

# 3.6 Live Tutorial

Plotting data in R

# Tutorial Materials

[http://bit.ly/BA\\_R\\_Demo](http://bit.ly/BA_R_Demo)

**HELLO**  
my name is

Ryo

- [r.sakai@imperial.ac.uk](mailto:r.sakai@imperial.ac.uk)
- works at GlaxoSmithKline (Data Visualisation Lead)
- PhD in Data Visualisation in Bioinformatics

[http://bit.ly/BA\\_R\\_Demo](http://bit.ly/BA_R_Demo)

# Overview

- Introduction to R Markdown
  - HTML output for assignments
- Introduction to ggplot2 (tidyverse)
  - geometric objects
  - layered grammar of graphics
  - statistical transformation
  - position adjustment
- Other topics
  - Data transformation
  - Network visualization
  - Interactive data visualisation (plotly)

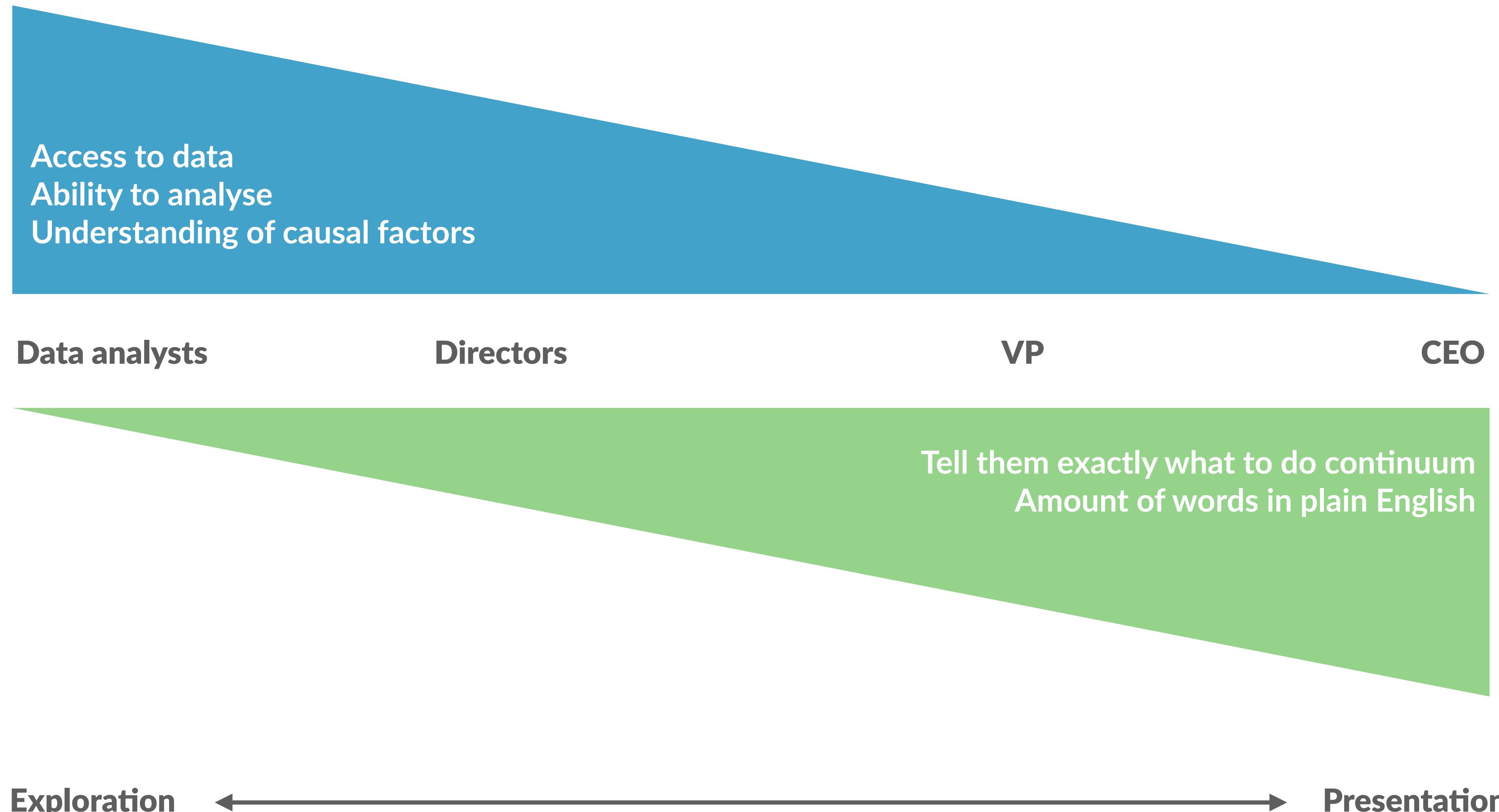


[http://bit.ly/BA\\_R\\_Demo](http://bit.ly/BA_R_Demo)

# Setting the expectations

- Introduction to visual analytics in R
- Ability to:
  - apply learning/theories from lectures
  - apply Data transformations
  - create static data visualisation to understand the data
  - refine visual outputs for communication
  - use visualisation techniques (e.g. interactive) for visual analytics
- Introduction to other data visualisation systems and libraries
  - Tableau, D3.js, jupyter notebook, etc
- Vigilant and resourceful analysts in R

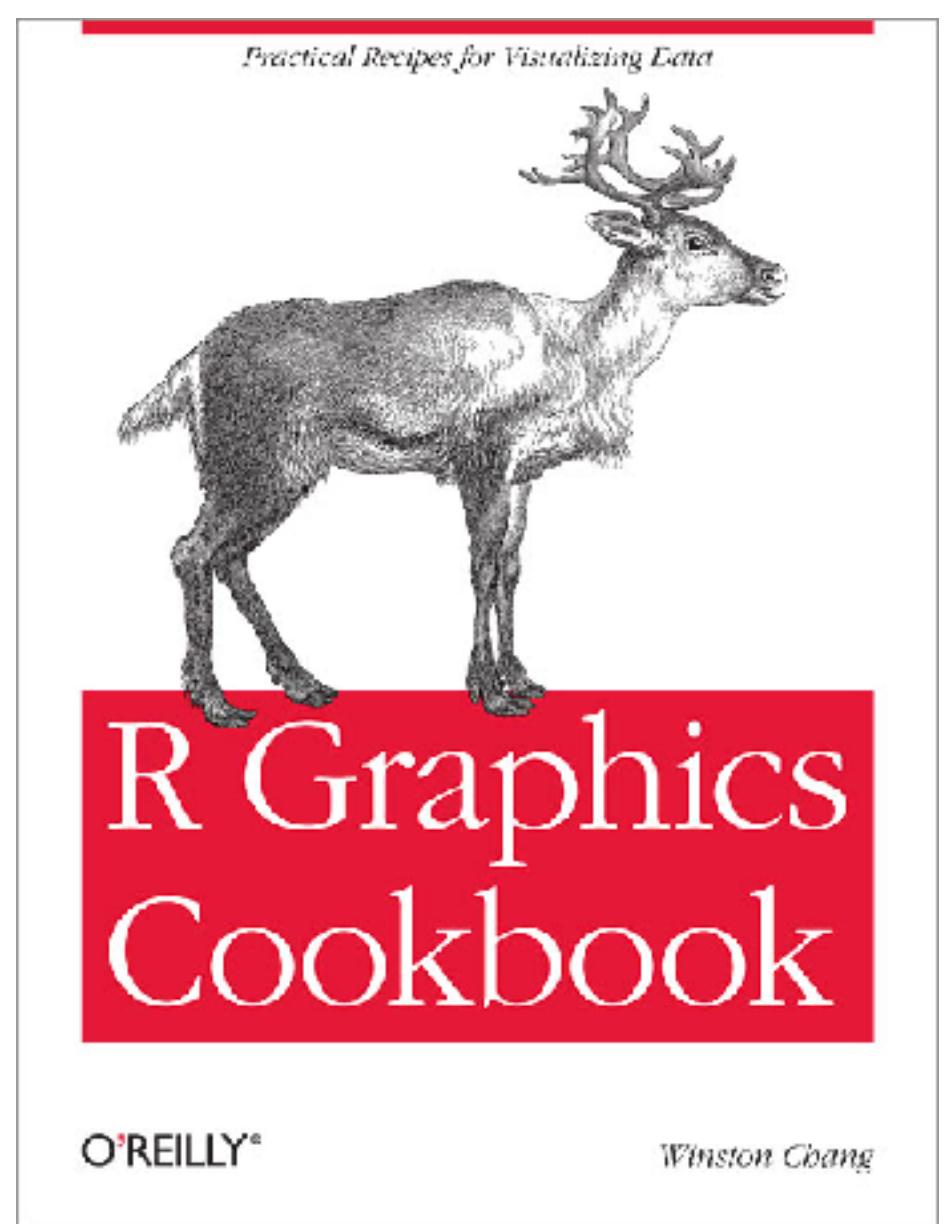
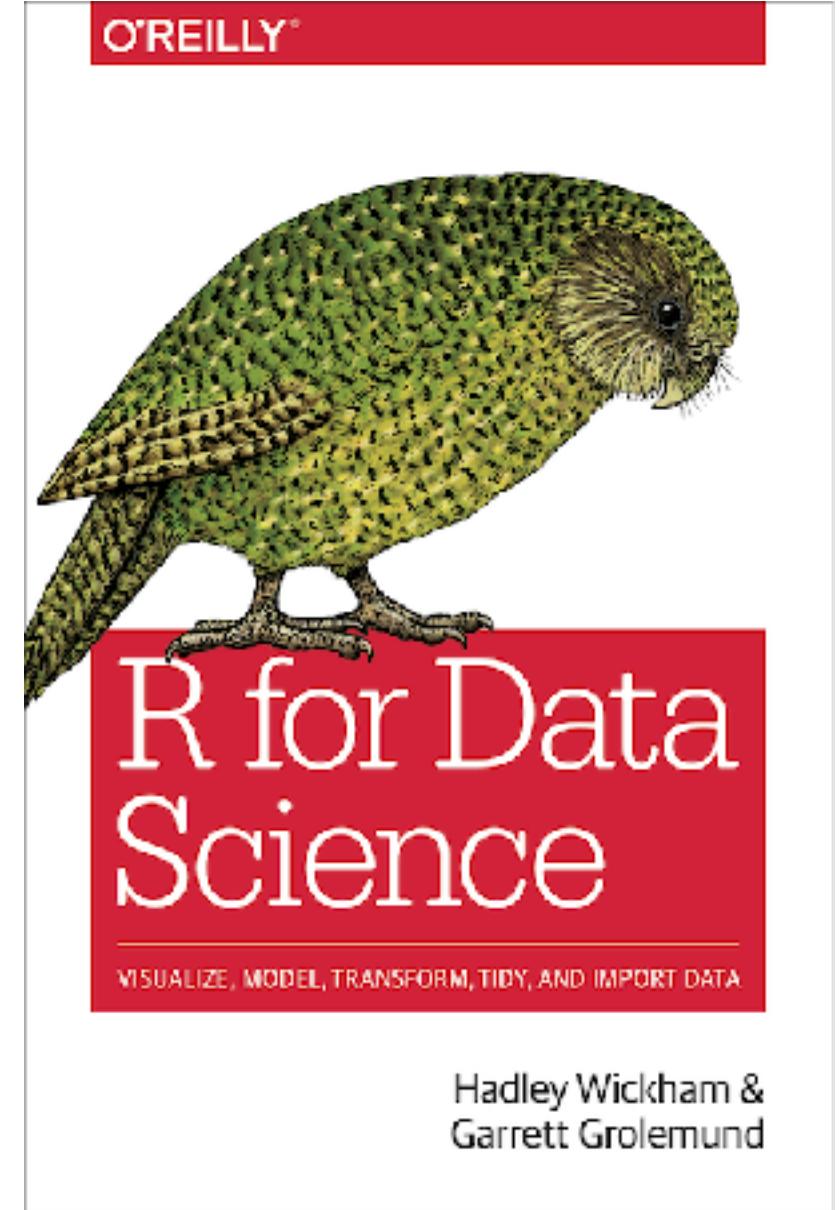
# designed for whom?



[http://bit.ly/BA\\_R\\_Demo](http://bit.ly/BA_R_Demo)

# Readings

- H Wickham and G Grolemund. R for Data Science. O'Reilly Media, 2017, <http://r4ds.had.co.nz/>
  - Ch 5: Data Transformation
  - Ch 12: Tidy Data
- Winston Chang, R Graphics Cookbook 2e. O'Reilly Media.  
<https://r-graphics.org>
  - Optional. Useful book.



[http://bit.ly/BA\\_R\\_Demo](http://bit.ly/BA_R_Demo)

# Useful Links

- ggplot2 documentation
  - <https://ggplot2.tidyverse.org/reference/>
- Other free books online on R
  - Featured books: <https://bookdown.org>
- Network visualisation tutorial (5.4)
  - igraph : perhaps most commonly used, high level of customisation
  - [Tutorial](#) by Katya Ognyanova

# Course style

- Lots of hands-on coding exercises
  - Learn by typing, not copy-pasting
- Exercises are not evaluated and optional
  - but highly recommended

# R Markdown

unified authoring framework for data science

# R Markdown

- an unified framework for data science:
  - combining your codes, its results (visualisation) and your prose commentary
  - designed for communication, collaboration and doing data science
- comes with RStudio, otherwise with **rmarkdown** package
- notebook interface, R Notebook
- outputs formats: HTML, PDF, Word, eBook, blog etc...



# R Markdown

YAML header  
settings for the whole document  
“yet another markup language”

The screenshot shows the RStudio interface with an R Markdown file open. The left pane displays the R Markdown code, and the right pane shows the generated HTML output.

**YAML header:**

```
1 ---  
2 title: "RMarkdown_Introduction"  
3 author: "Ryo Sakai"  
4 date: "13/01/2019"  
5 output: html_document  
6 ---
```

**Content:**

```
7   
8 ```{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10   
11 ## R Markdown  
12   
13 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.  
14   
15 When you click the Knit button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:  
16   
17 ```{r cars}  
18 summary(cars)  
19 ```  
20   
21 

|         | speed | dist           |
|---------|-------|----------------|
| Min.    | 4.0   | 2.00           |
| 1st Qu. | 12.0  | 1st Qu.: 26.00 |
| Median  | 15.0  | Median : 36.00 |
| Mean    | 15.4  | Mean : 42.98   |
| 3rd Qu. | 19.0  | 3rd Qu.: 56.00 |
| Max.    | 25.0  | Max. : 120.00  |

  
22 ## Including Plots  
23   
24 You can also embed plots, for example:  
25   
26 ```{r pressure, echo=FALSE}  
27 plot(pressure)
```

**Preview:**

Environment is empty

# R Markdown

R code chunk

R code chunk

R code output

R code chunk

The screenshot shows the RStudio interface with an R Markdown file open. The code editor displays the following R Markdown code:

```
1 ---  
2 title: "RMarkdown_Introduction"  
3 author: "Ryo Sakai"  
4 date: "13/01/2019"  
5 output: html_document  
6 ---  
7   
8 ```{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10 ```  
11   
12 ## R Markdown  
13   
14 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on  
15 using R Markdown see <http://rmarkdown.rstudio.com>.  
16 When you click the Knit button a document will be generated that includes both content as well as the output of any embedded R code  
17 chunks within the document. You can embed an R code chunk like this:  
18 ```{r cars}  
19 summary(cars)  
20 ```  
21   
22 speed dist  
Min. : 4.0 Min. : 2.00  
1st Qu.:12.0 1st Qu.: 26.00  
Median :15.0 Median : 36.00  
Mean :15.4 Mean : 42.98  
3rd Qu.:19.0 3rd Qu.: 56.00  
Max. :25.0 Max. :120.00  
23   
24 ## Including Plots  
25 You can also embed plots, for example:  
26 ```{r pressure, echo=FALSE}  
27 plot(pressure)  
12:14 R Markdown
```

The code editor has three red boxes highlighting specific sections: one around the first R code chunk, one around the R code output (the summary of the 'cars' dataset), and one around the second R code chunk.

The global environment pane shows "Environment is empty". The viewer pane shows the summary of the 'cars' dataset:

	speed	dist
Min. :	4.0	2.00
1st Qu.:	12.0	26.00
Median :	15.0	36.00
Mean :	15.4	42.98
3rd Qu.:	19.0	56.00
Max. :	25.0	120.00

# R Markdown

Markdown  
Your notes/prose commentary

The screenshot shows the RStudio interface with an R Markdown file open. The file contains R code and prose commentary. Two sections of the prose are highlighted with red boxes:

```
1 ---  
2 title: "RMarkdown_Introduction"  
3 author: "Ryo Sakai"  
4 date: "13/01/2019"  
5 output: html_document  
6 ---  
7  
8 ```{r setup, include=FALSE}  
9 knitr::opts_chunk$set(echo = TRUE)  
10  
11 ## R Markdown  
12  
13 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on  
14 using R Markdown see <http://rmarkdown.rstudio.com>.  
15  
16 When you click the Knit button a document will be generated that includes both content as well as the output of any embedded R code  
17 chunks within the document. You can embed an R code chunk like this:  
18  
19 ```{r cars}  
20 summary(cars)  
21  
22 speed dist  
23 Min. :4.0 Min. : 2.00  
24 1st Qu.:12.0 1st Qu.: 26.00  
25 Median :15.0 Median : 36.00  
26 Mean :15.4 Mean : 42.98  
27 3rd Qu.:19.0 3rd Qu.: 56.00  
28 Max. :25.0 Max. :120.00  
29  
30  
31 ## Including Plots  
32  
33 You can also embed plots, for example:  
34  
35 ```{r pressure, echo=FALSE}  
36 plot(pressure)
```

The first highlighted section (lines 12-16) describes the basic functionality of R Markdown. The second highlighted section (lines 21-24) provides an example of embedding a plot.

# Your turn!

1. Create a new notebook using: ***File>New File >R Notebook***
  1. Read the instruction
  2. Practice running the code chunks
2. Create a new R Markdown document with ***File>New File>R Markdown...***
  1. Knit by clicking the “knit button” 
  2. Knit by keyboard short cut: **Cmd/Ctrl + Shift + K**
3. How do R notebook and R markdown files compare?
4. Create new R Markdown files to test 3 built-in formats:
  1. HTML
  2. PDF
  3. WORD

# Text format with Markdown

- a lightweight set of conventions for formatting plain text files
- Go to *Help > Markdown Quick Reference* on RStudio

## Text formatting

---

\*italic\* or italic  
\*\*bold\*\* bold  
`code`  
superscript<sup>2</sup> and subscript<sub>2</sub>

## Headings

---

# 1st Level Header  
## 2nd Level Header  
### 3rd Level Header

## Lists

---

\* Bulleted list item 1

# Code chunks

- Shortcut to insert code chunk: **Cmd/Ctrl + Alt + I**
- Codes are surrounded by ````{r}` and `````
- Shortcut to run code chunk: **Cmd/Ctrl + Shift + Enter**
- Options to set chunk name by ````{r by-name}`
- Chunk options:

Option	Run code	Show code	Output	Plots	Message	Warnings
<code>eval = FALSE</code>	X		X	X	X	X
<code>include = FALSE</code>		X	X	X	X	X
<code>echo = FALSE</code>		X				
<code>results = "hide"</code>			X			
<code>fig.show = "hide"</code>				X		
<code>message = FALSE</code>					X	
<code>warning = FALSE</code>						X

# Your turn!

1. Type the following code and knit your R Markdown.

```
```{r setup, include = FALSE}
library(ggplot2)
library(dplyr)

smaller <- diamonds %>%
  filter(carat <= 2.5)
```
```

We have data about `r nrow(diamonds)` diamonds. Only `r nrow(diamonds) - nrow(smaller)` are larger than 2.5 carats. The distribution of the remainder is shown below:

```
```{r, echo = FALSE}
smaller %>%
  ggplot(aes(carat)) +
  geom_freqpoly(binwidth = 0.01)
```
```

# Table output

- `knitr::kable` output nice tables
- For more functionalities and styling options, check:
  - `kableExtra` package
  - [https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome\\_table\\_in\\_html.html](https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html)

```
knitr::kable(  
  mtcars[1:5, ],  
  caption = "Motor Trend Car Data"  
)
```

# More information

- R Markdown cheat sheet [[link](#)]
- R Markdown: The definitive guide by Yihui Xie [[link](#)]
- RStudio website [[link](#)]

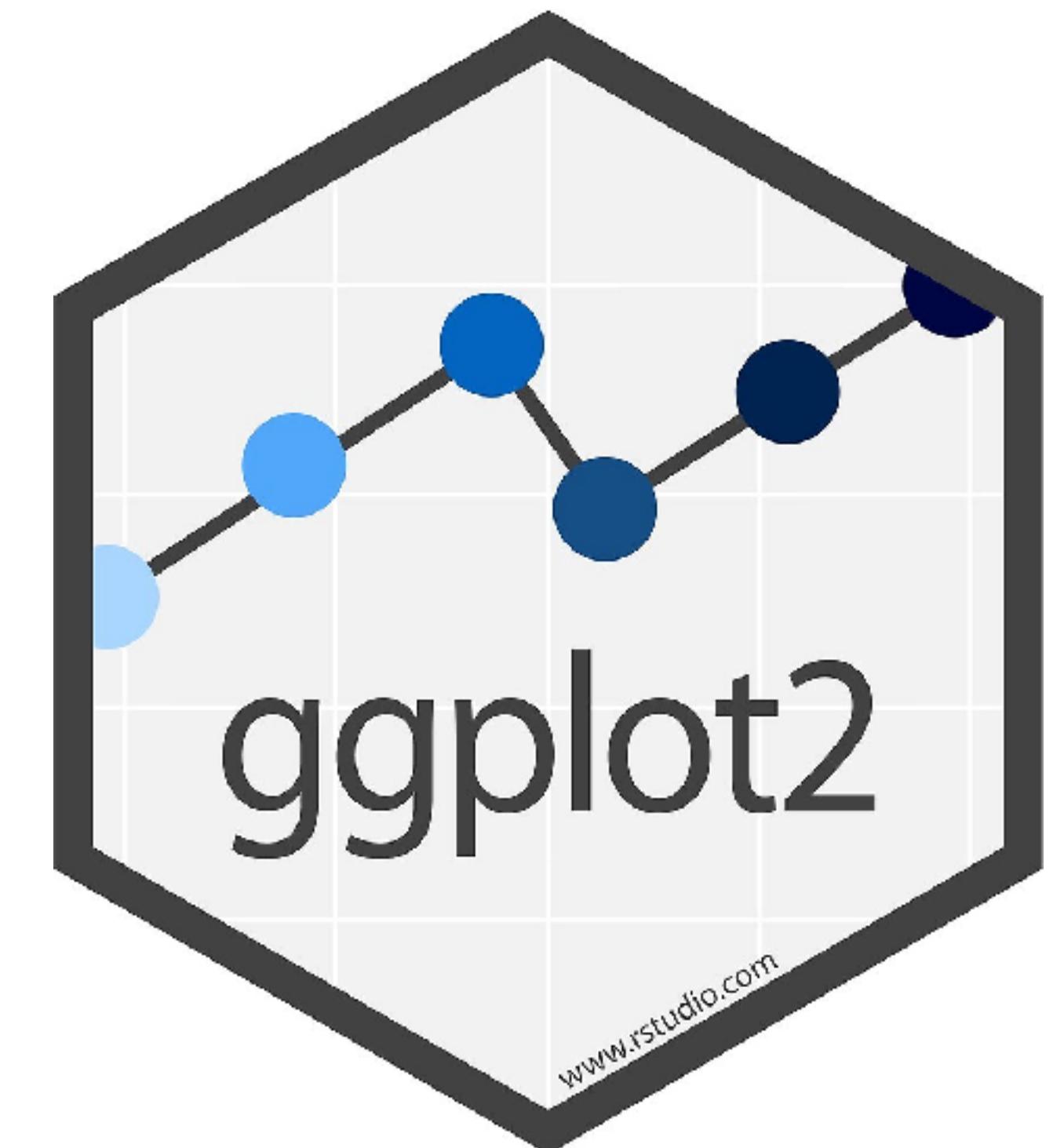
# Assignment Instruction

- Submitting R Markdown output (HTML)
  - 6.4 Proportinal symbol map
- Submiting images from ggplot2
  - 4.3 Box plot
  - 4.5 Heatmap
  - 4.7 Scatter plot with icons

# Introduction to ggplot2

# ggplot2

- R has several systems for making graphs (Base R, lattice, etc...)
- **ggplot2** is perhaps the most elegant and versatile
- Implemented based on the *grammar of graphics*
- part of **tidyverse** package



# tidyverse package

- a collection of R packages designed for data science

```
# Install if you have not install, or to update the package  
install.packages("tidyverse")
```

```
# Load the package  
library(tidyverse)
```

```
# We will use this dataset for this section
```

```
mpg
```



# Your turn!

1. Look up a vignette for the **tidyverse** package
  - Hint: function vignette(topic, package)
2. Look up the documentation on **mpg** dataset
  - **mpg** is a dataset that comes with ggplot2 package
    - Make sure you load the ggplot2 library before looking up
    - You can run either `library(ggplot2)`, or `library(tidyverse)`

# Solution

```
# if you do not specify the topic, it would list  
# all the topics available:  
vignette(package =“tidyverse”)  
  
# Selecting a specific topic:  
vignette("manifesto", package ="tidyverse")  
  
# Check the dataset:  
?mpg
```

# mpg dataset

- *Do cars with big engines use more fuel than cars with small engines?*

- What do you think the relationship between engine size and fuel efficiency is like?
- Is it a linear or non-linear function?

- Your turn!

- Look up which variable is for car's engine size in **mpg** dataset.
- Look up which variable is for car's fuel efficiency in **mpg** dataset.

`displ` variable for car's engine size in litres

`hwy` variable for car's fuel efficiency on the highway, in miles per gallon

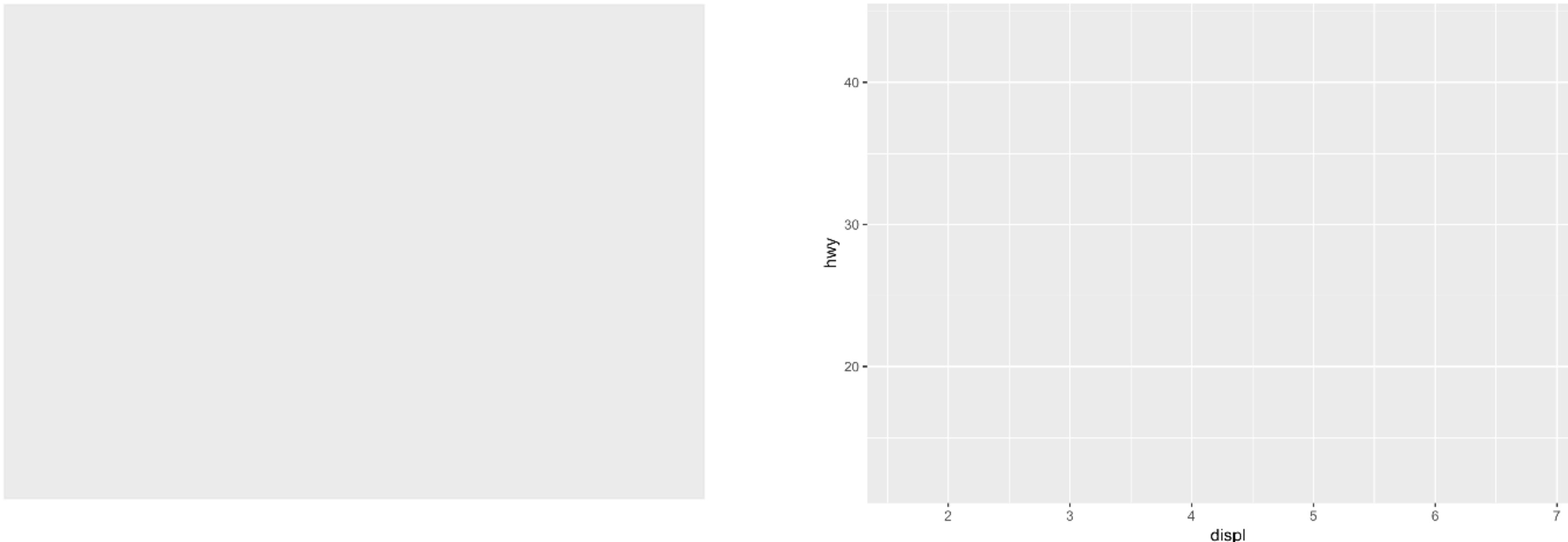
# `ggplot2::qplot()`

- equivalent to the `base plot()` function
- a convenient wrapper for creating a number of different types of plots using a consistent calling scheme

# Canvas

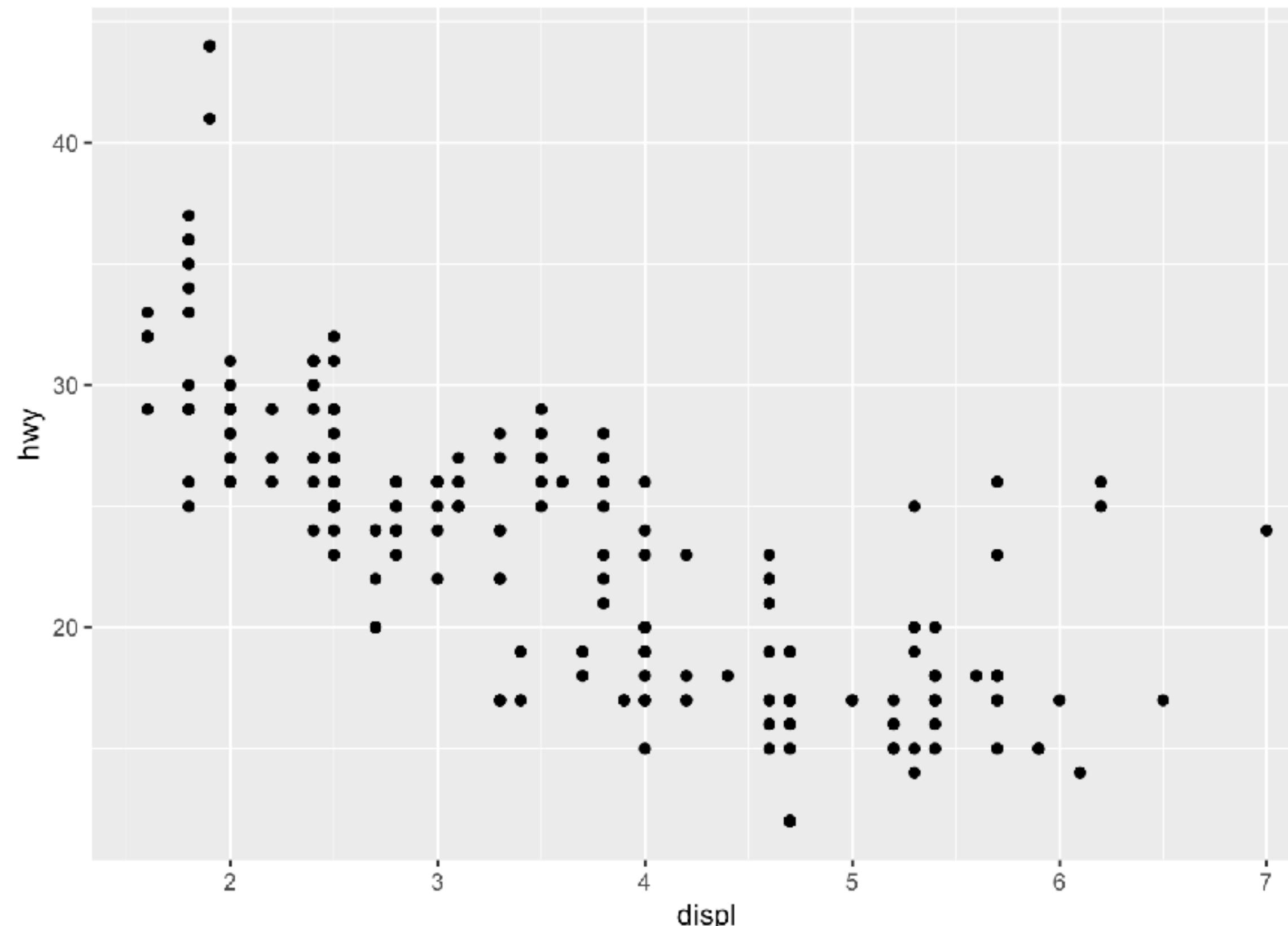
```
# Left
ggplot(data = mpg)

# right
ggplot(data = mpg, mapping = aes(x = displ, y = hwy))
```



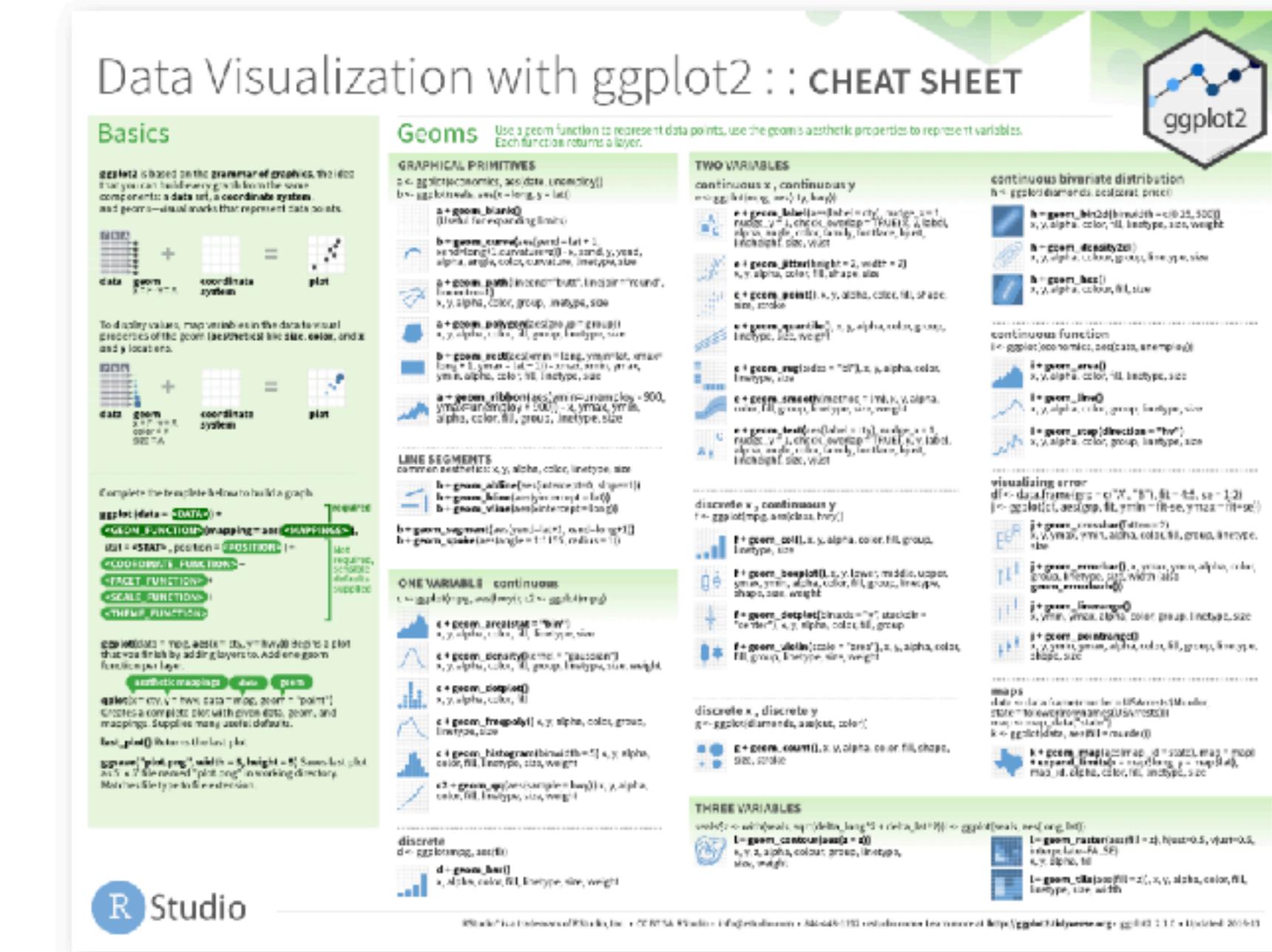
# First scatter plot

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point()
```



# Basic template

```
ggplot(data = <DATA>) +  
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```



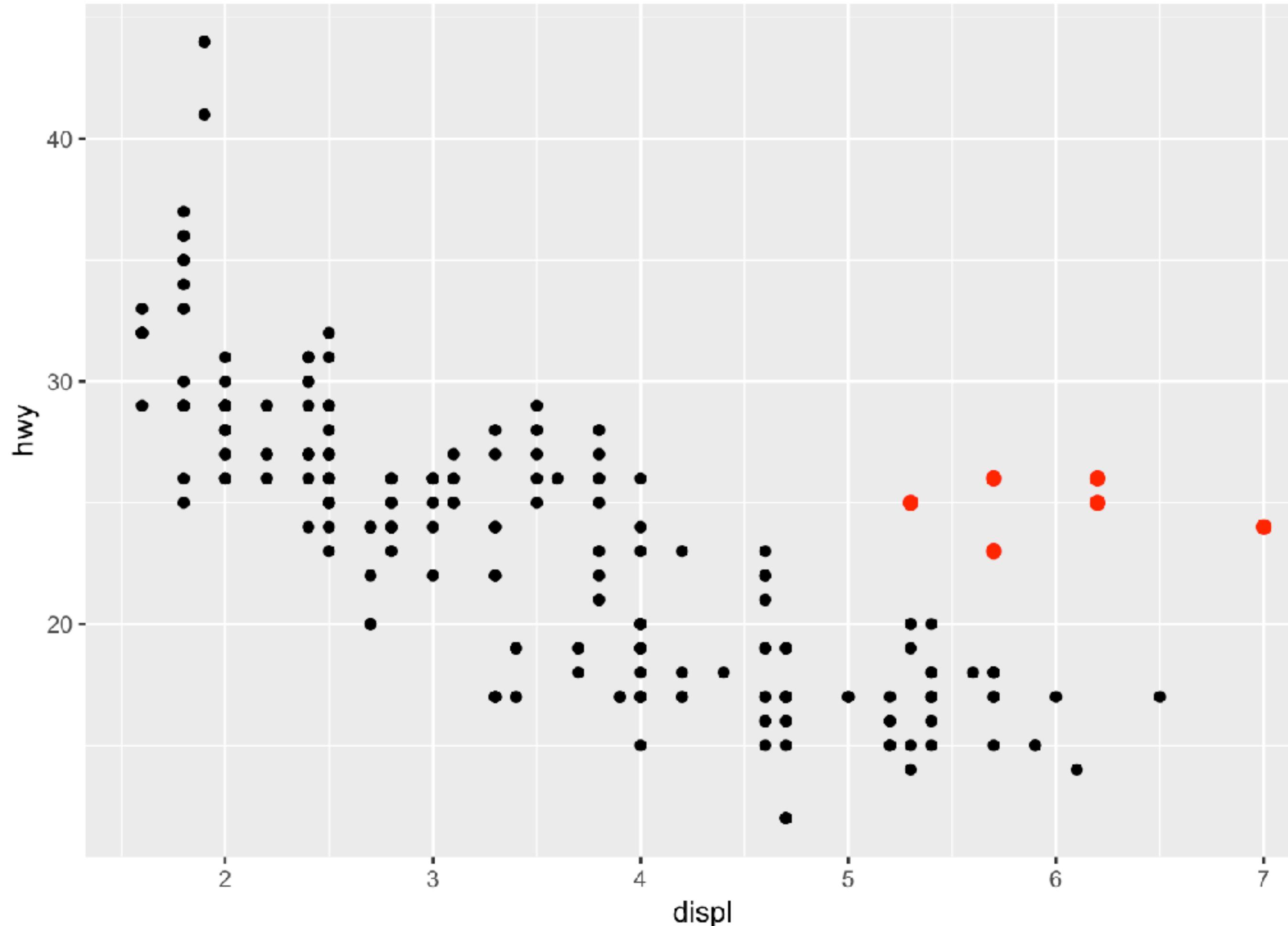
ggplot2 cheat sheet

# Your turn!

1. Run `ggplot(data = mpg)`. What do you see?
2. How many rows are in `mpg`? How many columns/variables?
3. What does the `drv` variable describe?
4. Make a scatter plot of `hwy` on the x-axis, `cyl` on y-axis.
5. Make a scatter plot of `class` vs. `drv`. Why is this plot not very useful?

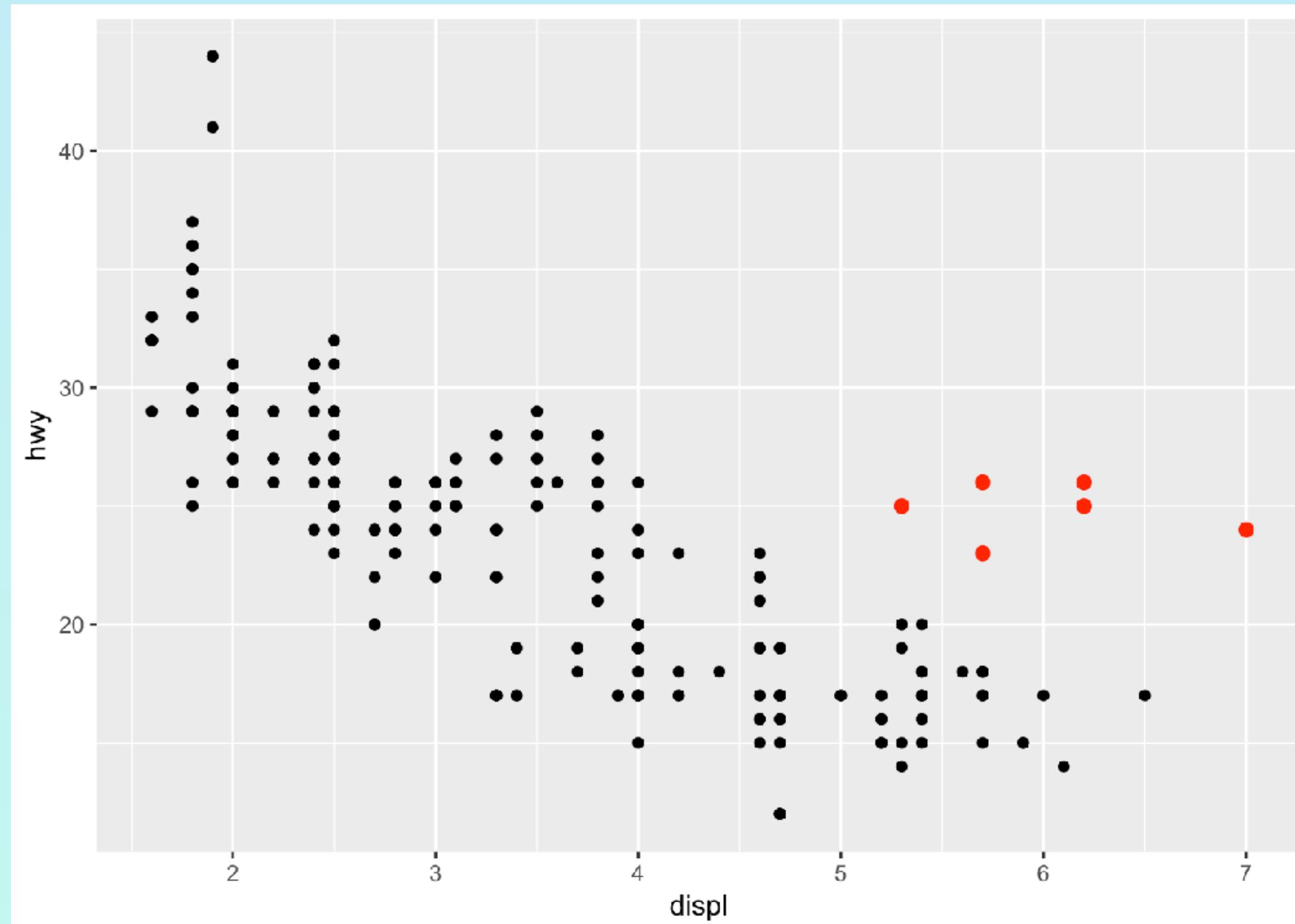
# Introduction to Visual Analytics

# Exploratory analysis



# Your turn!

1. Think about why this may be. Come up with hypotheses.
2. Look up `?mpg` to see which variable may help you address your queries.

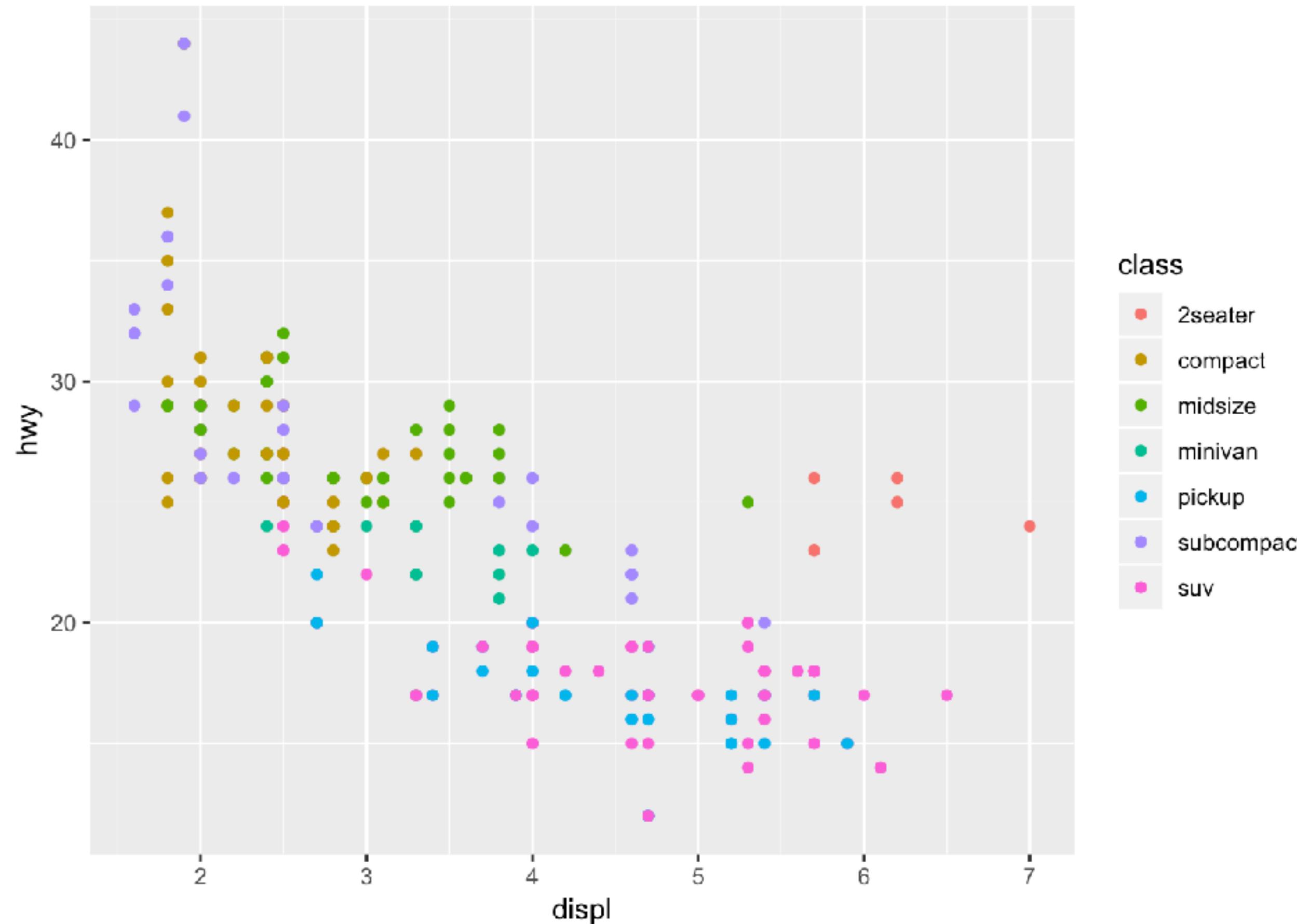


```
table(mpg$class)
```

| Var1       | Freq |
|------------|------|
| 2seater    | 5    |
| compact    | 47   |
| midsize    | 41   |
| minivan    | 11   |
| pickup     | 33   |
| subcompact | 35   |
| suv        | 62   |

# Categorical data

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color= class)) +  
  geom_point()
```



# Your turn!

1. Run the following code. Why are the points not blue?

```
ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy, color = 'blue'))
```

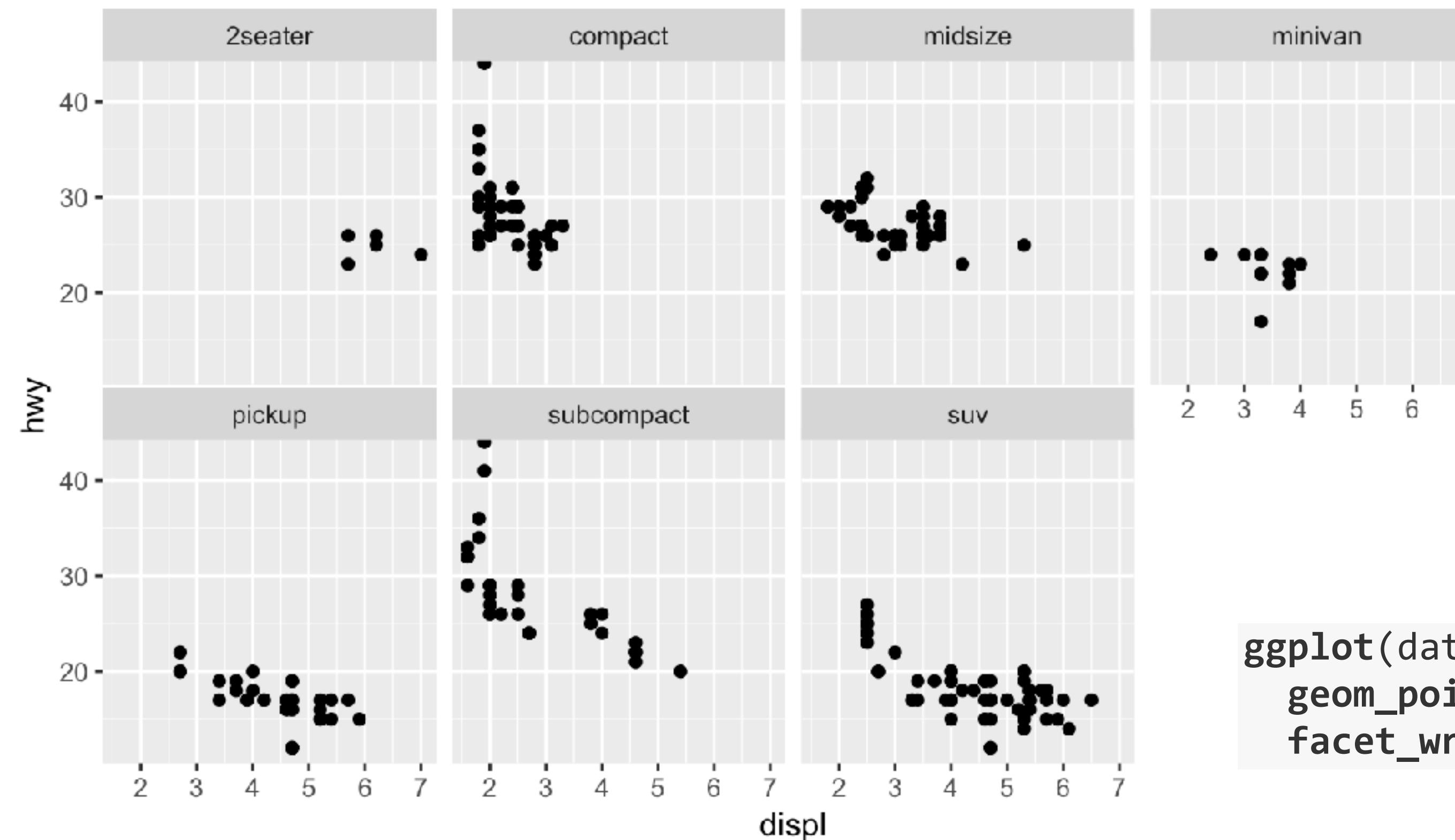
2. What happens if you map an aesthetic something other than a variable name, like `aes(color = displ < 5)`?

# Facets

- partition the data and create small multiple plots side by side
- facet by categorical variable
- **ggplot2** comes with 2 useful faceting functions:
  - `facet_wrap()`
  - `facet_grid()`

# Facets

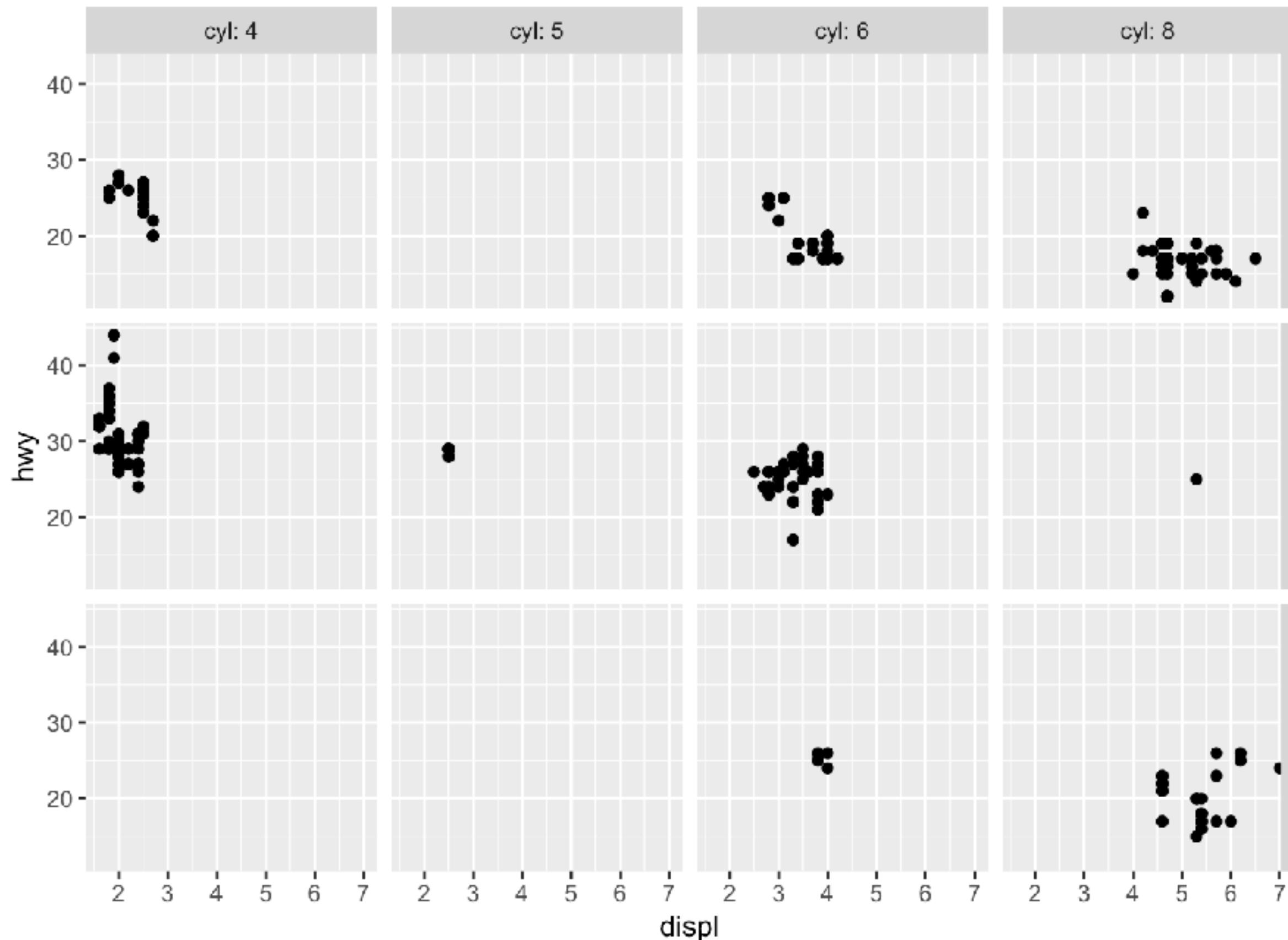
```
ggplot(data = mpg)+  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_wrap(~ class, nrow = 2)
```



```
ggplot(data = mpg)+  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_wrap(vars(class), nrow = 2)
```

# Facets

```
ggplot(data = mpg)+  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(drv ~ cyl, labeller = label_both)
```



```
ggplot(data = mpg)+  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(rows = vars(drv), cols = vars(cyl),  
             labeller = label_both)
```

# Your turn!

1. What happens if you facet on a continuous variable?
2. What does the following code produce? What does . do in the formula?

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(drv ~ ., labeller = label_both)
```

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(. ~ cyl, labeller = label_both)
```

# geoms

Geometric objects

# geoms

- ggplot2 uses geometric shapes to encode data
- ~30 built-in geoms, and more offered by other packages
- Type **geom\_** in console and hit the TAB key...
- Check out the ggplot2 cheatsheet from RStudio!

geom\_abline

geom\_bar

geom\_bin2d

geom\_blank

geom\_boxplot

geom\_contour

geom\_count

geom\_hex

geom\_crossbar

geom\_density

geom\_density\_2d

geom\_dotplot

geom\_errorbarh

geom\_freqpoly

geom\_histogram

geom\_jitter

geom\_label

geom\_map

geom\_path

geom\_point

geom\_polygon

geom\_quantile

geom\_raster

geom\_ribbon

geom\_rug

geom\_segment

geom\_smooth

geom\_violin

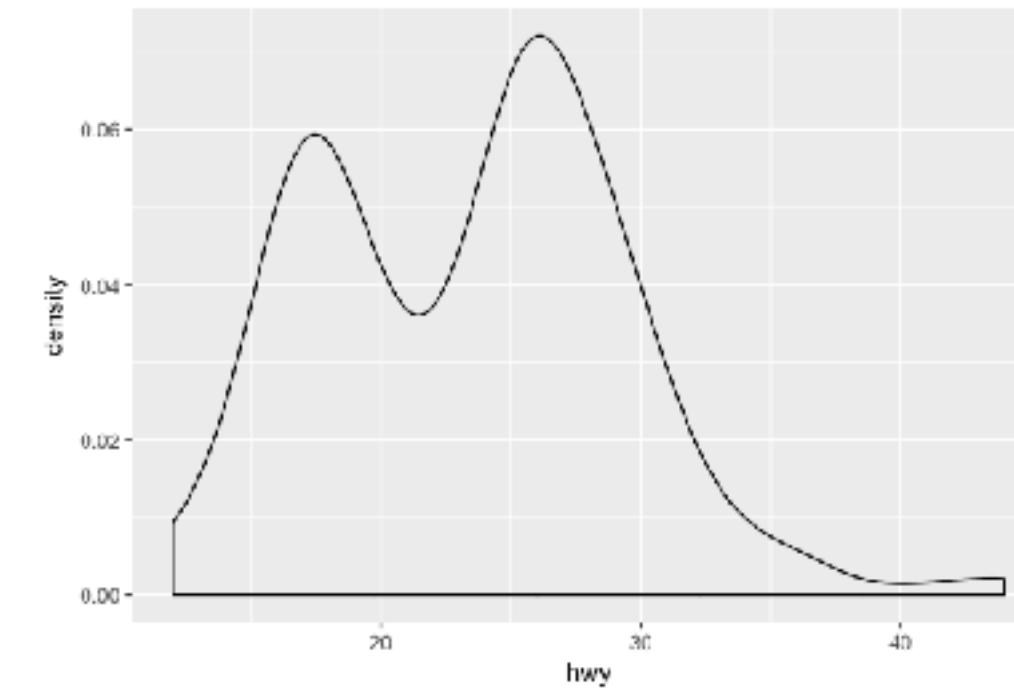
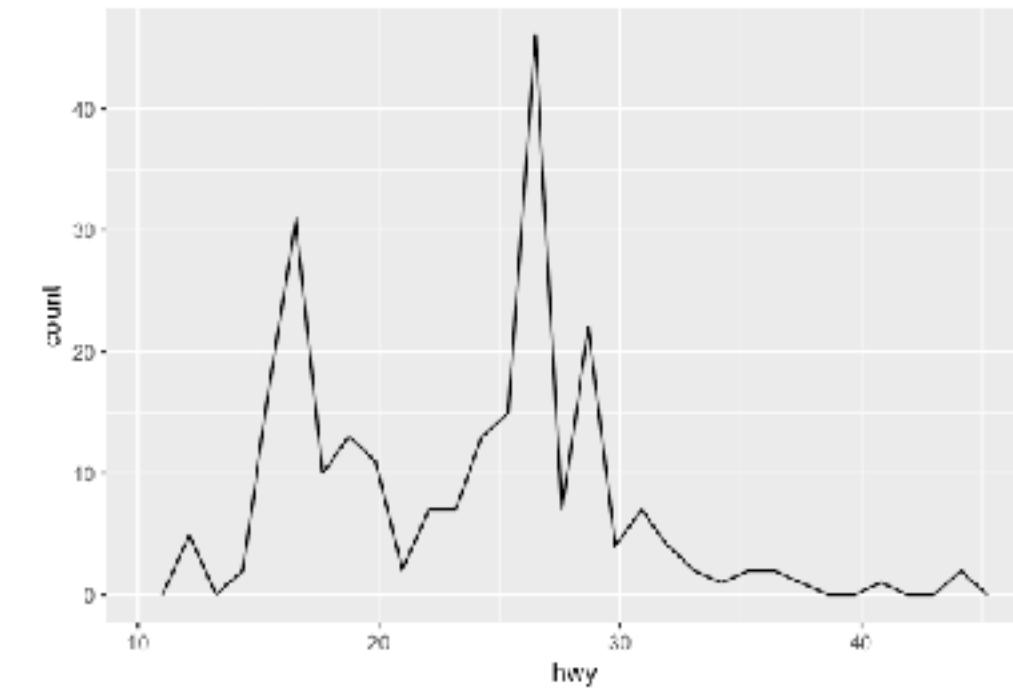
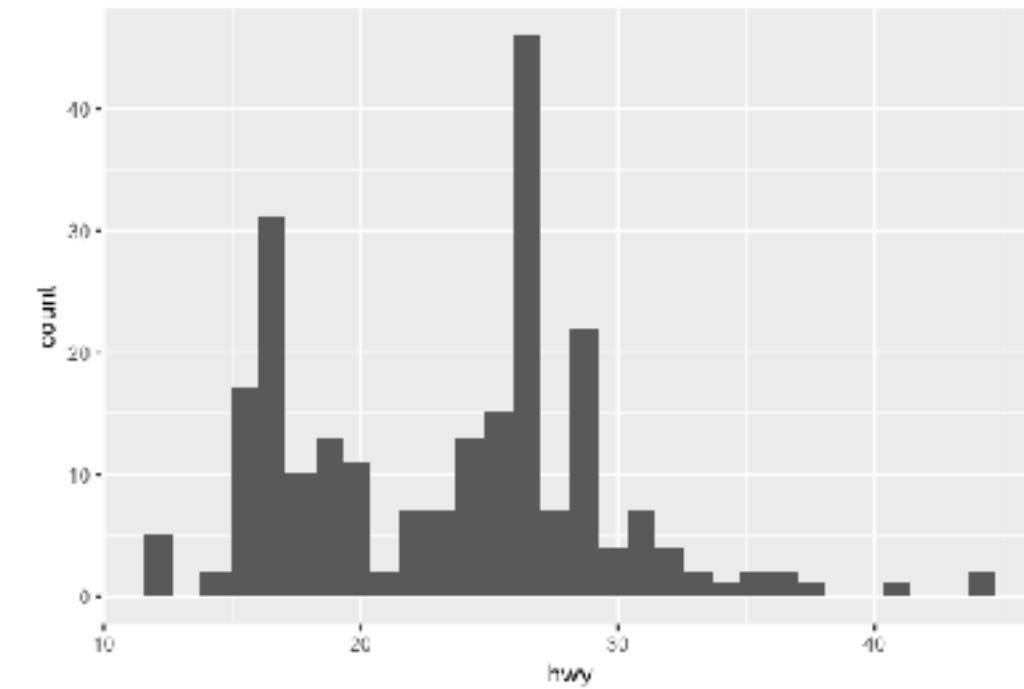
# continuous univariate

aesthetic mapping argument

