

Visualização de dados



fevereiro de 2023

Sobre a Curso-R

A empresa

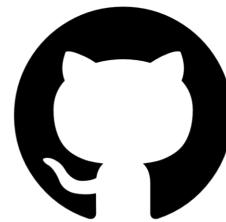


Filosofia de código aberto!

Livros



Material dos cursos



Confira o
nossa GitHub

Lives



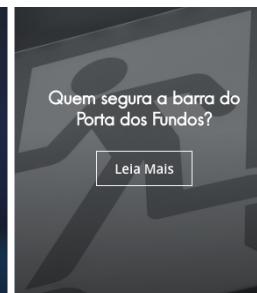
TODA
QUARTA,
ÀS 20H

Blog



Predições XGBoost
diretamente pelo SQL

[Leia Mais](#)



Quem segura a barra do
Porta dos Fundos?

[Leia Mais](#)



Explorando a base de
dados de CNPJ da
Receita Federal

[Leia Mais](#)

Nossos cursos

PROGRAMAÇÃO PARA CIÊNCIA DE DADOS

Introdução a programação com R
R para Ciência de Dados I
R para Ciência de Dados II
Pacotes
Python para quem usa R

DASHBOARDS E VISUALIZAÇÃO DE DADOS

Visualização de dados
Relatórios e apresentações
Dashboards I
Deploy
Dashboards II

WEB SCRAPING

Web scraping

MODELAGEM DE DADOS

Modelos Lineares
Introdução ao Machine Learning
Séries Temporais
Não supervisionado

Sobre o curso

Dinâmica curso

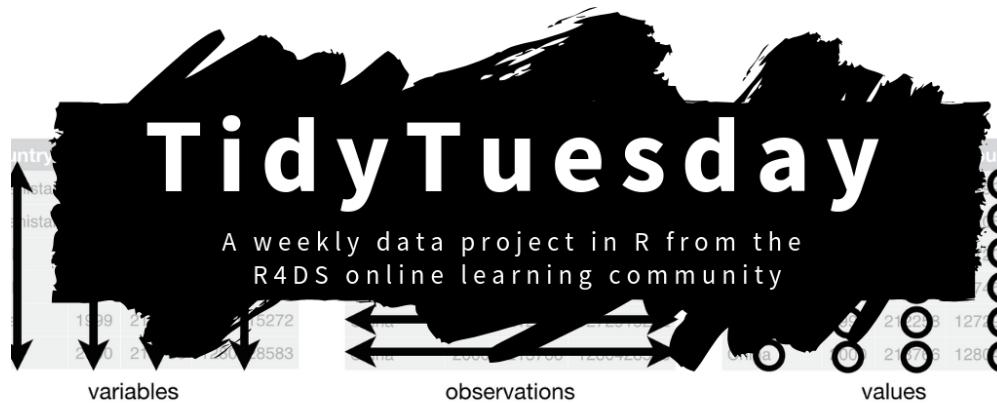
- As aulas terão uma seção teórica, de exposição de conceitos, e prática, de aplicação de conceitos.
- O objetivo dos exercícios é gerar dúvidas. **Com exceção do trabalho final, nenhum exercício precisa ser entregue.**
- O certificado será emitido mediante uma **entrega final**, a ser detalhada a seguir.
- Haverá monitoria para esclarecimento de dúvidas sempre 30 minutos antes do início das aulas.
- Usaremos os últimos minutos de cada aula para tirar dúvidas do conteúdo apresentado. Não haverá plantão de dúvidas pós aula.
- A gravação das aulas ficará disponível no Google Classroom por 1 ano após o final do curso.

Dinâmica das aulas

- Mande dúvidas e comentários no chat em qualquer momento.
- Para falar, levante a mão.
- Algumas dúvidas serão respondidas na hora. Outras serão respondidas mais tarde na própria aula ou em aulas futuras.

Trabalho final

Para o TCC do curso, entregará um TidyTuesday!



O TidyTuesday é um evento semanal criado para engajar a comunidade no uso do R para análise de dados. Se quiser conhecer mais, [siga a hashtag #tidytuesday no Twitter!](#)

Mais informações [aqui](#).

Trabalho final

O resultado deverá ser entregue em um **arquivo zip** contendo as visualizações e códigos. A submissão deve ser feita pelo classroom, subindo um arquivo .zip contendo

- Os **códigos em R** utilizados para gerar os gráficos.
- Os **gráficos em imagens** (.png, .jpeg, .gif etc)
- Um arquivo **README.md** contando qual foi sua ideia e o processo de construção das visualizações.

Observações

- Não é necessário que a **base de dados** esteja no repositório, já que ela pode ser lida diretamente da internet.
- **RMarkdown/Quarto.** Se você estiver confortável com relatórios reproduutíveis, pode entregar o trabalho em um relatório ou apresentação à sua escolha. Nesse caso, envie output final (em HTML, PDF, Word, etc) no lugar das imagens.
- **GitHub.** Se você estiver confortável com Git/GitHub, pode entregar o link do repositório no lugar do zip.

Tire suas dúvidas

- **Não fique com dúvidas.**
- Fora do horário de aula ou monitoria:
 - envie suas perguntas gerais **sobre o curso** no Classroom.
 - envie preferencialmente suas perguntas **sobre R** no [nossa discussão](#).
- Saber fazer a pergunta certa vai te ajudar bastante nos estudos de programação. [Veja aqui dicas de como fazer uma boa pergunta.](#)

Introdução

Objetivos de aprendizagem

- Compreender o papel da visualização em um projeto de ciência de dados.
- Compreender as diferenças entre análise exploratória e análise descritiva.
- Entender o funcionamento básico do pacote {ggplot2}.
- Utilizar o {ggplot2} em um problema concreto.

Ficaremos com esses conteúdos por 2 aulas!

O que é visualização de dados?

- É a representação de dados em gráficos, tabelas e diagramas que podem ser interpretados por pessoas.
- É uma área interdisciplinar, misturando estatística, arte e comunicação.
- É uma parte da área de *data storytelling*, que envolve organizar todos os resultados de uma análise de dados em uma ordem lógica para comunicar de forma efetiva com a audiência.



Ilustração de [Allison Horst](#)

Por que fazer visualizações de dados?

- Visualizações estão presentes na grande maioria dos projetos de ciência de dados.
- É a parte mais acessível da ciência de dados do ponto de vista de quem lê. Mostrar uma visualização costuma ser mais efetivo do que a saída de um modelo.
- É uma das partes mais difíceis de automatizar da ciência de dados. Uma carreira em *dataviz* dificilmente ficará obsoleta.

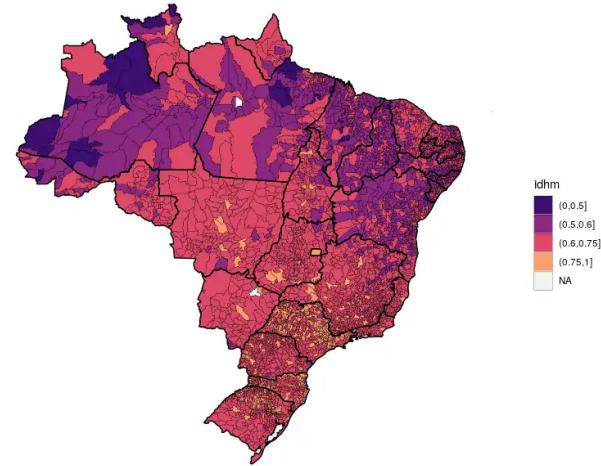


Imagen do [blog da Curso-R](#)

Para que servem as visualizações?

- Uma base de dados contém toda a informação que precisamos.
- No entanto, não somos capazes de tirar conclusões apenas olhando essas bases.
- Por isso, é necessário resumir esses dados em estatísticas.
- Nem sempre as estatísticas (os números) são úteis para uma comunicação efetiva... Por isso, faz sentido mostrá-las usando formas, cores e outros elementos que facilitam a absorção da informação pelas pessoas.
- Para o [Hadley Wickham](#), visualizar dados serve para ***surpreender***.

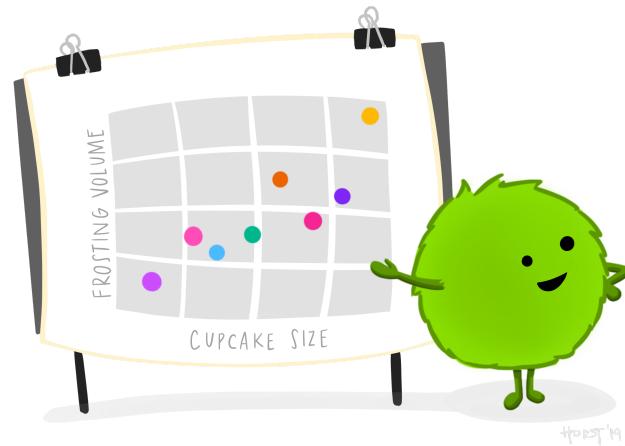


Ilustração de [Allison Horst](#)

Gráficos bons e ruins

Gráficos para evitar

- Barras que não começam no zero
- Gráficos de pizza (discutível!)
- ...

Gráficos para tomar cuidado

- Gráficos com dois eixos
- Gráficos pouco conhecidos
- ...

Ferramental

Em que momentos utilizamos?

Importar



Arrumar →

(Armazenar os dados
consistentemente)

Transformar

(Criar novas variáveis e
agregações)

Visualizar

(Surpreende, mas não é
escalável)

Modelar

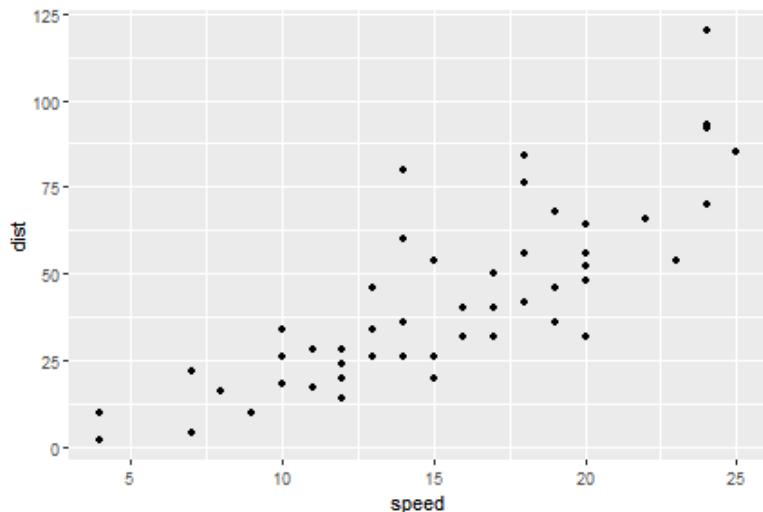
(É escalável, mas não
surpreende)

Comunicar

Automatizar

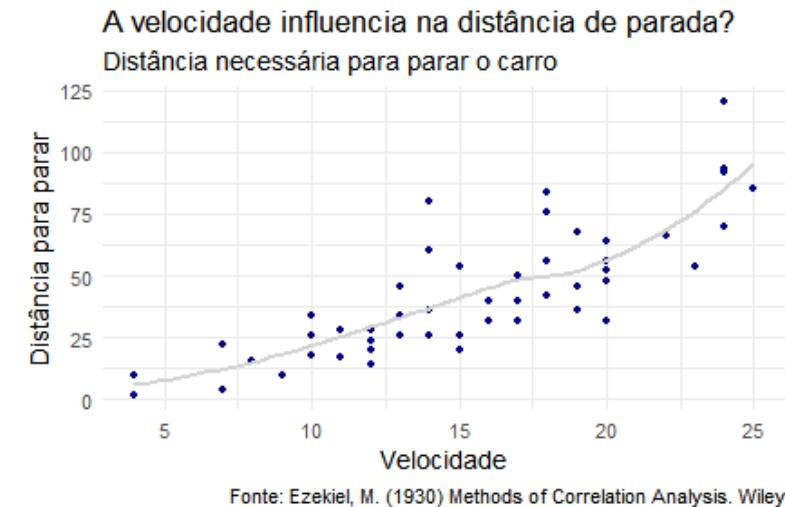
Análise exploratória

- É um trabalho de **investigação** de dados
- A ferramenta: precisa ser **rápida** de programar
- O objetivo é **aprender**



Análise descritiva

- É um trabalho de **otimização visual**
- A ferramenta: precisa ser **customizável**
- O objetivo é **comunicar**



Fonte: Ezekiel, M. (1930) Methods of Correlation Analysis. Wiley

O `{ggplot2}` permite fazer as duas coisas!

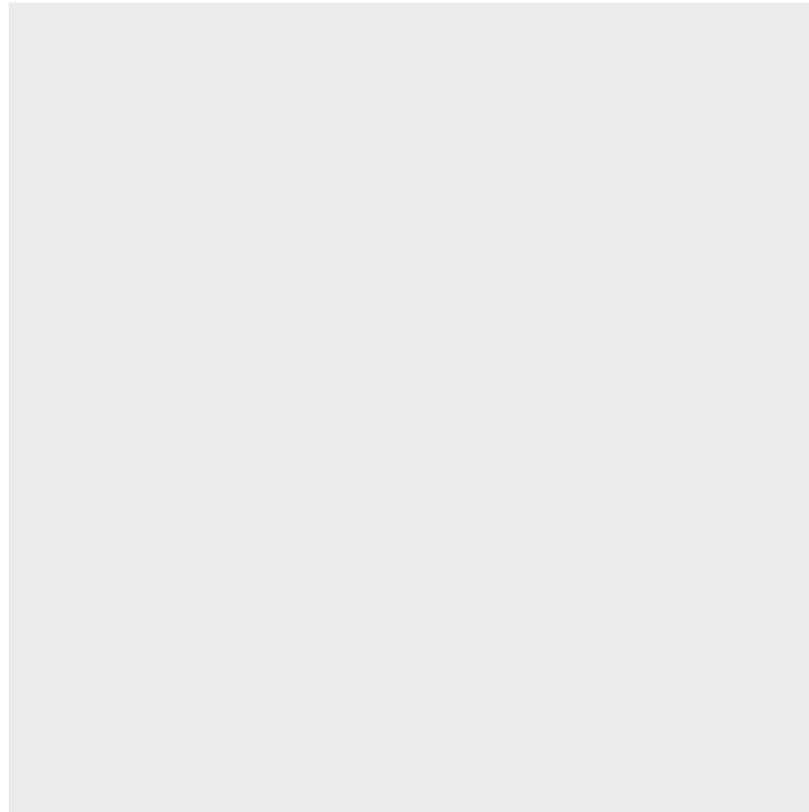
Gramática de gráficos *em camadas*

O pacote `{ggplot2}` segue duas filosofias que nos ajudam a entender o processo de construção dos gráficos:

1. Um gráfico estatístico é uma representação visual dos dados por meio de atributos estéticos (posição, cor, forma, tamanho, ...) de formas geométricas (pontos, linhas, barras, ...). [The Grammar of Graphics](#).
2. Um gráfico pode ser construído em camadas (um gráfico é a sobreposição de elementos visuais). [A layered grammar of graphics](#).

