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)
```

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

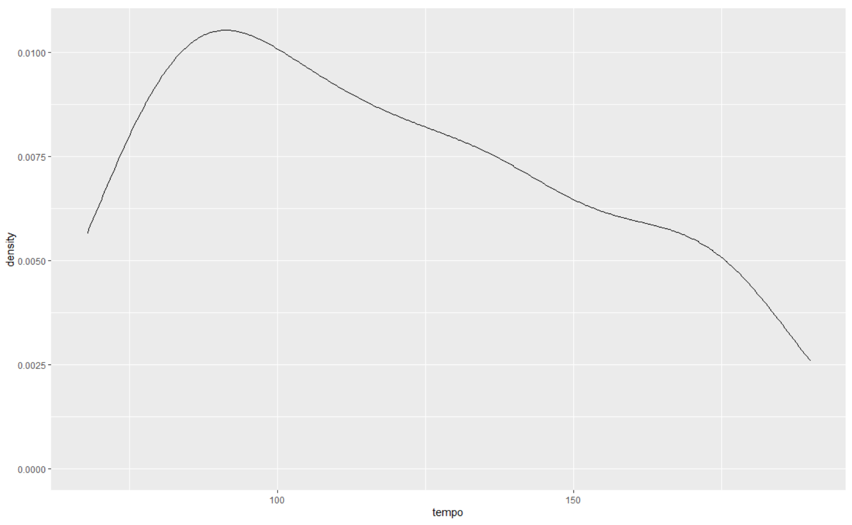
```
kendrick <- read_csv("https://bit.ly/kendrickdata")
kanye <- read_csv("https://bit.ly/kanyedata")
rihanna <- read_csv("https://bit.ly/rihannadata")
beyonce <- read_csv("https://bit.ly/beyonce_data")
queen <- read_csv("https://bit.ly/queen_data")
maccas <- read.csv("https://bit.ly/mcdonalds_data")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

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

Looking at distributions

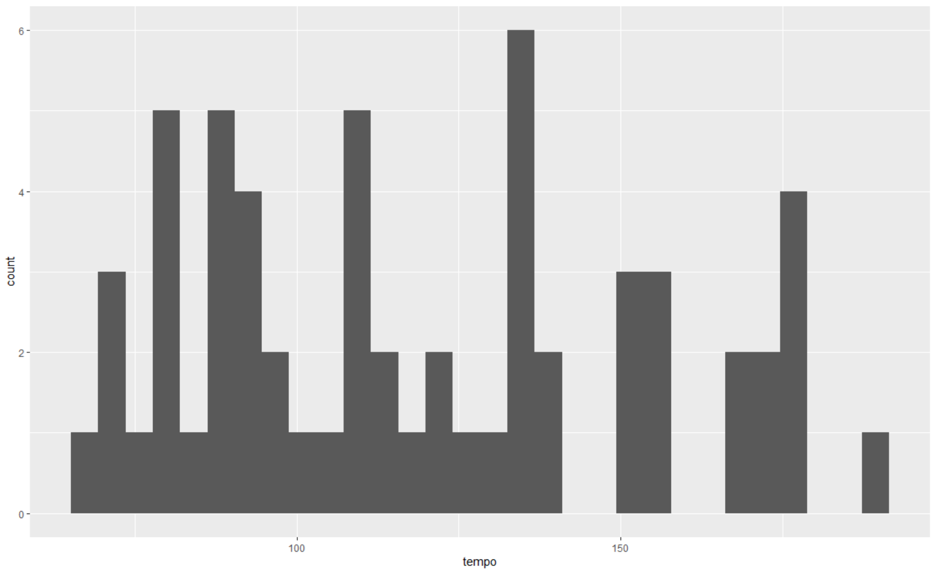
We've spent the last few weeks with the Kendrick Lamar data. Let's continue this by looking at the distribution of tempo of his tracks:

```
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.

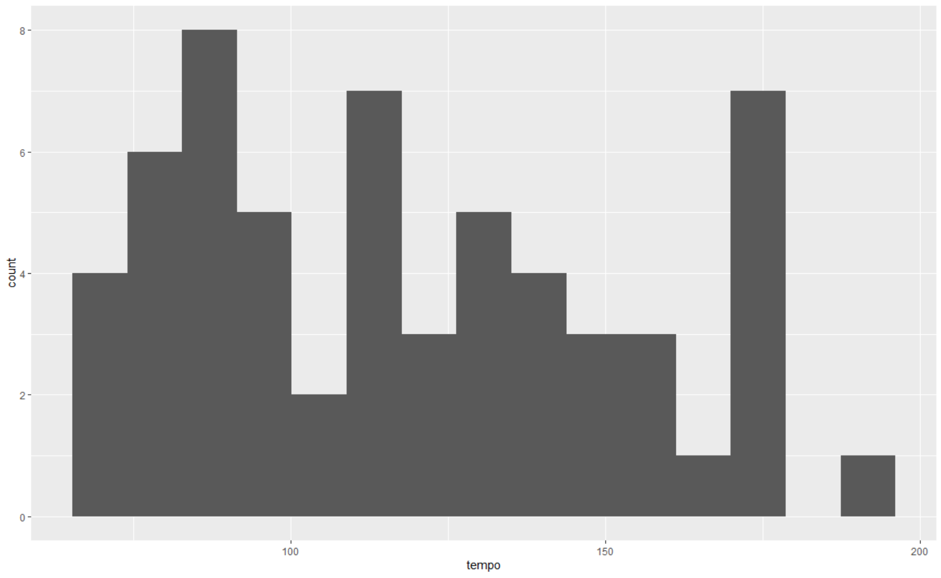
```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_histogram()
```

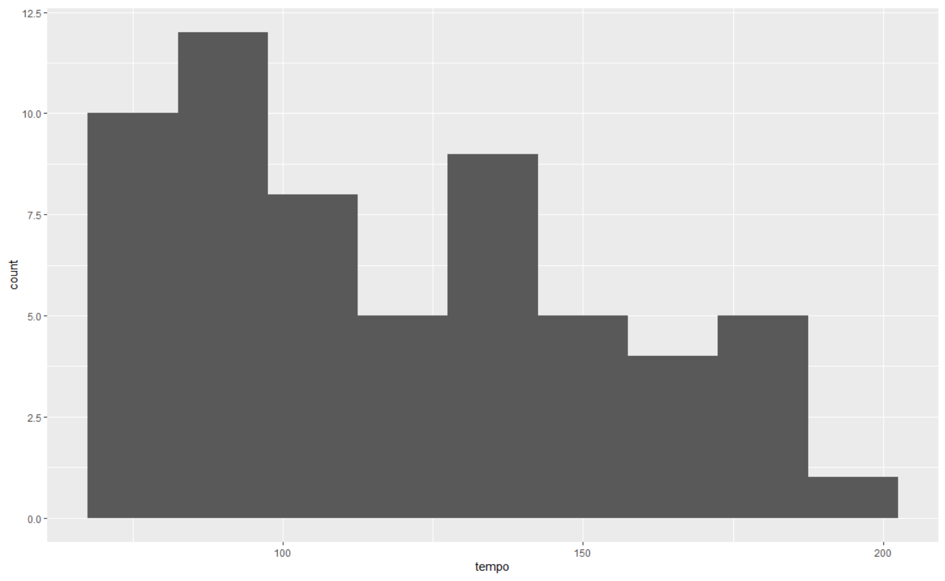


Once we run this, we end up with some red text in the console. This is fine, but it might invite us to think about bins (effectively, the bars that make up the histogram) – do we want them to be wider or narrower? Do we want more or fewer of them? Let's play around with bins and see what happens.

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

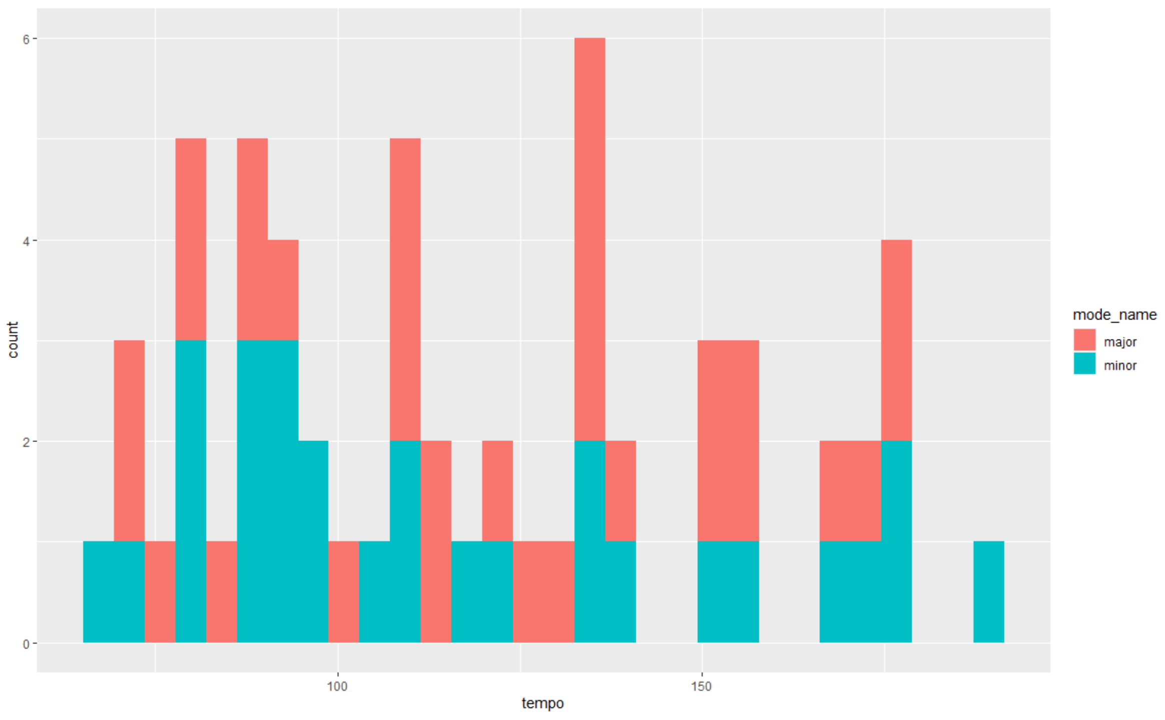
```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_histogram(bins = 15)

