

Crime Mapping and Spatial Data Analysis using R

Juanjo Medina and Reka Solymosi

2020-04-15

Contents

Prelude	7
0.1 Introduction	7
0.2 Disclaimer	7
1 A first lesson about R	9
1.1 Install R & RStudio	9
1.2 Open up and explore RStudio	9
1.3 Customising the RStudio look	12
1.4 Getting organised: R Projects	13
1.5 Talk to your computer	16
1.6 More on packages	17
1.7 Using objects	21
1.8 More on objects	22
1.9 Vectors	24
1.10 On comments	26
1.11 Factors	26
1.12 Naming conventions for objects in R	28
1.13 Dataframes	29
1.14 Exploring data	31
1.15 Quitting RStudio	36

2 Making your first maps in R	39
2.1 A quick introduction of terms	39
2.2 Getting some spatial data to put on a map	45
2.3 From dataframes to spatial objects: finding shapefiles	52
2.4 Reading shapefiles into R: the wonderful world of sf objects	59
2.5 Add some data	61
2.6 Join data to spatial object	63
2.7 So left_join(), eh?	66
2.8 Let's make this prettier: a quick glance at tmap	67
3 Thematic maps in R	69
3.1 Intro and recap	69
3.2 Creating choropleth maps	71
3.3 Producing small multiples to compare the effect of different classification systems	76
3.4 Using graduated symbols	79
3.5 Bringing additional census data in	81
3.6 Computing and mapping crime rates	84
3.7 More on small multiples and point pattern maps	87
4 Performing spatial operations in R	95
4.1 Getting some (more) data	97
4.2 Making interactive maps with leaflet	118
4.3 Adding points manually:	119
4.4 Spatial operations	121
4.5 Recap	133
4.6 Homework	133
5 More on thematic maps	135
5.1 Introduction	135
5.2 Mapping rates, learning from disease mapping	138
5.3 Binning points	142
5.4 A note of caution: MAUP	153

CONTENTS	5
5.5 Replacing polygons with grid or hex shapes	156
5.6 Cartograms	160
5.7 References and further reading	163
6 Studying spatial point patterns	165
6.1 What we'll do today	165
6.2 Getting the data	166
6.3 Getting the data into spatstat: the problem with duplicates . . .	168
6.4 Inspecting our data with spatstat	172
6.5 Density estimates	175
6.6 Adding some context	182
6.7 Spatial point patterns along networks	183
7 Global and local spatial autocorrelation	189
7.1 Get data	189
7.2 What is a neighbour?	194
7.3 Putting ‘neighbourhood’ in our analysis - constructing a spatial weight matrix	196
7.4 Creating a list of neighbours	198
7.5 Generating the weight matrix	201
7.6 Moran’s I	204
7.7 Local spatial autocorrelation	208
7.8 Getting Manchester data and weights	209
7.9 Generating and visualising the LISA measures	211
8 Regression analysis (a refresher)	223
8.1 Introduction	223
8.2 Motivating regression	227
8.3 Fitting a simple regression model	231
8.4 Residuals revisited: R squared	234
8.5 Inference with regression	238
8.6 Fitting regression with categorical predictors	241
8.7 Motivating multiple regression	244

8.8	Fitting and interpreting a multiple regression model	245
8.9	Presenting your regression results.	247
8.10	Rescaling input variables to assist interpretation	253
8.11	Testing conditional hypothesis: interactions	255
8.12	Model building and variable selection	258
8.13	Regression assumptions	259
9	Spatial regression models	261
9.1	Introduction	261
9.2	Looking at the residuals and testing for spatial autocorrelation in regression	263
9.3	What to do now?	268
9.4	Spatial Regimes	269
9.5	Lagrange multipliers	270
9.6	Fitting and interpreting a spatially lagged model	274
9.7	Fitting an interpreting a spatial error model	283
10	Time matters	289
10.1	Getting time series data into R and plotting it	290
10.2	Lubridate: your guardian angel when it comes to working with temporal data	294
10.3	Calendar heatmaps	298
10.4	Decomposing time series	300
10.5	Static maps with ggplot2	303
10.6	Small multiples to show temporal variation	309
10.7	Spaghetti plots	311
10.8	Animations!!!	314

Prelude

0.1 Introduction

This workbook contains the lab materials and homework assignments for an introduction to crime mapping course designed for LAWS31152 and 60142 Crime Mapping, an optional module open to 3rd year undergraduates on the BA Criminology programme as well as postgraduates on the MA and MRes Criminology programmes at the University of Manchester.

It makes use of R, as it is a free, open source tool, that has tremendous community support, and great versatility in mapping applications. You can find more details about the advantages of R for geospatial work [here](#)

Crime Mapping introduces students to the concepts of spatial data analysis. The aim is to familiarise students with basic concepts of GIS, and get acquainted with spatial statistics to be able to talk about data about crime, policing, and criminal justice topics situated in the places they occur. Details can be found in the Syllabus.

0.2 Disclaimer

Please beware that:

- In making these notes, while we briefly cover some concepts, students are expected to do the weekly reading, and attend the weekly lectures, as well as participate in lab discussions to receive a complete course experience. These notes are *not* intended to be a stand-alone reference or textbook, rather a set of exercises to gain hands-on practice with the concepts introduced during the course.
- These pages are the content of the university course mentioned above. They are meant to (very gently) introduce students to the concept of spatial data analysis, and cover descriptive statistics and the key concepts required to build an understanding of quantitative data analysis in crime research.

- The handouts below use, among other data sets, data from the UK data service such as the Crime Survey for England and Wales that is available under a Open Government Licence. This dataset is designed to be a learning resource and should not be used for research purposes or the production of summary statistics.

Chapter 1

A first lesson about R

1.1 Install R & RStudio

We recommend that you use your own laptops for this course. This way you get used to working in an environment which you will continue to use after this semester. However, our lab sessions will be held in computer clusters in case you do not have access to a laptop (or something goes wrong...).

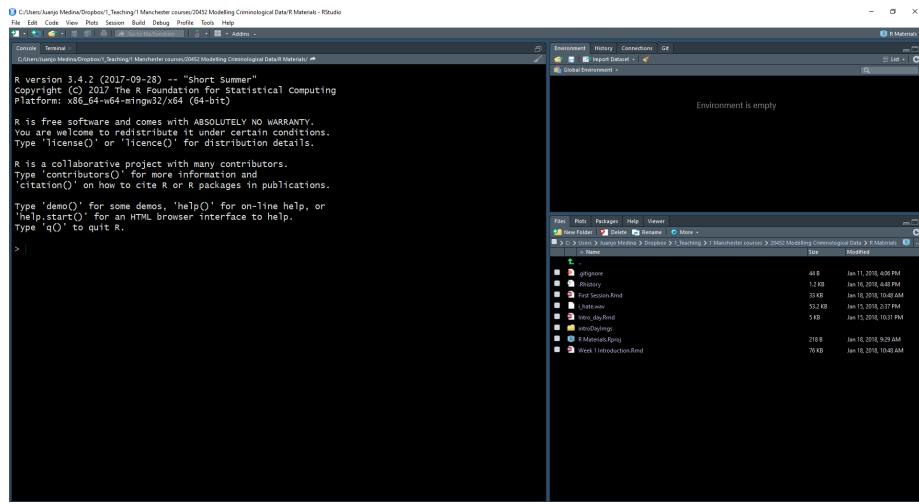
You *don't need to install the software in the computers available in the clusters*, because it is already there. Beware though, the installation may vary a bit across different computer clusters in the University. This, on itself, is another good reason to use your own laptops -for it will provide you with a more stable environment. If you have not already, then please download and install R and R Studio onto your laptops. Otherwise use the cluster machines.

- click [here](#) for instructions using Windows or
- [here](#) for instructions using a Mac.

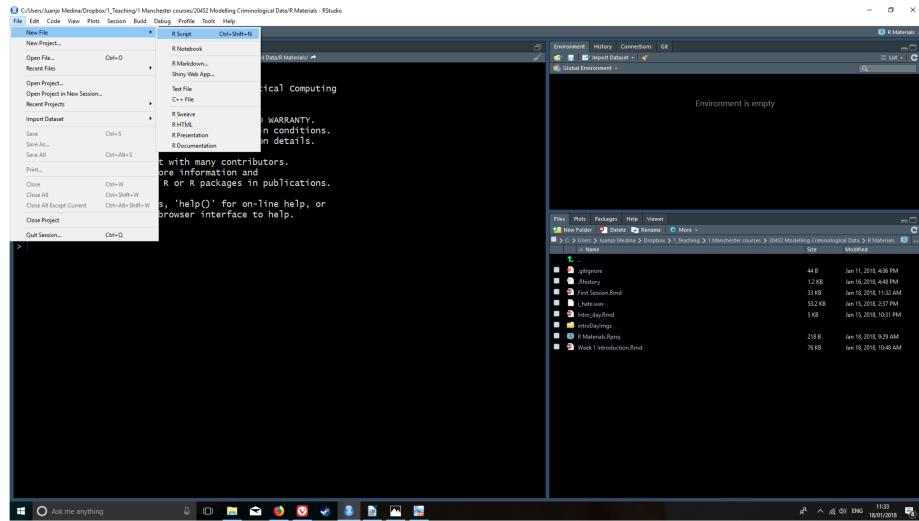
1.2 Open up and explore RStudio

In this session we will focus in developing basic familiarity with R Studio. You can use R without using R Studio, but R Studio is an app that makes it easier to work with R.

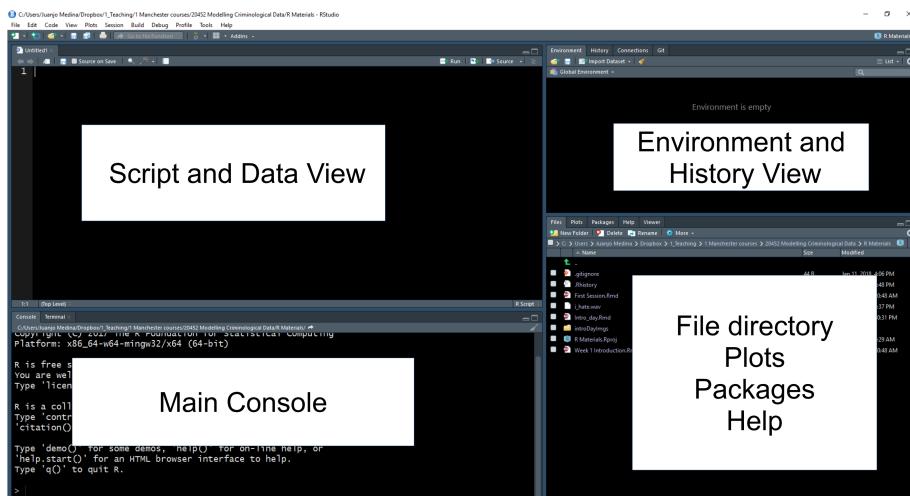
R Studio is what we call an IDE, an **integrated development environment**. It is a fancy way of saying that it is a cool interface designed to write programming code. Every time you open up R Studio you are in fact starting a R session. R Studio automatically runs R in the background. We will be interacting with R in this course unit via R Studio.



When you first open R Studio, you will see (as in the image above) that there are 3 main windows. The bigger one to your left is the console. If you read the text in the console you will see that R Studio is indeed opening R and you can see what version of R you are running. Depending on whether you are using the cluster machines or your own installation this may vary, but don't worry too much about it. R is constantly being updated.



The view in R Studio is structured so that you have 4 open windows in a regular session. Click in the *File* drop down Menu, select *New File*, then *R Script*. You will now see the 4 window areas in display. On each of these areas you can shift between different views and panels. You can also use your mouse to re-size the different windows if that is convenient.



Look for example at the bottom right area. Within this area you can see that there are different tabs, which are associated with different views. You can see in the tabs in this section that there are different views available: *Files*, *Plots*, *Packages*, *Help*, and *Viewer*. The **Files** allow you to see the files in the physical directory that is currently set up as your working environment. You can think of it like a window in Windows Explorer that lets you see the content of a folder.

In the **plots** panel you will see any data visualisations or graphical displays of data that you produce. We haven't yet produced any, so it is empty at the moment. If you click in **packages** you will see the packages that are currently available in your installation. What is a "package" in this context?

You can think of R as a Lego monster. You can make the monster scarier and more powerful by adding new bits to it. Packages are those bits. They are modules that expand what R can do. There are thousands of them. Which is pretty cool!!! R can do many more things than Excel. That is down to the fact that researchers all over the world write packages that continuously expand the functionality of R. You can think of a package as another drop down menu that gets added to you menu tab with loads of new options for doing fancy stuff, only they are not really drop down menus. You need to access their added functionality via programming code. So yeah, R is like Excel or SPSS only with over 10,000 "drop down menus." And all for free.

The other really useful panel in this part of the screen is the **Help** viewer. Here you can access the documentation for the various packages that make up R. Learning how to use this documentation will be essential if you want to be able to get the most from R.

In the diagonally opposite corner, the top left, you should now have an open script window. The **script** is where you write your programming code. A script is nothing but a text file with some code on it. Unlike other programs for data analysis you may have used in the past (Excel, SPSS), you need to interact

with R by means of writing down instructions and asking R to evaluate those instructions. R is an *interpreted* programming language: you write instructions (code) that the R engine has to interpret in order to do something. And all the instructions we write can and should be saved in a script, so that you can return later to what you did.

One of the key advantages of doing data analysis this way - with code versus with a point and click interface like Excel or SPSS is that you are producing a written record of every step you take in the analysis. First time around it will take you time to write these instructions, it may be slower than pointing and clicking. And unlike with pointing and clicking you need to know the “words” and “grammar” of this language.

Luckily you don’t need to memorise or know all these words. As with any language the more you practice it, the easier it will become. More often than not you will be doing a lot of cutting and pasting from chunks of code we will give you. But we will also expect you to develop a basic understanding of what these bits of code do. It is a bit like cooking. At first you will just follow recipes as they are given to you, but as you become more comfortable in your “kitchen” you will feel more comfortable experimenting.

The advantage of doing analysis this way is that once you have written your instructions and saved them in a file, you will be able to share it with others and run it every time you want in a matter of seconds. This creates a *reproducible* record of your analysis: something that your collaborators or someone else anywhere (including your future self, the one that will have forgotten how to do the stuff) could run and get the same results than you did at some point earlier. This makes science more transparent and transparency brings with it many advantages. For example, it makes your research more trustworthy. Don’t underestimate how critical this is. **Reproducibility** is becoming a key criteria to assess good quality research. And tools like R allow us to enhance it. You may want to read more about reproducible research [here](#).

1.3 Customising the RStudio look

R Studio allows you to customise the way it looks. Working with white backgrounds is not generally a good idea if you care about your eyesight. If you don’t want to end up with dry eyes not only it is good you follow the 20-20-20 rule (every 20 minutes look for 20 seconds to an object located 20 feet away from you), but it may also be a good idea to use more eye friendly screen displays.

Click in the *Tools* menu and select *Global options*. This will open up a pop up window with various options. Select *Appearance*. In this section you can change the font type and size, but also the kind of theme background that R will use in the various windows. I suffer from poor sight, so I often increase the font type. I also use the *Tomorrow Night Bright* theme to prevent my eyes to go too dry

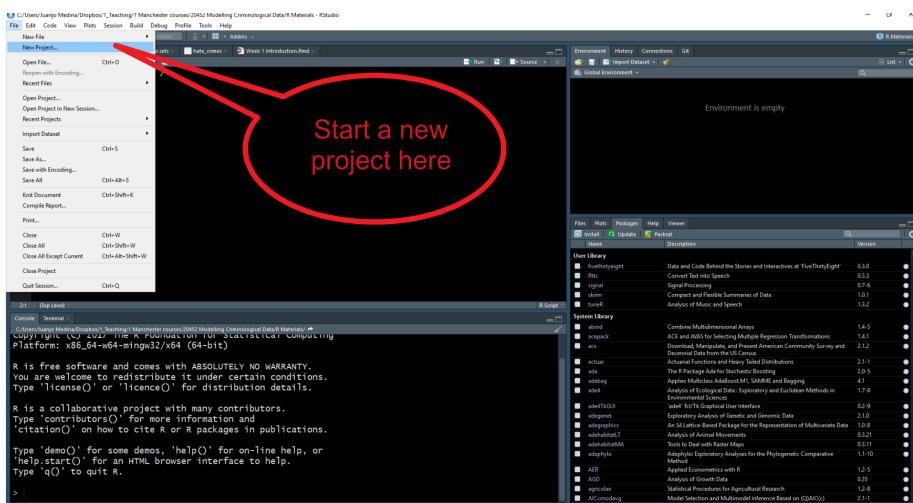
from the effort of reading a lightened screen, but you may prefer a different one. You can preview them and then click apply to select the one you like. This will not change your results or analysis. This is just something you may want to do in order to make things look better and healthier for your.

1.4 Getting organised: R Projects

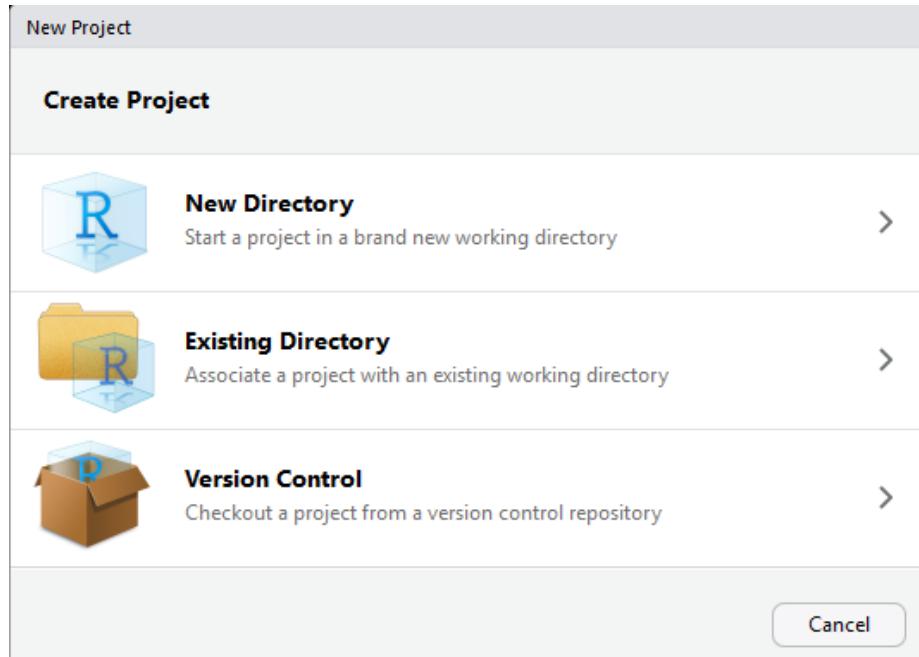
We finished the previous section talking about sharing your analytic code. Let's face it. You would not bring a new partner or somebody that you want to impress to your place before tidying a little bit first, wouldn't you? In the same way, if you know you may have to share your code, if you know you may have guests, you may want to keep your analysis, data, and results tidy. R Studio helps a little bit with that. R Studio helps with this by virtue of something called **R Projects**.

Technically, a R Studio project is just a directory with the name of the project, and a few files and folders created by R Studio for internal purposes. This is where you should hold your scripts, your data, and reports. You can manage this folder with your own operating system manager (eg., Windows Explorer) or through the R Studio file manager (that you access in the bottom right corner set of windows in R Studio).

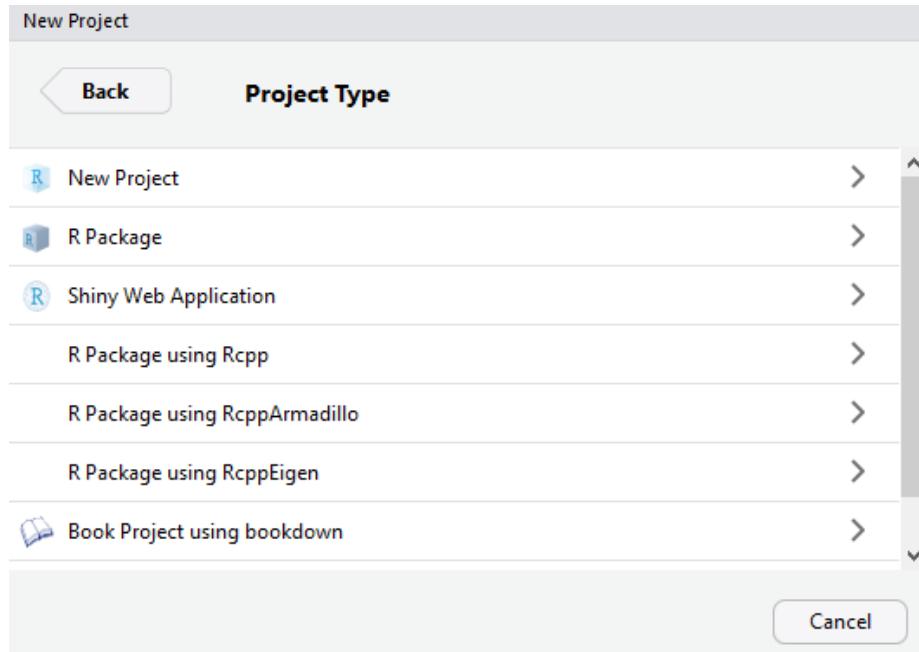
When a project is reopened, R Studio opens every file and data view that was open when the project was closed last time around. Let's learn how to create a project. Go to the *File* dropdown menu and select *New Project*.



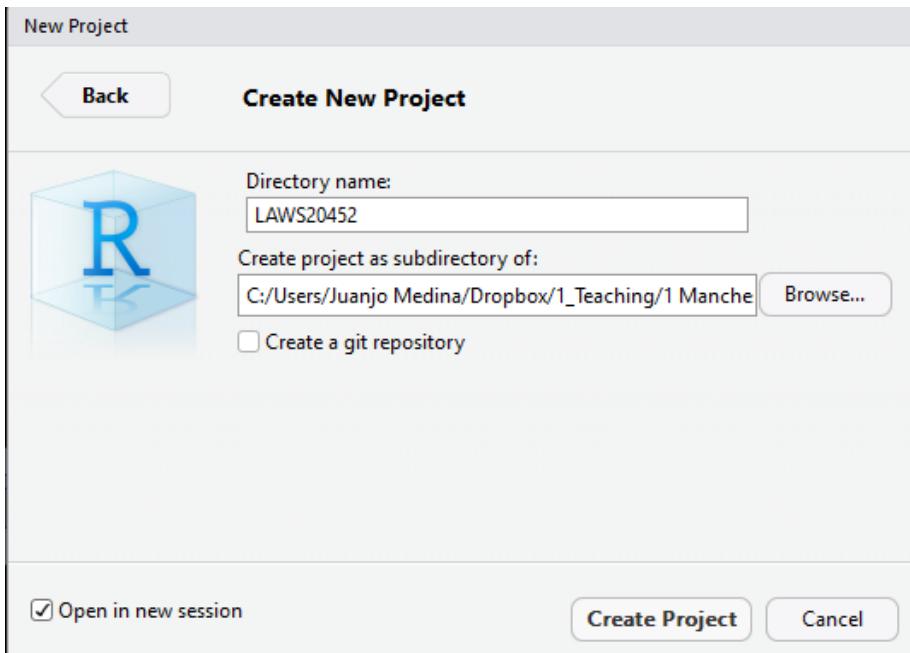
That will open a dialog box where you ask to specify what kind of directory you want to create. Select new working directory in this dialog box.



Now you get another dialog box (at least you have an older version of R Studio) where you have to specify what kind of project you want to create. Select the first option *New Project*.



Finally, you get to select a name for your project (in the image below I use the code for this course unit, but you can use any sensible name you prefer) and you will need to specify the folder in which to place this directory. If you are using a cluster machine use the P: drive, otherwise select what you prefer in your laptop (preferably not your desktop in Windows machines).



With simple projects a single script file and a data file is all you may have. But with more complex projects, things can rapidly become messy. So you may want to create subdirectories within this project folder. I typically use the following structure in my own work to put all files of a certain type in the same subdirectory:

- *Scripts and code:* Here I put all the text files with my analytic code, including rmarkdown files which is something we will introduce much later in the semester.
- *Source data:* Here I put the original data. I tend not to touch this once I have obtained the original data.
- *Documentation:* This is the subdirectory where I place all the data documentation (e.g., codebooks, questionnaires, etc.)
- *Modified data:* All analysis involve doing transformations and changing things in the original data files. You don't want to mess up the original data files, so what you should do is create new data files as soon as you start changing your source data. I go so far as to place them in a different subdirectory.

- *Literature*: Analysis is all about answering research questions. There is always a literature about these questions. I place the relevant literature for the analytic project I am conducting in this subdirectory.
- *Reports and write up*: Here is where I file all the reports and data visualisations that are associated with my analysis.

If you come to my office, you will see it is a very messy place. But my computer is, in contrast, a very tidy environment. You should aim for your computer workspace to be very organised as well. You can create these subdirectories using Windows Explorer or the Files window in R Studio. Why don't you have a go?

1.5 Talk to your computer

Enough background, let's write some very simple code to talk to your computer. First open a new script within the project you just created. Type the following instructions in the script window. After you are done click in the top right corner where it says *Run* (if you prefer quick shortcuts, you can select the text and then press Ctrl + Enter):

```
print("I hate computers")
## [1] "I hate computers"
```

Congratulations!!! You just run your first line of R code! Though that was a really mean thing to say to your machine!

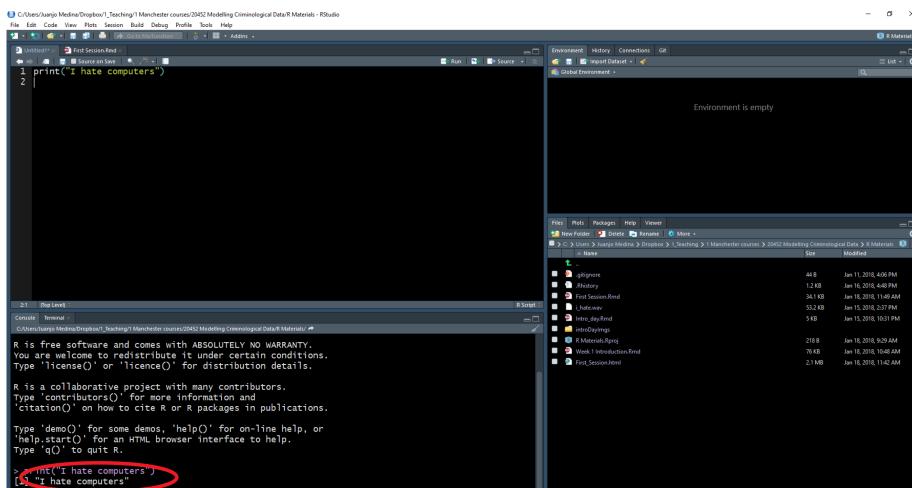
In these handouts (written in html format) you will see grayed boxes with bit of code on it. You can cut and paste this code into your script window and run the code from it to reproduce our results. As we go along we will be covering new bits of code. You should start thinking about creating a file with some of the most useful bits of code we cover so that when you do your work you can just cut and paste from this file rather than having to come back to these handouts.

Sometimes in these documents you will see on them the results of running the code, what you see printed in your console or in your plot viewer. The results will appear enclosed in a box as above.

The R languages uses **functions** to tell the computer what to do. In the R *language* functions are the *verbs*. You can think of functions as predefined commands that somebody has already programmed into R and tell R what to do. Here you learnt your first R function: *print*. All this function does is to ask R to print whatever it is you want in the main console (see the window in the bottom left corner).

In R, you can pass a number of **arguments** to any function. These arguments control what the function will do in each case. The arguments appear between brackets. Here we passed the text “I hate computers” as an argument. Once you execute the program, by clicking on *Run*, the R engine sends this to the CPU of your machine in the form of binary code and this produces a result. In this case we see that result printed in the main console.

Every R function admits different kind of arguments. Learning R involves not only learning different functions but also learning what are the valid arguments you can pass to each function. If you want to know more about a specific function, move your cursor onto it and press the F1-key. The documentation of the function will then appear in the **Help**-viewer.



As indicated above, the window in the bottom left corner is the main **console**. You will see that the words “I hate computers” appear printed there. If rather than using R Studio you were working directly from R, that’s all you would get: the main console where you can write code interactively (rather than all the different windows you see in R Studio). You can write your code directly in the main console and execute it line by line in an interactive fashion. However, we will be running code from scripts, so that you get used to the idea of properly documenting all the steps you take,

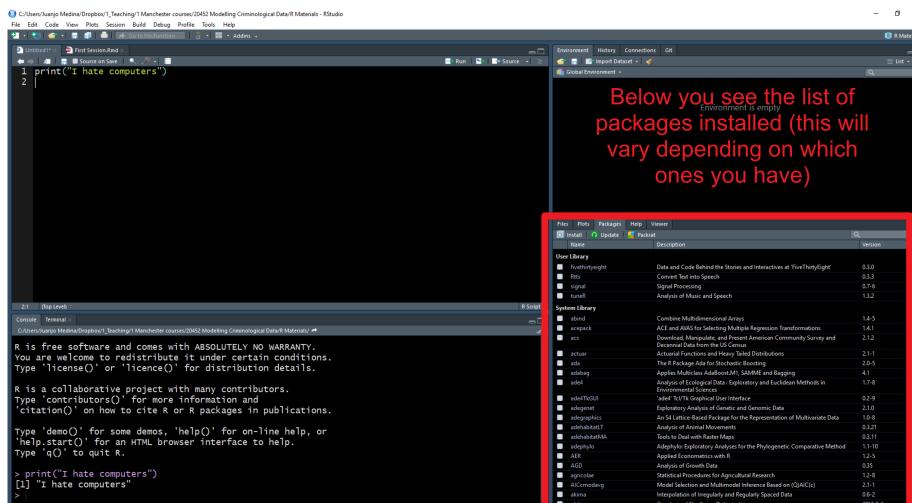
1.6 More on packages

Before we described packages as elements that add the functionality of R. What most packages do is they introduce new functions that allow you to ask R to do new different things.

Anybody can write a package, so consequently R packages vary on quality and complexity. You can find packages in different places, as well, from official

repositories (which means they have passed a minimum of quality control), something called Git Hub (a webpage where software developers post work in progress), to personal webpages (danger danger!). In early 2017 we passed the 10,000 mark just in the main official repository, so the number of things that can be done with R grows exponentially every day as people keep adding new packages.

When you install R you only install a set of basic packages, not the full 10,000 plus. So if you want to use any of these added packages that are not part of the basic installation you need to first install them. You can see what packages are available in your local install by looking at the *packages* tab in the bottom right corner panel. Click there and check. We are going to install a package that is not there so that you see how the installation is done.



If you just installed R in your laptop you will see a shortish list of packages that constitute the basic installation of R. If you are using one of the machines in the computer cluster this list is a bit longer, because we asked IT to install some of the most commonly used packages. But knowing how to install packages is pretty essential, since you will want to do it very often.

We are going to install a package called “cowsay” to demonstrate the process. In the Packages panel there is an *Install* menu that would open a dialog box and allows you to install packages. Instead we are going to use code to do this. Just cut and paste the code below into your script and then run it:

```
install.packages("cowsay")
```

Here we are introducing a new function “`install.packages`” and what we have passed as an argument is the name of the package that we want to install. This is how we install a package *that is available in the official CRAN repository*. If

we wanted to install a package from somewhere else we would have to adapt the code. Later this semester you will see how we install packages from Git Hub.

This line of code (as it is currently written) will install this package in a personal library that will be located in your P: drive if you are using a cluster machine. If you are using a Windows machine this will place this package within a personal library in your Documents folder. Once you install a package is in the machine/location where you install it until you physically delete it. The code we have used by default connects to a cloud repository called CRAN that has a collection of R packages that meet a minimum set of quality criteria. CRAN is the official repository of all things R. It's a fairly safe place to get packages from. But beware, judging whether a package is good or not requires your input. We will come back to this several times during the semester to help you make wise choices regarding packages. Given that you are connecting to an online repository you will need an internet connection every time you want to install a package.

How do you find out what a package does? You look at the relevant documentation. In the Packages window scroll down until you find the new package we installed listed. Here you will see the name of the package (*cowsay*), a brief description of what the program is about, and the version you have installed (an indication that a package is a good package is that it has gone through several versions, that means that someone is making sure the package gets regular updates and improvements). The version I have for *cowsay* is 0.7.0. Yours may be older or newer. It doesn't matter much at this point.

Click in the name *cowsay*. You will see that R Studio has now brought you to the Help tab. Here is where you find the help files for this package, including all the available documentation.

Every beginner in R will find these help files a bit confusing. But after a while, their format and structure will begin to make sense to you. Click where it says *User guides, package vignettes, and other documentation*. Documentation in R has become much better since people started to write **vignettes** for their packages. They are little tutorials that explain with examples what each package does. Click in the *cowsay::cowsay_tutorial* that you see listed here (the html link). What you will find there is an html file that gives you a detailed tutorial on this package. You don't need to read it now, but remember that this is one way to find help when using R. You will learn to love vignettes.

Let's try to use some of the functions of this package. We will use the "say" function:

```
say("I hate computers")
```

You will get an error message telling you that this function could not be found. What happened? Installing a package is only the first step. The next step, when you want to use it in a given session, is to **load** it.

Think of it as a pair of shoes. You buy it once, but you have to take them from your closet and put them on when you want to use them. Same with packages, you only install once, but need to load it from your library every time you want to use it -within a given session (once loaded it will remain loaded until you finish your session).

To see what packages you currently have **loaded** in your session, you use the **search()** function (you do not need to pass it any arguments in this case).

```
search()
```

```
## [1] ".GlobalEnv"           "package:stats"      "package:graphics"
## [4] "package:grDevices"    "package:utils"       "package:datasets"
## [7] "package:methods"       "Autoloads"         "package:base"
```

If you run this code, you will see that **cowsay** is not in the list of loaded packages. To load a package we use the **library** function. So if we want to load the new package we installed in our machine we would need to use the following code:

```
library("cowsay")
```

Run the **search** function again. You will see now this package is listed. So now we can try using the function “say” again.

```
say("I hate computers")
```

```
##
## -----
## I hate computers
## -----
##   \
##     \
##       \
##         \|_--/|
##         ==) ^Y^ (==
##             \  ^  /
##             )===( 
##             /      \
##             |      |
##             /| | | |\ \
##             \| | | |\ /\
##             jgs  //_// ___/
##                   \_)
```

You get a random animal in the console repeating the text we passed as an argument. If we like a different animal we could pass an argument to select it. So, say, we want to have cow rather than a random animal, then we would pass the following arguments to our function.

```
say("I hate computers", "cow")
```

```
##  
## -----  
## I hate computers  
## -----  
##      \  ^__^  
##      \  (oo)\-----  
##          (__)\       )\/\ /\  
##              ||----w|  
##              ||     ||
```

Remember, you only have to install a package that is not already installed once. But if you want to use it in a given session you will have to load it within that session using the `library` function. Once you load it within a session the package will remain loaded until you terminate your session (for example, by closing R Studio).

1.7 Using objects

We have seen how the first argument that the “say” function takes is the text that we want to convert into speech for our given animal. We could write the text directly into the function, but now we are going to do something different. We are going to create an object to store the text.

An **object**? What do I mean? In the same way that everything you do in R you do with functions (your verbs), everything that exists in R is an object. You can think of objects as boxes where you put stuff. In this case we are going to create an object called `my_text` and inside this object we are going to store the text “I hate computers”. How do you do this? We will use the code below:

```
my_text <- "I hate computers."
```

This bit of code is simply telling R we are creating a new object with the assigned name (“`my_text`”). We are creating a box with such name and inside this box we are placing a bit of text (“I hate computers”). The arrow you see is the **assignment operator**. This is an important part of the R language that tells R what we are including inside the object in question.

Run the code. Look now at the *Environment* window in the right top corner. We see that this object is now listed there. You can think of the Environment as a warehouse where you put stuff in, your different objects. Is there a limit to this environment? Yes, your RAM. R works on your RAM, so you need to be aware that if you use very large objects you will need loads of RAM. But that won't be a problem you will encounter in this course unit.

Once we put things into these boxes or objects we can use them as arguments in our functions. See the example below:

```
say(my_text, "cow")
```

```
##  
##  -----  
## I hate computers.  
##  -----  
##      \  ^__^  
##      \  (oo)\-----  
##          (__)\       )\/\  
##              ||----w|  
##              ||     ||
```

1.8 More on objects

Isn't this a course on data analysis? Yes, of course, but before we get there, you need to get used to the basics of R and R Studio, which is what we will be doing in these early sessions. Let's go through some of these basics a bit more slowly to ensure you get them and then we will bring some data you can look at.

In Excel you are used to see your data in a spreadsheet format. If you did the prep for this session, you should have reviewed some of the materials we covered in *Making Sense of Criminological Data* last semester. You should be familiar with the notion of a data set, levels of measurement, and tidy data. If you have not. This is your chance to do it in this link.

R is considerably more flexible than Excel. Most of the work we do here will use data sets or **dataframes** as they are called in R. But as you have seen earlier you can have *objects* other than data frames in R. These objects can relate to external files or simple textual information ("I hate computers"). This flexibility is a big asset because among other things it allows us to break down data frames or the results from doing analysis on them to its constitutive parts (this will become clearer as we go along).

Technically R is an *Object Oriented language*. Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic.

As we have seen earlier, to create an object you have to give it a name, and then use the assignment operator (the `<-` symbol) to assign it some value.

For example, if we want to create an object that we name “`x`”, and we want it to represent the value of 5, we write:

```
x <- 5
```

We are simply telling R to create a **numeric object**, called `x`, with one element (5) or of length 1. It is numeric because we are putting a number inside this object. The length is 1 because it only has one element on it, the number 5.

You can see the content of the object `x` in the main console either by using the `print` function we used earlier or by auto-printing, that is, just typing the name of the object and running that as code:

```
x
```

```
## [1] 5
```

When writing expressions in R is very important you understand that **R is case sensitive**. This could drive you nuts if you are not careful. More often than not if you write an expression asking R to do something and R returns an error message, chances are that you have used lower case when upper case was needed (or vice-versa). So always check for the right spelling. For example, see what happens if I use a capital ‘`X`’:

```
X
```

```
## Error in eval(expr, envir, enclos): object 'X' not found
```

You will get the following message: “`Error in eval(expr, envir, enclos): object 'X' not found`”. R is telling us that `X` does not exist. There isn’t an object `X` (upper case), but there is an object `x` (lower case). Error messages in R are pretty good at telling you exactly what went wrong.

Remember computers are very literal. They are like dogs. You can tell a dog “sit” and if it has been trained it will sit. But if you tell a dog “would you be so kind as to relax a bit and lay down in the sofa?”, it won’t have a clue what you are saying and will stare at you like you have gone mad. Error messages are computers ways of telling us “I really want to help you but I don’t really understand what you mean” (never take them personal, computers don’t hate you).

When you get an error message or implausible results, you want to look back at your code to figure out what is the problem. This process is called **debugging**.

There are some proper systematic ways to write code that facilitate debugging, but we won't get into that here. R is very good with automatic error handling at the levels we'll be using it at. Very often the solution will simply involve correcting the spelling.

A handy tip is to cut and paste the error message into Google and find a solution. If anybody had given me a penny for every time I had to do that myself, I would be Bill Gates by now. People make mistakes all the time. It's how we learn. Don't get frustrated, don't get stuck. Instead look for a solution. These days we have Google. We didn't back in the day. Now you have the answer to your frustration within quick reach. Use it to your advantage.

1.9 Vectors

Now that you are a bit more familiar with the idea of an object, we can start talking about a particular type of objects, specifically we are going to discuss **vectors**.

What is a vector? A vector is simply a set of elements of *the same class* (typically these classes are: character, numeric, integer, or logical -as in True/False). Vectors are the basic data structure in R.

Typically you will use the `c()` function (c stands for concatenate) to create vectors. The code below exemplifies how to create vectors of different classes (numeric, logical, etc.). Notice how the listed elements (to simplify there are two elements in each vector below) are separated by commas:

```
my_1st_vector <- c(0.5, 0.6) #creates a numeric vector with two elements
my_2nd_vector <- c(1L, 2L) #creates an integer vector
my_3rd_vector <- c(TRUE, FALSE) #creates a logical vector
my_4th_vector <- c(T, F) #creates a logical vector using abbreviations of True and False
my_5th_vector <- c("a", "b", "c") #creates a character vector
my_6th_vector <- c(1+0i, 2+4i) #creates a complex vector (we won't really use this class)
```

Cut and paste this code into your script and run it. You will see how all these vectors are added to your global environment and stored there.

The beauty of an object oriented statistical language like R is that once you have these objects you can use them as **inputs** in functions, use them in operations, or to create other objects. This makes R very flexible. See some examples below:

```
class(my_1st_vector) #a function to figure out the class of the vector

## [1] "numeric"
```

```

length(my_1st_vector) #a function to figure out the length of the vector

## [1] 2

my_1st_vector + 2 #Add a constant to each element of the vector

## [1] 2.5 2.6

my_7th_vector <- my_1st_vector + 1 #Create a new vector that contains the elements of my1stvector
my_1st_vector + my_7th_vector #Adds the two vectors and auto-print the results (note how the sum

## [1] 2.0 2.2

```

As indicated earlier, when you create objects you will place them in your working memory or workspace. Each R session will be associated to a workspace (called “global environment” in R Studio). In R Studio you can visualise the objects you have created during a session in the **Global Environment** screen. But if you want to produce a list of what’s there you can use the `ls()` function (the results you get may differ from the ones below depending on what you actually have in your global environment).

```

ls() #list all objects in your global environment

## [1] "my_1st_vector" "my_2nd_vector" "my_3rd_vector" "my_4th_vector"
## [5] "my_5th_vector" "my_6th_vector" "my_7th_vector" "my_text"
## [9] "x"

```

If you want to delete a particular object you can do so using the `rm()` function.

```
rm(x) #remove x from your global environment
```

It is also possible to remove all objects at once:

```
rm(list = ls()) #remove all objects from your global environment
```

If you mix in a vector elements that are of a different class (for example numerical and logical), R will **coerce** to the minimum common denominator, so that every element in the vector is of the same class. So, for example, if you input a number and a character, it will coerce the vector to be a character vector -see the example below and notice the use of the `class()` function to identify the class of an object.

```
my_8th_vector <- c(0.5, "a")
class(my_8th_vector) #The class() function will tell us the class of the vector

## [1] "character"
```

1.10 On comments

In the bits of code above you will have noticed parts that were grayed out. See for example in the last example provided. You can see that after the hash-tag all the text is being grayed out. What is this? What's going on?

These are **comments**. Comments are simply annotations that R will know is not code (and therefore doesn't attempt to understand and execute). We use the hash-tag symbol to specify to R that what comes after is not programming code, but simply bits of notes that we write to remind ourselves what the code is actually doing. Including these comments will help you to understand your code when you come back to it.

To create a comment you use the hash-tag/ number sign `#` followed by some text. Whenever the R engine sees the number sign it knows that what follows is not code to be executed. You can use this sign to include *annotations* when you are coding. These annotations are a helpful reminder to yourself (and others reading your code) of **what** the code is doing and (even more important) **why** you are doing it.

It is good practice to often use annotations. You can use these annotations in your code to explain your reasoning and to create “scannable” headings in your code. That way after you save your script you will be able to share it with others or return to it at a later point and understand what you were doing when you first created it -see here for further details on annotations and in how to save a script when working with the basic R interface.

Just keep in mind:

- You need one `#` per line, and anything after that is a comment that is not executed by R.
- You can use spaces after (its not like a hash-tag on twitter).

1.11 Factors

An important thing to understand in R is that categorical (ordered, also called ordinal, or unordered, also called nominal) data are typically encoded as **factors**, which are just a special type of vector. A factor is simply an integer

vector that can contain *only predefined values* (this bit is very important), and is used to store categorical data. Factors are treated specially by many data analytic and visualisation functions. This makes sense because they are essentially different from quantitative variables.

Although you can use numbers to represent categories, *using factors with labels is better than using integers to represent categories* because factors are self-describing (having a variable that has values “Male” and “Female” is better than a variable that has values “1” and “2” to represent male and female). When R reads data in other formats (e.g., comma separated), by default it will automatically convert all character variables into factors. If you rather keep these variables as simple character vectors you need to explicitly ask R to do so. We will come back to this next week with some examples.

Factors can also be created with the `factor()` function concatenating a series of *character* elements. You will notice that is printed differently from a simply character vector and that it tells us the levels of the factor (look at the second printed line).

```
the_smiths <- factor(c("Morrisey", "Marr", "Rourke", "Joyce")) #create a new factor
the_smiths #auto-print the factor

## [1] Morrisey Marr      Rourke    Joyce
## Levels: Joyce Marr Morrisey Rourke

#Alternatively for similar result using the as.factor() function
the_smiths_bis <- c("Morrisey", "Marr", "Rourke", "Joyce") #create a character vector
the_smiths_f <- as.factor(the_smiths_bis) #create a factor using a character vector
the_smiths_f #auto-print factor

## [1] Morrisey Marr      Rourke    Joyce
## Levels: Joyce Marr Morrisey Rourke
```

Factors in R can be seen as vectors with a bit more information added. This extra information consists of a record of the distinct values in that vector, called **levels**. If you want to know the levels in a given factor you can use the `levels()` function:

```
levels(the_smiths)

## [1] "Joyce"     "Marr"      "Morrisey"   "Rourke"
```

Notice that the levels appear printed by alphabetical order. There will be situations when this is not the most convenient order. Later on we will discuss in these tutorials how to reorder your factor levels when you need to.

1.12 Naming conventions for objects in R

You may have noticed the various names I have used to designate objects (`my_1st_vector`, `the_smiths`, etc.). You can use almost any names you want for your objects. Objects in R can have names of any length consisting of letters, numbers, underscores (“`_`”) or the period (“`.`”) and should begin with a letter. In addition, when naming objects you need to remember:

- *Some names are forbidden.* These include words such as `FALSE` and `TRUE`, logical operators, and programming words like `Inf`, `for`, `else`, `break`, `function`, and words for special entities like `NA` and `NAN`.
- *You want to use names that do not correspond to a specific function.* We have seen, for example, that there is a function called `print()`, you don’t want to call an object “`print`” to avoid conflicts. To avoid this use nouns instead of verbs for naming your variables and data.
- *You don’t want them to be too long* (or you will regret it every time you need to use that object in your analysis: your fingers will bleed from typing).
- *You want to make them as intuitive to interpret as possible.*
- *You want to follow consistent naming conventions.* R users are terrible about this. But we could make it better if we all aim to follow similar conventions. In these handouts you will see I follow the `underscore_separated` convention -see here for details.

It is also important to remember that R will always treat numbers as numbers. This sounds straightforward, but actually it is important to note. We can name our variables almost anything. EXCEPT they cannot be numbers. Numbers are **protected** by R. `1` will always mean `1`.

If you want, give it a try. Try to create a variable called `12` and assign it the value “`twelve`”. As we did last week, we can assign something a meaning by using the “`<-`” characters.

```
12 <- "twelve"
```

```
## Error in 12 <- "twelve": invalid (do_set) left-hand side to assignment
```

You get an error!

1.13 Dataframes

Ok, so now that you understand some of the basic types of objects you can use in R, let's start talking about data frames. One of the most common objects you will work with in this course are **data frames**. Data frames can be created with the `data.frame()` function.

Data frames are *multiple vectors* of possibly different classes (e.g., numeric, factors), but of the same length (e.g., all vectors, or variables, have the same number of rows). This may sound a bit too technical but it is simply a way of saying that a data frame is what in other programmes for data analysis gets represented as data sets, the tabular spreadsheets you have seen when using Excel.

Let's create a data frame with two variables:

```
#We create a data frame called mydata_1 with two variables, an integer vector called foo and a logical vector called bar
mydata_1 <- data.frame(foo = 1:4, bar = c(T,T,F,F))
mydata_1
```

```
##   foo   bar
## 1  1  TRUE
## 2  2  TRUE
## 3  3 FALSE
## 4  4 FALSE
```

Or alternatively for the same result:

```
x <- 1:4
y <- c(T, T, F, F)
mydata_2 <- data.frame (foo = x, bar = y)
mydata_2
```

```
##   foo   bar
## 1  1  TRUE
## 2  2  TRUE
## 3  3 FALSE
## 4  4 FALSE
```

As you can see in R, as in any other language, there are multiple ways of saying the same thing. Programmers aim to produce code that has been optimised: it is short and quick. It is likely that as you develop your R skills you find increasingly more efficient ways of asking R how to do things. What this means too is that when you go for help, from your peers or us, we may teach you

slightly different ways of getting the right result. As long as you get the right result that's what at this point really matters.

These are silly toy examples of data frames. In this course, we will use real data. Next week we will learn in greater detail how to read data into R. But you should also know that R comes with pre-installed data sets. Some packages in fact are nothing but collections of data frames.

Let's have a look at some of them. We are going to look at some data that are part of the *fivethirtyeight* package. This package contains data sets and code behind the stories in this particular online newspaper. This package is not part of the base installation of R, so you will need to install it first. I won't give you the code for it. See if you can figure it out by looking at previous examples.

Done? Ok, now we are going to look at the data sets that are included in this package. Remember first we have to load the package if we want to use it:

```
library("fivethirtyeight")
data(package="fivethirtyeight") #This function will return all the data frames that ar
```

Notice that this package has some data sets that relate to stories covered in this journal that had a criminological angle. Let's look for example at the *hate_crimes* data set. How do you that? First we have to load the data frame into our global environment. To do so use the following code:

```
data("hate_crimes")
```

This function will search among all the *loaded* packages and locate the *hate_crimes* data set. Notice that it now appears in the global environment, although it also says "promise" next to it. To see the data in full you need to do something to it first. So let's do that.

Every object in R can have **attributes**. These are: names; dimensions (for matrices and arrays: number of rows and columns) and dimensions names; class of object (numeric, character, etc.); length (for a vector this will be the number of elements in the vector); and other user-defined. You can access the attributes of an object using the **attributes()** function. Let's query R for the attributes of this data frame.

```
attributes(hate_crimes)
```

```
## $row.names
## [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [51] 51
##
```

```
## $class
## [1] "tbl_df"      "tbl"        "data.frame"
##
## $names
##  [1] "state"                  "state_abbrev"
##  [3] "median_house_inc"       "share_unemp_seas"
##  [5] "share_pop_metro"        "share_pop_hs"
##  [7] "share_non_citizen"      "share_white_poverty"
##  [9] "gini_index"              "share_non_white"
## [11] "share_vote_trump"       "hate_crimes_per_100k_splic"
## [13] "avg_hatecrimes_per_100k_fbi"
```

These results printed in the may console may not make too much sense to you at this point. We will return to this next week, so do not worry.

Go now to the global environment panel and left click on the data frame hate_crimes. This will open the data viewer in the top left section of R Studio. What you get there is a spreadsheet with 12 variables and 51 observations. Each variable in this case is providing you with information (demographics, voting patterns, and hate crime) about each of the US states.

The screenshot shows the RStudio interface. In the Global Environment panel, the 'hate_crimes' data frame is listed with a count of 51 observations and 12 variables. A red arrow points to the 'hate_crimes' entry with the text 'First, left click here'. In the Data Viewer pane, the 'hate_crimes' data frame is displayed as a spreadsheet with 12 columns (variables) and 51 rows (observations). A red arrow points to the spreadsheet with the text 'You will then get the spreadsheet view'. The bottom right corner of the viewer pane features the R logo.

1.14 Exploring data

Ok, let's now have a quick look at the data. There are so many different ways of producing summary stats for data stored in R that is impossible to cover them all! We will just introduce a few functions that you may find useful for summarising data. Before we do any of that it is important you get a sense for what is available in this data set. Go to the help tab and in the search box

input the name of the data frame, this will take you to the documentation for this data frame. Here you can see a list of the available variables.

The raw data behind the story "Higher Rates Of Hate Crimes Are Tied To Income Inequality"
<https://fivethirtyeight.com/features/higher-rates-of-hate-crimes-are-tied-to-income-inequality/>.

Usage

```
hate_crimes
```

Format

A data frame with 51 rows representing US states and DC and 12 variables:

- state
- median_house_inc
- share_unemp_seas
- share_pop_metro

Let's start with the *mean*. This function takes as an argument the numeric variable for which you want to obtain the mean. Because of the way that R works you cannot simply put the name of the variable you have to tell R as well in which data frame is that variable located. To do that you write the name of the data frame, the dollar sign, and then the name of the variable you want to summarise. If you want to obtain the mean of the variable that gives us the proportion of people that voted for Donald Trump you can use the following expression:

```
mean(hate_crimes$share_vote_trump)
```

```
## [1] 0.49
```

Another function you may want to use with numeric variables is **summary()**:

```
summary(hate_crimes$share_vote_trump)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.040	0.415	0.490	0.490	0.575	0.700

This gives you the five number summary (minimum, first quartile, median, third quartile, and maximum, plus the mean and the count of missing values if there are any).

You don't have to specify a variable you can ask for these summaries from the whole data frame:

```
summary(hate_crimes)
```

```
##      state           state_abbrev      median_house_inc share_unemp_seas
##  Length:51          Length:51        Min.   :35521     Min.   :0.02800
##  Class :character   Class :character  1st Qu.:48657     1st Qu.:0.04200
##  Mode  :character   Mode  :character Median  :54916     Median :0.05100
##                                         Mean   :55224     Mean   :0.04957
##                                         3rd Qu.:60719     3rd Qu.:0.05750
##                                         Max.  :76165     Max.  :0.07300
##
##      share_pop_metro    share_pop_hs    share_non_citizen share_white_poverty
##  Min.   :0.3100      Min.   :0.7990     Min.   :0.01000   Min.   :0.04000
##  1st Qu.:0.6300     1st Qu.:0.8405     1st Qu.:0.03000   1st Qu.:0.07500
##  Median :0.7900      Median :0.8740     Median :0.04500   Median :0.09000
##  Mean   :0.7502      Mean   :0.8691     Mean   :0.05458   Mean   :0.09176
##  3rd Qu.:0.8950     3rd Qu.:0.8980     3rd Qu.:0.08000   3rd Qu.:0.10000
##  Max.   :1.0000      Max.   :0.9180     Max.   :0.13000   Max.   :0.17000
##                                         NA's   :3
##      gini_index      share_non_white share_vote_trump hate_crimes_per_100k_splic
##  Min.   :0.4190      Min.   :0.0600     Min.   :0.040     Min.   :0.06745
##  1st Qu.:0.4400     1st Qu.:0.1950     1st Qu.:0.415     1st Qu.:0.14271
##  Median :0.4540      Median :0.2800     Median :0.490     Median :0.22620
##  Mean   :0.4538      Mean   :0.3157     Mean   :0.490     Mean   :0.30409
##  3rd Qu.:0.4665     3rd Qu.:0.4200     3rd Qu.:0.575     3rd Qu.:0.35694
##  Max.   :0.5320      Max.   :0.8100     Max.   :0.700     Max.   :1.52230
##                                         NA's   :4
##      avg_hatecrimes_per_100k_fbi
##  Min.   : 0.2669
##  1st Qu.: 1.2931
##  Median : 1.9871
##  Mean   : 2.3676
##  3rd Qu.: 3.1843
##  Max.   :10.9535
##  NA's   :1
```

There are multiple ways of getting results in R. Particularly for basic and intermediate-level statistical analysis many core functions and packages can give you the answer that you are looking for. For example, there are a variety

of packages that allow you to look at summary statistics using functions defined within those packages. You will need to install these packages before you can use them.

I am only going to introduce one of them here *skimr*. It is neat and is maintained by one of my former stats teachers, the criminologist Elin Waring. You will need to install it before anything else. Use the code you have learnt to do so and then load it. I won't be providing you the code for it, by now you should now how to do this.

Once you have loaded the *skimr* package you can use it. Its main function is *skim*. Like *summary* for data frames, *skim* presents results for all the columns and the statistics will depend on the class of the variable. However, the results are displayed and stored in a nicer way -though we won't get into the details of this right now.

```
skim(hate_crimes)
```

```
## Creating new skimming functions for the following classes: hist.
## They did not have recognized defaults. Call get_default_skimmers() for more information

function (data, ...) { data_name <- rlang::expr_label(substitute(data)) if
(!is.data.frame(data)) { data <- as.data.frame(data) } stopifnot(is.data.frame(data))
.vars <- rlang::quos(...) cols <- names(data) if (length(.vars) == 0) { selected
<- cols } else { selected <- tidyselect::vars_select(cols, !!!.vars) } grps <-
dplyr::groups(data) if (length(grps) > 0) { group_variables <- selected
%in% as.character(grps) selected <- selected[!group_variables] } skimmers
<- purrr::map(selected, get_final_skimmers, data, local_skimmers, append)
types <- purrr::map_chr(skimmers, "skim_type") unique_skimmers <- re-
duce_skimmers(skimmers, types) combined_skimmers <- purrr::map(unique_skimmers,
join_with_base, base) ready_to_skim <- tibble::tibble(skim_type =
unique(types), skimmers = purrr::map(combined_skimmers, mangle_names,
names(base$fun)), skim_variable = split(selected, types)[unique(types)]$grouped <-
dplyr :: group_by(ready_to_skim, .dataskim_type) nested <- dplyr::summarize(grouped,
skimmed = purrr::map2(.dataskimmers, .dataskim_variable, skim_by_type,
data)) structure(tidyr::unnest(nested, .dataskimmed), class = c("skim_df", "tbl_df", "tbl",
"data.frame")) nrow(data), data_cols = ncol(data), df_name = data_name, groups = dplyr :: groups(data),
base_skimmers = names(base$fun), skimmers_used = get_skimmers_used(unique_skimmers)) } <bytecode: 0x7fd9ea590398>
<environment: 0x7fd9ea594d48>

## Creating new skimming functions for the following classes: hist.
## They did not have recognized defaults. Call get_default_skimmers() for more information

function (data, ...) { data_name <- rlang::expr_label(substitute(data)) if
(!is.data.frame(data)) { data <- as.data.frame(data) } stopifnot(is.data.frame(data))
```

```
.vars <- rlang::quos(...) cols <- names(data) if (length(.vars) == 0) { selected
<- cols } else { selected <- tidyselect::vars_select(cols, !!!.vars) } grps <-
dplyr::groups(data) if (length(grps) > 0) { group_variables <- selected
%in% as.character(grps) selected <- selected[!group_variables] } skimmers
<- purrr::map(selected, get_final_skimmers, data, local_skimmers, append)
types <- purrr::map_chr(skimmers, "skim_type") unique_skimmers <- re-
duce_skimmers(skimmers, types) combined_skimmers <- purrr::map(unique_skimmers,
join_with_base, base) ready_to_skim <- tibble::tibble(skim_type =
unique(types), skimmers = purrr::map(combined_skimmers, mangle_names,
names(basefun)), skim_variable = split(selected, types)[unique(types)])grouped <
-dplyr :: group_by(ready_to_skim, .data$skim_type) nested <- dplyr::summarize(grouped,
skimmed = purrr::map2(.data$skimmers, .data$skim_variable, skim_by_type,
data)) structure(tidy::unnest(nested, .data$skimmed), class = c("skim_df", "tbl_df", "tbl", "data.frame")), data$ows =
nrow(data), data$ols = ncol(data), df_name = data$name, groups = dplyr ::
groups(data), base_skimmers = names(basefun), skimmers_used =
get_skimmers_used(unique_skimmers)) } <bytecode: 0x7fd9ea590398>
<environment: 0x7fd9ea77d448>
```

skim_type	skim_variable	n_missing	complete_rate	character.min	character.max	char
character	state	0	1.0000000	4	20	
character	state_abbrev	0	1.0000000	2	2	
numeric	median_house_inc	0	1.0000000	NA	NA	
numeric	share_unemp_seas	0	1.0000000	NA	NA	
numeric	share_pop_metro	0	1.0000000	NA	NA	
numeric	share_pop_hs	0	1.0000000	NA	NA	
numeric	share_non_citizen	3	0.9411765	NA	NA	
numeric	share_white_poverty	0	1.0000000	NA	NA	
numeric	gini_index	0	1.0000000	NA	NA	
numeric	share_non_white	0	1.0000000	NA	NA	
numeric	share_vote_trump	0	1.0000000	NA	NA	
numeric	hate_crimes_per_100k_splic	4	0.9215686	NA	NA	
numeric	avg_hatecrimes_per_100k_fbi	1	0.9803922	NA	NA	

Apart from summary statistics, last semester we discussed a variety of ways to graphically display variables. In week 3 we covered scatterplots, a graphical device to show the relationship between two quantitative variables. I don't know if you remember the amount of point and click you had to do in Excel for getting this done. If not you can review the notes here.

There's also many different ways of producing graphics in R. In this course we rely on a package called *ggplot2*. It is already in the clusters, but if you are using your own laptop will need to install it first and then load it.