Camadas

Para construir um gráfico, começamos com o *canvas*. A função `ggplot()` cria a primeira camada do nosso gráfico: uma tela em branco (cinza).



Camadas

Canvas

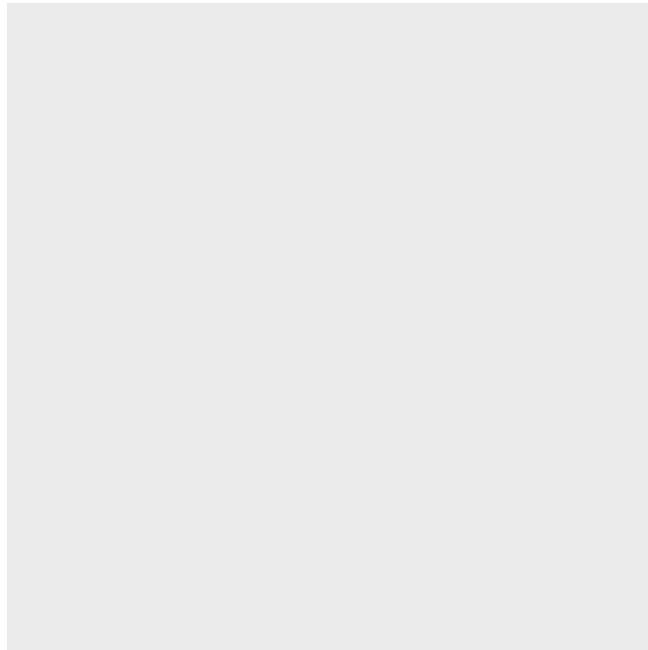
Canvas (R)

Eixos

Eixos (R)

Geometria

Geometria (R)

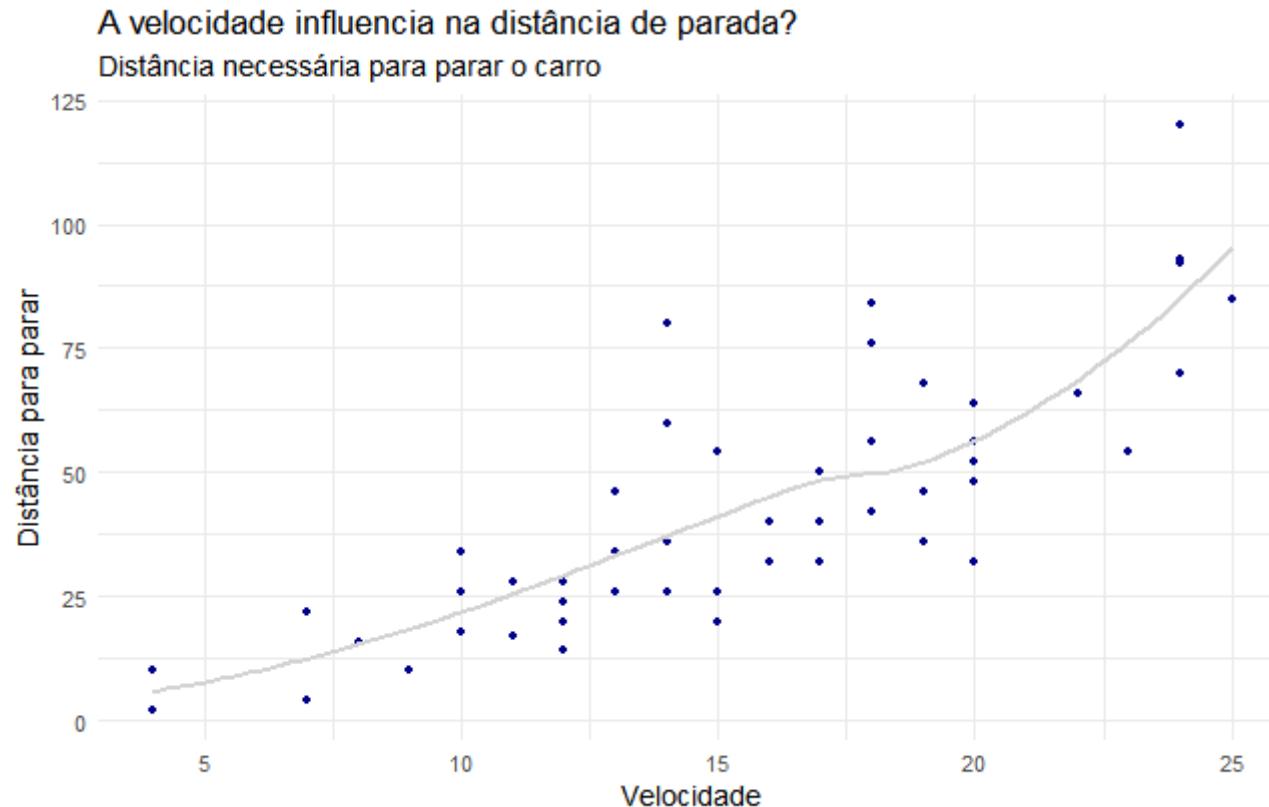


Camadas

Depois, podemos trabalhar a estética com temas e detalhamentos.

Completo

Completo (R)



Fonte: Ezekiel, M. (1930) Methods of Correlation Analysis. Wiley

Curiosidade: por que o +?

O `{ggplot2}`, diferentemente dos outros pacotes do `tidyverse`, não usa o *pipe* (`|>` ou `|>` depois do R 4.1). Isso acontece pois o `{ggplot2}` surgiu **antes que o autor tomasse conhecimento do pipe**.



u/hadley • Commented on 7 anos ago



ggplot worked using function composition instead of addition. So instead of

```
ggplot(mtcars, aes(wt, mpg)) +  
  geom_point() +  
  geom_smooth()
```

You wrote something like

```
geom_smooth(geom_point(ggplot(mtcars, aes(wt, mpg))))
```

I changed the name of the package because changing the code dramatically would break a lot of people's code (which is ironic given that now making the tiniest change in ggplot2 affects more people than ever used ggplot).

An interesting historical note is that if I'd discovered the pipe earlier, there never would've been a ggplot2, because you could write ggplot graphics as

```
ggplot(mtcars, aes(wt, mpg)) %>%  
  geom_point() %>%  
  geom_smooth()
```

11 upvotes 0 replies

Dúvidas?

Data visualization with ggplot2 :: CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data set**, a **coordinate system**, and **geoms**—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOGRAPHICAL_FUNCTION> + <COORDINATE_FUNCTION> +
  <POSITION_FUNCTION> + <SCALE_FUNCTION> + <THEME_FUNCTION>
```

required

Not required, sensible defaults supplied

ggplot(data = mpg, aes(x = cyl, y = hwy)) Builds a plot that will finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes Common aesthetic values.

```
color and fill - string ("red", "#RRGGBB")
linetype - integer or string (0 = "blank", 1 = "solid",
2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash",
6 = "twodash")
lineend - string ("round", "butt", or "square")
linejoin - string ("round", "mitre", or "bevel")
size - integer (line width in mm)
shape - integer/shape name or a single character ("a")
```



Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank() and a + expand_limits()
Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1,
xend = long + 1), curvature = 1), x, yend, y, yend,
alpha, angle, color, curvature, linetype, size

a + geom_path(linewidth = "butt",
linejoin = "round", linemiter = 1)

x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)) - x, y, alpha,
color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat,
xmax = long + 1, ymax = lat + 1)), x, xmax, xmin,
ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900,
ymax = unemploy + 900)) - x, ymax, ymin,
alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend = lat + 1, xend = long + 1))
b + geom_spoke(aes(angle = 1:155, radius = 1))

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, group, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill, group, linetype, size

f + geom_violin(scale = "area")
x, y, alpha, color, fill, group, linetype, size, weight

both discrete

g <- ggplot(diamonds, aes(cut, color))
b + geom_count()
x, y, alpha, color, fill, shape, size, stroke

c + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy))
x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(ffill))
d + geom_bar()
```

x, alpha, color, fill, linetype, size, weight

sealsSz <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z))
x, y, alpha, color, fill, group, linetype, size, subgroup

TWO VARIABLES both continuous

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1,
nudge_y = -1), x, y, label, alpha, angle, color,
family, fontface, hjust, lineheight, size, vjust

e + geom_point()
x, y, alpha, color, fill, shape, size, stroke

e + geom_quartile()
x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl")
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm)
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cyl), nudge_x = 1,
nudge_y = -1), x, y, label, alpha, angle, color,
family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha,
color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill, group, linetype, size

f + geom_violin(scale = "area")
x, y, alpha, color, fill, group, linetype, size, weight

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
```

```
map <- map_data("state")
```

```
K <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map)

+ expand_limits(x = map\$x, y = map\$lat)

map_id, alpha, color, fill, linetype, size

l + geom_raster(aes(fill = z), vjust = 0.5,
interpolate = FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size, width



Qual gráfico escolher?

from Data to Viz

EXPLORE STORY ALL CAVEATS POSTER ABOUT CONTACT



from Data to Viz

From Data to Viz leads you to the most appropriate graph for your data. It links to the code to build it and lists common caveats you should avoid.

EXPLORE

27 / 110

Base de dados do exemplo

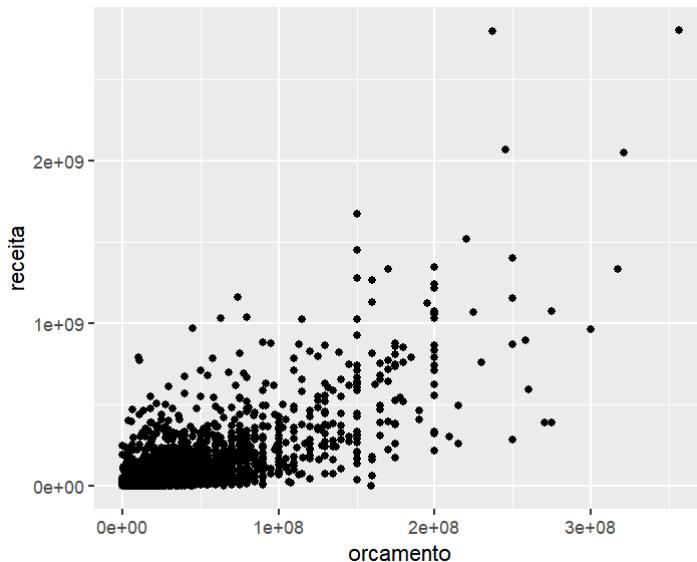
Para os slides a seguir, vamos usar a base do imdb. Atenção para a coluna lucro, que criamos fazendo a diferença entre receita e orçamento.

```
library(tidyverse)
imdb <- read_rds("dados/imdb.rds")
imdb <- imdb |> mutate(lucro = receita - orcamento)
```

Um gráfico de pontos (dispersão)

Podemos fazer um gráfico de dispersão da receita contra o orçamento dos filmes acrescentando a função `geom_point()` ao código anterior.

```
imdb |>  
  ggplot() +  
  geom_point(aes(x = orcamento, y = receita))
```



Muitos pontos para discutir:

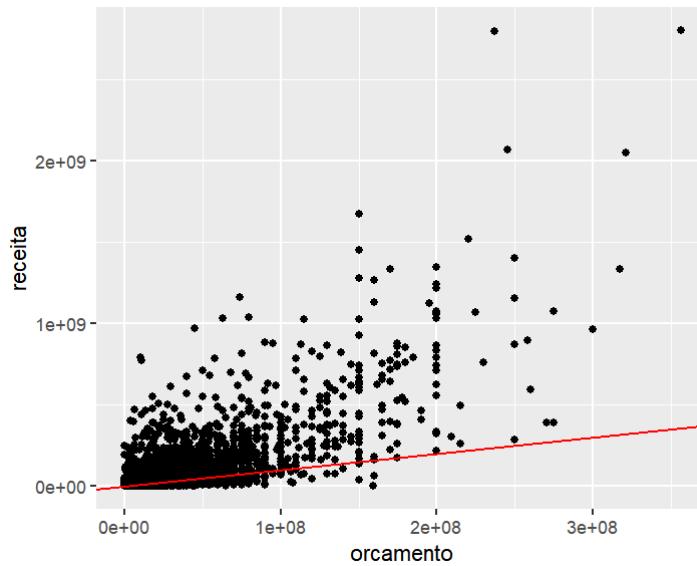
- Esse gráfico tem duas camadas: o canvas, gerado pela função `ggplot()`, e os pontos, gerado pela função `geom_point()`.
- Unimos as camadas de um `ggplot` usando um `+`. Sim, precisamos controlar a nossa vontade de colocar um `|>` em vez de `+`, e essa é uma fonte de erro bem comum. O motivo para precisarmos usar `+` em vez do `|>` é o pacote `ggplot` ter nascido primeiro que o pipe.
- A função `geom_point()` define que a forma geométrica (daí o prefixo `geom`) utilizada para representar os dados será pontos. Existe uma família de funções `geom`, sendo que cada uma vai representar uma forma geométrica diferente.
- O primeiro argumento de qualquer função `geom` é o `mapping`. Esse argumento serve para mapear os dados nos atributos estéticos da forma geométrica escolhida. Ele sempre receberá a função `aes()`. No código, nós omitimos o nome do argumento, mas poderíamos ter escrito `geom_point(mapping = aes(x = orcamento, y = receita))`.

- A função `aes()` serve para *mapear os dados aos elementos estéticos do gráfico*. Os argumentos dela vão sempre depender da forma geométrica que estamos utilizando. No caso de um gráfico de pontos, precisamos definir como as posições do eixo x e y serão construídas. No exemplo, a posição do ponto no eixo x será dada pela coluna `orcamento` e a posição do ponto no eixo y será dada pela coluna `receita`.
- O *warning* indica quantas observações (linhas) precisaram ser removidas, por não possuir informação de orçamento ou receita.
- Veremos nos próximos exemplos que será muito comum manipularmos a base (aplicarmos diversas funções do `dplyr`, por exemplo) antes de chamarmos a função `ggplot`.

O mapeamento das COLUNAS nas FORMAS GEOMÉTRICAS deve ser SEMPRE feito dentro da função `aes()`.

