

# Visualização de dados



Outubro de 2022

# Sobre a Curso-R

# A empresa

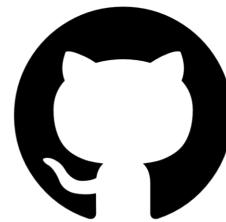


# Filosofia de código aberto!

## Livros



## Material dos cursos



Confira o  
nossa GitHub

## Lives



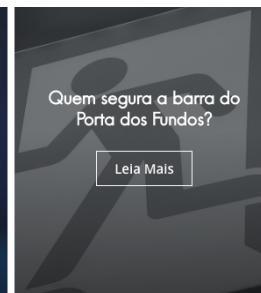
TODA  
QUARTA,  
ÀS 20H

## 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 especificada nas últimas aulas do curso.
- 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.

# 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.

# 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 quê 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.

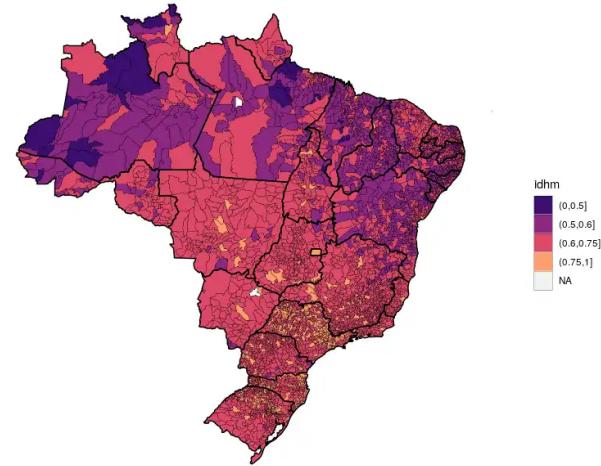


Imagen do [blog da Curso-R](#)

# Para quê 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***.

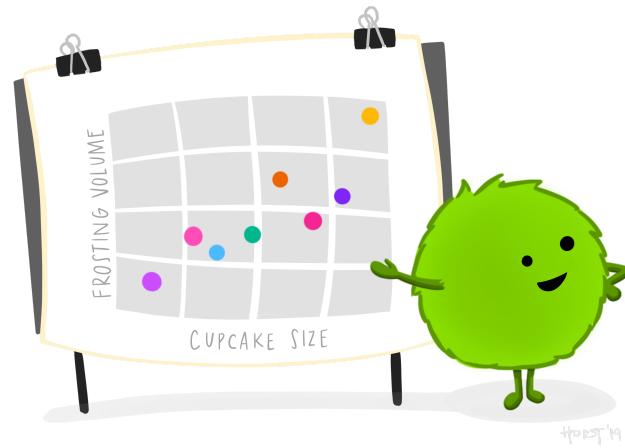


Ilustração de [Allison Horst](#)

# 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
- ...

Vamos montar uma lista aqui!

# Ferramental

# Em que momentos utilizamos?

Importar



Arrumar →

(Armazenar os dados  
consistentemente)

Transformar

(Criar novas variáveis e  
agregações)

Visualizar

(Surpreende, mas não é  
escalável)

Modelar

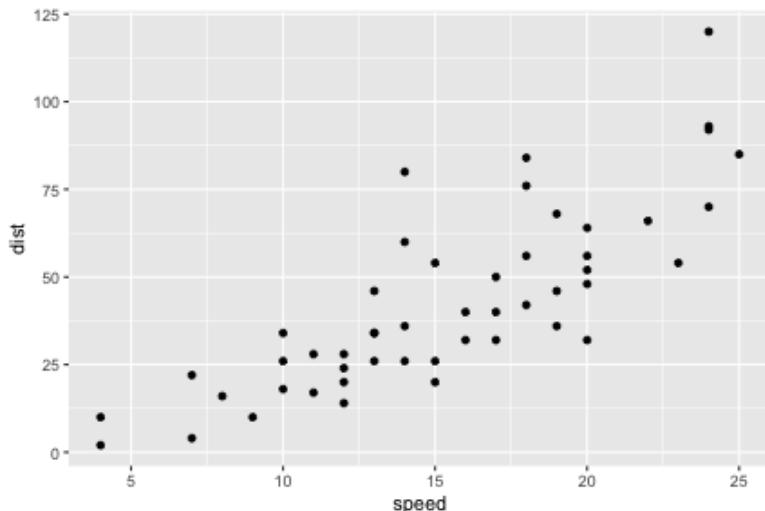
(É escalável, mas não  
surpreende)

