Andrew's Two-Day R Course
Day One

Andrew Stewart

Andrew.Stewart@manchester.ac.uk

@ajstewart_lang

Plan for Today

- This morning you're going to learn about the importance of Open Science and how research in the biomedical sciences in response to issues around replication and reproducibility.

- You'll learn the role that R can play in reproducible research.

- We will focus on the Tidyverse R packages for data tidying, data wrangling, generating summary statistics, and data visualisation (incl. animations).

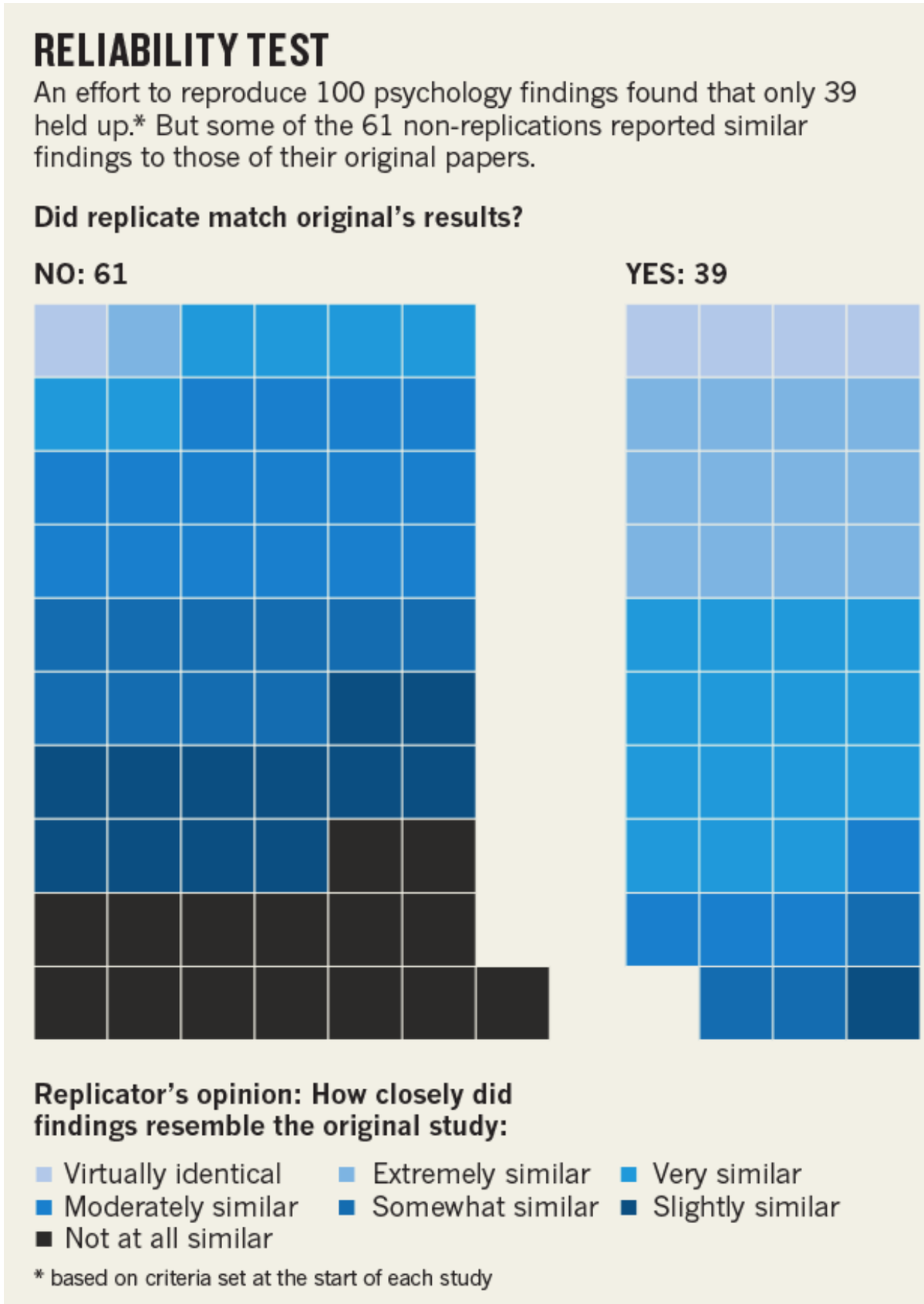
- This afternoon you'll be programming in R practicing all of the above!

Replication and Reproducibility in Science

- Ionnidis (2005), *PLOS Medicine*, most published research findings are false.
- Prinz et al. (2011), *Nature Reviews Drug Discovery*, around 65% of cancer biology studies do not replicate.
- Button et al. (2013), *Nature Reviews Neuroscience*, small sample size undermines the reliability of neuroscience.
- MacLeod et al. (2014), *Lancet*, 85% of biomedical research resources are wasted.
- Baker (2015), *Nature*, 90% of scientists recognise a ‘reproducibility crisis’.
- Nosek & Errington (2017), *eLife*, out of first 5 replication attempts of preclinical cancer biology work, only 2 have replicated.

(In)famous studies…

- Power posing
- Ego depletion
- Social priming
- Marshmallow test performance predicts future achievement
- Stanford prison experiment
- Growth mindset
- Learning styles
- Any others you know of?



270 authors tried to replicate 100 experiments drawn from high profile Psychology journals - *Psychological Science*, *Journal of Personality and Social Psychology*, and *Journal of Experimental Psychology: Learning, Memory, and Cognition*.

- Button et al. (2013), *Nature Reviews Neuroscience*, small sample size undermines the reliability of neuroscience. Nord et al., (2017), *Journal of Neuroscience,* highlight wide heterogeneity in power in neuroscience studies.

Table 2. Median, maximum, and minimum power subdivided by study type

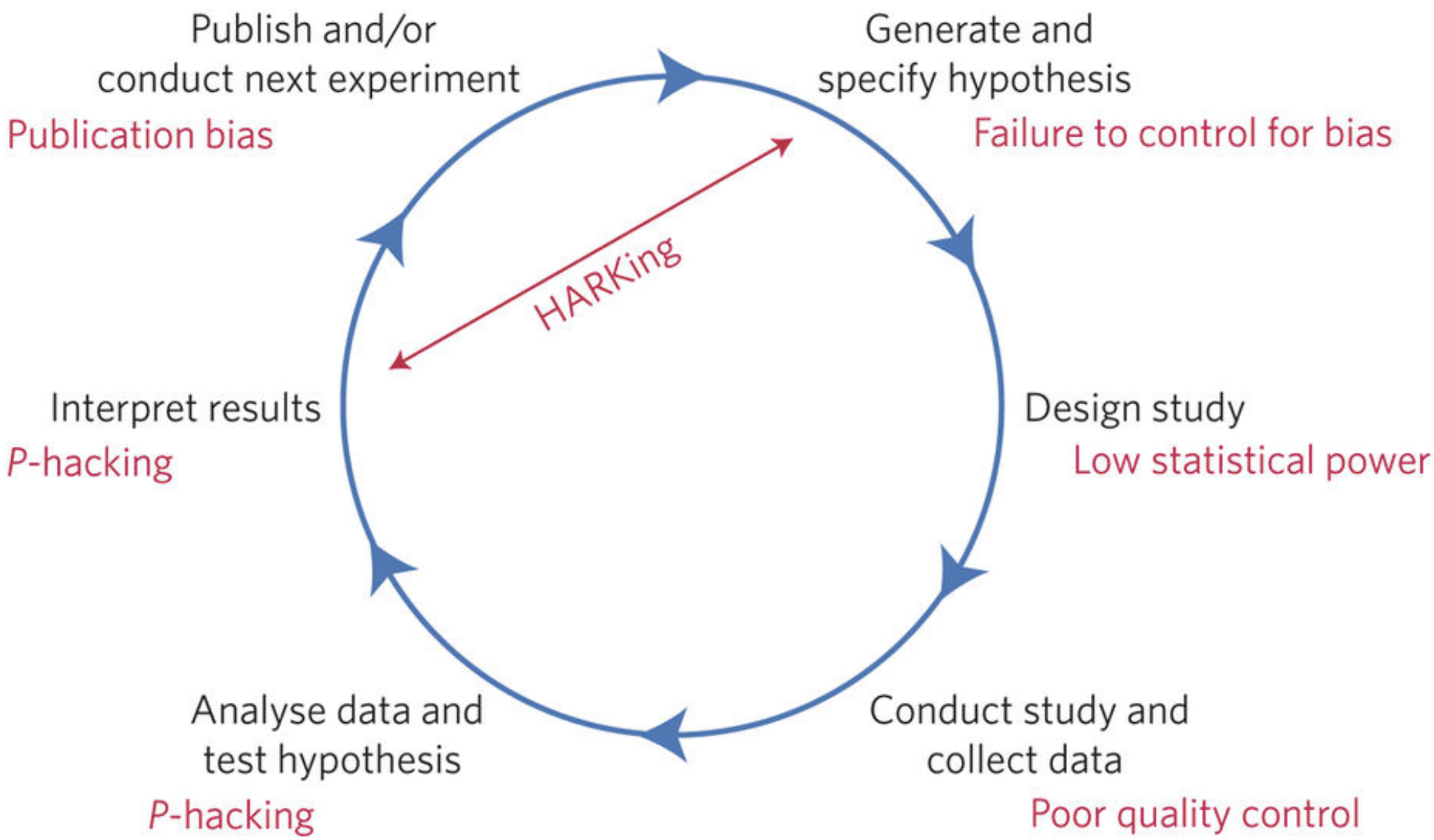
Group	Median power (%)	Minimum power (%)	Maximum power (%)	2.5th and 97.5th percentile (based on raw data)	95% HDI (based on GMMs)	Total N
All studies	23	0.05	1	0.05–1.00	0.00 – 0.72, 0.80 – 1.00	730
All studies excluding null	30	0.05	1	0.05–1.00	0.01 – 0.73, 0.79 – 1.00	638
Genetic	11	0.05	1	0.05 – 0.94	0.00 – 0.44, 0.63 – 0.93	234
Treatment	20	0.05	1	0.05–1.00	0.00 – 0.65, 0.91–1.00	145
Psychology	50	0.07	1	0.07–1.00	0.02 – 0.24, 0.28 – 1.00	198
Imaging	32	0.11	1	0.11–1.00	0.03 – 0.54, 0.71–1.00	65
Neurochemistry	47	0.07	1	0.07–1.00	0.02 – 0.79, 0.92–1.00	50
Miscellaneous	57	0.11	1	0.11–1.00	0.09 – 1.00	38

Is there not just “good science” and “bad science”?

Without realising it, good scientists have been engaging in questionable research practices (QRPs)…

Problems include *p*-hacking, lack of power, HARKing, failing (refusal) to share data and code, too many researcher degrees of freedom…

From: A manifesto for reproducible science



Munafo et al. (2017), *Nature Human Behaviour*

Replicable Science ≠ Reproducible Science

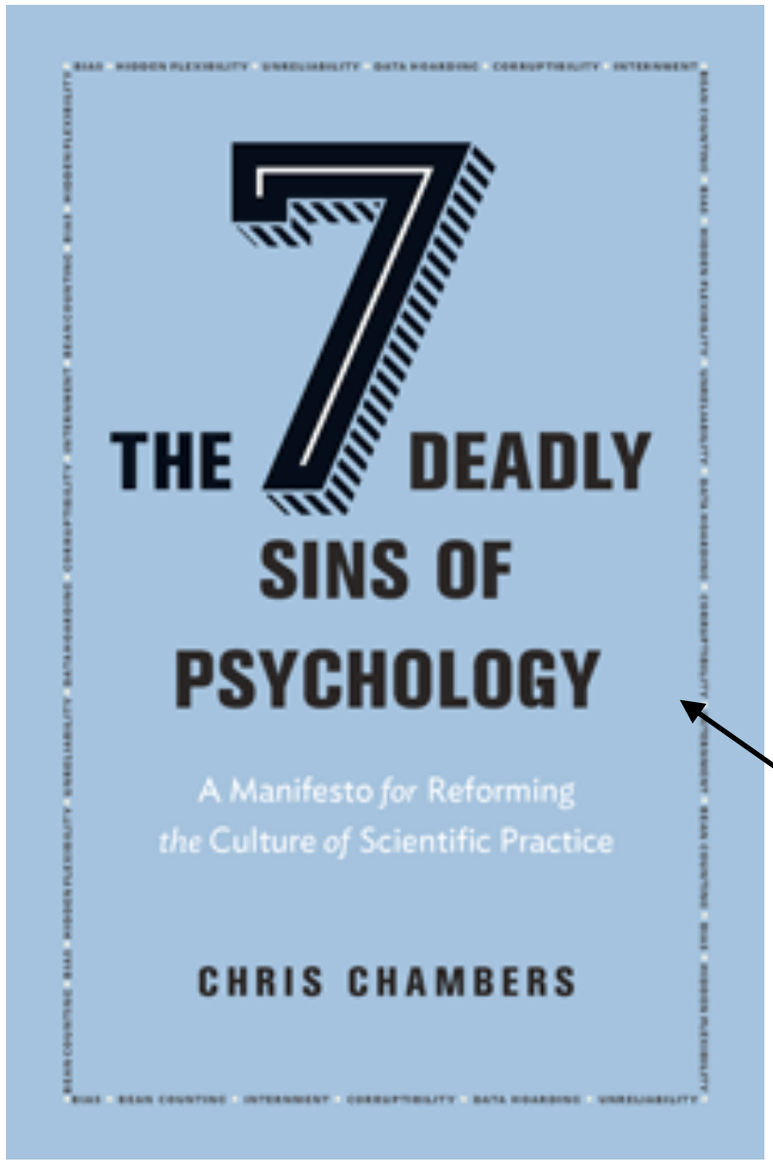
Replicable Science is when someone else can run a study the same as or conceptually equivalent to your one, and find a similar pattern of effects.

Reproducible Science is when someone else can take your data and your analysis code, run it and then find the same effects that you have reported.

How do we make our science more replicable?

How do we make our science more reproducible?

A move towards open science…



Sins include *p*-hacking, lack of power, HARKing, failing (refusal) to share data and code, too many researcher degrees of freedom…

You really should read this book!

http://www.stat.columbia.edu/~gelman/

Andrew Gelman gives the following recommendations to researchers:

- Analyze all your data.
- Present all your comparisons.
- Make your data public.
- Put in the effort to take accurate measurements (low bias, low variance, and a large enough sample size).
- Do repeated-measures comparisons where possible.

Open Science practices include…

- Pre-registering experiments.
- Registered reports.
- Using preprint servers (e.g., bioRxiv, PsyArXiv).
- Making data and analysis code freely available (e.g., via GitHub, OSF).
- Open access to journal articles.
- …and more.

Open Science recently recognised by G7 Science Ministers…

Focus: Incentives and the researcher ecosystem

Ambition: Foster a research environment in which career advancement takes into account Open Science activities, through incentives and rewards for researchers, and valuing the skills and capabilities in the Open Science workforce.

Recommendations:

At national levels: G7 nations should each engage with research stakeholders to identify and implement enhancements to research evaluation and reward systems that take into consideration the Open Science activities carried out by researchers and research institutions. Topics that could be discussed include:

- Recognizing Open Science practices during evaluation of research funding proposals, and research outcomes;
- Recognizing and rewarding research productivity and impact that reflect open science activities by researchers during career advancement reviews;
- Including credit for service activities such as reviewing, evaluating, and curation and management of research data; and,
- Developing metrics of Open Science practices.

In REF2021 UoA Environment…

29. The revised template will also include a **section on 'open research'**, detailing the submitting unit's open access strategy, including where this goes above and beyond the REF open access policy requirements, and wider activity to encourage the effective sharing and management of research data. The panels will set out further guidance on this in the panel criteria.

is beginning to appear in tenure-track job adverts…

> Our Department embraces the values of open and reproducible science, and candidates are encouraged to address (in their statements and/or cover letter) how they have pursued and/or plan to pursue these goals in their work.

and is forming part of Universities' teaching manifestos.

Teaching with Open Science commitment:
To teach the practices and skills of open research and science in our undergraduate and postgraduate degree programmes

a. Promote open science in our teaching.
b. Design a Research Methods curriculum that teaches skills for open science and uses open science to enhance teaching (for example: teach R and use open data to practice analysis skills).
c. Learn about and adopt open educational practices in our teaching.
d. Produce and promote tools for helping student researchers adopt open practices, including training and guidance suitable to their level of study.
e. Author, share and use open educational resources to promote teaching with open science beyond our School and Institution.
f. Support our colleagues to learn the skills of teaching Open Science.

Some Academic Twitter Accounts on Open Science to Follow

- Brian Nosek (Virginia) - @BrianNosek
- Dorothy Bishop (Oxford) - @deevybee
- Marcus Munafò (Bristol) - @MarcusMunafo
- Chris Chambers (Cardiff) - @chrisdc77
- The UK Reproducibility Network - @UKRepro
- Center for Open Science - @OSFramework

Part of doing better science involves knowing how to build appropriate statistical models, and how to understand what those models are telling you (and what they are not…)

ASA Principles on *p*-values

1. *p*-values can indicate how incompatible the data are with a specified statistical model.

2. *p*-values do not measure the probability that the studied hypothesis is true, or the probability that the data were produced by random chance alone.

3. Scientific conclusions and business or policy decisions should not be based only on whether a p-value passes a specific threshold.

4. Proper inference requires full reporting and transparency.

5. A p-value, or statistical significance, does not measure the size of an effect or the importance of a result.

6. By itself, a p-value does not provide a good measure of evidence regarding a model or hypothesis.

Ronald L. Wasserstein & Nicole A. Lazar (2016) The ASA's Statement on p-Values: Context, Process, and Purpose, The American Statistician, 70:2, 129-133, DOI: 10.1080/00031305.2016.1154108

Why R?

- R allows you to engage in reproducible research.
- Statistical packages in R reflect the latest advances in the fields of Statistics and Data Science.
- Advanced models such as Linear Mixed Models (LMMs) are easy to build in R - many of the best journals now require LMMs to be used as they are more powerful and flexible than classical techniques such as ANOVA.
- Lots of amazing data visualisation packages available.
- In many institutions, the next generation of academics (M-level and PhD students, post-docs etc.) are learning R skills as part of their training.
- Plays an important role in Open Science.

What role can R play in Open Science?

- R scripts are easy to share allowing for reproducibility and easy public sharing of data and code.
- R is free, open source software that is much more flexible and powerful than SPSS.
- There is an active R community continuously updating statistical tests and packages that run in R.
- As R is a programming language, it forces you to know your data.

R *vs.* SPSS

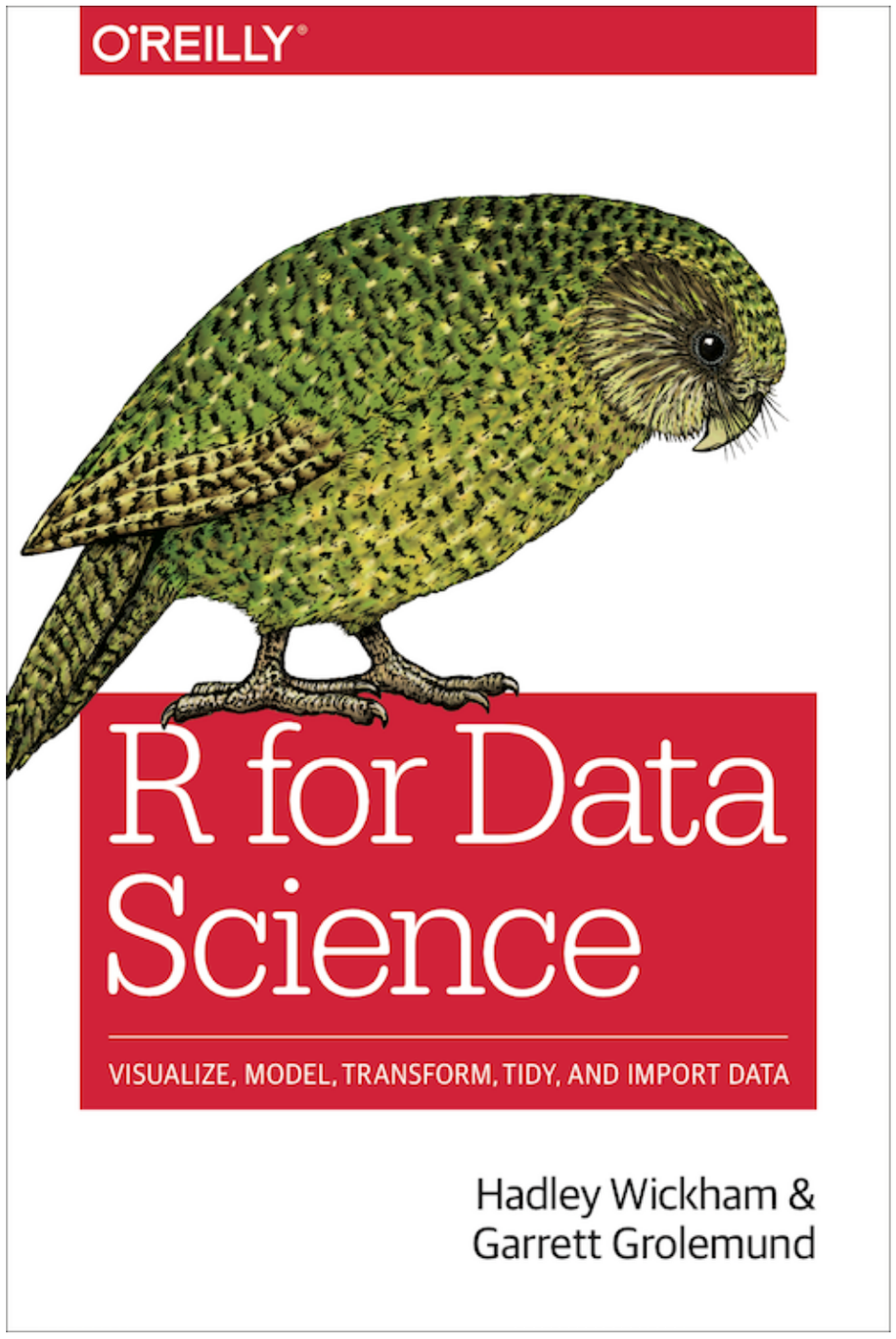
“SPSS is like a bus - easy to use for the standard things, but very frustrating if you want to do something that is not already pre-programmed.

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).

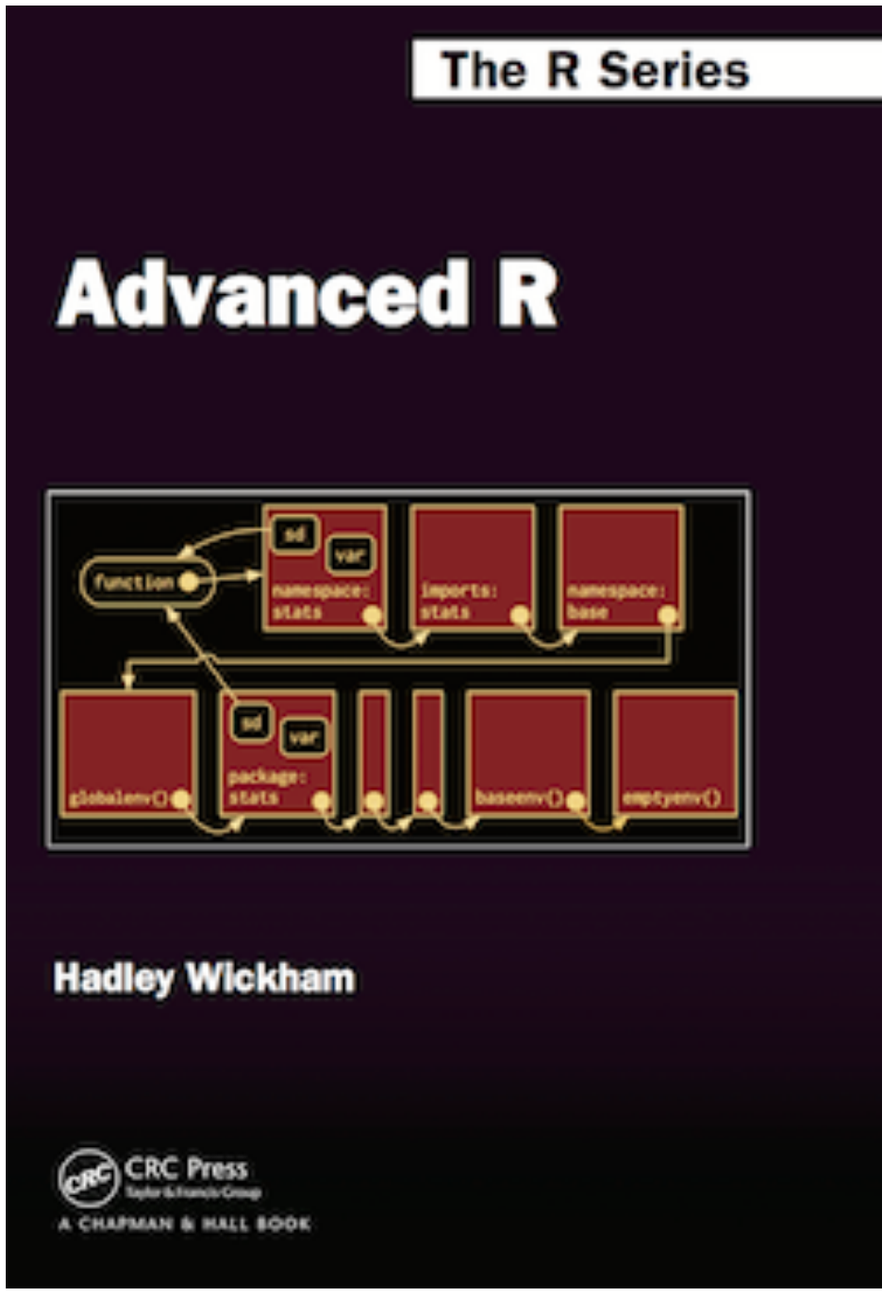
In meme form…


If statistics programs/languages were cars...
Excel
SPSS
SAS
STATA
R



Available electronically for free at:

http://r4ds.had.co.nz



Available electronically for free at:

https://adv-r.hadley.nz

“Hadley Wickham, the Man Who Revolutionized R”

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

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

R User Groups…

R user groups exist all over the world with many in the UK (incl. Manchester)…

UK

England

- Birmingham: BRUM
- Canterbury: CanterburyR
- Cambridge: CambR
- Coventry: Warwick R User Group; @WarwickRUG
- Exeter: Exeter R Users Group
- London: LondonR
- Manchester: ManchesterR
- Newcastle: R North East; @RstatsNE
- Nottingham: Nottingham R User group
- Oxford: R user group Oxford; @rusersoxford
- Sheffield: SheffieldR;@Sheffield_R_

You can even buy the t-shirt…

RStudio

$17

magrittr

$17

Shiny

$17

I like the way you R

$17

…and conferences

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.

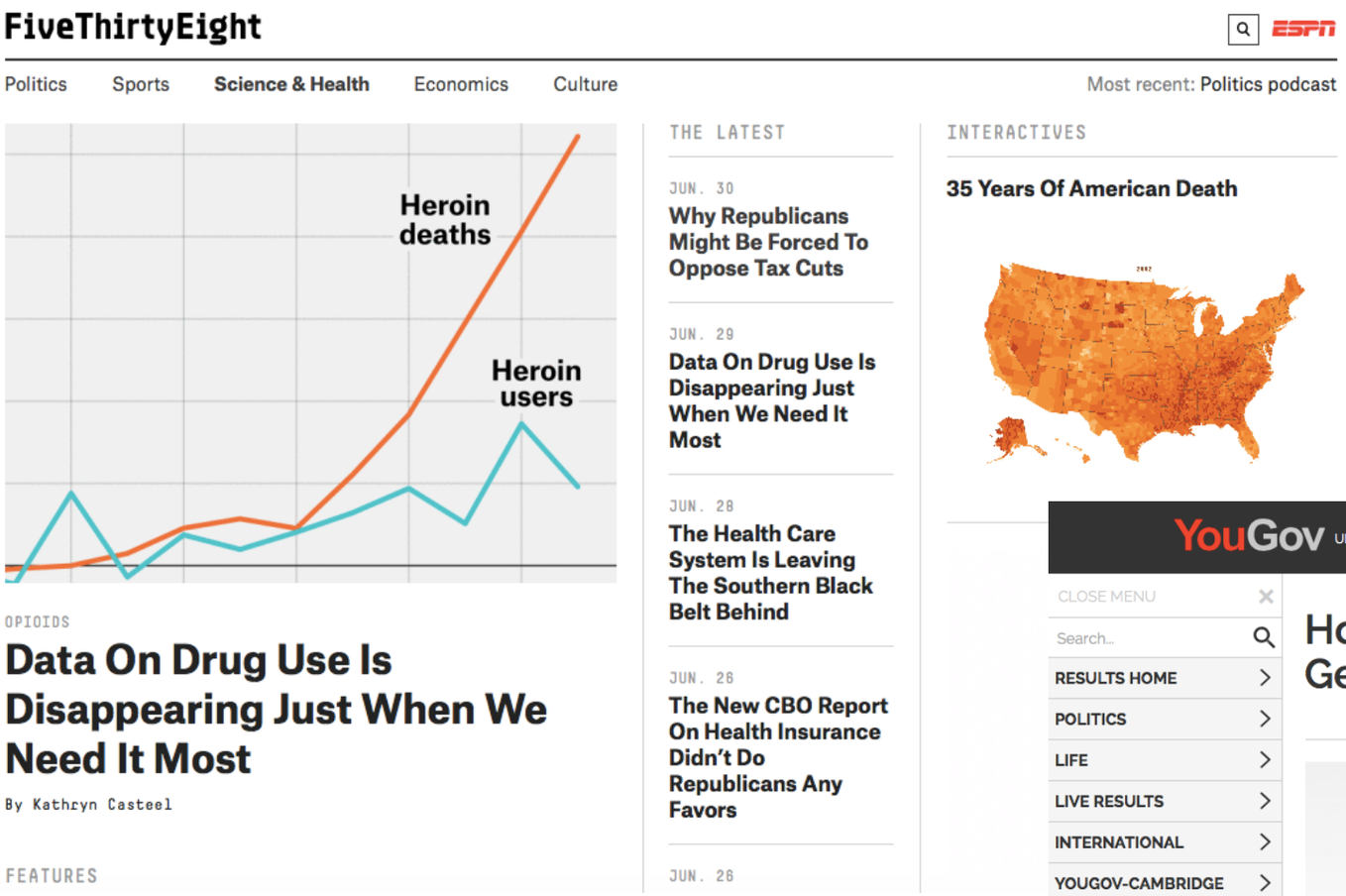
In the 2014 Dice Tech Salary Survey of over 17,000 technology professionals, the highest-paid IT skill was R programming. While big-data skills in general featured strongly in the top tier, having R at the top of the list reflects the strong demand for skills to make sense of, and extract value from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

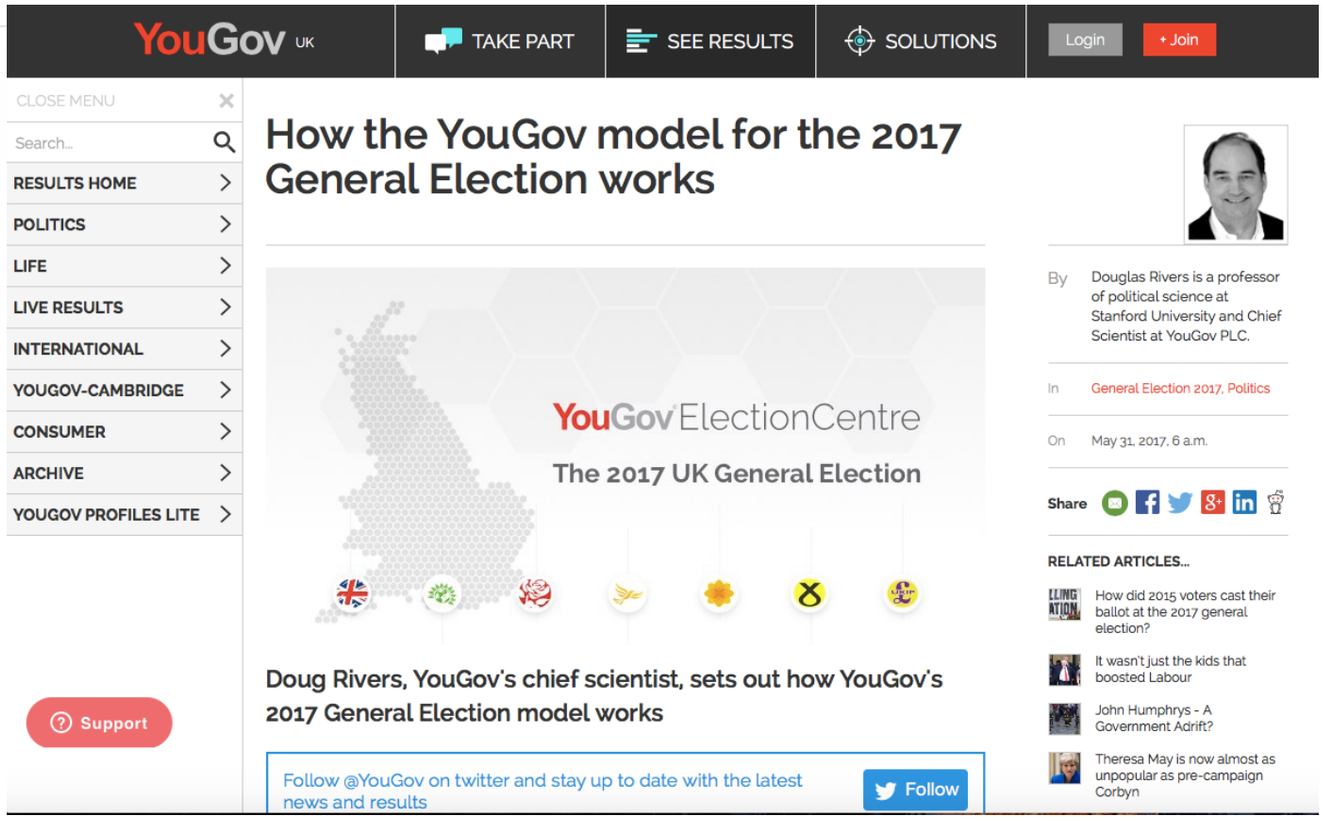
AVERAGE SALARY FOR High Paying Skills and Experience

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%

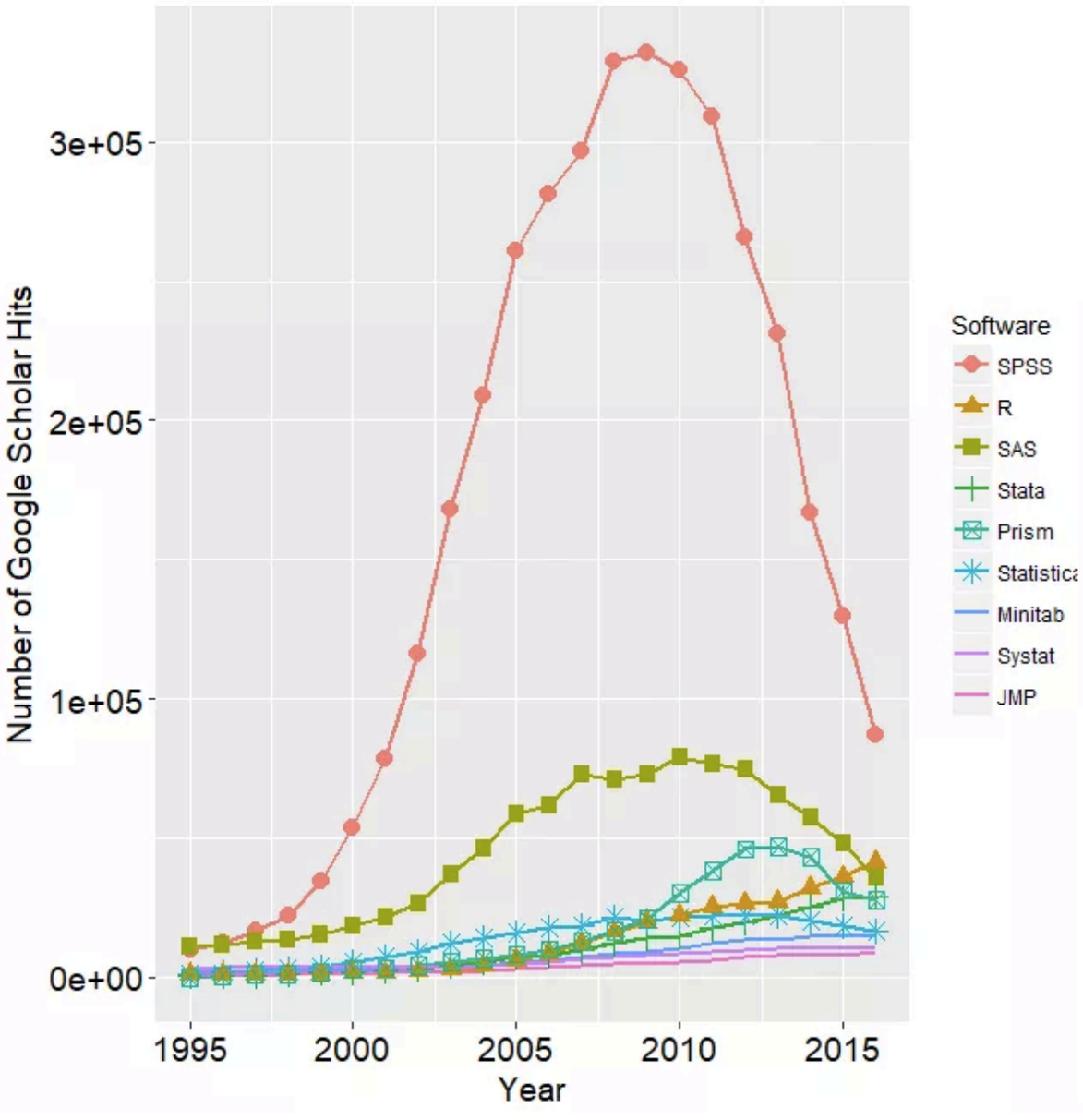
Similarly, the recent O'Reilly Data Scientist Survey also found R skills amongst those that pay in the $110,000-$125,000 range (albeit amongst a much smaller and specialized sample of respondents).

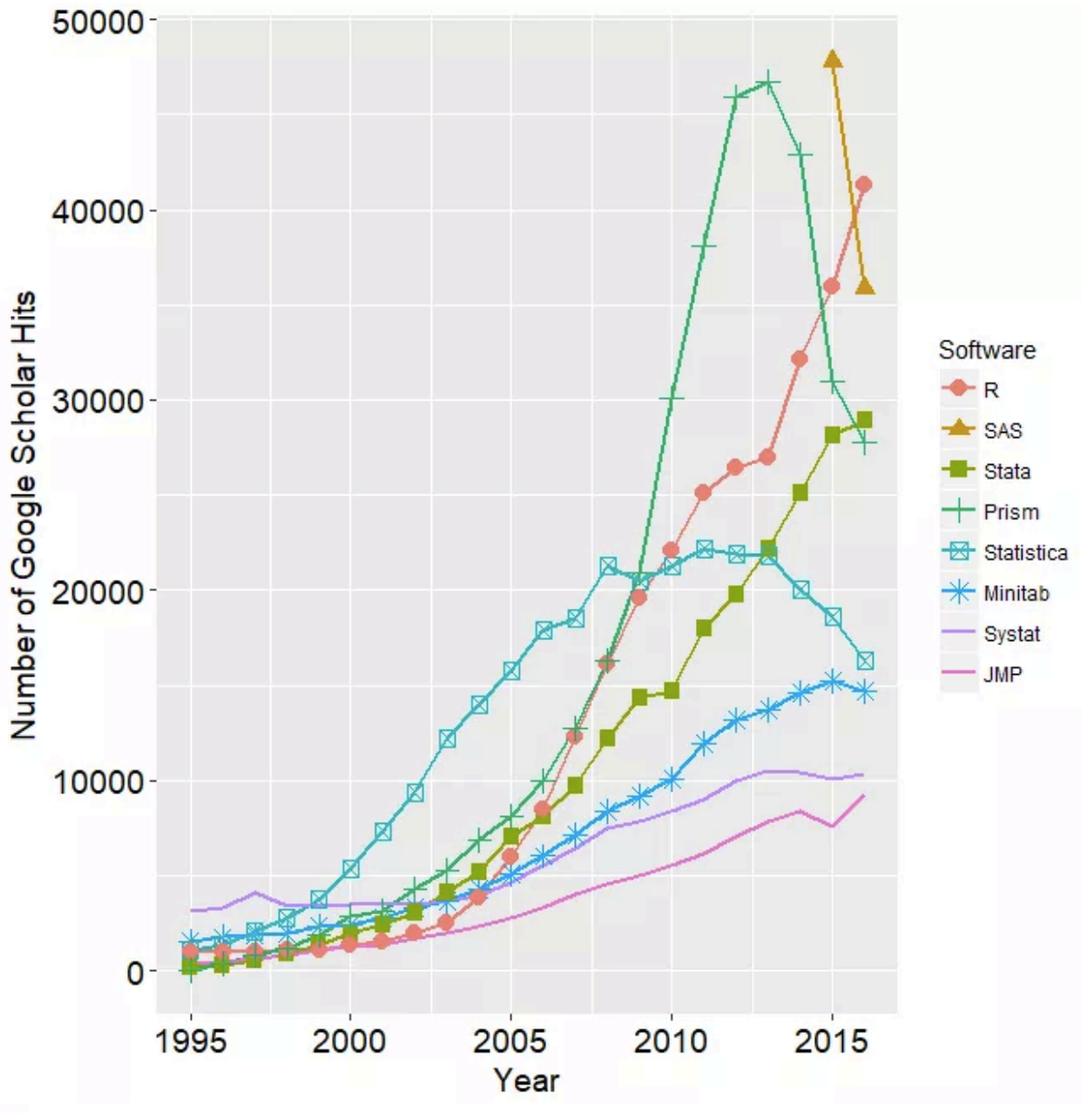
and R is widely used by a number of organisations (incl. the ONS)…





and across academia…





with Data Science a growing employment destination for Psychologists.

CAREER CENTER

Hot jobs: Big-data psychologists

Wanted: Scientists who can help companies make sense of consumer and employee data.

By Rebecca Voelker

Print version: page 18

Every time you use an Internet search engine, sign up for a company "rewards" program or swipe your credit card, that information is saved and stored. The industries that amass these billions of bytes of data are increasingly hiring psychologists to help make sense of it, says Douglas Reynolds, PhD, president of APA's Div. 14 (Society for Industrial and Organizational Psychology).