```
library(ggplot2)
```

Then we will use one of its functions to create a scatterplot. Don't worry about understanding this code below, we will have a whole session on the ggplot

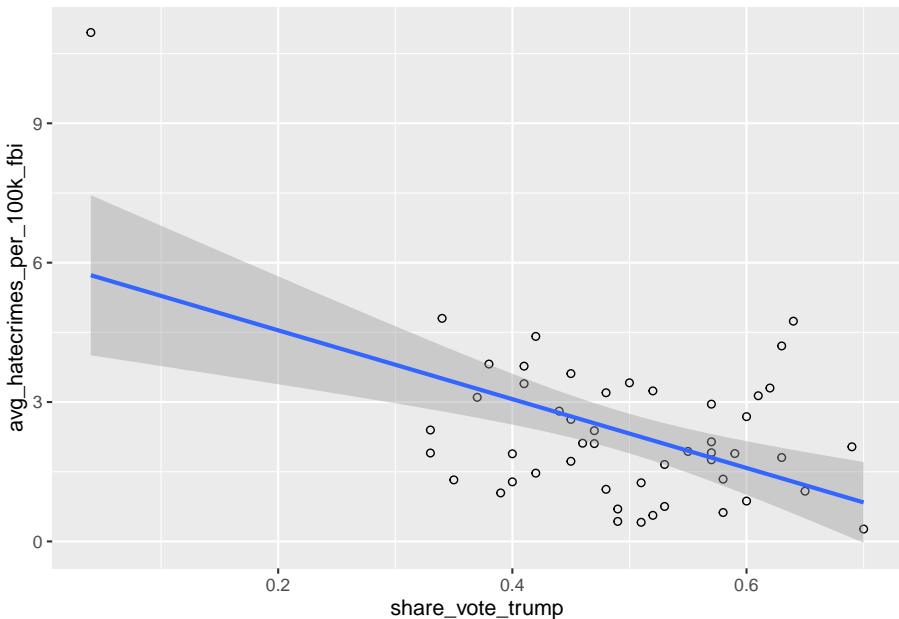
function:

```
ggplot(hate_crimes, aes(x=share_vote_trump, y=avg_hatecrimes_per_100k_fbi)) +
  geom_point(shape=1) +
  geom_smooth(method=lm)

## `geom_smooth()` using formula 'y ~ x'

## Warning: Removed 1 rows containing non-finite values (stat_smooth).

## Warning: Removed 1 rows containing missing values (geom_point).
```



What do you think this graphic is telling you?

1.15 Quitting RStudio

At some point, you will quit your R/R Studio session. I know, hard to visualise, right? Why would you want to do that? Anyhow, when that happens R Studio will ask you a hard question: “Save work space image to bla bla bla/.RData?” What to do? What does that even mean?

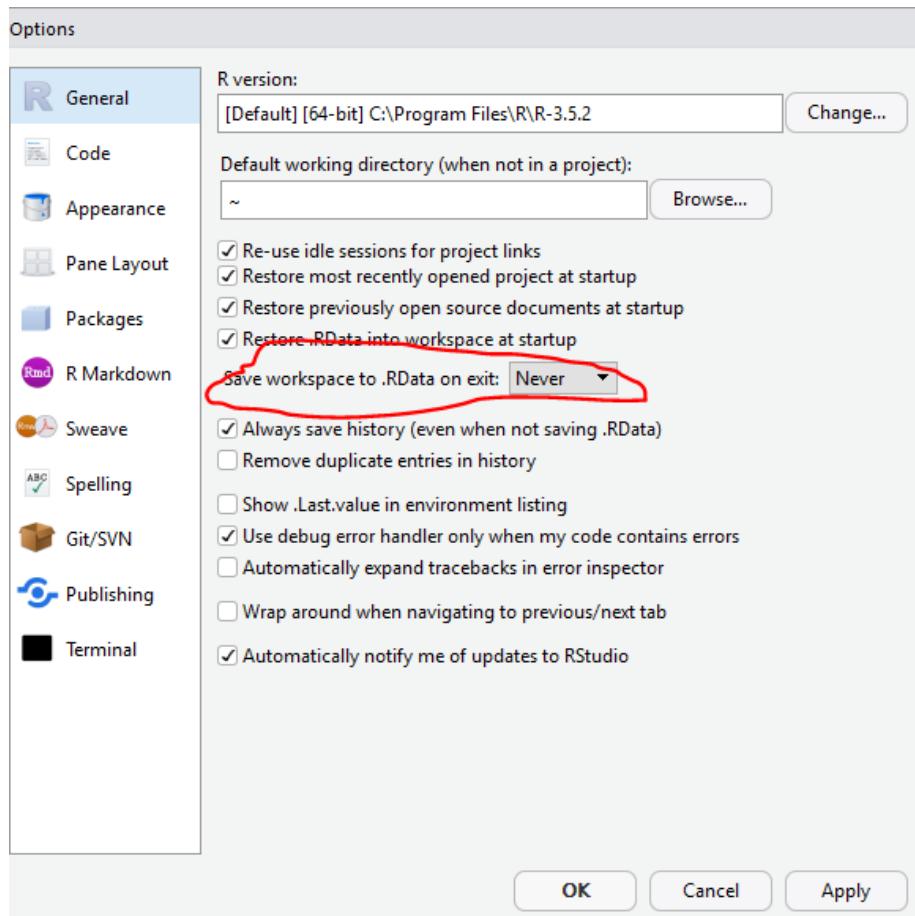
If you say “yes” what will happen is that all the objects you have in your environment will be preserved, alongside the *History* (which you can access in

the top right set of windows) listing all the functions you have run within your session. So, next time you open this project all will be there. If you think that what is *real* is those objects and that history, well then you may think that's what you want to do.

Truth is what is real is your scripts and the data that your scripts use as inputs. You don't need anything that is in your environment, because you can recreate those things by re-running your scripts. I like keeping things tidy, so when I am asked whether I want to save the image, my answer is always no. Most long time users of R never save the workspace, nor care about saving the history either. Remember what is real is your scripts and the data.

Keep in mind though that you should not then panic if you open your next R Studio session and you don't see any objects in your environment. The good news is you can generate them quickly enough (if you really need them) by re-running your scripts. I would suggest that at this point it may be helpful for you to get into this habit as well. I suspect otherwise you will be in week 9 of the semester and have an environment full of garbage you don't really need.

What is more. I would suggest you go to the Tools drop down menu, select Global Options, and make sure you select "Never" where it says "Save workspace". Then click "Apply". This way you will never be asked to save what is in your global environment when you terminate a session.



Chapter 2

Making your first maps in R

2.1 A quick introduction of terms

2.1.1 Geospatial Perspective - The Basics

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

2.1.1.1 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.

2.1.1.2 Attributes

Attribute has become the preferred term for any recorded characteristic or property of a place. 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 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 data analysis courses. 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.

2.1.1.3 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
- 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.

POINTS: Individual **x, y** locations.

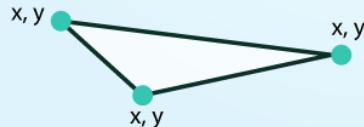
ex: Center point of plot locations, tower locations, sampling locations.

**LINES:** Composed of many (at least 2) vertices, or points, that are connected.

ex: Roads and streams.

**POLYGONS:** 3 or more vertices that are connected and **closed**.

ex: Building boundaries and lakes.



neon

Objects can also be Raster data. Raster data is made up of pixels (or cells), and each pixel 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) cells.

2.1.1.4 Maps

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.

2.1.1.5 Map projections

Map projections try to portray the surface of the earth or a portion of the earth on a flat piece of paper or computer screen. A coordinate reference system (CRS) then defines, with the help of coordinates, how the two-dimensional, projected map in your GIS is related to real places on the earth. 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.

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 see 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 labelled X, and the vertical axis is normally labelled Y.

Working 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,

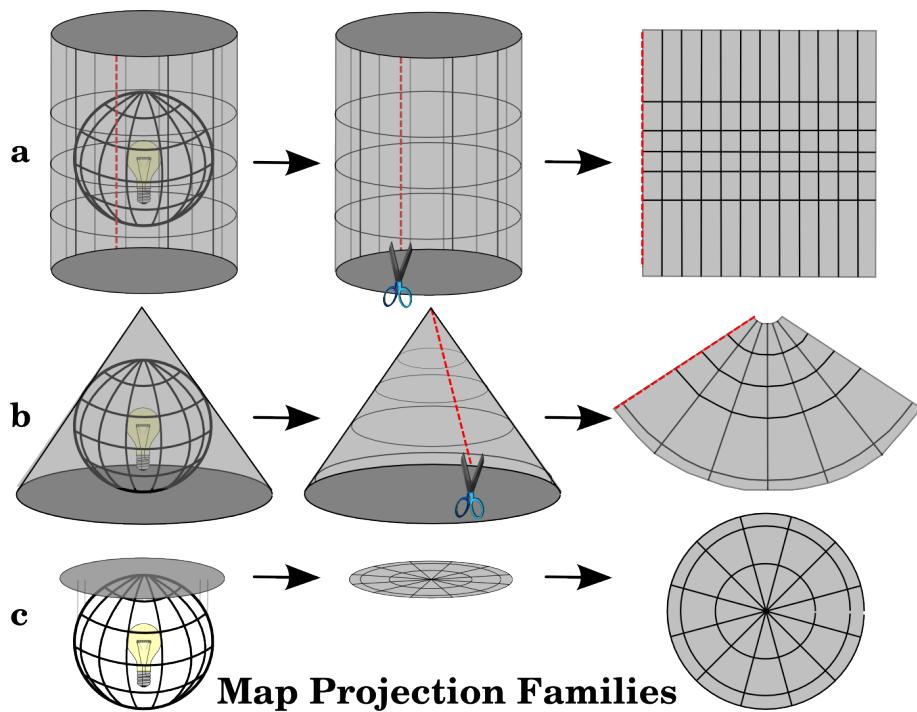


Figure 2.1: figure_projection_families

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

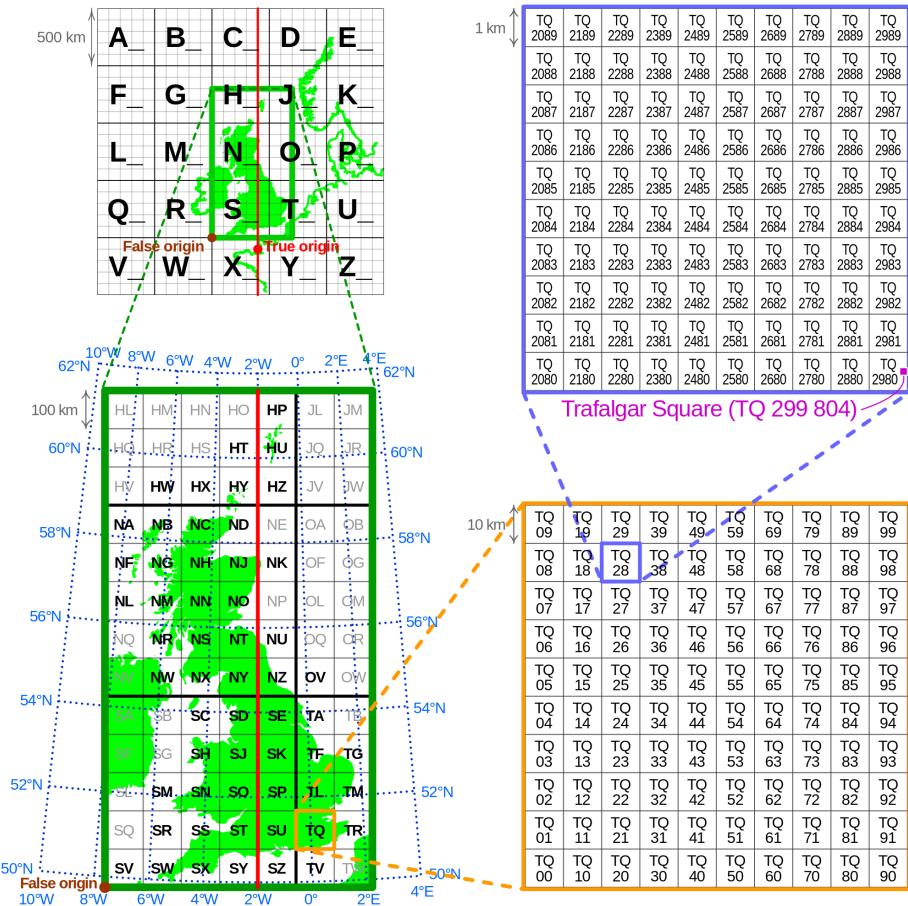


Figure 2.2: BNG

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”.

2.1.1.6 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. You can have a look at the lesson on network analysis in the QGIS documentation

2.1.1.7 Density estimation

One of the more useful concepts in spatial analysis is density - the density of humans in a crowded city, or the density of retail stores in a shopping centre. Mathematically, the density of some kind of object is calculated by counting the number of such objects in an area, and dividing by the size of the area. To read more about this, I recommend Silverman, Bernard W. Density estimation for statistics and data analysis. Vol. 26. CRC press, 1986.

2.1.2 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

2.2 Getting some spatial data to put on a map

2.2.1 A first look at basemaps

Maps of the kind we will cover in this course are simply a form of data visualisation. In previous courses you may have learnt about histograms, scatterplots, and other forms of representing data in a two dimensional space. R is pretty good for producing data visualisation and there are three big approaches to producing this within R, which are rooted to particular packages. The oldest one is what people refer to as base R. The oldest R configuration has loads of plotting capabilities and follows a very particular philosophy about how to produce

graphs. More modern packages are `lattice`, for multivariate data visualisation, and `ggplot2`, which relies in a theoretical model called the grammar of graphics.

In the same way, there are many different packages that can be used to produce maps, some of which rely on the functionality provided by base R and others that rely on `ggplot2` or other external graphical packages. In this course we will play around with several of these R packages to produce maps. Many offer similar functionality, but they all have certain special advantages (and disadvantages). So, in practice you may shift among them depending on what it is that you want to achieve.

Leaflet is one of the most popular open-source JavaScript libraries for interactive maps. It's 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*. And lucky for us, there is an R package which makes it easy to integrate and control Leaflet maps in R. It allows to easily produce maps with contextual information from static maps such as GoogleMaps, OpenStreet Maps, or Stamen maps.

We are going to start now with some code. So, it would be a good idea for you to make sure you have your RStudio project open and ready to go. As usual, first you will need to install this package in your machine, something you should know by now. Then load the package using the code below.

```
library(leaflet)
```

You create a Leaflet map with these basic steps:

- Create a map widget by calling `leaflet()`.
- Add layers (i.e., features) to the map by using layer functions (e.g. `addTiles`, `addMarkers`, `addPolygons`) to modify the map widget.
- Repeat step 2 as desired.
- Print the map widget to display it.

Let's start with the first and last points there, before discussing step 2 (and 3). It's possible to create a map without any arguments at all. You can do this by calling the `leaflet()` function, and adding some background tiles with the `addTiles()` function. Like so:

```
m <- leaflet() %>%
  addTiles() # Add default OpenStreetMap map tiles

m # Print the map
```

And while yepp, you've made a map, it's not a particularly exciting map.

As the most important characteristic of any map is location, the most important argument of our code is the location argument. You can do this with the `setView()` function. Here you can specify location. Ideally, location is a longitude/latitude pair specifying the center of the map. Here we can choose the longitude and latitude for University of Manchester. If you wonder how we know these, there are many tools (such as Google Maps) for looking up latitude and longitude.

You could also accompany the coordinates with a zoom argument, an integer from 3 to 20 specifying how large the spatial extent should be around the center, with 3 being the continent level and 20 being roughly the single building level. Note that many maps do not support that high zoom levels. Find out by modifying the example below, what the maximum zoom level of the default tiles is.

```
m <- leaflet() %>%
  addTiles() %>% # Add default OpenStreetMap map tiles
  setView(lng=-2.233885, lat=53.466852, zoom = 15)
m # Print the map, zoom = 15
```

So what you see above 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.

In the lecture today we described *reference maps*. We often may want to use these reference maps as **basemaps** for our thematic maps. They may give us context and help with the interpretation. But what we want to learn in this course is about thematic maps, maps that tell stories and for that we need data. We will look at cool data in the next section, but before just a couple of things about these basemaps. You can see above that you are seeing the *Open Street Map* Basemap. This is one option but there are others.

For example stamen maps are one alternative. If you wanted to use this, you have to specify what sort of tile you add in the `addProviderTiles()` function. Notice that it's different to `addTiles()` because you're getting tiles from an external provider, in this case Stamen. So to get stamen we would pass the argument `providers$Stamen.Toner`. Like so:

```
m <- leaflet() %>%
  addProviderTiles(providers$Stamen.Toner) %>% # Add Stamen tile map tiles
  setView(lng=-2.233885, lat=53.466852, zoom = 15)
m # Print the map, stamen toner
```

Stamen also has other maps, watercolour for example:

```
m <- leaflet() %>%
  addProviderTiles(providers$Stamen.Watercolor) %>% # Add default Stamen Watercolor map
  setView(lng=-2.233885, lat=53.466852, zoom = 15)
m # Print the map
```

You can use other sources such as carto db

```
m <- leaflet() %>%
  addProviderTiles(providers$CartoDB) %>% # Add CartoDB map tiles
  setView(lng=-2.233885, lat=53.466852, zoom = 15)
m # Print the map
```

You can use the `View()` function to see all the possible options for the providers to see a list of the types of basemaps you could use with leaflet. Have a go at choosing some of your own!

```
View(providers)
```

2.2.2 Find some relevant data to show: obtaining data on crime

We can play around with police recorded crime data, which can be downloaded from the police.uk website.

Let's download some data for crime in Manchester.

To do this, open the data.police.uk/data website.

- In **Date range** just select one month of data. Choose whatever month you like. I will choose November 2017, so if you want to see the same results as will be here, pick that month.
- In **Force** find **Greater Manchester Police**, and tick the box next to it.
- In **Data sets** tick **Include crime data**.
- Finally click on **Generate File** button.

This will 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. Navigate to the project directory folder you've created and save it there. Unzip the file.

```
#You can use the unzip function for this. A cool function you may want to use as well is file.choose()
unzip(file.choose())
```

If you look at the *Files* window in the bottom right corner of RStudio you should see now a new subdirectory that contains a .csv file with the data that we need. Since I downloaded the data from November 2017 in my case this subdirectory is called 2017-11.

Before we can use this data we need to read it or import it into R and turn it into a dataframe object. To read in the .csv file, which is the format we just downloaded, the command is `read.csv()`.

Again there are two ways to read in the data, if you want to open a window where you can manually navigate and open the file, you can pass `file.choose()` argument to the `read.csv()` function as illustrated earlier.

```
#This code creates a dataframe object called crimes which will include the spreadsheet in the fi
crimes <- read.csv(file.choose())
```

Or, if you know the path to your file, you can hard-code it in there, within quotation marks:

```
crimes <- read.csv("data/2017-11-greater-manchester-street.csv")
```

You might notice that `crimes` has appeared in your work environment window. It will tell you how many observations (rows - and incidentally the number of recorded crimes in November 2017 within the GMP jurisdiction) and how many variables (columns) your data has.

Let's have a look at the `crimes` dataframe:

```
#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 required geographical coordinates that we need to plot this data.

```
library(tibble)
glimpse(crimes)

## #> #> #> #> #> #> #>
```

```
## #> #> #> #> #> #> #>
```

```
## #> #> #> #> #> #> #>
```

```
## Rows: 34,052
## Columns: 12
## #> $ Crime.ID      <fct> , f892dce3e7a4c45fe4f8f09f24d6a494f2b49783a97...
## #> $ Month          <fct> 2017-11, 2017-11, 2017-11, 2017-11, ...
```

```

## $ Reported.by          <fct> Greater Manchester Police, Greater Manchester...
## $ Falls.within         <fct> Greater Manchester Police, Greater Manchester...
## $ Longitude            <dbl> -2.462774, -2.462774, -2.464422, -...
## $ Latitude             <dbl> 53.62210, 53.62210, 53.62210, 53.61250, 53.61...
## $ Location              <fct> On or near Scout Road, On or near Scout Road, ...
## $ LSOA.code             <fct> E01012628, E01012628, E01012628, E01004768, E...
## $ LSOA.name             <fct> Blackburn with Darwen 018D, Blackburn with Da...
## $ Crime.type            <fct> Anti-social behaviour, Criminal damage and ar...
## $ Last.outcome.category <fct> , Investigation complete; no suspect identifi...
## $ Context               <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...

```

2.2.3 Add a point layer to your basemap

In GIS settings you may have multiple vector data that you want to represent in the same map simultaneously. So creating maps is often a function of adding new layers to an existing map. Before we saw how to generate a basemap with leaflet. Now we are going to add a new layer to that basemap with information about the location of crimes (as indexed in our crime dataframe).

To do that we will use the `addMarkers()` function. Within the `addMarkers()` command we need to identify where the information is coming from, so we need to pass an argument identifying the dataframe object, and we need to identify how the aesthetics (in short, aes) are going to be defined (in this case, we need to identify the variables that define the location of the points in a two dimensional plane).

```

m <- leaflet() %>%
  addProviderTiles(providers$Stamen.Toner) %>%
  setView(lng=-2.233885, lat=53.466852, zoom = 15) %>%
  addCircleMarkers(lng=crimes$Longitude, lat=crimes$Latitude)
m # Print the map

```

The crime appear as blue dots.

Keep in mind these are not exact locations, for privacy reasons the data in police.UK add noise to longitude and latitude so that it is harder to identify individuals based on this data. But this gives you an approximate idea of where crime takes place in and around the university area.

Let's get more detailed information even. When you glimpse at the data, you may have noticed that one of the attributes in the dataframe was type of crime, as indexed by the `Crime.type` variable. We can look at the frequency distribution of this variable using the `table()` function.

```
table(crimes$Crime.type)
```

```
##          Anti-social behaviour      Bicycle theft
##                5417                      356
##          Burglary      Criminal damage and arson
##                2774                      3500
##          Drugs           Other crime
##                591                       612
##          Other theft      Possession of weapons
##                2087                      236
##          Public order            Robbery
##                4223                      647
##          Shoplifting      Theft from the person
##                1427                      751
##          Vehicle crime Violence and sexual offences
##                2786                      8645
```

This is interesting, but keep in mind that this is data for all types of crimes in Greater Manchester. What if we wanted to differentiate between different types of crime. But instead, we can ask R to use the information from Crime.type to use a different colour for each type of crime. For this, we first need to make an object that creates a list of colours matched to each crime type. For this we use the `colorFactor()` function, and give it two arguments, one is the `topo.colors` list, which in brackets we put the number of categories we'll need. That's all the possible values that the Crime.type variable can take. To see what this is we can use the `unique()` function:

```
unique(crimes$Crime.type)
```

```
## [1] Anti-social behaviour      Criminal damage and arson
## [3] Violence and sexual offences Other theft
## [5] Other crime              Vehicle crime
## [7] Burglary                  Public order
## [9] Bicycle theft             Possession of weapons
## [11] Robbery                  Shoplifting
## [13] Drugs                     Theft from the person
## 14 Levels: Anti-social behaviour Bicycle theft ... Violence and sexual offences
```

We can see that there are 14 possible values, so we will need 14 colours. The second argument is the variable, in this case Crime.type. Like so:

```
factpal <- colorFactor(topo.colors(14), crimes$Crime.type)
```

Now that we've created this object, we can use it to specify crime type through colour. We add a `color=` argument to the `addCircleMarkers()` function:

```
m <- leaflet(crimes) %>%
  addProviderTiles(providers$Stamen.Toner) %>%
  setView(lng=-2.233885, lat=53.466852, zoom = 15) %>%
  addCircleMarkers(lng=crimes$Longitude, lat=crimes$Latitude, color = ~factpal(Crime.type))
m # Print the map
```

But which colour means which crime type?! To know, we should add a legend. We can do this with the helpfully named `addLegend()` function.

```
m <- leaflet(crimes) %>%
  addProviderTiles(providers$Stamen.Toner) %>%
  setView(lng=-2.233885, lat=53.466852, zoom = 15) %>%
  addCircleMarkers(lng=crimes$Longitude, lat=crimes$Latitude, color = ~factpal(Crime.type),
  addLegend("bottomright", pal = factpal, values = ~Crime.type,
  title = "Crime types")
m # Print the map
```

It's not the nicest map, but it's our first one, so that's exciting! We'll refine it more later on.

2.2.4 HOMEWORK 1

Think about this visualisation. 1. How could you characterise the basemap: is it a vector map or a raster image? 2. How could you characterise the layer representing the crimes: is it a vector map or a raster image? 3. Is the resulting visualisation clear? Is it helpful? If you could change anything, what would it be? Write your thoughts up.

This is going to be your first assignment to be submitted as part of next week. Export the image file into a Word document and write your answers there.

2.3 From dataframes to spatial objects: finding shapefiles

You can get a long way with spatial data stored in data frames, but it makes life easier if they are stored in special spatial objects. In the previous exercise we

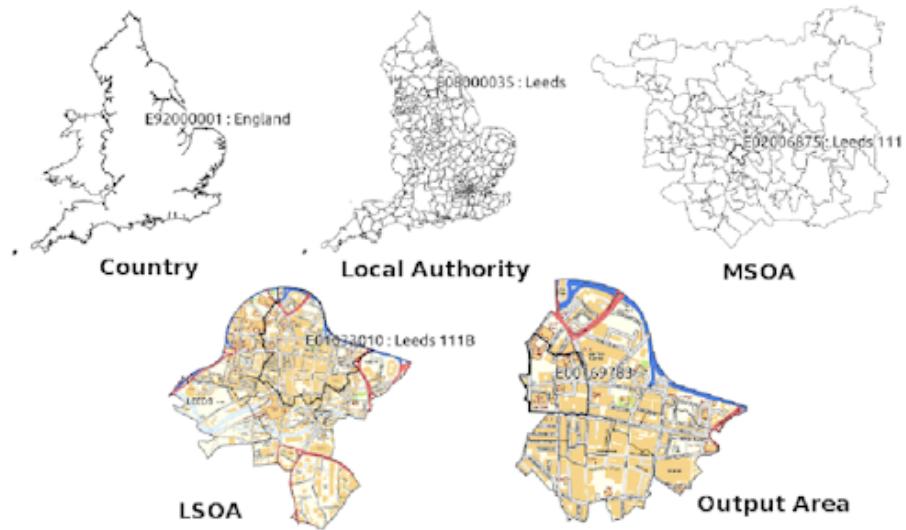
2.3. FROM DATAFRAMES TO SPATIAL OBJECTS: FINDING SHAPEFILES

saw how we can easily display point patterns in ggmap just using data extracted from a dataframe with a longitude and latitude columns. In many instances, when you work with GIS you rely on spatial objects that are a bit more complex in structure and that are stored in a wide variety of proprietary and open source formats.

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 developed by ESRI, the developers and vendors of 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 [here](#).

We are going to learn here how to obtain shapefiles for British census geographies. In the class today we talked about the idea of neighbourhoods and we explained how a good deal of sociological and criminological work traditionally used census geographies as proxies for neighbourhoods. As of today, they still are the geographical subdivisions for which we can obtain a greater number of attribute information (e.g., sociodemographics, etc.).

You can read more about census boundary data [here](#). “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 goes from Country to Local Authority to Middle Layer Super Output Area (MSOA) to Lower Layer Super Output Area (LSOA) to Output Area:



Here we will get some boundaries for Manchester. Let's use the LSOA level. 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.

So to get some boundary data, you can use the UK Data Service website. There is a simple Boundary Data Selector.

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 you are feeling unsure at any point, feel free to click on that help to guide you.

For now, let's focus on the selector options. Here you can choose the country you want to select shapefiles for. We select "England". You can also choose the type of geography we want to use. Here we select "Statistical Building Block", as discussed above. And finally you can select when you want it for. If you are working with historical data, it makes sense to find boundaries that match the timescale for your data. Here we will be dealing with contemporary data, and therefore we want to be able to use the newest available boundary data.

2.3. FROM DATAFRAMES TO SPATIAL OBJECTS: FINDING SHAPEFILES55

The screenshot shows the 'Boundary Data Selector' interface. At the top, there are two tabs: 'What and Where?' (selected) and 'Format'. Below the tabs, a message reads: 'For help with using the Boundary Data Selector interface, choose one of the following links: How to use Boundary Data Selector link'. Under the heading 'Select:', there are three dropdown menus: 'England' (selected), 'Statistical Building Block' (selected), and '2011 and later'. To the right of these dropdowns is a blue 'Find' button.

Once you have selected these options, click on the “Find” button. That will populate the box below:

The screenshot shows the 'Boundary Data Selector' interface after clicking the 'Find' button. The 'Boundaries' section is now populated with a list of options: 'English Census Output Areas, 2011' (selected), 'English Lower Layer Super Output Areas, 2011' (highlighted in grey), and 'English Middle Layer Super Output Areas, 2011'. Below this list is a note: 'Area(s) of interest: English Districts, UAs and London Boroughs, 2011 (Derived by aggregating 2011 Output Areas using exact-fit OA to higher geography lookup table)'. At the bottom right of the list area is a blue 'List Areas' button.

Here you can select the boundaries we want. As discussed, we want the census lower super output areas. But again, your future choices here will depend on what data you want to be mapping.

Once you've made your choice, click on “List Areas”. This will now populate the box below. We are here concerned with Manchester. However you can select more than one if you want boundaries for more than one area as well. Just hold down “ctrl” to select multiple areas individually, or the shift key to select everything in between.

The screenshot shows a list of geographical areas: Liverpool, Luton, Maidstone, Maldon, Malvern Hills, Manchester, Mansfield, and Medway. The area 'Manchester' is highlighted with a blue background. To the right of the list are three buttons: 'List Areas', 'Expand Selection', and 'Previous List'. Below the list is a 'Search Summary' table with three rows: 'Search Summary' (Target Geography: English Census Output Areas with OAC, 2011; Selected Area(s): Manchester), 'Data Format' (SHAPE), and 'Archive Method' (Zip). At the bottom right is a large blue button labeled 'Extract Boundary Data'.

Search Summary:	[Target Geography : English Census Output Areas with OAC, 2011] [Selected Area(s) : Manchester]
Data Format:	SHAPE
Archive Method:	Zip

Once you've made your decision click on the "Extract Boundary Data" button. You will see the following message:

The screenshot displays a progress message: "The geospatial dataset you requested through the UK Data Service is being prepared for you but it may take some time before it is ready to download. You will be automatically redirected to our data download bookmarking facility if after 3 minutes your data download is still being prepared for you. To go to the bookmarking facility now click on the Go To Bookmark button below." A blue button labeled "Go To Bookmark" is visible at the bottom of the message box.

You can bookmark, or just stay on the page and wait. How long you have to wait will depend on how much data you have requested to download.

When your data is read, you will see the following message:

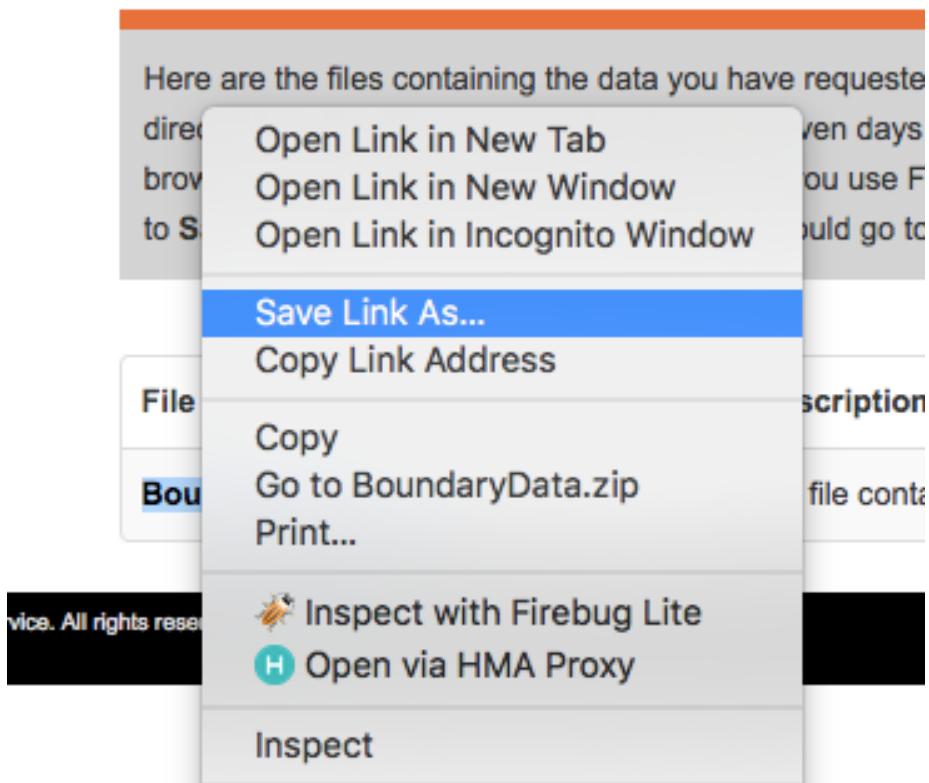
2.3. FROM DATAFRAMES TO SPATIAL OBJECTS: FINDING SHAPFILES57

FILES EXTRACTED

Here are the files containing the data you have requested from Census Support, together with their type and size. The files in this directory will be automatically deleted within seven days of being generated. You can download an individual file using your web browser by clicking on its name in the table. If you use Firefox, the best way to do this is to click with the right mouse button and go to Save Link As. With Internet Explorer you should go to Save Target As after clicking with the right mouse button.

File Name	Description	Size
BoundaryData.zip	Zip file containing your data	3 MB

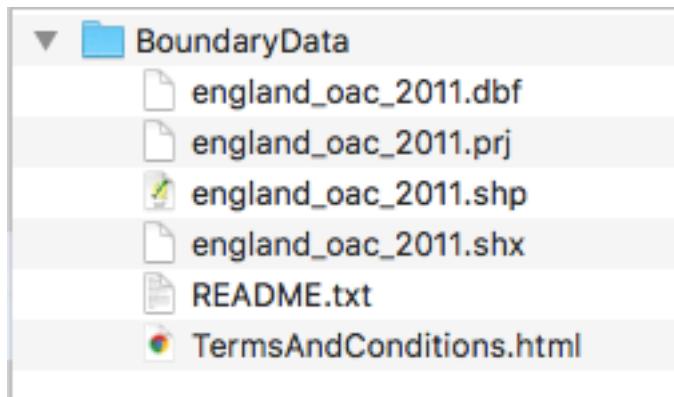
You have to right click on the “BoundaryData.zip”, and hit Save Target as on a PC or Save Link As on a Mac:



Navigate to the folder you have created for this analysis, and save the .zip file there. Extract the file contents using whatever you like to use to unzip compressed files.

```
#For example,
unzip("BoundaryData.zip", exdir="BoundaryData")
```

You should end up with a folder called “BoundaryData”. Have a look at its contents:



So you can see immediately that there are some documentations around the usage of this shapefile, in the readme and the terms and conditions. Have a look at these as they will 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.

But then you will also notice that there are 4 files with the same name “england_oac_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 spatial data, 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. Please do remember the suggestions we provided last week as to how to organise your RStudio project directories.

2.4 Reading shapefiles into R: the wonderful world of sf objects

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 rely on `sp`, making it an important part of the R ecosystem. More recently a new package, `sf` (which stands for “simple features”), is revolutionising the way that R does spatial analysis. This new 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, so that those packages that still don’t use this new format can be used. But in this course 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 this book.

2.4.1 HOMEWORK 2

Read Section 2.1 of the Geocomputation book linked above. Answer the following questions: 1. What are some strengths/advantages of the `sf` package? 2. What code do you need to transform a `sf` object into a `sp` object. 3 What is simply a `sf` object?

Install `sf` if you don’t already have. Then load it.

```
library(sf)  
  
## Linking to GEOS 3.8.1, GDAL 2.4.4, PROJ 4.9.1
```

On Mac and Linux a few requirements must be met to install sf. These are described in the package's README at github.com/r-spatial/sf.

To read in your data, you will need to know the path to where you have saved it. Ideally this will be in your working 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 `st_read()` function to read in the shapefile:

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

```
## Reading layer `england_lsoa_2011' from data source `/Users/reka/Dropbox (The University of Manchester)  
## Simple feature collection with 282 features and 3 fields  
## geometry type: POLYGON  
## dimension: XY  
## bbox: xmin: 378833.2 ymin: 382620.6 xmax: 390350.2 ymax: 405357.1  
## CRS: 27700
```

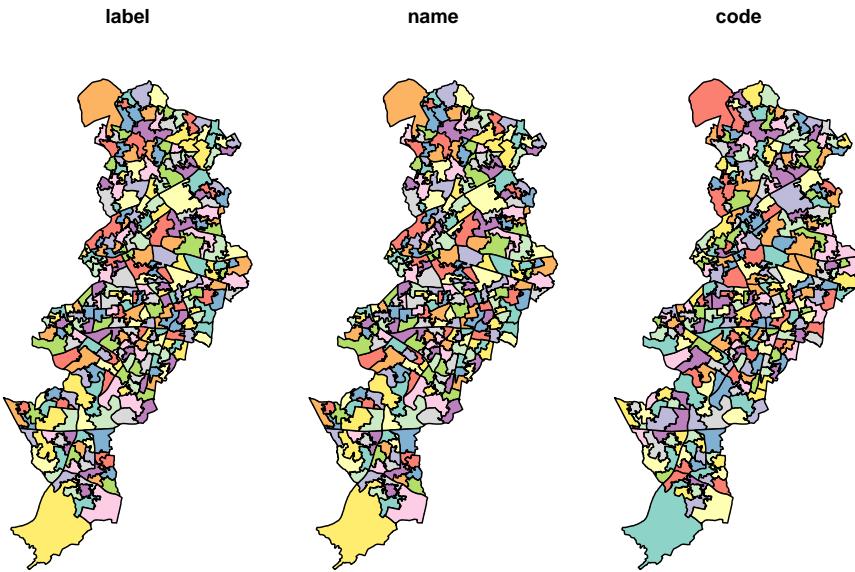
Now you have your spatial data file. You can have a look at what sort of data it contains, the same way you would view a dataframe, with the `View()` function:

```
View(manchester_lsoa)
```

```
## Rows: 282  
## Columns: 4  
## $ label      <fct> E08000003E02001062E01005066, E08000003E02001092E01005073, ...  
## $ name       <fct> Manchester 018E, Manchester 048C, Manchester 018A, Manches...  
## $ code       <fct> E01005066, E01005073, E01005061, E01005062, E01005063, E01...  
## $ geometry   <POLYGON [m]> POLYGON ((384850 397432, 38..., POLYGON ((382221.1...)
```

And of course, since it's spatial data, you can finally map it:

```
plot(manchester_lsoa)
```



This is the main way that we will be creating maps. OK so you see that three maps appeared - any ideas why? Do you know what the three maps correspond to? Discuss.

2.5 Add some data

2.5.1 Data wrangling with dplyr

Now let's get some crime data to add to this map. We can do this by using the police.uk data we obtained earlier. Have a look again at the information stored in the crimes spreadsheet:

```
glimpse(crimes)
```

```
## Rows: 34,052
## Columns: 12
## $ Crime.ID          <fct> , f892dce3e7a4c45fe4f8f09f24d6a494f2b49783a97...
## $ Month              <fct> 2017-11, 2017-11, 2017-11, 2017-11, 2017-11, ...
## $ Reported.by        <fct> Greater Manchester Police, Greater Manchester...
## $ Falls.within        <fct> Greater Manchester Police, Greater Manchester...
```

```

## $ Longitude          <dbl> -2.462774, -2.462774, -2.464422, -...
## $ Latitude           <dbl> 53.62210, 53.62210, 53.62210, 53.61250, 53.61...
## $ Location            <fct> On or near Scout Road, On or near Scout Road, ...
## $ LSOA.code           <fct> E01012628, E01012628, E01012628, E01004768, E...
## $ LSOA.name            <fct> Blackburn with Darwen 018D, Blackburn with Da...
## $ Crime.type           <fct> Anti-social behaviour, Criminal damage and ar...
## $ Last.outcome.category <fct> , Investigation complete; no suspect identifi...
## $ Context              <lgl> NA, N...

```

You should be able to see that there is a variable, a column in this spreadsheet, called LSOA.code. Yep, that is the unique identifier that is telling us in which lower super output area each crime took place. If only we could use this information to create a new dataset counting the number of criminal events that took place within each of these areas!!!

Ok, here is where you are introduced to the wonderful world of **dplyr**. 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 this tutorial), but we are going to cover three different very useful elements of dplyr: the select function, the group_by function, and the piping operator.

Load the library:

```

library(dplyr)

## 
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
## 
##     filter, lag

## The following objects are masked from 'package:base':
## 
##     intersect, setdiff, setequal, union

```

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 data frame and store it in a new object we could write the following code:

```
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 object
grouped_crimes <- group_by(new_object, LSOA.code)
#Then we could count the number of cases within each category and use the summarise function to p
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 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 column
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.

2.6 Join data to spatial object

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.

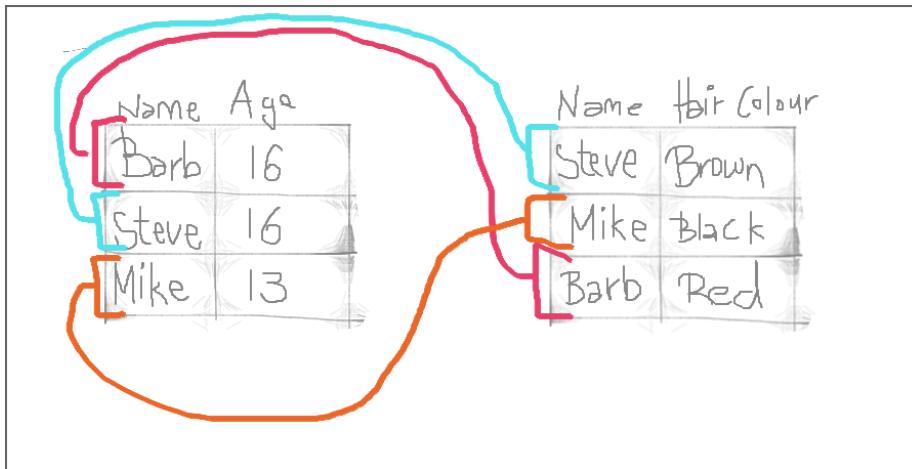
Merging rows is pretty straightforward, but merging columns can be tricky if the rows of one table don't always line up with the rows in the other table. By using `left_join()` from the `dplyr` package, you can avoid some of the alignment problems.

`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:



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.

2.7 So left_join(), eh?

Well actually there is a whole family of join functions as part of dplyr. But here we use left_join, because that way we keep all the rows in x (the left-hand side datafram), and join to it all the matched columns in y (the right-hand side datafram).

So let's join the crimes data to the spatial data, using left_join():

We have to tell left_join 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:

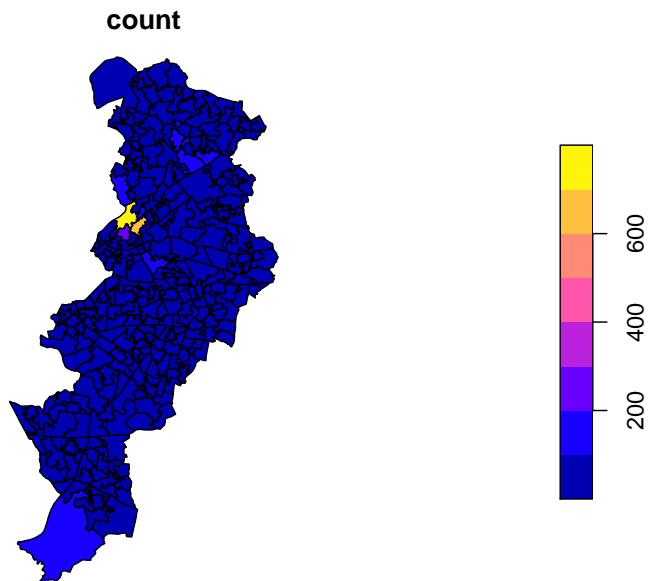
```
manchester_lsoa <- left_join(manchester_lsoa, crimes_per_lsoa, by = c("code"="LSOA.code"))

## Warning: Column `code`/`LSOA.code` joining factors with different levels,
## coercing to character vector
```

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 can now use this to create a thematic choropleth map

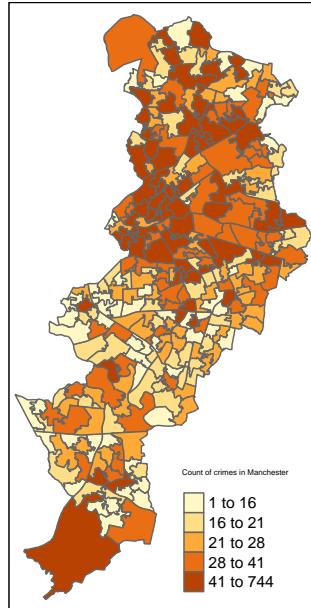
```
plot(manchester_lsoa[4])
```



2.8 Let's make this prettier: a quick glance at tmap

Very quickly, but just to illustrate things can be prettier, we are going to use this data with another package `tmap`, short for thematic maps. This package also borrows from the `ggplot` syntax and is specifically designed to make creation of thematic maps more convenient. It takes care of a lot of the styling and aesthetics. This reduces our amount of code significantly. So, look at what we can do with our previous map:

```
library(tmap)
tm_shape(manchester_lsoa) +
  tm_polygons("count", style="quantile", title="Count of crimes in Manchester")
```



And we can even add some interactivity!

```
tmap_mode("view")
## tmap mode set to interactive viewing
tmap_last()
```

So, this is all for today, next week we will come back to `tmap` and explain the different arguments that we use when thematic maps. This was just an

introduction to some of the things we can do. Next week we will spend a bit of more time discussing how to make good choices when producing maps.

Chapter 3

Thematic maps in R

3.1 Intro and recap

Last week we showed you fairly quickly how to create maps of spatial point patterns using `leaflet` and we also introduced the `tmap` package for thematic maps. Besides doing that we introduced a set of key concepts we hope you have continued studying over the week. We also discussed the `sf` package for storing spatial objects in R.

This week we will carry on where we left the session last week. In the presentations last week we introduced various kind of thematic maps and in our lecture this week we discuss in detail issues with choropleth maps. So the focus of today's lab is going to be around thematic maps and some of the choices we discussed in our presentation last week and also this week.

We will also introduce faceting and **small multiples**, which is a format for comparing the geographical distribution of different social phenomena. For this session we will be using the spatial object that you created last week and complement it with additional information from the census. So first of all you will have to rerun the code you used to create the `manchester_lsoa sf` object. Apart from doing so, you want to start your session loading the libraries you know for sure you will need:

```
library(sf)
library(tmap)
library(dplyr)
```

You may not remember all of what you did to generate that file so let's not waste time and just cut and paste from below (but try to remember what each of the lines of code is doing and if you are not clear look at the notes from last

week). Imagine you had to do all of this again by pointing and clicking in a graphical user interface rather than just sending the code to the console! As you will see time and time again, code in the end is a much more efficient way of talking to a computer.

```
crimes <- read.csv("https://raw.githubusercontent.com/maczkni/2018_labs/master/data/2011_crime.csv")
#The following assumes you have a subdirectory called BoundaryData in your working directory
shp_name <- "data/BoundaryData/england_lsoa_2011.shp"
manchester_lsoa <- st_read(shp_name)

## Reading layer `england_lsoa_2011` from data source `/Users/reka/Dropbox (The University of Manchester) - Google Drive`
## Simple feature collection with 282 features and 3 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:            xmin: 378833.2 ymin: 382620.6 xmax: 390350.2 ymax: 405357.1
## CRS:             27700

crimes_per_lsoa <- crimes %>%
  select(LSOA.code) %>%
  group_by(LSOA.code) %>%
  summarise(count=n())
manchester_lsoa <- left_join(manchester_lsoa, crimes_per_lsoa, by = c("code"="LSOA.code"))

## Warning: Column `code`/`LSOA.code` joining factors with different levels,
## coercing to character vector
```

You may not want to have to go through this process all the time. 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:

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

Notice how four files have appeared in your working directory, in your “BoundaryData” subdirectory or whatever you called it. Remember what we said last week about shapefiles, there are a collection of files that need to be kept together.

If you wanted to bring this shapefile back into R at any future point, you would only need to use the `st_read()` function.

Before we carry on, can you tell What is different between `manchester_lsoa.shp` and `manchester_crime_lsoa.shp`? Think about it.

3.2 Creating choropleth maps

The `tmap` 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. There are a number of vignettes in the CRAN repository and the GitHub repo for this package that you can explore. GitHub is a collaborative website used by software developers and data scientist, also contains a useful *readme* section with additional resources to familiarise yourself with this package. Each map can be plotted as a static map (*plot mode*) and shown interactively (*view mode*) as we briefly saw last week. We will start by focusing on static maps.

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. 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. 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`”.

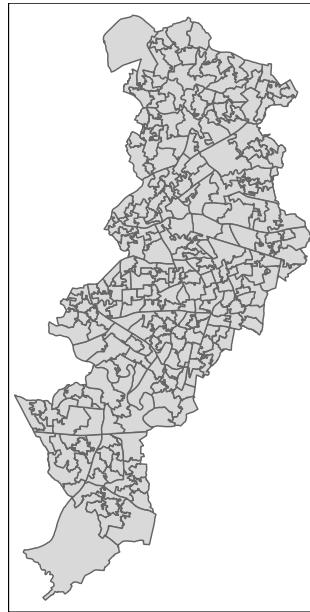
```
tm_shape(manchester_lsoa)
```

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. We will add a polygon using `tm_polygon()`. 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")
```

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

```
tm_shape(manchester_lsoa) +
  tm_polygons()
```



Given that we are not passing any additional arguments all we are getting is a map with the shape of the geographies that we are representing, the census LSOAs for Manchester city. We can, however, ask R to produce a choropleth map by mapping the values of a variable in our data table using colour. In tmap we need to denote our variables between quotes. The first argument we pass then would be the name of the variable we want to visualise. If you remember we have a count for crimes (“count”), so let’s visualise that by creating a thematic map.

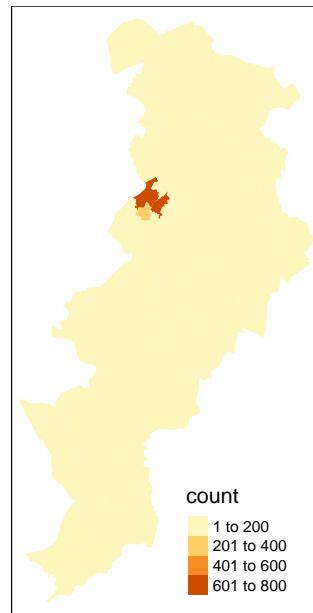
```
tm_shape(manchester_lsoa) +  
  tm_polygons("count")
```



Notice how this map is different from last week. What do you think the reason for this is? You may remember last week we used one additional argument `style` specifying the classification method we were going to use. If you remember we used quantiles. We will in a second look at how a map of the counts of crime looks different when we use different classification systems. But before we get to that, let's think about aesthetics a bit more.

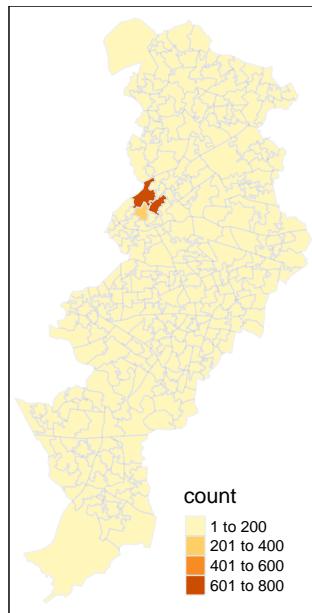
We have been using `tm_polygons()` but we can also add the elements of a polygon map using different functions that break down what we represent here. In the map above you see the polygons have a dual representation, the borders are represented by lines and the colour is mapped to the intensity of the quantitative variable we are representing. 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_lsoa) +
  tm_fill("count")
```



As you can see here, the look is a bit cleaner. 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_lsoa) +  
  tm_fill("count") +  
  tm_borders(alpha = 0.1)
```

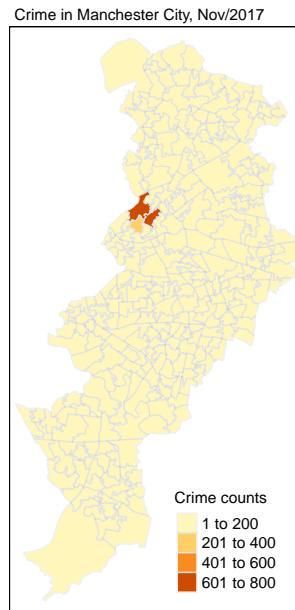


The alpha parameter that we are inserting within `tm_borders()` controls the transparency of the borders, we can go from 0 (totally transparent) to 1 (not transparent). You can play around with this value and see the results.

Notice in the last few maps we did not have to specify whether we wanted the map to be static or interactive. When you use `tmap`, R will remember the mode you want to use. So once you specify `tmap_mode("plot")`, all the subsequent maps will be static. It is only when you want to change this behaviour that you would need another `tmap_mode` call.

Notice as well that the legend in this map is (a) not very informative and (b) located in a place that is less than optimal, since it covers part of the map. 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_lsoa) +
  tm_fill("count", title = "Crime counts") +
  tm_borders(alpha = 0.1) +
  tm_layout(main.title = "Crime in Manchester City, Nov/2017", main.title.size = 0.7 ,
            legend.position = c("right", "bottom"), legend.title.size = 0.8)
```



```
#We are also going to change the current style of the maps by making them more friendly
current_style <- tmap_style("col_blind")

## tmap style set to "col_blind"

## other available styles are: "white", "gray", "natural", "cobalt", "albatross", "beaut...
```

3.3 Producing small multiples to compare the effect of different classification systems

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.

3.3. PRODUCING SMALL MULTIPLES TO COMPARE THE EFFECT OF DIFFERENT CLASSIFICATION SYSTEMS

Let's make four maps, each one using a different classification method: Equal interval, Natural breaks (Jenks), Quantile, and Unclassed (no classification). You should be familiar with these methods from the reading and lecturing material. If not, please, do make sure you read the material before continuing.

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

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

```
map1 <- tm_shape(manchester_lsoa) +          #use tm_shape function to specify spatial object
      tm_fill("count", style="equal", title = "Equal") + #use tm_fill to specify variable, classification method
      tm_layout(legend.position = c("right", "bottom"),    #use tm_layout to make the legend look nice
                legend.title.size = 0.8,
                legend.text.size = 0.5)
```

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

```
map2 <- tm_shape(manchester_lsoa) +
      tm_fill("count", style="jenks", title = "Jenks") +
      tm_layout(legend.position = c("right", "bottom"),
                legend.title.size = 0.8,
                legend.text.size = 0.5)
```

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

```
map3 <- tm_shape(manchester_lsoa) +
      tm_fill("count", style="quantile", title = "Quantile") +
      tm_layout(legend.position = c("right", "bottom"),
                legend.title.size = 0.8,
                legend.text.size = 0.5)
```

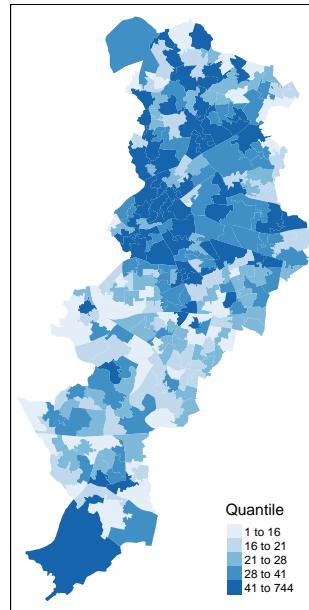
And finally make *map4*, an unclassed choropleth map, which maps the values of our variable to a smooth gradient.

```
map4 <- tm_shape(manchester_lsoa) +
      tm_fill("count", style="cont", title = "Unclassed") +
      tm_borders(alpha=0.1) +
      tm_layout(legend.position = c("right", "bottom"),
                legend.title.size = 0.8,
                legend.text.size = 0.5)
```

Notice that we are not plotting the maps, we are storing them into R objects (*map1* to *map4*). 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.

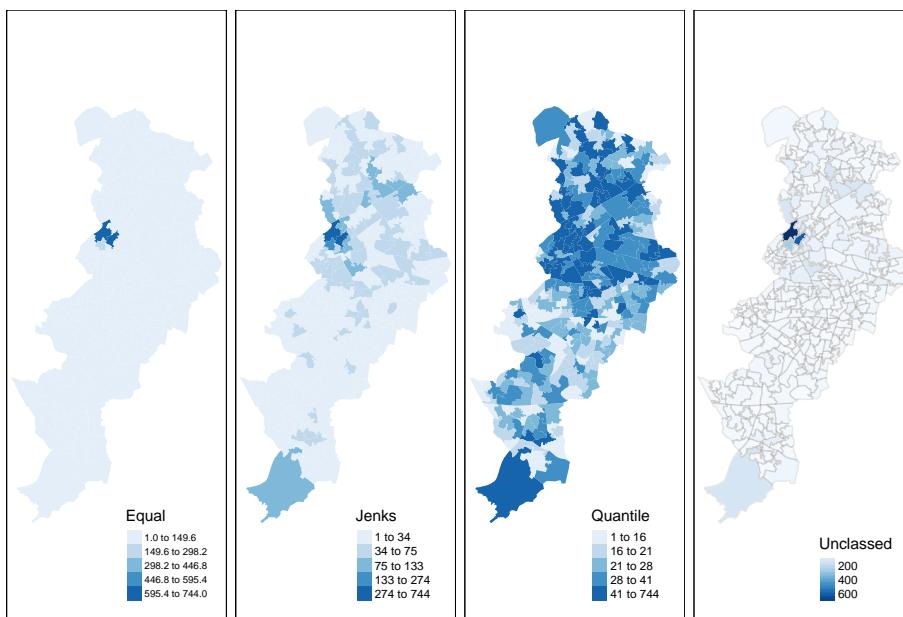
So if you wanted to map just *map3* for example, all you need to do, is call the *map3* object. Like so:

```
map3
```



But now we will plot all 4 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)
```



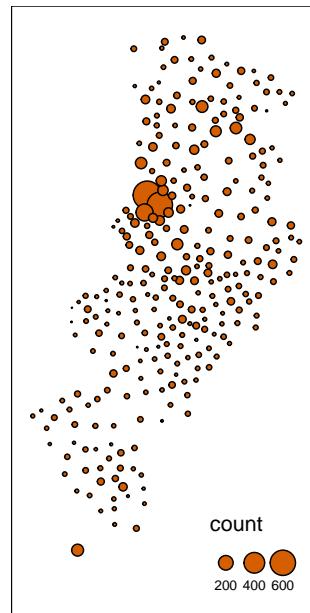
3.3.1 Homework 3.1

Discuss which of these classification methods gives you the best visualisation of crime in Manchester city. Insert the produced maps in your answer.

3.4 Using graduated symbols

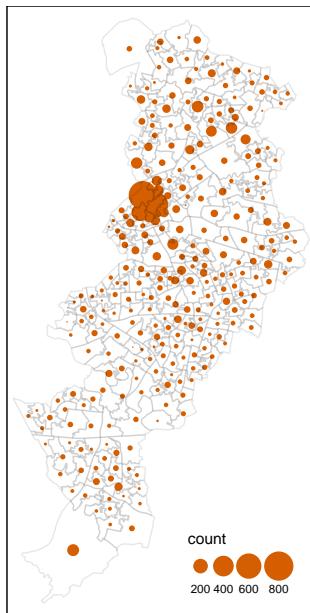
The literature on thematic cartography highlights how counts, like the ones above, are best represented using graduated symbols rather than choropleth maps. So let's try to go for a more appropriate representation. In `tmap` you can use `tm_symbols` for this. We will use `tm_borders` to provide some context.

```
tm_shape(manchester_lsoa) +
  tm_bubbles("count")
```



First thing you see is that we loose the context (provided by the polygon borders) that we had earlier. The `border.lwd` argument set to NA in the `tm_symbols()` 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 I am modifying the transparency of the borders with the `alpha` parameter.