Vamos agora inserir um novo elemento visual ao gráfico: a reta $x = y$.

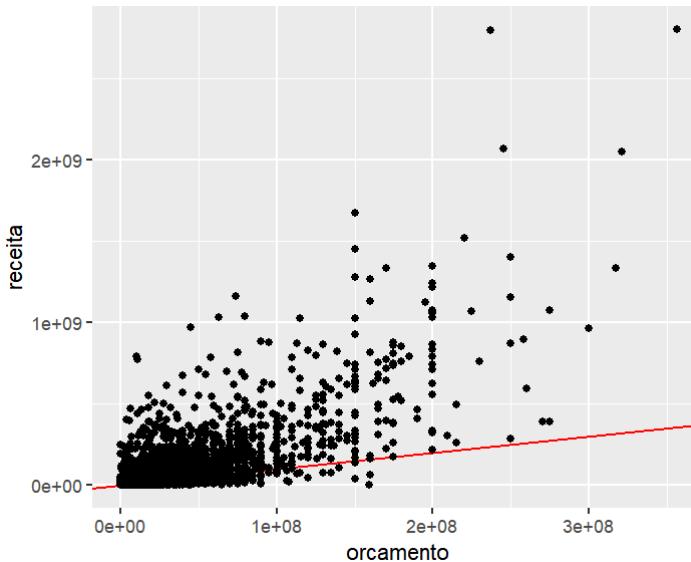
```
imdb |>  
  ggplot() +  
  geom_point(aes(x = orcamento, y = receita)) +  
  geom_abline(intercept = 0, slope = 1, color = "red")
```



- A reta $x = y$ é acrescentada ao gráfico pela função `geom_abline()`. Esse `geom` pode ser utilizado para desenhar qualquer reta do tipo $y = a + b * x$, sendo a o intercepto (*intercept*) da reta e b o seu coeficiente angular (*slope*).
- Essa reta nos permite observar o número de filmes que obtiveram lucro (pontos acima da reta) e aqueles que obtiveram prejuízo (pontos abaixo da reta).
- Como não estamos mapeando colunas a essa reta, não precisamos colocar os argumentos da função `geom_abline()` do `aes()`.

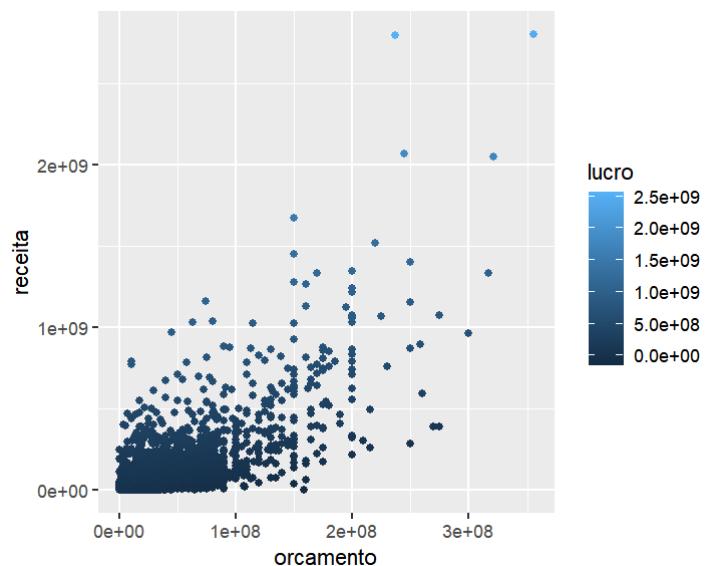
Veja como o ggplot realmente é construído por camadas. Agora, colocamos a camada da reta antes da camada dos pontos. Os pontos ficam em cima da reta.

```
imdb |>  
  ggplot() +  
  geom_abline(intercept = 0, slope = 1, color = "red") +  
  geom_point(aes(x = orcamento, y = receita))
```



Os atributos x e y são necessários para construirmos um gráfico de pontos. Outros atributos também podem ser mapeados em pontos, como a cor. Como a coluna `lucro` é numérica, um degradê de cores é criado para os pontos, a depender do lucro.

```
imdb |>  
  ggplot() +  
  geom_point(aes(x = orcamento, y = receita, color = lucro))
```



Poderíamos também classificar os filmes entre aqueles que lucraram ou não. Uma cor é atribuída a cada categoria.

```
imdb |>
  mutate(
    lucrou = ifelse(lucro <= 0, "Não", "Sim")
  ) |>
  ggplot() +
  geom_point(aes(x = orçamento, y = receita, color = lucrou))
```

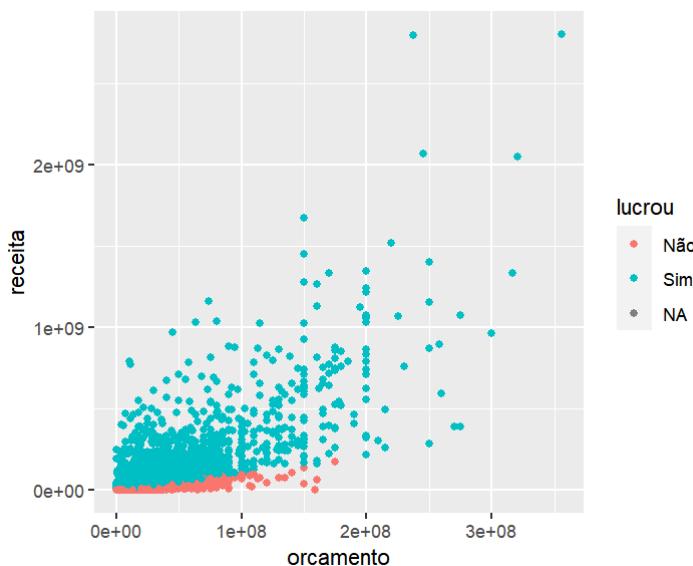


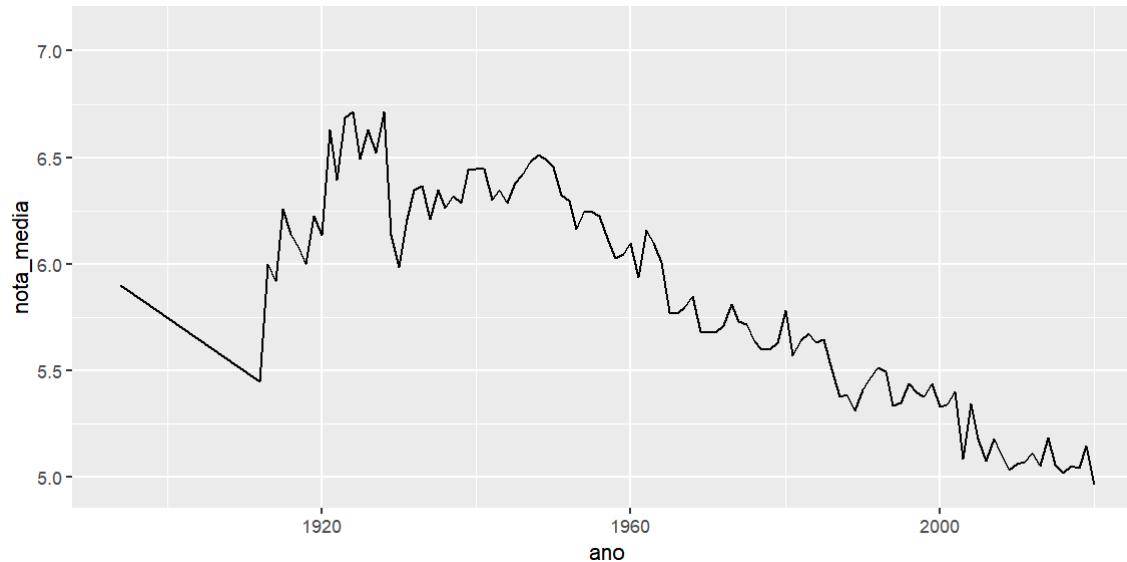
Gráfico de linhas

Utilizamos o `geom_line` para fazer gráficos de linhas. Gráficos de linha são muito utilizados para representar *séries temporais*, isto é, observações medidas repetidamente em intervalos (em geral) equidistantes de tempo.

Assim como nos gráficos de pontos, precisamos definir as posições x e y para construirmos gráficos de linhas.

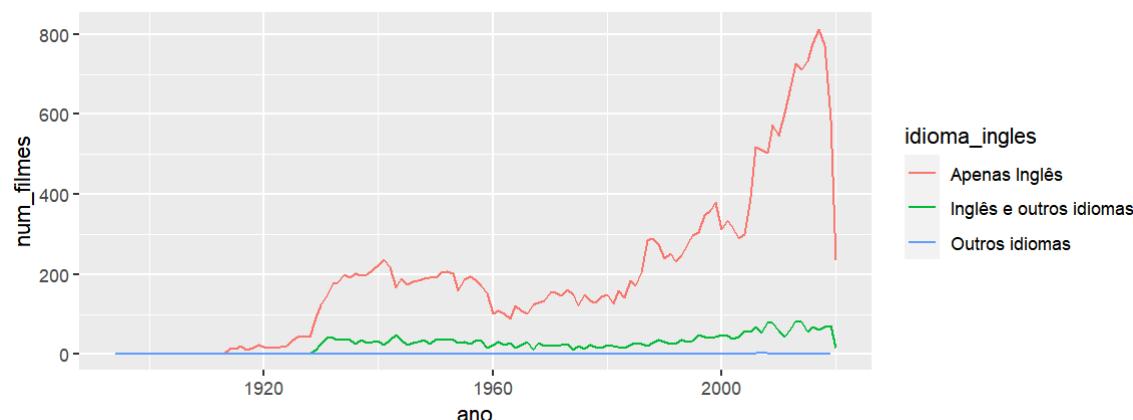
A seguir, construímos o gráfico das notas médias dos filmes produzidos em cada ano,

```
imdb |>  
  group_by(ano) |>  
  summarise(nota_media = mean(nota_imdb, na.rm = TRUE)) |>  
  ggplot() +  
  geom_line(aes(x = ano, y = nota_media))
```



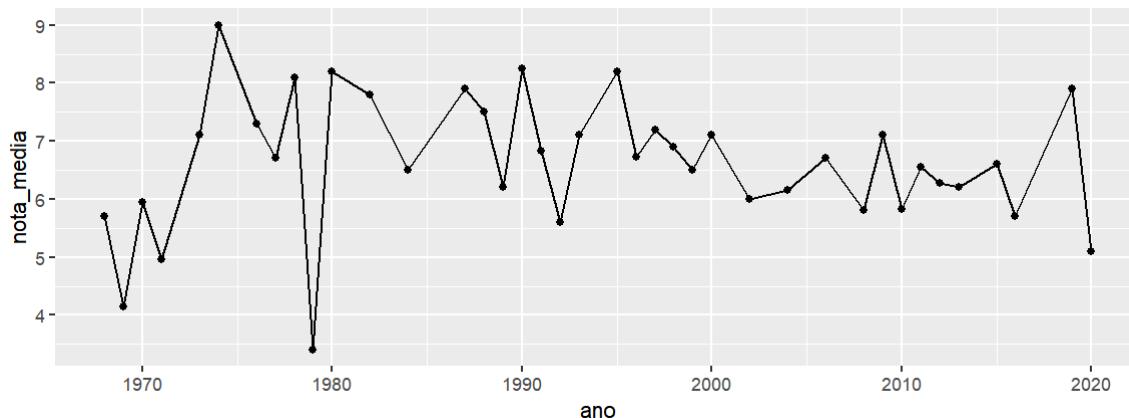
Se mapearmos uma coluna à cor das retas, serão construídas uma reta correspondente a cada categoria distinta dessa coluna.

```
imdb |>
  filter(!is.na(idioma)) |>
  mutate(idioma_ingles = case_when(idioma == "English" ~ "Apenas Inglês"
                                    str_detect(idioma, "English") ~ "Inglês e outros idiomas"
                                    TRUE ~ "Outros idiomas")) |>
  group_by(ano, idioma_ingles) |>
  summarise(num_filmes = n(), .groups = "drop") |>
  ggplot() +
  geom_line(aes(x = ano, y = num_filmes, color = idioma_ingles))
```



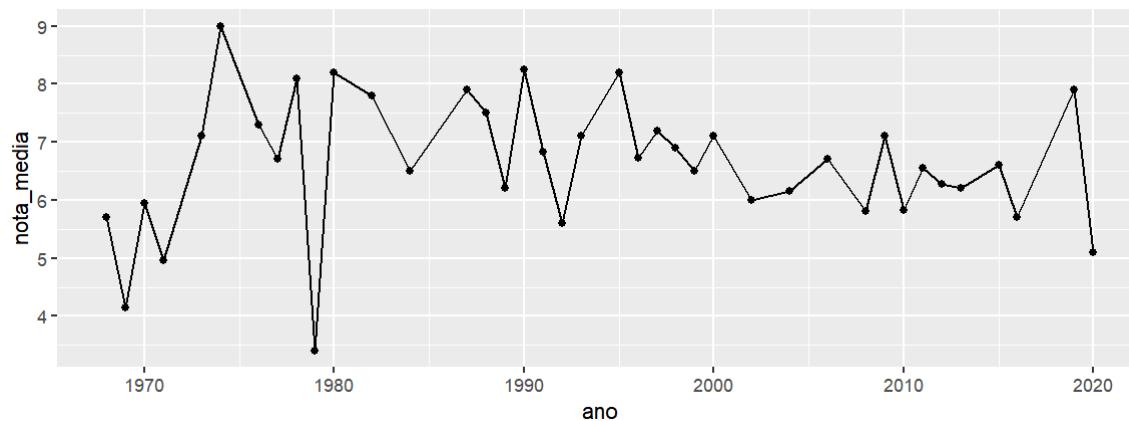
Podemos colocar pontos e retas no mesmo gráfico. Basta acrescentar os dois geoms. O gráfico abaixo mostra nota média anual dos filmes do Robert De Niro.

```
imdb |>
  filter(str_detect(elenco, "Robert De Niro")) |>
  group_by(ano) |>
  summarise(nota_media = mean(nota_imdb, na.rm = TRUE)) |>
  ggplot() +
  geom_line(aes(x = ano, y = nota_media)) +
  geom_point(aes(x = ano, y = nota_media))
```



Quando precisamos usar o mesmo aes() em vários geoms, podemos defini-lo dentro da função ggplot(). Esse aes() será então distribuído para todo geom do gráfico. O código anterior pode ser reescrito da seguinte forma.

```
imdb |>  
  filter(str_detect(elenco, "Robert De Niro")) |>  
  group_by(ano) |>  
  summarise(nota_media = mean(nota_imdb, na.rm = TRUE)) |>  
  ggplot(aes(x = ano, y = nota_media)) +  
  geom_line() +  
  geom_point()
```