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?

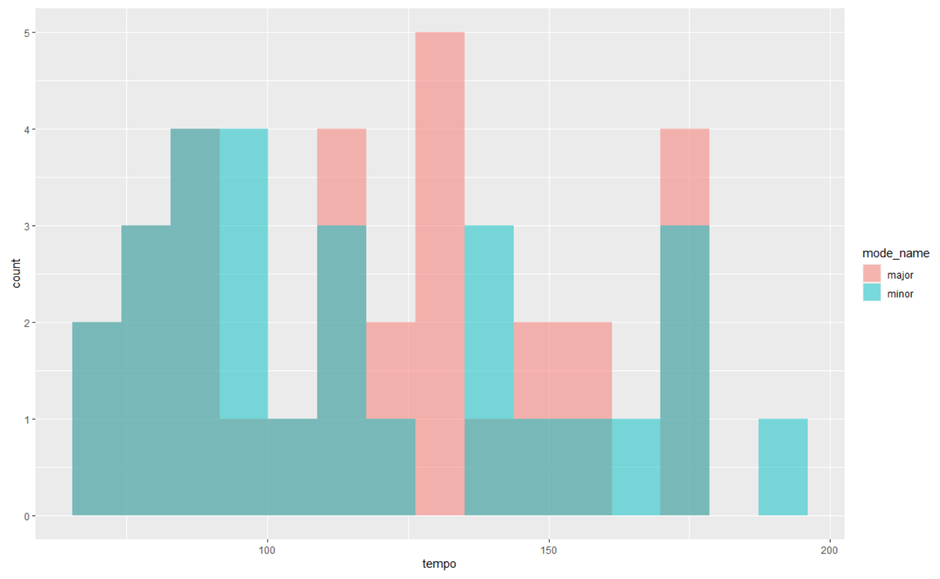
```r
ggplot(data = kendrick) +
  aes(x = tempo, fill = mode_name) +
  geom_histogram()
```



