Open Science, Reproducibility, and R.

Andrew Stewart

Division of Neuroscience and Experimental Psychology,

University of Manchester

andrew.stewart@manchester.ac.uk

@ajstewart_lang

Timings...

1000 - Start of morning session
1120 - 1130 Coffee
1130 - Morning session continues
1230 - Lunch
1345 - Start of afternoon session
1445 - 1500 Coffee
1500 - Afternoon session continues
1545 - Building a Binder for fully reproducible research

We have a replication problem...

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.
- Eisner (2018), *Journal of Molecular and Cellular Cardiology*. Reproducibility of science: Fraud, impact factors and carelessness.

Psychological Science
XX(X) 1–6

DOI: 10.1177/0956797610383437
http://pss.sagepub.com

Power Posing: Brief Nonverbal Displays Affect Neuroendocrine Levels and Risk Tolerance

Dana R. Carney[1], **Amy J.C. Cuddy**[2], **and Andy J. Yap**[1]
[1]Columbia University and [2]Harvard University

Abstract

Humans and other animals express power through open, expansive postures, and they express powerlessness through closed, contractive postures. But can these postures actually cause power? The results of this study confirmed our prediction that posing in high-power nonverbal displays (as opposed to low-power nonverbal displays) would cause neuroendocrine and behavioral changes for both male and female participants: High-power posers experienced elevations in testosterone, decreases in cortisol, and increased feelings of power and tolerance for risk; low-power posers exhibited the opposite pattern. In short, posing in displays of power caused advantaged and adaptive psychological, physiological, and behavioral changes, and these findings suggest that embodiment extends beyond mere thinking and feeling, to physiology and subsequent behavioral choices. That a person can, by assuming two simple 1-min poses, embody power and instantly become more powerful has real-world, actionable implications.

Failed replications or effect sizes much smaller than in the original…

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

https://digest.bps.org.uk/2016/09/16/ten-famous-psychology-findings-that-its-been-difficult-to-replicate/

Why are so many studies not replicating?

- There are too many studies with experimental power too low to detect the effect size of interest.

- One of the consequences of a low powered study is that when real effects are detected their magnitude is likely to be over-estimated.

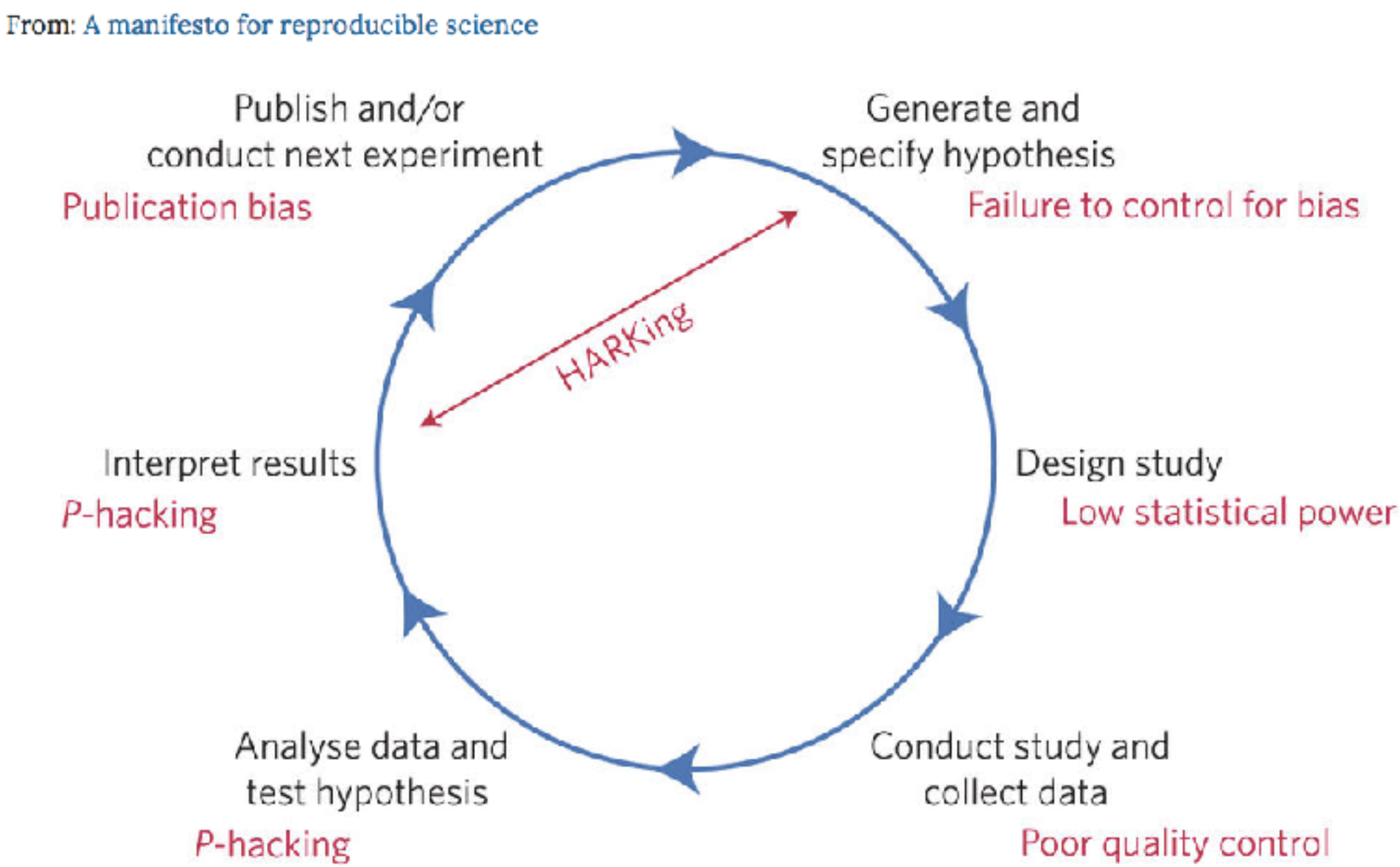
- Studies which find the effect are published and studies that don’t are not published - due to a bias to publish positive results.

- Future work may use the published effect size during *a priori* power analysis (and then fail to find the effect as the new study is effectively under-powered for what it’s looking for).

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

Without realising it, good scientists have been engaging in questionable research practices (QRPs) partly driven by an incentive structure that doesn’t incentivise good scientific practice…

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



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

Personality and Social Psychology Review
1998, Vol. 2, No. 3, 196–217

HARKing: Hypothesizing After the Results are Known

Norbert L. Kerr
Department of Psychology
Michigan State University

This article considers a practice in scientific communication termed HARKing (Hypothesizing After the Results are Known). HARKing is defined as presenting a post hoc hypothesis (i.e., one based on or informed by one's results) in one's research report as if it were, in fact, an a priori hypotheses. Several forms of HARKing are identified and survey data are presented that suggests that at least some forms of HARKing are widely practiced and widely seen as inappropriate. I identify several reasons why scientists might HARK. Then I discuss several reasons why scientists ought not to HARK. It is conceded that the question of whether HARKing's costs exceed its benefits is a complex one that ought to be addressed through research, open discussion, and debate. To help stimulate such discussion (and for those such as myself who suspect that HARKing's costs do exceed its benefits), I conclude the article with some suggestions for deterring HARKing.

ROBERT TAYLOR

Rein in the four horsemen of irreproducibility

Dorothy Bishop *describes how threats to reproducibility, recognized but unaddressed for decades, might finally be brought under control.*

More than four decades into my scientific career, I find myself an outlier among academics of similar age and seniority: I strongly identify with the movement to make the practice of science more robust. It's not that my contemporaries are unconcerned about doing science well; it's just that many of them don't seem to recognize that there are serious problems with current practices. By contrast, I think that, in two decades, we will look back on the past 60 years — particularly in biomedical science — and marvel at how much time and money has been wasted on flawed research.

How can that be? We know how to formulate and test hypotheses in controlled experiments. We can account for unwanted variation with statistical techniques. We appreciate the need to replicate observations.

Yet many researchers persist in working in a way almost guaranteed not to deliver meaningful results. They ride with what I refer to as the four horsemen of the reproducibility apocalypse: publication bias, low statistical power, *P*-value hacking and HARKing (hypothesizing after results are known). My generation and the one before us have done little to rein these in.

In 1975, psychologist Anthony Greenwald noted that science is prejudiced against null hypotheses; we even refer to sound work supporting such conclusions as 'failed experiments'. This prejudice leads to publication bias: researchers are less likely to write up studies that show no effect, and journal editors are less likely to accept them. Consequently, no one can learn from them, and researchers waste time and resources be adequately powered. Other disciplines have yet to catch up.

I stumbled on the issue of *P*-hacking before the term existed. In the 1980s, I reviewed the literature on brain lateralization (how sides of the brain take on different functions) and developmental disorders, and I noticed that, although many studies described links between handedness and dyslexia, the definition of 'atypical handedness' changed from study to study — even within the same research group. I published a sarcastic note, including a simulation to show how easy it was to find an effect if you explored the data after collecting results (D. V. M. Bishop *J. Clin. Exp. Neuropsychol.* **12**, 812–816; 1990). I subsequently noticed similar phenomena in other fields: researchers try out many analyses but report only the ones that are 'statistically significant'.

This practice, now known as *P*-hacking, was once endemic to most branches of science that rely on *P* values to test significance of results, yet few people realized how seriously it could distort findings. That started to change in 2011, with an elegant, comic paper in which the authors crafted analyses to prove that listening to the Beatles could make undergraduates younger (J. P. Simmons *et al. Psychol. Sci.* **22**, 1359–1366; 2011). "Undisclosed flexibility," they wrote, "allows presenting anything as significant."

The term HARKing was coined in 1998 (N. L. Kerr *Pers. Soc. Psychol. Rev.* **2**, 196–217; 1998). Like *P*-hacking, it is so widespread that researchers assume it is good practice. They look at the data, pluck out a finding that looks exciting and write a paper to tell a story around this result. Of course, researchers should be free to explore their

MANY RESEARCHERS PERSIST IN WORKING IN A WAY ALMOST GUARANTEED NOT TO DELIVER MEANINGFUL RESULTS.

https://www.nature.com/articles/d41586-019-01307-2

Distinguishing between replicability and reproducibility (note, both are important!)

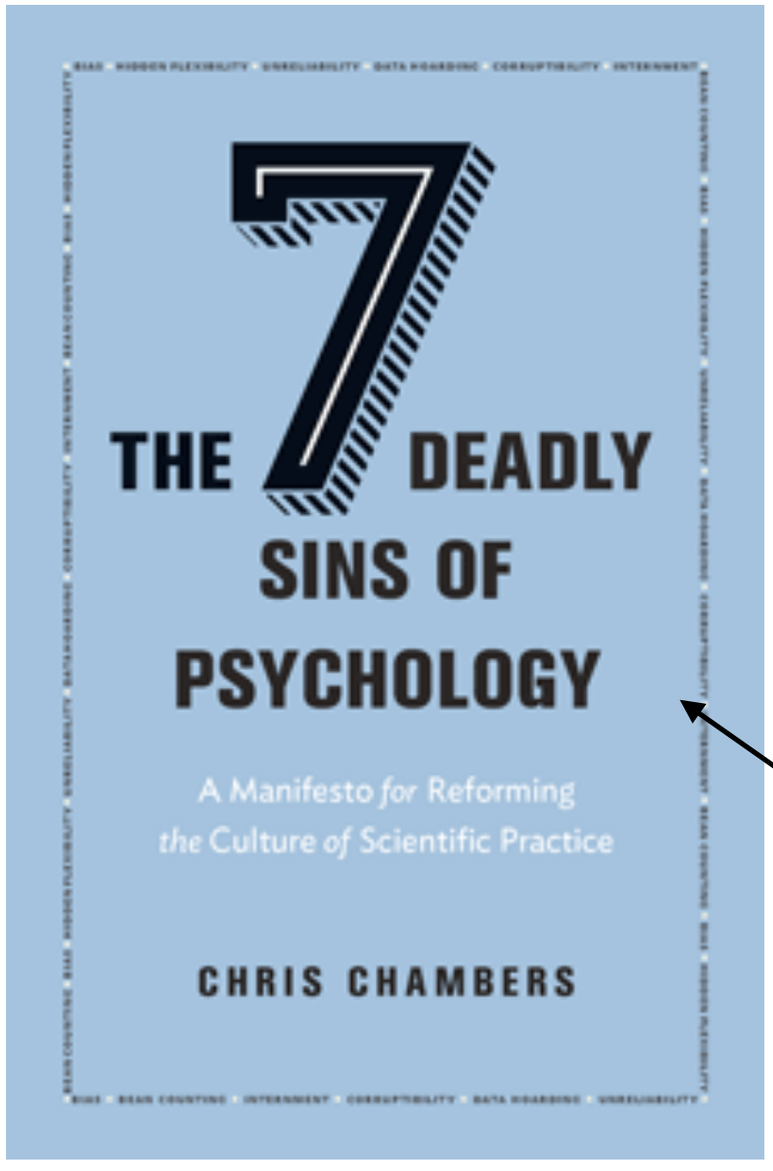
Replicable Science is when someone else can run a study the same as or conceptually equivalent to your one, and find a similar pattern of effects.

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 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.

REF 2019/02
January 2019

Panel criteria and working methods

200. The sub-panels welcome research practice that supports reproducible science and the application of best practice. Examples include registered reports, pre-registration, publication of data sets, experimental materials, analytic code, and use of reporting checklists for publication purposes and those relating to the use of animals in research. These contribute to the evaluation of rigour for submitted outputs. Replication studies may be submitted as outputs and will be evaluated on the extent to which they contribute significant new knowledge, improved methods, or advance theory or practice[1].

346...

Within the context of the institution's strategy, how the submitting unit is progressing towards an open research environment, 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, as appropriate to the discipline. Consideration of reproducibility should also be included where relevant to the discipline.

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.

How do you do Open Science?

Before Data Collection

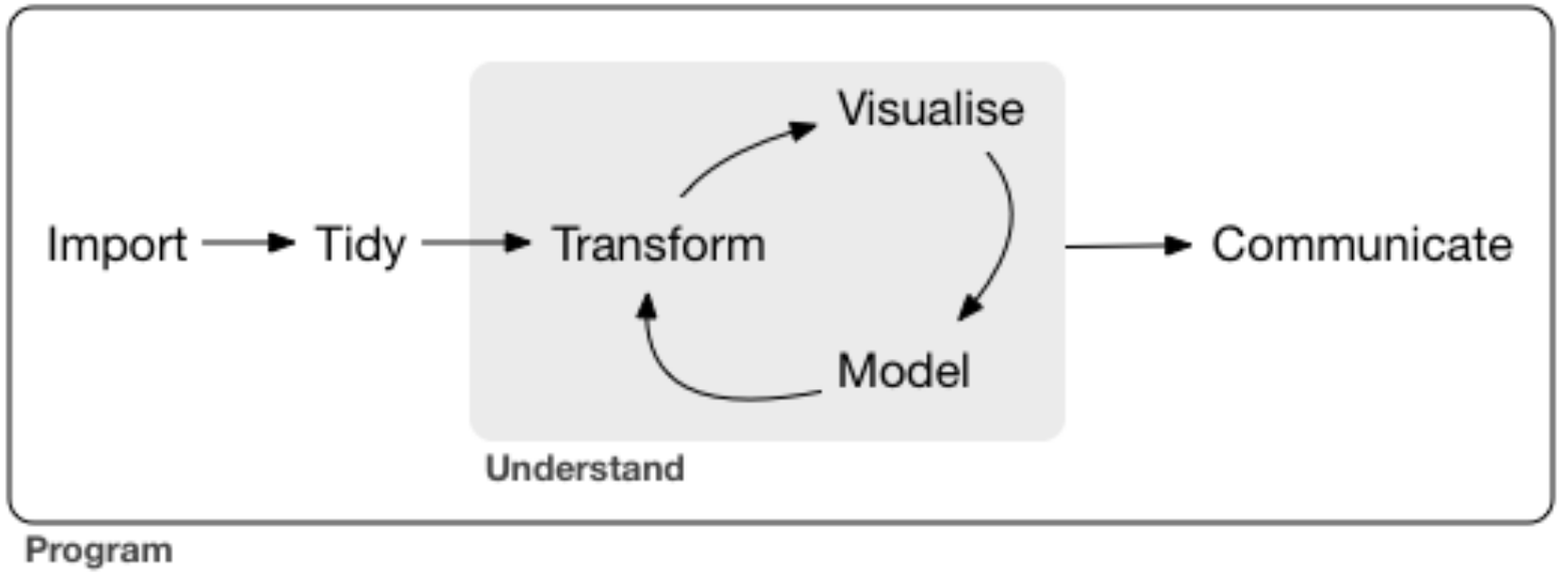
- Specify your hypotheses and analysis plan.
- **Pre-register** your hypotheses and analysis plan at `osf.io`
- Consider data simulation so that you can write your analysis script before you have your real data.
- Consider submitting as a **registered report** - currently **186** journals now support this route. This involves acceptance in principle before you have even started collecting your data.

Registered Reports

https://cos.io/rr/?_ga=2.49773158.1336120275.1555407527-1361001319.1494339346

After Data Collection

- You need to use analysis software that allows for open sharing and reproducibility of the entire data wrangling/ analysis/write-up workflow.



Hadley Wickham and Garrett Grolemund

- You can share your data at `osf.io` or on GitHub:

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

Watch 0 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Branch: master ▾ Comprehension-of-indirect-requests-is-influenced-by-their-degree-of-imposition / RPs.csv

Find file | Copy path

ajstewartlang Made consistent the labelling of factors in data files and in paper 7b3b3b1 on 29 Mar 2017

0 contributors

1681 lines (1681 sloc) | 69.7 KB

Raw | Blame | History

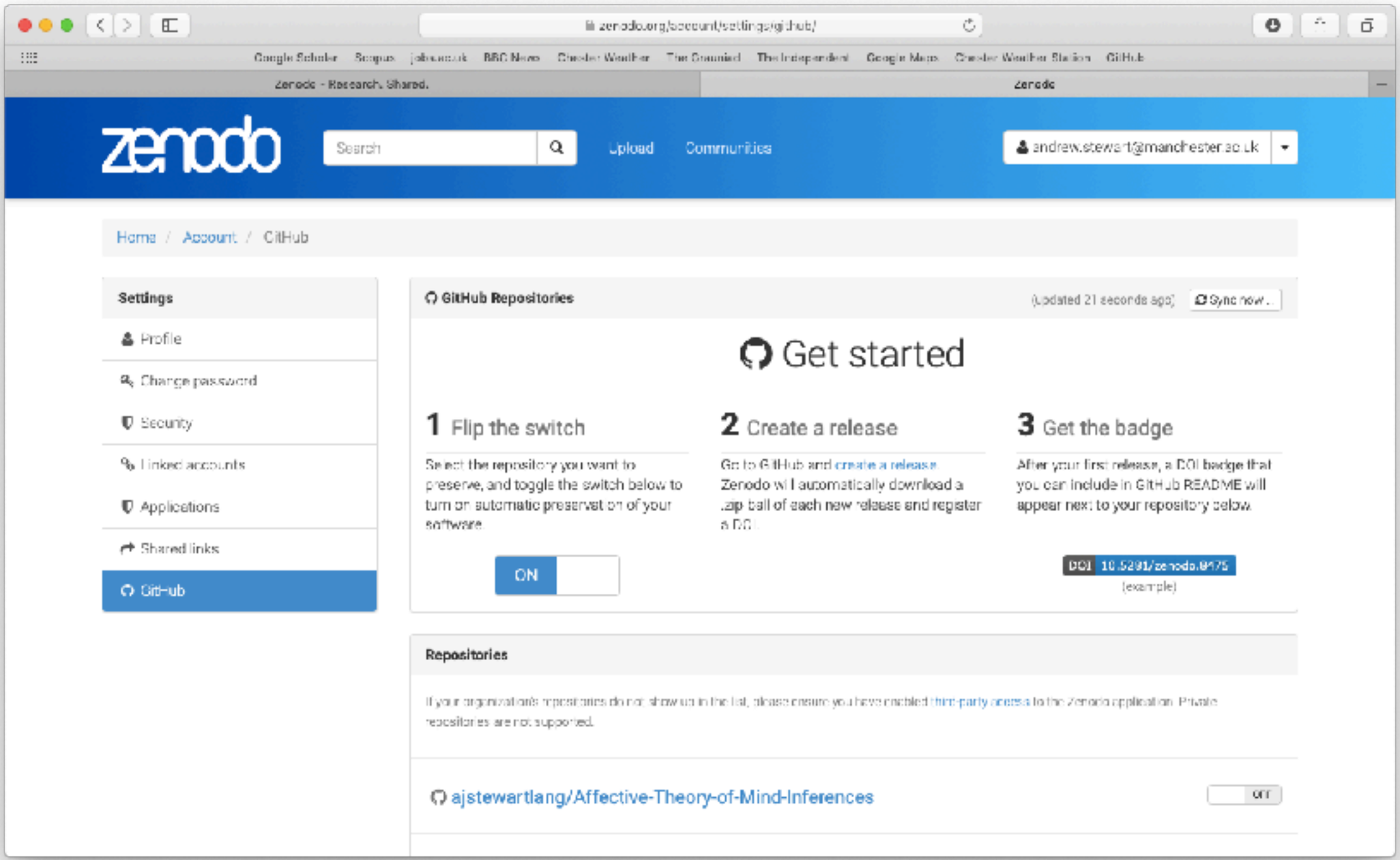
Search this file...