"Big data, and what it means for business, is a hot topic right now," says Reynolds, who also serves as vice president of assessment technology at Development Dimensions International Inc., a human resources consulting firm. "We're in high demand."

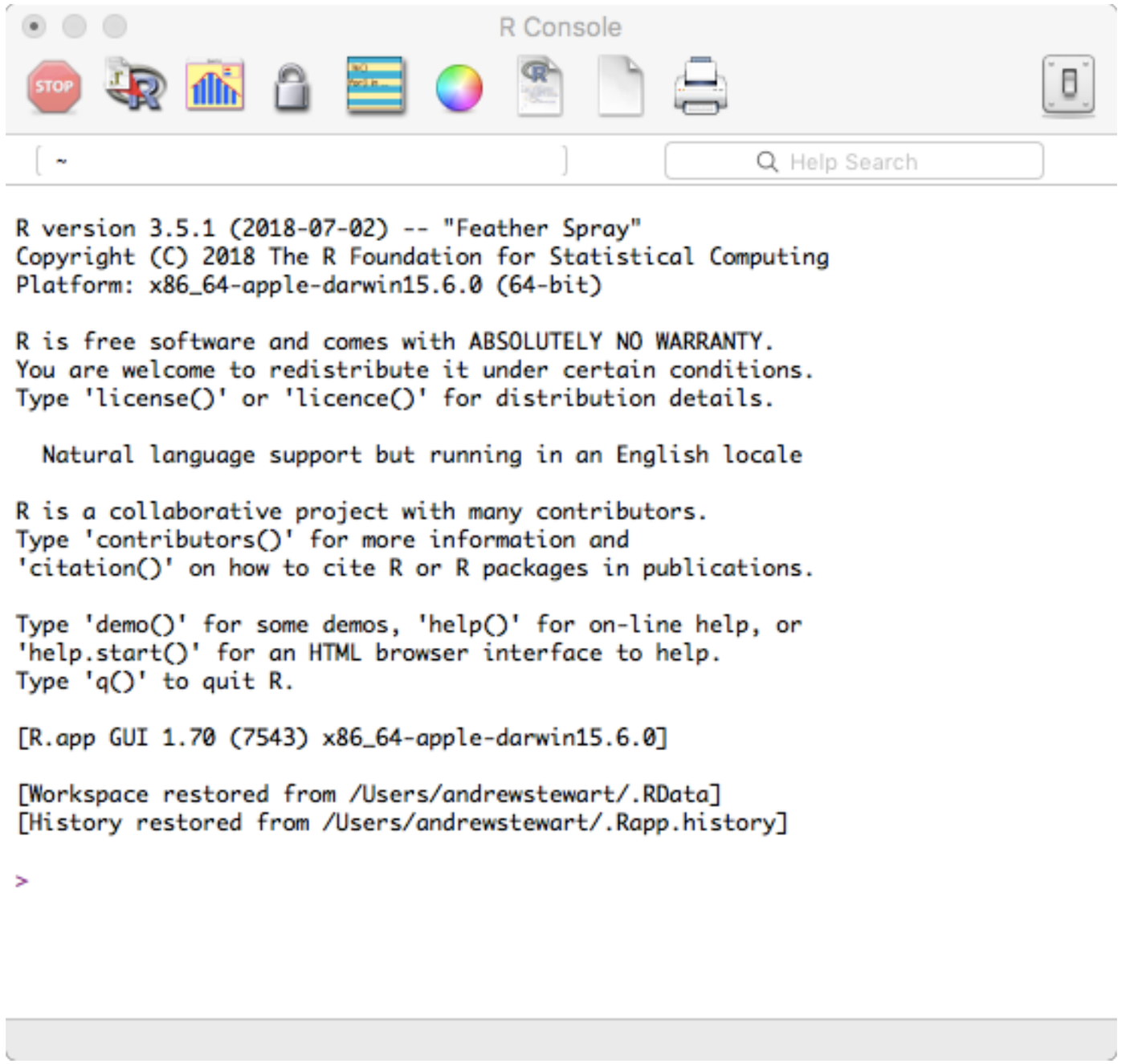
Psychologists' ability to interpret numbers and human behavior makes them key members of many industry analytics teams, adds Suzie Weaver, PhD, a psychologist and senior analytic consultant with Epsilon, a global marketing and analytics company. "Analytics is at the core of everything we do, whether it's in research, the academic sphere or the business world."

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

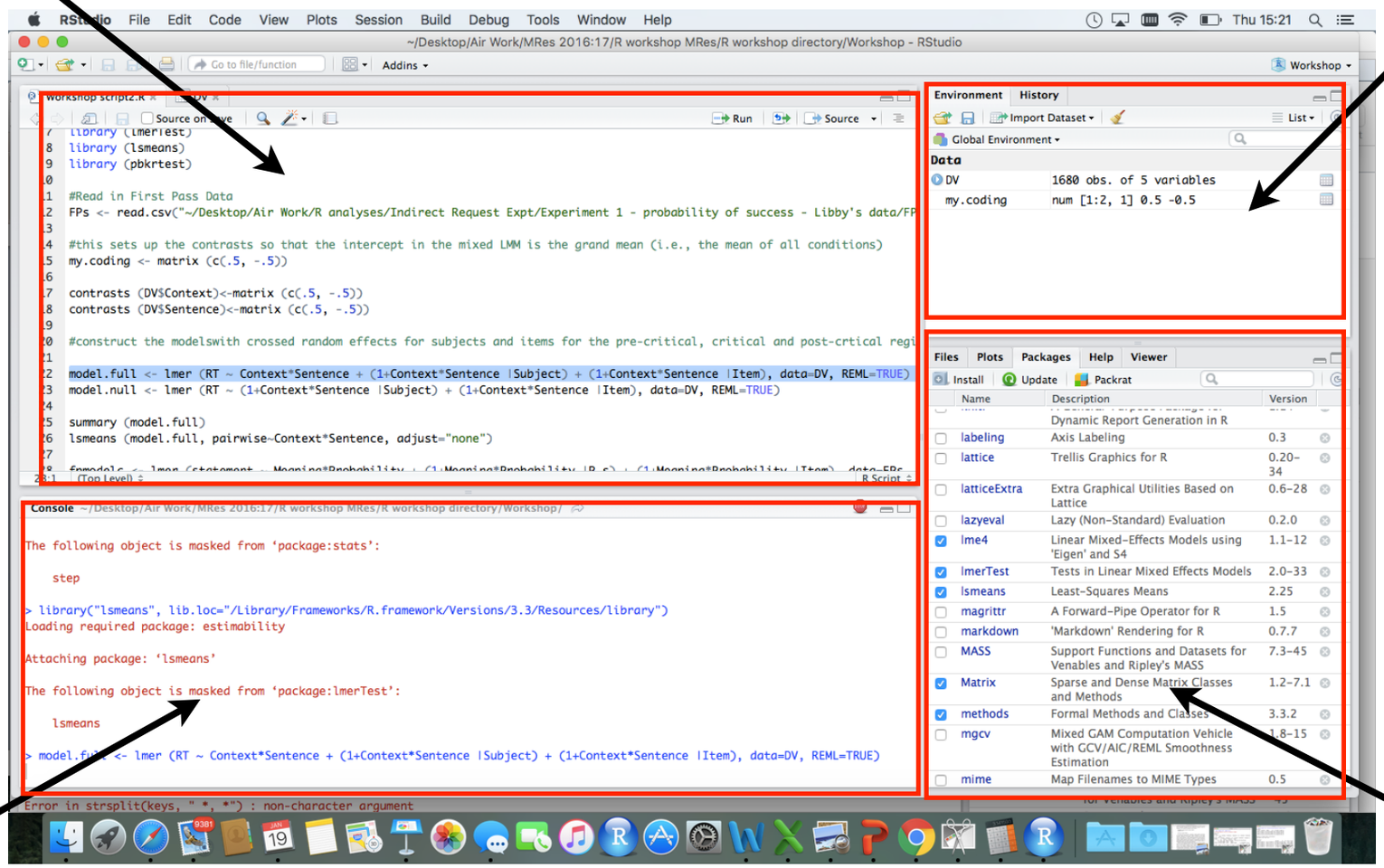
Files | Plots | Packages | Help | Viewer

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

This is where you build your script and where data can be seen.

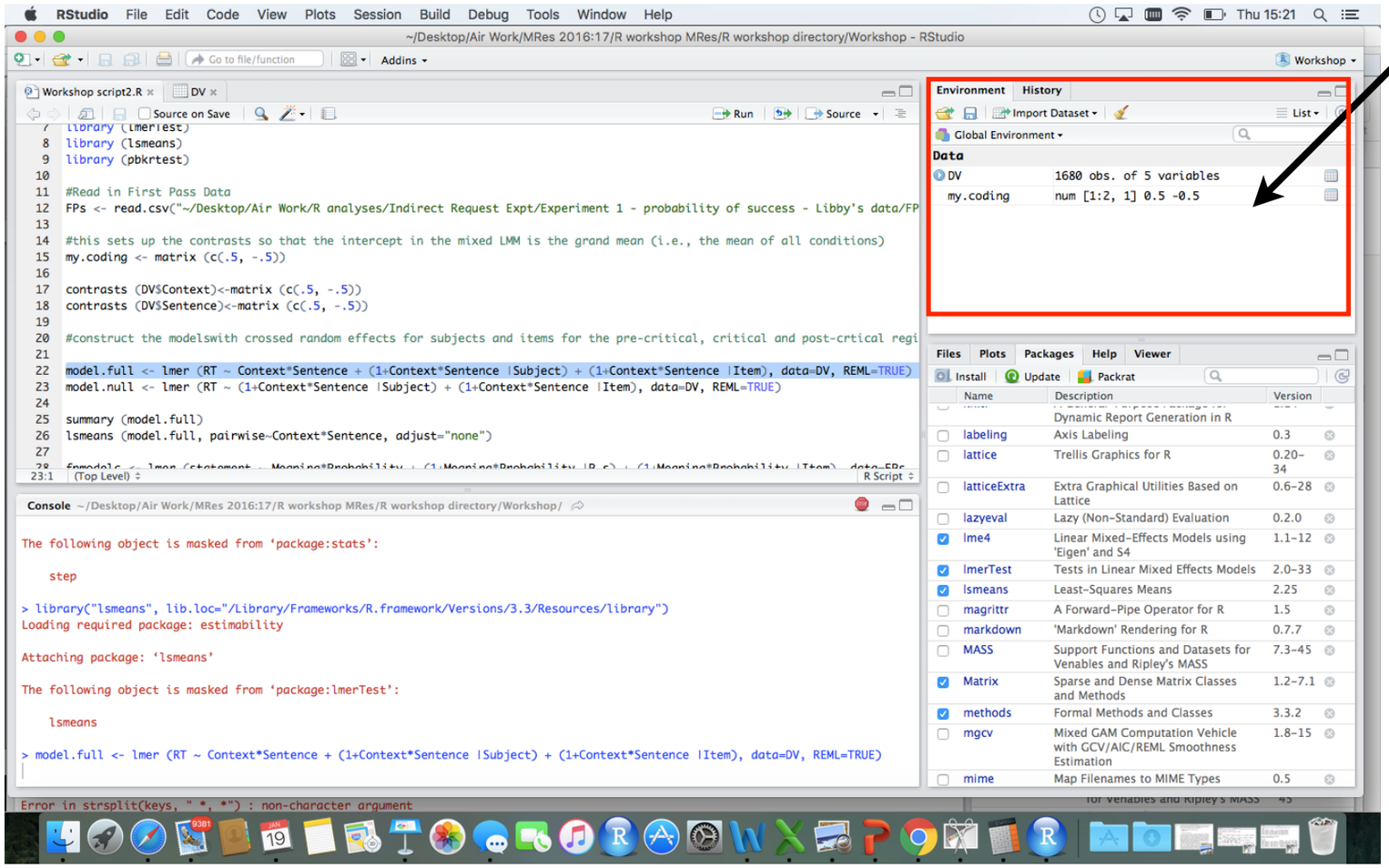
This contains information about your variables and open data sets.



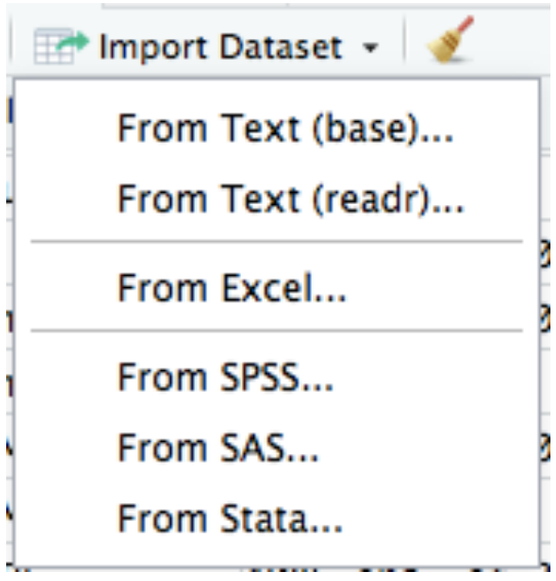
This is where you type commands.

This lists the packages you have loaded, and has tabs for help, graphs etc.

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)
```

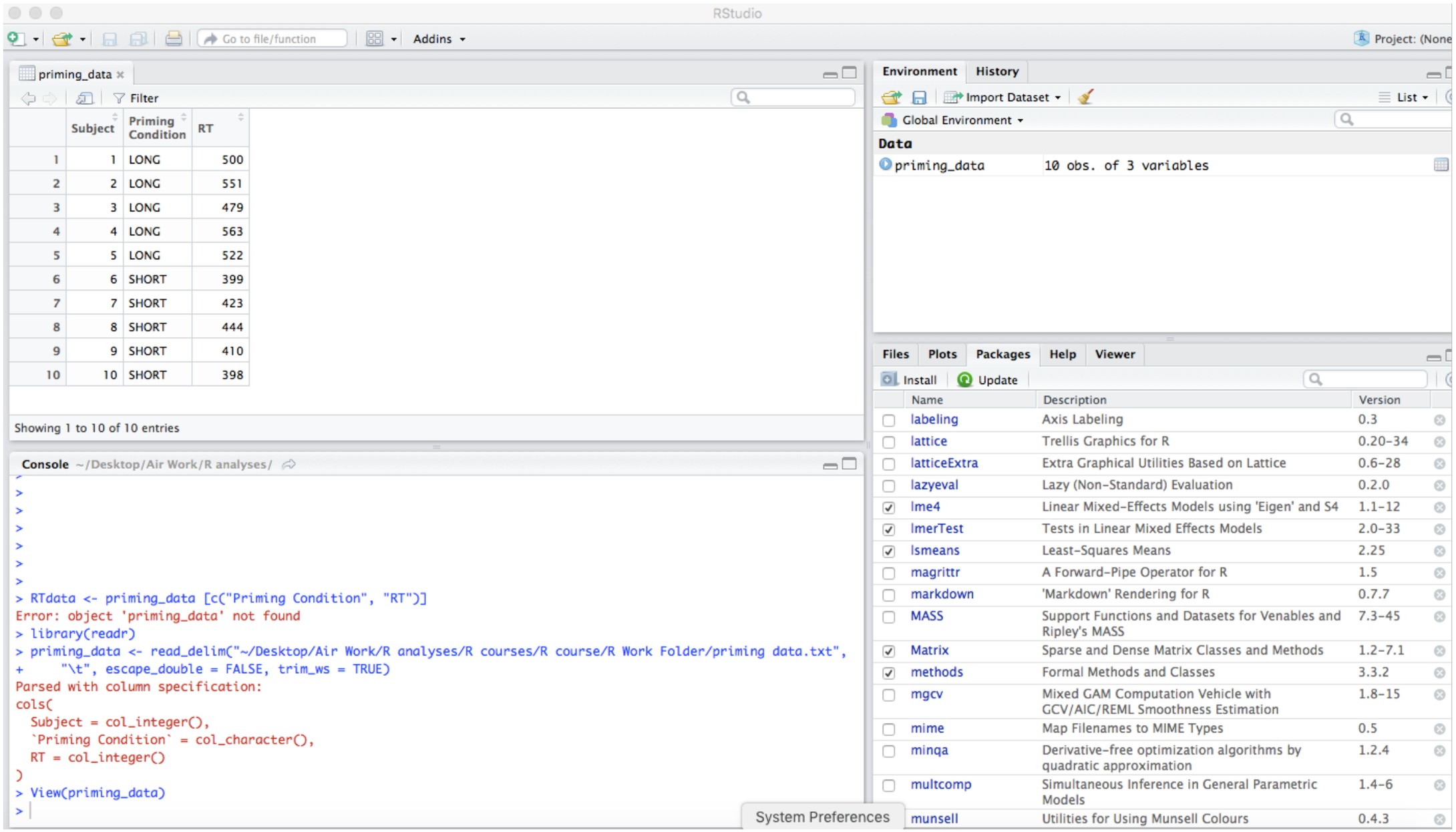
Import

Cancel

Can change here how the columns are delimited.

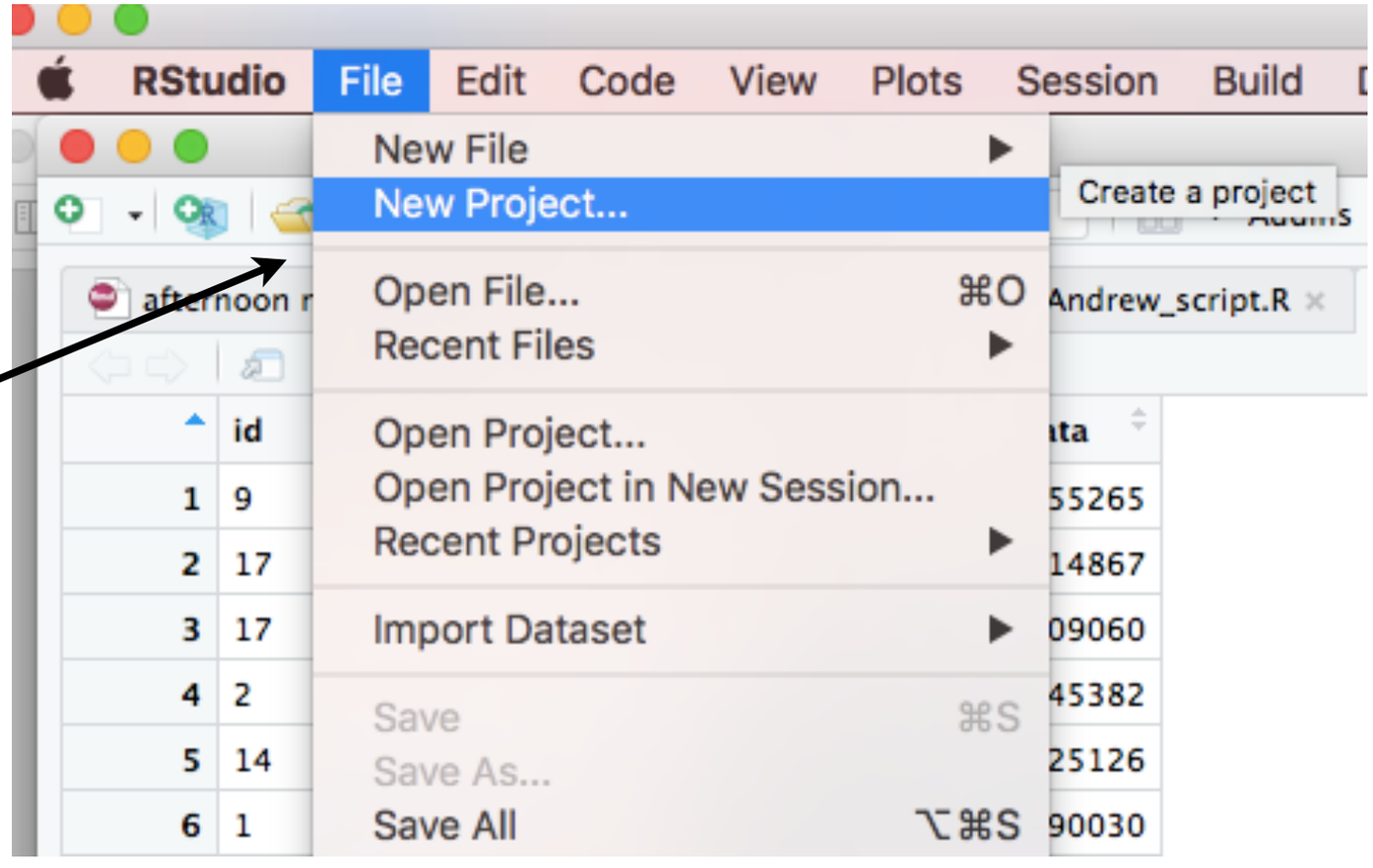
This is the code that corresponds to what you're getting RStudio to do.

- RStudio now looks like this:



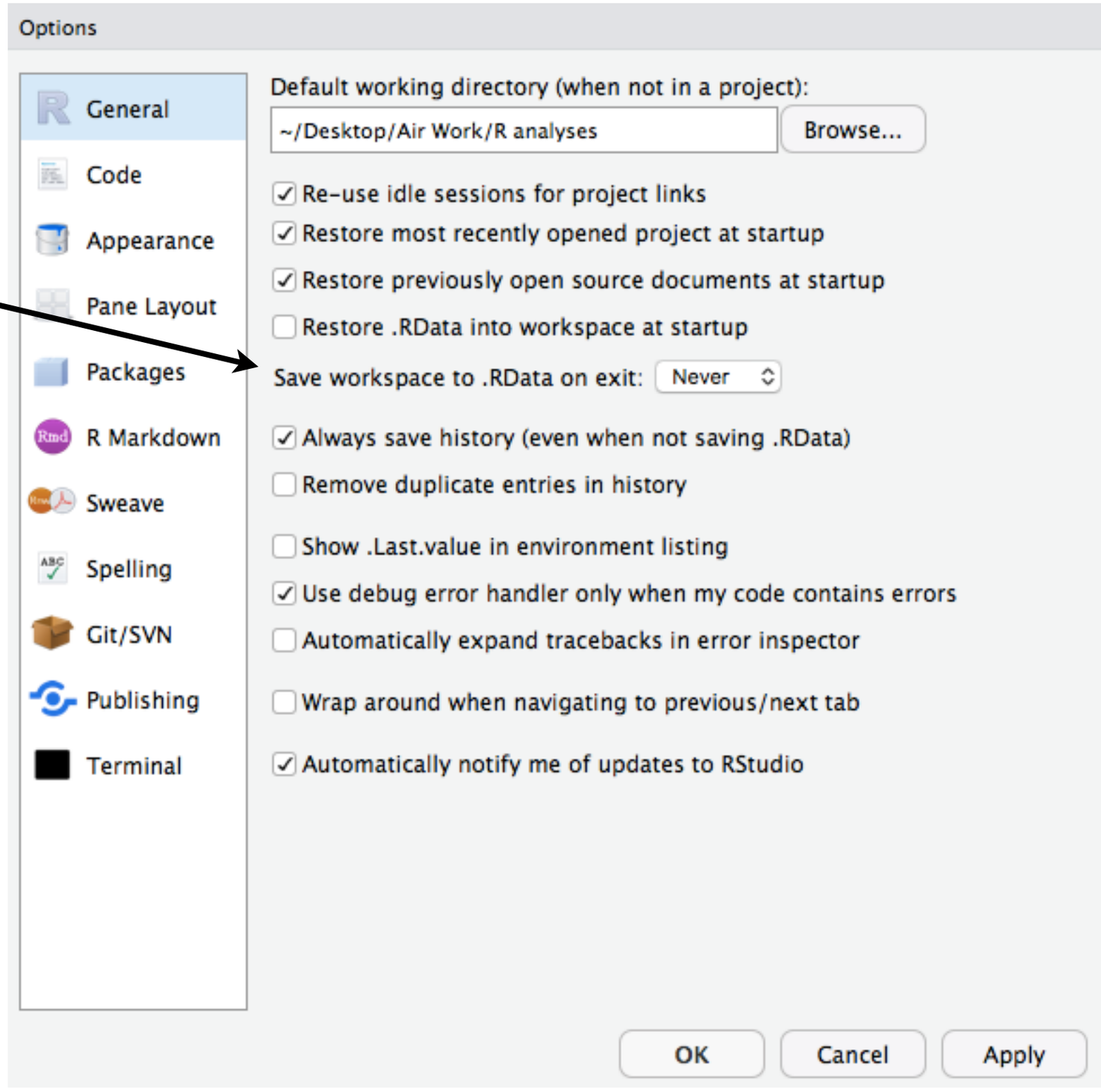
When Starting R

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

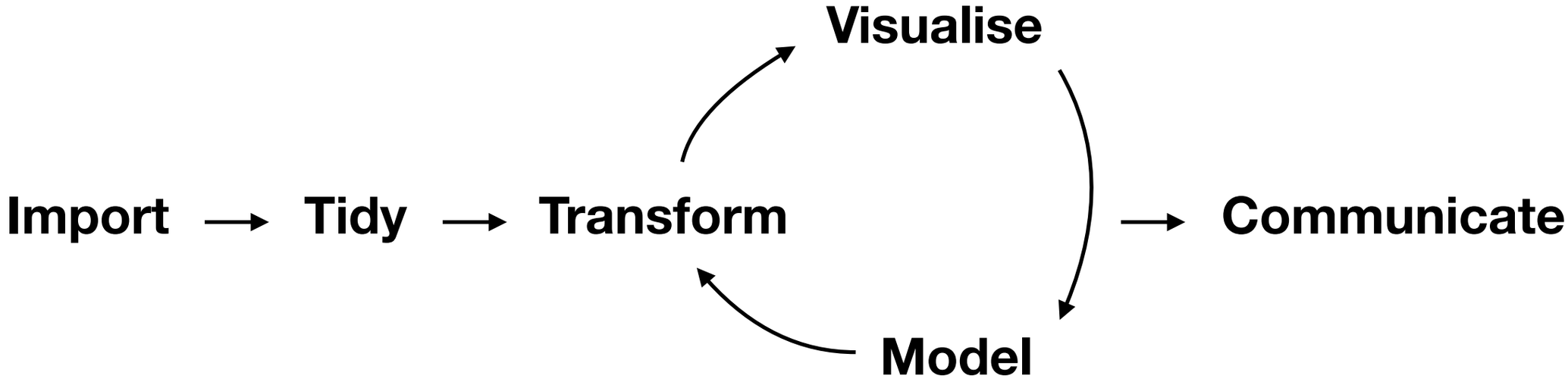


When Starting R

Under preferences, make sure you <u>uncheck</u> the Restore .RData option and select “Never” under the save workspace option…



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



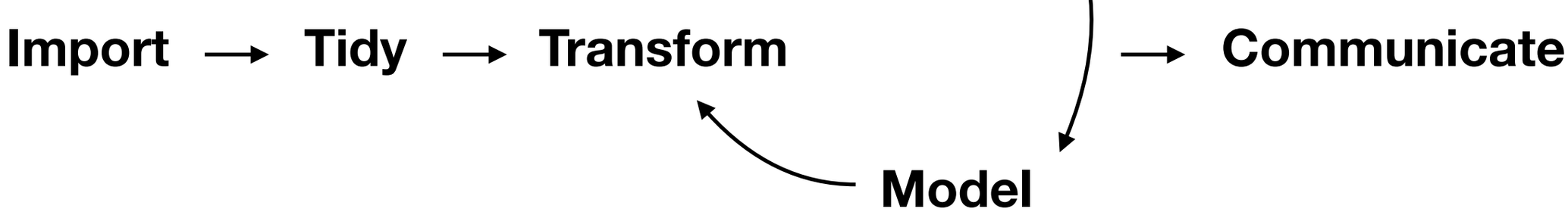
Workflow

Data plots using `ggplot()`

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

Visualise

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()`

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)
```

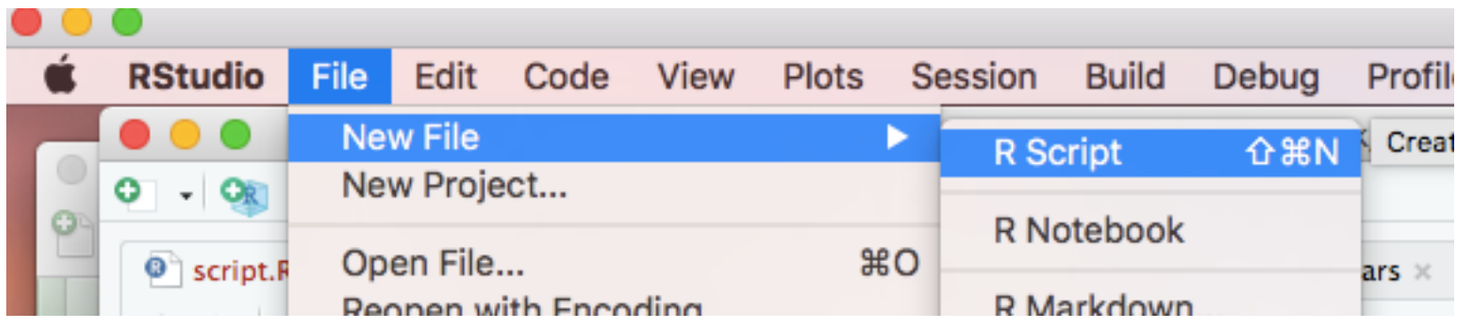
Every time you start R afresh, you need to load the packages you will use by using the `library` function. If you see notation like the following, it means we're using a function (e.g., `summarise`) from within a package (e.g., `dplyr`)

```
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.
- ~12,000 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.

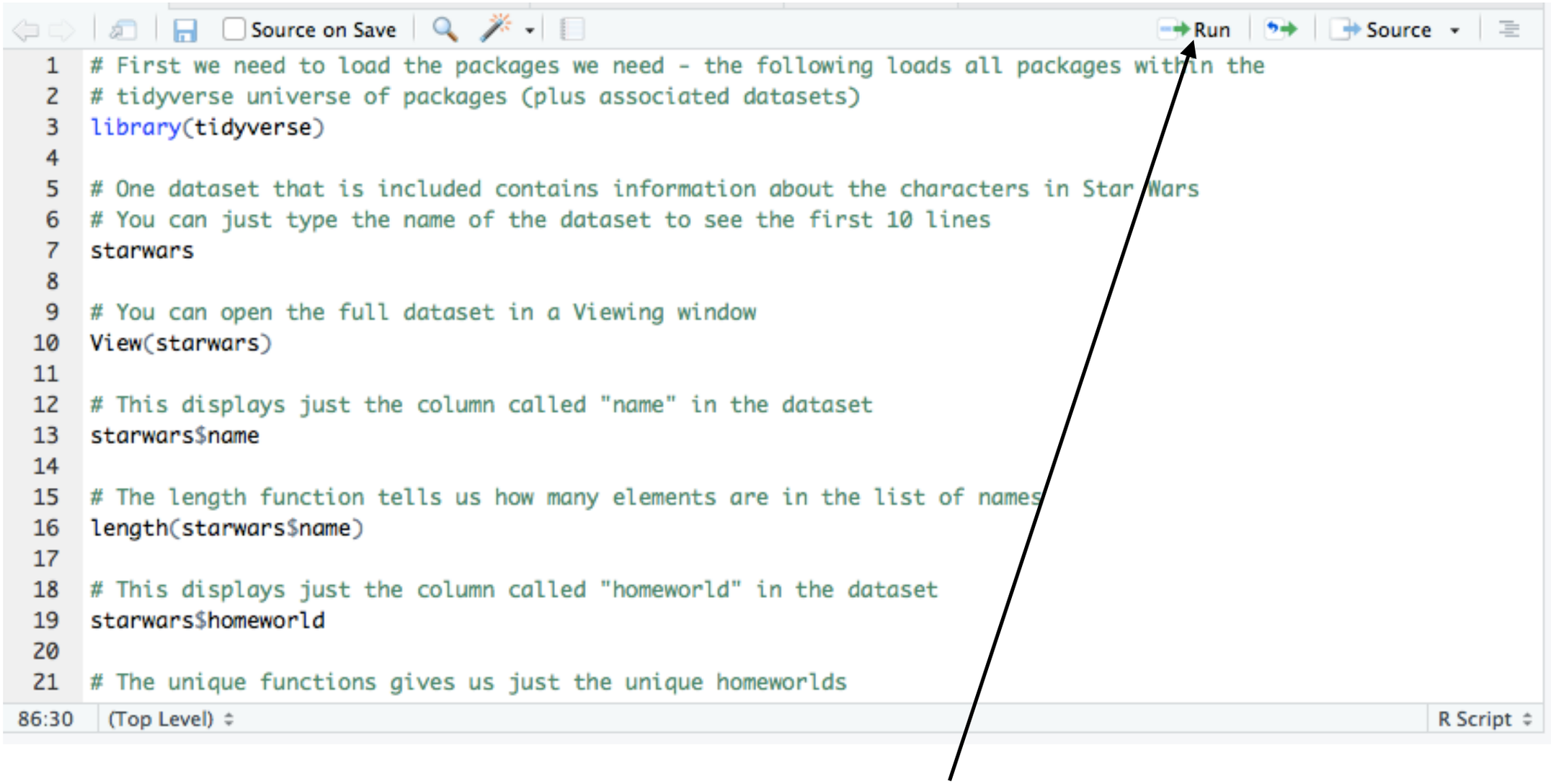
If you aren't sharing your data and code, you aren't engaged in generating reproducible or open research…

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.

Like this…

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 `+ , - , = , <-`

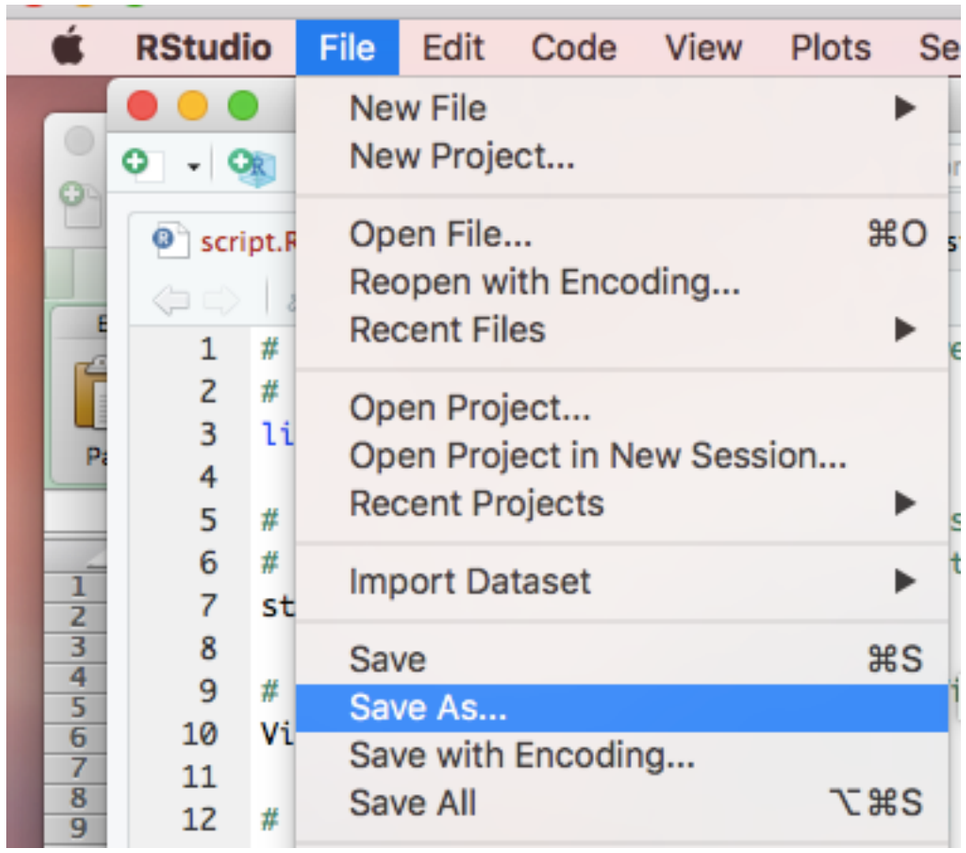
Whitespace used sensibly makes your code easier to read. Separate discrete sections of code in your script with a blank line.

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

Comment, comment, comment…

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

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



Sharing your analysis

Journals are increasingly requesting analysis code and data to be published alongside the published paper (and sometimes at the review stage).

You can share your analysis and data even at the submission stage using something like GitHub or OSF.

Git is a powerful tool that allows for the version control in collaborative projects, and the sharing of code and projects.

You can set up a GitHub account, and install GitHub Desktop on your own computer.

GitHub, Inc. github.com/ajstewartlang

Live Trains | Google Scholar | Scopus | jobs.ac.uk | Apple | BBC News | Chester Weather | The Telegraph | The Grauniad | The Independent | Google Maps | Chester Weather Station

Search GitHub | Pull requests | Issues | Gist

Overview | Repositories 4 | Stars 0 | Followers 0 | Following 0

Andrew Stewart

ajstewartlang

Senior Lecturer in the Division of Neuroscience and Experimental Psychology, University of Manchester.

University of Manchester

http://personalpages.manche...

Joined 21 days ago

Pinned repositories

Customize your pinned repositories

List-of-which-repository-goes-with-which-paper

Affective-Theory-of-Mind-Inferences

R code and data for the two experiments

R

Comprehension-of-indirect-requests-is-influenced-by-their-degree-of-imposition

R code and data

R

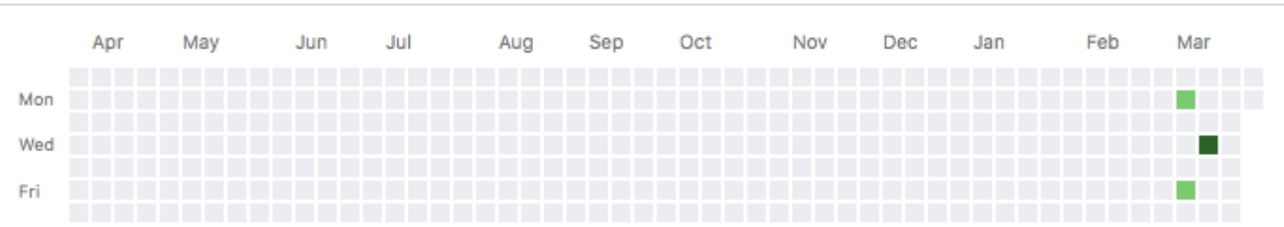
It-s-hard-to-write-a-good-article

R code and data

R

5 contributions in the last year

Contribution settings



Learn how we count contributions.

Less More

This repository Search Pull requests Issues Gist

ajstewartlang / **Affective-Theory-of-Mind-Inferences**

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

R code and data for the two experiments

Edit

Add topics

5 commits 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

ajstewartlang Capitalisation corrected Latest commit 7e14ea2 3 hours ago

Both Expts comparison	Tidied up code with consistent labelling	3 hours ago
Expt 1 Script and data	Tidied up code with consistent labelling	3 hours ago
Expt 2 Script and data	Tidied up code with consistent labelling	3 hours ago
README.txt	Capitalisation corrected	3 hours ago

README.txt

The repository "Both Expts comparison" contains the R analysis script and RT data for both experiments in 1 file. The conditions are relabeled to congruent vs. incongruent to allow a comparison of the magnitude of the congruency effect across the two experiments.

The repository "Expt 1 Script and Data" contains the R analysis script for the Anger/Fear experiment, the RT data, the accuracy data and the two data files (containing the word "graph") used by ggplot2 to graph the means and SEs.

The repository "Expt 2 Script and Data" contains the R analysis script for the Happy/Sad experiment, the RT data, the accuracy data and the two data files (containing the word "graph") used by ggplot2 to graph the means and SEs.

This repository Search Pull requests Issues Gist

ajstewartlang / Affective-Theory-of-Mind-Inferences

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Branch: master

Create new file Upload files Find file History

Affective-Theory-of-Mind-Inferences / Expt 1 Script and data /

ajstewartlang Tidied up code with consistent labelling — Latest commit de7b584 3 hours ago

..		
AngerFearAcc.csv	Tidied up code with consistent labelling	3 hours ago
AngerFearRT.csv	Tidied up code with consistent labelling	3 hours ago
AngerFeargraphacc.csv	R code and data	17 days ago
AngerFeargraphdata.csv	R code and data	17 days ago
Expt1_AngerFear_script.R	Tidied up code with consistent labelling	3 hours ago

ajstewartlang / **Affective-Theory-of-Mind-Inferences**

Watch 0 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Branch: master Find file Copy path

Affective-Theory-of-Mind-Inferences / Expt 1 Script and data / **Expt1_AngerFear_script.R**

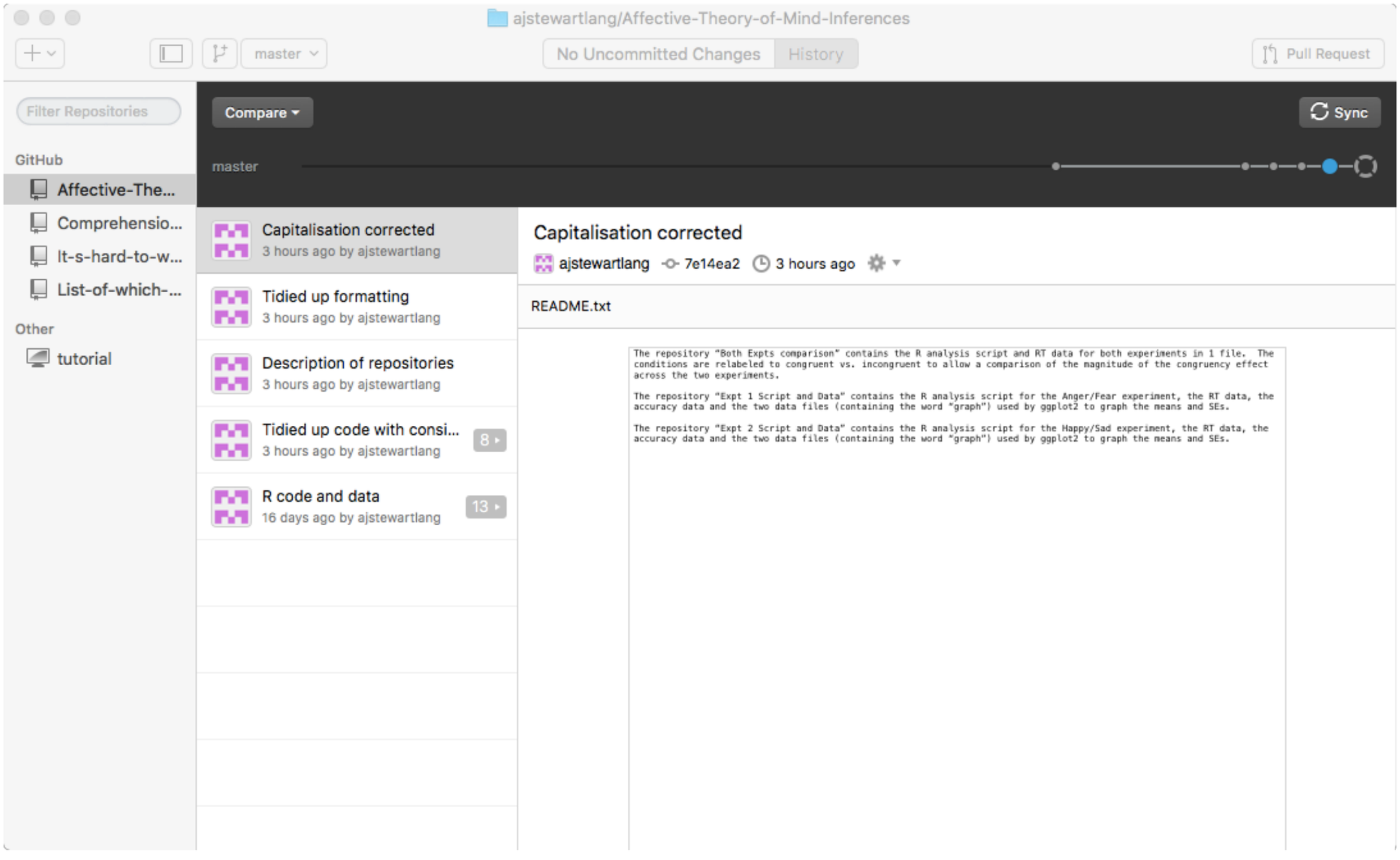
ajstewartlang Tidied up code with consistent labelling de7b584 3 hours ago

0 contributors

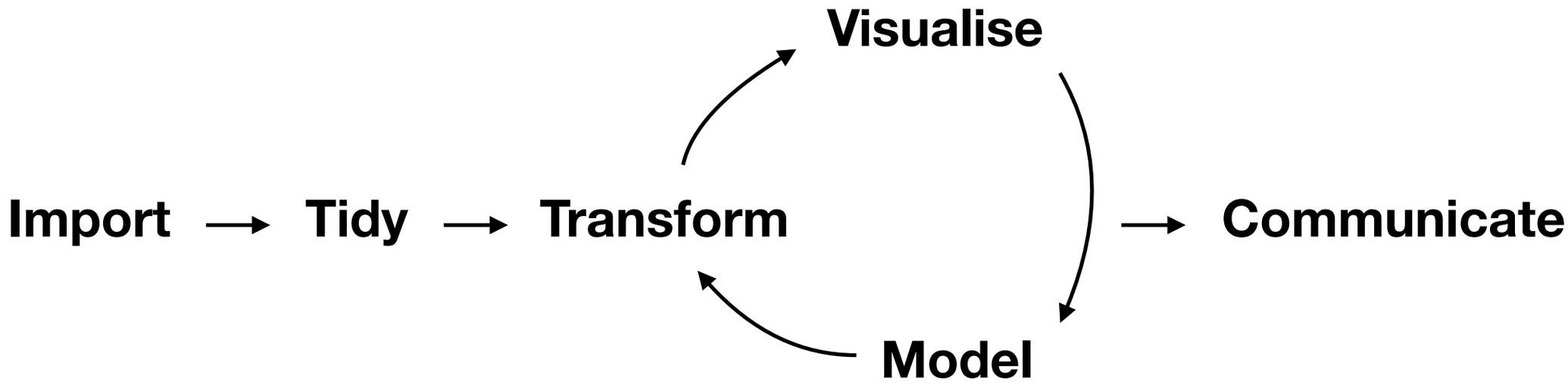
59 lines (46 sloc) | 3.17 KB Raw Blame History