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)
```

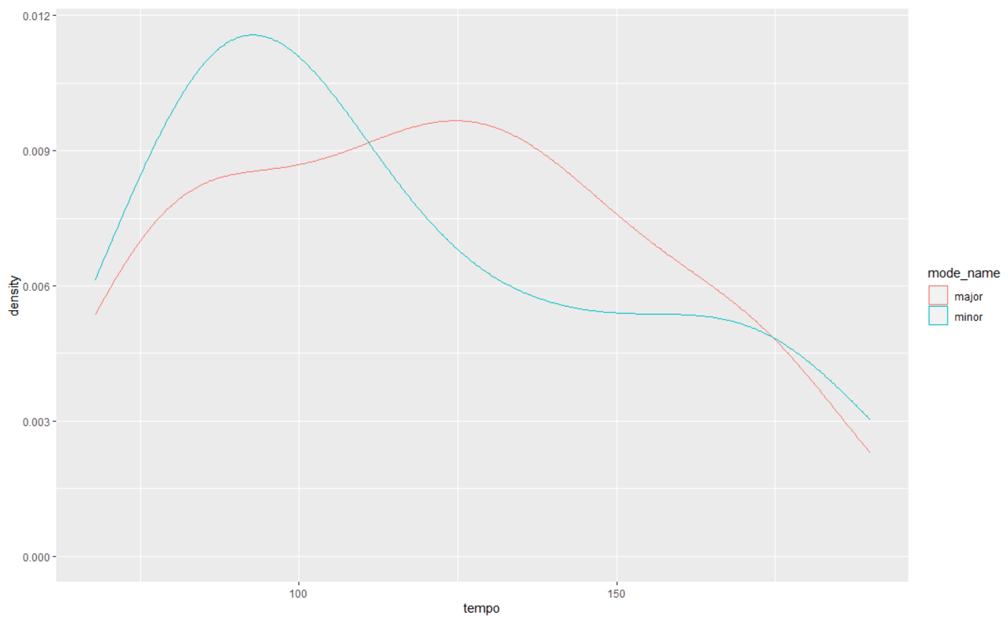


This helps, but has its own issues. Let's try some alternatives.

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.

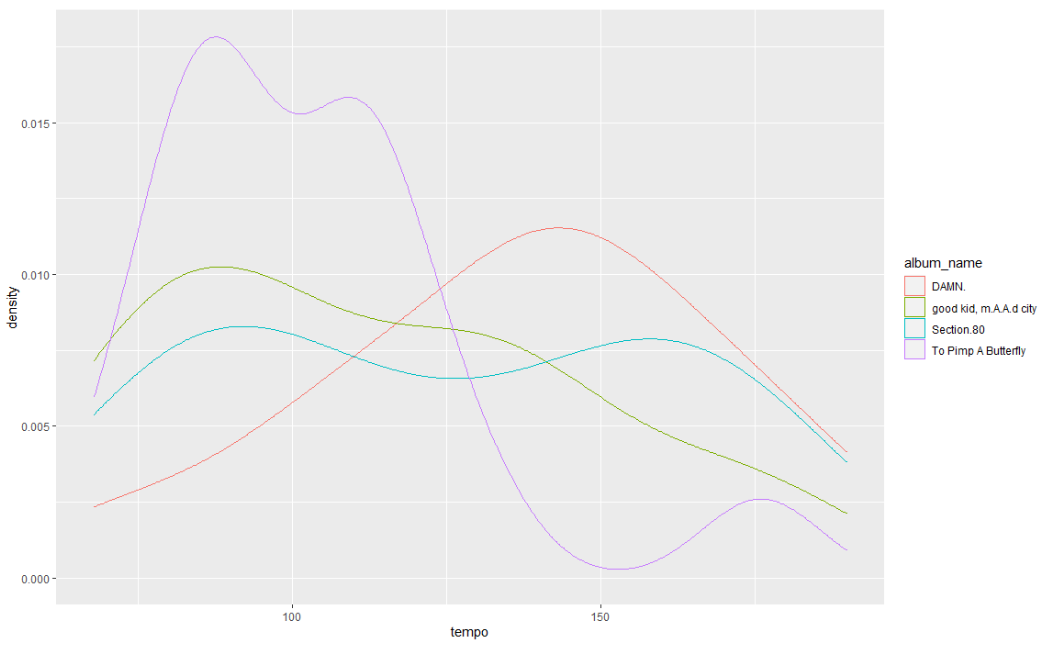
```
ggplot(data = kendrick) +
  aes(x = tempo, colour = mode_name) +
  geom_density()
```



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?

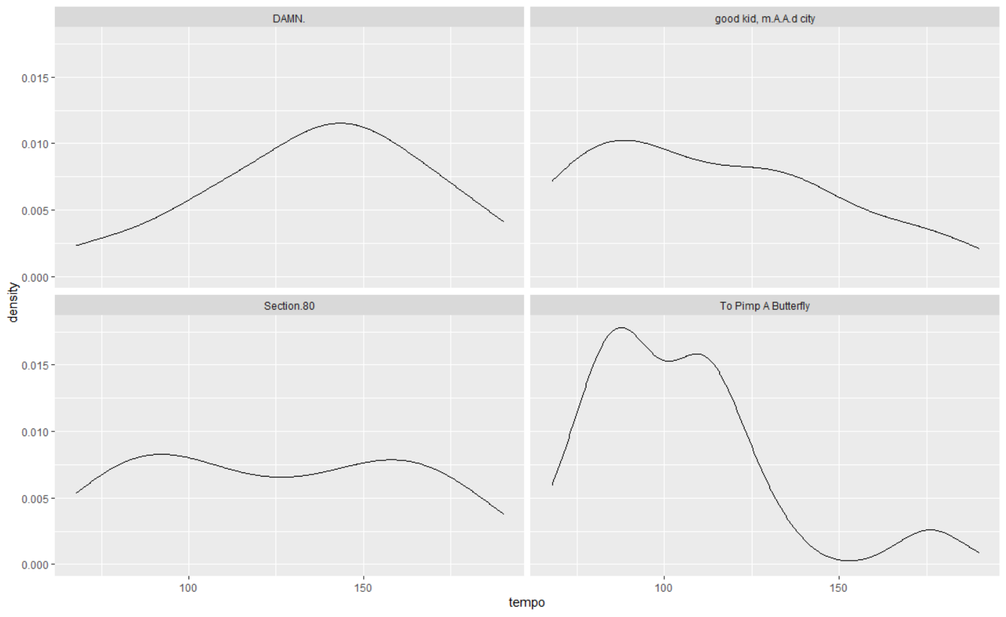
```
ggplot(data = kendrick) +
  aes(x = tempo, colour = album_name) +
  geom_density()
