Crime Mapping and Spatial Data Analysis using R

Juanjo Medina and Reka Solymosi

2021-05-14

Contents

Preamble: how to use this book

This book aims to provide the reader with an introduction to R as an engine for spatial data visualisation and analysis for those interested in crime analysis and criminology. The book is based in our lab materials for course units we have offered to upper level undergraduate and MA students on criminology interested in crime mapping, as well as in similar material we have used when training crime analysts working for the police or other law enforcement agencies.

Given the source material, this book is intended to be uses it as a companion text for similar course units in crime mapping, the geography of crime, environmental criminology, or crime analysis. Equally, it can be used by students, practitioners, and academics alike interested in learning more about R and its GIS and spatial analysis capacity. It is not an advanced statistics text, but rather an applied text. Someone "self teaching" R for these purposes will find it helpful and, thus, unlike reference books, it is better to read and practice each chapter in sequence.

Crime mapping and analysis sits at the intersection of geocomputation, data visualisation and cartography, spatial statistics, environmental criminology, and crime analysis. Environmental criminology provides the substantive background. It concerns the analysis of the spatial and geographical distribution of crime.

This text cannot make justice to each of this bodies of enquiry, professional practice, and literature. We cannot offer a comprehensive and systematic treatment of each of these areas. What we do is provide a, we hope helpful, introduction to R as a way to bring together these specialties and offer adequate references to our readers so that they can deepen their understanding of each of them.

Although all the examples we use concern the study of crime, there is fairly limited substantive criminology in the text. In fact, and despite the title, the volume could also be used, more generally, as a companion text for courses on social science spatial data analysis for the techniques we cover are quite general.

In the first part of this volume we introduce key concepts for geographic analysis and representation and provide the reader with the basis to visualise spatial crime data. We then introduce a series of tools and techniques that are relevant to study spatial homogeneity and dependence of crime data. A key focus in this section is how to visualise and detect local clusters of crime and repeat victimisation. The final chapters introduce the students models to account for the distribution of crime across space. In terms of spatial data analysis the focus of the book is on spatial point pattern analysis and lattice or area data analysis. Geostatistics have fewer applications in crime analysis and research and, therefore, we do not cover it here.

Chapter 1

Chapter 1: Producing your first crime map

1.1 Introduction

This chapter introduces some basic concepts and will get you started to make some maps in R. We will learn how we can take regular crime data, and assign the appropriate geometry for our chosen unit of analysis. Spatial or environmental criminology works with data that often is discrete. In other scientific disciplines the mapped data is continuous (e.g., temperature) across the study surface. In environmental criminology, on the other hand, we mostly work with data represented by points (e.g., locations of a crime) or counts and rates of crimes within a particular geographical unit (e.g., a neigbourhoud, a census trace, a municipality, etc). Therefore in these and the first few chapters we will focus on ways to visualise this kind of discrete data. We will produce some maps, using the package `ggplot2`, and learn some key terms around projection and coordinate reference systems which will be essential for subsequent chapters.

As we will discover throughout the book there are multiple R packages that have been developed to visualise spatial data. We will introduce the ones we feel are more important in different parts of the book. They all have advantages and disadvantages, but many offer similar functionality. Sometimes choosing one or the other is a matter of personal preference. In this chapter we will focus on mapmaking with `ggplot2`, a general package for all kinds of data visualisations -not just maps- based on the theory of the grammar of graphics. If you are not new to R, you may already be familiar with this package. Our intention in this chapter is to let you to quickly produce your first map. In subsequent chapters we will further refine their look and aesthetic appeal.

In this chapter we will use the following packages:

```r
# Packages for reading data and data carpentry
library(readr)
library(tibble)
library(janitor)
library(dplyr)

# Packages for handling spatial data
library(sf)

# Packages for visualisation and mapping
library(ggplot2)
library(ggspatial)
```

1.2 Geospatial Perspective: key terms and ideas

Geospatial analysis provides a distinct perspective on the world, a unique lens through which to examine events, patterns, and processes that operate on or near the surface of our planet. Ultimately geospatial analysis concerns what happens where, and makes use of geographic information that links features and phenomena on the Earth's surface to their locations.

We can talk about a few different concepts when it comes to spatial information. These are:

- Place
- Attributes
- Objects

1.2.1 Place

At the center of all spatial analysis is the concept of *place*. People identify with places of various sizes and shapes, from the room with the parcel of land, to the neighbourhood, to the city, the country, the state or the nation state. Places often have names, and people use these to talk about and distinguish names. Names can be official. Places also change continually as people move. The basis of rigorous and precise definition of place is a coordinate system, a set of measurements that allows place to be specified unambiguously and in a way that is meaningful to everyone.

In environmental criminology, place has acquired different meanings through history. The first geographical studies of crime during the XIX century looked at variation across provinces in France. Later on, in the first decades of the XX century, the Chicago School of Sociology focused on the study of neighbourhoods. Today there is a greater interest in microplaces such as street segments or particular addresses. Although an interest in variation across large administrative units (as "place") remains within criminology, there has been a trend towards understanding place as particular locations.

1.2.2 Attributes

Attribute has become the preferred term for any recorded characteristic or property of a place. It is what more generally in statistics we may call a variable and in data science a feature. A place's name is an obvious example of an attribute. But there can be other pieces of information, such as number of crimes in a neighbourhood, or the GDP of a country. Within geographic information science (GIS) the term 'attributes' usually refers to records in a data table associated with individual elements in a vector map, or cells in a grid (raster or image file). These data behave exactly as data you have encountered in your past experience. The rows represent observations, and the columns represent variables. The variables can be numeric or categorical, and depending on what they are, you can apply different methods to making sense of them. The difference with other kind of data table is that the observations, your rows, correspond to places or locations and that the data necessarily will include elements that allow us to place it in a map.

1.2.3 Spatial objects

In spatial analysis it is customary to refer to places as objects. These objects can be a whole country, or a road. In forestry, the objects of interest might be trees, and their location will be represented as points. On the other hand, studies of social or economic patterns may need to consider the two-dimensional extent of places, which will therefore be represented as areas. These representations of the world are part of what is called the **vector data model**: A representation of the world using points, lines, and polygons. Vector models are useful for storing data that have discrete boundaries, such as country borders, land parcels, and streets. This is made up of points, lines, and areas (polygons):

- *Points*
    - *Points are pairs of coordinates, in latitude/longitude or some other standard system. In the context of crime analysis, the typical point we work with, though not the only one, represents the specific location of a criminal event. These points form patterns we explore and analyse.
- *Lines*
    - Lines are ordered sequences of points connected by straight lines.
- *Areas* (polygons)

- Areas are ordered rings of points, also connected by straight lines to form polygons. It can contain holes, or be linked with separate islands. Areas can represent neighbourhouds, police districts, municipal terms, etc. You may come across the term **lattice data** to denote the type of data we work with when we observed attributes of areas and want to explore and analyse this data.

Objects can also be **raster data**. Raster data is made up of pixels (or cells), and each cell has an associated value. Simplifying slightly, a digital photograph is an example of a raster dataset where each pixel value corresponds to a particular colour. In GIS, the pixel values may represent elevation above sea level, or chemical concentrations, or rainfall, etc. The key point is that all of this data is represented as a grid of (usually square or hexagonal) cells.

1.2.4 Networks

We already mentioned lines that constitute objects of spatial data, such as streets, roads, railroads, etc. Networks constitute one-dimensional structures embedded in two or three dimensions. Discrete point objects may be distributed on the network, representing phenomena such as landmarks, or observation points. Mathematically, a network forms a graph, and many techniques developed for graphs have application to networks. These include various ways of measuring a network's connectivity, or of finding the shortest path between pairs of points on a network.

1.2.5 Maps and their types

Historically maps have been the primary means to store and communicate spatial data. Objects and their attributes can be readily depicted, and the human eye can quickly discern patterns and anomalies in a well-designed map.

In GIS we distinguish between reference and thematic maps. A **reference map** places the emphasis on the location of spatial objects such as cities, mountains, rivers, parks, etc. You use these maps to orient yourself in space and find out the location of particular places.

Thematic maps or statistical maps, on the other hand, are about the spatial distribution of attributes or statistics. For example, the number of crimes across different neighbourhouds. Our focus in this book is on thematic maps, but often when producing thematic maps you may use a reference map as a backdrop, as a *basemap*, to help interpretation and to provide context. In this and subsequent chapters we will introduce different types of thematic maps.

Another useful distinction is between **static** and **interactive** maps. Your traditional printed road map is an example of a static map, whereas the web application Google maps is an example of an interactive map. In an interactive map you can zoom in and out, you can select and query information about objects in an interactive fashion, etc. In this chapter we introduce `ggplot2`, which excels at static maps. In other chapters we will introduce `leaflet`, which is particularly useful for interactive purposes, and `tmap`, that can shift between static and interactive mode.

1.2.6 Map projections and geographic coordinate systems

Whenever we put something into a map we need some sort of system to pinpoint the location. A coordinate system allows you to integrate any dataset with other geographical datasets within a common framework. There are hundreds of them. It is common to distinguish between **geographic coordinate systems** and **projected coordinate systems**. A *geographic coordinate system* is a *three dimensional* reference system that enables you to locate any location on earth. Often this is done with longitude, latitude and elevation. *Projected coordinate systems* or map projections, on the other hand, try to portray the surface of the earth or a portion of the earth on a *two dimensional* flat piece of paper or computer screen.

All projections of a sphere like the earth in a two dimensional map involve some sort of distortion. You can't fit a three dimensional object into two dimensions without doing so. Projections differ to a large extent on the kind of distortion that they introduce. The decision as to which map projection and coordinate reference system to use, depends on the regional extent of the area you want to work in, on the analysis you want to do and often on the availability of data. Knowing the system you use would allow you to translate your data into other systems whenever this may be necessary. Often you may have to integrate data that is provided to you in different coordinate or projected systems. As long as you know the systems, you can do this.

A traditional method of representing the earth's shape is the use of globes. When viewed at close range the earth appears to be relatively flat. However when viewed from space, we can see that the earth is relatively spherical. Maps, are representations of reality. They are designed to not only represent features, but also their shape and spatial arrangement. Each map projection has advantages and disadvantages. The best projection for a map depends on the scale of the map, and on the purposes for which it will be used. For your purposes, you just need to understand that essentially there are different ways to flatten out the earth, in order to get it into a 2-dimensional map.

The process of creating map projections can be visualised by positioning a light source inside a transparent globe on which opaque earth features are placed. Then project the feature outlines onto a two-dimensional flat piece of paper. Different ways of projecting can be produced by surrounding the globe in a cylindrical fashion, as a cone, or even as a flat surface. Each of these methods produces what is called a map projection family. Therefore, there is a family of planar projections, a family of cylindrical projections, and another called conical projections.

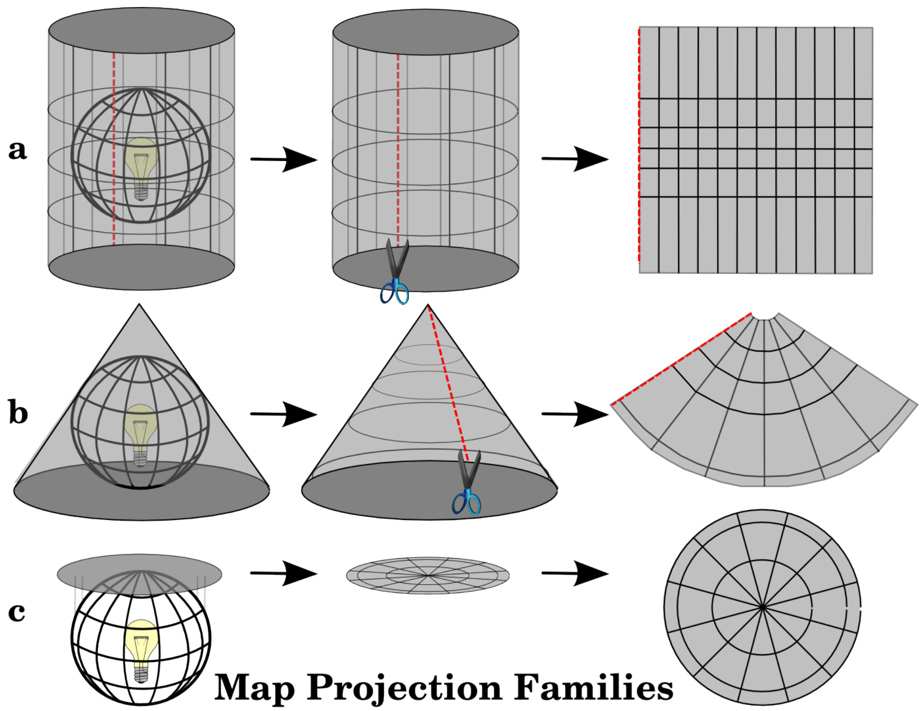


Figure 1.1: figure_projection_families

With the help of **coordinate reference systems** (CRS) every place on the earth can be specified by a set of three numbers, called coordinates. In general CRS can be divided into **projected coordinate reference systems** (also called Cartesian or rectangular coordinate reference systems) and **geographic coordinate reference systems**.

The use of Geographic Coordinate Reference Systems is very common. They use degrees of latitude and longitude and sometimes also a height value to describe a location on the earth's surface. The most popular is called **WGS 84**. This is the one you will most likely be using, and if you get your data in latitude and longitude, then this is the CRS you are working in. It is also possible that you will be using a projected CRS. This two-dimensional coordinate reference system is commonly defined by two axes. At right angles to each other, they form a so called XY-plane. The horizontal axis is normally labeled X, and the vertical axis is normally labeled Y.

Working, for example, with data in the UK, on the other hand, you are most likely to be using **British National Grid (BNG)**. The Ordnance Survey

National Grid reference system is a system of geographic grid references used in Great Britain, different from using Latitude and Longitude. In this case, points will be defined by "Easting" and "Northing" rather than "Longitude" and "Latitude." It basically divides the UK into a series of squares, and uses references to these to locate something. The most common usage is the six figure grid reference, employing three digits in each coordinate to determine a 100 m square. For example, the grid reference of the 100 m square containing the summit of Ben Nevis is NN 166 712. Grid references may also be quoted as a pair of numbers: eastings then northings in meters, measured from the southwest corner of the SV square. For example, the grid reference for Sullom Voe oil terminal in the Shetland Islands may be given as HU396753 or 439668,1175316

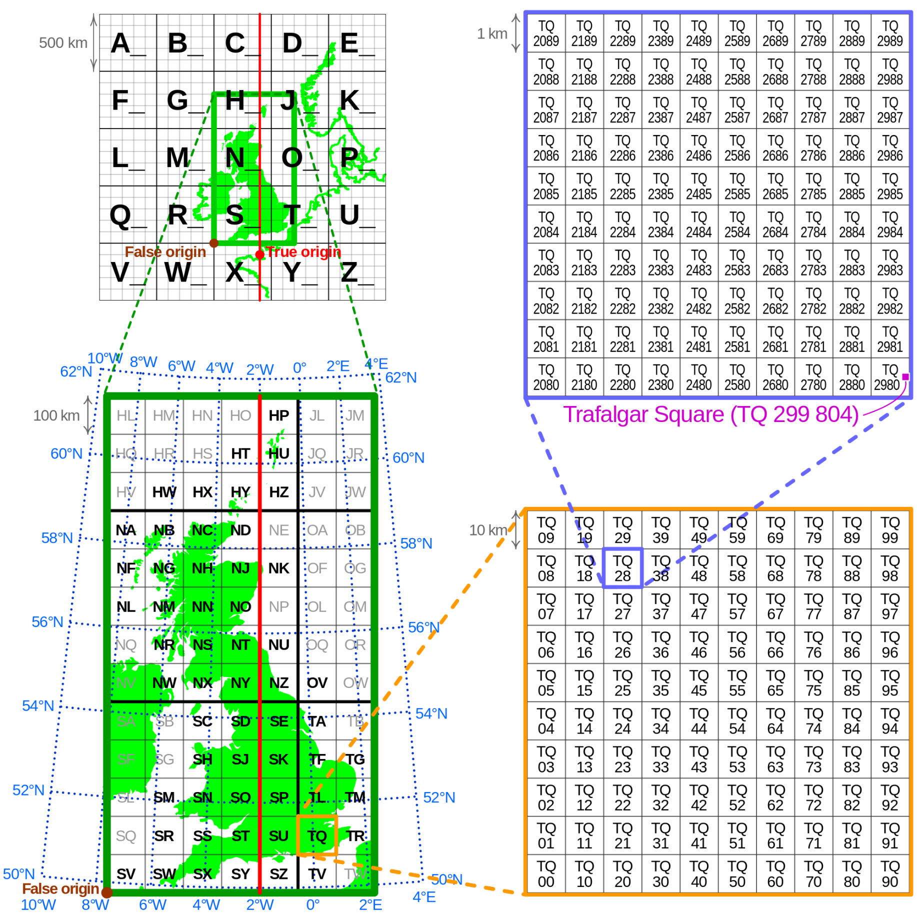


Figure 1.2: BNG

This will be important later on when we are linking data from different projections, or when you look at your map and you try to figure out why it might look “squished.”

You will often see the notation *epsg* when referring to different coordinate systems. This refers to the EPSG registry or EPSG Geodetic Parameter Dataset. It is a collection of geodetic datums, spatial reference systems, earth ellipsoids, coordinate transformations, and related units of measurement. All standard coordinate systems will have one of these codes assigned to it. So, for example, the WGS84 coordinate system corresponds the epsg key 4326, whereas the British National Grid has the key 27700. Having this common framework to identify projections make things much easier when we want to change our data from one system to another. You can query the website https://epsg.io/ for the different keys and information associated to each of them. This EPSG dataset was originally developed by the European Petroleum Survey Group(EPSG), thus the name.

1.2.7 Summary

Right so hopefully this gives you a few things to think about. Be sure that you are confident to know about:

- Spatial objects - what they are and how they are represented
- Attributes - the bits of information that belong to your spatial objects
- Maps and projections - especially what WSG84 and BNG mean, and why it’s important that you know what CRS your data have

1.3 Getting crime data to put on a map: reading data from the web

We live in a world awash with data and through this book we will use the different examples to show you some useful places and sources you can use to obtain your spatial data. Most geographically referenced crime data is based on crime reported to the police, although there are some notable exceptions (e.g., public health data on violence, geocoded victim survey data, etc.). Many police departments across the world make this data readily available to the public or researchers. This is more common in places like the US, where crime mapping applications first got established, but increasingly we see publicly available crime data from other countries as well.

For our first map we will use data from the UK, which can be downloaded from the police.uk website. Let’s download some data for crime in Manchester. To do acquire the data, we could open the data.police.uk/data website and then choose the data tab, in order to manually download some data. After selecting a data range, a police force, and the type of data, one could generate a file. This would take you to a download page, where you have to click the `Download now` button. This will open a dialogue to save a .zip file. After all this pointing and clicking, one would have to unzip the file and read it into R.

But, programmers are lazy, and the whole point of using code-based interfaces is that we get to avoid doing unneccessary work, like point-and-click downloading of files. And when data exists online in a suitable format, we can tell R to read the data in from the web directly, and cut out the middle man (that being ourselves in our pointing-and-clicking activity).

How can we do this? We say, "hello R, i would like to create a new object please and I will call this new object `my_data`." We do this by typing the name we are giving the object and the assignment function `<-`. Then on the right hand side of the assignment function, there is the value that we are assigning the variable. So it could be a bit of text, or it could be some function, for example when you read a csv file with the `read_csv()` function from the `readr` package.

So if we're reading a csv, we also need to specify *where* to read the csv from. Where should R look to find this data? This is where normally you are putting in the path to your file. Something like:

```
my_data <- read_csv("path to my file here")
```

Well what if your data does not live on your laptop or PC? If there is a way that R can still access this data just by following a path, then this approach will still work. So how can we apply this to getting data from the web? All you need is the url that indicates the file location.

```
my_data <- read_csv("www.data.com/the_data_i_want")
```

The lion share of the data we use in this book are, for convenience, stored in the webpage for this book and so can be retrieved from there. We will be providing the url and code needed to get this data as we go along. For the illustrations in this chapter we are using a csv file that looks at crime in Greater Manchester during June of 2019. The url for obtaining this file is shown below:

```
crimes <- read_csv("https://raw.githubusercontent.com/maczokni/crime_mapping/master/data/2019-06-
```

If you look at the *Environment* window in the top right corner of RStudio you should see now a new object that contains a *tibble*, a particular format for dataframes, and that is called `crimes`. It will tell you how many observations (rows - and incidentally the number of recorded crimes in June 2019 within the GMP jurisdiction) and how many variables (columns) your data has.

Let's have a look at the crimes dataframe with the `View()` function. This will open the data browser in RStudio

```
View(crimes)
```

If you rather just want your results in the console, you can use the `glimpse()` function from the `tibble` package. This function does just that, it gives you a quick glimpse of the first few cases in the dataframe. Notice that there are two columns (Longitude and Latitude) that provide the require geographical coordinates that we need to plot this data.

```
glimpse(crimes)
```

You may notice that a lot of the variable names are messy in that they have a space in them - this can cause issues, so before playing around too much with the data we want to clean this up. Luckily there is a very handy package you can use for this called `janitor` which contains the function `clean_names()`.

```
crimes <- clean_names(crimes)
```

Now the names are much neater. You can print them all for a view using the `names()` function:

```
names(crimes)
```

1.4 From dataframes to spatial objects: finding spatial information in our data

Having had a chance to inspect the data set you've downloaded, let's consider what sort of spatial information we might be able to use. If you have a look at the column names, what are some of the variables which you think might have some spatial component? Have a think about each column, and how you think it may help to put these crimes on the map.

There are a few answers here. In fact there are one each to map onto point, line, and polygon, which we read about earlier.

1.4.1 The point

First, and possibly most obvious, are the coordinates provided with each crime incident recorded. You can find this in the two columns - Longitude and Latitude. These two column help put each crime incident on a specific point on a map. For example, let's take the very first crime incident. Here we use the `head()` function and specify that we want the first 1 rows only with `n=1` parameter.

```
head(crimes, n = 1)
```

You can see that the values are for Longitude and for Latitude. These two numbers allow us to put this point on a map.

1.4.2 The line

Another column which contains information about *where* the crime happened is the aptly named *location* variable. This shows you a list of locations related to where the crimes happened. You may see a few values such as on or near XYZ street. Let's look again at the first entry.

You can see that the value is this isn't great, as we might struggle to identify *which* parking area… Some other ones are more useful, let's look at the last entry for example with the `tail()` function.

```
tail(crimes, n = 1)
```

You can see that the value is . This makes our crime much easier to find, we just need to locate where is . We might have a file of lines of all the roads of Manchester, and if we did, we can link the crime to that particular road, in order to map it.

1.4.3 The polygon

What more? You may also have seen the column "lsoa_name" seems to have some spatial component, Let's have a look at the first crime again. You see the value for LSOA name is - Bolton we know is a Borough of Greater Manchester, but what is the 001 for ?

Well, it denotes a particular geographical sub-unit within the municipality of Bolton called a **Lower Layer Super Output Area**. This is a unit of UK Census Geography. The basic unit for Census Geography in the UK is an 'Output area' - this is the resolution at which we can access data from the UK Census. The Output Area (OA) is therefore the smallest unit we could use. The census in other countries use different names for the units for which they publish information, but the logic is similar.

There are 181,408 OAs, 34,753 lower layer super output areas (LSOA) and 7,201 middle layer super output areas (MSOA) in England and Wales. The neat thing about these Census geographies is the idea is that they don't change much from census to census (unlike other administrative boundaries) and in the UK case were created with statistical analysis in mind (they were designed to be as homogeneous as possible). The less neat thing is that although we use them to operationalise the concept of neighbourhood a lot, they may not bear much resemblance to what residents might think of as their neighbourhood. This is a common problem in the UK and elsewhere (that has been widely discussed in the literature) when relying on census units as our proxy for community or neighbourhood, but one that is hard to escape from, for these units are those at which key demographic variable is typically sourced and published.

Anyway back to our crime data. You see we have two columns that reference LSOAs, `lsoa_name` and `lsoa_code`. We can use these to *link* our crime data to a file containing the geometries needed to put the crime data on the map.

1.5 Mapping crime events with simple features and ggplot2

1.5.1 The simple features framework

So how can we use these spatial details in our dataset to put our crimes on the map? We need to somehow specify a geometry for our data, which links each unit of analysis (whether that is the point, line, or polygon) to a relevant geographical representation, which allows us to put this thing on the map.

How you add geographical information will vary with the type of information we have, but in all of these, we will use the **simple features** framework. What are simple features? `sf` package author Edzer Pebesma describes simple features as a standardized way of encoding spatial *vector data* (points, lines, polygons) in computers (we will return to this concept next chapter). The `sf` package is an R package for reading, writing, handling, and manipulating simple features in R, implementing the vector (points, lines, polygons) data handling functionality.

Traditionally spatial analysis in R were done using the `sp` package which creates a particular way of storing spatial objects in R. When most packages for spatial data analysis in R and for thematic cartography were first developed `sp` was the only way to work with spatial data in R. There are more than 450 packages that rely on `sp`, making it an important part of the R ecosystem. More recently `sf` is changing the way that R does spatial analysis. This package provides a new way of storing spatial objects in R and most recent R packages for spatial analysis and cartography are using it as the new default. It is easy to transform `sf` objects into `sp` objects and viceversa, so that those packages that still don't use this new format can be used. But in this book we will emphasise the use of `sf` whenever possible. You can read more about the history of spatial packages and the `sf` package in the first two chapters of Lovelace, Nowosad, and Muenchow (2019).

Features can be thought of as "things" or objects that have a spatial location or extent; they may be physical objects like a building, or social conventions like a political state. Feature geometry refers to the spatial properties (location or extent) of a feature, and can be described by a point, a point set, a linestring, a set of linestrings, a polygon, a set of polygons, or a combination of these. The simple adjective of simple features refers to the property that linestrings and polygons are built from points connected by straight line segments. Features typically also have other properties (temporal properties, color, name, measured quantity), which are called feature attributes. For more detailed insight we recommend reading Pebesma (2018).

Let's get started with making some maps using `sf`. First, make sure you install the package, and then load with `library()` function. We know that we have two columns one for Longitude and one for Latitude, which pinpoint each crime event to a specific point, close to where it happened. Not *quite* where it happened, as the data are anonymised (more on this later), but for our purposes here, we can assume this is the location of the crime. To map these points, we can transform our ordinary dataframe into a simple features object.

To do so, we can use the `st_as_sf()` function from `sf`, into which we need to specify what we are to transform (our dataframe), where the spatial data can be found (our columns which hold the latitude and longitude information), and also what coordinate reference system the object has (see above our discussion about projections and coordinate reference systems).

Latitude longitude coordinates specify location on the **WGS 84** CRS. We can tell R that this is our CRS of choice by including its EPSG identifier (see https://en.wikipedia.org/wiki/EPSG_Geodetic_Parameter_Dataset) as a parameter in our function. It is handy to know the more common EPSG identifiers. For example, for WGS84 the EPSG identifier is *4326*. For British National Grid, the identifier is *27700*.

Putting it all together in practice, we can create a simple features object from our dataframe using the latitude and longitude columns:

```
crimes_sf <- st_as_sf(crimes,                                    #dataframe
                      coords = c("longitude", "latitude"),       #columns with coordinates
                      crs = 4326)                                #crs is WGS84
```

We can see that this is now a simple features object using the generic`class()` function the we see the result "sf":

```
class(crimes_sf)
```

You might also notice something else that is different between "crimes" and "crimes_sf." Have a look at the dimension (hint: look in your 'Environment' tab). In the `sf` object you will see that the information provided by the longitude and latitude variables have been "merged" into a new variable called *geometry*, that `sf` uses to store the kind of object we have (a point in this case) and where to locate it.

1.5.2 A brief intro to ggplot2 and the grammar of graphics

Now that we have this `sf` object, how can we map it? We mentioned before about the graphical package `ggplot2`. We can use this, and its syntax, in order to map spatial data using the `geom_sf()` geometry.

First, a quick refresher on `ggplot2` and the grammar of graphics. The grammar of graphics upon which this package is based on defines various components of a graphic. Some of the most important are:

-**The data**: For using `ggplot2` the data has to be stored as a data frame or tibble (`sf` objects are of class tibble).

-**The geoms**: They describe the objects that represent the data (e.g., points, lines, polygons, etc..). This is what gets drawn. And you can have various different types layered over each other in the same visualisation.

-**The aesthetics**: They describe the visual characteristics that represent data (e.g., position, size, colour, shape, transparency).

-**Facets**: They describe how data is split into subsets and displayed as multiple small graphs.

-**Stats**: They describe statistical transformations that typically summarise data.

Let's take it one step at the time.

Essentially the philosophy behind this is that all graphics are made up of layers. You can build every graph from the same few components: a data set, a set of geoms—visual marks that represent data points, and a coordinate system.

Take this example (all taken from Wickham (2010))

[[[NEED TO REPLACE THESE FIGURES]]]

You have a table such as:

Table 3. Simple dataset with variables mapped into aesthetic space.

x	*y*	Shape
25	11	circle
0	0	circle
75	53	square
200	300	square

You then want to plot this. To do so, you want to create a plot that combines the following layers:

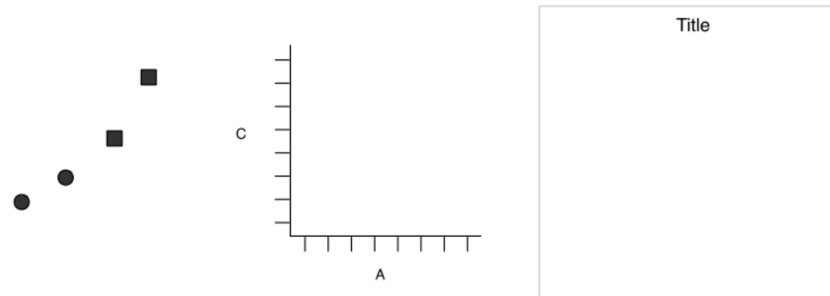


Figure 1. Graphics objects produced by (from left to right): geometric objects, scales and coordinate system, plot annotations.

This will result in a final plot:

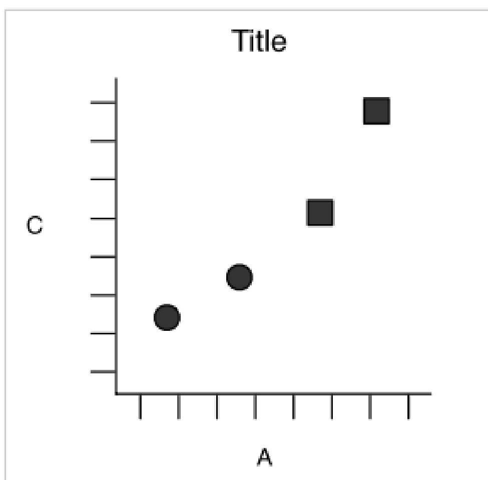
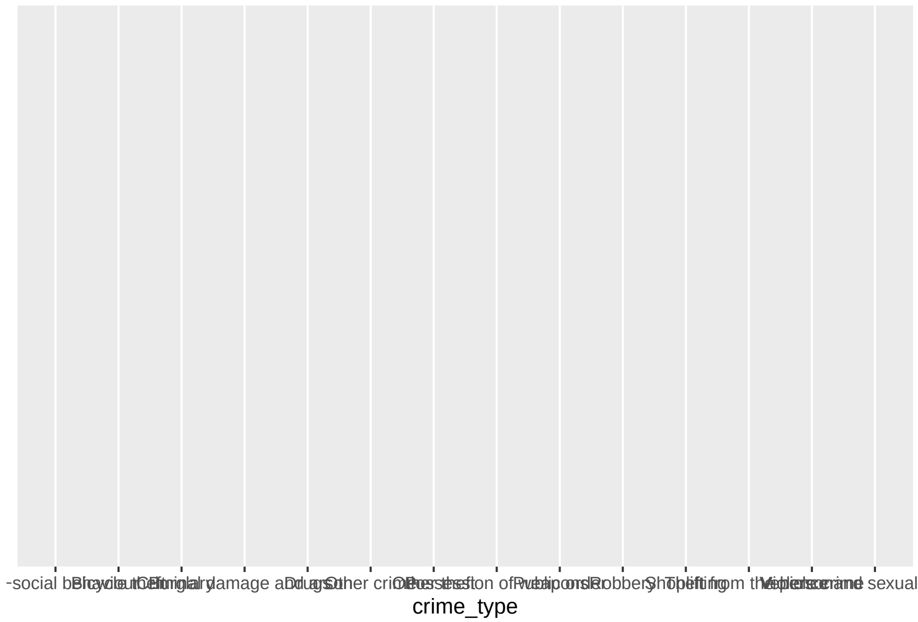


Figure 2. The final graphic, produced by combining the pieces in Figure 1.

Taking our crime data as an example, we would build up our plot as follows:

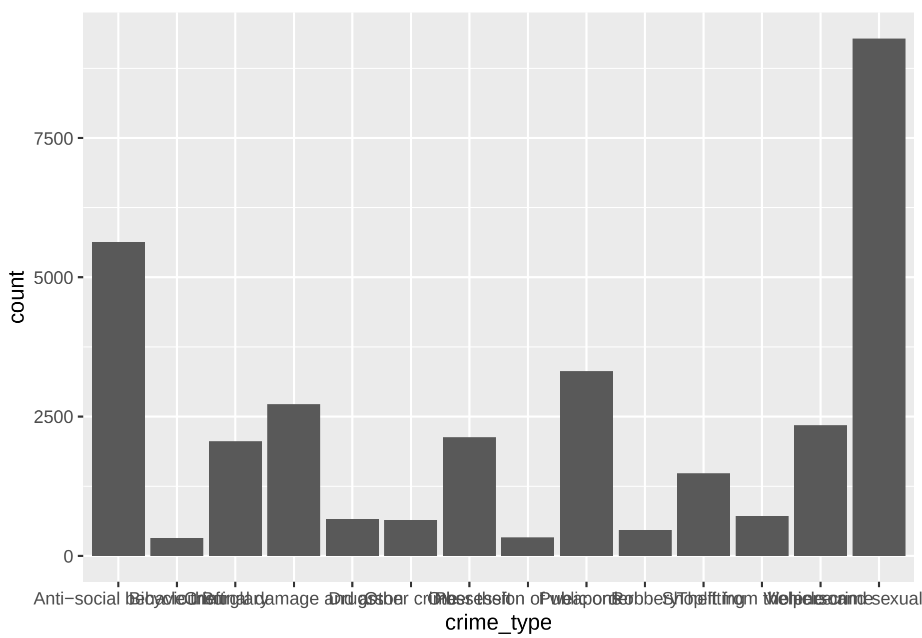
Data:

```
ggplot(crimes, aes(x = crime_type))
```



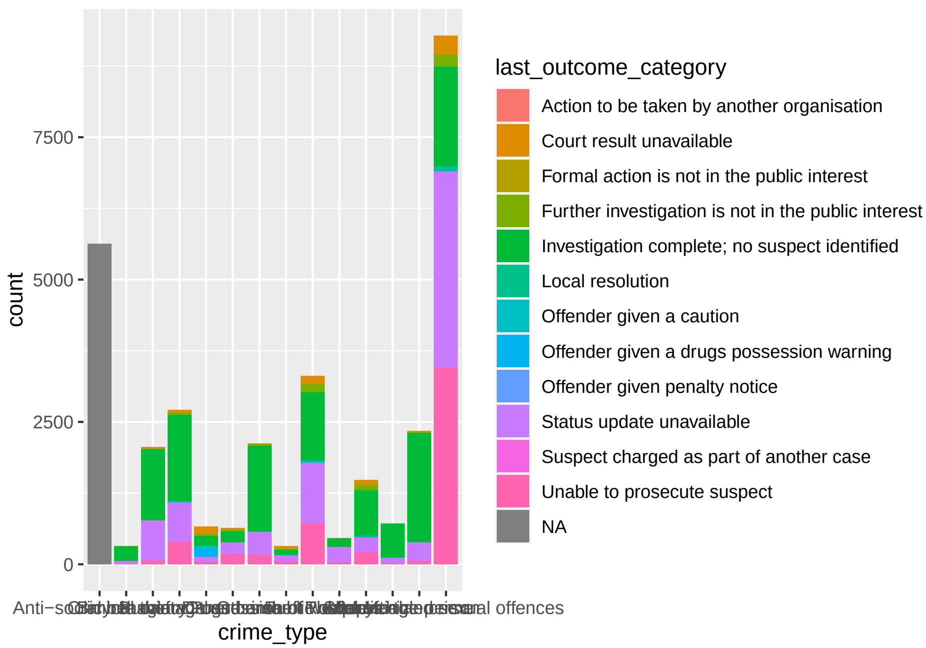
Geometry:

```
ggplot(crimes, aes(x = crime_type)) + geom_bar()
```



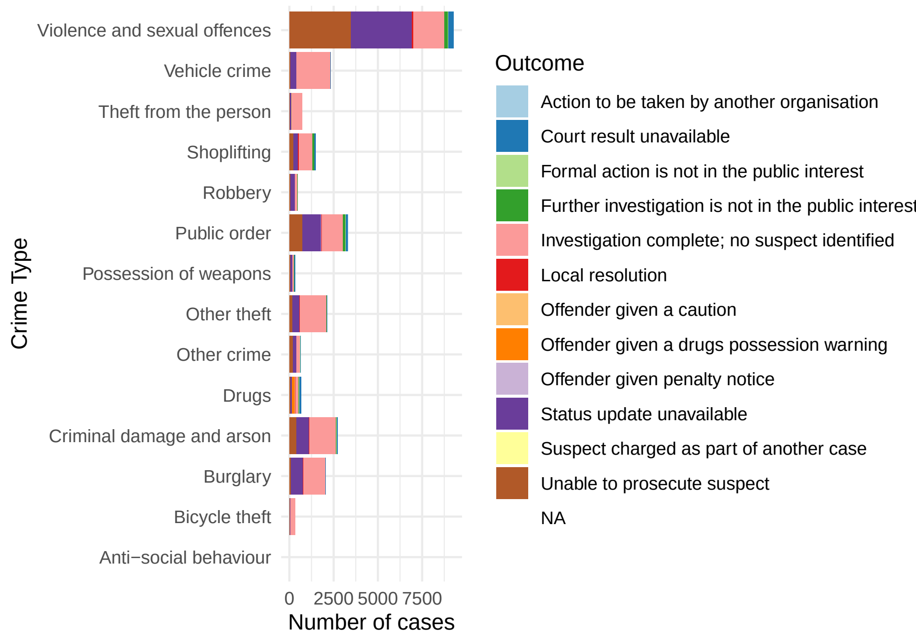
Aesthetics:

```
ggplot(crimes, aes(x = crime_type, fill = last_outcome_category)) + geom_bar()
```



And then you could add any facets (for example if we had more than one month of data) or any statistics (for example error bars) with facet and stats layers. One more thing I do want to show is tidying up your chart, you can add theme and clean up your labels and titles and colour scheme.

```
ggplot(crimes, aes(x = crime_type, fill = last_outcome_category)) + geom_bar() +
    coord_flip() + theme_minimal() + xlab("Crime Type") + ylab("Number of cases") +
    scale_fill_brewer(type = "qual", palette = 3, name = "Outcome")
```

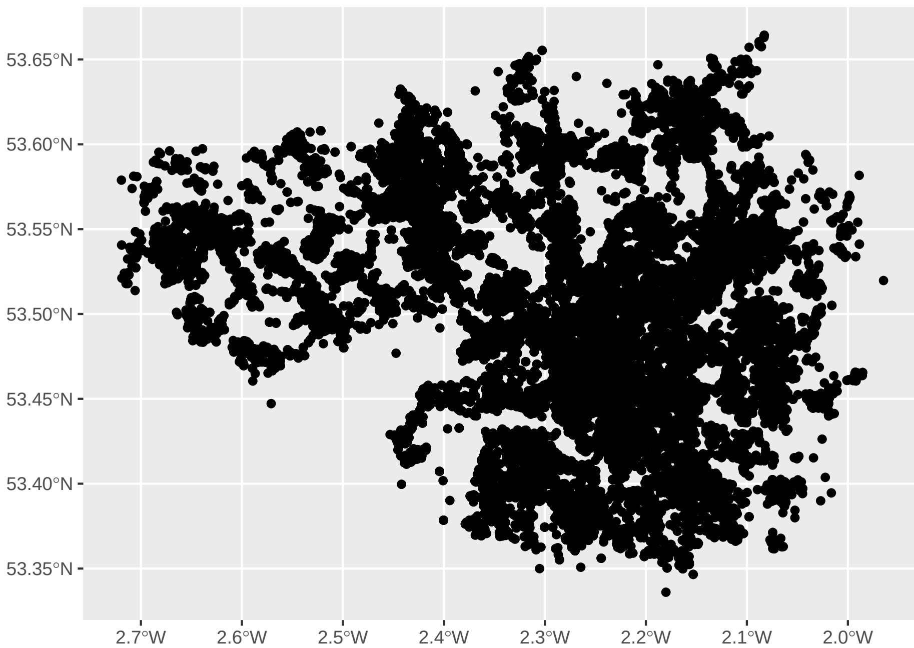


This is not the greatest graph you'll ever see, but it illustrates the process. Do read up on `ggplot2` for example in Wickham and Grolemund (2017).

1.5.3 Using ggplot2 for mapping crime event locations as points

So how can we use this for spatial data? We can use the `geom_sf()` function to do so. Using `geom_sf` is slightly different to other geometries, for example how we used `geom_bar()` above. First we initiate the plot with the `ggplot()` function but don't include the data in there. Instead, it is in the geometry where we add the data. And second we don't need to specify the mapping of x and y. Since this is in the geometry column of our spatial object. Like so:

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



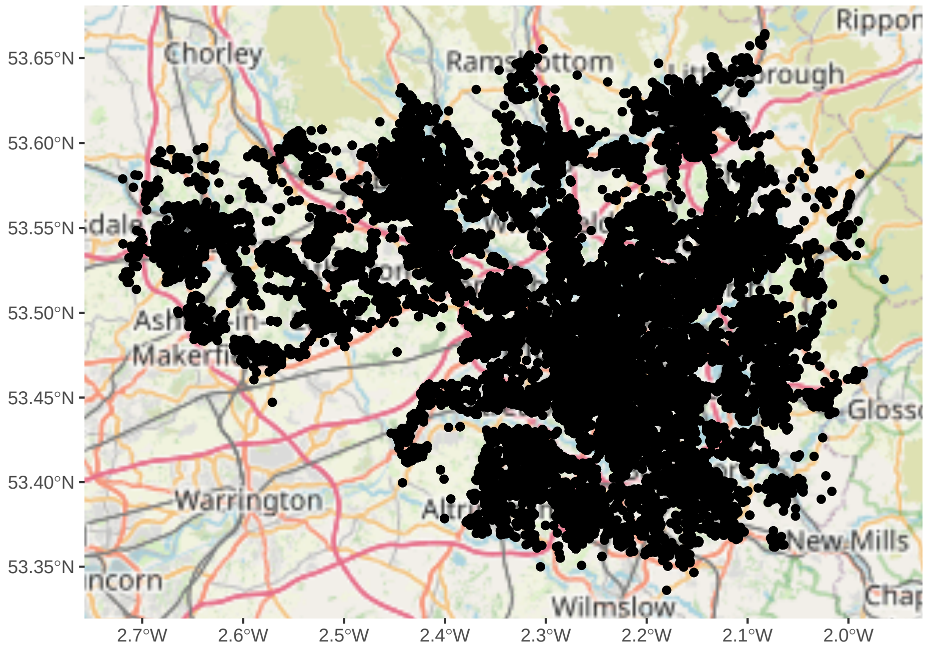
And here we have a map of each point in our data set, each recorded crime in June 2019 in Greater Manchester.

Would you call this a map though? While it is presenting spatial data, there is not a lot of meaning being communicated. Point maps generally can be messy and their uses are specific to certain situations and cases, usually when you have fewer points, but here, these points are especially devoid of any meaning, as they are floating in a graph grid. So let's give it a **basemap**.

We can do this by adding a layer to our graph object. Specifically we will use the `annotation_map_tile()` from the `ggspatial` package. This provides us with a static Open Street Map layer behind our data, giving it (some) more context. Remember to load the package (and install if you haven't already). And then use the `annotation_map_tile()` function, making sure to place it before the `geom_sf` points layer, so the background map is placed first, and the points on top of that:

```
ggplot() +
 annotation_map_tile() +
  geom_sf(data = crimes_sf)
```

```
## Zoom: 9
```



So what you see above behind the points is what we can call a **basemap**. The term basemap is seen often in GIS and refers to a collection of GIS data and/or orthorectified imagery that form the background setting for a map. The function of the basemap is to provide background detail necessary to orient the location of the map. Basemaps also add to the aesthetic appeal of a map. **Basemaps** are essentially reference maps that may give us context and help with the interpretation. You can see above that you are seeing the *Open Street Map* Basemap. This is one option but there are others.

Let's leave out points for now, and move on to how we might map our lines and polygons.

1.6 From mapping crime events as points to mapping number of crimes in an area

What about our other two columns, location, and LSOAs? Well to put these on the map, we need a geometry representation of them. We need boundary data representing the areas we want to map put. We will learn in this section where you may find these, how to download them, turn them into sf objects, and how to link your data to them to be able to map them.

