

Welcome to Data Science for Psychologists



S. Mason Garrison
Assistant Professor
Wake Forest University



Data Science for Psychologists Workshop

APS Annual Convention

Mason Garrison

Assistant Professor of Quantitative Psychology

Wake Forest University



Data Science for Psychologists

A Little About Me



Data Science for Psychologists

Who Am I?

- + My name is S. Mason Garrison
- + I'm an Assistant Professor of Quantitative Psychology
 - + at Wake Forest University
- + I'm a Quantitative Psychologist and
 - + Behavior Geneticist by training.



My Work



Why I'm Here Today

- + I'm passionate about sharing my love for data science with others, especially those within the field of psychology
- + I believe that the tools and methodologies of data science can significantly enhance our ability to conduct impactful research in psychology
- + I'm excited to be here and share this journey with you!



Fun Facts About Me

- + I'm a cat lover! (as you might have guessed from my use of )
- + I enjoy hiking and being in nature during my downtime
- + I'm a coffee enthusiast – I love exploring new blends and brewing methods. Happy to chat about coffee during breaks!



Workshop Overview!



Data Science for Psychologists

Workshop Overview

- +  = Introduction to R and the tidyverse
- +  = Data Wrangling
- +  </> = Data Visualization

DataScience4Psych.github.io/DataScience4Psych/



Data Science for Psychologists

Introduction to R and the tidyverse

- + Getting started with R and RStudio
- + Understanding the structure and philosophy of the tidyverse
- + Learning basic R syntax and operations



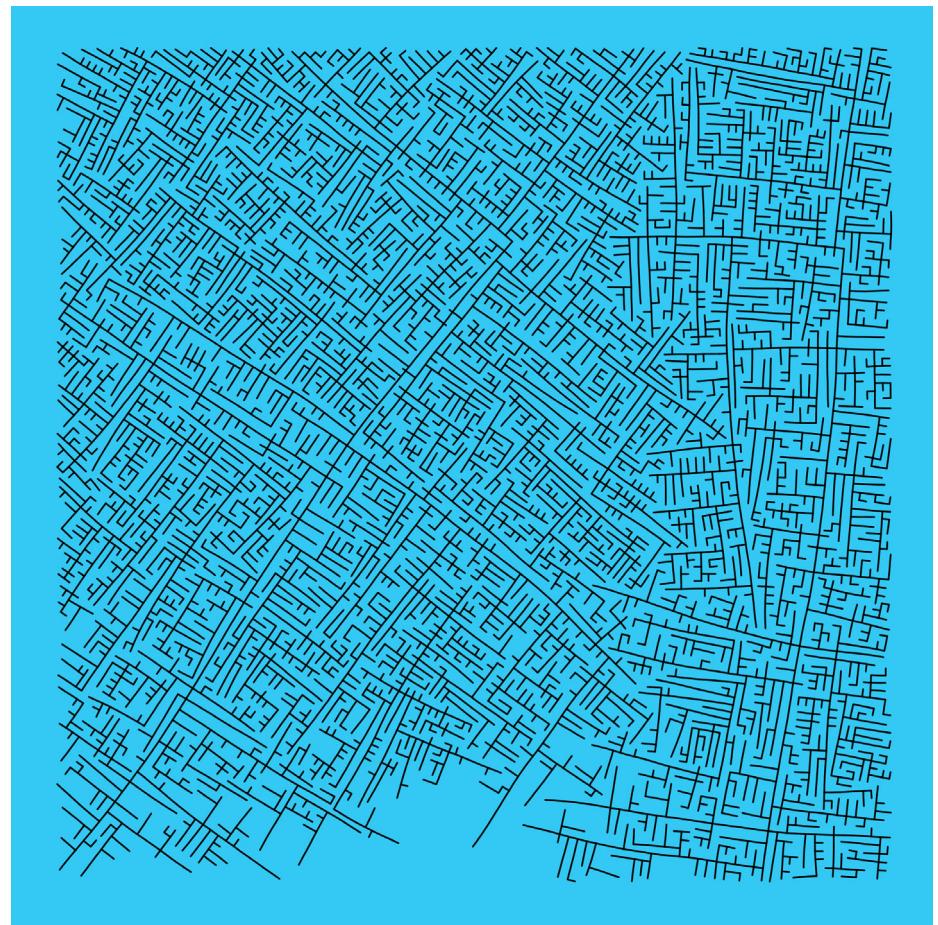
Data Wrangling

- + Importing and exporting data in various formats
- + Cleaning, transforming, and reshaping data



Data Visualization

- + Creating static and interactive visualizations using ggplot2
- + Customizing visual elements to enhance your plots
- + Interpreting and describing visualizations



Hello world!



What is data science?

- +  +  = data science?
- +  + </> = data science?
- +  +  + </> = data science?
- +  +  + </> = data science?

Data science is an exciting discipline that allows you to turn raw data into understanding, insight, and knowledge. We're going to learn to do this in a tidy way -- more on that later!



What is this course?

This course is an adaptation of my introduction to data science that is designed for psychologists. It emphasizes statistical thinking and best practices.

Q - What data science background does this course assume?

A - None.

Q - Is this an intro CS course?

A - Although statistics and computer science \neq data science, they are very closely related and have tremendous of overlap. Hence, this course is a great way to get comfortable with those topics. However this course is **not** your typical course.

Q - Will we be doing computing?



A - Yes.

Q - What computing language will we learn?

A - R.

Q: Why not language X?

A: We can discuss that *remotely* over ☕.



Where is this course?

DataScience4Psych.github.io/DataScience4Psych/



Meet the toolkit



Introduction to R and the tidyverse

- + Getting started with R and RStudio
- + Understanding the structure and philosophy of the tidyverse
- + Learning basic R syntax and operations



Reproducible data analysis



Reproducibility checklist

What does it mean for a data analysis to be "reproducible"?

Near-term goals:

- + Are the tables and figures reproducible from the code and data?
- + Does the code actually do what you think it does?
- + In addition to what was done, is it clear **why** it was done? (e.g., how were parameter settings chosen?)

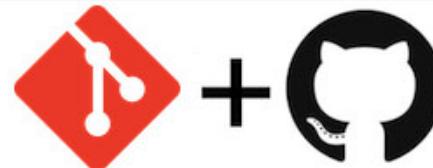
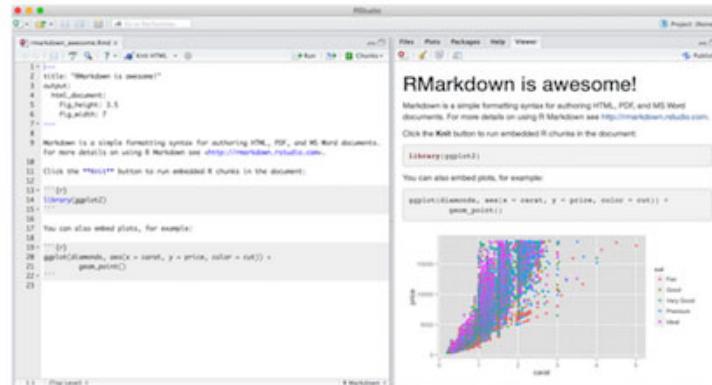
Long-term goals:

- + Can the code be used for other data?
- + Can you extend the code to do other things?



Data Science for Psychologists

Toolkit



- + Scriptability → R
- + Literate programming (code, narrative, output in one place) → R Markdown
- + Version control → Git / GitHub



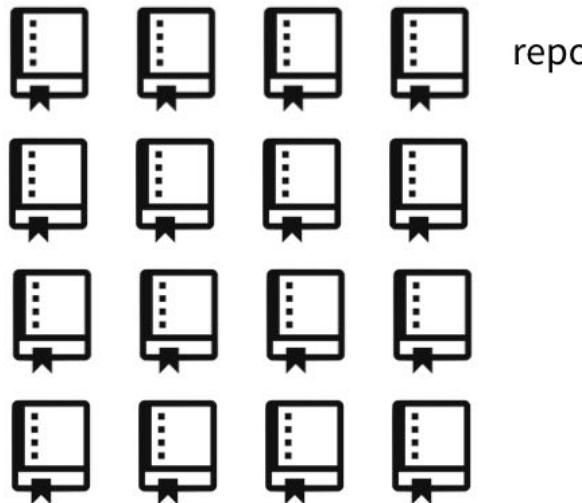
Toolkit overview





organization

ids-s1-19



...

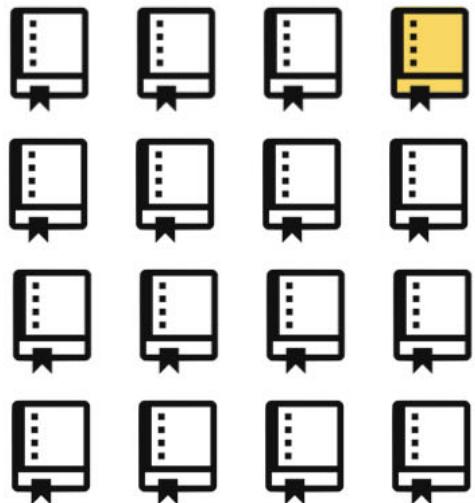


Data Science for Psychologists



organization

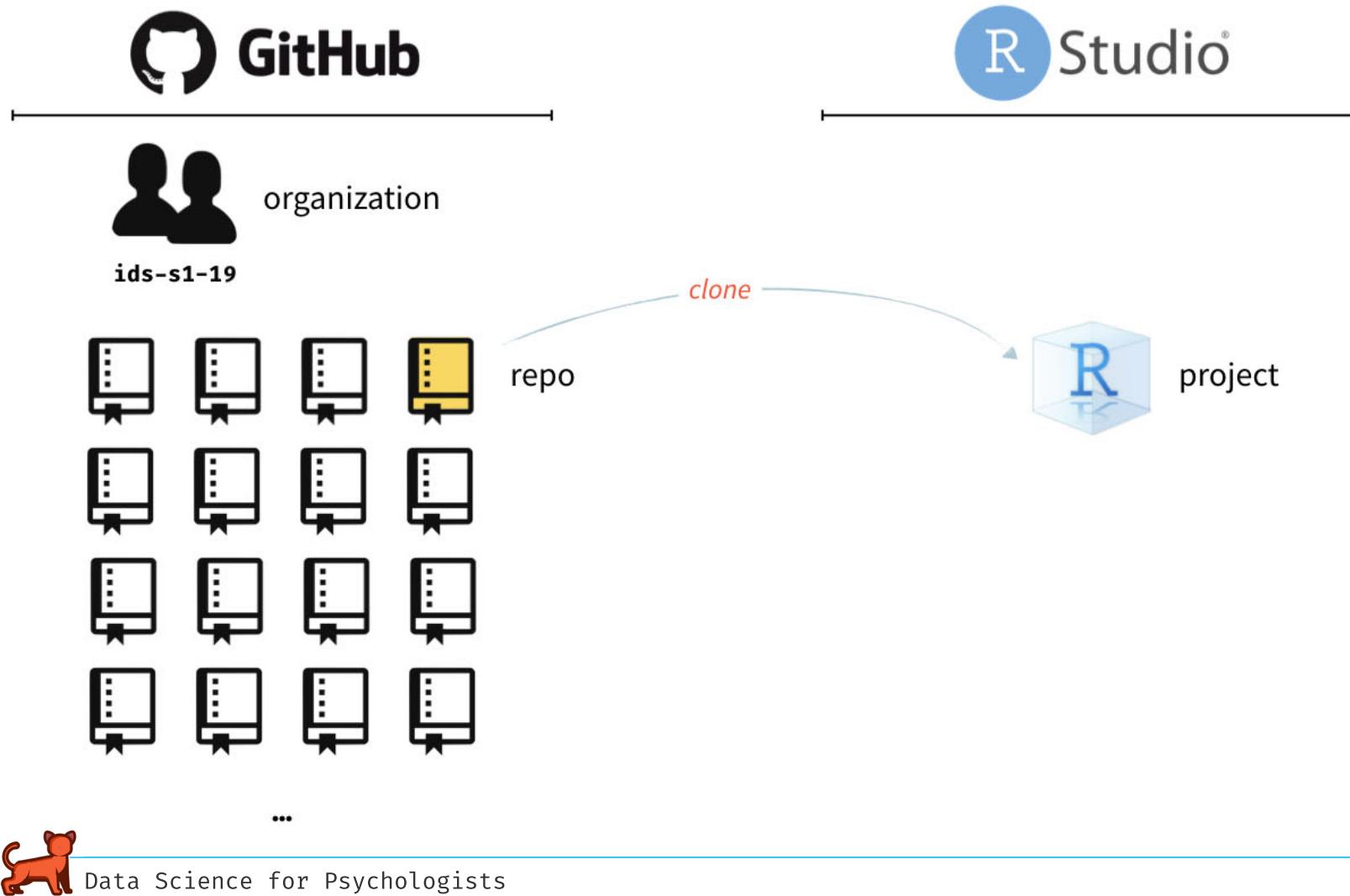
ids-s1-19

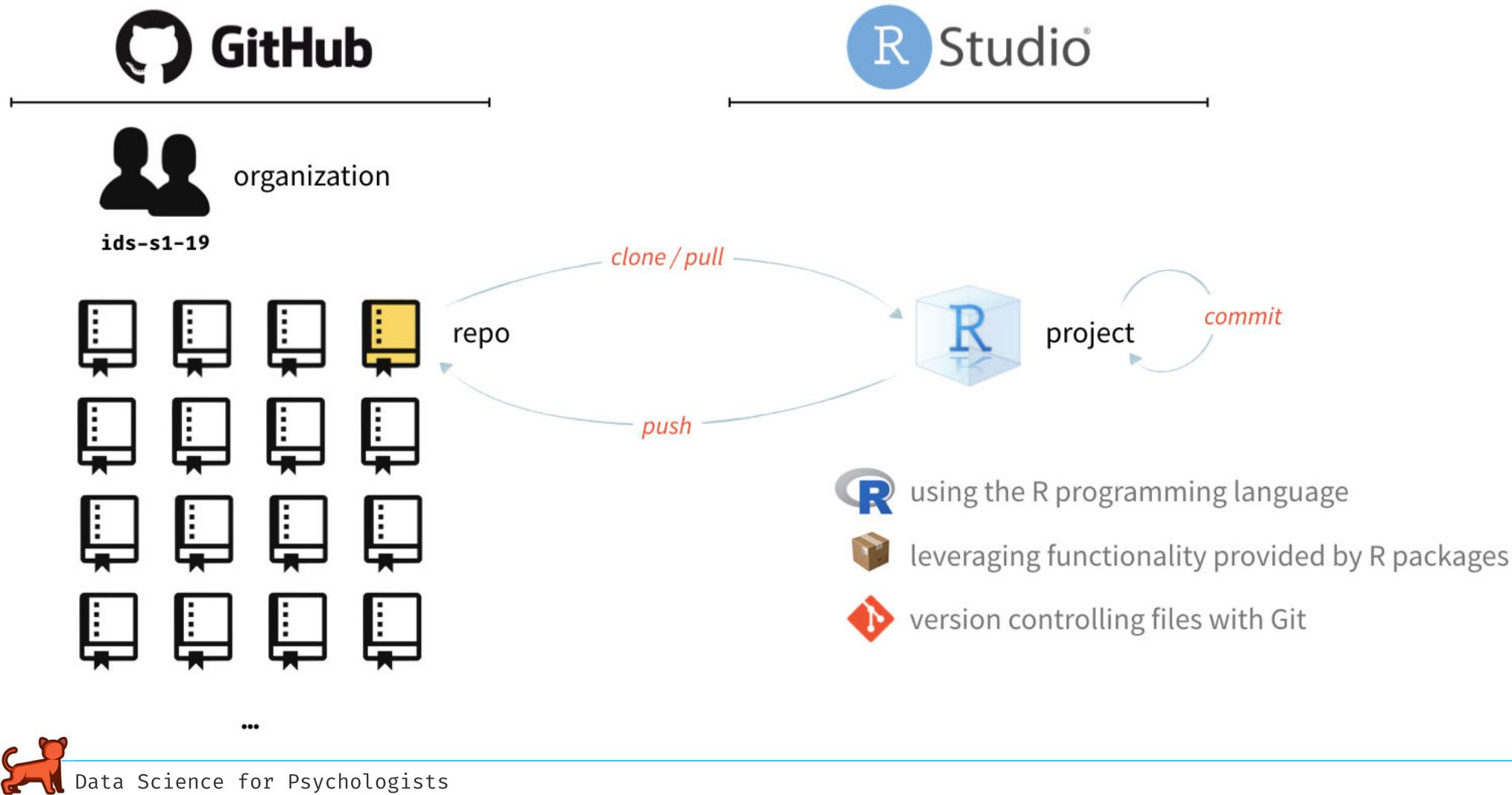


...



Data Science for Psychologists





R and RStudio



Data Science for Psychologists

What is R/RStudio?

- + R is a statistical programming language
- + RStudio is a convenient interface for R (an integrated development environment, IDE)
- + At its simplest:
 - + R is like a car's engine
 - + RStudio is like a car's dashboard

R: Engine



RStudio: Dashboard



Let's take a tour - R / RStudio



- + Console
- + Using R as a calculator
- + Environment
- + Loading and viewing a data frame
- + Accessing a variable in a data frame
- + R functions



Working with R at the command line

- + Launch RStudio/R.
- + Notice the default panes:
 - + Console (entire left)
 - + Environment/History (tabbed in upper right)
 - + Files/Plots/Packages/Help (tabbed in lower right)
- + FYI: You can change the default location of the panes, among many other things
 - + Customizing RStudio



Working with R at the command line (pt 2)

- + Go into the Console, where we interact with the live R process.
- + Make an assignment and then inspect the object you just created:

```
x <- 3 * 4  
x
```

```
## [1] 12
```

- + All R statements where you create objects -- "assignments" -- have this form:

```
objectName <- value
```

- + Read this as 'x gets 12'



R essentials

A short list (for now):

- + Functions are (most often) verbs, followed by what they will be applied to in parentheses:

```
do_this(to_this)  
do_that(to_this, to_that, with_those)
```

- + Columns (variables) in data frames are accessed with \$:

```
dataframe$var_name
```

- + Packages are installed with the `install.packages` function and loaded with the `library` function, once per session:

```
install.packages("package_name")  
library(package_name)
```



tidyverse



tidyverse.org

- + The tidyverse is an opinionated collection of R packages designed for data science.
- + All packages share an underlying philosophy and a common grammar.



R Markdown



Data Science for Psychologists

R Markdown

- + Fully reproducible reports -- each time you knit the analysis is ran from the beginning
- + Simple markdown syntax for text
- + Code goes in chunks, defined by three backticks, narrative goes outside of chunks



Let's take a tour - R Markdown



Concepts introduced:

- + Copying a project of mine
- + Knitting documents
- + R Markdown and (some) R syntax



Your turn!

- + The Bechdel test asks whether a work of fiction features at least two women who talk to each other about something other than a man, and there must be two women named characters.
- + Go to github page and fork the assignment Bechdel + R Markdown.
- + Open and knit the R Markdown document `bechdel.Rmd` and follow along with the instructions.



R Markdown cheat sheet

R Markdown :: CHEAT SHEET

What is R Markdown?

 **.Rmd files** - An R Markdown (.Rmd) file is a record of your research. It contains the code that a scientist needs to reproduce your work along with the narration that a reader needs to understand your work.

Reproducible Research • At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.

Dynamic Documents • You can choose to export the finished report in a variety of formats, including html, pdf, MS Word, or RTF documents; html or pdf based slides, Notebooks, and more.

Workflow



- ① Open a new .Rmd file at File ► New File ► R Markdown. Use the wizard that opens to pre-populate the file with a template
 - ② Write document by editing template
 - ③ Knit document to create report; use knit button or render() to knit
 - ④ Preview Output in IDE window
 - ⑤ Publish (optional) to web server
 - ⑥ Examine build log in R Markdown console
 - ⑦ Use output file that is saved along side .Rmd

The screenshot shows the RStudio interface with an R Markdown file named "report.Rmd". The code includes various R Markdown syntax elements like `#` for comments, `##` for section headings, and `knitr::` functions for output control. A context menu is open over the code editor, with several options highlighted:

- set preview location**: Points to the "Set Preview Location" option in the context menu.
- insert code chunk**: Points to the "Insert Code Chunk" option in the context menu.
- run code chunk(s)**: Points to the "Run Code Chunk(s)" option in the context menu.
- go to code chunk**: Points to the "Go To Code Chunk" option in the context menu.
- publish**: Points to the "Publish" option in the context menu.
- show outline**: Points to the "Show Outline" option in the context menu.
- run all previous chunks**: Points to the "Run All Previous Chunks" option in the context menu.
- modify chunk options**: Points to the "Modify Chunk Options" option in the context menu.
- run current chunk**: Points to the "Run Current Chunk" option in the context menu.

The browser preview window shows the rendered HTML version of the R Markdown document, which includes the title "R Markdown", a subtitle "RStudio", and the main content of the R code.

render

Use `rmarkdown::render()` to render/knit at cmd line. Important args:

input - file to render	output_options - List of render actions (none, render)	output_file	params - list of params to use
output_format	output_dir		

Embed code with knitr syntax

INLINE CODE

Insert with `r <code>` . Results appear as text without code.
Built with `r getRversion()`

CODE CHUNKS

CODE CHUNKS
One or more lines surrounded with ````{r}` and `````. Place chunk options within curly braces, after `r`. Insert with 

GLOBAL OPT

Set with knitr::opts_chunk

IMPORTANT CHUNK OPTIONS

IMPROVE CHUNK OPTIONS Enable results for future knits (default =

depends-on - chunk dependencies for caching

fig-align - 'left' | 'right' | 'center' (default: 'center')

The Apache logo, which is a stylized orange bull.

Data Science for Psychologists

R Markdown help

Help -> Markdown Quick Reference

Files Plots Packages Help Viewer

Markdown Quick Reference ▾ Find in Topic

Markdown Quick Reference

R Markdown is an easy-to-write plain text format for creating dynamic documents and reports. See [Using R Markdown](#) to learn more.

Emphasis

```
*italic* **bold**  
_italic_ __bold__
```

Headers

```
# Header 1  
## Header 2  
### Header 3
```

Lists

Unordered List

```
* Item 1  
* Item 2  
    + Item 2a  
    + Item 2b
```

Ordered List

```
1. Item 1  
2. Item 2  
3. Item 3  
    + Item 3a  
    + Item 3b
```

Workspaces

Remember this, and expect it to bite you a few times as you're learning to work with R Markdown: The workspace of your R Markdown document is separate from the Console!

- + Run the following in the console

```
x <- 2  
x * 3
```

All looks good, eh?

- + Then, add the following chunk in your R Markdown document

```
x * 3
```

What happens? Why the error?



Data Science for Psychologists

Wrapping Up... R and R Markdown



Getting help in R



Data Science for Psychologists

Reading help files

The name of the function, and the library it is in.

What it does.

More details on each named argument. This will tell you what class of thing each argument has to be—an object, a number, a data frame, a logical value, etc.

What the function returns—i.e., the result of whatever operation or calculation it performs. This can be a single number, as here, or a multi-part object such as a list, a data frame, a plot, or a model.

R Documentation
Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)
```

Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for `trim = 0`, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of `trim` outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether `NA` values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.

The ellipsis allows other arguments to be passed to and from the function.

Value

If `trim` is zero (the default), the arithmetic mean of the values in `x` is computed, as a numeric or complex vector of length one. If `x` is not logical (coerced to numeric), numeric (including integer) or complex, `NA_real_` is returned, with a warning.

If `trim` is non-zero, a symmetrically trimmed mean is computed with a fraction of `trim` observations deleted from each end before the mean is computed.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also

[weighted.mean](#), [mean.POSIXct](#), [colMeans](#) for row and column means.

Examples

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))
```

Self-contained examples that you can run at the console. These may use built-in datasets or other R functions.

[Package `base` version 3.4.3 [Index](#)] Visit the package's Index page to look for Demos and Vignettes detailing how it works.



Asking good questions

- + **Good:** Describe your intention and include your code and the error
- + **Better:** Describe your intention and create a minimum working example
- + **Best:** Write a **reproducible example** (reprex) -- we'll introduce this concept more formally and teach you the tools for it a little later in the semester



- + Use code formatting
- + For issues with R code: copy / paste your code and resulting error, don't use screenshots!



Wrapping Up... Getting help in R



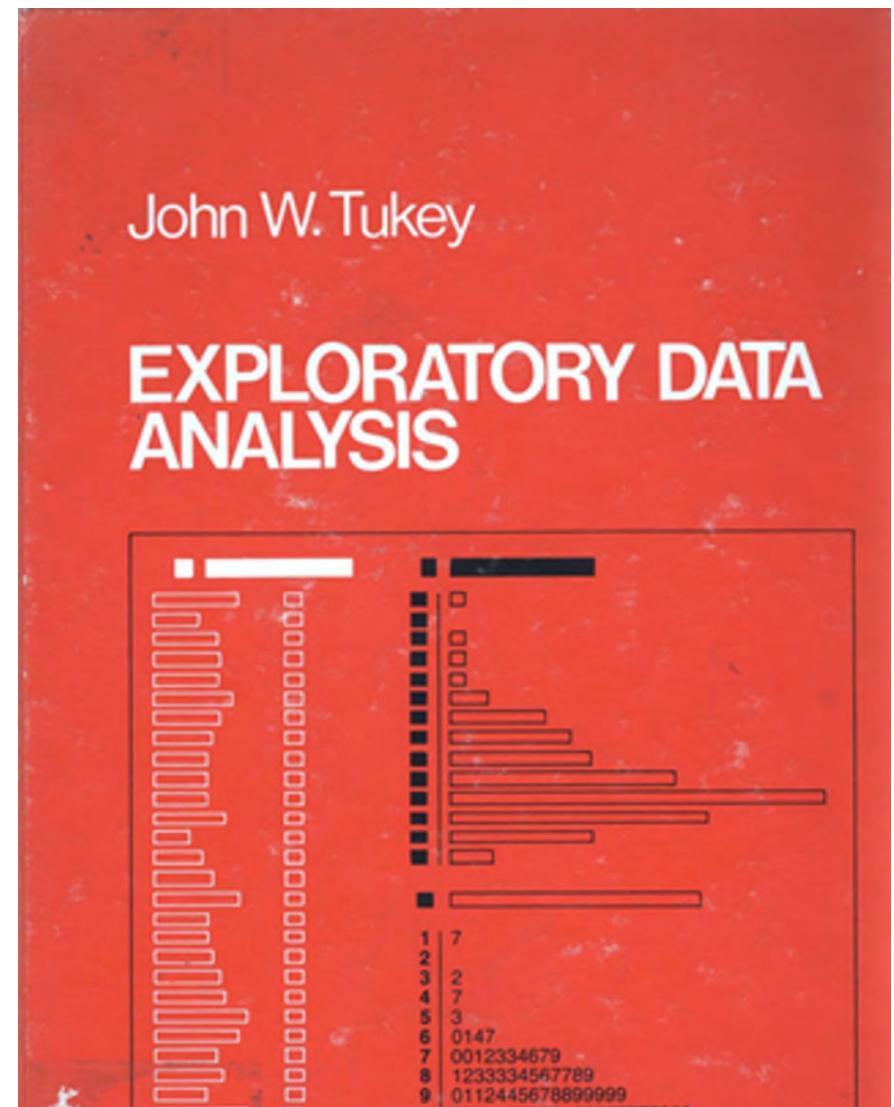
"Data and visualization



Exploratory data analysis



"We will be exploring numbers. We need to handle them easily and look at them effectively. Techniques for handling and looking â€” whether graphical, arithmetic, or intermediate â€” will be important."



What is EDA?



- + Exploratory data analysis (EDA) is an approach to analyzing data sets to summarize its main characteristics.
- + Today, we'll be focusing on the visual aspects
- + But we might also calculate summary statistics and perform data wrangling/manipulation/transformation at (or before) this stage of the analysis.
- + That's what we'll focus on next.



Data visualization



Data visualization

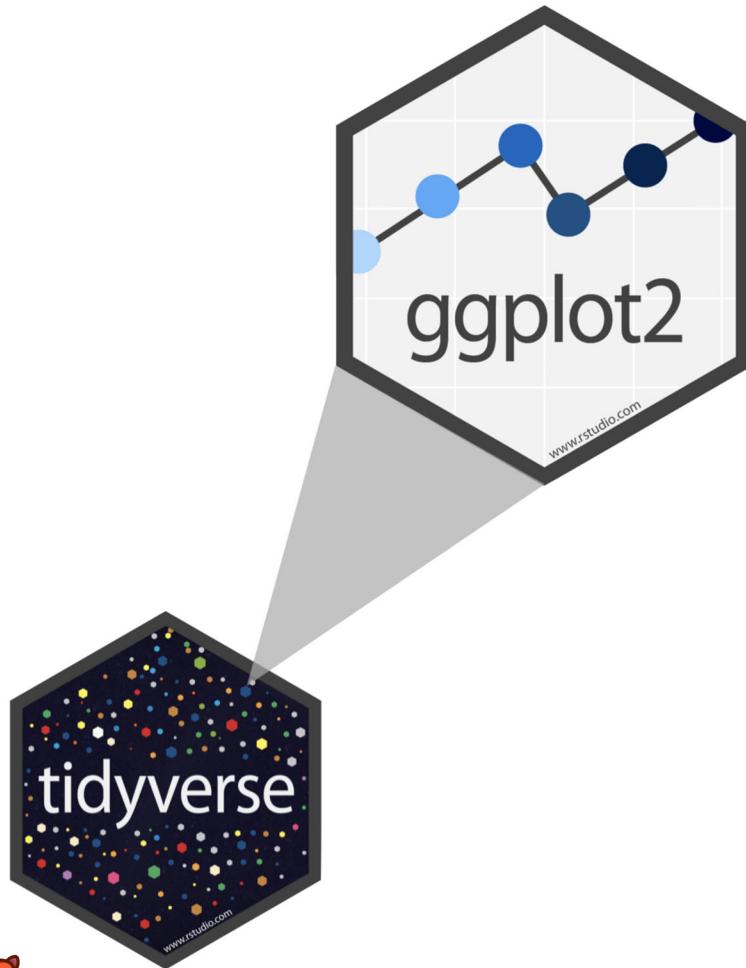
"The simple graph has brought more information to the data analyst's mind than any other device."

John Tukey

- + Data visualization is the creation and study of the visual representation of data.
- + Many tools for visualizing data (R is one of them) exist, as do many approaches/systems within R for making data visualizations (**ggplot2** is one of them, and that's what we're going to use).



ggplot2 ∈ tidyverse

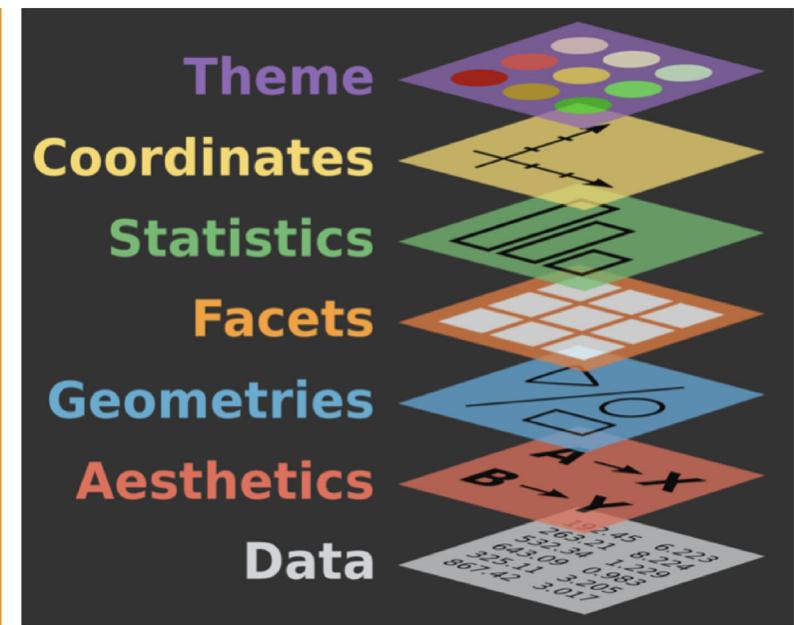
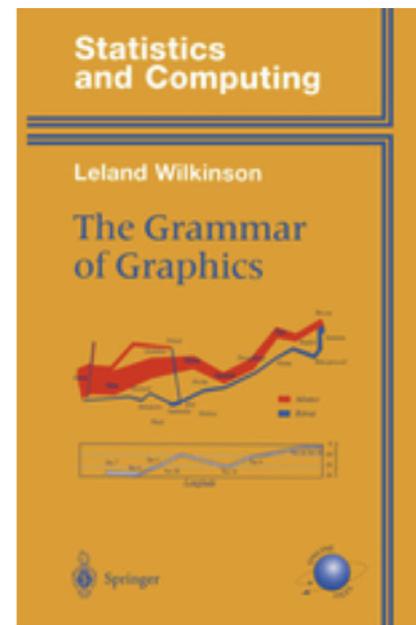


- + **ggplot2** is tidyverse's data visualization package
- + gg in "ggplot2" stands for Grammar of Graphics
- + Inspired by the book **Grammar of Graphics** by Leland Wilkinson



Grammar of Graphics

A grammar of graphics is a tool to concisely describe the components of a graphic



Source: BloggoType



Data Science for Psychologists

A First (and Reproducible) Example



Reading in the Data

- + Heights of the highest points by state

```
##  
## load required packages and data  
library(tidyverse)  
options(tibble.print_min = 10)  
heights = read_csv("data/highest-points-by-state.csv")  
## switch from feet to meters  
heights$elevation = heights$elevation * .3048
```



Taking a look

heights

```
## # A tibble: 50 × 2
##       elevation state
##             <dbl> <chr>
## 1           733. Alabama
## 2           6168. Alaska
## 3           3851. Arizona
## 4           839. Arkansas
## 5           4418. California
## 6           4399. Colorado
## 7           725. Connecticut
## 8           137. Delaware
## 9           105. Florida
## 10          1458. Georgia
## # i 40 more rows
```



Data Science for Psychologists

Taking another look

```
arrange(heights, elevation)
```

```
## # A tibble: 50 × 2
##       elevation state
##              <dbl> <chr>
## 1            105. Florida
## 2            137. Delaware
## 3            163. Louisiana
## 4            246. Mississippi
## 5            247. Rhode Island
## 6            376. Illinois
## 7            383. Indiana
## 8            472. Ohio
## 9            509. Iowa
## 10           540. Missouri
## # i 40 more rows
```



Taking another another look

```
arrange(heights, desc(elevation))
```

```
## # A tibble: 50 × 2
##       elevation state
##           <dbl> <chr>
## 1         6168. Alaska
## 2         4418. California
## 3         4399. Colorado
## 4         4392. Washington
## 5         4207. Wyoming
## 6         4205. Hawaii
## 7         4123. Utah
## 8         4011. New Mexico
## 9         4005. Nevada
## 10        3901. Montana
## # i 40 more rows
```



Taking another³ look... thoughtfully

Goals:

- + Write down the set of numbers, keeping as much detail as possible
- + Pack the numbers efficiently, so you can see all of them at once



Taking another³ look: Stem and Leaf

```
stem(heights$elevation)
```

```
##  
##      The decimal point is 3 digit(s) to the right of the |  
##  
## 0 | 11222445555667778  
## 1 | 0011123355566779  
## 2 | 0027  
## 3 | 4999  
## 4 | 00122444  
## 5 |  
## 6 | 2
```

- + Notice that parts of the numbers (the beginnings) are repeated.
- + The first digit of each number is printed at the beginning of the line, the remainder at the ends.
- + The first digit is the "stem", the remainder are the "leaves".



What have we learned?

The stem-and-leaf plot shows that there are three groups of states:

- + Alaska
- + The western and Rocky Mountain states (California, Colorado, Washington, Wyoming, Hawaii, Utah, New Mexico, Nevada, Montana, Idaho, Arizona, Oregon)
- + All the other states

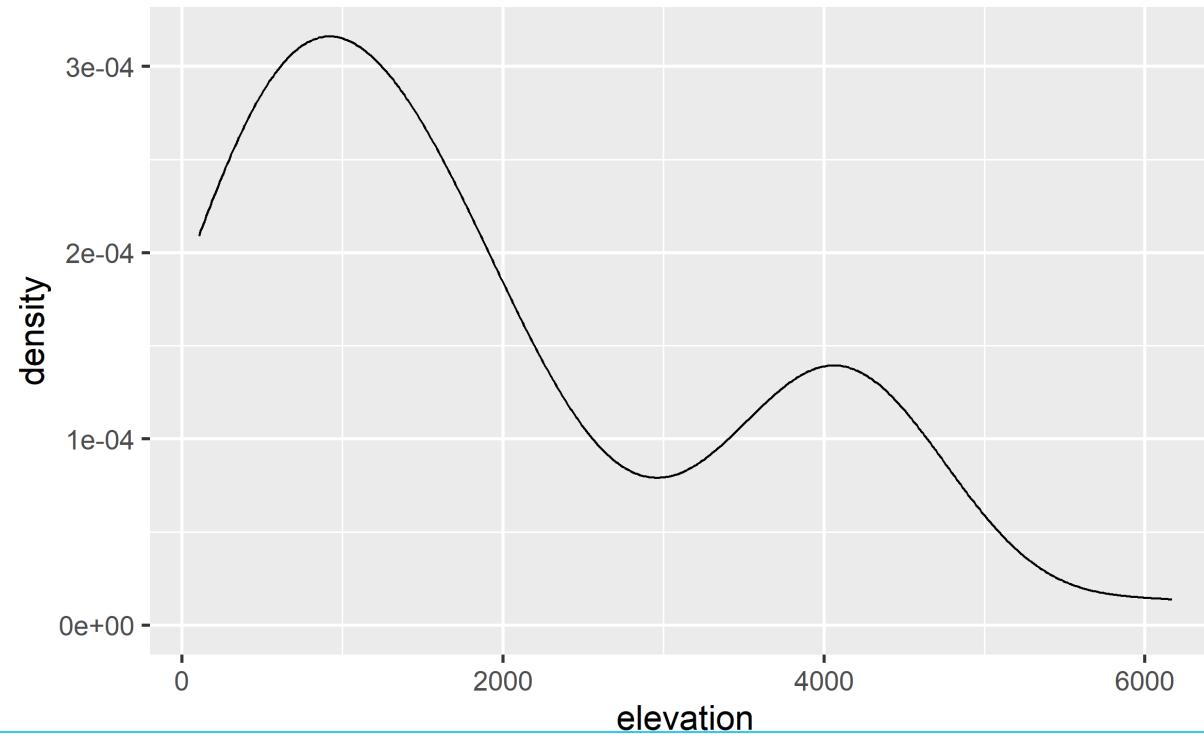
```
##  
##      The decimal point is 3 digit(s) to the right of the |  
##  
## 0 | 11222445555667778  
## 1 | 0011123355566779  
## 2 | 0027  
## 3 | 4999  
## 4 | 00122444  
## 5 |  
## 6 | 2
```



Taking another^{â ´} look: Density

Compare the stem-and-leaf plot with a density estimate

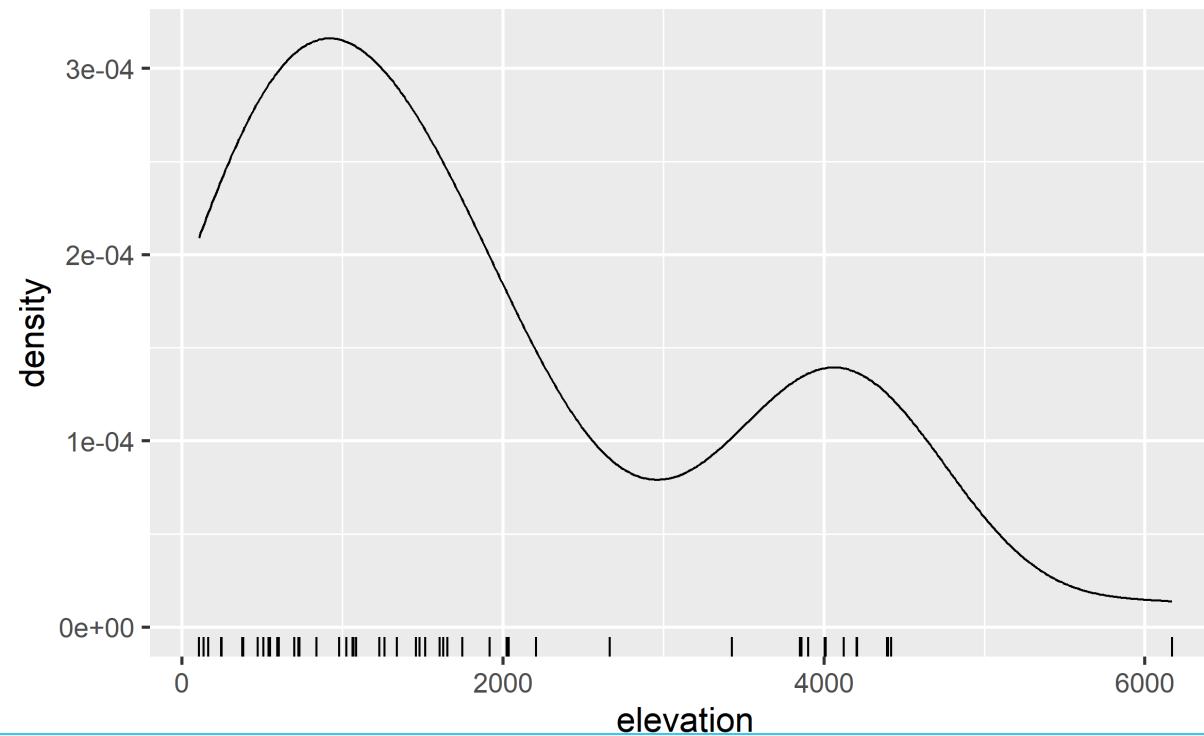
```
ggplot(heights, aes(x = elevation)) + geom_density()
```



Taking another[†] look: Density+

Compare the stem-and-leaf plot with a density estimate

```
ggplot(heights, aes(x = elevation)) + geom_density() + geom_rug()
```



Wrapping Up...



Data Science for Psychologists

What is in a dataset?



Dataset terminology

- + Each row is an **observation**
- + Each column is a **variable**

```
starwars
```

```
## # A tibble: 87 × 15
##   name    height  mass hair_color skin_color eye_color birth_year
##   <chr>     <int> <dbl> <chr>       <chr>      <chr>        <dbl>
## 1 Luke ...     172    77 blond      fair       blue         19
## 2 C-3PO       167    75 <NA>       gold       yellow       112
## 3 R2-D2        96    32 <NA>       white, bl... red        33
## 4 Darth...     202   136 none       white       yellow      41.9
## 5 Leia ...     150    49 brown      light       brown       19
## 6 Owen ...     178   120 brown, gr... light       blue        52
## 7 Beru ...     165    75 brown      light       blue        47
## 8 R5-D4        97    32 <NA>       white, red red        NA
## 9 Biggs...     183    84 black      light       brown       24
## 10 Obi-W...     182    77 auburn, w... fair       blue-gray     57
## # i 77 more rows
## # i 8 more variables: sex <chr>, gender <chr>, homeworld <chr>,
## #   species <chr>, films <list>, vehicles <list>,
## #   starships <list>, hair_color2 <fct>
```



Luke Skywalker

```
height = 172 cm      name = "Luke Skywalker"
```

```
weight = 77 kg        hair_color = "blond"
```

```
eye_color = "blue"    birth_year = 19 BBY
```

```
skin_color = "fair"    films = c("The Empire Strikes Back",  
species = "Human"      "Revenge of the Sith",  
                           "Return of the Jedi",  
                           "A New Hope",  
                           "The Force Awakens")
```

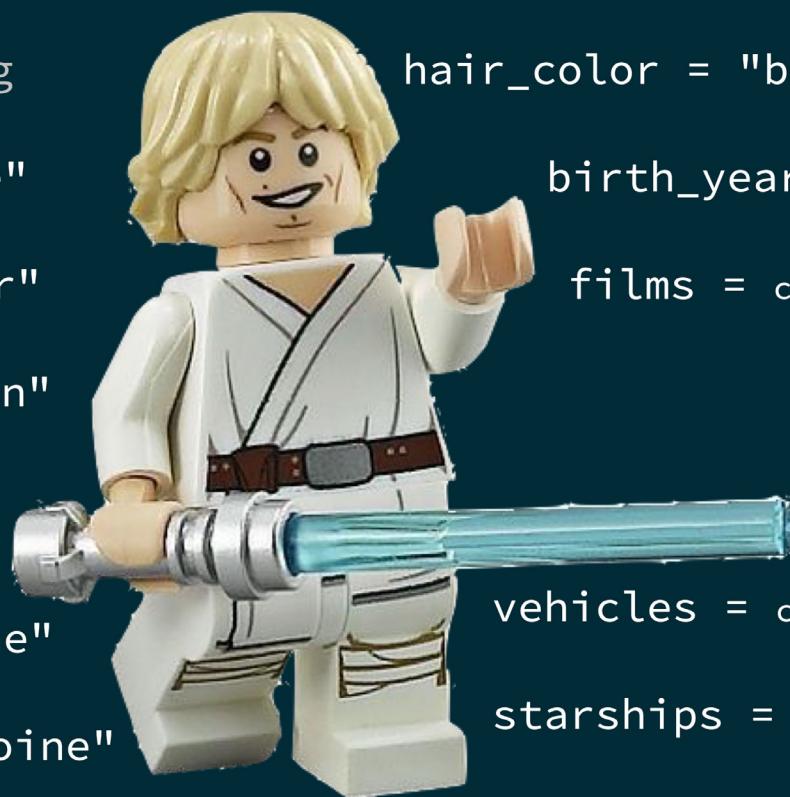
```
sex = "male"
```

```
vehicles = c("Snowspeeder",  
           "Imperial Speeder Bike")
```

```
gender = "masculine"
```

```
starships = c("X-wing",  
            "Imperial shuttle")
```

```
homeworld = "Tatooine"
```



What's in the Star Wars data?

Take a glimpse at the data:

```
glimpse(starwars)
```

```
## Rows: 87
## Columns: 15
## $ name      <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth ...
## $ height    <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, ...
## $ mass       <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, ...
## $ hair_color <chr> "blond", NA, NA, "none", "brown", "brown", g...
## $ skin_color <chr> "fair", "gold", "white", "blue", "white", "li...
## $ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", ...
## $ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, ...
## $ sex        <chr> "male", "none", "none", "male", "female", "...
## $ gender     <chr> "masculine", "masculine", "masculine", "mas...
## $ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine"...
## $ species    <chr> "Human", "Droid", "Droid", "Human", "Human"...
## $ films      <list> <"The Empire Strikes Back", "Revenge of th...
```



How many rows and columns does this dataset have? What does each row represent? What does each column represent?

?starwars

starwars {dplyr}

R Documentation

Starwars characters

Description

This data comes from SWAPI, the Star Wars API, <https://swapi.dev/>

Usage

```
starwars
```

Format

A tibble with 87 rows and 14 variables:

name	Name of the character
height	Height (cm)
mass	Weight (kg)
hair_color,skin_color,eye_color	Hair, skin, and eye colors
birth_year	Year born (BBY = Before Battle of Yavin)



Data Science for Psychologists

How many rows and columns does this dataset have?

```
nrow(starwars) # number of rows
```

```
## [1] 87
```

```
ncol(starwars) # number of columns
```

```
## [1] 15
```

```
dim(starwars) # dimensions (row co
```

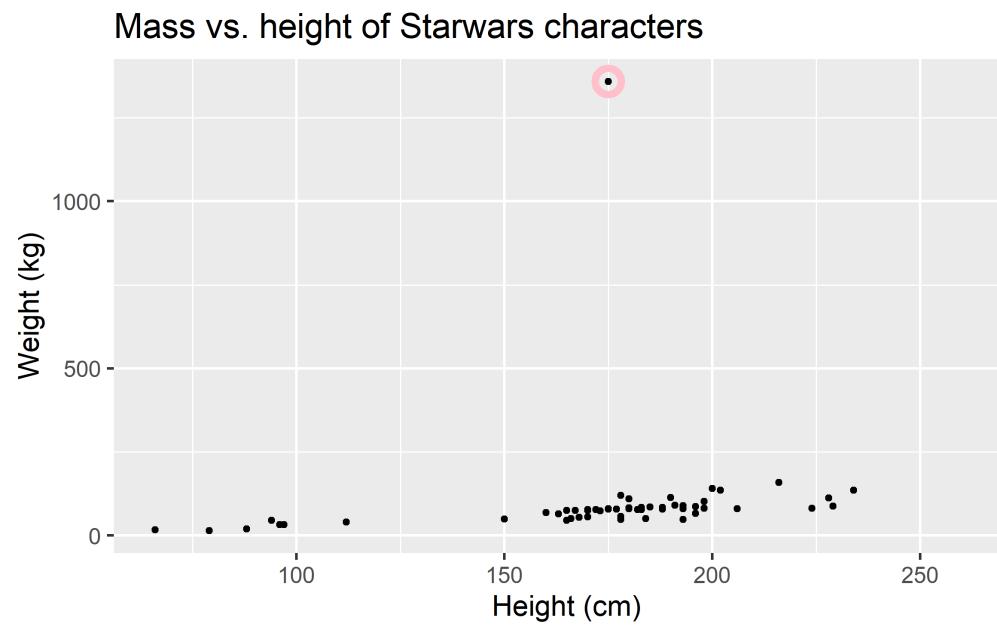


```
## [1] 87 15
```



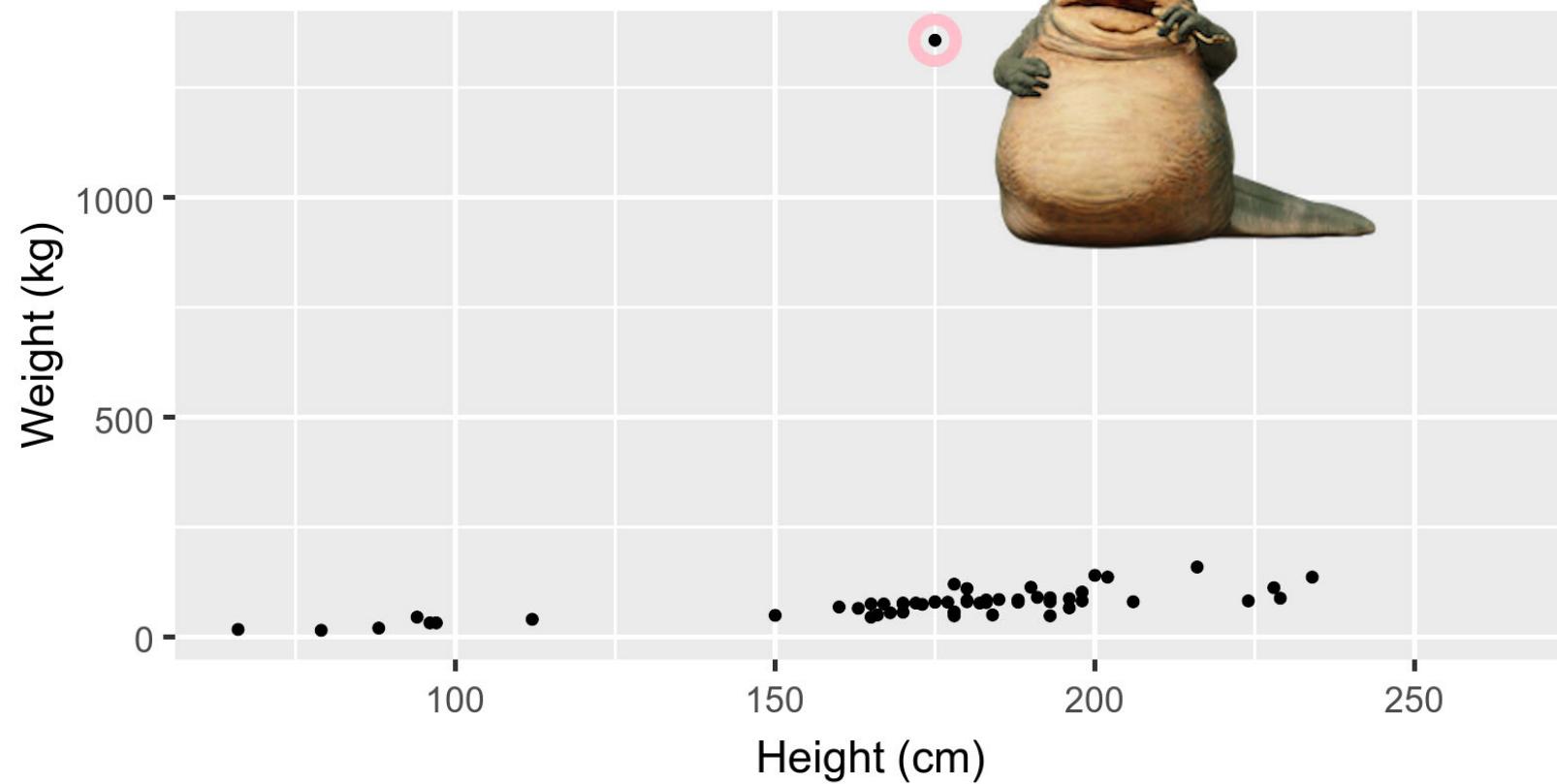
Mass vs. height

How would you describe the relationship between mass and height of Starwars characters? What other variables would help us understand data points that don't follow the overall trend? Who is the not so tall but really chubby character?



Jabba!

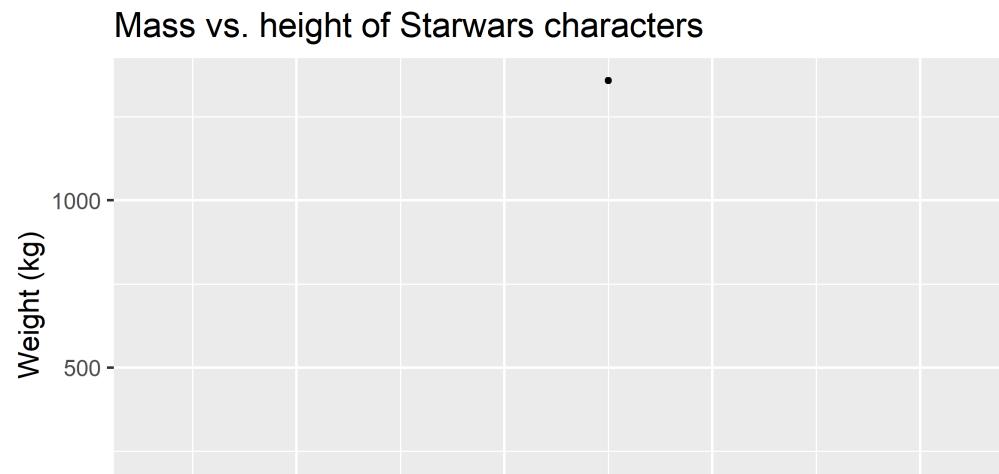
Mass vs. height of Starwars characters



Mass vs. height

```
ggplot(data = starwars, mapping = aes(x = height, y = mass)) +  
  geom_point() +  
  labs(title = "Mass vs. height of Starwars characters",  
       x = "Height (cm)", y = "Weight (kg)")
```

```
## Warning: Removed 28 rows containing missing values  
## (geom_point()).
```



- + What are the functions doing the plotting?
- + What is the dataset being plotted?
- + Which variables map to which features (aesthetics) of the plot?
- + What does the warning mean?⁺

```
ggplot(data = starwars, mapping = aes(x = height, y = mass)) +  
  geom_point() +  
  labs(title = "Mass vs. height of Starwars characters",  
       x = "Height (cm)", y = "Weight (kg)")
```

```
## Warning: Removed 28 rows containing missing values  
## (`geom_point()`).
```

⁺Suppressing warning to subsequent slides to save space



Hello ggplot2!