```



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.

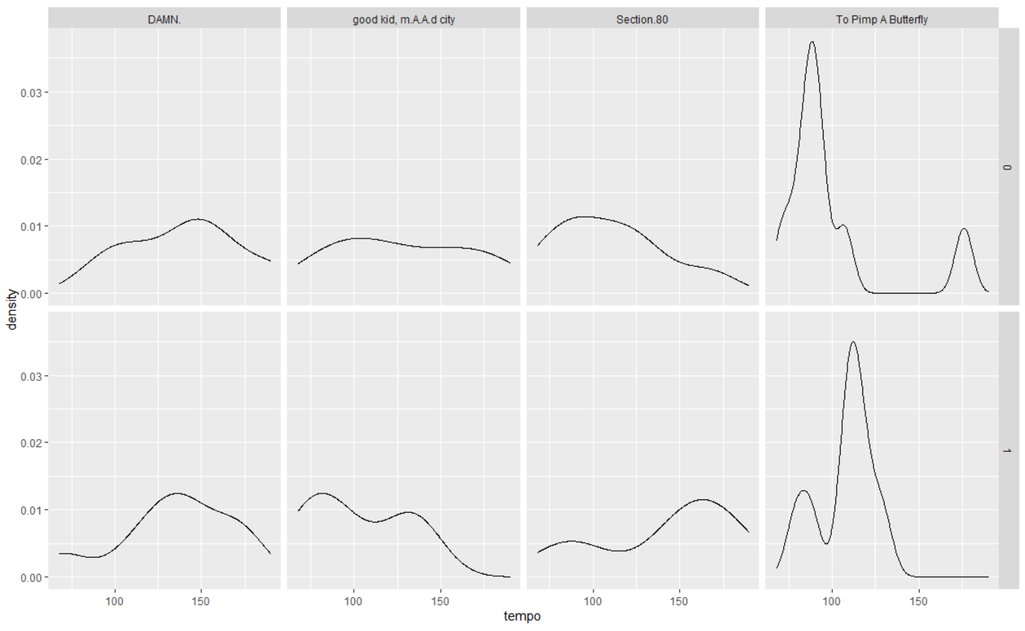
```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_density() +
  facet_wrap(~ album_name)
```



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().

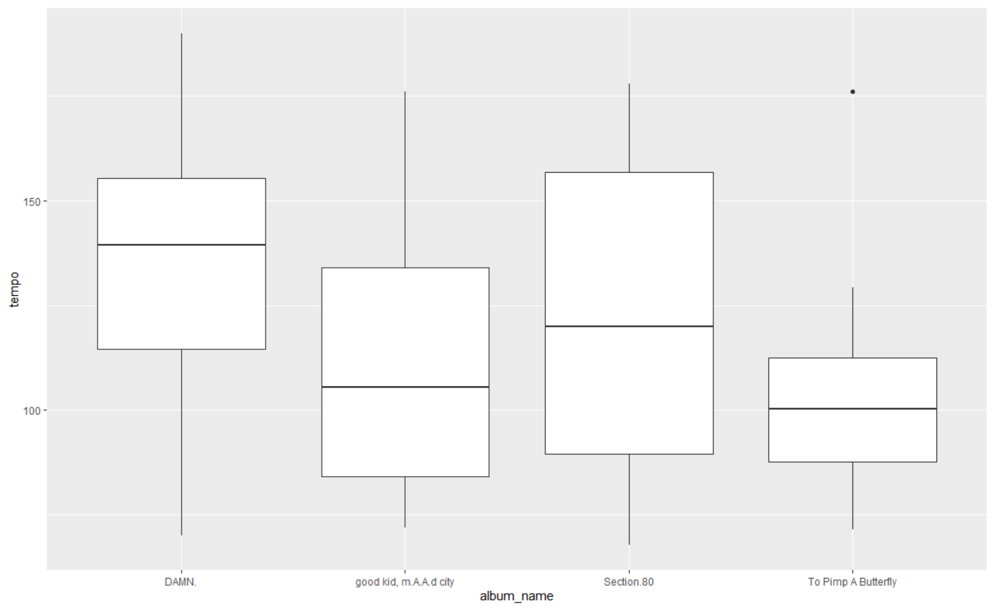
```
ggplot(data = kendrick) +
  aes(x = tempo) +
  geom_density() +
  facet_grid(mode ~ album_name)
```

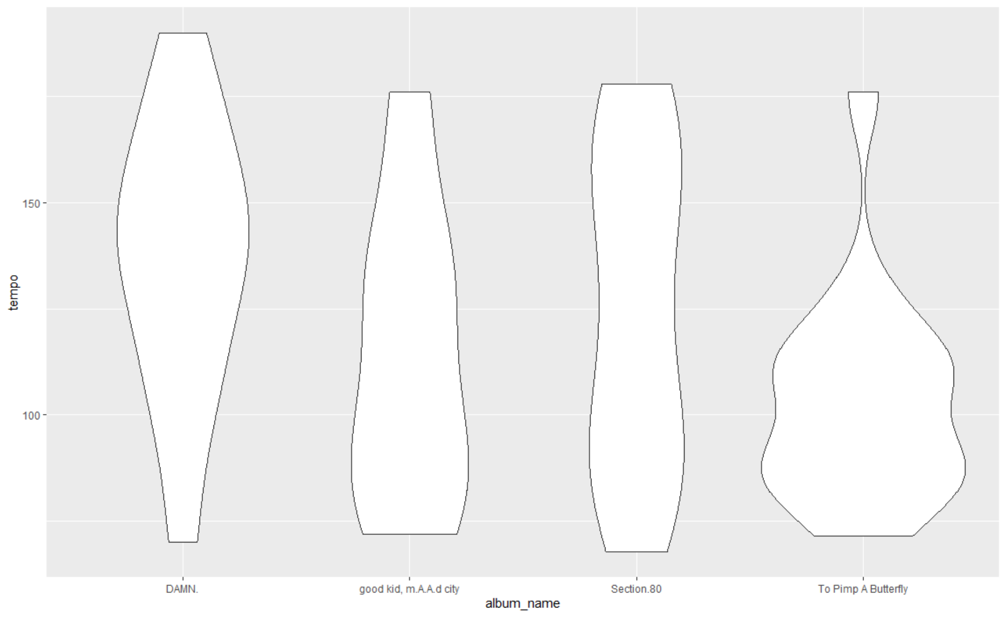


OK, we're making progress!

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()
```