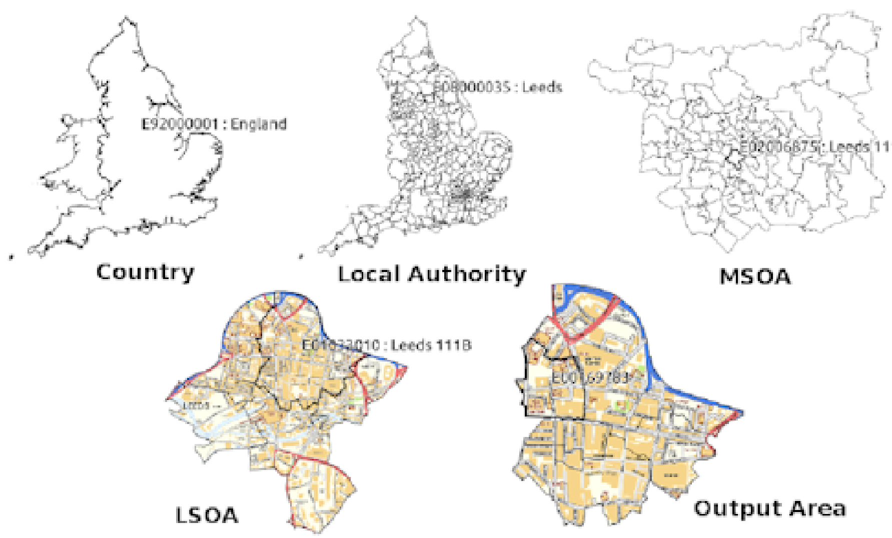
1.6.1 Finding boundary data for areas: using shapefiles

In this section you are going to learn how you take one of the most popular data formats for spatial objects, the **shapefile**, and read it into R. The shapefile was introduced by ESRI, the developers and vendors or ArcGIS. And although many other formats have developed since and ESRI no longer holds the same market position it once occupied (though they're still the player to beat), shapefiles continue to be one of the most popular formats you will encounter in your work. You can read more about shapefiles in Wikipedia (https://en.wikipedia.org/wiki/Shapefile).

We are going to learn here how to obtain shapefiles for British census geographies. For this activity we will focus on the polygon (the LSOA) rather than the lines of the streets, but the logic is more or less the same.

Census boundary data are a digitised representation of the underlying geography of the census. Census Geography is often used in research and spatial analysis because it is divided into units based on population counts, created to form comparable units, rather than other administrative boundaries such as wards or police force areas. However depending on your research question and the context for your analysis, you might be using different units.

The hierarchy of the census geographies in the UK goes from Country to Local Authority to Middle Layer Super Output Area (MSOA) to Lower Layer Super Output Area (LSOA) to Output Area (in other countries you have similar levels in the census hierarchies):



Here we will get some boundaries for Manchester. Let's use the LSOA level, so that we can link back to our crime data easily. These are geographical regions designed to be more stable over time and consistent in size than existing administrative and political boundaries. LSOAs comprise, on average, 600 households that are combined on the basis of spatial proximity and homogeneity of dwelling type and tenure.

To get some boundary data, you can use the UK Data Service website. There is a simple Boundary Data Selector tool which you could use. When you get to the link, you will see on the top there is some notification to help you with the boundary data selector. If in the future you are looking for UK boundary data and you are feeling unsure at any point, feel free to click on that note "**How to use Boundary Data Selector**" which will help to guide you. In other countries you have similar data repositories where you can get the relevant census geographies (see Appendix B for details) and there are also some websites that aim to provide this data for all countries (http://www.diva-gis.org/gdata). We have already compiled all data we used in this book in our own website, so all you need to do is use the code we provide to download and read this data instead.

```
##Downloading a zip file in R requires several steps
#create placeholder for the url address
fileurl <-"https://github.com/maczokni/crime_mapping/raw/master/data/manchester_lsoa_2
# create a temporary directory
td <- tempdir()
# create the placeholder file in the temporary directory we created
tf <- tempfile(tmpdir=td, fileext=".zip")
# download into the placeholder file
download.file(fileurl, tf)
# get the name of files in the zip archive
fname <- unzip(tf, list=TRUE)$Name
fname
```

So you can see immediately that there are some documentations around the usage of this shapefile, in the "README.txt" and the "TermsAndConditionshtml" files. They contain information about how you can use this map. For example, all your maps will have to mention where you got all the data from. So since you got this boundary data from the UKDS, you will have to note the following:

"Contains National Statistics data © Crown copyright and database right [year] Contains OS data © Crown copyright [and database right] (year)"

You can read more about this in the terms and conditions document once we unzip the file.

But then you will also notice that there are 4 files with the same name "england_lsoa_2011." **It is important that you keep all these files in the same location as each other!** They all contain different bits of information about your shapefile (and they are all needed):

- .shp — shape format; the feature geometry itself - this is what you see on the map
- .shx — shape index format; a positional index of the feature geometry to allow seeking forwards and backwards quickly
- .dbf — attribute format; columnar attributes for each shape, in dBase IV format.
- .prj — projection format; the coordinate system and projection information, a plain text file describing the projection using well-known text format

Sometimes there might be more files associated with your shapefile as well, but we will not cover them here. So unlike when you work with spreadsheets and data in tabular form, which typically is just all included in one file; when you work with shapefiles, you have to live with the required information living in separate files that need to be stored together. So, being tidy and organised is even more important when you carry out projects that involve spatial data.

Let´s now unzip all these files:

```
#First you need to decide where you want to place the folders. In my machine I will use the follo
outdir <- "C:/crime_mapping/data/BoundaryData"
#Then you can unzip the file to this folder
unzip(tf, exdir = outdir)
```

1.6.2 Reading shapefiles into R

To read in your data into R, you will need to know the path to where you have saved it. Ideally this will be in your data folder in your project directory.

Let's create an object and assign it our shapefile's name:

```
# Remember to use the appropriate pathfile in your case
shp_name <- "data/BoundaryData/england_lsoa_2011.shp"
```

Make sure that this is saved in your working directory, and you have set your working directory.

Now use the `sf::st_read()` function to read in the shapefile:

```
manchester_lsoa <- st_read(shp_name)
```

```
## Reading layer `england_lsoa_2011' from data source `/Users/reka/Desktop/crime_mapping/crime_ma
## Simple feature collection with 282 features and 3 fields
## Geometry type: POLYGON
## Dimension:     XY
```

```
## Bounding box:  xmin: 378833.2 ymin: 382620.6 xmax: 390350.2 ymax: 405357.1
## Projected CRS: OSGB 1936 / British National Grid
```

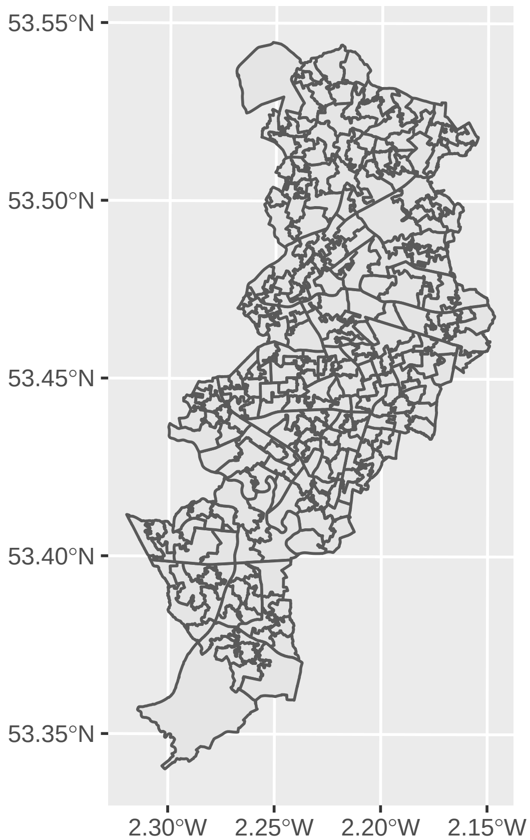
Now you have your spatial data file. Notice how running the function sends to the console some metadata about your data. You have a polygon, with 282 rows, and the CRS is the projected British National Grid. You can have a look at what sort of data it contains, the same way you would view a dataframe, with the `View()` function:

```
View(manchester_lsoa)
```

```
## Rows: 282
## Columns: 4
## $ label    <chr> "E08000003E02001062E01005066", "E08000003E02001092E01005073",~
## $ name     <chr> "Manchester 018E", "Manchester 048C", "Manchester 018A", "Man~
## $ code     <chr> "E01005066", "E01005073", "E01005061", "E01005062", "E0100506~
## $ geometry <POLYGON [m]> POLYGON ((384850 397432, 38..., POLYGON ((382221.1 38~
```

And of course, since it's spatial data, you can map it using the `geom_sf()` function, as we did with our points:

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



Great, we now have an outline of the LSOAs in Manchester. Notice how the shape is different to that of the points in our crime data since here we only obtained the data for the city of Manchester rather than for the whole metropolitan area - and which includes other municipalities aside from Manchester city.

1.6.3 Data wrangling with dplyr

In order to map crimes to LSOAs we might want to take a step back and think about unit of analysis at which our data are collected. In our original dataframe of crimes, we saw that each crime incident is one row. So the unit of analysis is each crime. Since we were looking to map each crime at the location it happened, we used the latitude and longitude supplied with each one, and this supplied a geometry each for each crime type. However, when we are looking to map our data to LSOA level, we need to match the crime data to the geometry we wish to display.

Have a look at the "manchester_lsoa" object we mapped above. How many rows (observations) does it have? You can check this by looking in the Environment pane, or by using the `nrow()` function.

```
nrow(manchester_lsoa)
```

You can see this has 282 rows. This means we have geometries for 282 LSOAs. On the other hand, our crimes dataframe has 32058 rows, one for each crime (observation). So how can we match these up? The answer lies in thinking about what it is that our map using LSOAs as our unit of analysis will be able to tell us. Think of other maps of areas - what are they usually telling you? Usually we expect to see crimes per neighbourhood - something like this. So our unit of analysis needs to be LSOA, and for each one we need to know how many crimes occurred in that area.

To achieve this, we will wrangle our data using functions from the `dplyr` package. This is a package for conducting all sorts of operations with data frames. We are not going to cover the full functionality of `dplyr` (which you can consult in the `dplyr` vignette (https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html)), but we are going to cover three different very useful elements of dplyr: the `select` function, the `group_by` function, and the piping operator.

The `select()` function provides you with a simple way of subsetting columns from a data frame. So, say we just want to use one variable, "lsoa_code," from the "crimes" dataframe and store it in a new object we could write the following code. This variable tell us the LSOA in which the crime took place and it is essential if we want to group our crimes by LSOA (by using this standard method of merging information from datasets):

```
new_object <- select(crimes, lsoa_code)
```

We can also use the `group_by()` function for performing group operations. Essentially this function ask R to group cases within categories and then do something with those grouped cases. So, say, we want to count the number of cases within each LSOA, we could use the following code:

```
#First we group the cases by LSOA code and stored this organised data into a new objec
grouped_crimes <- group_by(new_object, lsoa_code)

#Then we could count the number of cases within each category and use the summarise fu
summarise(grouped_crimes, count = n())

#We could infact create a new dataframe with these results

crime_per_LSOA <- summarise(grouped_crimes, count = n())
```

As you can see we can do what we wanted, create a new dataframe with the required info, but if we do this we are creating many objects that we don't need, one at each step. Instead there is a more efficient way of doing this, without so many intermediate steps clogging up our environment with unnecessary objects. That's where the piping operator comes handy. The piping operator is written like `%>%` and it can be read as "and then." Look at the code below:

```
#First we say create a new object called crime_per_lsoa, and then select only the LSOA.code colum

crimes_per_lsoa <- crimes %>%
  group_by(lsoa_code) %>%
  summarise(count=n())
```

Essentially we obtain the same results but with more streamlined and elegant code, and not needing additional objects in our environment.

And now we have a new object, "crimes_per_lsoa" if we have a look at this one, we can now see what each row represents one LSOA, and next to it we have a variable for the number of crimes from each area. We created a new dataframe from a frequency table, and as each row of the crimes data was one crime, the frequency table tells us the number of crimes which occurred in each LSOA.

Those of you playing close attention might note that there are still more observations in this dataframe (1671) than in the "manchester_lsoas" one (282). Again, this is because "crimes_per_lsoa" also includes data from census areas in municipalities of the metropolitan area of Greater of Manchester other than Manchester city.

1.6.4 Join data to sf object

Our next task is to link our crimes data to our `sf` spatial object to help us map this. Notice anything similar between the data from the shapefile and the frequency table data we just created? Do they share a column?

Yes! You might notice that the "lsoa_code" field in the crimes data matches the values in the "code" field in the spatial data. In theory we could join these two data tables.

So how do we do this? Well what you can do is to link one data set with another. Data linking is used to bring together information from different sources in order to create a new, richer dataset. This involves identifying and combining information from corresponding records on each of the different source datasets. The records in the resulting linked dataset contain some data from each of the source datasets. Most linking techniques combine records from different datasets if they refer to the same entity (an entity may be a person, organisation, household or even a geographic region.)

You can merge (combine) rows from one table into another just by pasting them in the first empty cells below the target table—the table grows in size to include the new rows. And if the rows in both tables match up, you can merge columns from one table with another by pasting them in the first empty cells to the right of the table—again, the table grows, this time to include the new columns.

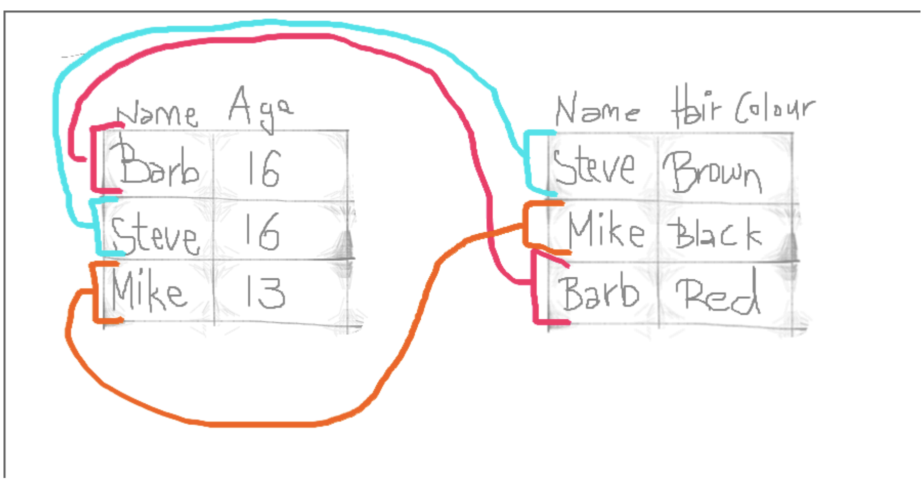
Merging rows is pretty straightforward, but merging columns can be tricky if the rows of one table don't always line up with the rows in the other table. By using `left_join()` from the `dplyr` package, you can avoid some of the alignment problems.

`left_join()` will return all rows from x, and all columns from x and y. Rows in x with no match in y will have NA values in the new columns. If there are multiple matches between x and y, all combinations of the matches are returned.

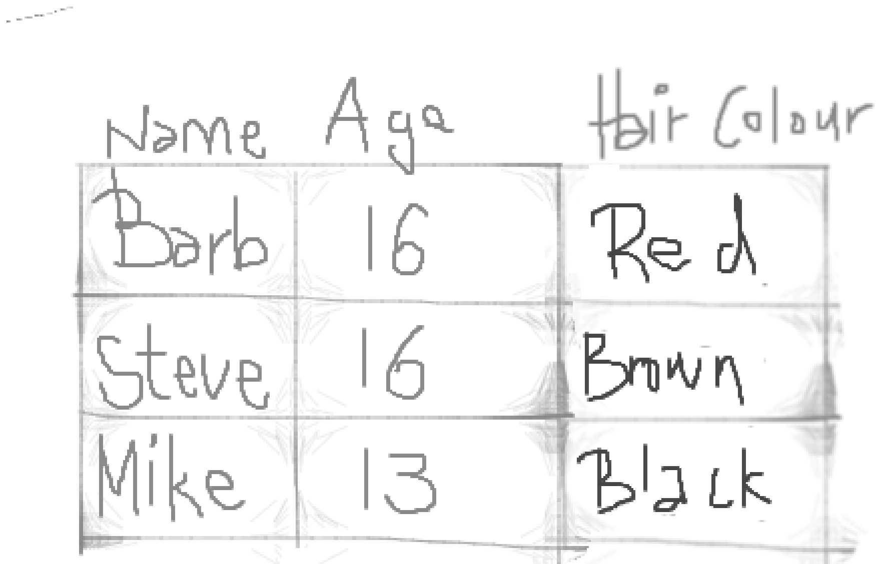
So we've already identified that both our crimes data, and the spatial data contain a column with matching values, the codes for the LSOA that each row represents.

You need a unique identifier to be present for each row in all the data sets that you wish to join. This is how R knows what values belong to what row. What you are doing is matching each value from one table to the next, using this unique identified column, that exists in both tables. For example, let's say we have two data sets from some people in Hawkins, Indiana. In one data set we collected information about their age. In another one, we collected information about their hair colour. If we collected some information that is unique to each observation, and this is the *same* in both sets of data, for example their names, then we can link them up, based on this information. Something like this:

[[[REPLACE IMAGES]]]



And by doing so, we produce a final table that contains all values, lined up *correctly* for each individual observation, like this:



Name	Age	Hair Colour
Barb	16	Red
Steve	16	Brown
Mike	13	Black

This is all we are doing, when merging tables, is we are making use that we line up the correct value for all the variables, for all our observations.

Why are we using *left* join though? There is a whole family of join functions as part of dplyr (http://stat545.com/bit001_dplyr-cheatsheet.html) which join data sets. There is also a `right_join()`, and an `inner_join()` and an `outer_join()` and a `full_join()`. But here we use `left_join()`, because that way we keep all the rows in x (the left-hand side dataframe), and join to it all the matched columns in y (the right-hand side dataframe).

So let's join the crimes data to the spatial data, using `left_join()`. We have to tell the function what are the dataframes we want to join, as well as the names of the columns that contain the matching values in each one. This is "code" in the "manchester_lsoa" dataframe and "lsoa_code" in the "crimes_per_lsoa" dataframe. Like so:

```
manchester_lsoa <- left_join(manchester_lsoa, crimes_per_lsoa, by = c("code"="lsoa_code"))
```

Now if you have a look at the data again, you will see that the column of number of crimes (n) has been added on.

You may not want to have to go through this process all the time you want to work with this data. One thing you could do is to save the "manchester_lsoa" object as a physical file in your machine. You can use the `st_write()` function from the `sf` package to do this. If we want to write into a shapefile format we would do as shown below. Make sure you save this file, for we will come back to it in subsequent chapters.

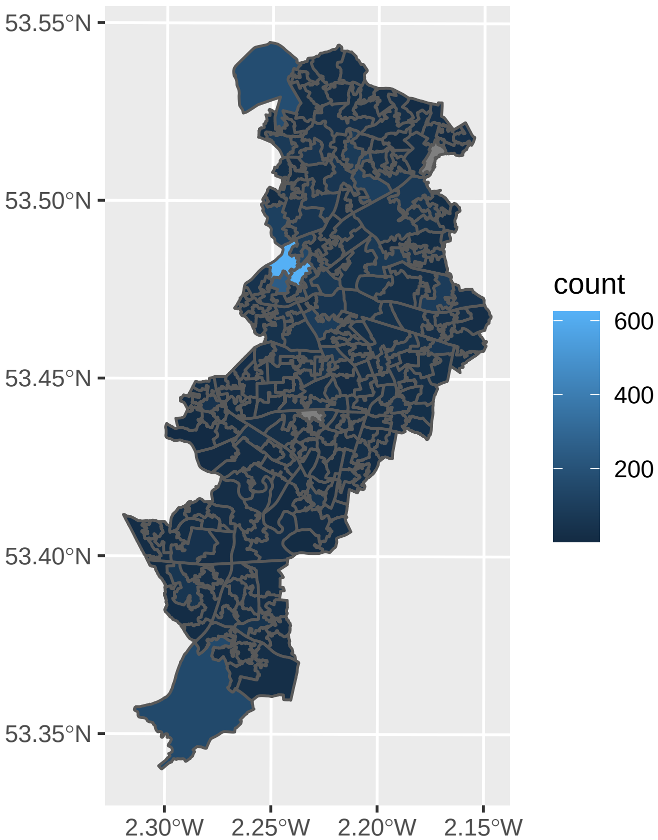
```
st_write(manchester_lsoa, "data/BoundaryData/manchester_crime_lsoa.shp")
```

1.6.5 Mapping our data at polygon level

Now that we have joined the crimes data to the geometry, you can use this to make our map!

Remember our original empty map? Well now, since we have the column (variable) for number of crimes here, we can use that to share the polygons based on how many crimes there are in each LSOA. We can do this by specifying the `fill=` parameter of the `geom_sf` function.

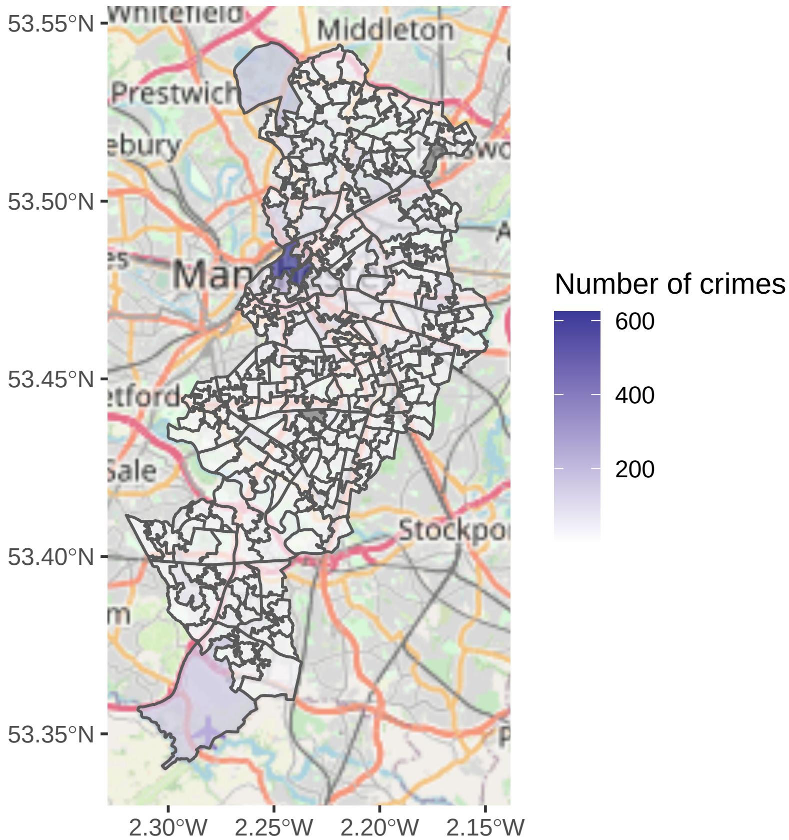
```
ggplot() +
geom_sf(data = manchester_lsoa, aes(fill = count))
```



We can add a basemap, and adjust the colour scheme, and even the opacity to see under our shape files.

```
ggplot() +
  annotation_map_tile() +  # add basemap
geom_sf(data = manchester_lsoa, aes(fill = count), alpha = 0.7) + # alpha sets the opacity
  scale_fill_gradient2(name ="Number of crimes") #use scale_fill_gradient2() for a different pale
```

```
## Zoom: 10
```



In subsequent chapters we will play around with other packages in R that you can use to produce this kind of maps and gradually we will discuss the kind of choices you can make in order to select the adequate type of representation for the type of spatial data and question you have, and the aesthethic choices that are adequate depending on the purposes and medium in which you will publish your map.

1.7 Further reading

In this chapter we had a play around with some regular old crime data and discovered how we can use the `sf` package in R to assign it a geometry (both at point and polygon level), and how that can help us visualise our results. We covered some very important concepts such as projections and coordinate reference systems, and we had a play at acquiring shapefiles which can help us visualise our data. We had a think about units of analysis, and how that will affect how we visualise our data. In the next chapter we will spend a bit of more time discussing how to make good choices when producing maps.

There are a number of introductory texts to geographic information systems and analysis that provide adequate background to some of the key concepts we introduce in this chapter (O´Sullivan and Unwin (2010), Bolstad (2019)). The report by Harries (1999) produced for the National Institute of Justice still provides a good general introduction for basic ideas around crime mapping and CAN be accessed for free online. Chapter 3 of Spatial and Approach (2020) offers invaluable observations on the nature of spatial and spatial-temporal attribute data, often using examples from crime research. From a more domain-knowledge point of view, the early chapters of Santos (2013) and Chainey and Rattcliffe (2005) set the stage for the use of GIS as part of the crime analysis process, whereas Bruinsma and Johnson (2018) edited handbook provides an excellent introduction to environmental criminology (that provides the theoretical and empirical backbone to spatial analysis of crime). Finally, Healy (2019) provides a general introduction to data visualisation with R using the `ggplot2` package. Although Healy´s text is not just about mapping, it offers a very practical and helpful introduction to using `ggplot2` where you can learn how to further customise your maps and other charts.

This is a book about maps and spatial analysis, but clearly one of the first questions you need to ask yourself is whether producing a map is the right answer to your question. Just because your data is spatial doesn´t mean you need a map for every question you pose to this data. There may be other forms of data visualisation that are more appropriate for exploring and summarising the story you want to tell with your data. If you are uncertain about whether you need a map or other kind of plot books such as Cairo (2016), Camoes (2016), or Schwabish (2021) provide useful guidance.

Chapter 2

Chapter 2: Basic geospatial operations in R

2.1 Introduction

In this chapter we get your hands dirty with **spatial manipulation of data**. Thus far, our data manipulation exercises (using `dplyr`) were such that you might be familiar with, from your earlier exposures to data analysis. For example, linking datasets using a common column is a task which you can perform on spatial or non-spatial data. These are referred to as attribute operations. However today we will explore some exercises in data manipulation which are specific to *spatial* data analysis. We will be learning some key spatial operations, a set of functions that allow you to create new and manipulate spatial data.

The main objectives for this chapter are that by the end you will have:

- met a new format for accessing boundary data, called **geojson**.
- carried out **spatial operations** such as:
  - **subset** points that are within a certain area,
  - created new polygons by generating **buffers** around points,
  - counted the number of points that fall within a polygon (known as **points in polygon**),
  - finding the **nearest feature** in one data set to observations in another data set, and
  - **measured distance** between points in a map.
- made interactive point map with leaflet.
- used **geocoding** methods to translate text fields such as addresses into geographic coordinates.

These are all very useful tools for the spatial crime analyst, and we will hope to demonstrate this by working through an example project.

The packages we will use in this chapter are:

```
# Packages for reading data and data carpentry
library(readr)
library(dplyr)
library(janitor)
library(units)
library(purrr)

# Packages for handling spatial data and for geospatial carpentry
library(sf)
library(tidygeocoder)

# Packages for mapping and visualisation
library(leaflet)
library(RColorBrewer)

# Packages providing accesss to spatial data
library(osmdata)
```

2.2 Exploring the relationship between alcohol outlets and crime

The main example we will work through most of the chapter considers the assumption that licenced premises which serve alcohol are associated with increased crimes. We might have some hypotheses about why this may be.

One theory might be that some of these serve as *crime attractors*.

> Crime attractors are particular places, areas, neighbourhoods, districts which create well-known criminal opportunities to which strongly motivated, intending criminal offenders are attracted because of the known opportunities for particular types of crime. Examples might include bar districts; prostitution areas; drug markets; large shopping malls, particularly those near major public transit exchanges; large, insecure parking lots in business or commercial areas. The intending offender goes to rough bars looking for fights or other kinds of 'action.'

On the other hand, it is possible that these areas are *crime generators*.

> Crime generators are particular areas to which large numbers of people are attracted for reasons unrelated to any particular level of criminal motivation they might have or to any particular crime they might end up committing. Typical examples might include shopping precincts; entertainment districts; office concentrations; or sports stadiums.

To read further in crime attractors vs crime generators turn to the recommended readings by Brantingham and Brantingham (1995) and Newton (2018). There have since been more developments, for example about crime radiators and absorbers as well (watch this risky places lecture from Kate Bowers (n.d.) to learn more!)

It's possible that some licensed premises attract crimes, due to their reputation. However it is also possible that some of them are simply located in areas that are busy, attracts lots of people for lots of reasons, and crimes occur as a result of an abundance of opportunities instead.

In any case, what we want to do is to examine whether certain outlets have more crimes near them than others. We can do this using open data, some R code, and the spatial operations discussed above. We will return to data from Manchester, UK for this example, however as we will be using Open Street Map, you can easily replicate this for any other location where you have point-level crime data.

2.3 Get the data

We will be using three different sources of data in this chapter. First, we will acquire our crime data, which is what we used in the previous chapter, so this should be familiar. Then we will meet the new format for boundary data, geojson. And finally, we will look at Open Street Map for data on our points of interest.

2.3.1 Reading in crime data

```
crimes <- read_csv("data/2019-06-greater-manchester-street.csv")
```

Notice that in this case the columns are spelled with upper case "L." You should always familiarise yourself with your data set to make sure you are using the relevant column names. You can see just the column names using the `names()` function like so :

```
names(crimes)
```

```
##  [1] "Crime ID"              "Month"                 "Reported by"
##  [4] "Falls within"          "Longitude"             "Latitude"
##  [7] "Location"              "LSOA code"             "LSOA name"
## [10] "Crime type"            "Last outcome category" "Context"
```

So messy! Let's use our handy helpful `clean_names()` function from the `janitor` package:

```
crimes <- crimes %>% clean_names()
names(crimes)
```

```
##  [1] "crime_id"              "month"                 "reported_by"
##  [4] "falls_within"          "longitude"             "latitude"
##  [7] "location"              "lsoa_code"             "lsoa_name"
## [10] "crime_type"            "last_outcome_category" "context"
```

Much better! Now let's get some boundary data for Manchester.

2.3.2 Meet a new format: geojson

GeoJSON is an open standard format designed for representing simple geographical features, along with their non-spatial attributes. It is based on JSON, the JavaScript Object Notation. It is a format for encoding a variety of geographic data structures and is the most common format for geographical representation in the web. Unlike ESRI shapefiles, with GeoJSON data everything is stored in a single file.

Geometries are shapes. All simple geometries in GeoJSON consist of a type and a collection of coordinates. The features include points (therefore addresses and locations), line strings (therefore streets, highways and boundaries), polygons (countries, provinces, tracts of land), and multi-part collections of these types. GeoJSON features need not represent entities of the physical world only; mobile routing and navigation apps, for example, might describe their service coverage using GeoJSON.

To tinker with GeoJSON and see how it relates to geographical features, try geojson.io, a tool that shows code and visual representation in two panes.

Let's read in a geoJSON spatial file, again from the web. This particular geojson represents the wards of Greater Manchester.

```
manchester_ward <- st_read("data/wards.geojson")
```

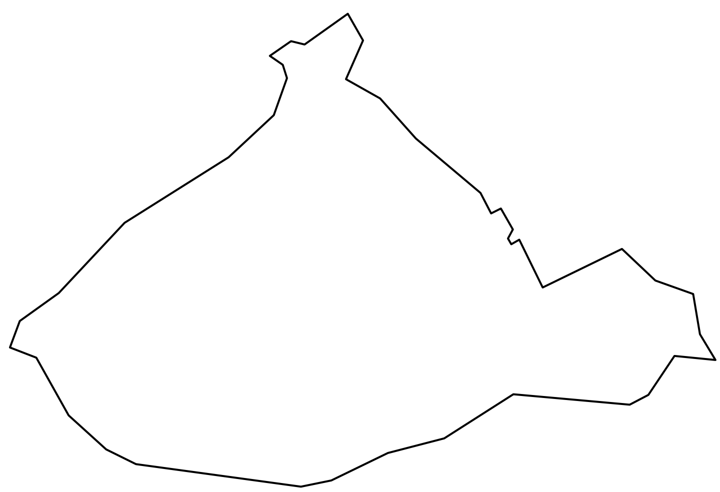
```
## Reading layer `wards' from data source `/Users/reka/Desktop/crime_mapping/crime_mapping/data/w
## Simple feature collection with 215 features and 12 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 351664 ymin: 381168.6 xmax: 406087.5 ymax: 421039.8
## Projected CRS: OSGB 1936 / British National Grid
```

Let's select only the city centre ward, using the `filter()` function from dplyr

```
city_centre <- manchester_ward %>%
  filter(wd16nm == "City Centre")
```

Let's see how this looks, using the `plot()` function:

```
plot(st_geometry(city_centre))
```



Now we could use this to make sure that our points are in fact only licensed premises in the city centre. This will be your first spatial operation. Excited? Let's do this!

2.3.3 Open Street Map and points of interest

To map our licenced premises we will be accessing data from Open Street Map, a database of geospatial information built by a community of mappers, enthusiasts and members of the public, who contribute and maintain data about all sorts of environmental features, such as roads, green spaces, restaurants and railway stations. You can see all about open street map on their online mapping platform (https://www.openstreetmap.org/). One feature of Open Street Map, unlike Google Map, is that the underlying data is openly available for download *for free*. In R, we can take advantage of a package written specifically for querying Open Street Map's API [[[REF SAM CHAPTER HERE]]], called `osmdata`.

If we load the package `osmdata` we can use its functions to query the Open Street Map API. To find out more about the capabilities of this package, see the package documentation and the associated vignette online: https://cran.r-project.org/web/packages/osmdata/vignettes/osmdata.html. While this is outside the scope of our chapter here, you may want to explore `osmdata` more, as it is an international database, and has lots of data may come in handy for research and analysis.

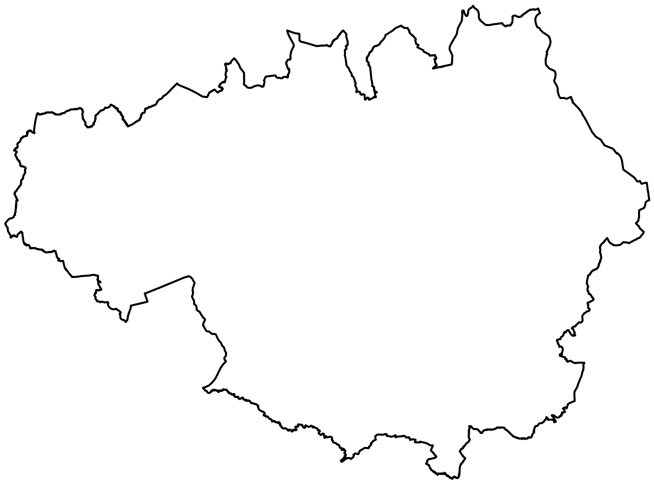
Here we focus specifically on Manchester. To retrieve data for a specific area, we must create a **bounding box**. You can think of the bounding box as a box drawn around the area that we are interested in (in this case, Manchester, UK) which tells the Open Street Map API that we want everything *inside* the box, but nothing *outside* the box.

So, how can we name a bounding box specification to define the study region? One way to do this is through a search term. Here, we want to select Greater Manchester, so we can use the search term "greater manchester united kingdom" within the `getbb()` function (stands for *get bounding box*). Using this function we can also specify what format we want the data to be in. In this case, we want a spatial object, specifically an `sf` polygon object (from the `sf` package), which we name `bb_sf`.

```
bb_sf <- getbb(place_name = "greater manchester united kingdom", format_out = "sf_polyg
```

We can see what this bounding box looks like by plotting it:

```
plot(st_geometry(bb_sf))
```



We can see the bounding box takes the form of Greater Manchester. We can now use this to query data from the Open Street Map API using the `opq()` function. The function name is short for 'Overpass query,' which is how users can query the Open Street Map API using search criteria.

Besides specifying what area we want to query with our bounding box object(s) in the `opq()` function, we must also define the feature which we want returned. Features in Open Street Map are defined through 'keys' and 'values.' Keys are used to describe a broad category of features (e.g. highway, amenity), and values are more specific descriptions (e.g. cycleway, bar). These are tags which contributors to Open Street Map have defined. A useful way to explore these is by using the comprehensive Open Street Map Wiki page on map features (https://wiki.openstreetmap.org/wiki/Map_Features).

We can select what features we want using the `add_osm_feature()` function, specifying our key as 'amenity' and our value as 'bar.' We also want to specify what sort of object (what class) to get our data into, and as we are still working with spatial data, we stick to the `sf` format, for which the function is `osmdata_sf()`. Here, we specify our bounding box as the `bb_sf` object we created above [^1]. [^1]: If you use the bounding box obtained through `getbb()` one can subsequently trim down the outputs from `add_osm_feature()` using `trim_osmdata()`. For instance, we could add `trim_osmdata(bb_poly = bb_sf)` to our initial query.

```
osm_bar_sf <- opq(bbox = bb_sf) %>%                               # select bounding box
  add_osm_feature(key = 'amenity', value = 'bar') %>% # select features
  osmdata_sf()                                                     # specify class
```

The resulting object `osm_bar_sf` contains lots of information. We can view the contents of the object by simply executing the object name into the **Console**.

```
osm_bar_sf
```

This confirms details like the bounding box coordinates, but also provides information on the features collected from the query. As one might expect, most information relating to bar locations has been recorded using points (i.e. two-dimensional vertices, coordinates) of which we have 576 at the time of writing. We also have around fifty polygons. For now, let's extract the point information.

```
osm_bar_sf <- osm_bar_sf$osm_points
```

We now have an `sf` object with all the bars in our study region mapped by Open Street Map volunteers, along with ~90 variables of auxiliary data, such as characteristics of the bar (e.g.: `brewery`, or `cocktail`) or whether it offers `delivery` as well as address and contact information, amongst many others. Of course, it is up to the volunteers whether they collect all these data, and in many cases, they have not added information (you may see lots of missing values if you look at the data). Given the work relies on voluntaries there are unavoidably some accuracy issues (including how up to date the information may be). Nevertheless, when the details are recorded, they provide rich insight and local knowledge that we may otherwise be unable to obtain.

One column we should consider is the `name` which tells us the name of the bar. There are missing values here as well, and for this example, we will choose to exclude those lines where there is no name included, as we would like at least a little bit of context around our bars. To do this, perform an attribute operation using the`filter()` function:

```
osm_bar_sf <- osm_bar_sf %>% filter(!is.na(name))
```

We are still left with 259 bars in our data set.

2.4 Attribute operations

We've mentioned above that we are using **attribute operations**. These are changes to the data which we make based on manipulation of elements in the attribute table. For example, the use of `filter()` is an attribute operation, because we rely on the data in the attribute table in order to accomplish this task.

For example, let's say we want to focus only on violent crime. To do this, we use the information in the attribute table, namely the values for the `crime_type` variable for each observation (crime) in our data set.

```
crimes <- crimes %>% filter(crime_type == "Violence and sexual offences")
```

With the above, we select only those crimes (rows of the attribute table) where the crime_type variable meets a certain criteria (takes the value of "Violence and sexual offences"). **Spatial operations** on the other hand manipulate the *geometry* part of our data. We rely on the spatial information to accomplish the tasks of interest. In the next section, we will work through some examples of these.

2.5 Spatial operations

Spatial operations are a vital part of geocomputation. Spatial objects can be modified in a multitude of ways based on their location and shape. For a comprehensive overview of spatial operations in R I would recommend chapter 4 of Lovelace, Nowosad, and Muenchow (2019).

> "Spatial operations differ from non-spatial operations in some ways. To illustrate the point, imagine you are researching road safety. Spatial joins can be used to find road speed limits related with administrative zones, even when no zone ID is provided. But this raises the question: should the road completely fall inside a zone for its values to be joined? Or is simply crossing or being within a certain distance sufficent? When posing such questions it becomes apparent that spatial operations differ substantially from attribute operations on data frames: the type of spatial relationship between objects must be considered."

- (Lovelace & Nowosad, 2018)

So you can see we can do exciting spatial operations with our spatial data, which we cannot with the non-spatial stuff.

2.5.1 Reprojecting coordinates

It is important to recall here some of the learning from the previous chapter on map projections and coordinate reference systems. We learned about ways of flattening out the earth, and ways of making sense of what that means for how to be able to point to specific locations in our maps. **Coordinate Reference System** or CRS is this method of how to refer to locations with our data. You might use a *Geographic Coordinate System*, which tells you where your data are located on the surface of the Earth. The most commonly used one (at least by us!) is the **WGS 84**, where we define our locations with latitude and longitude points. The other type is a *Projected Coordinate System* which tells the date how to draw on a flat, 2-dimensional surface (such as a computer screen). In our case here, we will often encounter the **British National Grid** when working with British data. Here our locations are defined with Eastings and Northings.

So why are we talking about this?

> It is important to note that spatial operations that use two spatial objects rely on both objects having the same coordinate reference system

If we are looking to carry out operations that involve two different spatial objects, they need to have the same CRS! Funky weird things happen when this condition is not met, so beware! So how do we know what CRS our spatial objects are? Well the `sf` package contains a handy function called `st_crs()` which let's us check. All you need to pass into the brackets of this function is the name of the object you want to know the CRS of.

So let's check what is the CRS of our crimes:

```
st_crs(crimes)
```

```
## Coordinate Reference System: NA
```

You can see that we get the CRS returned as `NA`. Can you think of why? Have we made this into a spatial object? Or is this merely a dataframe with a latitude and longitude column? The answer is really in the question here. So we need to convert this to a `sf` object, or a spatial object, and make sure that R knows that the latitude and the longitude columns are, in fact, coordinates.

In the `st_as_sf()` function we specify what we are transforming (the name of our dataframe), the column names that have the coordinates in them (longitude and latitude), the CRS we are using (4326 is the code for WGS 84, which is the CRS that uses latitude and longitude coordinates (remember BNG uses Easting and Northing)), and finally *agr*, the attribute-geometry-relationship, specifies for each non-geometry attribute column how it relates to the geometry, and can have one of following values: `"constant"`, `"aggregate"`, `"identity"`. Yhe option `"constant"` is used for attributes that are constant throughout the geometry (e.g. land use), `"aggregate"` where the attribute is an aggregate value over the geometry (e.g. population density or population count), `"identity"` when the attributes uniquely identifies the geometry of particular "thing," such as a building ID or a city name. The default value, NA_agr_, implies we don't know.

```
crimes <- st_as_sf(crimes, coords = c("longitude", "latitude"),
                   crs = 4326, agr = "constant", na.fail = FALSE)
```

Now let's check the CRS of this spatial version of our licensed premises:

```
st_crs(crimes)
```

```
## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["World"],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

We can now see that we have this coordinate system as WGS 84. We need to then make sure that any other spatial object with which we want to perform spatial operations is also in the same CRS.

So let's look at our city centre ward boundary file:

```
st_crs(city_centre)
```

```
## Coordinate Reference System:
##   User input: OSGB 1936 / British National Grid
##   wkt:
## PROJCRS["OSGB 1936 / British National Grid",
##     BASEGEOGCRS["OSGB 1936",
##         DATUM["OSGB 1936",
##             ELLIPSOID["Airy 1830",6377563.396,299.3249646,
```

```
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4277]],
##     CONVERSION["British National Grid",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
##         PARAMETER["Latitude of natural origin",49,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",-2,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["Scale factor at natural origin",0.9996012717,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",400000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",-100000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["(N)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["UK - Britain and UKCS 49°46'N to 61°01'N, 7°33'W to 3°33'E"],
##         BBOX[49.75,-9.2,61.14,2.88]],
##     ID["EPSG",27700]]
```

We see that this is in fact in a projected coordinate system, namely the British National Grid we mentioned. To make them align, we can **re-project** this object into the *WGS84* geographic coordinate system. To do this, we can use the `st_transform()` function.

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

Now we can check the projection again:

```
st_crs(city_centre)
```

```
## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["World"],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

And we can also check whether the CRS of the two objects match:

```
st_crs(crimes) == st_crs(city_centre)
```

```
## [1] TRUE
```

It is true! Finally, to check our bar data from Open Street Map:

```
st_crs(osm_bar_sf)
```

```
## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
```

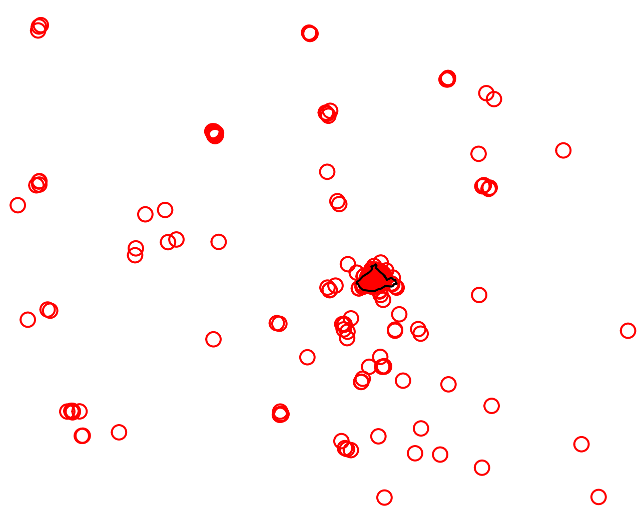
```
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["World"],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

Also in WGS84. We can now move on to carry out some spatial operations!

2.5.2 Subsetting points

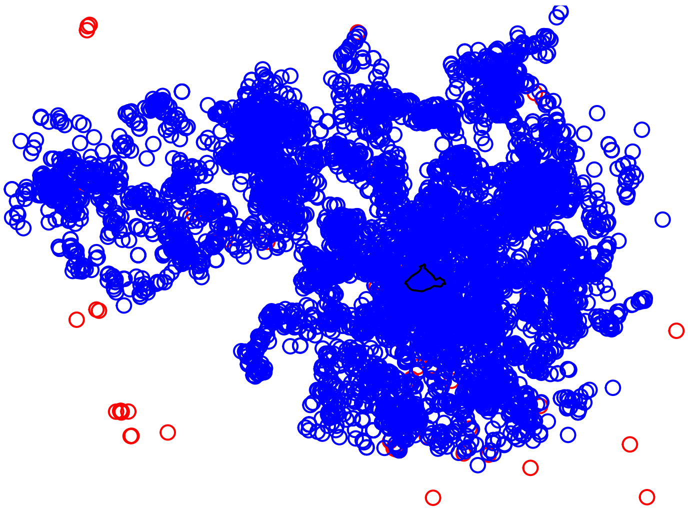
Recall above that we wanted to focus our efforts on the City Centre ward of Greater Manchester, however for our bounding box to download OSM data we used Greater Manchester. If we were to plot our bars, we would see that we have many which fall ourside of the City Centre ward:

```
plot(st_geometry(osm_bar_sf), col = 'red')
plot(st_geometry(city_centre),  add = TRUE)
```



This is also the case for our crimes data:

```
plot(st_geometry(osm_bar_sf), col = 'red')
plot(st_geometry(crimes), col = 'blue', add = TRUE)
plot(st_geometry(city_centre), add = TRUE)
```



So if we really want to focus on city centre, we should create spatial objects for the crimes and the bars which include only those which fall within the City Centre ward boundary.

First things first, we check whether they have the same CRS.

