Lecture 3 - Data Wrangling and Visualisation

Andrew Stewart

Andrew.Stewart@manchester.ac.uk

@ajstewart_lang

Session	Topic	Lecturer
1	Introduction, Open Science, and Power	Andrew Stewart
2	Introduction to R	Andrew Stewart
3	Data Wrangling and Visualisation	Andrew Stewart
4	General Linear Model - Regression	Andrew Stewart
5	General Linear Model - Regression	Andrew Stewart
6	General Linear Model - ANOVA	Andrew Stewart
7	General Linear Model - ANOVA	Andrew Stewart
8	General Linear Model - ANOVA	Andrew Stewart
9	Signal Detection Theory	Ellen Poliakoff
10	Signal Detection Theory	Ellen Poliakoff
11	Revision Session	Andrew Stewart

Semester 1 Assignments

ANOVA – Due around the end of November

Signal Detection Analysis – Due around mid-January

Last Week

- We had our first introduction to R and R Studio.

- In the second half of class, you went from zero to hero in terms of using R and running a script for some data manipulation and graphing using ggplot.

- Last reminder to sign up to our Slack channel (i.e., no more hassling from me about it)…

Some Academic Twitter Accounts on Open Science to Follow

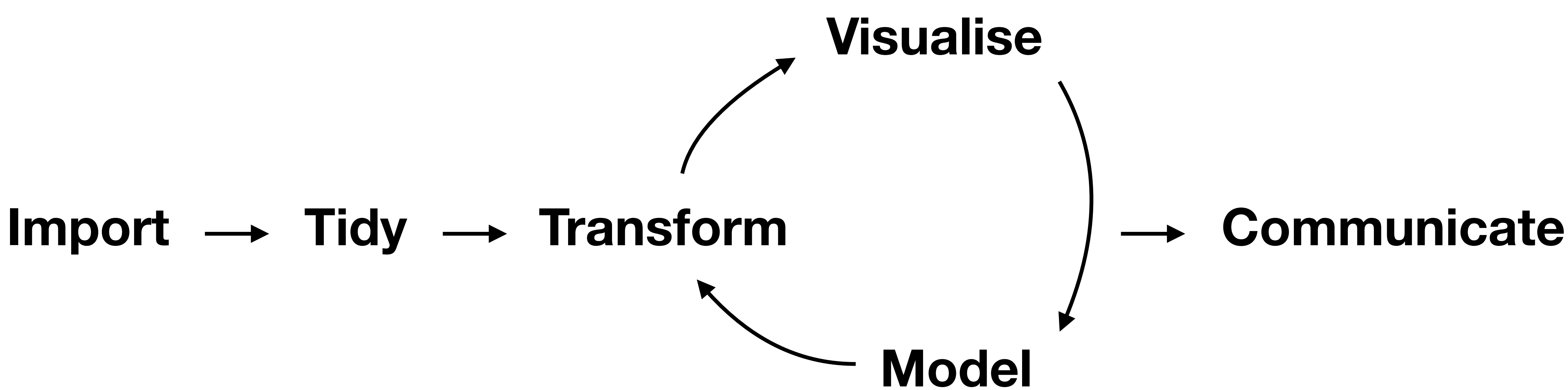
- Brian Nosek (Virginia) - @BrianNosek
- Dorothy Bishop (Oxford) - @deevybee
- Marcus Munafò (Bristol) - @MarcusMunafo
- Chris Chambers (Cardiff) - @chrisdc77
- The UK Reproducibility Network - @UKRepro
- Center for Open Science - @OSFramework

This Week

- We’re going to look at some data wrangling - getting your data into the right format and shape for analysis.

- We’re also looking at data visualisation (aka data viz.) - you should always visualise your data (often in more than one way) before you move onto statistical modelling…

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



https://www.tidyverse.org

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`.

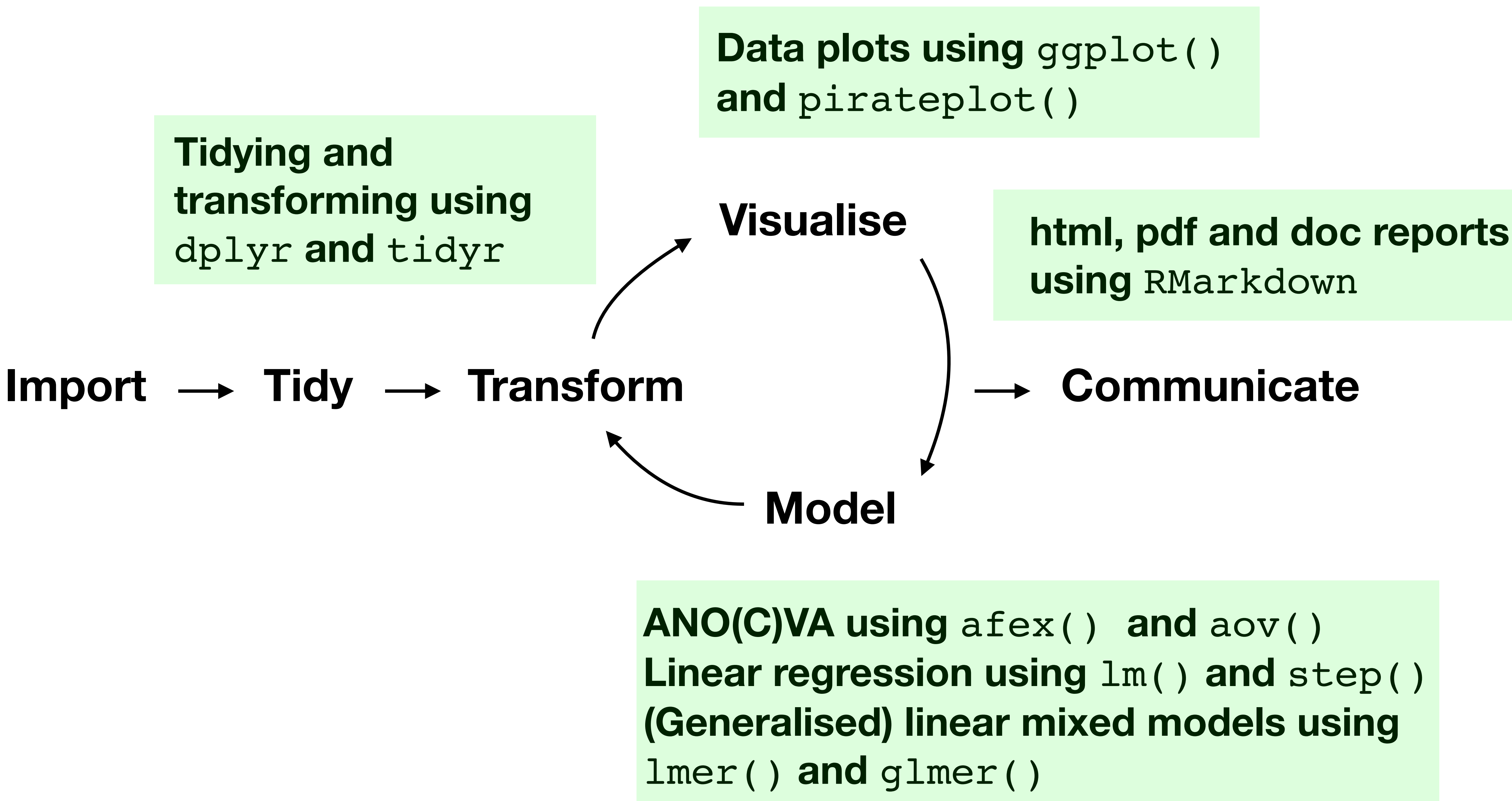
- You can load each package separately with (e.g.)

```
> library(ggplot2)
```

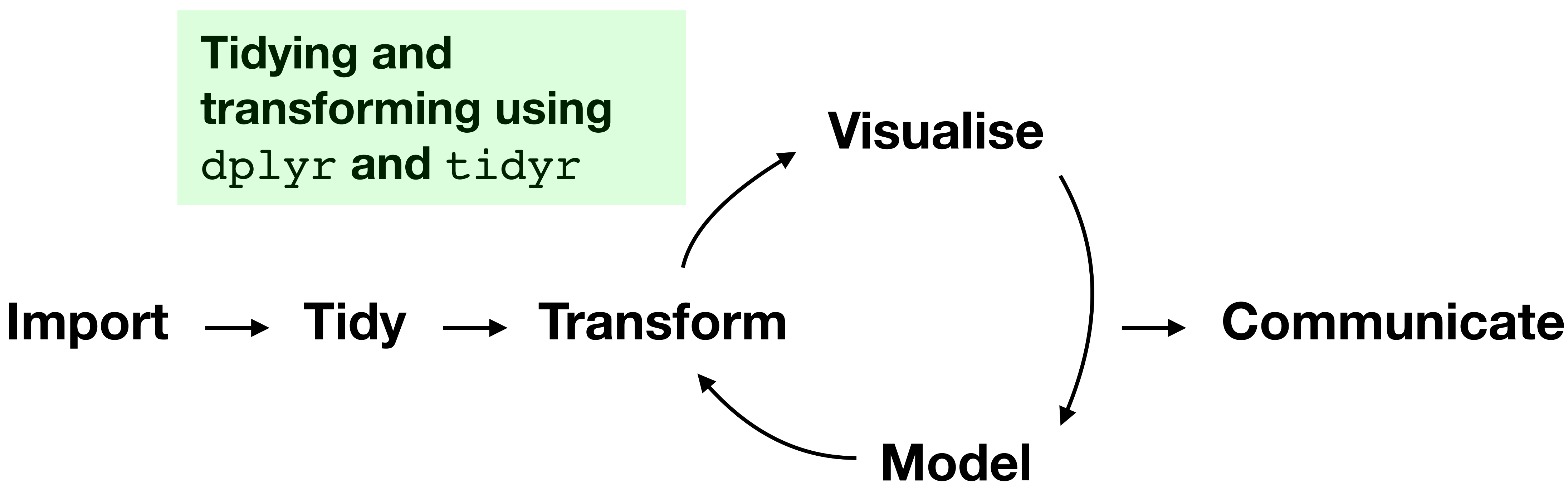
- or load all tidyverse packages with

```
> library(tidyverse)
```

Workflow


Data plots using ggplot() and pirateplot()
Tidying and transforming using dplyr and tidyr
Visualise
html, pdf and doc reports using RMarkdown
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 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  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 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 comprised of their reading times and their individual difference measures from two separate (and different sized) data frames…with one line of code…

	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

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  9937            1996             2551
2  1506            2235             2310
3  5212            2177             2244
4   374            1824             2483
5  6757            2113             2567
6  1778            2056             2791
7  9421            2037             2226
8  5576            2073             2270
9  7326            1830             2640
10 4166            1824             2386
```

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:

	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
11	6929	Simple Sentence	1771
12	5444	Simple Sentence	1836

Showing 1 to 13 of 96 entries

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

```
> data_wide <- spread(data_long, "Condition", "RT", c("Simple
Sentence", "Complex Sentence”))
> View(data_wide)
```

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

	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
11	2324	2349	1933
12	2579	2391	2356

Showing 1 to 12 of 48 entries

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 transparency).

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))
```

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), sd =
sd(RT))

# A tibble: 2 x 3
  Condition          mean    sd
  <fct>             <dbl> <dbl>
1 Complex Sentence 2393.  181.
2 Simple Sentence  1987.  143.
```

`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).

```
#now separate the Condition column using "_" as our separator
> data <- separate(data, col = "Condition", into = c("Prime", "Target"),
sep = "_")

> head(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
6           2 PrimeA TargetB 1050
```

- 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).

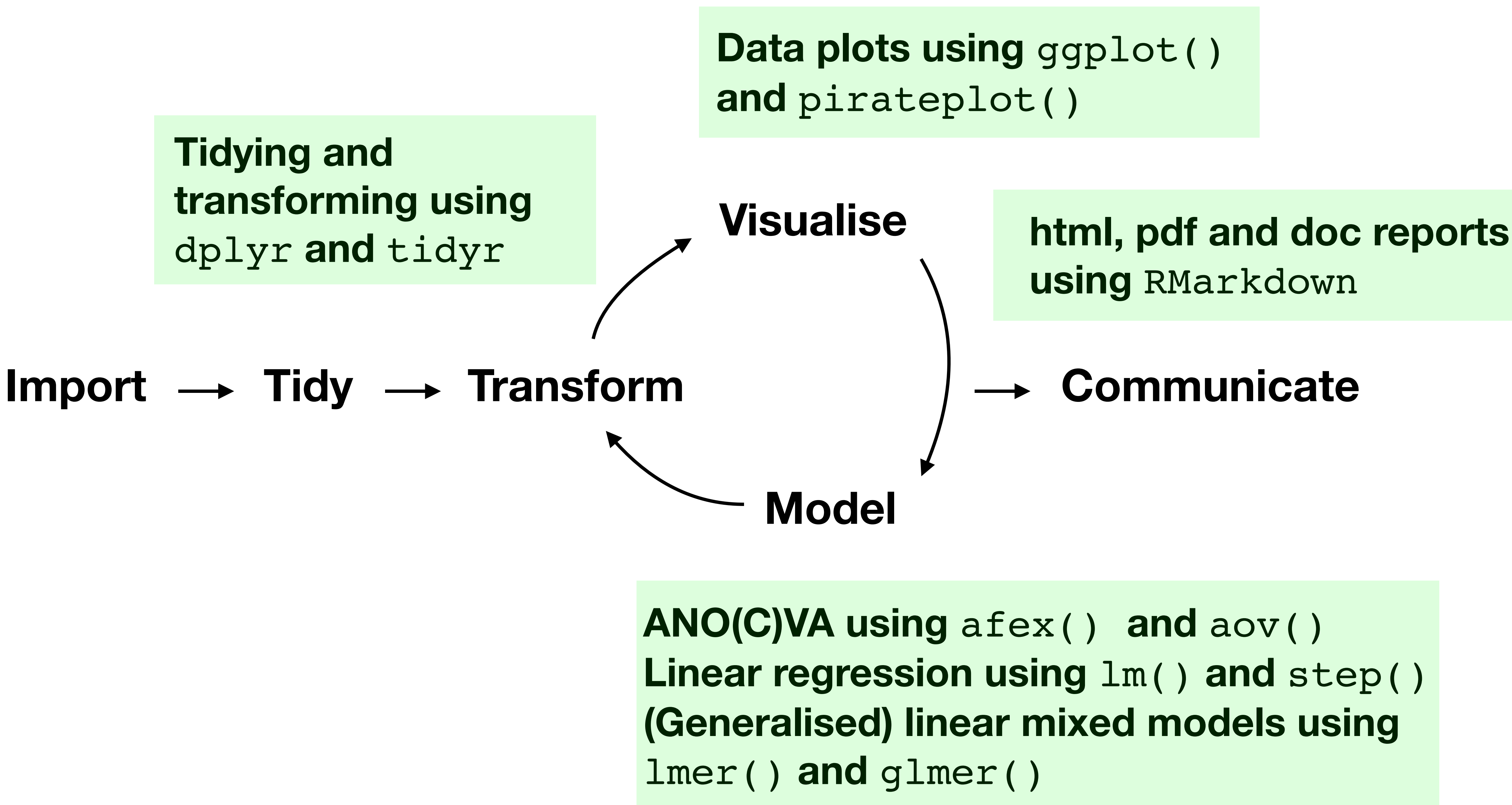
- Perhaps we want to go from the data in long format, to wide format.

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

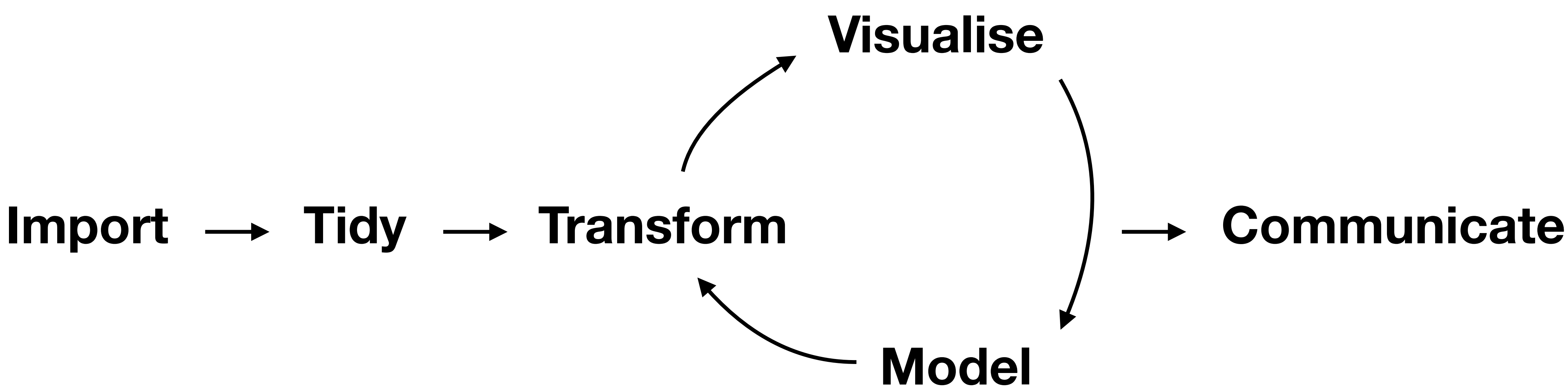
- No matter what format your data are in originally, you can use functions from the `dplyr` and `tidyr` packages to quickly get it into whatever format you need for analysis.

Workflow


Data plots using ggplot()
and pirateplot()
Tidying and
transforming using
dplyr and tidyr
Visualise
html, pdf and doc reports
using RMarkdown
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()

Visualise

Data plots using `ggplot()` **and** `pirateplot()`



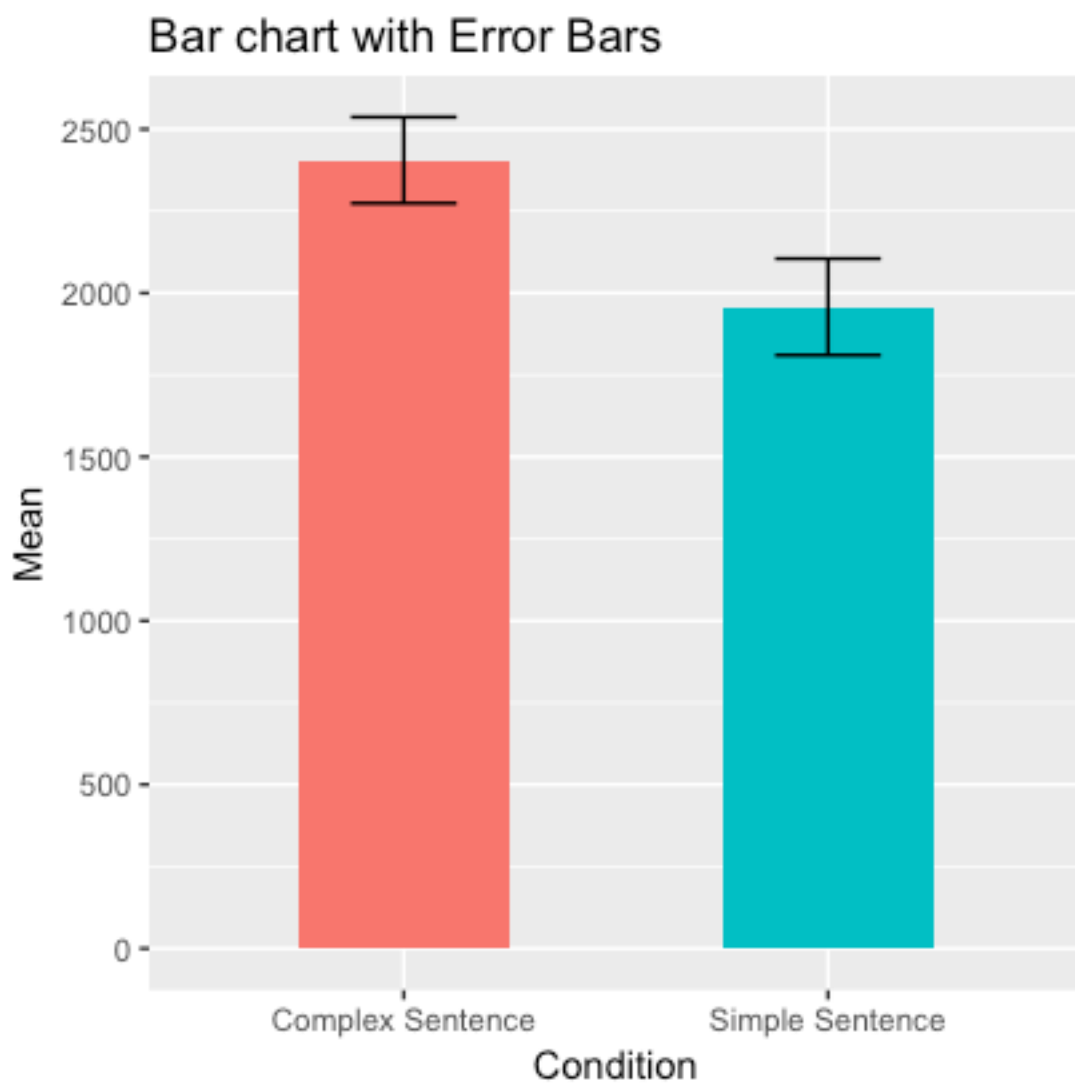
Visualising Your Data

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