Comunicar

Automatizar

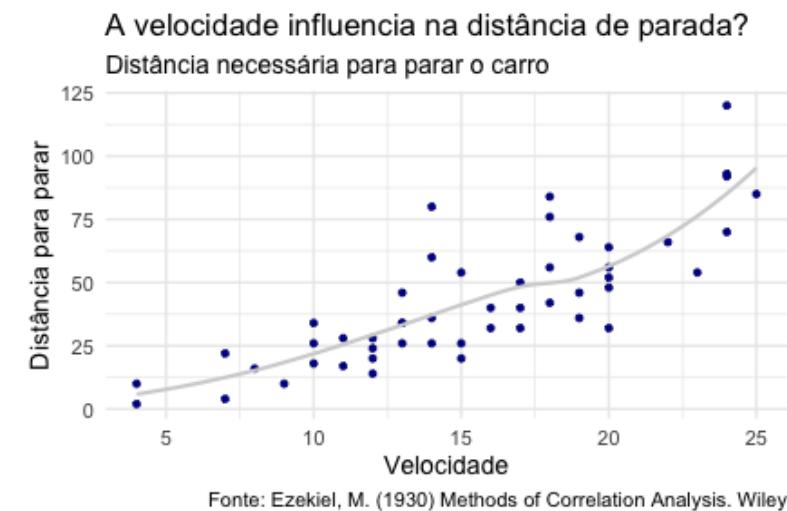
# Análise exploratória

- 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!