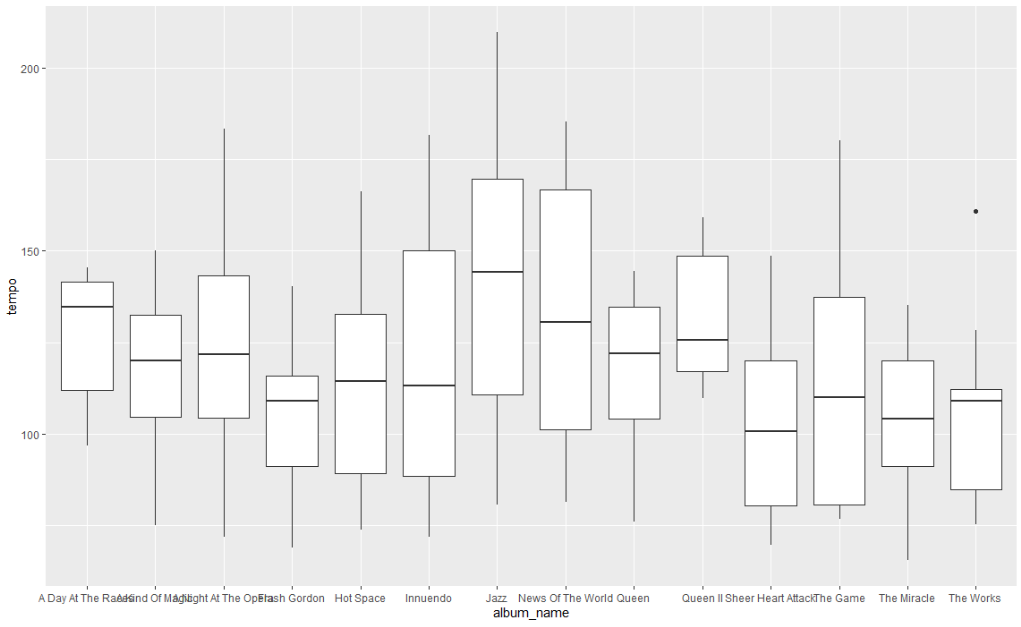




Moving to Queen

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.)

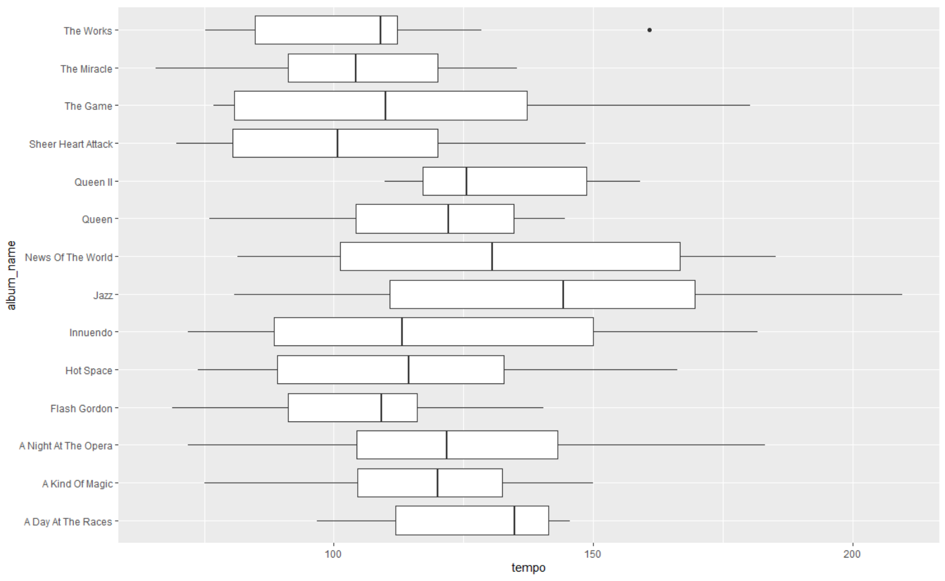
```
ggplot(data = queen) +
  aes(album_name, tempo) +
  geom_boxplot()
```



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.

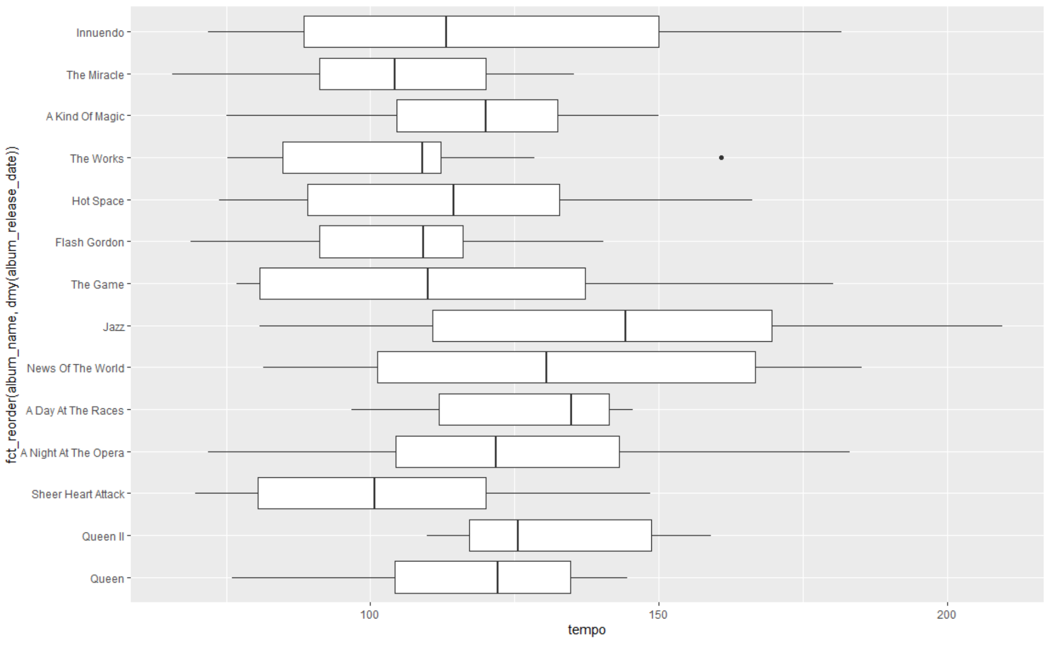
```
ggplot(data = queen) +
  aes(album_name, tempo) +
  geom_boxplot() +
  coord_flip()
```