	P.s	Item	Condition	Probmanip	Speaker	statement	response	final	Meaning	Imposition
2	1	1	1	1708	302	1389	1867	1205	Indirect	High
3	1	2	2	1466	296	1377	1674	828	Indirect	Low
4	1	3	3	1393		1494	1950	1812	Direct	High
5	1	4	4	2453	530	1691	1866	965	Direct	Low
6	1	5	1	1562	267	1332	1477	1345	Indirect	High
7	1	6	2	1446	444	1004	1067	797	Indirect	Low
8	1	7	3	2159	501	739	1231	2240	Direct	High
9	1	8	4	1459		1086	946	978	Direct	Low
10	1	9	1	3302		1503	900	1736	Indirect	High

- alongside your analysis code

```
FPs$Meaning <- as.factor(FPs$Meaning)
FPs$Imposition <- as.factor(FPs$Imposition)

#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 (FPs$Meaning) <- my.coding
contrasts (FPs$Imposition) <- my.coding

#construct the models with crossed random effects for subjects and items for the pre-critical, critical and post-crtical region
fpmodelprec <- lmer (Probmanip ~ Meaning*Imposition + (1+Meaning*Imposition |P.s) + (1+Meaning+Imposition |Item), data=FPs, REM
summary (fpmodelprec)
lsmeans (fpmodelprec, pairwise~Meaning*Imposition, adjust="none")

fpmodelc <- lmer (statement ~ Meaning*Imposition + (1+Meaning*Imposition |P.s) + (1+Meaning*Imposition |Item), data=FPs, REML=T
summary (fpmodelc)
lsmeans (fpmodelc, pairwise~Meaning*Imposition, adjust="none")

fpmodelpostc <- lmer (response ~ Meaning*Imposition + (1+Meaning*Imposition |P.s) + (1+Meaning+Imposition |Item), data=FPs, REM
summary (fpmodelpostc)
lsmeans (fpmodelpostc, pairwise~Meaning*Imposition, adjust="none")

#Regression Path Analysis
#Read in Regression Path data
RPs <- read.csv("~/RPs.csv")

RPs$Meaning <- as.factor(RPs$Meaning)
RPs$Imposition <- as.factor(RPs$Imposition)

contrasts (RPs$Meaning) <- my.coding
contrasts (RPs$Imposition) <- my.coding

#construct the models with crossed random effects for subjects and items for the pre-critical, critical and post-crtical region
rpmodelprec <- lmer (Probmanip ~ Meaning*Imposition + (1+Meaning*Imposition |P.s) + (1+Meaning*Imposition |Item), data=RPs, REM
```

And preserve it with a DOI via Zenodo



Qualitative

- You can pre-register your analysis protocol in advance - much like with quantitative research.
- Although there are some caveats around making qualitative data fully open (espec. in light of confidentiality issues), there is a good discussion of some possible solutions here:

https://www.mededpublish.org/manuscripts/1986

Using R for Reproducible Data Science

"Hadley Wickham, the Man Who Revolutionized R"

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

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

Data Science is 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

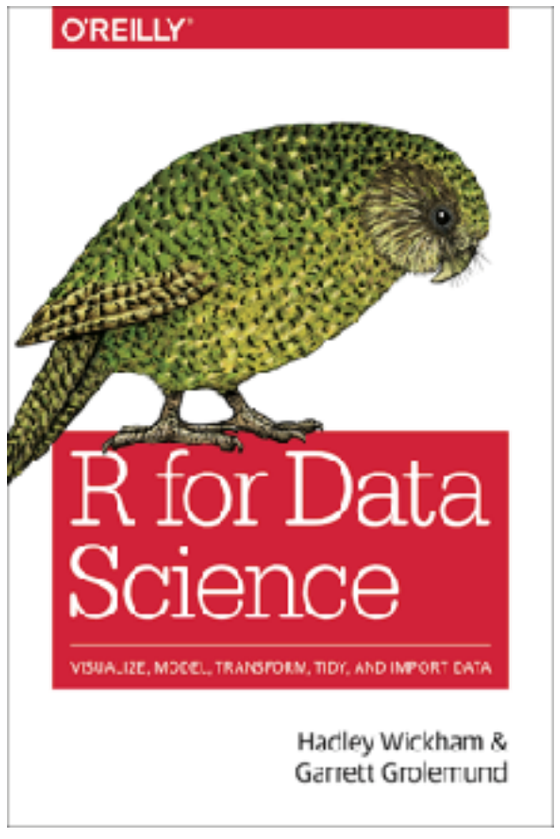
Every time you use an Internet search engine, sign up for a company "rewards" program or swipe your credit card, that information is saved and stored. The industries that amass these billions of bytes of data are increasingly hiring psychologists to help make sense of it, says Douglas Reynolds, PhD,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

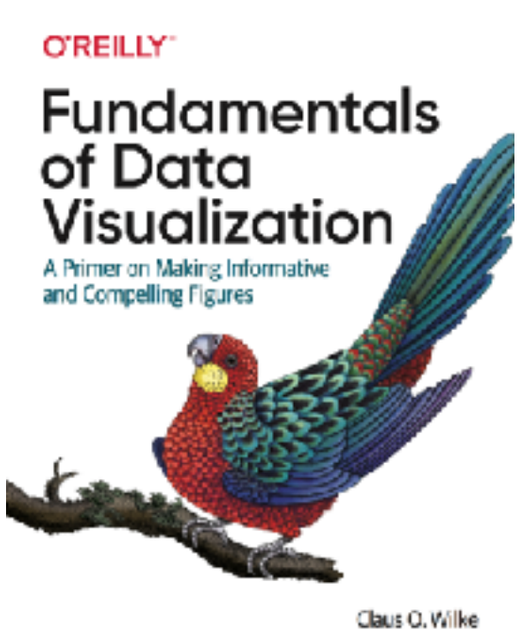
What role can R play in Open Science?

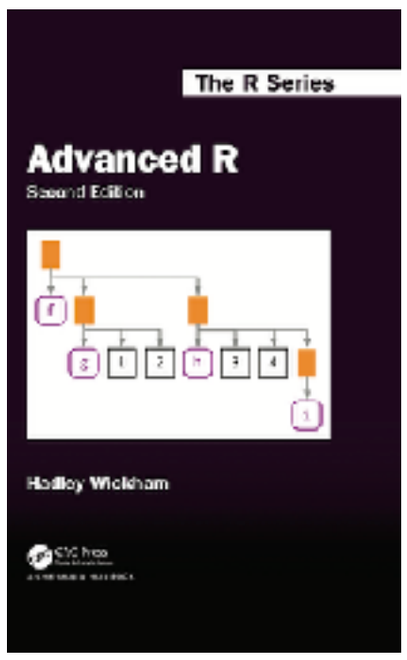
- R scripts are easy to share allowing for reproducibility and easy public sharing of data and code.
- R is free, open source software that is much more flexible and powerful than SPSS.
- There is an active R community continuously updating statistical tests and packages that run in R.
- As R is a programming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

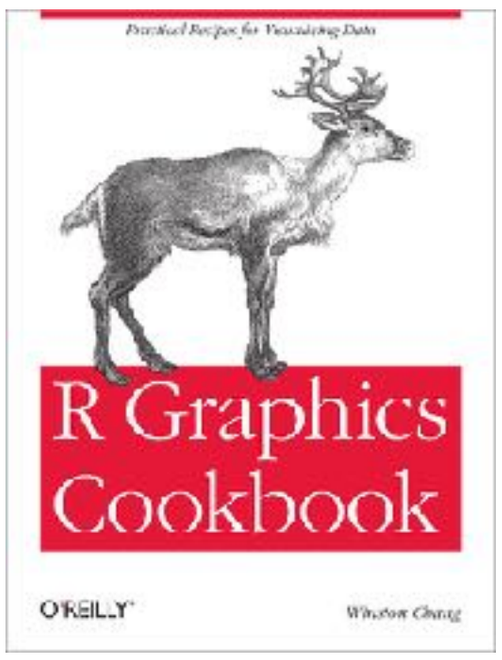


Available electronically for free at:

`http://r4ds.had.co.nz`







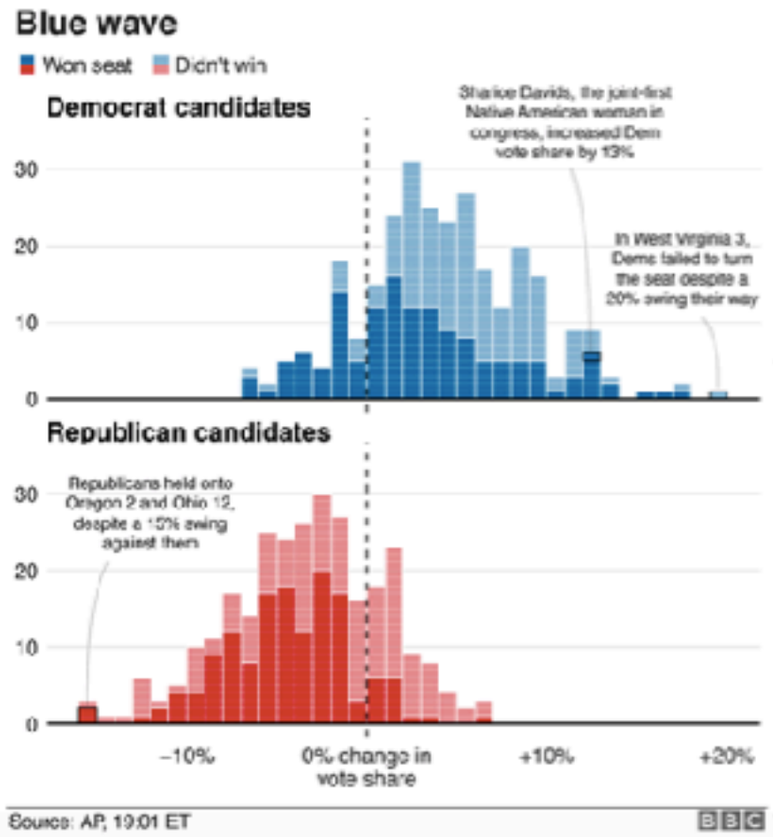
BBC Visual and Data Journalism cookbook for R graphics

Last updated: 2019-01-24

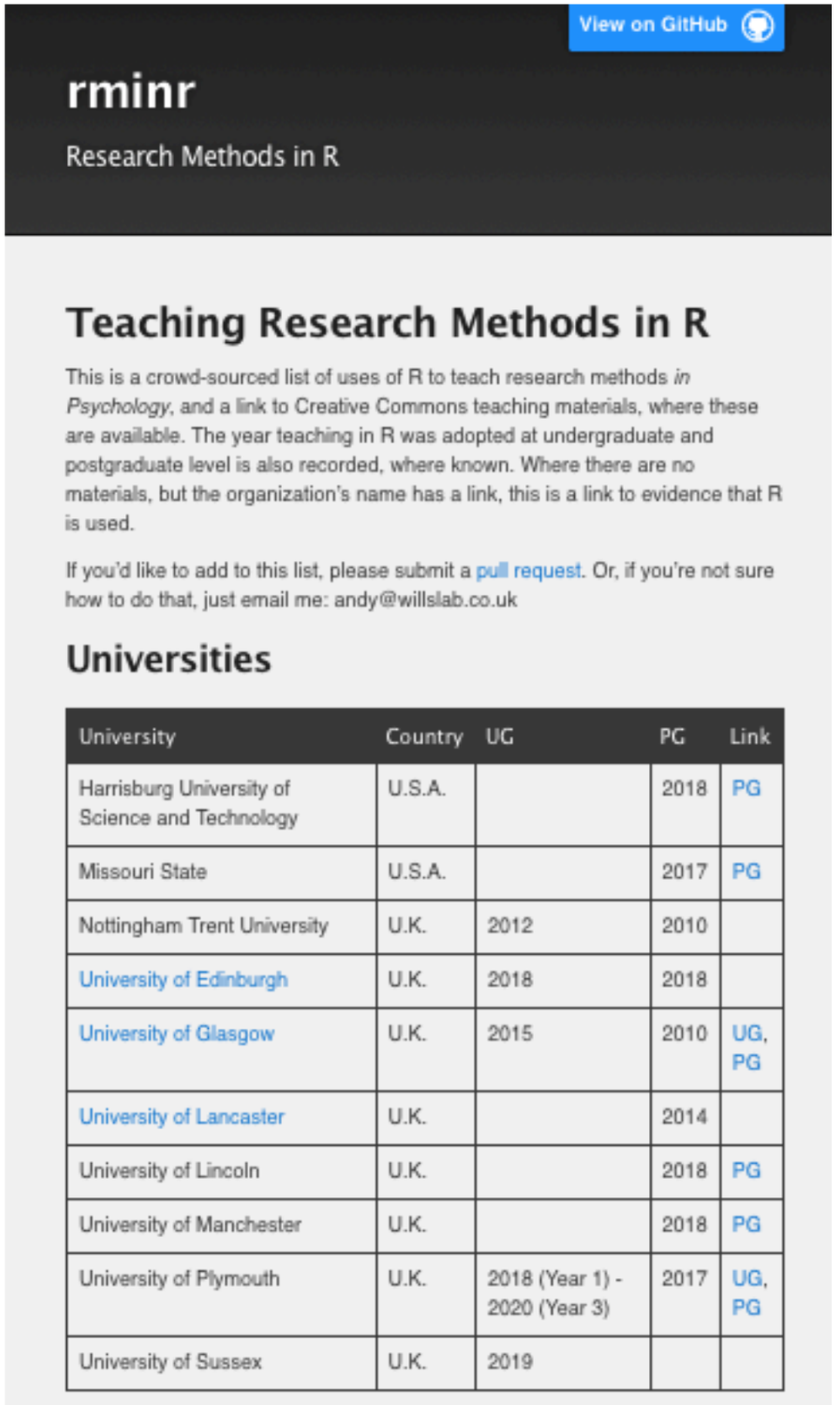
How to create BBC style graphics

At the BBC data team, we have developed an R package and an R cookbook to make the process of creating publication-ready graphics in our in-house style using R's ggplot2 library a more reproducible process, as well as making it easier for people new to R to create graphics.

The cookbook below should hopefully help anyone who wants to make graphics like these:



Handy list of Psychology groups that teach R, plus links to course materials - list compiled by Andy Wills at Plymouth.



View on GitHub

rminr

Research Methods in R

Teaching Research Methods in R

This is a crowd-sourced list of uses of R to teach research methods *in Psychology*, and a link to Creative Commons teaching materials, where these are available. The year teaching in R was adopted at undergraduate and postgraduate level is also recorded, where known. Where there are no materials, but the organization's name has a link, this is a link to evidence that R is used.

If you'd like to add to this list, please submit a pull request. Or, if you're not sure how to do that, just email me: andy@willslab.co.uk

Universities

University	Country	UG	PG	Link
Harrisburg University of Science and Technology	U.S.A.		2018	PG
Missouri State	U.S.A.		2017	PG
Nottingham Trent University	U.K.	2012	2010	
University of Edinburgh	U.K.	2018	2018	
University of Glasgow	U.K.	2015	2010	UG, PG
University of Lancaster	U.K.		2014	
University of Lincoln	U.K.		2018	PG
University of Manchester	U.K.		2018	PG
University of Plymouth	U.K.	2018 (Year 1) - 2020 (Year 3)	2017	UG, PG
University of Sussex	U.K.	2019		

https://ajwills72.github.io/rminr/rminrinpsy.html

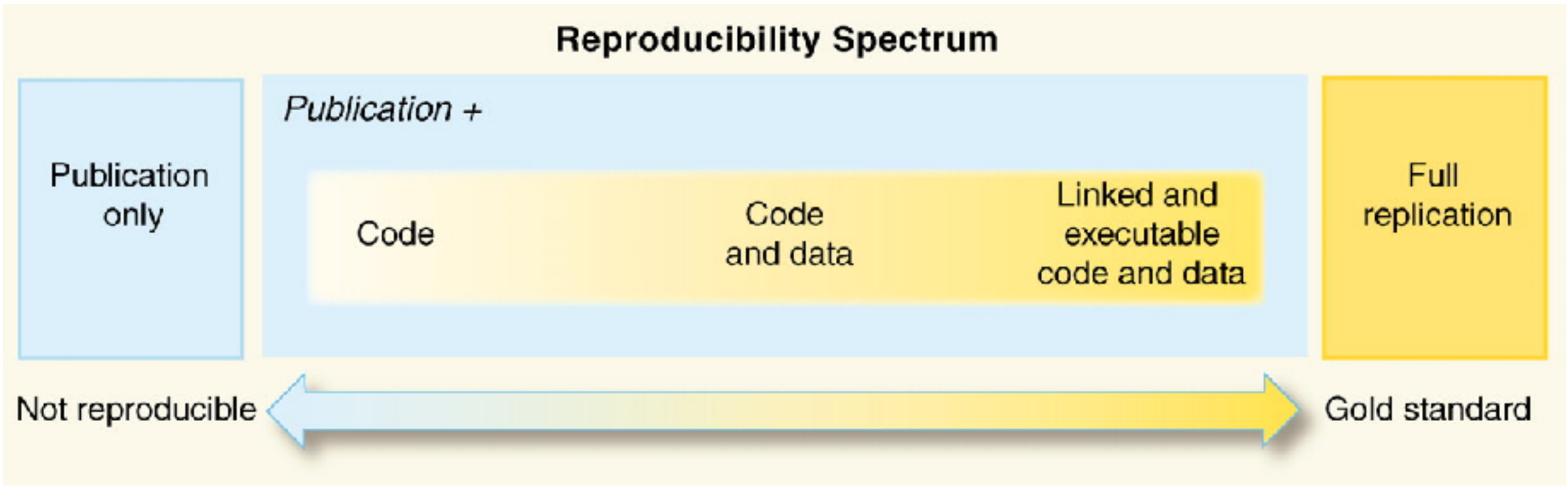
PERSPECTIVE

Reproducible Research in Computational Science

Roger D. Peng

+ See all authors and affiliations

Science 02 Dec 2011:
Vol. 334, Issue 6060, pp. 1226-1227
DOI: 10.1126/science.1213847



"*You can't do reproducible research in a GUI*", Hadley Wickham (probably!)