```
ggplot(data = mpg, mapping = aes(x = hwy)) +  
geom_histogram()
```

```
ggplot(data = mpg, mapping = aes(x = hwy)) +  
geom_freqpoly()
```

```
ggplot(data = mpg, mapping = aes(x = hwy)) +  
geom_density()
```



geom\_abline

geom\_bar

geom\_bin2d

geom\_blank

geom\_boxplot

geom\_contour

geom\_count

geom\_hex

geom\_crossbar

**geom\_density**

geom\_density\_2d

geom\_dotplot

geom\_errorbarh

**geom\_freqpoly**

**geom\_histogram**

geom\_jitter

geom\_label

geom\_map

geom\_path

geom\_point

geom\_polygon

geom\_quantile

geom\_raster

geom\_ribbon

geom\_rug

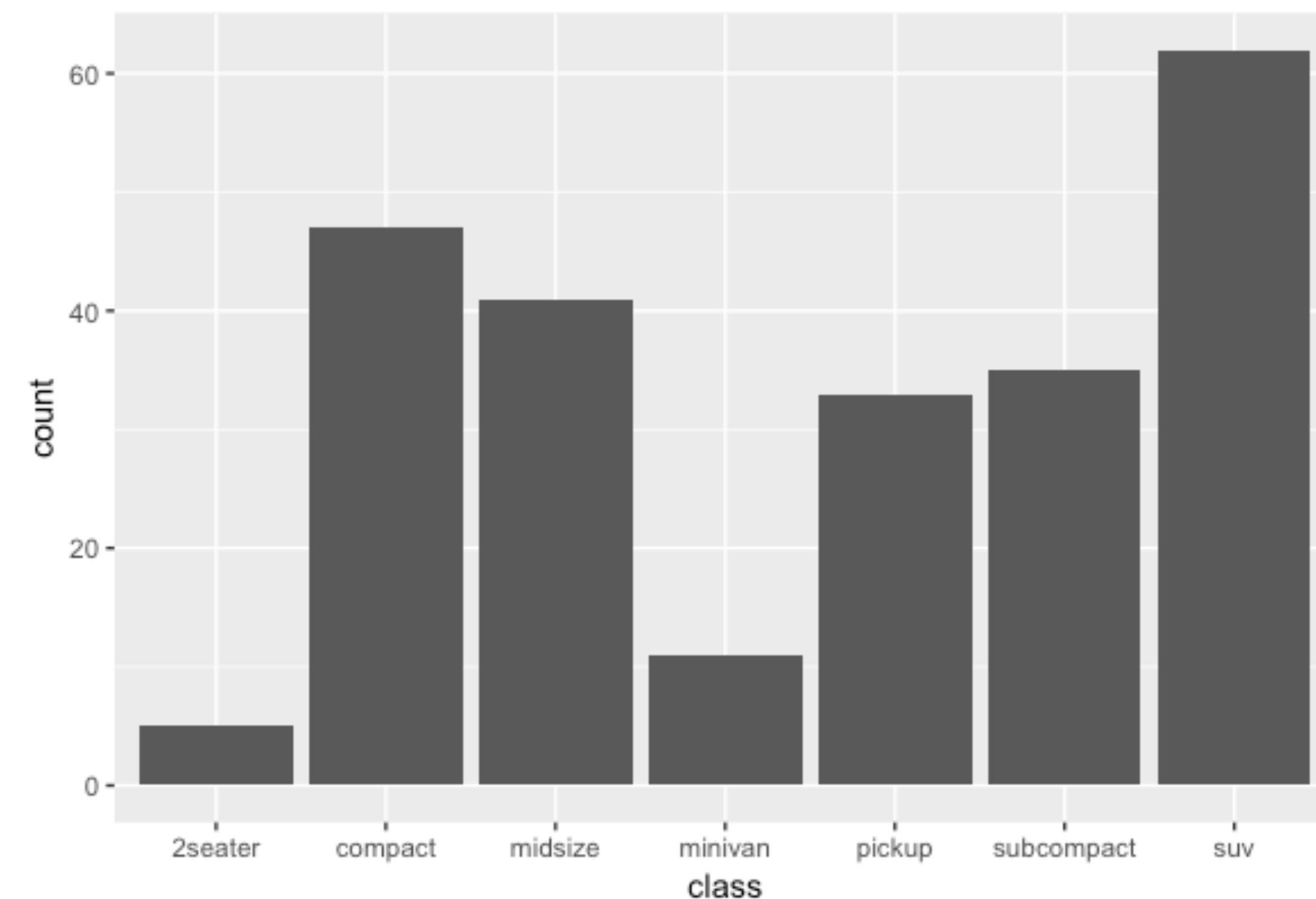
geom\_segment

geom\_smooth

geom\_violin

# categorical univariate

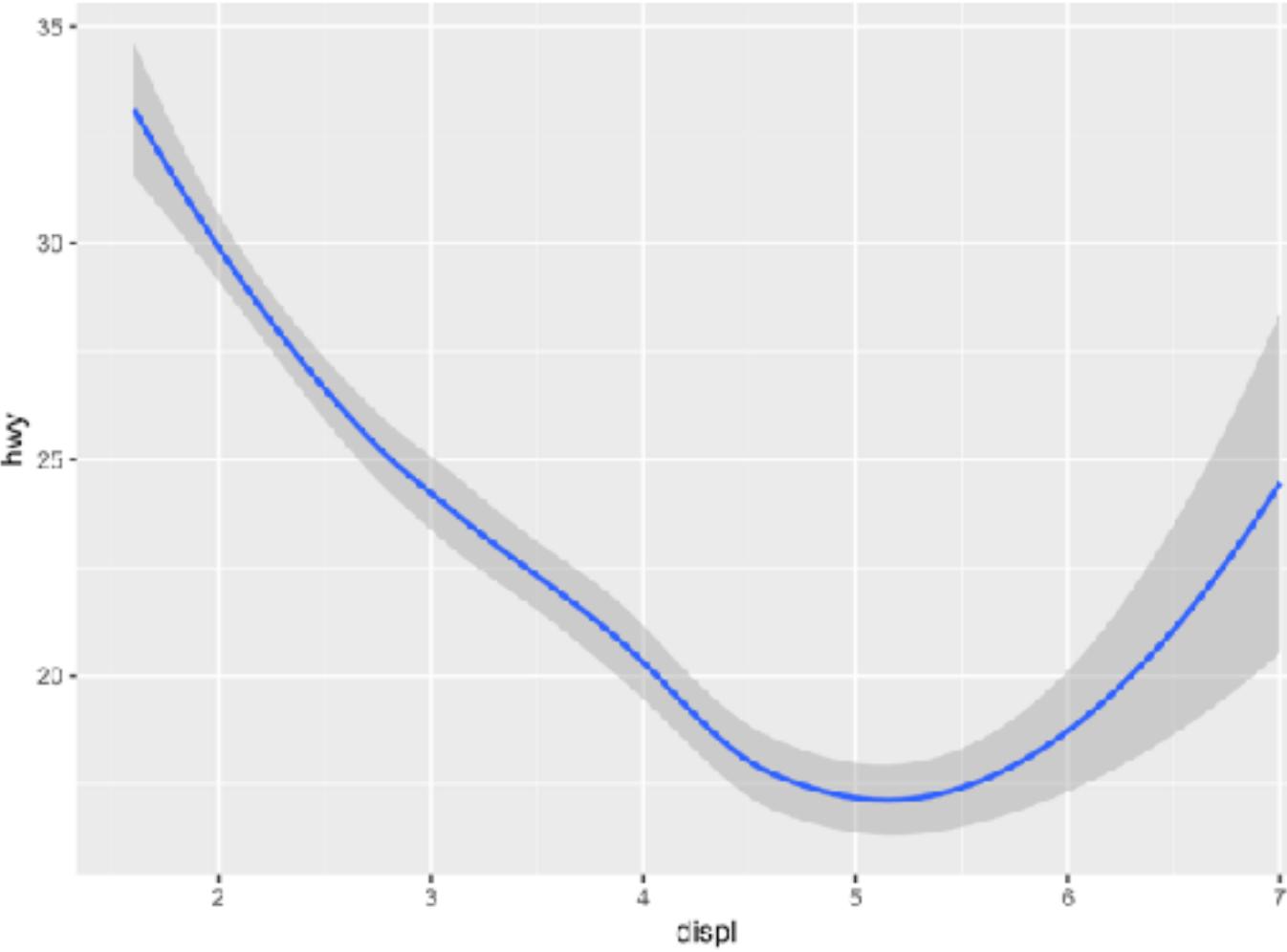
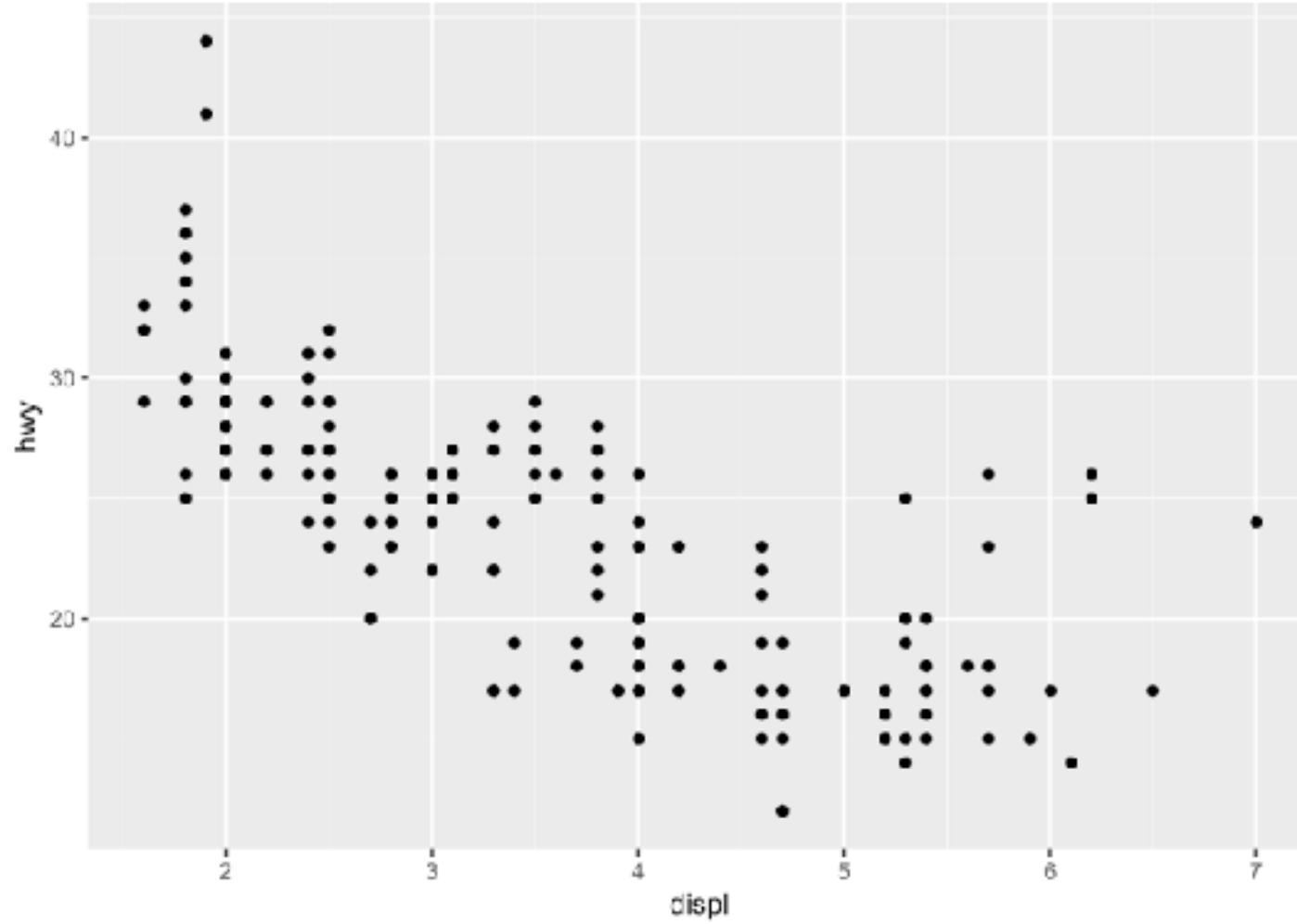
```
ggplot(data = mpg, mapping = aes(x = class)) +  
  geom_bar()
```



geom\_abline  
geom\_bar  
geom\_bin2d  
geom\_blank  
geom\_boxplot  
geom\_contour  
geom\_count  
geom\_hex  
geom\_crossbar  
geom\_density  
geom\_density\_2d  
geom\_dotplot  
geom\_errorharh  
geom\_freqpoly  
geom\_histogram  
geom\_jitter  
geom\_label  
geom\_map  
geom\_path  
geom\_point  
geom\_polygon  
geom\_quantile  
geom\_raster  
geom\_ribbon  
geom\_rug  
geom\_segment  
geom\_smooth  
geom\_violin

# continuous bivariate

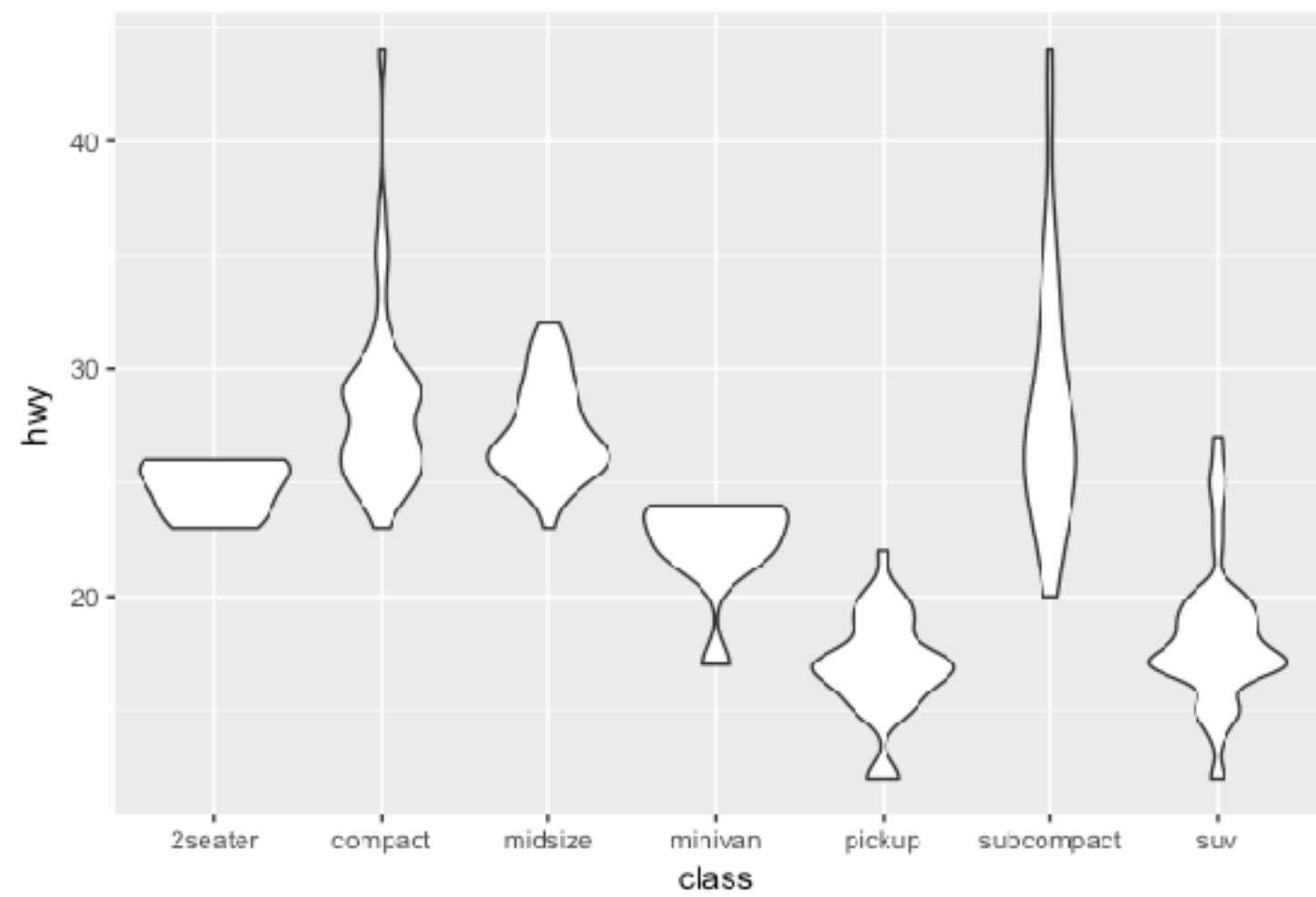
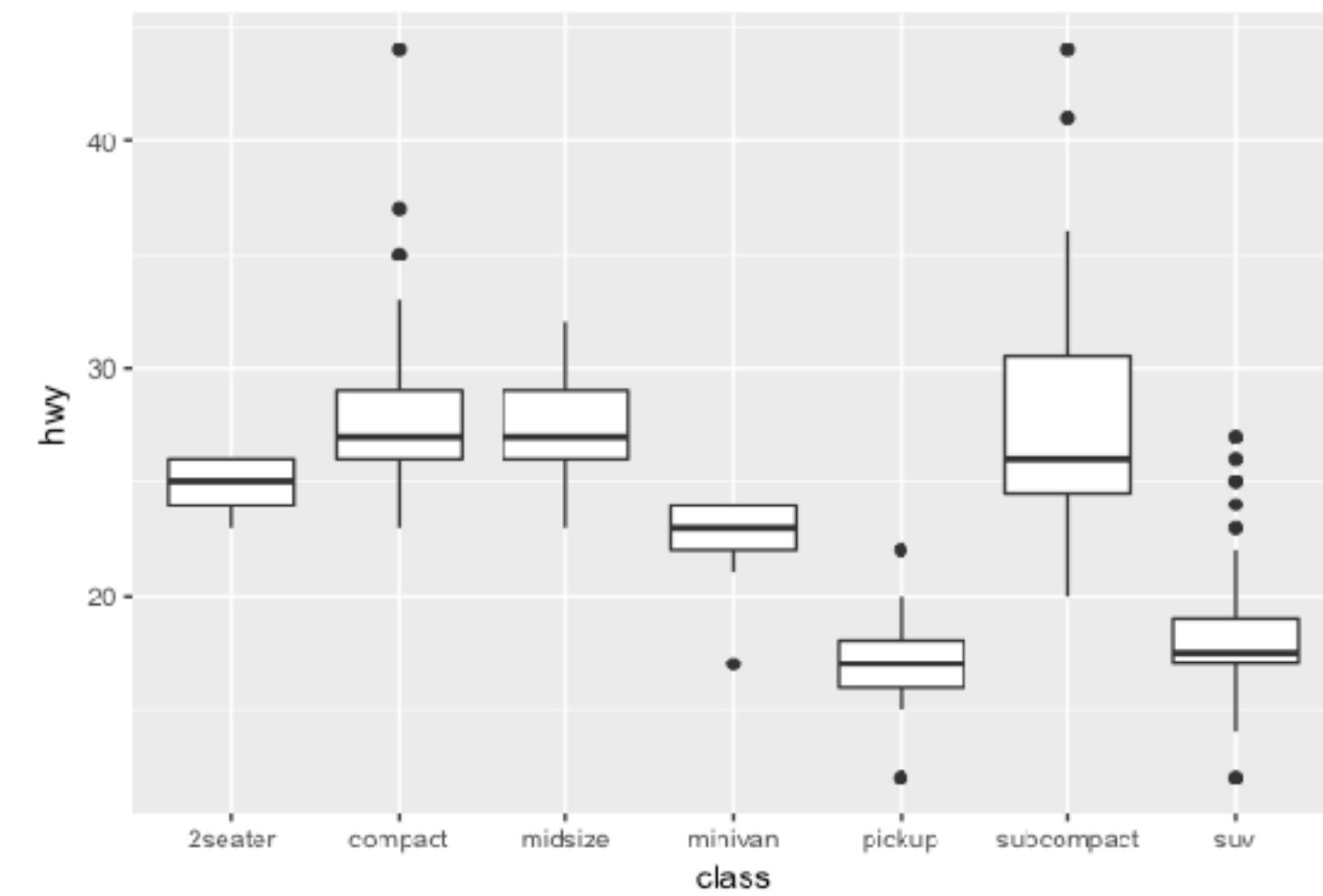
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))  
  
ggplot(data = mpg) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```



geom\_abline  
geom\_bar  
geom\_bin2d  
geom\_blank  
geom\_boxplot  
geom\_contour  
geom\_count  
geom\_hex  
geom\_crossbar  
geom\_density  
geom\_density\_2d  
geom\_dotplot  
geom\_errorbarh  
geom\_freqpoly  
geom\_histogram  
geom\_jitter  
geom\_label  
geom\_map  
geom\_path  
**geom\_point**  
geom\_polygon  
geom\_quantile  
geom\_raster  
geom\_ribbon  
geom\_rug  
geom\_segment  
**geom\_smooth**  
geom\_violin

# bivariate

```
ggplot(data = mpg, aes(x = class, y = hwy)) +  
  geom_boxplot()  
  
ggplot(data = mpg, aes(x = class, y = hwy)) +  
  geom_violin()
```



geom\_abline  
geom\_bar  
geom\_bin2d  
geom\_blank  
geom\_boxplot  
geom\_contour  
geom\_count  
geom\_hex  
geom\_crossbar  
geom\_density  
geom\_density\_2d  
geom\_dotplot  
geom\_errorharh  
geom\_freqpoly  
geom\_smooth  
geom\_violin

# Your turn!

1. Create a chart that illustrates the distribution of **cty** variable
2. Create a chart that shows the number of observations for each manufacturer
3. Create a scatter plot of **cty** vs. **displ**

# Solution

```
# distribution of cty variable  
ggplot(data = mpg, mapping = aes(x = cty)) +  
  geom_histogram()  
  
# number of observations for each manufacturer  
ggplot(data = mpg, mapping = aes(x = manufacturer)) +  
  geom_bar()  
  
# scatter plot for cty vs displ  
ggplot(data = mpg, mapping = aes(x = displ, y = cty)) +  
  geom_point()
```

# layering multiple geoms

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point() +  
  geom_smooth()
```

- The 2nd code also produces the same graph. Why is it a better code than its above?

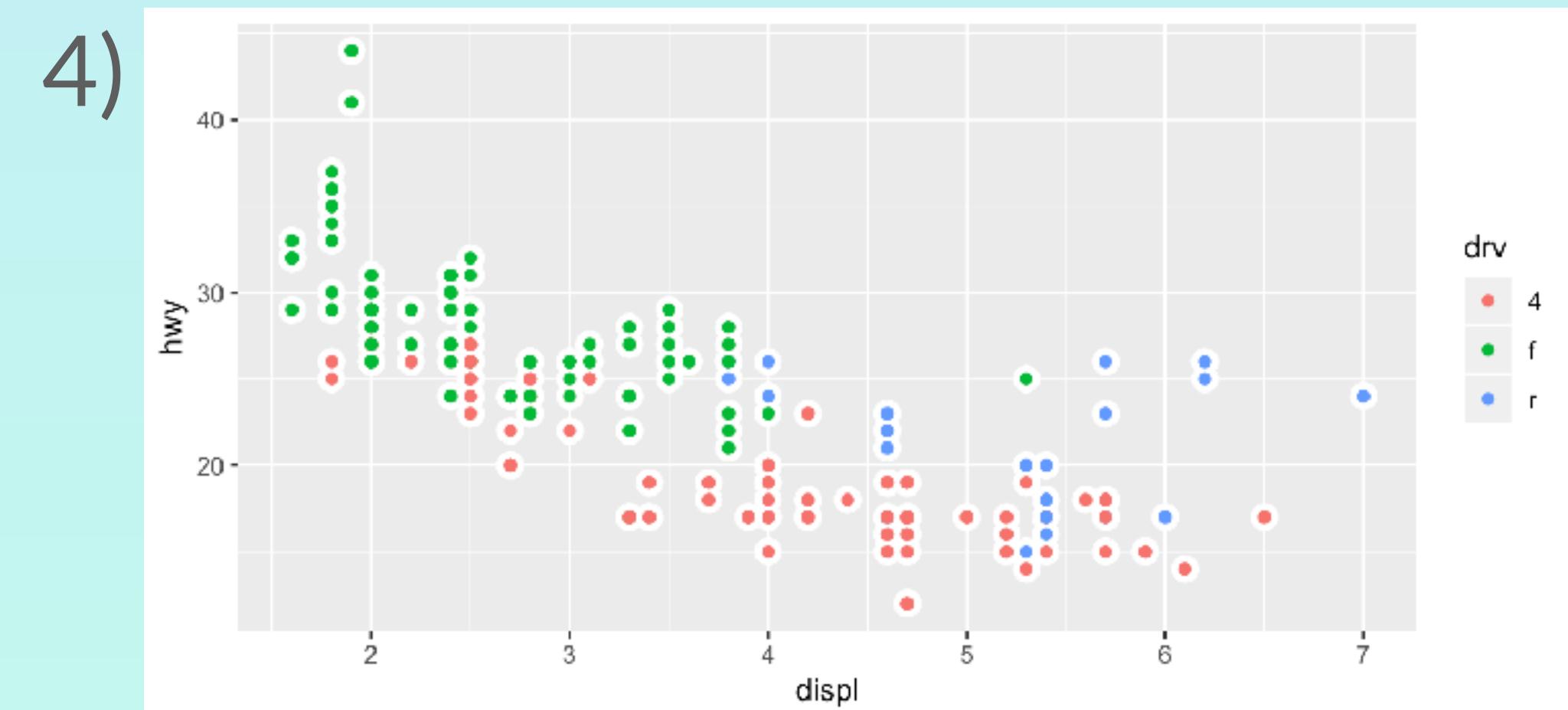
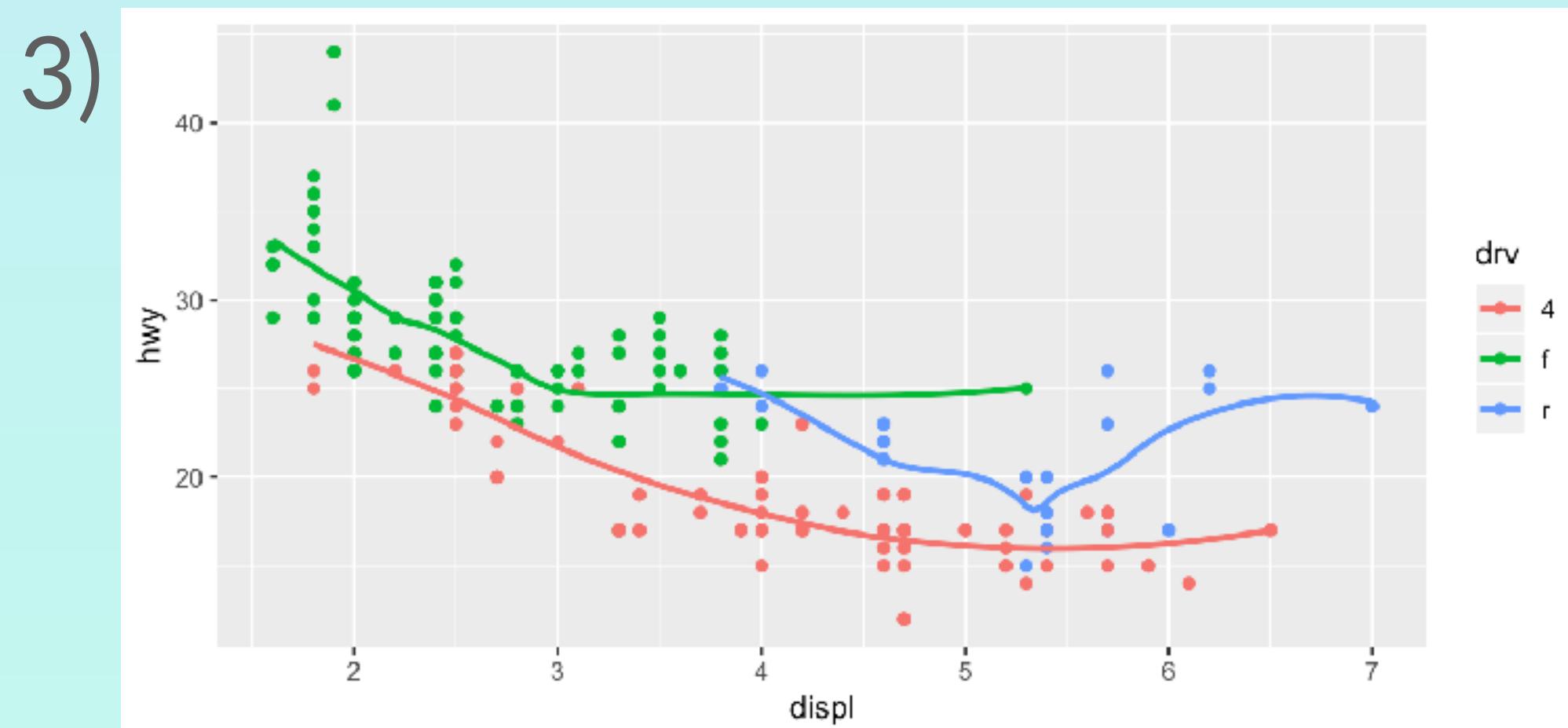
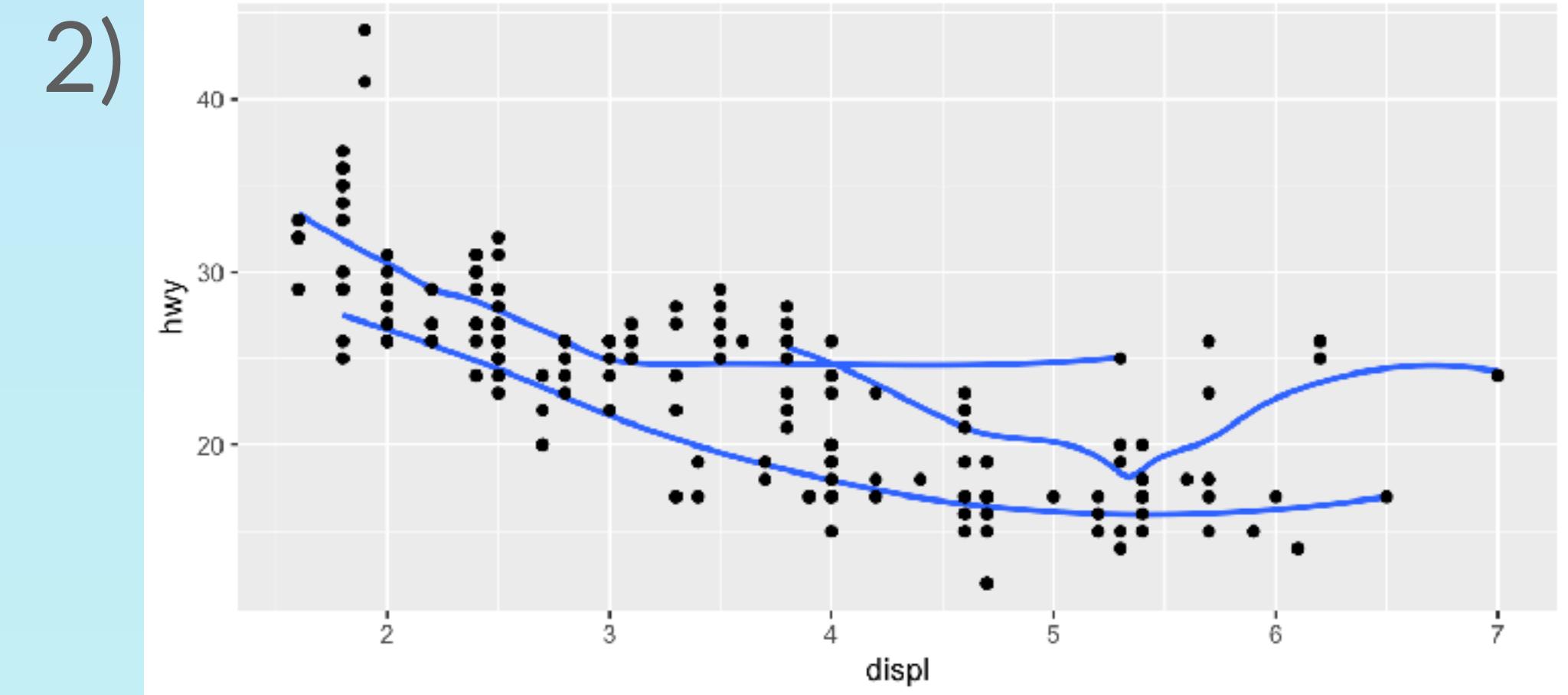
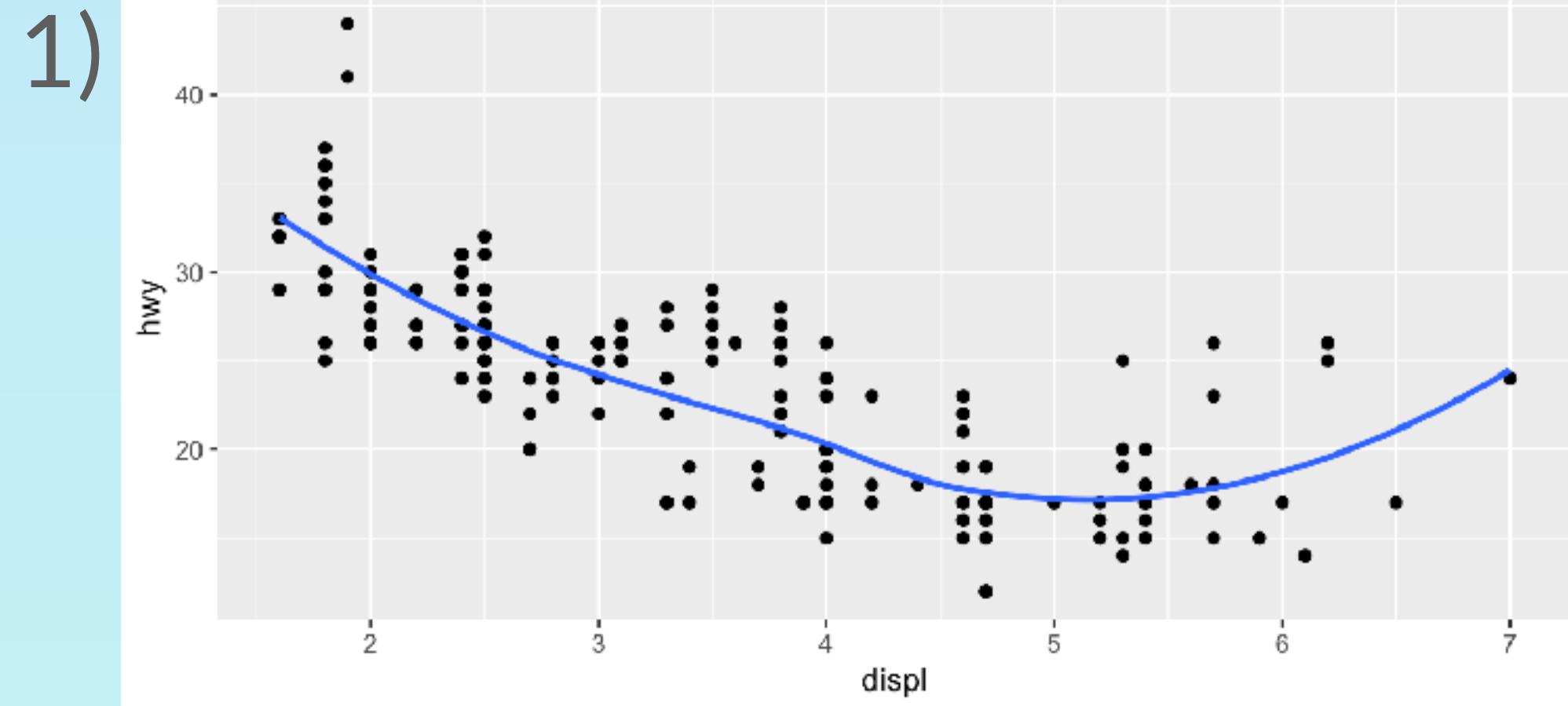
# Your turn!

1. Think of what the output will look like, then run the following code to check your prediction.
  1. What does **se** argument do?
  2. What does **show.legend** argument do?

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +  
  geom_point() +  
  geom_smooth(se = FALSE, show.legend = FALSE)
```

# Your turn!

See if you can reproduce the following plots:

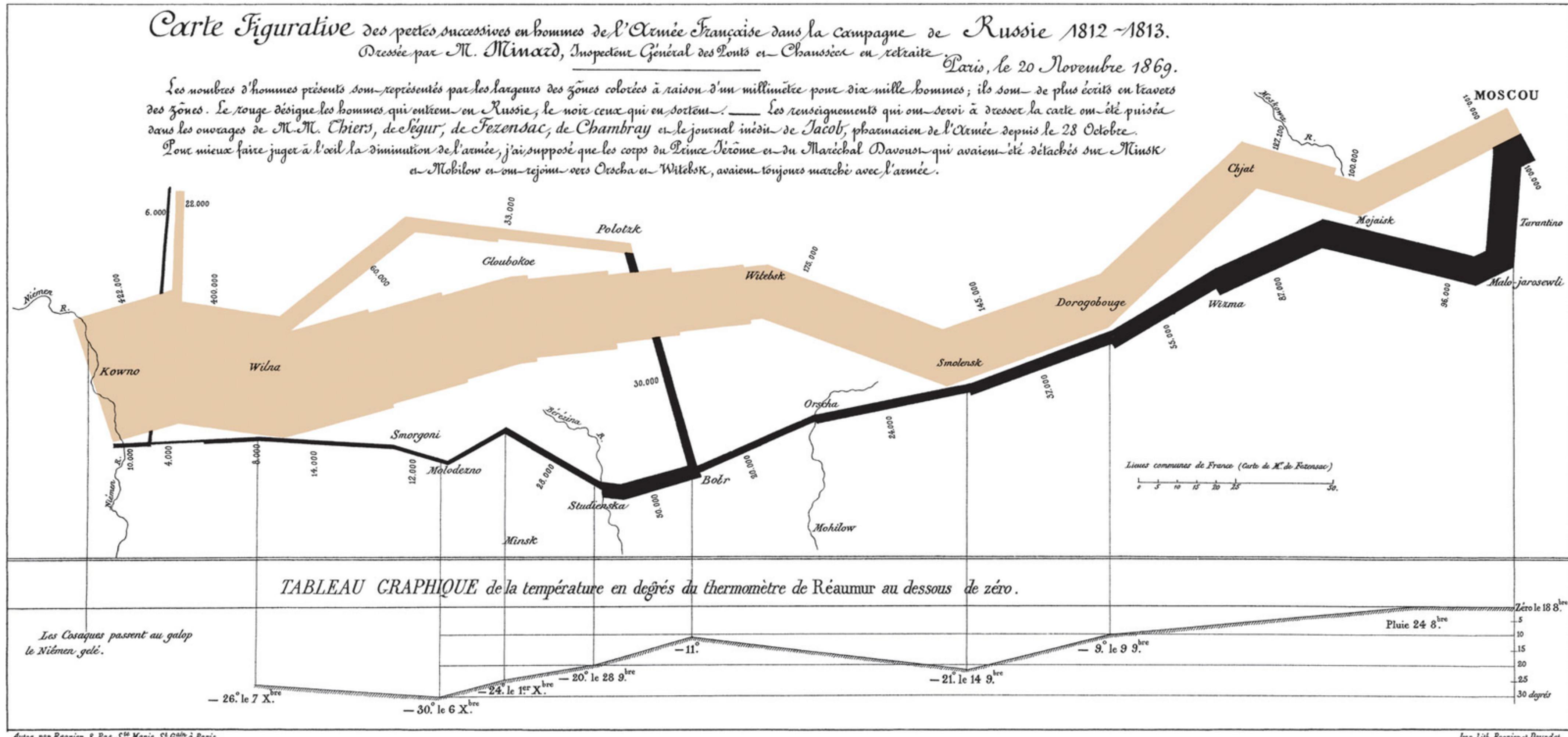


# Solution

- 1) `ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
 geom_point() +  
 geom_smooth(se = FALSE)`
- 2) `ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
 geom_smooth(aes(group = drv), se = FALSE) +  
 geom_point()`
- 3) `ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +  
 geom_point() +  
 geom_smooth(se = FALSE)`
- 4) `ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
 geom_point(size = 4, colour = "white") +  
 geom_point(aes(colour = drv))`

# A layered grammar of graphics

# Minard's map



# Datasets

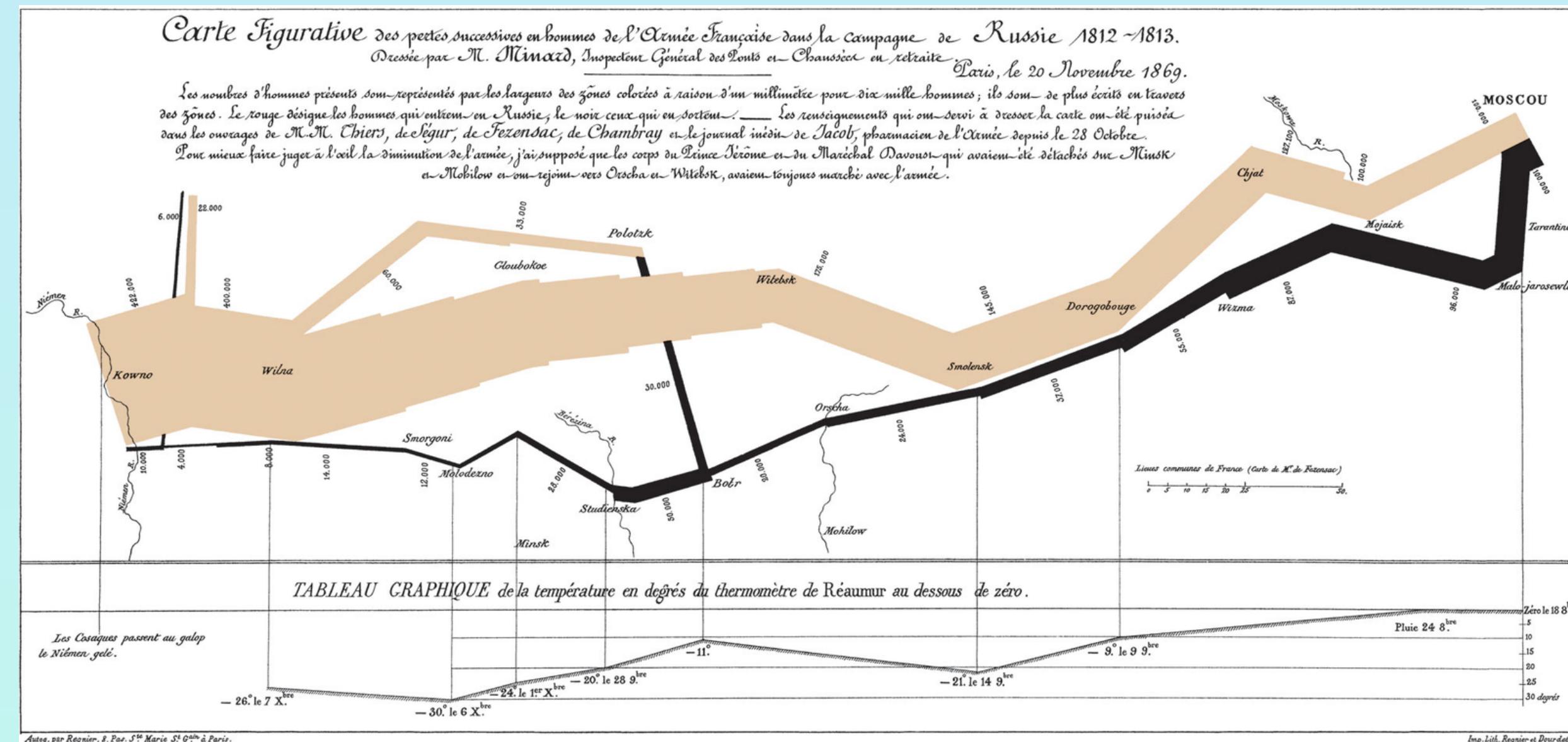
- Download the datasets: <http://bit.ly/minardDataset>
- Load the data. More details of importing data in another lecture...

```
# tidyverse package
library(tidyverse)
# read table with function from readr
troops <- read_table2("data/minard-troops.txt")

cities <- read_table2("data/minard-cities.txt")
```

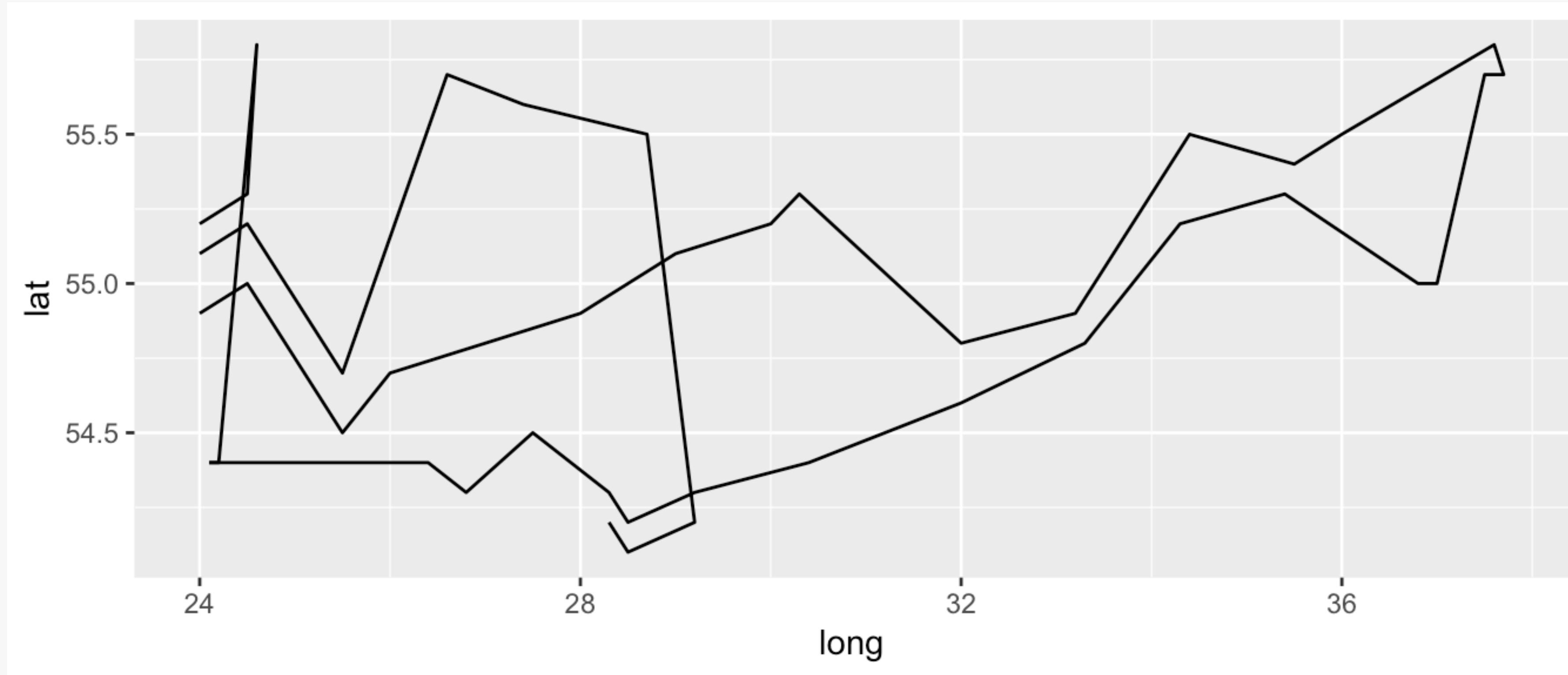
# Your turn!

1. Examine the **troops** and **cities** data tables and consider how you would approach reproducing the Minard's map.



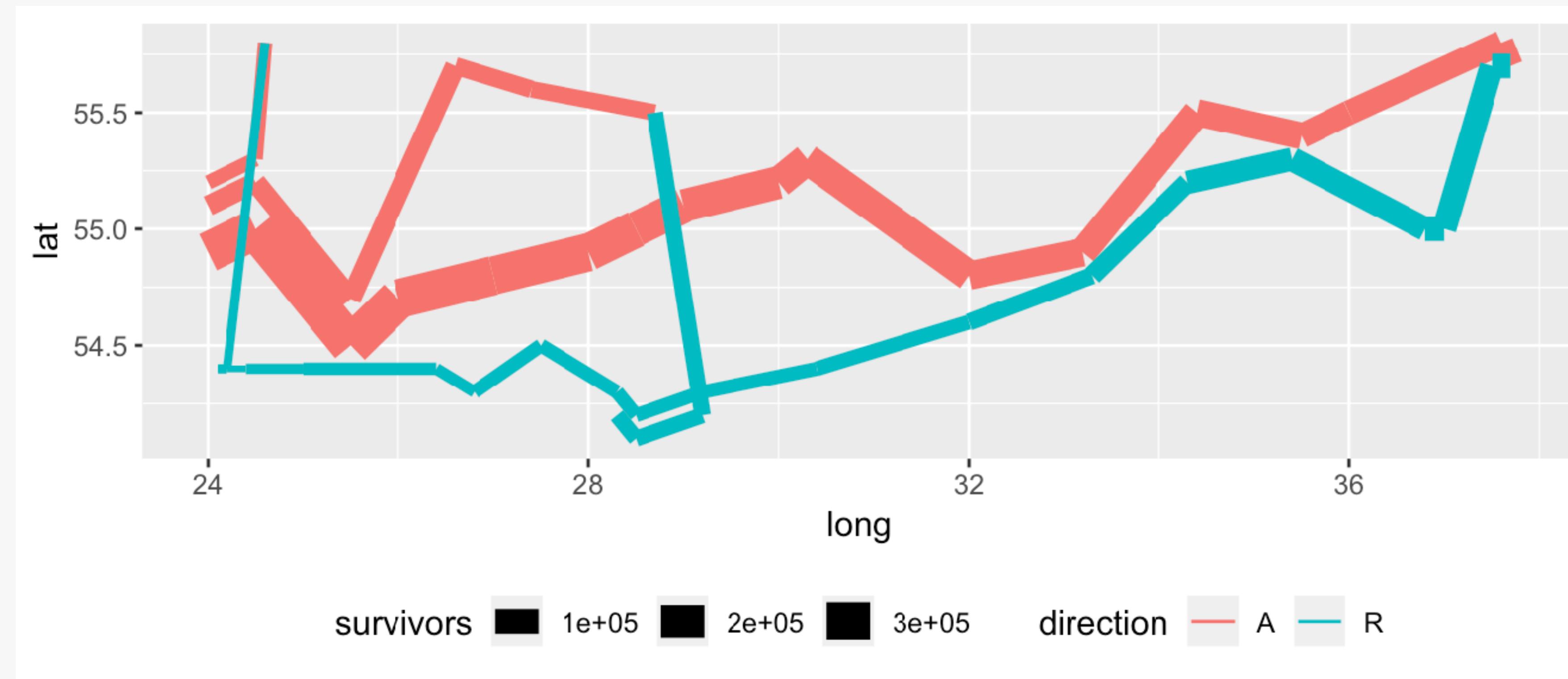
# Solution

```
# drawing the marching paths  
ggplot(data = troops) +  
  geom_path(mapping = aes(x = long, y = lat, group = group))
```



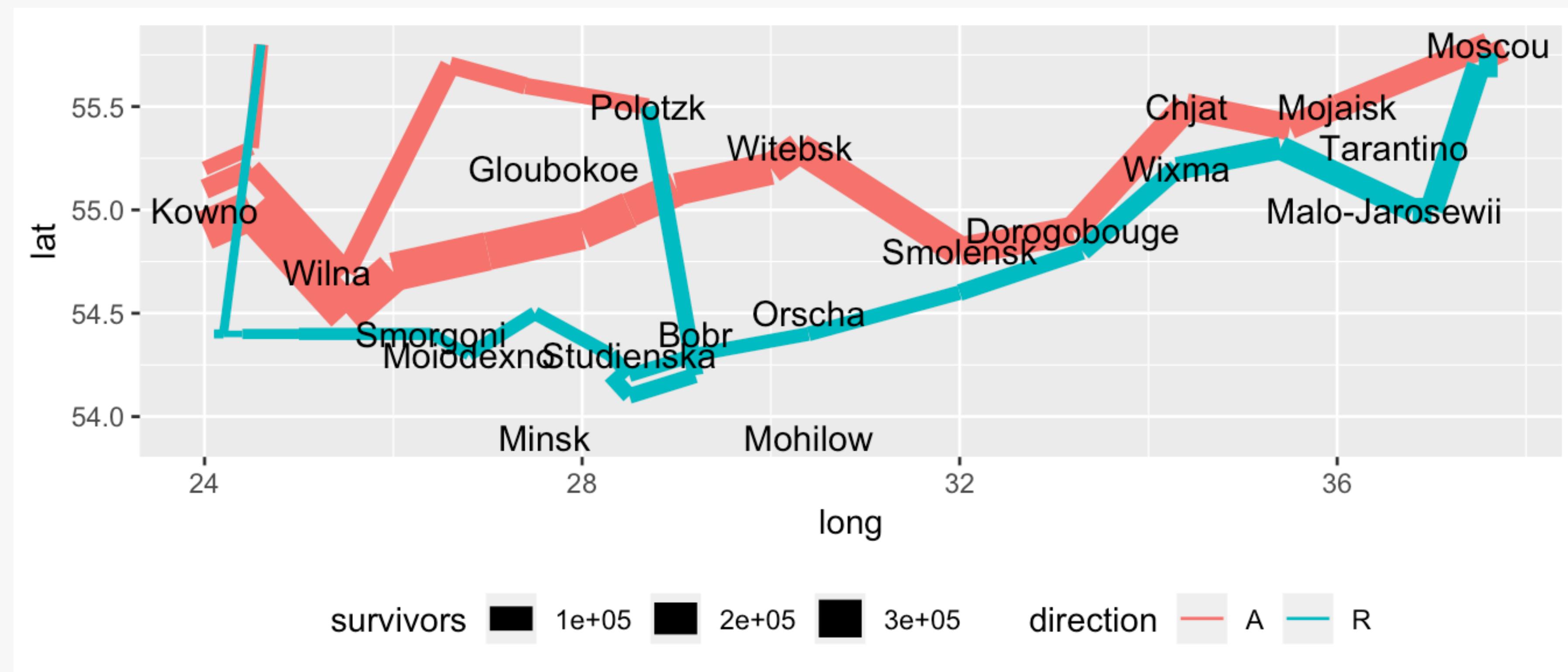
# Solution

```
# drawing the marching paths
ggplot(data = troops) +
  geom_path(mapping = aes(x = long, y = lat, size = survivors,
colour = direction, group = group)) +
  theme(legend.position="bottom")
```



# Solution

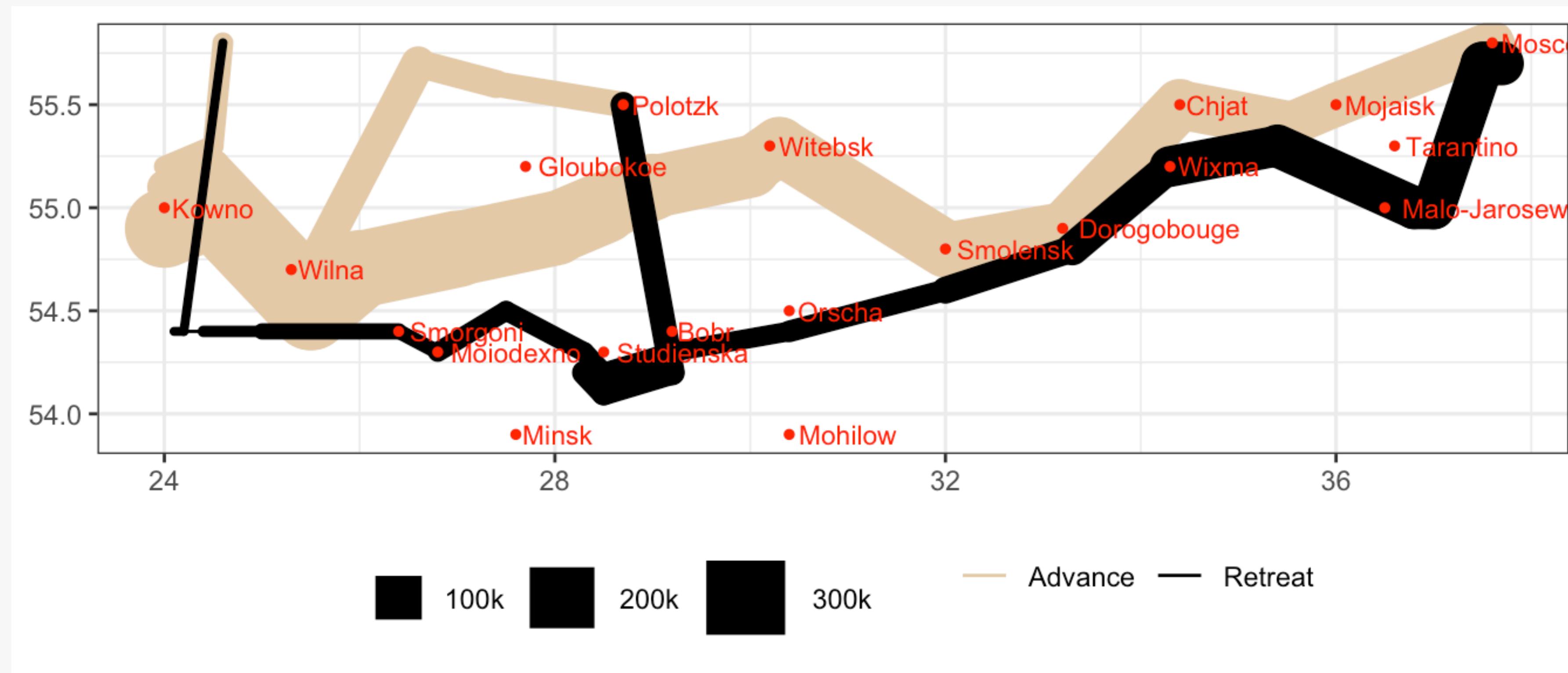
```
# Adding text Labels
ggplot() +
  geom_path(data = troops, mapping = aes(x = long, y = lat, size = survivors, color = direction, group = group)) +
  geom_text(data = cities, mapping = aes(x = long, y = lat, label = city), size = 4) +
  theme(legend.position="bottom")
```



# Solution

```
ggplot() +  
  geom_path(data = troops, mapping = aes(x = long, y = lat, size = survivors, color = direction, group = group),  
lineend = "round", linejoin = "mitre") +  
  scale_size(range = c(0.5, 12), limits = c(4000, 350000), trans = "identity", breaks = c(100000, 200000, 300000),  
labels = c("100k", "200k", "300k"))+  
  scale_color_manual(values = c("#E5CBAA","black"), labels = c("Advance", "Retreat")) +  
  xlab(NULL) +  
  ylab(NULL) +  
  geom_point(data = cities, mapping = aes(x = long, y = lat, label = city), size = 1, color = "red") +  
  geom_text(data = cities, mapping = aes(x = long, y = lat, label = city), size = 3, color = "red", hjust = -0.1) +  
  theme_bw() +  
  theme(legend.position="bottom") +  
  guides(size = guide_legend(title = NULL), color = guide_legend(title = NULL))
```

# Solution



Further reading link

**10 minutes break**

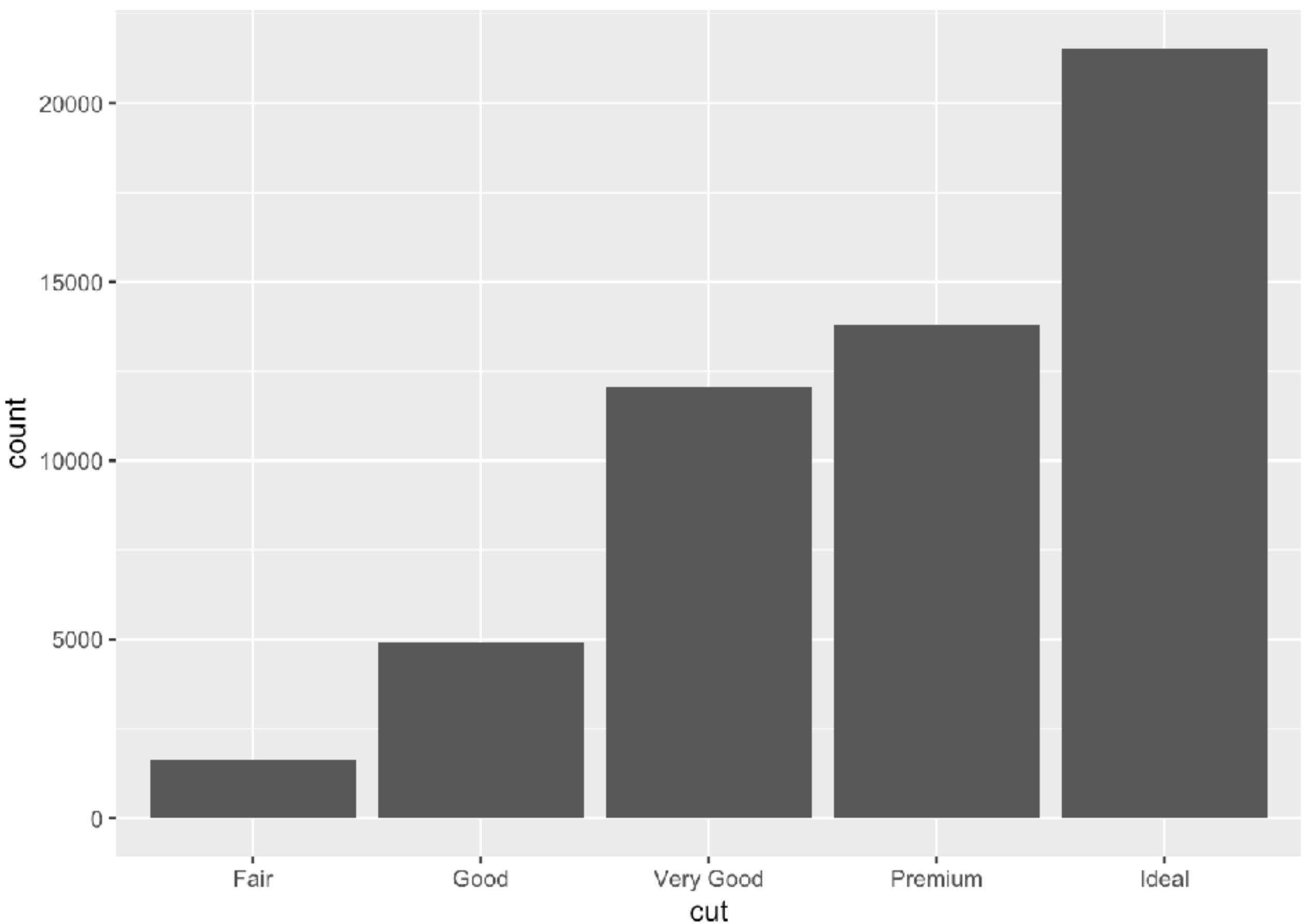
# Statistical transformation

behind the scene

# Statistical transformation

- Inner working of a seemingly simple bar chart
- diamonds datasets

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut))
```



# Computed variables

- look up in documentation: `?geom_bar()`

`geom_bar {ggplot2}`

R Documentation

## Bar charts

### Description

There are two types of bar charts: `geom_bar()` and `geom_col()`. `geom_bar()` makes the height of the bar proportional to the number of cases in each group (or if the `weight` aesthetic is supplied, the sum of the weights). If you want the heights of the bars to represent values in the data, use `geom_col()` instead. `geom_bar()` uses `stat_count()` by default: it counts the number of cases at each `x` position. `geom_col()` uses `stat_identity()`: it leaves the data as is.

### Usage

```
geom_bar(mapping = NULL, data = NULL, stat = "count",
  position = "stack", ..., width = NULL, binwidth = NULL,
  na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)

geom_col(mapping = NULL, data = NULL, position = "stack", ...,
  width = NULL, na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE)

stat_count(mapping = NULL, data = NULL, geom = "bar",
  position = "stack", ..., width = NULL, na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE)
```

### Arguments

# Computed variables

- look up in documentation: `?geom_bar()`

- `group`
- `linetype`
- `size`

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

## Computed variables

`count`

number of points in bin

`prop`

groupwise proportion

## See Also

[geom\\_histogram\(\)](#) for continuous data, [position\\_dodge\(\)](#) and [position\\_dodge2\(\)](#) for creating side-by-side bar charts.

[stat\\_bin\(\)](#), which bins data in ranges and counts the cases in each range. It differs from `stat_count`, which counts the number of cases at each `x` position (without binning into ranges). [stat\\_bin\(\)](#) requires continuous `x` data, whereas `stat_count` can be used for both discrete and continuous `x` data.

## Examples

```
# geom_bar is designed to make it easy to create bar charts that show
```

# Overriding case 1

- WYSIWYG (what you see is what you get)

- You may want to pre-calculate and supply the value exactly to create a bar chart
- user **stat = “identity”** to define values for **x** and **y**.

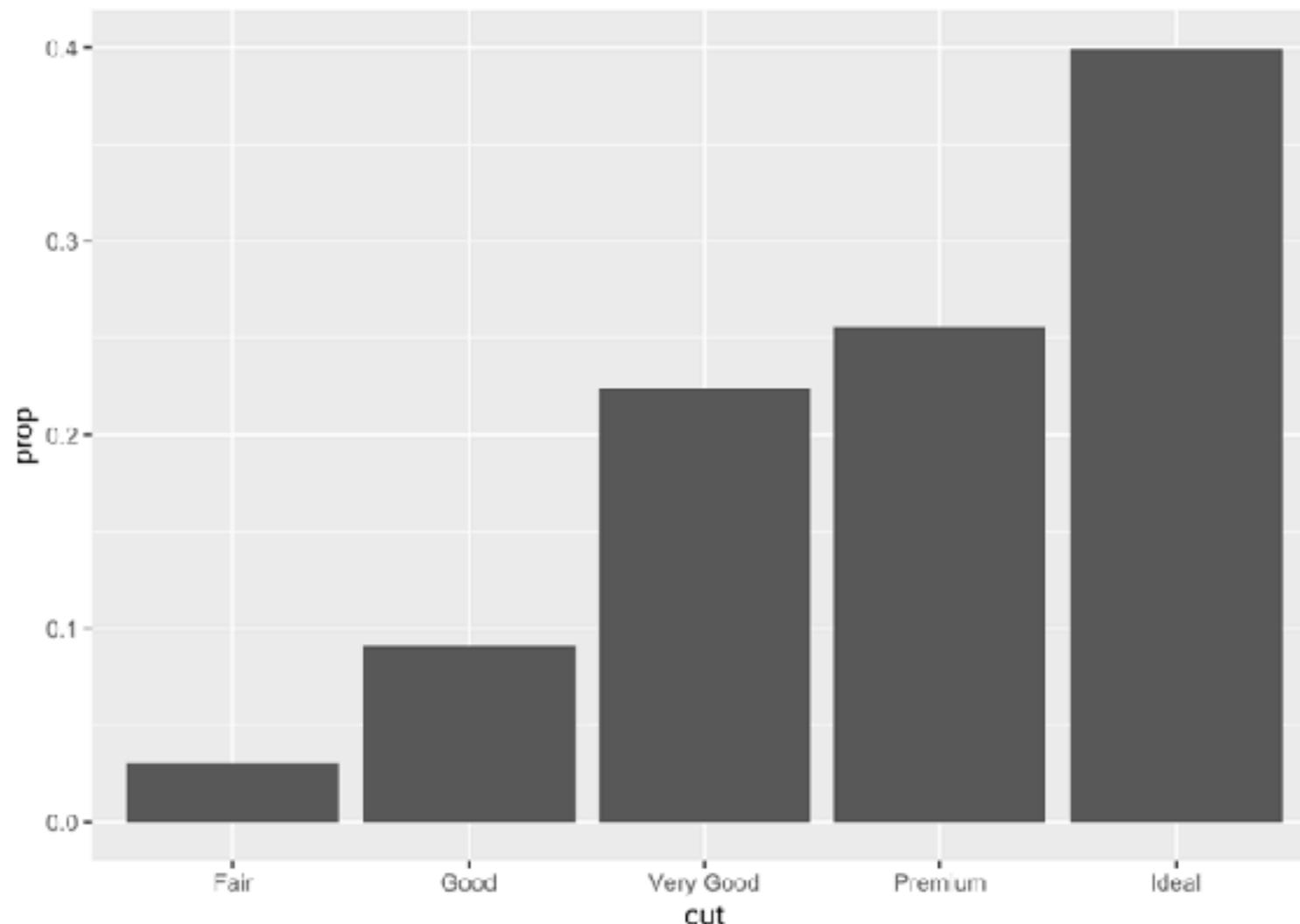
```
# pre-calculate
cut <- c("Fair", "Good", "Very Good", "Premium", "Ideal")
freq <- c(1610, 4906, 12082, 13791, 21551)
demo <- data_frame(cut, freq)

ggplot(data = demo) +
  geom_bar(mapping = aes(x = cut, y = freq), stat = "identity")
```

# Overriding case 2

- Using computed variables
  - `geom_bar()` : `prop` returns groupwise proportion
  - access a computed variable by surrounding with two periods, e.g. `..prop..`

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, y = ..prop.., group = "demo"))
```

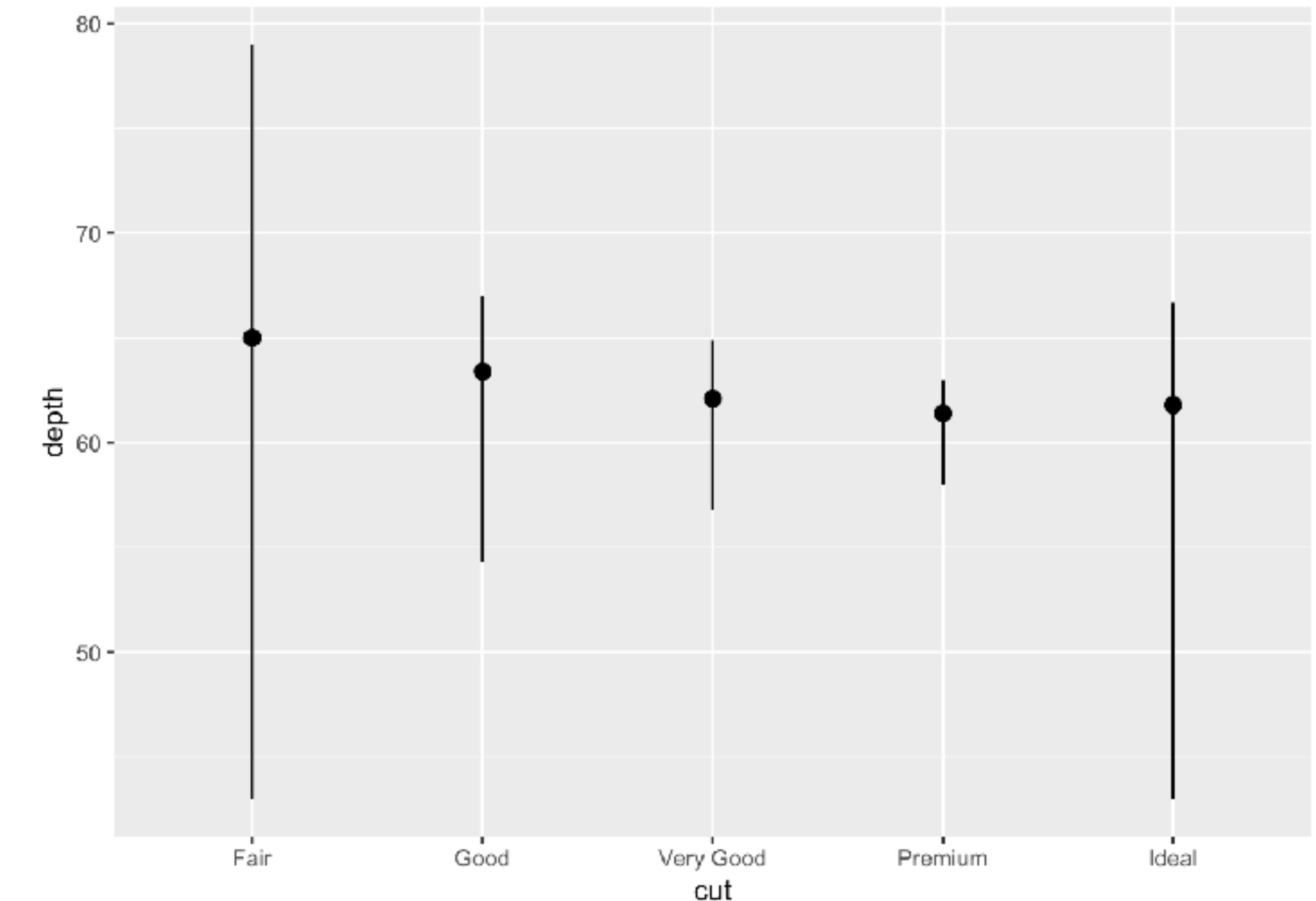


# Overriding case 3

- Using non-default statistical transformation

- see the cheatsheet for other statistical transformations
- example **stat\_summary()**
- **cut**: a categorical variable
- **depth**: a quantitative variable for the total depth percentage

```
ggplot(data = diamonds) +  
  stat_summary(mapping = aes(x = cut, y = depth),  
  fun.ymin = min, fun.ymax = max, fun.y = median)
```



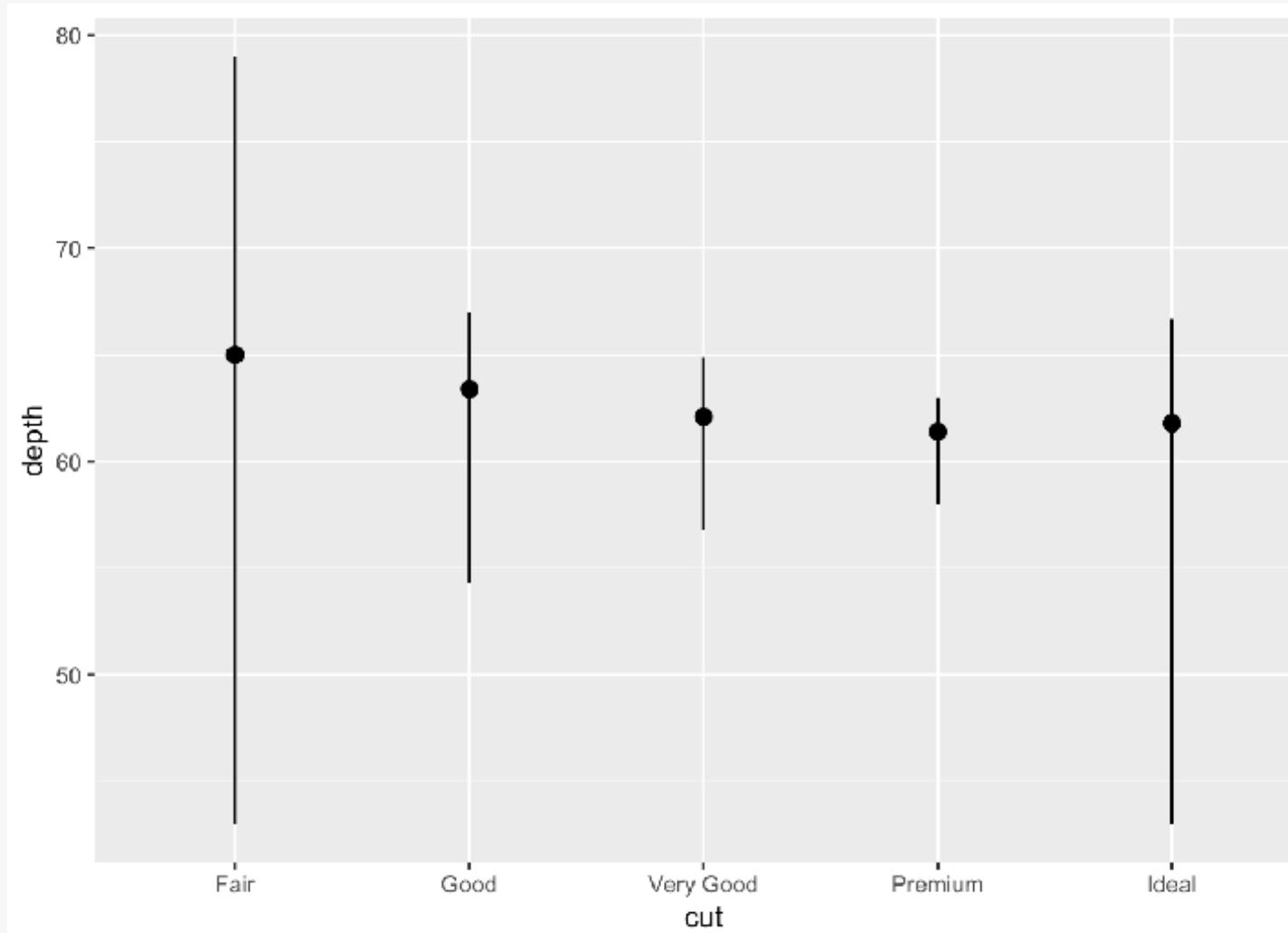
# Your turn!

1. What is the default geom associated with `stat_summary()`? How can you rewrite the previous plot to use that geom function instead of the stat function? (Answer: `geom_pointrange()`)
2. What does `geom_col()` do? How is it different to `geom_bar()`? (Hint: Read the documentation)
3. What variables does `stat_smooth()` compute? What are parameters involved in controlling its behaviour?

