais os meses do ano com maior número de acidentes fatais?
- Quais os dias da semana com maior número de acidentes fatais?
  - Dica: Busque por uma função que retorne o dia da semana a partir de uma data.



# Solução

```
# Quais os meses do ano com maior número de acidentes fatais?
car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(ano = year(data),
         mes = month(data)) %>%
  select(data, ano, mes, mortos) %>%
  filter(mortos > 0) %>%
  group_by(mes) %>%
  summarise(total_mortos = sum(mortos)) %>%
  arrange(desc(total_mortos))
```



# Solução

```
# Quais os dias da semana com maior número de acidentes fatais?
car_crash %>%
  mutate(data = dmy(data)) %>%
  mutate(dia_semana = lubridate::wday(data, label = T, abbr = F)) %>%
  select(dia_semana, mortos) %>%
  filter(mortos > 0) %>%
  group_by(dia_semana) %>%
  summarise(total_mortos_dia = sum(mortos)) %>%
  arrange(desc(total_mortos_dia))
```



# Manipulação de dados

- Muitas vezes, temos informações que estão distribuídas em diferentes tabelas e precisamos juntá-las para realizar análises mais completas.
- Para isso, precisamos realizar alguma espécie de junção entre as tabelas.

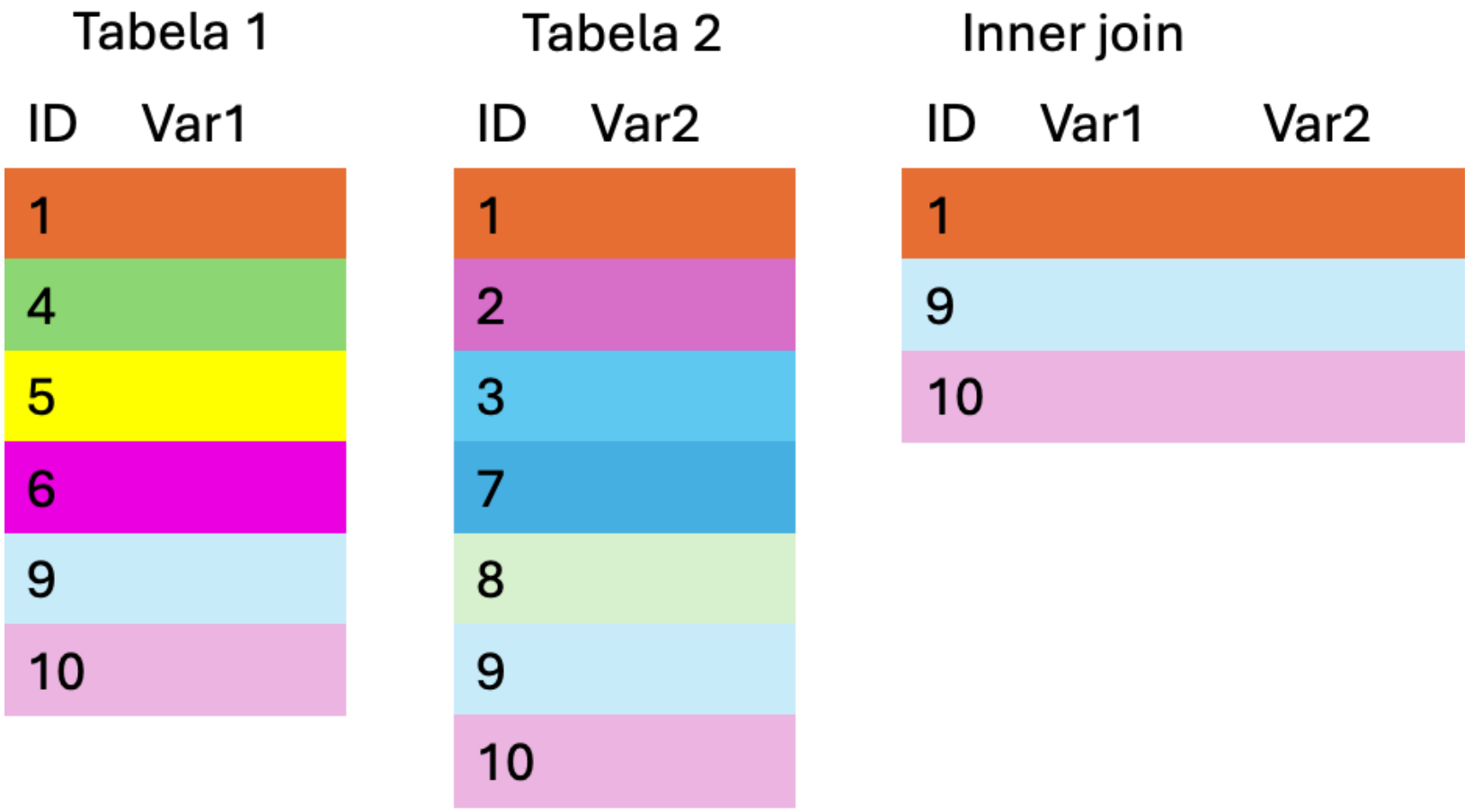
# Junção de tabelas

- Existem diferentes tipos de junção de tabelas:
  - `inner_join()`: Retorna apenas as linhas que possuem correspondência em ambas as tabelas.
  - `left_join()`: Retorna todas as linhas da tabela da esquerda e as linhas correspondentes da tabela da direita.
  - `right_join()`: Retorna todas as linhas da tabela da direita e as linhas correspondentes da tabela da esquerda.
  - `full_join()`: Retorna todas as linhas de ambas as tabelas.
  - `semi_join()`: Retorna todas as linhas da tabela da esquerda que possuem correspondência na tabela da direita.
  - `anti_join()`: Retorna todas as linhas da tabela da esquerda que não possuem correspondência na tabela da direita.



# Junção de tabelas: `inner_join()`

- Retorna apenas as linhas que possuem correspondência em ambas as tabelas.



# Junção de tabelas: `inner_join()`

```
tabela1 <- data.frame(id = c(1, 2, 3, 4),
                      nome = c("João", "Maria", "José", "Ana"))
tabela1
```

```
  id  nome
1  1  João
2  2 Maria
3  3  José
4  4   Ana
```

```
tabela2 <- data.frame(id = c(1, 2, 5, 6),
                      idade = c(20, 25, 30, 35))
tabela2
```

```
  id idade
1  1    20
2  2    25
3  5    30
4  6    35
```

```
inner_join(tabela1, tabela2, by = "id")
```

```
  id  nome idade
1  1  João    20
2  2 Maria    25
```



# Junção de tabelas: `left_join()`

- Retorna todas as linhas da tabela da esquerda e as linhas correspondentes da tabela da direita.

Tabela 1

| ID | Var1 |
|---|---|
| 1 | |
| 4 | |
| 5 | |
| 6 | |
| 9 | |
| 10 | |

Tabela 2

| ID | Var2 |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |

Left join

| ID | Var1 | Var 2 |
|---|---|---|
| 1 | | |
| 4 | | NA |
| 5 | | NA |
| 6 | | NA |
| 9 | | |
| 10 | | |

# Junção de tabelas: `left_join()`

```
tabela1
```

```
  id  nome
1  1  João
2  2 Maria
3  3  José
4  4   Ana
```

```
tabela2
```

```
  id idade
1  1    20
2  2    25
3  5    30
4  6    35
```

```
left_join(tabela1, tabela2, by = "id")
```

```
  id  nome idade
1  1  João    20
2  2 Maria    25
3  3  José    NA
4  4   Ana    NA
```



# Junção de tabelas: `right_join()`

- Retorna todas as linhas da tabela da direita e as linhas correspondentes da tabela da esquerda.

Tabela 1

| ID | Var1 |
|---|---|
| 1 | |
| 4 | |
| 5 | |
| 6 | |
| 9 | |
| 10 | |

Tabela 2

| ID | Var2 |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |

Right join

| ID | Var1 | Var2 |
|---|---|---|
| 1 | | |
| 2 | NA | |
| 3 | NA | |
| 4 | | |
| 7 | NA | |
| 8 | NA | |
| 9 | | |
| 10 | | |

# Junção de tabelas: `right_join()`

```
tabela1
```

```
  id  nome
1  1  João
2  2 Maria
3  3  José
4  4   Ana
```

```
tabela2
```

```
  id idade
1  1    20
2  2    25
3  5    30
4  6    35
```

```
right_join(tabela1, tabela2, by = "id")
```

```
  id  nome idade
1  1  João    20
2  2 Maria    25
3  5  <NA>    30
4  6  <NA>    35
```



# Junção de tabelas: `full_join()`

- Retorna todas as linhas de ambas as tabelas.