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



3.5 Bringing additional census data in

Last week you learned how to obtain crime data from the police UK website and you also developed the skills to obtain shapefiles with the boundaries for the UK census geographies. Specifically you learnt how to obtain LSOAs boundaries. Then we taught you how to join these data tables using `dplyr`. If you open your `manchester_lsoa` object you will see that at the moment you only have one field in this data frame providing you with statistical information. However, there is a great deal of additional information that you could add to these data frame. Given that you are using census geographies you could add to it all kind of socio demographic variables available from the census.

You may want to watch this 4 minute video to get a sense for how to obtain the data. If you don't have headphones make sure you read this brief tutorial before carrying on. We are going to get some data for Manchester city LSOAs. Let me warn you though, the census data portal is one of the closest things to hell you are going to come across on the internet. Using it will be a good reminder of why point and click interfaces can suck the life out of you.

From the main Infuse portal select the 2011 census data then when queried pick selection by geography:

InFuse is a free service providing easy access to aggregate data from the UK 2011 and 2001 censuses

Access:

-
-

InFuse contains UK 2011 and 2001 English and Welsh Census aggregate data.

About the data

Expand the local authorities and select Manchester. Expand Manchester and select LSOAs:

- + Malvern Hills
- Manchester
 - + Wards and Electoral Divisions (32 areas)
 - + Middle Super Output Areas and Intermediate Zones (57 areas)
 - + Lower Super Output Areas and Data Zones (282 areas)
 - + Output Areas and Small Areas (1530 areas)
 - + Workplace Zone Layer (567 areas)
- + Mansfield
- + Medway
- + Melton

At the bottom of the page click in *Add* and then where it says *Next*. Now big tip. Do not press back in your browser. If you need to navigate back once you get to that point use the *previous* button at the bottom of the screen. You will regret it if you don't do this.

Now you will need to practice navigating the Infuse system to generate a data table that has a number of relevant fields we are going to use today and at a later point this semester. I want you to create a file with information about the resident population, the workday population, and the number of deprivation households. This will involve some trial and error but you should end up with a selection like the one below:

Help Guidance

Your category combination and area selections are summarised below. A reference for download files is suggested, but can be altered. If you are happy with your selections, click the 'Download' button to prepare your download files. When the files are ready, an orange 'Download' button will appear. Click on this to download your files.

Selected Category Combinations

- 1000 - Classification of household depr: Household is deprived in 3 dimensions - Unit: Households
- 1001 - Classification of household depr: Household is deprived in 4 dimensions - Unit: Households
- 2301 - Unit - Persons in the resident population: All usual residents
- 2333 - Unit - Persons in the resident population: Working population - Sex: Total Sex - Unit: Persons
- 990 - Classification of household depr: Total: Classification of household deprivation - Unit: Households
- 991 - Classification of household depr: Household is deprived in 1 dimension - Unit: Households
- 992 - Classification of household depr: Household is deprived in 2 dimensions - Unit: Households
- 993 - Classification of household depr: Household is deprived in 3 dimensions - Unit: Households

Selected Areas

- All Lower Super Output Areas and Data Zones (282 areas) in Local Authorities (324 areas) - Manchester

File reference: AGE_APPSOCG_DAYPOP_UNIT_URESPOP

Once you have those fields click next to get the data and download the file. Unzip them and open the .csv file in Excel. If you view the data in Excel you

will notice it is a bit untidy. The first row has no data but the labels for the variable names and the second row has the data for Manchester as a whole. We don't need those rows. Because this data is a bit untidy we are going to use `read_csv()` function from the `readr` package rather than the base `read.csv` function.

```
library(readr)
census_lsoa_m <- read_csv("https://www.dropbox.com/s/e4nkqmefovlsvib/Data_AGE_APPSOCG_DAYPOP_UNIT

## Warning: Missing column names filled in: 'X14' [14]
```

Notice that even all the variables that begin with “f” are numbers they have been read into R as characters. This is to do with the fact the first two lines do not represent cases and do have characters. R is coercing everything into character vectors. Let's clean this a bit.

First we will get rid of the first two rows. In particular we will use the `slice()` function from `dplyr`. We can use slice to select cases based on row number. We don't need the first two rows so we can select rows 3 to 284.

```
census_lsoa_m <- slice(census_lsoa_m, 3:284)
```

There are also fields that we don't need. We only need the variables beginning with F for those have the information about population and deprivation, and the *GEO_CODE* tag which will allow us to link this table to the *manchester_lsoa* file.

```
census_lsoa_m <- select(census_lsoa_m, GEO_CODE, F996:F323339)
```

We also want to convert the character variables into numeric ones, whilst preserving the *id* as a character variable. For this we will use the `lapply` function. This is a convenient function that will administer a function to the elements we pass as an argument. In this case we are asking to apply the `as.numeric()` function to the columns 2 to 9 of the *census_lsoa_m* data frame. This is turning into numeric all those character columns.

```
census_lsoa_m[2:9] <- lapply(census_lsoa_m[2:9], as.numeric)
```

The only problem we have now is that the variable names are not very informative. If you look at the metadata file that came along you can see that there is a key there to understand what these variables mean. We could use that information to create more meaningful names for the variables we have. We will use the `rename()` function from the `dplyr` package to do the renaming:

```
census_lsoa_m <- rename(census_lsoa_m, tothouse = F996, notdepr = F997, depriv1 = F998,
                         depriv2 = F999, depriv3 = F1000, depriv4 = F1001, respop = F238,
                         wkdpop = F323339)
```

The rename function takes as the first argument the name of the dataframe. Then for each variable you want to change you write down the new name followed by the old name. Now that we have the file ready we can link it to our *manchester_lsoa* file using code we learnt last week. We use again the `left_join()` function to add to the *manchester_lsoa* dataframe the variables that are present in the *census_lsoa_m*. The first argument in the function is the name of the dataframe to which we want to add fields, the second argument the name of the dataframe from which those fields come, and then you need to specify using “by” the name of the variables on each of these two dataframes that have the id variable that will allow us to ensure that we are linking the information across the same observations.

```
manchester_lsoa <- left_join(manchester_lsoa, census_lsoa_m, by = c("code"="GEO_CODE"))
```

And there you go... Now you have a datafile with quite a few pieces of additional information about LSOAs in Manchester. The next step is to use this information.

3.6 Computing and mapping crime rates

Ok, so now we have a field that provides us with the number of crimes and two alternative counts of population for each LSOA in Manchester in the same dataframe. We could compute the rate of crime in each using the population counts as our denominator. Let’s see how the maps may compare using these different denominators.

But 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_lsoa* 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.

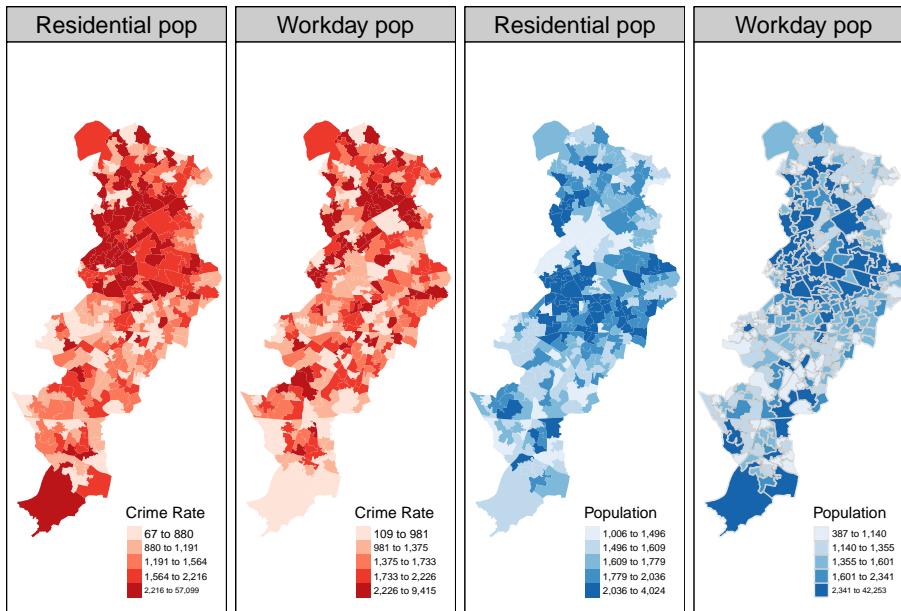
```
manchester_lsoa <- mutate(manchester_lsoa, crimr1 = (count/respop)*100000, crimr2 = (count/wkdpop)
```

It should not be difficult for you to produce now a choropleth map like the one below. Clue: to change the colors for the fill of the polygons you can use the palette argument within the `tm_fill` calls. You can explore different palettes running the following code:

```
tmaptools:::palette_explorer()
```

3.6.1 Homework 2

Reproduce the map below, you will need to include the code you used as part of your homework submission. Discuss the results. What are the most notable differences? Which denominator do you think is more appropriate (you will need to think about this quite carefully). Are you comparing like with like? Why? Why not? Could you make these comparisons more equivalent if you think you are not comparing like with like?

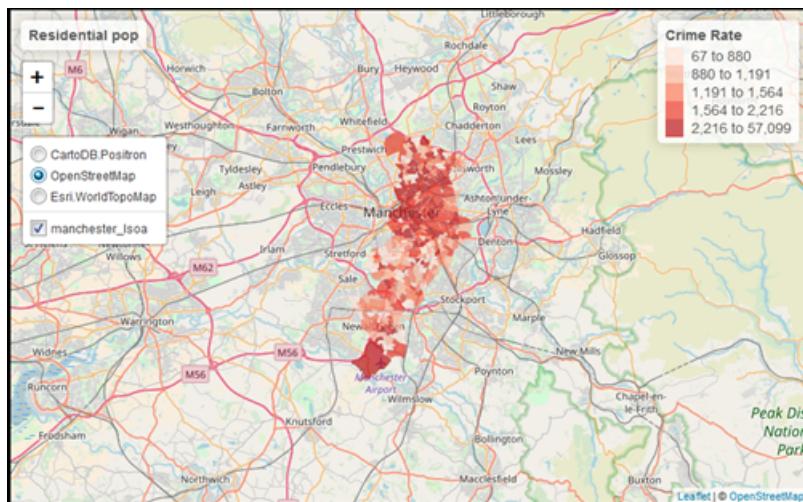


Once you have completed this activity, let's explore your map with the crime rate using the usual residents as the denominator using the interactive way. Assuming you name that visualisation map5 you could use the following code.

```
tmap_mode("view")
map5
```

You may find it useful to shift to the OpenStreetMap view by clicking in the box to the left, since it will give you a bit more contextual information than the default CartoDB basemap.

In the first lecture we spoke a bit about Open Street Map, but if you're interested it's definitely worth reading up on. As I mentioned, Open Street Map is a non-profit foundation whose aim is to support and enable the development of freely-reusable geospatial data, and relies heavily on volunteers participating in this project to map their local areas. You can have a look here for ongoing humanitarian projects, or read here about the mapping parties I told you about. At the very least though, in the spirit of open source and crowdsourcing, take a moment to appreciate that all these volunteers of people just like you have contributed to creating such a detailed geographical database of our world. That's definitely something kinda cool to think about!



3.6.2 Homework 3

What areas of the city seem to have the largest concentration of crime? Why? Does deprivation help you to understand the patterns? What's going on in the LSOA farthest to the South? Why does it look like a crime hotspot in this map and not in the one using the workday population?

3.7 More on small multiples and point pattern maps

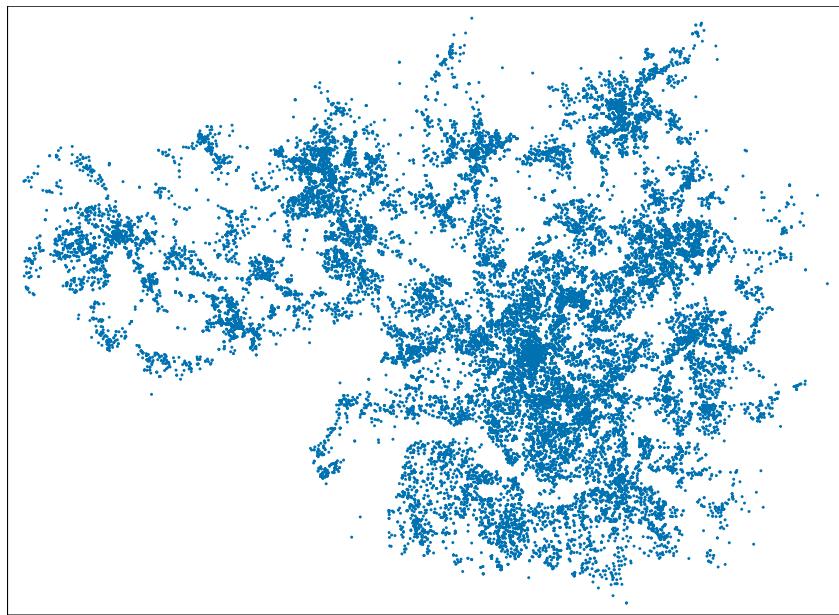
Last week we showed you how to visualise point patterns using data from the Police UK website. One of the homeworks ask you to discuss the map you produced using `leaflet` in which you used type of crimes to colour your points. One of the problems with that map was that you had so many levels within that variable that it was very difficult to tell what was what. Even some of the colors in different categories were not that different from each other. That's a typical situation where faceting or using small multiples would have been a better solution.

What we are going to do require our data to be stored as a spatial object -as it is the case with `tmap`. So first we need to turn our data into a spatial object. For this we will use the `st_as_sf()` function. The `st_as_sf` will return a `sf` object using the geographical coordinates we specify, below you can see we also specify the coordinate system we are using. Since we didn't modify the `tmap_mode` from our last call we would still be running on the view rather than the plot format. We can go back to the plot format with a new `tmap_mode` call.

```
crime_sf <- st_as_sf(x = crimes,
                      coords = c("Longitude", "Latitude"),
                      crs = "+proj=longlat +datum=WGS84")
#For a simple map with all the points
tmap_mode("plot")
```

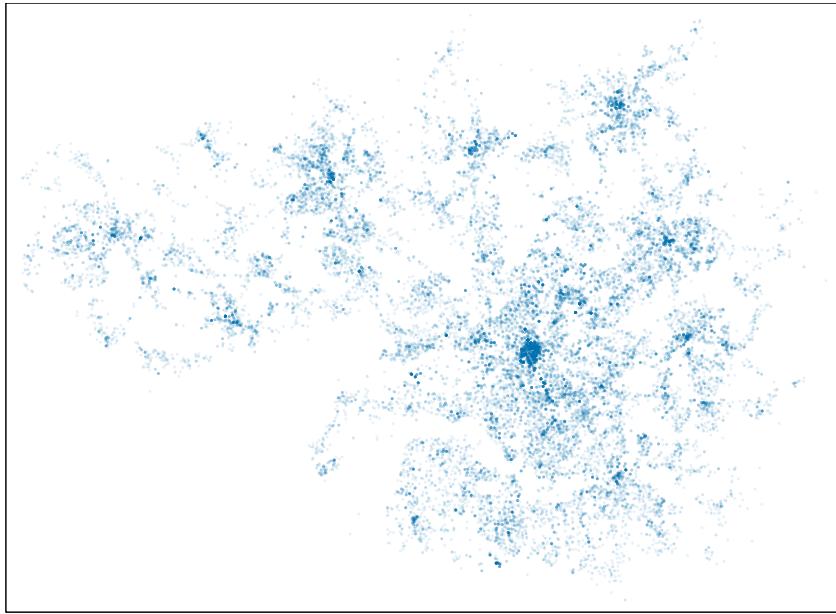
```
## tmap mode set to plotting
```

```
tm_shape(crime_sf) +
  tm_dots()
```



Here we are getting all the data points in the “crime_sf” object, which includes the whole of Greater Manchester. Also, since there are so many crimes, dots, it is hard to see patterns. We can add some transparency with the alpha argument as part of the `tm_dots` call.

```
tm_shape(crime_sf) +
  tm_dots(alpha=0.1)
```



Straight away you can start seeing some patterns, like the concentration of crimes in the town centres of all the different local authorities that conform Greater Manchester. This would be easier for you to see, if you are not familiar with the geography of Greater Manchester, if you place a basemap as a layer.

You can also use basemaps when working on the plot mode of `tmap`. We could for example use OpenStreet maps. We use the `read_osm` function from `tmaptools` package to do that, by passing `crime_sf` as an argument we will be bounding the scope of the basemap to the area covered by our point pattern of crimes. We will also need to load the `OpenStreetMap` package for this to work.

Remember that the first time you use a package you always have to install it, using the `install.packages()` function!

Now load the `OpenStreetMap` library into your working environment with the `library()` function:

```
library(OpenStreetMap)
```

Mac Users's Note: So if you're on a Mac, it's possible that at this point (or when trying to install the package) you've experienced an error. It might look something like this:

```
Error: package or namespace load failed for 'OpenStreetMap':  
.onLoad failed in loadNamespace() for 'rJava', details:  
call:  
dyn.load(file, DLLpath = DLLpath, ...) error: unable to load  
shared object '/Library/Frameworks/R.framework/Versions/3.4/Resources/library/rJava/libs/rJava.so'
```

```
dlopen(/Library/Frameworks/R.framework/Versions/3.4/Resources/library/rJava/libs/rJava
6): Library not loaded: @rpath/libjvm.dylib Referenced from:
/Library/Frameworks/R.framework/Versions/3.4/Resources/library/rJava/libs/rJava.so
Reason: image not found
```

This is because of some drama between the rJava and your Mac OS's dealing with Java. Do not despair though, the solution is simple. Open up your Terminal app. This is your command line for Mac. You might have used before for other things, or you might not. If you don't know what I'm on about with Terminal have a look at this helpful tutorial.

Now, once you have Terminal open, all you have to do is copy and paste the code below, and press Enter to run it:

```
sudo ln -f -s $(/usr/libexec/java_home)/jre/lib/server/libjvm.dylib
/usr/local/lib
```

After you press Enter, Terminal will ask you for your password. This is the password to your laptop. Type in your password, and hit enter again. Once that's all done, you can go back to R, and you will have to load a package called rJava(). Like so:

```
library(rJava)
```

Now you can again try loading the OpenStreetMap package.

```
library(OpenStreetMap)
```

Hopefully should all work smoothly now!

Back to everybody here!

You will also need another package called `tmaptools`. This gets installed when you installed `tmap`, but you still have to load it up.

```
library(tmaptools)
```

We will then add the basemap layer using the `qtm()` function that is a function provided by `tmap` for quick plotting.

First, create an object, let's call it `gm_osm`, by reading Open Street Map (OSM) data with the `read_osm()` function. This function reads and returns OSM tiles are read and returns as a spatial raster, or queries vectorized OSM data and returns as spatial polygons, lines, and/or points.

```
gm_osm <- read_osm(crime_sf)
```

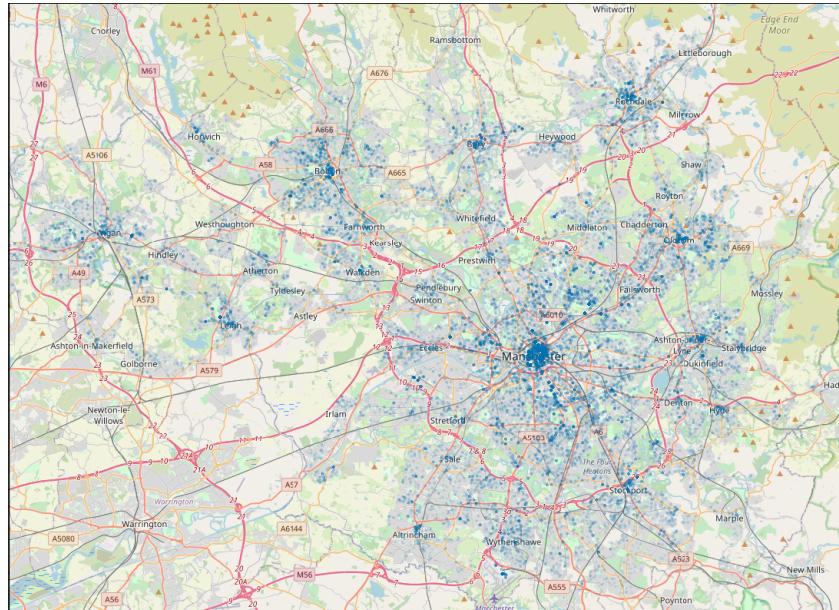
It might take a while to get this data, as it is quite a large set of data, being extracted from the open street map database. Just think how long it took all the volunteers to map these areas. Compared to that, the little waiting time to get this onto your computer for your map should be nothing!

Now you can use the `qtm()` function to draw a quick thematic map. Any ideas as to why it's called qtm yet? Let's try again. You can draw a **Quick Thematic Map**.

Again we use the `tm_shape()` and `tm_dots()` functions to specify the presentation of the map, with `tm_shape()` specifying the shape object (in this case `crime_sf`), and `tm_dots()` to draw symbols, including specifying the color, size, and shape of the symbols. In this case we adjust the transparency with the `alpha` parameter.

```
qtm(gm_osm) +
  tm_shape(crime_sf) +
  tm_dots(alpha=0.1)
```

```
## Warning: Raster values found that are outside the range [0, 255]
```

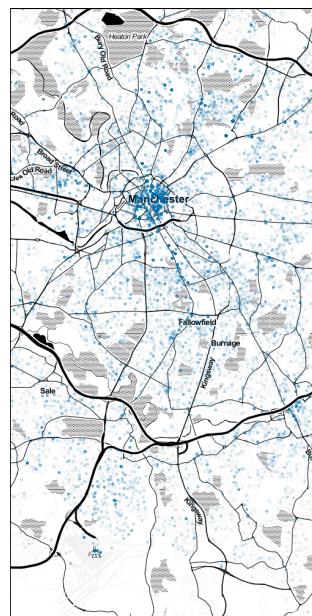


Let's zoom in into Manchester city, for which we can use our `manchester_lsoa` map.

```
mc_osm <- read_osm(manchester_lsoa, type = "stamen-toner")

qtm(mc_osm) +
  tm_shape(crime_sf) +
  tm_dots(alpha=0.1)

## Warning: Raster values found that are outside the range [0, 255]
```



We can use the `bb()` function from `tmaptools` to create a bounding box around the University of Manchester, the width and height parameters I am using determine the degree of zoom in (I got these experimenting with different ones until I got the right zooming around the University location). Once you have this bounding box you can pass it as an argument to the `read_osm()` function that will look for the basemap around that location.

```
#Create the bounding box
UoM_bb <- bb("University of Manchester", width=.03, height=.02)
#Read the basemap from using the stamen toner background
UoM_osm <- read_osm(UoM_bb, type = "stamen-toner")
#Plot the basemap
qtm(UoM_osm)

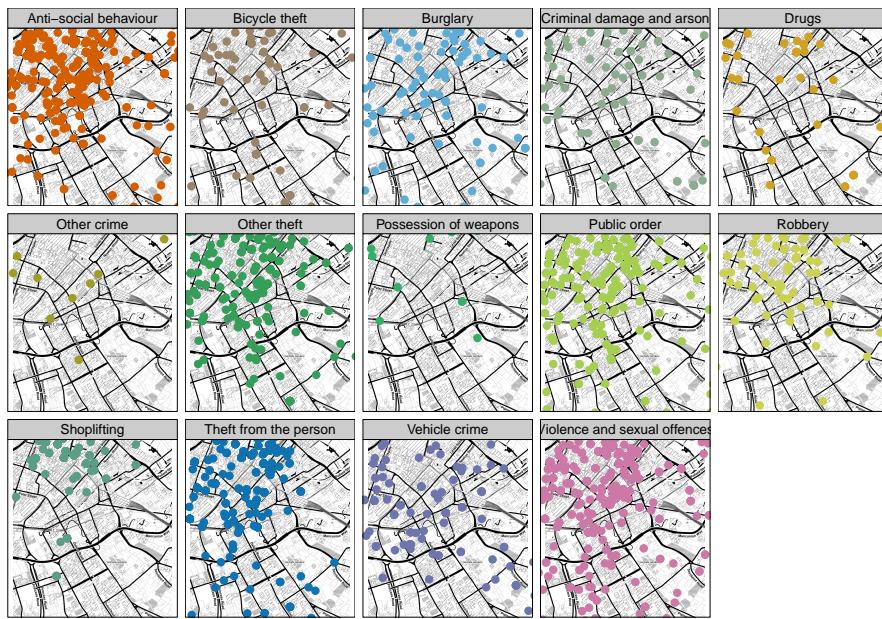
## Warning: Raster values found that are outside the range [0, 255]
```



Now that we have our basemap we can run our small multiples. Because we have a variable that determines the types we can use a different way to `tmap_arrange()` explained above, we can essentially ask `tmap` to create a map for each of the levels in our organising variable (in this case `Crime.type`). So instead of `tmap_arrange()` that requires the creation of each map, when each map simply represents different levels of an organising variable we can simplify the syntax using `tm_facets()` and within this function we specify as the first argument the name of the variable that has the different categories we want to map out. The second argument you see below `free.coords` set to FALSE simply ensures that the map gets bounded to the basemap, if you want to see what happens if you change it, just set it to TRUE instead.

```
qtm(UoM_osm) +
  tm_shape(crime_sf) +
  tm_dots(size=0.5, col = "Crime.type") +
  tm_facets("Crime.type", free.coords=FALSE) +
  tm_layout(legend.show=FALSE)
```

```
## Warning: Raster values found that are outside the range [0, 255]
```



We could do some further tweaking around for ensuring things look a bit neater. But we have covered a lot of ground today, and you should all give yourself a congratulatory “well done”.

Chapter 4

Performing spatial operations in R

By now you have come a long way in terms of taking your spatial data, and visualising it using maps, and being able to present the values of a variable using thematic maps. You have had some practice in taking data which has a spatial component, and joining it to a shapefile, using the common column, in order to be able to visually demonstrate variation on something, such as the crime rate, across space.

I hope that you are finding this to be really exciting stuff, and an opportunity to get yourselves accustomed to spatial data. If there is anything you are unsure about, or want to catch up on, please do not hesitate to revisit older material, and ask us questions about it. We build on each week acquiring knowledge cumulatively, so don't let yourself get stuck anywhere down the line. But, if you're ready, today we will go a step further, and get your hands dirty with **spatial manipulation of your data**.

Thus far, our data manipulation exercises were such that you might be familiar with, from your earlier exposures to data analysis. Linking datasets using a common column, calculating a new variable (new column) from values of existing variables, these are all tasks which you can perform on spatial or non-spatial data. However today we will explore some exercises in data manipulation which are specific to *spatial* data analysis. After this session you can truly say you are masters of spatial data manipulation. So let's get started with that!

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

- used **geocoding** methods to translate postcodes into geographic coordinates
- made interactive point map with leaflet

- met a new format of spatial shape file called **geojson**
- subset points that are within a certain area using a **spatial operation**
- created new polygons by generating **buffers** around points
- counted the number of points that fall within a polygon (known as **points in polygon**)

These are all very useful tools for the spatial crime analyst, and we will hope to demonstrate this by working through an example project, where you would make use of all of these tools.

Let's consider 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.

(If you are interested further in crime attractors vs crime generators I recommend a read of Brantingham, P., & Brantingham, P. (1995). *Criminality of place*. European journal on criminal policy and research, 3(3), 5-26.)

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. So let's get to it!

4.1 Getting some (more) data

Manchester City Council have an Open Data Catalogue on their website, which you can use to browse through what sorts of data they release to the public. 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 or Dallas but we'll take it.

One interesting data set, especially for our questions about the different alcohol outlets is the Licensed Premises data set. This details all the currently active licenced premises in Manchester. You can see there is a link to download now.

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 *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. If however you've saved this elsewhere, you will need to work out the file path.

Note my favourite shortcut to finding the file path is to simply run the `file.choose()` function, and use the popup window to navigate to the file. When you open this file through the popup window, if you're not assigning this to an object, it will simply print out the filepath to your console window. Like so:

```
> file.choose()
[1] "/Users/reka/Dropbox (The University of Manchester)/CrimeMapping/data/2017-11-greater-manchester-street.csv"
>
```

You can then copy and paste this path to whatever fuction you are assigning it to, to read in your data.

4.1.1 Reading data in from the web

But, programmers are lazy, and the whole point of using code-based interfaces is that we get to avoid doing unnecessary 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? Well think about what we do when we read in a file. 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 `<-`. Right? 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 (such as when you're creating a `shp_name` object and you pass it the string "path to my file"), or it could be some function, for example when you read a csv file with the `read.csv()` function.

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, right? Something like:

```
my_data <- read.csv("path to my file here")
```

Well what if your data does not live on your laptop or PC? Well, 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 the Licensed Premises data from the web?

You know when you right click on the link, and select "Save As..." or whatever you click on to save? You could, also select "Copy Link Address". This just copies the webpage where this data is stored. Give it a go! Copy the address, and then paste it into your browser. It will take you to a blank page where a forced download of the data will begin. So what if you pasted this into the `read.csv()` function?

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

Well in this case, the `my_data` object would be assigned the value returned from the `read.csv()` function reading in the file from the url you provided. File path is no mysterious thing, file path is simply the *path* to the *file* you want to read. If this is a website, then so be it.

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/licen
```

You can always check if this worked by looking to your global environment on the right 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 can see there are some interesting and perhaps less interesting columns in there. There are quite a lot of venues in this list as well. Let's think about subsetting them. Let's say we're interested in city centre manchester. We can see that there is a column for postcodes. We know (from our local domain knowledge) That city centre postcodes are M1-M4. So let's start by subsetting the data to include these.

4.1.2 Subsetting using pattern matching

We could use spatial operations here, and geocode all the postcodes at this point, then use a spatial file of city centre to select only the points contained in this area. The only reason we're not doing this is because the geocode function takes a bit of time to geocode each address. It would only be about 10 - 15 minutes, but we don't want to leave you sitting around in the lab for this long, so instead we will try to subset the data using pattern matching in text. In particular we will be using the `grepl()` function. This function takes a **pattern** and looks for it in some text. If it finds the pattern, it returns TRUE, and if it does not, it returns FALSE. So you have to pass two parameters to the `grepl()` function, one of them being the pattern that you want it to look for, and the other being the object in which to search.

So for example, if we have an object that is some text, and we want to find if it contains the letter "a", we would pass those inside the `grepl()` function, which would tell us TRUE (yes it's there) or FALSE (no it's not there):

```
some_text <- "this is some text that has some letter 'a's"
grepl("a", some_text)

## [1] TRUE
```

You can see this returns TRUE, because there is at least one occurrence of the letter a. If there wasn't, we'd get FALSE:

```
some_text <- "this is some text tht hs some letters"
grepl("a", some_text)

## [1] FALSE
```

So we can use this, to select all the cases where we find the pattern "M1" in the postcode. *NOTICE* the space in our search pattern. It's not "M1" it's "M1". Can you guess why?

Well, M1 will be found in M1 but also in M13, which is the University of Manchester's postcode, and not the part of city centre in which we are interested.

So let's subset our data by creating a new object `city_centre_prems`, and using the piping (%>%) and `filter()` functions from the `dplyr` package:

```
#remember to load dplyr package if you haven't already:
library(dplyr)
```

```

## 
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
## 
##     filter, lag

## The following objects are masked from 'package:base':
## 
##     intersect, setdiff, setequal, union

#then create the city_centre_prems object:
city_centre_prems <- lic_prem %>%
  filter(grep("M1 ", POSTCODE) )

```

Now we only have 353 observations (see your global environment), which is a much more manageable number.

4.1.3 Geocoding from an address

Great OK so we have this list of licensed premises, and we have their address, which is clearly *some* sort of spatial information, but how would you put this on a map?

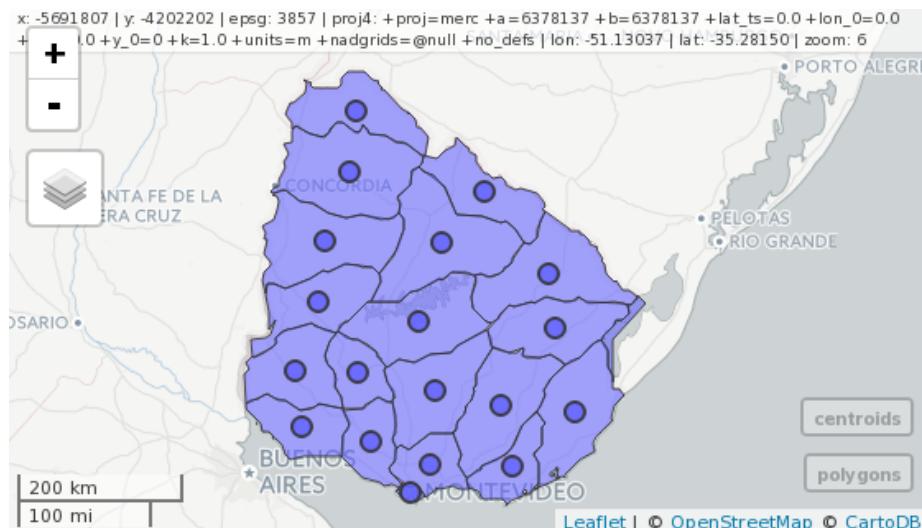
Any ideas?

We can, at the most basic, geocode the postcode. This will put all the establishments to the centroid of the postcode. Postcodes are used in the United Kingdom as alphanumeric codes, that were devised by Royal Mail. A full postcode is known as a “postcode unit” and designates an area with a number of addresses or a single major delivery point. You can search the Royal Mail for information on post codes here..

Here is a map of the postcode areas in Greater Manchester:



Now the centroid of the post code area represents the central point of the shapefile. For example, here you can see some polygons with their centroids illustrated by points:



This is not quite as precise as geocoding the actual address, but let's just stick with this approach for now.

So we need something that will help us get the coordinates for the relevant post code centroid. For this we can use the Open postcode geo from data.gov.uk. Open Postcode Geo is a postcode dataset and API optimised for geocoding applications. You can use Open Postcode Geo to geocode a dataset, geocode user input, and therefore build a proximity search. Data is derived from the Office for National Statistics postcode database and is free to use, **subject to including attributions to ONS, OS (Ordnance Survey) and Royal Mail**.

Postcodes can be entered at area, district, sector, and unit level - see Postcode map for the geographical relationship between these. We can use the Application Programme Interface (API) to query postcodes and read them directly into R, attaching a latitude and a longitude to our dataframe.

DETAIL AN EXAMPLE ONE HERE

```
library(rjson)

address <- "M13 9PL"
geocode_result <- fromJSON(readLines(paste0("http://api.getthedata.com/postcode/", gsub

## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :
## incomplete final line found on 'http://api.getthedata.com/postcode/M13+9PL'

geocode_result$data$latitude

## [1] "53.466926"
```

```
geocode_result$data$longitude
```

```
## [1] "-2.233578"
```

Wrap into functions to apply to dataframe

```
geocode_addys_getlng <- function(x){

  geocode_result <- fromJSON(readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", "", x)))
  return(geocode_result$data$longitude)
}

geocode_addys_getlat <- function(x){

  geocode_result <- fromJSON(readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", "", x)))
  return(geocode_result$data$latitude)
}
```

apply to df - describe more and figure out why it falls over on whole df

```
library(purrr)

city_centre_prems <- head(city_centre_prems, n = 100) %>%
  mutate(longitude = map_chr(POSTCODE, geocode_addys_getlng),
         latitude = map_chr(POSTCODE, geocode_addys_getlat))

## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :
## incomplete final line found on 'http://api.getthedata.com/postcode/M15JG'

## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :
## incomplete final line found on 'http://api.getthedata.com/postcode/M15QA'

## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WD'

## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :
## incomplete final line found on 'http://api.getthedata.com/postcode/M16DE'

## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :
## incomplete final line found on 'http://api.getthedata.com/postcode/M17AG'

## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :
## incomplete final line found on 'http://api.getthedata.com/postcode/M12EQ'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13AQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13EF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HN'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16DD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11HP'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14EE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14AH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15NJ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12DA'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12AP'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14RL'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17AG'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13JE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15JQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13BH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16NF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DZ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11PE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11NA'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HW'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11LY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14BH'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13NR'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13NR'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14QU'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14PY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14GX'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11WT'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LZ'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13GF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15NZ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HU'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DU'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12JW'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11RG'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14QX'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LA'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12BS'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15SH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14LY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14GS'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13NJ'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13AQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16NG'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12AP'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14FD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LZ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17EN'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14GX'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16DD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15WW'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16FT'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17HL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13NB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11JN'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11PW'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14FH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12AN'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13EZ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17EL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15NH'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16JB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12EE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17HL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13PJ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16FU'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12DB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13EF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16HY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16DD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15JG'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15QA'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16DE'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17AG'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12EQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13AQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13EF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HN'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16DD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11HP'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14EE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14AH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15NJ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12DA'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12AP'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14RL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17AG'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13JE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15JQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13BH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16NF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DZ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11PE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11NA'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HW'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11LY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14BH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13NR'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13NR'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14QU'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14PY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14GX'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HE'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11WT'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LZ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13GF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15NZ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HU'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DU'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12JW'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11RG'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14QX'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LA'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12BS'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15SH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14LY'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14GS'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13NJ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13AQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16NG'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13HE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12AP'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14FD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LZ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17EN'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14GX'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LY'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16DD'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15WW'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13WF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16FT'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17HL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15LH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13NB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11JN'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M11PW'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14HQ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M14FH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12AN'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17DL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13EZ'
```

```
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17EL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M15NH'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16JB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12EE'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M17HL'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13PJ'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16FU'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M12DB'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13EF'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M13LY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16HY'  
  
## Warning in readLines(paste0("http://api.getthedata.com/postcode/", gsub(" ", :  
## incomplete final line found on 'http://api.getthedata.com/postcode/M16DD'
```

Be patient, this will take a while, each postcode has to be referenced against their database and the relevant coordinates extracted. For each point you will see a note appear in red, and while R is working you will see the red stop sign on the top right corner of the Console window:

```
239:25 | Geocoding from an address
Console R Markdown ×
~/Dropbox (The University of Manchester)/CrimeMapping/ ↗
Information from URL : http://www.datasciencetoolkit.org/maps/api/geocode/json?address=%20M1%203BB,%20UK&sensor=f
else
Information from URL : http://www.datasciencetoolkit.org/maps/api/geocode/json?address=%20M1%206HA,%20UK&sensor=f
else
Information from URL : http://www.datasciencetoolkit.org/maps/api/geocode/json?address=%20M1%203NY,%20UK&sensor=f
else
Information from URL : http://www.datasciencetoolkit.org/maps/api/geocode/json?address=%20M1%205NQ,%20UK&sensor=f
else
Information from URL : http://www.datasciencetoolkit.org/maps/api/geocode/json?address=%20M1%204JY,%20UK&sensor=f
else
Information from URL : http://www.datasciencetoolkit.org/maps/api/geocode/json?address=%20M1%20%20%203LY,%20UK&se
nsor=false
Information from URL : http://www.datasciencetoolkit.org/maps/api/geocode/json?address=%20M1%205WN,%20UK&sensor=f
else
```

Also think about how incredibly fast and easy this actually is, if you consider a potential alternative where you have to manually find some coordinates for each address. That sounds pretty awful, doesn't it? Compared to that, setting the above functions running, and stepping away to make a cup of tea is really a pretty excellent alternative, no?

Right so hopefully that is done now, and you can have a look at your data again to see what this new column looks like. Remember you can use the `View()` function to make your data appear in this screen.

```
View(city_centre_prem)
```

And now we have a column called `longitude` for longitude and a column called `latitude` for latitude. Neat!

4.2 Making interactive maps with leaflet

Thus far we have explored a few approaches to making maps. We made great use of the `tmaps` package for example in the past few weeks.

As we saw in earlier sessions, Leaflet is 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 whose 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:

4.2.1 Making a map

To make a map, just load the leaflet library:

```
library(leaflet)
```

You then create a map with this simple bit of code:

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

And just print it:

```
m
```

This should all be familiar from earlier.

4.2.2 Adding some content:

You might of course want to add some content to your map.

4.3 Adding points manually:

You can add a point manually:

```
m <- leaflet() %>%
  addTiles() %>%
  addMarkers(lng=-2.230899, lat=53.464987, popup="You are here")
m
```

Or 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

```

4.3.1 Adding data from elsewhere

Last time around we added crime data to our map. In this case, we want to be mapping our licensed premises in the city centre, right? So let's do this:

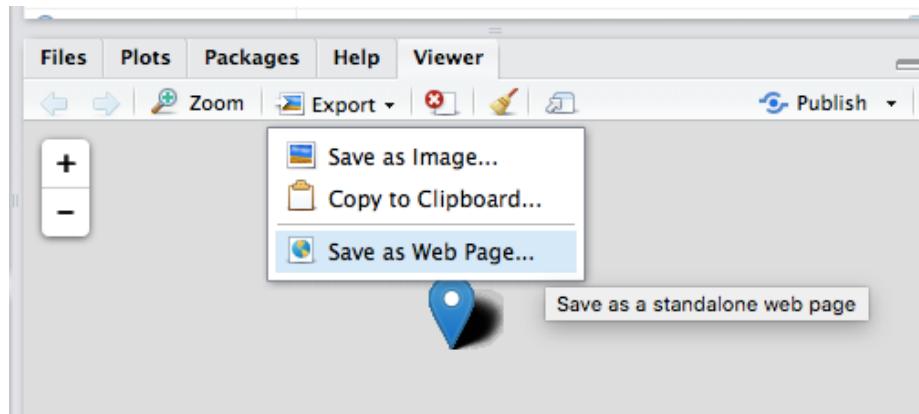
```

city_centre_prems$latitude <- as.numeric(city_centre_prems$latitude)
city_centre_prems$longitude <- as.numeric(city_centre_prems$longitude)

m <- leaflet(data = city_centre_prems) %>%
  addProviderTiles("Stamen.Toner") %>%
  addMarkers(lng=~longitude, lat=~latitude, popup=~as.character(PREMISESNAME), label =
m

```

Should be looking familiar as well. 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 not on this course, and make them jealous of your fancy map making skills.

One thing you might have noticed is that we still have some points that are not in Manchester. This should illustrate that the pattern matching approach is really just a work-around. Instead, what we really should be doing to subset our data spatially is to use spatial operations. So now we'll learn how to do some of these in the next section.

4.4 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 the relevant chapter Chapter 4: Spatial Operations from the project of Robin Lovelace and Jakub Nowosad, Geocomputation with R.

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 sufficient? 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.

For our spatial operations we will be using functions that belong to the `sf` package. So make sure you have this loaded up:

```
library(sf)
```

```
## Linking to GEOS 3.8.1, GDAL 2.4.4, PROJ 4.9.1
```

4.4.1 Coordinate reference systems revisited

One important note before we begin to do this brings us back to some of the learning from the second session on map projections and coordinate reference systems, like we discussed in the lecture today. We spoke about all the ways

of flattening out the earth, and ways of making sense what that means for the maps, and also how to be able to point to specific locations within these. The latter refers to the **Coordinate Reference System** or CRS the most common ones we will use are **WGS 84** and **British National Grid**.

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 licenced premises:

```
st_crs(city_centre_prems)
```

```
## 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”. “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.

```
cc_spatial = st_as_sf(city_centre_prems, coords = c("longitude", "latitude"),
                      crs = 4326, agr = "constant")
```

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

```
st_crs(cc_spatial)

## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
## GEOGCS["WGS 84",
##       DATUM["WGS_1984",
##             SPHEROID["WGS 84",6378137,298.257223563,
##                   AUTHORITY["EPSG","7030"]],
##             AUTHORITY["EPSG","6326"]],
##       PRIMEM["Greenwich",0,
##              AUTHORITY["EPSG","8901"]],
##       UNIT["degree",0.0174532925199433,
##             AUTHORITY["EPSG","9122"]],
##       AUTHORITY["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.

4.4.2 Meet a new format of shapefile: 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.

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("https://raw.githubusercontent.com/RUMgroup/Spatial-data-in-R/master/r
```

```
## Reading layer `wards` from data source `https://raw.githubusercontent.com/RUMgroup/...`
```

```
## Simple feature collection with 215 features and 12 fields
```

```
## geometry type:  POLYGON
```

```
## dimension:      XY
```

```
## bbox:           xmin: 351664 ymin: 381168.6 xmax: 406087.5 ymax: 421039.8
```

```
## CRS:            27700
```

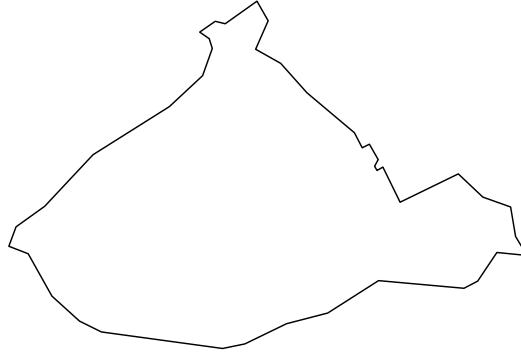
Let's select only the city centre ward, using the `filter()` function from dplyr

```
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 included in `cc_spatial` are in fact only licensed premises in the city centre. This will be your first spatial operation. Excited? Let's do this!

4.4.3 Subset points to those within a polygon

So we have our polygon, our spatial file of the city centre ward. We now want to subset our point data, the `cc_spatial` data, which has points representing licensed premises.

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

```
st_crs(city_centre) == st_crs(cc_spatial)
```

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

Uh oh! They do not! So what can we do? Well we already know that cc_spatial is in WGS 84, because we made it so a little bit earlier. What about this new city_centre polygon?

```
st_crs(city_centre)
```

```
## Coordinate Reference System:
##   User input: 27700
##   wkt:
## PROJCS["OSGB 1936 / British National Grid",
##        GEOGCS["OSGB 1936",
##              DATUM["OSGB_1936",
##                    SPHEROID["Airy 1830",6377563.396,299.3249646,
##                           AUTHORITY["EPSG","7001"]]],
##              TOWGS84[446.448,-125.157,542.06,0.15,0.247,0.842,-20.489],
##              AUTHORITY["EPSG","6277"]],
##              PRIMEM["Greenwich",0,
##                     AUTHORITY["EPSG","8901"]],
##              UNIT["degree",0.0174532925199433,
##                  AUTHORITY["EPSG","9122"]],
##              AUTHORITY["EPSG","4277"]],
##              PROJECTION["Transverse_Mercator"],
##              PARAMETER["latitude_of_origin",49],
##              PARAMETER["central_meridian",-2],
##              PARAMETER["scale_factor",0.9996012717],
##              PARAMETER["false_easting",400000],
##              PARAMETER["false_northing",-100000],
##              UNIT["metre",1,
##                  AUTHORITY["EPSG","9001"]],
##              AXIS["Easting",EAST],
##              AXIS["Northing",NORTH],
##              AUTHORITY["EPSG","27700"]]
```

Aha, the key is in the 27700. This code in fact stands for.... British National Grid...!

So what can we do? We can **transform** our spatial object. Yepp, we can convert between CRS.

So let's do this now. To do this, we can use the **st_transform()** function.

```
cc_WGS84 <- st_transform(city_centre, 4326)
```

Let's check that it worked:

```
st_crs(cc_WGS84)
```

```
## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
##     GEOGCS["WGS 84",
##       DATUM["WGS_1984",
##         SPHEROID["WGS 84",6378137,298.257223563,
##           AUTHORITY["EPSG","7030"]],
##           AUTHORITY["EPSG","6326"]],
##       PRIMEM["Greenwich",0,
##         AUTHORITY["EPSG","8901"]],
##       UNIT["degree",0.0174532925199433,
##         AUTHORITY["EPSG","9122"]],
##         AUTHORITY["EPSG","4326"]]
```

Looking good. Triple double check:

```
st_crs(cc_WGS84) == st_crs(cc_spatial)
```

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

YAY!

Now we can move on to our spatial operation, where we select only those points within the city centre polygon. To do this, we can use the `st_intersects()` function:

```
# intersection
cc_intersects <- st_intersects(cc_WGS84, cc_spatial)

## although coordinates are longitude/latitude, st_intersects assumes that they are pl
```

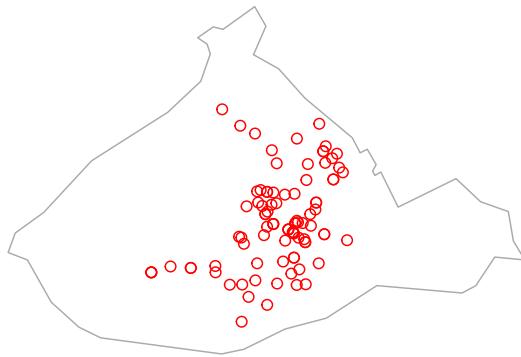
```
# subsetting
cc_intersects <- cc_spatial[unlist(cc_intersects),]
```

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

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

We can plot this too to have a look:

```
# plot
plot(st_geometry(cc_WGS84), border="#aaaaaa")
plot(st_geometry(cc_intersects), col = "red", add=T)
```



COOL, we have successfully performed our first spatial operation, we managed to subset our points data set to include only those points which are inside the polygon for city centre. See how this was much easier, and more reliable than the hacky workaround using pattern matching? Yay!

4.4.4 Spatial operations two: building buffers

Right, but what we want to do really to go back to our original question. We want to know about crime in and around out areas of interest, in this case our licensed premises. But how can we count this?

Well first we will need crime data. Let's use the same data set from last week. I'm not going over the detail of how to read this in, if you forgot, go back to the notes from last week.

```
crimes <- read.csv("data/2017-11-greater-manchester-street.csv")
```

Now let's make sure again that R is aware that this is a spatial set of points, and that the columns

```
crimes_spatial = st_as_sf(crimes, coords = c("Longitude", "Latitude"),
                           crs = 4326, agr = "constant")
```

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"
```

Or you can have a look at the first 6 lines of your dataframe with the `head()` function:

```
head(crimes)
```

```
##                                            Crime.ID Month
## 1                               2017-11
## 2 f892dce3e7a4c45fe4f8f09f24d6a494f2b49783a976afba3d6f51e01f72521f 2017-11
## 3 f48d18a3e88eaeba043807cd95642dccb2a2e007021de06e55956c3ee7bad0a4 2017-11
## 4                               2017-11
## 5 bee6bb133a8ae0b35cf19d2d994ad2bdc3852d1d0d560ce8510359dd02d2e503 2017-11
## 6 7f5838988383f952ec1445fb266cd731d5fa32831c3cc27c234f2a3696dfc2d7 2017-11
##               Reported.by Falls.within Longitude Latitude
## 1 Greater Manchester Police Greater Manchester Police -2.462774 53.62210
## 2 Greater Manchester Police Greater Manchester Police -2.462774 53.62210
## 3 Greater Manchester Police Greater Manchester Police -2.462774 53.62210
## 4 Greater Manchester Police Greater Manchester Police -2.464422 53.61250
## 5 Greater Manchester Police Greater Manchester Police -2.444807 53.61151
## 6 Greater Manchester Police Greater Manchester Police -2.444043 53.62939
##               Location LSOA.code LSOA.name
## 1 On or near Scout Road E01012628 Blackburn with Darwen 018D
## 2 On or near Scout Road E01012628 Blackburn with Darwen 018D
## 3 On or near Scout Road E01012628 Blackburn with Darwen 018D
```

```

## 4 On or near Parking Area E01004768 Bolton 001A
## 5 On or near Belmont Road E01004768 Bolton 001A
## 6 On or near West Walk E01004803 Bolton 001B
## Crime.type Last.outcome.category
## 1 Anti-social behaviour
## 2 Criminal damage and arson Investigation complete; no suspect identified
## 3 Criminal damage and arson Unable to prosecute suspect
## 4 Anti-social behaviour
## 5 Violence and sexual offences Under investigation
## 6 Other theft Investigation complete; no suspect identified
## Context
## 1 NA
## 2 NA
## 3 NA
## 4 NA
## 5 NA
## 6 NA

```

Or you can view, with the `View()` function.

Now, 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 licensed premises, and say that we will count all the crimes which fall within a specific radius of this licensed premise. What should this radius be? Well this is where your domain knowledge as criminologist 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 again one of them *it depends* questions. 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 the recommended distance for accessible bus stop guidance, so basically as far as people should walk to get to a bus stop (TfL, 2008). 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:

```

prem_buffer <- st_buffer(cc_intersects, 1)

## Warning in st_buffer.sfc(st_geometry(x), dist, nQuadSegs, endCapStyle =
## endCapStyle, : st_buffer does not correctly buffer longitude/latitude data

## dist is assumed to be in decimal degrees (arc_degrees).

```

You should get a warning here, like I did above. This message indicates that sf assumes a distance value is given in degrees. This is because we have lat/long data (WSG 48)

One quick fix to avoid this message, is to convert to BNG:

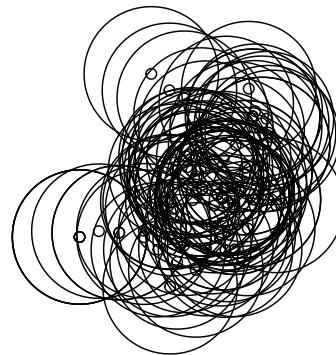
```
prem_BNG <- st_transform(cc_intersects, 27700)
```

Now we can try again, with meters

```
prem_buffer <- st_buffer(prem_BNG, 400)
```

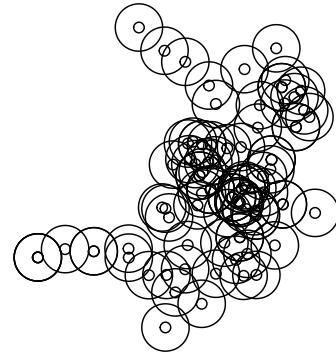
Let's see how that looks:

```
plot(st_geometry(prem_buffer))
plot(st_geometry(prem_BNG), add = T)
```



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

```
prem_buffer_100 <- st_buffer(prem_BNG, 100)
plot(st_geometry(prem_buffer_100))
plot(st_geometry(prem_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.

Well now let's have a look at our crimes. I think it might make sense (again using domain knowledge) to restrict the analysis to violent crime. So let's do this:

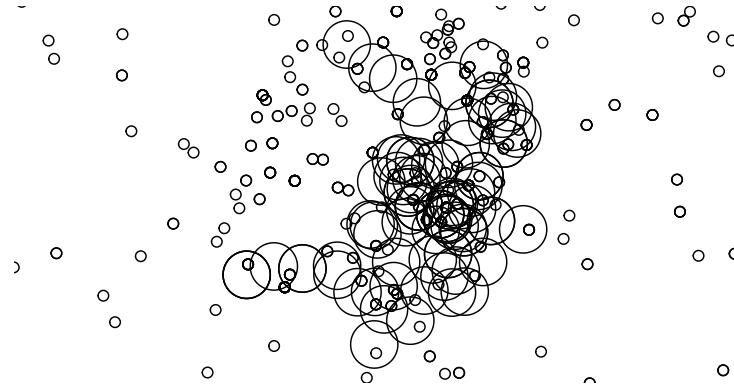
```
violent_spatial <- crimes_spatial %>%
  filter(Crime.type == "Violence and sexual offences")
```

Now, remember the CRS is WGS 48 here, so we will need to convert our buffer layer back to this:

```
buffer_WGS84 <- st_transform(prem_buffer_100, 4326)
```

Now let's just have a look:

```
plot(st_geometry(buffer_WGS84))
plot(st_geometry(violent_spatial), add = T)
```



OKAY, so some crimes fall inside some buffers, others not so much. Well, let's get to our last spatial operation of the day, the famous points in polygon, to get to answering which licensed premises have the most violent crimes near them.

4.4.5 Points in 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 <- violent_spatial %>%
  st_join(buffer_WGS84, ., left = FALSE) %>%
  count(PREMISESNAME)
```

```
## although coordinates are longitude/latitude, st_intersects assumes that they are pl
```

You now have a new dataframe, `crimes_per_prem` which has a column for the name of the premises, 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? Are you surprised?

Now as a final step, let's plot this, going back to leaflet. We can shade by the number of crimes within the buffer, and include a little popup label with the name of the establishment:

```
pal <- colorBin("RdPu", domain = crimes_per_prem$n, bins = 5, pretty = TRUE)
leaflet(crimes_per_prem) %>%
  addTiles() %>%
  addPolygons(fillColor = ~pal(n), fillOpacity = 0.8,
              weight = 1, opacity = 1, color = "black",
              label = ~as.character(PREMISESNAME)) %>%
  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 can talk about prettifying maps another day. You've done enough today.

4.5 Recap

Today we learned to:

- use **geocoding** methods to translate postcodes into geographic coordinates
- make interactive point map with leaflet
- about a new format of spatial shape file called **geojson**
- subset points that are within a certain area using a **spatial operation**
- create new polygons by generating **buffers** around points
- count the number of points that fall within a polygon (known as **points in polygon**)

4.6 Homework

In the homework you will now apply your learning to a new problem. To do so, please complete the following tasks:

Chapter 5

More on thematic maps

5.1 Introduction

In this session we are going to discuss some additional features around thematic maps we did not cover in week 3. We are going to discuss how to address some of the problems we confront when we are trying to use choropleth maps, as well as some alternatives to point based maps. We will also introduce the modifiable area unit problem.

Before we do any of this, we need to load the libraries we will use today:

```
library(sf)

## Linking to GEOS 3.8.1, GDAL 2.4.4, PROJ 4.9.1

library(tmap)
library(sp)
library(spdep)

## Loading required package: spData

## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge',
## repos='https://nowosad.github.io/drat/', type='source')` 

library(DCluster)

## Loading required package: boot
```

```

## Loading required package: MASS

library(cartogram)
library(ggmap)

## Loading required package: ggplot2

## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.

## Please cite ggmap if you use it! See citation("ggmap") for details.

library(ggplot2)
library(dplyr)

## 
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':
## 
##     select

## The following objects are masked from 'package:stats':
## 
##     filter, lag

## The following objects are masked from 'package:base':
## 
##     intersect, setdiff, setequal, union

```

There are also some new libraries we will use. If you don't already have these you will need to install them.

5.1.1 Pro-tip: do I need to install this package?

You might have noticed that in your list of available packages you might see more than you remember downloading. The idea of *dependencies* has come up throughout the semester. Packages have *dependencies* when their code is *dependent* on (uses code from) another package. For example, if I write some code that I think will be useful, so I release this in the form of the package “rekaR”, but I use ggplot2 in the code, then ggplot2 will be a dependency of rekaR. As a default, R will install all the dependencies for a package when you

install your package. So this way you might end up with some packages there that you didn't realise you had.

Why am I telling you this? Well you should always check if you have a package, before installing it. And I wanted to share with you some neat code from a Stackoverflow discussion (if you are not yet familiar with Stackoverflow you have not been Google-ing your error messages enough) here to do this. I'll comment it a bit, so you can follow along what it does **but you don't have to if you don't want to**. This is just an optional extra.

So as a first step, you have to assign a list of all the packages you have to check to an object. Let's say I tell you that today we will be using the following packages: "sp", "rgdal", "classInt", "RColorBrewer", "ggplot2", "hexbin", "ggmap", "XML", and "dplyr". Then you can add these to an object called `libs`, using the `c()` function:

```
libs <- c("sf", "tmap", "sp", "spdep", "DCluster", "cartogram")
```

Now you can run the below bit of code, and you will see in the console an output of what is and isn't installed, as well as install the packages that are not!

```
for (x in libs){
  if(x %in% rownames(installed.packages()) == FALSE) {
    print(paste0("installing ", x, "..."))
    install.packages(x)
  }
  else{
    print (paste0(x, " is already installed "))
  }
  library(x, character.only = TRUE)
}

## [1] "sf is already installed "
## [1] "tmap is already installed "
## [1] "sp is already installed "
## [1] "spdep is already installed "
## [1] "DCluster is already installed "
## [1] "cartogram is already installed "
```

#cycle through each item in libs
if the package is not installed
print a message to tell me
and then install the packages

#otherwise (if it is installed)
print a message to tell me

#and then load the packages

As you can see if you read through the comments there, this bit of code checks each package in the list you pass to the `libs` object when you create it, and if it is not installed it installs for you, and if it is, it just loads it for you. It can be a handy bit of code to keep around.

