Human Geography through Merseyside - Quantitative Block: Seeing Liverpool through numbers

Zi Ye and Ron Mahabir

2026-01-19

Table of contents

Welcome

This is the website for "Human Geography through Merseyside - Quantitative Block: Seeing Liverpool through numbers" (module **ENVS162**) at the University of Liverpool. This block of the module is designed and delivered by Dr. Zi Ye and Dr. Ron Mahabir from the Geographic Data Science Lab at the University of Liverpool. The module seeks to provide hands-on experience and training in introductory statistics for human geographers.

Contact

Zi Ye - zi.ye [at] liverpool.ac.uk Lecturer in Geographic Information Science Office 107, Roxby Building, University of Liverpool - 74 Bedford St S, Liverpool, L69 7ZT, United Kingdom.

Ron Mahabir - gfilo [at] liverpool.ac.uk Lecturer in Geographic Data Science Office 4xx, Roxby Building, University of Liverpool - 74 Bedford St S, Liverpool, L69 7ZT, United Kingdom.

Overview

Aim and Learning Objectives

This sub-module aims to provide training and skills on a set of basic quantitative skills for data collection, analysis, and interpretation and to enable you to link conceptual ideas with real world examples. **This block serves as the foundation for Year 2 BA field class and, optionally, for Year 3 dissertation.**

Background

Data and research are key pillars of the global economy and society today. We need rigorous approaches to collecting and analysing both the statistics that can tell us 'how much' and if there are observable relationships between phenomena; and the information gives us a nuanced understanding of cultural contexts and human dynamics. Quantitative skills enable us to explore and measure socio-economic activities and processes at large scales, while qualitative skills enable understanding of social, cultural, and political contexts and diverse lived experiences. Rather than being in opposition, qualitative and quantitative research can complement one another in the investigation of today's pressing research questions.

To these ends, this block will help you develop your quantitative skills, as critical tools. This course will help you understand what quantitative statistical researchers use and develop a set of research techniques that can be used in your field classes and dissertations.

Learning objectives:

- Understand how to explore a dataset, containing a number of observations described by a set of variables.
- Demonstrate an understanding in the application and interpretation of commonly used quantitative research methods.
- Ability to work with quantitative data to understand real-world social phenomenian and patterns.

Module Structure

Staff: Dr Zi Ye and Dr Ron Mahabir

Where and When

Week 1 - 5 Lecture: Tuesday (12am – 1pm) @ Mathematical Sciences, Proudman Lecture Theatre

Week 1 - 6 Practical PC session: Friday (9 – 11 am) @ Central Teaching Lab: PC Teaching Centre

Lectures will introduce and explain the fundamentals of quantitative methods, with the opportunity to apply the method introduced in the labs later in the week.

The computer practical sessions, will give you the chance to use and apply quantitative methods to real-world data. These are primarily self-directed sessions, but with support on hand if you get stuck. Support and training in R will be provided through these sessions. Weekly sessions will be driven by empirical research questions.

Week	Topic	Format	Staff
1	Introduction Getting Started in RStudio: Knowing Merseyside	Lecture Computer Lab Practical	ZY/RM
2	Exploratory Data Analysis: UK Election	Lecture and Computer Lab Practical	ZY
3	Sampling and data manipulation: Happiness around the world	Lecture and Computer Lab Practical	ZY
4	Correlation, data reliability and the issue of scale: Health	Lecture and Computer Lab Practical	RM
5	Publication-standard Research	Lecture and Computer Lab Practical	RM
6	Online Assessment	Computer Lab	RM/ZY

Software and Data

For quantitative training sessions, ensure you have installed and/or have access to **RStudio**. To run the analysis and reproduce the code in R, you need the following software installed on your machine:

- R-4.2.2 (or later)
- RStudio 2022.12.0-353 (or later)

To install and update:

- R, download the appropriate version from The Comprehensive R Archive Network (CRAN).
- RStudio, download the appropriate version from here.

This software is already installed on University Machines. But you will need it to run the analysis on your personal devices.

Data

Example datasets could be accessed through Canvas or (some) on GitHub Repository of the module. These include:

- 2021 UK Census Data.
- 2021 Annulation Population Survey (APS) - only on Canvas.
- 2016 Family Resource Survey (FRS) - only on Canvas.
- 2011 Sample of Anonymised Records (SAR).

Note: The Annual Population Survey requires the completion of a quiz prior to its usage, as it is licensed.

Assessment

1 Lab: Getting Started in RStudio

1.1 Overview

This practical intend to prepare students who have limited experiences with R and RStudio. The content are adapted based on

- Brunsdon, Chris, and Lex Comber. 2018. *An Introduction to r for Spatial Analysis and Mapping (2e)*. Sage.
- Comber, Lex, and Chris Brunsdon. 2021. *Geographical Data Science and Spatial Data Analysis: An Introduction in r*. Sage.

1.2 Getting set up with RStudio

1.2.1 Install R and RStudio (if necessary)

R is a free, open-source programming language used for statistical analysis, data visualization, and data science

RStudio is a free front-end to R, designed to make using R easier

All of the PCs in the University PC Teaching Centre used for this class come with R and RStudio pre-installed, as do the PCs in many other University PC Teaching Centres.

However, you may wish to install R and RStudio on your own computer, or on a University PC that lacks them.

University computers: Use the *Install University Applications* app on the computer to install the latest version of RStudio (this will also install the latest version of R)

Your own computer: R and RStudio can be downloaded from the CRAN website and installed your own computer - see below for details. **A key point is that you should install R before you install RStudio.**

The simplest way to get R installed on your computer is to go the download pages on the R website - a quick search for 'download R' should take you there, but if not you could try:

- Windows: https://cran.r-project.org/bin/windows/base/

- Mac: https://cran.r-project.org/bin/macosx/
- Linux: http://cran.r-project.org/bin/linux/

The Windows and Mac version come with installer packages and are easy to install whilst the Linux binaries require use of a command terminal.

RStudio can be downloaded from https://www.rstudio.com/products/rstudio/download/ and the free version of RStudio Desktop is more than sufficient for this module and all the other things you will to do at degree level.

If you experience any problems installing R or RStudio on your own computer, bring it to one of the class lab sessions where we will be able to provide advice.

1.2.2 File management

Before you start installing software or downloading data, create a folder on your M-Drive (if working on a University networked machine) or locally if working on your own device – name this 'ENVS162' and within this create a sub-folder for each practical session. For this session, create a sub-folder called `Week1` in your `ENVS162` folder on your M-Drive. Take care to ensure you do not delete any work you do complete in the practical sessions. It is imperative that you practice good file management!

