Data visualisation in R

Sam Langton

5 February 2020

Preamble

The page contains the course material for a workshop hosted jointly between the University of Manchester (https://www.manchester.ac.uk/) and the UK Data Service (https://ukdataservice.ac.uk/). All material and associated scripts are available on GitHub (https://github.com/langtonhugh/data_viz_R_workshop).

Background

Earlier today we covered data visualisation in R using `ggplot2` , a graphics package based on the grammar of graphics (https://vita.had.co.nz/papers/layered-grammar.html) which is fully integrated into the tidyverse (https://www.tidyverse.org/). By now you will have a solid understanding of how to build graphics using the package. Fortunately, due to the integration of spatial packages in R and ggplot, the step from making standard graphics to making spatial visualisations (e.g. thematic maps) is a small one. We'll be making use of the same skills you picked up earlier, but with spatial data. As we discussed, there are additional considerations to make when creating maps in R in terms of projections, data types and visualisation techniques, but in terms of the practical skills in code, you are going to notice a lot of similarities.

Spatial data using `sf`

One of the fastest growing packages in this area is sf (https://github.com/r-spatial/sf), which gives you access to a whole host of features and functions for use with spatial data, including visualisation. There is a highly comprehensive book available online (https://geocompr.robinlovelace.net/), written by some of its creators, which outlines sf and its uses in detail. If you are interested in taking things further after this course, that book is the way to go, but there are also plenty of general introductions (https://www.research.manchester.ac.uk/portal/en/publications/gis-and-geovisual-analysis(6f08f8f5-fb0c-4280-8c82-c8f1b7e02d76).html) to spatial data out there. For this exercise, we'll keep things simple, and focus on how to use sf to make spatial data visualisations in combination with ggplot. Should you want to know more, or would like additional resources on using spatial data in R, please do not hesitate to ask!

Raw data

Earlier today we examined neighbourhood-level burglary concentrations in Greater Manchester. This information was created from individual records which are openly available (https://data.police.uk/) online. These raw records, amongst other things, contain the location of where each crime occurred with latitude and longitude (https://www.latlong.net/) coordinates. Well, they tell you *roughly* where the crime occurred, to ensure confidentiality. The first thing we might want to do when exploring burglary victimisation is to map out these locations to identify interesting patterns and hotspots across space.

First, let's load in some individual records as a standard data frame from a .csv file. This makes use of the `readr` package within the tidyverse, just as we have done previously. This specific data set contains burglary records occurring in Manchester during January 2017.

```
burg_records_df <- read_csv("https://github.com/langtonhugh/data_viz_R_workshop/raw/master/data/burglary_records.csv")
```

Taking some time to explore the data, you'll notice two variables called `latitude` and `longitude` which collectively specify the location of each burglary record. But, in its current form, this spatial information is simply being treated like any other numeric variable in R. Using the sf package, we can convert these variables to coordinates with spatial properties for point mapping and spatial analysis. We do this using the `st_as_sf()` function, in which we specify the original data, and the variables defining the coordinates of our points. In doing so, we create a new object `burg_records_sf` (note the subtle name change) which is an sf object with spatial properties.

```
burg_records_sf <- st_as_sf(x = burg_records_df, coords = c(x = "Longitude", y = "Latitude"), crs = 4326)
```

You will notice that we also specify a Coordinate Reference System (CRS). This brings us to the topic of projections and requires a little bit more explanation.