```
stat_summary(mapping = NULL, data = NULL, geom = "pointrange",
             position = "identity", ..., fun.data = NULL, fun.y = NULL,
             fun.ymax = NULL, fun.ymin = NULL, fun.args = list(),
             na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

# Solution

- **stat\_summary()**:
  - default **geom** is **geom\_pointrange()**
- **geom\_pointrange()**:
  - default **stat** is **stat\_identity()**



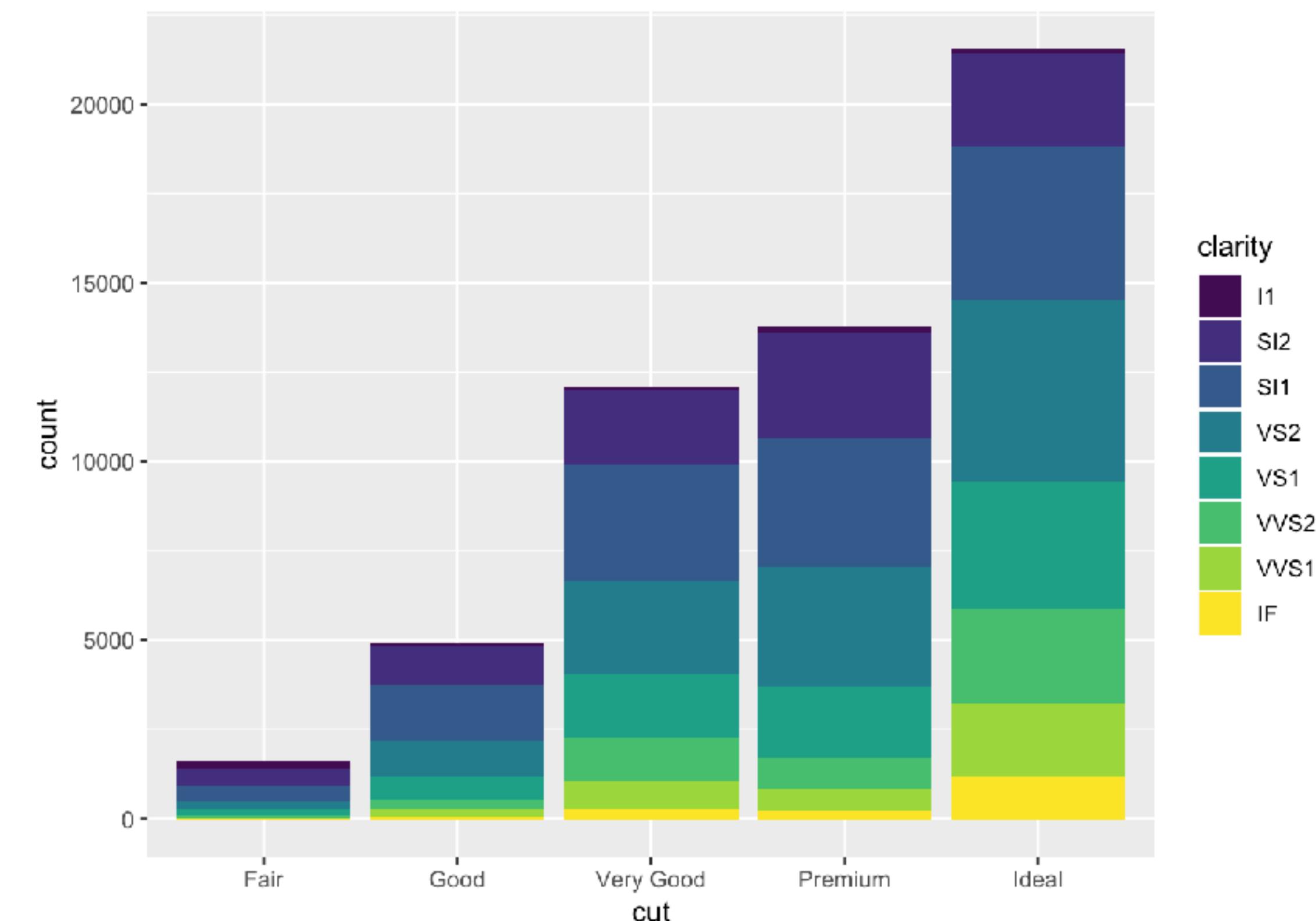
```
ggplot(data = diamonds) +  
  stat_summary( mapping = aes(x = cut, y = depth),  
    fun.ymin = min,  
    fun.ymax = max,  
    fun.y = median)  
  
ggplot(data = diamonds) +  
  geom_pointrange( mapping = aes(x = cut, y = depth),  
    stat = "summary",  
    fun.ymin = min,  
    fun.ymax = max,  
    fun.y = median)
```

# Position adjustments

# Position adjustments

- The **position** argument specifies how the graphs are drawn, while **stat** defines the statistical transformation.

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



# geom\_bar

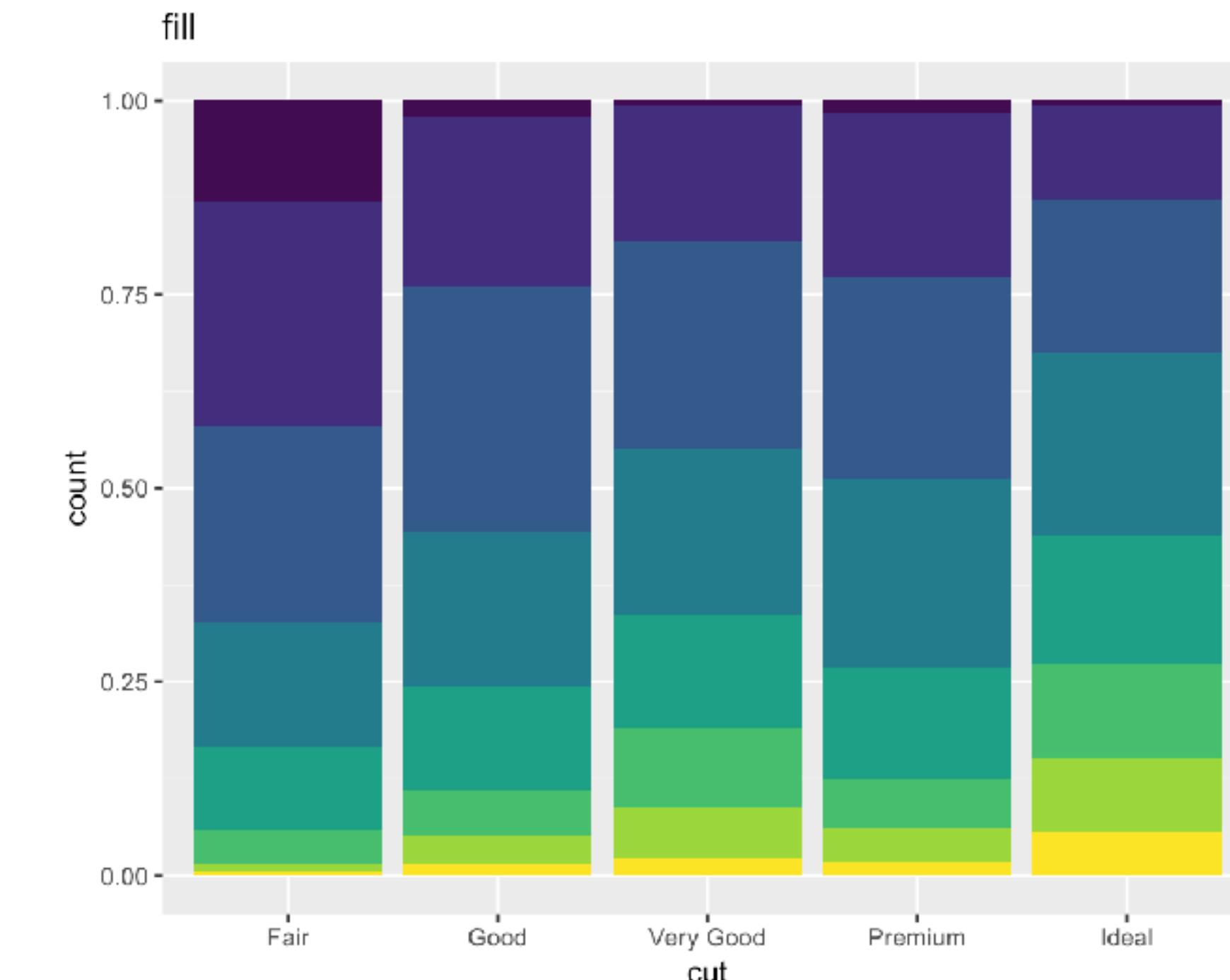
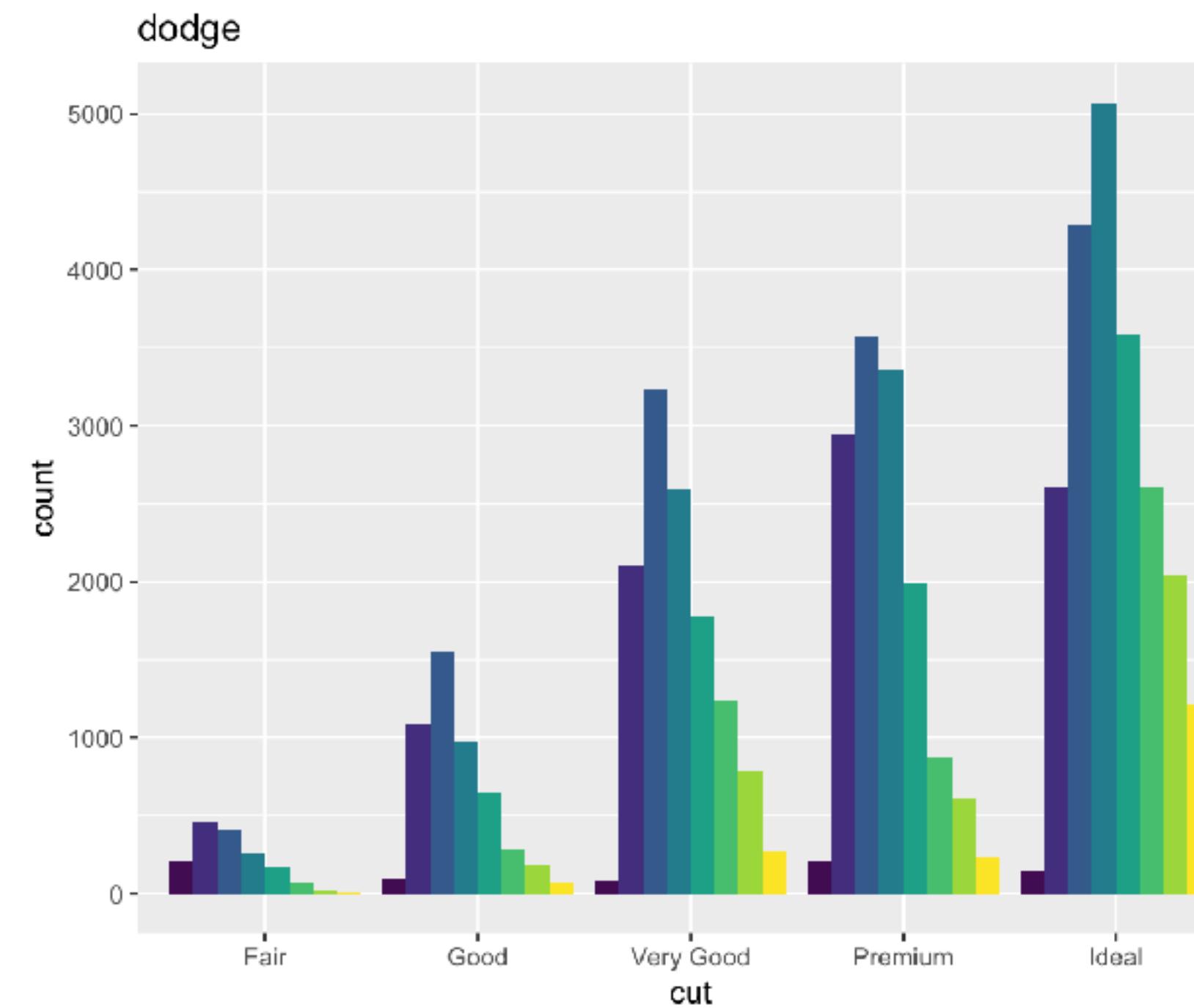
- 4 options:

1. **stack** : default option to create a stacked barcharts
2. **identity** : un-stacked, draws each object exactly where it falls in the context of the graph. This option is not very helpful as bars overlap
3. **dodge** : avoids overlapping bars by placing beside one another
4. **fill** : works like stack but visualise the proportions across groups

# geom\_bar

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
```

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



clarity

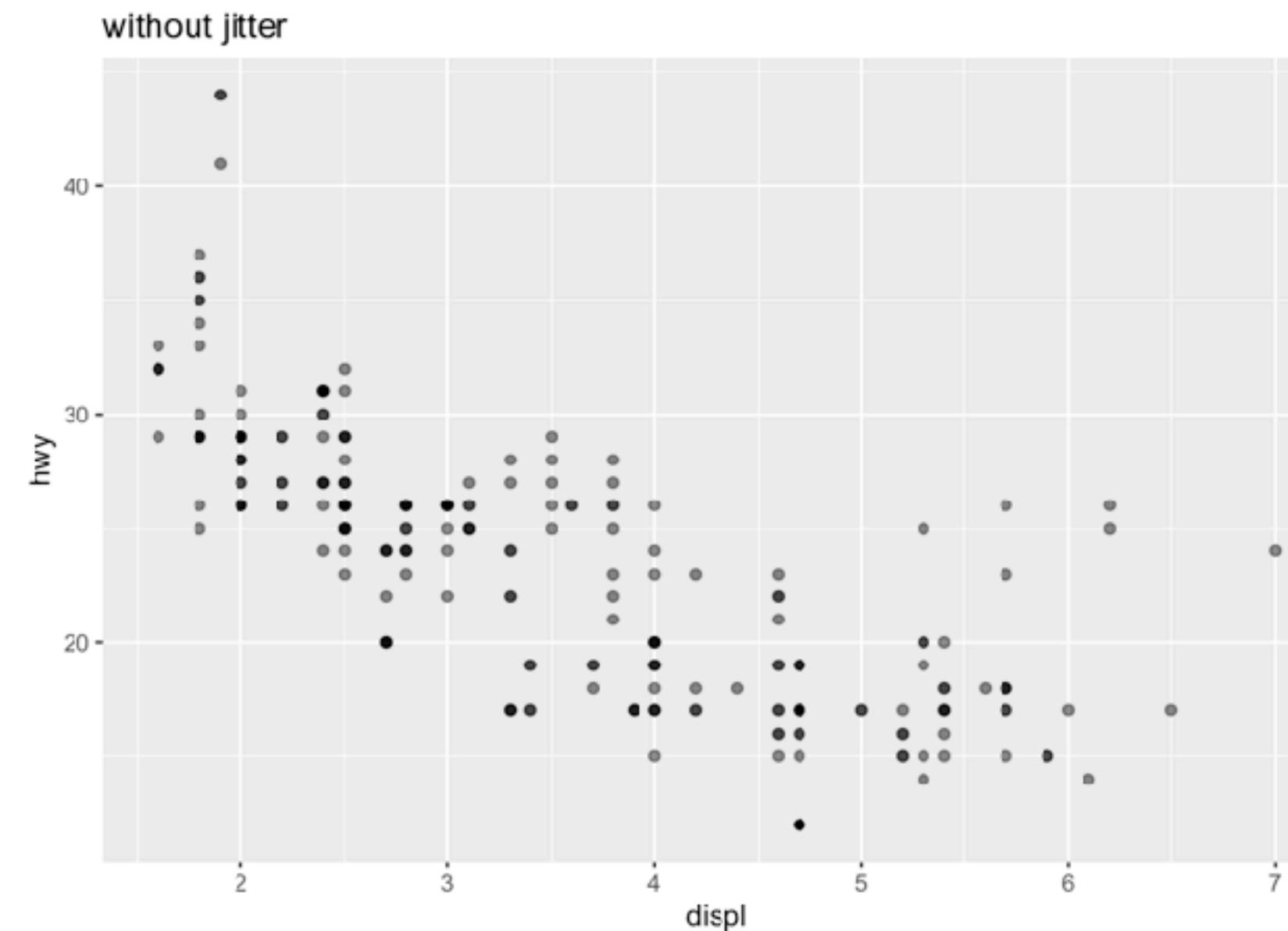
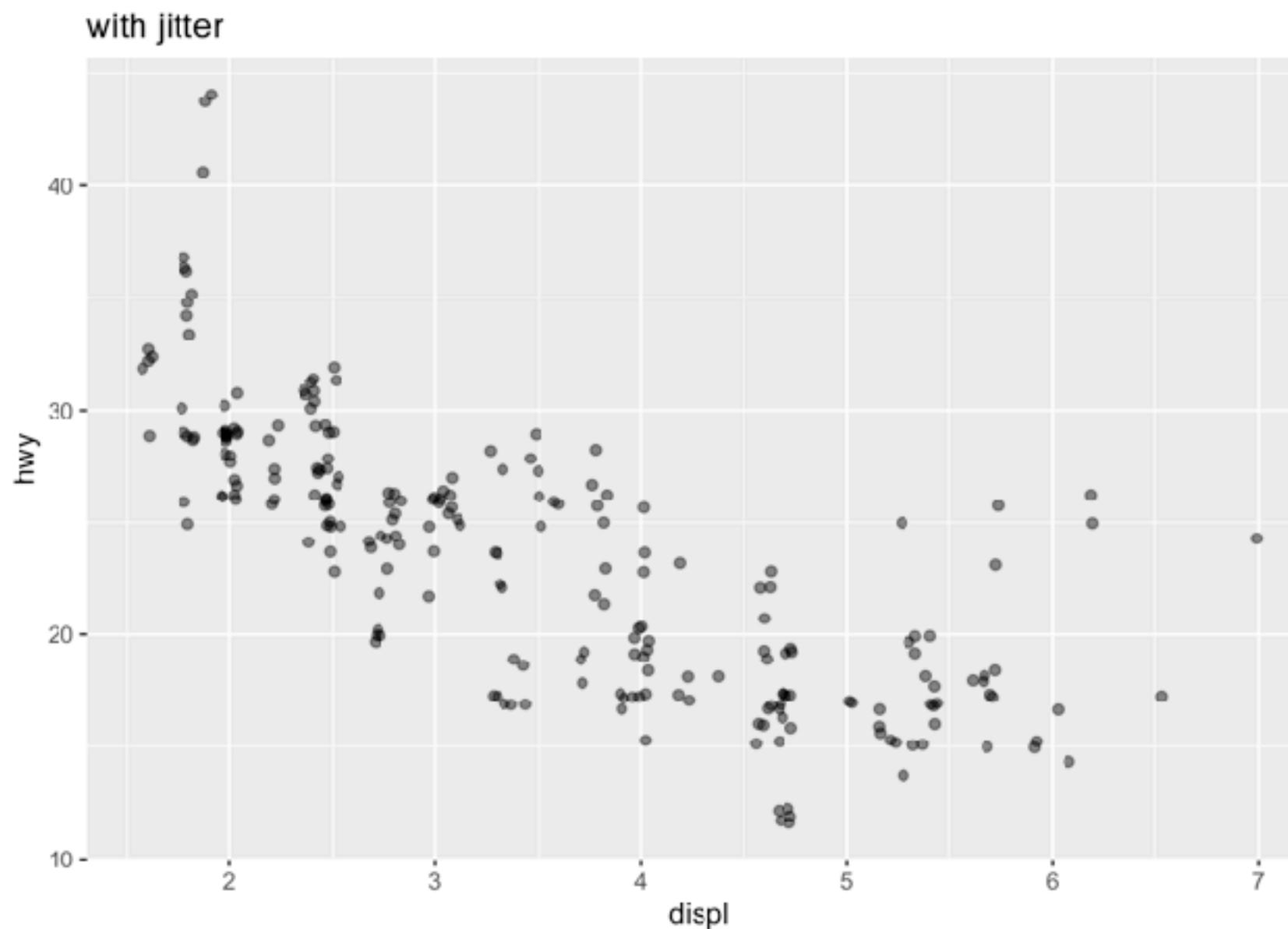
- I1
- SI2
- SI1
- VS2
- VS1
- VVS2
- VVS1
- IF

# geom\_jitter()

- avoid overplotting, e.g. scatter plot
  - adds small amount of random noise to each point to spread the overlapping points out

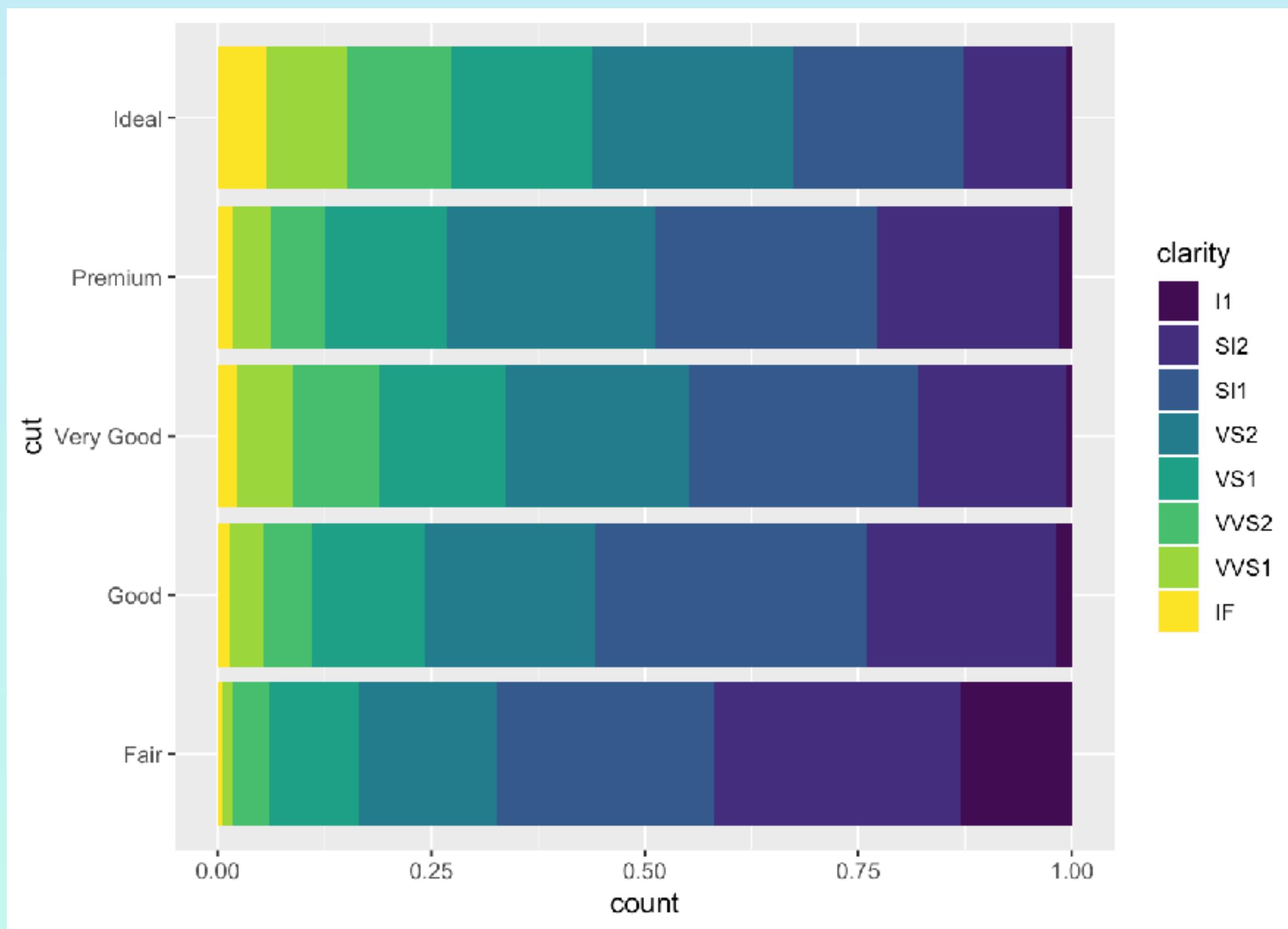
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy), position = "jitter", alpha = 0.5)
```

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy), alpha = 0.5)
```



# Your turn!

1. What parameters to `geom_jitter()` control the amount of displacement?
2. Try reproducing the following plot. (Hint: look up `coord_flip()`)

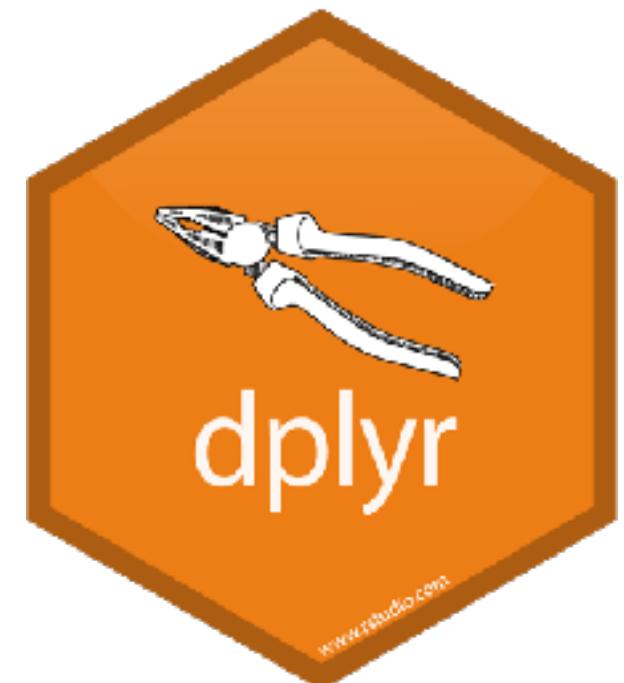


# Data Transformation

## dplyr package

“Up to 80% of data analysis is spent on the process of cleaning and preparing data.”  
– cf. Wickham, 2014 and Dasu & Johnson, 2003

# dplyr



- You are going to learn the 7 key dplyr functions that allow you to solve the vast majority of your data manipulation challenges:

| Function                 | Description                    | Equivalent SQL |
|--------------------------|--------------------------------|----------------|
| <code>select()</code>    | selecting columns              | SELECT         |
| <code>filter()</code>    | filtering rows / subsetting    | WHERE          |
| <code>group_by()</code>  | grouping data                  | GROUP BY       |
| <code>summarise()</code> | summarising / aggregating data | -              |
| <code>arrange()</code>   | sorting data                   | ORDER BY       |
| <code>join()</code>      | joining data tables            | JOIN           |
| <code>mutate()</code>    | creating new columns           | COLUMN ALIAS   |

# Basic principle

- All dplyr functions work similarly:

- The first argument is a data frame
- Subsequent argument describe what to do
- Output is a new data frame

# Prerequisites

- Download and load packages: `library(nycflights13)`  
`library(tidyverse)`

```
flights
#> # A tibble: 336,776 x 19
#>   year month   day dep_time sched_dep_time dep_delay arr_time
#>   <int> <int> <int>     <int>           <int>     <dbl>     <int>
#> 1 2013     1     1      517              515        2       830
#> 2 2013     1     1      533              529        4       850
#> 3 2013     1     1      542              540        2       923
#> 4 2013     1     1      544              545       -1      1004
#> 5 2013     1     1      554              600       -6       812
#> 6 2013     1     1      554              558       -4       740
#> # ... with 3.368e+05 more rows, and 12 more variables: sched_arr_time <int>,
#> #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#> #   minute <dbl>, time_hour <dttm>
```

# Your turn!

1. Are there vignettes for the **dplyr** package?
2. Can you find additional documentation explaining the **flights** dataset?
3. What variables are in the **flights** dataset? How many rows of data are in the **flights** dataset?

# Solution

1. Are there vignettes for the dplyr package?

```
# Lists topics  
vignette(package = "dplyr")  
# Select a topic  
vignette(package = "dplyr", topic = "dplyr")
```

2. Can you find additional documentation explaining the **flights** dataset?

```
?flights
```

3. What variables are in the **flights** dataset? How many rows of data are in the **flights** dataset?

```
str(flights)
```

```
dim(flights)
```

# **filter()**

pick observations by their values

# Basic filtering

```
filter(flights, month == 1)
# A tibble: 27,004 × 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
  <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>      <dbl>
1 2013     1     1      517          515       2     830        819       11
2 2013     1     1      533          529       4     850        830       20
3 2013     1     1      542          540       2     923        850       33
4 2013     1     1      544          545      -1    1004       1022      -18
5 2013     1     1      554          600      -6     812        837      -25
6 2013     1     1      554          558      -4     740        728       12
7 2013     1     1      555          600      -5     913        854       19
8 2013     1     1      557          600      -3     709        723      -14
9 2013     1     1      557          600      -3     838        846       -8
10 2013    1     1      558          600      -2     753        745        8
# ... with 26,994 more rows, and 10 more variables: carrier <chr>, flight <int>,
```

# Basic filtering

```
filter(flights, month == 1, day == 1)
# A tibble: 842 × 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
  <int> <int> <int>     <int>          <int>      <dbl>    <int>          <int>      <dbl>
1 2013     1     1      517          515        2     830          819       11
2 2013     1     1      533          529        4     850          830       20
3 2013     1     1      542          540        2     923          850       33
4 2013     1     1      544          545       -1    1004         1022      -18
5 2013     1     1      554          600       -6     812          837      -25
6 2013     1     1      554          558       -4     740          728       12
7 2013     1     1      555          600       -5     913          854       19
8 2013     1     1      557          600       -3     709          723      -14
9 2013     1     1      557          600       -3     838          846       -8
10 2013    1     1      558          600       -2     753          745        8
# ... with 832 more rows, and 10 more variables: carrier <chr>, flight <int>, tailnum <chr>,
```

# Basic filtering

```
filter(flights, month == 1, day == 1, dep_delay > 0)
# A tibble: 352 × 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
  <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>     <dbl>
1 2013     1     1      517          515       2         830        819       11
2 2013     1     1      533          529       4         850        830       20
3 2013     1     1      542          540       2         923        850       33
4 2013     1     1      601          600       1         844        850      -6
5 2013     1     1      608          600       8         807        735       32
6 2013     1     1      611          600      11         945        931       14
7 2013     1     1      613          610       3         925        921       4
8 2013     1     1      623          610      13         920        915       5
9 2013     1     1      632          608      24         740        728      12
10 2013    1     1      644          636       8         931        940      -9
# ... with 342 more rows, and 10 more variables: carrier <chr>, flight <int>, tailnum <chr>,
```

# Save a new data frame

```
# Save data frame using assignment operator "<-"  
jan1 <- filter(flights, month == 1, day == 1)  
  
# Save and view filtered data frame by wrapping entire code with ()  
(dec25 <- filter(flights, month == 12, day == 25))
```

# Logical tests

```
12 == 12
```

```
12 <= c(12, 11)
```

```
12 %in% c(12, 11, 8)
```

```
x <- c(12, NA, 11, NA, 8)  
is.na(x)
```

|               |                          |
|---------------|--------------------------|
| <             | Less than                |
| >             | Greater than             |
| ==            | Equal to                 |
| <=            | Less than or equal to    |
| >=            | Greater than or equal to |
| !=            | Not equal to             |
| %in%          | Group membership         |
| <b>is.na</b>  | Is NA                    |
| <b>!is.na</b> | Is not NA                |

# Comparison

- What will these operations produce?

- 1) `filter(flights, month == 12)`
- 2) `filter(flights, month != 12)`
- 3) `filter(flights, month %in% c(11, 12))`
- 4) `filter(flights, arr_delay <= 120)`
- 5) `filter(flights, !(arr_delay <= 120))`
- 6) `filter(flights, is.na(tailnum))`

# Multiple logical tests

```
12 == 12 & 12 < 14
```

```
12 == 12 & 12 < 10
```

```
12 == 12 | 12 < 10
```

```
any(12 == 12, 12 < 10)
```

```
all(12 == 12, 12 < 10)
```

## ?base::Logical

|     |                    |
|-----|--------------------|
| &   | boolean and        |
|     | boolean or         |
| xor | exclusively x or y |
| !   | not                |
| any | any true           |
| all | all true           |

# Multiple comparisons

```
# Using comma is same as using &
filter(flights, month == 12, day == 25)
filter(flights, month == 12 & day == 25)

# Use %in% as a shortcut for |
filter(flights, month == 11 | month == 12)
filter(flights, month %in% c(11, 12))

# Are the outputs the same?
filter(flights, !(arr_delay > 120 | dep_delay > 120))
filter(flights, arr_delay <= 120, dep_delay <= 120)
```

# Your turn!

Find the number of flights that:

- 1) Had an arrival delay of two or more hours
- 2) Flew to Houston (IAH or HOU)
- 3) Were operated by United, American, or Delta
- 4) Departed in summer (July, August, and September) and have missing departure time values
- 5) Arrived more than two hours late, but didn't leave late
- 6) Were delayed by at least an hour, but made up over 30 minutes in flight

# Solution

```
# Had an arrival delay of two or more hours
filter(flights, arr_delay >= 120)

# Flew to Houston (IAH or HOU)
filter(flights, dest %in% c("IAH", "HOU"))

# Were operated by United, American, or Delta
filter(flights, carrier %in% c("UA", "AA", "DL"))

# Departed in summer (July, August, and September) and have missing departure time values
filter(flights, month %in% c(7, 8, 9) & is.na(dep_time))

# Arrived more than two hours late, but didn't leave late
filter(flights, arr_delay > 120, dep_delay <= 0)

# Were delayed by at least an hour, but made up over 30 minutes in flight
filter(flights, dep_delay >= 60, arr_delay <= (dep_delay - 30))
```

# arrange()

Reorder the rows

# Ordering your data

```
arrange(flights, dep_delay)

# A tibble: 336,776 × 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
  <int> <int> <int>    <int>        <int>    <dbl>    <int>        <int>    <dbl> <chr> <int>
1 2013     12     7    2040         2123     -43       40      2352      48     B6    97
2 2013      2     3    2022         2055     -33      2240      2338     -58     DL   1715
3 2013     11    10    1408         1440     -32      1549      1559     -10     EV   5713
4 2013      1    11    1900         1930     -30      2233      2243     -10     DL   1435
5 2013      1    29    1703         1730     -27      1947      1957     -10     F9    837
6 2013      8     9     729          755     -26      1002      955       7     MQ   3478
7 2013     10    23    1907         1932     -25      2143      2143       0     EV   4361
8 2013      3    30    2030         2055     -25      2213      2250     -37     MQ   4573
9 2013      3     2    1431         1455     -24      1601      1631     -30     9E   3318
10 2013      5     5     934          958     -24      1225      1309     -44     B6   375
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

# Ordering your data

```
arrange(flights, dep_delay, arr_delay)

# A tibble: 336,776 × 19
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
  <int> <int> <int>    <int>        <int>    <dbl>    <int>        <int>    <dbl> <chr>   <int>
1  2013     12     7    2040         2123     -43       40        2352      48     B6     97
2  2013      2     3    2022         2055     -33      2240        2338     -58     DL    1715
3  2013     11    10    1408         1440     -32      1549        1559     -10     EV    5713
4  2013      1    11    1900         1930     -30      2233        2243     -10     DL    1435
5  2013      1    29    1703         1730     -27      1947        1957     -10     F9     837
6  2013      8     9     729          755     -26      1002        955      7      MQ    3478
7  2013      3    30    2030         2055     -25      2213        2250     -37     MQ    4573
8  2013     10    23    1907         1932     -25      2143        2143      0      EV    4361
9  2013      5     5     934          958     -24      1225        1309     -44     B6     375
10 2013      9    18    1631         1655     -24      1812        1845     -33     AA    2223
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

# Ordering your data

```
arrange(flights, desc(dep_delay))  
# A tibble: 336,776 × 19  
# ... with 336,766 more rows, and 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,  
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>  
  
  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight  
  <int> <int> <int>    <int>        <int>    <dbl>    <int>        <int>    <dbl> <chr>   <int>  
1  2013     1     9      641            900    1301    1242            1530    1272    HA      51  
2  2013     6    15     1432           1935    1137    1607            2120    1127    MQ      3535  
3  2013     1    10     1121           1635    1126    1239            1810    1109    MQ      3695  
4  2013     9    20     1139           1845    1014    1457            2210    1007    AA      177  
5  2013     7    22      845            1600    1005    1044            1815    989     MQ      3075  
6  2013     4    10     1100           1900    960     1342            2211    931     DL      2391  
7  2013     3    17     2321            810    911     135             1020    915     DL      2119  
8  2013     6    27      959            1900    899     1236            2226    850     DL      2007  
9  2013     7    22     2257            759    898     121             1026    895     DL      2047  
10 2013    12      5      756           1700    896     1058            2020    878     AA      172
```

# Missing data order

- Note: missing values are always sorted at the end:

```
df <- tibble(x = c(5, 2, 5, NA))  
# A tibble: 4 × 1  
  
      x  
    <dbl>  
1     5  
2     2  
3     5  
4    NA
```

```
arrange(df, x)  
# A tibble: 4 × 1  
  
      x  
    <dbl>  
1     2  
2     5  
3     5  
4    NA
```

```
arrange(df, desc(x))  
# A tibble: 4 × 1  
  
      x  
    <dbl>  
1     5  
2     5  
3     2  
4    NA
```

# Your turn!

- 1)Sort flights to find the most delayed flights
- 2)Find the flights that left earliest
- 3)Which flights traveled the longest?
- 4)Which traveled the shortest?

# Solution

```
# Sort flights to find the most delayed flights  
arrange(flights, desc(dep_delay))
```

```
# Find the flights that left earliest  
arrange(flights, dep_time)
```

```
# Which flights travelled the longest?  
arrange(flights, desc(distance))
```

```
# Which travelled the shortest?  
arrange(flights, distance)
```

# select()

Pick variables by their names

# Selecting variables

```
select(flights, year, month, day)  
# A tibble: 336,776 × 3
```

year month day

<int> <int> <int>

|    |      |   |   |
|----|------|---|---|
| 1  | 2013 | 1 | 1 |
| 2  | 2013 | 1 | 1 |
| 3  | 2013 | 1 | 1 |
| 4  | 2013 | 1 | 1 |
| 5  | 2013 | 1 | 1 |
| 6  | 2013 | 1 | 1 |
| 7  | 2013 | 1 | 1 |
| 8  | 2013 | 1 | 1 |
| 9  | 2013 | 1 | 1 |
| 10 | 2013 | 1 | 1 |

# ... with 336,766 more rows

```
select(flights, year:day)  
# A tibble: 336,776 × 3
```

year month day

<int> <int> <int>

|    |      |   |   |
|----|------|---|---|
| 1  | 2013 | 1 | 1 |
| 2  | 2013 | 1 | 1 |
| 3  | 2013 | 1 | 1 |
| 4  | 2013 | 1 | 1 |
| 5  | 2013 | 1 | 1 |
| 6  | 2013 | 1 | 1 |
| 7  | 2013 | 1 | 1 |
| 8  | 2013 | 1 | 1 |
| 9  | 2013 | 1 | 1 |
| 10 | 2013 | 1 | 1 |

# ... with 336,766 more rows

# Deselecting variable

- Deselect with “-” sign

```
select(flights, -(year:day))

# A tibble: 336,776 × 16
  dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier flight
  <int>      <int>     <dbl>    <int>      <int>     <dbl>   <chr>   <int>
1     517        515       2       830        819       11     UA     1545
2     533        529       4       850        830       20     UA     1714
3     542        540       2       923        850       33     AA     1141
4     544        545      -1      1004       1022      -18     B6      725
5     554        600      -6       812        837      -25     DL      461
6     554        558      -4       740        728       12     UA     1696
7     555        600      -5       913        854       19     B6      507
8     557        600      -3       709        723      -14     EV     5708
9     557        600      -3       838        846       -8     B6      79
10    558        600      -2       753        745        8     AA     301
```

# Useful select functions

|                      |  |
|----------------------|--|
| -                    | Select everything but                                    |
| :                    | Select range   |
| <b>contains()</b>    | Select columns whose name contains a character string    |
| <b>ends_with()</b>   | Select columns whose name ends with a string             |
| <b>everything()</b>  | Select every column                                      |
| <b>matches()</b>     | Select columns whose name matches a regular expression   |
| <b>num_range()</b>   | Select columns named x1, x2, x3, x4, x5                  |
| <b>one_of()</b>      | Select columns whose names are in a group of names       |
| <b>starts_with()</b> | Select columns whose name starts with a character string |

# select by name patterns

```
select(flights, ends_with("time"))
# A tibble: 336,776 × 5
  dep_time sched_dep_time arr_time sched_arr_time air_time
  <int>        <int>     <int>        <int>      <dbl>
1     517          515     830          819       227
2     533          529     850          830       227
3     542          540     923          850       160
4     544          545    1004         1022       183
5     554          600     812          837       116
6     554          558     740          728       150
7     555          600     913          854       158
8     557          600     709          723       53
9     557          600     838          846       140
10    558          600     753          745       138
# ... with 336,766 more rows
```

# Multiple name patterns

```
select(flights, c(carrier, ends_with("time"), contains("delay")))
# A tibble: 336,776 × 8
  carrier dep_time sched_dep_time arr_time sched_arr_time air_time dep_delay arr_delay
  <chr>     <int>        <int>     <int>        <int>      <dbl>     <dbl>      <dbl>
1 UA         517          515       830        819       227        2       11
2 UA         533          529       850        830       227        4       20
3 AA         542          540       923        850       160        2       33
4 B6         544          545      1004       1022       183      -1      -18
5 DL         554          600       812        837       116      -6      -25
6 UA         554          558       740        728       150      -4       12
7 B6         555          600       913        854       158      -5       19
8 EV         557          600       709        723       53       -3      -14
9 B6         557          600       838        846       140      -3       -8
10 AA        558          600       753        745       138      -2        8
# ... with 336,766 more rows
```

# Reorder variables

```
select(flights, time_hour, air_time, everything())
# A tibble: 336,776 × 19
# ... with 10 more variables: sched_dep_time <int>,
#   dep_delay <dbl>, arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, distance <dbl>, hour <dbl>, minute <dbl>
```

# Renaming variables

```
rename(flights, newName = dep_delay)
# A tibble: 336,776 × 19
  year month   day dep_time sched_dep_time  newName arr_time sched_arr_time arr_delay
  <int> <int> <int>    <int>          <int>    <dbl>    <int>          <int>        <dbl>
1 2013     1     1      517            515       2       830         819        11
2 2013     1     1      533            529       4       850         830        20
3 2013     1     1      542            540       2       923         850        33
4 2013     1     1      544            545      -1      1004        1022       -18
5 2013     1     1      554            600      -6       812         837       -25
6 2013     1     1      554            558      -4       740         728        12
7 2013     1     1      555            600      -5       913         854        19
8 2013     1     1      557            600      -3       709         723       -14
9 2013     1     1      557            600      -3       838         846        -8
10 2013    1     1      558            600      -2       753         745         8
# ... with 336,766 more rows, and 10 more variables: carrier <chr>, flight <int>,
# tailnum <chr>, origin <chr>, dest <chr>, air time <dbl>, distance <dbl>,
```

# Your turn!

- 1) What happens if you include the name of a variable multiple times in a `select()` call?
- 2) What does the `one_of()` function do? Why might it be helpful in conjunction with this vector?

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")
```

- 3) Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

```
select(flights, contains("TIME"))
```

# Solution

```
# 1. what happens if you call a variable multiple times in select()
select(flights, month, month)
# A tibble: 336,776 × 1
  month
  <int>
1     1
2     1
3     1
4     1
5     1
6     1
7     1
```

# Solution

```
# 2. what does one_of() do?  
# Matches variable names in a character vector.  
  
vars <- c("year", "month", "day", "dep_delay", "arr_delay")  
select(flights, one_of(vars))  
  
## # A tibble: 336,776 x 5  
##       year   month   day dep_delay arr_delay  
##   <int> <int> <int>     <dbl>      <dbl>  
## 1 2013     1     1        2        11  
## 2 2013     1     1        4        20  
## 3 2013     1     1        2        33  
## 4 2013     1     1       -1       -18  
## 5 2013     1     1       -6       -25
```

# Solution

*#3 Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?*

*# The default helper functions are insensitive to case. This can be changes by setting `ignore.case=FALSE`.*

```
select(flights, contains("TIME"))
select(flights, contains("TIME", ignore.case = FALSE))
```

# mutate()

create new variables

# create new variables

```
# Smaller dataset for demo
flights_sml <- select(flights,
  year:day,
  ends_with("delay"),
  distance,
  air_time
)
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

# Useful tips

```
# You can refer to columns that you've just created
mutate(flights_sml,
       gain = dep_delay - arr_delay,
       hours = air_time / 60,
       gain_per_hour = gain / hours
     )
```

```
# If you only want to keep the new variables
transmute(flights,
           gain = dep_delay - arr_delay,
           hours = air_time / 60,
           gain_per_hour = gain / hours
         )
```

# Useful creation functions

- There are a wide variety of functions you can use with **mutate()**

| Functions  | Description                      |
|--|----------------------------------|
| <code>+, -, *, /, ^</code>                         | arithmetic                       |
| <code>x / sum(x)</code>                            | arithmetic w/aggregate functions |
| <code>%/%, %%</code>                               | modular arithmetic               |
| <code>log, exp, sqrt</code>                        | transformations                  |
| <code>lag, lead</code>                             | offsets                          |
| <code>cumsum,<br/>cumprod, cum...</code>           | cum/rolling aggregates           |
| <code>&gt;, &gt;=, &lt;, &lt;=, !<br/>=, ==</code> | logical comparisons              |
| <code>min_rank,<br/>dense_rank, etc</code>         | ranking                          |
| <code>between</code>                               | are values between a and b?      |
| <code>ntile</code>                                 | bin values into buckets          |

# arithmetic

```
transmute(flights,
  normalized_delay =
    dep_delay / (mean(dep_delay, na.rm = TRUE)))
```

| Functions                                     | Description                      |
|---|----------------------------------|
| <code>+, -, *, /, ^</code>                    | arithmetic                       |
| <code>x / sum(x)</code>                       | arithmetic w/aggregate functions |
| <code>%/%, %%</code>                          | modular arithmetic               |
| <code>log, exp, sqrt</code>                   | transformations                  |
| <code>lag, lead</code>                        | offsets                          |
| <code>cumsum, cumprod, cum...</code>          | cum/rolling aggregates           |
| <code>&gt;, &gt;=, &lt;, &lt;=, !=, ==</code> | logical comparisons              |
| <code>min_rank, dense_rank, etc</code>        | ranking                          |
| <code>between</code>                          | are values between a and b?      |
| <code>ntile</code>                            | bin values into buckets          |

# Modular arithmetic

```
transmute(flights,  
          dep_time,  
          hour = dep_time %/% 100,  
          minute = dep_time %% 100  
)
```

%% (Integer division)  
%% (Remainder)

```
#> # A tibble: 336,776 x 3  
#>   dep_time    hour  minute  
#>   <int>     <dbl>    <dbl>  
#> 1      517      5       17  
#> 2      533      5       33  
#> 3      542      5       42  
#> 4      544      5       44  
#> 5      554      5       54
```

| Functions                    | Description                      |
|------------------------------|----------------------------------|
| +, -, *, /, ^                | arithmetic                       |
| x / sum(x)                   | arithmetic w/aggregate functions |
| %/%, %%                      | modular arithmetic               |
| log, exp, sqrt               | transformations                  |
| lag, lead                    | offsets                          |
| cumsum,<br>cumprod, cum...   | cum/rolling aggregates           |
| >, >=, <, <=, !<br>=, ==     | logical comparisons              |
| min_rank,<br>dense_rank, etc | ranking                          |
| between                      | are values between a and b?      |
| ntile                        | bin values into buckets          |

# non-linear transformations

```
transmute(flights,  
  log_air_time = log2(air_time),  
  exp_delay = exp(dep_delay))
```

| Functions                    | Description                      |
|------------------------------|----------------------------------|
| +, -, *, /, ^                | arithmetic                       |
| x / sum(x)                   | arithmetic w/aggregate functions |
| %/%, %%                      | modular arithmetic               |
| log, exp, sqrt               | transformations                  |
| lag, lead                    | offsets                          |
| cumsum,<br>cumprod, cum...   | cum/rolling aggregates           |
| >, >=, <, <=, !<br>=, ==     | logical comparisons              |
| min_rank,<br>dense_rank, etc | ranking                          |
| between                      | are values between a and b?      |
| ntile                        | bin values into buckets          |

# offsets & cumulative aggregates

```
transmute(flights,
  dep_delay = dep_delay,
  lag_delay = lag(dep_delay),
  sum_delay = cumsum(dep_delay))
```

```
## # A tibble: 336,776 x 3
##   dep_delay Lag_delay sum_delay
##       <dbl>      <dbl>      <dbl>
## 1 2          NA        2
## 2 4          2         6
## 3 2          4         8
## 4 -1         2         7
## 5 -6         -1        1
```

| Functions                 | Description                      |
|---------------------------|----------------------------------|
| +, -, *, /, ^             | arithmetic                       |
| x / sum(x)                | arithmetic w/aggregate functions |
| %/%, %%                   | modular arithmetic               |
| log, exp, sqrt            | transformations                  |
| lag, lead                 | offsets                          |
| cumsum, cumprod, cum...   | cum/rolling aggregates           |
| >, >=, <, <=, !   =, ==   | logical comparisons              |
| min_rank, dense_rank, etc | ranking                          |
| between                   | are values between a and b?      |
| ntile                     | bin values into buckets          |

# Ranking

```
y <- c(1, 2, 2, NA, 3, 4)

min_rank(y)
#> [1] 1 2 2 NA 4 5

row_number(y)
#> [1] 1 2 3 NA 4 5

dense_rank(y)
#> [1] 1 2 2 NA 3 4

percent_rank(y)
#> [1] 0.00 0.25 0.25     NA 0.75 1.00

cume_dist(y)
#> [1] 0.2 0.6 0.6   NA 0.8 1.0
```

| Functions                    | Description                      |
|------------------------------|----------------------------------|
| +, -, *, /, ^                | arithmetic                       |
| x / sum(x)                   | arithmetic w/aggregate functions |
| %/%, %%                      | modular arithmetic               |
| log, exp, sqrt               | transformations                  |
| lag, lead                    | offsets                          |
| cumsum,<br>cumprod, cum...   | cum/rolling aggregates           |
| >, >=, <, <=, !<br>=, ==     | logical comparisons              |
| min_rank,<br>dense_rank, etc | ranking                          |
| between                      | are values between a and b?      |
| ntile                        | bin values into buckets          |

# ntile

```
transmute(flights,  
  arr_delay = arr_delay,  
  bucket = ntile(arr_delay, 10))
```

```
## # A tibble: 336,776 x 2  
##       arr_delay   bucket  
##       <dbl>     <int>  
## 1      11         8  
## 2      20         8  
## 3      33         9  
## 4     -18         3  
## 5     -25         2  
## 6      12         8  
## 7      19         8  
## 8     -14         3  
## 9      -8         5  
## 10     8          7  
## # ... with 336,766 more rows
```

| Functions                    | Description                      |
|------------------------------|----------------------------------|
| +, -, *, /, ^                | arithmetic                       |
| x / sum(x)                   | arithmetic w/aggregate functions |
| %/%, %%                      | modular arithmetic               |
| log, exp, sqrt               | transformations                  |
| lag, lead                    | offsets                          |
| cumsum,<br>cumprod, cum...   | cum/rolling aggregates           |
| >, >=, <, <=, !<br>=, ==     | logical comparisons              |
| min_rank,<br>dense_rank, etc | ranking                          |
| between                      | are values between a and b?      |
| ntile                        | bin values into buckets          |

# Your turn!

- 1) Create a new variable **distance\_km** that converts distance in miles to kilometres
- 2) Create a **time\_per\_km** variable based on **air\_time** and **distance\_km**.

# Solution

```
transmute(flights,  
          distance_km = distance * 1.60934,  
          time_per_km = air_time / distance_km)
```

# **summarise() & group\_by()**

Create grouped summaries

# summarise()

- **summarise()** collapses a data frame into a single row

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

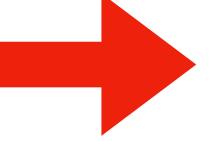
```
#> # A tibble: 1 x 1
#>   delay
#>   <dbl>
#> 1 12.6
```

Why is this important?  
Try without this argument.

- **summarise()** is not really useful without pairing it with **group\_by()**

# group\_by()

| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |

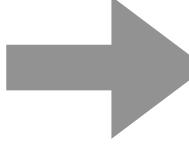


| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |

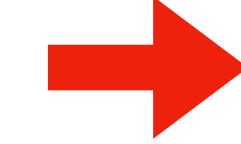
group\_by(data, country)

# group\_by()

| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |



| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |



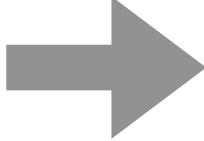
| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |

group\_by(data, country)

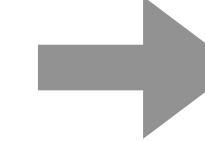
group\_by(data, country, year)

# group\_by()

| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |



| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |



| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |



| country     | year | sex    | cases |
|-------------|------|--------|-------|
| Afghanistan | 1999 | female | 1     |
| Afghanistan | 1999 | male   | 1     |
| Afghanistan | 2000 | female | 1     |
| Afghanistan | 2000 | male   | 1     |
| Brazil      | 1999 | female | 2     |
| Brazil      | 1999 | male   | 2     |
| Brazil      | 2000 | female | 2     |
| Brazil      | 2000 | male   | 2     |
| China       | 1999 | female | 3     |
| China       | 1999 | male   | 3     |
| China       | 2000 | female | 3     |
| China       | 2000 | male   | 3     |

group\_by(data, country)

group\_by(data, country, year)

ungroup(data)

# Your turn!

- 1) Which carrier had the largest mean departure delay?  
Smallest?
- 2) Which carrier had the largest difference between their max  
and min departure delay?
- 3) Which month has the largest variance for arrival delays?

# Solution

```
# Which carrier had the Largest mean departure delay? Smallest?  
by_carrier <- group_by(flights, carrier)  
summarise(by_carrier, delay = mean(dep_delay, na.rm = TRUE))
```

```
# Which carrier had the Largest difference between their max and min departure delay?  
summarise(by_carrier,  
          max = max(dep_delay, na.rm = TRUE),  
          min = min(dep_delay, na.rm = TRUE),  
          delta = max - min)
```

```
# Which month has the Largest variance for arrival delays?  
by_month <- group_by(flights, month)  
summarise(by_month, delay = sd(arr_delay, na.rm = TRUE))
```

# Pipe operator

Chaining functions together

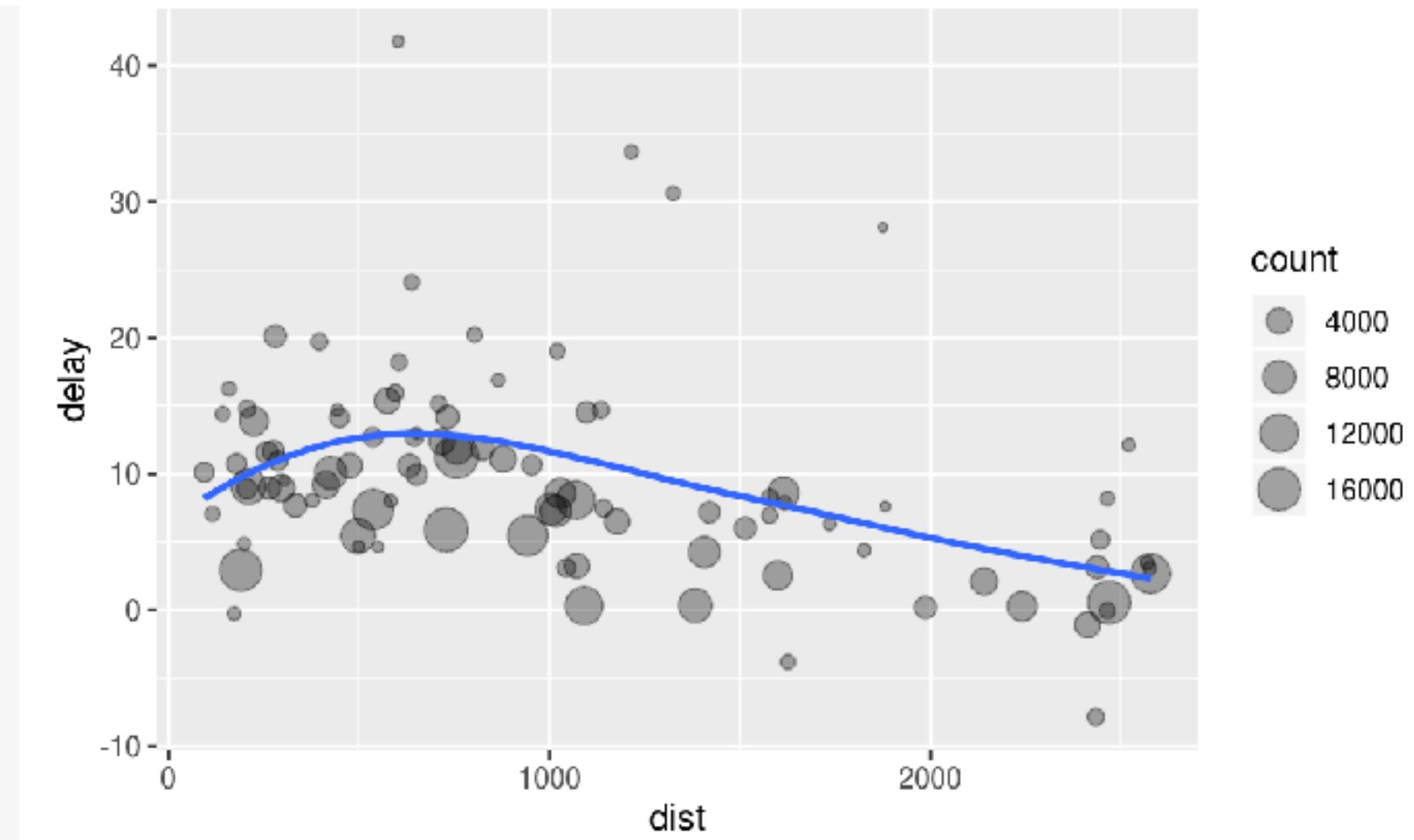
# Example

- Imagine that we want to explore the relationship between the distance and average delay for each location. Using what you know about dplyr, you might write code like this:

```
by_dest <- group_by(flights, dest)

delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")

ggplot(data = delay, mapping = aes(x = dist, y = delay)) +
  geom_point(aes(size = count), alpha = 1/3) +
  geom_smooth(se = FALSE)
```



# Example

- Imagine that we want to explore the relationship between the distance and average delay for each location. Using what you know about dplyr, you might write code like this:

```
by_dest <- group_by(flights, dest)
```

grouping by destination

```
delay <- summarise(by_dest,  
  count = n(),  
  dist = mean(distance, na.rm = TRUE),  
  delay = mean(arr_delay, na.rm = TRUE)  
)
```

summarising count, distance and arrival delay

```
delay <- filter(delay, count > 20, dest != "HNL")
```

filtering out low counts and Honolulu

```
ggplot(data = delay, mapping = aes(x = dist, y = delay)) +  
  geom_point(aes(size = count), alpha = 1/3) +  
  geom_smooth(se = FALSE)
```

creating a plot

# Example

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
```

```
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
```

Command/CTRL + Shift + M

# Pipe operator

```
# without pipe to subset "suv" cars  
filter(mpg, class == "suv")  
  
# equivalent form with pipe  
mpg %>% filter(class == "suv")
```

Command/CTRL + Shift + M

# Your turn!

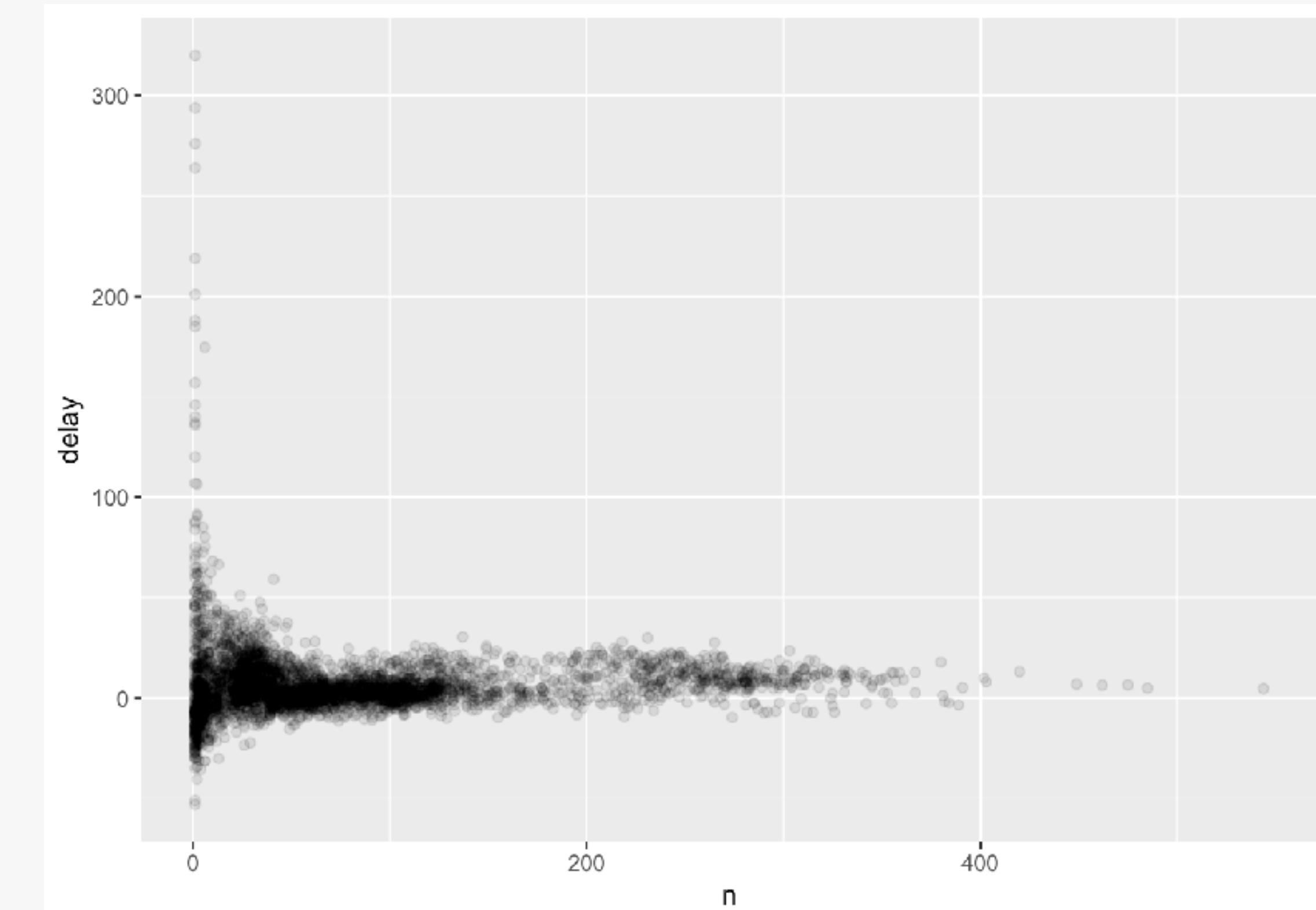
Using the pipe operator follow these steps with the flight data:

- 1) filter out missing values that exist in the tail number and arrival delay variables
- 2) group the data by tail number
- 3) calculate the average arrival delay and the number of observations by tail number
- 4) plot average arrival delay vs. number of observations

*What do these findings tell you?*

# Solution

```
flights %>%  
  filter(!is.na(tailnum), !is.na(arr_delay)) %>%  
  group_by(tailnum) %>%  
  summarise(delay = mean(arr_delay, na.rm = TRUE), n = n()) %>%  
  ggplot(aes(n, delay)) +  
  geom_point(alpha = .1)
```



# Your turn!

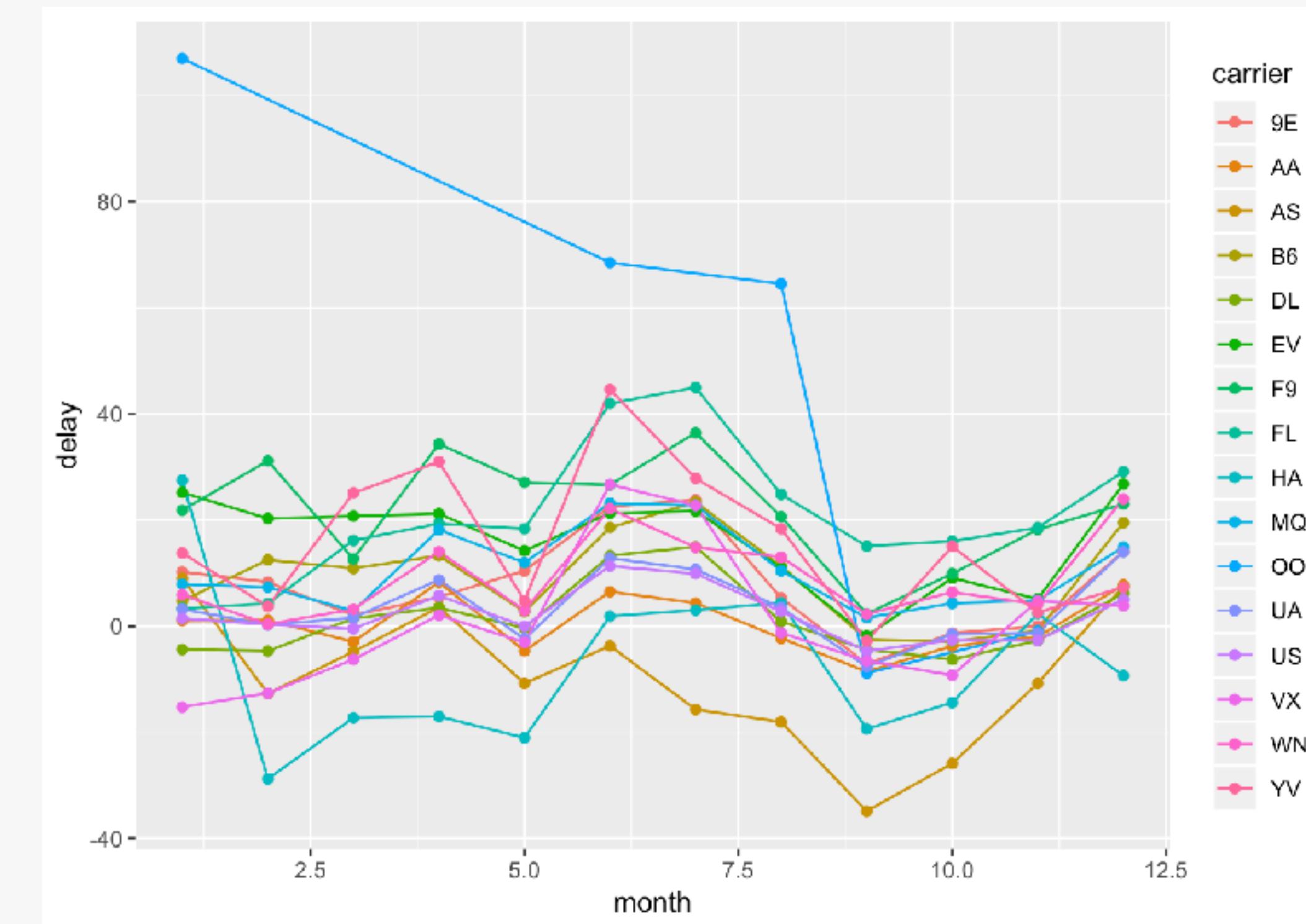
Using the pipe operator follow these steps with the flight data:

- 1) group by month and carrier
- 2) compute the average arrival delay
- 3) plot a line chart with month on the x-axis, average delay on the y-axis
- 4) colour-code each line by carrier or try facetting by carrier

*What do these findings tell you?*

# Solution

```
flights %>%
  group_by(month, carrier) %>%
  summarise(delay = mean(arr_delay, na.rm = TRUE)) %>%
  ggplot(aes(month, delay, color = carrier)) +
  geom_line()+
  geom_point()
```



*“Tidy datasets are all alike,  
but every messy dataset is messy in its own way.”*

*- Hadley Wickham*

# Tidy data

## data structure

# 3 rules of tidy data

- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1990 | 745    | 18367071   |
| Afghanistan | 2000 | 2666   | 20595360   |
| Brazil      | 1999 | 31737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 213766 | 128042583  |

variables

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 745    | 18367071   |
| Afghanistan | 2000 | 2666   | 20595360   |
| Brazil      | 1999 | 31737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 213766 | 128042583  |

observations

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 745    | 18367071   |
| Afghanistan | 2000 | 2666   | 20595360   |
| Brazil      | 1999 | 31737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 213766 | 128042583  |

values

# Your turn!

Is this a tidy table?