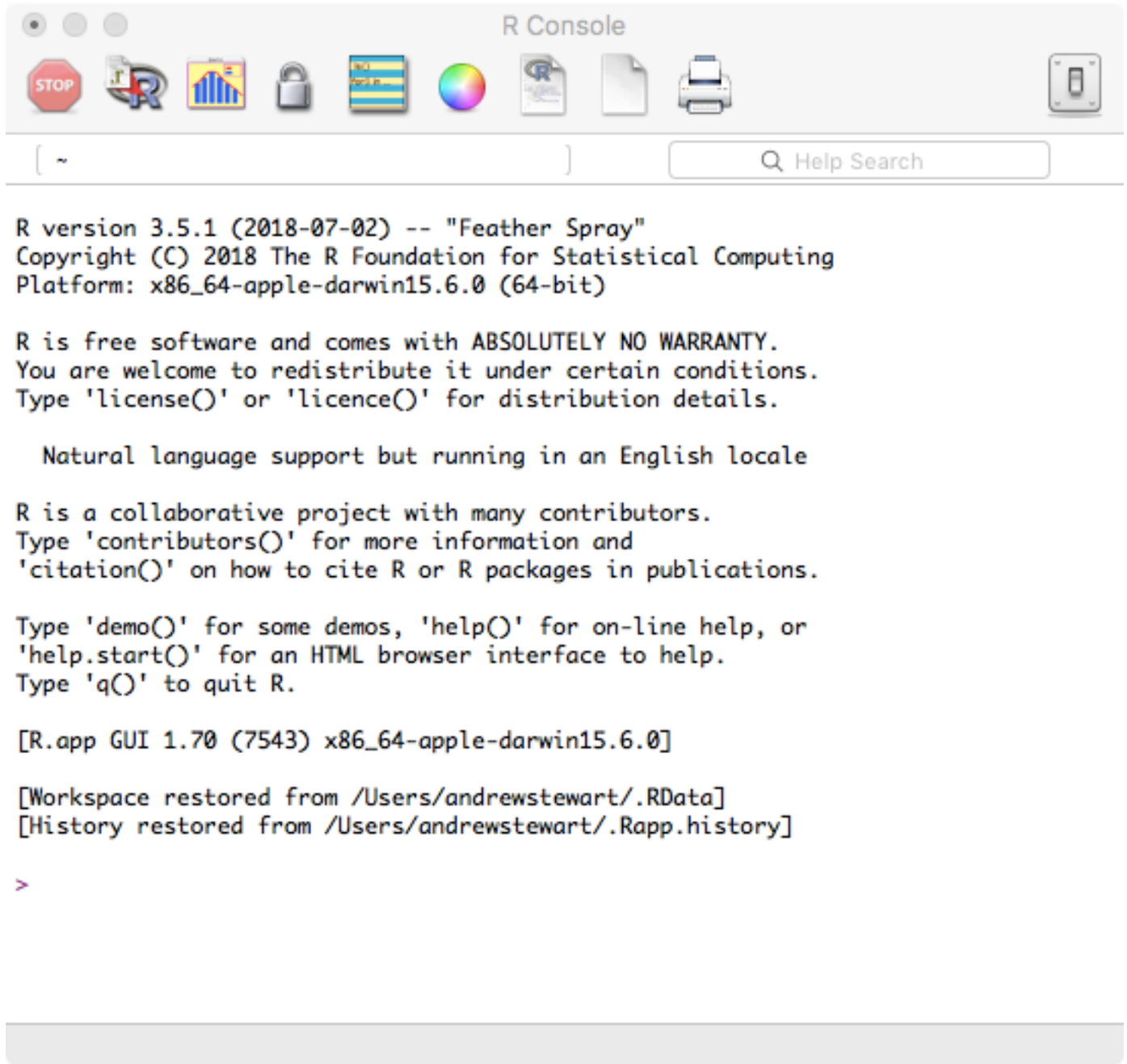
Discussion / Q & As about reproducibility

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

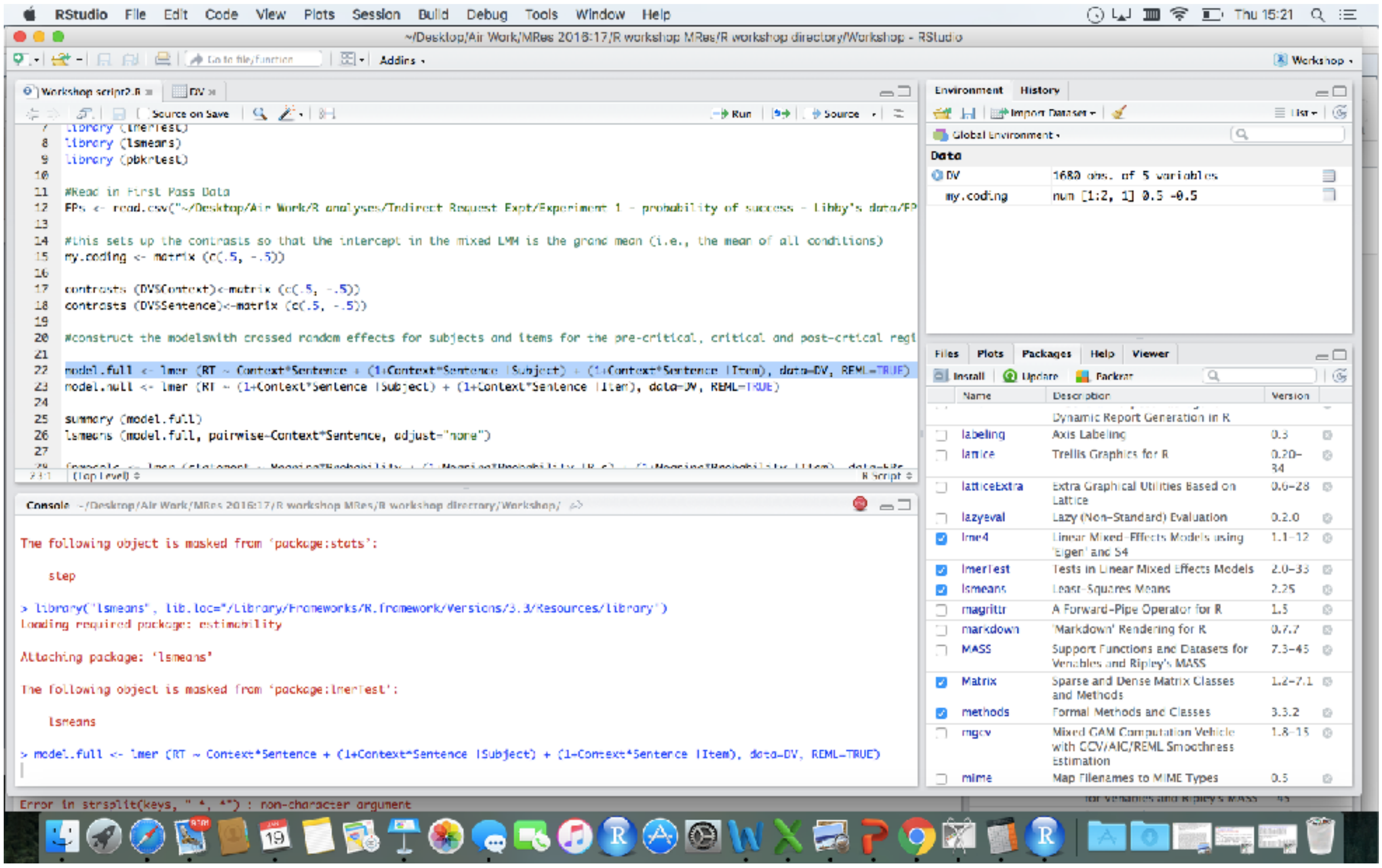
- If you load R directly it looks something like this:



- 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
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 models with crossed random effects for subjects and items for the pre-critical, critical and post-critical 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")
(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
Packages
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

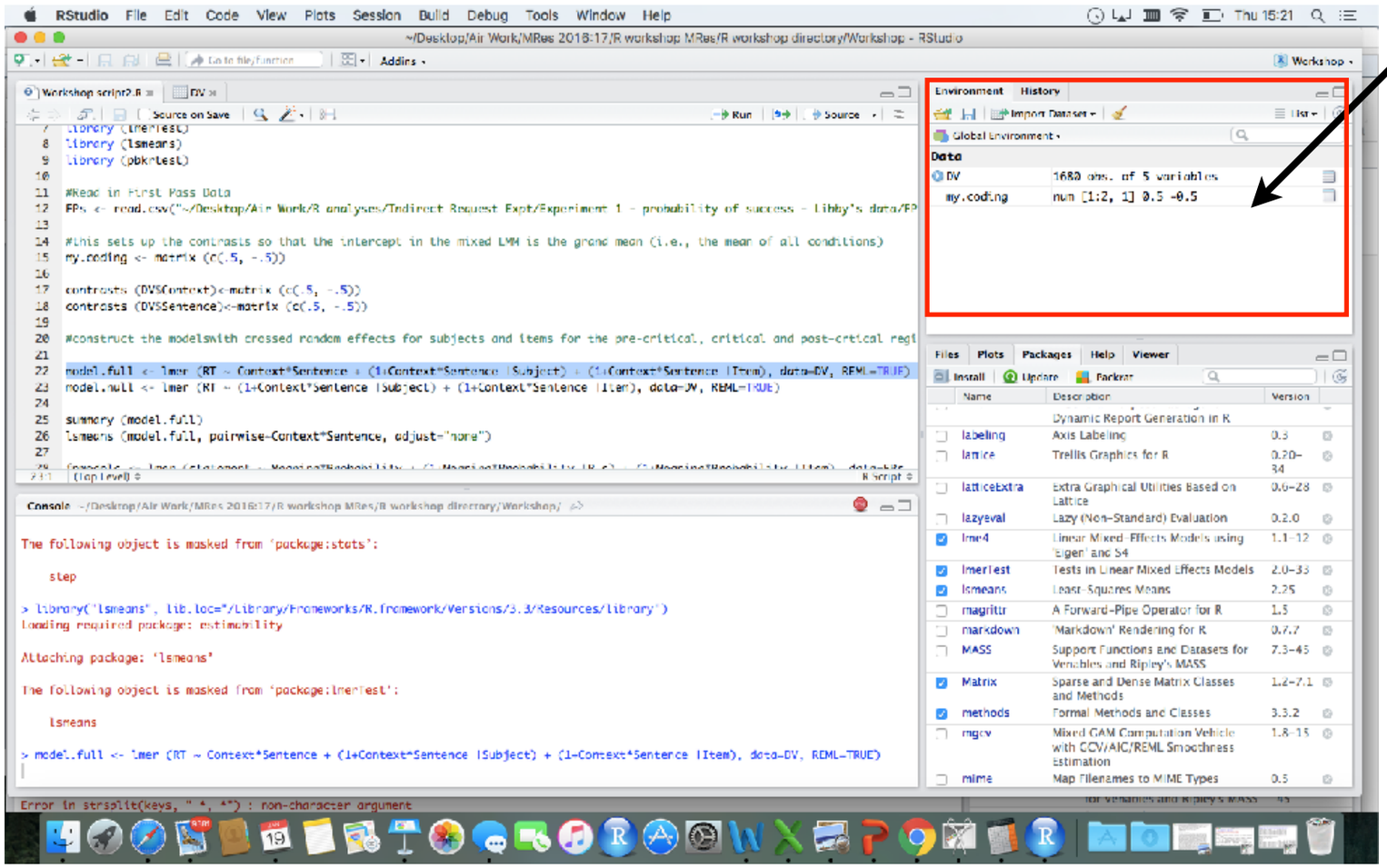
This is where you build your script and where data can be seen.

This contains information about your variables and open data sets.

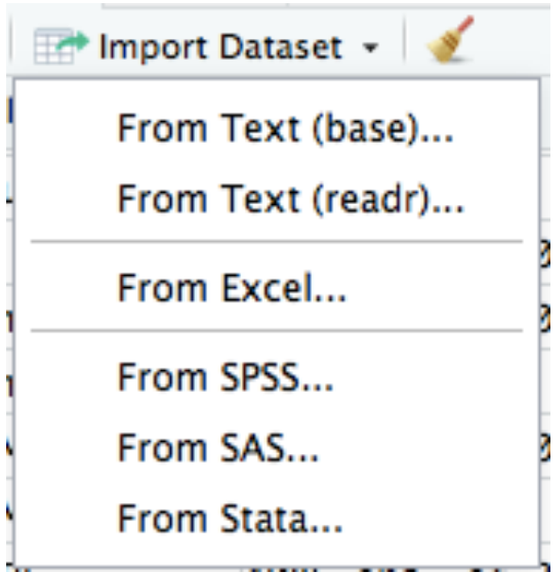
This is where you type commands.

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	503
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

Quote: Default

Locale: Configure...

Escape: None

Comment: Default

NA: Default

Code Preview:

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

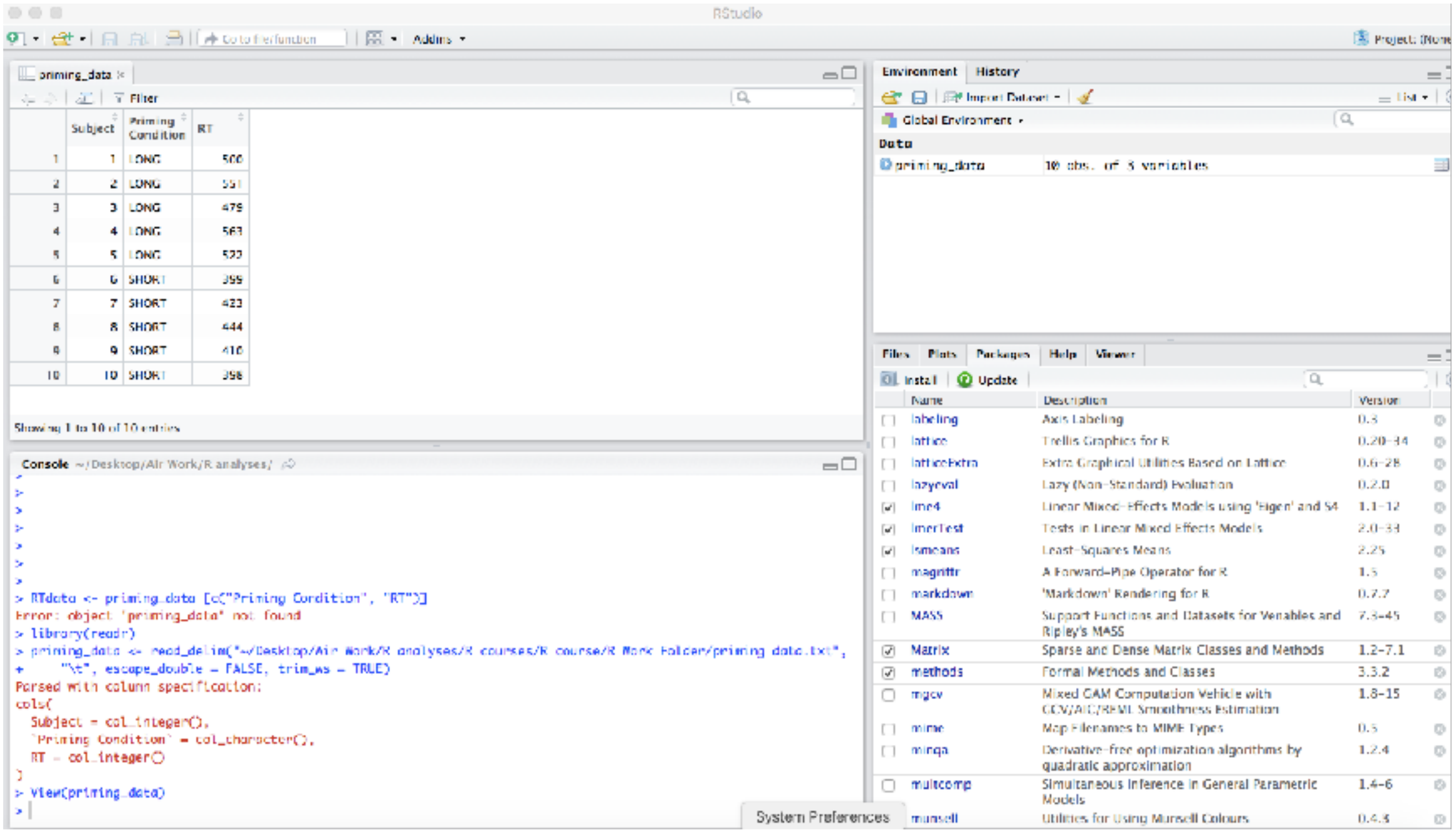
Import

Cancel

Can change here how the columns are delimited.

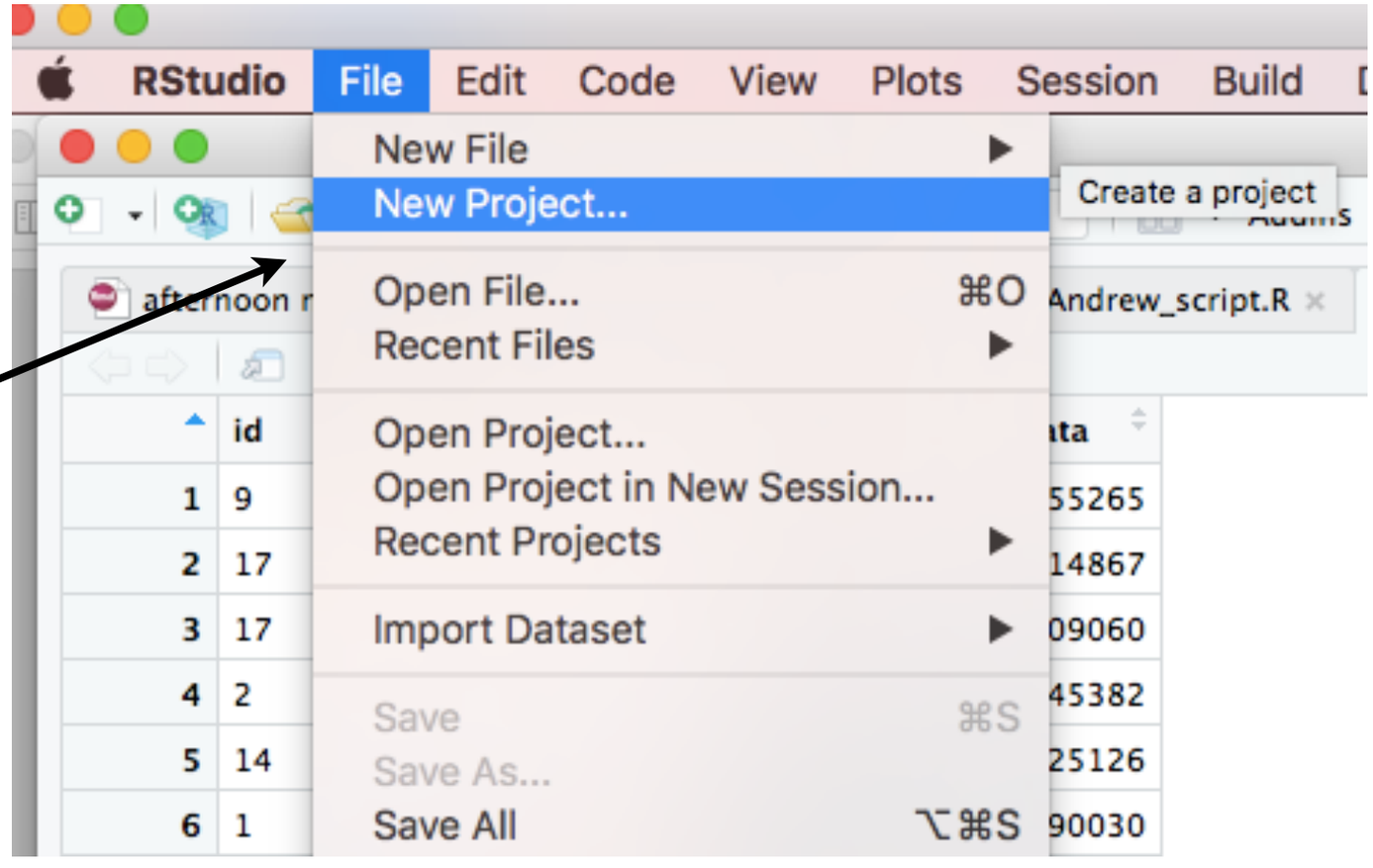
This is the code that corresponds to what you're getting RStudio to do.