```
library (lme4)
library (lmerTest)
library (lsmeans)
library (pbkrtest)
library (readr)
library (ggplot2)

#script for AngerFear RT and accuracy data analysis with arousal

#this is the analysis of the RT data
AngerFearRT <- read_csv("~/AngerFearRT.csv")

AngerFearRT$StoryEmotion <- as.factor (AngerFearRT$StoryEmotion)
AngerFearRT$FaceExpression <- as.factor (AngerFearRT$FaceExpression)

contrasts (AngerFearRT$StoryEmotion) <- matrix (c(.5, -.5))
contrasts (AngerFearRT$FaceExpression) <- matrix (c(.5, -.5))

#with Subject, Vignette, and Face as crossed random effects with arousal
#full model does not converge so need to drop interaction term from the random effects - in addition, Face random effect has only random in
modelRTAr1 <- lmer (RT ~ StoryEmotion*FaceExpression*Arousal + (1+StoryEmotion+FaceExpression|Subject) + (1+StoryEmotion+FaceExpression|Vig
summary (modelRTAr1)
modelRT <- lmer (RT ~ StoryEmotion*FaceExpression + (1+StoryEmotion+FaceExpression|Subject) + (1+StoryEmotion+FaceExpression|Vignette) + (1
anova (modelRTAr1, modelRT)

#difference between models not signif - arousal does not interact with effect so drop arousal from subsequent analysis

modelRTnull <- lmer (RT ~  (1+StoryEmotion+FaceExpression|Subject) + + (1+StoryEmotion+FaceExpression|Vignette) + (1+FaceExpression|Face),
anova (modelRT, modelRTnull)
summary (modelRT)
```

This is GitHub Desktop



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



https://www.tidyverse.org

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`.

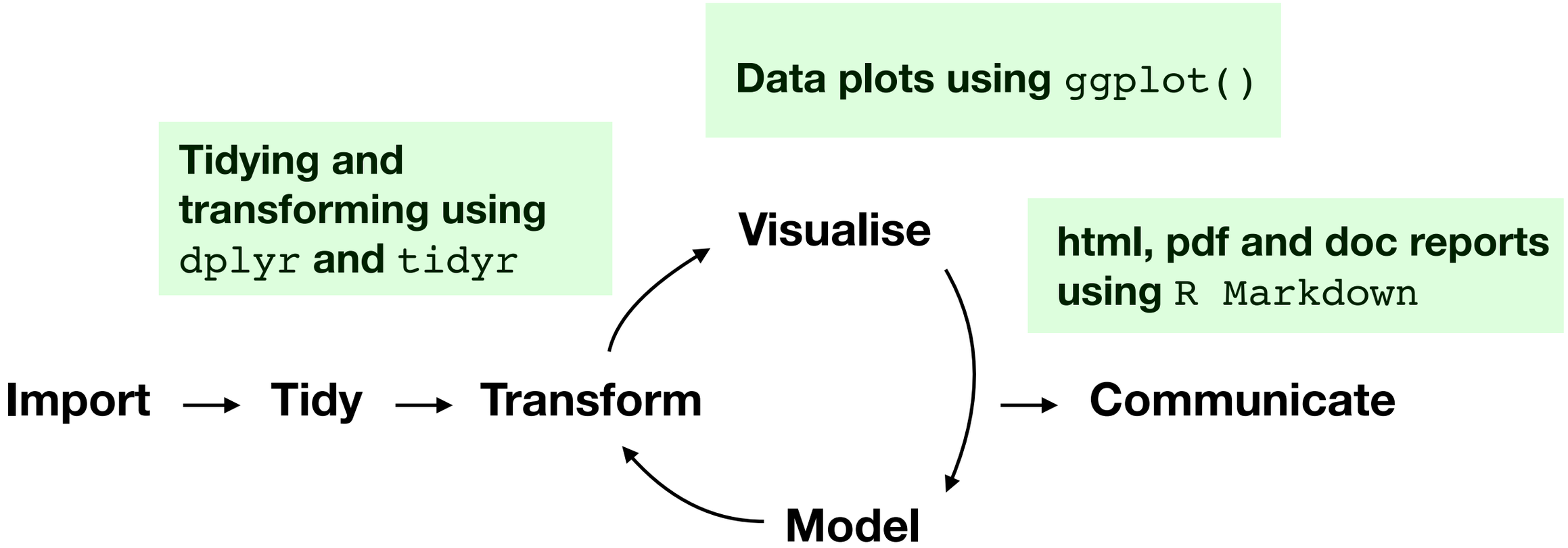
- You can load each package separately with (e.g.)

```
> library(ggplot2)
```

- or load all tidyverse packages with

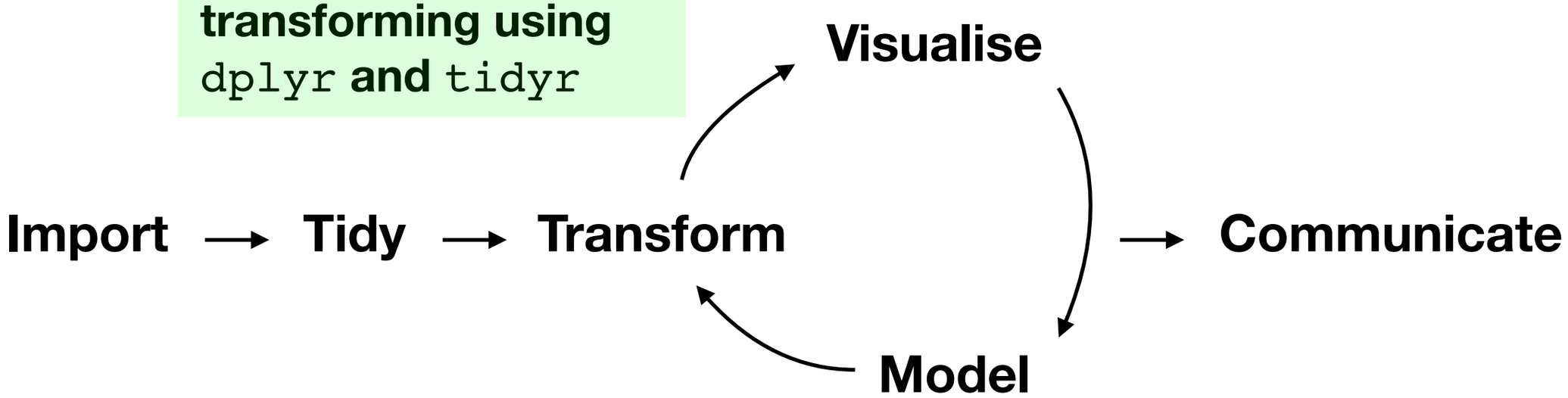
```
> library(tidyverse)
```

Workflow


Data plots using ggplot()
Tidying and transforming using dplyr and tidyr
Visualise
html, pdf and doc reports using R Markdown
Import
Tidy
Transform
Communicate
Model
ANO(C)VA using afex() and aov()
Linear regression using lm() and step()
(Generalised) linear mixed models using lmer() and glmer()

Tidying and Transforming Data

Tidying and transforming using `dplyr` and `tidyr`



Tidyverse Philosophy

- Each Tidyverse package contains a series of functions or **verbs** - the name of the verb gives a clue as to what it does.

- In the `dplyr` package there are verbs such as `filter`, `select`, `mutate`, `gather`, `spread`, `group_by`, `summarise` etc.

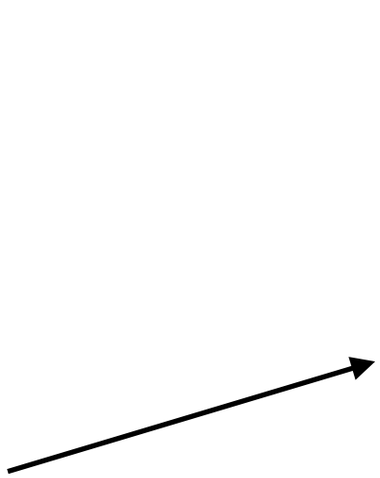
Tidying and Transforming Data

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

If we type into the Console:

```
> View(data)
```

the data frame is displayed like this…



Filter

	ID	WM	IQ	Comp
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
11	11	47	86	17
12	12	45	113	21
13	13	46	114	19
14	14	50	99	20

Showing 1 to 14 of 10,000 entries

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

```
> str(data)
'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)
'data.frame':    48 obs. of  3 variables:
 $ ID              : int  1138 6223 6092 6232 8606 6400 95 2324 6656 5138 ...
 $ 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 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 `data`.

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
      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(data, dataRT, by = (c("ID")))
> str(dataRT_all)
'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…

Filter

	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

Lots of other join functions are available in the `dplyr` package including:

`full_join()` - returns a data frame of two data frames joined by your specific variable of interest but ALSO cases where there isn't a match (represented by NA)

`anti_join()` - returns a data frame of two data fame for cases where there isn't a match by your specific variable of interest present i the second one.

This could save you *hours* relative to doing this all manually in Excel for situations where you're trying to bring together two spreadsheets.

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
(dataRT_all$Simple_sentence), log_Complex = log (dataRT$Complex_sentence))
> data_transformed
     ID WM  IQ Comp Simple_sentence Complex_sentence log_Simple log_Complex
1    95 47  94   19            2154              2441   7.675082    7.758333
2   400 45 118   18            1824              2456   7.508787    7.825245
3   457 42 100   22            1857              2324   7.526718    7.912423
4  1138 41  77   18            1902              2341   7.550661    7.772753
5  1587 54  67   21            1844              2320   7.519692    7.685703
6  1805 52 109   19            2224              2256   7.707063    7.733684
7  1864 57 111   19            1880              2391   7.539027    7.800163
8  2006 44 110   19            2091              2456   7.645398    7.761745
9  2183 55 125   23            1926              2218   7.563201    7.771067
10 2318 51  91   21            1960              2440   7.580700    7.771489
```

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 not 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
      ID WM  IQ Comp Simple_sentence Complex_sentence log_Simple log_Complex
1     95 47  94   19            2154              2441   7.675082    7.758333
2    400 45 118   18            1824              2456   7.508787    7.825245
3    457 42 100   22            1857              2324   7.526718    7.912423
4   1138 41  77   18            1902              2341   7.550661    7.772753
5   1587 54  67   21            1844              2320   7.519692    7.685703
6   1805 52 109   19            2224              2256   7.707063    7.733684
7   1864 57 111   19            1880              2391   7.539027    7.800163
8   2183 55 125   23            1926              2218   7.563201    7.771067
9   2318 51  91   21            1960              2440   7.580700    7.771489
10 2324 43 120   20            1933              2349   7.566828    7.687080
```

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 format (each row is one *observation*).

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

```
> dataRT
    ID Simple_sentence Complex_sentence
1  1138            1902             2341
2  6223            1797             2503
3  6092            2080             2731
4  6232            1856             2375
5  8606            1997             2177
6  6400            1868             2284
7    95            2154             2441
8  2324            1933             2349
9  6656            1900             2371
10 5138            1929             2372
```

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:

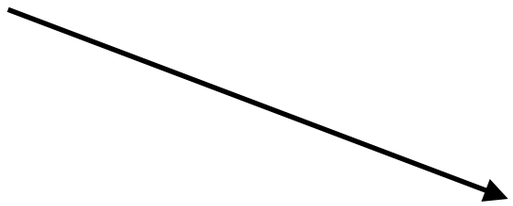
ID	Condition	RT
…	…	…

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
      ID         Condition   RT
1   1138  Simple_sentence 1902
2   6223  Simple_sentence 1797
3   6092  Simple_sentence 2080
4   6232  Simple_sentence 1856
5   8606  Simple_sentence 1997
6   6400  Simple_sentence 1868
7     95  Simple_sentence 2154
8   2324  Simple_sentence 1933
9   6656  Simple_sentence 1900
10 5138  Simple_sentence 1929
```

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

```
> data_wide <- spread(data_long, "Condition", "RT")
> View(data_wide)
```

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

```
> data_wide
      ID Complex_sentence Simple_sentence
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
```

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> <dbl> <dbl> <dbl>
1 Complex_sentence 2405.  2177  2739  132.
2 Simple_sentence  1957.  1694  2356  147.
```

Generating Descriptives - using `psych`

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

> library(psych)
```

You can read documentation about any package by typing :

```
> help(package name)
```

"psych" contains many helpful functions including `describeBy`

To get help on any function, just type:

```
> help(function name)
```

This parameter specifies what data frame and variable we want descriptives for.

This parameter specifies how we want our descriptives grouped.

```
> describeBy(data_long$RT, group = data_long$Condition)

 Descriptive statistics by group
group: Complex_sentence
   vars  n   mean    sd median trimmed    mad  min  max range skew kurtosis    se
X1    1 48 2405.4 131.7   2393  2399.8 108.97 2177 2739   562 0.42     0.03 19.01
------------------------------------------------------------------------------
group: Simple_sentence
   vars  n    mean     sd median trimmed    mad  min  max range skew kurtosis    se
X1    1 48 1957.46 147.41 1927.5 1947.28 111.19 1694 2356   662 0.79    -0.11 21.28
```

If we had a 2 x 2 design with `Factor_1`, `Factor _2` and one DV in a data frame called `data`, to calculate the descriptives for each of our 4 conditions we would group like this:

```
> describeBy(data$DV, group=list(data$Factor_1, data$Factor_2))
```

`dplyr` **or** `psych`**?**

- Although both packages allow you to generate the same descriptives, the `dplyr` functions are part of the Tidyverse and share the same underlying philosophy. Different Tidyverse packages and functions play well with each other.

- It's all down to personal preference - oftentimes there are many different ways to achieve the same thing…

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.

```
> head(data)
  Participant Condition   RT
1           1         1  879
2           1         2 1027
3           1         3 1108
4           1         4  765
5           2         1 1042
6           2         2 1050
```

- 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
data$Condition <- recode (data$Condition, “1" = "PrimeA_TargetA", “2"
= “PrimeA_TargetB", “3" = "PrimeB_TargetA", “4” = "PrimeB_TargetB")
```

- Now our data frame looks like this:

```
> head(data)
  Participant      Condition   RT
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
```

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

```
> data <- separate(data, col = "Condition", into = c("Prime", "Target"),
sep = "_")
> data
   Participant  Prime  Target   RT
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
```

- 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).

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

```
> data <- unite(data, col = "Condition", c("Prime", "Target"), sep = "_")
> wide_data <- spread(data, key = "Condition", value = “RT")
> wide_data

   Participant PrimeA_TargetA PrimeA_TargetB PrimeB_TargetA PrimeB_TargetB
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
```

- 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.

```
data %>%
  separate(col = "Condition", into = c("Prime", "Target"), sep = "_") %>%
  unite(col = "Condition", c("Prime", "Target"), sep = "_") %>%
  spread(key = "Condition", value = "RT")
```

The NHANES dataset

The NHANES dataset 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 1960s.

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.

We can find out a little about the structure of the dataset using the `colnames()` and `head()` functions. The first will tell us the names of the column variables, and the second will show us the first few rows of data.

```
> colnames(NHANES)
 [1] "ID"               "SurveyYr"         "Gender"           "Age"
 [5] "AgeDecade"        "AgeMonths"        "Race1"            "Race3"
 [9] "Education"        "MaritalStatus"    "HHIncome"         "HHIncomeMid"
[13] "Poverty"          "HomeRooms"        "HomeOwn"          "Work"
[17] "Weight"           "Length"           "HeadCirc"         "Height"
[21] "BMI"              "BMICatUnder20yrs" "BMI_WHO"          "Pulse"
[25] "BPSysAve"         "BPDiaAve"         "BPSys1"           "BPDia1"
[29] "BPSys2"           "BPDia2"           "BPSys3"           "BPDia3"
[33] "Testosterone"     "DirectChol"       "TotChol"          "UrineVol1"
[37] "UrineFlow1"       "UrineVol2"        "UrineFlow2"       "Diabetes"
[41] "DiabetesAge"      "HealthGen"        "DaysPhysHlthBad"  "DaysMentHlthBad"
[45] "LittleInterest"   "Depressed"        "nPregnancies"     "nBabies"
[49] "Age1stBaby"       "SleepHrsNight"    "SleepTrouble"     "PhysActive"
[53] "PhysActiveDays"   "TVHrsDay"         "CompHrsDay"       "TVHrsDayChild"
[57] "CompHrsDayChild"  "Alcohol12PlusYr"  "AlcoholDay"       "AlcoholYear"
[61] "SmokeNow"         "Smoke100"         "Smoke100n"        "SmokeAge"
[65] "Marijuana"        "AgeFirstMarij"    "RegularMarij"     "AgeRegMarij"
[69] "HardDrugs"        "SexEver"          "SexAge"           "SexNumPartnLife"
[73] "SexNumPartYear"   "SameSex"          "SexOrientation"   "PregnantNow"
```

```
> head(NHANES)
# A tibble: 6 x 76
     ID SurveyYr Gender   Age AgeDecade AgeMonths Race1 Race3 Education MaritalStatus HHIncome
  <int> <fct>    <fct>  <int> <fct>          <int> <fct> <fct> <fct>     <fct>         <fct>
1 51624 2009_10  male      34 " 30-39"         409 White NA    High Sch… Married       25000-3…
2 51624 2009_10  male      34 " 30-39"         409 White NA    High Sch… Married       25000-3…
3 51624 2009_10  male      34 " 30-39"         409 White NA    High Sch… Married       25000-3…
4 51625 2009_10  male       4 " 0-9"            49 Other NA    NA        NA            20000-2…
5 51630 2009_10  female    49 " 40-49"         596 White NA    Some Col… LivePartner   35000-4…
6 51638 2009_10  male       9 " 0-9"          115 White NA    NA        NA            75000-9…
# ... with 65 more variables: HHIncomeMid <int>, Poverty <dbl>, HomeRooms <int>,
```

We can use the `nrow()` function to tell us how many rows there are and the `length()` function with `unique()` to tell us how many unique participant IDs there are:

```
> nrow(NHANES)
[1] 10000
> length(unique(NHANES$ID))
[1] 6779
```

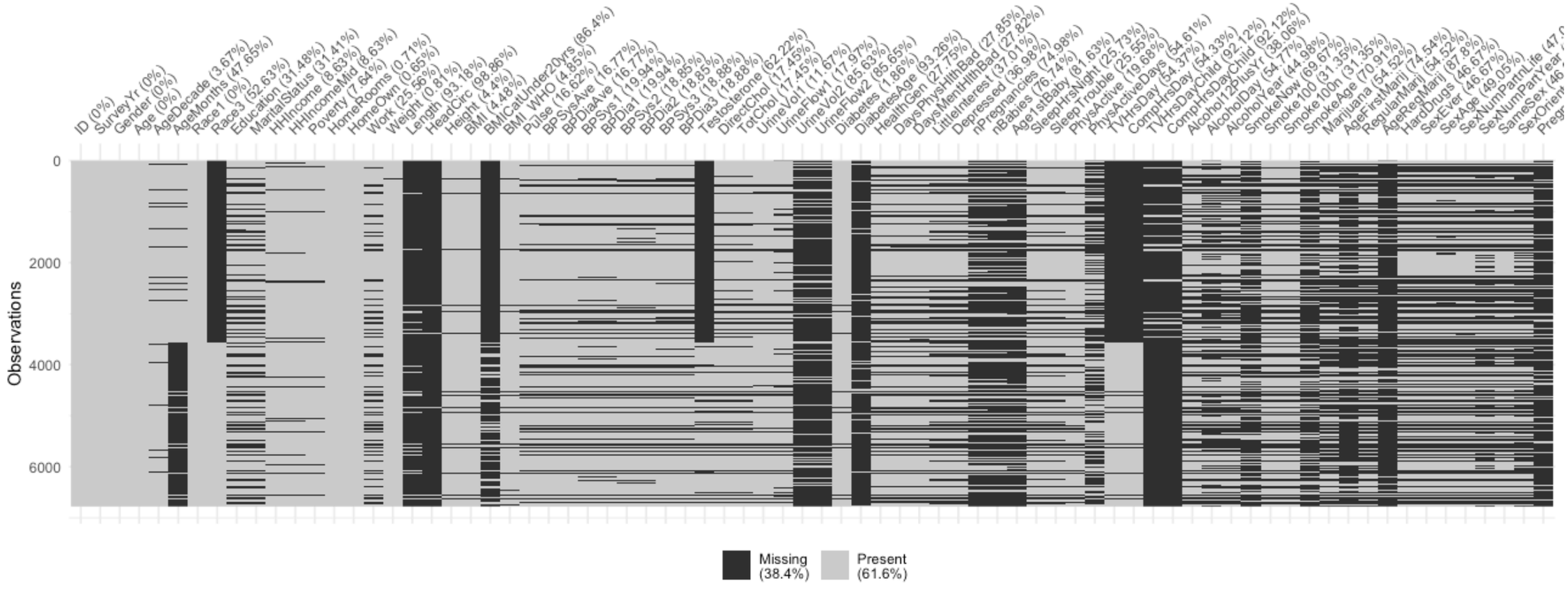
- So it appears we have a number of duplicate ID numbers - let's make a new data variable that has only one instance of each ID:

```
> NHANES_tidied <- NHANES %>% distinct(ID, .keep_all = TRUE)
> nrow(NHANES_tidied)
[1] 6779
```

- We now have a tidied dataset with 6, 779 rows (one for each unique ID number).

- We can use `visdat::vis_miss()` to get a feeling for the amount of data that might be missing in `NHANES_tidied`:

```
> vis_miss(NHANES_tidied)
```



```
> colnames(NHANES_tidied)
 [1] "ID"                "SurveyYr"          "Gender"            "Age"
 [5] "AgeDecade"         "AgeMonths"         "Race1"             "Race3"
 [9] "Education"         "MaritalStatus"     "HHIncome"          "HHIncomeMid"
[13] "Poverty"           "HomeRooms"         "HomeOwn"           "Work"
[17] "Weight"            "Length"            "HeadCirc"          "Height"
[21] "BMI"               "BMICatUnder20yrs"  "BMI_WHO"           "Pulse"
[25] "BPSysAve"          "BPDiaAve"          "BPSys1"            "BPDia1"
[29] "BPSys2"            "BPDia2"            "BPSys3"            "BPDia3"
[33] "Testosterone"      "DirectChol"        "TotChol"           "UrineVol1"
[37] "UrineFlow1"        "UrineVol2"         "UrineFlow2"        "Diabetes"
[41] "DiabetesAge"       "HealthGen"         "DaysPhysHlthBad"   "DaysMentHlthBad"
[45] "LittleInterest"    "Depressed"         "nPregnancies"      "nBabies"
[49] "Age1stBaby"        "SleepHrsNight"     "SleepTrouble"      "PhysActive"
[53] "PhysActiveDays"    "TVHrsDay"          "CompHrsDay"        "TVHrsDayChild"
[57] "CompHrsDayChild"   "Alcohol12PlusYr"   "AlcoholDay"        "AlcoholYear"
[61] "SmokeNow"          "Smoke100"          "Smoke100n"         "SmokeAge"
[65] "Marijuana"         "AgeFirstMarij"     "RegularMarij"      "AgeRegMarij"
[69] "HardDrugs"         "SexEver"           "SexAge"            "SexNumPartnLife"
[73] "SexNumPartYear"    "SameSex"           "SexOrientation"    "PregnantNow"
```

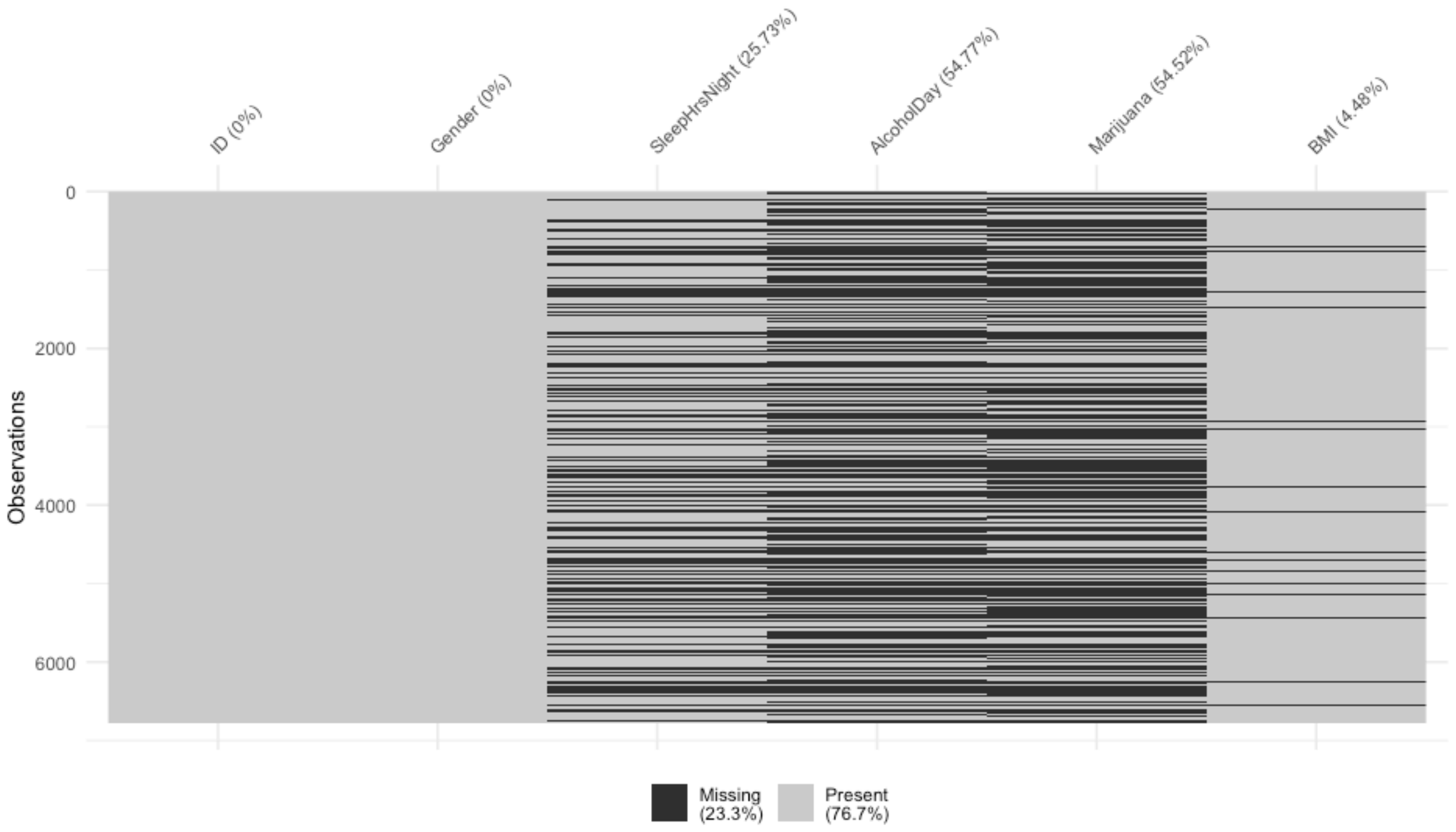
- We still have a plethora of columns though - let's cut that number down.

- Let's use `select()` to keep participant ID, Gender, SleepHrsNight, AlcoholDay, Marijuana, and BMI.

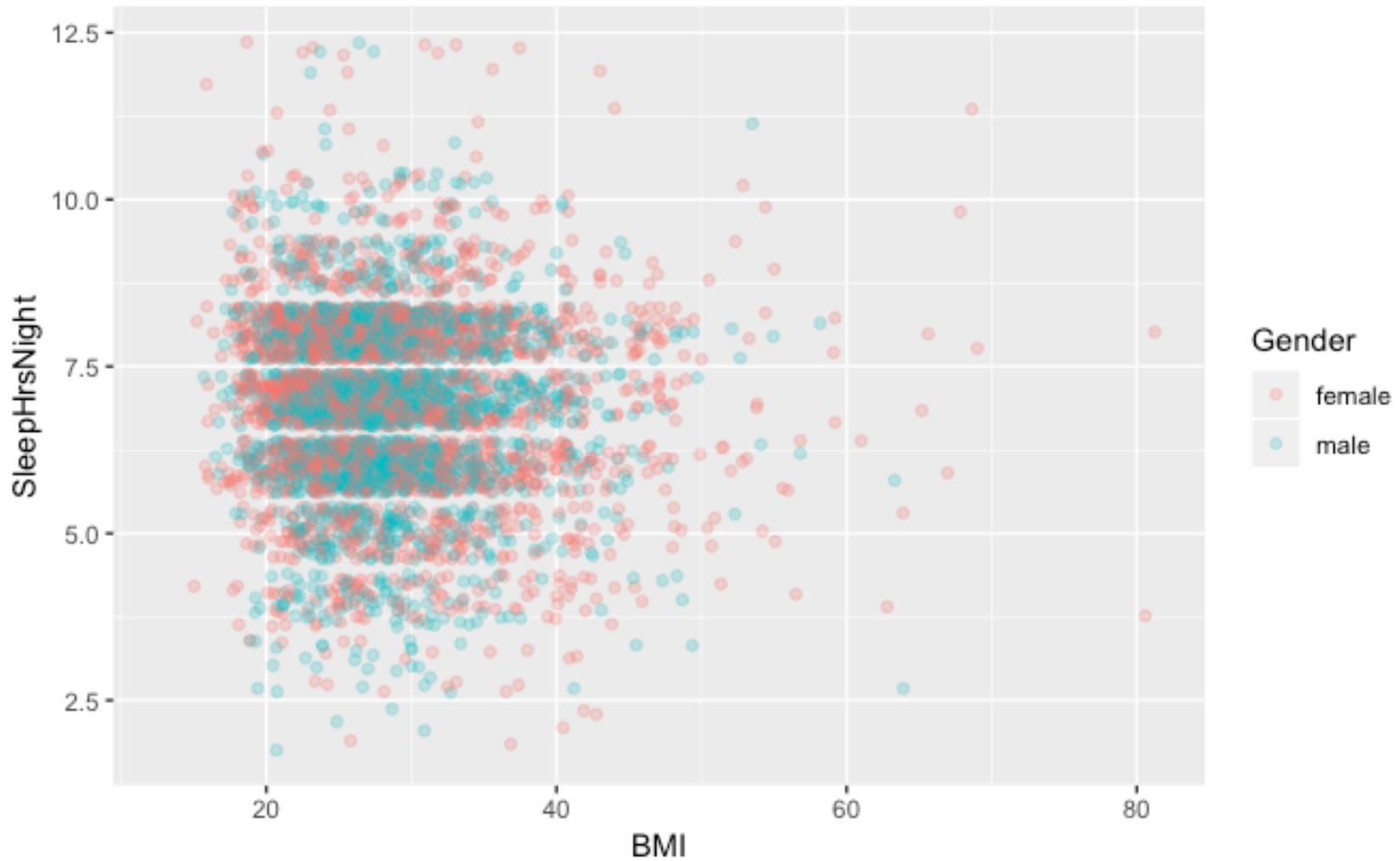
```
> NHANES_focused <- select(NHANES_tidied, ID, Gender, SleepHrsNight,
AlcoholDay, Marijuana, BMI)
> NHANES_focused
# A tibble: 6,779 x 6
      ID Gender SleepHrsNight AlcoholDay Marijuana   BMI
   <int> <fct>          <int>      <int> <fct>     <dbl>
 1 51624 male               4         NA Yes        32.2
 2 51625 male              NA         NA NA         15.3
 3 51630 female             8          2 Yes        30.6
 4 51638 male              NA         NA NA         16.8
 5 51646 male              NA         NA NA         20.6
 6 51647 female             8          3 Yes        27.2
 7 51654 male               7          1 NA         23.7
 8 51656 male               5          2 Yes        23.7
 9 51657 male               4          6 Yes        26.0
10 51659 female            NA         NA NA         19.2
# ... with 6,769 more rows
```

- We could perhaps now visually explore this smaller dataset…

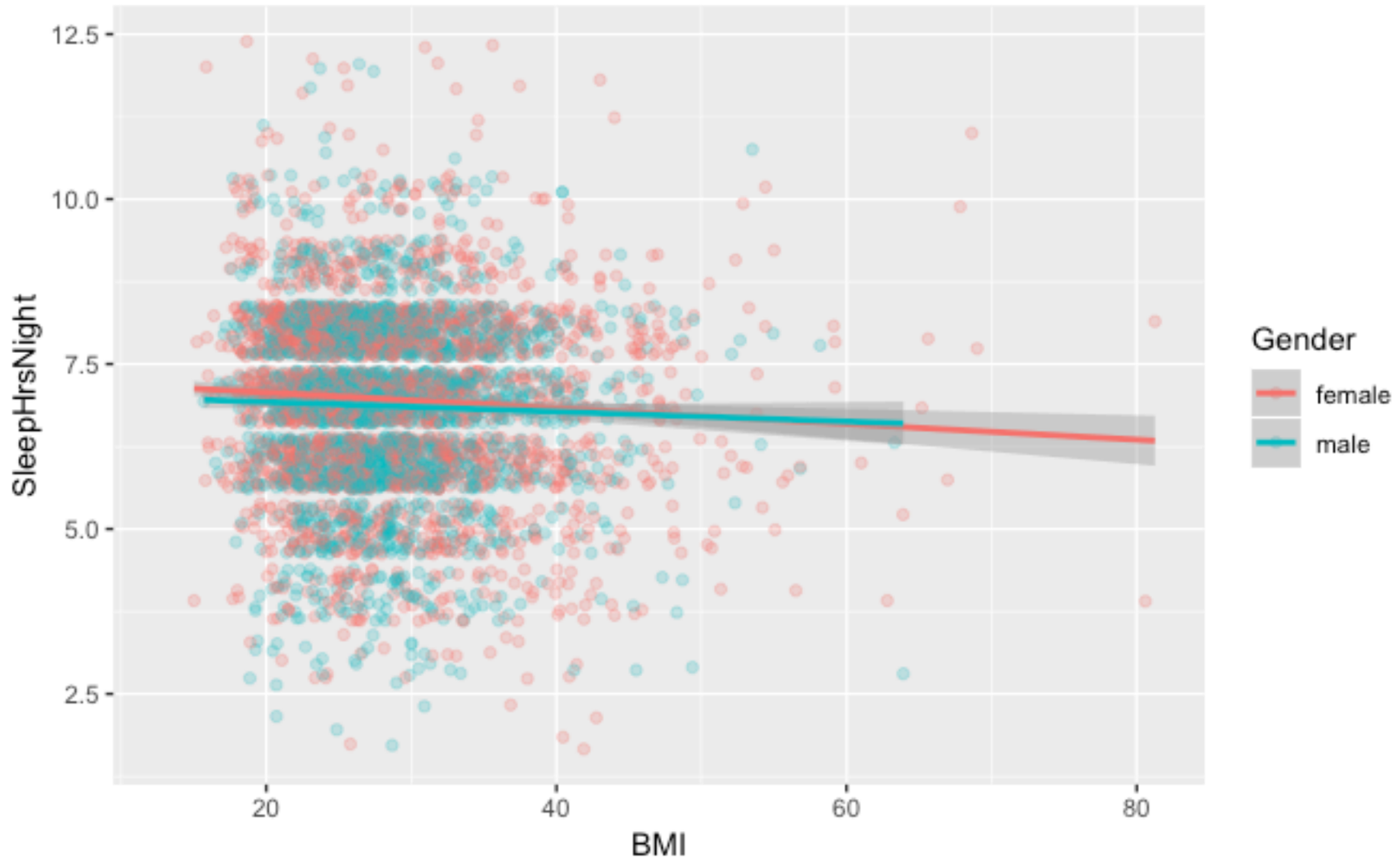
```
> vis_miss(NHANES_focused)
```



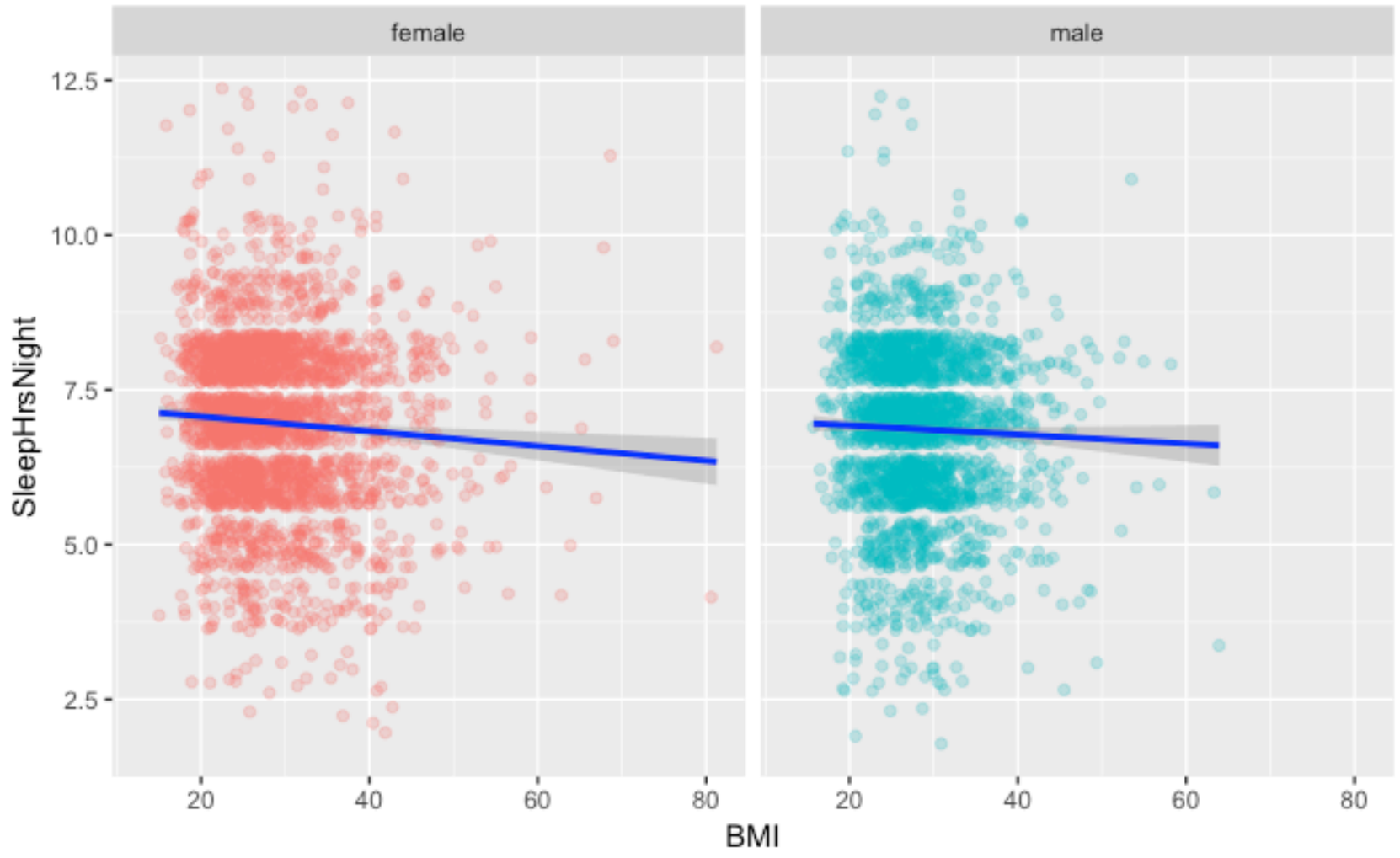
```
NHANES_focused %>%
  ggplot(aes(x = BMI, y = SleepHrsNight, colour = Gender)) +
  geom_jitter(alpha = .25)
```