Table 1:

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 745    | 19987071   |
| Afghanistan | 2000 | 2666   | 20595360   |
| Brazil      | 1999 | 37737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 213766 | 1280428583 |

# Your turn!

Is this a tidy table?

Table 2:

| country     | year | type       | count      |
|-------------|------|------------|------------|
| Afghanistan | 1999 | cases      | 745        |
| Afghanistan | 1999 | population | 19987071   |
| Afghanistan | 2000 | cases      | 2666       |
| Afghanistan | 2000 | population | 20595360   |
| Brazil      | 1999 | cases      | 37737      |
| Brazil      | 1999 | population | 172006362  |
| Brazil      | 2000 | cases      | 80488      |
| Brazil      | 2000 | population | 174504898  |
| China       | 1999 | cases      | 212258     |
| China       | 1999 | population | 1272915272 |
| China       | 2000 | cases      | 213766     |
| China       | 2000 | population | 1280428583 |

# Your turn!

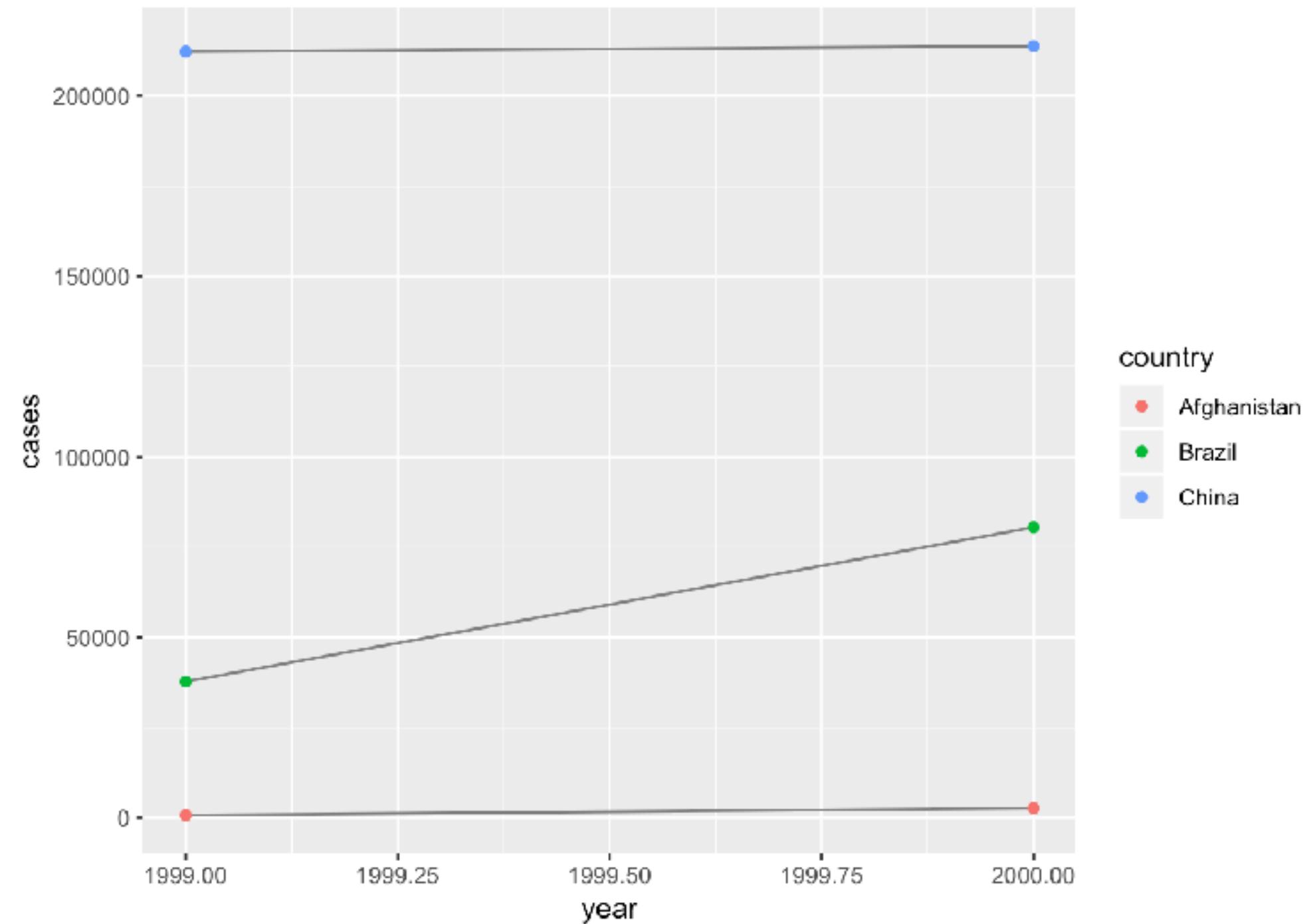
Is this a tidy table?

Table 3:

| country     | year | rate              |
|-------------|------|-------------------|
| Afghanistan | 1999 | 745/19987071      |
| Afghanistan | 2000 | 2666/20595360     |
| Brazil      | 1999 | 37737/172006362   |
| Brazil      | 2000 | 80488/174504898   |
| China       | 1999 | 212258/1272915272 |
| China       | 2000 | 213766/1280428583 |

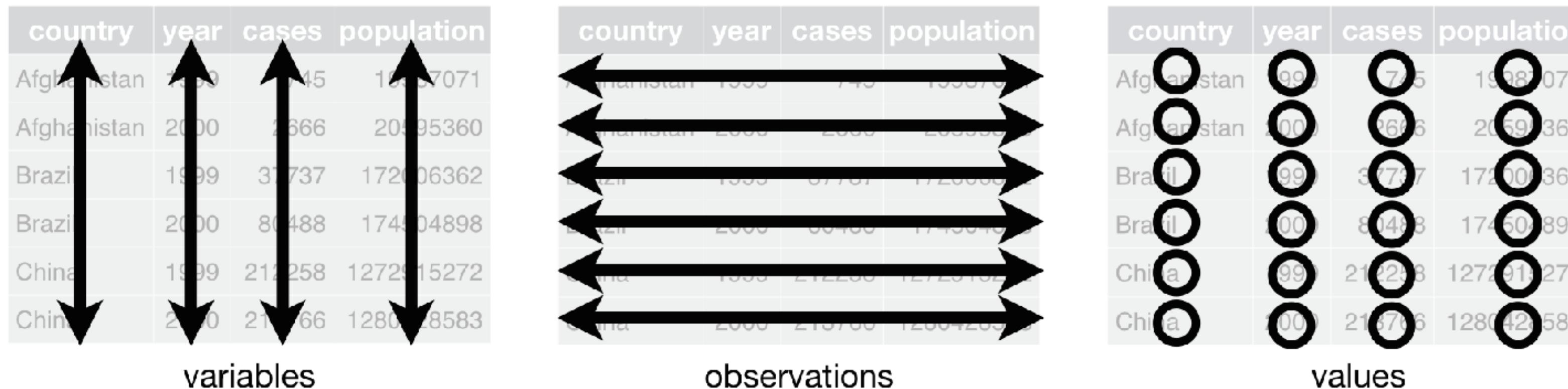
# Tidy data example

```
# using a sample dataset from tidyverse (tidyverse::table1)
# check the table View(tidyverse::table1)
ggplot(data = table1, mapping = aes(year, cases)) +
  geom_line(mapping = aes(group = country), colour = "grey50") +
  geom_point(mapping = aes(colour = country))
```



# 4 key functions

| Function   | Description                                    |
|------------|--|
| gather()   | transforms data from wide to long              |
| spread()   | transforms data from long to wide              |
| separate() | splits a single column into multiple columns   |
| unite()    | combines multiple columns into a single column |



# gather()

Transform data from wide to long

# gather() example

```
table4a %>%  
  gather(key = "year", value = "cases", `1999`, `2000`)
```

Collapsed columns

| country     | 1999   | 2000   | key   |
|-------------|--------|--------|-------|
| Afghanistan | 745    | 2666   |       |
| Brazil      | 37737  | 80488  | value |
| China       | 212258 | 213766 |       |

gather()

| country     | year | cases  |
|-------------|------|--------|
| Afghanistan | 1999 | 745    |
| Afghanistan | 2000 | 2666   |
| Brazil      | 1999 | 37737  |
| Brazil      | 2000 | 80488  |
| China       | 1999 | 212258 |
| China       | 2000 | 213766 |

# Your turn!

1. Download the data (bomber\_wide.rds) in the data folder
2. Import the .rds with `read_rds()` function
3. Reshape this data from wide to long

# Solution

```
# One example  
read_rds("data/bomber_wide.rds") %>%  
  gather(key = Year, value = Value, -c(Type, MD))  
  
# alternative  
read_rds("data/bomber_wide.rds") %>%  
  gather(key = Year, value = Value, `1996`:`2014`)
```

# spread()

Transform data from long to wide

# spread() example

```
table2 %>%  
  spread(key = type, value = count)
```

| country     | year | type       | key | count      | value |
|-------------|------|------------|-----|------------|-------|
| Afghanistan | 1999 | cases      |     | 745        |       |
| Afghanistan | 1999 | population |     | 19987071   |       |
| Afghanistan | 2000 | cases      |     | 2666       |       |
| Afghanistan | 2000 | population |     | 20595360   |       |
| Brazil      | 1999 | cases      |     | 37737      |       |
| Brazil      | 1999 | population |     | 172006362  |       |
| Brazil      | 2000 | cases      |     | 80488      |       |
| Brazil      | 2000 | population |     | 174504898  |       |
| China       | 1999 | cases      |     | 212258     |       |
| China       | 1999 | population |     | 1272915272 |       |
| China       | 2000 | cases      |     | 213766     |       |
| China       | 2000 | population |     | 1280428583 |       |

spread()

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 745    | 19987071   |
| Afghanistan | 2000 | 2666   | 20595360   |
| Brazil      | 1999 | 37737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 213766 | 1280428583 |

# Your turn!

1. Download the data (bomber\_long.rds) in the data folder
2. Import the .rds with `read_rds()` function
3. Reshape this data from long to wide

# Solution

```
read_rds("data/bomber_long.rds") %>%  
  spread(key = Output, value = Value)
```

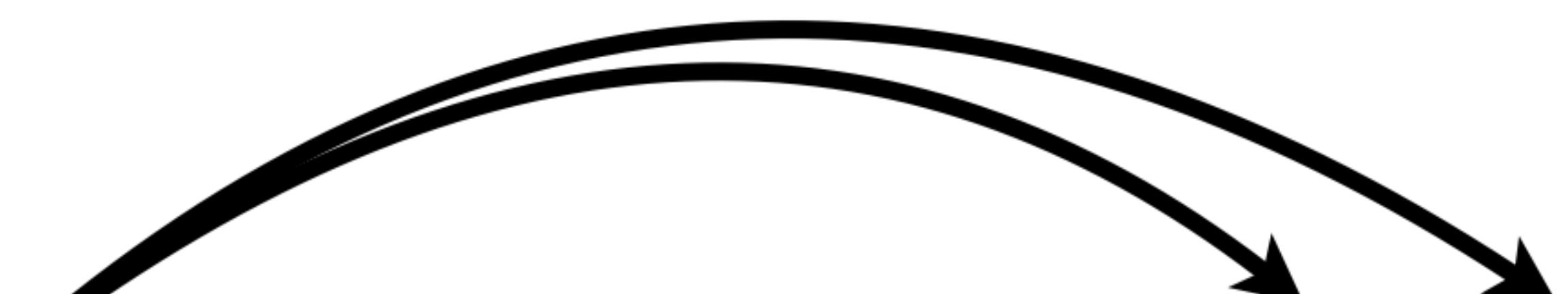
# separate()

split a single column into multiple columns

# separate() example

```
table3 %>%  
  separate(rate, into = c("cases", "population"), sep="/", convert = TRUE)
```

| country     | year | rate                       |
|-------------|------|----------------------------|
| Afghanistan | 1999 | <b>745</b> / 19987071      |
| Afghanistan | 2000 | <b>2666</b> / 20595360     |
| Brazil      | 1999 | <b>37737</b> / 172006362   |
| Brazil      | 2000 | <b>80488</b> / 174504898   |
| China       | 1999 | <b>212258</b> / 1272915272 |
| China       | 2000 | <b>213766</b> / 1280428583 |



The diagram illustrates the transformation of the 'rate' column into two new columns, 'cases' and 'population'. A curved arrow originates from the 'rate' column in the first table and points to the 'cases' column in the second table.

| country     | year | cases         | population |
|-------------|------|---------------|------------|
| Afghanistan | 1999 | <b>745</b>    | 19987071   |
| Afghanistan | 2000 | <b>2666</b>   | 20595360   |
| Brazil      | 1999 | <b>37737</b>  | 172006362  |
| Brazil      | 2000 | <b>80488</b>  | 174504898  |
| China       | 1999 | <b>212258</b> | 1272915272 |
| China       | 2000 | <b>213766</b> | 1280428583 |

table3

# separate() example

```
table3 %>%  
  separate(year, into = c("century", "year"), sep = 2)  
  
#> # A tibble: 6 x 4  
#>   country    century year   rate  
#>   <chr>       <chr>   <chr> <chr>  
#> 1 Afghanistan 19      99     745/19987071  
#> 2 Afghanistan 20      00     2666/20595360  
#> 3 Brazil       19      99     37737/172006362  
#> 4 Brazil       20      00     80488/174504898  
#> 5 China        19      99     212258/1272915272  
#> 6 China        20      00     213766/1280428583
```

# Your turn!

1. Download the data (bomber\_combined.rds) in the data folder
2. Import the .rds with `read_rds()` function
3. Separate the AC variable into “Type” and “MD”

# Solution

```
read_rds("data/bomber_combined.rds") %>%  
  separate(AC, into = c("Type", "MD"), sep = " ")
```

|   | Type   | MD  | FY   | Cost     | FH    | Gallons  |
|---|--------|-----|------|----------|-------|----------|
| 1 | Bomber | B-1 | 1996 | 72753781 | 26914 | 88594449 |
| 2 | Bomber | B-1 | 1997 | 71297263 | 25219 | 85484074 |
| 3 | Bomber | B-1 | 1998 | 84026805 | 24205 | 85259038 |
| 4 | Bomber | B-1 | 1999 | 71848336 | 23306 | 79323816 |
| 5 | Bomber | B-1 | 2000 | 58439777 | 25013 | 86230284 |
| 6 | Bomber | B-1 | 2001 | 94946077 | 25059 | 86892432 |

# unite()

combine multiple columns into a single column

# unite() example

```
table5 %>%  
  unite(new, century, year, sep = "")
```

| country     | year | rate                |
|-------------|------|---------------------|
| Afghanistan | 1999 | 745 / 19987071      |
| Afghanistan | 2000 | 2666 / 20595360     |
| Brazil      | 1999 | 37737 / 172006362   |
| Brazil      | 2000 | 80488 / 174504898   |
| China       | 1999 | 212258 / 1272915272 |
| China       | 2000 | 213766 / 1280428583 |

| country     | century | year | rate                |
|-------------|---------|------|---------------------|
| Afghanistan | 19      | 99   | 745 / 19987071      |
| Afghanistan | 20      | 0    | 2666 / 20595360     |
| Brazil      | 19      | 99   | 37737 / 172006362   |
| Brazil      | 20      | 0    | 80488 / 174504898   |
| China       | 19      | 99   | 212258 / 1272915272 |
| China       | 20      | 0    | 213766 / 1280428583 |



# Your turn!

1. Download the data (bomber\_prefix.rds) in the data folder
2. Import the .rds with `read_rds()` function
3. Unite the prefix and number columns into a “MD” variable with “-“ as separator

# Solution

```
read_rds("data/bomber_combined.rds") %>%  
  unite(MD, prefix, number, sep = "-")
```

|   | Type   | MD  | FY   | Output | Value |
|---|--------|-----|------|--------|-------|
| 1 | Bomber | B-1 | 1996 | FH     | 26914 |
| 2 | Bomber | B-1 | 1997 | FH     | 25219 |
| 3 | Bomber | B-1 | 1998 | FH     | 24205 |
| 4 | Bomber | B-1 | 1999 | FH     | 23306 |
| 5 | Bomber | B-1 | 2000 | FH     | 25013 |

# Your turn!

1. Download the data (bomber\_mess.rds) in the data folder
2. Import the .rds with `read_rds()` function
3. Clean this data up so it looks like:

```
# A tibble: 57 × 6
  Type     MD    FY    Cost    FH Gallons
  <chr>   <chr> <chr> <int> <int>   <int>
1 Bomber  B-1   1996 72753781 26914 88594449
2 Bomber  B-1   1997 71297263 25219 85484074
3 Bomber  B-1   1998 84026805 24205 85259038
4 Bomber  B-1   1999 71848336 23306 79323816
5 Bomber  B-1   2000 58439777 25013 86230284
6 Bomber  B-1   2001 94946077 25059 86892432
7 Bomber  B-1   2002 96458536 26581 89198262
8 Bomber  B-1   2003 68650070 21491 74485788
9 Bomber  B-1   2004 101895634 28118 101397707
10 Bomber B-1   2005 124816690 21859 78410415
# ... with 47 more rows
```

# Solution

```
read_rds("data/bomber_mess.rds") %>%  
  unite(col = MD, prefix:number, sep = "-") %>%  
  separate(Metric, into = c("FY", "Output")) %>%  
  spread(Output, Value) %>%  
  as_tibble()
```

# Your turn!

Let's apply **dplyr** and **tidyr** functions you have learned.

- **tidyr**::**:who** is a dataset of TB cases broken down by year, country, age, gender and diagnosis methods
- Examine the table and its documentation, and clean the data into a tidy data
- Hint:
  1. Gather columns and remove missing values
  2. Separate the values like “new\_sp\_m014” into columns
  3. Separate the gender and age
  4. Drop redundant columns using **dplyr**::**select()**
- Create a graph with the tidy data

# Solution

1. Gather new\_sp\_m014:newrel\_f65 columns and remove missing values

```
who1 <- who %>%
  gather(key = "key", value = "cases", new_sp_m014:newrel_f65, na.rm = TRUE)

## # A tibble: 76,046 x 6
##   country     iso2   iso3   year key      cases
##   * <chr>      <chr>  <chr>  <int> <chr>    <int>
## 1 Afghanistan AF     AFG     1997 new_sp_m014     0
## 2 Afghanistan AF     AFG     1998 new_sp_m014    30
## 3 Afghanistan AF     AFG     1999 new_sp_m014     8
## 4 Afghanistan AF     AFG     2000 new_sp_m014    52
## 5 Afghanistan AF     AFG     2001 new_sp_m014   129
```

# Solution

## 2. Clean up the key column, e.g. new\_sp\_m014

- **new** encodes new cases of TB
- next 3 letters describe the type of TB:
  - **rel** = relapse
  - **ep** = extrapulmonary TB
  - **sn** = could not be diagnosed by a pulmonary smear (smear negative)
  - **sp** = smear positive
- **m** = males, **f** = females
- age group: **014** = 1 - 14 years old age group

```
## # A tibble: 76,046 x 6
##   country     iso2   iso3   year   key    cases
##   * <chr>      <chr>  <chr>  <int>  <chr>  <int>
## 1 Afghanistan AF     AFG    1997 new_sp_m014     0
## 2 Afghanistan AF     AFG    1998 new_sp_m014    30
```

# Solution

2. But, there is an inconsistency in naming the key.

- eg. `new_rel` and `newrel`
- use `stringr::str_replace()` function to replace `newrel` with `new_rel`

```
who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
```

|   | country     | iso2  | iso3  | year  | key         | cases |
|---|-------------|-------|-------|-------|-------------|-------|
|   | <chr>       | <chr> | <chr> | <int> | <chr>       | <int> |
| 1 | Afghanistan | AF    | AFG   | 1997  | new_sp_m014 | 0     |
| 2 | Afghanistan | AF    | AFG   | 1998  | new_sp_m014 | 30    |
| 3 | Afghanistan | AF    | AFG   | 1999  | new_sp_m014 | 8     |
| 4 | Afghanistan | AF    | AFG   | 2000  | new_sp_m014 | 52    |
| 5 | Afghanistan | AF    | AFG   | 2001  | new_sp_m014 | 129   |

# Solution

2. Now use **separate()** to split key into 3 columns

```
who3 <- who2 %>%  
  separate(key, into = c("new", "type", "sex-age"), sep = "_")
```

```
##       country      iso2    iso3   year new    type `sex-age` cases  
##       <chr>       <chr>  <chr> <int> <chr> <chr> <chr>     <int>  
## 1 Afghanistan AF     AFG 1997 new    sp    m014        0  
## 2 Afghanistan AF     AFG 1998 new    sp    m014       30  
## 3 Afghanistan AF     AFG 1999 new    sp    m014        8  
## 4 Afghanistan AF     AFG 2000 new    sp    m014      52  
## 5 Afghanistan AF     AFG 2001 new    sp    m014     129
```

# Solution

## 3. Separate **sex-age** into 2 columns

```
who4 <- who3 %>%
  separate(`sex-age`, into = c("sex", "age"), sep = 1)
```

```
##   country    iso2  iso3  year new type sex  age cases
##   <chr>      <chr> <chr> <int> <chr> <chr> <chr> <chr> <int>
## 1 Afghanistan AF   AFG   1997 new   sp    m   014    0
## 2 Afghanistan AF   AFG   1998 new   sp    m   014   30
## 3 Afghanistan AF   AFG   1999 new   sp    m   014    8
## 4 Afghanistan AF   AFG   2000 new   sp    m   014   52
## 5 Afghanistan AF   AFG   2001 new   sp    m   014  129
```

# Solution

## 4. Drop redundant columns:

```
who5 <- who4 %>%
  select(-new, -iso2, -iso3)
```

|   | country     | year  | var   | sex   | age   | value |
|---|-------------|-------|-------|-------|-------|-------|
|   | <chr>       | <int> | <chr> | <chr> | <chr> | <int> |
| 1 | Afghanistan | 1997  | sp    | m     | 014   | 0     |
| 2 | Afghanistan | 1998  | sp    | m     | 014   | 30    |
| 3 | Afghanistan | 1999  | sp    | m     | 014   | 8     |
| 4 | Afghanistan | 2000  | sp    | m     | 014   | 52    |
| 5 | Afghanistan | 2001  | sp    | m     | 014   | 129   |

# Solution

You can build up a complete piped operations as shown below:

```
who %>%  
  gather(key, value, new_sp_m014:newrel_f65, na.rm = TRUE) %>%  
  mutate(key = stringr::str_replace(key, "newrel", "new_rel")) %>%  
  separate(key, c("new", "var", "sexage")) %>%  
  separate(sexage, c("sex", "age"), sep = 1) %>%  
  select(-new, -iso2, -iso3)
```

# Summary

- Data transformation with dplyr

| Function                 | Description                    | Equivalent SQL |
|--------------------------|--------------------------------|----------------|
| <code>select()</code>    | selecting columns              | SELECT         |
| <code>filter()</code>    | filtering rows / subsetting    | WHERE          |
| <code>group_by()</code>  | grouping data                  | GROUP BY       |
| <code>summarise()</code> | summarising / aggregating data | -              |
| <code>arrange()</code>   | sorting data                   | ORDER BY       |
| <code>join()</code>      | joining data tables            | JOIN           |
| <code>mutate()</code>    | creating new columns           | COLUMN ALIAS   |

# Summary

- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 745    | 18367071   |
| Afghanistan | 2000 | 2666   | 20595360   |
| Brazil      | 1999 | 31737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 213766 | 128042583  |

variables

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 745    | 18367071   |
| Afghanistan | 2000 | 2666   | 20595360   |
| Brazil      | 1999 | 31737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 213766 | 128042583  |

observations

| country     | year | cases  | population |
|-------------|------|--------|------------|
| Afghanistan | 1999 | 745    | 18367071   |
| Afghanistan | 2000 | 2666   | 20595360   |
| Brazil      | 1999 | 31737  | 172006362  |
| Brazil      | 2000 | 80488  | 174504898  |
| China       | 1999 | 212258 | 1272915272 |
| China       | 2000 | 213766 | 128042583  |

values

# Importing data

readr package

# Tabular data

- to import tabular data, use functions from **readr** package (**tidyverse**)
  - It is much faster than base R function such as `read.csv()`
  - It loads as `tibble` instead of R's traditional `data.frame`

| Function                  | Description  |
|---------------------------|--|
| <code>read_csv()</code>   | comma separated (csv) files                              |
| <code>read_tsv()</code>   | tab separated files                                      |
| <code>read_delim()</code> | general delimited files                                  |
| <code>read_fwf()</code>   | fixed width files  |
| <code>read_table()</code> | tabular files where columns are separated by white-space |

# example

```
car_data <- read_csv(file = "data/mtcars.csv")
```

```
## Parsed with column specification:  
## cols(  
##   mpg = col_double(),  
##   cyl = col_integer(),  
##   disp = col_double(),  
##   hp = col_integer(),  
##   drat = col_double(),  
##   wt = col_double(),  
##   qsec = col_double(),  
##   vs = col_integer(),  
##   am = col_integer(),  
##   gear = col_integer(),  
##   carb = col_integer()  
## )
```

# example

```
car_cata <- read_csv(file = "data/mtcars.csv", col_types =  
  cols(  
    mpg = col_double(),  
    cyl = col_integer(),  
    disp = col_double(),  
    hp = col_integer(),  
    drat = col_double(),  
    vs = col_integer(),  
    wt = col_double(),  
    qsec = col_double(),  
    am = col_integer(),  
    gear = col_integer(),  
    carb = col_integer())  
)  
)
```

# Excel file

- to load .xls or .xlsx files, use **read\_excel()** function in the **readxl** package.
  - You can specify the sheet of of an Excel spreadsheet.
  - Read its documentation for more details.
  - Installed as a part of **tidyverse**, but not a core **tidyverse** package

```
# Need to Load the package explicitly
library(readxl)

# Loading the first sheet of an Excel file
data <- read_excel("data/datasets.xlsx", 1)
```

# Exporting images

# ggsave

- call `ggsave()` after calling `ggplot()` to save the last ggplot object:

```
ggplot(mtcars, mapping = aes(x = wt, y = mpg))+
  geom_point(shape = 1)
# save PDF
ggsave("myplot.pdf", width = 8, height = 8, units = "cm", useDingbats = F)
# save PNG
ggsave("myplot.png", width = 8, height = 8, units = "cm")
```

# Labels

## ggplot2

# Labels

|                          |  |
|--------------------------|--|
| <b>ggttitle("Title")</b> | Add a main title above the plot  |
| <b>xlab("x-axis")</b>    | Change the label on the x-axis   |
| <b>ylab("y-axis")</b>    | Change the label on the y-axis   |
| <b>labs(...)</b>         | title = "main title above the plot", subtitle = "subtitle below title", caption = "caption below plot" |
| <b>annotate()</b>        | annotate(geom="text", x = 10, y = 10, label="A")   |

# Scales

## ggplot2

# Scales

- Scales map data values to the visual values of an aesthetic.
- To change a mapping, add a new scale:

```
aesthetic to adjust = mpg) + types of scale  
geom_bar( mapping = aes(x = drv, fill = drv)) +  
scale_fill_manual(  
  values = c("skyblue", "royalblue", "navy"), scale-specific arguments  
  limits = c("4", "f", "r"), range of values to include in mapping  
  breaks = c("4", "f", "r"), breaks to use in legend/axis  
  name = "drv",  
  labels = c("4wd", "front-wheel drive", "rear wheel drive")  
) labels to use in legend/axis
```

# General purpose scales

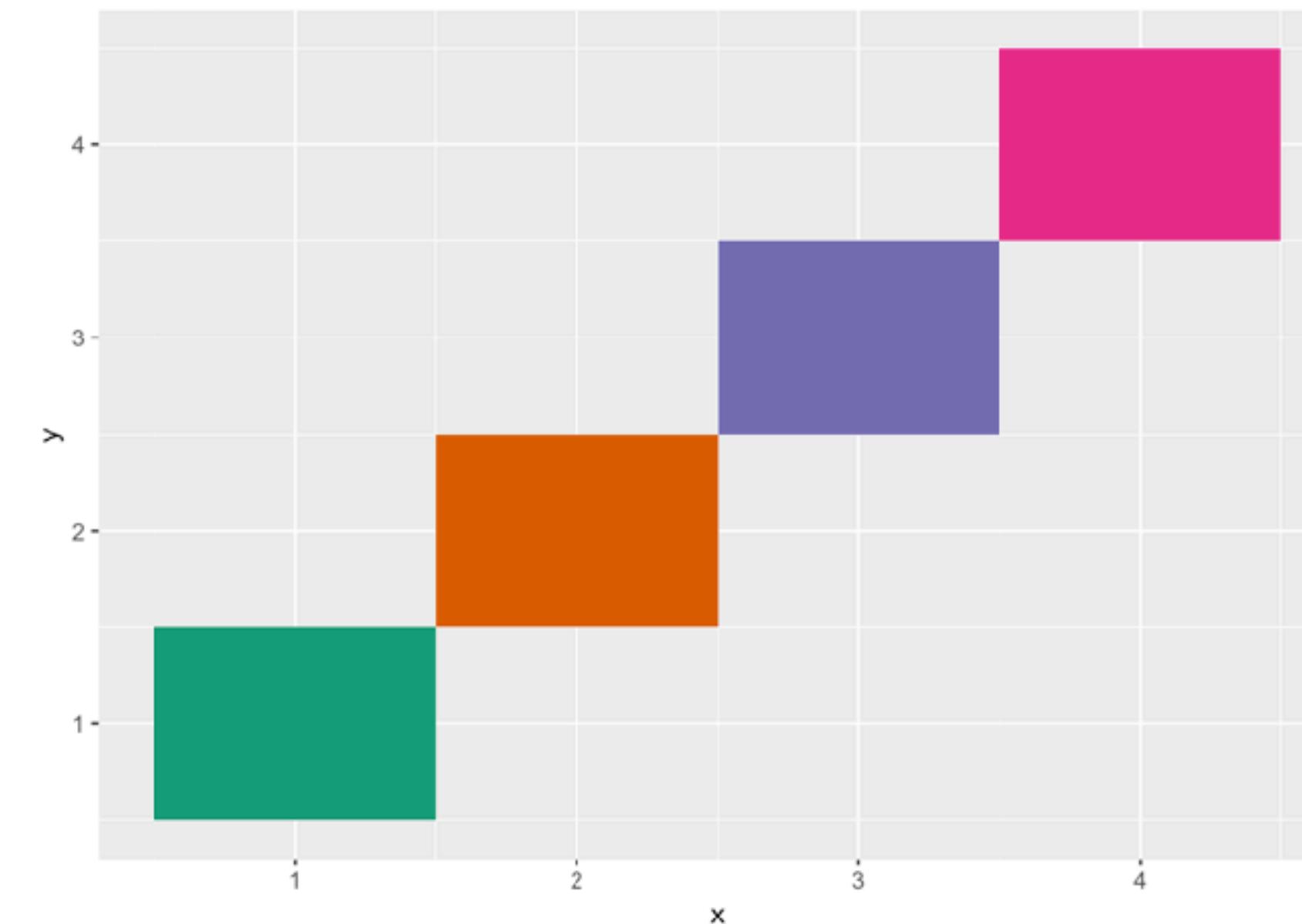
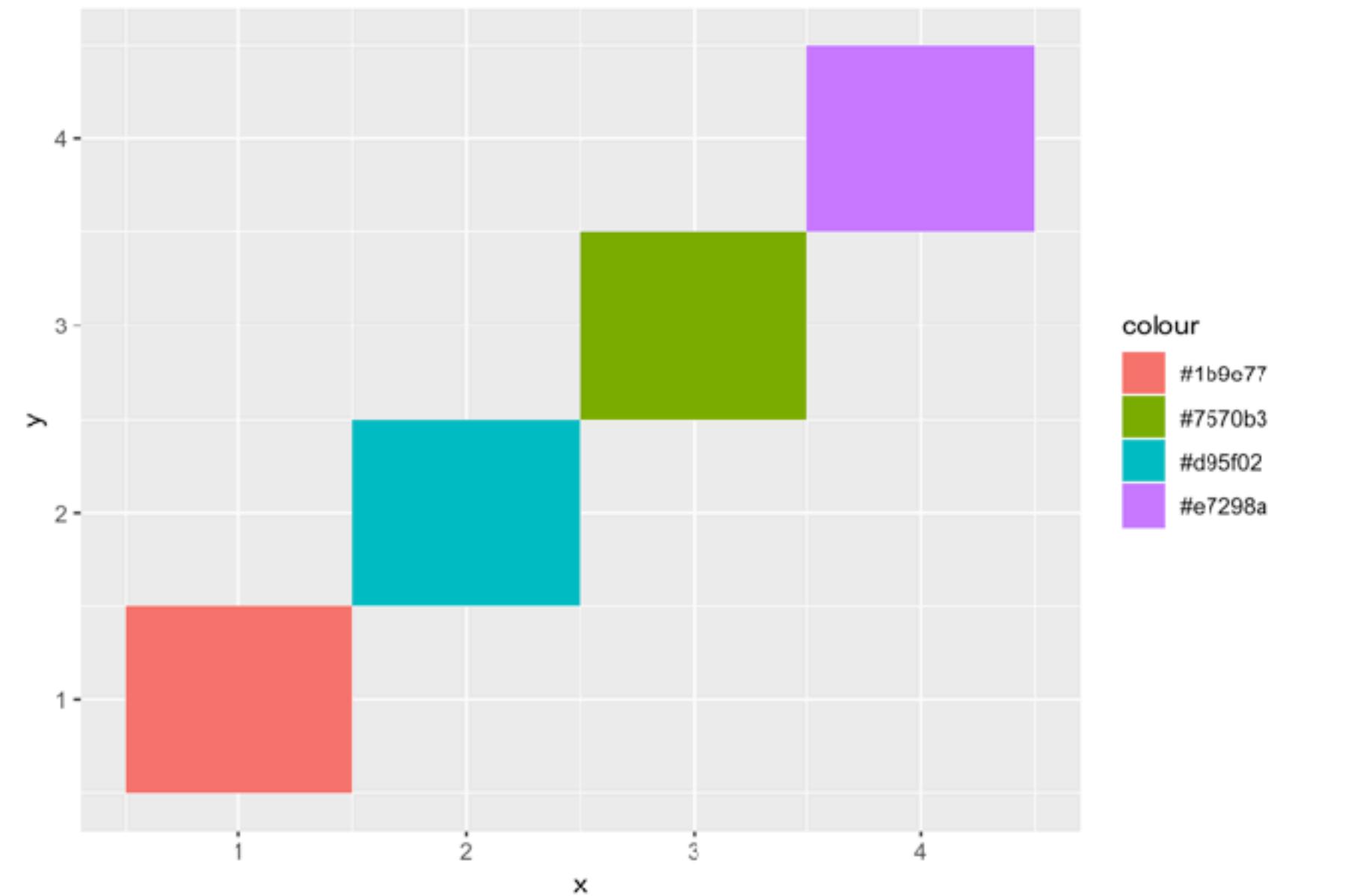
|   |  |
|---|--|
| <code>scale_*_continuous()</code>   | map continuous values to visual scale                |
| <code>scale_*_discrete()</code>   | map discrete values to visual scale                  |
| <code>scale_*_identity()</code>   | use data values as visual ones                       |
| <code>scale_*_manual(values = c())</code>                                     | map discrete values to manually chosen visual ones   |
| <code>scale_*_date(date_labels = "%m/%d",<br/>date_breaks = "2 weeks")</code> | treat data values as dates (class = Date)            |
| <code>scale_*datetime()</code>  | treat data x values as date times. (class = POSIXct) |

\* = `alpha`, `color`, `fill`, `linetype`, `shape`, `size`, `x`, `y`

# scale\_\*\*\_identity()

```
df <- data.frame(  
  x = 1:4,  
  y = 1:4,  
  colour = c('#1b9e77', '#d95f02', '#7570b3', '#e7298a')  
)  
  
ggplot(df, aes(x, y)) + geom_tile(aes(fill = colour))  
  
ggplot(df, aes(x, y)) + geom_tile(aes(fill = colour)) +  

```



# X and Y location scales

**scale\_x\_log10()**

plot x on log10 scale

**scale\_x\_reverse()**

reverse direction of x-axis

**scale\_x\_sqrt()**

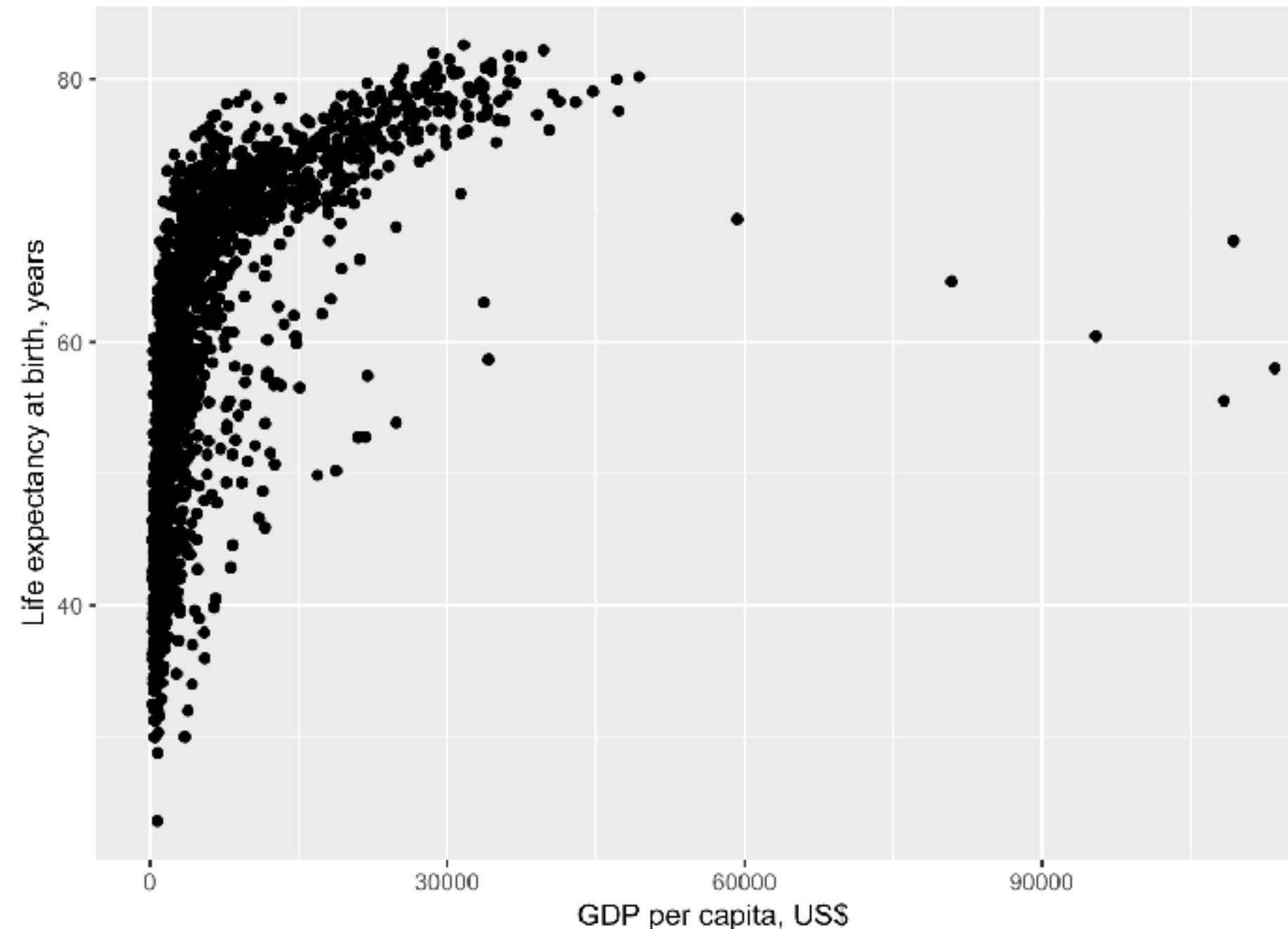
plot x on square root scale

# gapminder

```
# install.packages("gapminder")
library(gapminder)

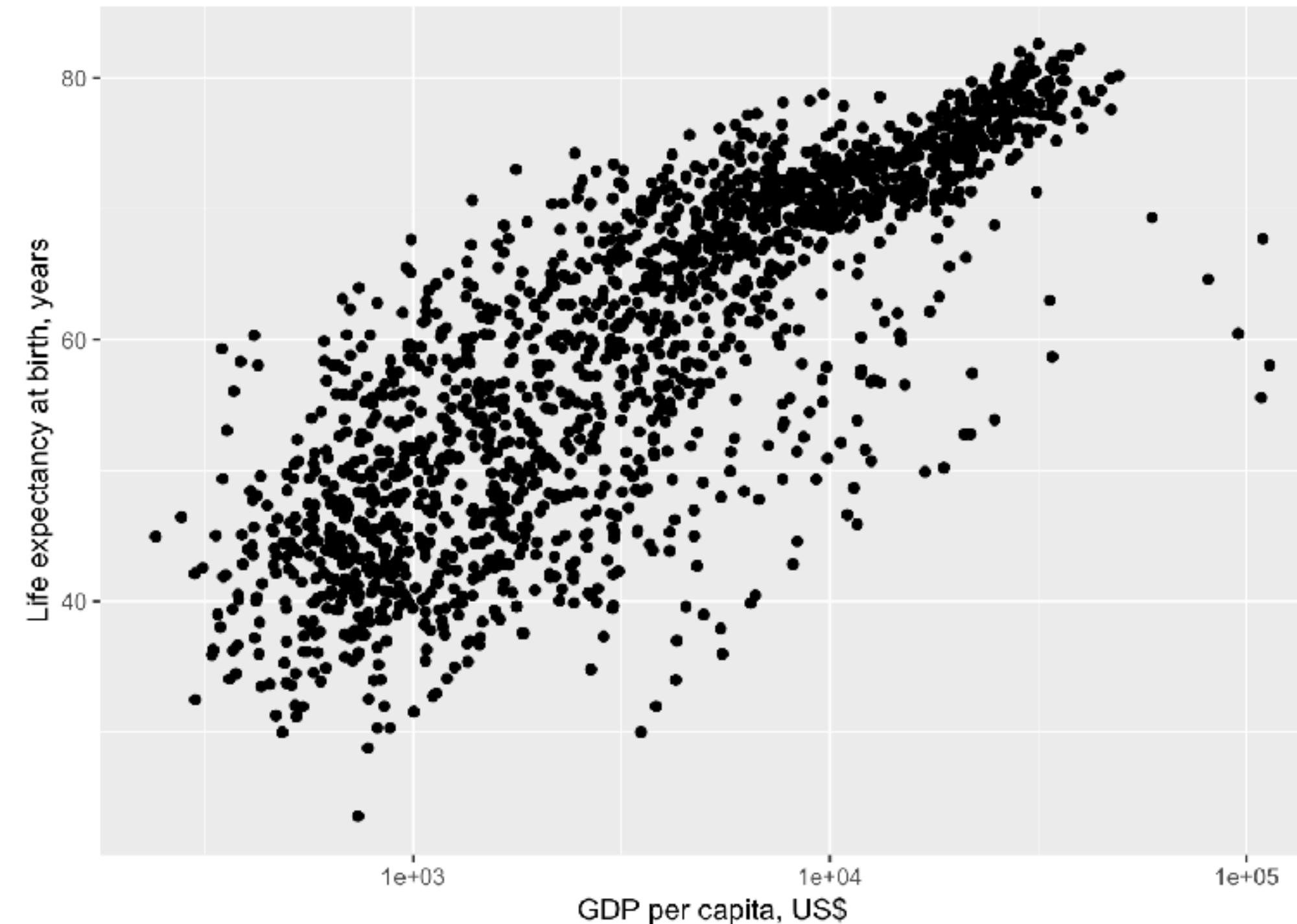
# Looking up ?gapminder to see the documentation

# scatter plot of GDP per capita and life expectancy
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +
  geom_point() +
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



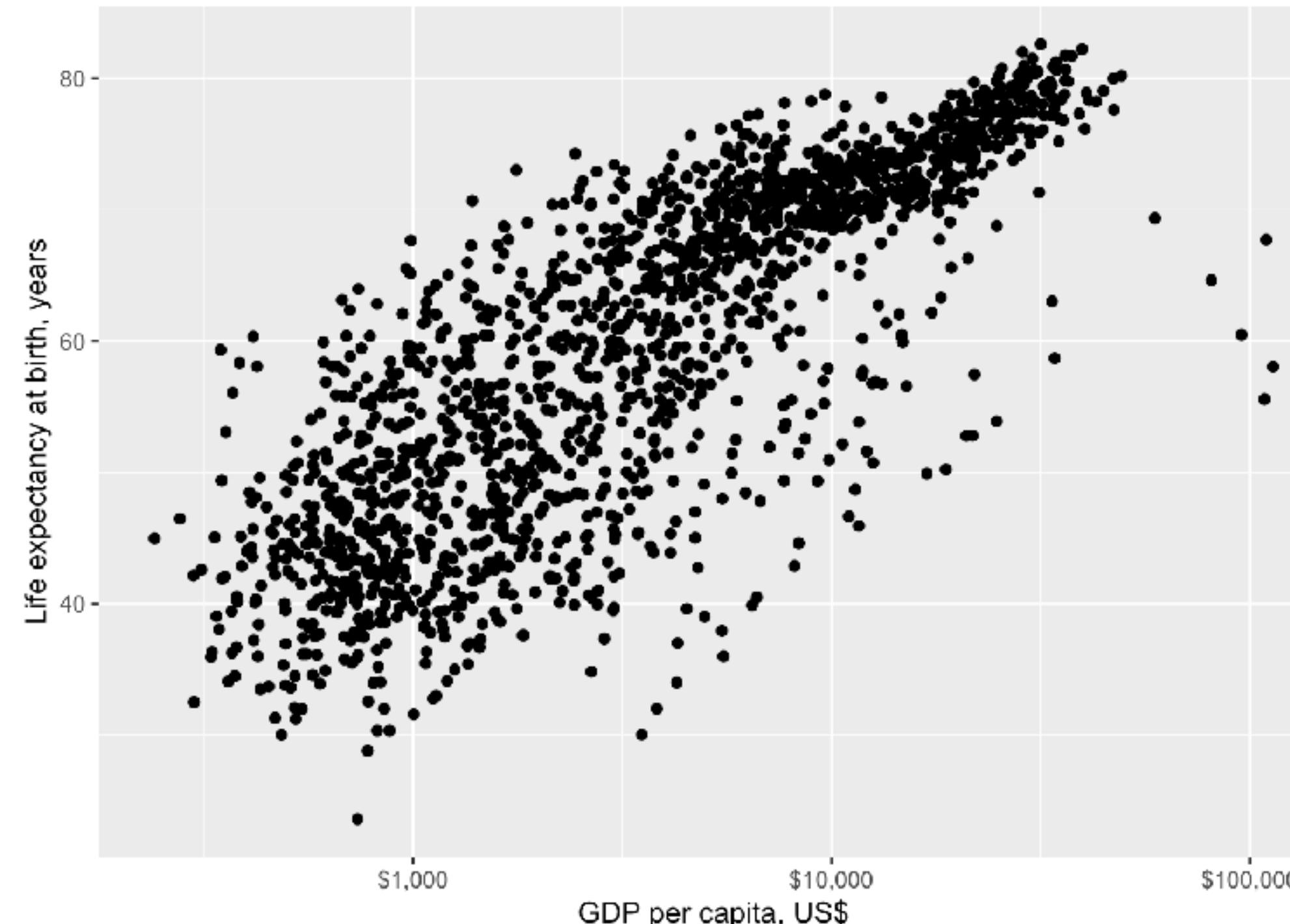
# gapminder

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  scale_x_log10() +  
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



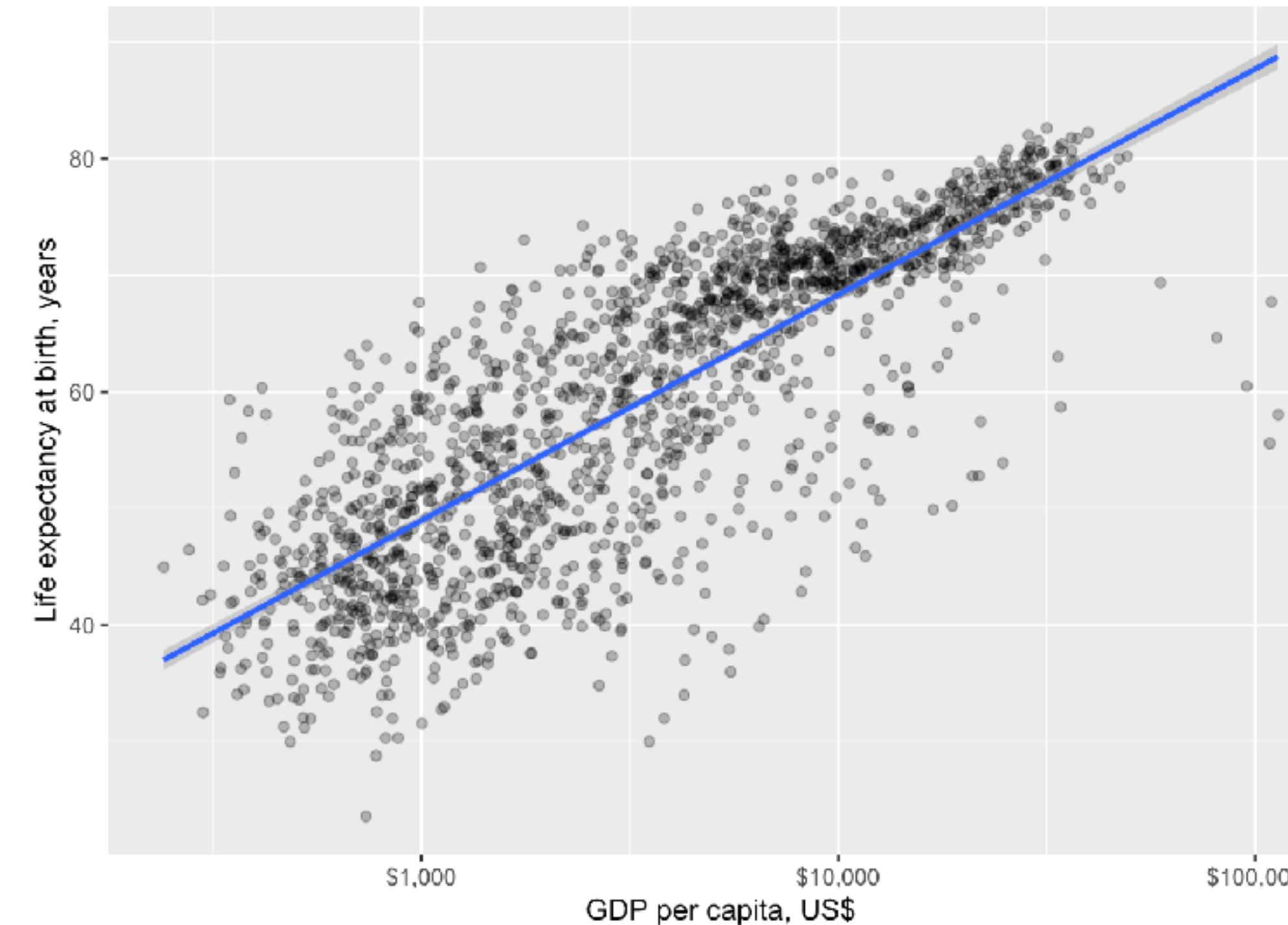
# gapminder

```
# handy function scale::dollar, scale::comma by passing a function  
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  scale_x_log10(labels = scales::dollar) +  
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



# gapminder

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +  
  geom_point(alpha = 0.3) +  
  geom_smooth(method = "gam") +  
  scale_x_log10(labels = scales::dollar) +  
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



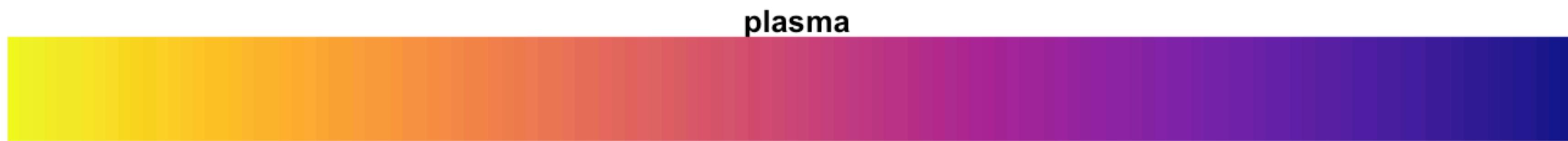
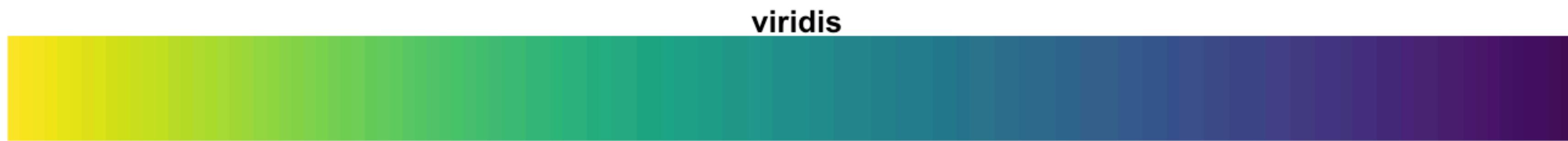
# Colour scales

ggplot2

# Colour scale

- Is your data **discrete** or **continuous**?
- Colourblind-friendly palette
  - Package: viridis (comes with ggplot2 v3.0.0)
    - <https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html>
  - Package: RColorBrewer
    - `display.brewer.all()`
    - `display.brewer.pal(8, "Blues")`

# viridis

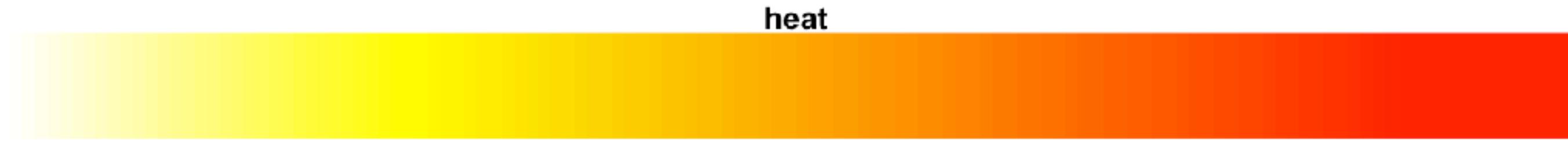


# Comparisons

*Base R: rainbow.colours*



*Base R: heat.colours*



*ggplot2 default*



*colorbrewer blues*



*colorbrewer  
yellow-green-blue*



*viridis*



*magma*



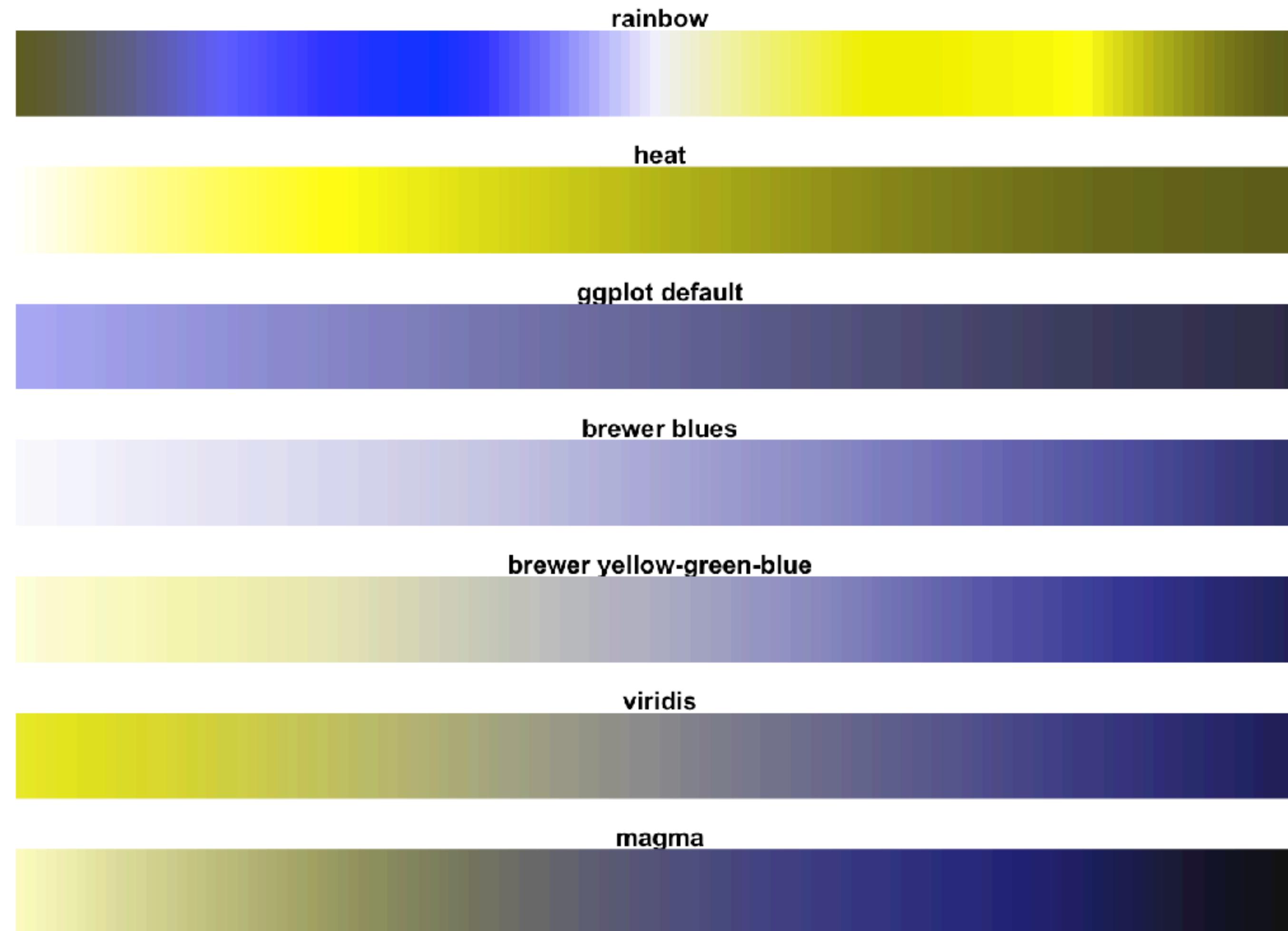
*Green-blind (Deutanopia)*

# Comparisons



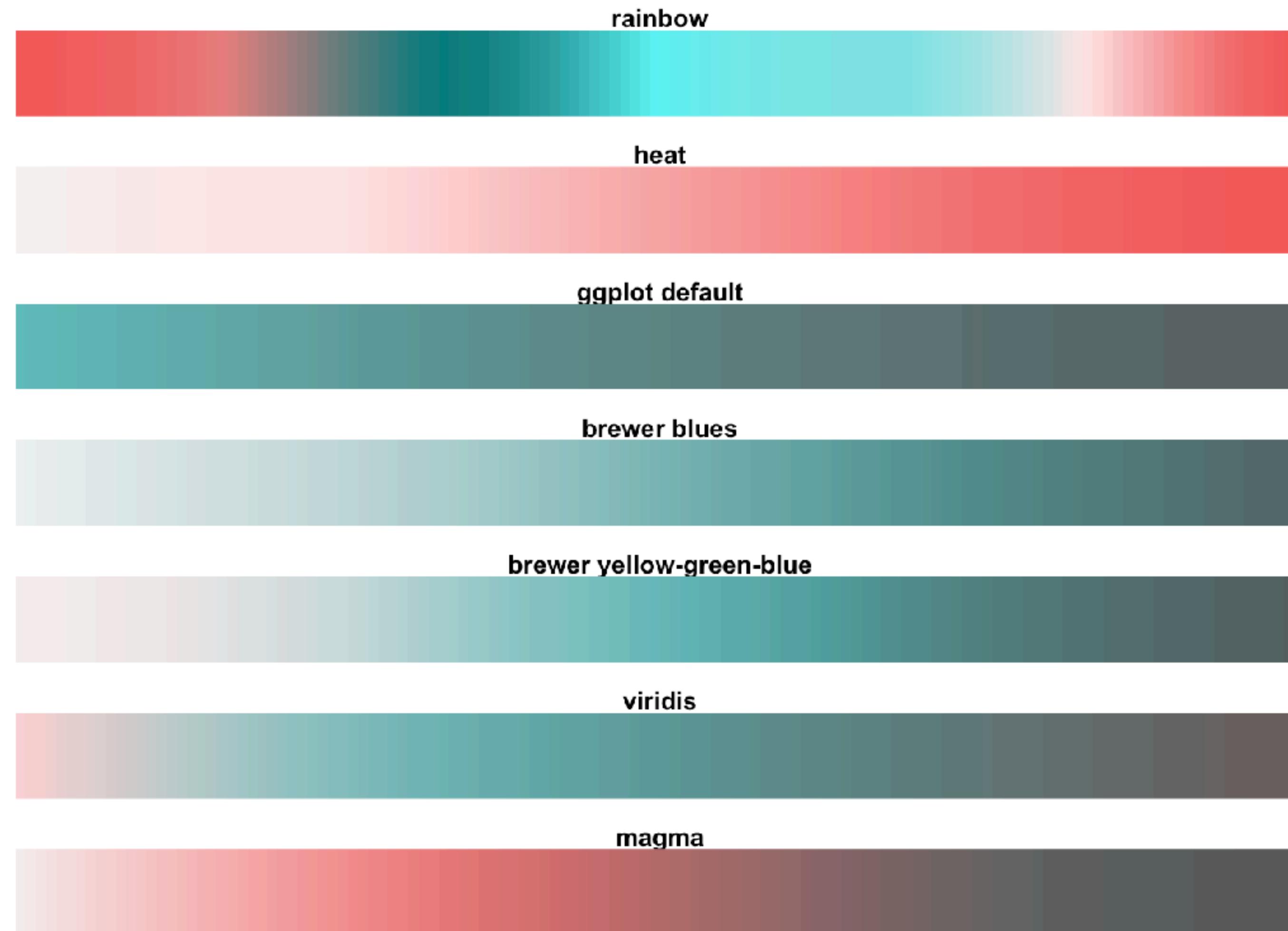
*Red-blind (Protanopia)*

# Comparisons



*Blue-blind (Tritanopia)*

# Comparisons



*Desaturated*

# Comparisons



# Fill & colour scale - Discrete

|                            |   |
|----------------------------|---|
| <b>scale_*_discrete()</b>  | Colours evenly spaced around the colour wheel (same as hue) |
| <b>scale_*_hue()</b>       | Colours evenly spread around the colour wheel               |
| <b>scale_*_grey()</b>      | Greyscale palette   |
| <b>scale_*_viridis_d()</b> | Viridis palettes  |
| <b>scale_*_brewer()</b>    | ColorBrewer palettes  |
| <b>scale_*_manual()</b>    | Manually specified colours                                  |

# ColorBrewer

display.brewer.all()



# Fill & colour scale - Continuous

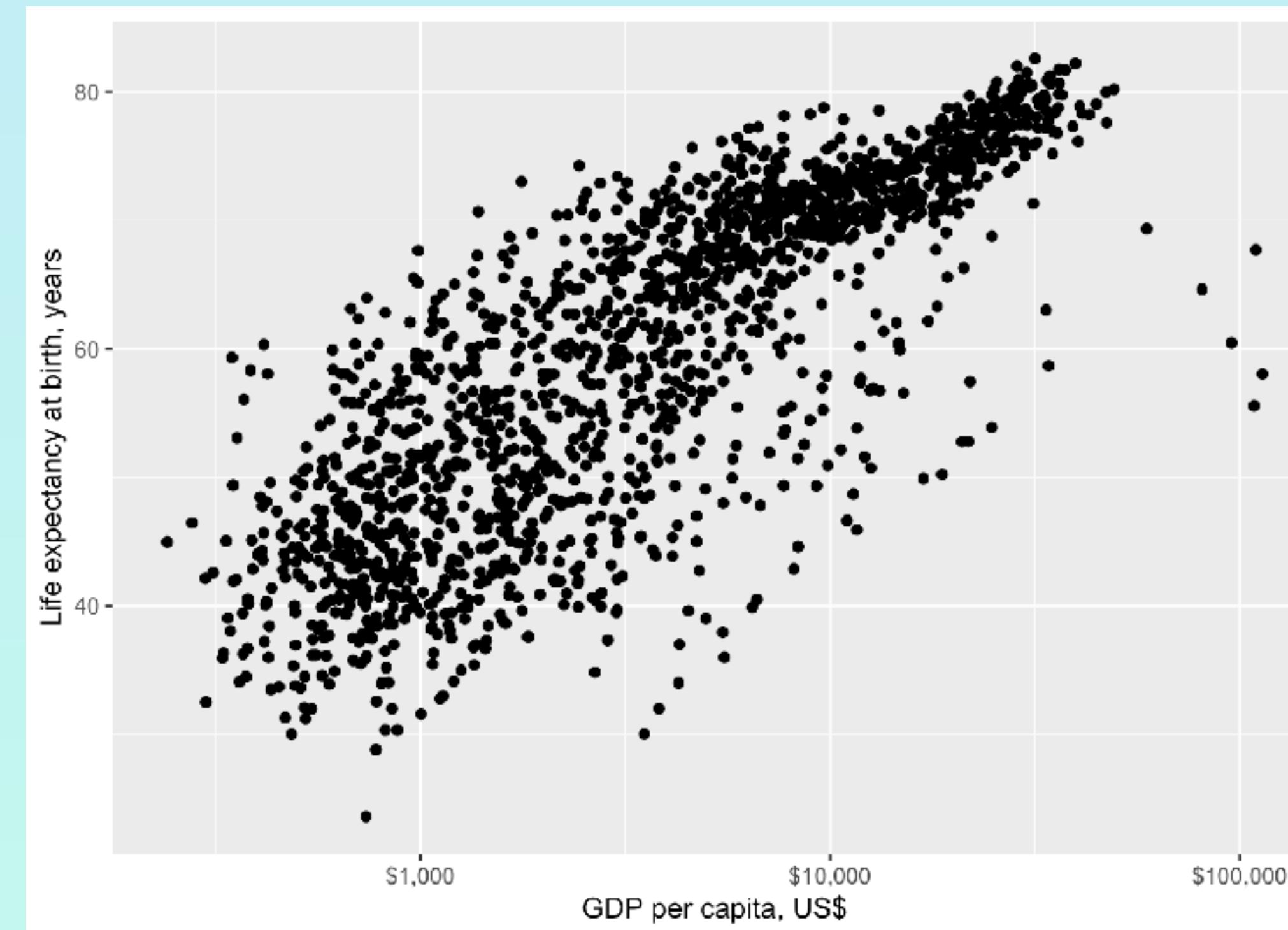
|                            |  |
|----------------------------|--|
| <b>scale_*_gradient()</b>  | Two-colour gradient  |
| <b>scale_*_gradient2()</b> | Gradient with a middle colour and two colours that diverge from it |
| <b>scale_*_gradientn()</b> | Gradient with n colour, equally spaced                             |
| <b>scale_*_viridis_c()</b> | Viridis palettes   |

- Do you really need a continuous scale?
  - Can you bin the data to use discrete scale?

