DATA VISUALISATION

LAB WORKSHEET 1

Creator: Dr Mark Taylor

What is this document?

Welcome to Data Visualisation!

What happens in these lab worksheets?

Today, as with most weeks we're making a bunch of different graphs. This involves working with three main things:

- **R**
- **RStudio**
- **ggplot2**

Let's explain what each of these is:

R's a statistical programming language, which has been developed over several years by a huge range of people.

RStudio's what's called an "Integrated Development Environment" (or IDE) for R. In practice, this means that RStudio's a more manageable way to work in R.

ggplot2's a package for R. R packages allow R to do more different things than what you get when you download R on its own, and there are thousands of them; different people install different packages based on their own needs.

I know these terms are a bit dry, and might seem a bit confusing. As we go through the lab worksheets, we will go into more depth and detail.

Q-Step A step-change in quantitative social science skills Funded by the Nuffield Foundation, ESRC and HEFCE
Nuffield Foundation
Sheffield Methods Institute.
The University Of Sheffield.

Before We Start...

Get R and RStudio installed on your own machine. You can install R here (https://cloud.r-project.org/); if you're on a Windows machine click "Download R for Windows", if you're on a Mac click "Download R for (Mac) OS X". You can install RStudio here: https://www.rstudio.com/products/rstudio/download/. Click the first "Download" button, under "RStudio Desktop".

Please install R first and RStudio second. (It can cause some problems if you don't.)

Let's draw some graphs

Let's start drawing some graphs!

The way this handout works is that when you see plain text in a box like the one below:

```

```

You should type it into the top left pane exactly as written, and hit Ctrl-Enter (or the "Run" button - or Cmd- Enter if you're on an Apple machine).

-**Please type out the commands rather than copy-pasting them**. This will help you learn all of the small steps that we do in ggplot, and it will help you catch the spelling mistakes that happen so often when we learn to program and do new things. It's very common to think you've copied something exactly as written, but made a small error in transcription – this is totally normal, don't panic if it happens and just re-read your typing carefully.

It's likely that we'll need to install some packages before we get any further. To start, let's type and run the following:

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

What this does is install the tidyverse, which is a wide range of different packages, many of which we'll be using in these worksheets.

Once it is installed - and this process might take a couple of minutes - please load it by typing the following:

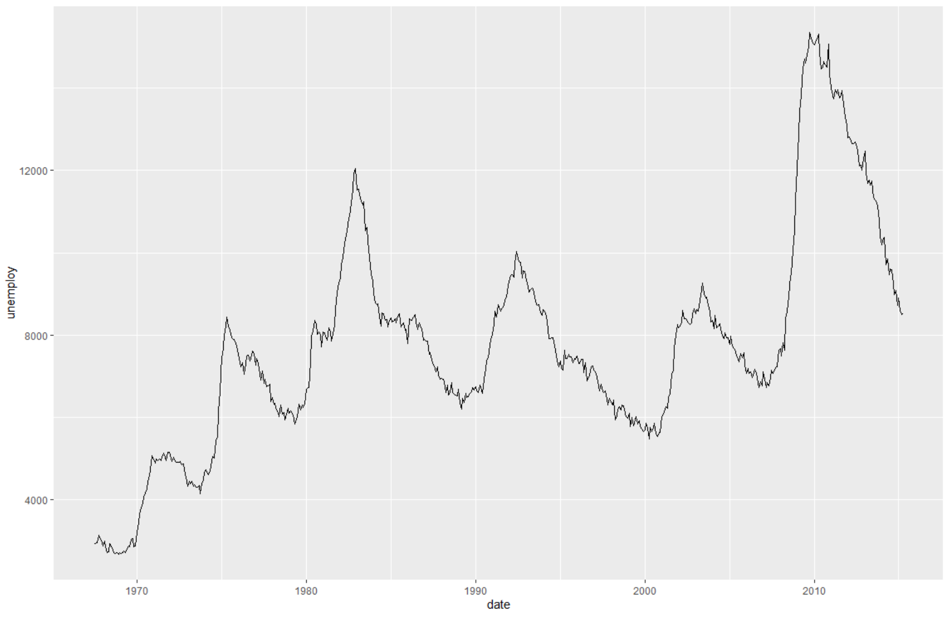
```
library(tidyverse)
```

By running the above command, we’ve loaded the tidyverse, which includes ggplot2, the main package we’ll be using this semester. By loading the ggplot2 package, we’ve also loaded a bunch of different datasets (I’ll explain this more in the coming worksheets). So, let’s draw some graphs.

1. The economics data contains some information about, amongst other things, the number of people in the US who were unemployed over time. Let's make a graph of how it's changed.

```
ggplot(data = economics) +
  aes(x = date, y = unemploy) +
  geom_line()
```

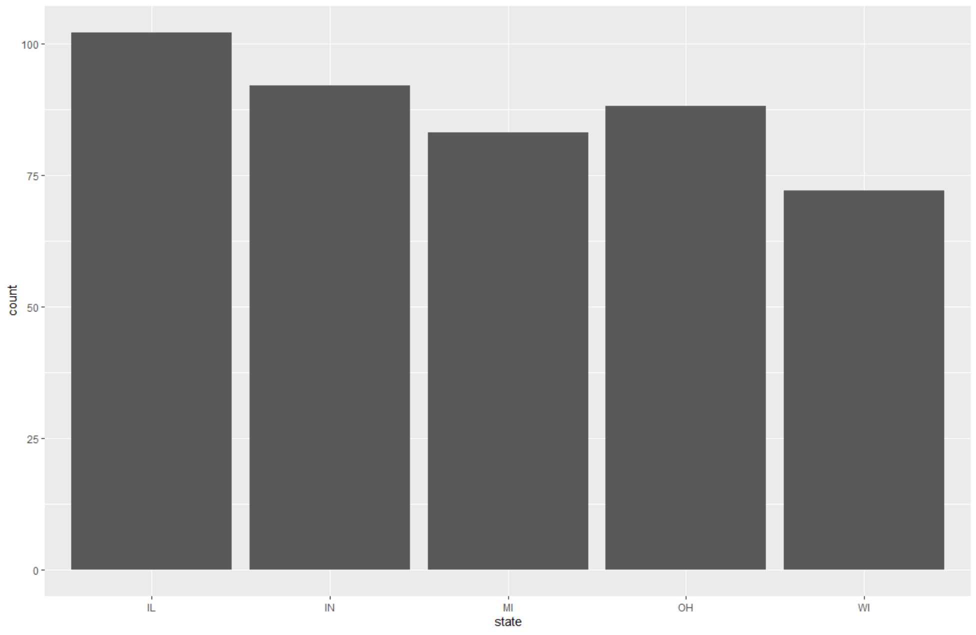
This is what it should look like:



2. The midwest data contains some information about different counties in the states of the Midwest of the US. Let's make a graph of how many counties there are in each state in the Midwest.

```
ggplot(data =
  midwest) +
  aes(x = state)
  + geom_bar()
```

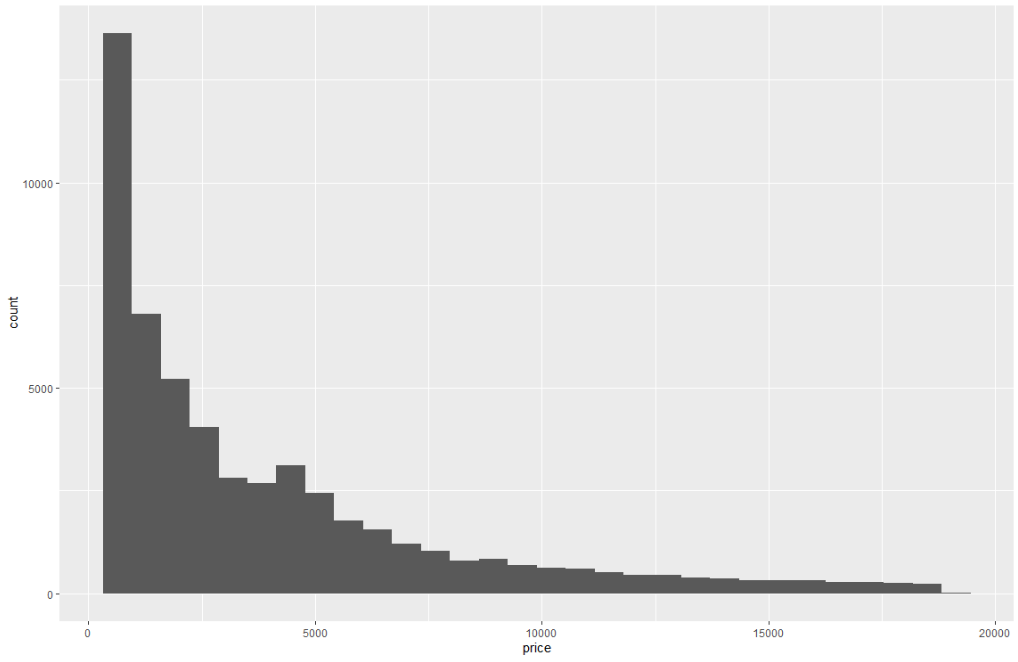
This is what it should look like:



3. The diamonds data contains information about diamonds sold in the US. Let’s make a graph of how much they each sold for.

```
ggplot(data = diamonds) +
  aes(x = price) +
  geom_histogram()
```

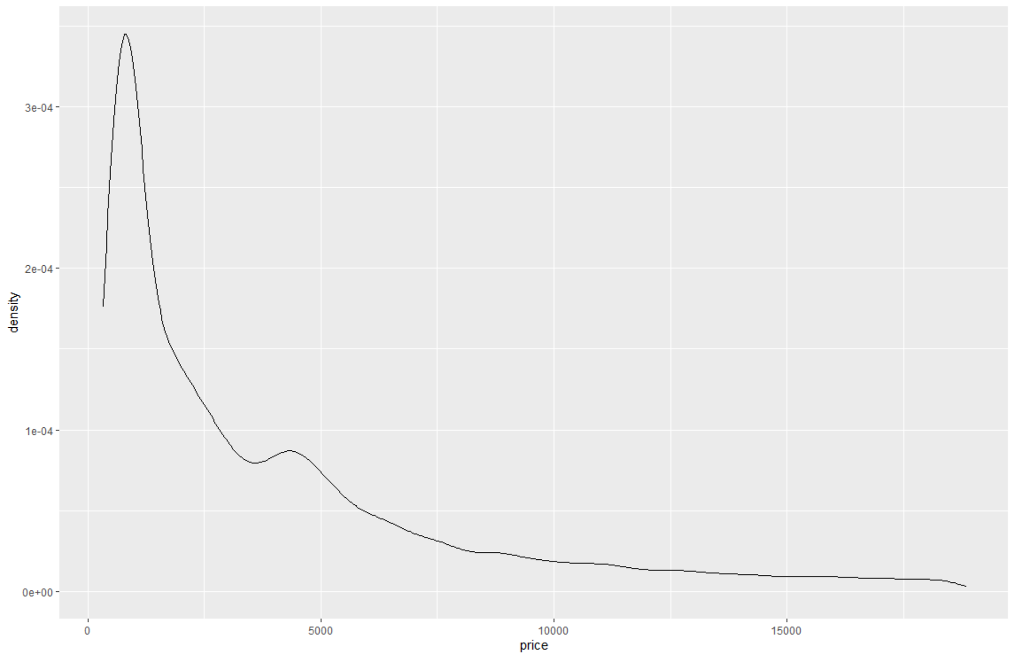
This is what it should look like:



4. ...and another one...

```
ggplot(data = diamonds) +
  aes(price) +
  geom_density()
```

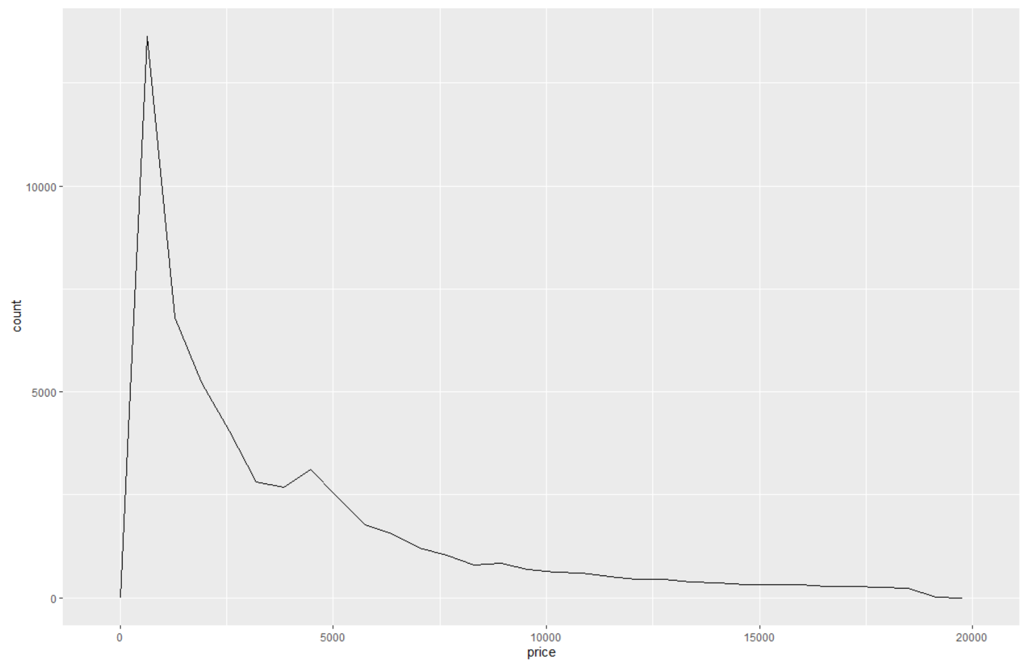
This is what it should look like:



5. ...and another one.

```
ggplot(data = diamonds)+
  aes(x = price) +
  geom_freqpoly()
```

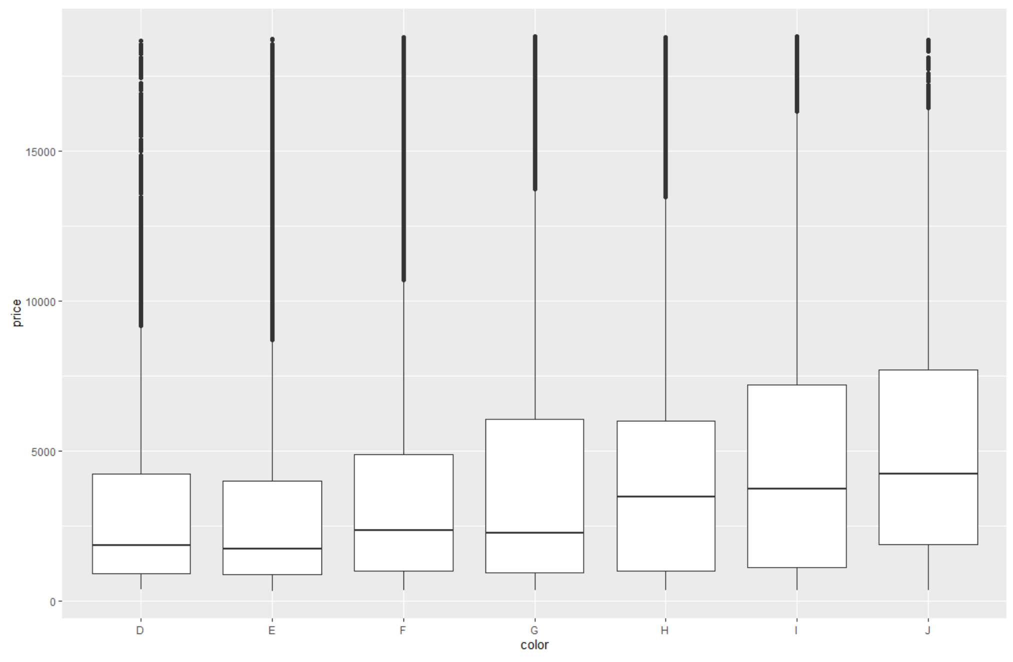
This is what it should look like:



6. Let's also use the diamonds data to make a graph of how diamond prices vary by what colour they are...

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

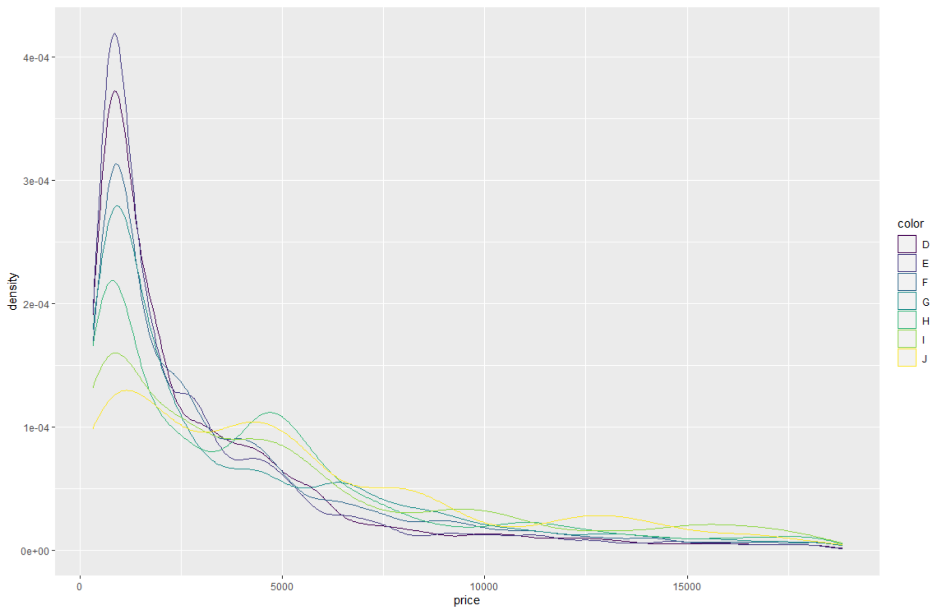
This is what it should look like:



7. …and another one.

```
ggplot(data = diamonds) +
  aes(x = price, color = color) +
  geom_density()
```

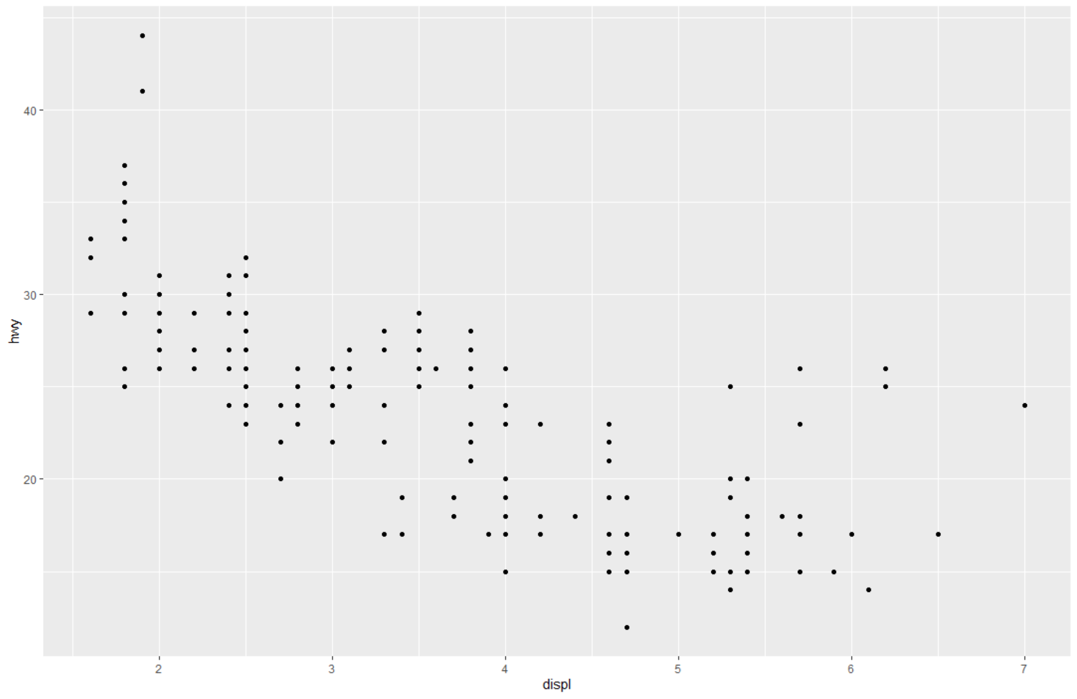
This is what it should look like:



8. Finally, the mpg data contains information on cars, including the miles per gallon they get on highways, and their engine displacement. (I don't know anything about cars). Let's look at the relationship between those two variables.

```
ggplot(data = mpg) +
  aes(x = displ, y = hwy) +
  geom_point()
```

This is what it should look like:



What was that?

In each of the above cases, we wrote **three* lines of code, stitched them together with the + sign, and ran the lines of code we ran.

The lines began as follows:

- ggplot
- aes
- geom

What does that mean?

Starting with **ggplot** means we're using the ggplot command: we're telling R that we're using the ggplot2 package, and - in practice - that we're drawing a graph. We then open a bracket where we specify what data we're using. We used a few different things: diamonds, mpg, economics, and so on. Finally on this line we closed the bracket and added a +, to indicate that something more was coming.

We then use an **aes** parenthesis, to indicate that we're adding aesthetics. This doesn't mean aesthetics in the sense of "making things look nice", but specifying what variables are going where. (We'll go through this in more detail in the coming worksheets.) Within this bracket, we indicate exactly what's going where, using the equals sign =. Before the equals sign we indicate the location of the variable we're interested in – x, y, colour, there's more to come – and after the equals sign we indicate the variable we want to use – price, state, date, and so on.

Finally, we add a geometric object, starting with **geom_**. These geometric objects are the specific objects that our graphs are made up of. Bar charts are made up of bars, hence geom_bar(), but scatterplots are made up of points, hence geom_point(). (For the moment these brackets are empty; that won't be the case all the way through the semester).

So, while we're going to be extending this a lot through the course of the worksheets fundamentally what we're doing is:

- declaring a graph using ggplot(), and declaring the data;

- adding aesthetic mappings with aes(), pairing aesthetics with variables;

- adding geometric objects.

In the next worksheet we will learn more about what these functions are – Layers in the Grammar of Graphics!

DATA VISUALISATION

LAB WORKSHEET 2

Creator: Dr Mark Taylor

What's happening in this Document?

In our last worksheet, we had a very gentle introduction into ggplot2, involving drawing basic versions of several different types of graphs. What we're doing today is extending bar charts: we're going beyond drawing a graph of how many counties there are in each of the different states of the Midwest in the United States, by looking at some of the different things that can be communicated with different kinds of bar charts.

The following data is entirely pulled from Spotify, using the Spotifyr R package. (You're not obliged to look into how I did this, but if you're interested in doing other work with music, I'd encourage you to do so.)

Some quick housekeeping:

You'll remember that we started last week by getting the tidyverse installed, from which we used ggplot2; this week, we'll be using another package from tidyverse: dplyr

Now, let's get prepared for the workshop by loading the tidyverse again.

```
library(tidyverse)
```

A step-change in quantitative social science skills
Funded by the Nuffield Foundation, ESRC and HEFCE
Q-Step
Nuffield Foundation
Sheffield Methods Institute.
The University Of Sheffield.

Loading data

In our last worksheet, we talked about graphs in a ggplot2 context having a minimum of three explicit elements: data, aesthetic mappings, and geometric objects. The aesthetic mappings and geometric objects we explicitly specified from the data that were available, but where did the data come from? The datasets we used – economics, midwest, diamonds, etc – all come preloaded with ggplot2. But most of the time, we want to use our own data to find out interesting things about the world. Let's load some data on Kendrick Lamar, Kanye West, and Rihanna.

```
kendrick <-read_csv("https://bit.ly/kendrickdata")
kanye <- read_csv("https://bit.ly/kanyedata")
rihanna <- read_csv("https://bit.ly/rihannadata")
```

What's happening here?

We have a bunch of URLs. These URLs consist of .csv files with information from Spotify about each of the relevant artists. (If you're interested, you can copy the URLs and open them in a web browser to see what the files look like in Excel.) These URLs are nested in (brackets) and "quotes", to indicate that R's dealing with strings rather than objects (you'll pick this distinction up, don't worry).

Preceding the URLs is the read_csv() command. read.csv means that we're telling R that the thing inside the bracket is a .csv file and not any other kind of file – .xlsx, .txt, whatever else you like.

Preceding that is an arrow: <-. The arrow is used to tell R that we want to turn the thing on the right-hand side into the thing on the left hand side, which is what we're calling our new object.

Finally, on the left-hand side we've got the objects we want to create. I've called them kendrick, kanye and rihanna, largely for simplicity, but I could have called them more-or-less whatever I wanted.

Having run the commands, we've got a bunch of red text. If your red text starts with something like "Parsed with column specification", you're fine; the important differences between character, double, and numeric, etc, will become clear, and most of the differences aren't important for the purposes of this module.

(One quick note: I've stripped out live albums, collaborations, mixtapes, EPs, best ofs, different releases in different countries, etc; we're just dealing with the main albums for each artist.)

Let’s look at the data

We’ve loaded some data. What is it? There’s a few different commands we can try to get a feel for what we’re dealing with.

```
names(kendrick)
glimpse(kendrick)
head(kendrick)
summary(kendrick)
```

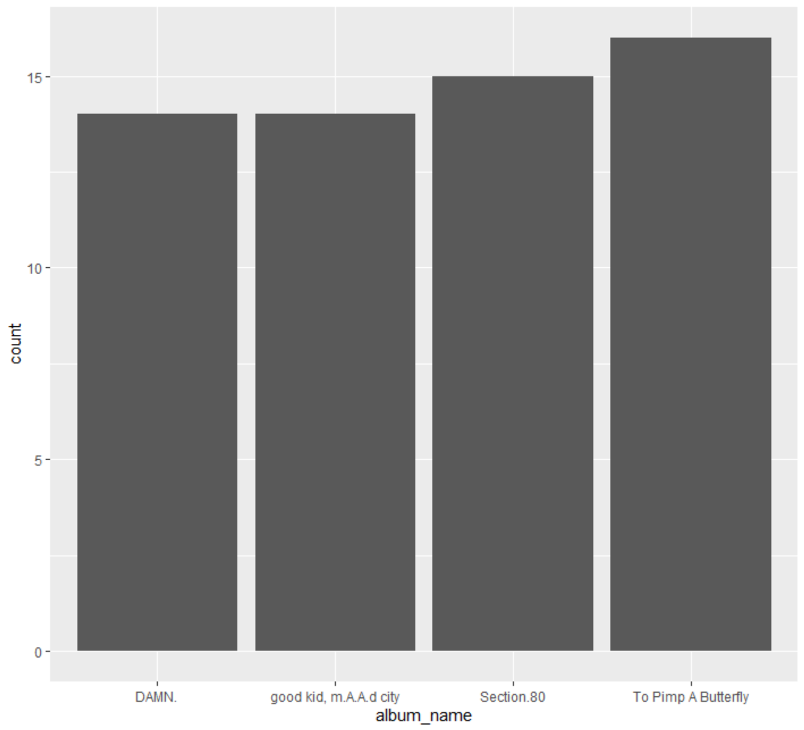
OK, we’ve got a bunch of different variables: some categorical, some continuous, some to do with the music itself, some to do with popularity, some you can ignore for the moment.

Let’s draw some graphs

1. Let’s draw a graph of how many songs there are on each Kendrick Lamar’s album.

```
ggplot(data = kendrick) +
  aes(x = album_name) +
  geom_bar()
```

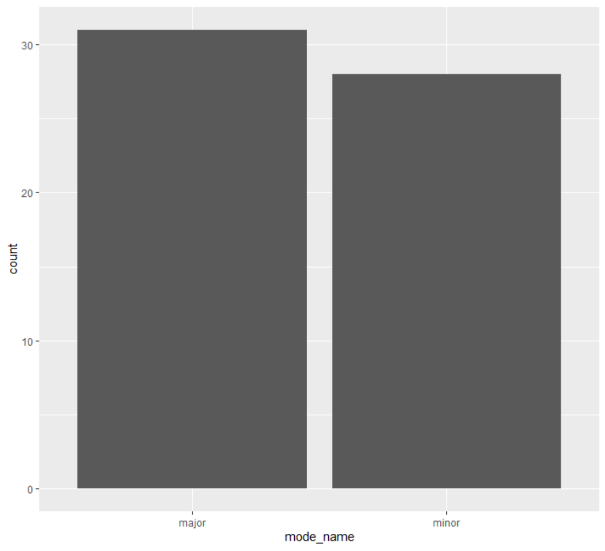
This is what it should look like:



2. Now, let's draw a graph of how many of Kendrick's songs are in major keys, and how many are in minor keys.

```
ggplot(data = kendrick) +
  aes(x = mode_name) +
  geom_bar()
```

This is what it should look like:

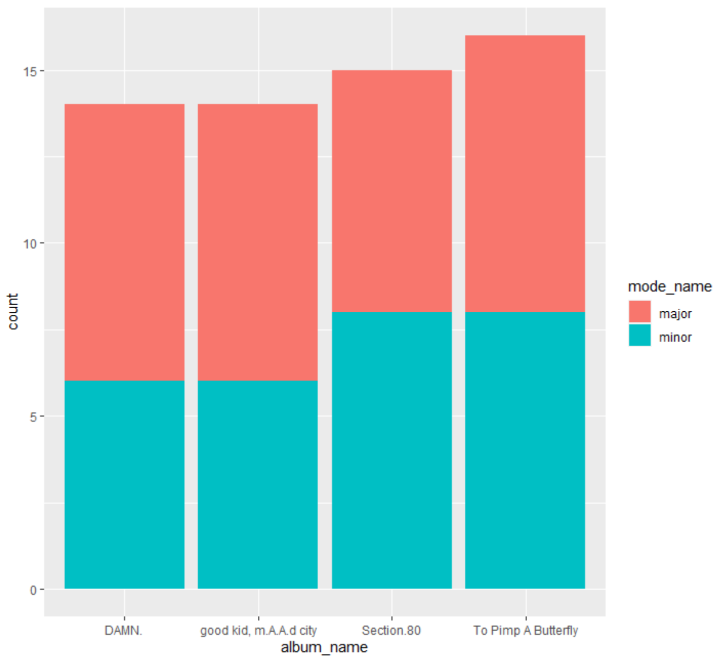


What's the relationship like between these two variables? Do some albums skew more towards major keys, and others more towards minor keys?

3. Let's find out by colouring in the first graph according to how many songs on each album are major, and how many are minor.

```
ggplot(data = kendrick) +
  aes(x = album_name, fill = mode_name) +
  geom_bar()
```

This is what it should look like:

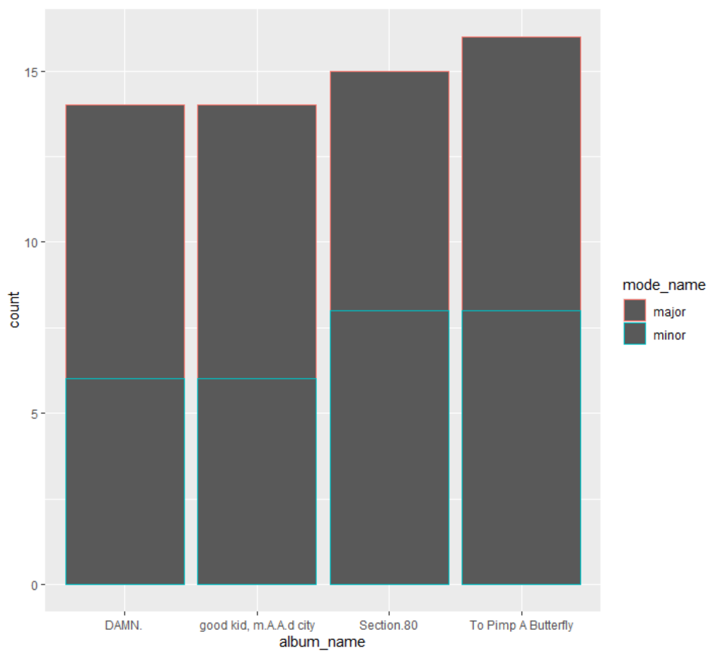


4. So, we've changed what we did before by adding a second aesthetic mapping, in the shape of fill. You might ask why it's called fill rather than colour: see what happens when you use colour

```
ggplot(data = kendrick) +
  aes(x = album_name, colour = mode_name) +
  geom_bar()
```

instead and you'll figure out why.