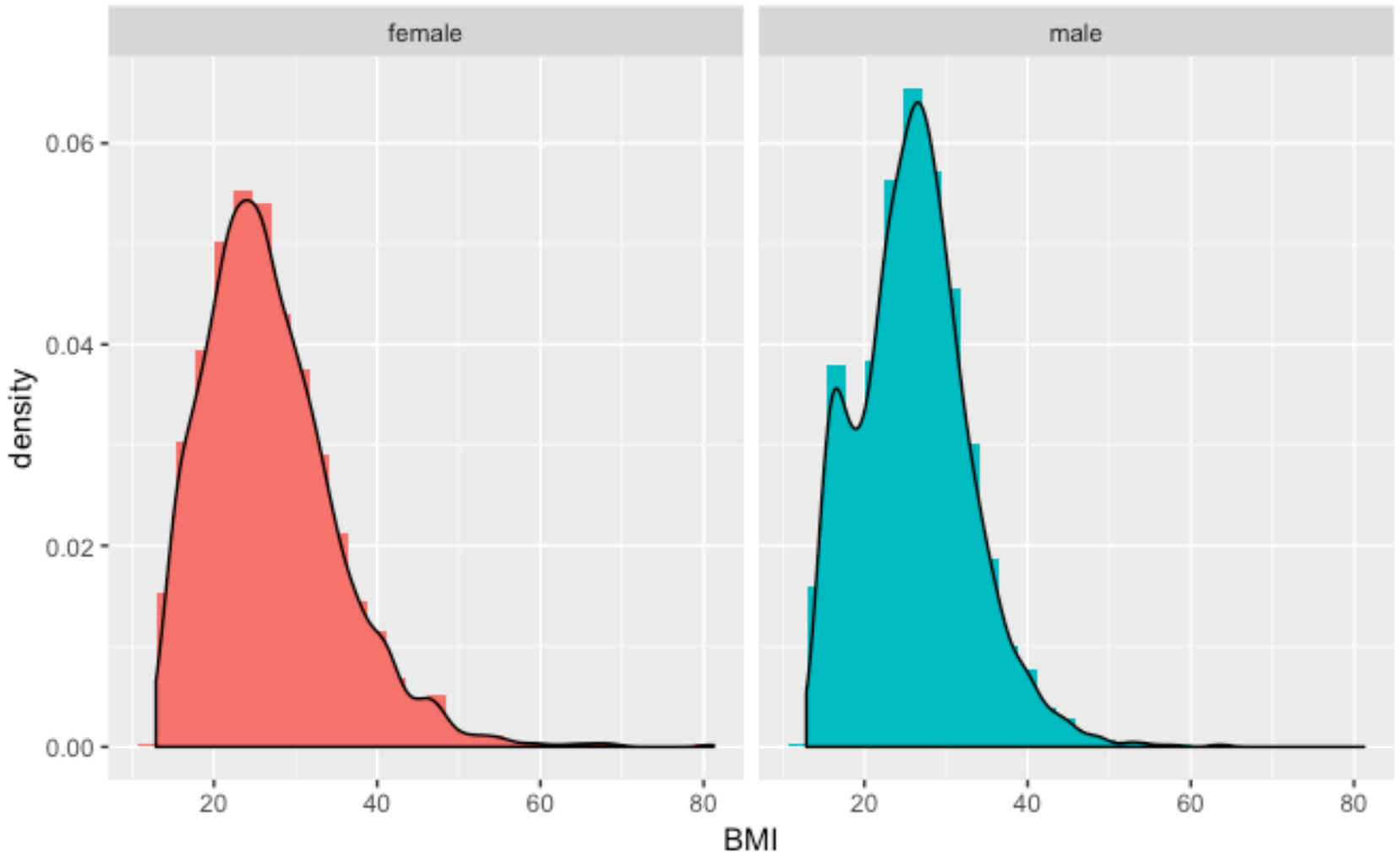
```
NHANES_focused %>%
  ggplot(aes(x = BMI, y = SleepHrsNight, colour = Gender)) +
  geom_jitter(alpha = .25) +
  geom_smooth(method = "lm")
```



```
NHANES_focused %>%
  ggplot(aes(x = BMI, y = SleepHrsNight, colour = Gender)) +
  geom_jitter(alpha = .25) +
  geom_smooth(method = "lm", colour = "blue") +
  guides(colour = FALSE) +
  facet_wrap(~ Gender)
```



```
NHANES_focused %>%
  ggplot(aes(x = BMI, fill = Gender)) +
  geom_histogram(aes(y = ..density..)) +
  geom_density(aes(y = ..density..)) +
  guides(fill = FALSE) +
  facet_wrap(~ Gender)
```



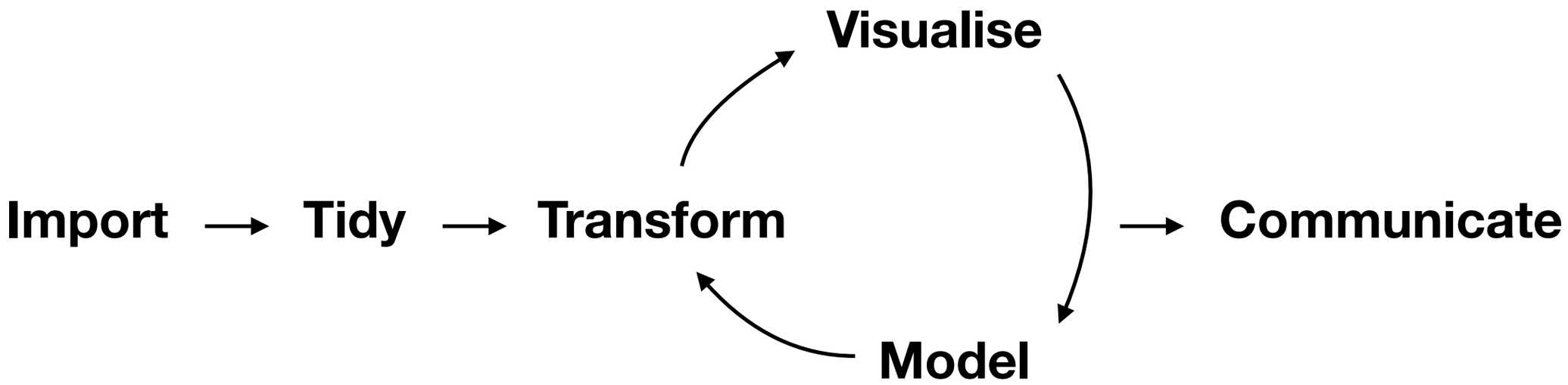
This is just a small example of functions in the `dplyr` and `tidyr` packages that allow you to tidy, transform, and reshape your data all ready for analysis or visualisation. All of your code for doing this should appear at the start of your analysis script so that others (and you in 5 years or 5 days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

And this gets us nicely onto data visualisation…

Visualise

Data plots using `ggplot()`

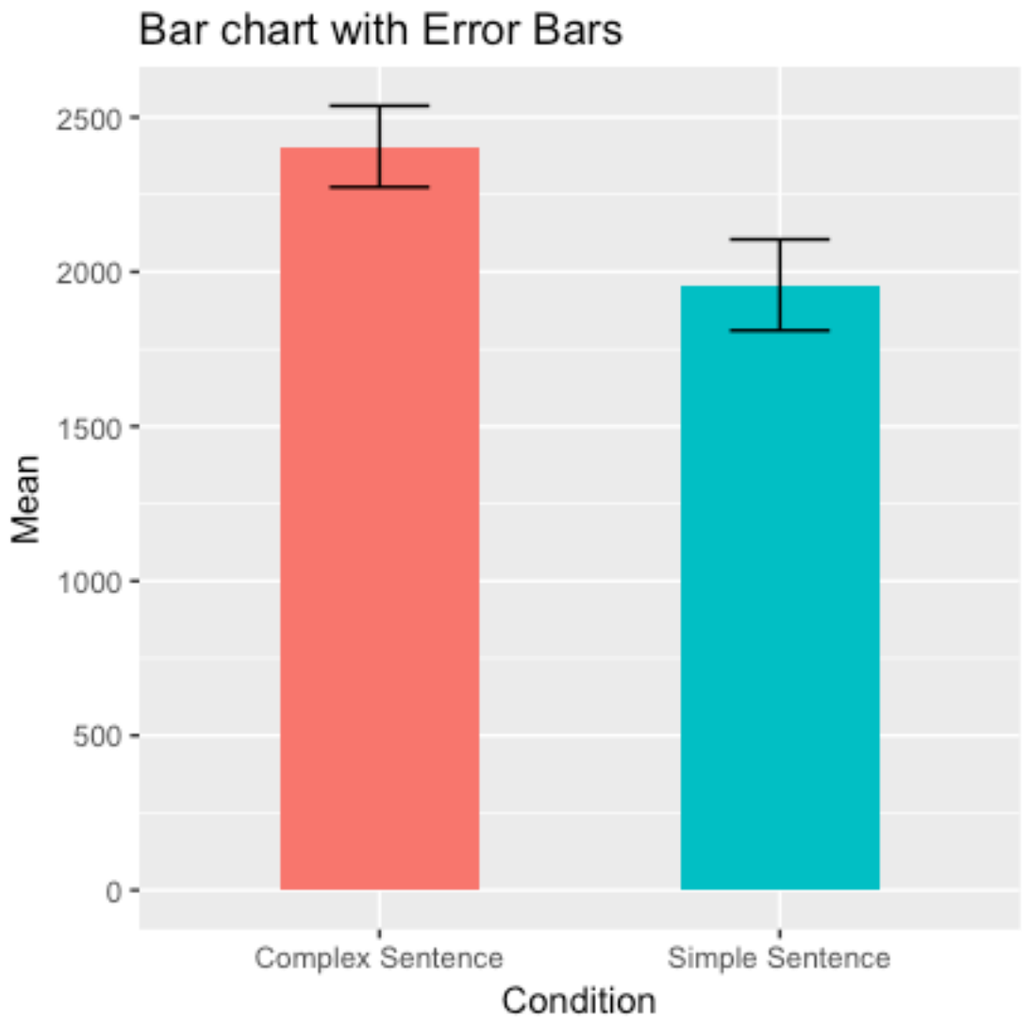


Visualising Your Data

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

```
> library(ggplot2) # Loads just ggplot2

> library(tidyverse) # Loads all the Tidyverse packages incl. ggplot2
```

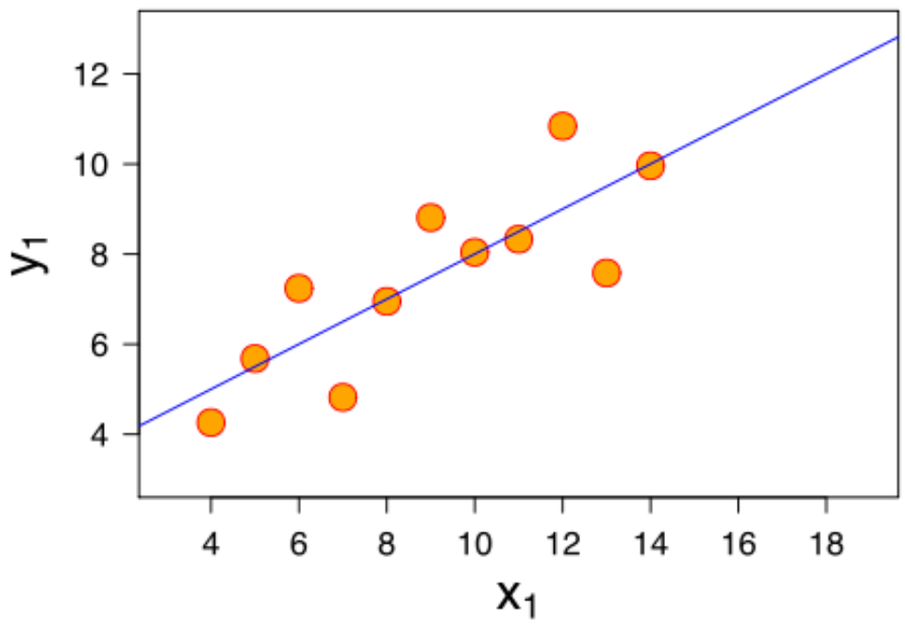
Bar Graphs

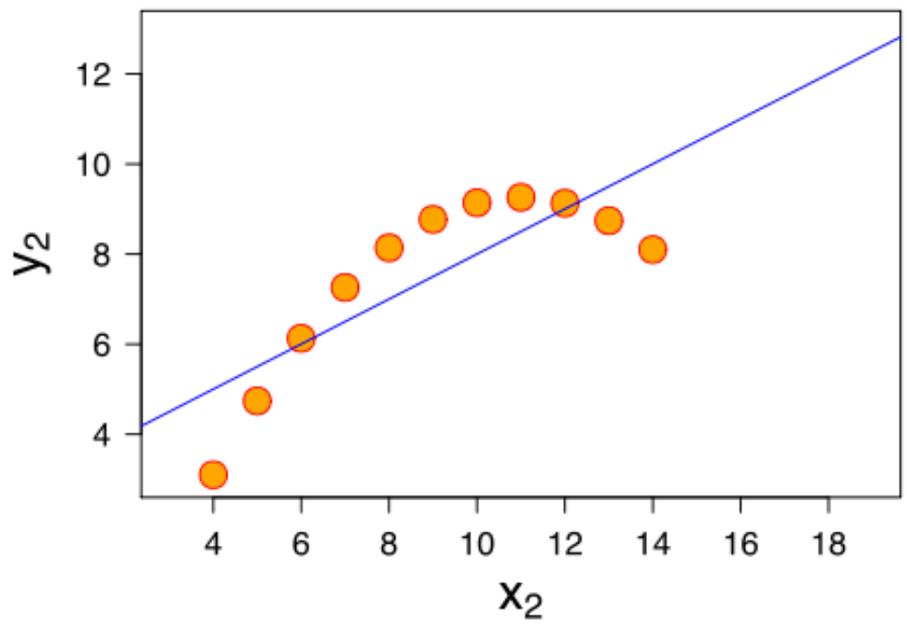


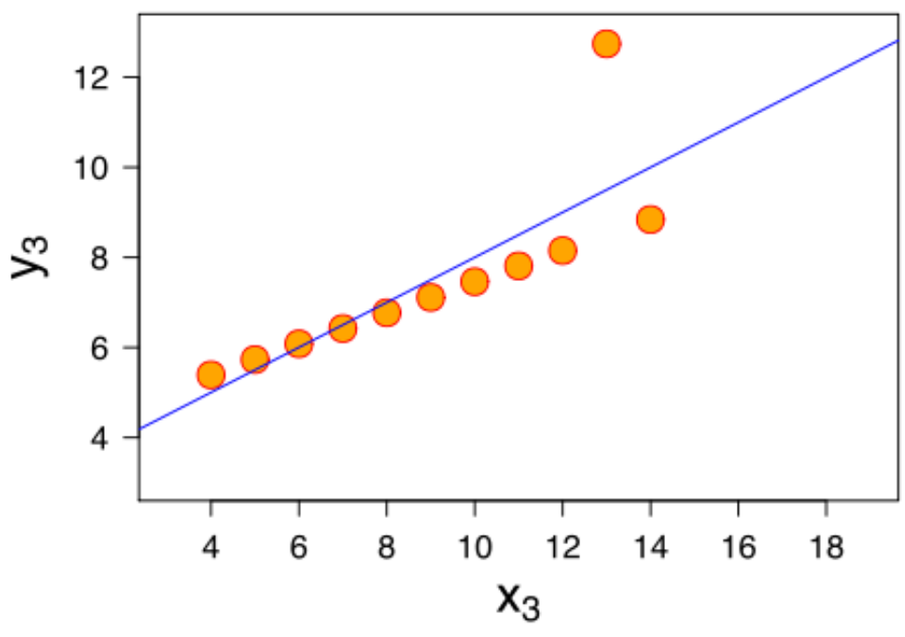
Bar graphs tend to be quite limited in terms of what they communicate. Here they communicate the means for levels of a factor and information about variance. But they don't tell us anything about the *distribution* of the data.

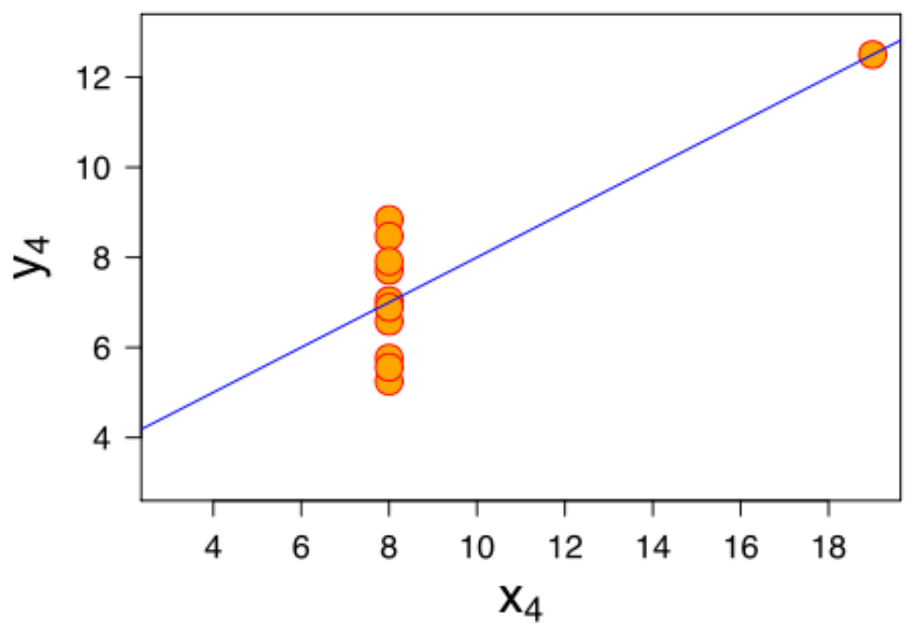
```
> data_summ <- data_long %>% group_by(Condition) %>% summarise(Mean = mean(RT), sd = sd(RT))
> ggplot(data_summ, aes(x = Condition, y = Mean, group = Condition, fill = Condition, ymin =
Mean-sd, ymax = Mean+sd)) + geom_bar(stat = "identity", width = .5) + geom_errorbar(width = .25)
+  ggtitle("Bar chart with Error Bars") + guides(fill = FALSE)
```

Anscombe's Quartet

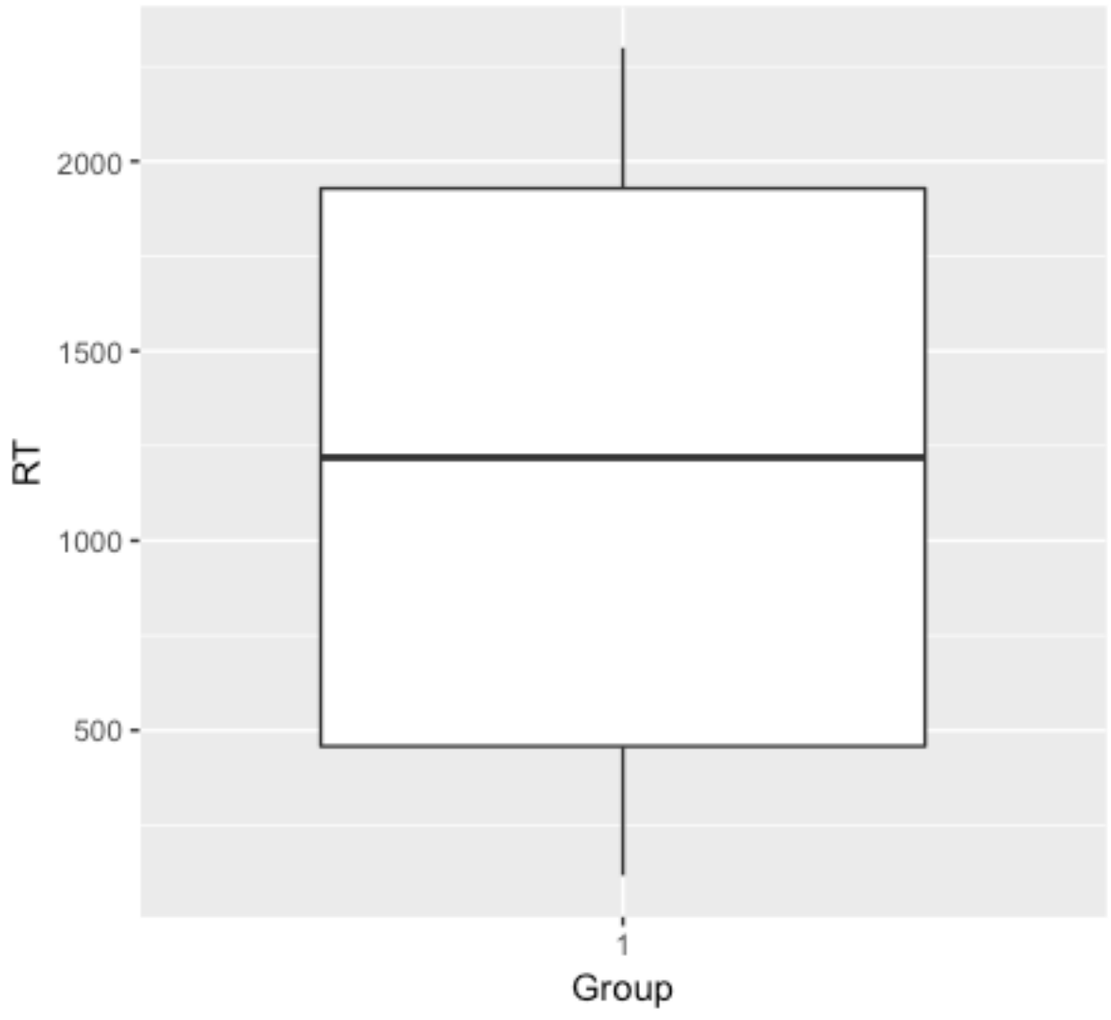






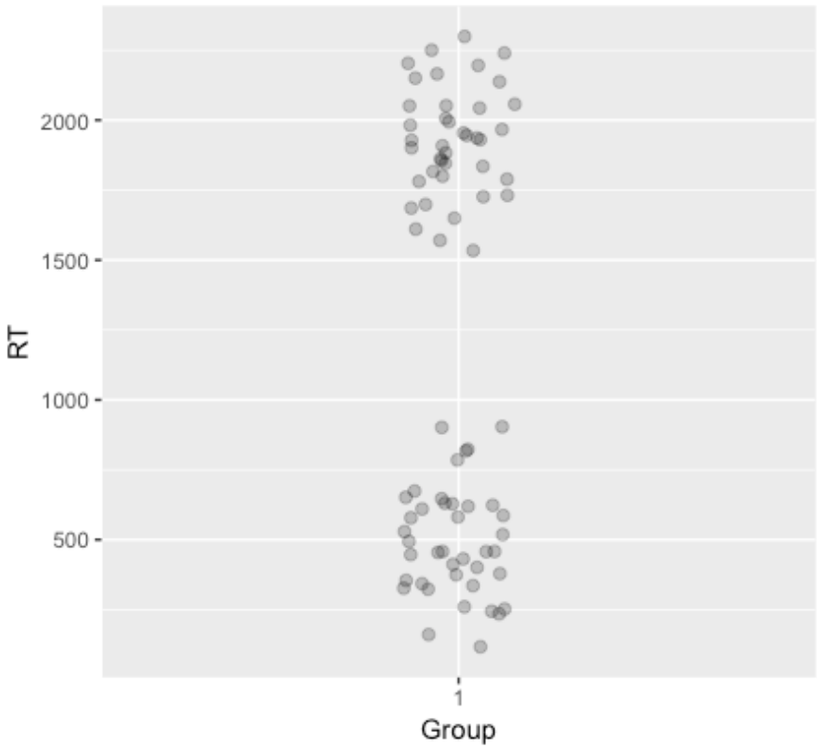


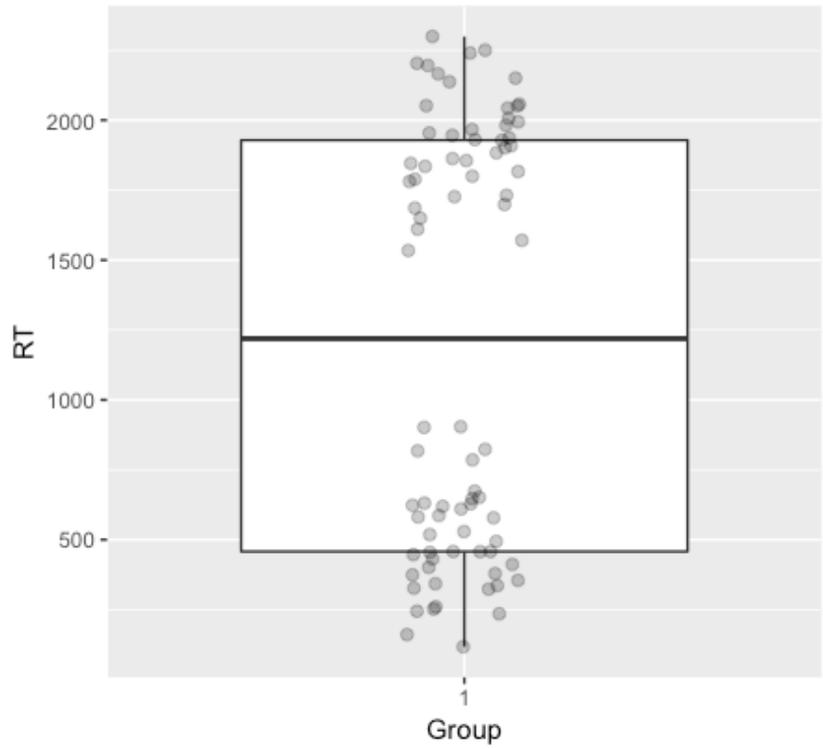
Plots Based on Aggregated Data Can Mislead…

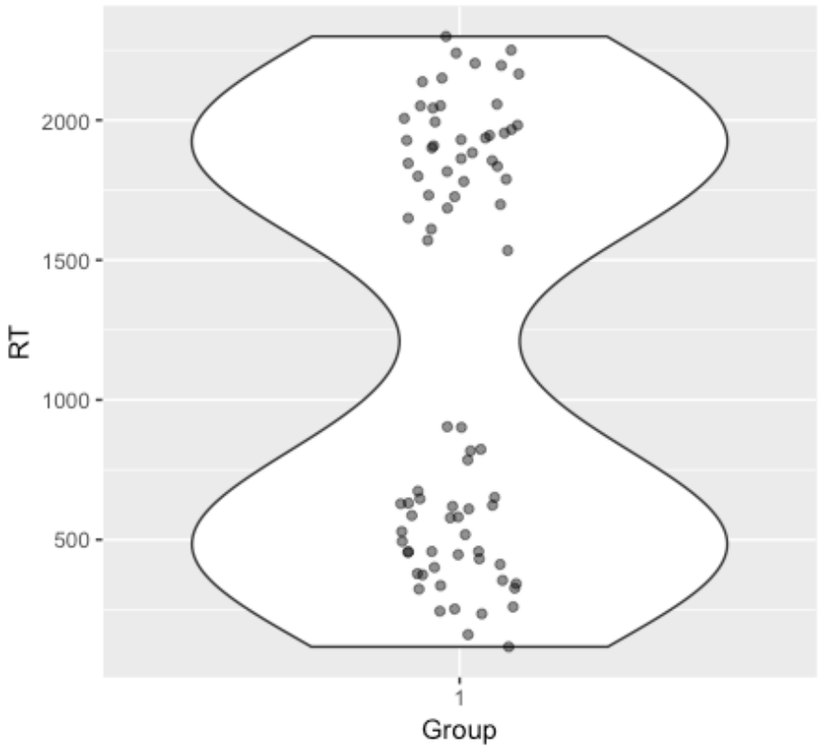


You might make one set of inferences based on this boxplot - maybe a measure of central tendency around 1,250 with the 25th and 75th percentiles associated with the data being ~480 to ~1,980…

But look more closely at the actual data…

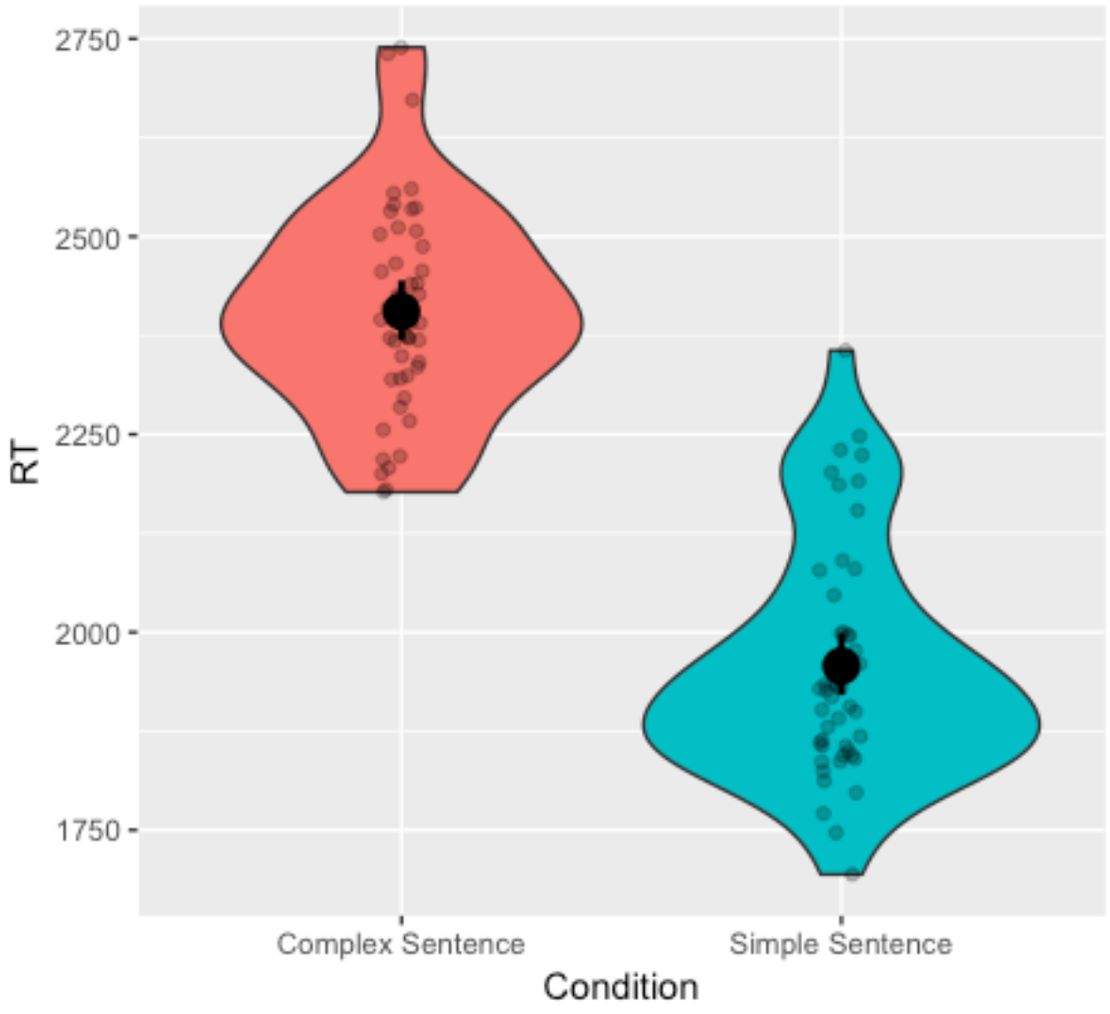






The data are clearly bimodal with no actual data point near the mean. **Distribution shape matters** and we need to capture that in our data visualisations.

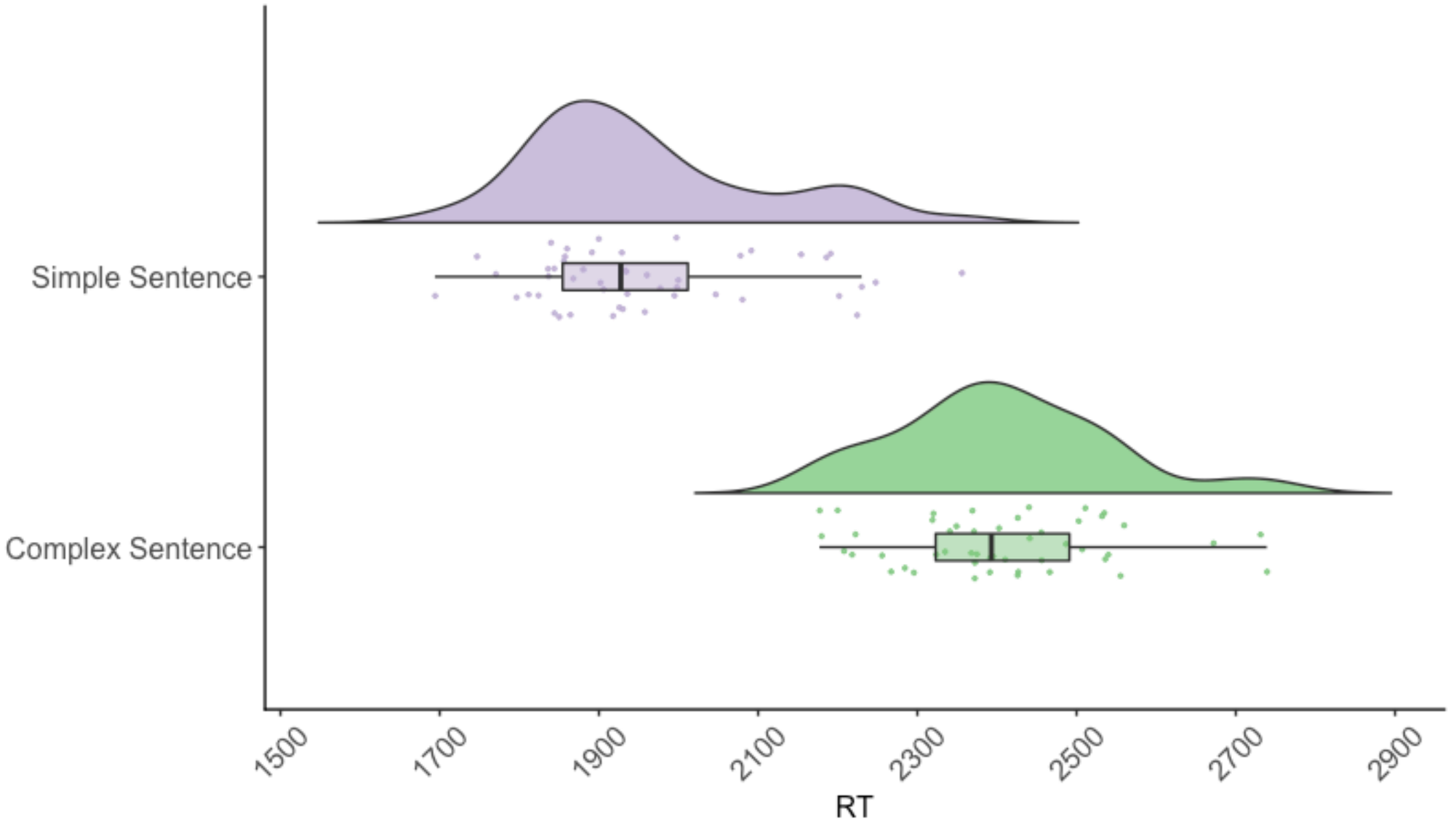
Violin Plots



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.

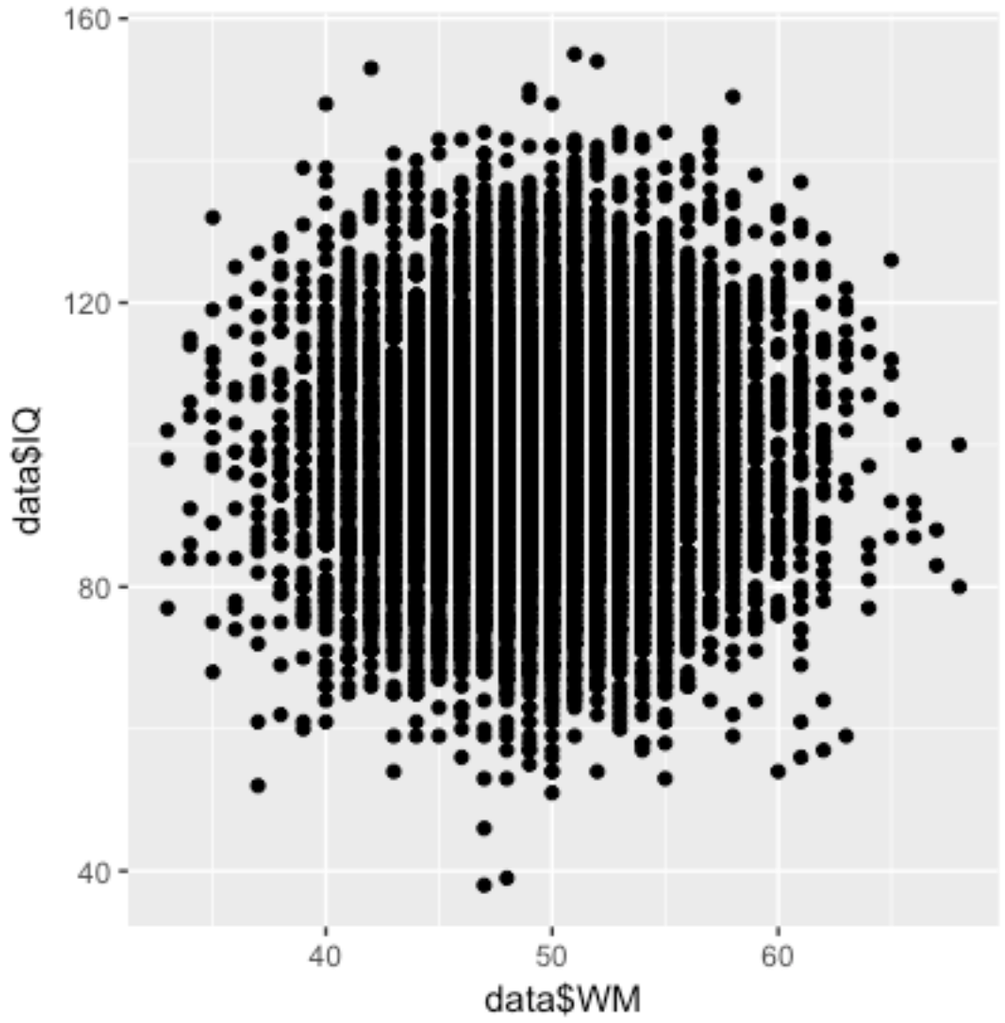
```
ggplot (data_long, aes (x = Condition, y = RT, group = Condition, fill =
Condition)) + geom_violin() + geom_jitter(alpha = .25, position =
position_jitter(0.05)) + guides(colour = FALSE, fill = FALSE) + stat_summary
(fun.data = “mean_cl_boot", colour = "black", size = 1)
```

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.)