- RStudio now looks like this:



When Starting R

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

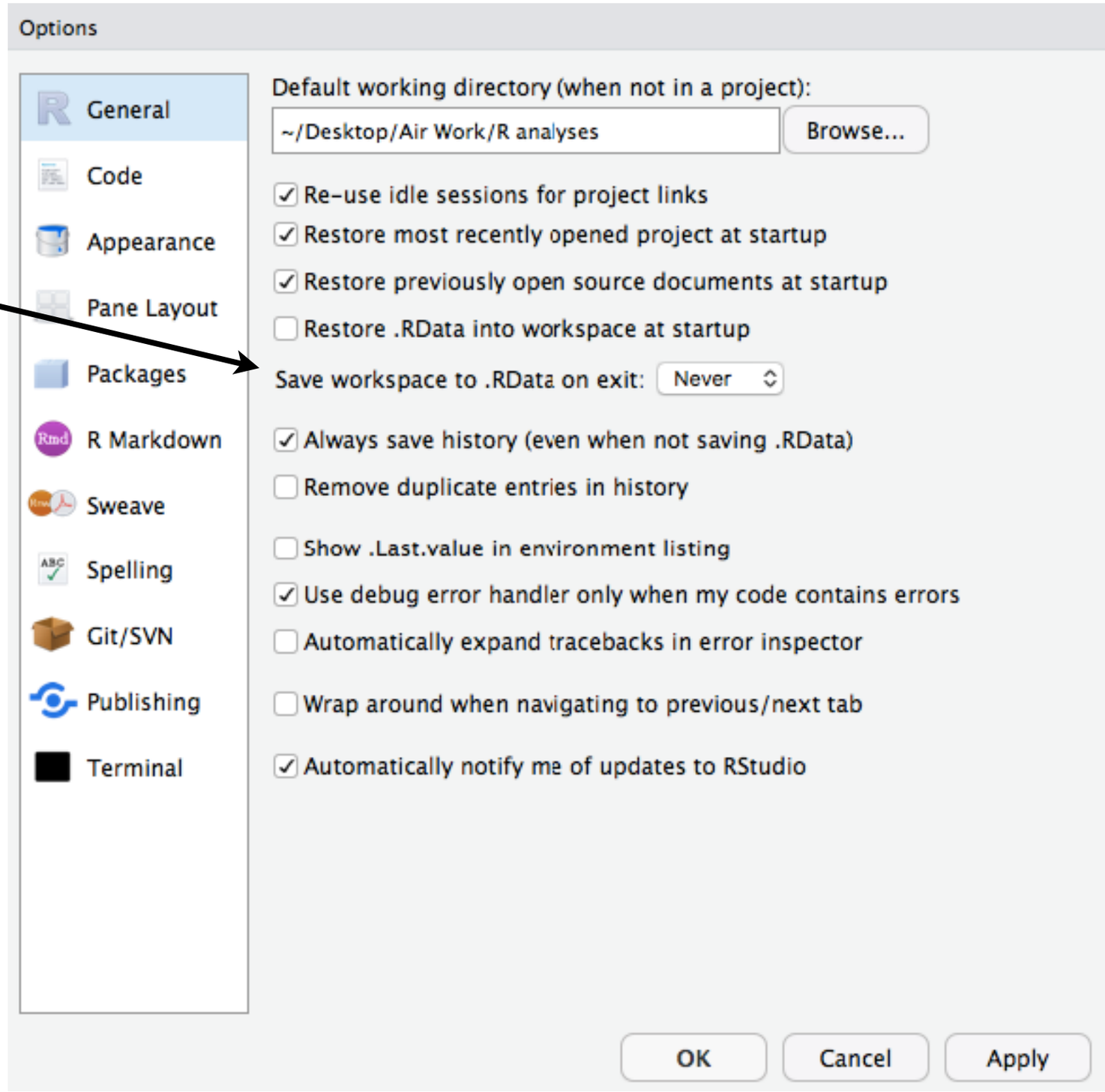


When you create an R Project in a folder, it's a good idea to keep that folder itself nicely organised:

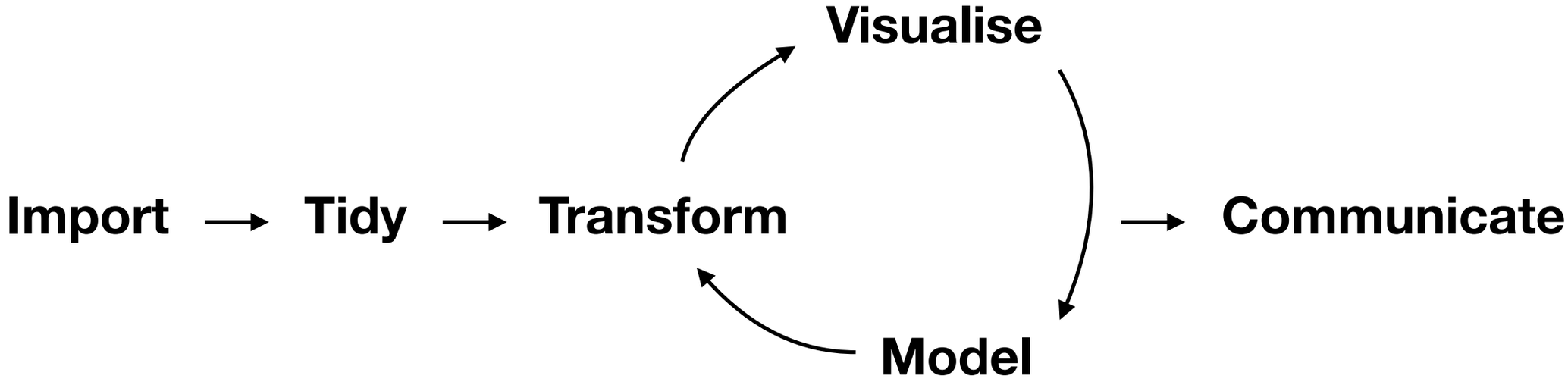
```
name_of_project
|--name_of_project.Rproj
|--raw_data
    |--data.csv
|--tidied_data
    |--tidy_data.csv
|--R_scripts
    |--data_tidy.R
    |--data_vis.R
    |--analysis.R
|--markdown_scripts
    |--analysis.Rmd
|--output_reports
    |--analysis.html
    |--analysis.pdf
```

When Starting R

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



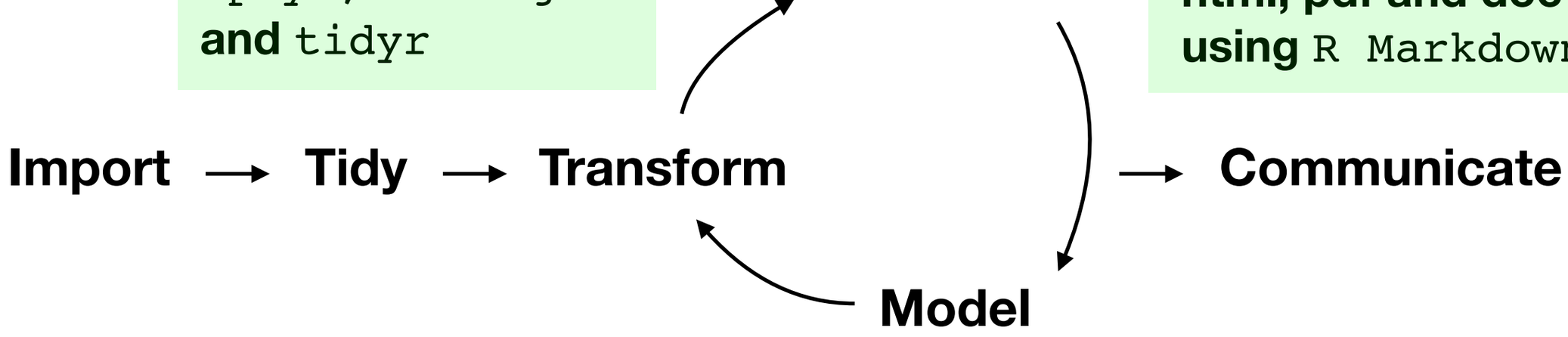
Analysis Workflow in the Tidyverse (Garrett Grolemund and Hadley Wickham) - from Data to Write-up



A Reproducible Workflow

Data vis using `ggplot()`

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



html, pdf and doc reports using `R Markdown`

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

Packages in R

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

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

```
> library(packagename)
```

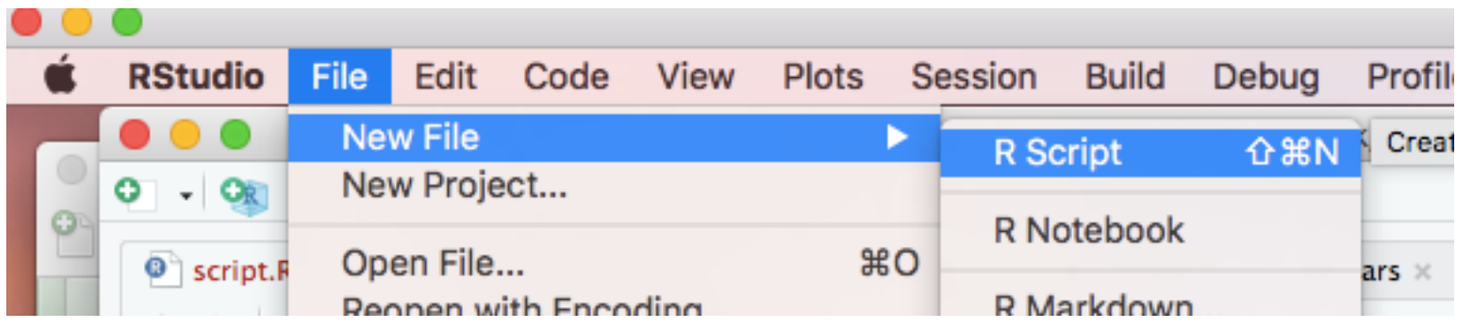
Every time you start R afresh, you need to load the packages you will use by using the `library` function. If you see notation like the following, it means we're using a function (e.g., `summarise`) from within a package (e.g., `dplyr`)

```
dplyr::summarise
```

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

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

Writing Scripts in R



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

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

Writing Scripts in R

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

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

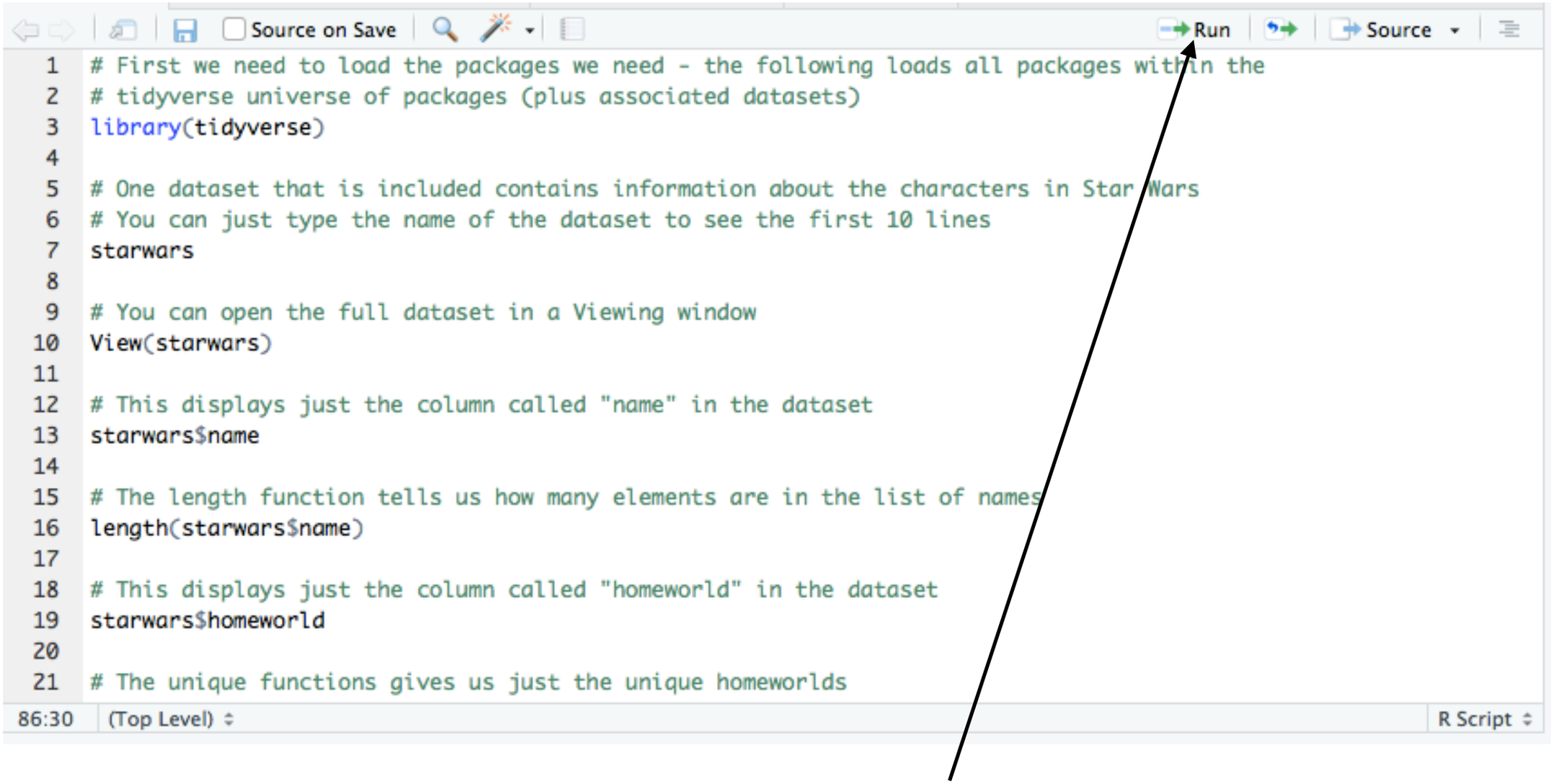
If you aren't sharing your data and code, you aren't engaged in generating reproducible or open research…

Source on Save | Run | Source

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

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

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

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

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

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

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

86:30 (Top Level) R Script

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



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

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 - consider using snake_case or CamelCase, but never use whitespace in variable names:

`age - Good`

`A1 - Bad`

`expt1_data - Good`

`Expiriment1datawithoutliersremoved - Bad`

Style Guide

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

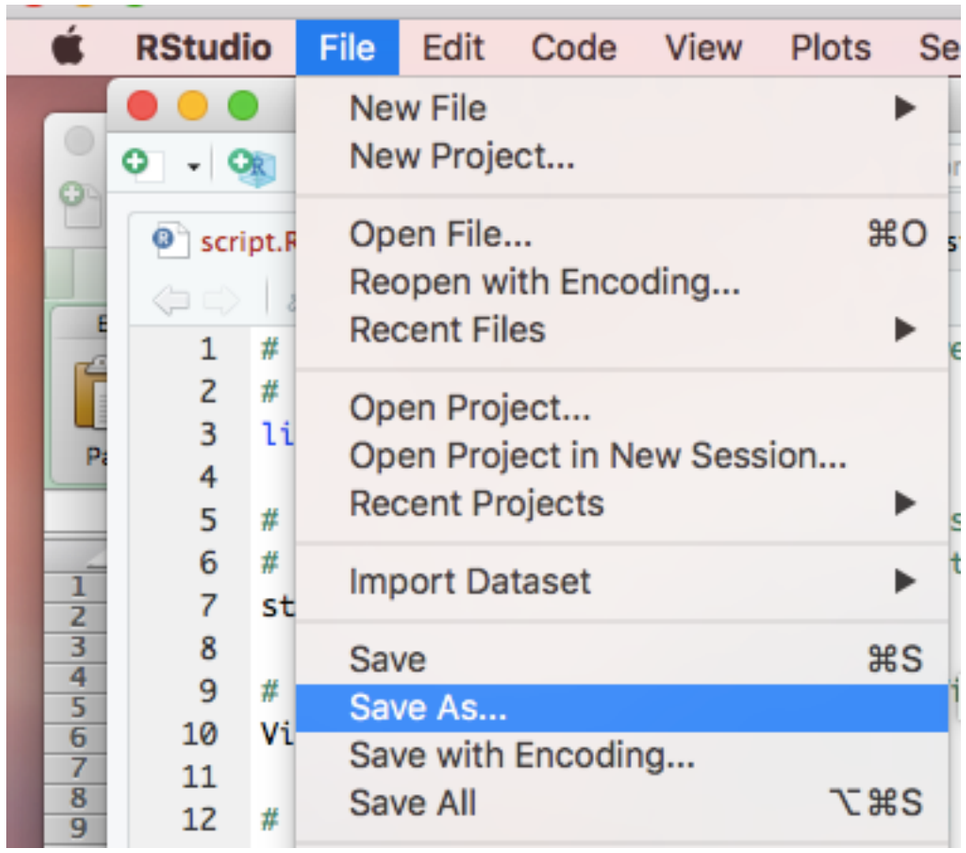
Whitespace used sensibly in your script makes your code easier to read. Separate discrete sections of code in your script with a blank line.

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, OSF or (even better) build a Binder.

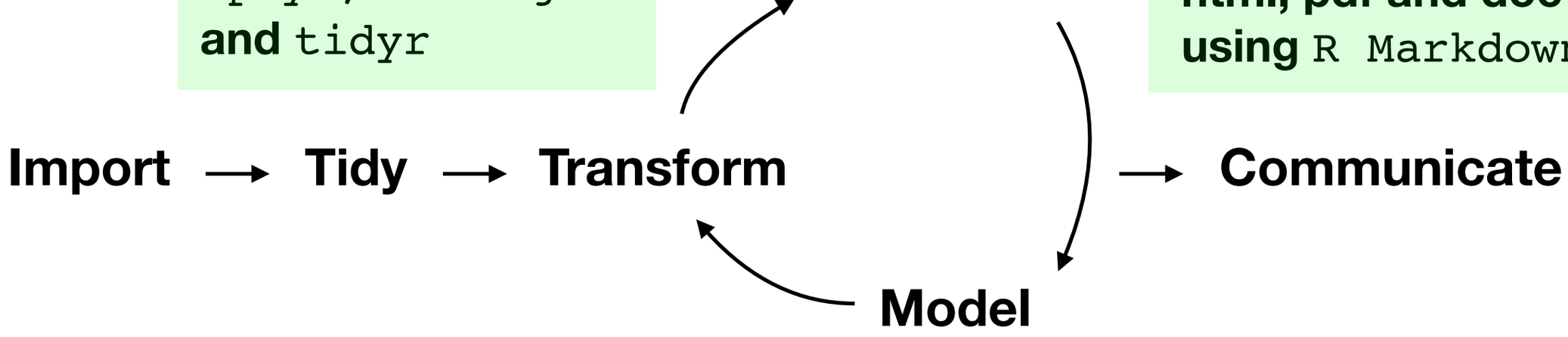
Git is a powerful tool that allows for the version control in collaborative projects, and the sharing of code and projects.