1.2.3 Open RStudio

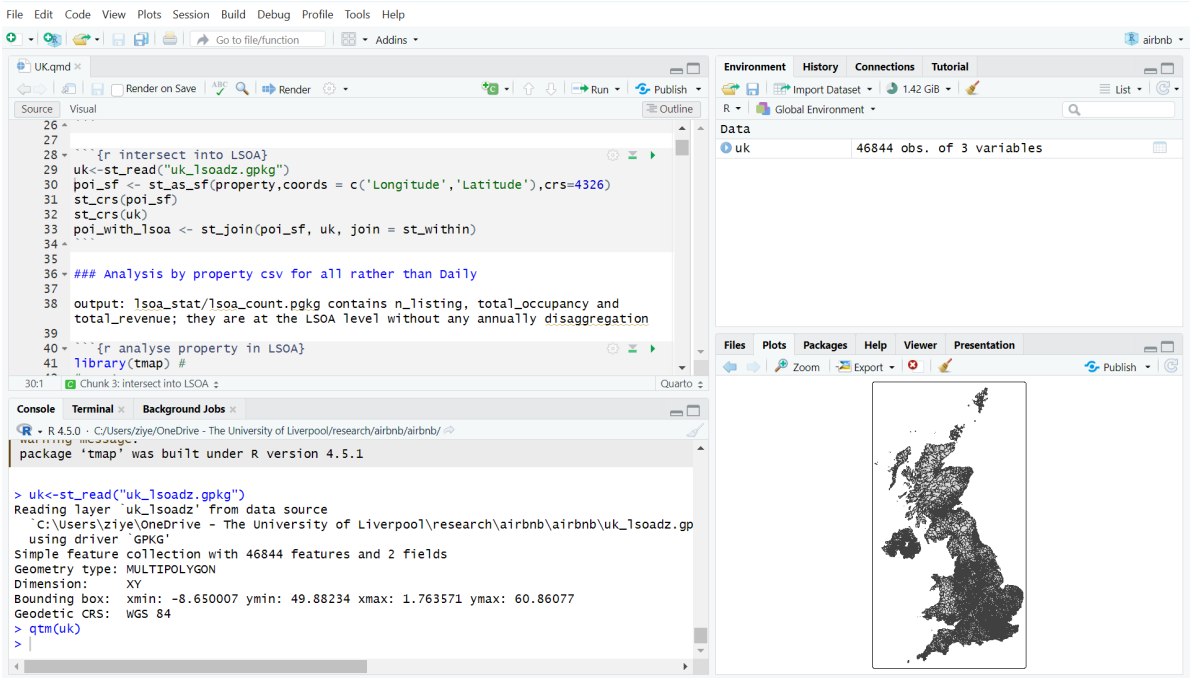
RStudio provides an interface to the different things that R can do via the 4 panes: the Console where code is entered (bottom left), a Source pane with R scripts (top left), the variables in the working Environment (top right), Files, Plots, Help etc (bottom right) - see the RStudio environment in Figure below.

In the figure above of the RStudio interface, a new script has been opened, a line of code had been written and then run in the console. The code assigns a value of 100 to `x`. The file has been saved into the current working environment. You are expected to define a similar set up for each practical as you work through the code. Note that **in the script**, anything that follows a `#` is a comment and ignored by R.

Users can set up their personal preferences for how they like their RStudio interface. Similar to straight R, there are very few pull-down menus in R, and therefore you will type lines of code into your script and run these in what is termed a *command line interface* (the console). Like all command line interfaces, the learning curve is steep but the interaction with the software is more detailed which allows greater flexibility and precision in the specification of commands.

Beyond this there are further choices to be made. Commands can be entered in two forms: directly into the *R console* window or as a series of commands into a script window. We

strongly advise that all code should be **written in a script** - (a `.R` file) and then **run from the script**. - To run code in a script, place the cursor on the line of code and then run by pressing the 'Run' icon at the top left of the script pane, or by pressing **Ctrl Enter** (PC) (or **Cmd Enter** on a Mac).



1.2.4 Ways of working

The first set of consideration relate to *how* you should work in R/RStudio. The key things to remember are:

- R is a learning curve if you have never done anything like this before. It can be scary. It can be intimidating. But once you have a bit of familiarity with how things work, it is incredibly powerful.
- You will be working from practical worksheets which will have all the code you need. Your job is to try to **understand** what the code is doing and **not** to remember the code. Comments in your code really help.
- To help you do this, the very strong suggestion is use the R scripts that are provided, and that you add your own comments to help you understand what is going on when you return to them. Comments are prefaced by a hash (**#**) that is ignored by R. Then you can save your code (with comments), run it and return to it later and modify at your leisure.

The module places a strong emphasis placed on learning by doing, which means that you encouraged to unpick the code that you are given, adapt it and play with it. It is not about remembering or being able to recall each function used but about understanding what is being done. If you can remember what you did previously (i.e. the operations you undertook) and understand what you did, you will be able to return to your code the next time you want to do something similar. To help you with this you should:

1. Always run your code from an R script… **always**! These are provided for each practical;
2. Annotate you scripts with comments. These are prefixed by a hash (`#`) in the code;
3. Save your R script to your folder;

::: {#To summarise} **To summarise...**

- You should **always** use a script (a text file containing code) for your code which can be saved and then re-run at a later date.
- You can write your own code into a script, copy and paste code into it or use an existing script (for example as provided for each of the R/RStudio practicals in this module).
- To open a new R script go to **File > New File > R Script** to open a new R file, and save it with a sensible name.
- To load an existing script file go to **File > Open File** and then navigate to your file. Or, if you have recently opened the file, go to **File > Recent Files >**.
- It is good practice to set the working directory at the beginning of your R session. This can be done via the menu in RStudio **Session > Set Working Directory > ...**. This points the R session to the folder you choose and will ensure that any files you wish to read, write or save are placed in this directory.
- To run code in a script, place the cursor on the line of code and then run by pressing the 'Run' icon at the top left of the script pane, or by pressing **Ctrl Enter** (PC) or **Cmd Enter** (Mac). :::

1.2.5 Your first R code

In this section you will undertake a few generic operations. You will:

- undertake **assignment**: the allocation of values to an R object.
- use assignment to create a **vector** of elements and a **matrix** of elements.
- undertake **operations** on R objects.
- apply some **functions** to R objects (functions nearly always return a value).

- access some of R in-built data to examine a data table (or `data.frame` which is like an Excel spreadsheet).
- do some basic **plotting**, including scatter plots and histograms.
- create data summaries.

On the way you will also be introduced to **indexing**.

First, you should **create a new R script** (see above) and save it as `week1.R` in the working directory you are using for this practical. This should be the `Week1` sub-directory you created in the `GEOG162` folder. Note that you should create a separate folder for each week's practical.

1.2.5.1 Assignment

The command line prompt in the Console window, the `>`, is an invitation to start typing in your commands.

Write the following into your script: `3+5` and run it. Recall that code is run done by either by pressing the Run icon at the top left of the script pane, or by pressing **Ctrl Enter** (PC) or **Cmd Enter** (Mac).

```
3+5
```

```
[1] 8
```

Here the result is 8. The `[1]` that precedes the output it formally indicates, *first requested element will follow*. In this case there is just one element. The `>` indicates that R is ready for another command.

Now type the following in to your script and run it:

```
y <- 3+5
y
```

```
[1] 8
```

Here the value of the `3+5` has been ***assigned*** to `y`. The syntax `y <- 3+5` can be read as `y` ***gets*** `3+5`. When `y` is invoked its value is returned (8).

For the purposes of this module, in R the equals sign (`=`) is the same as `<-`, a left diamond bracket `<` followed by a minus sign `-`. This too is interpreted by R as ***is assigned to*** or ***gets*** when the code is read **right to left**.

Now copy and paste the following into your R script and run both lines:

```r
x <- matrix(c(1,2,3,4,5,6,7,8), nrow = 4)
y = matrix(1:8, nrow = 4, byrow = T)
```

You should see the `x` appear with the `y` in the Environment pane. `y` has now been overwritten with a new assignment. If you click on the icon next to them, you will get a 'spreadsheet' view of the data you have created.