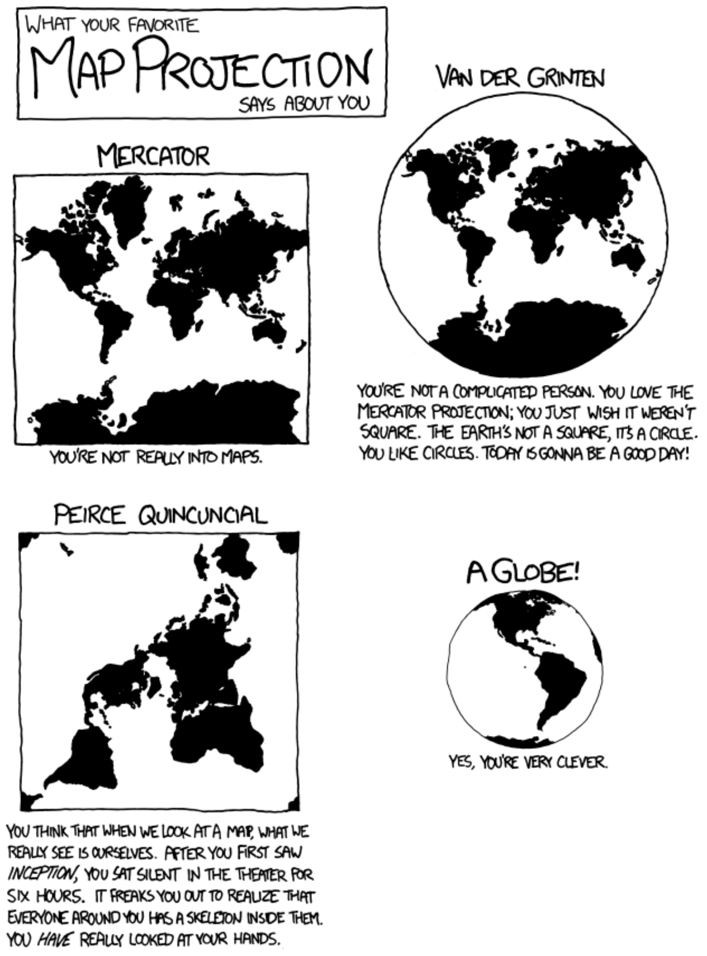
Projections

Coordinate Reference Systems (CRS) allow us to position and locate the spatial entities in your data (in this case, crime locations) on earth. Of course, when we are making maps and spatial data visualisations, we are simply *representing* real-world information. Importantly, we are doing so on a flat surface (a computer screen or piece of paper) even though the earth itself is more-or-less spherical. In an attempt to portray spatial entities, whether it be crime locations or any other phenomena, on a flat surface, we perform a transformation known as a 'projection'. This is quite the mathematical challenge, and can be carried out in countless different ways, each of which have their own advantages and disadvantages. You might be aware of the heated discussion (https://twitter.com/googlemaps/status/1025130620471656449?ref_src=twsrc%5Etfw%7Ctwcamp%5Etweetembed%7Ctwterm%5E1025130620471656449&ref_url=https%3A%2F%2Fwww.theverge.com%2F2018%2F8%2F5%2F176 maps-update-mercator-projection-earth-isnt-flat) over how Google Maps portrayed the earth on its online platform. Until recently, they used a projection known as the Mercator projection, which whilst useful for navigational purposes, also distorts the earth in a manner which makes land masses near the equator, such as Africa, appear much smaller than they actually are, and land masses near the poles, such as Greenland, much larger.

For what we are focusing on today, the important thing to know is that degrees of latitude and longitude, which you are probably fairly familiar with by name at least, use a geographic CRS called *WGS 84*. This is probably the most common one out there, so you will come across it a lot. Each reference system has a corresponding unique code from the EPSG registry (https://en.wikipedia.org/wiki/EPSG_Geodetic_Parameter_Dataset). And yes, you guessed it, the EPSG code for WGS 84 is `4326` , as stated in our code chunk above. It's problematic to do anything involving spatial data (in R, or any other software) without ensuring it is using an appropriate CRS. Some UK data, for instance, might come as Eastings and

Northings coordinates (not longitude and latitude), which would require a projected CRS known as the British National Grid (https://www.ordnancesurvey.co.uk/documents/resources/guide-to-nationalgrid.pdf), with an EPSG code of 27700. When you view the object using `View(burg_records_sf)` you will notice that there is a new column called 'geometry' containing the coordinates for each observation.