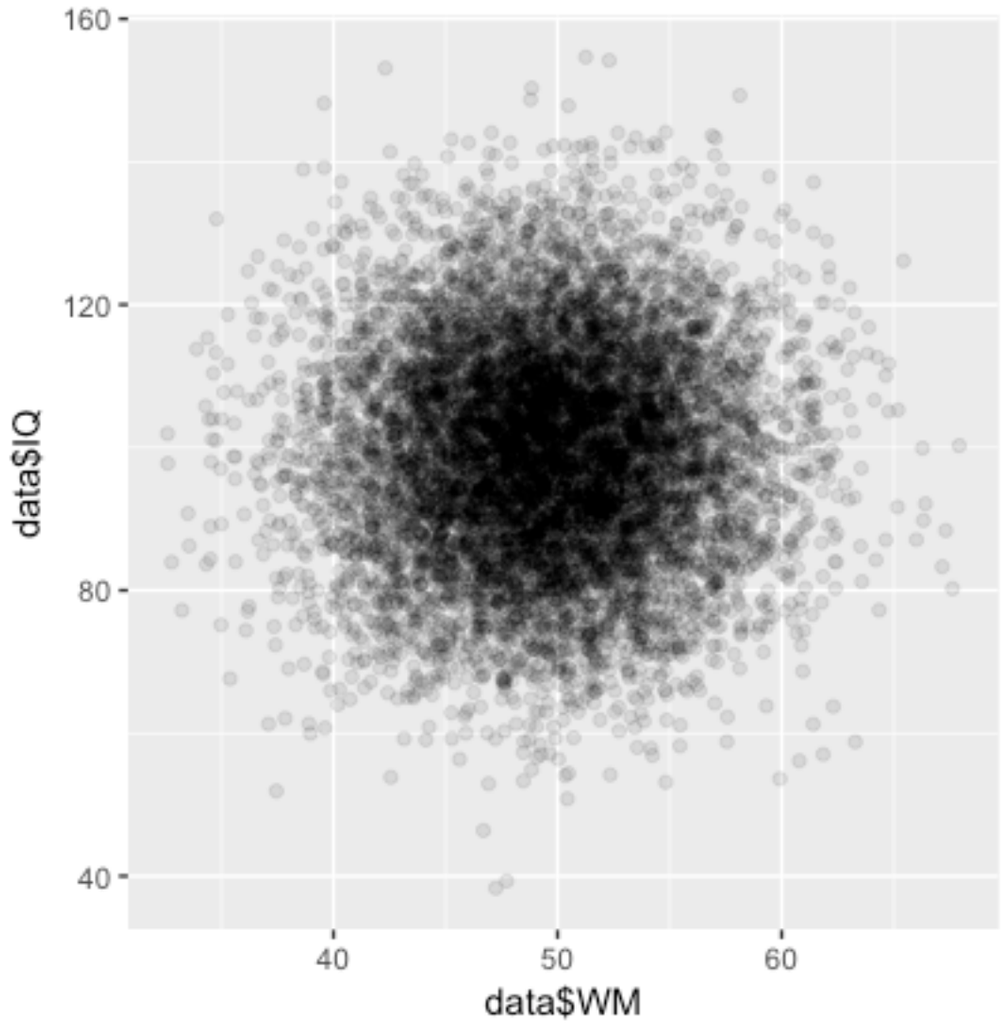
Plotting IQ against WM for our 10,000 participants



The problem of over-plotting - as we have many data points, a number are plotted on top of each other so it is tricky to get a feel for the data.

```
ggplot(data, aes(x = WM, y = IQ)) + geom_point()
```

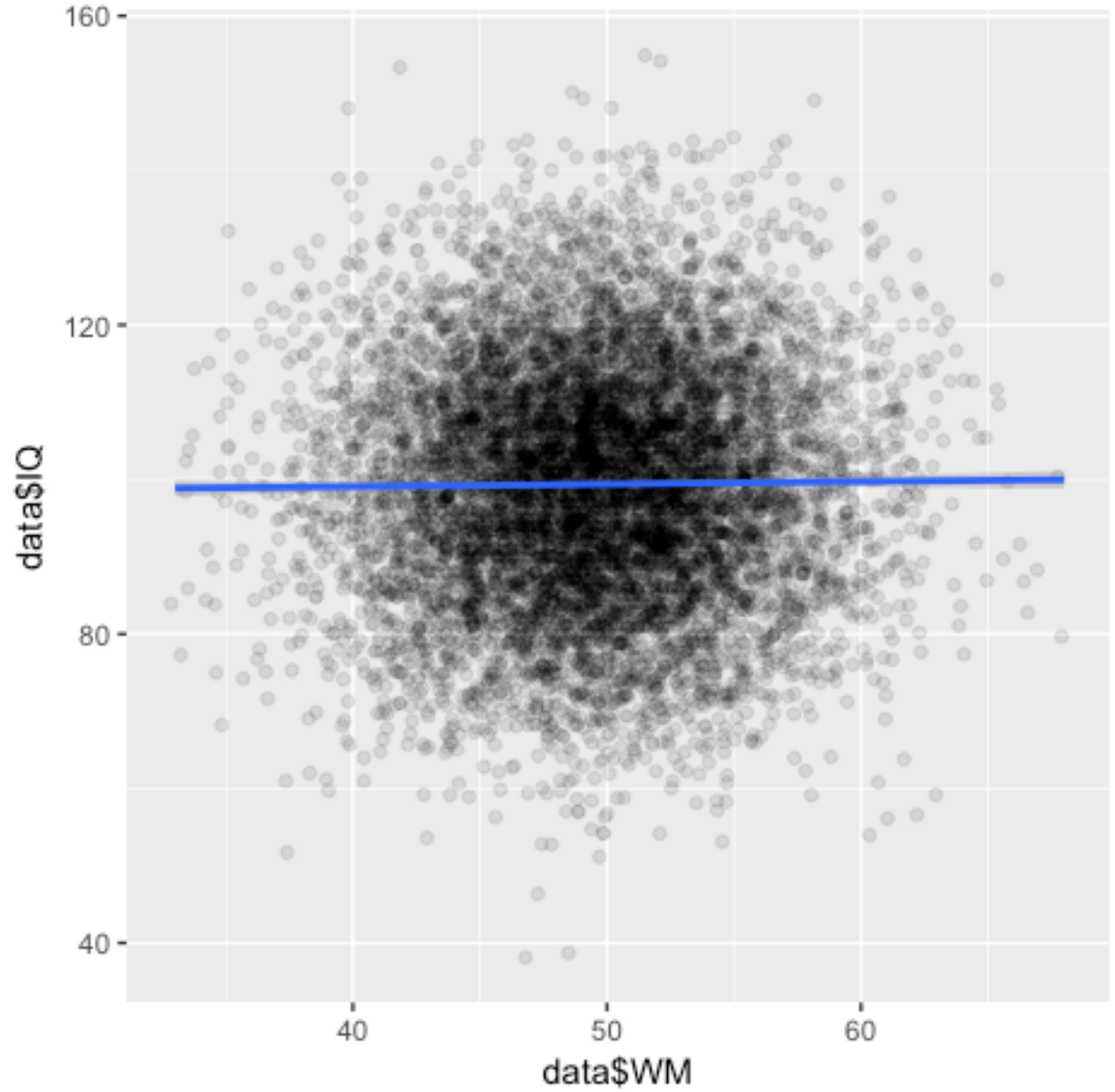
Plotting IQ against WM for our 10,000 participants



To avoid over-plotting, you can jitter the points and set them to be translucent via the alpha parameter.

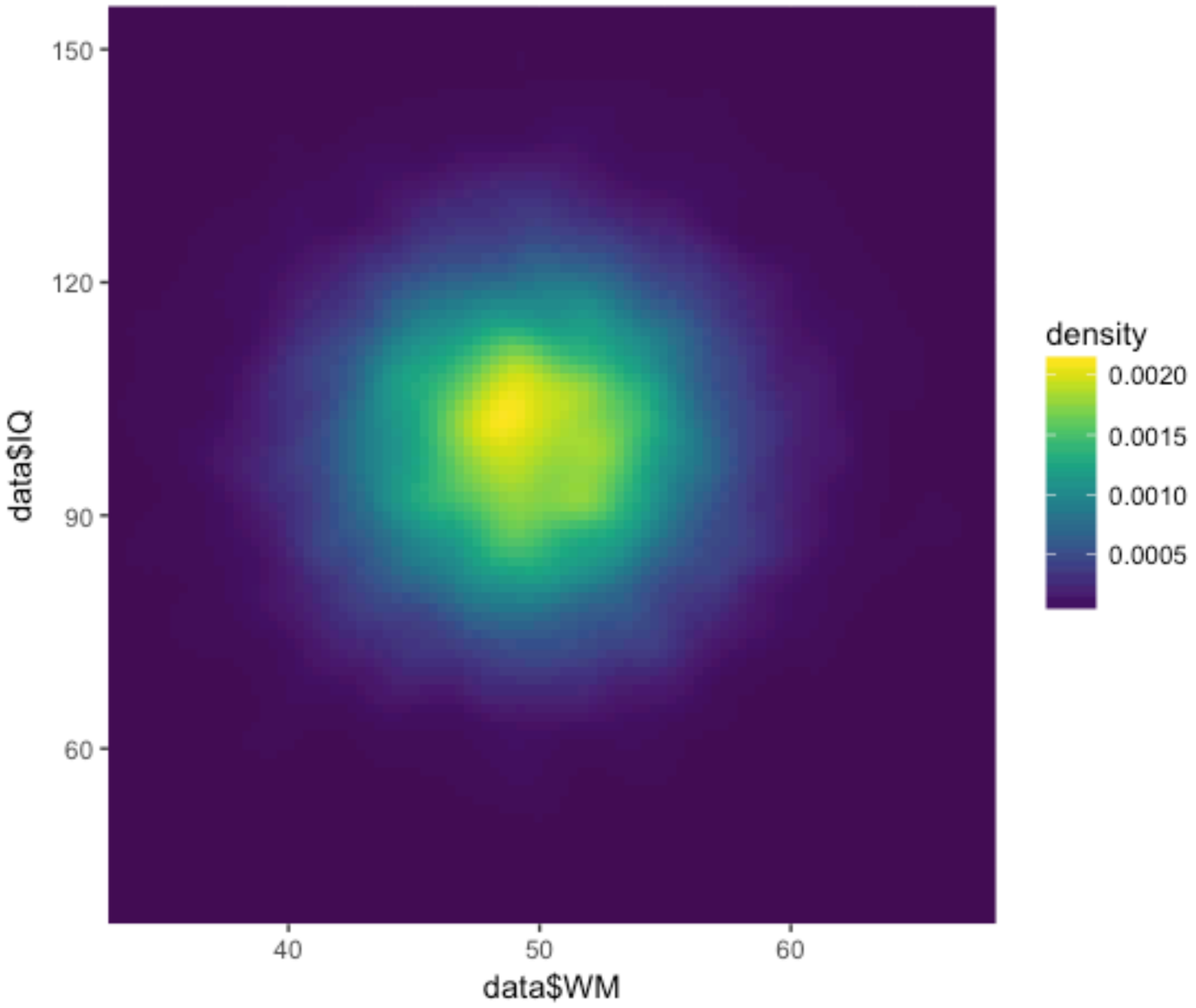
```
> ggplot(data, aes(x = WM, y = IQ)) + geom_jitter(alpha = .1,
position = position_jitter(0.5))
```

With a regression line




```
> ggplot(data, aes(x = WM, y = IQ)) + geom_jitter(alpha = .1,
position = position_jitter(0.5)) + geom_smooth(method = "lm")
```

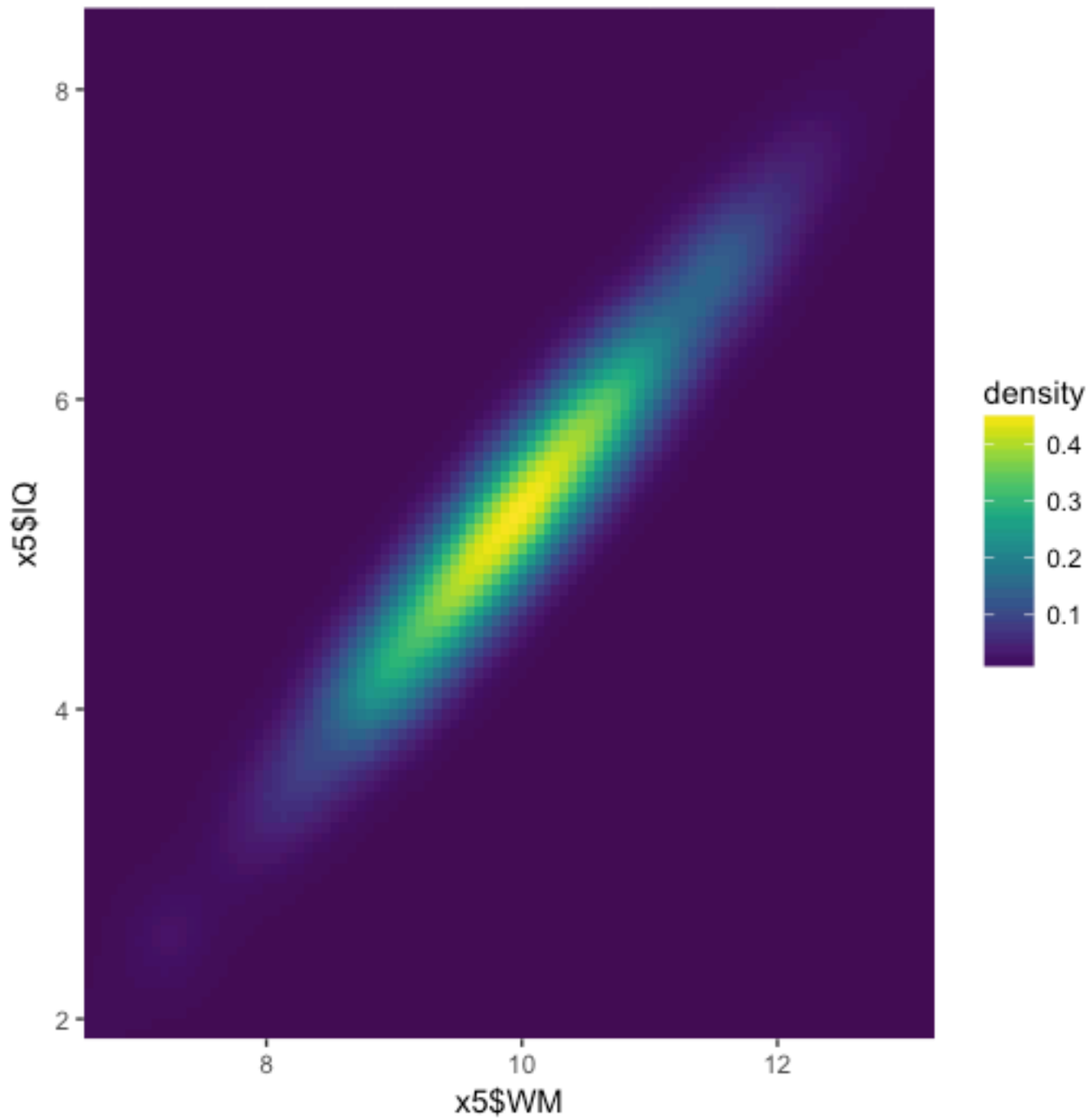
And as a Density Heat Map




```
> ggplot(data, aes(x = WM, y = IQ)) + stat_density_2d(aes(fill = ..density..),
geom = 'raster', contour = FALSE) + scale_fill_viridis() +
coord_cartesian(expand = FALSE)
```

If IQ and WM were perfectly (positively) correlated, we'd have something like this…

```
> #creating two perfectly
correlated variables
> set.seed(1234)
> mysigma <- matrix(c(1,1,1,1),
2,2)
> x1 <- mvrnorm(n = 1000,
c(5.3,10), mysigma)
> x5 <- as.data.frame(x1)
> colnames(x5) <- c("IQ", "WM")

> ggplot(x5, aes(x = WM, y = IQ))
+ stat_density_2d(aes(fill
= ..density..), geom = 'raster',
contour = FALSE) +
scale_fill_viridis() +
coord_cartesian(expand = FALSE)
```



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:

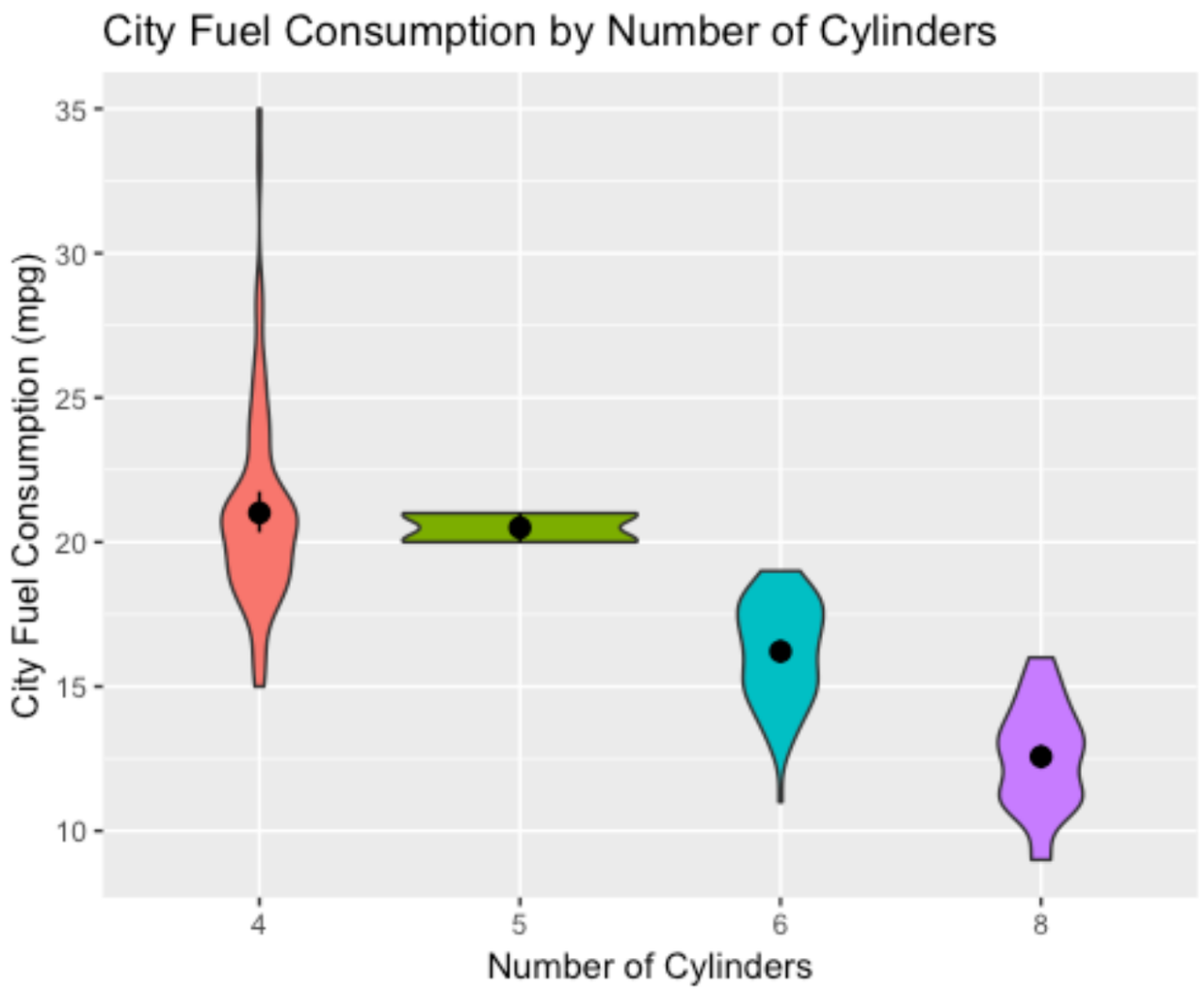
```
> unique(mpg$manufacturer)
 [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"
 [8] "land rover" "lincoln"    "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"
[15] "volkswagen"
```

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

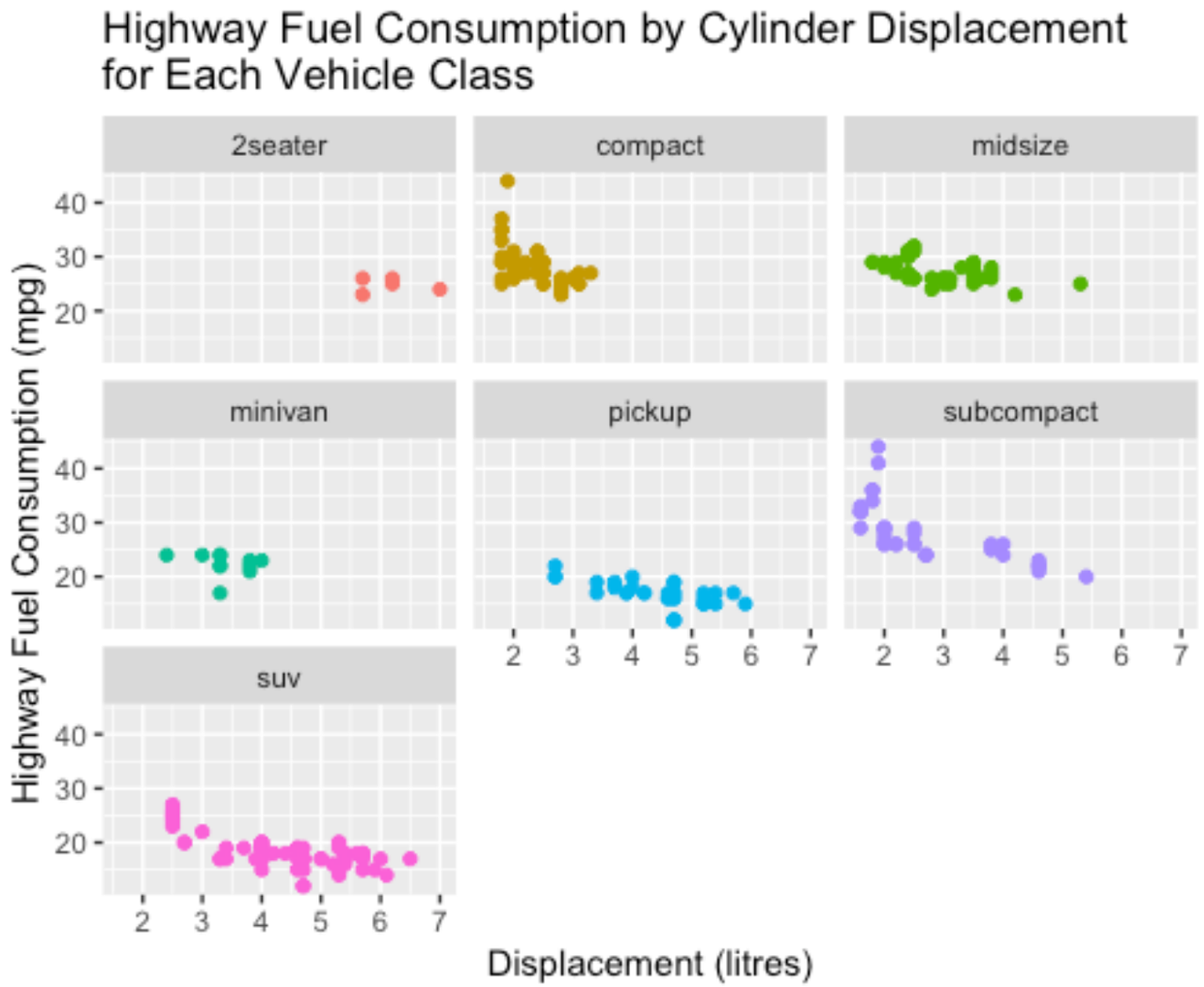
```
> length(unique(mpg$manufacturer))
[1] 15
```

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

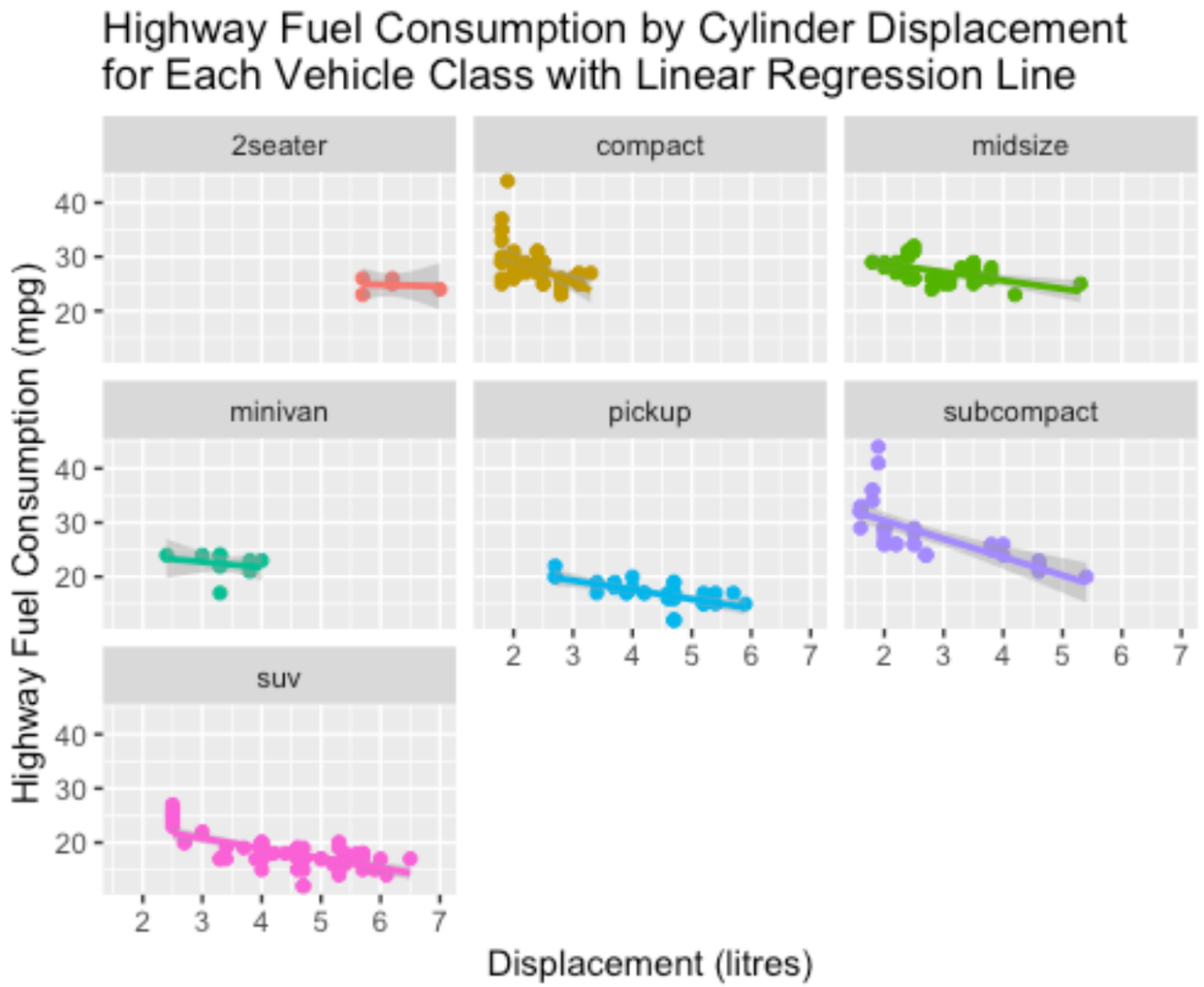
This illustrates the idea that there is not one 'correct' way to visualise the data, but rather that your choice of visualisation will be influenced by the question you're investigating, or the story you're wanting to tell…



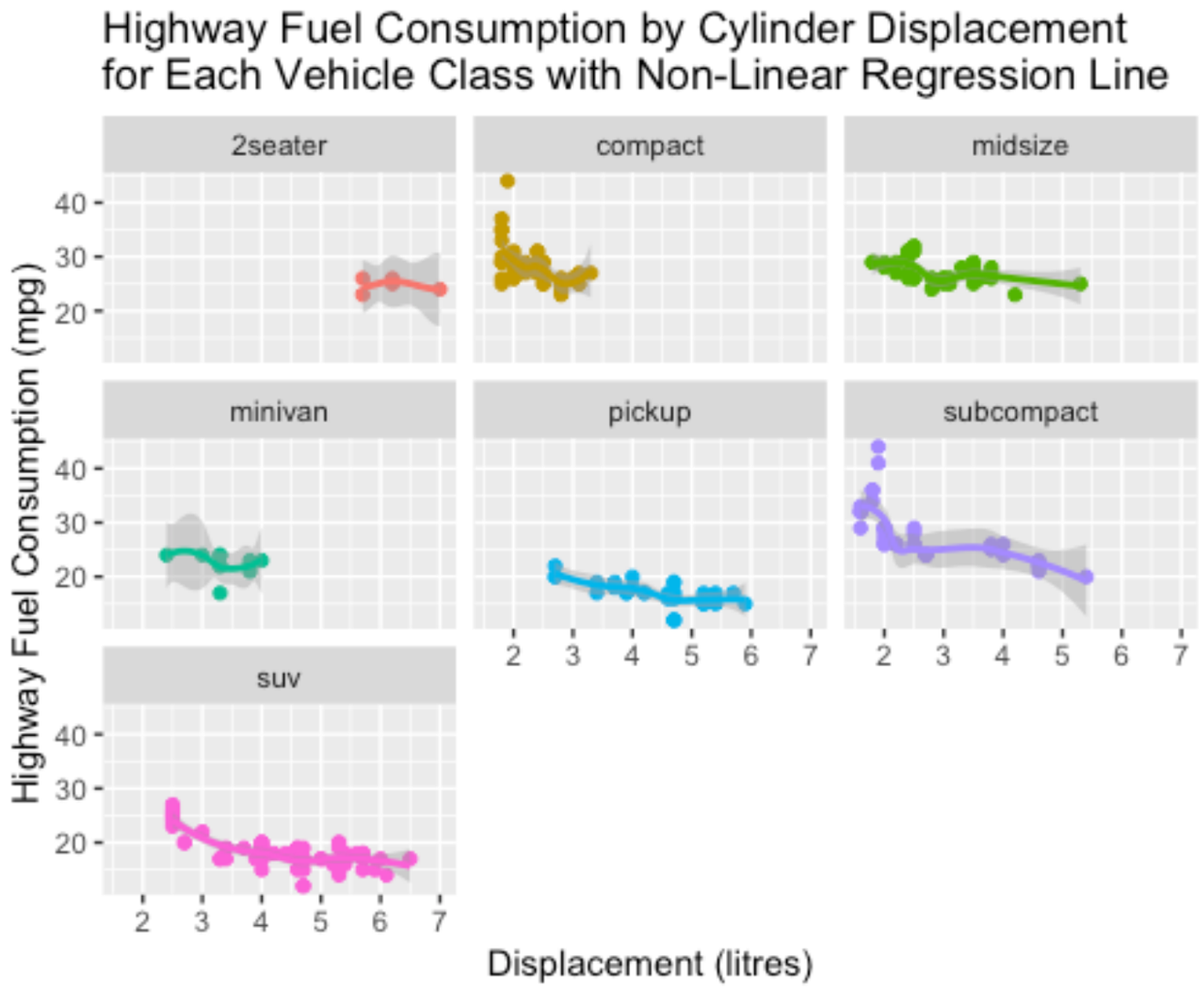

```
#build a violin plot with added descriptives
ggplot(mpg, aes(x = factor (cyl), y = cty, fill = factor (cyl))) +
  geom_violin() +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary (fun.data = mean_cl_boot, colour = "black", size = .5) +
  xlab("Number of Cylinders") +
  ylab("City Fuel Consumption (mpg)") +
  ggtitle ("City Fuel Consumption by Number of Cylinders")
```




```
#facet wrap by vehicle class with displacement instead of cylinder number
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_point() +
  facet_wrap(~class) +
  guides(colour = FALSE) +
  xlab("Displacement (litres)") +
  ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Highway Fuel Consumption by Cylinder Displacement \nfor Each Vehicle Class")
```

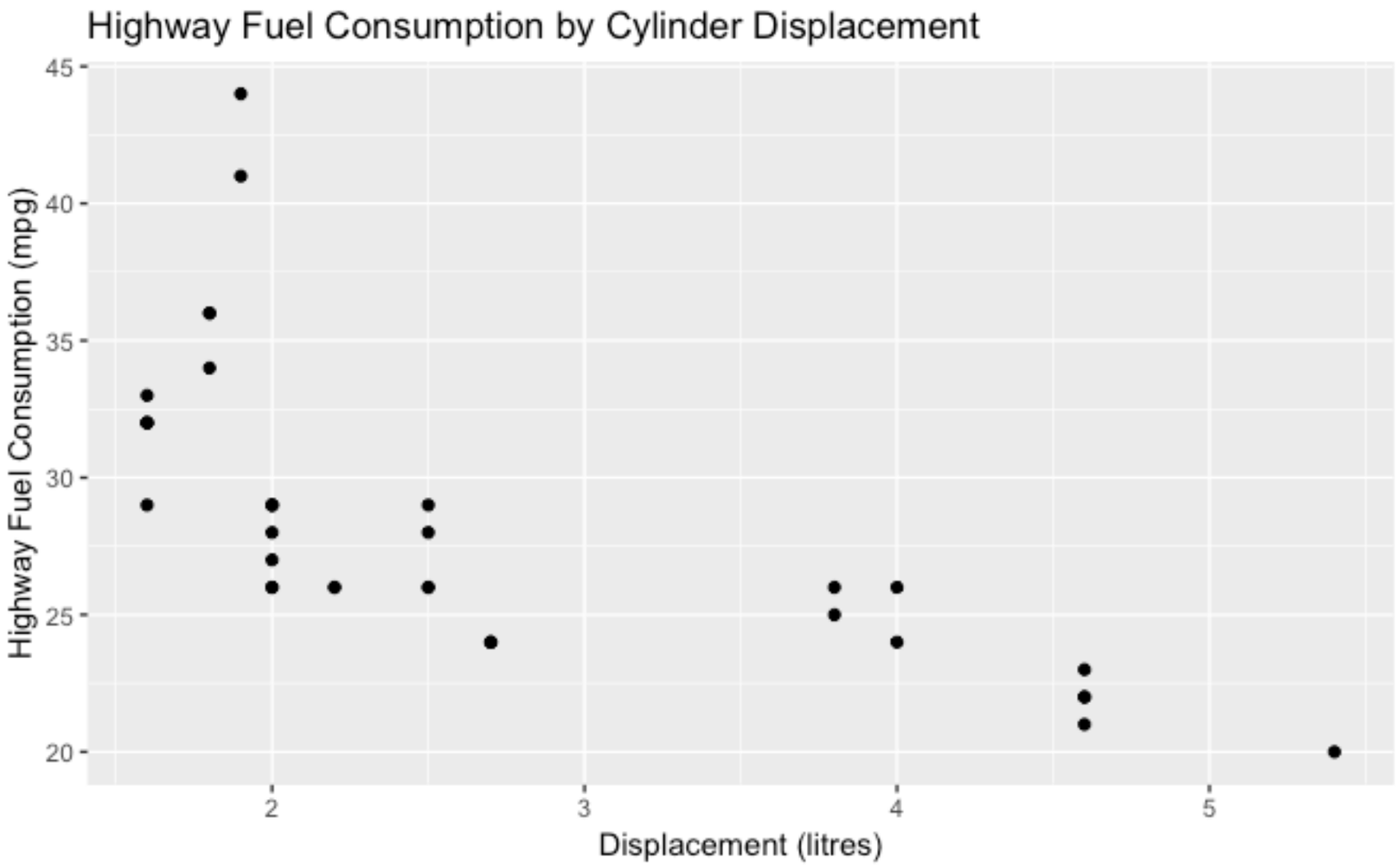



```
#now add a linear function to each
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_point() +
  facet_wrap(~ class) +
  guides(colour = FALSE) +
  geom_smooth(method = "lm") +
  xlab("Displacement (litres)") +
  ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Highway Fuel Consumption by Cylinder Displacement \nfor Each Vehicle Class with
  Linear Regression Line")
```



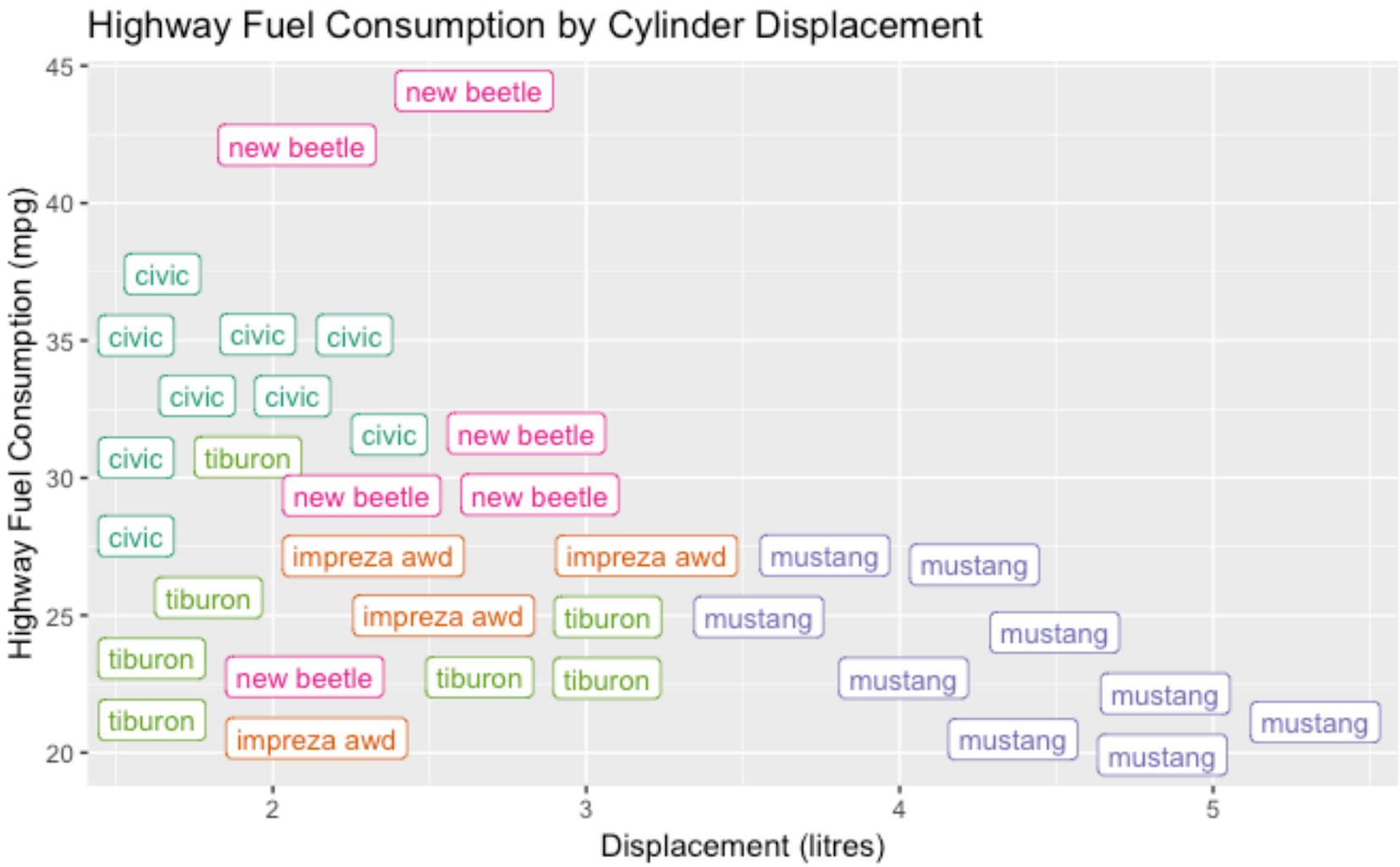

```
#now with a non-linear function to each
ggplot(mpg, aes(displ, hwy, colour = class)) +
    geom_point() +
    facet_wrap(~ class) +
    guides(colour = FALSE) +
    geom_smooth(method = "loess") +
    xlab("Displacement (litres)") +
    ylab("Highway Fuel Consumption (mpg)") +
    ggtitle ("Highway Fuel Consumption by Cylinder Displacement \nfor Each Vehicle Class with
    Non-Linear Regression Line")
```

Scatterplot of subcompact cars

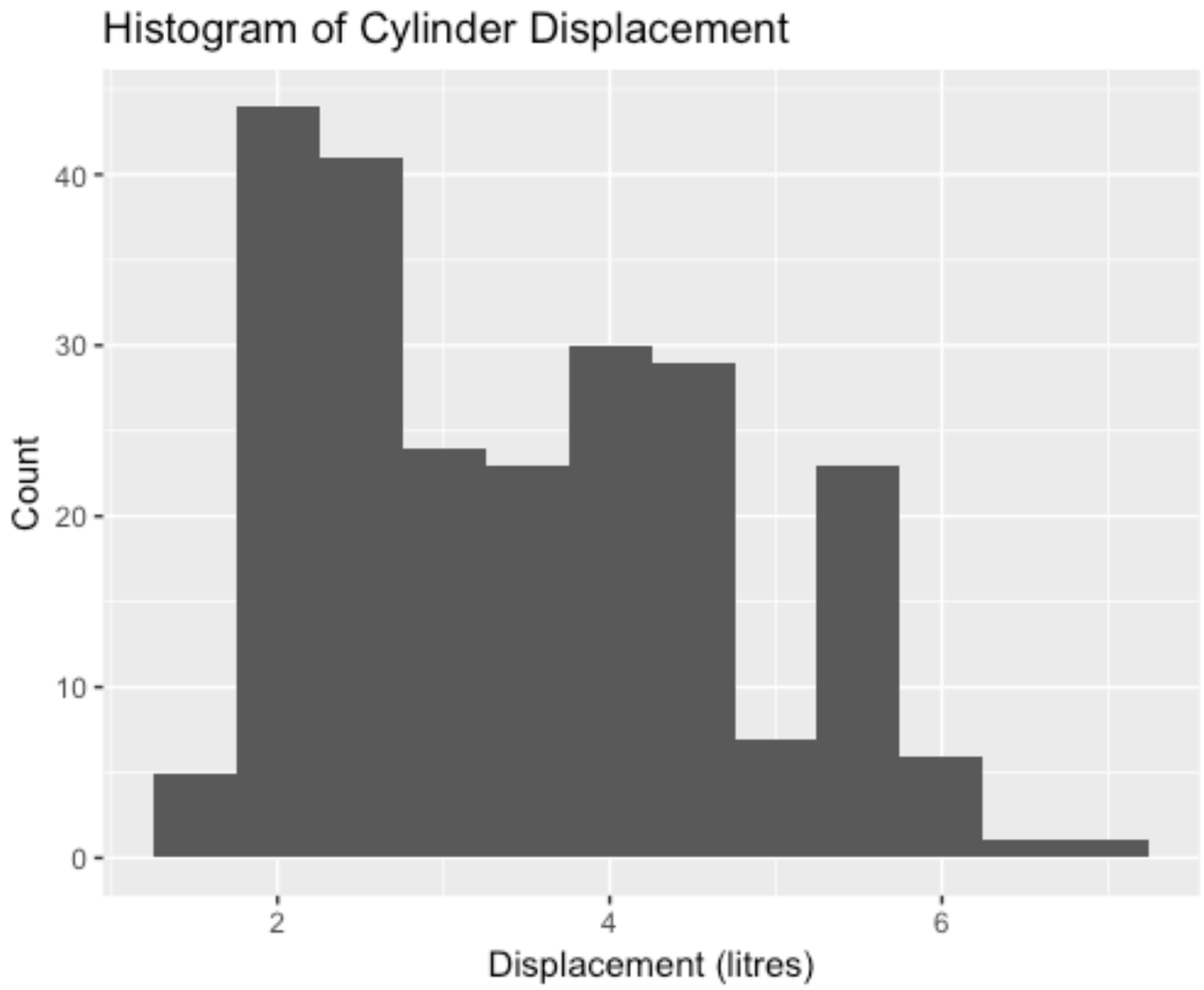



```
ggplot(filter(mpg, class == "subcompact"), aes(x = displ, y = hwy, label = model))
  geom_point() +
  guides(colour = FALSE) +
  xlab("Displacement (litres)") +
  ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Highway Fuel Consumption by Cylinder Displacement")
```