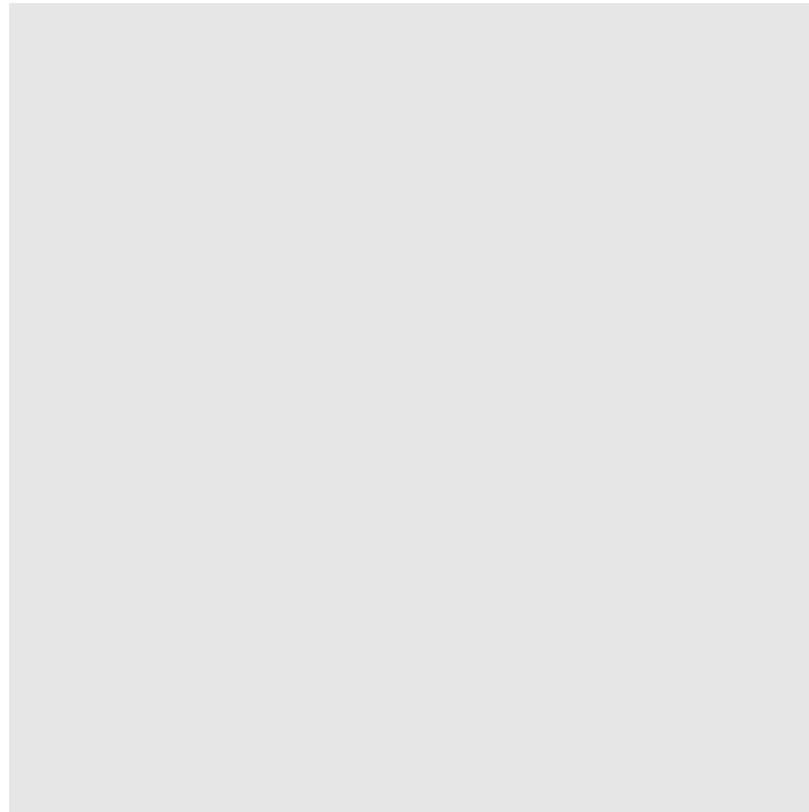
# Gramática de gráficos *em camadas*

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

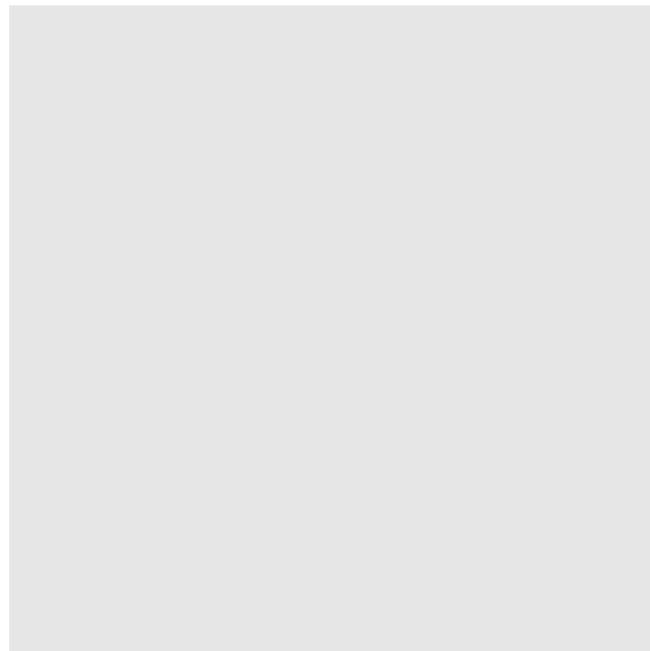
Canvas (R)

Eixos

Eixos (R)

Geometria

Geometria (R)



# Camadas

Depois, podemos trabalhar a estética com temas e detalhamentos.

---

Completo

Completo (R)

# Curiosidade: por quê 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 years 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> + (mapping = aes(<MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

required  
Not required, sensible defaults supplied

`ggplot(data = mpg, aes(x = cyl, y = hwy))` Begins a plot that you 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(lyend = 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)) -> 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_spline(aes(end = 1:115, 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
c + geom_freqpoly()
x, y, alpha, color, group, linetype, size
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(frt))
d + geom_bar()
x, alpha, color, fill, linetype, size, weight
```

#### TWO VARIABLES both continuous

```
e + geom_label(aes(label = cyl), 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_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(cyl, 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
f + geom_violin(scale = "area")
x, y, alpha, color, fill, group, linetype, size, weight
```

#### both discrete

```
g <- ggplot(diamonds, aes(cut, color))
g + geom_count()
x, y, alpha, color, fill, shape, size, stroke
c + geom_histogram()
x, y, alpha, color, fill, linetype, size, weight
c2 + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size
```

#### THREE VARIABLES

```
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
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
```



#### continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

```
h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight
h + geom_density_2d()
x, y, alpha, color, group, linetype, size
h + geom_hex()
x, y, alpha, color, fill, size
```

#### continuous function

```
i <- ggplot(economics, aes(date, unemploy))
i + geom_area()
x, y, alpha, color, fill, linetype, size
i + geom_line()
x, y, alpha, color, group, linetype, size
i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size
```

#### visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4.5, se = 1.2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
j + geom_crossbar(fatten = 2) -> x, y, ymax,
ymin, alpha, color, fill, group, linetype, size
j + geom_errorbar()
x, y, max, ymin, alpha, color, group, linetype, size
Also geom_errorbarh().
```

```
j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size
j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size
```

#### maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
k + geom_map(aes(map_id = state), map = map)
  + expand_limits(x = map$long, y = map$lat)
map_id, alpha, color, fill, linetype, size
```

# Qual gráfico escolher?

from Data to Viz

EXPLORE STORY ALL CAVEATS POSTER ABOUT CONTACT



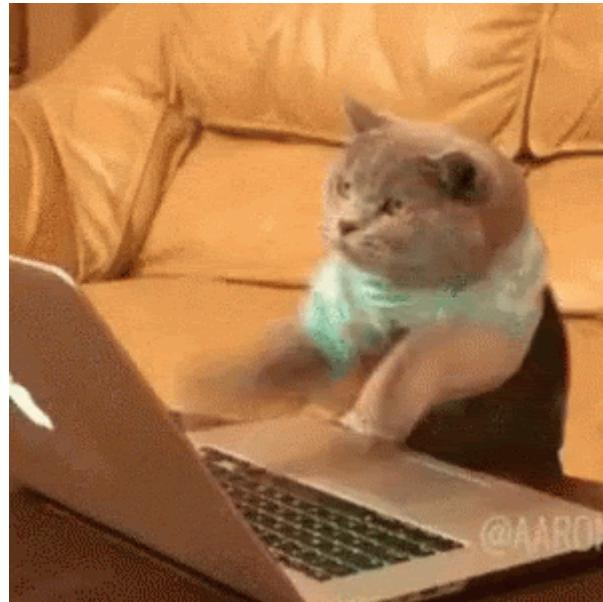
from Data to Viz

*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

25 / 66

# 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

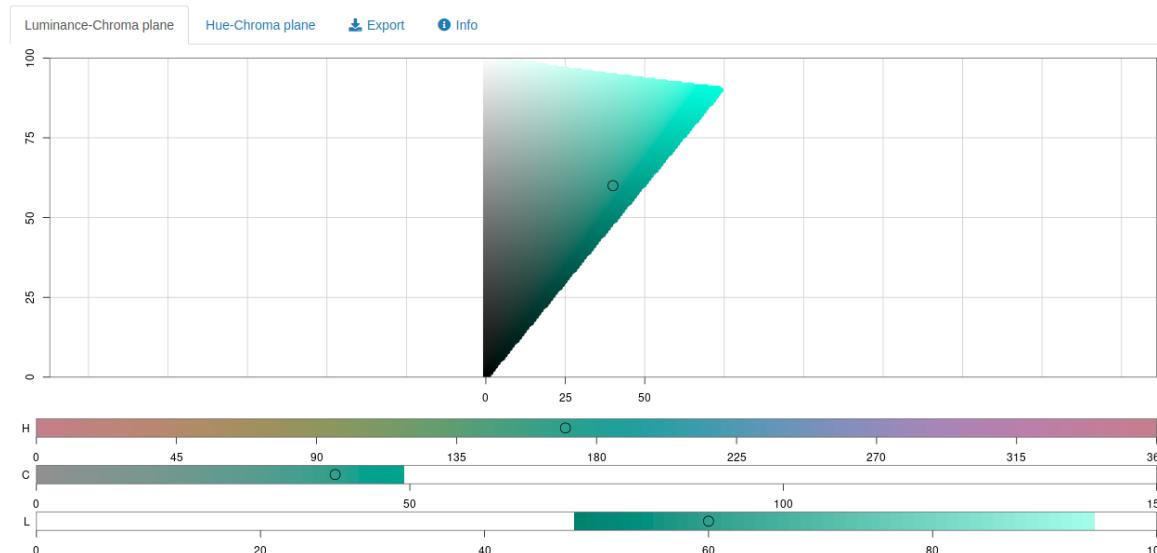
Depois você lerá isso

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}

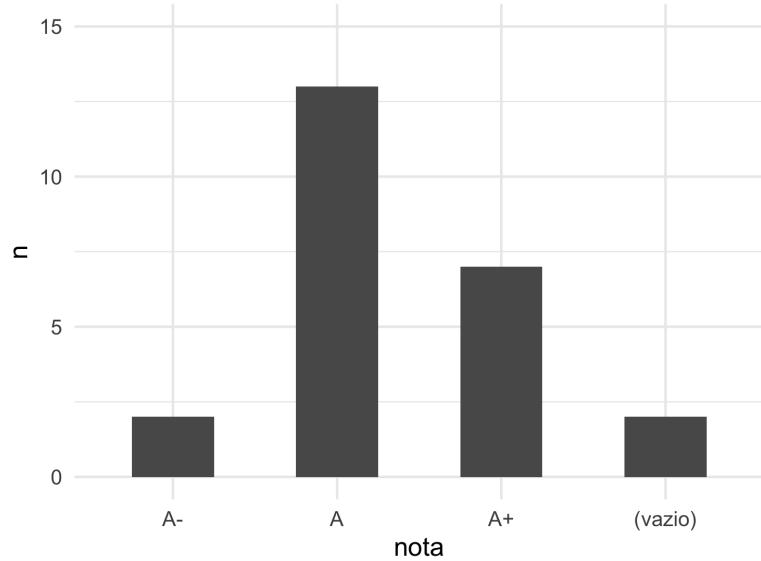