Se algum geom necessitar de um atributo que os outros não precisam, esse atributo pode ser especificado normalmente dentro dele. Abaixo, utilizamos o `geom_label` para colocar as notas médias no gráfico. Além do x e y, o `geom_label` também precisa do texto que será escrito no gráfico.

```
imdb |>  
  filter(str_detect(elenco, "Robert De Niro")) |>  
  group_by(ano) |>  
  summarise(nota_media = mean(nota_imdb, na.rm = TRUE)) |>  
  mutate(nota_media = round(nota_media, 1)) |>  
  ggplot(aes(x = ano, y = nota_media)) +  
  geom_line() +  
  geom_label(aes(label = nota_media))
```

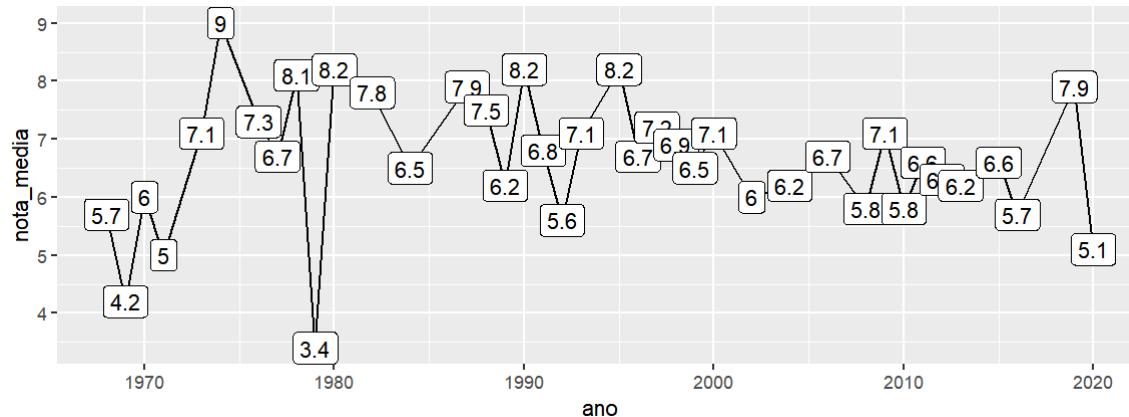
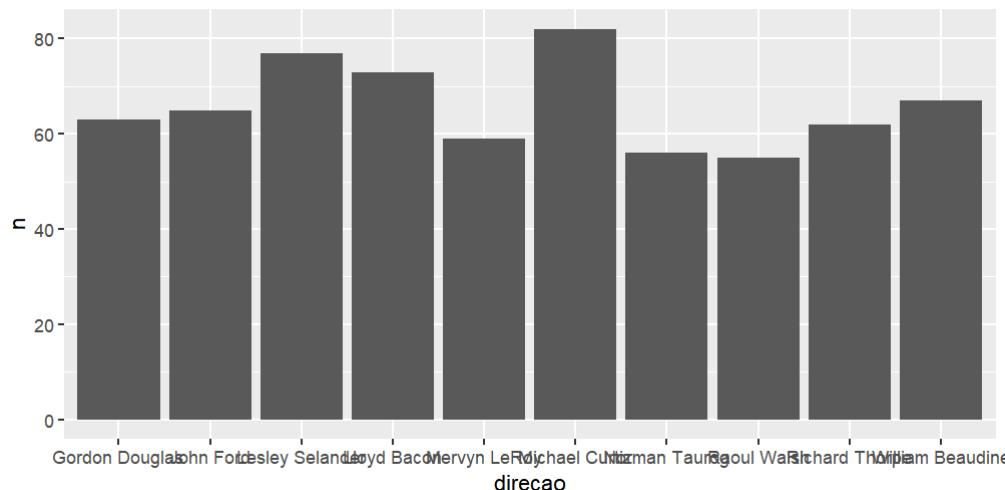


Gráfico de barras

Para construir gráficos de barras, utilizamos o `geom_col`. A seguir, construímos um gráfico de barras do número de filmes das 10 pessoas que mais dirigiram filmes na nossa base do IMDB.

```
imdb |>  
  count(direcao) |>  
  slice_max(order_by = n, n = 10) |>  
  ggplot() +  
  geom_col(aes(x = direcao, y = n))
```

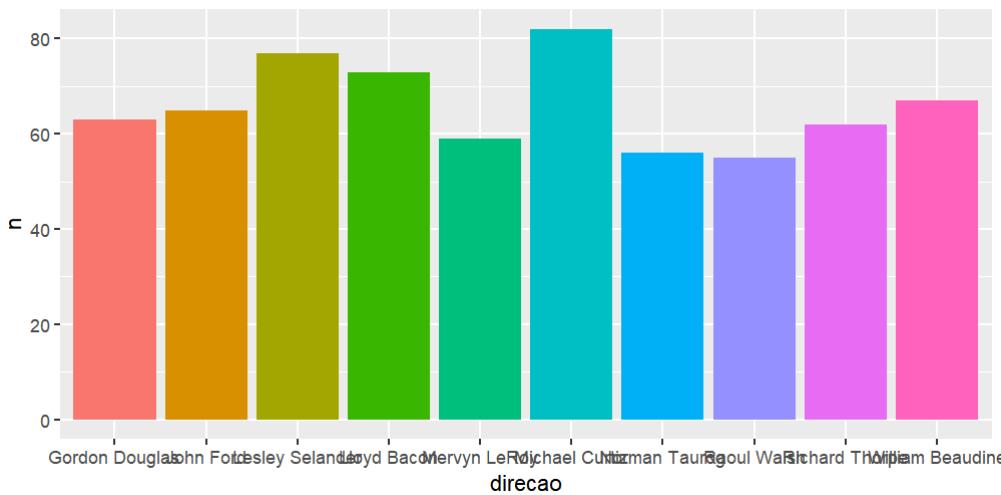


Gráficos de barras também precisam dos atributos `x` e `y`, sendo que o atributo `y` representará a altura de cada barra.

No gráfico anterior, vemos que o NA é considerado uma "opção" de direcao e entra no gráfico. Podemos retirar os NAs dessa coluna previamente utilizando a função `filter()`.

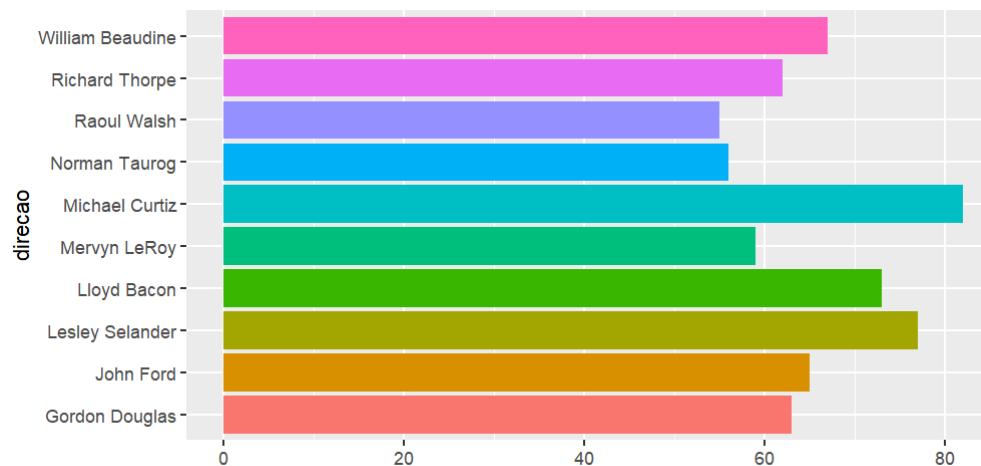
A seguir, além de retirar os NAs, atribuímos a coluna `direcao` à cor das colunas. Repare que, nesse caso, não utilizamos o atributo `color` e sim `fill`. A regra é a seguinte: o atributo `color` colore objetos sem área (pontos, linhas, contornos), o atributo `fill` preenche objetos com cor (barras, áreas, polígonos em geral).

```
imdb |>
  count(direcao) |>
  filter(!is.na(direcao)) |>
  slice_max(order_by = n, n = 10) |>
  ggplot() +
  geom_col(
    aes(x = direcao, y = n, fill = direcao), show.legend = FALSE
  )
```



Para consertar as labels do eixo x, uma alternativa é inverter os eixos do gráfico, construindo barras horizontais.

```
imdb |>
  count(direcao) |>
  filter(!is.na(direcao)) |>
  slice_max(order_by = n, n = 10) |>
  ggplot() +
  geom_col(
    aes(x = direcao, y = n, fill = direcao), show.legend = FALSE
  ) +
  coord_flip()
```

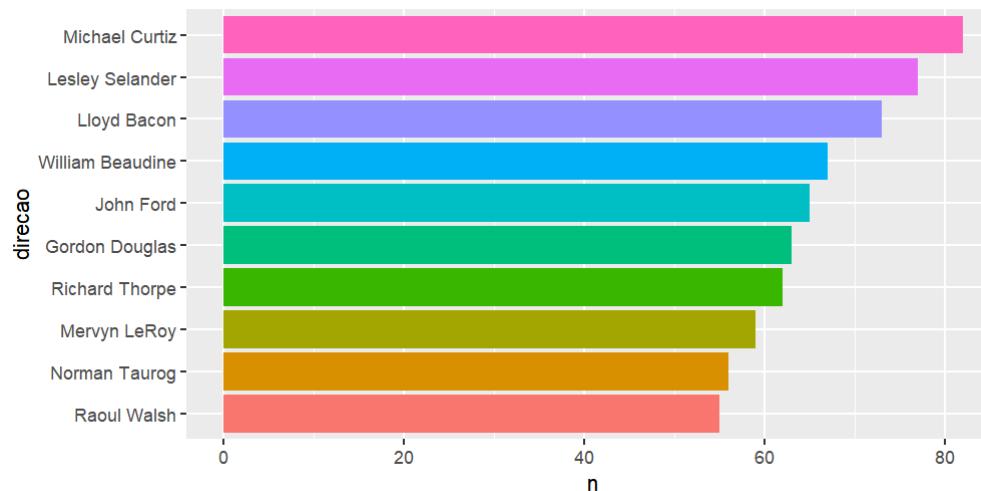


Para ordenar as colunas, precisamos mudar a ordem dos níveis do *fator direcao*. Para isso, utilizamos a função `fct_reorder()` do pacote `forcats`. A nova ordem será estabelecida pela coluna `n` (quantidade de filmes).

Fatores dentro do R são números inteiros (1, 2, 3, ...) que possuem uma representação textual. Variáveis categóricas são transformadas em fatores pelo `ggplot` pois todo eixo cartesiano é numérico. Assim, os textos de uma variável categórica são, internamente, números inteiros.

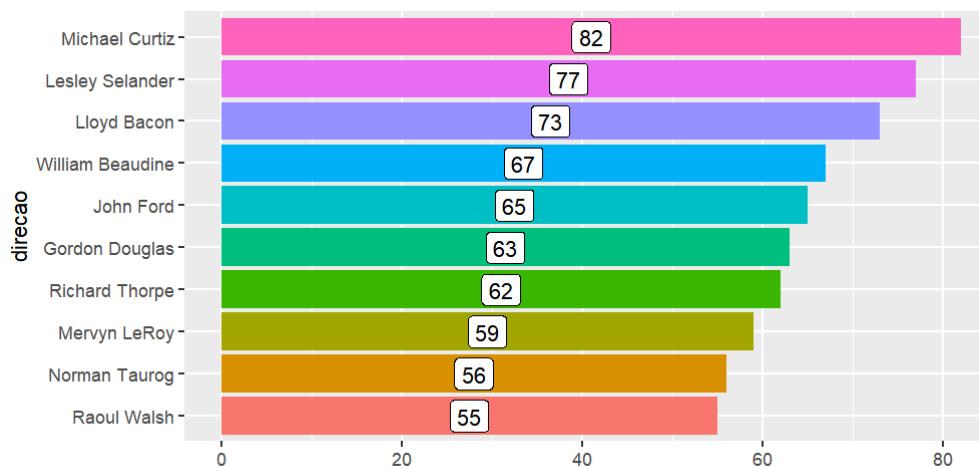
Por padrão, os inteiros são atribuídos a cada categoria de uma variável pela ordem alfabética (repare na ordem das pessoas que dirigiram filmes nos gráficos anteriores). Assim, se transformássemos o vetor `c("banana", "uva", "melancia")` em um fator, a atribuição de inteiros seria: "banana" vira 1, "melancia" vira 2 e "uva" vira 3. Embora sejam inteiros internamente, sempre que chamássemos esse novo vetor, ainda sim veríamos os textos "banana", "uva" e "melancia".

```
imdb |>
  count(direcao) |>
  filter(!is.na(direcao)) |>
  slice_max(order_by = n, n = 10) |>
  mutate(direcao =forcats::fct_reorder(direcao, n)) |>
  ggplot() +
  geom_col(
    aes(x = direcao, y = n, fill = direcao),
    show.legend = FALSE
  ) +
  coord_flip()
```



Por fim, podemos colocar uma label com o número de filmes de cada pessoa que dirigiu filmes dentro de cada barra.

```
imdb |>
  count(direcao) |>
  filter(!is.na(direcao)) |>
  slice_max(order_by = n, n = 10) |>
  mutate(direcao =forcats::fct_reorder(direcao, n)) |>
  ggplot() +
  geom_col(aes(x = direcao, y = n, fill = direcao), show.legend = FALSE) +
  geom_label(aes(x = direcao, y = n/2, label = n)) +
  coord_flip()
```



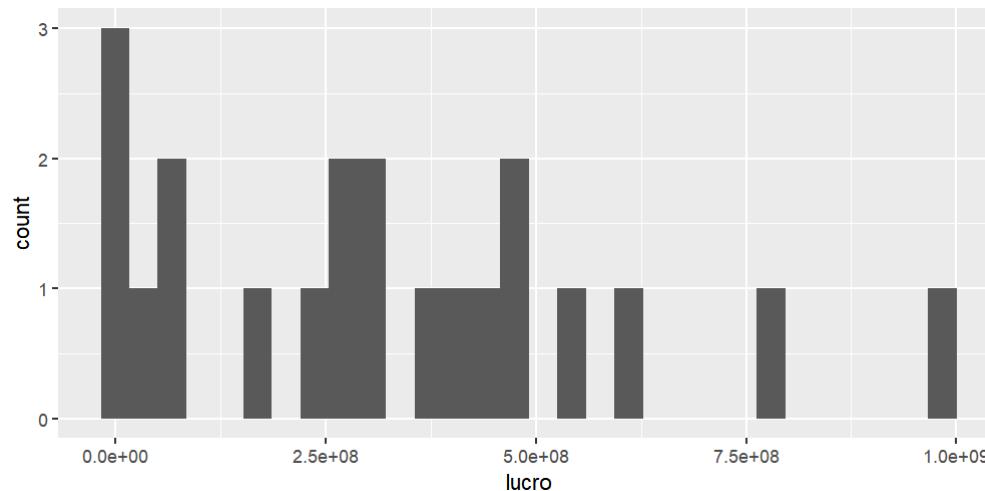
Histograma

Para construir histogramas, usamos o `geom_histogram`. Esse geom só precisa do atributo `x` (o eixo `y` é construído automaticamente). Histogramas são úteis para avaliarmos a distribuição de uma variável.