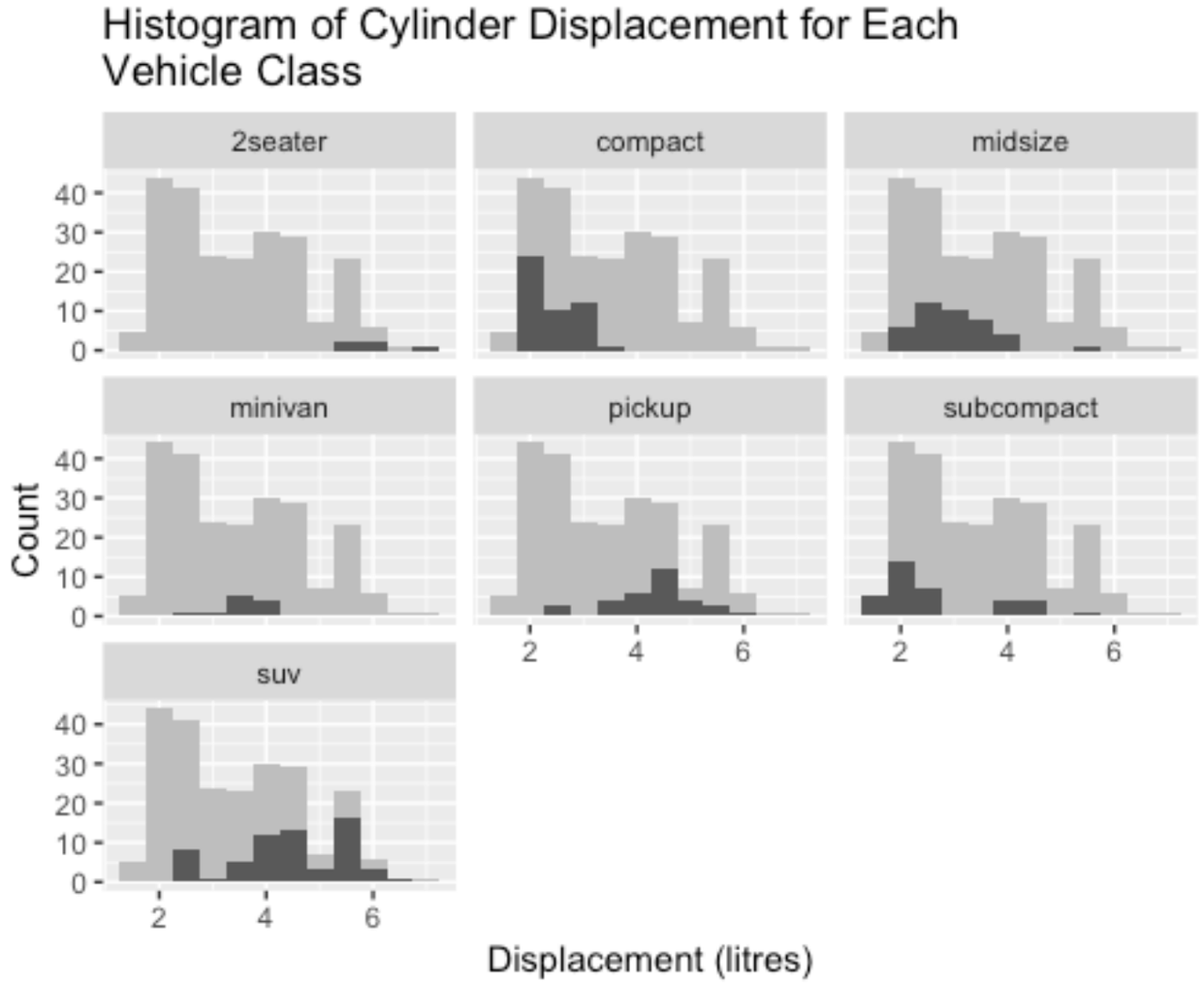
Replacing each point with the label of that point



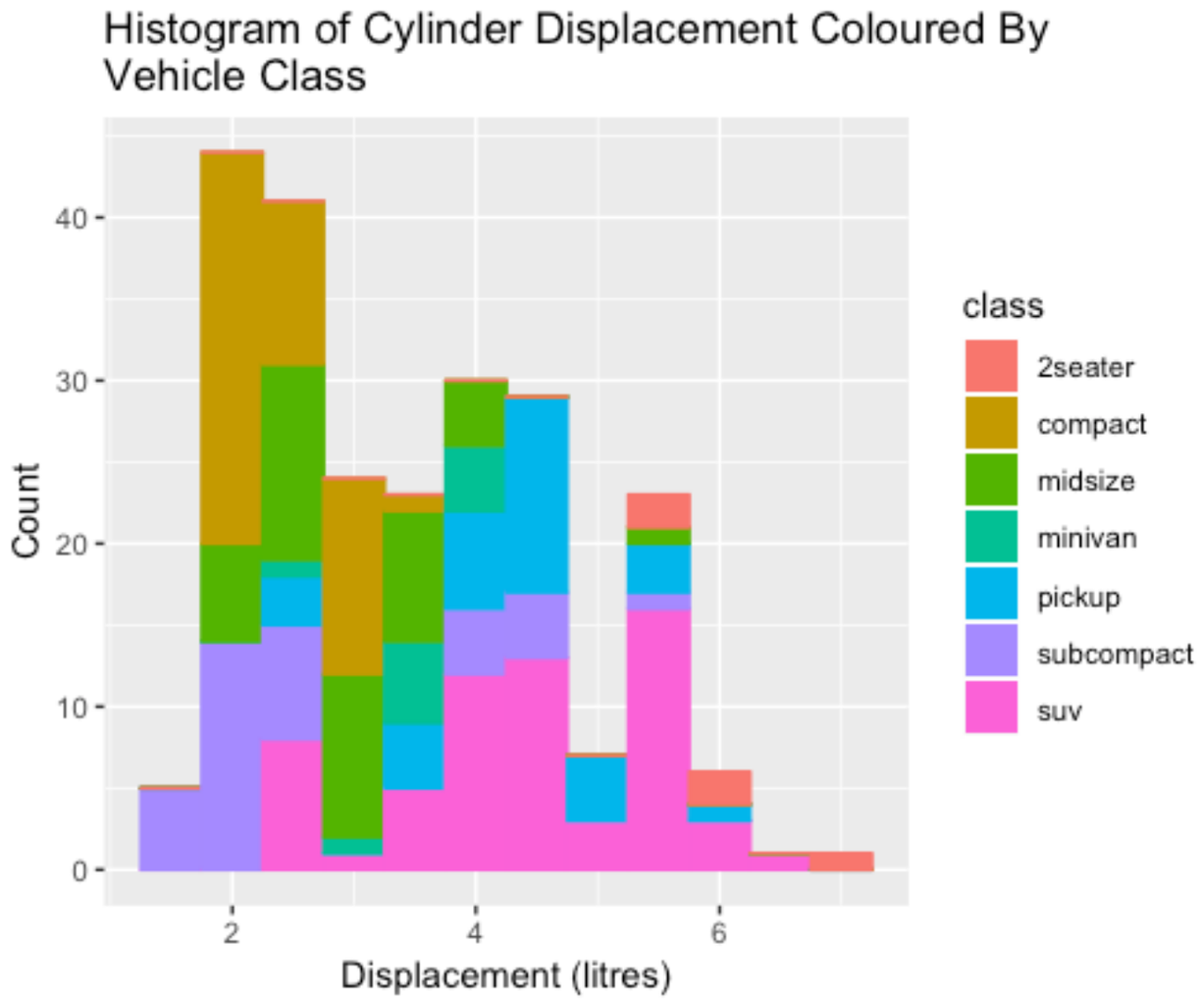

```
ggplot(filter(mpg, class == "subcompact"), aes(x = displ, y = hwy, colour = model,
                                               label = model)) +
  scale_colour_brewer(palette = "Dark2") +
  geom_label_repel(label.padding = 0.25, label.size = .25,
                   min.segment.length = 100, size = 3.5) +
  guides(colour = FALSE) +
  xlab("Displacement (litres)") +
  ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Highway Fuel Consumption by Cylinder Displacement")
```



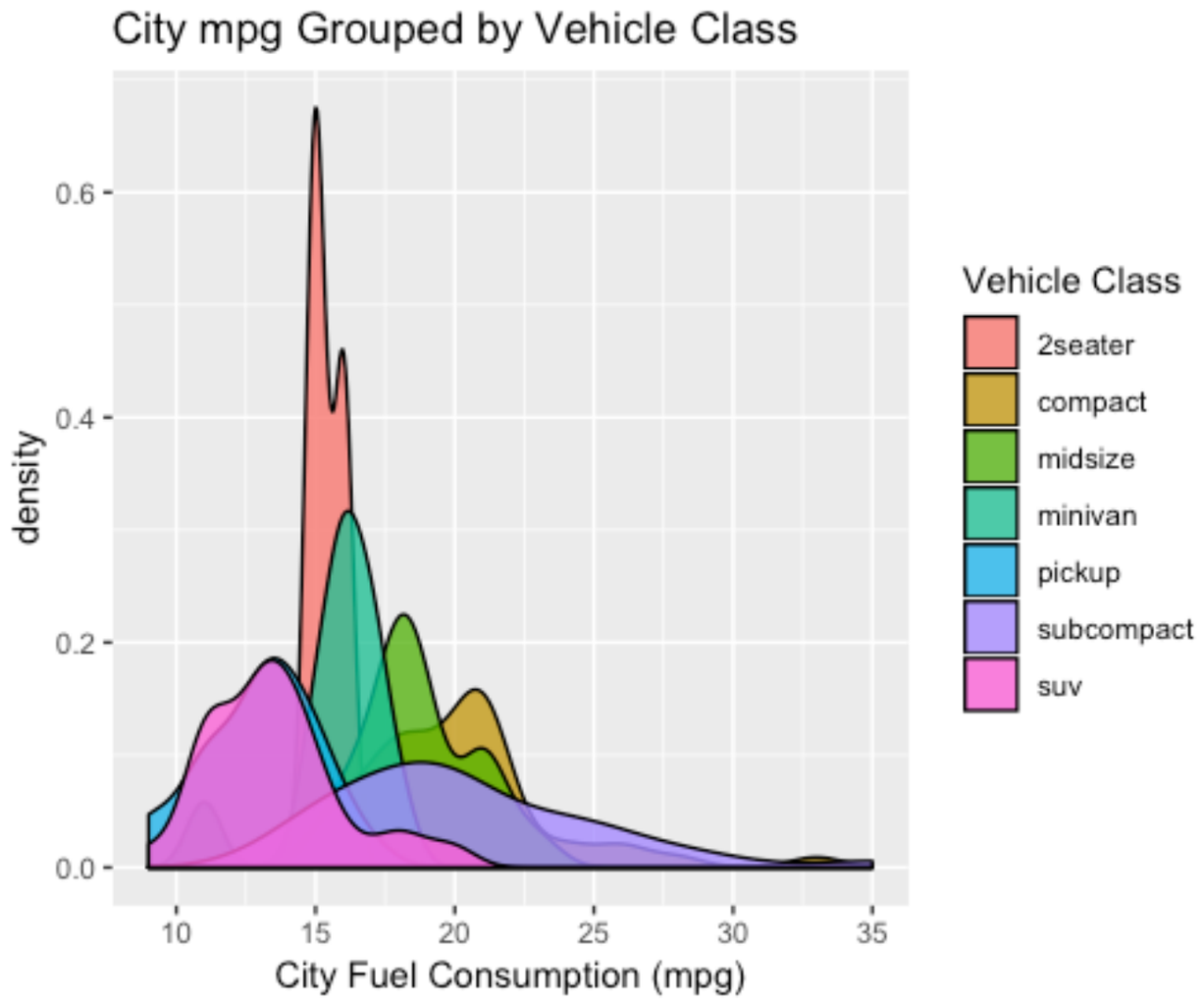

```
#plot basic histogram
ggplot(mpg, aes(displ)) +
  geom_histogram(binwidth = .5) +
  guides(colour = FALSE) +
  xlab("Displacement (litres)") + ylab("Count") +
  ggtitle ("Histogram of Cylinder Displacement")
```



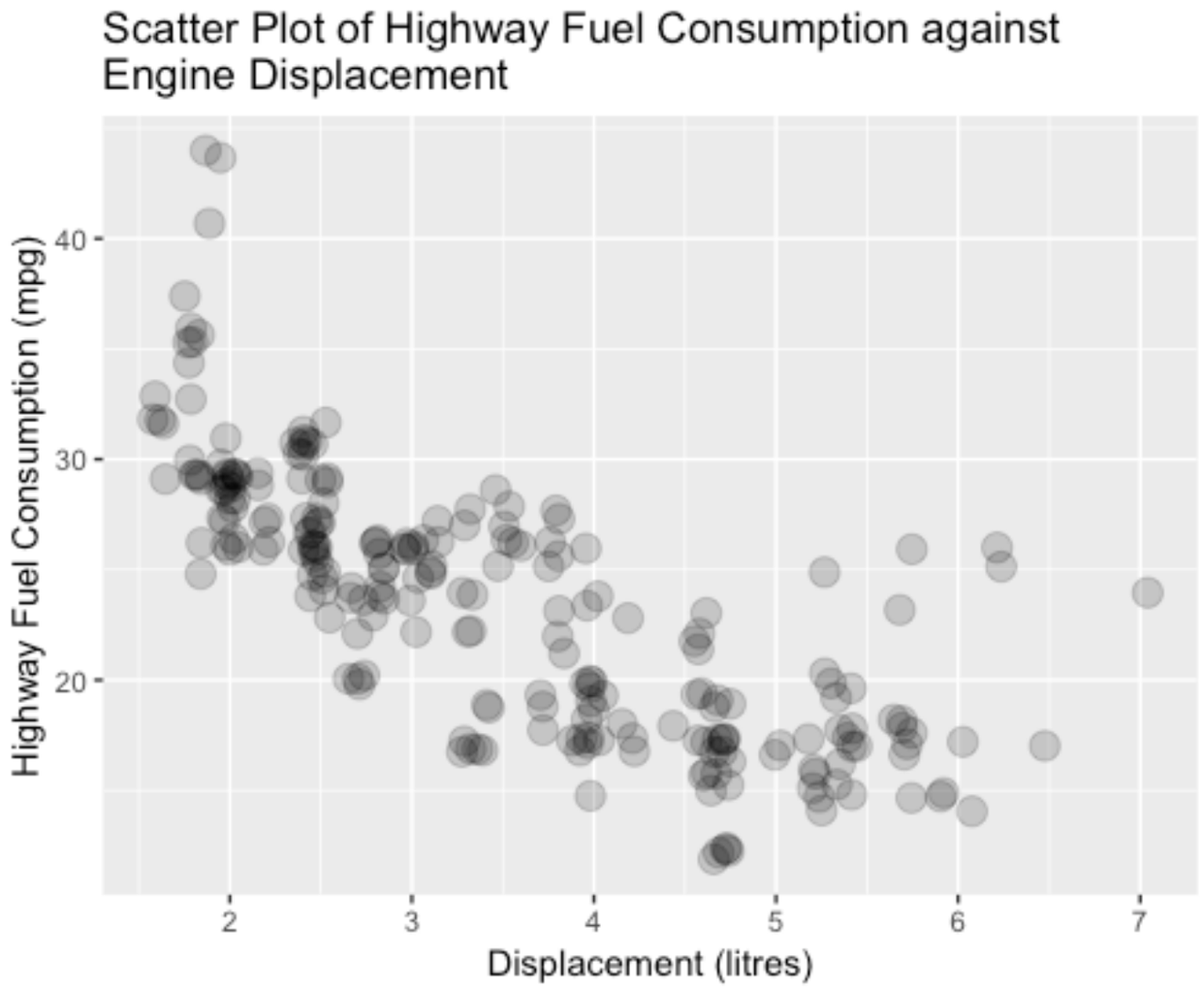

```
#facet by class and include background histogram of all data in each facet
mpg1 <- select(mpg, -class)
ggplot(mpg, aes(x = displ)) +
  geom_histogram(data = mpg1, fill = "grey", binwidth = .5) +
  geom_histogram(binwidth = .5) +
  guides(colour = FALSE) +
  facet_wrap (~ class) +
  xlab("Displacement (litres)") +
  ylab("Count") +
  ggtitle ("Histogram of Cylinder Displacement for Each \nVehicle Class")
```



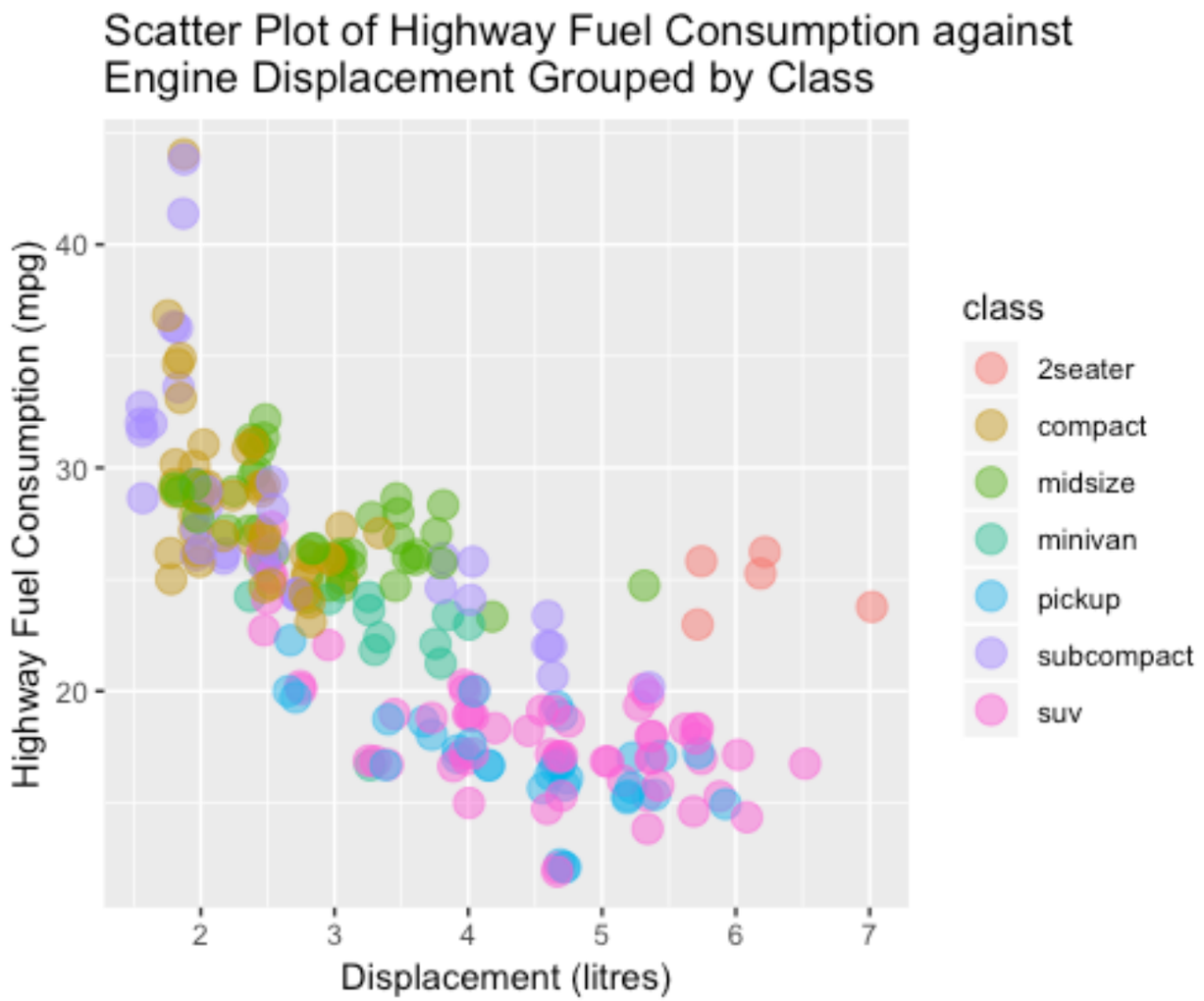

```
#plot histogram of displacement, coloured by class
ggplot(mpg, aes(x = displ, colour = class, fill=class)) +
  geom_histogram(binwidth = .5) +
  guides(colour = FALSE) +
  xlab("Displacement (litres)") +
  ylab("Count") +
  ggtitle ("Histogram of Cylinder Displacement Coloured By \nVehicle Class")
```



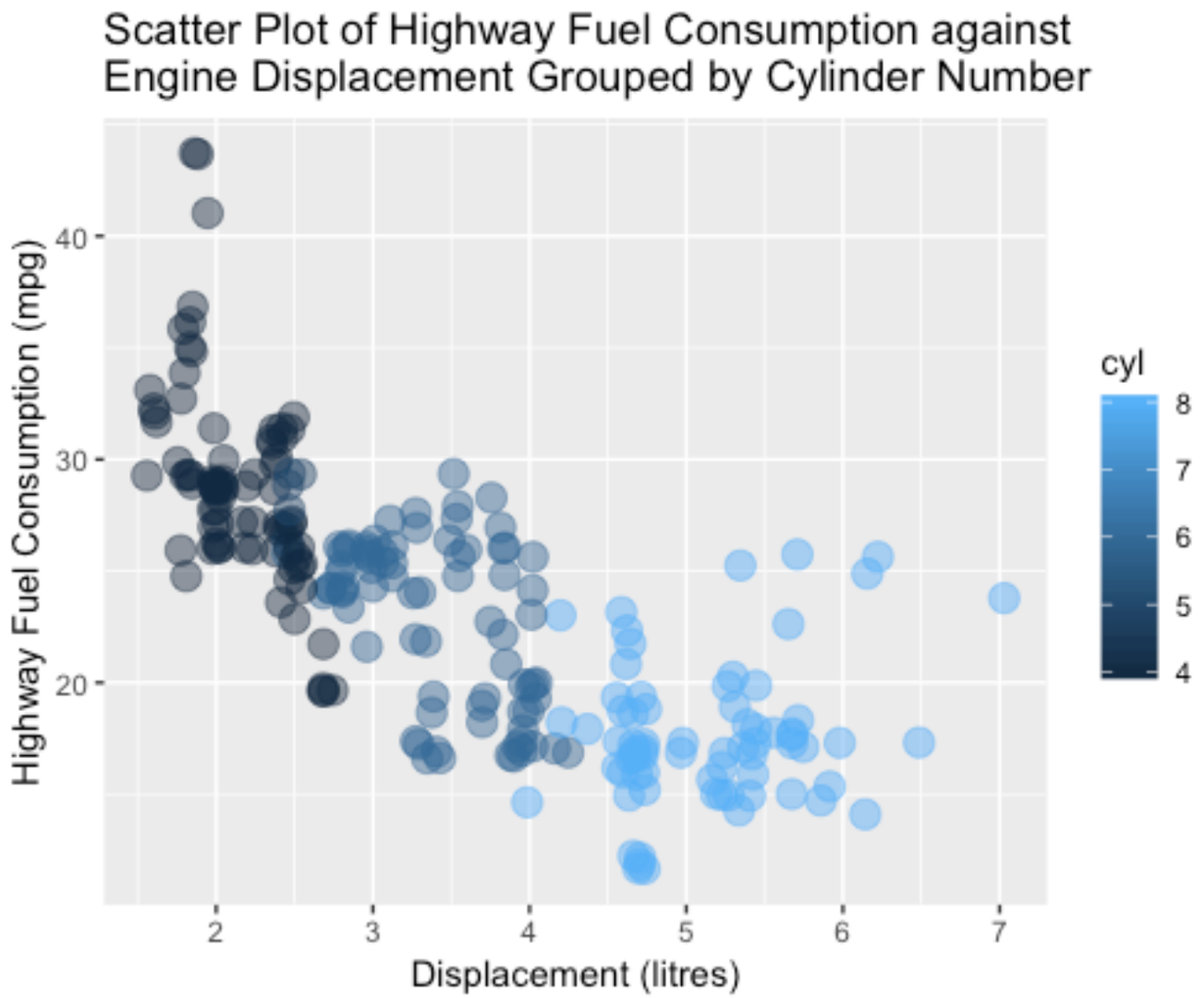

```
ggplot(mpg, aes(x = cty)) +
  geom_density(aes(fill = factor(class)), alpha = 0.8) +
  labs(title = "City mpg Grouped by Vehicle Class”, x = "City Fuel Consumption (mpg)",
       fill = "Vehicle Class")
```



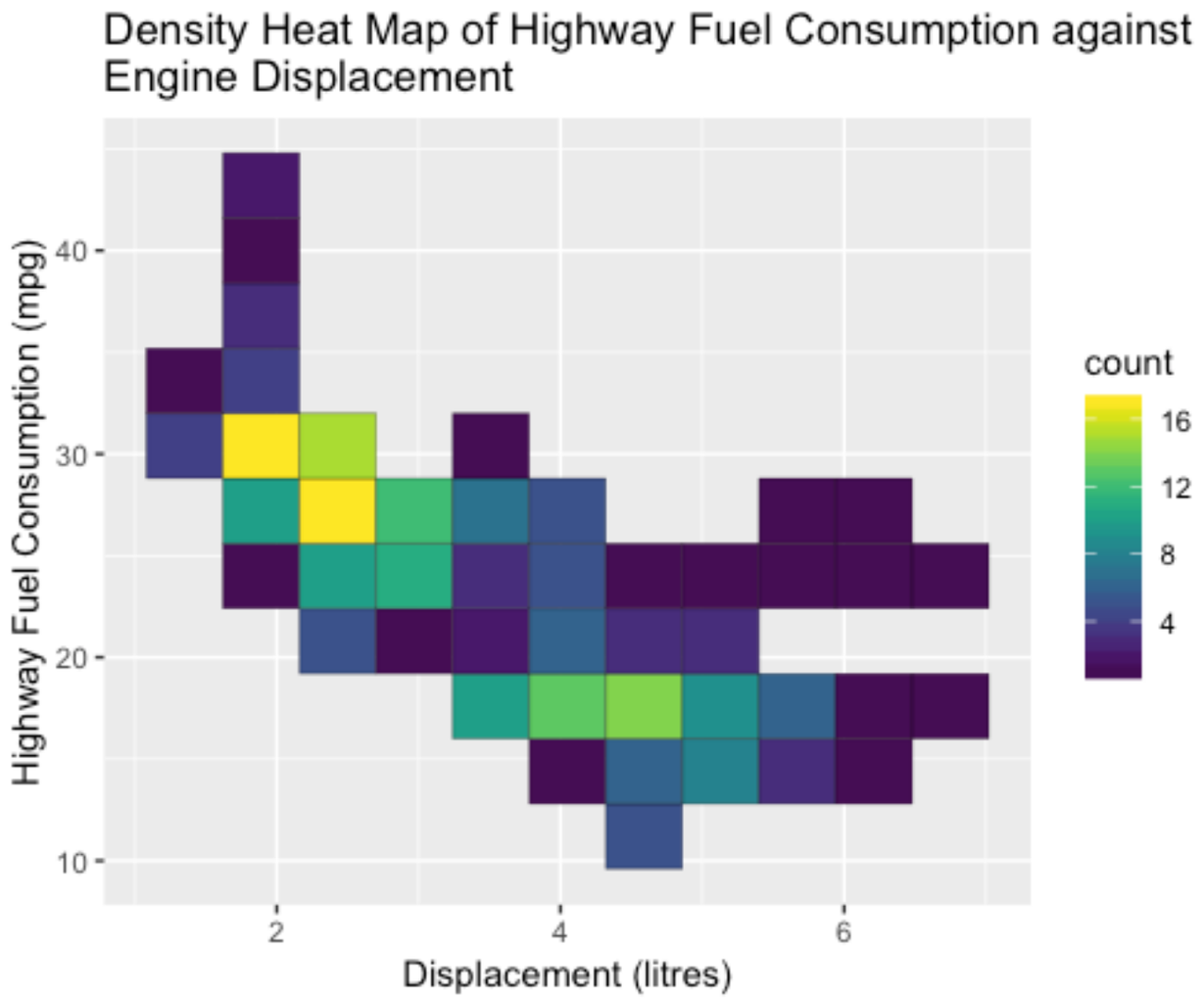

```
#scatterplot with jitter
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_jitter(width = 0.05, alpha = .2, size = 4) +
  xlab("Displacement (litres)") +
  ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Scatter Plot of Highway Fuel Consumption against \nEngine Displacement")
```




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




```
ggplot(mpg, aes(x = displ, y = hwy, colour = cyl)) +
  geom_jitter(width = 0.05, alpha = .5, size = 4) +
  xlab("Displacement (litres)") +
  ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Scatter Plot of Highway Fuel Consumption against \nEngine Displacement
  Grouped by Cylinder Number")
```




```
#2d histogram with density heatmap
ggplot(mpg, aes(x = displ, y = hwy)) +
  stat_bin2d(bins = 10, colour = "black") +
  scale_fill_viridis() +
  xlab("Displacement (litres)") +
  ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Density Heat Map of Highway Fuel Consumption against \nEngine Displacement")
```

Plotting Time Series Data

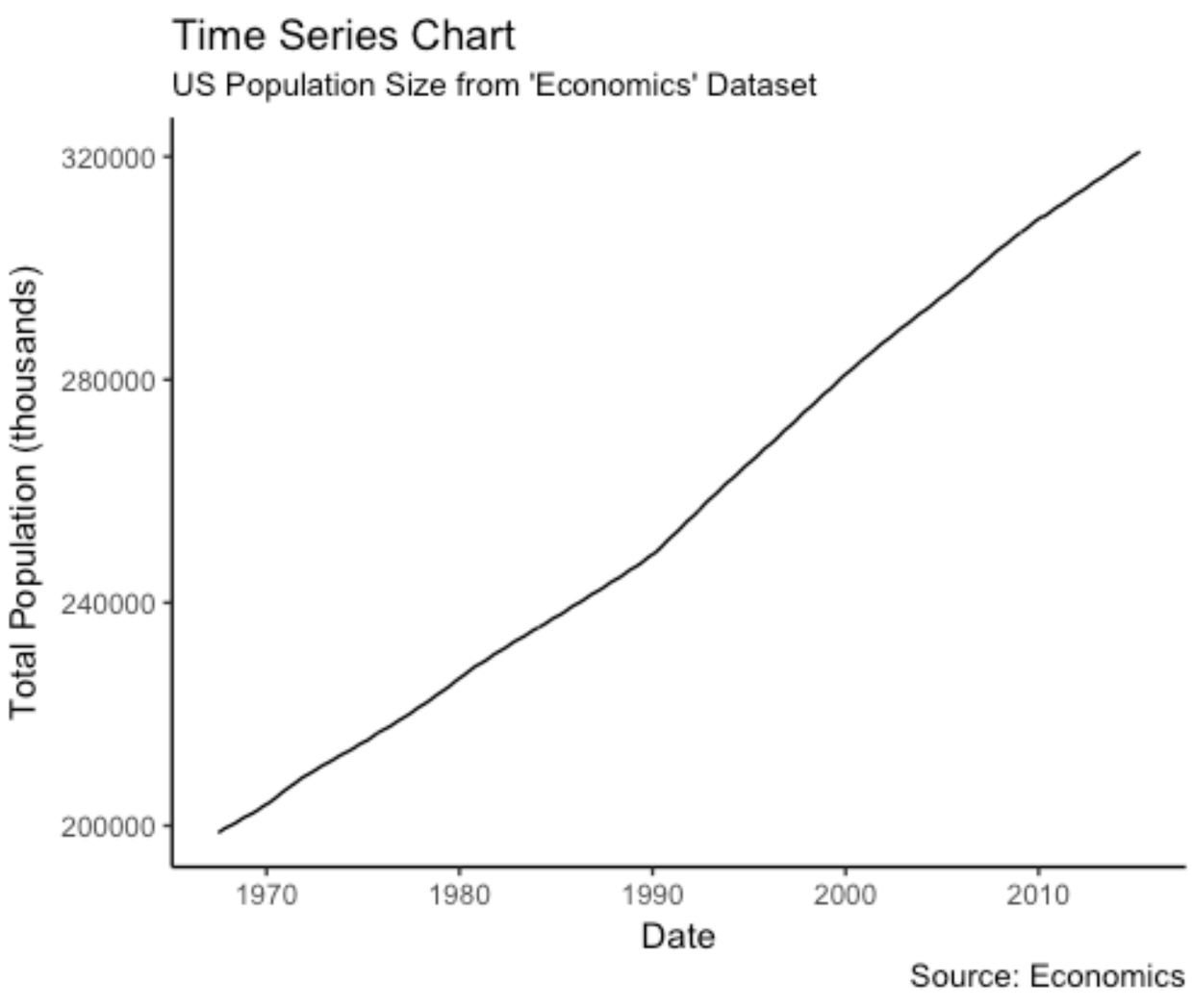
We're going to use the in-built dataset "Economics". This contains monthly data corresponding to US population size, personal savings rate, unemployment numbers (and much more)…

```
> str(economics)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':   574 obs. of  6 variables:
 $ date    : Date, format: "1967-07-01" "1967-08-01" "1967-09-01" ...
 $ pce     : num  507 510 516 513 518 ...
 $ pop     : int  198712 198911 199113 199311 199498 199657 199808 199920 200056
200208 ...
 $ psavert : num  12.5 12.5 11.7 12.5 12.5 12.1 11.7 12.2 11.6 12.2 ...
 $ uempmed : num  4.5 4.7 4.6 4.9 4.7 4.8 5.1 4.5 4.1 4.6 ...
 $ unemploy: int  2944 2945 2958 3143 3066 3018 2878 3001 2877 2709 ...
```

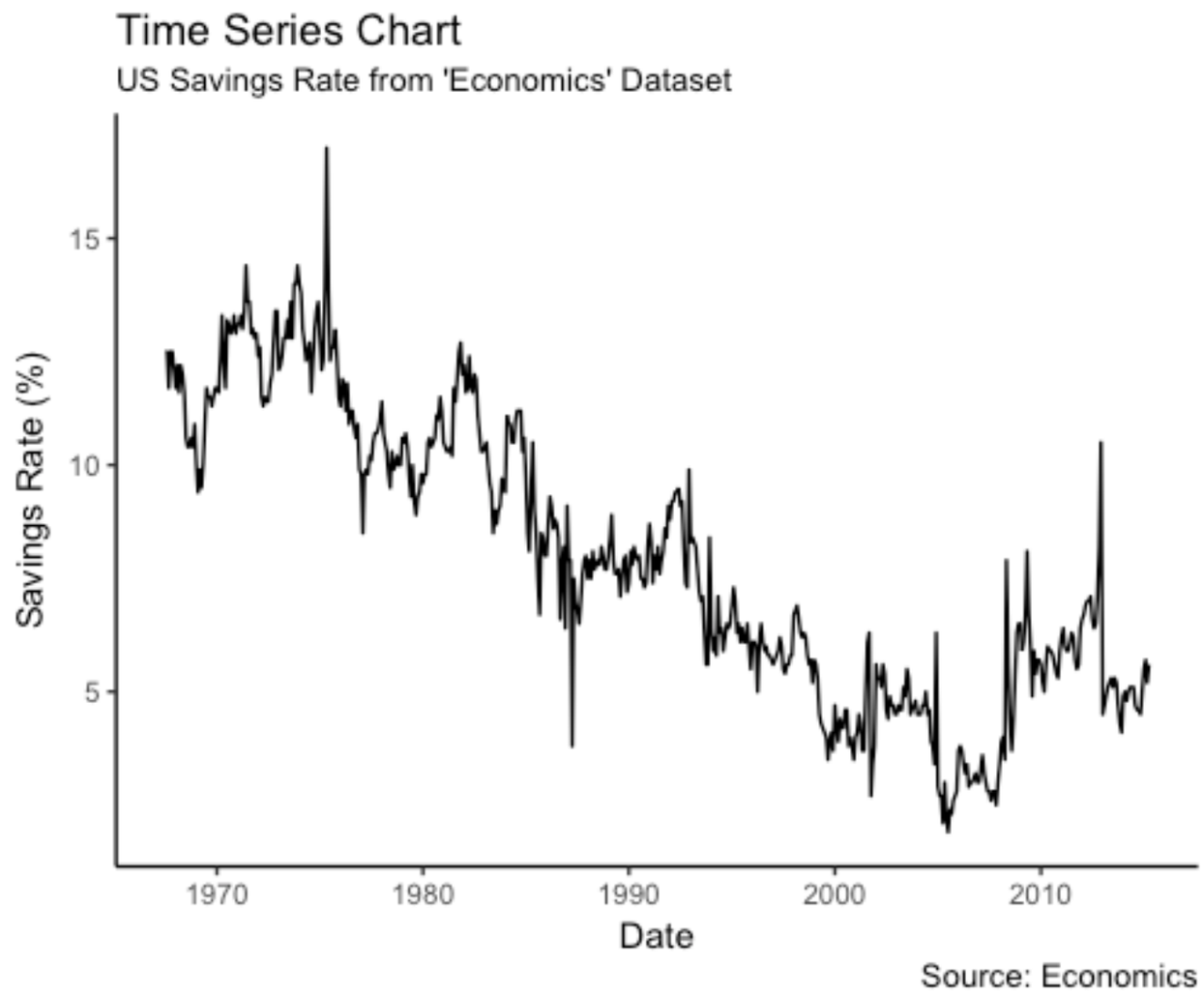
We're going to plot some time series graphs revealing population size, personal savings rate, and unemployment numbers over time.

Plotting Time Series Data

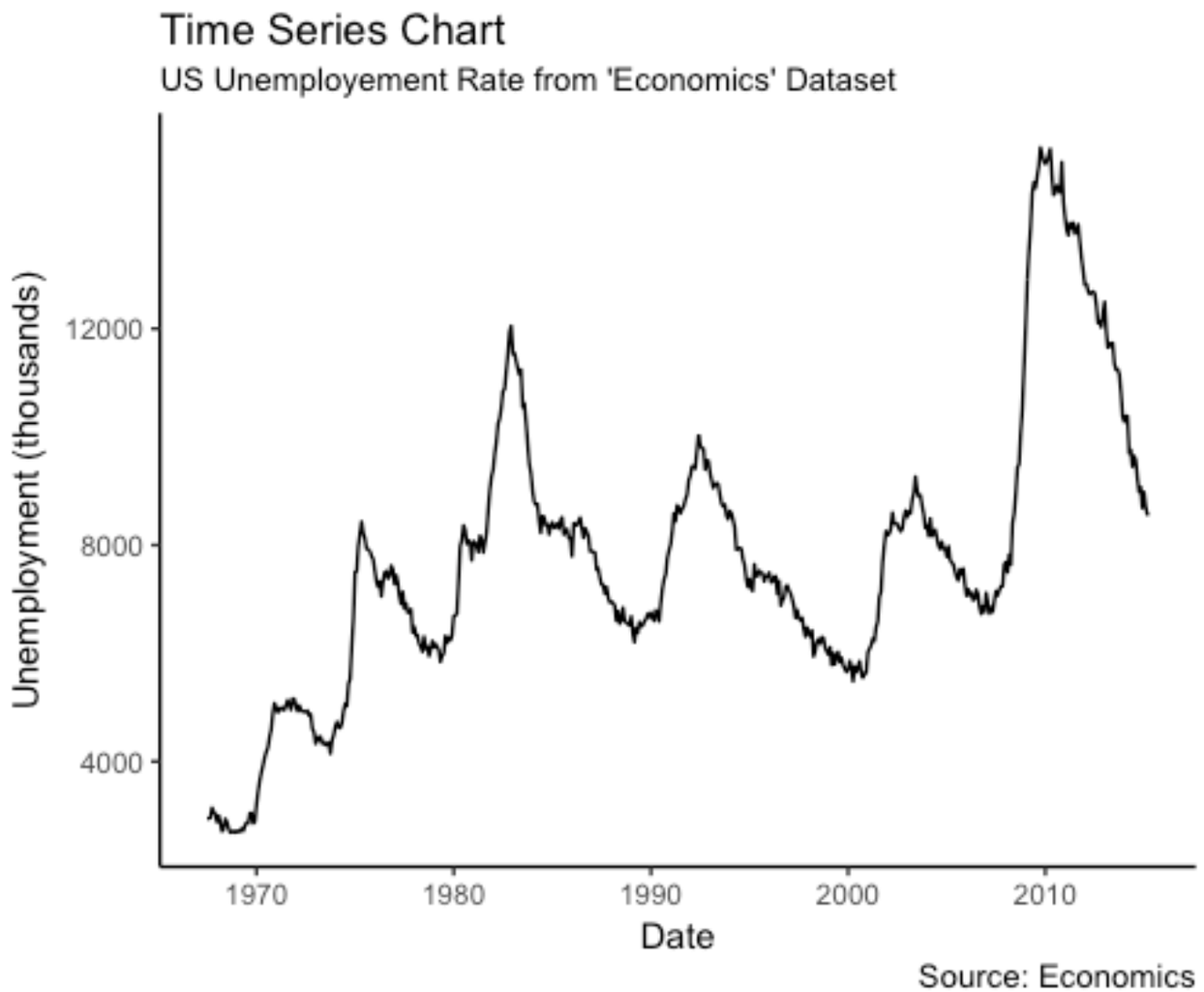
```
ggplot(economics, aes(x = date)) +
  geom_line(aes(y = pop)) +
  labs(title = "Time Series Chart",
       subtitle = "US Population
Size from 'Economics' Dataset",
       caption = "Source:
Economics",
       y = "Total Population
(thousands)", x = “Date")
```



```
ggplot(economics, aes(x = date))
+ geom_line(aes(y = psavert)) +
  labs(title = "Time Series
Chart",
       subtitle = "US Savings
Rate from 'Economics' Dataset",
       caption = "Source:
Economics",
       y = "Savings Rate (%)",
x="Date")
```

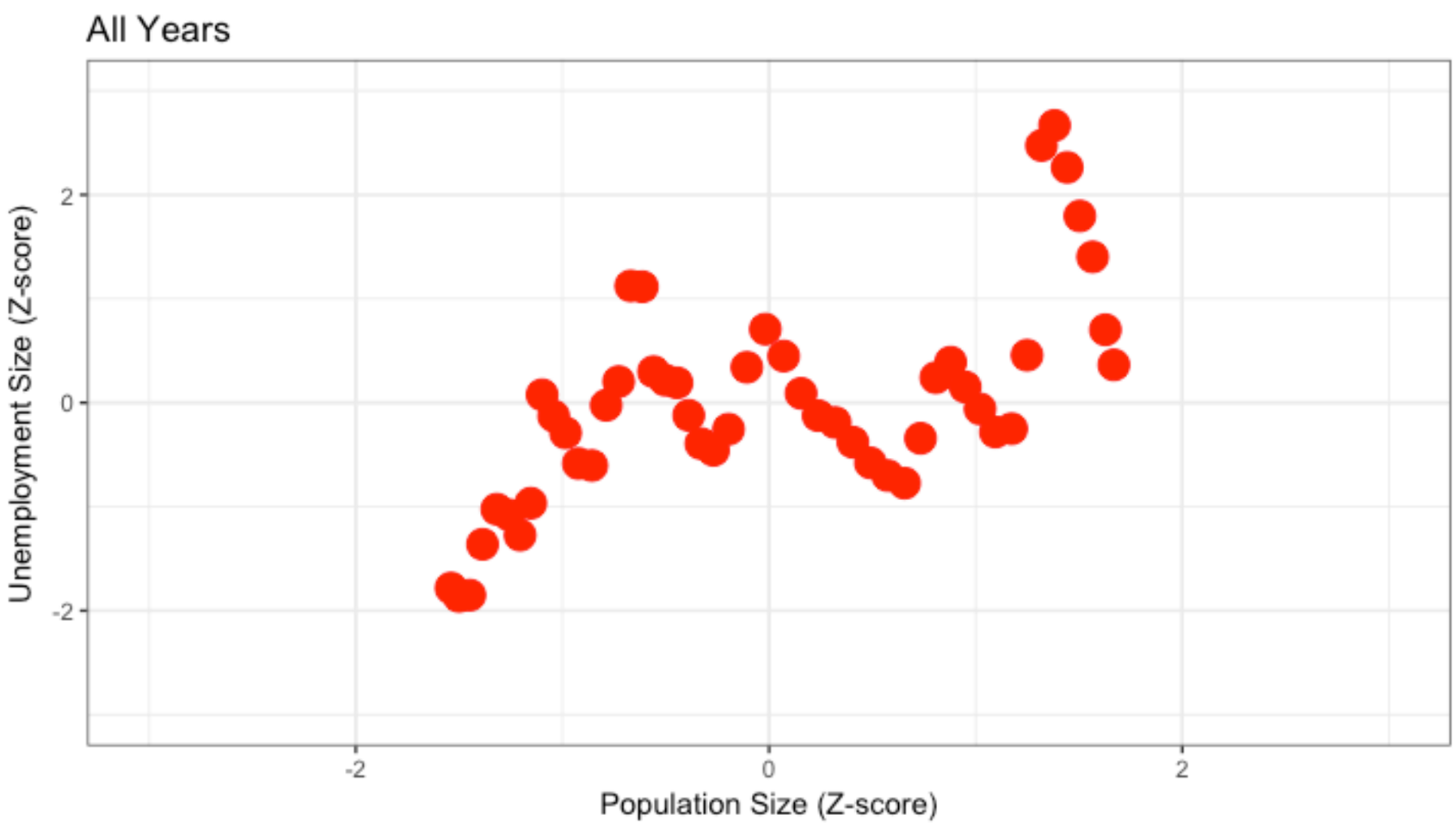


```
ggplot(economics, aes(x = date))
+ geom_line(aes(y = unemploy)) +
  labs(title = "Time Series
Chart",
       subtitle = "US
Unemployement Rate from
'Economics' Dataset",
       caption = "Source:
Economics",
       y = "Unemployment
(thousands)", x = "Date")
```



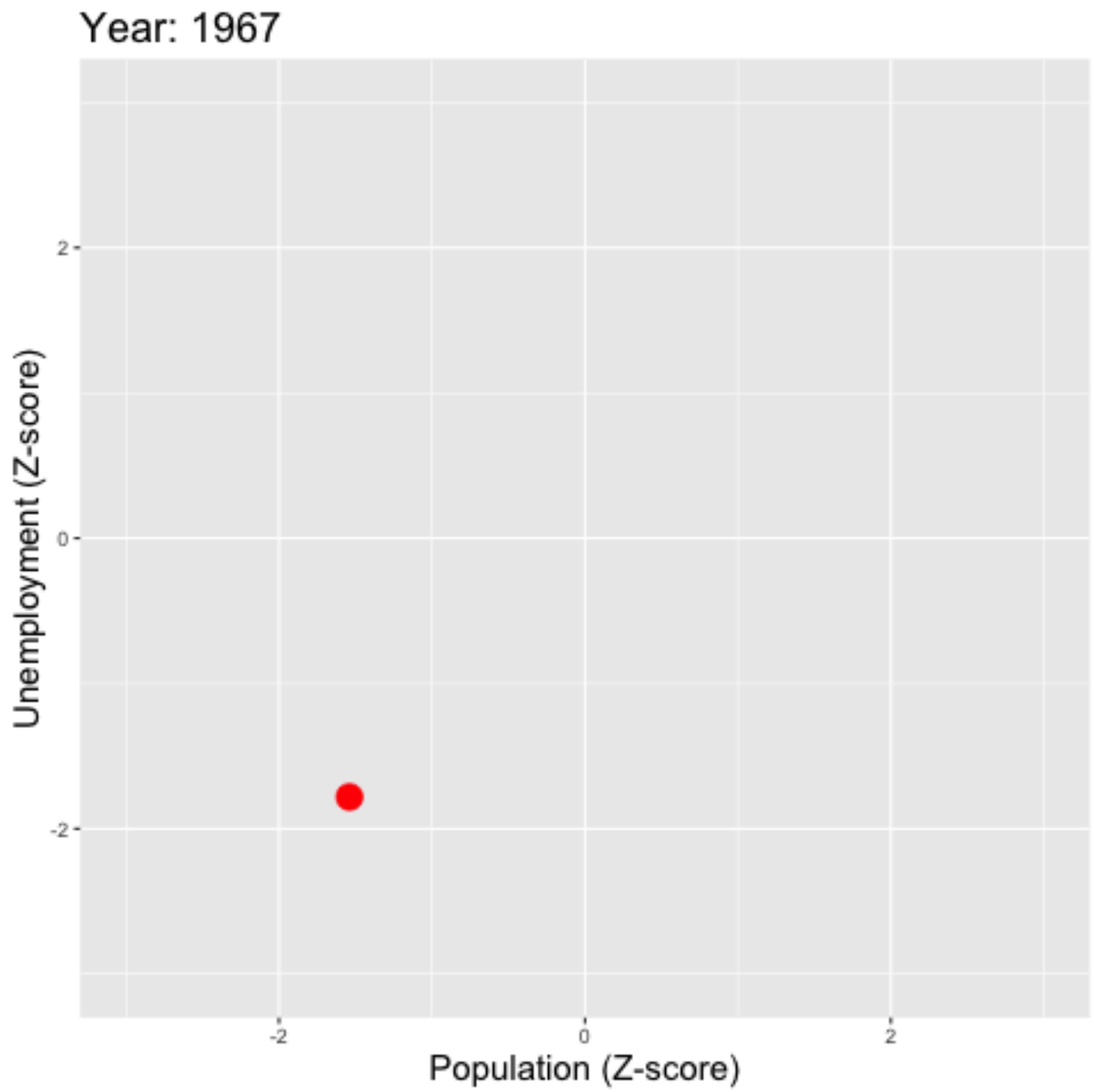
Animated Time Series Data

Now we're plotting Unemployment Size (transformed to Z-Scores) against Population Size (transformed to Z-scores) animated by Year.



