R *vs.* SPSS

“SPSS is like a bus - easy to use for the standard things, but very frustrating if you want to do something that is not already pre-programmed.

R is a 4-wheel drive off-roader, with a bike on the back, a kayak on top, good walking and running shoes in the passenger seat, and mountain climbing and spelunking gear in the back.

R can take you anywhere you want to go if you take time to learn how to use the equipment, but that is going to take longer than learning where the bus stops are in SPSS.” (Greg Snow, 2010, stackoverflow.com).

In meme form…

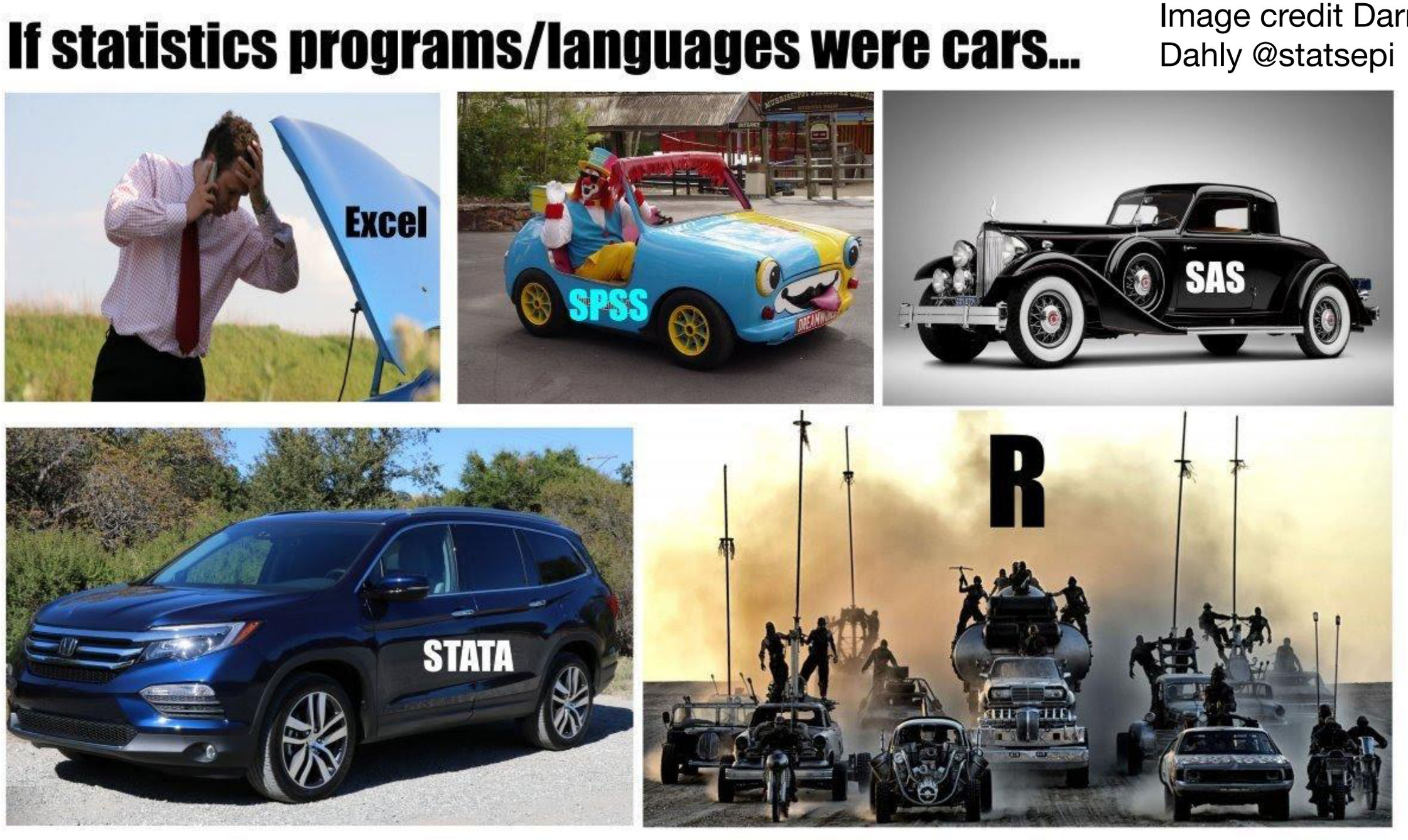
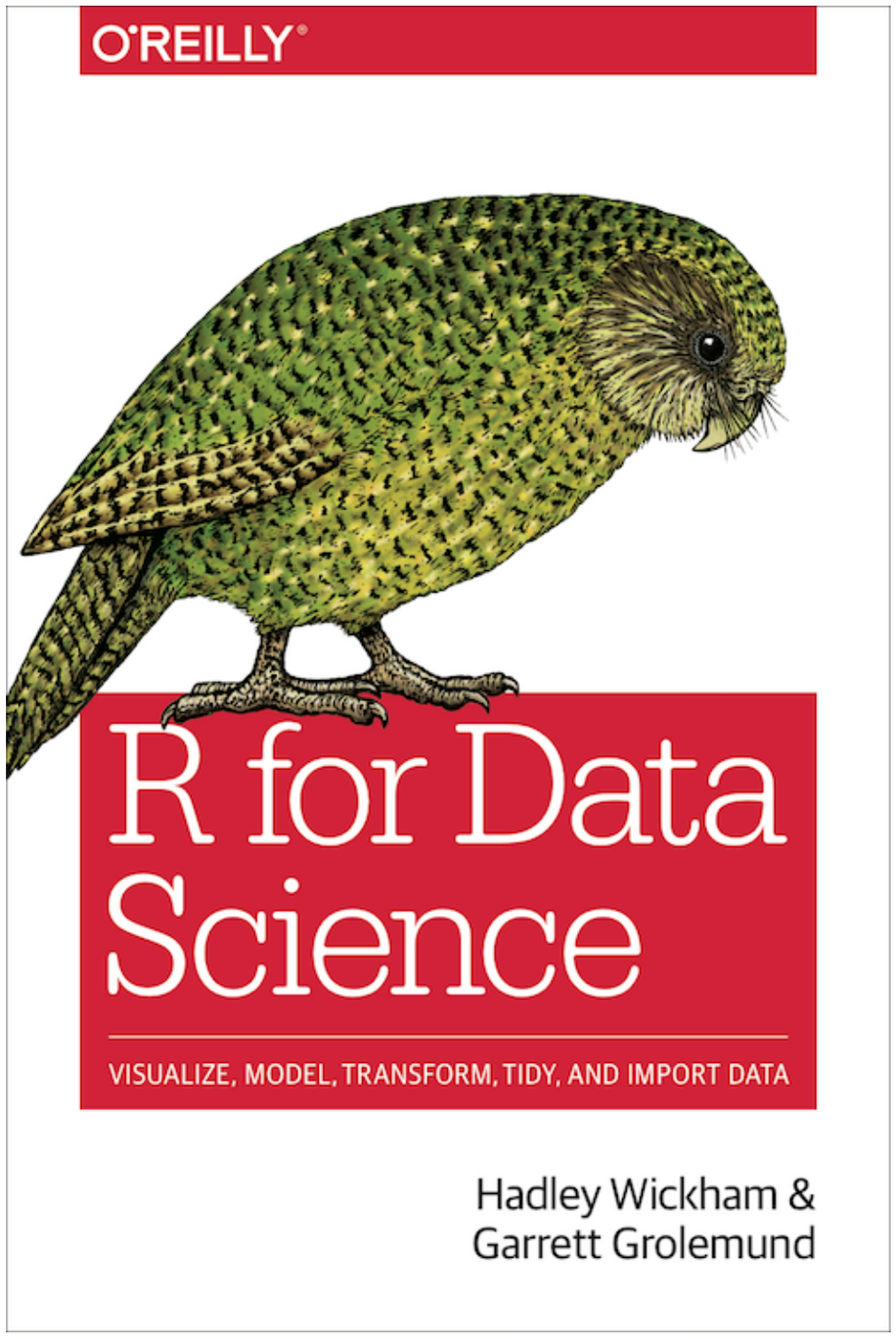


Image credit Darren Dahly @statsepi



Available electronically for free at:

http://r4ds.had.co.nz

“Hadley Wickham, the Man Who Revolutionized R”

Chief Scientist at RStudio, author of key R packages incl. `ggplot2`, `tidyr`, `dplyr` - all components of the tidyverse.

https://priceonomics.com/hadley-wickham-the-man-who-revolutionized-r/

R skills are in high demand…

February 11, 2014

R skills attract the highest salaries

Two recent salary surveys have shown that R language skills attract median salaries in excess of $110,000 in the United States.

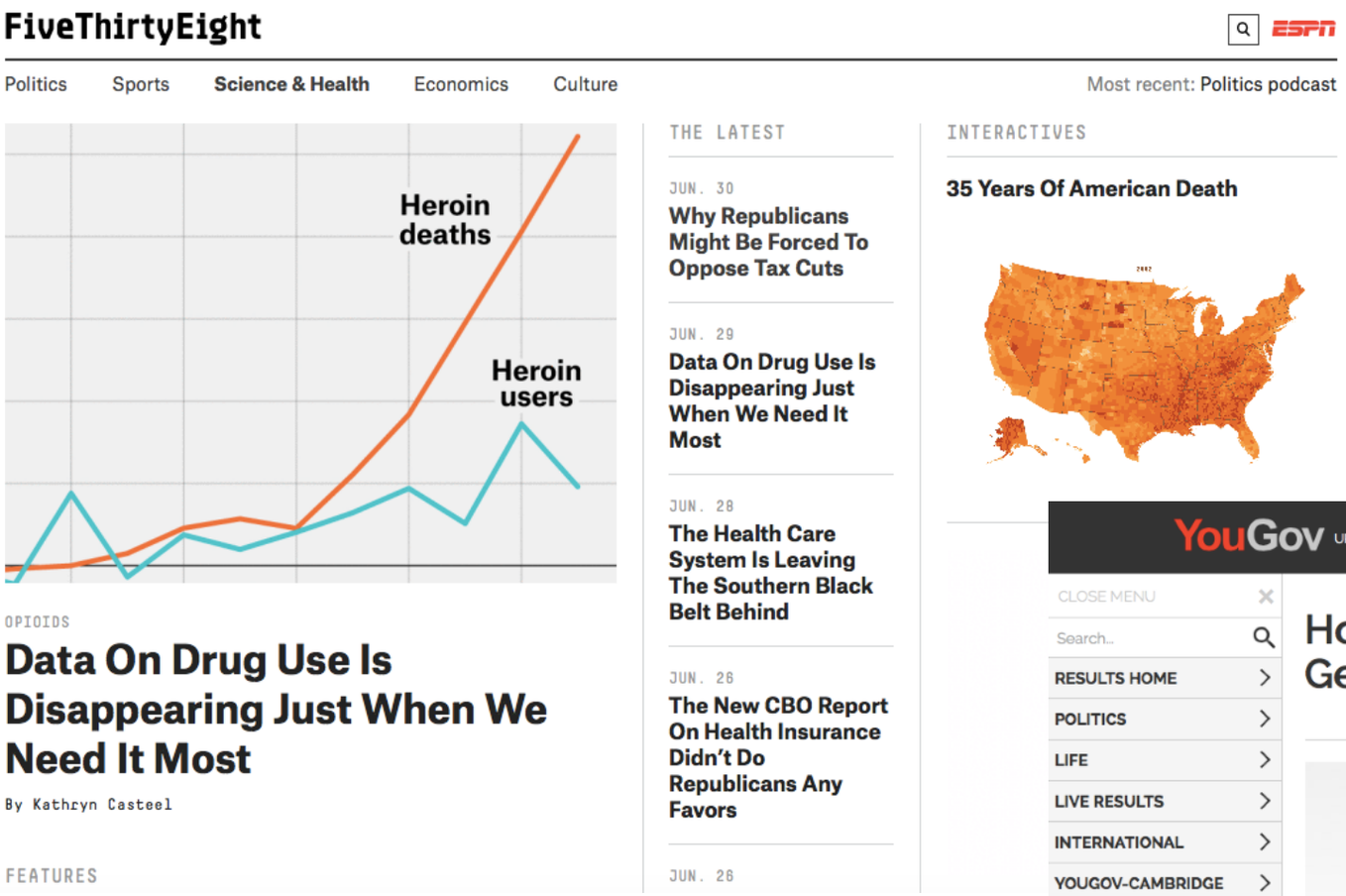
In the 2014 Dice Tech Salary Survey of over 17,000 technology professionals, the highest-paid IT skill was R programming. While big-data skills in general featured strongly in the top tier, having R at the top of the list reflects the strong demand for skills to make sense of, and extract value from big data.

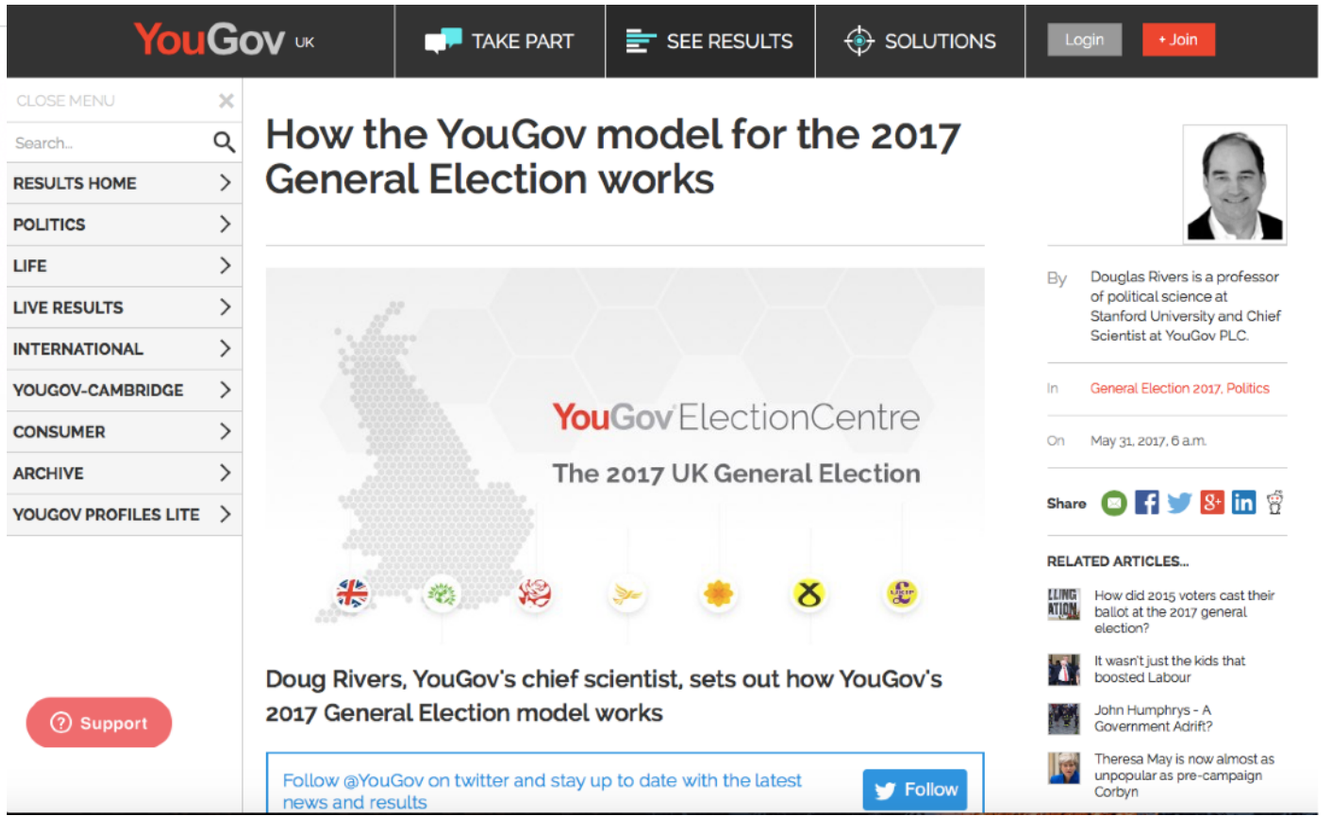
AVERAGE SALARY FOR High Paying Skills and Experience

SKILL	2013	YR/YR CHANGE
R	$ 115,531	n/a
NoSQL	$ 114,796	1.6%
MapReduce	$ 114,396	n/a
PMBok	$ 112,382	1.3%
Cassandra	$ 112,382	n/a
Omnigraffle	$ 111,039	0.3%
Pig	$ 109,561	n/a
SOA (Service Oriented Architecture)	$ 108,997	-0.5%
Hadoop	$ 108,669	-5.6%
Mongo DB	$ 107,825	-0.4%

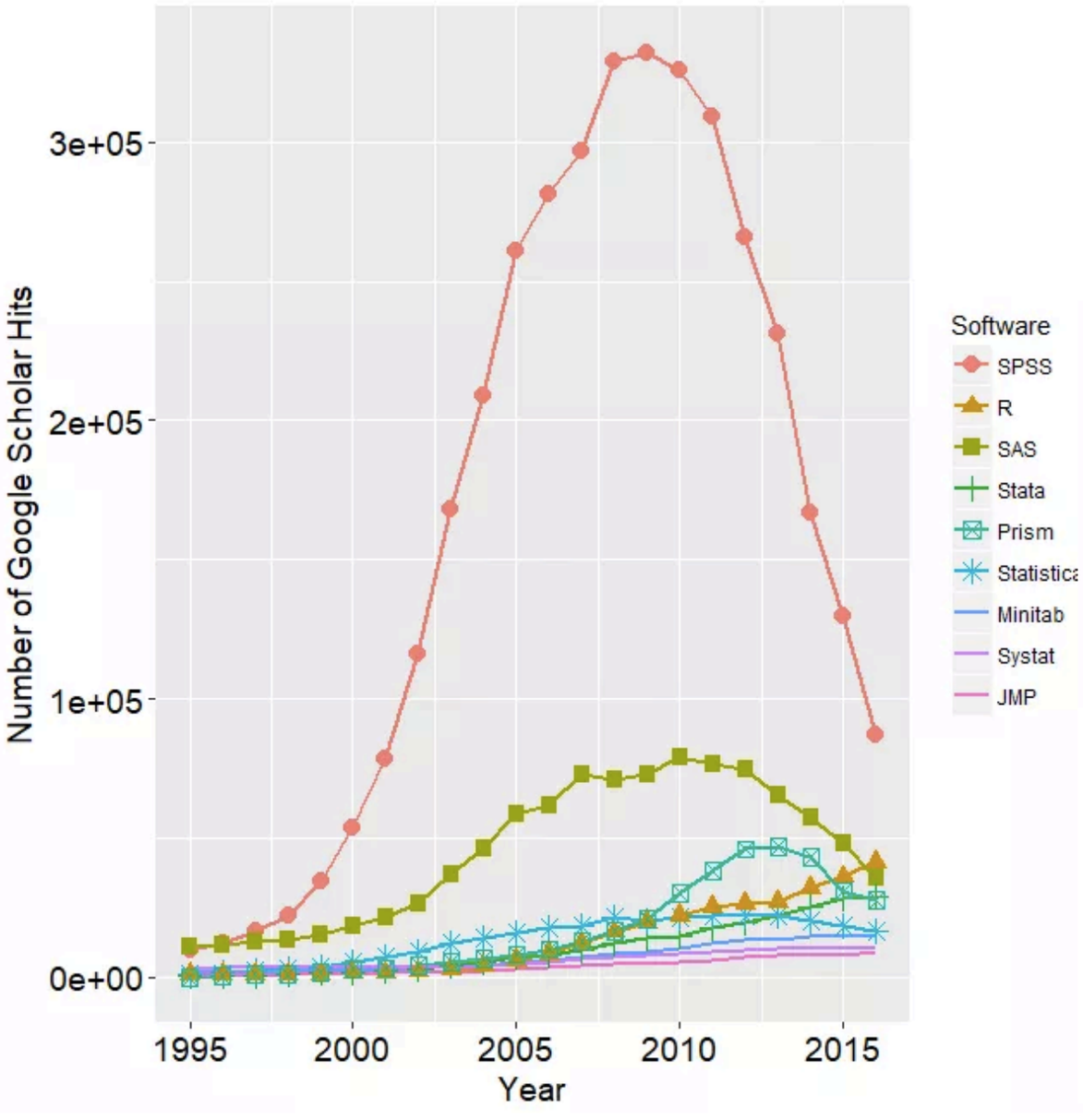
Similarly, the recent O'Reilly Data Scientist Survey also found R skills amongst those that pay in the $110,000-$125,000 range (albeit amongst a much smaller and specialized sample of respondents).

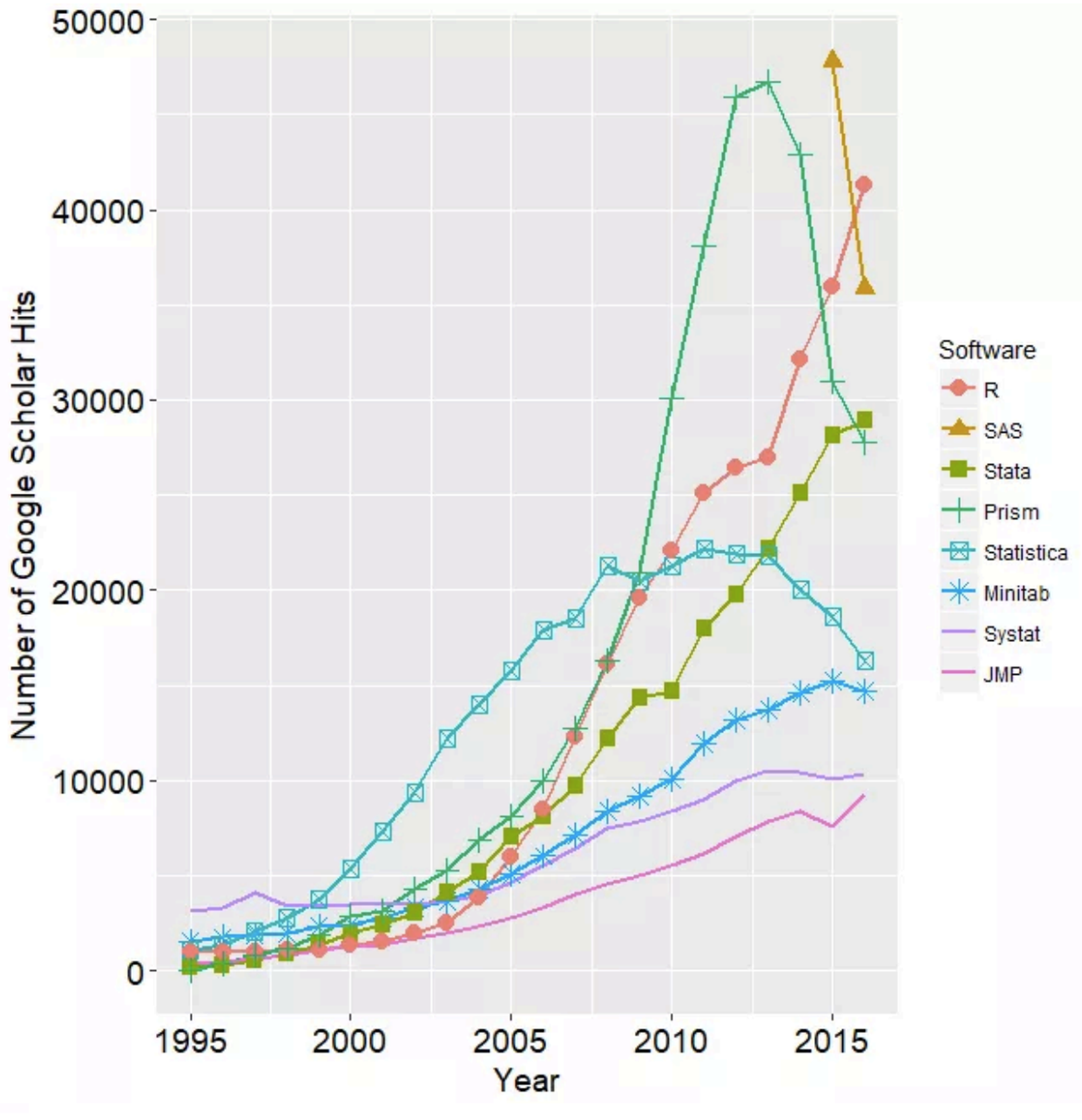
and R is widely used by a number of organisations (incl. the ONS)…





and across academia…





Tyler Burleigh
@tylerburleigh

If your department teaches quantitative skills in SPSS, SAS, or Matlab, is it really preparing students for quantitative jobs outside of academia? #datascientist #academia #rstats #python

Data scientist jobs

4893 data scientist job ads were included in this analysis.

Tools / languages | Other skills | Degrees | Disciplines

Tools and languages in "Data Scientist" job ads (N = 4893)

SQL
Python
Excel
R
Hive
Spark
Tableau
SAS
Matlab
JavaScript
SPSS

0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100

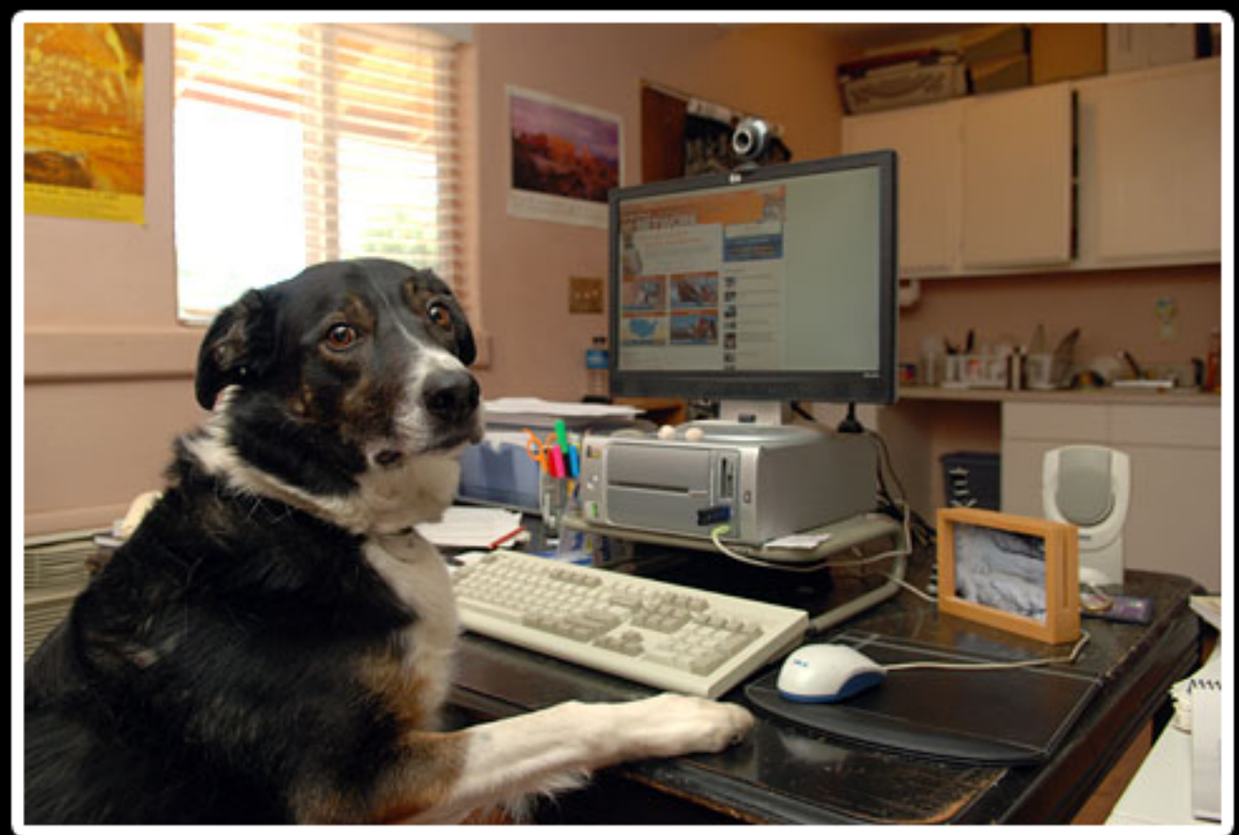
percent of jobs

legend
Entry level
Associate
Mid-Senior level

Jobs were scraped from LinkedIn in Sept 2019
Locations included NYC, SF, Seattle, Boston, and Toronto
Positions were Entry, Associate, and Senior levels

THE TWO STATES OF EVERY PROGRAMMER

I AM A GOD.



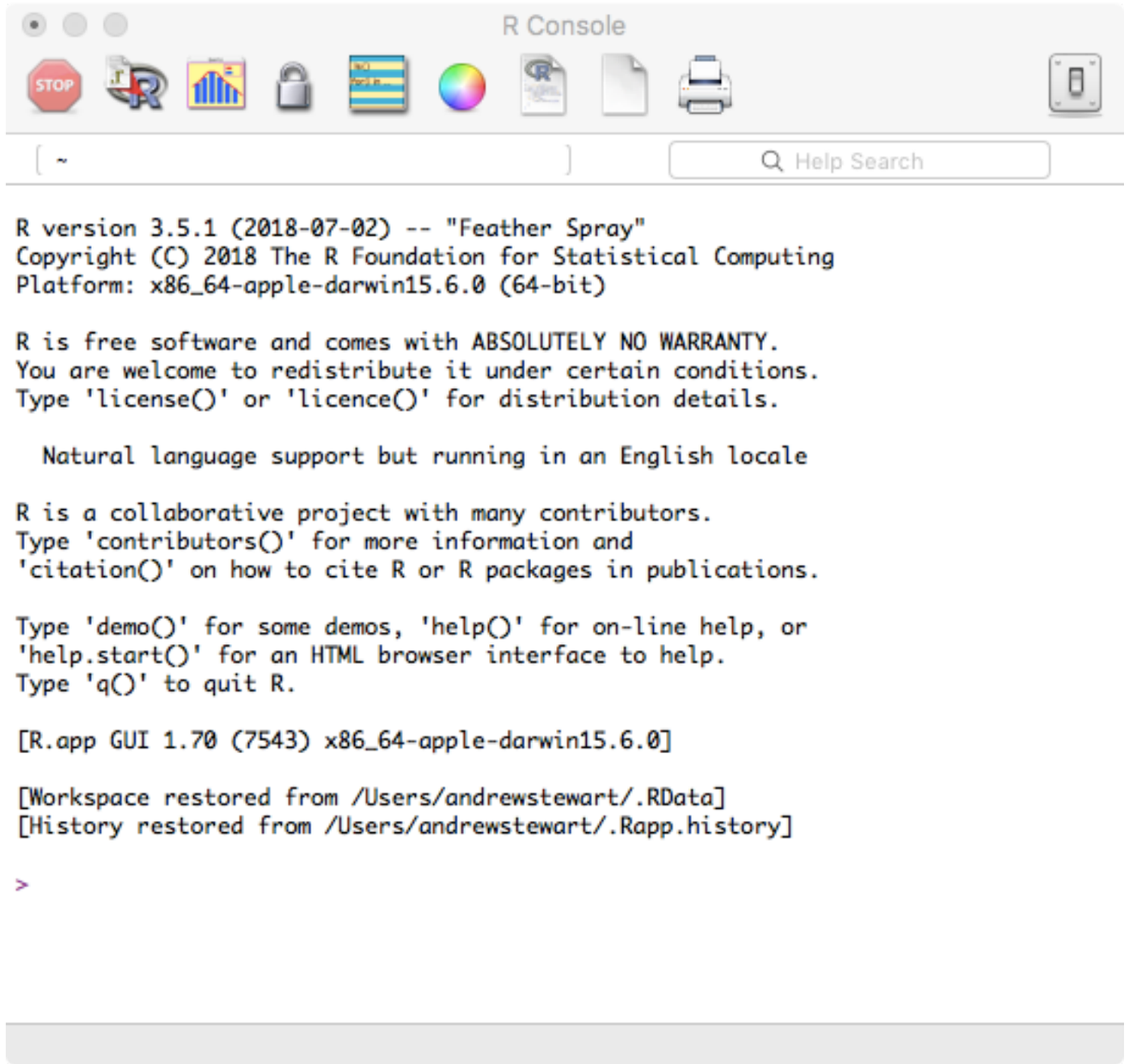
I HAVE NO IDEA WHAT I'M DOING.

Starting R

- R is open source (i.e., free to download and add to). Main R site is:
- *www.r-project.org*
- From here you can download R (from one of the CRAN[1] mirrors for Windows, Mac and UNIX).
- R updates regularly (you need to update manually).

[1]CRAN = The Comprehensive R Archive Network

- When you first load R it looks like this:



- Rather than using the R interface, you should use the RStudio graphical interface.

- Download it from www.rstudio.com

- When you use RStudio, it looks like this:

RStudio File Edit Code View Plots Session Build Debug Tools Window Help — Thu 15:21

~/Desktop/Air Work/MRes 2016:17/R workshop MRes/R workshop directory/Workshop - RStudio

Go to file/function | Addins | Workshop

Workshop script2.R | DV

Source on Save | Run | Source

```
library (lmerTest)
library (lsmeans)
library (pbkrtest)

#Read in First Pass Data
FPs <- read.csv("~/Desktop/Air Work/R analyses/Indirect Request Expt/Experiment 1 - probability of success - Libby's data/FP

#this sets up the contrasts so that the intercept in the mixed LMM is the grand mean (i.e., the mean of all conditions)
my.coding <- matrix (c(.5, -.5))

contrasts (DV$Context)<-matrix (c(.5, -.5))
contrasts (DV$Sentence)<-matrix (c(.5, -.5))

#construct the modelswith crossed random effects for subjects and items for the pre-critical, critical and post-crtical regi

model.full <- lmer (RT ~ Context*Sentence + (1+Context*Sentence |Subject) + (1+Context*Sentence |Item), data=DV, REML=TRUE)
model.null <- lmer (RT ~ (1+Context*Sentence |Subject) + (1+Context*Sentence |Item), data=DV, REML=TRUE)

summary (model.full)
lsmeans (model.full, pairwise~Context*Sentence, adjust="none")

fpmodels <- lmer (statement ~ Meaning*Probability + (1+Meaning*Probability |R_s) + (1+Meaning*Probability |Item), data=FPs
```

23:1 (Top Level) | R Script

Console ~/Desktop/Air Work/MRes 2016:17/R workshop MRes/R workshop directory/Workshop/

```
The following object is masked from 'package:stats':

    step

> library("lsmeans", lib.loc="/Library/Frameworks/R.framework/Versions/3.3/Resources/library")
Loading required package: estimability

Attaching package: 'lsmeans'

The following object is masked from 'package:lmerTest':

    lsmeans

> model.full <- lmer (RT ~ Context*Sentence + (1+Context*Sentence |Subject) + (1+Context*Sentence |Item), data=DV, REML=TRUE)
```

Error in strsplit(keys, " *, *") : non-character argument

Environment | History

Import Dataset | List

Global Environment

Data

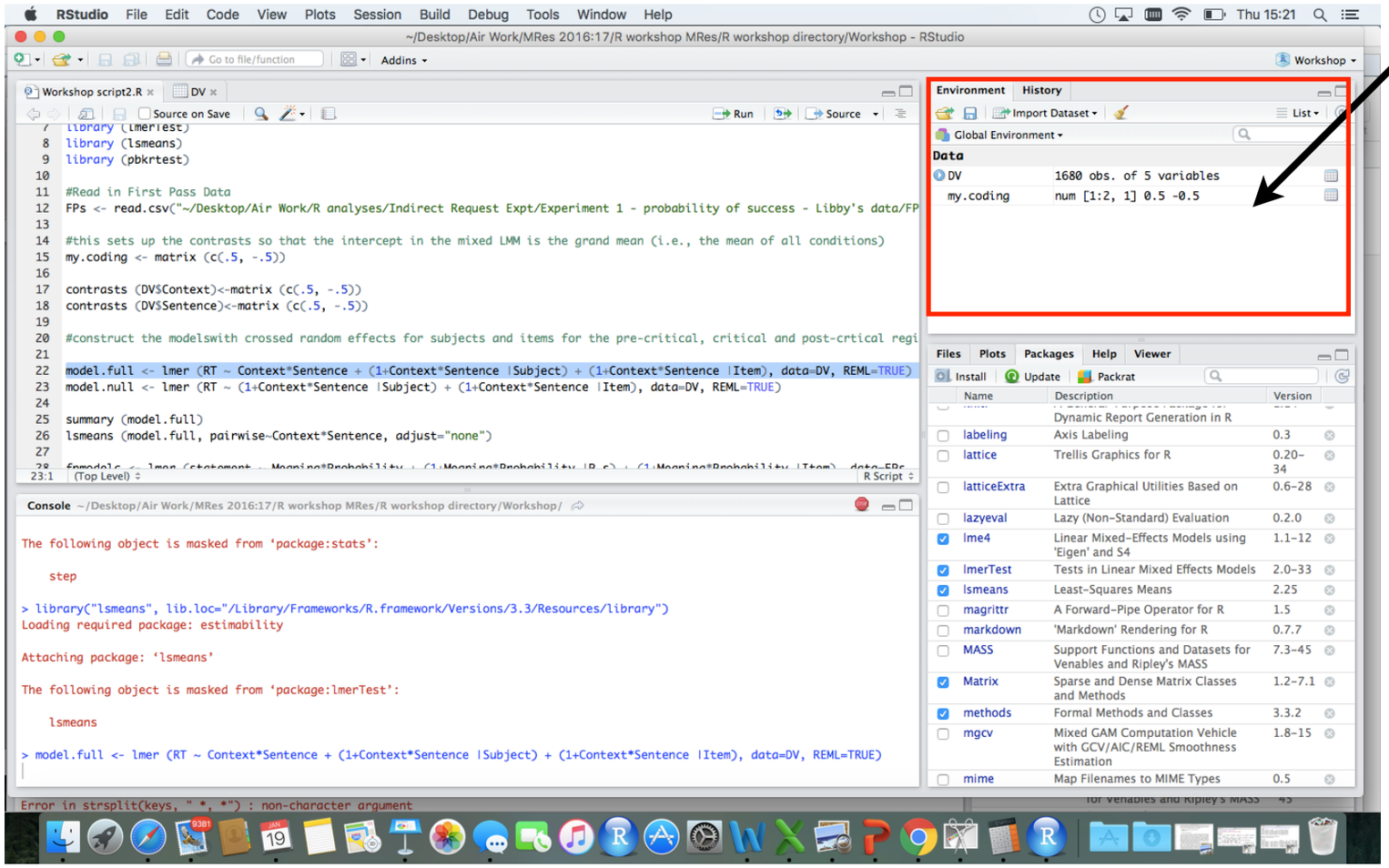
DV	1680 obs. of 5 variables
my.coding	num [1:2, 1] 0.5 -0.5

Files | Plots | Packages | Help | Viewer

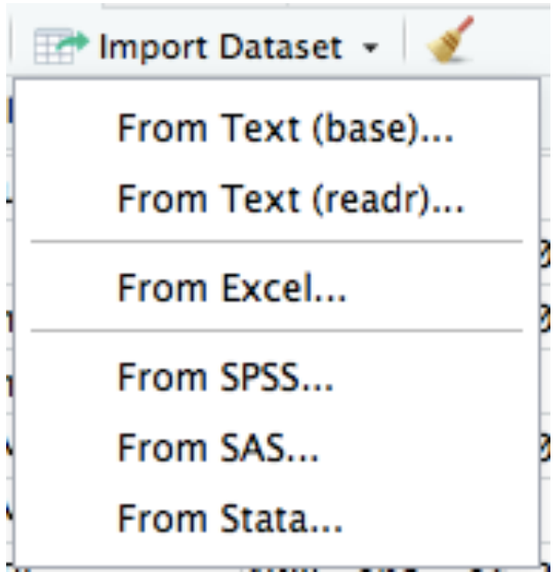
Install | Update | Packrat

Name	Description	Version
	Dynamic Report Generation in R	
labeling	Axis Labeling	0.3
lattice	Trellis Graphics for R	0.20-34
latticeExtra	Extra Graphical Utilities Based on Lattice	0.6-28
lazyeval	Lazy (Non-Standard) Evaluation	0.2.0
lme4	Linear Mixed-Effects Models using 'Eigen' and S4	1.1-12
lmerTest	Tests in Linear Mixed Effects Models	2.0-33
lsmeans	Least-Squares Means	2.25
magrittr	A Forward-Pipe Operator for R	1.5
markdown	'Markdown' Rendering for R	0.7.7
MASS	Support Functions and Datasets for Venables and Ripley's MASS	7.3-45
Matrix	Sparse and Dense Matrix Classes and Methods	1.2-7.1
methods	Formal Methods and Classes	3.3.2
mgcv	Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness Estimation	1.8-15
mime	Map Filenames to MIME Types	0.5

Here is where you import your data.