Tabela 1

| ID | Var1 |
|---|---|
| 1 | |
| 4 | |
| 5 | |
| 6 | |
| 9 | |
| 10 | |

Tabela 2

| ID | Var2 |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |

Full join

| ID | Var1 | Var2 |
|---|---|---|
| 1 | | |
| 2 | NA | |
| 3 | NA | |
| 4 | | NA |
| 5 | | NA |
| 6 | | NA |
| 7 | NA | |
| 8 | | |
| 9 | | |
| 10 | | |

# Junção de tabelas: `full_join()`

```
tabela1
```

```
  id  nome
1  1  João
2  2 Maria
3  3  José
4  4   Ana
```

```
tabela2
```

```
  id idade
1  1    20
2  2    25
3  5    30
4  6    35
```

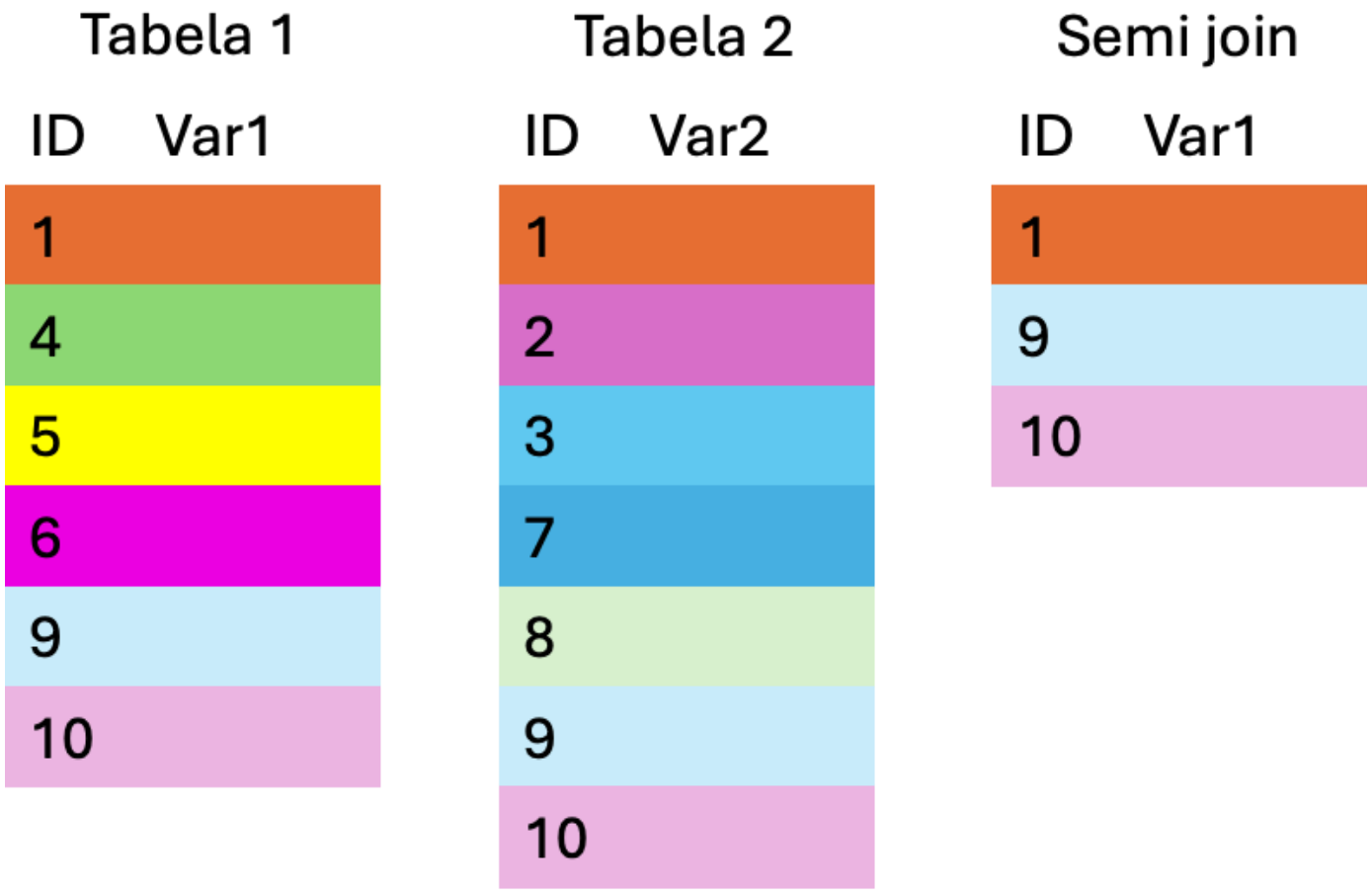
```
full_join(tabela1, tabela2, by = "id")
```

```
  id  nome idade
1  1  João    20
2  2 Maria    25
3  3  José    NA
4  4   Ana    NA
5  5  <NA>    30
6  6  <NA>    35
```



# Junção de tabelas: `semi_join()`

- É um filtro! Retorna todas as linhas da tabela da esquerda que possuem correspondência na tabela da direita.



# Junção de tabelas: `semi_join()`

```
tabela1
```

```
  id  nome
1  1  João
2  2 Maria
3  3  José
4  4   Ana
```

```
tabela2
```

```
  id idade
1  1    20
2  2    25
3  5    30
4  6    35
```

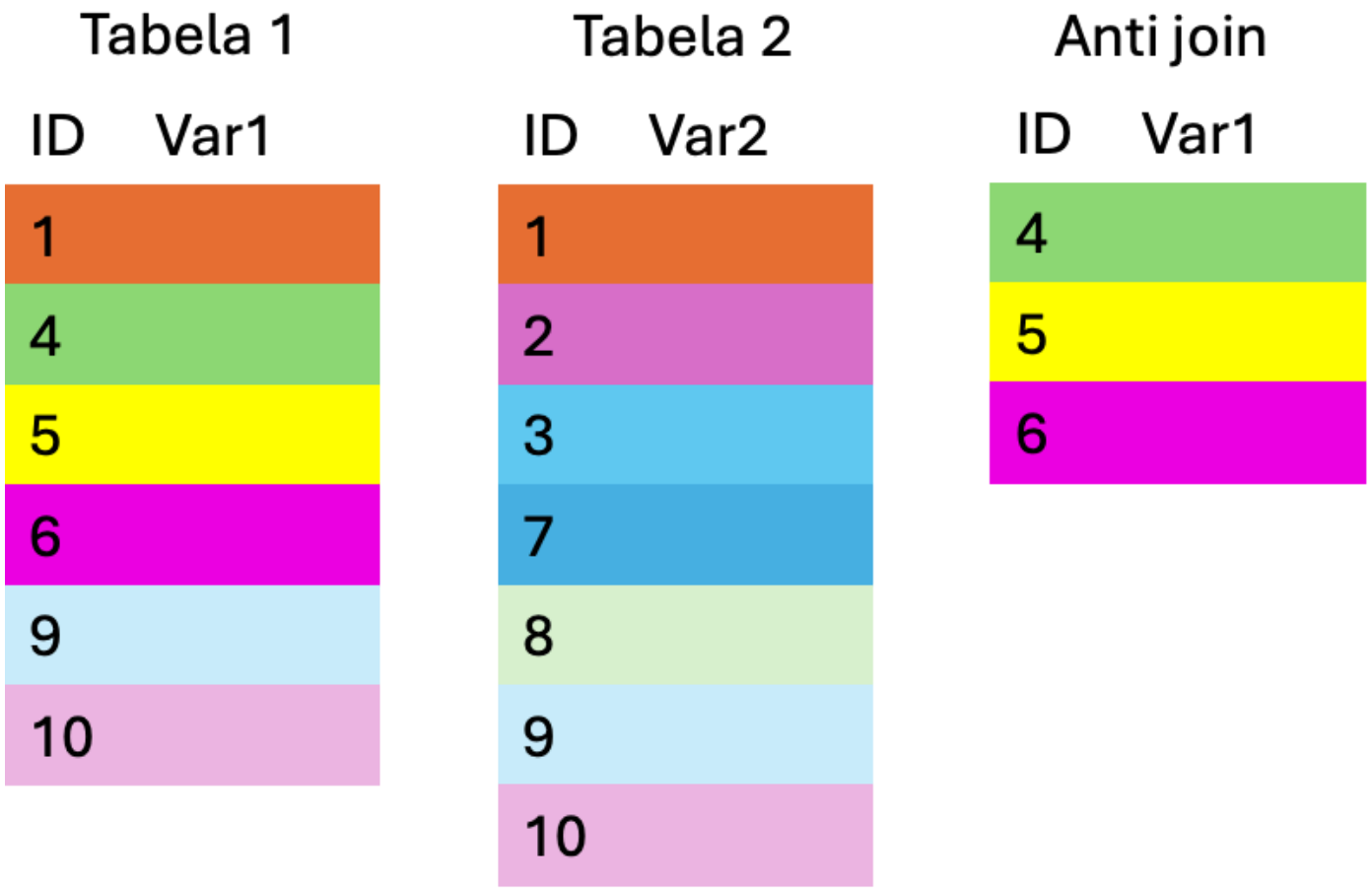
```
semi_join(tabela1, tabela2, by = "id")
```

```
  id  nome
1  1  João
2  2 Maria
```



# Junção de tabelas: `anti_join()`

- É um filtro! Retorna todas as linhas da tabela da esquerda que **não** possuem correspondência na tabela da direita.