5.1.2 Data

Then we will bring back data about homicide across US counties. Some of you that have taken previous classes with us may be familiar with this dataset. The dataset was used as the basis for this study. We will be using it also later on in the semester. 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.

```
##R in Windows have some problems with https addresses, that's why we need to do this .
urlfile<-'https://s3.amazonaws.com/geoda/data/ncovr.zip'
download.file(urlfile, 'ncovr.zip')
#Let's unzip and create a new directory (ncovr) in our working directory to place the .shp files
unzip('ncovr.zip', exdir = 'ncovr')
```

Remember that to treat the data as spatial we need to load the shapefile. With that we can create a spatial object:

```
shp_name <- "ncovr/ncovr/NAT.shp"
ncovr_sf <- st_read(shp_name)
```

```
## Reading layer `NAT' from data source `/Users/reka/Dropbox (The University of Manchester)
## Simple feature collection with 3085 features and 69 fields
## geometry type: MULTIPOLYGON
## dimension: XY
## bbox: xmin: -124.7314 ymin: 24.95597 xmax: -66.96985 ymax: 49.37173
## CRS: 4326
```

5.2 Mapping rates, learning from disease mapping

In previous sessions 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 your 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. Let's look at the ncovr data.

```
summary(ncovr_sf$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 call Borden. Check it out:

```
borden <- subset(ncovr_sf, NAME == "Borden")
borden$HR60

## [1] 92.9368
```

Borden county 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 that 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 or the setting for Midsomer Murders? 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$P060
```

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

Well, there were about 1076 people living there.

```
summary(ncovr_sf$P060)
```

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

If you contrast that with 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 is 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.

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** 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 <- empbayessmooth(ncovr_sf$HC60, ncovr_sf$P060)
```

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

Instead of shrinking to the global rate, we can shrink to a rate based on the neighbours of each county. If instead of shrinking to a global rate, we shrink to a local rate, we may be able to take unobserved heterogeneity into account; for this we need the list of neighbours (we will discuss this code in a later session, so for now just trust us we are computing the rate of the areas that surround each country):

```
ncovr_sp <- as(ncovr_sf, "Spatial")
w_nb <- poly2nb(ncovr_sp, row.names=ncovr_sp$FIPSNO)
eb2 <- EBlocal(ncovr_sf$HC60, ncovr_sf$P060, w_nb)
ncovr_sf$HR60EBSL <- eb2$est * 100000
```

We can now plot the maps and compare them:

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

## tmap style set to "col_blind"

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

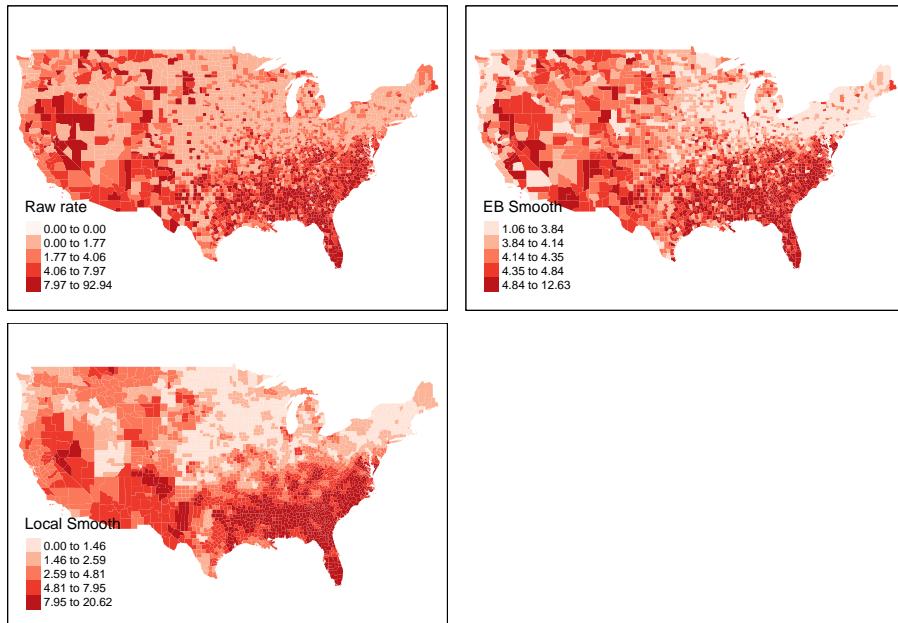
map1<- tm_shape(ncovr_sf) +
  tm_fill("HR60", style="quantile", title = "Raw rate", palette = "Reds") +
  tm_layout(legend.position = c("left", "bottom"),
            legend.title.size = 0.8,
            legend.text.size = 0.5)

map2<- tm_shape(ncovr_sf) +
  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_sf) +
  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.

5.3 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.

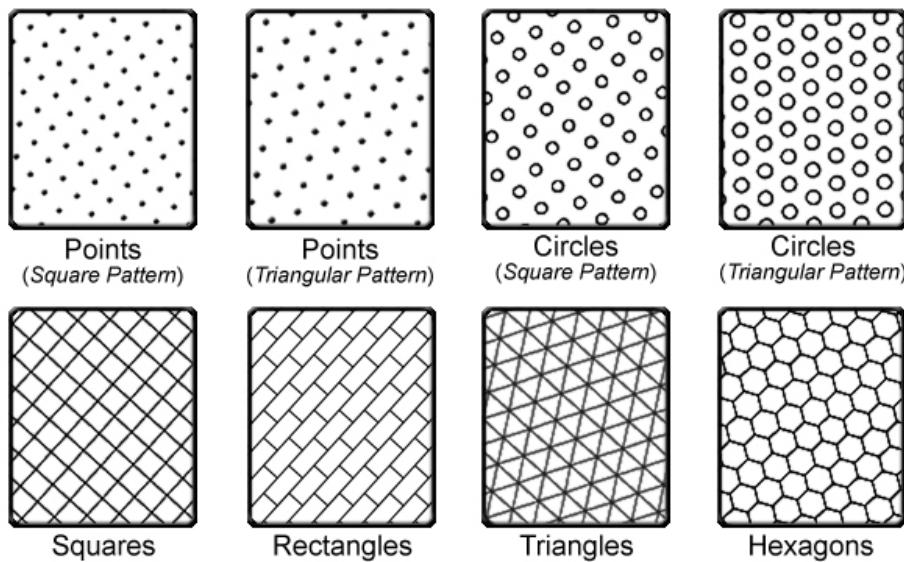
In many maps there are so many data points included that little can be interpreted from them. In order to reduce congestion on maps many GIS users and cartographers have turned to a process known as binning.

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.



5.3.1 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.

5.3.2 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.

5.3.2.1 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 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 it is generally 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.

5.3.2.2 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.

5.3.2.3 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). Hexagonal 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:

```
library(ggplot2)
library(ggmap)
library(hexbin)
```

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, I 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.

We can take our GMP crimes data, and select only the cases from ASB using the crime.type variable. If you want, however, I have already done this, so you can also download from my dropbox using the link here:

```
manchester_asb <- read.csv("https://www.dropbox.com/s/4tk0aps3jfd9nh4/manchester_asb.csv?dl=1")
```

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 found 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 a bespoke function, `sfc_as_cols()` created by Josh M. London in response to this issue opened on github for the sf package. To create this function here, run the below code:

```
sfc_as_cols <- function(x, names = c("x", "y")) {
  stopifnot(inherits(x, "sf") && inherits(sf::st_geometry(x), "sfc_POINT"))
  ret <- sf::st_coordinates(x)
  ret <- tibble::as_tibble(ret)
  stopifnot(length(names) == ncol(ret))
  x <- x[, !names(x) %in% names]
  ret <- setNames(ret, names)
  dplyr::bind_cols(x, ret)
}
```

As we are not covering making your own function much here, don't worry too much about the above code, but if you're curious, raise your hand in the labs and we can come around and talk through it.

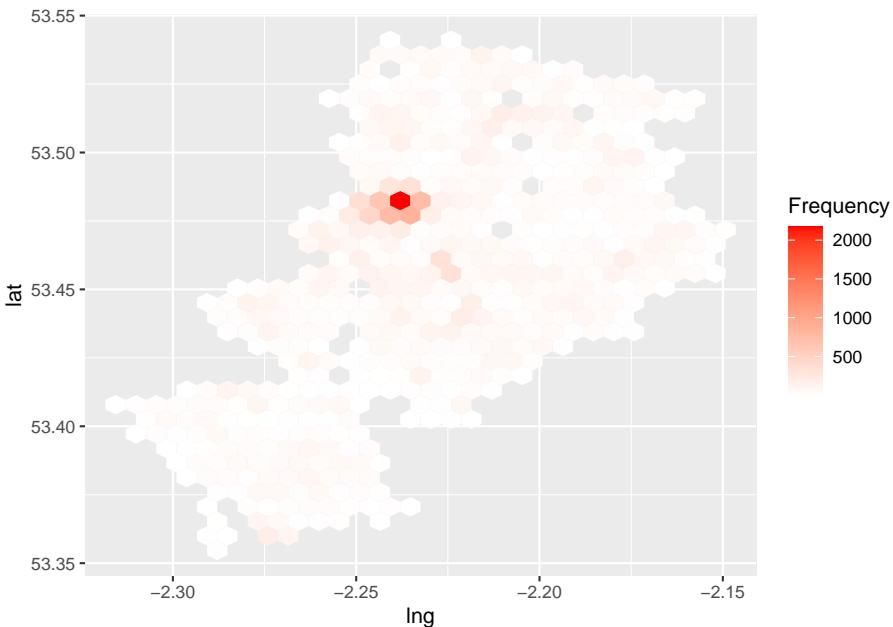
Now finally, let's use this function we just created to extract the coords to some columns. So in this function, we have to pass as parameters the name of the data frame, ma_spatial in our case, and also what we want the column to be called in a concatenated list (created with the `c()` function). Let's call these "lng" and "lat":

```
ma_spatial <- sfc_as_cols(ma_spatial, c("lng", "lat"))
```

As a first step, we can plot asb in the borough of Manchester using simple ggplot! Remember the data visualisation session from weeks ago? We discussed how ggplot 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. 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:

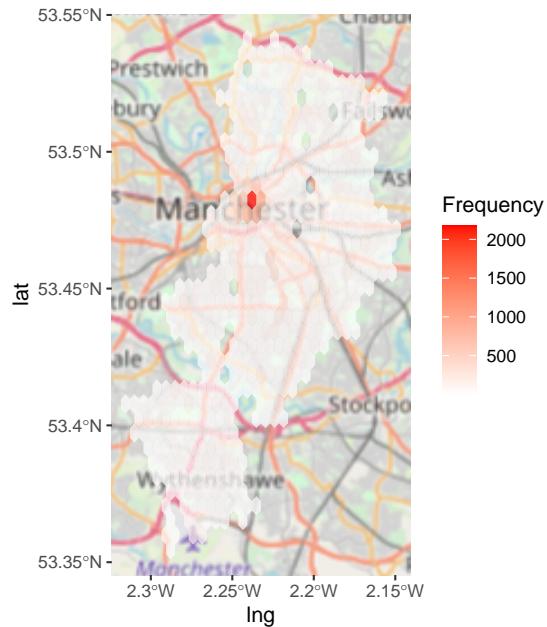
```
ggplot(ma_spatial, aes(lng, lat)) +  
  stat_binhex() +  
  scale_fill_gradientn(colours = c("white","red"), name = "Frequency")  
  #define data and variables for x and y  
  #add binhex layer (hexbin)  
  #add shading based on frequency
```



Neat, but doesn't quite tell us *where* that really dark hexbin 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:

```
library(ggspatial) #load ggspatial package for background map tiles  
  
ggplot(ma_spatial, aes(x = lng, y = lat)) +  
  annotation_map_tile() +  
  stat_binhex(alpha=0.7) +  
  scale_fill_gradientn(colours = c("white","red"), name = "Frequency")  
  #add binhex layer  
  #add shading based on frequency  
  
## although coordinates are longitude/latitude, st_intersects assumes that they are planar  
  
## Zoom: 10
```



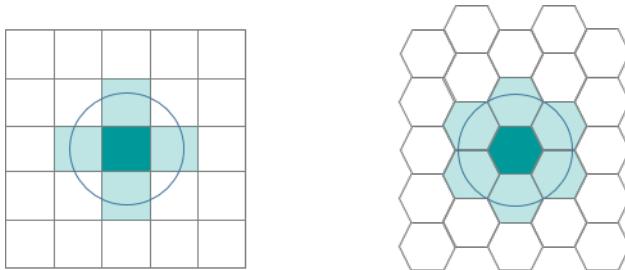
Now this should give you some more context! Woo!

So I mentioned we'd go over some reasons why you should consider aggregating into a hexagon grid rather than other shape:

- 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.



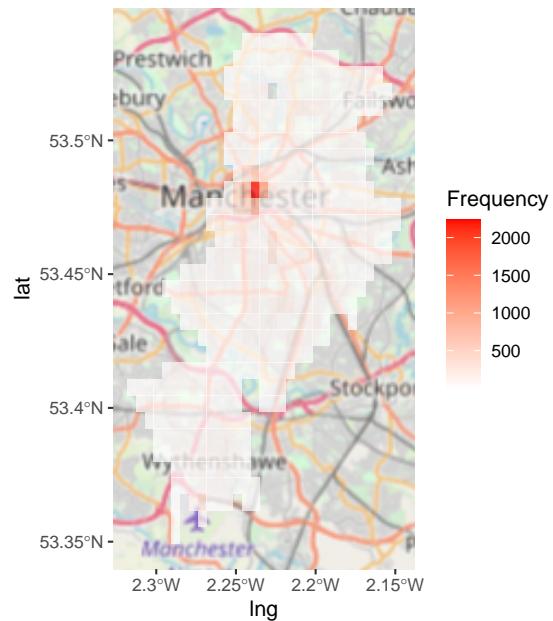
Now, to again illustrate the differences of different approaches, let's see what this map would look like with:

a) rectangular binning:

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

## although coordinates are longitude/latitude, st_intersects assumes that they are planar

## Zoom: 10
```

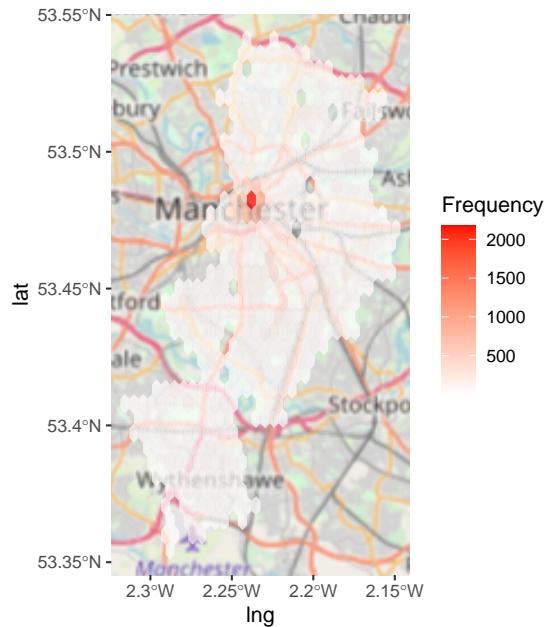


b) hexagonal binning:

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

```
## although coordinates are longitude/latitude, st_intersects assumes that they are pl
```

```
## Zoom: 10
```

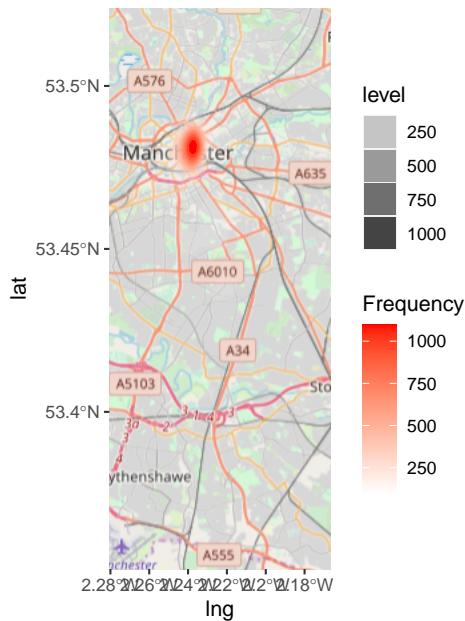


c) a simple “heatmap” (we will discuss these more thoroughly next week):

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

## although coordinates are longitude/latitude, st_intersects assumes that they are planar

## Zoom: 11
```



5.3.3 Homework 5.1

Look at the difference between the three maps (hex, rectangle, and density). How would your conclusions change if you were given these maps? Would you make different decisions about where to place your booths for the days of action? Why or why not? Discuss.

5.3.4 Multivariate binning

Multivariate binning is another binning method that lets you visualise slightly more complex data. In this method there can be many different variables consisting of different types of data. Like other binning methods the data is typically grouped with the sum or average of the data. Different types of symbology (such as size, shape and color) can also be used to represent this data as well.

We won't be covering this here but just so you can have a look at some examples here.

5.3.5 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.

5.4 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.

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.

In this lab we will focus on **MAUP**, but if you are interested in this kind of work, you should definitely read their paper to consider the other issues as well. There are techniques that can be used to alleviate each of the methodological difficulties, and they are described in accessible detail in their paper: 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.

5.4.1 What is MAUP?

The Modifiable Areal Unit Problem (MAUP) is an important issue for those who conduct spatial analysis using units of analysis at aggregations higher than incident level. It is one of the better-known problems in geography and spatial analysis. This phenomenon illustrates both the need for considering space in one's analysis, and the fundamental uncertainties that accompany real-world analysis.

The MAUP is "a problem arising from the imposition of artificial units of spatial reporting on continuous geographical phenomena, leading to artifacts or errors are created when one groups data into units for analysis."

The classic text on MAUP is the 1983 paper Openshaw, Stan. "The modifiable areal unit problem. CATMOG (Concepts and techniques in modern geography) 38." Geo Abstracts, Norwich. 1984..

There are two distinct types of MAUP: Scale (i.e. determining the appropriate size of units for aggregation) and zone (i.e. drawing boundaries or grouping).

5.4.1.1 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.

5.4.1.2 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

5.4.2 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.

5.4.3 What can we do?

Most often you will just have to remain aware of the MAUP and it's 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.

5.4.4 Homework 5.2

Look at the question for homework 5.1 about the three maps of binning and the hotspot map. Answer this question again, but now in light of what you have learned about MAUP.

5.5 Replacing polygons with grid or hex shapes

When you have meaningful spatial units of analysis in your polygons, for example you are interested specifically in Local Authorities, it might make sense to stick with what we did last week, and aggregate the points into these polygons to cheare 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. This article provides 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.

Broadly, the options are:

- cartogram
- hexmap
- grid

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. We focus on methods for transforming our maps to hex and gridap. We make grid and hex maps where instead of creating a grid, we reshape the existing polygons into grid or hex shapes. We will also speak about **cartograms**, but I'll leave that to the next section.

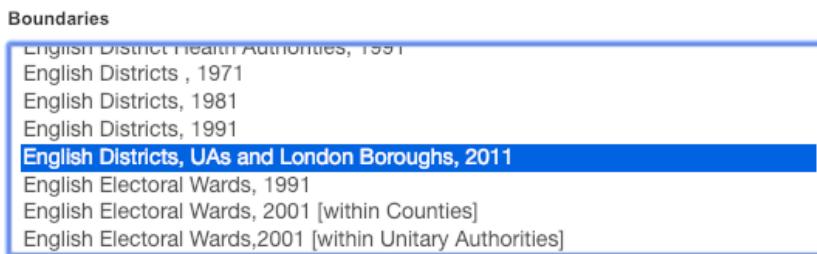
Let's explore this using the example of the results of the 2016 EU referendum at Local Authority level, where remain areas clustered in London. A simple thematic map does not necessarily communicate this well because Local Authorities are both small and densely populated in London.

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

```
eu_ref <- read.csv("https://www.electoralcommission.org.uk/sites/default/files/2019-07/EU-referendum-2016-data.csv")
```

OKAY, now we need a shapefile to join it to. Remember when we got the Manchester lsoa shapefile with the boundary selector? Let's go back, and this time get *Local Authority Districts for England*.

In this case that means select “English Districts, UAs and London Boroughs, 2011”:



Once you have the file, download, extract (unzip) and put the folder in your working directory. Mine is in a subfolder in my working directory called data, so I point R inside that folder to find my shape file.

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

```
## Reading layer `england_lad_2011_gen` from data source `/Users/reka/Dropbox (The Uni
## Simple feature collection with 326 features and 4 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:            xmin: 82644.8 ymin: 5349.399 xmax: 655976.9 ymax: 657599.5
## CRS:             27700
```

We can now join the EU referendum data, as we have learned in the past weeks:

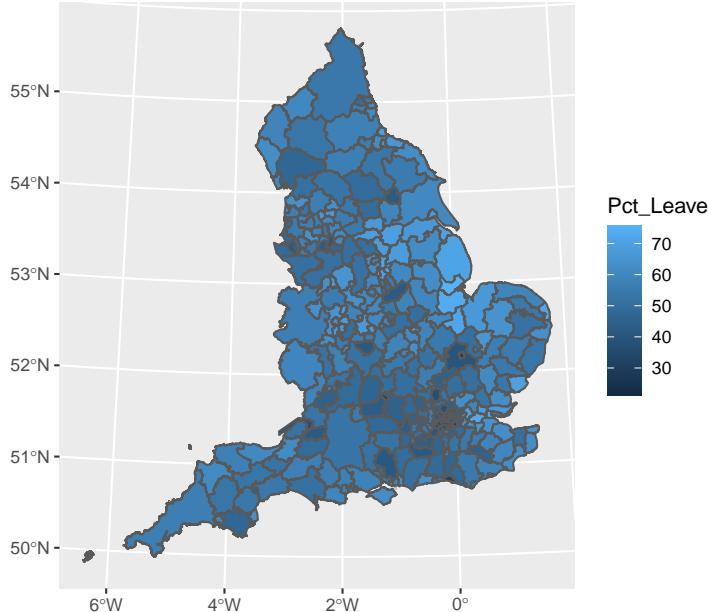
```
eu_sf <- left_join(las, eu_ref, by = c("name" = "Area"))
```

```
## Warning: Column `name`/`Area` joining factors with different levels, coercing to
## character vector
```

```
#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. Now what we can do is transform the shapes into squares or hexagons. Let's have a go at squares.

We can use the functions in the geogrid package.

```
library(geogrid)
```

Now we can use the `calculate_grid()` function. This function, given an input multipolygon spatial data frame this function calculates a hexagonal or regular grid, that strives to preserve the original geography. Once this is done, we assign each polygon in the original file to a new location in the gridded geometry. **NOTE THIS WILL TAKE A WHILE** again this is something that is quite computationally intensive, so you set it running, and go for a short break. You need short breaks in coding.

However this particular bit of code will actually take about 10 minutes. So if you're about to leave, wait until you are home to run it.

```
#first make our sf object into an sp object
eu_sp <- as(eu_sf, 'Spatial')

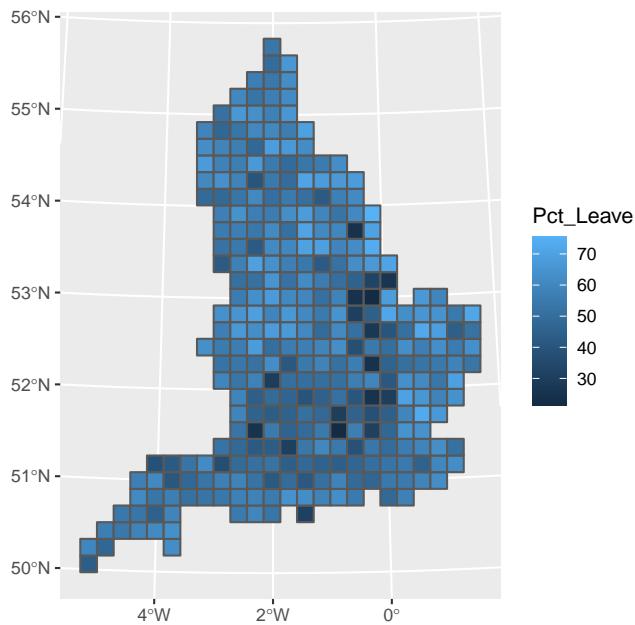
#then use the calculate_grid function. Note how we specify grid type to be "regular".
eu_reg <- calculate_grid(shape = eu_sp, grid_type = "regular", seed = 1)
#assign the polygons
eu_reg <- assign_polygons(eu_sp, eu_reg)

#now turn it back into sf object for easy ggplot plotting
eu_reg <- st_as_sf(eu_reg)
```

While you're waiting for that, you can have a read about how the polygons are assigned. The help file will tell you they use “the Hungarian algorithm”. This is from a paper called The Hungarian Method for the Assignment Problem by Harold W. Kuhn. Apparently, this method was so promising, that when Kuhn came across it referenced in a paper, even though the original paper is in Hungarian, Kuhn describes he “took out a Hungarian grammar and a large Hungarian-English dictionary and taught myself enough Hungarian to translate Egervary’s paper.” I can assure you, Hungarian is not an easy language to learn, so this must have been a most excellent theory to prompt such an act.

Now that your code has run, and you have your `eu_reg` shapefile, we can plot this!

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



Cool, eh?

5.5.1 Homework 5.3

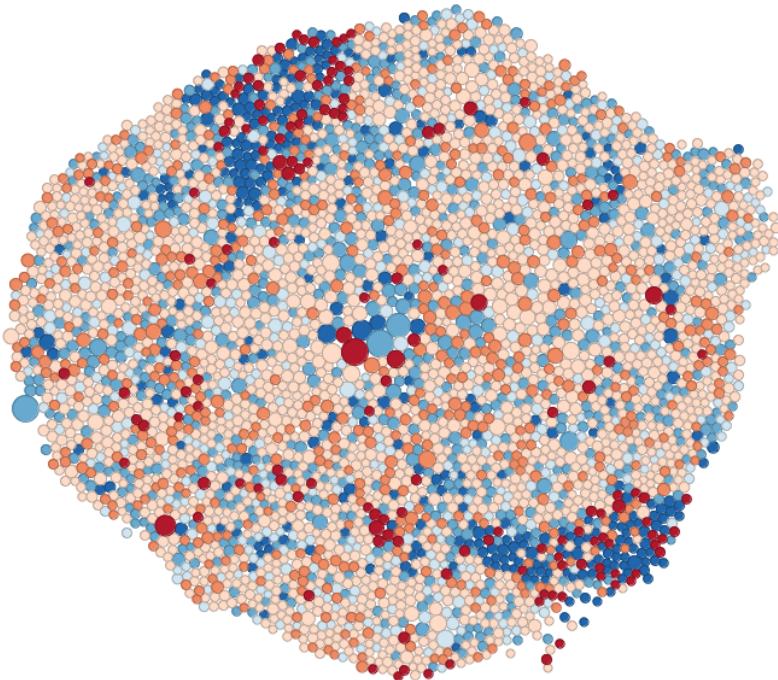
Reproduce the above grid map using hexagons instead of squares.

5.6 Cartograms

The last thing we will do today is make some cartograms! Cartogram types of maps distort reality to convey information. They resize and exaggerate any variable on an attribute value.

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 Cartogram (named after professor Danny Dorling) uses shapes like circles and rectangles to depict area. These types of cartograms make it easy to recognize patterns. For example, here is a Dorling Cartogram I made of FixMyStreet reports in London:



Now we can make our own as well, using the `cartogram` package.

```
library(cartogram)
```

Within that there is the `cartogram()` function, which takes 2 arguments, 1 - the shape file (it can be a `SpatialPolygonDataFrame` or an `sf` object), and 2 - 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

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

Again this is some labour intensive work, much like the grid making, you have some time to chill now. Maybe read up on the maths behind this transformation as well, in the paper Dougenik, J. A., Chrisman, N. R., & Niemeyer, D. R. (1985). An Algorithm To Construct Continuous Area Cartograms. In *The Professional Geographer*, 37(1), 75-81..

I do have a tip for you if you want to make sure the process does not take too long. You can set a parameter in the `cartogram` function which is the “`itermax`” parameter. This specifies the maximum number of iterations we are happy with. If you don’t 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

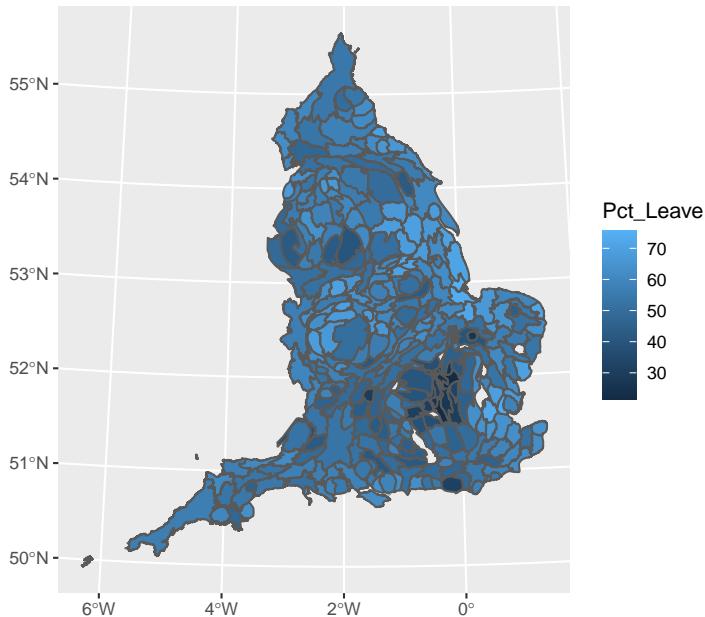
## Mean size error for iteration 3: 3.05848955329872

## Mean size error for iteration 4: 2.51480273537651

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

And if 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.

Okay that's probably enough for the day. Nice work crime mappers!

5.7 References and further reading

5.7.1 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

5.7.2 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.

5.7.3 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 6

Studying spatial point patterns

6.1 What we'll do today

We have now covered quite a bit! You've learnt about spatial objects and various formats in which they come and are stored by R, how to produce maps using a variety of packages, and also provided you with a brief introduction to common spatial operations. In what remains of the semester we are going to shift the emphasis and start focusing a bit more on spatial statistics. First we will focus on techniques that are used to explore and analyse points in a geographical space and in subsequent sessions we will cover techniques that are used to analyse spatial data when our unit of analysis are polygons (e.g., postal code areas, census areas, police beats, etc.).

We will introduce a new R package called `spatstat`, that was developed for spatial point pattern analysis and modelling. It was written by Adrian Baddeley and Rolf Turner. There is a webpage dedicated to this package. The thickest book in my library, at 810 pages, is dedicated to this package. So as you can imagine the theory and practice of spatial pattern analysis is something one could devote an entire course to. You can get a pdf document used in a course the authors of this package develop here. In our course we are only going to provide you with an introductory practical entry into this field of techniques. If this package is not installed in your machine, make sure you install it before we carry on.

```
library(sf)
library(tmap)
library(dplyr)
library(spatstat)
```

6.2 Getting the data

We will be using the crime data from Greater Manchester police we have been using so far. Let's focus on burglary in the Fallowfield area. The code below has already been explained and used in previous sessions, so we won't go over the detail again. But rather than cut and paste automatically, try to remember what each line of code is doing.

By the way, the police data for Manchester we have used in previous sessions correspond to only one month of the year. Here we are using a full year worth of data, so the data import will take a bit longer.

```
#Read a geojson file with Manchester wards (remember we learned about geojson files in
manchester_ward <- st_read("https://raw.githubusercontent.com/RUMgroup/Spatial-data-in

## Reading layer `wards` from data source `https://raw.githubusercontent.com/RUMgroup/
## Simple feature collection with 215 features and 12 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:            xmin: 351664 ymin: 381168.6 xmax: 406087.5 ymax: 421039.8
## CRS:             27700

#Create a new object that only has the fallowfield ward
df1 <- manchester_ward %>%
  filter(wd16nm == "Fallowfield")

#Change coordinate systems
fallowfield <- st_transform(df1, 4326)

#Get rid of objects we no longer need
rm(manchester_ward)
rm(df1)

#Read Greater Manchester police data
crimes <- read.csv("https://raw.githubusercontent.com/jjmedinaariza/CrimeMapping/master

burglary <- filter(crimes, crime_type == "Burglary")

#Transform the dataframe with crime information into a sf object
burglary_spatial = st_as_sf(burglary, coords = c("long", "lat"),
                            crs = 4326, agr = "constant")

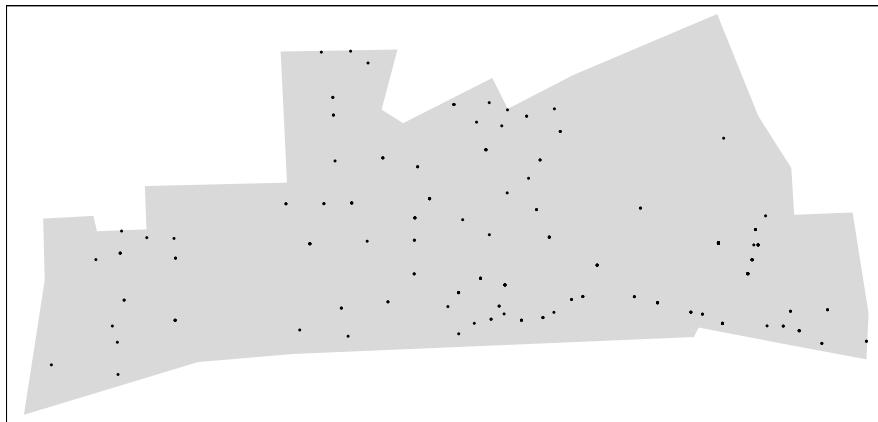
#Select only the crimes that take place within the space defined by the Ward boundaries
# intersection
bur_fal <- st_intersects(fallowfield, burglary_spatial)
```

```
## although coordinates are longitude/latitude, st_intersects assumes that they are planar

# subsetting
bur_fal <- burglary_spatial[unlist(bur_fal),]
#again remove things we don't need
rm(crimes)
rm(burglary)
```

Now we have all our data cleaned and all our files prepared. Let's see the results!

```
tm_shape(fallowfield) +
  tm_fill() +
  tm_shape(bur_fal) +
  tm_dots()
```



In the point pattern analysis literature each point is often referred to as an **event** and these events can have **marks**, attributes or characteristics that are also encoded in the data. In our spatial object one of these *marks* is the type of crime (although in this case it's of little interest since we have filtered on it).

6.3 Getting the data into spatstat: the problem with duplicates

So let's start using spatstat. The first thing we need to do is to transform our `sf` object into a `ppp` object which is how `spatstat` likes to store its point patterns. Unfortunately, `spatstat` and many other packages for analysis of spatial data precede `sf`, so the transformation is a bit awkward. Also before we do that, it is important to realise that a point pattern is defined as a series of events in a given area, or window, of observation. It is therefore extremely important to precisely define this window. In `spatstat` the function `owin()` is used to set the observation window. However, the standard function takes the coordinates of a rectangle or of a polygon from a matrix, and therefore it may be a bit tricky to use. Luckily the package `maptools` provides a way to transform a `SpatialPolygons` into an object of class `owin`, using the function `as.owin()`. Here are the steps:

First we transform the CRS of our Fallowfield polygon into projected coordinates (British National Grid) as opposed to geographic coordinates (WGS84) :

```
fallowfield_proj <- st_transform(fallowfield, 27700)
```

Then we use the `as.owin` function to define the window.

```
window <- as.owin(fallowfield_proj)
```

Now, use the `class` function and print the window object to check that this worked:

```
class(window)

## [1] "owin"

window

## window: polygonal boundary
## enclosing rectangle: [382951.5, 385869.8] x [393616.3, 394988.8] units
```

Now that we have created the window as an `owin` object let's get the points. First we will extract the coordinates from our `sf` point data into a matrix:

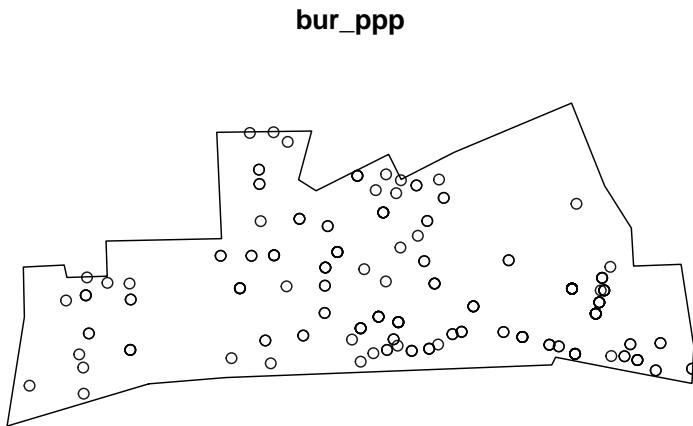
```
bur_fal <- st_transform(bur_fal, 27700) #we must transform these too to match our window in BNG
sf_bur_fal_coords <- matrix(unlist(bur_fal$geometry), ncol = 2, byrow = T)
```

Then we use the `ppp` function to create the object using the information from our matrix and the window that we created.

```
bur_ppp <- ppp(x = sf_bur_fal_coords[,1], y = sf_bur_fal_coords[,2],
                 window = window, check = T)
```

```
## Warning: data contain duplicated points
```

```
plot(bur_ppp)
```



Notice the warning message about duplicates. In spatial point pattern analysis an issue of significance is the presence of duplicates. The statistical methodology used for spatial point pattern processes is based largely on the assumption that processes are *simple*, that is, that the points cannot be coincident. That assumption may be unreasonable in many contexts (for example, the literature on repeat victimisation indeed suggests that we should expect the same households to be at a higher risk of being hit again). Even so the point (no pun intended) is that “*when the data has coincidence points, some statistical procedures will be severely affected. So it is always strongly advisable to check for duplicate points and to decide on a strategy for dealing with them if they are present*” (Baddeley et al., 2016: 60).

We can check the duplication in a `ppp` object with the following syntax:

```
any(duplicated(bur_ppp))
## [1] TRUE
```

To count the number of coincidence points we use the `multiplicity()` function. This will return a vector of integers, with one entry for each observation in our dataset, giving the number of points that are identical to the point in question (including itself).

```
multiplicity(bur_ppp)
```

If you want to know how many locations have more than one event you can use:

```
sum(multiplicity(bur_ppp) > 1)
## [1] 190
```

That's quite something. 190 points out of 223 here share coordinates.

```
tm_shape(fallowfield) +
  tm_fill() +
  tm_shape(bur_fal) +
  tm_dots(alpha=0.4, size=1)
```



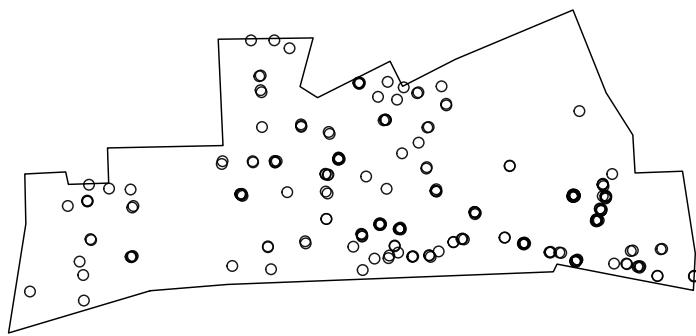
In the case of crime, as we have hinted some of this may be linked to the nature of crime itself. Hint: repeat victimisation. However, this pattern of duplication is fairly obvious across all crime categories in the police.uk website.

This is due to the way in which spatial anonymisation of police.uk data is carried out. This is done using geomasking, whereby there exist a pre-determined list of points that each crime event gets “snapped” to its nearest one. So, the coordinates provided in the open data are not the exact locations of crimes, but they come from a list of points generated for purposes of data publication. You can see the details here. This process is likely inflating the amount of duplication we observe, because each snap point might have many crimes near it, resulting in those crimes being geo-coded to the same exact location. So keep in mind when analysing and working with this data set that it is not the same as working with the real locations. If you are interested in the effects of this read the paper Lisa Tompson, Shane Johnson, Matthew Ashby, Chloe Perkins & Phillip Edwards (2015) UK open source crime data: accuracy and possibilities for research, *Cartography and Geographic Information Science*, 42:2, 97-111, DOI: 10.1080/15230406.2014.972456.

What to do about duplicates in spatial point pattern analysis is not always clear. You could simply delete the duplicates, but of course that may ignore issues such as repeat victimisation. You could also use jittering, which will add a small perturbation to the duplicate points so that they do not occupy the exact same space. Which again, may ignore things like repeat victimisation. Another alternative is to make each point “unique” and then attach the multiplicities of the points to the patterns as *marks*, as attributes of the points. Then you would need analytical techniques that take into account these marks.

If you were to be doing this for real you would want access to the real thing, not this public version of the data and then go for the latter solution suggested above. We don’t have access to the source data, so for the sake of simplicity and so that we can illustrate how `spatstat` works we will instead add some jittering to the data. The first argument for the function is the object, `retry` asks whether we want the algorithm to have another go if the jittering places a point outside the window (we want this so that we don’t lose points), and the `drop` argument is used to ensure we get a `ppp` object as a result of running this function (which we do).

```
jitter_bur <- rjitter(bur_ppp, retry=TRUE, nsim=1, drop=TRUE)
plot(jitter_bur)
```

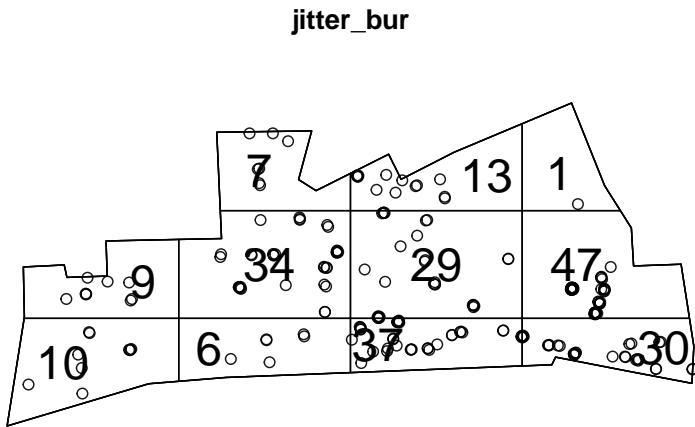
jitter_bur

Notice the difference with the original plot. Can you see how the circumferences do not overlap perfectly now?

6.4 Inspecting our data with spatstat

This package supports all kind of exploratory point pattern analysis. One example of this is **quadrant counting**. One could divide the window of observation into quadrants and count the number of points into each of these quadrants. For example, if we want four quadrants along the X axis and 3 along the Y axis we could used those parameters in the `quadratcount()` function. Then we just use standard plotting functions from R base.

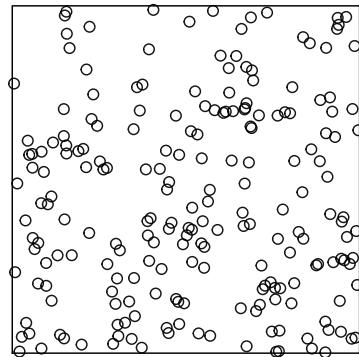
```
Q <- quadratcount(jitter_bur, nx = 4, ny = 3)
plot(jitter_bur)
plot(Q, add = TRUE, cex = 2)
```



In the video lectures for this week, Luc Anselin introduced the notion of **complete spatial randomness** (CSR for short). When we look at a point pattern process the first step in the process is to ask whether it has been generated in a random manner. Under CSR, points are independent of each other and have the same propensity to be found at any location. We can generate data that conform to complete spatial randomness using the *rpoispp()* function. The r at the beginning is used to denote we are simulating data (you will see this is common in R) and we are using a Poisson point process, a good probability distribution for these purposes. Let's generate 223 points in a random manner:

```
plot(rpoispp(223))
```

rpoispp(223)



You will notice that the points in a homogeneous Poisson process are not ‘uniformly spread’: there are empty gaps and clusters of points. Run the previous command a few times. You will see the map generated is different each time.

In classical literature, the *homogeneous Poisson process* (CSR) is usually taken as the appropriate ‘null’ model for a point pattern. Our basic task in analysing a point pattern is to find evidence against CSR. We can run a Chi Square test to check this. So, for example:

```
quadrat.test(jitter_bur, nx = 3, ny = 2)

##
##  Chi-squared test of CSR using quadrat counts
##
##  data: jitter_bur
##  X2 = 111.38, df = 5, p-value < 2.2e-16
##  alternative hypothesis: two.sided
##
##  Quadrats: 6 tiles (irregular windows)
```

Observing the results we see that the p value is well below conventional standards for rejection of the null hypothesis. Observing our data of burglary in Fallowfield would be extremely rare if the null hypothesis was true. We can then conclude that the burglary data is not randomly distributed in the observed space. But no cop nor criminologist would really question this. They would rarely be surprised by your findings! We do know that crime is not randomly distributed in space.

6.5 Density estimates

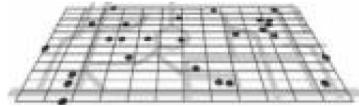
In the presentations by Luc Anselin and the recommended reading materials we introduced the notion of density maps. **Kernel density estimation** involves applying a function (known as a “kernel”) to each data point, which averages the location of that point with respect to the location of other data points. The surface that results from this model allows us to produce **isarithmic maps**, also referred to in common parlour as heatmaps. Beware though, cartographers really dislike this common parlour. We saw this kind of maps when covering the various types of thematic maps.

Kernel density estimation maps are very popular among crime analysts. According to Chainey (2012), 9 out of 10 intelligence professionals prefer it to other techniques for hot spot analysis. As compared to visualisations of crime that relies on point maps or thematic maps of geographic administrative units (such as LSOAs), kernel density estimation maps are considered best for location, size, shape and orientation of the hotspot (Chainey, 2012). Spencer Chainey and his colleagues (2008) have also suggested that this method produces some of the best prediction accuracy. The areas identified as hotspots by KDE (using historical data) tend to be the ones that better identify the areas that will have high levels of crime in the future. Yet, producing these maps (as with any map, really) requires you to take a number of decisions that will significantly affect the resulting product and the conveyed message. Like any other data visualisation technique they can be powerful, but they have to be handled with great care.

Essentially this method uses a statistical technique (kernel density estimation) to generate a smooth continuous surface aiming to represent the density or volume of crimes across the target area. The technique, in one of its implementations (quartic kernel), is described in this way by Eck and colleagues (2005):

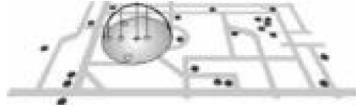
- “*a fine grid is generated over the point distribution;*
- *a moving three-dimensional function of a specified radius visits each cell and calculates weights for each point within the kernel’s radius. Points closer to the centre will receive a higher weight, and therefore contribute more to the cell’s total density value;*
- *and final grid cell values are calculated by summing the values of all kernel estimates for each location”*

Exhibit 11. A fine grid is placed over the area covered by the crime points



Source: Ratcliffe, 1999a.

Exhibit 12. A search radius (or bandwidth) is then selected



Source: Ratcliffe, 1999a.

Note: Within the bandwidth, intensity values for each point are calculated. Points are weighted, so that incidents closer to the center contribute a higher value to the cell's intensity value.

(Reproduced from Eck et al. 2012)

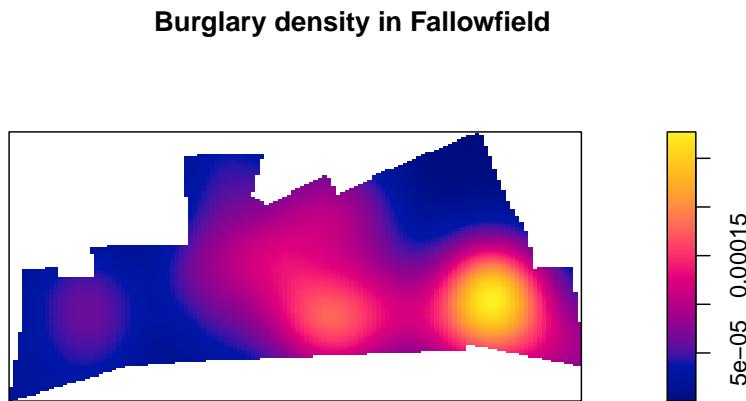
The values that we attribute to the cells in crime mapping will typically refer to the number of crimes within the area's unit of measurement. We don't have the time to elaborate further on this technique now, but if you did the required reading you should have at least a notion of how this works.

Let's produce one of this density maps:

```
ds <- density(jitter_bur)
class(ds)
```

```
## [1] "im"
```

```
plot(ds, main='Burglary density in Fallowfield')
```



The density function is estimating a kernel density estimate. Density is nothing but the number of points per unit area. This method computes the intensity continuously across the study area and the object returns a raster image.

To perform this analysis in R we need to define the **bandwidth** of the density estimation, which basically determines the area of influence of the estimation. There is no general rule to determine the correct bandwidth; generally speaking if the bandwidth is too small the estimate is too noisy, while if bandwidth is too high the estimate may miss crucial elements of the point pattern due to oversmoothing (Scott, 2009).

The key argument to pass to the density method for point pattern objects is **sigma=**, which determines the bandwidth of the kernel. In spatstat the functions **bw.diggle()**, **bw.ppl()**, and **bw.scott()** can be used to estimate the bandwidth according to difference methods. The helpfiles recommend the use of the first two. These functions run algorithms that aim to select an appropriate bandwidth.

```
bw.diggle(jitter_bur)
```

```
##     sigma
## 3.693017
```

```
bw.ppl(jitter_bur)
```

```

##      sigma
## 37.95565

bw.scott(jitter_bur)

##  sigma.x  sigma.y
## 270.32769 93.84499

```

You can see the Diggle algorithm gives you the narrower bandwith. We can test how they work with our dataset using the following code:

```

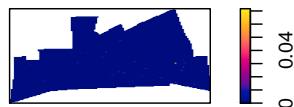
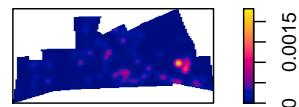
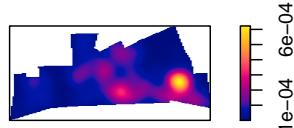
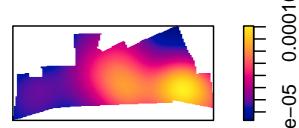
par(mfrow=c(2,2))
plot(density.ppp(jitter_bur, sigma = bw.diggle(jitter_bur),edge=T),
     main = paste("h = 0.000003"))

plot(density.ppp(jitter_bur, sigma = bw.ppl(jitter_bur),edge=T),
     main=paste("h =0.0005"))

plot(density.ppp(jitter_bur, sigma = bw.scott(jitter_bur)[2],edge=T),
     main=paste("h = 0.0008"))

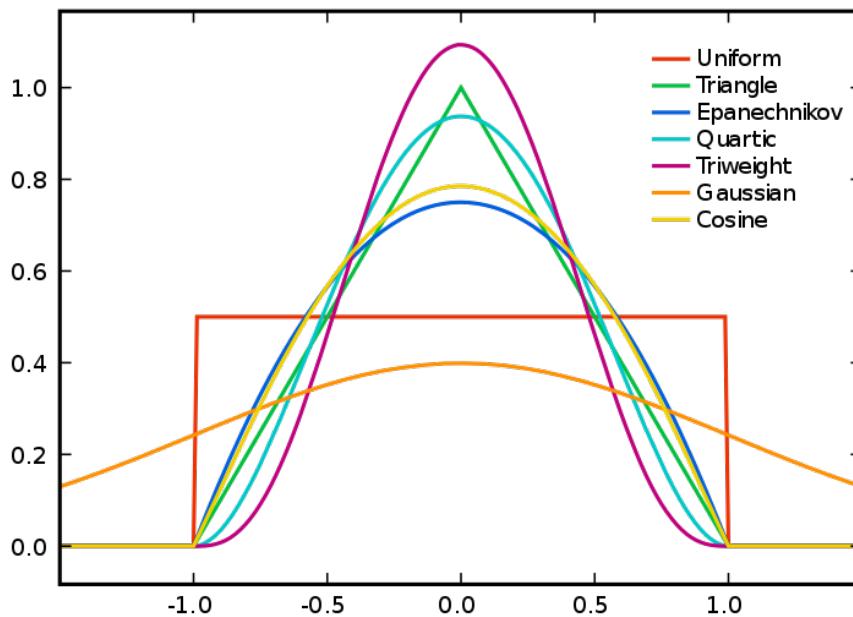
plot(density.ppp(jitter_bur, sigma = bw.scott(jitter_bur)[1],edge=T),
     main=paste("h = 0.004"))

```

 $h = 0.000003$  **$h = 0.0005$**  **$h = 0.0008$**  **$h = 0.004$** 

Baddeley et al (2016) suggest the use of the `bw.ppl()` algorithm because in their experience it tends to produce the more appropriate values when the pattern consists predominantly of tight clusters. But they also insist that if your purpose is to detect a single tight cluster in the midst of random noise then the `bw.diggle()` method seems to work best.

Apart from selecting the bandwidth we also need to specify the particular kernel we will use. In density estimation there are different types of kernel (as illustrated below):



Source: wikipedia

You can read more about kernel types in the Wikipedia entry. This relates to the type of kernel drawn around each point in the process of counting points around each point. The use of these functions will result in slightly different estimations. They relate to the way we weight points within the radius: “*The normal distribution weighs all points in the study area, though near points are weighted more highly than distant points. The other four techniques use a circumscribed circle around the grid cell. The uniform distribution weighs all points within the circle equally. The quartic function weighs near points more than far points, but the fall off is gradual. The triangular function weighs near points more than far points within the circle, but the fall off is more rapid. Finally, the negative exponential weighs near points much more highly than far points within the circle and the decay is very rapid.*” (Levine, 2013: 10.10).

Which one to use? Levine (2013) produces the following guidance: “*The use of any of one of these depends on how much the user wants to weigh near points*

relative to far points. Using a kernel function which has a big difference in the weights of near versus far points (e.g., the negative exponential or the triangular) tends to produce finer variations within the surface than functions which weight more evenly (e.g., the normal distribution, the quartic, or the uniform); these latter ones tend to smooth the distribution more. However, Silverman (1986) has argued that it does not make that much difference as long as the kernel is symmetrical. Chainey (2013) suggest that in his experience most crime mappers prefer the quartic function, since it applies greater weight to crimes closer to the centre of the grid. The authors of the CrimeStat workbook (Smith and Bruce, 2008), on the other hand, suggest that the choice of the kernel should be based in our theoretical understanding of the data generating mechanisms. By this they mean that the processes behind spatial autocorrelation may be different according to various crime patterns and that this is something that we may want to take into account when selecting a particular function. They provide a table with some examples that may help you to understand what they mean:

Table 6-1: interpolation method and interval possibilities for different types of crime

Incident Type	Interval	Interpolation Method	Reasoning
Residential burglaries	1 mile	Moderately dispersed: quartic or uniform	Some burglars choose particular houses, but most cruise neighborhoods looking for likely targets. A housebreak in any part of a neighborhood transfers risk to the rest of the neighborhood.
Domestic violence	0.1 mile	Tightly focused: negative exponential	Domestic violence occurs among specific individuals and families. Incidents at one location do not have much chance of being contagious in the surrounding area.
Commercial robberies	2 miles	Focused: triangular or negative exponential	A commercial robber probably chooses to strike in a commercial area, and then looks for preferred targets (banks, convenience stores) within that area. The wide area may thus be at some risk, but the brunt of the weight should remain with the particular target that has already been struck.
Thefts from vehicles	0.25 mile	Dispersed: uniform	If a parking lot experiences a lot of thefts from vehicles, your GIS will probably geocode them at the center of the parcel. This method ensures that the risk disperses evenly across the parcel and part of the surrounding area (which probably makes sense)—but not too far, since we know that parking lots tend to be hot spots for specific reasons.

(Source: Smith and Bruce, 2008.)

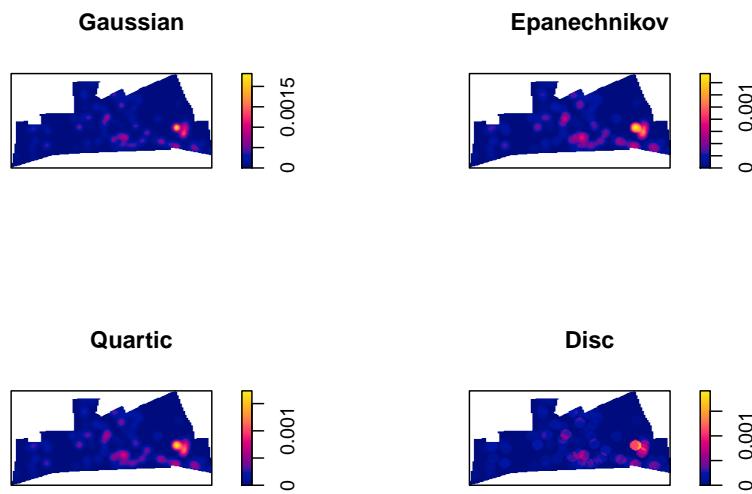
The default kernel in `density.ppp()` is the `gaussian`. But there are other options. We can use the `epanechnikov`, `quartic` or `disc`. There are also further options for customisation. We can compare these kernels:

```
par(mfrow=c(2,2))
plot(density.ppp(jitter_bur, sigma = bw.ppl(jitter_bur), edge=T),
     main=paste("Gaussian"))
plot(density.ppp(jitter_bur, kernel = "epanechnikov", sigma = bw.ppl(jitter_bur), edge=T),
     main=paste("Epanechnikov"))
plot(density.ppp(jitter_bur, kernel = "quartic", sigma = bw.ppl(jitter_bur), edge=T),
     main=paste("Quartic"))
plot(density.ppp(jitter_bur, kernel = "disc", sigma = bw.ppl(jitter_bur), edge=T),
     main=paste("Disc"))
```

```

main=paste("Epanechnikov"))
plot(density.ppp(jitter_bur, kernel = "quartic", sigma = bw.ppl(jitter_bur),edge=T),
     main=paste("Quartic"))
plot(density.ppp(jitter_bur, kernel = "disc", sigma = bw.ppl(jitter_bur),edge=T),
     main=paste("Disc"))

```



When reading these maps you need to understand you are only looking at counts of crime in a smooth surface. Nothing more, nothing less. Unlike with choropleth maps we are not normalising the data. We are simply showing the areas where there is more crime, but we are not adjusting for anything (like number of people in the area, or number of houses to burgle). So, it is important you keep this in the back of your mind. As this comic suggests you may end up reading too much into it if you don't remember this. There are ways to produce density maps adjusting for a second variable, such as population size, but we do not have the time to cover this.

There are also general considerations to keep in mind. Hot spots of crime are simply a convenient perceptual construct. As Ned Levine (2013: 7.1) highlights “*Hot spots do not exist in reality, but are areas where there is sufficient clustering of certain activities (in this case, crime) such that they get labeled such. There is not a border around these incidents, but a gradient where people draw an imaginary line to indicate the location at which the hot spot starts.*” Equally, there is not a unique solution to the identification of hot spots. Different techniques and algorithms will give you different answers. As Levine (2013: 7.7) emphasises: “*It would be very naive to expect that a single technique can reveal*

the existence of hot spots in a jurisdiction that are unequivocally clear. In most cases, analysts are not sure why there are hot spots in the first place. Until that is solved, it would be unreasonable to expect a mathematical or statistical routine to solve that problem.” So, as with most data analysis exercises one has to try different approaches and use professional judgement to select a particular representation that may work best for a particular use. Equally, we should not reify what we produce and, instead, take the maps as a starting point for trying to understand the underlying patterns that are being revealed. Critically you want to try several different methods. You will be more persuaded a location is a hot spot if several methods for hot spot analysis point to the same location.

6.6 Adding some context

Often it is convenient to use a basemap to provide context. In order to do that we first need to turn the image object generated by the `spatstat` package into a raster object, a more generic format for raster image used in R. Remember rasters from the first week? Now we finally get to use them a bit!

```
library(raster)
dmap1 <- density.ppp(jitter_bur, sigma = bw.ppl(jitter_bur), edge=T)
r1 <- raster(dmap1)
#remove very low density values
r1[r1 < 0.0001] <- NA
class(r1)

## [1] "RasterLayer"
## attr(,"package")
## [1] "raster"
```

Now that we have the raster we can add it to a basemap.

Two-dimensional `RasterLayer` objects (from the `raster` package) can be turned into images and added to `Leaflet` maps using the `addRasterImage()` function.

The `addRasterImage()` function works by projecting the `RasterLayer` object to EPSG:3857 and encoding each cell to an RGBA color, to produce a PNG image. That image is then embedded in the map widget.

It's important that the `RasterLayer` object is tagged with a proper coordinate reference system. Many raster files contain this information, but some do not. Here is how you'd tag a raster layer object “`r1`” which contains WGS84 data:

```
library(leaflet)

#make sure we have right CRS, which in this case is British National Grid
```

```

epsg27700 <- "+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000 +ellps=air"
crs(r1) <- sp::CRS(epsg27700)

#we also create a colour palet
pal <- colorNumeric(c("#0C2C84", "#41B6C4", "#FFFFCC"), values(r1),
na.color = "transparent")

#and then make map!
leaflet() %>%
  addTiles() %>%
  addRasterImage(r1, colors = pal, opacity = 0.8) %>%
  addLegend(pal = pal, values = values(r1),
            title = "Burglary map")

```

And there you have it. Perhaps those familiar with Fallowfield have some guesses as to what may be going on there?