```
> library(ggplot2)

> library(yarrr)
```

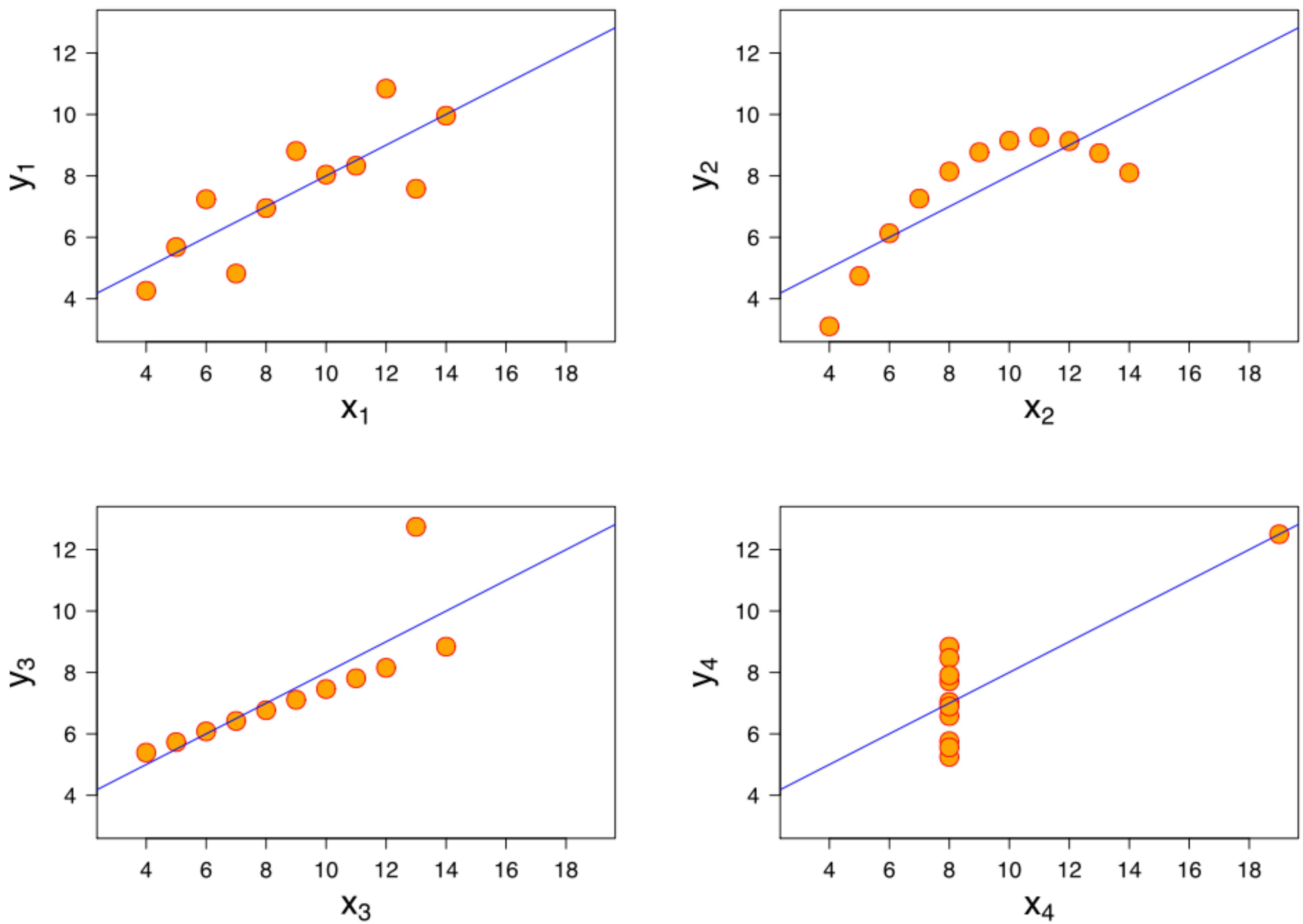
Bar Graphs



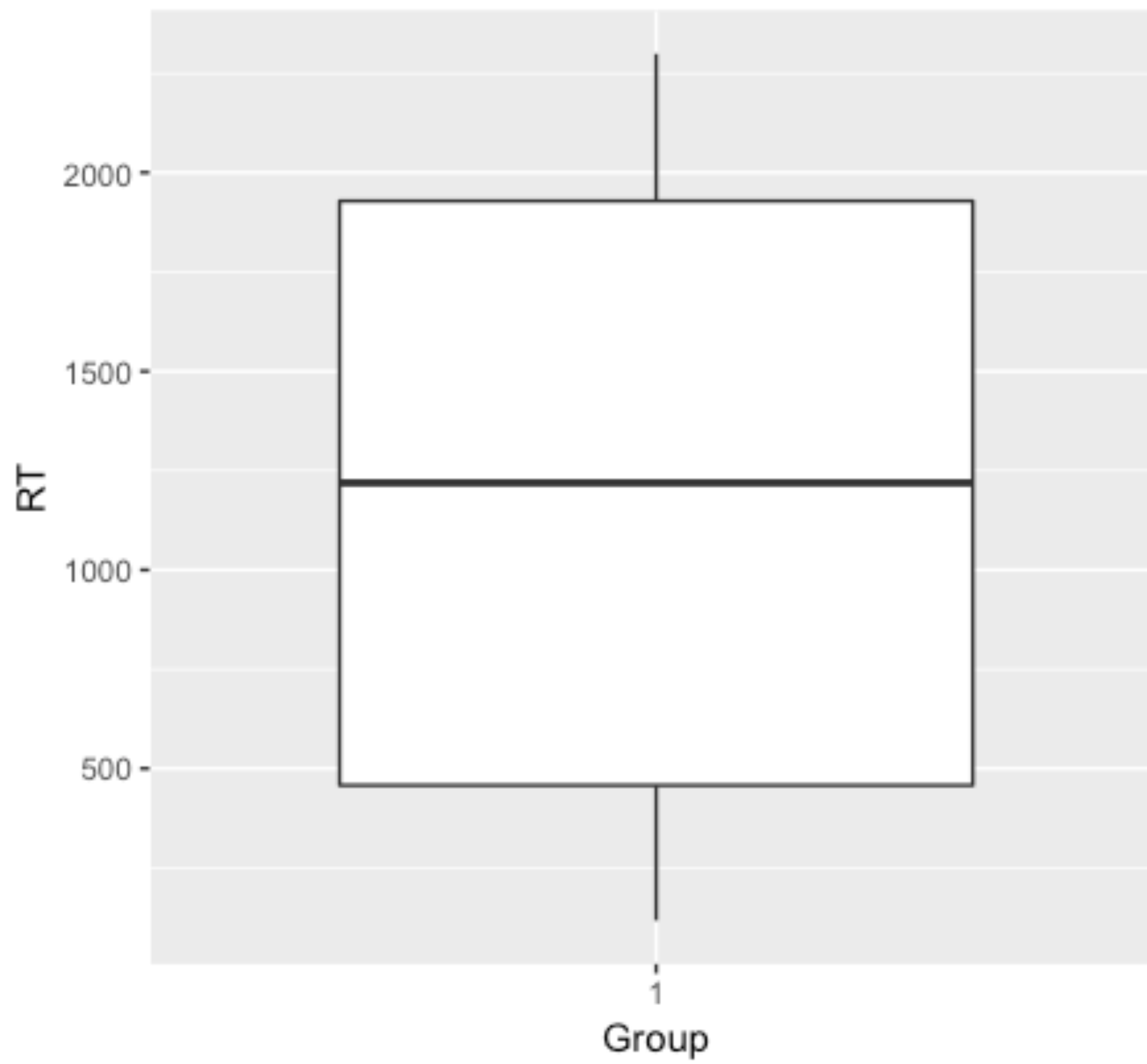
Bar graphs tend to be quite limited in terms of what they communicate. Here they communicate the means for levels of a factor and information about variance. But they don't tell us anything about the *distribution* of the data.

```
> 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

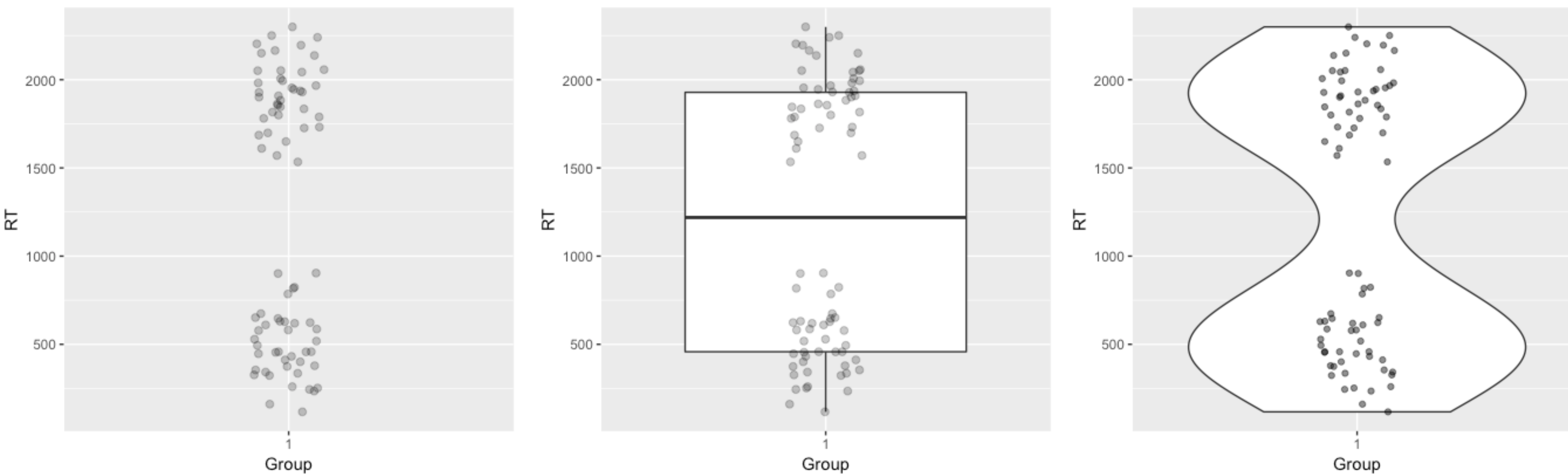


Plots Based on Aggregated Data Can Mislead…



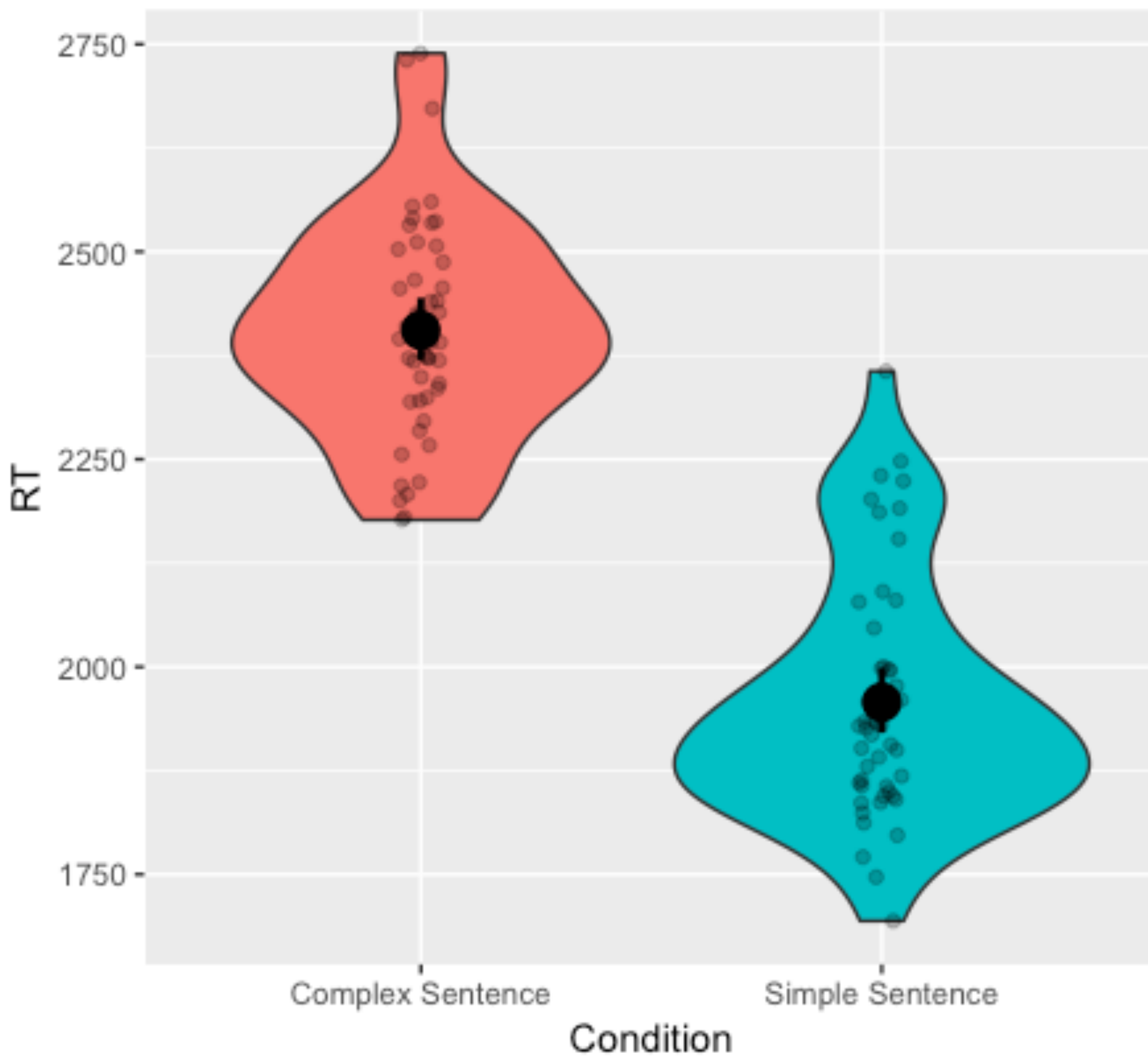
You might make one set of inferences based on this boxplot - maybe a measure of central tendency around 1,250 with the 25th and 75th percentiles associated with the data being ~480 to ~1,980…

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.