This is what happens:

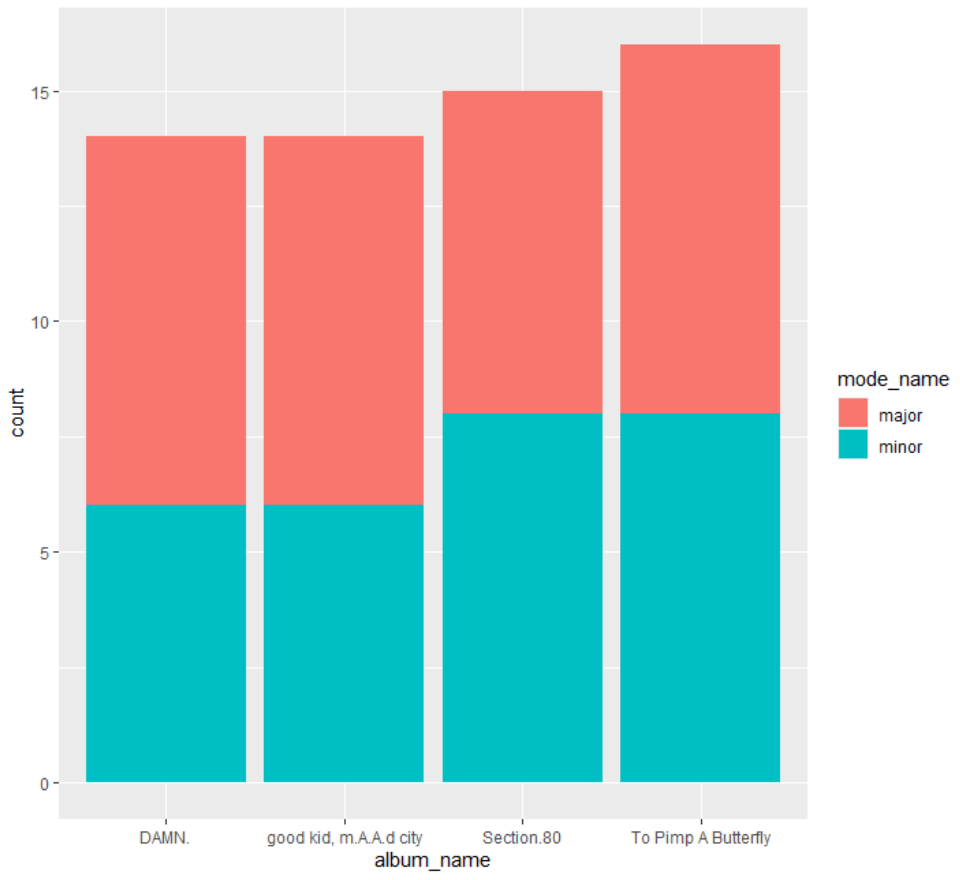


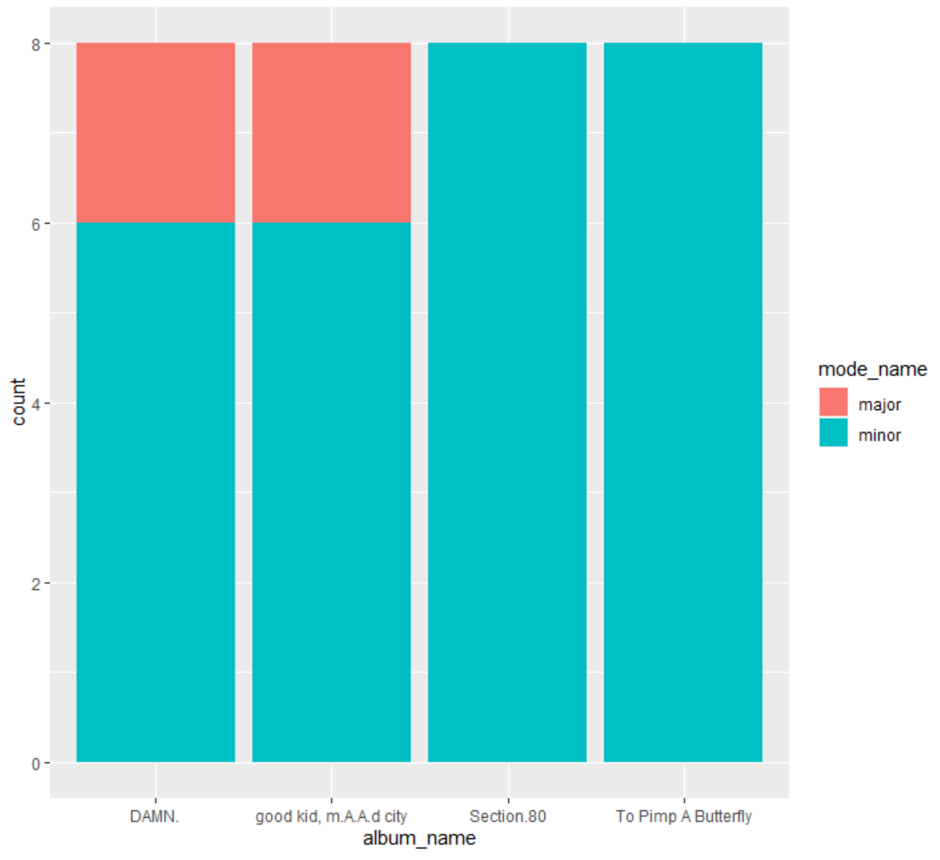
5. However, this isn't the only way to communicate this information. We can use position adjustments! Let's see how they work in practice. We're going to look at four position adjustments: stack, identity, dodge, and jitter. You'll note that the first one looks exactly the same as the earlier graph – that's because it's the default, and doesn't need to be manually specified. The others aren't the defaults, which is why they need to be specified.

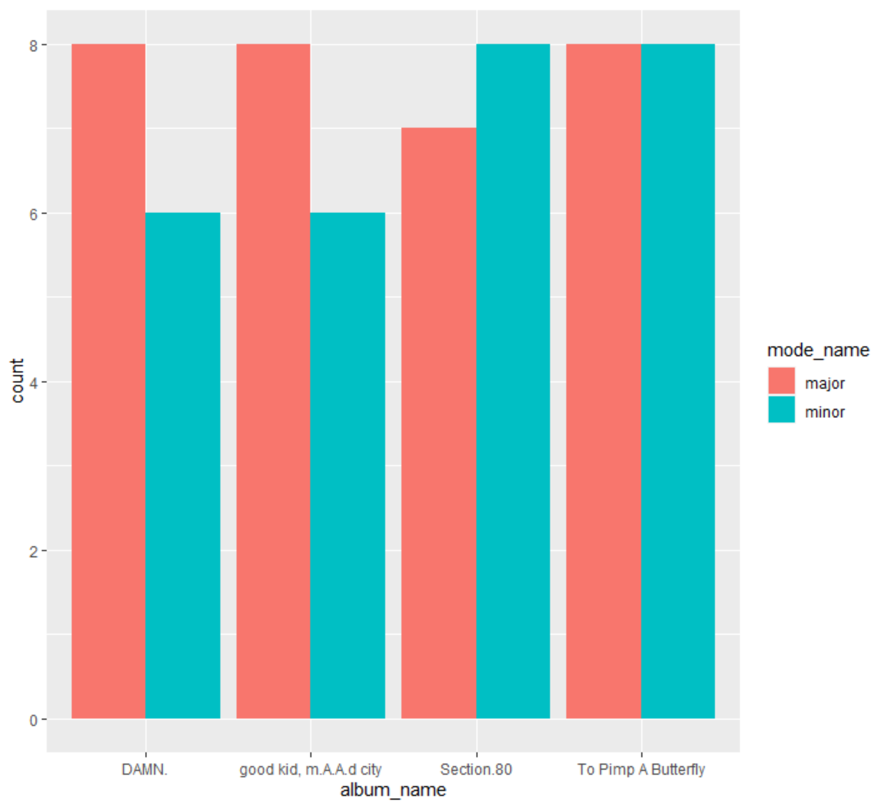
```
ggplot(data = kendrick) +
  aes(x = album_name, fill = mode_name) +
  geom_bar(position = "stack")

ggplot(data = kendrick) +
  aes(x = album_name, fill = mode_name) +
  geom_bar(position = "identity")

ggplot(data = kendrick) +
  aes(x = album_name, fill = mode_name) +
  geom_bar(position = "dodge")

ggplot(data = kendrick) +
  aes(x = album_name, fill = mode_name) +
  geom_bar(position = "fill")

ggplot(data = kendrick) +
  aes(x = album_name, fill = mode_name) +
  geom_bar(position = "jitter")
```

What are the pros and cons of each? (In what circumstances can you imagine jitter being useful?)

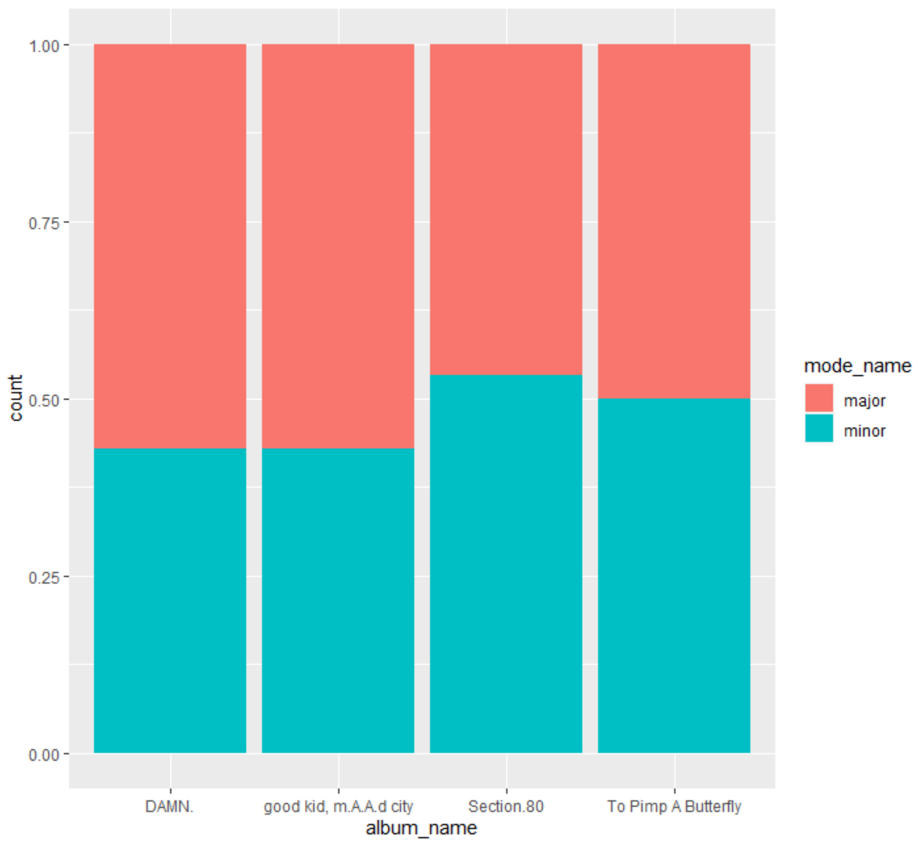
This is what these graphs look like:



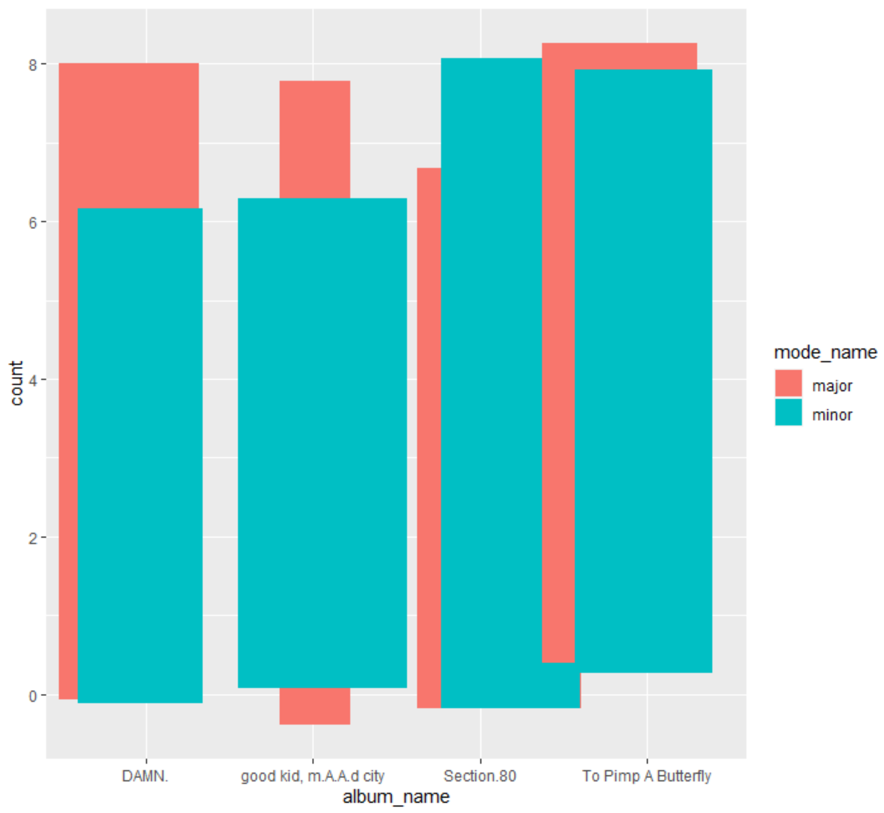




8
6
4
2
0
count
DAMN.
good kid, m.A.A.d city
Section.80
To Pimp A Butterfly
album_name
mode_name
major
minor



1.00
0.75
0.50
0.25
0.00
count
DAMN.
good kid, m.A.A.d city
Section.80
To Pimp A Butterfly
album_name
mode_name
major
minor

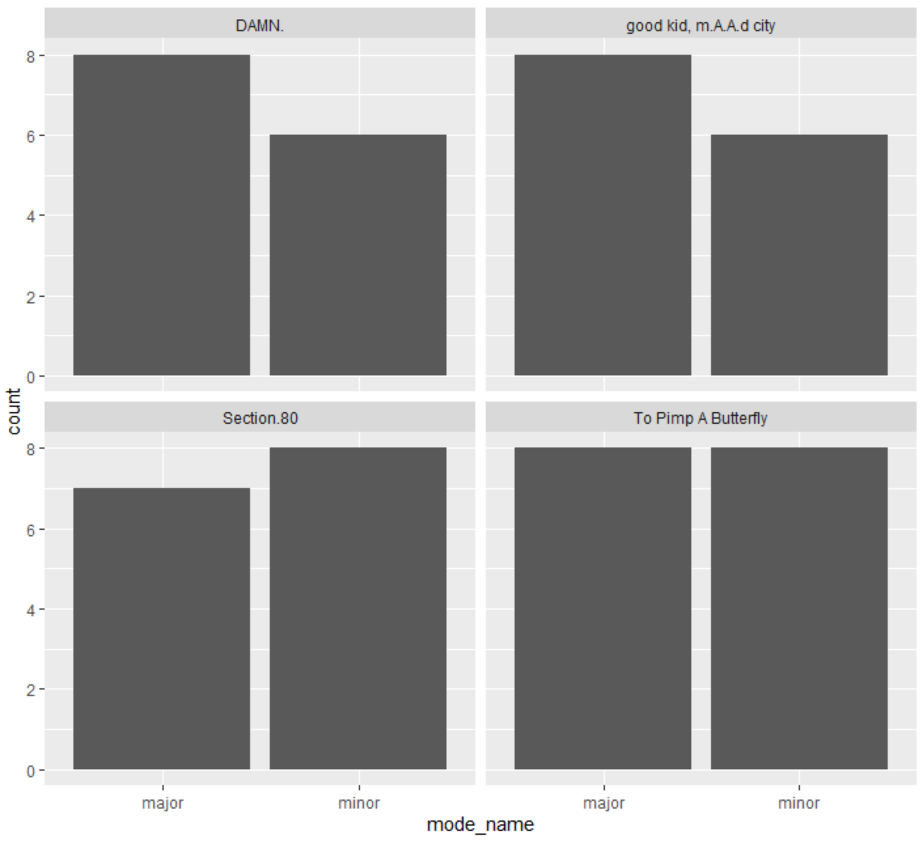

8
6
4
2
0
count
DAMN.
good kid, m.A.A.d city
Section.80
To Pimp A Butterfly
album_name
mode_name
major
minor

6. One alternative way we can communicate the balance of major and minor keys on each album is through the use of faceting. We can do this like so.

```
ggplot(data = kendrick) +
  aes(x = mode_name) +
  geom_bar() +
  facet_wrap(~ album_name)
```

We've now hit a fourth line in a ggplot command for the first time. This facet_wrap() command indicates we want to facet based on album_name. (The tilde - ~ - is for reasons we'll come back to.)

This is what it looks like:



Trying something harder

OK, so everything we've seen so far consists of counting things up: how many songs are there on each album, how many songs are in major keys. But we might also want to use bar charts to denote some kind of transformation: on average, which album has tracks that are the fastest? This is a bit harder to do, but it's not beyond us.

The main thing we have to do is to spend a bit longer on preparing the data in the first place. The data as we have it doesn't include information on average tempo by album, so we'll have to construct it ourselves.

7. We can do this using the dplyr package (it comes with the Tidyverse package, like ggplot). It's not the only way to do so, but it's a way that can be useful, and it's a way that's included in the Healy book. We can start by getting the means like so:

```
kendrick %>%
  group_by(album_name) %>%
  summarise(mean_tempo = mean(tempo))
```

You can see this has given us a small table in the Console (the box below where you have been typing in your code) with the information we were hoping for. What's going on here?

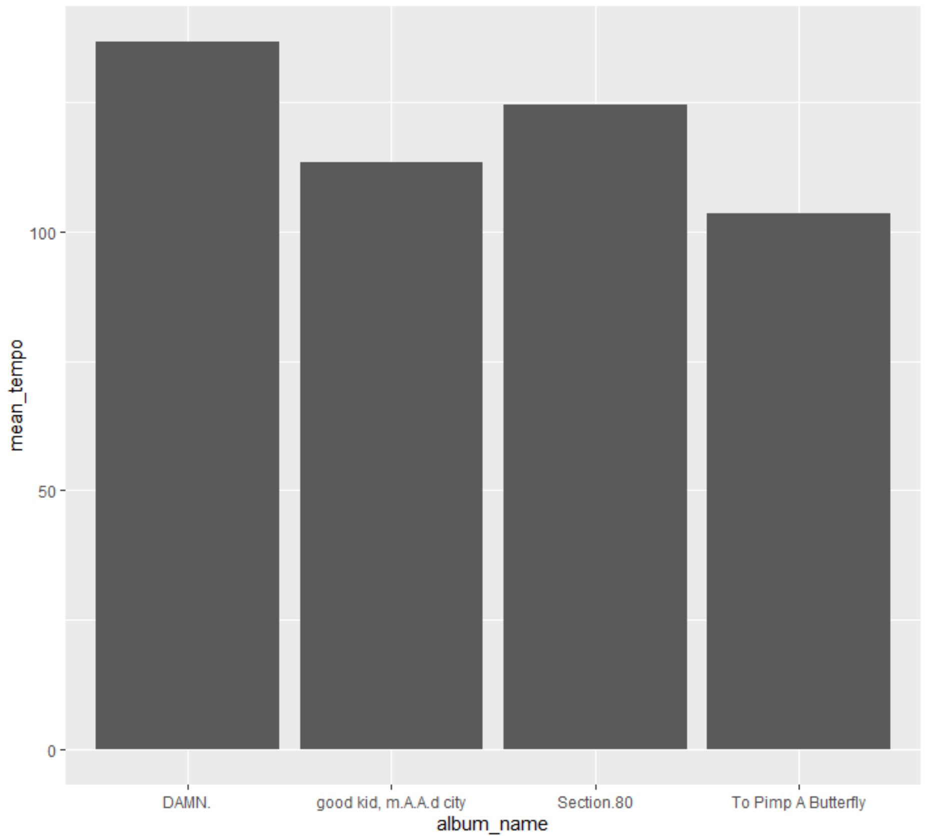
- we've started with the object kendrick. We've then added what's referred to as a pipe, in the shape of
  %>%, which basically means "and then" – it's an alternative to nesting things in loads of brackets, which is what you normally get in programming. (If you type ctrl-shift-m on a Windows machine, or cmd-shift-m on a Mac, it'll appear)
- we've then typed group_by(album_name), which means we're grouping all the observations, or tracks, according to the variable album_name. The relevance of which is...
- we've used the summarise() command to generate a new variable, mean track popularity. We've done that by specifying our name of our new variable – mean_tempo – on the left-hand side of the equals sign, and the way we want to generate it – mean(tempo) on the right-hand side. The relevance of group_by was to identify mean popularity by what - we also could have looked at whether songs in major or minor keys were more popular, for example.

Now what?

8. Now let's draw a graph.

```
kendrick %>%
  group_by(album_name) %>%
  summarise(mean_tempo = mean(tempo)) %>%
  ggplot() +
  aes(x = album_name, y = mean_tempo) +
  geom_col()
```

This is what it looks like:



So, a couple of things to note here.

- the first three lines are exactly the same as what we were looking at before, save for the final pipe at the end leading into the following line.
- you'll note we haven't specified data in the ggplot() parenthesis. That's because we've already set up the data in the previous few lines.
- hopefully the aes() bracket looks familiar: specifying two variables with aesthetic mappings.
- we've used geom_col() instead of geom_bar(). We use geom_col when we want to map a variable to the y-axis, which in this case is the mean popularity of each track on each album; we use geom_bar when we want the y-axis to denote the number of observations for each category

Bonus Exercises

Here are some independent exercises for you to employ the skills you've just learned:

1) Please answer the following questions by drawing a graph that reveals the answer.

   A. How many of each of Kendrick Lamar's songs are in each key? (C sharp, E, etc)

   B. On which Kendrick Lamar albums are there tracks with time signatures other than 4/4?

   C. Which Kendrick Lamar album has the highest average danceability?

2) Following this, please answer those same questions for either Rihanna or Kanye West.

3) Finally, please come up with another question you're interested in answering using the available data; having answered it, please create a graph.

DATA VISUALISATION

LAB WORKSHEET 3

Creator: Dr Mark Taylor

What's happening in this Document?

In our last lab worksheet, we looked at bar charts in a lot of detail. We drew simple bar charts showing the frequencies of different categories within variables, we drew bar charts that included information about two different variables in various different ways, and we drew bar charts that included information about continuous variables, in the shape of showing the mean values of different categories.

In this worksheet, we're focusing on scatterplots. Instead of looking primarily at categorical variables as we did last week, we're now looking primarily at relationships between two continuous variables. This involves a few different things. Once again, we're going to look at a way to add more variables than just the ones that are necessary for the basic plot; we're also going to introduce another way of tidying up our data to make it comprehensible.

Getting set up

Let's get set up. Let's first load some packages, and then load some data.

```
library(tidyverse)
library(lubridate)
```

In the last few weeks, we've started by loading the tidyverse. This time round, I've also loaded lubridate, which is a helpful package for working with dates (this will become clear in due course).

A step-change in quantitative social science skills
Funded by the Nuffield Foundation, ESRC and HEFCE
Nuffield Foundation
Sheffield Methods Institute.
The University Of Sheffield.

Loading Data

Let's also load some Spotify data, as we did in our last worksheet.

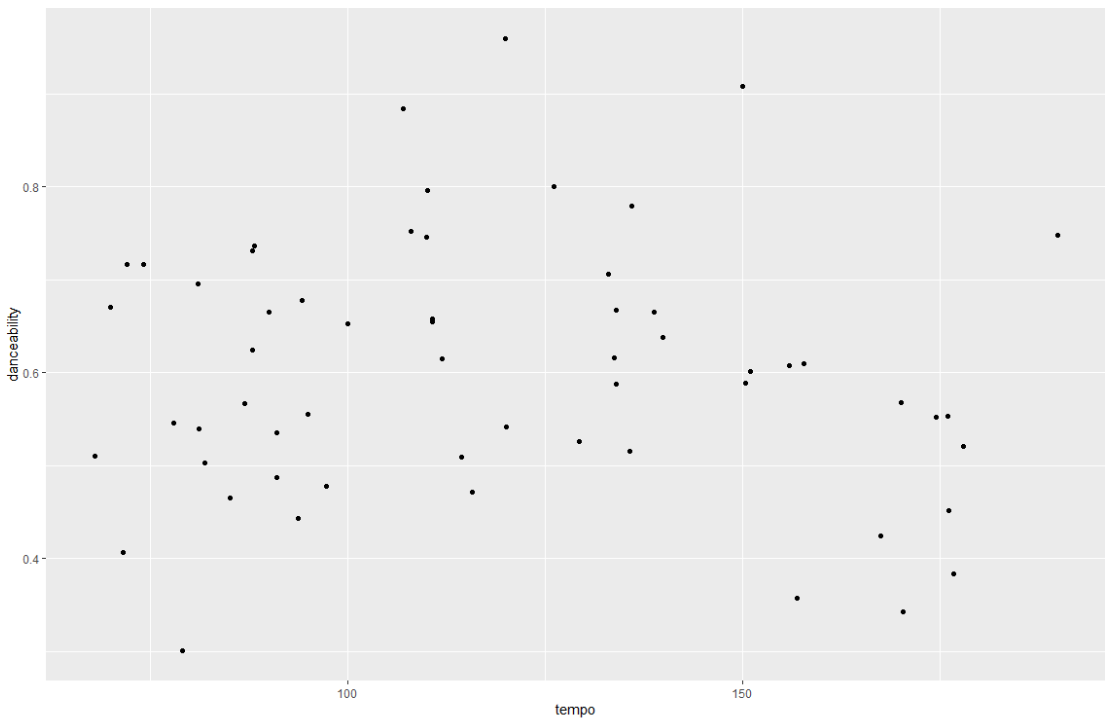
```
kendrick <-read_csv("https://bit.ly/kendrickdata")
kanye <- read_csv("https://bit.ly/kanyedata")
rihanna <- read_csv("https://bit.ly/rihannadata")
beyonce <- read_csv("https://bit.ly/beyonce_data")
```

Drawing some scatterplots:

1. Are Kendrick Lamar's quicker tracks more danceable? Let's find out.

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability) +
  geom_point()
```

This is what it should look like:



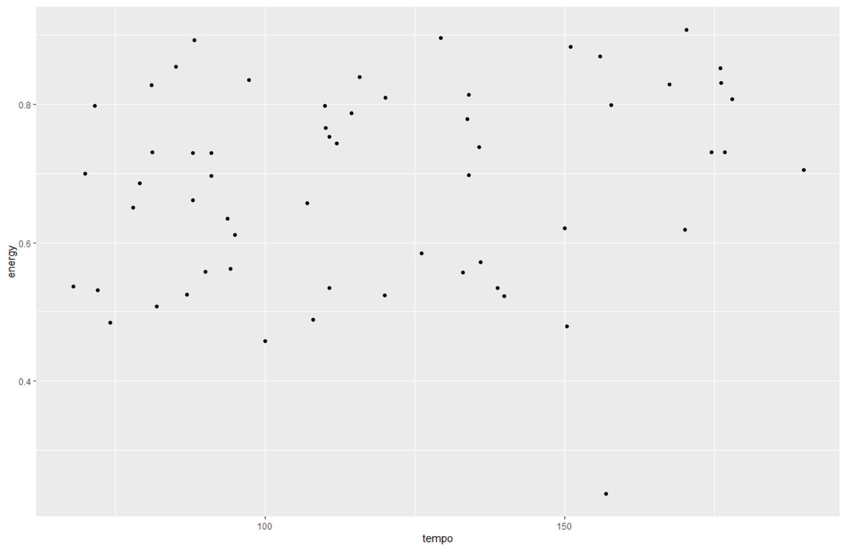
So, this is similar to the kinds of things that we've seen before. Three elements to the command:

- ggplot(), where we've specified the data, followed by:
- aes(), specifiying which variables are going on each axis, followed by:
- a geometric object: this time, a point (in fact, several of them) What does this show?

2. Let's draw another scatterplot. Are Kendrick's quicker tracks more energetic? Let's find out.

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = energy) +
  geom_point()
```

This is what it should look like:

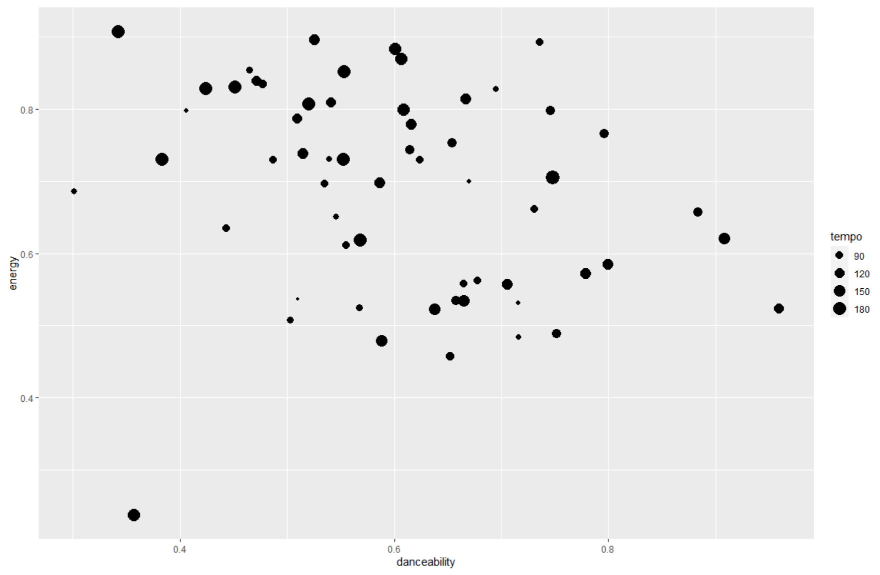


OK, so drawing a scatterplot is a fairly straightforward and manageable exercise. We can also add encode more information in scatterplots, through more aesthetic mappings.