- + `ggplot()` is the main function in ggplot2
- + Plots are constructed in layers
- + Structure of the code for plots can be summarized as

```
ggplot(data = [dataset],  
       mapping = aes(x = [x-variable], y = [y-variable]),  
       geom_xxx() +  
       other options
```



- + The ggplot2 package comes with the tidyverse

```
library(tidyverse)
```

+ For help with ggplot2 see ggplot2.tidyverse.org

Data Science for Psychologists

Wrapping Up...



Data Science for Psychologists

Why do we visualize?



Anscombe's quartet

##	set	x	y	##	set	x	y
## 1	I	10	8.04	## 23	III	10	7.46
## 2	I	8	6.95	## 24	III	8	6.77
## 3	I	13	7.58	## 25	III	13	12.74
## 4	I	9	8.81	## 26	III	9	7.11
## 5	I	11	8.33	## 27	III	11	7.81
## 6	I	14	9.96	## 28	III	14	8.84
## 7	I	6	7.24	## 29	III	6	6.08
## 8	I	4	4.26	## 30	III	4	5.39
## 9	I	12	10.84	## 31	III	12	8.15
## 10	I	7	4.82	## 32	III	7	6.42
## 11	I	5	5.68	## 33	III	5	5.73
## 12	II	10	9.14	## 34	IV	8	6.58
## 13	II	8	8.14	## 35	IV	8	5.76
## 14	II	13	8.74	## 36	IV	8	7.71
## 15	II	9	8.77	## 37	IV	8	8.84
## 16	II	11	9.26	## 38	IV	8	8.47
## 17	II	14	8.04	## 39	IV	8	7.04



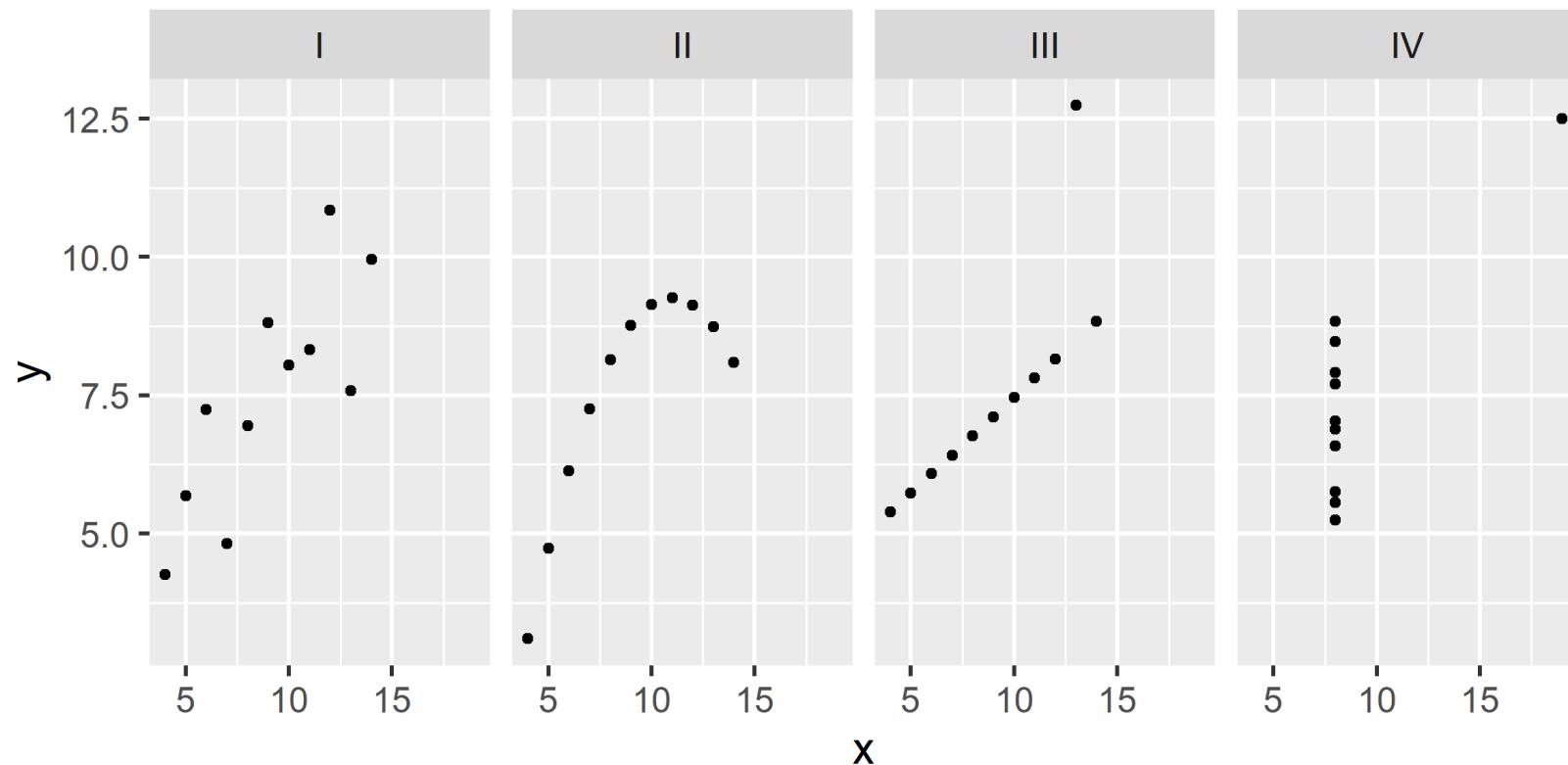
Summarizing Anscombe's quartet

```
quartet %>%
  group_by(set) %>%
  summarize(
    mean_x = mean(x),
    mean_y = mean(y),
    sd_x = sd(x),
    sd_y = sd(y),
    r = cor(x, y)
  )
```

```
## # A tibble: 4 × 6
##   set   mean_x  mean_y   sd_x   sd_y      r
##   <fct>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 I       9     7.50    3.32    2.03  0.816
## 2 II      9     7.50    3.32    2.03  0.816
## 3 III     9     7.5     3.32    2.03  0.816
## 4 IV      9     7.50    3.32    2.03  0.817
```



Visualizing Anscombe's quartet



Wrapping Up...



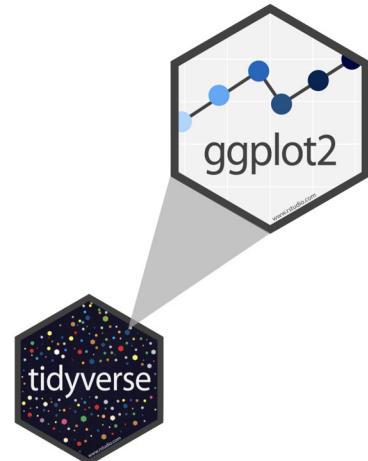
Visualizing data with ggplot2



ggplot2 ❤️ 🐧



Data Science for Psychologists



ggplot2 ∈ tidyverse

- + **ggplot2** is tidyverse's data visualization package
- + Structure of the code for plots can be summarized as

```
ggplot(data = [dataset],  
       mapping = aes(x = [x-variable],  
                     y = [y-variable])) +  
       geom_xxx() +  
       other options
```



Data: Palmer Penguins

Measurements for penguin species, island in Palmer Archipelago, size (flipper length, body mass, bill dimensions), and sex.



Data: Palmer Penguins

Measurements for penguin species, island in Palmer Archipelago, size (flipper length, body mass, bill dimensions), and sex.

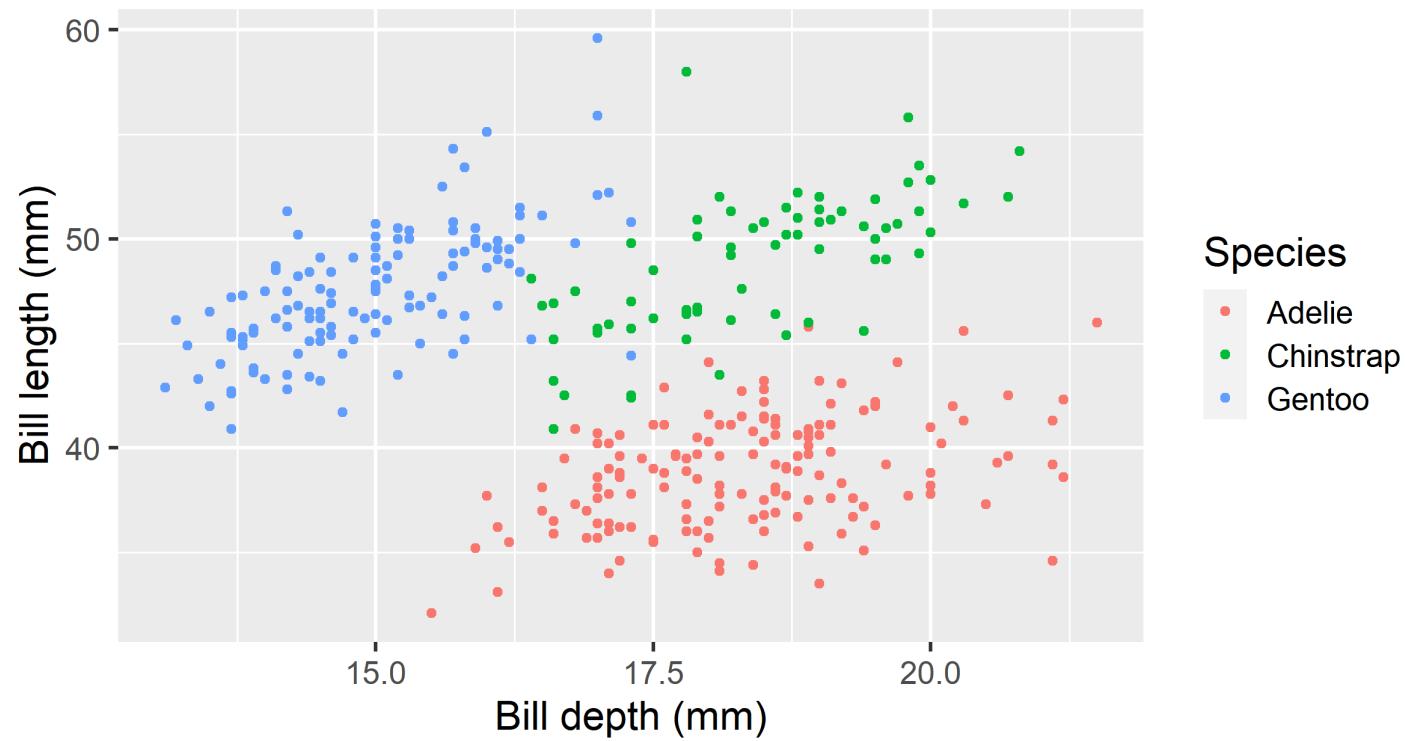
```
library(palmerpenguins)
glimpse(penguins)
```

```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adeli...
## $ island             <fct> Torgersen, Torgersen, Torgersen, Torg...
## $ bill_length_mm     <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38....
## $ bill_depth_mm      <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17....
## $ flipper_length_mm  <int> 181, 186, 195, NA, 193, 190, 181, 195...
## $ body_mass_g         <int> 3750, 3800, 3250, NA, 3450, 3650, 362...
## $ sex                <fct> male, female, female, NA, female, mal...
## $ year               <int> 2007, 2007, 2007, 2007, 2007, 2007, 2...
```



Plot

Bill depth and length
Dimensions for Adelie, Chinstrap, and Gentoo Penguins



Code

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm, y = bill_length_mm,
                      color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",
       x = "Bill depth (mm)", y = "Bill length (mm)",
       color = "Species")
```

```
## Warning: Removed 2 rows containing missing values
## (`geom_point()`).
```



Wrapping Up...



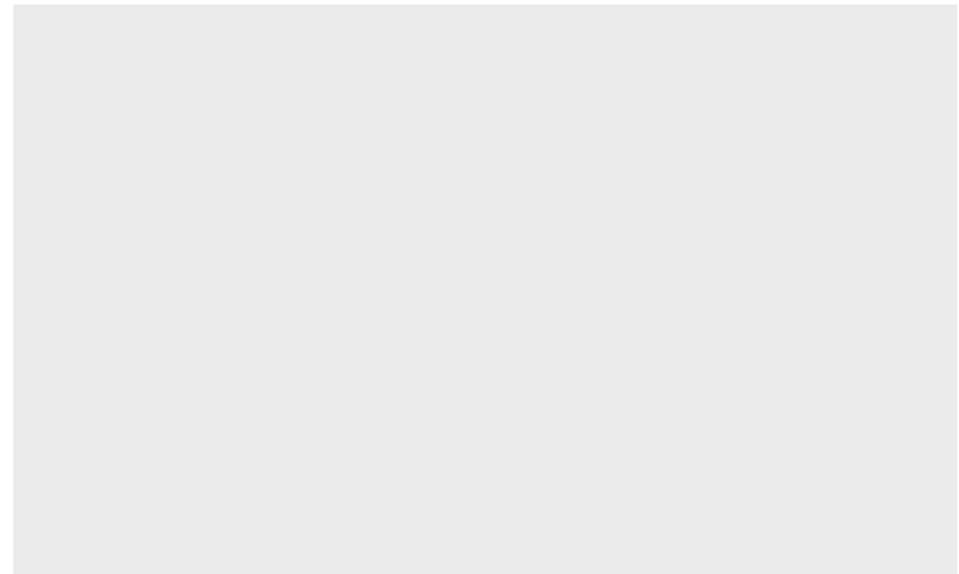
Data Science for Psychologists

Coding out loud



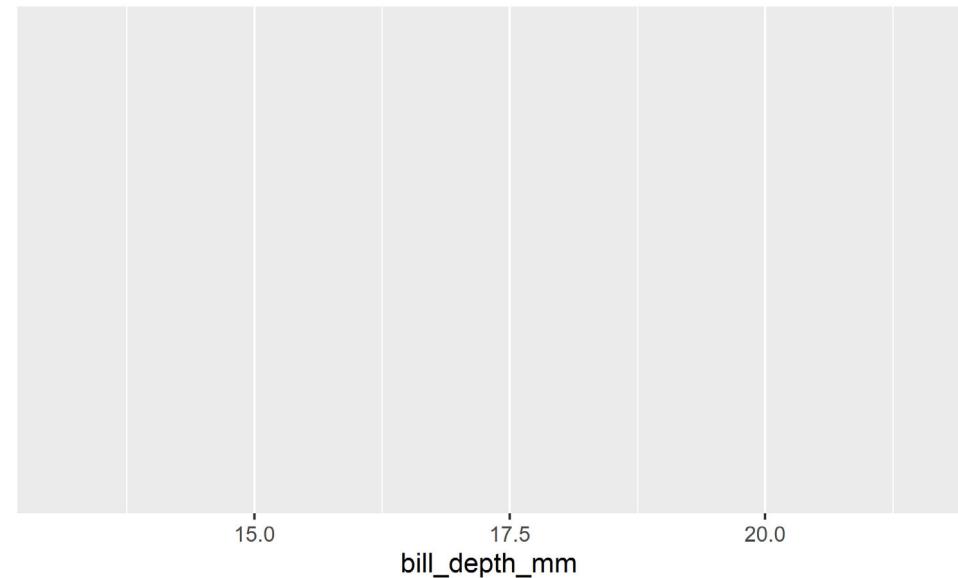
| Start with the penguins data frame

```
ggplot(data = penguins)
```



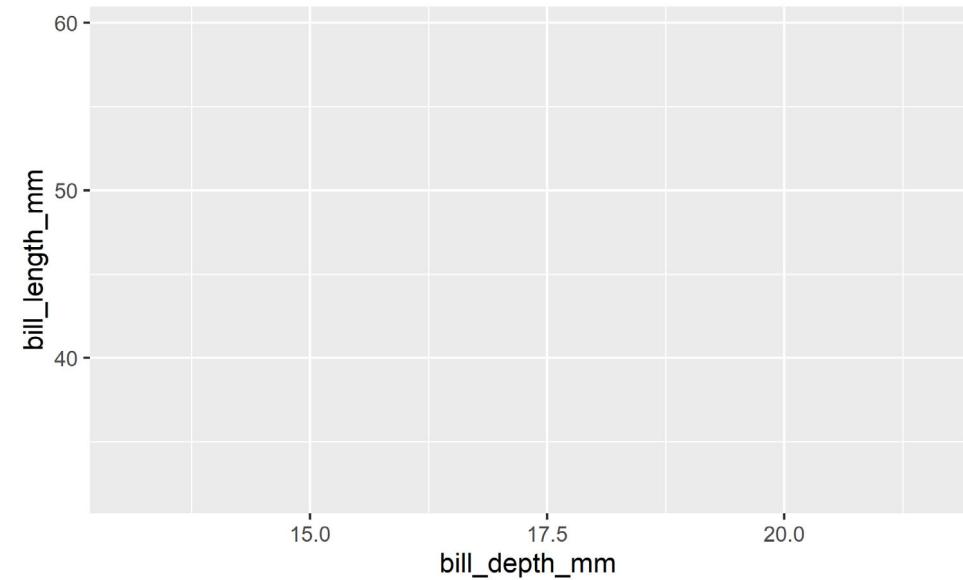
Start with the penguins data frame, **map bill depth to the x-axis**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth
```



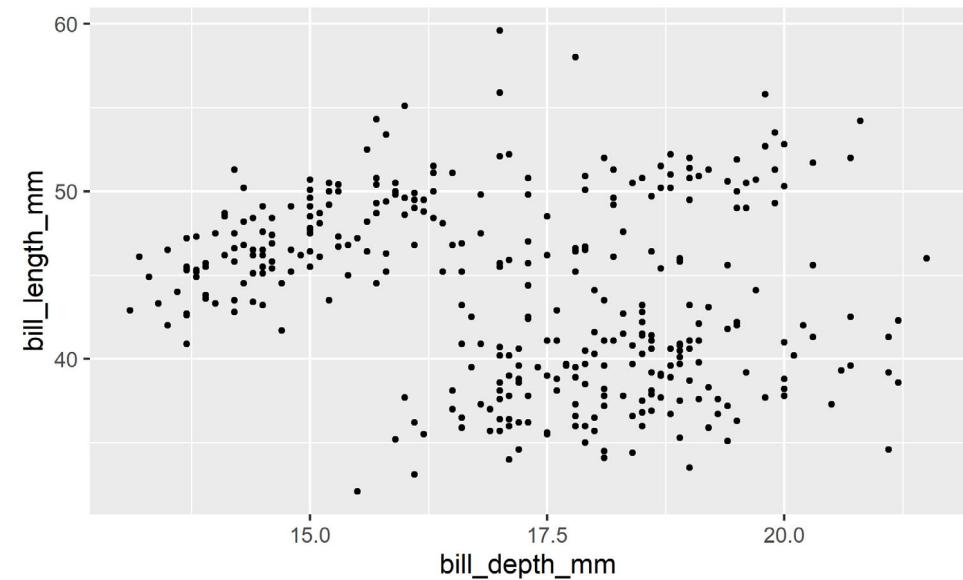
Start with the penguins data frame, map bill depth to the x-axis **and map bill length to the y-axis.**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length))
```



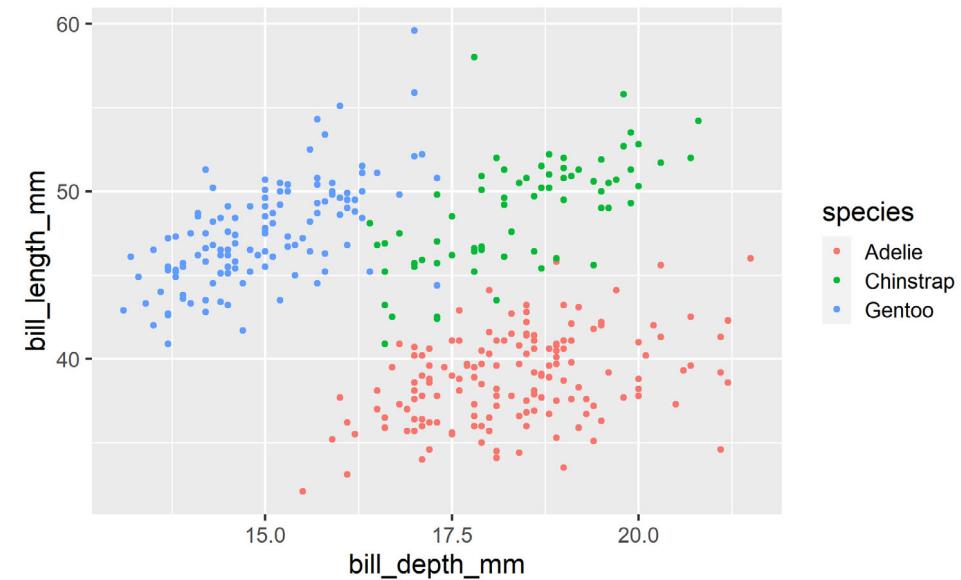
Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis. **Represent each observation with a point**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length))  
  geom_point()
```



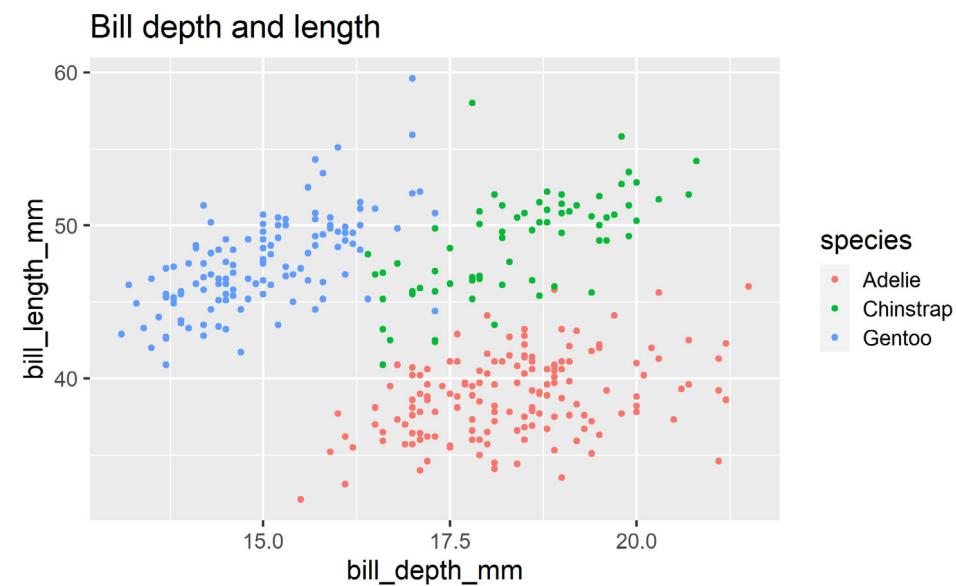
Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point **and map species to the color of each point.**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length  
                      color = species)  
       geom_point()
```



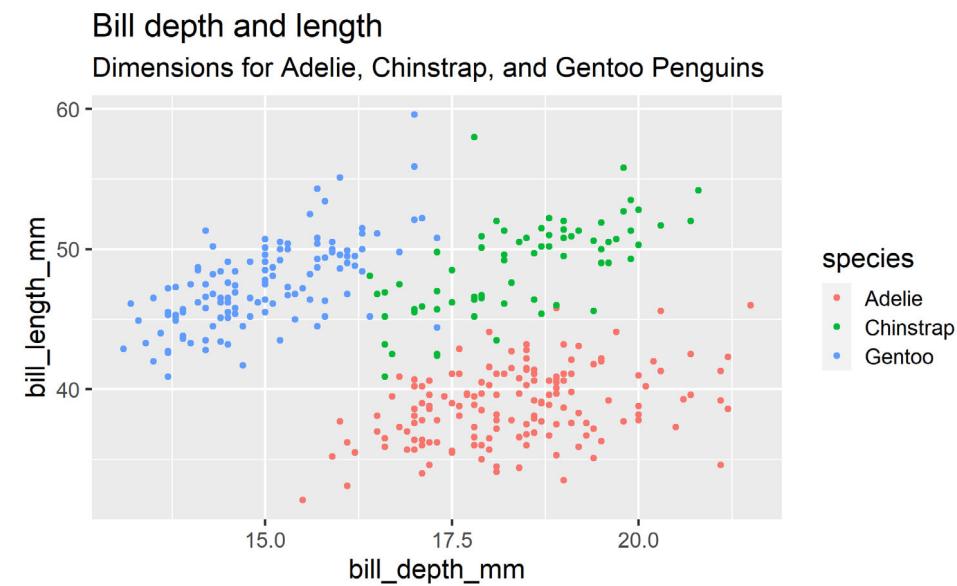
Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. **Title the plot "Bill depth and length"**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length  
                      color = species)) +  
  geom_point() +  
  labs(title = "Bill depth and length")
```



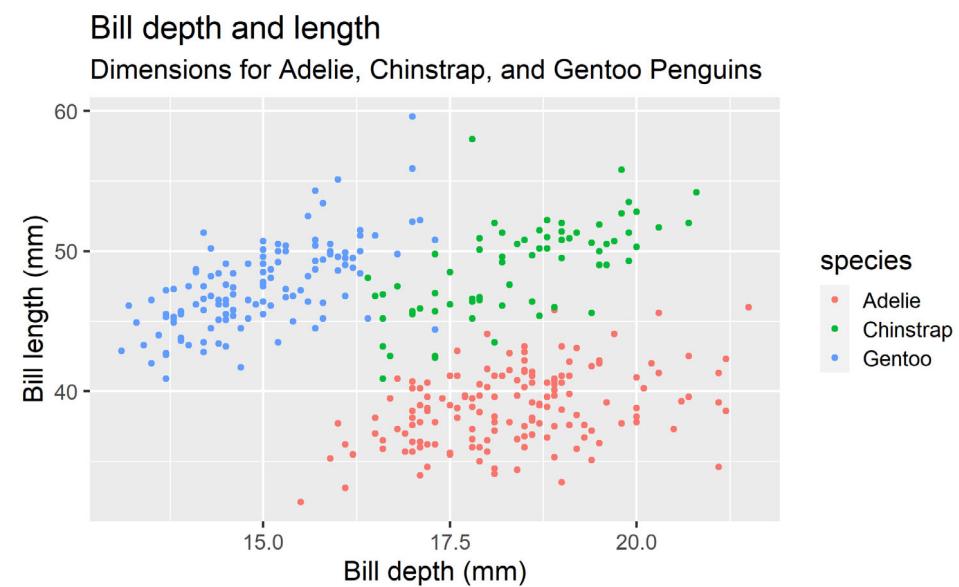
Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", **add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins"**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length  
                      color = species)) +  
  geom_point() +  
  labs(title = "Bill depth and length",  
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins")
```



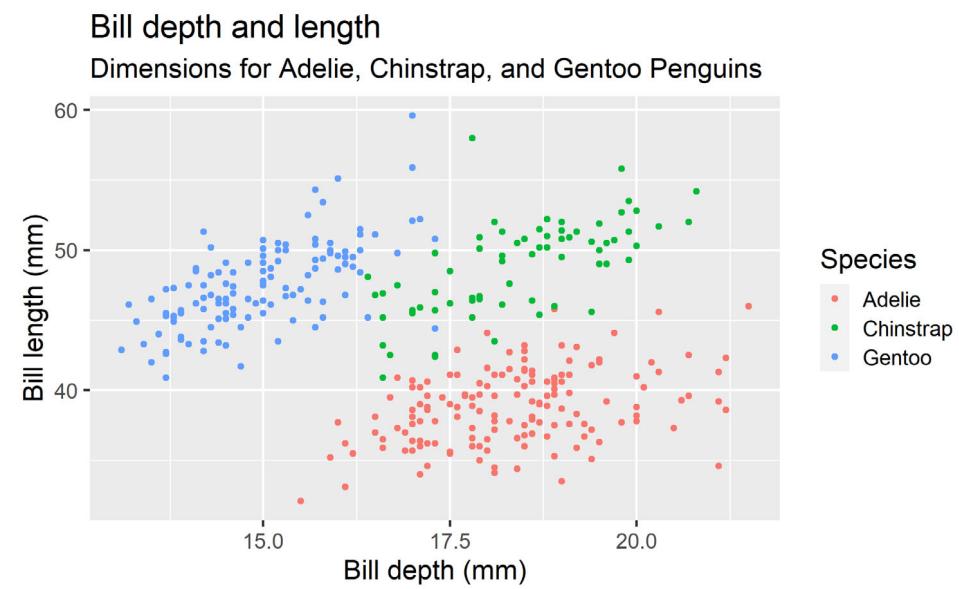
Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", **label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length  
                      color = species)) +  
  geom_point() +  
  labs(title = "Bill depth and length",  
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",  
       x = "Bill depth (mm)", y = "Bill length (mm)")
```



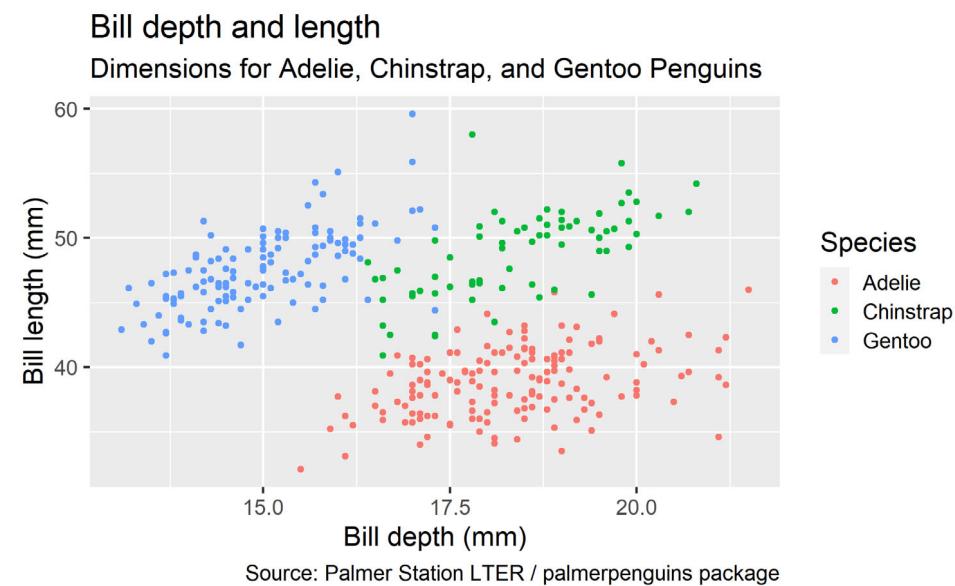
Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively, **label the legend "Species"**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length  
                      color = species)) +  
  geom_point() +  
  labs(title = "Bill depth and length",  
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",  
       x = "Bill depth (mm)", y = "Bill length (mm)",  
       color = "Species")
```



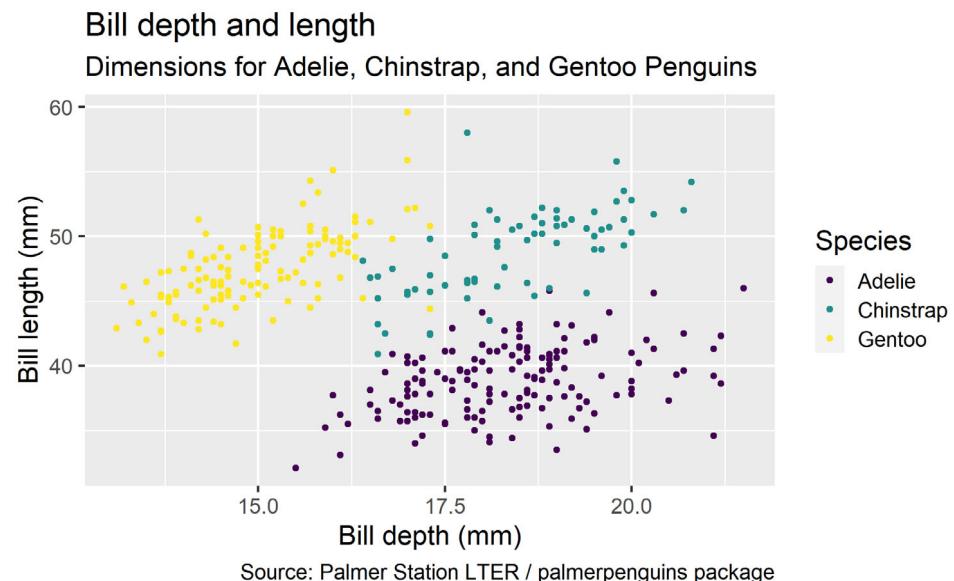
Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively, label the legend "Species", **and add a caption for the data source.**

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length  
                      color = species)) +  
  geom_point() +  
  labs(title = "Bill depth and length",  
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",  
       x = "Bill depth (mm)", y = "Bill length (mm)",  
       color = "Species",  
       caption = "Source: Palmer Station LTER / palmerpenguins package")
```



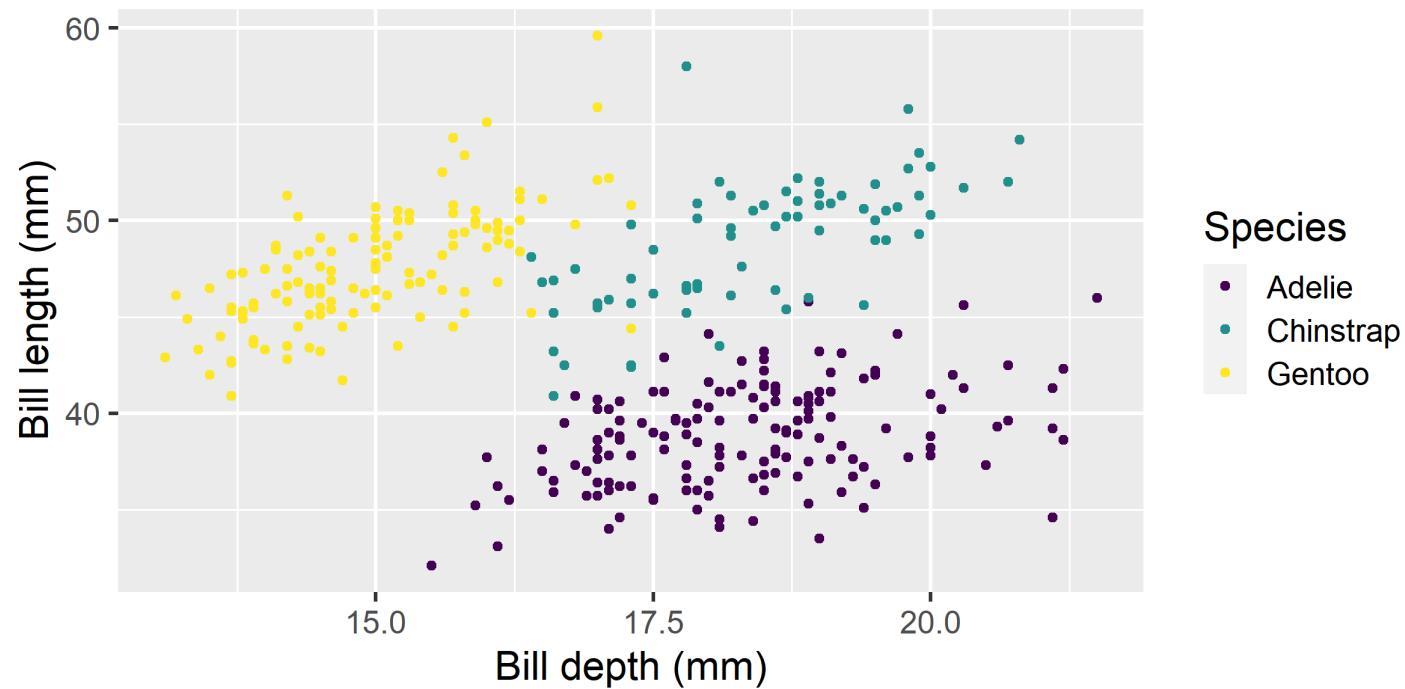
Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis. Represent each observation with a point and map species to the color of each point. Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively, label the legend "Species", and add a caption for the data source. **Finally, use a discrete color scale that is designed to be perceived by viewers with common forms of color blindness.**

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth,
                      y = bill_length,
                      color = species))
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",
       x = "Bill depth (mm)", y = "Bill length (mm)",
       color = "Species",
       caption = "Source: Palmer Station LTER / palmerpenguins package")
  scale_color_viridis_d()
```



Plot

Bill depth and length Dimensions for Adelie, Chinstrap, and Gentoo Penguins



Source: Palmer Station LTER / palmerpenguins package



Code

```
ggplot(data = penguins,
       mapping = aes(x = bill_depth_mm,
                      y = bill_length_mm,
                      color = species)) +
  geom_point() +
  labs(title = "Bill depth and length",
       subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",
       x = "Bill depth (mm)", y = "Bill length (mm)",
       color = "Species",
       caption = "Source: Palmer Station LTER / palmerpenguins package") +
  scale_color_viridis_d()
```

```
## Warning: Removed 2 rows containing missing values
## (`geom_point()`).
```



Narrative

- + Start with the penguins data frame, map bill depth to the x-axis and map bill length to the y-axis.
- + Represent each observation with a point and map species to the color of each point.
- + Title the plot "Bill depth and length", add the subtitle "Dimensions for Adelie, Chinstrap, and Gentoo Penguins", label the x and y axes as "Bill depth (mm)" and "Bill length (mm)", respectively, label the legend "Species", and add a caption for the data source.
- + Finally, use a discrete color scale that is designed to be perceived by viewers with common forms of color blindness.



Argument names

Tip:

You can omit the names of first two arguments when building plots with `ggplot()`.

```
ggplot(data = penguins,  
       mapping = aes(x = bill_depth  
                      y = bill_length  
                      color = species)) +  
  geom_point() +  
  scale_color_viridis_d()
```

```
ggplot(penguins,  
       aes(x = bill_depth_mm,  
            y = bill_length_mm,  
            color = species)) +  
  geom_point() +  
  scale_color_viridis_d()
```



Wrapping Up...



Data Science for Psychologists

Aesthetics



Data Science for Psychologists

Aesthetics options

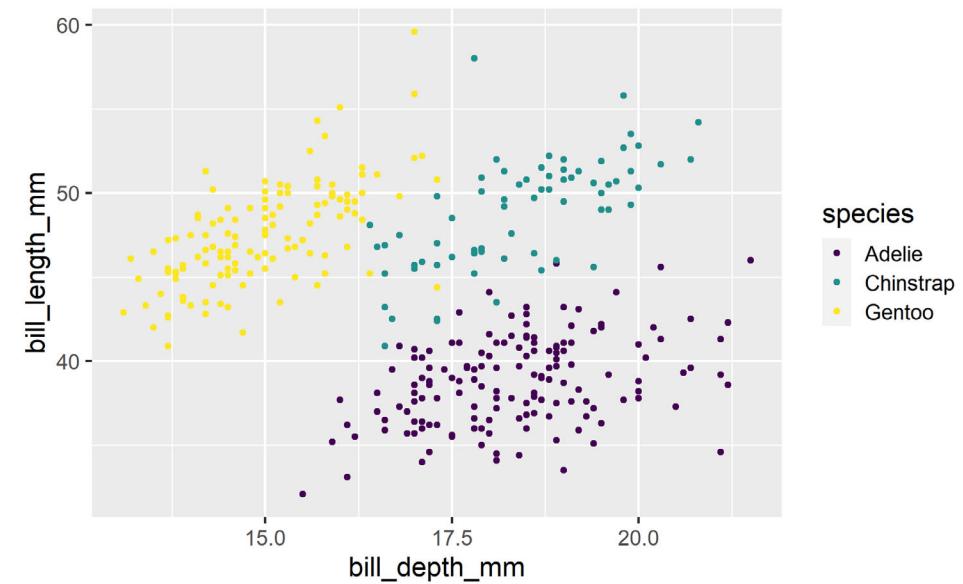
Commonly used characteristics of plotting characters that can be **mapped to a specific variable** in the data are

- + color
- + shape
- + size
- + alpha (transparency)



Color

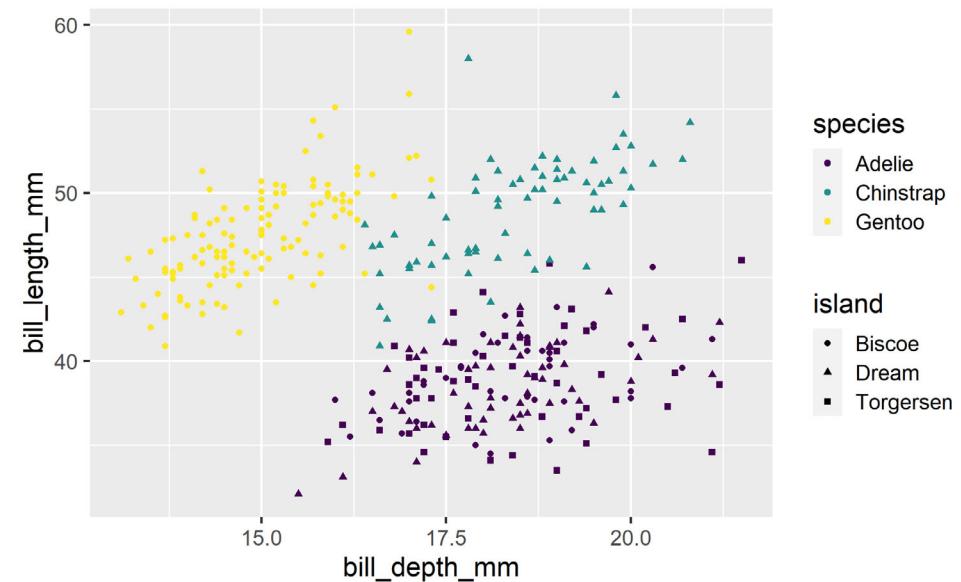
```
ggplot(penguins,  
       aes(x = bill_depth_mm,  
            y = bill_length_mm,  
            color = species)) +  
  geom_point() +  
  scale_color_viridis_d()
```



Shape

Mapped to a different variable than color

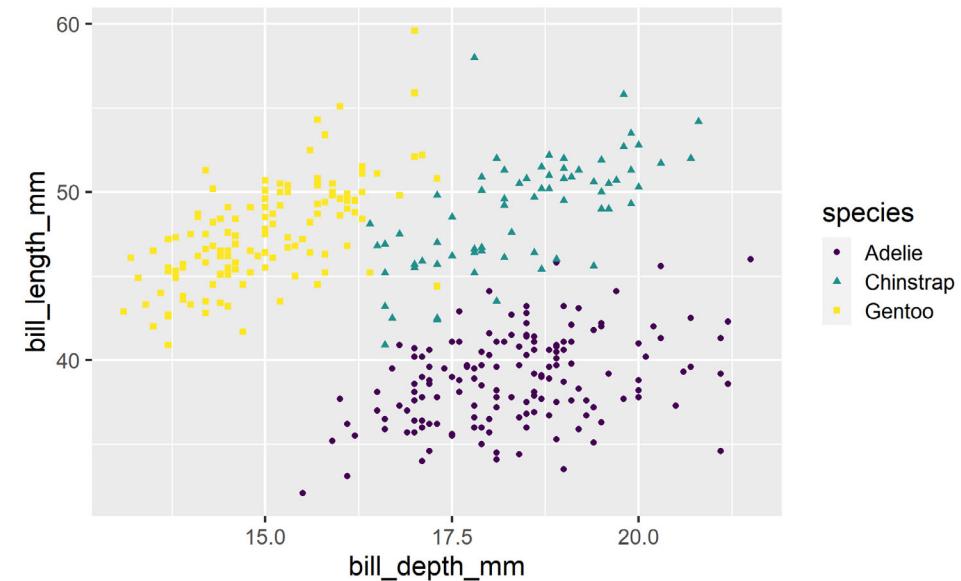
```
ggplot(penguins,  
       aes(x = bill_depth_mm,  
            y = bill_length_mm,  
            color = species,  
            shape = island)) +  
  geom_point() +  
  scale_color_viridis_d()
```



Shape

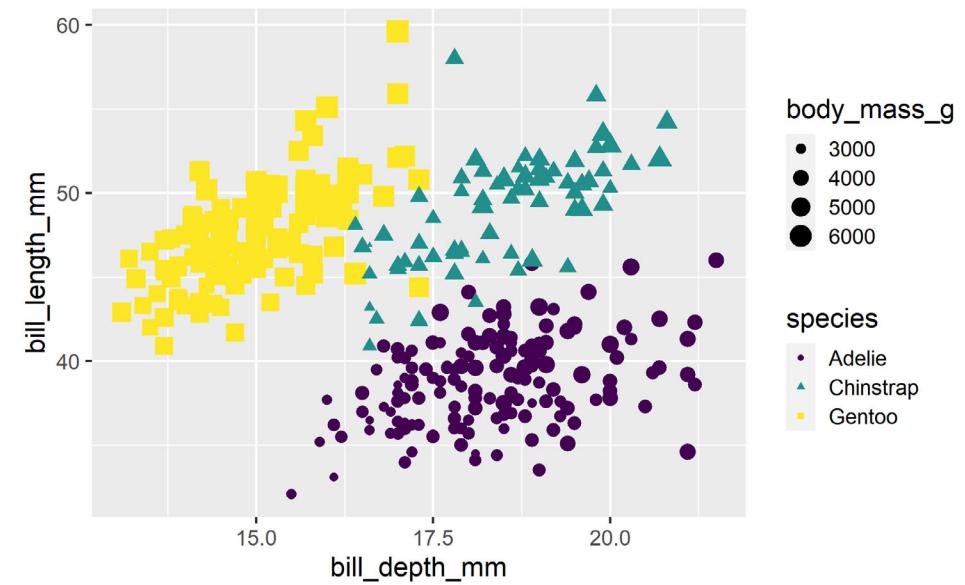
Mapped to same variable as color

```
ggplot(penguins,  
       aes(x = bill_depth_mm,  
            y = bill_length_mm,  
            color = species,  
            shape = species)) +  
  geom_point() +  
  scale_color_viridis_d()
```



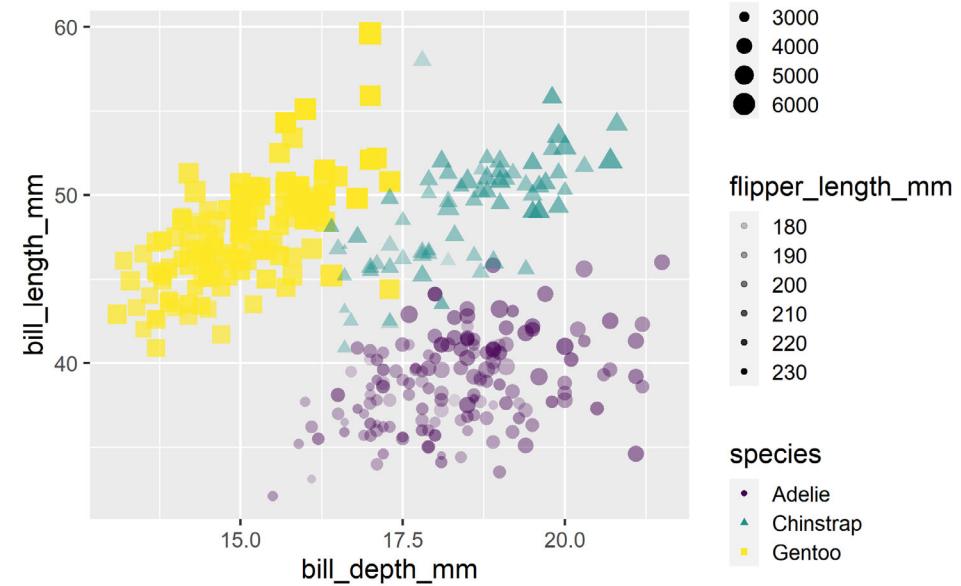
Size

```
ggplot(penguins,  
       aes(x = bill_depth_mm,  
            y = bill_length_mm,  
            color = species,  
            shape = species,  
            size = body_mass_g)) +  
  geom_point() +  
  scale_color_viridis_d()
```



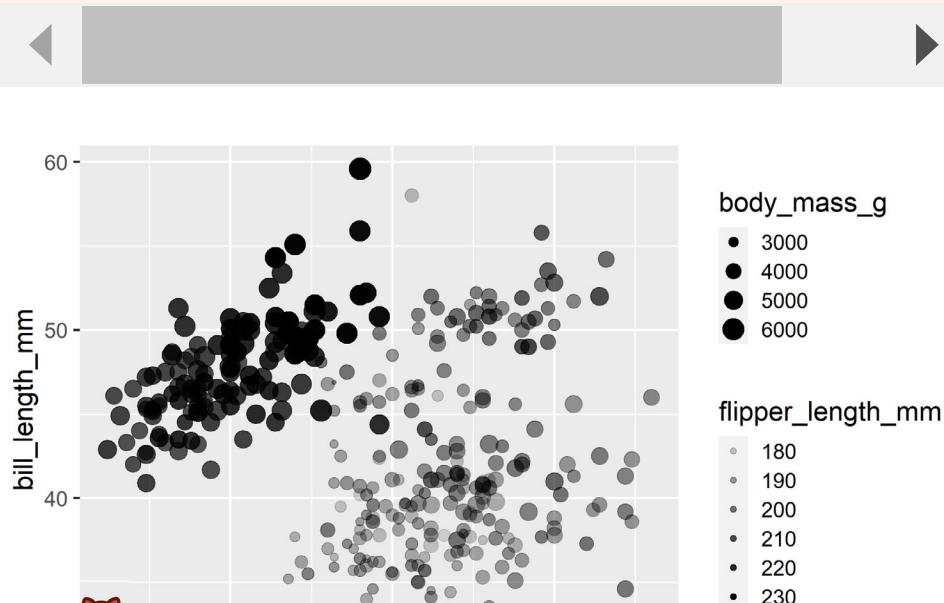
Alpha

```
ggplot(penguins,  
       aes(x = bill_depth_mm,  
            y = bill_length_mm,  
            color = species,  
            shape = species,  
            size = body_mass_g,  
            alpha = flipper_length_mm)  
       geom_point() +  
       scale_color_viridis_d()
```



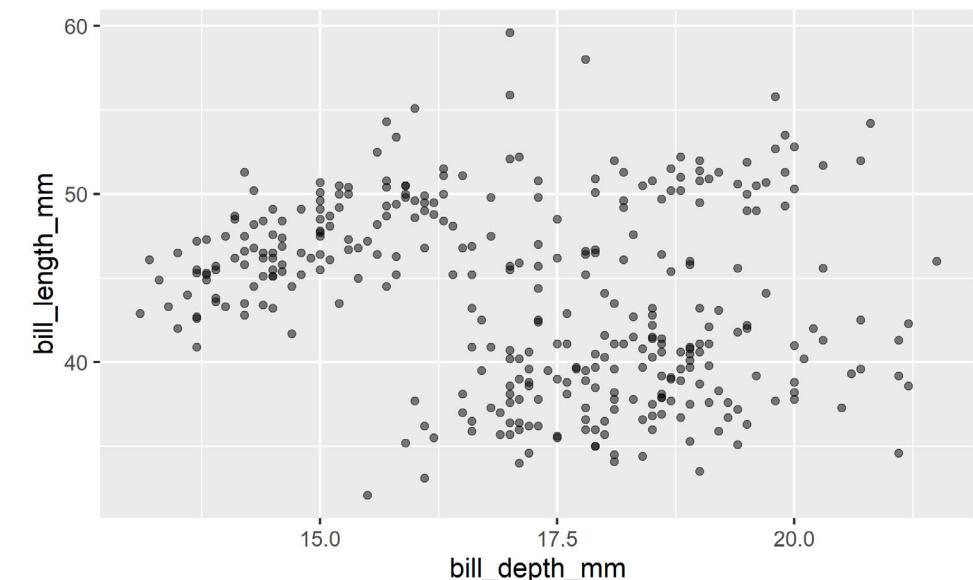
Mapping

```
ggplot(penguins,  
       aes(x = bill_depth_mm,  
            y = bill_length_mm,  
            size = body_mass_g,  
            alpha = flipper_length_mm)) +  
  geom_point()
```



Setting

```
ggplot(penguins,  
       aes(x = bill_depth_mm,  
            y = bill_length_mm)) +  
  geom_point(size = 2, alpha = 0.5)
```



Mapping vs. setting

- + **Mapping:** Determine the size, alpha, etc. of points based on the values of a variable in the data
 - + goes into aes()
- + **Setting:** Determine the size, alpha, etc. of points **not** based on the values of a variable in the data
 - + goes into geom_*()
 - + (in the previous example, we used geom_point() ,
 - + but we'll learn about other geoms soon!)



Faceting

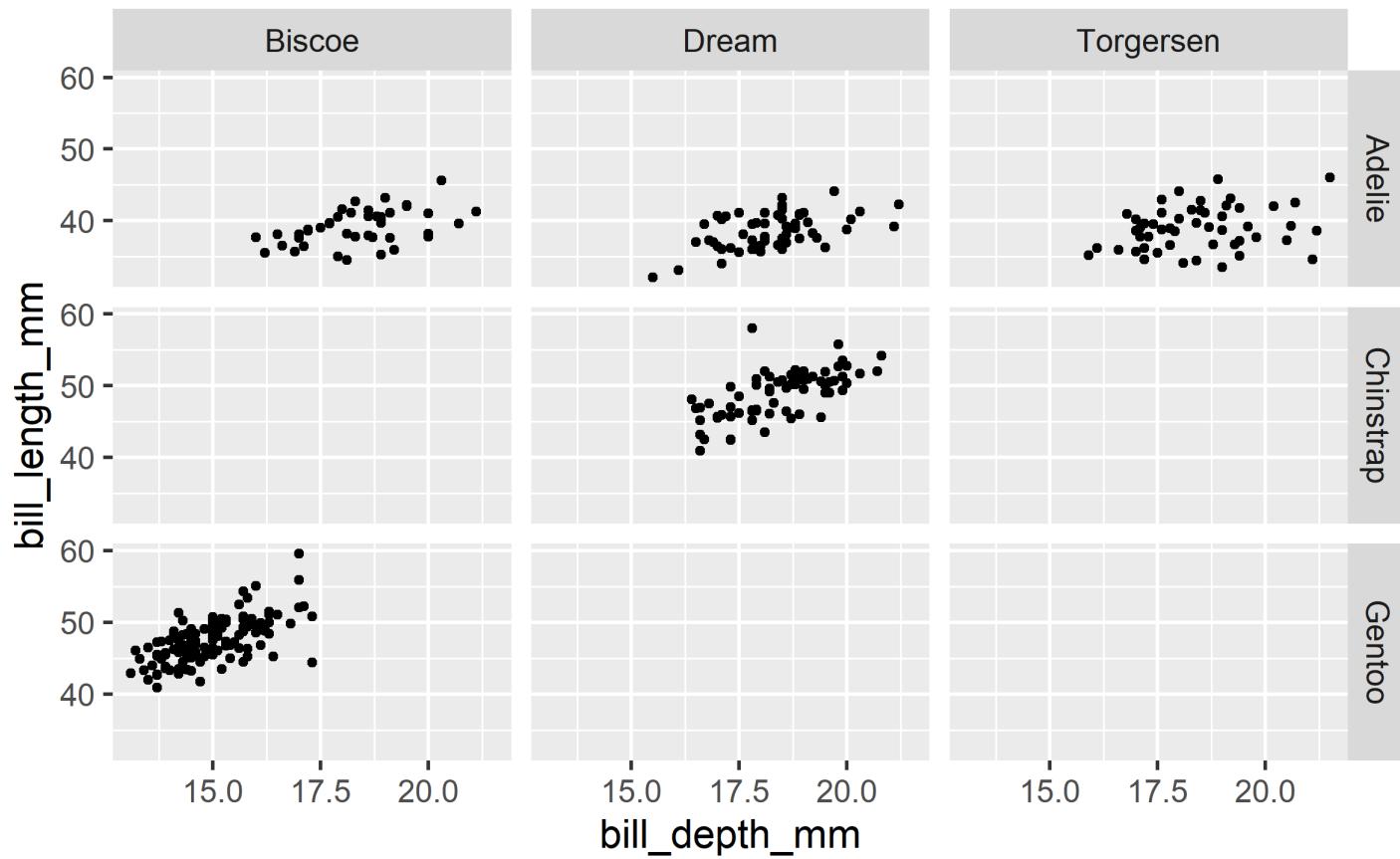


Faceting

- + Smaller plots that display different subsets of the data
- + Useful for exploring conditional relationships and large data



Plot



Code