Animated Time Series Data

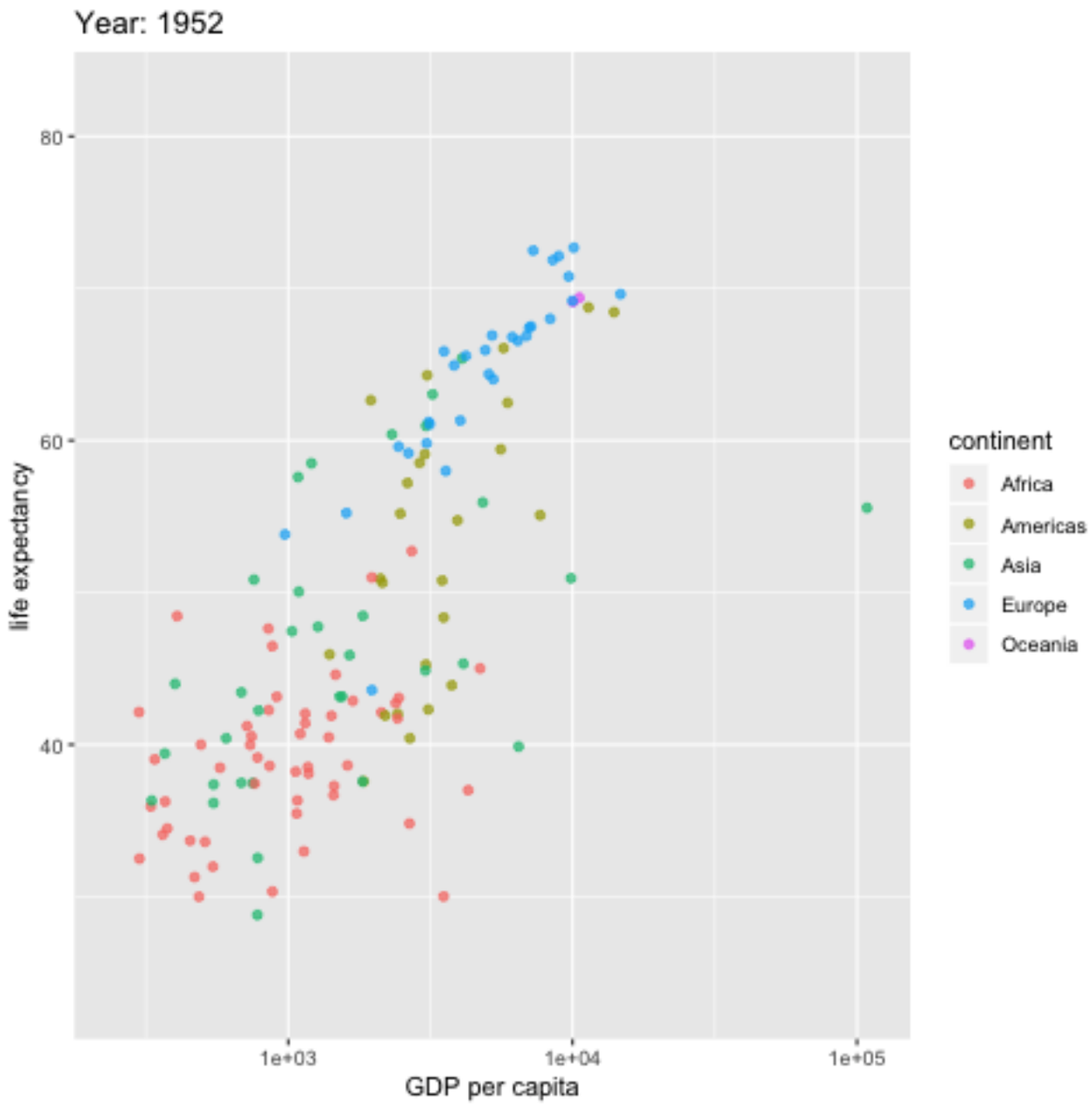
Now we're plotting Unemployment Size (transformed to Z-Scores) against Population Size (transformed to Z-scores) animated by Year.



Visualising Data with 4 Variables Simultaneously

Animated Time Series Data

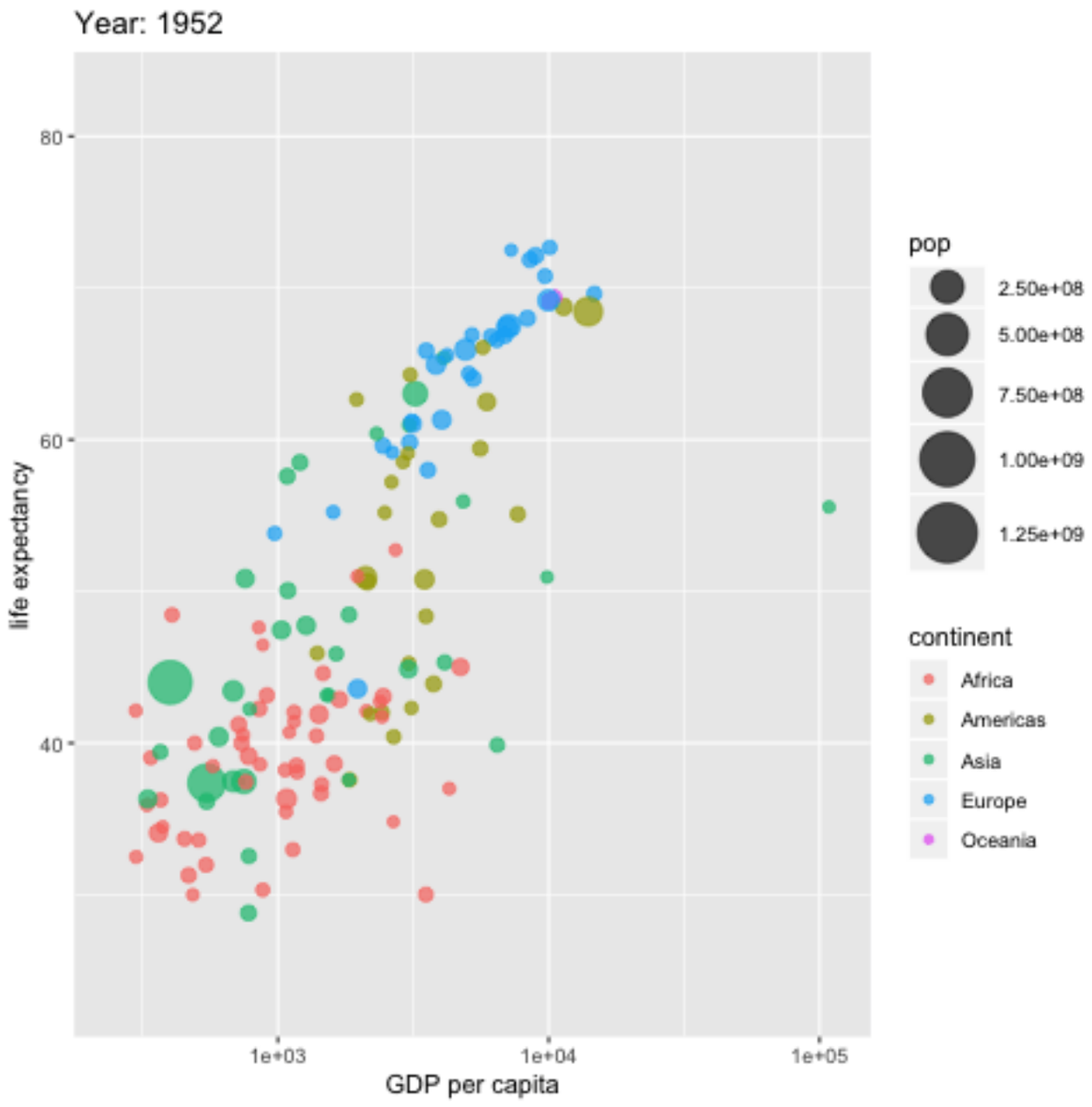
Life expectancy by GDP over time by Continent.



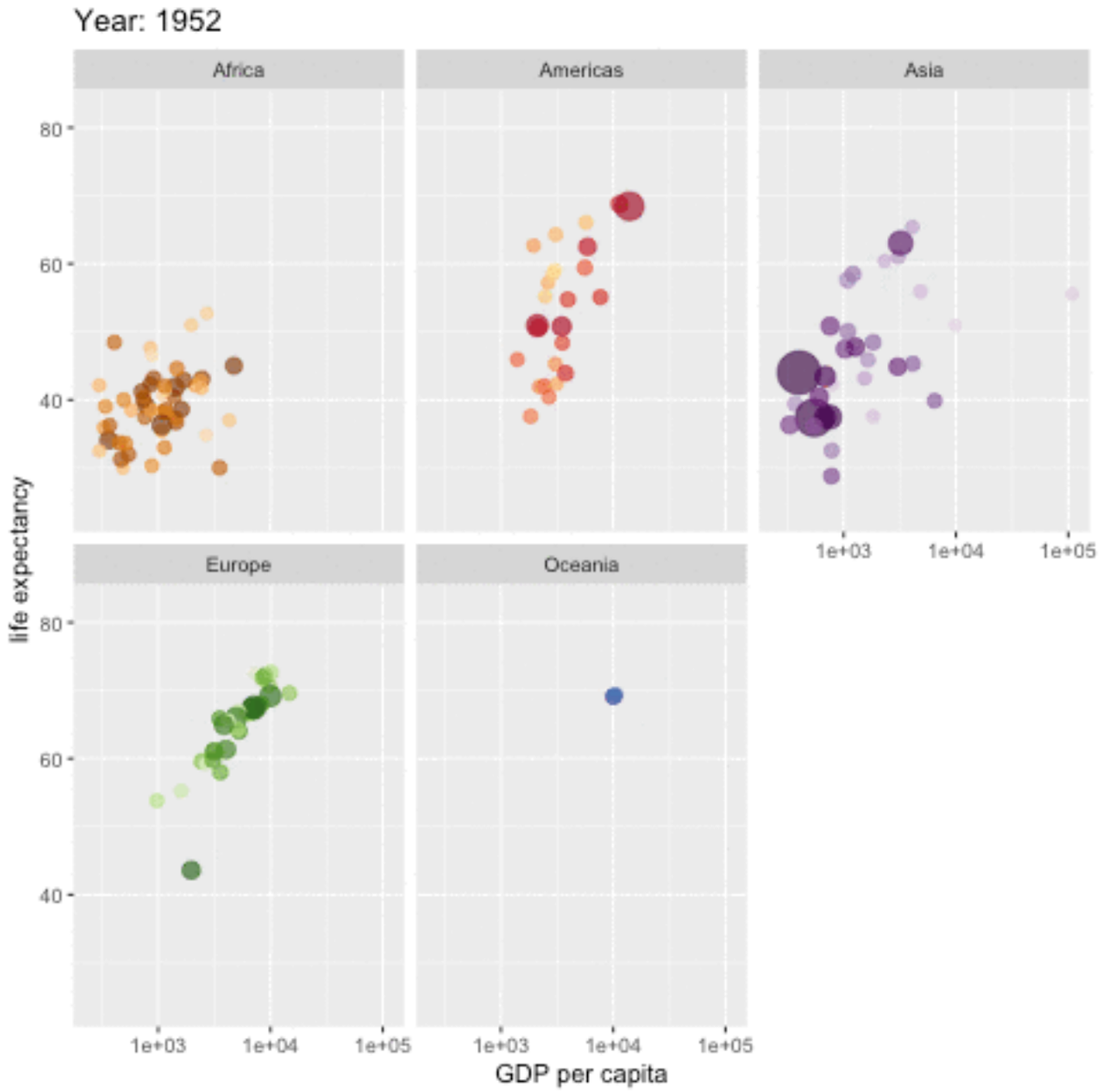
Visualising Data with 5 Variables Simultaneously

Animated Time Series Data

Now with a representation of population size.

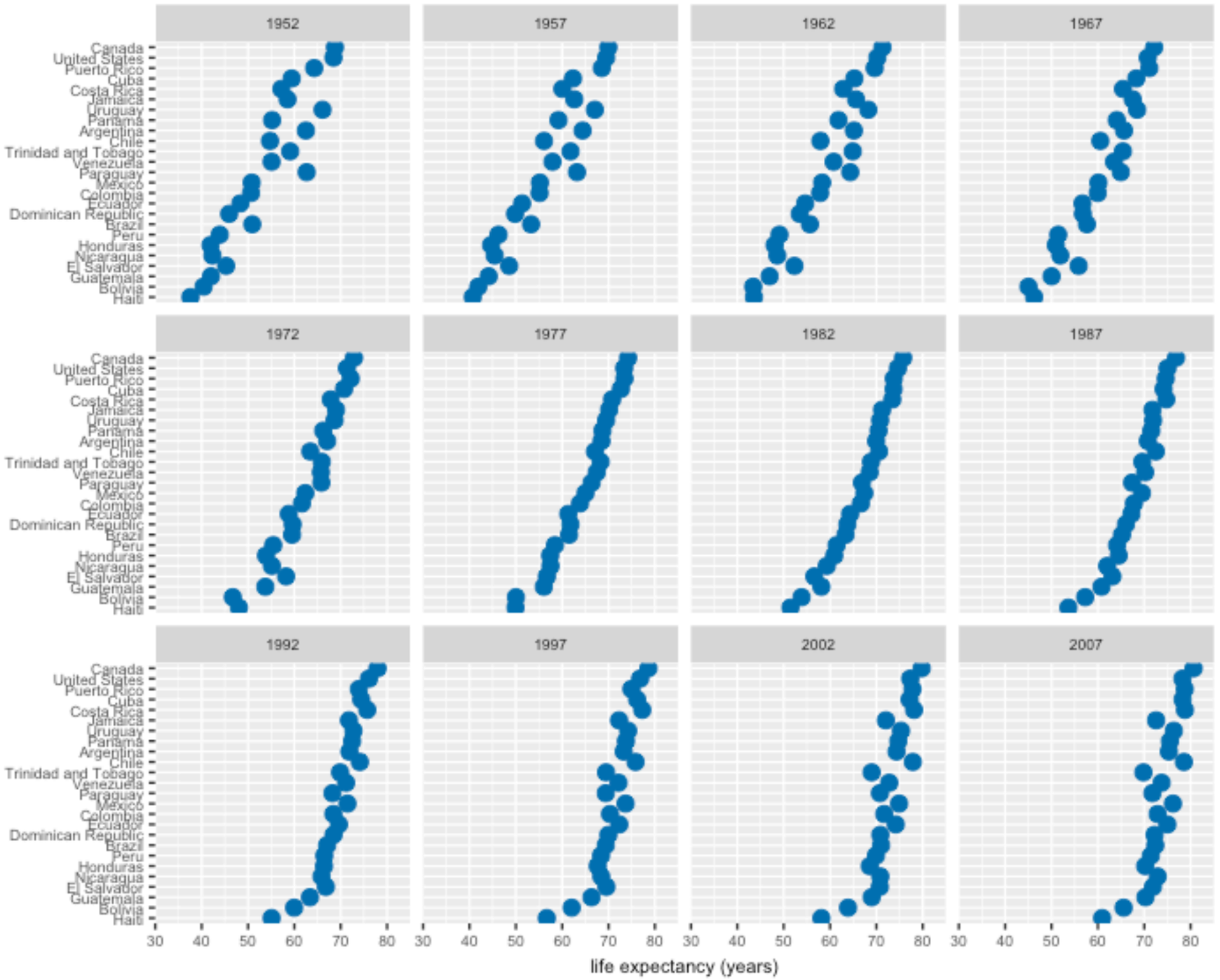


Separately by Continent

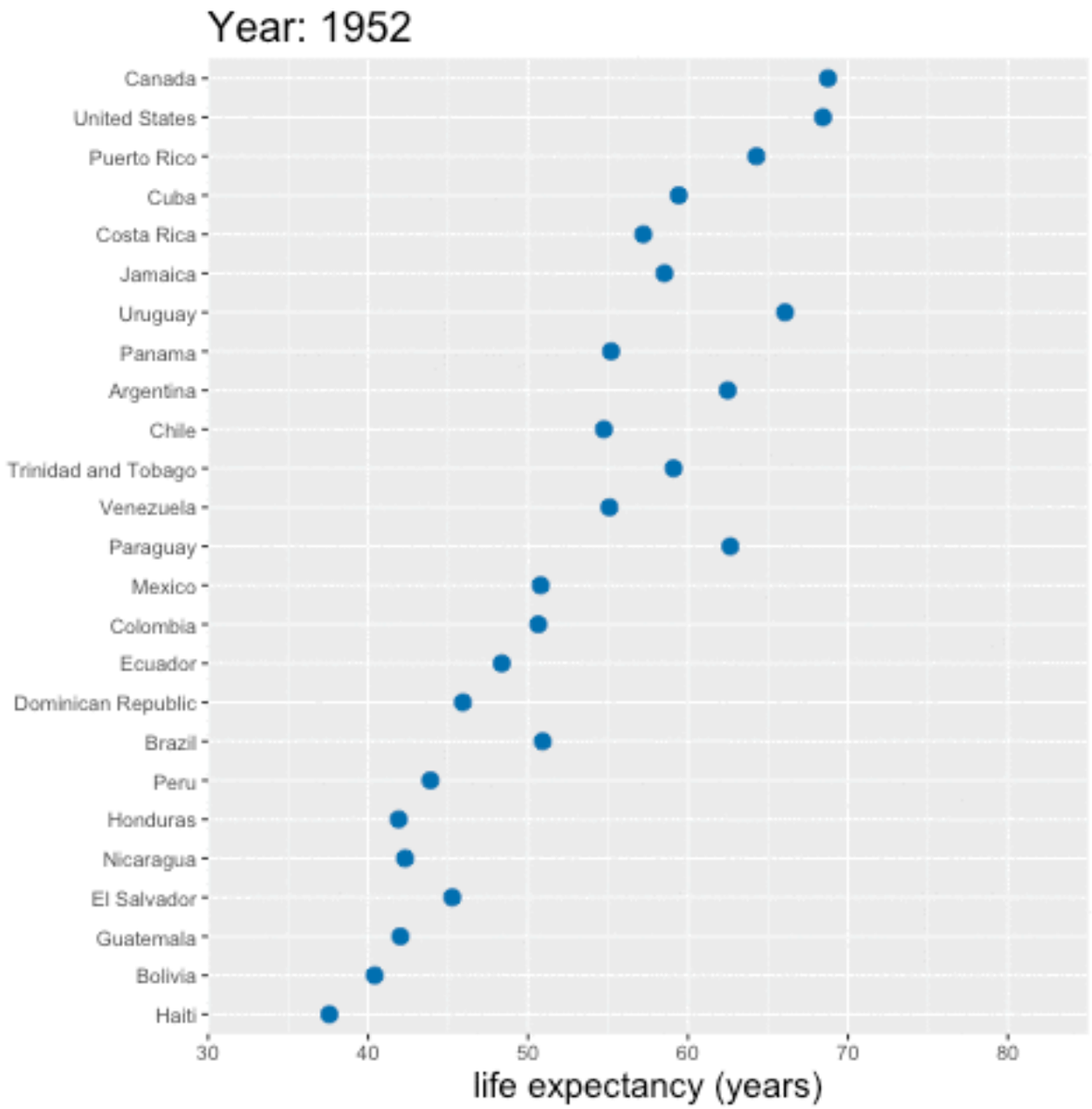


https://github.com/thomasp85/gganimate

Life Expectancy - Americas - Static



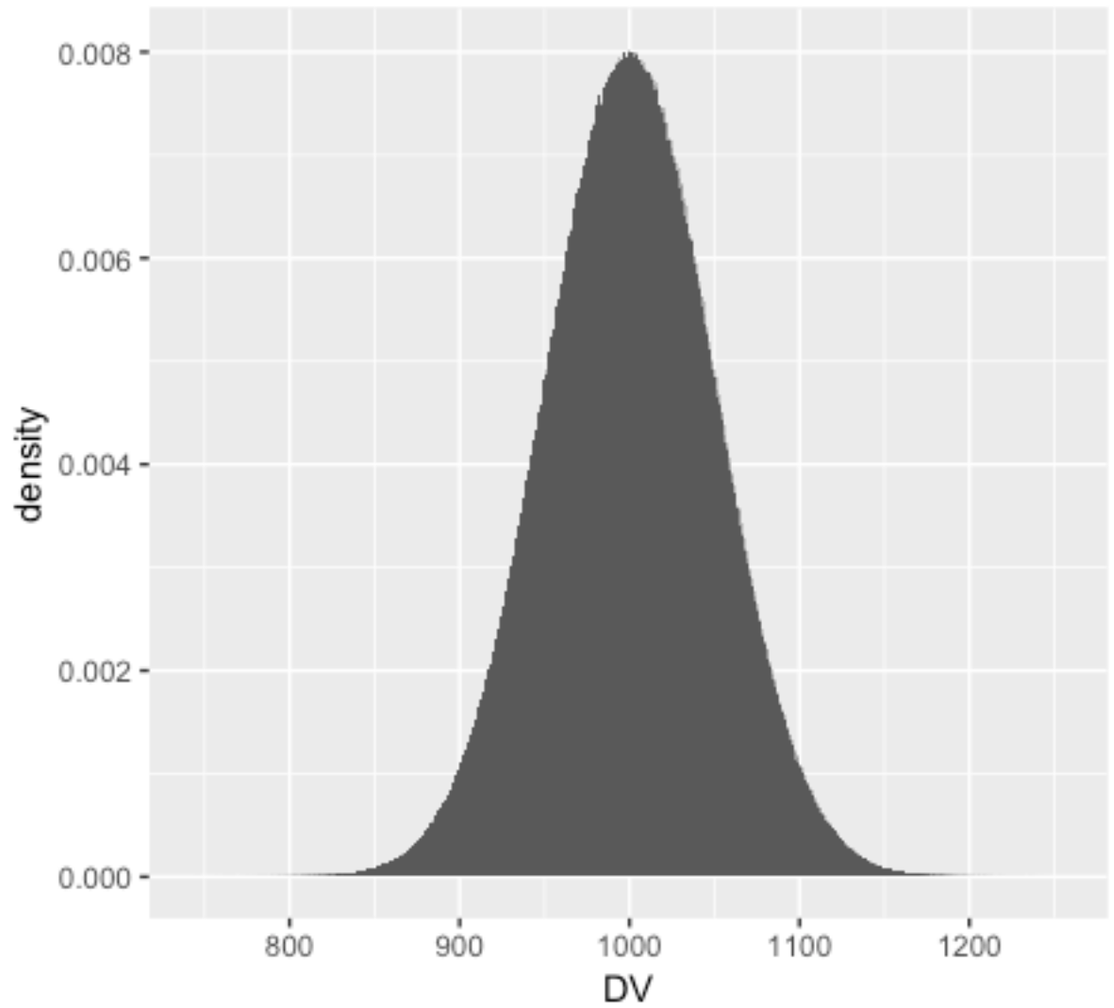
Life Expectancy - Americas - Animated



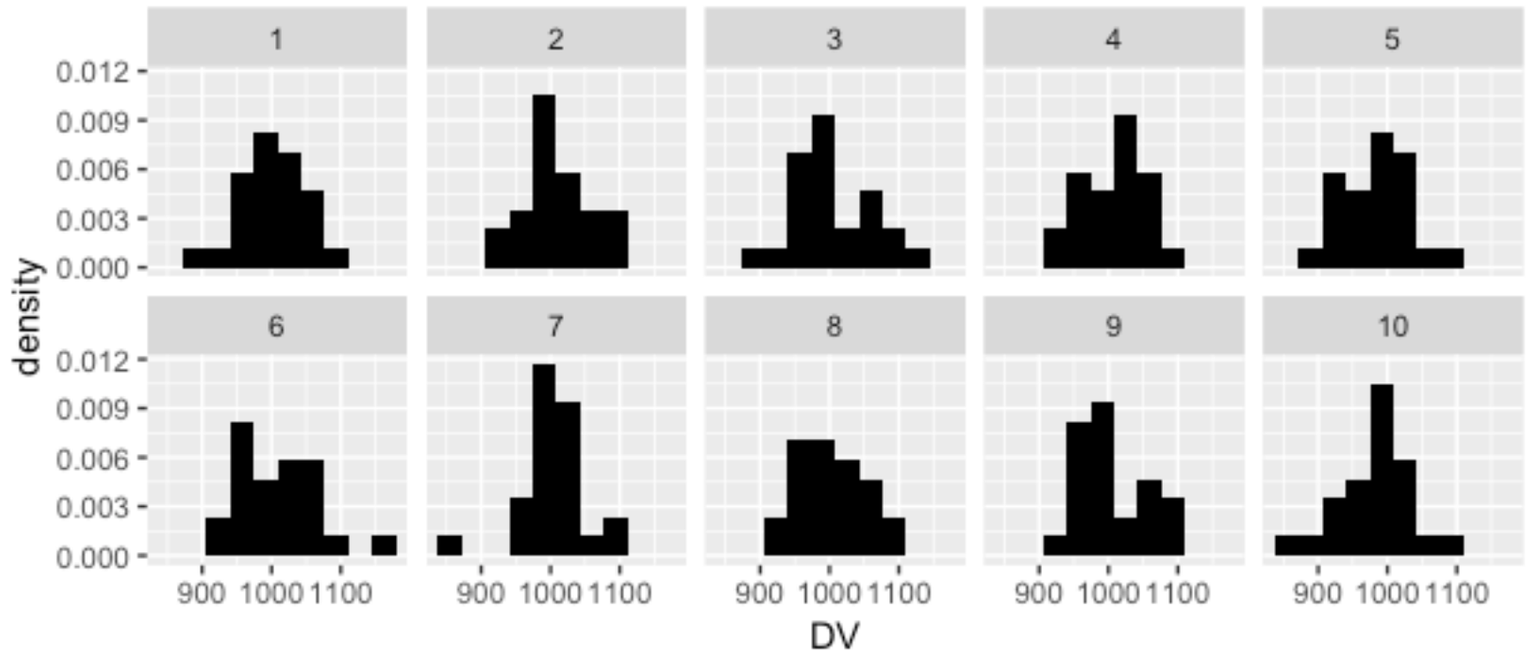
Although we have data only once every 5 years, the `gganimate` package interpolates between each census date to provide a smoother animation.

Using animation and data vis. to understand statistical concepts

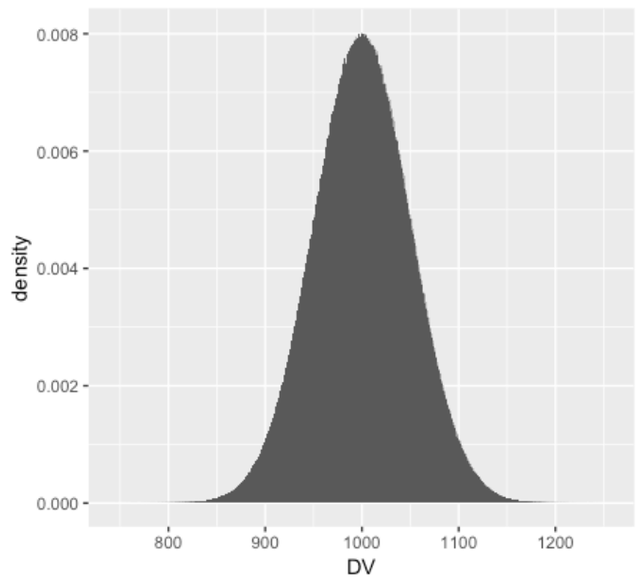
When we sample from a population, we are taking a sample of data points from the population distribution - the population could look something like this:



If our sample sizes are small, few sample distributions actually look like the population from which they're drawn and most sample means are a little different from the population mean:



None of these look much like:

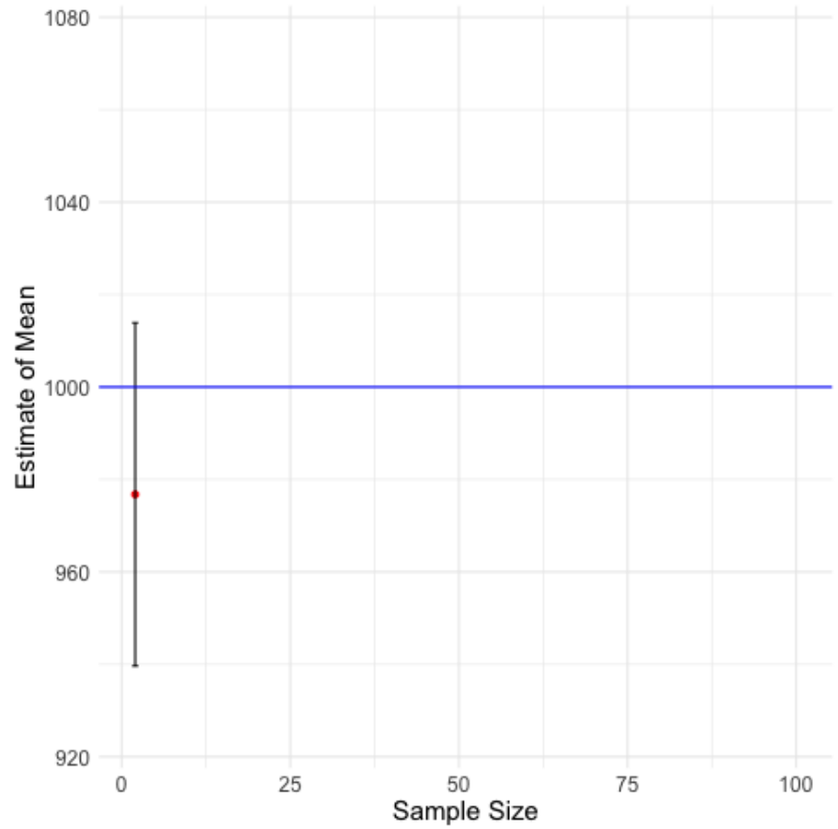


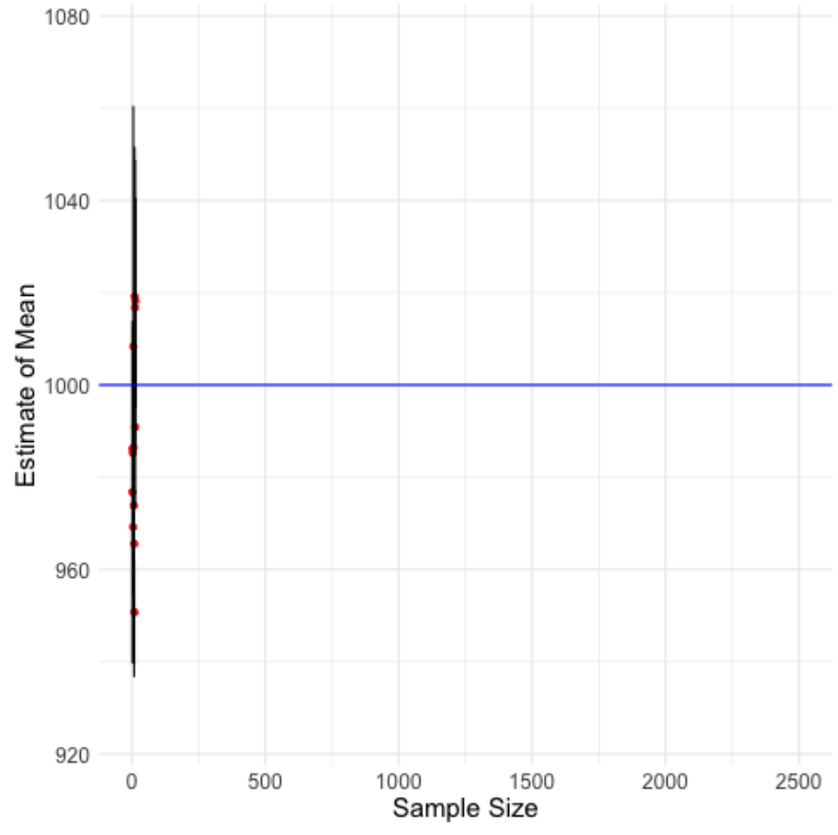
- When we take a sample from a population the mean of the sample may be quite different from the mean of the population (aka sampling error).

- If I take two samples, and work out the mean of these two samples, I should have a better estimate of the population mean than if I just looked at the mean of one of the samples.

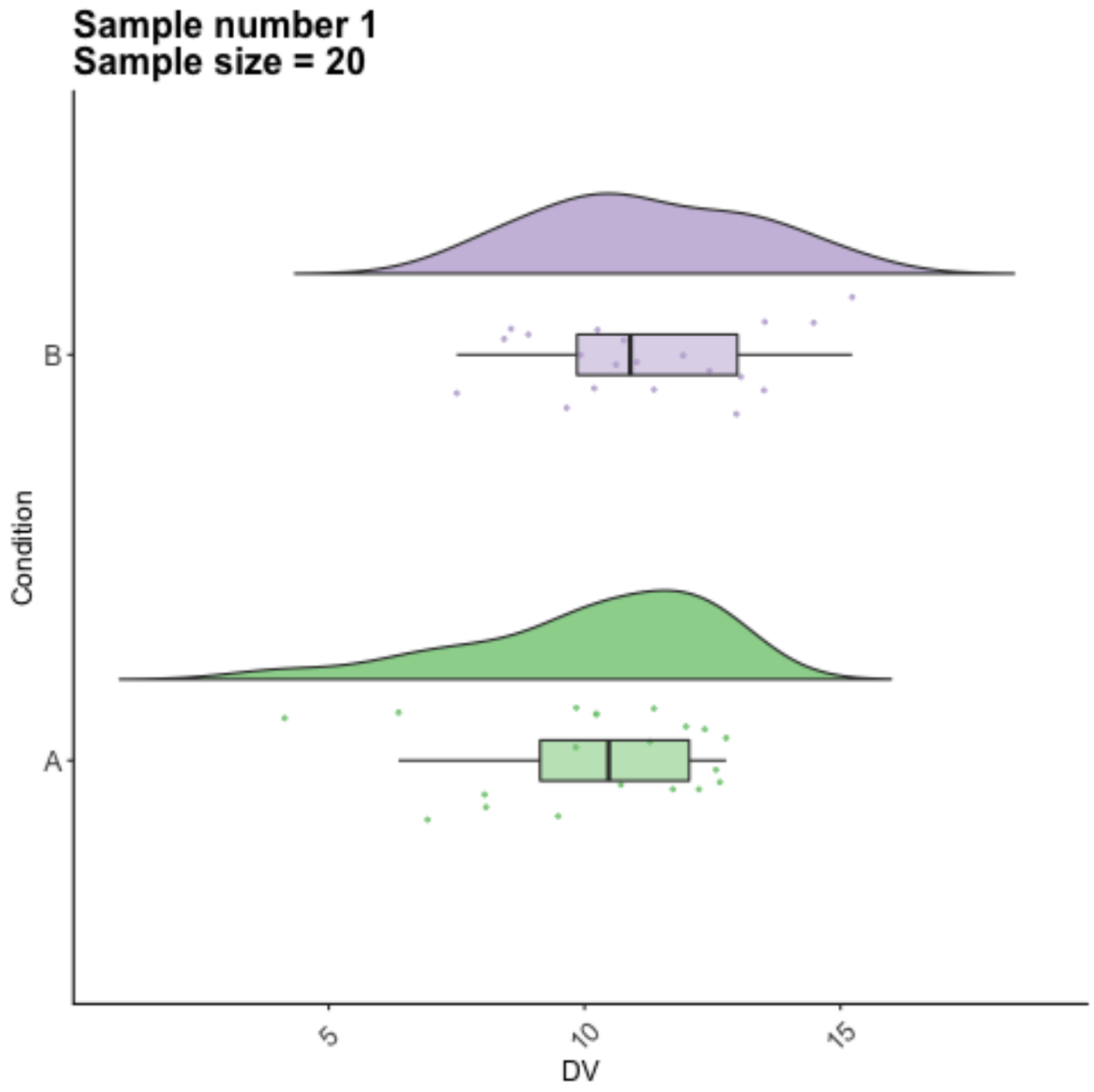
- If I take three samples, work out the mean of these three samples etc. etc.