3. Let's see the relationship between danceability, energy, and tempo, by making quicker tracks bigger.

```
ggplot(data = kendrick) +
  aes(x = danceability,
      y = energy,
      size = tempo) +
  geom_point()
```

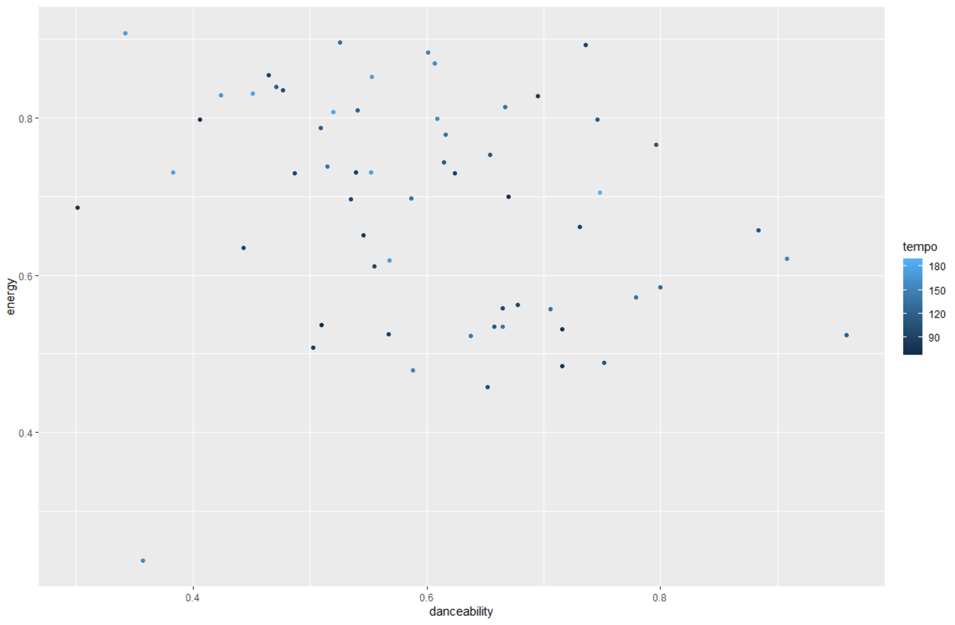
This is what it should look like:



4. As an alternative, we can vary points' colours by how quick they are:

```
ggplot(data = kendrick) +
  aes(x = danceability,
      y = energy,
      colour = tempo) +
  geom_point()
```

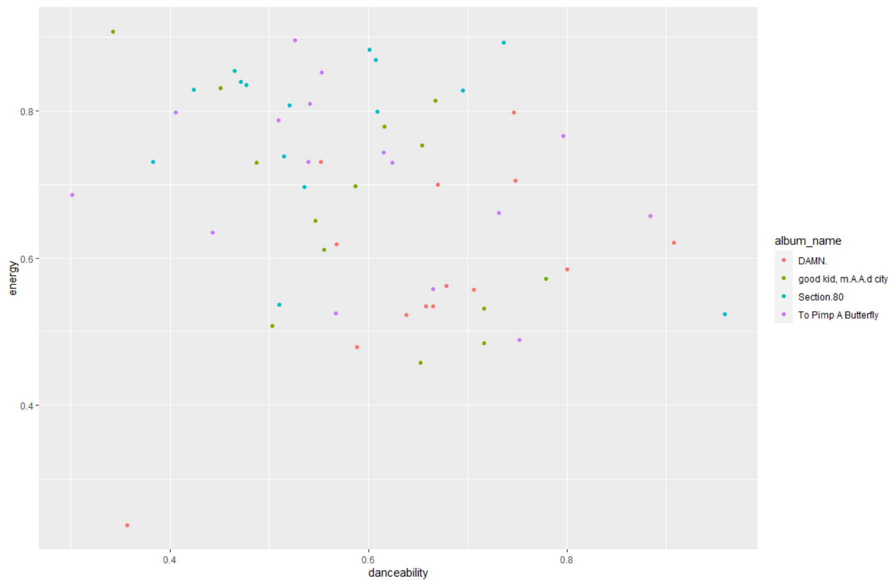
This is what it should look like:



5. We can also colour points using categorical variables rather than continuous variables, like so:

```
ggplot(data = kendrick) +
  aes(x = danceability,
      y = energy,
      colour = album_name) +
  geom_point()
```

This is what it should look like:



You'll note that the earlier colour ramp went obviously from low to high, while this one uses a categorical set of colours. This is done automatically in ggplot2: it detects whether the variable ascribed to colour is continuous or categorical, and colours accordingly. (Sometimes it gets it wrong, like when you've got a variable you're treating as categorical but actually takes numeric values. In those instances, you'll want to look at the forcats package.)

Dates

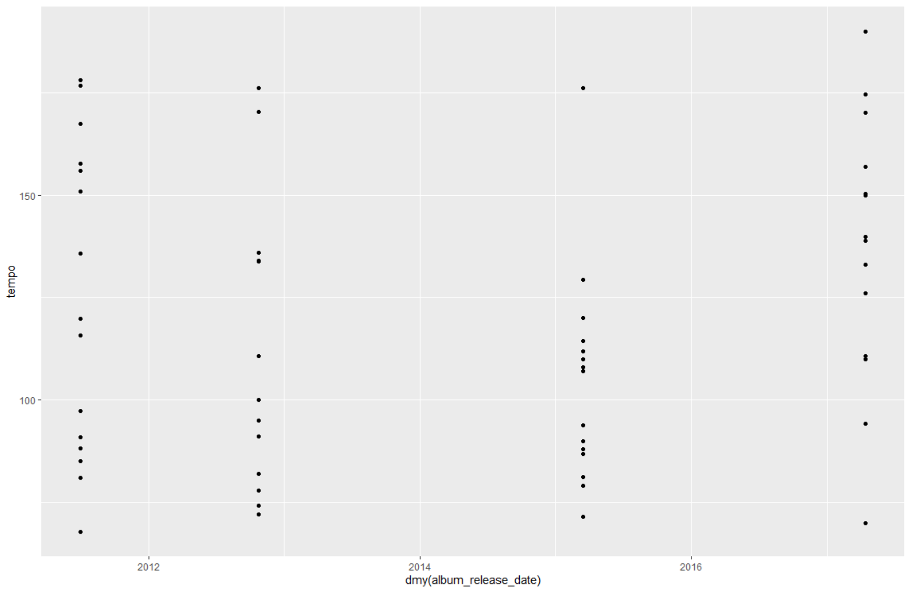
(We are not going to push this point in general, but in case you want to work with dates in future...)

In the bonus exercise from our last worksheet, you may have decided to make a graph showing when different albums had come out. This can be a bit fiddly, as R can read this data in alphabetical order, while we really want it in date order.

6. Let's see if Kendrick Lamar's more recent tracks are quicker.

```
ggplot(data = kendrick) +
  aes(x =dmy(album_release_date),
      y = tempo) +
  geom_point()
```

This is what it should look like:



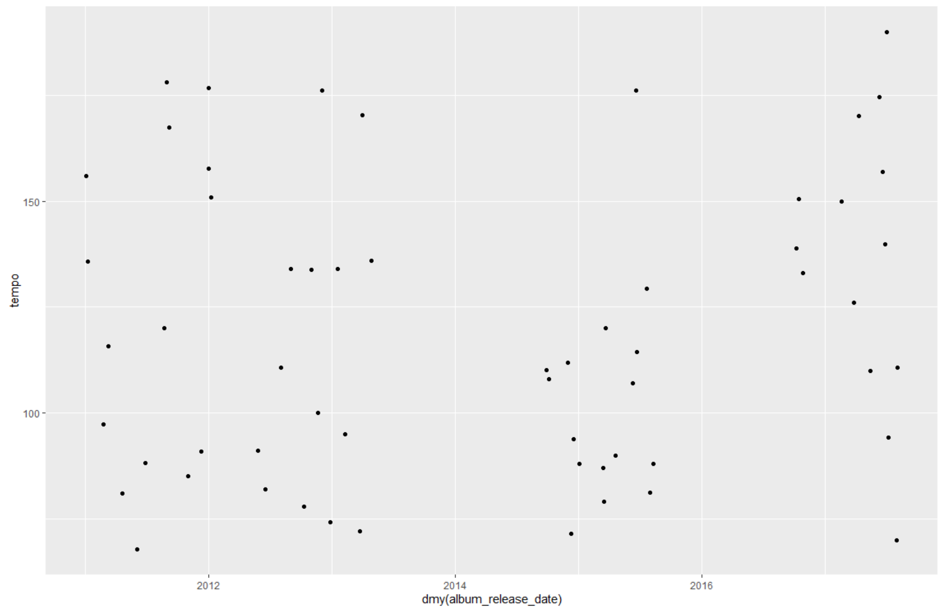
OK, a couple of things to note here, first.

- you'll note that the variable album_release_date is wrapped in dmy(). This is so we're declaring to R that that values of that variable are of the format day-month-year. Try it without the dmy() wrapper and see what happens. (If this doesn't work, you probably didn't load the lubridate package.)
- scatterplots don't really work when some of the continuous variables only hold particular values: it should come as no surprise that the album release date is the same for each of the tracks on DAMN.

7. So, let's add some jitter.

```
ggplot(data = kendrick) +
  aes(x =dmy(album_release_date),
      y = tempo) +
  geom_point(position = "jitter")
```

This is what it should look like:

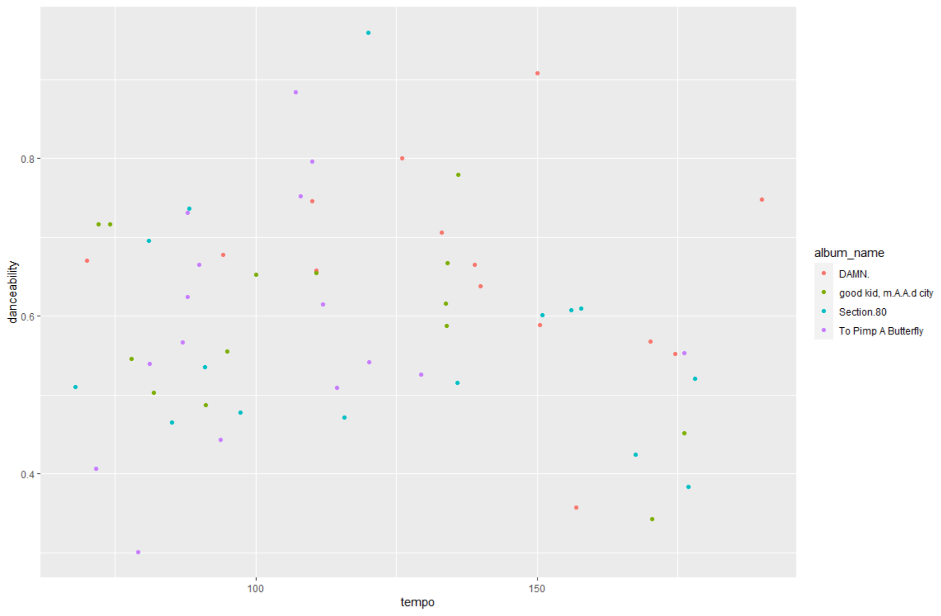


What's the value in having done that?

8. Let's now combine that information with the earlier graph. Maybe the relationship between tempo and danceability is actually driven by which albums the different tracks are on.

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = album_name) +
  geom_point()
```

This is what it should look like:



Playing around with factors

So. One issue with this data so far is that, while I know the order that Kendrick Lamar's albums came out in, not everyone does. If I looked at that previous graph without that knowledge, I wouldn't know whether his more recent albums were more popular or not. At the moment, the albums are just in alphabetical order. Let's change that.

9. First, let's create a vector of albums. This just involves specifying a list of albums in the order we want them.

```
kendrick_levels <- c("Section.80",
                     "good kid, m.A.A.d city",
                     "To Pimp A Butterfly",
                     "DAMN.")
```

You will see that we created a new vector as it creates a "Values" section in the Environment Pane (the top right pane in RStudio)

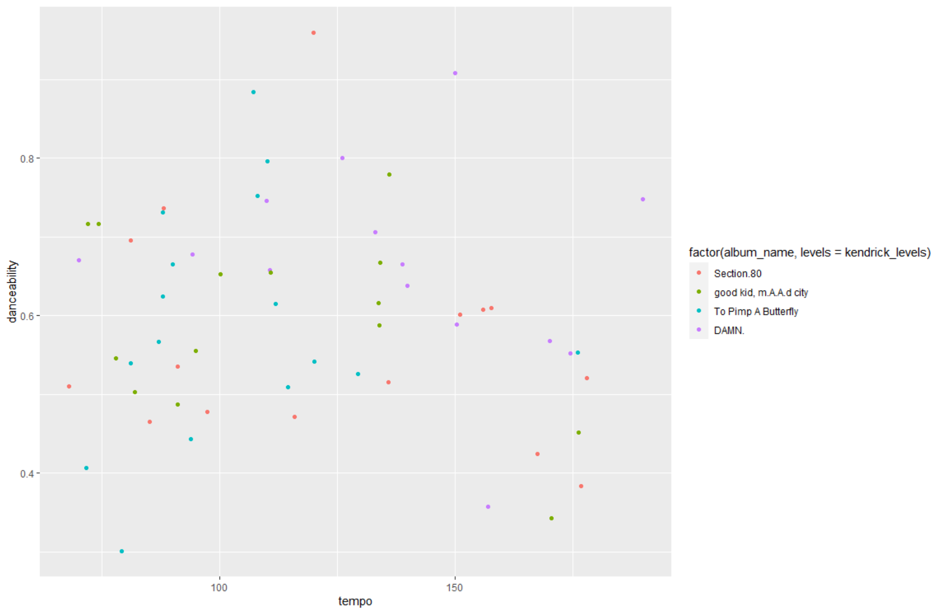
What's this?

- I want a new object called kendrick_levels. So far, when we've created objects, they've been matrices, but here we just want a vector (or list).
- we've used the arrow, to convey that the new object is to be made up of something on the right hand side of the arrow.
- we've used the c() command to indicate a list of several elements. c() here stands for
- concatenate. finally, we've created a list of albums in chronological order. Each album's in quotes, and they're separated by commas.

10. Now what?

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = factor(album_name,
                      levels = kendrick_levels)) +
  geom_point()
```

This is what it should look like:

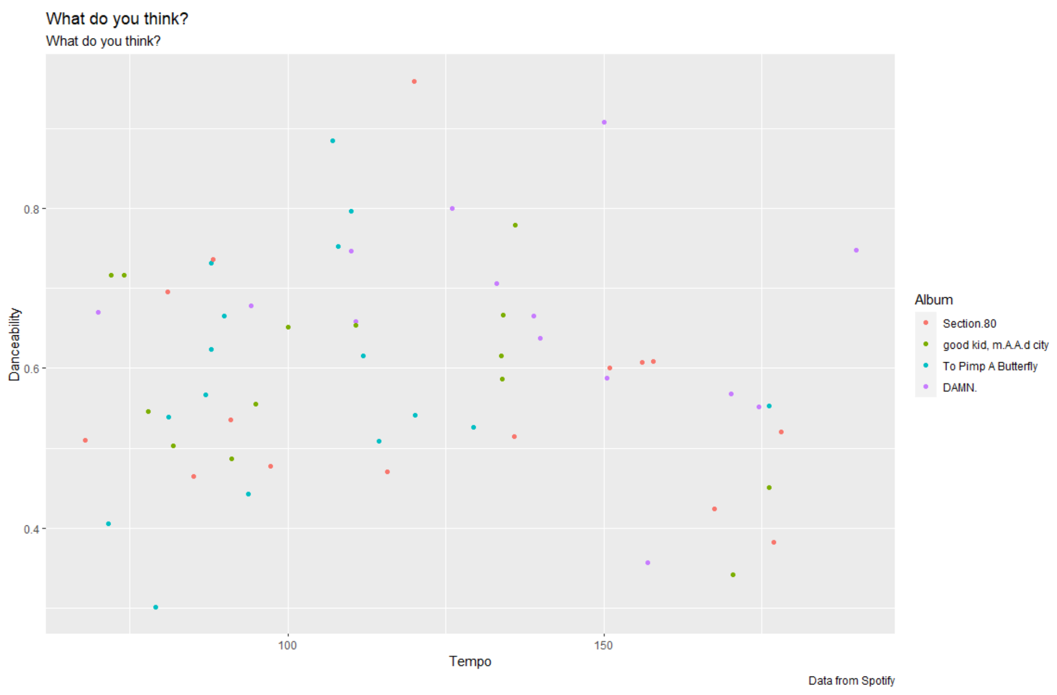


The difference between the code for this and for the earlier graph is in the line beginning colour =. Instead of just specifying album_name, we've written factor(album_name, levels = kendrick_levels), which means that we've manually specified the order of levels of the factor to be that found in our new object, kendrick_levels.

11. However, it doesn't look great. The legend is now taking up loads of space, and the title of the legend doesn't make sense. In general, though, our graphs could look better with better labelling. Let's sort that out now.

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = factor(album_name,
                      levels = kendrick_levels)) +
  geom_point() +
  labs(x = "Tempo",
       y ="Danceability",
       colour = "Album",
       title = "What do you think?",
       subtitle = "What do you think?",
       caption = "Data from Spotify")
```

This is what it should look like:



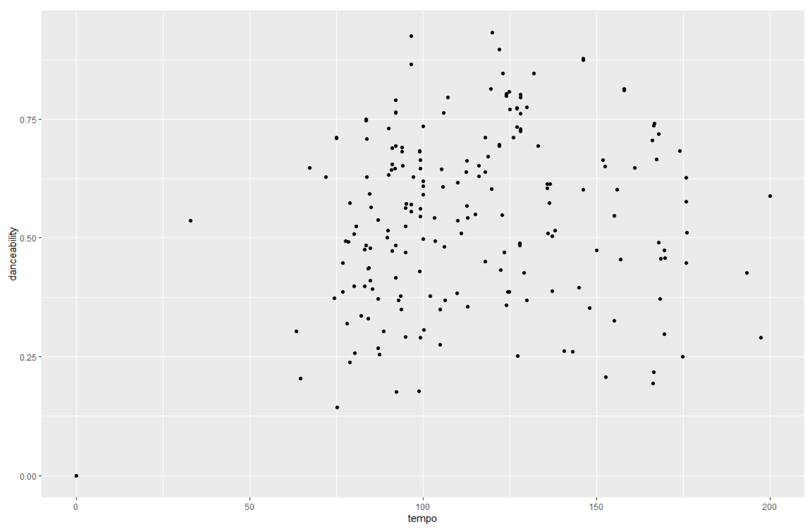
You'll note what I've added here is a labs() command. Here, labs() stands for labels(), and in this parenthesis, I've specified how each element of the graph should be labelled: x, y, colour, a title, a subtitle, and a caption (you'll also note the title and subtitle I've provided invites you to think what they should be).

Moving to Beyonce

12. We're looking good on Kendrick Lamar. Let's look at Beyonce.

```
ggplot(data = beyonce) +
  aes(x = tempo,
      y = danceability) +
  geom_point()
```

This is what it should look like:

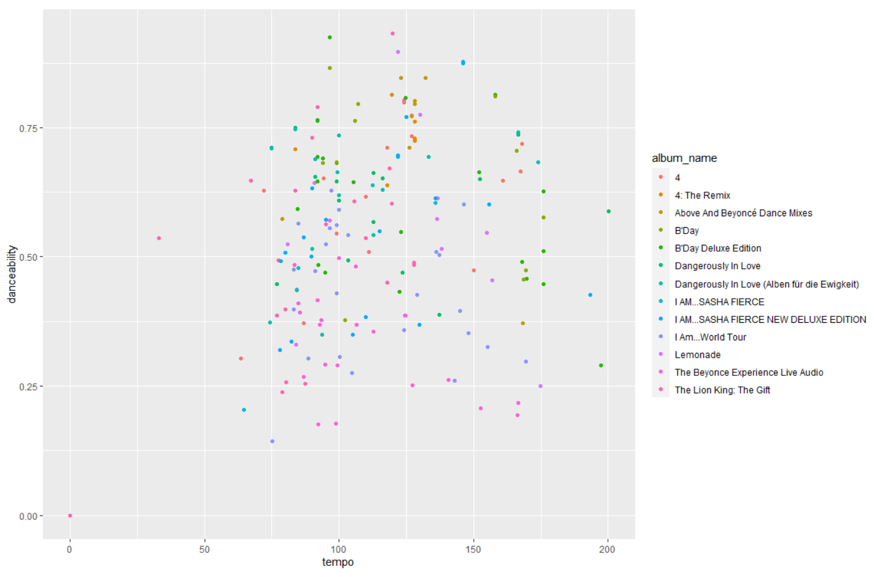


This looks weird. The range on tempo is quite a bit wider than we've seen before, and one song seems to have a zero for both danceability and tempo. What's going on?

13. Maybe it's something to do with the albums we're looking at.

```
ggplot(data = beyonce) +
  aes(x = tempo,
      y = danceability,
      colour = album_name) +
  geom_point()
```

This is what it should look like:



OK, this is providing a bit more context. As there's so many albums it can be hard to distinguish the colours, but this might be something to do with the Beyonce Experience Live Audio. There's also a bunch of remix albums and deluxe editions, meaning we've got some redundancy in tracks.

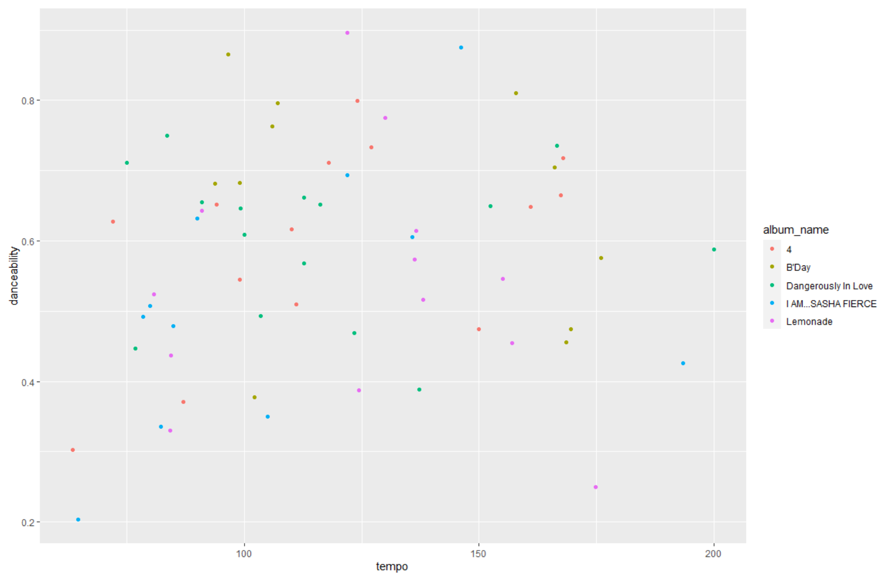
(We're also missing Beyonce – as in, the album with XO and Hunted on it. I'm not sure why this is. Sorry!)

14. We don't want all this information, so we need to discard some. How can we do that? Let's use the filter

command to subset our data, so we've just got the main albums.

```
beyonce                  %>%
  filter(album_name == "4" |
           album_name == "B'Day"  |
           album_name == "Dangerously In Love" |
           album_name == "I AM...SASHA FIERCE" |
           album_name == "Lemonade") %>%
  ggplot()     +
  aes(x = tempo,
      y    =    danceability,
      colour = album_name) +
  geom_point()
```

This is what it should look like:



What have we done here?

- as with our last worksheet, when we used the group_by() and summarise() commands, we've started with an object – beyonce – and followed it with a pipe – %>%.
- we've then used the filter() command and opened a bracket.
- inside the bracket, we've specified album_name == "", for one of the albums we want to keep;
- we've then used the | character, meaning or;
- we've then repeated this for every album we want to include;
- we've finally followed this up with a regular ggplot command as before; the only difference is that we've not specified the data, because we generated that in the previous lines.

15. Alternatively, we could have got the same result like so:

```
beyonce %>%
  filter(album_name != "4: The Remix" &
           album_name != "Above And Beyoncé Dance Mixes" &
           album_name != "B'Day Deluxe Edition" &
           album_name != "Dangerously In Love (Alben für die Ewigkeit)" &
           album_name != "I AM...SASHA FIERCE NEW DELUXE EDITION" &
           album_name != "I Am...World Tour" &
           album_name != "The Beyonce Experience Live Audio" &
           album_name != "The Lion King: The Gift") %>%
  ggplot() +
  aes(x = tempo,
      y = danceability,
      colour = album_name) +
  geom_point()
```

The graph should look exactly like the previous graph you just made in 14.

The difference here is that we've specified what we're *excluding*, rather than what we're including. So instead of the double equals – == – we've used not equals – !=. We've also used and – & – rather than or – | – because we don't want *any* of these albums.

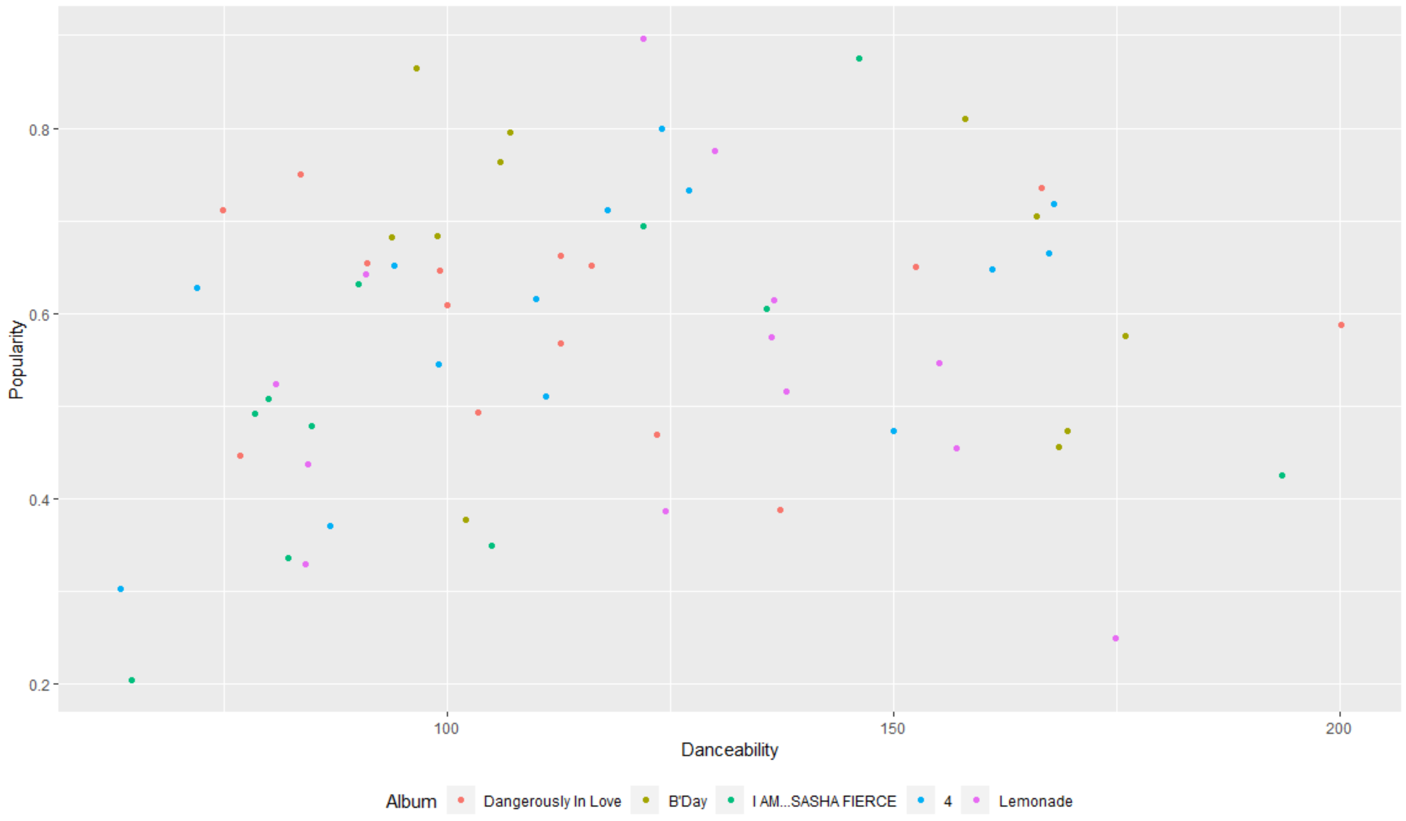
16. Finally, let's combine all today's elements. So let's draw a scatterplot of this information, with albums in chronological order in the legend, labelled up properly. (For something new, let's also move the legend to the bottom, to handle the fact that some of these album titles are pretty long.)

```
beyonce_levels <- c("Dangerously In Love",
                    "B'Day",
                    "I AM...SASHA FIERCE",
                    "4",
                    "Lemonade")

beyonce                  %>%
  filter(album_name == "4" |
           album_name == "B'Day" |
           album_name == "Dangerously In Love" |
           album_name == "I AM...SASHA FIERCE" |
           album_name == "Lemonade" ) %>%
  ggplot()     +
  aes(x = tempo,
      y = danceability,
      colour = factor(album_name, levels = beyonce_levels)) +
  geom_point() +
  labs(x ="Danceability",
       y = "Popularity",
       colour = "Album",
       title = "What do you think?",
       subtitle = "What do you think?",
       caption = "Data from Spotify")+
  theme(legend.position = "bottom")
```

So there are loads of lines of code, but each of them isn't that complicated; it's just a question of stitching everything together:

What do you think?

What do you think?



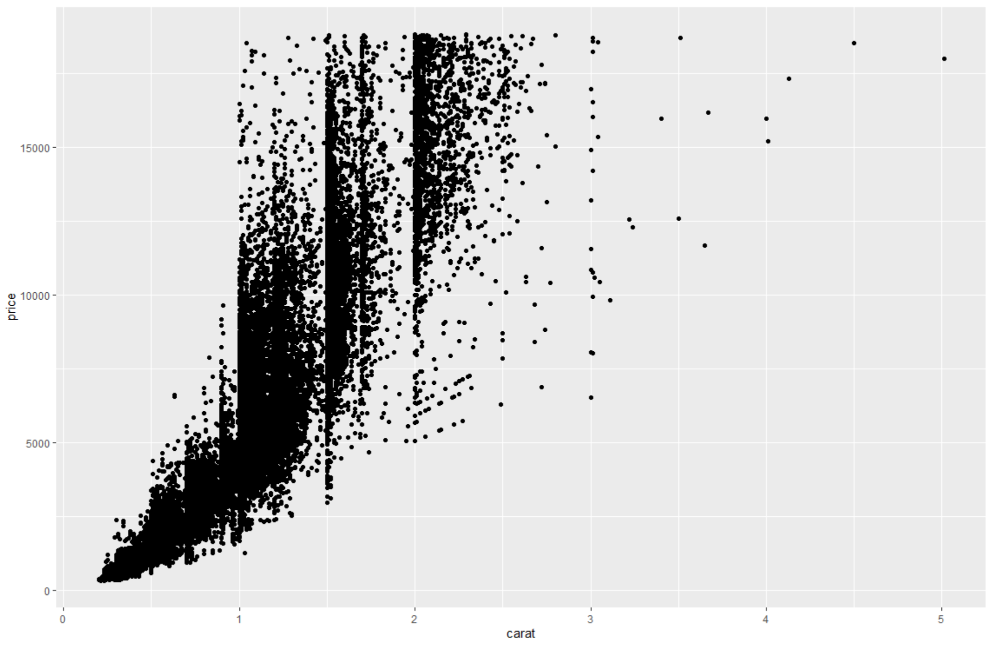
Data from Spotify

Bigger Datasets

So far, we've been looking at datasets with fairly manageable numbers of observations: there's 64 tracks on the 5 Beyonce albums we're looking at, for example. What happens if we have a lot more? Let's go back to the diamonds data from before, which has around 50,000 observations.

17. What's the relationship between carat and price?

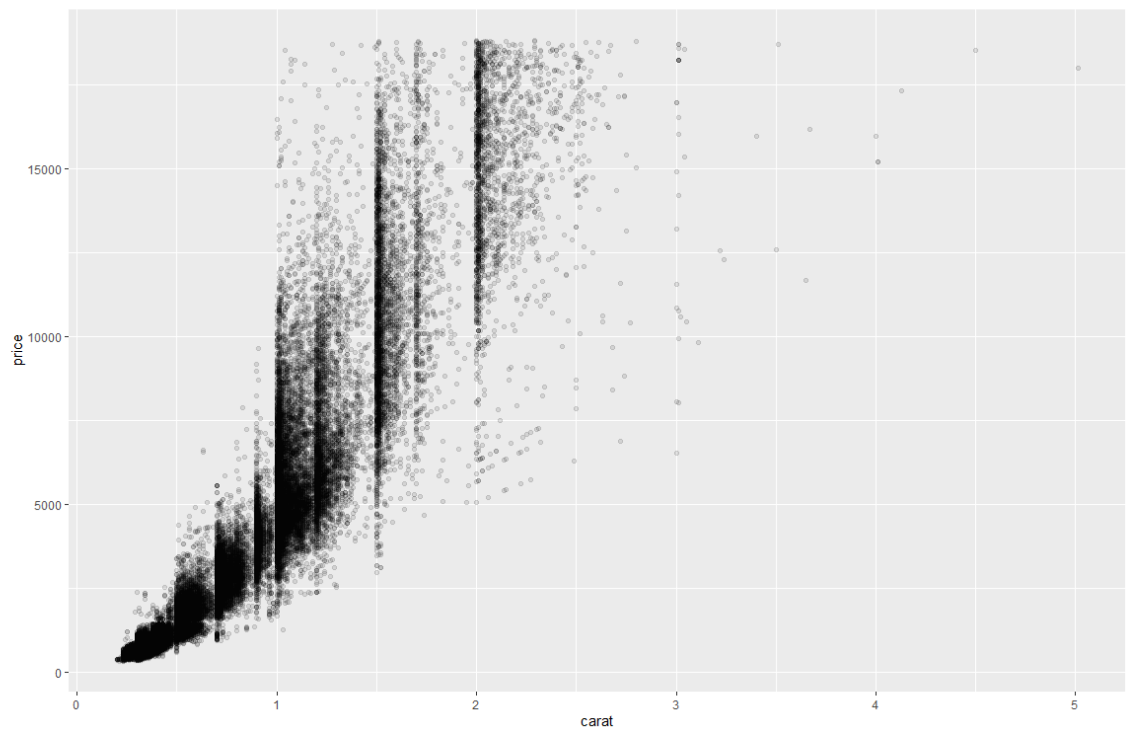
```
ggplot(data = diamonds) +
  aes(x = carat, y = price) +
  geom_point()
```



This is basically unreadable. There's so many observations that we can't figure out what's going on. However, what we can do is make the points semi-transparent. In the context of geometric objects, alpha refers to how opaque they are: if alpha = 1, they're fully opaque, if alpha = 0, they're invisible, between those they're semi- transparent.

18. Let's try an alpha value of 0.1: this means that if there's 10 or more points on top of each other, we'll end up with a fully opaque point.