Of course you can also enter the following in the console and see what is returned:

```r
x
```

```
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

```r
y
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
```

Note In the code snippets above you have used `parentheses` - round brackets. Different kinds of brackets are used in different ways in R. Parentheses are used with **functions**, and contain the **arguments** that are passed to the function, separated by commas (`,`).

In this case the functions are `c()` and `matrix()`. The function `c()` combines or concatenates elements into a vector, and `matrix()` creates a matrix of elements in a tabular format.

In the line of code `x = matrix(c(1,2,3,4,5,6,7,8), nrow = 4)`, the arguments passed to the `matrix()` function are the vector of values `c(1,2,3,4,5,6,7,8)` and `nrow = 4`. Other kinds of brackets are used in different ways as you will see later.

One final thing to note is that the matrix is `y` is has the numbers 1 to 8, but this is specified by `1:8`. Try entering `3:65`, `19:11`, and `1.5:5` to see how the colon (`:`) works in this context.

1.2.5.2 Operations

Now you can undertake *operations* on R objects and apply *functions* to them. Write the following code into your script and then run it:

```
# x is a matrix
x
```

```
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

```
# multiplication
x*2
```

```
     [,1] [,2]
[1,]    2   10
[2,]    4   12
[3,]    6   14
[4,]    8   16
```

```
# sum of x
sum(x)
```

```
[1] 36
```

```
# mean of x
mean(x)
```

```
[1] 4.5
```

Operations can be undertaken directly using mathematical notation like `*` for multiplication or using functions like `max` to find the maximum value in an R object.

1.2.5.3 Functions

Functions are always followed by parenthesis (round brackets) `( )`. These are different from square and curly brackets `[ ]` and `{ }`. Functions always return something, a result if you like, and have the generic form:

```
# don't run this or write this into your script!
result = function(value or R object, other parameters)
```

Do not run or enter this code in your script - it is an example!

1.2.5.4 Data Tables

Here we will load a data table in `data.frame` (like a spreadsheet) in R/RStudio. R has number of in-built datasets that we can use the code below loads one of these:

```
data(mtcars)
class(mtcars)
```

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

Have a look at what is loaded by listing the objects in the current R session

```
ls()
```

```
[1] "mtcars" "x"      "y"
```

You should see the `mtcars` object. You can examine this data in a number of ways

```
# the structure of mtcars
str(mtcars)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
```

```
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
# the first six rows (or head) of mtcars
head(mtcars)
```

```
                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

The `mtcars` object is a `data.frame`, a kind of data table, and it has a number of attributes which are all numeric. The code below prints it all out to the console:

```
mtcars
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
```

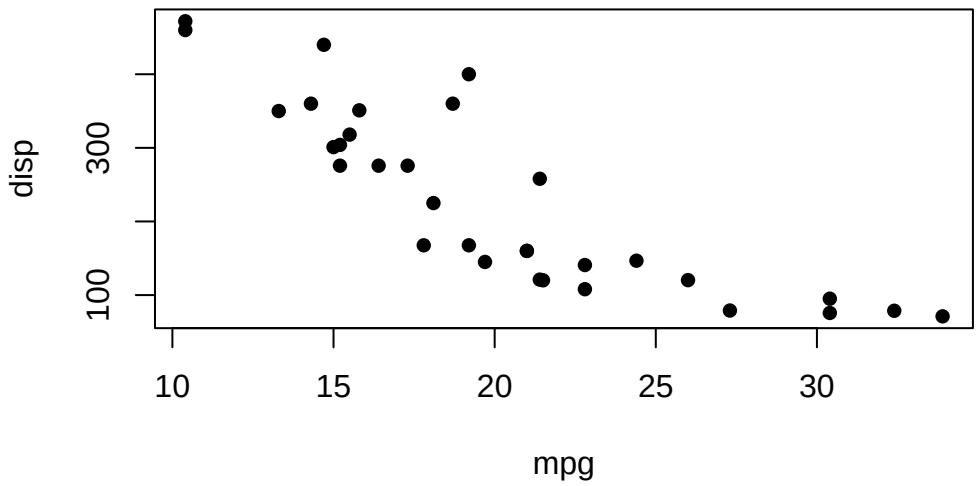
```
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Data frames are 'flat' in that they typically have a rectangular layout like a spreadsheet, with rows typically relating to observations (individuals, areas, people, houses, etc) and columns relating to their properties or attributes (height, age, etc). The columns in data frames can be of different types: vectors of numbers, factors (classes) or text strings. In matrices all of the columns have to be off the same type. Data frames are central to what we will do in R.

1.2.5.5 Plotting the data: 'Hello World!'

The code below creates a plot of 2 variables counts in the data: `mpg` and `disp`.

```r
plot(disp ~ mpg,  data = mtcars, pch=16)
```



The option `pch=16` sets the plotting character to a solid black dot. More plot characters are available - examine the help for `points()` to see these:

```
?points
```

```
starting httpd help server ... done
```

This plot can be improved greatly. We can specify more informative axis labels, change size of the text and of the plotting symbol, and so on.

We can also specify the same plot by passing named variables to the `plot` function directly as well as other parameters, as in the figure. Notice how the syntax for this is different to the `plot` function above, and the different **parameters** that are passed to the `plot()` function:

```
plot(x = mtcars$mpg, y = mtcars$disp,   pch = 1, col = "dodgerblue",
     cex = 1.5, xlab = "Miles per Gallon", ylab = "Displacement",
     main = "Hello World!")
```

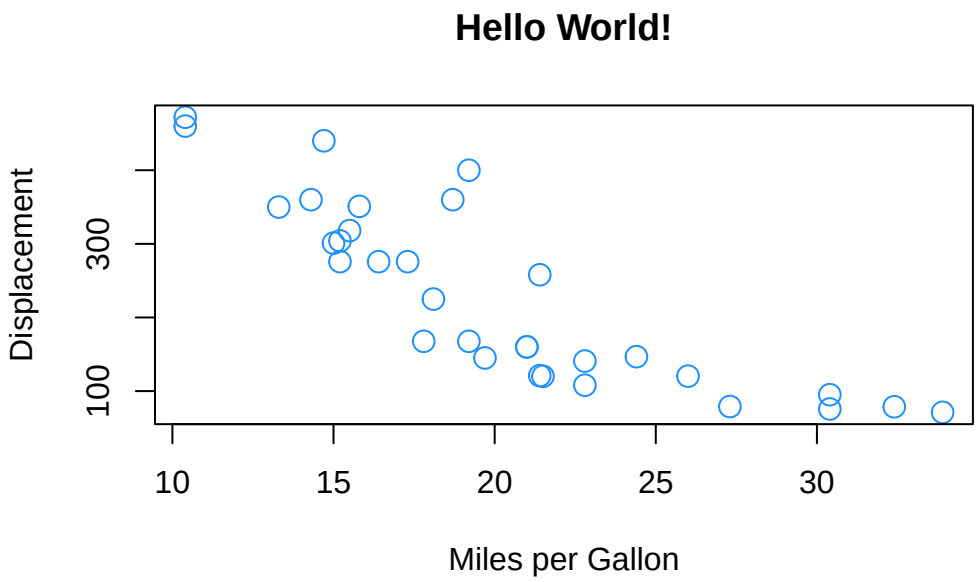


Figure 1.3: A scatter plot.

Notice how the dollar sign (`$`) is used to access variables in the `mtcars` data table compared to the first plot command, which specified `data = mtcars`.

1.2.5.6 Data summaries and indexing

We may for example require information on variables in `mtcars`. The `summary` function is very useful:

```
summary(mtcars)
```

```
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
```

```
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
       am              gear            carb
 Min.   :0.0000   Min.   :3.000   Min.   :1.000
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
 Median :0.0000   Median :4.000   Median :2.000
 Mean   :0.4062   Mean   :3.688   Mean   :2.812
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

This shows different summaries of the individual attributes in `mtcars`.

The main R graphics function is `plot()`. In addition to `plot()` there are functions for adding points and lines to existing graphs, for placing text at specified positions, for specifying tick marks and tick labels, for labelling axes, and so on.

There are various other alternative helpful forms of graphical summary. A helpful graphical summary for the `mtcars` data frame is the scatterplot matrix, shown in Figure 1.4.