- Importing data (CSV (Text), Excel, SPSS, SAS and Stata format). CSV can be delimited by commas, tabs etc. Most of the time you'll use the `readr` import function…



Import Text Data

File/Url:

~/Desktop/Air Work/R analyses/R courses/R course/R Work Folder/priming data.txt

Browse...

Data Preview:

Subject *(integer)*	Priming Condition *(character)*	RT *(integer)*
1	LONG	500
2	LONG	551
3	LONG	479
4	LONG	583
5	LONG	522
6	SHORT	399
7	SHORT	423
8	SHORT	444
9	SHORT	410
10	SHORT	398

Previewing first 50 entries.

You can change the type of each variable by clicking on the down arrow.

You should change this to type Factor (LONG vs. SHORT).

Import Options:

Name: priming_data

Skip: 0

☑ First Row as Names

☑ Trim Spaces

☑ Open Data Viewer

Delimiter: Tab

Quotes: Default

Locale: Configure...

Escape: None

Comment: Default

NA: Default

Code Preview:

```
library(readr)
priming_data <- read_delim("~/Desktop/Air Work/R analyses/R courses
/R course/R Work Folder/priming data.txt",
    "\t", escape_double = FALSE, trim_ws = TRUE)
View(priming_data)
```

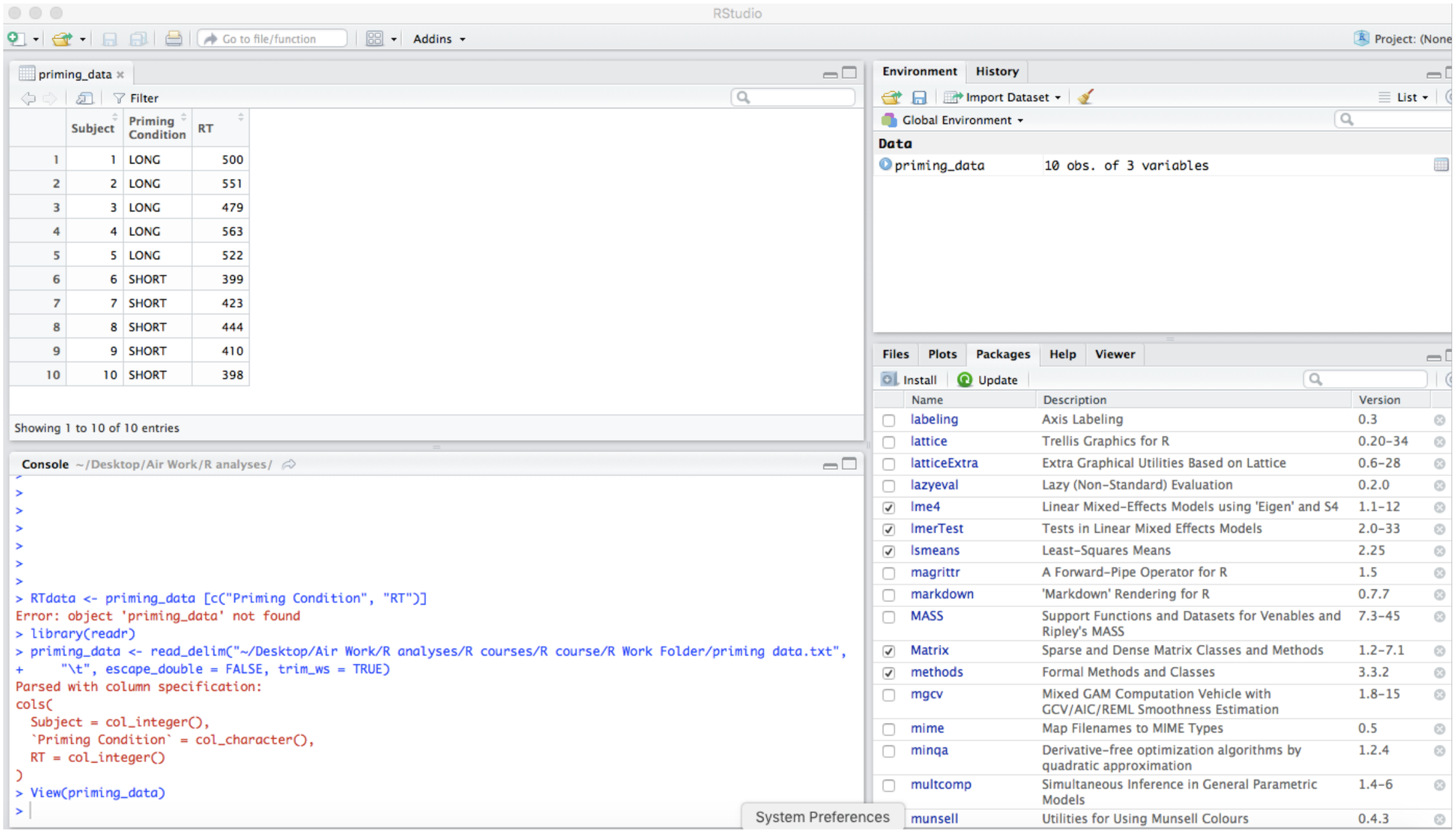
Import

Cancel

Can change here how the columns are delimited.

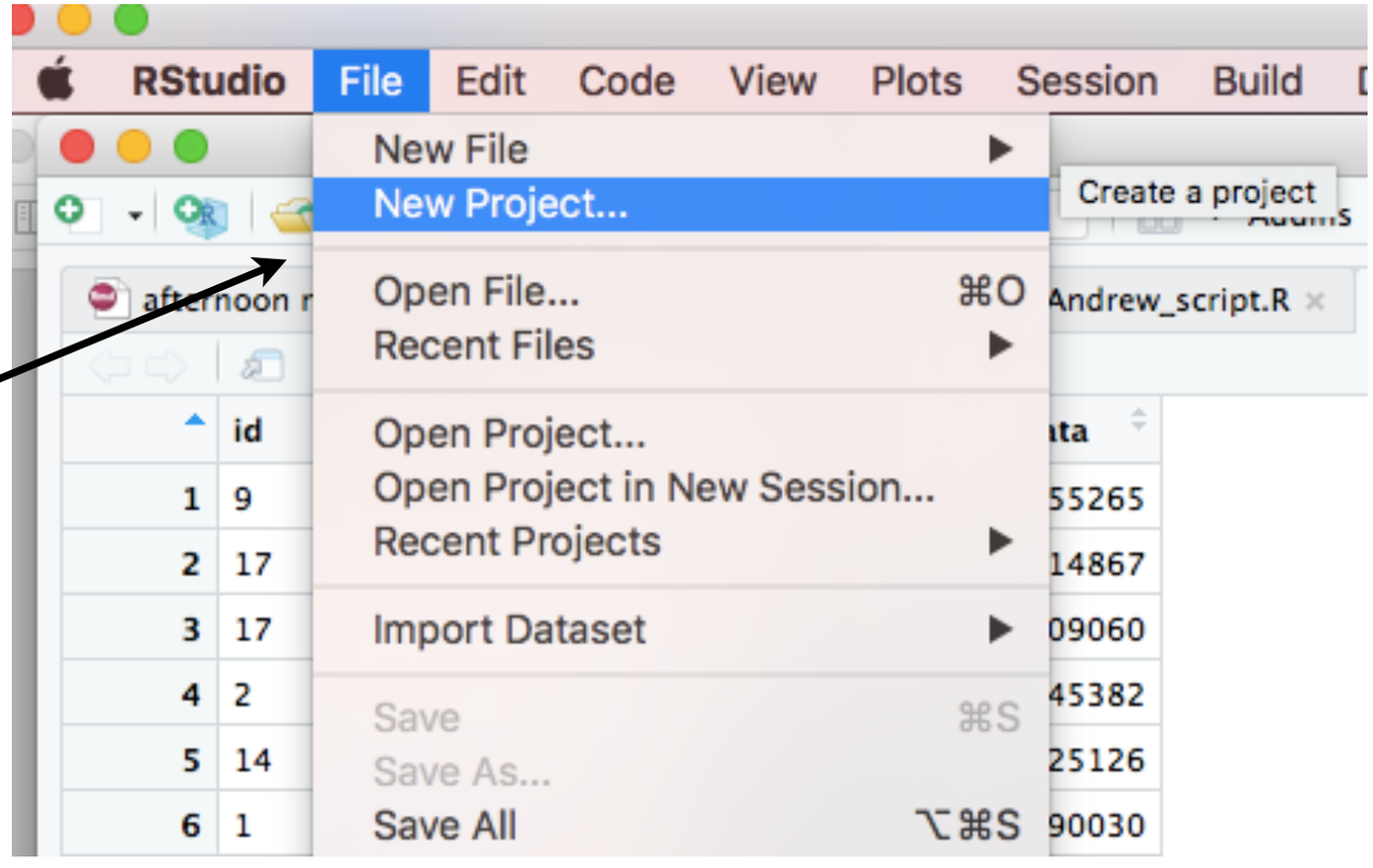
This is the code that corresponds to what you're getting RStudio to do.

- RStudio now looks like this:



When Starting R

Always create a new project - this will keep all your files nice and organised.

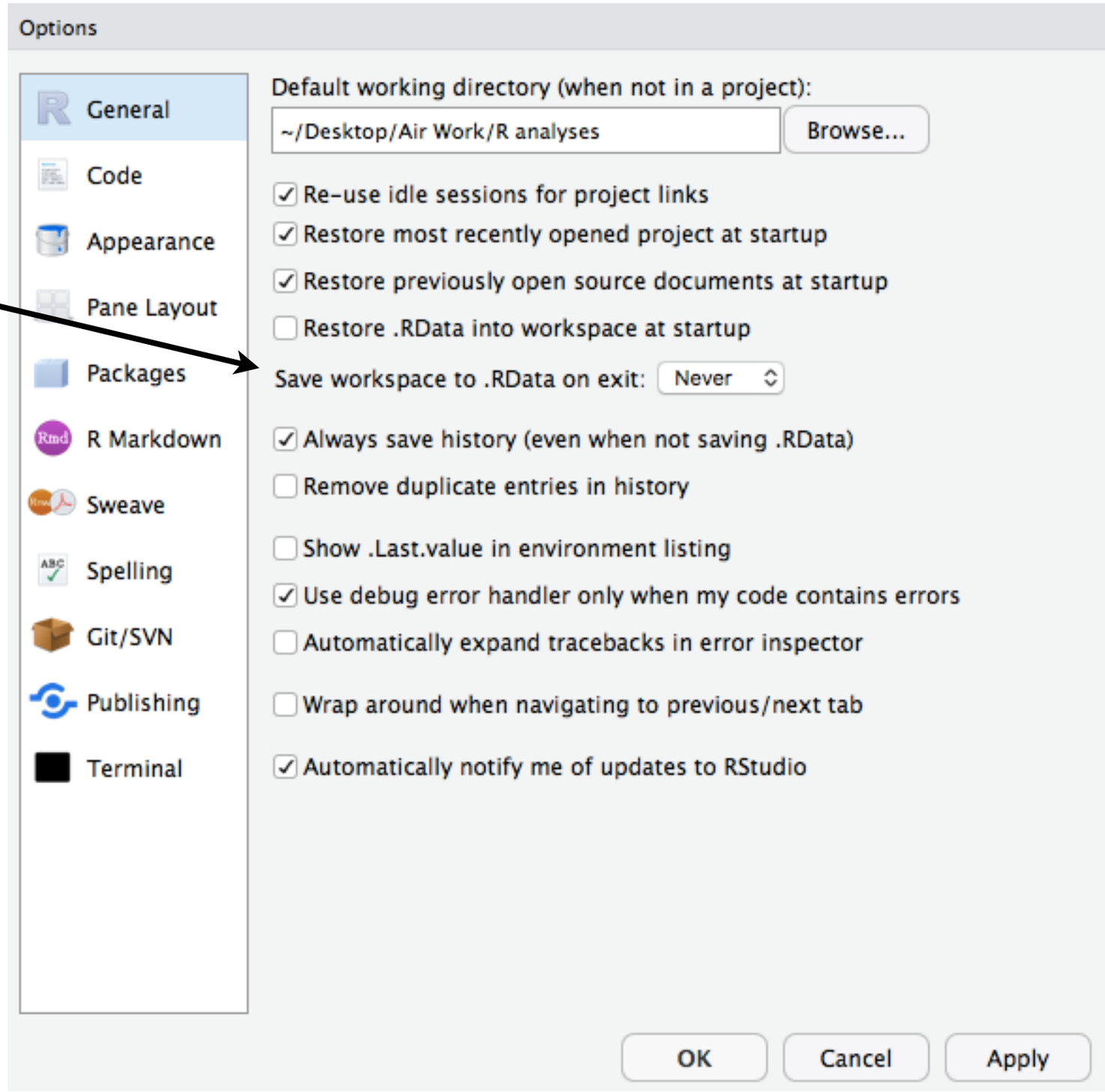


When you create an R Project in a folder, it's a good idea to keep that folder itself nicely organised:

```
name_of_project
|--name_of_project.Rproj
|--raw_data
    |--data.csv
|--tidied_data
    |--tidy_data.csv
|--R_scripts
    |--data_tidy.R
    |--data_vis.R
    |--analysis.R
|--markdown_scripts
    |--analysis.Rmd
|--output_reports
    |--analysis.html
    |--analysis.pdf
```

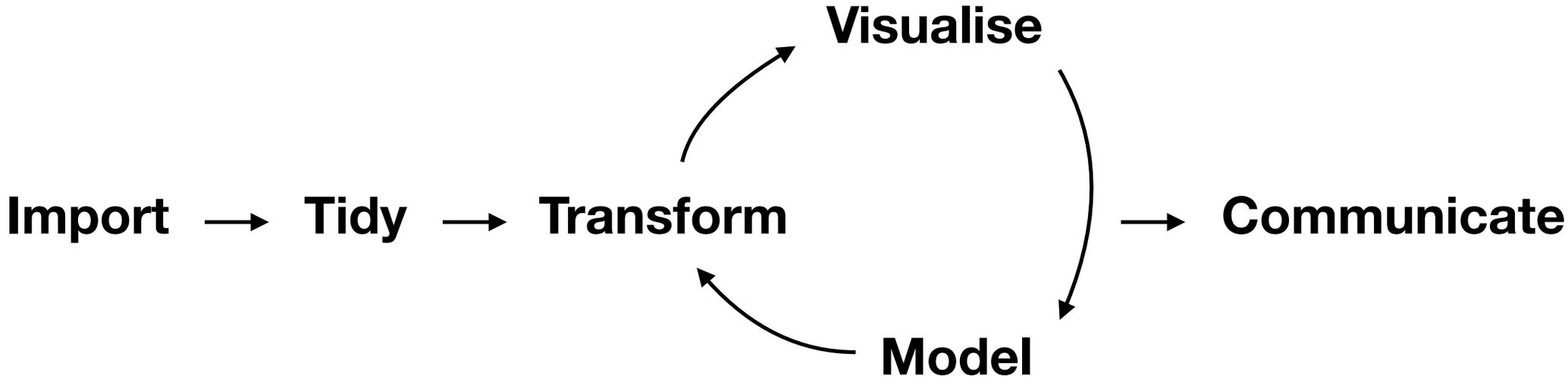
When Starting R

Under preferences, make sure you <u>uncheck</u> the saving workspace option…



Setting up R

Analysis Workflow in the Tidyverse (Garrett Grolemund and Hadley Wickham) - from Data to Write-up

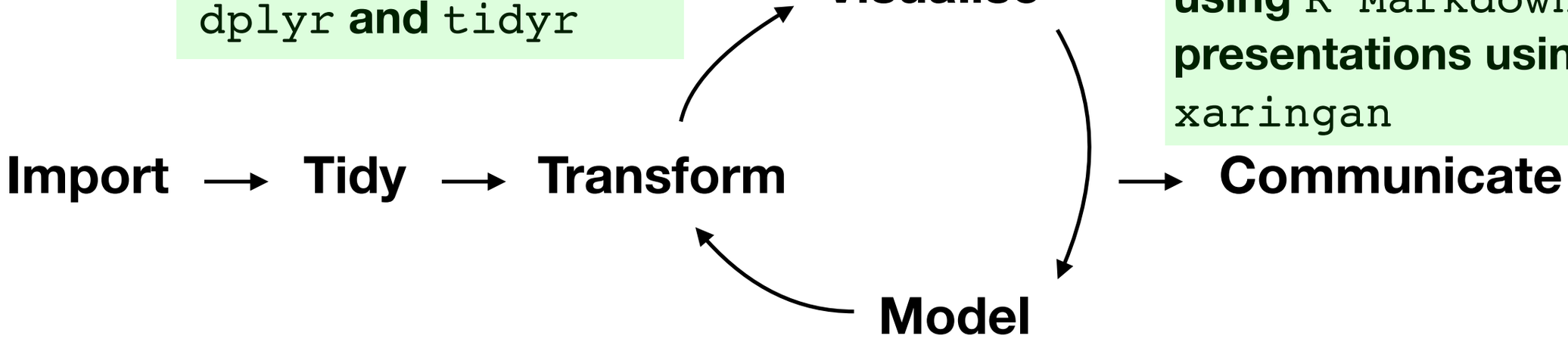


Workflow

Data plots using `ggplot()`

Tidying and transforming using `dplyr` **and** `tidyr`

html, pdf and doc reports using `R Markdown`**, presentations using** `xaringan`



ANO(C)VA using `afex()` **and** `aov()`
Linear regression using `lm()` **and** `step()`
(Generalised) linear mixed models using `lmer()` **and** `glmer()`

Packages in R

R has a number of functions already built in. These are part of "base R". For much of what we do we need to load particular packages to let us use additional functions. We do this by:

```
> install.packages("packagename")
```

```
> library(packagename)
```

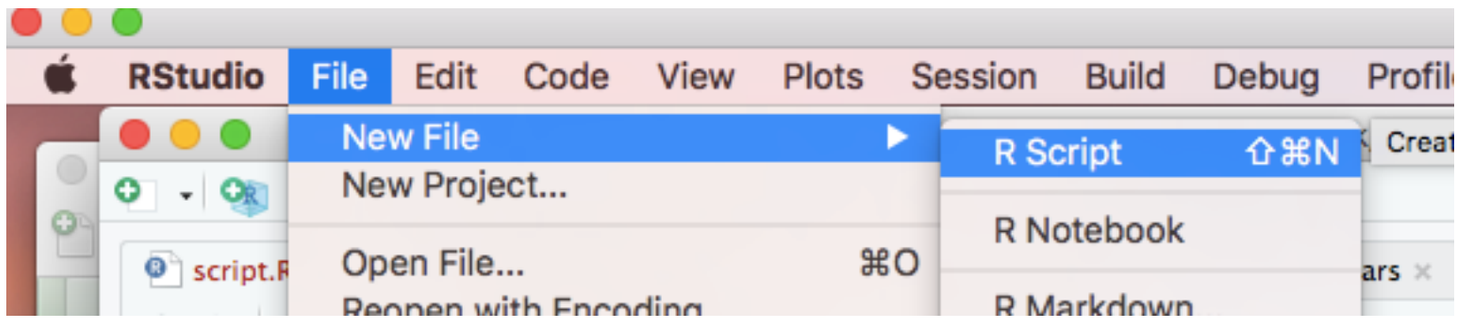
Every time you start R afresh, you need to load the packages you will use by using the `library` function. If you see notation like the following, it means we're using a function (e.g., `summarise`) from within a package (e.g., `dplyr`)

```
dplyr::summarise
```

You don’t need to re-invent the wheel…

- For any problem you want to solve, chances are others have had the same problem, solved it and have created an R package to do exactly what you want…
- One of the many great things about R is that you can add freely available packages to your library.
- ~15,000 R packages currently available
- The sites r-bloggers.com and rweekly.org are good places to find out about new packages.

Writing Scripts in R



Ultimately you will be writing scripts that will allow yourself and others to recreate your analysis just by running the script - the code at the start of the script will load the packages your analysis needs, then load your data, tidy, transform and visualise your data, and then code for your model…

You should write your code as a script, and use the Console window for single command instructions relatively rarely…

Writing Scripts in R

Scripts should have lots of comments, indicated by a # symbol - this helps others read your code, and also yourself when you look back at it later.

Scripts can be saved and uploaded to (e.g.) OSF, GitHub, or submitted as an electronic supplement alongside your journal submission. Even when journals don't require you to adhere to Open Science practices, it's a good idea to make your code and data available during the reviewing process - and publicly available once your paper is published.

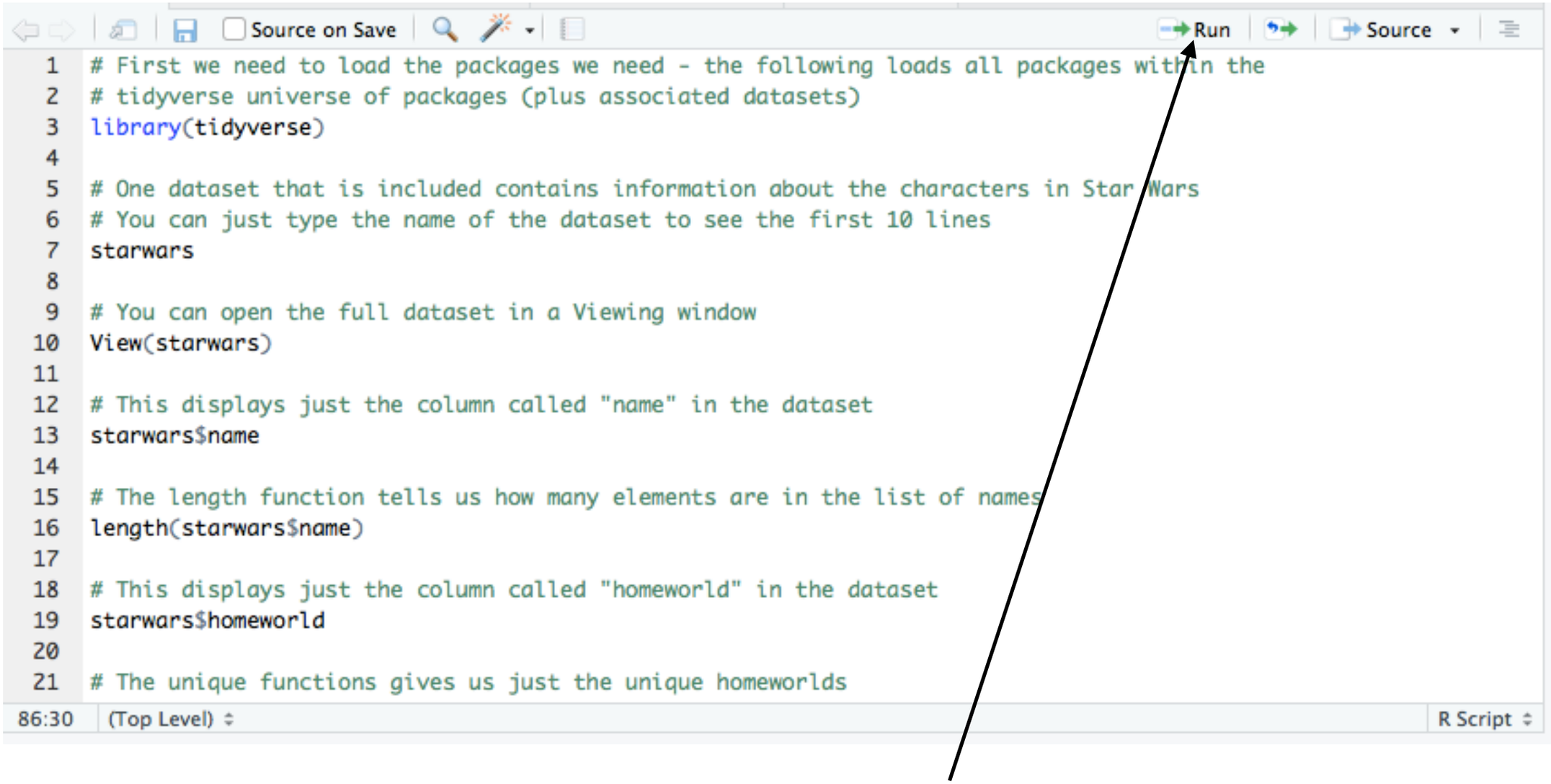
If you aren't sharing your data and code, you aren't engaged in generating reproducible or open research…

Source on Save | Run | Source

```r
# First we need to load the packages we need - the following loads all packages within the
# tidyverse universe of packages (plus associated datasets)
library(tidyverse)

# One dataset that is included contains information about the characters in Star Wars
# You can just type the name of the dataset to see the first 10 lines
starwars

# You can open the full dataset in a Viewing window
View(starwars)

# This displays just the column called "name" in the dataset
starwars$name

# The length function tells us how many elements are in the list of names
length(starwars$name)

# This displays just the column called "homeworld" in the dataset
starwars$homeworld

# The unique functions gives us just the unique homeworlds
```

86:30 (Top Level) R Script

Lots of comments explaining what each section of code does, and lots of white space.



Once a script is written, you can run the entire script by highlighting it all (CMD-A in OSX) and clicking on ‘Run’, or you can highlight a section and run only that. CMD-Return will also Run the highlighted code.

Style Guide

File names (both scripts and data files) should be meaningful:

`regression_model.R - Good`

`somemodel.R - Bad`

Variable names should be meaningful and in lowercase:

`age - Good`

`A1 - Bad`

`expt1_data - Good`

`Expiriment1datawithoutliersremoved - Bad`

Style Guide

Place spaces around operators like `+ , - , = , <-`

Whitespace used sensibly makes your code easier to read. Separate discrete sections of code in your script with a blank line.

When writing scripts, continue on a subsequent line rather than writing one very long line.

Comment, comment, comment…

```
# First we load our datafile.
```

https://style.tidyverse.org

R Style Guide

Welcome

I Analyses

1 Files

1.1 Names

1.2 Organisation

1.3 Internal structure

2 Syntax

2.1 Object names

2.2 Spacing

2.3 Argument names

2.4 Code blocks

2.5 Long lines

2.6 Assignment

2.7 Semicolons

2.8 Quotes

2.9 Comments

3 Functions

3.1 Naming

The tidyverse style guide

Hadley Wickham

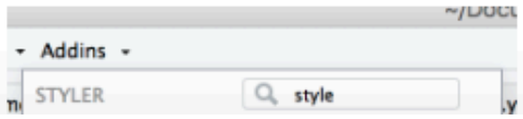
Welcome

Good coding style is like correct punctuation: you can manage without it, butitsuremakesthingseasiertoread. This site describes the style used throughout the tidyverse. It was derived from Google's original R Style Guide - but Google's current guide is derived from the tidyverse style guide.

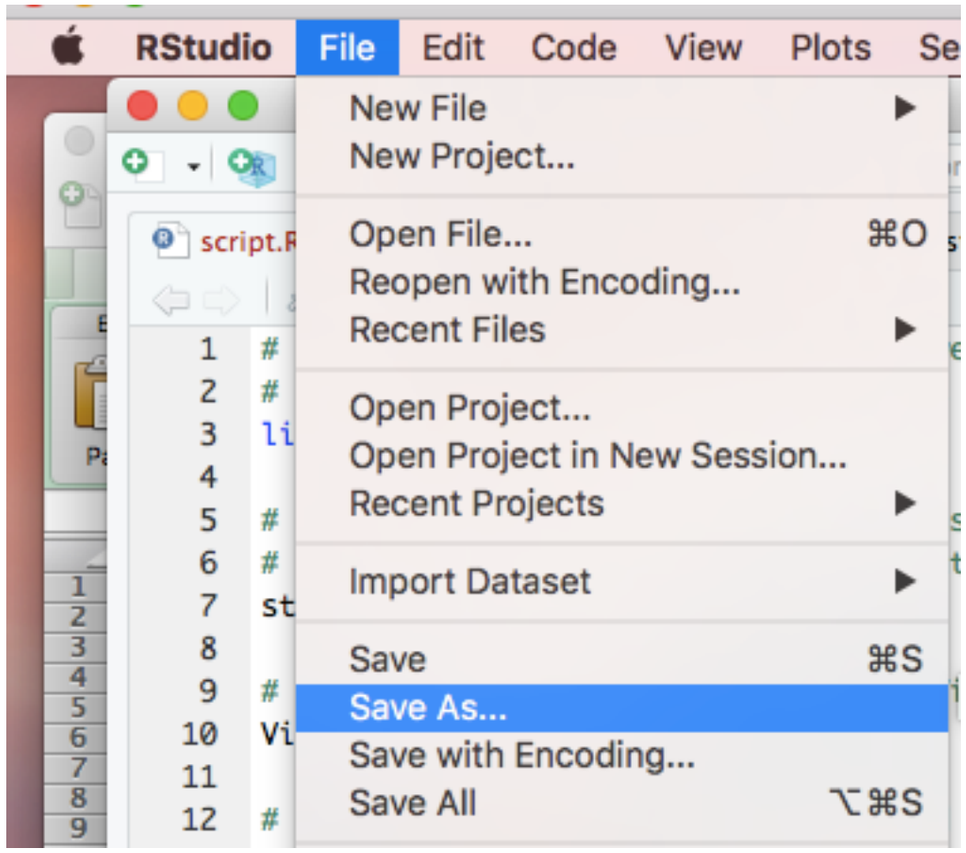
All style guides are fundamentally opinionated. Some decisions genuinely do make code easier to use (especially matching indenting to programming structure), but many decisions are arbitrary. The most important thing about a style guide is that it provides consistency, making code easier to write because you need to make fewer decisions.

Two R packages support this style guide:

- styler allows you to interactively restyle selected text, files, or entire projects. It includes an RStudio add-in, the easiest way to re-style existing code.



Once your script is finished, don't forget to save it...



Let’s run through a script…

Tidyverse

Packages Articles Learn Help Contribute

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

Tidyverse packages

- The Tidyverse contains a number of packages, all containing functions that are designed to ‘play well’ with each other. Packages include `ggplot2`, `dplyr`, and `tidyr`.

- You can load each package separately with (e.g.)

```
> library(ggplot2)
```

- or load all tidyverse packages with

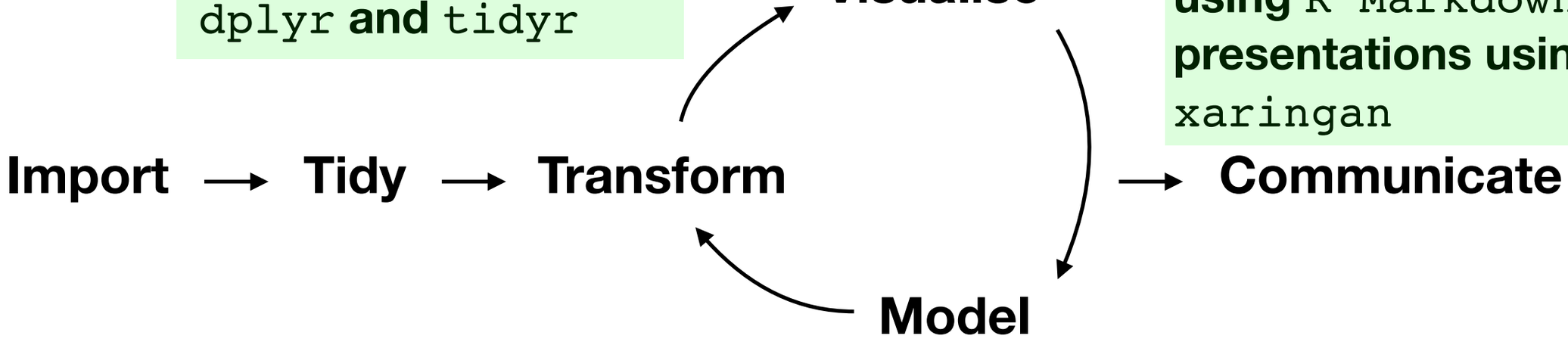
```
> library(tidyverse)
```

Workflow

Data plots using `ggplot()`

Tidying and transforming using `dplyr` **and** `tidyr`

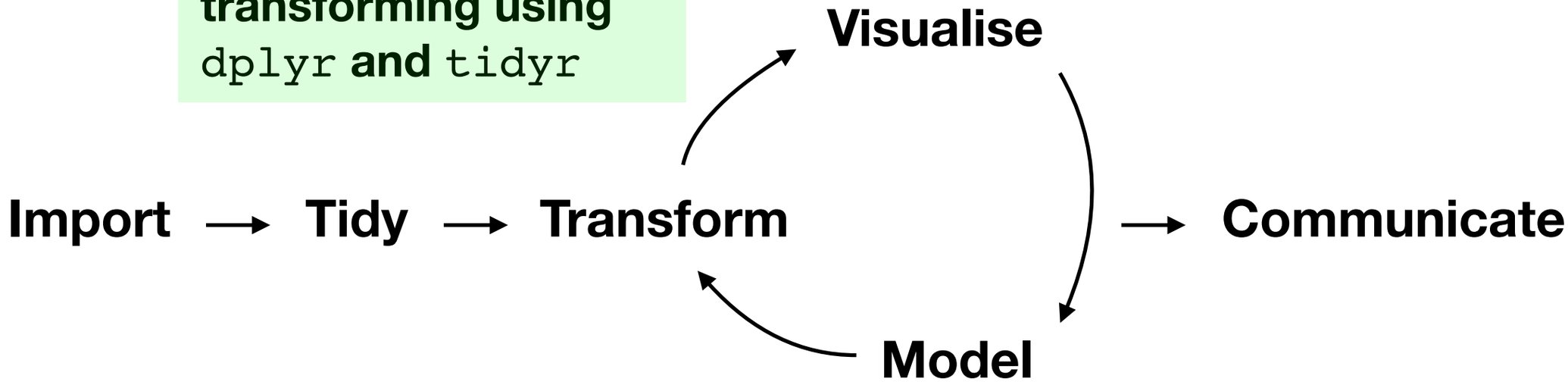
html, pdf and doc reports using `R Markdown`**, presentations using** `xaringan`



ANO(C)VA using `afex()` **and** `aov()`
Linear regression using `lm()` **and** `step()`
(Generalised) linear mixed models using `lmer()` **and** `glmer()`

Workflow

Tidying and transforming using `dplyr` **and** `tidyr`



Let's get ready to code...

On your computer, fire up RStudio and install the tidyverse:

```
> install.packages("tidyverse")
```

Then create a new Project in a new folder, and fire up a new script...

On the first line of your script type:

```
library(tidyverse)
```

Then run the script...

Let's now load two datasets that I've created:

```
data1 <- read_csv("https://bit.ly/31Te6HQ")

dataRT <- read_csv("https://bit.ly/2ZflWOr")
```

Tidying and Transforming Data

Imagine we have two rectangular datasets - one (called `data1`) contains a large number of records of individual participants with measures of Working Memory, IQ, and reading comprehension.

If we type into the Console:

```
> data1
```

the data frame is displayed like this…

```
> data1
# A tibble: 10,000 x 4
      id    wm    iq  comp
   <int> <int> <int> <int>
 1     1    43    72    16
 2     2    51   109    18
 3     3    55   107    18
 4     4    38   102    20
 5     5    52   121    17
 6     6    52    92    16
 7     7    47    68    21
 8     8    47    97    23
 9     9    47    93    22
10    10    45   101    17
# … with 9,990 more rows
```

To get more information about the structure of our data frame we can type:

```
> str(data1)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame': 10000 obs. of  4 variables:
 $ id  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ wm  : int  43 51 55 38 52 52 47 47 47 45 ...
 $ iq  : int  72 109 107 102 121 92 68 97 93 101 ...
 $ comp: int  16 18 18 20 17 16 21 23 22 17 ...
```

So we have 10,000 observations with 4 variables associated with each observation - all of them of type integer.

If you ever need help about a function (e.g. str), just type:

```
>?str
```

or

```
>help(str)
```

in the Console window.

Imagine that 48 of these 10,000 people also took part in a reading time experiment and we have their reading data (called `dataRT`) for Simple Sentence and Complex Sentence reading conditions:

```
> str(dataRT)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':  48 obs. of  3 variables:
 $ id              : int  6400 457 8291 4998 2579 9122 1138 5138 5244 3160 ...
 $ simple_sentence : int  1902 1797 2080 1856 1997 1868 2154 1933 1900 1929 ...
 $ complex_sentence: int  2341 2503 2731 2375 2177 2284 2441 2349 2371 2372 ...
```

We are maybe interested in analysing the data of these 48 people in the data frame called `dataRT` but covarying out the effect of IQ captured in our data frame called `data1`.

Problem - how can we combine these two data frames so that we end up with one data frame of 48 people, their reading times plus their individual difference measures?

Manually, in Excel we could open the two data frames as spreadsheets and cut and paste cases where the id number matches…

Probably ok for 48 participants, but what if you had 200 or 2,000?