6.6.1 Homework 1

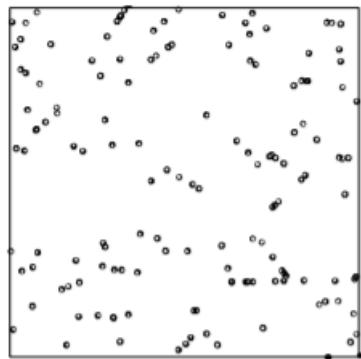
Ok, so see if you can do something like what we have done today, but for violent crime in the city centre. Produce the density estimates and then plot the density plot. In addition add a layer of points with the licenced premises we looked at last week.

6.6.2 Homework 2

Produce a kernel density estimate for burglary across the whole of the city. Where is burglary more concentrated?

6.7 Spatial point patterns along networks

Have a look at this maps. Can we say that the spatial point process is random here? Can you identify the areas where we have hotspots of crime? Think about these questions for a little while.

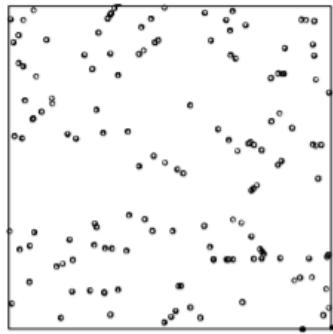


(a)

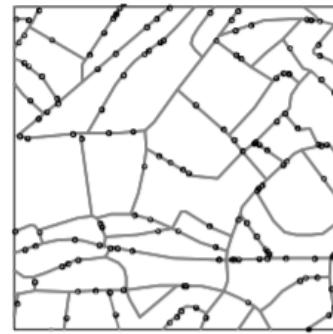
(Source: Okabe and Sugihara, 2012)

Ok, so most likely you concluded that the process wasn't random, which it isn't in truth. It is also likely that you identified a number of potential hotspots?

Now, look at the two maps below:



(a)



(b)

(Source: Okabe and Sugihara, 2012)

We are representing the same spatial point pattern process in each of them. But we do have additional information in map B. We now know the street layout. The structure we observed in the map is accounted by the street layout. So what look like a non random spatial point process when we considered the full two dimensional space, now looks less random when we realise that the points can only appear alongside the linear network.

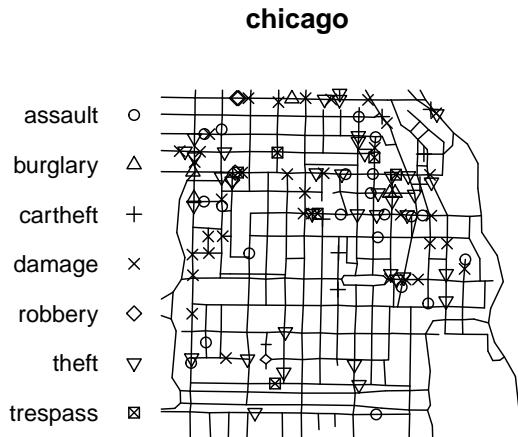
This problem is common in criminal justice applications. Crime is geocoded alongside a linear street network. Even if in physical space crime can take place along a spatial continuum, once crime is geocoded it will only be possible alongside the street network used for the geocoding process.

For exploring this kind of spatial point pattern processes along networks we need special techniques. Some researchers have developed special applications,

such as SANET. The **spatstat** package also provides some functionality for this kind of data structures.

In **spatstat** a point pattern on a linear network is represented by an object of class **lpp**. The functions **lpp()** and **as.lpp()** convert raw data into an object of class **lpp** (but they require a specification of the underlying network of lines, which is represented by an object of class **linnet**). For simplicity and illustration purposes we will use the **chicago** dataset that is distributed as part of the **spatstat** package. The **chicago** data is of class **lpp** and contains information on crime in an area of Chicago.

```
data("chicago")
plot(chicago)
```



```
summary(chicago)
```

```
## Multitype point pattern on linear network
## 116 points
## Linear network with 338 vertices and 503 lines
## Total length 31150.21 feet
## Average intensity 0.003723891 points per foot
## Types of points:
##           frequency proportion    intensity
## assault      21 0.18103450 0.0006741528
```

```

## burglary      5 0.04310345 0.0001605126
## cartheft     7 0.06034483 0.0002247176
## damage       35 0.30172410 0.0011235880
## robbery      4 0.03448276 0.0001284100
## theft        38 0.32758620 0.0012198950
## trespass     6 0.05172414 0.0001926151
## Enclosing window: rectangle = [0.3894, 1281.9863] x [153.1035, 1276.5602] feet

```

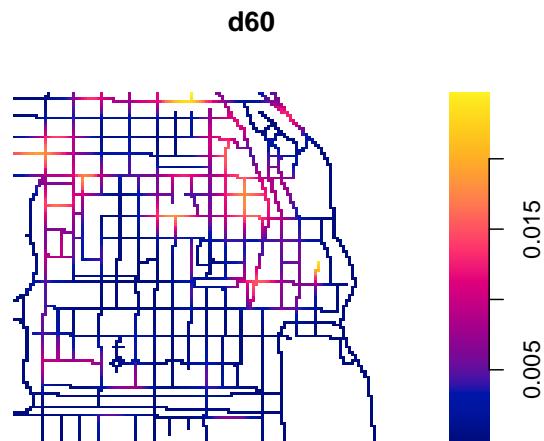
An `lpp` object contains the linear network information, the spatial coordinates of the data points, and any number of columns of *marks* (in this case the mark is telling us the type of crime we are dealing with). It also contains the local coordinates `seg` and `tp` for the data points. The local coordinate `seg` is an integer identifying the particular street segment the data point is located in. A segment is each of the sections of a street between two vertices (marking the intersection with another segment). The local coordinate `tp` is a real number between 0 and 1 indicating the position of the point within the segment: `tp=0` corresponds to the first endpoint and `tp=1` correspond to the second endpoint.

The visual inspection of the map suggest that the intensity of crime along the network is not spatially uniform. Crime seems to be concentrated in particular segments. Like we did before we can estimate the density of data points along the networks using Kernel estimation (with the `density.lpp()` function), only now we only look at the street segments (rather than areas of the space that are outside the segments). The authors of the package are planning to introduce methods for automatic bandwidth selection but for now this is not possible, so we have to select a bandwidth. We could for example select 60 feet.

```
d60 <- density.lpp(unmark(chicago), 60)
```

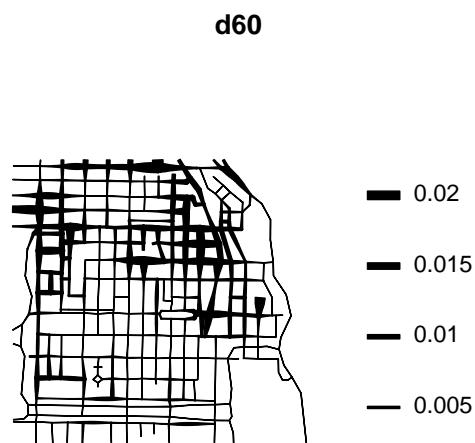
We use `unmark()` to ignore the fact the data points are marked (that is they provide marks with information, in this case about the crime type). By using `unmark()` in this example we will run density estimation for all crimes (rather than by type of crime). We can see the results below:

```
plot(d60)
```



If rather than colour you want to use the thickness of the street segment to identify hotspots you would need to modify the code as shown below:

```
plot(d60, style="width", adjust=2.5)
```



This is very important for crime research, as offending will be constrained by all sorts of networks. Traditionally, hotspot analysis has been directed at crimes that are assumed to be situated across an infinite homogeneous environment (e.g., theft of motor vehicle), we must develop an increased awareness of perceptible geographical restrictions. There has been increasing recognition in recent years that the spatial existence of many phenomena is constrained by networks.

These networks may be roads or rail networks, but there may be many more:

Environmental crimes could exist along waterways such as streams, canals, and rivers; and thefts of metal could occur along utility networks such as pipelines. Those sociologically inclined might be able to offer more examples in the way of interpersonal networks.

- Tompson, Lisa, Henry Partridge, and Naomi Shepherd. “Hot routes: Developing a new technique for the spatial analysis of crime.” *Crime Mapping: A Journal of Research and Practice* 1, no. 1 (2009): 77-96.

While sometimes there may be issues with linking points to routes due to problems such as bad geocoding, as we had discusses in great detail in week 4, there are obvious advantages to considering crime as distributed along networks, rather than continuous space.

Chapter 7

Global and local spatial autocorrelation

This session we begin to explore the analysis of local spatial autocorrelation statistics. Spatial autocorrelation is the correlation among data values, strictly due to the relative location proximity of the objects that the data refer to. Remember in the second week when we spoke about Tobler's first law of geography - "everything is related to everything else, but near things are more related than distant things"? Spatial autocorrelation is the measure of this *correlation* between near things. If correlation is not a familiar term, there is a recommended reading for you on blackboard to refresh your memory.

We'll be making use of the following packages:

```
library(sf)
library(tmap)
library(dplyr)
library(sp)
library(spdep)
```

7.1 Get data

So, let's start by getting some data. We are going to take some of the data from past weeks. In getting the data ready you will have one more opportunity to practice how to read data into R but also how to perform some basic spatial checks, transformations and operations. It may seem like you have already done some of this stuff. But that's the point: to force you to practice. The more you do this stuff, the easier it will be and -trust us- eventually things will click and become second nature. First let's get the LSOA boundary data.

```
shp_name <- "data/BoundaryData/england_lsoa_2011.shp"
manchester_lsoa <- st_read(shp_name)

## Reading layer `england_lsoa_2011` from data source `/Users/reka/Dropbox (The University of Manchester) - Google Drive/My Drive/Geography/Geospatial Data/Shapefiles/England/england_lsoa_2011.shp'
## Simple feature collection with 282 features and 3 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:            xmin: 378833.2 ymin: 382620.6 xmax: 390350.2 ymax: 405357.1
## CRS:             27700
```

Now check the coordinate system.

```
st_crs(manchester_lsoa)
```

```
## Coordinate Reference System:
##   User input: 27700
##   wkt:
##     PROJCS["OSGB 1936 / British National Grid",
##           GEOGCS["OSGB 1936",
##                  DATUM["OSGB_1936",
##                        SPHEROID["Airy 1830",6377563.396,299.3249646,
##                                AUTHORITY["EPSG","7001"]]],
##                  TOWGS84[446.448,-125.157,542.06,0.15,0.247,0.842,-20.489],
##                                AUTHORITY["EPSG","6277"]],
##                  PRIMEM["Greenwich",0,
##                        AUTHORITY["EPSG","8901"]]],
##                  UNIT["degree",0.0174532925199433,
##                        AUTHORITY["EPSG","9122"]],
##                        AUTHORITY["EPSG","4277"]],
##                  PROJECTION["Transverse_Mercator"],
##                  PARAMETER["latitude_of_origin",49],
##                  PARAMETER["central_meridian",-2],
##                  PARAMETER["scale_factor",0.9996012717],
##                  PARAMETER["false_easting",400000],
##                  PARAMETER["false_northing",-100000],
##                  UNIT["metre",1,
##                        AUTHORITY["EPSG","9001"]]],
##                  AXIS["Easting",EAST],
##                  AXIS["Northing",NORTH],
##                  AUTHORITY["EPSG","27700"]]
```

There is no EPSG code assigned, but notice the datum is given for BNG. Let's address this.

```
lsoa_WGS84 <- st_transform(manchester_lsoa, 4326)
st_crs(lsoa_WGS84)

## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
##     GEOGCS["WGS 84",
##       DATUM["WGS_1984",
##         SPHEROID["WGS 84",6378137,298.257223563,
##           AUTHORITY["EPSG","7030"]],
##         AUTHORITY["EPSG","6326"]],
##       PRIMEM["Greenwich",0,
##         AUTHORITY["EPSG","8901"]],
##       UNIT["degree",0.0174532925199433,
##         AUTHORITY["EPSG","9122"]],
##       AUTHORITY["EPSG","4326"]]
```

Now we have this WGS84 shapefile, so we can plot this to make sure all looks well, and then remove the old object which we no longer need.

```
plot(st_geometry(lsoa_WGS84))
```



```
rm(manchester_lsoa)
```

Let's add the burglary data from Greater Manchester. We have practiced this code in previous sessions so we won't go over it on detail again, but try to remember and understand what each line of code rather than blindly cut and paste. If you don't understand what each of these lines of codes is doing, raise your hand to call us over to help.

```
#Read into R
crimes <- read.csv("https://raw.githubusercontent.com/jjmedinaariza/CrimeMapping/master/data/crimes.csv")
#Filter out to select burglary
burglary <- filter(crimes, crime_type == "Burglary")
#Transform into spatial object
burglary_spatial = st_as_sf(burglary, coords = c("long", "lat"),
                            crs = 4326, agr = "constant")
#Remove redundant non spatial burglary object
rm(burglary)
rm(crimes)
#Select burglaries that intersect with the Manchester city LSOA map.
bur_mc <- st_intersects(lsoa_WGS84, burglary_spatial)
```

```
## although coordinates are longitude/latitude, st_intersects assumes that they are pl
```

```
bur_mc <- burglary_spatial[unlist(bur_mc),]
#Check results
plot(st_geometry(bur_mc))
```



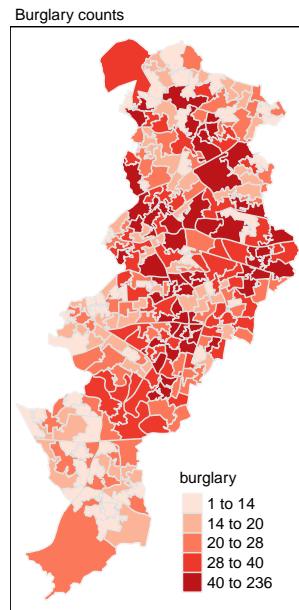
```
#Remove redundant objects
rm(burglary_spatial)
```

We now have the burglaries data, let's now count how many burglaries there are within each LSOA polygon. This is a point in polygon operation that we covered in week 4 when we counted the number of violent crimes in the buffers of the licenced premises for example. If the code or the notion does not make much sense to you make sure you review the relevant session from week 4.

```
#Point in polygon spatial operation (be patient this can take time)
burglaries_per_lsoa <- bur_mc %>%
  st_join(lsoa_WGS84, ., left = FALSE) %>%
  count(code)

## although coordinates are longitude/latitude, st_intersects assumes that they are planar

#Let's rename the column with the count of burglaries (n) into something more meaningful
burglaries_per_lsoa <- rename(burglaries_per_lsoa, burglary = n)
#Plot with tmap
tm_shape(burglaries_per_lsoa) +
  tm_fill("burglary", style = "quantile", palette = "Reds") +
  tm_borders(alpha = 0.1) +
  tm_layout(main.title = "Burglary counts", main.title.size = 0.7 ,
            legend.position = c("right", "bottom"), legend.title.size = 0.8)
```



Do you see any patterns? Are burglaries randomly spread around the map? Or would you say that areas that are closer to each other tend to be more alike? Is there evidence of clustering? Do burglaries seem to appear in certain pockets of the map?

In this session we are going to discuss ways in which you can quantify the answer to this question. We will discuss measures of **global spatial autocorrelation**, which essentially aim to answer the degree to which areas that are near each other tend to be more alike. We say global because we are interested in the degree of clustering not on the location of the clusters. Later we will also cover techniques to identify local clusters of autocorrelation, but for now we will focus in quantifying whether areas are (on average) alike their neighbours.

7.2 What is a neighbour?

Previously I asked whether areas are alike their neighbours or to areas that are close. But what is a neighbour? Or what do we mean by close? How can one define a set of neighbours for each area? If we want to know if what we measure in a particular area is similar to what happens on its neighbouring areas, we need to establish what we mean by a neighbour.

There are various ways of defining a neighbour. We can say that two areas are neighbours if they share boundaries, if they are next to each other. In this case we talk of neighbours by **contiguity**. By contiguous you can, at the same time, mean all areas that share common boundaries (what we call contiguity

using the **rook** criteria, like in chess) or areas that share common boundaries and common *corners*, that is, that have any point in common (and we call this contiguity using the **queen** criteria).

When defining neighbours by contiguity we may also specify the *order* of contiguity. **First order contiguity** means that we are focusing on areas immediately contiguous. **Second order** means that we consider neighbours only those areas that are immediately contiguous to our first order neighbours (only the yellow areas in the figure below) and you could go on and on. Look at the graphic below for clarification:

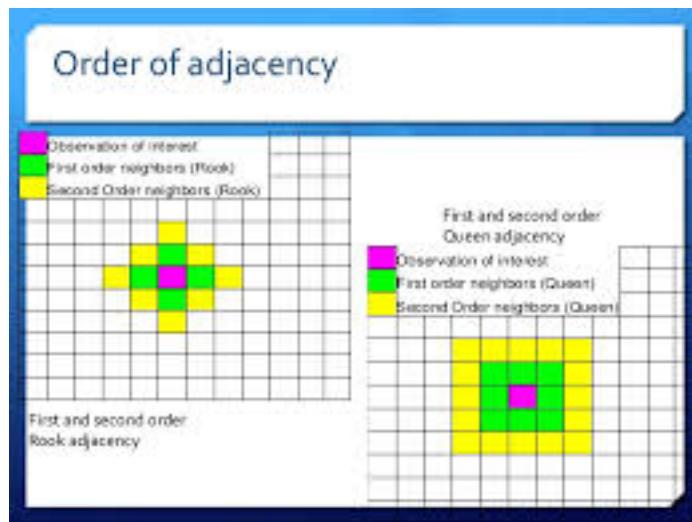


Figure 7.1: Figure 1

Source

Alternatively we may define neighbours by **distance**. You can consider neighbours those areas that distant-wise are close to each other (regardless of whether boundaries are shared). In other words, areas will be defined as neighbours if they are within a specified radius.

In sum, adjacency is an important concept in some spatial analysis. In some cases objects are considered adjacent when they “touch”, e.g. neighbouring countries. Contiguity measures tend to be more common when studying areas. It can also be based on distance. This is the most common approach when analysing point data, but can also be relevant when studying areas.

7.3 Putting ‘neighbourhood’ in our analysis - constructing a spatial weight matrix

You will come across the term **spatial weight matrix** at some point or, using mathematical notation, “W”. Essentially the spatial weight matrix is a n by n matrix with ones and zeroes (in the case of contiguity based definitions) identifying if any two observations are neighbours. So you can think of the spatial weight matrix as the new data table that we are constructing with our definition of neighbours (whichever definition that is).

So how do you build such a matrix with R? Well, let’s turn to that. But to make things a bit simpler, let’s focus not on the whole of Manchester, but just in the LSOAs within the city centre. Calculating a spatial weights matrix is a computationally intensive process, which means it takes a long time. The larger area you have (which will have more LSOAs) the longer this will take.

We will use familiar code to clip the spatial object with the counts of burglaries to only those that intersect with the City Centre ward. Again, we have covered this code elsewhere, so we won’t explain here in detail. But don’t just cut and paste, if there is anything in this code you don’t fully understand you are expected to ask us.

```
#Read a geojson file with Manchester wards
manchester_ward <- st_read("https://raw.githubusercontent.com/RUMgroup/Spatial-data-in-# Create a new object that only has the city centre ward
df1 <- manchester_ward %>%
  filter(wd16nm == "City Centre")
#Change coordinate systems
cc_ward <- st_transform(df1, 4326)
#Check if they match those of the imd_gm object
st_crs(cc_ward) == st_crs(burglaries_per_lsoa)

## [1] TRUE

#Get rid of objects we no longer need
rm(manchester_ward)
```

```

rm(df1)
#Intersect
cc_intersects <- st_intersects(cc_ward, burglaries_per_lsoa)

## although coordinates are longitude/latitude, st_intersects assumes that they are planar

cc_burglary <- burglaries_per_lsoa[unlist(cc_intersects),]
#Plot with tmap
tmap_mode("view")

## tmap mode set to interactive viewing

tm_shape(cc_burglary) +
  tm_fill("burglary", style = "quantile", palette = "Reds", id="code") +
  tm_borders() +
  tm_layout(main.title = "Burglary counts", main.title.size = 0.7 ,
            legend.position = c("right", "top"), legend.title.size = 0.8)

## legend.postion is used for plot mode. Use view.legend.position in tm_view to set the legend po

```

So now we have a new spatial object `cc_burglary` with the 23 LSOA units that compose the City Centre of Manchester. By focusing in a smaller subset of areas we can understand perhaps a bit better what comes next. But again we carry on. Do you perceive here some degree of spatial autocorrelation?

7.3.1 Homework 1

The id argument in the tm_fill ensures that when you click over any of the areas you get not only the count of burglaries in that LSOA (the quantity we are mapping) gets displayed within a bubble, but you also get to see the code that identifies that LSOA.

Move your cursor over the LSOA covering the West of Beswick. You will see this area had 58 burglaries in 2017 and that the LSOA identifier is E01033688. Using the rook criteria identify the first order neighbors of this LSOA. List their identifiers. Are things different if we use the queen criteria? If so, how does it change? Now. Think and think hard about what the lecture by Luc Anseling discussed. Have you identified all the neighbours of this area? (there are multiple ways of answering this question, just make sure you reason your answer).

7.4 Creating a list of neighbours

It would be very, very tedious having to identify the neighbours of all the areas in our study area by hand, in the way we have done in homework 1. That's why we love computers. We can automate tedious work so that they do it and we have more time to do fun stuff. We can use code to get the computer to establish what areas are next to each other (if we are using a contiguity based definition of being a neighbour).

We have already discussed how the `sf` package is new kid in town and although package developers are quickly trying to adapt their packages to work with `sf`, many of the existing spatial packages still expect spatial objects. So to illustrate the concepts in this week session we are first going to turn our `sf` object into spatial objects, so we can make use of the functions from the `sp` package that allow us to create a list of neighbours.

```
#We coerce the sf object into a new sp object that we are calling bur_ccsp (remember n
bur_ccsp <- as(cc_burglary, "Spatial")
```

Let's also change the code variable to be a character vector (rather than a factor) as we will need it to be in this format later on:

```
cc_burglary$lsoa_code <- as.character(cc_burglary$code)
```

In order to identify neighbours we will use the `poly2nb()` function from the `spdep` package that we loaded at the beginning of our session. The `spdep` package provides basic functions for building neighbour lists and spatial weights, tests for spatial autocorrelation for areal data like Moran's I (more on this below), and functions for fitting spatial regression models.

This function builds a neighbours list based on regions with contiguous boundaries. If you look at the documentation you will see that you can pass a “queen” argument that takes TRUE or FALSE as options. If you do not specify this argument the default is set to true, that is, if you don't specify `queen = FALSE` this function will return a list of first order neighbours using the Queen criteria.

```
w <- poly2nb(bur_ccsp, row.names=bur_ccsp$lsoa_code)
class(w)
```

```
## [1] "nb"
```

This has created a `nb`, neighbour list object. We can get some idea of what's there if we ask for a summary.

```
summary(w)
```

```
## Neighbour list object:
## Number of regions: 23
## Number of nonzero links: 100
## Percentage nonzero weights: 18.90359
## Average number of links: 4.347826
## Link number distribution:
##
## 2 3 4 5 6 7 8
## 3 4 6 5 3 1 1
## 3 least connected regions:
## 16 17 22 with 2 links
## 1 most connected region:
## 9 with 8 links
```

This is basically telling us that using this criteria each LSOA polygon has an average of 4.3 neighbours (when we just focus on the city centre) and that all areas have some neighbours (there is no islands). The link number distribution gives you the number of links (neighbours) per area. So here we have 3 polygons with 2 neighbours, 3 with 3, 6 with 4, and so on. The summary function here also identifies the areas sitting at both extreme of the distribution.

For more details we can look at the structure of w.

```
str(w)
```

```
## List of 23
## $ : int [1:4] 2 6 11 23
## $ : int [1:5] 1 3 4 6 8
## $ : int [1:5] 2 5 6 8 12
## $ : int [1:3] 2 8 21
## $ : int [1:7] 3 6 9 10 12 14 18
## $ : int [1:6] 1 2 3 5 10 11
## $ : int [1:3] 9 13 17
## $ : int [1:6] 2 3 4 12 18 21
## $ : int [1:8] 5 7 13 14 17 18 19 20
## $ : int [1:4] 5 6 11 14
## $ : int [1:5] 1 6 10 22 23
## $ : int [1:4] 3 5 8 18
## $ : int [1:3] 7 9 14
## $ : int [1:4] 5 9 10 13
## $ : int [1:4] 16 19 20 21
## $ : int [1:2] 15 19
```

```

## $ : int [1:2] 7 9
## $ : int [1:6] 5 8 9 12 20 21
## $ : int [1:4] 9 15 16 20
## $ : int [1:5] 9 15 18 19 21
## $ : int [1:5] 4 8 15 18 20
## $ : int [1:2] 11 23
## $ : int [1:3] 1 11 22
## - attr(*, "class")= chr "nb"
## - attr(*, "region.id")= chr [1:23] "1" "2" "3" "4" ...
## - attr(*, "call")= language poly2nb(pl = bur_ccsp, row.names = bur_ccsp$lsoa_code)
## - attr(*, "type")= chr "queen"
## - attr(*, "sym")= logi TRUE

```

We can graphically represent the links using the following code:

```

#We first plot the boundaries
plot(bur_ccsp, col='gray', border='blue', lwd=2)
#Then we use the coordinates function to obtain the coordinates of the polygon centroids
xy <- coordinates(bur_ccsp)
#Then we draw lines between the polygons centroids for neighbours that are listed as l
plot(w, xy, col='red', lwd=2, add=TRUE)

```



7.5 Generating the weight matrix

We can transform `w` into a spatial weights matrix. A spatial weights matrix reflects the intensity of the geographic relationship between observations. For this we use the `spdep` function `nb2mat()`.

```
wm <- nb2mat(w, style='B')
wm

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## 1     0    1    0    0    0    1    0    0    0    0    1    0    0    0    0
## 2     1    0    1    1    0    1    0    1    0    0    0    0    0    0    0
## 3     0    1    0    0    1    1    0    1    0    0    0    0    1    0    0
## 4     0    1    0    0    0    0    0    1    0    0    0    0    0    0    0
## 5     0    0    1    0    0    1    0    0    1    1    0    1    0    1    0
## 6     1    1    1    0    1    0    0    0    0    1    1    1    0    0    0
## 7     0    0    0    0    0    0    0    0    1    0    0    0    0    1    0
## 8     0    1    1    1    0    0    0    0    0    0    0    0    1    0    0
## 9     0    0    0    0    1    0    1    0    0    0    0    0    0    0    1
## 10    0    0    0    0    1    1    0    0    0    0    0    1    0    0    1
## 11    1    0    0    0    0    1    0    0    0    0    1    0    0    0    0
## 12    0    0    1    0    1    0    0    1    0    0    0    0    0    0    0
## 13    0    0    0    0    0    0    1    0    1    0    0    0    0    0    1
## 14    0    0    0    0    1    0    0    0    0    1    1    0    0    1    0
## 15    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## 16    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
## 17    0    0    0    0    0    0    1    0    1    0    0    0    0    0    0
## 18    0    0    0    0    1    0    0    0    1    1    0    0    0    1    0
## 19    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
## 20    0    0    0    0    0    0    0    0    0    1    0    0    0    0    0
## 21    0    0    0    1    0    0    0    1    0    0    0    0    0    0    0
## 22    0    0    0    0    0    0    0    0    0    0    0    1    0    0    0
## 23    1    0    0    0    0    0    0    0    0    0    0    1    0    0    0
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23]
## 1     0    0    0    0    0    0    0    0    1
## 2     0    0    0    0    0    0    0    0    0
## 3     0    0    0    0    0    0    0    0    0
## 4     0    0    0    0    0    0    1    0    0
## 5     0    0    0    1    0    0    0    0    0
## 6     0    0    0    0    0    0    0    0    0
## 7     0    0    1    0    0    0    0    0    0
## 8     0    0    0    1    0    0    1    0    0
## 9     0    0    1    1    1    1    0    0    0
## 10   0    0    0    0    0    0    0    0    0
## 11   0    0    0    0    0    0    0    1    1
```

```

## 12    0    0    0    1    0    0    0    0    0
## 13    0    0    0    0    0    0    0    0    0
## 14    0    0    0    0    0    0    0    0    0
## 15    0    1    0    0    1    1    1    0    0
## 16    1    0    0    0    1    0    0    0    0
## 17    0    0    0    0    0    0    0    0    0
## 18    0    0    0    0    0    1    1    0    0
## 19    1    1    0    0    0    1    0    0    0
## 20    1    0    0    1    1    0    1    0    0
## 21    1    0    0    1    0    1    0    0    0
## 22    0    0    0    0    0    0    0    0    1
## 23    0    0    0    0    0    0    0    1    0
## attr(,"call")
## nb2mat(neighbours = w, style = "B")

```

This matrix has values of 0 or 1 indicating whether the elements listed in the rows are adjacent (using our definition, which in this case was the Queen criteria) with each other. The diagonal is full of zeroes. An area cannot be a neighbour of itself. So, if you look at the first two and the second column you see a 1. That means that the LSOA with the code E01005065 is a neighbour of the second LSOA (as listed in the rows) which is E01005066. You will zeroes for many of the other columns because this LSOA only has 4 neighbours.

7.5.1 Homework 2

Looking at this matrix identify the other 3 neighbours of E01005065

In many computations we will see that the matrix is **row standardised**. We can obtain a row standardise matrix changing the code:

```

wm_rs <- nb2mat(w, style='W')
wm_rs

##      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## 1  0.0000000 0.2500000 0.0000000 0.0000000 0.0000000 0.2500000 0.0000000
## 2  0.2000000 0.0000000 0.2000000 0.2000000 0.0000000 0.2000000 0.0000000
## 3  0.0000000 0.2000000 0.0000000 0.0000000 0.2000000 0.2000000 0.0000000
## 4  0.0000000 0.3333333 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## 5  0.0000000 0.0000000 0.1428571 0.0000000 0.0000000 0.1428571 0.0000000
## 6  0.1666667 0.1666667 0.1666667 0.0000000 0.1666667 0.0000000 0.0000000
## 7  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## 8  0.0000000 0.1666667 0.1666667 0.1666667 0.0000000 0.0000000 0.0000000
## 9  0.0000000 0.0000000 0.0000000 0.0000000 0.1250000 0.0000000 0.1250000
## 10 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.2500000 0.2500000 0.0000000
## 11 0.2000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.2000000 0.0000000

```

```

## 12 0.0000000 0.0000000 0.2500000 0.0000000 0.2500000 0.0000000 0.0000000
## 13 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.3333333
## 14 0.0000000 0.0000000 0.0000000 0.0000000 0.2500000 0.0000000 0.0000000
## 15 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## 16 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## 17 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.5000000
## 18 0.0000000 0.0000000 0.0000000 0.0000000 0.1666667 0.0000000 0.0000000
## 19 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## 20 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## 21 0.0000000 0.0000000 0.0000000 0.2000000 0.0000000 0.0000000 0.0000000
## 22 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## 23 0.3333333 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##      [,8]      [,9]      [,10]     [,11]     [,12]     [,13]     [,14]     [,15]
## 1  0.0000000 0.0000000 0.0000000 0.2500000 0.0000000 0.0000000 0.0000000 0.00
## 2  0.2000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 3  0.2000000 0.0000000 0.0000000 0.0000000 0.2000000 0.0000000 0.0000000 0.00
## 4  0.3333333 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 5  0.0000000 0.1428571 0.1428571 0.0000000 0.1428571 0.0000000 0.1428571 0.00
## 6  0.0000000 0.0000000 0.1666667 0.1666667 0.0000000 0.0000000 0.0000000 0.00
## 7  0.0000000 0.3333333 0.0000000 0.0000000 0.0000000 0.3333333 0.0000000 0.00
## 8  0.0000000 0.0000000 0.0000000 0.0000000 0.1666667 0.0000000 0.0000000 0.00
## 9  0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.1250000 0.1250000 0.00
## 10 0.0000000 0.0000000 0.0000000 0.2500000 0.0000000 0.0000000 0.2500000 0.00
## 11 0.0000000 0.0000000 0.2000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 12 0.2500000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 13 0.0000000 0.3333333 0.0000000 0.0000000 0.0000000 0.0000000 0.3333333 0.00
## 14 0.0000000 0.2500000 0.2500000 0.0000000 0.0000000 0.2500000 0.0000000 0.00
## 15 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 16 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.50
## 17 0.0000000 0.5000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 18 0.1666667 0.1666667 0.0000000 0.0000000 0.1666667 0.0000000 0.0000000 0.00
## 19 0.0000000 0.2500000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.25
## 20 0.0000000 0.2000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.20
## 21 0.2000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.20
## 22 0.0000000 0.0000000 0.5000000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 23 0.0000000 0.0000000 0.0000000 0.3333333 0.0000000 0.0000000 0.0000000 0.00
##      [,16]     [,17]     [,18]     [,19]     [,20]     [,21]     [,22]     [,23]
## 1  0.00 0.0000000 0.0000000 0.00 0.0000000 0.0000000 0.0000000 0.25
## 2  0.00 0.0000000 0.0000000 0.00 0.0000000 0.0000000 0.0000000 0.00
## 3  0.00 0.0000000 0.0000000 0.00 0.0000000 0.0000000 0.0000000 0.00
## 4  0.00 0.0000000 0.0000000 0.00 0.0000000 0.3333333 0.0000000 0.00
## 5  0.00 0.0000000 0.1428571 0.00 0.0000000 0.0000000 0.0000000 0.00
## 6  0.00 0.0000000 0.0000000 0.00 0.0000000 0.0000000 0.0000000 0.00
## 7  0.00 0.3333333 0.0000000 0.00 0.0000000 0.0000000 0.0000000 0.00
## 8  0.00 0.0000000 0.1666667 0.00 0.0000000 0.1666667 0.0000000 0.00
## 9  0.00 0.1250000 0.1250000 0.125 0.1250000 0.0000000 0.0000000 0.00

```

```

## 10 0.00 0.0000000 0.0000000 0.000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 11 0.00 0.0000000 0.0000000 0.000 0.0000000 0.0000000 0.2000000 0.20
## 12 0.00 0.0000000 0.2500000 0.000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 13 0.00 0.0000000 0.0000000 0.000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 14 0.00 0.0000000 0.0000000 0.000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 15 0.25 0.0000000 0.0000000 0.250 0.2500000 0.2500000 0.0000000 0.0000000 0.00
## 16 0.00 0.0000000 0.0000000 0.500 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 17 0.00 0.0000000 0.0000000 0.000 0.0000000 0.0000000 0.0000000 0.0000000 0.00
## 18 0.00 0.0000000 0.0000000 0.000 0.1666667 0.1666667 0.0000000 0.0000000 0.00
## 19 0.25 0.0000000 0.0000000 0.000 0.2500000 0.0000000 0.0000000 0.0000000 0.00
## 20 0.00 0.0000000 0.2000000 0.200 0.0000000 0.2000000 0.0000000 0.0000000 0.00
## 21 0.00 0.0000000 0.2000000 0.000 0.2000000 0.0000000 0.0000000 0.0000000 0.00
## 22 0.00 0.0000000 0.0000000 0.000 0.0000000 0.0000000 0.0000000 0.0000000 0.50
## 23 0.00 0.0000000 0.0000000 0.000 0.0000000 0.0000000 0.0000000 0.3333333 0.00
## attr(,"call")
## nb2mat(neighbours = w, style = "W")

```

Row standardisation of a matrix ensure that the sum of the rows adds up to 1. So, for example, if you have four neighbours and that has to add up to 4, you need to divide 1 by 4, which gives you 0.25. So in the columns for a polygon with 4 neighbours you will see 0.25 in the column representing each of the neighbours.

7.6 Moran's I

The most well known measure of spatial autocorrelation is the Moran's I. It was developed by Patrick Alfred Pierce Moran, an Australian statistician. You can find the formula and some explanation in the wikipedia article. The video lecture by Luc Anselin covers an explanation of Moran's I. We strongly recommend you watch the video. You can also find helpful this link if things are still unclear. The formula you see may look intimidating but it is nothing but the formula for standard correlation expanded to incorporate the spatial weight matrix.

Before we can use the functions from `spdep` to compute the global Moran's I we need to create a `listw` type spatial weights object (instead of the matrix we used above). To get the same value as above we use `style='B'` to use binary (TRUE/FALSE) distance weights.

```

ww <- nb2listw(w, style='B')
ww

## Characteristics of weights list object:
## Neighbour list object:
## Number of regions: 23

```

```

## Number of nonzero links: 100
## Percentage nonzero weights: 18.90359
## Average number of links: 4.347826
##
## Weights style: B
## Weights constants summary:
##   n nn S0 S1 S2
## B 23 529 100 200 1960

```

Now we can use the `moran()` function. Have a look at `?moran` to see a description of this in R. The function is defined as `moran(y, ww, n, Szero(ww))`. Note the odd arguments `n` and `S0`. I think they are odd, because “`ww`” has that information. Anyway, we supply them and it works. There probably are cases where it makes sense to use other values.

```
moran(bur_ccsp$burglary, ww, n=length(ww$neighbours), S0=Szero(ww))
```

```

## $I
## [1] 0.1203294
##
## $K
## [1] 7.964184

```

So the Moran's I here is 0.12, which is not very large. The Index is bounded by -1 and 1, and 0.12 is not large, but it is still above zero, so we might want to explore whether this is significant using a test of statistical significance for this statistic.

The Spatial Autocorrelation (Global Moran's I) tool is an inferential statistic, which means that the results of the analysis are always interpreted within the context of its null hypothesis. For the Global Moran's I statistic, the null hypothesis states that the attribute being analysed is randomly distributed among the features in your study area; said another way, the spatial processes promoting the observed pattern of values is random chance. Imagine that you could pick up the values for the attribute you are analysing and throw them down onto your features, letting each value fall where it may. This process (picking up and throwing down the values) is an example of a random chance spatial process. When the p-value returned by this tool is statistically significant, you can reject the null hypothesis.

In some software you can use statistical tests invoking asymptotic theory, but the only appropriate way of doing these tests is by using a Monte Carlo procedure. The way Monte Carlo works is that the values of burglary are randomly assigned to the polygons, and the Moran's I is computed. This is repeated several times to establish a distribution of expected values. The observed value of Moran's I

206CHAPTER 7. GLOBAL AND LOCAL SPATIAL AUTOCORRELATION

is then compared with the simulated distribution to see how likely it is that the observed values could be considered a random draw.

If confused, watch this quick video on monte carlo simulations.

We use the function `moran.mc()` to run a permutation test for Moran's I statistic calculated by using some number of random permutations of our numeric vector, for the given spatial weighting scheme, to establish the rank of the observed statistic in relation to the simulated values.

We need to specify our variable of interest (*burglary*), the `listw` object we created earlier (*ww*), and the number of permutations we want to run (here we choose 99).

```
set.seed(1234) # The seed number you choose is the starting point used in the generation
burg_moranmc_results <- moran.mc(bur_ccsp$burglary, ww, nsim=99)
burg_moranmc_results

##
## Monte-Carlo simulation of Moran I
##
## data: bur_ccsp$burglary
## weights: ww
## number of simulations + 1: 100
##
## statistic = 0.12033, observed rank = 92, p-value = 0.08
## alternative hypothesis: greater
```

So, the probability of observing this Moran's I if the null hypothesis was true is 0.08. This is higher than our alpha level of 0.05. In this case, we can conclude that there isn't a significant global spatial autocorrelation.

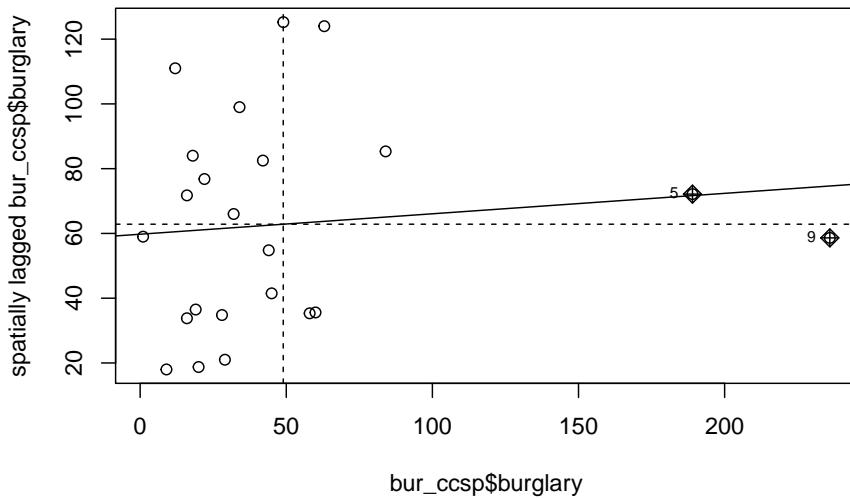
We can make a “Moran scatter plot” to visualize spatial autocorrelation. Note the row standardisation of the weights matrix.

```
rwm <- mat2listw(wm, style='W')
# Checking if rows add up to 1 (they should)
mat <- listw2mat(rwm)
#This code is simply adding each row to see if we get one when we add their values up,
apply(mat, 1, sum)[1:15]

##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
```

Now we can plot:

```
moran.plot(bur_ccsp$burglary, rwm)
```



The X axis represents the values of our burglary variable in each unit (each LSOA) and the Y axis represents a spatial lag of this variable. A spatial lag in this context is simply the average value of the burglary count in the areas that are considered neighbours of each LSOA. So we are plotting the value of burglary against the average value of burglary in the neighbours. And you can see the correlation is almost flat here. As with any correlation measure, you could get **positive spatial autocorrelation**, that would mean that as you move further to the right in the X axis you have higher levels of burglary in the surrounding area. This is what we see here. But the correlation is fairly low and as we saw is not statistically significant. You can also obtain **negative spatial autocorrelation**. That is, that would mean that areas with high level of crime *tend* (it's all about the global average effect!) to be surrounded by areas with low levels of crime. This is clearly not what we see here.

It is very important to understand that global statistics like the spatial autocorrelation (Global Moran's I) tool assess the overall pattern and trend of your data. They are most effective when the spatial pattern is consistent across the study area. In other words, you may have clusters (local pockets of autocorrelation), without having clustering (global autocorrelation). This may happen if the sign of the clusters negate each other.

But don't just take our word for how important this is, or how it's commonly applied in criminological research. Instead, now that you've gone through on

how to do this, and have begun to get a sense of understanding, read the following paper on <https://link.springer.com/article/10.1023/A:1007592124641> where the authors make use of Moran's I to explain spatial characteristics of homicide. You will likely see this in other papers as well, and now you will know what it means and why it's important.

7.6.1 Homework 3 (Optional Practice)

You know what is coming, don't you? Yes, you need to compute the Moran's I for burglary again. But this time, you need to do it for the whole of Manchester city. I won't mark you down if you don't try this. But why wouldn't you?

7.7 Local spatial autocorrelation

So now we know about global measures of spatial association, in particular the Moran's I statistic, which provide a mechanism to make inferences about a population from a sample. While this is useful for us to be able to assess quantitatively whether crime events cluster in a non-random manner, in the words of Jerry Ratcliffe "this simply explains what most criminal justice students learn in their earliest classes." For example, consider the study of robberies in Philadelphia:

2. Crime Mapping: Spatial and Temporal Challenges



FIGURE 2.1. Philadelphia robbery hotspots, from quartic kernel density estimation.

Aggregated to the police districts, this returns a global Moran's I value (range 1 to 1) of 0.56, which suggests that police sectors with high robbery counts adjoin sectors that also have high robbery counts, and low crime sectors are often neighbours of other low crime sectors, something that should hardly be surprising given the above map (Ratcliffe, Jerry. "Crime mapping: spatial and temporal challenges." *Handbook of quantitative criminology*. Springer, New York, NY, 2010. 5-24.).

In this section we will learn about local indicators of spatial association (LISA) and show how they allow for the decomposition of global indicators, such as Moran's I, into the contribution of each observation. The LISA statistics serve two purposes. On one hand, they may be interpreted as indicators of local pockets of nonstationarity, or hot spots. On the other hand, they may be used to assess the influence of individual locations on the magnitude of the global statistic and to identify "outliers" (Anselin, Luc. "Local indicators of spatial association—LISA." *Geographical analysis* 27.2 (1995): 93-115.).

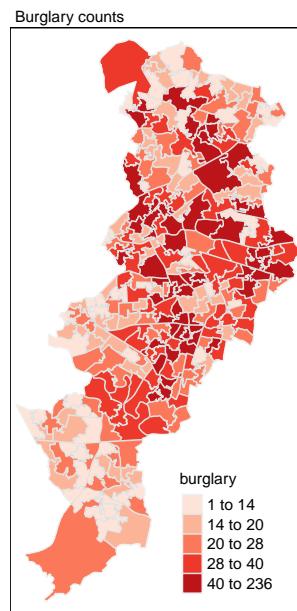
7.8 Getting Manchester data and weights

To explore local indicators of spatial correlation, let's go back to using all of Manchester, rather than just City Centre ward. We want to have enough data

to see local variation, and while the code may take slightly longer to run, we can have a go at more meaningful stats.

So this is our `burglaries_per_lsoa` object that we're referring back to. To check what our data look like, we can always plot again with `tmap`:

```
tmap_mode("plot")  
  
## tmap mode set to plotting  
  
tm_shape(burglaries_per_lsoa) +  
  tm_fill("burglary", style = "quantile", palette = "Reds") +  
  tm_borders(alpha = 0.1) +  
  tm_layout(main.title = "Burglary counts", main.title.size = 0.7,  
            legend.position = c("right", "bottom"), legend.title.size = 0.8)
```



Looks good! Now that we have the data we need to coerce into a spatial object, as before, for it to work well with the functions we use from the `sp` package, and then generate the weight matrix. Again, what we do here is stuff we did above for the global correlation. In fact, if you did the optional homework already, you will also have run this code.

```
#Coerce sf into sp  
burglary_m <- as(burglaries_per_lsoa, "Spatial")
```

```
#Generate list of neighbours using the Queen criteria
w <- poly2nb(burglary_m, row.names=burglary_m$lsoa_code)
#Generate list with weights using row standardisation
ww <- nb2listw(w, style='W')
```

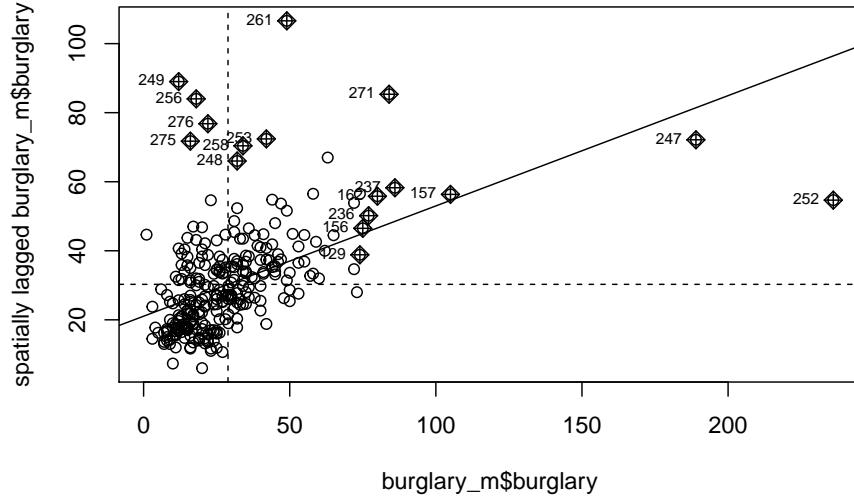
7.9 Generating and visualising the LISA measures

The first step before we can generate the LISA map is to compute the local Moran's I. The initial part of the video presentation by Luc Anselin that we expected you to watch explains the formula and logic underpinning these computations and we won't reiterate here that detail. But at least a general reminder:

Global tests for spatial autocorrelation [introduced last week] are calculated from the local relationships between the values observed at a spatial entity and its neighbours, for the neighbour definition chosen. Because of this, we can break global measures down into their components, and by extension, construct localised tests intended to detect '**clusters**' – observations with very similar neighbours – and '**hotspots**' [spatial outliers] – observations with very different neighbours. (Bivand et al. 2008, highlights added)

Let's first look at the Moran's scatterplot:

```
moran.plot(burglary_m$burglary, ww)
```



Notice how the plot is split in 4 quadrants. The top right corner belongs to areas that have high level of burglary and are surrounded by other areas that have above the average level of burglary. This are the high-high locations that Luc Anselin referred to. The bottom left corner belongs to the low-low areas. These are areas with low level of burglary and surrounded by areas with below average levels of burglary. Both the high-high and low-low represent clusters. A high-high cluster is what you may refer to as a hot spot. And the low-low clusters represent cold spots. In the opposite diagonal we have *spatial outliers*. They are not outliers in the standard sense, extreme observations, they are outliers in that they are surrounded by areas that are very unlike them. So you could have high-low spatial outliers, areas with high levels of burglary and low levels of surrounding burglary, or low-high spatial outliers, areas that have themselves low levels of burglary (or whatever else we are mapping) and that are surrounded by areas with above average levels of burglary.

You can also see here that the positive spatial autocorrelation is more noticeable when we focus on the whole of Manchester city, unlike what we observed when only looked at the city centre. You can check this running the global Moran's I.

```
moran(burglary_m$burglary, ww, n=length(ww$neighbours), S0=Szero(ww))
```

```
## $I
## [1] 0.319477
##
## $K
```

```

## [1] 33.59689

moran.mc(burglary_m$burglary, ww, nsim=99999)

##
## Monte-Carlo simulation of Moran I
##
## data: burglary_m$burglary
## weights: ww
## number of simulations + 1: 1e+05
##
## statistic = 0.31948, observed rank = 1e+05, p-value = 1e-05
## alternative hypothesis: greater

```

You can see that the global Moran's is 0.32 and that is highly significant. There is indeed global spatial autocorrelation, when we look at all of Manchester (not just city centre ward). Knowing this we can try to decompose this, figure out what is driving this global measure.

To compute the local Moran we can use a function from the `spdep` package.

```

locm_bm <- localmoran(burglary_m$burglary, ww)
summary(locm_bm)

```

	Ii	E.Ii	Var.Ii	Z.Ii
## Min.	-1.933870	Min. :-0.003559	Min. :0.07787	Min. :-4.12540
## 1st Qu.:-0.004654	1st Qu.:-0.003559	1st Qu.:0.14505	1st Qu.:-0.00256	
## Median : 0.123084	Median :-0.003559	Median :0.17460	Median : 0.29992	
## Mean : 0.319477	Mean :-0.003559	Mean :0.17805	Mean : 0.84062	
## 3rd Qu.: 0.314142	3rd Qu.:-0.003559	3rd Qu.:0.21894	3rd Qu.: 0.77152	
## Max. :13.201230	Max. :-0.003559	Max. :0.44062	Max. :37.50929	
## Pr(z > 0)				
## Min. :0.0000				
## 1st Qu.:0.2202				
## Median :0.3821				
## Mean :0.3707				
## 3rd Qu.:0.5010				
## Max. :1.0000				

The first column provides the summary statistic for the local moran statistic. Being local you will have one for each of the areas. The last column gives you a p value for this statistic. In order to produce the LISA map we need to do some previous work. First we are going to create some new variables that we are going to need:

214CHAPTER 7. GLOBAL AND LOCAL SPATIAL AUTOCORRELATION

First we scale the variable of interest. When we scale burglary what we are doing is re-scaling the values so that the mean is zero. See an explanation of what this does here.

We use `scale()`, which is a generic function whose default method centers and/or scales the variable. Here centering is done by subtracting the mean (omitting NAs) the corresponding columns, and scaling is done by dividing the (centered) variable by their standard deviations:

```
#scale the variable of interest and save it to a new column  
burglary_m$burglary <- scale(burglary_m$burglary) %>% as.vector()
```

We've also added `as.vector()` to the end, to make sure that the data type we get out of this is a vector, that maps neatly into our dataframe.

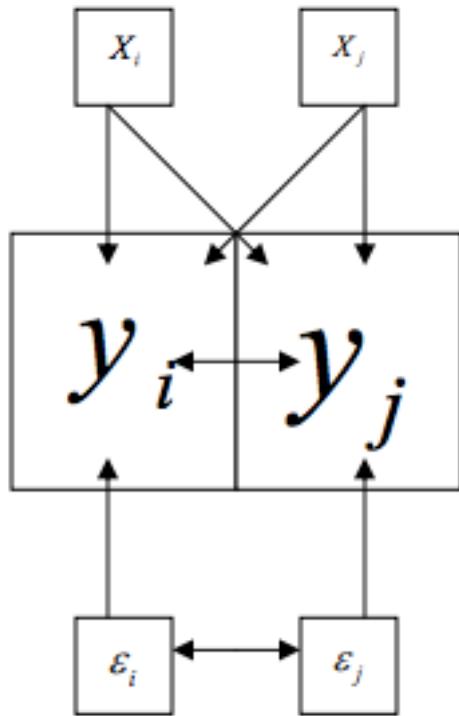
Now we also want to account for the spatial dependence of our values. Remember how we keep saying about “The First Law of Geography”, according to Waldo Tobler, is “everything is related to everything else, but near things are more related than distant things.” Seriously, we should all just tattoo this onto our foreheads because this is the key message of the module...!

So what do we mean by this spatial dependence? When a value observed in one location depends on the values observed at neighbouring locations, there is a **spatial dependence**. And spatial data may show spatial dependence in the variables and error terms.

Why should spatial dependence occur? There are two reasons commonly given. First, data collection of observations associated with spatial units may reflect measurement error. This happens when the boundaries for which information is collected do not accurately reflect the nature of the underlying process generating the sample data. A second reason for spatial dependence is that the spatial dimension of a social or economic characteristic may be an important aspect of the phenomenon. For example, based on the premise that location and distance are important forces at work, regional science theory relies on notions of spatial interaction and diffusion effects, hierarchies of place and spatial spillovers.

There are two types of dependence, spatial error and spatial lag. Here we focus on spatial lag.

Spatial lag is when the dependent variable y in place i is affected by the independent variables in both place i and j .



This will be important to keep in mind when considering spatial regression. With spatial lag in ordinary least square regression, the assumption of uncorrelated error terms is violated, because near things will have associated error terms. Similarly, the assumption of independent observations is also violated, as the observations are influenced by the other observations near them. As a result, the estimates are biased and inefficient. Spatial lag is suggestive of a possible diffusion process – events in one place predict an increased likelihood of similar events in neighboring places.

So to be able to account for the spatial lag in our model, we must create a variable to account for this. We can do this with the `lag.listw()` function. Remember from last week: a spatial lag in this context is simply the average value of the burglary count in the areas that are considered neighbours of each LSOA. So we are plotting the value of burglary against the average value of burglary in the neighbours.

For this we need our `listw` object, which is the `ww` object created earlier, when we generated the list with weights using row standardisation. We then pass this `listw` object into the `lag.listw()` function, which computes the spatial lag of a numeric vector using a `listw` sparse representation of a spatial weights matrix.

216CHAPTER 7. GLOBAL AND LOCAL SPATIAL AUTOCORRELATION

```
#create a spatial lag variable and save it to a new column  
burglary_m$lag_s_burglary <- lag.listw(ww, burglary_m$s_burglary)
```

Make sure to check the summaries to ensure nothing weird is going on

```
summary(burglary_m$s_burglary)
```

```
##      Min. 1st Qu. Median     Mean 3rd Qu.     Max.  
## -1.2149 -0.6048 -0.1908  0.0000  0.2667  9.0256
```

```
summary(burglary_m$lag_s_burglary)
```

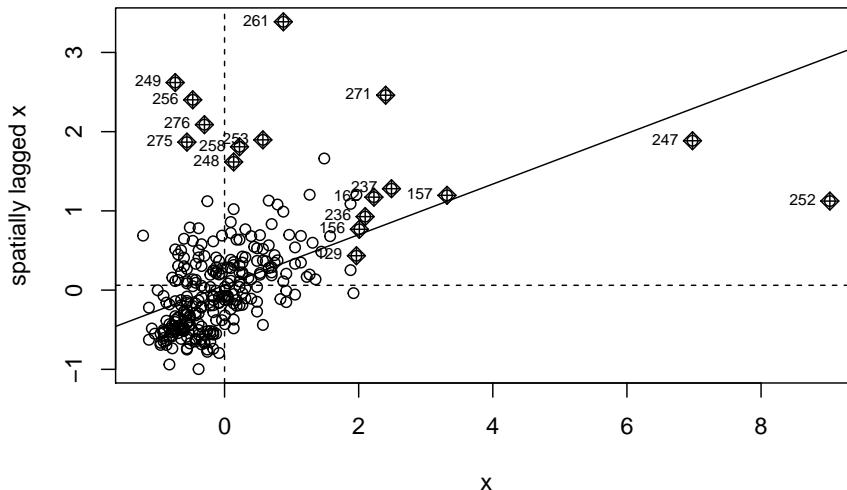
```
##      Min. 1st Qu. Median     Mean 3rd Qu.     Max.  
## -0.99700 -0.44086 -0.07464  0.06068  0.34738  3.38679
```

We can create a Moran scatter plot so that you see that nothing has changed apart from the scale in which we are using the variables. The observations that are influential are highlighted in the plot as you can see.

```
x <- burglary_m$s_burglary  
y <- burglary_m$lag_s_burglary  
xx <- data.frame(x,y)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.  
## Please use `tibble()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
moran.plot(x, ww)
```



We are now going to create a new variable to identify the quadrant in which each observation falls within the Moran Scatter plot, so that we can tell apart the high-high, low-low, high-low, and low-high areas. We will only identify those that are significant according to the p value that was provided by the local moran function.

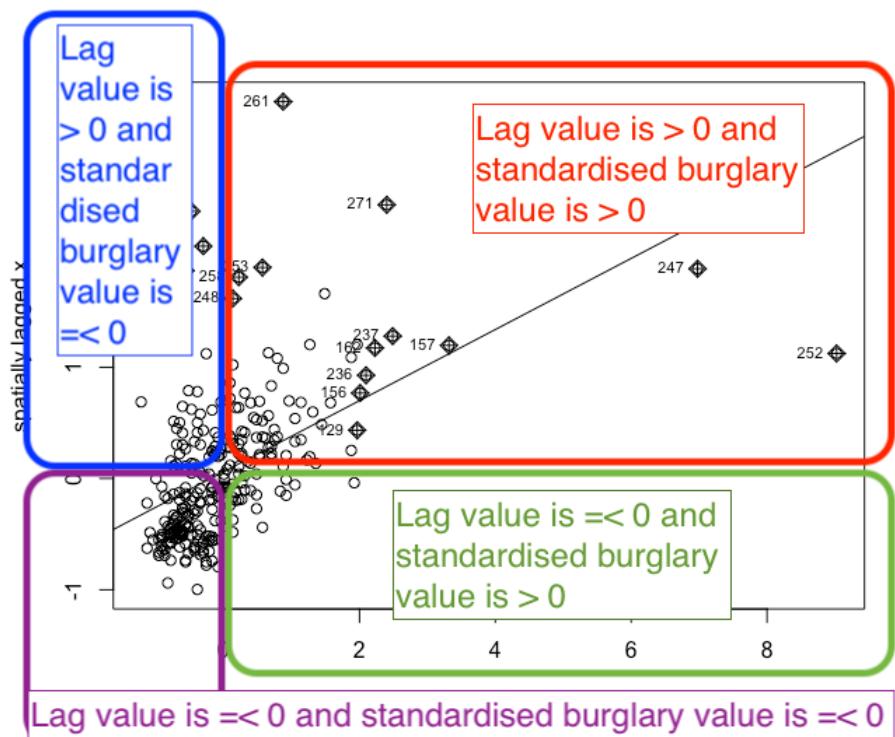
Before we get started, let's quickly review the tools we will use.

All our data is in this `burglary_m` dataframe. This has a variable for the LSOA code (`code`), a variable for the number of burglaries (`burglary`), and then also the two variables we created, the scaled measure of burglary (`s_burglary`), and the spatial lag measure (`lag_s_burglary`).

We also have our `locm_bm` object, which we created with the `localmoran()` function, that has calculated a variety of measures for each of our observations, which we explored with the `summary()` function. You can see (if you scroll up) that the 5th element in this object is the p-value ("Pr($z > 0$)"). To call the nth element of an object, you can use the square brackets after its name. So to return the nth column of thing, you can use `thing[,n]`. Again this should not be new to you, as we've been doing this sort of thing for a while.