```
ggplot(data = diamonds) +
  aes(x = carat, y = price) +
  geom_point(alpha = 0.1)
```



What do you think? What's the relationship between carat and price? What carats are most common among diamonds? Play around with the data and command to see what's revealed.

Bonus Exercises

1. Please answer the following questions in the form of a scatterplot:
   A. What's the relationship between energy and speechiness in Kendrick Lamar's albums? In Beyonce's?
   B. How does the relationship between energy and speechiness vary by album? (please use *facet* in answering this question)
2. Please answer A and B again for Rihanna.
3. Then, as with our last worksheet please come up with another question you're interested in answering using the available data we uploaded at the start of the worksheet (apart from Kendrick Lamar as we have used his work a lot already) that includes drawing a scatterplot. You can find out more about the Spotify variables at here: https://developer.spotify.com/documentation/web-api/reference/#endpoint-get-audio-features

DATA VISUALISATION

LAB WORKSHEET 4

Creator: Dr Mark Taylor

What's happening in this Document?

In the last two worksheets, we've looked at how to plot the relationship between two categorical variables (through different kinds of bar charts) and two continuous variables (through different kinds of scatterplots). We've also alluded to relationships between continuous and categorical variables, such as through our graph of how the tempo of Kendrick Lamar's tracks have changed with their release dates.

In this worksheet, we're focusing on how the distributions of continuous variables differ by categorical variables. You saw a bit of this in the first week, when we looked at density curves and boxplots; now we're going to do it a bit more thoroughly.

Getting set up

As with the last few weeks, we're going to start by loading some packages and some data. As with the last few weeks again, we're going to install and load a new package, ggridges. ggplot2 has loads of different extensions, and ggridges is one such example.

So, let's load some packages...

```
library(tidyverse)
library(ggridges)
library(lubridate)
```

Q-Step A step-change in quantitative social science skills Funded by the Nuffield Foundation, ESRC and HEFCE
Nuffield Foundation
Sheffield Methods Institute.
The University Of Sheffield.

You may need to install ggridges if you get an error message:

```
install.packages("ggridges")
```

Let’s load some data.

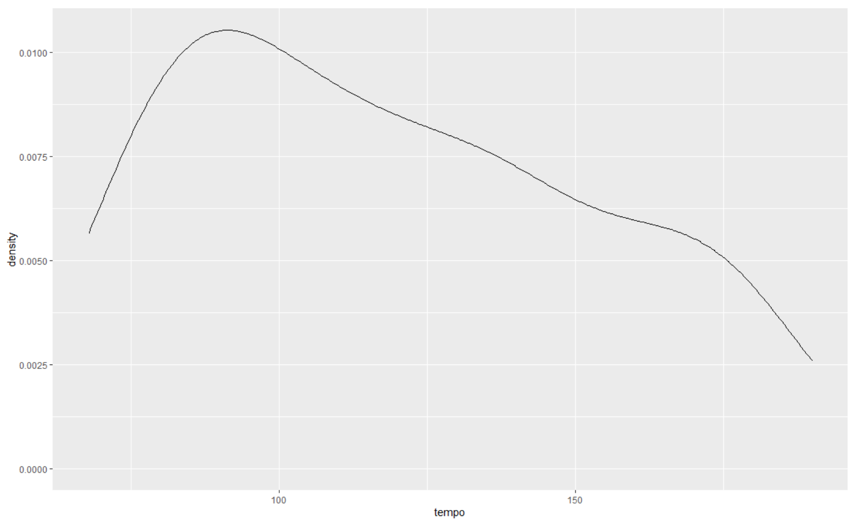
```
kendrick <- read_csv("https://bit.ly/kendrickdata")
kanye <- read_csv("https://bit.ly/kanyedata")
rihanna <- read_csv("https://bit.ly/rihannadata")
beyonce <- read_csv("https://bit.ly/beyonce_data")
queen <- read_csv("https://bit.ly/queen_data")
maccas <- read.csv("https://bit.ly/mcdonalds_data")
```

We are good to go! Let’s look at some distributions.

Looking at distributions

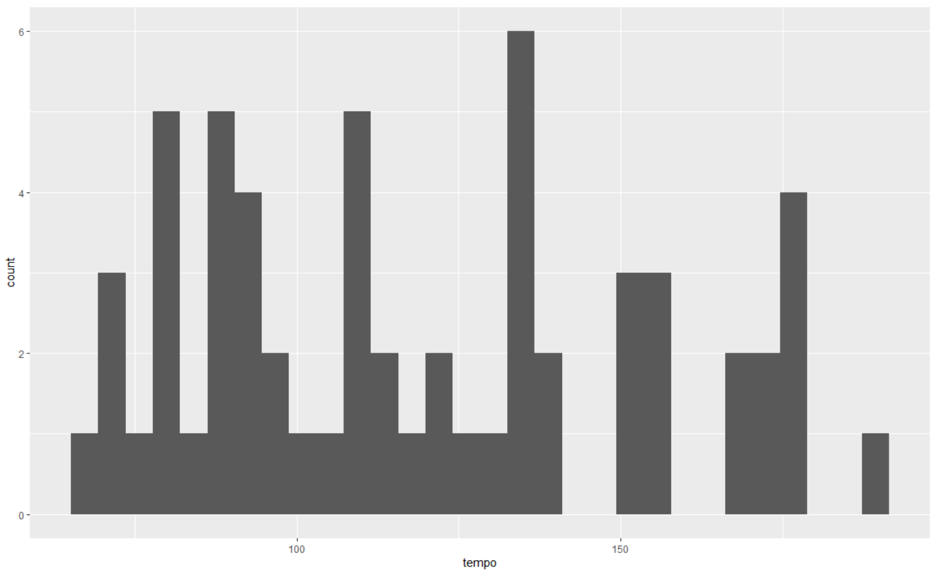
We’ve spent the last few weeks with the Kendrick Lamar data. Let’s continue this by looking at the distribution of tempo of his tracks:

```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_density()
```



2. If we want to look at the distribution of a single continuous variable, one approach is to use a density curve. Another is to use a histogram. Let's look at the same data with a histogram.

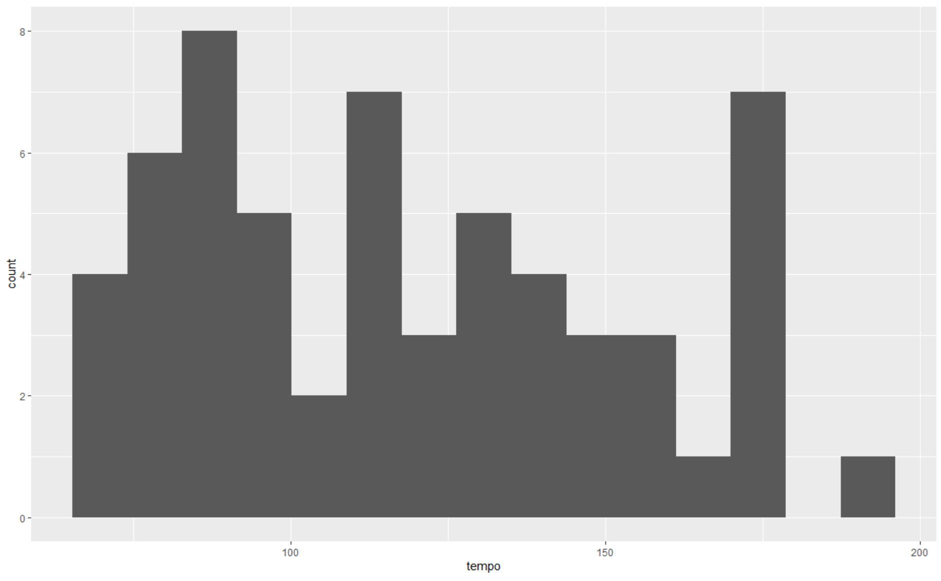
```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_histogram()
```

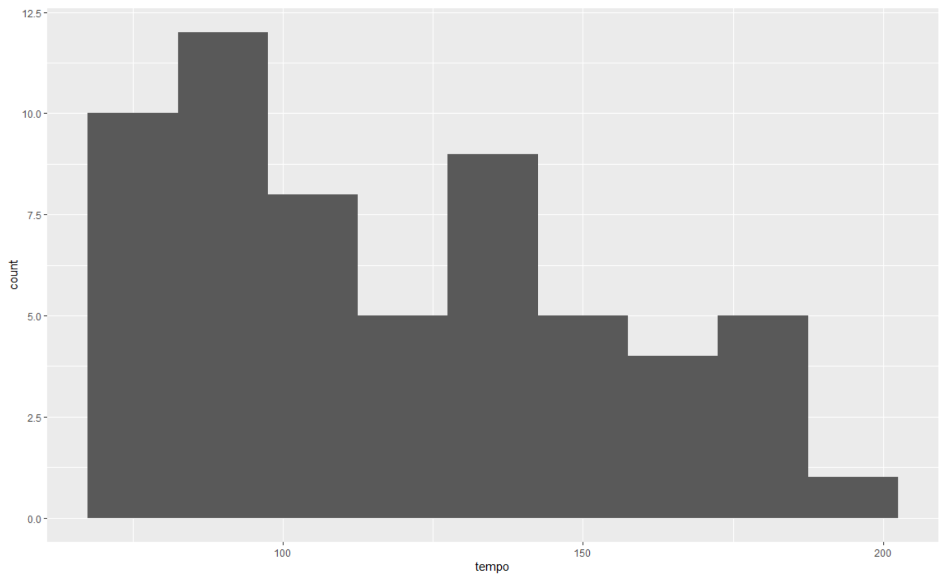


Once we run this, we end up with some red text in the console. This is fine, but it might invite us to think about bins (effectively, the bars that make up the histogram) – do we want them to be wider or narrower? Do we want more or fewer of them? Let's play around with bins and see what happens.

3. We can play around with them by specifying either the number or the width of the bins, like so:

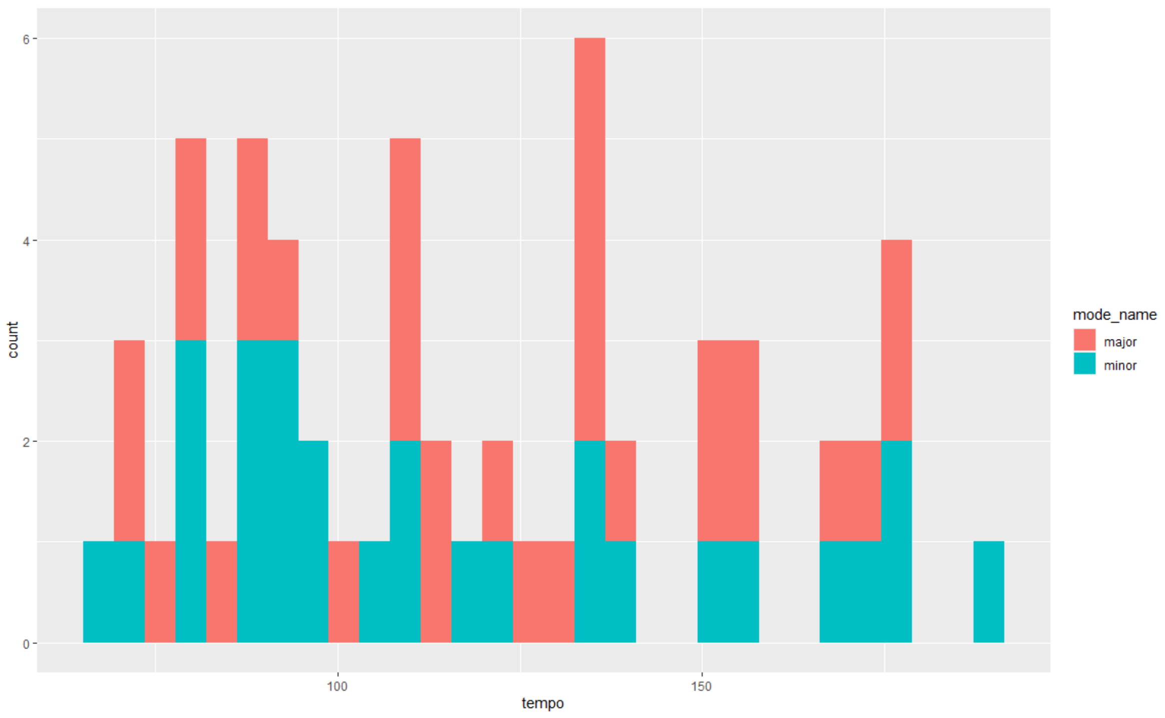
```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_histogram(bins = 15)

ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_histogram(binwidth = 15)
```





4. Time for a second variable. Are Kendrick Lamar’s minor tracks slower than his major tracks?

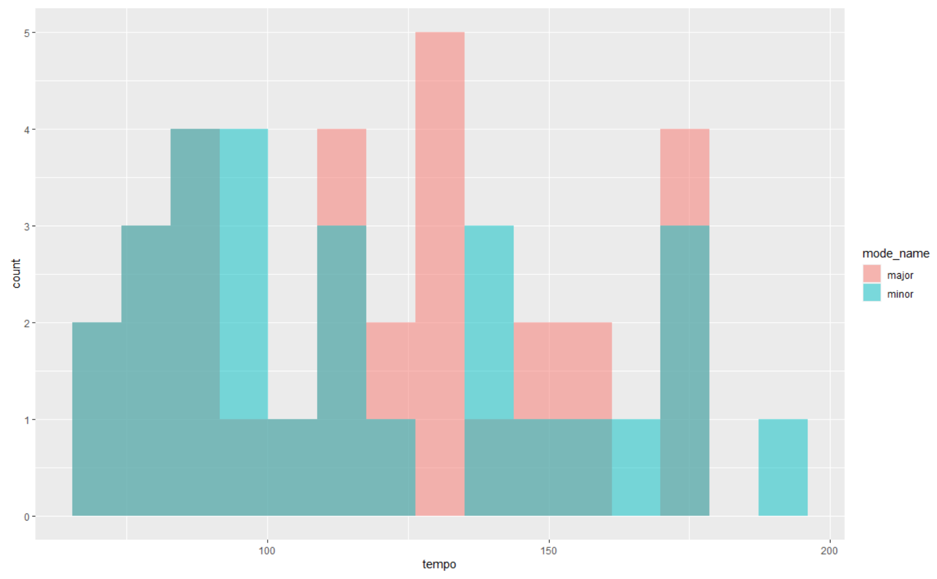
```
ggplot(data = kendrick) +
  aes(x = tempo, fill = mode_name) +
  geom_histogram()
```



We have talked about stacked bar charts before, and this seems even more severe.

5. Let's tweak this so that we can compare better.

```
ggplot(data = kendrick) +
  aes(x = tempo, fill = mode_name) +
  geom_histogram(bins = 15,
                 position = "identity",
                 alpha = .5)
```

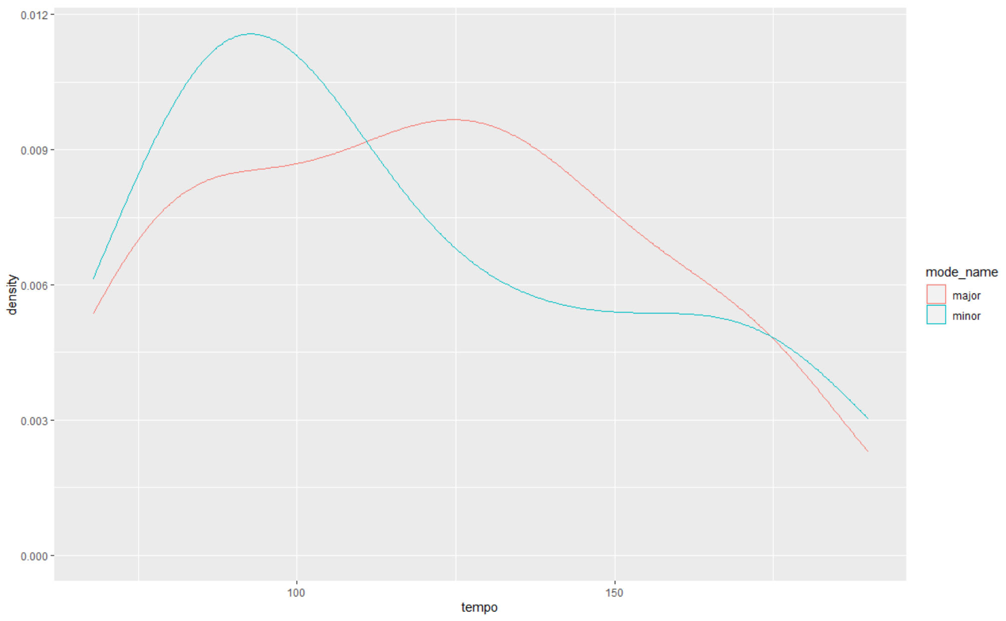


This helps, but has its own issues. Let's try some alternatives.

Back to density curves

6. Let's try answering that same question – are Kendrick Lamar's minor or major tracks quicker? – with some density curves instead.

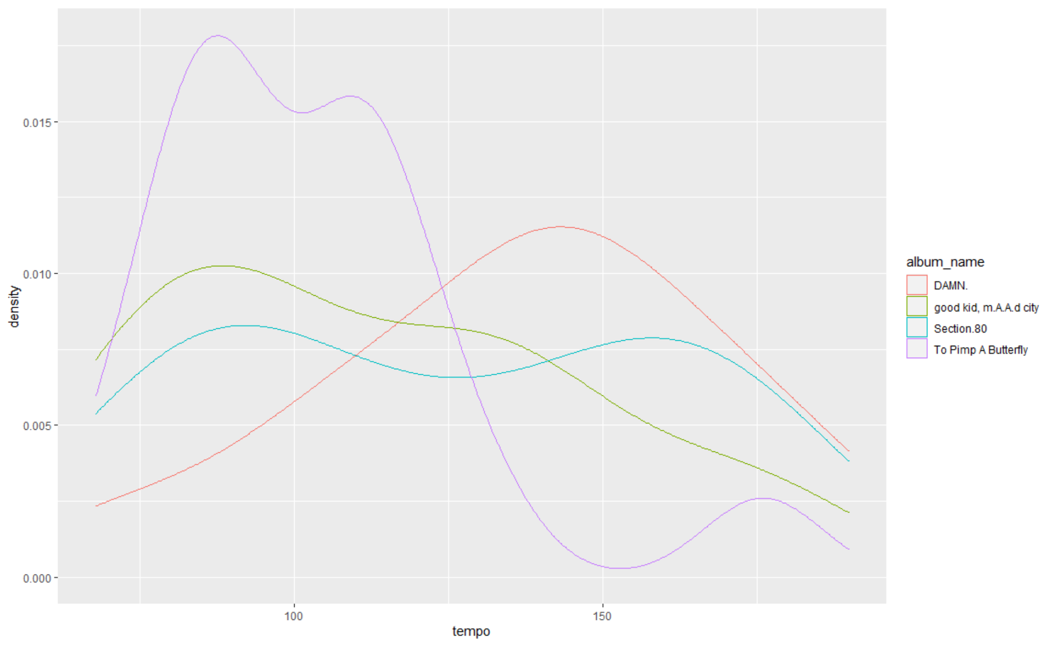
```
ggplot(data = kendrick) +
  aes(x = tempo, colour = mode_name) +
  geom_density()
```



This is a bit more like it! These curves adjust for the fact that there's different numbers of tracks in major and minor keys, and because they're curves rather than histograms we can see how they fit together more straightforwardly.

7. What about albums? Are Kendrick Lamar's more recent albums quicker?

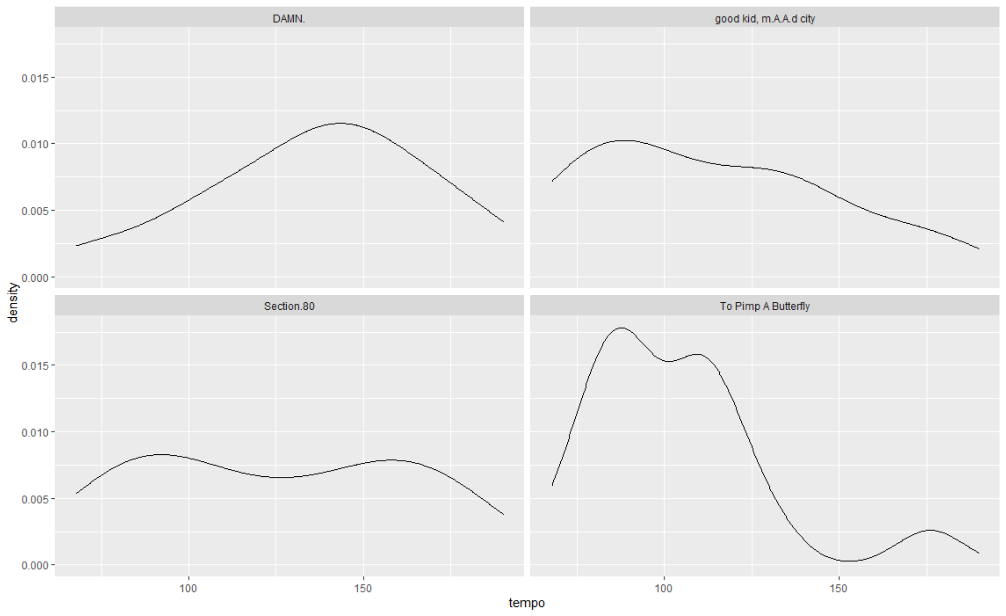
```
ggplot(data = kendrick) +
  aes(x = tempo, colour = album_name) +
  geom_density()
```



This doesn't work as well. Comparing two overlapping density curves is fairly straightforward to do, but comparing four is tricky; it's not as easy to identify the overall trends for all four. (It's also worth noting this comparison is a bit easier than it is for other artists - try it with the other data we've loaded).

8. As ever, a good way to deal with four lots of information in one graph is to turn it into four graphs, with the facet_wrap() command.

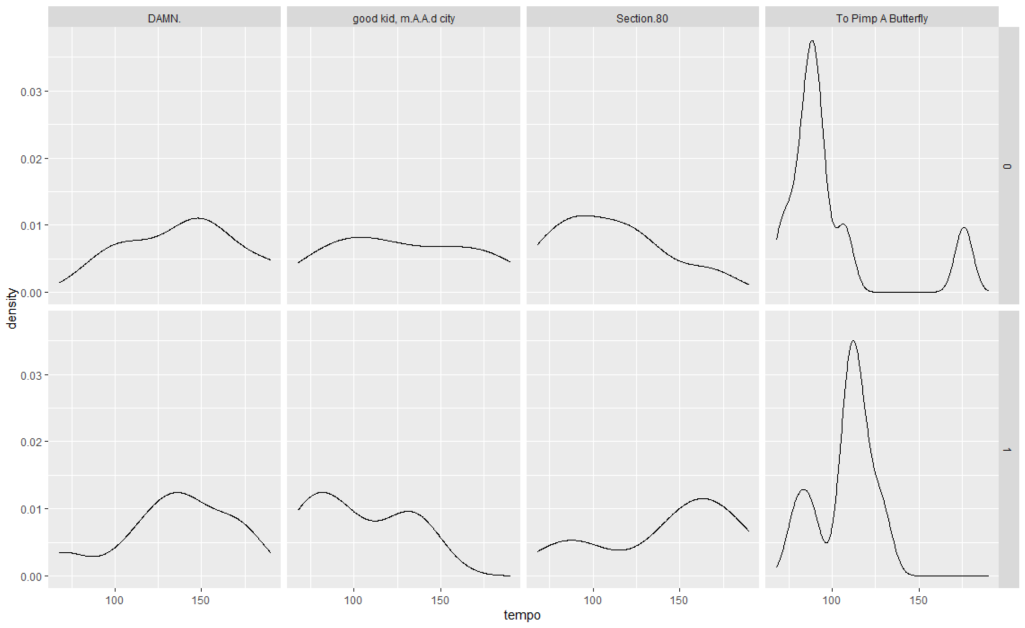
```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_density() +
  facet_wrap(~ album_name)
```



Did you ever wonder what the tilde – ~ – was doing in the facet_wrap() parenthesis, when with most other parentheses it's OK to just have the variable name?

9. As an example, let's look at how the tempo of Kendrick Lamar's tracks across his albums vary by whether they're in major or minor keys; this means we're faceting by two variables. It also means we're using facet_grid() rather than facet_wrap().

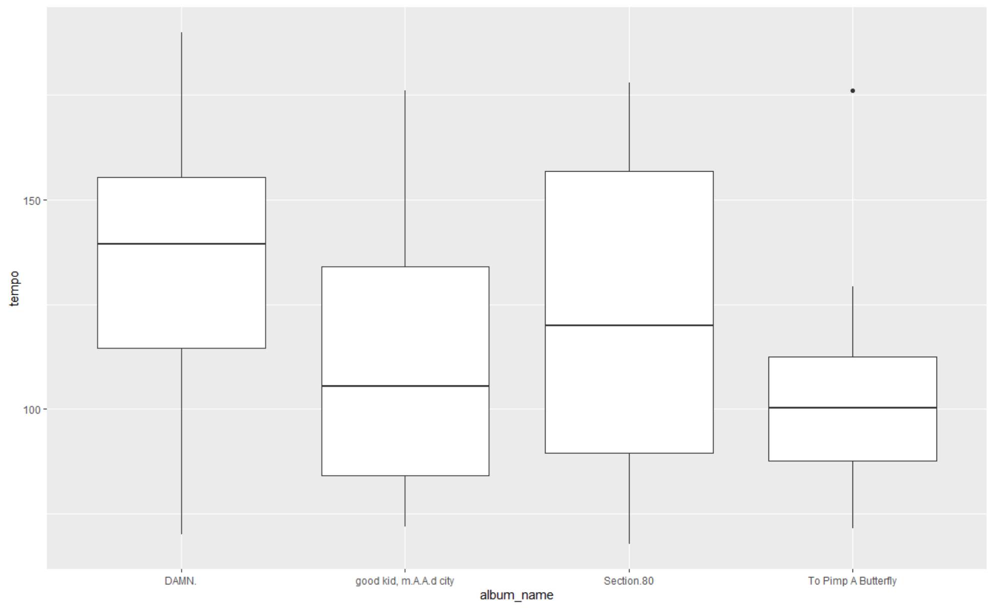
```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_density() +
  facet_grid(mode ~ album_name)
```

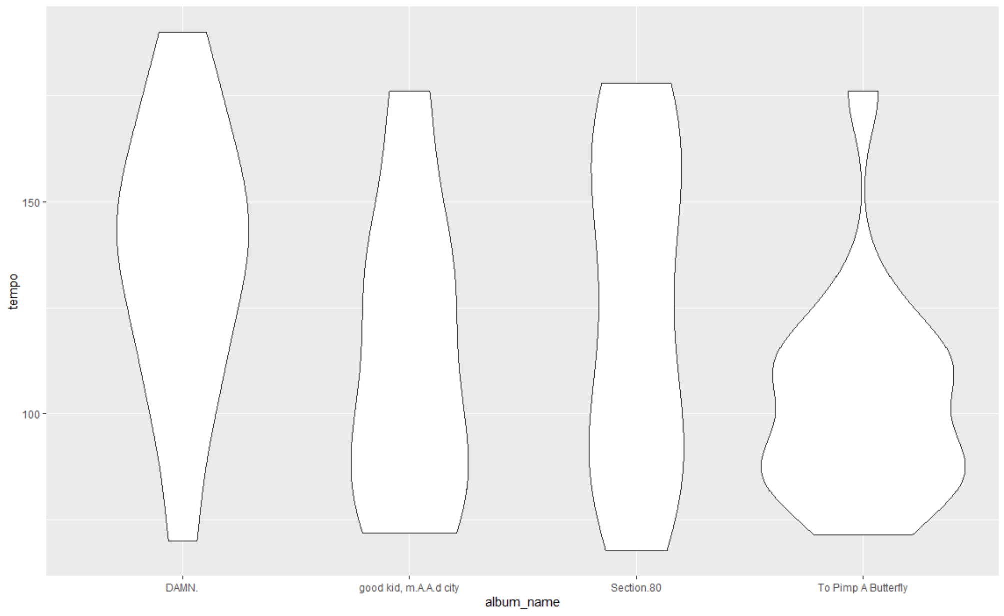


OK, we're making progress!

10. Let’s also consider two other ways to communicate this information: box plots and violin plots.