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

A Reproducible Workflow

Data vis using `ggplot()`

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

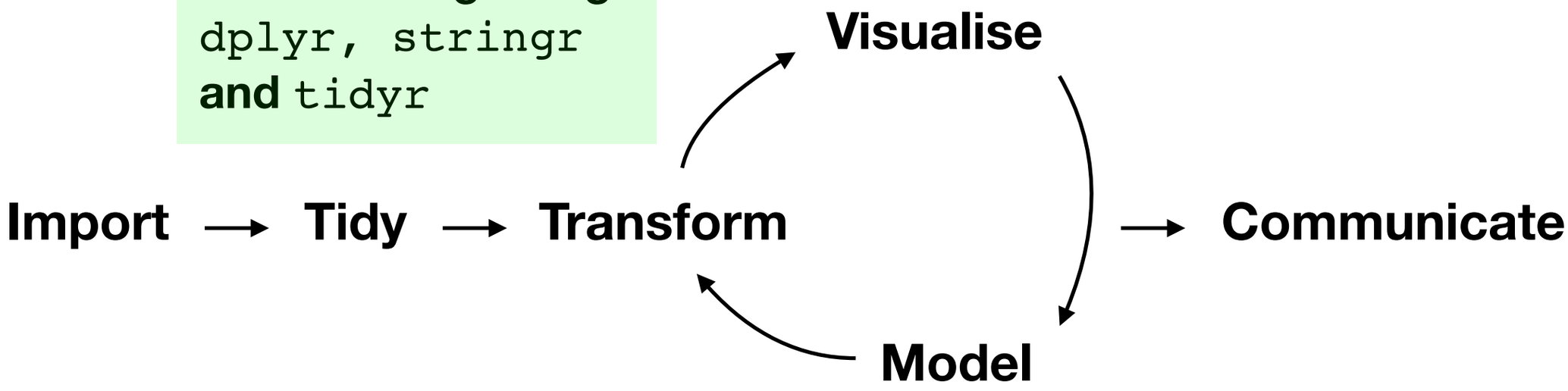


html, pdf and doc reports using `R Markdown`

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

A Reproducible Workflow

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



Let's get ready to code...

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

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

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

On the first line of your script type:

```
library(tidyverse)
```

Then run the script...

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

```
data1 <- read_csv("https://bit.ly/2OTkuNz")

dataRT <- read_csv("https://bit.ly/31xd6sG")
```

Tidying and Transforming Data

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

If we type into the Console:

```
> data1
```

the data frame is displayed like this…

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

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

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

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

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

```
>?str
```

or

```
>help(str)
```

in the Console window.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

id	condition	rt
…	…	…

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

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

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

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

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

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

```
> data_wide <- spread(data_long, "condition", "rt")
> data_wide
# A tibble: 48 x 3
      id complex_sentence simple_sentence
   <int>            <int>           <int>
 1    95             2440            1960
 2   400             2200            2186
 3   457             2503            1797
 4  1138             2441            2154
 5  1587             2395            1936
 6  1805             2560            1864
 7  1864             2540            1930
 8  2006             2267            2230
 9  2183             2324            1857
10  2318             2739            1918
# … with 38 more rows
```

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

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

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

Generating Descriptives - using `dplyr`

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

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

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

Tidying Up Some Real World Messy Data

Let's load another dataset that I created:

```
my_data <- read_csv("https://bit.ly/2YYcP4B")
```

Tidying Up Some Real World Messy Data

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

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

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

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

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

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

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

- Now our data frame looks like this:

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

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

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

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

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

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

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

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

Or using the pipe…

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

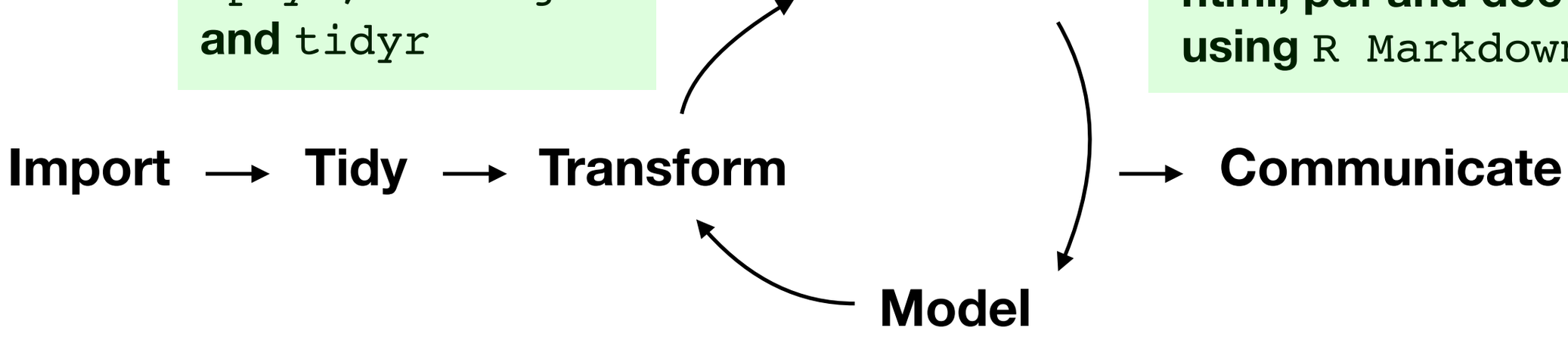
```
my_data %>%
   unite(col = "condition", c("prime", "target"), sep = "_") %>%
   spread(key = "condition", value = "rt")
```

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

A Reproducible Workflow

Data vis using `ggplot()`

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

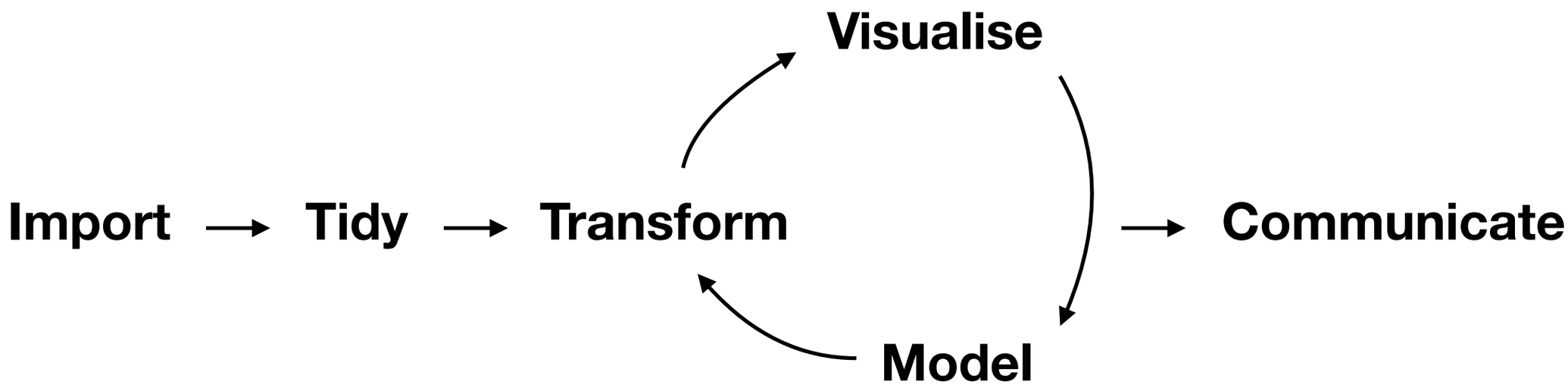


html, pdf and doc reports using `R Markdown`

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

A Reproducible Workflow

Data vis using `ggplot()`

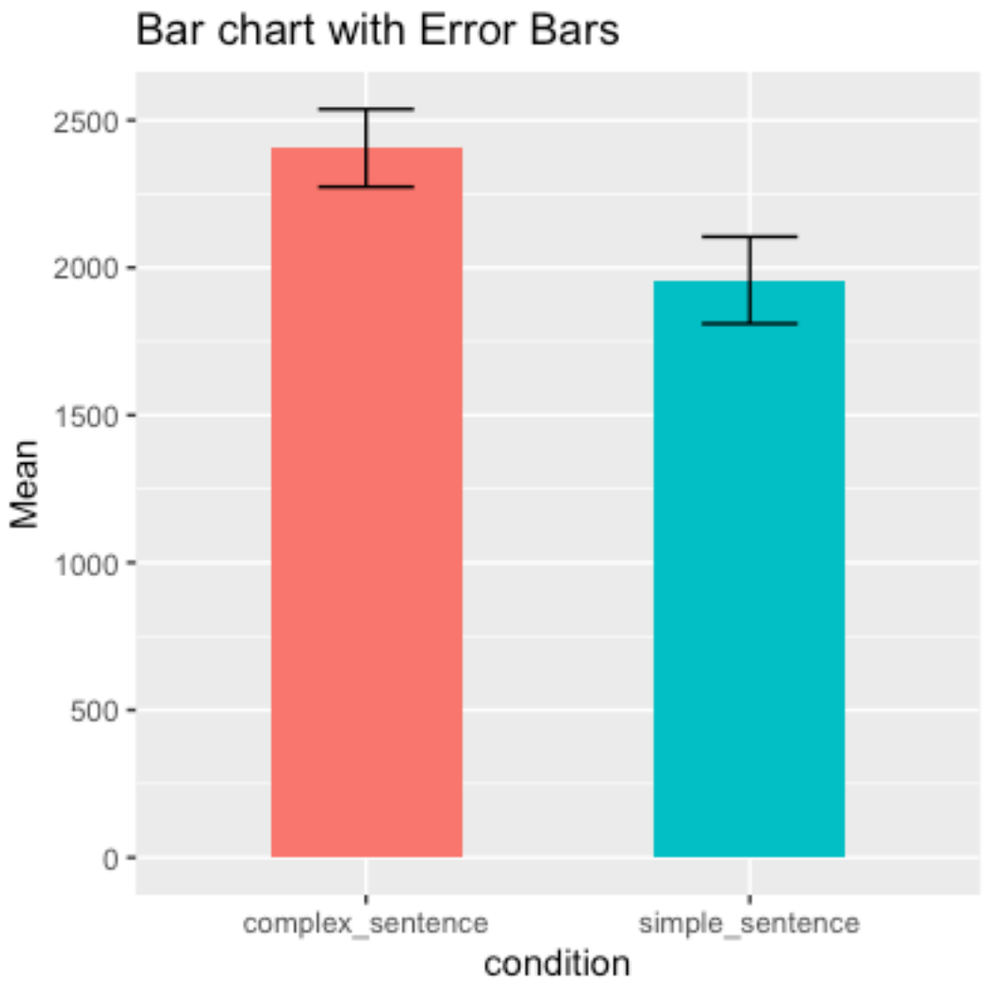


Visualising Your Data

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

```
> library(ggplot2)
```

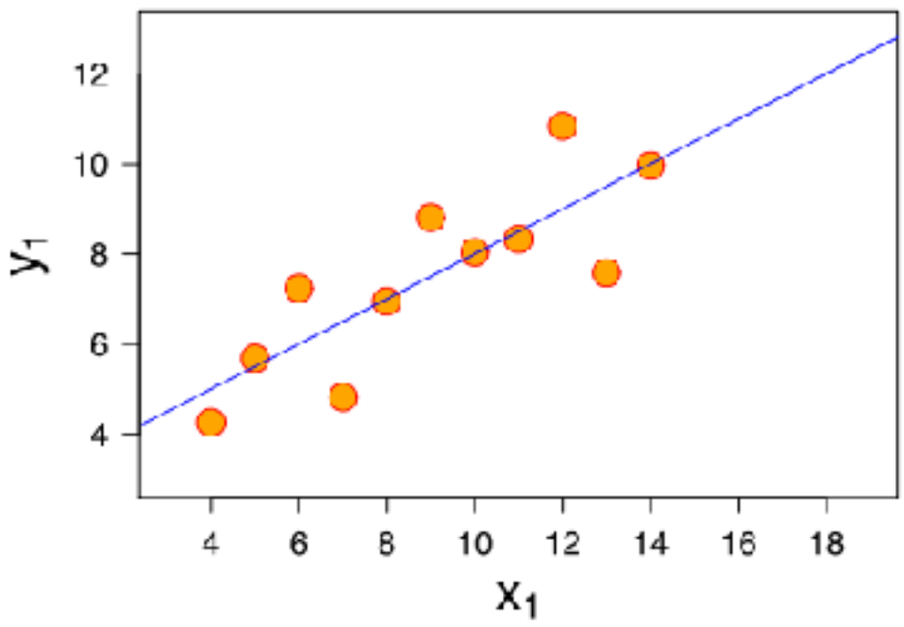
Bar Graphs (bleurgh!)

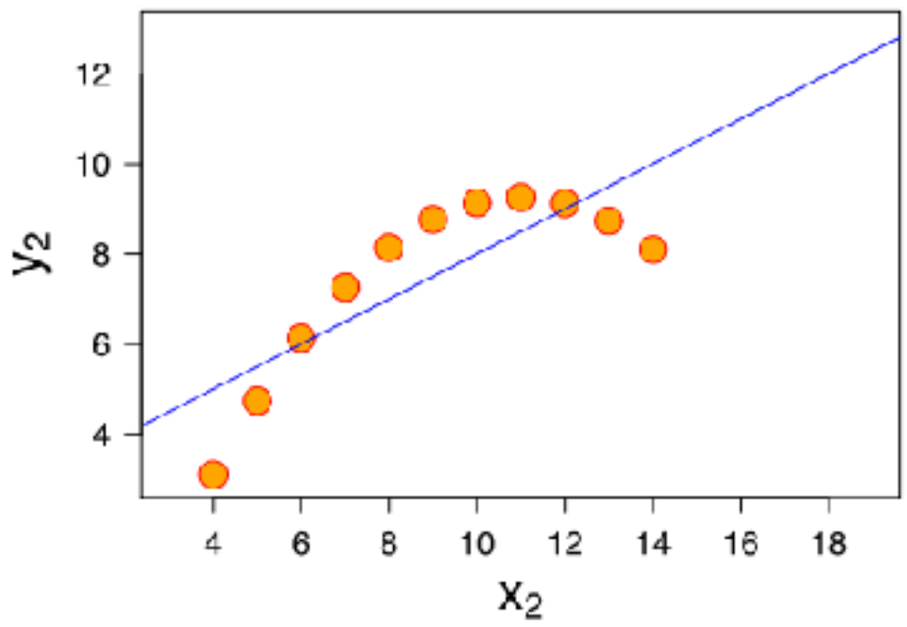


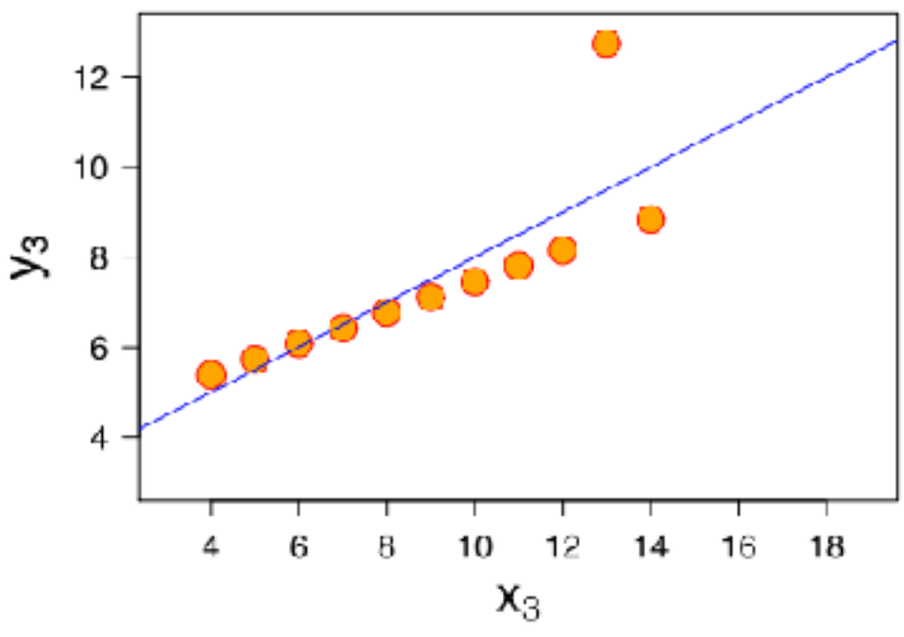
Bar graphs tend to be quite limited in terms of what they communicate. Here they communicate the means for levels of a factor and information about variance. But they don't tell us anything about the *distribution* of the data.