So the data we need for each observation, in order to identify whether it belongs to the high-high, low-low, high-low, or low-high quadrants are the standardised burglary score, the spatial lag score, and the p-value.

Essentially all we'll be doing, is assigning a variable values based on where in the plot it is. So for example, if it's in the upper right, it is high-high, and has values larger than 0 for both the burglary and the spatial lag values. If it's in the upper left, it's low-high, and has a value larger than 0 for the spatial lag value, but lower than 0 on the burglary value. And so on, and so on. Here's an image to illustrate:



So let's first initialise this variable. In this instance we are creating a new column in the `burglary_m` dataframe and calling it `quad_sig`.

We are using the `mutate()` function from the `dplyr` package to create our new variable, just as we have in previous labs.

We also use nested `ifelse()` statements. Nested `ifelse()` just means that it's an `ifelse()` inside another `ifelse()` statement. To help us with these sorts of situations is the `ifelse()` function. We saw this with the previous exercises, but I'll describe it briefly again. It allows us to conditionally assign some value to some variable. The structure of the function is so that you have to pass it a condition, then a value to assign if the condition is true, and then another value if the condition is false. You are basically using this function to say: "if this condition is true, do first thing, else, do the second thing". It would look something like this:

```
dataframe$new_variable <- ifelse(dataframe$some_numeric_var < 100, "smaller than 100", "not small
```

When nesting these, all you do is put another condition to check in the “thing to do if false”, so it checks all conditions. So in the first instance we check if the value for burglary is greater than zero, and the value for the lag is greater than zero, and the p-value is smaller than our threshold of 0.05. If it is, then this should belong to the “high-high” group. If any one of these conditions is not met, then we move into the ‘thing to do if false’ section, where we now check again another set of criteria, and so on and so on. If none of these are met, we assign it the non-significant value:

```
burglary_m <- st_as_sf(burglary_m) %>%
  mutate(quad_sig = ifelse(burglary_m$s_burglary > 0 &
                           burglary_m$lag_s_burglary > 0 &
                           locm_bm[,5] <= 0.05,
                           "high-high",
                           ifelse(burglary_m$s_burglary <= 0 &
                                   burglary_m$lag_s_burglary <= 0 &
                                   locm_bm[,5] <= 0.05,
                           "low-low",
                           ifelse(burglary_m$s_burglary > 0 &
                                   burglary_m$lag_s_burglary <= 0 &
                                   locm_bm[,5] <= 0.05,
                           "high-low",
                           ifelse(burglary_m$s_burglary <= 0 &
                                   burglary_m$lag_s_burglary > 0 &
                                   locm_bm[,5] <= 0.05,
                           "low-high",
                           "non-significant"))))
```

(Note we had to wrap our data in a `st_as_sf()` function, to convert back to sf object).

Now we can have a look at what this returns us:

```
table(burglary_m$quad_sig)
```

```
##          high-high      low-low non-significant
##                22                  1                 259
```

This looks like a lot of non-significant results. We want to be sure this isn’t an artefact of our code but is true, we can check how many values are under 0.05:

```
nrow(locm_bm[locm_bm[,5] <= 0.05,])
```

```
## [1] 23
```

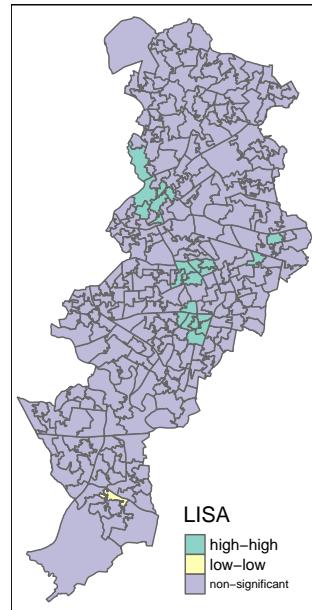
We can see that only 23 areas have p-values under 0.05 threshold. So this is in line with our results, and we can rest assured.

Well, this is exciting, but where are these regions?

Let's put 'em on a map, just simply, using quick thematic map (`qtm()`):

```
qtm(burglary_m, fill="quad_sig", fill.title="LISA")
```

```
## Some legend labels were too wide. These labels have been resized to 0.50. Increase ...
```



Very nice!

So how do we interpret these results? Well keep in mind:

- The LISA value for each location is determined from its individual contribution to the global Moran's I calculation.
- Whether or not this value is statistically significant is assessed by comparing the actual value to the value calculated for the same location by randomly reassigning the data among all the areal units and recalculating the values each time (the Monte Carlo simulation approach discussed earlier).

So essentially this map now tells us that there was statistically significant moderate clustering in burglaries in Manchester. When reporting your results, report at least the Moran's I test value and the p value. So, for this test, you should report Moran's I = 0.32, $p < .001$. Including the LISA cluster map is also a great way of showing how the attribute is actually clustering.

7.9.1 Homework 4

Using the same approach as Homework 1, can you tell me the code (ie. the LSOA identifier (e.g. E01033688)) for the safest neighbourhood to move to if you want to ensure that you have low levels of burglary to your house

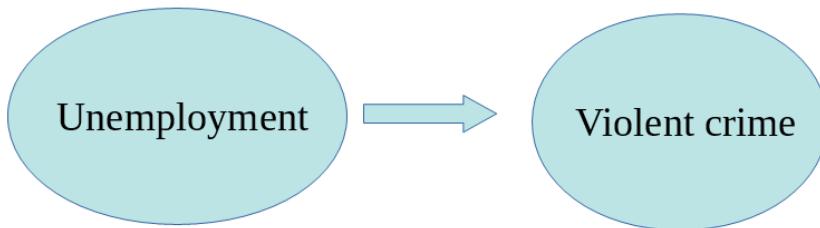
Chapter 8

Regression analysis (a refresher)

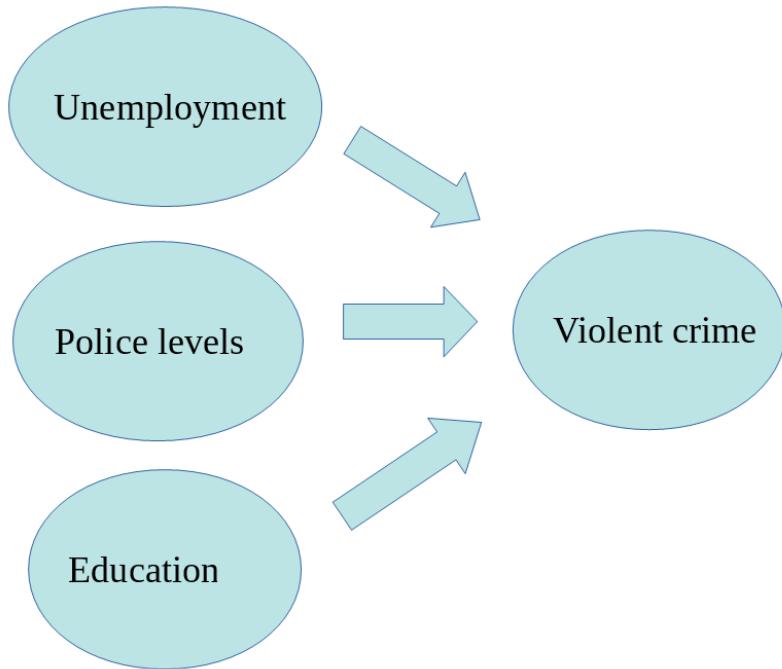
8.1 Introduction

In science one of our main concerns is to develop models of the world, models that help us to understand the world a bit better or to predict how things will develop better. You can read more about modelling in scientific research here. Statistics provides a set of tools that help researchers build and test scientific models.

Our models can be simple. We can think that unemployment is a factor that may help us to understand why cities differ in their level of violent crime. We could express such a model like this:



Surely we know the world is complex and likely there are other things that may help us to understand why some cities have more crime than others. So, we may want to have tools that allow us to examine such models. Like, for example, the one below:



In this session we are going to cover regression analysis or, rather, we are beginning to talk about regression modelling. This form of analysis has been one the main technique of data analysis in the social sciences for many years and it belongs to a family of techniques called generalised linear models. Regression is a flexible model that allows you to “explain” or “predict” a given outcome (Y), variously called your outcome, response or dependent variable, as a function of a number of what is variously called inputs, features or independent, explanatory, or predictive variables (X_1, X_2, X_3 , etc.). Following Gelman and Hill (2007), I will try to stick for the most part to the terms outputs and inputs.

Today we will cover something that is called linear regression or ordinary least squares regression (OLS), which is a technique that you use when you are interested in explaining variation in an interval level variable. First we will see how you can use regression analysis when you only have one input and then we will move to situations when we have several explanatory variables or inputs. For those of you already familiar with regression analysis this session can be a bit of a refresher, for those that aren’t a bit of an introduction. Today we will cover regression models more generally and in the next lab we will discuss adaptations to the regression model that are necessary when you have spatial autocorrelation.

We will use a dataset that includes information about homicides in the US, as well as information in a number of sociodemographic variables that are often thought of as associated with the geographical distribution of homicides.

```
##R in Windows have some problems with https addresses, that's why we need to do this first:
urlfile<-'https://s3.amazonaws.com/geoda/data/ncovr.zip'
download.file(urlfile, 'ncovr.zip')
#Let's unzip and create a new directory (ncovr) in our working directory to place the files
unzip('ncovr.zip', exdir = 'ncovr')
```

There is quite a few files and folders here. We don't need all of it. If you look inside the newly created folder you will find two subdirecoties MACOSX and ncovr. Look in the latter. You will see a pdf file called codebook.pdf. Read this file. It provides information on the variables and data that is included here. We are now going to read the main data file into R. This is the NAT.csv file.

```
ncovr <- read.csv('ncovr/ncovr/NAT.csv')
```

The dataset contains information about 3085 counties in the US and if you view it you will see it has information about several decades, the 60s, 70s, 80s, and 90s. The number at the end of the variable names denotes the relevant decade and you will see that for each decade we have the same variables.

The purpose of this and next session is to help you choose a model to represent the relationship between homicide and various predictors. You can think of a model as a map. A map aims to represent a given reality, but as you may have already discovered there are many ways of presenting the same information through a map. As an analyst you decide what the most appropriate representation for your needs is. Each representation you choose will involve an element of distortion. Maps (and models) are not exact representations of the real word, they are simply good approximations that may serve well a particular functional need. Look at the map of the metro in London.

London doesn't quite look like this. Yet, if you want to use the Metro system, this map will be extremely helpful to you. It serves a need in a good way. The same happens with models. They may not be terribly good reflections of the world, but may give us approximations that allows us to develop useful insights.

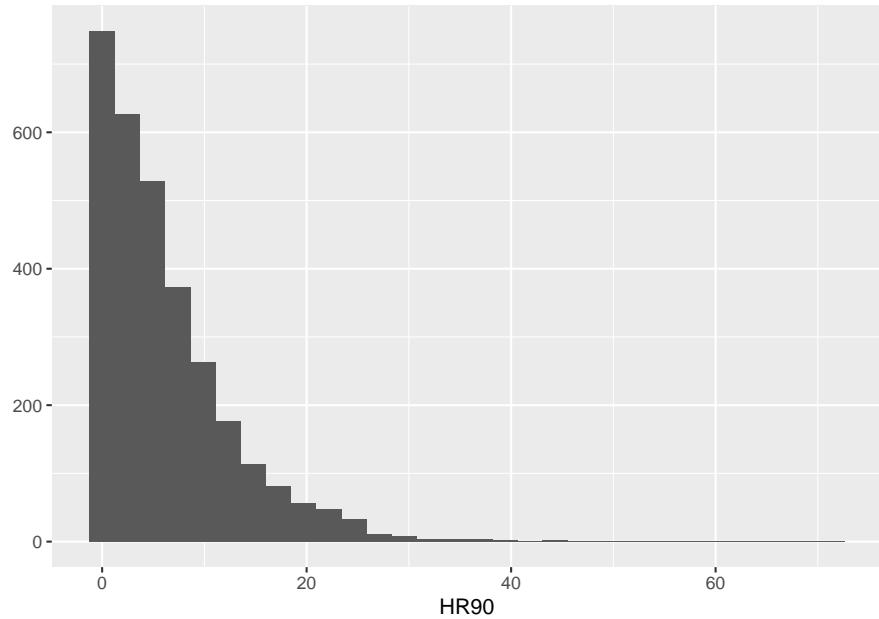
Choosing a good model is like choosing a good way for displaying quantitative information in a map. Decisions, decisions, decisions. There are many parameters and options one can choose from. This can be overwhelming, particularly as you are learning how to model and map phenomena. How to make good decisions is something that you learn on earnest by practice, practice, practice. Nobody expects you to get the maps you are doing as you are learning, and the models you are developing as you are learning spot on. So please do not stress out about this. All we can do here is to learn some basic principles and start getting some practice, which you will be able to further develop in a professional context or in further training.

The first step in any analysis is to develop some familiarity with the data you are going to be working with. We have been here before. Read the codebook.

Run summary statistics for your quantitative variables, frequency distributions for your categorical variables, and visualise your variables. This will help you to detect any anomalies and give you a sense for what you have. If, for example, you run a histogram for the homicide rate for 1990 (HR90), you will get a sense of the distribution form –which of course is skewed.

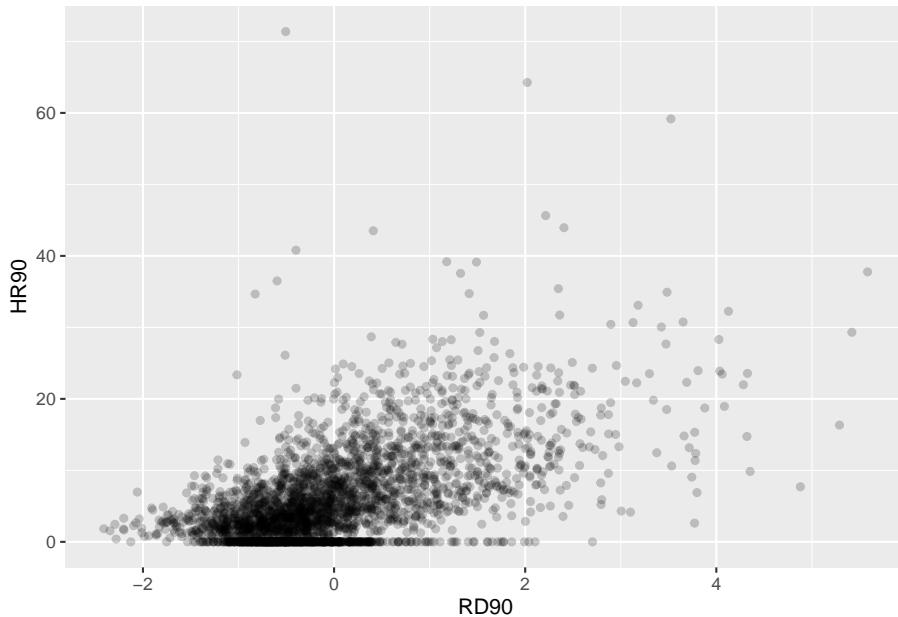
```
library(ggplot2)
qplot(x = HR90, data = ncovr)

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Once one has gone through the process of exploring the data in this way (or maybe using the `skimr` package from week 1 again) for all the variables you want to work, you can start exploring bivariate associations with your dependent, response or outcome variable. So, as an illustration, you could explore the association with resource deprivation (*RD90*), a measure of the level of concentrated disadvantage or social exclusion in an area, via a scatterplot:

```
ggplot(ncovr, aes(x = RD90, y = HR90)) +
  geom_point(alpha=.2)
```

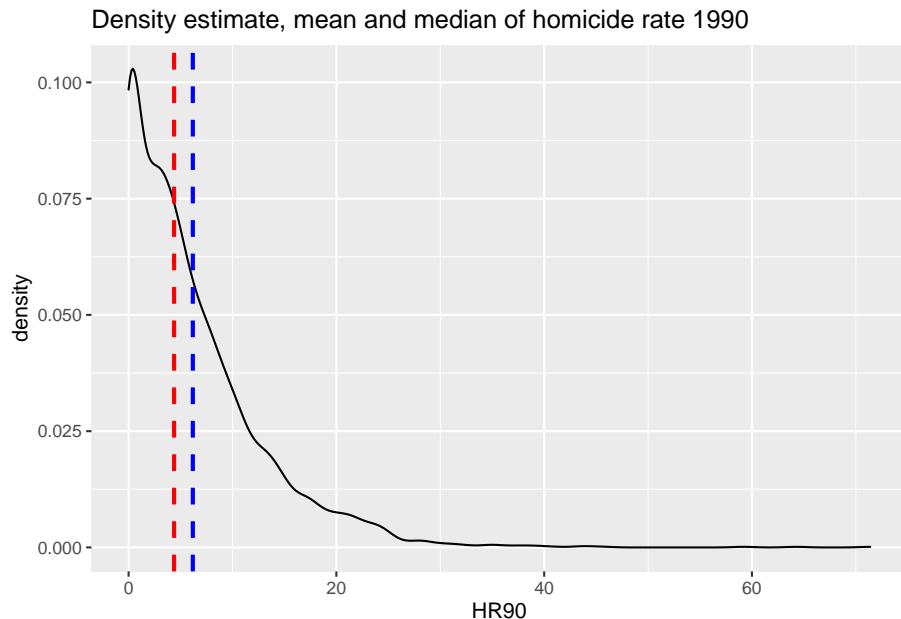


What do you think when looking at this scatterplot? Is there a relationship between the variables? Does it look as if individuals that have a high score on the X axis also have a high score on the Y axis? Or viceversa?

8.2 Motivating regression

Now, imagine that we play a game. Imagine I have all the respondents waiting in a room, and I randomly call one of them to the stage. You're sitting in the audience, and you have to guess the level of homicide (*HR90*) for that respondent. Imagine that I pay £150 to the student that gets the closest to the right value. What would you guess if you only have one guess and you knew (as we do) how homicide in the 90s is distributed?

```
ggplot(ncovr, aes(x = HR90)) +
  geom_density() +
  geom_vline(xintercept = median(ncovr$HR90), linetype = "dashed", size = 1, color="red") + # median
  geom_vline(xintercept = mean(ncovr$HR90), linetype = "dashed", size = 1, color="blue") + # mean
  ggtitle("Density estimate, mean and median of homicide rate 1990")
```



```
summary(ncovr$HR90)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	0.000	1.334	4.377	6.183	8.938	71.378

If I only had one shot, you could go for the median, in red, (given the skew) but the mean, in blue, perhaps would be your second best. Most of the areas here have values clustered around those values, which is another way of saying they are bound to be not too far from them.

Imagine, however, that now when someone is called to the stage, you are told the level of resource deprivation in the county - so the value of the *RD90* variable for the individual that has been selected (for example 4). Imagine as well that you have the scatterplot that we produced earlier in front of you. Would you still go for the value of "4.377" as your best guess for the value of the selected county?

I certainly would not go with the overall mean or median as my prediction anymore. If somebody said to me, the value *RD90* for the selected respondent is 4, I would be more inclined to guess the mean value for the level of homicide with that level of resource deprivation (the conditional mean), rather than the overall mean across all the counties. Wouldn't you?

If we plot the conditional means we can see that the mean of homicide rate for counties that report a value of 4 in *RD90* is around 22. So you may be better off guessing that.

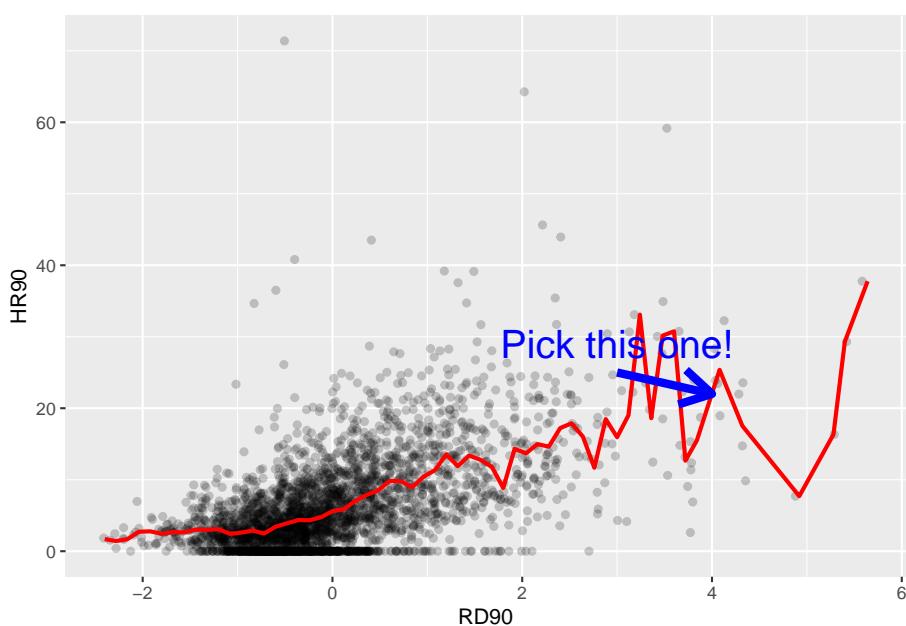
```

library(grid)
ggplot() +
  geom_point(data=ncovr, aes(x = RD90, y = HR90), alpha=.2) +
  geom_line(data=ncovr, aes(x = round(RD90/0.12)*0.12, y = HR90), # Can you guess why we do not wrap
            stat='summary',
            fun.y=mean,
            color="red",
            size=1) +
  annotate("segment", x=3, xend = 4, y = 25, yend= 22, color = "blue", size = 2, arrow = arrow())
  annotate("text", x = 3, y = 29, label = "Pick this one!", size =7, colour = "blue")

## Warning: Ignoring unknown parameters: fun.y

## No summary function supplied, defaulting to `mean_se()`^

```



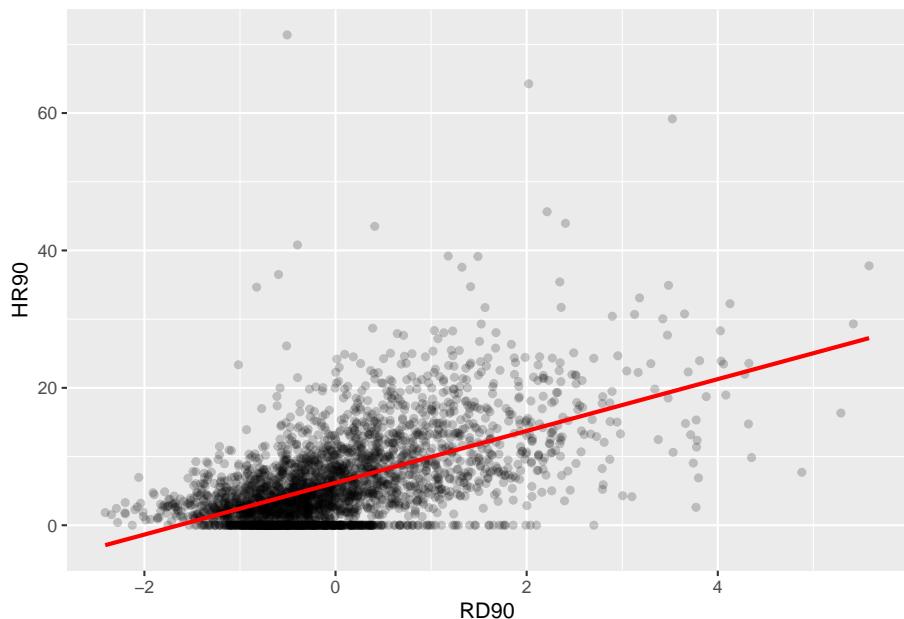
Linear regression tackles this problem using a slightly different approach. Rather than focusing on the conditional mean (smoothed or not), it draws a straight line that tries to capture the trend in the data. If we focus in the region of the scatterplot that are less sparse we see that this is an upward trend, suggesting that as resource deprivation increases so does the homicide rate.

Simple linear regression draws a single straight line of predicted values as the model for the data. This line would be a **model**, a *simplification* of the real world like any other model (e.g., a toy pistol, an architectural drawing, a subway

map), that assumes that there is approximately a linear relationship between X and Y. Let's draw the regression line:

```
ggplot(data = ncovr, aes(x = RD90, y = HR90)) +
  geom_point(alpha = .2) +
  geom_smooth(method = "lm", se = FALSE, color = "red", size = 1) #This ask for a geom

## `geom_smooth()` using formula 'y ~ x'
```



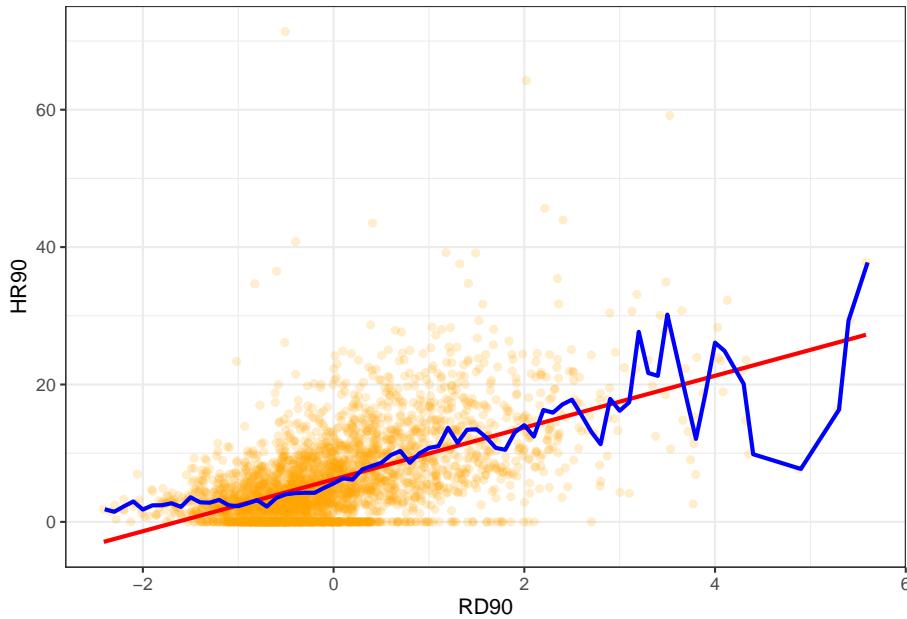
What that line is doing is giving you guesses (predictions) for the values of homicide based in the information that we have about the level of resource deprivation. It gives you one possible guess for the value of homicide for every possible value of resource deprivation and links them all together in a straight line.

The linear model then is a model that takes the form of the equation of a straight line through the data. The line does not go through all the points. In fact, you can see is a slightly less accurate representation than the (smoothed) conditional means:

```
## Warning: Ignoring unknown parameters: fun.y

## `geom_smooth()` using formula 'y ~ x'

## No summary function supplied, defaulting to `mean_se()`
```



Our regression line underpredicts at low levels of resource deprivation and does not seem to capture well the variability at higher levels of resource deprivation. But imperfect as a model as it might be it simplifies well the overall growing trend for homicide as resource deprivation increases.

As De Veaux et al (2012: 179) highlight: “like all models of the real world, the line will be wrong, wrong in the sense that it can’t match reality exactly. But it can help us understand how the variables are associated”. A map is never a perfect representation of the world, the same happens with statistical models. Yet, as with maps, models can be helpful.

8.3 Fitting a simple regression model

In order to draw a regression line (or in fact any line in a Cartesian coordinate system) we need to know two things:

- 1) We need to know where the line begins, what is the value of Y (our dependent variable) when X (our independent variable) is 0, so that we have a point from which to start drawing the line. The technical name for this point is the **intercept**.
- 2) And we need to know what is the **slope** of that line, that is, how inclined the line is, the angle of the line.

If you recall from elementary algebra (and you may not), the equation for any straight line is: $y = mx + b$

In statistics we use a slightly different notation, although the equation remains the same: $y = b_0 + b_1x$

We need the origin of the line (b_0) and the slope of the line (b_1). How does R get the intercept and the slope for the green line? How does R know where to draw this line? We need to estimate these **parameters** (or **coefficients**) from the data. How? We don't have the time to get into these more mathematical details now. You should study the required reading to understand this (*required means it is required, it is not optional*)¹. For now, suffice to say that for linear regression modes like the one we cover here, when drawing the line, R tries to minimise the distance from every point in the scatterplot to the regression line using a method called **least squares estimation**.

In order to fit the model we use the `lm()` function using the formula specification ($Y \sim X$). Typically you want to store your regression model in a “variable”, let's call it `fit_1`:

```
fit_1 <- lm(HR90 ~ RD90, data = ncovr)
```

You will see in your R Studio global environment space that there is a new object called `fit_1` with 12 elements on it. We can get a sense for what this object is and includes using the functions we introduced in previous weeks:

```
class(fit_1)
```

```
## [1] "lm"
```

```
attributes(fit_1)
```

```
## $names
##  [1] "coefficients"   "residuals"      "effects"       "rank"
##  [5] "fitted.values"  "assign"        "qr"           "df.residual"
##  [9] "xlevels"         "call"          "terms"        "model"
##
## $class
## [1] "lm"
```

R is telling us that this is an object of class `lm` and that it includes a number of attributes. One of the beauties of R is that you are producing all the results from running the model, putting them in an object, and then giving you the opportunity for using them later on. If you want to simply see the basic results from running the model you can use the `summary()` function.

¹This is a fine chapter too if you struggle with the explanations in the required reading. Many universities, like the University of Manchester, have full access to Springer ebooks. You can also have a look at these notes.

```
summary(fit_1)

##
## Call:
## lm(formula = HR90 ~ RD90, data = ncovr)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -17.796 -3.415 -0.719  2.540 67.103
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.18286   0.09844  62.81 <2e-16 ***
## RD90        3.77121   0.09846  38.30 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.468 on 3083 degrees of freedom
## Multiple R-squared:  0.3224, Adjusted R-squared:  0.3222
## F-statistic: 1467 on 1 and 3083 DF,  p-value: < 2.2e-16
```

Or if you prefer more parsimonious presentation you could use the `display()` function of the `arm` package:

```
arm::display(fit_1)

##
## lm(formula = HR90 ~ RD90, data = ncovr)
##           coef.est coef.se
## (Intercept) 6.18      0.10
## RD90        3.77      0.10
## ---
## n = 3085, k = 2
## residual sd = 5.47, R-Squared = 0.32
```

For now I just want you to focus on the numbers in the “Estimate” (or `coef.est`) column. The value of 6.18 estimated for the **intercept** is the “predicted” value for Y when X equals zero. This is the predicted value of the fear of crime score *when resource deprivation has a value of zero*.

```
summary(ncovr$RD90)

##
##    Min. 1st Qu. Median Mean 3rd Qu. Max.
## -2.4103 -0.6667 -0.2016 0.0000 0.4393 5.5831
```

RD90 is a variable that has been centered in 0. It has been created by the researchers in such a way that it has a mean value of 0. Since we only have one explanatory variable in the model this corresponds to the mean of the homicide rate, 6.18. In many other contexts the intercept has less of a meaning.

We then need the b1 regression coefficient for our independent variable, the value that will shape the **slope** in this scenario. This value is 3.77. This estimated regression coefficient for our independent variable has a convenient interpretation. When the value is positive, it tells us that *for every one unit increase in X there is a b1 increase on Y*. If the coefficient is negative then it represents a decrease on Y. Here, we can read it as “for every one unit increase in the resource deprivation score, there is a 3.77 unit increase in the homicide rate.”

Knowing these two parameters not only allows us to draw the line, we can also solve for any given value of X. Let’s go back to our guess-the-homicide-rate game. Imagine I tell you the level of resource deprivation is 1. What would be your best bet now? We can simply go back to our regression line equation and insert the estimated parameters:

```
y = b_0 + b_1x$  
y = 6.18 + 3.77 \times 1  
y = 9.95
```

Or if you don’t want to do the calculation yourself, you can use the **predict** function (differences are due to rounding error):

```
predict(fit_1, data.frame(RD90 = c(1))) #First you name your stored model and then you  
  
##           1  
## 9.954065
```

This is the expected value of Y, homicide rate, when X, resource deprivation is 1 **according to our model** (according to our simplification of the real world, our simplification of the whole cloud of points into just one straight line). Look back at the scatterplot we produced earlier with the red line. Does it look as if the green line when X is 1 corresponds to a value of Y of 9.95?

8.4 Residuals revisited: R squared

In the output above we saw there was something called the residuals. The residuals are the differences between the observed values of Y for each case minus the predicted or expected value of Y, in other words the distances between each point in the dataset and the regression line (see the visual example below).

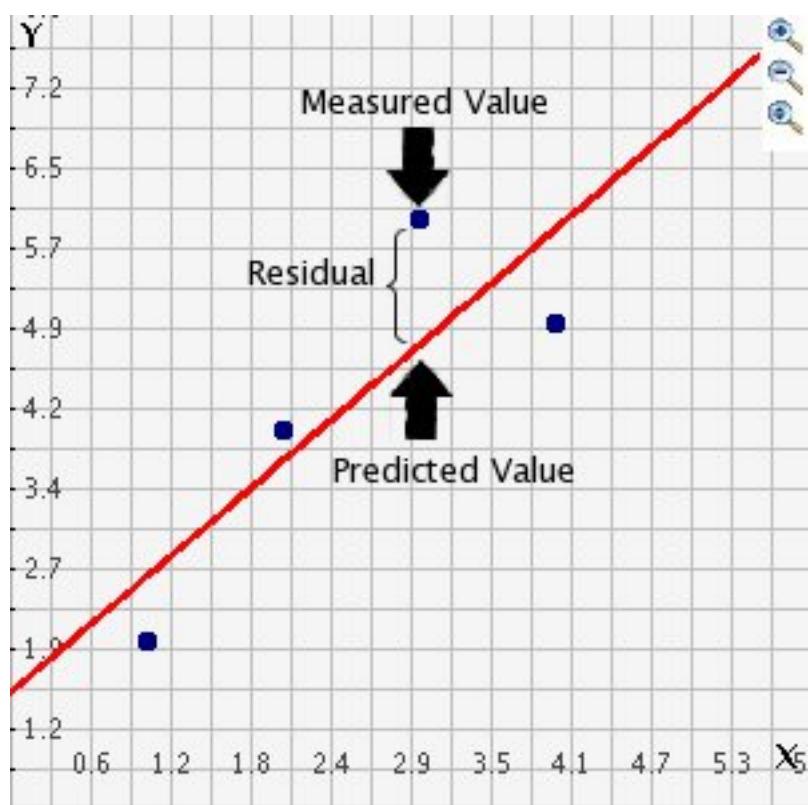


Figure 8.1: drawing

You see that we have our line, which is our predicted values, and then we have the black dots which are our actually observed values. The distance between them is essentially the amount by which we were wrong, and all these distances between observed and predicted values are our residuals. Least square estimation essentially aims to reduce the average of the squares of all these distances: that's how it draws the line.

Why do we have residuals? Well, think about it. The fact that the line is not a perfect representation of the cloud of points makes sense, doesn't it? You cannot predict perfectly what the value of Y is for every observation just by looking ONLY at their level of resource deprivation! This line only uses information regarding resource deprivation. This means that there's bound to be some difference between our predicted level of homicide given our knowledge of deprivation (the regression line) and the actual level of homicide (the actual location of the points in the scatterplot). There are other things that matter not being taken into account by our model to predict the values of Y. There are other things that surely matter in terms of understanding homicide. And then, of course, we have measurement error and other forms of noise.

We can re-write our equation like this if we want to represent each value of Y (rather than the predicted value of Y) then: $y = b_0 + b_1x + \text{residuals}$

The residuals capture how much variation is unexplained, how much we still have to learn if we want to understand variation in Y. A good model tries to maximise explained variation and reduce the magnitude of the residuals.

We can use information from the residuals to produce a measure of effect size, of how good our model is in predicting variation in our dependent variables. Remember our game where we try to guess homicide (Y)? If we did not have any information about X our best bet for Y would be the mean of Y. The regression line aims to improve that prediction. By knowing the values of X we can build a regression line that aims to get us closer to the actual values of Y (look at the Figure below).

The distance between the mean (our best guess without any other piece of information) and the observed value of Y is what we call the **total variation**. The residual is the difference between our predicted value of Y and the observed value of Y. This is what we cannot explain (i.e., variation in Y that is *unexplained*). The difference between the mean value of Y and the expected value of Y (the value given by our regression line) is how much better we are doing with our prediction by using information about X (i.e., in our previous example it would be variation in Y that can be *explained* by knowing about resource deprivation). How much closer the regression line gets us to the observed values. We can then contrast these two different sources of variation (explained and unexplained) to produce a single measure of how good our model is. The formula is as follows:

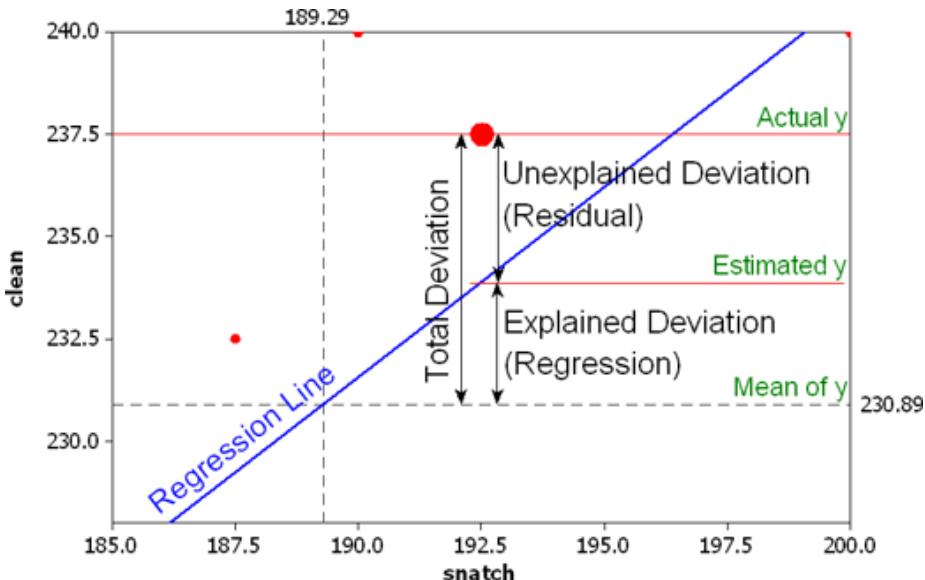


Figure 8.2: r squared

$$R^2 = \frac{SSR}{SST} = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

All this formula is doing is taking a ratio of the explained variation (the squared differences between the regression line and the mean of Y for each observation) by the total variation (the squared differences of the observed values of Y for each observation from the mean of Y). This gives us a measure of the **percentage of variation in Y that is “explained” by X**. If this sounds familiar is because it is a measure similar to eta squared (η^2) in ANOVA.

As then we can take this value as a measure of the strength of our model. If you look at the R output you will see that the R2 for our model was .32 (look at the multiple R square value in the output) . We can say that our model explains 32% of the variance in the fear of homicide. when doing regression, you will often find that regression models with aggregate data such as county level data will give you better results than when dealing with individuals. It is much harder understanding individual variation than county level variation.

```
#As an aside, and to continue emphasising your appreciation of the object oriented nature of R, we can look at the structure of the summary object:
```

```
## $names
## [1] "call"           "terms"          "residuals"       "coefficients"
## [5] "aliased"        "sigma"          "df"              "r.squared"
## [9] "adj.r.squared"  "fstatistic"     "cov.unscaled"
```

```

##  

## $class  

## [1] "summary.lm"  

#This means that we can access its elements if so we wish. So, for example, to obtain .  

summary(fit_1)$r.squared  

## [1] 0.3224335

```

Knowing how to interpret this is important. R^2 ranges from 0 to 1. The greater it is the more powerful our model is, the more explaining we are doing, the better we are able to account for variation in our outcome Y with our input. In other words, the stronger the relationship is between Y and X. As with all the other measures of effect size interpretation is a matter of judgement. You are advised to see what other researchers report in relation to the particular outcome that you may be exploring.

Weisburd and Britt (2009: 437) suggest that in criminal justice you rarely see values for R^2 greater than .40. Thus, if your R^2 is larger than .40, you can assume you have a powerful model. When, on the other hand, R^2 is lower than .15 or .2 the model is likely to be viewed as relatively weak. Our observed r squared here is rather poor. There is considerably room for improvement if we want to develop a better model to explain fear of violent crime². In any case, many people would argue that R^2 is a bit overrated. You need to be aware of what it measures and the context in which you are using it. Read here for some additional detail.

8.5 Inference with regression

In real applications, we have access to a set of observations from which we can compute the least squares line, but the population regression line is unobserved. So our regression line is one of many that could be estimated. A different sample would produce a different regression line. The same sort of ideas that we introduced when discussing the estimation of sample means or proportions also apply here. If we estimate b_0 and b_1 from a particular sample, then our estimates won't be exactly equal to b_0 and b_1 in the population. But if we could average the estimates obtained over a very large number of data sets, the average of these estimates would equal the coefficients of the regression line in the population.

We can compute standard errors for the regression coefficients to quantify our uncertainty about these estimates. These standard errors can in turn be used to produce confidence intervals. This would require us to assume that the

²This is a reasonable explanation of how to interpret R-Squared.

residuals are normally distributed. As seen in the image, and for a simple regression model, you are assuming that the values of Y are approximately normally distributed for each level of X:

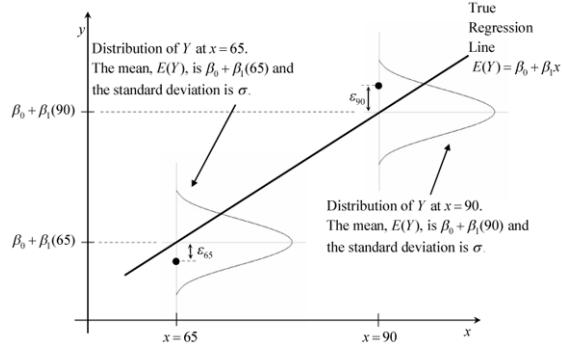


Figure 8.3: normalityresiduals

In those circumstances we can trust the confidence intervals that we can draw around the regression line as in the image below:

The dark-blue line marks the best fit. The two dark-pink lines mark the limits of the confidence interval. The light-pink lines show the sampling distributions around each of the confidence-interval limits (the many regression lines that would result from repeated sampling); notice that the best-fit line falls at the extreme of each sampling distribution.

You can also then perform standard hypothesis test on the coefficients. As we saw before when summarising the model, R will compute the standard errors and a t test for each of the coefficients.

```
summary(fit_1)$coefficients
```

```
##             Estimate Std. Error t value    Pr(>|t|)  
## (Intercept) 6.182860 0.09844166 62.80735 0.000000e+00  
## RD90        3.771206 0.09845761 38.30283 6.535551e-263
```

In our example, we can see that the coefficient for our predictor here is statistically significant³.

We can also obtain confidence intervals for the estimated coefficients using the `confint()` function:

³This blog post provides a nice animation of the confidence interval and hypothesis testing.

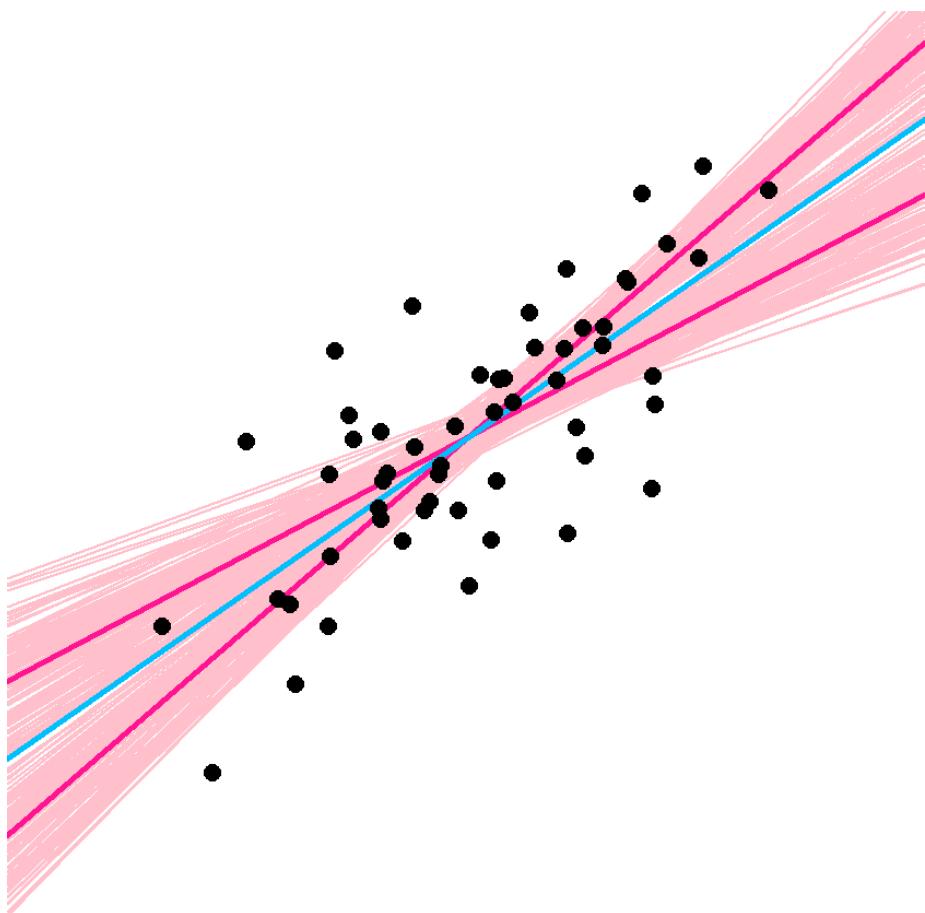


Figure 8.4: estimated

```
confint(fit_1)

##              2.5 %    97.5 %
## (Intercept) 5.989842 6.375877
## RD90         3.578156 3.964255
```

8.6 Fitting regression with categorical predictors

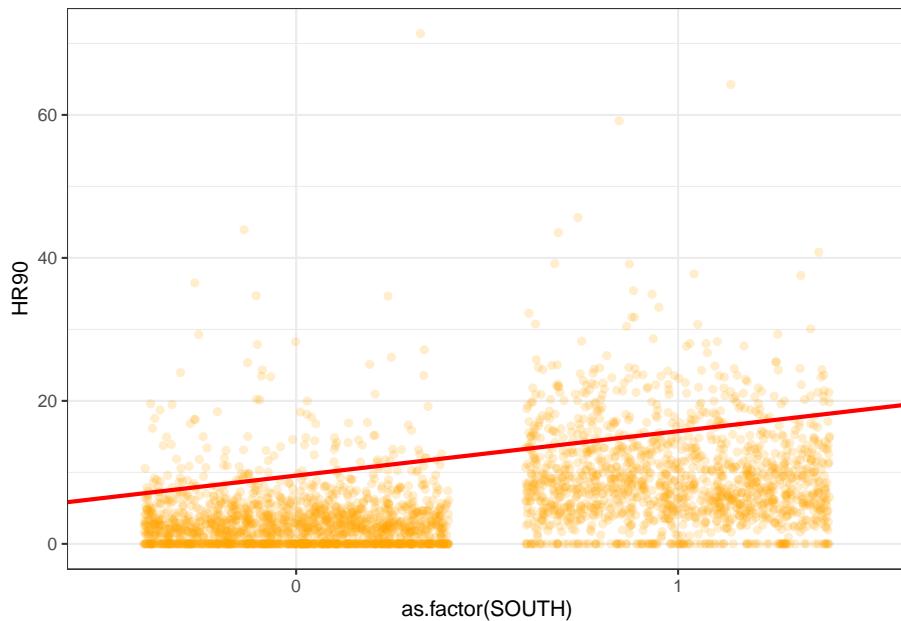
So far we have explained regression using a numeric input. It turns out we can also use regression with categorical explanatory variables. It is quite straightforward to run it.

There is only one categorical explanatory variable in this dataset, a binary indicator that indicates whether the county is in a Southern State or not. We can also explore this relationship using regression and a regression line. This is how you would express the model:

```
#We use the as.factor function to tell R that SOUTH is a categorical variable
fit_2 <- lm(HR90 ~ as.factor(SOUTH), data=ncovr)
```

Notice that there is nothing different in how we ask for the model. And see below the regression line:

```
ggplot(data=ncovr, aes(x=as.factor(SOUTH), y=HR90)) +
  geom_point(alpha=.2, position="jitter", color="orange") +
  geom_abline(intercept = fit_2$coefficients[1],
              slope = fit_2$coefficients[2], color="red", size=1) +
  theme_bw()
```



Let's have a look at the results:

```
summary(fit_2)

##
## Call:
## lm(formula = HR90 ~ as.factor(SOUTH), data = ncovr)
##
## Residuals:
##   Min     1Q Median     3Q    Max 
## -9.549 -3.342 -1.172  1.931 68.036 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 3.3416    0.1437  23.25 <2e-16 ***
## as.factor(SOUTH)1 6.2077    0.2124  29.22 <2e-16 ***
## ---      
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 5.878 on 3083 degrees of freedom
## Multiple R-squared:  0.2169, Adjusted R-squared:  0.2167 
## F-statistic: 854 on 1 and 3083 DF,  p-value: < 2.2e-16
```

As you will see the output does not look too different. But notice that in the

print out you see how the row with the coefficient and other values for our input variable `SOUTH` we see that R is printing 1. What does this mean?

It turns out that a linear regression model with just one dichotomous categorical predictor is just the equivalent of a t test. When you only have one predictor the value of the intercept is the mean value of the **reference category** and the coefficient for the slope tells you how much higher (if it is positive) or how much lower (if it is negative) is the mean value for the other category in your factor.

The reference category is the one for which R does not print the *level* next to the name of the variable for which it gives you the regression coefficient. Here we see that the named level is “1”. That’s telling you that the reference category here is “0”. If you look at the codebook you will see that 1 means the county is in a Southern state. Therefore the Y intercept in this case is the mean value of fear of violent crime for the northern counties, whereas the coefficient for the slope is telling you how much higher (since it is a positive value) the mean value is for the southern counties. Don’t believe me?

```
mean(ncovr$HR90[ncovr$SOUTH == 0], na.rm=TRUE)

## [1] 3.341614

mean(ncovr$HR90[ncovr$SOUTH == 1], na.rm=TRUE) - mean(ncovr$HR90[ncovr$SOUTH == 0], na.rm=TRUE)

## [1] 6.207679
```

So, to reiterate, for a binary predictor, the coefficient is nothing else than the difference between the mean of the two levels in your factor variable, between the averages in your two groups.

With categorical variables encoded as **factors** you always have a situation like this: a reference category and then as many additional coefficients as there are additional levels in your categorical variable. Each of these additional categories is included into the model as “**dummy**” **variables**. Here our categorical variable has two levels, thus we have only one dummy variable. There will always be one fewer dummy variable than the number of levels. The level with no dummy variable, northern counties in this example, is known as the **reference category** or the **baseline**.

It turns out then that the regression table is printing out for us a t test of statistical significance for every input in the model. If we look at the table above this t value is 29.22 and the p value associated with it is near 0. This is indeed considerably lower than the conventional significance level of 0.05. So we could conclude that the probability of obtaining this value if the null hypothesis is true is very low. The r squared is not too bad either, although lower than we saw when using resource deprivation.

8.7 Motivating multiple regression

So we have seen that we can fit models with just one predictor. We can build better models by expanding the number of predictors (although keep in mind you should also aim to build models as parsimonious as possible).

Another reason why it is important to think about additional variables in your model is to control for spurious correlations (although here you may also want to use your common sense when selecting your variables!). You must have heard before that correlation does not equal causation. Just because two things are associated we cannot assume that one is the cause for the other. Typically we see how the pilots switch the secure the belt button when there is turbulence. These two things are associated, they tend to come together. But the pilots are not causing the turbulences by pressing a switch! The world is full of **spurious correlations**, associations between two variables that should not be taking too seriously. You can explore a few here. It's funny.

Looking only at covariation between pair of variables can be misleading. It may lead you to conclude that a relationship is more important than it really is. This is no trivial matter, but one of the most important ones we confront in research and policy⁴.

It's not an exaggeration to say that most quantitative explanatory research is about trying to control for the presence of **confounders**, variables that may explain away observed associations. Think about any criminology question: Does marriage reduces crime? Or is it that people that get married are different from those that don't (and are those pre-existing differences that are associated with less crime)? Do gangs lead to more crime? Or is it that young people that join gangs are more likely to be offenders to start with? Are the police being racist when they stop and search more members of ethnic minorities? Or is it that there are other factors (i.e., offending, area of residence, time spent in the street) that, once controlled, would mean there is no ethnic disproportionality in stop and searches? Does a particular program reduces crime? Or is the observed change due to something else?

These things also matter for policy. Wilson and Kelling, for example, argued that signs of incivility (or antisocial behaviour) in a community lead to more serious forms of crime later on as people withdraw to the safety of their homes when they see those signs of incivilities and this leads to a reduction in informal mechanisms of social control. All the policies to tackle antisocial behaviour in this country are very much informed by this model and were heavily influenced by broken windows theory.

But is the model right? Sampson and Raudenbush argue it is not entirely correct. They argue, and tried to show, that there are other confounding factors (poverty, collective efficacy) that explain the association of signs of incivility and

⁴This is a nice illustration of the Simpon's Paradox, a well known example of omitted variable bias.

more serious crime. In other words, the reason why you see antisocial behaviour in the same communities that you see crime is because other structural factors explain both of those outcomes. They also argue that perceptions of antisocial behaviour are not just produced by observed antisocial behaviour but also by stereotypes about social class and race. If you believe them, then the policy implications are that only tackling antisocial behaviour won't help you to reduce crime (as Wilson and Kelling have argued). So as you can see this stuff matters for policy not just for theory.

Multiple regression is one way of checking the relevance of competing explanations. You could set up a model where you try to predict crime levels with an indicator of broken windows and an indicator of structural disadvantage. If after controlling for structural disadvantage you see that the regression coefficient for broken windows is still significant you may be into something, particularly if the estimated effect is still large. If, on the other hand, the t test for the regression coefficient of your broken windows variable is no longer significant, then you may be tempted to think that perhaps Sampson and Raudenbush were into something.

8.8 Fitting and interpreting a multiple regression model

It could not be any easier to fit a multiple regression model. You simply modify the formula in the `lm()` function by adding terms for the additional inputs.

```
ncovr$SOUTH_f <- as.factor(ncovr$SOUTH)
fit_3 <- lm(HR90 ~ RD90 + SOUTH_f, data=ncovr)
summary(fit_3)

##
## Call:
## lm(formula = HR90 ~ RD90 + SOUTH_f, data = ncovr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.480   -2.996   -0.576    2.216   68.151
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.7270     0.1394  33.90  <2e-16 ***
## RD90        2.9649     0.1108  26.77  <2e-16 ***
## SOUTH_f1    3.1809     0.2223  14.31  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##  
## Residual standard error: 5.295 on 3082 degrees of freedom  
## Multiple R-squared:  0.3647, Adjusted R-squared:  0.3642  
## F-statistic: 884.4 on 2 and 3082 DF,  p-value: < 2.2e-16
```

With more than one input, you need to ask yourself whether all of the regression coefficients are zero. This hypothesis is tested with a F test. Again we are assuming the residuals are normally distributed, though with large samples the F statistics approximates the F distribution.

You see the F test printed at the bottom of the summary output and the associated p value, which in this case is way below the conventional .05 that we use to declare statistical significance and reject the null hypothesis. At least one of our inputs must be related to our response variable.

Notice that the table printed also reports a t test for each of the predictors. These are testing whether each of these predictors is associated with the response variable when adjusting for the other variables in the model. They report the “partial effect of adding that variable to the model” (James et al. 2014: 77). In this case we can see that both variables seem to be significantly associated with the response variable.

If we look at the r squared we can now see that it is higher than before. r squared will always increase as a consequence of adding new variables, even if the new variables added are weakly related to the response variable.

We see that the coefficients for the predictors change somehow, it goes down a bit for *RD90* and it halves for *SOUTH*. **But their interpretation now changes.** A common interpretation is that now the regression for each variable tells you about changes in Y related to that variable **when the other variables in the model are held constant**. So, for example, you could say the coefficient for *RD90* represents the increase in homicide for every one-unit increase in the measure of resource deprivation *when holding all other variables in the model constant* (in this case that refers to holding constant *SOUTH*). But this terminology can be a bit misleading.

Other interpretations are also possible and are more generalizable. Gelman and Hill (2007: p. 34) emphasise what they call the *predictive interpretation* that considers how “the outcome variable differs, on average, when comparing two groups of units that differ by 1 in the relevant predictor while being identical in all the other predictors”. So if you’re regressing y on u and v, the coefficient of u is the average difference in y per difference in u, comparing pairs of items that differ in u but are identical in v.

So, for example, in this case we could say that comparing counties that have the same level of resource deprivation but that differed in whether they are South or North, the model predicts a expected difference of 3.18 in their homicide rate. And that respondents that do not vary in whether they are South or North, but that differ by one point in the level of resource deprivation, we would expect

to see a difference of 2.96 in their homicide rate. So we are interpreting the regression slopes as **comparisons of cases that differ in one predictor while being at the same levels of the other predictors**.

As you can see, interpreting regression coefficients can be kind of tricky⁵. The relationship between the response y and any one explanatory variable can change greatly depending on what other explanatory variables are present in the model.

For example, if you contrast this model with the one we run with only *SOUTH* as a predictor you will notice the intercept has changed. You cannot longer read the intercept as the mean value of homicide rate for Northern counties. *Adding predictors to the model changes their meaning*. Now the intercept index the value of homicide for southern counties that score 0 in *RD90*. In this case you have cases that meet this condition (equal zero in all your predictors), but often you may not have any case that does meet the definition of the intercept. More often than not, then, there is not much value in bothering to interpret the intercept.

Something you need to be particularly careful about is to interpret the coefficients in a causal manner. At least your data come from an experiment this is unlikely to be helpful. With observational data regression coefficients should not be read as indexing causal relations. This sort of textbook warning is, however, often neglectfully ignored by professional researchers. Often authors carefully draw sharp distinctions between causal and correlational claims when discussing their data analysis, but then interpret the correlational patterns in a totally causal way in their conclusion section. This is what is called the causation or causal creep. Beware. Don't do this, as tempting as it may be.

Comparing the simple models with this more complex model we could say that adjusting for *SOUTH* does not change much the impact of *RD90* in homicide, but that adjusting for resource deprivation halves the impact of the regional effect on homicide.

8.9 Presenting your regression results.

Communicating your results in a clear manner is incredibly important. We have seen the tabular results produced by R. If you want to use them in a paper you may need to do some tidying up of those results. There are a number of packages (`textrreg`, `stargazer`) that automatise that process. They take your `lm` objects and produce tables that you can put straight away in your reports or papers. One popular trend in presenting results is the **coefficient plot** as an alternative to the table of regression coefficients. There are various ways of producing coefficient plots with R for a variety of models. See here or here, for example.

⁵I recommend reading chapter 13 “Woes of regression coefficients” of an old book Mosteller and Tukey (1977) Data Analysis and Regression. Reading: Addison-Wesley Publishing.

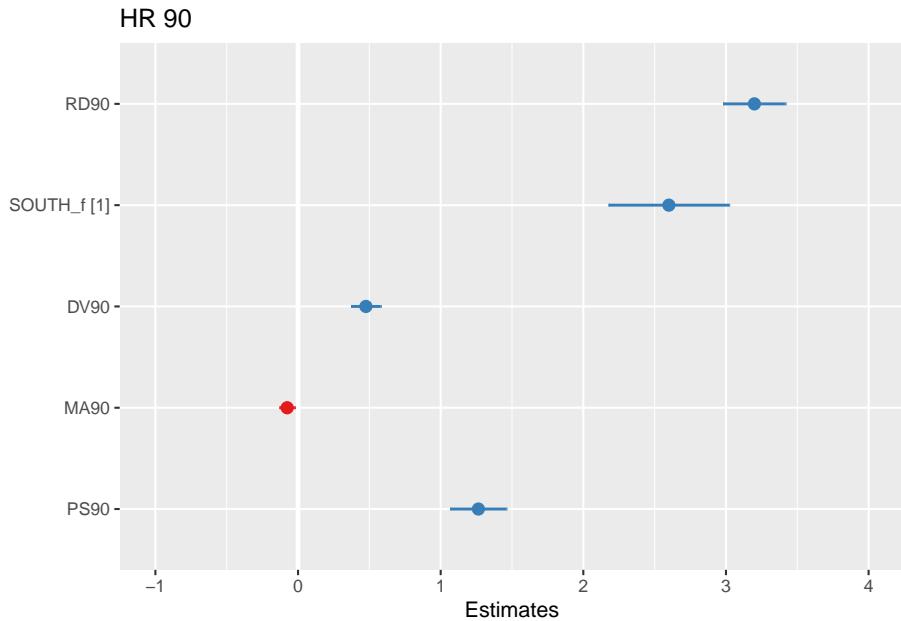
We are going to use instead the `plot_model()` function of the `sjPlot` package, that makes it easier to produce this sort of plots. You can find a more detailed tutorial about this function here. See below for an example:

```
library(sjPlot)
```

```
## #refugeeswelcome
```

Let's try with a more complex example:

```
fit_4 <- lm(HR90 ~ RD90 + SOUTH_f + DV90 + MA90 + PS90, data=ncovr)
plot_model(fit_4, breakLabelsAt = 30)
```



What you see plotted here is the point estimates (the circles), the confidence intervals around those estimates (the longer the line the less precise the estimate), and the colours represent whether the effect is negative (red) or positive (blue). There are other packages that also provide similar functionality, like the `dotwhisker` package that you may want to explore, see more details [here](#).