```
st_crs(city_centre) == st_crs(crimes)
```

```
## [1] TRUE
```

We do indeed, as we made sure in the previous section. Now we can move on to our spatial operation, where we select only those points within the city centre polygon. To do this, we first make a list of intersecting points to the polygon, using the `st_intersects()` function. This function takes two arguments, first the polygon which we want to subset our points within, and second, the points which we want to subset. We then use the resulting "cc_crimes" object to subset the crimes object to include only those which intersect (return `TRUE` for intersects):

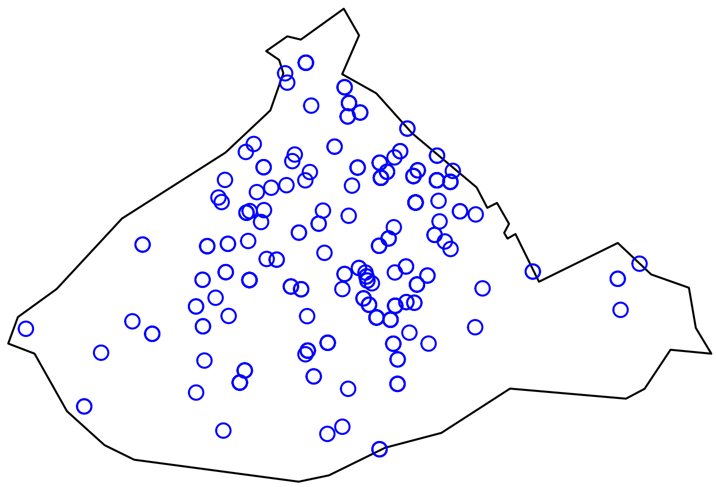
```
# intersection
cc_crimes <- st_intersects(city_centre, crimes)
# subsetting
cc_crimes <- crimes[unlist(cc_crimes),]
```

Have a look at this new "cc_crimes" object in your environment. How many observations does it have? Is this now fewer than the previous "crimes" object? Why do you think this is?

(hint: you're removing everything that is outside the city centre polygon)

We can plot this again to have a look:

```
plot(st_geometry(city_centre))
plot(st_geometry(cc_crimes), col = 'blue',  add = TRUE)
```



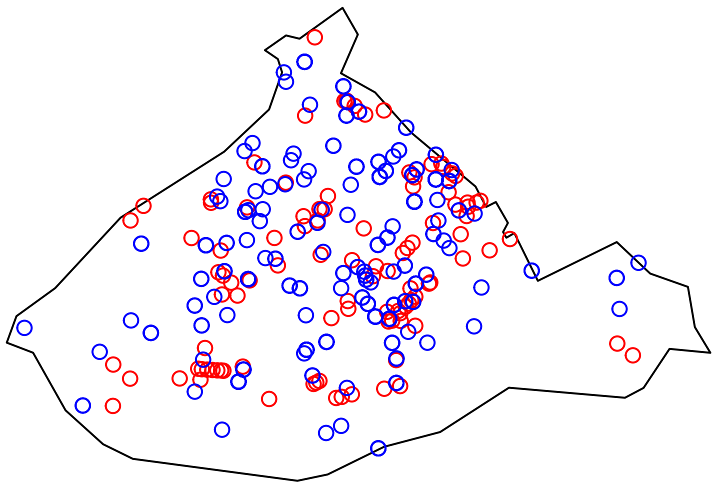
We have successfully performed our first spatial operation, we managed to subset our points data set or crimes to include only those crimes which are located inside the polygon for city centre.

We can do the same for the bars:

```
# intersection
cc_bars <- st_intersects(city_centre, osm_bar_sf)
# subsetting
cc_bars <- osm_bar_sf[unlist(cc_bars),]
```

We can see that of the previous 259 bars, 102 are within the City Centre ward. We can plot our data now:

```
plot(st_geometry(city_centre))
plot(st_geometry(cc_bars), col = 'red', add = TRUE)
plot(st_geometry(cc_crimes), col = 'blue', add = TRUE)
```



2.5.3 Building buffers

So we now have our bars and our violent crimes in Manchester City Centre. Let's go back to our original question. We want to know about crime in and around our areas of interest, in this case our bars. But how can we count this? We have our points that are crimes, right? Well… How do we connect them to our points that are licensed premises?

One approach is to build a buffer around our bars, and say that we will count all the crimes which fall within a specific radius of this bar. What should this radius be? Well this is where your domain knowledge as criminologist or crime analyst comes in. How far away would you consider a crime to still be related to this pub? 400 meters? 500 meters? 900 meters? 1 km? What do you think? This is one of them *it depends* questions, where there is no universal right answer, instead it will depend on the environment, the question, and contextual factors. Whatever buffer you choose you should justify, and make sure that you can defend when someone might ask about it, as the further your reach obviously the more crimes you will include, and these might alter your results.

So, let's say we are interested in all crimes that occur within 400 meters of each licensed premise. We chose 400m here as this is often the recommended distance for accessible bus stop guidance, so basically as far as people should walk to get to a bus stop. So in this case, we want to take our points, which represent the licensed premises, and build buffers of 400 meters around them.

You can do with the `st_buffer()` function. We pass two arguments to our function, the item which we want to buffer (the points in our 'cc_bars' object) and the size of this buffer.

Let's quickly illustrate:

```
prem_buffer <- st_buffer(cc_bars, 1)
```

You should get a warning here, saying *"st_buffer does not correctly buffer longitude/latitude datadist is assumed to be in decimal degrees (arc_degrees).".* This message indicates that sf assumes a distance value (our size of the buffer, specified as '1' above) is given in degrees. This is because we have our data in a Geographic Coordinate System (lat/long data in WSG 48).

If we want to calculate the size of the size of our buffer in a meaningful distance on our 2D surfaces, we can transform to a Projected Coordinate System, such as British National Grid. Let's do this now:

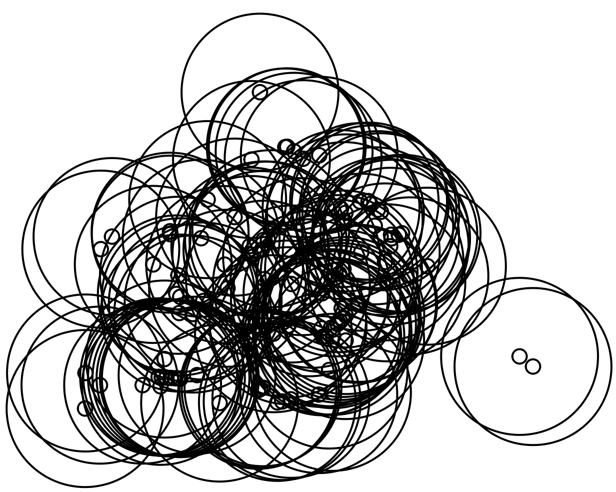
```
bars_bng <- st_transform(cc_bars, 27700) #The code for BNG is 27700
```

Now we can try again, with meters, specifying our indicated 400m radius:

```
bars_buffer <- st_buffer(bars_bng, 400)
```

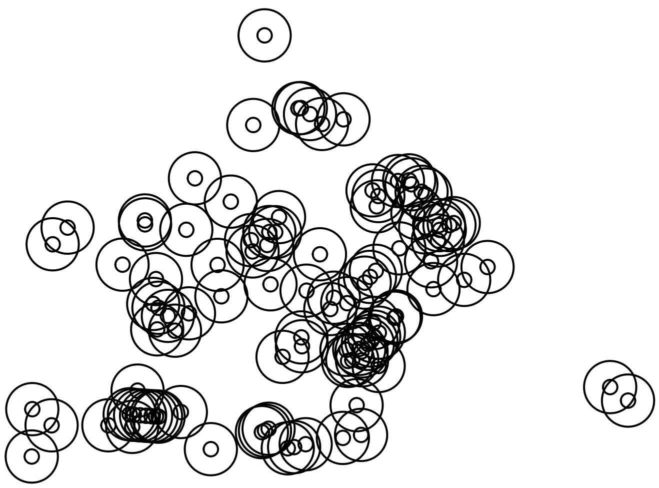
Let's see how that looks:

```
plot(st_geometry(bars_buffer))
plot(st_geometry(bars_bng), add = T)
```



That should look nice and squiggly. We can see it looks like there is *quite* a lot of overlap here. Should we maybe consider smaller buffers? Let's look at 100 meter buffers:

```
bar_buffer_100 <- st_buffer(bars_bng, 100) # create 100m buffer

# plot new buffers
plot(st_geometry(bar_buffer_100))
plot(st_geometry(bars_bng), add = T)
```



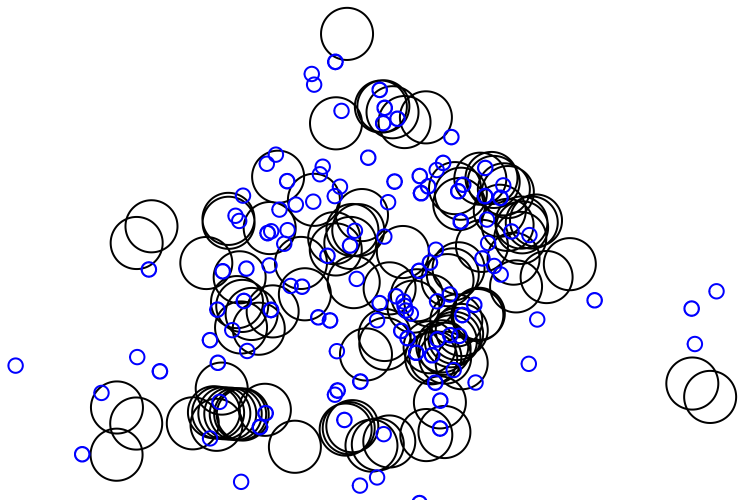
Still quite a bit of overlap, but this is possibly down to all the licensed premises being very densely close together in the city centre. We will discuss how to deal with this later on. For now, let's go with these 100m buffers, and where a crime falls into an area of overlap, we will count it towards both premises.

The next step will be to count the number of crimes which fall into each buffer. Before we move on though, remember the CRS for our crimes is WGS 48 here, so we will need to convert our buffer layer back to this:

```
buffer_WGS84 <- st_transform(bar_buffer_100, 4326)
```

Now let's just have a look:

```
plot(st_geometry(buffer_WGS84))
plot(st_geometry(cc_crimes), col = 'blue', add = T)
```



OKAY, so some crimes fall inside some buffers, others not so much. Well, let's get to our next spatial operation, to be able to determine how many crimes happened in the 100m radius of each bar in Manchester City Centre.

2.5.4 Counting Points within a Polygon

When you have a polygon layer and a point layer - and want to know how many or which of the points fall within the bounds of each polygon, you can use this method of analysis. In computational geometry, the point-in-polygon (PIP) problem asks whether a given point in the plane lies inside, outside, or on the boundary of a polygon. As you can see, this is quite relevant to our problem, wanting to count how many crimes (points) fall within 100 meters of our licensed premises (our buffer polygons).

```
crimes_per_prem <- cc_crimes %>%
  st_join(buffer_WGS84, ., left = FALSE) %>%
  count(name)
```

You now have a new dataframe, `crimes_per_prem` which has a column for the name of the bars, a column for the number of violent crimes that fall within the buffer, and a column for the geometry.

Take a moment to look at this table. Use the `View()` function. Which premises have the most violent crimes? If you are perhaps familiar with Manchester bars - what do you think? Are you surprised? I was!

Let's see the bar with the most crimes:

```
crimes_per_prem %>% filter(n == max(n)) %>% select(name, n)
```

```
## Simple feature collection with 1 feature and 2 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: -2.237934 ymin: 53.48095 xmax: -2.234921 ymax: 53.48275
## Geodetic CRS:  WGS 84
##        name  n                       geometry
## 1 Crafty Pig 59 POLYGON ((-2.234921 53.4818...
```

The bar with the highest number of crimes is Crafty Pig with 59 crimes. Keep this in mind for the next section…!

So in this case, we used the point-in-polygon approach, and counted the number of points which fell into each polygon. We saw earlier, with the buffers, that they often overlapped with one another. This means that a crime may have been counted multiple times. This resulting data therefore tells us: *How many crimes happened within 100 meters of each bar.* This is one way to approach the problem, but not the only way. In our next spatial operation, we will calculate distances in order to explore another way.

2.5.5 Distances: Finding the nearest point

Another way to solve this problem is to assign each crime (point) to the closest possible bar (other point). That is, look at the distances for each crime between it's location and the locations of all the bars in Manchester, and then, from those, choose the bar which is the closest. Then, we can assign this bar as the location for that crime.

We can achieve this using the `st_nearest_feature()` function. This function takes our two sf objects, and for each row of the first one (x = cc_crimes), simply returns us the index of the nearest features from the second one (y = cc_bars). We combine with the `mutate()` function in order to create a new variable which contains this index for each crime. Let's illustrate:

```
crime_w_bars <- cc_crimes %>% mutate(nearest_bar = st_nearest_feature(cc_crimes, cc_bars))
```

If we now have a look at this new object "crime_w_bars," we can see it is our crimes data, but we have a new column, which contains the index of the closest bar in the cc_bars dataframe, right at the end. So for example, for me the first point there, the nearest bar is that in location 84 (vectors in R are 1-indexed, not 0-indexed like many other languages). If we look at what is on the 84th row we see:

```
cc_bars[84]
```

This returns all the 90+ variables for this row. If we want only the name, we can query for the 84th row and the 2nd column (which is `name`):

```
cc_bars[84, 2]
```

```
## Simple feature collection with 1 feature and 1 field
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -2.236673 ymin: 53.4824 xmax: -2.236673 ymax: 53.4824
## Geodetic CRS:  WGS 84
##            name                  geometry
## 6404929536 Dive POINT (-2.236673 53.4824)
```

You can see the name is "Dive." For that first crime, in our data set, the nearest bar is "Dive" bar. Now, instead of going through this process manually for each point, we can use the index to subset within our `mutate()` function:

```
crime_w_bars  <- cc_crimes %>% mutate(nearest_bar = cc_bars[st_nearest_feature(cc_crime
```

Now we have new information in this `nearest_bar` column, the name of the nearest bar, and the geometry. We actually don't need the geometry for now, as we will simply be counting the frequency of each bar, which we can join back to our `cc_bars` object, which has a geometry, so we can extract the `$name` element only, and remove the geometry. Like so:

```
crimes_per_prem_2 <- crime_w_bars %>%        # create new crimes_per_prem_2 object
  st_drop_geometry() %>%                     # drop (remove) the geometry
  group_by(nearest_bar$name) %>%             # group by to find frequecy of each bar (
  summarise(num_crimes = n()) %>%            # count number of crimes which had each b
  rename(name = `nearest_bar$name`)          # rename variable to 'name'
```

To tie this back to our spatial object "cc_bars" we can use the `left_join()` function:

```
crimes_per_prem_2 <- left_join(cc_bars, crimes_per_prem_2, by = c("name" = "name"))
```

Let's see the bar with the most crimes with this approach:

```
crimes_per_prem_2 %>% filter(num_crimes == max(num_crimes, na.rm = TRUE)) %>% select(na
```

```
## Simple feature collection with 1 feature and 2 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -2.236427 ymin: 53.48185 xmax: -2.236427 ymax: 53.48185
## Geodetic CRS:  WGS 84
##         name num_crimes                   geometry
## 1 Crafty Pig         50 POINT (-2.236427 53.48185)
```

The bar with the highest number of crimes is still Crafty Pig, but now with 50 crimes. This means there is a difference in the number of crimes attributed to this bar bewteen the two approaches. Clearly there are 9 crimes which fell within the buffer in the first approach, but were closer to another bar in the dataset, and were instead attributed to that one using this approach.

So which is better?

This is once again up to you as the researcher and analyst to decide. They do slightly different things, and so will answer slightly different questions. With the nearest feature approach, instead of talking about the number of crimes within some distance to the bar, we are instead talking about for each crime, the closest venue. This might mean that we could be attributing crimes that happen quite far from the venue to it, just because it's the closest within our data set. However, we are counting each crime only once. Pros and cons need to be weighed up, to make decisions.

2.5.6 Measuring distances

Let's have a look at the bar called "Night & Day." We can select this from the `cc_bars`, the buffers, and the crimes

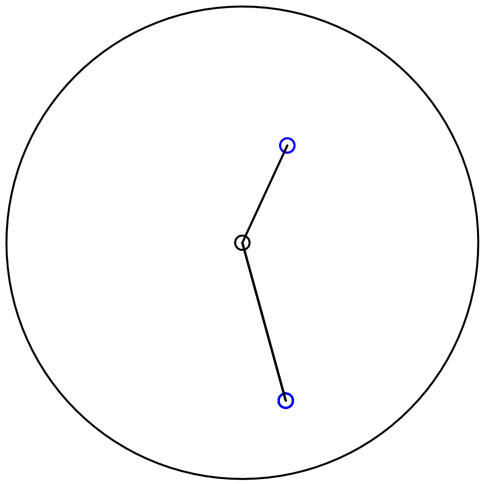
```
nd <- cc_bars %>%
  filter(name == "Night & Day")

nd_buffer <- bar_buffer_100 %>%
  filter(name == "Night & Day") %>%
  st_transform(4326)

nd_crimes <- crime_w_bars %>% filter(nearest_bar$name  == "Night & Day")
```

We can use `mapply()` and the `st_union()` function to draw a linestring between each crime and the closest bar (Night & Day in this case):

```
dist_lines <- st_sfc(mapply(function(a,b){st_cast(st_union(a,b),"LINESTRING")}, nd_crimes$geometr
```

We can then plot these to get an idea of what we're looking at:

```
plot(st_geometry(nd_buffer))
plot(st_geometry(nd), col = "black", add = TRUE)
plot(st_geometry(nd_crimes), col = "blue", add = TRUE)
plot(st_geometry(dist_lines), add = TRUE)
```



So we can see that the two locations where crimes occurred which were nearest to the Night & Day bar are both within the 100 meter buffer. But how far exactly are they?

You can use the `st_distance()` function to answer this question. We wrap this in the `mutate()` function in order to create a new column called *distance* which will contain for each row (each crime) the distance between that and its nearest bar (in this case Night & Day).

```
nd_crimes <- nd_crimes %>%
  mutate(distance = st_distance(geometry, nearest_bar$geometry))
```

Having a look at our newly created variable, we can see that the two crime locations are 69.3630151118733 and 45.3889988531455 away from Night & Day bar.

One thing you might find strange about the data is that why are all these crimes geocoded on top of one another? This is how the open data are released, using geo-masking by snapping crime locations to a geomask (a set of points). This is done to ensure anonymity in the data. In non-anonimysed data you might expect to see a little less overlap in your crime locations… Then with variation in distances between crimes and their nearest bars, we could use these distances to inform a buffer width for example. Anyway we will return to distances a little later with a better data set. But now, let's move on to putting these outcomes on a map!

2.6 Plotting interactive maps with leaflet

In the first chapter, we introduced the `ggplot2` package for making maps in R. In this chapter, we are going to introduce `leaflet` (http://leafletjs.com/) as one way to easily make some neat maps. It is the leading open-source JavaScript library for mobile-friendly interactive maps. It is very popular, used by websites ranging from The New York Times and The Washington Post to GitHub and Flickr, as well as GIS specialists like OpenStreetMap, Mapbox, and CartoDB, some of who's names you'll recognise from the various basemaps we played with in previous labs.

In this section of the lab we will learn how to make really flashy looking maps using `leaflet`. If you haven't already, you will need to have installed the following packages to follow along:

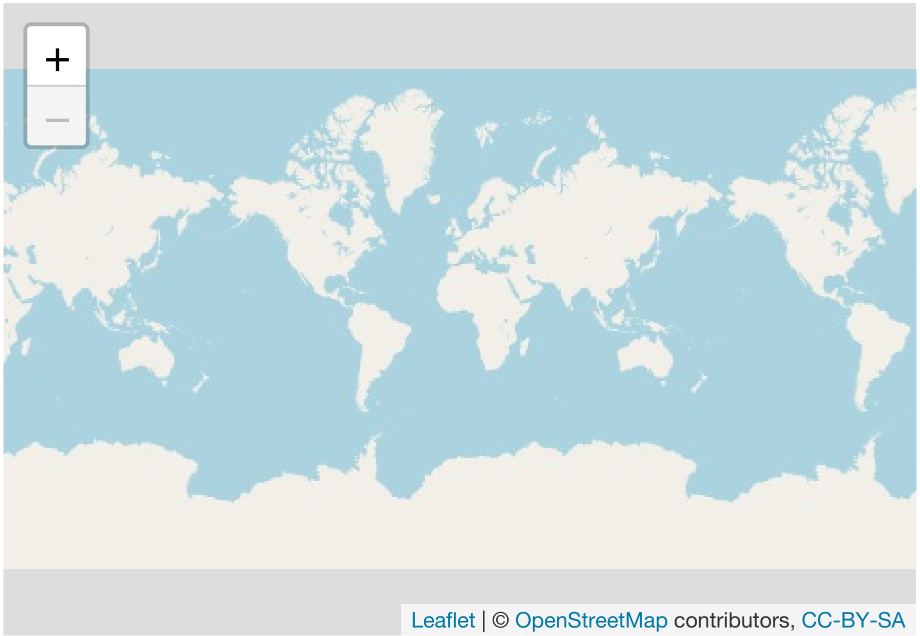
```
install.packages("leaflet") #for mapping
install.packages("RColorBrewer") #for getting nice colours for your maps
```

Once you have them installed, load them up with the `library()` function. To make a map, just load the leaflet library. You then create a map with this simple bit of code:

```
m <- leaflet() %>%
  addTiles()
```

And just print it:

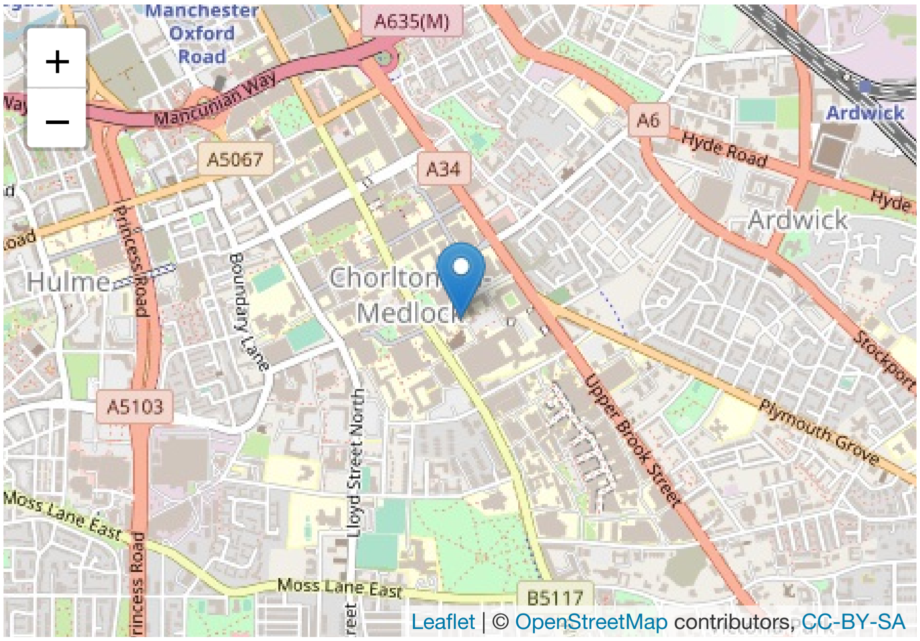
```
m
```



Not a super useful map, but it was really easy to make! You might of course want to add some content to your map.

You can add a point manually:

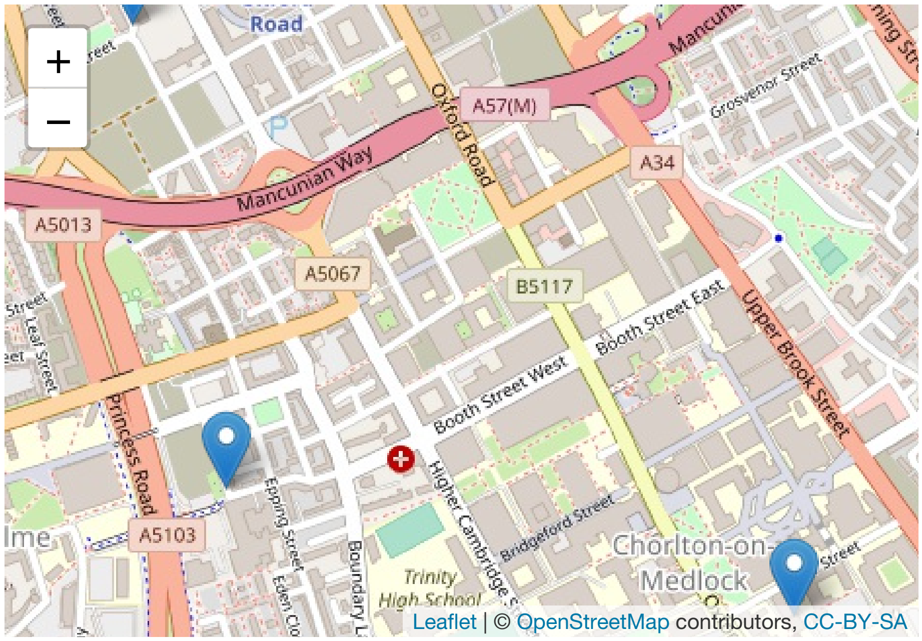
```
m <- leaflet() %>%
  addTiles()  %>%
  addMarkers(lng=-2.230899, lat=53.464987, popup="University of Manchester")
m
```



If you click over the highlighted point you will read our input text "University of Manchester."

You can add many points manually, with some popup text as well:

```
latitudes = c(53.464987, 53.472726, 53.466649)
longitudes = c(-2.230899, -2.245481, -2.243421)
popups = c("You are here", "Here is another point", "Here is another point")
df = data.frame(latitudes, longitudes, popups)

m <- leaflet(data = df) %>%
  addTiles()  %>%
  addMarkers(lng=~longitudes, lat=~latitudes, popup=~popups)
m
```



We can also map polygons, not just points. Let's plot our crimes on/near bars to illustrate. To do this, we can return to our buffers where we counted the number of crimes within 100m of each bar/ licensed premise (the "crimes_per_prem" object).

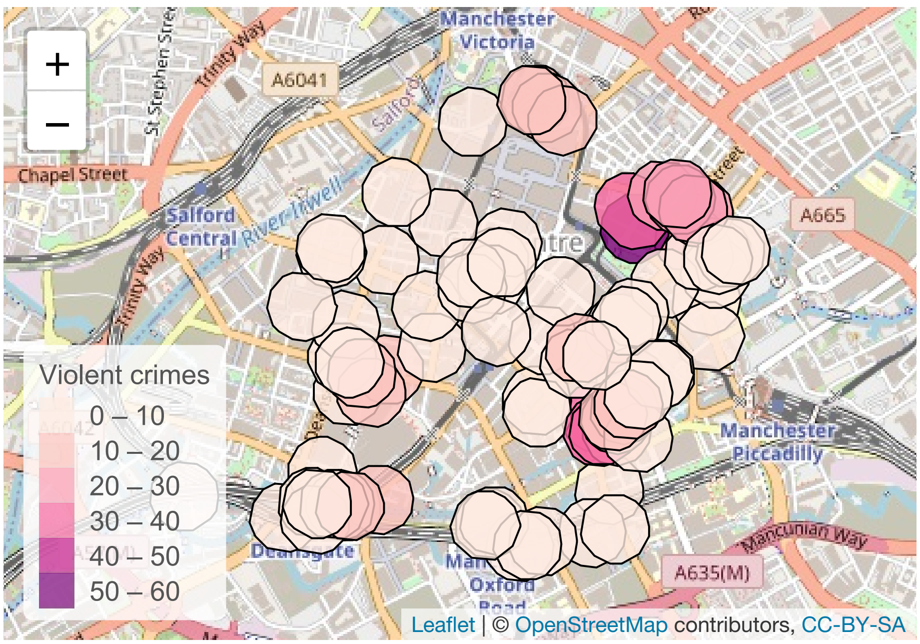
First, let's pick a colour palette. We do this with the `colorBin()` function. We will discuss colour choices in maps in Chapter 5, for now, let's just pick the *"RdPu"* palette. We should also specify the `domain =` parameter (what value to use for shading, in this case `n`, `bins =` - the number of crimes), the number of bins (in this case 5 - we will discuss this in detail in the coming chapters as well), and `pretty =` to use pretty breaks (this may actually mess with the number of bins specified in the bins parameter, but again, for now this is OK).

Let's create this palette and save in an object called `pal`:

```
pal <- colorBin("RdPu", domain = crimes_per_prem$n, bins = 5, pretty = TRUE)
```

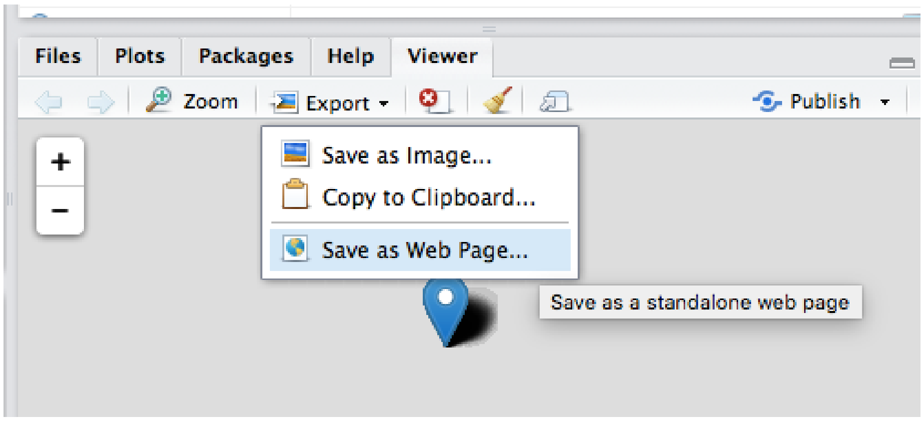
Now we can make a leaflet map, where we add these polygons (the buffers) with the `addPolygons()` function, and call our palette, specifying again the variable to use for shading, as well as some other parameters. One to highlight specifically is the `label` parameter. This allows us to use a variable as a label for when a user clicks on our polygon (buffer). Here we specify the name of the bar with `label = ~as.character(name)`.This way we not only shade each buffer with the number of crimes which fall inside it, but also include a little popup label with the name of the establishment:

```
leaflet(crimes_per_prem) %>%
  addTiles() %>%
  addPolygons(fillColor = ~pal(n), fillOpacity = 0.8,
              weight = 1, opacity = 1, color = "black",
              label = ~as.character(name)) %>%
  addLegend(pal = pal, values = ~n, opacity = 0.7,
            title = 'Violent crimes', position = "bottomleft")
```



It's not the neatest of maps, with all these overlaps, but we will talk about prettifying maps further down the line.

Now let's say you wanted to save this map. You can do this by clicking on the export button at the top of the plot viewer, and choose the *Save as Webpage* option saving this as a .html file:



Then you can open this file with any type of web browser (safari, firefox, chrome) and share your map that way. You can send this to your friends, and make them jealous of your fancy map making skills.

2.7 Geocoding

We were making use of point of interest data from Open Street Map above, but it's possible that we have a data set of bars that are not geocoded. In this case, we may have a list of bars with an associated address, which is clearly *some* sort of spatial information, but how would you put this on a map?

One solution to this problem is to geocode these addresses. We can use the package `tidygeocoder` to achieve this. This package takes an address given as character values, for example "221B Baker Street, Marylebone, London NW1 6XE" and returns coordinates, geocoding this address. So let's say we have a dataframe of addresses (in this case only one observation):

```
addresses <- data.frame(name = "Sherlock Holmes",
                        address = "221B Baker Street, London, UK")
```

We can then use the `geocode()` function to get coordinates for this address. We have to specify the column which has the address (in this case *address*), and the method to use for geocoding. See the help file for the function for the many options. For example if you are in the USA you may use "census." Since we are global, we will use "osm," which uses nominatim (OSM) to provide worldwide coverage. So given the above example:

```
addresses %>%
  geocode(address, method = 'osm')
```

```
## # A tibble: 1 x 4
##   name            address                              lat   long
##   <chr>           <chr>                              <dbl>  <dbl>
## 1 Sherlock Holmes 221B Baker Street, London, UK  51.5 -0.158
```

To illustrate on scale, let's have a look at another source of data on bars in Manchester. Manchester City Council have an Open Data Catalogue (http://open.manchester.gov.uk/open/homepage/3/manchester_open_data_catalogue) on their website, which you can use to browse through what sorts of data they release to the public. Like in many city open data portals, there are a some more and some less interesting data sets made available here. It's not quite as impressive as the open data from some of the cities in the US such as New York (https://opendata.cityofnewyork.us/) or Dallas (https://www.dallasopendata.com/) but we'll take it.

One interesting data set, especially for our questions about the different alcohol outlets is the Licensed Premises (http://www.manchester.gov.uk/open/downloads/file/169/licensed_premises) data set. This details all the currently active licenced premises in Manchester. You can see there is a link to download.

As always, there are a few ways you can download this data set. On the manual side of things, you can simply right click on the download link from the website, save it to your computer, and read it in from there, by specifying the file path. Remember, if you save it in your R *working directory*, then you just need to specify the file name, as the working directory folder is where R will first look for this file.

So without dragging this on any further, let's read in the licensed premises data directly from the web:

```
lic_prem <- read_csv("http://www.manchester.gov.uk/open/download/downloads/id/169/licensed_premis
```

You will likely get some warnings when reading this data, but you can safely ignore those. You can always check if this worked by looking to your global environment on the righ hand side and seeing if this *lic_prem* object has appeared. If it has, you should see it has 65535 observations (rows), and 36 variables (columns).

Let's have a look at what this data set looks like. You can use the `View()` function for this:

```
View(lic_prem)
```

We see that there is a field for "premisesname" which is the name of the premise, and two fields, "locationtext" "postcode" which refer to address information. To geocode these, let's create a new column which combines the address and post code, and then use the `geocode()` function introduced above.

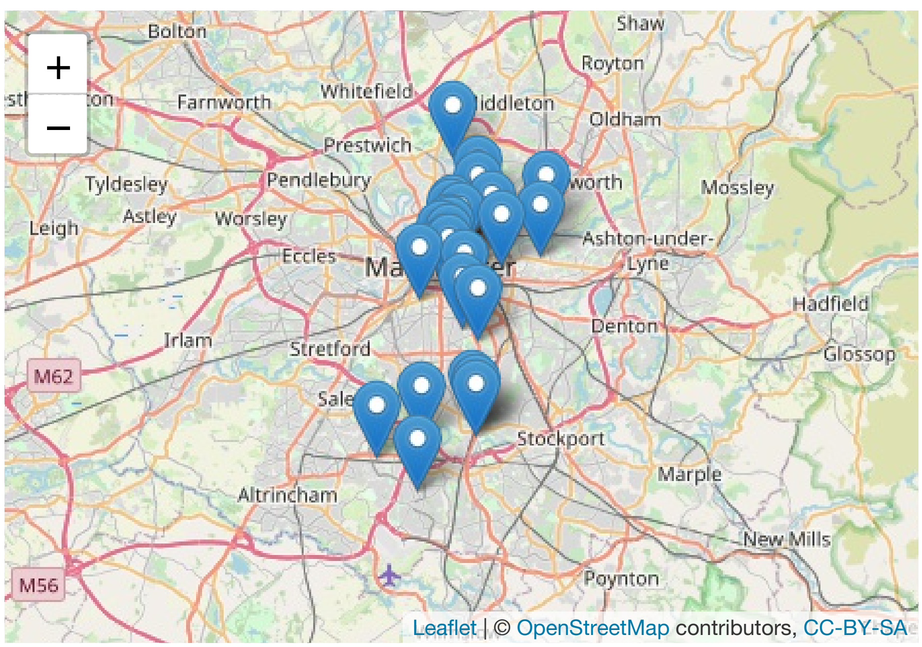
Note: this will take a while (like hours!) for the whole 65535 addresses data set, so just to illustrate for teaching purposes, we take the first 30

```
lic_prem <- lic_prem %>%
  slice(1:30) %>%                                                          # Select j
  mutate(complete_address = paste(locationtext, postcode, sep=", ")) %>%     # Create t
  geocode(complete_address, method = 'osm')                                # geocode
```

Now we have these licenced premises geocoded, with brand new latitude and longitude information! We can use this to make a leaflet map of our venues!

```
lic_prem$latitude <- as.numeric(lic_prem$lat)
lic_prem$longitude <- as.numeric(lic_prem$long)

leaflet(data = lic_prem) %>%
  addTiles() %>%
  addMarkers(lng=~longitude, lat=~latitude, popup=~as.character(premisesname), label =
```

```
## Warning in validateCoords(lng, lat, funcName): Data contains 4 rows with either
## missing or invalid lat/lon values and will be ignored
```



Geocoding may come in handy when we have address data, or something similar but no geometry to use to map it.

2.8 Measuring distance more thoroughly

Before we end the chapter, we want to return to the spatial operation of measuring the distance between points. In crime science there is a whole area of research that, for example, focuses on studying the journey to crime by offenders and a common parameter studied is the average distance to crime from their home locations. In order to estimate these parameters, we first need to have a way to generate the distances. In this section, we will use another data set (this time from Madrid, Spain) to show a simpler example to look at the issue of geographical distance.

2.8.1 How far are police stations in Madrid?

To illustrate how to measure distance we will download data from the city of Madrid in Spain. Specifically we will obtain a csv file with the latitude and longitude of the police stations and a geojson file with the administrative boundary for the city of Madrid. Both are stored in our GitHub repository and can be accessed with the code below. We will also turn the .csv into a `sf` object with the appropriate coordinate reference system for this data.

```
#read csv data
comisarias <- read_csv("data/nationalpolice.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   NOMBRE = col_character(),
##   X = col_double(),
##   Y = col_double()
## )
```

```
#set crs, read into sf object, and double check crs
polCRS <- st_crs(4326)
comisarias_sf <- st_as_sf(comisarias, coords = c("X", "Y"), crs = polCRS)
st_crs(comisarias_sf)
```

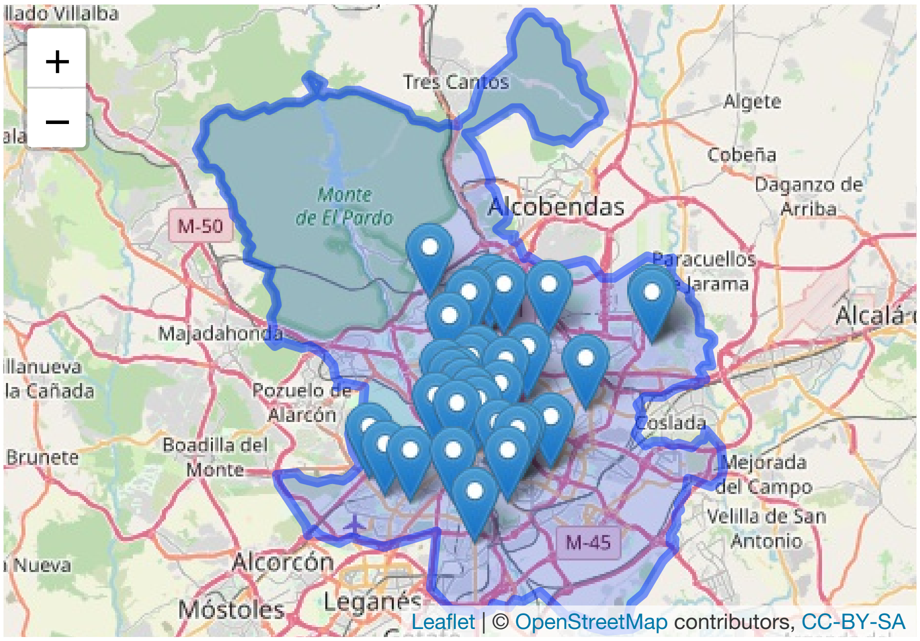
```
## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
```

```
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["World"],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

```
#create unique id for each row
comisarias_sf$id <- as.numeric(rownames(comisarias_sf))

#Read as sf boundary data for Madrid city
madrid <- st_read("data/madrid.geojson")
```

```
## Reading layer `madrid' from data source `/Users/reka/Desktop/crime_mapping/crime_ma
## Simple feature collection with 1 feature and 4 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -3.888963 ymin: 40.31206 xmax: -3.518126 ymax: 40.64328
## Geodetic CRS:  WGS 84
```

```
#plot with leaflet to check all is fine
leaflet(comisarias_sf) %>%
  addTiles() %>%
  addMarkers(data = comisarias_sf) %>%
  addPolygons(data = madrid)
```



We can clearly see here that there are areas of the municipal term that are far away from any national police station, the North West part of the city which you can see is a green area but noticeably also the South East, which is mostly urban and in fact is the location of a known shanty town and open drug market ("Cañada Real," you can read about it in the award-winning Briggs and Gamero (2017)).

2.8.2 Distance in geographical space

There are many definitions of distance in data science and spatial data science. A common definition of distance is the **Euclidean distance**, which simply is the length of a segment connecting two points in a two dimensional place. Because of the distortions caused by projections on a flat surface, a straight line on a map is not necessarily the shortest distance. Thus, another common definition used in geography is the **great circle distance**, which corresponds to an arc linking two points on a sphere and takes into account the spherical shape of the world.The great circle distance is useful, for example, to evaluate the shortest path when intercontinental distances are concerned.

We can compute both with the `st_distance` function of the `sf` package. This function can be used to measure the distance between two points, between one point and others or between all points. In the latter case we obtain a symmetric matrix of distances (NxN), taken pairwise between the points in our dataset. In the diagonal we find the combinations between the same points giving all null values.

Say we want to measure the distance between the main police headquarters ("Jefatura Superior de Policia," row 34) and three other stations (say row 1, row 10, and row 25 in our dataset). We could use the following code for that:

```
# calculate distance
dist_headquarters <- st_distance(slice(comisarias_sf, 34),
                                 slice(comisarias_sf, c(1, 10, 25)))

dist_headquarters # distance in meters
```

```
## Units: [m]
##          [,1]    [,2]     [,3]
## [1,] 8123.707 5096.02 8573.521
```

The result is a matrix with a single row or column (depending on the order of the spatial objects) with a class of *units*.

Often we may want to reexpress these distances in a different unit. For this purpose the `units` package offers useful functionality, through the `set_units()` function.

```
set_units(dist_headquarters, "km")
```

```
## Units: [km]
##          [,1]    [,2]     [,3]
## [1,] 8.123707 5.09602 8.573521
```

We can compute the distance between all police stations as well.

```
# calculate distance
m_distance <- st_distance(comisarias_sf)

# matrix dimensions
dim(m_distance)
```

```
## [1] 34 34
```

If you want to preview the top of the matrix you can use:

```
head(m_distance)
```

2.8.3 A practical example to evaluate distance

For this practical example we will look at the Madrid data.

```
plot(madrid, max.plot = 1)
```

We know this map is in EPSG 4326.

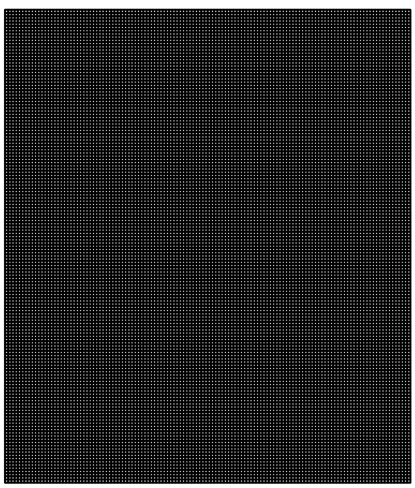
```
st_crs(madrid)
```

The distance here is expressed in degrees and we may prefer to evaluate distance in metrical system. Another way of saying this is that the current CRS uses geographic coordinates that account for earth's curvature, when we may prefer a flat surface to measure distances. For this we will transform and reproject to EPGS 2062, which is appropriate for Madrid (see here for details https://epsg.io/?q=Spain%20kind%3APROJCRS).