You can read more about CRS and projections on the QGIS (https://docs.qgis.org/2.8/en/docs/gentle_gis_introduction/coordinate_reference_systems.html) website, or take a look at the array of textbooks (https://books.google.co.uk/books?hl=en&lr=&id=toobg6OwFPEC) on Geographic Information Systems (GIS). For a more light-heated explanation, I would recommend this blog post (https://brilliantmaps.com/xkcd/) about what different map projections say about you. There is also a fantastic chapter (https://geocompr.robinlovelace.net/reproj-geo-data.html) on projections in a life-saving book called Geocomputation with R (https://geocompr.robinlovelace.net/).

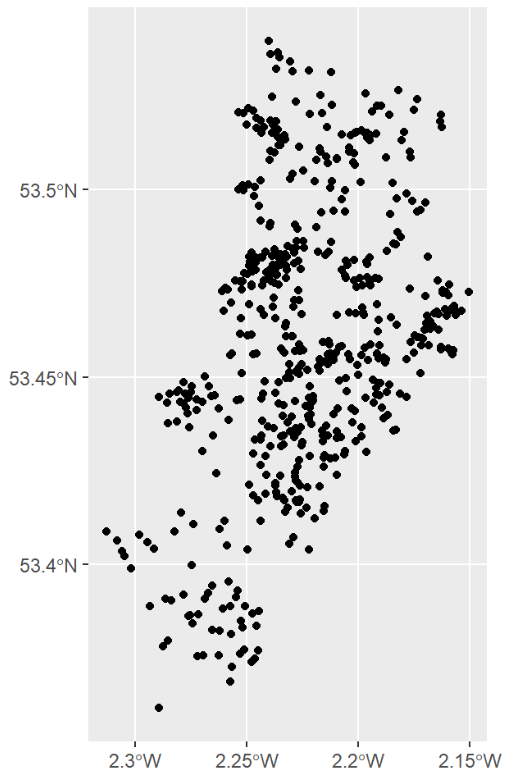

WHAT YOUR FAVORITE MAP PROJECTION SAYS ABOUT YOU
MERCATOR
YOU'RE NOT REALLY INTO MAPS.
VAN DER GRINTEN
YOU'RE NOT A COMPLICATED PERSON. YOU LOVE THE MERCATOR PROJECTION; YOU JUST WISH IT WEREN'T SQUARE. THE EARTH'S NOT A SQUARE, IT'S A CIRCLE. YOU LIKE CIRCLES. TODAY IS GONNA BE A GOOD DAY!
PEIRCE QUINCUNCIAL
YOU THINK THAT WHEN WE LOOK AT A MAP, WHAT WE REALLY SEE IS OURSELVES. AFTER YOU FIRST SAW INCEPTION, YOU SAT SILENT IN THE THEATER FOR SIX HOURS. IT FREAKS YOU OUT TO REALIZE THAT EVERYONE AROUND YOU HAS A SKELETON INSIDE THEM. YOU HAVE REALLY LOOKED AT YOUR HANDS.
A GLOBE!
YES, YOU'RE VERY CLEVER.

Source: Brilliant maps (https://brilliantmaps.com/xkcd/)

Point maps

Now we have an sf object `burg_records_sf` which contains point-level, spatially sensitive data about burglaries occurring in Manchester during January 2017. We can create a basic point map of these crimes using exactly the same skills we picked up with `ggplot2` this morning, but with a new geometry `geom_sf()`.