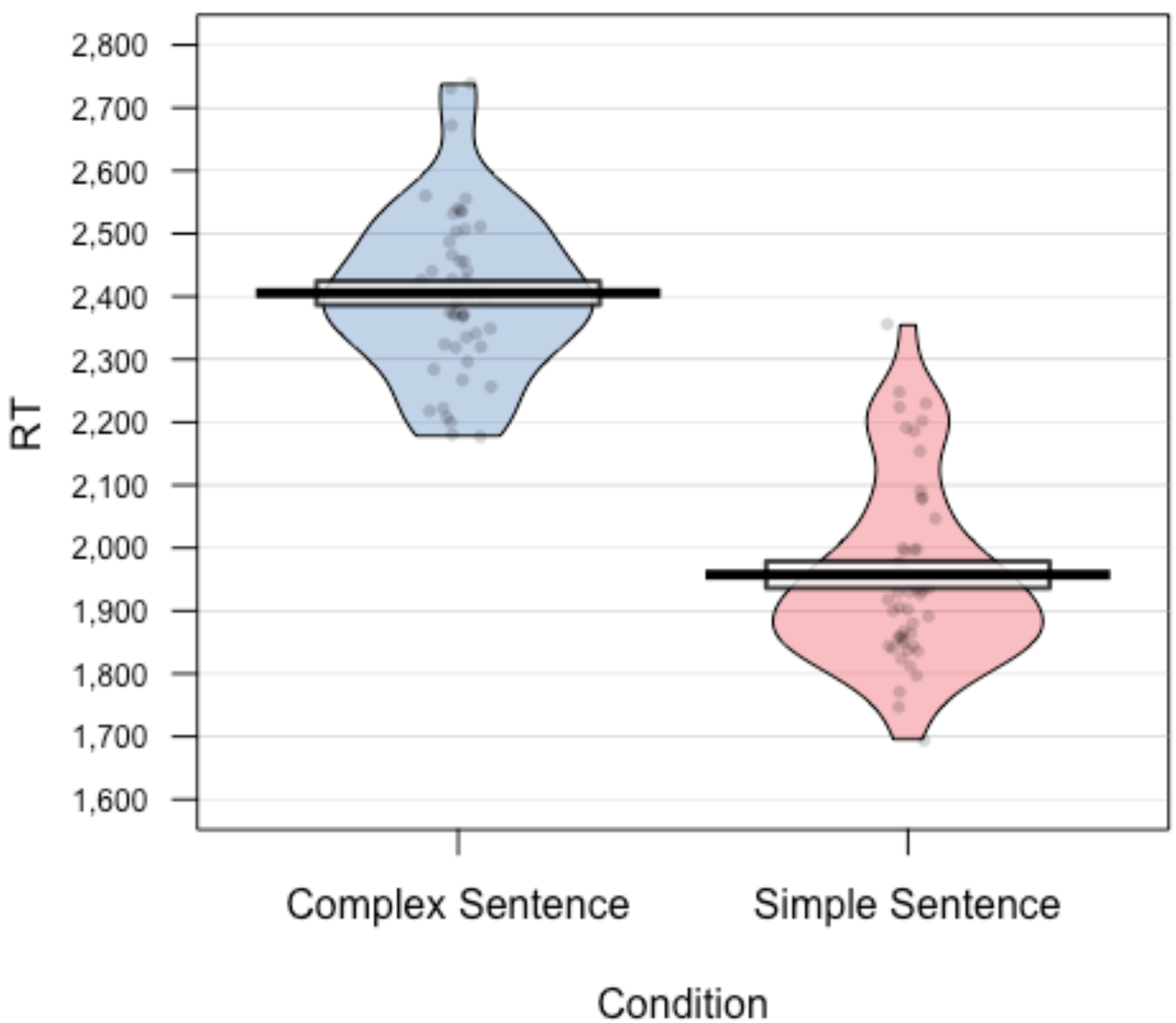
Violin Plots



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)
```

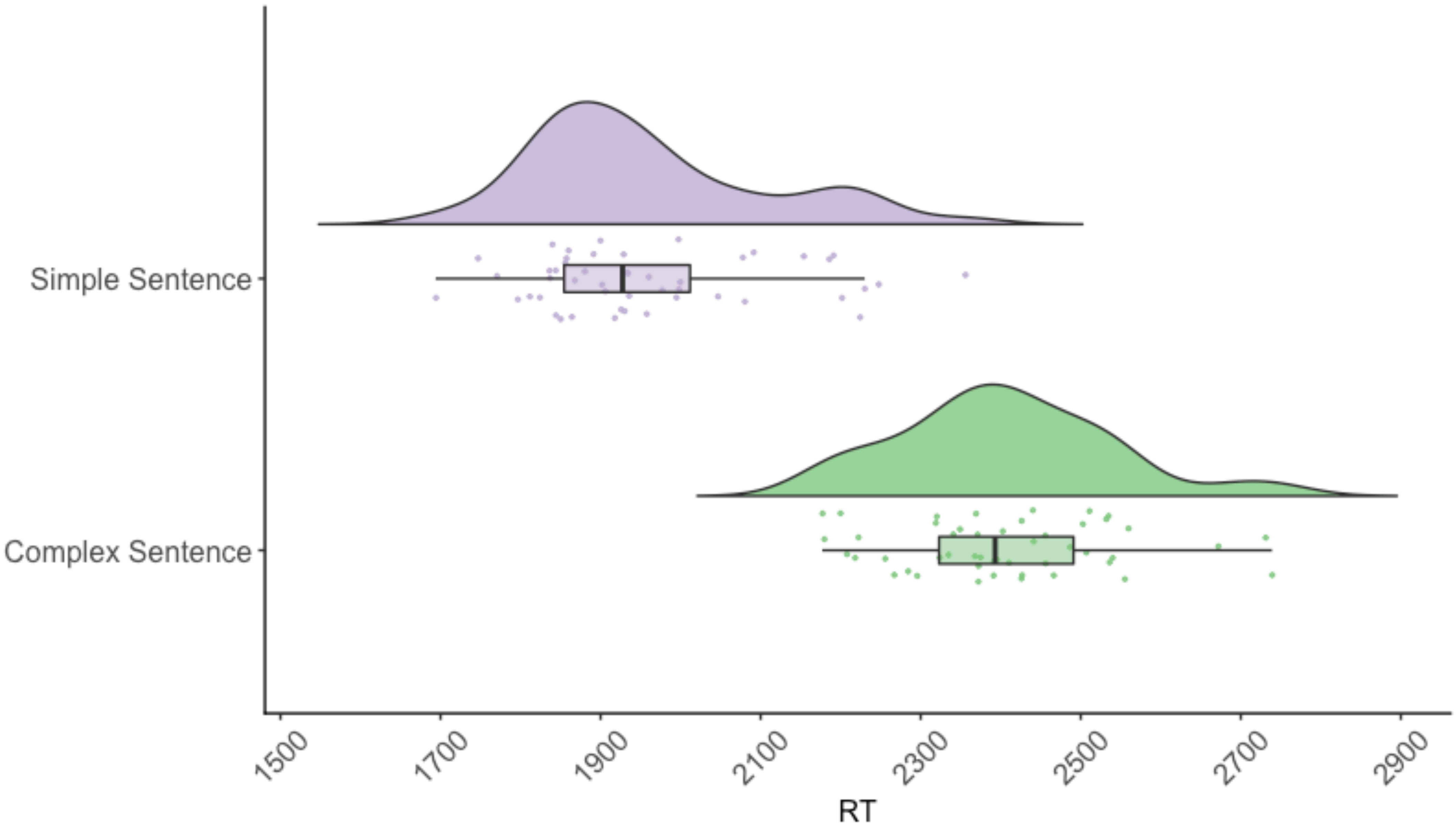
Pirate Plots



Pirate Plots are an example of RDI (Raw data, Descriptive and Inferential statistic) plots. Available in the package `yarrr`. Plots include shape of distribution, mean, and SE (all changeable as parameters).

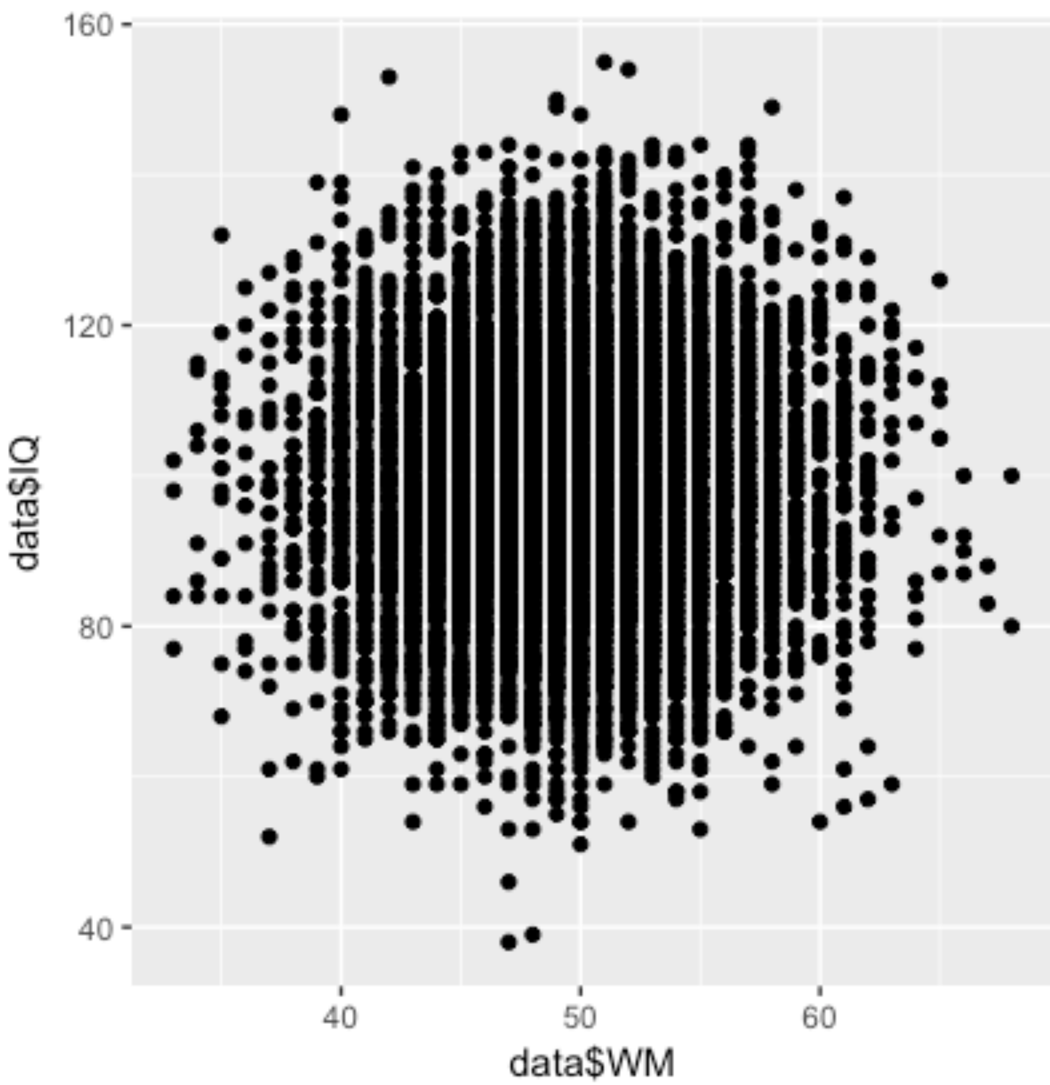
```
> pirateplot(formula = RT~Condition, data = data_long,
inf.method = "se", cex.axis = .75, theme = 1)
```

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.)

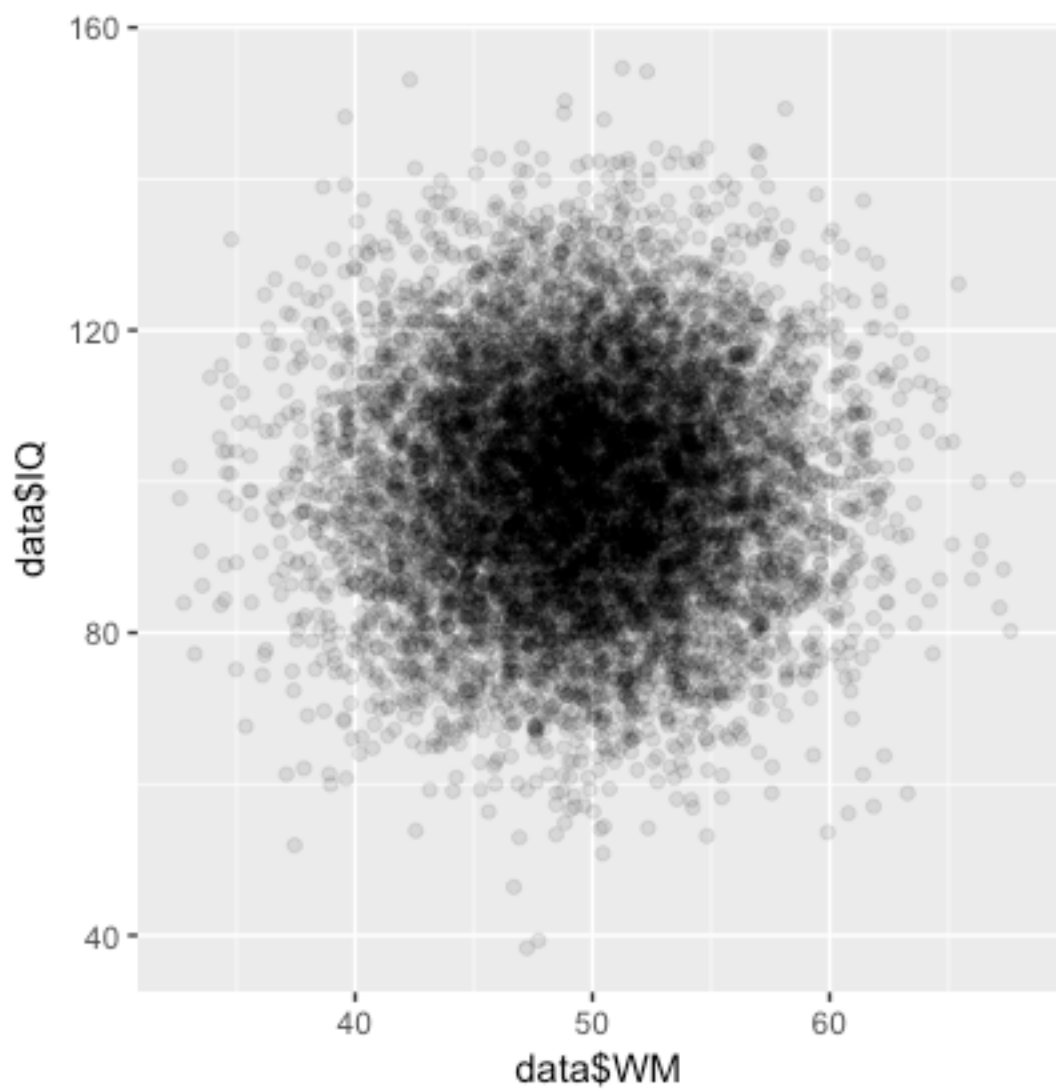
Plotting IQ against WM for our 10,000 participants



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()
```

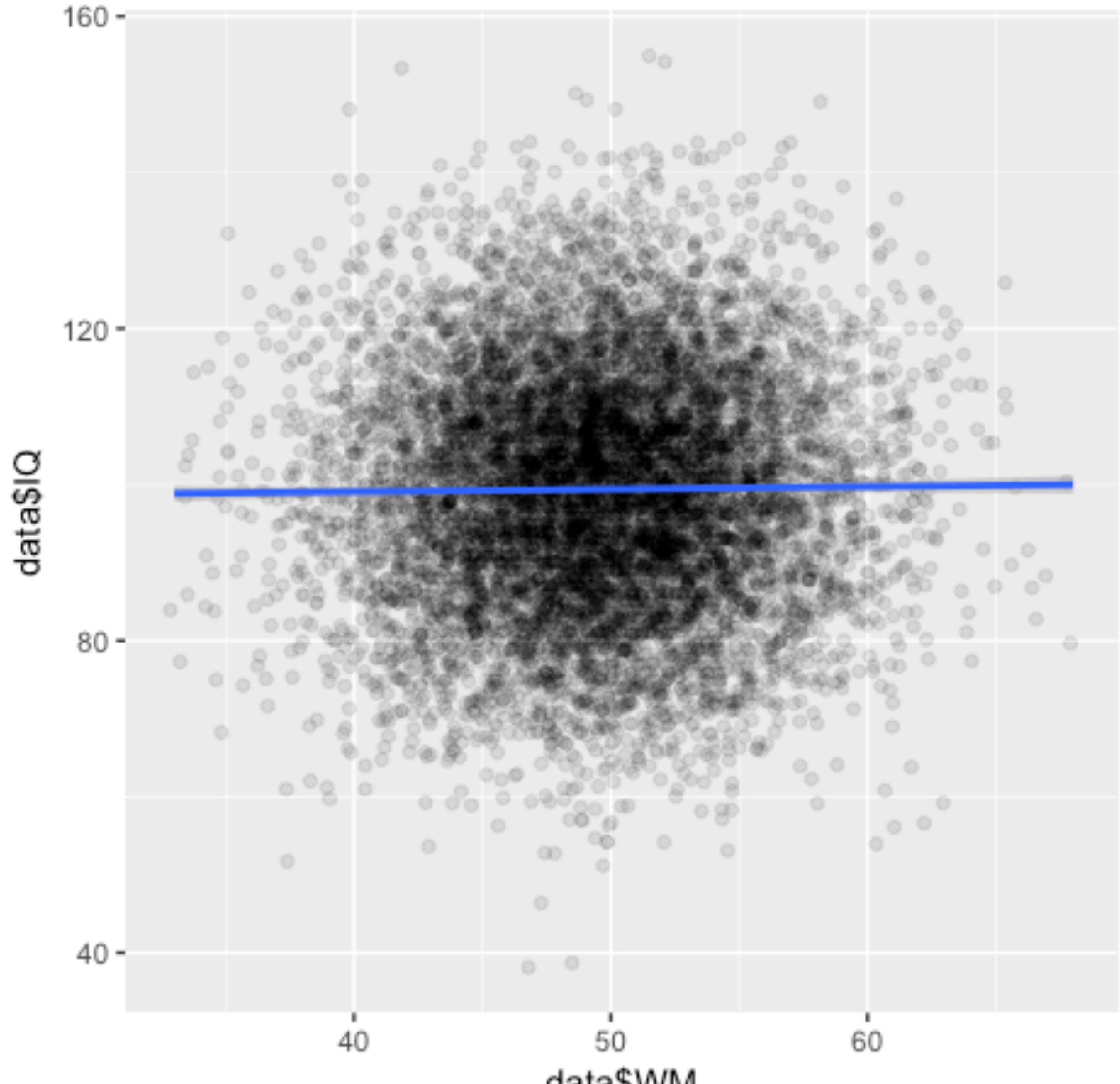
Plotting IQ against WM for our 10,000 participants



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

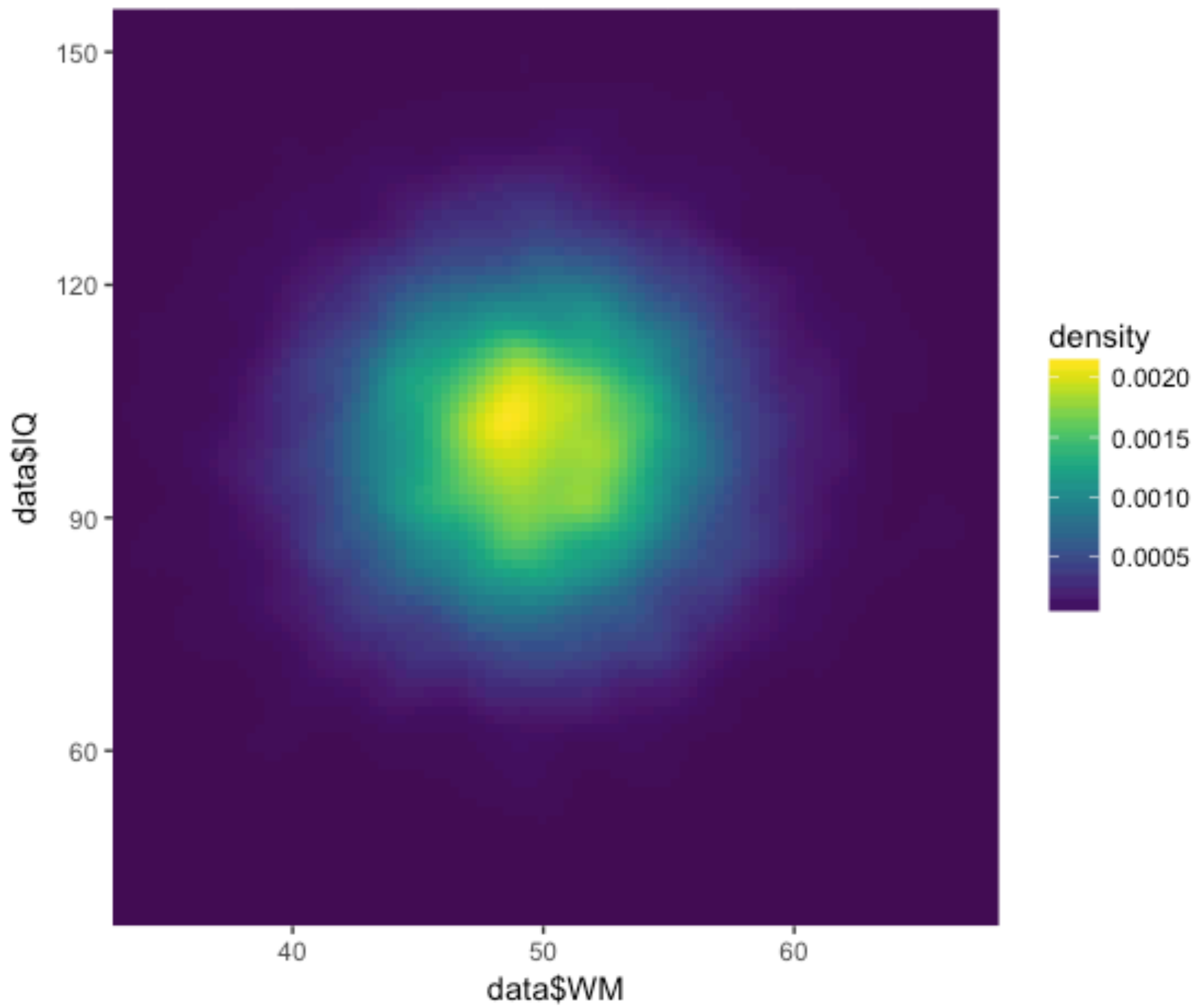
```
> ggplot(data, aes(x = WM, y = IQ)) + geom_jitter(alpha = .1, position
= position_jitter(0.5))
```