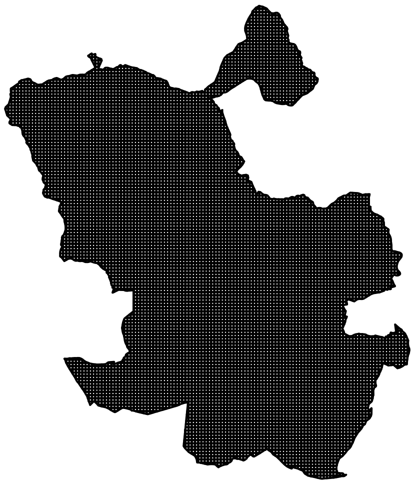
```
madrid_meters <- st_transform(madrid, crs = 2062)
```

Before we saw that some areas of Madrid are nowhere near a police station. Let´s say we want to get precise about this and we want to know how far different parts of the city of Madrid are from a police station, and we want to be able to show this in a map. Solving this means we have to define "parts of the city." What we will do is to divide the city of Madrid into different cells of 250 meters within a grid using the `st_make_grid` function of the `sf` package. "

```
madrid_grid <- st_make_grid(madrid_meters,  cellsize = 250)
plot(madrid_grid)
```

 We are now going to just extract the cells within the perimeter of Madrid:

```
#only extract the points in the limits of Iceland
madrid_grid <- st_intersection(madrid_grid, madrid_meters)

#our fishnet now
plot(madrid_grid)
```



```
#Reproject to 4326 for later use
madrid_grid_wgs <- st_transform(madrid_grid, crs = 4326)
```

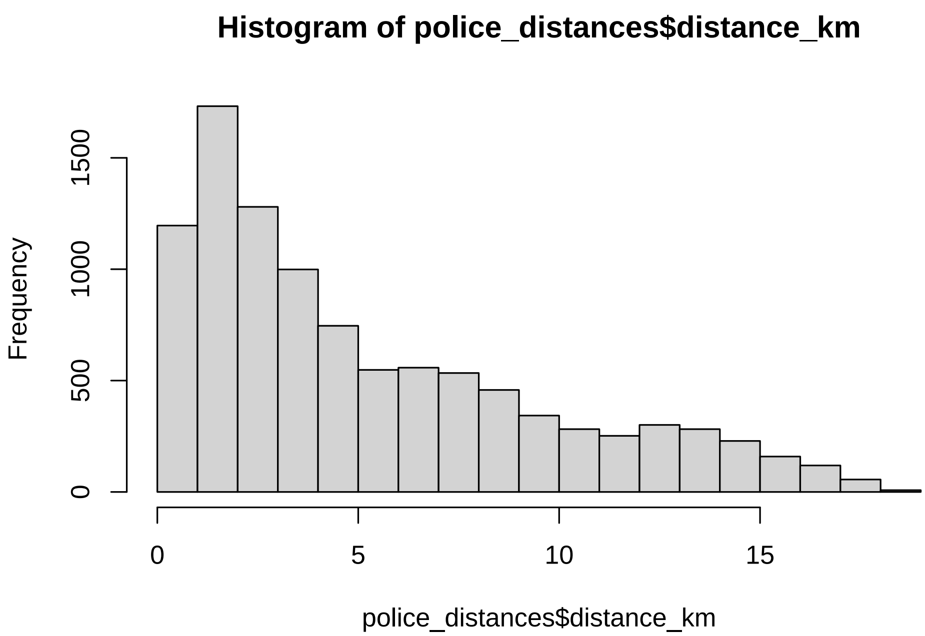
So how do we look at distance from police stations here? We will measure the distance between each grid cell to all 39 police stations. To estimate the distance to the nearest police station we will find the minimum distance value for each grid, i.e. the distance to the nearest station.

```
comisarias_sf_meters <- st_transform(comisarias_sf, crs = 2062)

distances <- st_distance(comisarias_sf_meters, st_centroid(madrid_grid)) %>%
as_tibble()
```

If you view the new object "distances" you will see there is a row for each police station and a column representing each of the 10082 cells in our grid. For using these distances in a leaflet map we will reproject back into 4326. And then will compute the shortest distance for each cell.

```
# Compute distances
police_distances <- data.frame(
  # We want grids in a WGS 84 CRS:
  us = st_transform(madrid_grid, crs = 4326),
  # Extract minimum distance for each grid
  distance_km = map_dbl(distances, min)/1000,
  # Extract the value's index for joining with the ice-cream location info
  location_id = map_dbl(distances, function(x) match(min(x), x))) %>%
  # Join with the police station table
  left_join(comisarias_sf, by = c("location_id" = "id"))
```

```r
# Plot and examine distances

hist(police_distances$distance_km)
```




```r
quantile(police_distances$distance_km)
```

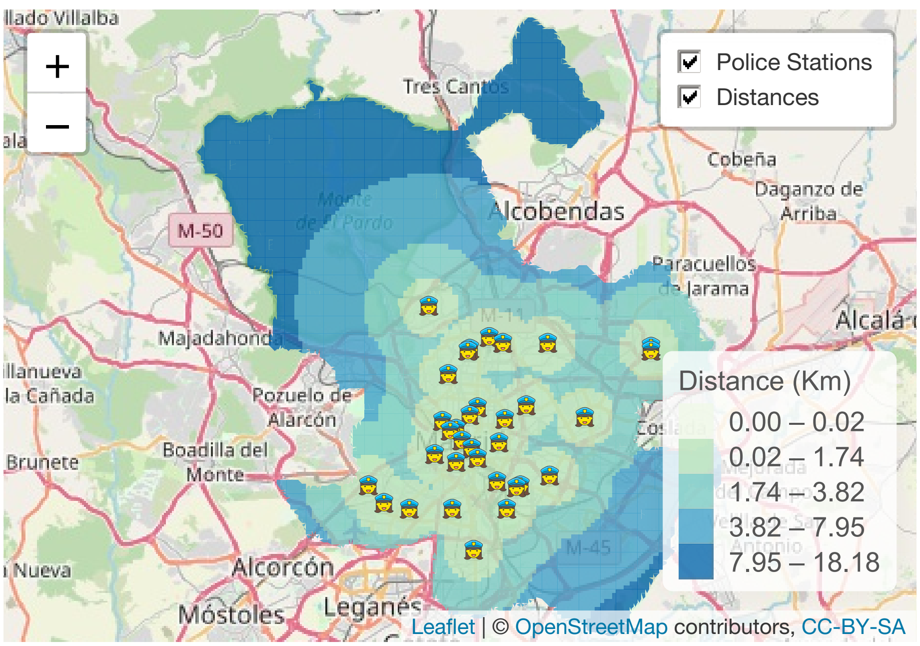
```
##          0%         25%         50%         75%        100%
##  0.01861606  1.73978931  3.81578329  7.95273442 18.17877100
```

Now we are ready to use this data to plot a map. First we will adjust some aesthetics.

```r
# Create more appropriate icon, taking it from Wikepedia commons and adjusting size
police_icon <- makeIcon("https://upload.wikimedia.org/wikipedia/commons/a/ad/189-woman-
```

```r
# Bin ranges for a nicer color scale
bins <- c(0,0.02,1.74,3.82,7.95,18.18)
# Create a binned color palette
pal <- colorBin(c("#0868AC", "#43A2CA", "#7BCCC4", "#BAE4BC", "#F0F9E8"),
                domain = police_distances$distance_km, bins = bins,
                reverse = TRUE)
```

```
full_map <- leaflet() %>%
  addTiles() %>%
  addMarkers(data = comisarias_sf, icon = ~police_icon,
             group = "Police stations") %>%
  addPolygons(data = police_distances[[1]],
              fillColor = pal(police_distances$distance_km),
              fillOpacity = 0.8, weight =0,
              opacity =1, color = "transparent", group = "Distances",
              highlight = highlightOptions(weight = 2.5, color = "#666", bringToFront = TRUE, opa
                                           closeOnClick = TRUE, textOnly = T)) %>%
   addLegend(pal = pal, values = (police_distances$distance_km), opacity = 0.8,
                       title = "Distance (Km)", position= "bottomright") %>%
  addLayersControl(overlayGroups = c("Police Stations", "Distances"),
                   options = layersControlOptions(collapsed = FALSE))
full_map
```



And there you go. Just remember something. It is easy to misinterpret data and maps. You always need to care a great deal about measurement, quality of your data, and other potential issues affecting interpretation. When it comes to distance, and the movements of people and law enforcement personnel, for example, physical distance is not trivial, but time to arrival is also important and this is determined by factors other than Euclidean distance (e.g., availability and speed of transport, physical barriers, etc.). Our representation is always as good as the data we have. In Spain there are two other police forces (Guardia Civil, that patrols rural areas, and municipal civil, with jurisdiction for local administrative enforcement) that we are not representing here (that is, our data is incomplete). And we are not plotting the police stations in the nearby municipalities that are part of Madrid metropolitan area, around the edges.

2.9 Further reading

The various topics covered here are typically discussed in standard GIS textbooks, such as those we have recommended in previous chapters. You could see, for example, Chapter 9 of Bolstad (2019). But probably the best follow up to what we discuss here is Chapter 4 and 5 of Lovelace, Nowosad, and Muenchow (2019), available for free as a digital text at "https://geocompr.robinlovelace.net/," for it provides a systematic introduction to how to perform these spatial operations with R. There is an online book on development by two giants of the R spatial community, Edzer Pebesma and Roger Bivand, that you can access at https://keen-swartz-3146c4.netlify.app/. The first 5 chapters of the book provide a strong backbone to understand `sf` objects in greater detail, coordinate systems, and key concepts for spatial data science.

Chapter 3

Chapter 3: Mapping rates and counts: choropleth and proportional symbol maps

3.1 Introduction

In the first chapter we showed you fairly quickly how to create maps by understanding how data may have spatial elements, and how that can be linked to geometries.

In this chapter we will get to know how to think about thematic maps, and how to apply your learning to creating your own maps of this variety. In the process we will discuss various types of thematic maps and the various issues they raise.

Thematic maps focus on representing the spatial pattern of a variable of interest (e.g, crimes, trust in the police, etc.) and they can be used for exploration and analysis or for presentation and communication to others. There are different types of thematic maps depending on how the variable of interest is represented. In this and the next chapters we will introduce some of these types of particular interest for crime analysis, the different challenges they pose, and some ideas that may help you to choose the best representation for your data. Critically, we need to think about the quality of the data we work with, for adequate measurement is the basis of any data analysis.

In this chapter, we introduce in particular two types of thematic maps used for mapping quantitative variables: **choropleth maps** and **proportional symbol maps**. In previous chapters we introduced two separate packages for creating maps in R (`ggplot2` and `leaflet`). In this chapter we will introduce a third package for creating maps: `tmap`. The libraries we use in this chapter are as follow:

```
# Packages for reading data and data carpentry
library(readr)
library(dplyr)
library(janitor)

# Packages for data exploration
library(skimr)

# Packages for handling spatial data and for geospatial carpentry
library(sf)

# Packages for mapping and visualisation
library(tmap)
library(tmaptools)

# Packages for spatial analysis
library(DCluster)

# Packages with spatial datasets
library(geodaData)
```

3.2 Thematic maps: key terms and ideas

3.2.1 Choropleth maps

Choropleth maps display variation in areas (postal codes, police districts, census administrative units, municipal boundaries, regional areas, etc) through the use of the colour that fills each of these areas in the map. A simple case can be to use a light to dark color to represent less to more of the quantitative variable. They are appropriate to compare values across areas.

Most choropleth maps you encounter are classified choropleth maps. They group the values of the quantitative variable into a number of classes, typically between 5 and 7. We will return to this later in the chapter. It is one of the most common forms of statistical maps and like pie charts they are subject to ongoing criticism. Tukey (1979) concluded they simply do not work. And there are indeed a number of known problems with choropleth maps: 1. They may not display variation within the geographical units being employed. You are imposing some degree of distortion by assuming all parts of an area display the same value. Our crime map may show a neighborhoud as secure, when there is a part of this neighbourhoud that has a high level of crime. And viceversa. 2. Boundaries of geographical areas are to a large extent arbitrary (and unlikely to be associated with major discontinuities in your variable of interest). In crime analysis we very often use census administrative units, but these rarely represent natural neighbourhoods. 3. They work better if areas are of similar size. Areas of greater size may be more heterogeneous internally than those of smaller size: that is, they potentially have the largest error of representation. Also, visual attention may be drawn by areas that are large in size (if size is not the variable use to create a ratio)

3.2.2 What do we use choropleth maps for?

3.2.2.1 Crime rates

In a choropleth map you "can" show raw totals (absolute values), for example number of crimes, or derived values (ratios), crime per 100,000 inhabitants. But as a general rule, you should restrict choropleth maps to show derived variables such as rates or percentages (Monmonier (1993), Field (2018)). This is because the areas we often use are of different size and this may introduce an element of confusion in the interpretation of choropleth maps. The size of say, a province or a county, "has a big effect on the amount of color shown on the map, but unit area may have little relationship, or even an inverse relationship, to base populations and related counts." (Brewer (2006): S30)

Mapping counts, mapping where a lot of the crime incidents concentrate is helpful, but we may want to understand as well if this is simply a function of the spatial distribution of the population at risk. As Ratcliffe (2010) suggests "practitioners often recognize that a substantial density of crime in a location is sufficient information to initiate a more detailed analysis of the problem," but equally we may want to know if "this clustering of crime is meaningfully non-random, and if the patterns observed are still present once the analysis has controlled for the population at risk." (p. 11-12).

A map of rates essentially aims to provide information into geographic variation of crime risk - understood as the probability that a crime may occur. Maps of rates are ultimately about communicating the risk of crime, with greater rates suggesting a higher probability of becoming a victim of crime.

In social science the denominator on mapped ratios is typically some form of population size, which typically we will want as current as possible. In other fields some measure of area size may be a preferred choice for the denominator.

However, a great deal of discussion in crime analysis has focused on the choice of the right denominator. Authors relate to this as the **denominator dilemma**: "the problem associated with identifying an appropriate target availability control that can overcome issues of spatial inequality in the areal units used to study crime" (Ratcliffe_2010). The best measure for your denominator is one which captures opportunities. If for example you are interested in residential burglary, it makes sense to use number of inhabited households as your denominator (rather than population size). Whatever denominator you choose, you will usually want to make a case as to why that is the best representation of the opportunities for the crime type you're interested in.

As noted, population is a common choice, but it is not always the one that best captures crime opportunities. Population, on the other hand, is also highly mobile during the day. People do not stay in the areas where they live, they go to work, they go to school, they travel for tourism purposes, and in doing so they alter the population structure of any given area. As Ratcliffe (2010) highlights with an example "the residential population (as is usually available from the census) tells the researcher little about the real number of people outside nightclubs at 2 a.m" Geographers and criminologists, thus, distinguish between the standard measures of population (that relate to people that live in an area) provided by the census and government statistical authorities; and the so called **ambient population**, that relates to people that occupy an area at a given time, and which typically are a bit more difficult to source (see for more details: Andresen (2011), [[[MORE REFERENCES]]]).

As we will see later, one of the key problems with mapping rates is that the estimated rates can be problematic if the enumeration areas we are studying have different population counts (or whatever it is we are counting in the denominator). When this happens, and those population counts produce small samples, we may have rates for some locations (those with more population) that are better estimated than others and, therefore, are less subject to noise.

Aside from problems with the denominator, we may also have problems with our numerator. In crime analysis, the key variable we map is geocoded crime, that is crime for which we know its exact location. The source for this variable tends to be crime reported to and recorded by the police. Yet, we know this is an imperfect source of data. A very large proportion of crime is not reported to the police. In England and Wales, for example, it is estimated that around 60% of crime is unknown to the police. And this is an average! The unknown figure of crime is larger for certain types of crime (e.g., interpersonal violence, sexual abuse, fraud, etc.). What is more, we know that there are community-level attributes that are associated with the level of crime *reporting* to the police (Goudriaan, Witterbrood, and Nieuwbeerta (2006)).

Equally, not all police forces or units are equally adept at properly *recording* all crimes reported to the police. An in-depth study conducted in England and Wales concluded that over 800,000 crimes reported to the police were not properly recorded (an under-recording of 19%, which is higher for violence and sexual abuse, 33% and 26% respectively) (Constabulary (2014)). There are, indeed, institutional, economic, cultural and political factors that may shape the quality of crime recording across different parts of the area we want to map out.

Finally, the quality of the geocoding process, which varies across crime type, also comes into the equation. Sometimes there are issues with positional accuracy or the inability to geocode an address. Some authors suggest we need to be able to geocode at least 85% of the crime incidents to get accurate maps (Ratcliffe (2004)), otherwise geocoded crime records may be spatially biased. More recent studies offer a less conservative estimate depending on the level of analysis and number of incidents (Andresen et al. (2020)). Although all this has been known for a while, criminologists and crime analysts are only now beginning to appreciate how this can affect the quality of crime mapping and the decision making based on this mapping (Buil-Gil, Medina, and Shlomo (2021); Hart and Zandbergen (2012)).

Although most of the time crime analysts are primarily concerned with mapping crime incidence and those factors associated with it, increasingly we see interest in the spatial representation of other variables of interest to criminologists such as fear of crime, trust in the police, etc. In these cases, the data typically come from surveys and the problem that may arise is whether we have a sample size large enough to derive estimates at the geographical level that we may want to work with. When this is not the case, methods for small area estimation are required (for details and criminological examples see [[[ADD REFERENCES]]]).

3.2.2.2 Prevalence vs incidence

There are many concepts in science that acquire multiple and confusing meanings. Two you surely will come across when thinking about rates are: incidence and prevalence. These are well defined in epidemiology. Criminology and epidemiology often use similar tools and concepts, but not in this regard. In criminological applications these terms often are understood differently and in, at least, two possible ways. Confused? You should be!

In public health and epidemiology prevalence refers to proportion of persons who have a condition at or during a particular time period, whereas incidence refers to the proportion or rate of persons who develop a condition during a particular time period. The numerator for prevalence is *all* cases during a given time period that have the condition, whereas the numerator for incidence is all *new* cases. What changes is the numerator and the key dimension in which it changes is time.

In criminology, on the other hand, you will find at least two ways of defining these terms. Those that focus on studying developmental criminology define prevalence as the percentage of a population that engages in crime during a specified period (number of offenders per population in a given time), while offending incidence refers to the frequency of offending among those criminally active during that period (number of offences per active offenders in a given time). For these criminologists the (total) crime rate in a population is the product of the prevalence and the incidence ((**Blumstein_1986?**)).

Confusingly, though, you will find authors and practitioners that consider incidence as equivalent to the (total) crime rate, as the number of crimes per population, and prevalence, as the number of victims per population. To make things more confusing, sometimes you see criminologists defining incidence as the number of crimes during a time period (say a year) and prevalence as the number of victims during the lifetime. To avoid this confusion when producing maps *is probably best to avoid these terms and simply refer to crime rate or victimisation rate and be very clear in your legends about the time period covered for both.*

3.2.3 Proportional and graduate symbol maps: mapping crime counts

Proportional symbol maps, on the other hand, are used to represent quantitative variables for either areas or point locations. Each area get a symbol and the size represents the intensity of the variable. It is expected with this type of map that the reader could estimate the different quantities mapped out and to be able to detecting patterns across the map (Field (2018)). The symbols typically will be squares or circles that are scaled "in proportion to the square root of each data value so that symbol areas visually represent the data values" ((**Brewer?**): S29). Circles are generally used since they are considered to perform better in facilitating visual interpretation.

We often use proportional symbol maps to represent count data (e.g., number of crimes reported in a given area). A common problem with proportional symbol maps is symbol congestion/overlap, especially if there are large variations in the size of symbols or if numerous data locations are close together.

A similar type of map use graduated symbol maps. As with choropleth classing, the symbol size may represent data ranges. They are sometimes used when data ranges are too great to practically represent the full range on a small map.

3.3 Creating proportional symbol maps: mapping count data with tmap

Today we are going to introduce the `tmap` package. This package was developed to easily produce thematic maps. It is inspired by the `ggplot2` package and the layered grammar of graphics. It was written by Martjin Tennekes, a Dutch data scientist. It is fairly user friendly and intuitive.

There are a number of vignettes in the CRAN repository with helpful explanations (https://cran.r-project.org/web/packages/tmap/index.html) and also in the GitHub repo for this package (https://github.com/mtennekes/tmap) that you can explore.

In `tmap` each map can be plotted as a static map (*plot mode*) and shown interactively (*view mode*). We will start by focusing on static maps.

For the purpose of demonstrating some of the functionality of this package we will use the Manchester crime data generated in the first chapter. We have now added a few variables obtained from the Census that will be handy later on. You can obtain the data in a GeoJSON format from our GitHub repo using the following code:

```
manchester <- st_read("https://raw.githubusercontent.com/maczokni/crime_mapping/master/data/manch
```

```
## Reading layer `manchester' from data source `https://raw.githubusercontent.com/maczokni/crime_
## Simple feature collection with 282 features and 5 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 378833.2 ymin: 382620.6 xmax: 390350.2 ymax: 405357.1
## Projected CRS: OSGB 1936 / British National Grid
```

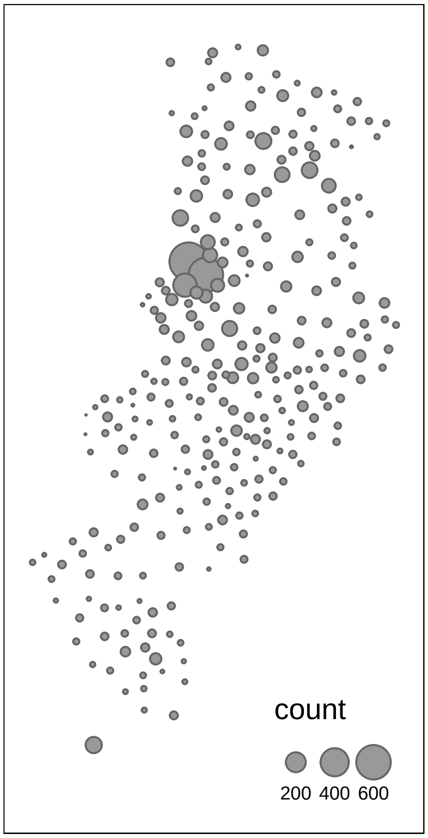
Every time you use this package you will need a line of code that specifies the spatial object you will be using. Although originally developed to handle `sp` objects only, it now also has support for `sf` objects. For specifying the spatial object we use the `tm_shape()` function and inside we specify the name of the spatial object we are using. Its key function is simply to do just that,to identify our spatial data. On its own, this will do nothing apparent. No map will be created. We need to add additional functions to specify what we are doing with that spatial object, how we want to represent it. If you try to run this line on its own, you'll get an error telling you you must "Specify at least one layer after each tm_shape."

```
tm_shape(manchester)
```

The main plotting method consists of elements that we can add. The first element is the `tm_shape()` function specifying the spatial object, and then we can add a series of elements specifying layers in the visualisation. They can include polygons, symbols, polylines, raster, and text labels as base layers.

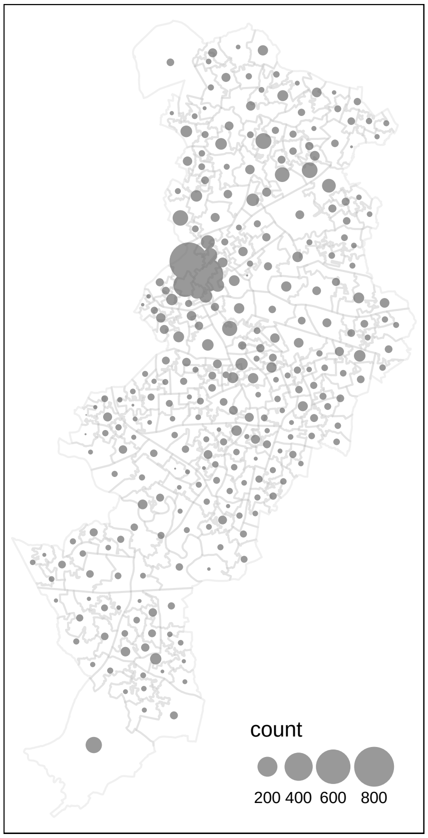
In `tmap` you can use `tm_symbols` for this. As noted, with `tmap` you can produce both static and interactive maps. The interactive maps rely on `leaflet`. You can control whether the map is static or interactive with the `tmap_mode()` function. If you want a static map you pass `plot` as an argument, if you want an interactive map you pass `view` as an argument. Let's create a static map first:

```
tmap_mode("plot")
tm_shape(manchester) +
  tm_bubbles("count")
```



Each bubble represents the number of crimes in each LSOA within the city of Manchester. We can add the borders of the census areas for better representation. The `border.lwd` argument set to NA in the `tm_bubbles()` is asking R not to draw a border to the circles. Whereas `tm_borders()` brings back a layer with the borders of the polygons representing the different LSOAs in Manchester city. Notice how we are modifying the transparency of the borders with the alpha parameter. In addition, we are adding a `tm_layout()` function that makes explicit where and how we want the legend.

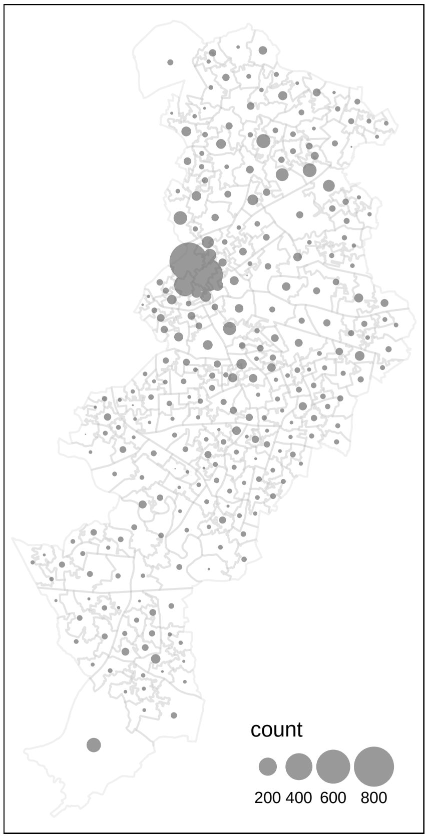
```
tm_shape(manchester) +                                  #use tm_shape function to specify spati
  tm_bubbles("count", border.lwd=NA) +                       #use tm_bubbles to add the bubble
  tm_borders(alpha=0.1) +                                    #add the LSOA border outlines usin
  tm_layout(legend.position = c("right", "bottom"), #use tm_layout to make the legend
            legend.title.size = 0.8,
            legend.text.size = 0.5)
```



There are several arguments that you can pass within the `tm_bubble` function that control the appearance of the symbols. The `scale` argument controls the symbol size multiplier number. Larger values will represent the largest bubble as larger. You can experiment with the value you find more appropriate. Another helpful parameter in `tm_bubble` is `alpha`, which you can use to make the symbols more or less transparent as a possible way to deal with situations where you may have a significant degree of overlapping between the symbols. You can play around with this and modify the code we provide above with different values for these two arguments.

By default, the symbol area sizes are scaled proportionally to the data variable you are using. As noted above this is done by taking the square root of the normalized data variable. This is called mathematical scaling. However, "it is well known that the perceived area of proportional symbols dose not match their mathematical area; rather, we are inclined to underestimate the area of larger symbols. As a solution to this problem, it is reasonable to modify the area of larger circles in order to match it with the perceived area." (Tanimura, Kuroiwa, and Mizota (2006)) The `perceptual = TRUE` option allows you to use a method that aims to compensate for how the default approach may underestimate the area of the larger symbols. Notice the difference between the map we produced and this new one.

```
tm_shape(manchester) +                              #use tm_shape function to specify spati
  tm_bubbles("count", border.lwd=NA, perceptual = TRUE) +           #use tm_bubbles
  tm_borders(alpha=0.1) +                                #add the LSOA border outlines usin
  tm_layout(legend.position = c("right", "bottom"), #use tm_layout to make the legend
            legend.title.size = 0.8,
            legend.text.size = 0.5)
```



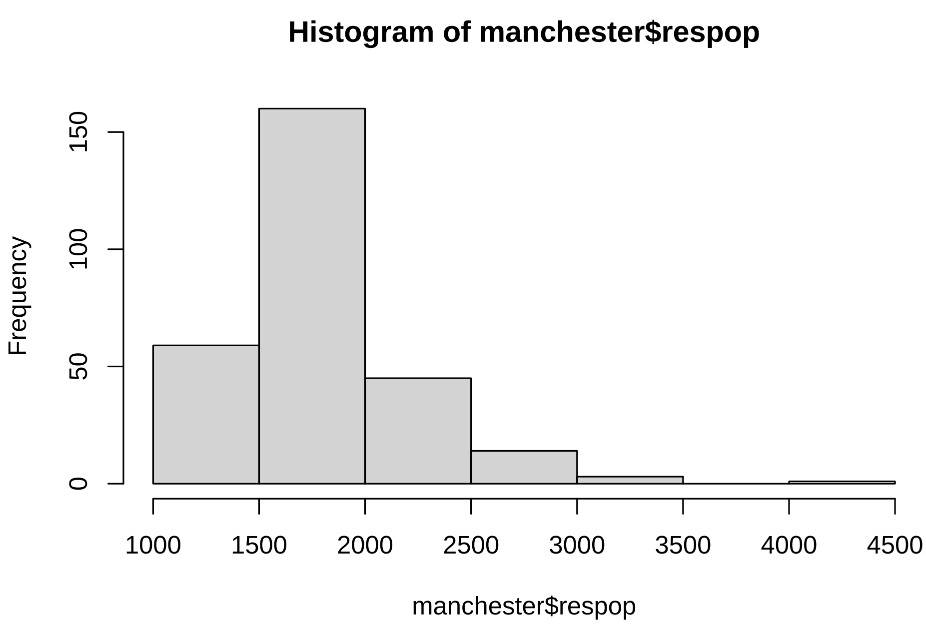
3.4 Mapping rates rather than counts

3.4.1 Generating the rates

We have now seen the importance to map rates rather than counts of things, and that is for the simple reason that population is not equally distributed in space. That means that if we do not account for how many people are somewhere, we end up mapping population size rather than our topic of interest.

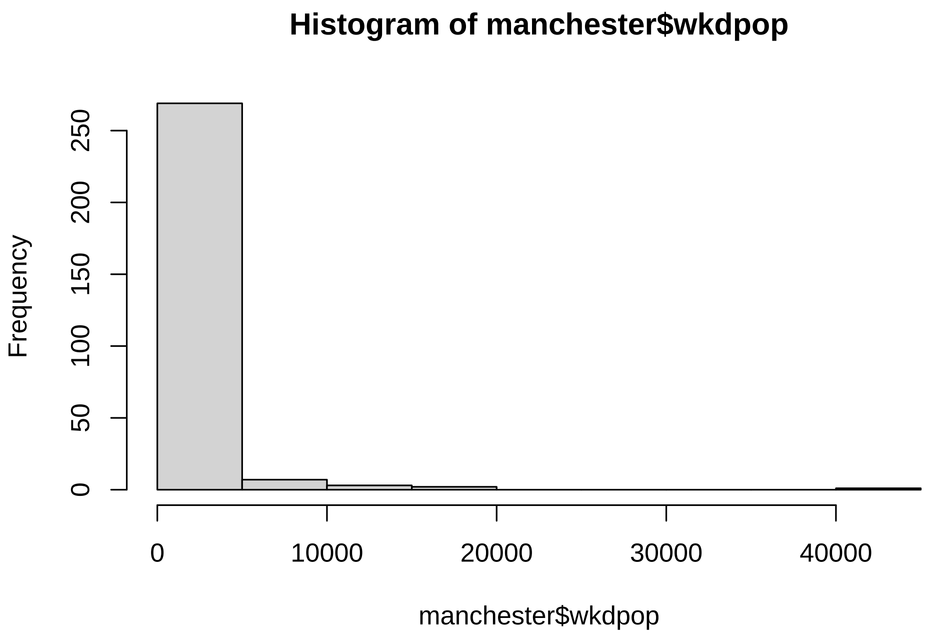
The *manchester* object we are working with has a column named "respop" that includes the residential population in each of the LSOA areas. These areas for the whole of the UK have an average population of around 1500 people. We can see a similar picture for Manchester city, with an average closer to 1700 here.

```
hist(manchester$respop)
```



We also have a variable "wkdpop" that represents the workday population. This variable re-distributes the usually resident population to their places of work, while those not in work are recorded at their usual residence. The picture it offers is much more diverse than the previous variable.

```
hist(manchester$wkdpop)
```



The distribution is much more skewed with some LSOAs in Manchester, with at least one LSOA attracting up to 42253 people during the working day. This gives an idea of the relevance of the "denominator dilemma" we mentioned earlier. In this section we will create rates of crime using both variables in the denominator to observe how the emerging picture varies.

First we need to create new variables. For this we can use the `mutate()` function from the `dplyr` package. This is a very helpful function to create new variables in a dataframe based on transformations or mathematical operations performed in other variables within the dataframe. In this function, the first argument is the name of the data frame, and then we can pass as arguments all new variables we want to create as well as the instructions as to how we are creating those variables.

First we want to create a rate using the usual residents, since crime rates are often expressed by 100,000 inhabitants we will multiply the division of the number of crimes by the number of usual residents by 100,000. We will then create another variable, "crimr2," using the workday population as the denominator. We will store this new variables in our existing "manchester" dataset. You can see that below then I specify the name of a new variable "crimr1" and then I tell the function I want that variable to equal (for each case) the division of the values in the variable "count" (number of crimes) by the variable "respop" (number of people residing in the area) and then we multiply the result of this division by 100,000 to obtain a rate expressed in those terms. Then we do likewise for the alternative measure of crime.

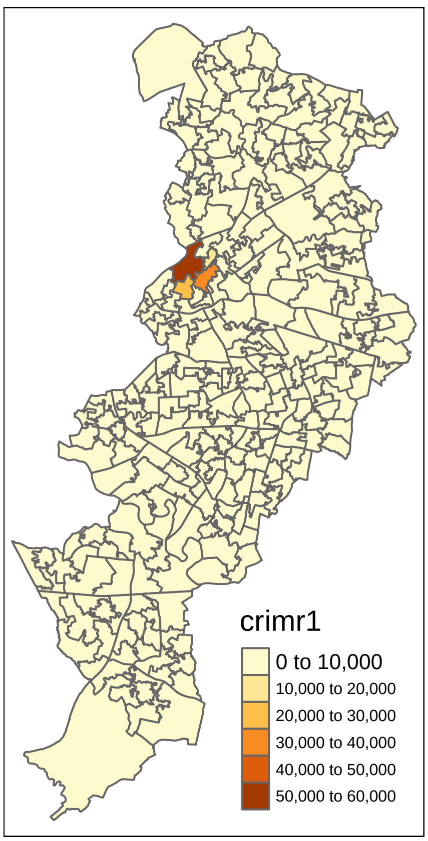
```
manchester <- mutate(manchester, crimr1 = (count/respop)*100000, crimr2 = (count/wkdpop)*100000)
```

And now we have two new variables, one for crime rate with residential population as a denominator, and another with workplace population as a denominator.

3.4.2 Creating a choropleth map with tmap

The structure of the grammar for producing a choropleth map is similar to what we use for proportional symbols. First we identify the object with `tm_shape()` and then we use a geometry to be represented. We will be using the `tm_fill()` passing as an argument the name of the variable with our crime rate.

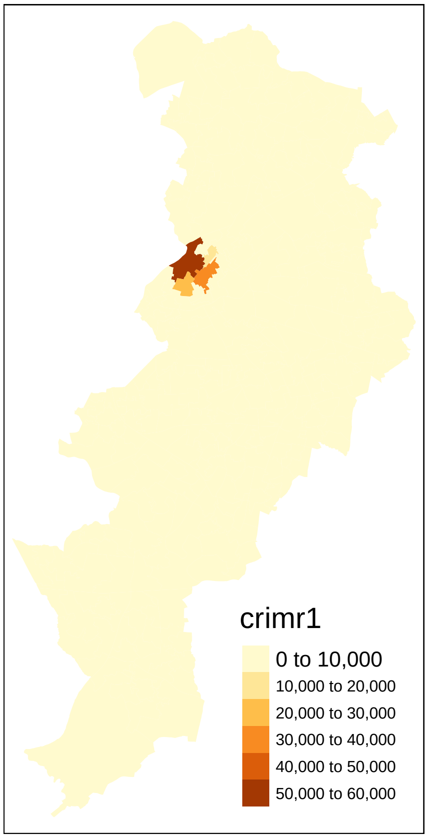
```
tm_shape(manchester) +
  tm_polygons("crimr1")
```



We have used `tm_polygons()` but we can also add the elements of a polygon map using different functions that break down what we represent here. In the map above you see the polygons have a dual representation, the borders are represented by lines and the colour is mapped to the intensity of the quantitative variable we are displaying. With darker colours representing more of the variable, the areas with more crimes.

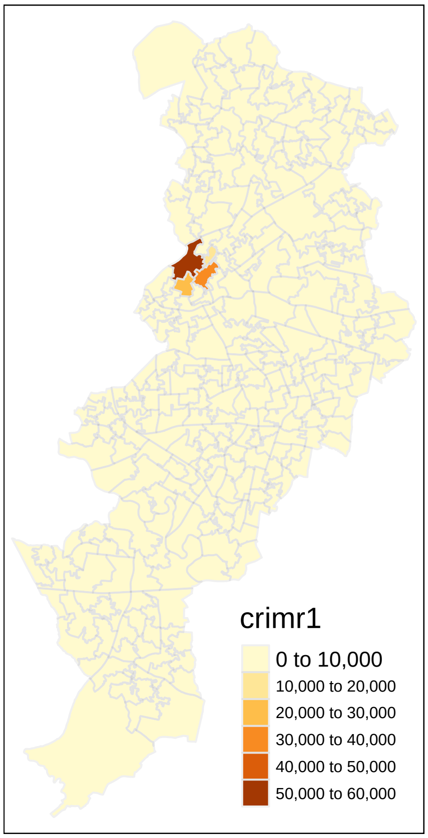
Instead of using `tm_polygon()` we can use the related functions `tm_fill()`, for the colour inside the polygons, and `tm_borders()`, for the aesthetics representing the border of the polygons. Say we find the borders distracting and we want to set them to be transparent. In that case we could just use `tm_fill()`.

```
tm_shape(manchester) +
  tm_fill("crimr1")
```



As you can see here, the look is a bit cleaner. However, we don't need to get rid of the borders completely. Perhaps we want to make them a bit more translucent. We could do that by adding the border element but making the drawing of the borders less pronounced.

```
tm_shape(manchester) +
  tm_fill("crimr1") +
  tm_borders(alpha = 0.1)
```

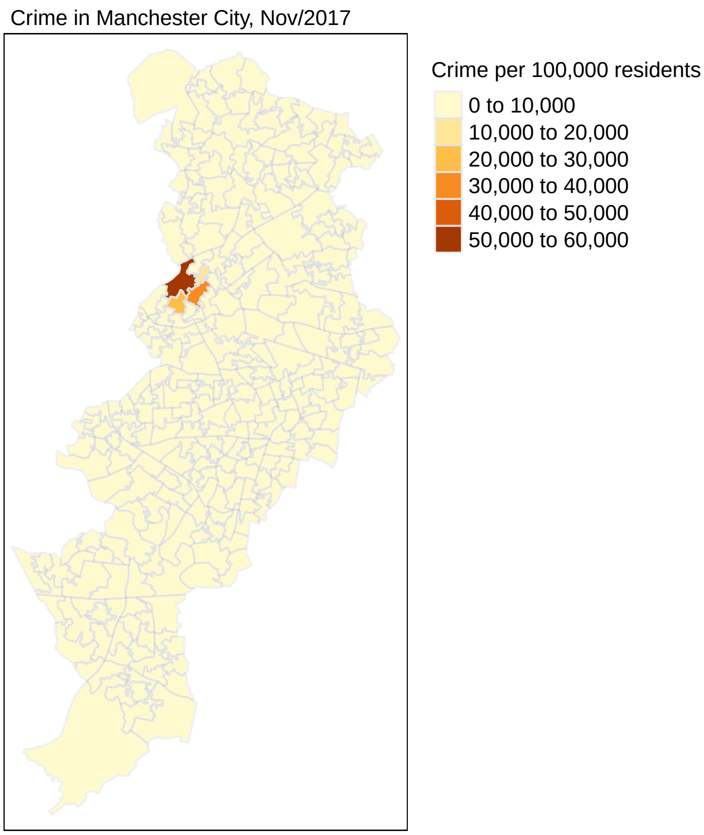


The alpha parameter that we are inserting within `tm_borders()` controls the transparency of the borders, we can go from 0 (totally transparent) to 1 (not transparent). You can play around with this value and see the results.

Notice as well that the legend in this map is not very informative and could be improved in terms of aesthetics. We can add a title within the `tm_fill` to clarify what count is and we can use the `tm_layout()` function to control the appearance of the legend. This later function `tm_layout` allows you to think about many of the more general cosmetics of the map.

```
tm_shape(manchester) +
  tm_fill("crimr1", title = "Crime per 100,000 residents") +
  tm_borders(alpha = 0.1) +
  tm_layout(main.title = "Crime in Manchester City, Nov/2017",
            main.title.size = 0.7 ,
```

```
            legend.outside = TRUE,  # Takes the legend outside the main map
            legend.title.size = 0.8)
```

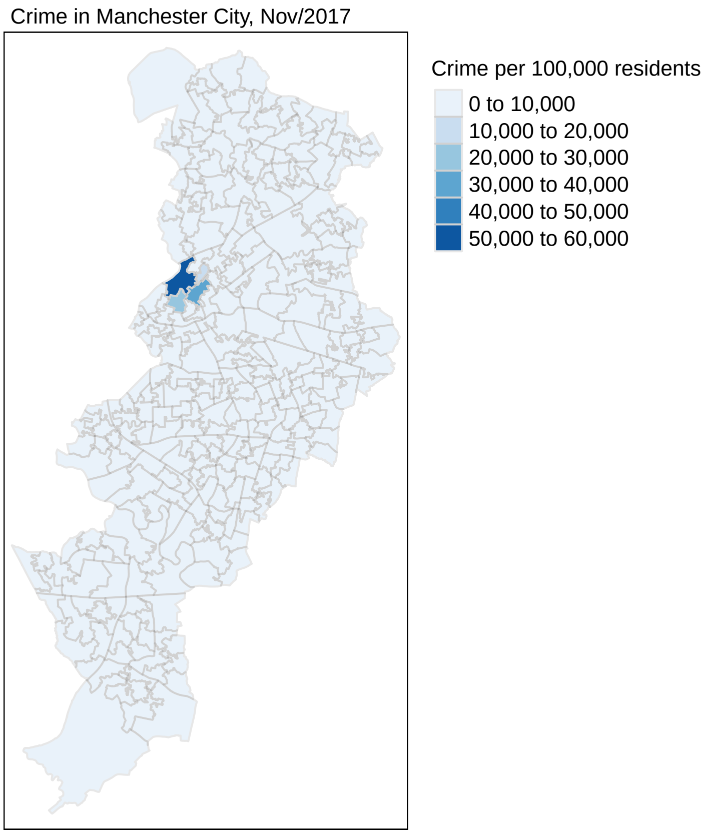


We are also going to change the current style of the maps by making them more friendly to colour blind people. We can use the `tmap_style()` function to do so.

```
current_style <- tmap_style("col_blind")
```

See how the map changes.

```
tm_shape(manchester) +
  tm_fill("crimr1", title = "Crime per 100,000 residents") +
  tm_borders(alpha = 0.1) +
  tm_layout(main.title = "Crime in Manchester City, Nov/2017",
            main.title.size = 0.7 ,
            legend.outside = TRUE,  # Takes the legend outside the main map
            legend.title.size = 0.8,
            )
```

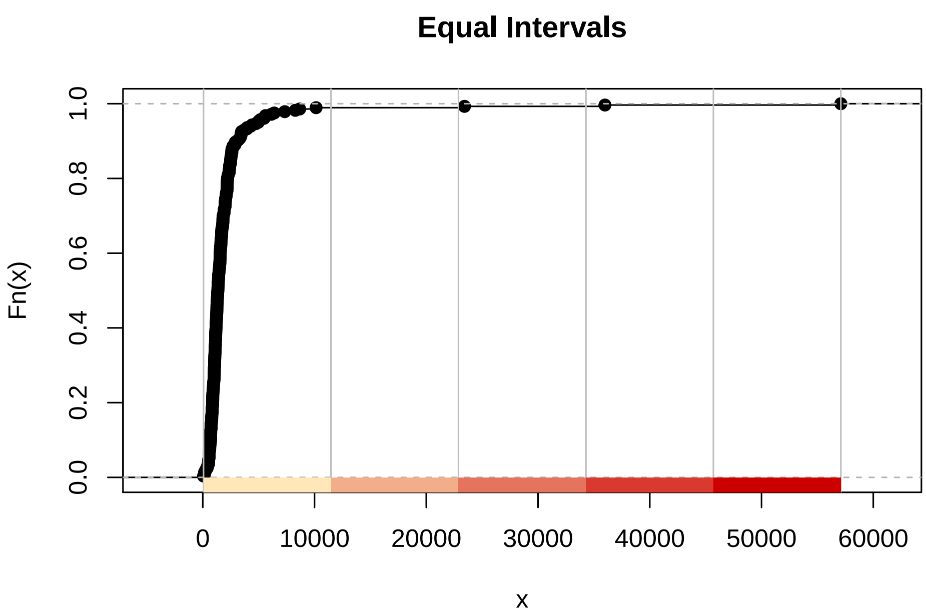


3.5 Classification systems for thematic maps

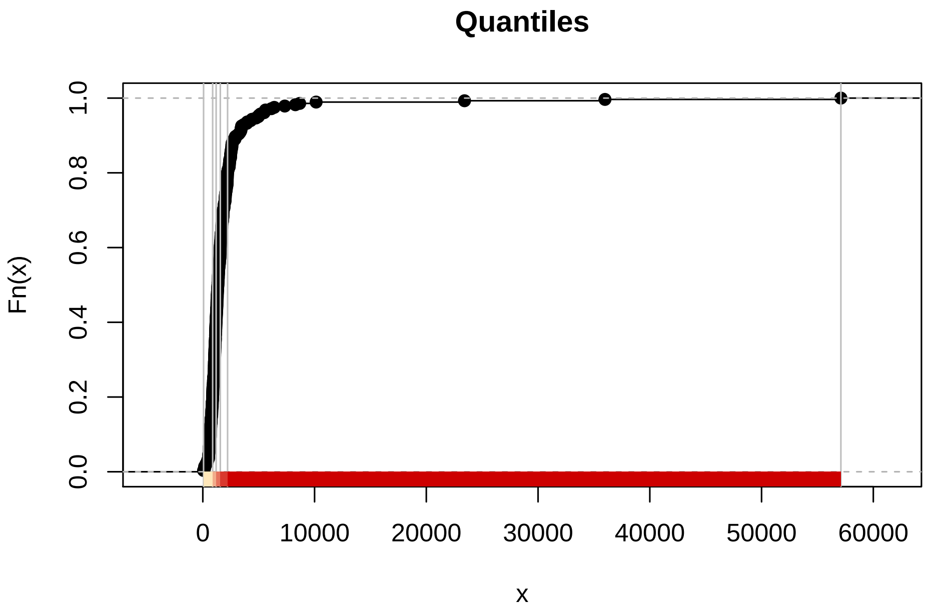
In thematic mapping, you have to make some key decisions, the most important one being how to display your data. When mapping a quantitative variable, we often have to "bin" this variable into groups. For example in the map we made below, the default binning applied was to display LSOAs grouped into those with a number of 0 to 10,000 crimes per 100,000 residents, then from 10,000 to 20,000, and so on. But why these? How were these groupings decided upon?