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()
```



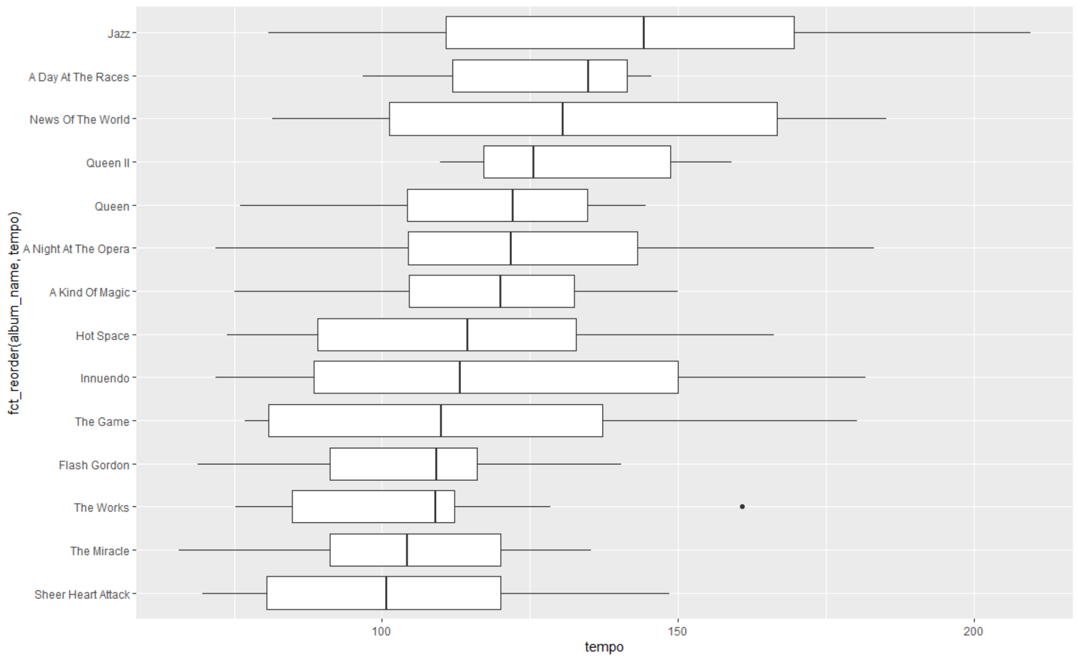
What's happened here? Everything's the same except the first mapping in the aes() parenthesis, so let's look through that.

- 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()
```

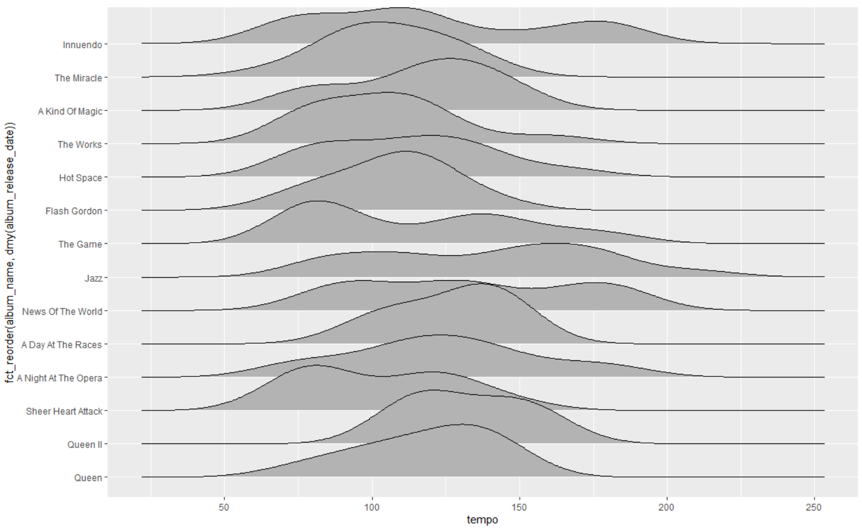


The difference here is that earlier we were reordering according to album_release_date; here, we're reordering according to tempo.