The `sjPlot` package also allows you to produce html tables for more professional presentation of your regression tables. For this we use the `tab_model()` function. This kind of tabulation may be particularly helpful for your final assignment.

```
tab_model(fit_4)
```

HR 90

Predictors

Estimates

CI

p

(Intercept)

4.20

2.27 – 6.13

<0.001

RD90

3.20

2.98 – 3.42

<0.001

SOUTH_f [1]

2.60

2.18 – 3.02

<0.001

DV90

0.48

0.37 – 0.58

<0.001

MA90

-0.08

-0.13 – -0.02

0.006

PS90

1.26

1.07 – 1.46

<0.001
Observations
3085
R2 / R2 adjusted
0.426 / 0.425

As before you can further customise this table. Let's change for example the name that is displayed for the dependent variable.

```
tab_model(fit_4, dv.labels = "Homicide rate 1990")
```

Homicide rate 1990
Predictors
Estimates
CI
p
(Intercept)
4.20
2.27 – 6.13
<0.001
RD90
3.20
2.98 – 3.42
<0.001
SOUTH_f [1]
2.60
2.18 – 3.02
<0.001
DV90
0.48
0.37 – 0.58
<0.001

MA90
 -0.08
 -0.13 – -0.02
 0.006
 PS90
 1.26
 1.07 – 1.46
 <0.001
 Observations
 3085
 R2 / R2 adjusted
 0.426 / 0.425

Or you could change the labels for the independent variables:

```
tab_model(fit_4, pred.labels = c("(Intercept)", "Resource deprivation", "South", "Percent divorce"))
```

Homicide rate
 Predictors
 Estimates
 CI
 p
 (Intercept)
 4.20
 2.27 – 6.13
 <0.001
 Resource deprivation
 3.20
 2.98 – 3.42
 <0.001
 South
 2.60

2.18 – 3.02

<0.001

Percent divorced males

0.48

0.37 – 0.58

<0.001

Median age

-0.08

-0.13 – -0.02

0.006

Population structure

1.26

1.07 – 1.46

<0.001

Observations

3085

R2 / R2 adjusted

0.426 / 0.425

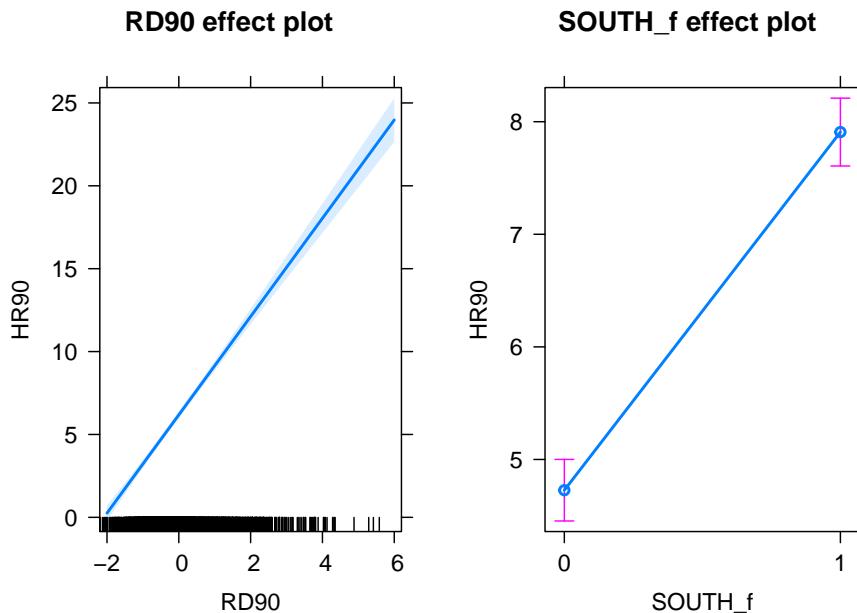
Visual display of the effects of the variables in the model are particularly helpful. The `effects` package allows us to produce plots to visualise these relationships (when adjusting for the other variables in the model). Here's an example going back to our model `fit_3` which contained *SOUTH* and *RD90* predictor variables:

```
library(effects)

## Loading required package: carData

## lattice theme set by effectsTheme()
## See ?effectsTheme for details.

plot(allEffects(fit_3), ask=FALSE)
```



Notice that the line has a confidence interval drawn around it and that the predicted means for southern and northern counties (when controlling for *RD90*) also have a confidence interval.

8.10 Rescaling input variables to assist interpretation

The interpretation or regression coefficients is sensitive to the scale of measurement of the predictors. This means one cannot compare the magnitude of the coefficients to compare the relevance of variables to predict the response variable. Let's look at the more recent model, how can we tell what predictors have a stronger effect?

```
summary(fit_4)
```

```
## 
## Call:
## lm(formula = HR90 ~ RD90 + SOUTH_f + DV90 + MA90 + PS90, data = ncovr)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -15.740  -2.588  -0.678   1.708  69.180 
## 
```

```

## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.20350   0.98475  4.269 2.03e-05 ***
## RD90        3.19923   0.11167 28.648 < 2e-16 ***
## SOUTH_f1    2.59975   0.21557 12.060 < 2e-16 ***
## DV90        0.47594   0.05308  8.967 < 2e-16 ***
## MA90        -0.07609   0.02743 -2.774 0.00557 **
## PS90         1.26451   0.10047 12.587 < 2e-16 ***
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.035 on 3079 degrees of freedom
## Multiple R-squared: 0.4262, Adjusted R-squared: 0.4252
## F-statistic: 457.3 on 5 and 3079 DF, p-value: < 2.2e-16

```

We just cannot. One way of dealing with this is by rescaling the input variables. A common method involves subtracting the mean and dividing by the standard deviation of each numerical input. The coefficients in these models is the expected difference in the response variable, comparing units that differ by one standard deviation in the predictor while adjusting for other predictors in the model.

Instead, Gelman (2008) has proposed dividing each numeric variables *by two times its standard deviation*, so that the generic comparison is with inputs equal to plus/minus one standard deviation. As Gelman explains the resulting coefficients are then comparable to untransformed binary predictors. The implementation of this approach in the `arm` package subtract the mean of each binary input while it subtract the mean and divide by two standard deviations for every numeric input.

The way we would obtain these rescaled inputs uses the `standardize()` function of the `arm` package, that takes as an argument the name of the stored fit model.

```

arm::standardize(fit_4)

##
## Call:
## lm(formula = HR90 ~ z.RD90 + c.SOUTH_f + z.DV90 + z.MA90 + z.PS90,
##      data = ncovr)
##
## Coefficients:
## (Intercept)      z.RD90      c.SOUTH_f      z.DV90      z.MA90      z.PS90
##           6.1829     6.3985     2.5998     1.6497    -0.5478     2.5290

```

Notice the main change affects the numerical predictors. The unstandardised coefficients are influenced by the degree of variability in your predictors, which

means that typically they will be larger for your binary inputs. With unstandardised coefficients you are comparing complete change in one variable (whether one is a Southern county or not) with one-unit changes in your numerical variable, which may not amount to much change. So, by putting in a comparable scale, you avoid this problem.

Standardising in the way described here will help you to make fairer comparisons. This standardised coefficients are comparable in a way that the unstandardised coefficients are not. We can now see what inputs have a comparatively stronger effect. It is very important to realise, though, that one **should not** compare standardised coefficients *across different models*.

8.11 Testing conditional hypothesis: interactions

In the social sciences there is a great interest in what are called conditional hypothesis or interactions. Many of our theories do not assume simply **additive effects** but **multiplicative effects**. For example, Wikstrom and his colleagues (2011) suggest that the threat of punishment only affects the probability of involvement on crime for those that have a propensity to offend but are largely irrelevant for people who do not have this propensity. Or you may think that a particular crime prevention programme may work in some environments but not in others. The interest in this kind of conditional hypothesis is growing.

One of the assumptions of the regression model is that the relationship between the response variable and your predictors is additive. That is, if you have two predictors x_1 and x_2 . Regression assumes that the effect of x_1 on y is the same at all levels of x_2 . If that is not the case, you are then violating one of the assumptions of regression. This is in fact one of the most important assumptions of regression, even if researchers often overlook it.

One way of extending our model to accommodate for interaction effects is to add additional terms to our model, a third predictor x_3 , where x_3 is simply the product of multiplying x_1 by x_2 . Notice we keep a term for each of the **main effects** (the original predictors) as well as a new term for the interaction effect. “Analysts should include all constitutive terms when specifying multiplicative interaction models except in very rare circumstances” (Brambor et al., 2006: 66).

How do we do this in R? One way is to use the following notation in the formula argument. Notice how we have added a third term `RD90:SOUTH_f`, which is asking R to test the conditional hypothesis that resource deprivation may have a different impact on homicide for southern and northern counties.

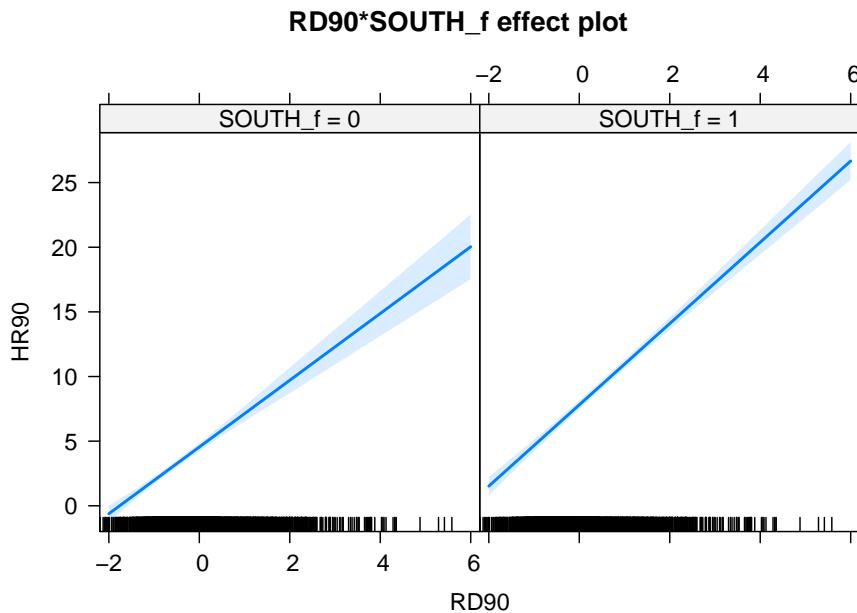
```
fit_5 <- lm(HR90 ~ RD90 + SOUTH_f + RD90:SOUTH_f , data=ncovr)
# which is equivalent to:
# fit_5 <- lm(HR90 ~ RD90 * SOUTH_f , data=ncovr)
summary(fit_5)
```

```
##
## Call:
## lm(formula = HR90 ~ RD90 + SOUTH_f + RD90:SOUTH_f, data = ncovr)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.055 -2.998 -0.566  2.227 68.136
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.5477    0.1586 28.675 <2e-16 ***
## RD90        2.5814    0.1963 13.148 <2e-16 ***
## SOUTH_f1    3.2612    0.2247 14.515 <2e-16 ***
## RD90:SOUTH_f1 0.5622    0.2377  2.365  0.0181 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.292 on 3081 degrees of freedom
## Multiple R-squared:  0.3658, Adjusted R-squared:  0.3652
## F-statistic: 592.4 on 3 and 3081 DF,  p-value: < 2.2e-16
```

You see here that essentially you have only two inputs (resource deprivation and south) but several regression coefficients. Gelman and Hill (2007) suggest reserving the term input for the variables encoding the information and to use the term predictor to refer to each of the terms in the model. So here we have two inputs and three predictors (one for SOUTH, another for resource deprivation, and a final one for the interaction effect).

In this case the test for the interaction effect is significant, which suggests there is such an interaction. Let's visualise the results with the `effects` package:

```
plot(allEffects(fit_5), ask=FALSE)
```



Notice that essentially what we are doing is running two regression lines and testing whether the slope is different for the two groups. The intercept is different, we know that Southern counties are more violent, but what we are testing here is whether the level of homicide goes up in a steeper fashion (and in the same direction) for one or the other group as the level of resource deprivation goes up. We see that's the case here. The estimated lines are almost parallel, but the slope is a bit more steep in the Southern counties. In southern counties resource deprivation seems to have more of an impact on homicide than in northern counties.

A word of warning, the moment you introduce an interaction effect the meaning of the coefficients for the other predictors changes (what it is often referred as to the “main effects” as opposed to the interaction effect). You cannot retain the interpretation we introduced earlier. Now, for example, the coefficient for the **SOUTH** variable relates the marginal effect of this variable when **RD90** equals zero. The typical table of results helps you to understand whether the effects are significant but offers little of interest that will help you to meaningfully interpret what the effects are. For this is better you use some of the graphical displays we have covered.

Essentially what happens is that the regression coefficients that get printed are interpretable only for certain groups. So now:

- The intercept still represents the predicted score of homicide for southern counties and have a score of 0 in resource deprivation (as before).

- The coefficient of *SOUTH_f1* now can be thought of as the difference between the predicted score of homicide rate for northern counties *that have a score of 0 in resource deprivation* and northern counties *that have a score of 0 in resource deprivation*.
- The coefficient of *RD90* now becomes the comparison of mean homicide rate *for southern* counties who differ by one point in resource deprivation.
- The coefficient for the interaction term represents the difference in the slope for *RD90* comparing southern and northern counties, the difference in the slope of the two lines that we visualised above.

Models with interaction terms are too often misinterpreted. I strongly recommend you read this piece by Brambor et al (2005) to understand some of the issues involved. When discussing logistic regression we will return to this and will consider tricks to ease the interpretation.

Equally, John Fox (2003) piece on the **effects** package goes to much more detail than we can here to explain the logic and some of the options that are available when producing plots to show interactions with this package.

8.12 Model building and variable selection

How do you construct a good model? This partly depends on your goal, although there are commonalities. You do want to start with theory as a way to select your predictors and when specifying the nature of the relationship to your response variable (e.g., additive, multiplicative). Gelman and Hill (2007) provide a series of general principles⁶. I would like to emphasise at this stage two of them:

- Include all input variables that, for substantive reasons, might be expected to be important in predicting the outcome.
- For inputs with large effects, consider including their interactions as well.

It is often the case that for any model, the response variable is only related to a subset of the predictors. There are some scenarios where you may be interested in understanding what is the best subset of predictors. Imagine that you want to develop a risk assessment tool to be used by police officers that respond to a domestic violence incident, so that you could use this tool for forecasting the future risk of violence. There is a cost to adding too many predictors. A police officer's time should not be wasted gathering information on predictors that are not associated with future risk. So you may want to identify the predictors that will help in this process.

⁶Look at this too.

Ideally, we would like to perform variable selection by trying out a lot of different models, each containing a different subset of the predictors. There are various statistics that help in making comparisons across models. Unfortunately, as the number of potentially relevant predictors increases the number of potential models to compare increases exponentially. So you need methods that help you in this process. There are a number of tools that you can use for **variable selection** but this goes beyond the aims of this introduction. If you are interested you may want to read this.

8.13 Regression assumptions

Although so far we have discussed the practicalities of fitting and interpreting regression models, in practical applications you want to first check your model and proceed from there. There is not much point spending time interpreting your model until you know that the model reasonably fits your data.

In previous data analysis modules we surely covered assumptions made by various statistical tests. The regression model also makes assumptions of its own. In fact, there are so many that we could spend an entire class discussing them. Gelman and Hill (2007) point out that the most important regression assumptions by decreasing order of importance are:

- **Validity.** The data should be appropriate for the question that you are trying to answer:

“Optimally, this means that the outcome measure should accurately reflect the phenomenon of interest, the model should include all relevant predictors, and the model should generalize to all cases to which it will be applied... Data used in empirical research rarely meet all (if any) of these criteria precisely. However, keeping these goals in mind can help you be precise about the types of questions you can and cannot answer reliably”

- **Additivity and linearity.** These are the most important mathematical assumptions of the model. We already talked about additivity in previous sessions and discussed how you can include interaction effects in your models if the additivity assumption is violated. We will discuss problems with non-linearities today as well as ways to diagnose and solve this problem. If the relationship is non linear (e.g, it is curvilinear) predicted values will be wrong in a biased manner, meaning that predicted values will systematically miss the true pattern of the mean of y (as related to the x -variables).
- **Independence of errors.** Regression assumes that the errors from the prediction line (or hyperplane) are independent. If there is dependency

between the observations (you are assessing change across the same units, working with spatial units, or with units that are somehow grouped such as students from the same class), you may have to use models that are more appropriate (e.g., multilevel models, spatial regression, etc.).

- **Equal variances of errors.** When the variance of the residuals is unequal, you may need different estimation methods. This is, nonetheless, considered a minor issue. There is a small effect on the validity of t-test and F-test results, but generally regression inferences are robust with regard to the variance issue.
- **Normality of errors.** The residuals should be normally distributed. Gelman and Hill (2007: 46) discuss this as the least important of the assumptions and in fact “do not recommend diagnostics of the normality of the regression residuals”. If the errors do not have a normal distribution, it usually is not particularly serious. Regression inferences tend to be robust with respect to normality (or nonnormality of the errors). In practice, the residuals may appear to be nonnormal when the wrong regression equation has been used. So, I will show you how to inspect normality of the residuals not because this is a problem on itself, but because it may be give you further evidence that there is some other problem with the model you are applying to your data.

Apart from this, it is convenient to diagnose multicollinearity (this affects interpretation) and influential observations.

So these are the assumptions of linear regression, and today we will go through how to test for them, and also what are some options that you can consider if you find that your model violates them. While finding that some of the assumptions are violated do not necessarily mean that you have to scrap your model, it is important to use these diagnostics to illustrate that you have considered what the possible issues with your model is, and if you find any serious issues that you address them.

You may have noticed the second of this assumption is independence of errors. This is an issue with spatial data. If you have spatial autocorrelation basically you are saying that your observations are not independent. What happens in area X is likely to be similar to what happens in its surrounding neighbours (if you have positive spatial autocorrelation). What do you do? Well, that’s what we will cover next week. We will learn how to fit regression models where you have spatial dependency.

8.13.1 HOMEWORK

Fit a regression model with a few relevant explanatory variables for the homicide rate in 1970. Make sure you interpret your results.

Chapter 9

Spatial regression models

9.1 Introduction

Last week we provided you with an introduction to regression analysis with R. The data we used had a spatial component. We were modelling the geographical distribution of homicide across US counties. However, we did not incorporate this spatial component into our models. As we have explained throughout the semester criminal events often cluster geographically in space. So if we want to develop a regression model for crime we may have to recognise this spatial component. Remember as well, from last week, that regression models assume independence between the observations. That is, a regression model is formally assuming that what happens in area X_i is not in any way related (it is independent) of what happens in area X_{ii} . But if those two areas are adjacent in geographical space we know that there is a good chance that this assumption may be violated. In previous weeks we covered formal tests for spatial autocorrelation, which allow us to test whether this assumption is met or not. So before we fit a regression model with spatial data we need to explore the issue of autocorrelation. We already know how to do this. In this session, we will examine the data from last week, explore whether autocorrelation is an issue, and then introduce models that allow us to take into account spatial autocorrelation. We will see that there are two basic ways of adjusting for spatial autocorrelation: through a spatial lag model or through a spatial error model.

Before we do any of this, we need to load the libraries we will use today:

```
library(sf)  
  
## Linking to GEOS 3.8.1, GDAL 2.4.4, PROJ 4.9.1
```

```

library(tmap)
library(sp)
library(spdep)

## Loading required package: spData

## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge',
## repos='https://nowosad.github.io/drat/', type='source')`
```

Then we will bring back the data from last week:

```

##R in Windows have some problems with https addresses, that's why we need to do this .
urlfile<-'https://s3.amazonaws.com/geoda/data/ncovr.zip'
download.file(urlfile, 'ncovr.zip')
#Let's unzip and create a new directory (ncovr) in our working directory to place the
unzip('ncovr.zip', exdir = 'ncovr')
```

Last week we did not treated the data as spatial and, consequently, relied on the csv file. But notice that in the unzip ncovr file there is also a shapefile that we can load as a spatial object into R:

```

shp_name <- "ncovr/ncovr/NAT.shp"
ncovr_sf <- st_read(shp_name)
```

```

## Reading layer `NAT' from data source `/Users/reka/Dropbox (The University of Manchester)'
## Simple feature collection with 3085 features and 69 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:            xmin: -124.7314 ymin: 24.95597 xmax: -66.96985 ymax: 49.37173
## CRS:             4326
```

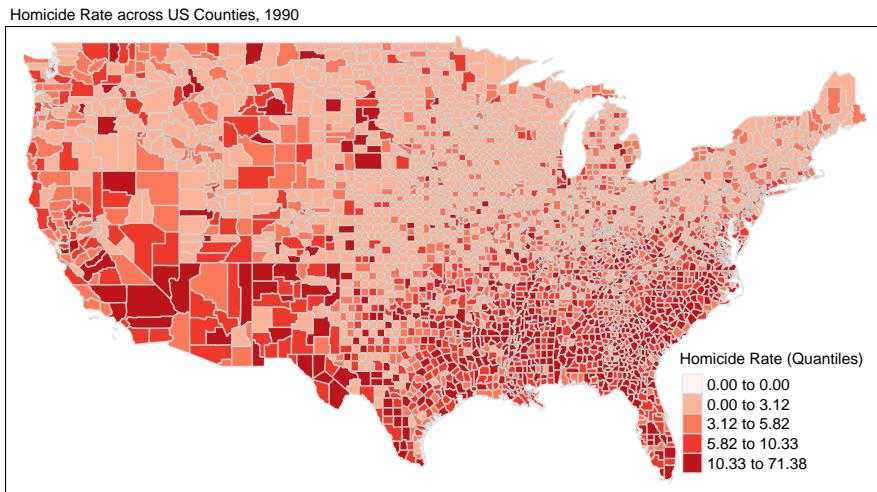
We can indeed represent our variable of interest using a choropleth map.

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

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

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

```
tm_shape(ncovr_sf) +
  tm_fill("HR90", title = "Homicide Rate (Quantiles)", style="quantile", palette = "Reds") +
  tm_borders(alpha = 0.1) +
  tm_layout(main.title = "Homicide Rate across US Counties, 1990", main.title.size = 0.7 ,
            legend.position = c("right", "bottom"), legend.title.size = 0.8)
```



Do you think there is spatial patterning for homicide?

9.2 Looking at the residuals and testing for spatial autocorrelation in regression

Residuals, as we have explained, give you an idea of the distance between our observed Y values and the predicted Y values. So in essence they are deviations of observed reality from your model. Your regression line or hyperplane is optimised to be the one that best represent your data if those assumptions are met. Residuals are very helpful to diagnose, then, whether your model is a good representation of reality or not. Most diagnostics of the assumptions for OLS regression rely on exploring the residuals.

In order to explore the residuals we need to fit our model first. Let's look at one of the models from last week.

```
fit_1 <- lm(HR90 ~ RD90 + SOUTH + DV90 + MA90 + PS90 + UE90, data=ncovr_sf)
```

Now that we have fitted the model we can extract the residuals. If you look at the `fit_1` object in your RStudio environment or if you run the `str()` function to look inside this object you will see that this object is a list with different elements, one of which is the residuals. An element of this object then includes the residual for each of your observations (the difference between the observed value and the value predicted by your model). We can extract the residuals using the `residuals()` function and add them to our spatial data set.

```
ncovr_sf$res_fit1 <- residuals(fit_1)
```

If you now look at the dataset you will see that there is a new variable with the residuals. In those cases where the residual is negative this is telling us that the observed value is lower than the predicted (that is, our model is *overpredicting* the level of homicide for that observation) when the residual is positive the observed value is higher than the predicted (that is, our model is *underpredicting* the level of homicide for that observation).

We could also extract the predicted values if we wanted. We would use the `fitted()` function.

```
ncovr_sf$fitted_fit1 <- fitted(fit_1)
```

Now look at the second county in the dataset. It has a homicide rate in 1990 of 15.88. This is the observed value. If we look at the new column we have created (“`fitted_fit1`”), our model predicts a homicide rate of 2.41. That is, knowing the level unemployment, whether the county is North or South, the level of resource deprivation, etc., we are predicting a homicide rate of 2.41. Now, this is lower than the observed value, so our model is underpredicting the level of homicide in this case. If you observed the residual you will see that it has a value of 13.46, which is simply the difference between the observed and the predicted value.

With spatial data one useful thing to do is to look at any spatial patterning in the distribution of the residuals. Notice that the residuals are the difference between the observed values for homicide and the predicted values for homicide, so you want your residual to NOT display any spatial patterning. If, on the other hand, your model displays a patterning in the areas of the study region where it performs poorly, then you may have a problem. This is telling your model is not a good representation of the social phenomena you are studying across the full study area: there is systematically more distortion in some areas than in others.

We are going to produce a choropleth map for the residuals, but we will use a common classification method we haven’t covered yet: standard deviations. Standard deviation is a statistical technique type of map based on how much

the data differs from the mean. You measure the mean and standard deviation for your data. Then, each standard deviation becomes a class in your choropleth maps.

In order to do that we will compute the mean and the standard deviation for the variable we want to plot and break the variable according to these values. The following code creates a new variable in which we will express the residuals in terms of standard deviations away from the mean. So, for each observation, we subtract the mean and divide by the standard deviation. Remember, this is exactly what the `scale` function does, which we have introduced in week 7:

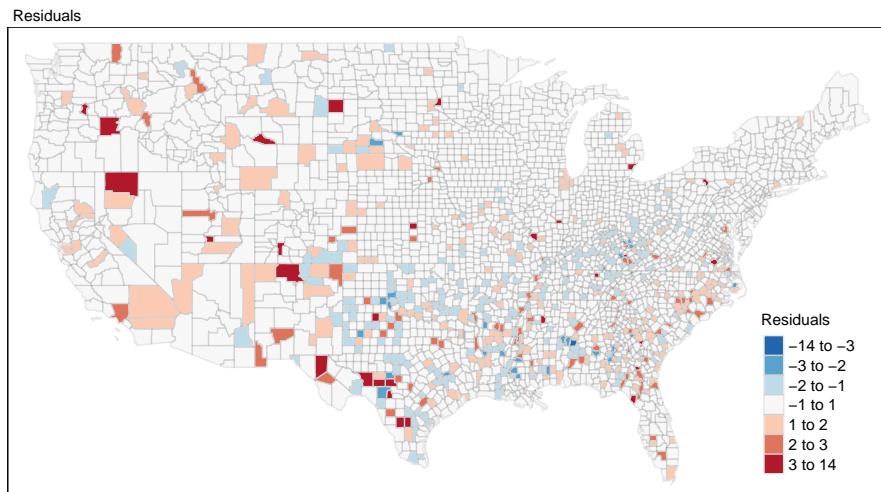
```
ncovr_sf$sd_breaks <- scale(ncovr_sf$res_fit1)[,1] # because scale is made for matrices, we just
# this is equal to (ncovr_sf$res_fit1 - mean(ncovr_sf$res_fit1)) / sd(ncovr_sf$res_fit1)
summary(ncovr_sf$sd_breaks)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## -3.5370 -0.5238 -0.1404  0.0000  0.3314 13.7407
```

Next we use a new style, `fixed`, within the `tm_fill` function. When we break the variable into classes using the `fixed` argument we need to specify the boundaries of the classes. We do this using the `breaks` argument. In this case we are going to ask R to create 7 classes based on standard deviations away from the mean. Remember that a value of 1 would be 1 standard deviation higher than the mean, and -1 respectively lower. If we assume normal distribution, then 68% of all counties should lie within the middle band from -1 to +1 (you can find a refresher of this on Wikipedia).

```
my_breaks <- c(-14, -3, -2, -1, 1, 2, 3, 14)

tm_shape(ncovr_sf) +
  tm_fill("sd_breaks", title = "Residuals", style = "fixed", breaks = my_breaks, palette = "-RdBu")
  tm_borders(alpha = 0.1) +
  tm_layout(main.title = "Residuals", main.title.size = 0.7 ,
            legend.position = c("right", "bottom"), legend.title.size = 0.8)
```



Notice the spatial patterning of areas of over-prediction (negative residuals, or blue tones) and under-prediction (positive residuals, or brown tones). This visual inspection of the residuals is telling you that spatial autocorrelation may be present here. This, however, would require a more formal test.

Remember from week 7 that in order to do this first we need to turn our `sf` object into a `sp` class object and then create the spatial weight matrix. If the code below and what it does is not clear to you, revise the notes from week 7, when we first introduced it.

```
#We coerce the sf object into a new sp object
ncovr_sp <- as(ncovr_sf, "Spatial")
#Then we create a list of neighbours using the Queen criteria
w <- poly2nb(ncovr_sp, row.names=ncovr_sp$FIPSNO)
summary(w)
```

```
## Neighbour list object:
## Number of regions: 3085
## Number of nonzero links: 18168
## Percentage nonzero weights: 0.190896
## Average number of links: 5.889141
## Link number distribution:
##
##      1      2      3      4      5      6      7      8      9      10     11     13     14 
##     24     36     91    281    620   1037    704    227     50     11      2      1      1
```

9.2. LOOKING AT THE RESIDUALS AND TESTING FOR SPATIAL AUTOCORRELATION IN REGRESSION

```
## 24 least connected regions:  
## 53009 53029 25001 44005 36103 51840 51660 6041 51790 51820 51540 51560 6075 51580 51530 51131  
## 1 most connected region:  
## 49037 with 14 links
```

This should give you an idea of the distribution of connectedness across the data, with counties having on average nearly 6 neighbours. Now we can generate the row standardise spatial weight matrix and the Moran Scatterplot.

```
wm <- nb2mat(w, style='B')  
rwm <- mat2listw(wm, style='W')
```

We obtain the Moran's test for regression residuals using the function `lm.morantest()` as below. It is important to realize that the Moran's I test statistic for residual spatial autocorrelation takes into account the fact that the variable under consideration is a residual, computed from a regression. The usual Moran's I test statistic does not. It is therefore incorrect to simply apply a Moran's I test to the residuals from the regression without correcting for the fact that these are residuals.

```
lm.morantest(fit_1, rwm, alternative="two.sided")  
  
##  
## Global Moran I for regression residuals  
##  
## data:  
## model: lm(formula = HR90 ~ RD90 + SOUTH + DV90 + MA90 + PS90 + UE90,  
## data = ncovr_sf)  
## weights: rwm  
##  
## Moran I statistic standard deviate = 10.321, p-value < 2.2e-16  
## alternative hypothesis: two.sided  
## sample estimates:  
## Observed Moran I      Expectation      Variance  
##      0.1093062514    -0.0014498532    0.0001151682
```

You will notice we obtain a statistically significant value for Moran's I. The value of the Moran's I test is not too high, but we still need to keep it in mind. If we diagnose that spatial autocorrelation is an issue, that is, that the errors (the residuals) are related systematically among themselves, then we have a problem and need to use a more appropriate approach: a spatial regression model.

9.3 What to do now?

If the test is significant (as in this case), then we possibly need to think of a more suitable model to represent our data: a spatial regression model. Remember spatial dependence means that (more typically) there will be areas of spatial clustering for the residuals in our regression model. So our predicted line (or hyperplane) will systematically under-predict or over-predict in areas that are close to each other. That's not good. We want a better model that does not display any spatial clustering in the residuals.

There are two general ways of incorporating spatial dependence in a regression model, through what we called a **spatial error model** or by means of a **spatially lagged model**. There are `spdep` functions that provides us with some tools to help us make a decision as to which of these two is most appropriate: the **Lagrange Multiplier tests**.

The difference between these two models is both technical and conceptual. The **spatial error model** treats the spatial autocorrelation as a nuisance that needs to be dealt with. A spatial error model basically implies that the:

“spatial dependence observed in our data does not reflect a truly spatial process, but merely the geographical clustering of the sources of the behaviour of interest. For example, citizens in adjoining neighborhoods may favour the same (political) candidate not because they talk to their neighbors, but because citizens with similar incomes tend to cluster geographically, and income also predicts vote choice. Such spatial dependence can be termed attributional dependence” (Darmofal, 2015: 4)

The **spatially lagged model**, on the other hand, incorporates spatial dependence explicitly by adding a “spatially lagged” variable y on the right hand side of our regression equation. Its distinctive characteristic is that it includes a spatially lagged “dependent” variable among the explanatory factors. It’s basically explicitly saying that the values of y in the neighbouring areas of observation $n \sim i$ is an important predictor of y on each individual area $n \sim i$. This is one way of saying that the spatial dependence may be produced by a spatial process such as the diffusion of behaviour between neighboring units:

“If so the behaviour is likely to be highly social in nature, and understanding the interactions between interdependent units is critical to understanding the behaviour in question. For example, citizens may discuss politics across adjoining neighbours such that an increase in support for a candidate in one neighbourhood directly leads to an increase in support for the candidate in adjoining neighbourhoods” (Darmofal, 2015: 4)

9.4 Spatial Regimes

Before we proceed to a more detailed description of these two models it is important that we examine another aspect of our model that also links to geography. Remember that when we brought up our data into R, we decided to test for the presence of an interaction. We looked at whether the role of unemployment was different in Southern and Northern states. We found that this interaction was indeed significant. Unemployment had a more significant effect in Southern than in Northern states. This was particularly obvious during the 1970s, when unemployment did not affect homicide rates in the Northern states, but it did lead to a decrease in homicide in the Southern states.

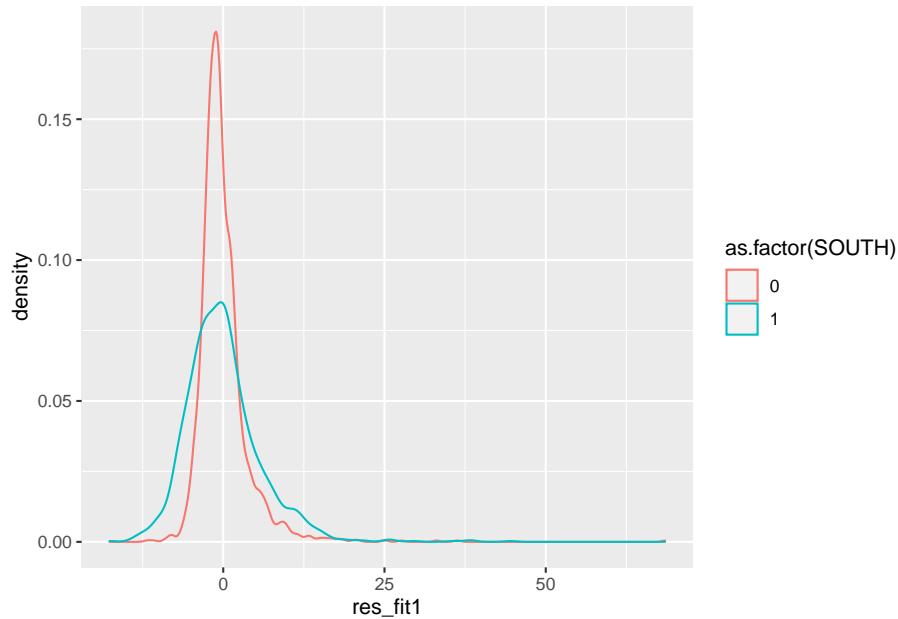
We could have attempted to test other interaction effects between some of our other predictors and their geographical location in the South or the North. But we did not.

If you have read the Ballen et al. (2001) paper that we are replicating in the lab last week and this week, you will have noticed that they decided that they needed to run separate models for the South and the North. This kind of situation, where sub-regions seem to display different patterns often is alluded-with the name of spatial regimes. In the context of regression analysis, spatial regimes relates to the possibility that we may need to split our data into two (or more sub-regions) in order to run our models, because we presume that the relationship of the predictors to the outcome may play out differently in these sub-regions (spatial regimes).

So how can we assess whether this is an issue in our data? As with many other diagnostics of regression, you may want to start by looking at your residuals. Look at the residual map we produced earlier. Do you think that the residuals look different in the South and in the North? If the pattern is not clear to you, you may want to run other forms of visualisation.

```
library(ggplot2)

ggplot(ncovr_sf, aes(x = res_fit1, colour = as.factor(SOUTH))) +
  geom_density()
```



HOMEWORK 9.1

What do you see in this plot? And, critically, what does it mean? What is this telling you about the predicted values that result from our model? (Remember what a residual is: the difference between the observed values and the predicted values).

There are formal tests that one can use to further explore these issues. The paper by Bollen et al. (2001) mentions them (Chow tests). But those are beyond the scope of this course. Sufficient to say that, as Bollen et al. (2001), we are going to split our analysis and run them separately for the Southern and the Northern states. We have covered the `filter()` function from `dplyr` to split datasets based on values of a variable. But to split `sf` objects it is better to rely on the more generic `subset` function, since `filter()` doesn't accommodate well the column with the geographic information that `sf` provides.

```
ncovr_s_sf <- subset(ncovr_sf, SOUTH == 1)
ncovr_n_sf <- subset(ncovr_sf, SOUTH == 0)
```

9.5 Lagrange multipliers

The Moran's I test statistic has high power against a range of spatial alternatives. However, it does not provide much help in terms of which alternative model would be most appropriate. The Lagrange Multiplier test statistics do allow a distinction between spatial error models and spatial lag models.

In order to practice their computation and interpretation, let's run two separate OLS regression models (one for the South and one for the North), using the same predictors as we used last week and, first, focusing on homicide in the northern counties in the earliest year for which we have data (1960). We have split the data in two, so that means that before we do this we need to create new files for the spatial weight matrix: in particular we will create one using first order queen criteria.

```
#We coerce the sf object into a new sp object
ncovr_n_sp <- as(ncovr_n_sf, "Spatial")
#Then we create a list of neighbours using the Queen criteria
w_n <- poly2nb(ncovr_n_sp, row.names=ncovr_n_sp$FIPSNO)
wm_n <- nb2mat(w_n, style='B')
rwm_n <- mat2listw(wm_n, style='W')

fit_2 <- lm(HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data=ncovr_n_sf)
```

First look at the Moran's I.

```
lm.morantest(fit_2, rwm_n, alternative="two.sided")

##
## Global Moran I for regression residuals
##
## data:
## model: lm(formula = HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data =
## ncovr_n_sf)
## weights: rwm_n
##
## Moran I statistic standard deviate = 2.84, p-value = 0.004511
## alternative hypothesis: two.sided
## sample estimates:
## Observed Moran I      Expectation      Variance
##      0.0394654814    -0.0022539192    0.0002157889
```

The p (probability) value associated with this Moran's I is below our standard threshold. So we will say that we have an issue with spatial autocorrelation that we need to deal with. OLS regression won't do. In order to decide whether to fit a spatial error or a spatially lagged model we need to run the Lagrange Multipliers.

Both Lagrange multiplier tests (for the error and the lagged models, **LMerr** and **LMlag** respectively), as well as their robust forms (**RLMerr** and **RLMLag**, also respectively) are included in the **lm.LMtests** function. Again, a regression object and a spatial **listw** object must be passed as arguments. In addition,

the tests must be specified as a character vector (the default is only LMerr), using the `c()` operator (concatenate), as illustrated below.

```
lm.LMtests(fit_2, rwm_n, test = c("LMerr", "LMlag", "RLMerr", "RLMlag", "SARMA"))

## Lagrange multiplier diagnostics for spatial dependence
##
## data:
## model: lm(formula = HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data =
## ncovr_n_sf)
## weights: rwm_n
##
## LMerr = 7.1326, df = 1, p-value = 0.007569
##
##
## Lagrange multiplier diagnostics for spatial dependence
## data:
## model: lm(formula = HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data =
## ncovr_n_sf)
## weights: rwm_n
##
## LMlag = 16.595, df = 1, p-value = 4.627e-05
##
##
## Lagrange multiplier diagnostics for spatial dependence
## data:
## model: lm(formula = HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data =
## ncovr_n_sf)
## weights: rwm_n
##
## RLMerr = 8.2906, df = 1, p-value = 0.003985
##
##
## Lagrange multiplier diagnostics for spatial dependence
## data:
## model: lm(formula = HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data =
## ncovr_n_sf)
## weights: rwm_n
##
## RLMlag = 17.753, df = 1, p-value = 2.515e-05
##
```

```

## 
## Lagrange multiplier diagnostics for spatial dependence
##
## data:
## model: lm(formula = HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data =
## ncovr_n_sf)
## weights: rwm_n
##
## SARMA = 24.886, df = 2, p-value = 3.946e-06

```

How do we interpret the Lagrange Multipliers? First we look at the standard ones (**LMerr** and **LMlag**). **If both** are below the .05 level this means we need to have a look at the robust version of these tests (Robust LM).

If the non-robust version is not significant, the mathematical properties of the robust tests may not hold, so we don't look at them in those scenarios. It is fairly common to find that both the lag (**LMlag**) and the error (**LMerr**) non-robust LM are significant. If only one of them were: problem solved. We would choose a spatial lag or a spatial error model according to this (i.e., if the lag LM was significant and the error LM was not we would run a spatial lag model or viceversa). Here we see that both are significant, thus, we need to inspect their robust versions.

Yet we look at the robust Lagrange multipliers (**RLMlag** and **RLMerr**) and encounter that both are significant again. What do we do? Luc Anselin (2008: 199-200) proposes the following criteria:

“When both LM test statistics reject the null hypothesis, proceed to the bottom part of the graph and consider the Robust forms of the test statistics. Typically, only one of them will be significant, or one will be orders of magnitude more significant than the other (e.g., $p < 0.00000$ compared to $p < 0.03$). In that case the decision is simple: estimate the spatial regression model matching the (most) significant” robust “statistic. In the rare instance that both would be highly significant, go with the model with the largest value for the test statistic. However, in this situation, some caution is needed, since there may be other sources of misspecification. One obvious action to take is to consider the results for different spatial weight and/or change the basic (i.e., not the spatial part) specification of the model. there are also rare instances where neither of the Robust LM test statistics are significant. In those cases, more serious specification problems are likely present and those should be addressed first”.

By other specification errors Prof. Anselin refers to problems with some of the other assumptions of regression that we covered last week.

Ok, so let's go back to our results and keep this criteria in mind. Both Robust LM are significant, but we can see that the test for the lag model is several orders more significant than for the error model ($p < 0.0000251$ vs $p < 0.004$). Notice as well that the test value is higher for the lag model (18 versus 8). In

this case we would say that the Lagrange Multiplier tests suggest we estimate a spatial lag model rather than a spatial error model.

HOMEWORK 9.2 *Run a OLS regression model for homicide rate in 1980 for the Southern states using the covariates we have been using so far and make sure that you ask for the spatial dependence tests and the Lagrange Multipliers (attach the results). What model do you think need to be estimated in this case? The spatial lag or the spatial error model*

9.6 Fitting and interpreting a spatially lagged model

Maximum Likelihood (ML) estimation of the spatial lag model is carried out with the `lagsarlm()` function. The required arguments are a regression “formula”, a data set and a `listw` spatial weights object. The default method uses Ord’s eigenvalue decomposition of the spatial weights matrix.

```
fit_2_lag <- lagsarlm(HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data=ncovr_n_sf, rwm_n)

## Warning: Function lagsarlm moved to the spatialreg package

## Warning in lagsarlm(HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data =
## ncovr_n_sf, : install the spatialreg package

## Warning: Function can.be.simmed moved to the spatialreg package

## Warning in can.be.simmed(listw): install the spatialreg package

## Warning: Function jacobianSetup moved to the spatialreg package

## Warning in jacobianSetup(method, env, con, pre_eig = con$pre_eig, trs = trs, :
## install the spatialreg package

## Warning: Function eigen_setup moved to the spatialreg package

## Warning in eigen_setup(env, which = which): install the spatialreg package

## Warning: Function as_dgRMatrix_listw moved to the spatialreg package

## Warning in as_dgRMatrix_listw(from): install the spatialreg package
```



```
## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(rho, env): install the spatialreg package

## Warning: Method summary.sarlm moved to the spatialreg package

## Warning in summary.sarlm(fit_2_lag): install the spatialreg package

## Warning: Method Wald1.sarlm moved to the spatialreg package

## Warning in Wald1.sarlm(object): install the spatialreg package

## Warning: Method logLik.sarlm moved to the spatialreg package

## Warning in logLik.sarlm(object): install the spatialreg package

## Warning: Method residuals.sarlm moved to the spatialreg package
```

```

## Warning in residuals.sarlm(object): install the spatialreg package

## Warning: Method LR1.sarlm moved to the spatialreg package

## Warning in LR1.sarlm(object): install the spatialreg package

## Warning: Method logLik.sarlm moved to the spatialreg package

## Warning in logLik.sarlm(object): install the spatialreg package

## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(object): install the spatialreg package

## Warning: Method print.summary.sarlm moved to the spatialreg package

## Warning in print.summary.sarlm(x): install the spatialreg package

## 
## Call:
## lagsarlm(formula = HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data = ncovr_n_sf,
##           listw = rwm_n)
##
## Residuals:

## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(x): install the spatialreg package

##      Min       1Q   Median       3Q      Max
## -11.86076 -1.54308 -0.55531  0.76832 28.30435
##
## Type: lag
## Coefficients: (asymptotic standard errors)
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept) 5.802071  0.677236 8.5673 < 2.2e-16
## RD60        1.734746  0.157771 10.9954 < 2.2e-16
## DV60        1.002757  0.077951 12.8640 < 2.2e-16
## MA60        -0.179328  0.020024 -8.9557 < 2.2e-16
## PS60         0.382956  0.077193  4.9610 7.014e-07
## UE60         0.087963  0.030109  2.9215  0.003483
##
## Rho: 0.13749, LR test value: 15.139, p-value: 9.9874e-05
## Asymptotic standard error: 0.035631
##      z-value: 3.8587, p-value: 0.000114
## Wald statistic: 14.89, p-value: 0.000114

```

```

## Warning: Method logLik.sarlm moved to the spatialreg package

## Warning in logLik.sarlm(x): install the spatialreg package

## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(object): install the spatialreg package

## 
## Log likelihood: -4273.43 for lag model
## ML residual variance (sigma squared): 9.6542, (sigma: 3.1071)
## Number of observations: 1673
## Number of parameters estimated: 8

## Warning: Method logLik.sarlm moved to the spatialreg package

## Warning in logLik.sarlm(object): install the spatialreg package

## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(object): install the spatialreg package

## AIC: 8562.9, (AIC for lm: 8576)
## LM test for residual autocorrelation
## test value: 10.659, p-value: 0.0010954

```

As expected, the spatial autoregressive parameter (Rho) is highly significant, as indicated by the p-value of 0.000114 on an asymptotic t-test (based on the asymptotic variance matrix). The Likelihood Ratio test (LR) on this parameter is also highly significant (p value 9.9874e-05).

Just to reiterate, we run the spatial lag model not only because the Lagrange Multiplier suggests this may be appropriate, but also because in this case we believe that the values of y in one county, i , are directly influenced by the values of y that exist in the “neighbors” of i . This is an influence that goes beyond other explanatory variables that are specific to i . Remember what we said earlier in the spatial lag model we are simply adding as an additional explanatory variable the values of y in the surrounding area. What we mean by “surrounding” will be defined by our spatial weight matrix. It’s important to emphasise that one has to think very carefully and explore appropriate definitions of “surrounding” (as we discussed, though just superficially, in the section on spatial clustering a few weeks ago). We are using here the first order queen criteria, but in real practice you would need to explore whether this is the best definition and one that makes theoretical sense.

How do you interpret these results? First you need to look at the general measures of fit of the model. I know what you are thinking. Look at the R Square and compare them, right? Well, don't. This R Square is not a real R Square, but a pseudo-R Square and therefore is not comparable to the one we obtain from the OLS regression model. Instead we can look at the Akaike Information Criterion (AIC). We see that the lag model has a AIC of 8562.9 whereas the linear model with no lags has a AIC of 8576, so this is telling us there is a better fit when we include the spatial lag.

In our spatial lag model you will notice that there is a new term Rho. What is this? This is our spatial lag. It is a variable that measures the homicide rate in the counties that are defined as surrounding each county in our spatial weight matrix. We are simply using this variable as an additional explanatory variable to our model, so that we can appropriately take into account the spatial clustering detected by our Moran's I test. You will notice that the estimated coefficient for this term is both positive and statistically significant. In other words, when the homicide rate in surrounding areas increases, so does the homicide rate in each country, even when we adjust for the other explanatory variables in our model. The fact the lag is significant adds further evidence that this is a better model than the OLS regression specification.

You also see at the bottom further tests for spatial dependence, a likelihood ratio test. This is not a test for residual spatial autocorrelation after we introduce our spatial lag. What you want is for this test to be significant because in essence is further evidence that the spatial lag model is a good fit.

How about the coefficients? It may be tempting to look at the regression coefficients for the other explanatory variables for the original OLS model and compare them to those in the spatial lag model. But you should be careful when doing this. Their meaning now has changed:

"Interpreting the substantive effects of each predictor in a spatial lag model is much more complex than in a nonspatial model (or in a spatial error model) because of the presence of the spatial multiplier that links the independent variables to the dependent. In the nonspatial model, it does not matter which unit is experiencing the change on the independent variable. The effect" in the dependent variable "of a change" in the value of an independent variable "is constant across all observations" (Darmofal, 2015: 107).

Remember we say, when interpreting a regression coefficient for variable X_{-i} , that they indicate how much Y goes up or down for every one unit increase in X_{-i} when holding all other variables in the model constant. In our example, for the nonspatial model this effect is the same for every county in our dataset. But in the spatial lag model things are not the same. We cannot interpret the regression coefficients for the substantive predictors in the same way because the "substantive effects of the independent variables vary by observation as a result of the different neighbors for each unit in the data" (Darmofal, 2015: 107).

In the OLS regression model, the coefficients for any of the explanatory variables

measure the absolute impact of these variables. It is a simpler scenario. We look at the effect of X in Y within each county. So X in county A affects Y in county A. In the spatial lag model there are two components to how X affect Y. X affects Y within each county directly but remember we are also including the spatial lag, the measure of Y in the surrounding counties (call them B, C, and D). So our model includes not only the effect of X in county A in the level of Y in county A. By virtues of including the spatial lag (a measure of Y in county B, C and D) we are indirectly incorporating as well the effects that X has on Y in counties B, C, and D. So the effect of a covariate (independent variable) is the sum of two particular effects: a direct, local effect of the covariate in that unit, and an indirect, spillover effect due to the spatial lag.

In other words, in the spatial lag model, the coefficients only focus on the “short-run impact” of $x_{\sim i}$ on $y_{\sim i}$, rather than the net effect. As Ward and Gleditsch (2008) explain “Since the value of $y_{\sim i}$ will influence the level of” homicide “in other” counties $y_{\sim j}$ and these $y_{\sim j}$, in turn, feedback on to $y_{\sim i}$, we need to take into account the additional effects that the short impact of $x_{\sim i}$ exerts on $y_{\sim i}$ through its impact on the level of” homicide “in other” counties. You can still read the coefficients in the same way but need to keep in mind that they are not measuring the net effect. Part of their effect will be captured by the spatial lag. Yet, you may still want to have a look at whether things change dramatically, particularly in terms of their significance (which is not the case in this example).

In sum, this implies that a change in the i th region’s predictor can affect the j th region’s outcome. We have 2 situations: (a) the direct impact of an observation’s predictor on its own outcome, and (b) the indirect impact of an observation’s neighbor’s predictor on its outcome. This leads to three quantities that we want to know:

- Average Direct Impact, which is similar to a traditional interpretation
- Average Total impact, which would be the total of direct and indirect impacts of a predictor on one’s outcome
- Average Indirect impact, which would be the average impact of one’s neighbors on one’s outcome

These quantities can be found using the `impacts()` function in the `spdep` library. We follow the example that converts the spatial weight matrix into a “sparse” matrix, and power it up using the `trW()` function. This follows the approximation methods described in Lesage and Pace, 2009. Here, we use Monte Carlo simulation to obtain simulated distributions of the various impacts.

```
W <- as(rwm_n, "CsparseMatrix")
## Warning: Function as_dgRMatrix_listw moved to the spatialreg package
```

```

## Warning in as_dgRMatrix_listw(from): install the spatialreg package

trMC <- trW(W, type="MC")

## Warning: Function trW moved to the spatialreg package

## Warning in trW(W, type = "MC"): install the spatialreg package

im<-impacts(fit_2_lag, tr=trMC, R=100)

## Warning: Method impacts.sarlm moved to the spatialreg package

## Warning in impacts.sarlm(fit_2_lag, tr = trMC, R = 100): install the spatialreg
## package

## Warning: Function intImpacts moved to the spatialreg package

## Warning in intImpacts(rho = rho, beta = beta, P = P, n = n, mu = mu, Sigma =
## Sigma, : install the spatialreg package

sums<-summary(im, zstats=T)

## Warning: Method summary.lagImpact moved to the spatialreg package

## Warning in summary.lagImpact(im, zstats = T): install the spatialreg package

#To print the coefficients
data.frame(sums$res)

##      direct    indirect     total
## 1  1.7407989  0.27047183  2.0112707
## 2  1.0062556  0.15634419  1.1625998
## 3 -0.1799533 -0.02795975 -0.2079131
## 4  0.3842918  0.05970828  0.4440001
## 5  0.0882703  0.01371476  0.1019851

#To print the p values
data.frame(sums$pzmat)

```

```

##          Direct    Indirect      Total
## RD60 0.000000e+00 0.002099970 0.000000e+00
## DV60 0.000000e+00 0.000808037 0.000000e+00
## MA60 0.000000e+00 0.001303242 0.000000e+00
## PS60 2.380990e-07 0.007099388 4.044393e-07
## UE60 6.620309e-03 0.026639988 5.996491e-03

```

We see that all the variables have significant direct, indirect and total effects. You may want to have a look at how things differ when you just run a non spatial model.

```

fit_1_OLS <- lm(HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data=ncovr_n_sf)
summary(fit_1_OLS)

##
## Call:
## lm(formula = HR60 ~ RD60 + DV60 + MA60 + PS60 + UE60, data = ncovr_n_sf)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.1581  -1.5986  -0.5770   0.7949  28.4121
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 6.39963   0.66312   9.651  < 2e-16 ***
## RD60        1.85730   0.15692  11.836  < 2e-16 ***
## DV60        1.11883   0.07604  14.713  < 2e-16 ***
## MA60       -0.19537   0.01976  -9.889  < 2e-16 ***
## PS60        0.37748   0.07782   4.851  1.34e-06 ***
## UE60        0.09277   0.03021   3.071  0.00217 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.132 on 1667 degrees of freedom
## Multiple R-squared:  0.1833, Adjusted R-squared:  0.1809
## F-statistic: 74.83 on 5 and 1667 DF,  p-value: < 2.2e-16

```

9.7 Fitting an interpreting a spatial error model

We saw (well you should have seen!) that for the case of homicide in the 80s for Southern states the spatial error model was more appropriate when using the 1 st order contiguity queen criteria. In this case then, we need to run a spatial error model.

Maximum likelihood estimation of the spatial error model is similar to the lag procedure and implemented in the `errorsarlm()` function. Again, the formula, data set and a `listw` spatial weights object must be specified, as illustrated below.

```
#We coerce the sf object into a new sp object
ncovr_s_sp <- as(ncovr_s_sf, "Spatial")
#Then we create a list of neighbours using the Queen criteria
w_s <- poly2nb(ncovr_s_sp, row.names=ncovr_s_sp$FIPSNO)
wm_s <- nb2mat(w_s, style='B')
rwm_s <- mat2listw(wm_s, style='W')

fit_3_err <- errorsarlm(HR80 ~ RD80 + DV80 + MA80 + PS80 + UE80, data=ncovr_s_sf, rwm_s)

## Warning: Function errorsarlm moved to the spatialreg package

## Warning in errorsarlm(HR80 ~ RD80 + DV80 + MA80 + PS80 + UE80, data =
## ncovr_s_sf, : install the spatialreg package

## Warning: Function can.be.simmed moved to the spatialreg package

## Warning in can.be.simmed(listw): install the spatialreg package

## Warning: Function jacobianSetup moved to the spatialreg package

## Warning in jacobianSetup(method, env, con, pre_eig = con$pre_eig, trs = trs, :
## install the spatialreg package

## Warning: Function eigen_setup moved to the spatialreg package

## Warning in eigen_setup(env, which = which): install the spatialreg package

## Warning: Function as_dgRMatrix_listw moved to the spatialreg package

## Warning in as_dgRMatrix_listw(from): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package

## Warning in do_ldet(lambda, env): install the spatialreg package

## Warning: Function do_ldet moved to the spatialreg package
```



```
summary(fit_3_err)

## Warning: Method summary.sarlm moved to the spatialreg package

## Warning in summary.sarlm(fit_3_err): install the spatialreg package

## Warning: Method Wald1.sarlm moved to the spatialreg package

## Warning in Wald1.sarlm(object): install the spatialreg package

## Warning: Method logLik.sarlm moved to the spatialreg package

## Warning in logLik.sarlm(object): install the spatialreg package

## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(object): install the spatialreg package

## Warning: Method LR1.sarlm moved to the spatialreg package

## Warning in LR1.sarlm(object): install the spatialreg package

## Warning: Method logLik.sarlm moved to the spatialreg package

## Warning in logLik.sarlm(object): install the spatialreg package

## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(object): install the spatialreg package

## Warning: Method print.summary.sarlm moved to the spatialreg package

## Warning in print.summary.sarlm(x): install the spatialreg package

##
## Call:errorsarlm(formula = HR80 ~ RD80 + DV80 + MA80 + PS80 + UE80,
##      data = ncovr_s_sf, listw = rwm_s)
##
## Residuals:
```

```
## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(x): install the spatialreg package

##      Min       1Q   Median       3Q      Max
## -17.70984 -3.46135 -0.56817  2.55031 24.84717
##
## Type: error
## Coefficients: (asymptotic standard errors)
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept) 11.395096  1.525679 7.4689 8.082e-14
## RD80        3.286154  0.189085 17.3793 < 2.2e-16
## DV80        0.732640  0.133462  5.4895 4.030e-08
## MA80       -0.177938  0.044032 -4.0411 5.319e-05
## PS80        1.540084  0.214637  7.1753 7.216e-13
## UE80       -0.174869  0.066194 -2.6418  0.008248
##
## Lambda: 0.3026, LR test value: 61.006, p-value: 5.6621e-15
## Asymptotic standard error: 0.037726
##      z-value: 8.0209, p-value: 1.1102e-15
## Wald statistic: 64.335, p-value: 9.992e-16

## Warning: Method logLik.sarlm moved to the spatialreg package

## Warning in logLik.sarlm(x): install the spatialreg package

## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(object): install the spatialreg package

##
## Log likelihood: -4384.823 for error model
## ML residual variance (sigma squared): 28.654, (sigma: 5.353)
## Number of observations: 1412
## Number of parameters estimated: 8

## Warning: Method logLik.sarlm moved to the spatialreg package

## Warning in logLik.sarlm(object): install the spatialreg package

## Warning: Method residuals.sarlm moved to the spatialreg package

## Warning in residuals.sarlm(object): install the spatialreg package
```

```
## AIC: 8785.6, (AIC for lm: 8844.7)
```

The spatial lag model is probably the most common specification and maybe the most generally useful way to think about spatial dependence. But we can also enter the spatial dependence as, we mentioned through the error term in our regression equation. Whereas the spatial lag model sees the spatial dependence as substantively meaningful (in that y , say homicide, in county i is influenced by homicide in its neighbours), the spatial error model simply treats the spatial dependence as a nuisance.