- `scale_*_brewer()`: utilizada para variáveis discretas. Possui três tipos: divergente, qualitativa e sequencial.
- `scale_*_distiller()`: utilizada para variáveis contínuas. Interpola as cores do *brewer* para lidar com todos os valores.
- `scale_*_fermenter()`: utilizada para variáveis contínuas, 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

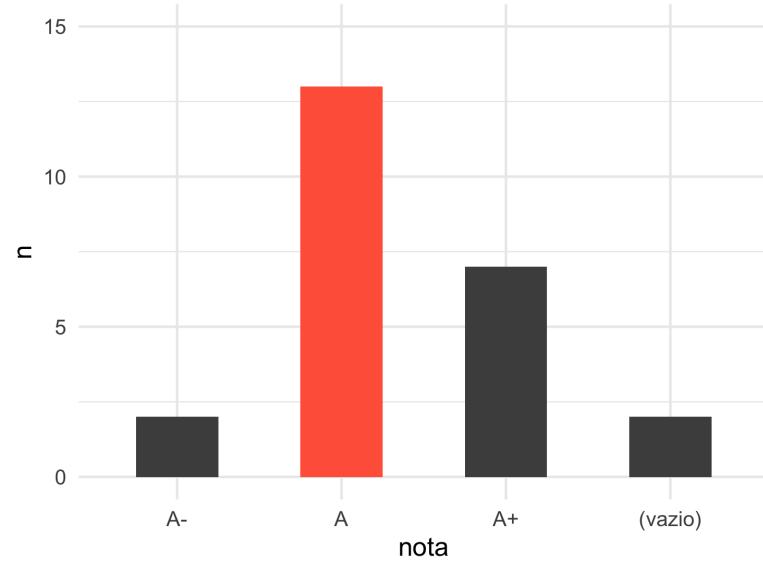
- {ggthemr} tem um monte de paletas, mas está um pouco desatualizado.
- {hrbrthemes} contém uma lista de temas escolhidos pelo Bob Rudis.
- {ghibli} tem paletas de cores relacionadas ao Studio Ghibli