```
ggplot(penguins, aes(x = bill_depth_mm, y = bill_length_mm)) +  
  geom_point() +  
  facet_grid(species ~ island)
```

```
## Warning: Removed 2 rows containing missing values  
## (`geom_point()`).
```



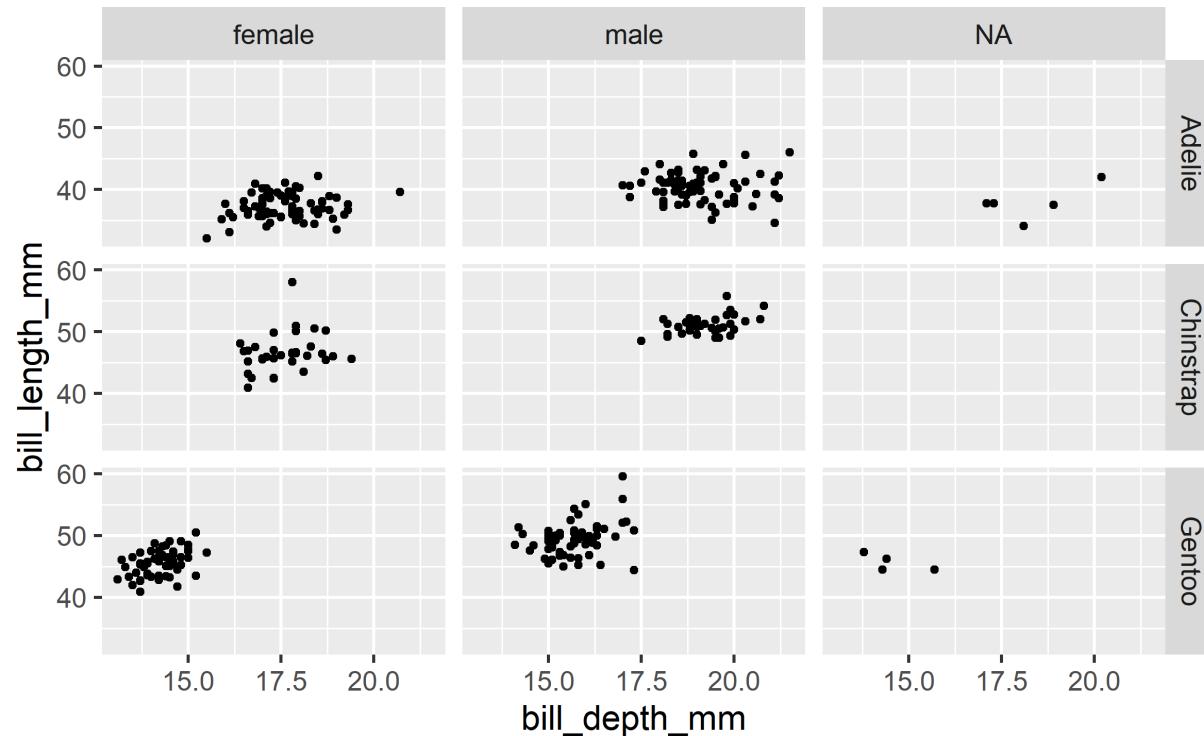
Various ways to facet

In the next few slides describe what each plot displays. Think about how the code relates to the output.

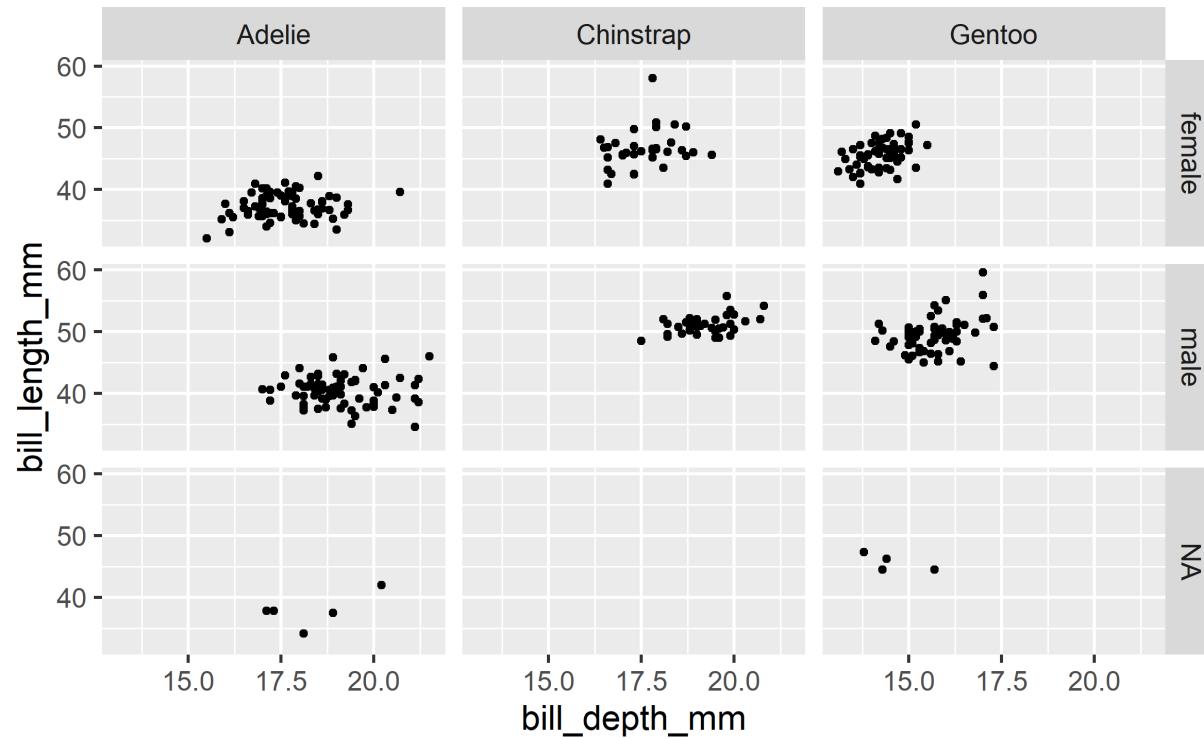
Note: The plots in the next few slides do not have proper titles, axis labels, etc. because we want you to figure out what's happening in the plots. But you should always label your plots!



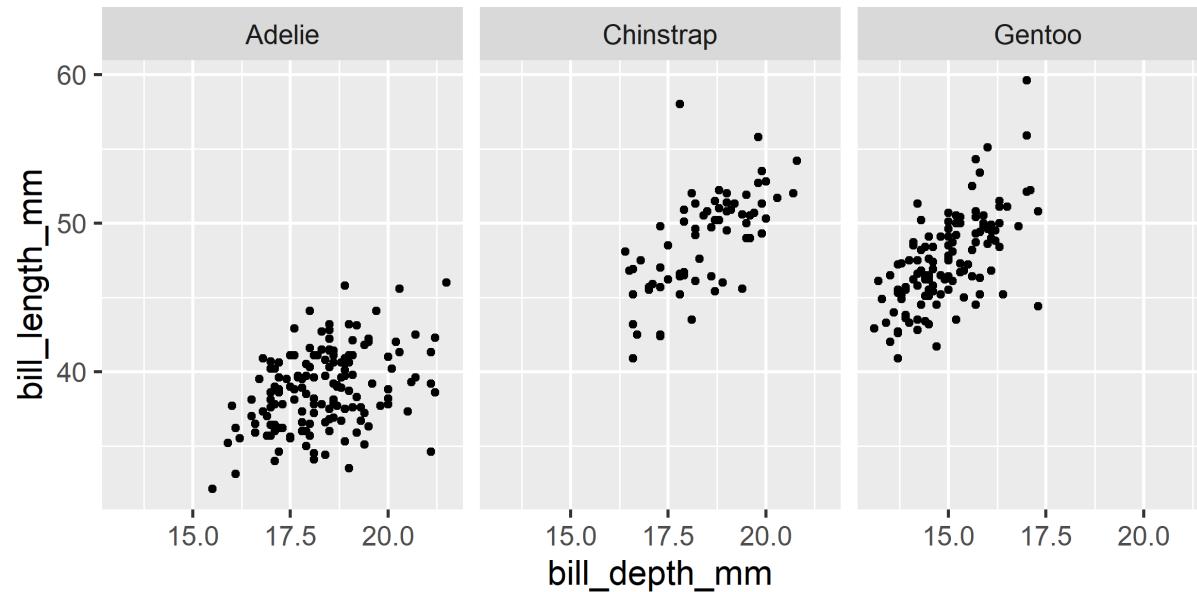
```
ggplot(penguins, aes(x = bill_depth_mm, y = bill_length_mm)) +  
  geom_point() +  
  facet_grid(species ~ sex)
```



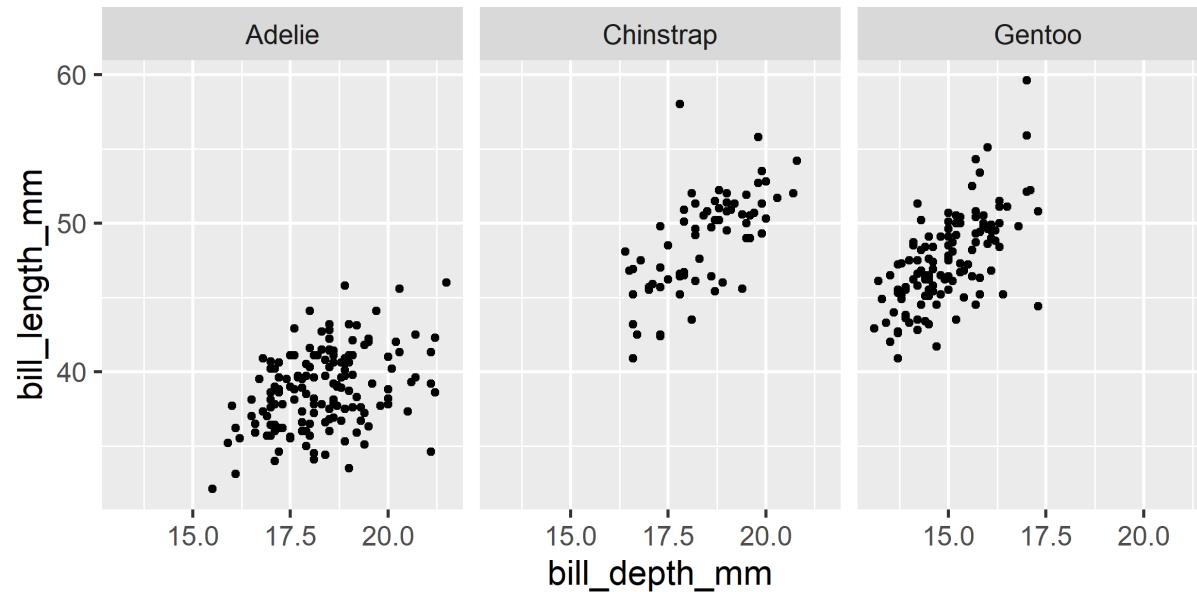
```
ggplot(penguins, aes(x = bill_depth_mm, y = bill_length_mm)) +  
  geom_point() +  
  facet_grid(sex ~ species)
```



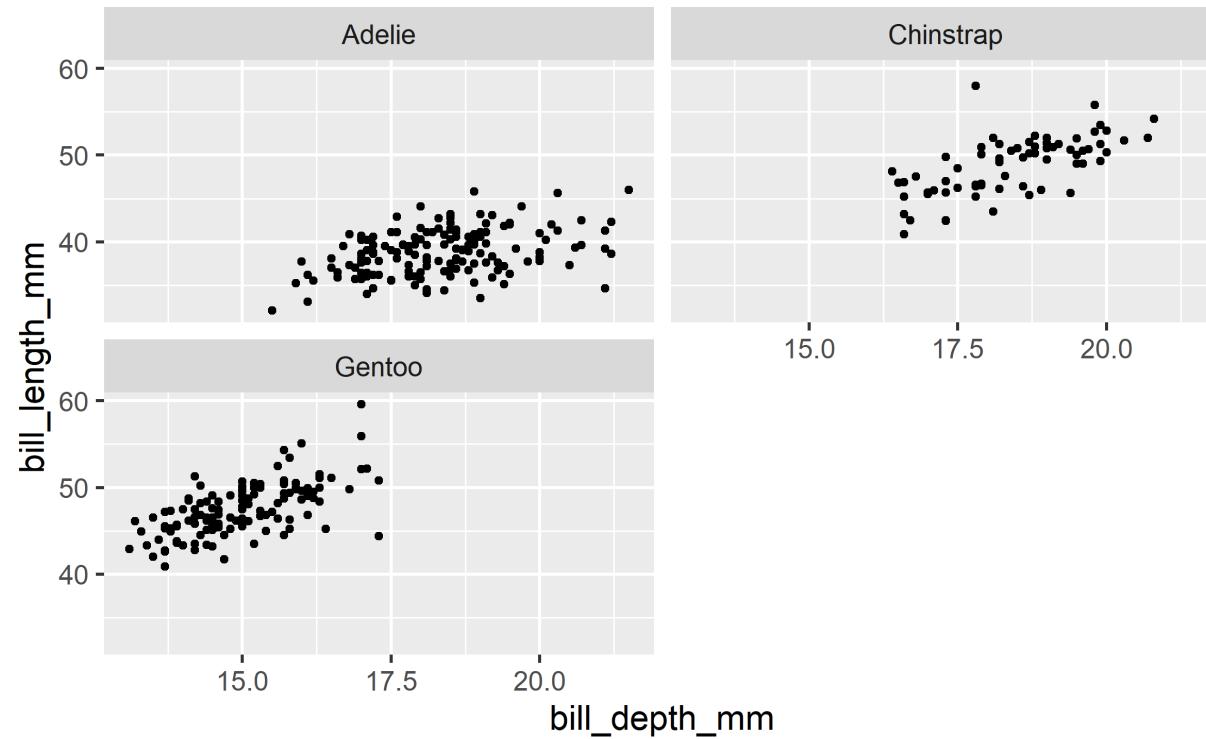
```
ggplot(penguins, aes(x = bill_depth_mm, y = bill_length_mm)) +  
  geom_point() +  
  facet_wrap(~ species)
```



```
ggplot(penguins, aes(x = bill_depth_mm, y = bill_length_mm)) +  
  geom_point() +  
  facet_grid(. ~ species)
```



```
ggplot(penguins, aes(x = bill_depth_mm, y = bill_length_mm)) +  
  geom_point() +  
  facet_wrap(~ species, ncol = 2)
```

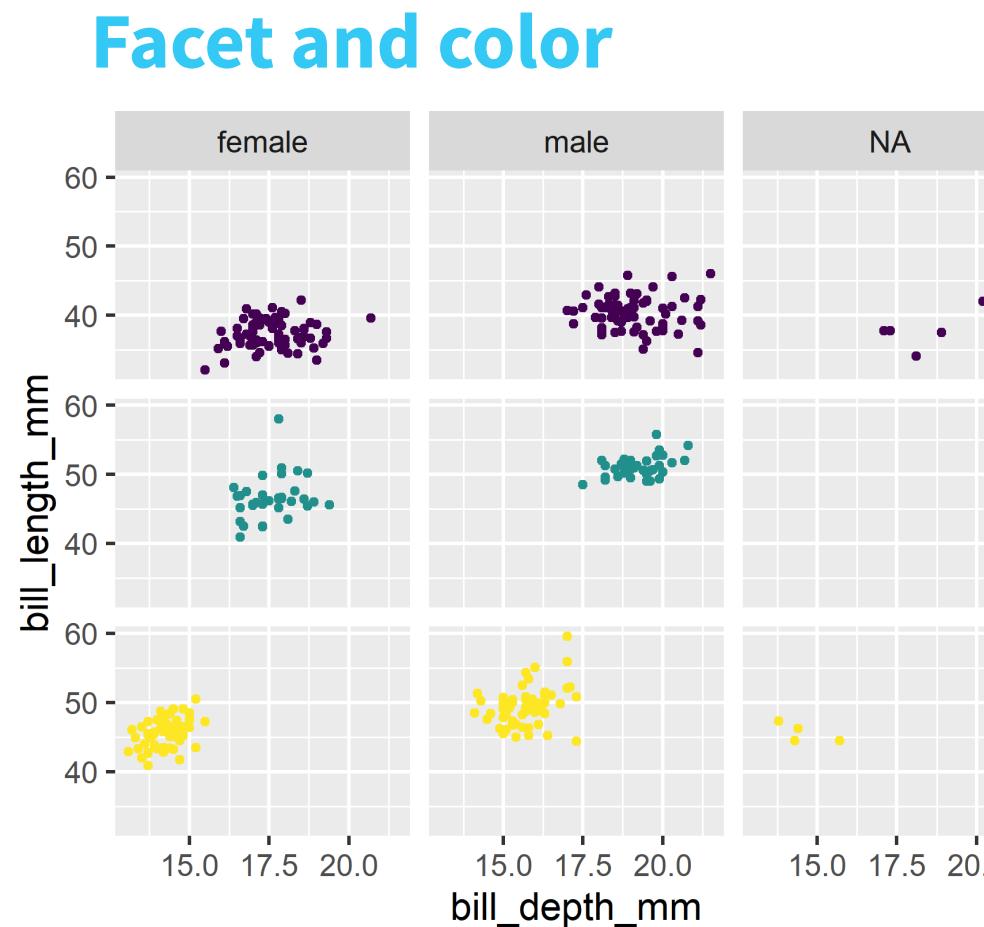


Faceting summary

- + `facet_grid()`:
 - + 2d grid
 - + `rows ~ cols`
 - + use `.` for no split
- + `facet_wrap()`: 1d ribbon wrapped according to number of rows and columns specified or available plotting area

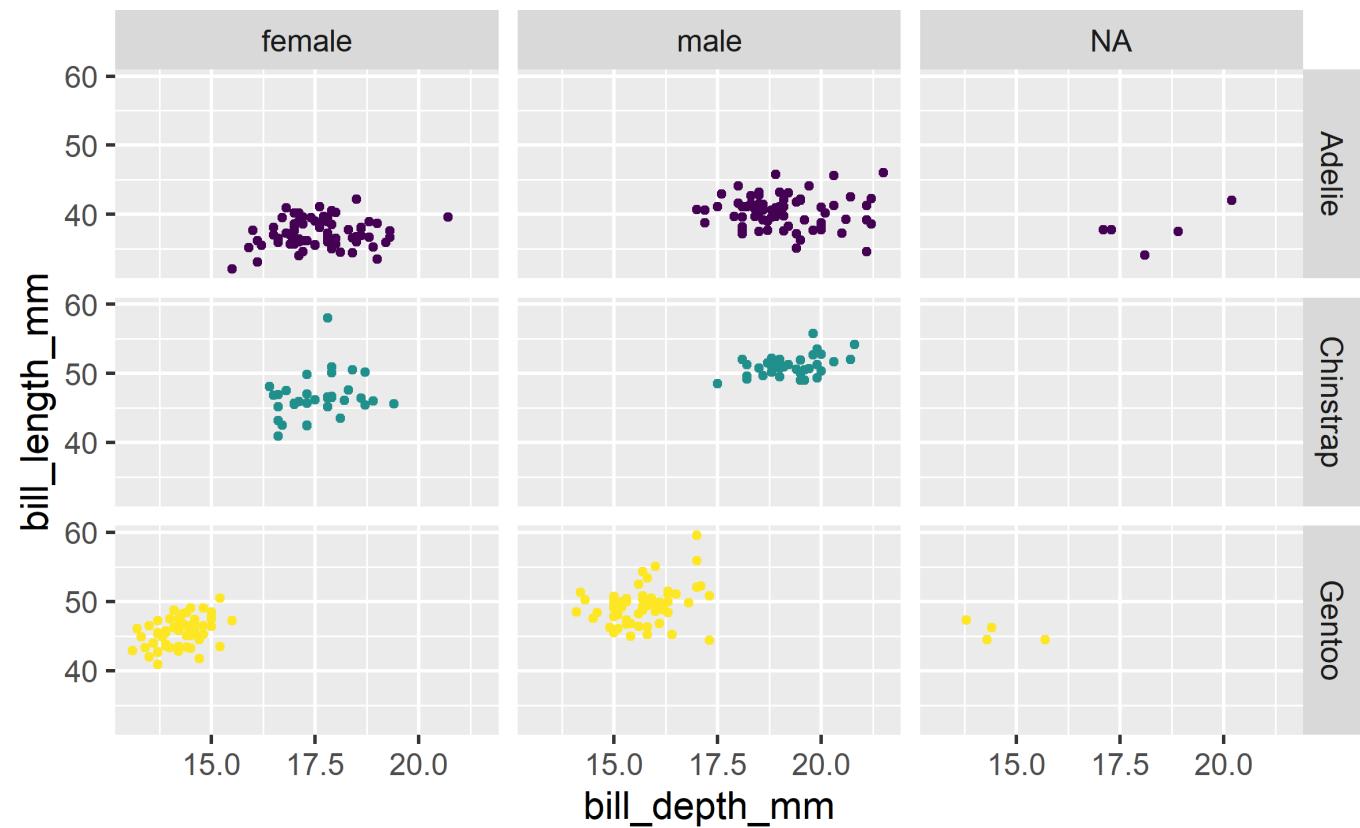


```
ggplot(  
  penguins,  
  aes(x = bill_depth_mm  
      , y = bill_length_mm  
      , color = species))  
  geom_point() +  
  facet_grid(species ~  
  scale_color_viridis)
```



Facet and color, no legend

```
ggplot(  
  penguins,  
  aes(x = bill_depth_mm  
      , y = bill_length_mm  
      , color = species))  
  geom_point() +  
  facet_grid(species ~  
  scale_color_viridis_c()  
  guides(color = FALSE)
```



Wrapping Up...



Data Science for Psychologists

Visualizing numerical data

12
34



Terminology



Data Science for Psychologists

Number of variables involved

- + Univariate data analysis - distribution of single variable
- + Bivariate data analysis - relationship between two variables
- + Multivariate data analysis - relationship between many variables at once,
 - + usually focusing on the relationship between two while conditioning for others



Types of variables

- + **Numerical variables** can be classified as **continuous** or **discrete** based on whether or not the variable can take on an infinite number of values or only non-negative whole numbers, respectively.
- + If the variable is **categorical**, we can determine if it is **ordinal** based on whether or not the levels have a natural ordering.



Data



Data Science for Psychologists

Data: Lending Club



- + Thousands of loans made through the Lending Club,
 - + a platform that allows individuals to lend to each other
- + Not all loans are created equal
 - + ease of getting a loan depends on (apparent) ability to repay the loan
- + Data includes loans *made*
 - + (these data are not loan applications)



Take a peek at data

```
library(openintro)  
glimpse(loans_full_schema)
```

```
## Rows: 10,000
## Columns: 55
## $ emp_title <chr> "global config enginee...
## $ emp_length <dbl> 3, 10, 3, 1, 10, NA, 1...
## $ state <fct> NJ, HI, WI, PA, CA, KY...
## $ homeownership <fct> MORTGAGE, RENT, RENT, ...
## $ annual_income <dbl> 90000, 40000, 40000, 3...
## $ verified_income <fct> Verified, Not Verified...
## $ debt_to_income <dbl> 18.01, 5.04, 21.15, 10...
## $ annual_income_joint <dbl> NA, NA, NA, NA, 57000, ...
## $ verification_income_joint <fct> , , , , Verified, , No...
## $ debt_to_income_joint <dbl> NA, NA, NA, NA, 37.66, ...
## $ delinq_2y <int> 0, 0, 0, 0, 0, 1, 0, 1...
## $ months_since_last_delinq <int> 38, NA, 28, NA, NA, 3...
```



Selected variables

```
loans <- loans_full_schema %>%
  select(loan_amount, interest_rate, term, grade,
         state, annual_income, homeownership, debt_to_income)
glimpse(loans)
```

```
## Rows: 10,000
## Columns: 8
## $ loan_amount    <int> 28000, 5000, 2000, 21600, 23000, 5000, 2...
## $ interest_rate   <dbl> 14.07, 12.61, 17.09, 6.72, 14.07, 6.72, ...
## $ term            <dbl> 60, 36, 36, 36, 36, 36, 60, 60, 36, 36, ...
## $ grade           <ord> C, C, D, A, C, A, C, B, C, A, C, B, C, B...
## $ state           <fct> NJ, HI, WI, PA, CA, KY, MI, AZ, NV, IL, ...
## $ annual_income   <dbl> 90000, 40000, 40000, 30000, 35000, 34000...
## $ homeownership   <fct> MORTGAGE, RENT, RENT, RENT, RENT, OWN, M...
## $ debt_to_income  <dbl> 18.01, 5.04, 21.15, 10.16, 57.96, 6.46, ...
```



Selected variables

variable	description
loan_amount	Amount of the loan received, in US dollars
interest_rate	Interest rate on the loan, in an annual percentage
term	The length of the loan, which is always set as a whole number of months
grade	Loan grade, which takes values A through G and represents the quality of the loan and its likelihood of being repaid
state	US state where the borrower resides
annual_income	Borrower's annual income, including any second income, in US dollars
homeownership	Indicates whether the person owns, owns but has a mortgage, or rents
debt_to_income	Debt-to-income ratio



Variable types

variable	type
loan_amount	numerical, continuous
interest_rate	numerical, continuous
term	numerical, discrete
grade	categorical, ordinal
state	categorical, not ordinal
annual_income	numerical, continuous
homeownership	categorical, not ordinal
debt_to_income	numerical, continuous



Wrapping Up...



Visualizing numerical data



Describing shapes of numerical distributions

- + shape:
 - + skewness: right-skewed, left-skewed, symmetric (skew is to the side of the longer tail)
 - + modality: unimodal, bimodal, multimodal, uniform
- + center: mean (mean), median (median), mode (not always useful)
- + spread: range (range), standard deviation (sd), inter-quartile range (IQR)
- + unusual observations



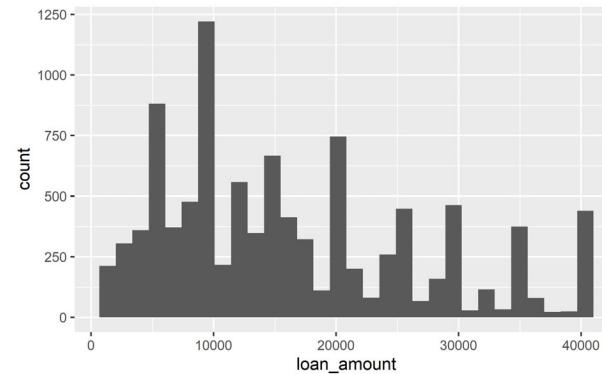
Histogram



Histogram

```
ggplot(loans, aes(x = loan_amount)) +  
  geom_histogram()
```

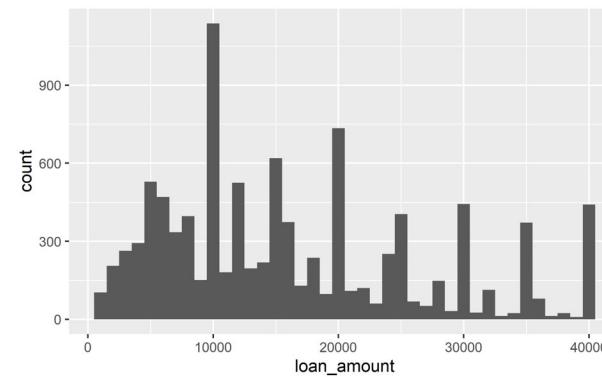
```
## `stat_bin()` using `bins = 30`. Pick better value with  
## `binwidth`.
```



Histograms and binwidth

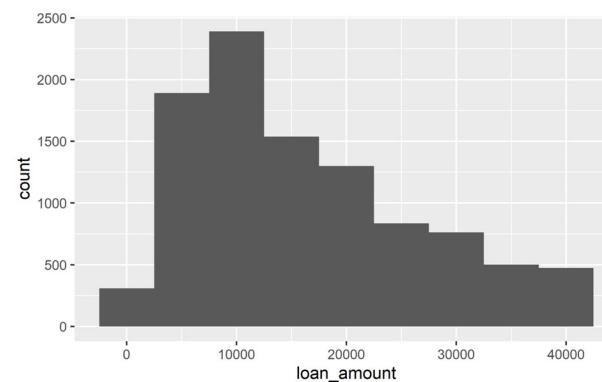
- + binwidth = 1000

```
ggplot(loans, aes(x = loan_amount)) +  
  geom_histogram(binwidth = 1000)
```



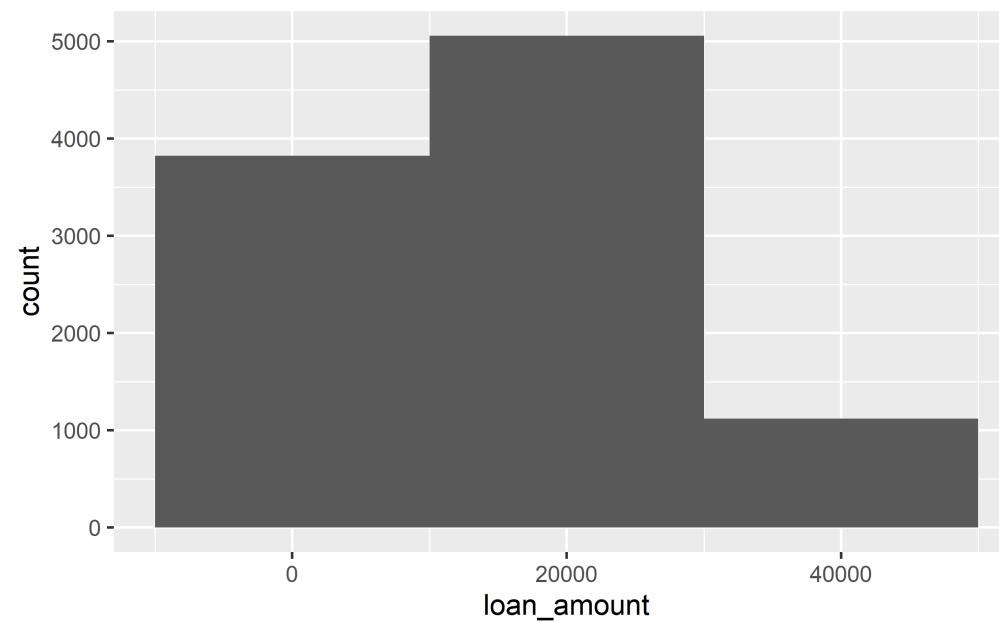
binwidth = 5000

```
ggplot(loans, aes(x = loan_amount)) +  
  geom_histogram(binwidth = 5000)
```

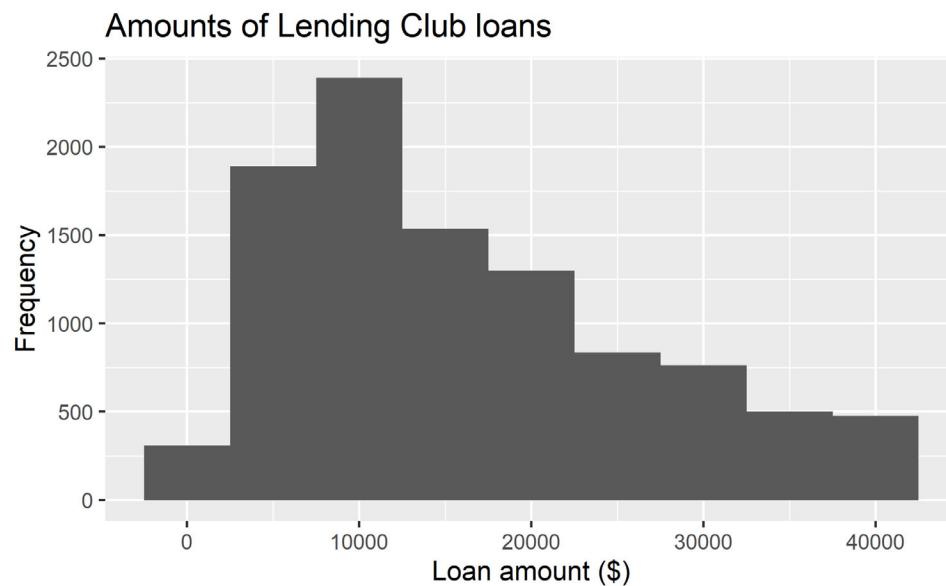


binwidth = 20000

```
ggplot(loans, aes(x = loan_amount)) +  
  geom_histogram(binwidth = 20000)
```



Customizing histograms

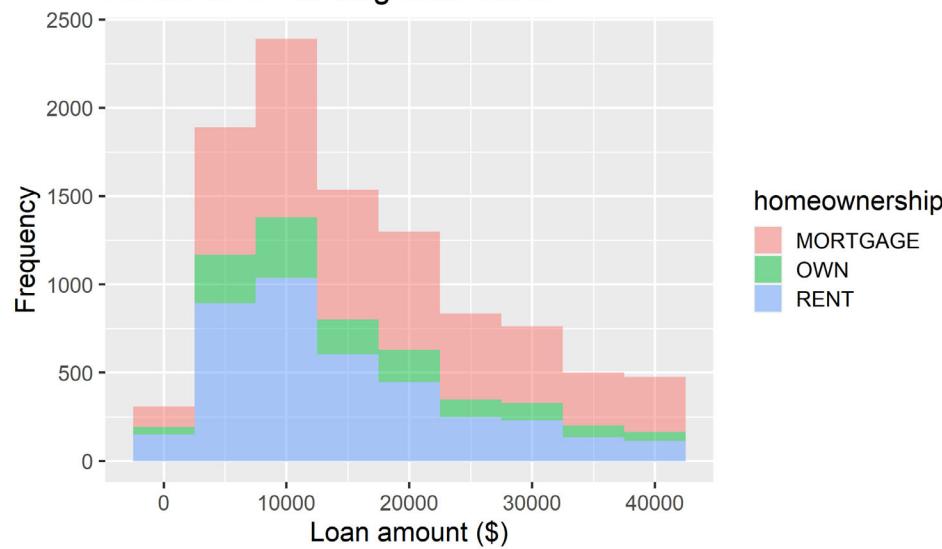


```
ggplot(loans, aes(x = loan_amount)) +  
  geom_histogram(binwidth = 5000) +  
  labs(  
    x = "Loan amount ($)",  
    y = "Frequency",  
    title = "Amounts of Lending Club loans"  
  )
```



Fill with a categorical variable

Amounts of Lending Club loans

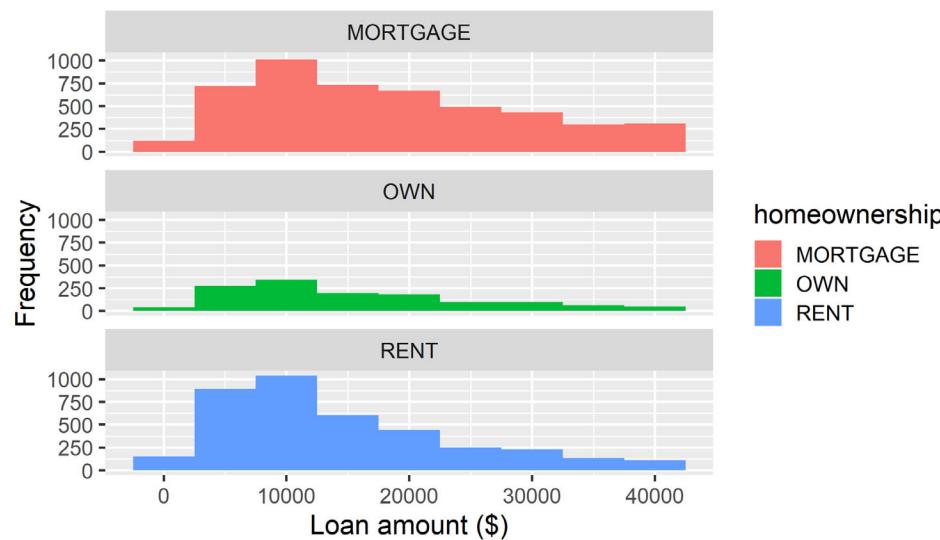


```
ggplot(loans, aes(x = loan_amount,  
                  fill = homeownership)) +  
  geom_histogram(binwidth = 5000,  
                 alpha = 0.5) +  
  labs(  
    x = "Loan amount ($)",  
    y = "Frequency",  
    title = "Amounts of Lending Club loans"  
)
```



Facet with a categorical variable

Amounts of Lending Club loans



```
ggplot(loans, aes(x = loan_amount, fill = homeownership)) +  
  geom_histogram(binwidth = 5000) +  
  labs(  
    x = "Loan amount ($)",  
    y = "Frequency",  
    title = "Amounts of Lending Club loans"  
) +  
  facet_wrap(~ homeownership, nrow = 3)
```

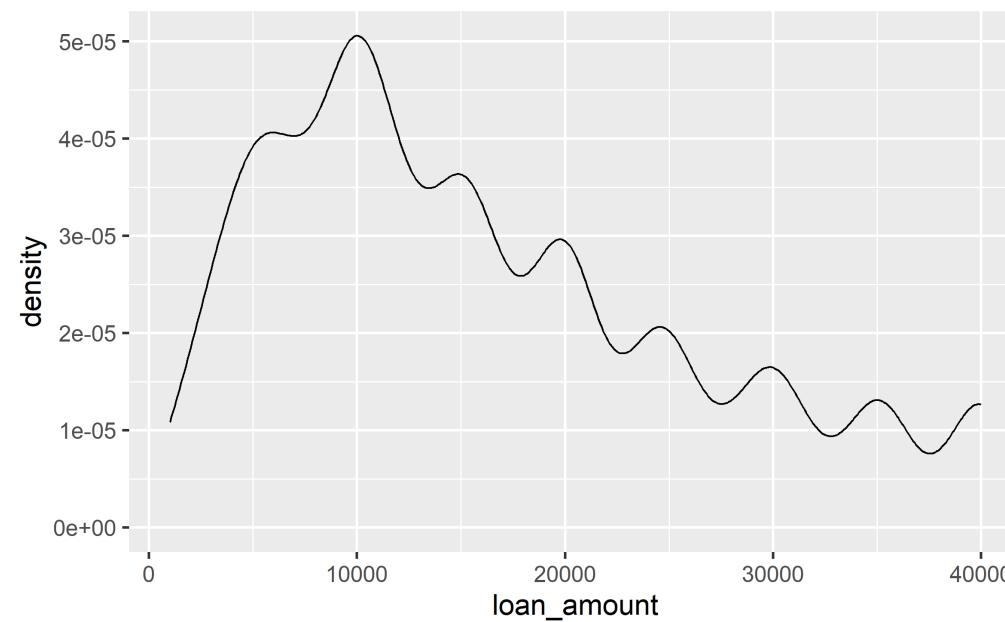


Density plot



Density plot

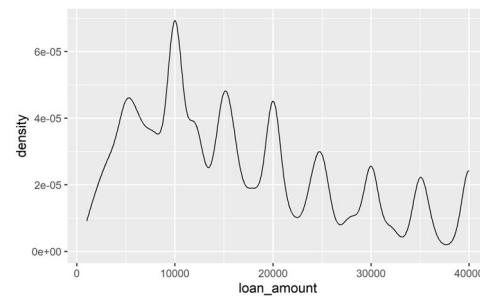
```
ggplot(loans, aes(x = loan_amount)) +  
  geom_density()
```



Density plots and adjusting bandwidth

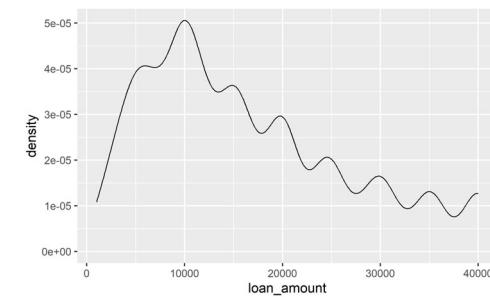
+ `adjust = 0.5`

```
ggplot(loans, aes(x = loan_amount))  
  geom_density(adjust = 0.5)
```



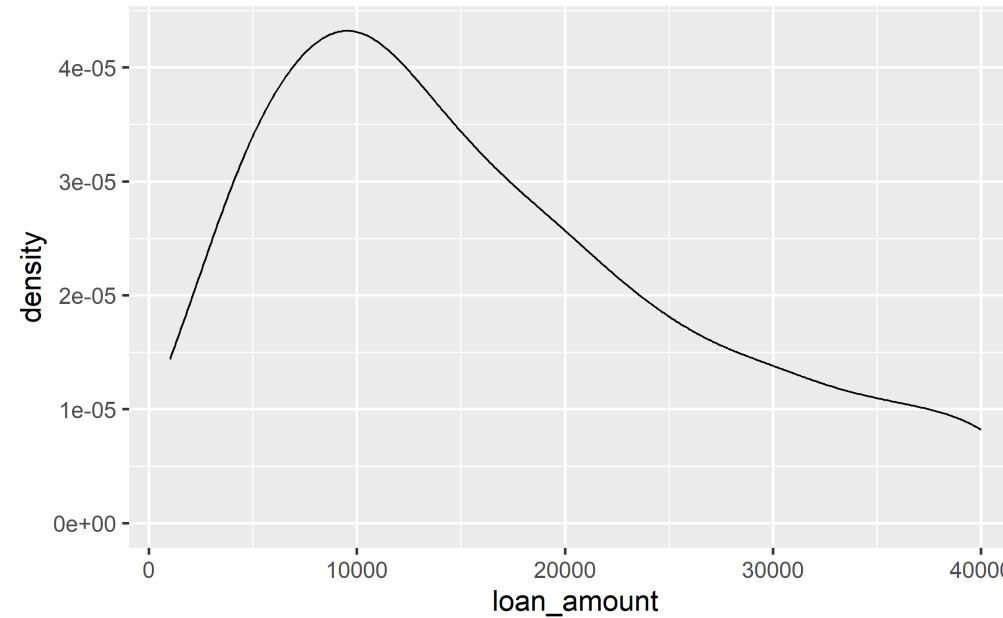
+ `adjust = 1`

```
ggplot(loans, aes(x = loan_amount))  
  geom_density(adjust = 1) # default
```



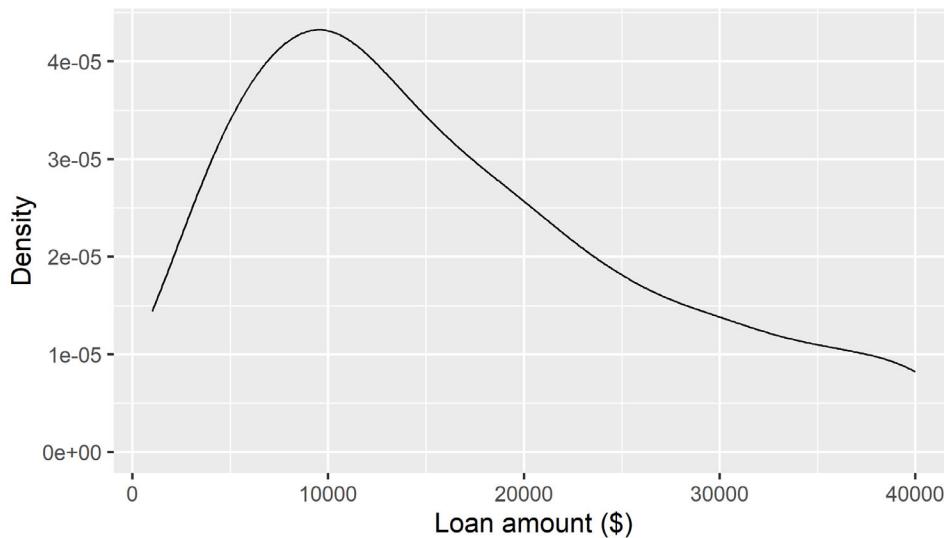
adjust = 2

```
ggplot(loans, aes(x = loan_amount)) +  
  geom_density(adjust = 2)
```



Customizing density plots

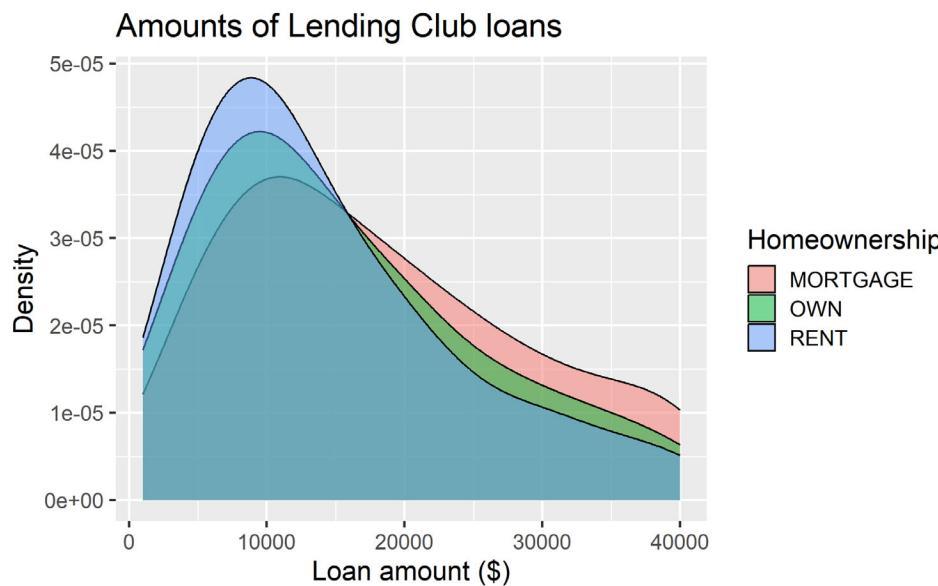
Amounts of Lending Club loans



```
ggplot(loans, aes(x = loan_amount)) +  
  geom_density(adjust = 2) +  
  labs(  
    x = "Loan amount ($)",  
    y = "Density",  
    title = "Amounts of Lending Club loans"  
  )
```



Adding a categorical variable



```
ggplot(loans, aes(x = loan_amount,  
                  fill = homeownership)) +  
  geom_density(adjust = 2,  
              alpha = 0.5) +  
  labs(  
    x = "Loan amount ($)",  
    y = "Density",  
    title = "Amounts of Lending Club loans",  
    fill = "Homeownership"  
)
```

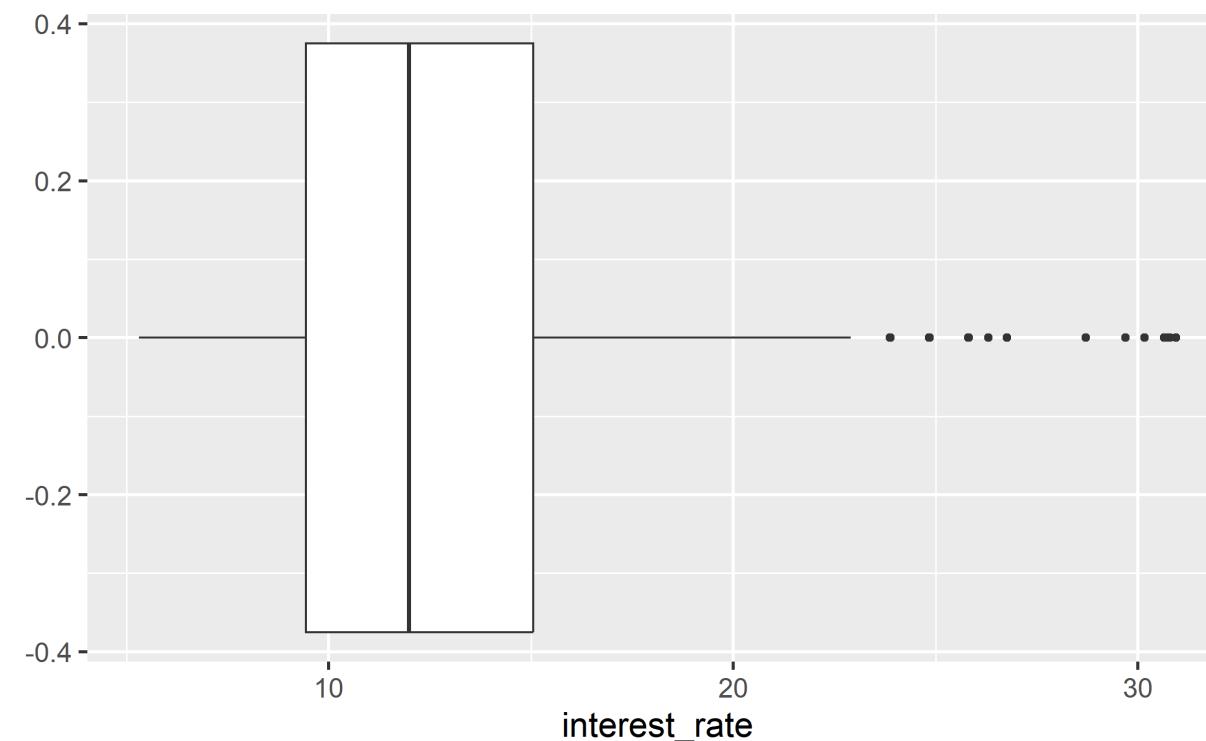


Box plot



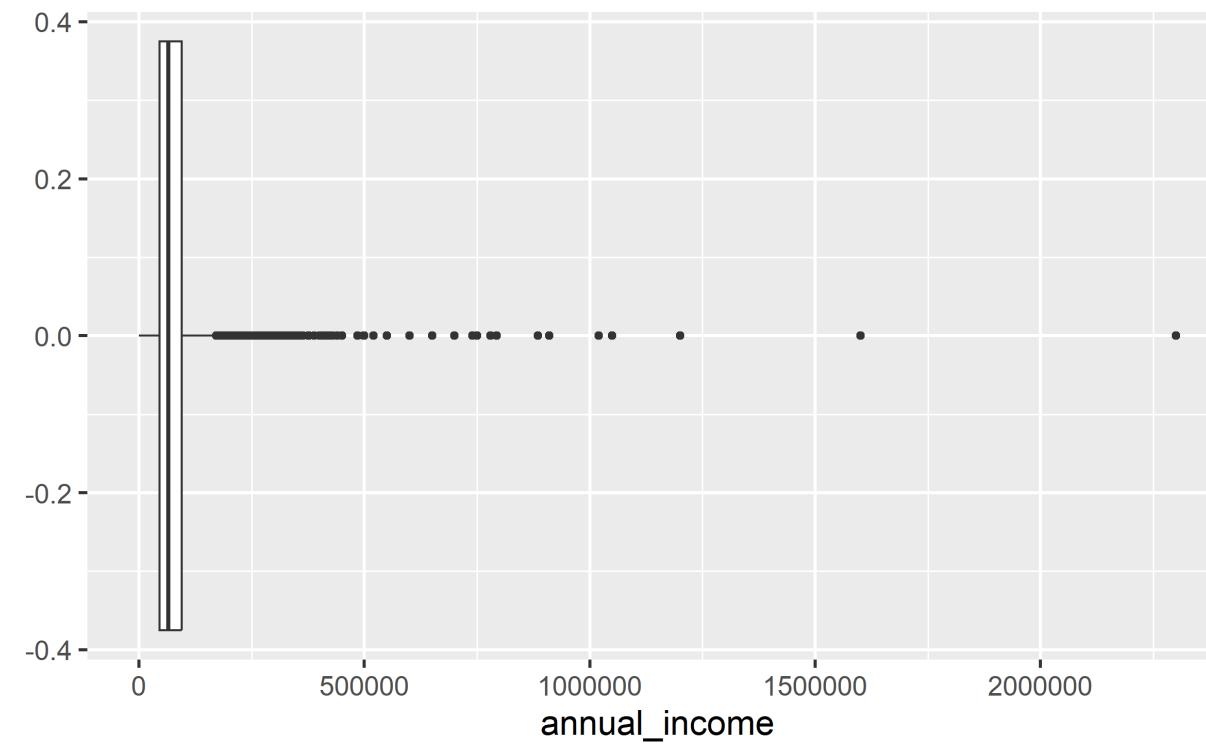
Box plot

```
ggplot(loans, aes(x = interest_rate)) +  
  geom_boxplot()
```



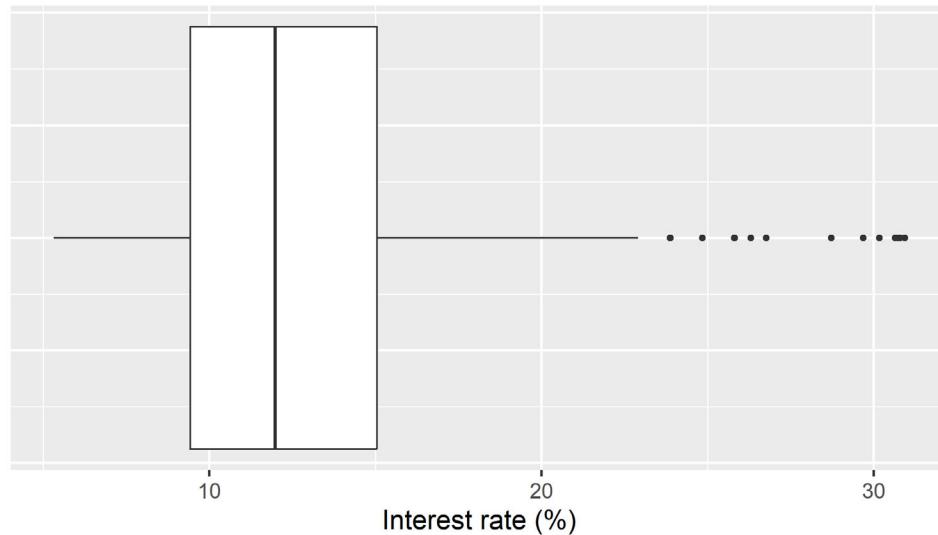
Box plot and outliers

```
ggplot(loans, aes(x = annual_income)) +  
  geom_boxplot()
```



Customizing box plots

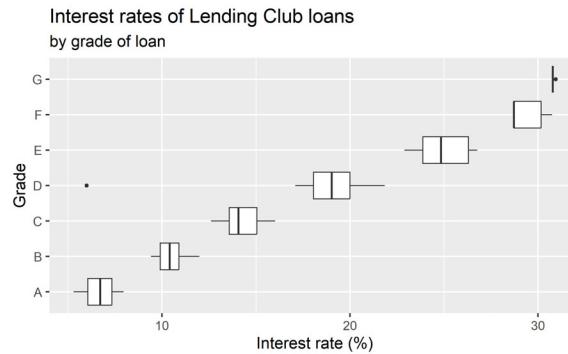
Interest rates of Lending Club loans