The quantitative information is usually classified before its symbolization in a thematic map. Theoretically, accurate classes that best reflect the distributional character of the data set can be calculated. There are different ways of breaking down your data into classes:

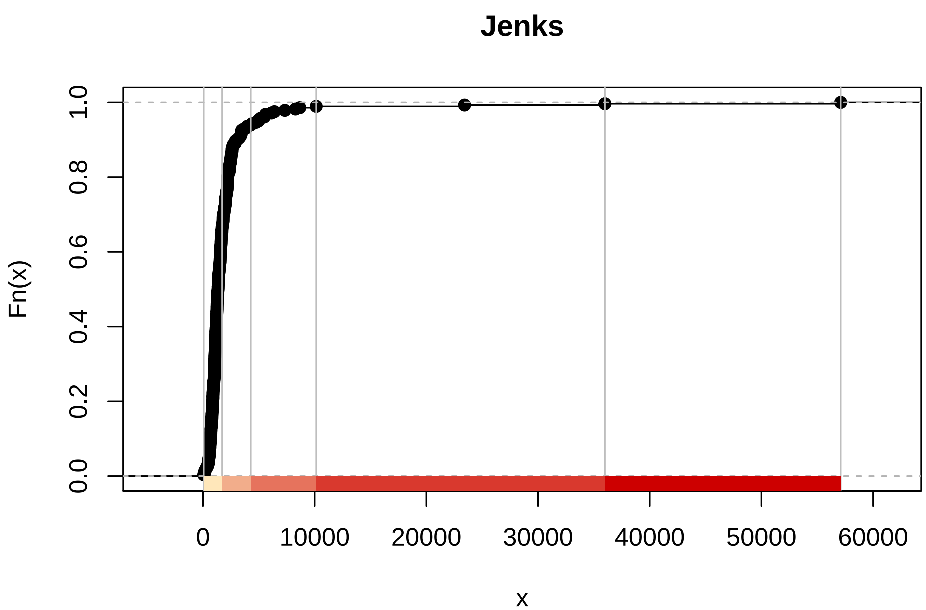
The equal interval (or equal step) classification method divides the range of attribute values into equally sized classes. What this means is that the values are divided into equal groups. Equal interval data classification subtracts the maximum value from minimum value in your plotted variable and then divides this by the number of classes to get the size of the intervals. This approach is best for continuous data. The range of the classes is structured so that it covers the same number of values in the plotted variable. With highly skewed variables this plotting method will focus our attention in areas with extreme values. When mapping crime it will create the impression that everywhere is safe, save a few locations. The graphic below shows five classes and the function shows the locations (represented by points) that fall within each class.



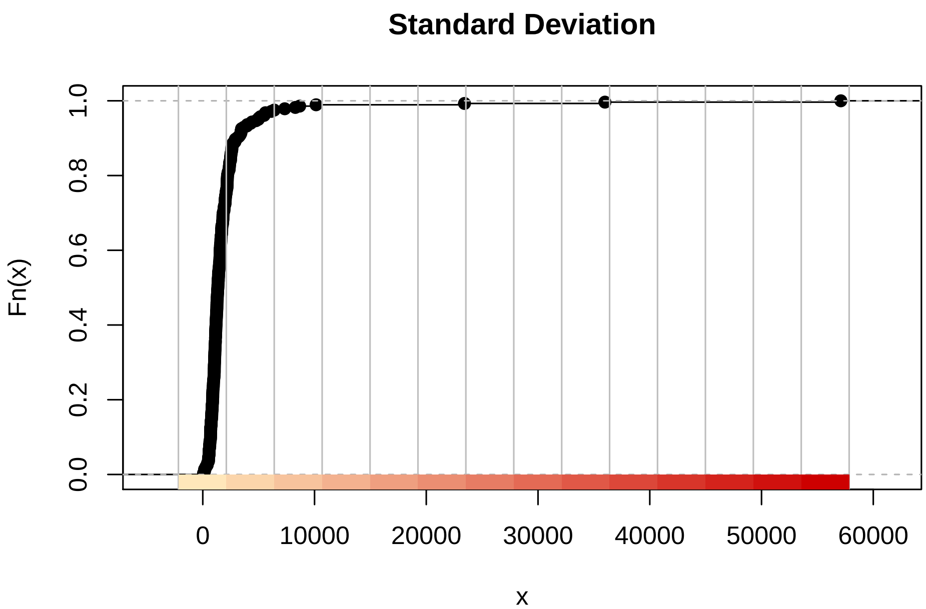
The quantile map bin the same count of features (areas) into each of its classes. This classification method places equal numbers of observations into each class. So, if you have five classes you will have 20 percent of the areas within each class. This method is best for data that is evenly distributed across its range. Here, given that our variable is highly skewed, notice how the top class includes areas with fairly different levels of crime, even if it does a better job at separating areas at the lower end of the crime rate continuum. Several authors consider that quantile maps are inappropriate to represent the skewed distributions of crime (Harries (1999))



The natural breaks (or Jenks) classification method utilizes an algorithm to group values in classes that are separated by distinct break points. It is an optimisation method which takes an iterative approach to its groupings to achieve least variation within each class. Cartographers recommend to use this method with data that is unevenly distributed but not skewed toward either end of the distribution.

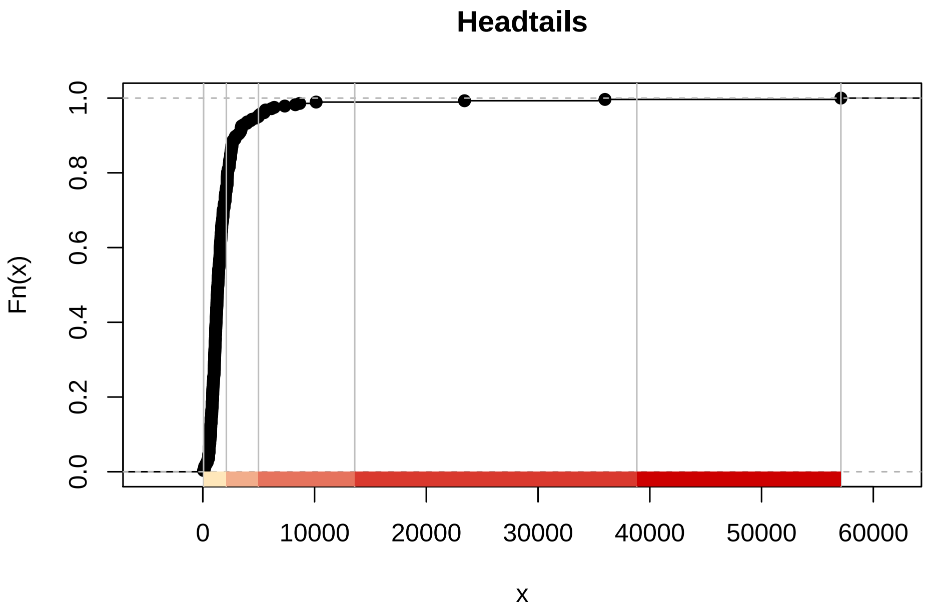


The standard deviation map uses the standard deviation (standardised measure of observations' deviation from the mean) to bin the observations into classes. This classification method forms each class by adding and subtracting the standard deviation from the mean of the dataset. It is best suited to be used with data that conforms to a normal distribution.



The headtails method This uses a method proposed by Jiang (2013) as a solution for variables with a heavy tail distribution, like we often have with crime. "This new classification scheme partitions all of the data values around the mean into two parts and continues the process iteratively for the values (above the mean) in the head until the head part values are no longer heavy-tailed distributed. Thus, the number of classes and the class intervals are both naturally determined." (Jiang (2013), p. 482)

```
pal1 <- c("wheat1", "red3")
plot(classIntervals(manchester$crimr1, style="headtails"), pal = pal1,
     main="Headtails")
```



Not only you need to choose the classification method, you also need to decide on the number of classes. This is critical. The convention in cartography is to choose between 5 to 7 classes, although some authors would say 5 to 6 (Field (2018)). Less than five and you loose detail, more than 6 or 7 and the audience looking at your map starts to have problems to perceive the differences in the symbology and to understand the spatial pattern displayed.

If you want to lie with a map, it would be very easy by using a data classification scheme that conveys the message that you want to get across. It is, thus, important that your decisions are based on good practice and are impartial.

For comparing the effects of using different methods we can use **small multiples**. Small multiples is simply a way of reproducing side by sides similar maps for comparative purposes. To be more precise small multiples are *sets of charts of the same type, with the same scale, presented together at a small size and with minimal detail, usually in a grid of some kind.* The term was at least popularized by Edward Tufte, appearing first in his *Visual Display of Quantitative Information* in 1983.

There are different ways of creating small multiples with `tmap` as you could see in the vignettes for the package, some of which are quicker but a bit more restricted. Here we are going to use `tmap_arrange()`. With `tmap_arrange()` first we need to create the maps we want and then we arrange them together.

Let's make five maps, each one using a different classification method: Equal interval, QUantile, Natural breaks (Jenks), Standard Deviation, and Headtails.

For each map, instead of visualising them one by one, just assign them to a new object. Let's call them *map1*, *map2*, *map3*, *map4*, *map5*.

So let's make *map1*. This will create a thematic map using equal intervals:

```
map1 <- tm_shape(manchester) +                   #use tm_shape function to specify spa
  tm_fill("crimr1", style="equal", title = "Equal") +  #use tm_fill to specify variabl
  tm_layout(legend.position = c("left", "top"),   #use tm_layout to make the legend lo
            legend.title.size = 0.7,
            legend.text.size = 0.5)
```

Now create *map2*, with the jenks method often preferred by geographers:

```
map2 <- tm_shape(manchester) +
  tm_fill("crimr1", style="jenks", title = "Jenks") +
  tm_layout(legend.position = c("left", "top"),
            legend.title.size = 0.7,
            legend.text.size = 0.5)
```

Now create *map3*, with the quantile method often preferred by epidemiologists:

```
map3 <- tm_shape(manchester) +
  tm_fill("crimr1", style="quantile", title = "Quantile") +
  tm_layout(legend.position = c("left", "top"),
            legend.title.size = 0.7,
            legend.text.size = 0.5)
```

Let´s make *map4*, standard deviation map, which maps the values of our variable to distance to the mean value.

```
map4 <- tm_shape(manchester) +
  tm_fill("crimr1", style="sd", title = "Standard Deviation") +
  tm_layout(legend.position = c("left", "top"),
            legend.title.size = 0.7,
            legend.text.size = 0.5)
```

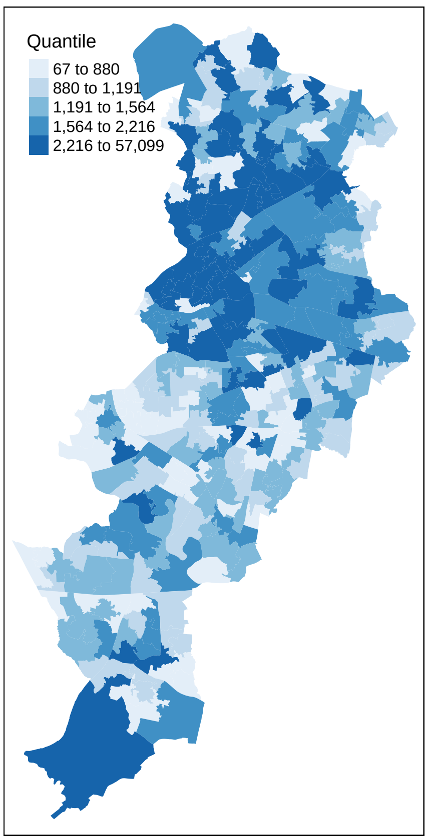
And finally let´s make *map5*, which is handy with skewed distributions.

```
map5 <- tm_shape(manchester) +
  tm_fill("crimr1", style="headtails", title = "Headtails") +
  tm_layout(legend.position = c("left", "top"),
            legend.title.size = 0.7,
            legend.text.size = 0.5)
```

Notice that we are not plotting the maps, we are storing them into R objects (map1 to map5). This way they are saved, and you can call them later, which is what we need in order to plot them together using the `tmap_arrange()` function.

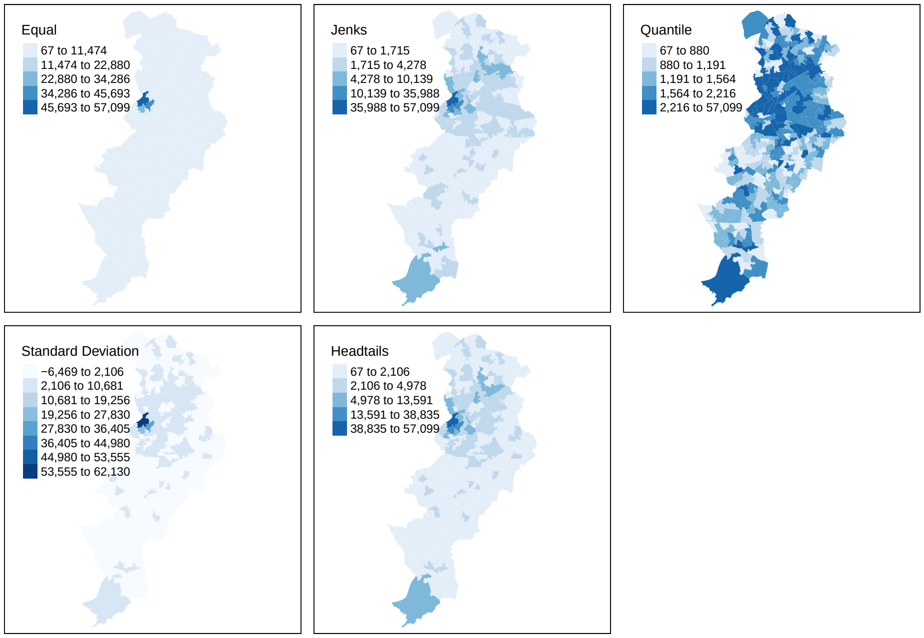
So if you wanted to map just *map3* for example, all you need to do, is call the map3 object. Like so:

```
map3
```



But now we will plot all 5 maps together, arranged using the `tmap_arrange()` function. Like so:

```
#And now we deploy tmap_arrange to plot these maps together
tmap_arrange(map1, map2, map3, map4, map5)
```



As we can see using natural breaks (as the Jenks or Headtails method do, in different ways) to classify data is useful when mapping data values that are not evenly distributed, since it places value clusters in the same class. The disadvantage of using this approach is that it is often difficult to make comparisons between maps (for example of different crimes or for different time periods) since the classification scheme used is unique to each data set.

There are some other classification methods built into `tmap` which you can experiment with if you'd like. Your discrete gradient options are "cat," "fixed," "sd," "equal," "pretty," "quantile," "kmeans," "hclust," "bclust," "fisher," "jenks," "dpih," "headtails," and "log10_pretty." A numeric variable is processed as a categorical variable when using "cat," i.e. each unique value will correspond to a distinct category.

Taken from the help file we can find more information about these, for example the "kmeans" style uses kmeans clustering technique (a form of unsupervised statistical learning) to generate the breaks. The "hclust" style uses hclust to generate the breaks using hierarchical clustering and the "bclust" style uses bclust to generate the breaks using bagged clustering. These approaches are outisde the scope of what we cover, but just keep in mind that there are many different ways to classify your data, and you must think carefully about the choice you make, as it may affect your readers' conclusions from your map.

Imagine you were a consultant working for one of the political parties in the city of Manchester. Which map would you choose to represent to the electorate the situation of crime in the city if you were the party in control of the local government and which map you would choose if you were working for the opposition? As noted above, it is very easy to mislead with maps and, thus, this means the professional map maker has to abide by strict deontological criteria and take well justified impartial decisions when visualising data.

(**Cameron_2005?**) suggests the following: >“To know which classification scheme to use, an analyst needs to know how the data are distributed and the mapping objective. If the data are unevenly distributed, with large jumps in values or extreme outliers, and the analyst wants to emphasize clusters of observations that house similar values, use the natural breaks classification approach. If the data are evenly distributed and the analyst wants to emphasize the percentage of observations in a given classification category or group of categories, use the quantile classification approach. If the data are normally distributed and the analyst wants to represent the density of observations around the mean, use the equal interval approach. If the data are skewed and the analyst wants to identify extreme outliers or clusters of very high or low values, use the standard deviation classification approach.”

3.6 Interactive mapping with tmap

So far we have been producing static maps with `tmap`. But this package also allows for interactive mapping by linking with leaflet. To change whether the plotted maps are static or interactive we need to use the `tmap_mode()` function. The default is `tmap_mode("plot")`, which corresponds to static maps. If we want to change to interactive display we need to change the argument we pass to `tmap_mode("view")`.

```
tmap_mode("view")
```

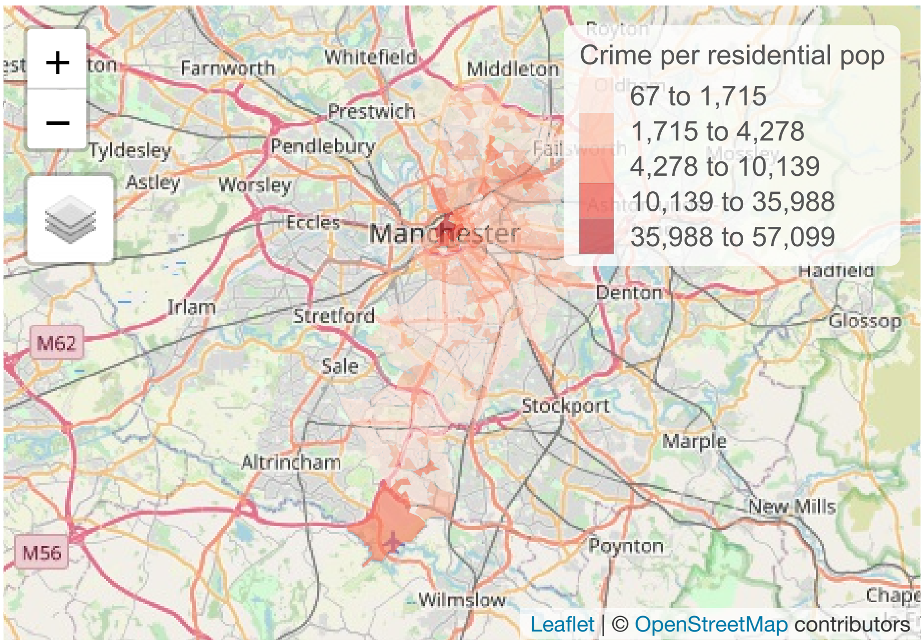
```
## tmap mode set to interactive viewing
```

When you use `tmap`, R will remember the mode you want to use. So once you specify `tmap_mode("view")`, all the subsequent maps will be interactive. It is only when you want to change this behaviour that you would need another `tmap_mode()` call. When using the interactive view we can also add a basemap with the `tm_basemap()` function and passing as an argument a particular source for the basemap. Here we specify OpenStreetMap, but there are many other choices (for a complete listing and preview see http://leaflet-extras.github.io/leaflet-providers/preview).

Let´s explore the distribution of the two alternative definitions of crime rates with in an interactive way.

```
tm_shape(manchester) +
  tm_fill("crimr1", style="jenks", palette= "Reds",
```

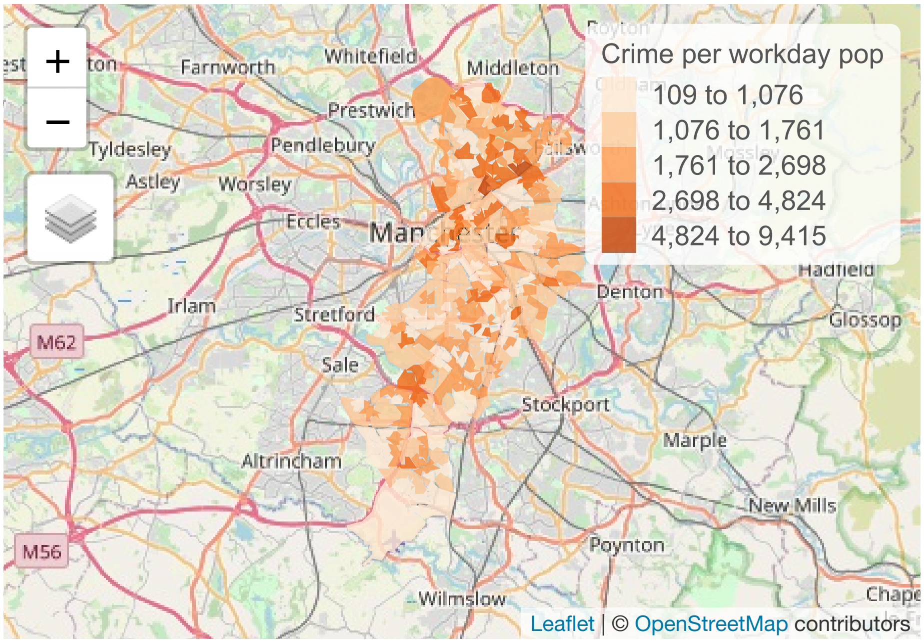
```
          title = "Crime per residential pop", alpha = 0.6,
          ) +
  tm_basemap(leaflet::providers$OpenStreetMap)
```



If we scroll down we can see that the crime rate is highest in the city centre of Manchester, but there are also pockets of high level of the crime rate in the North East of the city, Harpurhey and Moston (areas of high levels of deprivation) and in the LSOA farthest to the South (where the international airport is located).

What happens if we use the crime rate that uses the workday population?

```
tm_shape(manchester) +
  tm_fill("crimr2", style="jenks", palette= "Oranges",
          title = "Crime per workday pop", alpha = 0.8,
          ) +
  tm_basemap(leaflet::providers$OpenStreetMap)
```



Things look different, don´t they? For starters look at the values in the labels for the various classes. They are much less extreme. One of the reasons why we see such extreme rates in the first map is linked to the very large discrepancy between the residential population and the workday population in some parts of Manchester, like the international airport and the city centre (that attract very large volume of visitors). The LSOA with the highest crime rate (when using the residential population as the denominator) is E01033658.

```
filter(manchester, code=="E01033658")
```

```
## Simple feature collection with 1 feature and 7 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 383304.1 ymin: 397889 xmax: 384344.6 ymax: 399194.4
## Projected CRS: OSGB 1936 / British National Grid
##        code count tothouse respop wkdpop                       geometry crimr1
## 1 E01033658   744      711   1303  42253 POLYGON ((384273.9 398999.7...  57099
##     crimr2
## 1 1760.822
```

We can see in this area there were 744 crime incidents for a residential population of 1303, but the workday population is as high as 42253 people. So, of course, using these different denominators is bound to have an impact in the resulting rate. As noted in criminology, using as a denominator some measure of the population at risk is most appropriate.

As noted by (**Cameron_2005?**): >"In the final analysis, although choropleth maps are very useful for visualizing spatial distributions, using them for hot spot analyses of crime has certain disadvantages. First, attention is often focused on the relative size of an area, so large areas tend to dominate the map. Second, choropleth maps involve the aggregation of data within statistical or administrative areas that may not correspond to the actual underlying spatial distribution of the data."

3.7 Smoothing rates: adjusting for small sample noise

In previous sections we discussed how to map rates. It seems a fairly straightforward issue, you calculate a rate by dividing your numerator (eg: number of crimes) by an appropriately selected denominator (eg: daytime population). You get your variable with the relevant rate and you map it using a choropleth map. However, things are not always that simple.

Rates are funny animals. Gelman and Price (1999) goes so far as to suggest that all maps of rates are misleading. The problem at hand is well known in spatial epidemiology: " plotting observed rates can have serious drawbacks when sample sizes vary by area, since very high (and low) observed rates are found disproportionately in poorly-sampled areas" (Gelman and Price (1999), p. 3221). There is associated noise for those areas where the denominators give us a small sample size. And it is hard to solve this problem.

Let´s illustrate with an example. We are going to use historic data on homicide across US counties. The dataset was used as the basis for the study by (**Baller_2006?**). It contains data on homicide counts and rates for various decades across the US as well as information on structural factors often thought to be associated with violence. The data is freely available through the webpage of Geoda, a clever point-and-click interface developed by Luc Anselin (a spatial econometrician and coauthor of the above paper) and his team, to make spatial analysis accessible. It is also available as one of the packages in the `geodaData` package.

To read a data available in a library we have loaded we can use the `data()` function. If we check the `class()` of this object we will see it was already stored in `geodaData` as a `sf` object.

```
data("ncovr")
class(ncovr)
```

```
## [1] "sf"         "data.frame"
```

Let's look at the ncvor data. We can start by looking ath the homicide rate for 1960.

```
summary(ncovr$HR60)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   0.000   0.000   2.783   4.504   6.885  92.937
```

We can see that the county with the highest homicide rate in the 1960s had a rate of 92.937 homicides per 100,000 individuals. That is very high. Just to put it into context in the UK is about 0.92. Where is that place? I can tell you is a place called Borden. Check it out:

```
borden <- filter(ncovr, NAME == "Borden")
borden$HR60
```

```
## [1] 92.9368
```

Borden county (https://en.wikipedia.org/wiki/Borden_County,_Texas) in Texas. You may be thinking… "Texas Chainsaw Massacre" perhaps? No, not really. Ed Gein, who inspired the film, was based and operated in Wisconsin. Borden claim to fame is rather more prosaic: it was named after Gail Borden, the inventor of condensed milk. So, what's going on here? Why do we have a homicide rate in Borden that makes it look like a war zone? Is is that it is only one of the six counties where alcohol is banned in Texas (and people are consequently going nuts?).

Check this out too:

```
borden$HC60
```

```
## [1] 1
```

What? A total homicide count of 1. How can a county with just one homicide have a rate that makes it look like the most dangerous place in the US?

```
borden$PO60
```

```
## [1] 1076
```

Well, there were about 1076 people living there.

```
summary(ncovr$PO60)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##     208    9417   18408   57845   39165 7781984
```

If you contrast that population count with the population of the average county in the US, that's tiny. One homicide in such a small place can end up producing a big rate. Remember that the rate is simply dividing the number of relevant events by the exposure variable (in this case population) and multiplying by a constant (in this case 100,000 since we expressed crime rates in those terms). Most times Borden looks like a very peaceful place:

```
borden$HR70
```

```
## [1] 0
```

```
borden$HR80
```

```
## [1] 0
```

```
borden$HR90
```

```
## [1] 0
```

It has a homicide rate of 0 in most decades. But it only takes one homicide and, bang, it goes top of the league. So a standard map of rates is bound to be noisy. There is the instability that is introduced by virtue of having areas that may be sparsely populated and in which one single event, like in this case, will produce a very noticeable change in the rate.

In fact, if you look at the counties with the highest homicide rate in the "ncovr" dataset you will notice all of them are places like Borden, areas that are sparsely populated, not because they are that dangerous, but because of the instability of rates. Conversely the same happens with those places with the lowest rate. They tend to be areas with a very small sample size.

This is a problem that was first noted by epidemiologists doing disease mapping. But a number of other disciplines have now noted this and used some of the approaches developed by public health researchers that confronted this problem when producing maps of disease (PRO TIP: techniques and approaches used by spatial epidemiologists are very similar to those used by criminologists -in case you ever think of changing careers or need inspiration for how to solve a crime analysis problem).

One way of dealing with this is by **smoothing** or **shrinking** the rates. This basically as the word implies aims for a smoother representation that avoids hard spikes associated with random noise. There are different ways of doing that. Some ways use a non-spatial approach to smoothing, using something called a **empirical bayesian smoother**. How does this work? This approach takes the raw rates and tries to "shrunk" them towards the overall average.

What does this mean? Essentially, we compute a weighted average between the raw rate for each area and the global average across all areas, with weights proportional to the underlying population at risk. What this procedure does is to have the rates of smaller areas (those with a small population at risk) to have their rates adjusted considerably (brought closer to the global average), whereas the rates for the larger areas will barely change.

Here we are going to introduce the approach implemented in `DCluster`, a package developed for epidemiological research and detection of [clusters of disease.

```
res <- empbaysmooth(ncovr$HC60, ncovr$PO60)
```

In the new object we generate, which is a list, you have an element which contains the computed rates. We can add those to our dataset:

```
ncovr$HR60EBS <- res$smthrr * 100000
```

Instead of shrinking to the global rate, we can shrink to a rate based on the neighbours of each county. Shrinking to the global rate ignores the spatial dimension of the phenomenon being mapped out and may mask existing heterogeneity. If instead of shrinking to a global rate, we shrink to a local rate, we may be able to take unobserved heterogeneity into account. Marshall (1991) proposed a local smoother estimator in which the crude rate is shrunk towards a loca, "neighbourhood," rate. To compute this we need the list of neighbours that surround each county (we will discuss this code in the Chapter on Spatial Dependence, so for now just trust us we are computing the rate of the areas that surround each country):

```
ncovr_sp <- as(ncovr, "Spatial")
w_nb <- poly2nb(ncovr_sp, row.names=ncovr_sp$FIPSNO)
eb2 <- EBlocal(ncovr$HC60, ncovr$PO60, w_nb)
ncovr$HR60EBSL <- eb2$est * 100000
```

We can now plot the maps and compare them:

```
tmap_mode("plot")
```

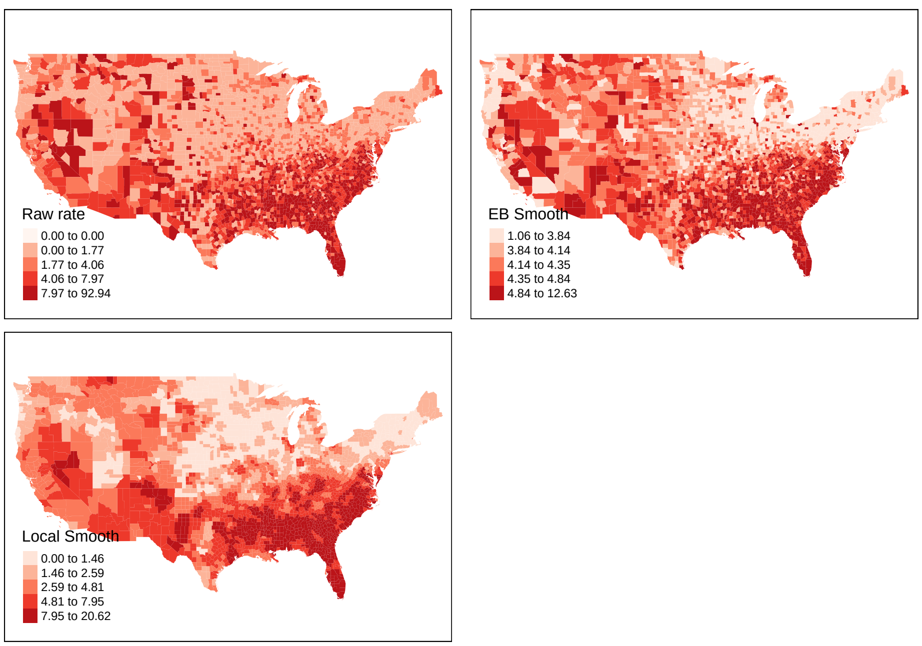
```
## tmap mode set to plotting
```

```
current_style <- tmap_style("col_blind")
```

```
## tmap style set to "col_blind"
```

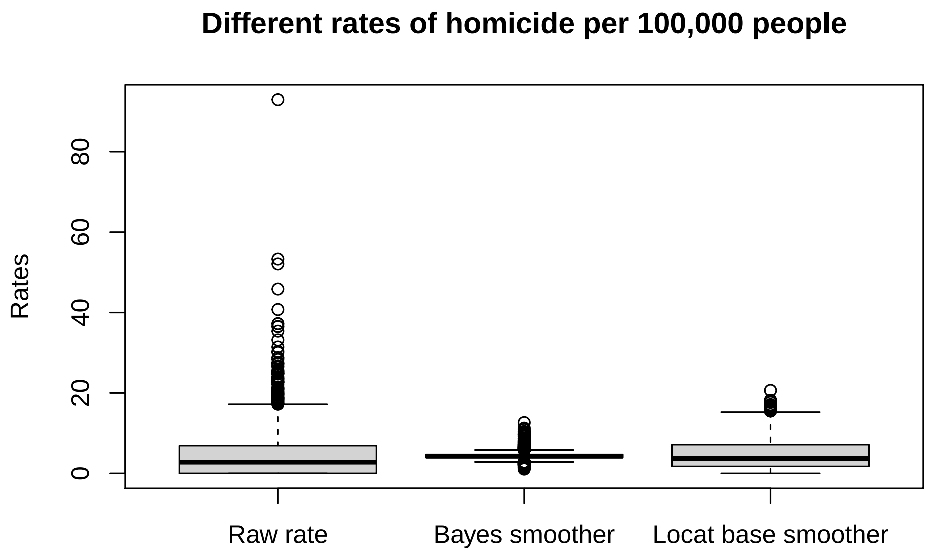
```
## other available styles are: "white", "gray", "natural", "cobalt", "albatross", "beaver", "bw",
```

```
map1<- tm_shape(ncovr) +
  tm_fill("HR60", style="quantile", title = "Raw rate", palette = "Reds") +
  tm_layout(legend.position = c("left", "bottom"),
            legend.title.size = 0.8,
            legend.text.size = 0.5)

map2<- tm_shape(ncovr) +
  tm_fill("HR60EBS", style="quantile", title = "EB Smooth", palette = "Reds") +
  tm_layout(legend.position = c("left", "bottom"),
            legend.title.size = 0.8,
            legend.text.size = 0.5)

map3<- tm_shape(ncovr) +
  tm_fill("HR60EBSL", style="quantile", title = "Local Smooth", palette = "Reds") +
  tm_layout(legend.position = c("left", "bottom"),
            legend.title.size = 0.8,
            legend.text.size = 0.5)

tmap_arrange(map1, map2, map3)
```



Notice that the quantiles are not the same, so that will make your comparison difficult. Let´s look at a boxplot of these variables. In the map of raw rates we have the most variation.

```
#Boxplots with base R graphics
boxplot(ncovr$HR60, ncovr$HR60EBS, ncovr$HR60EBSL,
        main = "Different rates of homicide per 100,000 people",
        names = c("Raw rate", "Bayes smoother", "Locat base smoother"),
        ylab = "Rates")
```



The range for the raw rates is nearly 93. Much of the variation in observed homicide rates by county is attributable to statistical noise due to the small number of (observed and expected) homicides in low-population counties. Because of this noise, a disproportionate fraction of low-population counties are observed to have extremely high (or low) homicide rates when compared to typical counties in the United States. The second distribution shows that everything has been brought closer to the main mean, whereas the local smoother allows for a bit more variation. The general Bayes smoother has the opposite problem of mapping out the raw rates, it likely hides existing spatial variation. If you contrast the maps you will see how this results in a clearer and smoother spatial pattern for the rate that is estimated borrowing information from their neighbours.

So to smooth or not too smooth? Clearly we can see how smoothing stabilises the rates and removes noise. But as Gelman and Price (1999) suggests this introduces other artifacts and autocorrelation into our estimates. Some people are also not too keen on maps of statistically adjusted estimates. Yet, the conclusions one can derive from mapping raw rates (*when* the denominator varies significantly and we have areas with small sample size) means that smoothing is often a preferable alternative (Waller and Gotway (2004)). The problem we have with maps of estimates is that we need information about the variability and it is hard to map this out in a convenient way (Gelman and Price (1999)).

Lawson (2021b) (p. 38), in relation to the similar problem of disease mapping, suggests that "at the minimum any map of relative risk for a disease should be accompanied with information pertaining to estimates of rates within each region as well as estimates of variability within each region" whereas "at the other extreme it could be recommended that such maps be only used as a presentational aid, and not as a fundamental decision-making tool."

3.8 Further reading

This chapter introduced some basic principles of thematic maps. We learned how to make them using the `tmap` package, we learned about the importance of classification schemes, and how each one may produce a different looking map, which may tell a different story. For further reading, (**Brewer_2016?**) provides a brief condensed introduction to thematic mapping for epidemiologists but that can be generally extrapolated for crime mapping purposes. We have talked about the potential to develop misleading maps and Monmonier (1996) "How to lie with maps" provides good guidance to avoid that our negligent choices when producing a map confuse the readers. Carr and Pickle (2010) offers a more detailed treatment of small multiples and micromaps.

Mapping rates has been more thoroughly discussed within spatial epidemiology than in criminology. There is ample literature on disease mapping that address in more sophisticated ways some of the issues we introduce here (see Waller and Gotway (2004), Lawson (2021b), Lawson (2021a), or Lawson (2021c)). Much of this work on spatial epidemiology adopts a Bayesian framework. We will talk a bit more about this later on, but if you want a friendly introduction to Bayesian data analysis there is no better introduction than (**McElreath_2020?**).

Chapter 4

Chapter 4: Variations of thematic mapping

(WE NEED TO INTRODUCE BIVARIATE THEMATIC MAPS: https://timogrossenbacher.ch/2019/04/bivariate-maps-with-ggplot2-and-sf/)

(AND DASYMETRIC MAPS) https://www.r-bloggers.com/2021/02/bivariate-dasymetric-map/

4.1 Introduction

In this chapter are going to discuss some additional features around thematic maps. Specifically, we will address some of the problems we confront when we are trying to use use choropleth maps, as well as some alternatives to point based maps. We will also briefly introduce the modifiable area unit problem.

The main objectives for this chapter are that by the end you will have:

- explored **binning** as an alternative to point maps
- gained an insight into the **Modifiable Areal Unit Problem**
- been introduced to alternative visualisations such as
  - **cartograms**,
  - **bi-variate** thematic maps, and
  - **dasymmetric** maps.

In this chapter, we will be making use of the following libraries:

```
# Packages for reading data and data carpentry
library(readr)
library(dplyr)
library(tidyr)

# Packages for handling spatial data and for geospatial carpentry
library(sf)

# Packages for mapping and visualisation
library(ggplot2)
library(ggspatial)
library(cartogram)
```

4.2 Binning points

In GIS it is often difficult to present point-based data because in many instances there are several different points and data symbologies that need to be shown. As the number of different data points grows they can become complicated to interpret and manage which can result in convoluted and sometimes inaccurate maps. This becomes an even larger problem in web maps that are able to be depicted at different scales because smaller scale maps need to show more area and more data. This makes the maps convoluted if multiple data points are included.

In many maps there are so many data points included that little can be interpreted from them. In order to reduce congestion on maps many GIS users and cartographers have turned to a process known as binning.

> **Binning** is defined as the process of grouping pairs of locations based on their distance from one another. These points can then be grouped as categories to make less complex and more meaningful maps.

Researchers and practitioners often require a way to systematically divide a region into equal-sized portions. As well as making maps with many points easier to read, binning data into regions can help identify spatial influence of neighbourhoods, and can be an essential step in developing systematic sampling designs.

This approach to binning generates an array of repeating shapes over a user-specified area. These shapes can be hexagons, squares, rectangles, triangles, circles or points, and they can be generated with any directional orientation.

4.2.1 The Binning Process

Binning is a data modification technique that changes the way data is shown at small scales. It is done in the pre-processing stage of data analysis to convert the original data values into a range of small intervals, known as a bin. These bins are then replaced by a value that is representative of the interval to reduce the number of data points.

Spatial binning (also called *spatial discretization*) discretizes the location values into a small number of groups associated with geographical areas or shapes. The assignment of a location to a group can be done by any of the following methods: - Using the coordinates of the point to identify which "bin" it belongs to. - Using a common variable in the attribute table of the bin and the point layers.

4.2.2 Different Binning Techniques

Binning itself is a general term used to describe the grouping of a dataset's values into smaller groups (Johnson, 2011). The bins can be based on a variety of factors and attributes such as spatial and temporal and can thus be used for many different projects.

4.2.2.1 Choropleth maps

You might be thinking, "grouping points into a larger spatial unit, haven't we already done this when making choropleth maps?" In a way you are right. Choropleth maps are another type of map to that uses binning. Proportional symbol and choropleth maps group similar data points together to show a range of data instead of many individual points. We've covered this extensively, and is often the best approach to consider spatial grouping of your point variables, because the polygons (shapes) to which you are aggregating your points are *meaningful*. You can group into LSOAs because you want to show variation in neighbourhoods. Or you can group into police force areas because you want to look at differences between those units of analysis. But sometimes there is just not a geography present to meet your needs.

Let's say you are conducting some days of action in Manchester city centre, focusing on antisocial behaviour. You are going to put up some information booths and staff them with officers to engage with the local population about antisocial behaviour. For these to be most effective, as an analyst you decide that they should go into the areas with the highest *count* of antisocial behaviour. You want to be very specific about where you put these as well, and so LSOA level would be too broad, you want to zoom in more. One approach can be to split central Manchester into some smaller polygons, and just calculate the number of antisocial behaviour incidents recorded in each. That way you can then decide to put your information booths somewhere inside the top 5 highest count bins.

4.2.2.2 Rectangular binning

The aggregation of incident point data to regularly shaped grids is used for many reasons such as normalizing geography for mapping or to mitigate the issues of using irregularly shaped polygons created arbitrarily (such as county boundaries or block groups that have been created from a political process). Regularly shaped grids can only be comprised of equilateral triangles, squares, or hexagons, as these three polygon shapes are the only three that can tessellate (repeating the same shape over and over again, edge to edge, to cover an area without gaps or overlaps) to create an evenly spaced grid.

Rectangular binning is the simplest binning method and as such it heavily used. However, there are some reasons why rectangular bins are less preferable over hexagonal bins. Before we cover this, let's have a look at hexagonal bins.

4.2.2.3 Hexagonal binning

In many applications binning is done using a technique called **hexagonal binning**. This technique uses hexagon shapes to create a grid of points and develops a spatial histogram that shows different data points as a range or group of pairs with common distances and directions. In hexagonal binning the number of points falling within a particular rectangular or hexagon in a gridded surface is what makes the different colors to easily visualize data (Smith, 2012). Hexagonnal binning was first developed in 1987 and today "hexbinning" is conducted by laying a hexagonal grid on top of 2-dimensional data (Johnson, 2011). Once this is done users can conduct data point counts to determine the number of points for each hexagon (Johnson, 2011). The bins are then symbolized differently to show meaningful patterns in the data.

So how can we use hexbinning to solve our antisocial behaviour days of action task? Well let's say we split Manchester city centre into hexagons, and count the number of antisocial behaviour instances in these. We can then identify the top hexagons, and locate our booths somewhere within these.

