# SMI105, week 8

## What's happening today?

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. (Although we do sometimes work with data that we've downloaded through R packages that help us interact with APIs: the Spotify data we were working with in the first half of this semester was of this format.) 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 (it'll probably get messier next week).

## 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.

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

As with last week, we're running from an R project folder. It's exactly the same file as last week, and it can be found under the "Week 7" bit of the Blackboard page. If this has worked, you'll be able to run the following line:

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

and a shapefile of parliamentary constituencies should load. (It's the same file as last week.)

## 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) (sadly, we are still stuck using 2011 Census data as 2021 Census data is not up on Nomis yet by Westminster Parliamentary Constituencies);
- 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** (or, if you're anything like me with a disorganised Downloads folder, something like **bulk(16).csv**).

## Using computers

Please now move your new file to the folder that contains the .Rproj file, 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. We'll contrast this next time…)

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.)

**This** is a task that might benefit from some automation — in SMI105's Module Guidance, I prohibit you from using generative AI to help you write code and suggest that you only use it to help with error checking, while carefully prompting it to not generate any code.

Let's see if Google Gemini can create a list of short names from our dataset using a prompt. First, let's get our names:

```
names(age_structure)
```

Now, let's give the following prompt to ChatGPT. Go to chatgpt.com and sign in via Google using your University account. Ask ChatGPT:

"Take this list of long, untidy variable names and provide me with a vector in R code of short variable names (less than 15 characters) that are all unique:

*Copy and paste the names of the variables from the names() function above*

"

Of course, you will probably get a slightly different output from me, but you should get something like this (or you can re-generate the output until you do):

```
# Don't write these labels - use the ones that ChatGPT generated for you!
short_names <- c(
  "date", "geo", "geo_code", "rural_urb", "age_all",
  "age_0_4", "age_5_7", "age_8_9", "age_10_14", "age_15",
  "age_16_17", "age_18_19", "age_20_24", "age_25_29", "age_30_44",
  "age_45_59", "age_60_64", "age_65_74", "age_75_84", "age_85_89",
  "age_90_plus", "mean_age", "med_age"
)
```

If we run this, it creates a vectore called short names which is stored in our environment. **Make sure you double check that these all look correct!** To change all of the variable names in our dataset to these generated ones, we can use the following code (though remember, your object may also be different to `short_names` !)

```
age_structure_ai <- age_structure %>% set_names(short_names)
```

If you get an “incorrect number” error, this probably means that ChatGPT failed to generate the right number of variables that we need. It is prone to mistakes!

Right, let’s join the data.

# Joining the data

We did some joining last week. This week, let’s look at it in a bit more detail.

```
age_with_map <-
  left_join(westminster, age_structure_renamed, by = "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 **left_join** command means we’re combining **two** datasets. (Some people might have tried joining three datasets in one join function in the Week 7 task; this won’t work, but you can pipe joins together to join the first two together, and then the third to the result, and then the fourth, and so on, e.g. left_join(dataset1, dataset2, by = “CODE”) %>% left_join(dataset3, by = “CODE”) %>% and so on );
- left_join is different to the inner_join we were using before, as it will keep all of the unique IDs in the left-hand side dataset;
- following the left_join function, we’re specifying the two datasets we want to combine;
- finally, we’re specifying the **key** variable that both datasets have using **by**. 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.

# Arranging variables in order

Has it worked? To find out, let's see which parliamentary constituency has the highest mean age.

We're going to use two new dplyr functions to do this – **select** and **arrange**.

**select** is used to choose variables. You can use it to reduce your dataset down to only a few variables (or even one). Or you can use it to reorder the variables. We'll use it below to show how we can reduce it down to a few variables.

```
age_with_map %>%
  select(mean_age, NAME, CODE)
```

This code has taken our dataset, and selected three variables to keep – mean_age, NAME, CODE – the rest have been discarded. It's also put them in the order that we've specified within the **select** function. But this doesn't tell us which constituency has the highest mean age, we need to use **arrange** for that.

**arrange** sorts variables into ascending order. Pass the name of the variable you want to sort to the arrange function like so…

```
age_with_map %>%
  select(mean_age, NAME, CODE) %>%
  arrange(mean_age)
```

This has sorted all the observations in the dataset in ascending order according to mean_age. But this still doesn't give us the highest mean age, instead we've got the constituencies with the lowest mean age. To solve this, we can use the **desc** function within arrange.

```
age_with_map %>%
  select(mean_age, NAME, CODE) %>%
  arrange(desc(mean_age))
```

This has now sorted the observations in descending order by mean_age, so the top observation in the dataset is the constituency with the highest average age – Christchurch.

# Let's draw some maps

Now let's draw a map of the constituencies, colouring constituencies based on their mean age.

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

As we did in previous workshops, if we wanted to get rid of the missing areas we could use **drop_na**

```
age_with_map %>%
  drop_na(mean_age) %>%
  ggplot() +
  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.

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.

```
age_with_map %>%
  drop_na(mean_age) %>%
  ggplot() +
  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?

# Task

A few more things for the task this week.

- repeat the left_join we did earlier, but this time try joining the spatial data to the data where we use generative AI to create short variable names (hint: you might come across some difficulties with the key variable, check the lecture slides about how you join data with two different names for the key!)
- please draw a map of the fraction of people in each parliamentary constituency who are 65 or older. Which constituencies in England and Wales have the smallest fractions of people aged 65+; which have the largest?
- please draw a **scatterplot** of the percentage of people in each constituency aged over 65, and of the percentage of voters who voted for Reform UK in 2024 (hint: you had a copy of that data in your assessment — check whether it has a variable you can use as a key to join it to this data!)
- please draw a map of the fraction of people in each constituency who don’t have passports (you’ll need to find and download this data — excellent practice for your second assessment)
- Recreate one of your maps using the House of Commons Non-Contiguous Cartogram of the UK for **constituencies** (https://github.com/houseofcommonslibrary/uk-hex-cartograms-noncontiguous?tab=readme-ov-file (https://github.com/houseofcommonslibrary/uk-hex-cartograms-noncontiguous?tab=readme-ov-file)), and post it to Blackboard. This is a “GEOPACKAGE” type mapping file, so you might want to check the lecture slides about this to find out how to read it into R!