```
ggplot(loans, aes(x = interest_rate)) +  
  geom_boxplot() +  
  labs(  
    x = "Interest rate (%)",  
    y = NULL,  
    title = "Interest rates of Lending Club loans"  
) +  
  theme(  
    axis.ticks.y = element_blank(),  
    axis.text.y = element_blank()  
)
```



Adding a categorical variable



```
ggplot(loans, aes(x = interest_rate,  
                   y = grade)) +  
  geom_boxplot() +  
  labs(  
    x = "Interest rate (%)",  
    y = "Grade",  
    title = "Interest rates of Lending Club loans",  
    subtitle = "by grade of loan"  
)
```

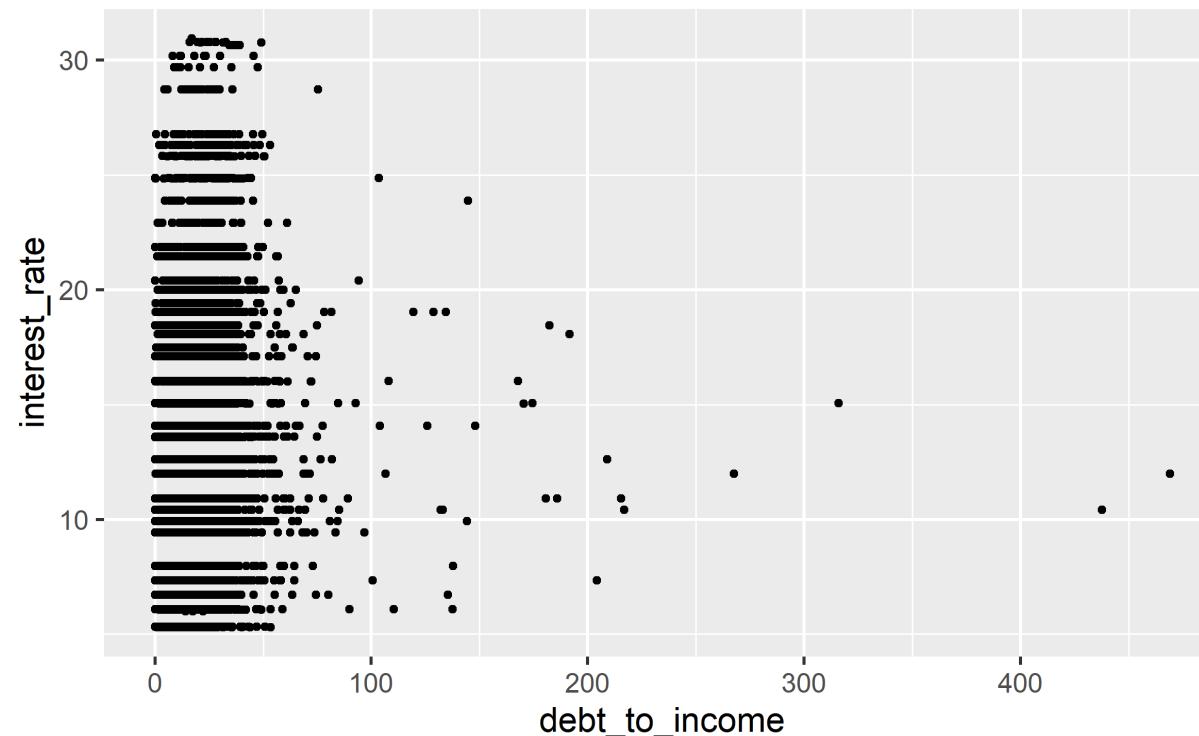


Relationships numerical variables



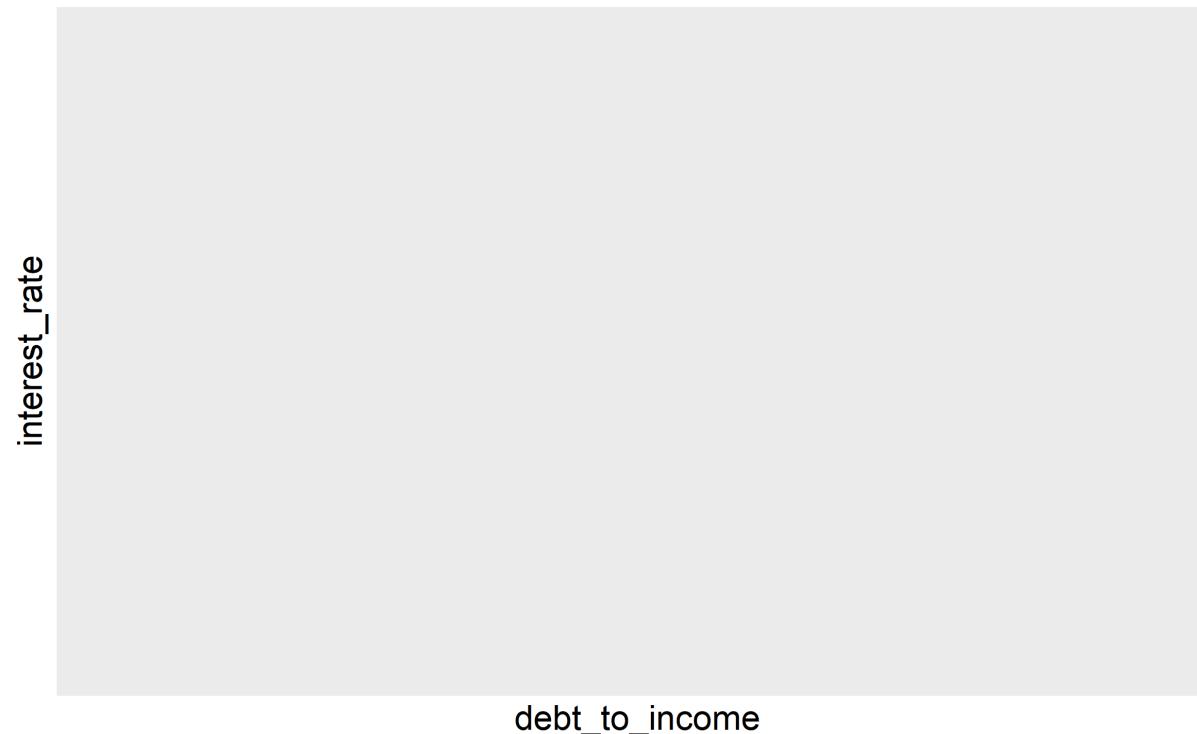
Scatterplot

```
ggplot(loans, aes(x = debt_to_income, y = interest_rate)) +  
  geom_point()
```



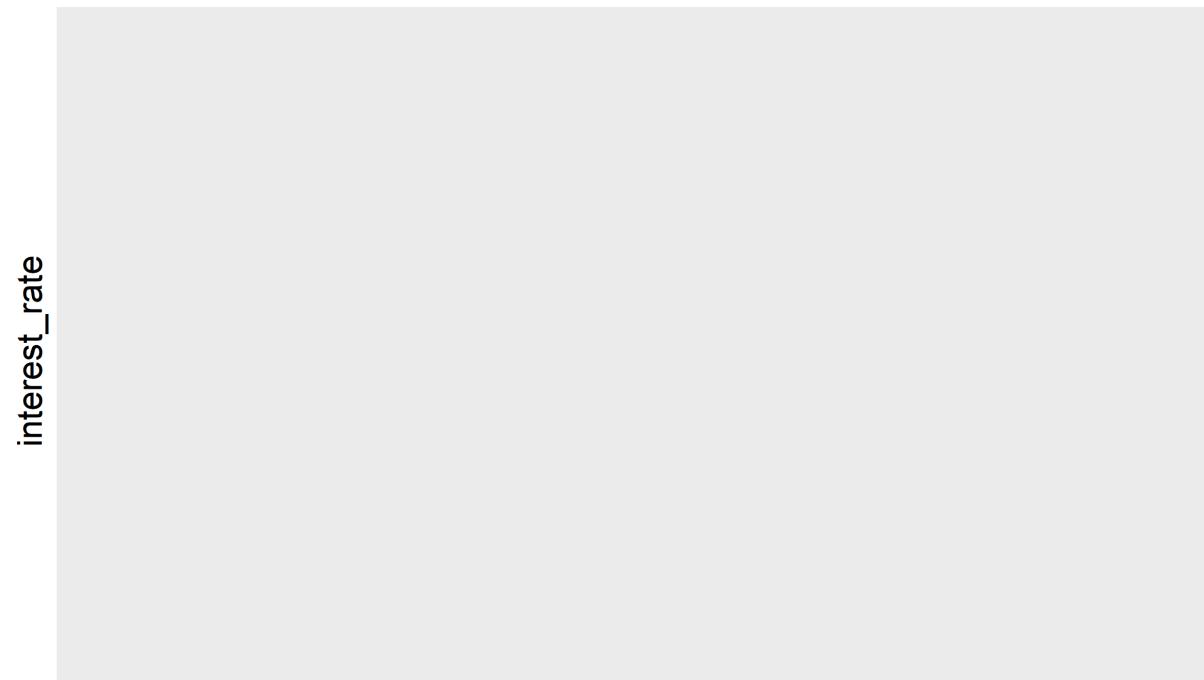
Hex plot

```
ggplot(loans, aes(x = debt_to_income, y = interest_rate)) +  
  geom_hex()
```



Hex plot

```
ggplot(loans %>% filter(debt_to_income < 100),  
       aes(x = debt_to_income, y = interest_rate)) +  
  geom_hex()
```



Sources

- + Mine Å‡etinkaya-Rundel's Data Science in a Box ([link](#))



Visualizing categorical data



Recap



Variables

- + **Numerical** variables can be classified as **continuous** or **discrete** based on whether or not the variable can take on an infinite number of values or only non-negative whole numbers, respectively.
- + If the variable is **categorical**, we can determine if it is **ordinal** based on whether or not the levels have a natural ordering.



Remember this Data?

```
library(tidyverse)  
starwars
```

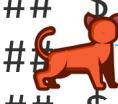
```
## # A tibble: 87 × 15  
##   name    height  mass hair_color skin_color eye_color birth_year  
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>          <dbl>  
## 1 Luke ...     172     77 blond      fair       blue           19  
## 2 C-3PO        167     75 <NA>       gold       yellow        112  
## 3 R2-D2         96     32 <NA>      white, bl... red            33  
## 4 Darth...      202    136 none       white      yellow        41.9  
## 5 Leia ...      150     49 brown      light      brown          19  
## 6 Owen ...      178    120 brown, gr... light      blue           52  
## 7 Beru ...      165     75 brown      light      blue           47  
## 8 R5-D4         97     32 <NA>      white, red red            NA  
## 9 Biggs...      183     84 black      light      brown          24  
## 10 Obi-W...      182     77 auburn, w... fair      blue-gray       57  
## # i 77 more rows
```



Perhaps now?

```
glimpse(starwars)
```

```
## Rows: 87
## Columns: 15
## $ name          <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth ...
## $ height        <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, ...
## $ mass          <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, ...
## $ hair_color    <chr> "blond", NA, NA, "none", "brown", "brown", g...
## $ skin_color    <chr> "fair", "gold", "white", "blue", "white", "li...
## $ eye_color     <chr> "blue", "yellow", "red", "yellow", "brown", ...
## $ birth_year    <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, ...
## $ sex           <chr> "male", "none", "none", "male", "female", "...
## $ gender         <chr> "masculine", "masculine", "masculine", "mas...
## $ homeworld     <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine"...
## $ species        <chr> "Human", "Droid", "Droid", "Human", "Human"...
## $ films          <list> <"The Empire Strikes Back", "Revenge of th...
## $ vehicles       <list> <"Snowspeeder", "Imperial Speeder Bike">, ...
## $ ...
```



Recode hair color

```
starwars <- starwars %>%
  mutate(hair_color2 =
    fct_other(hair_color,
              keep = c("black", "brown", "brown", "blond"))
  )
```

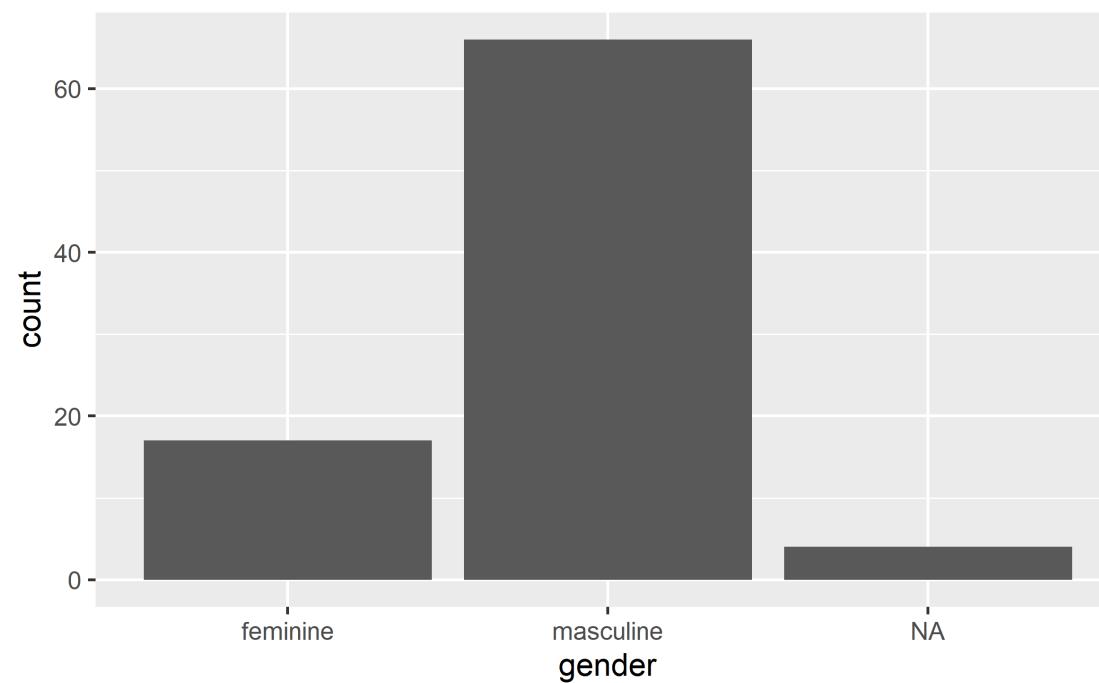


Bar plot



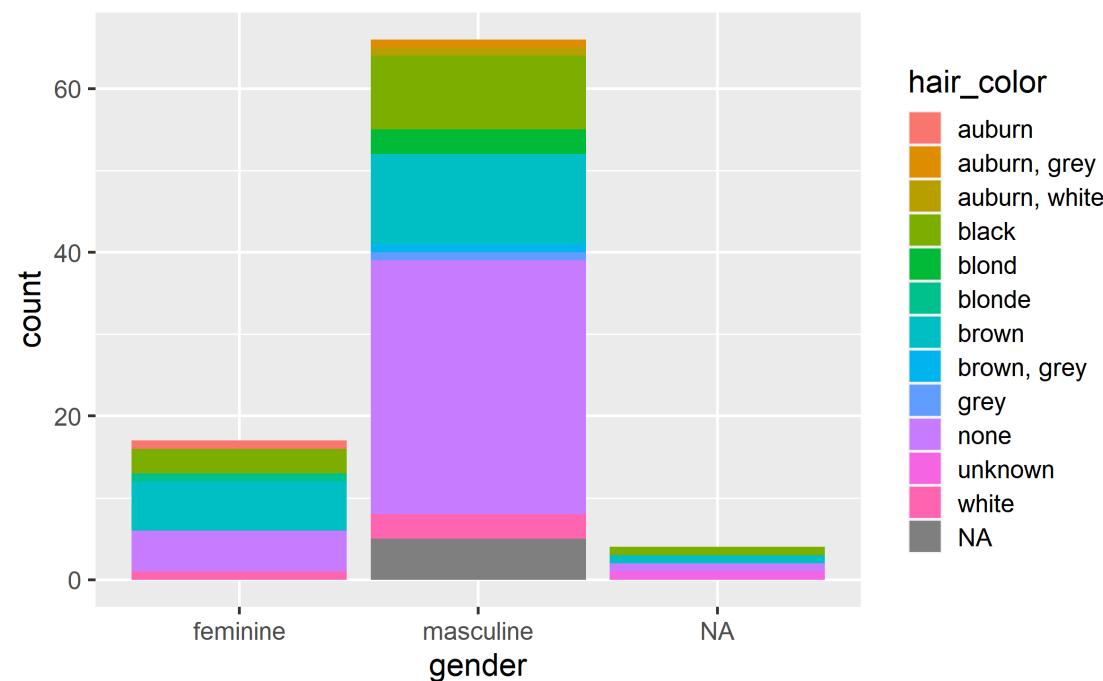
Bar plot

```
ggplot(data = starwars, mapping = aes(x = gender)) +  
  geom_bar()
```



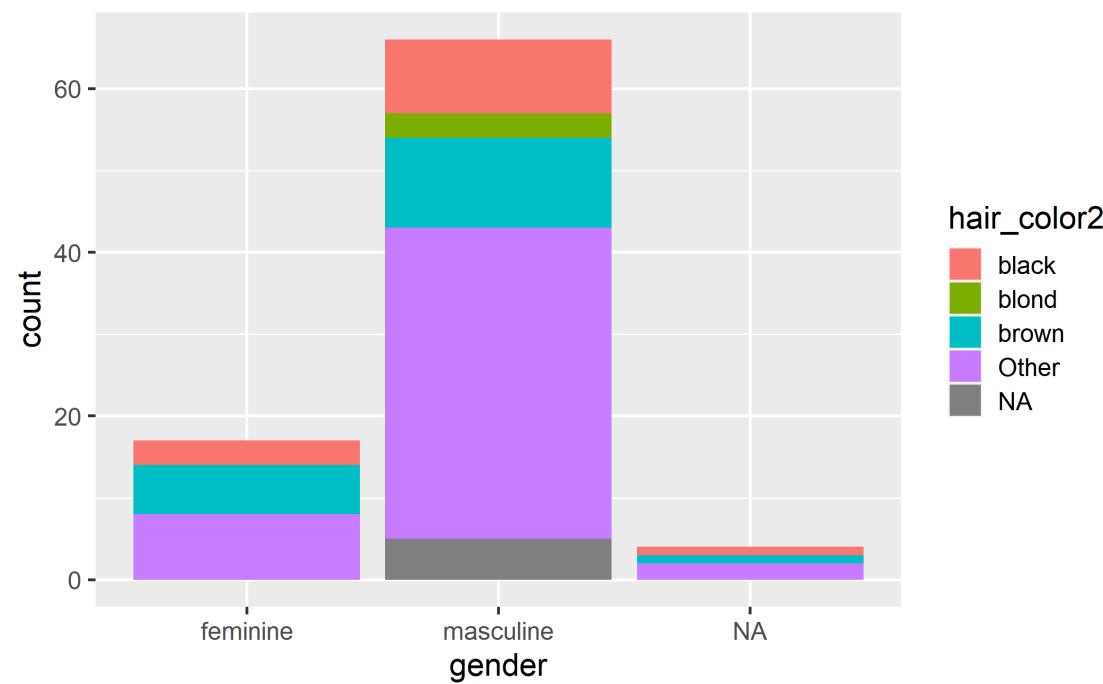
Segmented bar plot: counts

```
ggplot(data = starwars, mapping = aes(x = gender,  
                                         fill = hair_color))+  
  geom_bar()
```



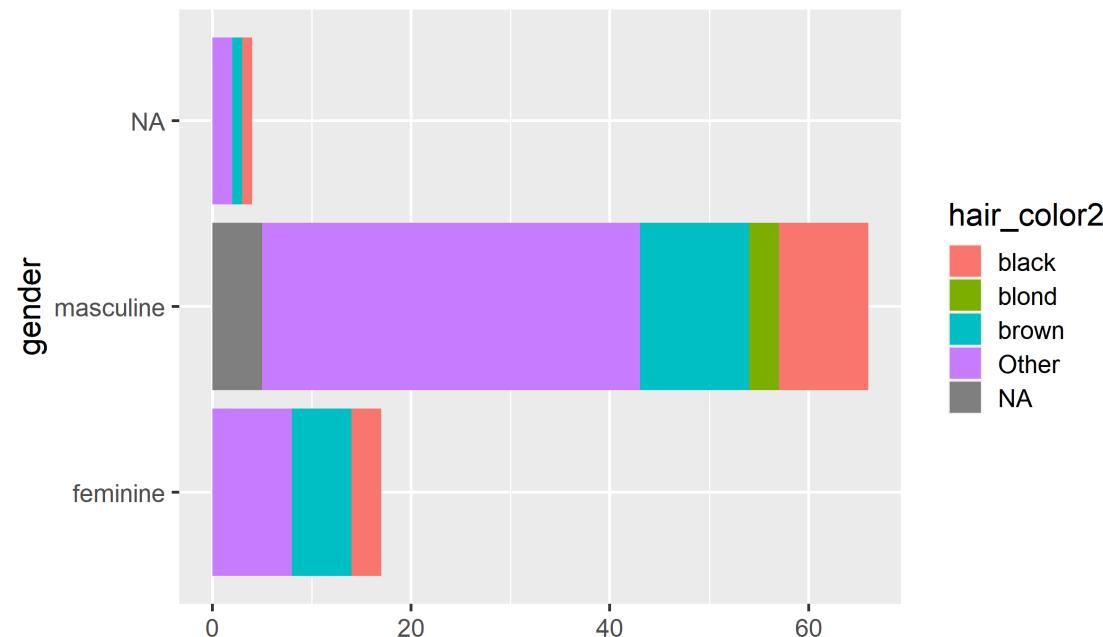
Segmented bar plots

```
ggplot(data = starwars, mapping = aes(x = gender,  
    fill = hair_color2))+  
  geom_bar()
```



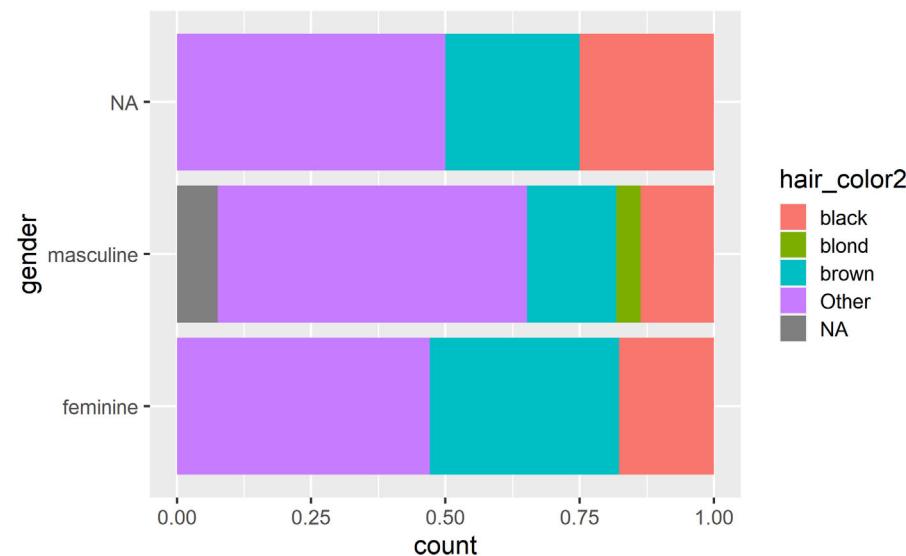
Segmented bar plots

```
ggplot(data = starwars, mapping = aes(x = gender,  
    fill = hair_color2))+  
  geom_bar() +  
  coord_flip()
```

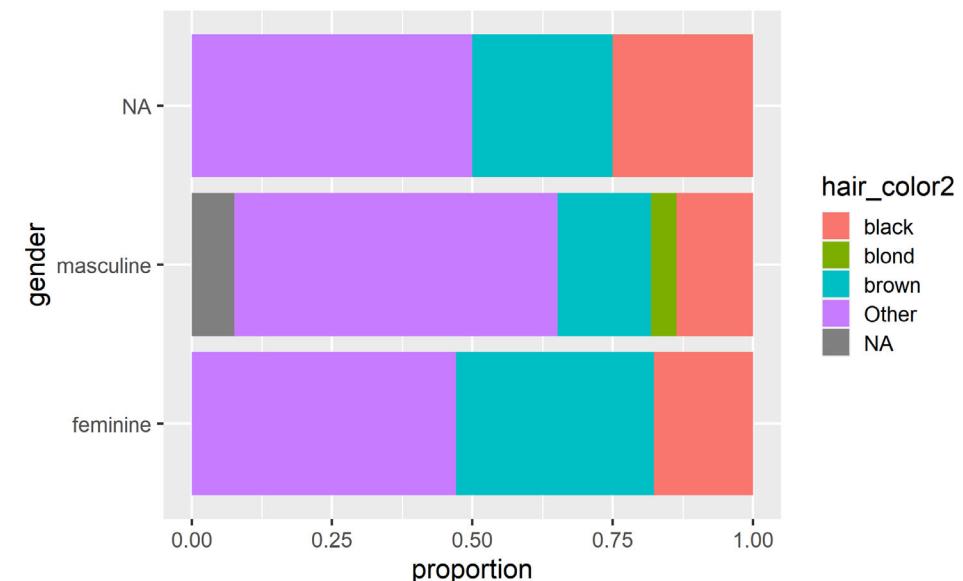
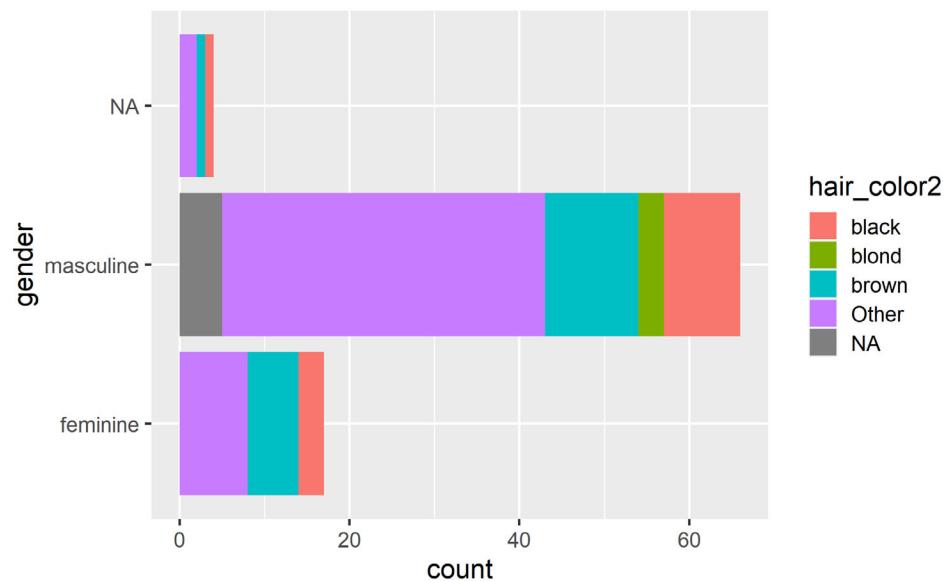


Segmented bar plots: proportions

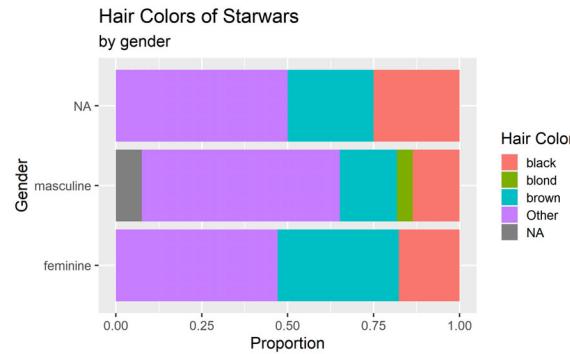
```
ggplot(data = starwars,  
       mapping = aes(x = gender, fill = hair_color2)) +  
  geom_bar(position = "fill") +  
  coord_flip()
```



Which bar plot is a more useful representation for visualizing the relationship between gender and hair color?



Customizing bar plots



```
ggplot(starwars, aes(y = gender,
                      fill = hair_color2)) +
  geom_bar(position = "fill") +
  labs(
    x = "Proportion",
    y = "Gender",
    fill = "Hair Color",
    title = "Hair Colors of Starwars",
    subtitle = "by gender"
  )
```

Your turn!

Time to actually play around with the Star Wars dataset!

- Go to class git repo ([\[github.com\]](#))
- Open the R Markdown document and c

class midtermsessionfinalsworkshop
Summer 2023, Fall 2023, Spring 2024, Fall 2024, Spring 2025
Spring 2026, Fall 2026, Spring 2027
Relationships



Student records

- + Survey missing

```
enrollment %>%  
  anti_join(survey, by = "id")
```

```
## # A tibble: 1 × 2  
##       id name  
##   <dbl> <chr>  
## 1     1 Dave Friday
```



Student records

- + Dropped

```
survey %>%  
  anti_join(enrollment, by = "id")
```

```
## # A tibble: 2 × 3  
##       id name   username  
##   <dbl> <chr> <chr>  
## 1     4 Peter peter_bakes  
## 2     5 Mark  thebakingbuddha
```



Case study: Grocery sales



Grocery sales

- + Have:
 - + Purchases: One row per customer per item, listing purchases they made
 - + Prices: One row per item in the store, listing their prices
- + Want: Total revenue

purchases

```
## # A tibble: 5 × 2
##   customer_id item
##       <dbl> <chr>
## 1             1 bread
## 2             1 milk
## 3             1 banana
## 4             2 milk
## 5             2 toilet paper
```

prices

```
## # A tibble: 5 × 2
##   item          price
##   <chr>        <dbl>
## 1 avocado      0.5
## 2 banana       0.15
## 3 bread         1
## 4 milk          0.8
## 5 toilet paper 3
```



Grocery sales

+ Total revenue

```
purchases %>%  
  left_join(prices)
```

```
## # A tibble: 5 × 3  
##   customer_id item      price  
##       <dbl> <chr>    <dbl>  
## 1           1 bread     1  
## 2           1 milk     0.8  
## 3           1 banana   0.15  
## 4           2 milk     0.8  
## 5           2 toilet paper 3
```

```
purchases %>%  
  left_join(prices) %>%  
  summarize(total_revenue = sum(pric
```



```
## # A tibble: 1 × 1  
##   total_revenue  
##       <dbl>  
## 1         5.75
```



Grocery sales

- + Revenue per customer

```
purchases %>%
  left_join(prices)
```

```
## # A tibble: 5 × 3
##   customer_id item                price
##       <dbl> <chr>              <dbl>
## 1           1 bread                 1
## 2           1 milk                 0.8
## 3           1 banana               0.15
## 4           2 milk                 0.8
## 5           2 toilet paper         3
```

```
purchases %>%
  left_join(prices) %>%
  group_by(customer_id) %>%
  summarize(total_revenue = sum(price))

## # A tibble: 2 × 2
##   customer_id total_revenue
##       <dbl>          <dbl>
## 1           1             1.95
## 2           2             3.8
```



Wrapping Up...



Data Science for Psychologists

Data types and recoding



Why should you care about data types?



Example: Cat lovers

A survey asked respondents their name and number of cats. The instructions said to enter the number of cats as a numerical value.

```
cat_lovers <- read_csv("data/cat-lovers.csv")  
  
## # A tibble: 60 × 3  
##   name      number_of_cats handedness  
##   <chr>        <chr>       <chr>  
## 1 Bernice Warren 0          left  
## 2 Woodrow Stone  0          left  
## 3 Willie Bass   1          left  
## 4 Tyrone Estrada 3          left  
## 5 Alex Daniels  3          left  
## 6 Jane Bates   2          left  
## 7 Latoya Simpson 1          left  
## 8 Darin Woods   1          left  
## 9 Agnes Cobb    0          left  
## 10 Tabitha Grant 0         left
```



Oh why won't you work?!

```
cat_lovers %>%  
  summarize(mean_cats = mean(number_of_cats))
```

```
## Warning: There was 1 warning in `summarize()`.  
## i In argument: `mean_cats = mean(number_of_cats)`.  
## Caused by warning in `mean.default()`:  
## ! argument is not numeric or logical: returning NA  
  
## # A tibble: 1 × 1  
##   mean_cats  
##       <dbl>  
## 1       NA
```



?mean

mean {base}

R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

- x An R object. Currently there are methods for numeric/logical vectors and [date](#), [date-time](#) and [time interval](#) objects. Complex vectors are allowed for trim = 0, only.
- trim the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- na.rm a logical value indicating whether NA values should be stripped before the computation proceeds.
- ... further arguments passed to or from other methods.



Oh why won't you still work??!!

```
cat_lovers %>%  
  summarize(mean_cats = mean(number_of_cats, na.rm = TRUE))
```

```
## Warning: There was 1 warning in `summarize()`.  
## i In argument: `mean_cats = mean(number_of_cats, na.rm = TRUE)`.  
## Caused by warning in `mean.default()`:  
## ! argument is not numeric or logical: returning NA  
  
## # A tibble: 1 × 1  
##   mean_cats  
##       <dbl>  
## 1        NA
```



Take a breath and look at your data

What is the type of the number_of_cats variable?

```
glimpse(cat_lovers)
```

```
## Rows: 60
## Columns: 3
## $ name           <chr> "Bernice Warren", "Woodrow Stone", "Will...
## $ number_of_cats <chr> "0", "0", "1", "3", "3", "2", "1", "1", ...
## $ handedness     <chr> "left", "left", "left", "left", "left", ...
```



Let's take another look

Show 10 entries

Search:

	name	number_of_cats	handedness
1	Bernice Warren	0	left
2	Woodrow Stone	0	left
3	Willie Bass	1	left
4	Tyrone Estrada	3	left
5	Alex Daniels	3	left
6	Jane Bates	2	left
7	Latoya Simpson	1	left
8	Darin Woods	1	left
9	Agnes Cobb	0	left
10	Tabitha Grant	0	left



You might need to babysit your respondents

```
cat_lovers %>%  
  mutate(number_of_cats = case_when(  
    name == "Ginger Clark" ~ 2,  
    name == "Doug Bass" ~ 3,  
    TRUE ~ as.numeric(number_of_cats))  
  ) %>%  
  summarize(mean_cats = mean(number_of_cats))
```

```
## Warning: There was 1 warning in `mutate()`.  
## i In argument: `number_of_cats = case_when(...)`.  
## Caused by warning:  
## ! NAs introduced by coercion
```

```
## # A tibble: 1 × 1  
##   mean_cats  
##       <dbl>  
## 1     0.833
```



Always you need to respect data types

```
cat_lovers %>%  
  mutate(  
    number_of_cats = case_when(  
      name == "Ginger Clark" ~ "2",  
      name == "Doug Bass" ~ "3",  
      TRUE ~ number_of_cats  
    ),  
    number_of_cats = as.numeric(number_of_cats)  
  ) %>%  
  summarize(mean_cats = mean(number_of_cats))
```

```
## # A tibble: 1 × 1  
##   mean_cats  
##     <dbl>  
## 1     0.833
```



Now that we know what we're doing...

```
cat_lovers <- cat_lovers %>%
  mutate(
    number_of_cats = case_when(
      name == "Ginger Clark" ~ "2",
      name == "Doug Bass"    ~ "3",
      TRUE                  ~ number_of_cats
    ),
    number_of_cats = as.numeric(number_of_cats)
  )
```



Moral of the story

- + If your data does not behave how you expect it to, type coercion upon reading in the data might be the reason.
- + Go in and investigate your data, apply the fix, *save your data*, live happily ever after.



Wrapping Up...



Data Science for Psychologists

now that we have a good motivation for learning about data types in R

let's learn about data types in R!



Data types



Data types in R

- + **logical**
- + **double**
- + **integer**
- + **character**
- + and some more, but we won't be focusing on those



Logical & character

logical - boolean values TRUE and FALSE

```
typeof(TRUE)
```

character - character strings

```
typeof("hello")
```

```
## [1] "logical"
```

```
## [1] "character"
```



Double & integer

double - floating point numerical values (default numerical type)

```
typeof(1.335)
```

```
## [1] "double"
```

```
typeof(7)
```

```
## [1] "double"
```

integer - integer numerical values (indicated with an L)

```
typeof(7L)
```

```
## [1] "integer"
```

```
typeof(1:3)
```

```
## [1] "integer"
```



Concatenation

Vectors can be constructed using the `c()` function.

```
c(1, 2, 3)
```

```
## [1] 1 2 3
```

```
c("Hello", "World!")
```

```
## [1] "Hello"  "World!"
```

```
c(c("hi", "hello"), c("bye", "jello"))
```

```
## [1] "hi"      "hello"   "bye"     "jello"
```



Converting between types

with intention...

```
x <- 1:3  
x
```

```
## [1] 1 2 3
```

```
typeof(x)
```

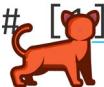
```
## [1] "integer"
```

```
y <- as.character(x)  
y
```

```
## [1] "1" "2" "3"
```

```
typeof(y)
```

```
## [1] "character"
```



Data Science for Psychologists

```
x <- c(TRUE, FALSE)  
x
```

```
## [1] TRUE FALSE
```

```
typeof(x)
```

```
## [1] "logical"
```

```
y <- as.numeric(x)  
y
```

```
## [1] 1 0
```

```
typeof(y)
```

```
## [1] "double"
```

Converting between types

without intention...

R will happily convert between various types without complaint when different types of data are concatenated in a vector, and that's not always a great thing!

```
c(1, "Hello")
```

```
c(1.2, 3L)
```

```
## [1] "1"      "Hello"
```

```
## [1] 1.2 3.0
```

```
c(FALSE, 3L)
```

```
c(2L, "two")
```

```
## [1] 0 3
```

```
## [1] "2"    "two"
```



Explicit vs. implicit coercion

Let's give formal names to what we've seen so far:

- + **Explicit coercion** is when you call a function like `as.logical()`, `as.numeric()`, `as.integer()`, `as.double()`, or `as.character()`.
- + **Implicit coercion** happens when you use a vector in a specific context that expects a certain type of vector.



Your turn!

- + `class git repo > AE 05 - Hotels + Data types > open type-coercion.Rmd and knit.`
- + What is the type of the given vectors? First, guess. Then, try it out in R. If your guess was correct, great! If not, discuss why they have that type.

Example: Suppose we want to know the type of `c(1, "a")`. First, I'd look at:

```
typeof(1)
```

```
## [1] "double"
```

```
typeof("a")
```

```
## [1] "character"
```

and make a guess based on these. Then finally I'd check:

```
typeof(c(1, "a"))
```

```
## [1] "character"
```



Wrapping Up...



Data Science for Psychologists

Special values



Special values

- + NA: Not available
- + NaN: Not a number
- + Inf: Positive infinity
- + -Inf: Negative infinity

```
pi / 0
```

```
1/0 - 1/0
```

```
## [1] Inf
```

```
## [1] NaN
```

```
0 / 0
```

```
1/0 + 1/0
```

```
## [1] NaN
```

```
## [1] Inf
```



NAs are special s

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

```
mean(x)
```

```
## [1] NA
```

```
mean(x, na.rm = TRUE)
```

```
## [1] 2.5
```

```
summary(x)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
##	1.00	1.75	2.50	2.50	3.25	4.00	1



Data Science for Psychologists

NAs are logical

R uses NA to represent missing values in its data structures.

```
typeof(NA)
```

```
## [1] "logical"
```



Mental model for NAs

- + Unlike NaN, NAs are genuinely unknown values
- + But that doesn't mean they can't function in a logical way
- + Let's think about why NAs are logical...

Why do the following give different answers?

```
# TRUE or NA  
TRUE | NA
```

```
# FALSE or NA  
FALSE | NA
```

```
## [1] TRUE
```

```
## [1] NA
```

→ See next slide for answers...



+ NA is unknown, so it could be TRUE or FALSE

+ TRUE or TRUE is TRUE and TRUE or FALSE is also TRUE, and since both are TRUE

TRUE | TRUE

```
## [1] TRUE
```

FALSE | TRUE

```
## [1] TRUE
```

+ FALSE or TRUE is TRUE and FALSE or FALSE is also FALSE, so you can't tell which should be the right answer

FALSE | TRUE

```
## [1] TRUE
```

FALSE | FALSE

```
## [1] FALSE
```

+ Doesn't make sense for mathematical operations but make sense in the context of missing data



Wrapping Up...



Data Science for Psychologists

Data classes



Data classes

We talked about *types* so far, next we'll introduce the concept of *classes*

- + Vectors are like Lego building blocks
- + We stick them together to build more complicated constructs, e.g. *representations of data*
- + The **class** attribute relates to the S3 class of an object which determines its behaviour
 - + You don't need to worry about what S3 classes really mean, but you can read more about it [here](#) if you're curious
- + Examples: factors, dates, and data frames



Factors

R uses factors to handle categorical variables, variables that have a fixed and known set of possible values

```
x <- factor(c("BS", "MS", "PhD", "MS"))
x
```

```
## [1] BS  MS  PhD MS
## Levels: BS MS PhD
```

```
typeof(x)
```

```
class(x)
```

```
## [1] "integer"
```

```
## [1] "factor"
```



More on factors

We can think of factors like character (level labels) and an integer (level numbers) glued together

```
glimpse(x)
```

```
## Factor w/ 3 levels "BS", "MS", "PhD": 1 2 3 2
```

```
as.integer(x)
```

```
## [1] 1 2 3 2
```



Dates

```
y <- as.Date("2020-01-01")  
y
```

```
## [1] "2020-01-01"
```

```
typeof(y)
```

```
## [1] "double"
```

```
class(y)
```

```
## [1] "Date"
```



More on dates

We can think of dates like an integer (the number of days since the origin, 1 Jan 1970) and an integer (the origin) glued together

```
as.integer(y)
```

```
## [1] 18262
```

```
as.integer(y) / 365 # roughly 50 yrs
```

```
## [1] 50.03288
```



Data frames

We can think of data frames like vectors of equal length glued together

```
df <- data.frame(x = 1:2, y = 3:4)  
df
```

```
##   x y  
## 1 1 3  
## 2 2 4
```

```
typeof(df)
```

```
class(df)
```

```
## [1] "list"
```

```
## [1] "data.frame"
```



Lists

Lists are a generic vector container vectors of any type can go in them

```
l <- list(  
  x = 1:4,  
  y = c("hi", "hello", "jello"),  
  z = c(TRUE, FALSE)  
)  
l
```

```
## $x  
## [1] 1 2 3 4  
##  
## $y  
## [1] "hi"     "hello"   "jello"  
##  
## $z  
## [1] TRUE FALSE
```



Lists and data frames

- + A data frame is a special list containing vectors of equal length
- + When we use the `pull()` function, we extract a vector from the data frame

```
df
```

```
##      x  y
## 1  1  3
## 2  2  4
```

```
df %>%
  pull(y)
```

```
## [1] 3 4
```



Wrapping Up...



Data Science for Psychologists

Working with factors



Read data in as character strings

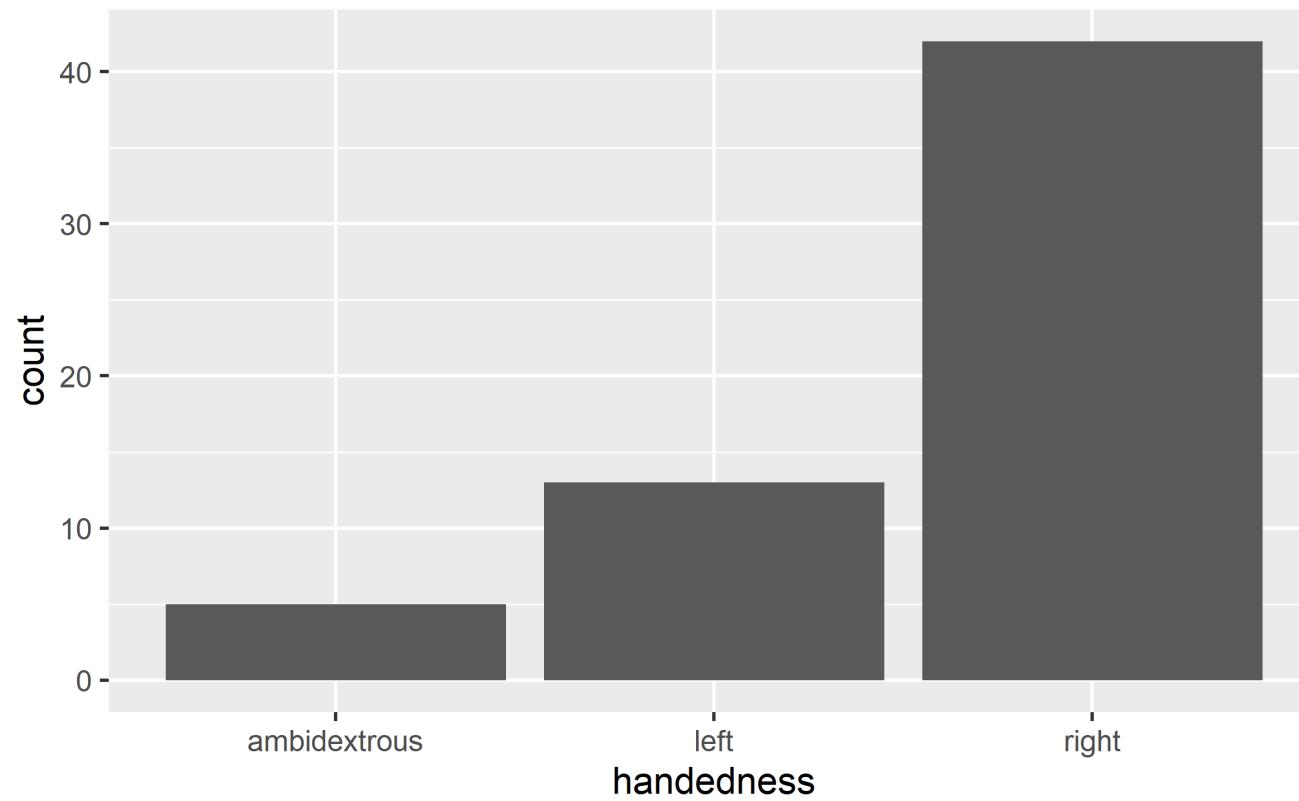
```
glimpse(cat_lovers)
```

```
## Rows: 60
## Columns: 3
## $ name      <chr> "Bernice Warren", "Woodrow Stone", "Will...
## $ number_of_cats <dbl> 0, 0, 1, 3, 3, 2, 1, 1, 0, 0, 0, 1, 3...
## $ handedness <chr> "left", "left", "left", "left", "left", ...
```



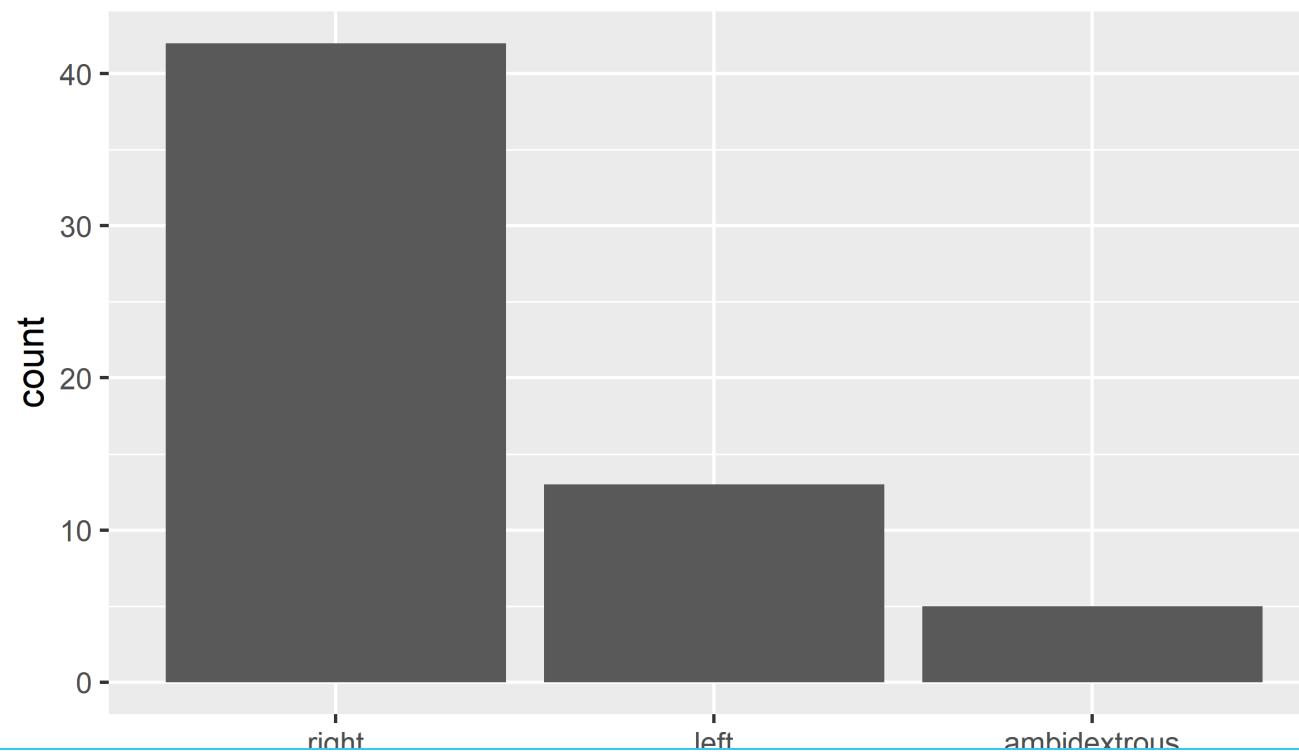
But coerce when plotting

```
ggplot(cat_lovers, mapping = aes(x = handedness)) +  
  geom_bar()
```



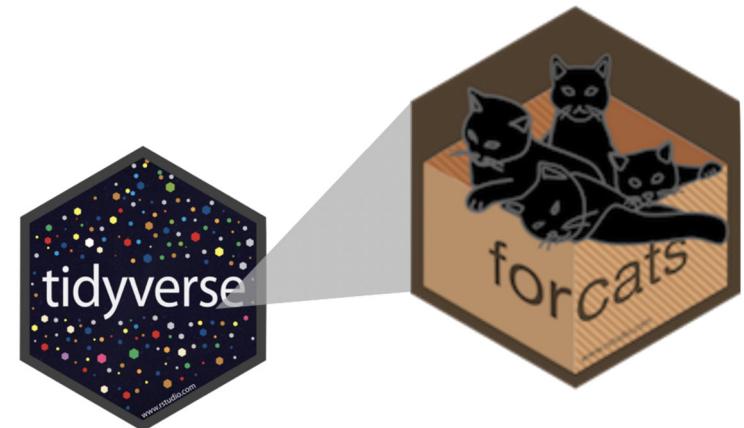
Use forcats to manipulate factors

```
cat_lovers %>%  
  mutate(handedness = fct_infreq(handedness)) %>%  
  ggplot(mapping = aes(x = handedness)) +  
  geom_bar()
```



Come for the functionality

... stay for the logo



- + Factors are useful when you have true categorical data and you want to override the ordering of character vectors to improve display
- + They are also useful in modeling scenarios
- + The **forcats** package provides a suite of useful tools that solve common problems with factors

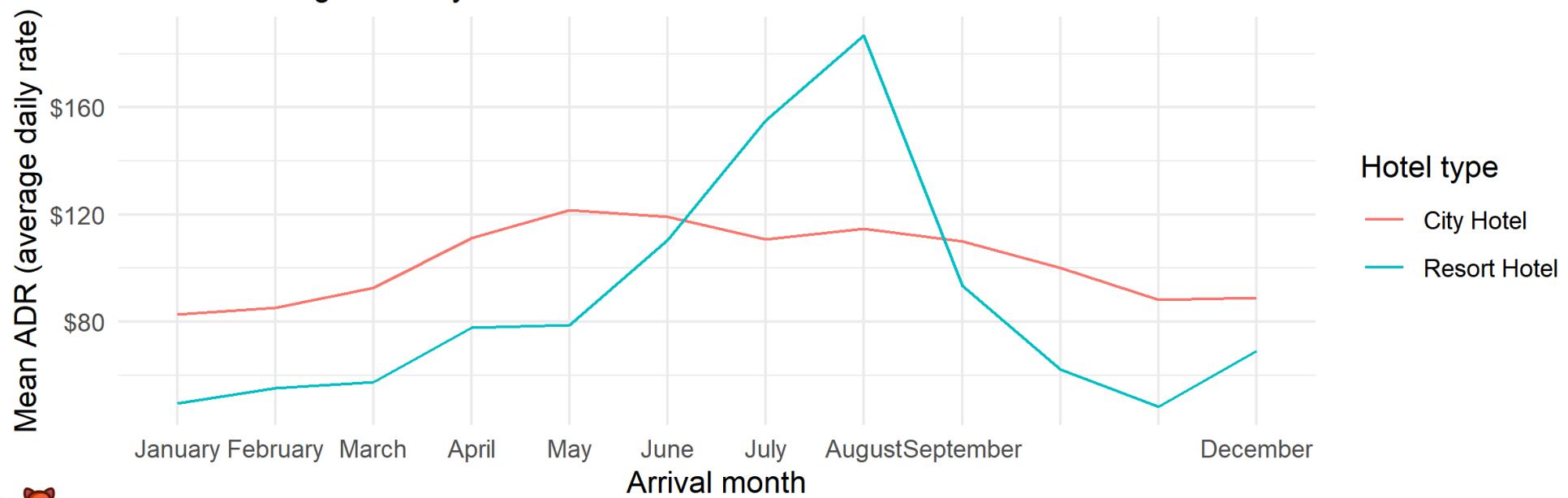


Your turn!

- + `class git repo > AE 05 - Hotels + Data types > hotels-forcats.Rmd > knit`
- + Recreate the following. The x-axis first, then, as a stretch goal, the y-axis.

Comparison of resort and city hotel prices across months

Resort hotel prices soar in the summer, yet city hotel prices remain constant throughout the year



Wrapping Up...



Data Science for Psychologists

Importing data



Reading rectangular data into R





readr

- + `read_csv()` - comma delimited files
- + `read_csv2()` - semicolon separated files (common in countries where , is used as the decimal place)
- + `read_tsv()` - tab delimited files
- + `read_delim()` - reads in files with any delimiter
- + `read_fwf()` - fixed width files
- + `read_table()` - common variation of fixed width files where columns are separated by white space
- + ...