First make sure you have the appropriate packages loaded. Also let's get some data. You could go and get this data yourself from police.uk, we've been through all the steps for downloading data from there a few times now. But for now, we have a tidied set of data ready for you. This data is one year's worth of antisocial behaviour from the police.uk data, from May 2016 to May 2017, for the borough of Manchester.

The data are included in the supplementary material, and also available on our GitHub page.

```
manchester_asb <- read_csv("https://raw.githubusercontent.com/maczokni/crime_mapping/master/data/
```

```
## Warning: Missing column names filled in: 'X1' [1]

##
## -- Column specification --------------------------------------------------------
## cols(
##   X1 = col_double(),
##   X = col_double(),
##   Crime.ID = col_logical(),
##   Month = col_character(),
##   Reported.by = col_character(),
##   Falls.within = col_character(),
##   Longitude = col_double(),
##   Latitude = col_double(),
##   Location = col_character(),
##   LSOA.code = col_character(),
##   LSOA.name = col_character(),
##   Crime.type = col_character(),
##   Last.outcome.category = col_logical(),
##   Context = col_logical(),
##   borough = col_character()
## )
```

This is currently just a text dataframe, so we need to let R know that actually this is a spatial object, who's geometry can be find in its longitude and latitude coordinates. As we have long/lat we can assure it's in WGS 84 projection.

```
ma_spatial <- st_as_sf(manchester_asb, coords = c("Longitude", "Latitude"),
                 crs = 4326, agr = "constant")
```

Now one thing that this does is it consumes our Long and Lat columns into a geometry attribute. This is generally OK, but for the binning we will do, we would like to have them as separate coordinates. To do this, we can use the `st_coordinates()` function from the **sf** package. This function extracts the longitude and latitude from the geometry within the sf object, in this case "ma_spatial" object. For example, if we look at the first row:

```
ma_spatial %>%
  slice(1) %>%
  st_coordinates()
```

```
##           X        Y
## 1 -2.228809 53.53493
```

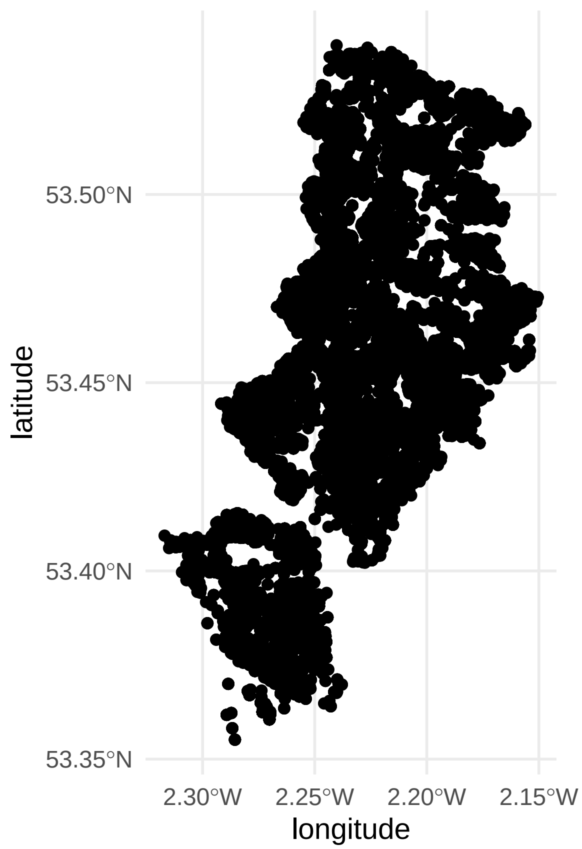
We have our longitude (X) and latitude (Y). We can select the first and second element of this to get only one or the other. To go through our whole dataframe, we can use the `mutate()` function, and assign each element to a longitude and latitude column respectively:

```
ma_spatial <- ma_spatial %>%
  mutate(longitude = st_coordinates(.)[,1],
         latitude = st_coordinates(.)[,2])
```

As a first step, we can plot asb in the borough of Manchester using simple ggplot, as demonstrated in Chapter 1.

We can plot our points first:

```
ggplot(ma_spatial, aes(x = longitude, y = latitude)) +
  geom_sf() +
  theme_minimal()
```

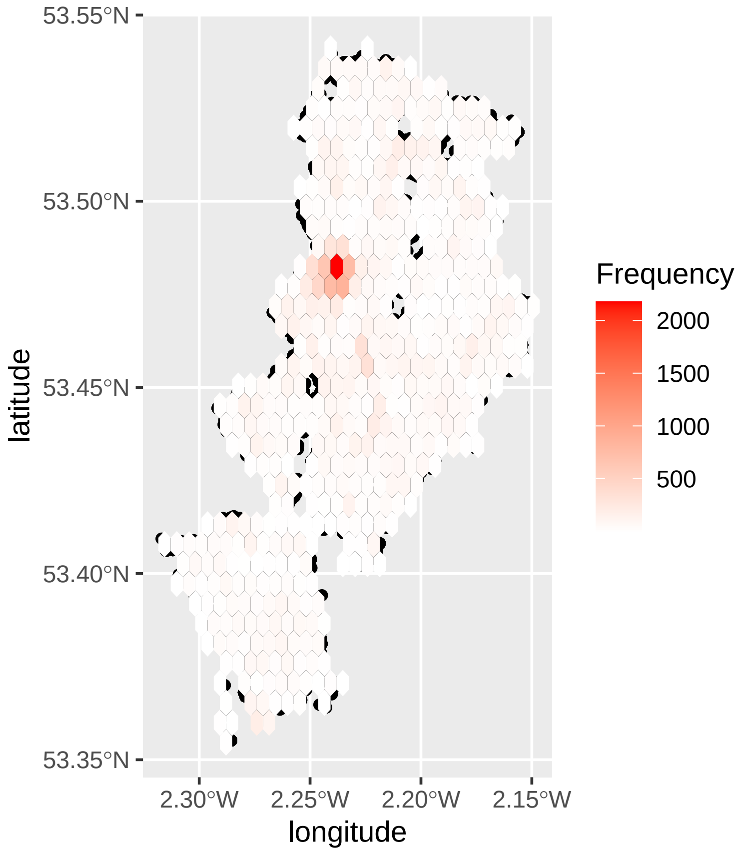


We see our nice map of Manchester, as outlined by ASB across the Local Authority.

To create a bin of this map, what we do, is generate a grid of tessellating map of our desired shape, and then count the number of points which fall into each one of these shapes. We can use ‘`ggplot` for this. It is such a great tool for building visualisations, because you can apply whatever geometry best suits your data. So for us to just have a look at the hexbinned version of our point data of antisocial behaviour, we can use the `stat_binhex()` function as a layer on our ggplot object. We can also recreate the thematic map element, as we can use the frequency of points in each hex to shade each hexbin from white (least number of incidents) to red (most number of incidents).

So let’s have a go:

```
ggplot(ma_spatial, aes(longitude, latitude)) +                         #define data and variables
  geom_sf() +
  stat_binhex() +                                                         #add binhex layer (hexb
  scale_fill_gradientn(colours = c("white","red"), name = "Frequency")    #add shading based on n
```

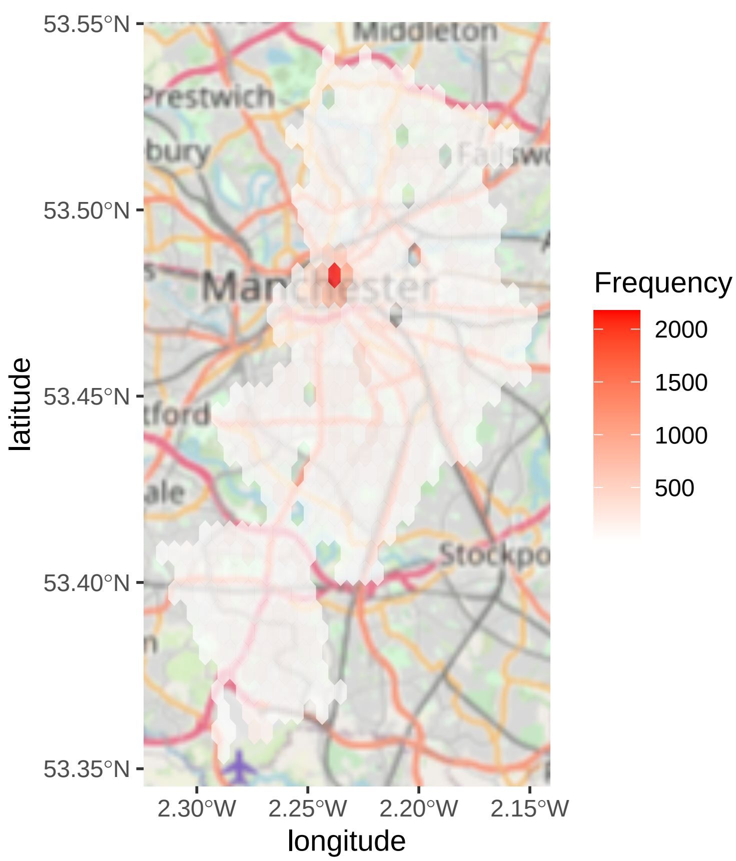


Neat, but doesn’t quite tell us *where* that really dark hexbon actually is. So it would be much better if we could do this with a basemap as the background, rather than our grey `ggplot` theme.

Now, we can apply the same code as we used above, for the `ggplot`, to this ggmap, to add our hexbins on top of this basemap. For this we use the function `annotation_map_tile()` from `ggspatial`:

```
ggplot(ma_spatial, aes(x = longitude, y = latitude)) +
  annotation_map_tile() +
  stat_binhex(alpha=0.7) +                                                           #ad
  scale_fill_gradientn(colours = c("white","red"), name = "Frequency")     #add shading
```

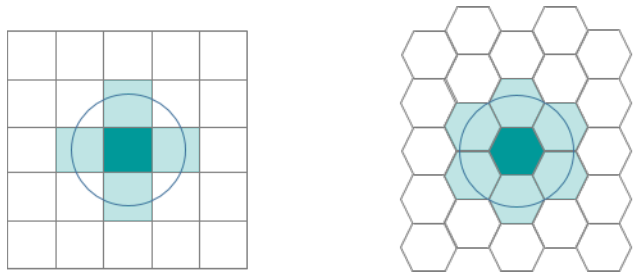
```
## Zoom: 10
```



Adding this basemap provides us with a bit more context. And combined with the hexbin map, it is much easier to see where in Manchester borough Antisocial Behaviour concentrates (as opposed to with the point map!).

Above we used a hexagon shape for our binning, however you might choose other shapes as well. I will illustrate in a moment the approach to use rectangular binning, but first, I want to highlight why hexagon might still be your ideal choice. Here are some thoughts:

- Hexagons reduce sampling bias due to edge effects of the grid shape. The edge effects of bounded space refers to the problem of truncated data that can skew the results of subsequent analyses (we'll get to this in the next section). This is related to the low perimeter-to-area ratio of the shape of the hexagon. A circle has the lowest ratio but cannot tessellate to form a continuous grid. Hexagons are the most circular-shaped polygon that can tessellate to form an evenly spaced grid.
- This circularity of a hexagon grid allows it to represent curves in the patterns of your data more naturally than square grids.
- When comparing polygons with equal areas, the more similar to a circle the polygon is, the closer to the centroid the points near the border are (especially points near the vertices). This means that any point inside a hexagon is closer to the centroid of the hexagon than any given point in an equal-area square or triangle would be (this is due to the more acute angles of the square and triangle versus the hexagon).
- Hexagons are preferable when your analysis includes aspects of connectivity or movement paths. Due to the linear nature of rectangles, fishnet grids can draw our eyes to the straight, unbroken, parallel lines which may inhibit the underlying patterns in the data. Hexagons tend to break up the lines and allow any curvature of the patterns in the data to be seen more clearly and easily. This breakup of artificial linear patterns also diminishes any orientation bias that can be perceived in fishnet grids.
- If you are working over a large area, a hexagon grid will suffer less distortion due to the curvature of the earth than the shape of a fishnet grid.
- Finding neighbors is more straightforward with a hexagon grid. Since the edge or length of contact is the same on each side, the centroid of each neighbor is equidistant. However, with a fishnet grid, the Queen's Case (above/below/right/left) neighbor's centroids are N units away, while the centroids of the diagonal (Rook) neighbors are farther away (exactly the square root of 2 times N units away).
- Since the distance between centroids is the same in all six directions with hexagons, if you are using a distance band to find neighbors or are using the Optimized Hot Spot Analysis, Optimized Outlier Analysis or Create Space Time Cube By Aggregating Points tools, you will have more neighbors included in the calculations for each feature if you are using hexagonal grid as opposed to a fishnet grid.

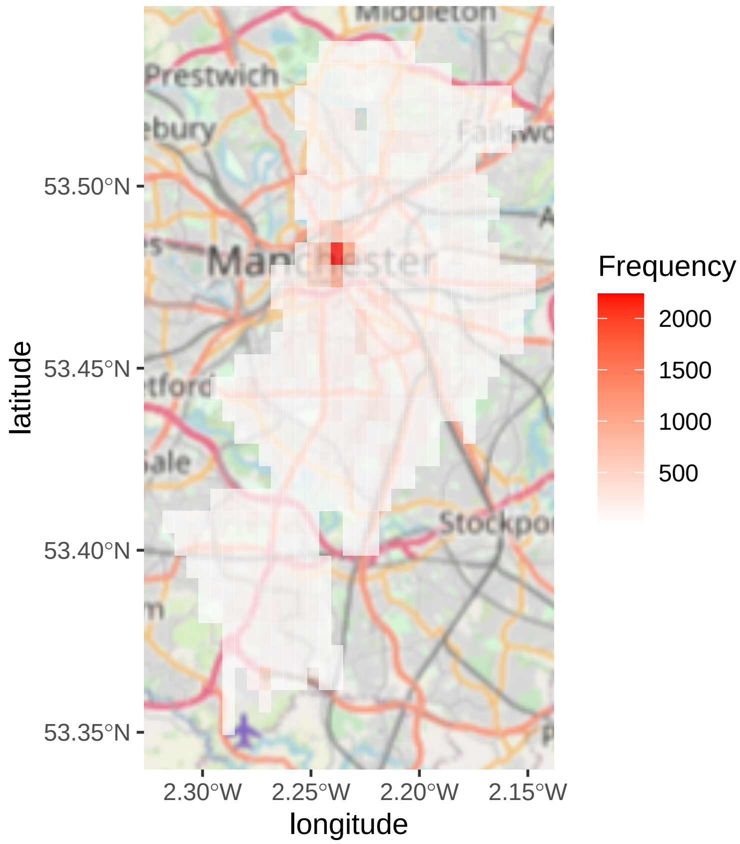


To illustrate the differences of different approaches, let's see what this map would look like with:

a) rectangular binning:

```
ggplot(ma_spatial, aes(x = longitude, y = latitude)) +
  annotation_map_tile() +
  stat_bin2d(alpha=0.7) +
  scale_fill_gradientn(colours = c("white","red"),
                       name = "Frequency")
```

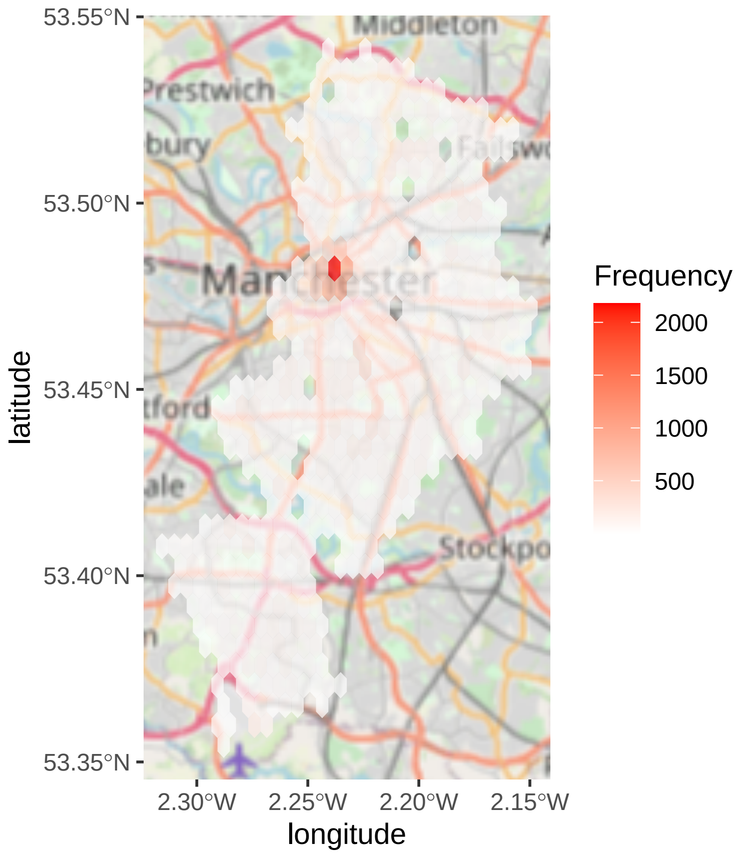
```
## Zoom: 10
```



b) hexagonal binning:

```
ggplot(ma_spatial, aes(x = longitude, y = latitude)) +
  annotation_map_tile() +
  stat_binhex(alpha=0.7) +
  scale_fill_gradientn(colours = c("white","red"),
                       name = "Frequency")
```

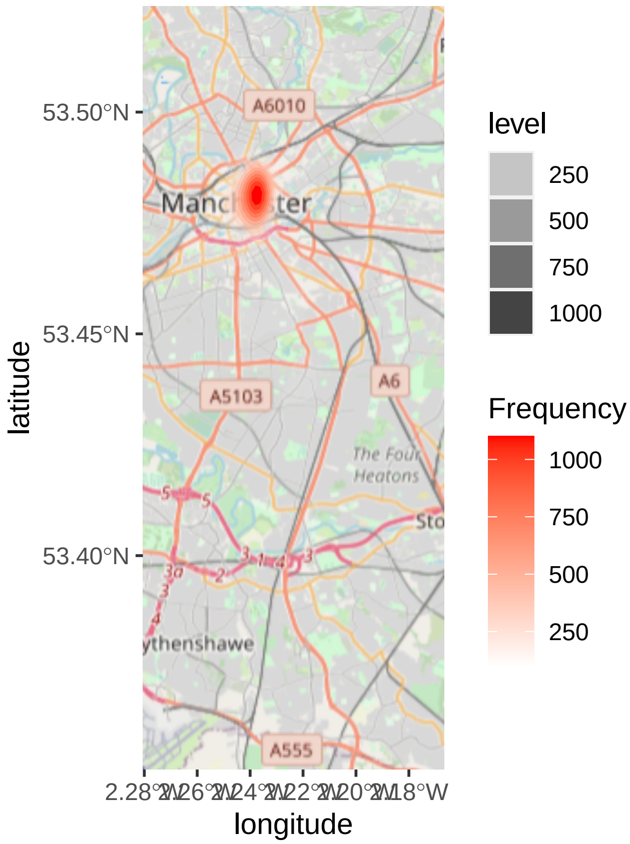
```
## Zoom: 10
```



c) a simple "heatmap" (we will discuss these more thoroughly in Chapter 6):

```
ggplot(ma_spatial, aes(x = longitude, y = latitude)) +
  annotation_map_tile() +
  stat_density2d(aes(fill = ..level.., # value corresponding to discretized density es
                     alpha = ..level..),
                 geom = "polygon") +  # creates the bands of differenc colors
  ## Configure the colors, transparency and panel
  scale_fill_gradientn(colours = c("white","red"),
                       name = "Frequency")
```

```
## Zoom: 11
```



4.2.3 Benefits of Binning

Because of the plethora of data types available and the wide variety of projects being done in GIS, binning is a popular method for mapping complex data and making it meaningful. Binning is a good option for map makers as well as users because it makes data easy to understand and it can be both static and interactive on many different map scales. If every different point were shown on a map it would have to be a very large scale map to ensure that the data points did not overlap and were easily understood by people using the maps.

According to Kenneth Field, an Esri Research Cartographer:

> "Data binning is a great alternative for mapping large point-based data sets which allows us to tell a better story without interpolation. Binning is a way of converting point-based data into a regular grid of polygons so that each polygon represents the aggregation of points that fall within it."

By using binning to create categories of data maps are easier to understand, more accurate and more visually appealing. Hexbin plots can be viewed as an alternative to scatter plots. The hexagon-shaped bins were introduced to plot densely packed sunflower plots. They can be used to plot scatter plots with high-density data.

4.3 Transforming polygons

When you have meaningful spatial units of analysis in your polygons, for example you are interested specifically in Local Autorities, it might make sense to stick with what we did last week, and aggregate the points into these polygons to create thematic maps. However, while thematic maps are an accessible and visually appealing method for displaying spatial information, they can also be highly misleading. Irregularly shaped polygons and large differences in the size of areas being mapped can introduce misrepresentation. The message researchers want to get across might be lost, or even worse, misdirect the viewers to erroneous conclusions. Field and Dorling (2016) provide a helpful discussion of the problem illustrating the case with UK election maps. It is worth reading.

Fortunately, there are many methods in R to enhance the legibility of geographic information and the interpretability of what it is trying to be communicated. Selecting the appropriate method might depend on the research question being posed (e.g. clustering) and the data itself. Even once a method has been selected, there are different ways of operationalising them. Here we focus on **cartograms**. A cartogram is is "a thematic map of a set of features (countries, provinces, etc.), in which their geographic size is altered to be directly proportional to a selected ratio-level variable, such as travel time, population, or GNP" (Wikipedia).

Let's explore this using the example of the results of the 2016 EU referendum at Local Authority level, where remain areas clustered in London. A simple thematic map does not necessarily communicate this well because Local Authorities are both small and densely populated in London.

You can download the full set of EU referendum result data as a csv from the Electoral Commission webside. Let's read it straight into R:

```
eu_ref <- read_csv("https://www.electoralcommission.org.uk/sites/default/files/2019-07/
```

We also need a spatial object to join this to. See the appendix on how to source boundary data for more detail on how we can find relevant boundary data for our analyses. For now, we can use the data which comes with this book from the supplementary materials.

```
las <- st_read("data/England_lad_2011_gen/england_lad_2011_gen.shp")
```

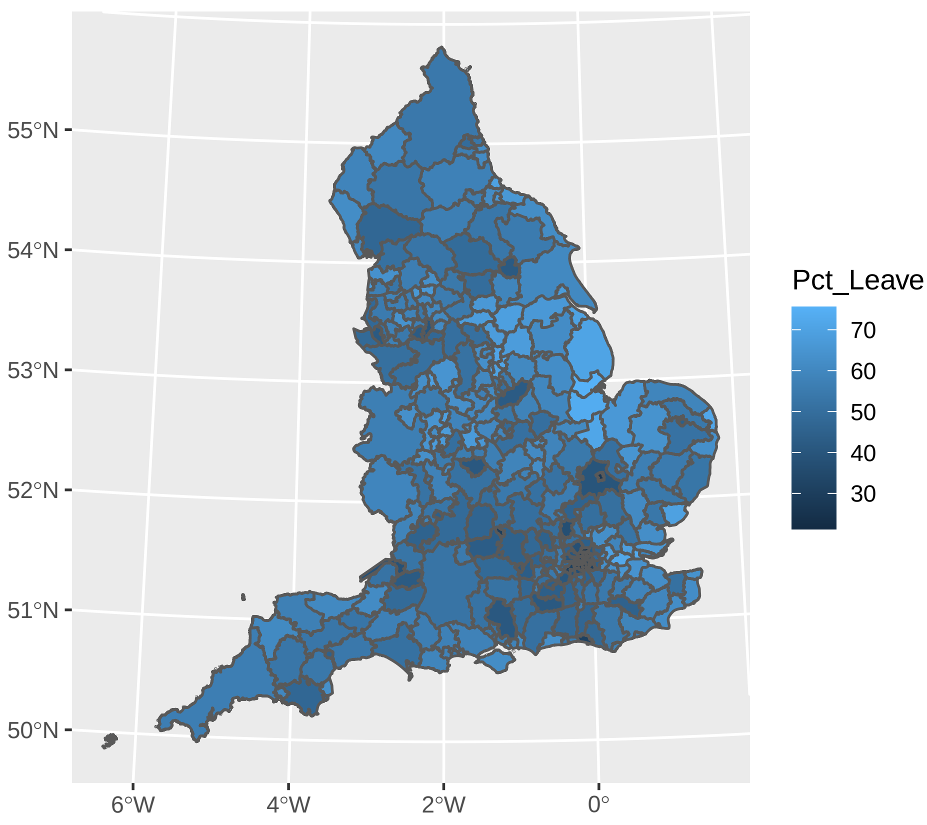
```
## Reading layer `england_lad_2011_gen' from data source `/Users/reka/Desktop/crime_mapping/crime
## Simple feature collection with 326 features and 4 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 82644.8 ymin: 5349.399 xmax: 655976.9 ymax: 657599.5
## Projected CRS: OSGB 1936 / British National Grid
```

We can now join the EU referendum data using the attribute operation `left_join()`, as we have illustrated in detail in Chapter 1.

```
eu_sf <- left_join(las, eu_ref, by = c("name" = "Area"))

#make sure we are in British National Grid Projection
eu_sf <- st_transform(eu_sf, 27700)
```

Now we can have a look at these data:

```
ggplot() +
  geom_sf(data = eu_sf, aes(fill = Pct_Leave))
```



We can see that in smaller LAs we don't even really see the result, as the boundary lines pretty much cover everything. Hmm. In this case, we cannot really see what was happening with the EU referendum in London. This is where augmenting our polygons may be handy. Cartograms offer one way to achieve this.

There are different types of cartograms. **Density-equalizing (contiguous) cartograms** are your traditional cartograms. In density-equalizing cartograms, map features bulge out a specific variable. Even though it distorts each feature, it remains connected during its creation. On the other hand, you can have **Non-Contiguous Cartograms**, where features in non-contiguous cartograms don't have to stay connected. Finally, **Dorling Cartograms** (named after professor Danny Dorling) use shapes like circles and rectangles to depict area. These types of cartograms make it easy to recognize patterns.

We can explore cartograms using the `cartogram` package. Within that we will use the `cartogram()` function. In this function, we will specify two parameters: 1 - `shp =`, which asks for the shape file (it can be a SpatialPolygonDataFrame or an sf object), and 2 - `weight =`, which asks for the variable which it should use to distort the polygon by.

In our data set we have a variable "Electorate" which refers to the total number of registered electors in that Local Authority. It serves to give an indicator of the total number of people who were eligible to vote in the referendum. We can use this variable to distort our polygons and create our cartogram.

```
eu_cartogram <- cartogram(eu_sf, "Electorate")
```

If you run this, it might take a long time. This function, while it looks nice and simple, is actually very computationally taxing for your computer. For those interested, you may like to take the time while R works this out for you to read up on the maths behind this transformation in Dougenik, Chrisman, and Niemeyer (1985).

I do have a tip for you if you want to make sure the process does not take too long. You can set another parameter in the cartogram function which is the `itermax=` parameter. This specifies the maximum number of iterations we are happy to sit through for our cartogram. If you do not specify it's set to 15. Let's set to 5 for the sake of speed:

```
# construct a cartogram using the percentage voting leave
eu_cartogram <- cartogram_cont(eu_sf, "Electorate", itermax = 5)
```

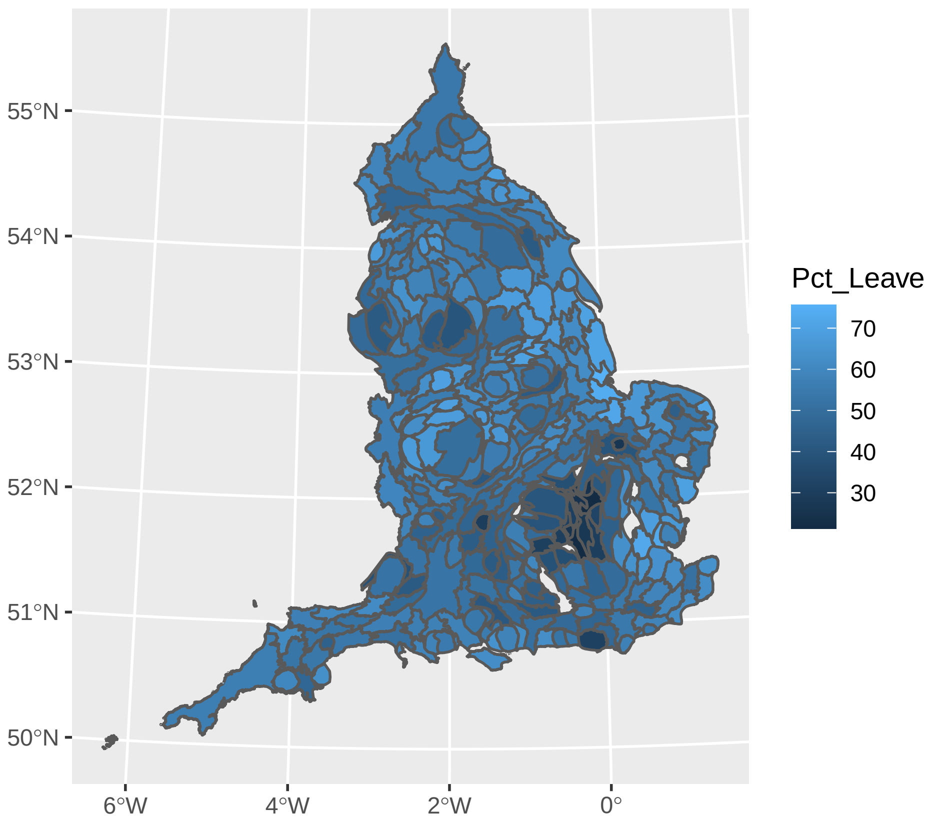
```
## Mean size error for iteration 1: 4.11884641547245
```

```
## Mean size error for iteration 2: 21.1096266805133
```

```
## Mean size error for iteration 3: 3.05848955329872

## Mean size error for iteration 4: 2.51480273537651

## Mean size error for iteration 5: 2.92812701653567
```

This will be faster (but may not result in the best possible cartogram output). Once your cartogram has been created, you can now plot again the referendum results, but using the electorate to change the size of the Local Authority:

```
ggplot() +
  geom_sf(data = eu_cartogram, aes(fill = Pct_Leave))
```



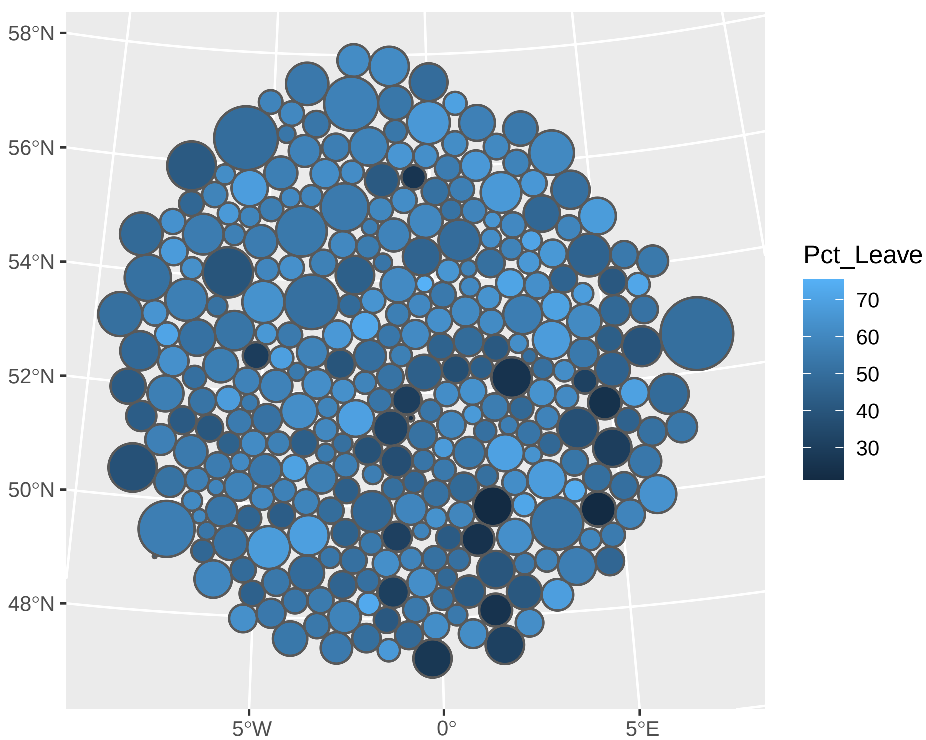
We can now see London much better, and see that darker coloured cluster where much smaller percentage of people voted leave.

4.3.1 Dorling Cartogram

I mentioned there are many types of cartogram. The one we created above tries to maintain some fidelity to our original shapes, while weighting them by some variable of interest, in our case the electorate in each Local Authority. However sometimes this may not be important. In that case, you might be interested in creating a **Dorling cartogram**.

```
# construct a Dorling cartogram using the percentage voting leave
eu_dorling_cartogram <- cartogram_dorling(eu_sf, "Electorate")
```

```
ggplot() +
  geom_sf(data = eu_dorling_cartogram, aes(fill = Pct_Leave))
```



This map has transformet each Local Authority's shape into a circle, where the radius is determined as a function of the variable we supplied, which is our Electorate in each LA. The shading here is again provided by the percentage of people who voted to leave in each area, with lighter values indicating more people voting to leave, and darker values indicating fewer people voting to leave the EU. However, the relations of these Local Authorities is tough to maintain here, and I may be hard-pressed to identify London's boroughs out of this collection. However, Dorling cartograms may have their use and place. You may read up more about them in Dorling (1996) .

4.4 Bivariate Maps

Usually in your thematic maps you are mapping one variable. For example in the maps above, we mapped the distribution of the percepntage who voted to leave the European Union (the `Pct_Leave` variable). But occasionally, you might want to use your map to illustrate the relationship between *two* variables. In this case, you may want to create a **Bivariate Choropleth Map**. Bivariate choropleths follow the same concept as the univariate ones (which display only one variable), except they show two variables at once. This is achieved through the creative use of colour.

I first came across the idea of bivariate choropleth maps from the blog of Stevens (2015). I recommend having a read as he discusses in great detail how to develop the colour schemes necessary for these maps. The idea is that there are two main colours, each one representing gradual change in one of our two variables of interest. Then, these two colours are combined to create an overlapping colour scheme.

The process itself is not too complicated. We start with binning our two variables into classes. If we bin variable 1 into n classes, and then again variable 2 into n classes, then when we compare them across one another to create our bivariate map, we will end up with n^2 classes. For example, if we bin both variables into 3 classes (low, medium, high), then when displaying them together, we will have 9 classes to visualise. Stevens (2015) blog illustrates with nice visuals, so I recommend having a look. However the practical example is in QGIS, and here we are working in R. There had been adaptations into R (eg see Grossenbacher (2019)) which we can borrow from here too.

Creating the map boils down to 5 key steps. First, we take our two variables of interest, and create bins. Second, we create a new variable, which we will use for the shading. Third we create our colour scheme. Fourth, we join this to our sf object. And, finally we map! Let's go through these steps now. Using the example of voting to leave the EU.

It may be an interesting question to look into not only how the voting outcome was distributed (`Pct_Leave`) but how this varies with voter turnout (`Pct_Turnout`). We might be interested in areas with high turnout and high or percentage on voting to leave, as these may be areas where people felt passionately. On the other hand, other areas may have had low turnout, which may have influenced the result, and in those places, were people who did turn out voting to leave or remain? These are the kinds of questions we can answer with a bivariate map.

4.4.1 Step 1: Bin our variables of interest

We are interested in two key variables, `Pct_Leave` - the percentage of people who voted to leave the EU, and `Pct_Turnout` - the percentage of the electorate who actually voted. These are both numeric continuous variables. In order to create our bivariate choropleth map, we have to bin these values into n discrete categories. Here let's go with $n = 3$.

We can use the `cut()` and the `quantile()` functions in base R to class our variable into three quantiles. The `quantile()` function identifies the sample quantiles in a continuous variable. We need to include the parameters `x=`: the numeric vector whose sample quantiles are wanted (in this case our variables `Pct_Leave` and `Pct_Turnout`), and `probs=`: numeric vector of probabilities with values in [0,1]. We can use the sequence generator function `seq()` to generate these from 0 to 1, by the increment of $\frac{1}{3}$ for 3 groups.

So first, let's create the breaks

```
leave_breaks <- quantile(eu_sf$Pct_Leave, probs = seq(0,1, by = 1/3), na.rm=TRUE, names

tunrout_breaks <- quantile(eu_sf$Pct_Turnout, probs = seq(0,1, by = 1/3), na.rm=TRUE, n
```

We can have a look at the output if we'd like:

```
leave_breaks
```

```
##        0% 33.33333% 66.66667%      100%
##  21.38000  51.72000  59.12333  75.56000
```

You can see these are the cutoff values which we want to use to "cut" our numeric variable. We do this with the `cut()` function, where we specify again what to cut, and the breaks at which to cut:

```
eu_sf <- eu_sf %>%
  mutate(leave_quantiles = cut(Pct_Leave, breaks = leave_breaks),
         turnout_quantiles = cut(Pct_Turnout, breaks = tunrout_breaks))
```

We have two resulting variables, `leave_quantiles` and `turnout_quantiles` which classify each one of our observations into one of these quartiles for both the variables. In the next step, we use these to create a new variable.

4.4.2 Step 2: New variable

This step is really quite easy. What we want to do is create a new variable, this time let's call it `group`, which tells us which quartile the specific Local Authority (each row) falls into. By applying the `as.numeric()` function we translate the ranges of the quartile into their label (i.e. 1, 2, or 3rd quartile). We do this for both variables, and paste them together using the `paste()` function, and the separator "`-`":

```
eu_sf <- eu_sf %>%
  mutate(group = paste(
      as.numeric(turnout_quantiles), "-",
      as.numeric(leave_quantiles))
    )
```

We now have a new column, called group, which tells us for each Local Authority, which quartile it falls into for each variable. For example, a value of "1 - 1" means the Local Authority belongs to the first quartile in both variables. This area would be considered to have low percent voting leave, and also low turnout. On the other hand, "3 - 1" means that there was high turnout, but a low percentage voted to leave. We use this variable to assign the appropriate colour for our colour scheme for each of the 3^2 (9) combinations.

4.4.3 Step 3: Create colour scheme

Picking a colour scheme which reflects both gradual change in each individual variable, and the combined change in both is not an easy task! Luckily Stevens (2015) has created some scale recommendations for us to choose from. I copy two of them below, but you can see the blog for another 2 options.

In this code below, I simply specify for each of the 9 values of the variable created in the previous step ("1 - 1," "1 - 2," "1 - 3," "2 - 1," … "3 - 3") an associated colour using the relevant hex code.

```
library(tibble)

bivariate_color_scale_1 <- tibble(
  "3 - 3" = "#574249", # high - high
  "2 - 3" = "#985356",
  "1 - 3" = "#c85a5a", # low - high
  "3 - 2" = "#627f8c",
  "2 - 2" = "#ad9ea5", # medium - medium
  "1 - 2" = "#e4acac",
  "3 - 1" = "#64acbe", # high - low
  "2 - 1" = "#b0d5df",
  "1 - 1" = "#e8e8e8" # low - low
) %>%
  gather("group", "fill_col")

bivariate_color_scale_2 <- tibble(
  "3 - 3" = "#3b4994", # high - high
  "2 - 3" = "#5698b9",
  "1 - 3" = "#5ac8c8", # low - high
```

```
  "3 - 2" = "#8c62aa",
  "2 - 2" = "#a5add3", # medium - medium
  "1 - 2" = "#ace4e4",
  "3 - 1" = "#be64ac", # high - low
  "2 - 1" = "#dfb0d6",
  "1 - 1" = "#e8e8e8" # low - low
) %>%
  gather("group", "fill_col")
```

4.4.4 Step 4: Join colour scheme

Now that we have a colour scheme, we can join to the spatial object, using `left_join()`. The common element is the `group` variable, so with this approach, we join the relevant colour to each value of group in that column. Let's join `bivariate_color_scale_2`

```
eu_sf <- eu_sf %>%  left_join(., bivariate_color_scale_2, by = "group")
```

4.4.5 Step 5: Create legend

The legend is a little tricky, as we need to separate out the values into separate Leave and Turnout columns. We can achieve this with the `separate()` function

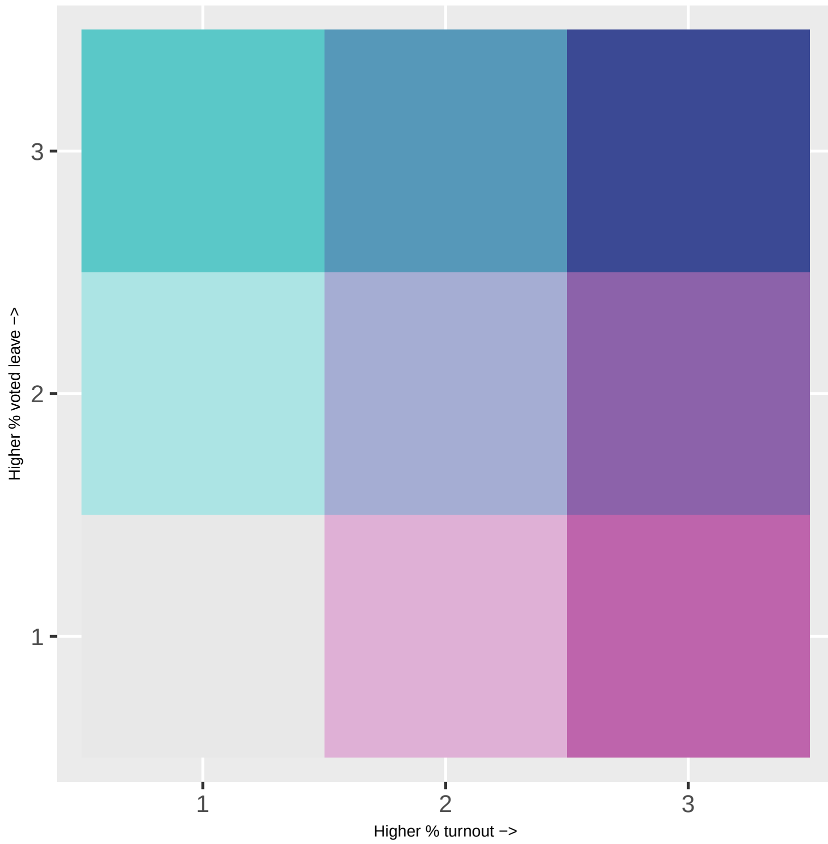
```
# separate the groups
bivariate_color_scale <- bivariate_color_scale_2 %>%
  separate(group, into = c("Pct_Turnout", "Pct_Leave"), sep = " - ")
```

Then to create the legend, we actually build a `ggplot()` object. This genius bit of code is borrowed from Grossenbacher (2019) implementation of bivariate choropleth maps.

```
legend <- ggplot() +
  geom_tile( data = bivariate_color_scale, aes(x = Pct_Turnout, y = Pct_Leave, fill = 
  scale_fill_identity() +
  labs(x = "Higher % turnout ->",
       y = "Higher % voted leave ->") +
  theme(axis.title = element_text(size = 6)) +   # makes text small for when we add le
  coord_fixed()  # forces a specified ratio between the physical representation of dat
```

This code returns the legend as a chart itself. Have a look what it looks like:

```
legend
```



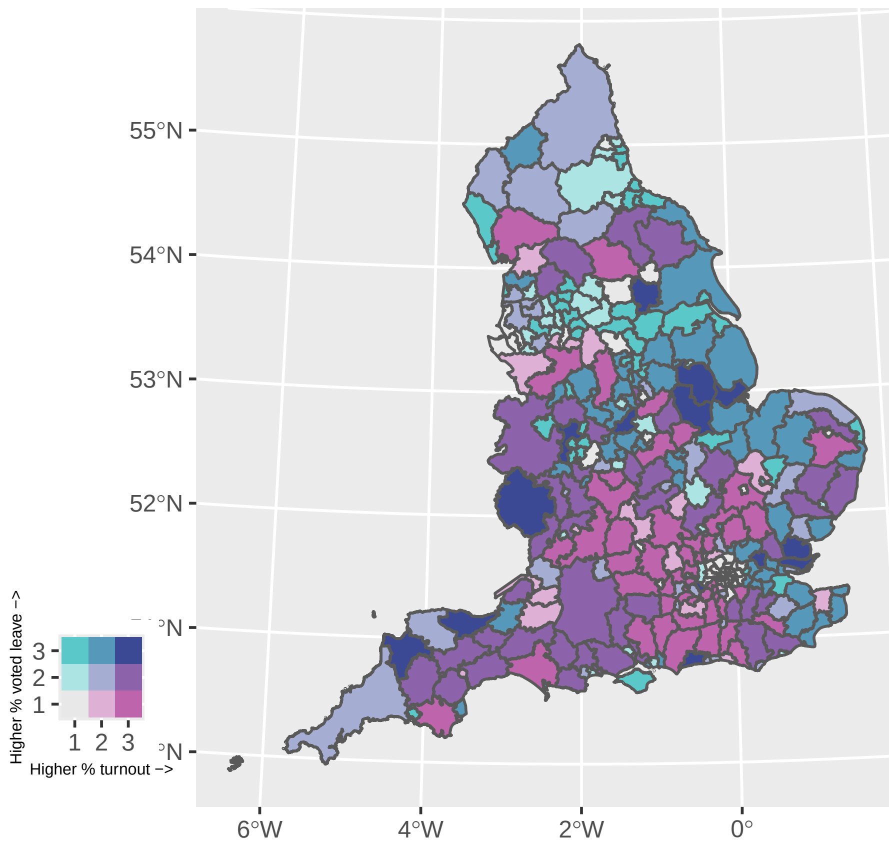
4.4.6 Step 6: Map

Now finally we put it all on the map. For the choropleth map, we use our variable `fill_col` which contains the matched colour to the group that each observation belongs to. We pass this in the familiar `geom_sf()` geometry and use the `fill=` parameter to colour the Local Authorities according to their turnout / voted leave combination. We also have to add the `scale_fill_identity()` function, as the values in the `fill_col` variable are actually the hex codes for the colour which we use to shade the Local Authorities.

```
map <- ggplot(eu_sf) +
  geom_sf(aes( fill = fill_col)) +
  scale_fill_identity()
```

Finally, to display the legend and the map together, we can use the `ggdraw()` and `draw_plot()` functions from the `cowplot` package.

```
library(cowplot)
ggdraw() +
  draw_plot(map, 0, 0, 1, 1) +
  draw_plot(legend, 0.05, 0.075, 0.2, 0.2)
```



This map may now be able to provide insight into spatial patterns in turnout and voting to leave. For example, in the South you can see lots of pink, representing areas of high turnout and low % voting to leave the EU. You can also spot the dark blue areas, these are Local Authorities which saw high voter turnout and a high proportion voting to leave.