# Your turn!

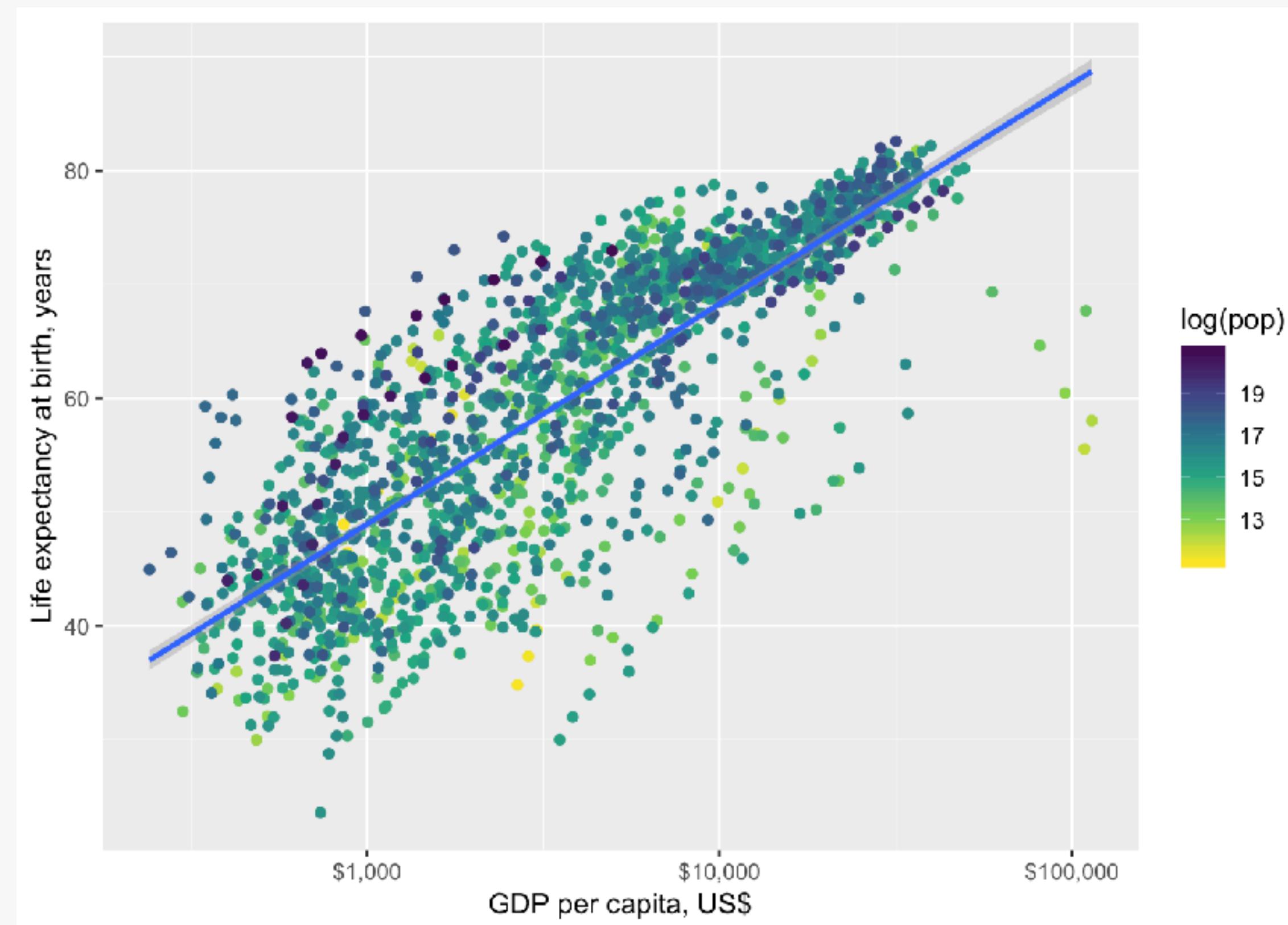
- 1) Modify the code to map pop variable (population) to colour aesthetics.
- 2) Did you map colour using a linear scale or log scale?

```
ggplot(data = gapminder, mapping = aes(x = gdpPercap, y = lifeExp)) +  
  geom_point() +  
  scale_x_log10(labels = scales::dollar) +  
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



# Solution

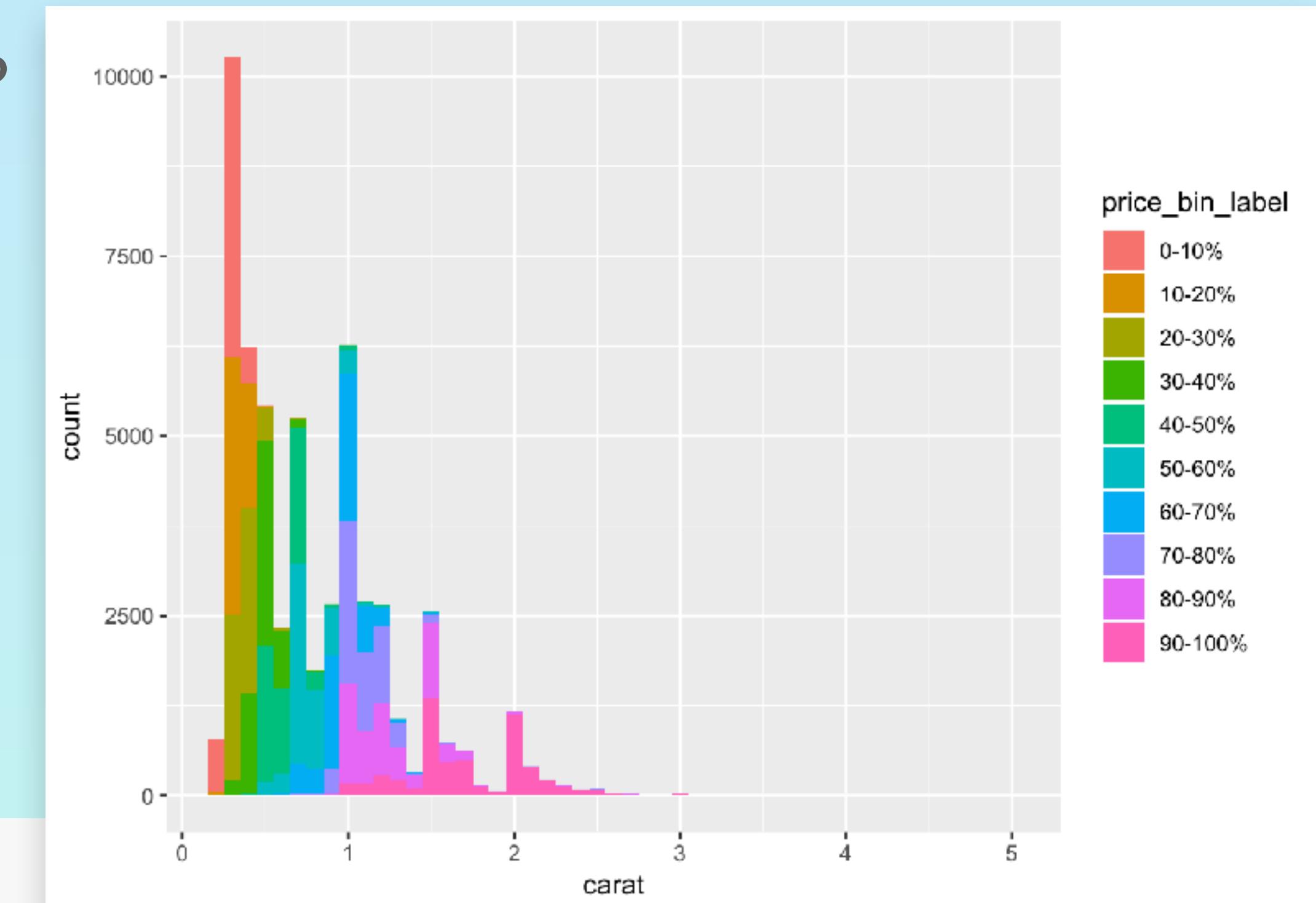
```
ggplot(data = gapminder %>% arrange(pop), mapping = aes(x = gdpPercap, y = lifeExp)) +  
  geom_point(mapping = aes(colour = log(pop))) +  
  geom_smooth(method = "gam") +  
  scale_x_log10(labels = scales::dollar) +  
  scale_color_viridis_c(direction = -1)+  
  labs(x = "GDP per capita, US$", y = "Life expectancy at birth, years" )
```



# Your turn!

- 1) Apply `scale_*_viridis_d()` to the following code:
- 2) Can you use other viridis colour palettes?

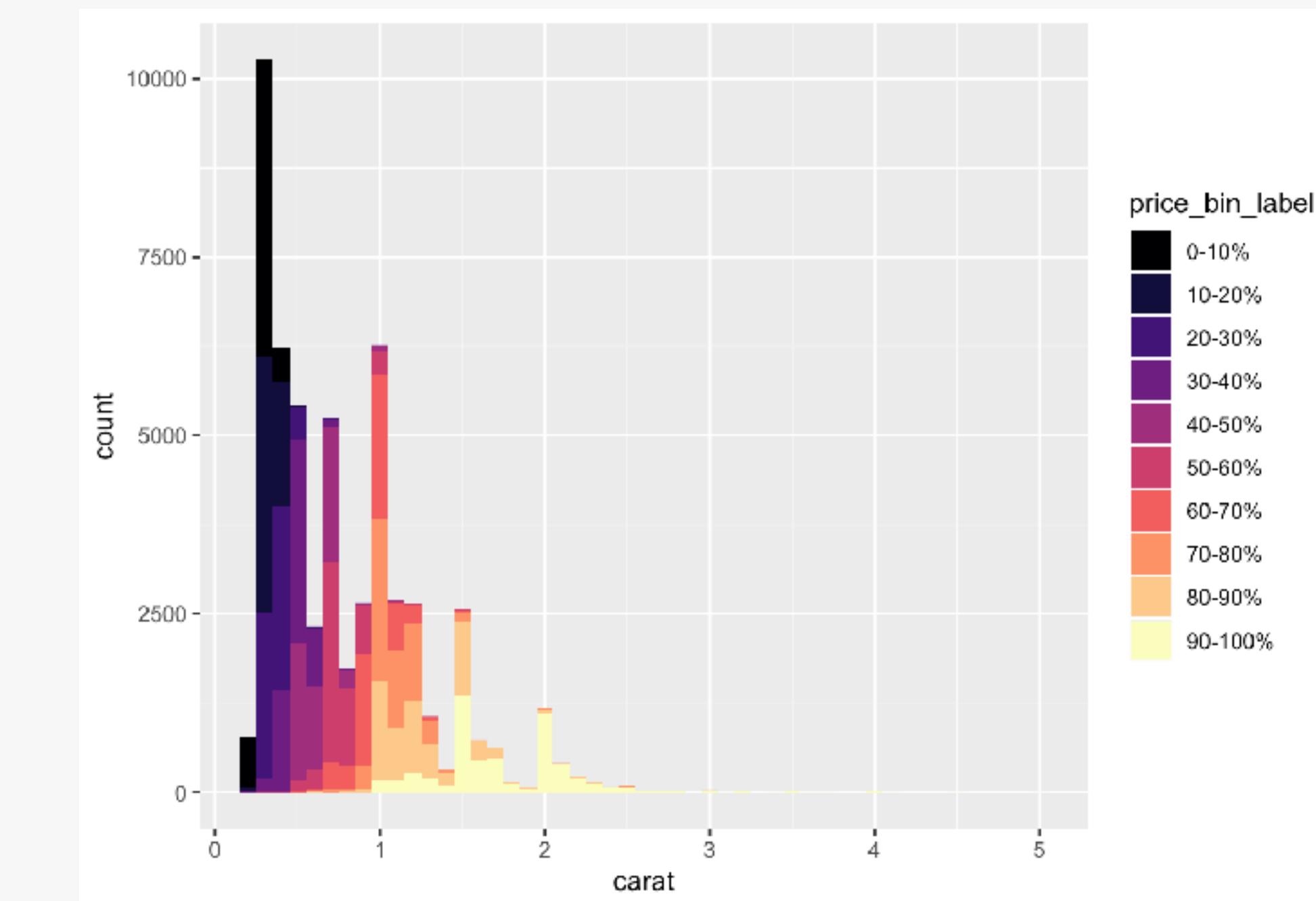
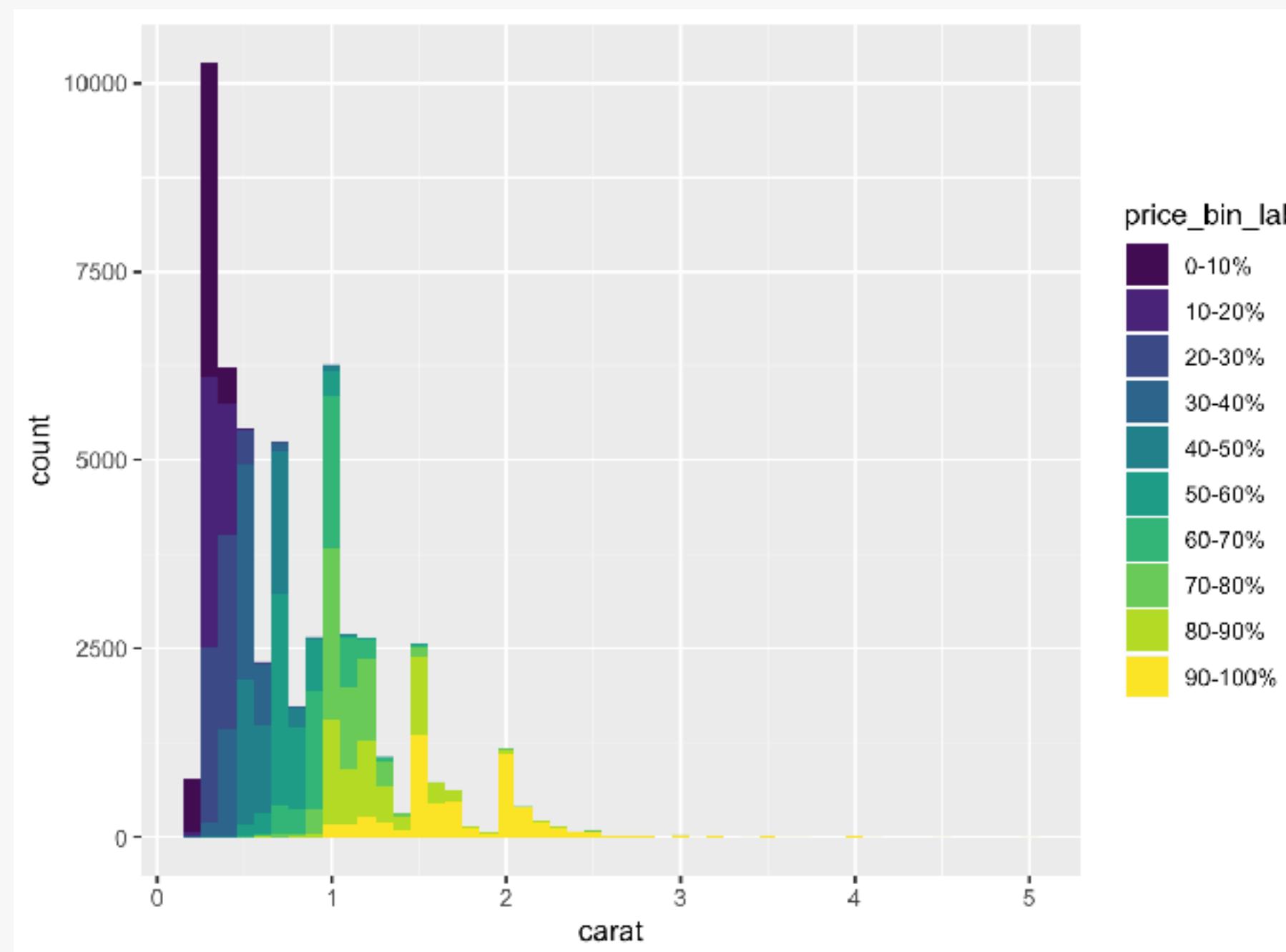
```
# create a new variable by binning by price
df <- diamonds %>%
  mutate(price_bin = ntile(price, 10),
        price_bin_label = factor(price_bin, levels = c(1:10),
                                  labels = c("0-10%", "10-20%", "20-30%", "30-40%", "40-50%",
                                            "50-60%", "60-70%", "70-80%", "80-90%", "90-100%")))
ggplot(data = df) +
  geom_histogram(mapping = aes(x = carat, fill = price_bin_label), binwidth = 0.1)
```



# Solution

```
ggplot(data = df) +  
  geom_histogram(mapping = aes(x = carat, fill = price_bin_label), binwidth = 0.1)+  
  scale_fill_viridis_d()
```

```
ggplot(data = df) +  
  geom_histogram(mapping = aes(x = carat, fill = price_bin_label), binwidth = 0.1)+  
  scale_fill_viridis_d(option="magma")
```



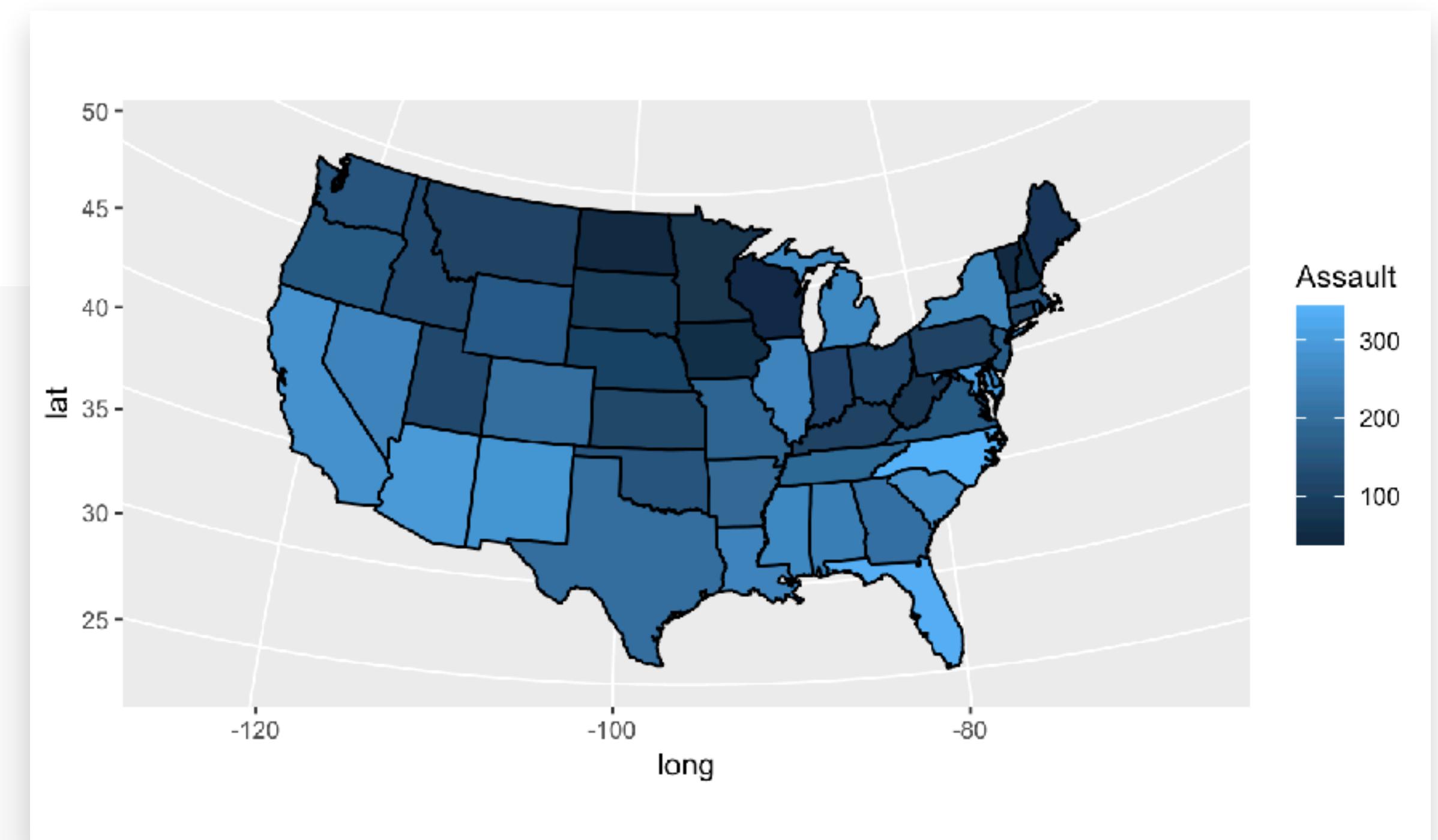
# Choropleth

# Choropleth Map

```
# using USArrests datasets
crimes <- data.frame(USArrests) %>%
  mutate(state = tolower(rownames(USArrests)))

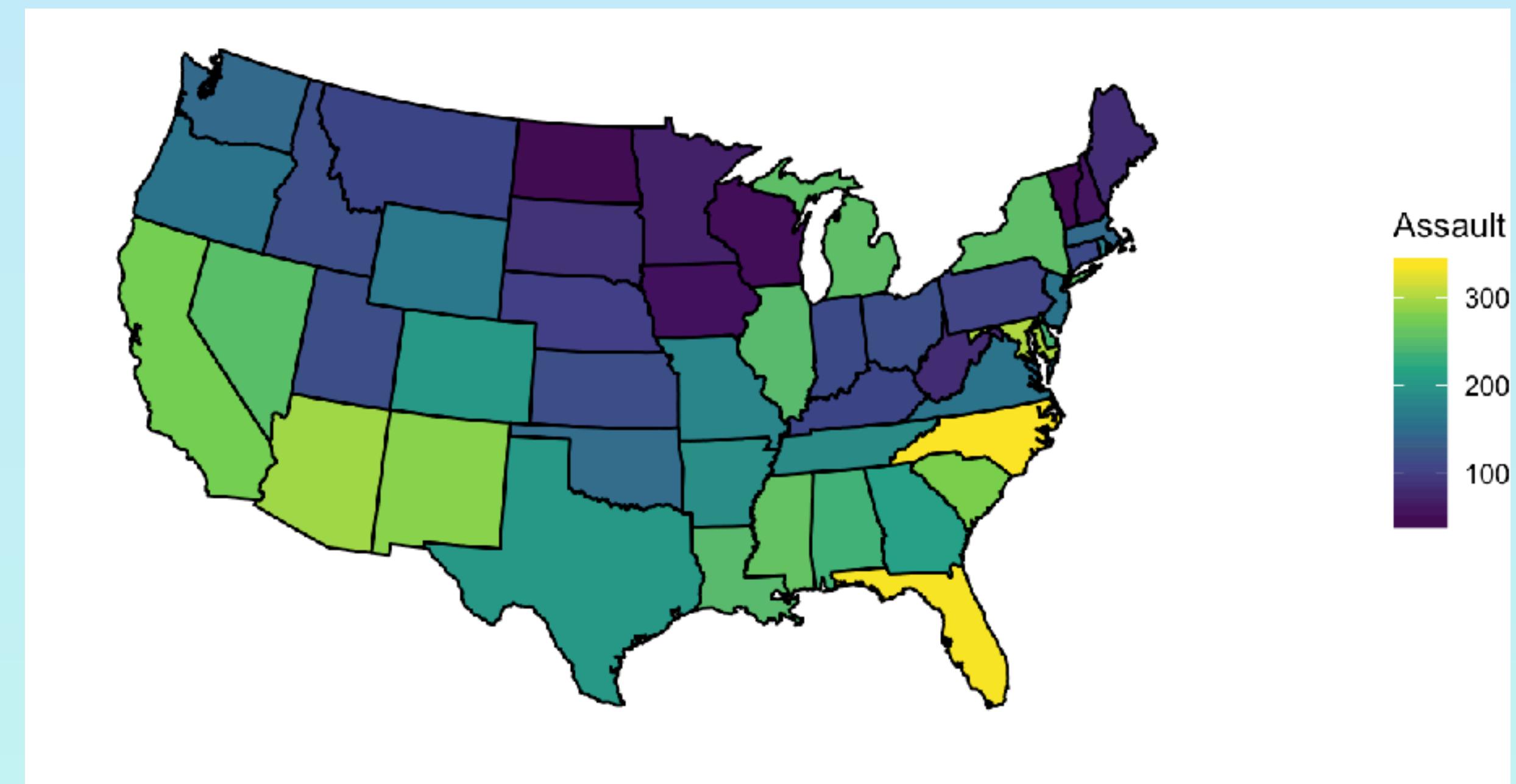
# using maps library
library(maps)
states_map <- map_data("state")
# merge datasets together
crime_map <- states_map %>%
  left_join(crimes, by = c("region" = "state")) %>%
  arrange(group, order)

ggplot(data = crime_map, aes(x=long, y=lat, group = group, fill = Assault))+
  geom_polygon(colour = "black")+
  coord_map("polyconic")
```

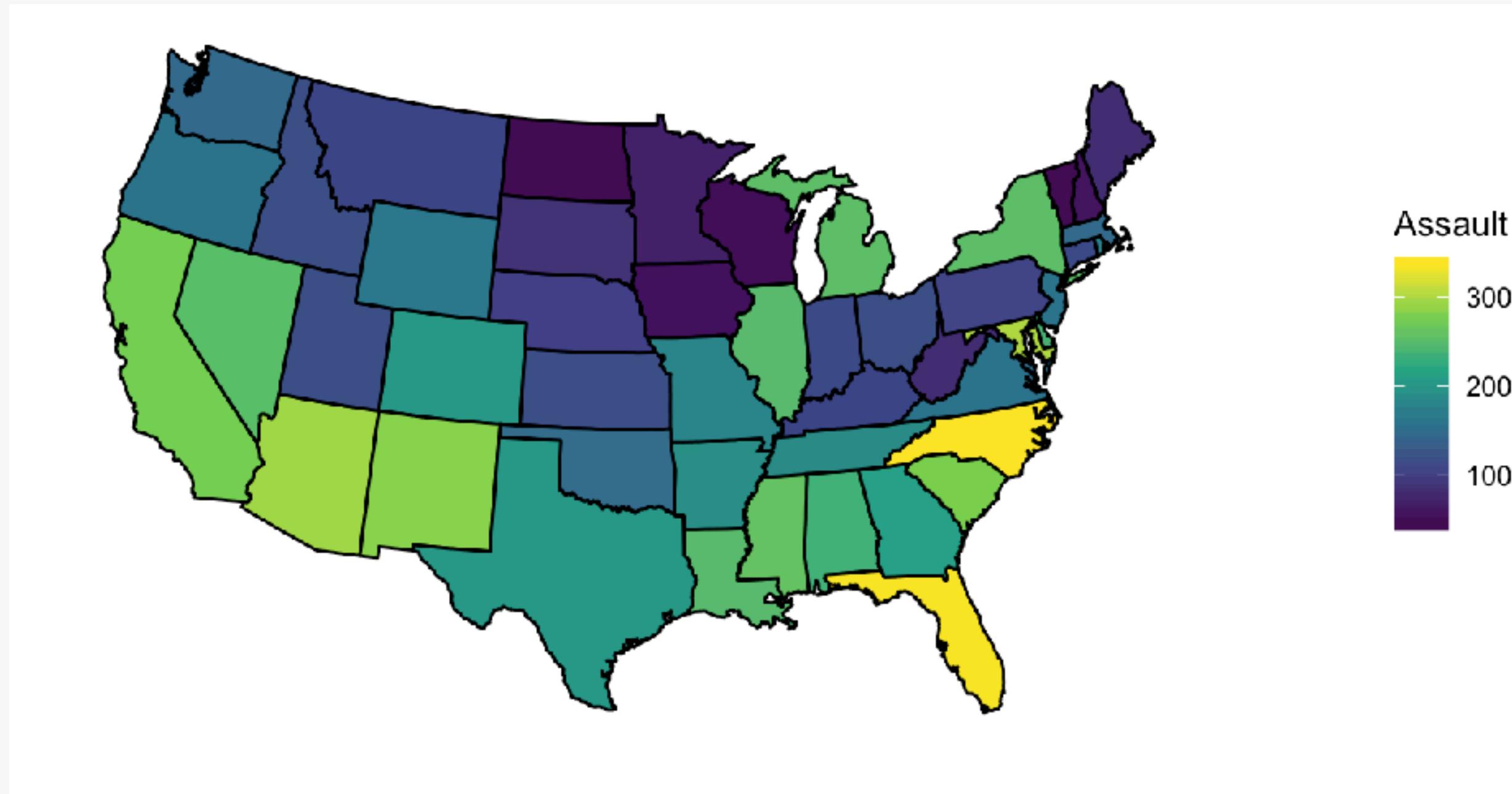


# Your turn!

1) Apply `scale_fill_viridis_c()` to produce a map like this:



# Solution



```
ggplot(data = crime_map, aes(x=long, y=lat, group = group, fill = Assault))+  
  geom_polygon(colour = "black") +  
  coord_map("polyconic") +  
  scale_fill_viridis_c() +  
  theme_void()
```

# Map of EU countries

```
# Map of Europe
# Get map data for world
world_map <- map_data("world")

# Check $regions
sort(unique(world_map$region))

## [1] "Afghanistan"
## [2] "Albania"
## [3] "Algeria"
## [4] "American Samoa"
## [5] "Andorra"
## [6] "Angola"
## [7] "Anguilla"
## [8] "Antarctica"
## [9] "Antigua"
## [10] "Argentina"
```

# Map of EU countries

```
# A vector of countries
europeanUnion <- c("Austria", "Belgium", "Bulgaria", "Croatia", "Cyprus",
  "Czech Republic", "Denmark", "Estonia", "Finland", "France",
  "Germany", "Greece", "Hungary", "Ireland", "Italy", "Latvia",
  "Lithuania", "Luxembourg", "Malta", "Netherlands", "Poland",
  "Portugal", "Romania", "Slovakia", "Slovenia", "Spain",
  "Sweden", "United Kingdom")

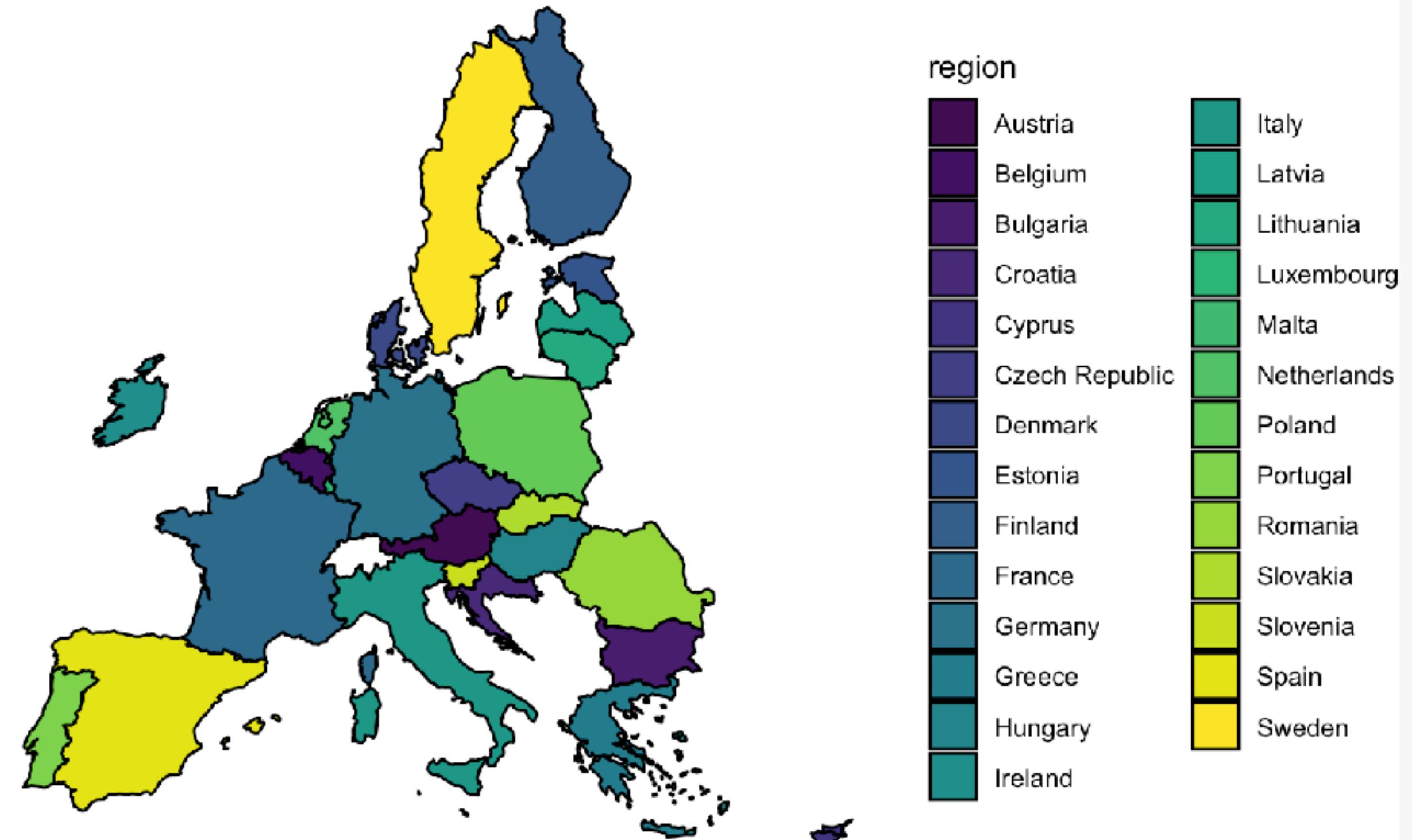
# get the map data
eu_map <- map_data("world", region = europeanUnion)
```

# Your turn!

- 1) Create a map of EU countries with `eu_map` data.

# Solution

```
ggplot(eu_map, mapping = aes(x = long, y = lat, group = group, fill = region)) +  
  geom_polygon(colour = "black") +  
  scale_fill_viridis_d() +  
  coord_map("sinusoidal") +  
  theme_void()
```



# Network Visualisation

# R libraries

- **igraph** : perhaps most commonly used, high level of customisation
  - Tutorial by Katya Ognyanova
- ggnets2: network visualisation with ggplot2
- networkD3: D3 JavaScript Network graphs from R
- visNetwork: uses vis.js javascript library

# Exploratory data analysis

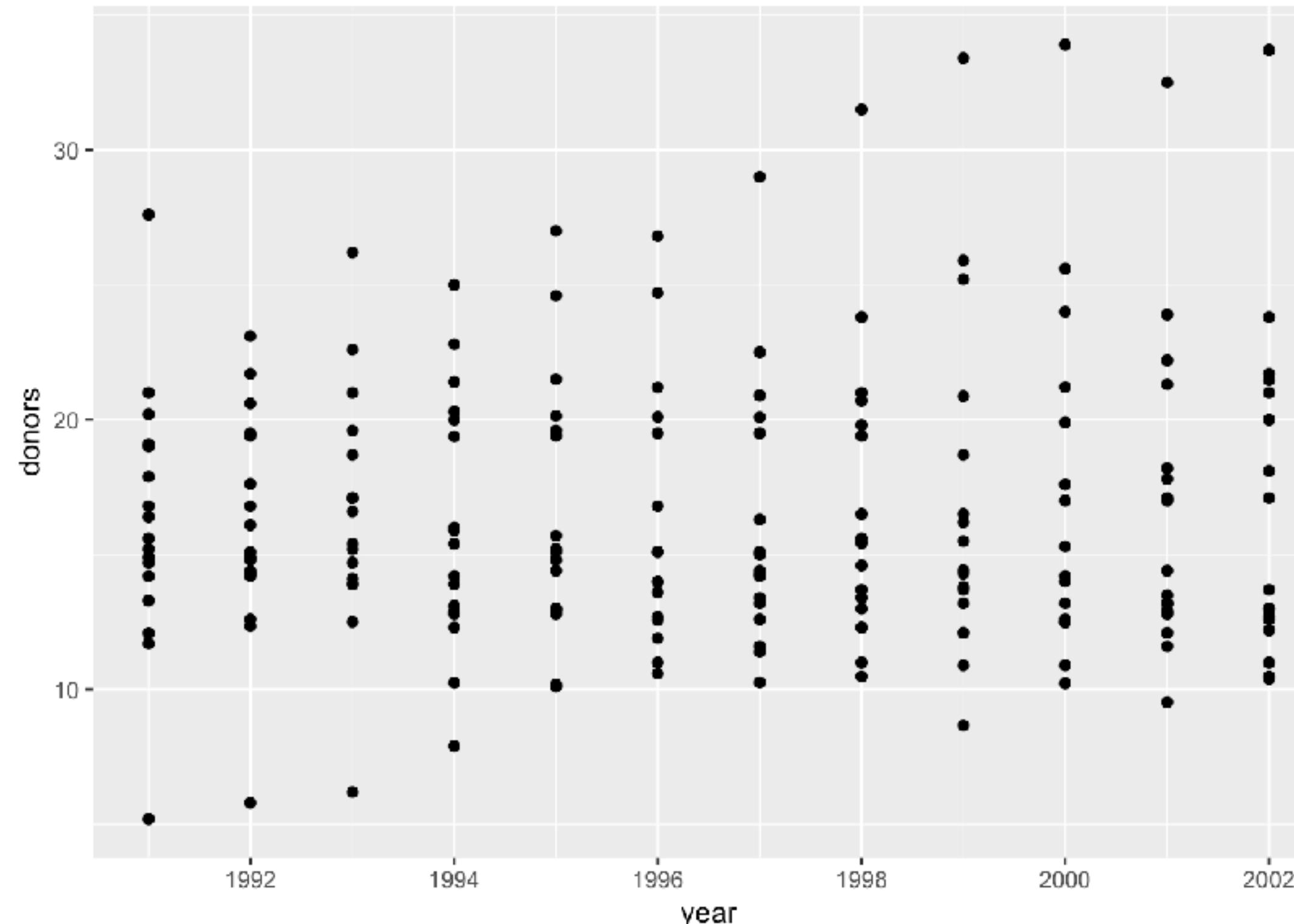
Organ data

# Organ data

- install a package called `socviz`
- `socviz::organdata`
  - Organ donation data of 17 countries between 1991 and 2002
- check the data documentation
  - `?organdata`

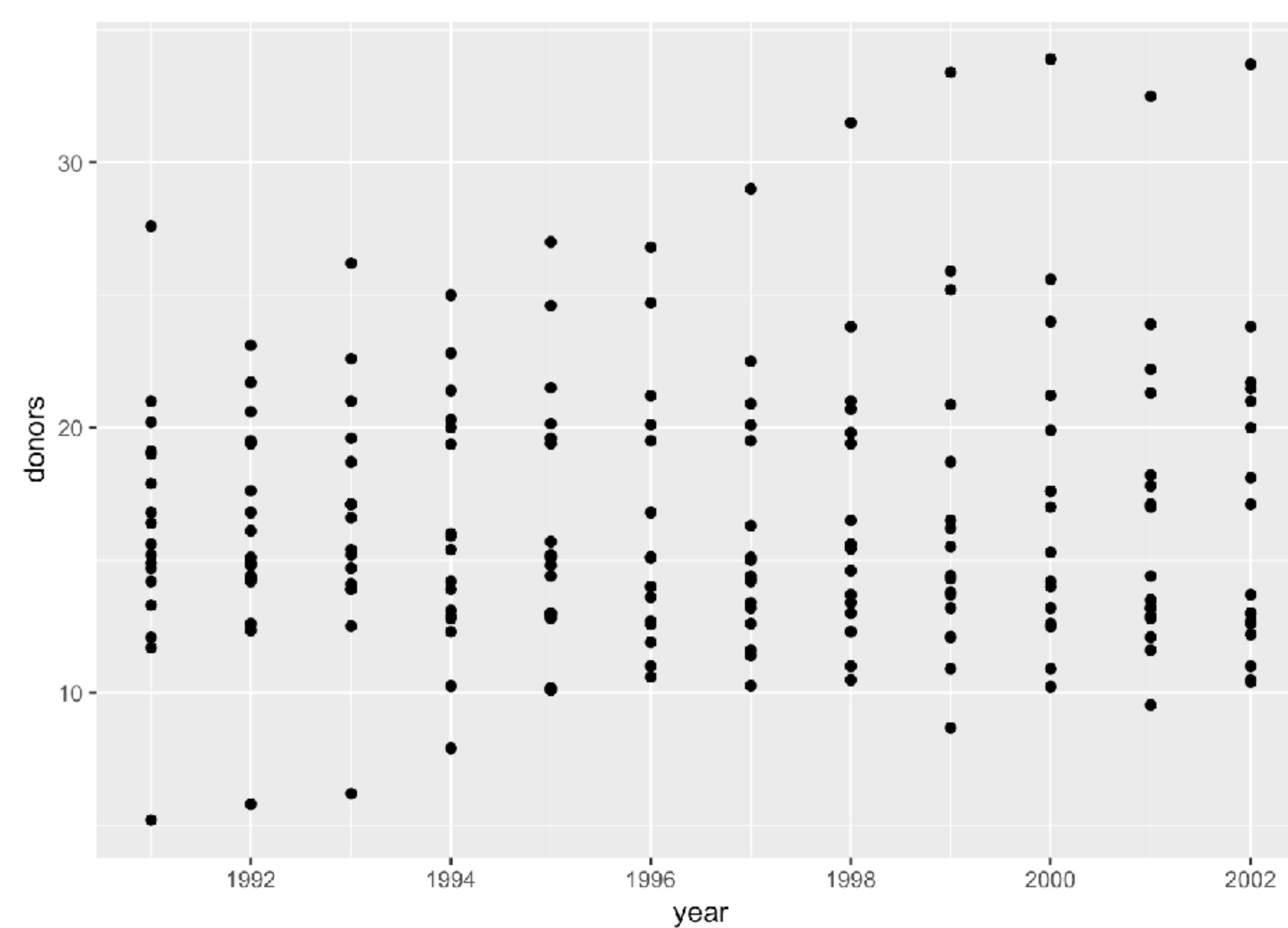
# Exploration

```
ggplot(data = organdata, mapping = aes(x = year, y = donors)) +  
  geom_point()  
  
## Warning: Removed 34 rows containing missing values (geom_point).
```



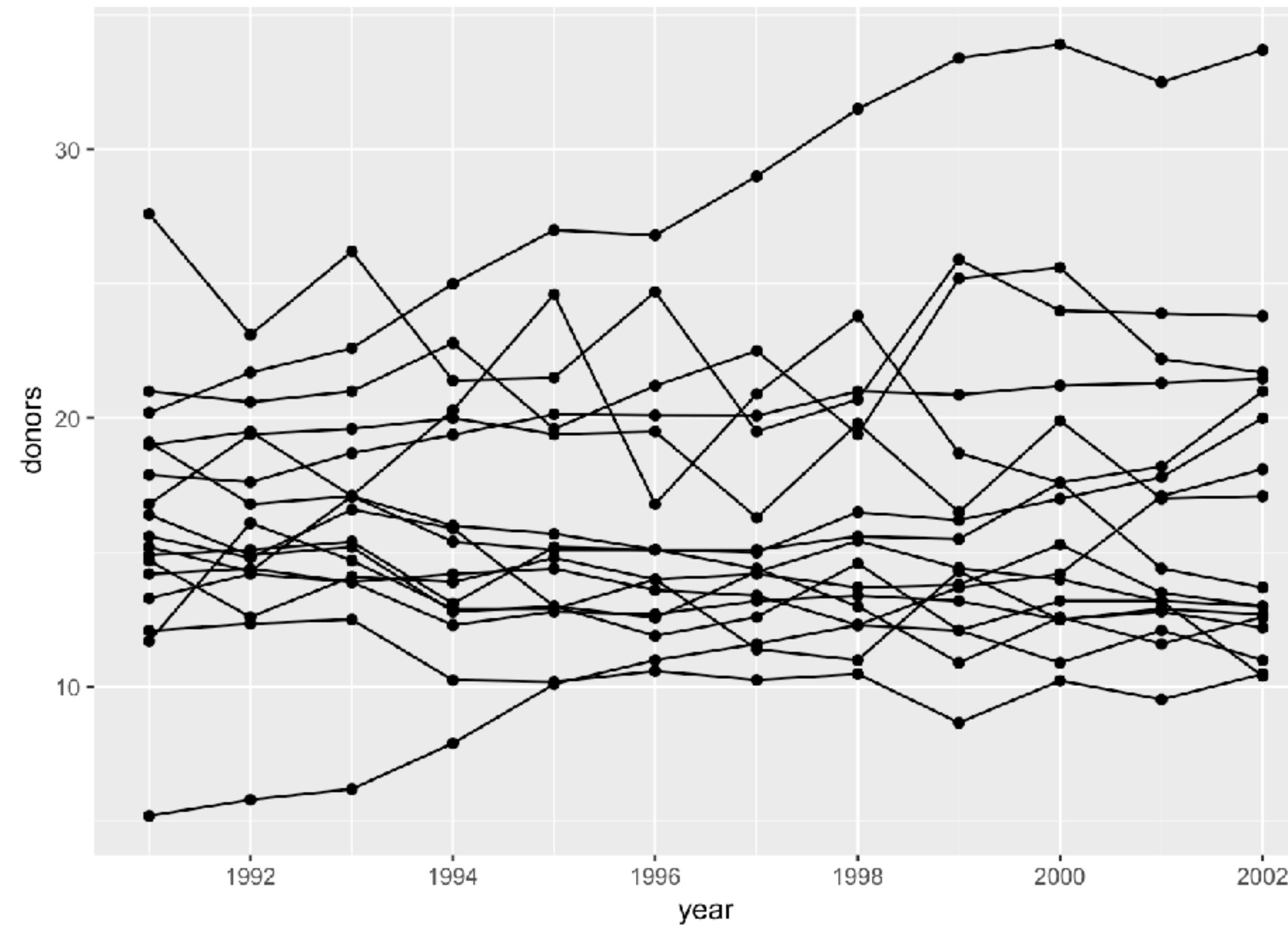
# Exploration

```
ggplot(data = organdata, mapping = aes(x = year, y = donors)) +  
  geom_point(na.rm = T)
```



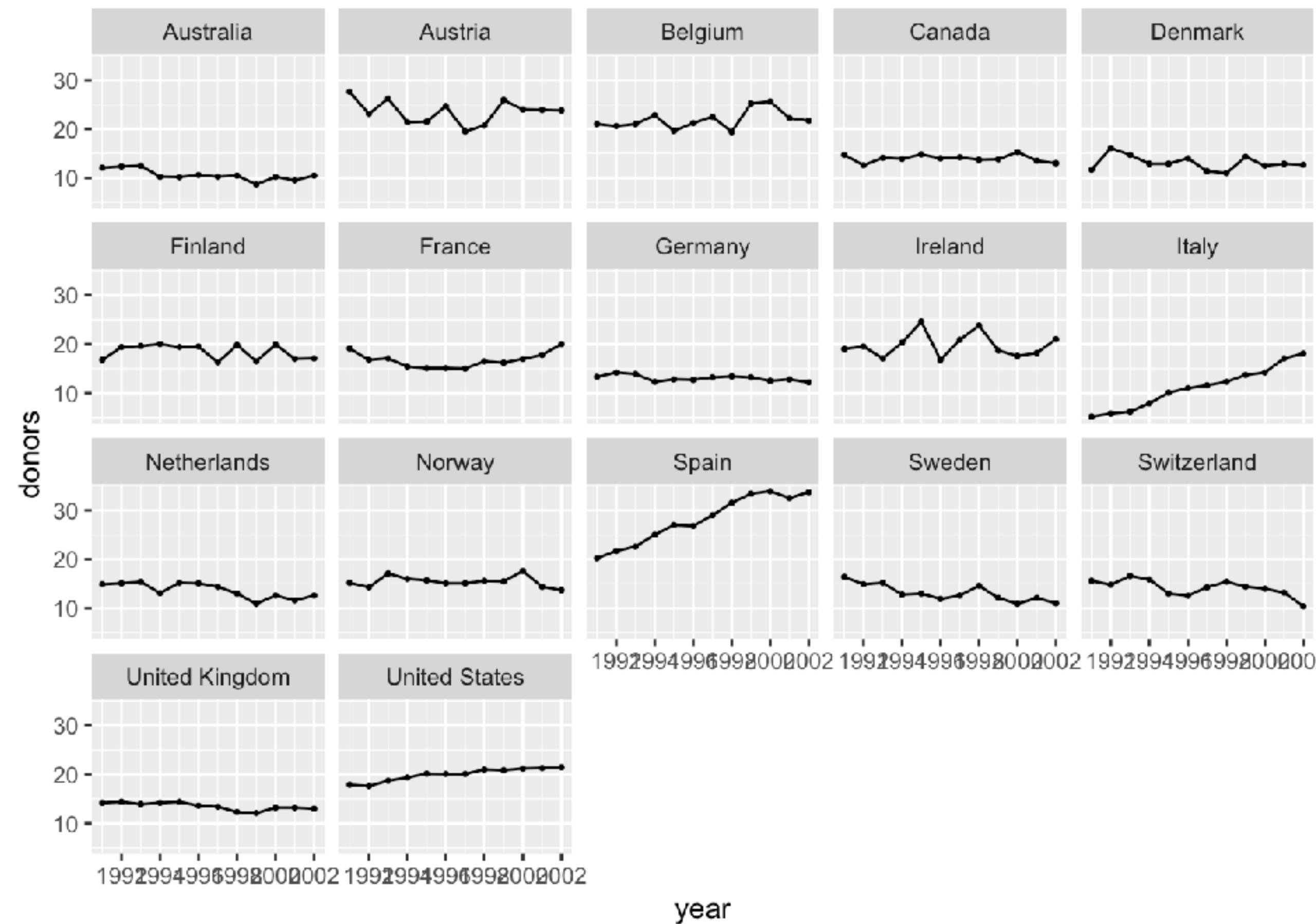
# Exploration

```
ggplot(data = organdata, mapping = aes(x = year, y = donors)) +  
  geom_point(na.rm = T) +  
  geom_line(aes(group = country), na.rm = T)
```



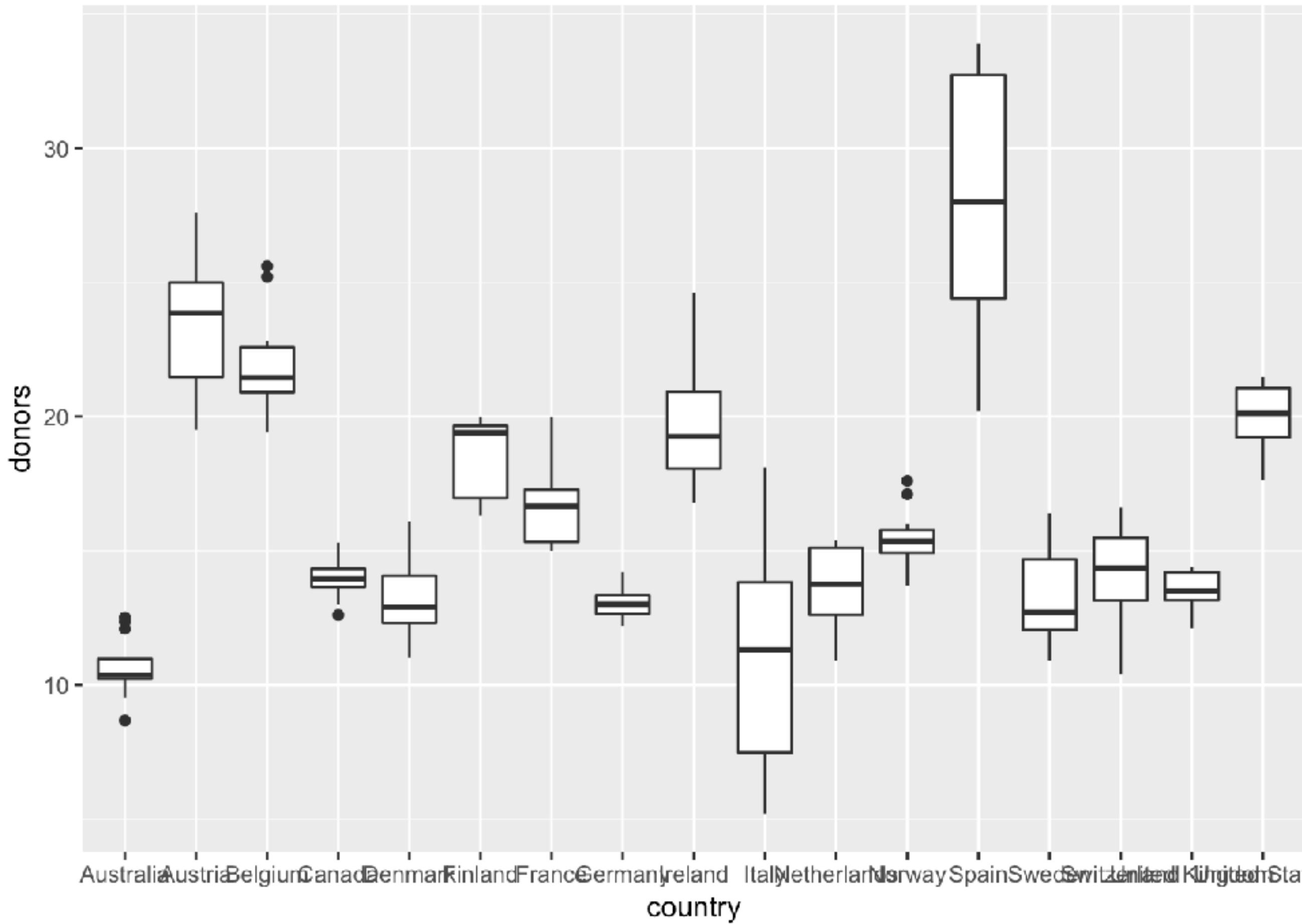
# Exploration

```
ggplot(data = organdata, mapping = aes(x = year, y = donors)) +  
  geom_point(na.rm = T, size = 0.5) +  
  geom_line(aes(group = country), na.rm = T) +  
  facet_wrap(~ country)
```



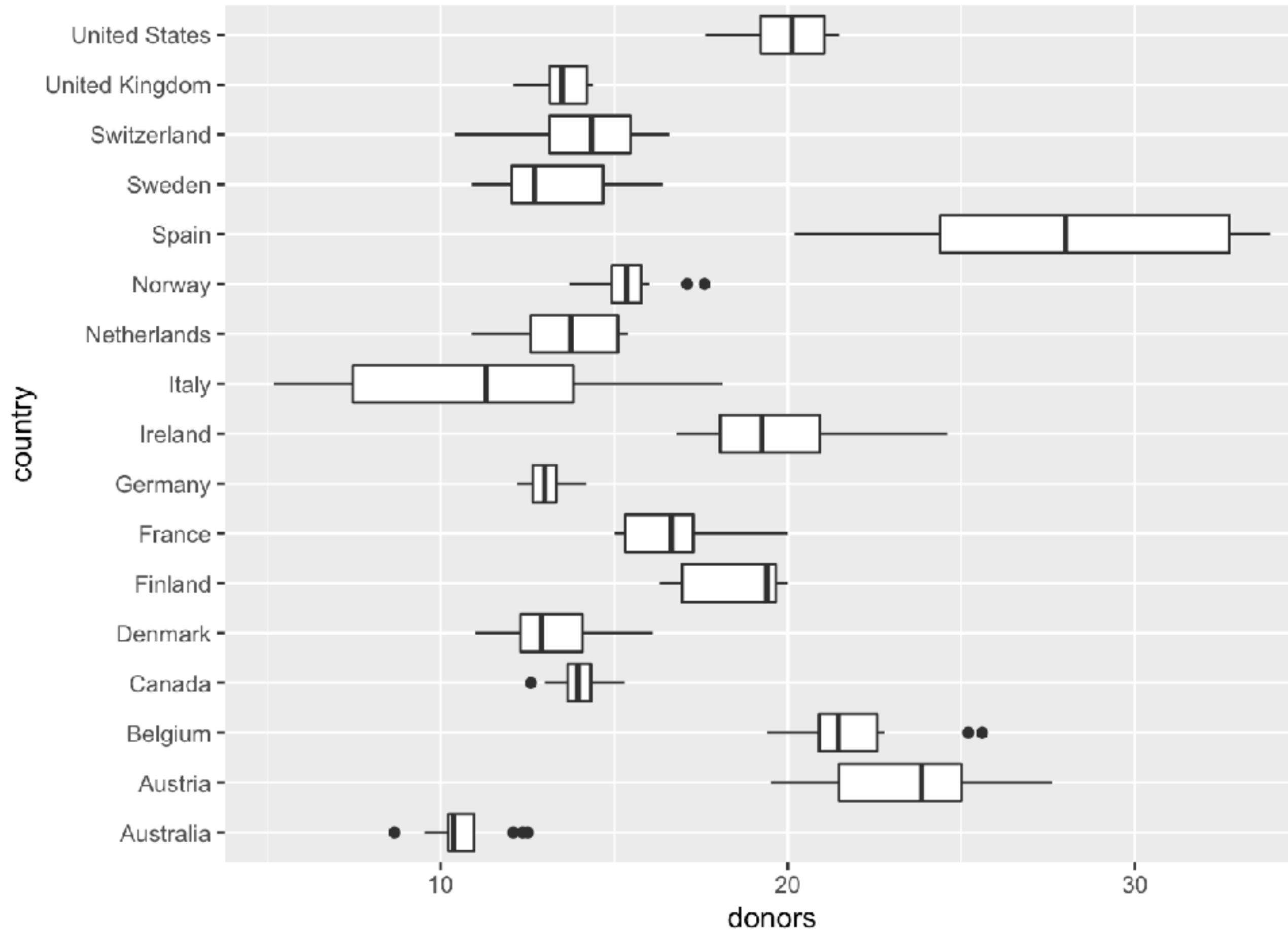
# Country-level

```
ggplot(data = organdata, mapping = aes(x = country, y = donors)) +  
  geom_boxplot(na.rm = T)
```



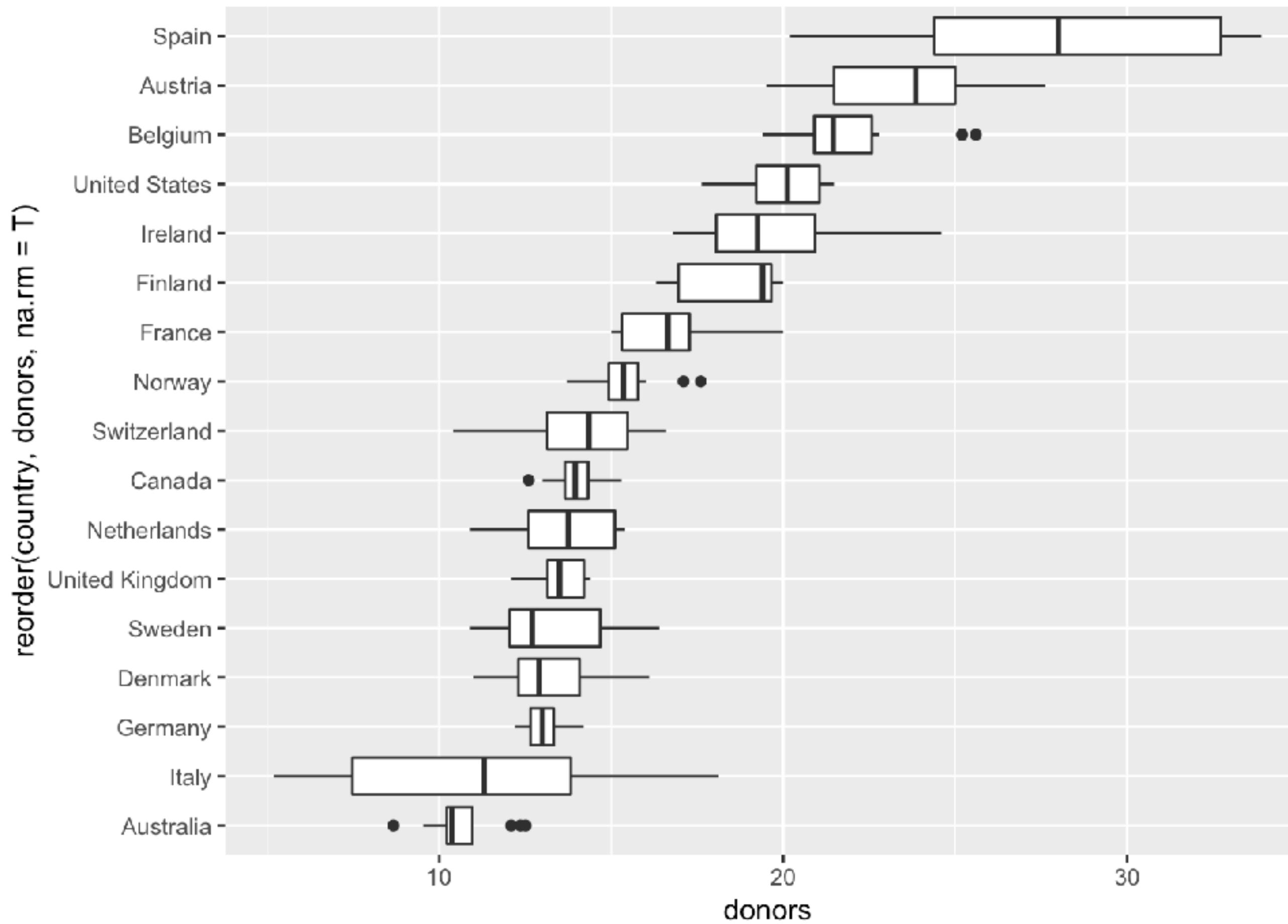
# Country-level

```
ggplot(data = organdata, mapping = aes(x = country, y = donors)) +  
  geom_boxplot(na.rm = T) +  
  coord_flip()
```



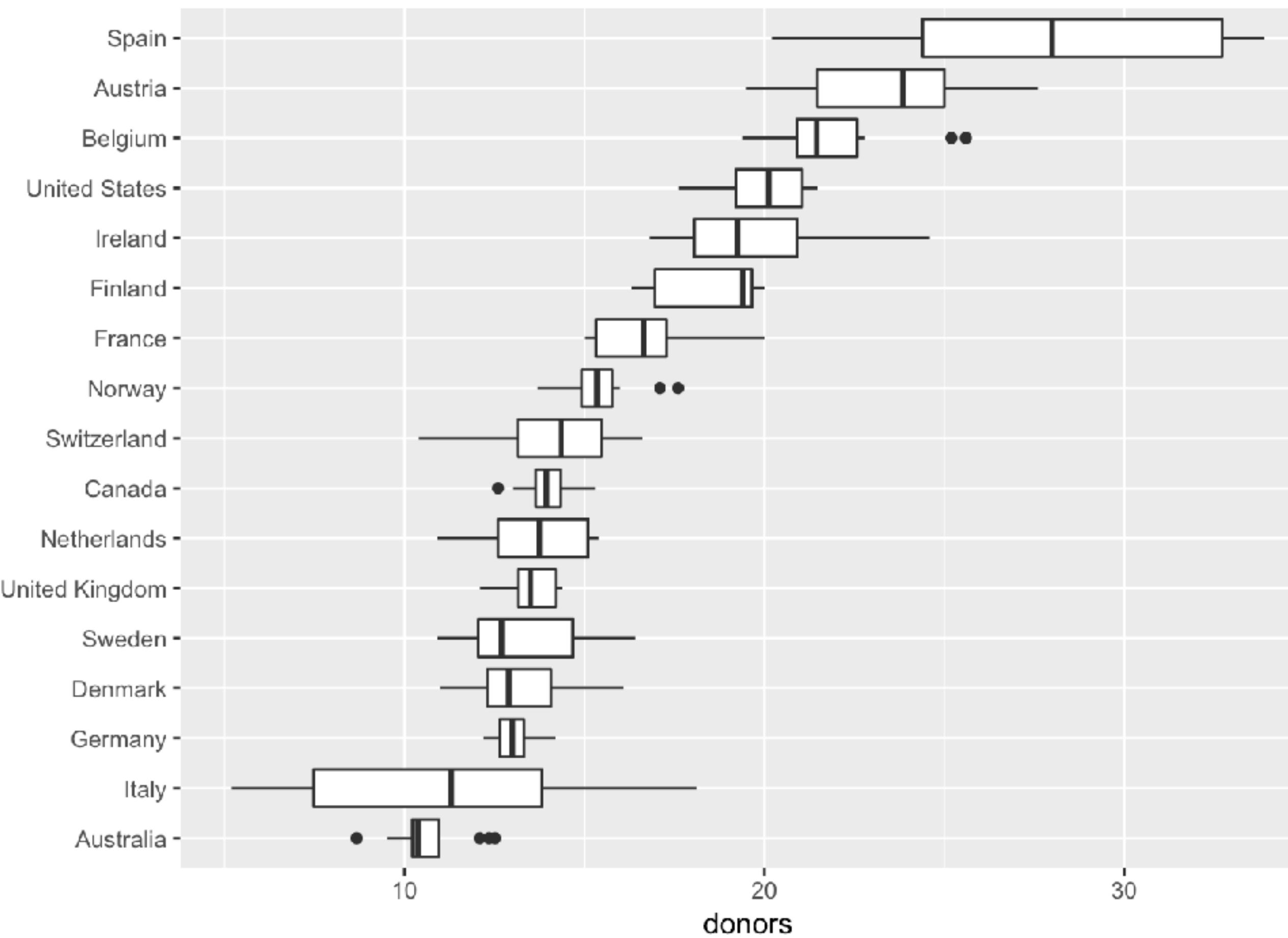
# Country-level

```
ggplot(data = organdata,  
       mapping = aes(x = reorder(country, donors, na.rm = T), y = donors)) +  
  geom_boxplot(na.rm = T) +  
  coord_flip()
```



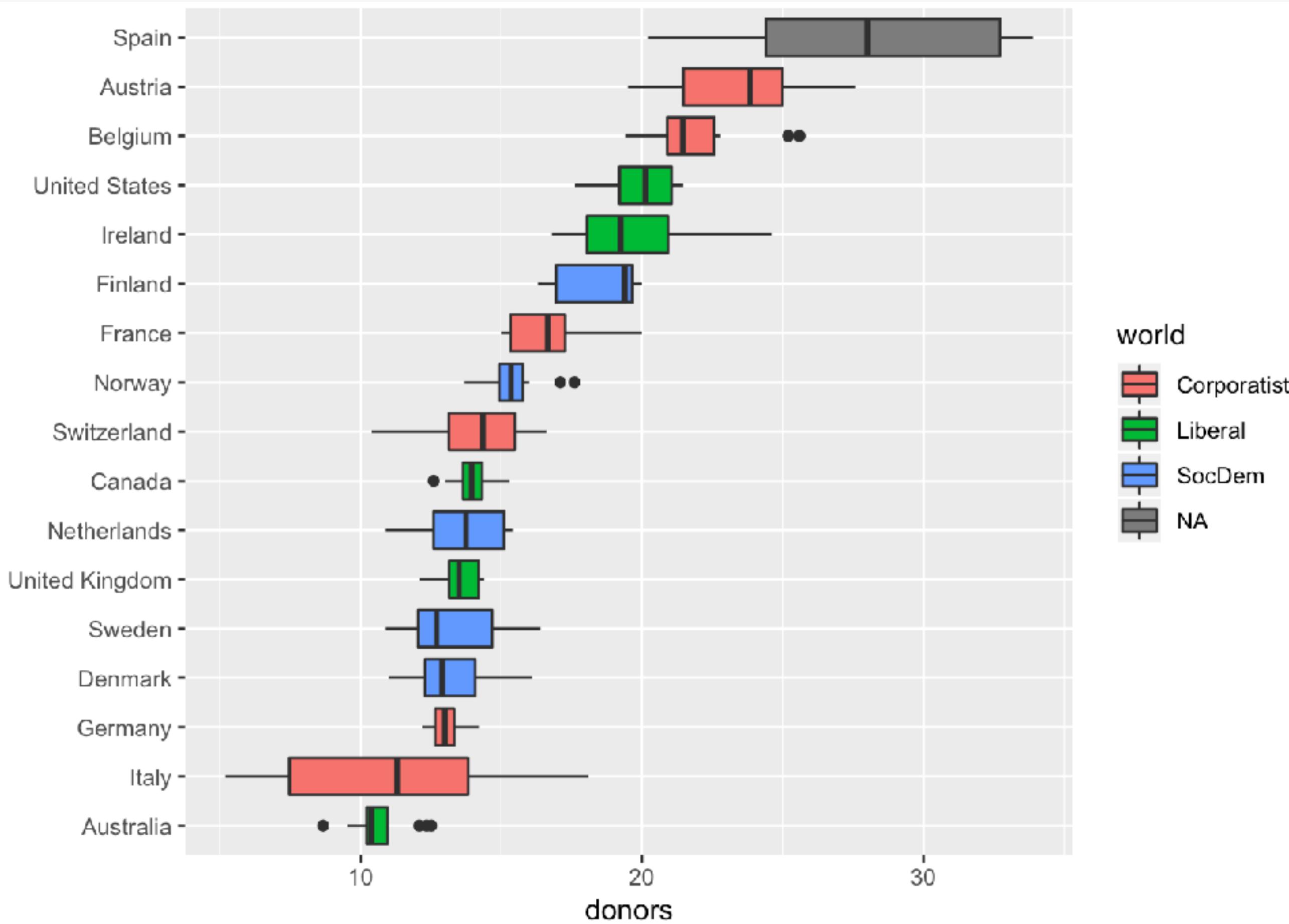
# Country-level