In R, we can use the `inner_join` function from the `dplyr` package where we join the two data frames matched by ID.

```
> dataRT_all <- inner_join(data1, dataRT, by = (c("id")))
> dataRT_all
      id wm  iq comp simple_sentence complex_sentence
1     95 47  94   19            2154              2441
2    400 45 118   18            1824              2456
3    457 42 100   22            1857              2324
4   1138 41  77   18            1902              2341
5   1587 54  67   21            1844              2320
6   1805 52 109   19            2224              2256
7   1864 57 111   19            1880              2391
8   2006 44 110   19            2091              2456
9   2183 55 125   23            1926              2218
10 2318 51  91   21            1960              2440
```

We can use the assignment symbol <- to assign the output of this `inner_join` function to a new variable I'm calling `dataRT_all`. We can ask for the structure of this new data frame using the `str()` function:

```
> dataRT_all <- inner_join(data1, dataRT, by = (c("id")))
> str(dataRT_all)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':  48 obs. of  6 variables:
 $ id              : int  95 400 457 1138 1587 1805 1864 2006 2183 2318 ...
 $ wm              : int  47 45 42 41 54 52 57 44 55 51 ...
 $ iq              : int  94 118 100 77 67 109 111 110 125 91 ...
 $ comp            : int  19 18 22 18 21 19 19 19 23 21 ...
 $ simple_sentence : int  2154 1824 1857 1902 1844 2224 1880 2091 1926 1960 ...
 $ complex_sentence: int  2441 2456 2324 2341 2320 2256 2391 2456 2218 2440 ...
```

So we have created a new data frame of 48 participants consisting of their reading times and their individual difference measures from two separate (and different sized) data frames…with one line of code…

Now imagine we find the distributions of reading times for our two conditions are positively skewed (and we discover the residuals are non-normal). We could log transform these two columns and have two new columns in our data frame - let's call them `log_simple` and `log_complex`. We can use the `mutate` function in the `dplyr` package to create two new columns.

```
> data_transformed <- mutate(dataRT_all, log_simple = log(simple_sentence),
log_complex = log(complex_sentence))
> data_transformed
# A tibble: 48 x 8
      id    wm    iq  comp simple_sentence complex_sentence log_simple log_complex
   <int> <int> <int> <int>           <int>             <int>      <dbl>       <dbl>
 1    95    47    94    19            1960              2440       7.58        7.80
 2   400    45   118    18            2186              2200       7.69        7.70
 3   457    42   100    22            1797              2503       7.49        7.83
 4  1138    41    77    18            2154              2441       7.68        7.80
 5  1587    54    67    21            1936              2395       7.57        7.78
 6  1805    52   109    19            1864              2560       7.53        7.85
 7  1864    57   111    19            1930              2540       7.57        7.84
 8  2006    44   110    19            2230              2267       7.71        7.73
 9  2183    55   125    23            1857              2324       7.53        7.75
10  2318    51    91    21            1918              2739       7.56        7.92
# … with 38 more rows
```

Perhaps we have a reason to exclude a particular participant - number 2006 for example. We can use the filter function in `dplyr` to keep those participants where the ID number does <u>not</u> equal 2006.

```
filtered_data <- filter(data_transformed, id != 2006)
```

!= stands for "not equal to"- here are other useful logical operators in R:

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to

We can now apply our logical vector to our `dataRT_all` data frame and create a new filtered data frame (which I am calling `filtered_data`):

```
> filtered_data <- filter(data_transformed, id != 2006)
> filtered_data
# A tibble: 47 x 8
      id    wm    iq  comp simple_sentence complex_sentence log_simple log_complex
   <int> <int> <int> <int>           <int>            <int>      <dbl>       <dbl>
 1    95    47    94    19            1960             2440       7.58        7.80
 2   400    45   118    18            2186             2200       7.69        7.70
 3   457    42   100    22            1797             2503       7.49        7.83
 4  1138    41    77    18            2154             2441       7.68        7.80
 5  1587    54    67    21            1936             2395       7.57        7.78
 6  1805    52   109    19            1864             2560       7.53        7.85
 7  1864    57   111    19            1930             2540       7.57        7.84
 8  2183    55   125    23            1857             2324       7.53        7.75
 9  2318    51    91    21            1918             2739       7.56        7.92
10  2324    43   120    20            1891             2426       7.54        7.79
# … with 37 more rows
```

We could then run an ANCOVA over the log transformed RTs while covarying out the individual participant effects…

Problem - imagine our data are in the wrong 'shape' - they are in Wide format (each row is one *participant*) but we need them in Long or Tidy format (each row is one *observation*).

In SPSS, most data will be in Wide format with each experimental condition its own column:

```
> dataRT
# A tibble: 48 x 3
      id simple_sentence complex_sentence
   <int>           <int>            <int>
 1  6400            1902             2341
 2   457            1797             2503
 3  8291            2080             2731
 4  4998            1856             2375
 5  2579            1997             2177
 6  9122            1868             2284
 7  1138            2154             2441
 8  5138            1933             2349
 9  5244            1900             2371
10  3160            1929             2372
# … with 38 more rows
```

For many analyses in R, data need to be in Long format with each row being one observation. So, we want to transform our `dataRT` data frame so it looks like this:

id	condition	rt
…	…	…

To do this we can use the `gather()` function in the `tidyr` package.

```
> data_long <- gather(dataRT, "condition", "rt", c("simple_sentence",
"complex_sentence"))
```

The first parameter is the name of the data frame we want to reshape, the second is the name of the new 'Key' column, the third is the name of the new value column and the fourth the names of the columns we want to collapse.

We can use this to create a new data frame called `data_long` which looks like this:

```
> data_long <- gather(dataRT, "condition", "rt", c("simple_sentence",
"complex_sentence"))
> data_long
# A tibble: 96 x 3
      id condition          rt
   <int> <chr>           <int>
 1  6400 simple_sentence  1902
 2   457 simple_sentence  1797
 3  8291 simple_sentence  2080
 4  4998 simple_sentence  1856
 5  2579 simple_sentence  1997
 6  9122 simple_sentence  1868
 7  1138 simple_sentence  2154
 8  5138 simple_sentence  1933
 9  5244 simple_sentence  1900
10  3160 simple_sentence  1929
# … with 86 more rows
```

And in reverse we can use the `spread()` function to go from Long to Wide data format:

```
> data_wide <- spread(data_long, "condition", "rt")
> data_wide
# A tibble: 48 x 3
      id complex_sentence simple_sentence
   <dbl>            <dbl>           <dbl>
 1    95             2441            2154
 2   400             2456            1824
 3   457             2324            1857
 4  1138             2341            1902
 5  1587             2320            1844
 6  1805             2256            2224
 7  1864             2391            1880
 8  2006             2456            2091
 9  2183             2218            1926
10  2318             2440            1960
# … with 38 more rows
```

We're now back to where we started with data in Wide format:

This is just a small example of functions in the `dplyr` and `tidyr` packages that allow you to tidy, transform, and reshape your data. All of your code for doing this should appear at the start of your analysis script so that others (and you in 5 years or 5 days time) can see exactly what you did.

This allows for fully reproducible data preparation in the first part of your analysis workflow (important for Open Science and reproducibility).

Generating Descriptives - using `dplyr`

- You can use the `group_by()` and `summarise()` functions in the `dplyr` package to generate descriptives.

- In the following example, we are also using the pipe operator `%>%` which passes a value into an expression or function call from left to right:

```
> data_long %>%
   group_by(condition) %>%
   summarise(Mean = mean(rt), Min = min(rt), Max = max(rt), SD = sd(rt))
# A tibble: 2 x 5
  condition          Mean   Min   Max    SD
  <chr>             <dbl> <int> <int> <dbl>
1 complex_sentence 2405.  2177  2739  132.
2 simple_sentence  1957.  1694  2356  147.
```

Tidying Up Some Real World Messy Data

Let's load another dataset that I created:

```
my_data <- read_csv("https://bit.ly/2KPZEe9")
```

Tidying Up Some Real World Messy Data

- We ran a reaction time experiment with 24 participants and 4 conditions - they are numbered 1-4 in our datafile.

```
> my_data
# A tibble: 96 x 3
   participant condition    rt
         <int>     <int> <int>
 1           1         1   879
 2           1         2  1027
 3           1         3  1108
 4           1         4   765
 5           2         1  1042
 6           2         2  1050
 7           2         3   942
 8           2         4   945
 9           3         1   943
10           3         2   910
# … with 86 more rows
```

- But actually it was a repeated measures design where we had one factor (Prime Type) with two levels (A vs. B) and a second factor (Target Type) with two levels (A vs. B)

- We want to recode our data frame so it better matches our experimental design.

- First we need to recode our 4 conditions like this:

```
# Recode condition columns follows:
# Condition 1 = prime A, target A
# Condition 2 = prime A, target B
# Condition 3 = prime B, target A
# Condition 4 = prime B, target B

my_data <- my_data %>%
            mutate(condition = recode(condition,
            "1" = "primeA_targetA",
            "2" = "primeA_targetB",
            "3" = "primeB_targetA",
            "4" = "primeB_targetB"))
```

- Now our data frame looks like this:

```
> my_data
# A tibble: 96 x 3
   participant condition          rt
         <int> <chr>           <int>
 1           1 primeA_targetA    879
 2           1 primeA_targetB   1027
 3           1 primeB_targetA   1108
 4           1 primeB_targetB    765
 5           2 primeA_targetA   1042
 6           2 primeA_targetB   1050
 7           2 primeB_targetA    942
 8           2 primeB_targetB    945
 9           3 primeA_targetA    943
10           3 primeA_targetB    910
# … with 86 more rows
```

- We then need to separate out our Condition column into two - one for our first factor (Prime), and one for our second factor (Target).

```
> my_data <- separate(my_data, col = "condition", into = c("prime",
"target"), sep = "_")
> my_data
# A tibble: 96 x 4
   participant prime  target     rt
         <int> <chr>  <chr>   <int>
 1           1 primeA targetA   879
 2           1 primeA targetB  1027
 3           1 primeB targetA  1108
 4           1 primeB targetB   765
 5           2 primeA targetA  1042
 6           2 primeA targetB  1050
 7           2 primeB targetA   942
 8           2 primeB targetB   945
 9           3 primeA targetA   943
10           3 primeA targetB   910
# … with 86 more rows
```

- This is looking good - we now have our two factors coded separately and our data are in tidy format (i.e., one observation per row).

- How long would this have taken you in Excel? Would it have been reproducible?

- Perhaps we want to go from the data in long format, to wide format.

```
> my_data <- unite(my_data, col = "condition", c("prime", "target"), sep = "_")
> wide_data <- spread(my_data, key = "condition", value = "rt")
> wide_data
# A tibble: 24 x 5
   participant primeA_targetA primeA_targetB primeB_targetA primeB_targetB
         <int>          <int>          <int>          <int>          <int>
 1           1            879           1027           1108            765
 2           2           1042           1050            942            945
 3           3            943            910            952            900
 4           4            922           1006           1095            988
 5           5            948            908            916           1241
 6           6           1013            950            955           1045
 7           7            930            855           1057            897
 8           8            998            906           1110            952
 9           9            929            949            837            883
10          10            781            865            970            953
# … with 14 more rows
```

- No matter what format your data are in originally, you can use functions from the `dplyr` and `tidyr` packages to quickly get it into whatever format you need for analysis.

Or using the pipe…

- We could alternatively have used the %>% operator to combine all the last few operations which would have avoided the need to create temporary variables.

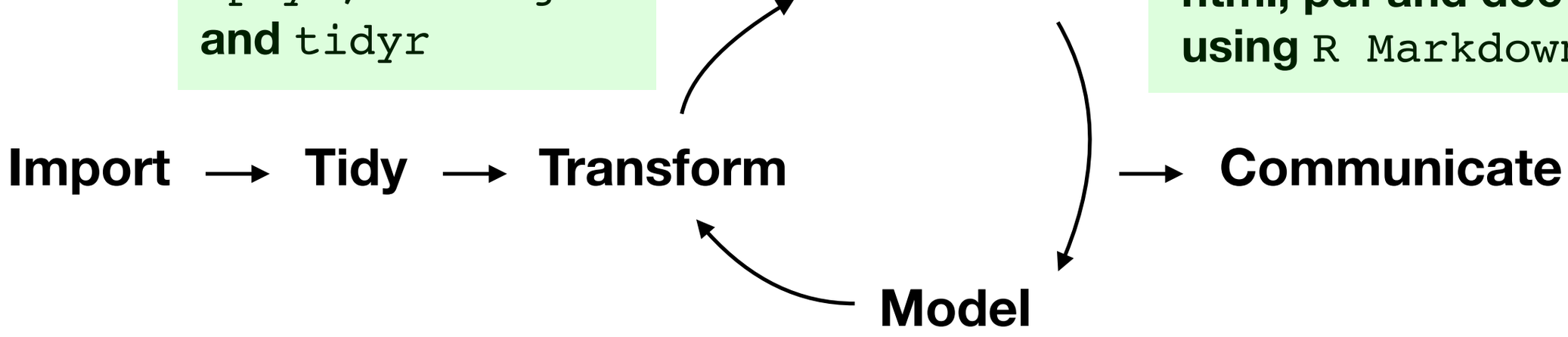
```
my_data %>%
   unite(col = "condition", c("prime", "target"), sep = "_") %>%
   spread(key = "condition", value = "rt")
```

- Take `my_data` and then unite and then spread…

A Reproducible Workflow

Data vis using `ggplot()`

Tidying and transforming using `dplyr, stringr` **and** `tidyr`

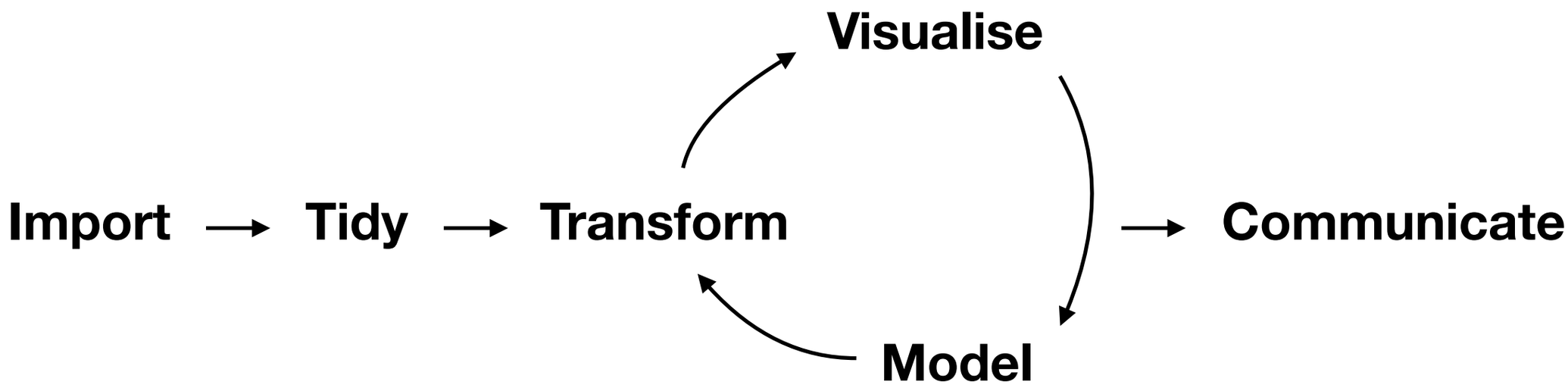


html, pdf and doc reports using `R Markdown`

ANO(C)VA using `afex()` **and** `aov()`
Linear regression using `lm()` **and** `step()`
(Generalised) linear mixed models using `lmer()` **and** `glmer()`

A Reproducible Workflow

Data vis using `ggplot()`

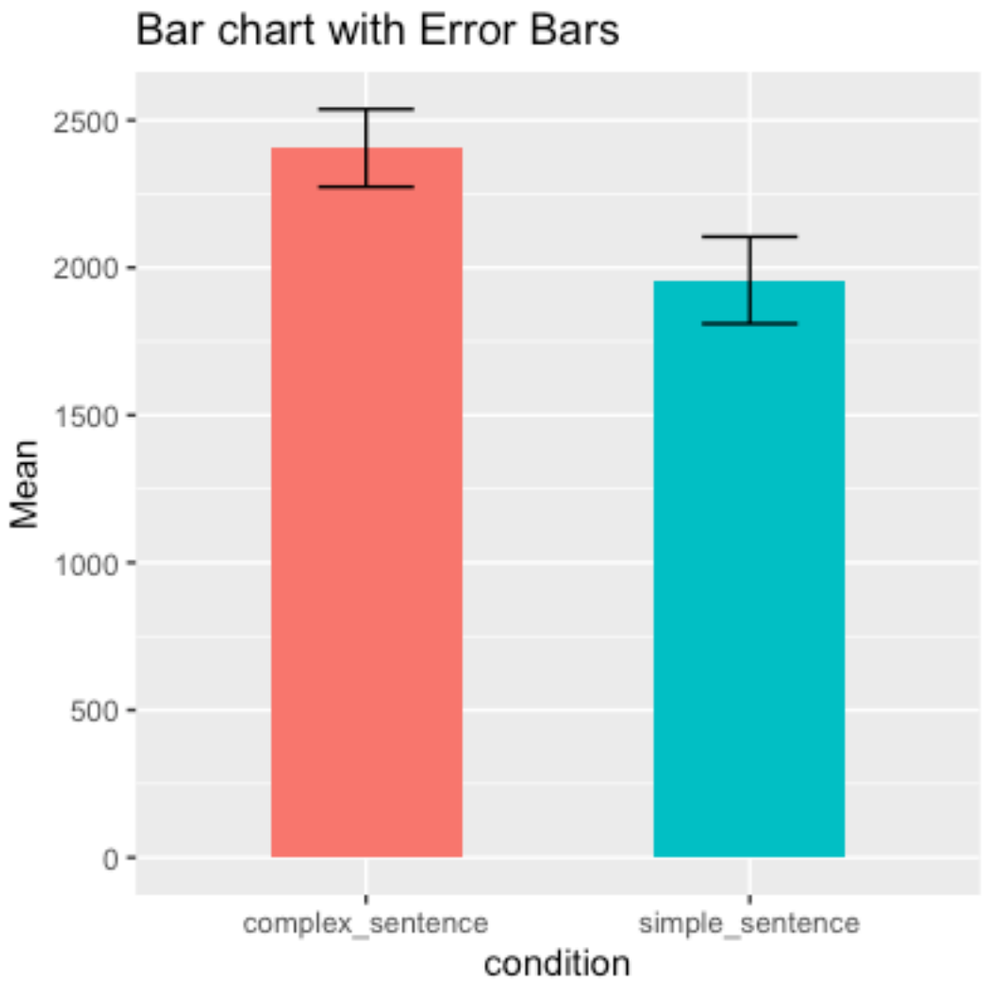


Visualising Your Data

- R has a number of in built (base) graphics functions, but you're more likely to use functions from within the `ggplot2` packages. `ggplot2` is part of the `tidyverse` so if you have used `library(tidyverse)` then `ggplot2` will already be loaded.

```
> library(ggplot2)
```

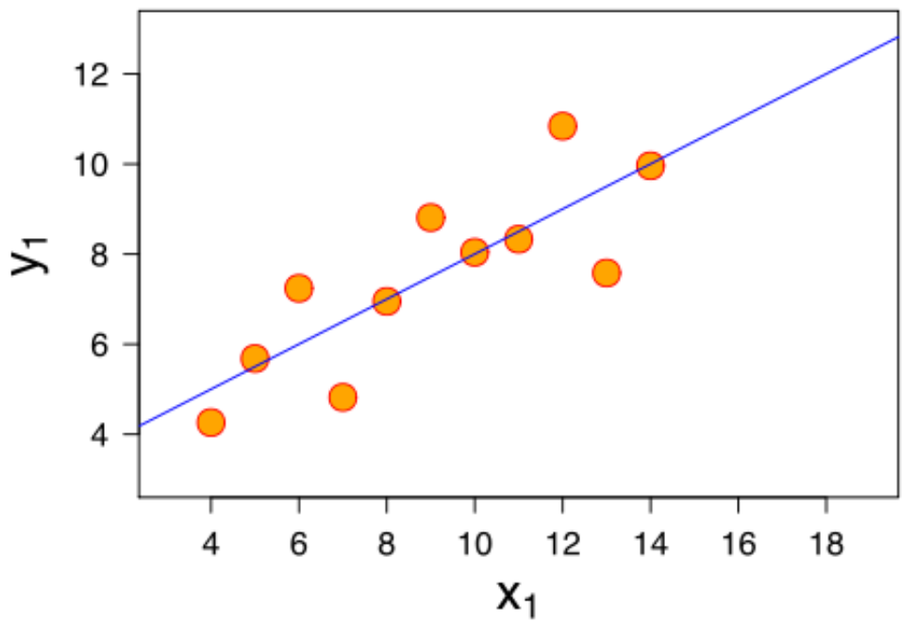
Bar Graphs (bleurgh!)

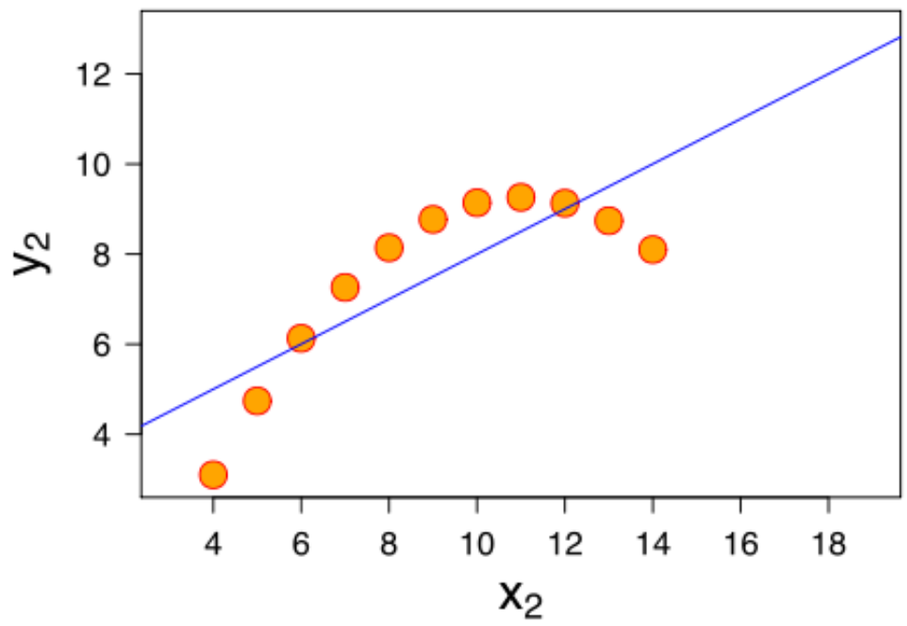


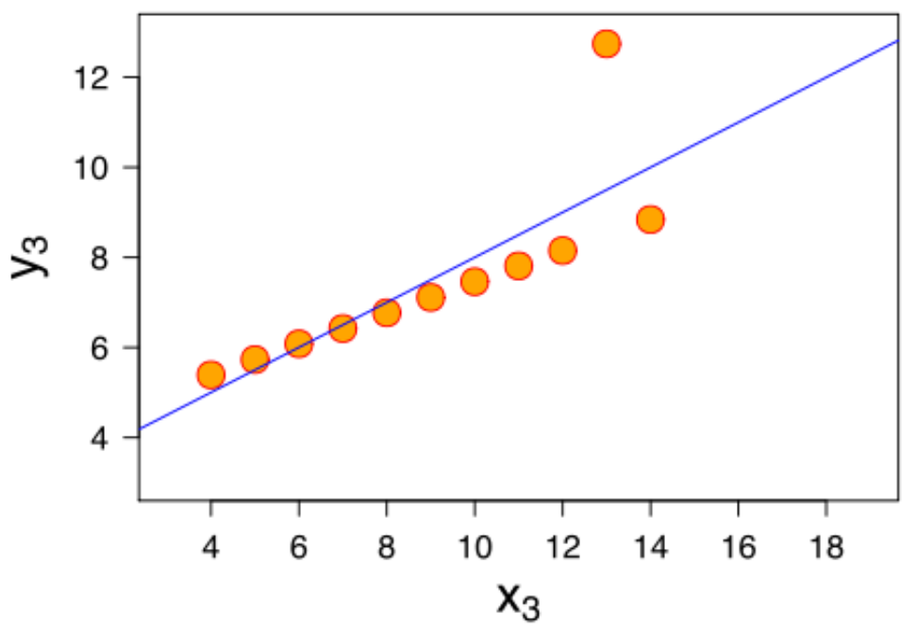
Bar graphs tend to be quite limited in terms of what they communicate. Here they communicate the means for levels of a factor and information about variance. But they don't tell us anything about the *distribution* of the data.

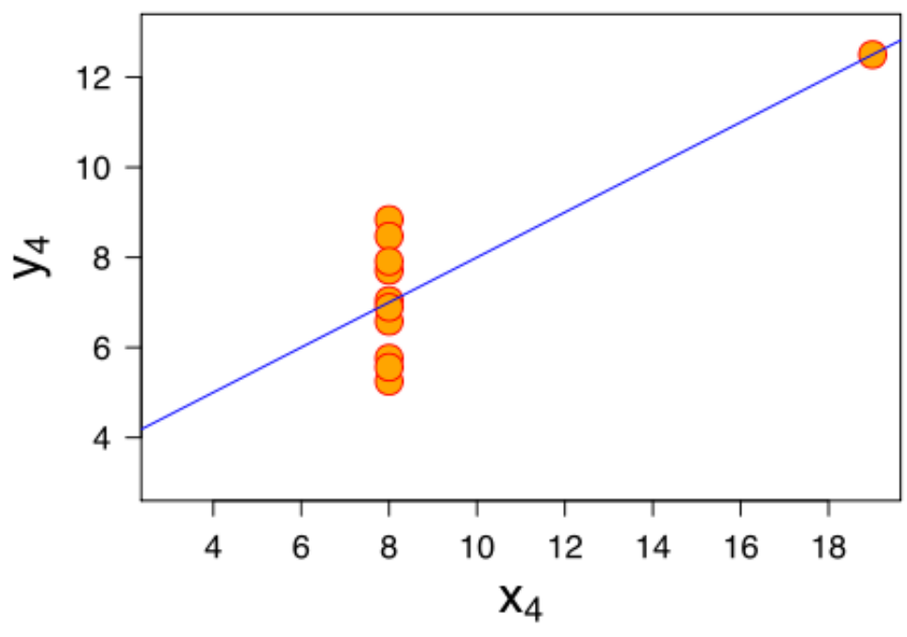
```
data_summ <- data_long %>% group_by(condition) %>% summarise(Mean = mean(rt), sd = sd(rt))

ggplot(data_summ, aes (x = condition, y = Mean, group = condition,
                       fill = condition, ymin = Mean - sd, ymax = Mean + sd)) +
  geom_bar(stat = "identity", width = .5) +
  geom_errorbar(width = .25) +
  ggtitle("Bar chart with Error Bars") +
  guides(fill = FALSE)
```

Anscombe's Quartet

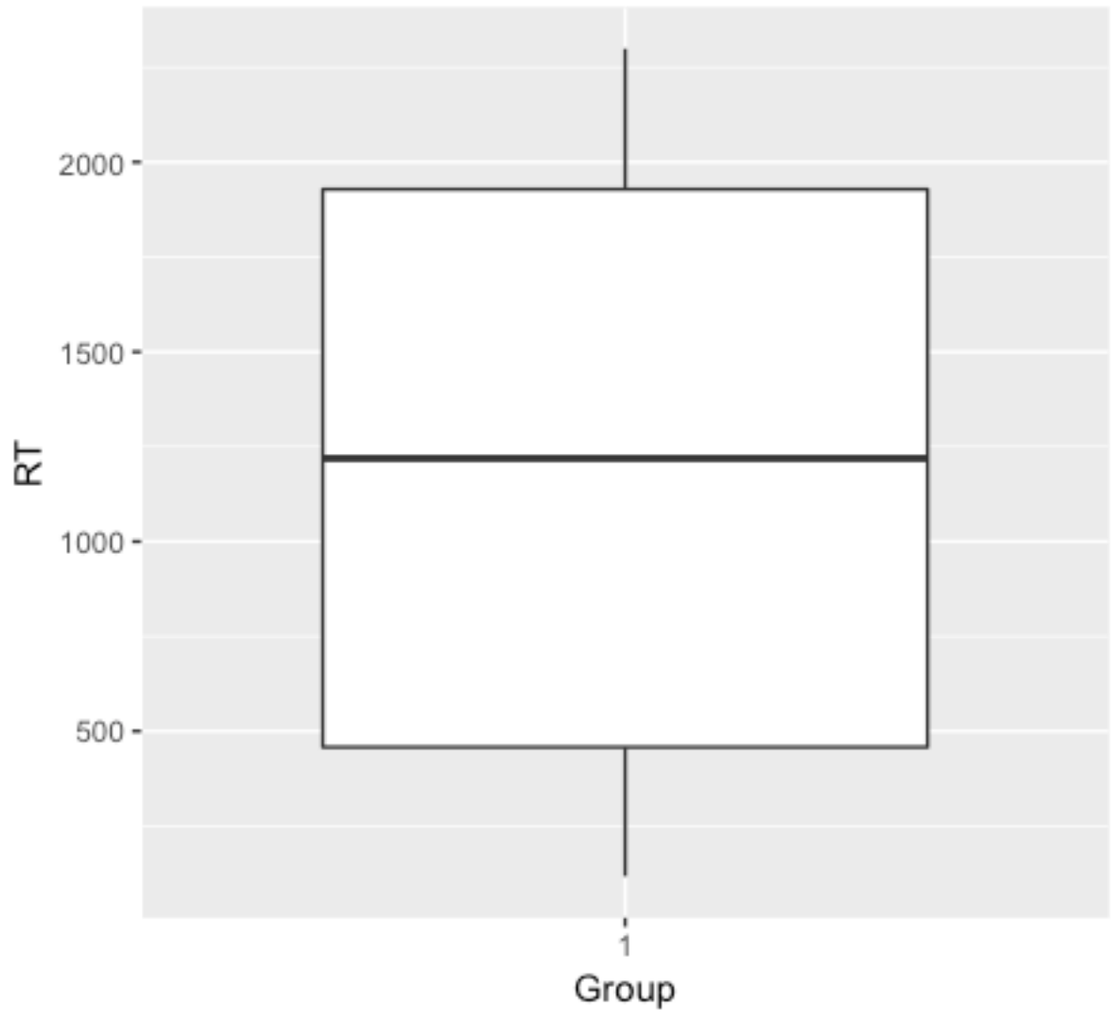






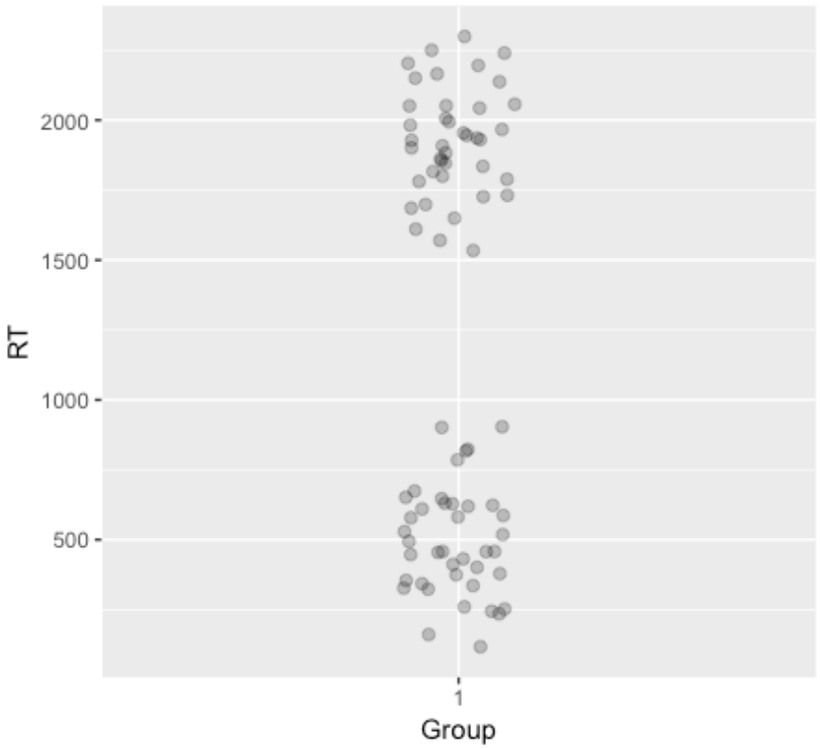


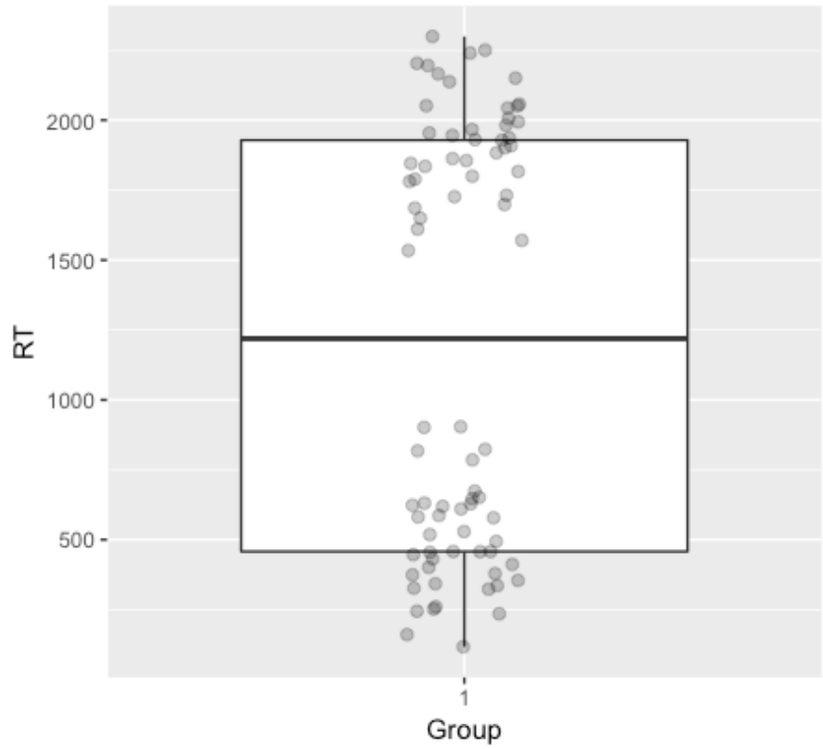
Plots Based on Aggregated Data Can Mislead…

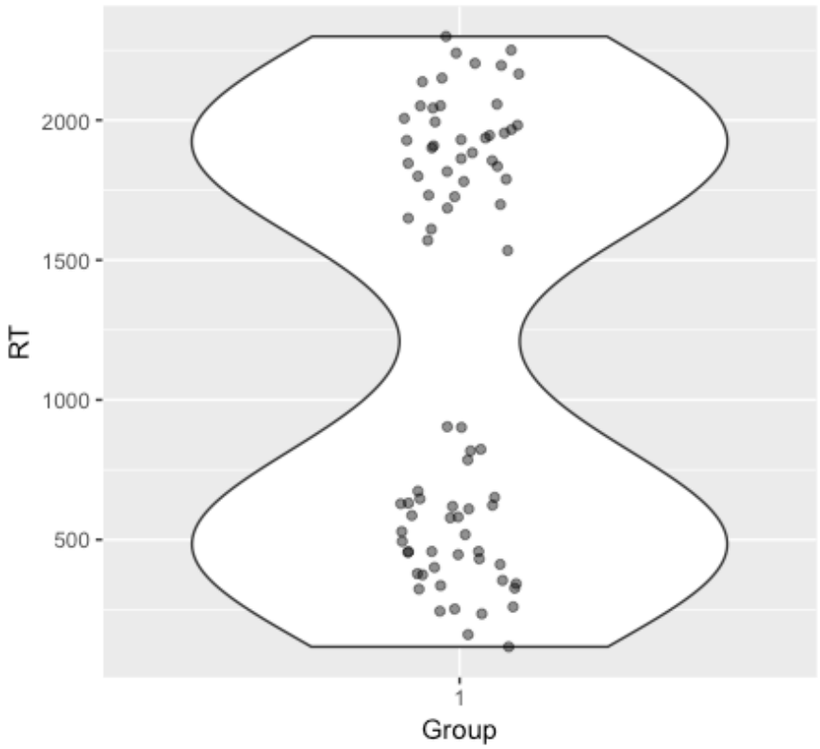


You might make one set of inferences based on this boxplot - maybe a median around 1,250 with the 25th and 75th percentiles being ~480 to ~1,980…

But look more closely at the actual data…

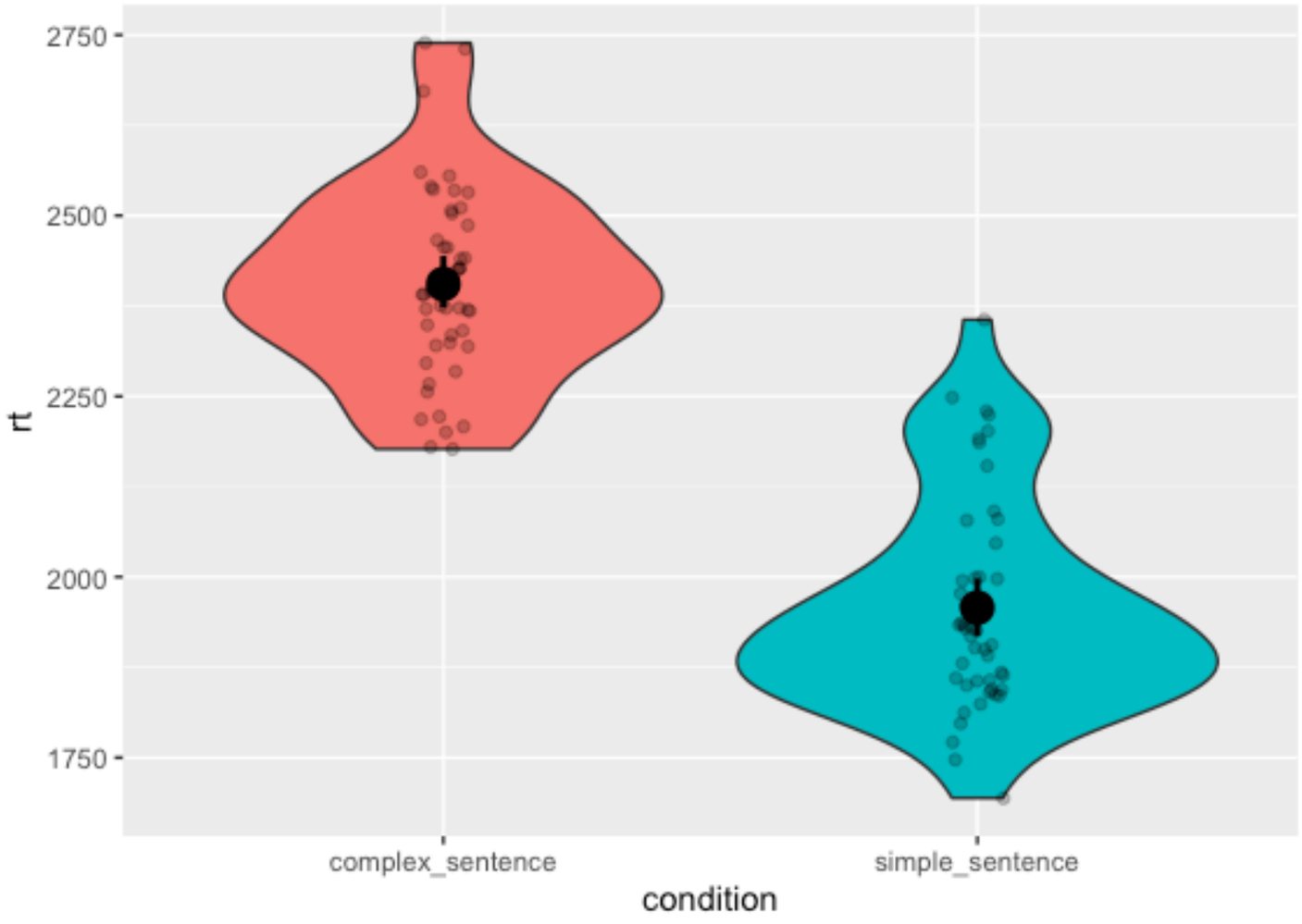






The data are clearly bimodal with no actual data point near the mean. **Distribution shape matters** and we need to capture that in our data visualisations.