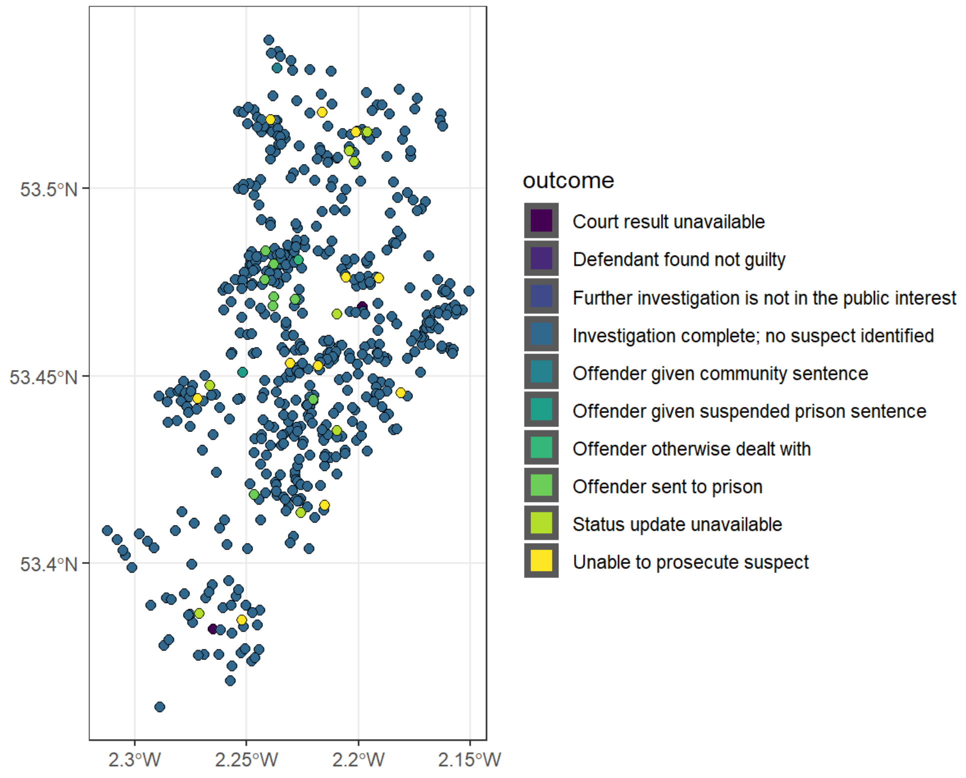
```
ggplot(data = burg_records_sf) +
  geom_sf()
```



You'll notice that to plot the points contained in our sf object we don't even need to specify any aesthetics. The axis have been automatically defined by `geom_sf()` based on the range of latitude and longitude coordinates of each observation, which are retained within the `geometry` column of the object. Even with this basic visualisation, one can identify some interesting patterns. with many burglaries clustering in specific areas of the city.

One can add additional variables to these point maps using additional aesthetics, just as we did with regular graphics earlier. For instance, to explore the outcome of these crimes we can colour each point by the `outcome` variable, which details how each burglary was resolved. Note that we specify the shape of the points to ensure we can use the fill aesthetic as required, and make some additional tweaks using the skills we learnt earlier today.

```
ggplot(data = burg_records_sf) +
  geom_sf(mapping = aes(fill = outcome), pch = 21, size = 2) +
  scale_fill_viridis_d() +
  theme_bw()
```


53.5°N
53.45°N
53.4°N
2.3°W
2.25°W
2.2°W
2.15°W
outcome
Court result unavailable
Defendant found not guilty
Further investigation is not in the public interest
Investigation complete; no suspect identified
Offender given community sentence
Offender given suspended prison sentence
Offender otherwise dealt with
Offender sent to prison
Status update unavailable
Unable to prosecute suspect

Whilst the above might help us identify interesting patterns, point-level open crime data is rarely used in isolation for detailed analysis. For one thing, the data is not entirely accurate (https://www.tandfonline.com/doi/full/10.1080/15230406.2014.972456) because the raw coordinates have been 'snapped' to pre-defined points to ensure confidentiality. This means that points are highly likely to be overlapping, giving a skewed picture of the distribution. There are ways round this, such as through jittering (https://r-spatial.github.io/sf/reference/st_jitter.html), but even so, not much other data is collected at such a fine-grained scale. Instead, area-based visualisations of crime data can be deployed, and matched with associative data, such as that of the census, to provide a more comprehensive picture of crime concentrations.