# Junção de tabelas: `anti_join()`

```
tabela1
```

```
  id  nome
1  1  João
2  2 Maria
3  3  José
4  4   Ana
```

```
tabela2
```

```
  id idade
1  1    20
2  2    25
3  5    30
4  6    35
```

```
anti_join(tabela1, tabela2, by = "id")
```

```
  id nome
1  3 José
2  4  Ana
```



# Exercícios

- Utilize os dados do pacote `nycflights13` para responder as perguntas abaixo:
  1. Para vôos com atraso superior a 12 horas em `flights`, verifique as condições climáticas em `weather`. Quais os meses do ano em que você encontra o maior número de atrasos?
  2. Encontre os 20 destinos mais comuns e identifique seu aeroporto.
  3. Inclua uma coluna com a cia aérea na tabela `planes`. Quantas companhias áreas voaram cada avião naquele ano?
  4. Inclua a latitude e longitude de cada origem destino na tabela `flights`.



# Solução

1. Para vôos com atraso superior a 12 horas em `flights`, verifique as condições climáticas em `weather`. Há algum padrão? Quais os meses do ano em que você encontra o maior número de atrasos? Nota: `arr_delay` está em minutos, `precip` em polegadas, `temp` em Fahrenheit, `wind_speed` em mph.

```
require(nycflights13)
flights %>%
  filter(arr_delay > 12*60 | dep_delay > 12*60) %>%
  inner_join(weather, by = c("year", "month", "day", "hour", "origin")) %>%
  group_by(month) %>%
  summarise(total_atrasos = n(),
mean_temp = mean(temp, na.rm = T),
mean_dewp = mean(dewp, na.rm = T),
mean_humid = mean(humid, na.rm = T),
mean_wind_speed = mean(wind_speed, na.rm = T),
mean_precip = mean(precip, na.rm = T)
) %>%
  arrange(desc(total_atrasos))
```

```
# A tibble: 10 × 7
   month total_atrasos mean_temp mean_dewp mean_humid mean_wind_speed
   <int>         <int>     <dbl>     <dbl>      <dbl>           <dbl>
 1     4             5      59.8      54.2       83.5            23.2
 2     6             5      75.6      67.2       75.4            11.7
 3     2             4      34.7      22.1       61.8            12.1
 4    12             4      38.5      32.8       80.8            12.7
 5     1             3      42.3      24.3       50.6             9.21
```



# Solução

2. Encontre os 20 destinos mais comuns e identifique seu aeroporto.

```
flights %>%
  group_by(dest) %>%
  summarise(total_voos = n()) %>%
  arrange(desc(total_voos)) %>%
  head(20) %>%
  inner_join(airports, by = c("dest" = "faa"))
```

```
# A tibble: 20 × 9
   dest  total_voos name                      lat    lon   alt    tz dst   tzone
   <chr>      <int> <chr>                   <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
 1 ORD        17283 Chicago Ohare Intl       42.0  -87.9   668    -6 A     Amer…
 2 ATL        17215 Hartsfield Jackson Atl…  33.6  -84.4  1026    -5 A     Amer…
 3 LAX        16174 Los Angeles Intl         33.9 -118.    126    -8 A     Amer…
 4 BOS        15508 General Edward Lawrenc…  42.4  -71.0    19    -5 A     Amer…
 5 MCO        14082 Orlando Intl             28.4  -81.3    96    -5 A     Amer…
 6 CLT        14064 Charlotte Douglas Intl   35.2  -80.9   748    -5 A     Amer…
 7 SFO        13331 San Francisco Intl       37.6 -122.     13    -8 A     Amer…
 8 FLL        12055 Fort Lauderdale Hollyw…  26.1  -80.2     9    -5 A     Amer…
 9 MIA        11728 Miami Intl               25.8  -80.3     8    -5 A     Amer…
10 DCA         9705 Ronald Reagan Washingt…  38.9  -77.0    15    -5 A     Amer…
```



# Solução

3. Inclua uma coluna com a cia aérea na tabela `planes`. Quantas companhias áreas voaram cada avião naquele ano?

```
flights %>%
  inner_join(planes, by = "tailnum") %>%
  group_by(tailnum, carrier) %>%
  summarise(total_voos = n()) %>%
  arrange(desc(total_voos))
```

```
# A tibble: 3,339 × 3
# Groups:   tailnum [3,322]
   tailnum carrier total_voos
   <chr>   <chr>        <int>
 1 N711MQ  MQ             486
 2 N258JB  B6             427
 3 N298JB  B6             407
 4 N353JB  B6             404
 5 N351JB  B6             402
 6 N328AA  AA             393
 7 N228JB  B6             388
 8 N338AA  AA             388
 9 N327AA  AA             387
```



# Solução

4. Inclua a latitude e longitude de cada origem destino na tabela `flights`.

```
flights %>%
  inner_join(airports, by = c("origin" = "faa")) %>%
  inner_join(airports, by = c("dest" = "faa")) %>%
  select(origin, dest, lat.x, lon.x, lat.y, lon.y)
```

```
# A tibble: 329,174 × 6
   origin dest  lat.x lon.x lat.y lon.y
   <chr>  <chr> <dbl> <dbl> <dbl> <dbl>
 1 EWR    IAH    40.7 -74.2  30.0 -95.3
 2 LGA    IAH    40.8 -73.9  30.0 -95.3
 3 JFK    MIA    40.6 -73.8  25.8 -80.3
 4 LGA    ATL    40.8 -73.9  33.6 -84.4
 5 EWR    ORD    40.7 -74.2  42.0 -87.9
 6 EWR    FLL    40.7 -74.2  26.1 -80.2
 7 LGA    IAD    40.8 -73.9  38.9 -77.5
 8 JFK    MCO    40.6 -73.8  28.4 -81.3
 9 LGA    ORD    40.8 -73.9  42.0 -87.9
10 JFK    PBI    40.6 -73.8  26.7 -80.1
```



# Tidy data



# O que é tidy data?

- Hadley Wickham, em seu artigo “Tidy Data” (2014), define que um conjunto de dados é tidy se:
  1. Cada variável forma uma coluna.
  2. Cada observação forma uma linha.
  3. Cada tipo de unidade observacional forma uma tabela.
- Existem dois formatos de dados que podem ser tidy:
  1. Dados em formato *wide*.
  2. Dados em formato *long*.



# Dados no formato *wide*

- Cada variável é representada por uma coluna separada e cada observação (ou instância) ocupa uma única linha.
- Adequado para conjuntos de dados com poucas variáveis, onde as informações são bem condensadas.

| ID | Var1 | Var2 | Var3 | Var4 |
|---|---|---|---|---|
| a | 1 | 2 | 3 | 4 |
| b | 5 | 6 | 7 | 8 |
| c | | | | |
| d | | | | |
| ... | | | | |

# Dados no formato *long*

- As variáveis estão empilhadas em uma única coluna, enquanto uma coluna adicional é usada para indicar o nome da variável. Cada observação é representada por uma linha separada.
- Adequado para conjuntos de dados com muitas variáveis, onde as informações são mais detalhadas.

| ID | nome | valor |
|---|---|---|
| a | Var1 | 1 |
| a | Var2 | 2 |
| a | Var3 | 3 |
| a | Var4 | 4 |
| ... | | |

# Exemplos

- Considere o seguinte conjunto de dados em formato *wide*:

| id | nome | idade | sexo | altura |
|---|---|---|---|---|
| 1 | Ana | 25 | F | 1.65 |
| 2 | João | 30 | M | 1.80 |
| 3 | Maria | 22 | F | 1.70 |
| 4 | Pedro | 28 | M | 1.75 |

- Para passar para o formato *long*, basta empilhar as variáveis:

| id | variável | valor |
|---|---|---|
| 1 | nome | Ana |
| 1 | idade | 25 |
| 1 | sexo | F |
| 1 | altura | 1.65 |
| 2 | nome | João |
| 2 | idade | 30 |
| 2 | sexo | M |
| 2 | altura | 1.80 |



# Exemplos

- Considere o seguinte conjunto de dados com informações sobre tratamento de indivíduos com pedra nos rins:

| Tamanho da pedra | Tratamento A (Recuperados) | Tratamento A (Falhas) | Tratamento B (Recuperados) | Tratamento B (Falhas) |
|---|---|---|---|---|
| Pequena | 10 | 5 | 15 | 3 |
| Média | 5 | 3 | 10 | 2 |
| Grande | 2 | 1 | 5 | 1 |

- Para passar para o formato *long*, basta empilhar as variáveis:

| Tamanho da pedra | Tratamento | Recuperados | Falhas |
|---|---|---|---|
| Pequena | A | 10 | 5 |
| Pequena | B | 15 | 3 |
| Média | A | 5 | 3 |
| Média | B | 10 | 2 |
| Grande | A | 2 | 1 |
| Grande | B | 5 | 1 |

# Pivotando dados em R

- Vamos utilizar o banco de dados `table1`.
- Dados de casos reportados de Tuberculose e o tamanho da população em dois anos para três países.
- Esses dados são provenientes dos dados WHO.

```
table1
```

```
# A tibble: 6 × 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```



# pivot_wider()

- A função `pivot_wider()` é utilizada para transformar dados de formato *long* para *wide*.
- A função `pivot_wider()` requer os seguintes argumentos:
  - `names_from`: coluna que contém os nomes das variáveis que serão transformadas em colunas.
  - `values_from`: coluna que contém os valores das variáveis que serão transformadas em colunas.
- Vamos transformar os dados de `table1` para o formato *wide*. Suponha que queremos que os dados sejam organizados por país e ano, e as observações sejam os casos de tuberculose.