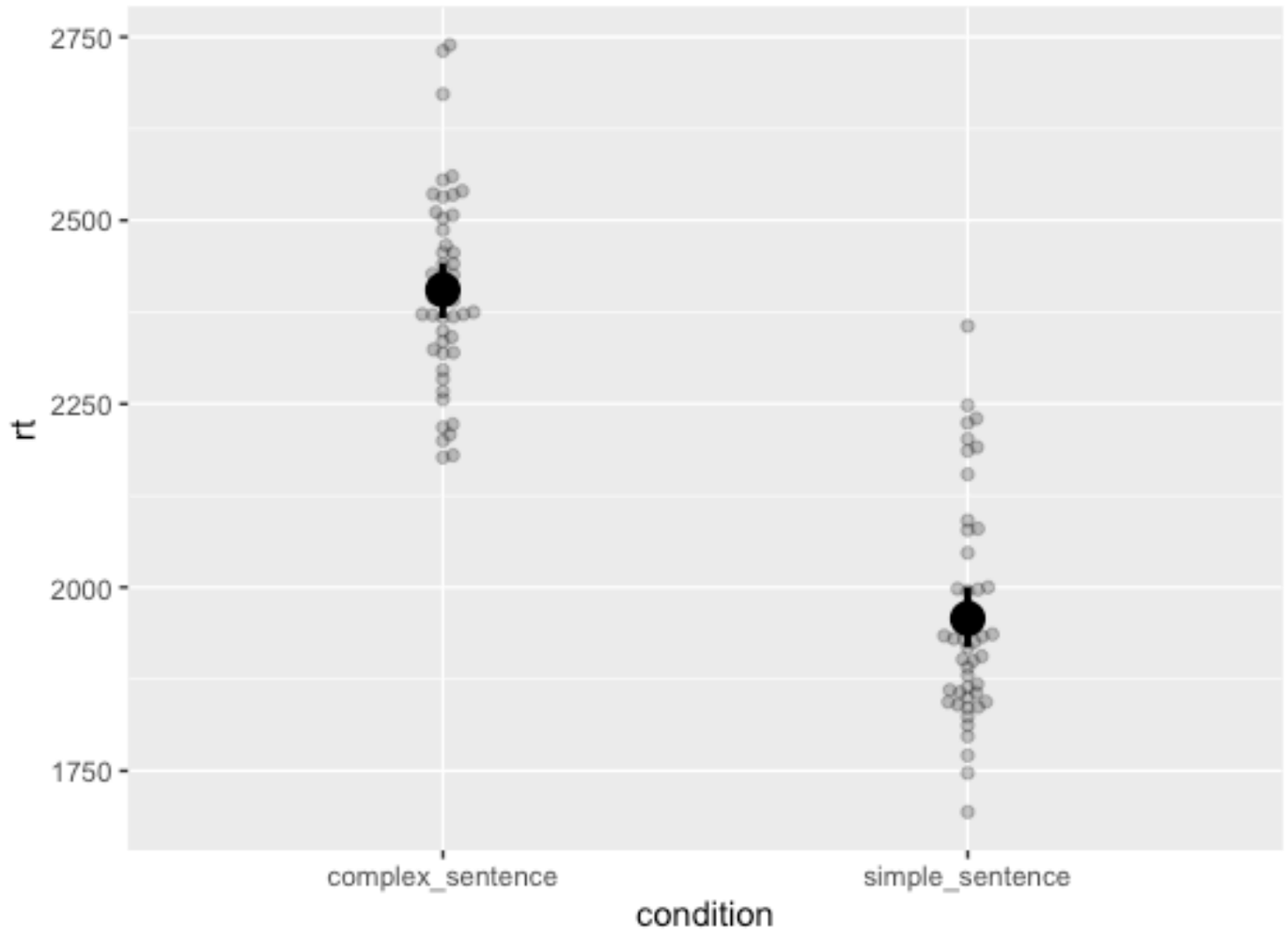
Violin Plots



Violin plots tell us about the distribution of the data. The width at any point corresponds to the *density* of the data at that value.

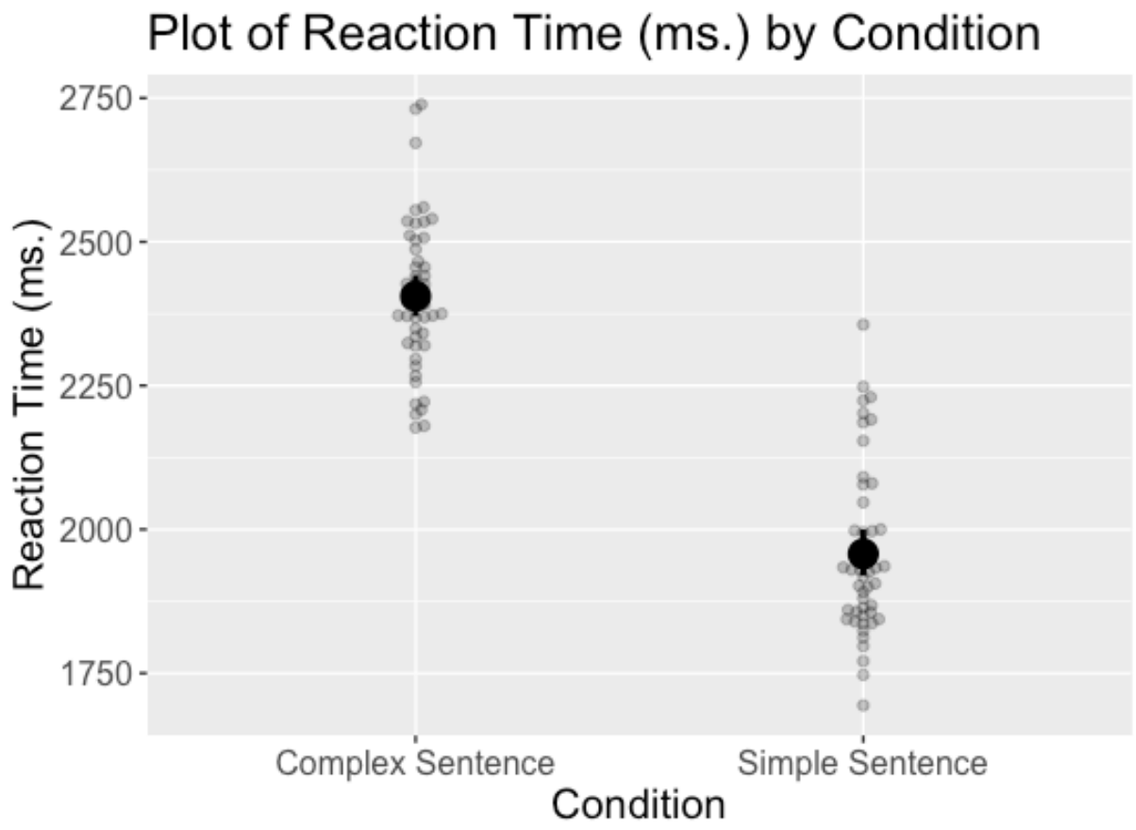
```
ggplot(data_long, aes(x = condition, y = rt,
    group = condition, fill = condition)) +
    geom_violin() +
    geom_jitter(alpha = .25, position = position_jitter(0.05)) +
    guides(colour = FALSE, fill = FALSE) +
    stat_summary(fun.data = "mean_cl_boot", colour = "black", size = 1)
```

Beeswarm Plots




```
ggplot(data_long, aes(x = condition, y = rt, group = condition, fill = condition)) +
  geom_beeswarm(alpha = .25) +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary(fun.data = "mean_cl_boot", colour = "black", size = 1)
```

Tidying up our labels

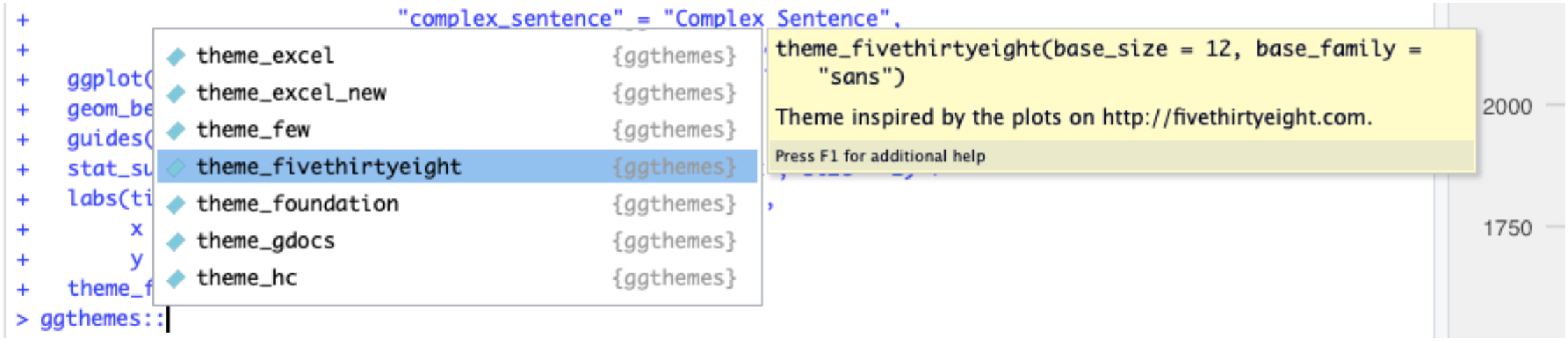


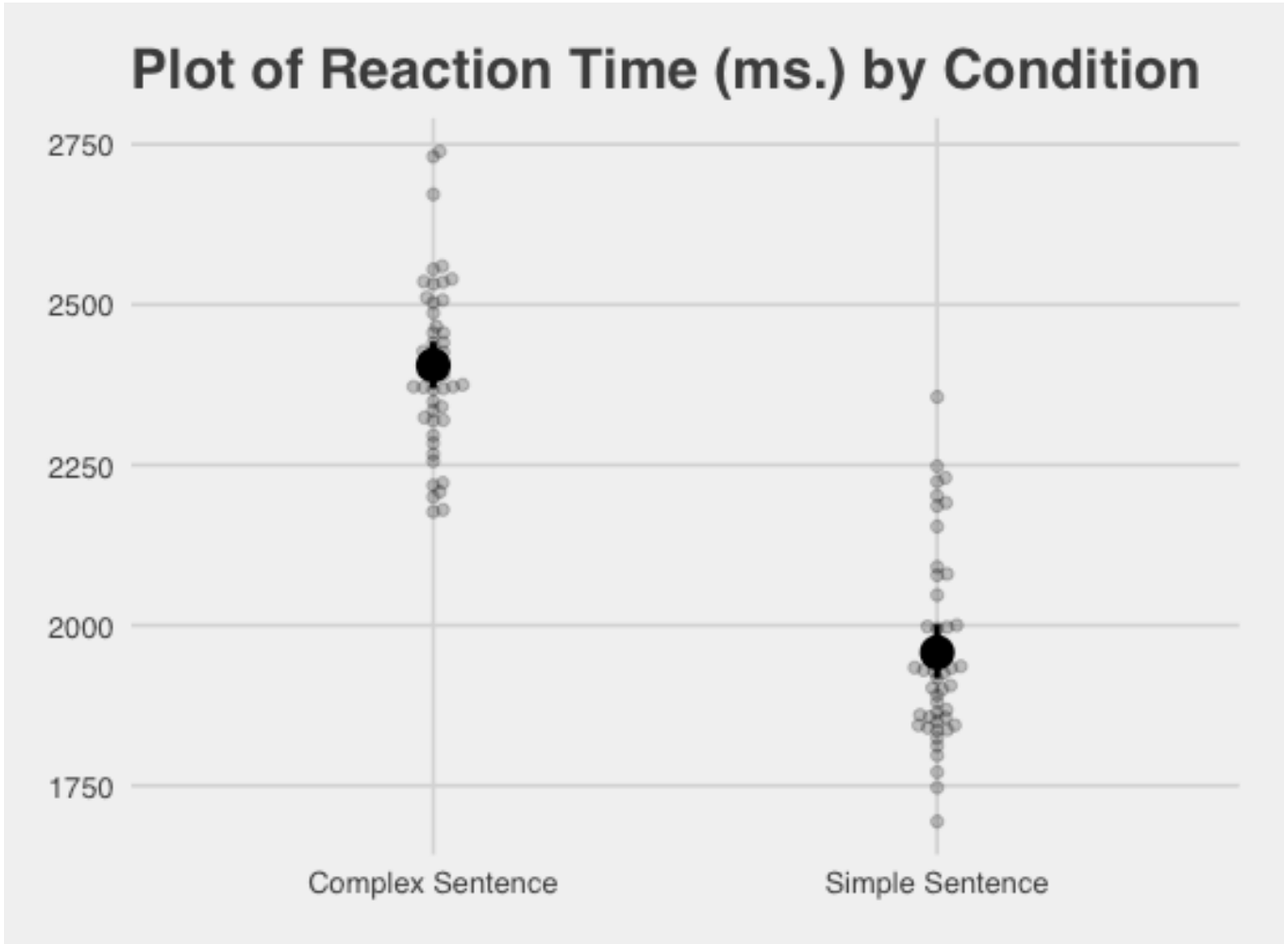

```
data_long %>%
  mutate(Condition = recode(condition,
                            "complex_sentence" = "Complex Sentence",
                            "simple_sentence" = "Simple Sentence")) %>%
  ggplot(aes(x = Condition, y = rt, group = Condition, fill = Condition)) +
  geom_beeswarm(alpha = .25) +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary(fun.data = "mean_cl_boot", colour = "black", size = 1) +
  labs(title = "Plot of Reaction Time (ms.) by Condition",
       x = "Condition",
       y = "Reaction Time (ms.)") +
  theme(text = element_text(size = 15))
```

Themes

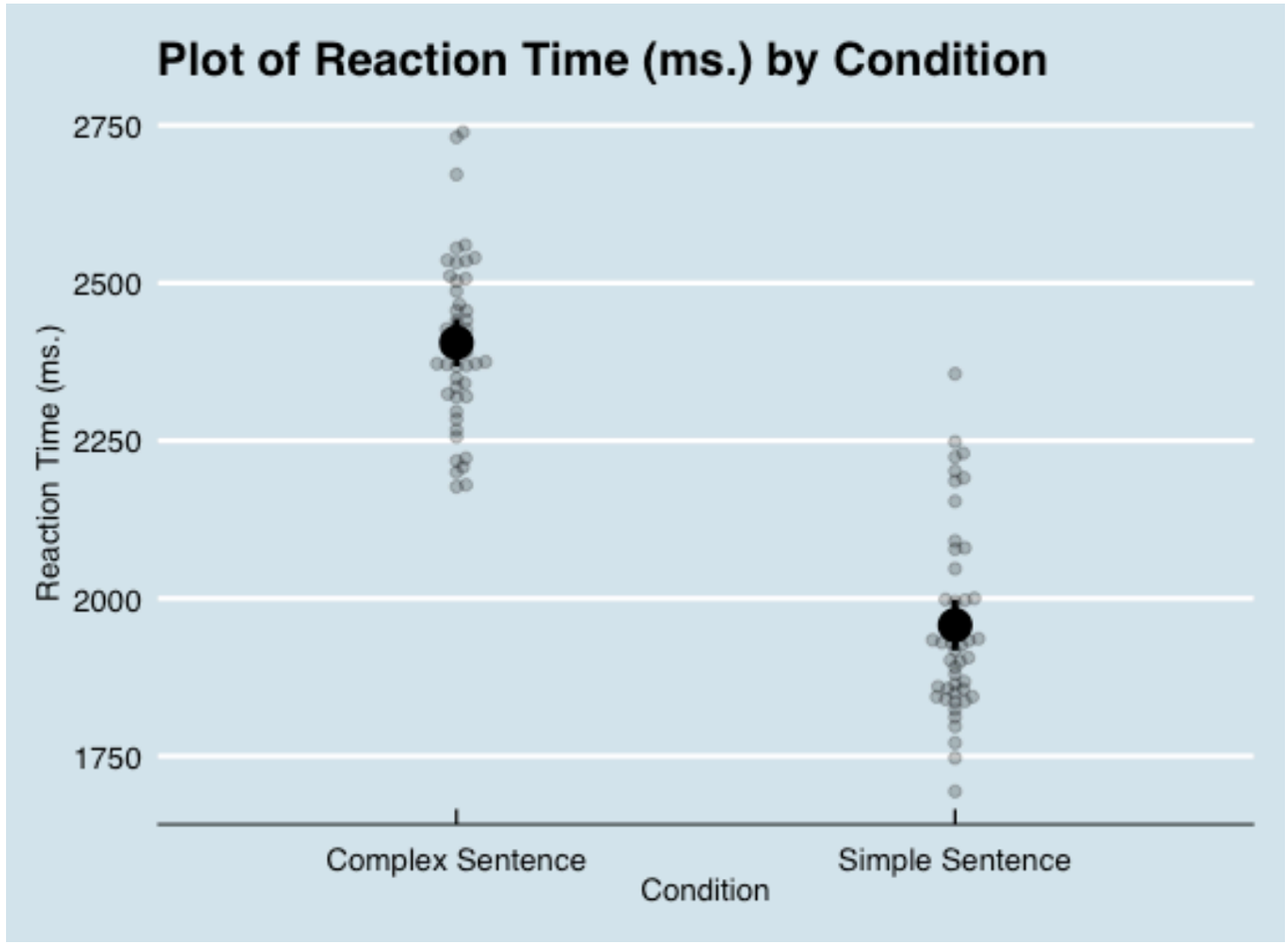
- The ggthemes package has lots of pre-built ggplot themes that we can apply to our ggplot visualisations.

```
>library(ggthemes)
```



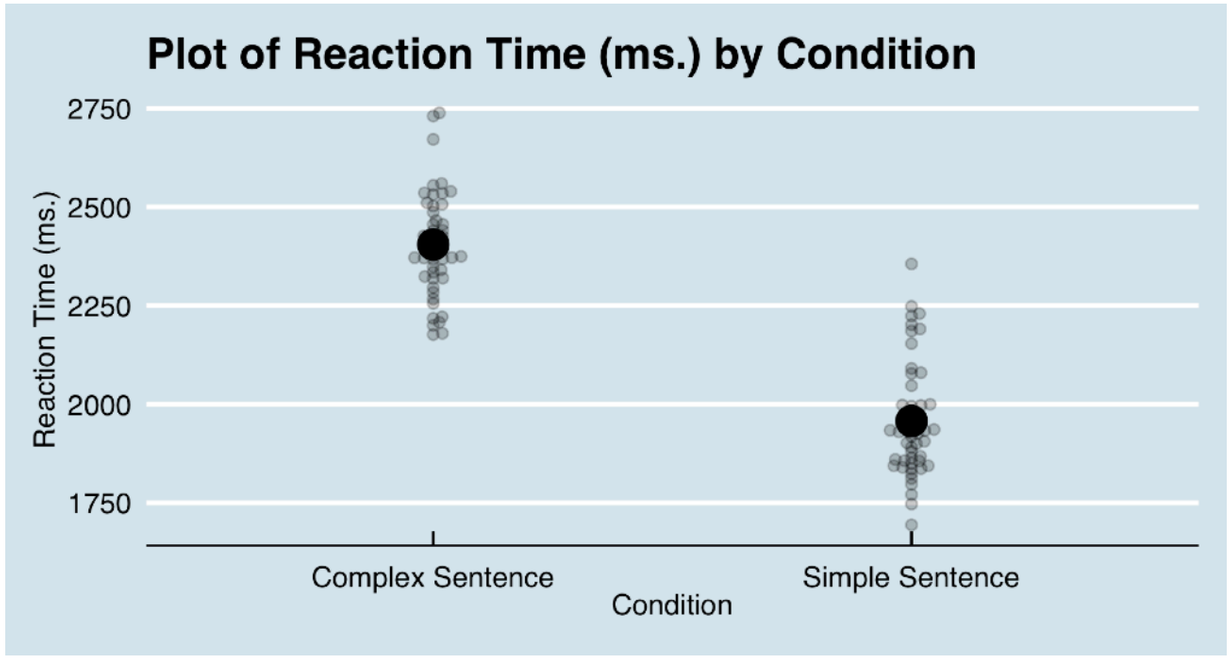



```
data_long %>%
  mutate(Condition = recode(condition,
                            "complex_sentence" = "Complex Sentence",
                            "simple_sentence" = "Simple Sentence")) %>%
  ggplot(aes(x = Condition, y = rt, group = Condition, fill = Condition)) +
  geom_beeswarm(alpha = .25) +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary(fun.data = "mean_cl_boot", colour = "black", size = 1) +
  labs(title = "Plot of Reaction Time (ms.) by Condition",
       x = "Condition",
       y = "Reaction Time (ms.)") +
  theme_fivethirtyeight()
```



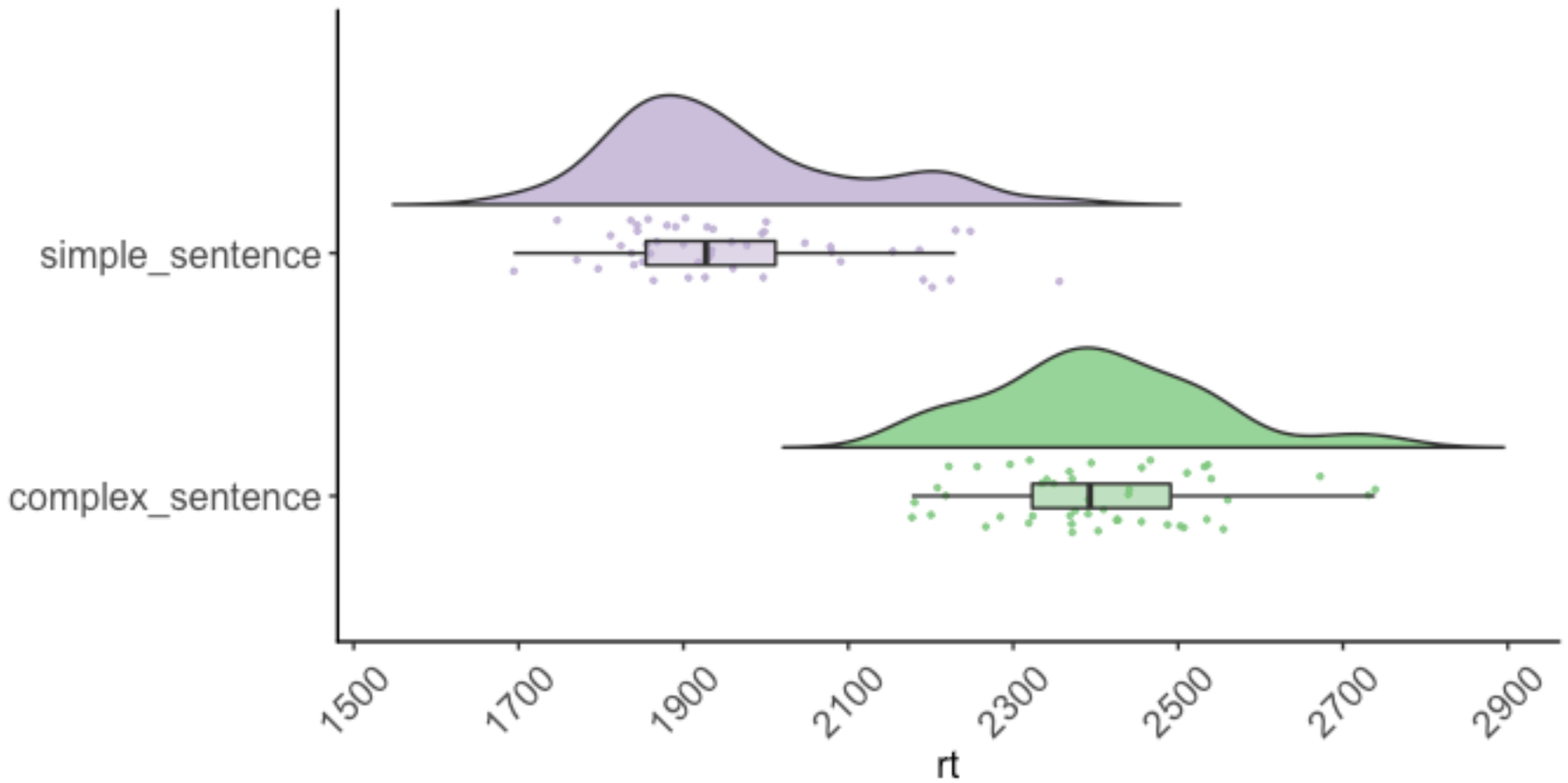

```
data_long %>%
  mutate(Condition = recode(condition,
                            "complex_sentence" = "Complex Sentence",
                            "simple_sentence" = "Simple Sentence")) %>%
  ggplot(aes(x = Condition, y = rt, group = Condition, fill = Condition)) +
  geom_beeswarm(alpha = .25) +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary(fun.data = "mean_cl_boot", colour = "black", size = 1) +
  labs(title = "Plot of Reaction Time (ms.) by Condition",
       x = "Condition",
       y = "Reaction Time (ms.)") +
  theme_economist()
```

Changing Plot Dimensions




```
my_plot <- data_long %>%
  mutate(Condition = recode(condition,
                            "complex_sentence" = "Complex Sentence",
                            "simple_sentence" = "Simple Sentence")) %>%
  ggplot(aes(x = Condition, y = rt, group = Condition, fill = Condition)) +
  geom_beeswarm(alpha = .25) +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary(fun.data = "mean_cl_boot", colour = "black", size = 1) +
  labs(title = "Plot of Reaction Time (ms.) by Condition",
       x = "Condition",
       y = "Reaction Time (ms.)") +
  theme_economist()

ggsave("my_plot.png", my_plot, height = 8, width = 15, units = "cm")
```

Raincloud Plots



Developed by Micah Allen (UCL), raincloud plots allow you to see the raw data, and the shape of the distribution alongside a box plot (capturing the median, 25th and 75th percentiles as hinges, and 1.5 * IQR from the hinges as the whisker length.)

A Variety of Plots Using the Same Dataset

We're going to use the built-in dataset 'mpg' to build a variety of plots. First, let's find out about the data by using the `head` function to view the first part of the data.

```
> head(mpg)
# A tibble: 6 x 11
  manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
  <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compact
2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compact
3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compact
4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compact
5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compact
6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compact
```

We can explore the data further by asking for all the possibilities in each column using the `unique` function. For example, we can check to see how many different types of cars there are. Note that the $ after the dataset name allows us to refer to a column in the mpg dataset.

```
> unique(mpg$manufacturer)
 [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"
 [8] "land rover" "lincoln"    "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"
[15] "volkswagen"
```

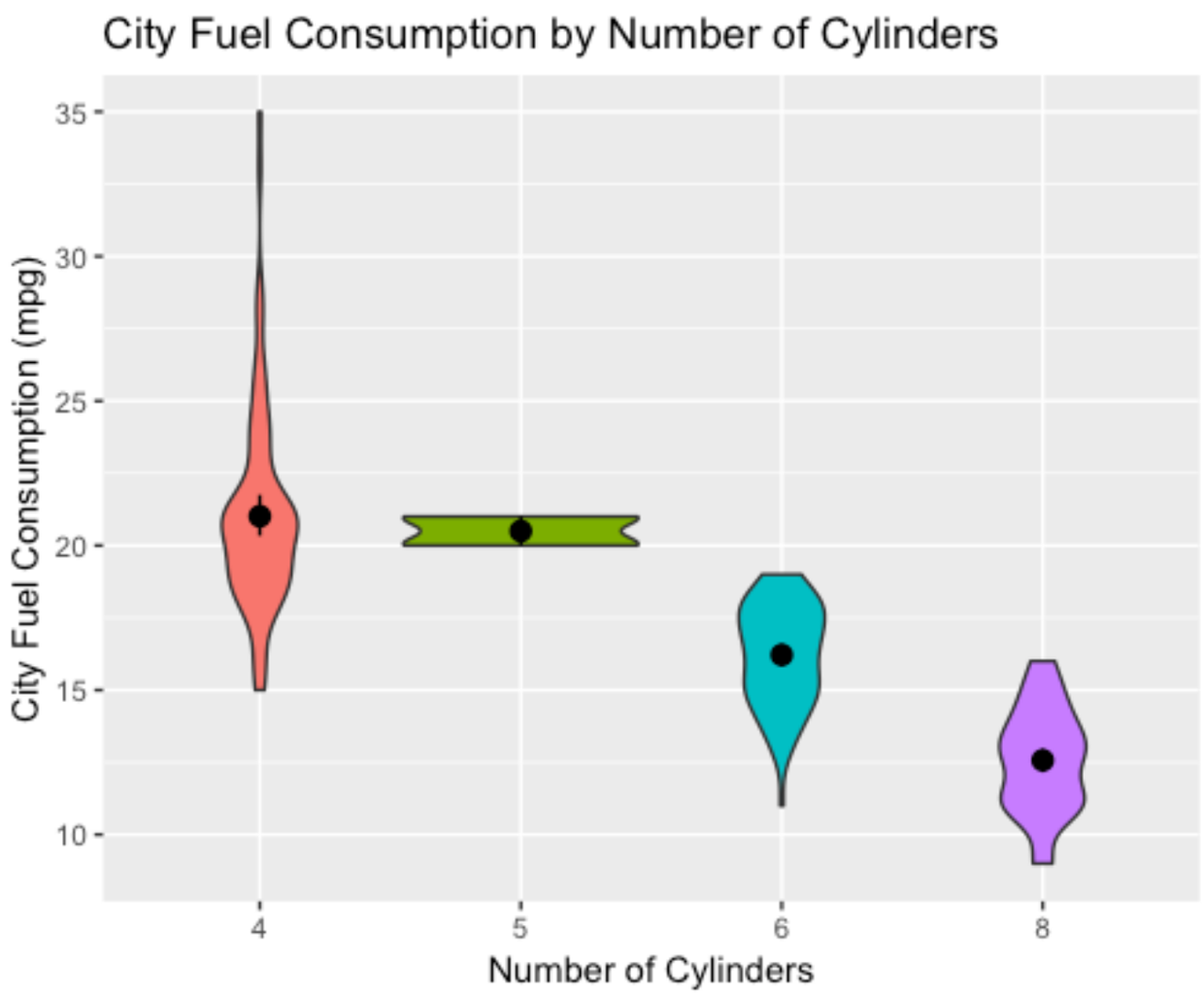
We can use the `length` function to give us the total number of unique possibilities:

```
> length(unique(mpg$manufacturer))
[1] 15
```

Let's look at a whole bunch of different visualisations using the mpg data set…

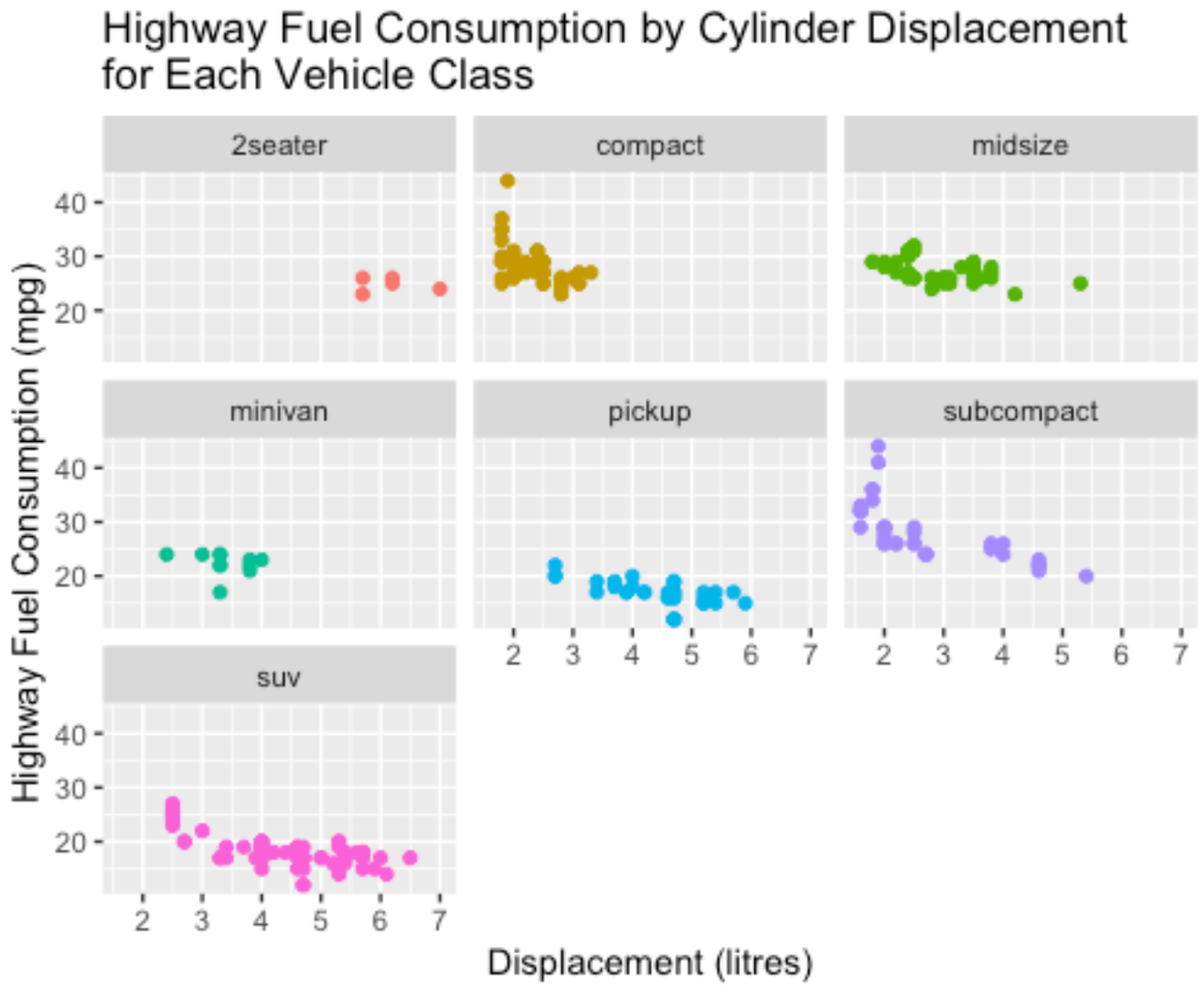
This illustrates the idea that there is not one 'correct' way to visualise the data, but rather that your choice of visualisation will be influenced by the question you're investigating, or the story you're wanting to tell…

Violin Plots



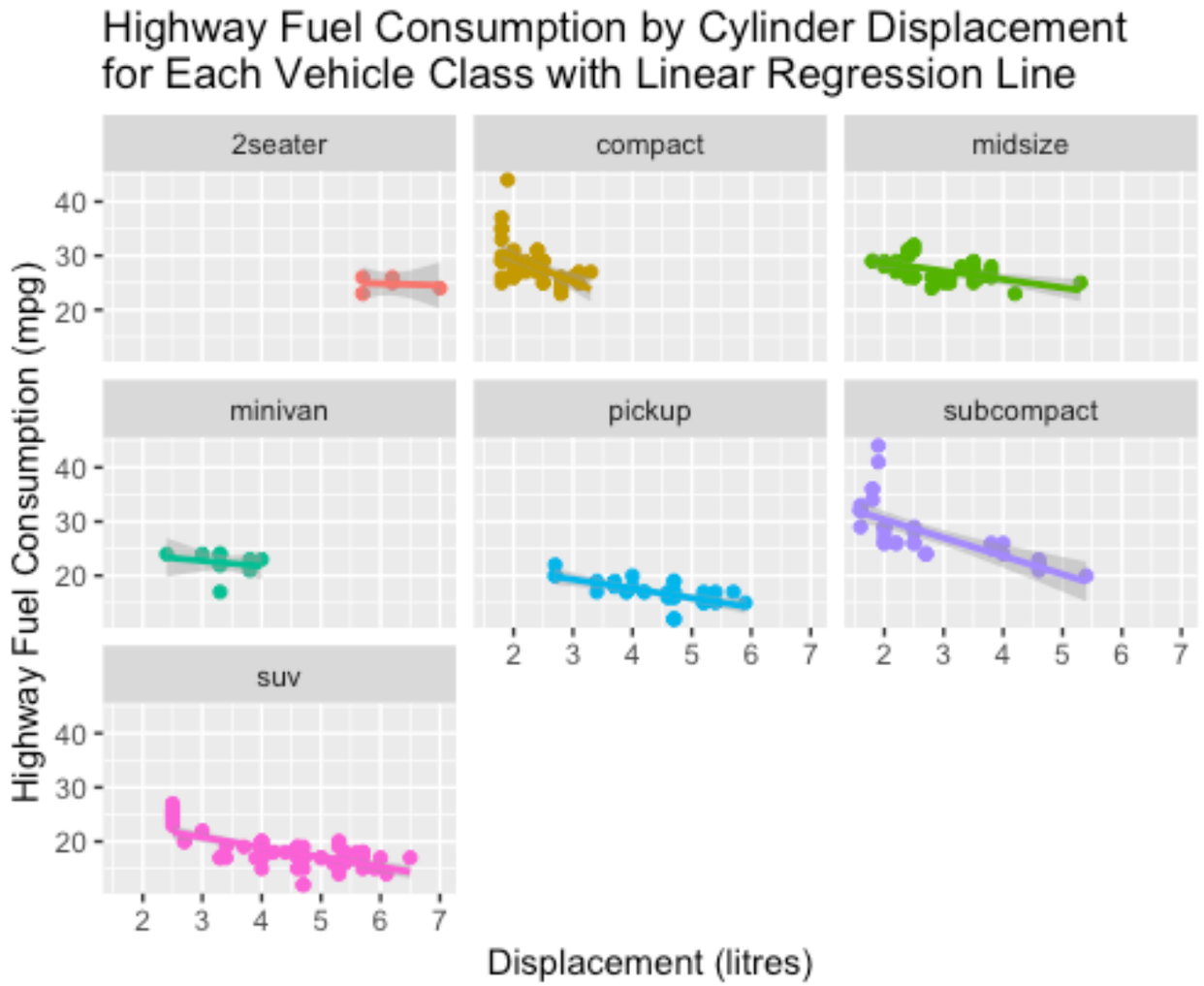

```
ggplot(mpg, aes(x = factor(cyl), y = cty, fill = factor(cyl))) +
  geom_violin() +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary(fun.data = mean_cl_boot, colour = "black", size = .5) +
  labs(title = "City Fuel Consumption by Number of Cylinders",
       x = "Number of Cylinders",
       y = "City Fuel Consumption (mpg)")
```