Area-based maps

To demonstrate an area-based map using sf, we will return to our neighbourhood-level data on burglary counts, but this time only for Manchester. Remember, we defined our 'neighbourhoods' as Lower Super Output Areas (https://census.ukdataservice.ac.uk/use-data/guides/boundary-data.aspx) (LSOA) which are useful units of analysis due to their linkage with census data. You can download these boundaries (and a host of other spatial units) from the UK Data Service (https://census.ukdataservice.ac.uk/get-data/boundary-data.aspx). Here, we are going to download a .zip folder containing the shapefile (https://doc.arcgis.com/en/arcgis-online/reference/shapefiles.htm) of the most recent LSOA boundaries for Manchester.

You can download (https://github.com/langtonhugh/data_viz_R_workshop/raw/master/data/manc_lsoa.zip) the .zip folder directly from the GitHub associated with this workshop. Save it to a local folder on your machine and ensure it is unzipped. If you feel comfortable enough, you can download and unzip (https://stackoverflow.com/questions/3053833/using-r-to-download-zipped-data-file-extract-and-import-data) the file directly from within R, but this is not necessary.

Once you have the folder downloaded and unzipped on your machine, you can load in the shapefile using `st_read()` , assigning the output to a new object. Please remember that you will need to alter the working directory below with your own!

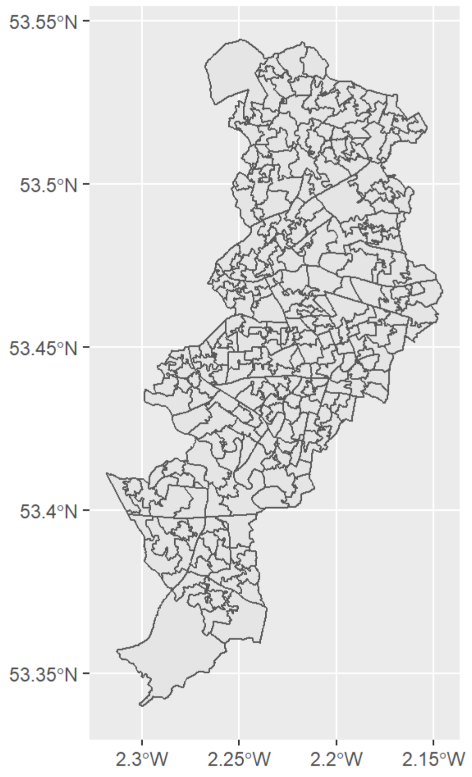
```
manc_sf <- st_read(dsn = "data/burglary_lsoa.shp")
```

Once you've loaded this into your R environment, you will get a brief summary of the data. One of these includes a note on the CRS. We can ensure it is identical to the point-level data we used earlier (WGS 84) using `st_transform()` and the relevant EPSG code.

```
manc_sf <- st_transform(manc_sf, crs = 4326)
```

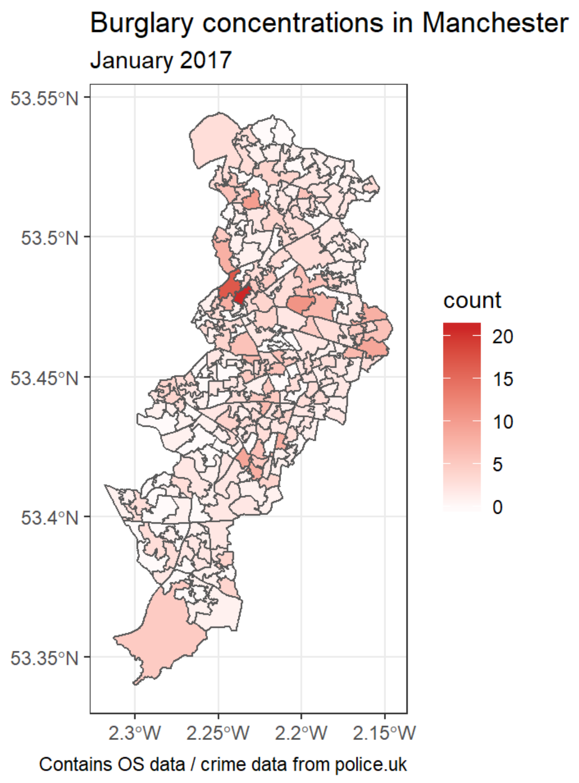
You will notice using `View(manc_sf)` that we have a number of variables at LSOA-level, including the burglary counts and deprivation measures, just as we had this morning, as well as the `gemometry` column, which defines the boundaries of each neighbourhood. Once again, we can take a look at this geometry using a quick bit of ggplot code.