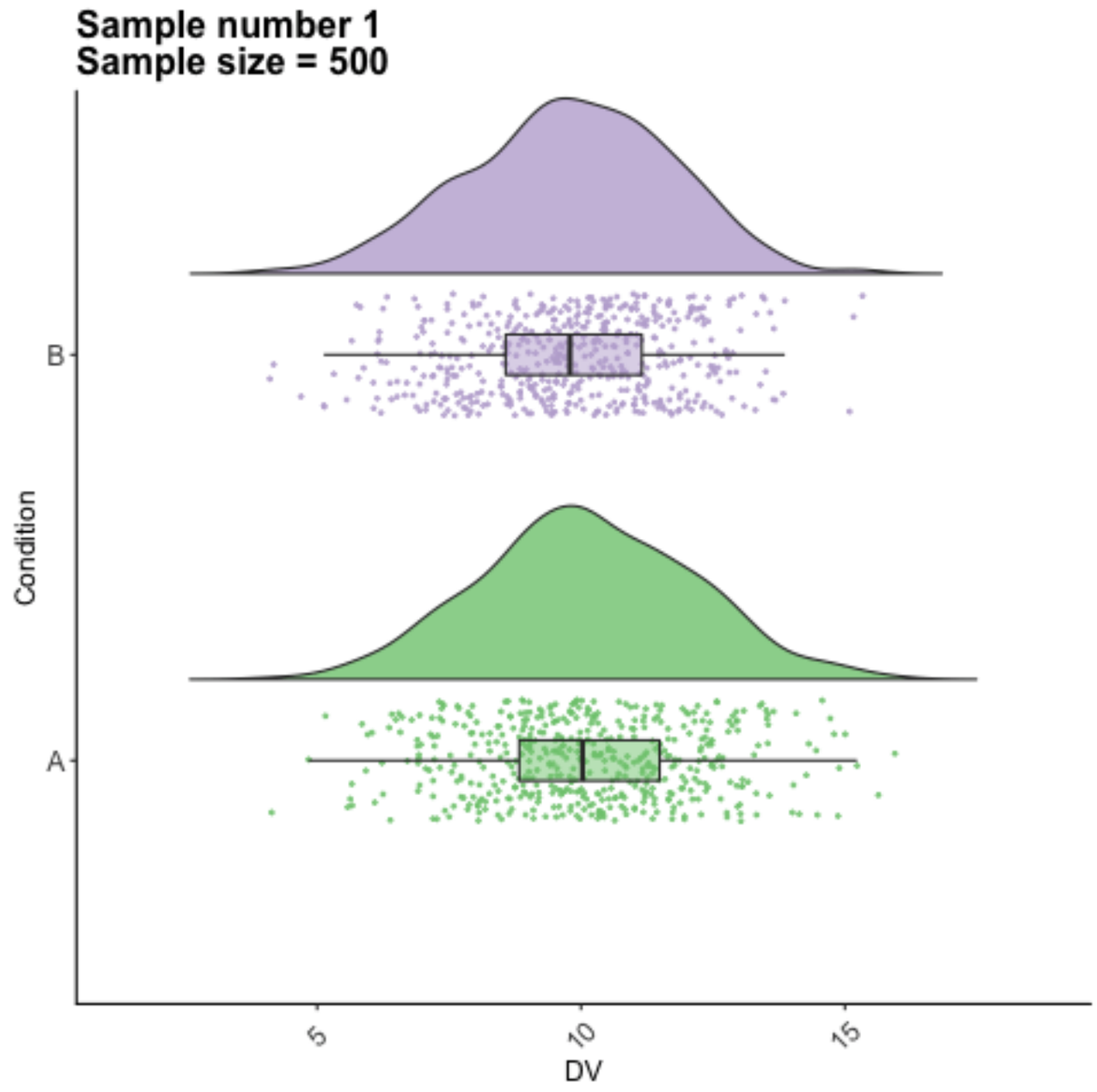
- The larger the sample size we draw from the population, the closer we get to the true mean of the population. As sample size increases, the 95% CI bands around the mean narrow.



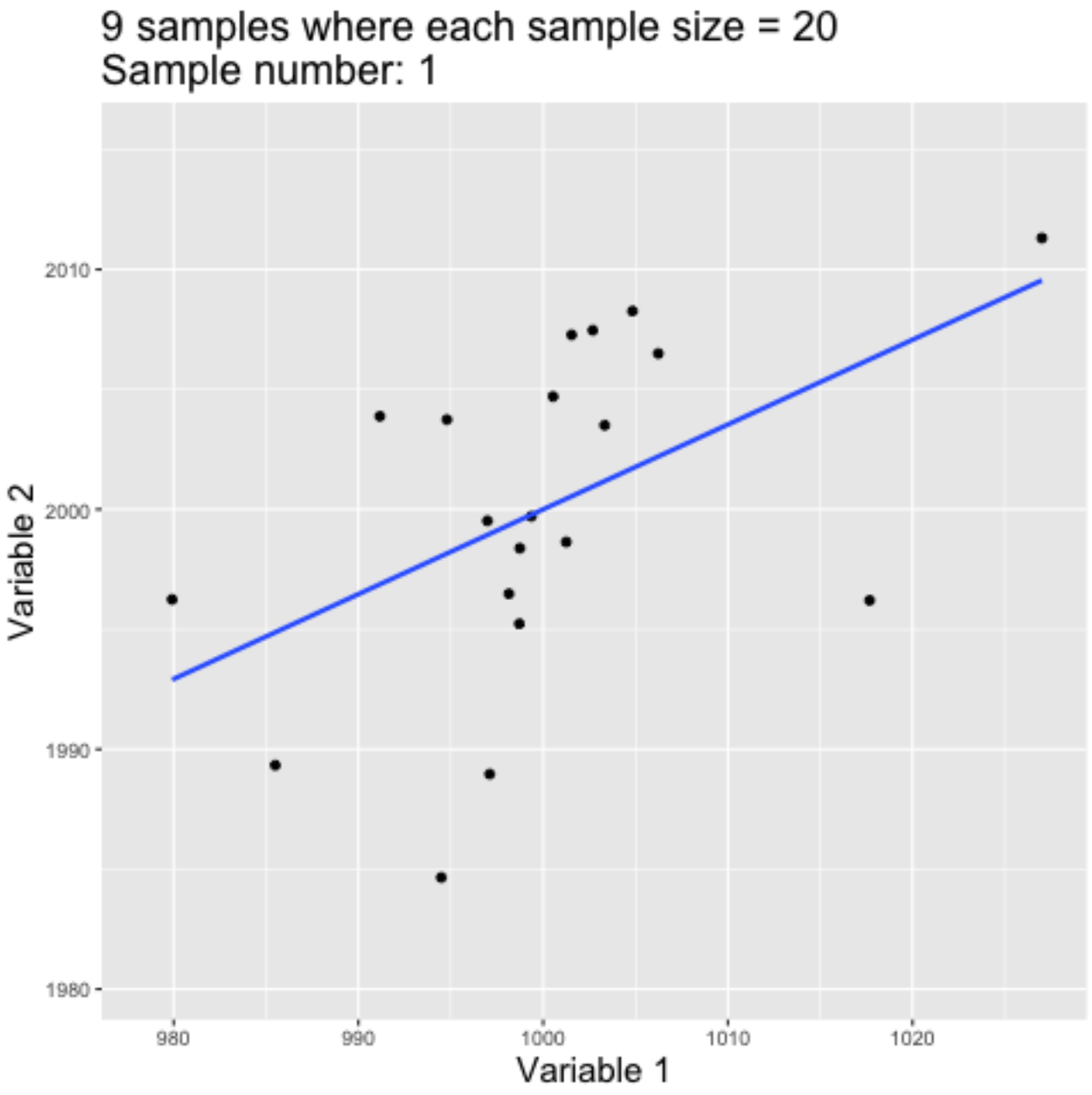


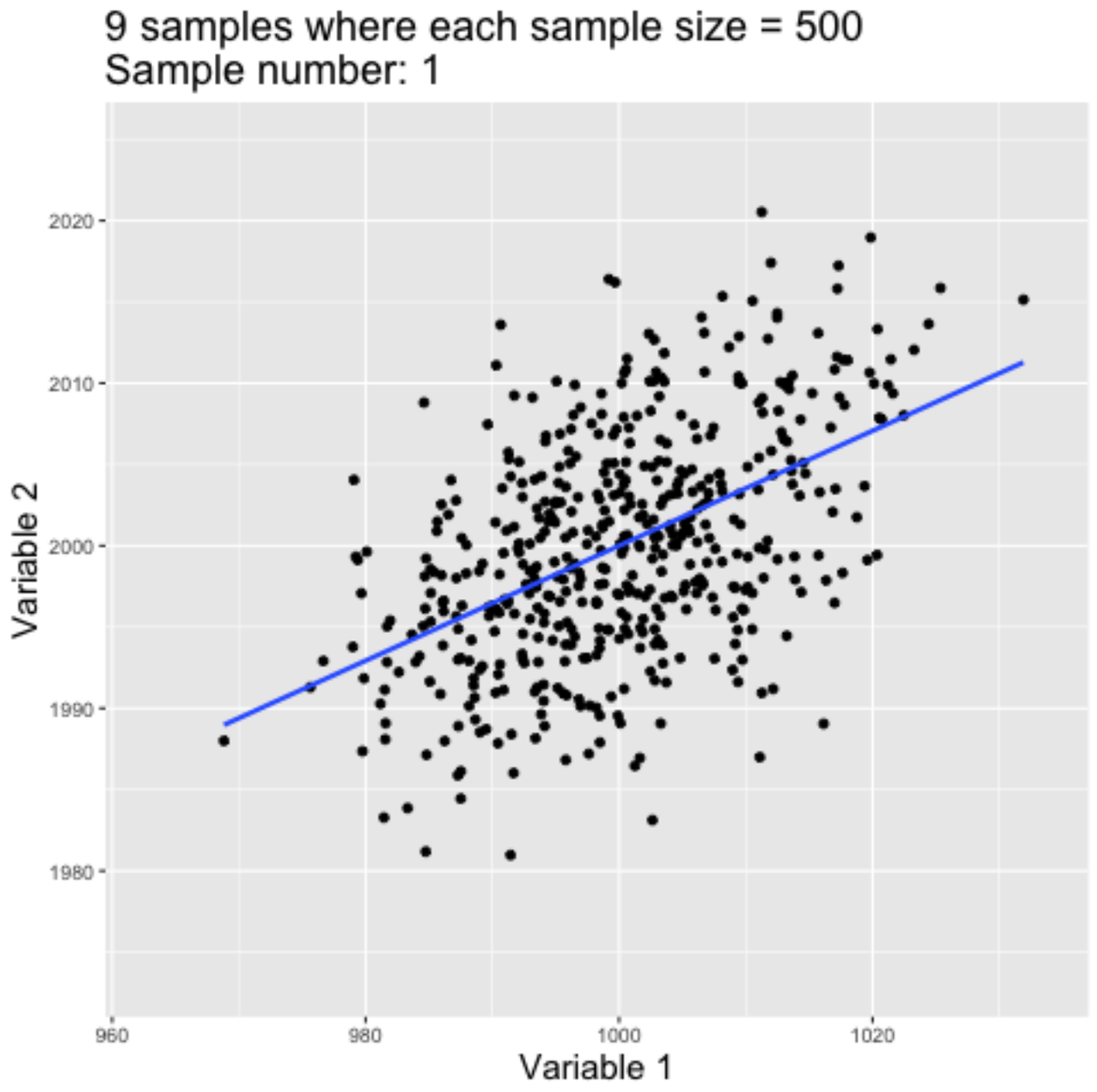
Type I Error - no true effect but sampling error sometimes suggests there is





Type II Error - the true Pearson's r correlation is .5 but sampling error means we can be quite far off that

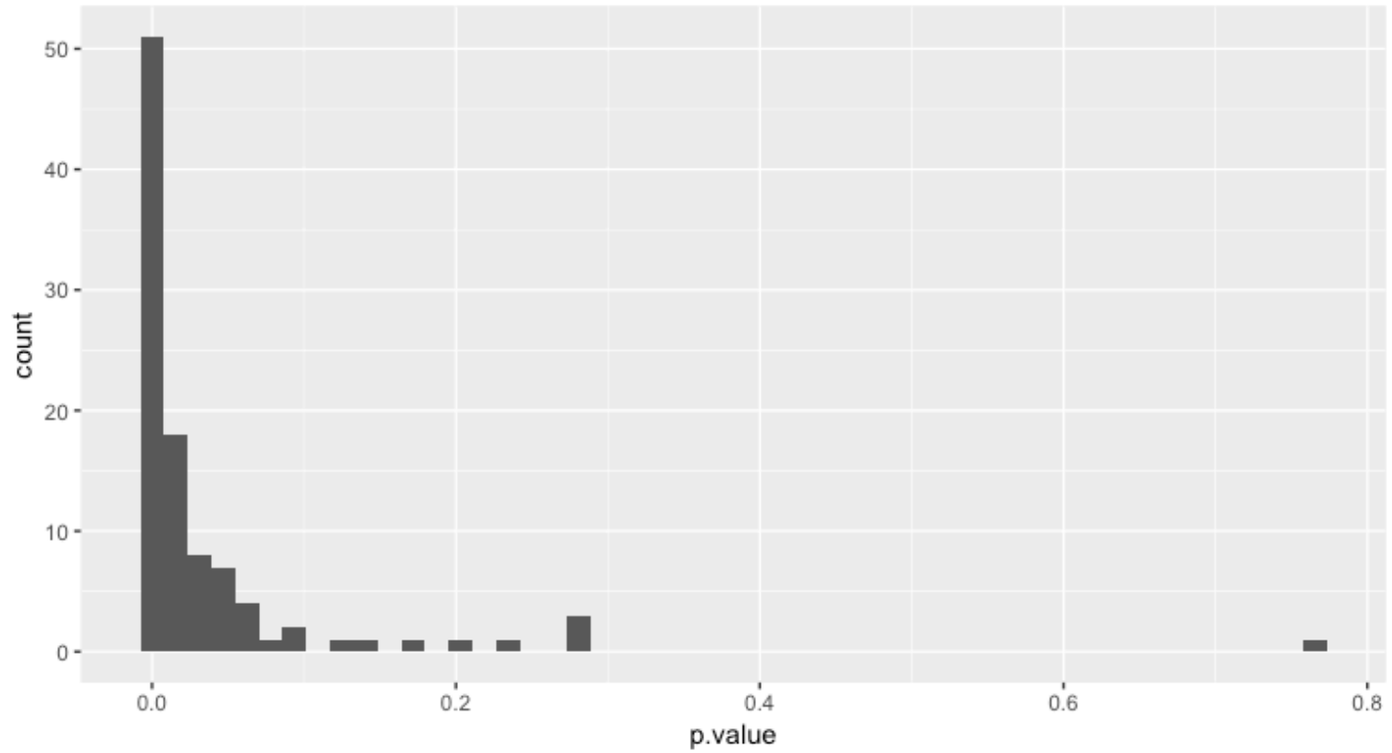


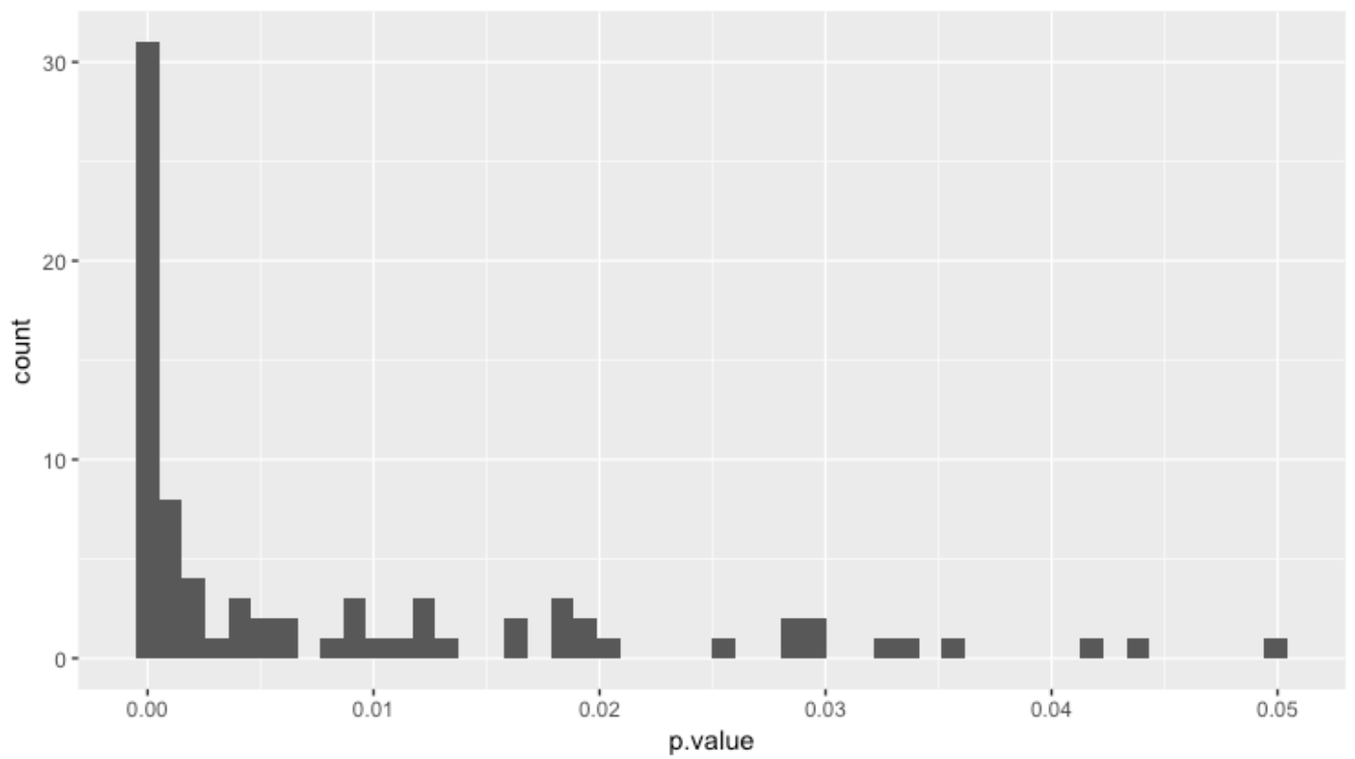



```
0.5000000 0.2260040 0.7353951
0.5880567 0.4925826 0.3528754
0.7855160 0.7630927 0.3522933
```

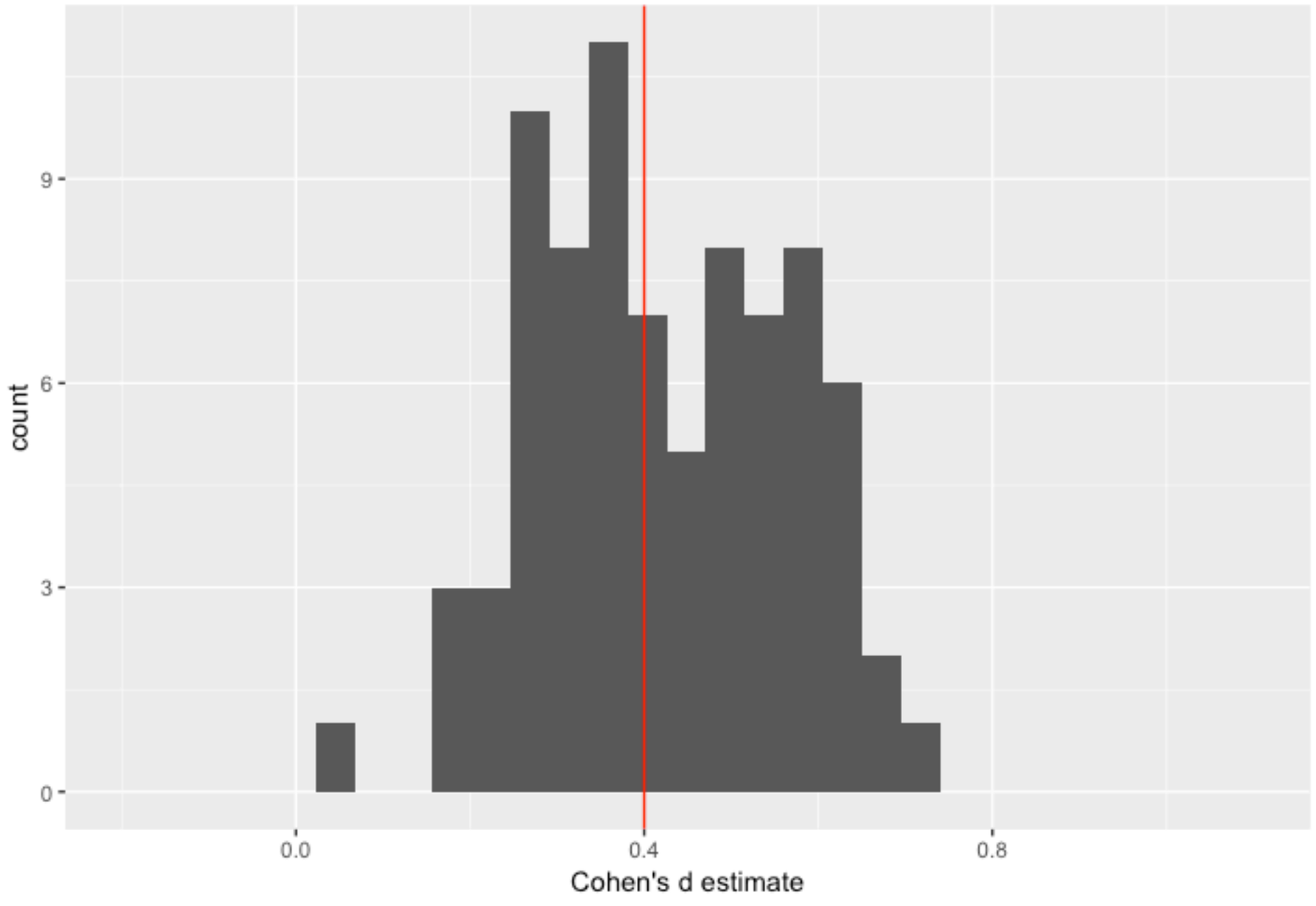
```
0.5000000 0.4784725 0.4890735
0.5256646 0.4814009 0.5406220
0.4959991 0.4909443 0.5155139
```

p-curve for 100 simulations of an experiment with 0.8 power

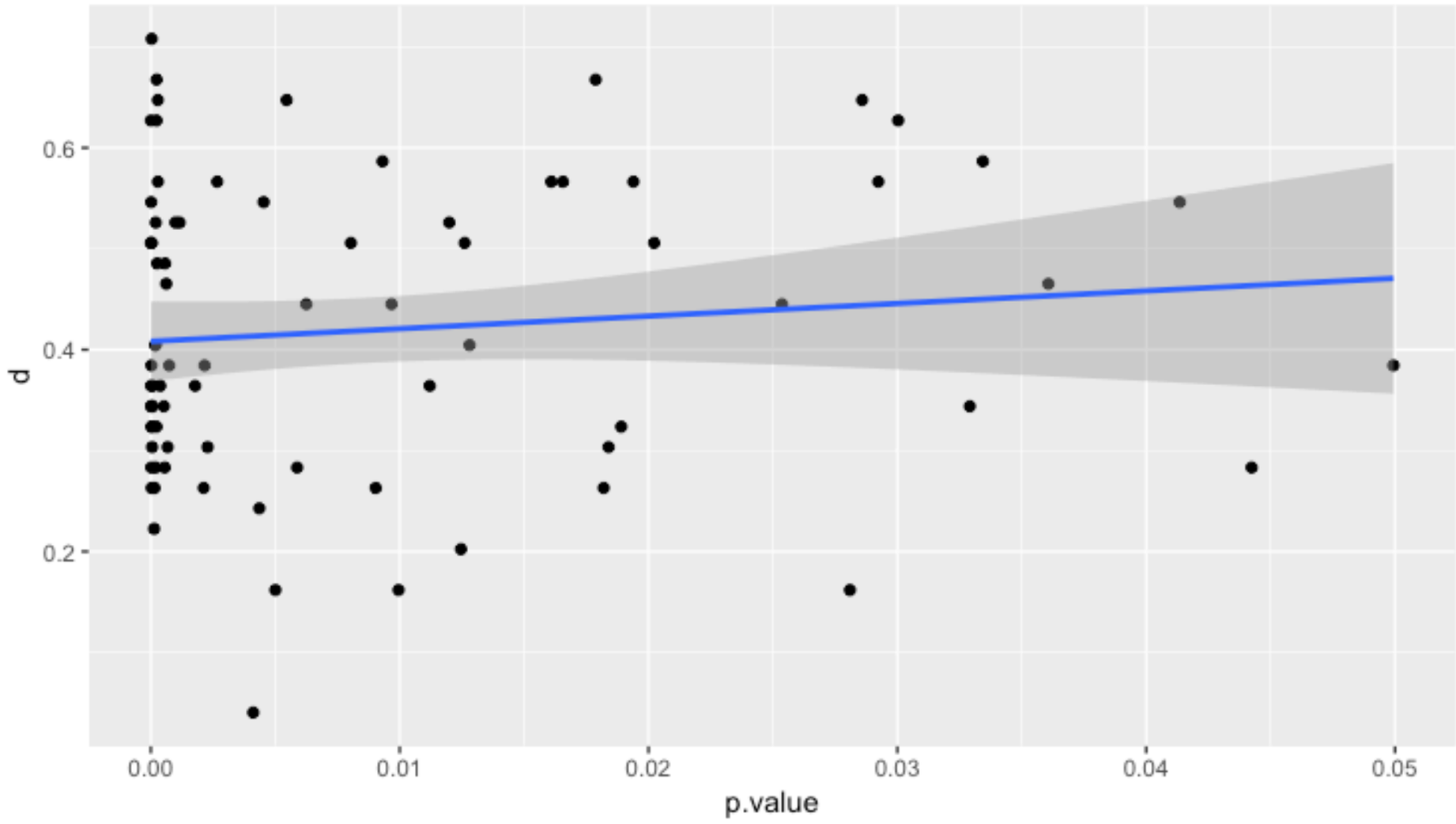




Cohen's d estimates for the 80% of experiments that detected the effect (true effect size - 0.4)



- It’s worth noting, there’s no clear relationship between the p-value of a test and Cohen’s d (i.e., smaller p-values don’t mean bigger effect size estimates). Pearson’s r in this case = 0.11



And using R to run simulations…

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.)

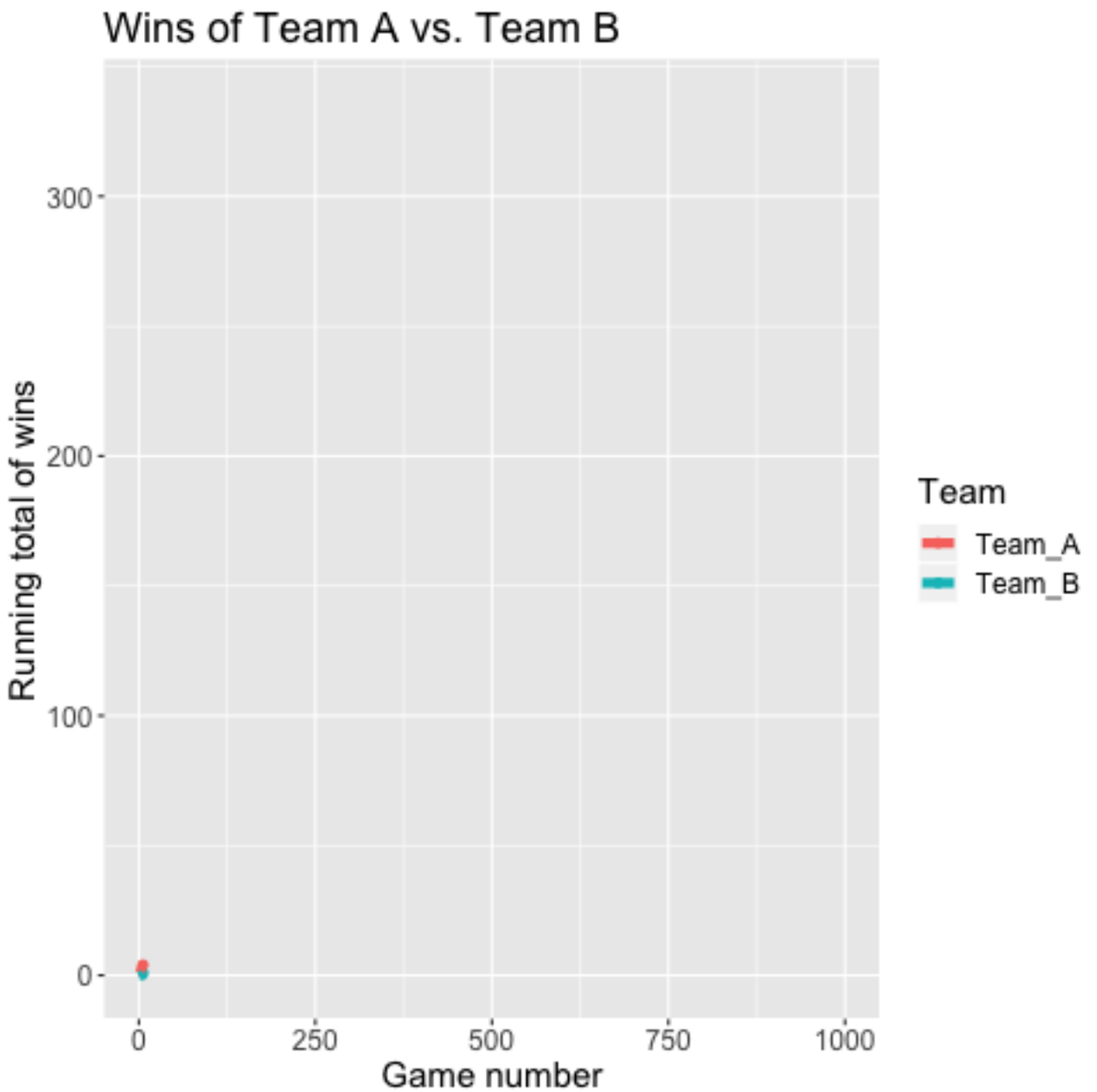
We can code it to simulate the outcomes of 100,000 such games…

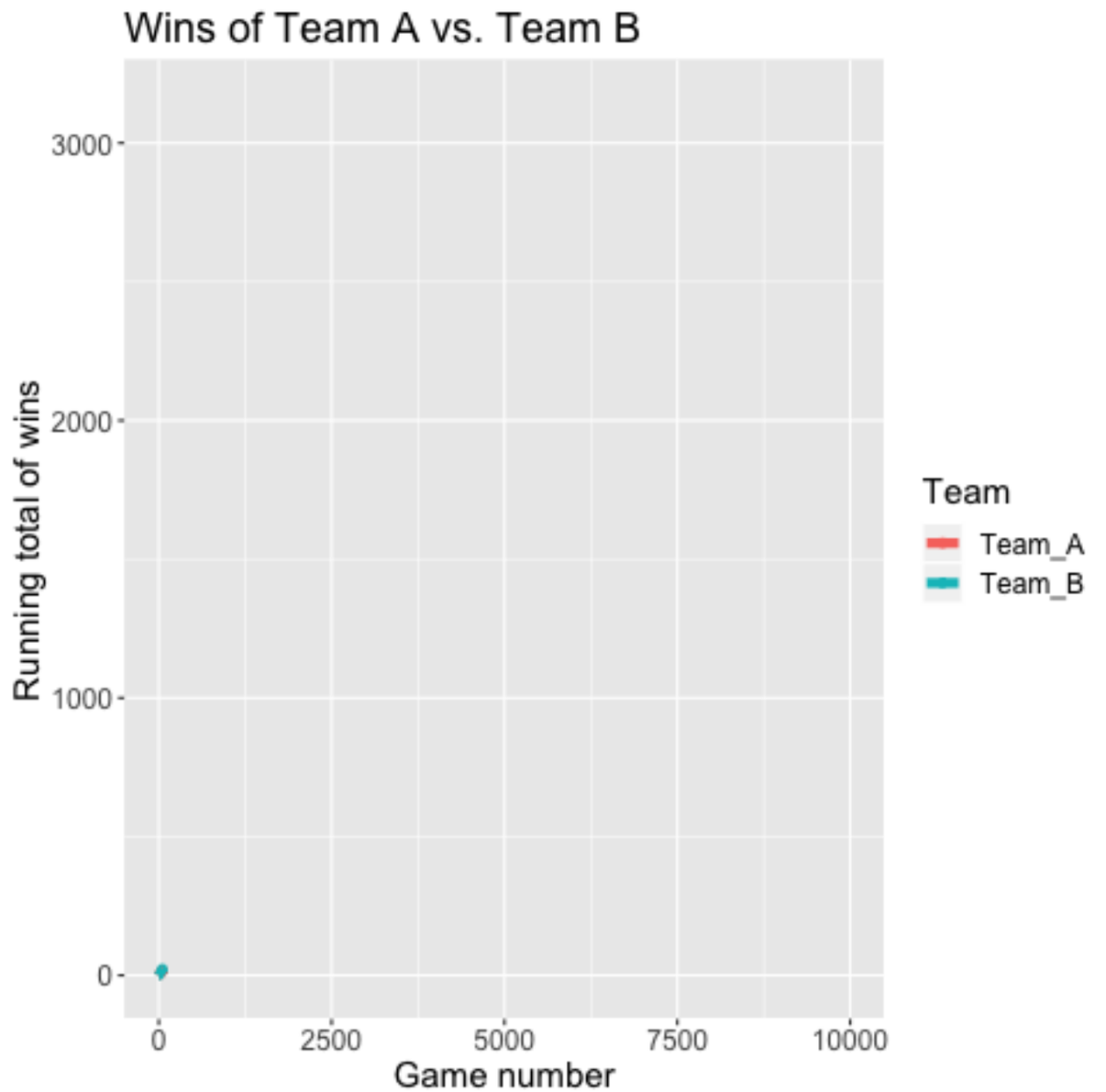
```
set.seed(1234)

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.

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

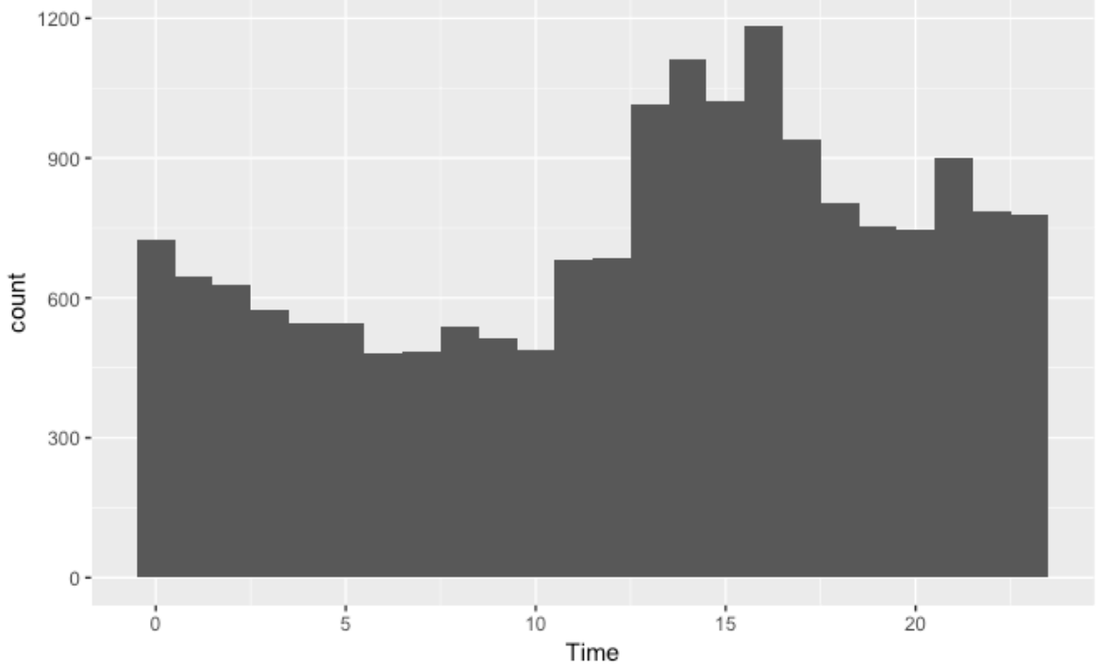




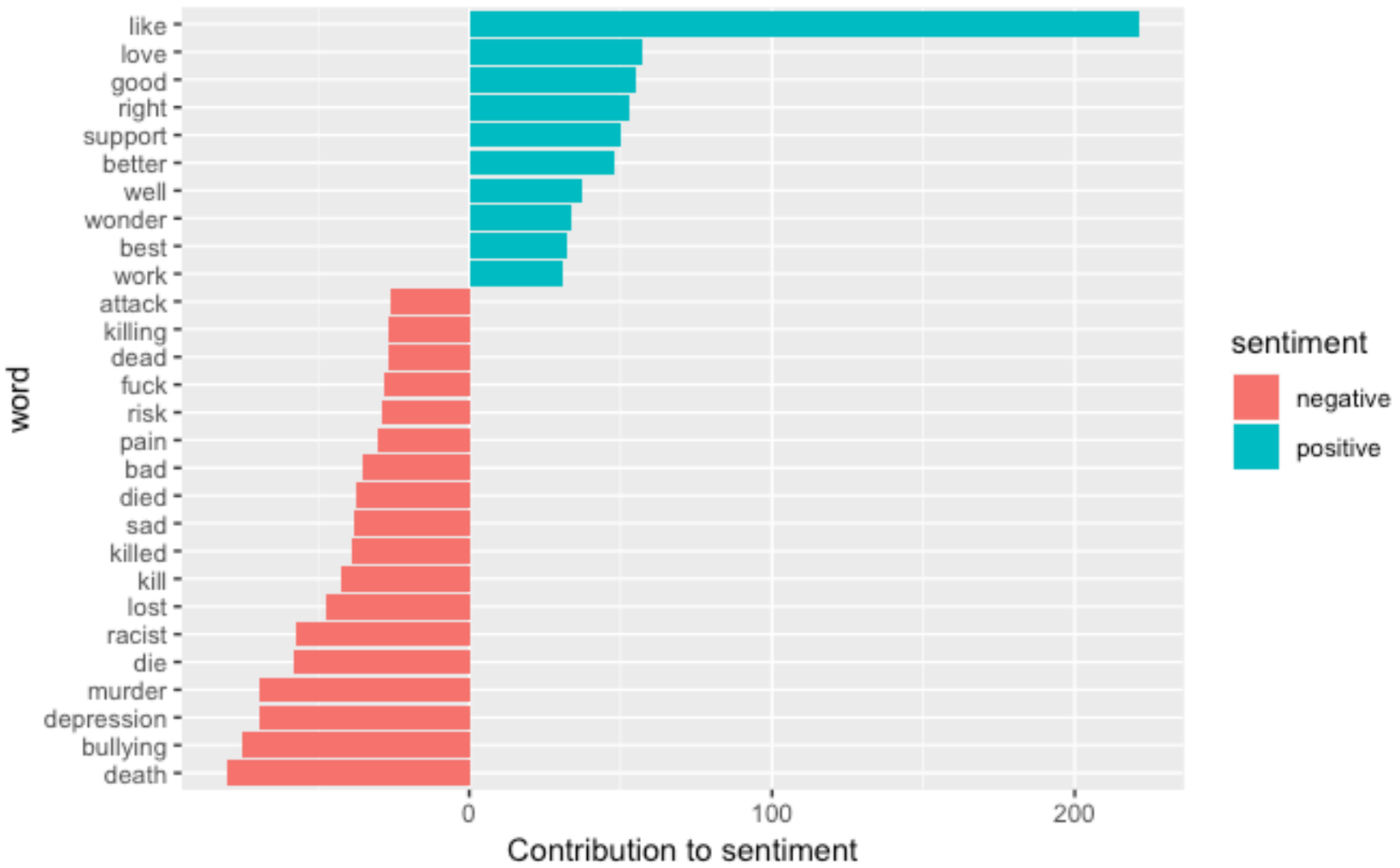
Visualising Text Data

- 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.

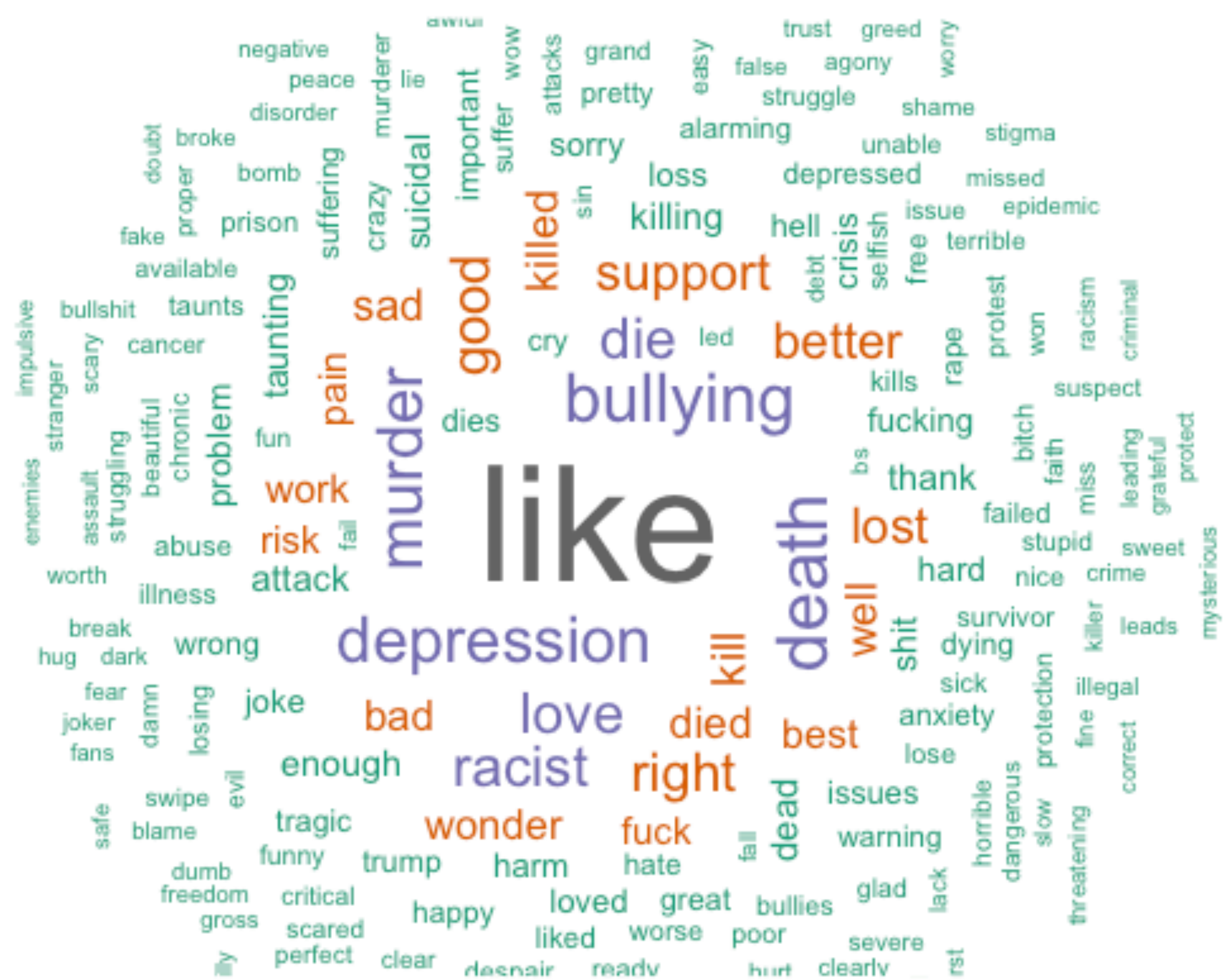
```
tweets <- search_tweets(q = "suicide", n =
1000, include_rts = FALSE, retryonratelimit =
TRUE)

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

time %>%
  filter(!is.na(Time)) %>%
  group_by(Time) %>%
  summarise(count = n()) %>%
  arrange(desc(count))

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.



So, visualising and animation can be used not just to communicate information, but also principles…

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.

That was a brief run through a bunch of key Tidyverse packages and functions for data tidying, wrangling, and visualisation. In the lab, you're going to be doing these kinds of things…