```
table1 %>%
  select(-population) %>%
  pivot_wider(names_from = year,
              values_from = cases)
```

```
# A tibble: 3 × 3
  country     `1999` `2000`
  <chr>        <dbl>  <dbl>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```



# Pivotando com mais de uma variável

- Suponha que queremos que os dados sejam organizados por país, e as observações sejam os casos de tuberculose, separados por ano e tamanho da população.

```
table1 %>%
  pivot_wider(names_from = year,
              values_from = c(cases, population))
```

```
# A tibble: 3 × 5
  country     cases_1999 cases_2000 population_1999 population_2000
  <chr>            <dbl>      <dbl>           <dbl>           <dbl>
1 Afghanistan        745       2666        19987071        20595360
2 Brazil           37737      80488       172006362       174504898
3 China           212258     213766      1272915272      1280428583
```



# `pivot_longer()`

- A função `pivot_longer()` é utilizada para transformar dados de formato *wide* para *long*.
- A função `pivot_longer()` requer os seguintes argumentos:
  - `cols`: colunas que serão empilhadas.
  - `names_to`: coluna que conterá os nomes das variáveis empilhadas.
  - `values_to`: coluna que conterá os valores das variáveis empilhadas.
  - `values_fill`: valor que preencherá as células vazias.
  - `values_fn`: função que será aplicada aos valores empilhados.
- Vamos transformar os dados de `table1` para o formato *long*. Suponha que queremos que os dados sejam organizados por país, e as observações sejam os casos de tuberculose e a população.

```
table1 %>%
  pivot_longer(cols = c(cases, population),
               names_to = "variable",
               values_to = "total")
```

```
# A tibble: 12 × 4
   country      year variable         total
   <chr>       <dbl> <chr>            <dbl>
 1 Afghanistan  1999 cases              745
 2 Afghanistan  1999 population    19987071
```

# Separando observações

- Algumas vezes, as observações estão agrupadas em uma única coluna e precisamos separar elas.
- A função `separate()` é utilizada para separar observações em diferentes colunas.
- Observe os dados em `table3`.

```
table3
```

```
# A tibble: 6 × 3
  country      year rate
  <chr>       <dbl> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```



# Separando observações

- Suponha que queremos separar a coluna `rate` em duas colunas: `cases` e `population`.

```
table3 %>%
  separate(rate, into = c("cases", "population"))
```

```
# A tibble: 6 × 4
  country      year cases  population
  <chr>       <dbl> <chr>  <chr>
1 Afghanistan  1999 745    19987071
2 Afghanistan  2000 2666   20595360
3 Brazil       1999 37737  172006362
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```



# Juntando observações

- A função `unite()` é utilizada para juntar observações de diferentes colunas em uma única coluna.
- Suponha que queremos juntar as colunas `cases` e `population` em uma única coluna chamada `rate`.

```
table1 %>%
  unite(rate, cases, population, sep = "/")
```

```
# A tibble: 6 × 3
  country      year rate
  <chr>       <dbl> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```



# Exercícios

1. Utilizando os dados de `flights`, crie uma matriz que mostra o número de voos entre cada par de aeroportos.



# Solução

```
flights %>%
  count(origin, dest) %>%
  pivot_wider(names_from = origin,
              values_from = n,
              values_fill = 0)
```