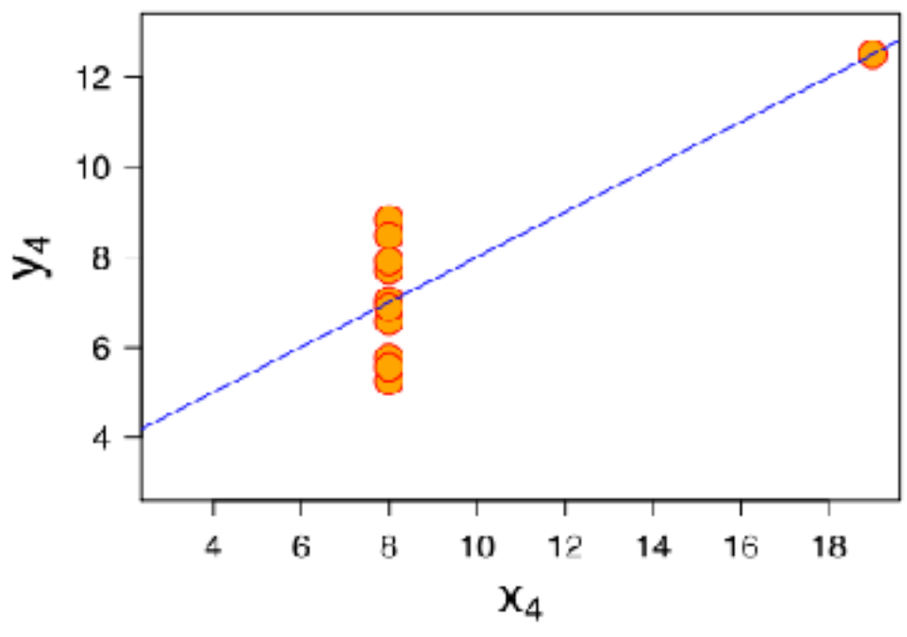
```
data_summ <- data_long %>% group_by(condition) %>% summarise(Mean = mean(rt), sd = sd(rt))

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

Anscombe's Quartet

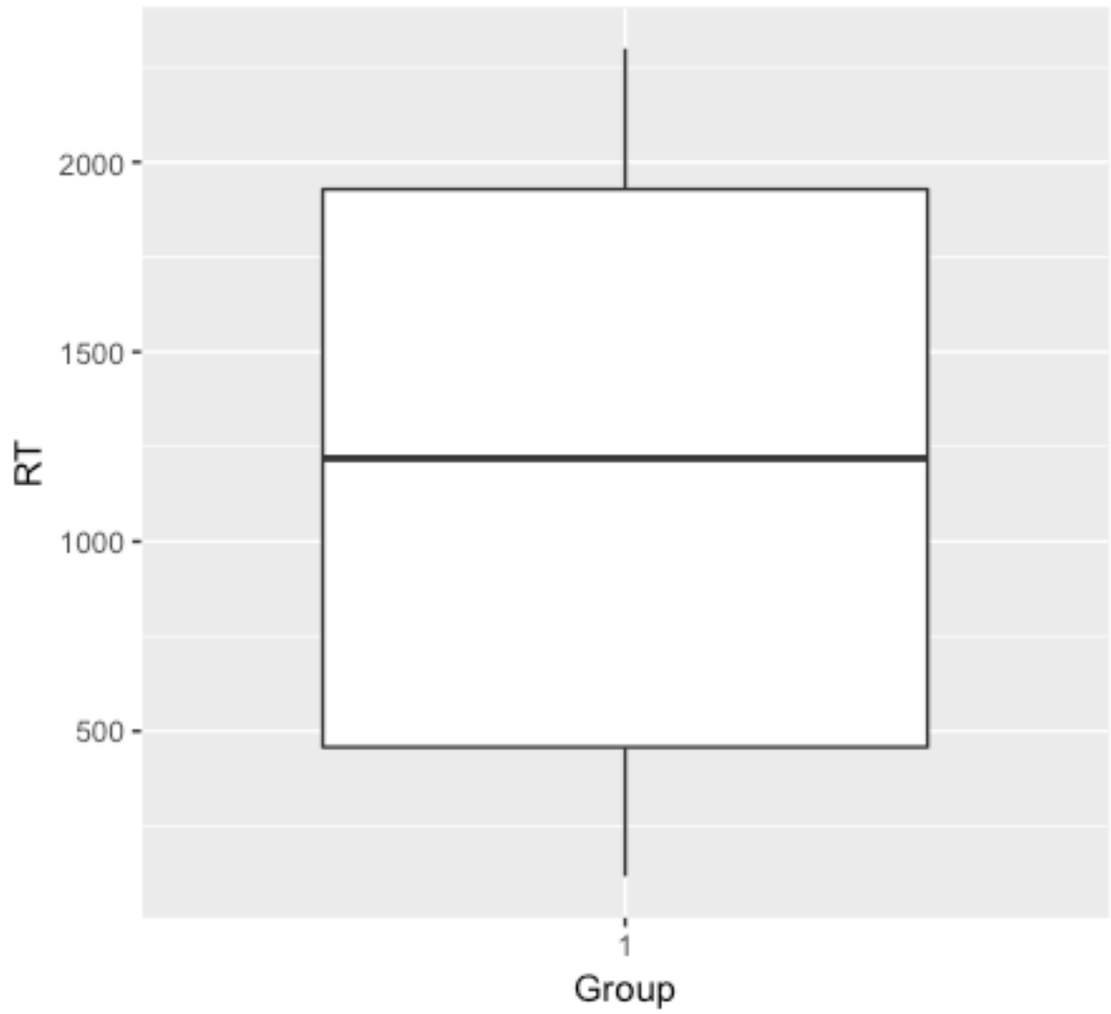






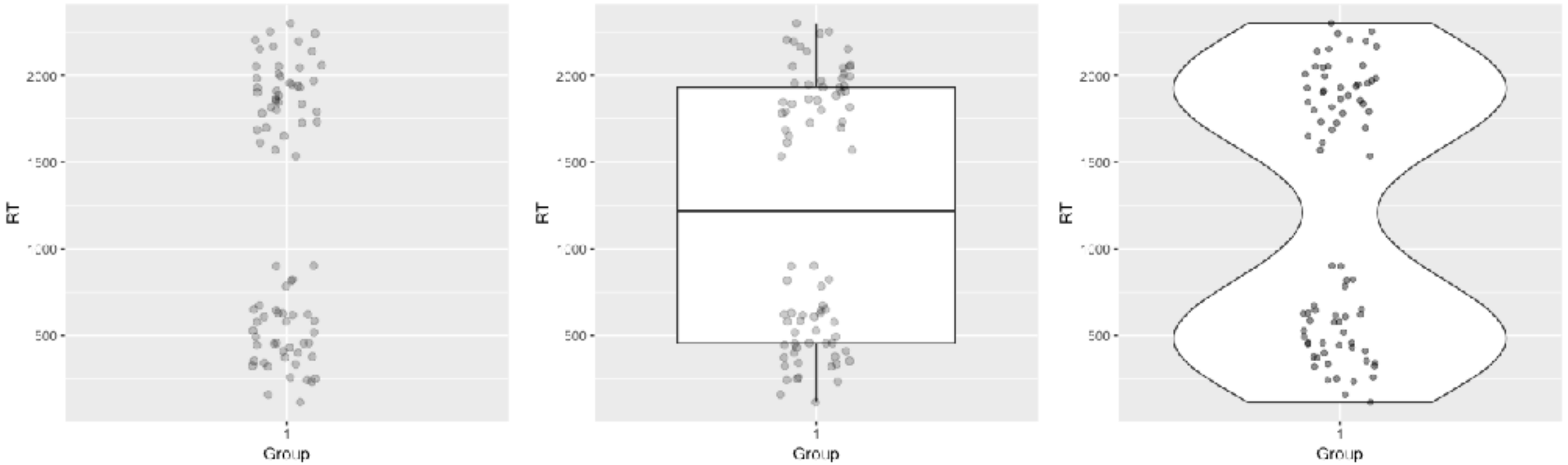


Plots Based on Aggregated Data Can Mislead…



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

But look more closely at the actual data…

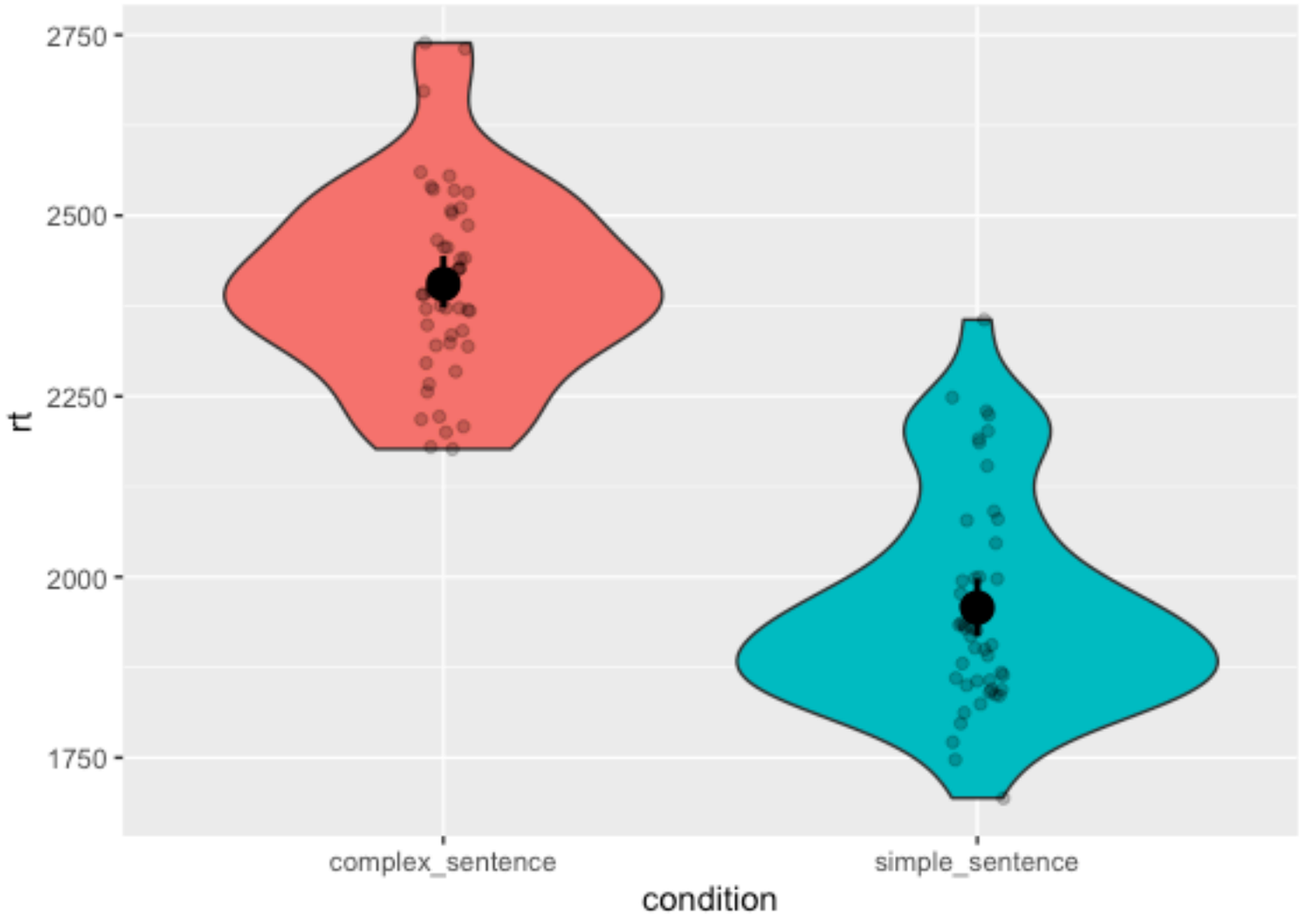


The data are clearly bimodal with no actual data point near the mean. **Distribution shape matters** and we need to capture that in our data visualisations.

You try...

```
data2 <- read_csv("https://bit.ly/2YYr6dm")

ggplot(data2, aes(x = group, y = rt)) +
   geom_boxplot()

ggplot(data2, aes(x = group, y = rt)) +
   geom_jitter(size = 2, width = .1, alpha = .25)

ggplot(data2, aes(x = group, y = rt)) +
   geom_boxplot() +
   geom_jitter(size = 2, width = .1, alpha = .25)

ggplot(data2, aes(x = group, y = rt)) +
    geom_violin() +
    geom_jitter(width = .1, alpha = .5)
```

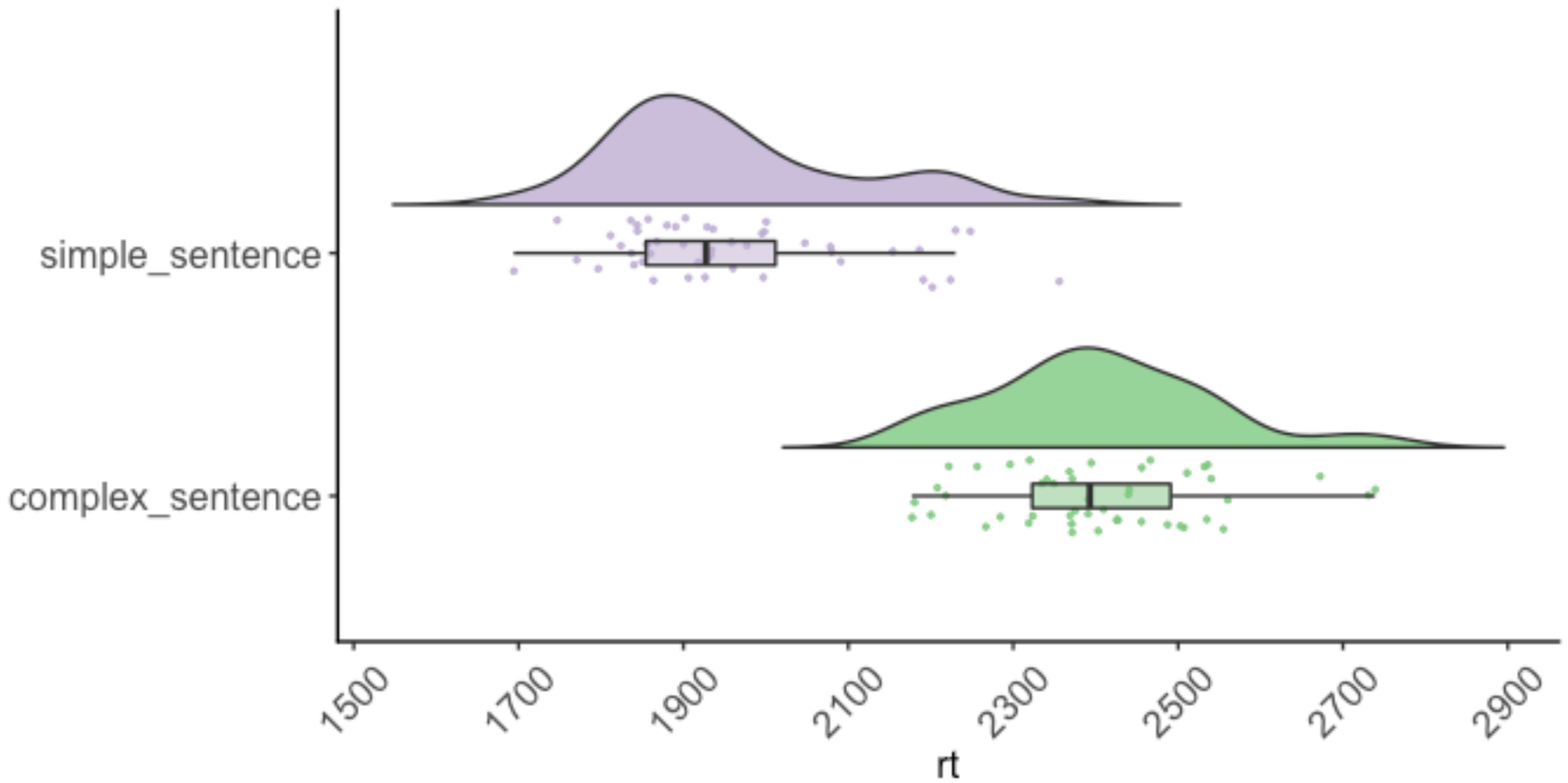
Violin Plots



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.)

A Variety of Plots Using the Same Dataset

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

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

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

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

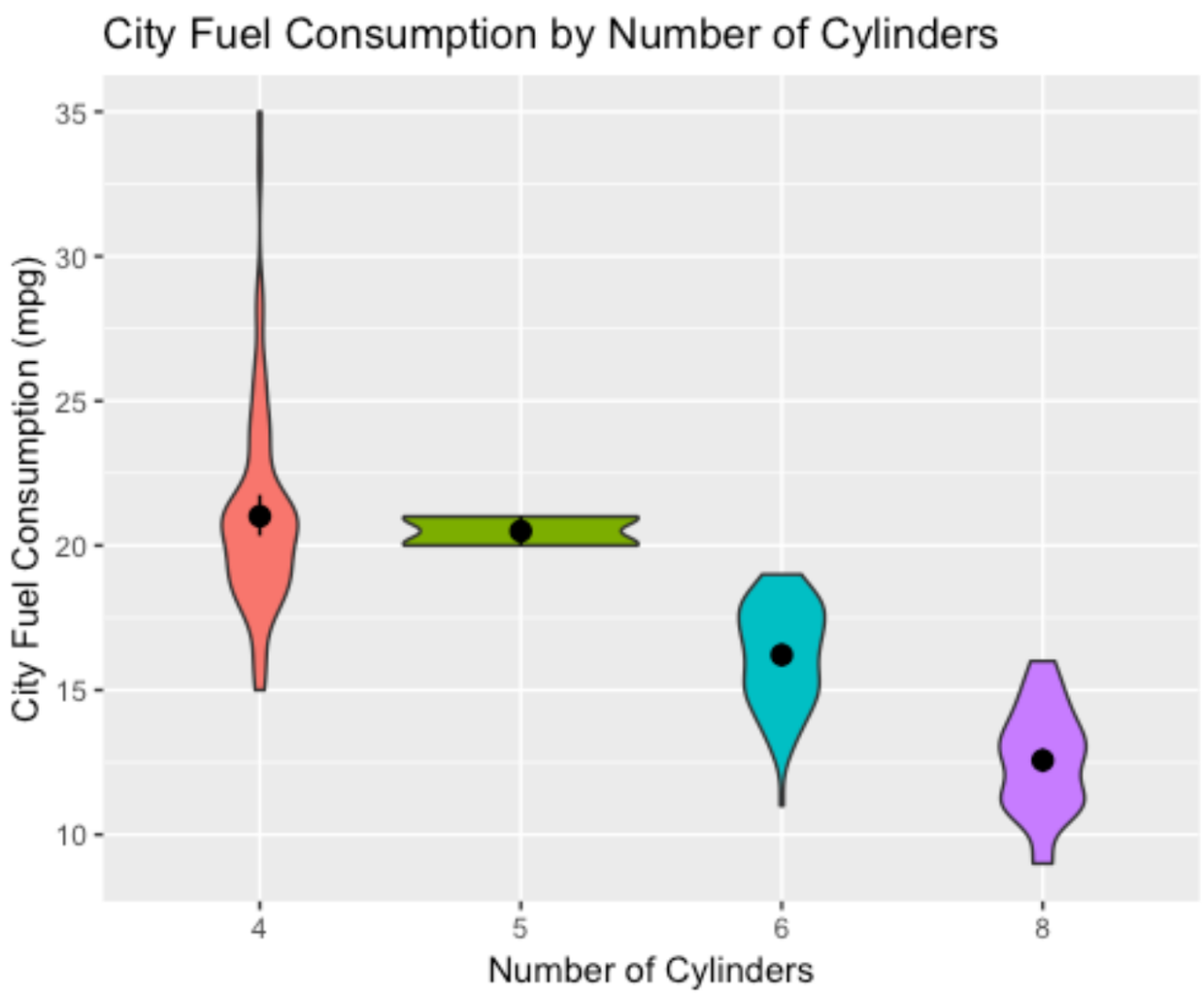
We can use the `length` function to give us the total number of unique possibilities:

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

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

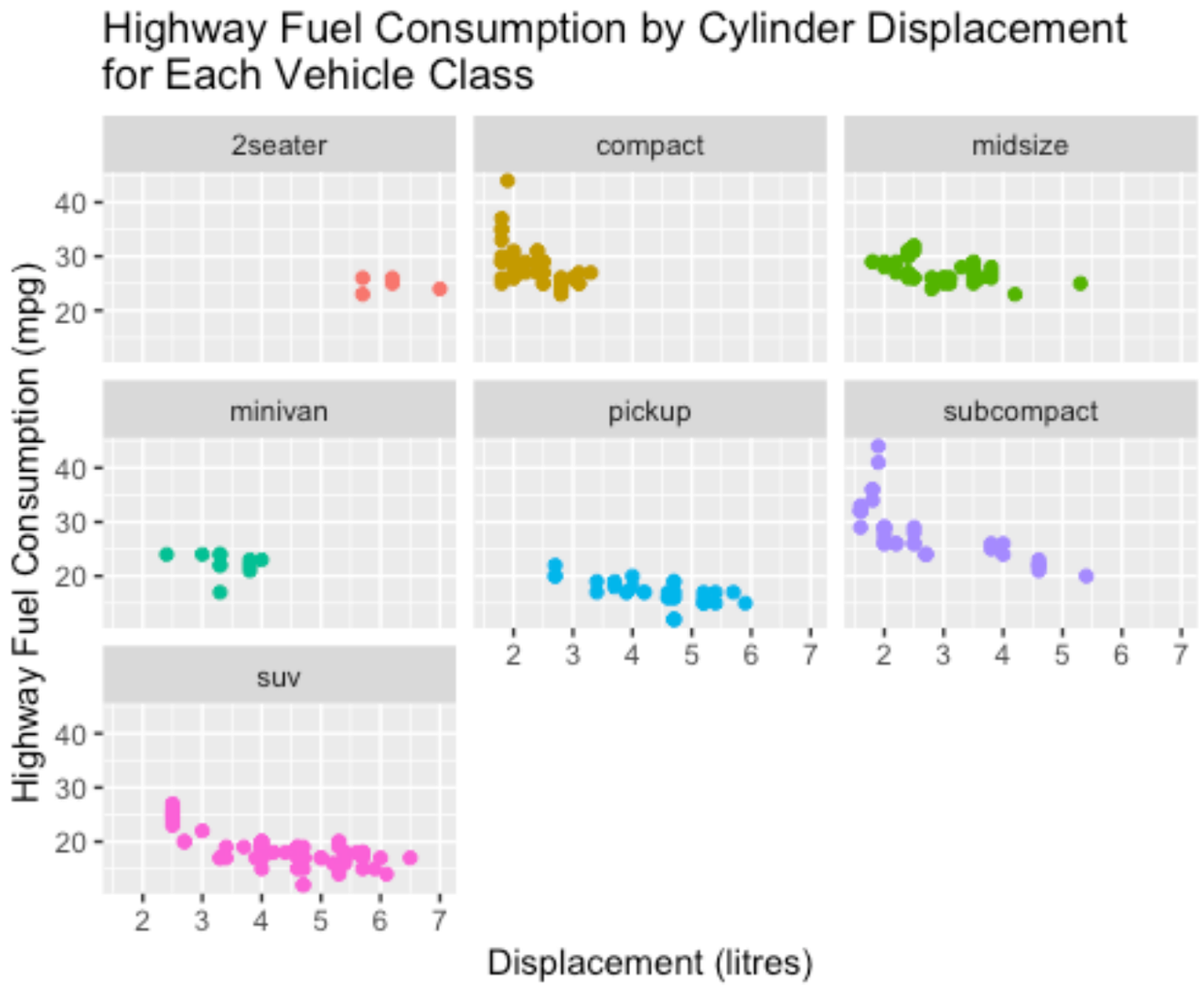
This illustrates the idea that there is not one 'correct' way to visualise the data, but rather that your choice of visualisation will be influenced by the question you're investigating, or the story you're wanting to tell…

Violin Plots



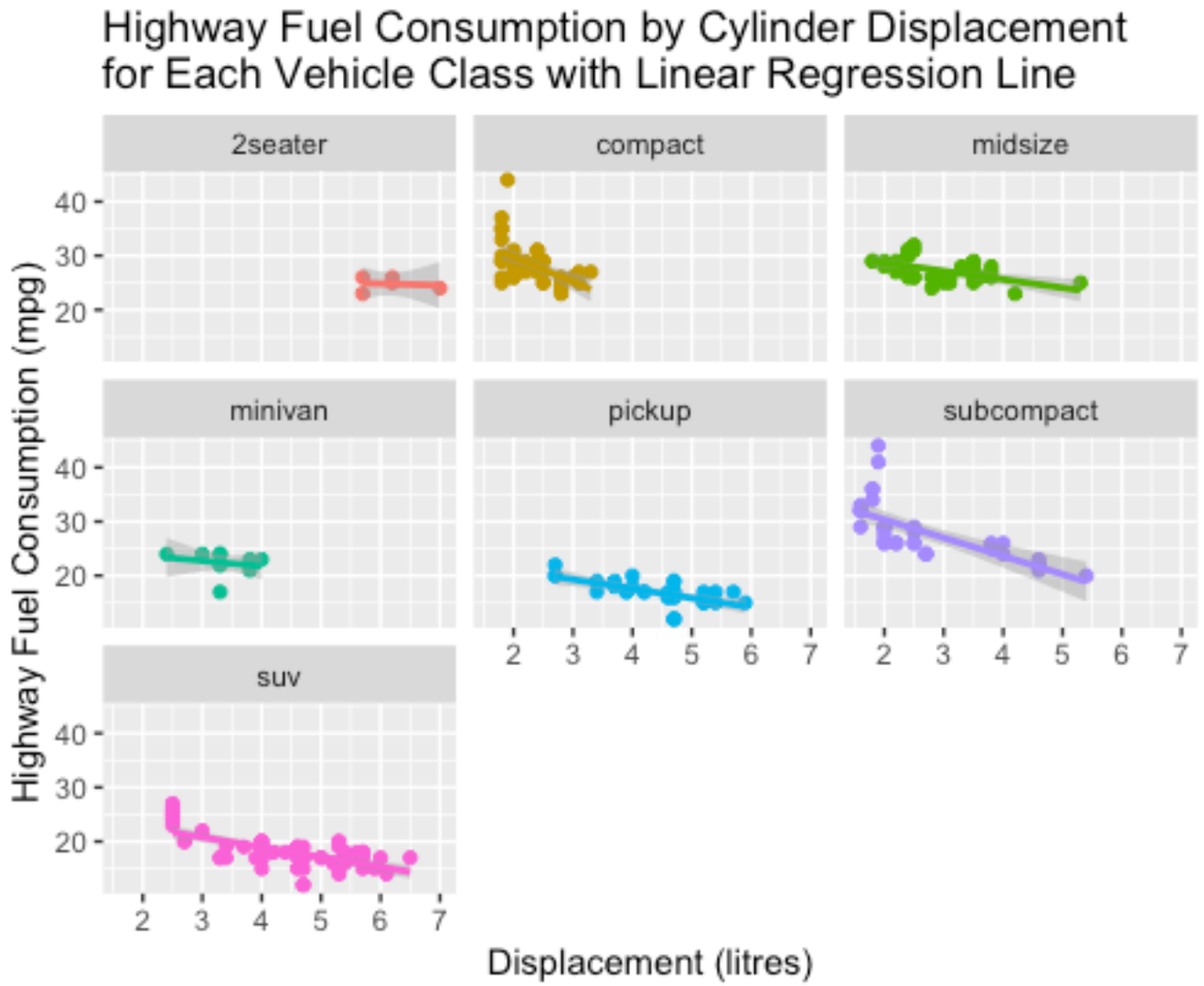