A seguir, construímos o histograma do lucro dos filmes do diretor Steven Spielberg. O primeiro *warning* nos diz que o eixo `x` foi dividido em 30 intervalos para a construção do histograma.

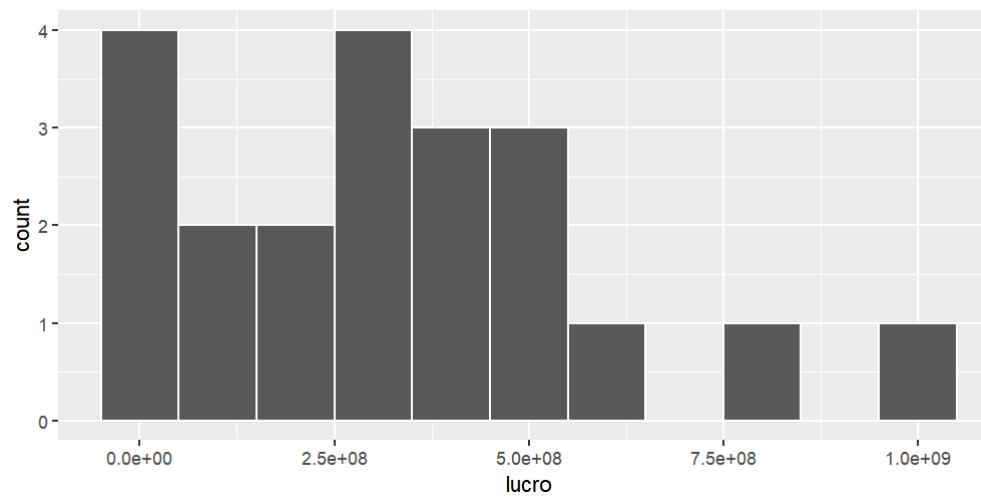
```
imdb |>
  filter(direcao == "Steven Spielberg") |>
  ggplot() +
  geom_histogram(aes(x = lucro))
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.



Para definir o tamanho de cada intervalo, podemos utilizar o argumento `binwidth`.

```
imdb |>
  filter(direcao == "Steven Spielberg") |>
  ggplot() +
  geom_histogram(
    aes(x = lucro),
    binwidth = 1000000000,
    color = "white"
  )
```



Boxplot

Boxplots também são úteis para estudarmos a distribuição de uma variável, principalmente quando queremos comparar várias distribuições.

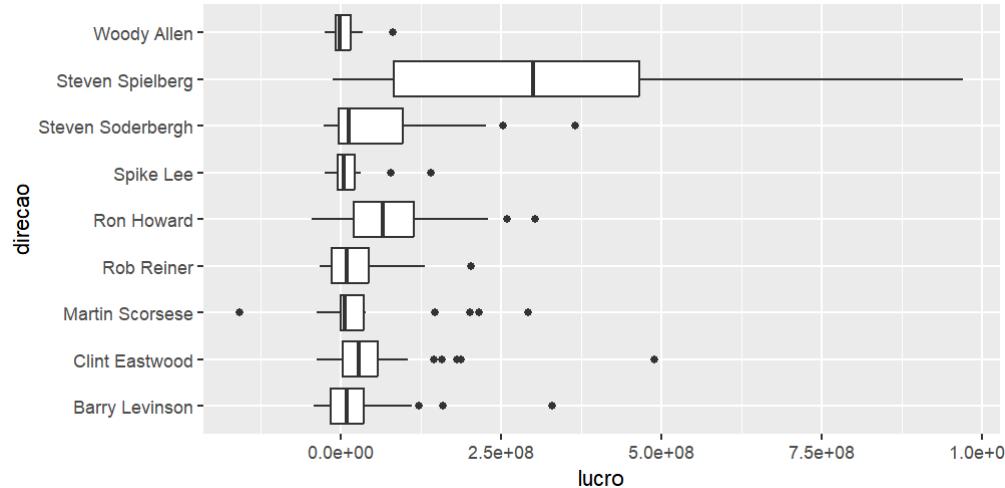
Para construir um boxplot no ggplot, utilizamos a função `geom_boxplot`. Ele precisa dos atributos `x` e `y`, sendo que ao atributo `x` devemos mapear uma variável categórica.

A seguir, construímos boxplots do lucro dos filmes das pessoas que dirigiram de 15 filmes.

```

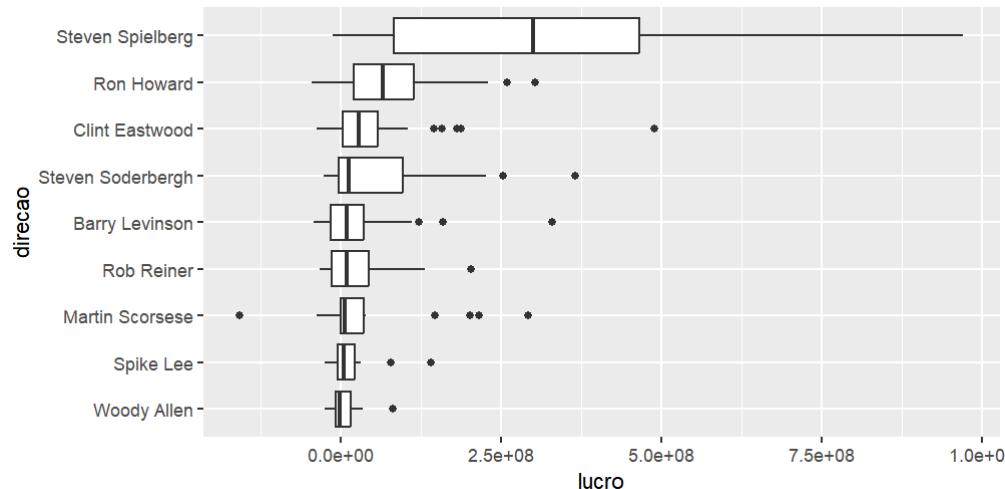
direcao_mais_15_filmes <- imdb |>
  filter(!is.na(direcao), !is.na(lucro) ) |>
  group_by(direcao) |>
  count(direcao, sort = TRUE) |>
  filter(n >= 15)
imdb |>
  filter(direcao %in% direcao_mais_15_filmes$direcao) |>
  ggplot() +
  geom_boxplot(aes(x = direcao, y = lucro)) +
  coord_flip()

```



Também podemos reordenar a ordem dos boxplots utilizando a função `forcats::fct_reorder`.

```
imdb |>  
  filter(direcao %in% direcao_mais_15_filmes$direcao) |>  
  mutate(direcao = forcats::fct_reorder(direcao, lucro, .na_rm = TRUE)) |>  
  ggplot() +  
  geom_boxplot(aes(x = direcao, y = lucro)) +  
  coord_flip()
```



Títulos, labels e escalas

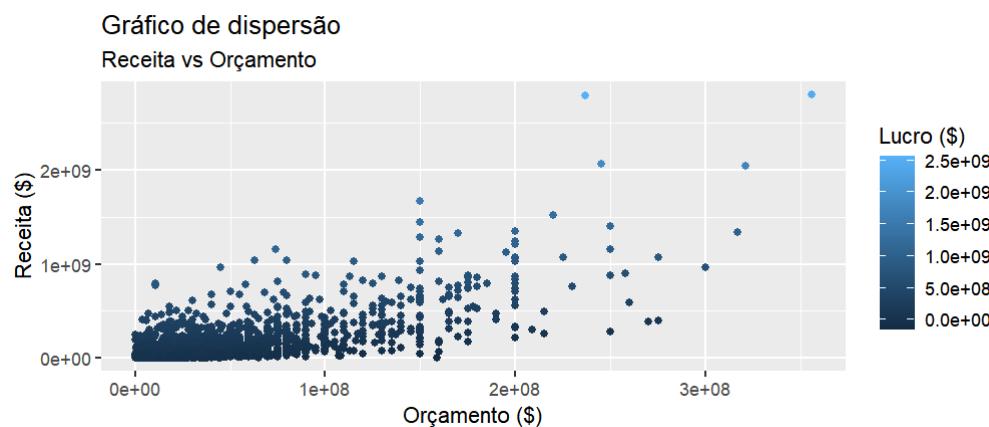
Para colocar títulos no gráfico ou trocar as labels dos atributos, utilizamos a função `labs()`.

Para mudar as escalas (textos e quebras), utilizamos as funções da família `scale_`.

Podemos usar a função `coord_cartesian()` para definir qual porção do gráfico deve ser mostrada.

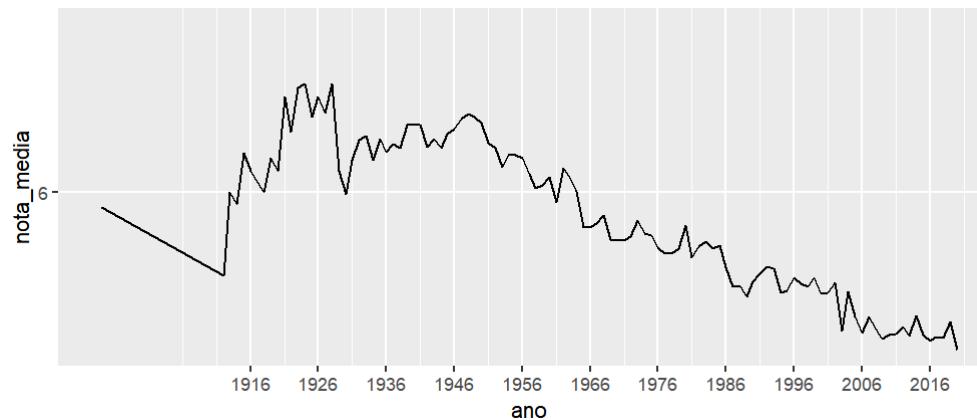
Colocando título, subtítulo e mudando as labels.

```
imdb |>  
  ggplot() +  
  geom_point(mapping = aes(x = orçamento, y = receita, color = lucro))  
  labs(  
    x = "Orçamento ($)", y = "Receita ($)", color = "Lucro ($)",  
    title = "Gráfico de dispersão", subtitle = "Receita vs Orçamento")  
)
```



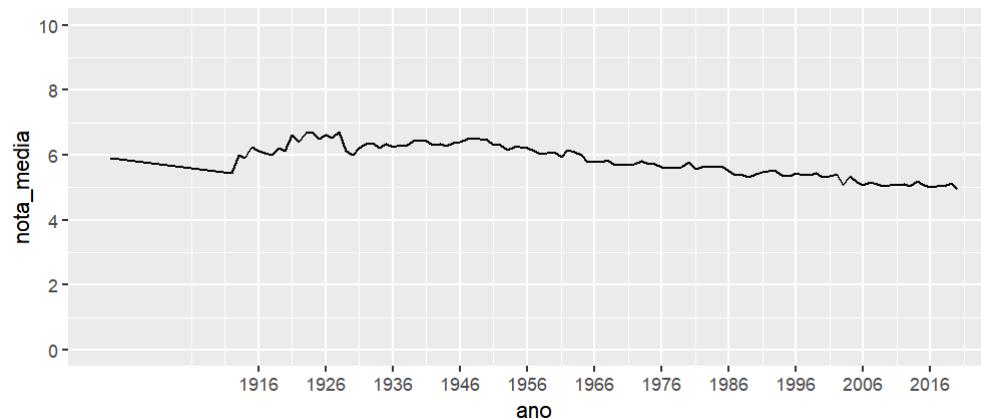
Mudando as quebras dos eixos x e y.

```
imdb |>
  group_by(ano) |>
  summarise(nota_media = mean(nota_imdb, na.rm = TRUE)) |>
  ggplot() +
  geom_line(aes(x = ano, y = nota_media)) +
  scale_x_continuous(breaks = seq(1916, 2016, 10)) +
  scale_y_continuous(breaks = seq(0, 10, 2))
```



Mudando a escala do gráfico.

```
imdb |>  
  group_by(ano) |>  
  summarise(nota_media = mean(nota_imdb, na.rm = TRUE)) |>  
  ggplot() +  
  geom_line(aes(x = ano, y = nota_media)) +  
  scale_x_continuous(breaks = seq(1916, 2016, 10)) +  
  scale_y_continuous(breaks = seq(0, 10, 2)) +  
  coord_cartesian(ylim = c(0, 10))
```



Cores

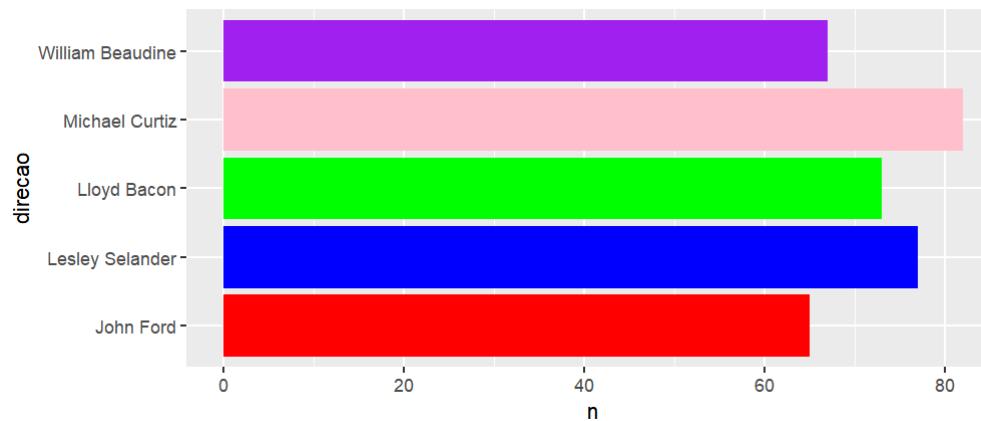
Também existe funções `scale` para os atributos de cor: `scale_color_` e `scale_fill_`.

Para escolher manualmente as cores de um gráfico, utilize as funções `scale_color_manual` e `scale_fill_manual()`.

Para trocar o nome nas legendas geradas por atributos de cor, utilize as funções `scale_color_discrete` e `scale_fill_discrete`.

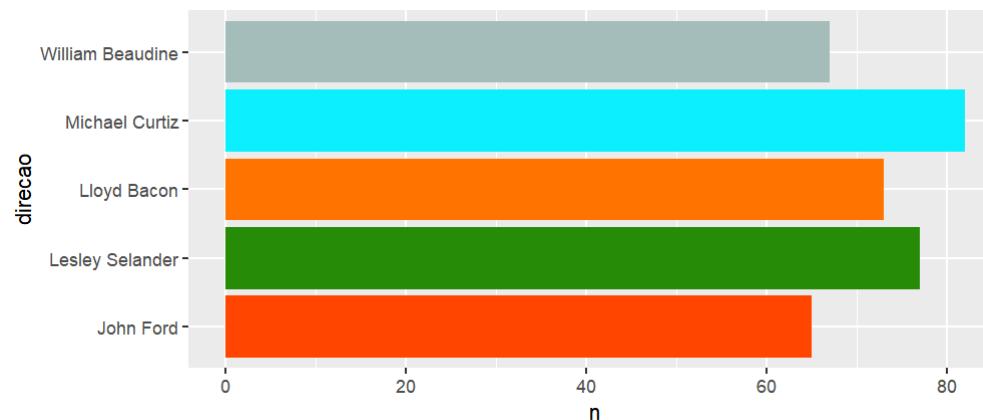
Substituindo as cores padrão do gráfico por um outro conjunto de cores.