```
# A tibble: 105 × 4
   dest    EWR   JFK   LGA
   <chr> <int> <int> <int>
 1 ALB     439     0     0
 2 ANC       8     0     0
 3 ATL    5022  1930 10263
 4 AUS     968  1471     0
 5 AVL     265     0    10
 6 BDL     443     0     0
 7 BNA    2336   730  3267
 8 BOS    5327  5898  4283
 9 BQN     297   599     0
10 BTV     931  1364   294
```



# Visualização de dados

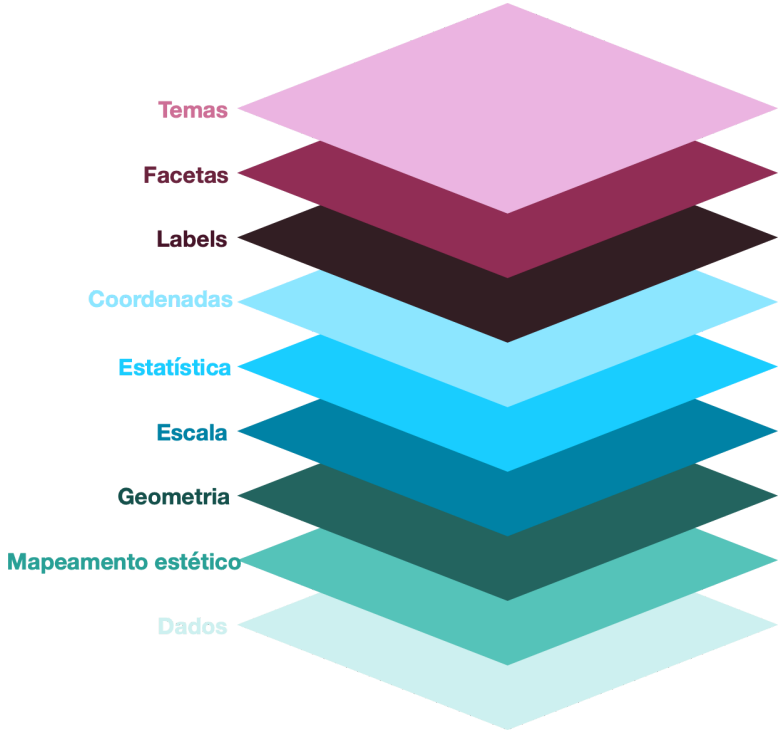
# ggplot2

- O pacote `ggplot2` é um pacote para criação de gráficos em R.
- `gg` significa “Grammar of Graphics”.
- O `ggplot2` é baseado na gramática de gráficos, que é uma forma de descrever gráficos por meio de camadas.



# Camadas

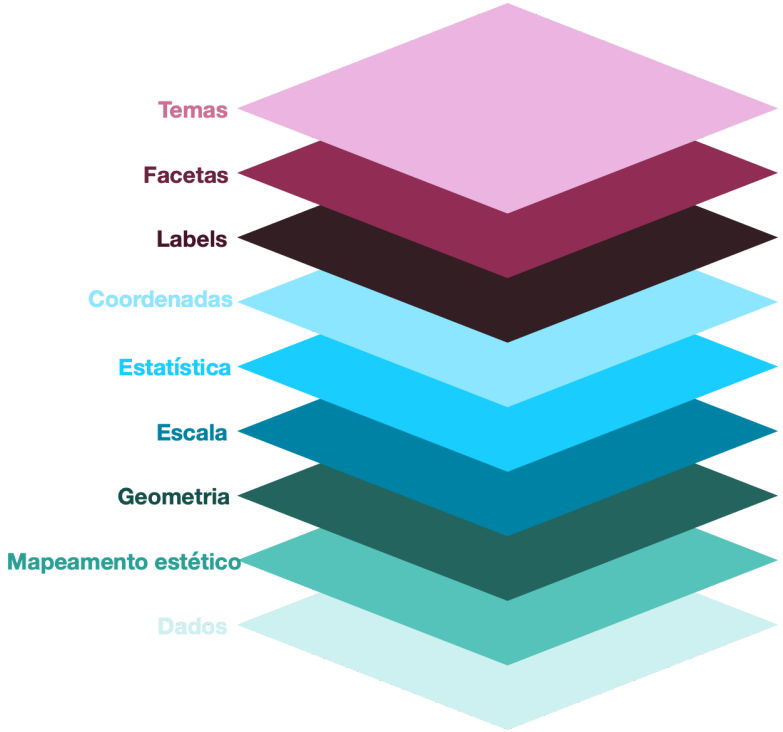
- **Dados**: conjunto de dados que serão utilizados para criar o gráfico.
- **Mapeamento estético**: mapeamento dos dados para os elementos visuais do gráfico.
- **Geometria**: forma geométrica que será utilizada para representar os dados.
- **Escala**: transformação dos dados em escalas (log, sqrt, etc).
- **Estatística**: transformação dos dados em estatísticas (média, mediana, etc).
- **Coordenadas**: sistema de coordenadas que será utilizado para representar os dados.
- **Labels**: rótulos dos eixos.
- **Facetas**: subdivisões dos dados em painéis.
- **Temas**: aparência visual do gráfico.



# Exemplo

- **Dados:** `mpg`
- **Mapeamento estético:** `x = displ`, `y = hwy`
- **Geometria:** `geom_point()`
- **Escala:** `scale_x_continuous()`, `scale_y_continuous()`
- **Estatística:** `stat_summary()`
- **Coordenadas:** `coord_flip()`
- **Labels:** `xlab()`, `ylab()`
- **Facetas:** `facet_wrap()`
- **Temas:** `theme_bw()`

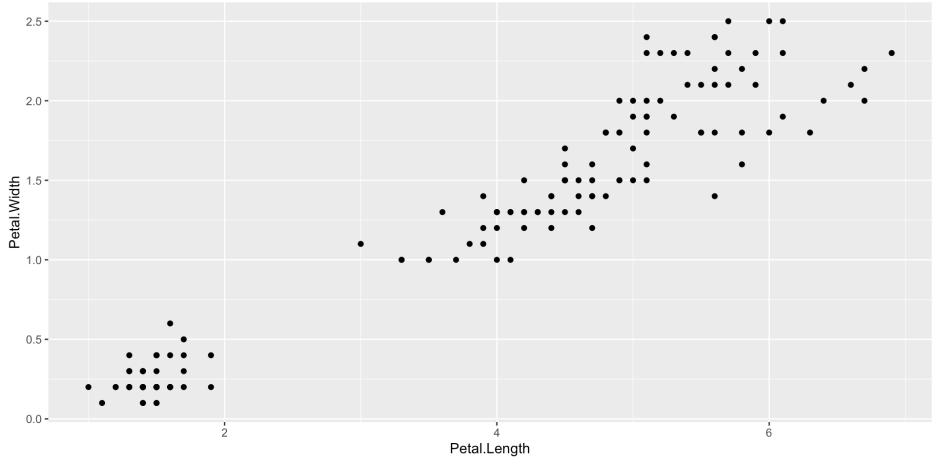
```
ggplot(data = mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  scale_x_continuous() +
  scale_y_continuous() +
  stat_summary() +
  coord_flip() +
  xlab("Engine displacement") +
  ylab("Highway miles per gallon") +
  facet_wrap(~class) +
  theme_bw()
```



# Gráfico de dispersão

- O gráfico de dispersão é utilizado para visualizar a relação entre duas variáveis **quantitativas**.
- A geometria `geom_point()` é utilizada para criar um gráfico de dispersão.
- Vamos utilizar o banco `iris` para criar um gráfico de dispersão entre a altura e o comprimento das pétalas.

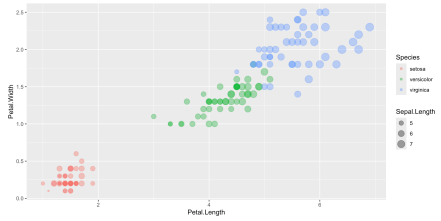
```
iris %>%