This model focuses on estimating the regression parameters for the explanatory variables of interest and disregards the possibility that the spatial clustering, the spatial autocorrelation, may reflect something meaningful (other than attributional dependence as explained earlier). So instead of assuming that a spatial lag influences the dependent variable we estimate a model that relaxes the standard regression model assumption about the need for the errors to be independent. It's beyond the scope of this introductory course to cover the mathematical details of this procedure, though you can use the suggested reading (particularly the highly accessible Ward and Gleditsch, 2008, book or the more recent Darmofal, 2015) or some of the video lectures in the matter that are available in the GeoDa website.

As before the AIC is better for the spatial model (8786) than for the non spatial model (8845). In this case, you can compare the regression coefficients with those from the OLS model, since we don't have a spatial lag capturing some of their effect. Notice how one of the most notable differences is the fact that unemployment nearly halves its impact in the new model. You will see the table includes a new parameter (λ) but you don't need to worry about this for the purpose of the course. It is something you would understand if you get into the mathematical estimation details.

HOMEWORK 9.3

Estimate an appropriate regression model for the homicide rate for the 1970s for the Northern States. Justify and interpret the model that you have selected.

Chapter 10

Time matters

This week we consider another important factor that is present in our data that we don't always talk about, and that is the importance of *time*. The importance of place in criminology and crime analysis is widely discussed. We know certain areas can be crime hotspots, and we know that whether you come from a well off or deprived area you have different access to resources, and therefore your outcomes in terms of involvement with the criminal justice system also differs. However time is just as important as place. We often hear that crime is "going up" or "going down" over time. It is very important, that as well-rounded criminologists, you are able to talk about these concepts with appropriate knowledge and understanding.

When violence increases between March and August, is that because we are seeing an increase in crime and offending? Or is it possible that the time of year has something to do with this? How much must crime increase and over how long of a time, in order to be able to confidently say that crime is on the increase? These are important, and not always easy questions to answer, and this week we will begin to think about this.

All crimes occur at a specific date and time, however such definite temporal information is only available when victims or witnesses are present, alarms are triggered, etc., at the time of occurrence. This specific temporal data is most often collected in crimes against persons. In these cases, cross-tabulations or histogram of weekday and hour by count will suffice. The great majority of reported events are crimes against property. In these cases, there are seldom victims or witnesses present. These events present the analyst with 'ranged' temporal data, that is, an event reported as occurring over a range of hours or even days. In the case of ranged temporal data, analysis is possible through use of equal chance or probability methods. If an event was reported as having occurred from Monday to Tuesday, in the absence of evidence to the contrary, it is assumed the event had an equal chance or probability of occurring on each of the two days, or .5 (%50). In the same manner, if an event was reported as

having occurred over a 10 hour span there is a 10% chance the event occurred during any one of the hours. This technique requires a reasonable number of events in the data set to be effective. The resulting probabilities are totalled in each category and graphed or cross-tabulated. This produces a comparison of relative frequency, by weekday or hour source.

Temporal crime analysis looks at trends in crime or incidents. A crime or incident trend is a broad direction or pattern that specific types or general crime and/or incidents are following.

Three types of trend can be identified:

- overall trend – highlights if the problem is getting worse, better or staying the same over a period of time
- seasonal, monthly, weekly or daily cycles of offences – identified by comparing previous time periods with the same period being analysed
- random fluctuations – caused by a large number of minor influences, or a one-off event, and can include displacement of crime from neighbouring areas due to partnership activity or crime initiatives.

Decomposing these trends is an important part of what **time series** analysis is all about. Let's see some examples.

10.1 Getting time series data into R and plotting it

We are going to start with fairly simple data. We are just going to look at monthly counts of crime for Greater Manchester obtained from the Police.UK website. We have aggregated 36 months of data into a file for you to use. Grab the data using the following code:

```
data_url <-"https://raw.githubusercontent.com/maczokni/crimemapping_textbook_bookdown/main/gmp_month.csv"
gmp_month <- read.csv(url(data_url))
```

If you view the data you will see it has two columns, the date and the count of crimes for that month. Once you have read the time series data into R, the next step is to store the data in a time series object in R, so that you can use R's many functions for analysing time series data. To store the data in a time series object, we use the `ts()` function in R.

```
library(dplyr)
#First we select the relevant column
gmp_month_c <- select(gmp_month, count)
#Then we create the time series object
gmp_timeseries <- ts(gmp_month_c)
```

You can auto print to see the result:

```
gmp_timeseries
```

Sometimes the time series data set that you have may have been collected at regular intervals that were less than one year, for example, monthly or quarterly. In this case, you can specify the number of times that data was collected per year by using the `frequency` parameter in the `ts()` function. For monthly time series data, you set `frequency=12`, while for quarterly time series data, you set `frequency=4`.

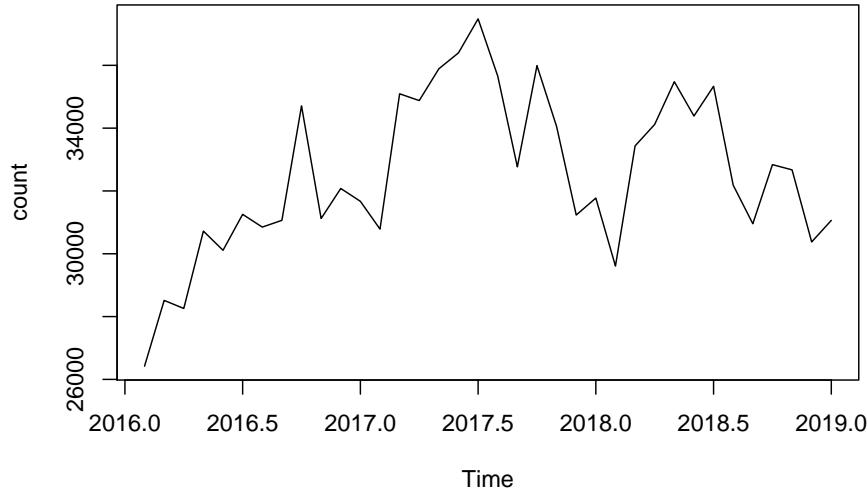
You can also specify the first year that the data was collected, and the first interval in that year by using the `start` parameter in the `ts()` function. So, in our case, we would do as follows:

```
gmp_timeseries <- ts(gmp_month_c, frequency=12, start=c(2016,2))
gmp_timeseries
```

```
##           Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
## 2016      26422 28516 28258 30722 30111 31255 30850 31064 34709 31122 32079
## 2017      31670 30784 35094 34878 35890 36397 37480 35658 32768 35990 34052 31233
## 2018      31774 29609 33434 34120 35478 34386 35333 32184 30956 32838 32672 30377
## 2019      31063
```

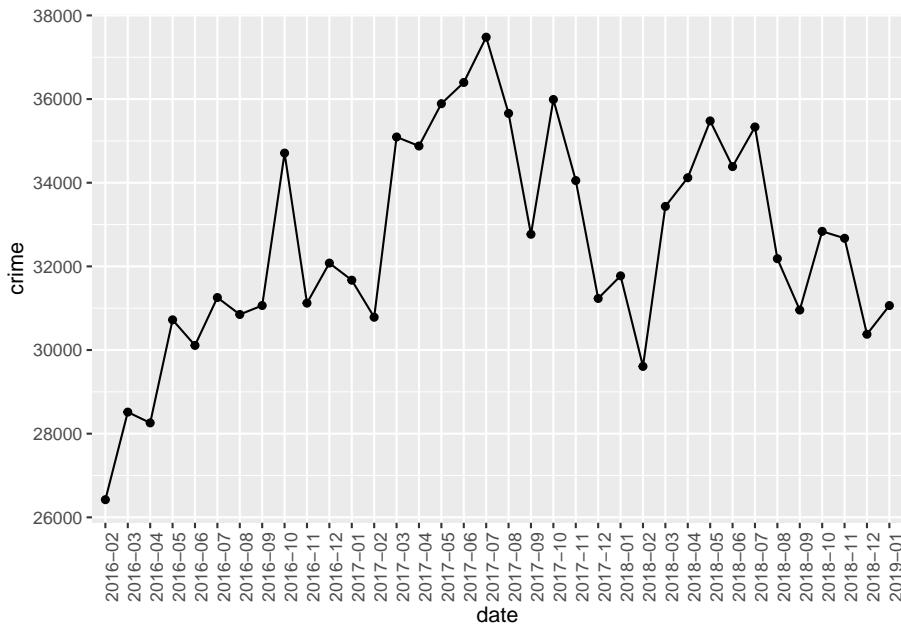
Once you have read a time series into R, the next step is usually to make a plot of the time series data, which you can do with the `plot.ts()` function in R.

```
plot.ts(gmp_timeseries)
```



We can of course also use `ggplot2` to plot a time series like the one we just did but here we would need a variable encoding the date (and preferably a full date, not just month and year as here). If you look at “`gmp_month`” you will see the name of the variables are not optimal, so we will rename them first.

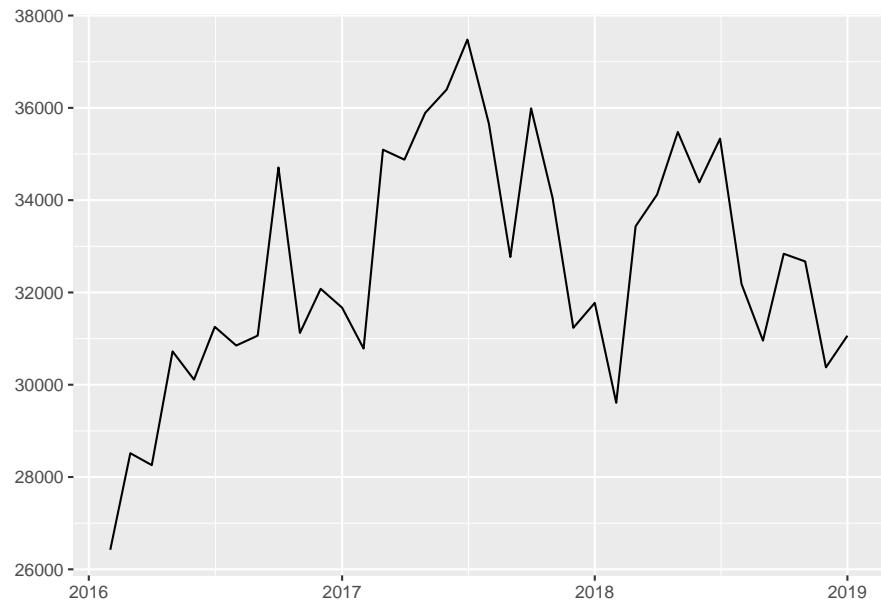
```
library(ggplot2)
gmp_month <- rename(gmp_month, date = as.factor.Month., crime = count)
ggplot(data=gmp_month, aes(x=date, y=crime, group=1)) +
  geom_line() +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



If we had a date vector rather than a factor encoding our date we would need to adapt our code. See here for further details.

Alternatively we can use `ggfortify`, an extension to `ggplot2` that would allow us to use `ts` objects as inputs. For this we use the `autoplot` function.

```
library(ggfortify)
autoplot(gmp_timeseries)
```



10.2 Lubridate: your guardian angel when it comes to working with temporal data

As you saw, the data from Police.Uk is aggregated by months. We do not know when the offences happened, only the month, but nothing more granular than that. American police data on the other hand is much more granular.

Cities release their own data. Here, we will be looking at crimes from New York City on aggravated assault. The data from an open source initiative maintained by Matt Ashby here. I have filter and selected the data so that we are only working with aggravated assaults from New York for a period of five years.

```
library(readr)
agassault_ny<-read_csv("https://raw.githubusercontent.com/maczokni/crimemapping_textbook/master/data/agassault_ny.csv")
```

```
## Parsed with column specification:
## cols(
##   uid = col_double(),
##   date_single = col_datetime(format = ""),
##   longitude = col_double(),
##   latitude = col_double(),
##   location_type = col_character(),
##   location_category = col_character(),
```

```
##   census_block = col_double(),
##   date_start = col_datetime(format = ""),
##   date_end = col_datetime(format = "")
## )
```

When you read the data into R, you will see that there is a column for date called `date_single`. The date is in the format dd/mm/yyyy and h/m/s. So the first date on there you can see is `2014-01-01 00:03:00`. What kind of variable is this?

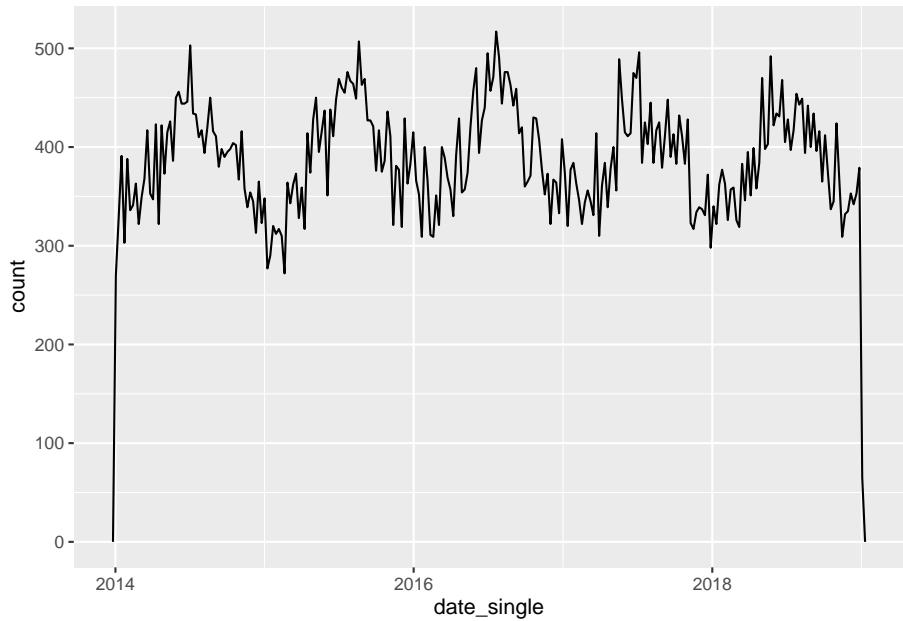
```
class(agassault_ny$date_single)

## [1] "POSIXct" "POSIXt"
```

Our date and time variables are of class `POSIXct` and `POSIXt`. These are the two basic classes of date/times. Class “`POSIXct`” represents the (signed) number of seconds since the beginning of 1970 as a numeric vector. Class “`POSIXt`” is a named list of vectors representing seconds (0–61), minutes (0–59), hours (0–23), day of the month (1–31), months after the first of the year (0–11), years since 1900, day of the week, starting on Sunday (0–6), and a flag for whether it is daylight savings time or not (positive if in force, zero if not, negative if unknown).

Let's plot this data:

```
agassault_ny %>%
  ggplot(aes(date_single)) +
  geom_freqpoly(binwidth = 7*24*60*60) # 7 days in seconds
```



Notice what `geom_freqpoly` is doing. We have a data frame with rows for each case. The data is not aggregated in any form. But this function counts on the fly the number of cases (of rows) for each of the bins as we define them. It is, thus, a convenient function that saves us from having to first do that aggregation ourselves when we want to plot it. This may be helpful when you get your data for the assignment.

An alternative approach to plotting individual components is to round the date to a nearby unit of time, with `floor_date()`, `round_date()`, and `ceiling_date()`. Each function takes a vector of dates to adjust and then the name of the unit round down (floor), round up (ceiling), or round to.

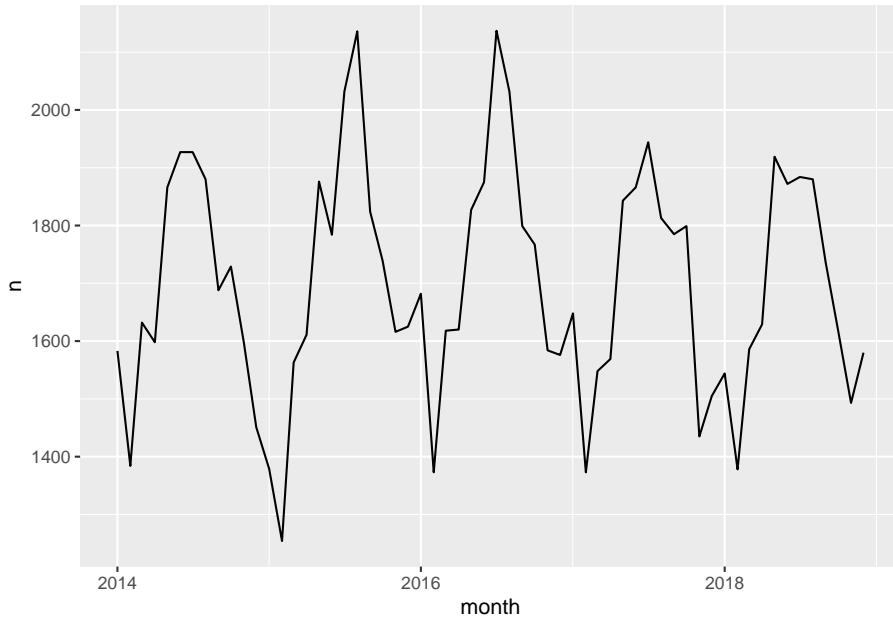
```
library(lubridate)

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:dplyr':
##     intersect, setdiff, union

## The following objects are masked from 'package:base':
##     date, intersect, setdiff, union
```

```
agassault_ny %>%
  count(month = floor_date(date_single, "month")) %>%
  ggplot(aes(month, n)) +
  geom_line()
```



What if I asked you the question: which year had the most aggravated assaults? Or what if I want to know if aggravated assaults happen more in the weekday, when people are at work, or in the weekends, maybe when people are away for a holiday? You have the date, so you should be able to answer these questions, right?

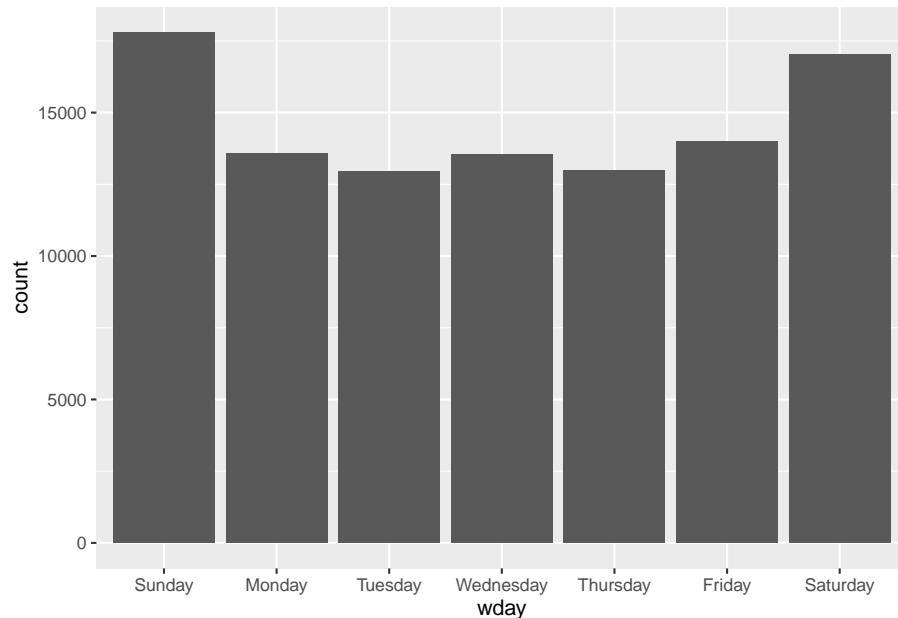
Well you need to be able to have the right variables to answer these questions. To know what year saw the most aggravated assaults, you need to have a variable for year. To know what day of the week has the most aggravated assaults, you need to have a variable for day of the week. So how can we extract these variables from your date column? Well luckily the `lubridate` package can help us do this.

```
agassault_ny$year <- year(agassault_ny$date_single)
agassault_ny$month <- month(agassault_ny$date_single, label = TRUE, abbr=FALSE)
agassault_ny$day <- day(agassault_ny$date_single)
agassault_ny$wday <- wday(agassault_ny$date_single, label = TRUE, abbr=FALSE)
```

And as you can see now you have a set of additional variables that have extracted information from your original time of occurrence variable.

We can now look at the distribution of events per day of the week.

```
agassault_ny %>%
  mutate(wday = wday(date_single, label=TRUE, abbr=FALSE)) %>%
  ggplot(aes(x = wday)) +
  geom_bar()
```



The `lubridate` package is incredibly helpful for anything date related. We don't have time to go over it on details but you can find more information on the official page or in this tutorial. The relevant chapter in R for Data Science is also very helpful.

10.3 Calendar heatmaps

We can also parse the temporal information that is available in our date variable. For the next chart we introduce we only need date, month, and year. We don't need the time.

```
agassault_ny$date <- format(as.POSIXct(agassault_ny$date_single, format="%Y-%m-%d %H:%M"))
head(agassault_ny$date)

## [1] "2014-01-01" "2014-01-01" "2014-01-01" "2014-01-01" "2014-01-01" "2014-01-01"
## [6] "2014-01-01"
```

```
class(agassault_ny$date)
```

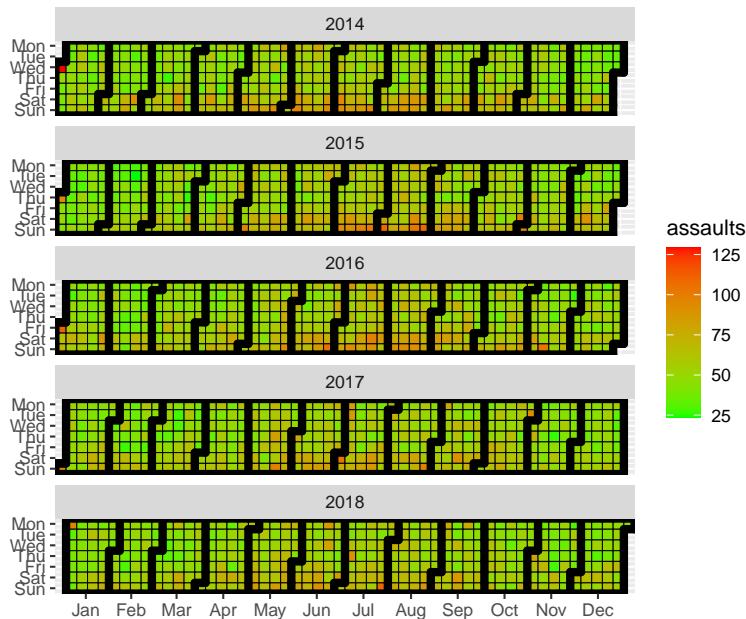
```
## [1] "character"
```

On this character vector can now use `dplyr` to counts the events per day.

```
agassault_ny_d <- agassault_ny %>%
  group_by(date) %>%
  summarise(assaults = n())
```

We will now use a `ggplot2` extension that allow us to produce calendar heat visualisations.

```
agassault_ny_d$date <- ymd(agassault_ny_d$date) # we parse the date again from the character vector
library(ggTimeSeries)
p1 <- ggplot_calendar_heatmap(
  agassault_ny_d,
  'date',
  'assaults')
#Now we customise the plot a bit more
p1 +
  xlab(NULL) +
  ylab(NULL) +
  scale_fill_continuous(low = 'green', high = 'red') +
  facet_wrap(~Year, ncol = 1)
```

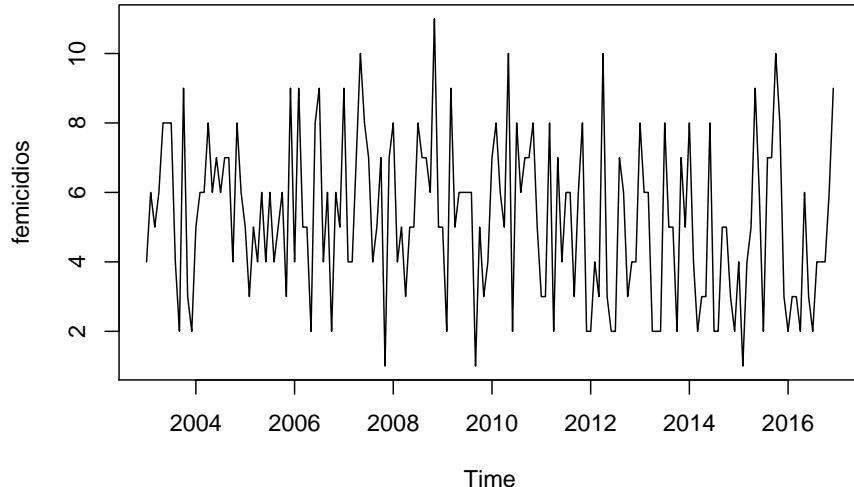


There are many things that peak on certain days of the week. If you're interested in some more examples, read this article in the Guardian about the most dangerous days of the week.

10.4 Decomposing time series

Decomposing a time series means separating it into its constituent components, which are usually a trend component and an irregular component, and if it is a seasonal time series, a seasonal component. Let's get some fresh data. These are intimate partner femicides from Spain;

```
fem <- read.csv("https://github.com/maczokni/crimemapping_textbook_bookdown/raw/master/femicidios.csv")
fem_timeseries <- ts(fem, frequency=12, start=c(2003,1))
plot.ts(fem_timeseries)
```



As you can see it is very noisy. Fortunately, the annual count for intimate partner femicides is low in Spain. There seems to be some seasonality too.

A seasonal time series consists of a trend component, a seasonal component and an irregular component. Decomposing the time series means separating the time series into these three components: that is, estimating these three components.

To estimate the trend component and seasonal component of a seasonal time series that can be described using an additive model, we can use the `decompose()`

function in R. This function estimates the trend, seasonal, and irregular components of a time series that can be described using an additive model.

The function `decompose()` returns a list object as its result, where the estimates of the seasonal component, trend component and irregular component are stored in named elements of that list objects, called “seasonal”, “trend”, and “random” respectively.

To estimate the trend, seasonal and irregular components of this time series, we type:

```
fem_timeseriescomponents <- decompose(fem_timeseries)
```

The estimated values of the seasonal, trend and irregular components are now stored in variables `fem_timeseriescomponents$seasonal`, `fem_timeseriescomponents$trend` and `fem_timeseriescomponents$random`. For example, we can print out the estimated values of the seasonal component by typing:

```
fem_timeseriescomponents$seasonal
```

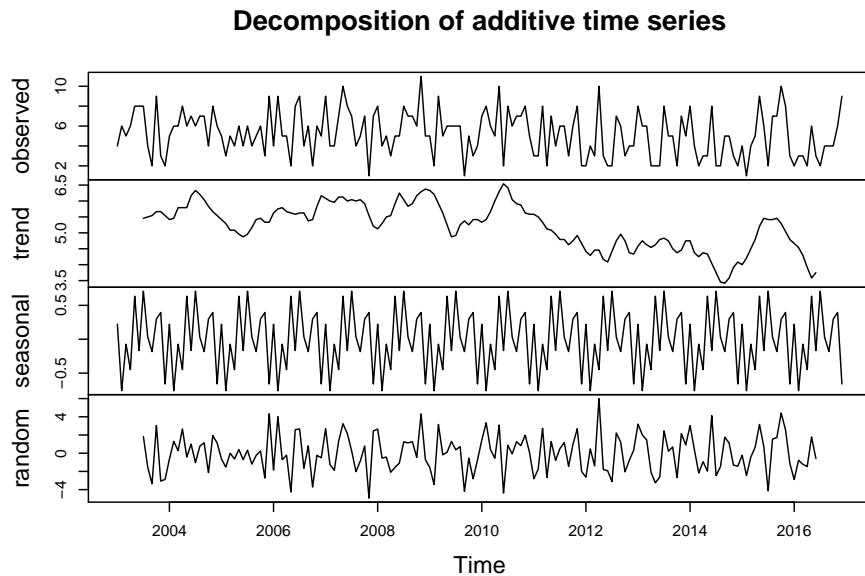
	Jan	Feb	Mar	Apr	May	Jun
## 2003	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2004	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2005	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2006	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2007	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2008	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2009	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2010	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2011	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2012	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2013	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2014	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2015	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
## 2016	0.22142094	-0.75934829	-0.07345085	-0.44845085	0.63488248	-0.16639957
	Jul	Aug	Sep	Oct	Nov	Dec
## 2003	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932
## 2004	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932
## 2005	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932
## 2006	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932
## 2007	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932
## 2008	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932
## 2009	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932
## 2010	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932
## 2011	0.70860043	0.03231838	-0.18242521	0.29834402	0.39449786	-0.65998932

```
## 2012 0.70860043 0.03231838 -0.18242521 0.29834402 0.39449786 -0.65998932
## 2013 0.70860043 0.03231838 -0.18242521 0.29834402 0.39449786 -0.65998932
## 2014 0.70860043 0.03231838 -0.18242521 0.29834402 0.39449786 -0.65998932
## 2015 0.70860043 0.03231838 -0.18242521 0.29834402 0.39449786 -0.65998932
## 2016 0.70860043 0.03231838 -0.18242521 0.29834402 0.39449786 -0.65998932
```

The estimated seasonal factors are given for the months January–December, and are the same for each year. The largest seasonal factor is for July (about 0.70), and the lowest is for February (about -0.76), indicating that there seems to be a peak in femicides in July and a trough in February each year.

We can plot the estimated trend, seasonal, and irregular components of the time series by using the `plot()` function, for example:

```
plot(fem_timeseriescomponents)
```



Once we remove the noise and the seasonal components, it becomes easier to see the estimated trend. Notice that while random and seasonal components still look messy, their scales are different and centred around zero.

Homework 10.1 Try creating a `ts` object using the `aggassault` data from New York and decompose the time series

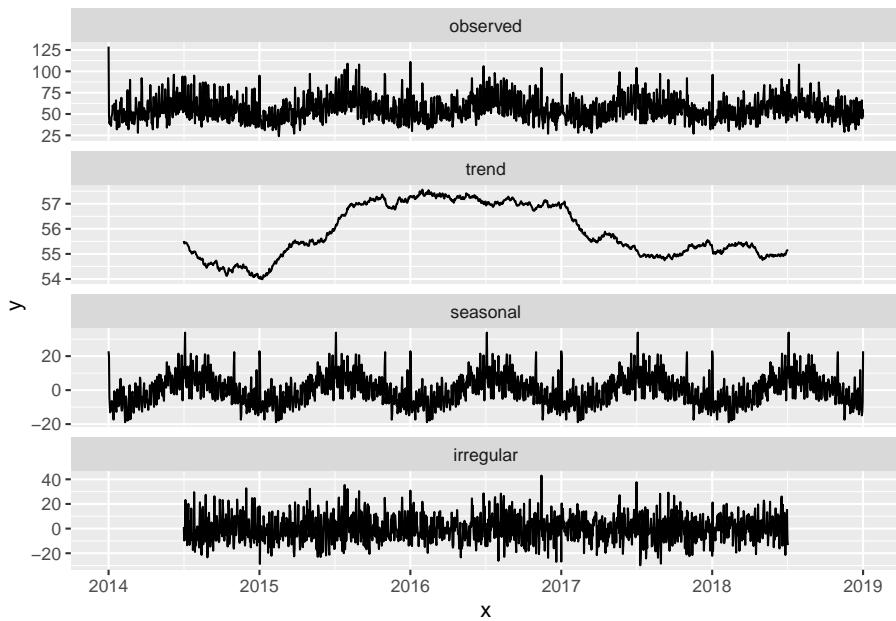
We can also use `ggplot2` for these purposes. In particular we can use the `ggseas` extension which allows for seasonal decomposition within `ggplot` — see here for details. First I use the `tsdf` function that turns the `ts` object you created for the

homework into a dataframe (you may have provided this object with a different name) and then plot the series.

```
library(ggseas)
ny_df <- tsdf(ny_timeseries)
ggsdc(ny_df, aes(x = x, y = y), method = "decompose") +
  geom_line()

## Calculating frequency of 365 from the data.

## Warning: Removed 182 row(s) containing missing values (geom_path).
```



10.5 Static maps with ggplot2

So far we have seen how we can produce maps with `tmap` for thematic maps and `leaflet` for interactive maps. But we can also use static maps with `ggplot2`. For the next set of exercises we are going to look at temporal variations on burglary across Greater Manchester. We are going to focus on wards as the unit of analysis. You will likely work with this dataset for crime for your assignment and you may use wards or smaller geographies. So we will focus on this data for this illustration of basic plotting and spatial operations (in the next section).

To load the ward shapefiles for Manchester into a `sf` object we use code we had already used in a previous session.

```
library(sf)

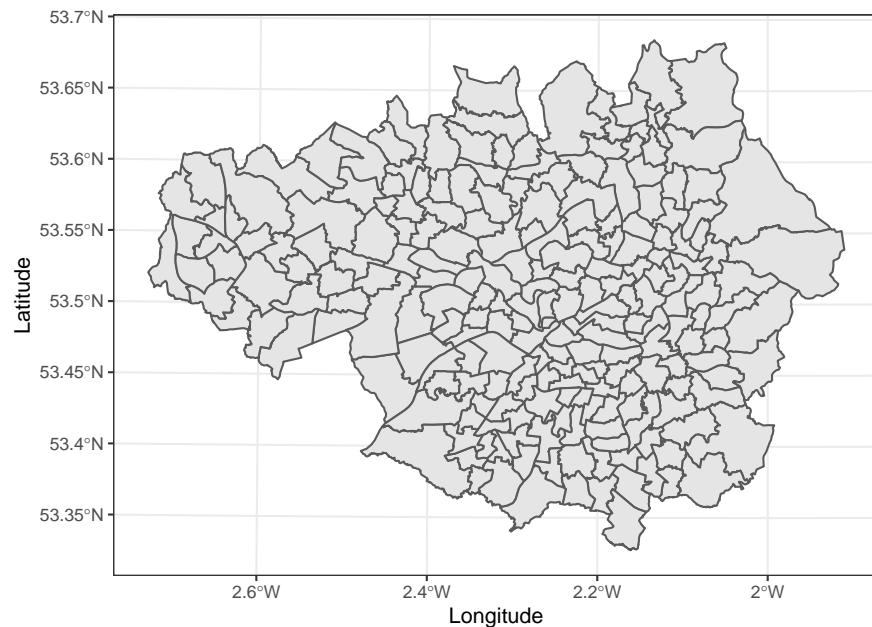
## Linking to GEOS 3.8.1, GDAL 2.4.4, PROJ 4.9.1

manchester_ward <- st_read("https://raw.githubusercontent.com/RUMgroup/Spatial-data-in-"

## Reading layer `wards' from data source `https://raw.githubusercontent.com/RUMgroup/Spatial-data-in-
## Simple feature collection with 215 features and 12 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:            xmin: 351664 ymin: 381168.6 xmax: 406087.5 ymax: 421039.8
## CRS:             27700
```

With this data in our environment we can plot a map using `ggplot2`. We use the `geom_sf` function to tell `ggplot2` we are dealing with a `sf` object.

```
library(ggplot2)
ggplot() + geom_sf(data = manchester_ward, aes()) +
  labs(x = "Longitude", y = "Latitude") +
  theme_bw()
```



We can also plot objects that are not `sf` but have geographic coordinates.

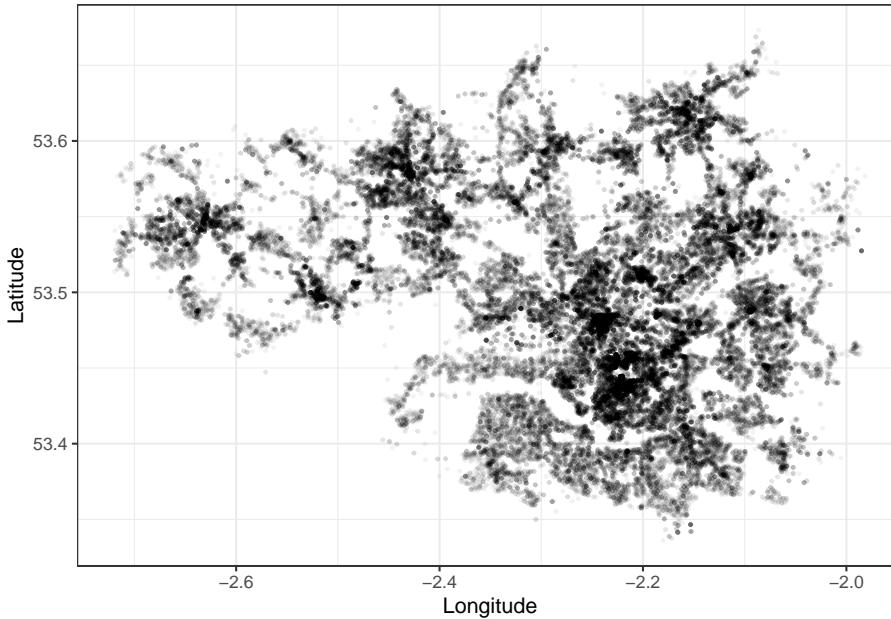
Let's load the data with the burglaries for Greater Manchester.

```
gmp_bur <- read_csv("https://raw.githubusercontent.com/maczokni/crimemapping_textbook_bookdown/main/data/gmp_burglaries.csv")
```

```
## Parsed with column specification:
## cols(
##   Month = col_character(),
##   Crime.type = col_character(),
##   Longitude = col_double(),
##   Latitude = col_double()
## )
```

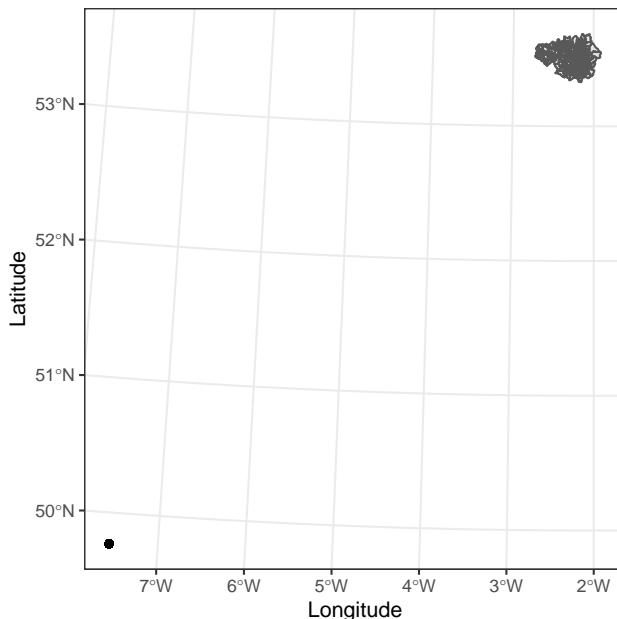
You can see there is a column with the longitude and another with latitude. For this we can use the standard `geom_point` function specifying X and Y as our longitude and latitude. Given the number of points in the data I am also modifying the transparency and size of the points to deal with overplotting.

```
ggplot() +
  geom_point(data = gmp_bur, aes(x = Longitude, y = Latitude),
             alpha = 0.05, size = 0.5) +
  labs(x = "Longitude", y = "Latitude") +
  theme_bw()
```



You could try to have a map with two layers, the points and the wards, so combining what we have done so far.

```
ggplot() + geom_sf(data = manchester_ward, aes()) +
  geom_point(data = gmp_bur, aes(x = Longitude, y = Latitude)) +
  labs(x = "Longitude", y = "Latitude") +
  theme_bw()
```



Ooops. What happened? Can you think of what is going on here? This is something we have already covered in previous lab sessions.

In week 4 we explained how it is important to check that spatial operations that use two spatial objects rely on both objects having the same coordinate reference system, otherwise funny things may happen. We can check the coordinate reference system with the `st_crs` function from `sf`.

```
st_crs(manchester_ward)
```

```
## Coordinate Reference System:
##   User input: 27700
##   wkt:
##   PROJCS["OSGB 1936 / British National Grid",
##          GEOGCS["OSGB 1936",
##                 DATUM["OSGB_1936",
##                       SPHEROID["Airy 1830",6377563.396,299.3249646,
```

```

##           AUTHORITY["EPSG","7001"]],
##           TOWGS84[446.448,-125.157,542.06,0.15,0.247,0.842,-20.489],
##           AUTHORITY["EPSG","6277"]],
##           PRIMEM["Greenwich",0,
##                   AUTHORITY["EPSG","8901"]],
##           UNIT["degree",0.0174532925199433,
##                 AUTHORITY["EPSG","9122"]],
##                 AUTHORITY["EPSG","4277"]],
##           PROJECTION["Transverse_Mercator"],
##           PARAMETER["latitude_of_origin",49],
##           PARAMETER["central_meridian",-2],
##           PARAMETER["scale_factor",0.9996012717],
##           PARAMETER["false_easting",400000],
##           PARAMETER["false_northing",-100000],
##           UNIT["metre",1,
##                 AUTHORITY["EPSG","9001"]],
##           AXIS["Easting",EAST],
##           AXIS["Northing",NORTH],
##           AUTHORITY["EPSG","27700"]]

```

What is EPSG: 27700? You can check here. This is the British National Grid. How about the police data? Well, we don't have a `sf` object, we have a dataframe for the burglaries. But if we look at the documentation for this data from the Police.UK website we can learn that they use WGS1984 instead. What do we do? First we need to create a `sf` object for the police data specifying the coordinate system (we covered how to do this in week 3).

```

burglary_sf <- st_as_sf(x = gmp_bur,
                         coords = c("Longitude", "Latitude"),
                         crs = "+proj=longlat +datum=WGS84")

```

If we now check:

```
st_crs(burglary_sf)
```

```

## Coordinate Reference System:
##   User input: +proj=longlat +datum=WGS84
##   wkt:
##   GEOGCS["WGS 84",
##         DATUM["WGS_1984",
##               SPHEROID["WGS 84",6378137,298.257223563,
##                     AUTHORITY["EPSG","7030"]],
##               AUTHORITY["EPSG","6326"]],
##         PRIMEM["Greenwich",0,
##                AUTHORITY["EPSG","8901"]]]

```

```

##           AUTHORITY["EPSG","8901"]],
##           UNIT["degree",0.0174532925199433,
##                 AUTHORITY["EPSG","9122"]],
##           AUTHORITY["EPSG","4326"]]

st_crs(manchester_ward) == st_crs(burglary_sf)

```

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

We can see the code 4326, which is basically the code for WGS84. And that the coordinate system across our two `sf` objects is different. We also covered this in week 4 but take this as a helpful refresher for the kind of things it is likely you need to do for your assignment. Essentially, we need to use the `st_transform` function. We need to make sure we translate one of these layers into the other coordinate reference system.

```

wards_WGS84 <- st_transform(manchester_ward, 4326)
st_crs(wards_WGS84) == st_crs(burglary_sf)

```

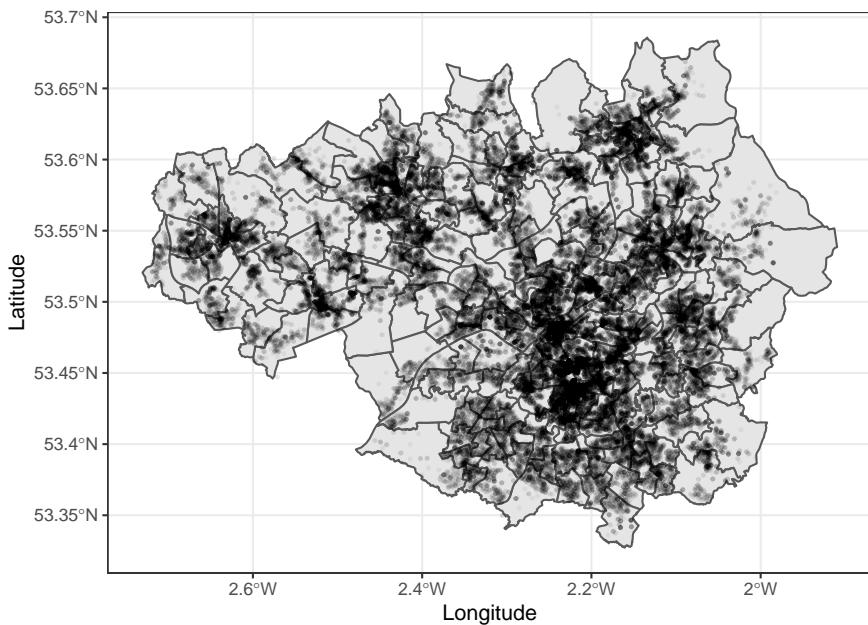
```
## [1] TRUE
```

Try now plotting the map.

```

ggplot() + geom_sf(data = wards_WGS84, aes()) +
  geom_point(data = gmp_bur, aes(x = Longitude, y = Latitude),
             alpha = 0.05, size = 0.5) +
  labs(x = "Longitude", y = "Latitude")+
  theme_bw()

```



10.6 Small multiples to show temporal variation

We have covered small multiples before. You can indeed use facets to show how things change (or don't change) across time. Let's just focus on 2018 to simplify things a bit.

```
bur_sf_2018 <- filter(burglary_sf, stringr::str_detect(Month, '2018')) # detect if there is 2018
table(bur_sf_2018$Month)

##
## 2018-01 2018-02 2018-03 2018-04 2018-05 2018-06 2018-07 2018-08 2018-09 2018-10
##     2649     2316     2566     2463     2534     2410     2458     2476     2450     2582
## 2018-11 2018-12
##     2701     2593
```

Let's simplify further by focusing just in Manchester City.

```
mc_wards_WGS84 <- filter(wards_WGS84, lad16nm == "Manchester")
```

Ok, let's do a point in polygon operation, count the number of burglaries per ward, so that we can display choropleth maps. Again, this is something we cover in week 4.

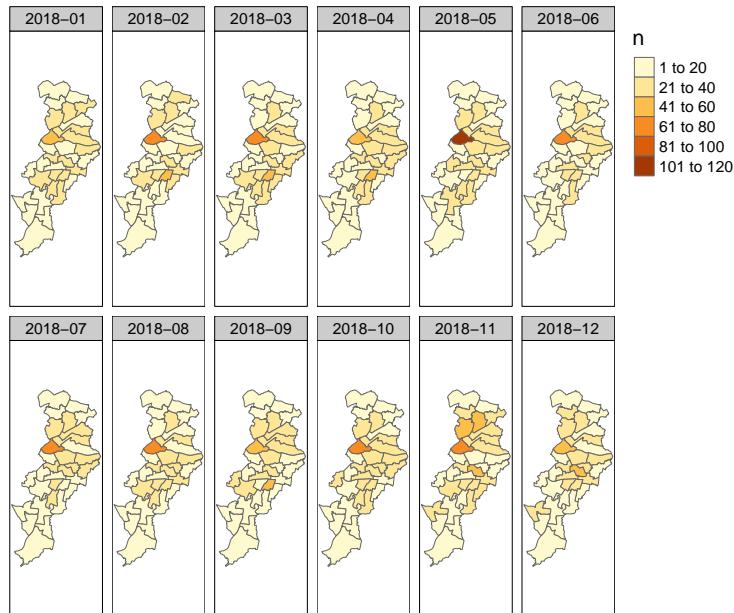
```
bur_per_ward <- bur_sf_2018 %>%
  st_join(mcwards_WGS864, ., left = FALSE) %>%
  count(wd16nm)
```

The code above only counts for the whole year. But if we want to have a map per month we don't need the total counts for the whole year but rather the monthly counts.

```
bur_per_ward <- bur_sf_2018 %>%
  st_join(mcwards_WGS864, ., left = FALSE) %>%
  count(wd16nm, Month) # by default, this saves the count to a column named 'n'
```

We can now try to produce the small multiples.

```
library(tmap)
tm_shape(bur_per_ward) +
  tm_fill("n") +
  tm_borders() +
  tm_facets("Month", free.coords=FALSE)
```



10.7 Spaghetti plots

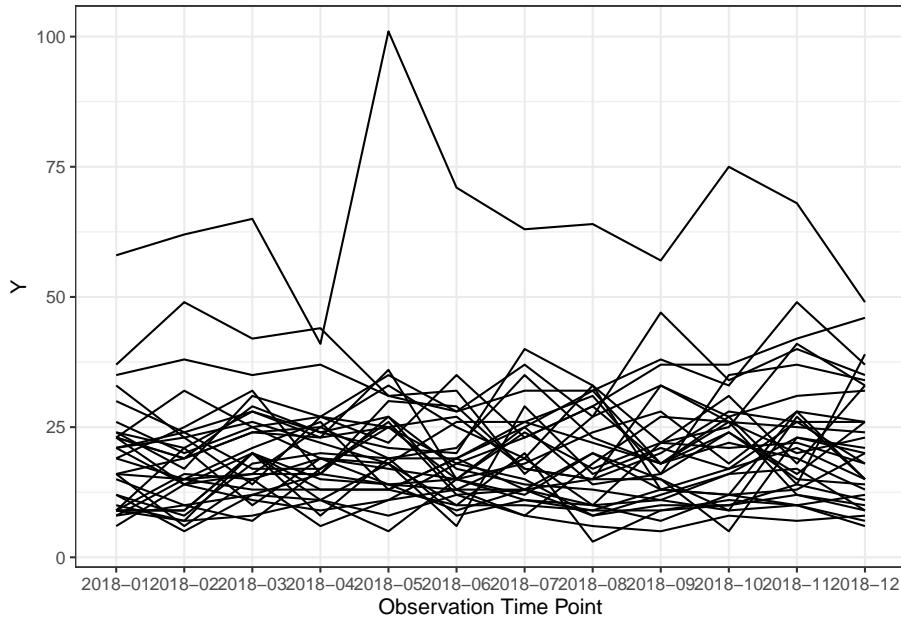
In longitudinal studies and in studies looking at areas over time, sometimes researchers use spaghetti plots. On their own they are not great, but they can be used when one wants to put a trajectory within a broader context or when comparing different trajectories. You can read more about them here. Let's first get rid of the "spatial component" for the data.

```
spa_df <- bur_per_ward
st_geometry(spa_df) <- NULL
class(spa_df)

## [1] "tbl_df"     "tbl"        "data.frame"
```

To produce a simple spaghetti plot we can use ggplot2.

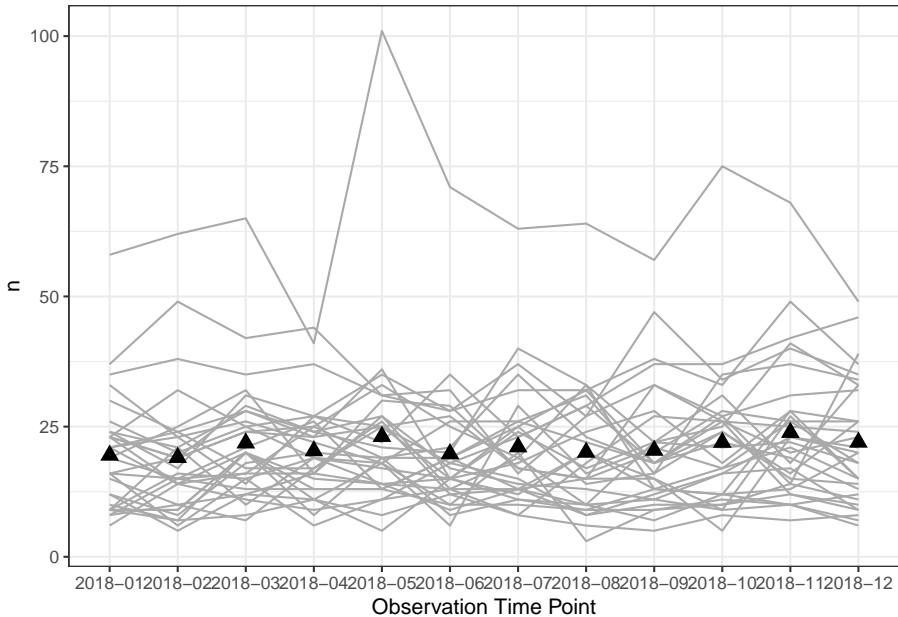
```
ggplot(spa_df, aes(x=Month, y=n, group = wd16nm)) +
  geom_line() + guides(colour=FALSE) + xlab("Observation Time Point") +
  ylab("Y") +
  theme_bw()
```



As you can see we are grouping by ward. So in this case each line is the trend in crime for a single ward. We can use some of the power of ggplot2 for example to extract the mean of Y and plot it along the trajectories. And we can also customise the graphic a bit.

```
ggplot(spa_df, aes(x=Month, y=n, group=wd16nm)) +
  geom_line(color="darkgrey") + guides(colour=FALSE) + xlab("Observation Time Point") +
  stat_summary(aes(group = 1), geom = "point", fun.y = mean, shape = 17, size = 3) +
  theme_bw()
```

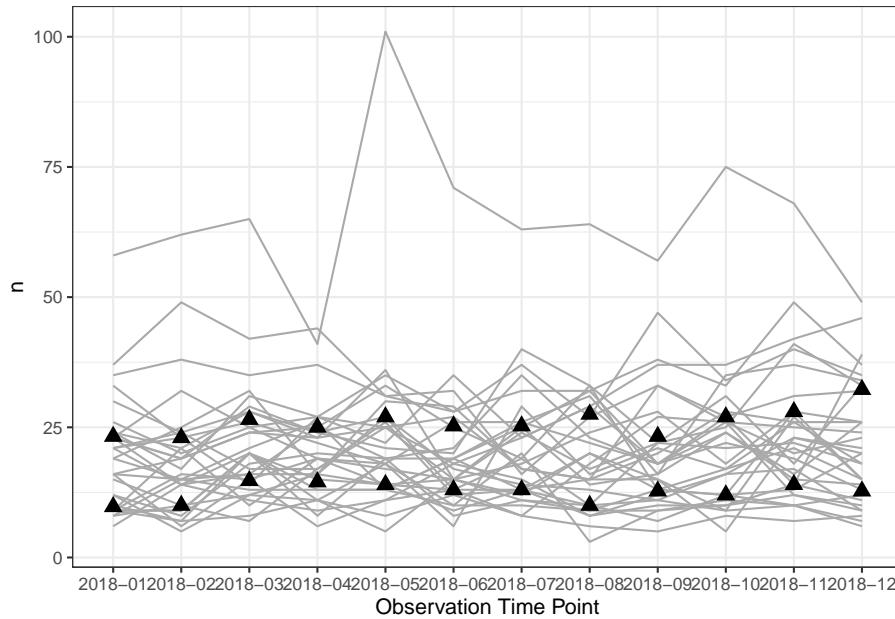
Warning: `fun.y` is deprecated. Use `fun` instead.



The `stat_summary` function is very helpful and we can use it to obtain other statistical summaries. See, for example, how we would plot the first and third quartile.

```
ggplot(spa_df, aes(x=Month, y=n, group=wd16nm)) +
  geom_line(color="darkgrey") + guides(colour=FALSE) + xlab("Observation Time Point") +
  stat_summary(aes(group = 1), geom = "point", fun.y = quantile,
  fun.args=(list(probs = c(0.25, 0.75))), shape = 17, size = 3) +
  theme_bw()
```

Warning: `fun.y` is deprecated. Use `fun` instead.



We can also obtain a smoothed representation of the average trend:

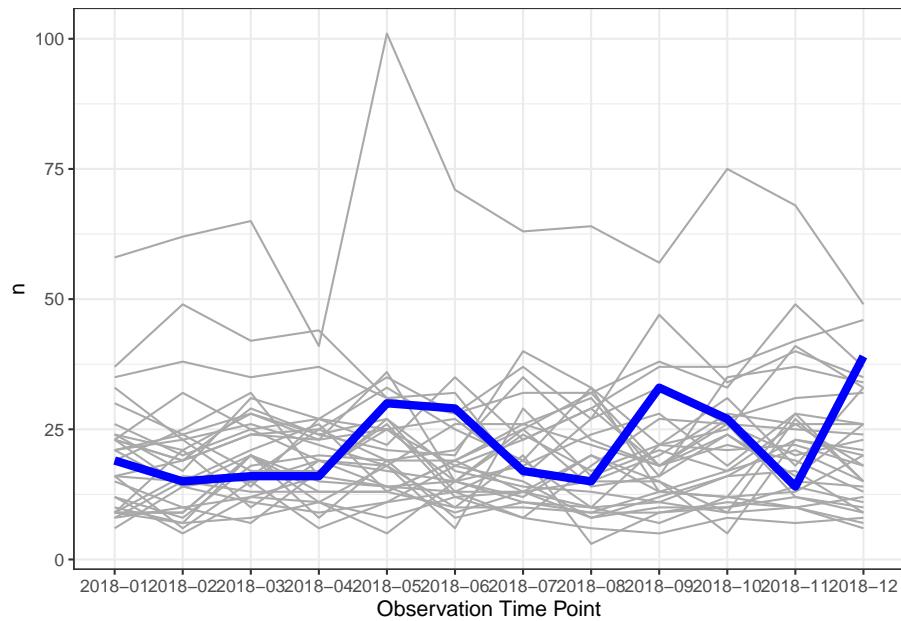
But typically you may want to use this kind of plots in different ways (see the article we linked to above). Here we are going to produce a plot highlighting the trajectory in Fallowfield. Before I do that I will create a new variable identifying Fallowfield.

```
spa_df$fallowfield <- "No"
spa_df$fallowfield[spa_df$wd16nm=="Fallowfield"] <- "Yes"
table(spa_df$fallowfield)
```

```
##
##  No Yes
## 372  12
```

Now I can use this variable to plot the distinct trajectory of Fallowfield.

```
ggplot() +
  geom_line(data= spa_df, aes(x=Month, y=n, group=wd16nm), color="darkgrey") +
  geom_line(data= subset(spa_df, fallowfield=="Yes"), aes(x=Month, y=n, group=1),
            colour="blue", size =2) +
  guides(colour=FALSE) + xlab("Observation Time Point") +
  theme_bw()
```



Cool! Well, you haven't seen it all yet.

10.8 Animations!!!

You want a data viz person get excited? Mention animations! Now they are brought to `ggplot2` thanks to the `ggridge` extension.

The next one is a non-spatial plot, in that we do not show a map. However we will show the varying rate of burglaries per month within each ward. It is interesting to see this as an animation over time, rather than in the small multiples format, because it can better drive home the message of change (and also it looks very cool!).

So first thing we do is to load the `ggridge` package:

```
library(ggridge)
```

```
## No renderer backend detected. ggridge will default to writing frames to separate files
## Consider installing:
## - the `gifski` package for gif output
## - the `av` package for video output
## and restarting the R session
```

Then, we need to make sure that our temporal variable is a date object. We can use the `ymd()` function, from the fantastic `lubridate` package (really I cannot

praise this package enough, it makes handling dates so easy...!) to make sure that our Month variable is a date object.

But we have no day you say! Only month! How can we use `ymd()` which clearly requires year month ad *day*? Well, one approach is to make this up, and just say that everything in our data happened on the 1st of the month. We can use the `paste0()` function to do this:

```
bur_per_ward$date_month <- ymd(paste0(bur_per_ward$Month, "-01"))
```

Now, we can create a simple static plot, the way we already know how. Let's plot the number of burglaries per ward, and save this in an object called `anim`:

```
anim <- ggplot(bur_per_ward, aes(x = wd16nm, y = n, fill=ifelse(wd16nm=="Fallowfield", 'Fallowfield',
  geom_bar(stat = 'identity') +
  scale_y_continuous(labels = scales::comma) +
  coord_flip() +
  scale_fill_manual(values = c('Fallowfield' = 'blue'), na.value='gray50', guide=F) + # we're high
  labs(title="Burglary counts",
       x = "",
       y = "")
```

Now, finally, we can animate this graph. Take the object of the static graph (`anim`) and add a form of transition, which will be used to animate the graph. In this case, we can use `transition_states()`. This transition splits your data into multiple states based on the levels in a given column, much like `ggplot2::facet_wrap()` splits up the data in multiple panels. It then shifts between the defined states and pauses at each state. Layers with data without the specified column will be kept constant during the animation (again, mimicking `facet_wrap`). States are the unquoted name of the column holding the state levels in the data. You can then use the `closest_state` to dynamically label the graph:

```
anim + transition_states(date_month, transition_length = 1, state_length = 1) +
  labs(title = "Month: {closest_state}")

## Warning: No renderer available. Please install the gifski, av, or magick package
## to create animated output
```

How cool is that!?

We can do this for maps as well. However, to apply `gganimate` to `sf` objects, you need to have a package called `transformr` installed. Install this package, and then we can apply `gganimate` to a map of burglaries.

Again make a static map:

```
anim_map <- ggplot(bur_per_ward, aes(fill = n, color = n)) +  
  geom_sf() +  
  geom_sf(data=subset(bur_per_ward, wd16nm=='Fallowfield'), color='black', fill=NA) #
```

And use `transition_states()` to animate by month.

```
anim_map + transition_states(date_month, transition_length = 1, state_length = 1) +  
  labs(title = "Month: {closest_state}")  
  
## Warning: No renderer available. Please install the gifski, av, or magick package  
## to create animated output
```

Ta-daaa!

What a way to end this course!