With a regression line




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

And as a Density Heat Map

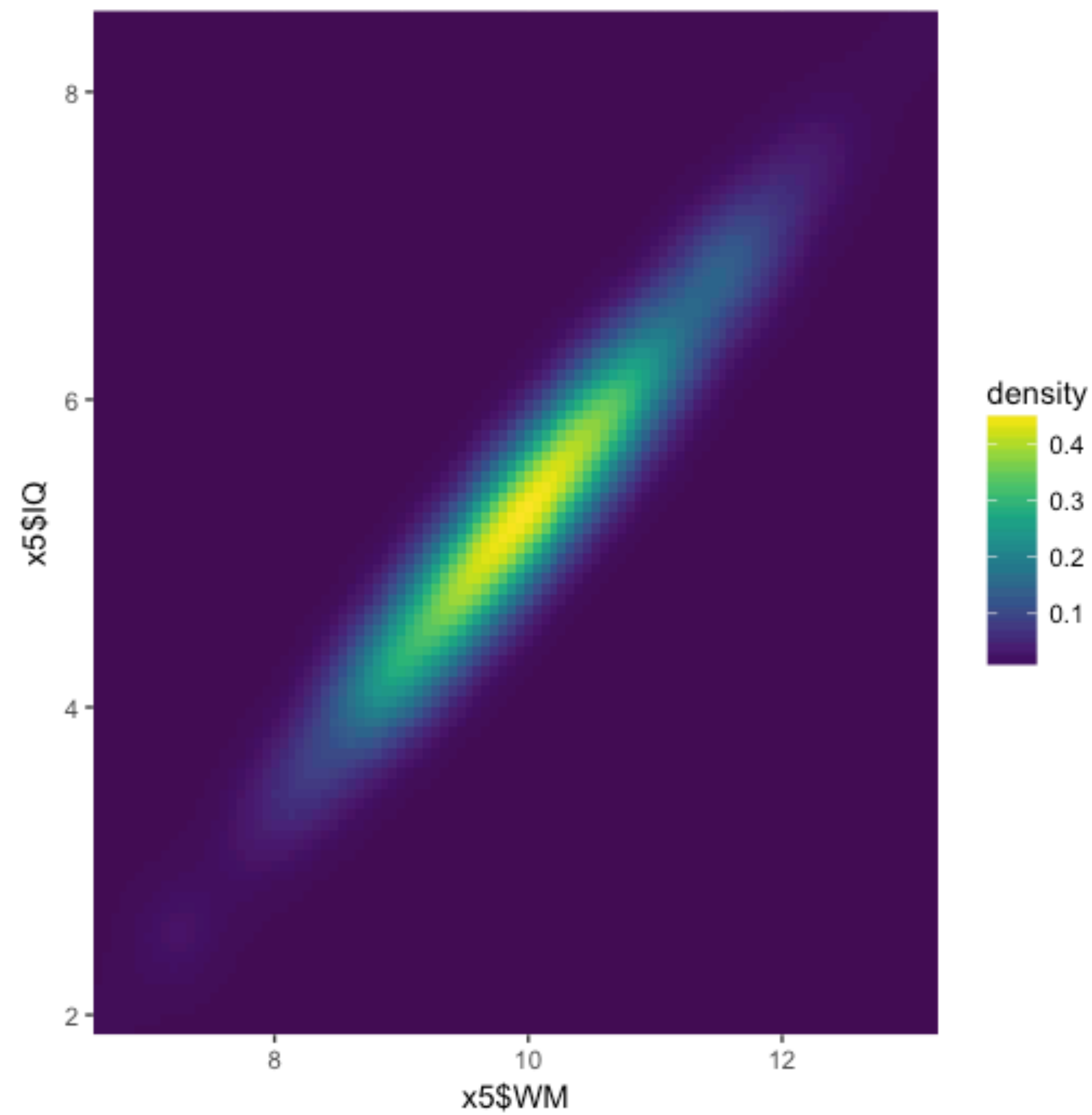



```
> ggplot(data, aes(x = WM, y = IQ)) + stat_density_2d(aes(fill = ..density..), ge
= 'raster', contour = FALSE) + scale_fill_viridis() + coord_cartesian(expand =
FALSE)
```

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:

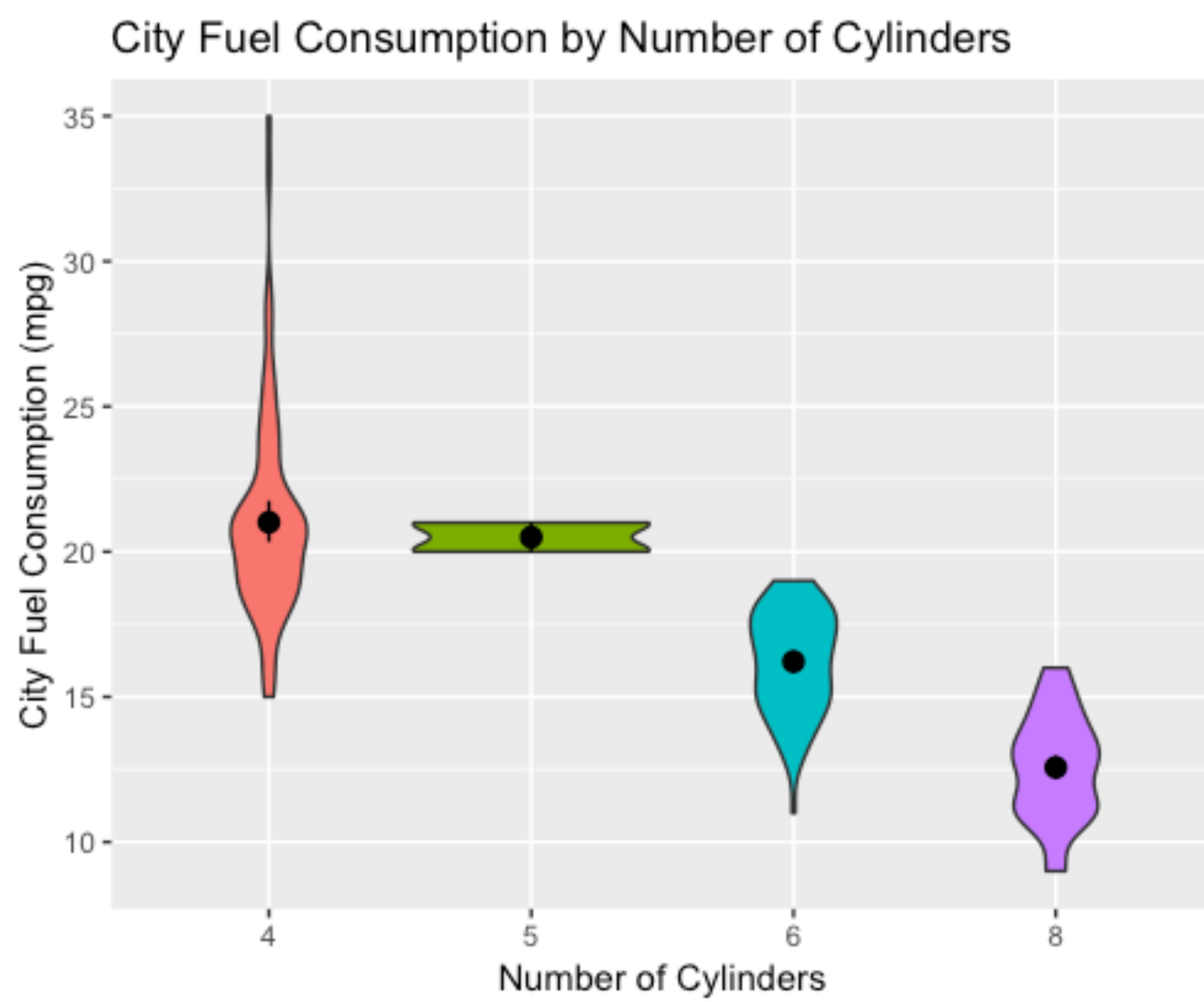
```
> unique(mpg$manufacturer)
 [1] "audi"       "chevrolet"  "dodge"      "ford"       "honda"      "hyundai"    "jeep"
 [8] "land rover" "lincoln"    "mercury"    "nissan"     "pontiac"    "subaru"     "toyota"
[15] "volkswagen"
```

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

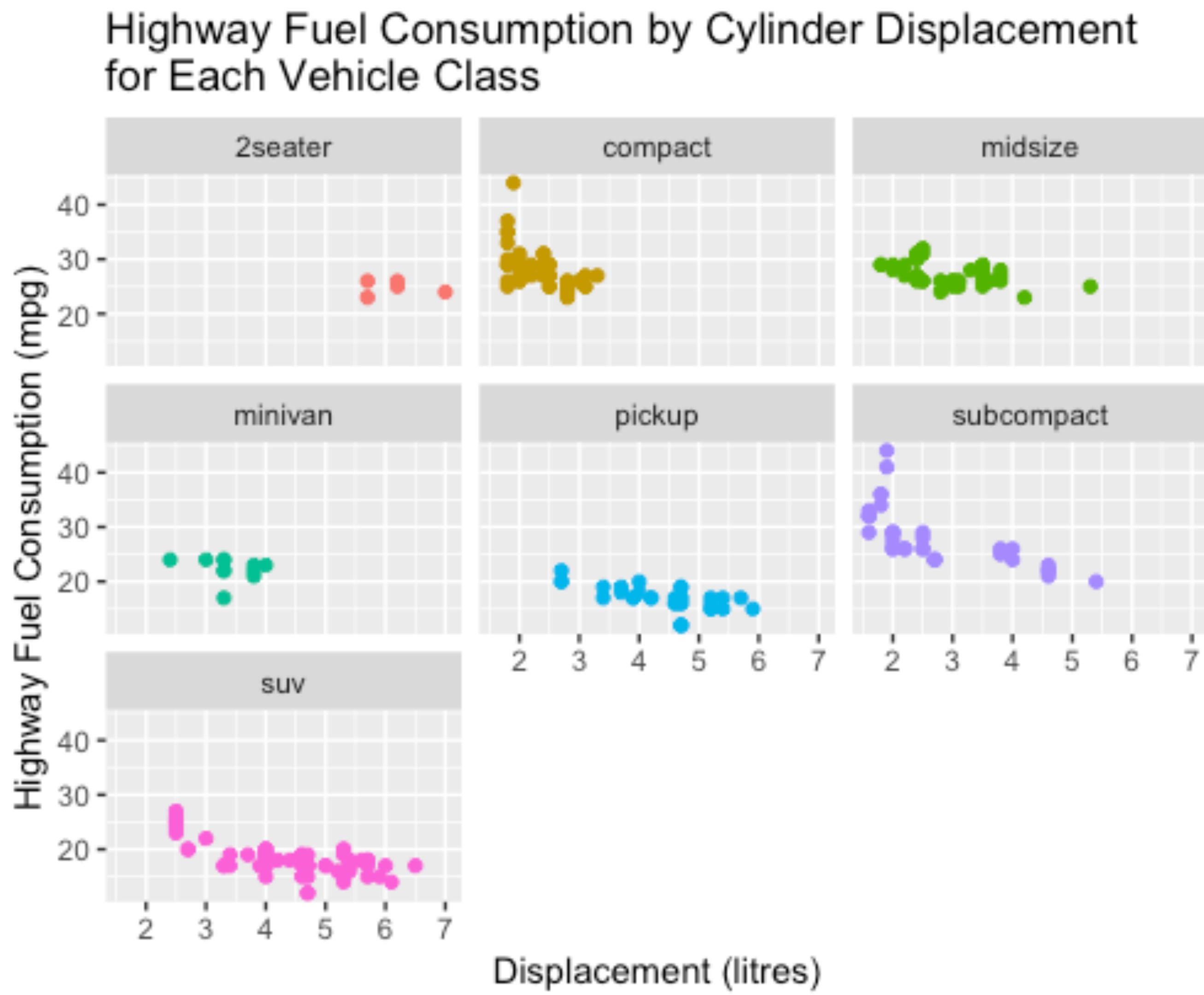
```
> length(unique(mpg$manufacturer))
[1] 15
```

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…

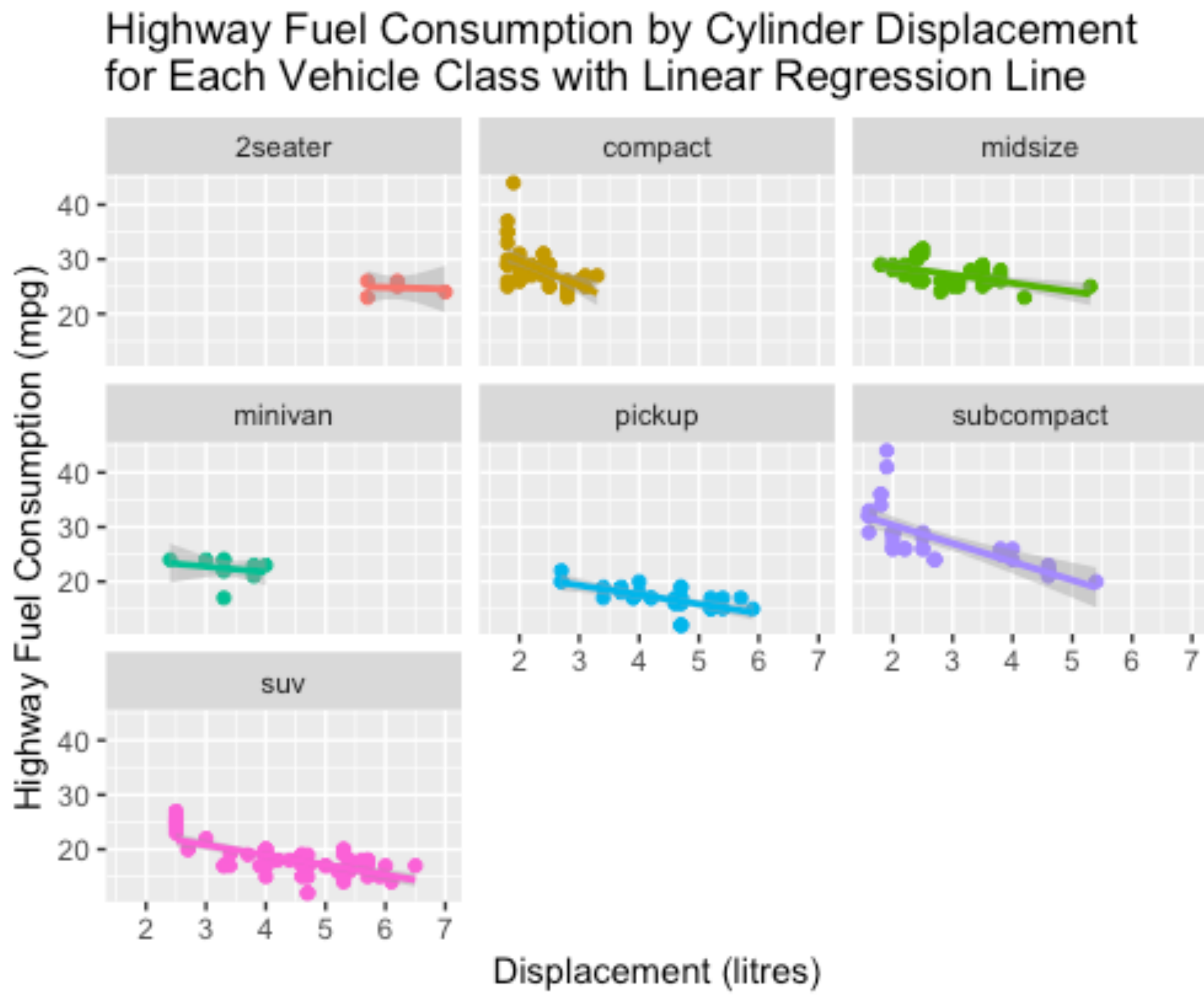



```
#build a violin plot with added descriptives
ggplot(mpg, aes(x = factor (cyl), y = cty, fill = factor (cyl))) + geom_violin() +
  guides(colour = FALSE, fill = FALSE) +
  stat_summary (fun.data = mean_cl_boot, colour = "black", size = .5) +
  xlab("Number of Cylinders") + ylab("City Fuel Consumption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

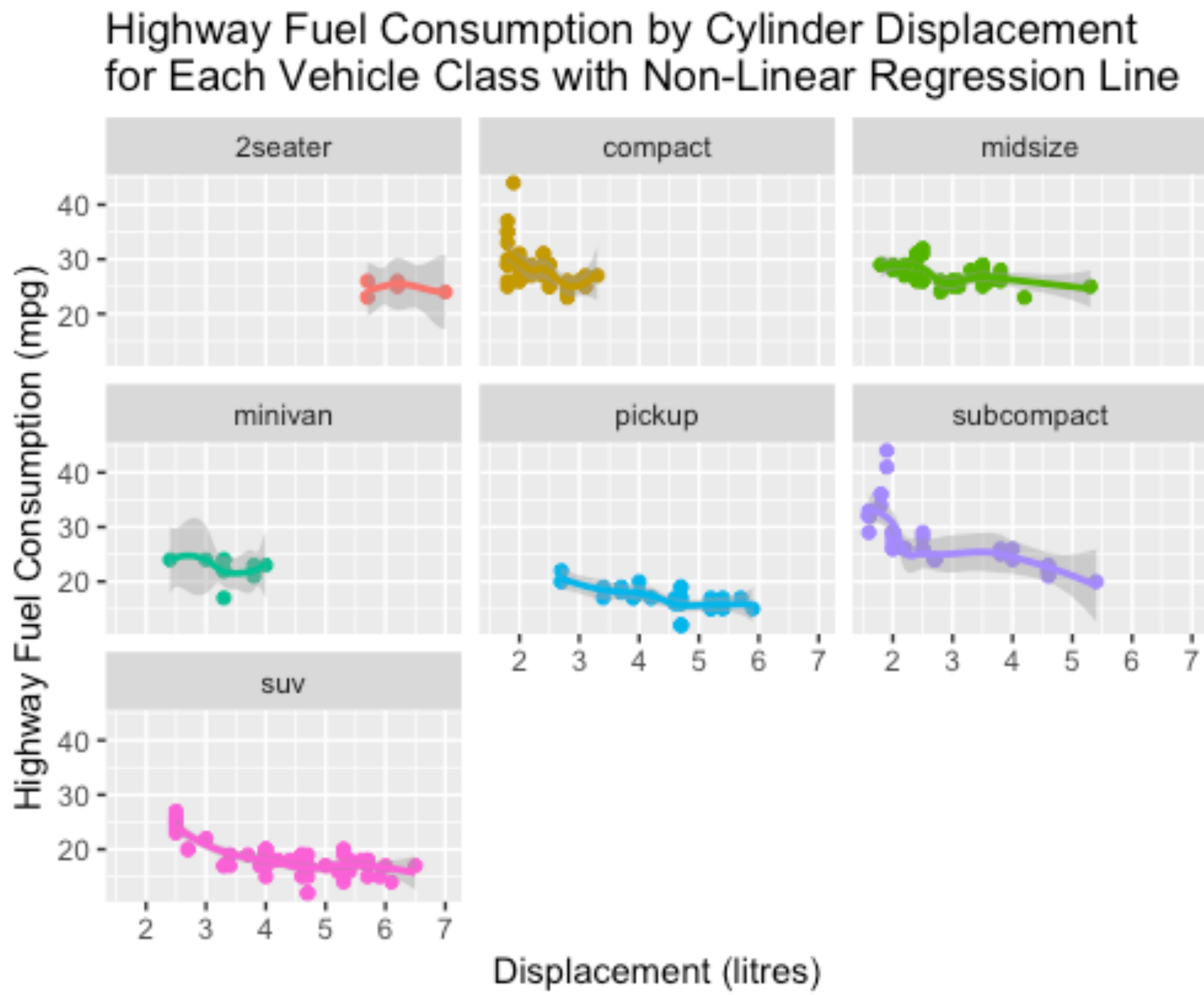



```
#facet wrap by vehicle class with displacement instead of cylinder number
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() + facet_wrap(~class) +
  guides(colour = FALSE) + xlab("Displacement (litres)") + ylab("Highway Fuel Consumption
(mpg)") +
  ggtitle ("Highway Fuel Consumption by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