Faceting

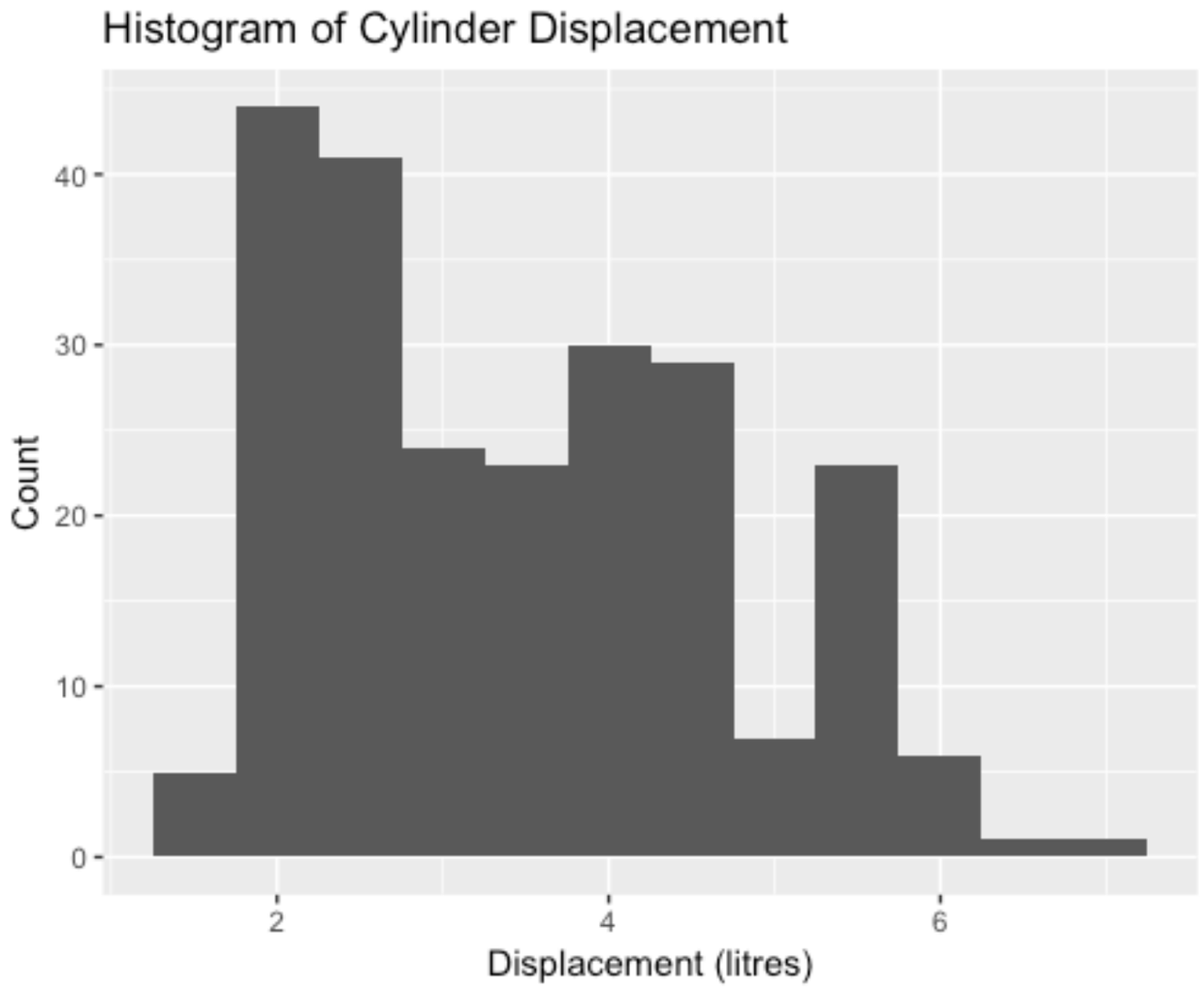



```
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_point() +
  facet_wrap(~ class) +
  guides(colour = FALSE) +
  labs(title = "Highway Fuel Consumption by Cylinder Displacement \nfor Each Vehicle Class",
       x = "Displacement (litres)",
       y = "Highway Fuel Consumption (mpg)")
```

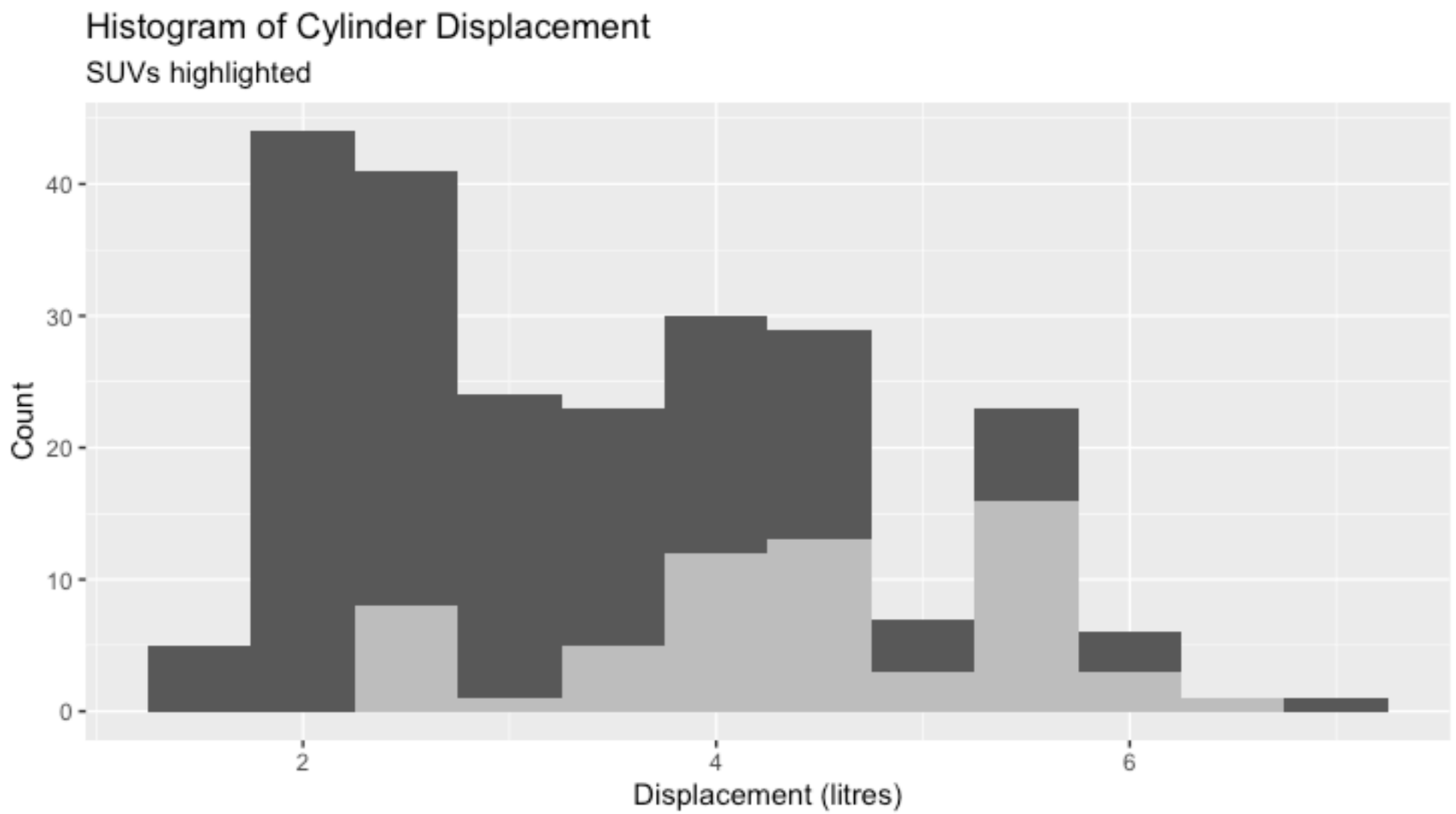
Adding a regression line



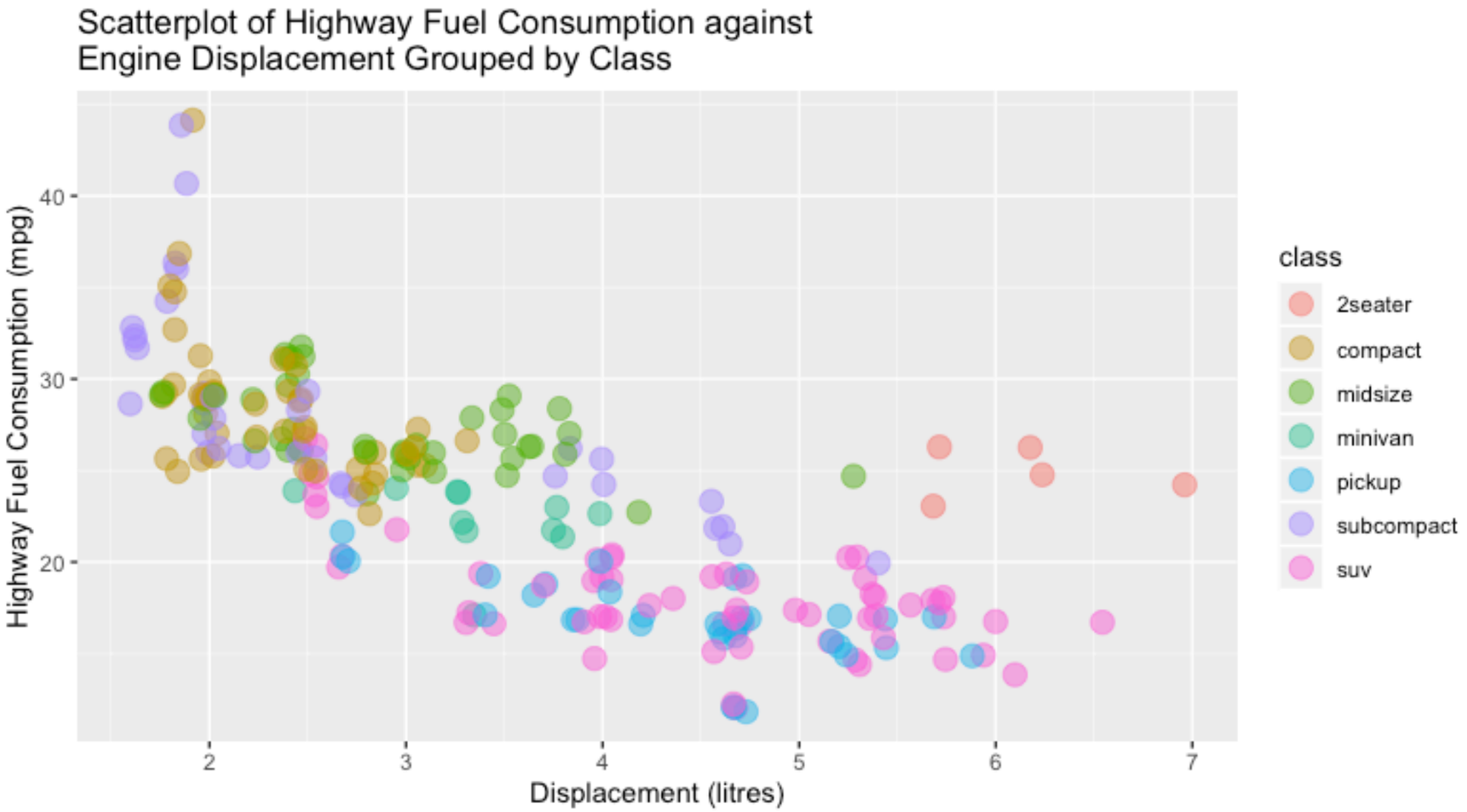

```
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_point() +
  facet_wrap(~ class) +
  guides(colour = FALSE) +
  geom_smooth(method = "lm") +
  labs(title = "Highway Fuel Consumption by Cylinder Displacement \nfor Each Vehicle Class",
       x = "Displacement (litres)",
       y = "Highway Fuel Consumption (mpg)")
```



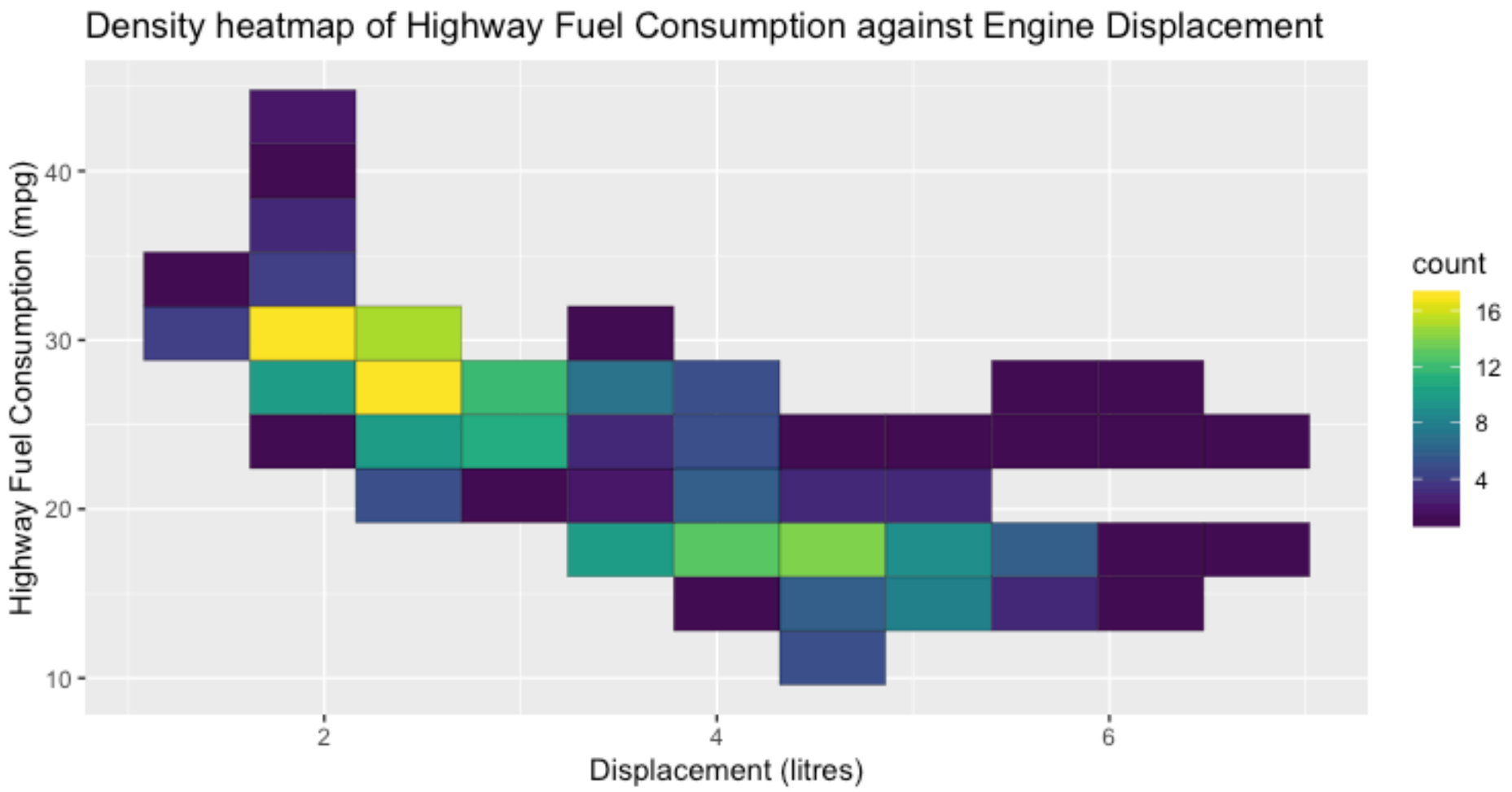

```
ggplot(mpg, aes(x = displ)) +
  geom_histogram(binwidth = .5) +
  guides(fill = FALSE) +
  labs(title = "Histogram of Cylinder Displacement",
       x = "Displacement (litres)",
       y = "Count")
```




```
ggplot(mpg, aes(x = displ)) +
  geom_histogram(binwidth = .5) +
  geom_histogram(data = filter(mpg, class == "suv"), fill = "grey", binwidth = .5) +
  guides(fill = FALSE) +
  labs(title = "Histogram of Cylinder Displacement",
       subtitle = "SUVs highlighted",
       x = "Displacement (litres)",
       y = "Count")
```




```
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_jitter(width = 0.05, alpha = .5, size = 4) +
  labs(title = "Scatterplot of Highway Fuel Consumption against \nEngine
Displacement Grouped by Class",
       x = "Displacement (litres)",
       y = "Highway Fuel Consumption (mpg)")
```




```
ggplot(mpg, aes(x = displ, y = hwy)) +
  stat_bin2d(bins = 10, colour = "black") +
  scale_fill_viridis() +
  labs(title = "Density heatmap of Highway Fuel Consumption against Engine Displacement",
       x = "Displacement (litres)",
       y = "Highway Fuel Consumption (mpg)")
```

Plotting Time Series Data

We can install the `gapminder` package which contains lots of interesting data about about life expectancy, population size, GDP for lots of countries collected over lots of years.

```
> gapminder
# A tibble: 1,704 x 6
   country     continent  year lifeExp      pop gdpPercap
   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
 1 Afghanistan Asia       1952    28.8  8425333      779.
 2 Afghanistan Asia       1957    30.3  9240934      821.
 3 Afghanistan Asia       1962    32.0 10267083      853.
 4 Afghanistan Asia       1967    34.0 11537966      836.
 5 Afghanistan Asia       1972    36.1 13079460      740.
 6 Afghanistan Asia       1977    38.4 14880372      786.
 7 Afghanistan Asia       1982    39.9 12881816      978.
 8 Afghanistan Asia       1987    40.8 13867957      852.
 9 Afghanistan Asia       1992    41.7 16317921      649.
10 Afghanistan Asia       1997    41.8 22227415      635.
# … with 1,694 more rows
```

Animated visualisations

For datasets with time series information, we might think it could be easier to tell our data story if we animate by time.

The `gganimate` package allows us to create animated visualisations from within R which we can import or embed in R Markdown documents.

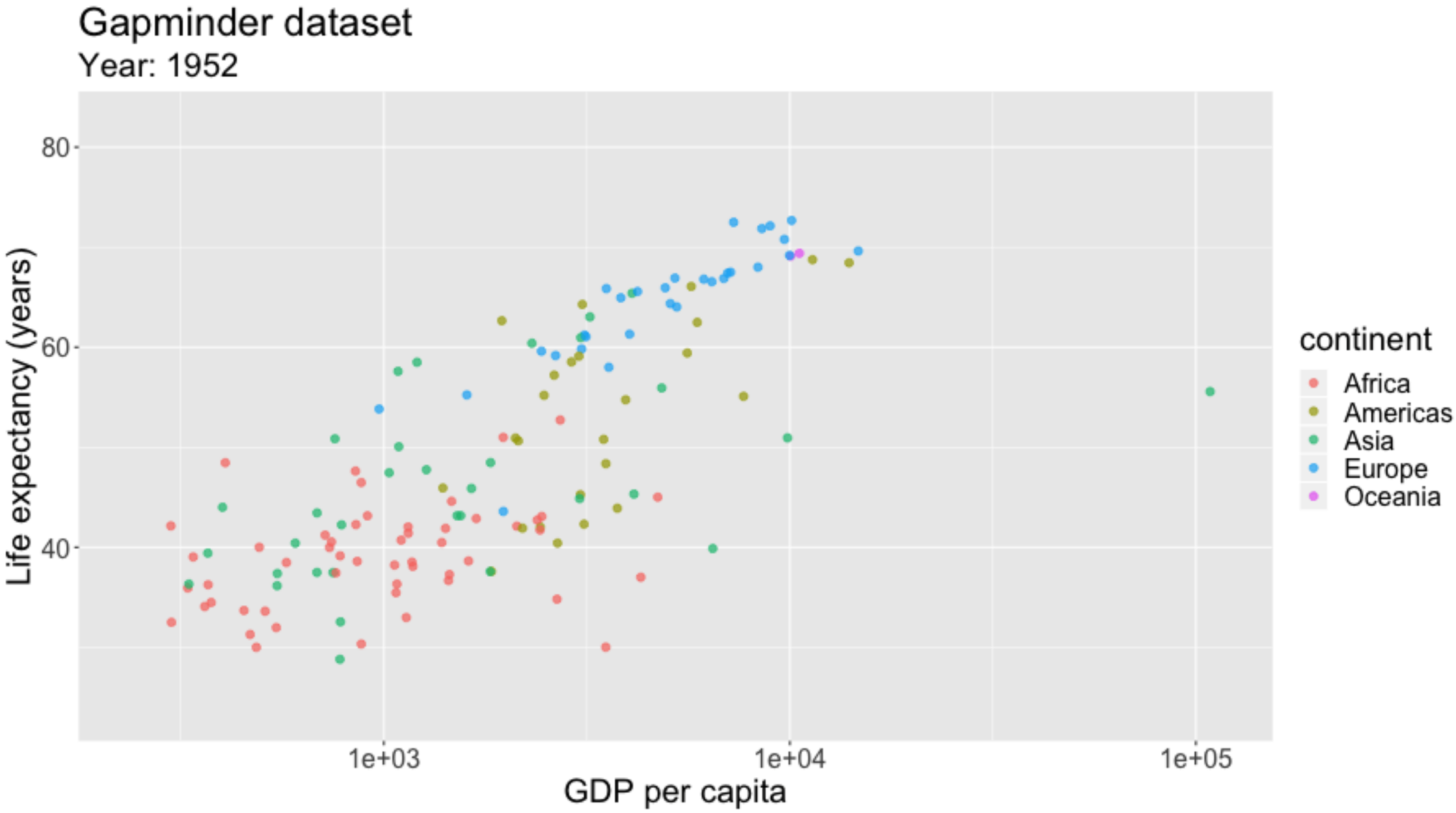
We need to install it via the usual route:

```
> install.packages("gganimate")
```

and then load it when we want to use it:

```
> library(gganimate)
```

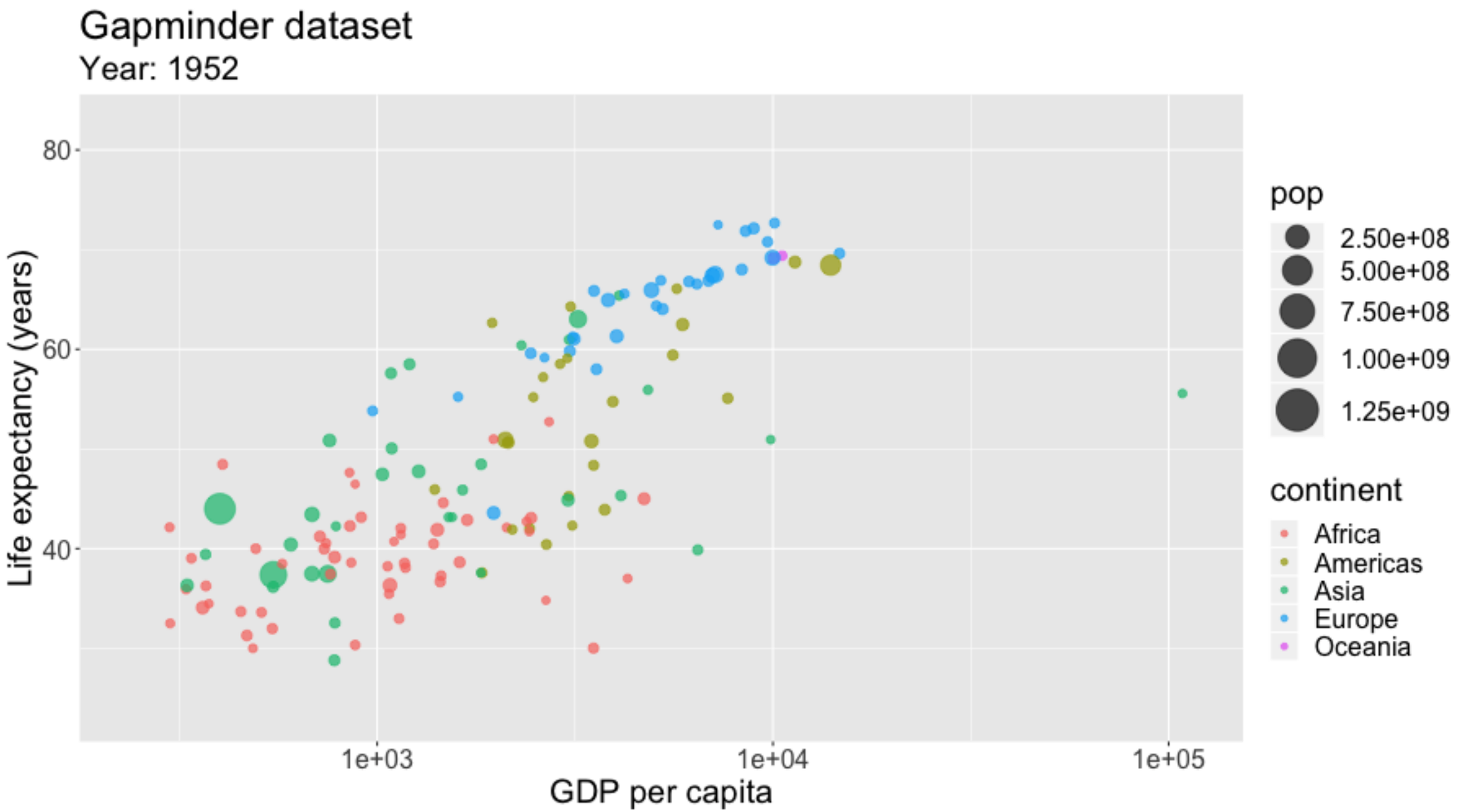
Animated Time Series Data



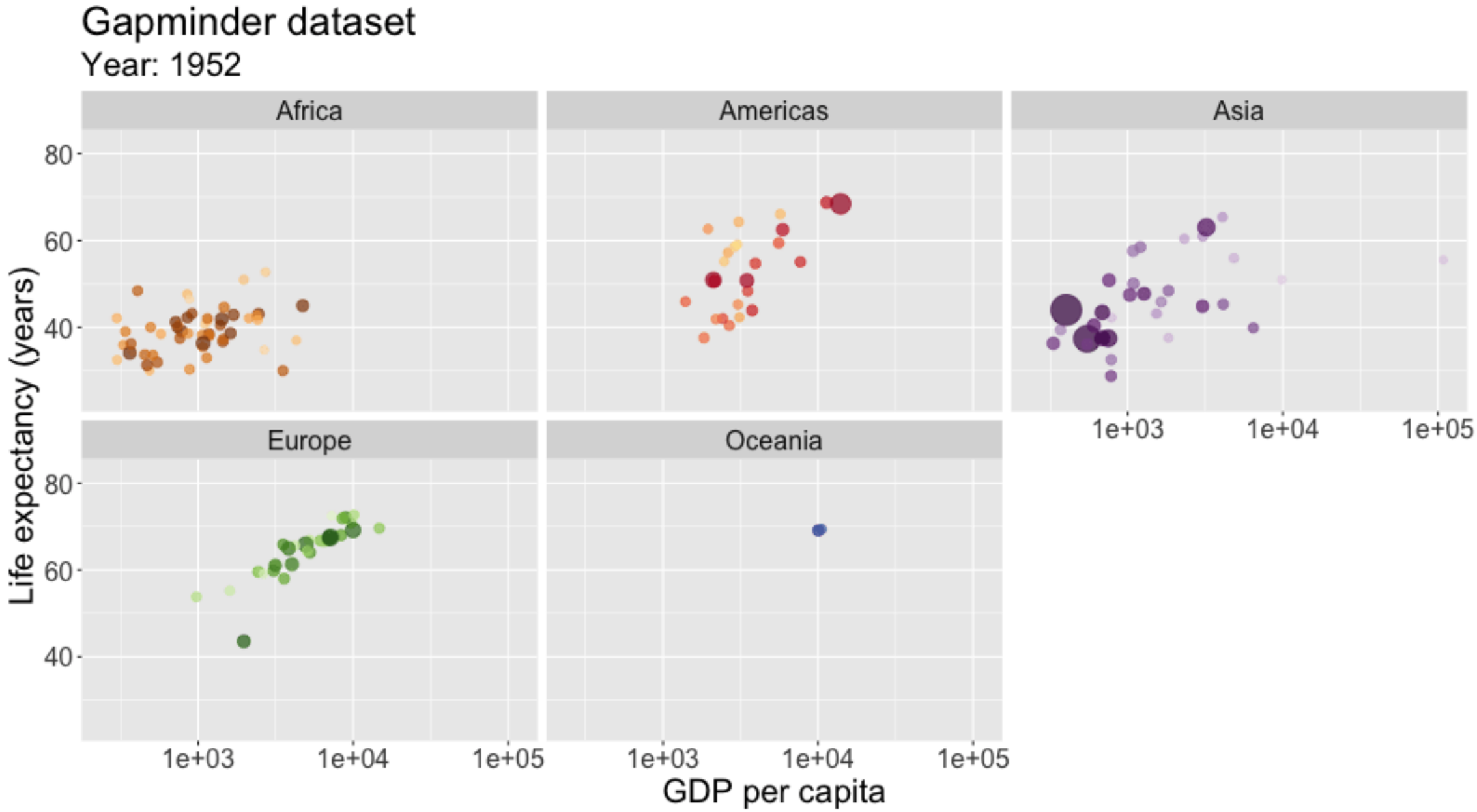
Visualising Data with 5 Variables Simultaneously

Animated Time Series Data

Now with a representation of population size.

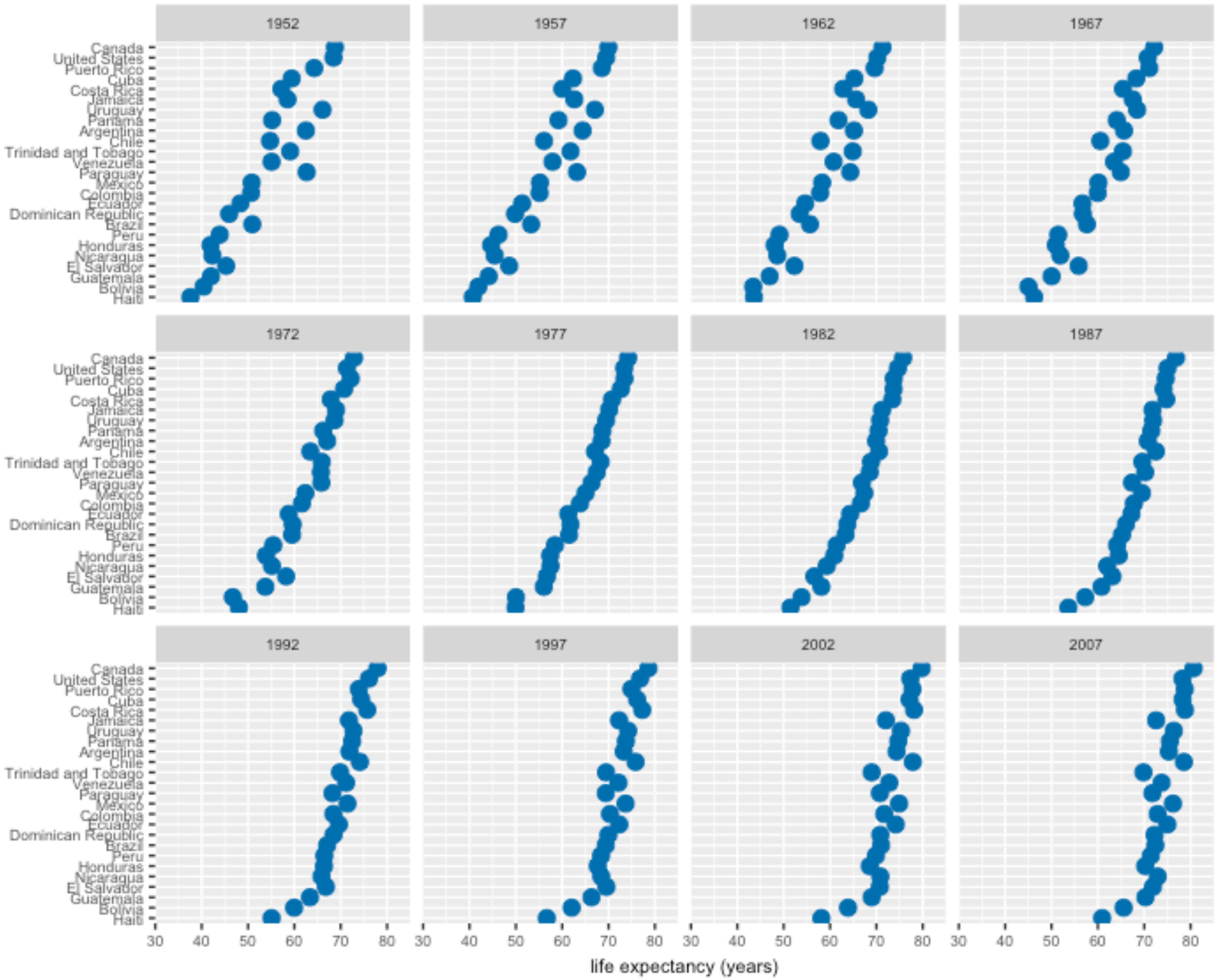


Separately by Continent

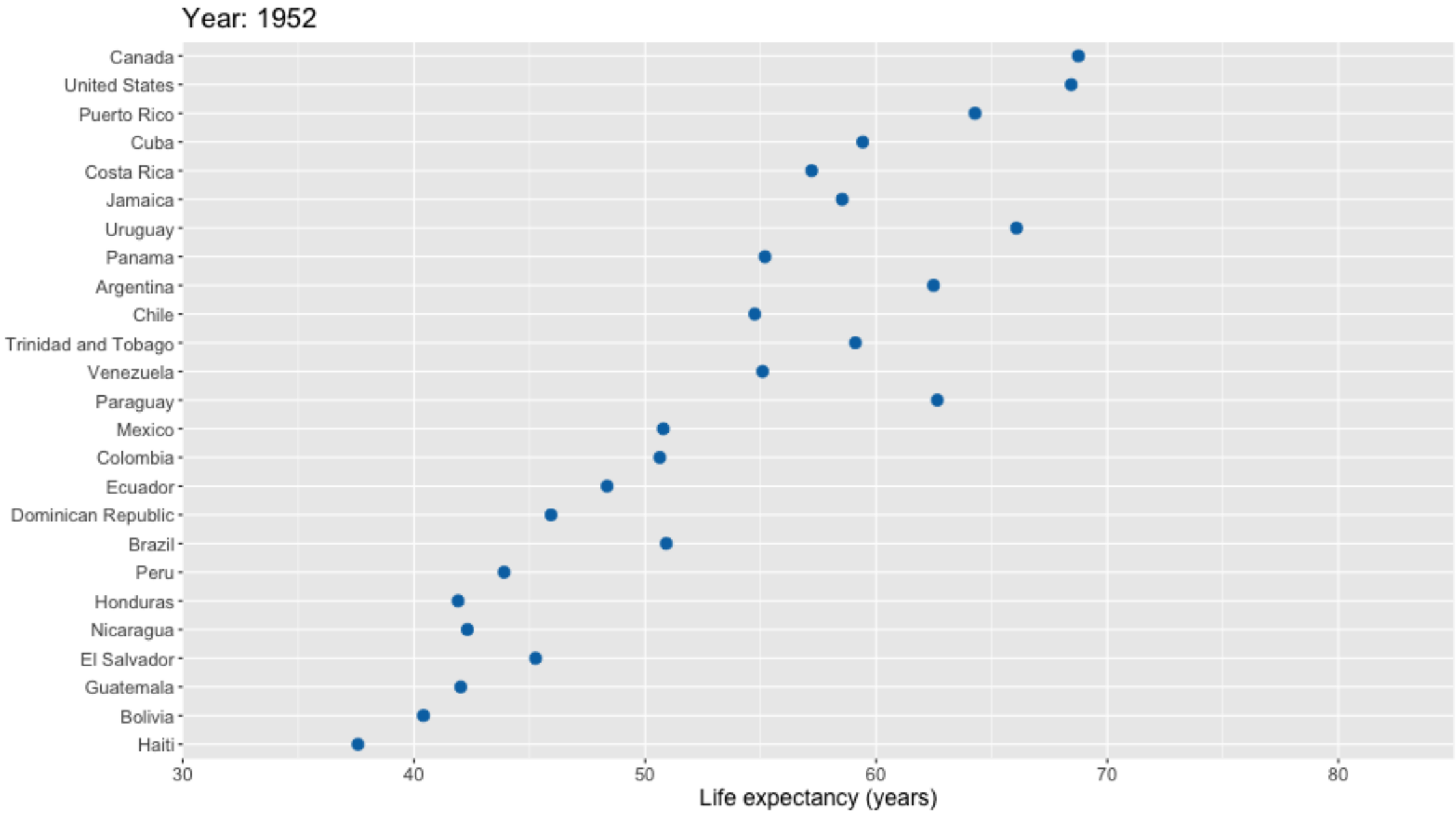


https://github.com/thomasp85/gganimate

Life Expectancy - Americas - Static



Life Expectancy - Americas - Animated



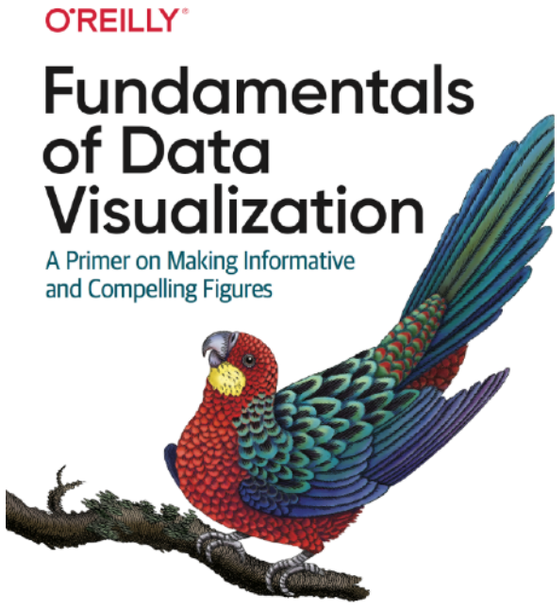
Although we have data only once every 5 years, the `gganimate` package interpolates between each census date to provide a smoother animation.