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()
```

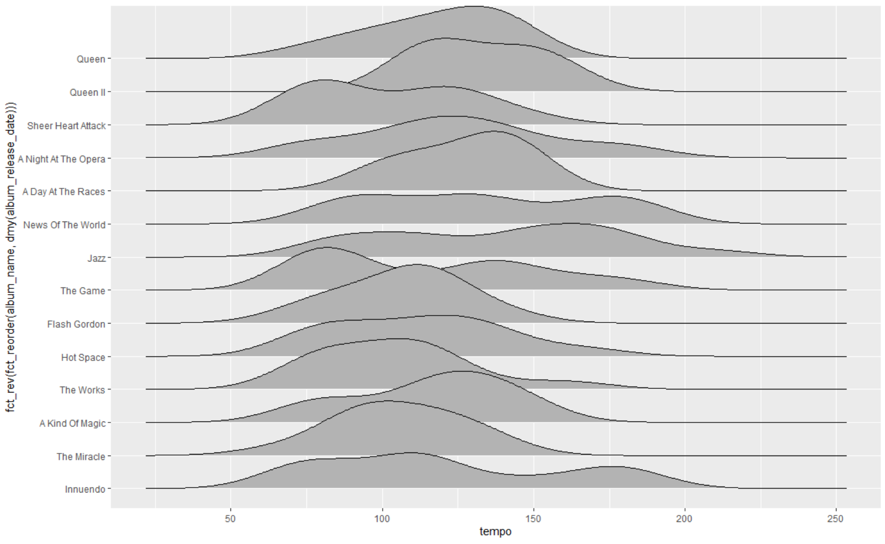


I really like ridgeline plots, which are a fairly new innovation in data visualisation. I think they make it more straightforward to compare distributions of a continuous variable across lots of different categories, as in this case (I like them for fewer categories as well, but I think they’re particularly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

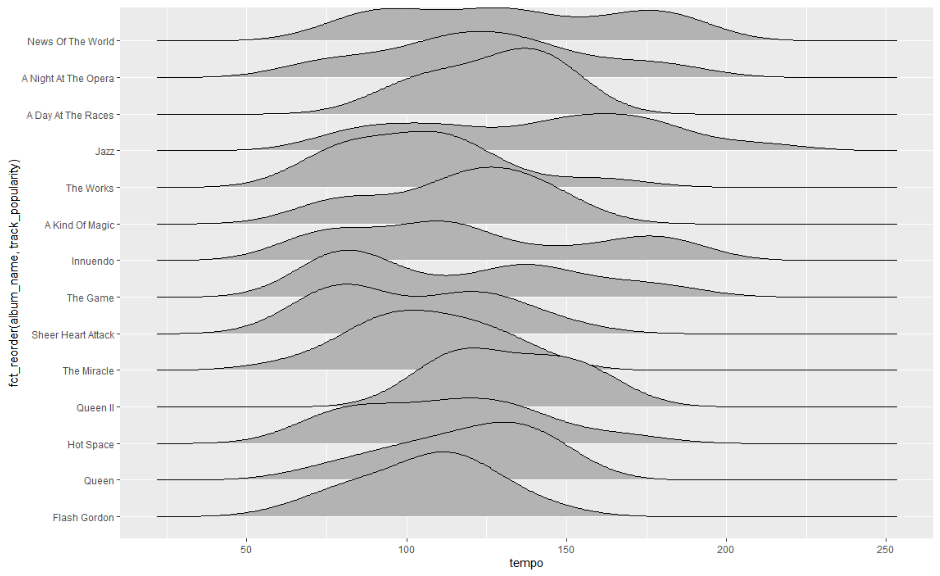
```
ggplot(data = queen) +
  aes(x = tempo,
      y = fct_rev(fct_reorder(album_name,
                              dmy(album_release_date)))) +
  geom_density_ridges()
```



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)
```

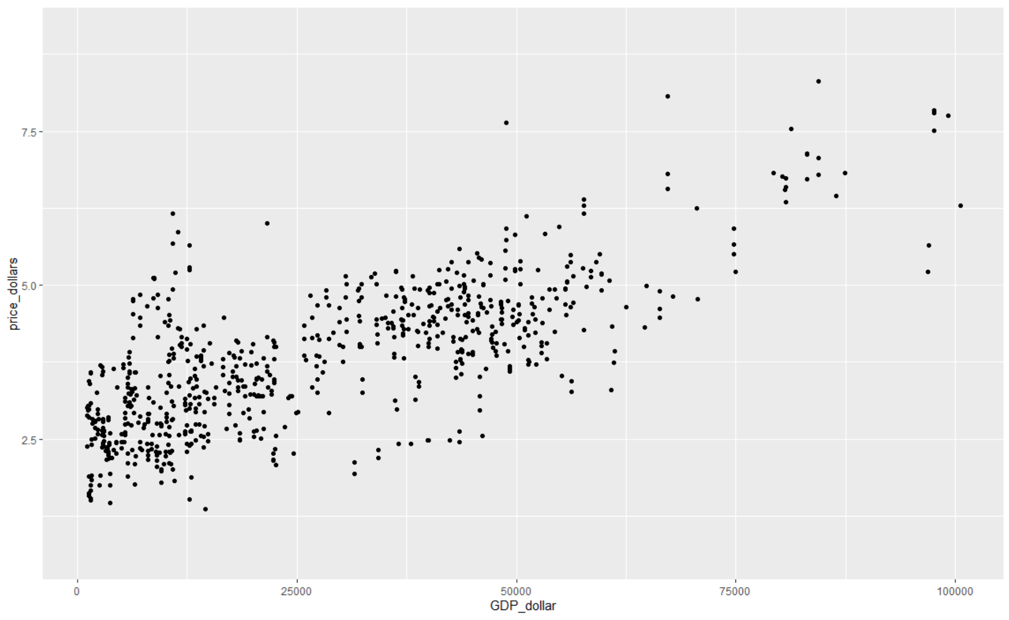
The **maccas** data is what the Economist's Big Mac index is based on; it's a measure of purchasing power in different parts of the world. (You can read more about the Big Mac index, where the data's from, and how it's compiled at this link (https://github.com/TheEconomist/big-mac-data).