```
ggplot(data = organdata,  
       mapping = aes(x = reorder(country, donors, na.rm = T), y = donors)) +  
  geom_boxplot(na.rm = T) +  
  coord_flip() + labs(x = NULL)
```



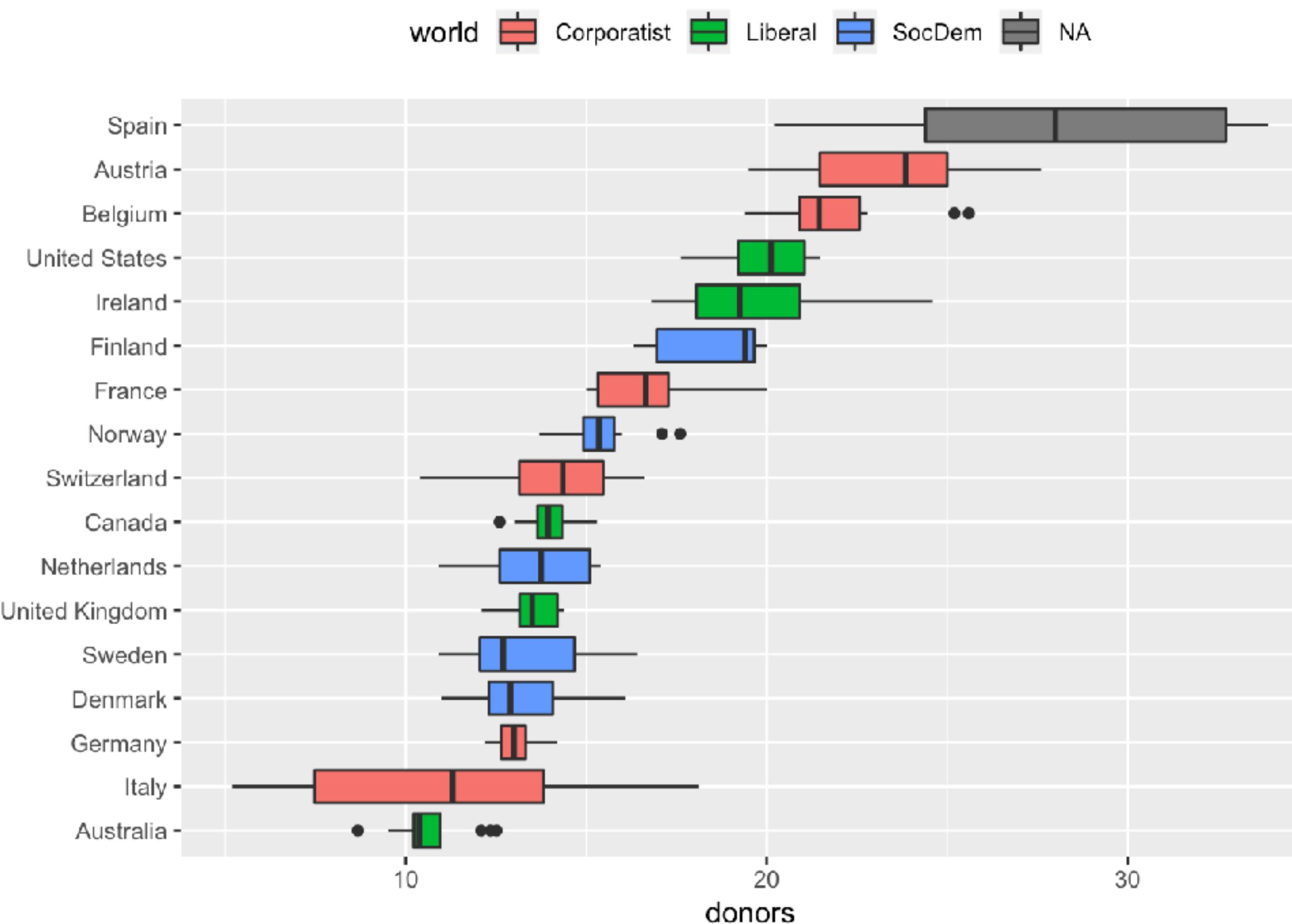
# Country-level

```
ggplot(data = organdata,  
       mapping = aes(x = reorder(country, donors, na.rm = T),  
                      y = donors, fill = world)) +  
  geom_boxplot(na.rm = T) +  
  coord_flip() + labs(x = NULL)
```



# Country-level

```
ggplot(data = organdata,  
       mapping = aes(x = reorder(country, donors, na.rm = T),  
                      y = donors, fill = world)) +  
  geom_boxplot(na.rm = T) +  
  coord_flip() + labs(x = NULL) + theme(legend.position = "top")
```



# Data transformation

```
by_country <- organdata %>%
  group_by(consent_law, country) %>%
  summarize(donors_mean= mean(donors, na.rm = TRUE),
            donors_sd = sd(donors, na.rm = TRUE),
            gdp_mean = mean(gdp, na.rm = TRUE),
            health_mean = mean(health, na.rm = TRUE),
            roads_mean = mean(roads, na.rm = TRUE),
            cerebvas_mean = mean(cerebvas, na.rm = TRUE))
```

```
summary(organdata)
##      country             year          donors        pop
##  Length:238   Min.   :1991-01-01   Min.   : 5.20   Min.   : 3514
##  Class :character  1st Qu.:1993-10-01  1st Qu.:13.00  1st Qu.: 6938
##  Mode  :character  Median :1996-07-02  Median :15.10  Median :15531
##                  Mean   :1996-07-02  Mean   :16.48  Mean   :39921
##                  3rd Qu.:1999-04-02  3rd Qu.:19.60  3rd Qu.:57301
##                  Max.   :2002-01-01  Max.   :33.90  Max.   :288369
##                  NA's    :34        NA's    :34        NA's    :17
```

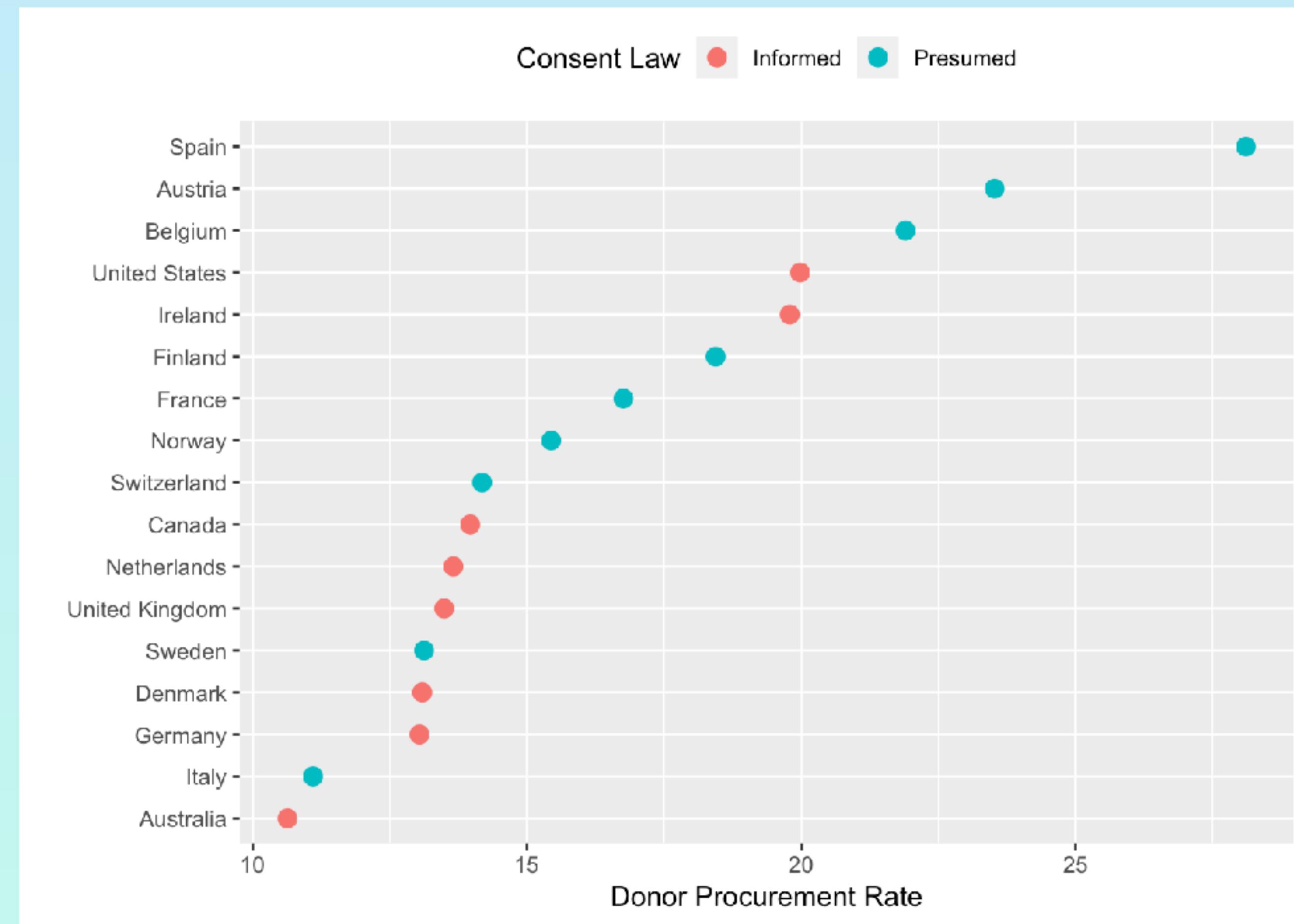
# Data transformation

```
by_country <- organdata %>%
  group_by(consent_law, country) %>%
  summarize(donors_mean = mean(donors, na.rm = TRUE),
            donors_sd = sd(donors, na.rm = TRUE),
            gdp_mean = mean(gdp, na.rm = TRUE),
            health_mean = mean(health, na.rm = TRUE),
            roads_mean = mean(roads, na.rm = TRUE),
            cerebvas_mean = mean(cerebvas, na.rm = TRUE))
```

```
by_country <- organdata %>%
  group_by(consent_law, country) %>%
  summarize_if(is.numeric, funs(mean, sd), na.rm = TRUE) %>%
  ungroup()
```

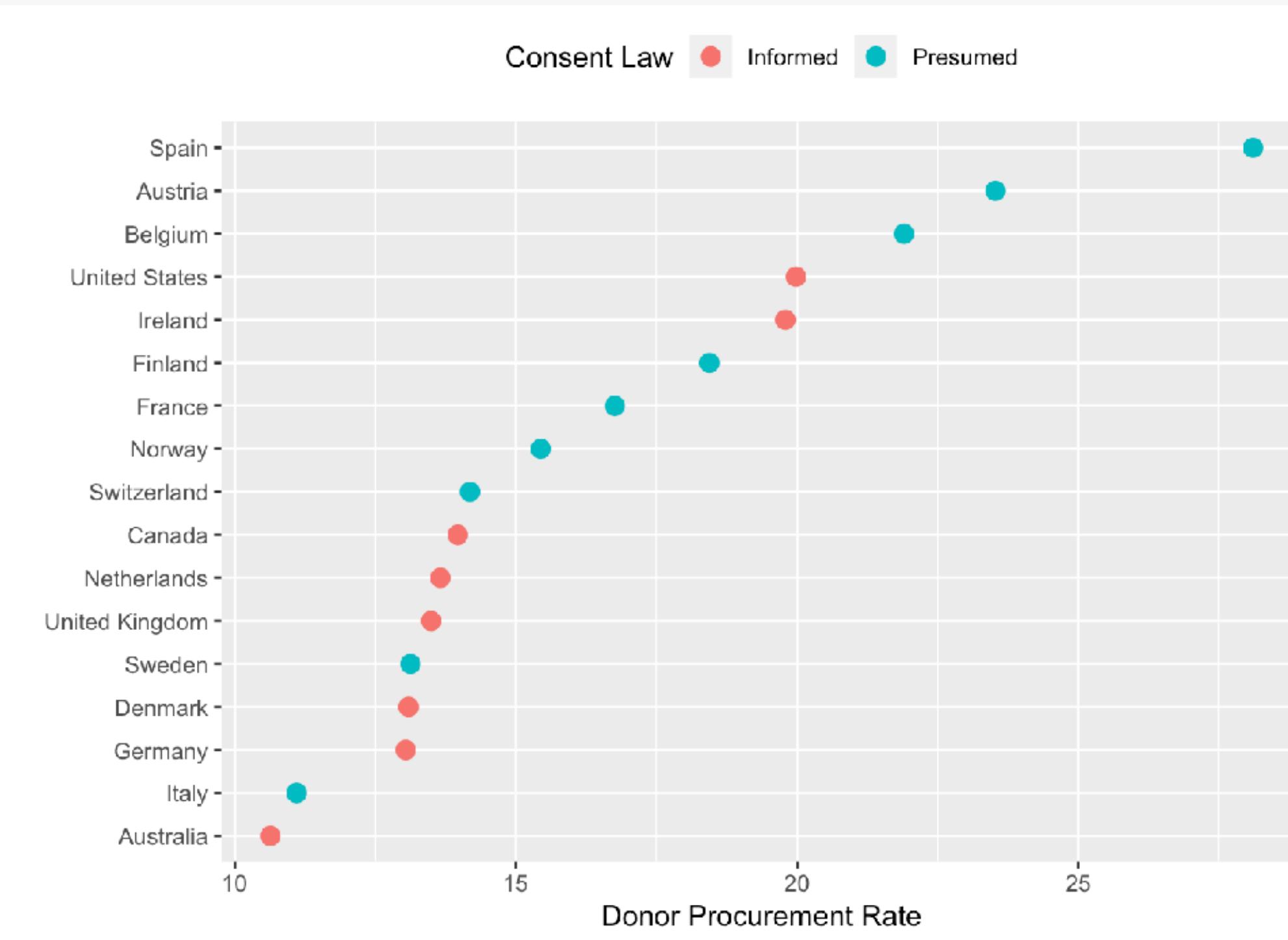
# Your turn!

- 1) Using the by\_country data, try to reproduce the following plot. This style of graph is called Cleveland dotplot.



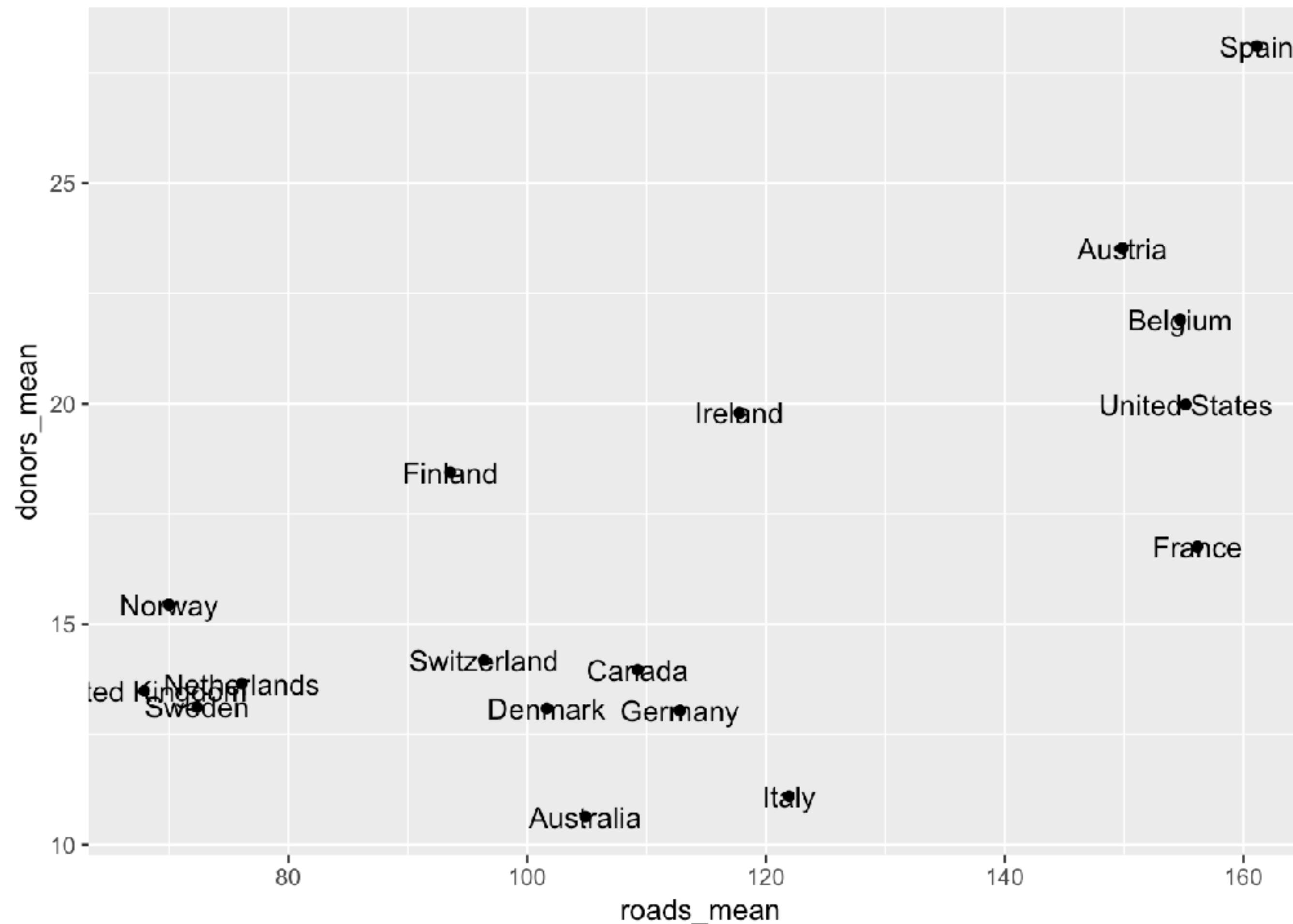
# Solution

```
ggplot(data = by_country,  
       mapping = aes(x = donors_mean, y = reorder(country, donors_mean),  
                      color = consent_law)) +  
  geom_point(size=3) +  
  labs(x = "Donor Procurement Rate", y = "", color = "Consent Law") +  
  theme(legend.position="top")
```



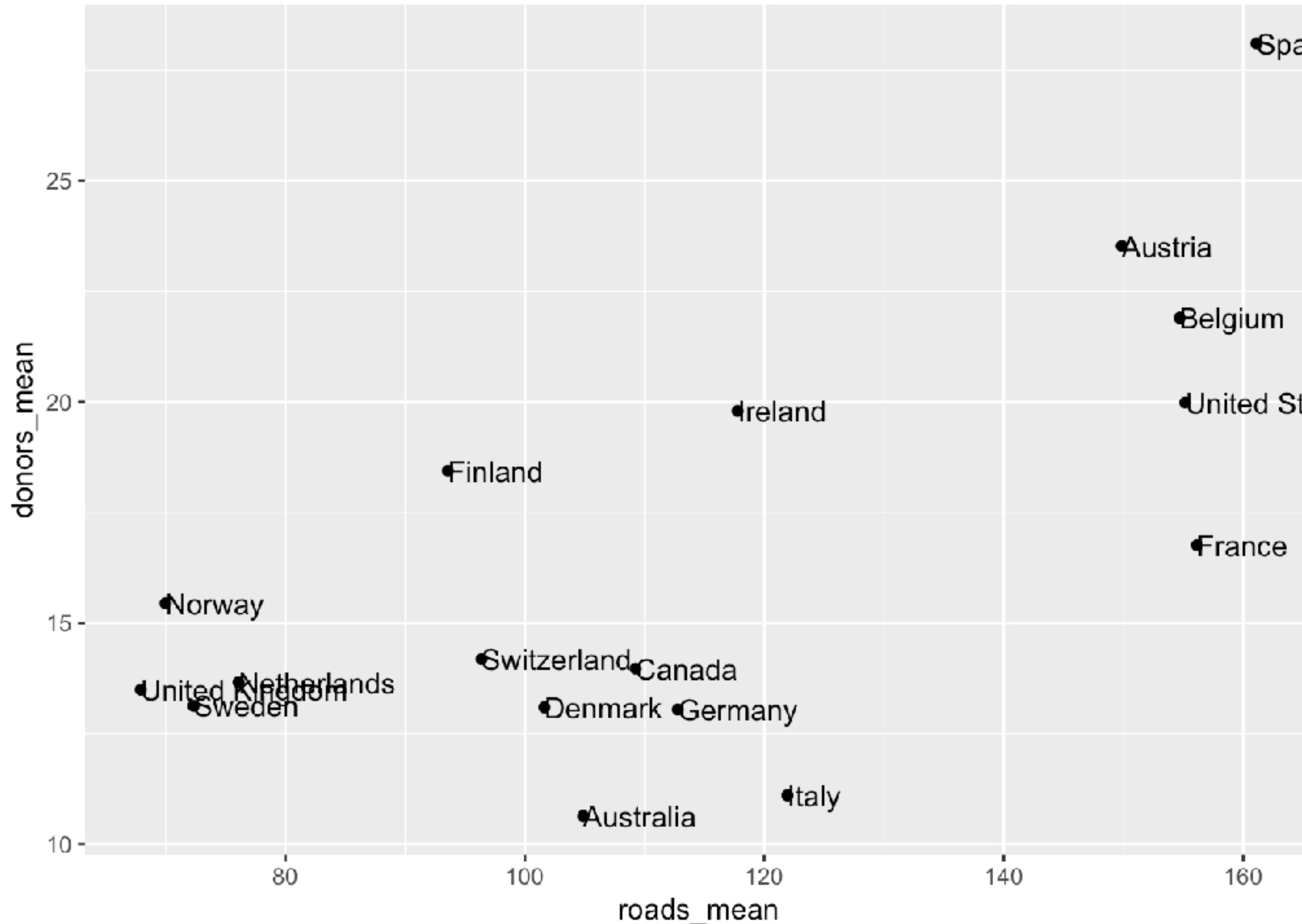
# Text labels

```
ggplot(data = by_country, mapping = aes(x = roads_mean, y = donors_mean)) +  
  geom_point() +  
  geom_text(mapping = aes(label = country))
```



# Text labels

```
ggplot(data = by_country, mapping = aes(x = roads_mean, y = donors_mean)) +  
  geom_point() +  
  geom_text(mapping = aes(label = country), hjust = 0)
```

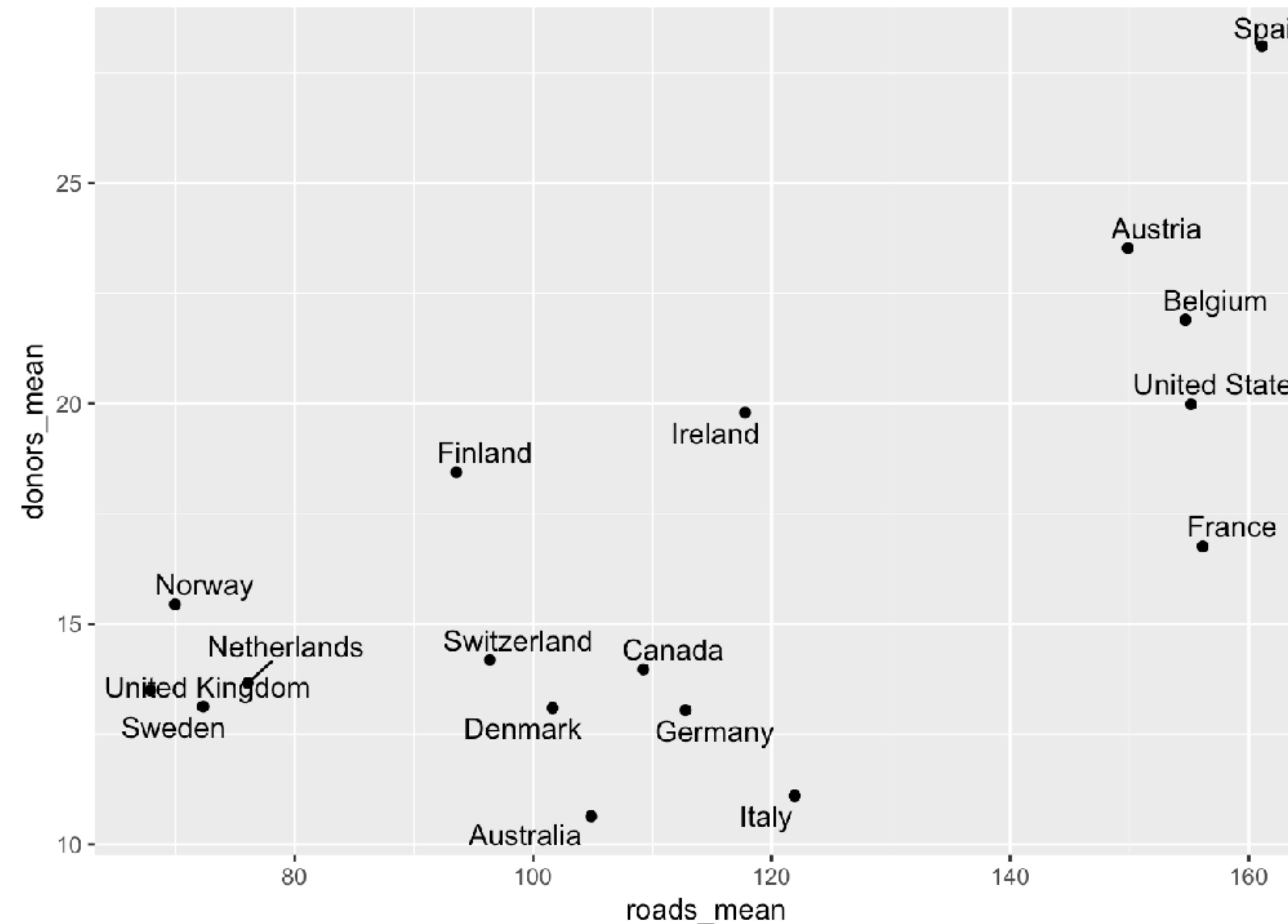


# ggrepel package

- Better and easier way to plot texts
- Key functions:
  - geom\_text\_repel()
  - geom\_label\_repel()
- Install ggrepel

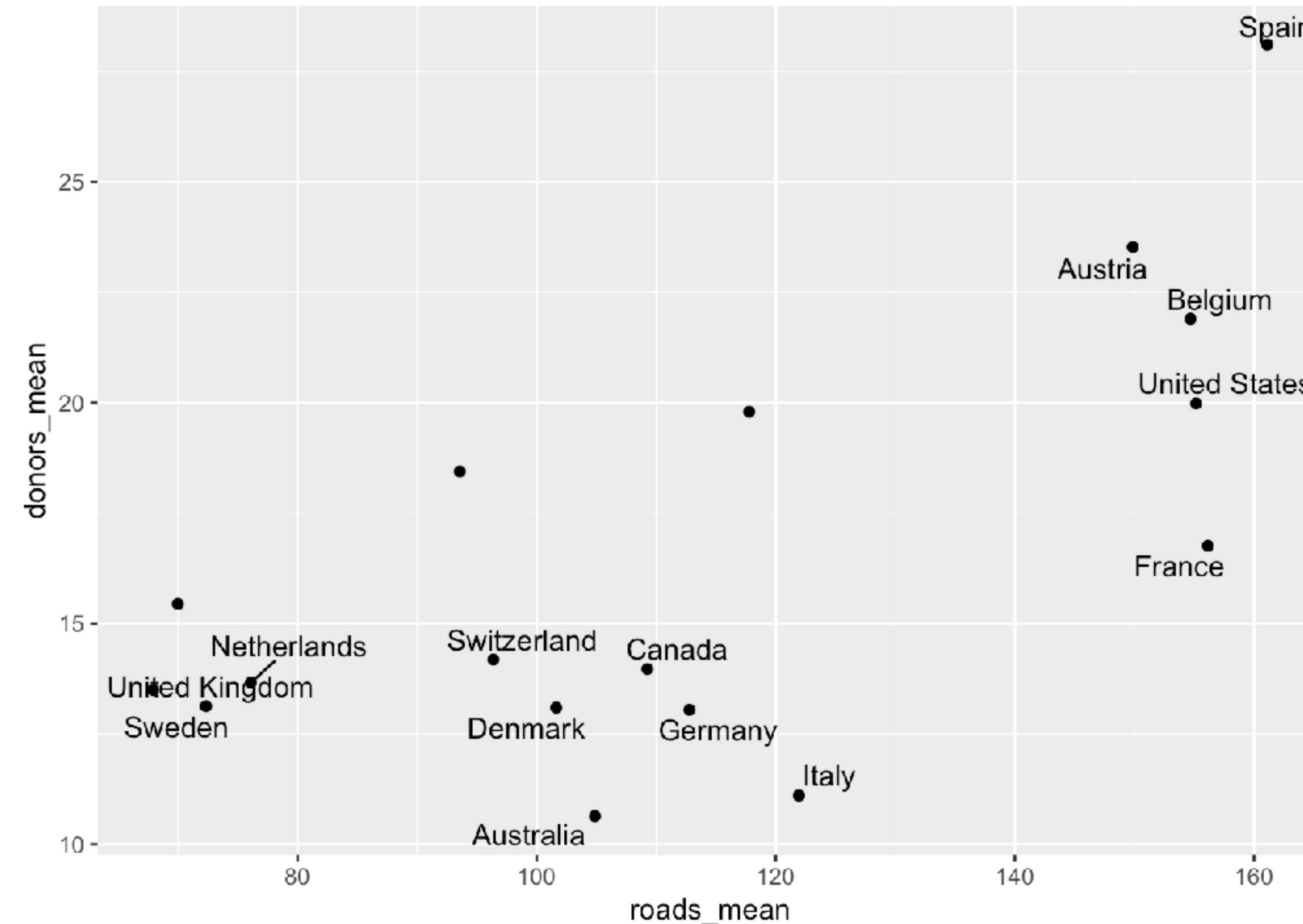
# ggrepel

```
ggplot(data = by_country, mapping = aes(x = roads_mean, y = donors_mean)) +  
  geom_point() +  
  geom_text_repel(mapping = aes(label = country))
```



# label outliers

```
ggplot(data = by_country, mapping = aes(x = roads_mean, y = donors_mean)) +  
  geom_point() +  
  geom_text_repel(data = subset(by_country, roads_mean > 140 | donors_mean < 15), mapping = aes(label = country))
```



# Movie data

ggplot2movies

# IMDB.com

- **ggplot2movies::movies**
  - 28819 movies and 24 variables
    - title: Title of the movie
    - year: Year of release.
    - budget: Total budget (if known) in US dollars
    - length: Length in minutes.
    - rating: Average IMDB user rating.
- Let's explore the data and find long/short movies

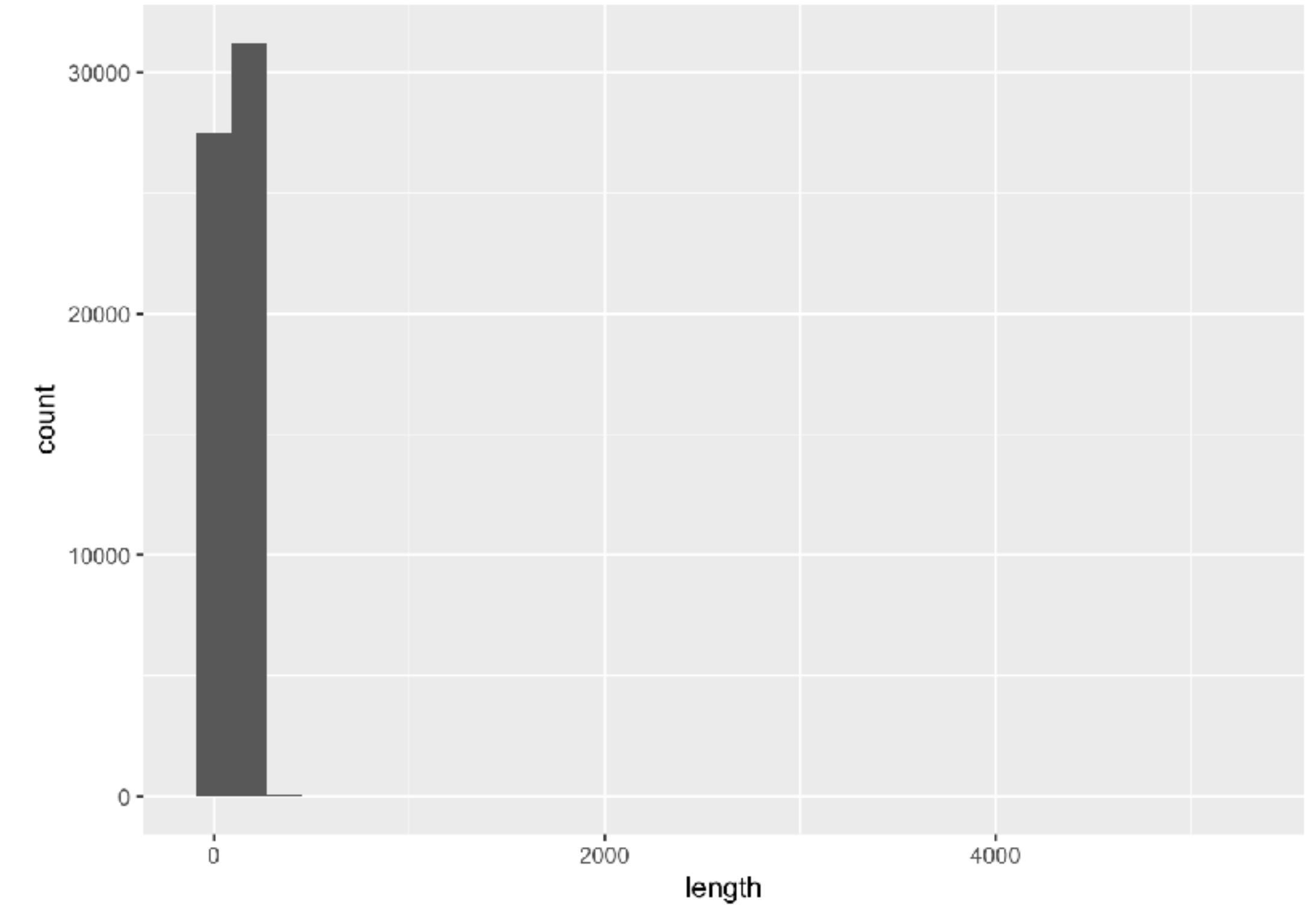
# Long films

1. Assess the distribution of movie lengths
2. How would you define “long”?
3. How many long movies are there?
4. What are the top 5 longest movies?
5. Create a new variable that signals these as “long” movies

1. Assess the distribution of movie lengths
2. How would you define “long”?
3. How many long movies are there?
4. What are the top 5 longest movies?
5. Create a new variable that signals these as “long” movies

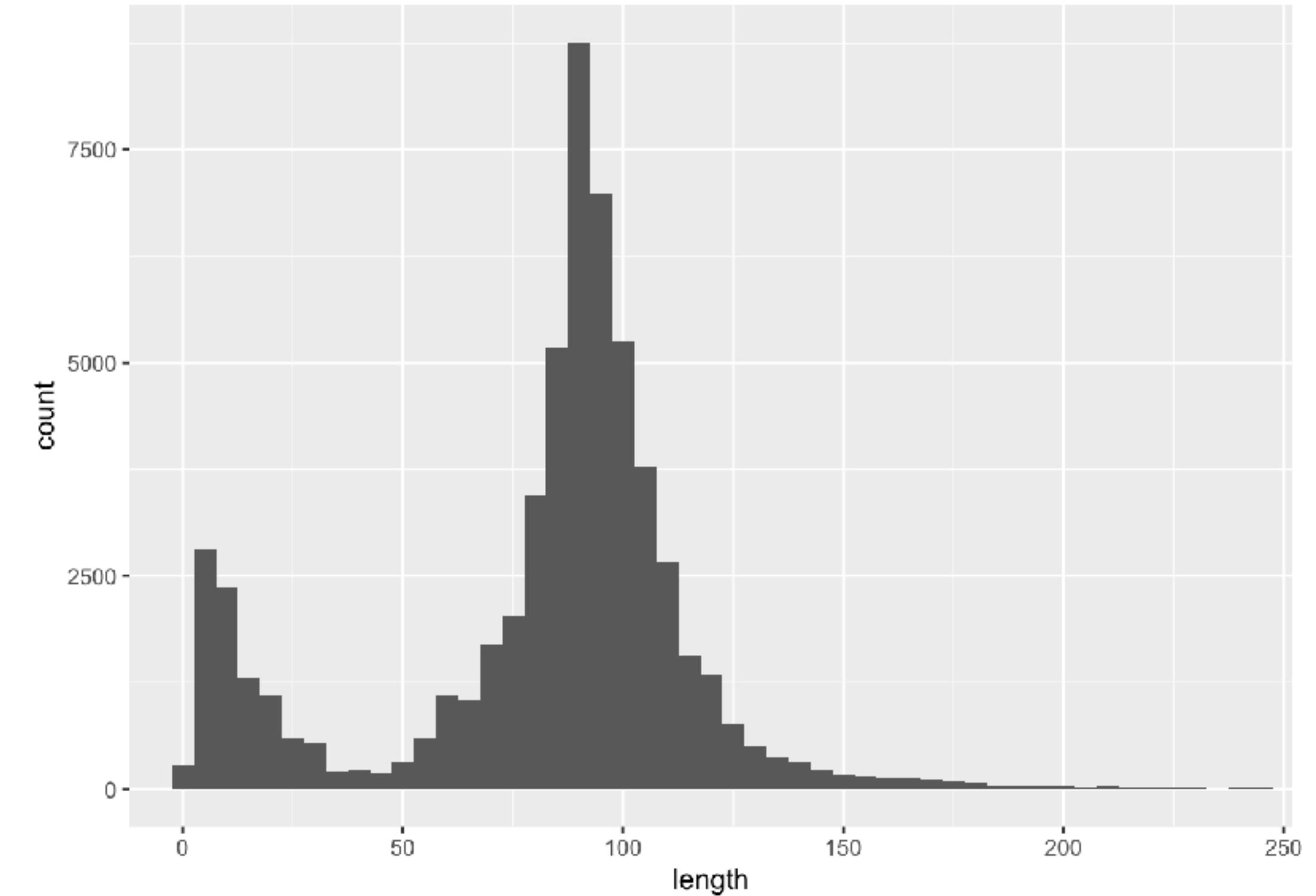
# Assessing the distribution

```
ggplot(data = movies, mapping = aes(x = length))+  
  geom_histogram()
```



# Assessing the distribution

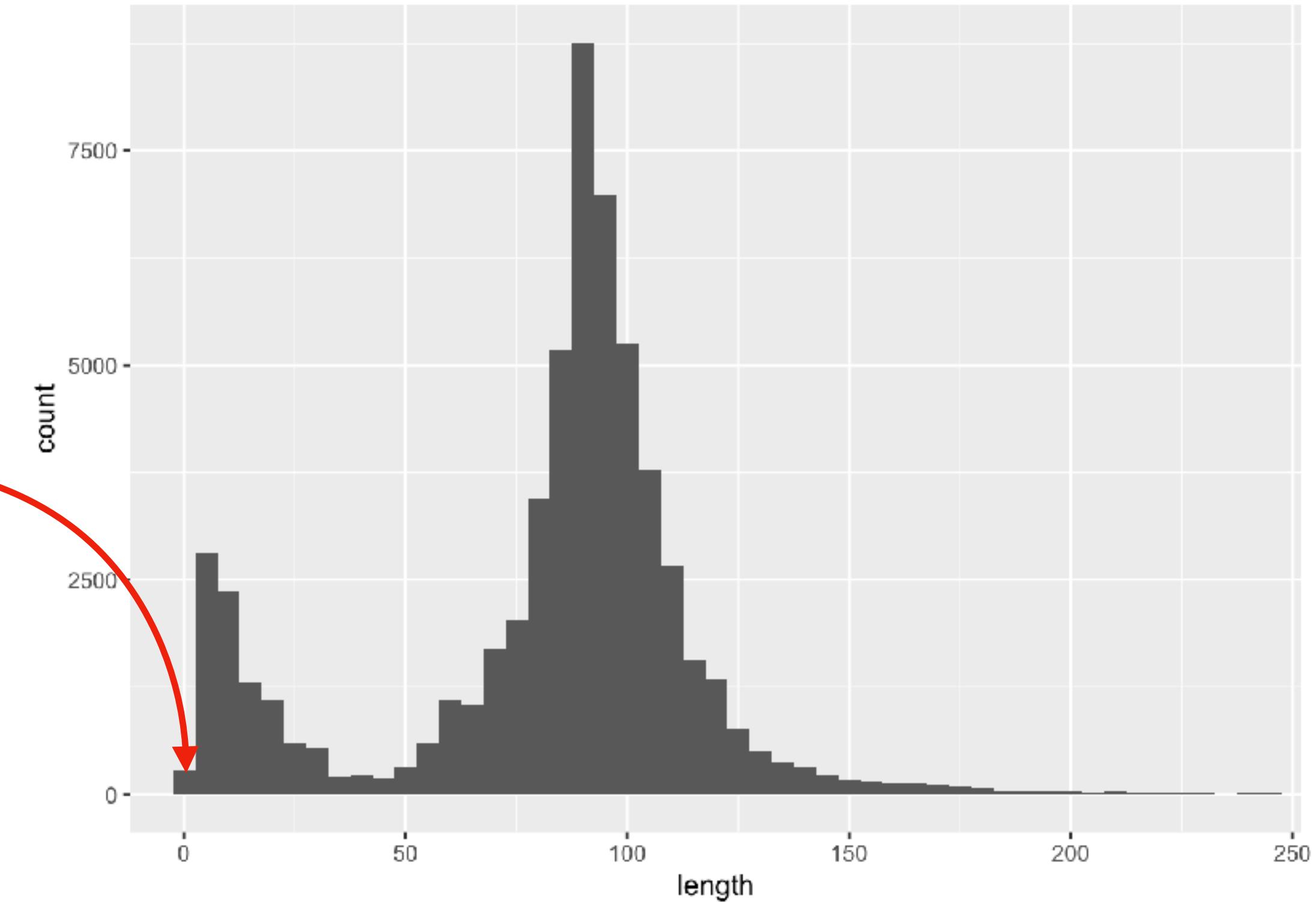
```
ggplot(data = movies, mapping = aes(x = length))+  
  geom_histogram(binwidth = 5) +  
  coord_cartesian(xlim = c(0, 60*4))
```



# Assessing the distribution

```
movies %>%  
  count(cut_width(length, 5))
```

```
## # A tibble: 85 x 2  
##   `cut_width(length, 5)` <fct>  
##   n <int>  
## 1 [-2.5,2.5]    285  
## 2 (2.5,7.5]    2812  
## 3 (7.5,12.5]   2366  
## 4 (12.5,17.5]  1300  
## 5 (17.5,22.5]  1092  
## 6 (22.5,27.5]  595  
## 7 (27.5,32.5]  532  
## 8 (32.5,37.5]  194  
## 9 (37.5,42.5]  236  
## 10 (42.5,47.5] 182  
## # ... with 75 more rows
```

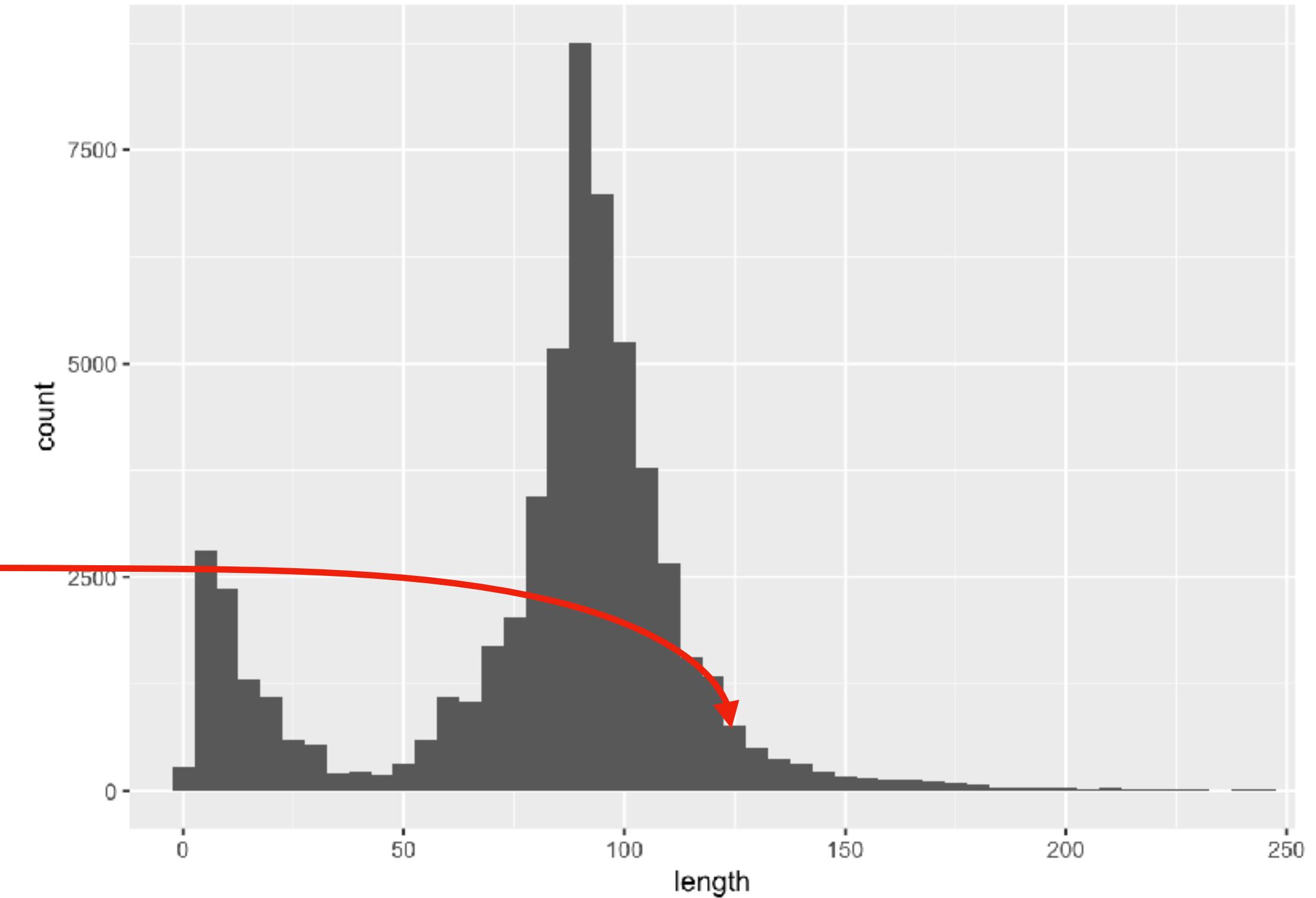


1. Assess the distribution of movie lengths
2. How would you define “long”?
3. How many long movies are there?
4. What are the top 5 longest movies?
5. Create a new variable that signals these as “long” movies

# Defining “long”

```
movies %>%  
  count(cut_width(length, 5)) %>%  
  mutate(cum_pct = cumsum(n)/sum(n)) %>%  
  filter(cum_pct > .95)
```

```
## # A tibble: 60 x 3  
##   `cut_width(length, 5)`     n  cum_pct  
##   <fct>                  <int>  <dbl>  
## 1 (122.5,127.5]           766  0.955  
## 2 (127.5,132.5]          514  0.964  
## 3 (132.5,137.5]          378  0.970  
## 4 (137.5,142.5]          322  0.976  
## 5 (142.5,147.5]          216  0.979  
## 6 (147.5,152.5]          165  0.982  
## 7 (152.5,157.5]          139  0.984  
## 8 (157.5,162.5]          135  0.987  
## 9 (162.5,167.5]          134  0.989  
## 10 (167.5,172.5]         114  0.991  
## # ... with 50 more rows
```

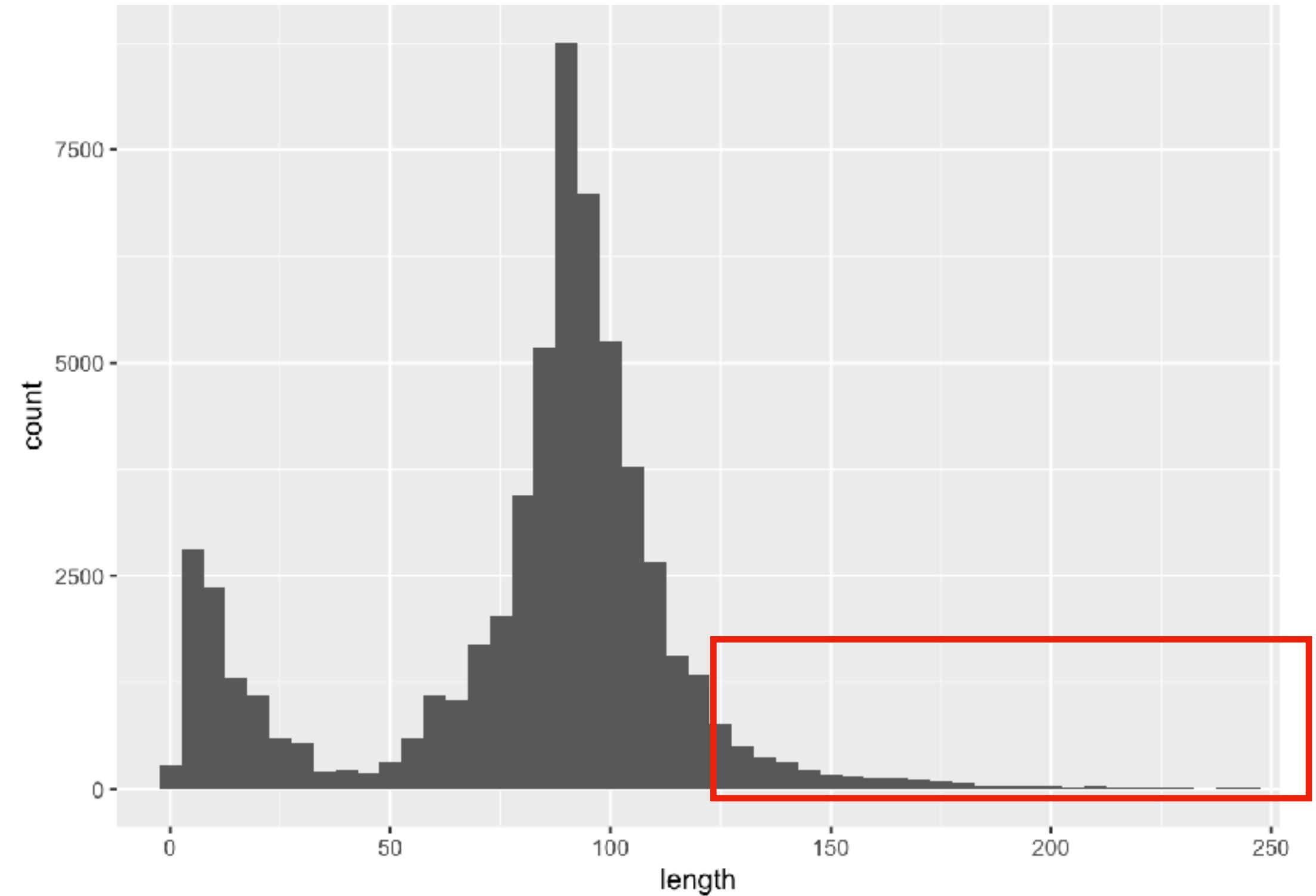


1. Assess the distribution of movie lengths
2. How would you define “long”?
3. How many long movies are there?
4. What are the top 5 longest movies?
5. Create a new variable that signals these as “long” movies

# How many long movies?

```
movies %>%  
  count(cut_width(length, 5)) %>%  
  mutate(cum_pct = cumsum(n)/sum(n)) %>%  
  filter(cum_pct > .95) %>%  
  summarise(sum(n))
```

```
## # A tibble: 1 x 1  
##   `sum(n)`  
##     <int>  
##   1     3415
```



1. Assess the distribution of movie lengths
2. How would you define “long”?
3. How many long movies are there?
4. What are the top 5 longest movies?
5. Create a new variable that signals these as “long” movies

# Top 5 longest

```
movies %>%  
  arrange(desc(length)) %>%  
  top_n(5, wt = length)
```

```
# A tibble: 5 × 24
```

| title  | year | length | budget | rating | votes |
|--------|------|--------|--------|--------|-------|
|        |      |        |        |        |       |
| a, The | 1987 | 5220   | NA     | 3.8    | 59    |
| d, The | 1970 | 2880   | NA     | 6.4    | 15    |
| Stars  | 1967 | 1100   | NA     | 3.0    | 12    |
| Resan  | 1987 | 873    | NA     | 5.5    | 12    |
| Out 1  | 1971 | 773    | NA     | 6.7    | 20    |

```
# ... with 18 more variables: r1 <dbl>, r2 <dbl>, r3 <dbl>, r4 <dbl>, r5 <dbl>,
#   r6 <dbl>, r7 <dbl>, r8 <dbl>, r9 <dbl>, r10 <dbl>, mpaa <chr>, Action <int>,
#   Animation <int>, Comedy <int>, Drama <int>, Documentary <int>, Romance <int>,
#   Short <int>
```

1. Assess the distribution of movie lengths
2. How would you define “long”?
3. How many long movies are there?
4. What are the top 5 longest movies?
5. Create a new variable that signals these as “long” movies

# Labelling long movies

```
# create a new variable that signals these as "Long movies"
movies %>%
  select(1:3) %>%
  mutate(Long = length >= 122.5)
```

# Short films

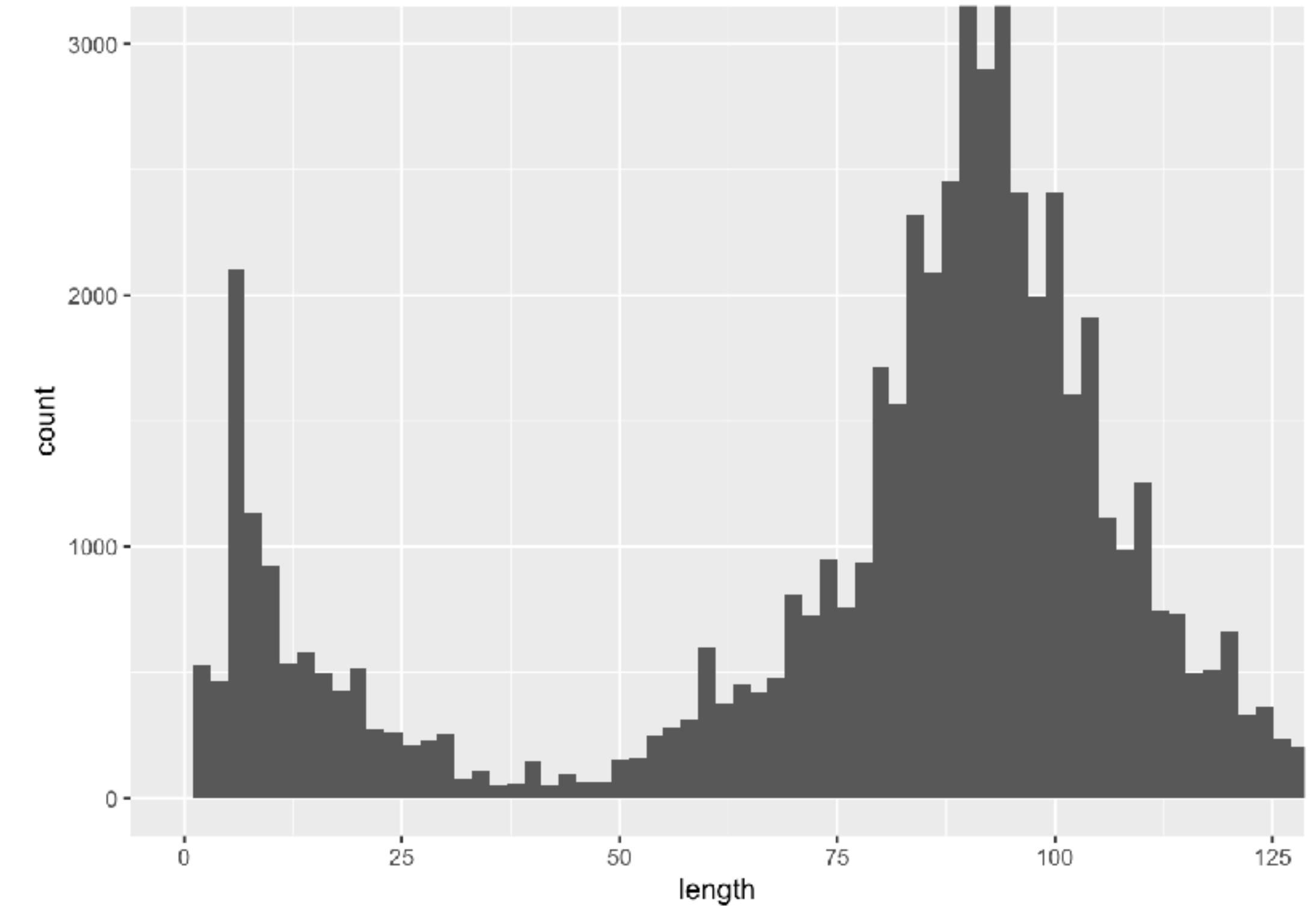
1. How did you determine where short films start and stop?
2. How many short films are there?
3. What is the average length of short films?
4. Create a new variable that signals these as “short” movies

# Short films

1. How did you determine where short films start and stop?
2. How many short films are there?
3. What is the average length of short films?
4. Create a new variable that signals these as “short” movies

# Defining short movies

```
ggplot(data = movies, mapping = aes(length)) +  
  geom_histogram(binwidth = 2) +  
  coord_cartesian(xlim = c(0, 122.5),  
                   ylim = c(0, 3000))
```



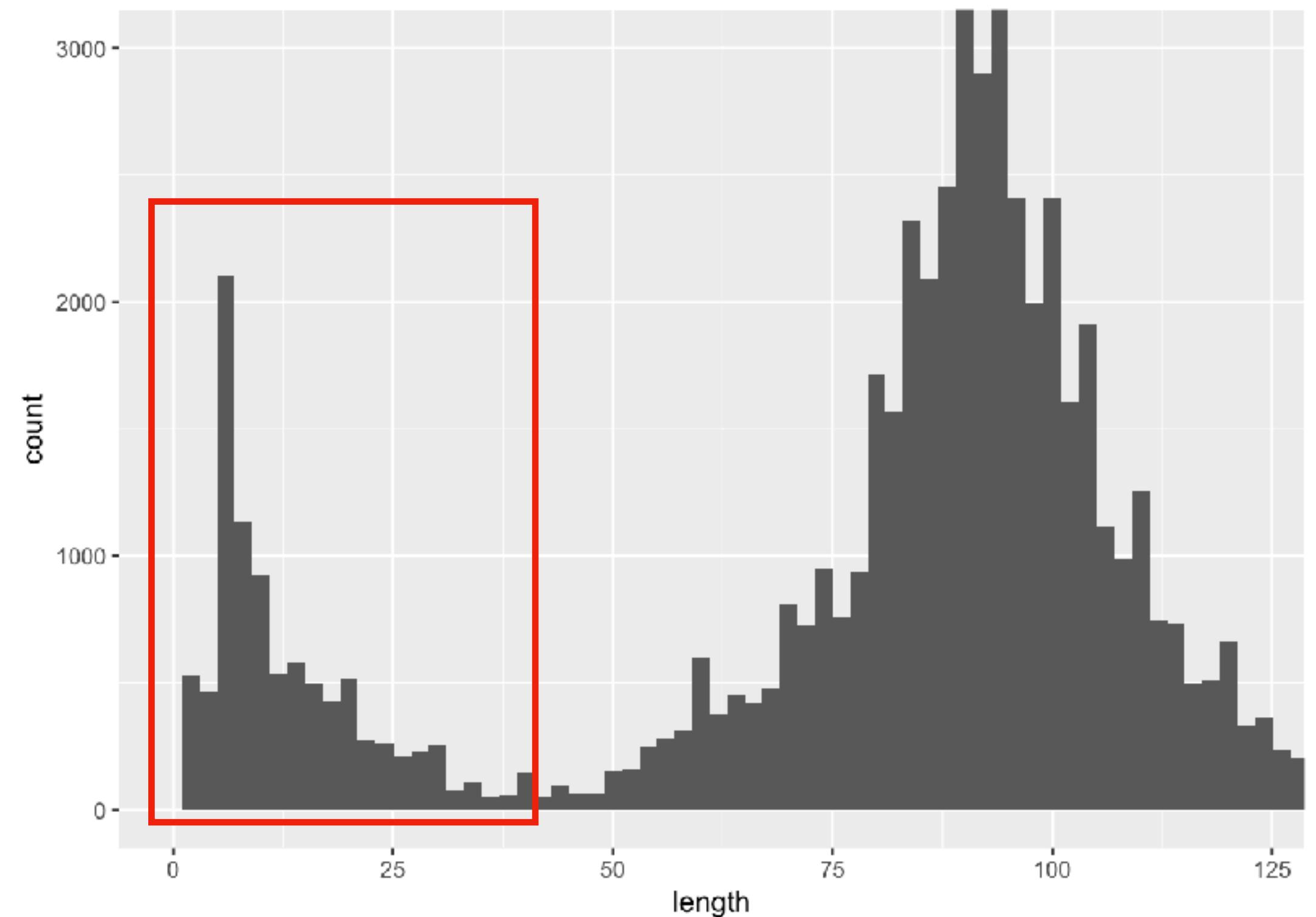
# Short films

1. How did you determine where short films start and stop?
2. How many short films are there?
3. What is the average length of short films?
4. Create a new variable that signals these as “short” movies

# How many short films?

```
movies %>%  
  filter(length <= 40) %>%  
  summarise(n())
```

```
## # A tibble: 1 x 1  
##   `n()`  
##   <int>  
## 1 9353
```



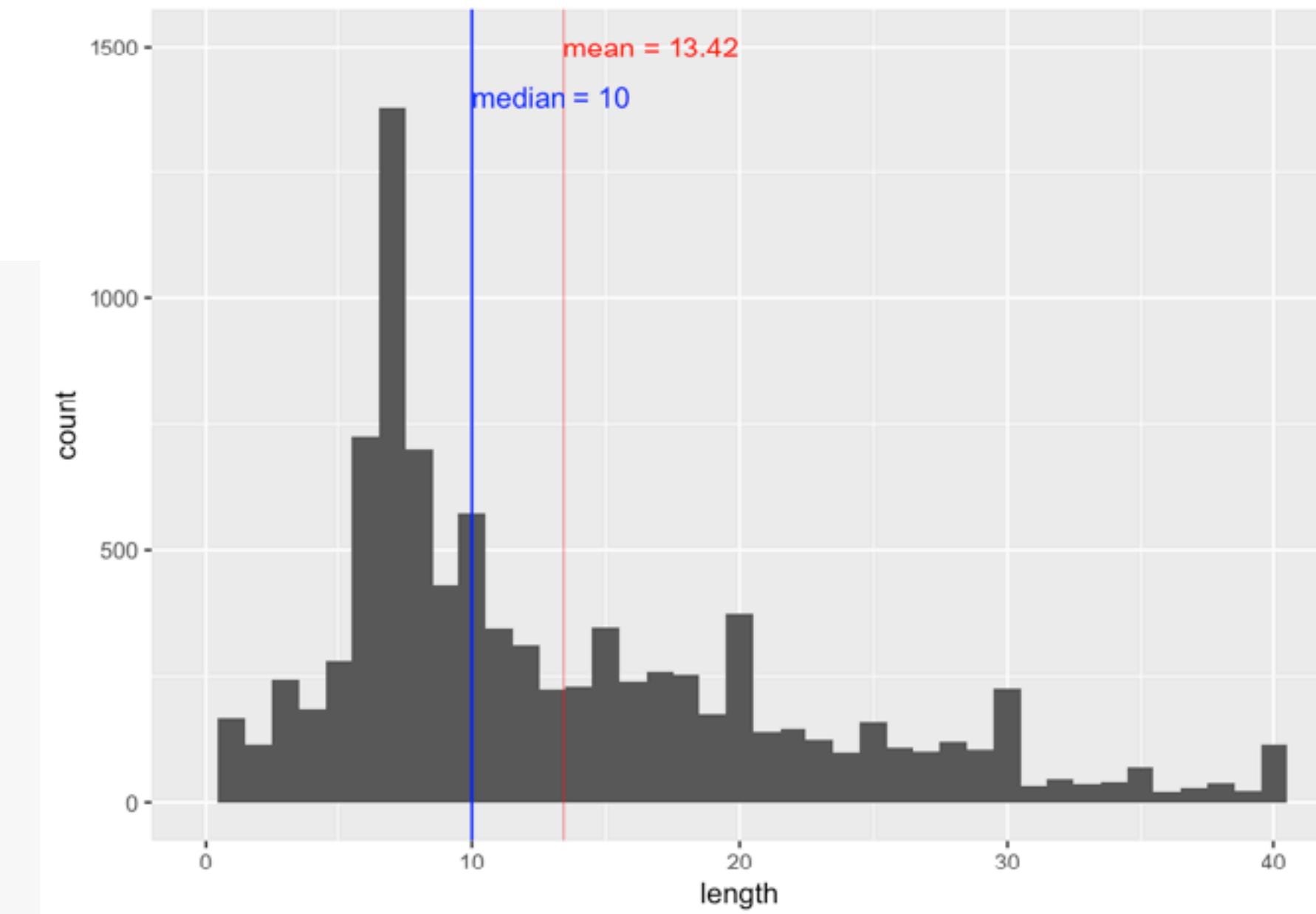
# Short films

1. How did you determine where short films start and stop?
2. How many short films are there?
3. What is the average length of short films?
4. Create a new variable that signals these as “short” movies

# Average length

```
m_summary <- movies %>%
  filter(length <= 40) %>%
  summarize(mean = mean(length, na.rm = T),
            median = median(length, na.rm = T))

ggplot(data = movies %>% filter(length <= 40), mapping = aes(length)) +
  geom_histogram(binwidth = 1) +
  coord_cartesian(xlim = c(0, 40), ylim = c(0, 1500)) +
  geom_vline(xintercept = m_summary$mean, colour = "red")+
  geom_vline(xintercept = m_summary$median, colour = "blue")+
  annotate("text", x = m_summary$mean, y = 1500, hjust = 0,
           label= paste("mean = ", round(m_summary$mean, digits = 2), sep = ""), color = "red")+
  annotate("text", x = m_summary$median, y = 1400, hjust = 0,
           label= paste("median = ", round(m_summary$median, digits = 2), sep = ""), color = "blue")
```



# Short films

1. How did you determine where short films start and stop?
2. How many short films are there?
3. What is the average length of short films?
4. Create a new variable that signals these as “short” movies

# How many short films?