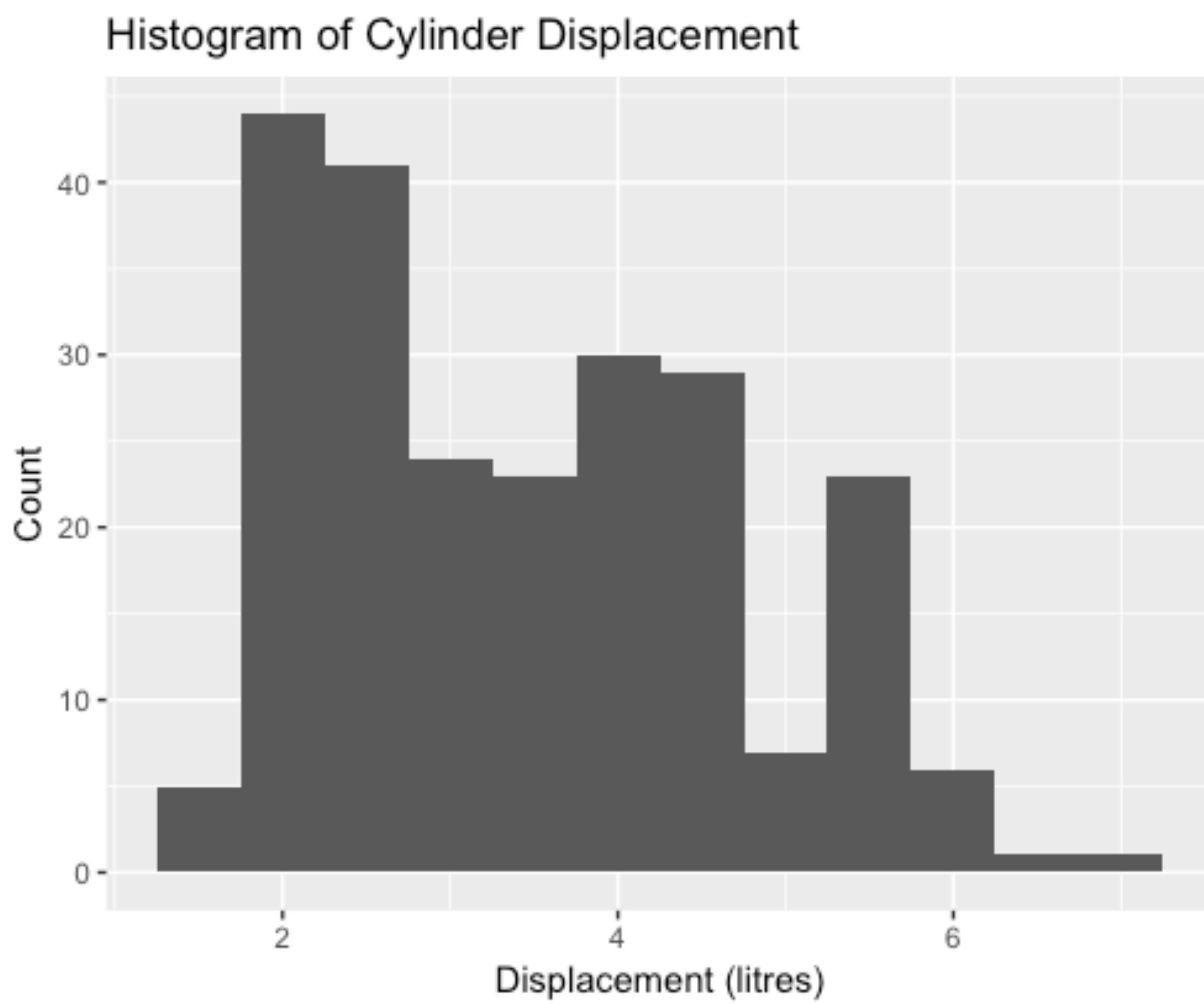
Note I haven’t explicitly used x = or y = here…



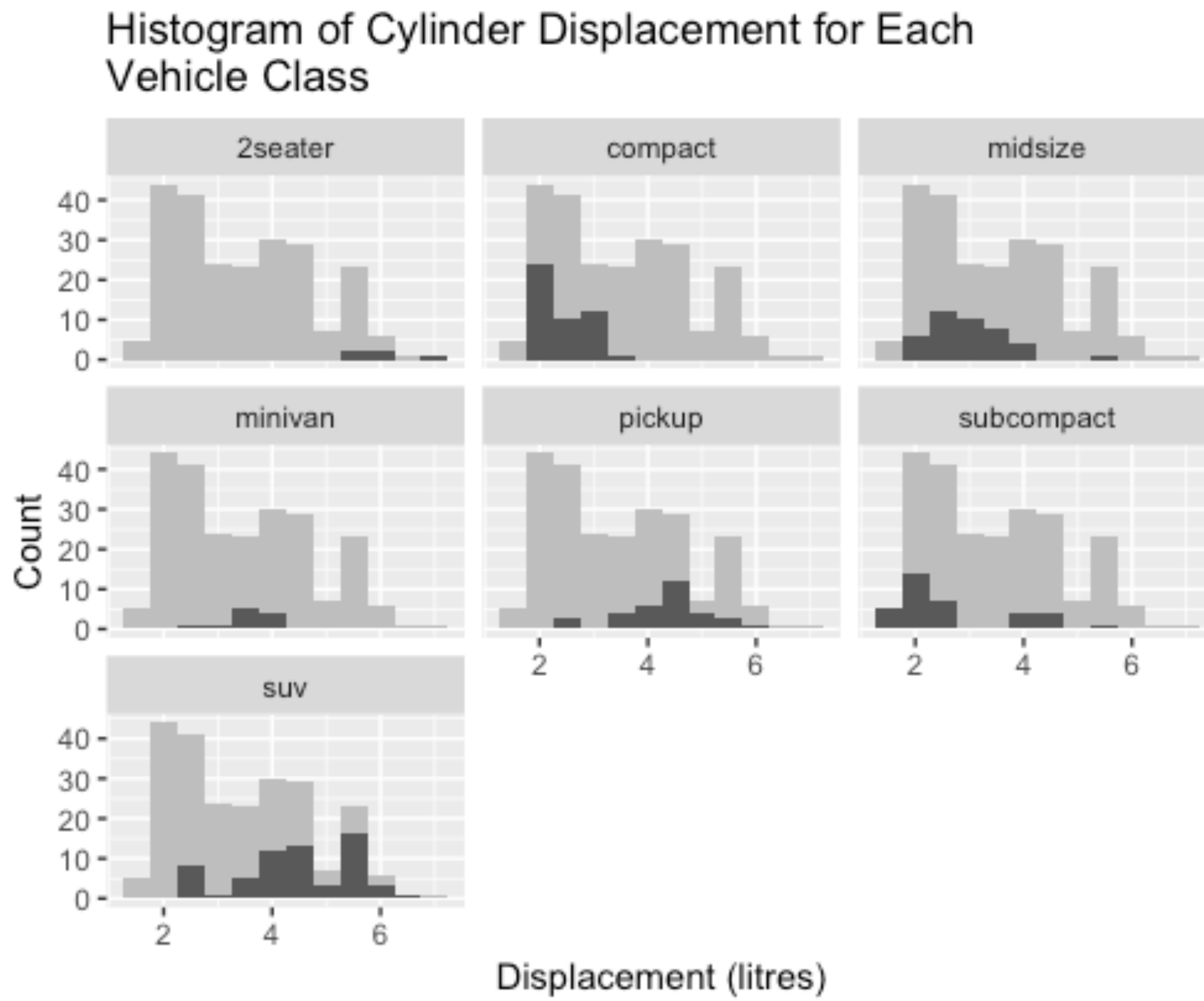

```
#now add a linear function to each
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() + facet_wrap(~class) +
  guides(colour = FALSE) + geom_smooth(method = "lm") +
  xlab("Displacement (litres)") + ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Highway Fuel Consumption by Cylinder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



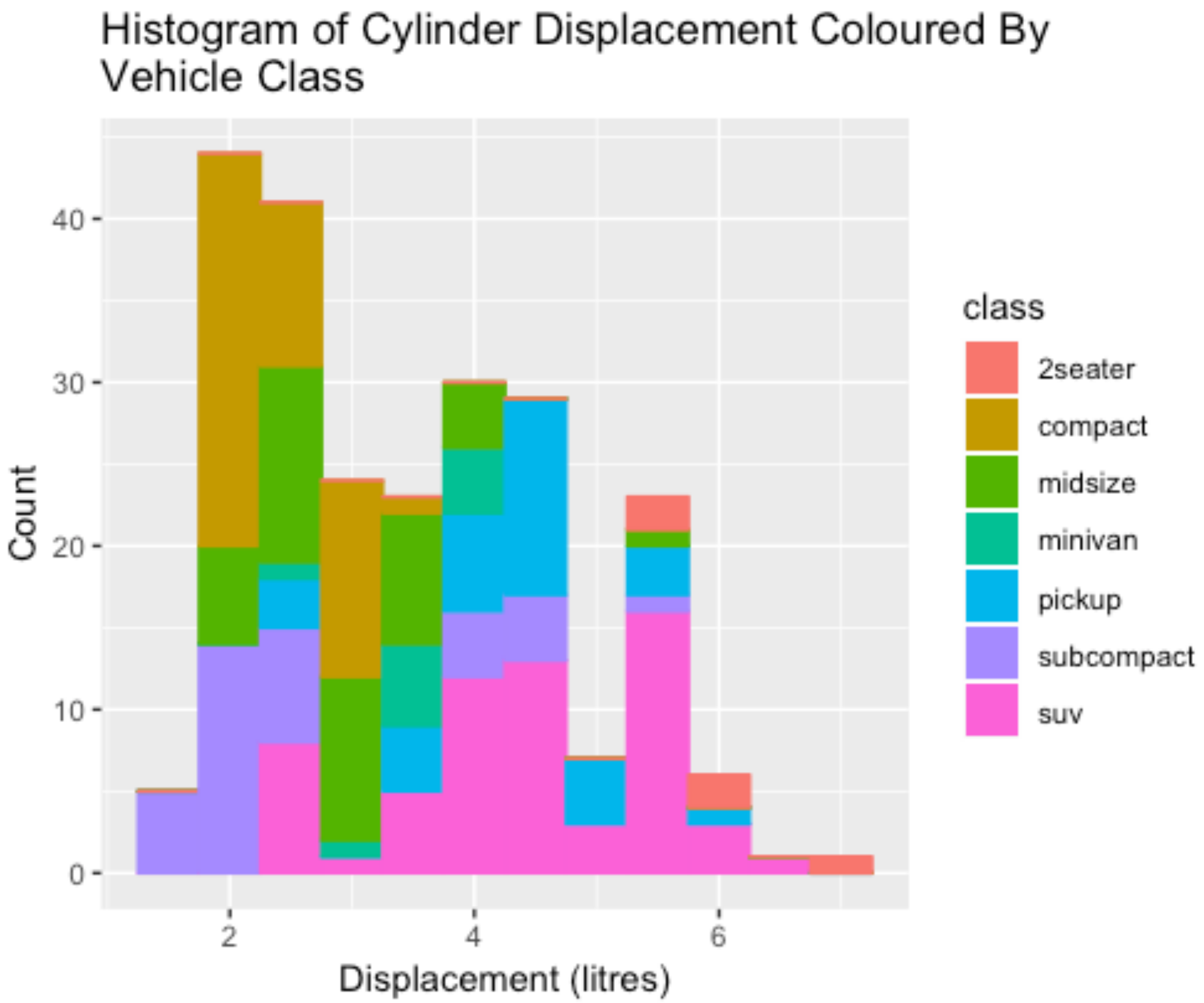

```
#now with a non-linear function to each
ggplot(mpg, aes(displ, hwy, colour = class)) +
  geom_point() + facet_wrap(~class) +
  guides(colour = FALSE) + geom_smooth(method = "loess") +
  xlab("Displacement (litres)") + ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Highway Fuel Consumption by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



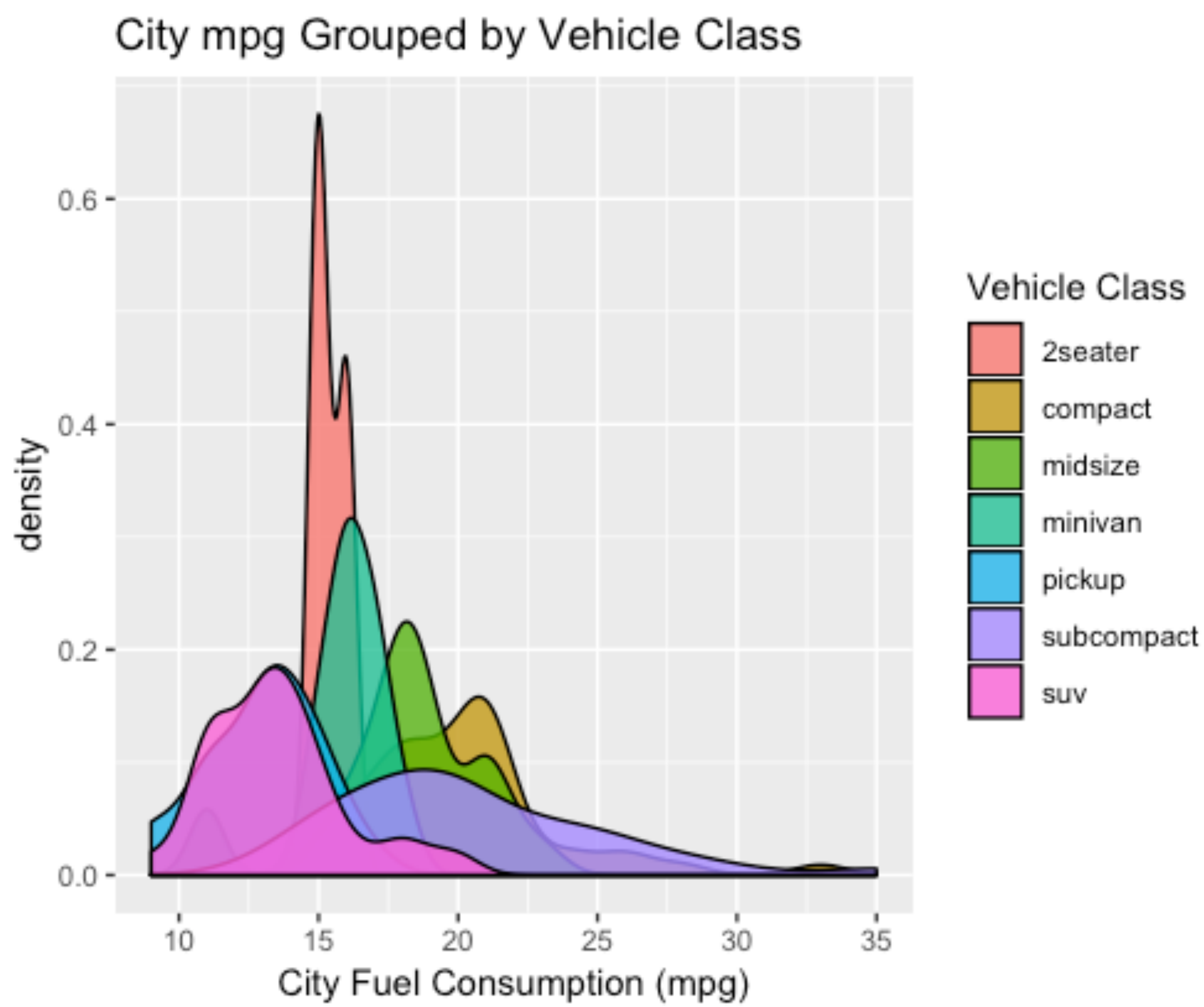

```
#plot basic histogram
ggplot(mpg, aes(displ)) +
  geom_histogram(binwidth = .5) +
  guides(colour = FALSE) +
  xlab("Displacement (litres)") + ylab("Count") +
  ggtitle ("Histogram of Cylinder Displacement")
```



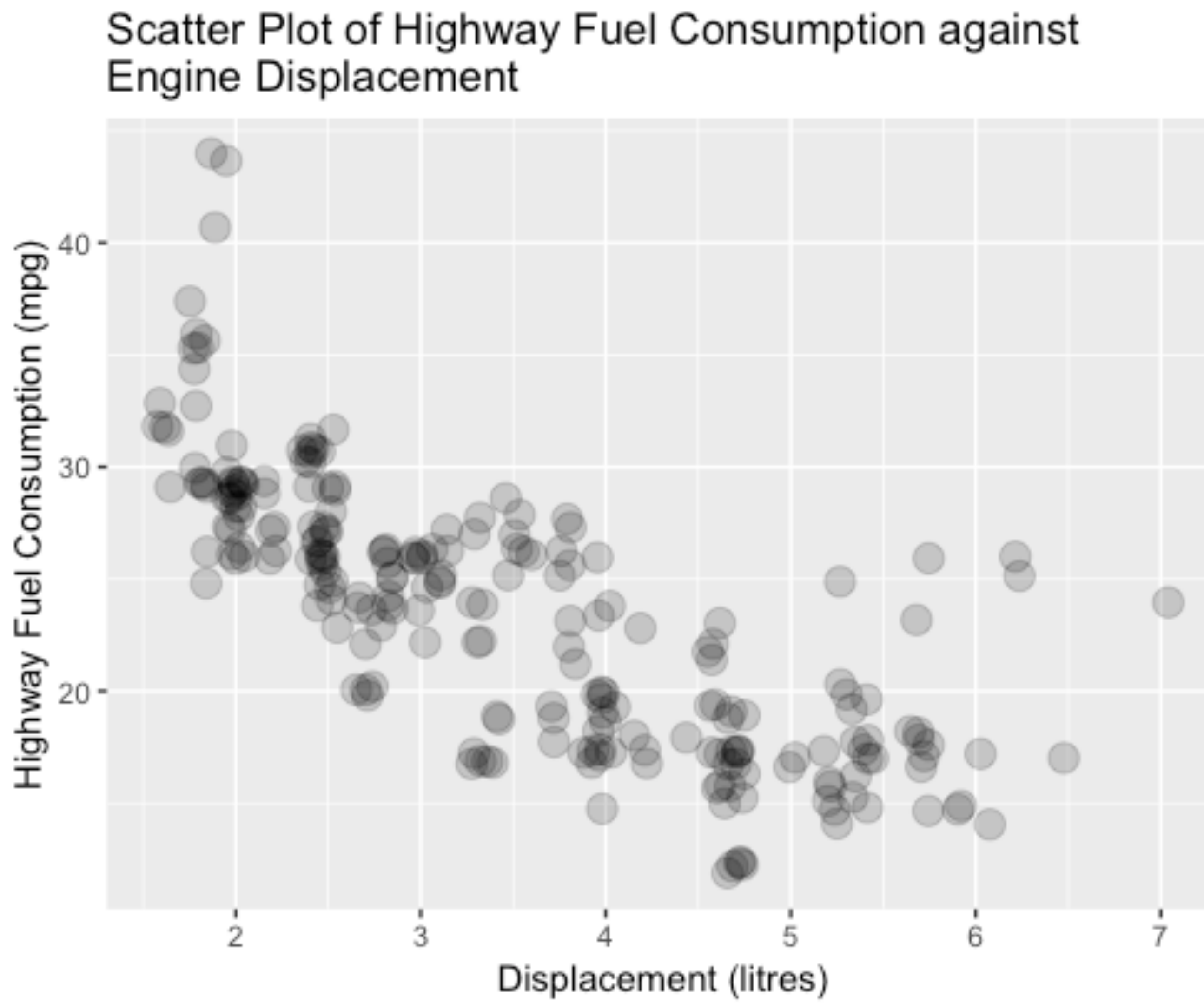

```
#facet by class and include background histogram of all data in each facet
mpg1 <- select (mpg, -class)
ggplot(mpg, aes(displ)) +
  geom_histogram(data = mpg1, fill = "grey", binwidth = .5) +
  geom_histogram(binwidth = .5) +
  guides(colour = FALSE) + facet_wrap (~class)  +
  xlab("Displacement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



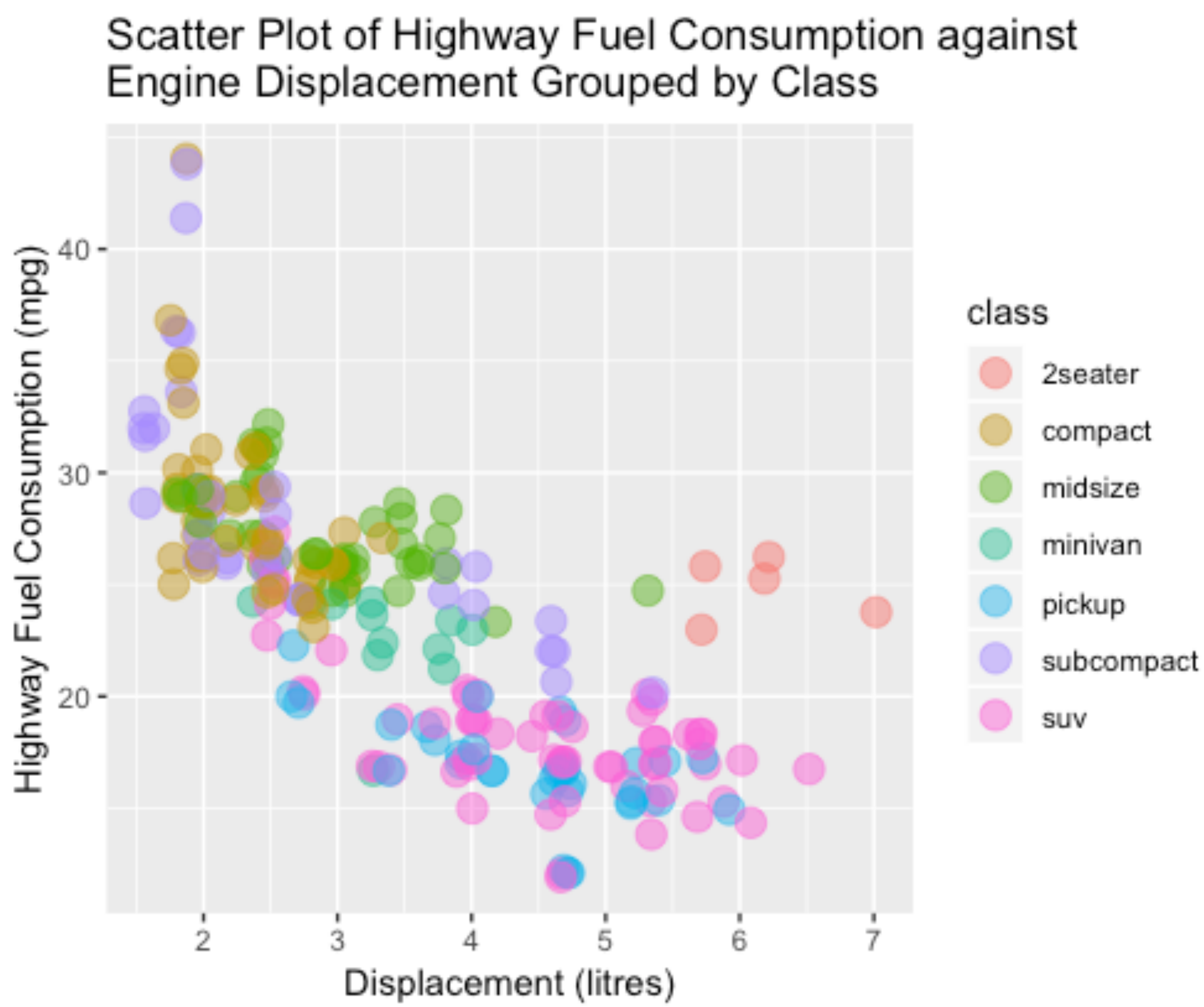