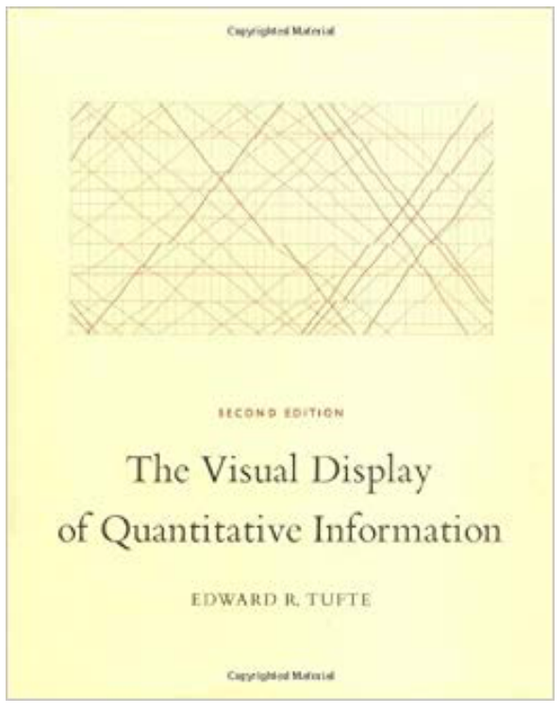
The Key Question

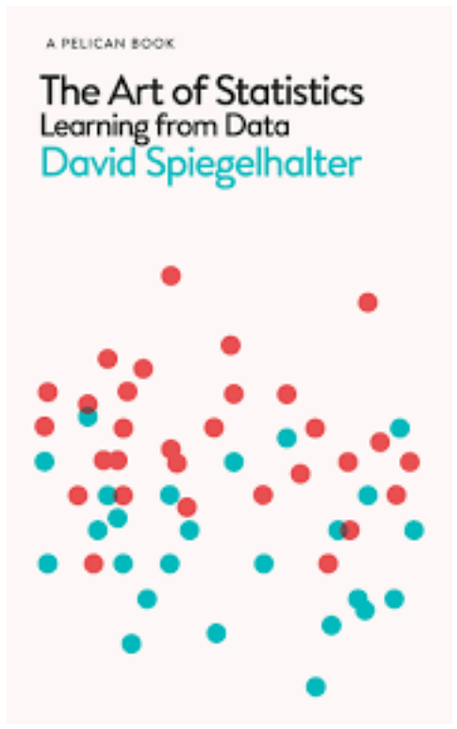
There is no such thing as the *best* way of visualising data - the method you choose will be determined by (e.g.) the type of data you have, the message you want to communicate, and the type of audience you will be communicating with.

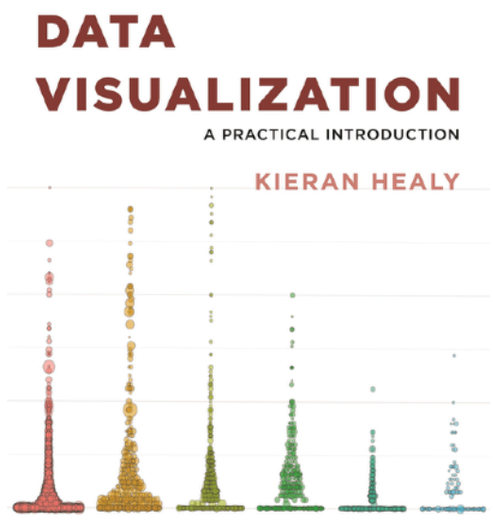
Animations can be helpful, but they involve data being presented at a pace that might not suit the viewer - probably best suited for communicating time series data.

Advanced Data Viz

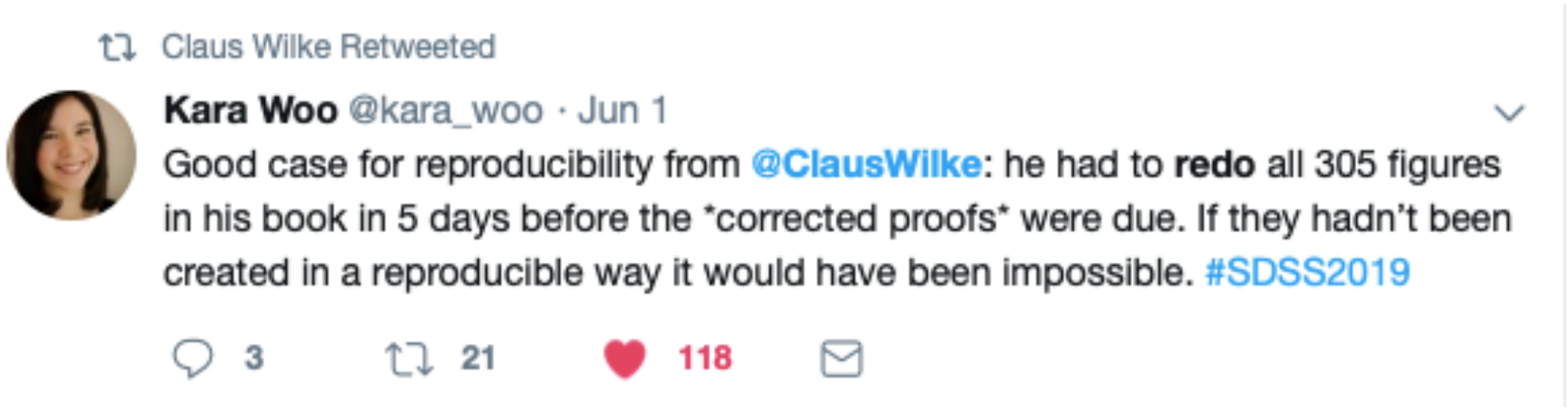








Make it reproducible...



"(visualizations) should be autogenerated as part of the data analysis pipeline (which should also be automated), and they should come out of the pipeline ready to be sent to the printer...

...the moment you manually edit a figure, your final figure becomes irreproducible. A third party cannot generate the exact same figure you did."

Claus Wilke, Fundamentals of Data Visualization (2019). Page xiii.

ggforce

- The aim of `ggplot2` is to aid in visual data investigations. This focus has led to a lack of facilities for composing specialised plots.

- `ggforce` aims to be a collection of mainly new stats and geoms that fills this gap.

https://cran.r-project.org/web/packages/ggforce/index.html

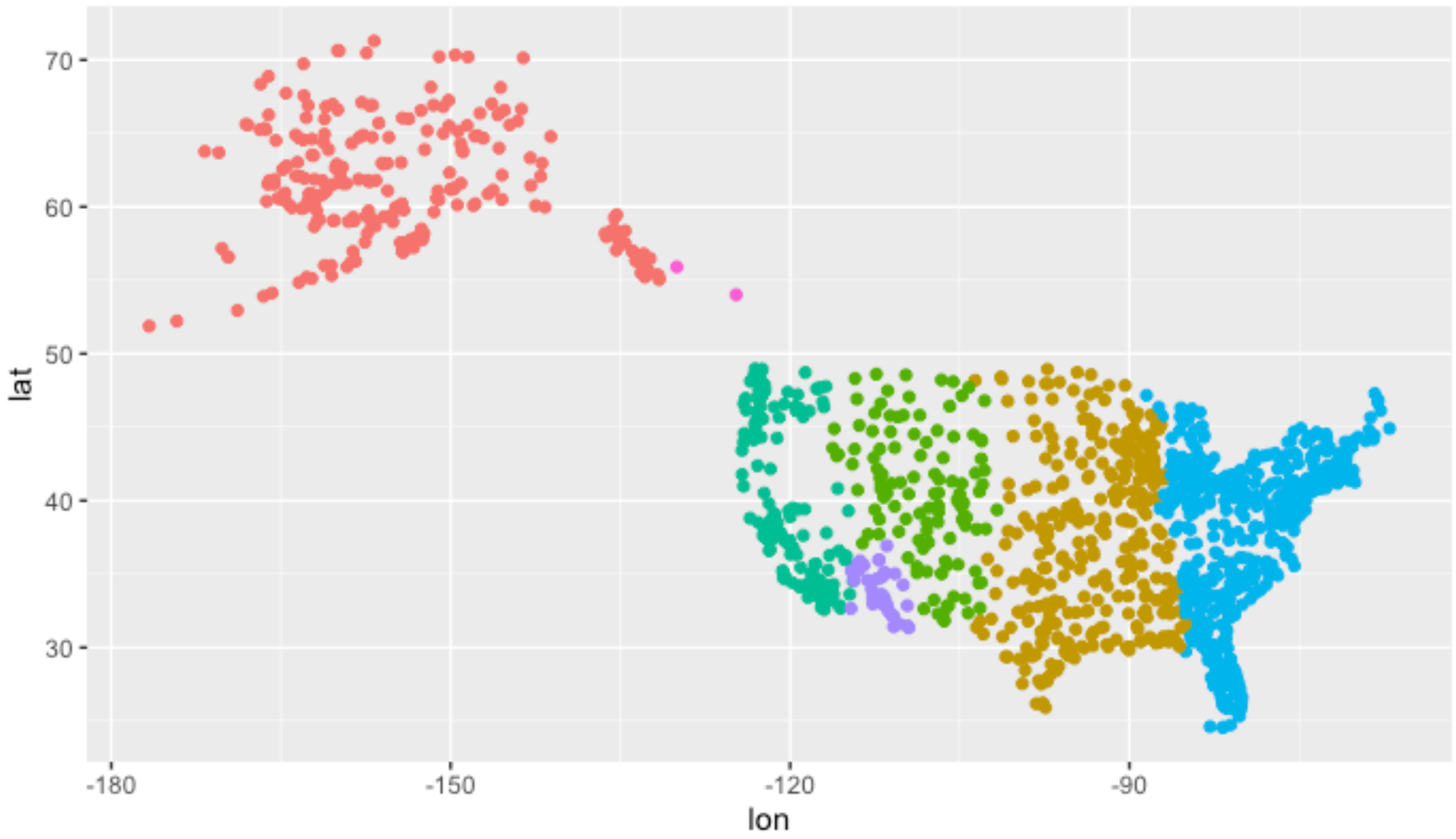
Spatial plotting

Using the package `nycflights13`.

```
> library(nycflights13)
> head(airports)
# A tibble: 6 x 8
  faa   name                            lat   lon   alt    tz dst   tzone
  <chr> <chr>                         <dbl> <dbl> <int> <dbl> <chr> <chr>
1 04G   Lansdowne Airport              41.1 -80.6  1044    -5 A     America/New_York
2 06A   Moton Field Municipal Airport  32.5 -85.7   264    -6 A     America/Chicago
3 06C   Schaumburg Regional            42.0 -88.1   801    -6 A     America/Chicago
4 06N   Randall Airport                41.4 -74.4   523    -5 A     America/New_York
5 09J   Jekyll Island Airport          31.1 -81.4    11    -5 A     America/New_York
6 0A9   Elizabethton Municipal Airport 36.4 -82.2  1593    -5 A     America/New_York
>
```

Simply plotting the latitude and longitude values for each row...

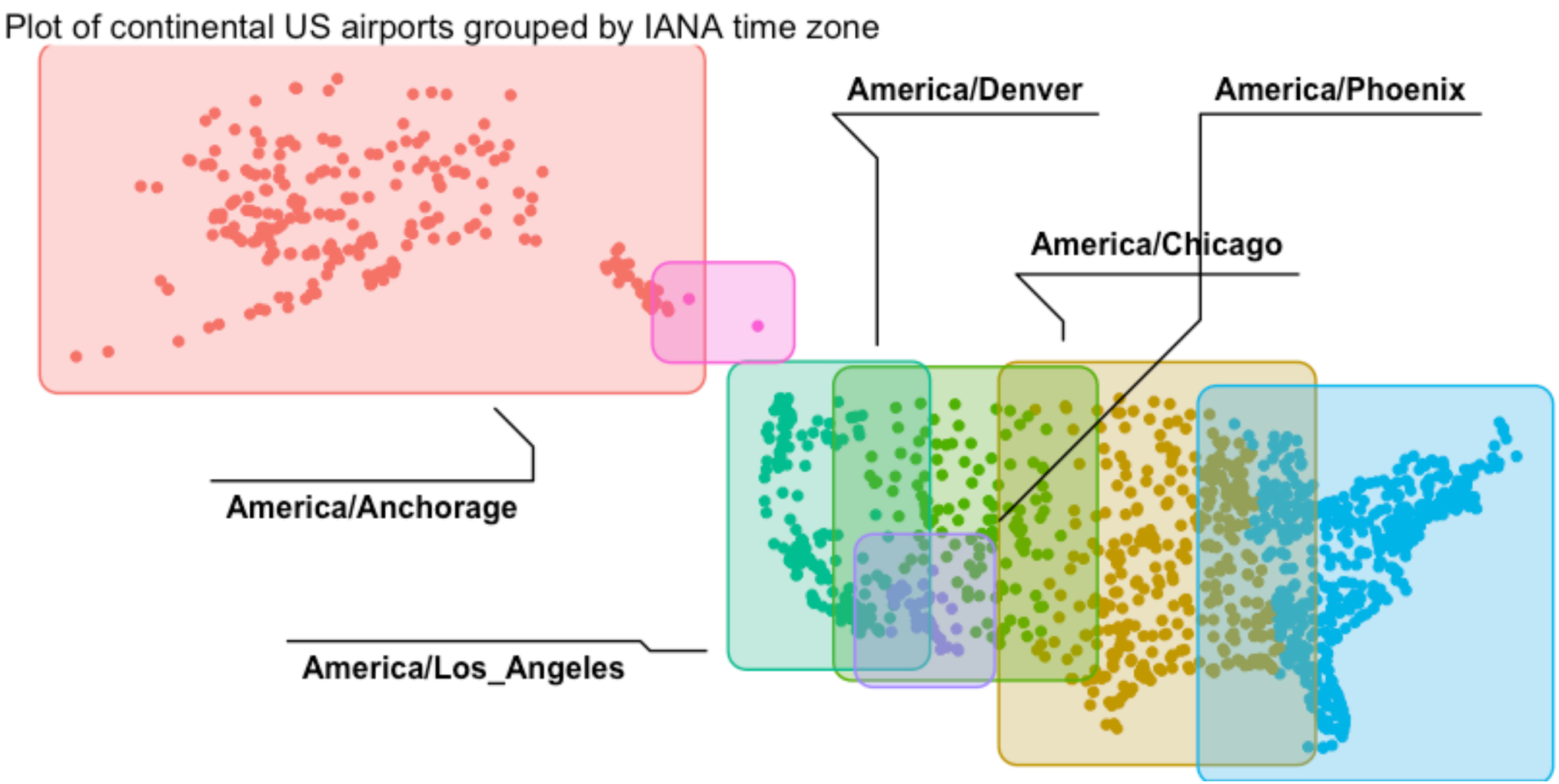
```
my_plot <- airports %>%
  filter(lon < 0, lat > 23, tzone != "\\N") %>%
  ggplot(aes(lon, lat, color = tzone)) +
  geom_point(show.legend = FALSE)
```



https://rviews.rstudio.com/2019/09/19/intro-to-ggforce/

Using `geom_mark_rect()`

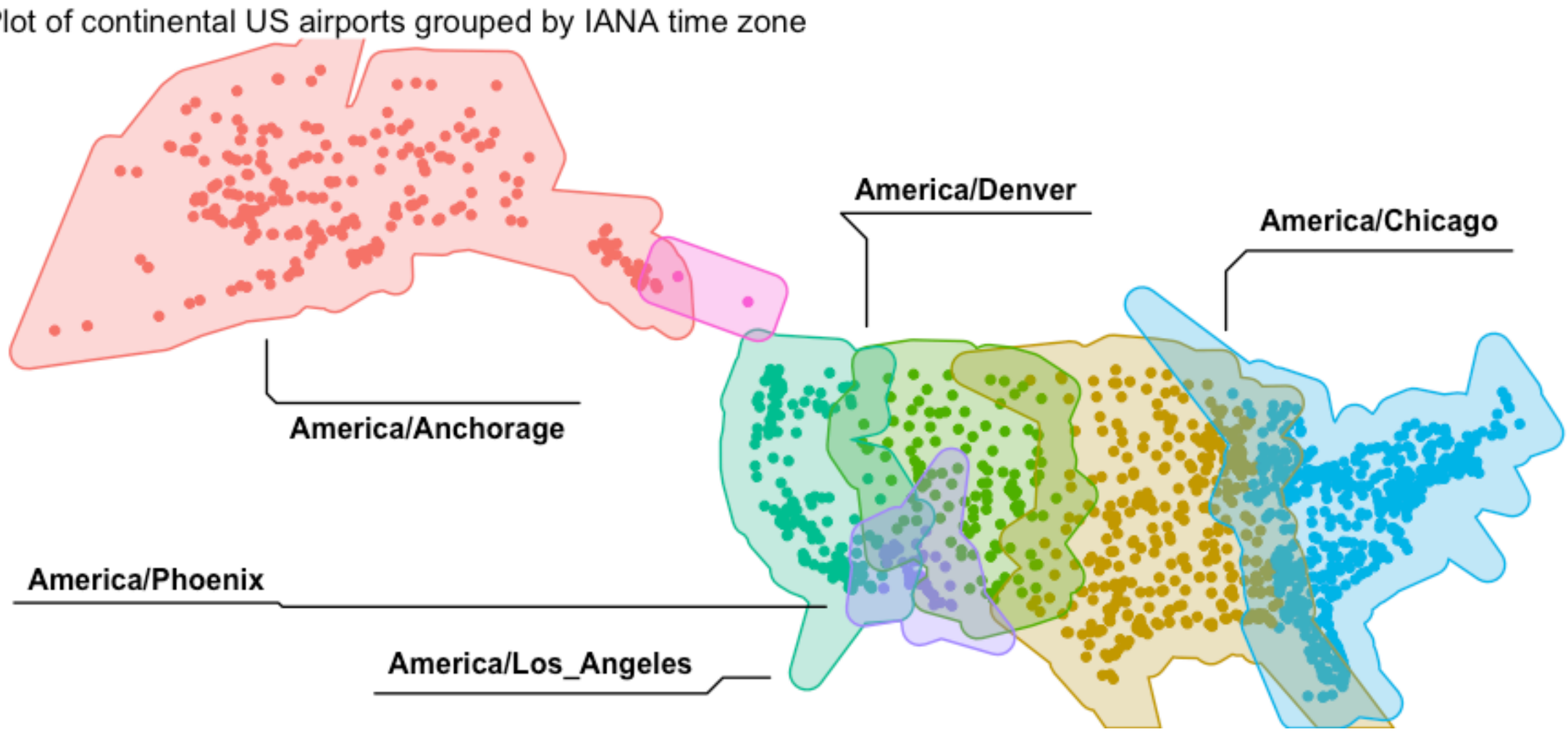
```
my_plot +
  geom_mark_rect(aes(label = tzone, fill = tzone), show.legend = FALSE) +
  labs(title = "Plot of continental US airports grouped by IANA time zone") +
  theme_void()
```



https://rviews.rstudio.com/2019/09/19/intro-to-ggforce/

Using `geom_mark_hull()`

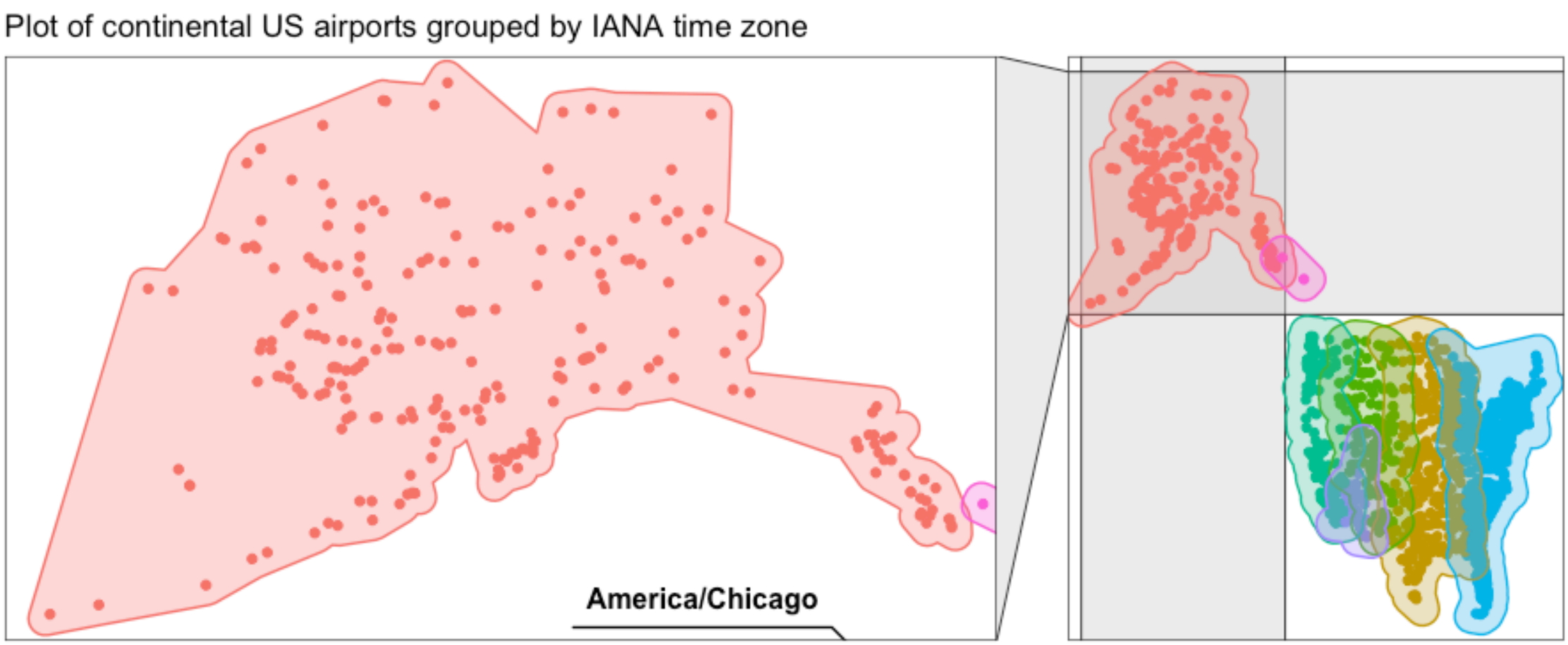
```
my_plot +
  geom_mark_hull(aes(label = tzone, fill = tzone), show.legend = FALSE) +
  labs(title = "Plot of continental US airports grouped by IANA time zone") +
  theme_void()
```



https://rviews.rstudio.com/2019/09/19/intro-to-ggforce/

Using `facet_zoom()`

```
my_plot +
  geom_mark_hull(aes(label = tzone, fill = tzone), show.legend = FALSE, expand =
unit(3, "mm")) +
  labs(title = "Plot of continental US airports grouped by IANA time zone") +
  facet_zoom(xy = (tzone == "America/Anchorage")) +
  theme_no_axes()
```



https://rviews.rstudio.com/2019/09/19/intro-to-ggforce/

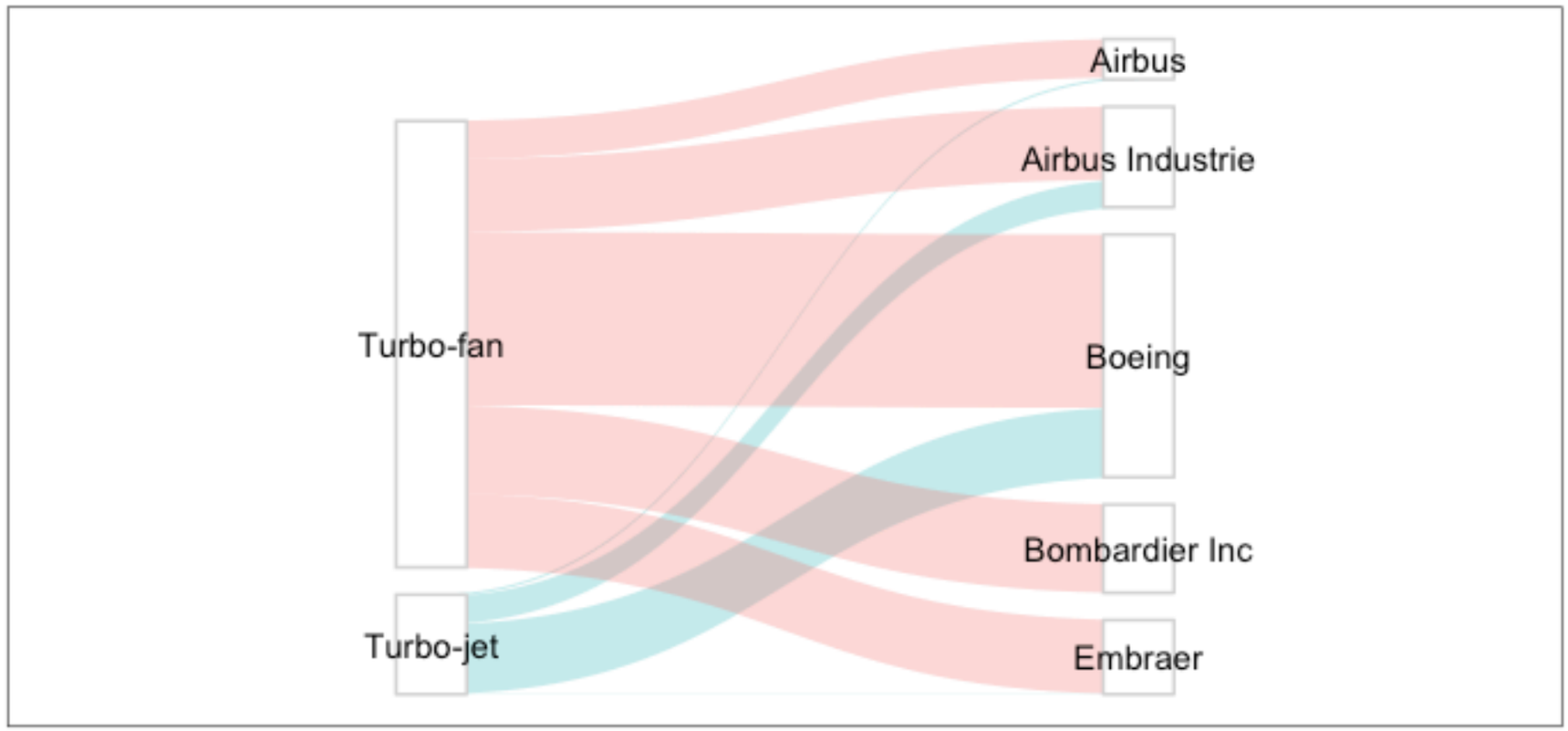
Alluvial Diagram

Alluvial diagrams can be used to visualise the relationship between categorical variables.

```
> prep_planes
# A tibble: 1,287 x 2
   manufacturer     engine
   <chr>            <chr>
 1 Embraer          Turbo-fan
 2 Airbus Industrie Turbo-fan
 3 Airbus Industrie Turbo-fan
 4 Embraer          Turbo-fan
 5 Airbus Industrie Turbo-fan
 6 Airbus Industrie Turbo-fan
 7 Airbus Industrie Turbo-fan
 8 Airbus Industrie Turbo-fan
 9 Airbus Industrie Turbo-fan
10 Embraer          Turbo-fan
# … with 1,277 more rows
```

Alluvial Diagram

```
prep_planes %>%
  gather_set_data(1:2) %>%
  ggplot(aes(x = x, id = id, split = y, value = 1))  +
  geom_parallel_sets(aes(fill = engine), show.legend = FALSE, alpha = 0.3) +
  geom_parallel_sets_axes(axis.width = 0.1, color = "lightgrey", fill =
"white") +
  geom_parallel_sets_labels(angle = 0) +
  theme_no_axes()
```

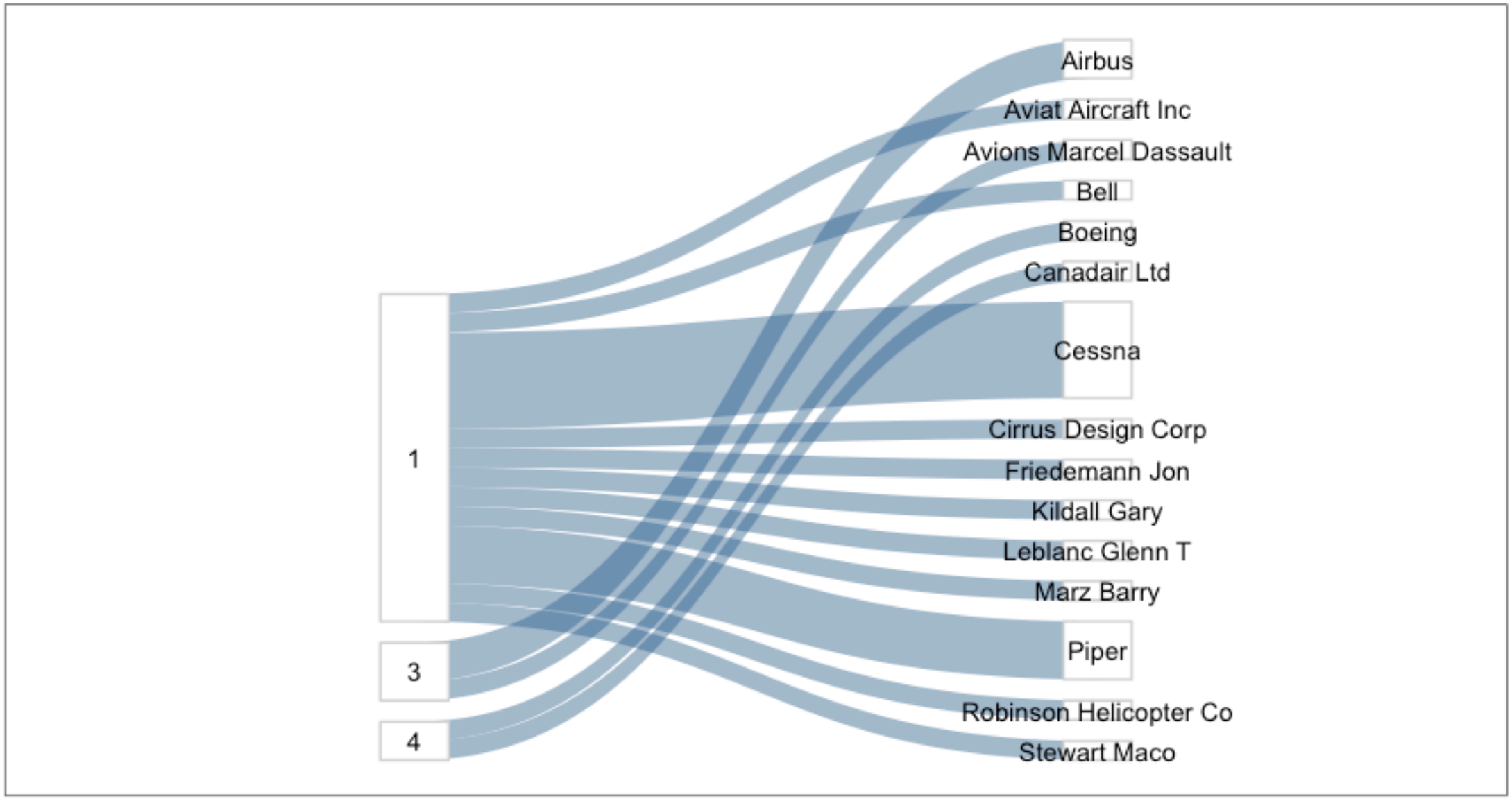


https://rviews.rstudio.com/2019/09/19/intro-to-ggforce/

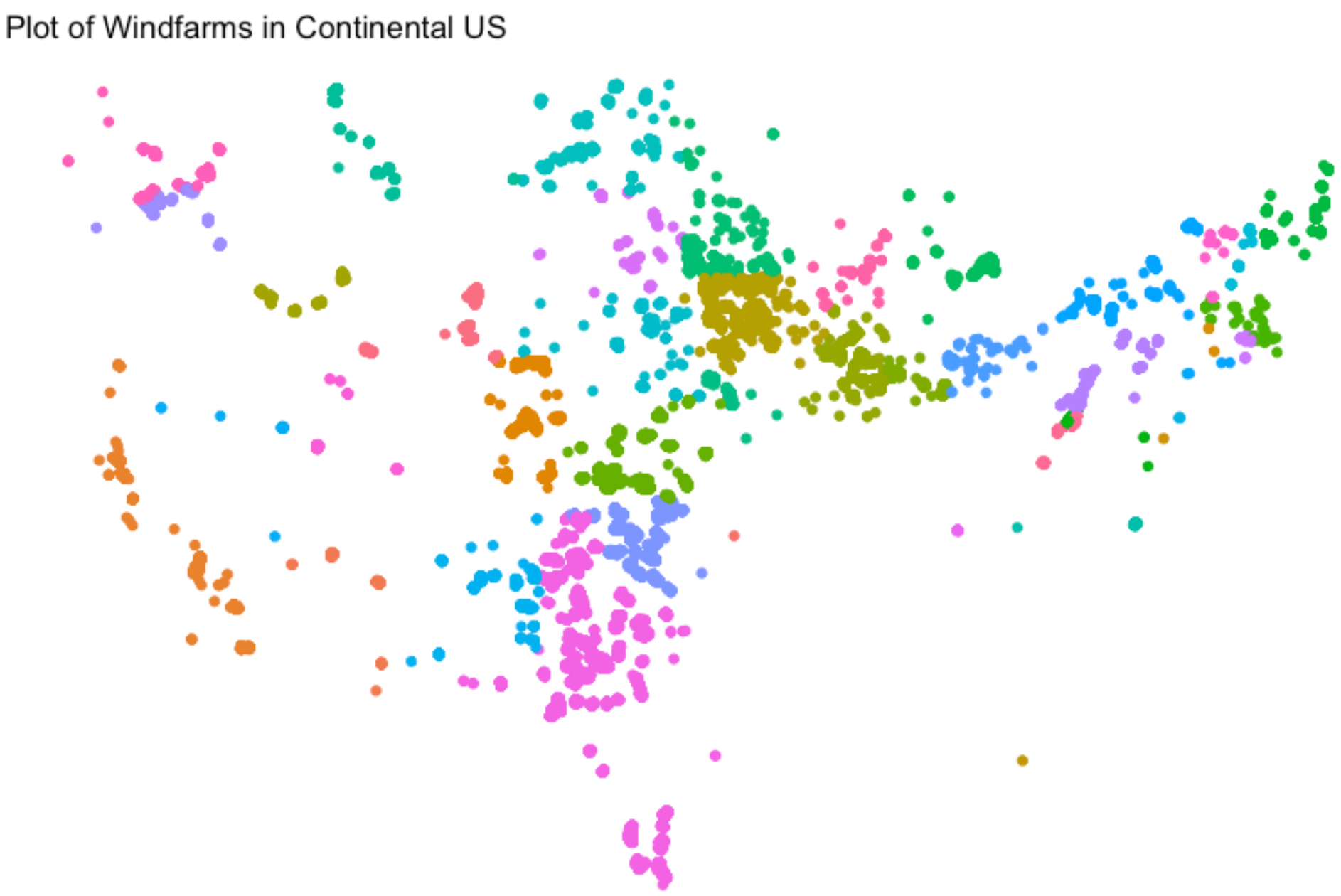
Grouping by Manufacturer and Number of Engines - Planes made after 1960 and excluding 2-engine aircraft

```
prep_planes <- planes %>%
  filter(year > 1960) %>%
  filter(engines != 2) %>%
  select(manufacturer, engines) %>%
  mutate(manufacturer = str_to_title(manufacturer))

> prep_planes %>%
+   group_by(manufacturer, engines) %>%
+   summarise(n())
# A tibble: 15 x 3
# Groups:   manufacturer [15]
   manufacturer           engines `n()`
   <chr>                    <int> <int>
 1 Airbus                       3     2
 2 Aviat Aircraft Inc           1     1
 3 Avions Marcel Dassault       3     1
 4 Bell                         1     1
 5 Boeing                       4     1
 6 Canadair Ltd                 4     1
 7 Cessna                       1     5
 8 Cirrus Design Corp           1     1
 9 Friedemann Jon               1     1
10 Kildall Gary                 1     1
11 Leblanc Glenn T              1     1
12 Marz Barry                   1     1
13 Piper                        1     3
14 Robinson Helicopter Co       1     1
15 Stewart Maco                 1     1
```