```
ggplot(data = kendrick) +
  aes(x = album_name, y = tempo) +
  geom_boxplot()

ggplot(data = kendrick) +
  aes(x = album_name, y = tempo) +
  geom_violin()
```

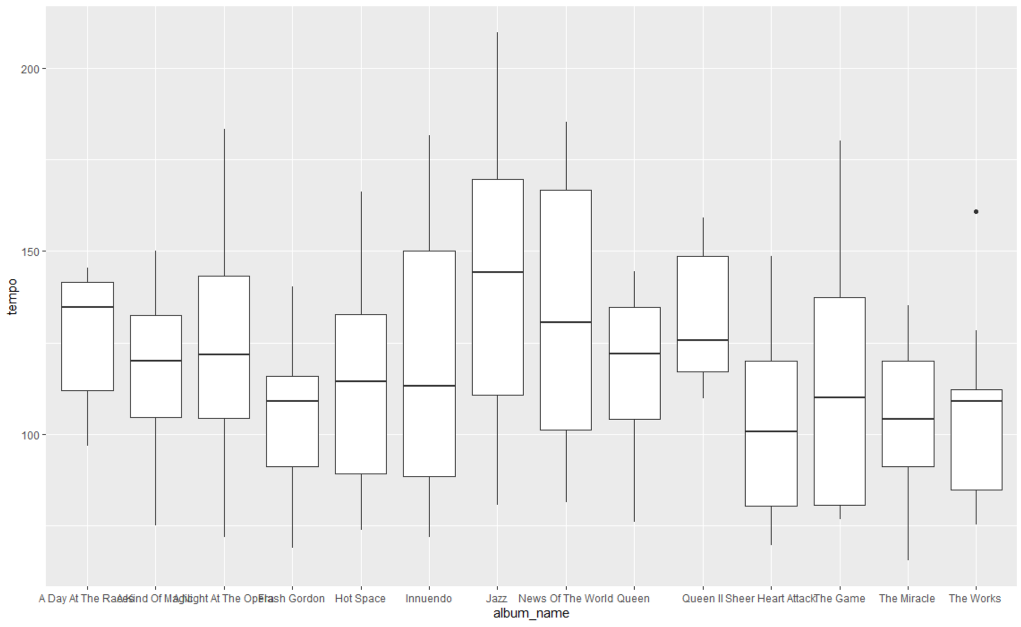




Moving to Queen

11. What happens when we're looking at a categorical variable with more than four levels? Queen released a lot of albums, let's look at Queen. (As with Worksheet 2, but different from the last worksheet, this is a version of the data where I've stripped out non-studio albums.)

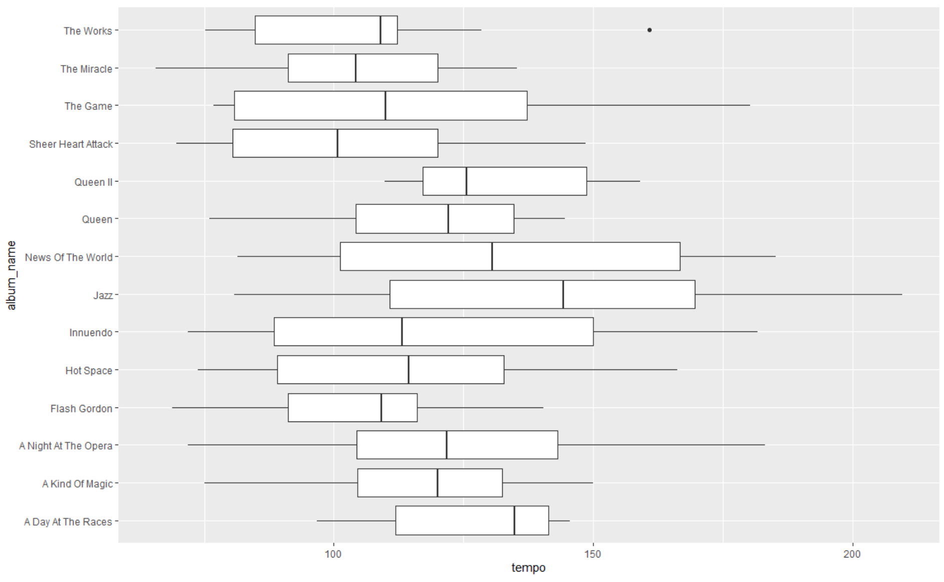
```
ggplot(data = queen) +
  aes(album_name, tempo) +
  geom_boxplot()
```



This is basically impossible to read because of the number of different albums.

12. One easy way around this problem is just to swap our axes over.

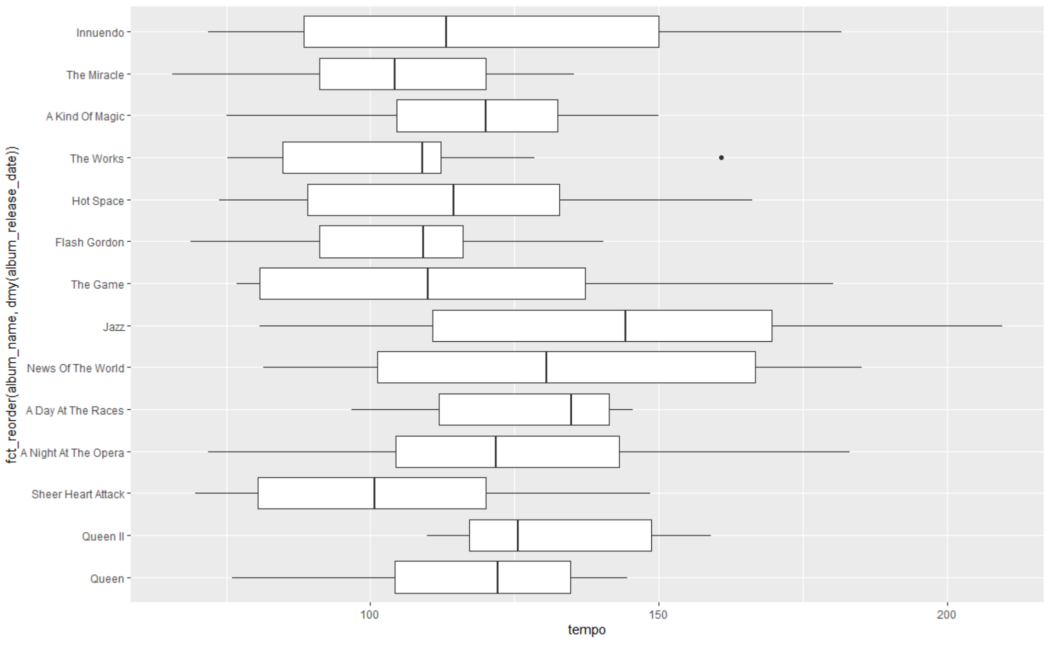
```
ggplot(data = queen) +
  aes(album_name, tempo) +
  geom_boxplot() +
  coord_flip()
```



But we want them in a different order; here, they're just chronological. In our last worksheet, I showed you the factor() command, which is a powerful way of organising your categorical variables. What I didn't show you is that sometimes it's easier. We can reorder our factor levels using the reorder() command

13. If we want to reorganise our albums based on when they came out, we can do it like so.

```
ggplot(data = queen) +
  aes(fct_reorder(album_name, dmy(album_release_date)),
      tempo) +
  geom_boxplot() +
  coord_flip()
```



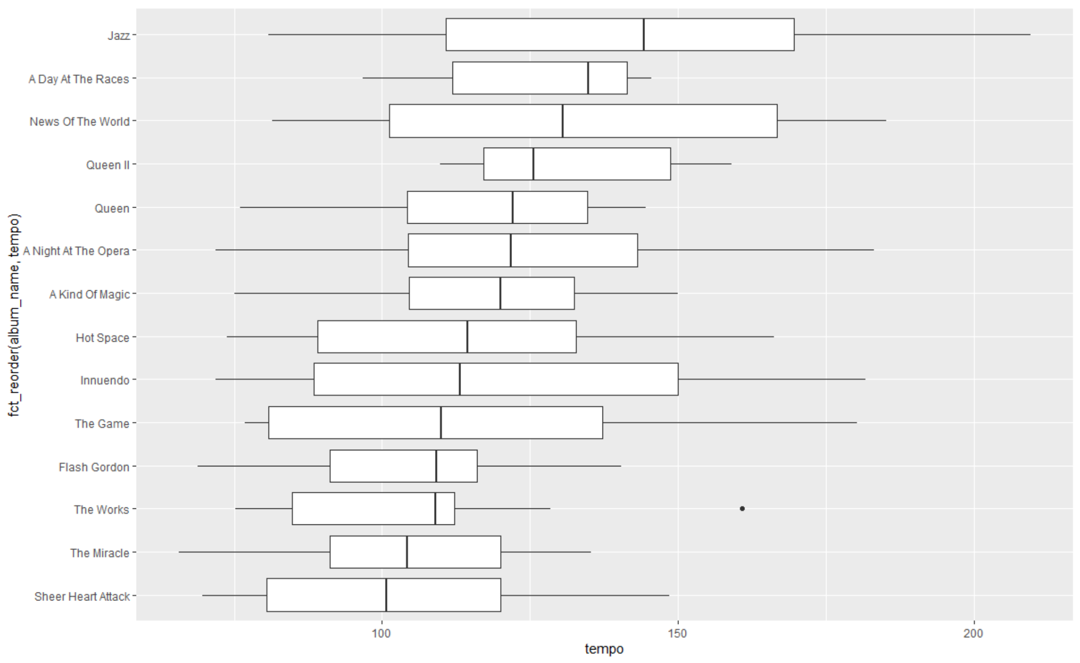
What's happened here? Everything's the same except the first mapping in the aes() parenthesis, so let's look through that.

- we start with fct_reorder(). That indicates we're using a categorical (or factor) variable, with levels that are different from the defaults.
- the first thing in the fct_reorder() bracket is the variable we want to put on that axis: album_name. But while we'd normally close the bracket here, instead we have a comma, anticipating...
- the variable the basis on which we're reordering that categorical variable. Here, we want to reorder according to when the album came out; that information's contained in
- album_release_date... which we need to surround with dmy(), to indicate it's a date of the format DD-MM-YYYY.

Exhausting? But, again, none of these individual bits is that complicated, there's just a lot of them.

14. And if we want to reorder the categories so that we're reorganising the boxplots from highest to lowest, we can do that with another reorder() command.

```
ggplot(data = queen) +
  aes(fct_reorder(album_name, tempo), tempo) +
  geom_boxplot() +
  coord_flip()
```

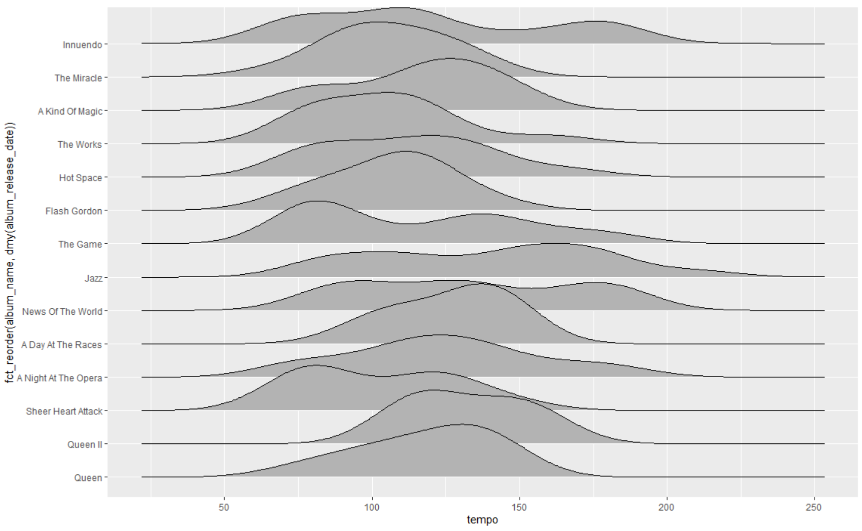


The difference here is that earlier we were reordering according to album_release_date; here, we're reordering according to tempo.

Ridgeline plots (Also known as joyplots.)

15. Did you manage to load ggridges earlier? Let's find out.

```
ggplot(data = queen) +
  aes(x = tempo,
      y = fct_reorder(album_name, dmy(album_release_date))) +
  geom_density_ridges()
```

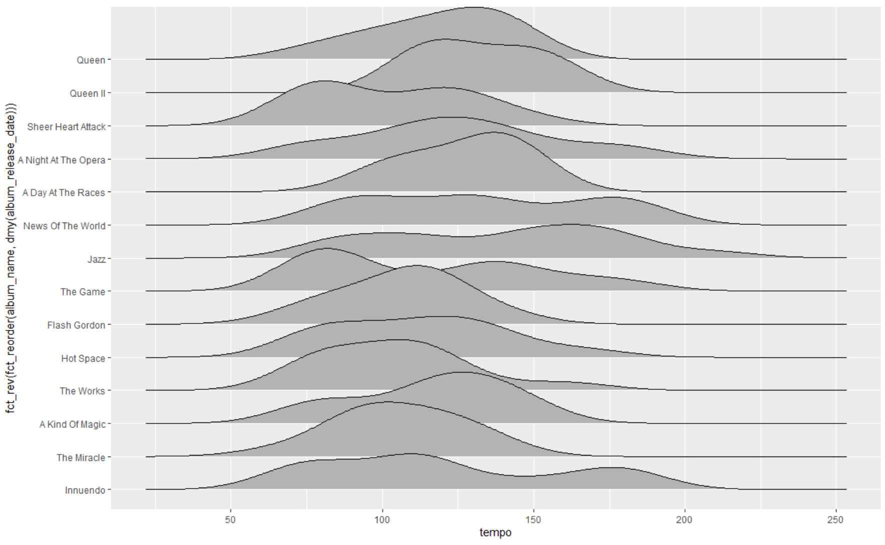


I really like ridgeline plots, which are a fairly new innovation in data visualisation. I think they make it more straightforward to compare distributions of a continuous variable across lots of different categories, as in this case (I like them for fewer categories as well, but I think they're particularly useful here).

However, one issue is that, by default, the y axis goes from smaller numbers at the bottom to bigger numbers at the top. Normally, when we're presenting a chronological list, the earliest items go at the top, and the latest at the bottom; here, we've effectively got the opposite.

16. So, let's turn this graph upside down.

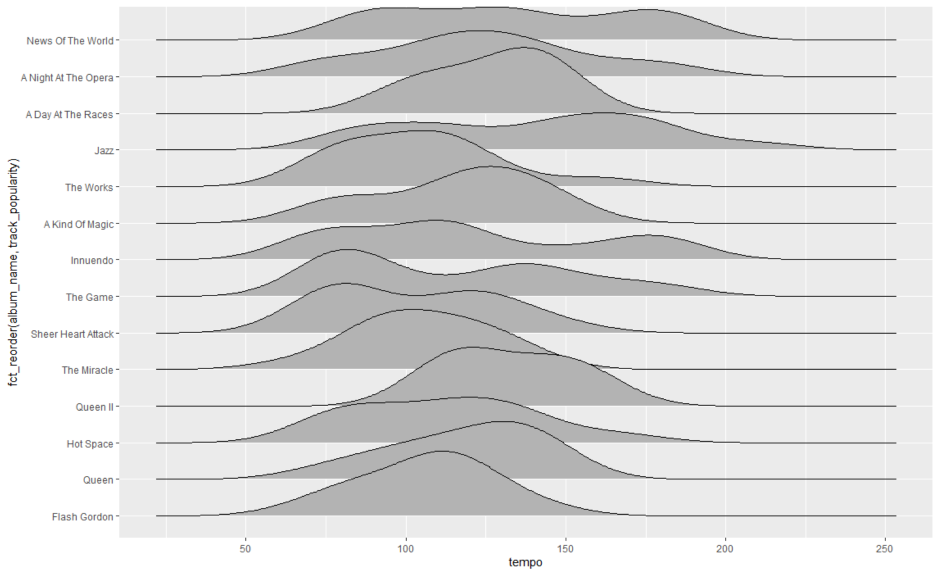
```
ggplot(data = queen) +
  aes(x = tempo,
      y = fct_rev(fct_reorder(album_name,
                              dmy(album_release_date)))) +
  geom_density_ridges()
```



This is clumsy, but it works. What we've done is we've wrapped the existing y aesthetic mapping – reorder(album_name, dmy(album_release_date)) – which included a lot more brackets than we're used to, in yet another set of brackets, preceded by the **fct_rev()** command. fct_rev() reverses the order of levels in a factor, so it's often useful in this kind of context.

17. We can also reorder from high to low, as we did before.

```
ggplot(data = queen) +
  aes(x = tempo,
      y = fct_reorder(album_name, track_popularity)) +
  geom_density_ridges()
```



Generating new variables

A lot of the time, we can make all the visualisations we want based not only on a single dataset, but also on variables provided in the dataset. Sometimes, though, we need to generate new variables. Let's look at an example.

```
names(maccas)
```

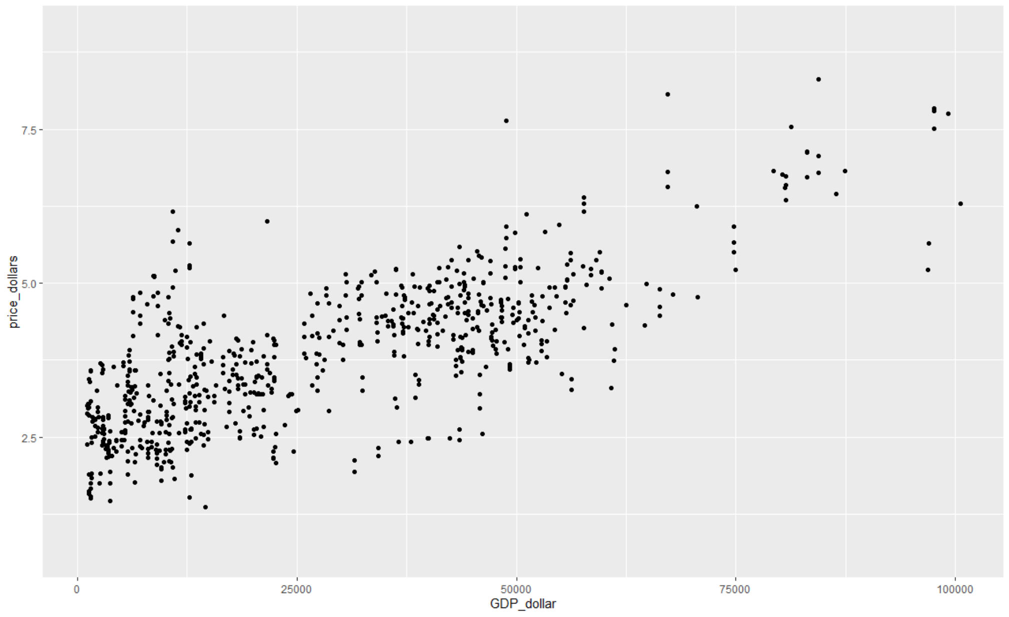
The **maccas** data is what the Economist's Big Mac index is based on; it's a measure of purchasing power in different parts of the world. (You can read more about the Big Mac index, where the data's from, and how it's compiled at this link (https://github.com/TheEconomist/big-mac-data).

You can see it contains a few different variables: the name of the country, the currency code, the price of a Big Mac in the local currency, the country's GDP in US dollars, the exchange rate between the local currency and the US dollar, and the date of the observation. Maybe we want to know what the relationship is between GDP and the price of Big Macs; are they cheaper in countries with lower GDPs?

One variable that isn't included is the price of a Big Mac in US dollars. A pound sterling's worth (just) more than a US dollar; an Australian dollar's worth less than a US dollar. So comparing prices in local currencies isn't that useful; we need a variable for how much a Big Mac costs in a single currency. And we might as well use US dollars (though we don't have to).

18. Let's start by generating a new variable, using the **mutate()** command.

```
maccas %>%
  mutate(price_dollars = local_price/dollar_ex) %>%
  ggplot() +
  aes(x = GDP_dollar, y = price_dollars) +
  geom_point()
```

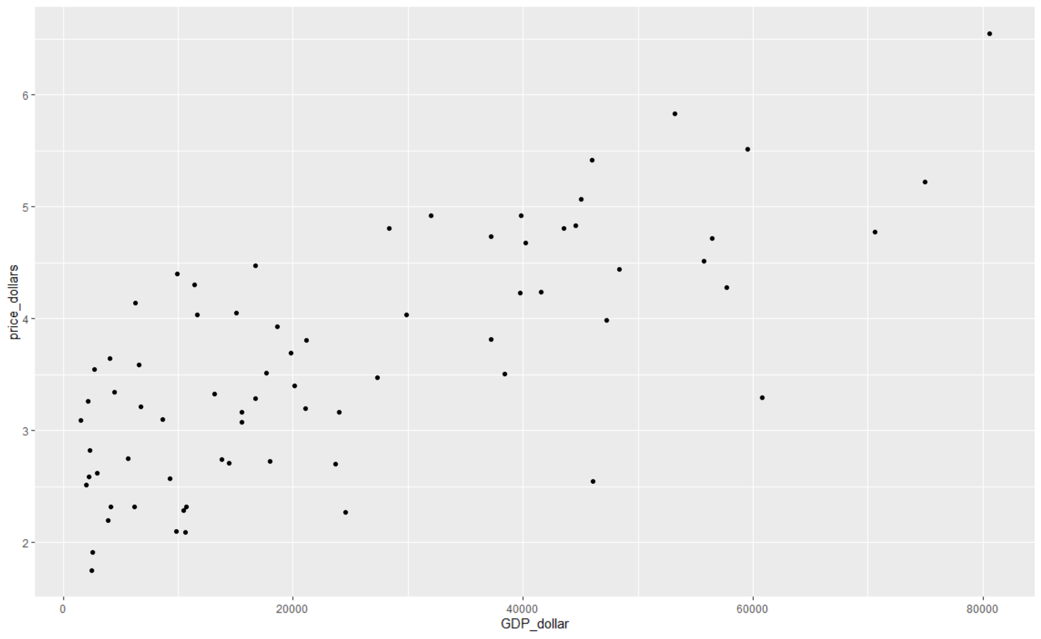


This is similar to what we've seen before; we've started with data, added a pipe, and specified what we want to do to the data. Here, we want a new variable, which I've called price_dollars: I've generated this by dividing the local price by the dollar exchange rate. The forward slash - / - means divide by; you can add variables with +, subtract with -, and multiply with *.

A problem here, though, is the number of observations.

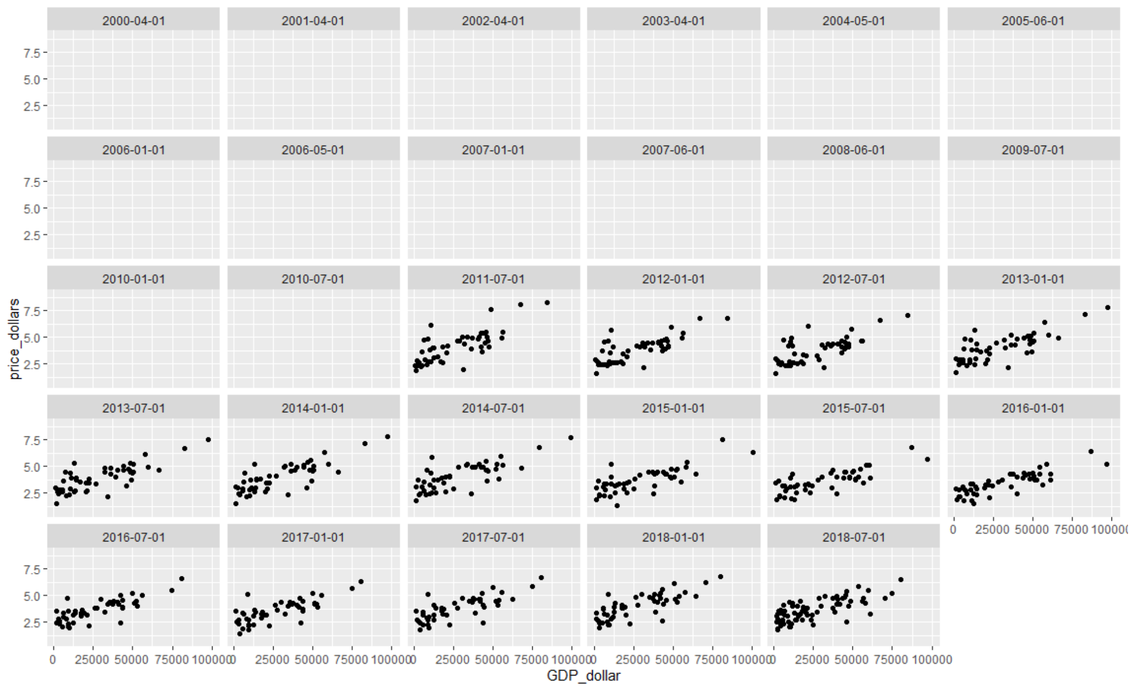
19. What if we just want the current relationship? To do this, we can limit our observations to the most recent wave of data, by going back to filter():

```
maccas %>%
  mutate(price_dollars = local_price/dollar_ex) %>%
  filter(date == "2018-07-01") %>%
  ggplot() +
  aes(x = GDP_dollar, y = price_dollars) +
  geom_point()
```



20. What if we wanted to know how this relationship had changed over time? An obvious starting point is to facet by date.

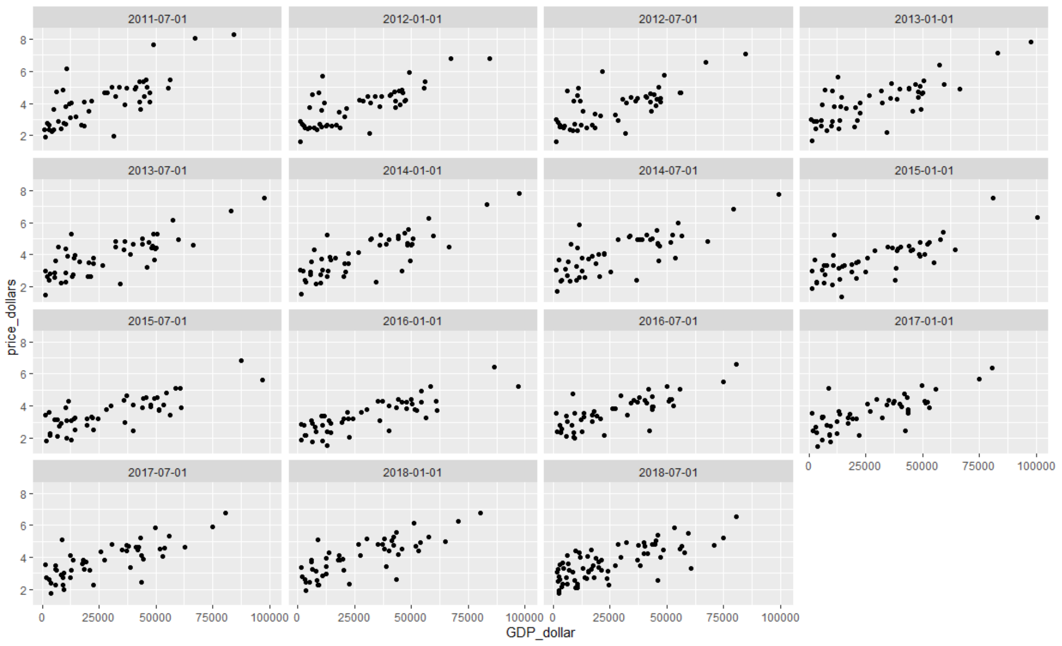
```
maccas %>%
  mutate(price_dollars = local_price/dollar_ex) %>%
  ggplot() +
  aes(x = GDP_dollar, y = price_dollars) +
  geom_point() +
  facet_wrap(~ date)
```



There's an obvious problem here, though. Not all the observations include all the data we want, which is why loads of our facets are empty. To get around this problem, we can use na.omit: this means we're just looking at complete cases, rather than cases with missing data.

21. Here's how we use it.

```
na.omit(maccas) %>%
  mutate(price_dollars = local_price/dollar_ex) %>%
  ggplot() +
  aes(x = GDP_dollar, y = price_dollars) +
  geom_point() +
  facet_wrap(~ date)
```



Bonus Exercise

There is only one bonus task for this worksheet, but it's a tricky one.

Please draw a **ridgeline plot** of the prices of a Big Mac in each country in the world, expressed as price in dollars as a fraction of GDP per head in dollars in each of those countries, sorted highest-to-lowest. So I want to see a graph of the range of those values in each country over the period that the Economist had data for, with the country where a Big Mac costs the largest proportion of GDP per head at the top, and the country where a Big Mac costs the smallest proportion of GDP per head at the bottom.

There will be some countries with missing ridgelines. This is fine - these are the cases where there's only one or two observations per country, and ggridges can't estimate a density curve with data that's so limited.

DATA VISUALISATION

LAB WORKSHEET 5

Creator: Dr Mark Taylor

What's Happening in this Document?

In this worksheet we're going to have a bit of a change of pace. In the last few weeks, we've introduced how to graph different kinds of variables and different kinds of relationships, including thinking about different ways to get our data into the kind of format that works for the graphs we want to draw.

While the graphs we've been drawing have been looking increasingly professional, they haven't been very flexible: we've been using default colour schemes, default layouts, and so on. While we looked at how to add labels, and change the order of levels in a factor, we haven't done much else. What if you like different colour schemes? What if you don't like your scatterplots to have grey backgrounds?

We also haven't discussed what we do once we've made our graphs. I have showed you how to export graphs but are there other ways? And how do we integrate them into other things we're doing? Let's address those issues now.

Getting Started

You know how this works at this point! Let's load some packages, including a new one:

```
library(tidyverse)
library(ggthemes)
library(lubridate)
```

Q-Step
A step-change in quantitative social science skills
Funded by the Nuffield Foundation, ESRC and HEFCE
Nuffield Foundation
Sheffield Methods Institute.
The University Of Sheffield.

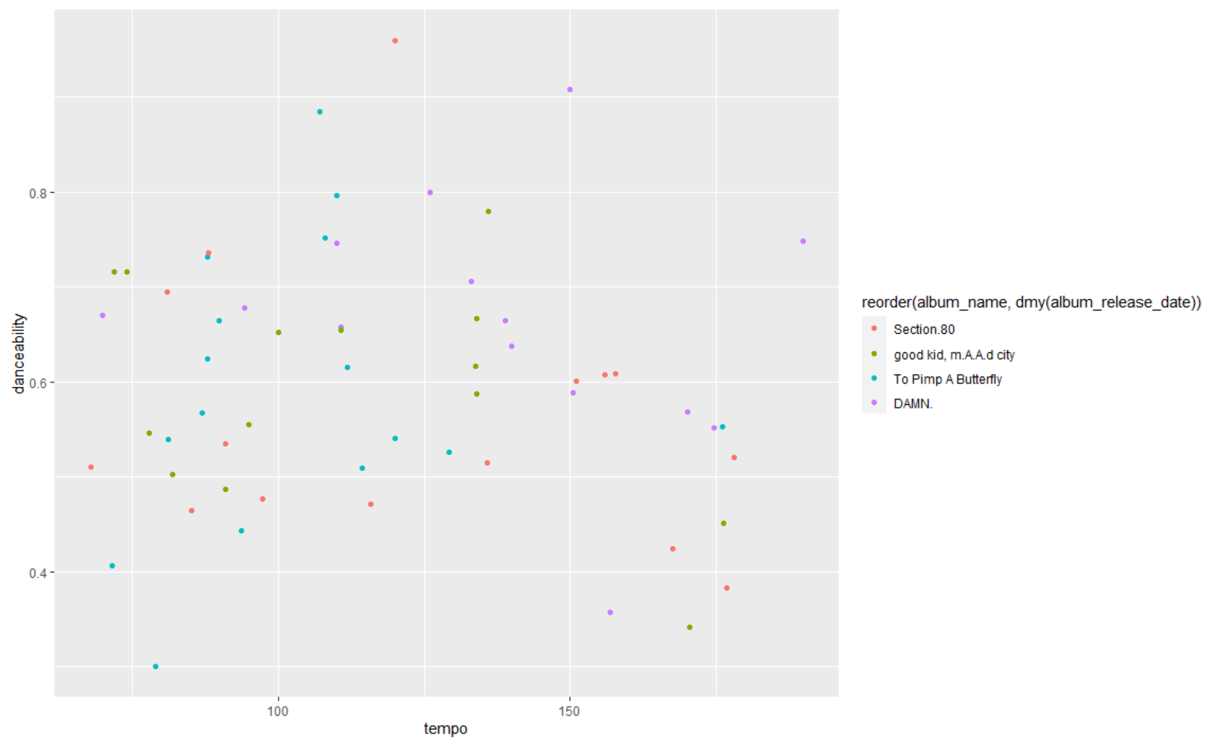
Now let's load some data:

```
kendrick <- read_csv("https://bit.ly/kendrickdata")
kanye <- read_csv("https://bit.ly/kanyedata")
rihanna <- read_csv("https://bit.ly/rihannadata")
beyonce <- read_csv("https://bit.ly/beyonce_data")
queen <- read_csv("https://bit.ly/queen_data")
```

Back to scatterplots, with some colour

1. Let's remind ourselves of the relationship between tempo and danceability among Kendrick Lamar's tracks, coloured according to which album they're on.

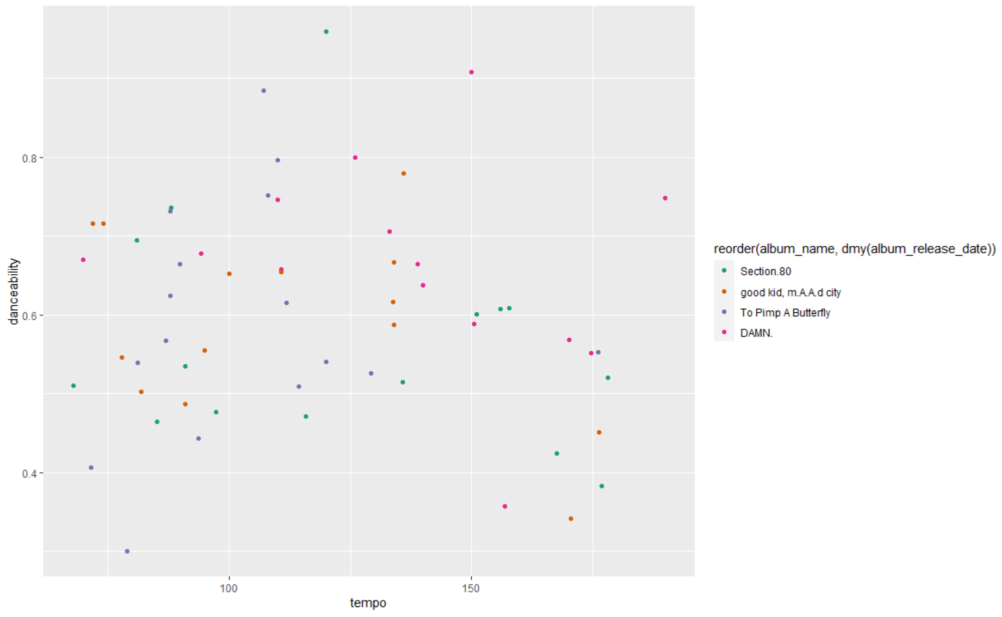
```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = reorder(album_name,
                       dmy(album_release_date))) +
  geom_point()
```



This looks OK (except for the legend taking up more than half the space, but you'll remember how we can deal with this issue.) But what if we want to use a different colour scheme?

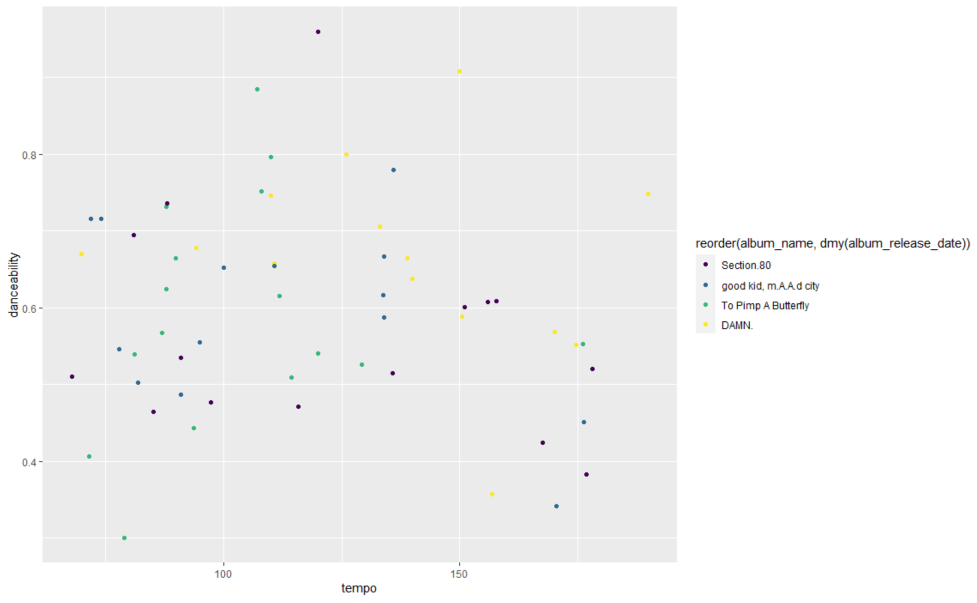
2. Let's start with Colourbrewer and Viridis. Both come preloaded with ggplot2, so we don't need to do anything special to load them. Let's start with a colour scheme from ColorBrewer:

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = reorder(album_name,
                       dmy(album_release_date))) +
  geom_point() +
  scale_colour_brewer(palette ="Dark2")
```



... and now let's follow it up with a colour scheme from viridis:

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = reorder(album_name,
                       dmy(album_release_date))) +
  geom_point() +
  scale_colour_viridis_d(option ="D")
```

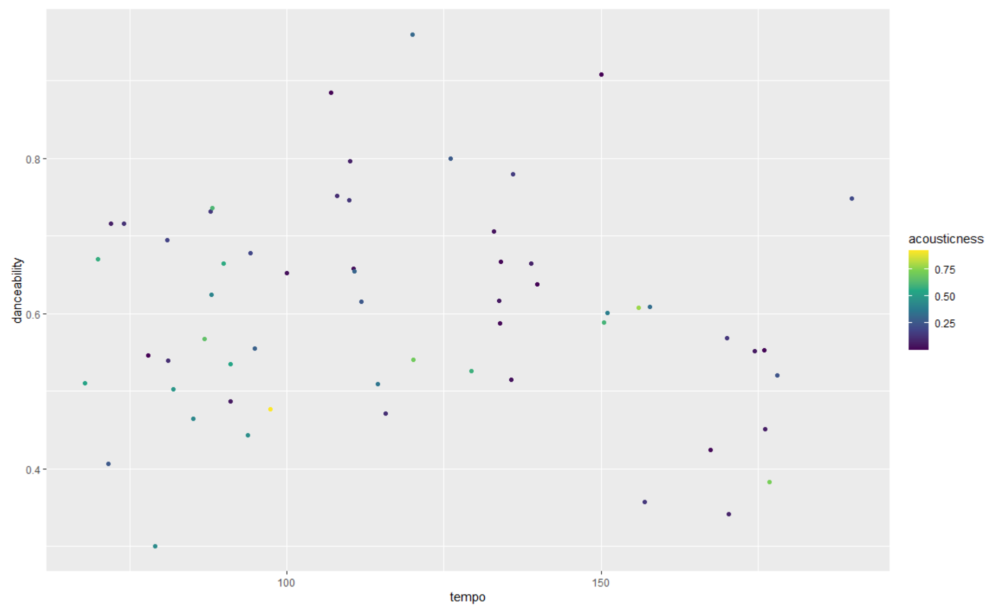


With viridis, guessing how you might find the other colour schemes is easy: you just swap out "D" for different letters of the alphabet and see what happens. For ColorBrewer, it's a bit less obvious: Google ColorBrewer and R and the full list of palettes will come up.

3. One other thing to mention on these colour schemes is that, with ColorBrewer, you specify the colour scheme to use in the same way regardless of whether you're using a continuous or discrete colour scheme. With viridis, it's a bit different; if you're mapping colour to a *discrete* variable, you want to add scale_colour_viridis_d(); if you're mapping colour to a *continuou*s variable, you want to add

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = acousticness) +
  geom_point() +
  scale_colour_viridis_c(option = "D")
```

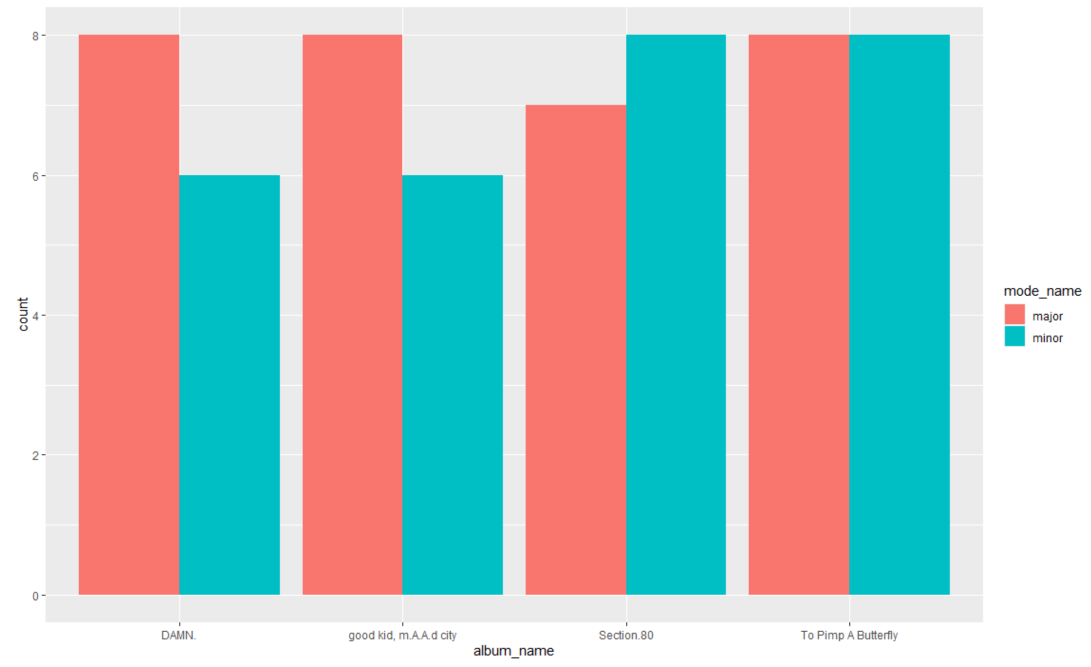
   scale_colour_viridis_c().



Back to bar charts, with some colour

4. Let's revisit another graph, from a few worksheets ago: the balance between major and minor on each of Kendrick Lamar's albums.

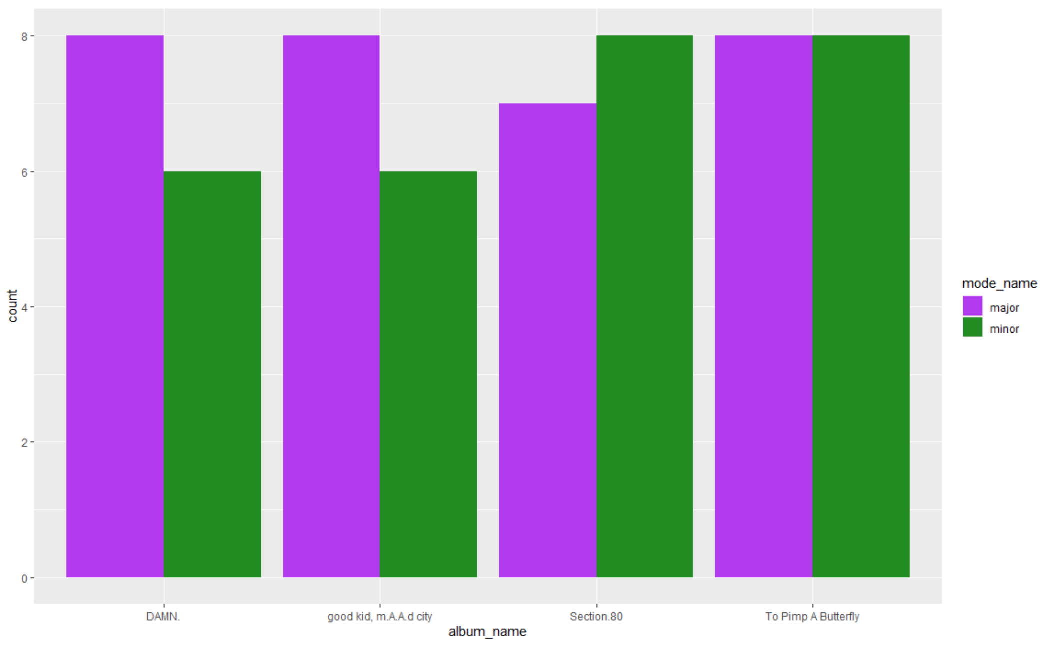
```
ggplot(data = kendrick) +
  aes(album_name, fill = mode_name) +
  geom_bar(position = "dodge")
```



5. What if you wanted your own colour scheme? What if you thought of major tracks being a particular shade of purple, and minor tracks being a particular shade of green?

```
ggplot(data = kendrick) +
  aes(album_name, fill = mode_name) +
  geom_bar(position = "dodge") +
  scale_fill_manual(values = c("darkorchid2",
                               "forestgreen"))
```

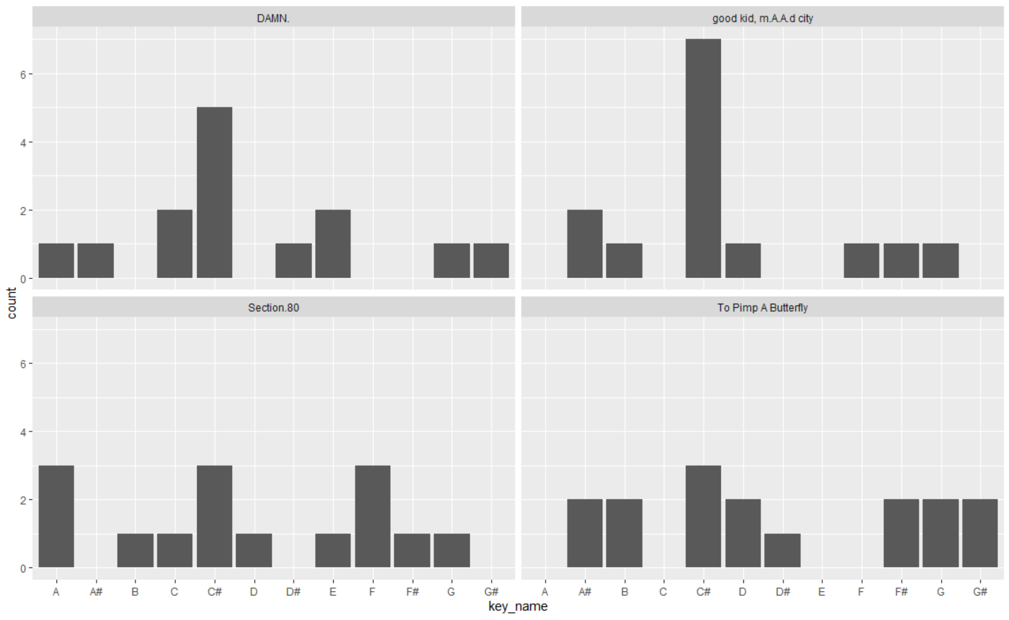
You could do something like this, with scale_fill_manual. (You can imagine a similar thing for scale_colour_manual if you were colouring a scatterplot.) You might wonder how I knew to use the colours darkorchid2 and forestgreen. All the colours available in R are available at this guide (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf).



Tidying up facets

6. Let's revisit facets, by looking at how the distribution of keys varies across Kendrick Lamar albums.

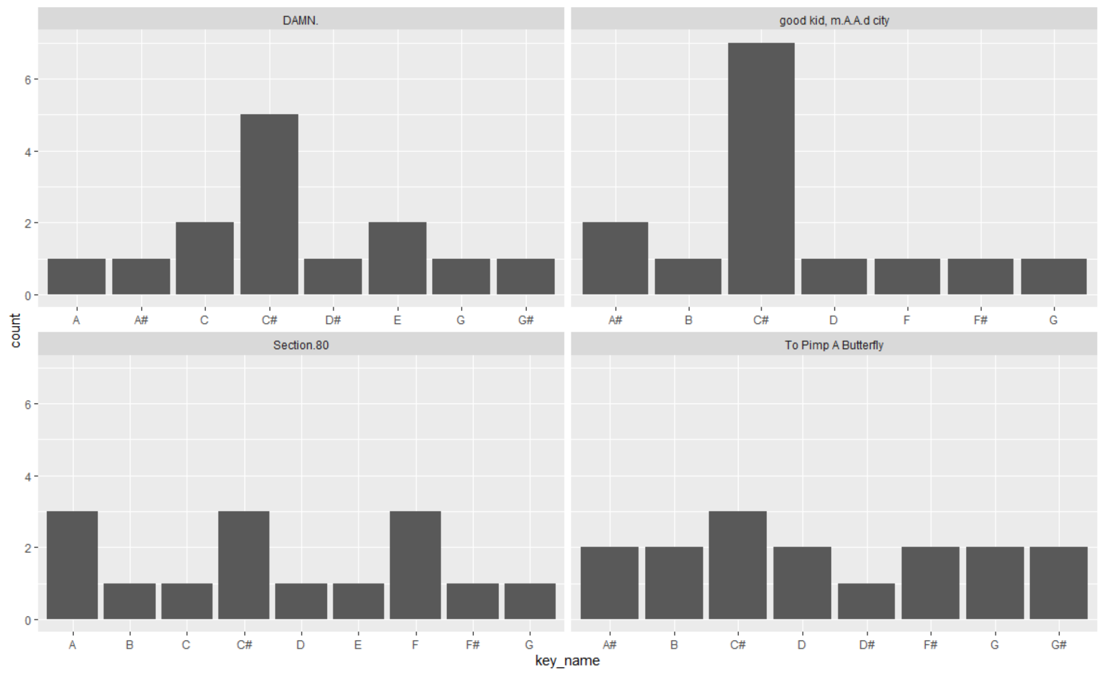
```
ggplot(data = kendrick) +
  aes(x = key_name) +
  geom_bar() +
  facet_wrap(~album_name)
```

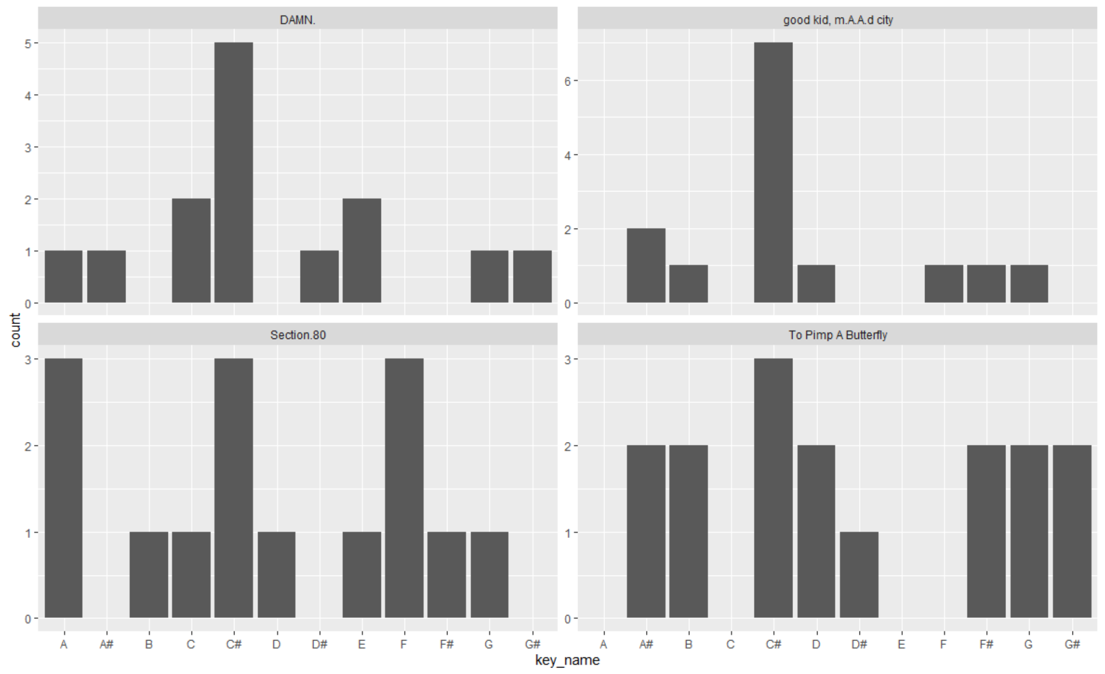


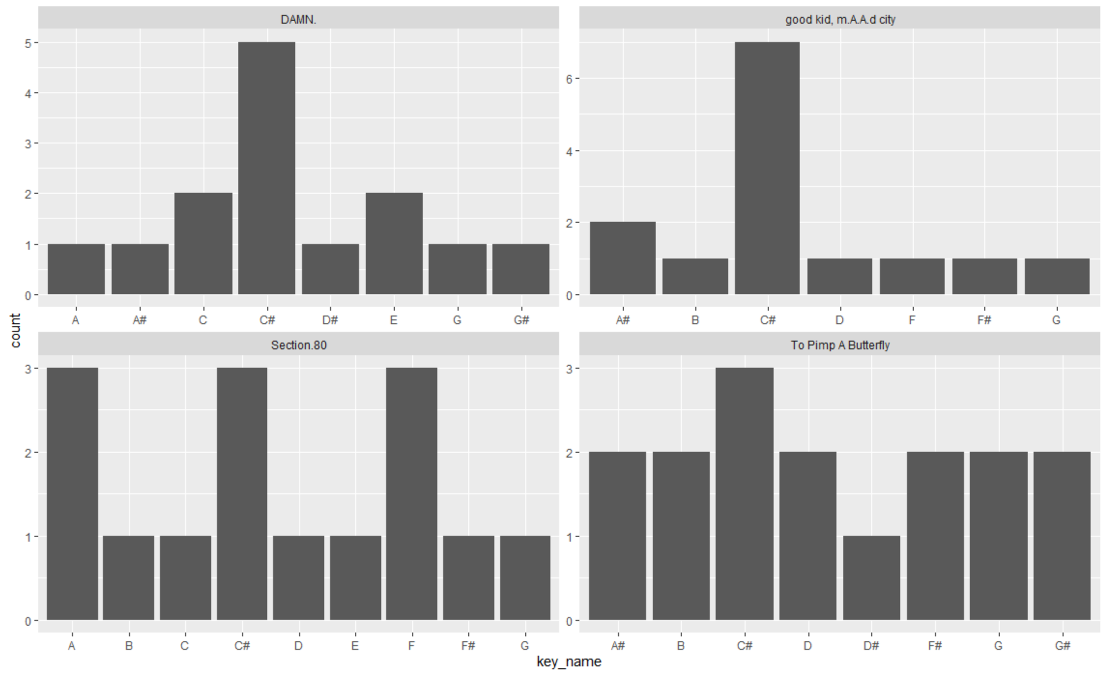
Any problems with this? One issue is that there's several albums that don't include every key. This isn't a problem per se, but you can imagine other settings where it is, particularly if you're looking at a categorical variable that varies a lot according to your faceting variable. You can also imagine settings where the number of observations within facets varies a lot: under those circumstances, having a common y-axis might make it difficult to understand what's going on in some facets.

7. What we can do is free up the axes: x, y, or both? Here's how we do it. Compare the following three options: which one works best? Which might work best in other contexts?

```
ggplot(data = kendrick) +
  aes(x = key_name) +
  geom_bar() +
  facet_wrap(~ album_name,
             scales = "free_x")

ggplot(data = kendrick) +
  aes(x = key_name) +
  geom_bar() +
  facet_wrap(~album_name,
             scales = "free_y")

ggplot(data = kendrick) +
  aes(x = key_name) +
  geom_bar() +
  facet_wrap(~album_name,
             scales = "free")
```


DAMN.
good kid, m.A.A.d city
Section.80
To Pimp A Butterfly
count
key_name


DAMN.
good kid, m.A.A.d city
Section.80
To Pimp A Butterfly
count
key_name

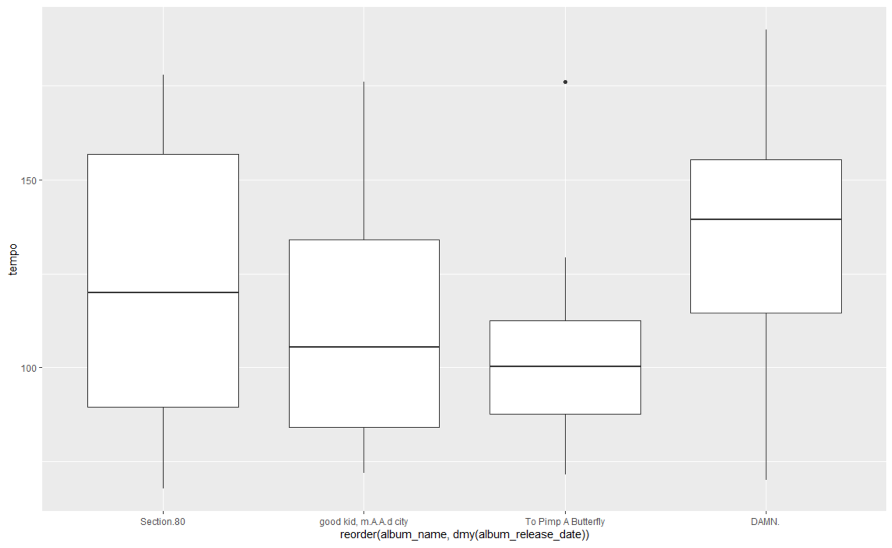

DAMN.
good kid, m.A.A.d city
Section.80
To Pimp A Butterfly
count
key_name

Who likes Tufte?

Edward Tufte is considered by many as a pioneer in the field of data visualization) and some people really like Tufte. If you're one of them, you might find yourself thinking "these boxplots use up too much ink, I wish ggplot2 would let me use the Tufte approach". It's easier to make your graphics look like Tufte's if you've loaded the ggthemes package (which hopefully you were able to do from the start of this workshop).

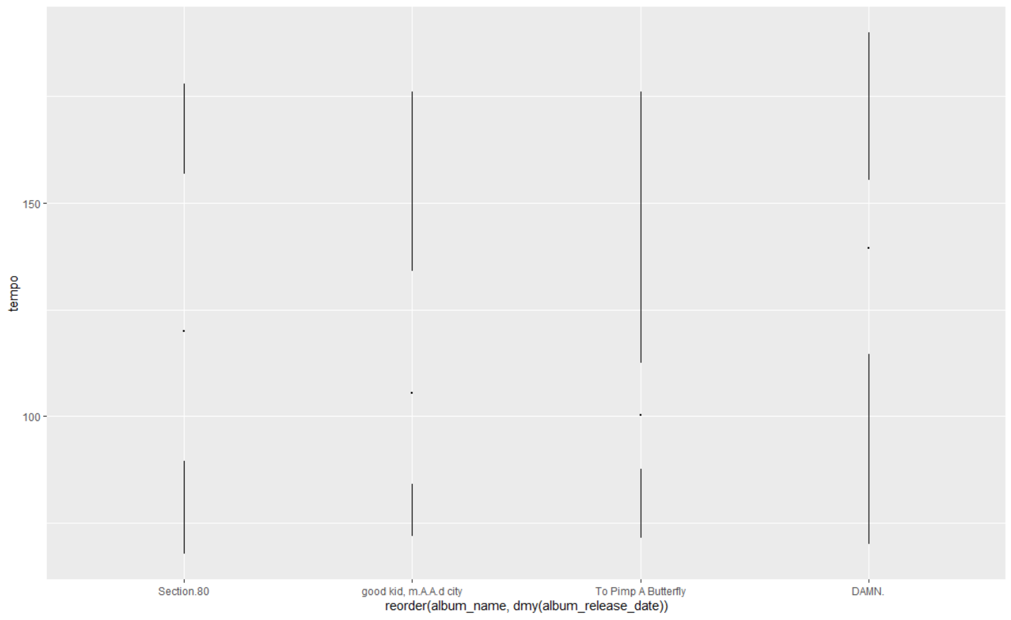
8. If you want to look at how the tempo across Drake's different albums have varied, why not compare:

```
ggplot(data = kendrick) +
  aes(x =reorder(album_name,
                 dmy(album_release_date)),
      y = tempo) +
  geom_boxplot()
```



with:

```
ggplot(data = kendrick) +
  aes(x =reorder(album_name,
                 dmy(album_release_date)),
      y = tempo) +
  geom_tufteboxplot()
```



(My instinct is that there are better ways to communicate this information than in box plots. But if you like box plots, and you like Tufte, here's a way of indulging that combination.)

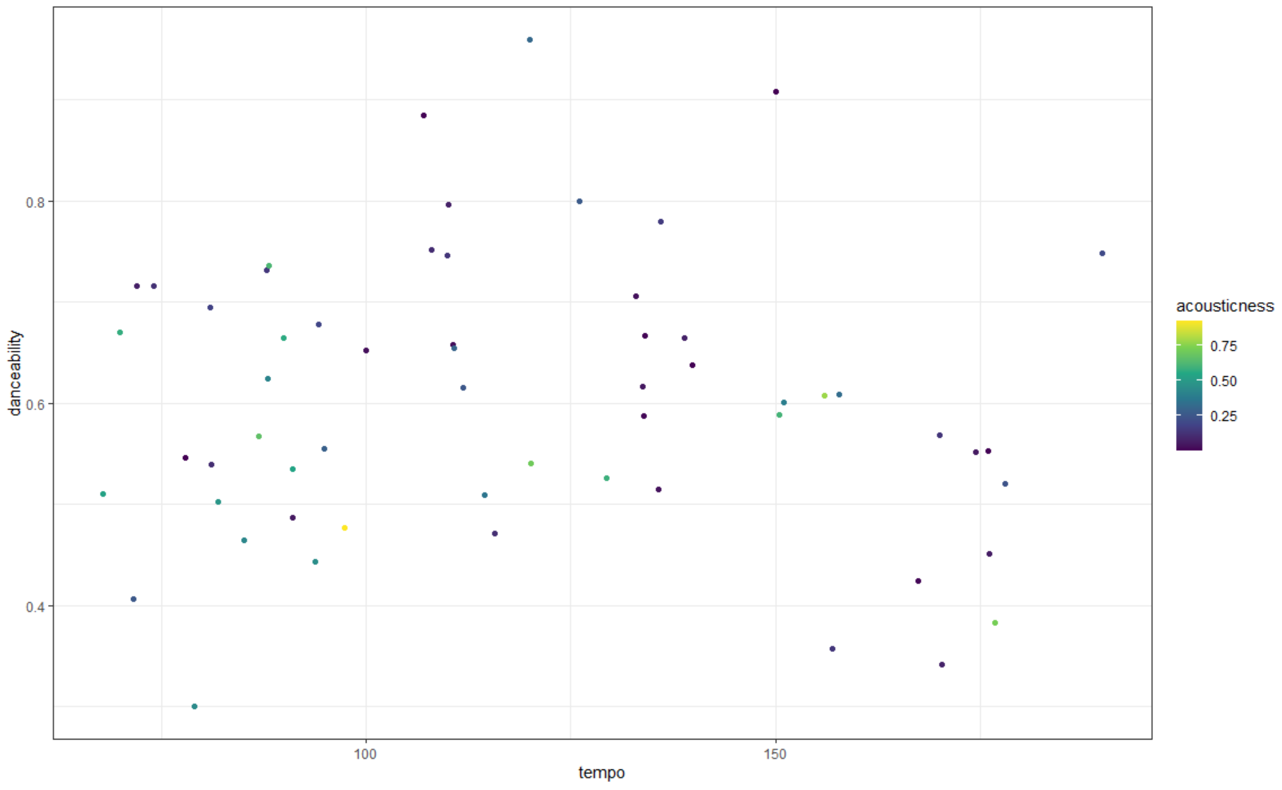
Prepackaged themes

You can change each element of a ggplot object separately, if you want. You can change the background colour, the colours of individual objects, the fonts used in the legend, and so on. Sometimes this is useful and practical: for example, if you're a journalist who's using a house style, it's easier to manually set this up using ggplot2 than it is to tweak the graph afterwards using Illustator or Photoshop. (This isn't a made-up example: you can read things John Burn-Murdoch at the FT has written about arranging his ggplot2 workflow so that ggplot graphics can go straight into the paper.)