```
# return the names of the mtcars variables
names(mtcars)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

```
# return the 3rd to 7th names
names(mtcars)[c(3:7)]
```

```
[1] "disp" "hp"   "drat" "wt"   "qsec"
```

```
# check what this does
c(3:7)
```

```
[1] 3 4 5 6 7
```

```
# plot the 3rd to 7th variables in mtcars
plot(mtcars[, c(3:7)], cex = 0.5,
     col = "red", upper.panel=panel.smooth)
```

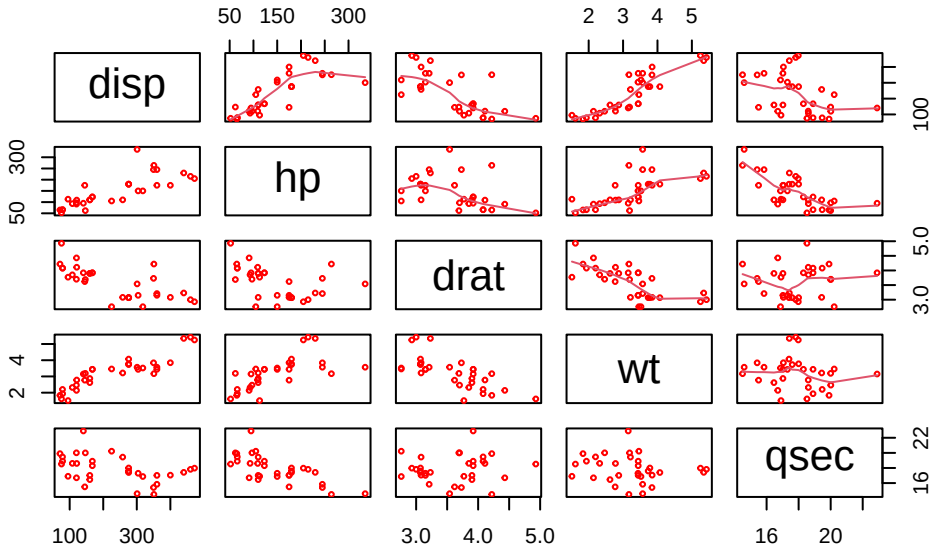


Figure 1.4: Multiple scatterplots.

The results show the correlations between the variables in the `mtcars` data frame, and the trend of their relationship is included with the `upper.panel=panel.smooth` parameter passed to `plot`.

There are number of things to notice here (as well as the figure). In particular note the use of the vector `c(2:7)` to subset the columns of `mtcars`:

- In the second line, this is was used to subset the vector of column names created by `names(mtcars)`.
- In the third line, it was printed out. Notice how `3:7` printed out all the number between 3 and 7 - very useful.
- For the plot, the vector was passed to the second argument, after the comma, in the square brackets `[,]` to indicate which columns were to be plotted.

The referencing in this way (or *indexing*) is **very important**: the individual rows and columns of 2 dimensional data structures like data frames, matrices, tibbles etc can be accessed by passing references to them in the square brackets.

```
# 1st row
mtcars[1,]
```

```
          mpg cyl disp  hp drat   wt  qsec vs am gear carb
Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4
```

```
# 3rd column
mtcars[,3]
```

```
 [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6 275.8
[13] 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0 304.0 350.0
[25] 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

```
# a selection of rows
mtcars[c(3:5,8),]
```

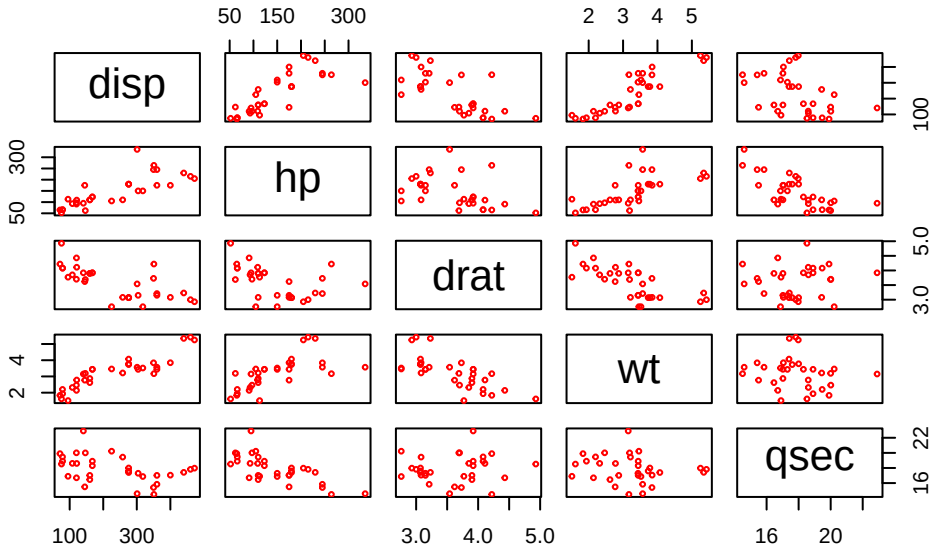
```
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Datsun 710       22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive   21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7  8 360.0 175 3.15 3.440 17.02  0  0    3    2
Merc 240D        24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

Such indexing could of course have been assigned to a R object and used to do the subsetting:

```
x = c(3:7)
names(mtcars)[x]
```

```
[1] "disp" "hp"   "drat" "wt"   "qsec"
```

```
plot(mtcars[,x], cex = 0.5, col = "red")
```



Thus indexing allows specific rows and columns to be extracted from the data as required.

Note You have encountered a second type of brackets, square brackets `[ ]`. These are used to reference or **index** positions in a vector or a data table.

Consider the object `x` above. It contains a vector of values, `3,4,5,6,7`. Entering `x[1]` would extract the first element of `x`, in this case 3. Similarly `x[4]` would return the 4th element and `x[c(1,4)]` would return the 1st and 4th elements of `x`.

However, in the examples above that index the 2-dimensional `mtcars` object, the square brackets are used to index **row** and **column** positions. The syntax for this is `[rows, columns]`. We will be using such indexing throughout this module.

1.2.5.7 Packages

The `base` installation of R includes many functions and commands. However, more often we are interested in using some particular functionality, encoded into **packages** contributed by the R developer community. Installing packages for the first time can be done at the command line in the R console using the `install.packages` command as in the example below to install the `tmap` library or via the RStudio menu via **Tools > Install Packages**.

When you install these packages it is strongly suggested you also install the *dependencies*. These are other packages that are required by the package that is being installed. This can be done by selecting check the box in the menu or including `dep=TRUE` in the command line as below (don't run this yet!):

```
# don't run this!
install.packages("tidyverse", dep = TRUE)
```

You may have to set a **mirror** site from which the packages will be downloaded to your computer. Generally you should pick one that is nearby to you.

Further descriptions of packages, their installation and their data structures will be given as needed in the practicals. There are literally 1000s of packages that have been contributed to the R project by various researchers and organisations. These can be located by name at http://cran.r-project.org/web/packages/available_packages_by_name.html if you know the package you wish to use. It is also possible to search the CRAN website to find packages to perform particular tasks at http://www.r-project.org/search.html. Additionally many packages include user guides and vignettes as well as a PDF document describing the package and listed at the top of the index page of the help files for the package.