```
imdb |>
  count(direcao) |>
  filter(!is.na(direcao)) |>
  slice_max(order_by = n, n = 5) |>
  ggplot() +
  geom_col(aes(x = direcao, y = n, fill = direcao), show.legend = FALSE) +
  coord_flip() +
  scale_fill_manual(values = c("red", "blue", "green", "pink", "purple"))
```



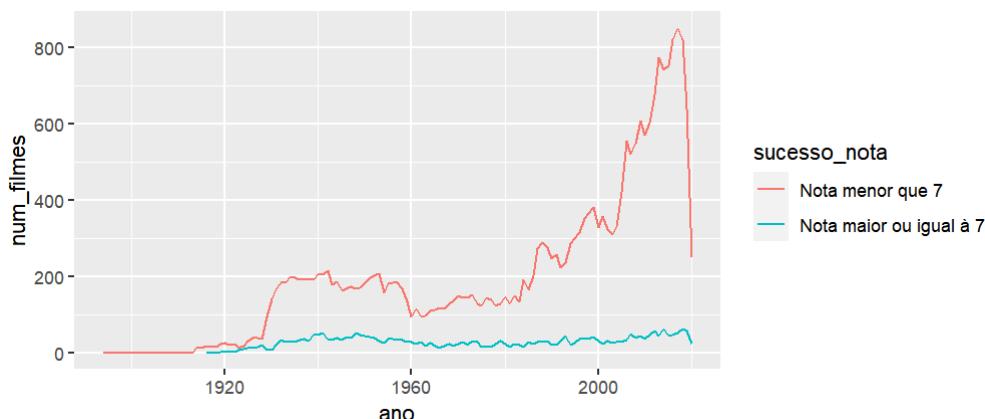
Também podemos usar códigos hexadecimais.

```
imdb |>
  count(direcao) |>
  filter(!is.na(direcao)) |>
  slice_max(order_by = n, n = 5) |>
  ggplot() +
  geom_col(aes(x = direcao, y = n, fill = direcao), show.legend = FALSE) +
  coord_flip() +
  scale_fill_manual(
    values = c("#ff4500", "#268b07", "#ff7400", "#0befff", "#a4bdba"))
)
```



Trocando os textos da legenda.

```
imdb |>
  mutate(sucesso_nota = case_when(nota_imdb >= 7 ~ "sucesso_nota_imdb"
                                    TRUE ~ "sem_sucesso_nota_imdb")) |>
  group_by(ano, sucesso_nota) |>
  summarise(num_filmes = n(), .groups = "drop") |>
  ggplot() +
  geom_line(aes(x = ano, y = num_filmes, color = sucesso_nota)) +
  scale_color_discrete(labels = c("Nota menor que 7", "Nota maior ou igual à 7"))
```

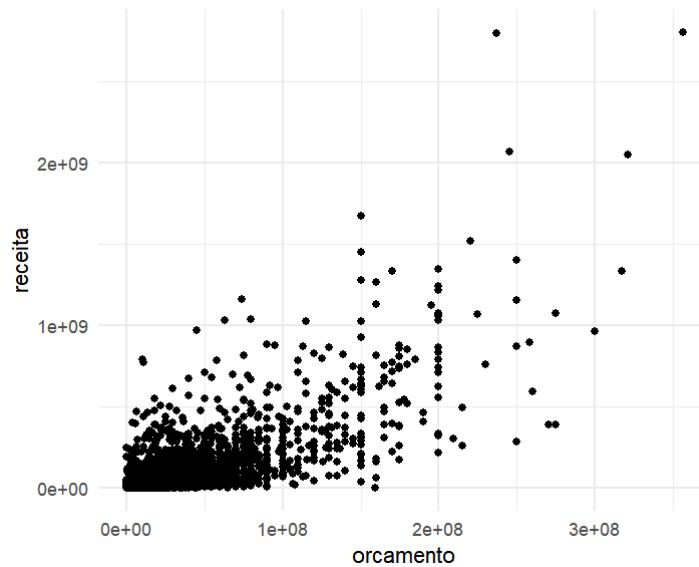


Temas

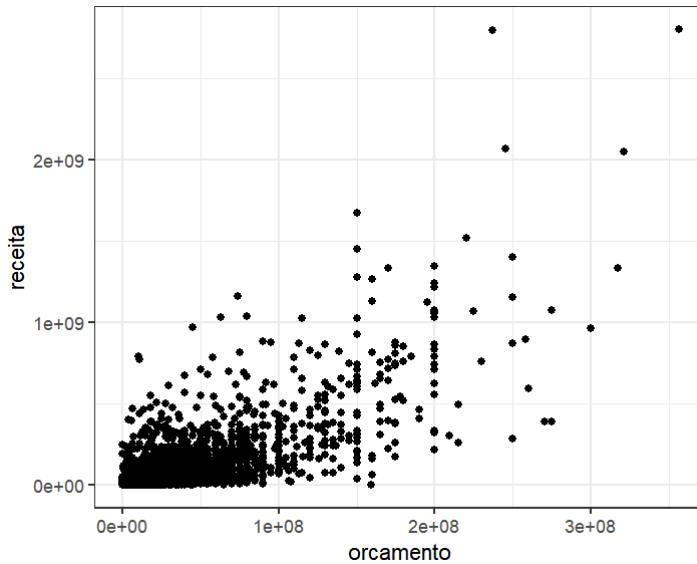
Os gráficos que vimos até agora usam o tema padrão do ggplot2. Existem outros temas prontos para utilizarmos presentes na família de funções `theme_`.

Você também pode criar o seu próprio tema utilizando a função `theme()`. Nesse caso, para trocar os elementos estéticos do gráfico precisamos usar as funções `element_text()` para textos, `element_line()` para linhas, `element_rect()` para áreas e `element_blank()` para remover elementos.

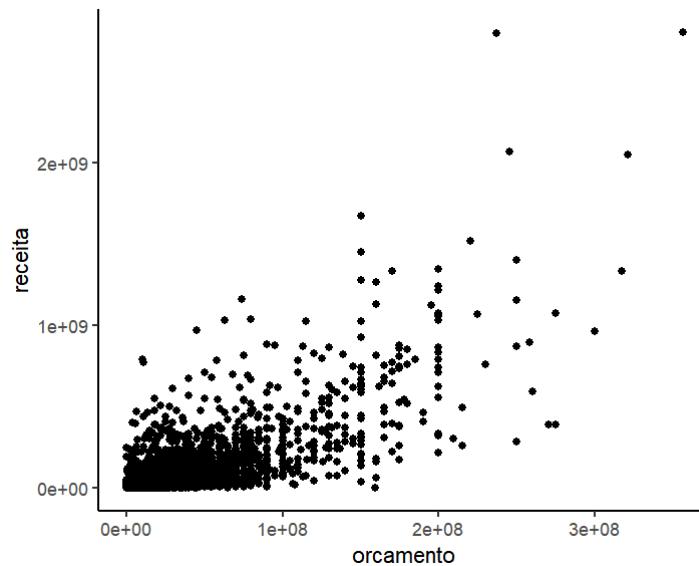
```
imdb |>  
  ggplot() +  
  geom_point(mapping = aes(x = orcamento, y = receita)) +  
  theme_minimal()
```

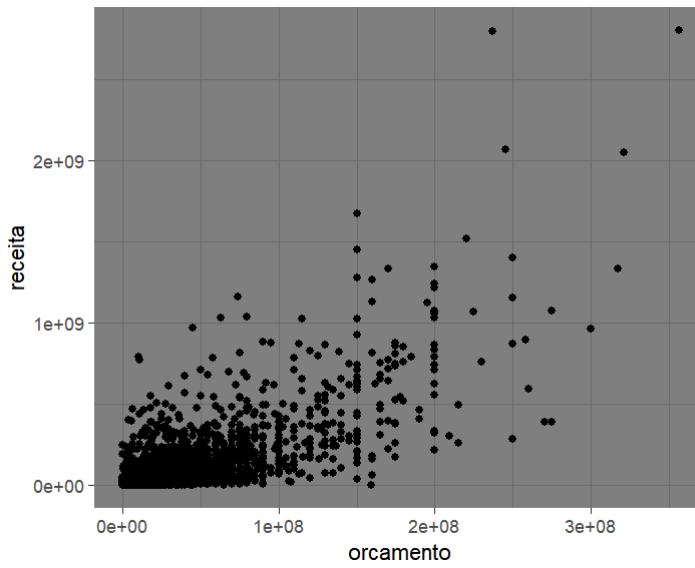


```
imdb |>  
  ggplot() +  
  geom_point(mapping = aes(x = orcamento, y = receita)) +  
  theme_bw()
```



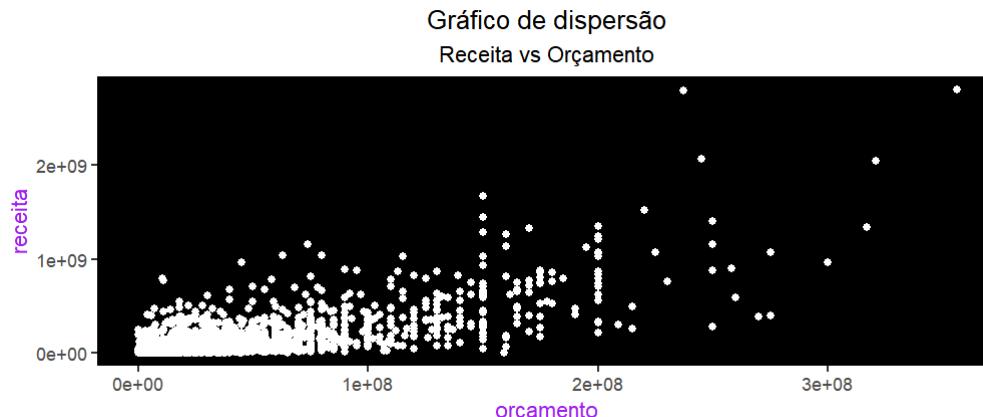
```
imdb |>  
  ggplot() +  
  geom_point(mapping = aes(x = orcamento, y = receita)) +  
  theme_classic()
```



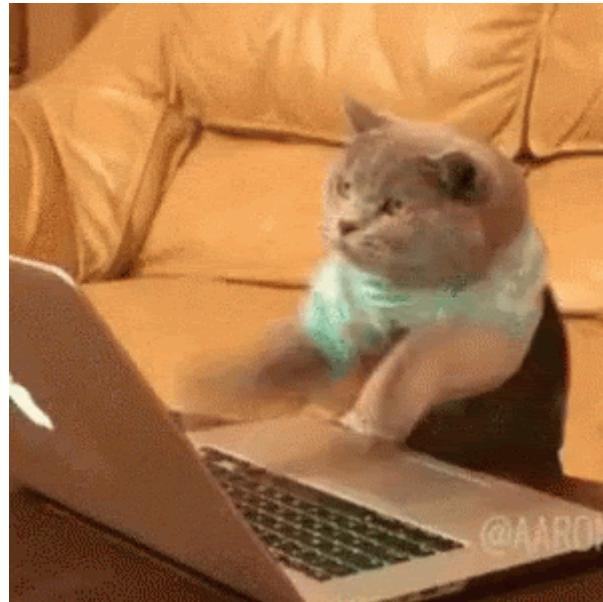


Criando um próprio tema.