- {paletteer} tem uma coleção de cores de vários outros pacotes de paletas.

# 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

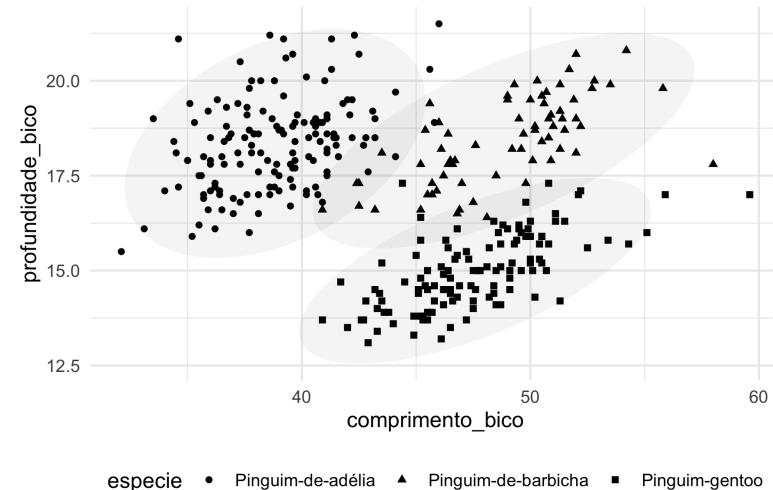
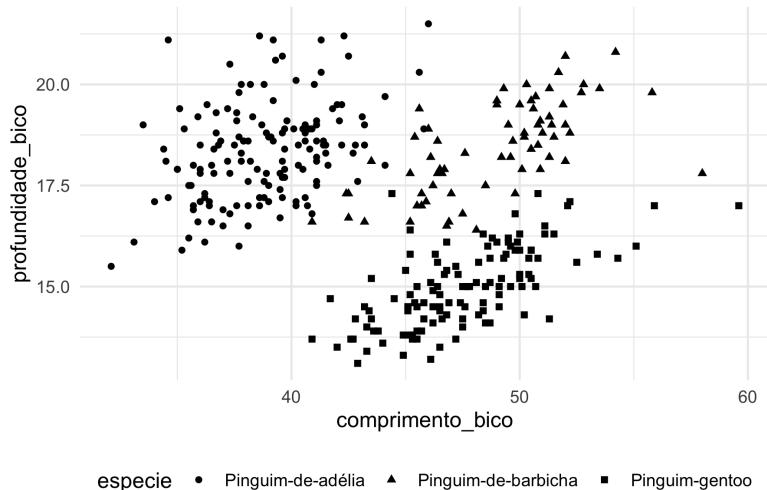
- Geralmente queremos contrastar formas para chamar a atenção...
- Sem com isso fazer uma bagunça visual.
- A ideia é utilizar apenas um elemento, como tamanho, forma, largura, marcações, angulações, etc. para mostrar o contraste.

# Exemplos

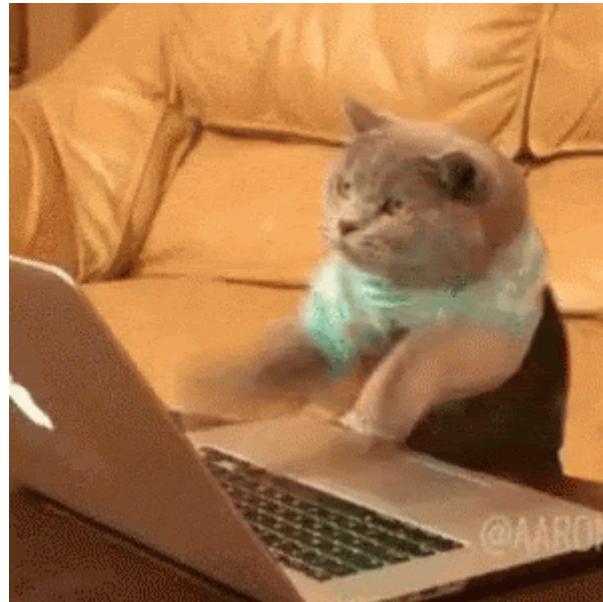
# Princípios Gestalt

- **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.

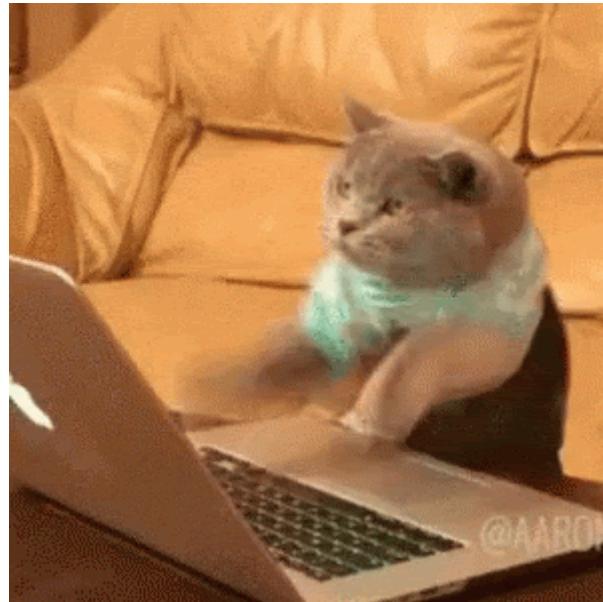
# O que é uma extensão?

- 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