As well as `tidyverse` you should install the `sf` package and dependencies. So we have 2 packages to install:

- `sf` for spatial data and spatial objects
- `tidyverse` for lots of lovely data science things - see https://www.tidyverse.org

You could do this in one go and this will take a bit of time:

```
install.packages(c("sf", "tidyverse"), dep = TRUE)
```

Remember: you will only have to install a package once!! So when the above code has run in your script you should comment it out. For example you might want to include something like the below in your R script.

```
# packages only need to be loaded once
# install.packages(c("sf", "tidyverse"), dep = TRUE)
```

Once the package has been installed on your computer then the package can be called using the `library()` function into each of your R sessions as below.

```
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.5.2
```

```
Warning: package 'ggplot2' was built under R version 4.5.1
```

```
Warning: package 'tibble' was built under R version 4.5.1

Warning: package 'tidyr' was built under R version 4.5.1

Warning: package 'readr' was built under R version 4.5.1

Warning: package 'purrr' was built under R version 4.5.1

Warning: package 'dplyr' was built under R version 4.5.1

Warning: package 'forcats' was built under R version 4.5.1

Warning: package 'lubridate' was built under R version 4.5.1

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.2
v ggplot2   3.5.2     v tibble    3.3.0
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.1.0
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
library(sf)
```

```
Warning: package 'sf' was built under R version 4.5.1

Linking to GEOS 3.13.1, GDAL 3.11.0, PROJ 9.6.0; sf_use_s2() is TRUE
```

1.3 Knowing Merseyside

1.3.1 Merseyside districts

Now we use these basic R command and newly installed packages to start our initial exploration by using some existing secondary dataset from the Census 2021.

In R we normally read in tabular dataset from .csv format. In your Week 1 data folder, you can find one .csv dataset: merseyside.csv. You can open them in excel to have a look, but here we are using R instead of Excel to load and examine them.

1.3.1.1 Loading tabular data

The survey data can be loaded into RStudio using the `read.csv` function.

However, you will need to tell R where to get the data from. The easiest way to do this is to use the menu if the R script file is open. Go to **Session > Set Working Directory > To Source File Location** to set the working directory to the location where your `week1.R` script is saved. When you do this you will see line of code print out in the Console (bottom left pane) similar to `setwd("SomeFilePath")`. You can copy this line of code to your script and paste into the line above the line calling the `read.csv` function.

```
# use read.csv to load a CSV file
# this is assignment to an object called `df`
df = read.csv(file = "merseyside.csv", stringsAsFactors = TRUE)
```

The `stringsAsFactors = TRUE` parameter tells R to read any character or text variables as classes or categories and not as just text.

You could inspect the help for the `read.csv` function to see the different parameters and their default values:

```
help(read.csv)
# or
?read.csv
```

Functions always return something and in this case `read.csv()` function has returned a tabular R object with 5 records and 14 fields. This has been *assigned to* `df`.

Finally in this section, lets have a look at the data. This can be done in a number of ways.

- you could look at the `df` object by entering `df` in the Console. However this is not particular helpful as it simply prints out everything that is in `df` to the Console.
- you could click on the `df` object in the Environment pane and this shows the structure of the attributes in different fields.
- you could click on the little grid-like icon next `df` in the Environment pane to get a `View` of the data and remember to close the tab that opens!.
- or you could use some code as in the examples below.

First, let's have a look at the internal structure of the data using the `str` function:

```
str(df)
```

```
'data.frame':   5 obs. of  12 variables:
 $ LAD21CD                                                                        :
 $ District                                                                       :
 $ Population                                                                     :
 $ Households                                                                     :
 $ Working.population                                                             :
 $ Full.time.students                                                             :
 $ Economic.activity.status..Economically.active..excluding.full.time.students...Unemployed:
 $ Age.over.65                                                                    :
 $ Disability                                                                     :
 $ No.central.heating                                                             :
 $ Overcrowding                                                                   :
 $ Working.from.home                                                              :
```

The **head** function does this by printing out the first six records of the data table and you may need to scroll up and down in the Console pane to see all of what is returned.

```
head(df)
```

```
    LAD21CD   District Population Households Working.population
1 E08000011   Knowsley     154519      66073              69495
2 E08000012  Liverpool     486088     207491             205749
3 E08000013 St. Helens     183248      81011              82622
4 E08000014     Sefton     279233     123075             124596
5 E08000015     Wirral     320199     143253             139500
  Full.time.students
1               7050
2              59628
3               7582
4              12636
5              14642
  Economic.activity.status..Economically.active..excluding.full.time.students...Unemployed
1                                                                                     3852
2                                                                                    13894
3                                                                                     4076
4                                                                                     6143
5                                                                                     6542
  Age.over.65 Disability No.central.heating Overcrowding Working.from.home
1       26242      34990               1020         1892             14880
2       74322     105962               4822         7352             53721
3       37642      40829               1003         1888             18973
4       64763      61134               1965         2700             34750
```

```
5       70391       73088                2125          2355              37299
```

Another way to explore the data is through the `summary` function:

```
summary(df)
```

```
     LAD21CD        District   Population       Households
 E08000011:1   Knowsley  :1   Min.   :154519   Min.   : 66073
 E08000012:1   Liverpool :1   1st Qu.:183248   1st Qu.: 81011
 E08000013:1   Sefton    :1   Median :279233   Median :123075
 E08000014:1   St. Helens:1   Mean   :284657   Mean   :124181
 E08000015:1   Wirral    :1   3rd Qu.:320199   3rd Qu.:143253
                              Max.   :486088   Max.   :207491
 Working.population Full.time.students
 Min.   : 69495     Min.   : 7050
 1st Qu.: 82622     1st Qu.: 7582
 Median :124596     Median :12636
 Mean   :124392     Mean   :20308
 3rd Qu.:139500     3rd Qu.:14642
 Max.   :205749     Max.   :59628
 Economic.activity.status..Economically.active..excluding.full.time.students...Unemployed
 Min.   : 3852
 1st Qu.: 4076
 Median : 6143
 Mean   : 6901
 3rd Qu.: 6542
 Max.   :13894
  Age.over.65      Disability     No.central.heating  Overcrowding
 Min.   :26242   Min.   : 34990   Min.   :1003       Min.   :1888
 1st Qu.:37642   1st Qu.: 40829   1st Qu.:1020       1st Qu.:1892
 Median :64763   Median : 61134   Median :1965       Median :2355
 Mean   :54672   Mean   : 63201   Mean   :2187       Mean   :3237
 3rd Qu.:70391   3rd Qu.: 73088   3rd Qu.:2125       3rd Qu.:2700
 Max.   :74322   Max.   :105962   Max.   :4822       Max.   :7352
 Working.from.home
 Min.   :14880
 1st Qu.:18973
 Median :34750
 Mean   :31925
 3rd Qu.:37299
 Max.   :53721
```

Finally in this section, we come back to the dollar sign (`$`). This is used to refer to or *extract* an individual named field or variable in an R object, like `df`.