Overall these maps can help visualise two variables on one map, and motivate discussion about relationships between variables in different places.

4.5 A note of caution: MAUP

Now that we've shown you how to do a lot of spatial crime analysis, we wanted to close with some words of caution. Remember that everything you've learned here are just tools that you will be applying to data you are working with, but it's up to you, the researcher, the analyst, the domain expert, to apply and use these with careful consideration and cautions. This discussion is very much part of spatial crime analysis, and an important field of thought.

I borrow here from George Renghert and Brian Lockwood:

> When spatial analysis of crime is conducted, the analyst should not ignore the spatial units that data are aggregated into and the impact of this choice on the interpretation of findings. Just as several independent variables are considered to determine whether they have statistical significance, a consideration of multiple spatial units of analysis should be made as well, in order to determine whether the choice of aggregation level used in a spatial analysis can result in biased findings.

- Rengert and Lockwood (2009)

In particular, they highlight four main issues inherent in most studies of space:

- issues associated with politically bounded units of aggregation,
- edge effects of bounded space
- the modifiable aerial unit problem (MAUP)
- and ways in which the results of statistical analyses can be manipulated by changes in the level of aggregation.

4.5.0.1 Scale

The scale problem involves results that change based on data that are analyzed at higher or lower levels of aggregation (Changing the number of units). For example, evaluating data at the state level vs. Census tract level.

The scale problem has moved to the forefront of geographical criminology as a result of the recent interest in small-scale geographical units of analysis. It has been suggested that smaller is better since small areas can be directly perceived by individuals and are likely to be more homogenous than larger areas. - Gerell, Manne. "Smallest is better? The spatial distribution of arson and the modifiable areal unit problem." Journal of quantitative criminology 33.2 (2017): 293-318.

4.5.0.2 Zone

The zonal problem involves keeping the same scale of research (say, at the state level) but changing the actual shape and size of those areas.

The basic issue with the MAUP is that aggregate units of analysis are often arbitrarily produced by whom ever is in charge of creating the aggregate units. A classic example of this problem is known as Gerrymandering. Gerrymandering involves shaping and re-shaping voting districts based on the political affiliations of the resident citizenry.

The inherent problem with the MAUP and with situations such as Gerrymandering is that units of analysis are not based on geographic principles, and instead are based on political and social biases. For researchers and practitioners the MAUP has very important implications for research findings because it is possible that as arbitrarily defined units of analysis change shape findings based on these units will change as well.

When spatial data are derived from counting or averaging data within areal units, the form of those areal units affects the data recorded, and any statistical measures derived from the data. Modifying the areal units therefore changes the data. Two effects are involved: a zoning effect arising from the particular choice of areas at a given scale; and an aggregation effect arising from the extent to which data are aggregated over smaller or larger areas. The modifiable areal unit problem arises in part from edge effect.

If you're interested, in particular about politics and voting, you can read this interesting piece to learn more about gerrymandering

4.5.1 Why does MAUP matter?

The practical implications of MAUP are immense for almost all decision-making processes involving GIS technology, since with the availability of aggregated maps, policy could easily focus on issues and problems which might look different if the aggregation scheme used were changed .

All studies based on geographical areas are susceptible to MAUP. The implications of the MAUP affect potentially any area level data, whether direct measures or complex model-based estimates. Here are a few examples of situations where the MAUP is expected to make a difference:

- The special case of the ecological fallacy is always present when Census area data are used to formulate and evaluate policies that address problems at individual level, such as deprivation. Also, it is recognised that a potential source of error in the analysis of Census data is 'the arrangement of continuous space into defined regions for purposes of data reporting'

- The MAUP has an impact on indices derived from areal data, such as measures of segregation, which can change significantly as a result of using different geographical levels of analysis to derive composite measures .
- The choice of boundaries for reporting mortality ratios is not without consequences: when the areas are too small, the values estimated are unstable, while when the areas are too large, the values reported may be over-smoothed, i.e. meaningful variation may be lost .
- Gerell, Manne. "Smallest is better? The spatial distribution of arson and the modifiable areal unit problem." Journal of quantitative criminology 33.2 (2017): 293-318.

4.5.2 What can we do?

Most often you will just have to remain aware of the MAUP and its possible effects. There are some techniques, that can help you address these issues, and the chapter pointed out at the beginning of this section is a great place to start to explore these. It is possible to use also an alternative, zone-free approach to mapping these crime patterns, perhaps by using kernel density estimation. Here we model the relative density of the points as a density surface - essentially a function of location (x,y) representing the relative likelihood of occurrence of an event at that point. We have covered KDE elsewhere in this course.

For the purposes of this course, it's enough that you know of, and understand the MAUP and its implications. Always be smart when choosing your appropriate spatial unit of analysis, and when you use binning of any form, make sure you consider how and if your conclusions might change compared to another possible approach.

4.6 References and further reading

4.6.1 Binning

- Johnson, Zachary Forest. (18 October 2011). "Hexbins!" Retrieved from: http://indiemaps.com/blog/2011/10/hexbins/ (8 August 2014).
- Smith, Nate. (25 May 2012). "Binning: An Alternative to Point Maps." Mapbox. Retrieved from: https://www.mapbox.com/blog/binning-alternative-point-maps/ (8 August 2014).
- Claudia A Engel's fantastic R-pub on *Making Maps in R*.
- Hexbin Graph Gallery
- US Drought Hexmap
- Hexbin with ggplot2

4.6.2 MAUP

- Gerell, Manne. "Smallest is better? The spatial distribution of arson and the modifiable areal unit problem." Journal of quantitative criminology 33.2 (2017): 293-318.
- Openshaw, Stan. "The modifiable areal unit problem. CATMOG (Concepts and techniques in modern geography) 38." Geo Abstracts, Norwich. 1984..
- Rengert, George F., and Brian Lockwood. "Geographical units of analysis and the analysis of crime." Putting crime in its place. Springer, New York, NY, 2009. 109-122.

4.6.3 Transforming polygons

- Waldo Tobler (2004) Thirty Five Years of Computer Cartograms, Annals of the Association of American Geographers, 94:1, 58-73, DOI: 10.1111/j.1467-8306.2004.09401004.x
- Langton, S.H. & Solymosi, R. (2018) 'Visualising geographic information: examining methods of improving the thematic map.' RPubs. Available: https://rpubs.com/langton_/visual_geography_study

Chapter 5

Chapter 5: Visualisation: good cartographic design

5.1 Introduction

This chapter aims to focus on introducing good practice in map design and presentation. When putting a map together you need to think about its intended audience (their level of expertise, whether you want them to interact with the map), purpose, and format of delivery (e.g., printed, web, projected in a screen, etc). There are many design decisions you need to consider: fonts, labels, colour, legends, layout, etc. In this chapter we provide a general introduction to some basic design principles for map production. These themes, and the appropriate election of symbol representation, are the subject matter of cartography, the art and science of map making. Within cartography a considerable body of research and scholarship has focused on studying the visual and psychological implications of our mapping choices. As noted on previous chapters one of the problems with maps is that powerful as a tool as they, they can lead to misunderstanding. What the mapmaker chooses to emphasise and what the map reader see may not be the same thing. We will work you through an example of a fairly basic map and the process of taking to a point where it could be ready for presentation to an audience other that yourself.

In this chapter we will be working with some data published by Hungarian police available online http://www.police.hu/hu/a-rendorsegrol/statisztikak/kozrendvedelem. Specifically we will be looking at some statistics related to drink driving. Drunk driving is one of a number of problems police confront that relate to impaired and dangerous driving. Hungary has a strict drink driving policy, with the maximum drink diving limit being 0.0 BAC. Most European countries are at 0.5 BAC, while the UK is 0.8 (except 0.5 for Scotland). We have record for each county the number of brethalyser checks carried out, and the number of these which returned a positive result. Let's read in this data from January 2020 (taken from here),

```
library(sf)
library(readr)
library(dplyr)



hungary <- st_read("data/hungary.geojson")
```

```
## Reading layer `hungary' from data source `/Users/reka/Desktop/crime_mapping/crime_ma
## Simple feature collection with 20 features and 7 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 16.11389 ymin: 45.73713 xmax: 22.89771 ymax: 48.58526
## Geodetic CRS:  WGS 84
```

```
drink_driving <- read_csv("data/drink_driving.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   name = col_character(),
##   total_breath_tests = col_double(),
##   positive_breath_tests = col_double()
## )
```

```
hu_dd <- left_join(hungary, drink_driving, by = c("name" = "name"))
```

We can now use this example to talk through the important principles of good visualisation of spatial data. We draw specifically from two areas of research: cartography and data visualisation.

Cartographers have always been concerned about the appearance of maps and how the display marries form with function Field and Demaj (2012). As there is no definitive definition for what is meant by *cartographic design* it can be challenging to evaluate what makes *good* design. However there are themes and elements which can be used to guide the map maker, and offer points of reflection to encourage thoughtful designs.

Te primary aim of maps is the communication of information in an honest and ethical way. This means each map should have a clear goal and know its audience, show all relevant data and not use the data to lie or mislead (**Dent_2009?**). It should also be reproducible, transparent, cite all data sources, and consider diversity in its audience (**Dent_2009?**). So what does that mean for specifically implementing these into practice. While a good amount of critical thought from the map maker will be required, there are aids we can rely upon. For example, (**Field_2007?**) developed a map evaluation checklist which you can access here: http://downloads.esri.com/MappingCenter2007/arcGISResources/more/MapEvaluationGuidelines.pdf. The questions fall into three broad categories: *Cartographic Requirements* such as what is the rationale for the map, who are the audience?. *Cartographic Complication and Design* such as are all the relevant features included and do the colours, symbold, and other features legible and appropriate to achieve the map's objectives? And finally, *Map Elements and Page Layout* which tackle some specific features such as orientation indicator, scale indicator, legend, titles and subtitles, and production notes. We will discuss these elements in this chapter to some degree, and the recommended reading will guide the reader to further advice on these topics.

Data visualisation is a somewhat newer field, however, it seems to encompass the same guiding principles when consdiering what makes good design. According to Kirk (2016) three principles offer a guide when deciding what makes a *good* data visualisation. It must be:

- Trustworthy
- Accessible, and
- Elegant

The first principle, of trust speaks to the integrity, accuracy, and legitimacy of any data visualisation we produce. Kirk (2016) suggests this principle to be held above all else, as our primary goal is to communicate truth (as far as we know it) and avoid at all cost to present what we know to be misleading content. Accessibility refers to our visualisation being useful, understandable, and unobtrusive, as well as accessible for all users. There are many things to consider in your audience such as *dynamic of need* (do they have to engage with your visualisation, or is it voluntary?), *subject-matter knowledge* are they experts in the area, are they lay people to whom you must communicate a complex message?, and many other factors (see Kirk (2016)). Finally, elegance refers to aesthetics, attention to detail, and an element of *doing as little design as possible* - meaning a certain invisibility whereby the viewer of your visualisation focussed on the content, rather than the design - that is the main point is the message that you are trying to communicate with your data!

In this chapter, we will aim to bring together the above principles, and work through a practical example of how to apply this to achieve good visualisations. Specifically we will cover:

- data representation
- colour

- text
  - titles and subtitles
  - legend
  - annotation
  - production notes
- composition
  - orientation indicator
  - scale indicator
  - borders
- inset maps
- interactivity (??)

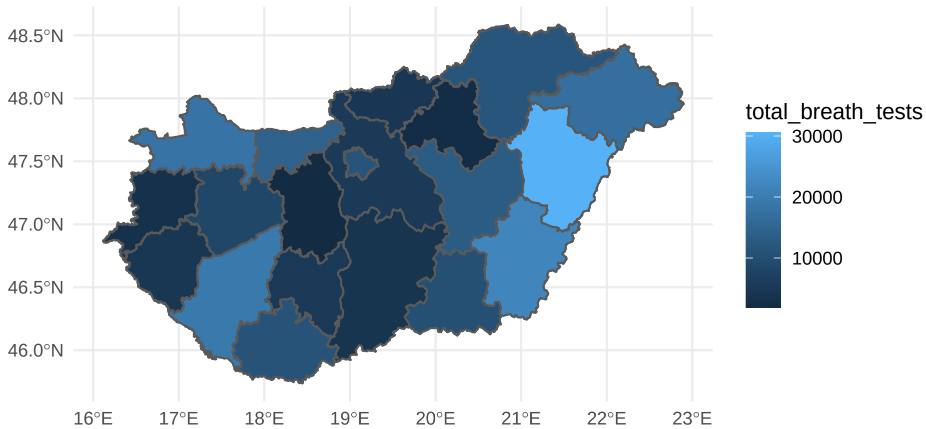
5.2 Data representation

5.2.1 Thematic maps

We've been working with thematic maps thus far. There are many decisions that go into making a thematic map, which we have explored at length in the previous chapters, such as how (and whether) to bin your data (Chapter 3) and how (or whether) to transform your polygons (Chapter 4). These are important considerations on how to represent your data to your audience, and require a technical understanding, not only an aesthetic one. So please do read over those chapters carefully when thinking about how to represent your data.

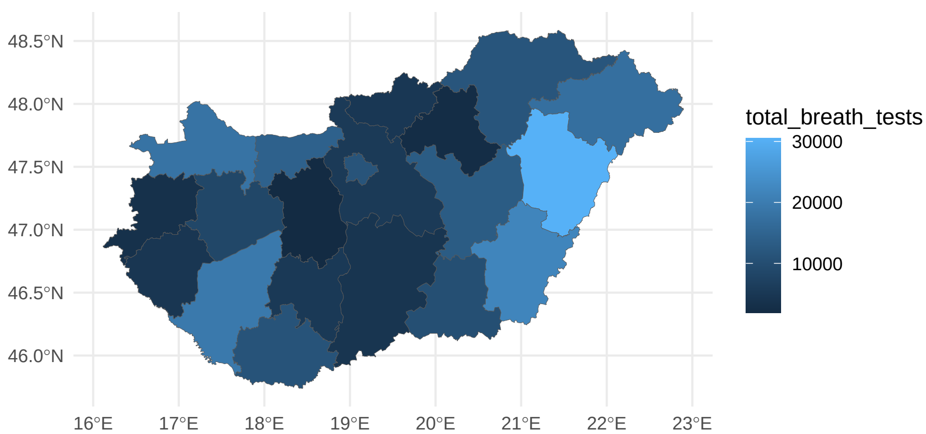
We can use the `ggplot2` package to plot our thematic map, using the `geom_sf()` function. To shade each polygon with the values of a specific variable, we use the `fill =` argument within the `aes()` (aesthetics) function. Most simply:

```
library(ggplot2)

ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_breath_tests)) +
  theme_minimal()
```



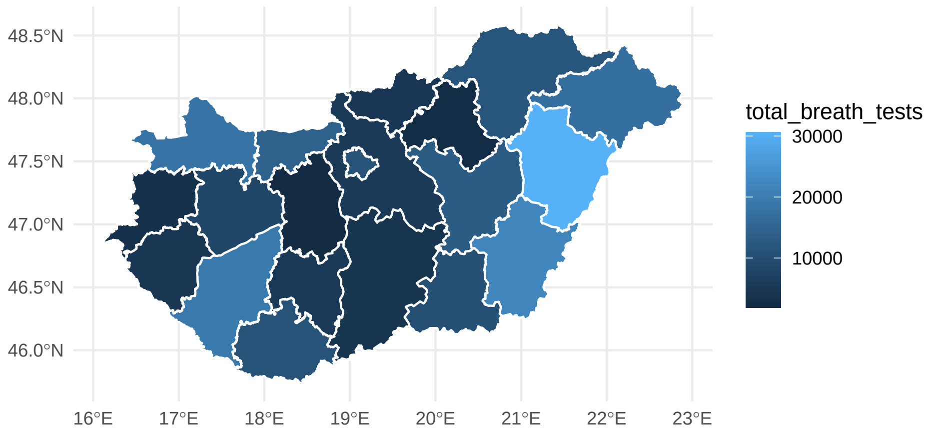
We can change the colour and size of the borders of our polygons with arguments inside the `geom_sf()` function, but outside the `aes()` function, as long as we're not using our data to define these. For example we can change the line width (`lwd =`) to 0, eliminating bordering lines between our polygons:

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_breath_tests), lwd = 0) +
  theme_minimal()
```



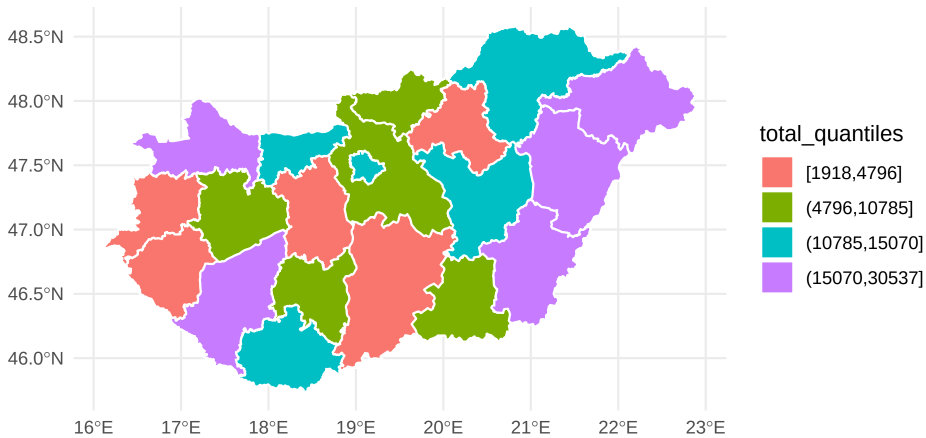
Or we can change the colour of the borders with the `col =` argument:

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_breath_tests), lwd = 0.5, col = "white") +
  theme_minimal()
```



Here we have a continuous fill for our values, however we can employ our learning from Chapter 3 and apply a classification system, such as quantiles. To do this we might create a new variable which contains the quantiles of our numeric variable, and then use that as our `fill =`.

```
hu_dd <- hu_dd %>%
  mutate(total_quantiles = cut(total_breath_tests, breaks = round(quantile(total_breatl


ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  theme_minimal()
```



The colour scheme is terrible, but we will talk about colour in the next section, so we can forgive that for now…

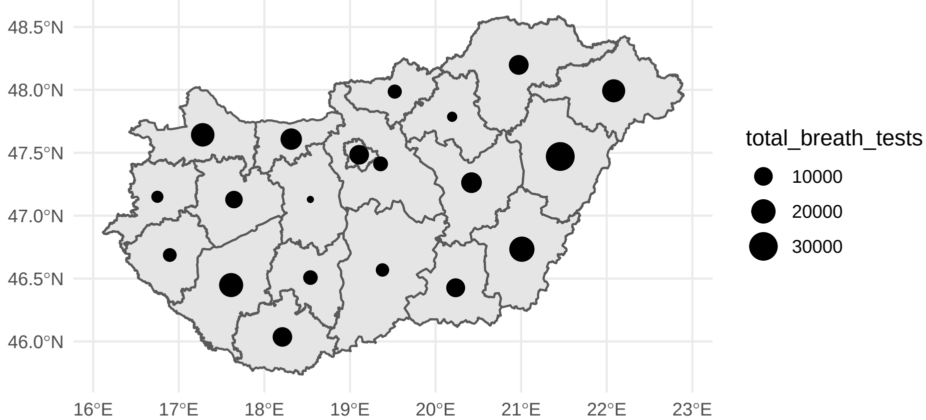
5.2.2 Symbols

You might not want to display your map as thematic map, you may want to use symbols. Again we explored this in Chapter 3, here is one way you can use graduated symbol map with `ggplot()`. You can take the centroid of each county polygon using the `st_centroid()` function from the `sf` package, and then when mapping with `geom_sf()`, within the `aes()` function specity the `size =` argument to the variable you wish to visualise:

```
ggplot(data = hu_dd) +
  geom_sf() +
  geom_sf(data = st_centroid(hu_dd), aes(size = total_breath_tests)) +
  theme_minimal()
```

```
## Warning in st_centroid.sf(hu_dd): st_centroid assumes attributes are constant
## over geometries of x
```

```
## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for longitude/
## latitude data
```

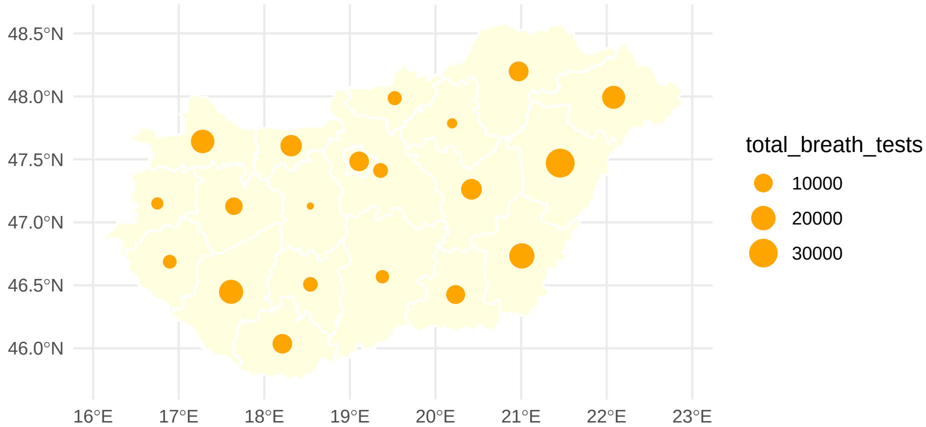


Like with the thematic map you can play around with colour and shape:

```
ggplot(data = hu_dd) +
  geom_sf(fill = "light yellow", col = "white") +
  geom_sf(data = st_centroid(hu_dd), aes(size = total_breath_tests), col = "orange") +
  theme_minimal()
```

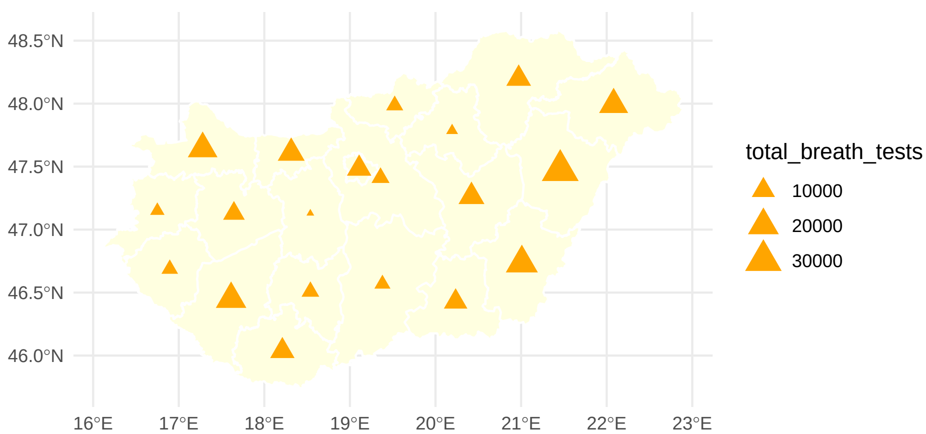
```
## Warning in st_centroid.sf(hu_dd): st_centroid assumes attributes are constant
## over geometries of x
```

```
## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for longitude/
## latitude data
```




```
ggplot(data = hu_dd) +
  geom_sf(fill = "light yellow", col = "white") +
  geom_sf(data = st_centroid(hu_dd), aes(size = total_breath_tests), col = "orange", sh
  theme_minimal()
```

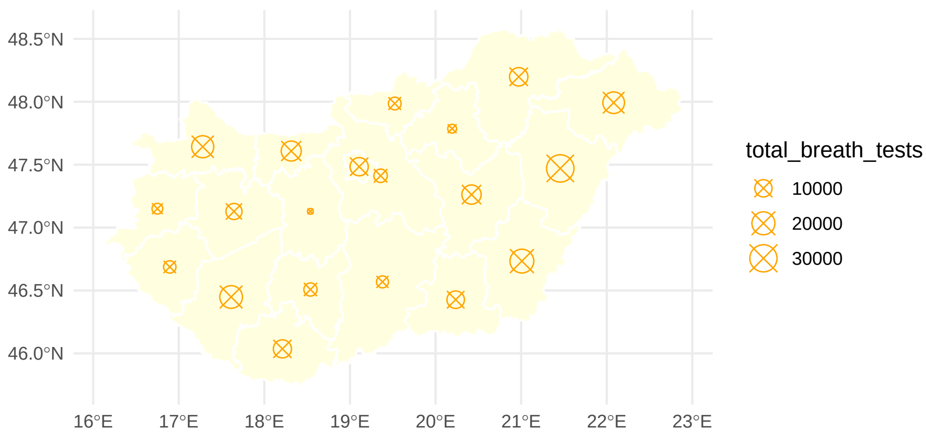
```
## Warning in st_centroid.sf(hu_dd): st_centroid assumes attributes are constant
## over geometries of x

## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for longitude/
## latitude data
```



```
ggplot(data = hu_dd) +
  geom_sf(fill = "light yellow", col = "white") +
  geom_sf(data = st_centroid(hu_dd), aes(size = total_breath_tests), col = "orange", shape = 13)
  theme_minimal()
```

```
## Warning in st_centroid.sf(hu_dd): st_centroid assumes attributes are constant
## over geometries of x
```

```
## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for longitude/
## latitude data
```



How you choose to represent your data will depend on your decisions to the questions asked above about audience, message, integrity, and so on.

5.2.3 Rate v count

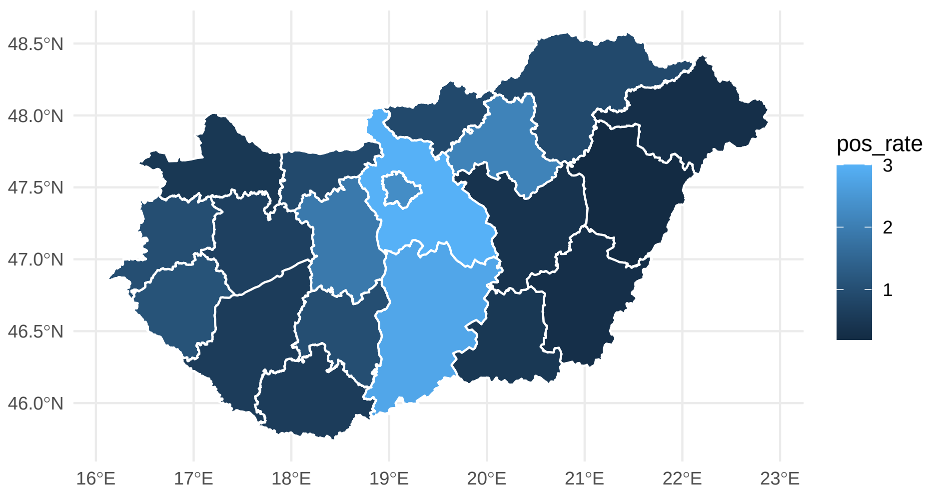
In chapter 3 we have discussed this already in great detail, so again I will not dwell on this, but it is important that your data are meaningful and easy to interpret. We might, in this case for example, want to consider the rate of positive breath tests per test carried out in each county. To compute this, we might want to consider the proportion of positive results on the breatalyser tests (where the person had been drinking and their result is over the limit). To compute this, we can simply divide the positive results by the total test, and multiply by 100. We also include the `round()` function in there

```
hu_dd <- hu_dd %>%
  mutate(pos_rate = round(positive_breath_tests/total_breath_tests*100,1))
```

We can see the county with the highers proportion of test yielding drink drivers is Pest megye with 3 %, while the county with the lowest is Hajdú-Bihar with 0.2 %.

We can visualise this rate on our thematic map in exactly the same way as the count data, but using our new variable in the `fill =` argument:

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = pos_rate), lwd = 0.5, col = "white") +
  theme_minimal()
```



But with a graduated symbol map we can get a little more creative. Sure, one approach may be to once again change the size of the symbol, but we could get a little more creative and use charts. For example there is the library `scatterpie` which has the function `geom_scatterpie()` which allows us to present our ratios in pie charts.

To use this function, we need our data to be in a dataframe which has separate columns for latitude and longitude (rather than a geometry object), where each observation has a unique id, and where we have a separate column for each proportion we wish to visualise.

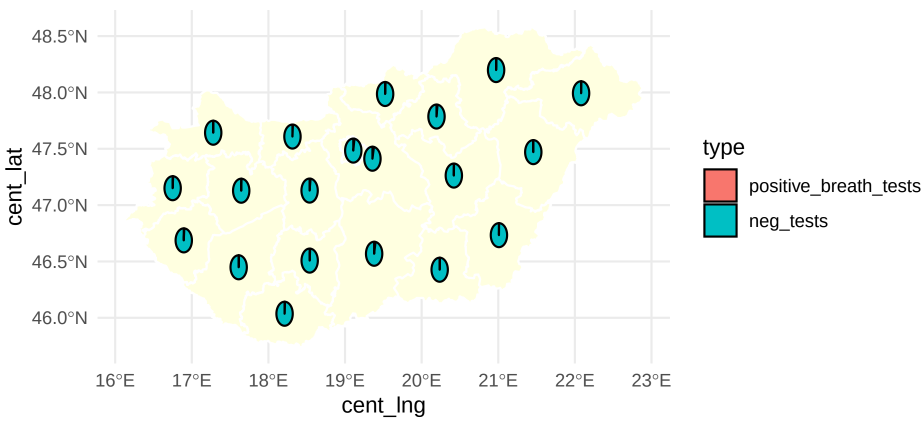
To create this, we first need to get the centroid of each polygon to get our longitude and latitude. We also need to calculate the number of negative breath tests. We then need to extract the coordinates into longitude and latitude columns, and finally we can drop the geometry and keep only the columns we need for this new dataframe.

```
scatter_map_df <- hu_dd %>%
  mutate(cent_lng = st_coordinates(st_centroid(.))[,1],     # extract centroid longitud
         cent_lat = st_coordinates(st_centroid(.))[,2],     # extract centroid latitude
```

```
        neg_tests = total_breath_tests - positive_breath_tests) %>%      # calculate negative tes
  st_drop_geometry() %>%        # remove the geometry column
  dplyr::select(cent_lng, cent_lat, osm_id, positive_breath_tests, neg_tests)   # keep only neces
```

We now have this new dataframe which has coordinates for the centroid of each polygon, and we can use the `geom_scatterpie()` function from the `scatterpie` package to plot a pie chart of the proportion of positive to negative breath tests in each county in Hungary:

```
ggplot(data = hu_dd) +
  geom_sf(data = hu_dd, fill = "light yellow", col = "white") +
  scatterpie::geom_scatterpie(data=scatter_map_df, aes(x=cent_lng, y=cent_lat, group=osm_id), col
  theme_minimal()
```



In this case, it's not super informative to visualise this, as discussed above our positive rates range between 0.2 %, and 3 %. However with more evenly distributed proportions this may be something to try.

5.3 Colour

When choosing a colour palette, the first thing to consider is what kind of colour scheme we need. This will depend on the variable we are trying to visualise. We go back, once again, to the first week of the course, where we discussed *Levels of Measurement*. Remember those? Still important!

Depending on the kind of varaible we want to visualise, we might want a Qualitative colour scheme (for categorical nominal variables), a Sequential colour scheme (for categorial ordinal, or for numeric variables) or a Diverging colour scheme (for categorial ordinal, or for numeric variables).

For qualitative colour schemes, we want each category (each value for the variable) to have a perceptible difference in colour. For sequential and diverging color schemes, we will want mappings from data to color that are not just numerically but also perceptually uniform.

- **sequential scales** (also called gradients) go from low to high saturation of a colour.
- **diverging scales** represent a scale with a neutral mid-point (as when we are showing temperatures, for instance, or variance in either direction from a zero point or a mean value), where the steps away from the midpoint are perceptually even in both directions.
- **qualitative scales** identify as different the different values of your categorical nominal variable from each other.

For your sequetial and diverging scales, the goal in each case is to generate a perceptually uniform scheme, where hops from one level to the next are seen as having the same magnitude.

Of course, perceptual uniformity matters for your qualitative scales for your unordered categorical variables as well. We often use color to represent data for different countries, or political parties, or types of people, and so on. In those cases we want the colors in our qualitative palette to be easily distinguishable, but also have the same valence for the viewer. Unless we are doing it deliberately, we do not want one color to perceptually dominate the others.

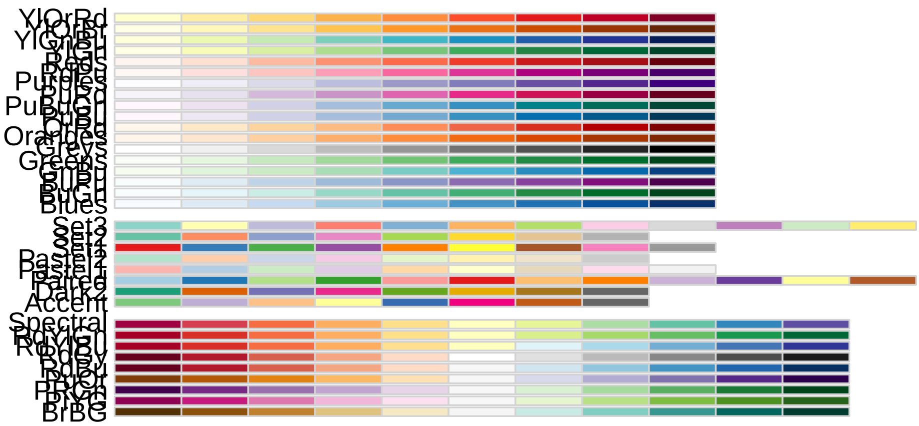
The main message here is that you should generally not put together your color palettes in an ad hoc way. It is too easy to go astray. In addition to the considerations we have been discussing, there we might also want to avoid producing plots that confuse people who are colour blind, for example, and color blindness comes in a variety of forms. Fortunately for us, almost all of the work has been done for us already. Different color spaces have been defined and standardized in ways that account for these uneven or nonlinear aspects of human color perception.

A good resource is colorbrewer. We have come across (**Brewer_2016?**) in Chapter 3. Colorbrewer is a resource developed by Cynthia Brewer and colleagues in order to help implement good colour practice in data visualisation and cartography. This site offers many colour schemes we can make use of for our maps, which are easily integrated into R using the `Rcolorbrewer` package.

```
library(RColorBrewer)
```

Once you have the package loaded, we can look at all the associated palettes with the function `display.brewer.all()`.

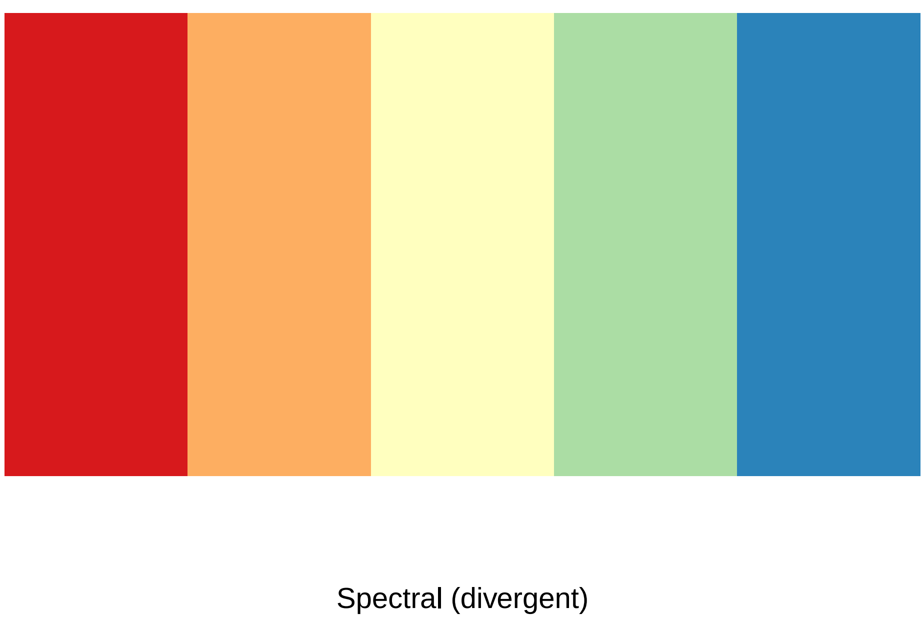
```
display.brewer.all()
```



The above gives a wide choice of pallettes, and while they are applicable to all sorts of data visualisations, they were created especially for the case of thematic maps.

We might use the above code to pick a palette we like. We might then want to examine the colours more closely. To do this we can use the `display.brewer.pal()` function, and specify `n=` - the number of colours we need, as well as the palette name with `name =`:

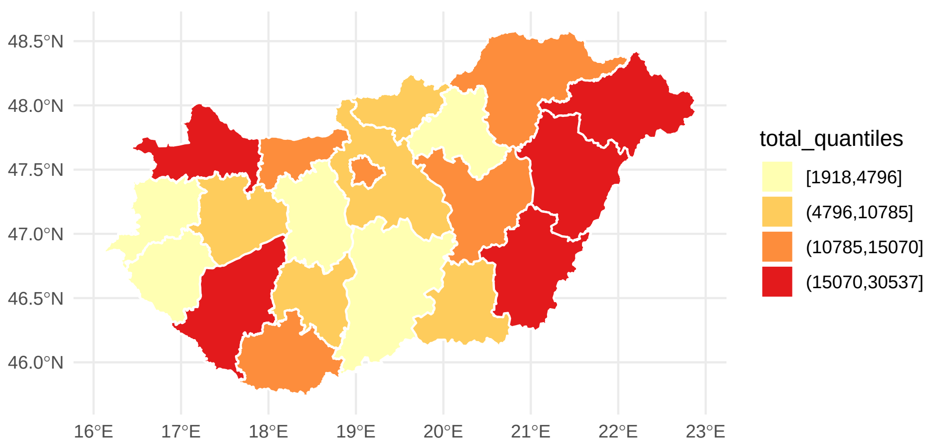
```
display.brewer.pal(n = 5, "Spectral")
```



Let's go back to our thematic map of the quantiles of total breath tests per county. We might be interested in this map to show distribution of policing activity for example. We made this map earlier with the default colour scheme, which didn't really communicate to use the graduated nature of our data we were visualising. To properly do this, we may imagine using a *seqantial* scale. We can use one of the sequential scales available within `RColorBrewer` with adding the `scale_fill_brewer()` function to our ggplot. In this function we can specify the `type=` parameter, i.e. if we want to use sequential, diverginging, or qualitative colour schemes (specified as either "seq" (sequential), "div" (diverging) or "qual" (qualitative)). We can then specify our preferred palette with the `palette =` argument.

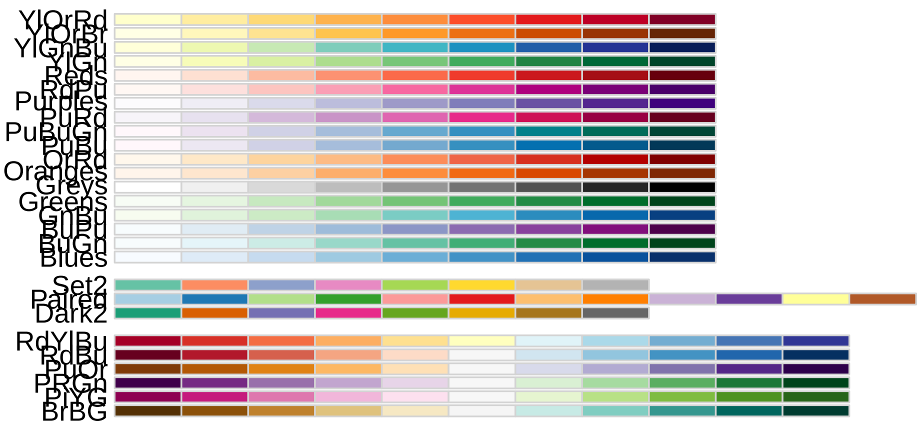
Let's demonstrate here with the "YlOrRd" sequential palette:

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "YlOrRd") +
  theme_minimal()
```



This looks much better, and communicates our message much more clearly. Is this accessible to our colourblind colleagues? Earlier, when we asked to view all the palettes with the `display.brewer.all()` function, we did not specify any arguments. However, we can do so in order to filter only those palettes which are accessible for all audiences. We can include the parameter `colorblindFriendly =` to do so:

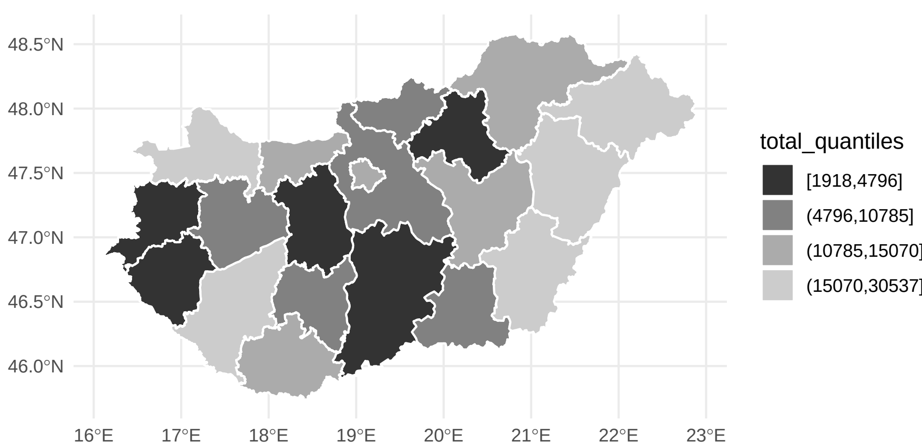
```
display.brewer.all(colorblindFriendly = TRUE)
```



You can see there are a few palettes missing from our earlier results, when we did not specify this requirement. My recommendation is to always use one of these palettes.

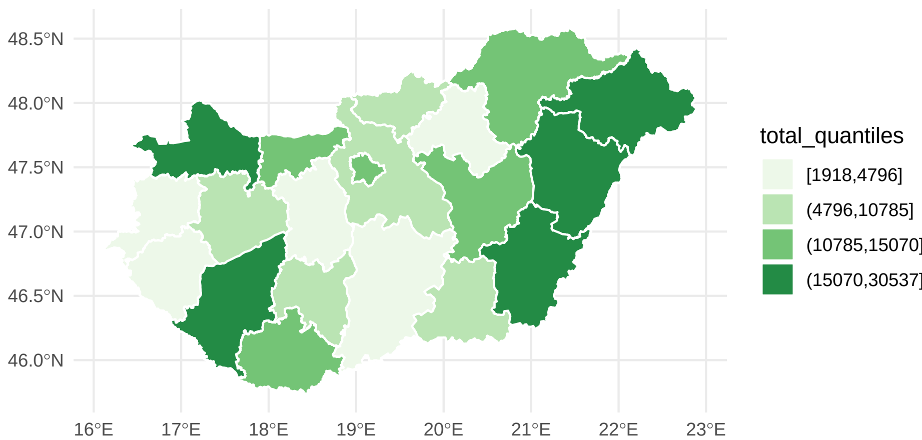
Another way to ensure that we are making accessible maps is to use greyscale (if your map is being printed, this may also save some money). To introduce a greyscale palette, you can use the function `scale_fill_grey()` from the `ggplot2` package:

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_grey() +
  theme_minimal()
```



Sometimes you might prefer such a map. However, do keep in mind, a number of studies have shown the desirability of monochrome colour (over greyscale) thematic maps, as they are linked to less observer variability in interpretation (Lawson (2021b)). So you might want to use something like this instead:

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens") +
  theme_minimal()
```



Overall, the key thing is to be conscious with the colours you choose to represent your data. Make sure that they are accessible for all audiences, and best represent the patterns in your data which you want to communicate to your audiences.