Reading data

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



```
## — Data Summary ——————
##                               Values
## Name                           nobel
## Number of rows                 935
## Number of columns                26
##
## Column type frequency:
##   character                      21
##   Date                            2
##   numeric                          3
##
## ----- Group variables           None
##
## — Variable type: character ——————
##   skim_variable n_missing complete_rate min max empty n_unique whitespace
## 1 firstname                  0       1     2 59    0    720      0
## 2 surname                     29      0.969  2 26    0    851      0
## 3 category                     0       1     5 10    0     6      0
## 4 affiliation                  250      0.733  4 110   0    303      0
## 5 city                         255      0.727  4 27    0    185      0
## 6 country                      254      0.728  3 14    0     27      0
## 7 gender                        0       1     3 6     0     3      0
## 8 born_city                     28      0.970  3 29    0    613      0
## 9 born_country                  28      0.970  3 28    0     80      0
## 10 born_country_code            28      0.970  2 2     0     77      0
## 11 died_city                    327      0.650  4 29    0    303      0
## 12 died_country                 321      0.657  3 16    0     48      0
## 13 died_country_code            321      0.657  2 2     0     46      0
## 14 overall_motivation          918      0.0182 55 114   0     7      0
## 15 motivation                   0       1     24 337   0    656      0
## 16 born_country_original        28      0.970  3 52    0    122      0
## 17 born_city_original           28      0.970  3 36    0    616      0
## 18 died_country_original        321      0.657  3 35    0     52      0
## 19 died_city_original           327      0.650  4 29    0    303      0
## 20 city_original                 255      0.727  4 27    0    185      0
## 21 country_original              254      0.728  3 35    0     29      0
##
## — Variable type: Date ——————
##   skim_variable n_missing complete_rate min           max           median      n_unique
## 1 born_date                  33       0.965 1817-11-30 1997-07-12 1916-06-28      885
## 2 died_date                  308      0.671 1903-11-01 2019-08-07 1983-03-09      616
##
## — Variable type: numeric ——————
##   skim_variable n_missing complete_rate      mean      sd      p0      p25      p50      p75      p100 hist
## 1 id                          0       1 475.    278.    1 234.    470 716. 969 ██████████
## 2 year                        0       1 1970.   33.3 1901 1947 1976 1999 2018 ██████████
## 3 share                       0       1 1.99    0.936  1   1   2   3   4 ████████
```



Writing data

+ Write a file

```
df <- tribble(  
  ~x, ~y,  
  1, "a",  
  2, "b",  
  3, "c"  
)  
  
write_csv(df, file = "data/df.csv")
```



- + Check that it got written out

```
fs::dir_ls("data")
```

```
## data/cat-lovers.csv
## data/df-na.csv
## data/df.csv
## data/edi-airbnb.csv
## data/favorite-food.xlsx
## data/favourite-food.xlsx
## data/highest-points-by-state.csv
## data/hotels.csv
## data/instructional-staff.csv
## data/lattice.RData
## data/minard-cities.csv
## data/minard-temps.csv
## data/minard.csv
## data/nobel.csv
## data/relig-income.xlsx
```



Data Science for Psychologists

Your turn!

- + `class git repo > AE 06 - Nobels and sales + Data import > open nobels-csv.Rmd and knit.`
- + Read in the `nobels.csv` file from the `data-raw/` folder.
- + Split into two (STEM and non-STEM):
 - + Create a new data frame, `nobel_stem`, that filters for the STEM fields (Physics, Medicine, Chemistry, and Economics).
 - + Create another data frame, `nobel_nonstem`, that filters for the remaining fields.
- + Write out the two data frames to `nobel-stem.csv` and `nobel-nonstem.csv`, respectively, to `data/`.

Hint: Use the `%in%` operator when filtering.



Pausing...



Variable names



```
edi_airbnb <- read_csv("data/edi-airbnb.csv")
names(edi_airbnb)
```

```
## [1] "ID"                  "Price"
## [3] "neighbourhood"       "accommodates"
## [5] "Number of bathrooms" "Number of Bedrooms"
## [7] "n beds"               "Review Scores Rating"
## [9] "Number of reviews"    "listing_url"
```

... but R doesn't allow spaces in variable names

```
ggplot(edi_airbnb, aes(x = Number of bathrooms, y = Price)) +
  geom_point()
```

```
## Error: <text>:1:35: unexpected symbol
## 1: ggplot(edi_airbnb, aes(x = Number of
##                                ^
```



Option 1 - Define column names

```
edi_airbnb_col_names <- read_csv("data/edi-airbnb.csv",
  col_names = c("id", "price", "neighbourhood", "accommodates",
    "bathroom", "bedroom", "bed",
    "review_scores_rating", "n_reviews", "url"))

names(edi_airbnb_col_names)
```

```
## [1] "id"                      "price"
## [3] "neighbourhood"           "accommodates"
## [5] "bathroom"                 "bedroom"
## [7] "bed"                      "review_scores_rating"
## [9] "n_reviews"                "url"
```



Option 2 - Format text to snake_case

```
edi_airbnb_cleaned_names <- edi_airbnb %>%  
janitor::clean_names()  
  
names(edi_airbnb_cleaned_names)
```

```
## [1] "id"                  "price"  
## [3] "neighbourhood"      "accommodates"  
## [5] "number_of_bathrooms" "number_of_bedrooms"  
## [7] "n_beds"              "review_scores_rating"  
## [9] "number_of_reviews"    "listing_url"
```



Wrapping Up...



Variable types



Which class is x? Why?

x	y	z
1	a	hi
NA	b	hello
3	Not applicable	9999
4	d	ola
5	e	hola
.	f	whatup
7	g	wassup
8	h	sup
9	i	

```
read_csv("data/df-na.csv")
```

```
## # A tibble: 9 × 3
##   x     y           z
##   <chr> <chr>      <chr>
## 1 1     a           hi
## 2 <NA>  b           hello
## 3 3     Not applicable 9999
## 4 4     d           ola
## 5 5     e           hola
## 6 .     f           whatup
## 7 7     g           wassup
## 8 8     h           sup
## 9 9     i           <NA>
```



Option 1. Explicit NAs

```
read_csv("data/df-na.csv",
         na = c("", "NA", ".", "9999", "Not applicable"))
```

x	y	z
1	a	hi
NA	b	hello
3	Not applicable	9999
4	d	ola
5	e	hola
.	f	whatup
7	g	wassup
8	h	sup

```
## # A tibble: 9 × 3
##       x     y     z
##   <dbl> <chr> <chr>
## 1     1     a    hi
## 2     NA    b   hello
## 3     3    <NA> <NA>
## 4     4     d    ola
## 5     5     e   hola
## 6     NA    f  whatup
## 7     7     g  wassup
## 8     8     h     sup
## 9     9     i  <NA>
```



Option 2. Specify column types

```
read_csv("data/df-na.csv",
  col_types = list(col_double(), col_character(), col_character()))
```

```
## Warning: One or more parsing issues, call `problems()` on your data frame
## for details, e.g.:
##   dat <- vroom(...)
##   problems(dat)

## # A tibble: 9 × 3
##       x     y     z
##   <dbl> <chr>  <chr>
## 1     1     a     hi
## 2     NA    b     hello
## 3     3 Not applicable 9999
## 4     4     d     ola
## 5     5     e     hola
## 6     NA    f     whatup
## 7     7     g     wassup
## 8     8     h     sup
## 9     9     i     <NA>
```



Column types

type function	data type
col_character()	character
col_date()	date
col_datetime()	POSIXct (date-time)
col_double()	double (numeric)
col_factor()	factor
col_guess()	let readr guess (default)
col_integer()	integer
col_logical()	logical
col_number()	numbers mixed with non-number characters
col_numeric()	double or integer
col_skip()	do not read
col_time()	time



Pause the video...

Your turn!

- + `class git repo > AE 06 - Nobels and sales + Data import > open food-excel.Rmd and knit.` Work on **Exercise 1**.
- + Read in the Excel file called `favourite-food.xlsx` from the `data-raw/` folder.
- + Clean up NAs and make sure you're happy with variable types.
- + Convert SES (socio economic status) to a factor variables with levels in the following order: Low, Middle, High.
- + Write out the resulting data frame to `favourite-food.csv` in the `data/` folder.
- + Finally, read `favourite-food.csv` back in from the `data/` folder and observe the variable types. Are they as you left them?



Ready to move forward?



read_rds() and write_rds()

- + CSVs can be unreliable for saving interim results if there is specific variable type information you want to hold on to.
- + An alternative is RDS files, you can read and write them with `read_rds()` and `write_rds()`, respectively.

```
read_rds(path)  
write_rds(x, path)
```



Your turn!

- + `class git repo > AE 06 - Nobels and sales + Data import > open food-excel.Rmd and knit.` Work on **Exercise 2**.
- + Repeat the first three steps from Exercise 1.
- + Write out the resulting data frame to `favourite-food.rds` in the `data/` folder.
- + Read `favourite-food.rds` back in from the `data/` folder and observe the variable types. Are they as you left them?



Ready to move forward?



Your turn!

- + `class git repo > AE 06 - Nobels and sales + Data import > open sales-excel.Rmd and knit.`
- + Load the `sales.xlsx` file from the `data-raw/` folder, using appropriate arguments for the `read_excel()` function such that it looks like the following.

```
## # A tibble: 9 × 2
##   id      n
##   <chr>   <chr>
## 1 Brand 1 n
## 2 1234  8
## 3 8721  2
## 4 1822  3
## 5 Brand 2 n
## 6 3333  1
## 7 2156  3
## 8 3987  6
## 9 3216  5
```



Your turn!

- + `class git repo > AE 06 - Nobels and sales + Data import > open sales-excel.Rmd and knit.`
- + Manipulate the sales data such that it looks like the following.

```
## # A tibble: 7 × 3
##   brand     id     n
##   <chr>   <dbl> <dbl>
## 1 Brand 1 1234 8
## 2 Brand 1 8721 2
## 3 Brand 1 1822 3
## 4 Brand 2 3333 1
## 5 Brand 2 2156 3
## 6 Brand 2 3987 6
## 7 Brand 2 3216 5
```



Wrapping Up...



Data Science for Psychologists

Deeper Dive into ggplot



Deep Diving into Grammar of Graphics



What does "grammar of graphics" mean?

- + The analogy with English grammar,
 - + or any language's grammar,
 - + is that it allows you to put together component parts
- + Better than "grammar of graphics" might be the
 - + "orthogonal components of graphics,"
 - + but that doesn't have the same alliterative appeal.
- + The power of the grammar of graphics is that it is modular:
 - + different aspects of the plot can be specified independently of each other.



An example

- + As an example, the coordinate system is specified separately
 - + from the geometric object used to represent the points.
- + We have three representations of the same data, where
 - + the only difference between them
 - + is the coordinate system used to represent them.

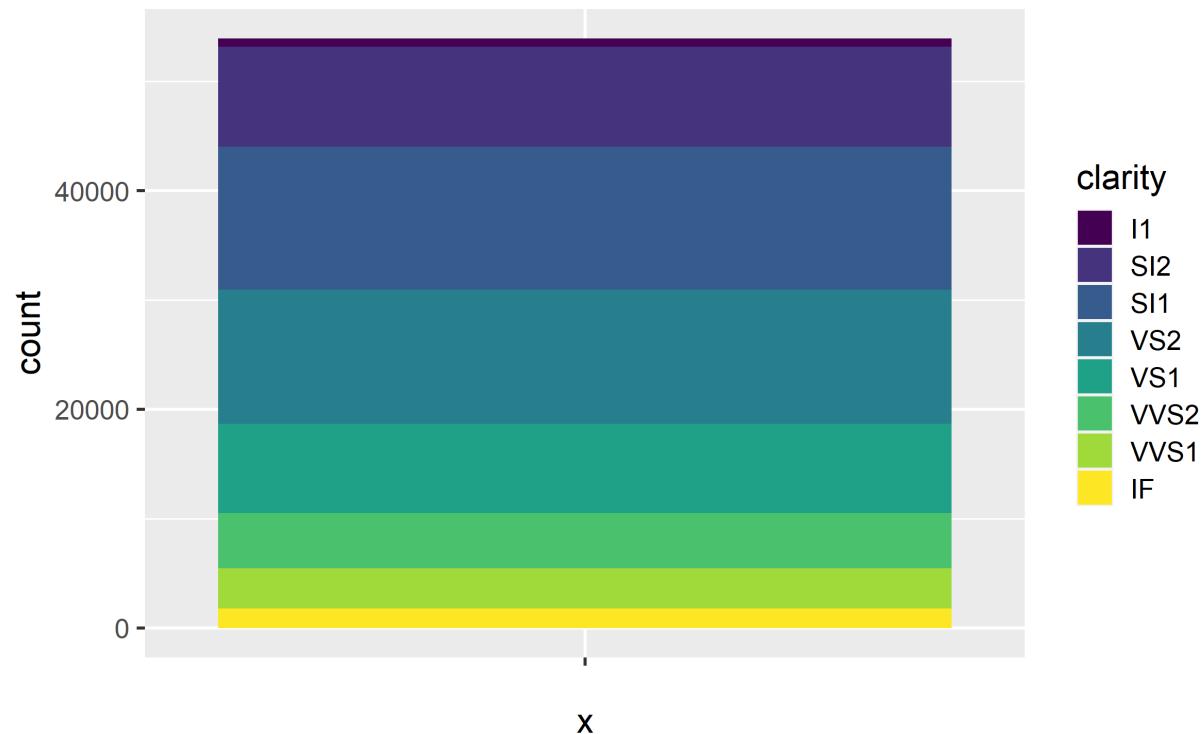
```
bar_plot = ggplot(diamonds) +  
  aes(x = "", fill = clarity) +  
  geom_bar(width = 1, position = "stack")  
bar_plot
```

```
bar_plot + coord_polar()
```

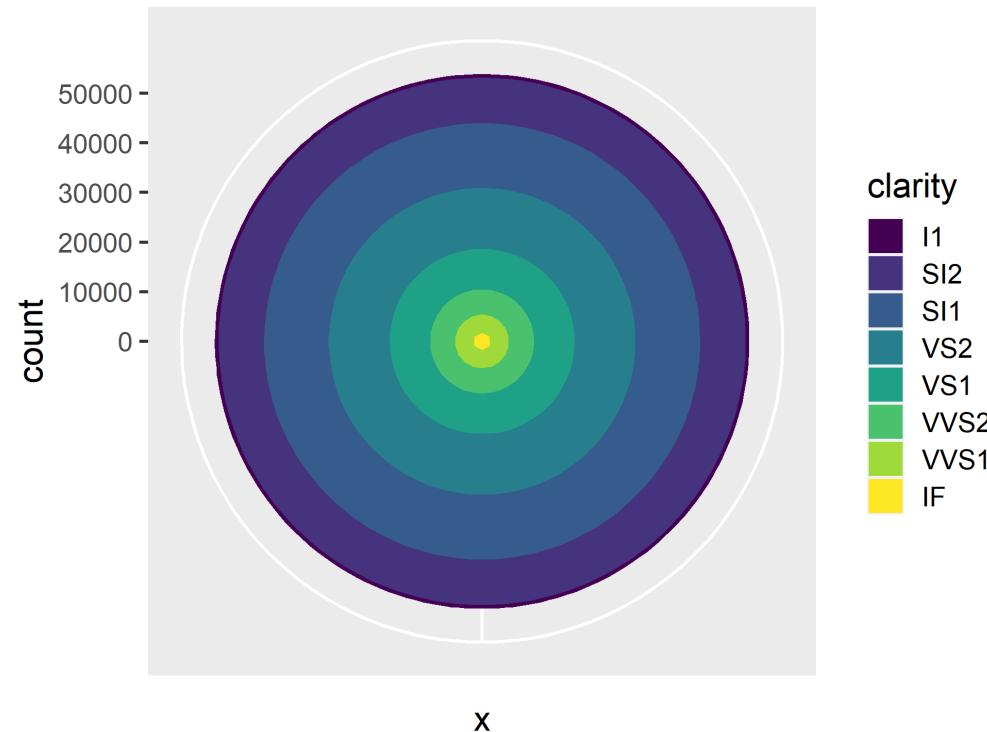
```
bar_plot + coord_polar(theta = "y")
```



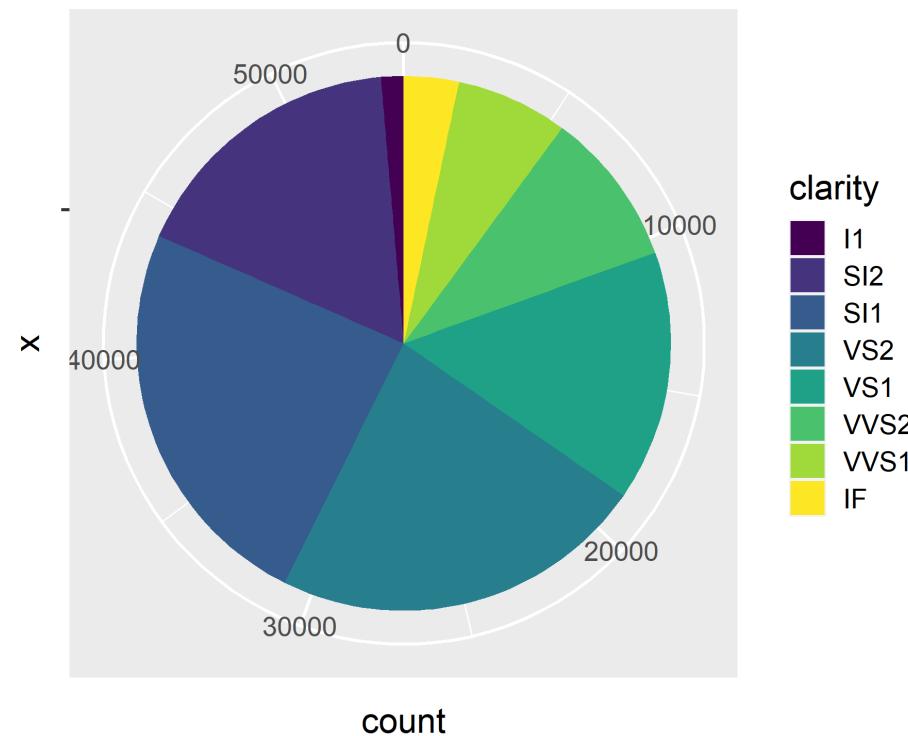
bar_plot



```
bar_plot + coord_polar()
```



```
bar_plot + coord_polar(theta = "y")
```



Again, the same dataset

Again, the same dataset, three different coordinate systems, very different representations:

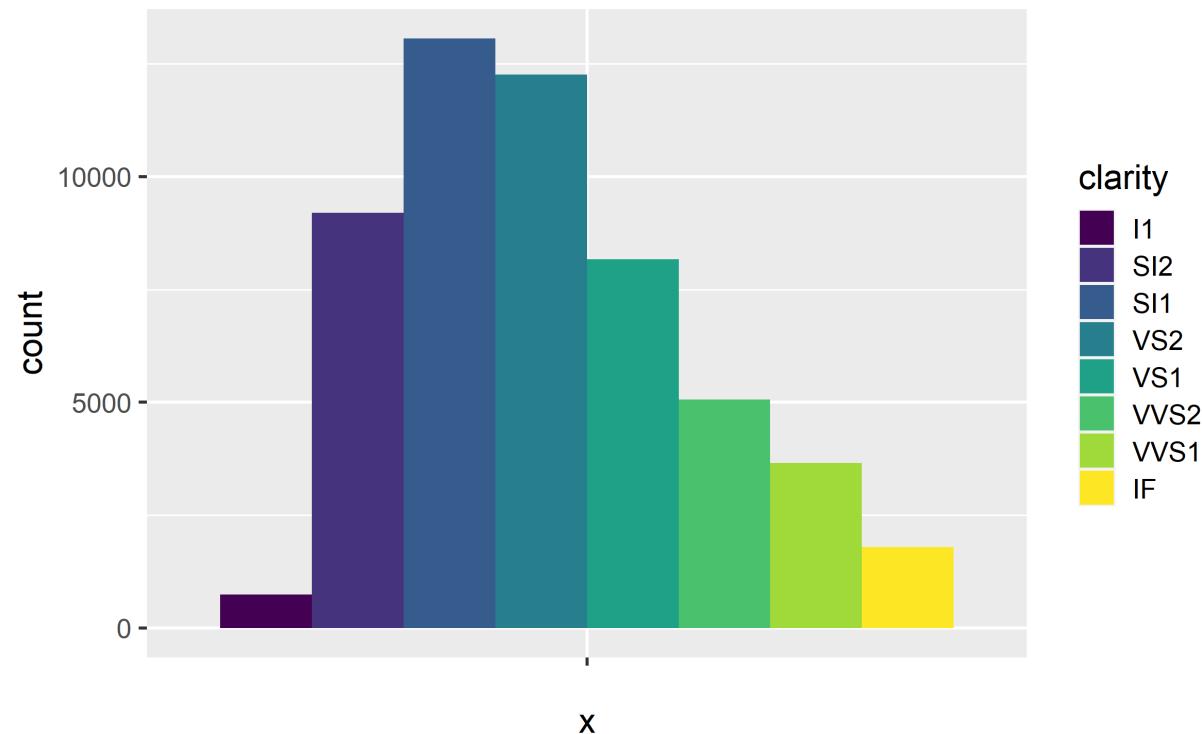
```
dodged_bar_plot = ggplot(diamonds) +  
    geom_bar(aes(x = "", fill = clarity), width = 1, position = "dodge")  
dodged_bar_plot
```

```
dodged_bar_plot + coord_polar()
```

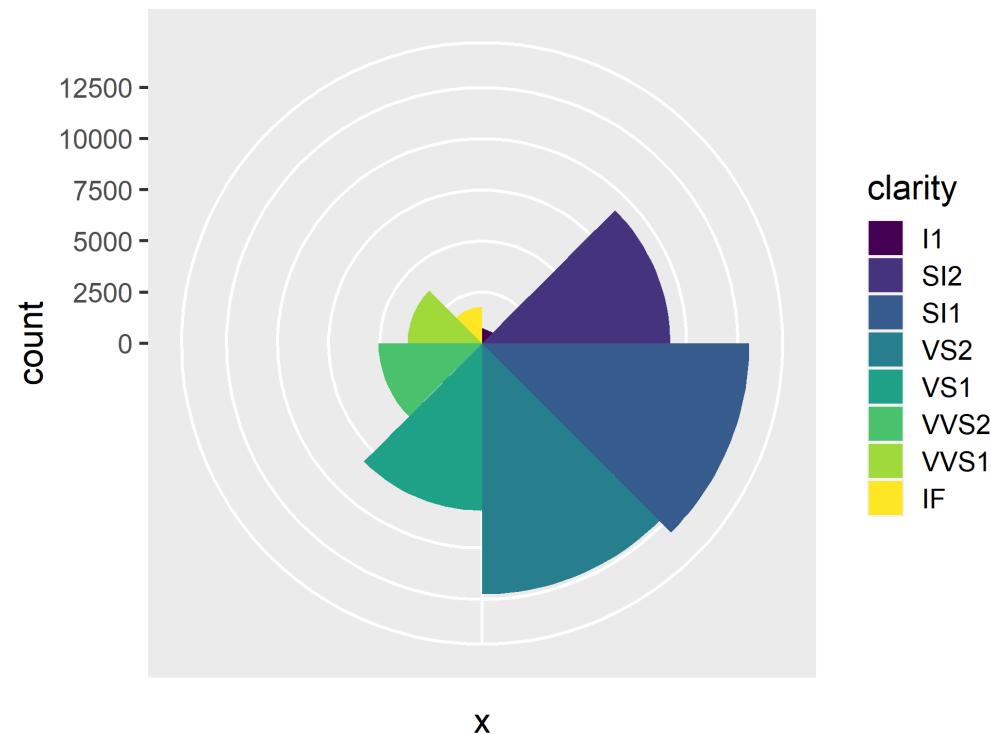
```
dodged_bar_plot + coord_polar(theta = "y")
```



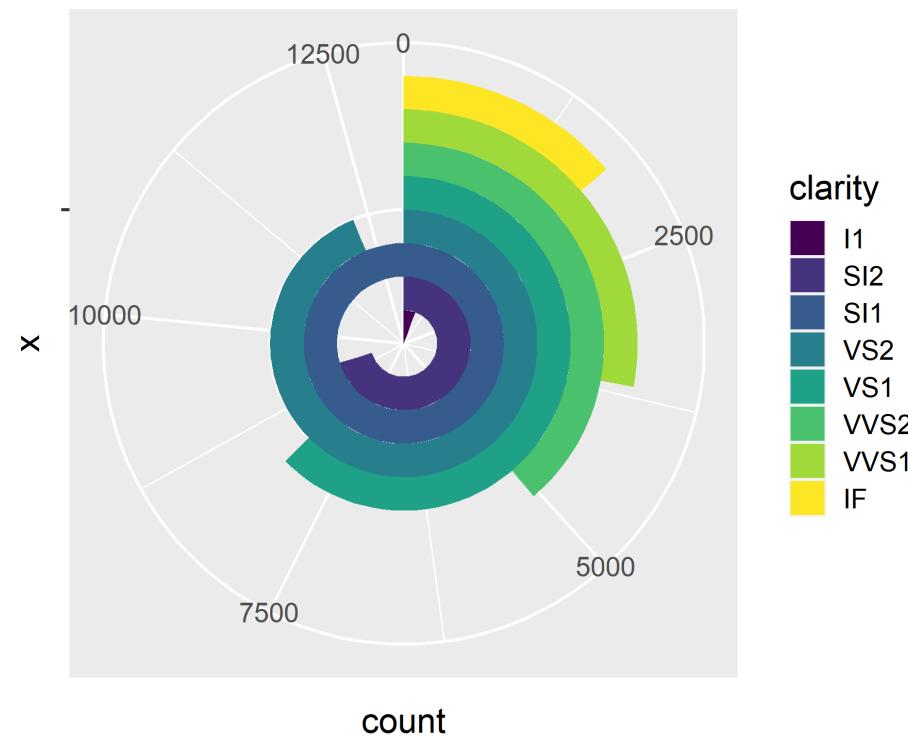
dodged_bar_plot



```
dodged_bar_plot + coord_polar()
```



```
dodged_bar_plot + coord_polar(theta = "y")
```



Wrapping Up...



Data Science for Psychologists

What are the components of a plot?



Components of a plot

- + A default dataset and set of mappings from variables to aesthetics.
- + One or more *layers*, each of which contains
 - + one geometric (`geom_*`) object,
 - + one statistical transformation (`stat_*`),
 - + one position adjustment (`position_*`),
- + and one dataset and set of aesthetic mappings.
- + One scale for each aesthetic mapping.
- + A coordinate system (`coord_*`).
- + A facet specification (`facet_*`).

Tip:

Layers are the most important and involved part of the plot.



What is a layer?

- + data and aesthetic mapping
- + statistical transformation (stat)
- + geometric object (geom)
- + position adjustment

```
p = ggplot(diamonds, aes(x = color, y = clarity)) + geom_point()  
p$layers
```

```
## [[1]]  
## geom_point: na.rm = FALSE  
## stat_identity: na.rm = FALSE  
## position_identity
```



Data and aesthetic mapping

- + Data: self evident. For ggplot the data needs to be formatted as a tibble or a data.frame.
- + Aesthetic mapping: Less evident
 - + Describes how variables in the dataset are mapping
 - + to "aesthetic" attributes of the plot.
 - + "Aesthetic" here means perceivable:
 - + something you can observe on the plot.



Aesthetic/perceivable attributes

+ Examples include

- + position along the x -axis,
- + color,
- + shape,
- + position along the y -axis,
- + opacity,
- + linetype



Data and aesthetic mapping go together

- + They are not independent of each other:
- + the aesthetic mapping takes variables in your data and
 - + maps them to aesthetics/perceivable parts of the plot and
 - + is therefore specific to a dataset.



Wrapping Up...



Stats, Geoms, and Positions



Statistical transformation

A statistical transformation is some transformation of the data.

Table 7. Some statistical transformations provided by `ggplot2`. The user is able to supplement this list in a straightforward manner.

Name	Description
bin	Divide continuous range into bins, and count number of points in each
boxplot	Compute statistics necessary for boxplot
contour	Calculate contour lines
density	Compute 1d density estimate
identity	Identity transformation, $f(x) = x$
jitter	Jitter values by adding small random value
qq	Calculate values for quantile-quantile plot
quantile	Quantile regression
smooth	Smoothed conditional mean of y given x
summary	Aggregate values of y for given x
unique	Remove duplicated observations

These stat transformations and position adjustments can overlap. Often there is more than one way to create the same plot.



Geometric objects

Geometric objects (`geom_*`) control the type of plot you create.

Examples are:

- + Points, text
 - + (zero-dimensional geometric objects)
- + Line, path
 - + (one-dimensional geometric objects)
- + Polygon, interval
 - + (two-dimensional geometric objects)
- + More complicated: boxplot



Relationship between stats and geoms

- + Every statistic has a default geometric object,
 - + and every geometric object has a default statistic.
- + Stats and geoms are not completely orthogonal:
 - + not every combination is valid
 - + (although many are).



For example:

- + stat_bin and geom_bar
 - + is valid and
 - + standard for a histogram.
- + stat_bin and geom_point or stat_bin + geom_line
 - + are valid
 - + but non-standard combinations.
 - + They give a plot that is resembles a histogram
 - + (interpretable that same way, too)
- + stat_identity and geom_boxplot
 - + is invalid, because
 - + boxplot needs certain computed quantities from the data.



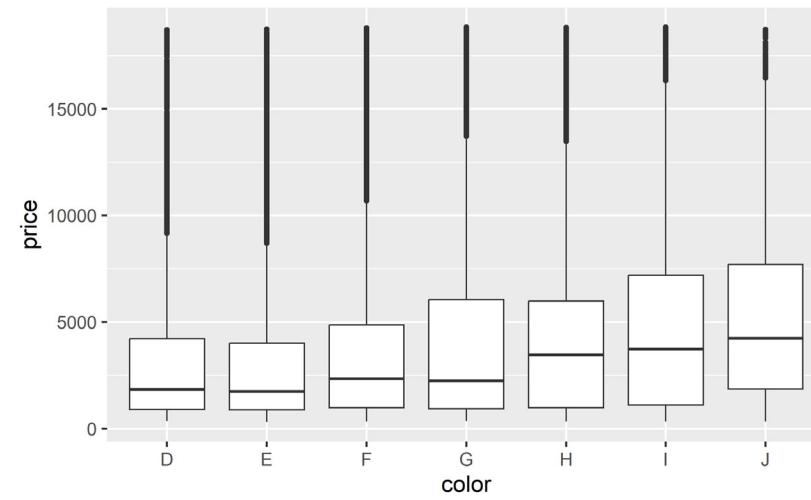
Position adjustment

Used to avoid "collisions" in the plot objects:

- + In bar plots where one of the aesthetics is height,
 - + the bars would often be plotted over each other.
 - + In this case we use the "dodge" or "stack" position adjustments.
- + If we have issues with overplotting (multiple points in exactly the same place),
 - + we can use the "jitter" position adjustment
 - + to randomly move the points a small amount.



```
(p = ggplot(diamonds, aes(x = color, y = price)) +  
  geom_boxplot())
```



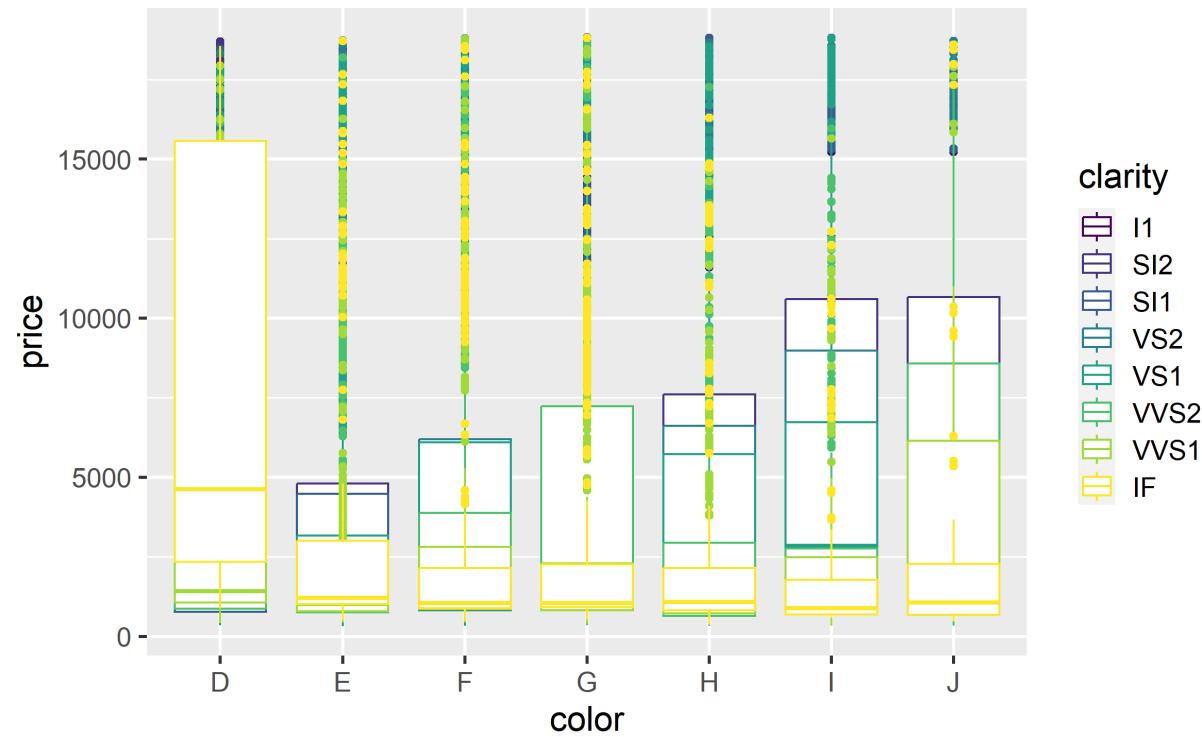
p\$layers

```
## [[1]]  
## geom_boxplot: outlier.colour = NULL, outlier.fill = NULL, outlier.shape = 19, outlier.size =  
## stat_boxplot: na.rm = FALSE, orientation = NA  
## position_dodge2
```

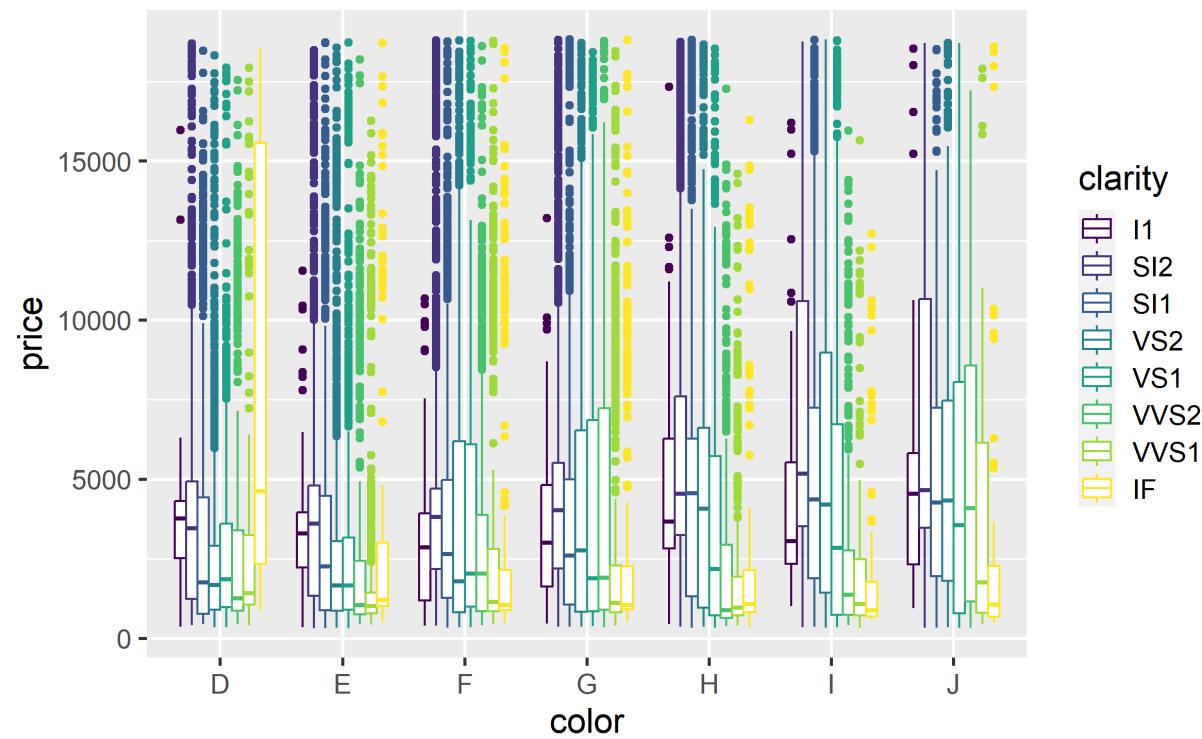


Data Science for Psychologists

```
ggplot(diamonds,aes(x = color, y = price, color = clarity)) +  
  geom_boxplot(position = "identity")
```



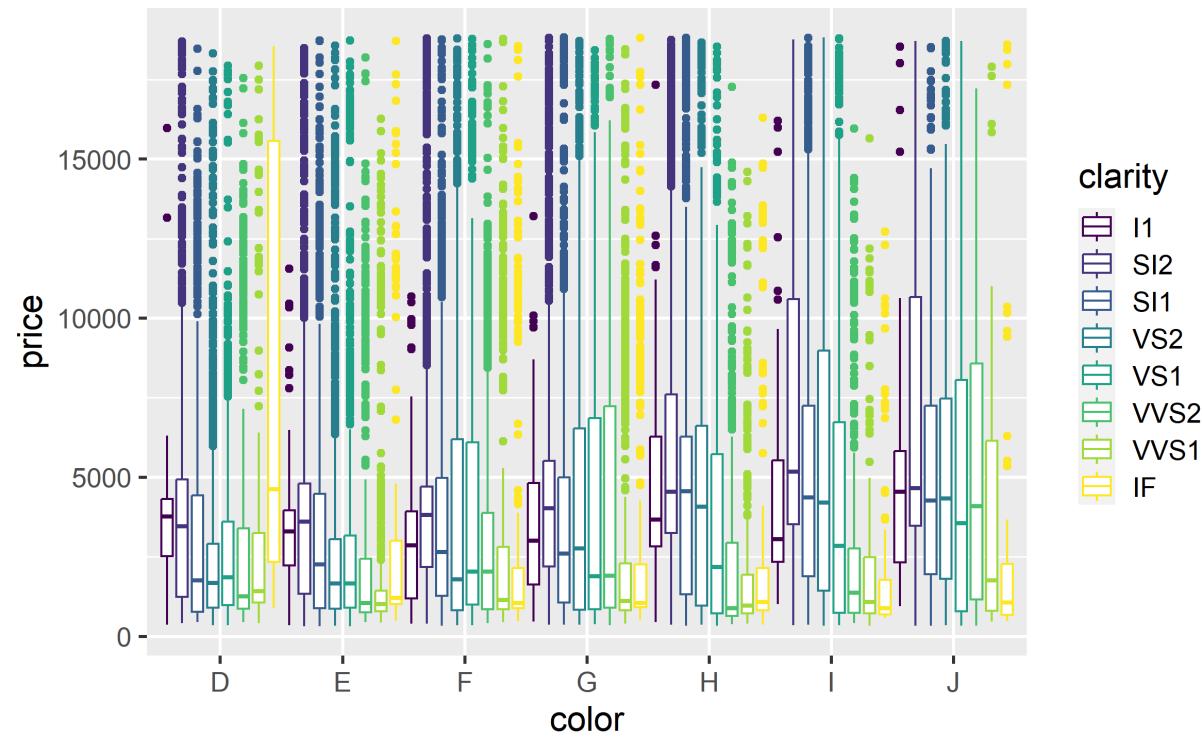
```
ggplot(diamonds,aes(x = color, y = price, color = clarity)) +  
  geom_boxplot(position = "dodge")
```



position = "dodge" is the default for boxplots, so you don't need to specify it.



```
ggplot(diamonds,aes(x = color, y = price, color = clarity)) +  
  geom_boxplot(position = position_dodge(width = 1))
```



position = "dodge" is the default for boxplots, so you don't need to specify it.



Wrapping Up...



Scales and Coordinates



Scales

- + So far, we've defined aesthetic mappings that specify,
 - + which perceivable aspects of the plot correspond
 - + to which variables.
- + However, perceivable aspects of the plot
 - + can be mapped to variables many other ways.



For example

- + If we have a categorical variable that takes values "A" and "B" to the color aesthetic,
 - + that means that color is going to represent
 - + whether variable took value "A" or "B".
- + But we could do that in a practically infinite number of ways
- + e.g.
 - + A maps to "red", B maps to "black"
 - + A maps to "green", B maps to "blue"
 - + A maps to "purple", B maps to "gold"
- + ... You probably get the picture



Modernized ggplot

- + Now, ggplot has good default mappings from values into aesthetic space,
 - + but you will sometimes want to set them by hand.
- + To do so, you use the `scale_*` functions.
- + The *old* version of ggplot had poor mappings from continuous values to colors,
 - + and the `viridis` color scheme was much better.
- + The most recent version of ggplot uses viridis by default
 - + for both continuous values and ordered factors.



Coordinate system

Another aspect of the plot that can be specified independently of everything else is the coordinate system.

- + `coord_cartesian` is the default,
 - + and is almost always what you want.
- + `coord_flip` is sometimes useful:
 - + for example, boxplots require the explanatory variable to be mapped to x,
 - + so if you want a horizontal boxplot you need to use `coord_flip`.
 - + `coord_polar` will often make your plots look cooler
 - + and more difficult to read.
 - + Not usually recommended.



Faceting

Allows you to make small multiples of plots.

- + Other grammars/plotting systems implement faceting with a fancy coordinate system,
- + but it turns out that it's easier to use if you think about it separately.
- + Each facet plots a subset of the data,
 - + and it takes as input
 - + what variable(s) to use to make the subsets and
 - + how to lay out the subsets.



Facet options

- + facet_wrap:
 - + Lays out the plots for each subset sequentially.
- + facet_grid:
 - + Subsets the data according to two separate variables.
- + The facet position along the x -axis represents levels of one variable,
 - + and the facet position along the y -axis represents levels of the other variable.



Wrapping Up...



Data Science for Psychologists

How this all works



Data Science for Psychologists

The long way

- + One way to specify a ggplot is to
 - + specify all of the components we've seen.
- + If you understand all the parts,
 - + this way is probably the least confusing method to specify a ggplot.
- + The problem is that it's verbose.

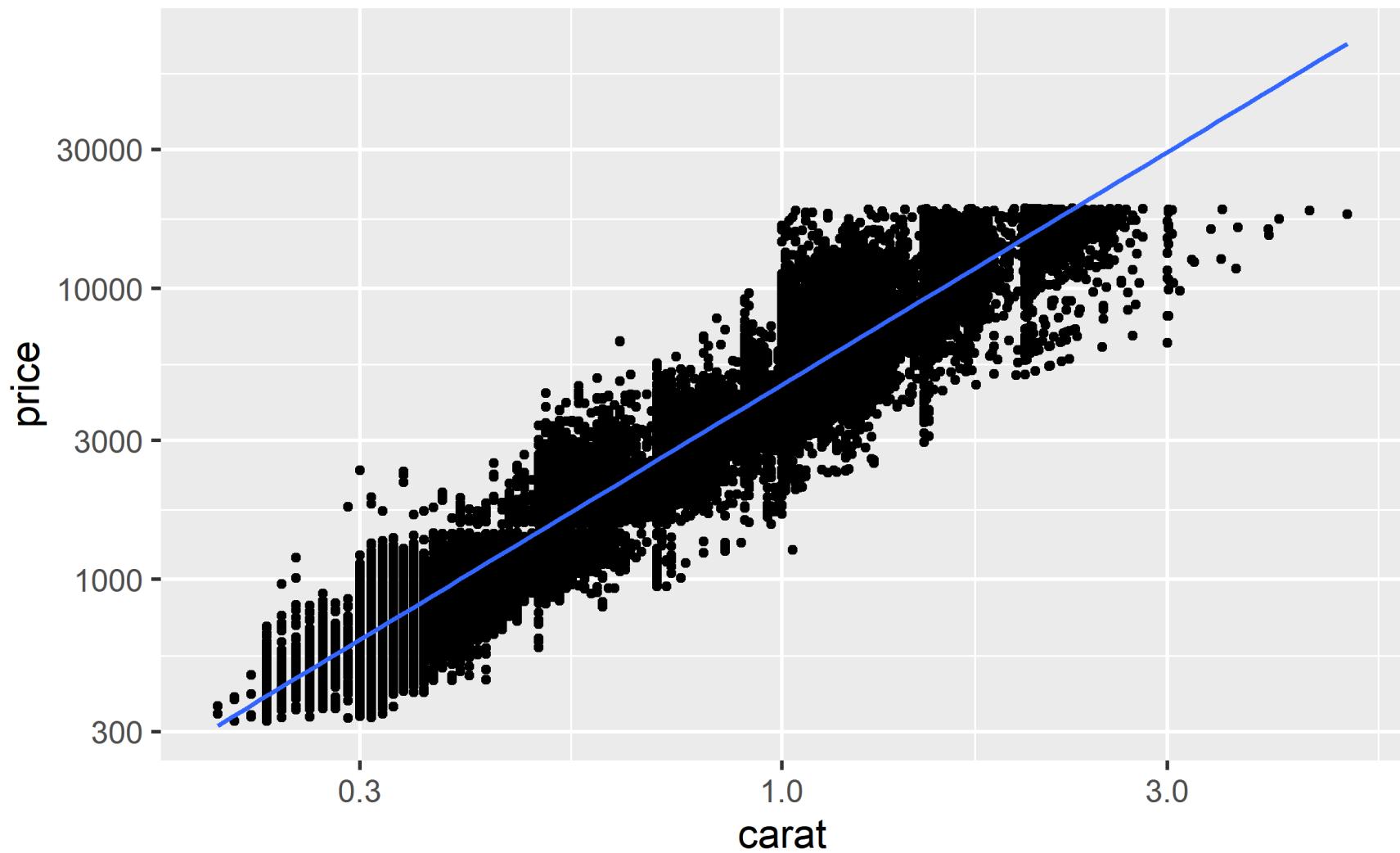


Verbose

- + Suppose we want to make a plot with points and a smoother
 - + from the diamonds dataset.
- + We can specify data, mapping, geom, stat, and positions for each layer,
 - + along with scales and the coordinate system as follows:

```
ggplot() +  
  layer(  
    data = diamonds, mapping = aes(x = carat, y = price),  
    geom = "point", stat = "identity", position = "identity") +  
  layer(  
    data = diamonds, mapping = aes(x = carat, y = price),  
    geom = "smooth", position = "identity", stat = "smooth", params = list(method = "lm")) +  
  scale_x_log10() + scale_y_log10() + coord_cartesian()
```





Defaults make the code shorter

The more standard way of writing the same plot would be:

```
p = ggplot(data = diamonds, aes(x = carat, y = price)) +  
  geom_point() +  
  stat_smooth(method = "lm") +  
  scale_x_log10() +  
  scale_y_log10()
```

This code is still fairly long, but we don't have to specify...

- + position: Default for both `geom_point` and `stat_smooth` is `position = "identity"`.
- + stat, for `geom_point`: The default stat for `geom_point` is `stat = "identity"`.
- + geom, for `stat_smooth`: The default geom for `stat_smooth` is `geom_smooth`.
- + coordinate system: `coord_cartesian` is always the default.



You can check what stat, geom, and position is used for each of the layers: .medi[

```
names(p)
```

```
## [1] "data"          "layers"        "scales"        "mapping"  
## [5] "theme"         "coordinates"   "facet"         "plot_env"  
## [9] "labels"
```

```
p$layers
```

```
## [[1]]  
## geom_point: na.rm = FALSE  
## stat_identity: na.rm = FALSE  
## position_identity  
##  
## [[2]]  
## geom_smooth: se = TRUE, na.rm = FALSE, orientation = NA  
## stat_smooth: method = lm, formula = NULL, se = TRUE, n = 80, fullrange = FALSE, level = 0.95  
## position_identity
```



Wrapping Up...



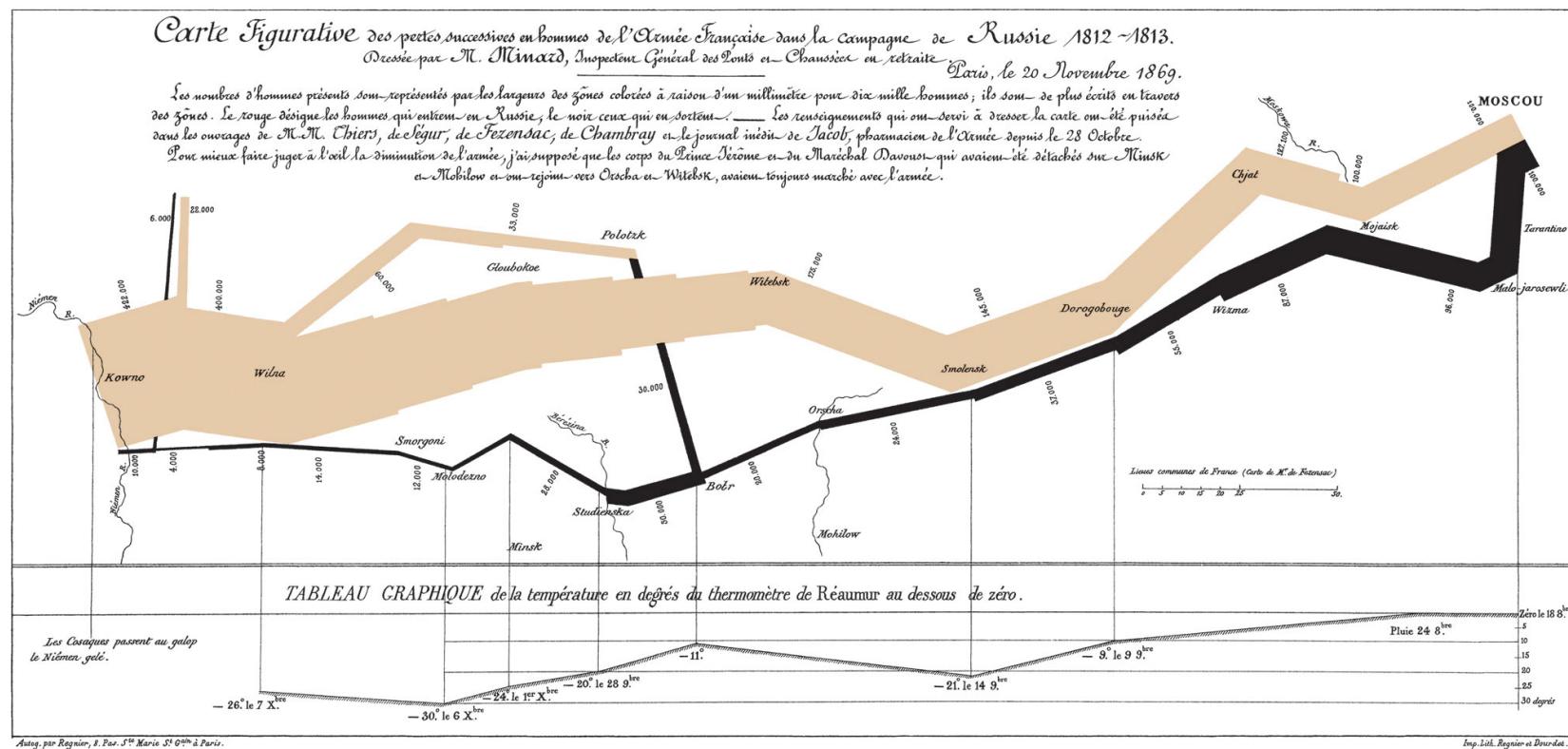
Data Science for Psychologists

Minard on Napoleon's invasion of Russia



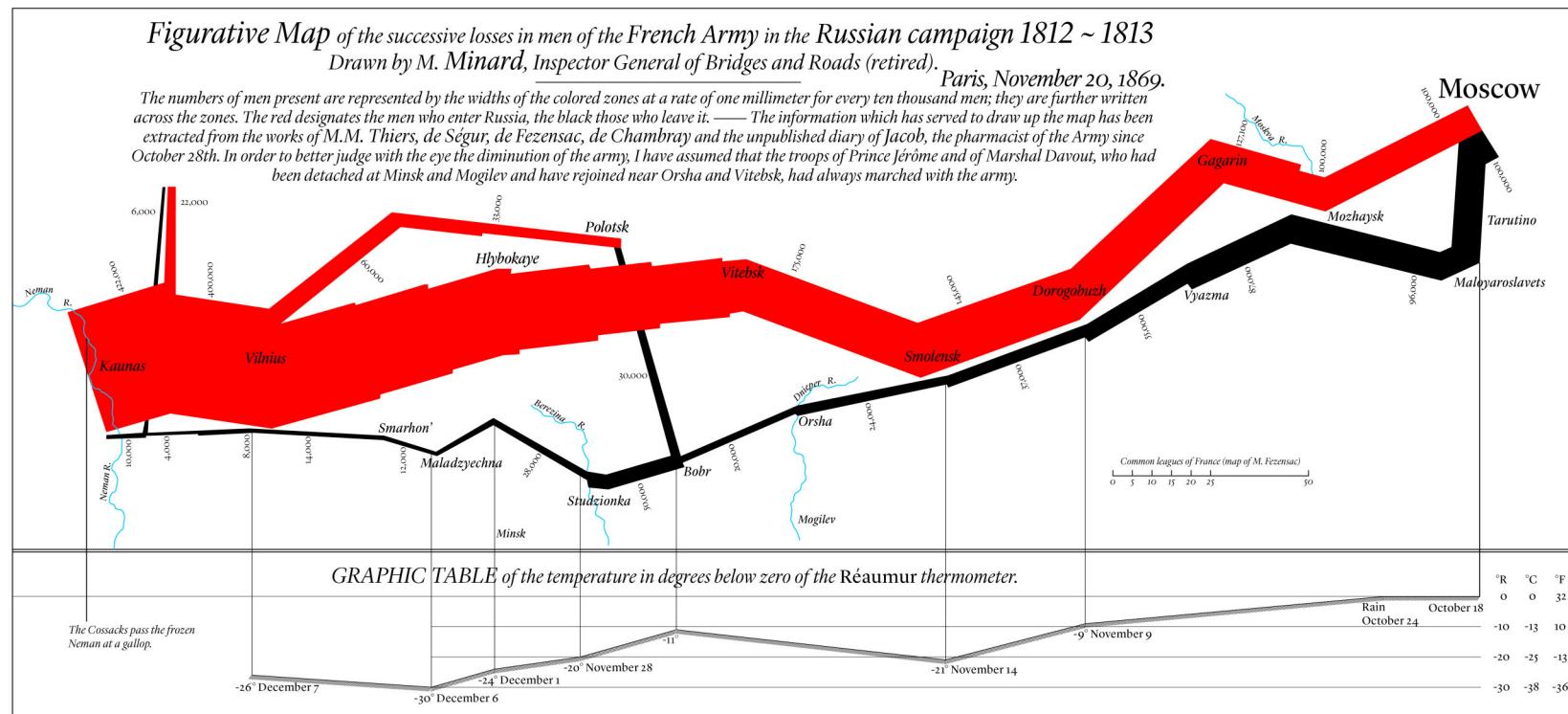
Minard on Napoleon's invasion of Russia

One of the most famous statistical graphics, created by Charles Minard depicts Napoleon's 1812 invasion of and retreat from Russia.