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()
```

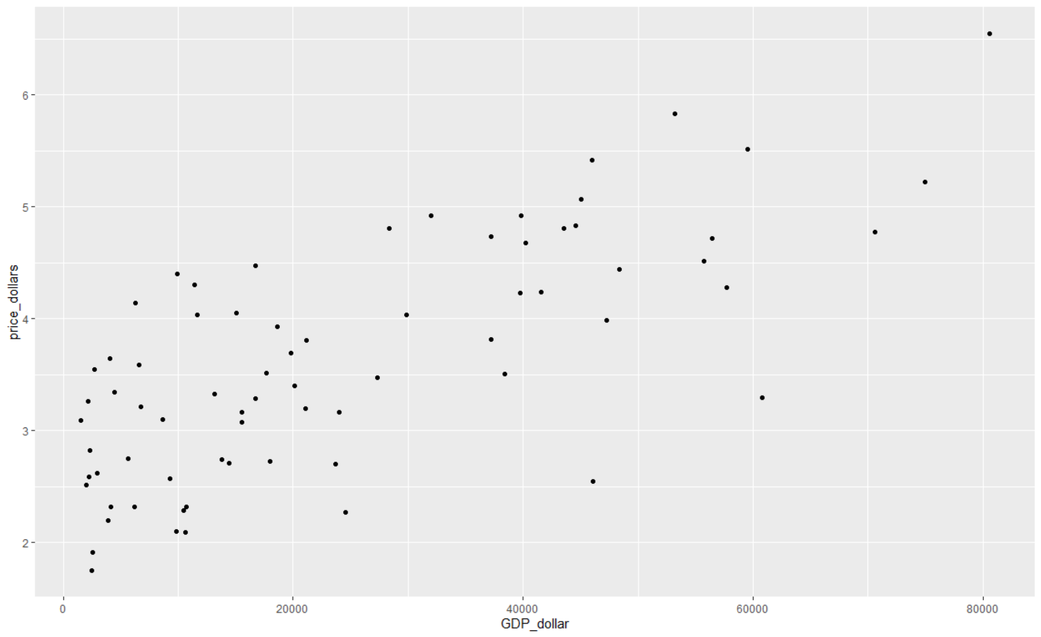


This is similar to what we've seen before; we've started with data, added a pipe, and specified what we want to do to the data. Here, we want a new variable, which I've called price_dollars: I've generated this by dividing the local price by the dollar exchange rate. The forward slash - / - means divide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

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

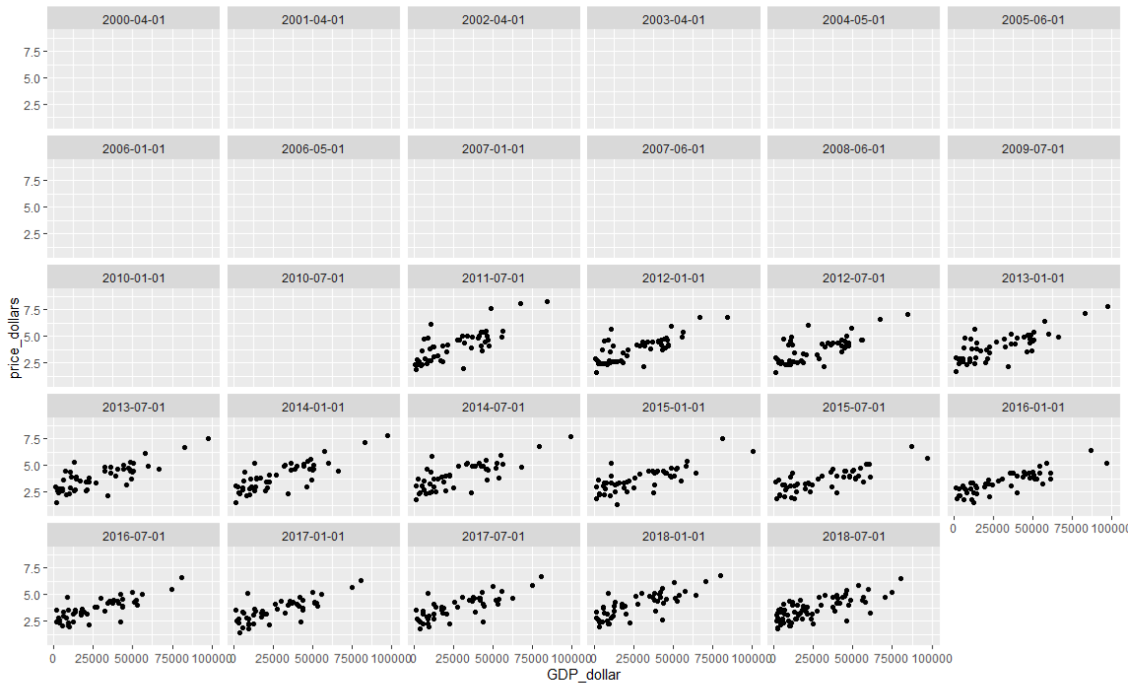
19. What if we just want the current relationship? To do this, we can limit our observations to the most recent wave of data, by going back to filter():

```
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.

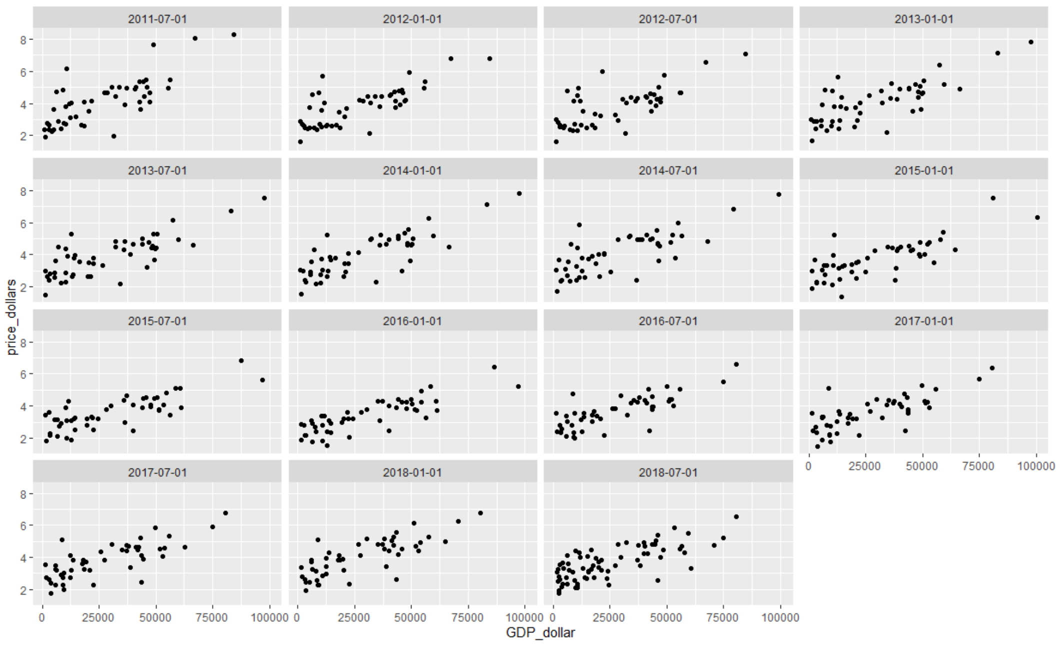
```
maccas %>%
  mutate(price_dollars = local_price/dollar_ex) %>%
  ggplot() +
  aes(x = GDP_dollar, y = price_dollars) +
  geom_point() +
  facet_wrap(~ date)
```



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.