The code below prints out the Population attribute and generates a summary of its values:

```
# extract an individual variable
df$Population
```

```
[1] 154519 486088 183248 279233 320199
```

```
# generate a summary of an individual variable
summary(df$Population)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 154519  183248  279233  284657  320199  486088
```

And of course we can use such operations to *assign* the result to new R objects. The code below extracts three variables from `df`, assigns them to `x`, `y` and `z`, and then uses the `data.frame` function to convert these into a new `data.frame` object called `my_df`

```
# extract three variables, assigning them to temporary R objects
x = df$District
y = df$Working.population
z = df$Full.time.students
# create a data.frame from these, naming the new variables
my_df = data.frame(district = x,worker = y,student = z)
```

You should have a look at what you have created:

```
head(my_df)
```

```
    district worker student
1   Knowsley  69495    7050
2  Liverpool 205749   59628
3 St. Helens  82622    7582
4     Sefton 124596   12636
5     Wirral 139500   14642
```

```
summary(my_df)
```

```
       district     worker          student
 Knowsley  :1   Min.   : 69495   Min.   : 7050
 Liverpool :1   1st Qu.: 82622   1st Qu.: 7582
 Sefton    :1   Median :124596   Median :12636
 St. Helens:1   Mean   :124392   Mean   :20308
 Wirral    :1   3rd Qu.:139500   3rd Qu.:14642
                Max.   :205749   Max.   :59628
```

The temporary R objects can be removed from the Environment using the `rm` function and a *combine* vector function, `c()` that you encountered in Week 19, that takes a vector of object names (hence they are in quotes) as its arguments.

```
rm(list = c("x","y","z"))
```

1.3.1.2 Basic data manipulation

Now we can do some basic data manipulation to know Merseyside more from the data perspective.

What is the total population in Merseyside?

```
sum(df$Population)
```

```
[1] 1423287
```

What is the total number of full-time students in Merseyside?

```
sum(df$Full.time.students)
```

```
[1] 101538
```

Which district in Merseyside has the most working population?

```
max(df$Working.population)
```

```
[1] 205749
```

Yes, using `max()` R returns use the greatest value in Working.population column. If we check back to the dataset, we know it is Liverpool. Instead, we can also ask R to tell us the answer:

```
df$District[which.max(df$Working.population)]
```

```
[1] Liverpool
Levels: Knowsley Liverpool Sefton St. Helens Wirral
```

Here, we request R to return the District which has the maximum value of the Working population.

Then, we can calculate the total number of workers that working from home:

```
sum(df$Working.from.home)
```

```
[1] 159623
```

What is the proportion of working population actually work from home in Merseyside? Yes, we need to use a division calculation of the total number of working from home vs. all the working population. R can do it by:

```
sum(df$Working.from.home) / sum(df$Working.population)
```

```
[1] 0.2566443
```

So the answer is 25.7% for the whole Merseyside - but which district has the highest proportion and which as the lowest? You may have your own guessing. But let R do the calculation:

```
df$Prop.WFH = df$Working.from.home / df$Working.population #add a new column called Prop.WFH
df #print out the df
```

```
    LAD21CD   District Population Households Working.population
1 E08000011   Knowsley     154519      66073              69495
2 E08000012  Liverpool     486088     207491             205749
3 E08000013 St. Helens     183248      81011              82622
4 E08000014     Sefton     279233     123075             124596
5 E08000015     Wirral     320199     143253             139500
  Full.time.students
1               7050
```

```
2              59628
3               7582
4              12636
5              14642
  Economic.activity.status..Economically.active..excluding.full.time.students...Unemployed
1                                                                                      3852
2                                                                                     13894
3                                                                                      4076
4                                                                                      6143
5                                                                                      6542
  Age.over.65 Disability No.central.heating Overcrowding Working.from.home
1       26242      34990               1020         1892             14880
2       74322     105962               4822         7352             53721
3       37642      40829               1003         1888             18973
4       64763      61134               1965         2700             34750
5       70391      73088               2125         2355             37299
   Prop.WFH
1 0.2141161
2 0.2610997
3 0.2296362
4 0.2789014
5 0.2673763
```

Here we ask R to add a new column named Prop.WFH which is the working from home proportion that calculated by the number of working from home people in each district divided by the total working population in that district. R will automatically do it row-by-row. We then print out the df, you may find at the very right end of the tabular, there is a new column called Prop.WFH.

You already know how to get the max of the district by the value, and we can also do that for the minimum:

```
df$District[which.max(df$Prop.WFH)]
```

```
[1] Sefton
Levels: Knowsley Liverpool Sefton St. Helens Wirral
```

```
df$District[which.min(df$Prop.WFH)]
```

```
[1] Knowsley
Levels: Knowsley Liverpool Sefton St. Helens Wirral
```

Have you got the right answer?

1.3.1.3 Your first map for Merseyside

Now let's try to do our first map in R and allow yourself know more about Merseyside.

We will use the library sf and tmap to help us at here. Run the install codes if you haven't install them. Remember: you will only have to install a package once!!

```
if (!requireNamespace("tmap")) {
  install.packages("tmap",dep =TRUE)
}
```

```
Loading required namespace: tmap
```

```
if (!requireNamespace("sf")) {
  install.packages("sf",dep =TRUE)
}
```

When they have been installed, we can start using them

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

```
Warning: package 'tmap' was built under R version 4.5.1
```

You may find in Week 1 data, we have another file named merseyside_districts.gpkg. A GeoPackage (GPKG) is a file-based format designed for storing geographic data. It supports the efficient storage and exchange of spatial datasets and can be readily used across GIS software such as QGIS and ArcGIS, as well as in programming environments including R and Python.

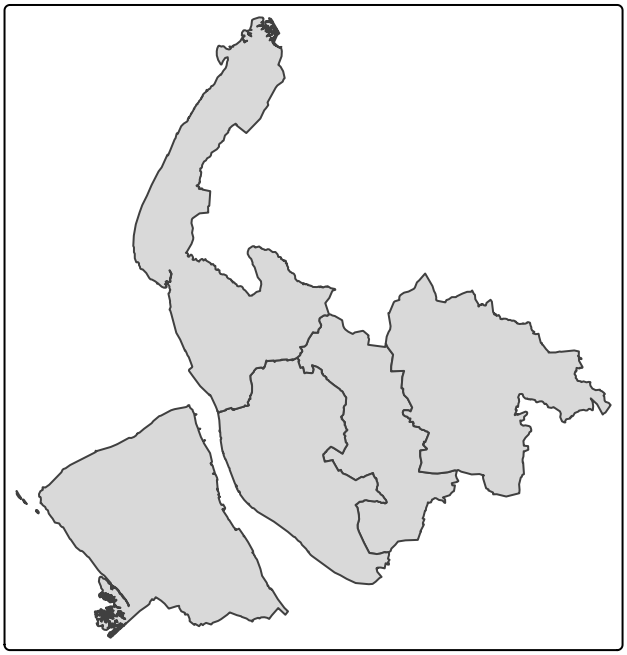
We first read it in by using the `st_read()` command in library sf.

```
sf <- st_read("merseyside_districts.gpkg")
```

```
Reading layer `lad_may_2025_uk_bgc_v2_4306843991635065087__lad_may_2025_uk_bgc_v2' from data
  using driver `GPKG'
Simple feature collection with 5 features and 8 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 318351.7 ymin: 377515.4 xmax: 361796.3 ymax: 422866.5
Projected CRS: OSGB36 / British National Grid
```

The fastest way to map it is the `qtm()` function.