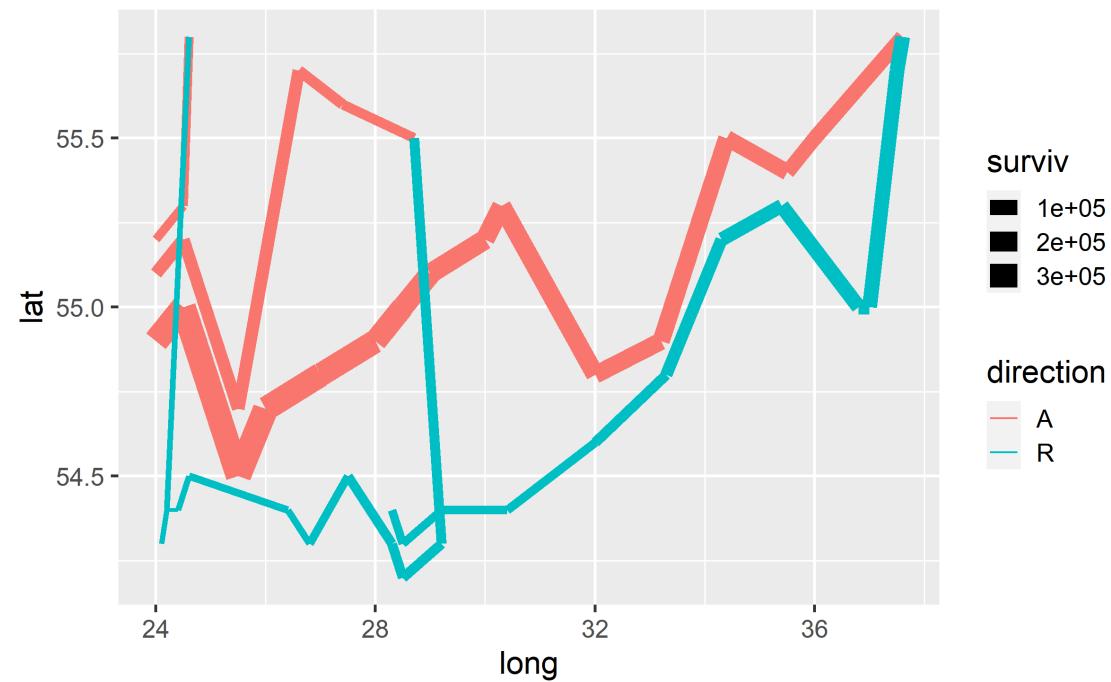


Minard on Napoleon's invasion of Russia

One of the most famous statistical graphics, created by Charles Minard depicts Napoleon's 1812 invasion of and retreat from Russia.

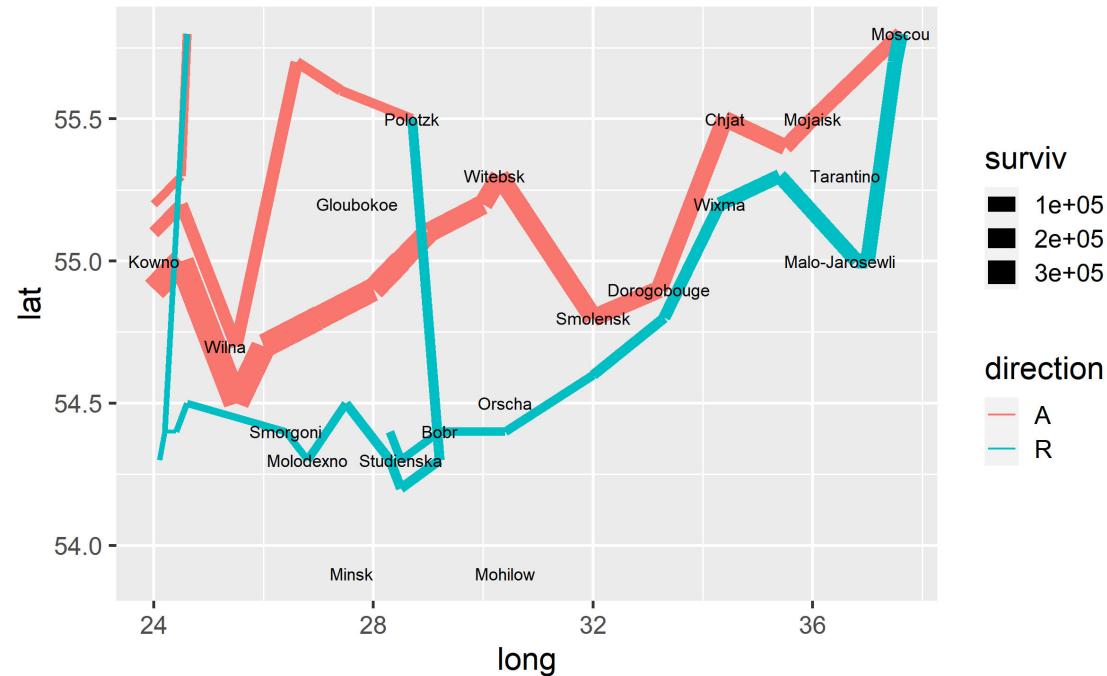


```
minard = read_csv("data/minard.csv");minard_cities = read_csv("data/minard-cities.csv")
(plot_troops = ggplot(minard) +
  geom_path(aes(x = long, y = lat,
                 color = direction,
                 size = surviv,
                 group = division)))
```



Let's add another layer for the cities:

```
(plot_both = plot_troops +
    geom_text(aes(x = long, y = lat, label = city),
              data = minard_cities, size = 3))
```



Notice: we have a new dataset for this layer.

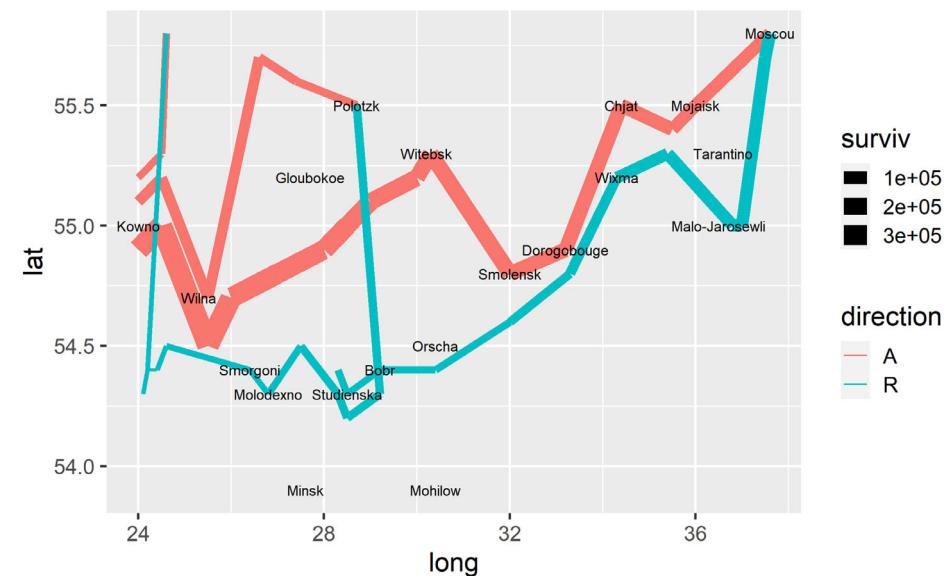


Data Science for Psychologists

Next Steps

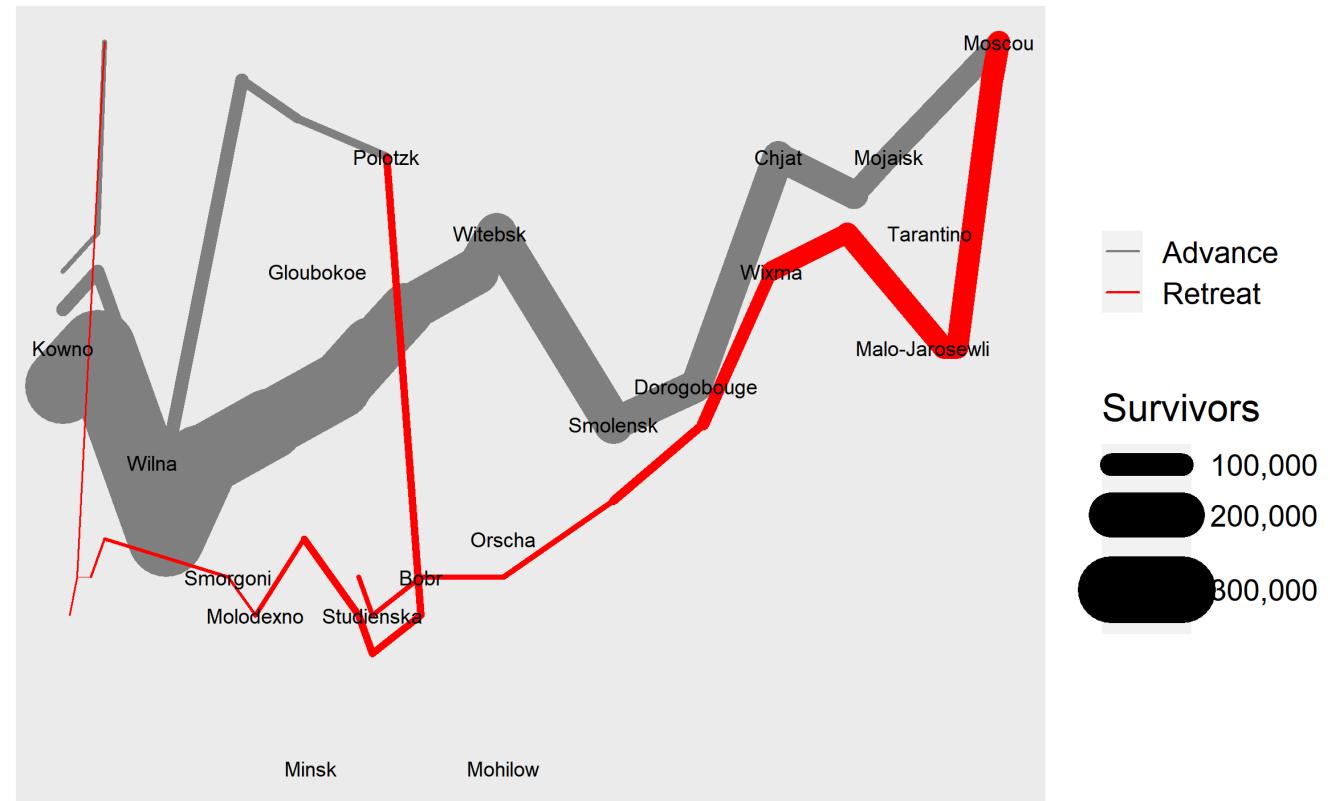
Some things that are still not great about this plot:

- + We would like the colors
 - + for advance and retreat to be gray and red.
- + We want the line widths to be
 - + proportional to the number of survivors.
- + We would like the line ends to be
 - + round instead of square.



An improved version of the plot, with better scales:

```
ggplot(minard) +
  geom_path(aes(x = long,
                 y = lat,
                 color = direction,
                 size = surviv^2,
                 group = division),
            lineend = "round") +
  geom_text(aes(x = long,
                 y = lat,
                 label = city),
            data = minard_cities,
            size = 3) +
  scale_size(range = c(.18, 15),
             breaks = (1:3 * 10^5)^2,
             labels =
               scales::comma(1:3 * 10^5),
             "Survivors") +
  scale_color_manual(values = c("grey50",
                                "red"),
                     breaks = c("A",
                               "R"),
                     labels = c("Advance",
                               "Retreat"),
                     "") +
  xlab(NULL) + ylab(NULL) +
  theme(axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.grid = element_blank())
```



Your turn!

- + Try creating a histogram on the diamonds dataset, for example with...

```
p = ggplot() + geom_histogram(aes(x = carat), data = diamonds)
```

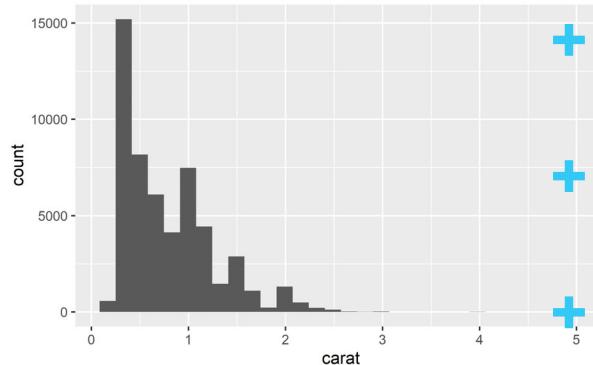
- + Re-write this using the `layer` function.

Tip:

- + If you don't know what the default values for some of the aspects of the plot, examine `p$layers`
- + Remember that a histogram is a plot with `stat_bin` and `geom_bar`.



Your turn!



- + Modify your histogram code so that it uses a different geom, for example `geom_line` or `geom_point`.
- + This change should be simple once you have the `layer` specification of a histogram.
- + In your histogram,
 - + add an aesthetic mapping from one of the factor variables (maybe color or clarity) to the color aesthetic.

What is the default position adjustment for a histogram?

Tip:

- + Try changing the position adjustment to something different
 - + (try `position_dodge`).



Data Science for Psychologists

Difficult Stretch Goal

- + Recreate the Minard map more precisely.
- + Add text for the number of troops surviving on each segment,
 - + and add the time and temperature data to the bottom of the plot.



ggplot and EDA

Remember our passage from Tukey:

Exploratory data analysis is detective work... As all detective stories remind us, many of the circumstances surrounding a crime are accidental or misleading. Equally, many of the indications to be discerned in bodies of data are accidental or misleading. To accept all appearances as conclusive would be destructively foolish, either in crime detection or in data analysis. To fail to collect all appearances because some -- or even most -- are only accidents would, however, be gross misfeasance deserving (and often receiving) appropriate punishment.



Concluding Thoughts

+

The flexibility in the grammar of graphics allows us to collect many more "appearances" in the data than we would be able to,

- + if we just have access to a handful of named plots.

+

Many of the plots that we can make with ggplot are not useful,

- + but the point is to try visualizing the data in many different ways.

+ ggplot opens up a very large space of statistical graphics to us for not very much effort.



Sources

- + Mine Å‡etinkaya-Rundel's Data Science in a Box ([link](#))
- + Julia Fukuyama's EDA ([link](#))



Wrapping Up...



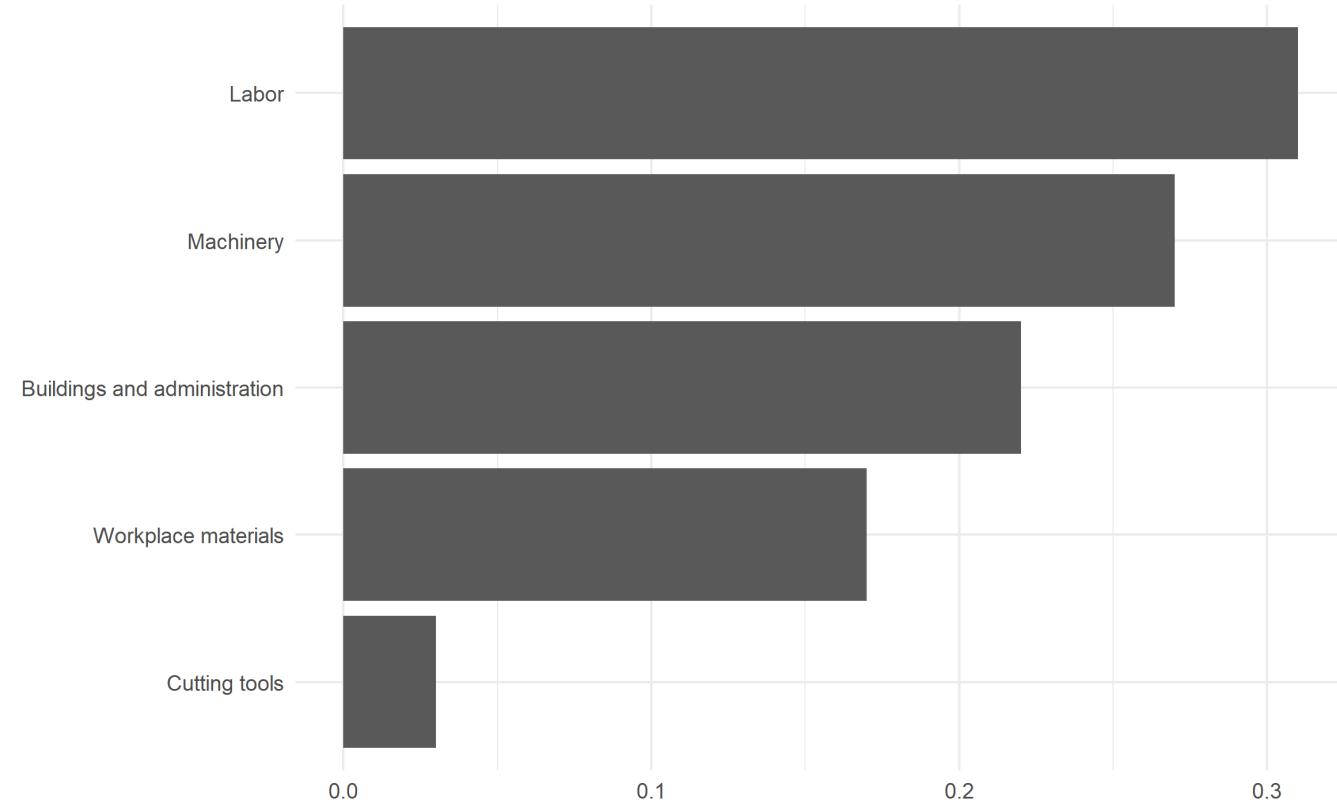
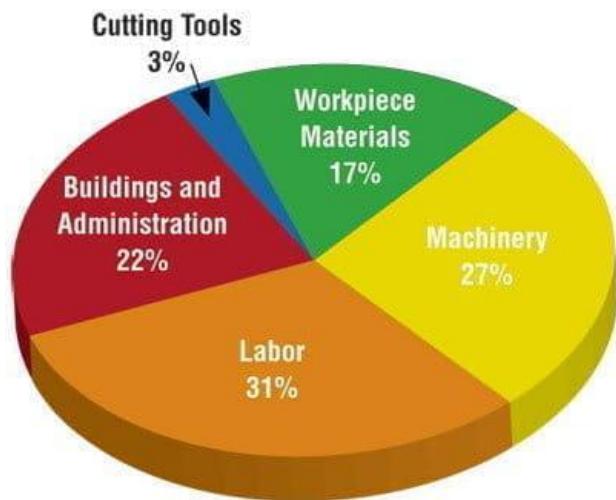
Tips for effective data visualization



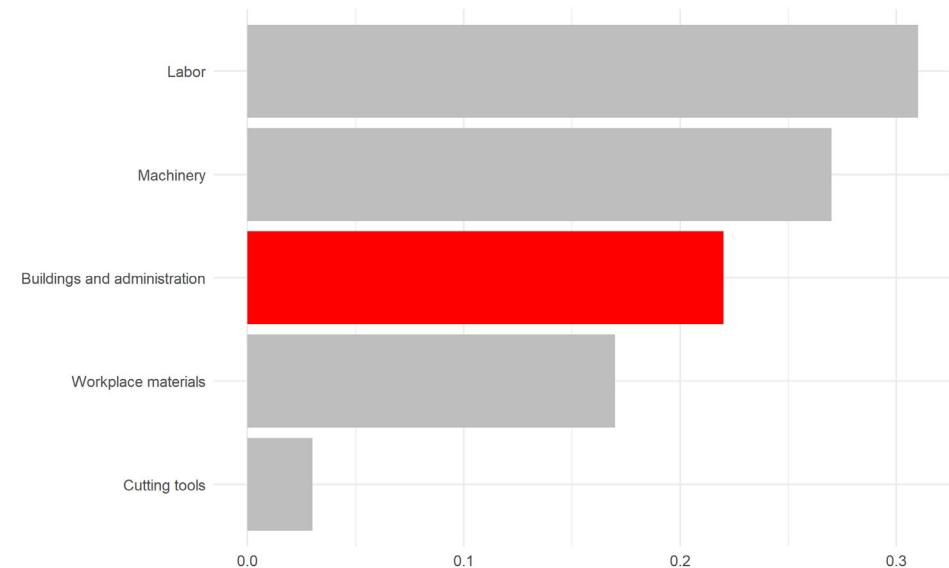
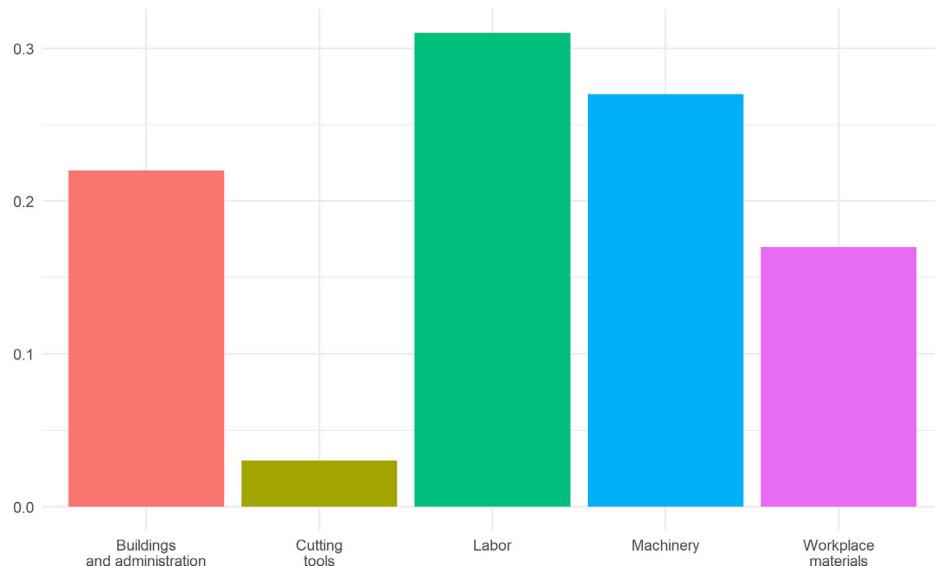
Designing effective visualizations



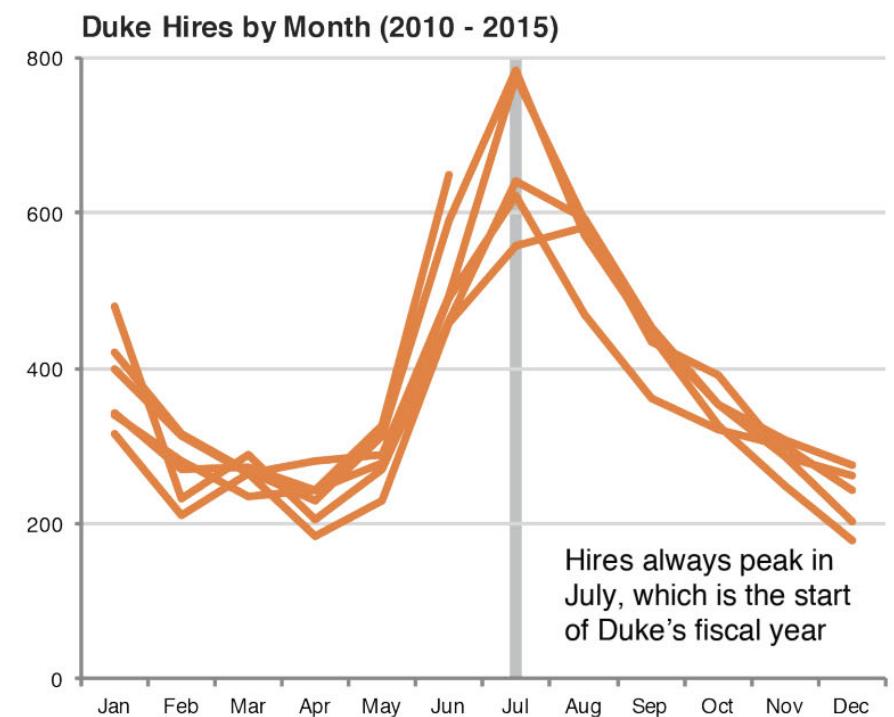
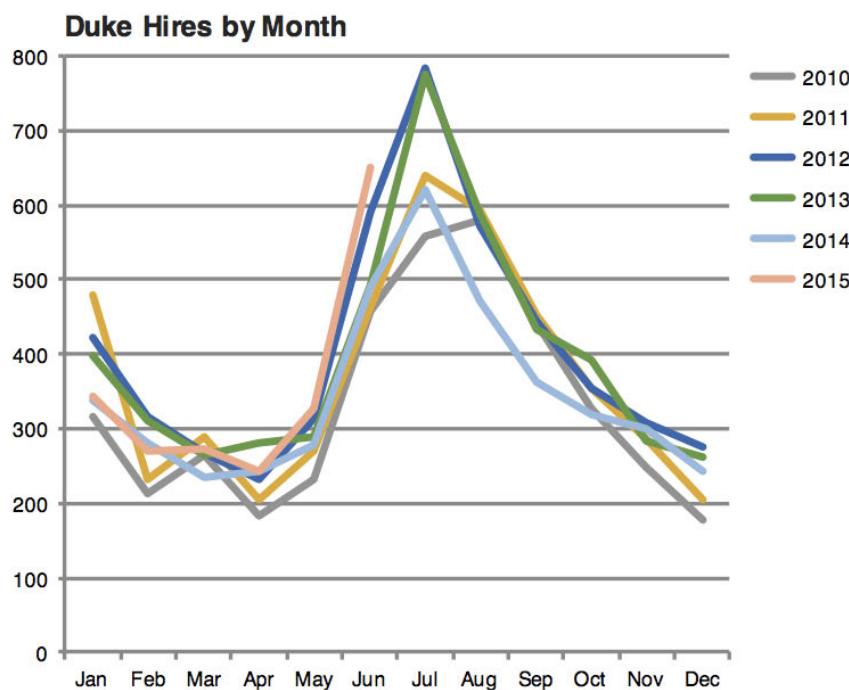
Keep it simple



Use color to draw attention



Tell a story



Credit: Angela Zoss and Eric Monson, Duke DVS



Data Science for Psychologists

Wrapping Up...



Data Science for Psychologists

Principles for effective visualizations



Principles for effective visualizations

- + Order matters
- + Put long categories on the y-axis
- + Keep scales consistent
- + Select meaningful colors
- + Use meaningful and nonredundant labels

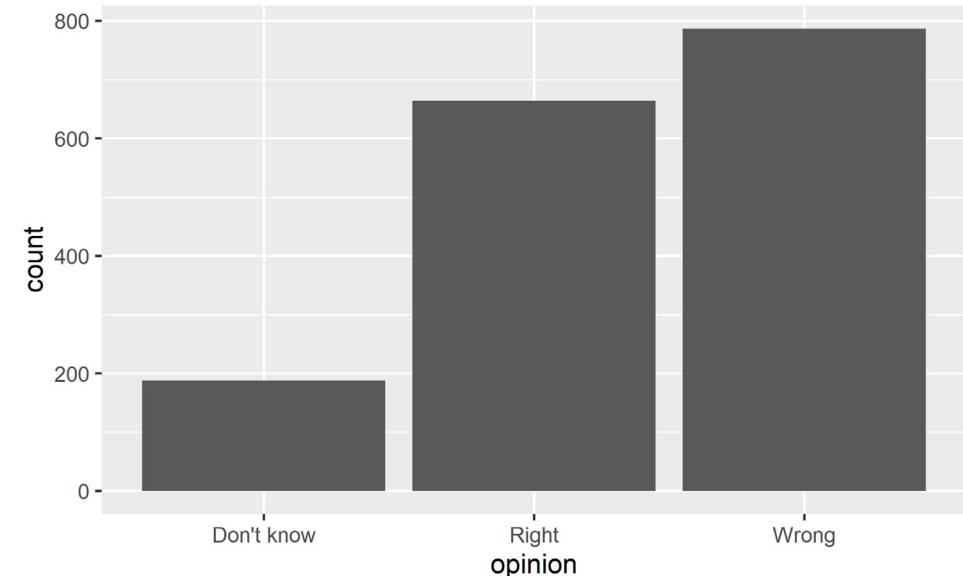


Data

In September 2019, YouGov survey asked 1,639 GB adults the following question:

In hindsight, do you think Britain was right/wrong to vote to leave EU?

- + Right to leave
- + Wrong to leave
- + Don't know



Source: YouGov Survey Results, retrieved Oct 7, 2019



Data Science for Psychologists

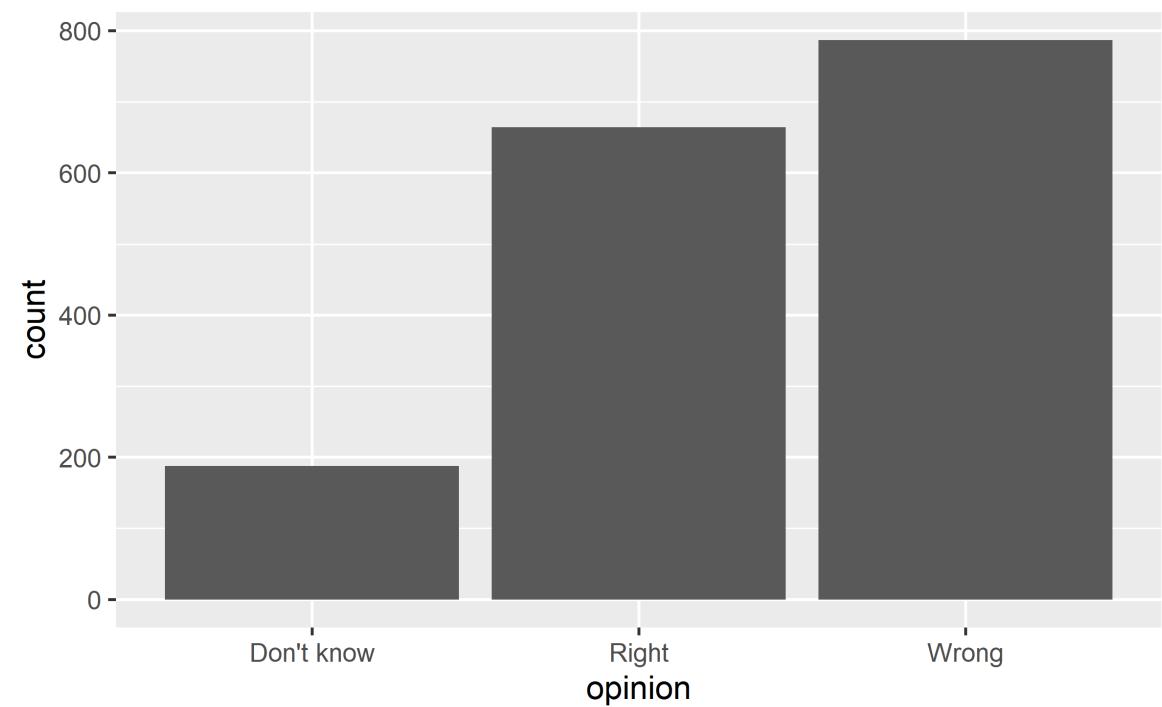
Order matters



Data Science for Psychologists

Alphabetical order is rarely ideal

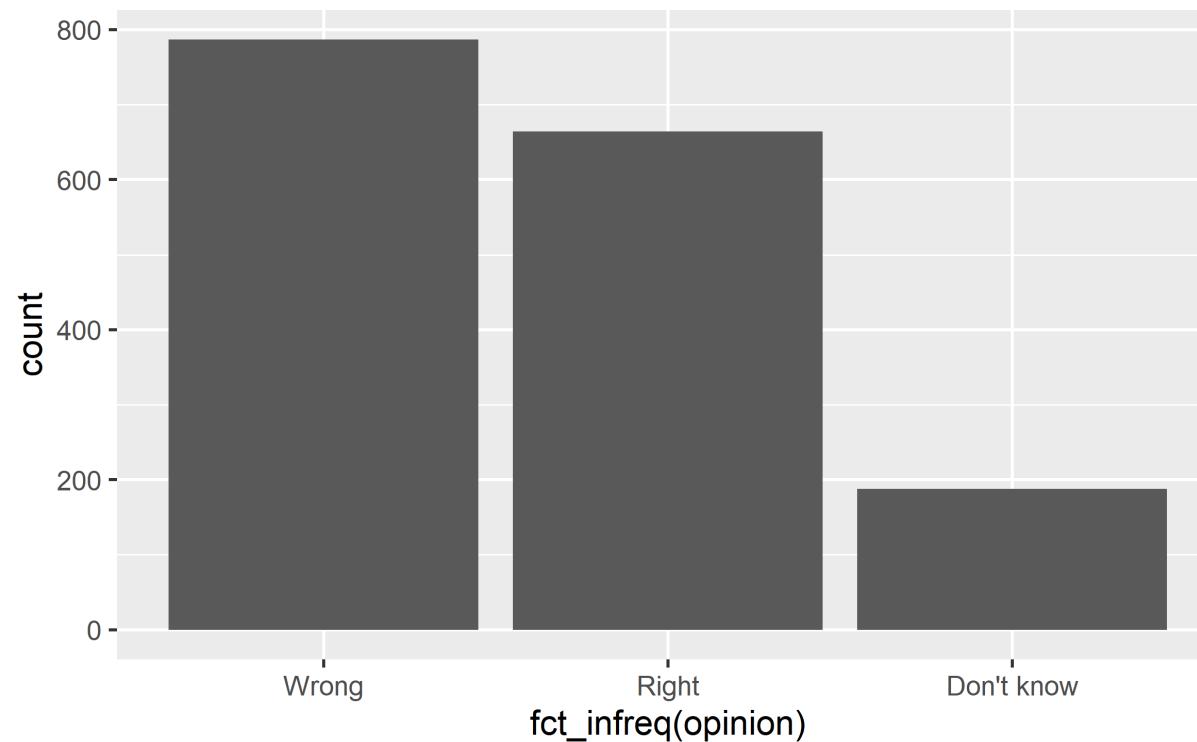
```
ggplot(brexit, aes(x = opinion)) +  
  geom_bar()
```



Order by frequency

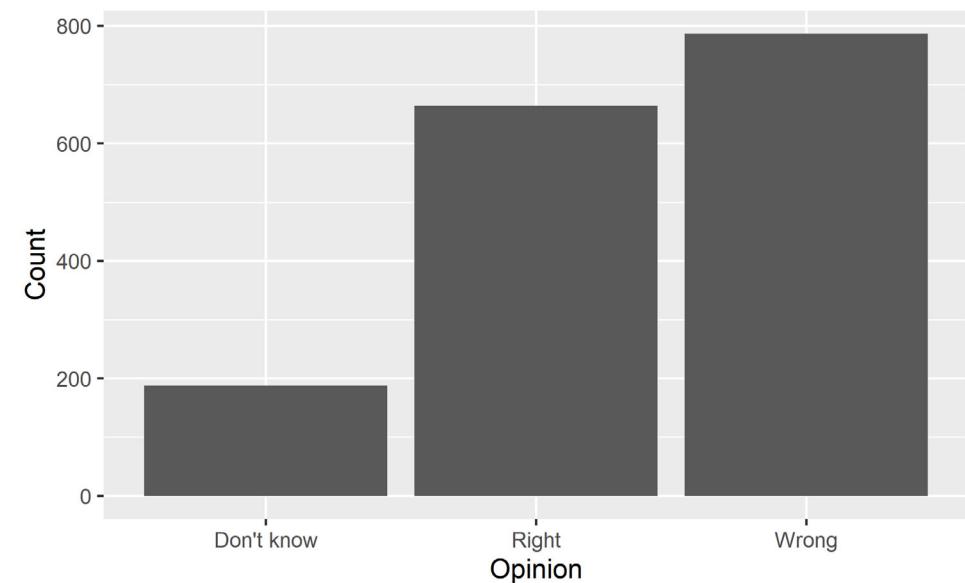
fct_infreq: Reorder factors' levels by frequency

```
ggplot(brexit, aes(x = fct_infreq(opinion))) +  
  geom_bar()
```



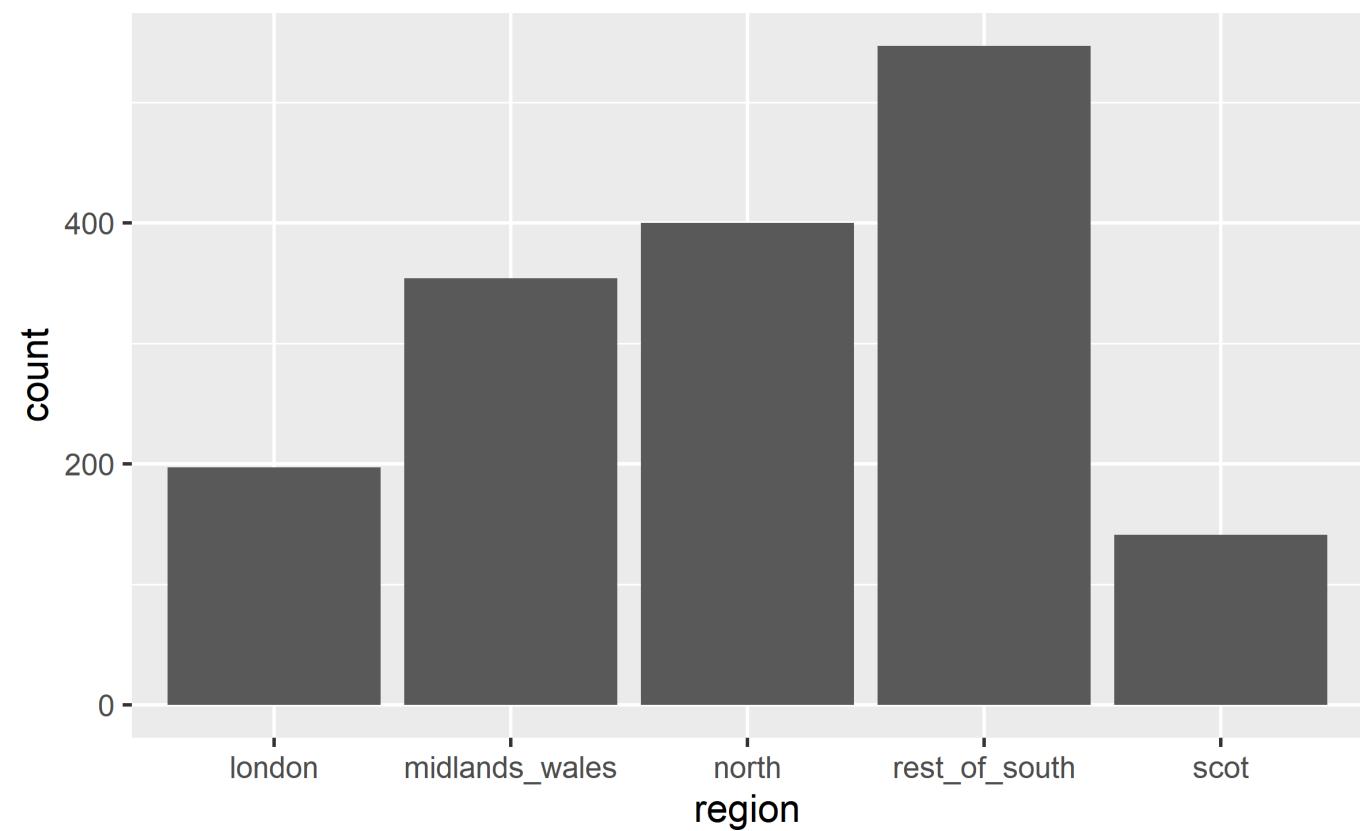
Clean up labels

```
ggplot(brexit, aes(x = opinion)) +  
  geom_bar() +  
  labs(  
    x = "Opinion",  
    y = "Count"  
)
```



Alphabetical order is rarely ideal

```
ggplot(brexit, aes(x = region)) +  
  geom_bar()
```

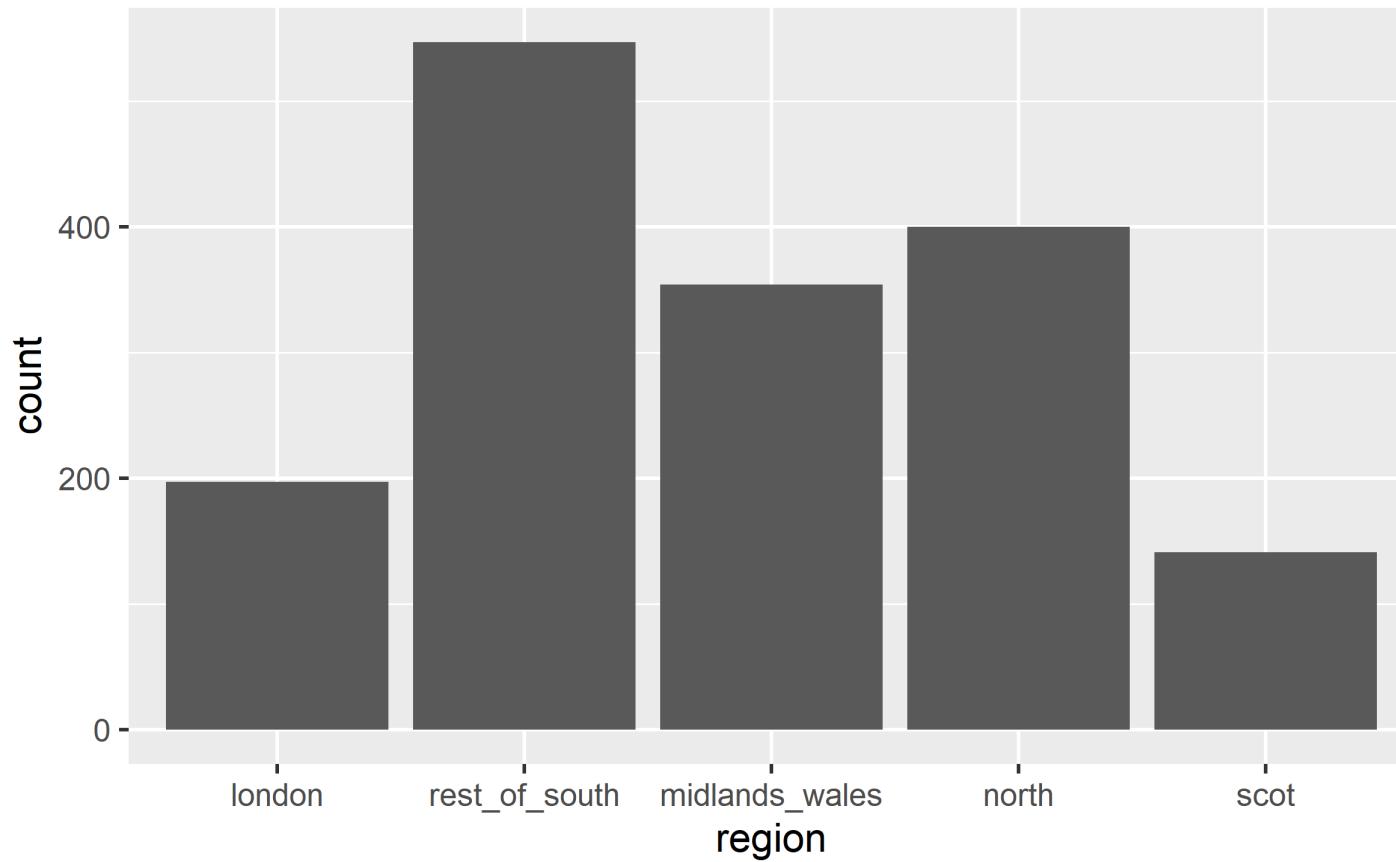


Use inherent level order

`fct_relevel`: Reorder factor levels using a custom order

```
brexit <- brexit %>%
  mutate(
    region = fct_relevel(
      region,
      "london", "rest_of_south", "midlands_wales", "north", "scot"
    )
  )
```





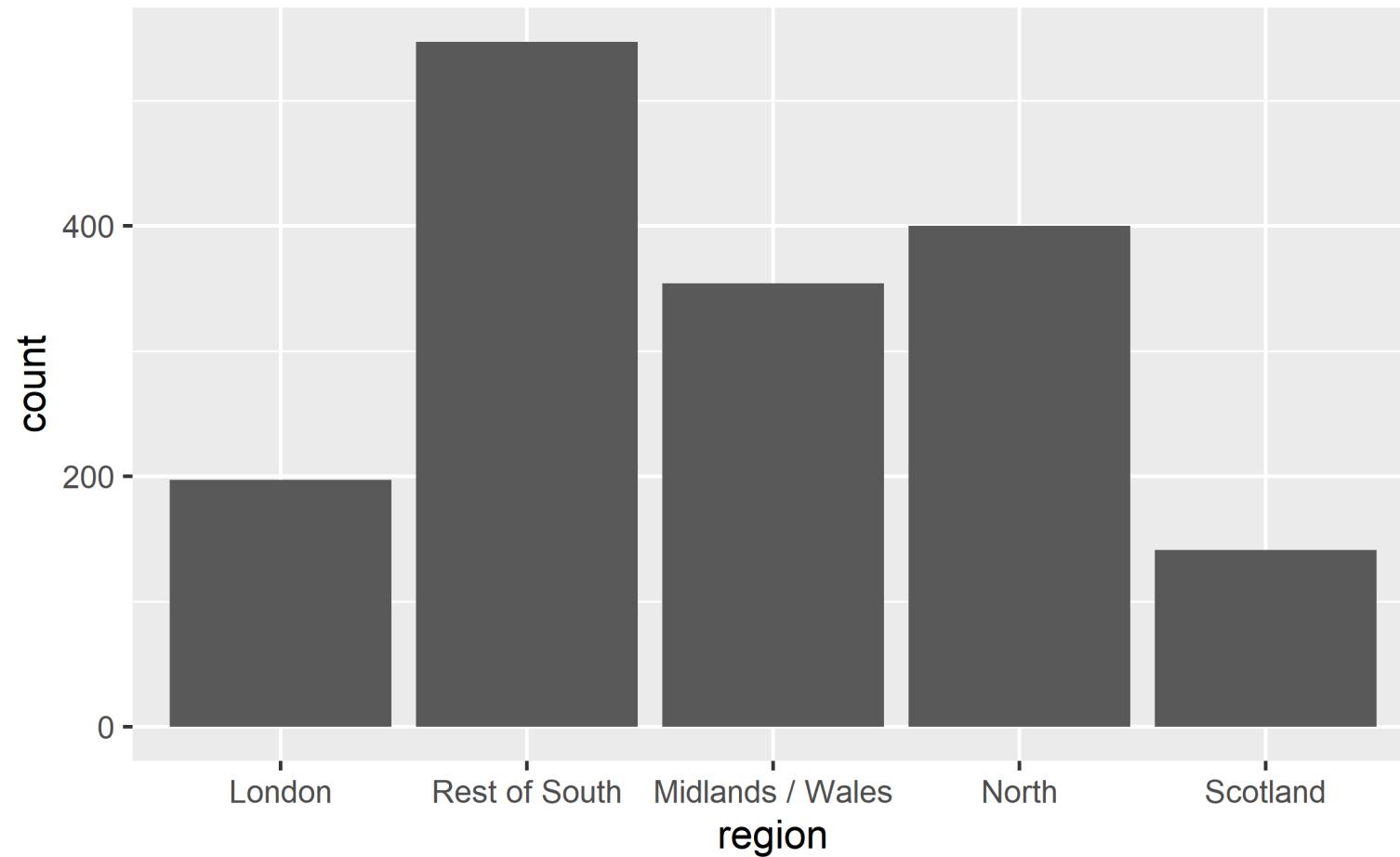
Clean up labels

fct_recode: Change factor levels by hand

```
brexit <- brexit %>%
  mutate(
    region = fct_recode(
      region,
      London = "london",
      `Rest of South` = "rest_of_south",
      `Midlands / Wales` = "midlands_wales",
      North = "north",
      Scotland = "scot"
    )
  )
```



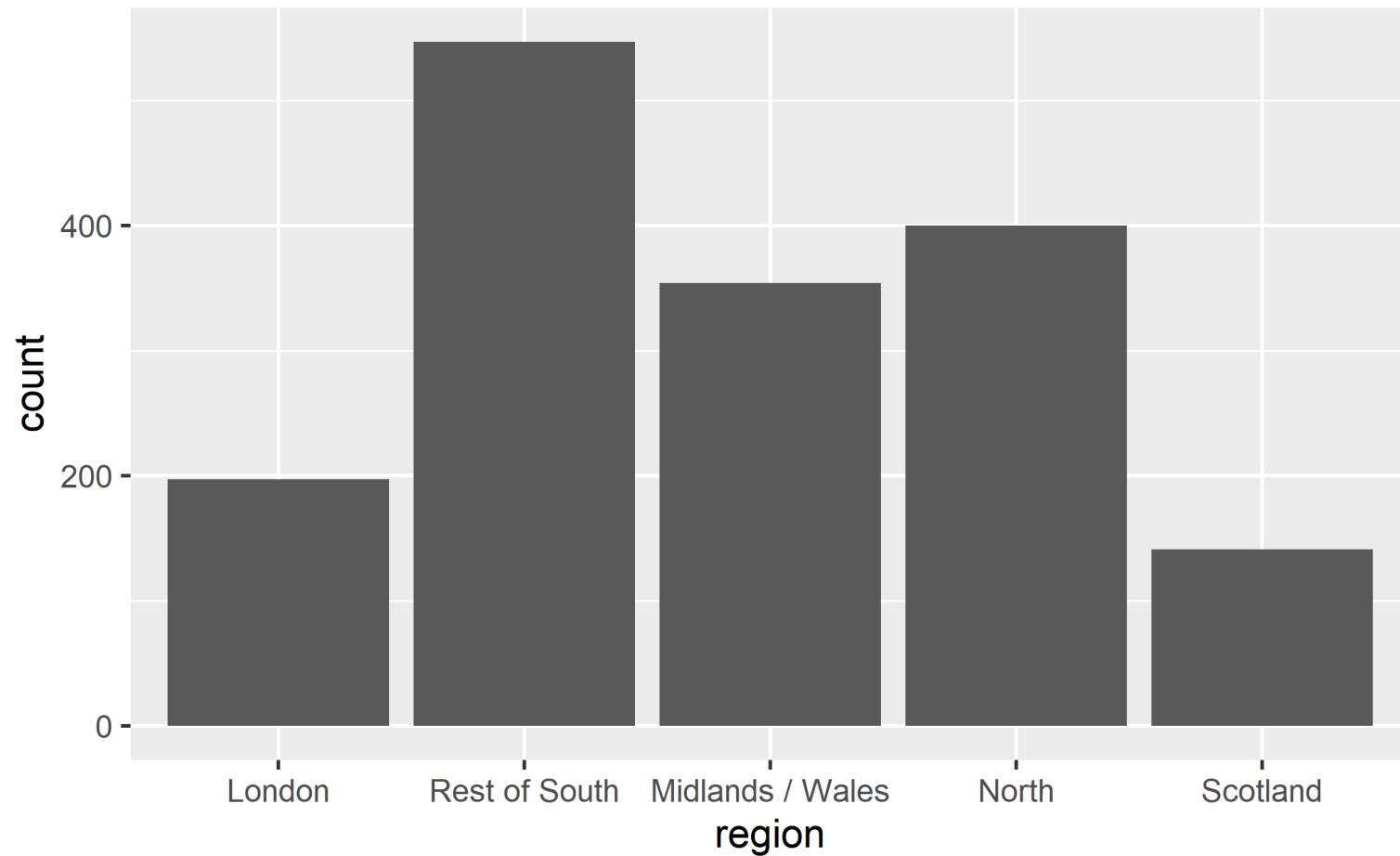
Clean!