```
movies %>%
  select(1:3) %>%
  mutate(Description = ifelse(length >= 122.5, "Long",
                             ifelse(length <= 40, "Short", "Regular")))
```

```
## # A tibble: 58,788 x 4
##   title                      year length Description
##   <chr>                     <int>  <int>  <chr>
## 1 $                           1971    121 Regular
## 2 $1000 a Touchdown          1939     71 Regular
## 3 $21 a Day Once a Month    1941      7 Short
## 4 $40,000                      1996    70 Regular
## 5 $50,000 Climax Show, The  1975     71 Regular
## 6 $pent                        2000    91 Regular
## 7 $windle                       2002    93 Regular
## 8 '15'                          2002    25 Short
## 9 '38                           1987    97 Regular
## 10 '49-'17                      1917    61 Regular
## # ... with 58,778 more rows
```

# “Regular” films

1. What is the average length of “regular” films?
2. Are there certain length cut-offs that are favoured over others?
3. How do ratings differ between short, regular, and long length films?

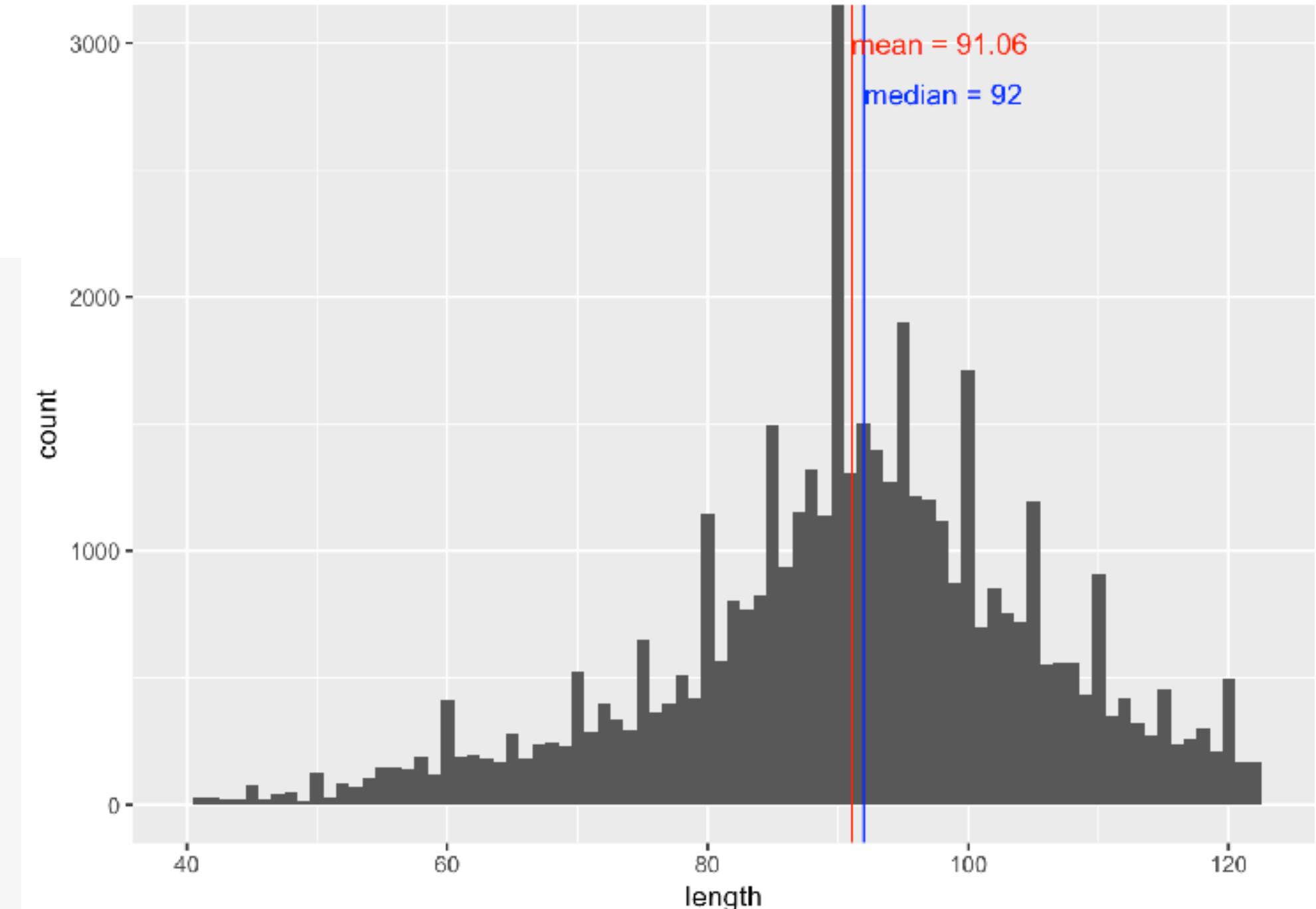
# “Regular” films

1. What is the average length of “regular” films?
2. Are there certain length cut-offs that are favoured over others?
3. How do ratings differ between short, regular, and long length films?

# Average length

```
m_summary <- movies %>%
  filter(length > 40 & length < 122.5) %>%
  summarize(mean = mean(length, na.rm = T),
            median = median(length, na.rm = T))

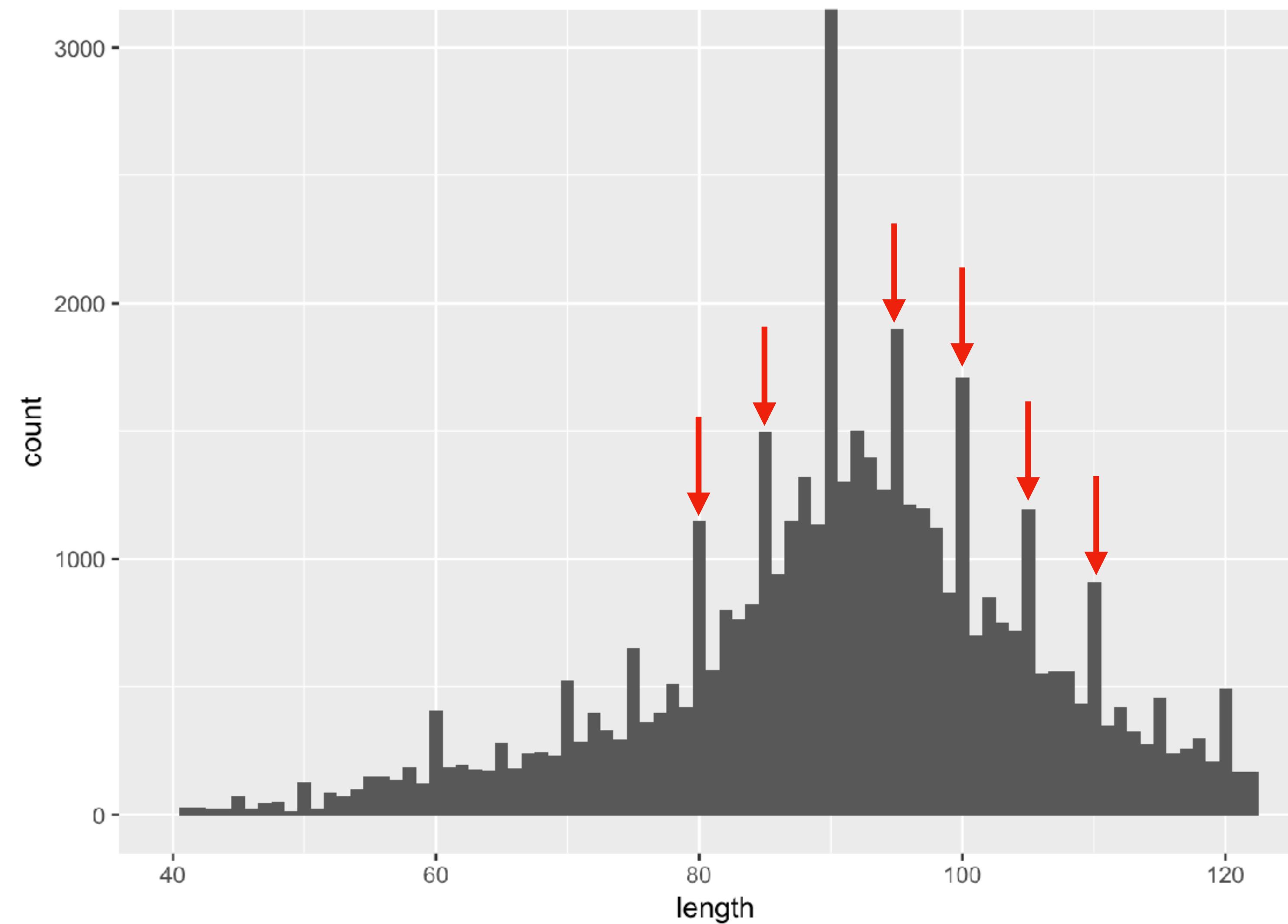
ggplot(data = movies %>% filter(length > 40 & length < 122.5),
       mapping = aes(length)) +
  geom_histogram(binwidth = 1) +
  coord_cartesian(xlim = c(40, 122.5), ylim = c(0, 3000)) +
  geom_vline(xintercept = m_summary$mean, colour = "red")+
  geom_vline(xintercept = m_summary$median, colour = "blue")+
  annotate("text", x = m_summary$mean, y = 3000, hjust = 0,
           label= paste("mean = ", round(m_summary$mean, digits = 2), sep = ""), color = "red")+
  annotate("text", x = m_summary$median, y = 2800, hjust = 0,
           label= paste("median = ", round(m_summary$median, digits = 2), sep = ""), color = "blue")
```



# “Regular” films

1. What is the average length of “regular” films?
2. Are there certain length cut-offs that are favoured over others?
3. How do ratings differ between short, regular, and long length films?

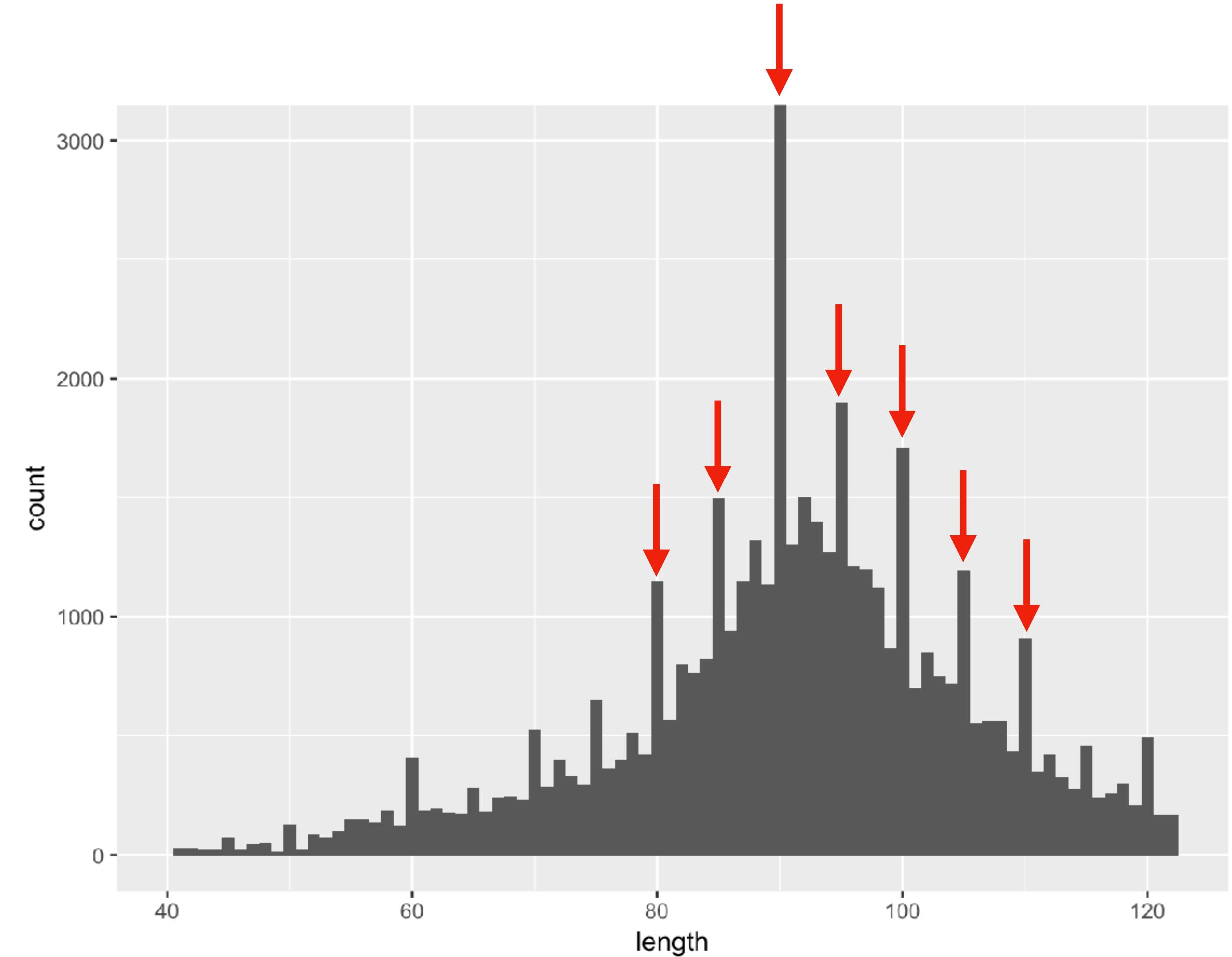
# Favoured cut-off length



# Favoured cut-off length

```
movies %>%  
  filter(length > 40 & length < 122.5) %>%  
  count(cut_width(length, 1 )) %>%  
  mutate(change = (n - lag(n)) / lag(n)) %>%  
  arrange(desc(change))
```

```
# A tibble: 82 x 3  
#> `cut_width(length, 1)`  n  change  
#> <fct>           <int>  <dbl>  
#> 1 (49.5,50.5]    129   9.75  
#> 2 (44.5,45.5]    74   2.36  
#> 3 (59.5,60.5]   409   2.33  ← 60  
#> 4 (51.5,52.5]    88   2.26  
#> 5 (89.5,90.5]  3506   2.09  ← 90  
#> 6 (79.5,80.5] 1149   1.73  ← 80  
#> 7 (119.5,120.5] 496   1.35  ← 120  
#> 8 (69.5,70.5]   523   1.25  ← 70
```

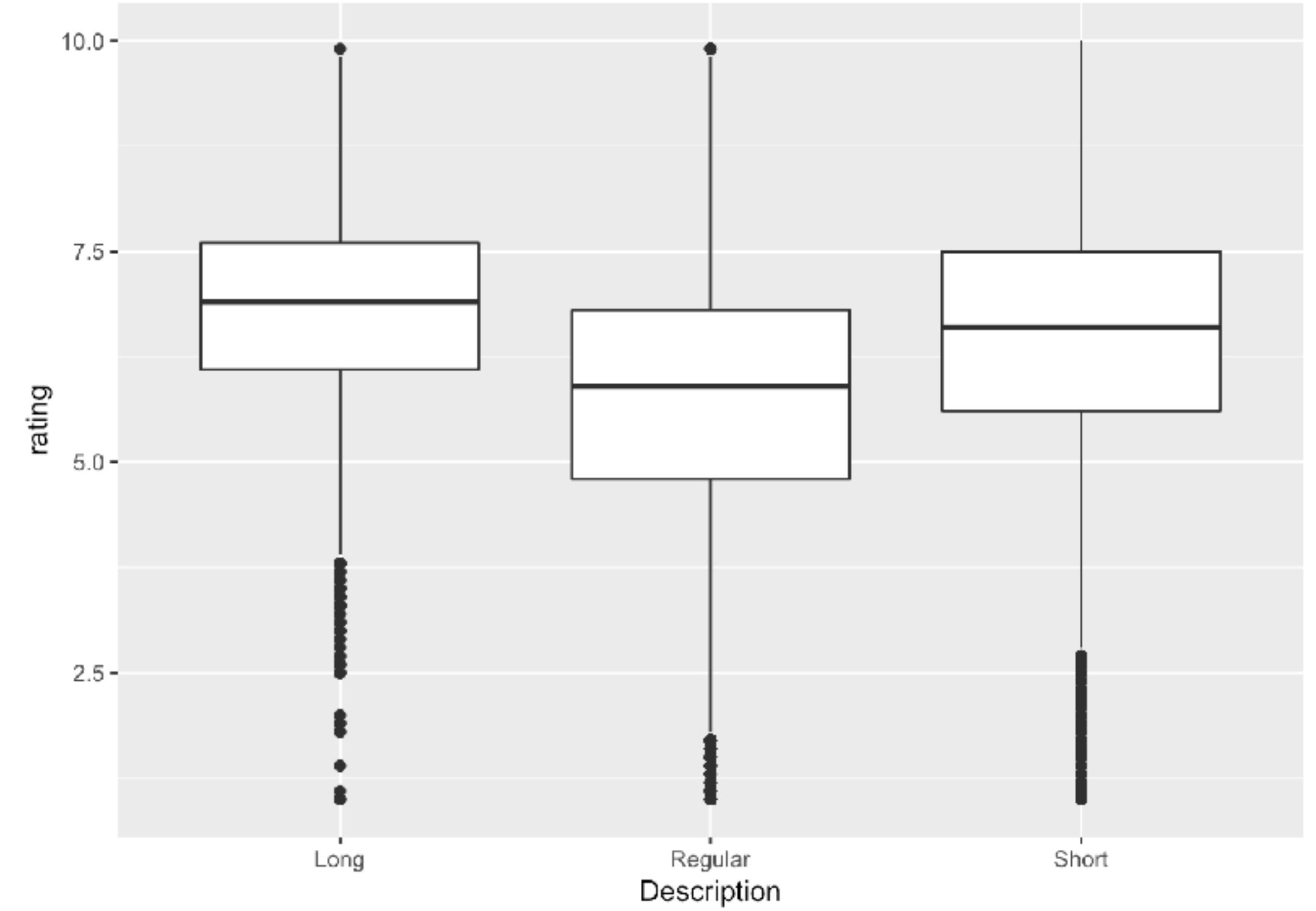


# “Regular” films

1. What is the average length of “regular” films?
2. Are there certain length cut-offs that are favoured over others?
3. How do ratings differ between short, regular, and long length films?

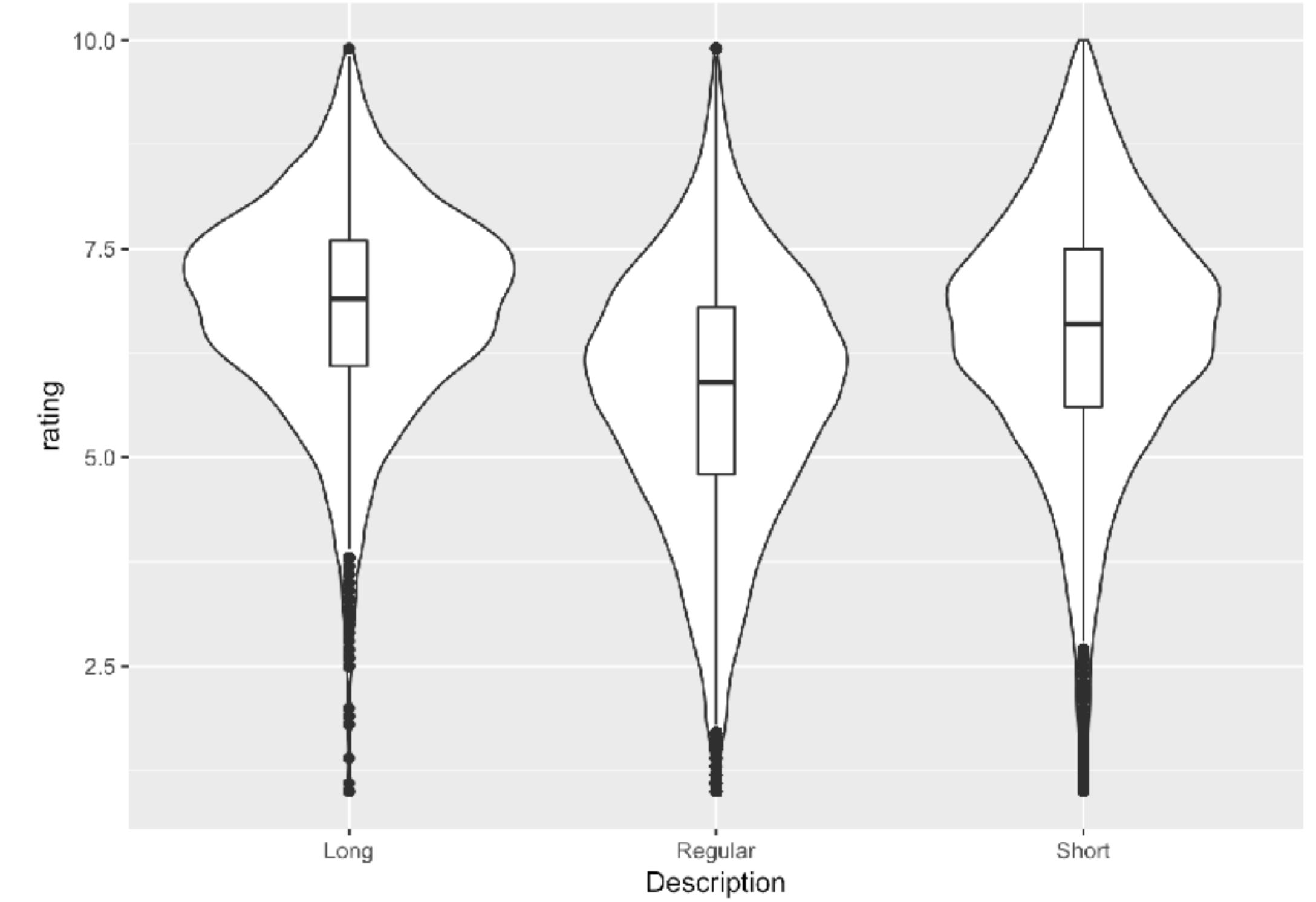
# How do ratings differ?

```
movies %>%  
  mutate(Description =  
    ifelse(length >= 122.5, "Long",  
          ifelse(length <= 40, "Short", "Regular")))) %>%  
  ggplot(aes(Description, rating)) +  
  geom_boxplot()
```



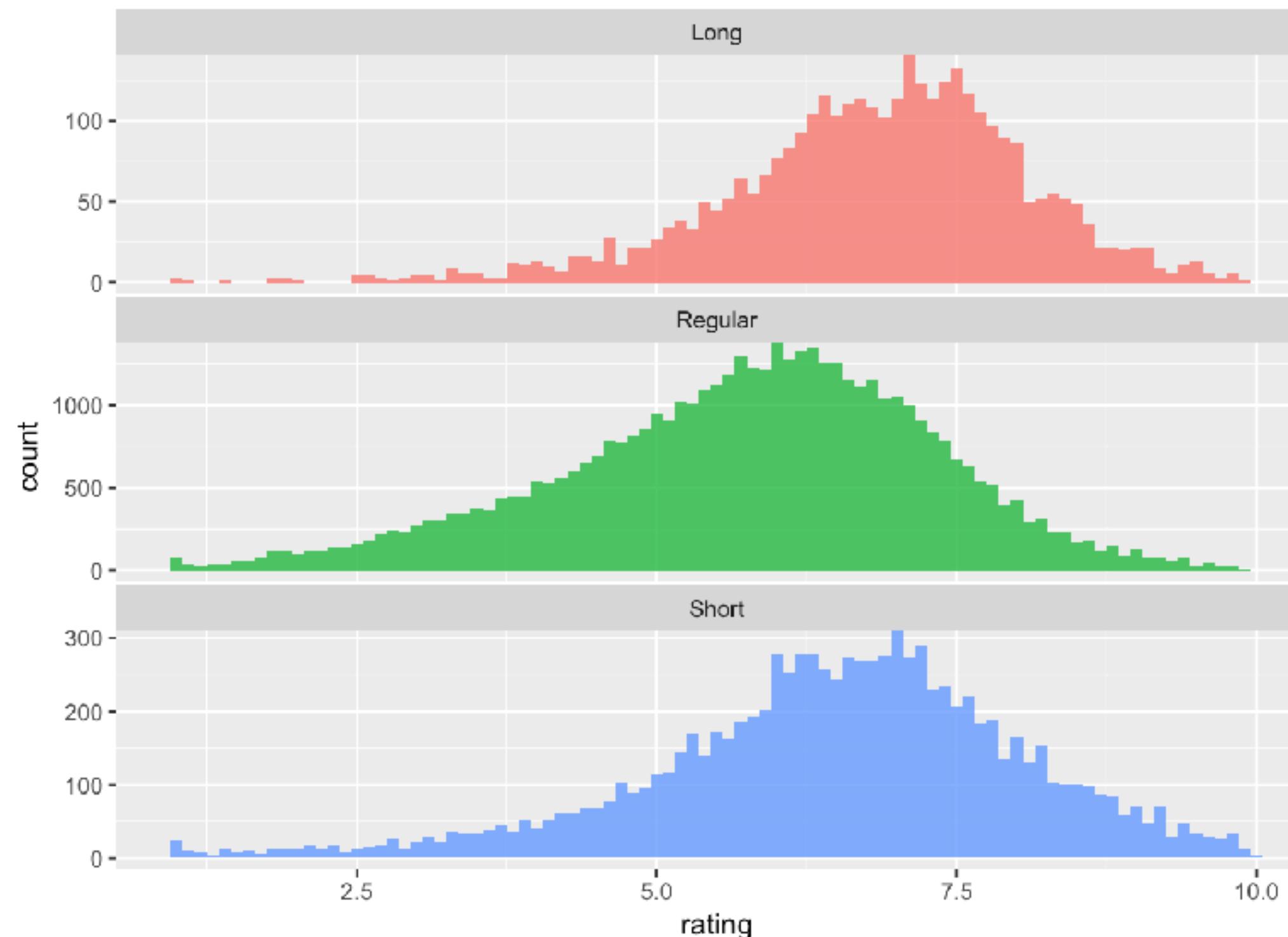
# How do ratings differ?

```
movies %>%  
  mutate(Description =  
    ifelse(length >= 122.5, "Long",  
          ifelse(length <= 40, "Short", "Regular")))) %>%  
  ggplot(aes(Description, rating)) +  
    geom_violin() +  
    geom_boxplot(width = 0.1)
```



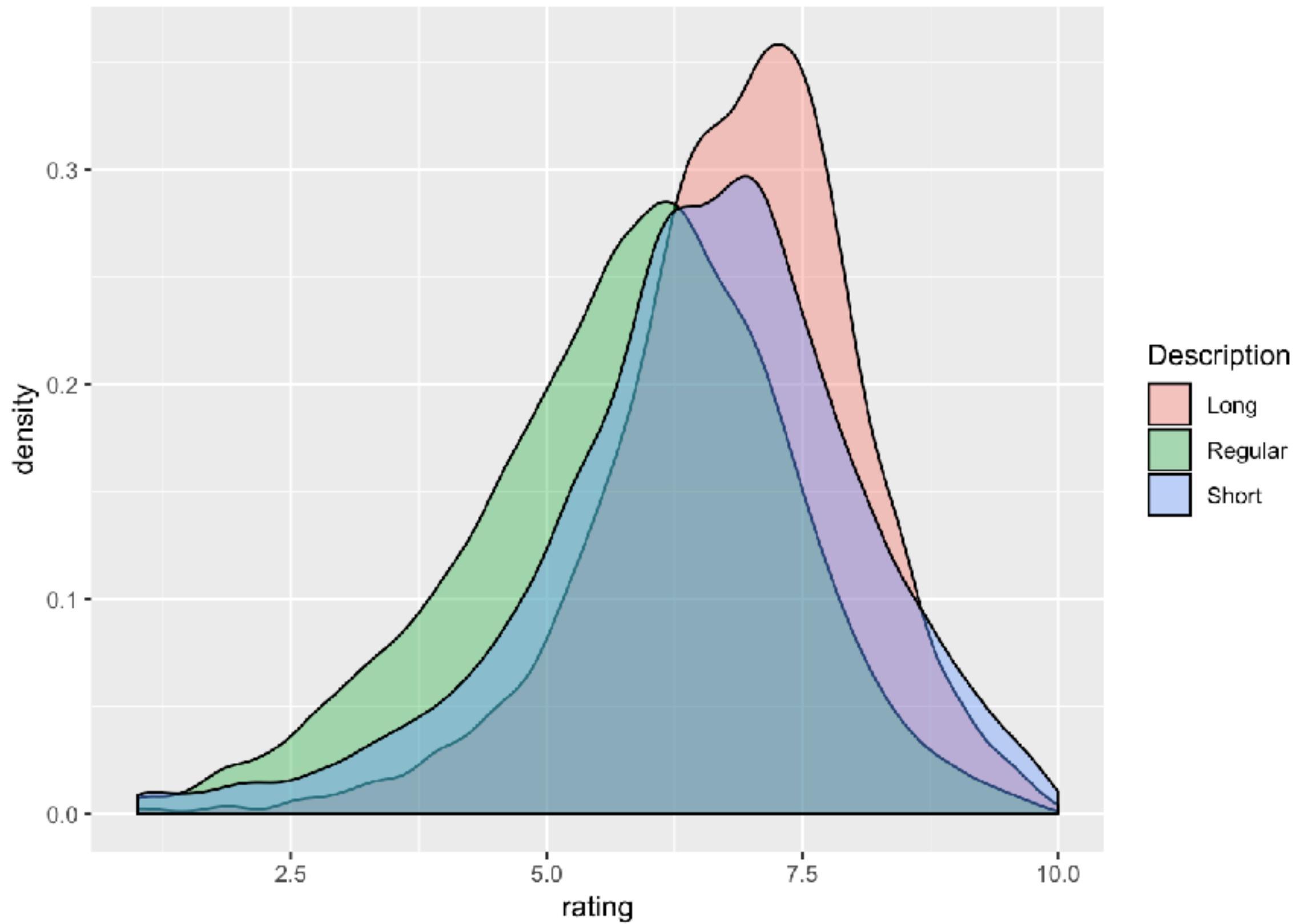
# How do ratings differ?

```
movies %>%
  mutate(Description =
    ifelse(length >= 122.5, "Long",
           ifelse(length <= 40, "Short", "Regular"))) %>%
  ggplot(aes(rating, fill = Description)) +
  geom_histogram(binwidth = .1, alpha = .8, show.legend = FALSE) +
  facet_wrap(~Description, ncol = 1, scales = "free_y")
```



# How do ratings differ?

```
movies %>%
  mutate(Description =
    ifelse(length >= 122.5, "Long",
          ifelse(length <= 40, "Short", "Regular"))) %>%
  ggplot(aes(rating, fill = Description)) +
  geom_density(alpha = 0.4)
```



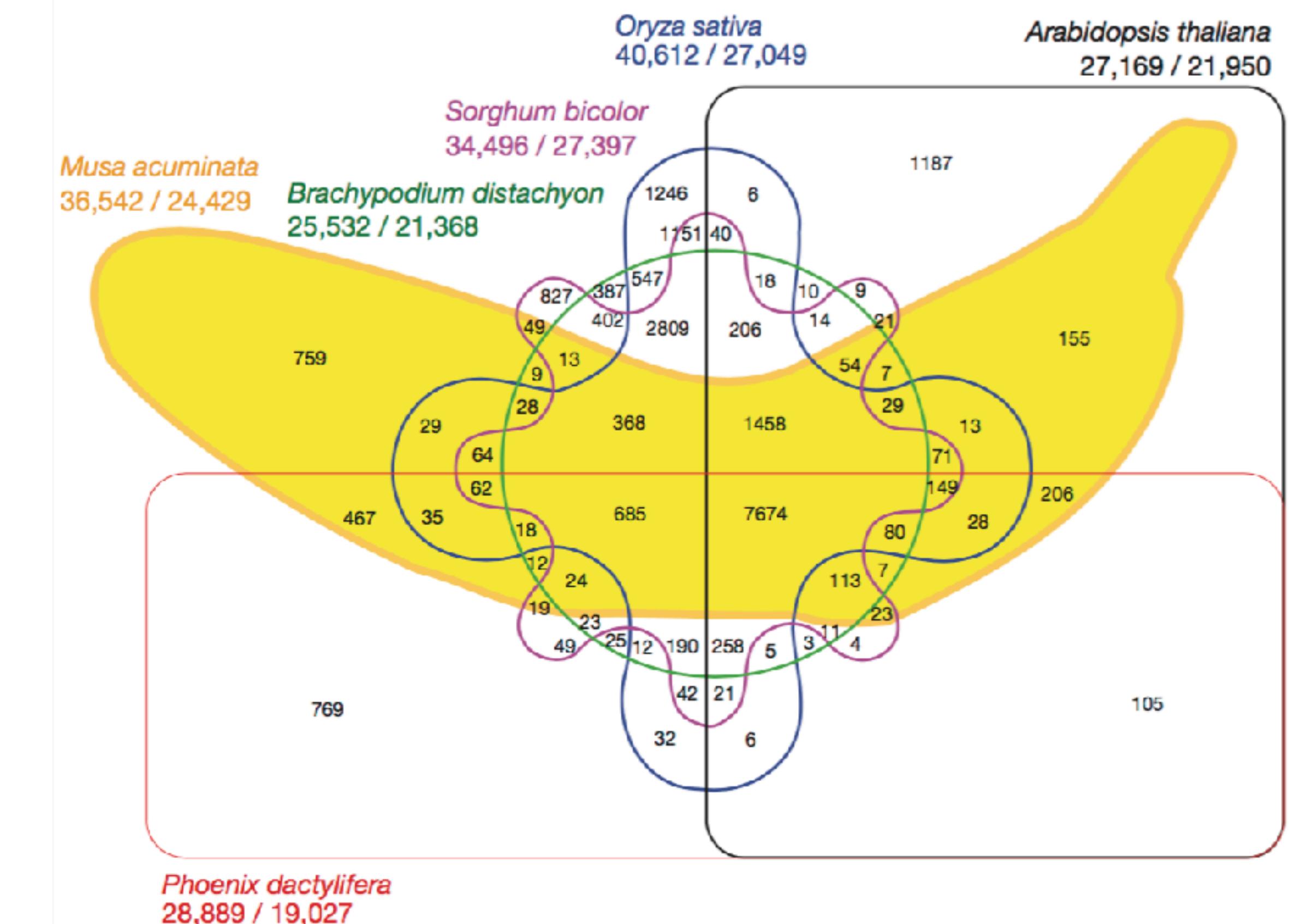
“Correlation ≠ Causation”

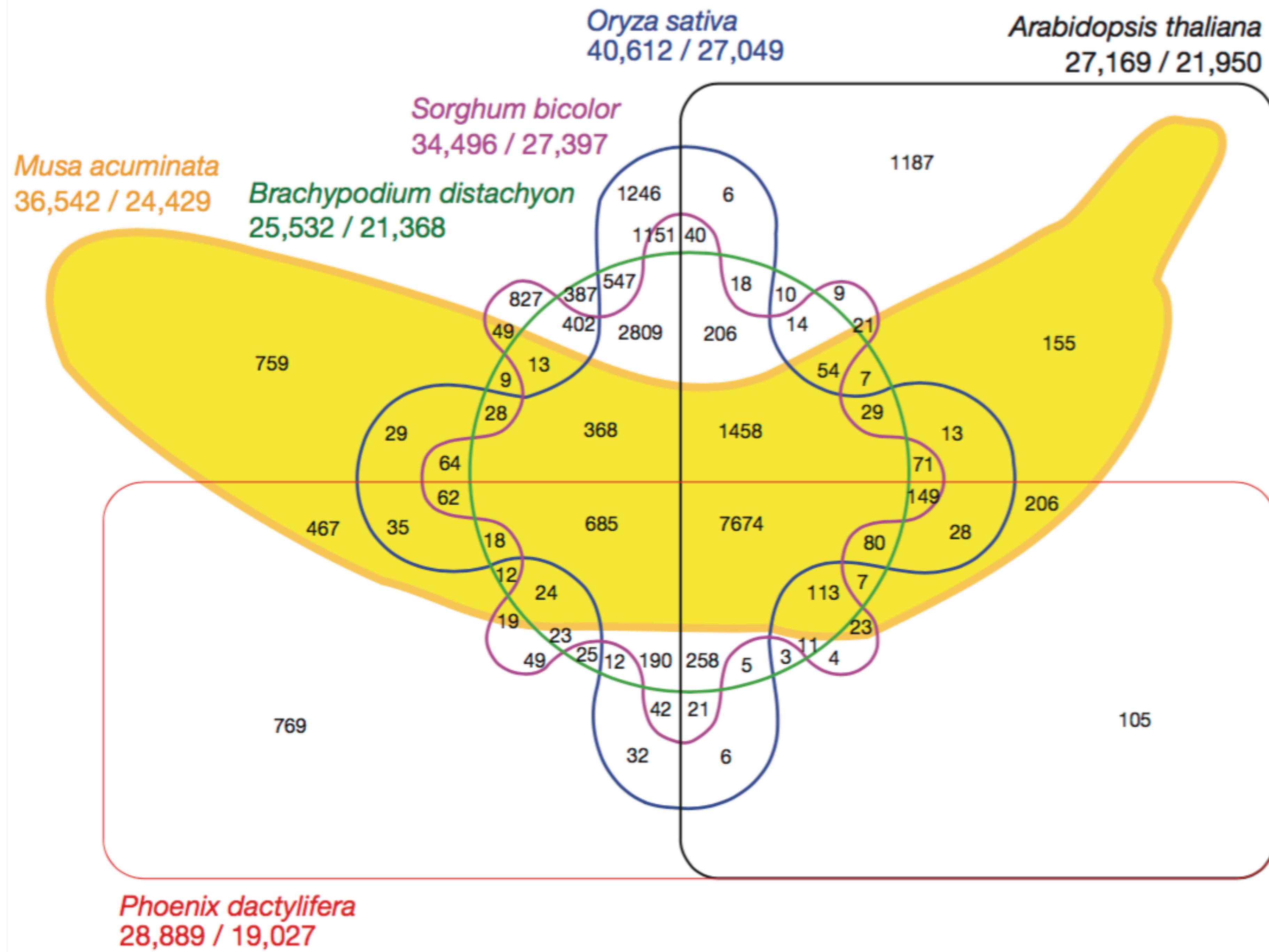
# UpSet diagram

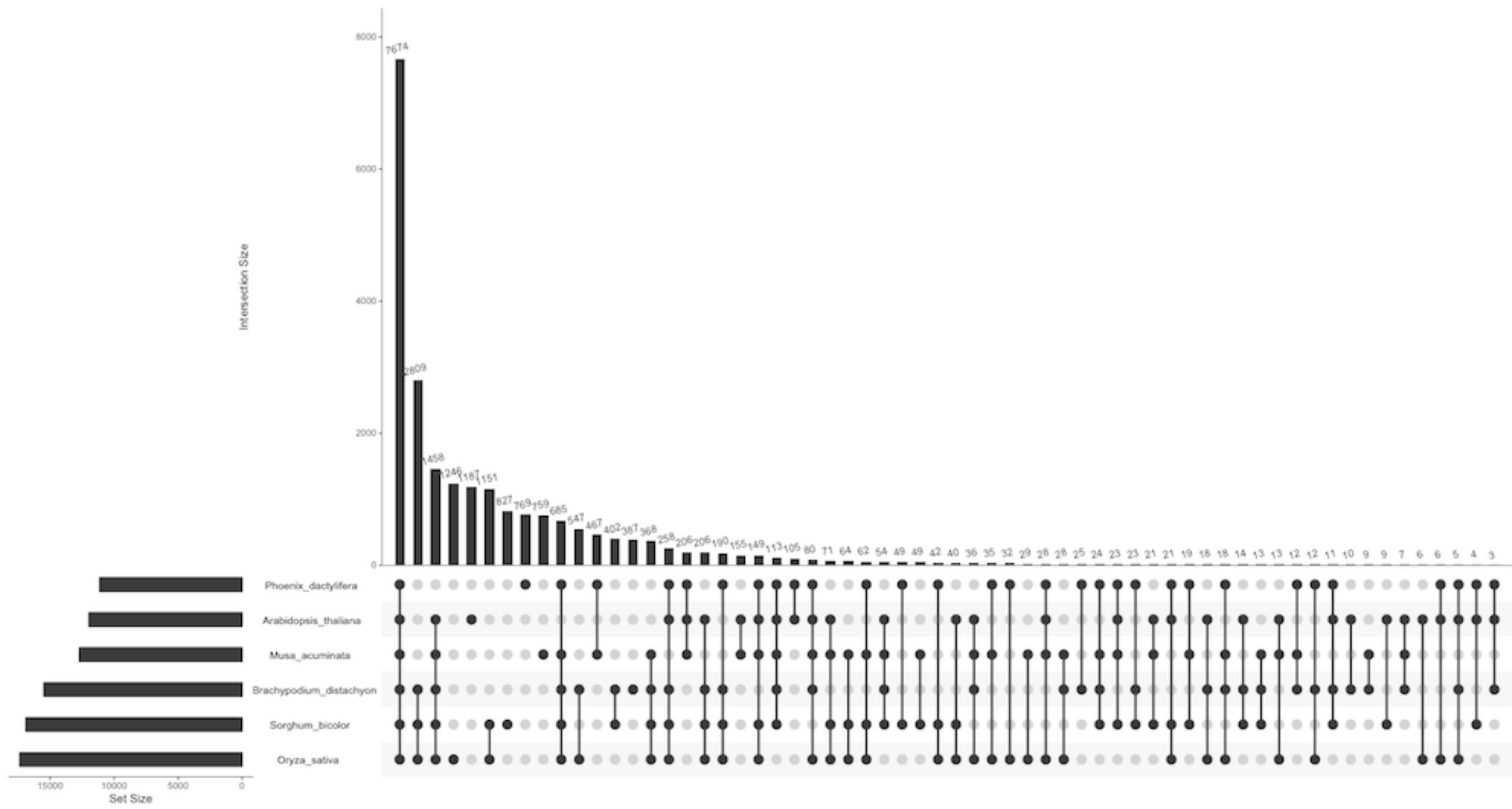
UpSetR

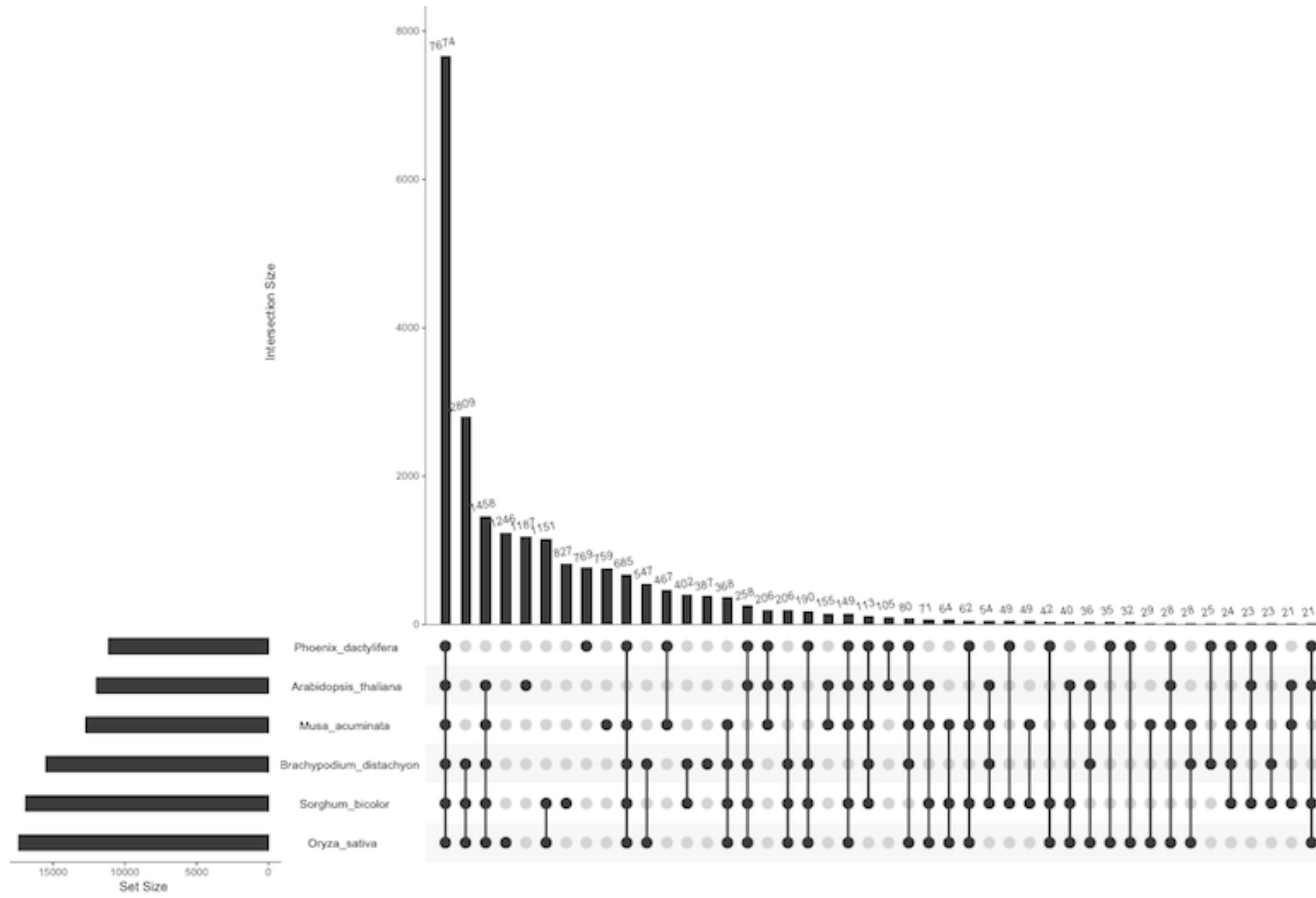
# UpSetR

- Visualise set intersections in a matrix layout
- Generates static UpSet plots
- Address limitations of Venn diagram





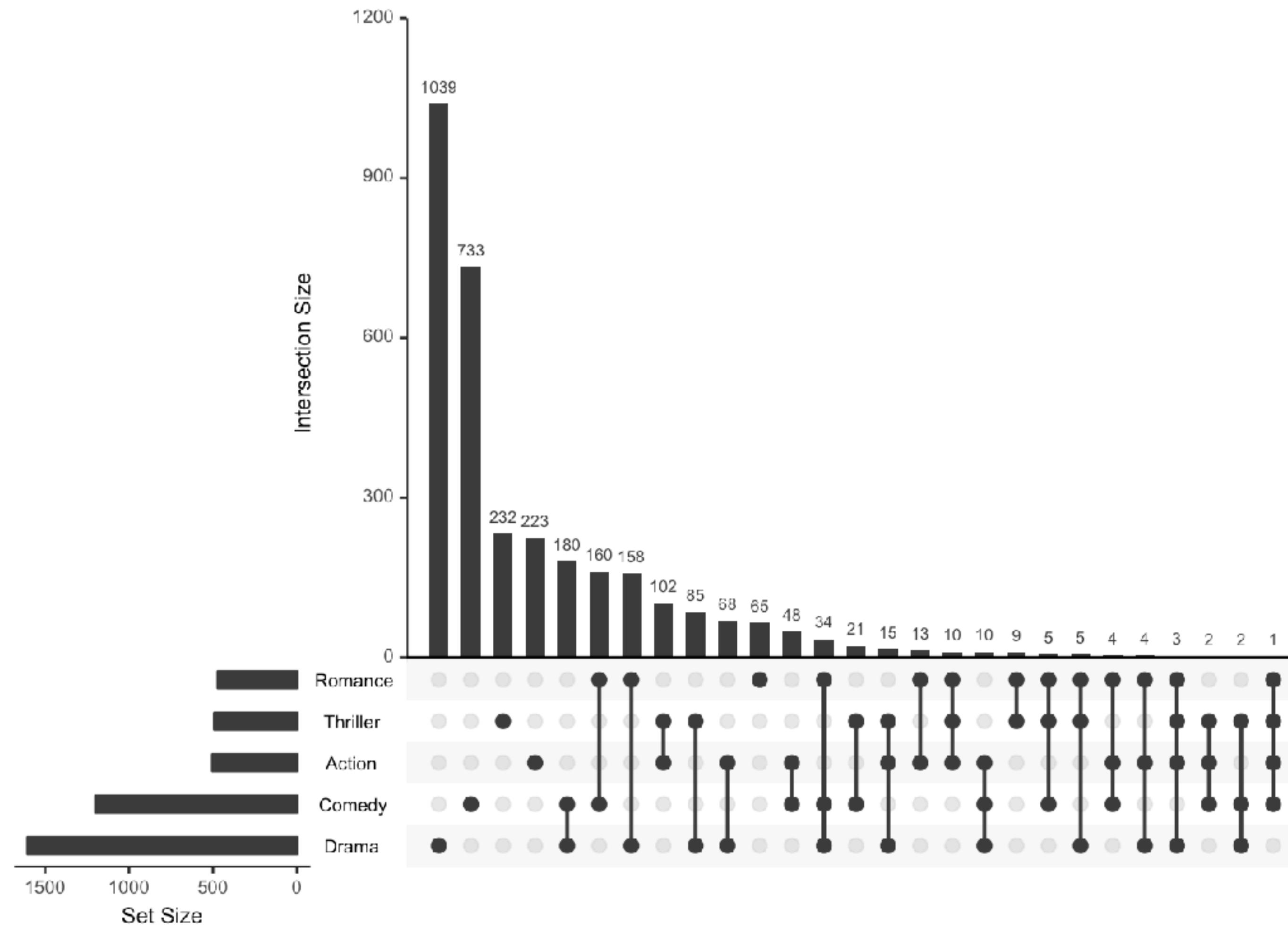




```

library(UpSetR)
# sample data
movies <- read.csv( system.file("extdata", "movies.csv", package = "UpSetR"), header=T, sep=";" )
upset(movies, order.by = "freq")

```



plotly in R

# plotly

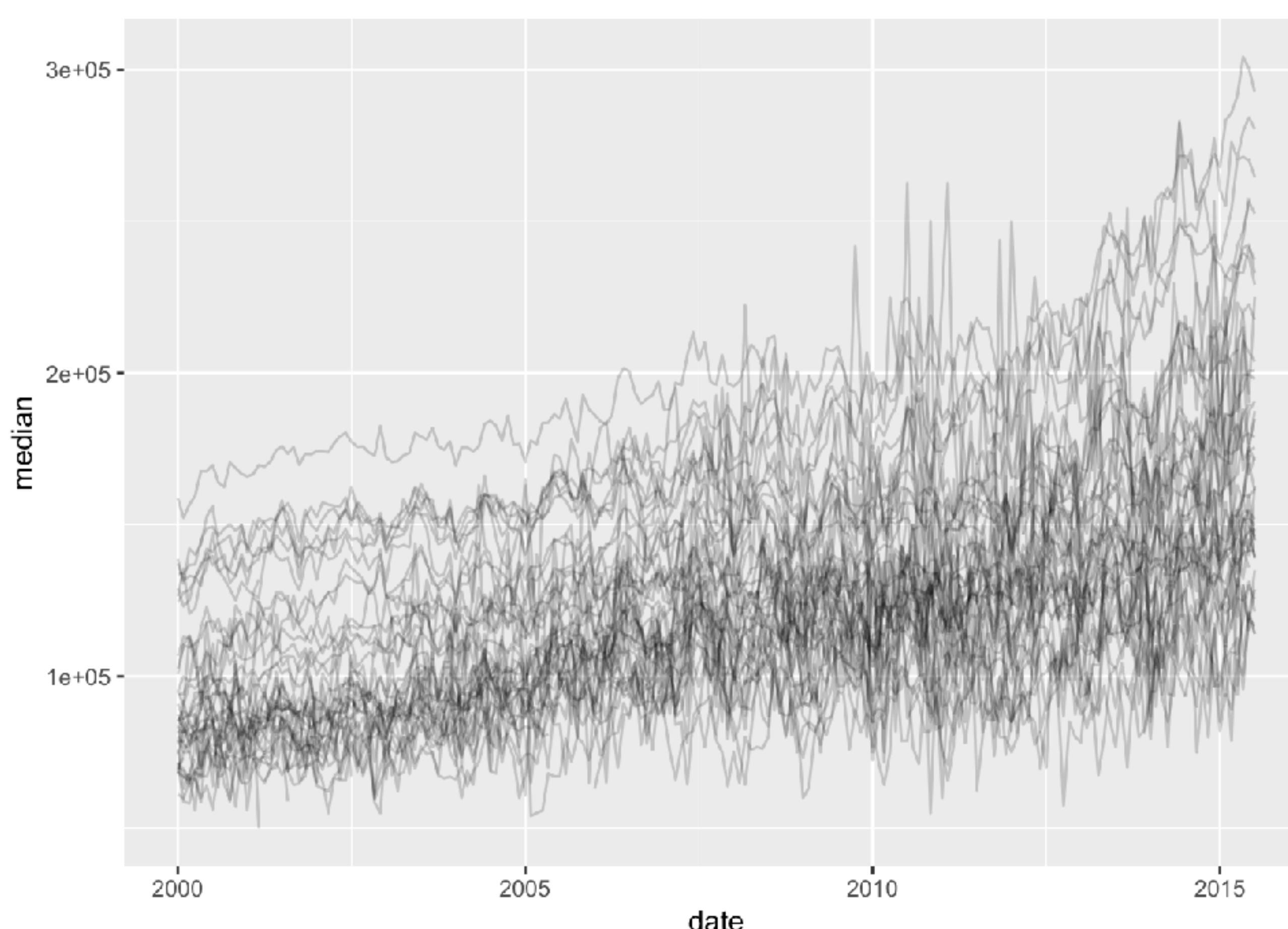
- creates interactive data visualisation
  - based on JavaScript graphing library plotly.js
  - compatible with ggplot2 and python
- plotly graphs are rendered locally through the htmlwidgets framework
- can publish your charts online via Plotly's web service
- work in R Markdown, shiny apps, inside RStudio offline
  - wrap plots with embed\_notebook() function to embed them inline inside a Jupyter notebook
- 2 approaches to generate plotly graphs
  1. ggplot2 extension
  2. plotly interface
- Reference: <https://plot.ly/r/reference/>

# 1) ggplotly()

- **ggplotly()** function converts a ggplot object to a plotly object
- extends ggplot2, majority of features but not all
  - Reference: <http://ropensci.github.io/plotly/ggplot2/>
- Texas housing data for demo

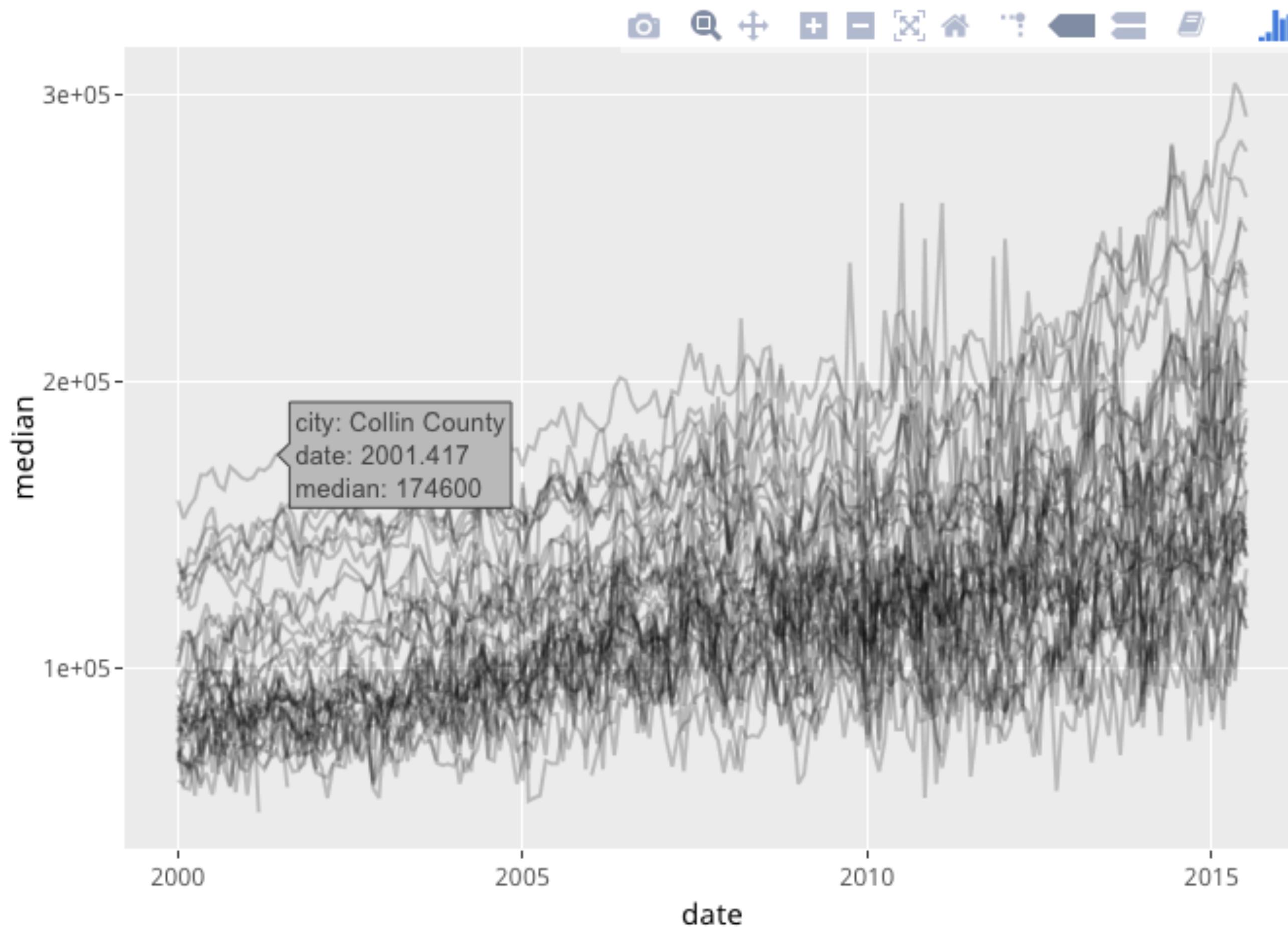
# Texas housing data

```
# Texas housing data, median house price over time  
ggplot(txhousing, aes(date, median)) +  
  geom_line(aes(group = city), alpha = 0.2, na.rm = T)
```



# Texas housing data

```
# ggplot object  
p <- ggplot(txhousing, aes(date, median)) +  
  geom_line(aes(group = city), alpha = 0.2, na.rm = T)  
ggplotly(p)
```



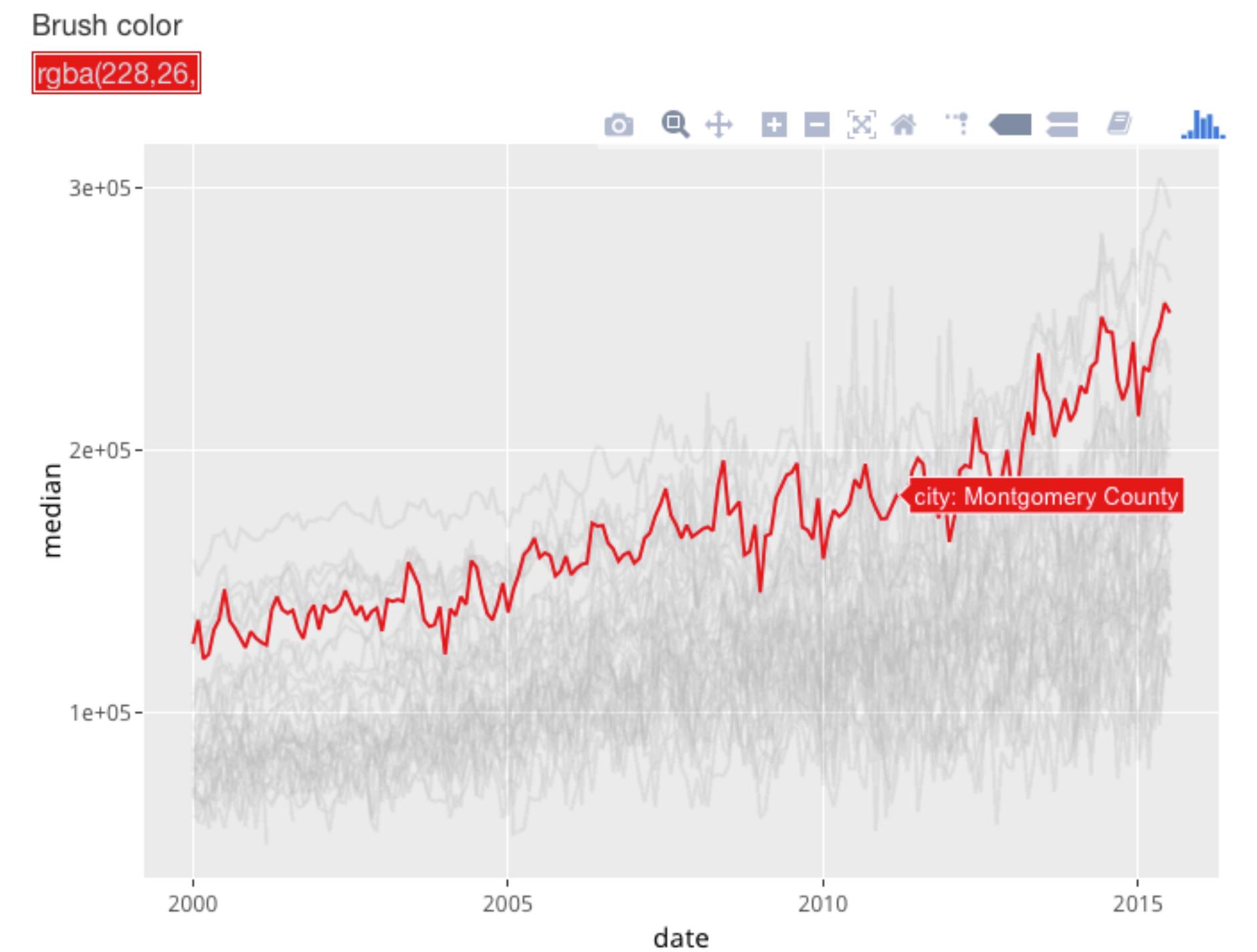
# Selection

```
# install.packages("crosstalk")
library(crosstalk)

# created SharedData class
d <- highlight_key(txhousing, ~city)

# create ggplot
p <- ggplot(d, aes(date, median)) +
  geom_line(aes(group = city), alpha = 0.2, na.rm = T)

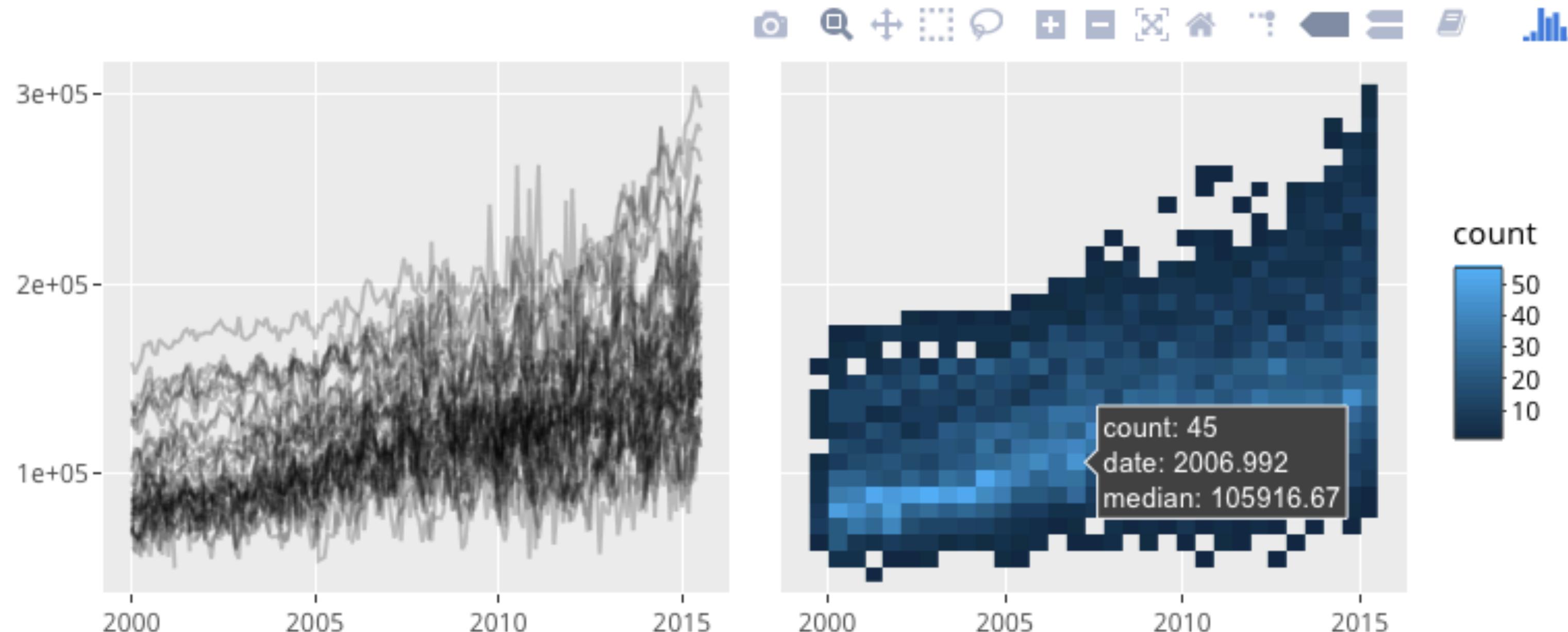
# create plotly object
ggplotly(p, tooltip = "city") %>%
  highlight(on = "plotly_click", dynamic = TRUE)
```



# subplot()

```
# subplot example
p <- ggplot(txhousing, aes(date, median)) +
  geom_line(aes(group = city), alpha = 0.2, na.rm = T)

subplot(
  ggplotly(p, tooltip = "city"), # restricting tooltip info
  ggplot(txhousing, aes(date, median)) + geom_bin2d(),
  nrows = 1, shareX = T, shareY = T, titleX = F, titleY = F
)
```



# 2) plot\_ly()

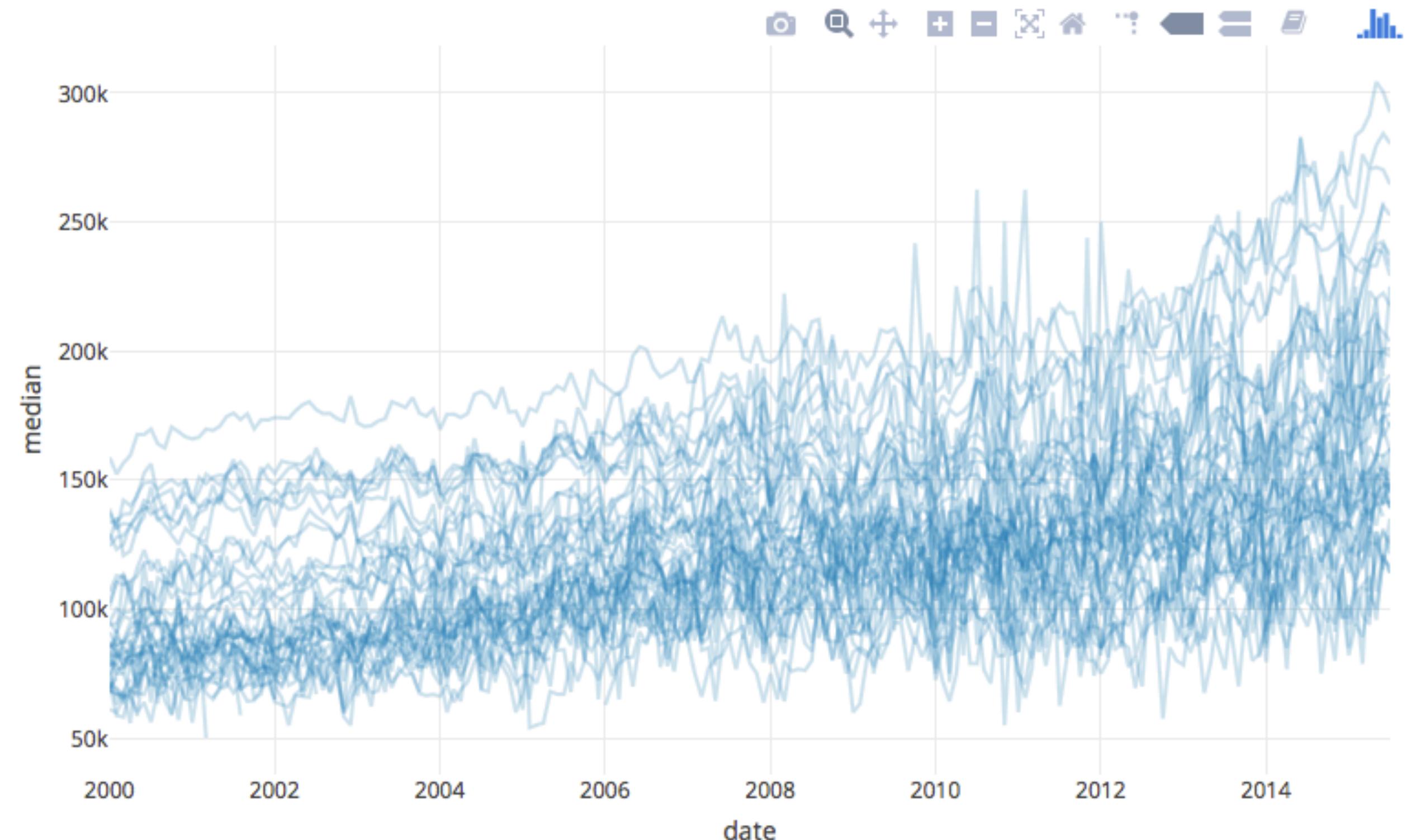
- more flexible and direct interface to plotly.js
- more flexible in the sense that data frames are not required
  - but use of data frame is highly recommended, especially when building up layers
- extends some functions from dplyr and tidyr packages
  - recognises dplyr's group\_by() function
- 3 key functions:
  - plot\_ly()
  - add\_trace()
  - layout()

# plot\_ly()

```
library(dplyr)