```
imdb |>
  ggplot() +
  geom_point(mapping = aes(x = orçamento, y = receita), color = "white")
  labs(title = "Gráfico de dispersão", subtitle = "Receita vs Orçamento",
       theme(
         plot.title = element_text(hjust = 0.5),
         plot.subtitle = element_text(hjust = 0.5),
         axis.title = element_text(color = "purple"),
         panel.background = element_rect(fill = "black"),
         panel.grid = element_blank()
       )
)
```



Vamos ao R!



Otimização Visual

Objetivos de aprendizagem

- Discutir os principais aspectos visuais a serem considerados em um gráfico estatístico
- Compreender como modificar um `ggplot` para comunicar resultados de maneira eficiente.
- Otimizar um gráfico que utilizamos na aula anterior.

O que é otimização visual?

- É o ato de trabalhar em uma visualização aprimorar a comunicação.
- Pode envolver alterações nas cores, fontes, elementos geométricos, entre outros, a partir de um gráfico exploratório.
- Não é uma ciência exata, mas recolhe elementos da ciência para aumentar a probabilidade de sucesso da visualização.

Recursos pré-atentativos

- Uma propriedade visual pré-atentativa é processada pelo nosso cérebro antes de uma ação consciente.
- Como isso esse processamento é muito rápido, trata-se de uma oportunidade para tornar visualizações mais amigáveis e diretamente interpretadas.
- Segundo Colin Ware, existem 4 propriedades pré-atentativas que podemos explorar:
 - Cor
 - Forma
 - Movimento
 - Posicionamento
- O objetivo dos recursos pré-atentativos é **chamar a atenção**.

Recursos pré-atentativos

- Uma propriedade visual pré-atentativa é processada pelo nosso cérebro antes de uma ação consciente.
- Como isso esse processamento é muito rápido, trata-se de uma oportunidade para tornar visualizações mais amigáveis e diretamente interpretadas.
- Segundo Colin Ware, existem 4 propriedades pré-atentativas que podemos explorar:
 - Cor
 - Forma
 - Movimento
 - Posicionamento
- O objetivo dos recursos pré-atentativos é chamar a atenção.

E por último isso

Primeiro você lerá isso

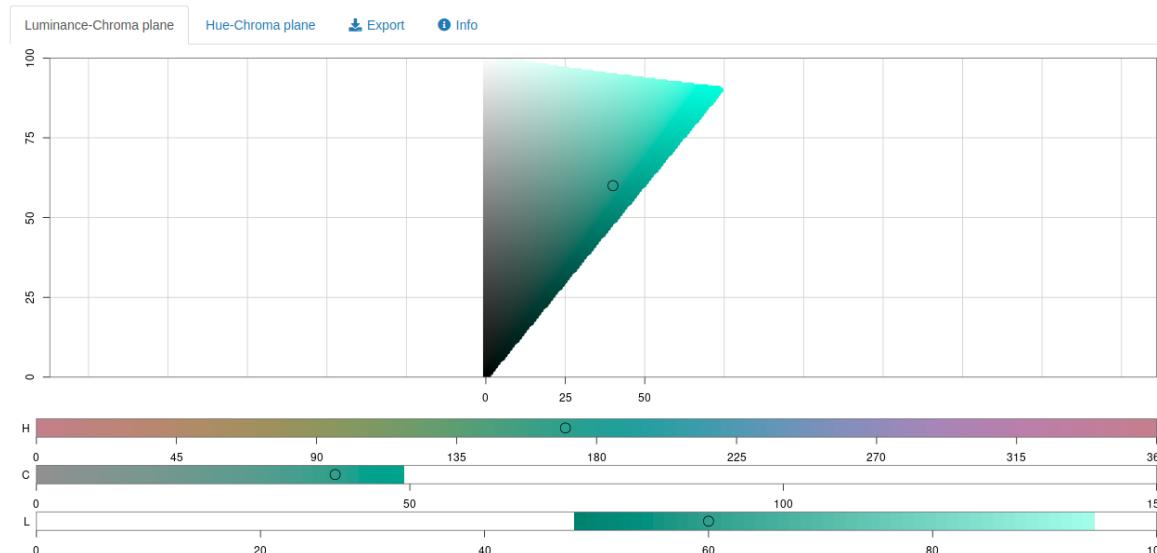
Depois você lerá isso

Então isso

Cor

Cores do zero: HCL vs RGB

- HCL (hue, chroma, luminance) é vantajoso pois é mais intuitivo de criar do que RGB, já que o HCL tem apenas um eixo de cores (*hue*), enquanto RGB é uma composição de três (*red, green, blue*)
- O pacote {colorspace} permite usar HCL para definir uma cor/paleta: experimente `colorspace::hcl_color_picker()`
- [Nesse site](#), também é possível criar algumas paletas



Paletas de cores prontas

- **Escalas qualitativas:** utilizado para variáveis nominais (sexo, cor/raça)
- **Escalas divergentes:** utilizado para variáveis que têm um centro neutro (favorável/neutro/desfavorável, correlação)
- **Escalas sequenciais:** utilizado para variáveis ordinais (faixa etária, renda)
- **Viridis:** útil para comunicar com pessoas com daltonismo

Paletas de cores no {ggplot2}

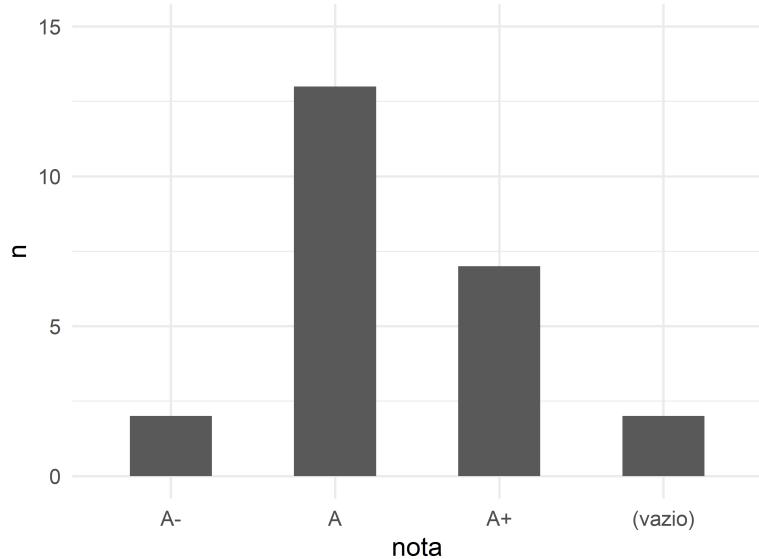
- `scale_*_brewer()`: utilizada para variáveis discretas. Possui três tipos: divergente, qualitativa e sequencial.
- `scale_*_distiller()`: utilizada para variáveis contínuas. Interpola as cores do *brewer* para lidar com todos os valores.
- `scale_*_fermenter()`: utilizada para variáveis contínuas, que são transformadas em discretas (binned).
- `scale_*_viridis_[cdb]`: Escala viridis para variáveis contínuas, discretas ou binned.
- `scale_*_manual()`: inclui um conjunto de cores manualmente.

Paletas de outros pacotes

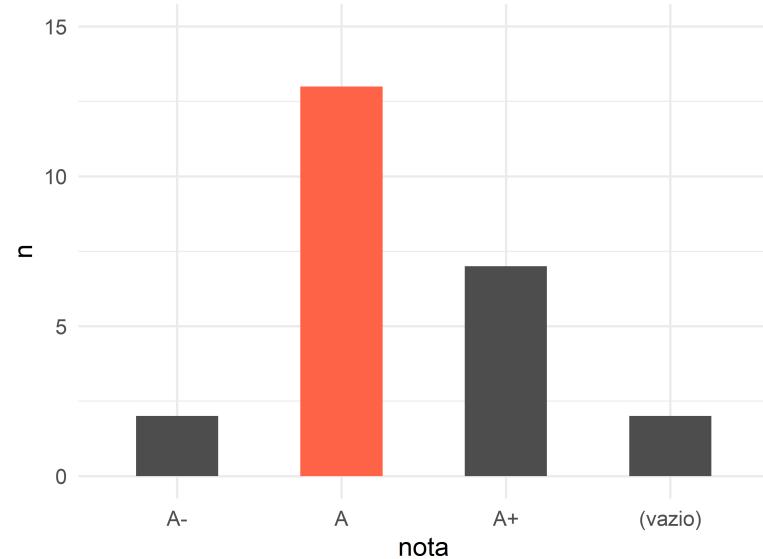
- {ggthemr} tem um monte de paletas, mas está um pouco desatualizado.
- {hrbrthemes} contém uma lista de temas escolhidos pelo Bob Rudis.
- {ghibli} tem paletas de cores relacionadas ao Studio Ghibli
- {paletteer} tem uma coleção de cores de vários outros pacotes de paletas.

Qual visualização é melhor?

A maioria dos filmes da pixar tem nota 'A'



A maioria dos filmes da pixar tem nota 'A'



É subjetivo...

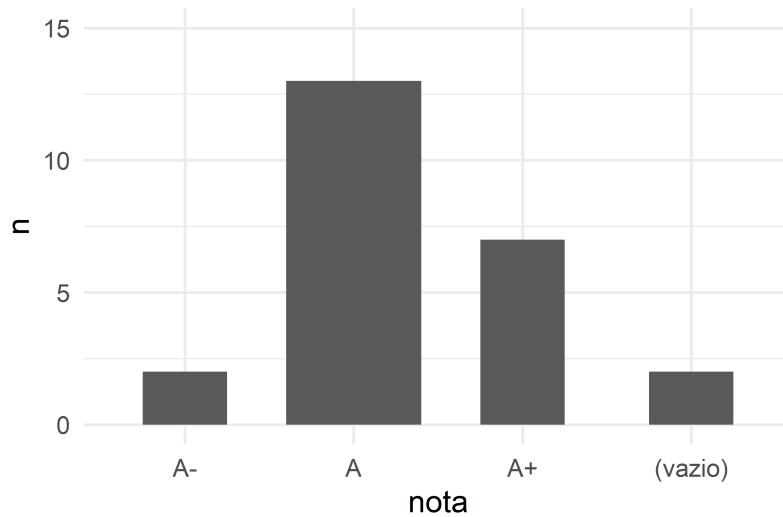
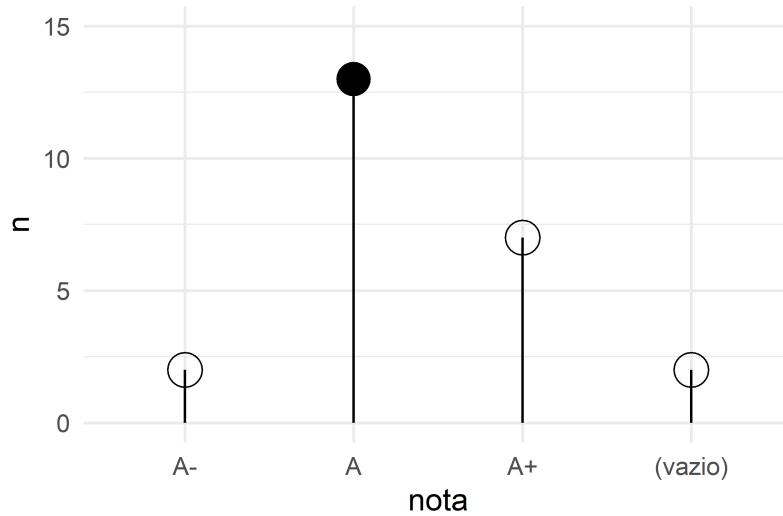
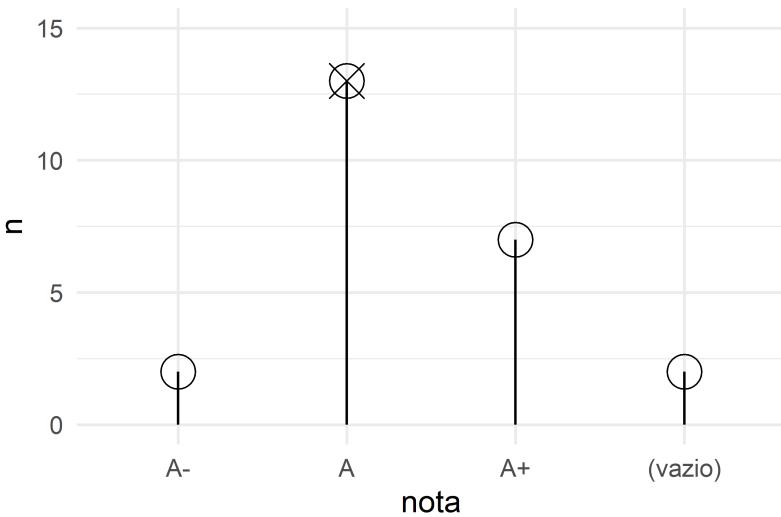
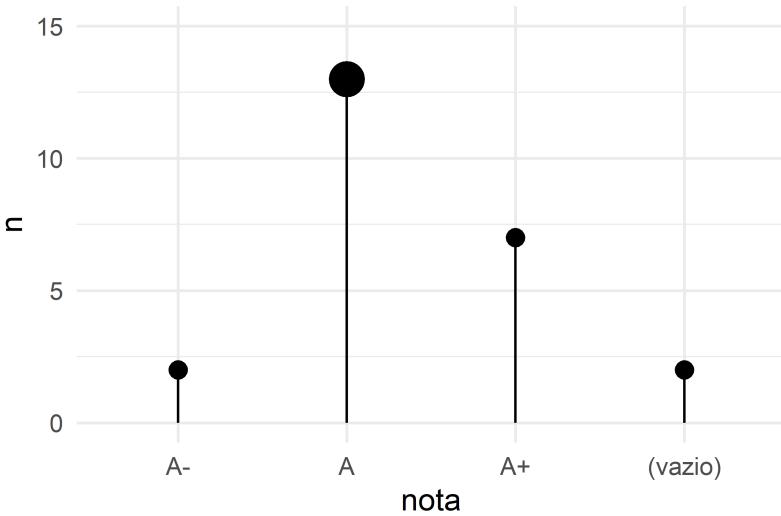


Forma

Formas

- Geralmente queremos contrastar formas para chamar a atenção...
- Sem com isso fazer uma bagunça visual.
- A ideia é utilizar apenas um elemento, como tamanho, forma, largura, marcações, angulações, etc. para mostrar o contraste.

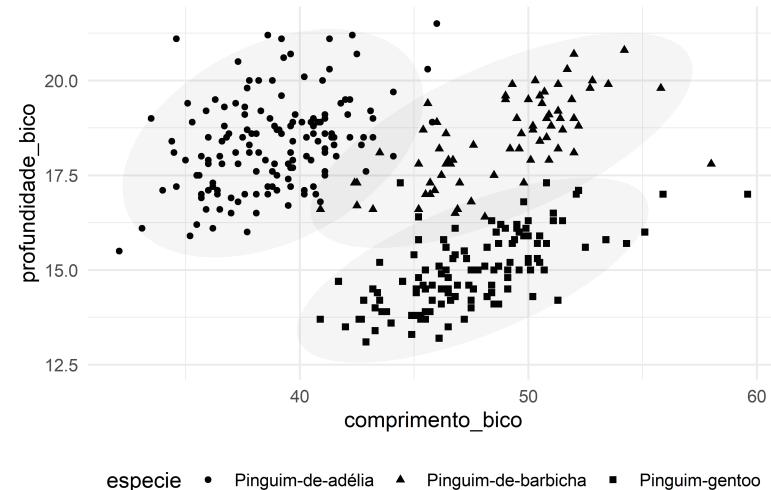
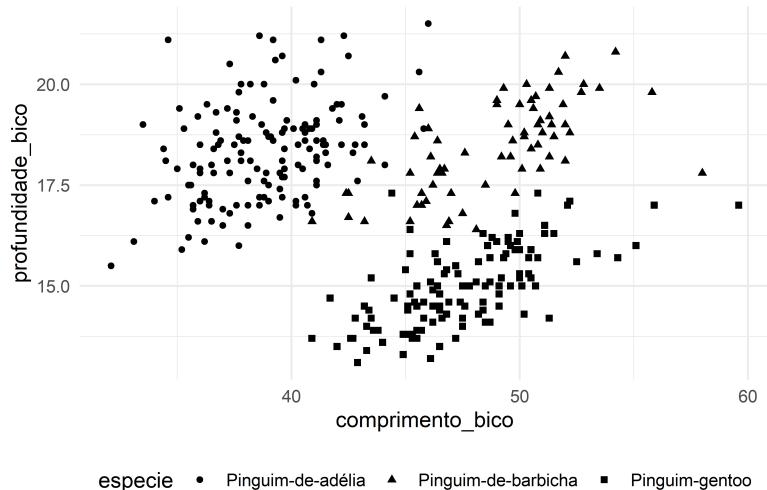
Exemplos



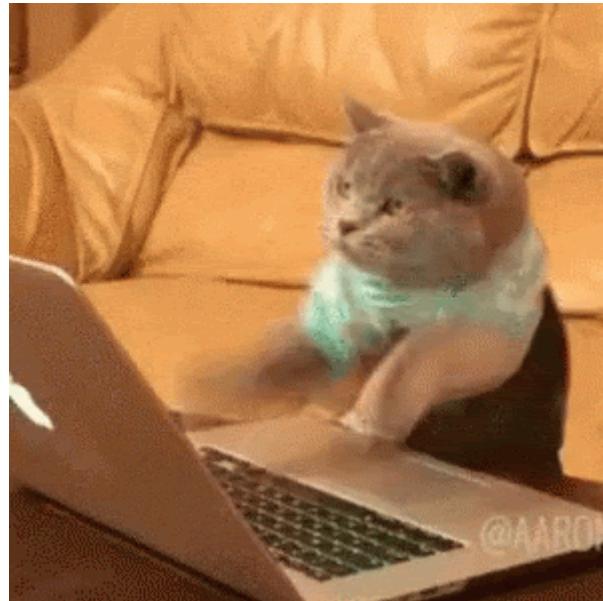
Princípios Gestalt

- **Gestalt** é uma área ampla da psicologia que não vamos aprofundar.
- Está ligada à ideia de que, em uma visualização, o todo é maior do que a soma das partes.
- Dois princípios Gestalt aplicáveis em visualização de dados são a nossa capacidade de:
 - Completar figuras
 - Agrupar objetos
- Ou seja, podemos criar visualizações que ativam essas capacidades.

Qual visualização é melhor?



Vamos ao R!



Extensões do ggplot2

Objetivos de aprendizagem

- Conhecer extensões do {ggplot2}
- Experimentar extensões separadamente
- Estudar um exemplo juntando várias extensões.

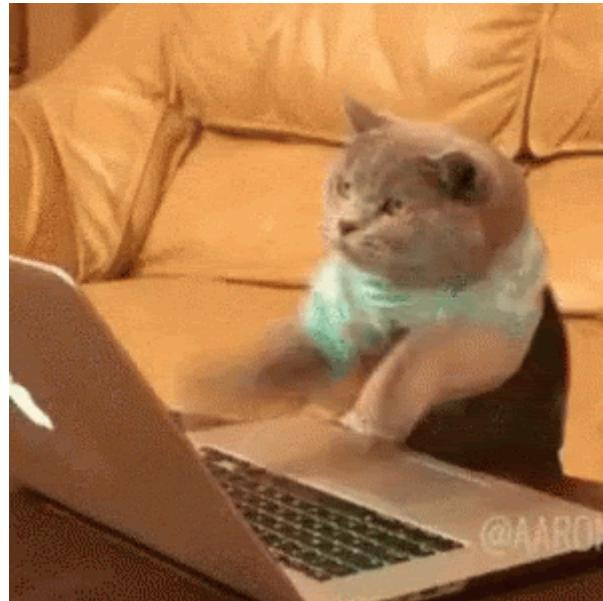
O que é uma extensão?

- O `{ggplot2}` é maravilhoso, mas não possui todos os gráficos possíveis.
- Por isso, pessoas do mundo todo desenvolvem extensões para a comunidade usar, que vão desde pacotes com novos temas/cores, geometrias eixos até a **possibilidade de desenhar gatinhos em barras**
- A equipe do tidyverse mantém uma [lista curada de extensões do `{ggplot2}`](#)

Como criar uma extensão?

- A [edição mais recente](#) do livro do `{ggplot2}` possui um tutorial de como criar extensões para o pacote.
-  Cuidado! Criar extensões do `{ggplot2}` não é fácil. Trata-se de um pacote complexo, exigindo bastante conhecimento de elementos internos do R.
- Uma forma legal de estudar extensões é olhando o código de pacotes. Recomendamos as extensões do [Thomas Lin Pedersen](#), já que ele é o mantenedor atual do `{ggplot2}` e, portanto, conhece muito bem as melhores práticas para criar extensões.

Vamos ao R!



Interatividade

Objetivos de aprendizagem

- Conhecer a diferença entre visualizações estáticas e dinâmicas
- Experimentar algumas visualizações
- Cobrir temas que não foram abordados no curso e encerrar.

Gráficos interativos e estáticos

- Gráficos interativos têm o poder de engajar mais, por conta do efeito *voosh*. Todo ser humano gosta de interagir com aquilo que está analisando.
- No entanto, isso vem com um custo: não são todos os documentos capazes de processar visualizações dinâmicas. Em particular, PDF, Word e PPT não rodam esses conteúdos.
- Vamos visitar tanto o mundo estático quanto dinâmico, para que você saiba por onde começar quando receber um novo desafio.

htmlwidgets

htmlwidgets são bibliotecas de visualização JavaScript encapsuladas em pacotes de R. Elas nos permitem usar diversas ferramentas JavaScript diretamente do R, adicionando algumas poucas linhas de código em nosso script.

Usando htmlwidgets, conseguimos construir tabelas, gráficos, mapas e muito outras visualizações interativas e naturalmente bonitas.

[Clique aqui](#) para acessar uma lista completa de todos os htmlwidgets disponíveis.

Tabelas com reactable

O pacote `reactable` nos permite criar tabelas interativas baseadas na biblioteca [React Table](#).

[Clique aqui](#) para acessar o tutorial completo do pacote `{reactable}`.

A interatividade dos `htmlwidgets` não depende de uma sessão R rodando por trás. Você pode utilizá-los em qualquer documento `.html`.

	mpg	cyl	disp	hp	drat	wt	qsec
Mazda RX4	21	6	160	110	3.9	2.62	16.46
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02
Datsun 710	22.8	4	108	93	3.85	2.32	18.61
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44

1–4 of 32 rows

Previous **1** 2 3 4 5 ... 8 Next

Pacotes alternativos para tabelas

A seguir, uma lista de pacotes/funções alternativos que trazem soluções para visualização de tabelas.

- `knitr::kable()`: não é um `htmlwidget` (não possui interatividade), mas é uma solução para formatar tabelas quando não precisamos que elas sejam interativas. Funciona em conjunto com o pacote `{kableExtra}`.
- `{flextable}`: também não é interativo, é um excelente pacote para editar tabelas. Funciona quando trabalhamos com relatórios em Word, e também integra bem com o pacote `{DT}`.
- `DT::datatable()`: outro `htmlwidget` para criar tabelas interativas. Funciona tal como o `reactable()`, mas um pouco mais burocrático para formatar as tabelas. Baseado na biblioteca JavaScript `DataTables`.

Tutoriais

- [Tutorial kable e kableExtra](#)
- [Tutorial flextable](#)
- [Tutorial DT](#)

Gráficos com plotly

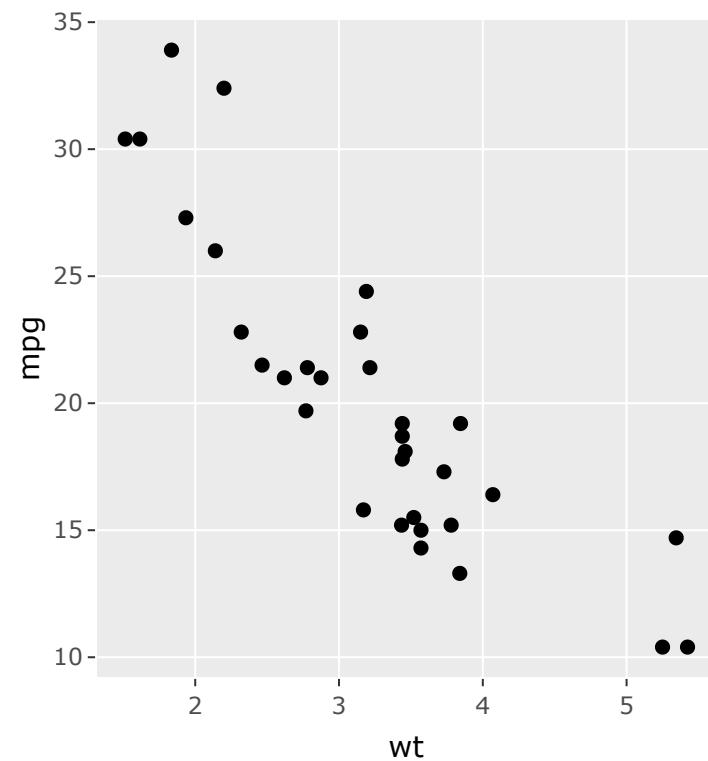
O pacote `plotly` nos permite criar gráficos interativos baseados na biblioteca `Plotly` (construída em `D3`).

Embora seja possível criar um `plotly` do zero usando a função `plot_ly()`, um jeito muito eficiente de utilizar essa biblioteca é criar um `ggplot` e então utilizar a função `ggplotly()`. Veja o exemplo a seguir.

Tutoriais

- [Tutorial plotly](#)
- [Interactive web-based data visualization with R, plotly, and shiny](#)