  ggplot() +
  aes(x = Petal.Length, y = Petal.Width) +
  geom_point()
```



# Gráfico de dispersão

- Para colorir os pontos de acordo com o gênero dos personagens, utilizamos o argumento `color` dentro da função `aes()`.
- Para aumentar o tamanho dos pontos de acordo com alguma variável, utilizamos o argumento `size` dentro da função `aes()`.
- Para reduzirmos a quantidade de pontos um em cima do outro, podemos aumentar a transparência dos pontos utilizando o argumento `alpha` dentro da geometria. O valor de `alpha` varia de 0 a 1, sendo 0 totalmente transparente e 1 totalmente opaco.

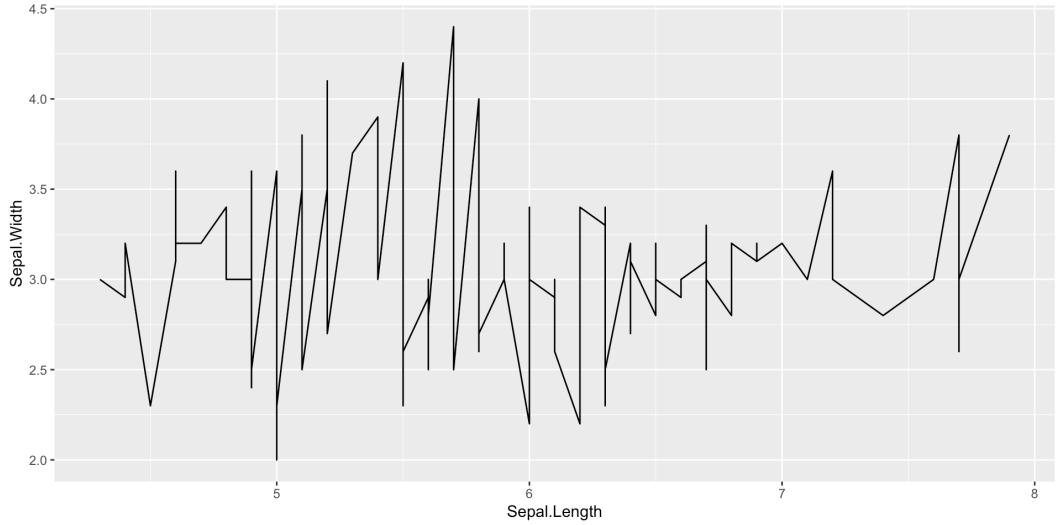
```
iris %>%
  ggplot() +
  aes(x = Petal.Length,
      y = Petal.Width,
      color = Species,
      size = Sepal.Length) +
  geom_point(alpha = 0.4, pch = "circle")
```



# Gráfico de linhas

- O gráfico de linhas é utilizado para visualizar a relação entre duas variáveis **quantitativas**.
- A geometria `geom_line()` é utilizada para criar um gráfico de linhas.
- Utilizando o mesmo banco de dados, vamos criar um gráfico de linhas para visualizar a relação entre o comprimento e a largura das sépalas.

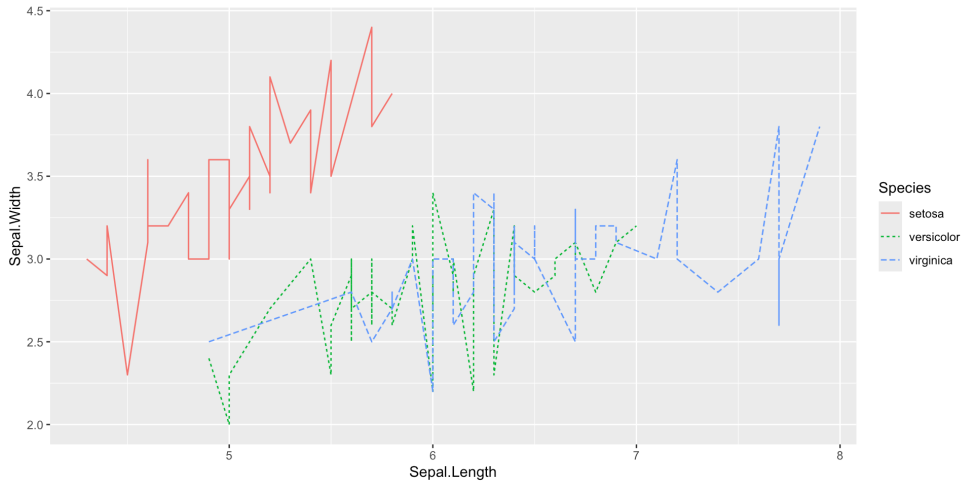
```
iris %>%
  ggplot() +
  aes(x = Sepal.Length, y = Sepal.Width) +
  geom_line()
```



# Gráfico de linhas

- Para colorir as linhas de acordo com o gênero dos personagens, utilizamos o argumento `color` dentro da função `aes()`.
- Para ajustarmos o tipo de linha, utilizamos o argumento `linetype` dentro da função `aes()`.

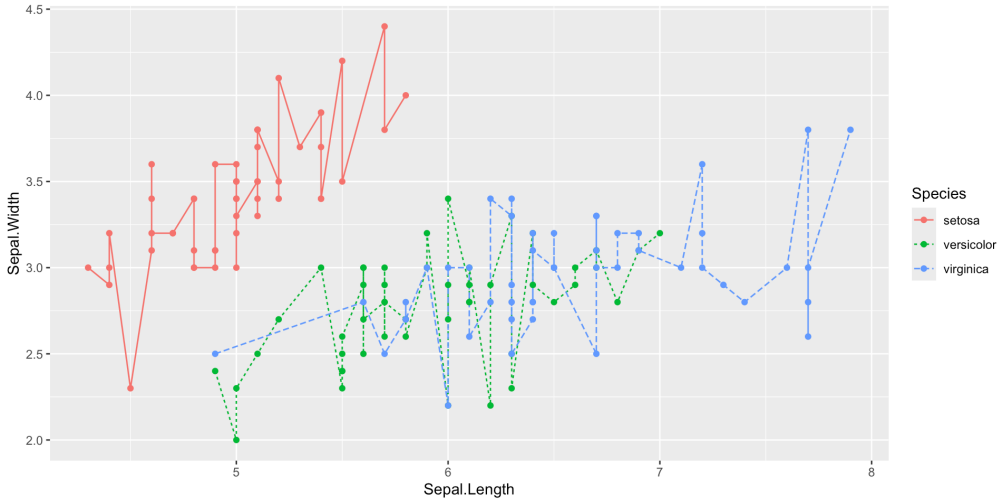
```
iris %>%
  ggplot() +
  aes(x = Sepal.Length,
      y = Sepal.Width,
      color = Species,
      linetype = Species) +
  geom_line()
```



# Combinando geometrias

- Podemos combinar diferentes geometrias em um mesmo gráfico.
- Por exemplo, podemos incluir no gráfico de linhas os pontos correspondentes a cada observação.

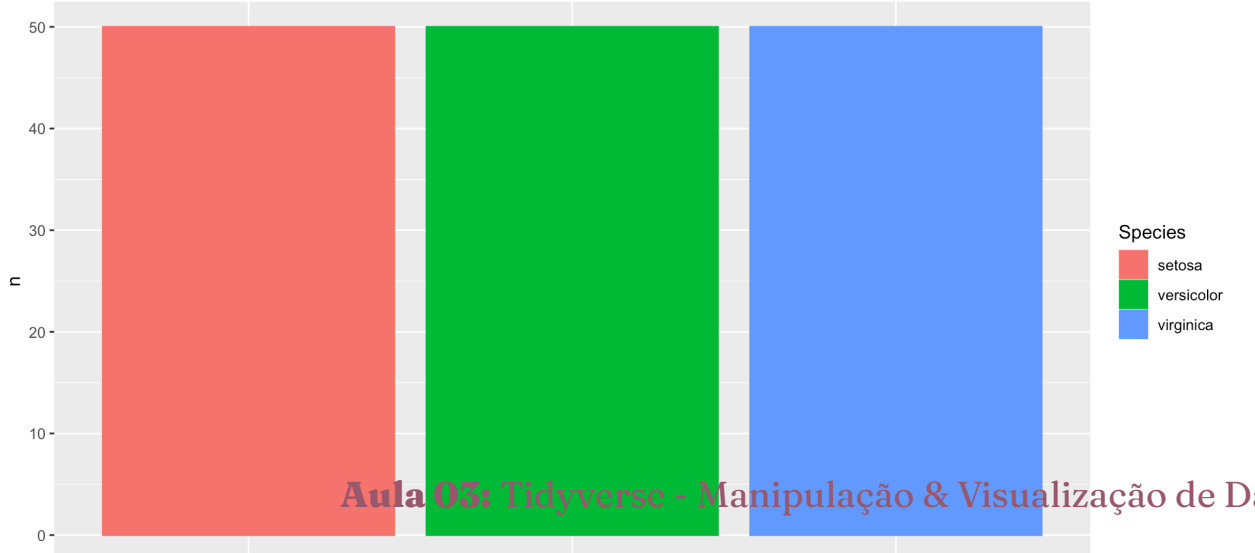
```
iris %>%
  ggplot() +
  aes(x = Sepal.Length,
      y = Sepal.Width,
      color = Species,
      linetype = Species) +
  geom_line() +
  geom_point()
```



# Gráfico de barras

- O gráfico de barras é utilizado para visualizar a relação entre uma variável **quantitativa** e uma **categórica**.
- A geometria `geom_bar()` e `geom_col` podem ser utilizadas para criar um gráfico de barras.