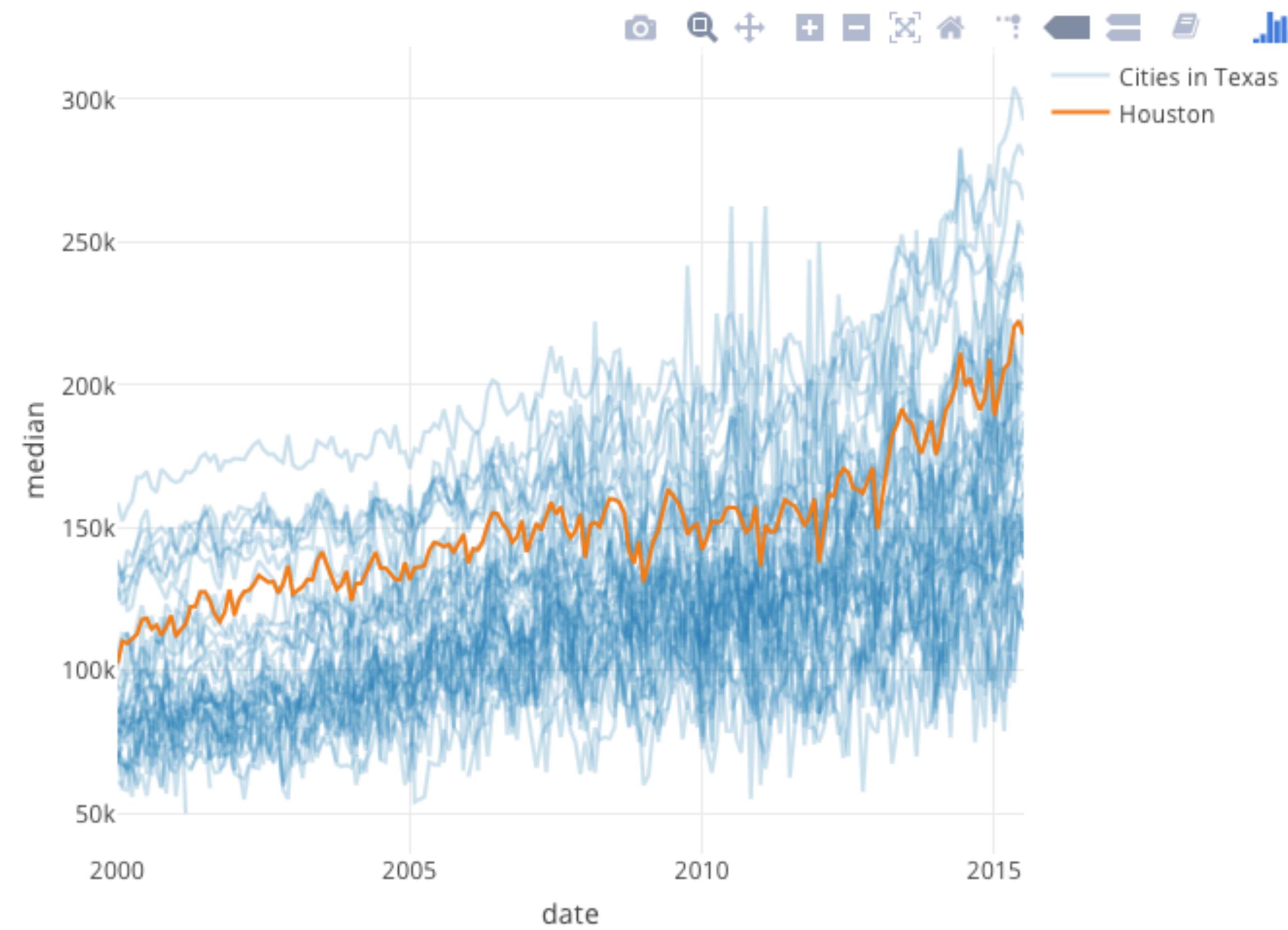
# group by city
tx <- txhousing %>%
  group_by(city)

# plot line
plot_ly(tx, x = ~date, y = ~median) %>%
  add_lines(alpha = 0.2, name = "Cities in Texas")
```



# layered grammar of graphics

```
# add highlight
plot_ly(tx, x = ~date, y = ~median) %>%
  add_lines(alpha = 0.2, name = "Cities in Texas") %>%
  add_lines(data = filter(tx, city == "Houston"), name = "Houston")
```



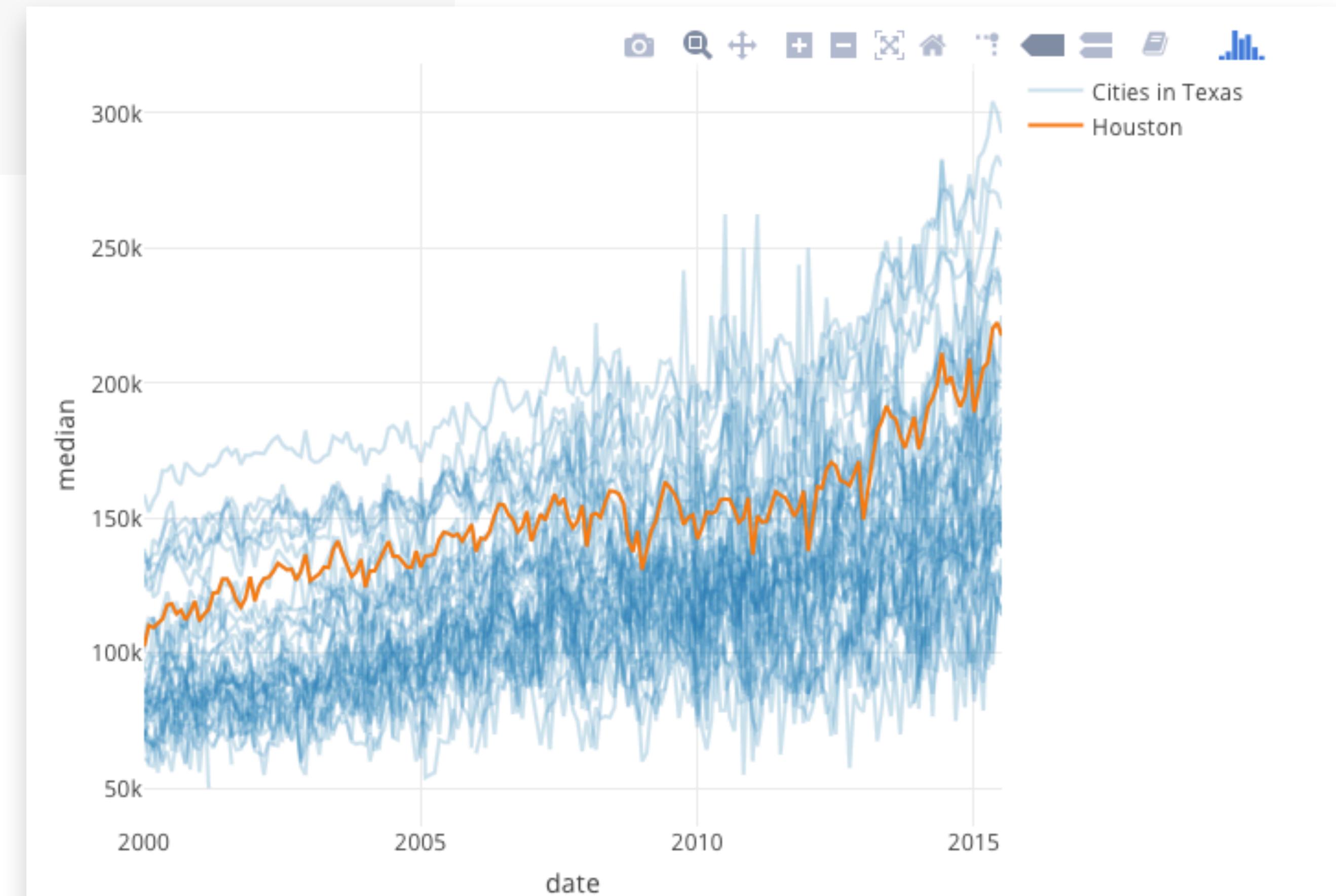
# add\_\*() functions

```
add_trace(p, ..., data = NULL, inherit = TRUE)
add_markers(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)
add_text(p, x = NULL, y = NULL, z = NULL, text = NULL, ..., data = NULL, inherit = TRUE)
add_paths(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)
add_lines(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)
add_segments(p, x = NULL, y = NULL, xend = NULL, yend = NULL, ..., data = NULL, inherit = TRUE)
add_polygons(p, x = NULL, y = NULL, ..., data = NULL, inherit = TRUE)
add_sf(p, ..., x = ~x, y = ~y, data = NULL, inherit = TRUE)
add_table(p, ..., rownames = TRUE, data = NULL, inherit = TRUE)
add_ribbons(p, x = NULL, ymin = NULL, ymax = NULL, ..., data = NULL, inherit = TRUE)
add_area(p, r = NULL, t = NULL, ..., data = NULL, inherit = TRUE)
add_pie(p, values = NULL, labels = NULL, ..., data = NULL, inherit = TRUE)
add_bars(p, x = NULL, y = NULL, ..., data = NULL, inherit = TRUE)
add_histogram(p, x = NULL, y = NULL, ..., data = NULL, inherit = TRUE)
add_histogram2d(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)
add_histogram2dcontour(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)
add_heatmap(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)
add_contour(p, z = NULL, ..., data = NULL, inherit = TRUE)
add_boxplot(p, x = NULL, y = NULL, ..., data = NULL, inherit = TRUE)
add_surface(p, z = NULL, ..., data = NULL, inherit = TRUE)
add_mesh(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)
add_scattergeo(p, ...)
add_choropleth(p, z = NULL, ..., data = NULL, inherit = TRUE)
```

# chaining plotly object

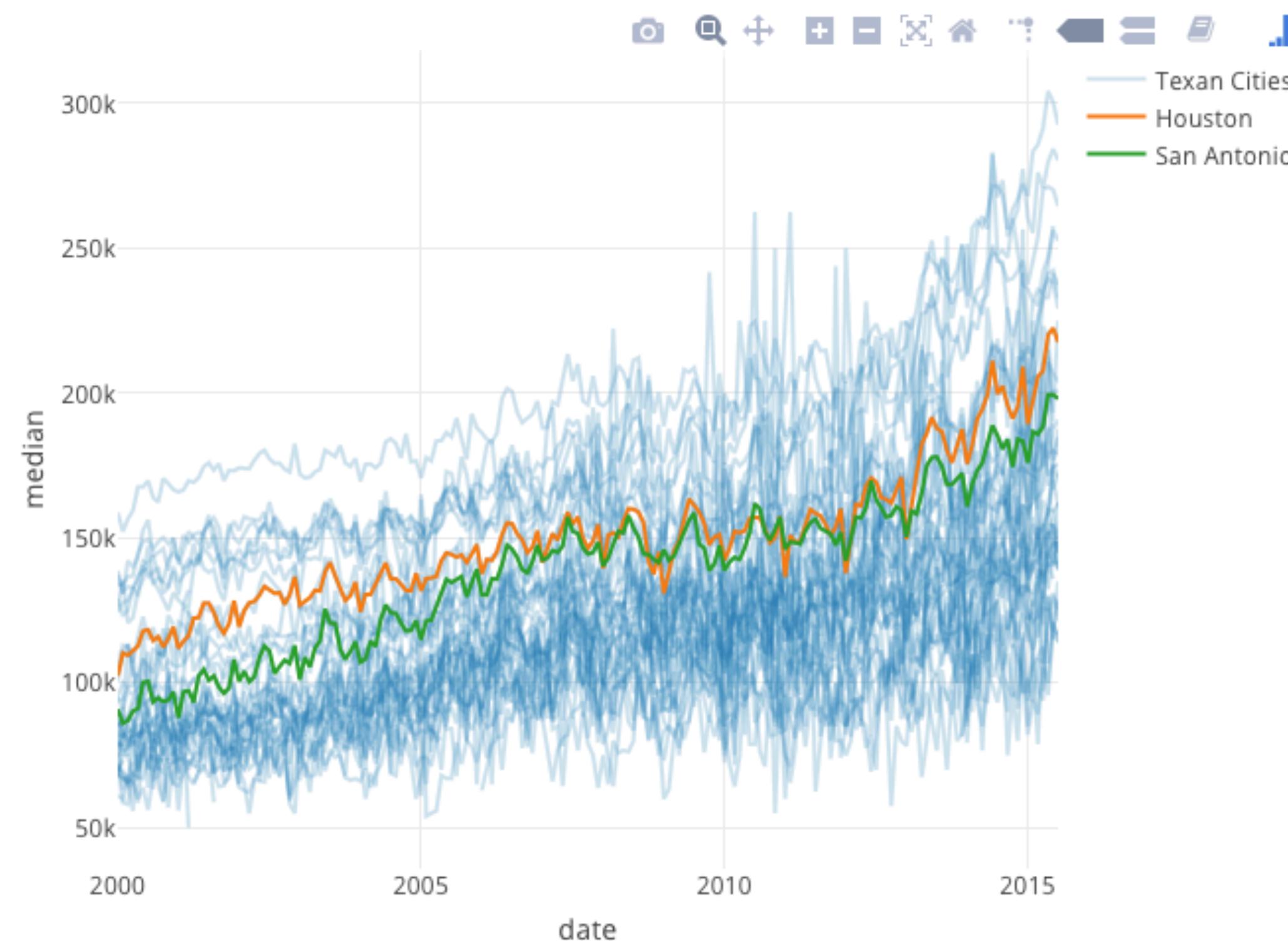
```
allCities <- txhousing %>%
  group_by(city) %>%
  plot_ly(x = ~date, y = ~median) %>%
  add_lines(alpha = 0.2, name = "Texan Cities", hoverinfo = "none")
```

```
allCities %>%
  filter(city == "Houston") %>%
  add_lines(name = "Houston")
```



# recovering the original data

```
allCities %>%  
  add_fun(function(plot) {  
    plot %>% filter(city == "Houston") %>% add_lines(name = "Houston")  
  }) %>%  
  add_fun(function(plot) {  
    plot %>% filter(city == "San Antonio") %>% add_lines(name = "San Antonio")  
  })
```



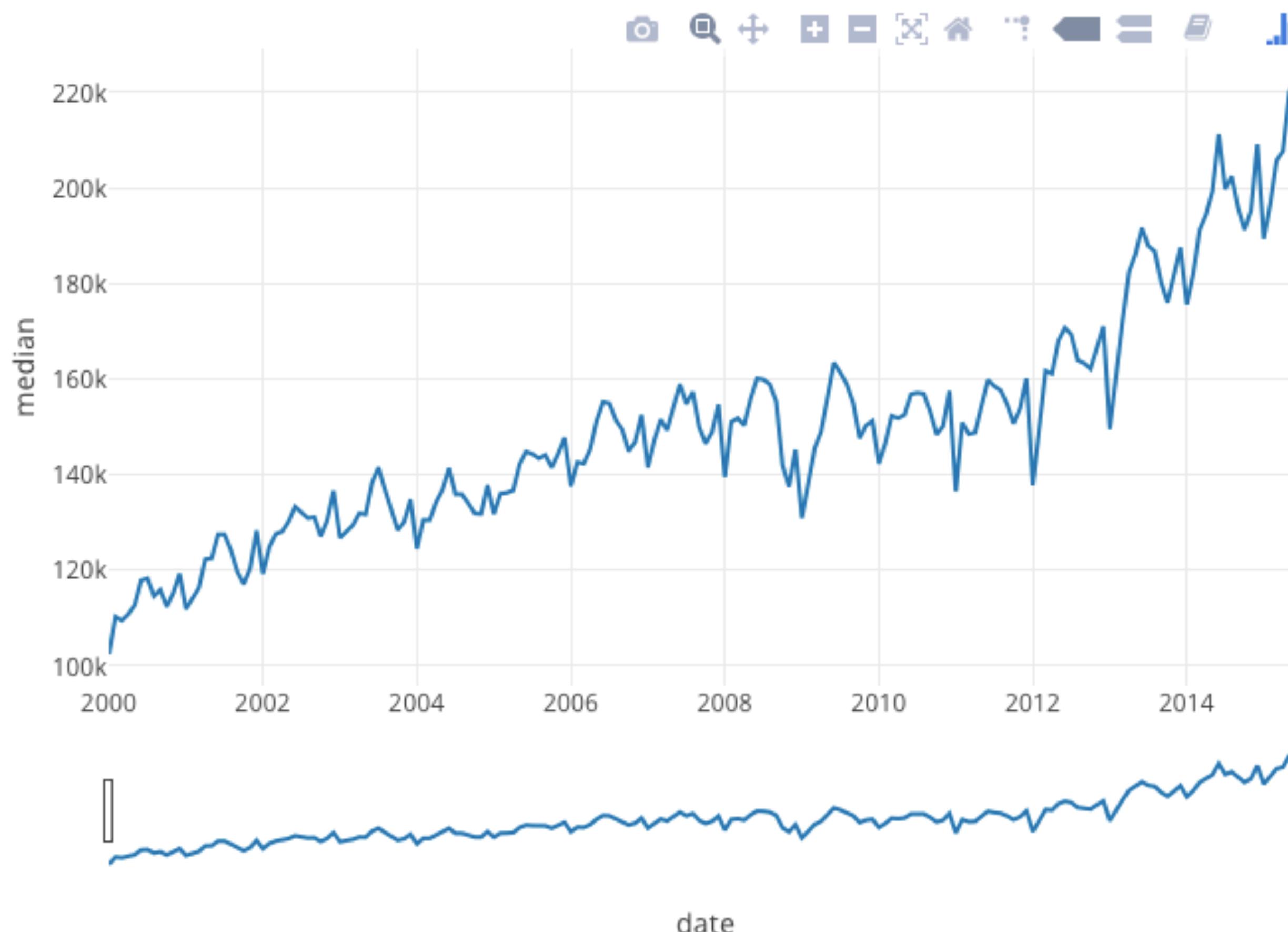
# recovering the original data

```
allCities %>%  
  add_fun(function(plot) {  
    plot %>% filter(city == "Houston") %>% add_lines(name = "Houston")  
  }) %>%  
  add_fun(function(plot) {  
    plot %>% filter(city == "San Antonio") %>% add_lines(name = "San Antonio")  
  })
```

```
# create a function  
layer_city <- function(plot, name) {  
  plot %>% filter(city == name) %>% add_lines(name = name)  
}  
  
allCities %>%  
  add_fun(layer_city, "Houston") %>%  
  add_fun(layer_city, "San Antonio")
```

# rangeslider()

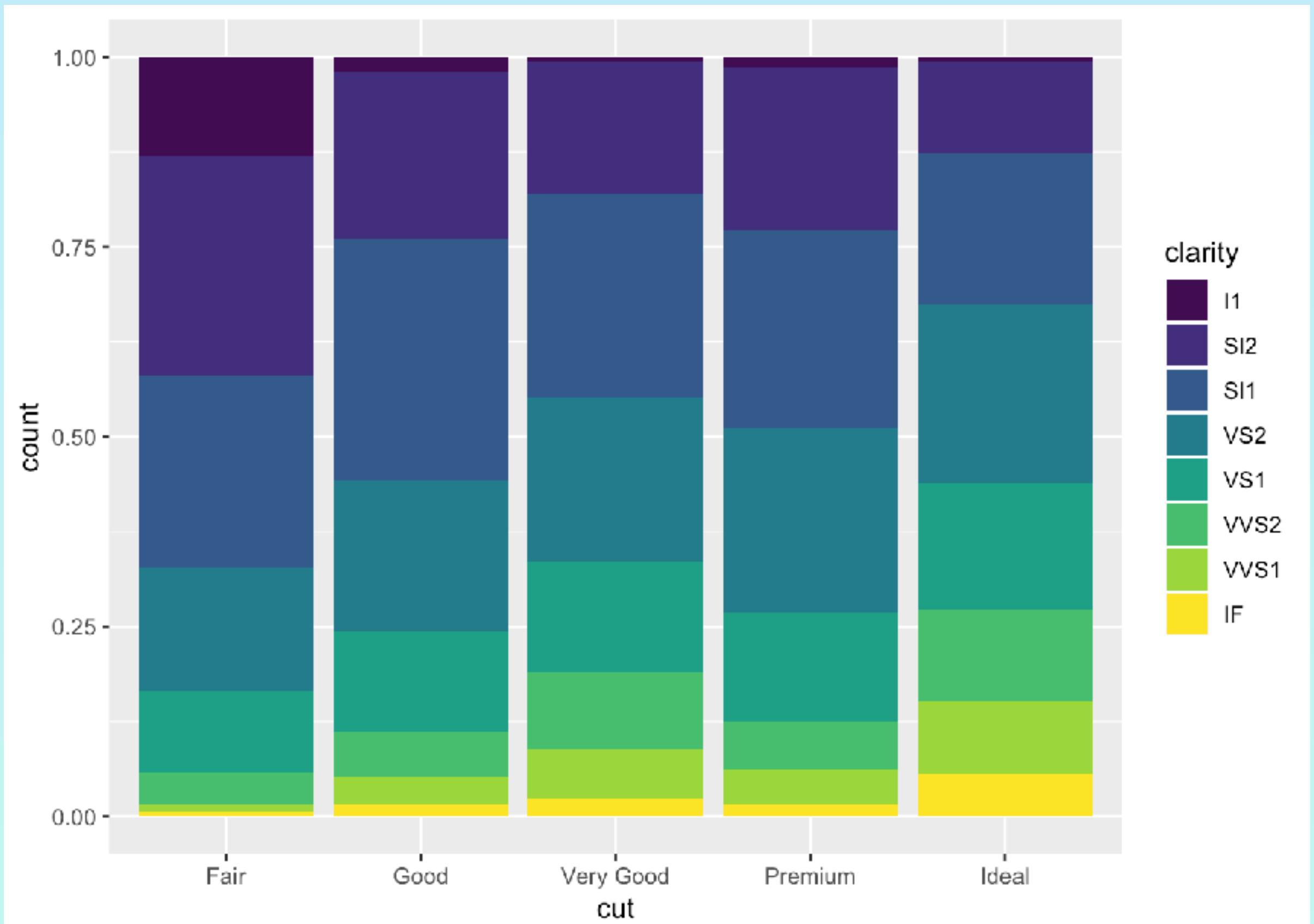
```
houston <- plot_ly(txhousing %>% filter(city == "Houston"), x = ~date, y = ~median) %>%  
  add_lines()  
rangeslider(houston)
```



# Your turn!

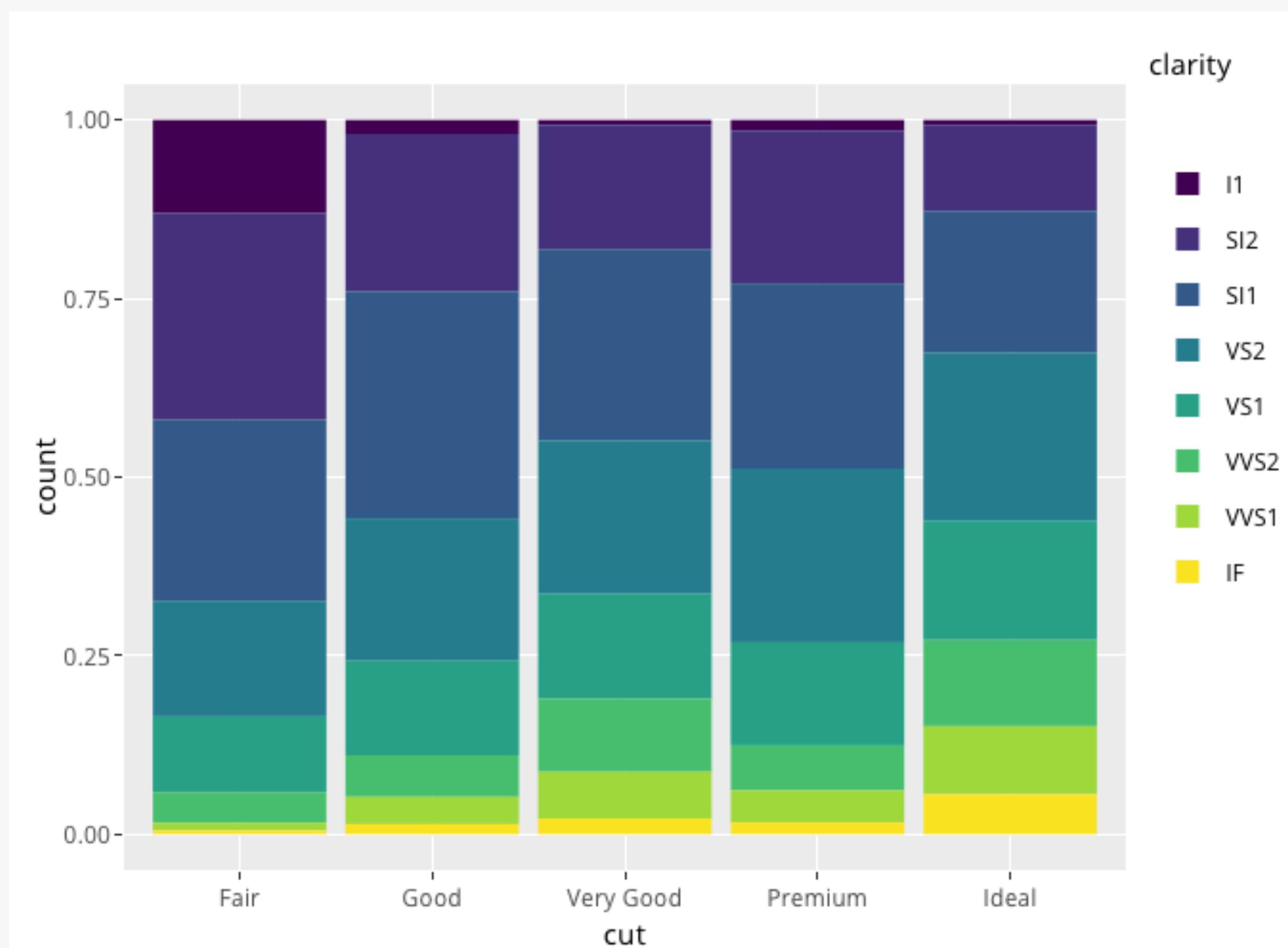
- In a previous lecture, you created this visualisation with diamonds data.  
Create a plotly version of this figure.

```
ggplot(diamonds, aes(cut, fill = clarity)) +  
  geom_bar(position = "fill")
```



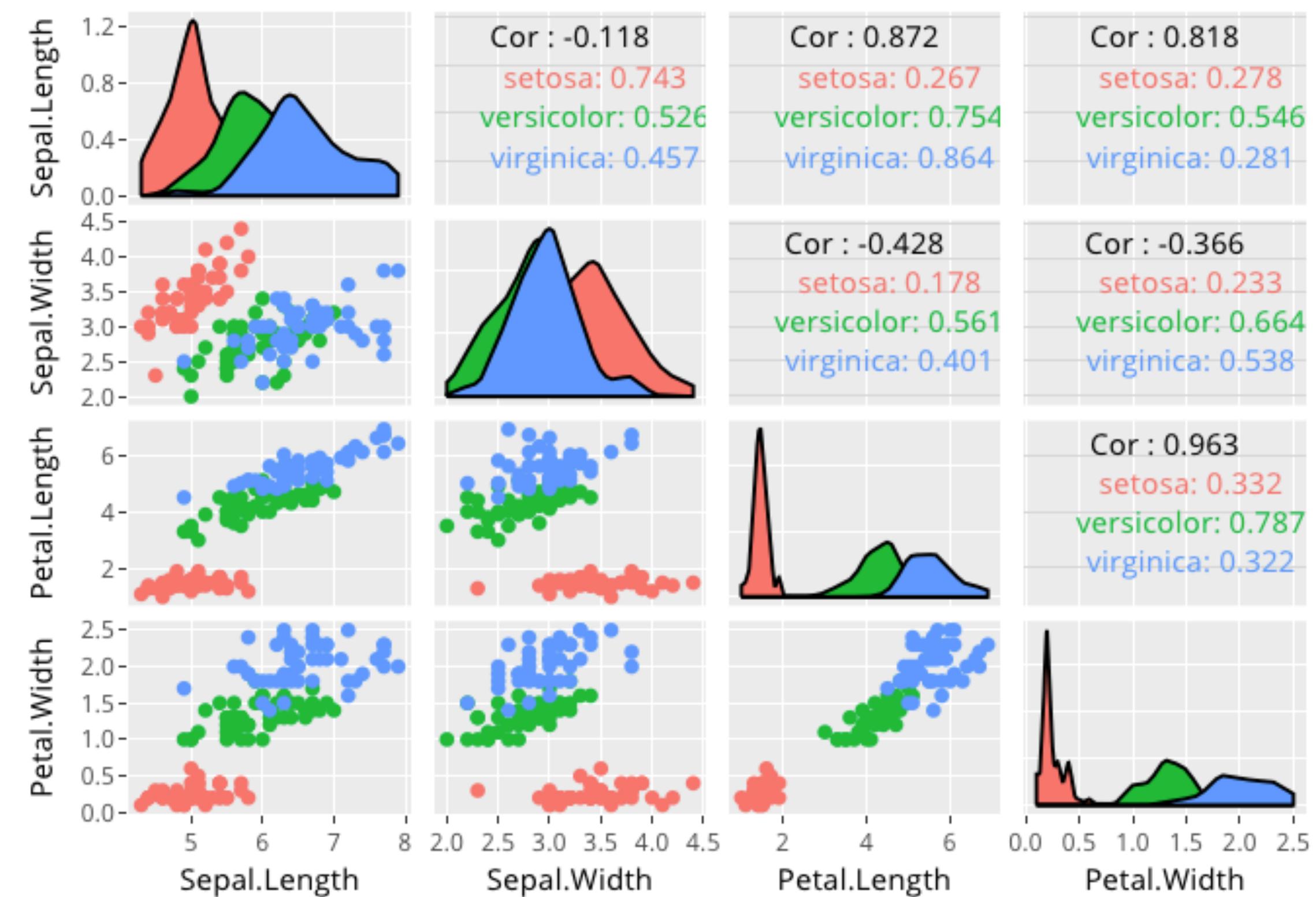
# Solution

```
p <- ggplot(diamonds, aes(cut, fill = clarity)) +  
  geom_bar(position = "fill")  
  
ggplotly(p)
```



# Scatter plot matrix (SPLOM)

```
# install.packages("GGally")  
  
# create a SharedData object  
d <- SharedData$new(iris)  
# draw scatter plot matrix using GGally::ppairs  
p <- GGally::ggpairs(d, aes(color = Species), columns = 1:4)  
# pass to ggplotly  
highlight(ggplotly(p), on = "plotly_selected")
```



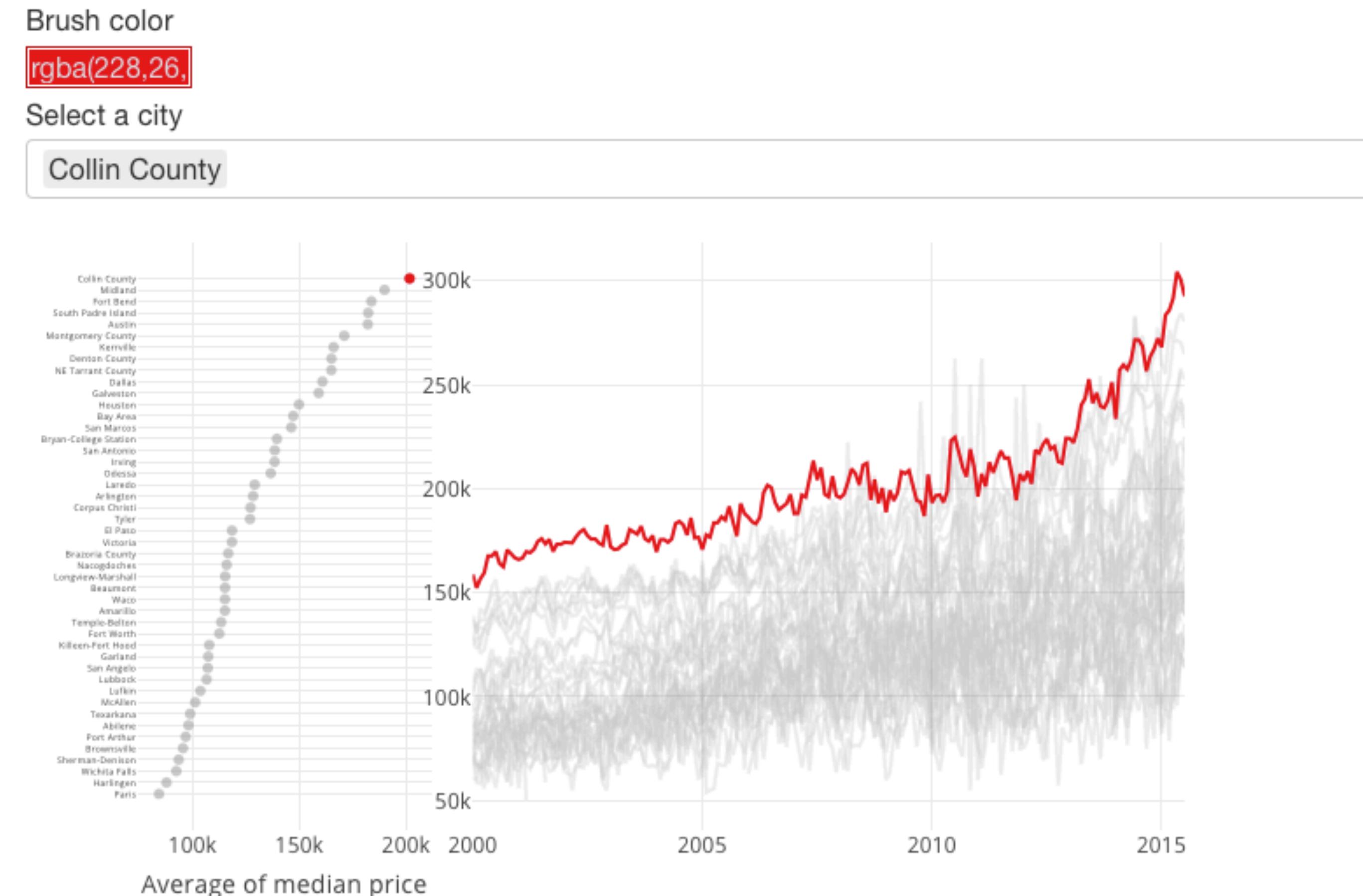
# Linked views

```
# SharedData and city as the shared key
sd <- highlight_key(txhousing, ~city, "Select a city")

# Create plotly object
base <- plot_ly(sd, color = I("black"), height = 400) %>%
  group_by(city)

# Cleveland plot
p1 <- base %>%
  summarise(mean_price = mean(median, na.rm = T)) %>%
  add_markers(x = ~mean_price,
               y = ~forcats::fct_reorder(city, mean_price),
               hoverinfo = "x+y") %>%
  layout(
    barmode = "overlay",
    xaxis = list(title = "Average of median price"),
    yaxis = list(title = "", tickfont = list(size = 5))
  )
# Line graph
p2 <- base %>%
  add_lines(x = ~date, y = ~median, alpha = 0.3) %>%
  layout(xaxis = list(title = ""))

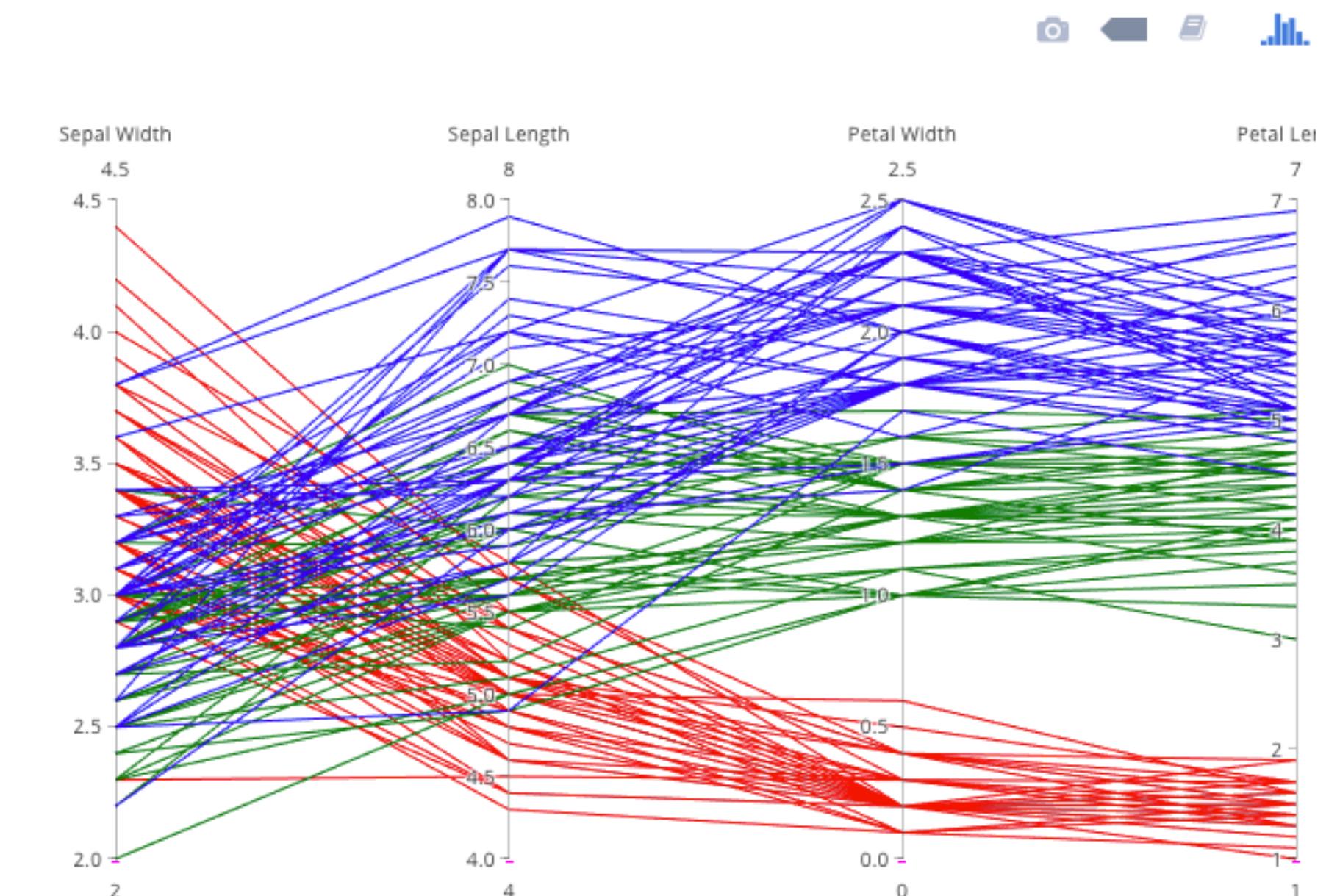
subplot(p1, p2, titleX = TRUE, widths = c(0.3, 0.7)) %>%
  hide_legend() %>%
  highlight(dynamic = TRUE, selectize = TRUE)
```



# Parallel coordinates

```
df <- datasets::iris %>%
  # numerical array is required for colour scale
  mutate(species_id = as.integer(Species))

plot_ly(data = df,
        type = 'parcoords',
        line = list(color = ~species_id,
                    colorscale = list(c(0,'red'),c(0.5,'green'),c(1,'blue'))),
        dimensions = list(
          list(range = c(2,4.5),label = 'Sepal Width', values = ~Sepal.Width),
          list(range = c(4,8),label = 'Sepal Length', values = ~Sepal.Length),
          list(range = c(0,2.5), label = 'Petal Width', values = ~Petal.Width),
          list(range = c(1,7), label = 'Petal Length', values = ~Petal.Length)
        )
      )
```



# Check out other charts

The screenshot shows the homepage of the Plotly R Open Source Graphing Library. The top navigation bar includes links for SIGN IN, SIGN UP, and REQUEST DEMO. Below the navigation is a search bar labeled "Search Plotly's R & ggplot2". The main content area features sections for "Plotly Fundamentals" and "Basic Charts".

**Plotly Fundamentals**

- Dashboard
- Updating Plotly Graphs
- Static Image Export
- Embedding Graphs with Knitr
- More Plotly Fundamentals

**Basic Charts**

- Scatter and Line Plots
- Line Plots
- Bar Charts
- Pie Charts
- More Basic Charts

<https://plot.ly/r/>

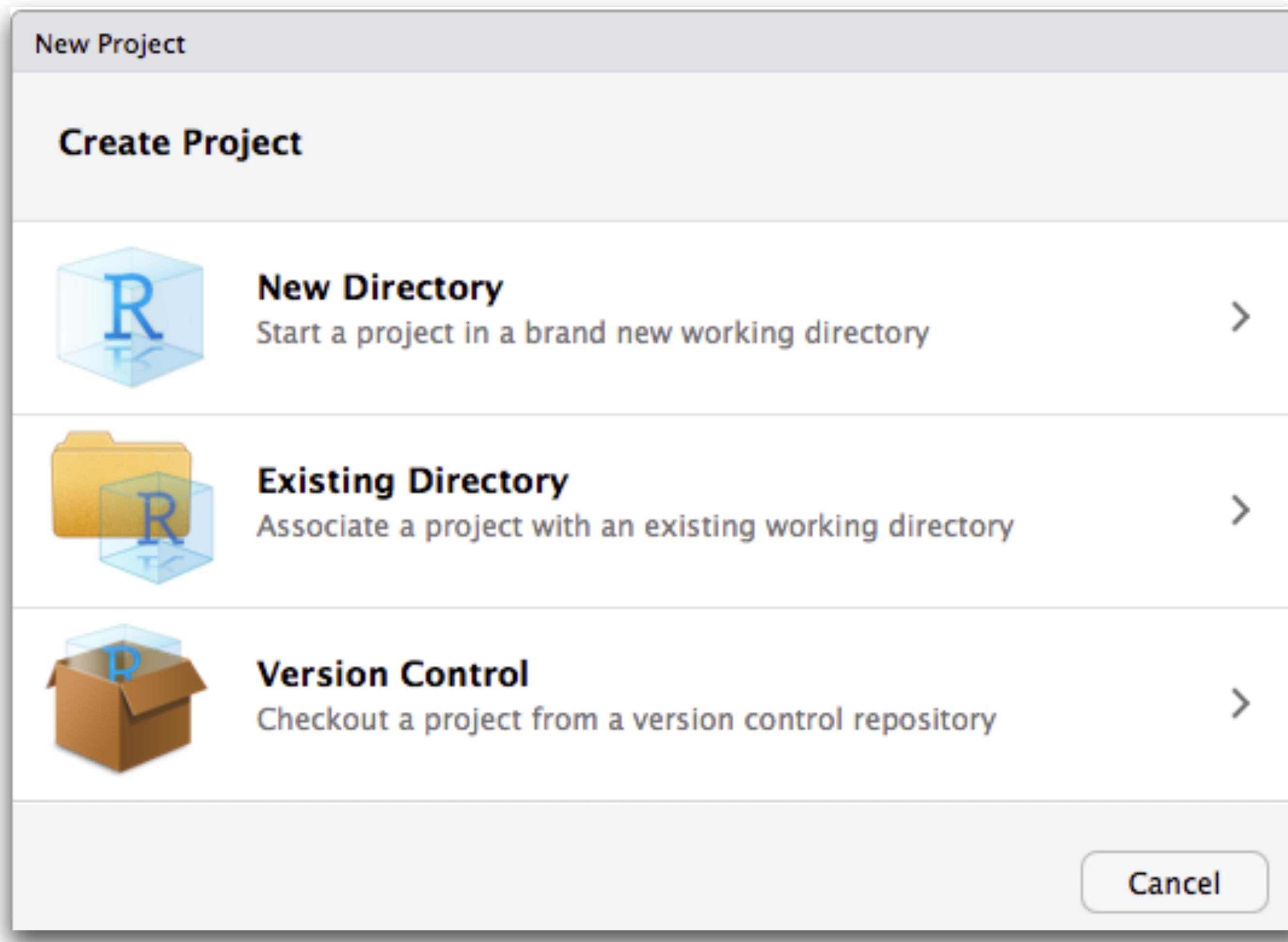
# Shiny

web application platform

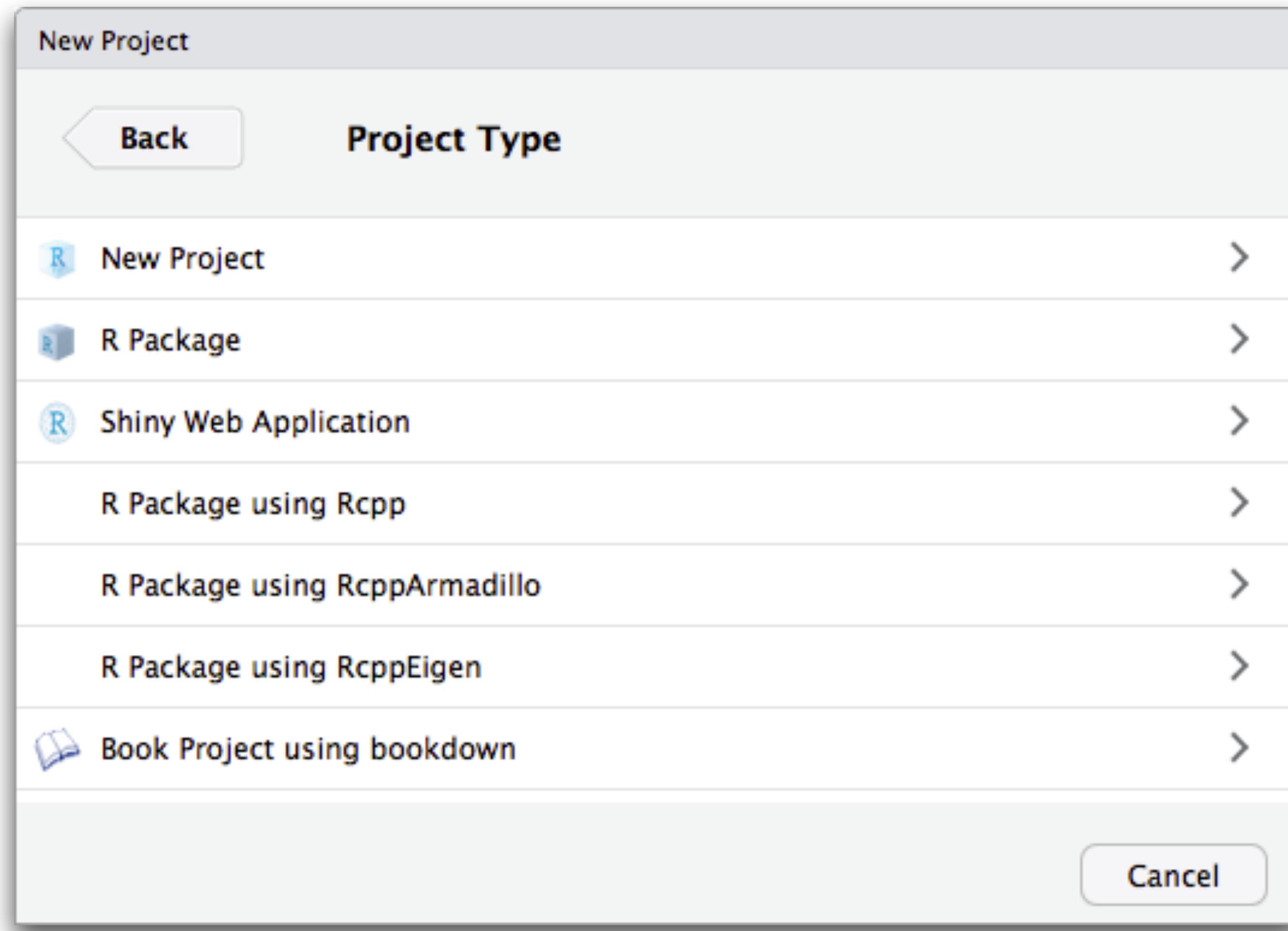
# Shiny

- package providing a web application framework
- Shiny apps
  - display R objects (plots, tables)
  - provide access to others
  - 2 components:
    - **UI** : user interface, taking the input from the user and dynamically displaying the generated output
    - **Server**: series of steps to convert the input given by the user into the designed output to be displayed
- no knowledge of HTML, CSS or JavaScript to get started

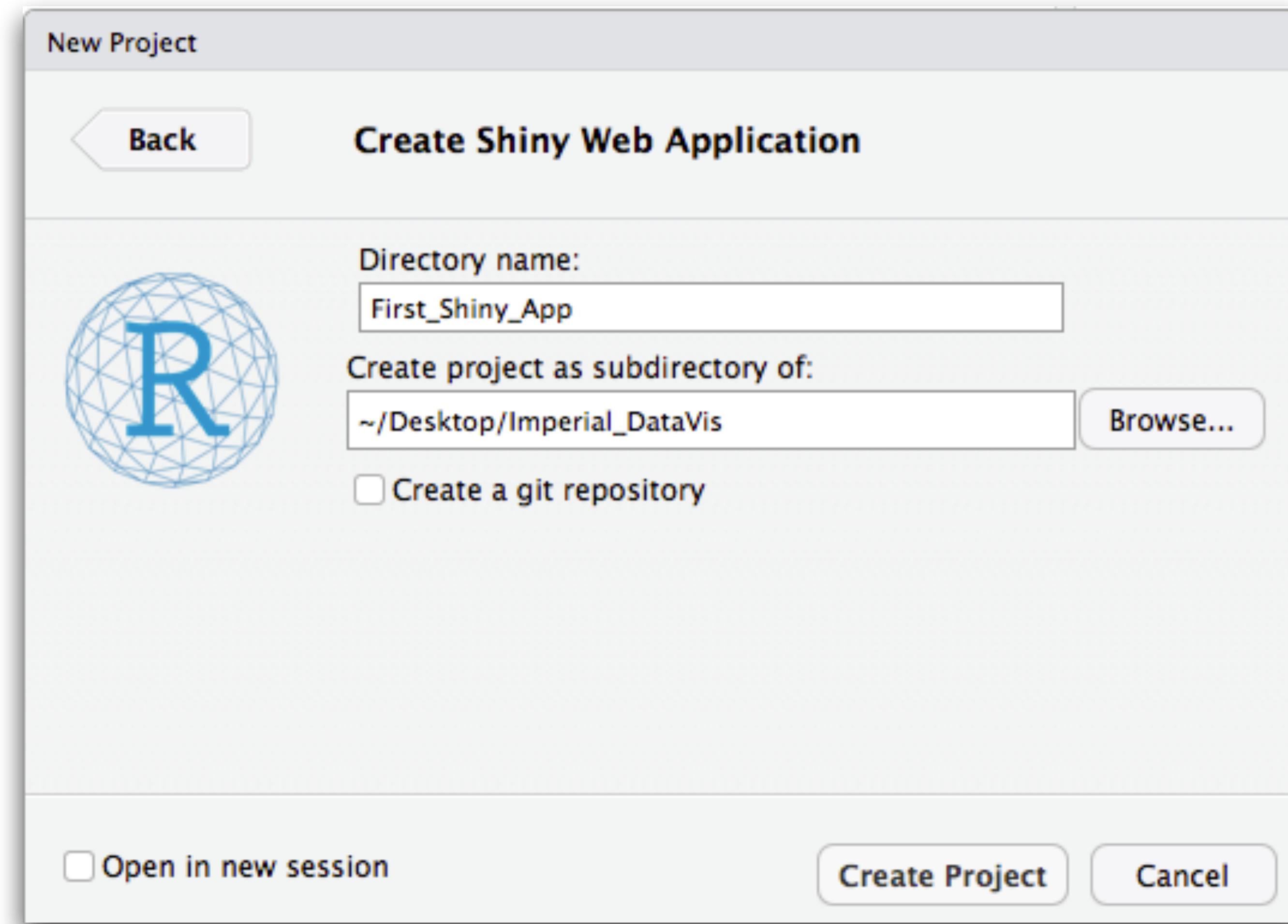
# 1) Create a new project



# 2) Select Shiny Web Application



# 2) Select Shiny Web Application



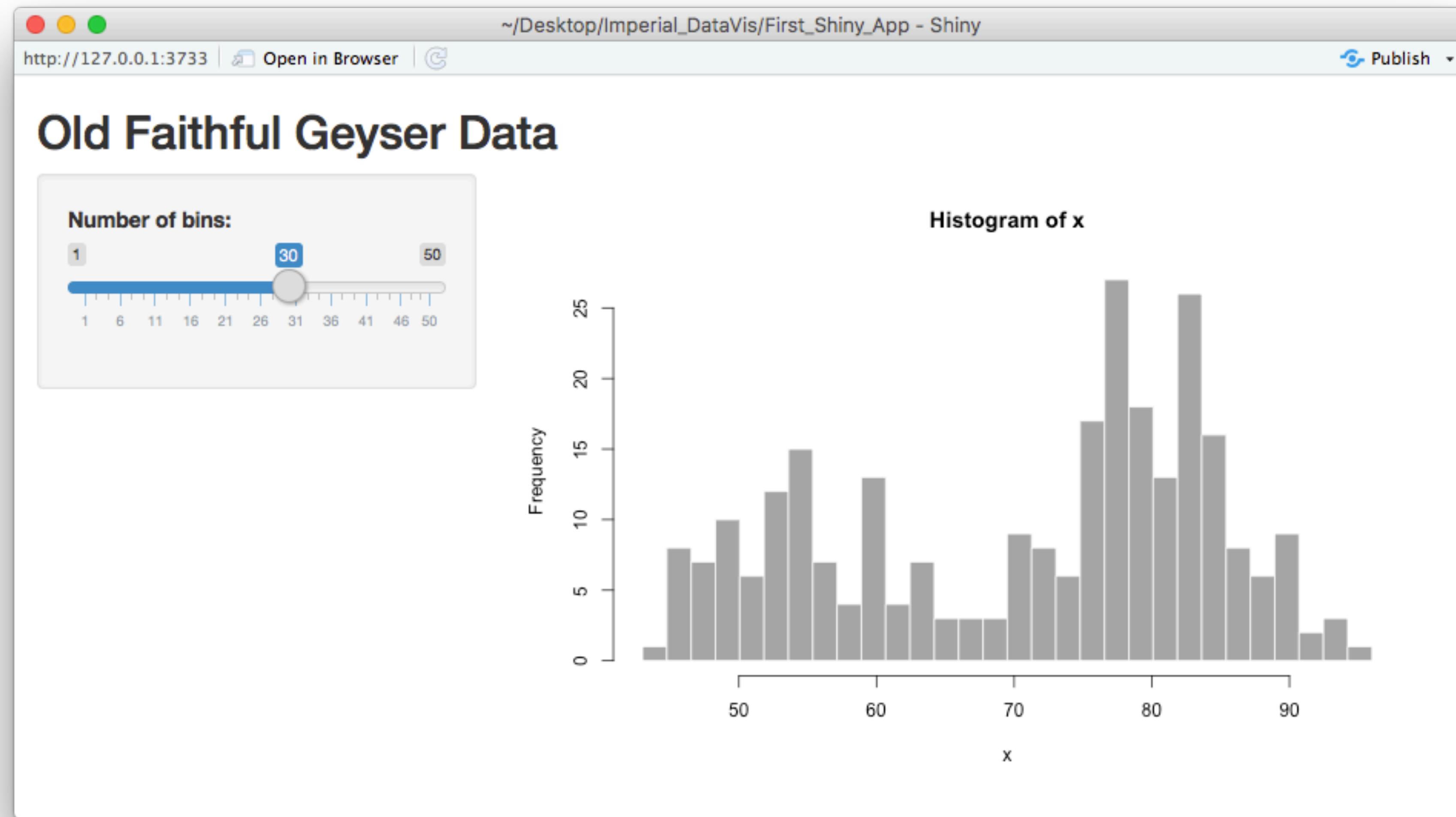
The screenshot shows an RStudio interface with the file 'app.R' open. The code defines a Shiny web application. A red box highlights the UI component, which includes the definition of the user interface (fluidPage, titlePanel, sidebarLayout, sidebarPanel, sliderInput, mainPanel, plotOutput). A blue box highlights the Server component, which includes the function definition for the server logic (renderPlot, hist), and the final line of code to run the shinyApp.

```
1 #  
2 # This is a Shiny web application. You can run the application by clicking  
3 # the 'Run App' button above.  
4 #  
5 # Find out more about building applications with Shiny here:  
6 #  
7 # http://shiny.rstudio.com/  
8 #  
9  
10 library(shiny)  
11  
12 # Define UI for application that draws a histogram  
13 ui <- fluidPage(  
14  
15   # Application title  
16   titlePanel("Old Faithful Geyser Data"),  
17  
18   # Sidebar with a slider input for number of bins  
19   sidebarLayout(  
20     sidebarPanel(  
21       sliderInput("bins",  
22         "Number of bins:",  
23         min = 1,  
24         max = 50,  
25         value = 30)  
26     ),  
27  
28     # Show a plot of the generated distribution  
29     mainPanel(  
30       plotOutput("distPlot")  
31     )  
32   )  
33 )  
34  
35 # Define server logic required to draw a histogram  
36 server <- function(input, output) {  
37  
38   output$distPlot <- renderPlot({  
39     # generate bins based on input$bins from ui.R  
40     x <- faithful[, 2]  
41     bins <- seq(min(x), max(x), length.out = input$bins + 1)  
42  
43     # draw the histogram with the specified number of bins  
44     hist(x, breaks = bins, col = 'darkgray', border = 'white')  
45   })  
46 }  
47  
48 # Run the application  
49 shinyApp(ui = ui, server = server)
```

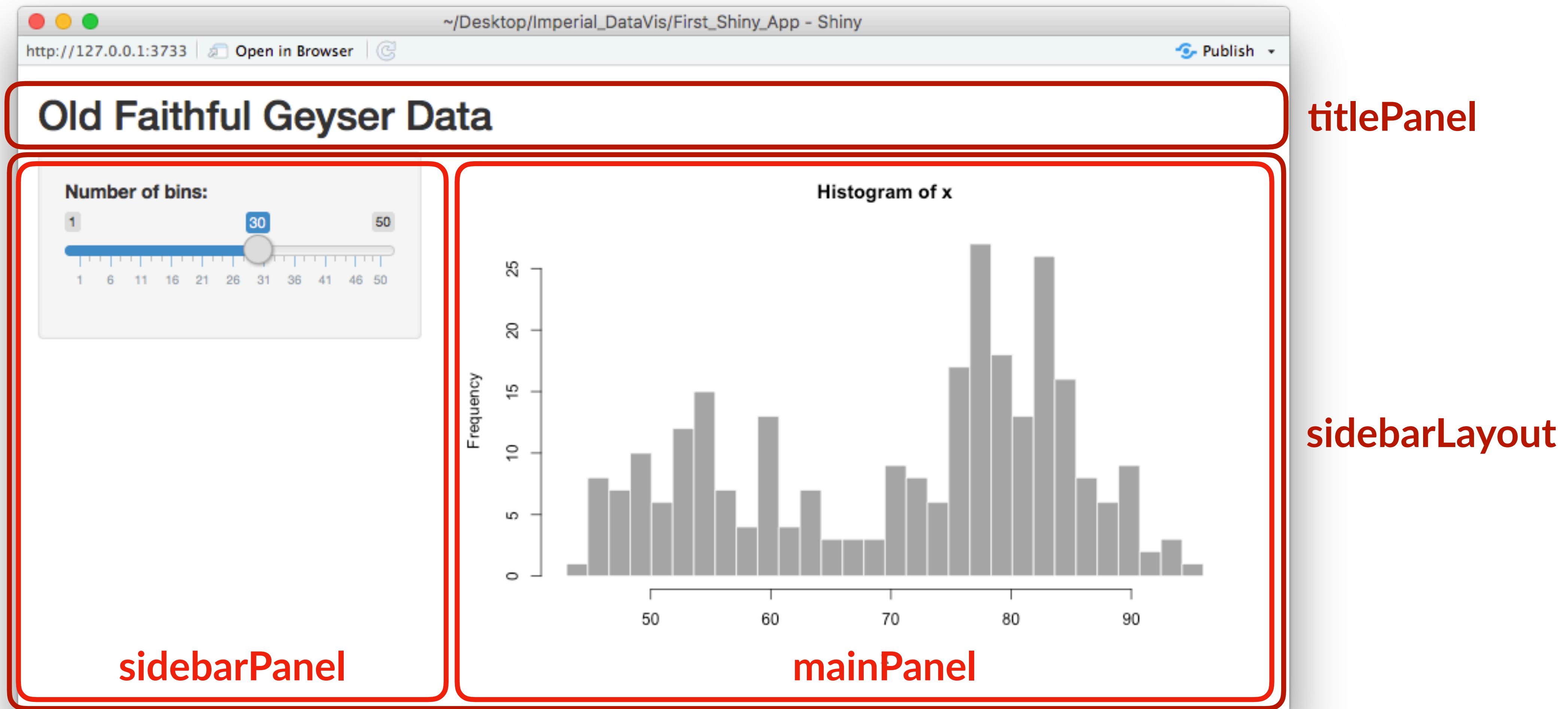
UI component

Server component

# 3) Hit “Run App”



# UI component



# Server component

```
35 # Define server logic required to draw a histogram
36 server <- function(input, output) {
37
38   output$distPlot <- renderPlot({
39     # generate bins based on input$bins from ui.R
40     x    <- faithful[, 2]
41     bins <- seq(min(x), max(x), length.out = input$bins + 1)
42
43     # draw the histogram with the specified number of bins
44     hist(x, breaks = bins, col = 'darkgray', border = 'white')
45   })
46 }
```

# replacing with ggplot

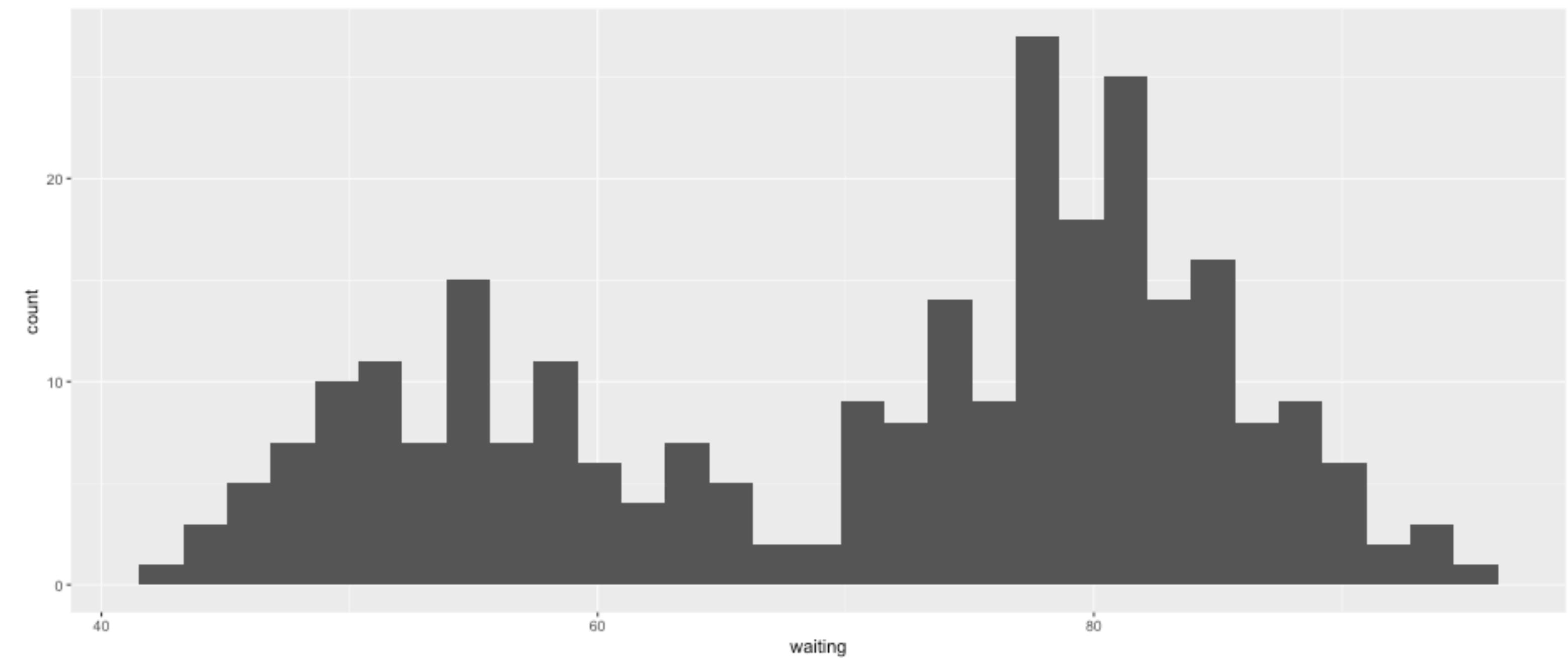
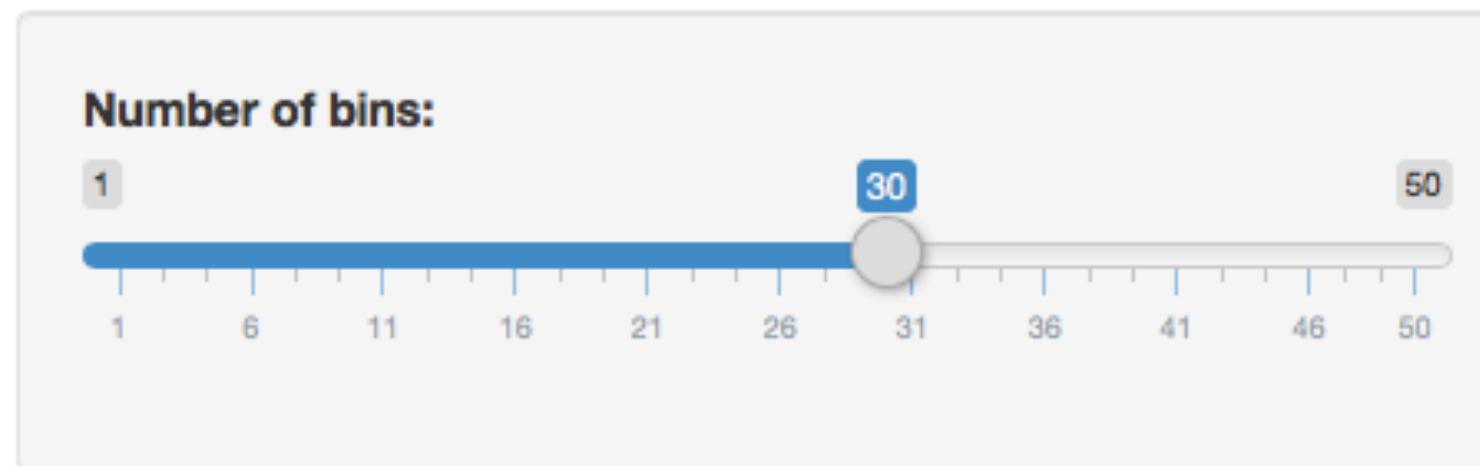
```
# Define server logic required to draw a histogram
server <- function(input, output) {

  output$distPlot <- renderPlot({
    # generate bins based on input$bins from ui.R
    # x      <- faithful[, 2]
    # bins   <- seq(min(x), max(x), length.out = input$bins + 1)
    # draw the histogram with the specified number of bins
    # hist(x, breaks = bins, col = 'darkgray', border = 'white')

    ggplot(data = faithful)+
      geom_histogram(mapping = aes(x = waiting), bins = input$bins +1, na.rm = T)
  })
}
```

# ggplot example

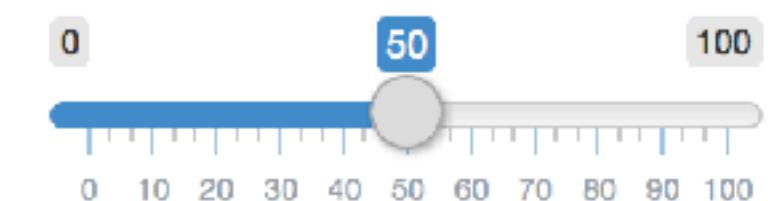
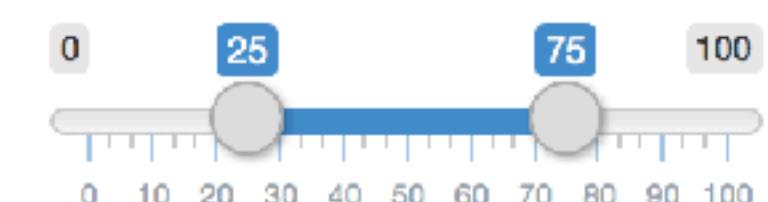
## Old Faithful Geyser Data



# Control widgets

http://127.0.0.1:3771 | [Open in Browser](#) | [G](#) | [Publish](#)

## Basic widgets

|   |  |  |                                    |
|---|--|--|------------------------------------|
| <b>Buttons</b><br><input type="button" value="Action"/><br><br><input type="button" value="Submit"/>                                  | <b>Single checkbox</b><br><input checked="" type="checkbox"/> Choice A         | <b>Checkbox group</b><br><input checked="" type="checkbox"/> Choice 1<br><input type="checkbox"/> Choice 2<br><input type="checkbox"/> Choice 3  | <b>Date input</b><br>2014-01-01    |
| <b>Date range</b><br>2017-06-21 to 2017-06-21   | <b>File input</b><br><input type="button" value="Browse..."/> No file selected | <b>Help text</b><br>Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.   | <b>Numeric input</b><br>1          |
| <b>Radio buttons</b><br><input checked="" type="radio"/> Choice 1<br><input type="radio"/> Choice 2<br><input type="radio"/> Choice 3 | <b>Select box</b><br><input type="button" value="Choice 1"/>                   | <b>Sliders</b><br><br> | <b>Text input</b><br>Enter text... |

[See Shiny Widgets Gallery](#)

# Pros and Cons

- Pros

- Real time analytics and outputs
- Minimal knowledge of HTML, CSS, or Javascript
- Share your web app: <http://www.shinyapps.io>, but with some limitations...
- Open source
- Run locally

- Cons

- Requires timely updates:
  - As functions used in the apps gets outdated sometimes with new package versions, it is necessary to update your shiny app time to time
- No selective access and permission
- Restriction on traffic in free version
  - 25 active hours of your app

# Additional resources

- <http://rstudio.github.io/shinythemes/>
  - stylise your shiny app
- <http://rstudio.github.io/leaflet/>
  - embedding interactive map
- <https://rstudio.github.io/shinydashboard/>
  - extension to help make a dashboard
- <http://shiny.rstudio.com/reference/shiny/latest/>
  - Function reference of shiny

# Any Questions?

Post questions on slack.