```
ggplot(mpg, aes(x = factor(cyl), y = cty, fill = factor(cyl))) +
  geom_violin() +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary(fun.data = mean_cl_boot, colour = "black", size = .5) +
  labs(title = "City Fuel Consumption by Number of Cylinders",
       x = "Number of Cylinders",
    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

Faceting

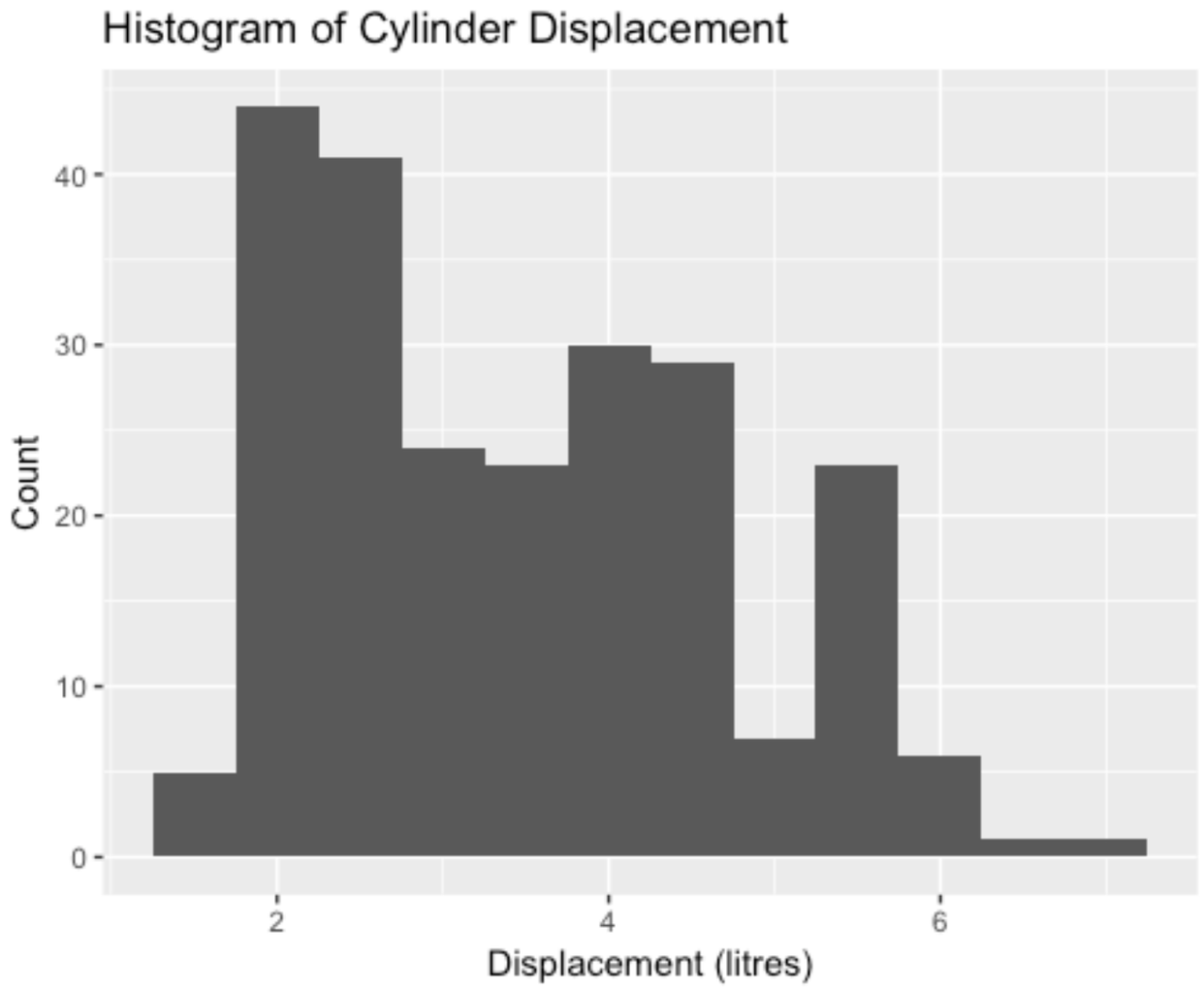



```
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_point() +
  facet_wrap(~ class) +
  guides(colour = FALSE) +
  labs(title = "Highway Fuel Consumption by Cylinder Displacement \nfor Each Vehicle Class",
       x = "Displacement (litres)",
       y = "Highway Fuel Consumption (mpg)")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

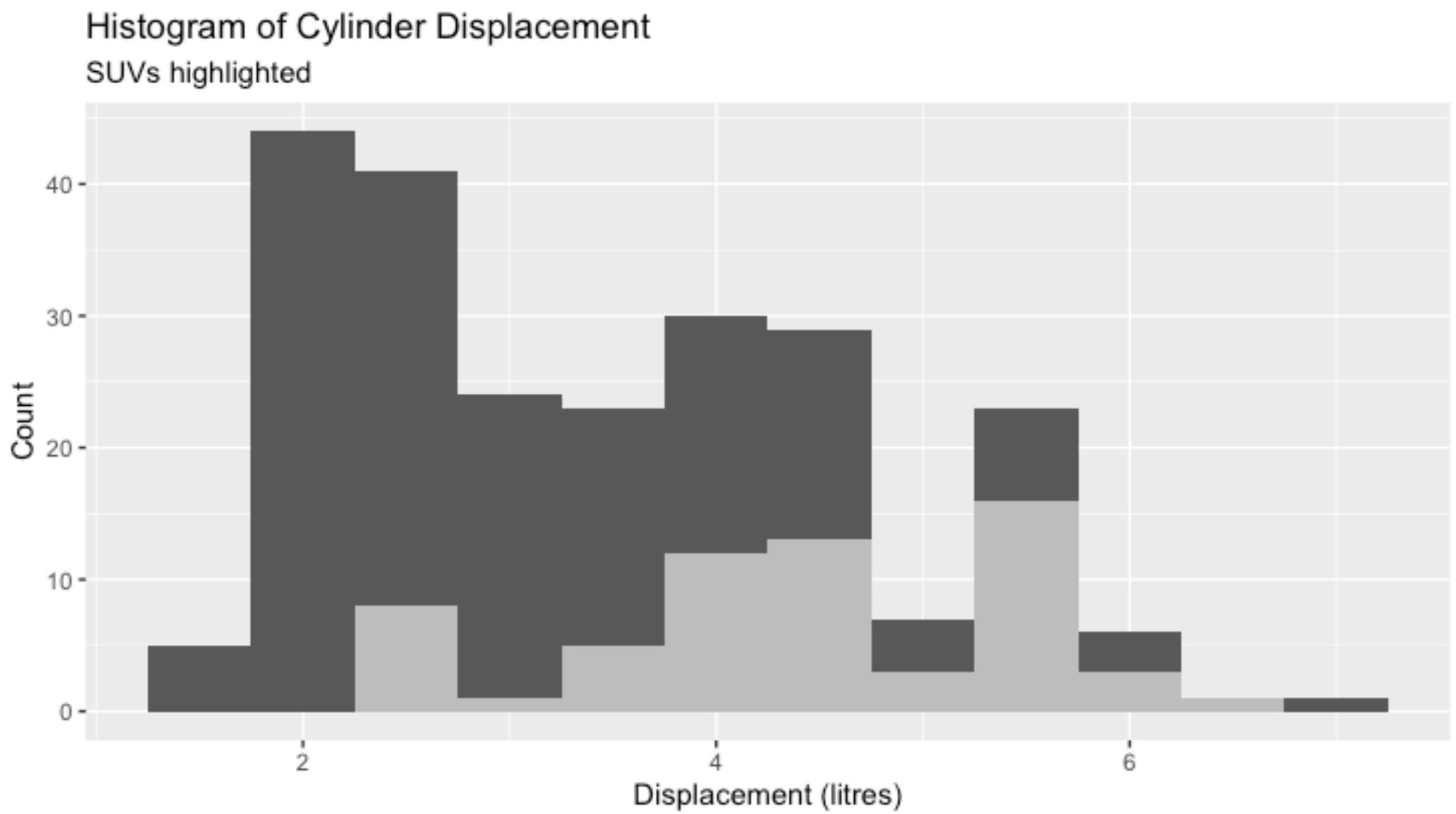
Adding a regression line



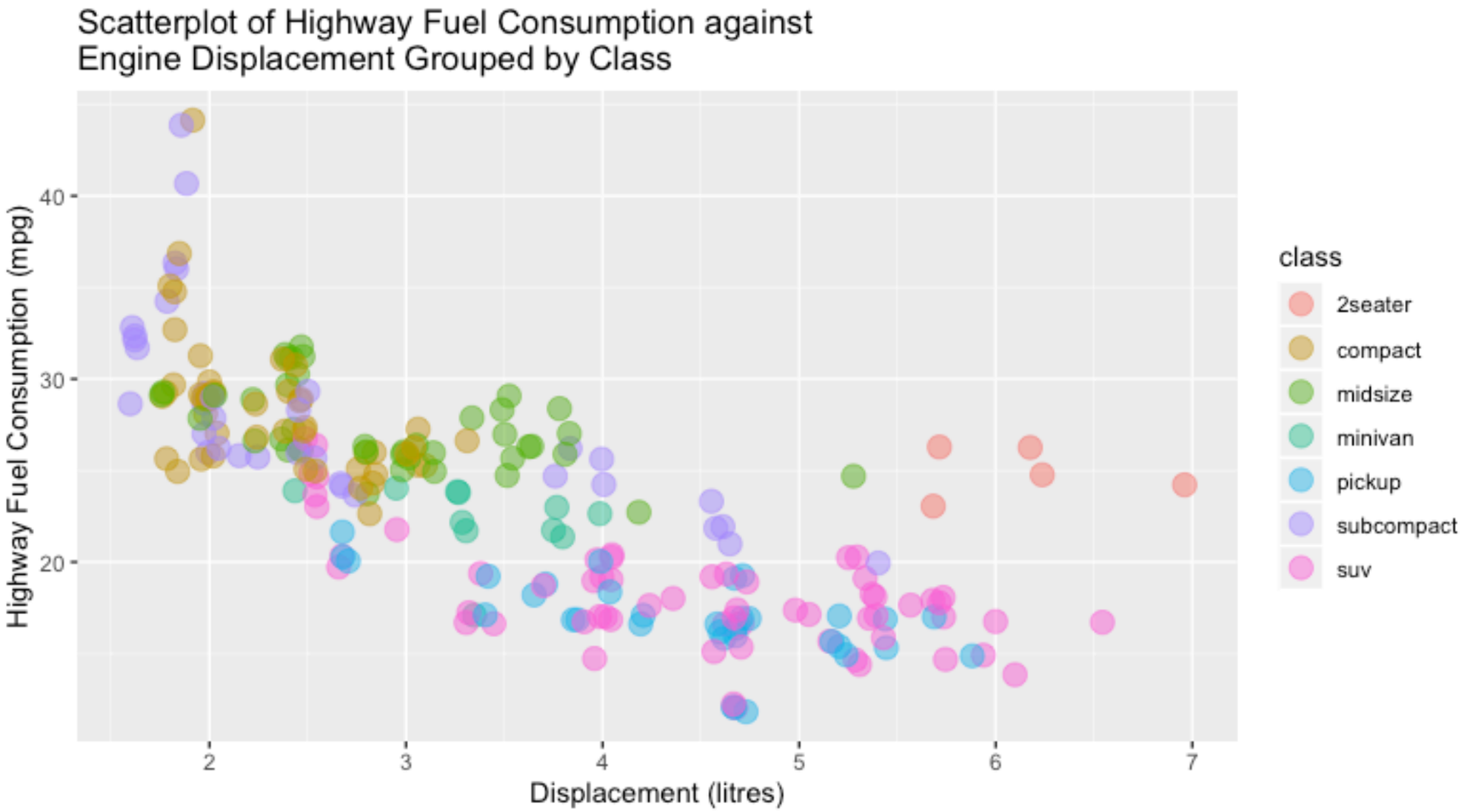

```
ggplot(mpg, aes(x = displ, y = hwy, colour = class)) +
  geom_point() +
  facet_wrap(~ class) +
  guides(colour = FALSE) +
  geom_smooth(method = "lm") +
  labs(title = "Highway Fuel Consumption by Cylinder Displacement \nfor Each Vehicle Class",
       x = "Displacement (litres)",
       y = "Highway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



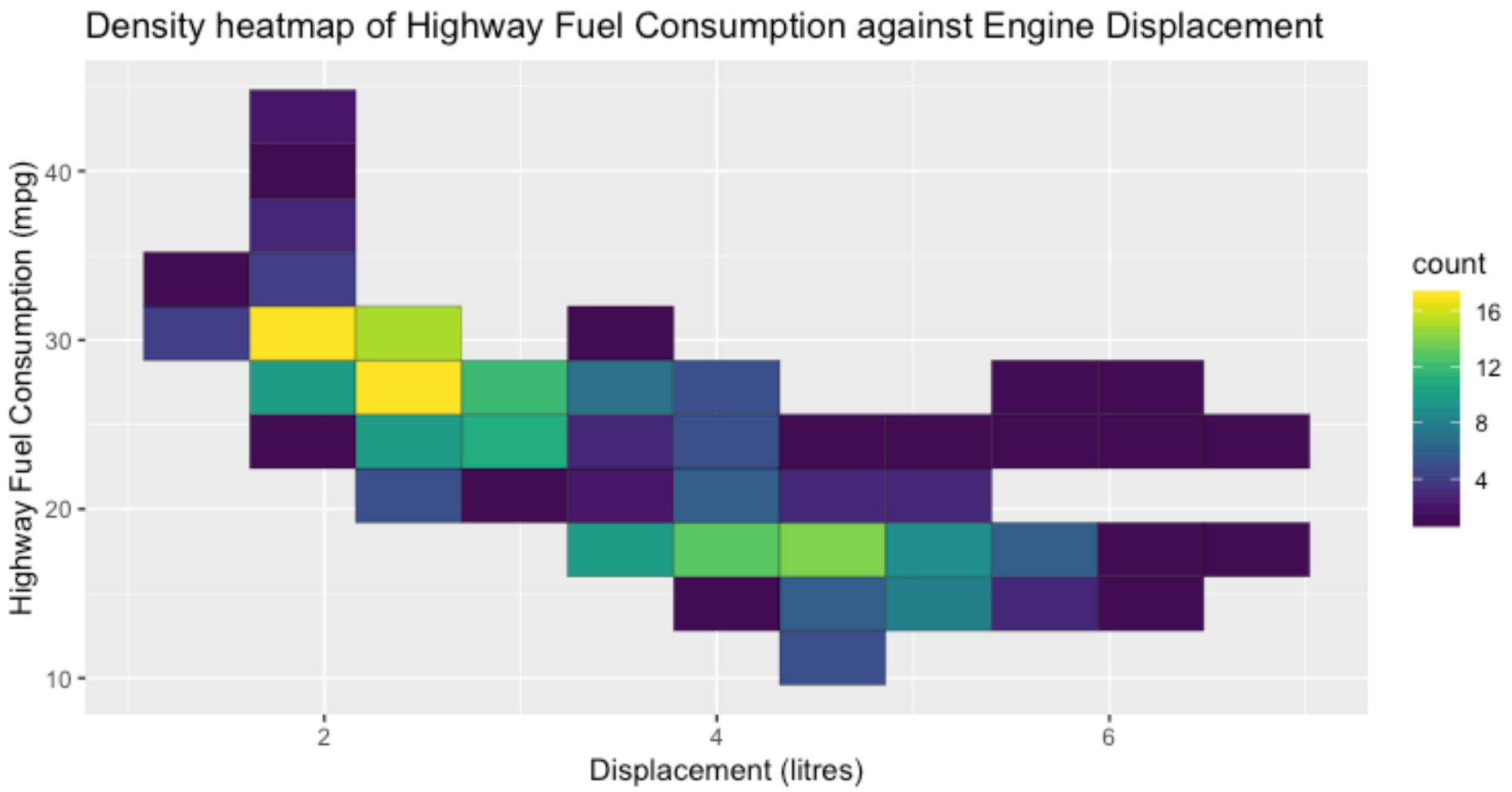

```
ggplot(mpg, aes(x = displ)) +
  geom_histogram(binwidth = .5) +
  guides(fill = FALSE) +
  labs(title = "Histogram of Cylinder Displacement",
       x = "Displacement (litres)",
       y = "Count")
```