```
#plot histogram of displacement, coloured by class
ggplot(mpg, aes(displ, colour = class, fill=class)) +
  geom_histogram(binwidth = .5) +
  guides(colour = FALSE) +
  xlab("Displacement (litres)") + ylab("Count") +
  ggtitle ("Histogram of Cylinder Displacement Coloured By \nVehicle Class")
```



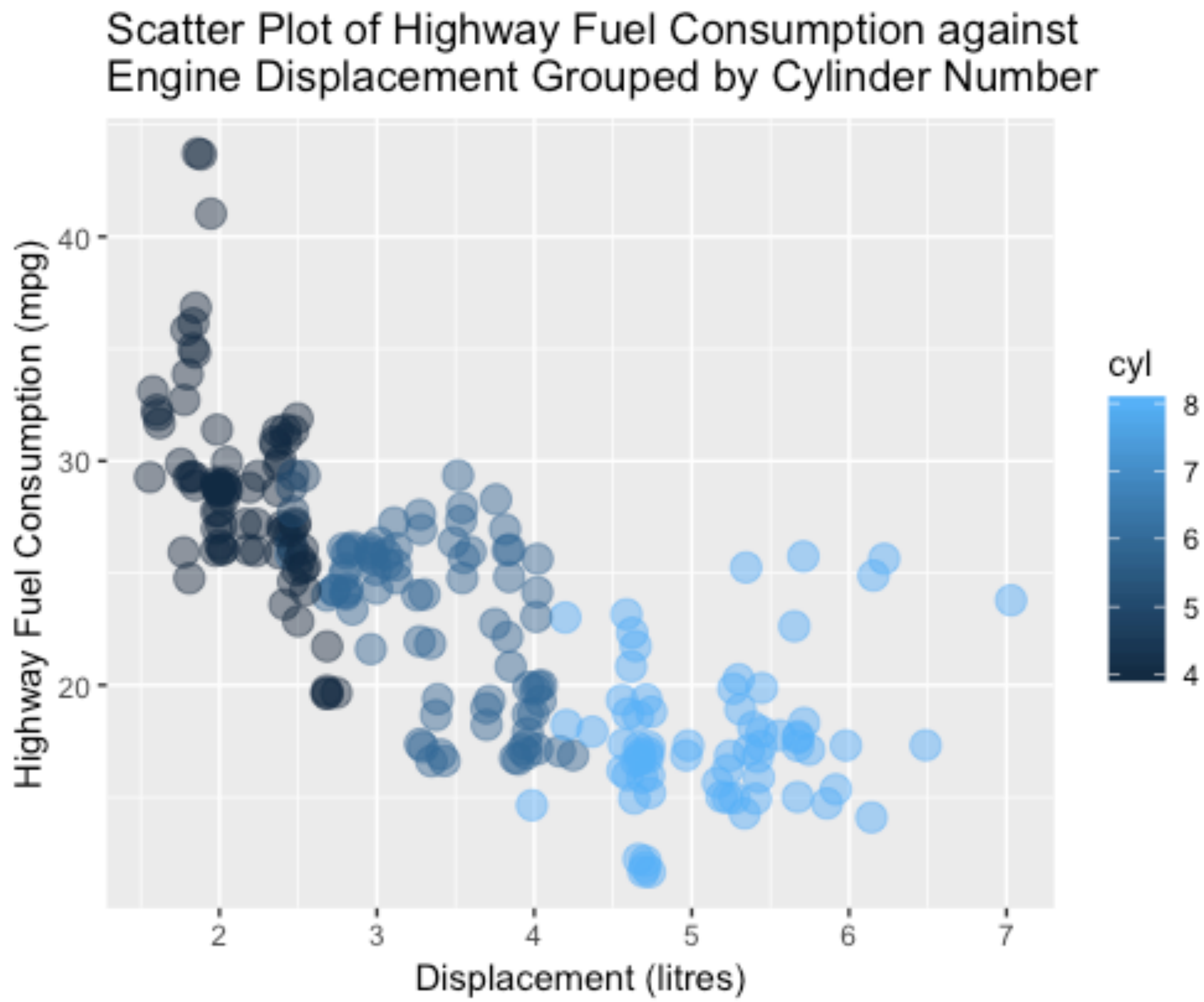

```
ggplot(mpg, aes(cty)) + geom_density(aes(fill = factor(class)), alpha = 0.8) +
  labs(title = "City mpg Grouped by Vehicle Class",
       x = "City Fuel Consumption (mpg)",
       fill = "Vehicle Class")
```



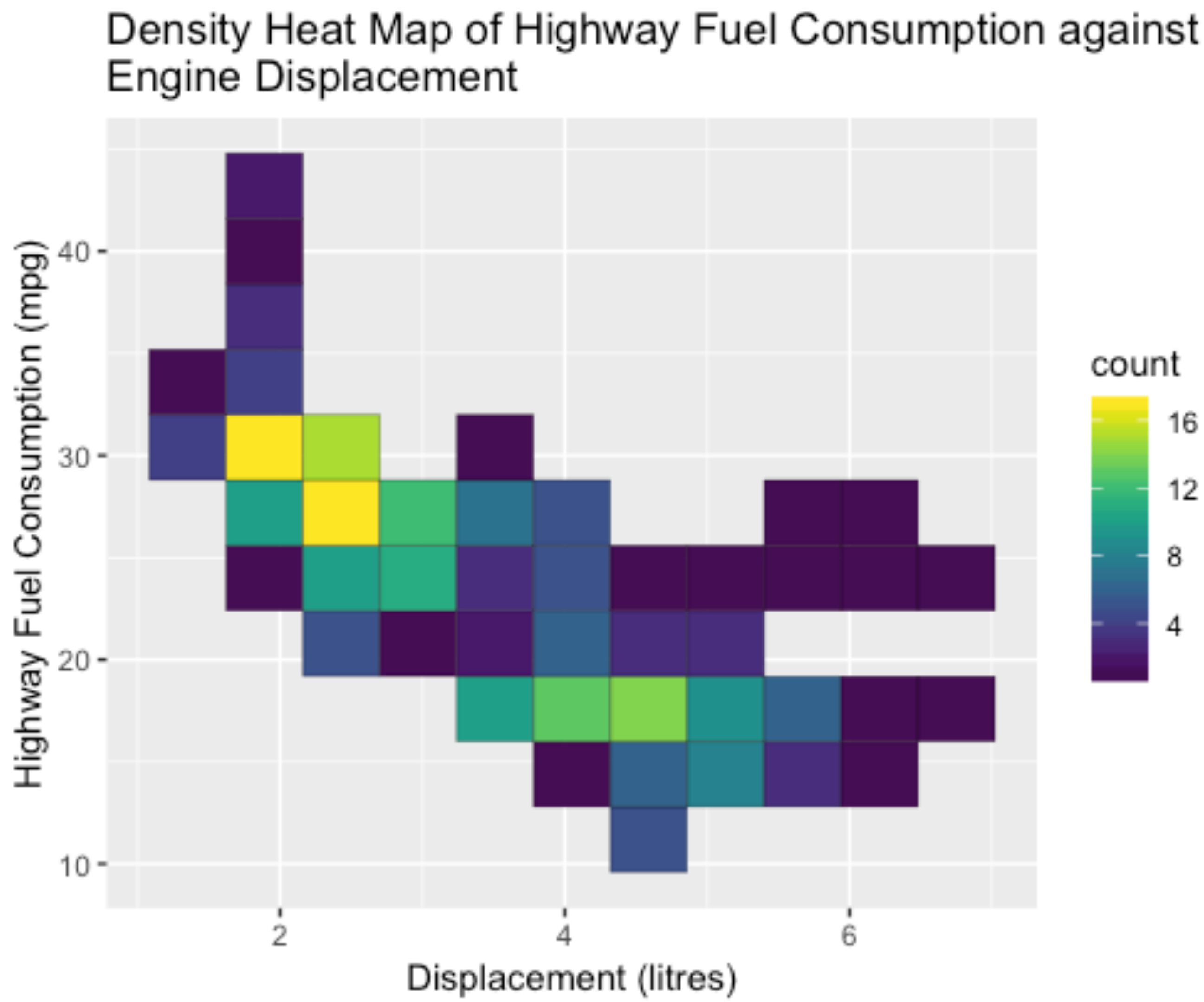

```
#scatterplot with jitter
ggplot(mpg, aes(displ, hwy)) + geom_jitter(width = 0.05, alpha = .2, size = 4) +
  xlab("Displacement (litres)") + ylab("Highway Fuel Consumption (mpg)") +
  ggtitle ("Scatter Plot of Highway Fuel Consumption against \nEngine Displacement")
```




```
ggplot(mpg, aes(displ, 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(displ, 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")
```




```r
#2d histogram with density heatmap
ggplot(mpg, aes(displ, 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

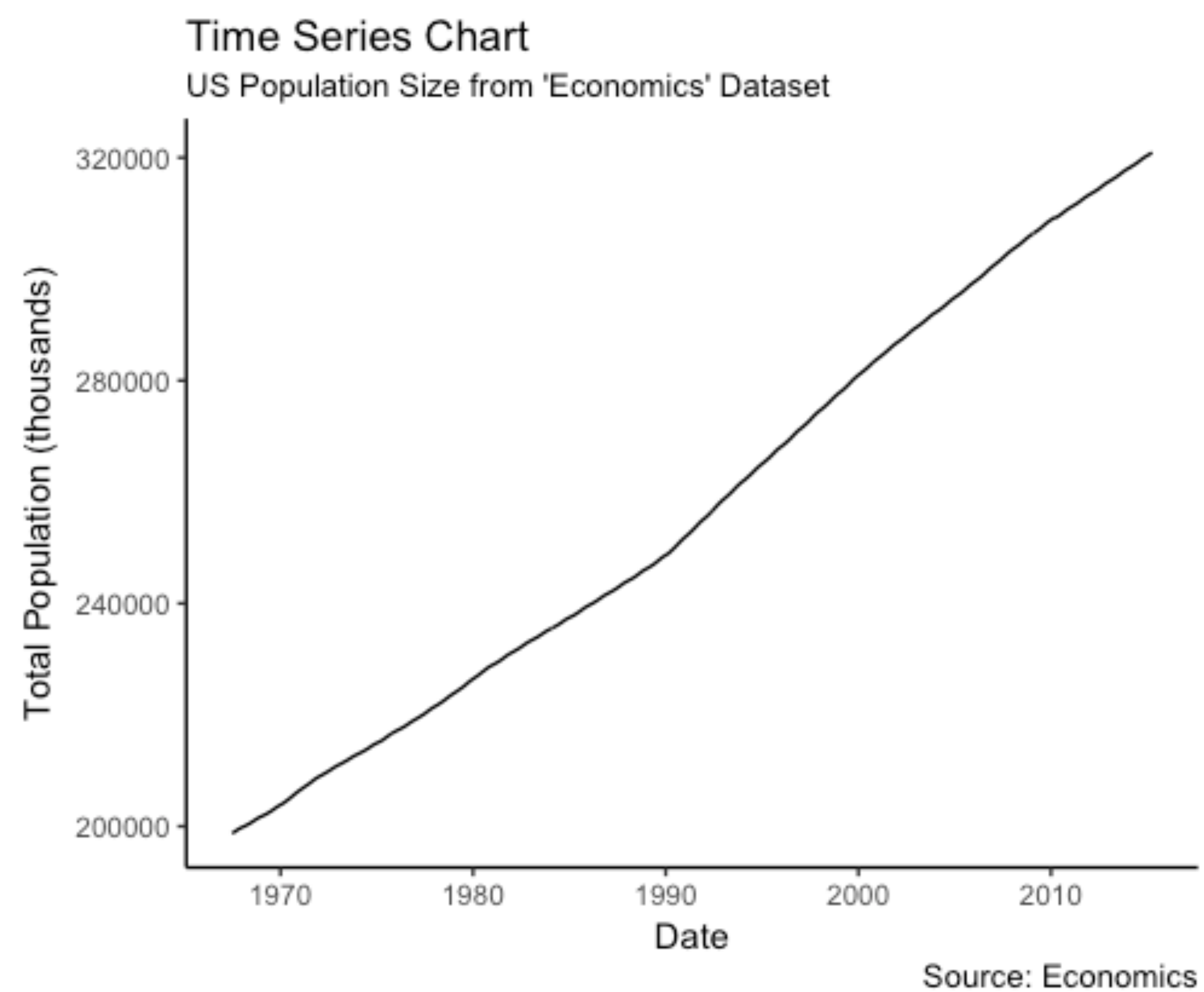
We're going to use the in-built dataset "Economics". This contains monthly data corresponding to US population size, personal savings rate, unemployment numbers (and much more)…

```
> 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.

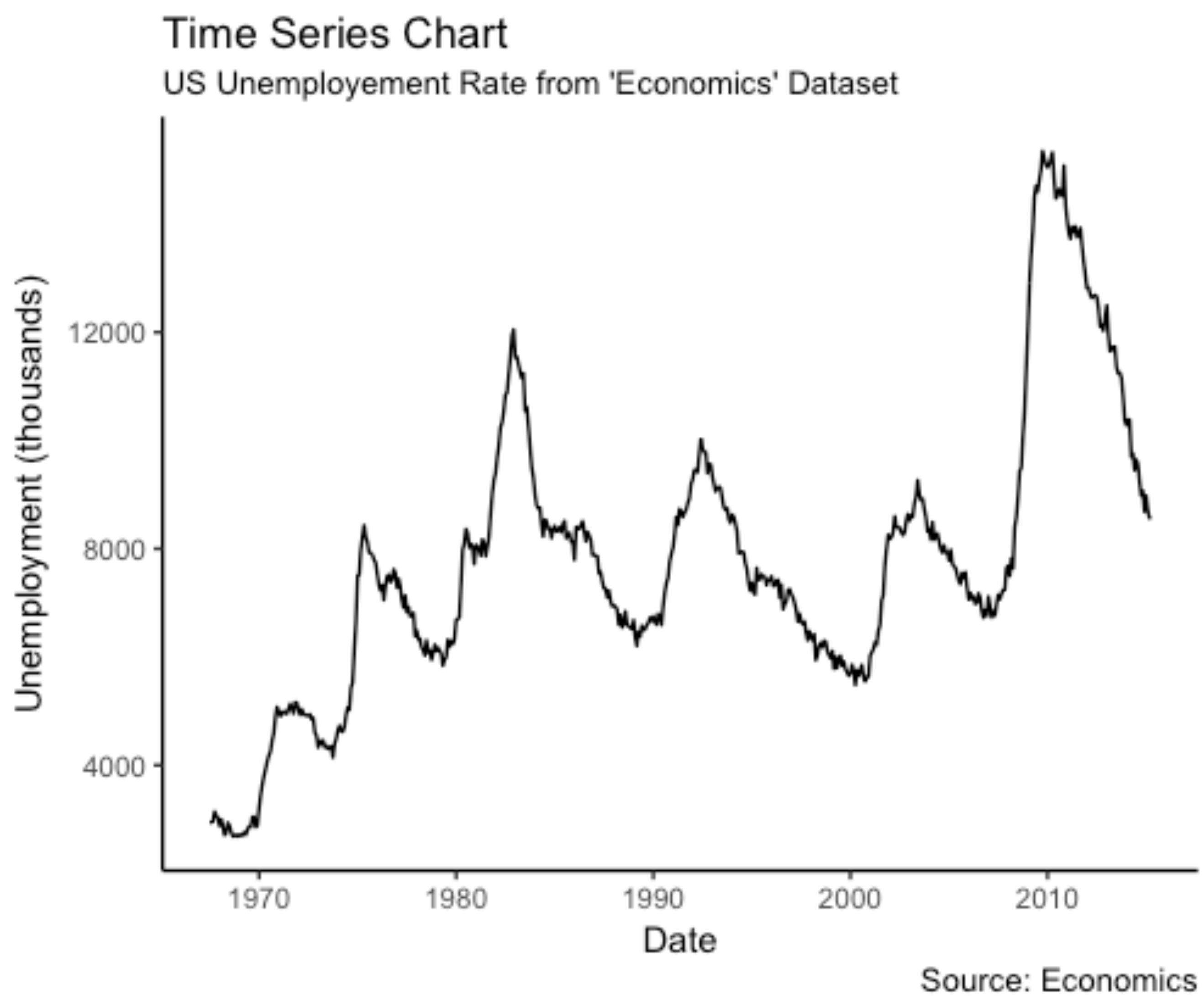
Plotting Time Series Data