Put long categories on the y-axis

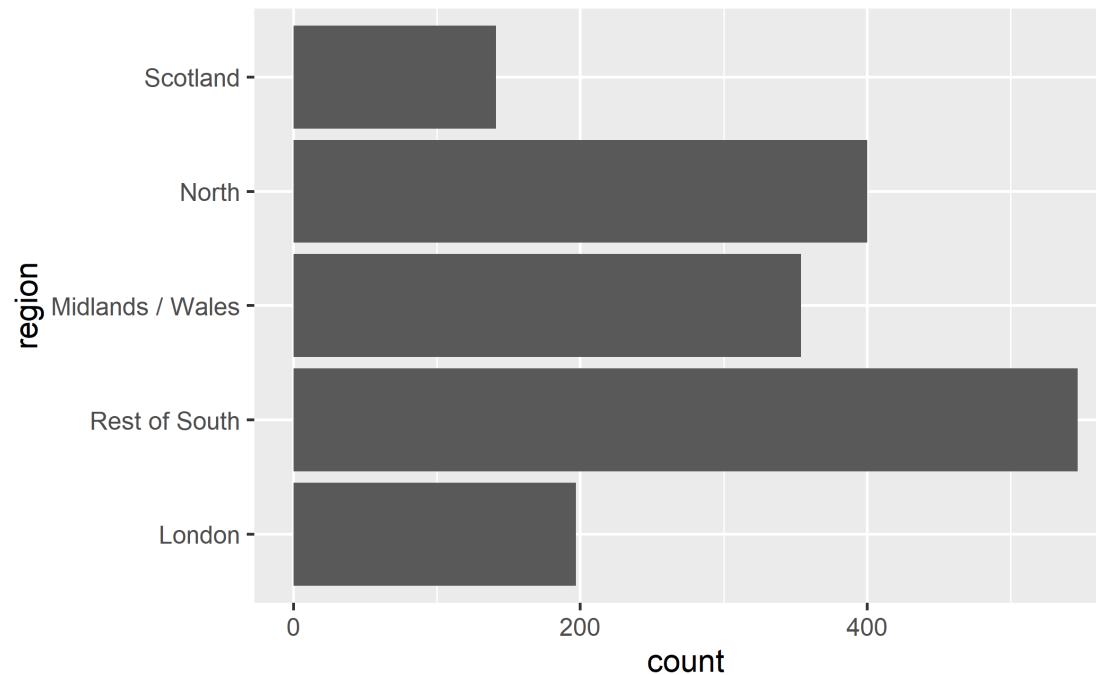


Long categories can be hard to read



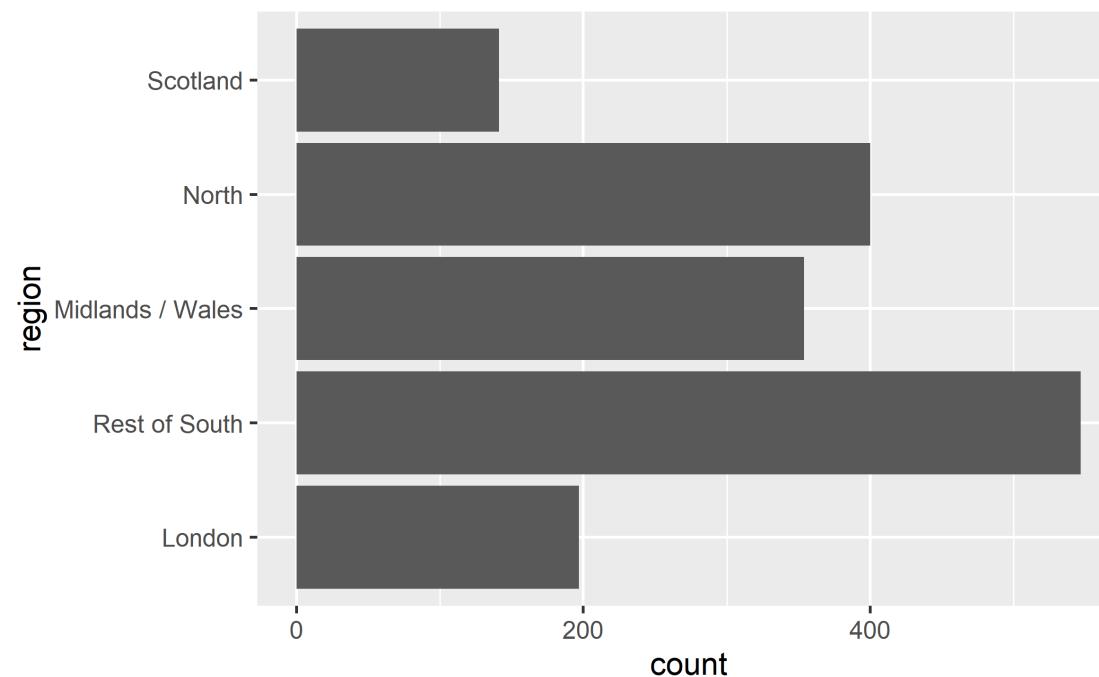
Move them to the y-axis

```
ggplot(brexit, aes(x = region)) +  
  geom_bar() +  
  coord_flip()
```



Move them to the y-axis

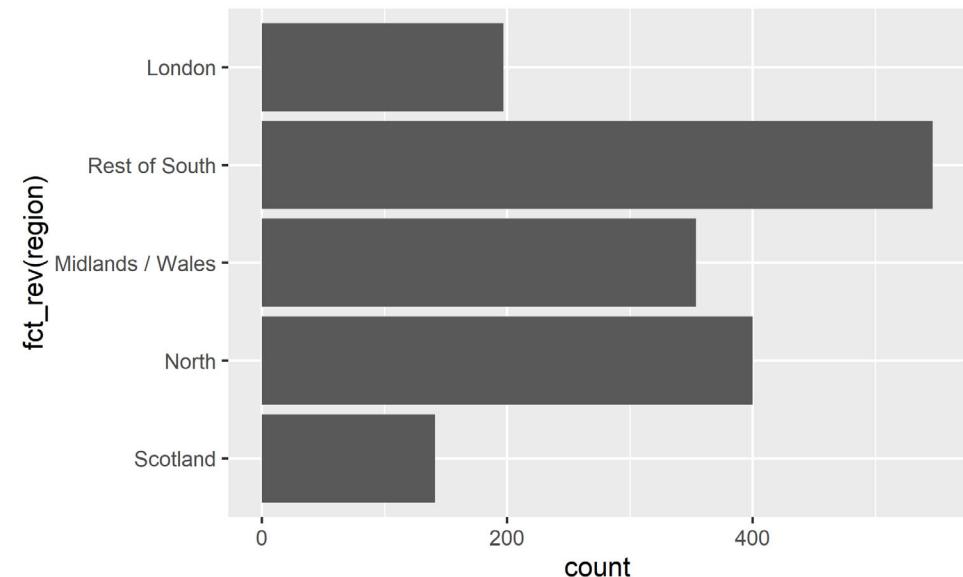
```
ggplot(brexit, aes(y = region)) +  
  geom_bar()
```



And reverse the order of levels

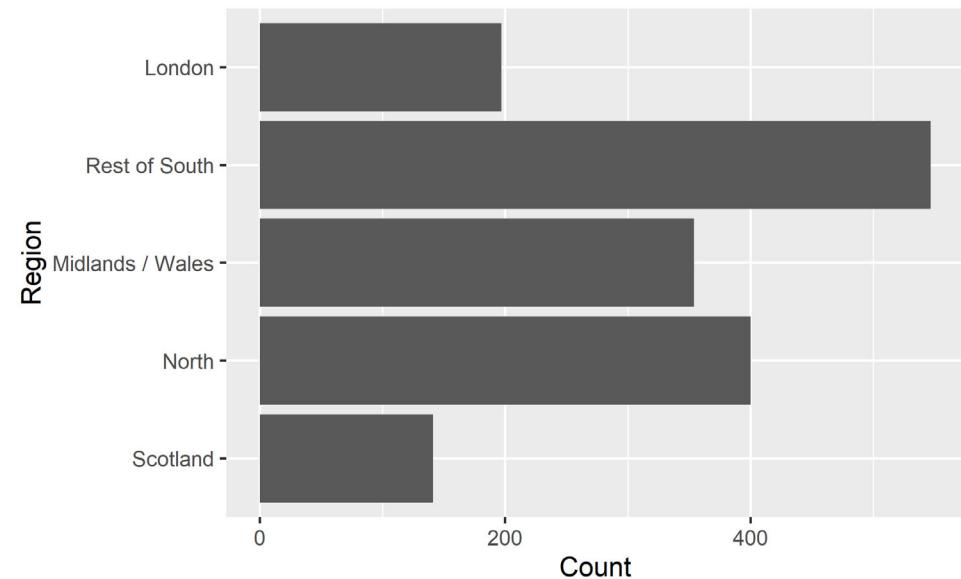
fct_rev: Reverse order of factor levels

```
ggplot(brexit, aes(y = fct_rev(region))) +  
  geom_bar()
```



Clean up labels

```
ggplot(brexit, aes(y = fct_rev(region))) +  
  geom_bar() +  
  labs(  
    x = "Count",  
    y = "Region"  
)
```



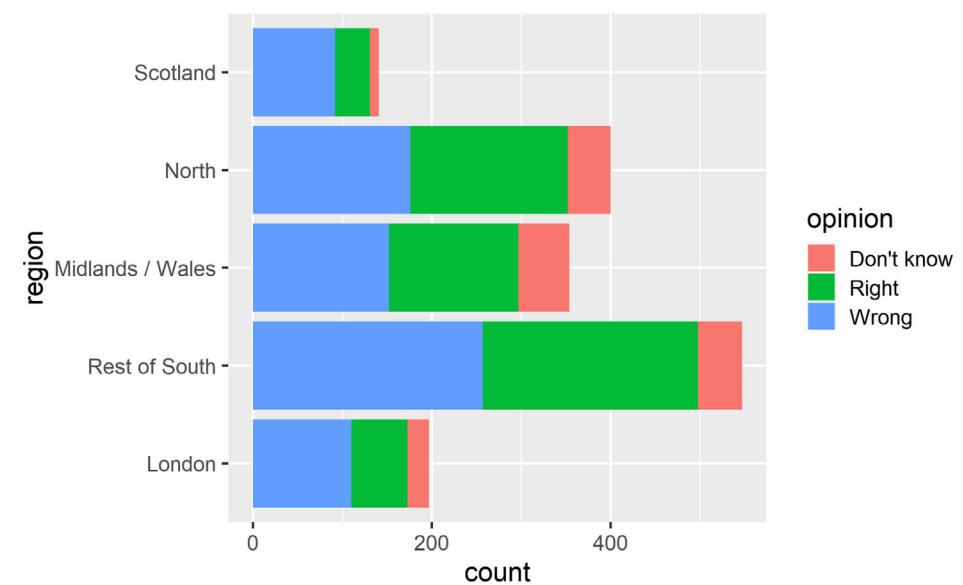
Pick a purpose

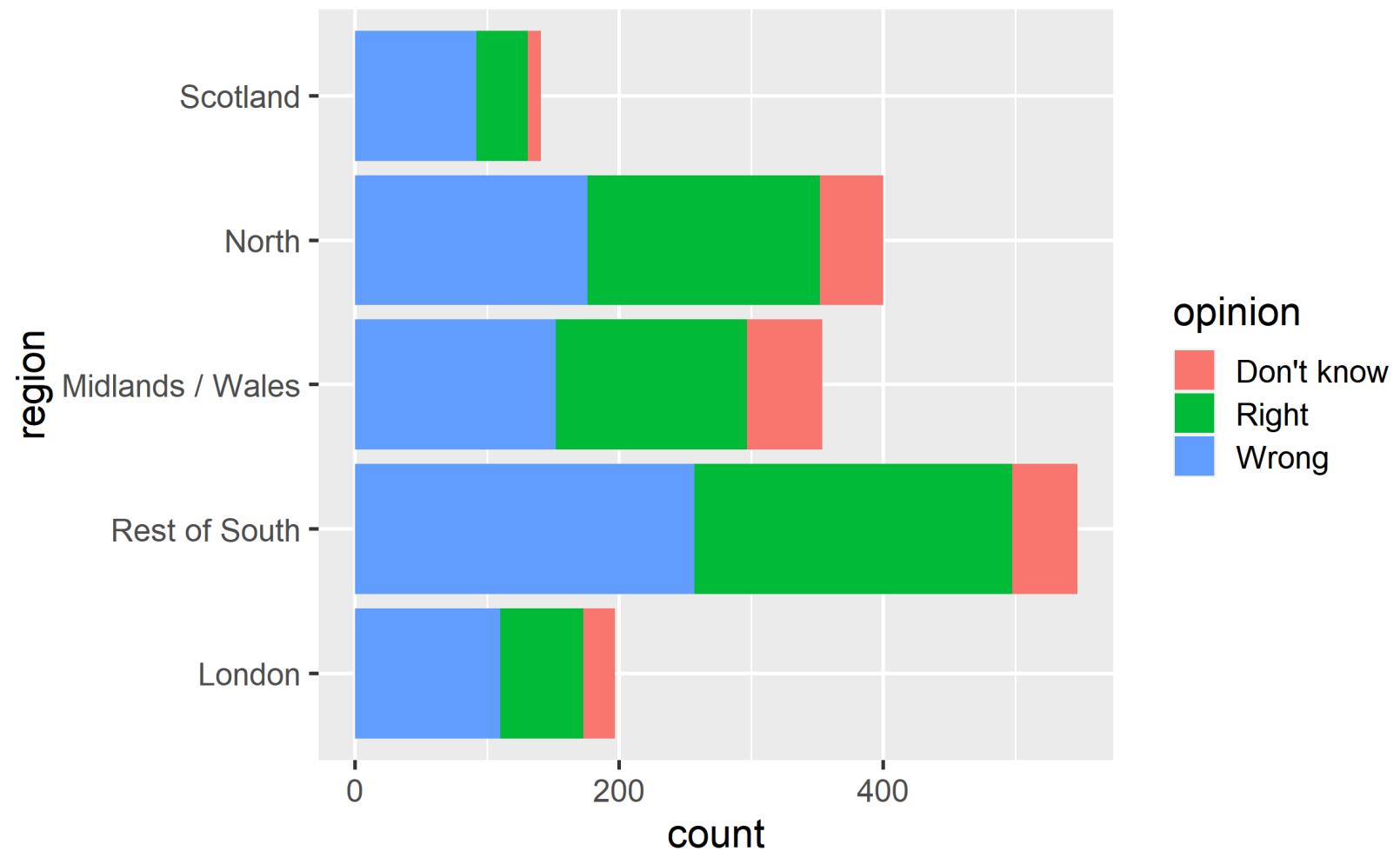


Data Science for Psychologists

Segmented bar plots can be hard to read

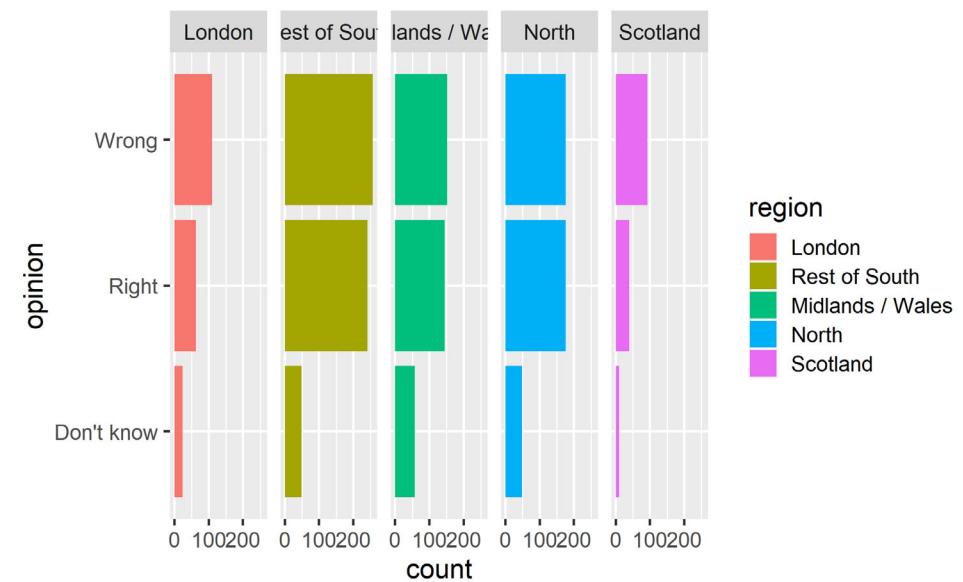
```
ggplot(brexit, aes(y = region, fill = opinion)) +  
  geom_bar()
```

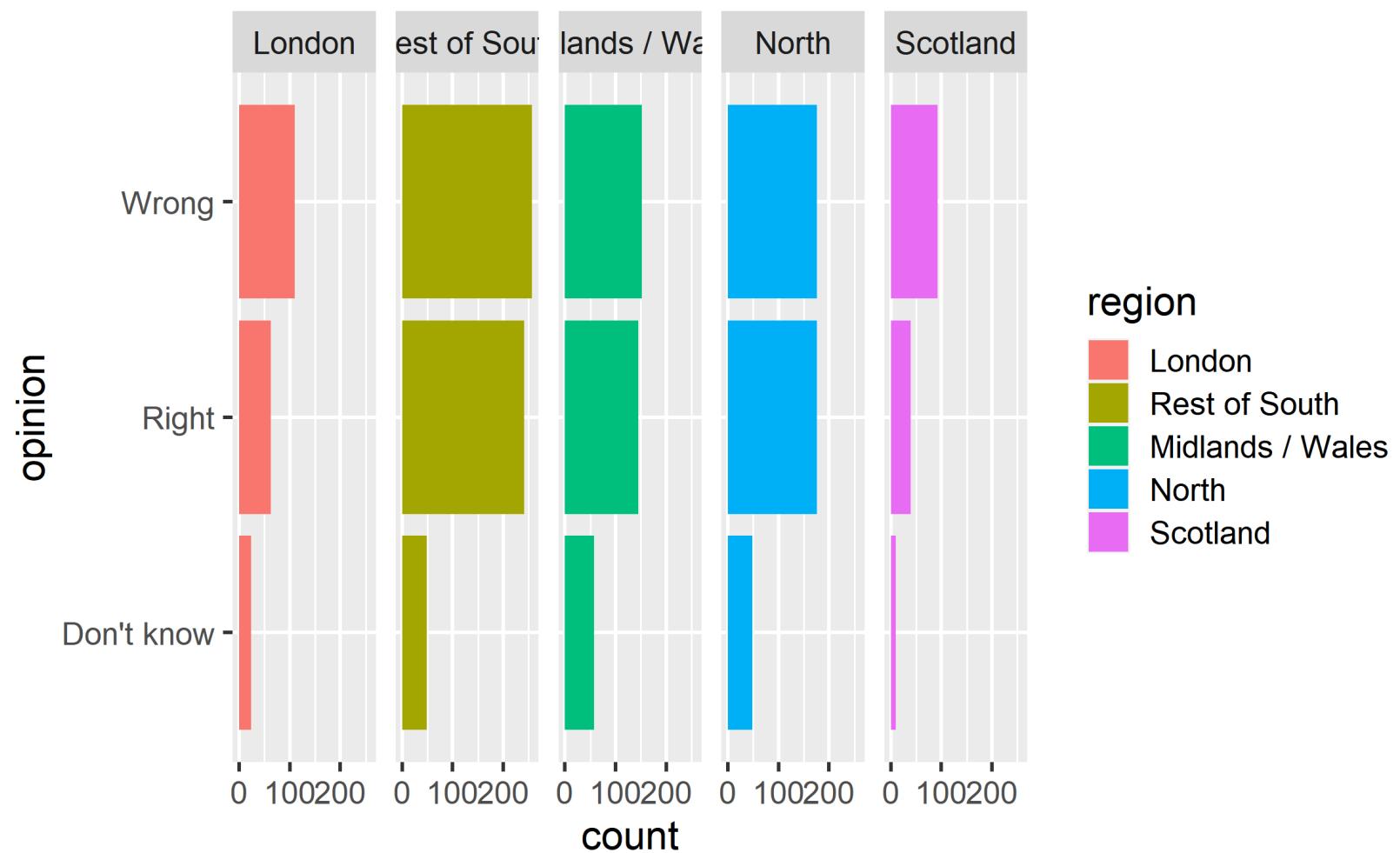




Use facets

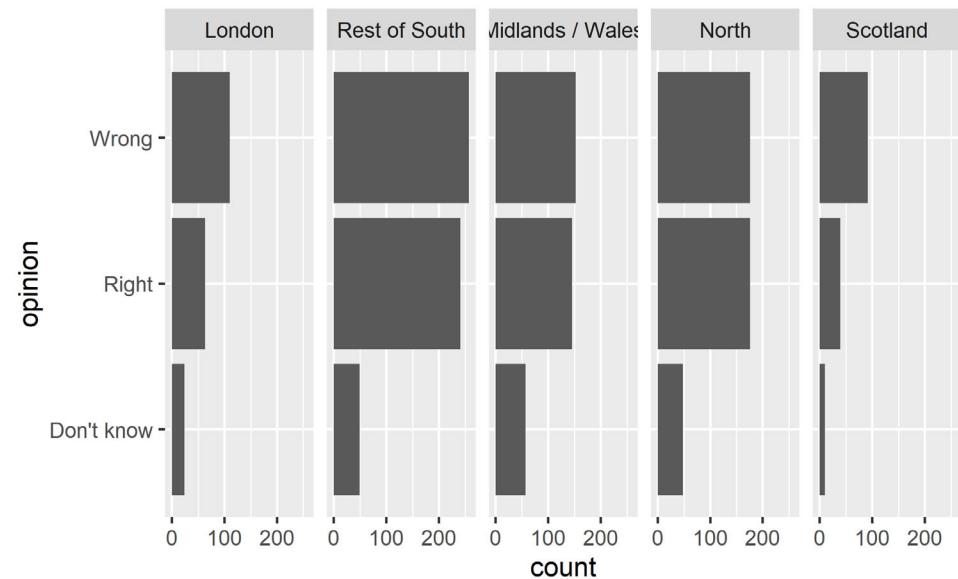
```
ggplot(brexit, aes(y = opinion, fill = region)) +  
  geom_bar() +  
  facet_wrap(~region, nrow = 1)
```

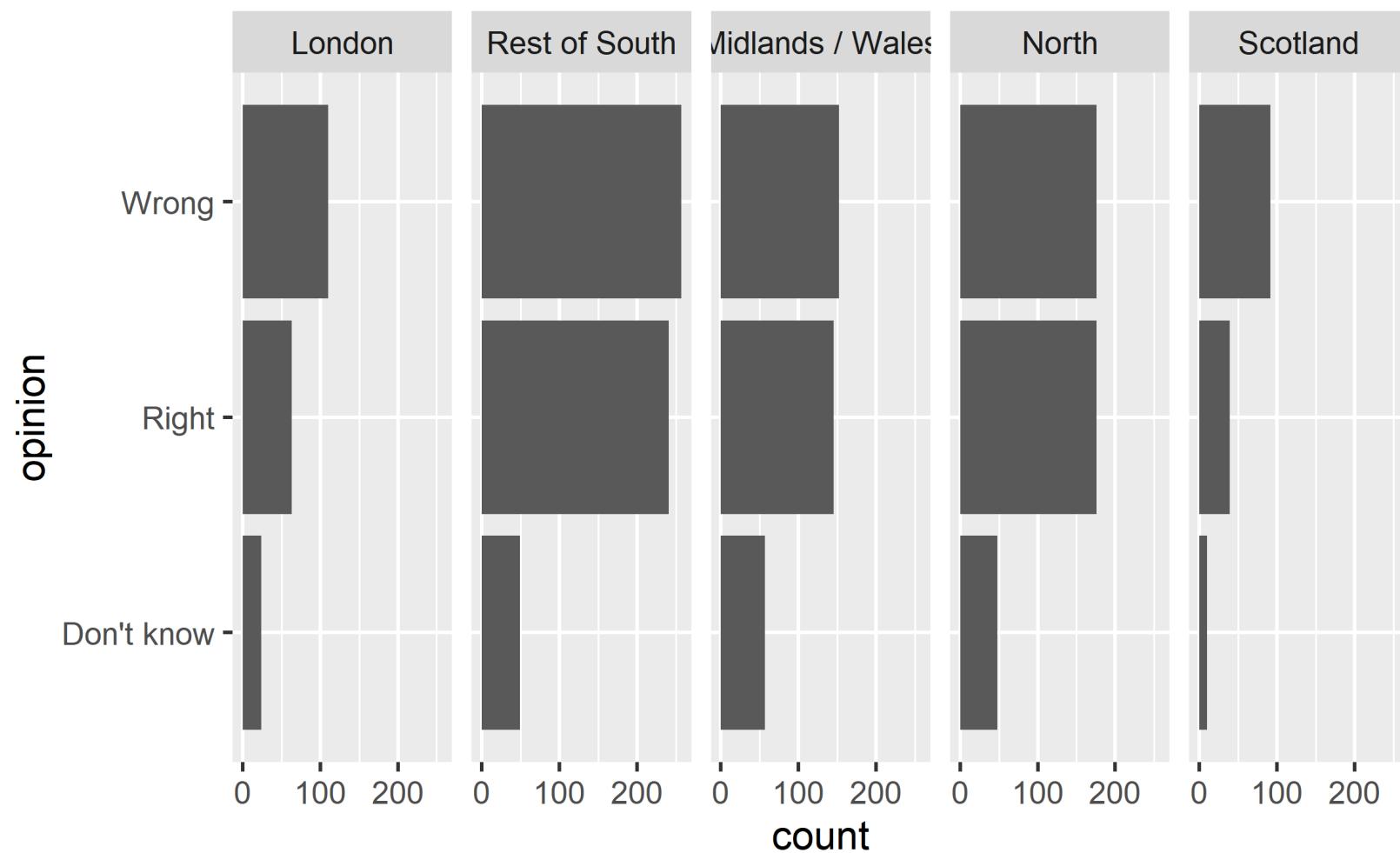




Avoid redundancy?

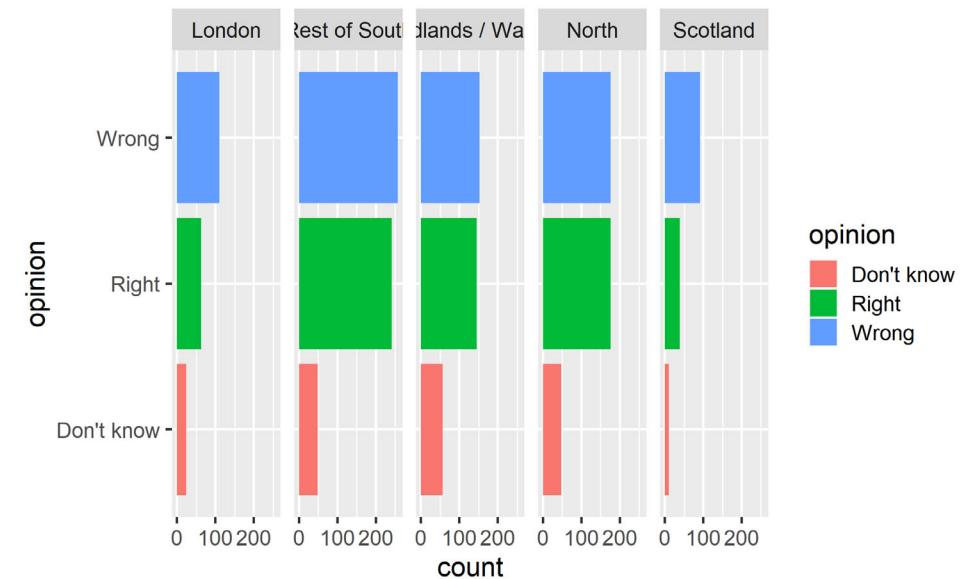
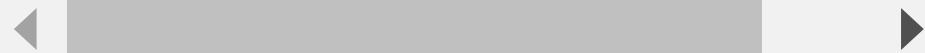
```
ggplot(brexit, aes(y = opinion)) +  
  geom_bar() +  
  facet_wrap(~region, nrow = 1)
```





Redundancy can help tell a story

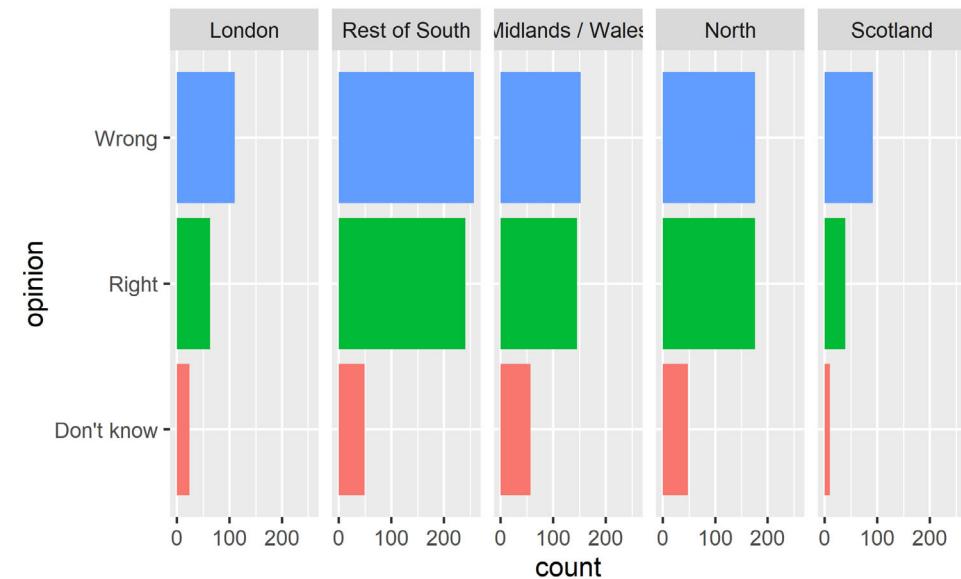
```
ggplot(brexit, aes(y = opinion, fill = opin  
geom_bar() +  
facet_wrap(~region, nrow = 1)
```

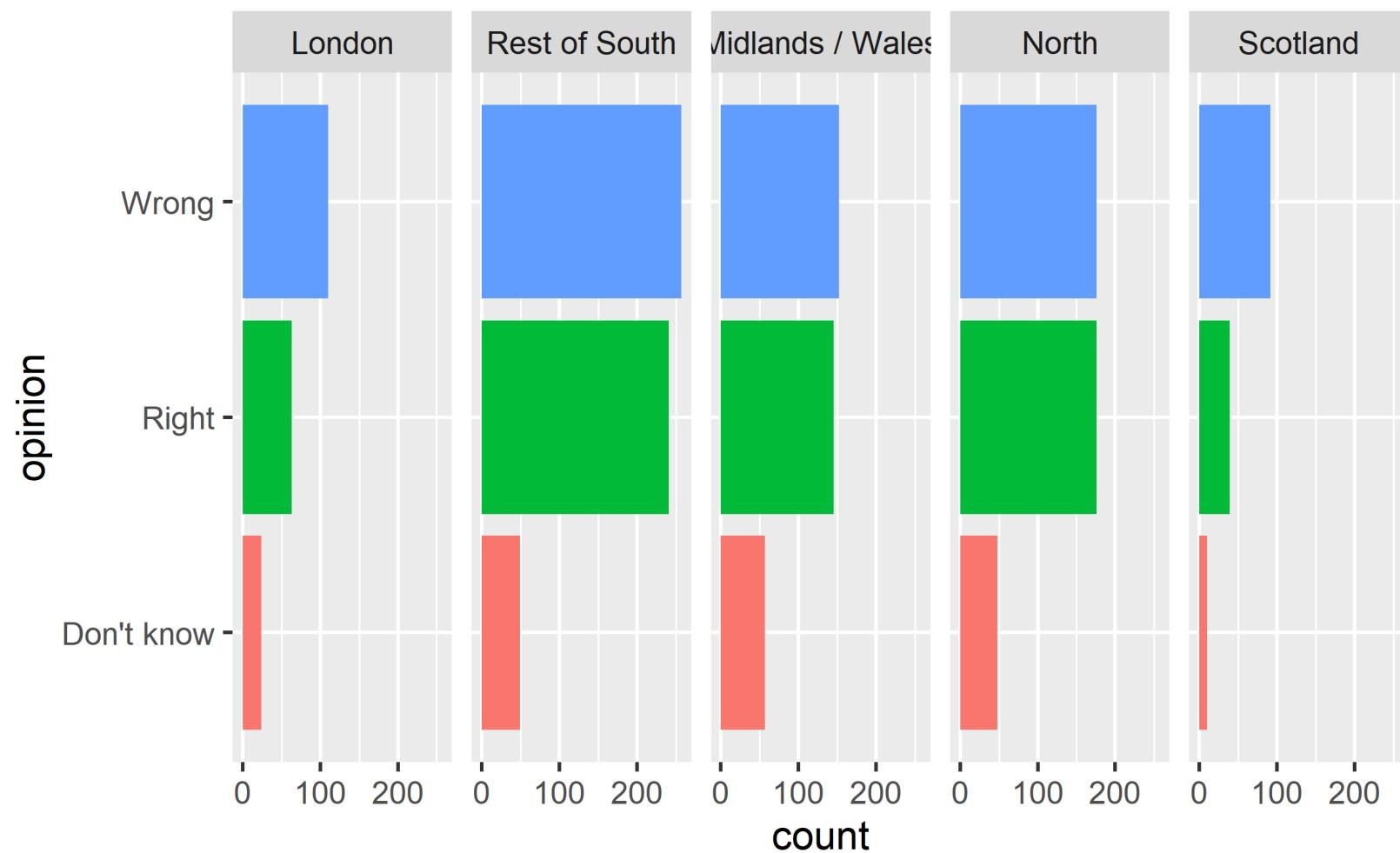




Be selective with redundancy

```
ggplot(brexit, aes(y = opinion, fill = opin  
geom_bar() +  
facet_wrap(~region, nrow = 1) +  
guides(fill = FALSE)
```





Informative labels

```
ggplot(brexit, aes(y = opinion, fill = opin  
geom_bar() +  
facet_wrap(~region, nrow = 1) +  
guides(fill = FALSE) +  
labs(  
  title = "Was Britain right/wrong to vot  
  x = NULL, y = NULL  
)
```

Was Britain right/wrong to vote to leave EU?



Was Britain right/wrong to vote to leave EU?



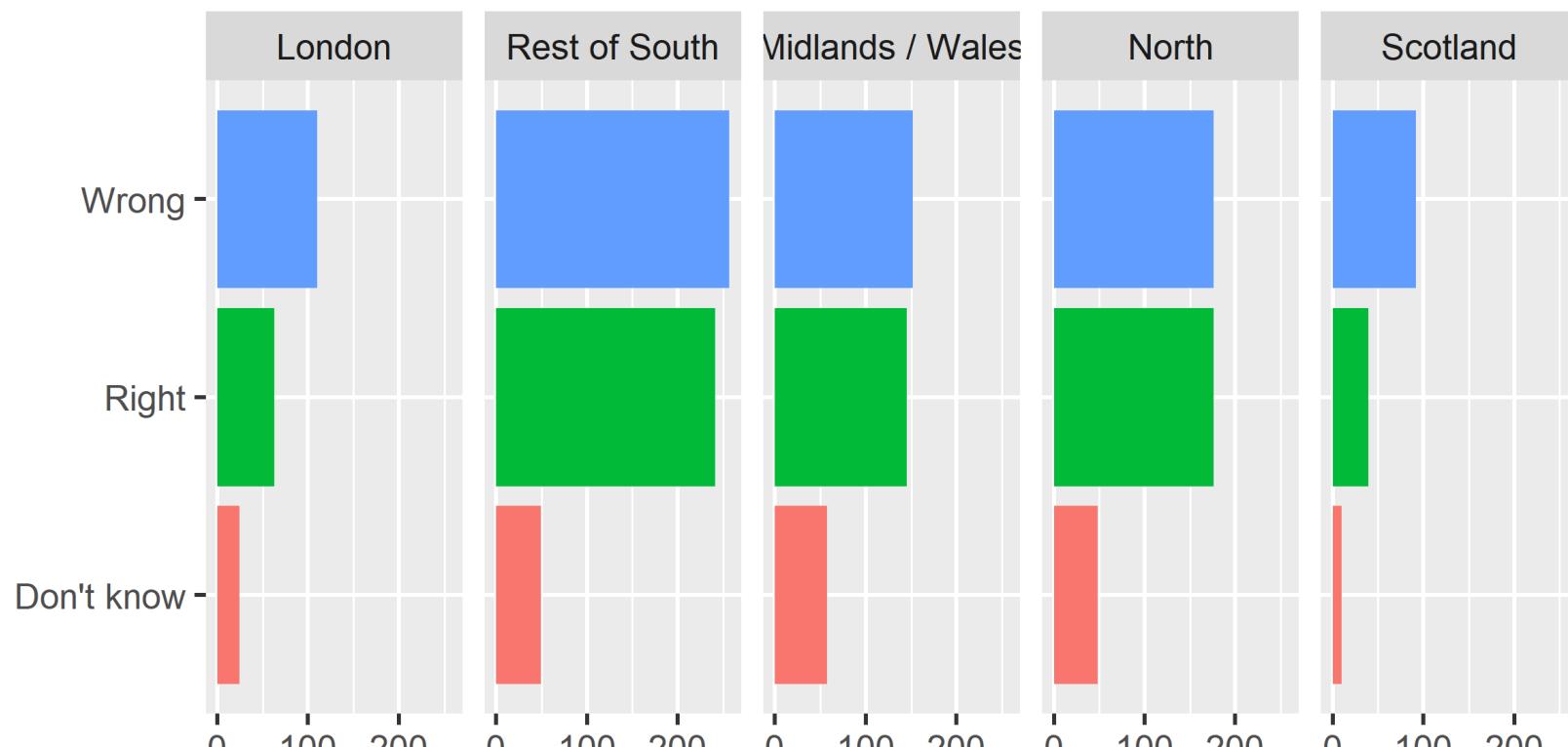
A bit more info

```
ggplot(brexit, aes(y = opinion, fill = opinion)) +  
  geom_bar() +  
  facet_wrap(~region, nrow = 1) +  
  guides(fill = FALSE) +  
  labs(  
    title = "Was Britain right/wrong to vote to leave EU?",  
    subtitle = "YouGov Survey Results, 2-3 September 2019",  
    caption = "Source: https://d25d2506sf94s.cloudfront.net/cumulus\_uploads/document/x0msmgg",  
    x = NULL, y = NULL  
)
```



Was Britain right/wrong to vote to leave EU?

YouGov Survey Results, 2-3 September 2019



https://cumulus_uploads/document/x0msmggx08/YouGov%20-%20Brexit%20and%202019%20election.pdf



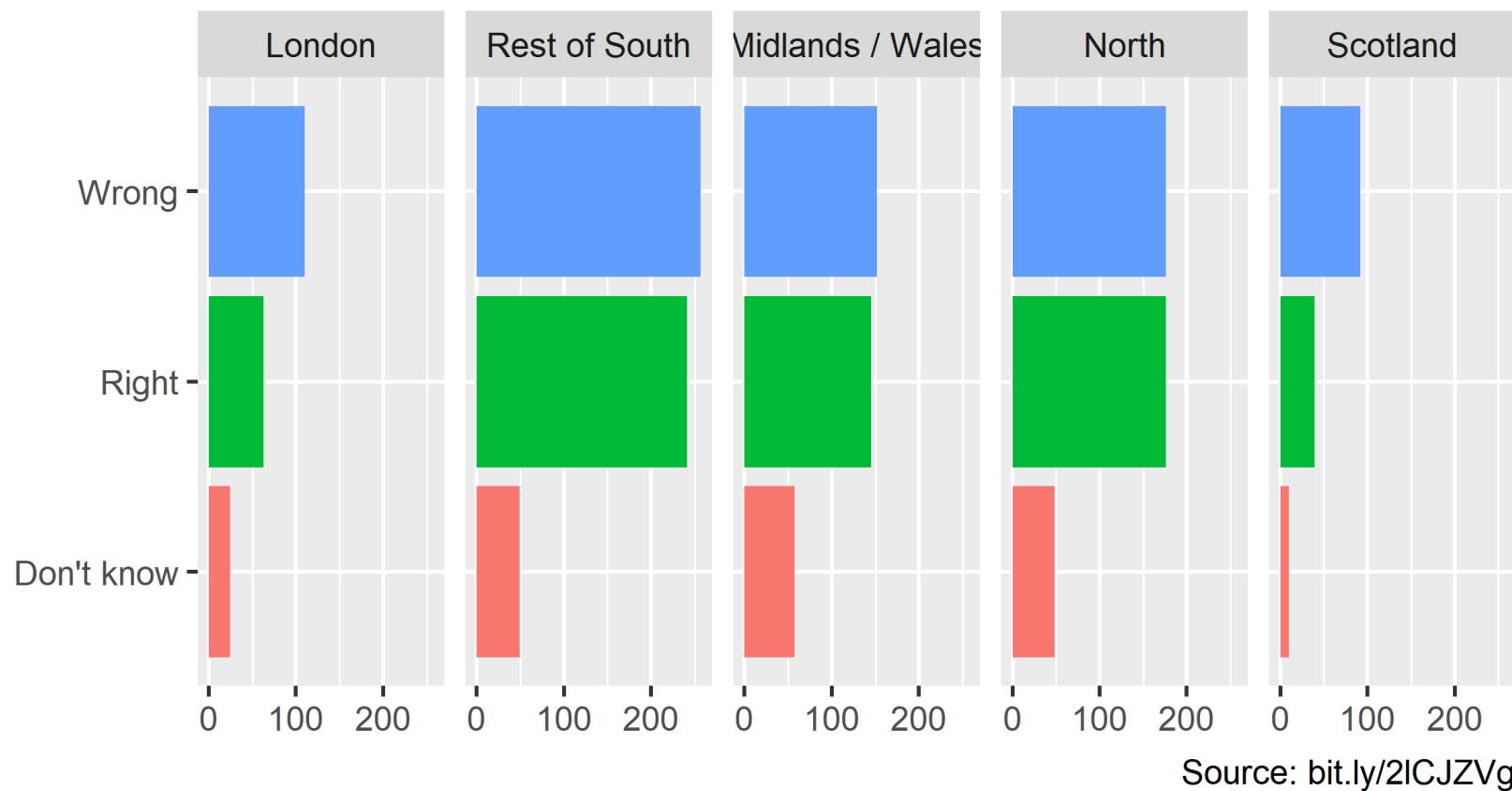
Let's do better

```
ggplot(brexit, aes(y = opinion, fill = opinion)) +  
  geom_bar() +  
  facet_wrap(~region, nrow = 1) +  
  guides(fill = FALSE) +  
  labs(  
    title = "Was Britain right/wrong to vote to leave EU?",  
    subtitle = "YouGov Survey Results, 2-3 September 2019",  
    caption = "Source: bit.ly/2lCJZVg",  
    x = NULL, y = NULL  
)
```



Was Britain right/wrong to vote to leave EU?

YouGov Survey Results, 2-3 September 2019

Source: bit.ly/2ICJZVg

Fix up facet labels

```
ggplot(brexit, aes(y = opinion, fill = opinion)) +  
  geom_bar() +  
  facet_wrap(~region,  
            nrow = 1,  
            labeller = label_wrap_gen(width = 12))  
  ) +  
  guides(fill = FALSE) +  
  labs(  
    title = "Was Britain right/wrong to vote to leave EU?",  
    subtitle = "YouGov Survey Results, 2-3 September 2019",  
    caption = "Source: bit.ly/2lCJZVg",  
    x = NULL, y = NULL  
)
```





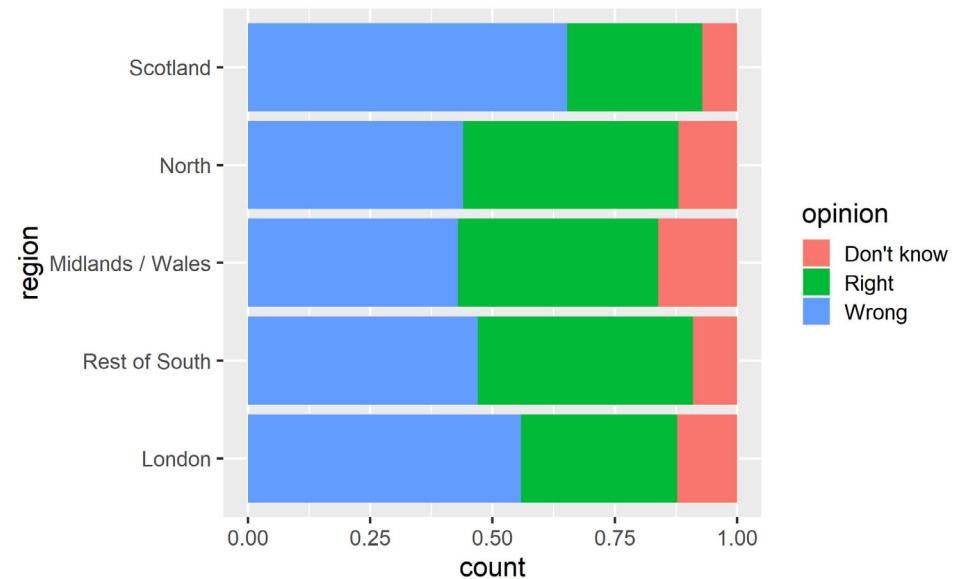
Select meaningful colors

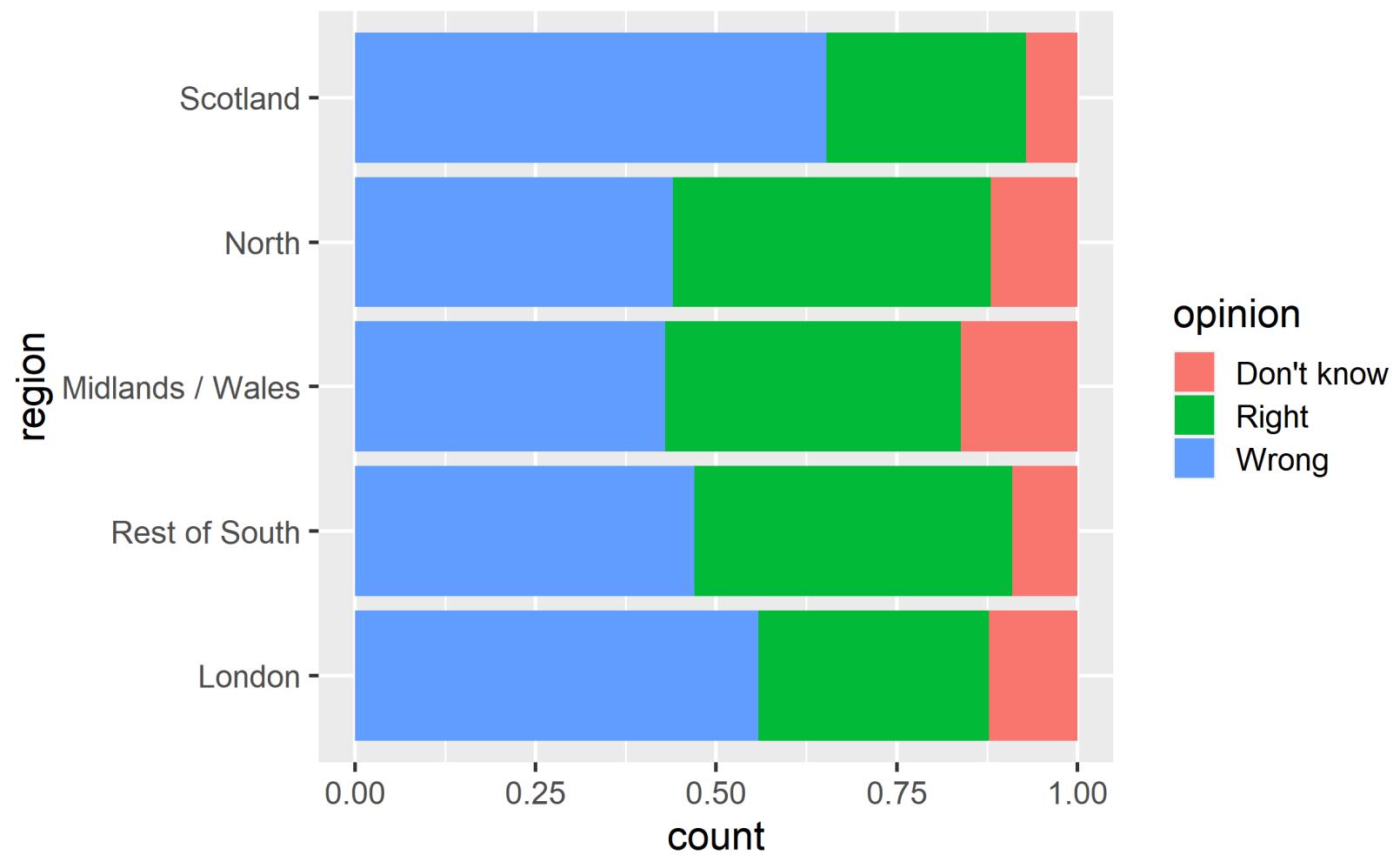


Data Science for Psychologists

Rainbow colors are not always the right choice

```
ggplot(brexit, aes(y= region, fill = opinio  
geom_bar(position = "fill")
```

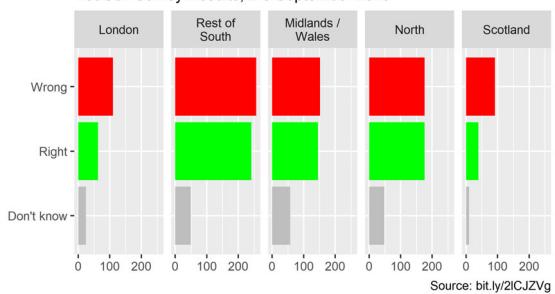




Manually choose colors when needed

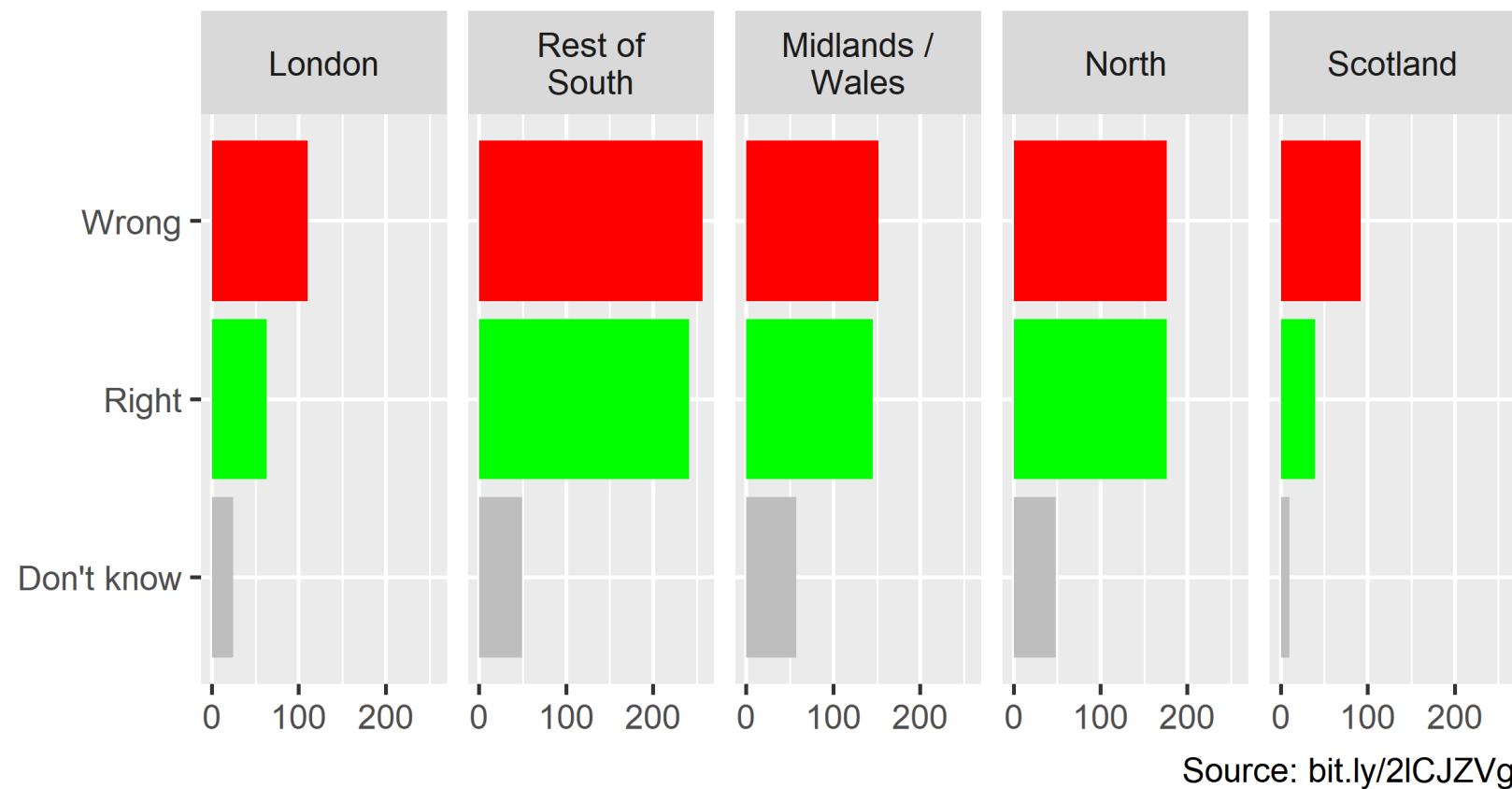
```
ggplot(brexit, aes(y = opinion, fill = opinion)) +
  geom_bar() +
  facet_wrap(~region, nrow = 1, labeller = label_wrap_gen(width = 12)) +
  guides(fill = FALSE) +
  labs(title = "Was Britain right/wrong to vote to leave EU?",
       subtitle = "YouGov Survey Results, 2-3 September 2019",
       caption = "Source: bit.ly/2lCJZVg",
       x = NULL, y = NULL) +
  scale_fill_manual(values = c(
    "Wrong" = "red",
    "Right" = "green",
    "Don't know" = "gray"
))
```

Was Britain right/wrong to vote to leave EU?
YouGov Survey Results, 2-3 September 2019



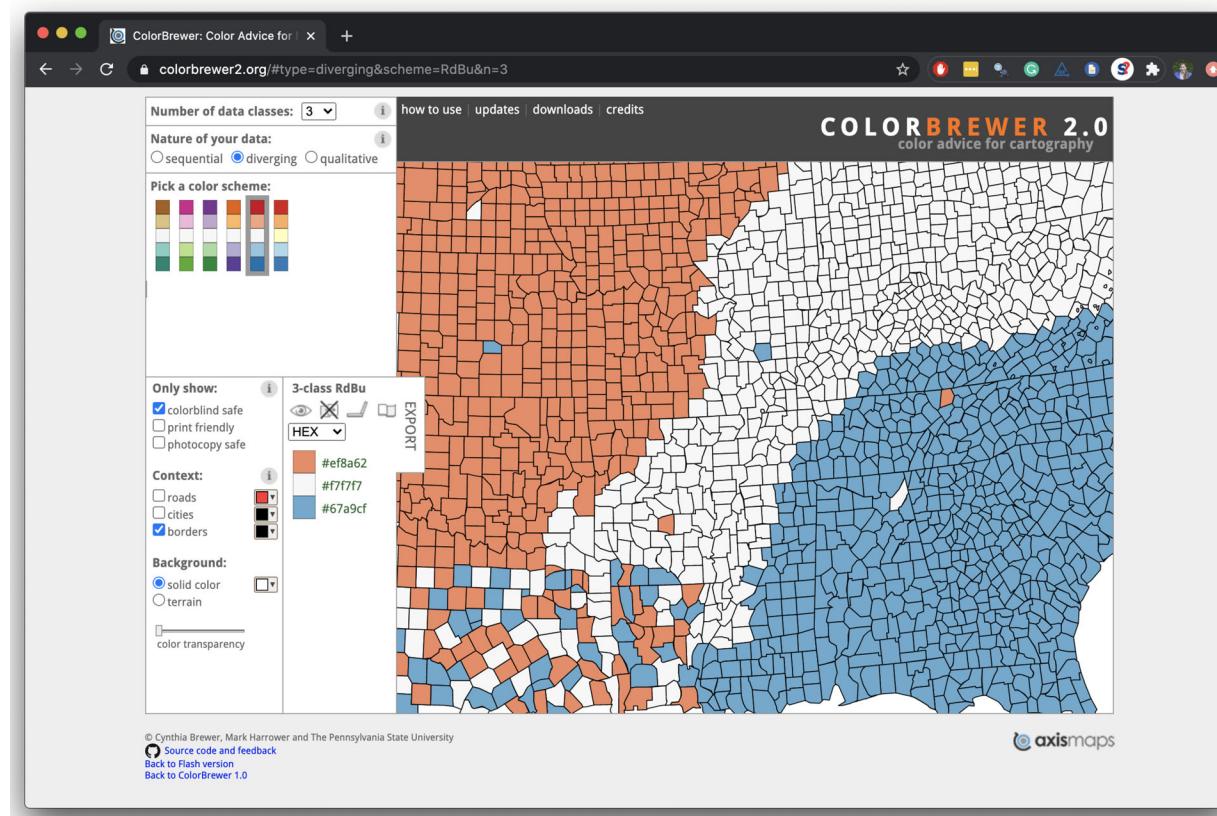
Was Britain right/wrong to vote to leave EU?

