Data Visualisation: Week 2 Bar Charts

What's happening today?

Last week, 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 commuicated with different kinds of bar charts.

Due to technical problems last week, I'm afraid you're stuck with the highest-scoring artist from last year: Taylor Swift, followed by Adele. I'm also taking the liberty of bringing in the artistry of My Chemical Romance.

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.

```r
library(tidyverse)
```

Loading data

Last week, 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 Taylor Swift, Adele, and My Chemical Romance.

Let's load some data on Taylor Swift, Fleetwood Mac, and the Arctic Monkeys.

```r
taylor <- read_csv("https://bit.ly/swifty-data")
fleet <- read_csv("https://bit.ly/fleetwood-mac-data")
monkeys <- read_csv("https://bit.ly/arctic-monkeys-data")
```

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** (I told you these terms would come back).

Finally, on the left-hand side we've got the objects we want to create. I've called them **taylor**, **fleet**, and **monkeys**, 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(taylor)
glimpse(taylor)
head(taylor)
summary(taylor)
```

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

Let's draw a graph of how many songs there are on each Taylor Swift album.

```
ggplot(data = taylor) +
  aes(x = album) +
  geom_bar()
```

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

```
ggplot(data = taylor) +
  aes(x = mode) +
  geom_bar()
```

What's the relationship like between these two variables? Do some albums skew more towards major keys, and others more towards minor keys? 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 = taylor) +
  aes(x = album, fill = mode) +
  geom_bar()
```

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 instead and you'll figure out why.

```
ggplot(data = taylor) +
  aes(x = album, colour = mode) +
  geom_bar()
  geom_bar()
```

However, this isn't the only way to communicate this information. Let's see position adjustments 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 = taylor) +
  aes(x = album, fill = mode) +
  geom_bar(position = "stack")

ggplot(data = taylor) +
  aes(x = album, fill = mode) +
  geom_bar(position = "identity")

ggplot(data = taylor) +
  aes(x = album, fill = mode) +
  geom_bar(position = "dodge")

ggplot(data = taylor) +
  aes(x = album, fill = mode) +
  geom_bar(position = "fill")

ggplot(data = taylor) +
  aes(x = album, fill = mode) +
  geom_bar(position = "jitter")
```

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

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 = taylor) +
  aes(x = mode) +
  geom_bar() +
  facet_wrap(~ album)
```

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.)

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.

We can do this using the **dplyr** package (remember that?). It's not the only way to do so, but it's a way that I like using, and it's a way that's included in the Healy book. We can start by getting the means like so:

```
taylor %>%
  group_by(album) %>%
  summarise(mean_tempo = mean(tempo))
```

You can see this has given us a small table with the information we were hoping for. What's going on here?

- we've started with the object **taylor**. 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), 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?

Now let's draw a graph.

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

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

Task

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

- how many of each of Taylor Swift's songs are in each *key*? (C sharp, E, etc)
- on which Taylor Swift albums are there the biggest number of tracks that are labelled as "explicit"?
- which Taylor Swift album has the highest average *danceability*?

Following this, please answer those same questions, as well as the question of balance between album and mode (major/minor) for either the Arctic Monkeys or Fleetwood Mac.

Finally, please come up with *another* question you're interested in answering using the available data; having answered it, please post both the question and the answer to Blackboard. You can find out more about the variables that are available at this link (https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/).