```
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")
```



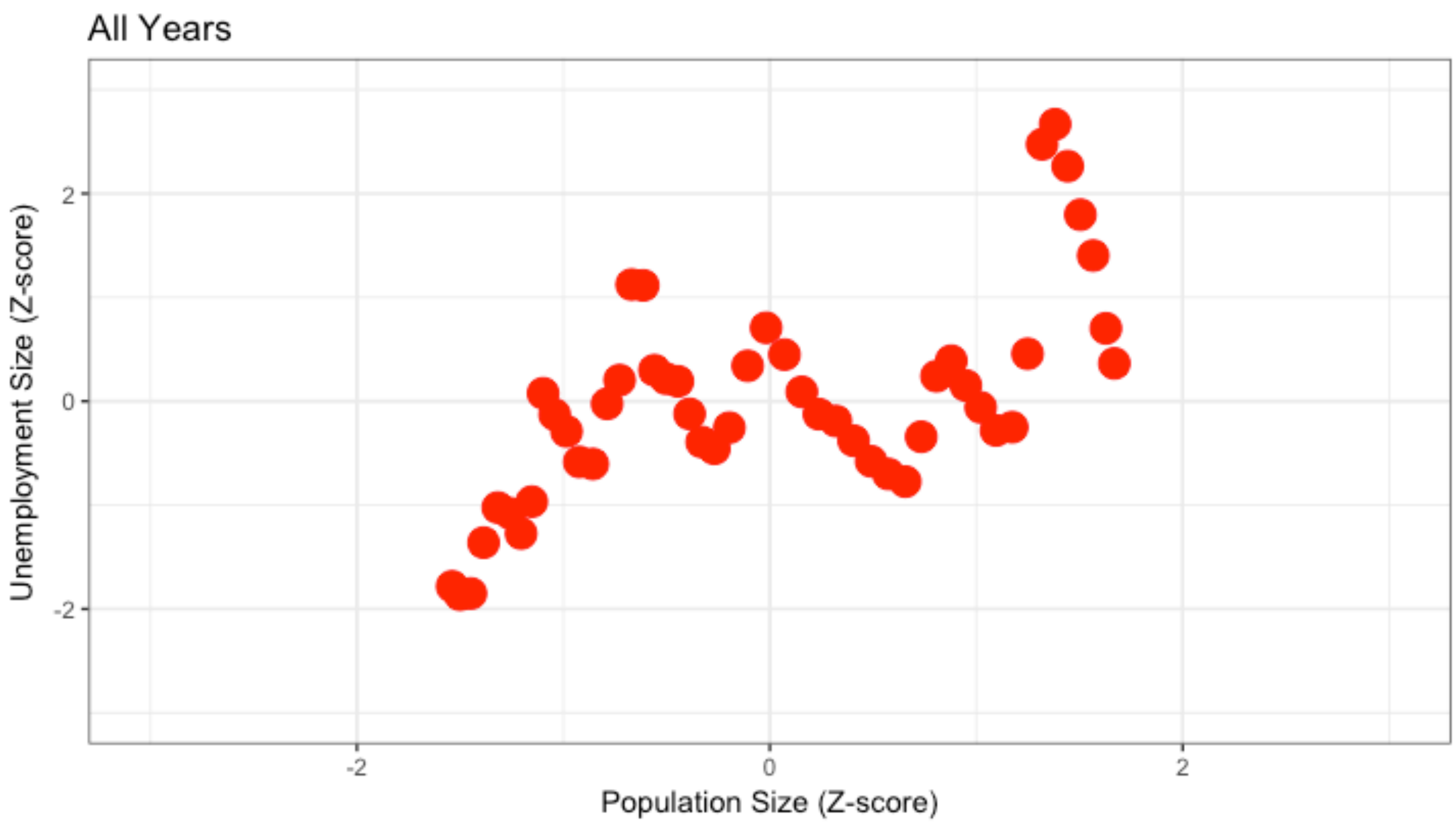
```
ggplot(economics, aes(x = date))
+ geom_line(aes(y = psavert)) +
  labs(title = "Time Series
Chart",
       subtitle = "US Savings
Rate from 'Economics' Dataset",
       caption = "Source:
Economics",
       y = “Savings Rate (%)",
x="Date")
```