You can also use prepackaged themes, if there are particular styles that you want to mimic. Not everyone likes the grey grid with white gridlines that ggplot2 uses by default, for example (I'm not the biggest fan.) Several themes come preloaded with ggplot2, and ggthemes adds several more; if you've found another theme that you like the look of, there's often a package available for that as well, you'll just have to install it.

9. Let's say we want to tweak our last graph so that we lose the grey grid, swapping it for something a bit more minimal.

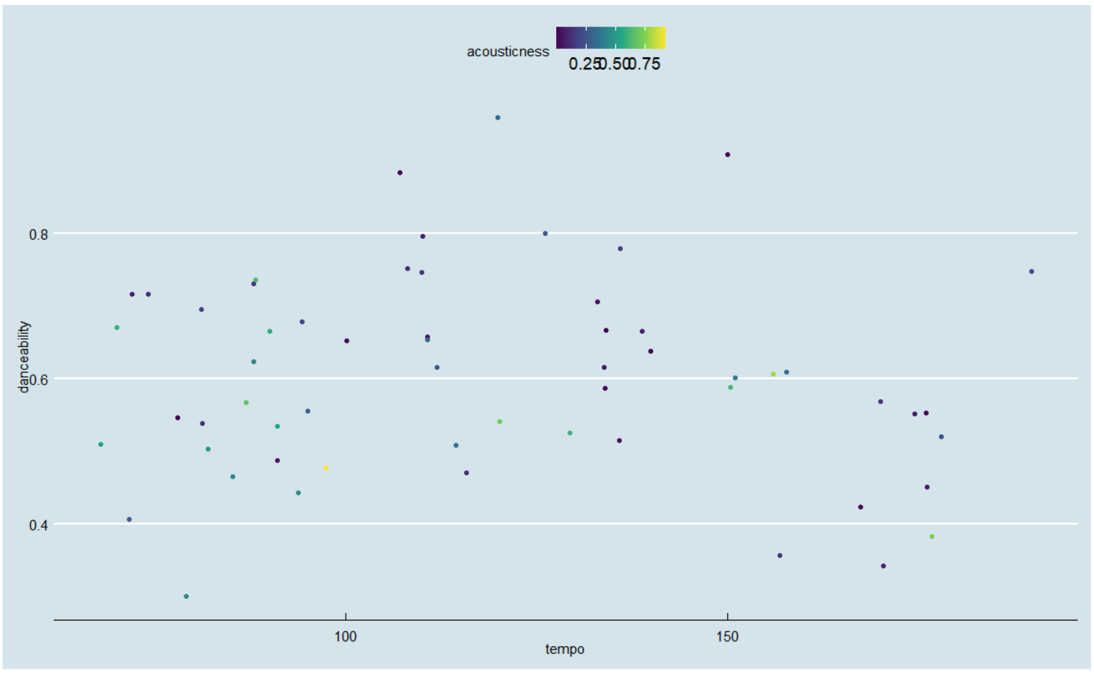
```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = acousticness) +
  geom_point() +
  scale_colour_viridis_c(option = "D") +
  theme_bw()
```



How do you feel about that?

Having done that, type it out again, but pause once you've typed as far as theme_. This will give you a sense of which other themes are available. For example, we can try the Economist one:

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = acousticness) +
  geom_point() +
  scale_colour_viridis_c(option = "D") +
  theme_economist()
```



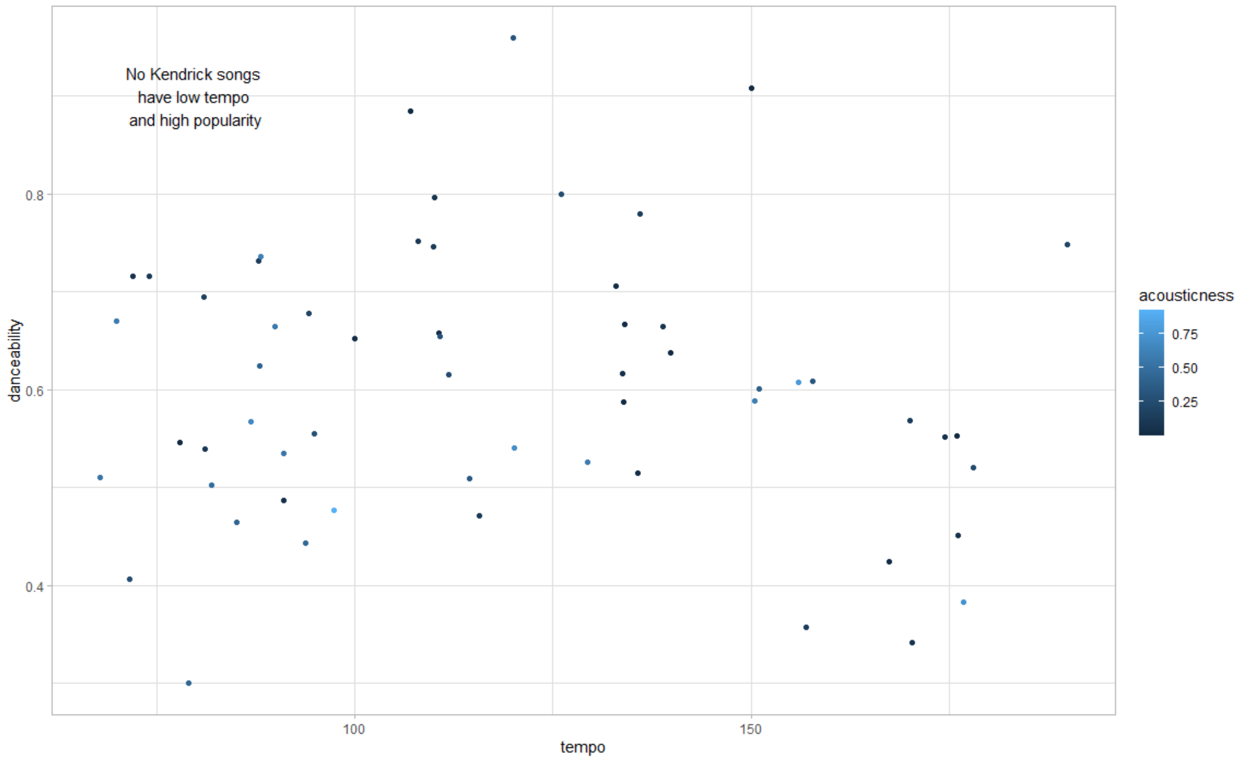
Some of them I'd advise against – the Excel theme is only really in there as a joke, for example – but see what you like and what you get on with!

Annotations

10. Our graphics have been fairly sparse so far. This isn't a problem per se – sparse graphs are often clear graphs – but you might find yourself wondering how it is that you sometimes see graphics with text on them. This is broadly pretty easy to do in ggplot, and this is how.

```
ggplot(data = kendrick) +
  aes(x = tempo,
      y = danceability,
      colour = acousticness) +
  geom_point() +
  theme_light() +
  annotate("text",
           x=80,
           y=0.9,
           label="No Kendrick songs
have low tempo
and high popularity")
```

Here, what I've done is add an annotate() in my ggplot2 code. You can see it needs four arguments: what kind of annotation it is, where it's going on each of the x and y axes, and a label. (You'll also note I've broken the label up into three lines).



You'll often want to annotate graphs if you think particular areas of them are interesting. (You might also want to annotate particular points, and you won't always want to do this manually. We won't go over this now, but if this is likely to be a focus for you, I'd recommend the **ggrepel** package.)

Exporting graphics

Once we've made graphs, then what? We don't want to take a photo on our phones, at the very least.

I've already showed you in a video that there's a little "export" button in the graphics pane.

However, you can also use ggsave command. One thing about ggsave is it starts to raise questions of what the Working Directory is. When you save graphics, you're saving graphics somewhere, and you need to know where that is. Let's find out with the getwd() command.

```
getwd()
```

This tells you what you have currently set as your Working Directory. If you are not happy with where your Working Directory is, then please go watch the 'SMI105 Worksheet 2' video to find out how to change it!

Once we are happy with where what our Working Directory has been saved as, we can both load data from this folder, like so (imagine I've saved drake.csv in this folder):

```
kendrick_again <- read_csv("kendrick.csv")
```

but I can also save the graphic I plotted most recently:

```
ggsave("kendrick_plot_01.png")
```

One thing you'll sometimes want to do is change the dimensions of the graph – maybe you want a long and thin graph, maybe you want a short and fit graph. Here's how you can change the dimensions of a graph (note that by default ggsave uses inches as its units).

```
ggsave("kendrick_plot_01.png", width = 10, height = 5)
```

And so on (You can also export to different file formats, like .jpg, and .eps; if you want to do this, just change the file extension in the command).

DATA VISUALISATION

LAB WORKSHEET 6

Creator: Dr Mark Taylor

What's Happening in this Document?

Maps!

We're also starting to load data locally, rather than from the web. In the first half of the semester, we pulled data from the web; this is fine when we've got a web host, or what we need is just a small file, but when we're working with more or bigger files this often isn't practical.

In addition to this, we're going to combine a couple of datasets together. We'll go into some more detail in the coming weeks about how to do this.

To go with our worksheet, we have .zip file that contains a few different things. Please make sure you save this file where you have set your working directory.

Health Warning

Broadly, drawing maps is more computationally intensive than drawing the other kinds of graphs that we're drawing on this module. For this reason, it generally takes longer for ggplot2 to draw the maps we're doing in this document than the graphs we have done previously, and sometimes won't draw them at all. This is particularly the case if you're on a Mac rather than on a Windows machine. If this happens, don't panic - it's normal.

Q-Step A step-change in quantitative social science skills Funded by the Nuffield Foundation, ESRC and HEFCE
Nuffield Foundation
Sheffield Methods Institute.
The University Of Sheffield.

Getting started

Let’s get started by installing a package:

```
install.packages(“sf”)
```

Now let’s load some packages:

```
library(tidyverse)
library(sf)
```

The sf package is new. sf standards for simple features, and it’s a package specifically for geographic data using R. By contrast, you’ve seen tidyverse before.

And then let’s load some data. Please make sure you have downloaded the zip files from the Q-Step Website (or from this link: https://bit.ly/lab_worksheet_6), that you have extracted the files, and that you have saved them to where you have set your working directory:

```
westminster <- read_sf(dsn = ".", layer = "westminster_const_region")
airports <- read_sf(dsn = ".", layer ="airports_gb")
housing <- read.csv("housing.csv")
h_and_w <- inner_join(westminster, housing, id = "CODE")
```

OK, there’s a few things going on here.

Loading housing is relatively straightforward; it’s a read.csv() command, with a csv name in the folder. You’ll note that it’s not a URL, like you’ve seen before; that’s because it’s in the same folder as this R project.

Loading westminster is a bit more complicated. You’ll note that there’s several different files in this folder that start “westminster_const_region”; this is because shapefiles, which are the flavours of spatial data that we’re working with, tend to consist of several different files, including geography, projections, and databases with data frames in. So there’s two elements in the bracket. The first, dsn, specifies where we’re looking; the . means we’re looking in the same folder as the files we’re using. (I’d generally recommend this.) The second, layer, asks for the name of the layer we’re working with; in this instance, that’s westminster_const_region.

Loading airports is exactly the same as loading westminster, as it’s another shapefile.

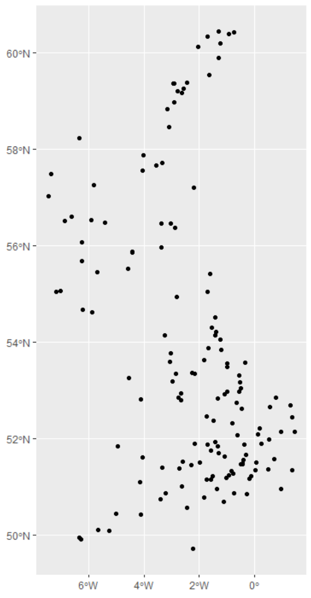
Merging

The last thing we did was to merge two files. westminster is a shapefile of parliamentary constituencies, without much information about them; housing is a data frame that contains information about how many people in each constituency own their home, without much information about what those constituencies look like. In merging the two files, we’ve now got a shapefile which includes information about how many people own their homes. (We’ll do this more systematically in our next worksheet)

Drawing maps

1. Let's draw some maps. Specifically, let's start by drawing a map of where different airports are.

```
ggplot(data = airports) +
  geom_sf()
```

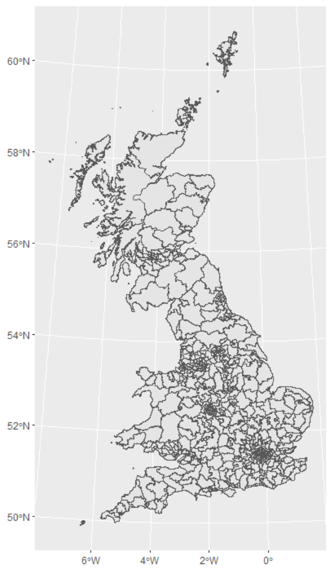


You'll note that this is the shortest ggplot we've run so far; there's no aesthetic mappings. What we have is a bunch of simple features; each observation in the airports data comes with longitude and latitude, which are effectively x and y coordinates.

While this probably looks vaguely like a map of the UK, it's not exactly what we're aiming for.

2. What we can do as an alternative is draw a map of each of Britain's parliamentary constituencies, using the westminster data, like so. (This will probably take a little while to load; don't panic if that happens. This might be a good opportunity to make a cup of tea.)

```
ggplot(data = westminster) +
  geom_sf()
```

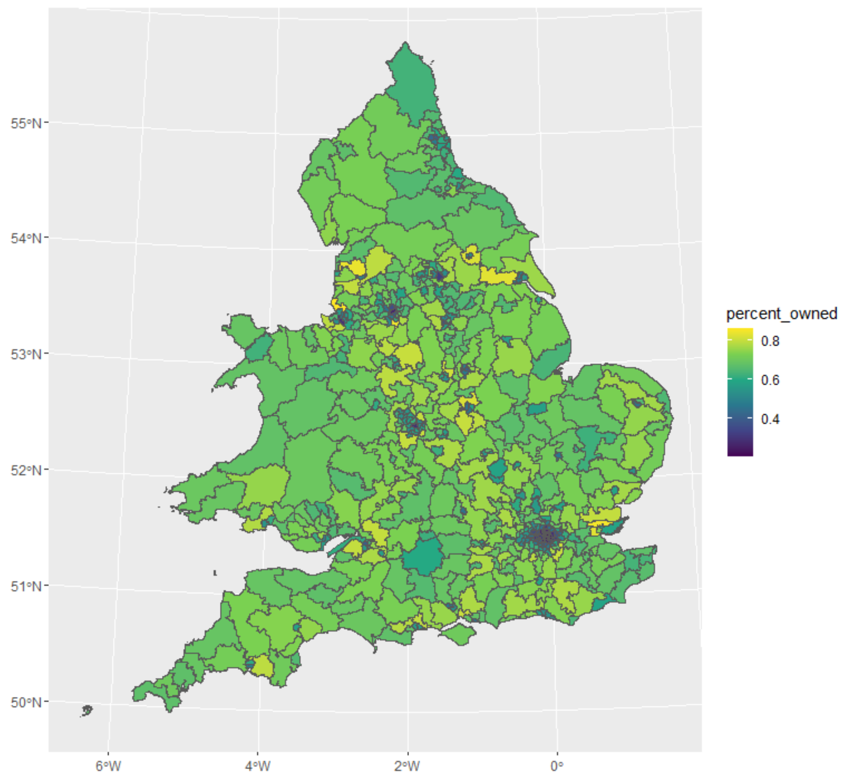


OK, this is looking familiar! Let's colour it in.

Colouring in

3. How do we add an aesthetic mapping to a map? The same way we've been adding aesthetic mappings to everything else. Let's extend what we've done here by colouring in each of the UK's parliamentary constituencies according to the numbers of people who own their homes, as opposed to renting (or living rent free, etc.)

```
ggplot(data = h_and_w)+
  aes(fill = percent_owned) +
  geom_sf()  +
  scale_fill_viridis_c()
```



This is exactly the same kind of thing we've seen before. We've specified data, an aesthetic mapping (putting
% owned in fill), a geometric object (sf), and tidied up a bit (changed the colour scheme to viridis).

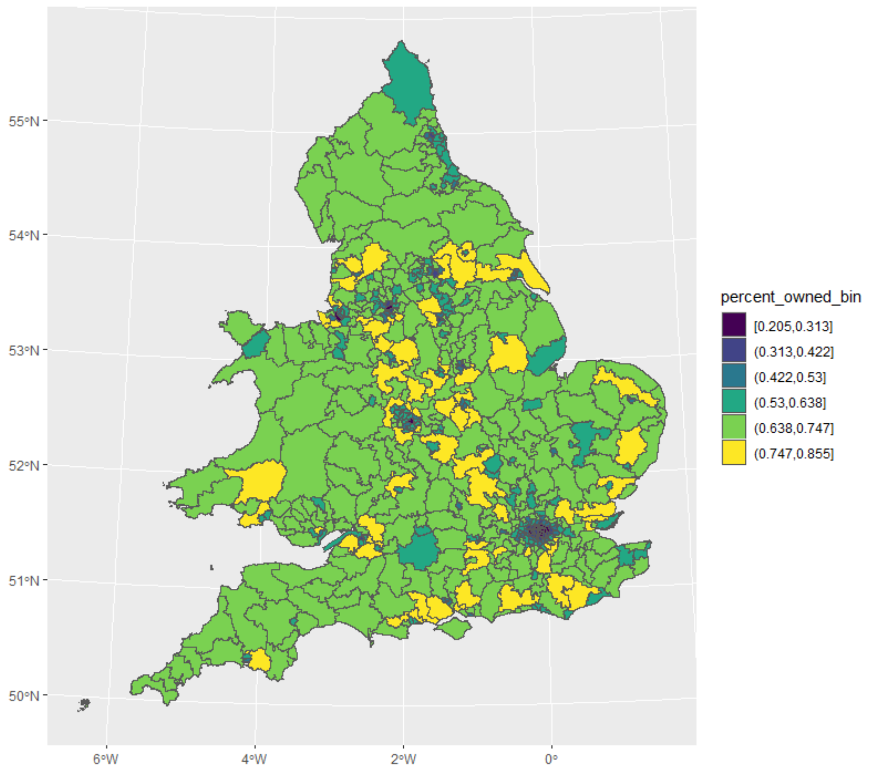
Binning colour schemes

We're currently using a continuous colour scheme. If you've got incredibly sharp eyes, you can distinguish between an area that has 61% of its residents in owned homes and an area with 62%.

But most people can't. There's also a problem here in that there's a few areas (mostly in London) with very low rates of home ownership - around a third - that distort the overall picture.

4. As an alternative, let's bin these categories. There's two approaches to binning that I want to flag; there's others, but this is a starting point.

```
h_and_w %>%
  mutate(percent_owned_bin = cut_interval(percent_owned, 6)) %>%
  ggplot() +
  geom_sf(aes(fill = percent_owned_bin)) +
  scale_fill_viridis_d()
```



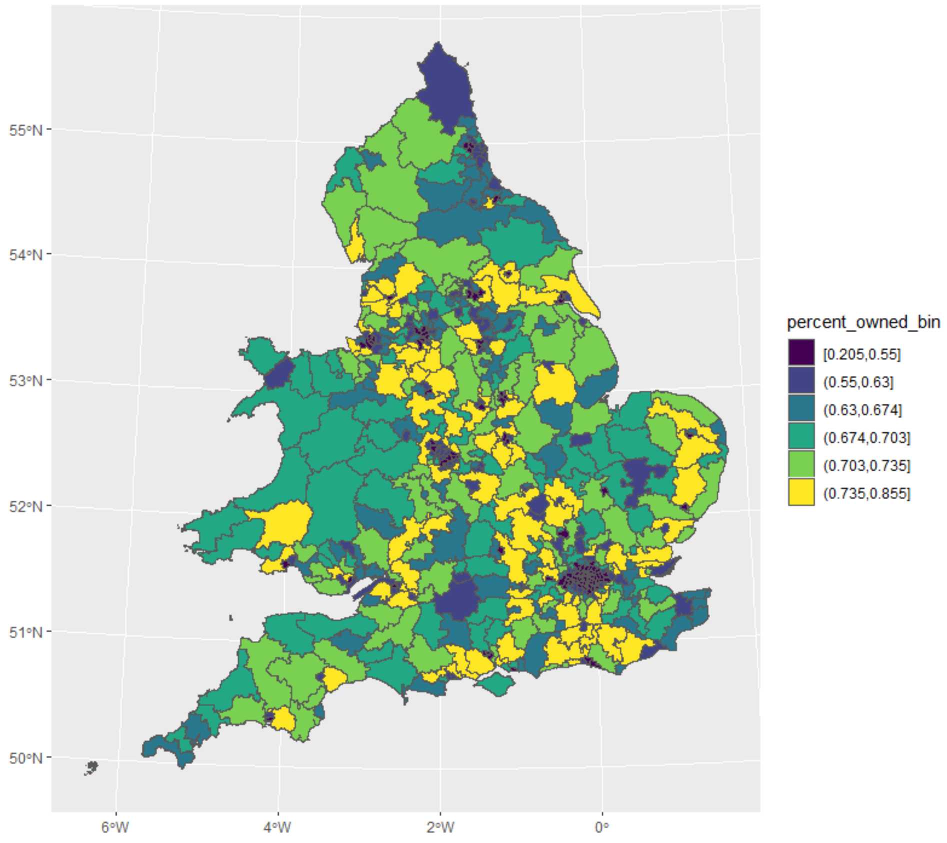
What have we done here?

You'll remember mutate() from before, as a method of generating new variables. What mutate's doing here is generating a new variable – percent_owned_bin – based on the percent_owned variable. Instead of doing maths to it, such as by dividing as we saw before, we're cutting it in such a way that we're generating a categorical variable, using the cut_interval function. Within this bracket, we're specifying which variable we want to turn into a discrete variable, and specifying the number of bins we want. Following this, we continue as before, running a ggplot with our new variable. The only difference is we're now using scale_fill_viridis_d, because this is now a discrete rather than continuous variable.

Here, you'll see we have six categories with equal ranges: 20% to 31%, 31% to 42%, 42% to 53%, and so on. But there's more constituencies in the higher categories than the lower ones; you'll see most of the map is green or yellow.

5. What if, instead, we wanted quartiles; what if we wanted to generate categories that would have equal numbers of constituencies in each? We can do this by changing cut_interval – which generates categories of equal intervals, but different numbers of observations – for cut_number, which generates categories of equal numbers of observations, but different intervals. Like so:

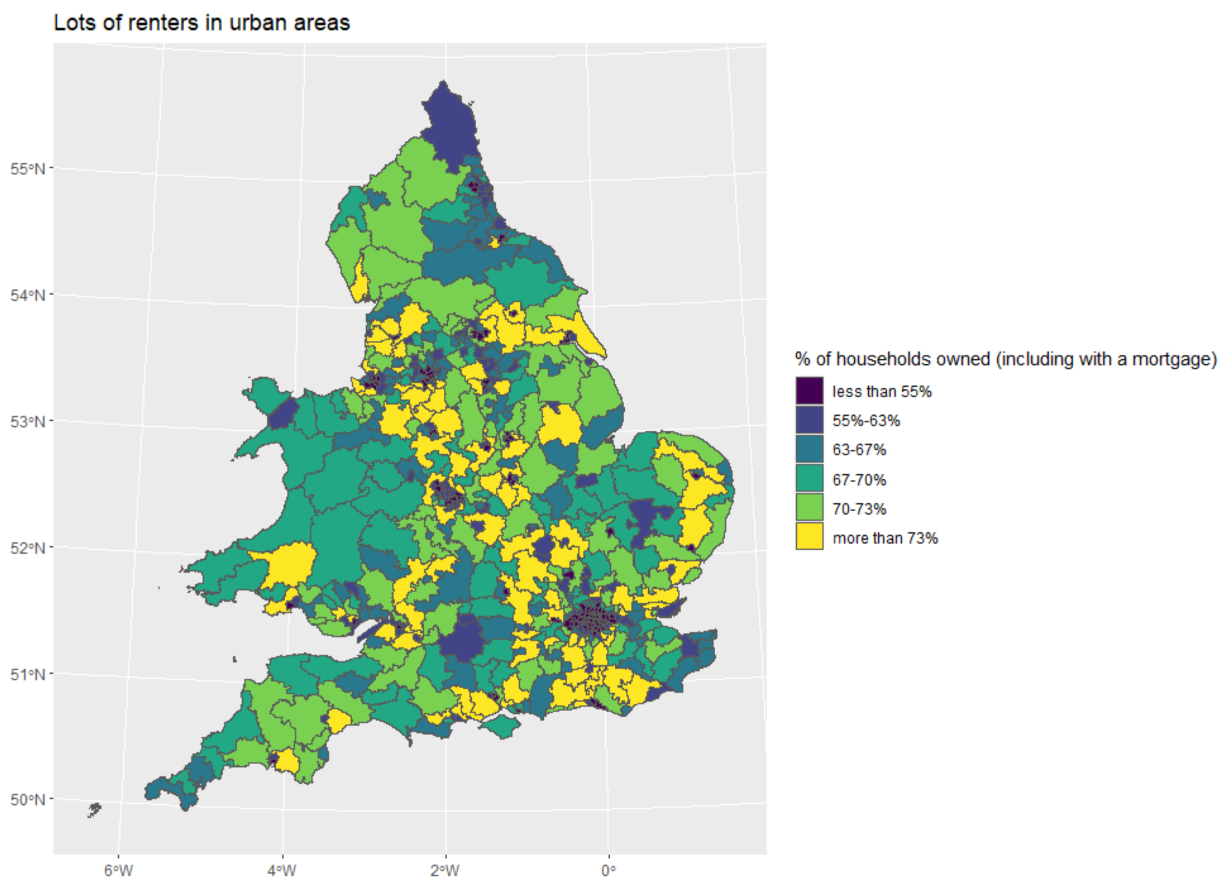
```
h_and_w %>%
  mutate(percent_owned_bin = cut_number(percent_owned, 6)) %>%
  ggplot() +
  geom_sf(aes(fill = percent_owned_bin)) +
  scale_fill_viridis_d()
```



Last couple of things

6. First, let's tidy up that last graph. The intervals look OK, but they're a bit messy. We've seen the labs() command before, which can address the title of the legend; let's also address the labels for each of the categories.

```
h_and_w %>%
  mutate(percent_owned_bin = cut_number(percent_owned, 6)) %>%
  ggplot() +
  geom_sf(aes(fill = percent_owned_bin)) +
  scale_fill_viridis_d(labels =
                           c("less than 55%",
                             "55%-63%",
                             "63-67%",
                             "67-70%",
                             "70-73%",
                             "more than 73%")) +
  labs(fill = "% of households owned
(including with a mortgage)",
       title = "Lots of renters in urban areas")
```

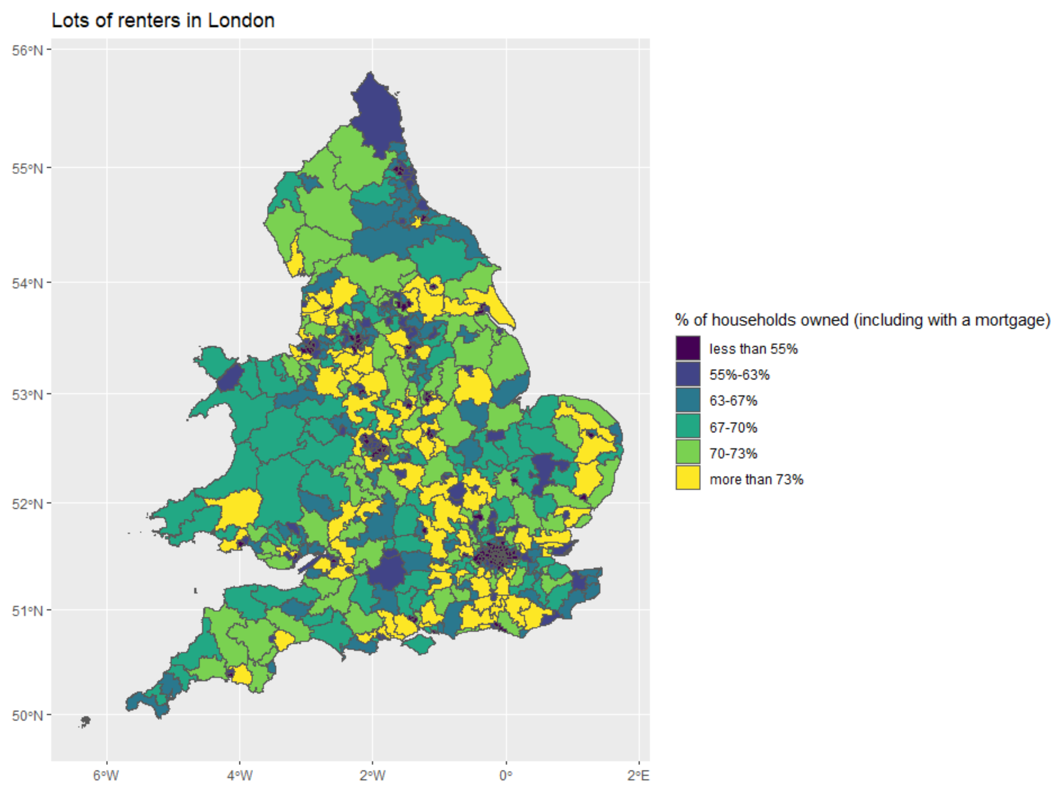


This is starting to look a bit better. (There's still an issue with respect to the level of detail – it's easier to see what's going on in rural parts of Wales than it is in densely populated urban areas, for example – but we'll come back to that.)

One last thing is coordinate systems. By definition any map on a screen involves a certain level of distortion, as we're portraying a three-dimensional object in two- dimensional space. (This gets worse as the amount of the world we're drawing gets bigger; a map of Sheffield is probably less distorted than a map of the northern hemisphere.)

7. We can tweak coordinate systems in ggplot2 and in sf like so:

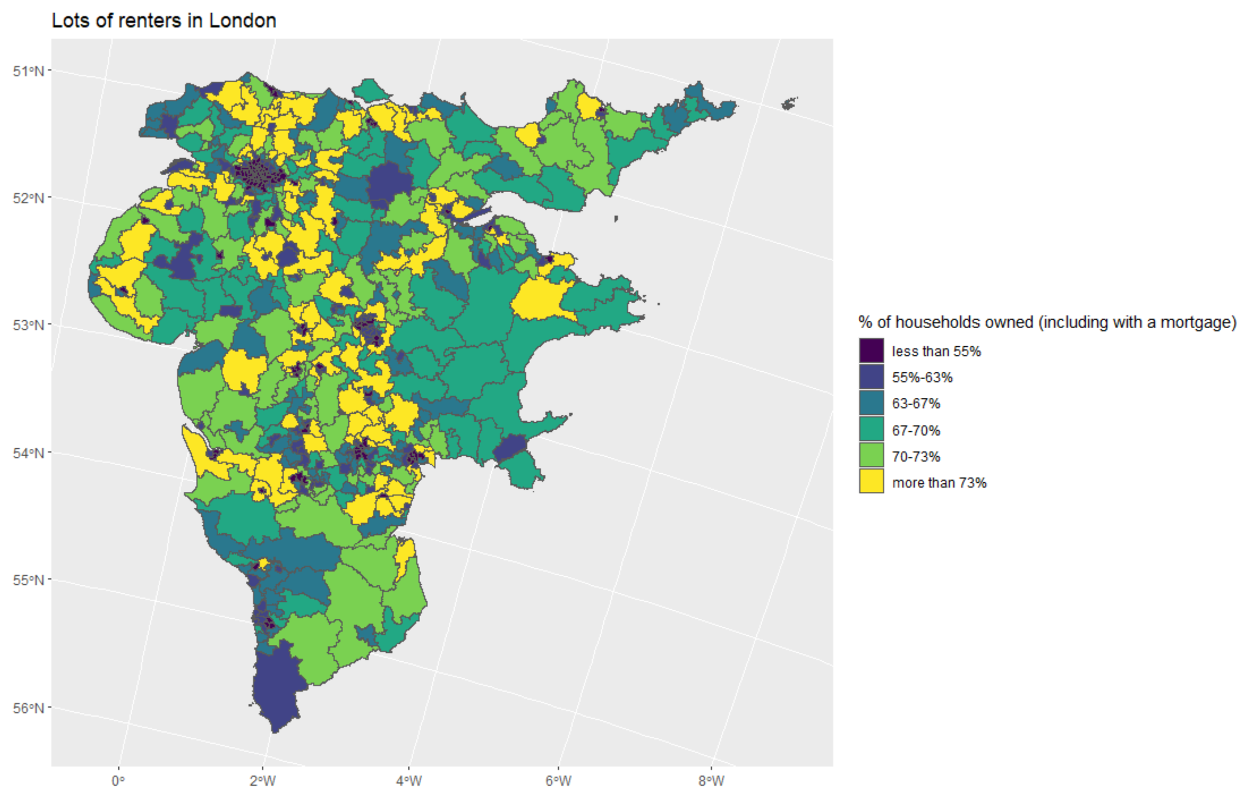
```
h_and_w %>%
  mutate(percent_owned_bin = cut_number(percent_owned, 6)) %>%
  ggplot() +
  geom_sf(aes(fill = percent_owned_bin)) +
  scale_fill_viridis_d(labels =
                         c("less than 55%",
                           "55%-63%",
                           "63-67%",
                           "67-70%",
                           "70-73%",
                           "more than 73%")) +
  labs(fill = "% of households owned
(including with a mortgage)",
       title = "Lots of renters in London") +
  coord_sf(crs = 3857)
```



What we've done here is manually specified the coordinate system with coord_sf. Inside the bracket we've specified **crs**, which stands for **coordinate reference systems**. If you look closely, you can see this map, where CRS = 3857, looks a bit different from the earlier one; it's not night and day, but there's differences (this is known as pseudo-Mercator).

8. To get a sense of why they might have more major implications, try the following:

```
h_and_w %>%
  mutate(percent_owned_bin = cut_number(percent_owned, 6)) %>%
  ggplot() +
  geom_sf(aes(fill = percent_owned_bin)) +
  scale_fill_viridis_d(labels =
                       c("less than 55%",
                         "55%-63%",
                         "63-67%",
                         "67-70%",
                         "70-73%",
                         "more than 73%")) +
  labs(fill = "% of households owned
(including with a mortgage)",
       title = "Lots of renters in London") +
  coord_sf(crs = 22275)
```



Why do you think this look weird?

Bonus Exercise

Here is what I am asking you to do:

- there's another spreadsheet that we extracted from our zip file called **degrees.csv**. This contains data on what fraction of adults in each parliamentary constituency hold degrees. Please load it into R;
- please then merge that file with the new file we created, which we called h_and_w;
- please draw a map of the fraction of adults in each parliamentary constituency with degrees, treating the DEGREES variable as continuous.

DATA VISUALISATION

LAB WORKSHEET 7

Creator: Dr Mark Taylor

What's Happening in this Document?

More maps!

Specifically, more maps where we're including data that we've downloaded from the web.

It's unusual that we're working with data that's already on our machines. So, today, what we're going to do is download some data from the web as it comes, in a format that's a tiny bit messy.

Getting set up

Let's start, as we always do, by loading some packages. We're going to continue drawing maps using the sf package, with which a handful of people ran into issues last week.