```
gg <- ggplot(mtcars) +  
  aes(wt, mpg) +  
  geom_point()  
plotly::ggplotly(  
  p = gg,  
  height = 400  
)
```



Pacotes alternativos

A seguir, uma lista de pacotes/funções alternativos que trazem soluções para visualização de gráficos.

- `highcharter::highcharter()`: pacote gráfico baseado na biblioteca JavaScript [Highcharts](#). A biblioteca Highcharts é gratuita apenas para fins educacionais e não lucrativos (exceto órgãos governamentais). Para outros usos, você pode precisar de uma licença. **[avançado]**
- Procure por pacotes para tipos específicos de gráficos na [galeria de htmlwidgets](#).

Tutoriais

- [Tutorial highcharter](#)
- [Documentação Highcharts](#)

Mapas com leaflet

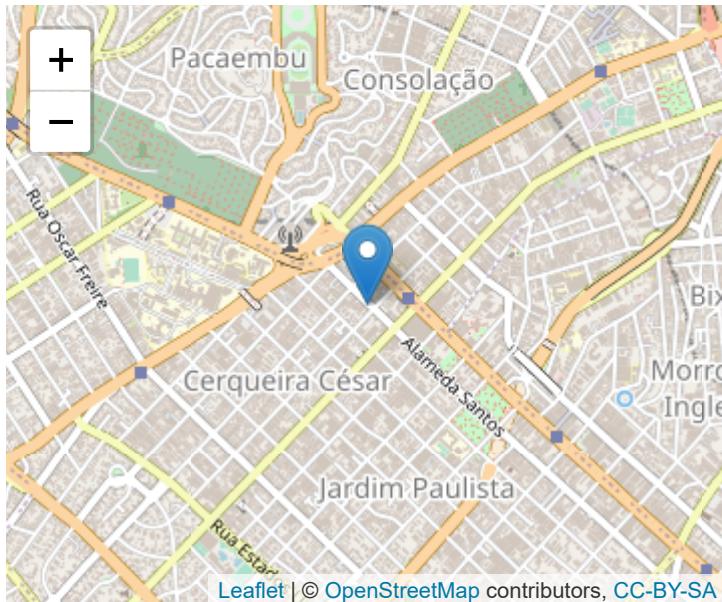
O pacote {leaflet} nos permite criar mapas interativos baseados na biblioteca JavaScript open-source [Leaflet](#).

Para criar um mapa leaflet, utilizamos a função `leaflet::leaflet()` e diversas funções auxiliares para caracterizar nosso mapa. Um tutorial de como utilizar o leaflet se encontra [aqui](#).

A seguir, mostramos um exemplo simples de como criar um mapa leaflet.

```
library(leaflet)

leaflet(height = 300) |>
  addTiles() |> # Adiciona a camada gráfica do OpenStreetMap (padrão)
  addMarkers(
    lng = -46.6623969, lat = -23.5581664,
    popup = "A Curso-R morava aqui antes da pandemia. Agora ela mora"
  )
```



Pacotes alternativos

- `highcharter::hcmap()`: variação do `highcharter` para mapas, baseada na biblioteca JavaScript **Highcharts**. **[avançado]**
- `{tmap}`: Pacote focado em mapas temáticos.

Tutoriais

- Construindo mapas com o `highcharter`
- Documentação Highmaps
- Documentação do `tmap`

Referências e material extra

htmlwidgets

- Galeria htmlwidgets

reactable

- A biblioteca React Table
- Tutorial reactable

DT

- Tutorial DT

plotly

- Tutorial plotly
- Interactive web-based data visualization with R, plotly, and shiny

highcharter/highcharts

- Tutorial highcharter
- Biblioteca Highcharts
- Galeria Highcharts
- Documentação Highcharts

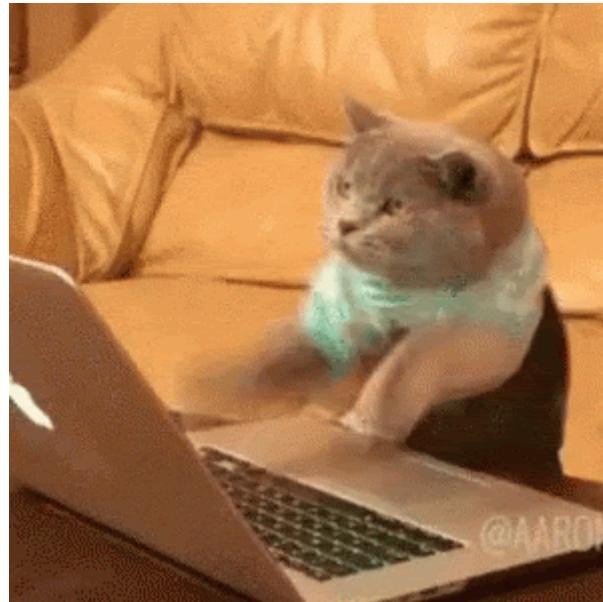
leaflet

- Biblioteca Leaflet
- Tutorial Leaflet

highmaps

- Galeria Highmaps
- Documentação Highmaps

Vamos ao R!



Fim!