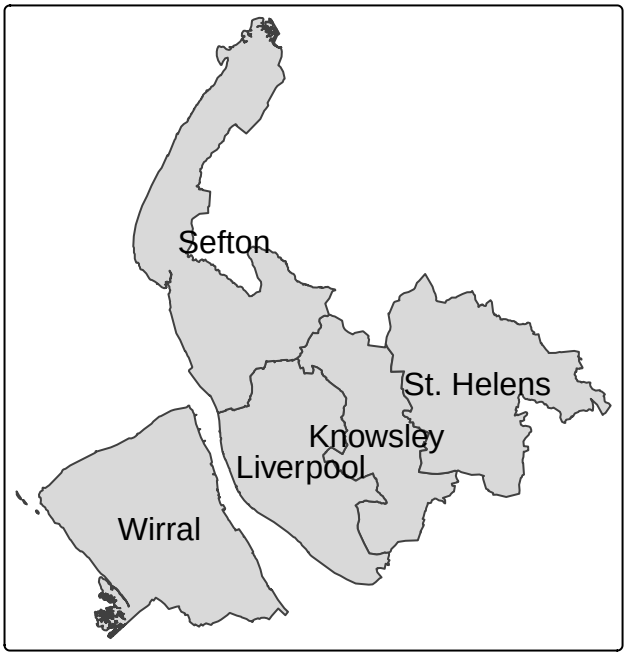
```
qtm(sf)
```



You can also add the district names on the map - which column in the sf contains district name? Use `names(sf)` to check for it.

Yes, the column should be LAD25NM. Now let's ask qtm() to also show the district names.

```
qtm(sf,text="LAD25NM")
```



But what if we want to make some meaningful maps, rather than just the boundaries of these five districts of Merseyside?

1.3.1.4 Link tabular data to geographical boundaries

Recall that in our df, we have 14 columns, containing different information about the districts. We can get all their names by using `names()`.

```
names(df)
```

```
 [1] "LAD21CD"
 [2] "District"
 [3] "Population"
 [4] "Households"
 [5] "Working.population"
 [6] "Full.time.students"
 [7] "Economic.activity.status..Economically.active..excluding.full.time.students...Unemploye
 [8] "Age.over.65"
 [9] "Disability"
[10] "No.central.heating"
[11] "Overcrowding"
[12] "Working.from.home"
[13] "Prop.WFH"
```

We can do the same thing for our geographical dataset to see what it includes:

```
names(sf)
```

```
[1] "LAD25CD"  "LAD25NM"  "LAD25NMW" "BNG_E"    "BNG_N"    "LONG"     "LAT"
[8] "GlobalID" "geom"
```

We can also show the whole sf as

```
sf
```

```
Simple feature collection with 5 features and 8 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 318351.7 ymin: 377515.4 xmax: 361796.3 ymax: 422866.5
Projected CRS: OSGB36 / British National Grid
    LAD25CD    LAD25NM LAD25NMW  BNG_E  BNG_N      LONG      LAT
1 E08000011   Knowsley          344762 393778 -2.832979 53.43789
2 E08000012  Liverpool          339359 390556 -2.913680 53.40833
3 E08000013 St. Helens          353413 395992 -2.703093 53.45862
4 E08000014     Sefton          334282 398835 -2.991771 53.48213
5 E08000015     Wirral          329109 386965 -3.067034 53.37478
                                GlobalID                           geom
1 {B4196BFE-EE90-4C31-ABD5-C7E743AE2F9B} MULTIPOLYGON (((341447.1 40...
2 {4FB47E7A-EF4E-4B9E-BF75-D4FC059CDE61} MULTIPOLYGON (((338860.9 39...
3 {943F0C6B-EB30-4C00-A42B-F6B3AEC3EFEE} MULTIPOLYGON (((349111.4 40...
4 {C6FD073B-CBEB-4E78-934A-A8FD11A20F0A} MULTIPOLYGON (((336374.5 42...
5 {88E9328B-371C-469C-91F1-3479C77D6950} MULTIPOLYGON (((331364.9 39...
```

Now we see that sf includes also the five districts, but also other geographical information. You may notice that although different column names, the first two columns of both df and sf are the district code and district name. This means what potentially we can link this two dataset together - appendix the df to sf to enrich the attributes of our geographical dataset.

```
sf2 <- left_join(sf, df,by=c("LAD25NM"="District"))
```

let's check out the new sf2 by View() it:

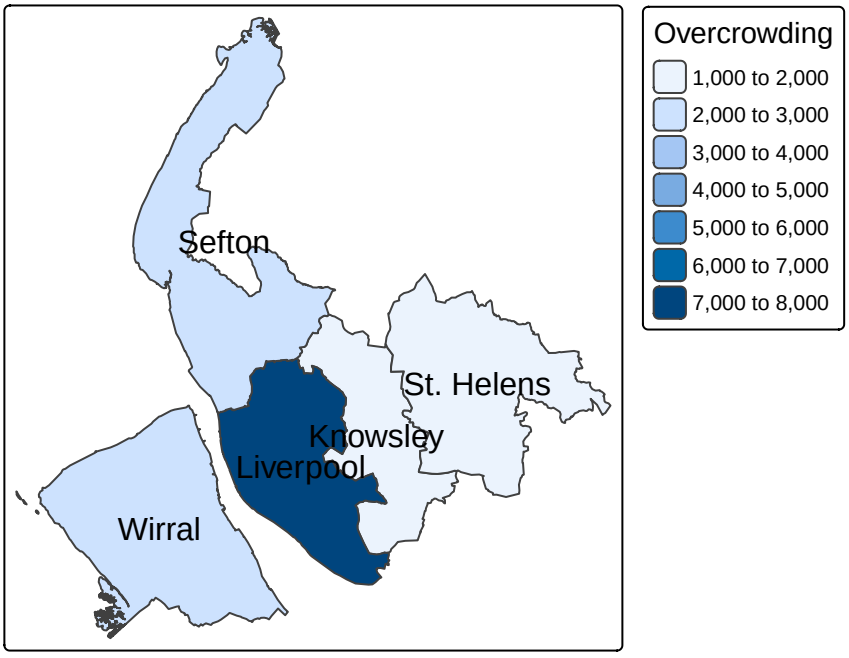
```
View(sf2)
```

In the open tab, we see all the df columns are now also attached to the sf, linking by the district names.

1.3.1.5 Choropleth map of Merseyside districts

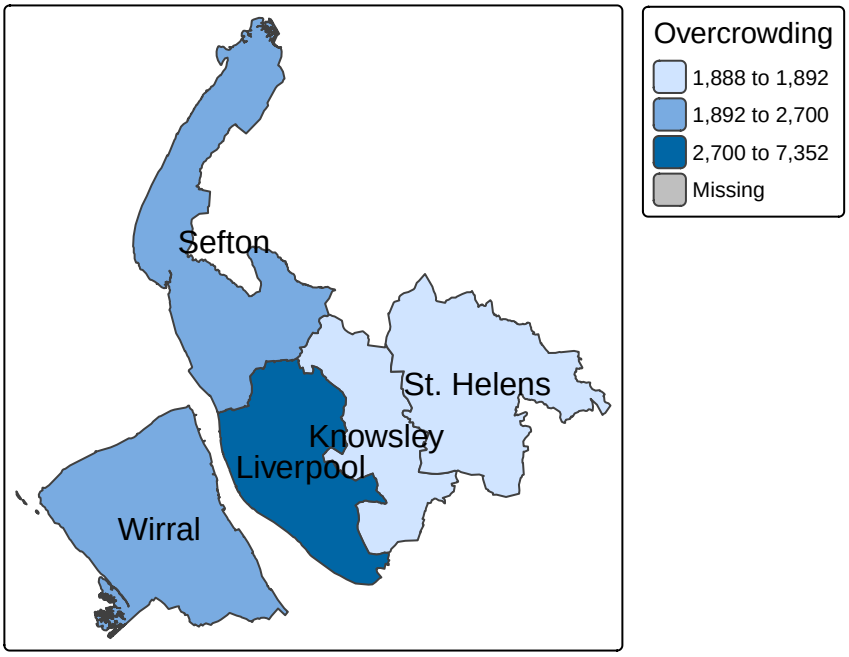
Now, we can use those new columns we attached from df to sf2 to make some meaningful choropleth maps! Here we make use of the mapping functions in tmap to do the work for us. Remember to run `library(tmap)` if you haven't.

```
tm_shape(sf2) + tm_polygons("Overcrowding") + tm_text("LAD25NM")
```