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



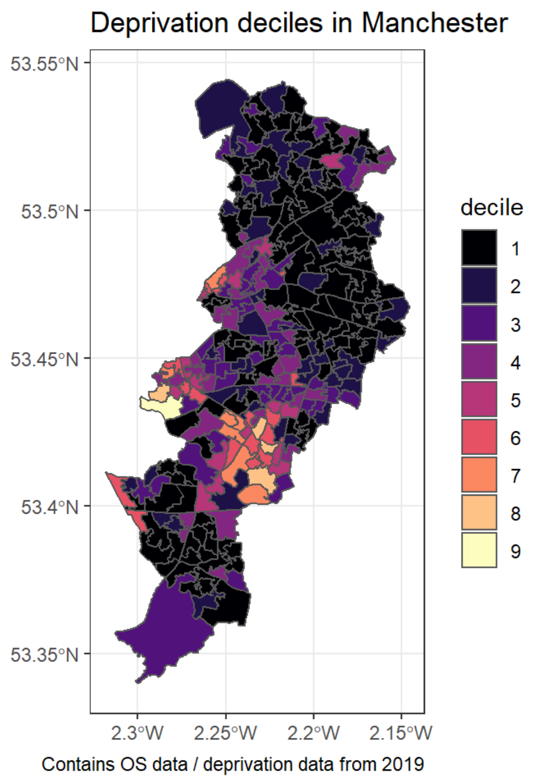
As you might have already guessed, we can colour each polygon (i.e. unit, LSOA) according to a variable using the `aesthetics` argument. Let's take a look at the distribution of burglary, with some tweaks to make the graphic more complete.

```
ggplot(data = manc_sf) +
  geom_sf(mapping = aes(fill = brglry_)) +
  scale_fill_continuous(low = "snow", high = "firebrick3") +
  labs(title = "Burglary concentrations in Manchester",
       subtitle = "January 2017",
       caption = "Contains OS data / crime data from police.uk",
       fill = "count") +
  theme_bw()
```


Burglary concentrations in Manchester
January 2017
53.55°N
53.5°N
53.45°N
53.4°N
53.35°N
2.3°W
2.25°W
2.2°W
2.15°W
count
20
15
10
5
0
Contains OS data / crime data from police.uk

Discrete (categorical) variables can be visualised similarly (again, not much difference from this morning!). Let's take a look at the deprivation deciles using a palette more suitable for those with colour blindness.

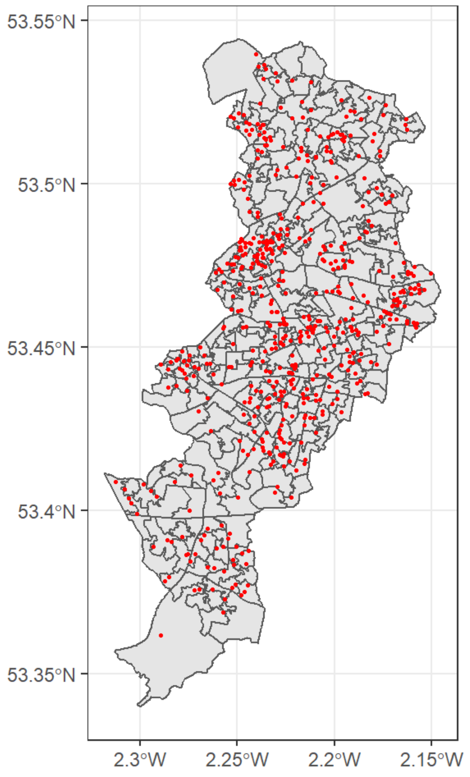
```
ggplot(data = manc_sf) +
  geom_sf(mapping = aes(fill = as.factor(IMDdeci))) +
  scale_fill_viridis_d(option = "magma") +
  labs(title = "Deprivation deciles in Manchester",
       caption = "Contains OS data / deprivation data from 2019",
       fill = "decile") +
  theme_bw()
```