```
iris %>%
  count(Species) %>%
  ggplot() +
  aes(x = Species,
      y = n,
      color = Species,
fill = Species) +
  geom_bar(stat = "identity")
```



# Histograma

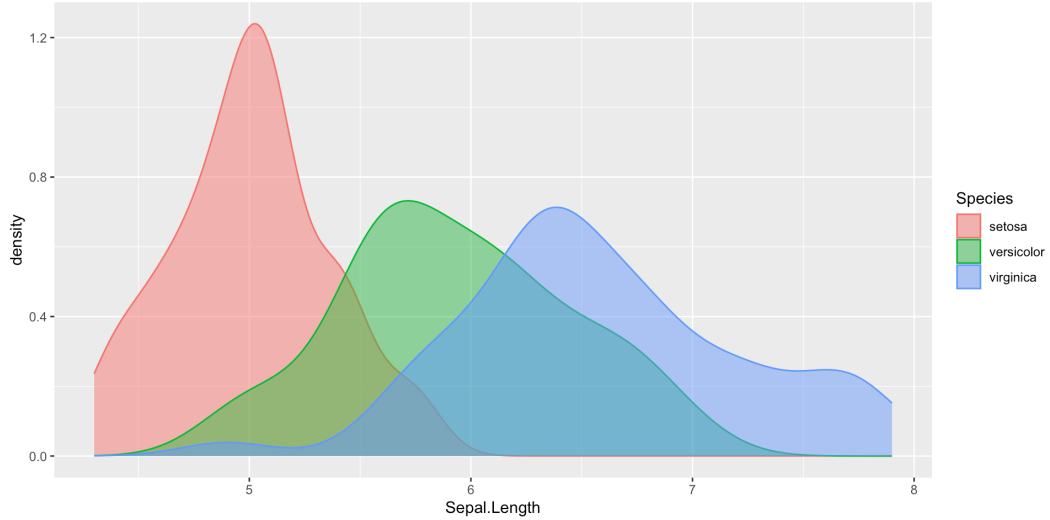
- O histograma é utilizado para visualizar a distribuição de uma variável **quantitativa**.
- A geometria `geom_histogram()` é utilizada para criar um histograma.

```
iris %>%
  ggplot() +
  aes(x = Sepal.Length) +
  geom_histogram(color = "red",
                 fill = "salmon",
                 bins = 30)
```

# Gráfico de densidade

- O gráfico de densidade é utilizado para visualizar a distribuição de uma variável **quantitativa**.
- A geometria `geom_density()` é utilizada para criar um gráfico de densidade.

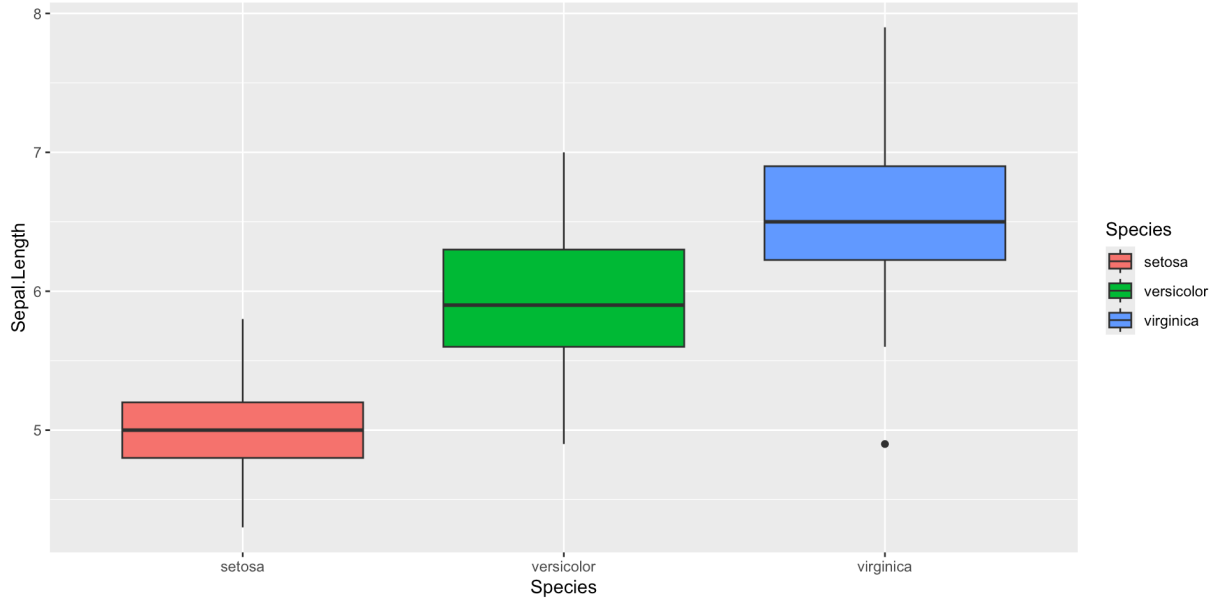
```
iris %>%
  ggplot() +
  aes(x = Sepal.Length,
      color = Species,
      fill = Species) +
  geom_density(alpha = 0.5)
```



# Boxplot e Gráfico de violino

- O boxplot e o gráfico de violino são utilizados para visualizar a distribuição de uma variável **quantitativa** de acordo com uma variável **categórica**, ou apenas a distribuição de uma variável **quantitativa**.
- A geometria `geom_boxplot()` é utilizada para criar um boxplot.

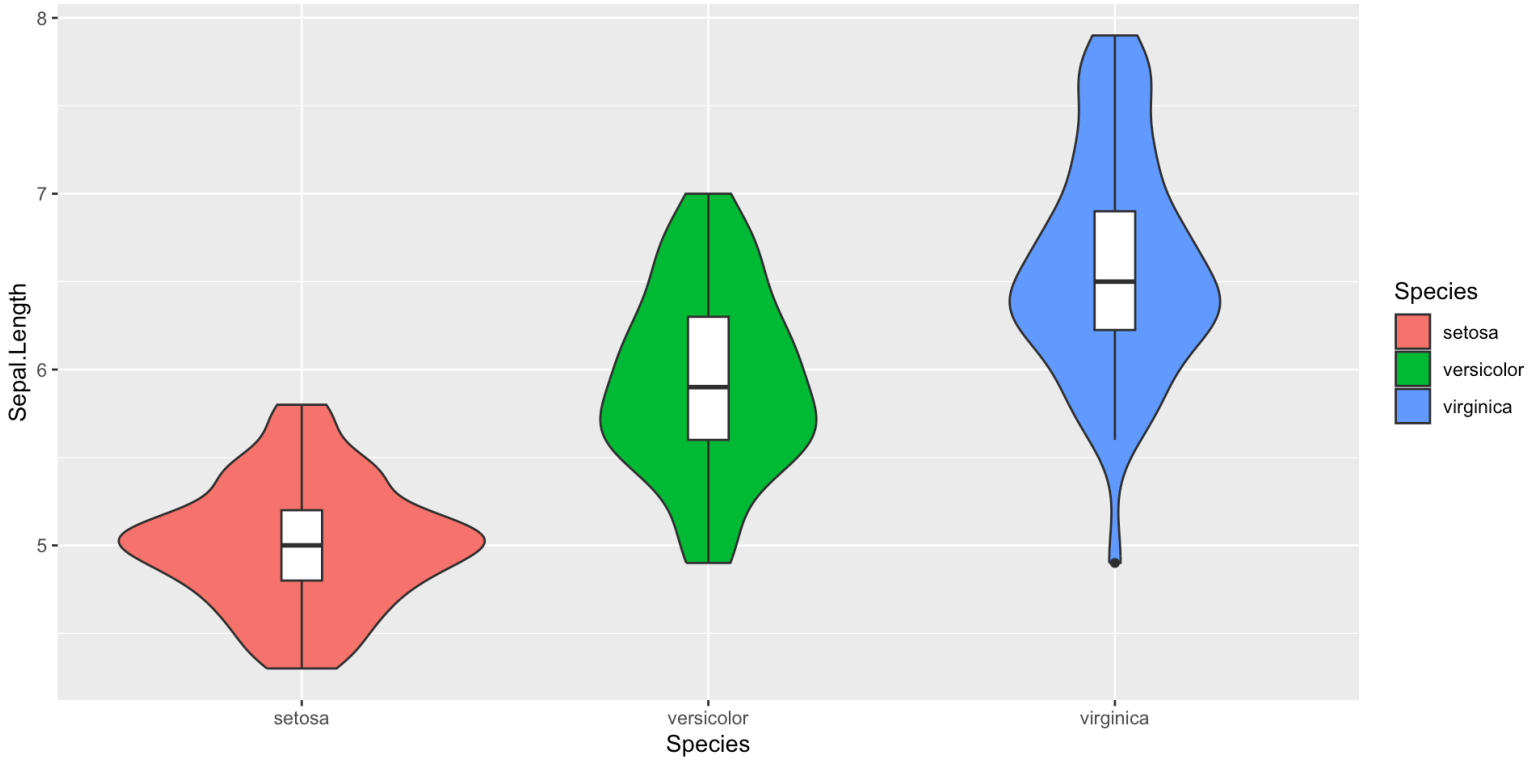
```
iris %>%
  ggplot() +
  aes(x = Species, y = Sepal.Length, fill = Species) +
  geom_boxplot()
```



# Gráfico de Violino

- A geometria `geom_violin()` é utilizada para criar um gráfico de violino.