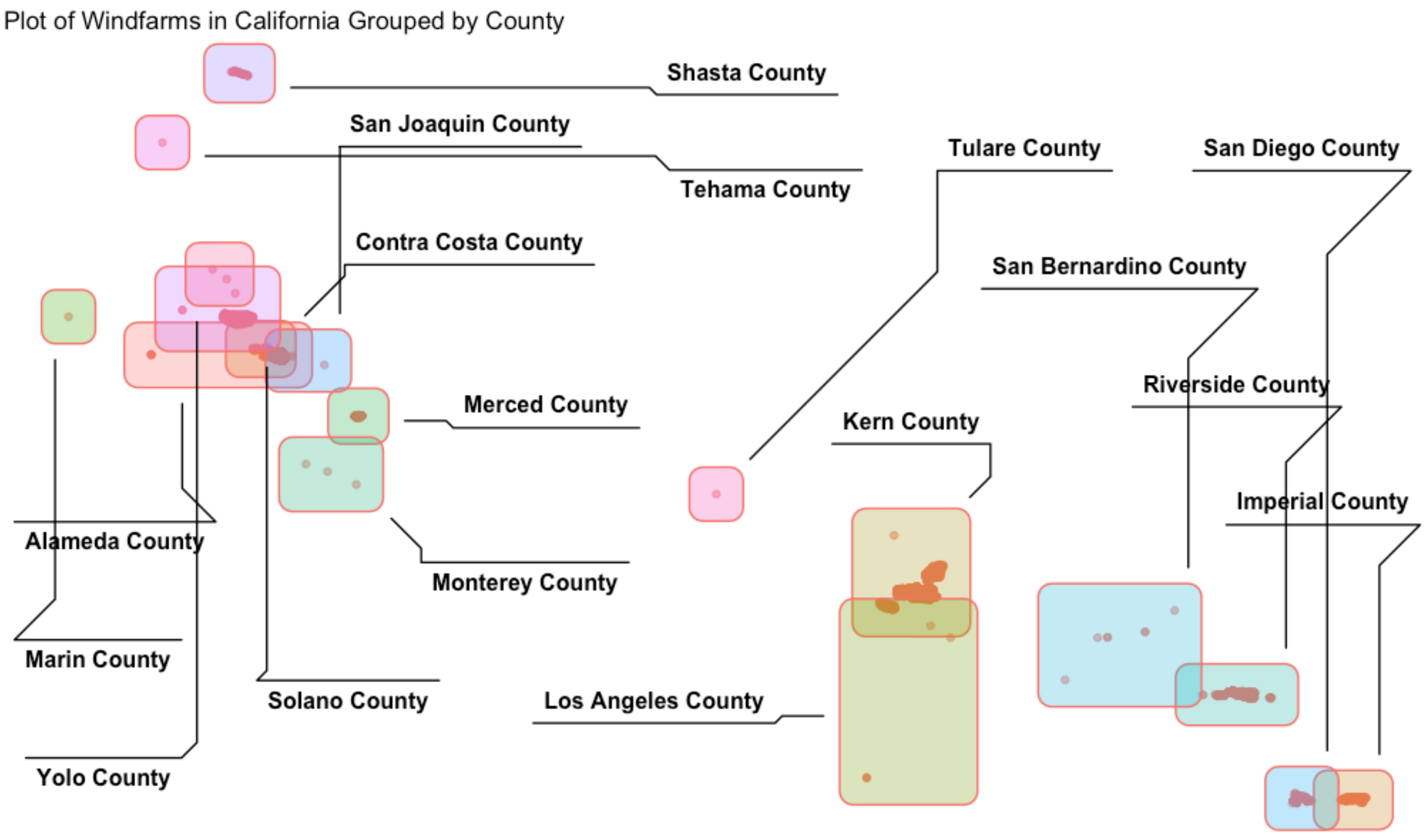
```
prep_planes %>%
  gather_set_data(1:2) %>%
  ggplot(aes(x = x, id = id, split = y, value = 1))  +
  geom_parallel_sets(aes(fill = engines), show.legend = FALSE, alpha = 0.5) +
  geom_parallel_sets_axes(axis.width = 0.1, color = "lightgrey", fill =
"white") +
  geom_parallel_sets_labels(angle = 0) +
  theme_no_axes()
```



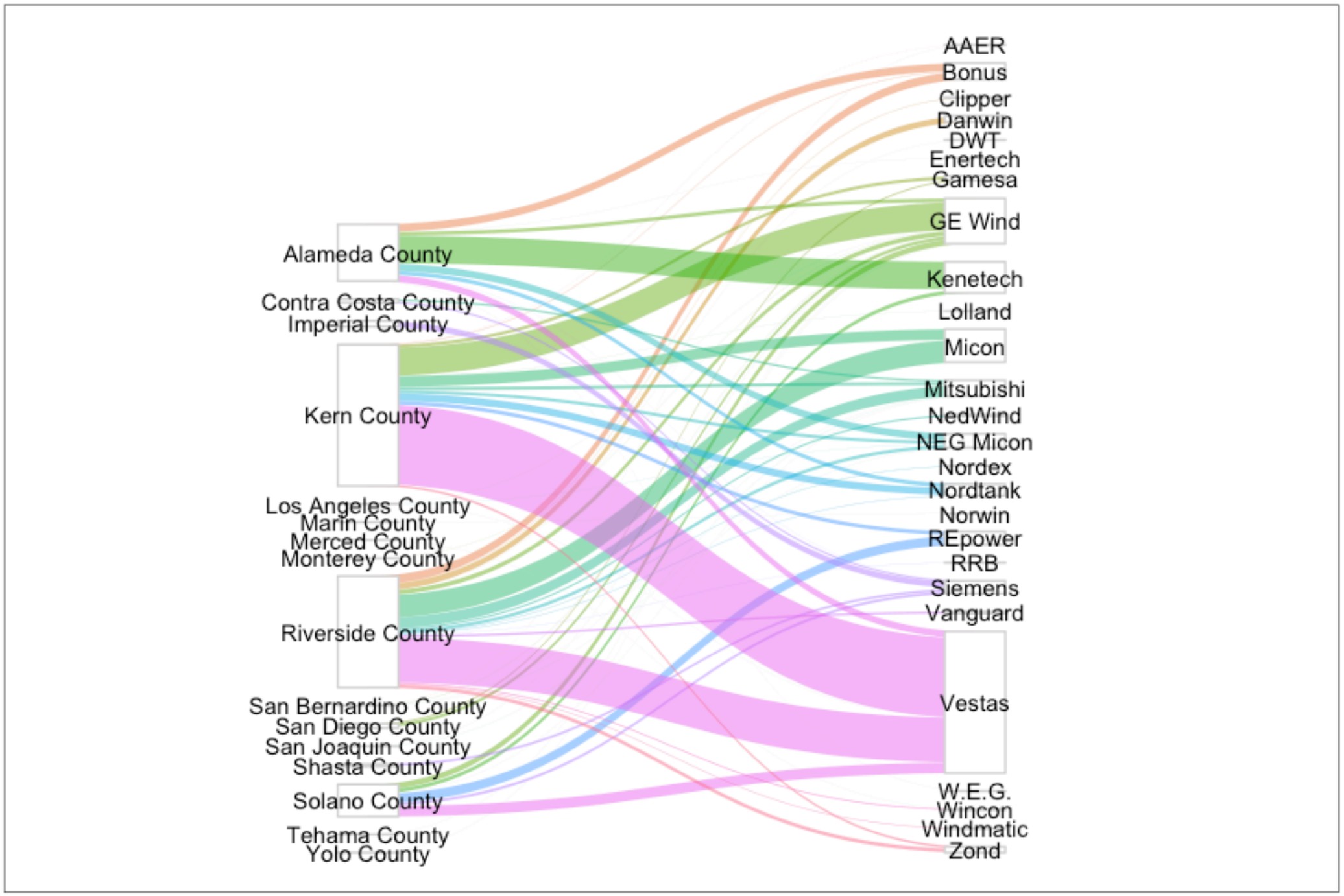
Another dataset



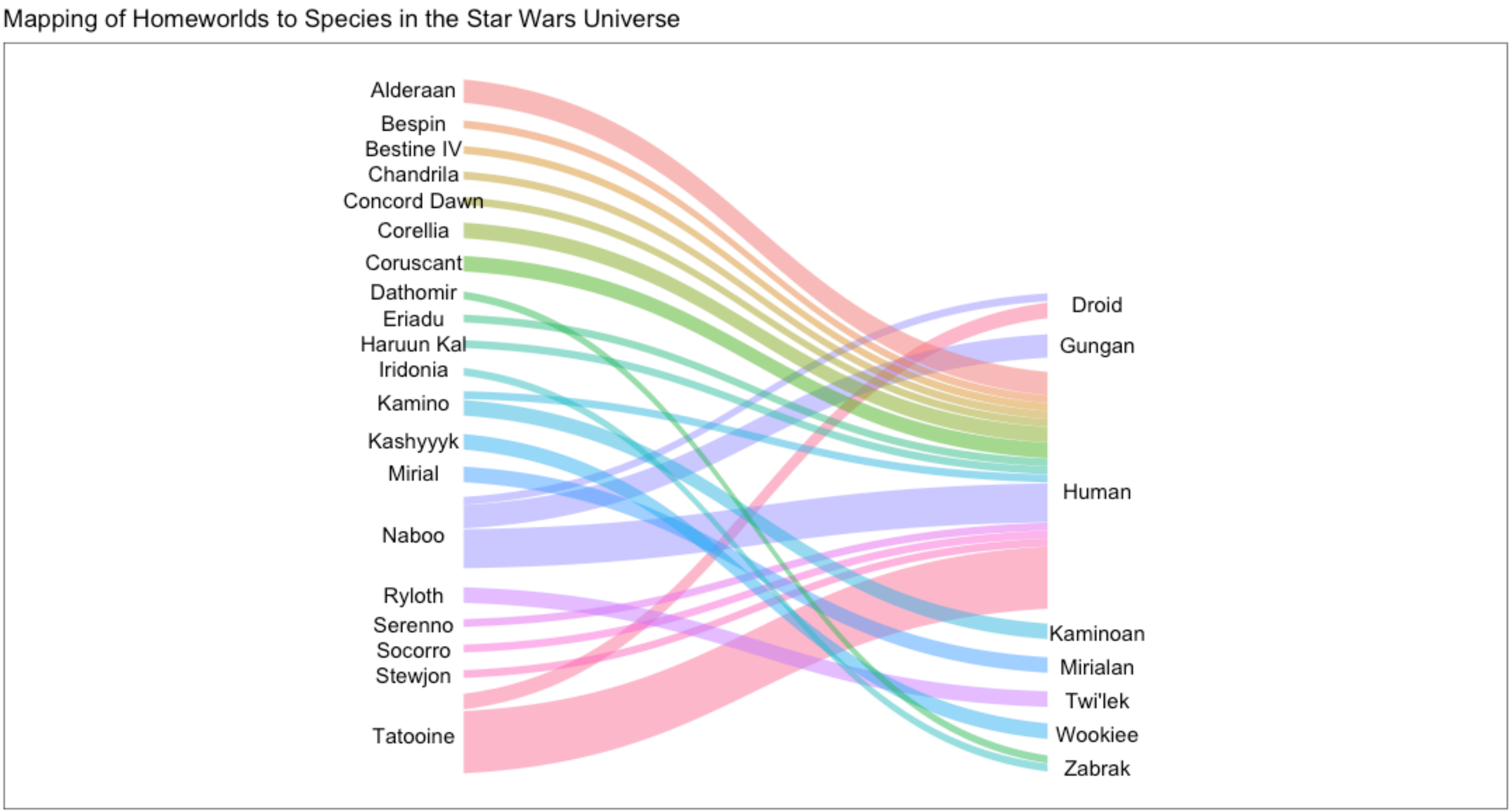
Focusing on California and Group by County



California - Mapping of Wind Turbine Manufacturer to County



The Star Wars dataset - with some minor aesthetics tweaks

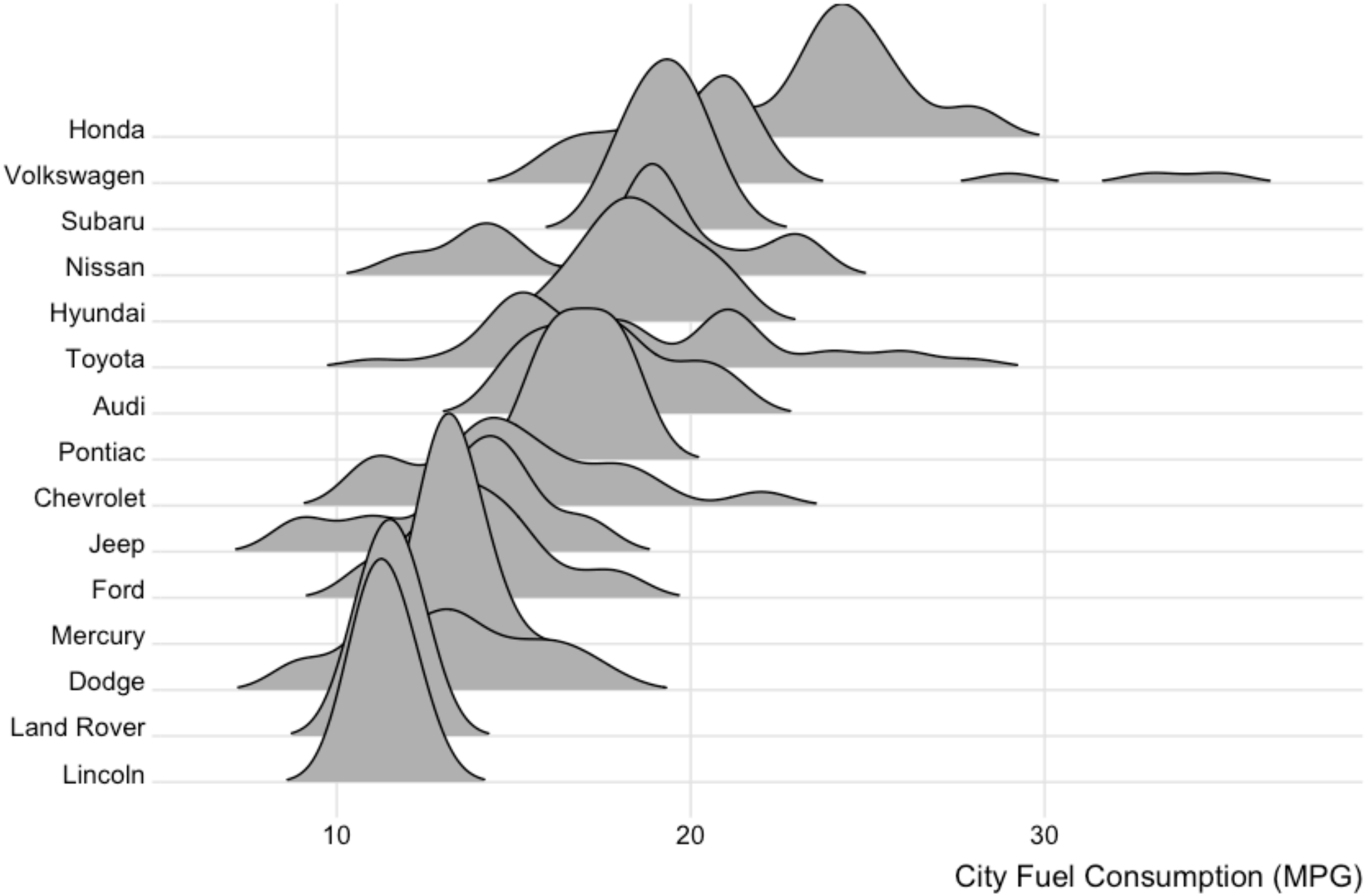


Ridge Plots with ggridges

Ridge Plots allow you to plot multiple histograms on the same group - good for quickly comparison different samples from a population (for example).

```
library(tidyverse)
library(ggridges)

mpg %>%
  mutate(manufacturer = str_to_title(manufacturer)) %>%
  ggplot(aes(x = cty, y = fct_reorder(manufacturer, cty), group =
manufacturer)) +
  geom_density_ridges(scale = 5, size = 0.5, rel_min_height = 0.01) +
  labs(title = "Ridge Plot of City Fuel Consumption for Different Car
Manufacturers",
       x = "City fuel consumption (mpg)",
       y = NULL) +
  theme_ridges()
```

Ridge Plot of City Fuel Consumption for Different Car Manufacturers



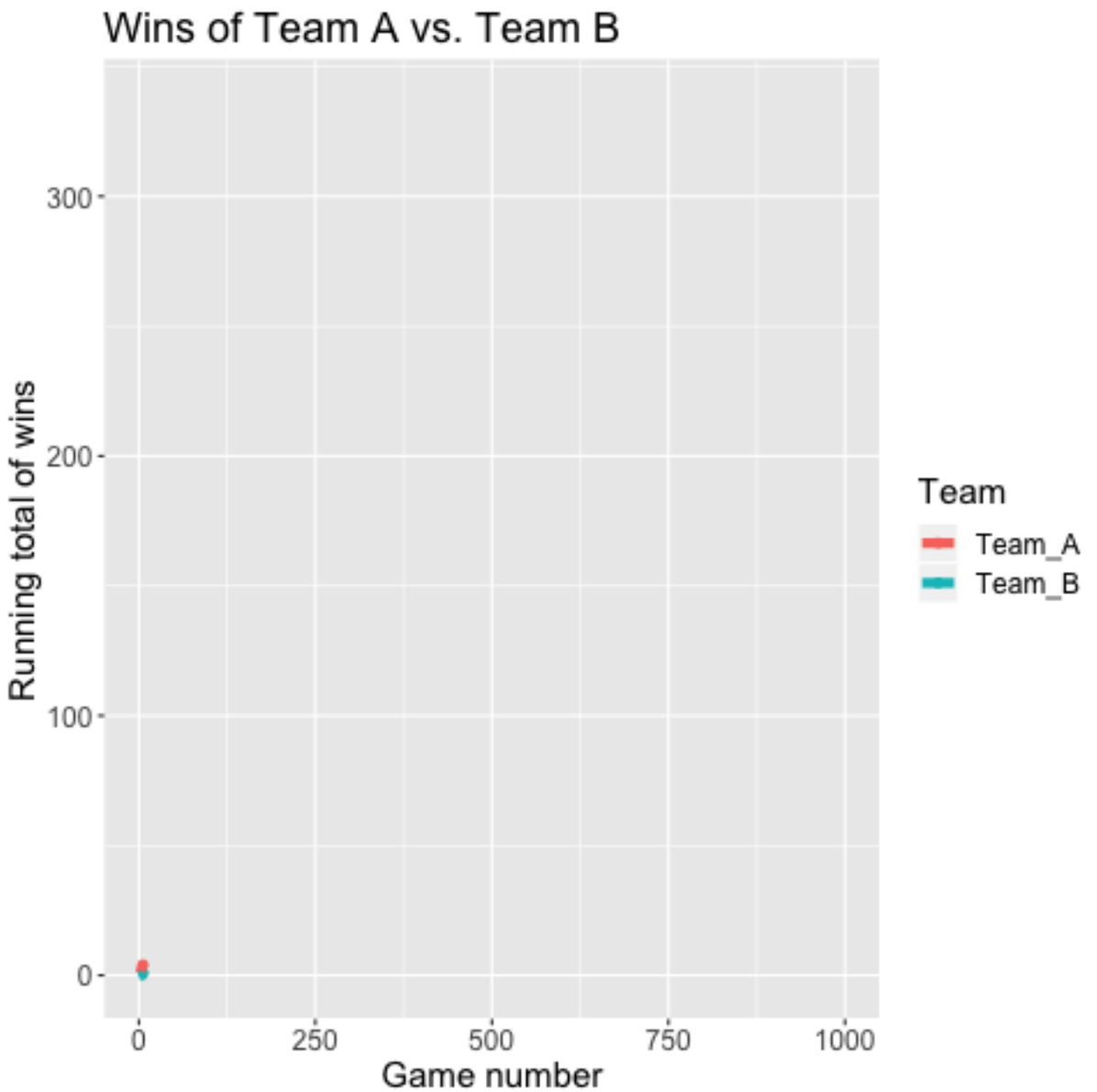
Animations - Hockey game simulation

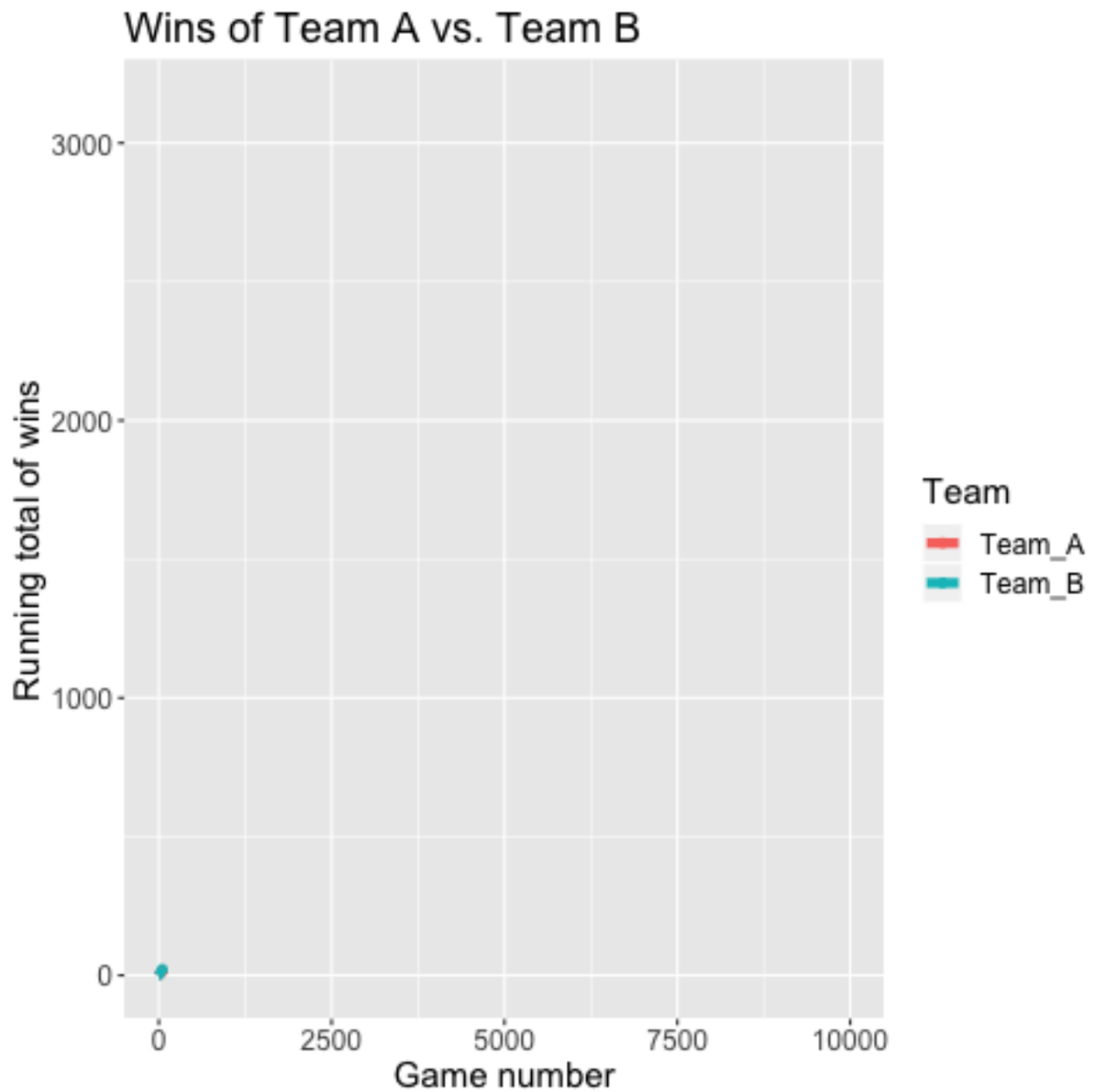
Imagine a hockey game where we know that Team A scores exactly 1 goal for sure and Team B takes 20 shots, each with a 5.5% chance of going in.

Which team would you rather be?

(nothing additional happens if you tie.)

Animation of the first 1,000 games on the left and 10,000 on the right.





We can code it to simulate the outcomes of 100,000 such games…

```
set.seed(1234)

team_b_goals <- NULL

for(i in 1:100000) {
  score <- sum(sample(c(1, 0), size = 20, replace = TRUE, prob = c(0.055, 1-.055)))
  team_b_goals <- c(team_b_goals, score)}

team_a_goals <- rep(1, 100000)

all_games <- as.tibble(cbind(team_a_goals, team_b_goals))
```

```
> nrow(filter(all_games, team_a_goals > team_b_goals))
[1] 32022
> nrow(filter(all_games, team_a_goals < team_b_goals))
[1] 30337
> nrow(filter(all_games, team_a_goals == team_b_goals))
[1] 37641
```

We see that out of 100,000 simulations, Team A wins on 32,022 occasions. Team B wins on 30,337 occasions and there are 37,641 ties.

The gganimate package

- The `gganimate` package needs to be installed separately from the core `tidyverse` packages.

- It follows the Tidyverse philosophy and extends the capabilities of the `ggplot()` function - in may ways it's like adding an extra layer to your plots in the same way you might use `facet_wrap()` but specifying parameters related to your animation frames (and how to transition between those frames).

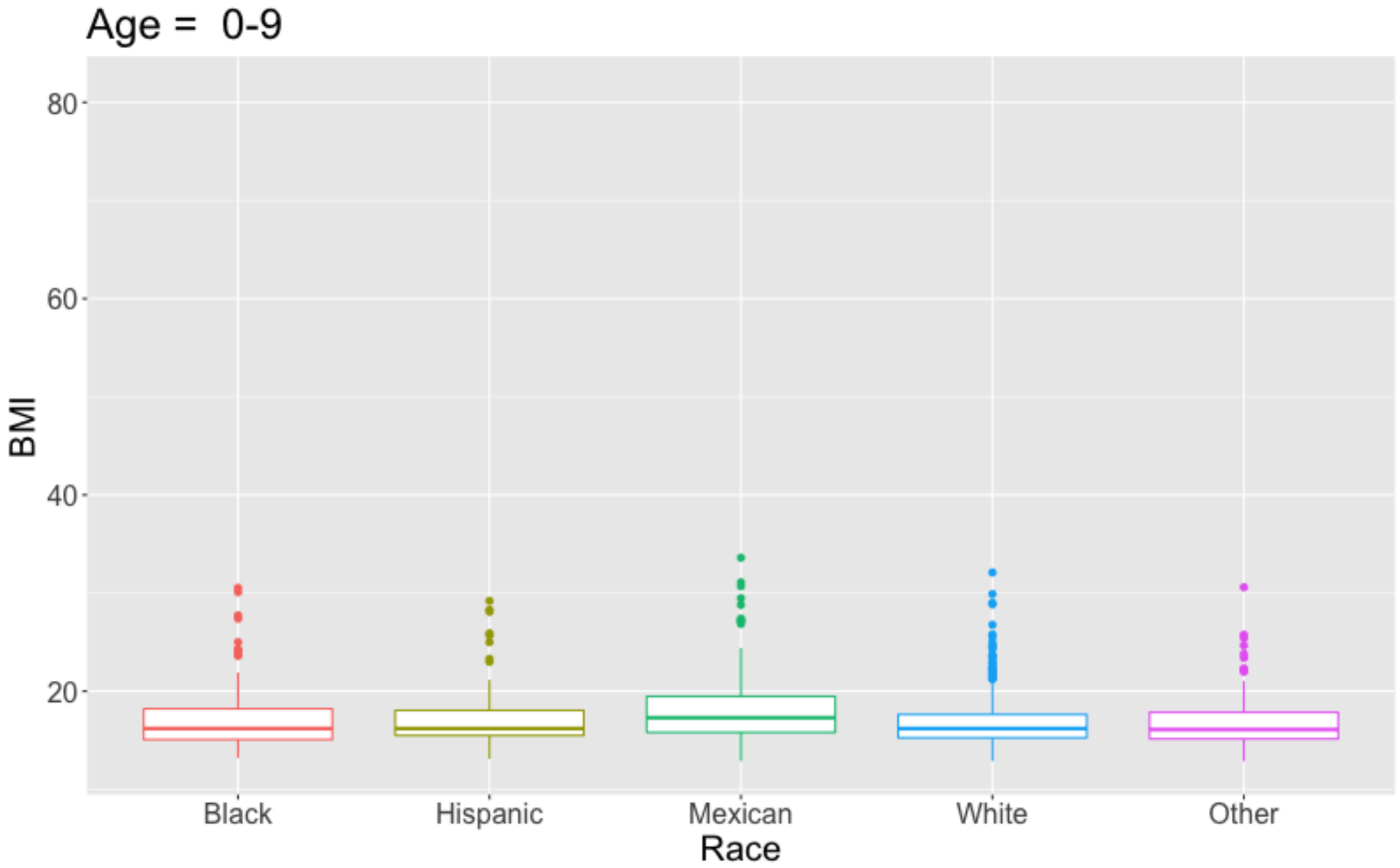
- Let's look at some examples…

Animating aspects of the NHANES dataset:

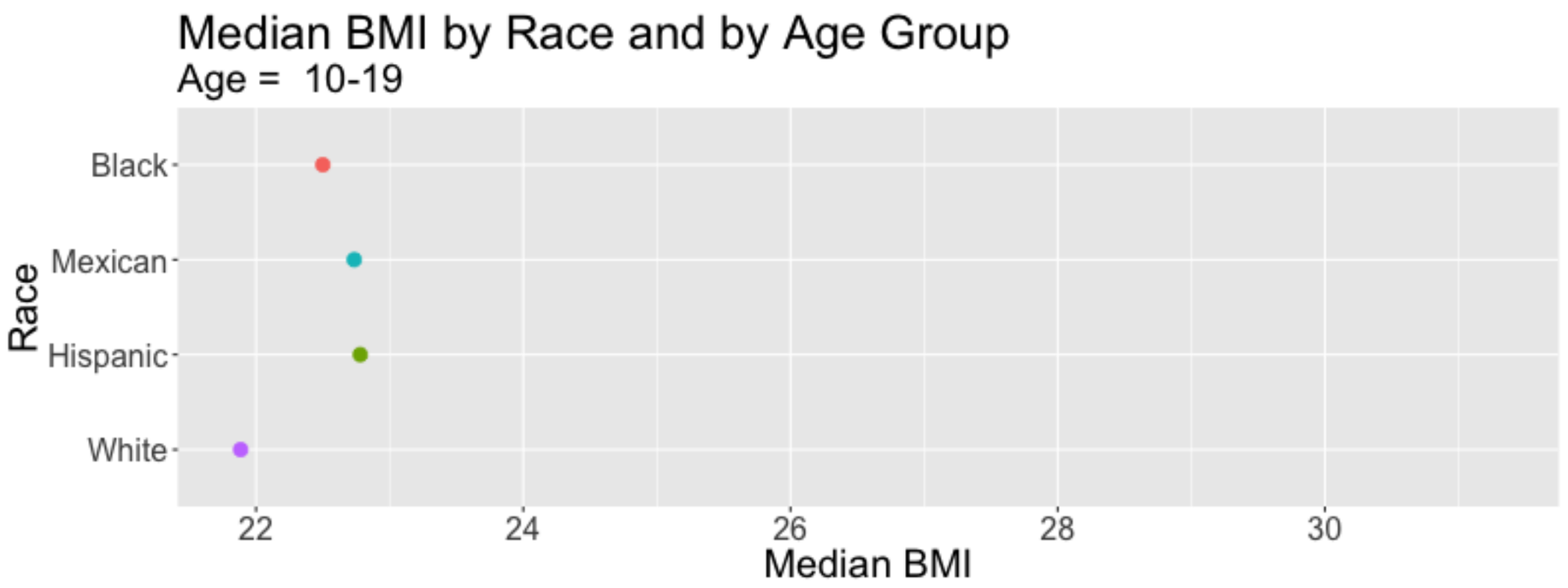
This is survey data collected by the US National Center for Health Statistics (NCHS) which has conducted a series of health and nutrition surveys since the early 1960's. Since 1999 approximately 5,000 individuals of all ages are interviewed in their homes every year and complete the health examination component of the survey. The health examination is conducted in a mobile examination centre (MEC).

```
library(NHANES)

# Boxplot of BMI by Race and AgeDecade
NHANES %>%
  distinct(ID, .keep_all = TRUE) %>%
  ggplot(aes(x = Race1, y = BMI, colour = Race1)) +
  geom_boxplot() +
  guides(colour = FALSE) +
  labs(x = "Race", title = "Age = {closest_state}") +
  transition_states(AgeDecade)
```

- Think of `transition_states()` like `facet_wrap()` with the transition between each panel animated. The variable `{closest_state}` is available from the `transition_states()` function and can be used outside the function (like I am in the title here).


Age = 0-9
80
60
40
20
BMI
Black
Hispanic
Mexican
White
Other
Race

```
NHANES_tidy <- NHANES %>%
  filter(Race1 != "Other") %>%
  filter(as.character(AgeDecade) != " 0-9")

my_plot <- NHANES_tidy %>%
  mutate(AgeDecade = fct_drop(AgeDecade," 0-9")) %>%
  group_by(AgeDecade, Race1) %>%
  summarise(median_BMI = median(BMI, na.rm = TRUE)) %>%
  ggplot(aes(x = median_BMI, y = reorder(Race1, median_BMI), colour = Race1)) +
  geom_point(size = 3) +
  labs(x = "Median BMI", y = "Race", title = "Median BMI by Race and byAge Group",
       subtitle = "Age = {closest_state}") +
  transition_states(AgeDecade) +
  theme(text = element_text(size = 20)) +
  guides(colour = FALSE)

animate(my_plot, height = 300, width = 800)
anim_save("example_plot.gif")
```



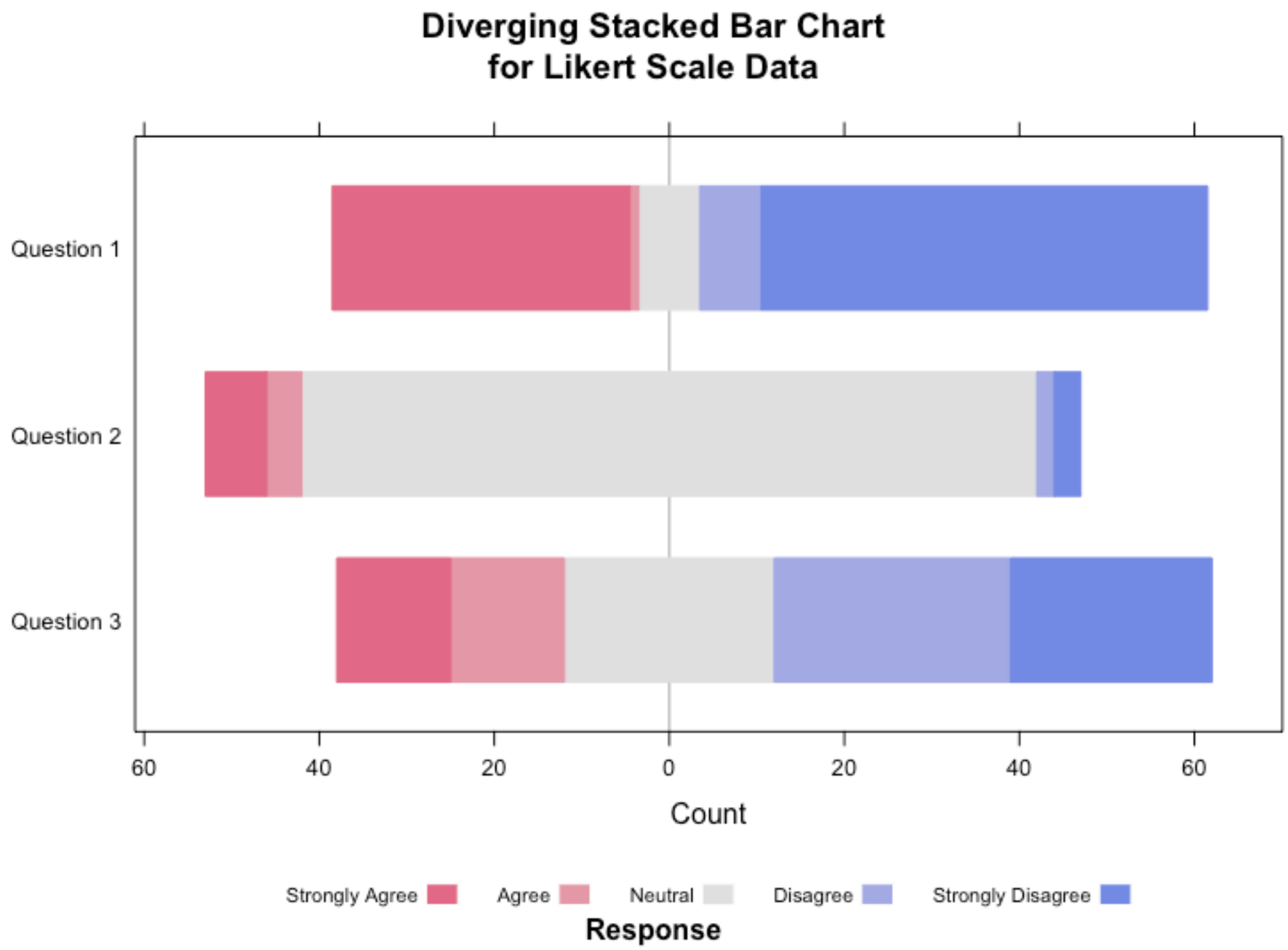
Visualising Likert Scale Data

Imagine you asked 100 people to respond to three questions using a 5-point Likert scale - their data might look like this:

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
Question 1	34	1	7	7	51
Question 2	7	4	84	2	3
Question 3	13	13	24	27	23

How might you best communicate the data? It might be misleading to report a measure of central tendency as the mean, median, and mode for Question 2 will all suggest people were Neutral (and ignore those in the tails).

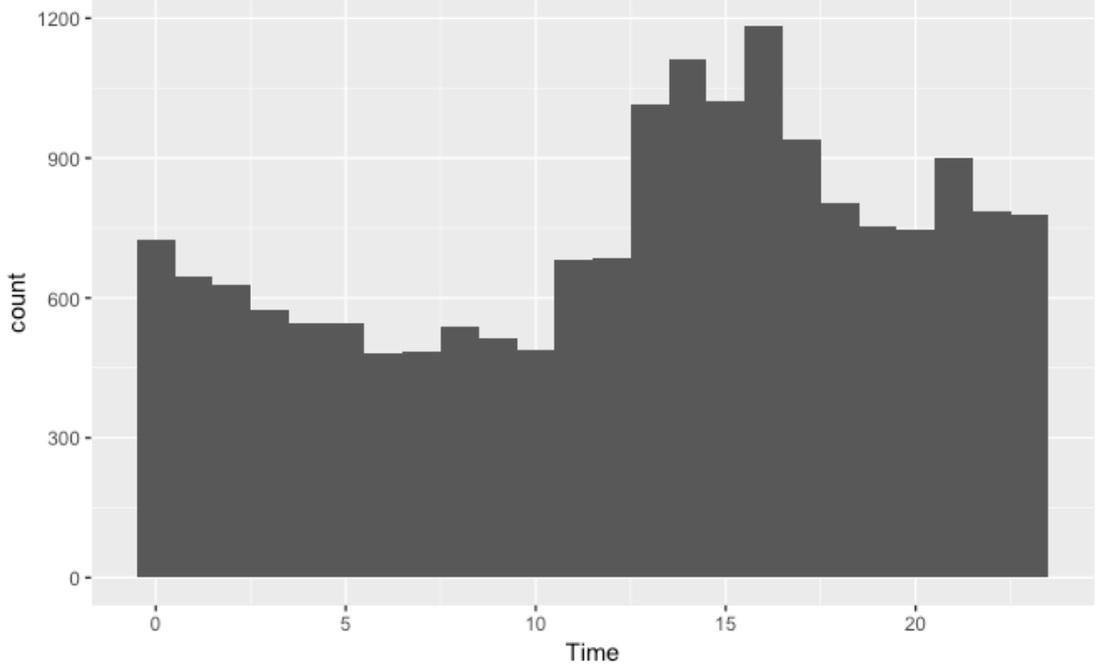
One option is via a diverging stacked bar chart using the `likert()` function in the package `HH`...