Deprivation deciles in Manchester
53.55°N
53.5°N
53.45°N
53.4°N
53.35°N
2.3°W
2.25°W
2.2°W
2.15°W
decile
1
2
3
4
5
6
7
8
9
Contains OS data / deprivation data from 2019

Overlaying maps

You might remember at the very beginning of the day, when covering the framework of ggplot code, the idea that the same thing can be achieved with different code. When overlaying different sf objects on top of one another, this proves especially useful. Here, we want to plot the point-level burglary locations on top of a basic outline of neighbourhoods in Manchester, to give a frame of reference. In such a circumstance, we first lay down at empty `ggplot()` function, and then subsequently add our maps, which are contained in different sf objects. Be careful which order you do things in! If you put `manc_sf` second, it will just obscure the points, unless you specify the fill as transparent.

```
ggplot() +
  geom_sf(data = manc_sf) +
  geom_sf(data = burg_records_sf, size = 0.5, col = "red") +
  theme_bw()
```



It's handy to know that the following would produce a similar result, although you will notice that boundaries themselves will cover the points.

```
ggplot() +
  geom_sf(data = burg_records_sf, size = 0.5, col = "red") +
  geom_sf(data = manc_sf, fill = "transparent")
```

As an aside, you might notice that on the above maps, some neighbourhoods are much larger than others. This is because LSOAs are designed to be fairly uniform by resident populations, but some residential areas are more densely populated than others. Variation between the sizes of areas being mapped can be problematic, and sometimes even misleading, so it is worth considering this when creating maps for dissemination to the public or colleagues (or even yourself!). There are creative ways of mapping which can account for this, such as cartograms or regular grids. If you are interested in exploring these, there's a great package called geogrid (https://github.com/jbaileyh/geogrid) for creating regular grids, and some open code (http://rpubs.com/profrichharris/342278) from Richard Harris (University of Bristol) for cartograms and 'hexograms'.

Now you are familiar with making spatial visualisations in R, try adding a map to your `cowplot` from this morning, and adjusting it with a customised theme.

Resources

- As discussed, *Geocomputation in R* (https://geocompr.robinlovelace.net/) is a comprehensive book covering all manner of spatial skills in R using `sf` . It is available for free online but you can also buy a paper copy (https://www.amazon.com/Geocomputation-Chapman-Hall-Robin-Lovelace/dp/1138304514).
- *An Introduction to R for Spatial Analysis and Mapping* is a useful resource. The second edition (https://www.amazon.co.uk/dp/1526428504/) focuses on `sf` .
- At the 2019 useR! (http://www.user2019.fr/) conference, there was a 1-day workshop on spatial and spatiotemporal data analysis in R. The material is freely available online (https://github.com/edzer/UseR2019). Clicking on the 'materials' hyperlinks will take you to the worksheets for the morning and afternoon sessions.
- Twitter is a good resource for keeping up-to-date with things. There are many people, but following Robin Lovelace (https://twitter.com/robinlovelace), Jakub Nowosad (https://twitter.com/jakub_nowosad), Angela Li (https://twitter.com/CivicAngela) and Edzer Pebesma (https://twitter.com/edzerpebesma) will give you a good start.