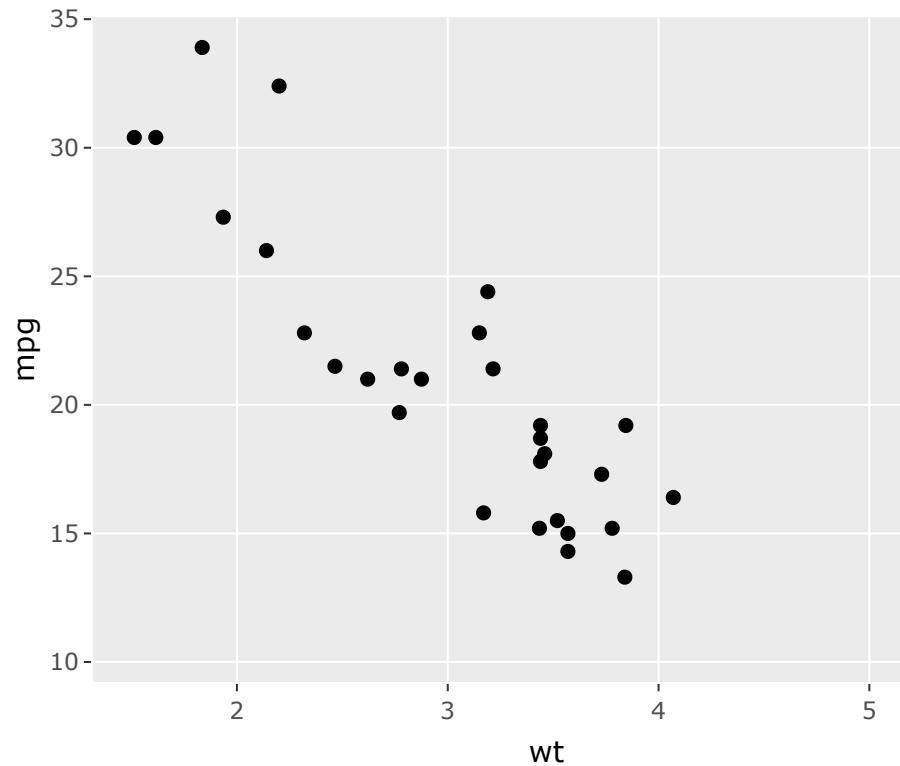
O pacote `plotly` nos permite criar gráficos interativos baseados na biblioteca `Plotly` (construída em `D3`).

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

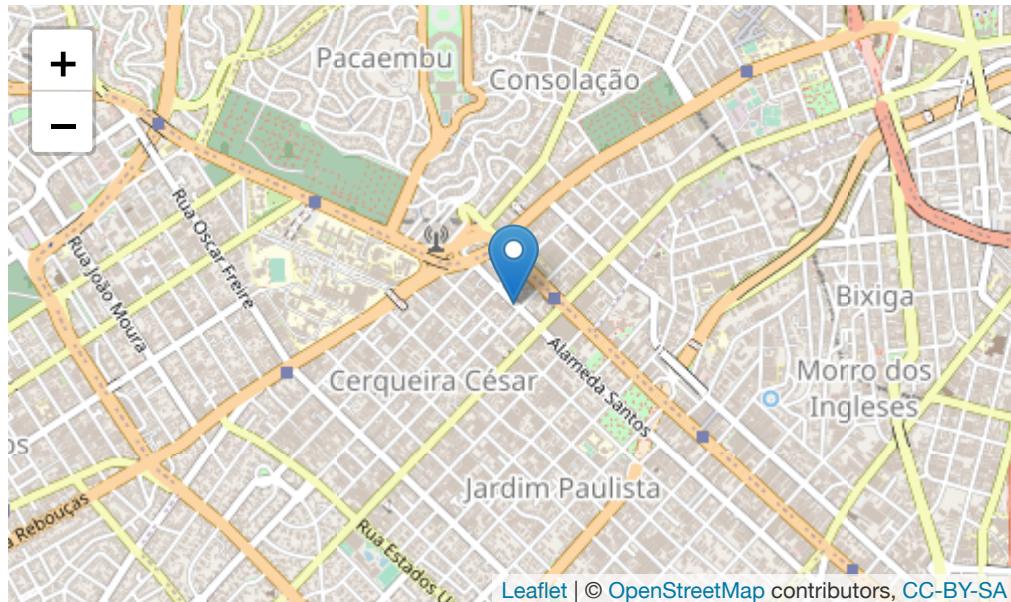
O pacote {leaflet} nos permite criar mapas interativos baseados na biblioteca JavaScript open-source [Leaflet](#).

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

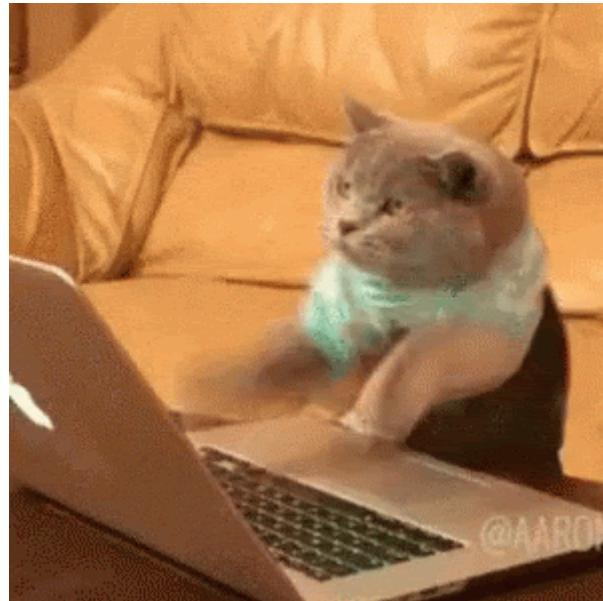
## **leaflet**

- Biblioteca Leaflet
- Tutorial Leaflet

## **highmaps**

- Galeria Highmaps
- Documentação Highmaps

# Vamos ao R!



Fim!