YouGov Survey Results, 2-3 September 2019

Source: bit.ly/2ICJZVg

Choosing better colors

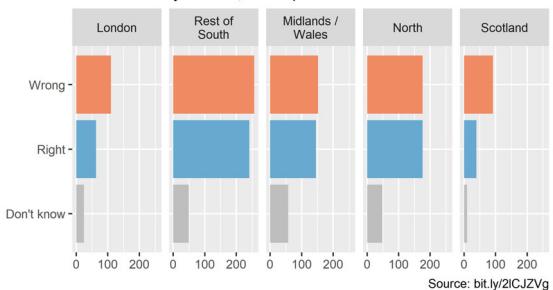
colorbrewer2.org



Use better colors

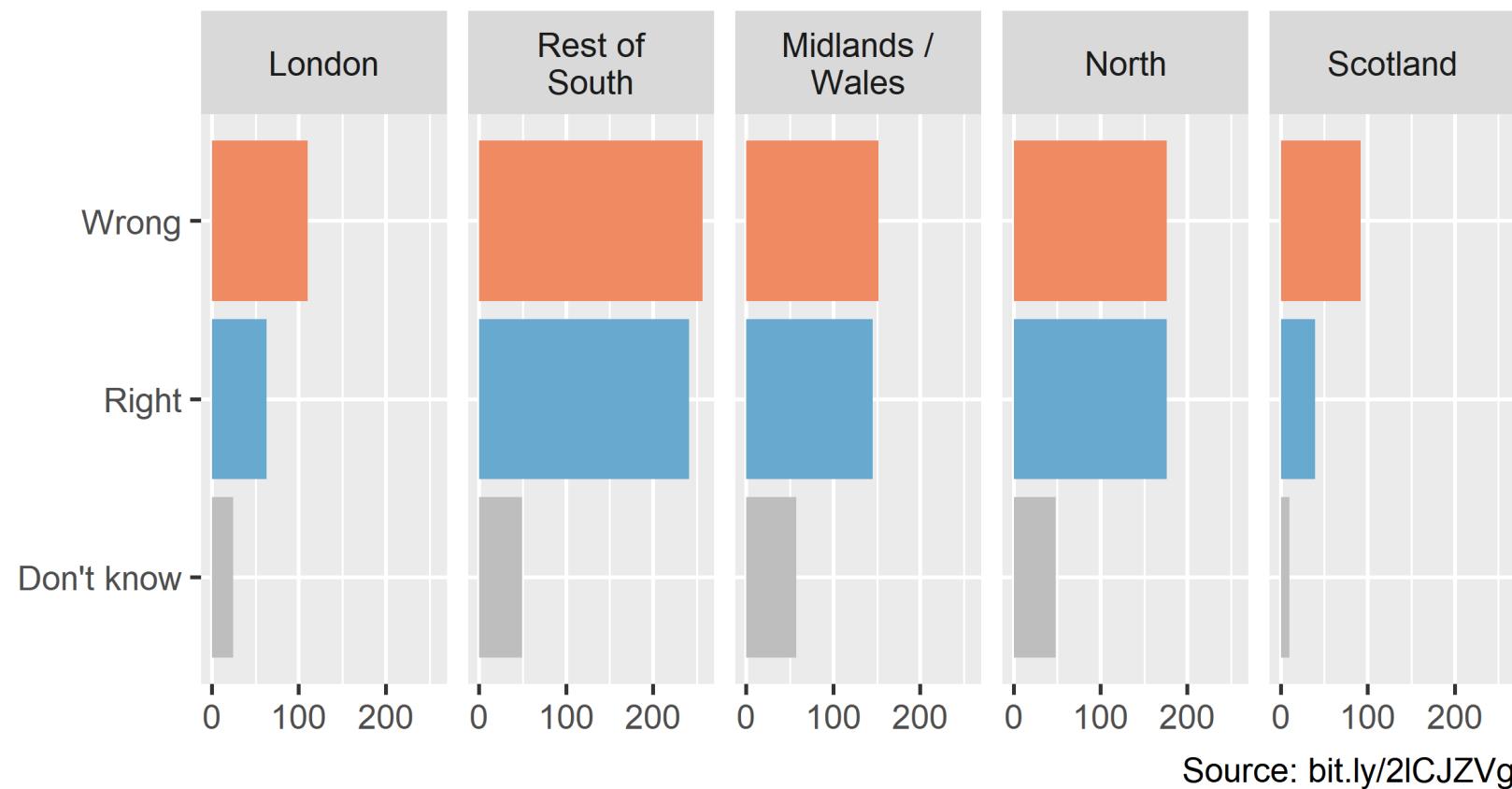
```
ggplot(brexit, aes(y = opinion, fill = opinion)) +
  geom_bar() +
  facet_wrap(~region, nrow = 1, labeller = label_wrap_gen(width = 12)) +
  guides(fill = FALSE) +
  labs(title = "Was Britain right/wrong to vote to leave EU?",
       subtitle = "YouGov Survey Results, 2-3 September 2019",
       caption = "Source: bit.ly/2lCJZVg",
       x = NULL, y = NULL) +
  scale_fill_manual(values = c(
    "Wrong" = "#ef8a62",
    "Right" = "#67a9cf",
    "Don't know" = "gray"
))
```

Was Britain right/wrong to vote to leave EU?
YouGov Survey Results, 2-3 September 2019



Was Britain right/wrong to vote to leave EU?

YouGov Survey Results, 2-3 September 2019

Source: bit.ly/2ICJZVg

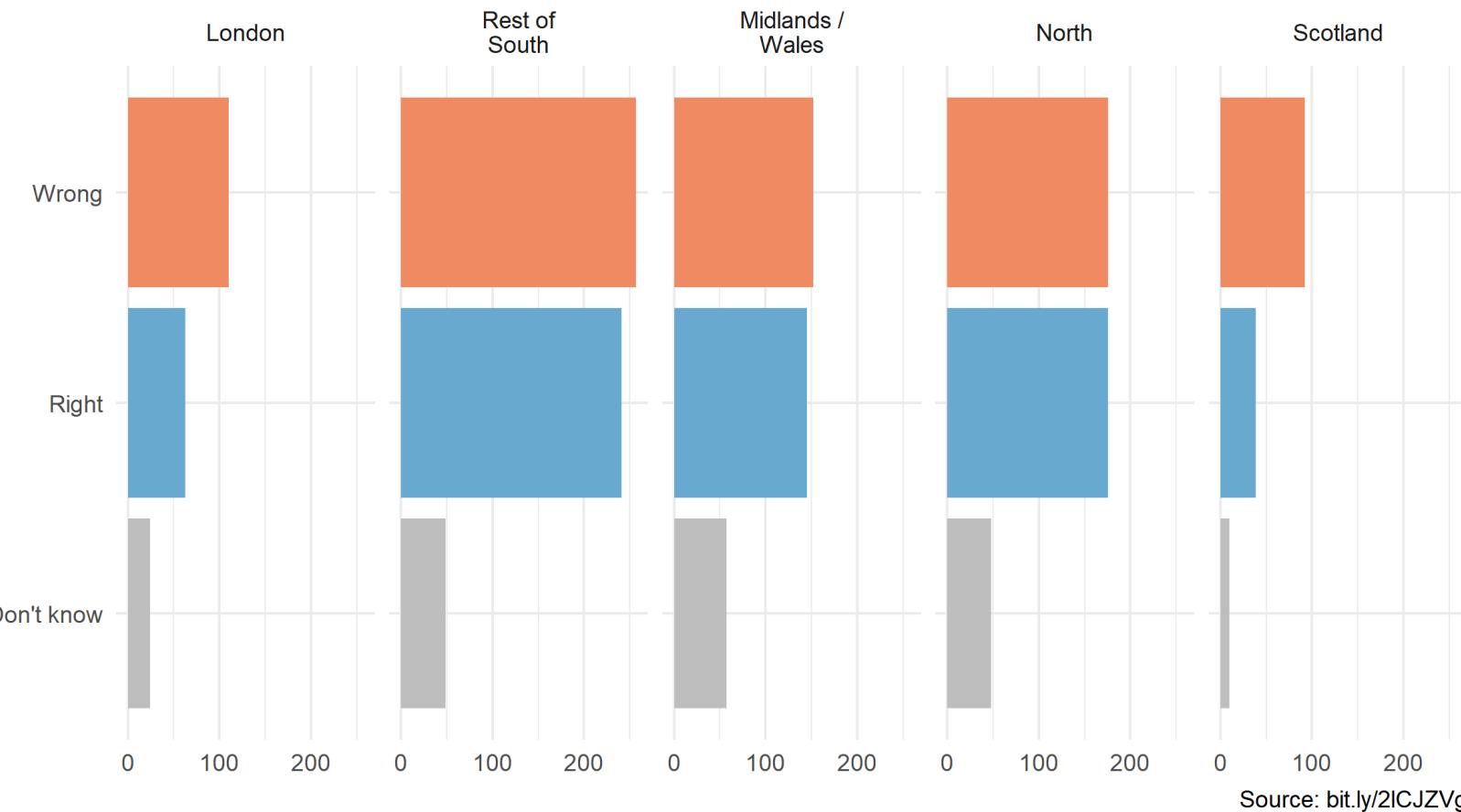
Select theme

```
ggplot(brexit, aes(y = opinion, fill = opinion)) +  
  geom_bar() +  
  facet_wrap(~region, nrow = 1, labeller = label_wrap_gen(width = 12)) +  
  guides(fill = FALSE) +  
  labs(title = "Was Britain right/wrong to vote to leave EU?",  
       subtitle = "YouGov Survey Results, 2-3 September 2019",  
       caption = "Source: bit.ly/2lCJZVg",  
       x = NULL, y = NULL) +  
  scale_fill_manual(values = c("Wrong" = "#ef8a62",  
                             "Right" = "#67a9cf",  
                             "Don't know" = "gray")) +  
  theme_minimal()
```



Was Britain right/wrong to vote to leave EU?

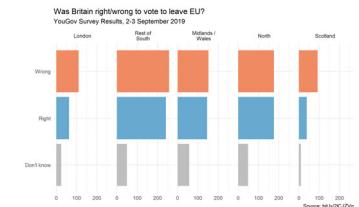
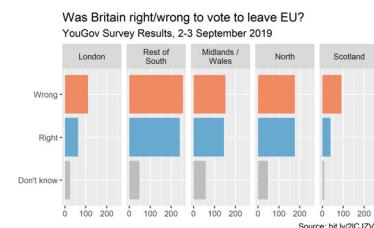
YouGov Survey Results, 2-3 September 2019



Source: bit.ly/2ICJZVg



Spot the Difference



Your turn!

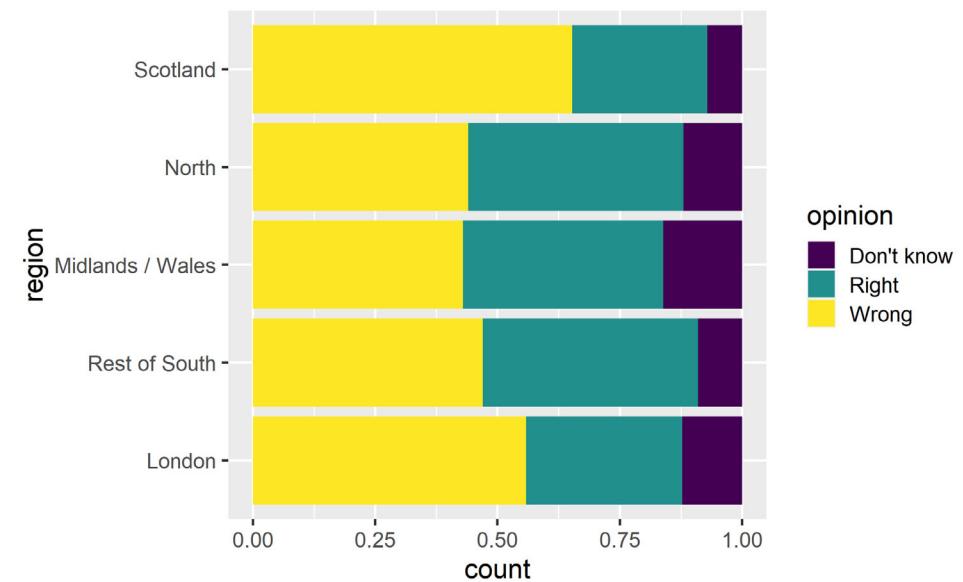
your turn!

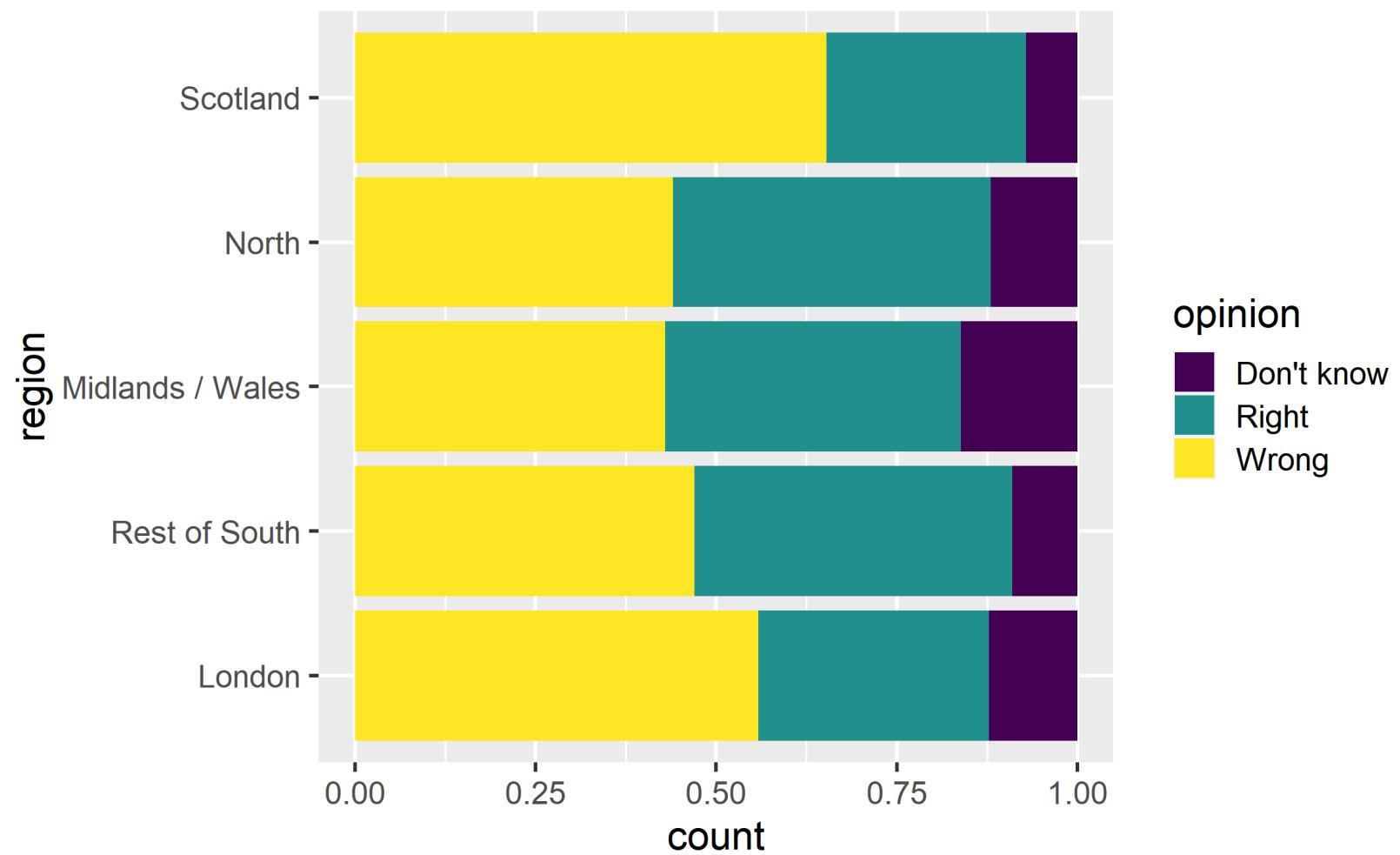
- + AE 07 - Brexit + Telling stories with `dataviz>brexit.Rmd`.
- + Change the visualization in three different ways to tell slightly different stories with it each time.



Viridis scale works well with ordinal data

```
ggplot(brexit, aes(y = region, fill = opinion)) +  
  geom_bar(position = "fill") +  
  scale_fill_viridis_d()
```

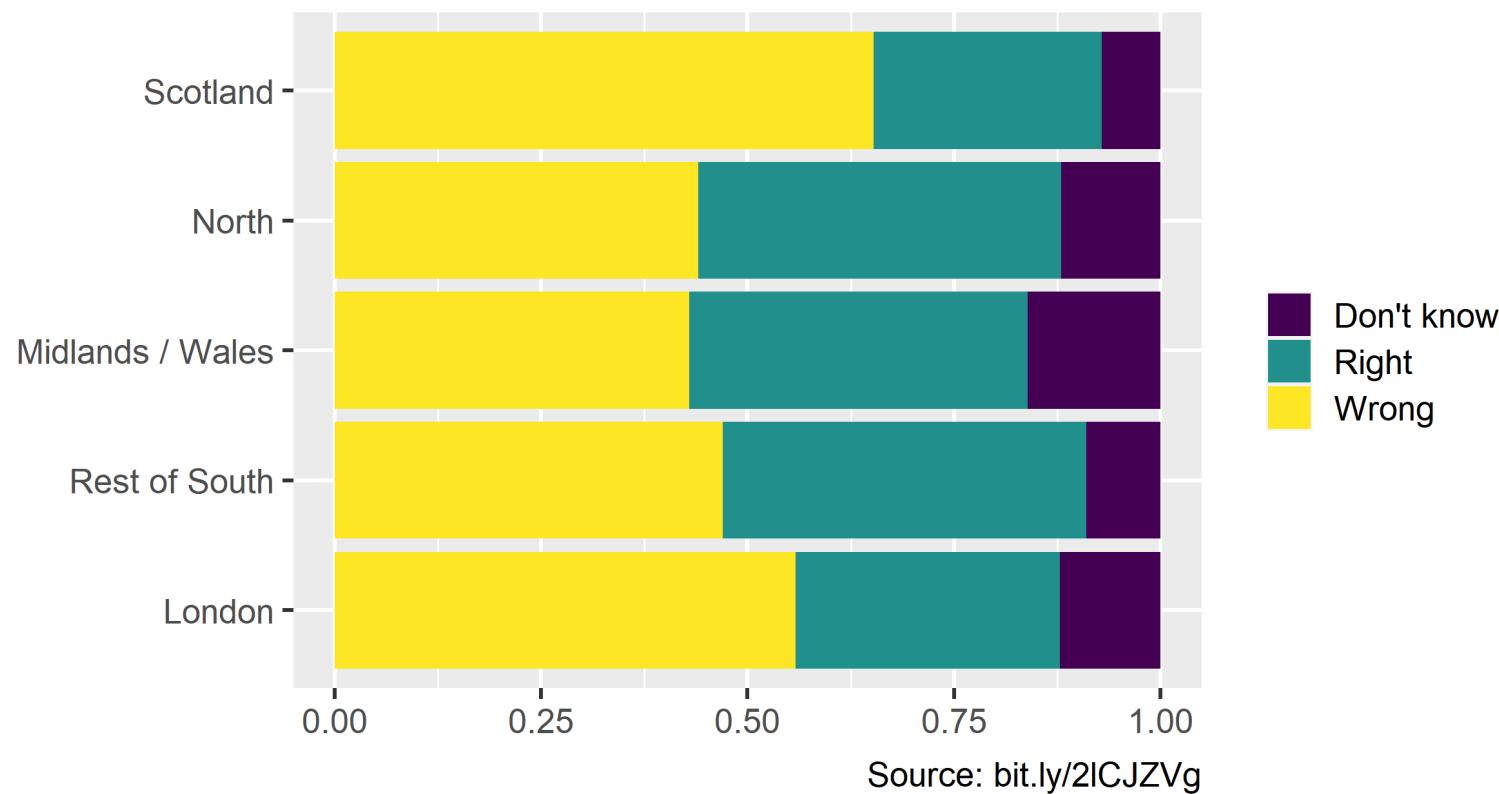




Clean up labels (Again)

Was Britain right/wrong to vote to leave EU?

YouGov Survey Results, 2-3 September 2019



Sources

- + Mine Å‡etinkaya-Rundel's Data Science in a Box ([link](#))



Wrapping Up...



Data Science for Psychologists

"Communicating data science results effectively

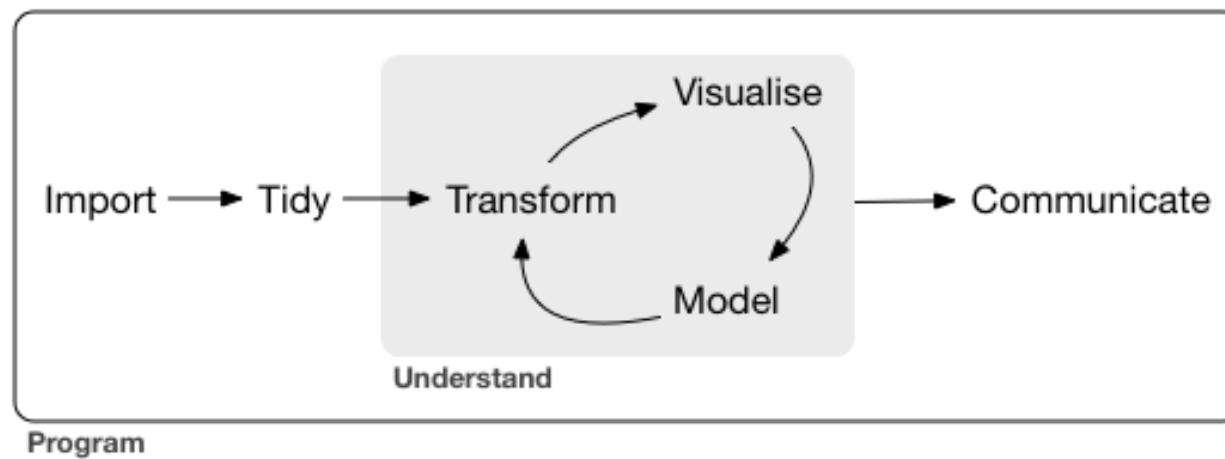


Communicating data science results effectively



Five core activities of data science

1. Stating and refining the question
2. Exploring the data
3. Building formal statistical models
4. Interpreting the results
5. Communicating the results



Roger D. Peng and Elizabeth Matsui. "The Art of Data Science." A Guide for Anyone Who Works with Data. Skybrude Consulting, LLC (2015).



Stating and refining the question



Six types of questions

1. **Descriptive:** summarize a characteristic of a set of data
2. **Exploratory:** analyze to see if there are patterns, trends, or relationships between variables (hypothesis generating)
3. **Inferential:** analyze patterns, trends, or relationships in representative data from a population
4. **Predictive:** make predictions for individuals or groups of individuals
5. **Causal:** whether changing one factor will change another factor, on average, in a population
6. **Mechanistic:** explore "how" as opposed to whether

Jeffery T. Leek and Roger D. Peng. "What is the question?." Science 347.6228 (2015): 1314-1315.



Ex: COVID-19 and Vitamin D

1. **Descriptive:** frequency of hospitalizations due to COVID-19 in a set of data collected from a group of individuals
2. **Exploratory:** examine relationships between a range of dietary factors and COVID-19 hospitalizations
3. **Inferential:** examine whether any relationship between taking Vitamin D supplements and COVID-19 hospitalizations found in the sample hold for the population at large
4. **Predictive:** what types of people will take Vitamin D supplements during the next year
5. **Causal:** whether people with COVID-19 who were randomly assigned to take Vitamin D supplements or those who were not are hospitalized
6. **Mechanistic:** how increased vitamin D intake leads to a reduction in the number of viral illnesses



Questions to data science problems

- + Do you have appropriate data to answer your question?
- + Do you have information on confounding variables?
- + Was the data you're working with collected in a way that introduces bias?

+ Suppose I want to estimate the average number of children in households in Winston-Salem.

+ I conduct a survey at an elementary school in Winston-Salem and ask those students:

- + how many children, including themselves, live in their house.

+ Then, I take the average of the responses.

+ Is this a biased or an unbiased estimate of the number of children in households in Winston-Salem?

+ If biased, will the value be an overestimate or underestimate?



Exploratory data analysis



Checklist

- + Formulate your question
- + Read in your data
- + Check the dimensions
- + Look at the top and the bottom of your data
- + Validate with at least one external data source
- + Make a plot
- + Try the easy solution first



Formulate your question

- + Consider scope:
 - + Are air pollution levels higher on the east coast than on the west coast?
 - + Are hourly ozone levels on average higher in New York City than they are in Los Angeles?
 - + Do counties in the eastern United States have higher ozone levels than counties in the western United States?
- + Most importantly: "Do I have the right data to answer this question?"



Read in your data

- + Place your data in a folder called data
- + Read it into R with `read_csv()` or friends (`read_delim()`, `read_excel()`, etc.)

```
library(readxl)
fav_food <- read_excel("data/favorite-food.xlsx")
fav_food
```

```
## # A tibble: 5 × 4
##   `Student ID` `Full Name`    favorite.food    mealPlan
##       <dbl> <chr>          <chr>            <chr>
## 1           1 Sunil Huffmann Strawberry yoghurt Lunch only
## 2           2 Barclay Lynn    French fries      Lunch only
## 3           3 Jayendra Lyne  Peaches          Breakfast and...
## 4           4 Leon Rossini   Anchovies       Lunch only
## 5           5 Chidiegwu Dunkel Pizza        Breakfast and...
```



clean_names()

If the variable names are malformatted, use `janitor::clean_names()`

```
library(janitor)  
fav_food %>% clean_names()
```

```
## # A tibble: 5 × 4  
##   student_id full_name      favorite_food    meal_plan  
##   <dbl> <chr>          <chr>           <chr>  
## 1 1     Sunil Huffmann  Strawberry yoghurt Lunch only  
## 2 2     Barclay Lynn    French fries       Lunch only  
## 3 3     Jayendra Lyne   Peaches          Breakfast and l...  
## 4 4     Leon Rossini    Anchovies        Lunch only  
## 5 5     Chidiegwu Dunkel Pizza          Breakfast and l...
```



Wrapping Up...



Data Science for Psychologists

Case study: NYC Squirrels!



NYC Squirrels!

- + The **Squirrel Census** is a multimedia science, design, and storytelling project focusing on the Eastern gray (*Sciurus carolinensis*). They count squirrels and present their findings to the public.
- + This table contains squirrel data for each of the 3,023 sightings, including location coordinates, age, primary and secondary fur color, elevation, activities, communications, and interactions between squirrels and with humans.

```
#install_github("mine-cetinkaya-rundel/nycsquirrels18")
library(nycsquirrels18)
```



Locate the codebook

mine-cetinkaya-rundel.github.io/nycsquirrels18/reference/squirrels.html

Check the dimensions

```
dim(squirrels)
```

```
## [1] 3023    38
```



Data Science for Psychologists

Look at the top...

```
squirrels %>% head()
```

```
## # A tibble: 6 × 38
##   long    lat unique_squirrel_id hectare NS   EW   shift
##   <dbl> <dbl> <chr>           <chr>  <chr> <chr> <chr>
## 1 -74.0  40.8 13A-PM-1014-04   13A    13    A    PM
## 2 -74.0  40.8 15F-PM-1010-06   15F    15    F    PM
## 3 -74.0  40.8 19C-PM-1018-02   19C    19    C    PM
## 4 -74.0  40.8 21B-AM-1019-04   21B    21    B    AM
## 5 -74.0  40.8 23A-AM-1018-02   23A    23    A    AM
## 6 -74.0  40.8 38H-PM-1012-01   38H    38    H    PM
## # i 31 more variables: date <date>,
## #   hectare_squirrel_number <dbl>, age <chr>,
## #   primary_fur_color <chr>, highlight_fur_color <chr>,
## #   combination_of_primary_and_highlight_color <chr>,
## #   color_notes <chr>, location <chr>,
## #   above_ground_sighter_measurement <chr>,
## #   specific_location <chr>, running <lgl>, chasing <lgl>, ...
```



...and the bottom

```
squirrels %>% tail()
```

```
## # A tibble: 6 × 38
##   long    lat unique_squirrel_id hectare NS   EW   shift
##   <dbl> <dbl> <chr>           <chr>  <chr> <chr> <chr>
## 1 -74.0  40.8  6D-PM-1020-01    06D    06    D    PM
## 2 -74.0  40.8  21H-PM-1018-01   21H    21    H    PM
## 3 -74.0  40.8  31D-PM-1006-02   31D    31    D    PM
## 4 -74.0  40.8  37B-AM-1018-04   37B    37    B    AM
## 5 -74.0  40.8  21C-PM-1006-01   21C    21    C    PM
## 6 -74.0  40.8  7G-PM-1018-04    07G    07    G    PM
## # i 31 more variables: date <date>,
## #   hectare_squirrel_number <dbl>, age <chr>,
## #   primary_fur_color <chr>, highlight_fur_color <chr>,
## #   combination_of_primary_and_highlight_color <chr>,
## #   color_notes <chr>, location <chr>,
## #   above_ground_sighter_measurement <chr>,
## #   specific_location <chr>, running <lgl>, chasing <lgl>, ...
```



Validate with at least one external data source

```
## # A tibble: 3,023 × 2
##   long     lat
##   <dbl> <dbl>
## 1 -74.0  40.8
## 2 -74.0  40.8
## 3 -74.0  40.8
## 4 -74.0  40.8
## 5 -74.0  40.8
## 6 -74.0  40.8
## 7 -74.0  40.8
## 8 -74.0  40.8
## 9 -74.0  40.8
## 10 -74.0  40.8
## 11 -74.0  40.8
## 12 -74.0  40.8
## 13 -74.0  40.8
## 14 -74.0  40.8
## 15 -74.0  40.8
## # i 3,008 more rows
```

Central Park / Coordinates

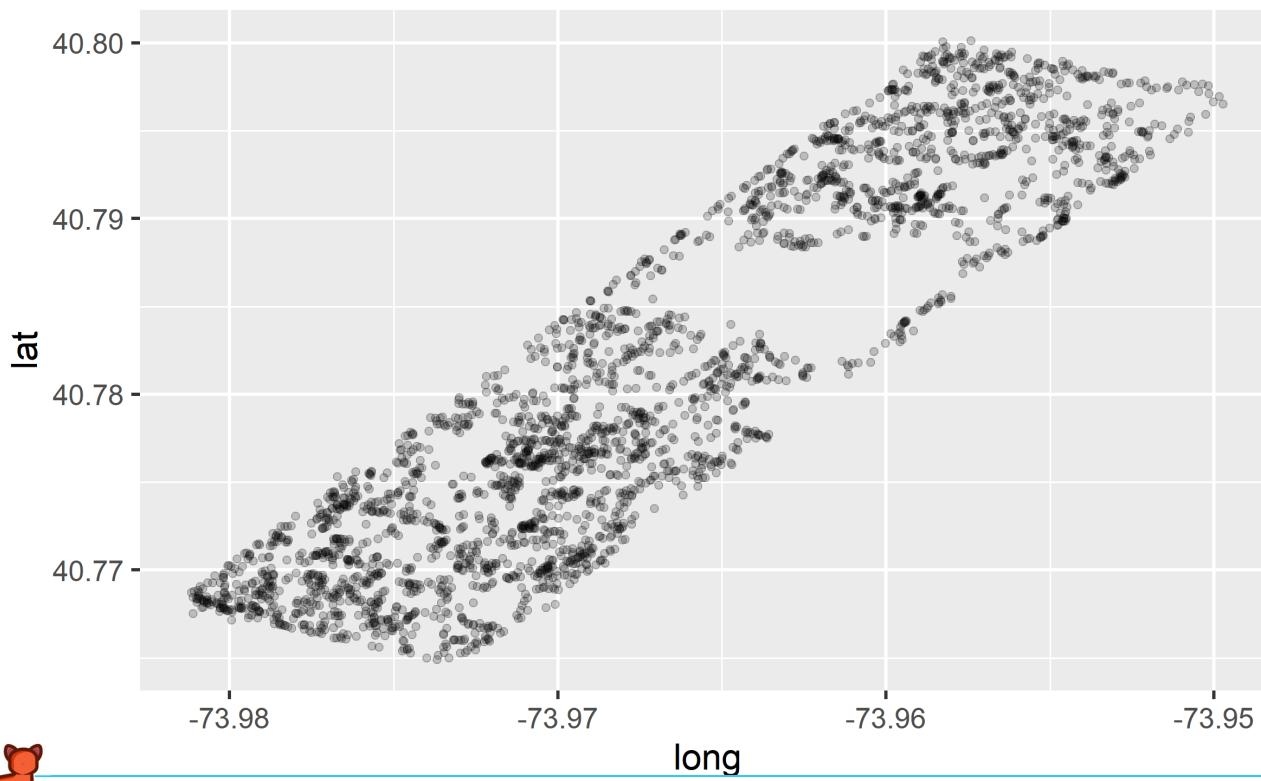
40.7829° N, 73.9654° W



Data Science for Psychologists

Make a plot

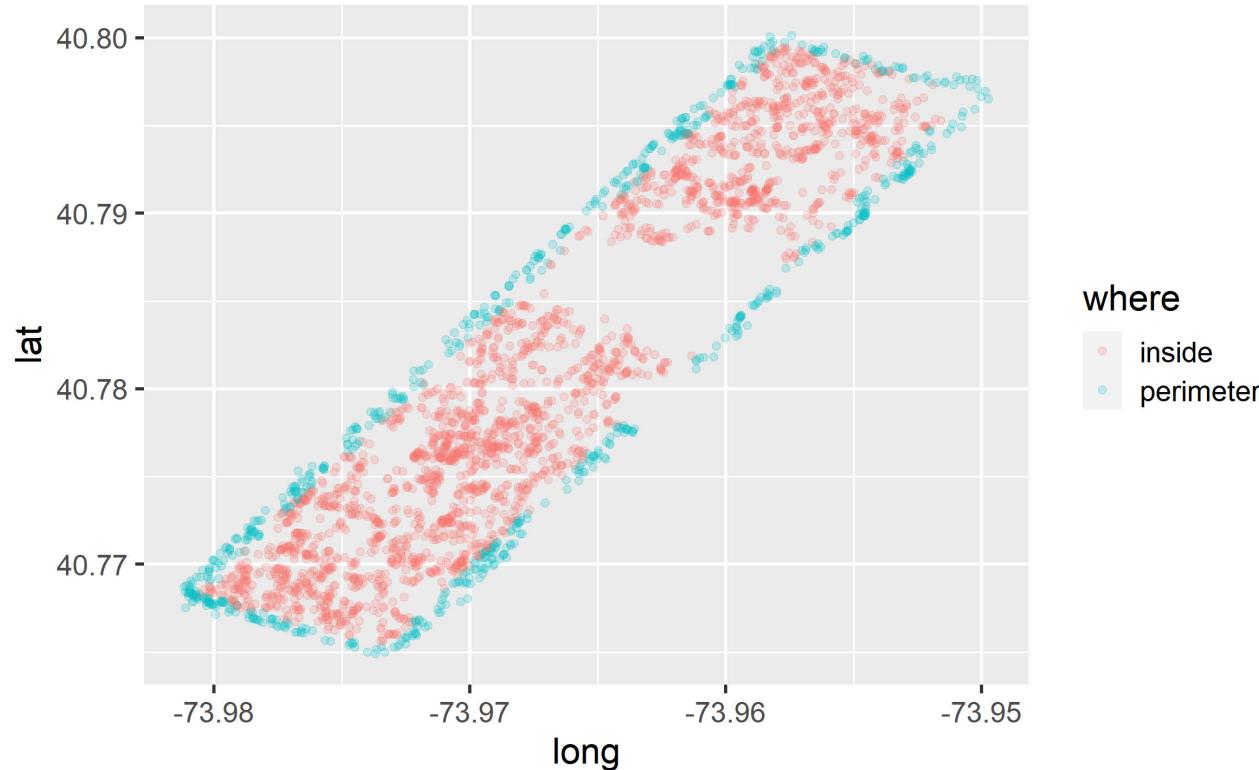
```
ggplot(squirrels, aes(x = long, y = lat)) +  
  geom_point(alpha = 0.2)
```



Hypothesis: Squirrel sightings will have a higher density on the perimeter than inside the park.



Try the easy solution first



```
squirrels <- squirrels %>%  
  separate(hectare,  
    into = c("NS", "EW"),  
    sep = 2, remove = FALSE) %>%  
  mutate(where =  
    if_else(NS %in% c("01", "42") |  
      EW %in% c("A", "I"),  
      "perimeter",  
      "inside"))  
  
ggplot(squirrels, aes(x = long, y = lat,  
                      color = where)) +  
  geom_point(alpha = 0.2)
```



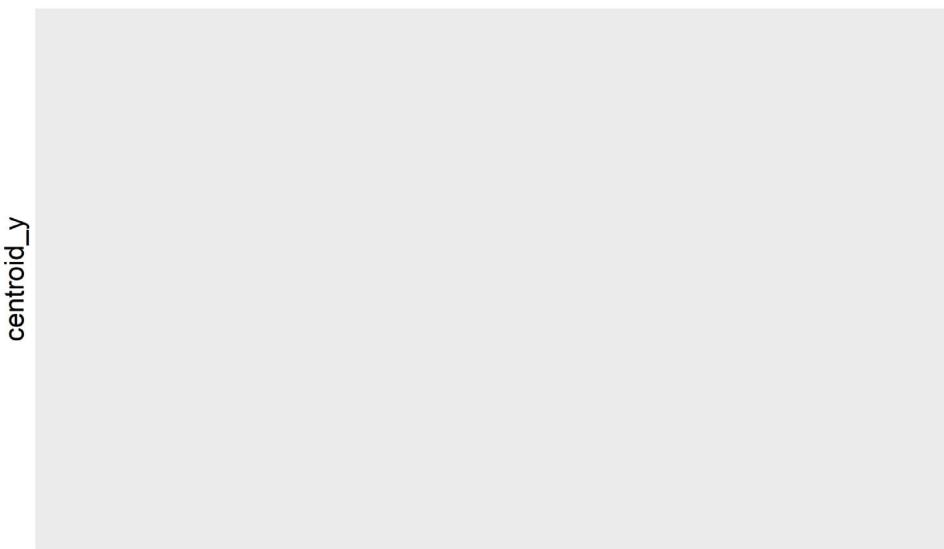
Then go deeper...

```
hectare_counts <- squirrels %>%
  group_by(hectare) %>%
  summarize(n = n())
```

```
hectare_centroids <- squirrels %>%
  group_by(hectare) %>%
  summarize(
    centroid_x = mean(long),
    centroid_y = mean(lat))
```

```
squirrels %>%
  left_join(hectare_counts,
            by = "hectare") %>%
  left_join(hectare_centroids,
            by = "hectare") %>%
  ggplot(aes(x = centroid_x,
             y = centroid_y,
             color = n)) +
  geom_hex()
```

```
## Warning: Computation failed in `stat_binhex()`
## Caused by error in `compute_group()`
## ! The package "hexbin" is required
```



```
## Warning: Computation failed in `stat_binhex()`
## Caused by error in `compute_group()`
## ! The package "hexbin" is required
```

```
hectare_counts <- squirrels %>%
  group_by(hectare) %>%
  summarize(n = n())

hectare_centroids <- squirrels %>%
  group_by(hectare) %>%
  summarize(
    centroid_x = mean(long),
    centroid_y = mean(lat)
  )

squirrels %>%
  left_join(hectare_counts, by = "hectare") %>%
  left_join(hectare_centroids, by = "hectare") %>%
  ggplot(aes(x = centroid_x, y = centroid_y, color = n)) +
  geom_hex()
```



The squirrel is staring at me!

```
squirrels %>%  
  filter(str_detect(other_interactions, "star")) %>%  
  select(shift, age, other_interactions)  
  
## # A tibble: 11 × 3  
##   shift    age other_interactions  
##   <chr> <chr> <chr>  
## 1 AM     Adult  staring at us  
## 2 PM     Adult  he took 2 steps then turned and stared at me  
## 3 PM     Adult  stared  
## 4 PM     Adult  stared  
## 5 PM     Adult  stared  
## 6 PM     Adult  stared & then went back up tree–then ran to diffe...  
## 7 PM     Adult  stared at me  
## 8 AM     Adult  approaches (saw me & came forward), runs from (sta...  
## 9 AM     Adult  started climbing down to me  
## 10 PM    Adult  stared at me  
## # i 1 more row
```



Communicating for your audience

- + Avoid:
 - + jargon,
 - + uninterpreted results,
 - + lengthy output
- + Pay attention to:
 - + organization,
 - + presentation,
 - + flow
- + Don't forget about:
 - + code style,
 - + coding best practices,
 - + meaningful commits
- + Be open to:
 - + suggestions,
 - + feedback,
 - + taking (calculated) risks



Sources

- + Mine Åtetinkaya-Rundel's Data Science in a Box ([link](#))
- + Jeffery T. Leek and Roger D. Peng. "What is the question?." *Science* 347.6228 (2015): 1314-1315.
- + Roger D. Peng and Elizabeth Matsui. "The Art of Data Science." A Guide for Anyone Who Works with Data. Skybrude Consulting, LLC (2015).

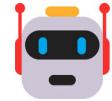


Wrapping Up...



Data Science for Psychologists

Functions and iteration

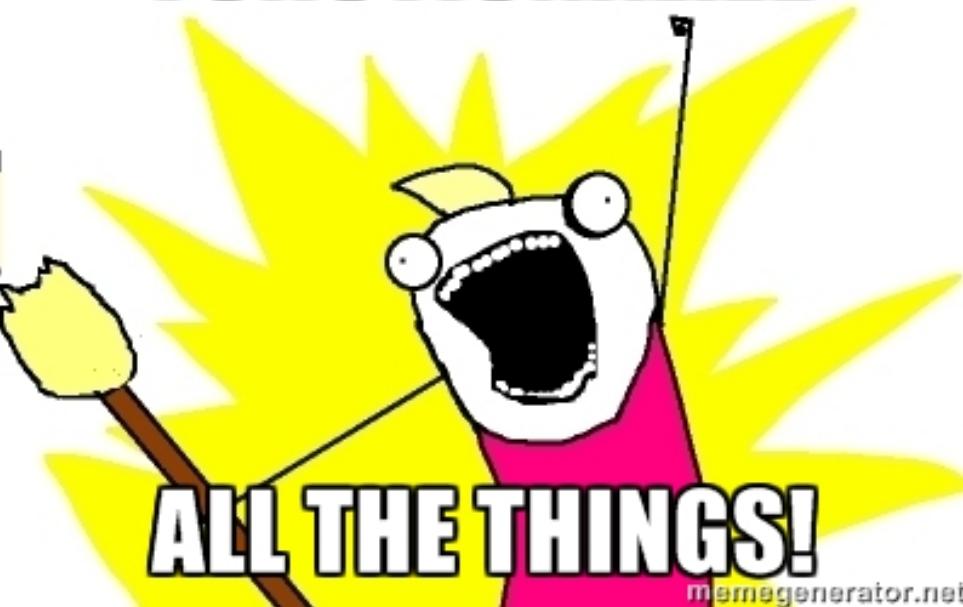


Functions



When should you write a function?

FUNCTIONALIZE



- + When you've copied and pasted a block of code more than twice.

How many times will we need to copy and paste the code we developed to scrape data on all of the art??



Data Science for Psychologists

Why functions?

- + Automate common tasks in a more powerful and general way than copy-and-pasting:
 - + Give your function an evocative name that makes your code easier to understand
 - + As requirements change, only need to update code in one place, instead of many
 - + Eliminate chance of making incidental mistakes when you copy and paste
 - + (for example... updating a variable name in one place, but not in another)
- + Long-term
 - + Improve your reach as a data scientist by writing functions (and packages!) that others use
 - + Useful for internal lab management as well...



Inputs

How many inputs does the following code have?

```
## set url ----  
first_info_url <- "https://collections.ed.ac.uk/art/record/22024?highlight=*:*"  
  
## read page at url ----  
page <- read_html(first_info_url)  
  
## scrape headers ----  
headers <- page %>%  
  html_nodes("th") %>%  
  html_text()  
  
## scrape values ----  
values <- page %>%  
  html_nodes("td") %>%  
  html_text() %>%  
  str_squish()  
  
## put together in a tibble and add link to help keep track ----  
tibble(headers, values) %>%  
  pivot_wider(names_from = headers, values_from = values) %>%  
  add_column(link = first_info_url)
```



Data Science for Psychologists

Inputs

How many inputs does the following code have?

```
## set url ----  
first_info_url <- "https://collections.ed.ac.uk/art/record/22024?highlight=*:*"  
  
## read page at url ----  
page <- read_html(first_info_url)  
  
## scrape headers ----  
headers <- page %>%  
  html_nodes("th") %>%  
  html_text()  
  
## scrape values ----  
values <- page %>%  
  html_nodes("td") %>%  
  html_text() %>%  
  str_squish()  
  
## put together in a tibble and add link to help keep track ----  
tibble(headers, values) %>%  
  pivot_wider(names_from = headers, values_from = values) %>%  
  add_column(link = first_info_url)
```



Turn your code into a function

- + Pick a short but informative **name**, preferably a verb.

```
scrape_art_info <-
```



Turn your code into a function

- + Pick a short but informative **name**, preferably a verb.
- + List inputs, or **arguments**, to the function inside `function`. If we had more arguments the call would look like `function(x, y, z)`.

```
scrape_art_info <- function(x){  
}  
}
```



Turn your code into a function

- + Pick a short but informative **name**, preferably a verb.
- + List inputs, or **arguments**, to the function inside `function`. If we had more the call would look like `function(x, y, z)`.
- + Place the **code** you have developed in body of the function, a `{ }` block that immediately follows `function(...)`.

```
scrape_art_info <- function(x){  
  # code we developed earlier  
  # to scrape info  
  # on single art piece goes here  
}
```



Your turn!

- + `class git repo > AE 09 - UoE Art + Functions.`
- + Open `02-functionalize.R`
- + Follow along, and fill in the blanks if you like



```
scrape_art_info <- function(x){  
  
  # read page at url ----  
  page <- read_html(x)  
  
  # scrape headers ----  
  headers <- page %>%  
    html_nodes("th") %>%  
    html_text()  
  
  # scrape values ----  
  values <- page %>%  
    html_nodes("td") %>%  
    html_text() %>%  
    str_squish()  
  
  # put together in a tibble and add link to help keep track ----  
  tibble(headers, values) %>%  
    pivot_wider(names_from = headers, values_from = values) %>%  
    add_column(link = x)  
}
```



Writing functions



What goes in / what comes out?

- + They take input(s) defined in the function definition

```
function([inputs separated by commas]){
  # what to do with those inputs
}
```

- + By default they return the last value computed in the function

```
scrape_page <- function(x){
  # do bunch of stuff with the input...
  # return a tibble
  tibble(...)
}
```

- + You can define more outputs to be returned in a list as well as nice print methods
 - + (but we won't go there for now...)



What is going on here?

```
add_2 <- function(x){  
  x + 2  
  1000  
}
```

```
add_2(3)
```

```
## [1] 1000
```

```
add_2(10)
```

```
## [1] 1000
```



Naming functions

"There are only two hard things in Computer Science: cache invalidation and naming things."
-- Phil Karlton



Naming functions

- + Names should be short but clearly evoke what the function does
- + Names should be verbs, not nouns
- + Multi-word names should be separated by underscores (snake_case as opposed to camelCase)
- + A family of functions should be named similarly (scrape_page(), scrape_speech() OR str_remove(), str_replace() etc.)
- + Avoid overwriting existing (especially widely used) functions

```
# JUST DON'T
mean <- function(x){
  x * 3
}
```



Wrapping Up



Data Science for Psychologists

Automation



Data Science for Psychologists

Define the task

- + Goal: Scrape info on all 2909 art pieces in the collection
- + So far:

```
scrape_art_info(ueo_art$link[1])  
scrape_art_info(ueo_art$link[2])  
scrape_art_info(ueo_art$link[3])
```

- + What else do we need to do?
 - + Run the `scrape_art_info()` function on all 2909 links
 - + Combine the resulting data frames from each run into one giant data frame with 2909 rows



Inputs

You now have a function that will scrape the relevant info on art pieces given the URL of its individual info page. Where can we get a list of URLs of each of the art pieces in the collection?

From the data frame you constructed previously: `uo_e_art$link`



Iteration

How can we tell R to apply the `scrape_art_info()` function to each link in `uo_e_art$link`?

- + Option 1: Write a **for loop**, i.e. explicitly tell R to visit a link, apply the function, store the result, then visit the next link, apply the function, append the result to the stored result from the previous link, and so on and so forth.
- + Option 2: **Map** the function to each element in the list of links, and let R take care of the storing and appending of results.
- + We'll go with Option 2!



How does mapping work?

Suppose we have exam 1 and exam 2 scores of 4 students stored in a list...

```
exam_scores <- list(  
  exam1 <- c(80, 90, 70, 50),  
  exam2 <- c(85, 83, 45, 60)  
)
```

...and we find the mean score in each exam

```
map(exam_scores, mean)
```

```
## [[1]]  
## [1] 72.5  
##  
## [[2]]  
## [1] 68.25
```



Data Science for Psychologists

...and suppose we want the results as a numeric (double) vector

```
map_dbl(exam_scores, mean)
```

```
## [1] 72.50 68.25
```

...or as a character string

```
map_chr(exam_scores, mean)
```

```
## Warning: Automatic coercion from double to character was deprecated in
## purrr 1.0.0.
```

```
## i Please use an explicit call to `as.character()` within
##   `map_chr()` instead.
```

```
## Call `lifecycle::last_lifecycle_warnings()` to see where this
## warning was generated.
```

```
## [1] "72.500000" "68.250000"
```



map_something

Functions for looping over an object and returning a value (of a specific type):

- + `map()` - returns a list
- + `map_lgl()` - returns a logical vector
- + `map_int()` - returns an integer vector
- + `map_dbl()` - returns a double vector
- + `map_chr()` - returns a character vector
- + `map_df()` / `map_dfr()` - returns a data frame by row binding
- + `map_dfc()` - returns a data frame by column binding
- + ...



Deeper Dive into Functions...



Writing and calling functions



Why?



Make your code more readable. - Allows you to re-use related lines of code for slightly different tasks.



Makes testing your code easier.



Data Science for Psychologists

What is a function?



A way of turning some inputs into some outputs.



A way of tying together related pieces of code.



An object.



Function creation

Syntax for function creation:

```
f = function(arguments) {  
  body  
}
```



arguments are assignments of values to variables.



body is the code you want to execute.



So for example, if we return to our steak-cooking example from the first week, we might define the following function:

```
steak_directions = function(temp, steak_type) {  
  if(steak_type == "rare" & temp > 115) {  
    return("take your steak off!")  
  } else if(steak_type == "med_rare" & temp > 125) {  
    return("take your steak off!")  
  }  
  "you can keep cooking"  
}
```

We can see the arguments and body of the function using `formals` and `body`, respectively.

```
formals(steak_directions)
```

```
## $temp
```

```
##
```

```
##
```

```
#
```

 Data Science for Psychologists

Function arguments

Once you have a function, you call it by specifying the values for all of the arguments.
The values can be specified in two ways:



By position: first argument is assigned to the first variable in the function definition, second argument to the second variable in the function definition, and so on



By name: arguments are specified by name instead of being inferred based on position.

The two can be combined.



So for example, the following are all the same:

```
steak_directions(temp = 120, steak_type = "rare")
```

```
## [1] "take your steak off!"
```

```
steak_directions(steak_type = "rare", temp = 120)
```

```
## [1] "take your steak off!"
```

```
steak_directions(120, "rare")
```

```
## [1] "take your steak off!"
```

But this is of course different and will not work:

```
simulate_birthdays("rare", 120)
```

Default arguments

When you define a function, you can set default values for any/all of the arguments.

When you call such a function, if you don't specify a value for that argument, it will automatically go to the default value.

For example, in the following function the default argument for `steak_type` is "rare".

```
steak_directions = function(temp, steak_type = "rare") {  
  if(steak_type == "rare" & temp > 115) {  
    return("take your steak off!")  
  } else if(steak_type == "med_rare" & temp > 125) {  
    return("take your steak off!")  
  }  
  "you can keep cooking"  
}
```

If we don't specify `steak_type`, we will get results as if we had specified it to be "rare", but we can also over-ride that argument if we set it explicitly:



Return values

When a function is called, the commands in the body of the function are executed, and a value is returned. The return value is either:

- + The value of last command executed, or
- + A value set explicitly using the `return` syntax.

The commands in the body of the function are executed until a `return` statement is encountered or the end of the body is reached, whichever comes first.



Let's think through what happens when we call the function these two ways:

```
steak_directions = function(temp, steak_type = "rare") {  
  if(steak_type == "rare" & temp > 115) {  
    return("take your steak off!")  
  } else if(steak_type == "med_rare" & temp > 125) {  
    return("take your steak off!")  
  }  
  "you can keep cooking"  
}  
steak_directions(stake_type = "rare", temp = 120)
```

```
## [1] "take your steak off!"
```

```
steak_directions(stake_type = "med_rare", temp = 120)
```

```
## [1] "you can keep cooking"
```



Invisible return

Invisible return is a bit R-specific:



If you use `invisible` instead of `return` in a function definition, the result will be discarded unless it is assigned.



Although `return` is not usually something that you will use, some of the built-in functions use them.

```
square_invisible = function(x) invisible(x^2)
square = function(x) x^2
```

If we call `square(4)` we get output: 16

```
square(4)
```

```
## [1] 16
```



Data Science for Psychologists

Another example: compare the two versions of oddcount:

```
oddcount = function(x) {  
  k = 0  
  for(n in x) {  
    if (n %% 2 == 1) k = k + 1  
  }  
  return(k)  
}  
oddcount(c(0, 5))
```

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

```
oddcount = function(x) {  
  k = 0  
  for(n in x) {  
    if (n %% 2 == 1) k = k + 1  
  }  
}
```

 oddcount(c(0, 5))

Data Science for Psychologists

Environments and scope

When you call a function, the commands in the function body are executed, but not in exactly the same way they would be if you simply ran them one at a time in an interactive R session.
The commands are executed in the function's *environment*.

Environments

Ok, so what is an environment?

- + An environment binds names to values.

- + Every environment has a parent (except for the empty environment).

What are they good for?

- + The purpose of environments is to describe where to look for variables.



For example, have you ever wondered how R finds functions?

The function `lm` is not in the global environment, as we can see if we just call `ls`:

`ls()`

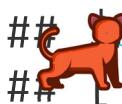
```
## [1] "add_2"                      "bar_plot"
## [3] "brexit"                     "cat_lovers"
## [5] "d"                           "dates"
## [7] "dev.off"                    "df"
## [9] "dodged_bar_plot"            "edi_airbnb"
## [11] "edi_airbnb_cleaned_names"  "edi_airbnb_col_names"
## [13] "enrollment"                 "exam_scores"
## [15] "exam1"                      "exam2"
## [17] "f"                           "fav_food"
## [19] "fig"                        "g"
## [21] "h"                           "hectare_centroids"
## [23] "hectare_counts"             "heights"
## [25] "hook_output"                "hotels"
## [27] "jabba"                      "l"
## [29] "last_row"                   "loans"
```



But we are able to access it and, for instance, ask what its arguments are:

```
head(formals(lm))
```

```
## $formula
##
## $data
##
## $subset
##
## $weights
##
## $na.action
##
## $method
```



Data Science for Psychologists

Functions live in environments corresponding to the package they are defined in. For `lm`, this is `stats`.

```
environment(lm)
```

```
## <environment: namespace:stats>
```

Package environments are all ancestral to the global environment, so when R found that `lm` wasn't defined in the global environment, it looked through the packages until it found `lm` defined in `stats`.



Function environments

...

When a function is called, its body is evaluated in an execution environment whose parent is the function's environment.

```
w = 12
f = function(y) {
  d = 8
  h = function() {
    return(d * (w + y))
  }
  cat("h's environment: ", "\n")
  print(environment(h))
  cat("h's parent environment:", "\n")
  print(parent.env(environment(h)))
  return(h())
}
f(1)
```



Compare with:

```
f = function(y) {  
  d = 8  
  return(h())  
}  
  
h = function() {  
  cat("h's environment:", "\n")  
  print(environment(h))  
  cat("h's parent environment:", "\n")  
  print(parent.env(environment(h)))  
  return(d * (w + y))  
}  
f(5)
```

```
## h's environment:  
## <environment: R_GlobalEnv>  
## h's parent environment:  
## <environment: 0x000001f84bd15cb0>  
## Data Science for Psychologists
```

This perhaps seems overly baroque, but the take-home points about environments (and the reason why they are set up the way they are) are:



Commands called in the body of the function usually have access to values of the variables in that function plus variables at higher levels.



Variables defined in the body of a function go away after the function exits.



Side effects

A function has a *side effect* if it does anything other than compute a return value, for instance, if it changes the values of other variables in the environment it is defined in, or adds variables to the environment. We generally don't want functions to have side effects, because they make code more confusing and more difficult to test.

In R, functions *can* have side effects, but it is discouraged by both the language itself and by programming norms.



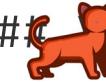
...

Remember that functions can see variables defined in the parent environments.

What they can't do is change the values of those variables (except with a special operator).

For example:

```
w = 12
f = function(y) {
  d = 8
  w = w + 1
  y = y - 2
  cat(sprintf("Value of w: %i", w))
  h = function() {
    return(d*(w+y))
  }
  return(h())
}
t = 4
f(t)
```

# Value of w: 12

Data Science for Psychologists

Sources

- + Mine Å‡etinkaya-Rundel's Data Science in a Box ([link](#))
- + Julia Fukuyama's EDA ([link](#))



Wrapping Up...



Data Science for Psychologists