```
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")
```



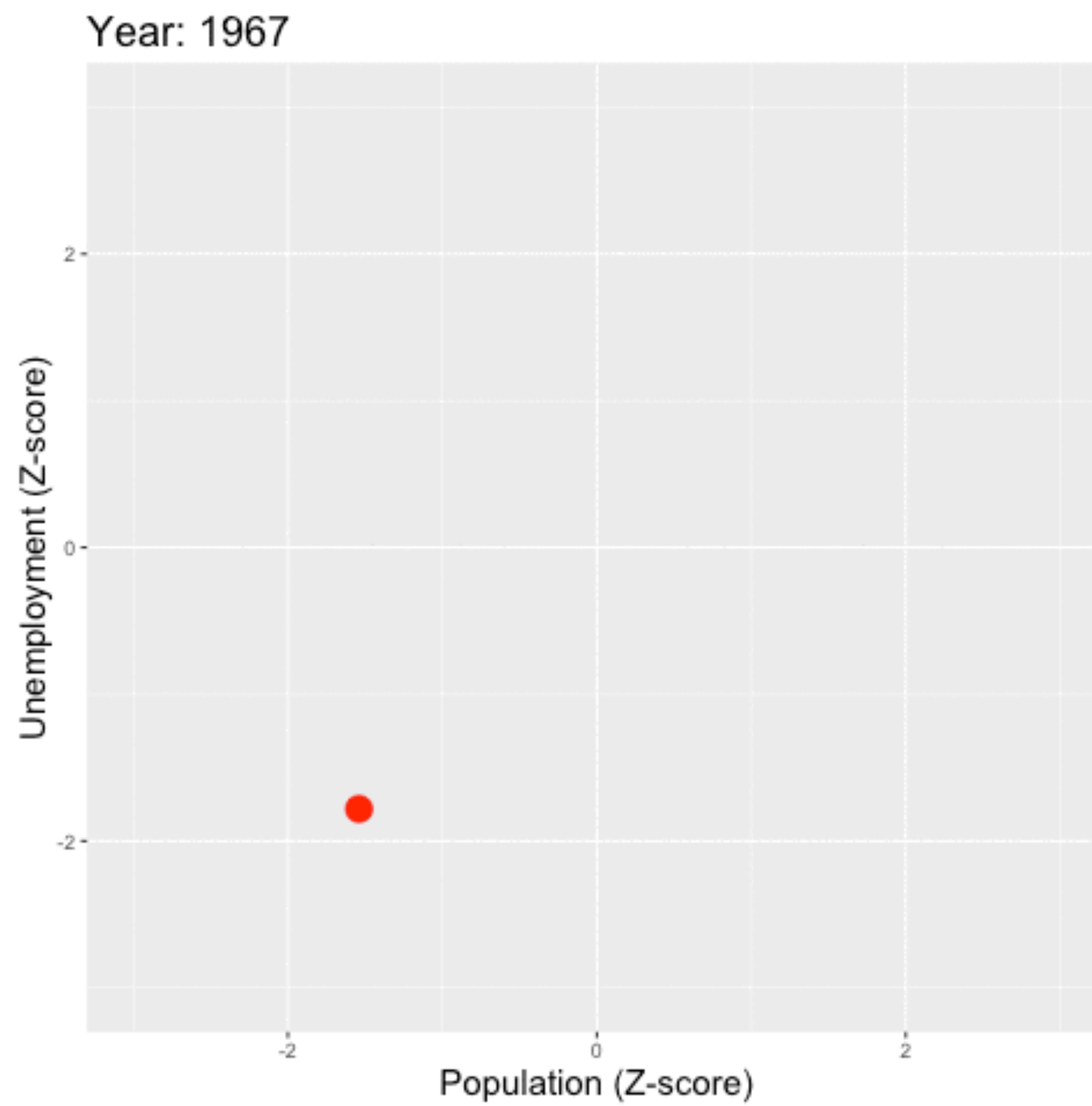
Animated Time Series Data

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



Animated Time Series Data

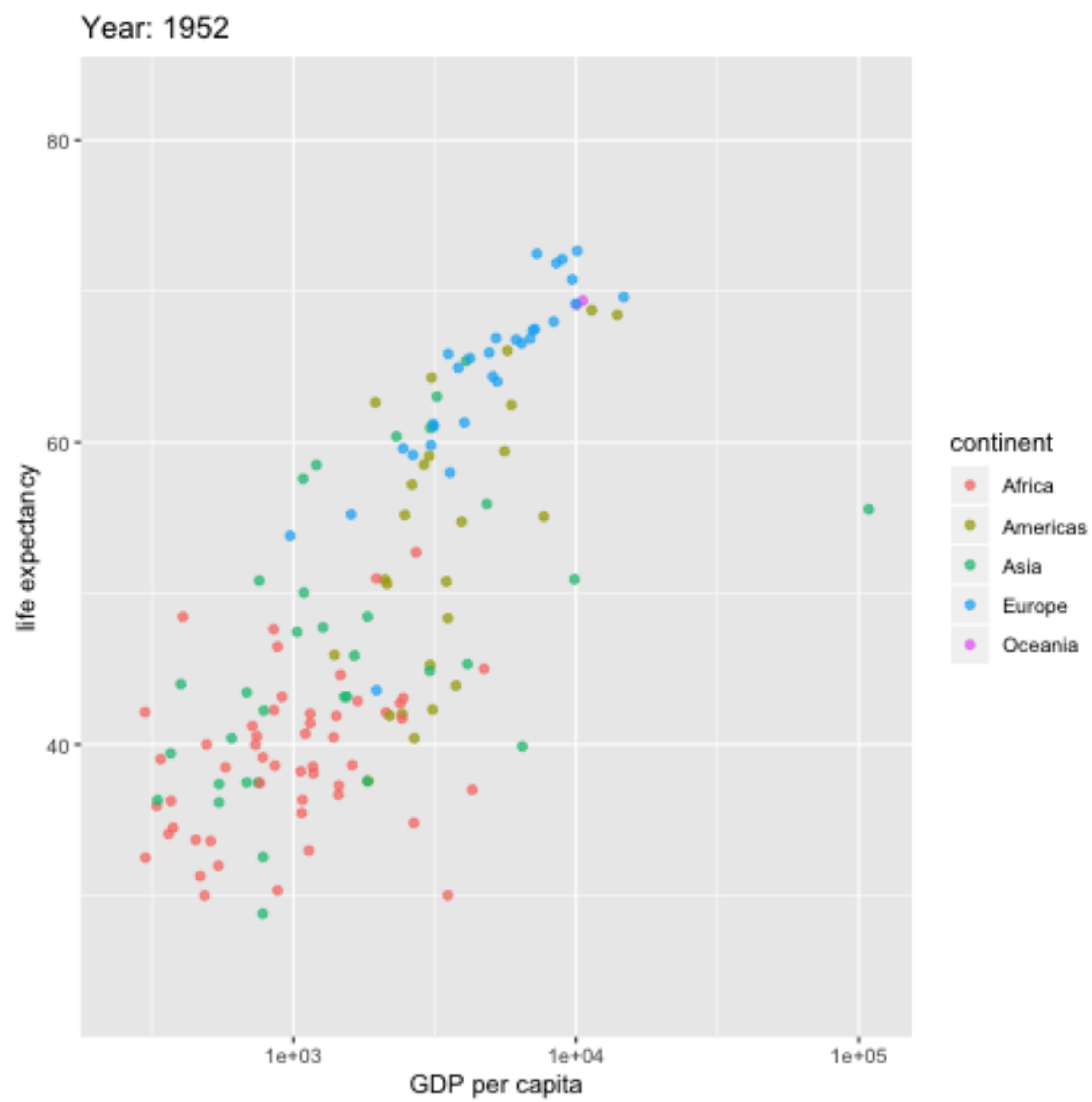
Now we're plotting Unemployment Size (transformed to Z-Scores) against Population Size (transformed to Z-scores) animated by Year.



Visualising Data with 4 Variables Simultaneously

Animated Time Series Data

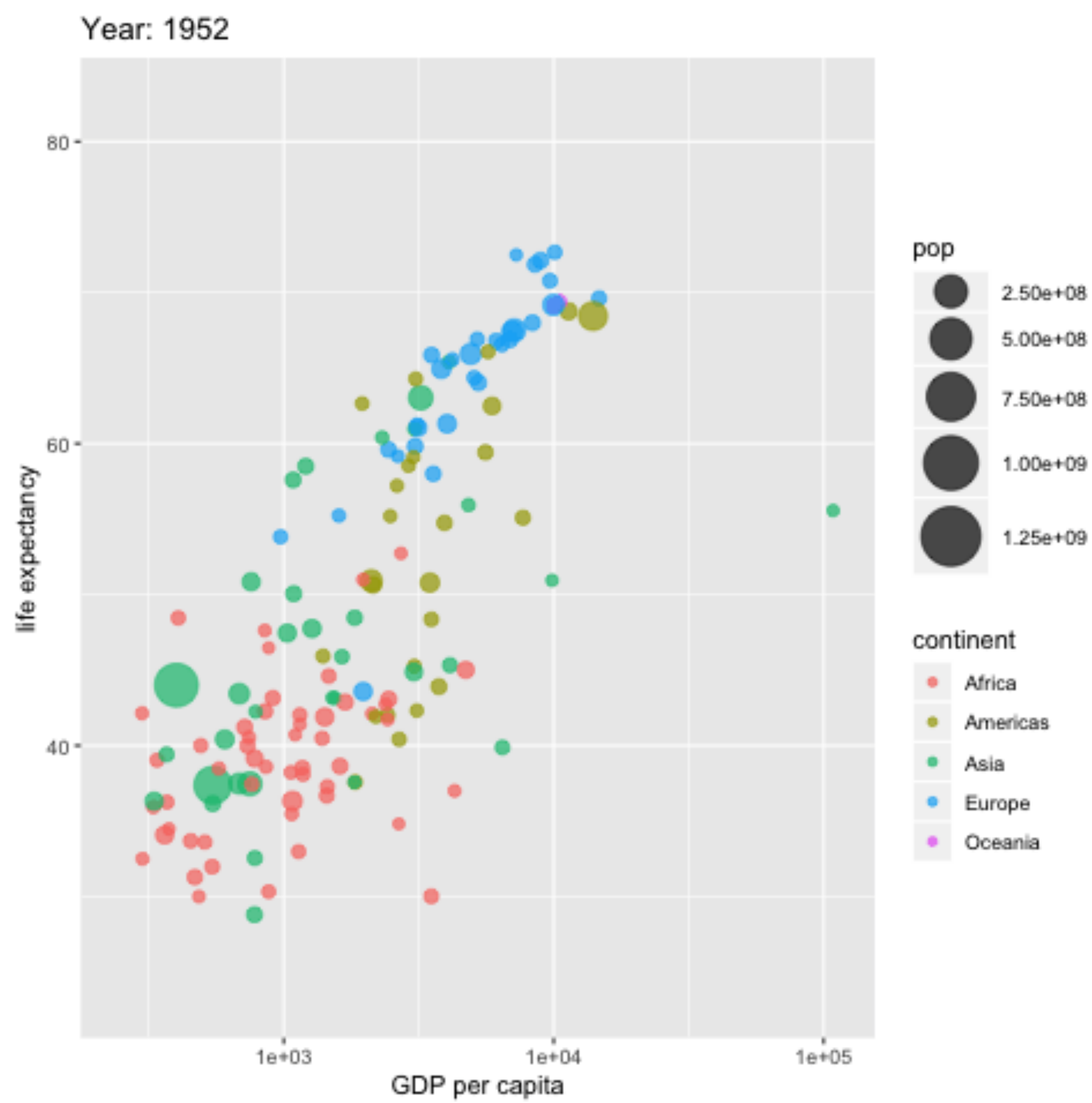
Life expectancy by GDP over time by Continent.



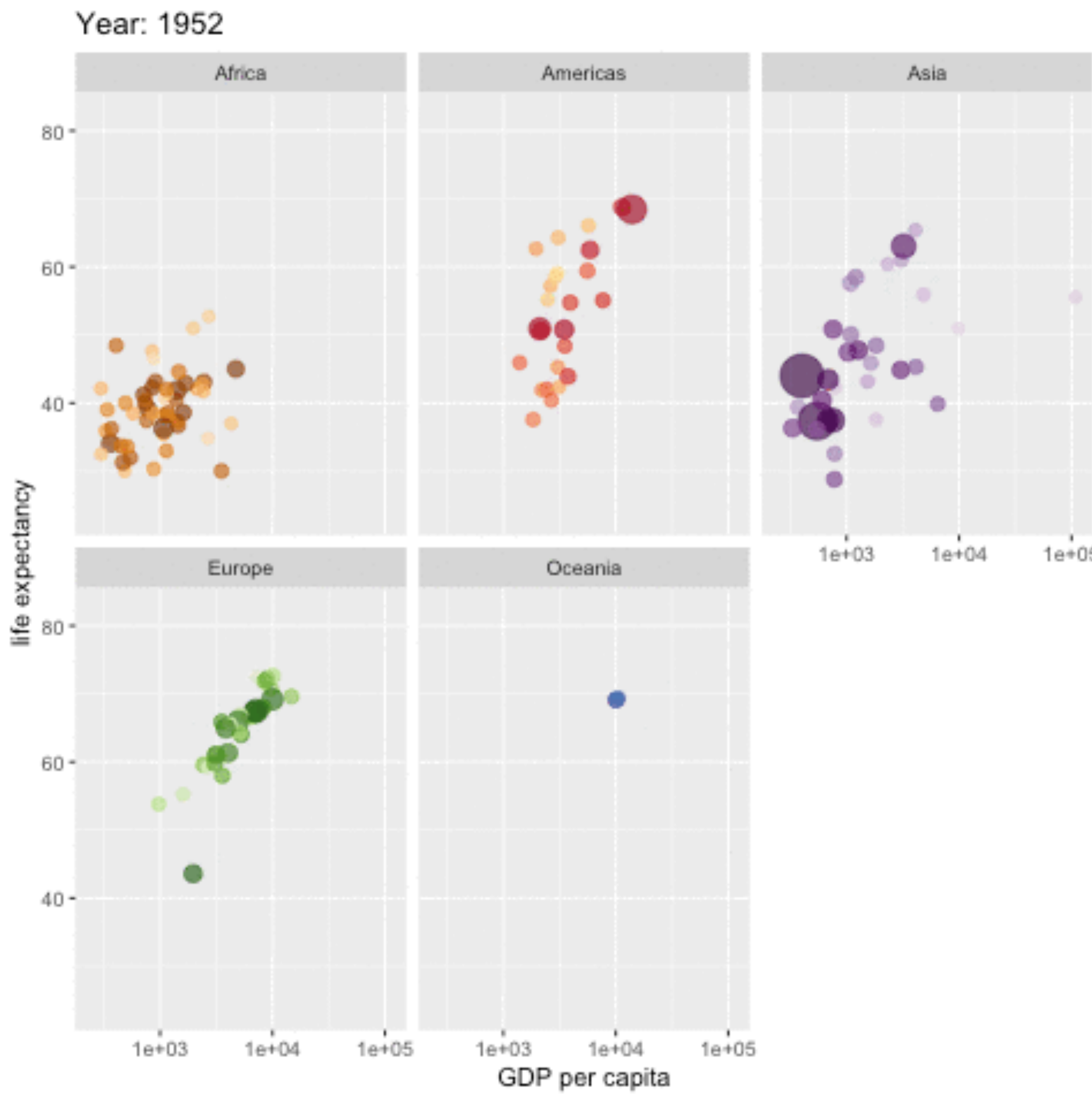
Visualising Data with 5 Variables Simultaneously

Animated Time Series Data

Now with a representation of population size.

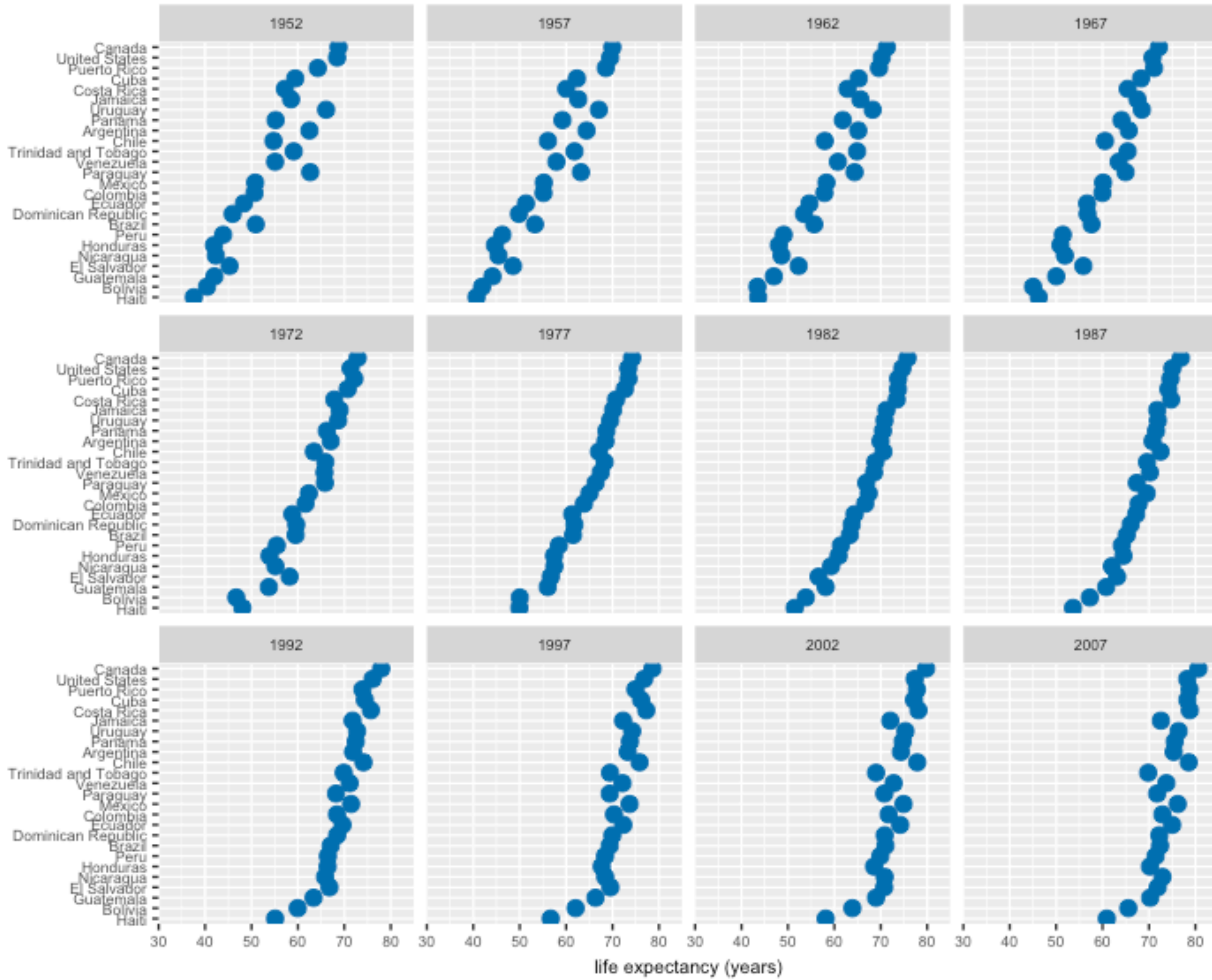


Separately by Continent

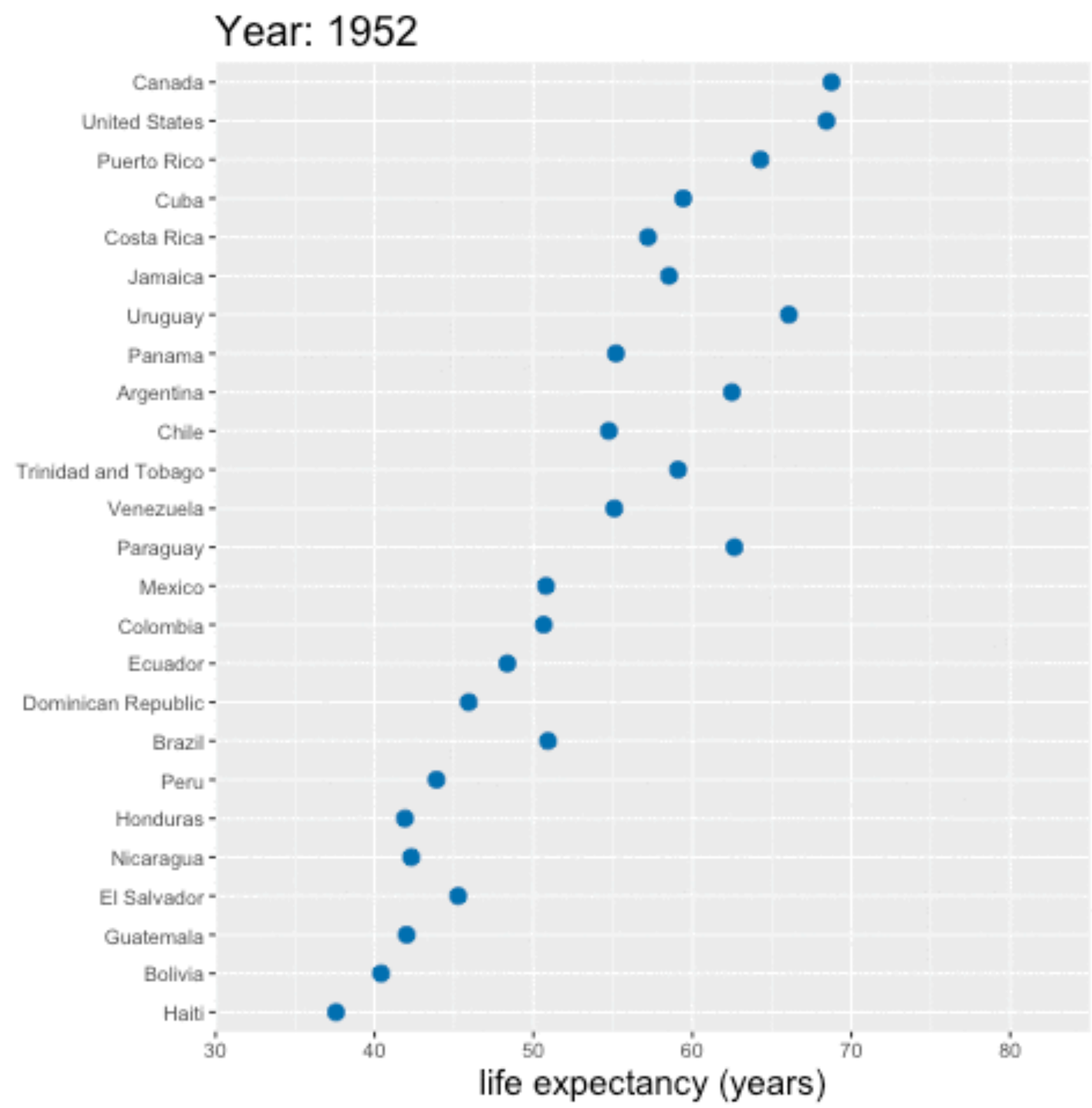


https://github.com/thomasp85/gganimate

Life Expectancy - Americas - Static



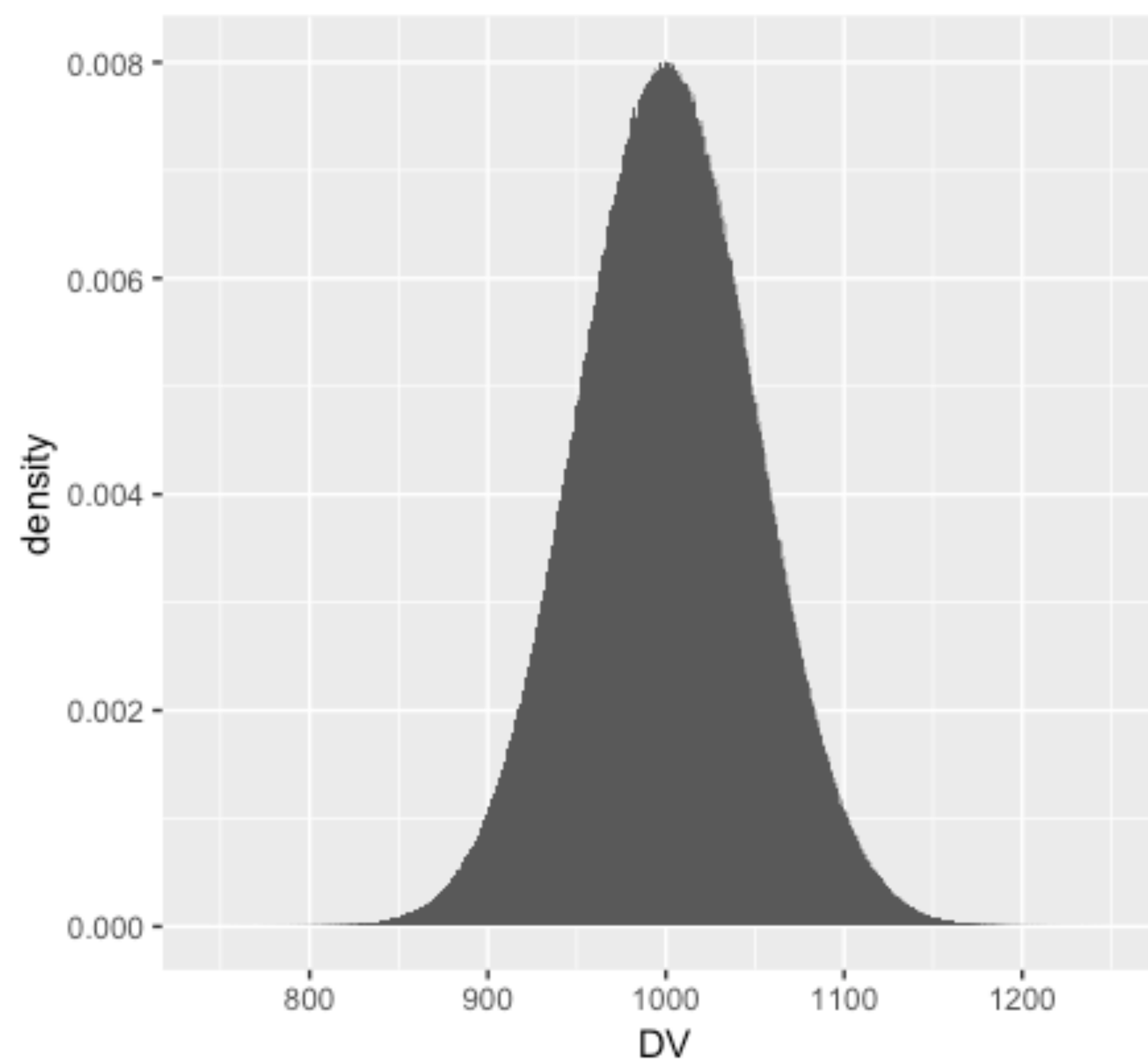
Life Expectancy - Americas - Animated



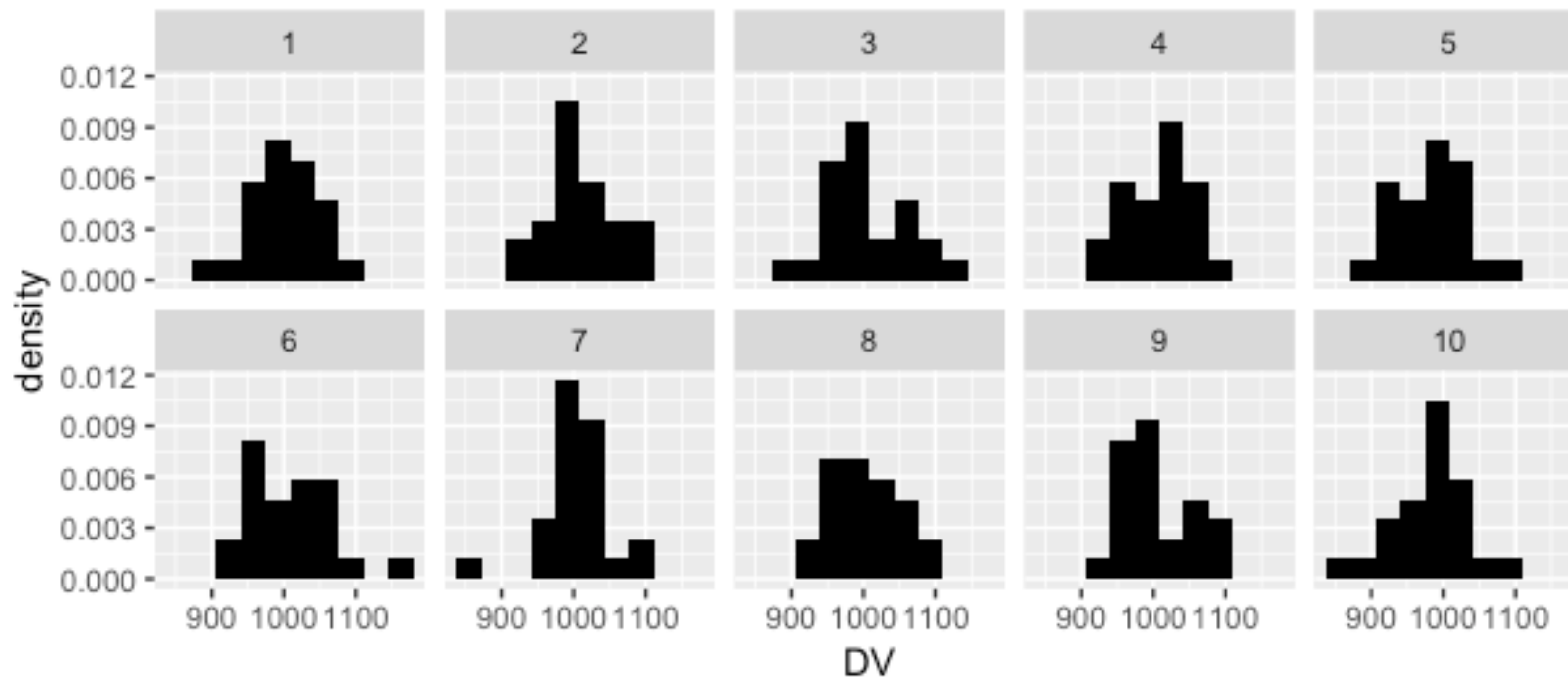
Although we have data only once every 5 years, the `gganimate` package interpolates between each census date to provide a smoother animation.

Using animation and data vis. to understand statistical concepts

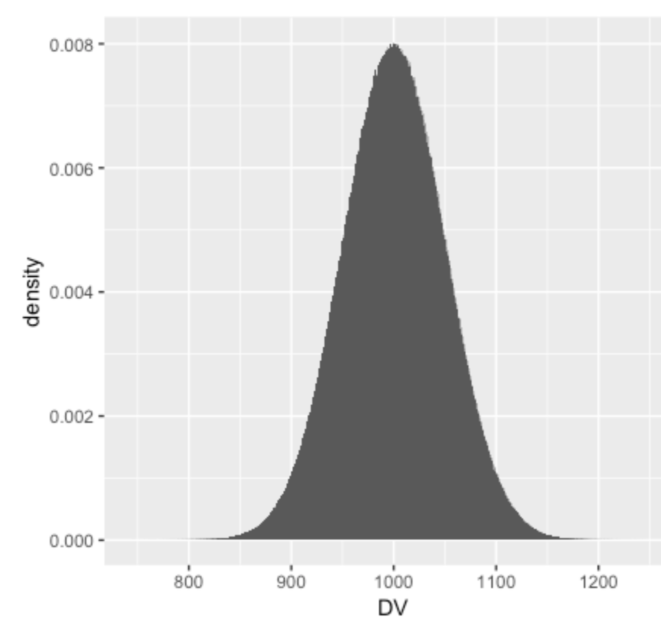
When we sample from a population, we are taking a sample of data points from the population distribution - the population could look something like this:



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:

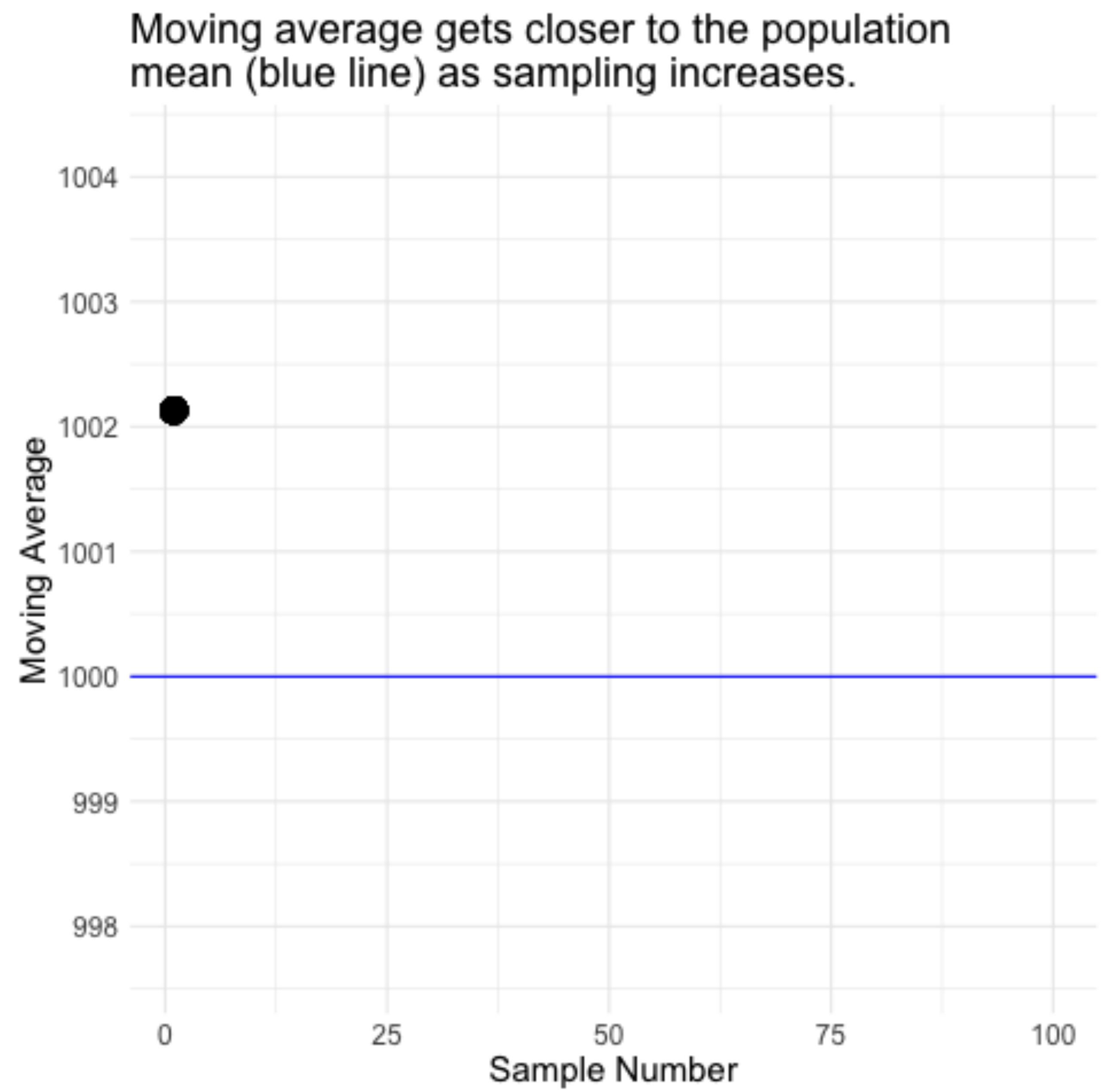


- When we take a sample from a population the mean of the sample may be quite different from the mean of the population (aka sampling error).

- 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 more samples (each with their own mean) we draw from the population, the closer we get to the true mean of the population.
- So, animation can be used not just to communicate information, but also principles…



The Key Question

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

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