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




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




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

Plotting Time Series Data

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

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

Animated visualisations

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

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

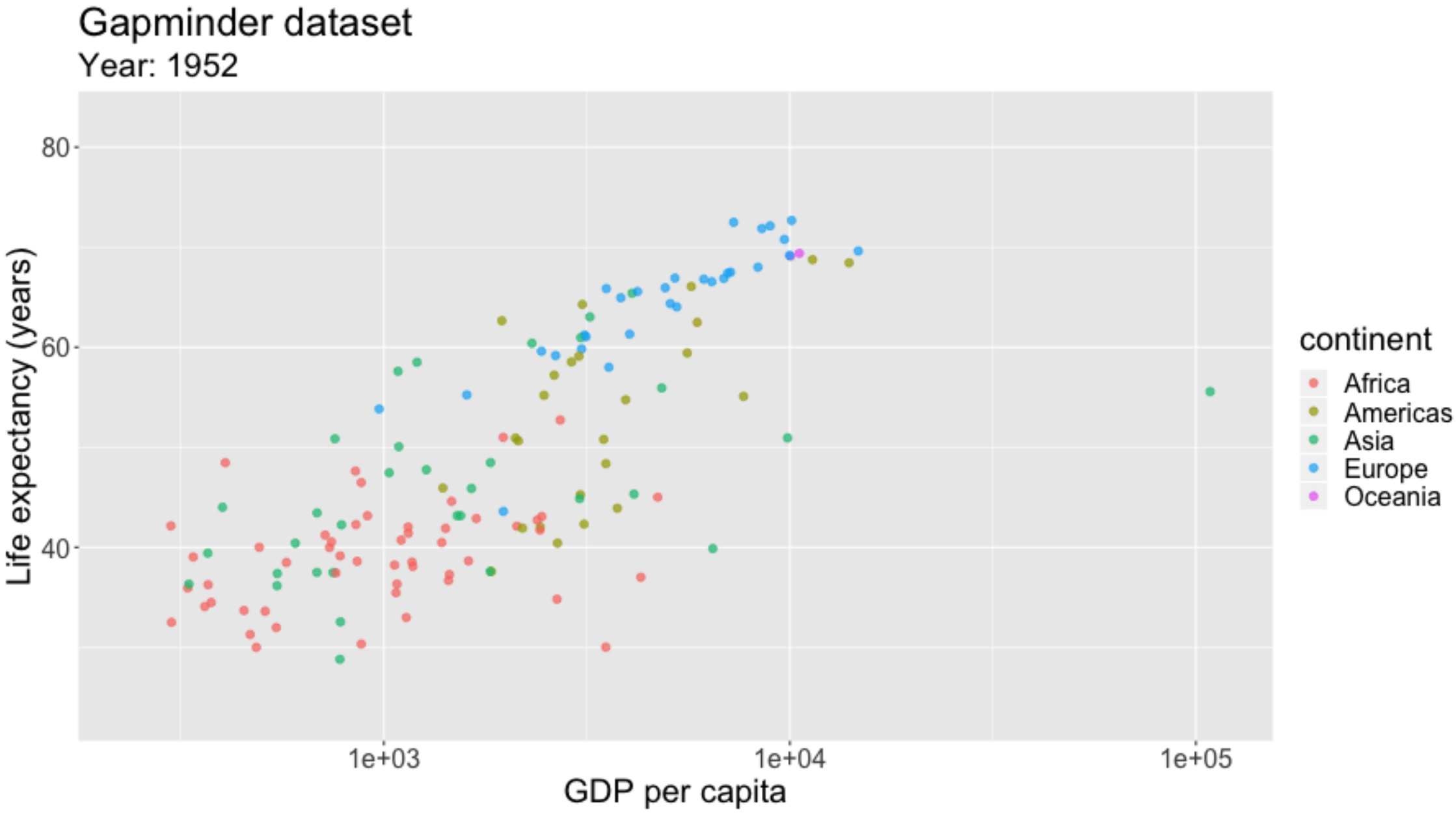
We need to install it via the usual route:

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

and then load it when we want to use it:

```
> library(gganimate)
```

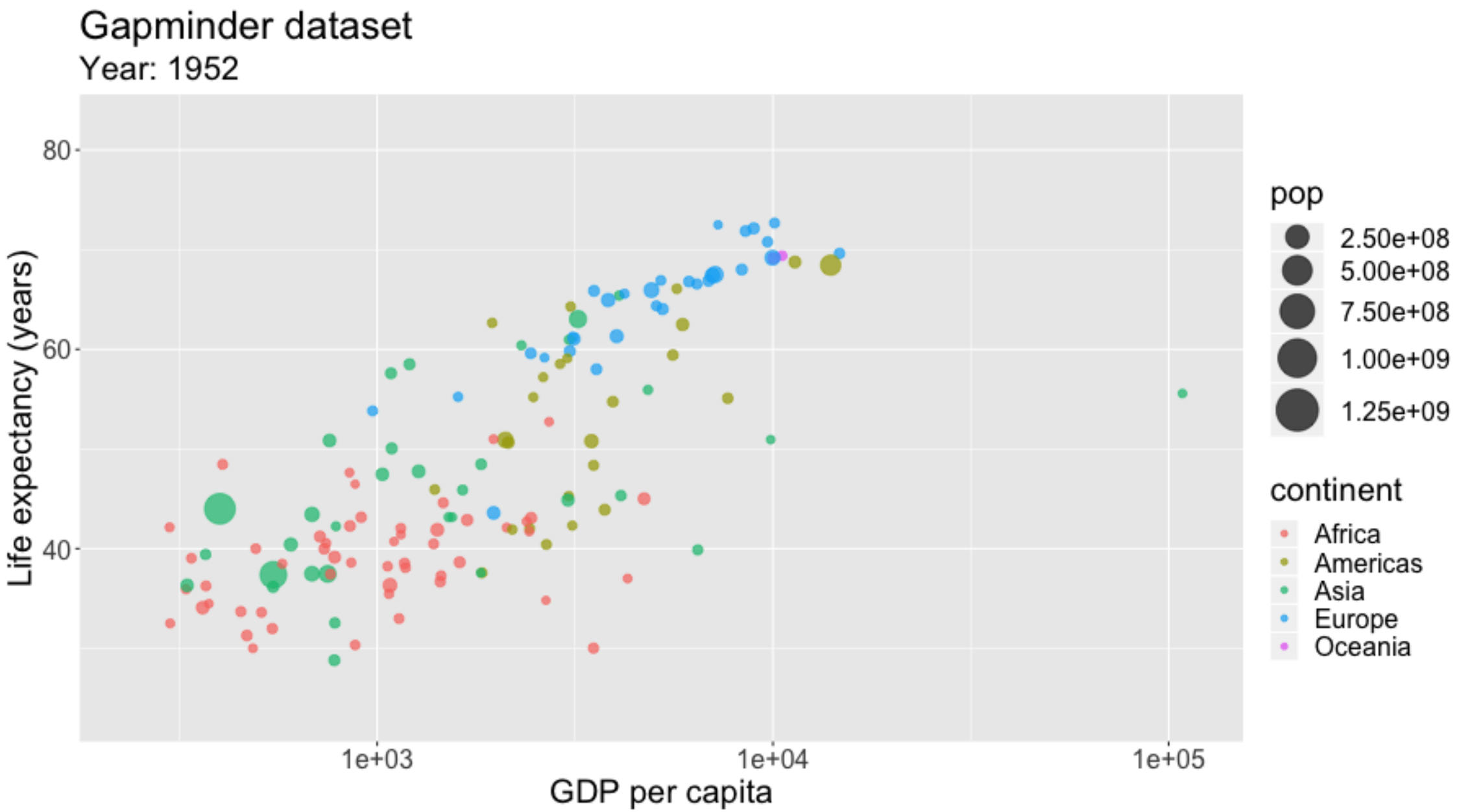
Animated Time Series Data



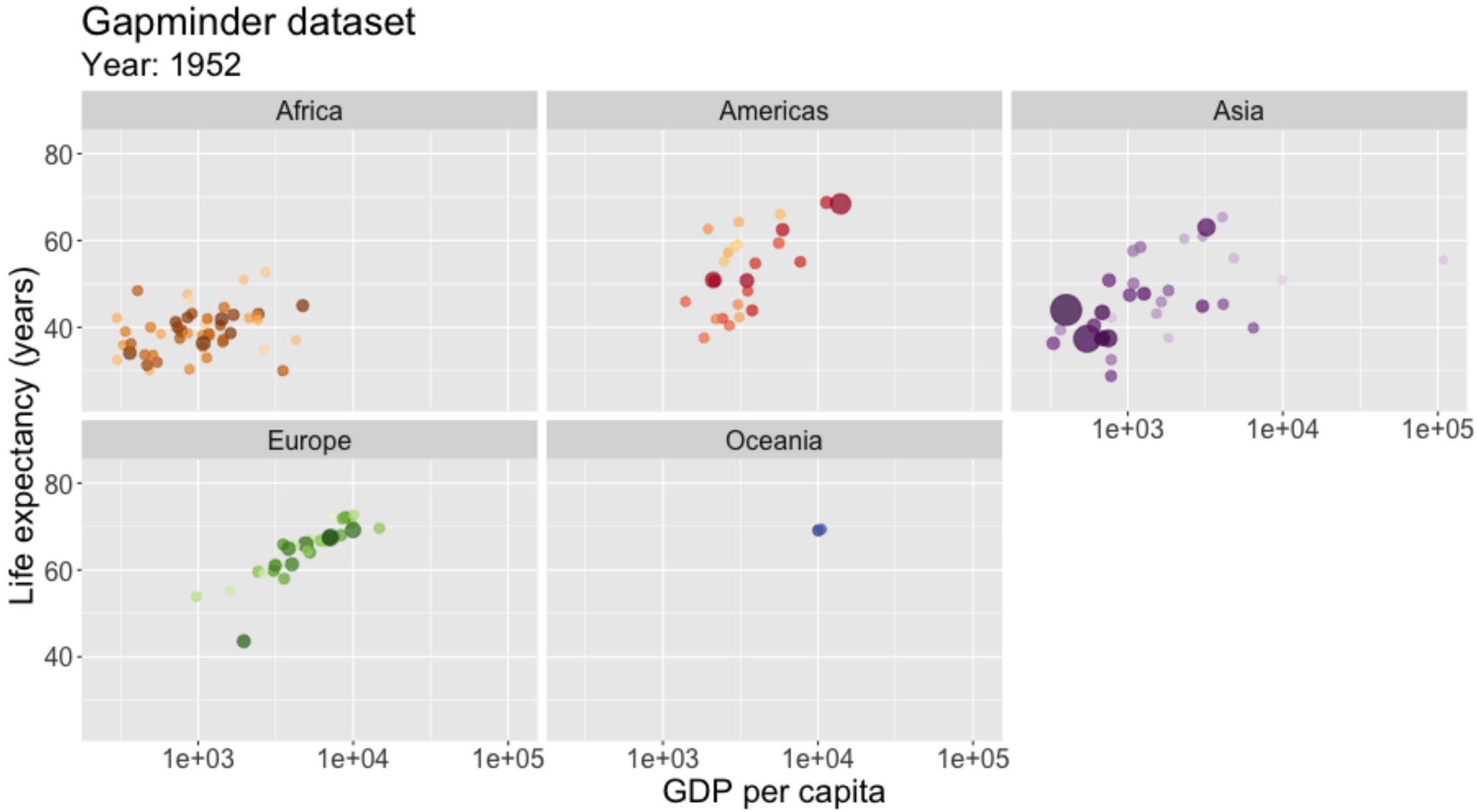
Visualising Data with 5 Variables Simultaneously

Animated Time Series Data

Now with a representation of population size.

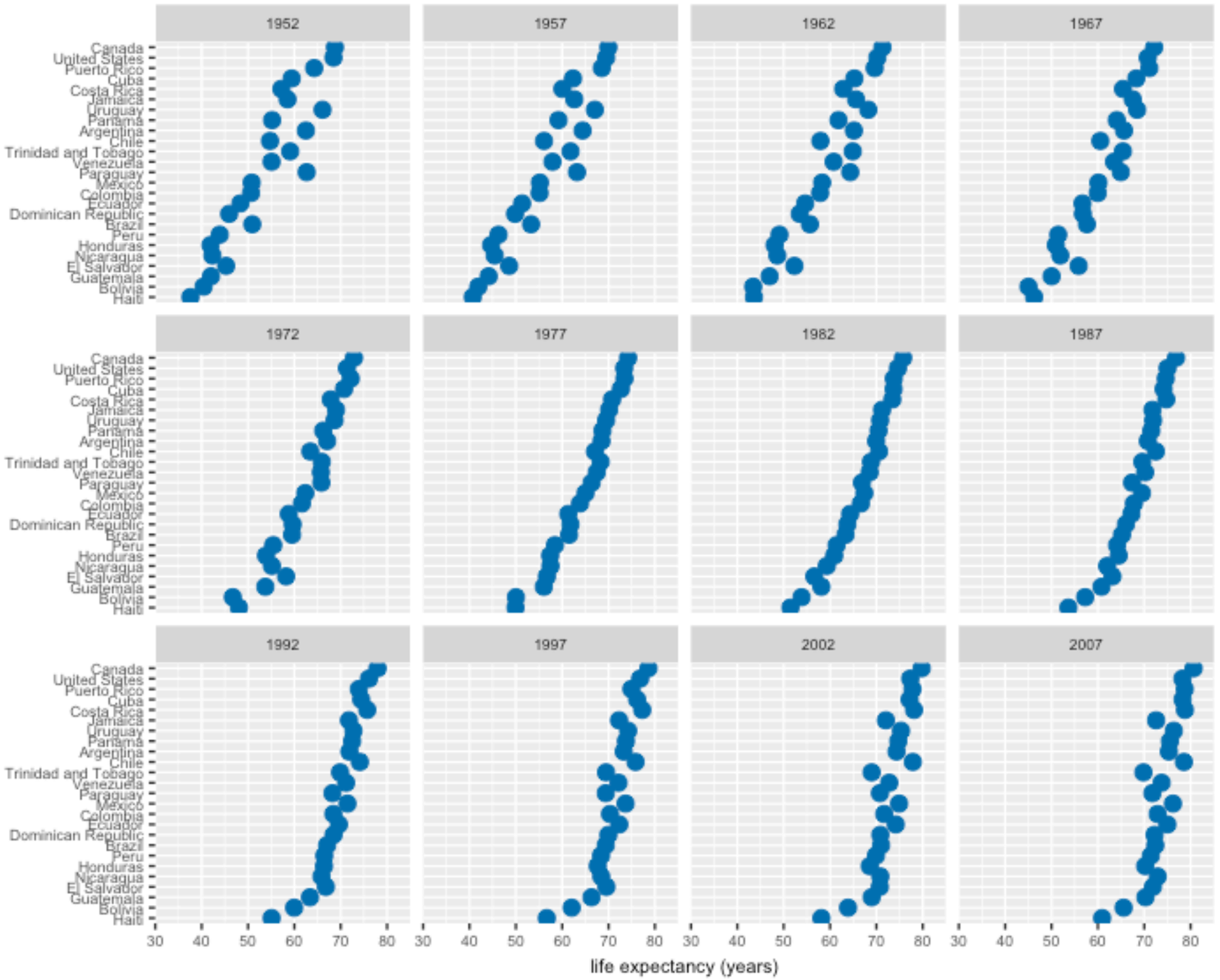


Separately by Continent

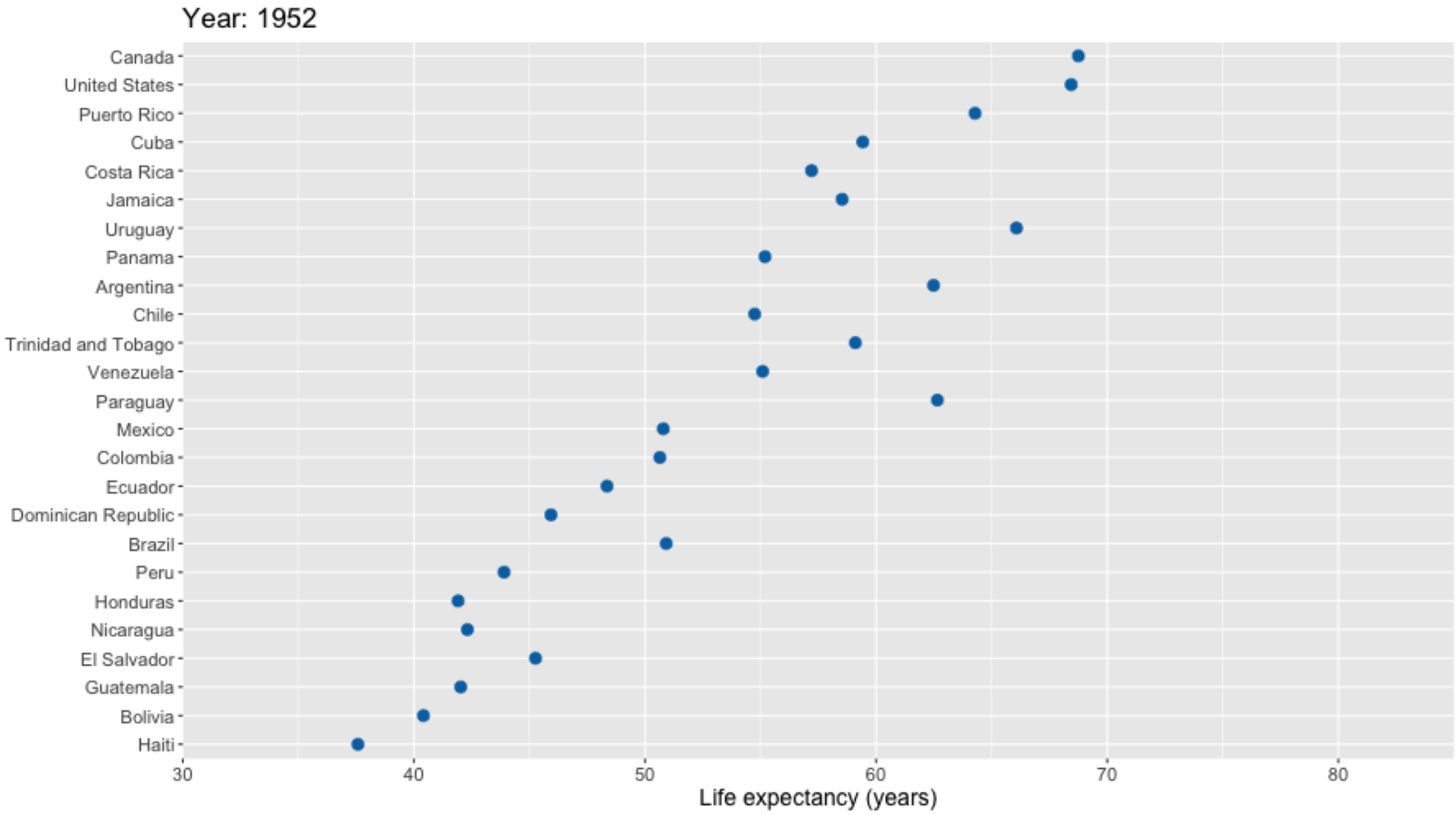


https://github.com/thomasp85/gganimate

Life Expectancy - Americas - Static



Life Expectancy - Americas - Animated



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

You try...

```
library(gganimate)

df_Americas <- gapminder %>% filter(continent == "Americas")

p <- ggplot(df_Americas, aes(x = lifeExp, y = fct_reorder(country, lifeExp))) +
  geom_point(color = "#0072B2", size = 3) +
  scale_x_continuous(name = "Life expectancy (years)",
                     limits = c(30, 85),
                     expand = c(0, 0)) +
  scale_y_discrete(name = NULL, expand = c(0, 0.5)) +
  labs(title = 'Year: {frame_time}') +
  theme(text = element_text(size = 14)) +
  transition_time(year) +
  ease_aes('linear')

animate(p, height = 500, width = 900)
```

Worksheet 1