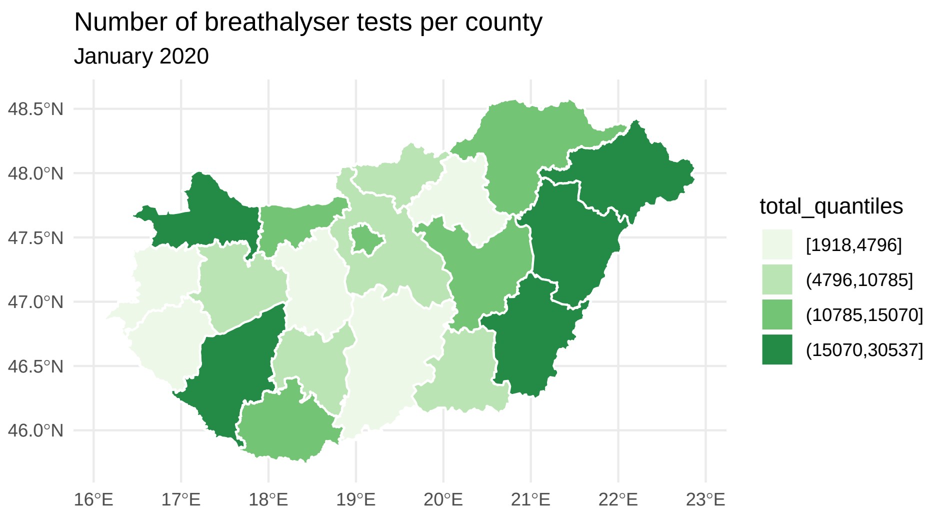
5.4 Text

There are important pieces of information with every map which are represented by text.

5.4.1 Titles and subtitles

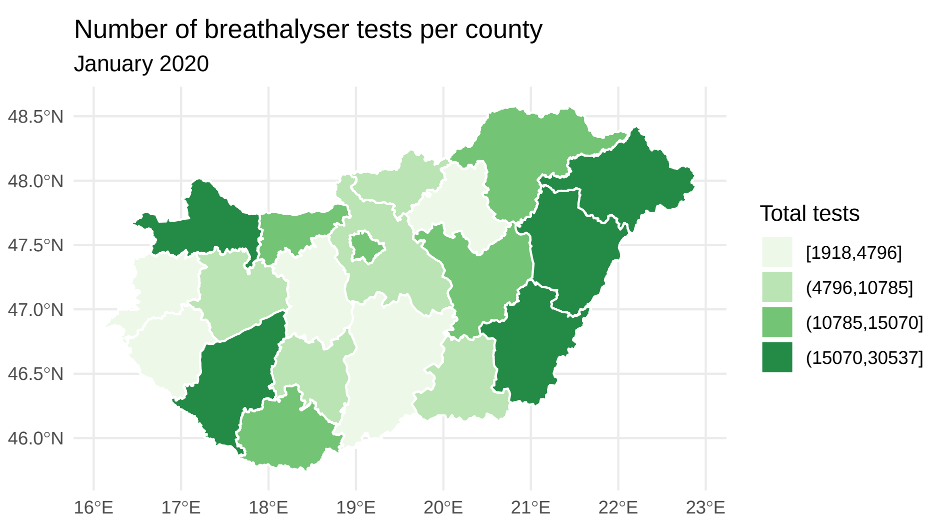
To give your map a title and subtitle, you can use the appropriate functions from the `ggplot2()` package.

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens") +
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```



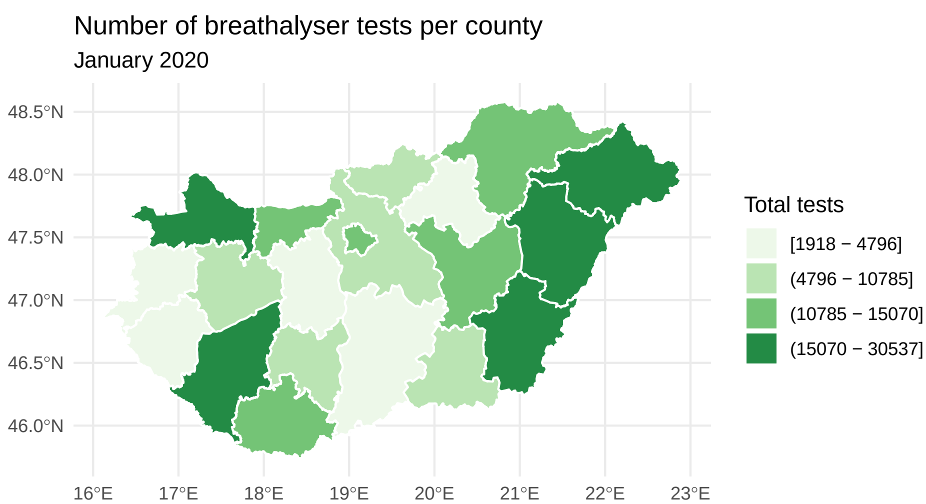
5.4.2 Legend

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```



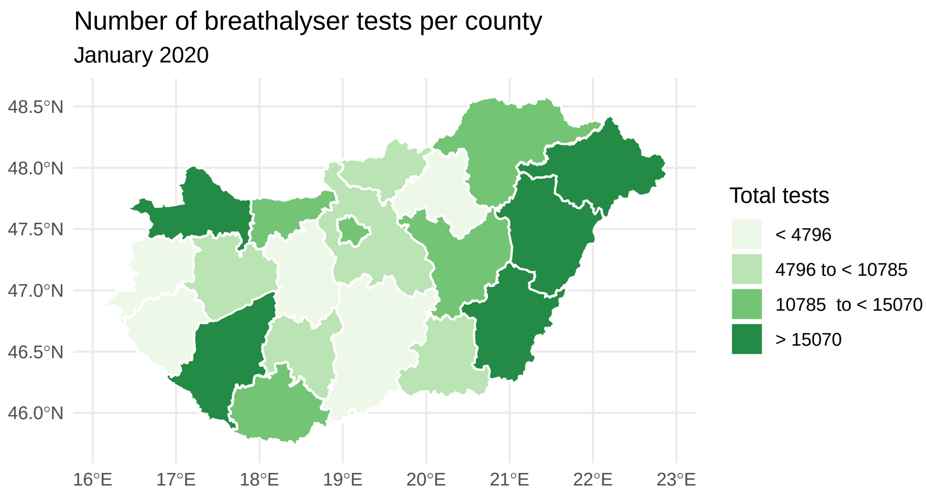
We can also change the levels. For example, we can replace the "," with a " - " if we'd like using the `gsub()` function, and assign the new levels object in the `labels =` parameter of the `scale_fill_brewer()` function:

```
new_levels <- gsub(",", " - ",levels(hu_dd$total_quantiles))

ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests", labels = ne
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```



Or we could completely rename them if we liked:

```
new_levels <- c("< 4796", "4796 to < 10785", "10785  to < 15070", "> 15070")

ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests", labels = ne
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```



You can change the labels however you would like, but do keep in mind any loss of information you may introduce. For example with this second version, we no longer know what are the minimum and maximum balues on our map, as we've removed that information with our new levels. Again, no wrong answers here, but whatever best fits the data and the purpose of the map.

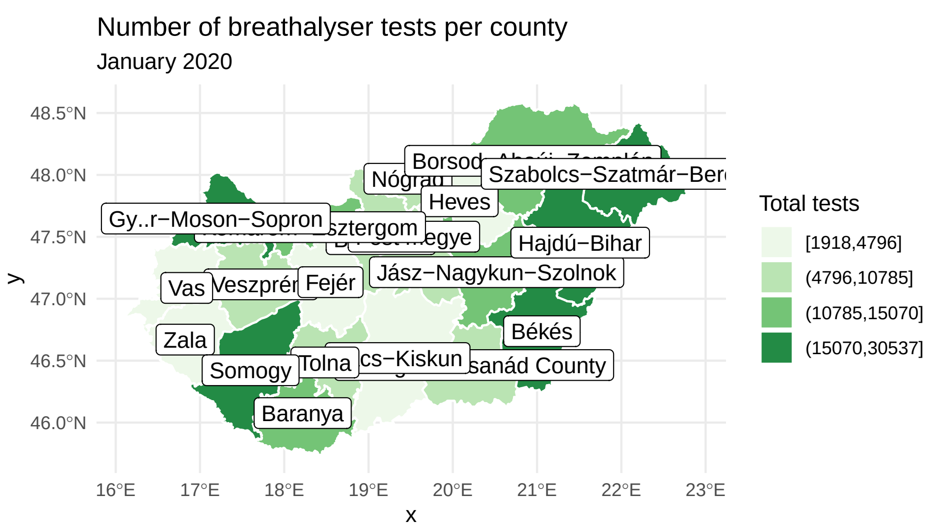
5.4.3 Annotation

Sometimes you might want to include some textual information on your map as well. From `ggplot2` versions v.3.1.0 the functions `geom_sf_text()` and `geom_sf_label()` make it very smooth for us to do this.

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_sf_label(aes(label = name)) +
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```



You may notice there is some overlapping here which renders some names unreadable. Well while there is work in this space to develop the function geom_sf_label_repel() at the time of writing this is not yet available. However this application of the `geom_label_repel()` function from the `ggrepel` package advised by (**Yutani_2018?**) achieves the same outcome:

```
library(ggrepel)

ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_label_repel(data = hu_dd, aes(label = name, geometry = geometry), stat = "sf_coo
```

```
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```

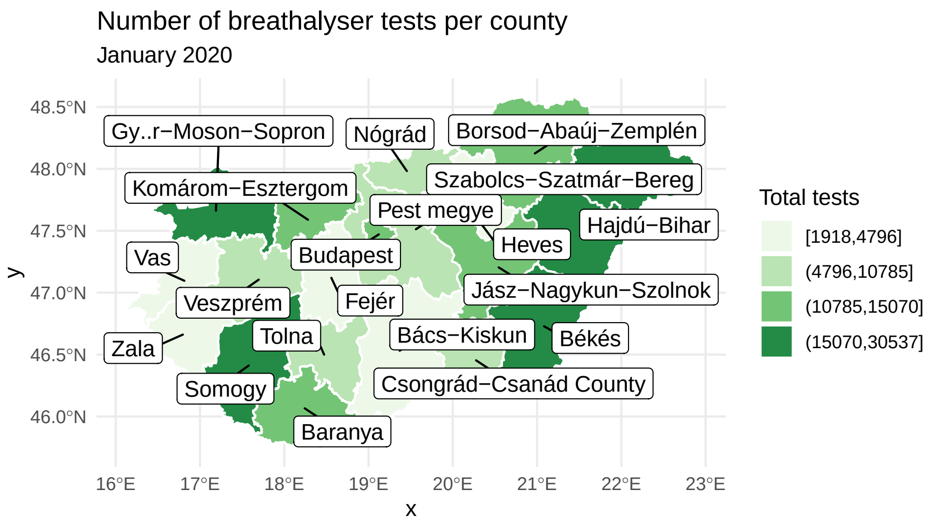
```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```



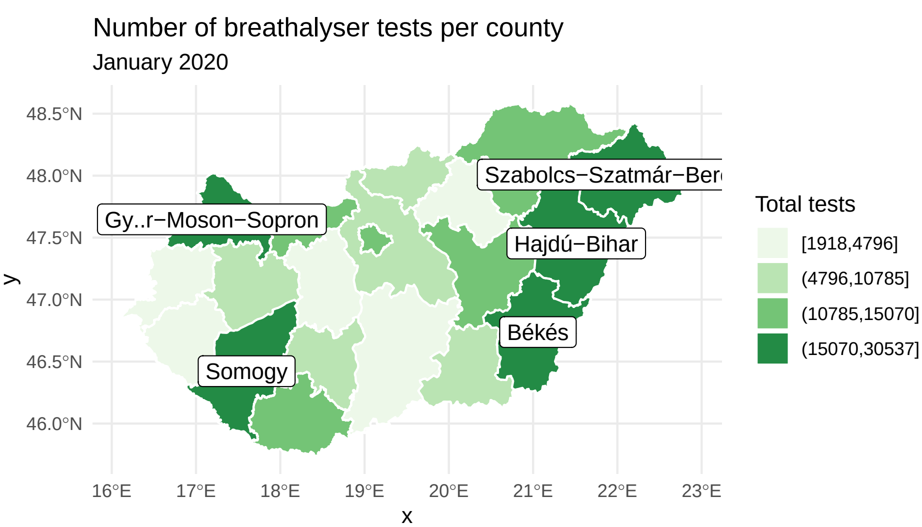
This map looks really busy, you might want to include only a few labels, which are of interest. For example, you might want to label only those which are in the top quartile:

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_sf_label(data = hu_dd %>% filter(total_breath_tests >= 15070), aes(label = name)
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```

```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <c5>
```

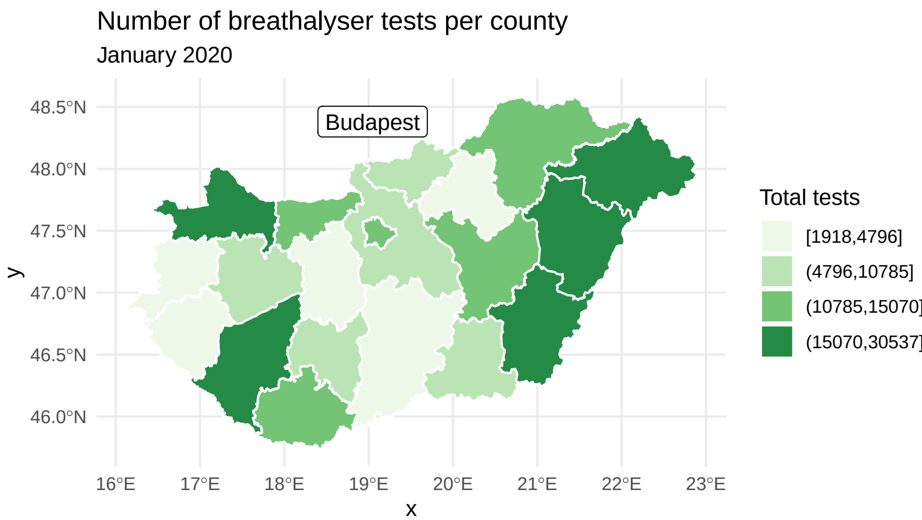
```
## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## conversion failure on 'Győr-Moson-Sopron' in 'mbcsToSbcs': dot substituted for
## <91>
```



Or you want to specifically highlight a county of interest, let's say Budapest. In this case, we might actually want to keep our annotation off the map. We can do this by using the `nudge_x` and `nudge_y` parameters of the `geom_sf_label()` function.

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_sf_label(data = hu_dd %>% filter(name == "Budapest"), aes(label = name), nudge_y = 0.9, nu
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



But this floating label is a little ambiguous. We might want to use an arrow to point out Budapest on the map. To do this, we can use `geom_curve()` within `ggplot2`. We will need two sets of x and y values for this segment, the start point (`x` and `y`) and the end point (`xend` and `yend`). The end point will be the coordinates where we want the arrow pointing to. This would be some x,y pair within Budapest. We can use the `st_coordinates()` function once again the extract the centroid, this time of the Budapest polygon. Let's extract the longitude of the centroid into an object called `bp_x` for our x value, and the latitude of the centroid into an object called `bp_y` for our y value.

```
bp_x <- hu_dd %>% filter(name == "Budapest") %>%
  mutate(cent_lng = st_coordinates(st_centroid(.))[,1]) %>%
  pull(cent_lng)
```

```
## Warning in st_centroid.sf(.): st_centroid assumes attributes are constant over
## geometries of x
```

```
## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for longitude/
## latitude data
```

```
bp_y <- hu_dd %>% filter(name == "Budapest") %>%
  mutate(cent_lat = st_coordinates(st_centroid(.))[,2]) %>%
  pull(cent_lat)
```

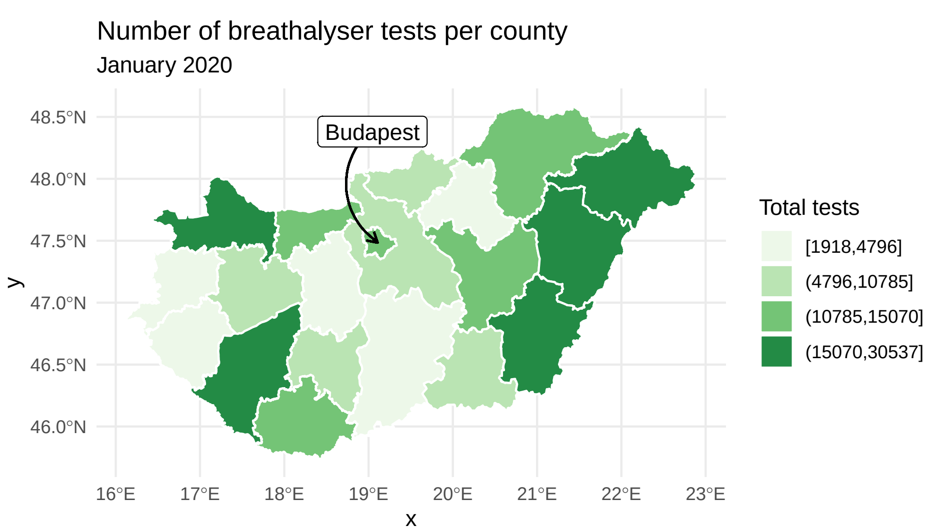
```
## Warning in st_centroid.sf(.): st_centroid assumes attributes are constant over
## geometries of x
```

```
## Warning in st_centroid.sf(.): st_centroid does not give correct centroids for
## longitude/latitude data
```

Great, so we have the end point for our segment, but where should it start. Well we want it pointing from our label, so we can think back to how we adjusted this label with the `nudge_x` and `nudge_y` parameters inside the `geom_sf_label()` function earlier. We can add (or subtract) these values to our bp_x and bp_y objects to determine the start points for our curve. Finally, we can also specify some characteristics of the arrow head on our curve with the `arrow =` parameter. Here we specify we want 2 milimeter size.

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_curve(x = bp_x - 0.1,
             y = bp_y + 0.9,
             xend = bp_x ,
             yend = bp_y,
             arrow = arrow(length = unit(2, "mm"))) +
  geom_sf_label(data = hu_dd %>% filter(name == "Budapest"),
                aes(label = name),
                nudge_y = 0.9,
                nudge_x = -0.1) +
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020")
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



5.4.4 Production notes

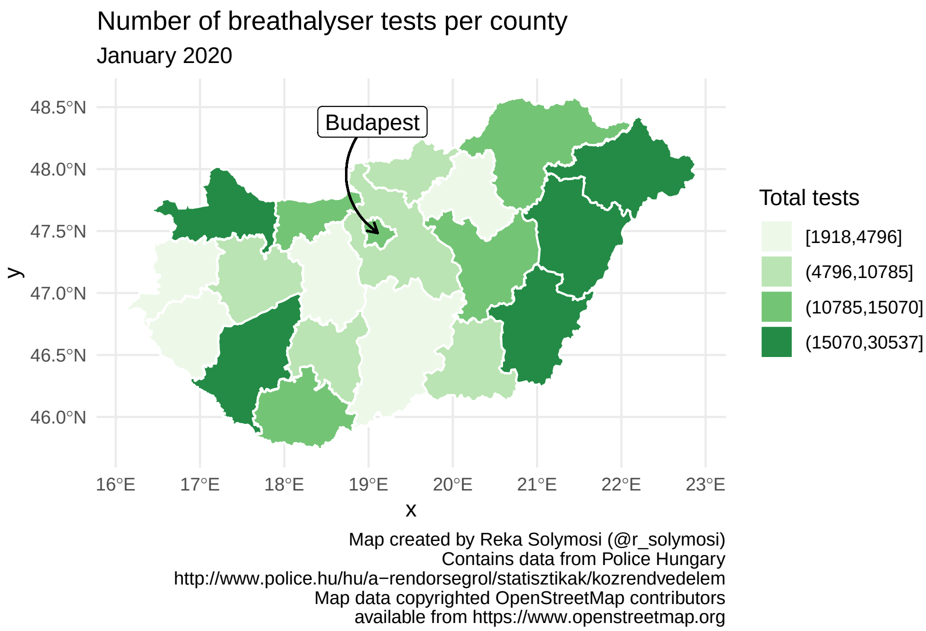
You should always include production notes with your map. This includes some information about you who made it, as well as any attributions for data. Here we can string together a series of information we want to include, appended with a newline character, in order to keep our notes nice and legible:

```
caption_text <- paste("Map created by Reka Solymosi (@r_solymosi)",
                      "Contains data from Police Hungary",
                      "http://www.police.hu/hu/a-rendorsegrol/statisztikak/kozrendvede
                      "Map data copyrighted OpenStreetMap contributors",
                      "available from https://www.openstreetmap.org",
                      sep = "\n")
```

Then we can include this `caption_text` object as a caption in the function `labs()`.

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_curve(x = bp_x - 0.1,
             y = bp_y + 0.9,
             xend = bp_x ,
             yend = bp_y,
             arrow = arrow(length = unit(2, "mm"))) +
  geom_sf_label(data = hu_dd %>% filter(name == "Budapest"),
                aes(label = name),
                nudge_y = 0.9,
                nudge_x = -0.1) +
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020") +
  labs(caption = caption_text)
```

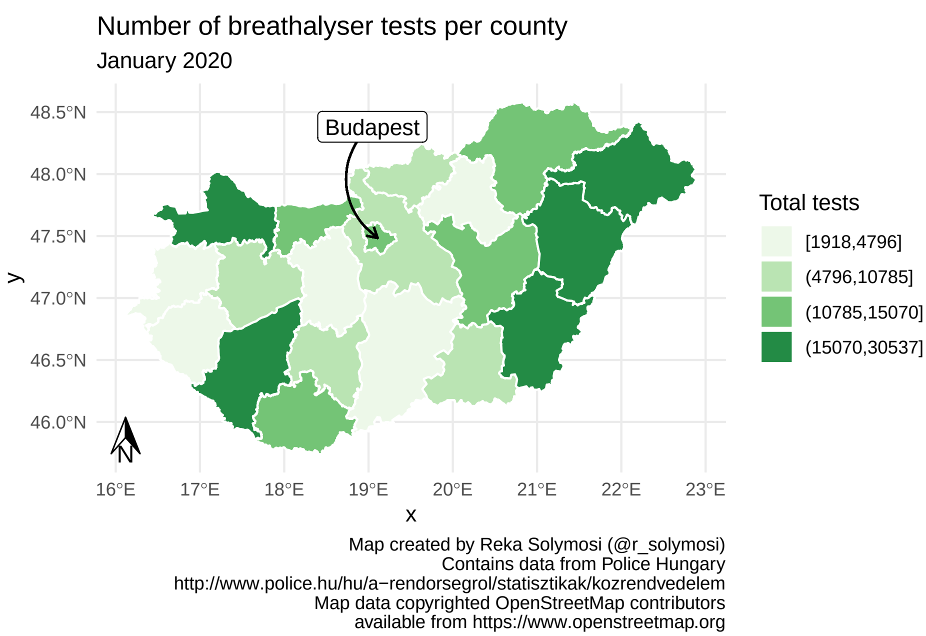
```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



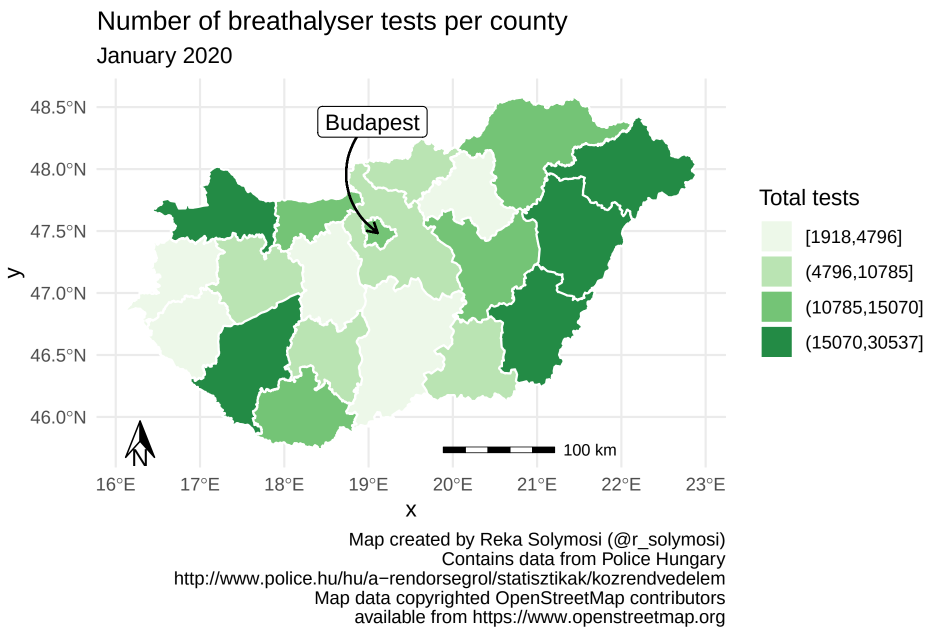
5.5 Composition

5.5.1 Orientation and scale indicators

```
library(ggspatial)

ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_curve(x = bp_x - 0.1,
             y = bp_y + 0.9,
             xend = bp_x ,
             yend = bp_y,
             arrow = arrow(length = unit(2, "mm"))) +
  geom_sf_label(data = hu_dd %>% filter(name == "Budapest"),
                aes(label = name),
                nudge_y = 0.9,
                nudge_x = -0.1) +
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020") +
  labs(caption = caption_text) +
  annotation_north_arrow(height = unit(7, "mm"), width = unit(5, "mm"))
```




```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_curve(x = bp_x - 0.1,
             y = bp_y + 0.9,
             xend = bp_x ,
             yend = bp_y,
             arrow = arrow(length = unit(2, "mm"))) +
  geom_sf_label(data = hu_dd %>% filter(name == "Budapest"),
                aes(label = name),
                nudge_y = 0.9,
                nudge_x = -0.1) +
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020") +
  labs(caption = caption_text) +
  annotation_north_arrow(height = unit(7, "mm"), width = unit(5, "mm"), pad_x = unit(5
  annotation_scale(line_width = 0.5, height = unit(1, "mm"), pad_x = unit(6, "cm"))
```

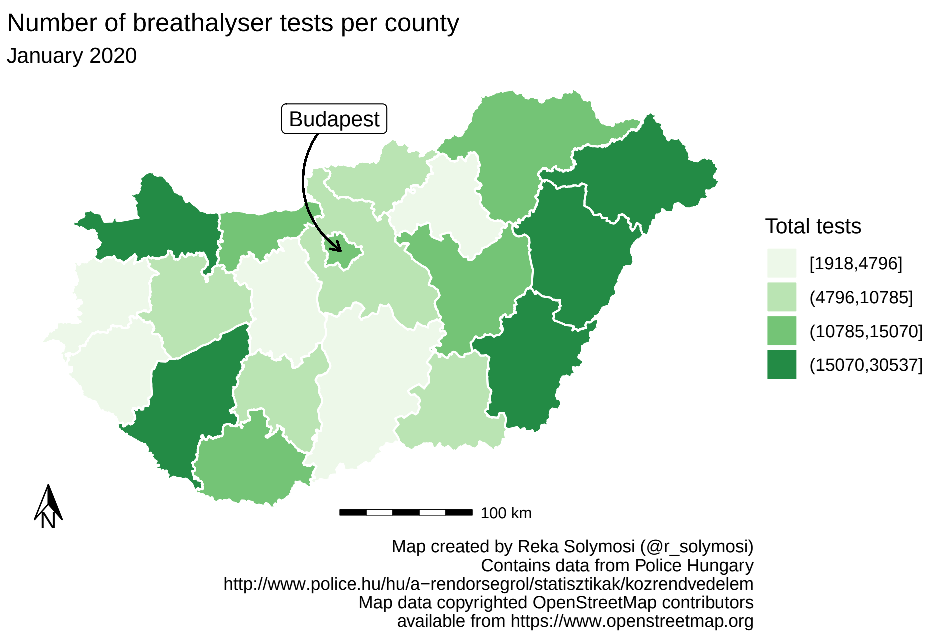


5.5.2 Strip away grids

Possibly the longitude and latitude and the grid are not adding much to our understanding of the map here. Instead we can strip these away. By using `theme_minimal()`, we're already stripping away a lot of the elements, but we can further specify to eliminate all elements with the `theme_void()` argument instead.

```
ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_curve(x = bp_x - 0.1,
             y = bp_y + 0.9,
             xend = bp_x ,
             yend = bp_y,
             arrow = arrow(length = unit(2, "mm"))) +
  geom_sf_label(data = hu_dd %>% filter(name == "Budapest"),
                aes(label = name),
                nudge_y = 0.9,
                nudge_x = -0.1) +
  theme_void() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020") +
  labs(caption = caption_text) +
  annotation_north_arrow(height = unit(7, "mm"), width = unit(5, "mm"), pad_x = unit(5, "mm"), pa
```

```
  annotation_scale(line_width = 0.5, height = unit(1, "mm"), pad_x = unit(6, "cm"))
```



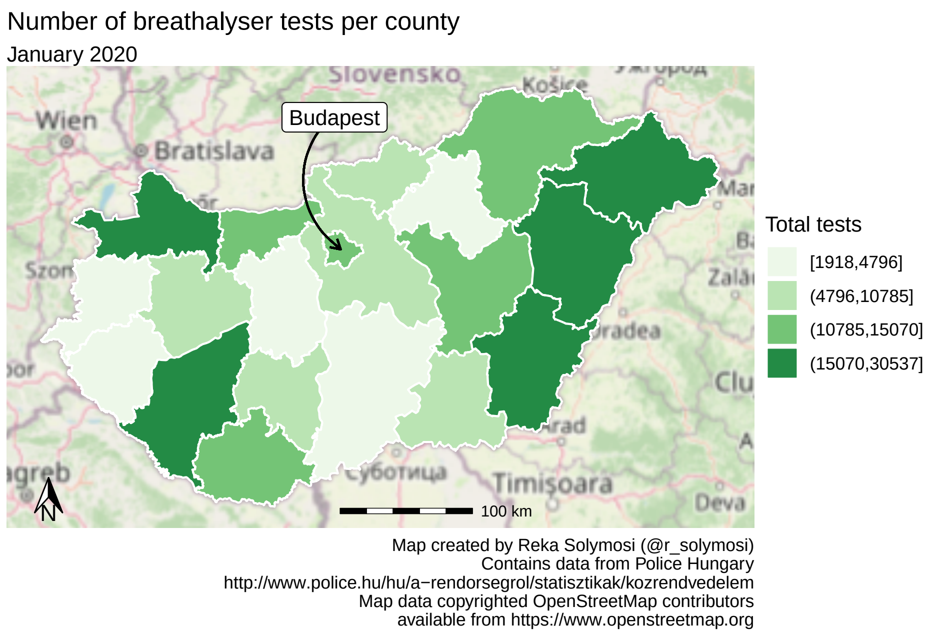
5.6 Context

5.6.1 Basemap

The `annotation_map_tile()` function also from the `ggspatial` package.

```
ggplot(data = hu_dd) +
  annotation_map_tile() +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_curve(x = bp_x - 0.1,
             y = bp_y + 0.9,
             xend = bp_x ,
             yend = bp_y,
             arrow = arrow(length = unit(2, "mm"))) +
  geom_sf_label(data = hu_dd %>% filter(name == "Budapest"),
                aes(label = name),
                nudge_y = 0.9,
                nudge_x = -0.1) +
  theme_void() +
```

```
ggtitle("Number of breathalyser tests per county", subtitle = "January 2020") +
labs(caption = caption_text) +
annotation_north_arrow(height = unit(7, "mm"), width = unit(5, "mm"), pad_x = unit(5, "mm"), pa
annotation_scale(line_width = 0.5, height = unit(1, "mm"), pad_x = unit(6, "cm"))
```



5.6.2 Inset maps

Inset maps provide context. You might use this to show where your main map fits into the context of a larger area, for example, here we might illustrate how Hungary is situated within Europe. You might also use an inset map in another situation, where you have additional areas which you want to show which may be geographically far but politically related to your region. For example, we might want to portray a map of the United States of America, and make sure to include Hawaii and Alaska on the map. The basic principles behind these maps is the same. Essentially we must create two map objects, and then bring these together. Let's illustrate how.

First, we need to create the map we will be displaying in the inset map. In this case, let's highlight the location of Hungary on the map. We can do this by creating a map of Europe (let's use the `rnaturalearth` package for this). We create a list of the countries from the world map 'countries1101, and filter only Europe (we also exclude Russia because it is so big it makes the rest of Europe hard do see on a smaller map, and Iceland as it's far, also making the map bigger than we need).

```
library(rnaturalearth)

europe_countries <- st_as_sf(countries110) %>%
  filter(region_un=="Europe" & name != "Russia" & name != "Iceland") %>% pull(name)

europe <- ne_countries(geounit = europe_countries, type = 'map_units', returnclass = "s
```

Now we can create a map where we highlight Hungary by making its border red and a thicker line width. Remember to add `theme_void()` in order to remove any background elements.

```
inset_map <- ggplot() +
  geom_sf(data = europe, fill = "white") +
  geom_sf(data = europe %>% filter(name == "Hungary"), fill = "white" , col = "red", lw
  theme_void() +
  theme(panel.border = element_rect(colour = "black", fill=NA))
```

We now have this separate map, which highlights where Hungary can be found, right there in Central Europe. To display this jointly with our map of breathalyser test, we must join the two maps. We will need both as objects. We've already assigned out inset map to the object `inset_map` but we must take our main map, and also assign it to an object. Let's call this main map.

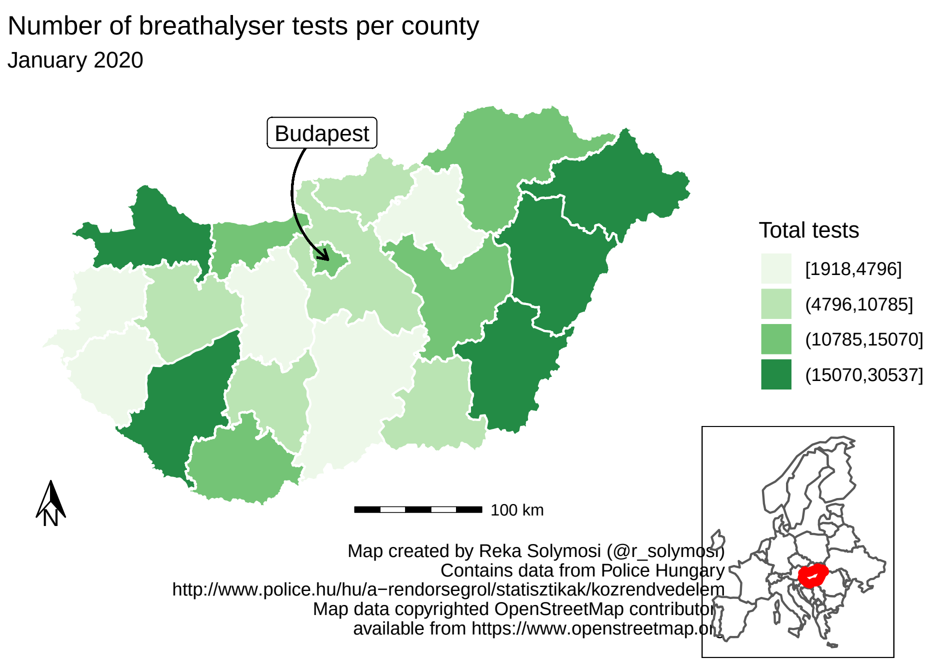
```
main_map <- ggplot(data = hu_dd) +
  geom_sf(aes(fill = total_quantiles), lwd = 0.5, col = "white") +
  scale_fill_brewer(type = "seq", palette = "Greens", name = "Total tests") +
  geom_curve(x = bp_x - 0.1,
             y = bp_y + 0.9,
             xend = bp_x ,
             yend = bp_y,
             arrow = arrow(length = unit(2, "mm"))) +
  geom_sf_label(data = hu_dd %>% filter(name == "Budapest"),
                aes(label = name),
                nudge_y = 0.9,
                nudge_x = -0.1) +
  theme_minimal() +
  ggtitle("Number of breathalyser tests per county", subtitle = "January 2020") +
  labs(caption = caption_text) +
  annotation_north_arrow(height = unit(7, "mm"), width = unit(5, "mm"), pad_x = unit(5
  annotation_scale(line_width = 0.5, height = unit(1, "mm"), pad_x = unit(6, "cm")) +
  theme(panel.grid = element_blank(),
        axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.background = element_blank())
```

Great, now to join them. For this, we will turn to functions from the `cowplot` package.

```
library(cowplot)
```

First, we set up an empty drawing layer for our ggplot using the `ggdraw()` function. Then we layer on the two maps both using the `draw_plot()` function. This allows us to draw plots and sub plots. This function places a plot (which we specify as the first parameter of this function) somewhere onto the drawing canvas. By default, coordinates run from 0 to 1, and the point (0, 0) is in the lower left corner of the canvas. We want to therefore specify where the plots go on our canvas explicitly. Alongside position, we can also specify size. This is important, we usually make the inset map smaller.

```
hu_dd_with_inset <- ggdraw() +
  draw_plot(main_map) +
  draw_plot(inset_map, x = 0.67, y = 0, width = 0.35, height = 0.35)

hu_dd_with_inset
```



You can play around with where you position your inset map by adjusting the x and y coordinates. You can also play around with the size of it by adjusting the parameters for height and width.

Andresen, Martin. 2011. "The Ambient Population and Crime Analysis." *The Professional Geographer* 63 (2): 193–212.

Andresen, Martin, Nick Malleson, Wouter Steenbek, Michael Townsley, and Christophe Vandeviver. 2020. "Minimum Geocoding Match Rates: An International Study of the Impact of Data and Areal Unit Sizes." *International Journal of Geographical Information Science* 34 (7): 1306–22.

Bolstad, Paul. 2019. *GIS Fundamentals: A First Text on Geographic Information Systems*. 6th ed. White Bear Lake, MN: Eider Press.

Bowers, Kate. n.d. "Risky Places: Crime Absorbers, Crime Radiators as Risky Places." https://play.kth.se/media/Risky+PlacesA+Crime+absorbers,+crime+radiators+as+risky+places,+Prof+Kate+Bowers,+UCL/0_zya4j354.

Brantingham, Patricia, and Paul Brantingham. 1995. "Criminality of Place: Crime Generators and Crime Attractors." *European Journal on Criminal Policy and Research* 3: 5–26.

Brewer, Cynthia. 2006. "Basic Mapping Principles for Visualizing Cancer Data Using Geographic Information Systems (GIS)." *American Journal of Preventive Medicine* 30 (2S): S25–36.

Briggs, Daniel, and Rubén Monge Gamero. 2017. *Dead-End Lives: Drugs and Violence in the City Shadows*. Bristol, UK: Policy Press.

Bruinsma, Gerben J. N., and Shane Johnson, eds. 2018. *The Oxford Handbook of Environmental Criminology*. Oxford, England: Oxford Univerisity Press.

Buil-Gil, David, Juanjo Medina, and Natalie Shlomo. 2021. "Measuring the Dark Figure of Crime in Geographic Areas: Small Area Estimation from the Crime Survey for England and Wales." *British Journal of Criminology* 61 (2): 364–88.

Cairo, Alberto. 2016. *The Truthful Art: Data, Charts, and Maps for Communication*. New Riders.

Camoes, Jorge. 2016. *Data at Work: Best Practices for Creating Effective Charts and Information Graphics in Microsoft Excel*. New Riders.

Carr, Daniel, and Linda Pickle. 2010. *Visualizing Data Patterns with Micromaps*. Boca Raton, CA: CRC Press.

Chainey, Spencer, and Jerry Rattcliffe. 2005. *GIS and Crime Mapping*. Chichester, England: John Wiley; Sons.

Constabulary, Her Majesty Inspectorate of. 2014. "Crime Recording: Making the Victim Count." Final Report. HMIC.

Dorling, D. 1996. "Area Cartograms: Their Use and Creation, Concepts and Techniques in Modern Geography."

Dougenik, James A, Nicholas R Chrisman, and Duane R Niemeyer. 1985. "An Algorithm to Construct Continuous Area Cartograms." *The Professional Geographer* 37 (1): 75–81.

Field, Kenneth. 2018. *Cartography.* Redlands, CA: ESRI Press.

Field, Kenneth, and Damien Demaj. 2012. "Reasserting Design Relevance in Cartography: Some Concepts." *The Cartographic Journal* 49 (1): 70–76.

Field, Kenneth, and Danny Dorling. 2016. "UK Election Cartography." *International Journal of Cartography* 2: 202–32.

Gelman, Andrew, and Phillip Price. 1999. "All Maps of Parameters Estimates Are Misleading." *Statistics in Medicine* 18: 3221–34.

Goudriaan, Heike, Karin Witterbrood, and Paul Nieuwbeerta. 2006. "Neigbourhood Characteristics and Reporting Crime." *British Journal of Criminology* 46: 719–42.

Grossenbacher, Timo. 2019. "Bivariate Maps with Ggplot2 and Sf." Blog. Tamedia. https://timogrossenbacher.ch/2019/04/bivariate-maps-with-ggplot2-and-sf/.

Harries, Keith. 1999. "Mapping Crime: Principle and Practice." Research Report. National Institute of Justice.

Hart, Timothy, and Paul Zandbergen. 2012. "Effects of Data Quality on Predictive Hotspot Mapping." Final Report. National Institute of Justice.

Healy, Kieran. 2019. *Data Visualization. A Practical Introduction.* Princeto, NJ: Princeton University Press.

Jiang, Bin. 2013. "Head/Tail Breaks: A New Classification Scheme for Data with a Heavy-Tailed Distribution." *The Professional Geographer* 65 (3): 482–94.

Kirk, Andy. 2016. *Data Visualisation: A Handbook for Data Driven Design.* London, UK: Sage.

Lawson, Andrew. 2021a. *Bayesian Disease Mapping.* 3rd ed. Boca Raton, CA: CRC Press.

———. 2021b. *Statistical Methods in Spatial Epidemiology.* 2nd ed. Chichester, UK: John Wiley; Sons.

———. 2021c. *Using r for Bayesian Spatial and Spatio-Temporal Health Modeling.* Boca Raton, CA: CRC Press.

Lovelace, Robin, Jakub Nowosad, and Jannes Muenchow. 2019. *Geocomputation with r.* Boca Raton, Florida: Chapman; Hall/CRC Press.

Marshall, Roger. 1991. "Mapping Disease and Mortality Rates Using Empirical Bayes Estimators." *Applied Statistics* 40 (2): 283–94.

Monmonier, Mark. 1993. *Mapping It Out: Expository Cartography for the Humanities and Social Sciences.* Chicago: The University of Chicago Press.

———. 1996. *How to Lie with Maps.* 2nd ed. Chicago: The University of Chicago Press.

Newton, Andrew David. 2018. "Macro-Level Generators of Crime, Including Parks, Stadiums, and Transit Stations." In *The Oxford Handbook of Environmental Criminology*, edited by Gerben J N Bruinsma and Shane D Johnson, 497–517. Oxford: Oxford University Press.

O´Sullivan, David, and David J. Unwin. 2010. *Geographic Information Analysis*. 2nd ed. Hoboken, NJ: John Wiley; Sons.

Pebesma, Edzer. 2018. "Simple Features for r: Standardized Support for Spatial Vector Data." *The R Journal* 10 (1): 439–46.

Ratcliffe, Jerry. 2004. "Geocoding Crime and a First Estimate of a Minimum Acceptable Hit Rate." *International Journal of Geographical Information Science* 18 (1): 61–72.

———. 2010. "Crime Mapping: Spatial and Temporal Challenges." In *Handbook of Quantitative Criminology*, edited by Alex Piquero and David Weisburd, 5–24. New York, NY: Springer.

Rengert, George F, and Brian Lockwood. 2009. "Geographical Units of Analysis and the Analysis of Crime." In *Putting Crime in Its Place*, edited by David Weisburd, Wim Bernasco, and Gerben J. N. Bruinsma, 109–22. New York, NY: Springer.

Santos, Rachel Boba. 2013. *Crime Analysis with Crime Mapping*. 3rd ed. Thousand Oaks, CA: Sage.

Schwabish, Jonathan. 2021. *Better Data Visualizations. A Guide for Scholars, Researchers, and Wonks*. New York: Columbia University Press.

Spatial, Modelling, and Spatial-Temporal Data. A Bayesian Approach. 2020. *Robert Haining and Guangquan Li*. Boca Raton, FL: CRC Press.

Stevens, Joshua. 2015. "Bivariate Choropleth Maps: A How-to Guide." Blog. NASA's Earth Observatory. https://www.joshuastevens.net/cartography/make-a-bivariate-choropleth-map/.

Tanimura, Susumu, Chusi Kuroiwa, and Tsutomu Mizota. 2006. "Propotional Symbol Mapping in r." *Journal of Statistical Software* 15 (5): 1–7.

Tukey, John. 1979. "Statistical Mapping: What Should Not Be Plotted." In *Proceedings of the 1976 Workshop on Automated Cartography and Epidemiology*, edited by DHEW, 18–21. Washington, DC: US Department of Health, Education,; Welfare.

Waller, HLance, and Carol Gotway. 2004. *Applied Spatial Statistics for Public Health Data*. Chichester, UK: John Wiley; Sons.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19 (1): 3–28.

Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. Boston, MA: O´Reilly.