```
iris %>%
  ggplot() +
  aes(x = Species, y = Sepal.Length, fill = Species) +
  geom_violin() +
  geom_boxplot(width = 0.1, fill = "white")
```



# Aprimore seu gráfico

- Altere os títulos dos eixos x e y, o título do gráfico e das legendas utilizando as funções `labs()`.
- Alterar a escala do eixo x ou y utilizando as funções `scale_x_continuous()` e `scale_y_continuous()`.
- Mudar a posição das legendas utilizando a função `theme()`.
- Mudar a cor do fundo do gráfico utilizando a função `theme()`.

```
iris %>%
  ggplot() +
  aes(x = Species, y = Sepal.Length, fill = Species) +
  geom_violin() +
  geom_boxplot(width = 0.1, fill = "white") +
  labs(title = "Distribuição do comprimento das sépalas por espécie",
       x = "Espécie",
       y = "Comprimento das sépalas") +
  scale_y_continuous(limits = c(4, 8)) +
  theme(legend.position = "bottom",
        plot.background = element_rect(fill = "lightblue"),
panel.background = element_rect(fill = "lightblue"),
panel.grid = element_line(color = "blue"),
        legend.background = element_rect(fill = "lightblue"),
        legend.title = element_text(color = "blue",
```



# Exercícios

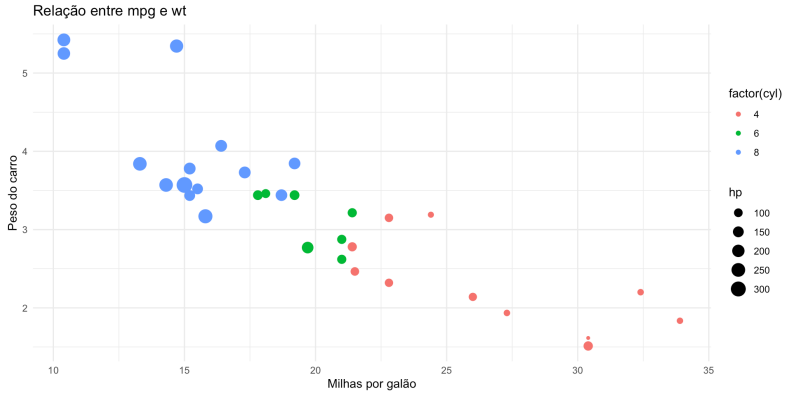
1. Utilize o banco de dados `mtcars` para criar um gráfico de dispersão entre as variáveis `mpg` e `wt`. Colora os pontos de acordo com a variável `cyl` e ajuste o tamanho dos pontos de acordo com a variável `hp`.
2. Utilize o banco de dados `mtcars` para criar um gráfico de linhas entre as variáveis `mpg` e `wt`. Colora as linhas de acordo com a variável `cyl` e ajuste o tipo de linha de acordo com a variável `cyl`.
3. Utilize o banco de dados `mtcars` para criar um gráfico de barras para visualizar a relação entre a variável `cyl` e a variável `mpg`.
4. Utilize o banco de dados `mtcars` para criar um histograma para visualizar a distribuição da variável `mpg`.
5. Utilize o banco de dados `mtcars` para criar um gráfico de densidade para visualizar a distribuição da variável `mpg`.
6. Utilize o banco de dados `mtcars` para criar um boxplot para visualizar a distribuição da variável `mpg` de acordo com a variável `cyl`.
7. Utilize o banco de dados `mtcars` para criar um gráfico de violino para visualizar a distribuição da variável `mpg` de acordo com a variável `cyl`.



# Soluções

1. Utilize o banco de dados `mtcars` para criar um gráfico de dispersão entre as variáveis `mpg` e `wt`. Colora os pontos de acordo com a variável `cyl` e ajuste o tamanho dos pontos de acordo com a variável `hp`.

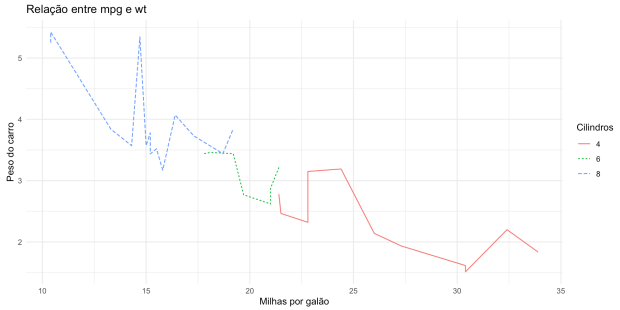
```
mtcars %>%
  ggplot() +
  aes(x = mpg,
      y = wt,
      color = factor(cyl),
      size = hp) +
  geom_point() +
  labs(title = "Relação entre mpg e wt",
       x = "Milhas por galão",
       y = "Peso do carro") +
  theme_minimal()
```



# Soluções

2. Utilize o banco de dados `mtcars` para criar um gráfico de linhas entre as variáveis `mpg` e `wt`. Colora as linhas de acordo com a variável `cyl` e ajuste o tipo de linha de acordo com a variável `cyl`.

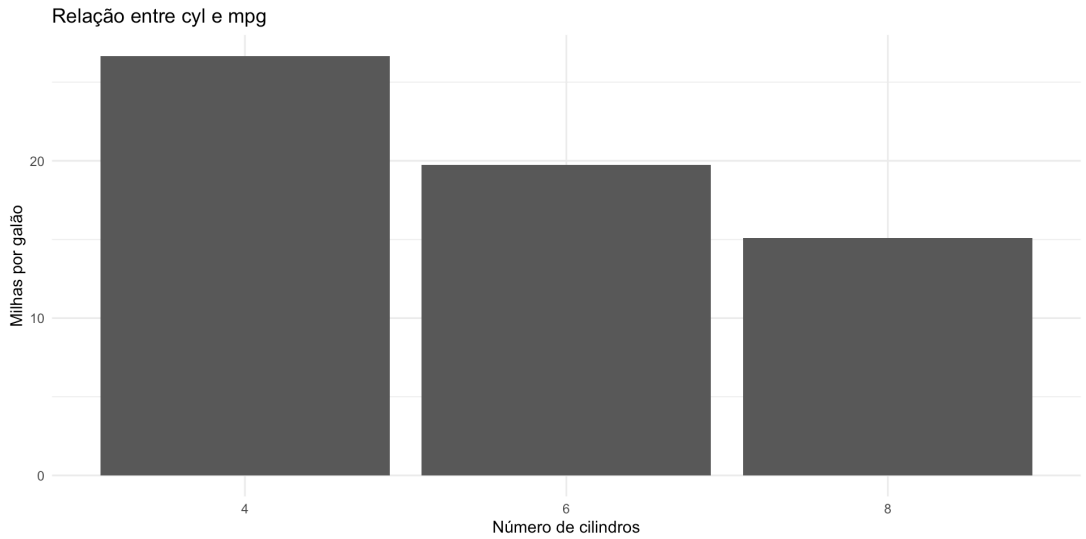
```
mtcars %>%
  ggplot() +
  aes(x = mpg,
      y = wt,
      color = factor(cyl),
      linetype = factor(cyl)) +
  geom_line() +
  labs(title = "Relação entre mpg e wt",
       x = "Milhas por galão",
       y = "Peso do carro",
       color = "Cilindros",
       linetype = "Cilindros") +
  theme_minimal()
```



# Soluções

3. Utilize o banco de dados `mtcars` para criar um gráfico de barras para visualizar a relação entre a variável `cyl` e a variável `mpg`.

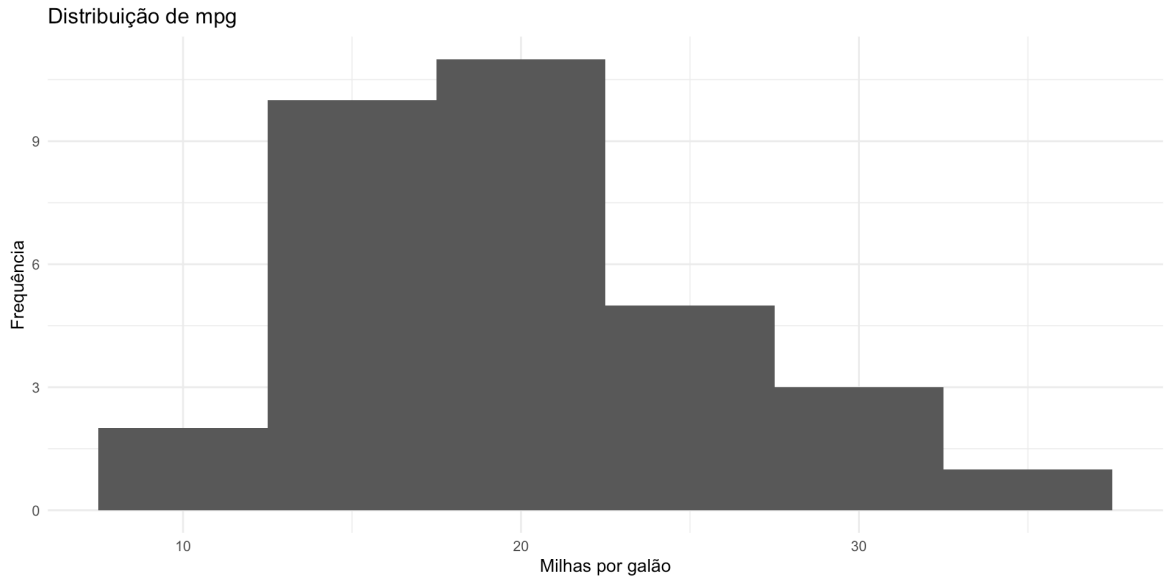
```
mtcars %>%
  ggplot() +
  aes(x = factor(cyl),
      y = mpg) +
  geom_bar(stat = "summary", fun = "mean") +
  labs(title = "Relação entre cyl e mpg",
       x = "Número de cilindros",
       y = "Milhas por galão") +
  theme_minimal()
```



# Soluções

4. Utilize o banco de dados `mtcars` para criar um histograma para visualizar a distribuição da variável `mpg`.

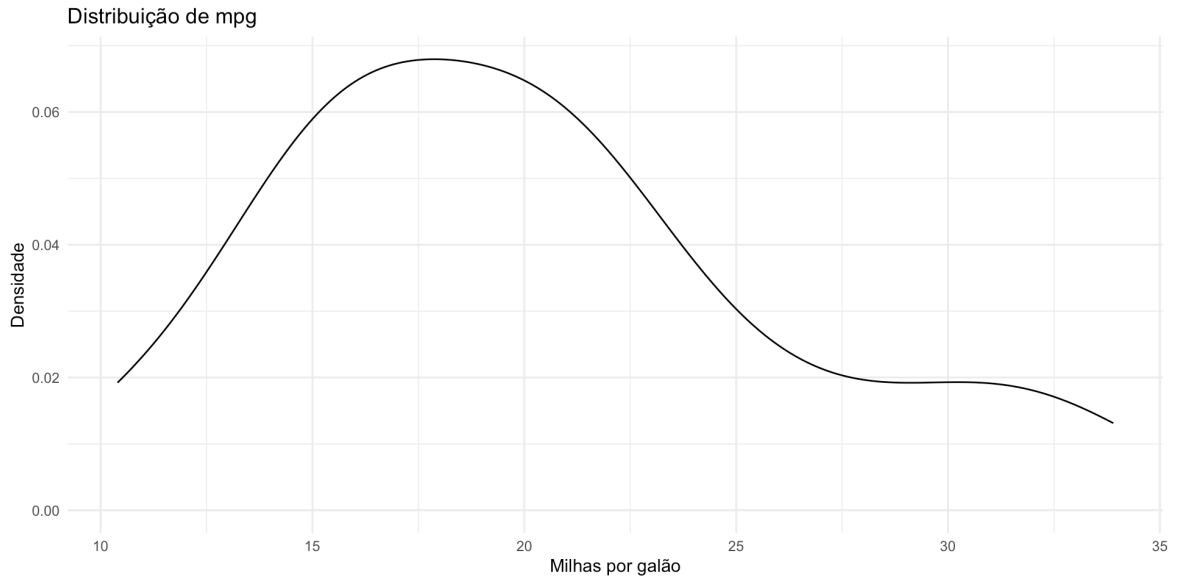
```
mtcars %>%
  ggplot() +
  aes(x = mpg) +
  geom_histogram(binwidth = 5) +
  labs(title = "Distribuição de mpg",
       x = "Milhas por galão",
       y = "Frequência") +
  theme_minimal()
```



# Soluções

5. Utilize o banco de dados `mtcars` para criar um gráfico de densidade para visualizar a distribuição da variável `mpg`.

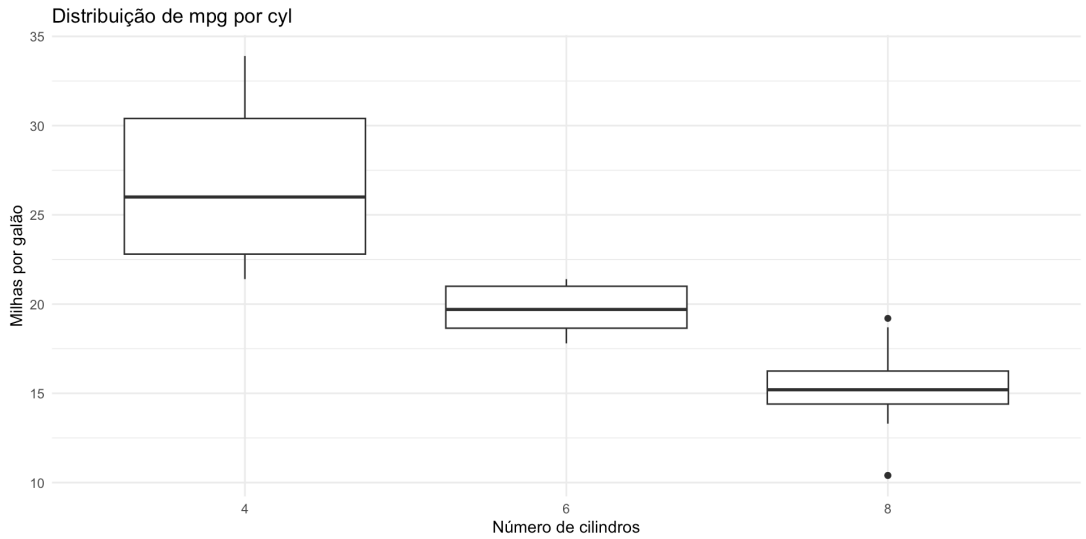
```
mtcars %>%
  ggplot() +
  aes(x = mpg) +
  geom_density() +
  labs(title = "Distribuição de mpg",
       x = "Milhas por galão",
       y = "Densidade") +
  theme_minimal()
```



# Soluções

6. Utilize o banco de dados `mtcars` para criar um boxplot para visualizar a distribuição da variável `mpg` de acordo com a variável `cyl`.

```
mtcars %>%
  ggplot() +
  aes(x = factor(cyl),
      y = mpg) +
  geom_boxplot() +
  labs(title = "Distribuição de mpg por cyl",
       x = "Número de cilindros",
       y = "Milhas por galão") +
  theme_minimal()
```



# Soluções

7. Utilize o banco de dados `mtcars` para criar um gráfico de violino para visualizar a distribuição da variável `mpg` de acordo com a variável `cyl`.

```
mtcars %>%
  ggplot() +
  aes(x = factor(cyl),
      y = mpg) +
  geom_violin() +
  labs(title = "Distribuição de mpg por cyl",
       x = "Número de cilindros",
       y = "Milhas por galão") +
  theme_minimal()
```