Diverging Stacked Bar Chart
for Likert Scale Data
Question 1
Question 2
Question 3
60
40
20
0
20
40
60
Count
Strongly Agree
Agree
Neutral
Disagree
Strongly Disagree
Response

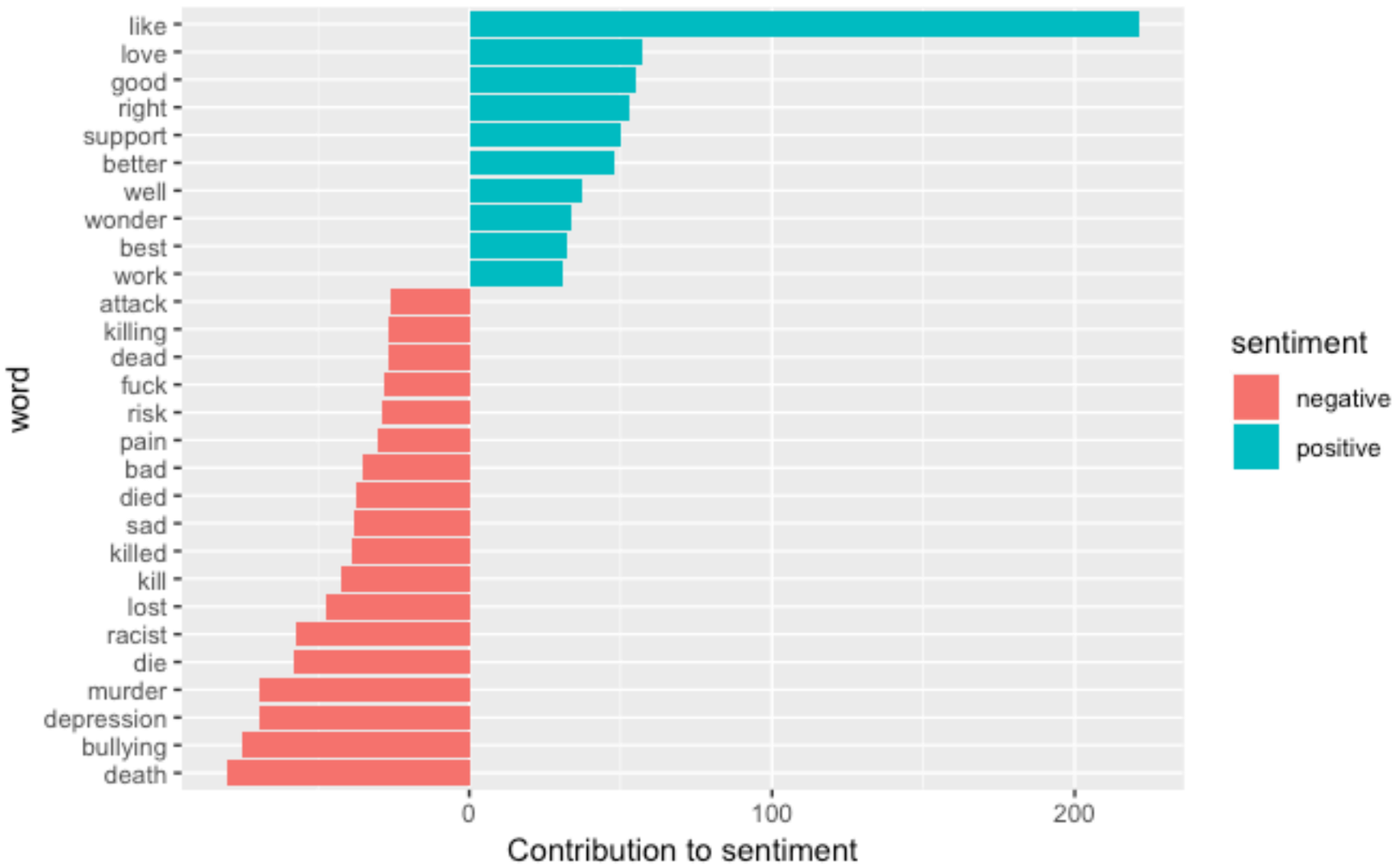
Visualising Text Data

- We can use the `rtweet` package by Mike Kearney to scrape data from Twitter using the `search_tweets()` function. In this case I'm scraping Twitter for mentions of the word 'suicide' in Tweets, extracting the time the Tweet was created and then plotting on a histogram.

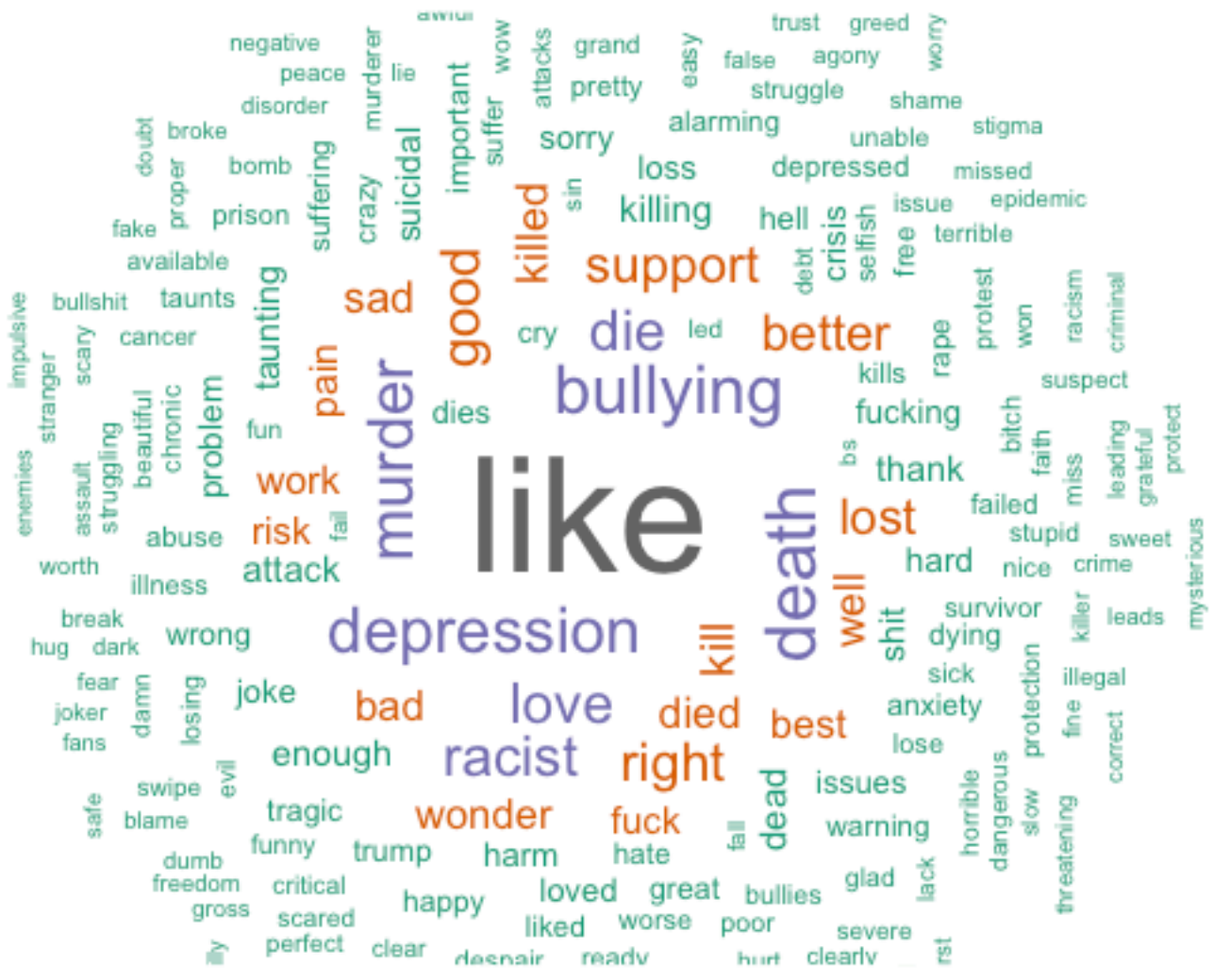
```
tweets <- search_tweets(q = "suicide", n =
1000, include_rts = FALSE, retryonratelimit =
TRUE)

time <- tibble(Time = hour(tweets$created_at))

time %>%
  filter(!is.na(Time)) %>%
  ggplot(aes(x = Time)) +
  geom_histogram()
```



- Now I'm using the `tidytext` package to do a sentiment analysis associated with the words in Tweets mentioning 'suicide' created between midnight and 6AM.



- And visualising the content of the Tweets as a Wordcloud using the `wordcloud` package.

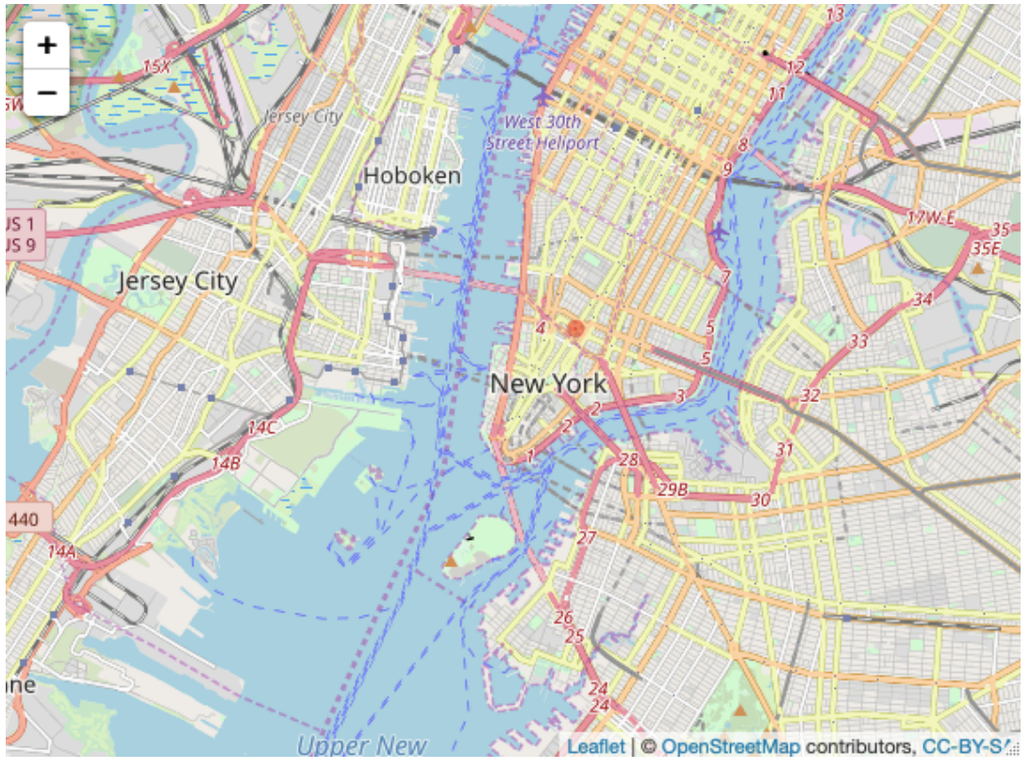


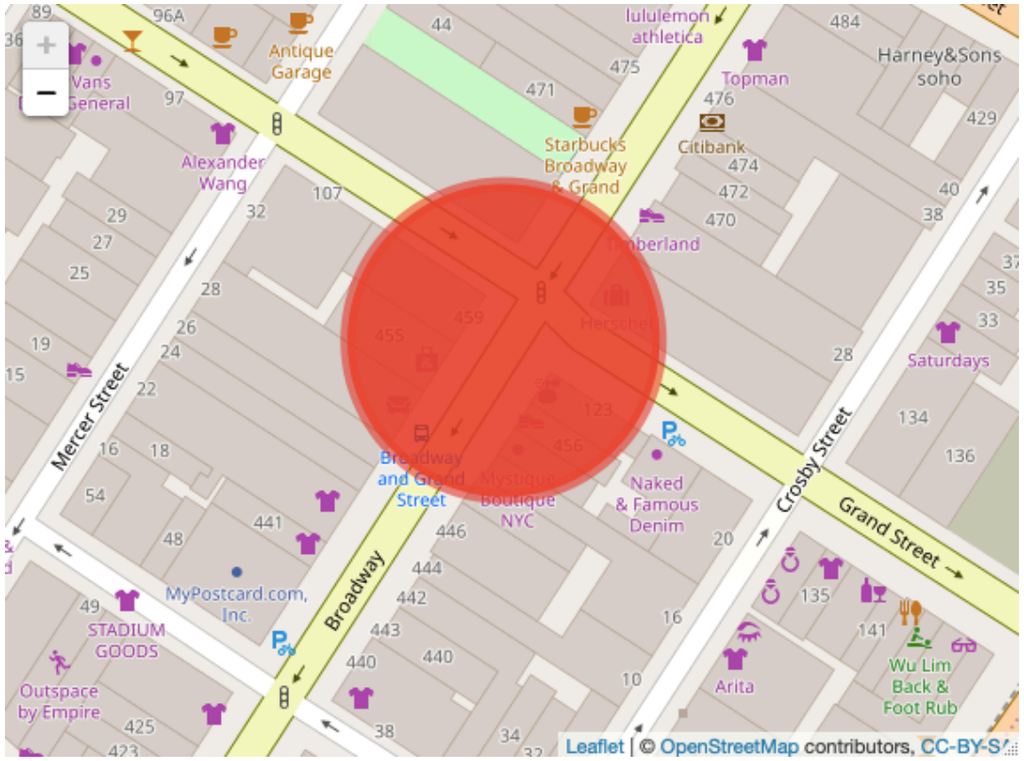
Searching for mentions of “Opeth”

Geospatial plotting

- Some Tweets have associated with them the latitude and longitude of where they were tweeted from - we can use the `leaflet` package to extract these coordinates and plot the location of tweets with geospatial tagging on a map…

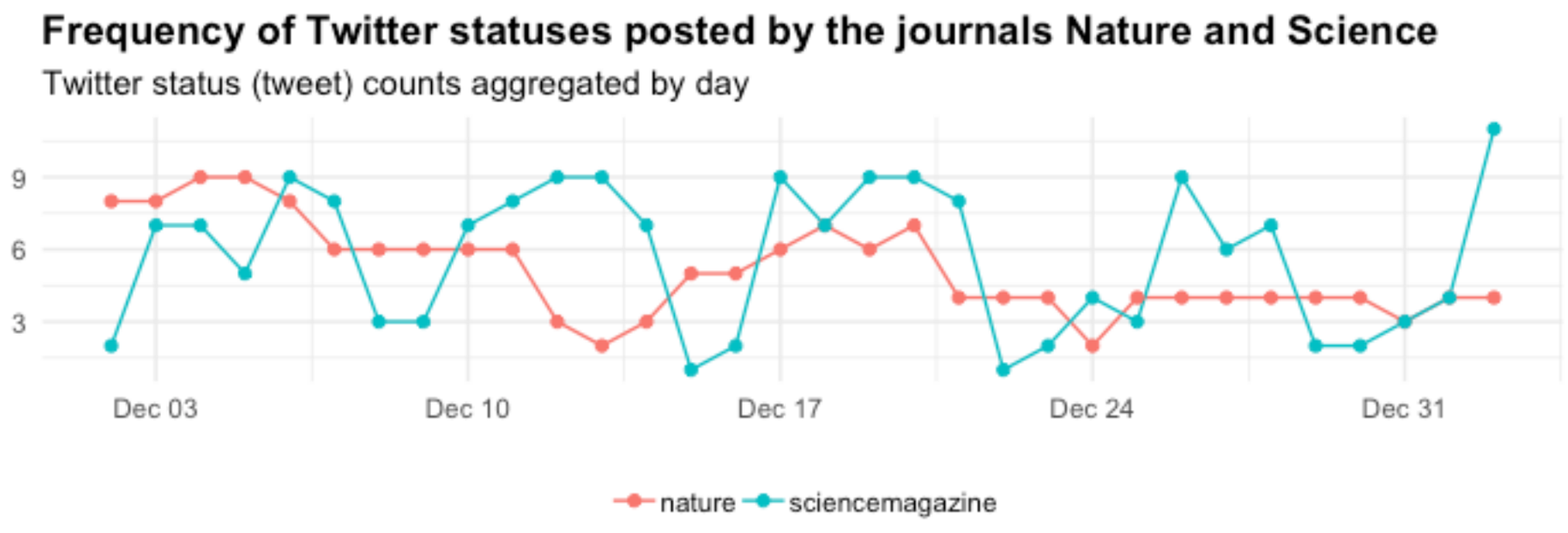
…and the map is zoomable



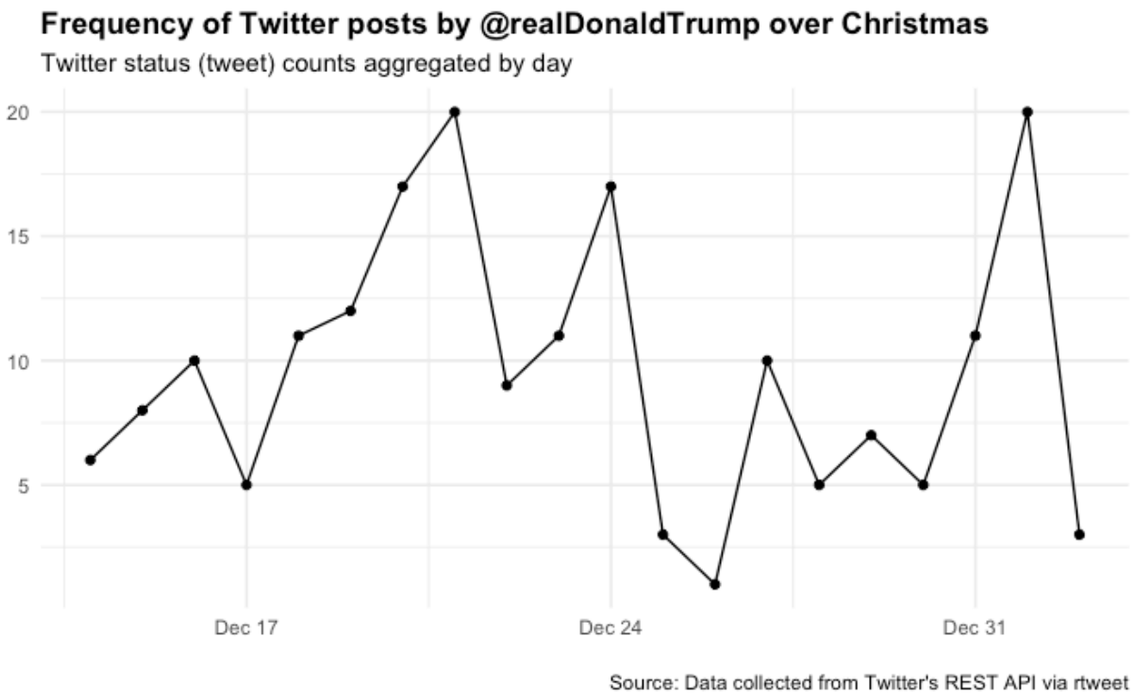


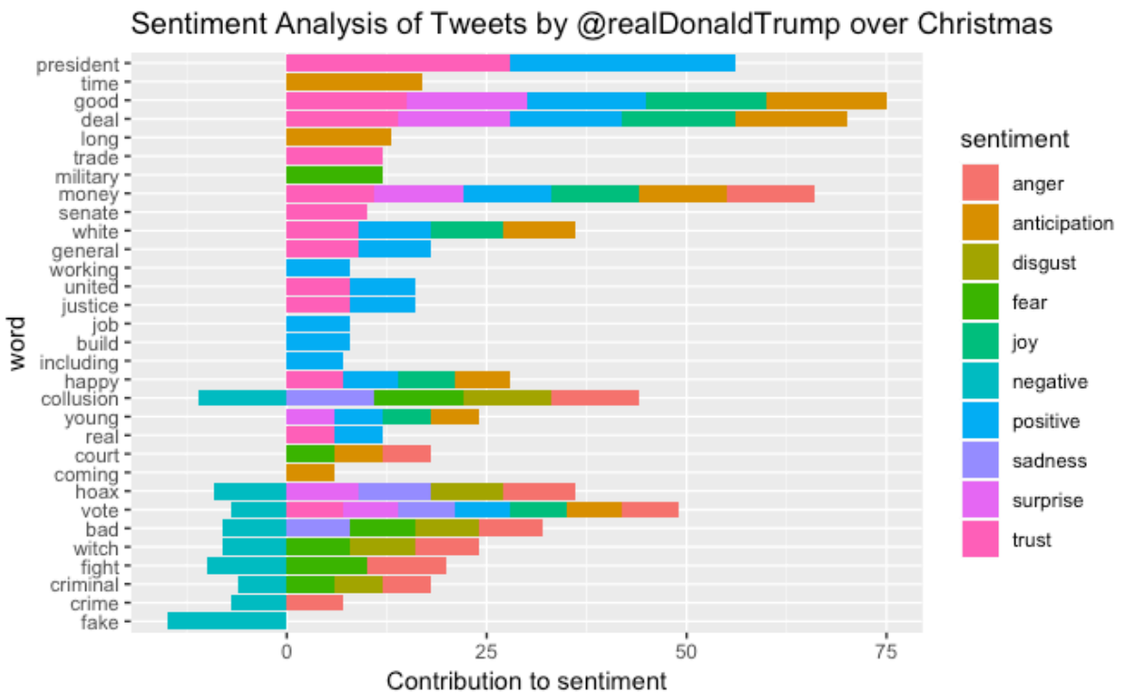
Collecting the number of Tweets from two timelines

```
tmls <- get_timelines(c("Nature", "sciencemagazine"), n = 1000)
tmls %>%
  filter(created_at > "2018-12-1") %>%
  group_by(screen_name) %>%
  ts_plot("days", trim = 1L) +
  geom_point() +
  theme_minimal() +
  theme(
    legend.title = ggplot2::element_blank(),
    legend.position = "bottom",
    plot.title = ggplot2::element_text(face = "bold")) +
  labs(
    x = NULL, y = NULL,
    title = "Frequency of Twitter statuses posted by the journals Nature and Science",
    subtitle = "Twitter status (tweet) counts aggregated by day",
    caption = "\nSource: Data collected from Twitter's REST API via rtweet"
)
```

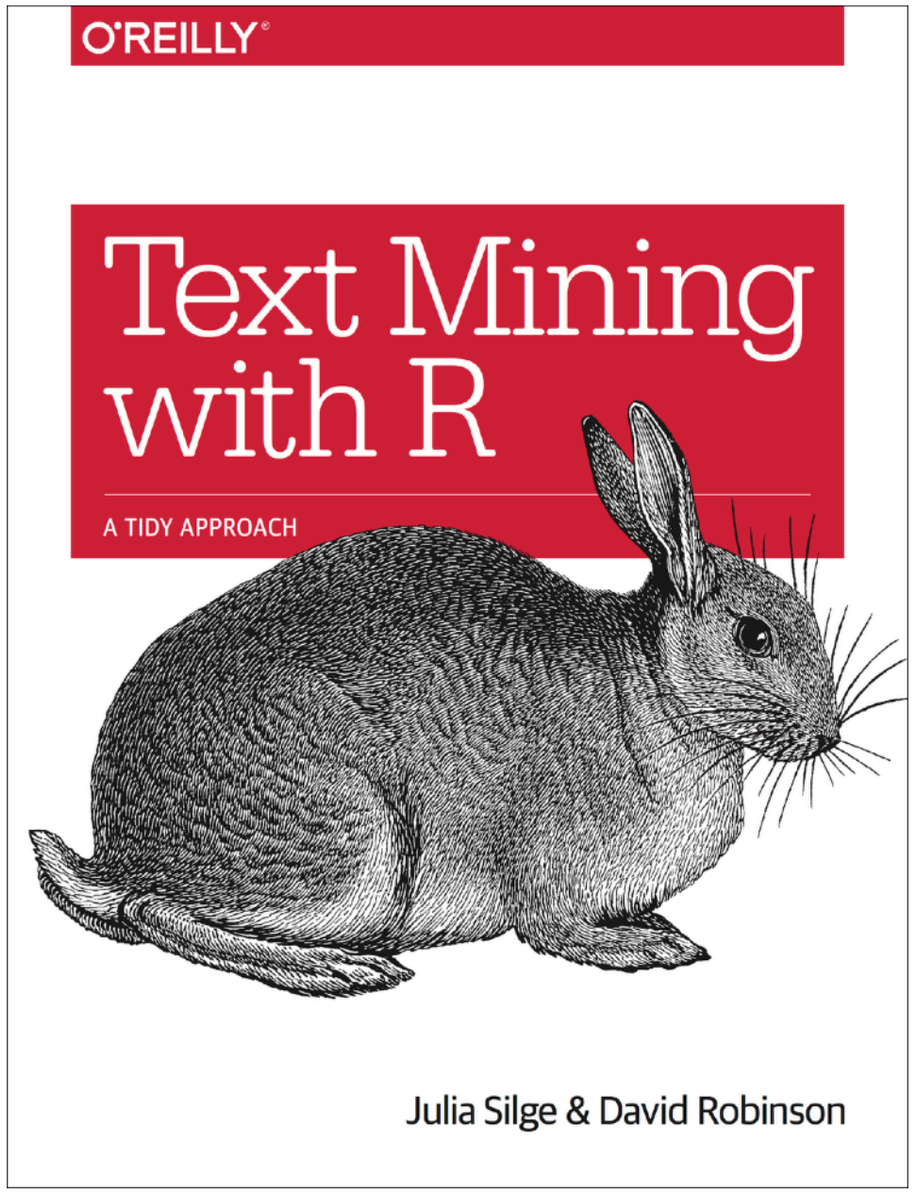


Collecting Tweets from one timeline





A good, detailed text mining book for those working on qualitative projects…



https://www.tidytextmining.com

Reading in texts using the `gutenbergr` package

```
titles <- c("Twenty Thousand Leagues under the Sea", "The War of the Worlds")
books <- gutenberg_works(title %in% titles) %>%
  gutenberg_download(meta_fields = "title")
```

```
> books
# A tibble: 18,609 x 3
   gutenberg_id text                                                    title
          <int> <chr>                                                   <chr>
 1           36 The War of the Worlds                                   The War of the Worlds
 2           36 ""                                                      The War of the Worlds
 3           36 by H. G. Wells [1898]                                   The War of the Worlds
 4           36 ""                                                      The War of the Worlds
 5           36 ""                                                      The War of the Worlds
 6           36 "     But who shall dwell in these worlds if they be"   The War of the Worlds
 7           36 "     inhabited? .  .  .  Are we or they Lords of the"  The War of the Worlds
 8           36 "     World? .  .  .  And how are all things made for man?--" The War of the Worlds
 9           36 "          KEPLER (quoted in The Anatomy of Melancholy)" The War of the Worlds
10           36 ""                                                      The War of the Worlds
# … with 18,599 more rows
```

- In the above code I read in two books using the `gutenbergr` package which reads them from the Project Gutenberg library (containing over 58,000 free books). The object `books` contains the text of both books contained in the “text” column

```
text_waroftheworlds <- books %>%
  filter(title == "The War of the Worlds") %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)

text_underthesea <- books %>%
  filter(title == "Twenty Thousand Leagues under the Sea") %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

- I then filter by book title and ‘unnest’ the text of each book so that we have one column in each of two dataframes corresponding to the words in each book…

```
> text_waroftheworlds
# A tibble: 60,513 x 3
   gutenberg_id title                     word
          <int> <chr>                     <chr>
 1           36 The War of the Worlds the
 2           36 The War of the Worlds war
 3           36 The War of the Worlds of
 4           36 The War of the Worlds the
 5           36 The War of the Worlds worlds
 6           36 The War of the Worlds by
 7           36 The War of the Worlds h
 8           36 The War of the Worlds g
 9           36 The War of the Worlds wells
10           36 The War of the Worlds 1898
# … with 60,503 more rows
```

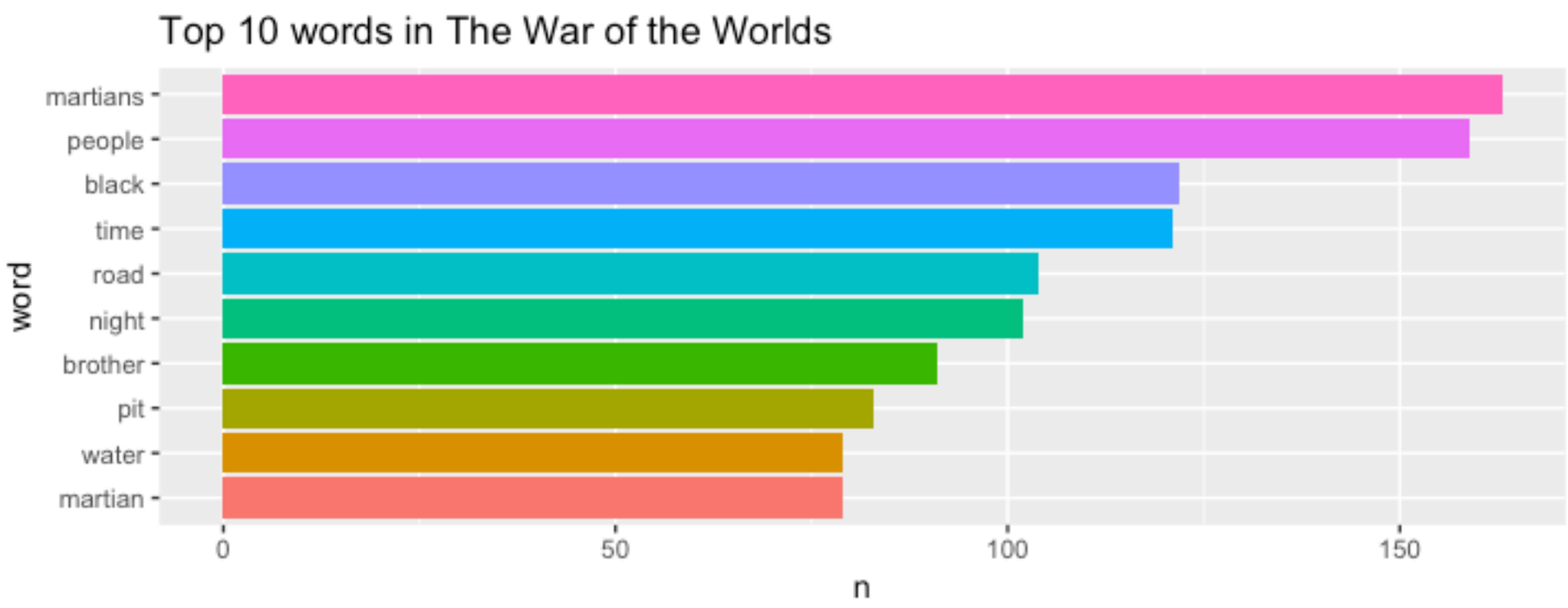
- If I hadn't use the `anti_join(stopwords)` call I would end up with a data frame like the above containing lots of common function words.

- But by adding the `anti_join(stopwords)` line to my code I exclude all stop words (the common function words) and end up with:

```
> text_waroftheworlds
# A tibble: 22,583 x 3
   gutenberg_id title                     word
          <int> <chr>                     <chr>
 1           36 The War of the Worlds war
 2           36 The War of the Worlds worlds
 3           36 The War of the Worlds 1898
 4           36 The War of the Worlds dwell
 5           36 The War of the Worlds worlds
 6           36 The War of the Worlds inhabited
 7           36 The War of the Worlds lords
 8           36 The War of the Worlds world
 9           36 The War of the Worlds kepler
10           36 The War of the Worlds quoted
# … with 22,573 more rows
```

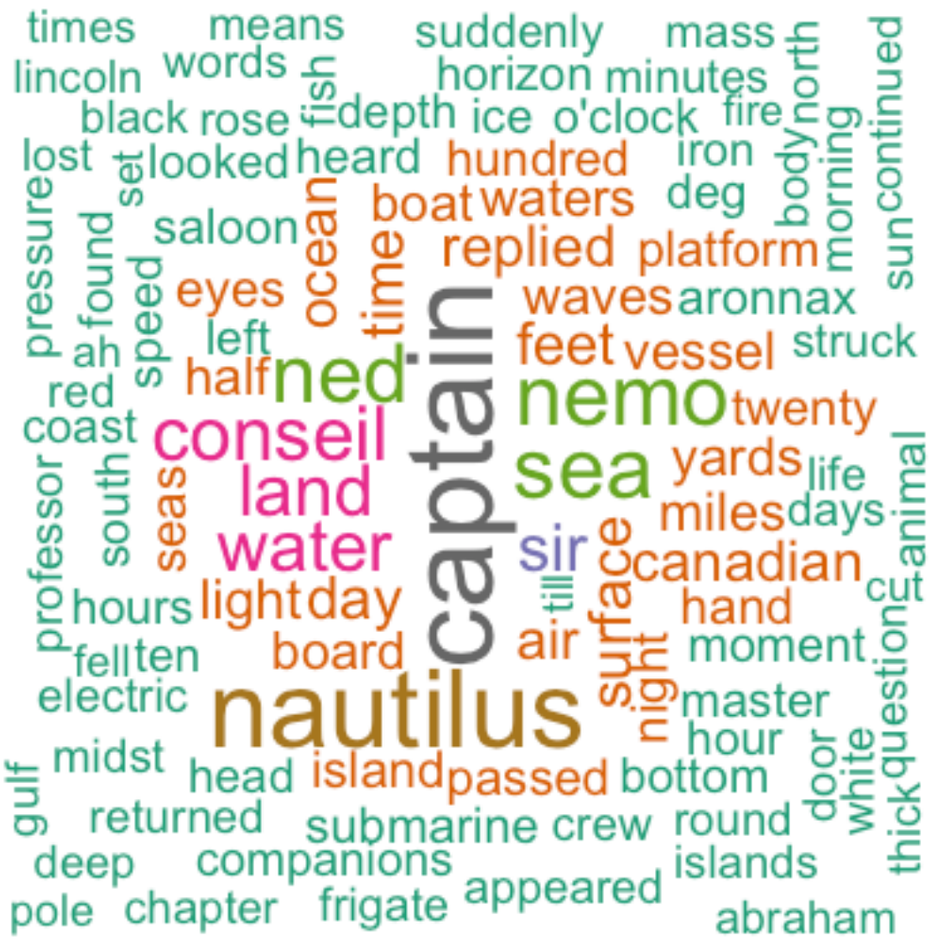
- I can the plot the top 10 most common words in The War of the Worlds with this code:

```
text_waroftheworlds %>%
  count(word) %>%
  top_n(10) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(x = word, y = n, fill = word)) +
  geom_col() +
  coord_flip() +
  guides(fill = FALSE) +
  labs(title = "Top 10 word in The War of the Worlds")
```



- Plotting a wordcloud based on the words in 20,000 Leagues Under the Sea.

```
text_underthesea_count <- text_underthesea %>%
  count(word) %>%
  top_n(200)

wordcloud(words = text_underthesea_count$word,
          freq = text_underthesea_count$n,
          min.freq = 1,
          scale = c(3, 1),
          max.words = 200,
          random.order = FALSE,
          rot.per = 0.35,
          colors = brewer.pal(8, "Dark2"))
```



BBC-style Visualisations

- The BBC have published their R graphics cookbook for generating data visualisations following the BBC style guide.

- The cookbook can be found here:

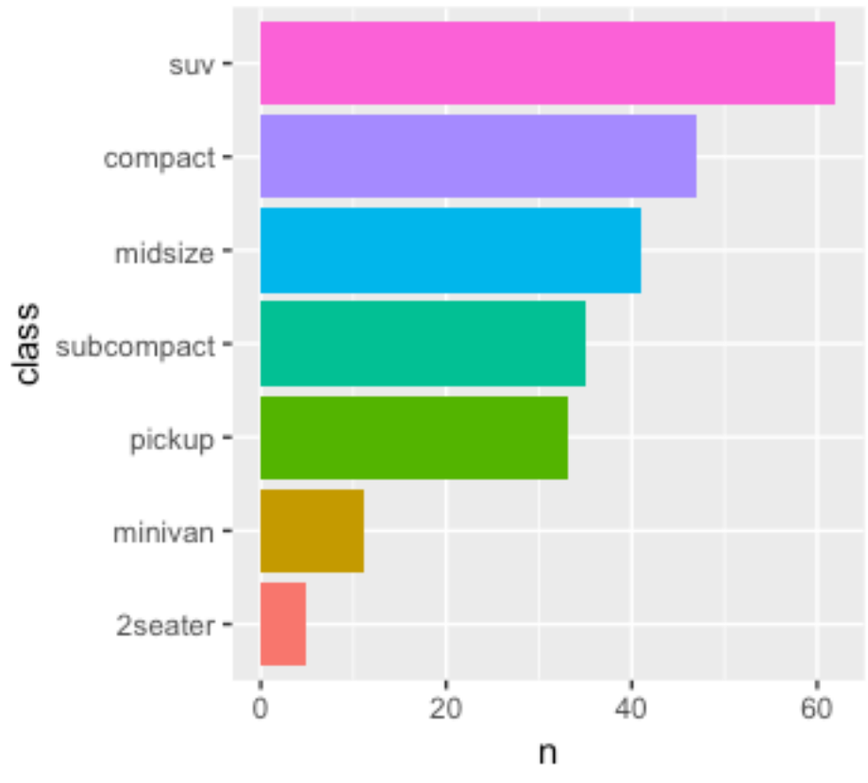
  `https://bbc.github.io/rcookbook/#how does the bbplot package work`

- With the `bbplot` package containing the functions used to generate BBC-style visualisations available on GitHub:

  `https://github.com/bbc/bbplot`

- Let's start with a basic plot - we're going to use the built-in `mpg` dataset.

```
devtools::install_github('bbc/bbplot')

library(tidyverse)
library(bbplot)

my_plot <- mpg %>%
  count(class) %>%
  mutate(class = fct_reorder(class, n)) %>%
  ggplot(aes(x = class, y = n, fill = class)) +
  geom_col() +
  coord_flip() +
  guides(fill = FALSE)

my_plot
```



- We can add an extra parameter to our ggplot code to set the BBC theme using `bbc_style()`:

```
my_plot <- mpg %>%
  count(class) %>%
  mutate(class = fct_reorder(class, n)) %>%
  ggplot(aes(x = class, y = n, fill = class)) +
  geom_col() +
  coord_flip() +
  guides(fill = FALSE)  +
  bbc_style()+
  labs(title="Number of each type of car",
       subtitle = "Data collected in the US, 1999-2008”)
```