```
library(tidyverse)
library(sf)
```

Q-Step A step-change in quantitative social science skills Funded by the Nuffield Foundation, ESRC and HEFCE
Nuffield Foundation
Sheffield Methods Institute.
The University Of Sheffield.

As with our last worksheet, we're loading data from our R Working Directory. We are using some of the same file and it should already be in your Working Directory. If this has worked, you'll be able to run the following line:

```
westminster <- read_sf(dsn = ".", layer = "westminster_const_region")
```

Once we have done this, a shapefile of parliamentary constituencies should load (It's the same file as our last worksheet).

Switch to the web

Let's download some data! Specifically, let's download some data that relates to England and Wales from the 2011 Census. (The data for Scotland, and for Northern Ireland, are a bit different, for tedious reasons, which is why we're sticking with England and Wales for now.)

Please navigate as follows: - navigate to https://www.nomisweb.co.uk/census/2011 (https://www.nomisweb.co.uk/census/2011); - from the wide range of options, click on Key Statistics; - from the wide range of options, click on KS102EW, described as "Age Structure"; - the page you end up with gives you some information about the variables associated with age structure for different areas. Under Download (.csv), on the bottom left, when you're presented with a range of different types of areas, scroll down to parliamentary constituencies 2010 - hit Download.

This should download you a file called bulk.csv

Please now move your new file to your Working Directory, and rename it to age_structure.csv.

Loading the new data

Let's load the new data! If you've managed to move it successfully, the following should work:

```
age_structure <- read.csv("age_structure.csv")
```

and we can have a look at what it looks like.

```
names(age_structure)
head(age_structure)
```

OK, this is pretty usable (compared with some other data that we can download from the web, this is astonishingly well put-together and accessible). However, when we load data that I've cleaned up, we normally have column names that are brief, and easy to understand. These are clear, but quite lengthy, which can make it a bit trickier to interpret what's going on.

So, what we can do is just go through and rename the variables. What we're not doing here is adding new variables with shorter names: by renaming them, the old variable names are disappearing. We can do this with the **rename** command, like so:

```
age_structure_renamed<- age_structure %>%
rename(date = date,
          geography =geography,
          CODE =geography.code,
          rural =Rural.Urban,
          population=Age..All.usual.residents..measures..Value,
          between0and4 = Age..Age.0.to.4..measures..Value,
          between5and7 = Age..Age.5.to.7..measures..Value,
          between8and9 = Age..Age.8.to.9..measures..Value,
          between10and14 = Age..Age.10.to.14..measures..Value,
          between15 = Age..Age.15..measures..Value,
          between16and17 = Age..Age.16.to.17..measures..Value,
          between18and19 = Age..Age.18.to.19..measures..Value,
          between20and24 = Age..Age.20.to.24..measures..Value,
          between25and29 = Age..Age.25.to.29..measures..Value,
          between30and44 = Age..Age.30.to.44..measures..Value,
          between45and59 = Age..Age.45.to.59..measures..Value,
          between60and64 = Age..Age.60.to.64..measures..Value,
          between65and74 = Age..Age.65.to.74..measures..Value,
          between75and84 = Age..Age.75.to.84..measures..Value,
          between85and89 =Age..Age.85.to.89..measures..Value,
          between90andmore=Age..Age.90.and.over..measures..Value,
          mean_age = Age..Mean.Age..measures..Value,
          median_age = Age..Median.Age..measures..Value)
```

That was a lot. (You might wonder whether we needed all the variable names that I've started with **between**: in practice, it might have been overkill to rename every single variable, rather than just the variables I was going to use. But we've come this far; we might as well carry on and get the practice!)

Right, let's merge the data.

Merging the data

We did some merging in our last worksheet. Let's now look at it in a bit more detail.

```
age_with_map <-
  merge(westminster, age_structure_renamed, id = "CODE")
```

What have we got here? Let's go through the steps one-by-one.

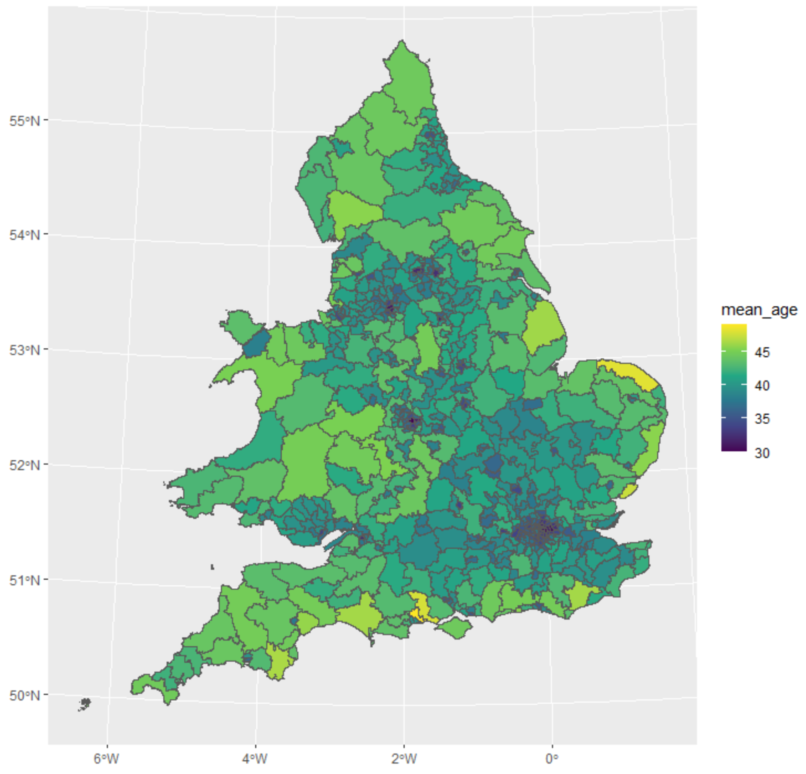
- age_with_map is the name of our new object;
- the arrow means it's going to consist of the thing on the right;
- the merge command means we're combining two datasets.
- following the merge command, we're specifying the two datasets we want to combine;
- finally, we're specifying the id variable that both datasets have. R can use this to match up observations. Crucially, this means that a dataset with a common id variable, but a flaky label variable (eg a difference between Cheshire West and Chester and Cheshire West & Chester) is a real problem if you're using the name field to join, but not a big deal if you're using the id field;
- you'll note that when we renamed our variables earlier we renamed geography_code to CODE; this was so that the common variable was called the same thing in both data frames.

Let's draw some maps

Has it worked? Let's find out.

1. Which parliamentary constituency has the highest mean age?

```
ggplot(data = age_with_map) +
  aes(fill = mean_age) +
  geom_sf() +
  scale_fill_viridis_c()
```



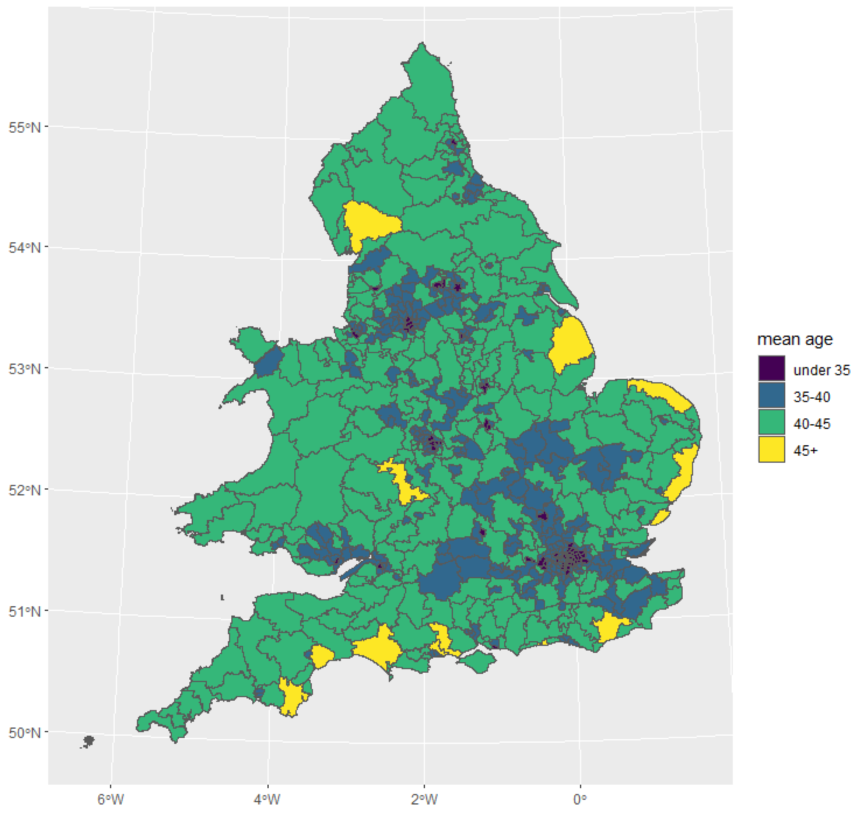
Let’s try some different bins

OK, this looks interesting. Let’s do some binning to understand the categories in more detail.

2. You’ll remember last time we tried two different approaches to binning: one where there were equal intervals, one where there were equal numbers. Let’s try specifying the bins ourselves: maybe we’re interested in particular ranges, such as areas that have mean ages of under 35.

```
ggplot(data = age_with_map) +
  aes(fill = cut(mean_age,
                 breaks = c(0, 35, 40, 45, 100),
                 labels = c("under 35",
                            "35-40",
                            "40-45",
                            "45+"))) +
  geom_sf() +
  scale_fill_viridis_d() +
  labs(fill = "mean age")
```

This looks OK. What does this show?



Bonus Exercise

Draw a map of the fraction of people in each parliamentary constituency who are 65 or older.

DATA VISUALISATION

LAB WORKSHEET 8

Creator: Dr Mark Taylor

What's Happening in this Document?

The main point of this worksheet is a quick introduction to interactives.

This is just to give you a sense of what's out there! I would also recommend getting your head round some of the several extensions to ggplot2 at https://exts.ggplot2.tidyverse.org/gallery/ in case you ever want to create some really exciting data visualisations in the future!

Getting started with plotly

Let's load the tidyverse...

```
library(tidyverse)
```

...and let's install and load plotly. It's also fairly likely that R will attempt to install several other packages in order to get plotly loaded:

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

Note: The usual resulting graphs have not been included after given code - this is on purpose.

Q-Step A step-change in quantitative social science skills Funded by the Nuffield Foundation, ESRC and HEFCE Nuffield Foundation

Sheffield Methods Institute.

The University Of Sheffield.

plotly is a tool for generating interactive graphics that isn't limited to ggplot2, and more generally isn't limited to R. What we're doing here is working within a ggplot2 framework to generate plotly objects: using R to generate interactives that can be dropped into different settings.

(In practice, the easiest thing is to integrate these interactives into .html files that you've generated using RMarkdown.)

Loading some data

Let's use some World Bank data! Let's draw a scatterplot of accountability and stability of governments across the world.
Our World Bank dataset contains data on Voice and Accountability, which captures perceptions of the extent to which a country's citizens are able to participate in selecting their government, as well as freedom of expression, freedom of association, and a free media.

```
accountability_and_voice <-
  read_csv("https://bit.ly/worldbank_av_data")

accountability_and_voice_recent <-
  accountability_and_voice %>%
  filter(year == 2017)
```

We've done two things here: first, we loaded the data in from the web (because it's quick); second, filtered it so we're just looking at 2017.

1. Let's now draw a simple scatterplot.

```
ggplot(data = accountability_and_voice_recent) +
  aes(x = accountability,
      y = stability) +
  geom_point()
```

Looks OK, save for the fact it's a bit boring (which we can fix through the standard techniques we've seen so far). How can we make it interactive?

Introducing ggplotly

2. The **ggplotly** command turns ggplot objects into plotly objects. Let's start with the simplest case, in which we'll just wrap the earlier graph in ggplotly.

```
ggplotly(
  ggplot(data = accountability_and_voice_recent) +
  aes(x = accountability,
      y = stability) +
  geom_point())
```

While this might look exactly the same as before, if you hover the mouse over different points in the **Viewer**
pane, you'll see it's now identifying the exact values of our points' x and y values: accountability, and stability.

3. However, the current presentation is a bit uninspiring: which countries are we looking at? We can tweak this by adding a **text** aesthetic. This is a bit of a bodged approach (I'm sure there's a more efficient way to do it): what we need is to combine the values of different variables for each observation with some bits of text.

```
ggplotly(
  ggplot(data = accountability_and_voice_recent) +
           aes(x = accountability,
               y = stability,
               text = paste('Country: ', country,
                            'Accountability: ', accountability,
                            'Stability: ', stability)) +
           geom_point())
```

This looks OK, but there's a couple of issues. The first is that our old text hasn't disappeared, introducing redundancy. The second is that our text is all along a single line, which probably isn't what we want.

4. Let's address both of these problems here.

```
ggplotly(
  ggplot(data = accountability_and_voice_recent) +
           aes(x = accountability,
               y = stability,
               text = paste('Country: ', country,
                            '<br> Accountability: ', accountability,
                            '<br> Stability: ', stability)) +
           geom_point(),
         tooltip = "text")
```

A couple of changes here.

The first is that our quoted segments in the text = paste() bit of aes() now include

.

means line break, so the instructive text that appears when we hover over a point now includes these line breaks.

The second is that we're no longer simply using ggplotly() to wrap the ggplot command: we're now including some code at the end, where we're specifying what the tooltip (the thing that appears when we hover over a point) should consist of. In this instance, we want it to consist of the text that we specified earlier, and nothing else.

5. One last thing to do to this graph before we move on. At the moment, the tooltip's giving us absolutely loads of decimal places, which can make it hard to interpret what's going on. We probably don't need more than two. And we can address this using the round command, like so:

```
ggplotly(
  ggplot(data = accountability_and_voice_recent) +
           aes(x = accountability,
               y = stability,
               text = paste('Country: ', country,
                            '<br> Accountability: ',
                            round(accountability,
                                  digits = 2),
                            '<br> Stability: ',
                            round(stability,
                                  digits = 2))) +
           geom_point(),
         tooltip = "text")
```

Here, we've specified which variables we want to round, and how many digits we want.

More interaction, more animation

6. So far, we've just been looking at how the relationship between variables. But how might we look at how it's changed over time? Instead of looking at the version of the data that's just from 2017, let's look at the data over time.

```
ggplotly(
  ggplot(data = accountability_and_voice) +
           aes(x = accountability,
               y = stability,
               frame = year,
               ids = country,
               text = paste('Country: ', country,
                            '<br> Accountability: ', round(accountability, digits = 2),
                            '<br> Stability: ', round(stability, digits = 2)))+
           geom_point() +
           labs(x ="Accountability",
                y = "Stability",
                title = "More accountable countries are more stable"),
         tooltip = "text")
```

A couple of changes here: - we've used the version of the data that covers the entire period - we've added a couple of extra aesthetic mappings: – frame (which determines which variable shows which observations are shown when), and – ids (which determines how points are connected between frames) - we've tidied up a bit, with titles and axis labels

In applying frame and ids, we've now got a Play button and a slider, so we can both see how this relationship has changed over time, and see what it was like at particular points.

Bonus Exercise

There's loads more you can do with plotly. If you're interested in this, spend some time Googling plotly combined with ggplot2 and try a new function that we have not covered in this worksheet.

And, indeed, spend some time with exploring other ggplot2 extensions and try some out if you like: https://exts.ggplot2.tidyverse.org/gallery/

DATA VISUALISATION

LAB WORKSHEET 9

Creator: Dr Mark Taylor

What's Happening in this Document?

We are going to go through the tiresome experience of downloading a real dataset from the web, where you might know what you want to do with it, but it's not in a format that's well-suited to what you want to do. This isn't an exhaustive set of different kinds of problems you might run into: the "R for Data Science Book" (in the Module Outline) walks you through several more. But this should give you a flavour of the amount of data cleaning you might have to do before you can start visualising data.

We'll need to do several things in order to draw the graphs we're interested in:

- First, we'll need to load the data into RStudio. This has been fairly low-stress so far; it isn't always.
- Second, we'll need to manipulate the data in order to get it into the shape we want. This involves several different steps.
- Finally, we'll need to ensure that our loaded, manipulated data looks how we want.

For this worksheet, we're going to use some recent World Bank data on the quality of democracy in different countries.

A step-change in quantitative social science skills. Funded by the Nuffield Foundation, ESRC and HEFCE
Q-Step
Nuffield Foundation
Sheffield Methods Institute.
The University Of Sheffield.

Amongst other things, the file contains information on how people in different countries feel they have voice and accountability, and how this has changed over time. This data is the original dataset where we got our data for the previous worksheet. Now, we are going to learn how to clean a dataset so that in the end we are able to draw a graph of how voice and accountability has varied in France, Italy, the UK, and USA.

The dataset is included on Blackboard, please download it and place it in your working directory.

Before we begin, please read over the worksheet (before loading it in into R) and have a look at what the dataset looks like in excel in order to give you a better idea of what we are doing.

Getting started

Let's start by loading some packages:

```
library(tidyverse)
library(readxl)
```

and attempting to load the data. We're going to use read_excel(), which is a new command; it's handy for loading Excel sheets (as opposed to csvs, and so on).

Trying to load the data

Let's try to load the data. Please make sure you have downloaded the file from the Q-Step Website (or from this link: https://bit.ly/lab_worksheet_9) and that you have saved it to where you have set your working directory:

```
world_bank_stuff <-
  read_excel("wgidataset_2020.xlsx")
```

We can see from the top right pane that this hasn't really worked. If we open the Excel file, we can see it consists of loads of different sheets, containing information about different variables. As we browse through, we can see that the one we're after is in the second sheet. Let's try loading it.

```
world_bank_stuff <-
  read_excel("wgidataset_2020.xlsx", sheet = 2)
```

While this looks better, R's given us some warning signs, and rightly so. It looks like what's happened is that there's loads of redundant stuff at the top of the file – this is metadata, which would ideally have been stored separately.

But we can't be precious about this. The World Bank have been good enough to give us this data; let's make the most of it.

Something we can do instead is to specify the range of the data we want to use. We can see the data seems to have 229 rows (when we bear in mind that the first row's been read in as variable names); we can also see the first 14 rows seem to be redundant. So we can specify the range of the data so we only include the stuff we want, and exclude the stuff we don't. We can do this like so.

```
world_bank_stuff <-
  read_excel("wgidataset_2020.xlsx",
             sheet = 2,
             range = cell_rows(15:229))
```

Getting there. But this still isn't quite what we want. There's lots of missing data. This shouldn't come as a surprise, as plenty of countries make it difficult to find out about their residents. However, ideally these cells would have just been blank; instead, they're specified with #N/A. So we need to tell R explicitly that that corresponds to missingness. Like so:

```
world_bank_stuff <-
  read_excel("wgidataset_2020.xlsx",
             sheet = 2,
             range = cell_rows(15:229),
             na = "#N/A")
```

Right. We've loaded the data. This is usually more painless than this…

Manipulating the data

So, what do our variables look like?

- We've got a variable for country name. This isn't as clean as it could be; it's annoying to have slashes in our variable names.
- We've got a variable for country code. This is a bit easier to deal with.
- We've then got, for each year since 1996 (it used to be measured every two years), a measure of an estimate, standard error, number of data sources, percentile rank, lower bound, and upper bound, on the variable in question.

The first one's a bit annoying; the last one's a real issue. This is what's referred to as wide data. In the tidyverse, we want every row to be a case, and every column to be a variable; this isn't the case here, where each row contains a lot of different cases (at different time points), and several columns consist of the same variables, just at different time points.

Let's start by just keeping the country names, country codes, and estimates for each year. It's very fiddly to keep everything; let's get the data down to a manageable size.

```
world_bank_stuff %>%
  select(`Country/Territory`, Code, starts_with("Estimate"))
```

This is a start. However, it's not ideal; we'll never remember what each of these estimates corresponds to, which is the year in which the data were collected. So, following what we did in our last worksheet, let's do loads of renaming variables. Please stick this on the end of the previous command:

```
%>%
  rename(country = `Country/Territory`,
         `1996` = Estimate...3,
         `1998` = Estimate...9,
         `2000` = Estimate...15,
         `2002` = Estimate...21,
         `2003` = Estimate...27,
         `2004` = Estimate...33,
         `2005` = Estimate...39,
         `2006` = Estimate...45,
         `2007` = Estimate...51,
         `2008` = Estimate...57,
         `2009` = Estimate...63,
         `2010` = Estimate...69,
         `2011` = Estimate...75,
         `2012` = Estimate...81,
         `2013` = Estimate...87,
         `2014` = Estimate...93,
         `2015` = Estimate...99,
         `2016` = Estimate...105,
         `2017` = Estimate...111,
         `2018` = Estimate...117,
         `2019` = Estimate...123)
```

While this is a bit painful, we're at least getting somewhere. But what we really want is a long dataset, where we have an observation for the relevant value (each case of estimate) for each year for each country, not a single row for each country.

We can address this using the gather command. This reorganises data in the way that we want, converting it from wide to long. (We can use spread to convert from long to wide; we might do this if we had a variable that explained which variable each value corresponded to. This is more common than you might think.)

To do this, we need to tell R:

What variables we're gathering together. In this case, we want all our year variables to end up a single variable, paired up with something that tells us what year we're looking at

What we want to call that new variable full of numbers

What we want to call the variable where the values are the original column names

Let's call them accountability (which is the thing being measured) and year. Like so: please stick this on the end of what we've already got.

```
    %>%
      gather(`1996`:`2019`,
             key = "year",
             value = "accountability")
```

Has this worked OK?

Assuming it has, let's...

Draw a graph

Please stick the following on the end of what we've already got. Having spent a bunch of time putting the thing together, it looks like we can finally draw a fairly simple graph; let's draw a graph of how perceived accountability in the US has changed since 1996. Please stick this on the end of what we've already got.

```
    %>%
      filter(country == "United States")  %>%
      ggplot() +
      aes(x = year, y = accountability) +
      geom_line()
```

This hasn't worked brilliantly. The problem is, despite the year variable consisting of numbers, R's treating it as categorical; that's because these values all used to be variable names. We can address this by tweaking the x variable, which is currently just year, into as.numeric(year), which forces R to treat is as a number. So please tweak the last change you made so it looks like the following:

```
      ggplot() +
      aes(x = as.numeric(year), y = accountability) +
      geom_line()
```

Finally, let's tweak the graph so that we can do what we originally set out to do. Let's compare a few countries. Let's change our old filter() command to the following:

```
    filter(country == "United States" |
               country == "United Kingdom" |
               country == "France" |
               country == "Italy")
```

and add some faceting:

```
    +
      facet_wrap(~ country)
```

We’ve now drawn the graph we originally set out to draw. While this process can be pretty tortured, this is what it’s actually like for real; we spend a lot of time getting our data into the format we want.

Going further

Were we right to discard loads of data earlier? It seems like there’s a lot of uncertainty associated with some of these measures, and that this is particularly an issue for parts of the world where accountability is poorly-measured (for example, there’s only one indicator, rather than several). So maybe we want to include those upper and lower bounds, as well as overall rank estimate. How can we do that? Can we gather in such a way that we’re generating three new variables, rather than just one?

One way to do it is to generate three new objects, rather than just one, and merge them together. As we’re making them out of a single original dataset, we can be confident that the names used are consistent.

So, let’s do that. Let’s repeat what we did before, but for three different variables rather than just one, and see what happens:

First, lower bounds:

```
lower <-
  world_bank_stuff %>%
  select(`Country/Territory`, Code, starts_with("Lower")) %>%
  rename(country = `Country/Territory`,
         `1996` = Lower...7,
         `1998` = Lower...13,
         `2000` = Lower...19,
         `2002` = Lower...25,
         `2003` = Lower...31,
         `2004` = Lower...37,
         `2005` = Lower...43,
         `2006` = Lower...49,
         `2007` = Lower...55,
         `2008` = Lower...61,
         `2009` = Lower...67,
         `2010` = Lower...73,
         `2011` = Lower...79,
         `2012` = Lower...85,
         `2013` = Lower...91,
         `2014` = Lower...97,
         `2015` = Lower...103,
         `2016` = Lower...109,
         `2017` = Lower...115,
         `2018` = Lower...121,
         `2019` = Lower...127 ) %>%
  gather(`1996`:`2019`,
         key = "year",
         value = "Lower")
```

Next, upper bounds,

```
upper <-
  world_bank_stuff %>%
  select(`Country/Territory`, Code, starts_with("Upper")) %>%
  rename(country = `Country/Territory`,
         `1996` = Upper...8,
         `1998` = Upper...14,
         `2000` = Upper...20,
         `2002` = Upper...26,
         `2003` = Upper...32,
         `2004` = Upper...38,
         `2005` = Upper...44,
         `2006` = Upper...50,
         `2007` = Upper...56,
         `2008` = Upper...62,
         `2009` = Upper...68,
         `2010` = Upper...74,
         `2011` = Upper...80,
         `2012` = Upper...86,
         `2013` = Upper...92,
         `2014` = Upper...98,
         `2015` = Upper...104,
         `2016` = Upper...110,
         `2017` = Upper...116,
         `2018` = Upper...122,
         `2019` = Upper...128) %>%
  gather(`1996`:`2019`,
         key = "year",
         value = "Upper")
```

Finally, rank estimates.

```
rank_estimate <-
  world_bank_stuff %>%
  select(`Country/Territory`, Code, starts_with("Rank")) %>%
  rename(country = `Country/Territory`,
`1996` = Rank...6,
         `1998` = Rank...12,
         `2000` = Rank...18,
         `2002` = Rank...24,
         `2003` = Rank...30,
         `2004` = Rank...36,
         `2005` = Rank...42,
         `2006` = Rank...48,
         `2007` = Rank...54,
         `2008` = Rank...60,
         `2009` = Rank...66,
         `2010` = Rank...72,
         `2011` = Rank...78,
         `2012` = Rank...84,
         `2013` = Rank...90,
         `2014` = Rank...96,
         `2015` = Rank...102,
         `2016` = Rank...108,
         `2017` = Rank...114,
         `2018` = Rank...120,
         `2019` = Rank...126) %>%
  gather(`1996`:`2019`,
         key = "year",
         value = "Rank estimate")
```

Then we can merge lower and rank:

```
lower_and_rank <-
  merge(lower,
        rank_estimate,
        id = c("country", "Code"))
```

And then we can merge all three:

```
all_three <-
  merge(lower_and_rank,
        upper,
        id = c("country", "Code"))
```

and, finally, gather once again.

```
for_final_plot <-
  all_three %>%
  gather(Lower, `Rank estimate`, Upper, key = "measure", value = "combined")
```

Finally we're in a position to start drawing a graph. This time, we can include the upper and lower bounds for each country, to see the uncertainty about how these figures seem to have changed over time. We'll need to revisit several different techniques we've seen in our previous worksheets: piping, filtering, plotting, using as.numeric, tweaking factor levels, faceting, using viridis, tidying up labels, and using custom themes.

But, in practice, this is how things work.

```
for_final_plot %>%
  filter(country == "United States" |
           country == "Italy" |
           country == "France" |
           country == "United Kingdom") %>%
  ggplot() +
  aes(x = as.numeric(year),
      y = combined,
      colour =
        factor(measure,
               levels = c("Lower", "Rank estimate", "Upper"))) +
  geom_line()+
  facet_wrap(~ country) +
  scale_colour_viridis_d() +
  labs(colour = "Uncertainty",
       x = "Year",
       y = "Voice and accountability percentile") +
  theme_bw()
```

Bonus Exercise

- Find a dataset online on any topic of your choice, and identify a graph you'd ideally like to be able to draw using the data that it contains.
- Have a look at the names of the potential variables - can you draw the graph you were hoping to straight away? If not, can you get it into a format that would allow you to do so?