SMI105, week 3

What's happening today?

Last week, 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.

This week, 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'll become clear in due course.)

Let's also load some Spotify data, as we did last week.

```
adele <- read_csv("https://bit.ly/AdeleAlbums")
taylor <- read_csv("https://bit.ly/SMI105TS")
mcr <- read_csv("https://bit.ly/SMI105MyChemicalRomance")
ghost <- read_csv("https://bit.ly/SMI105Ghost")
beyonce <- read_csv("https://bit.ly/SMI105BeyonceBig")
```

Drawing some scatterplots

Are Taylor Swift's quicker tracks more danceable? Let's find out.

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

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?

Let's draw another scatterplot. Are Taylor Swift's quicker tracks more energetic? Let's find out.

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

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

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

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

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

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

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

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

(I'm not going to push this point in general, but in case you want to work with dates in future…)

A few of you showed me findings last week that related to 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. Let's see if Taylor Swift's more recent tracks are quicker.

```
ggplot(data = taylor) +
  aes(x = ymd(album_release_date),
      y = tempo) +
  geom_point()
```

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

- you'll note that the variable *album_release_date* is wrapped in ymd(). This is so we're declaring to R that that values of that variable are of the format *year-month-day*. Try swapping dmy() for as.factor() 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 evermore (and so on).

So, let's add some *jitter*.

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

What's the value in having done that?

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 = taylor) +
  aes(x = tempo,
      y = danceability,
      colour = album_name) +
  geom_point()
```

Playing around with **factors**

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

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

```
taylor_levels <-
  c("Taylor Swift",
    "Fearless",
    "Speak Now",
    "Red",
    "1989",
    "reputation",
    "Lover",
    "folklore",
    "evermore")
```

What's this?

- I want a new object called **taylor_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.

Now what?

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

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 = taylor_levels)**, which means that we've manually specified the order of levels of the factor to be that found in our new object, **taylor_levels**.

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

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 Beyoncé

We're looking good on Taylor Swift. Let's look at Beyoncé.

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

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? 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()
```

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 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 == "BEYONCÉ [Platinum Edition]" |
           album_name == "Lemonade" |
           album_name == "RENAISSANCE") %>%
  ggplot() +
  aes(x = tempo,
      y = danceability,
      colour = album_name) +
  geom_point()
```

What have we done here?

- as with last week 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.

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 != "HOMECOMING: THE LIVE ALBUM" &
           album_name != "I AM...SASHA FIERCE NEW DELUXE EDITION" &
           album_name != "I AM...SASHA FIERCE - Platinum Edition" &
           album_name != "I Am...World Tour" &
           album_name != "The Beyonce Experience Live Audio" &
           album_name != "The Lion King: The Gift" &
           album_name != "The Lion King: The Gift [Deluxe Edition]") %>%
  ggplot() +
  aes(x = tempo,
      y = danceability,
      colour = album_name) +
  geom_point()
```

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.

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",
                    "BEYONCÉ [Platinum Edition]",
                    "Lemonade",
                    "RENAISSANCE")

beyonce %>%
  filter(album_name == "4" |
           album_name == "B'Day" |
           album_name == "Dangerously In Love" |
           album_name == "I AM...SASHA FIERCE" |
           album_name == "BEYONCÉ [Platinum Edition]" |
           album_name == "Lemonade" |
           album_name == "RENAISSANCE") %>%
  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. Loads of lines of code, but each of them isn't that complicated; it's just a question of stitching everything together.

I might not get time for this bit

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. What's the relationship between carat and price?

```
ggplot(data = ggplot2::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. 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 = ggplot2::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.

Task

Please answer the following questions:

- what's the relationship between energy and speechiness in Taylor Swift's albums? In Beyonce's?
- how do these relationships vary by album? (please *facet* in answering this question)
- what about for My Chemical Romance? (please ensure that the albums are in chronological order)

Then, as with last week, please come up with *another* question you're interested in answering using the available data that includes drawing a scatterplot; having answered it, please submit your answer to Blackboard under the "Weekly task submission" header, along with the graph that communicates that information. 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/).