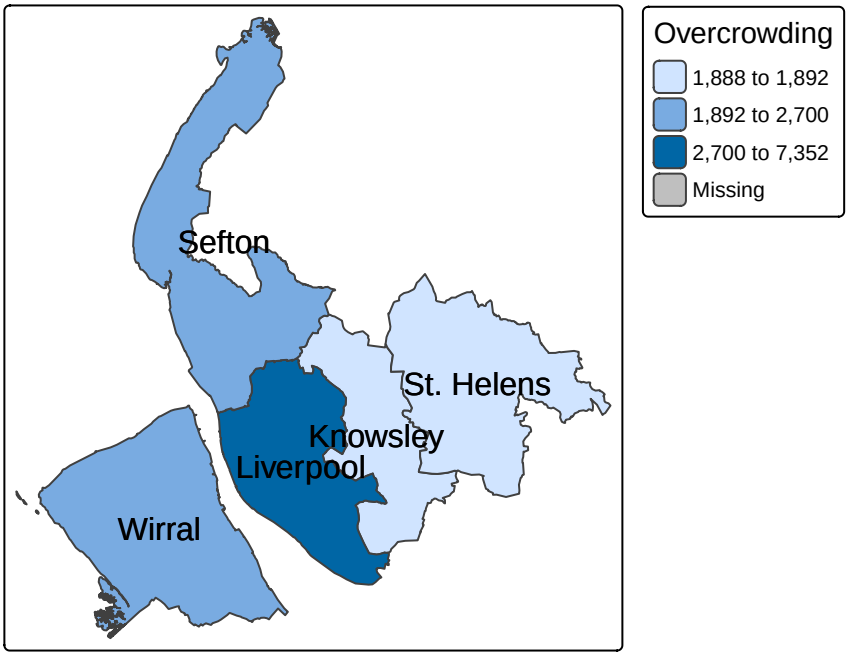

```
tm_shape(sf2) + tm_polygons("Overcrowding",style = "jenks",n=3) + tm_text("LAD25NM")
```

```
-- tmap v3 code detected -------------------------------------------------

[v3->v4] `tm_polygons()`: instead of `style = "jenks"`, use fill.scale =
`tm_scale_intervals()`.
i Migrate the argument(s) 'style', 'n' to 'tm_scale_intervals(<HERE>)'
```



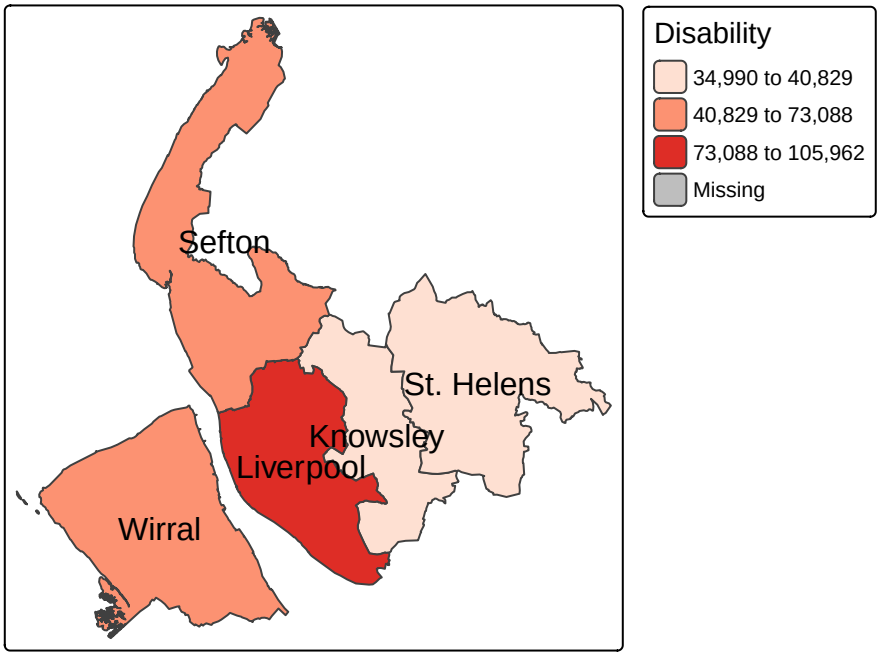

```
tm_shape(sf2) + tm_polygons("Overcrowding",style = "jenks",n=3) + tm_text("LAD25NM") + tm_tex
```

```
-- tmap v3 code detected -------------------------------------------------

[v3->v4] `tm_polygons()`: instead of `style = "jenks"`, use fill.scale =
`tm_scale_intervals()`.
i Migrate the argument(s) 'style', 'n' to 'tm_scale_intervals(<HERE>)'
```



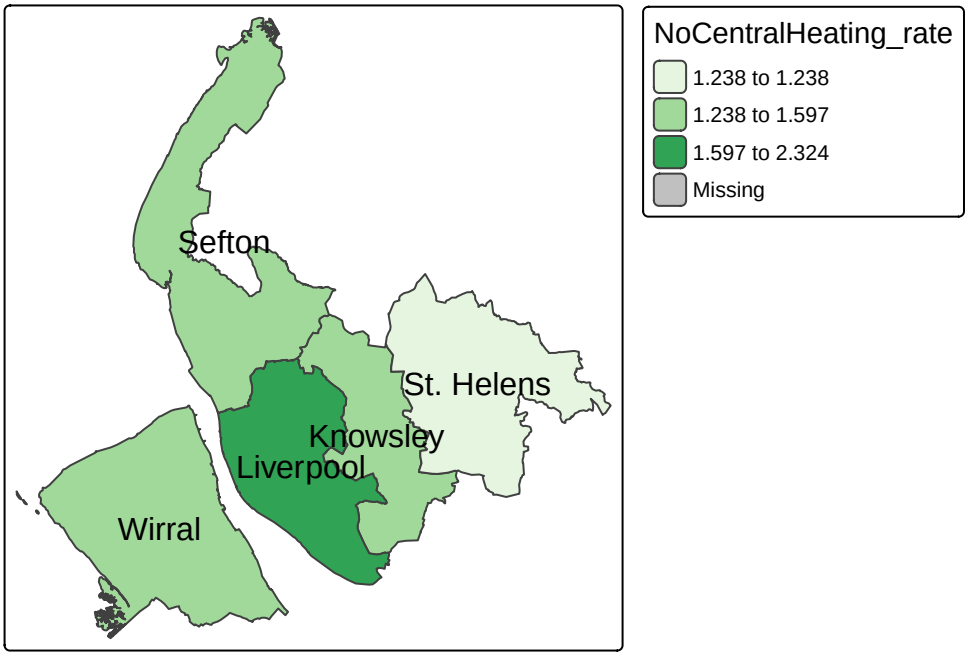

```
tm_shape(sf2) + tm_polygons("Disability",style = "jenks",n=3,palette="Reds") + tm_text("LAD2
```

```
-- tmap v3 code detected -------------------------------------------------

[v3->v4] `tm_polygons()`: instead of `style = "jenks"`, use fill.scale =
`tm_scale_intervals()`.
i Migrate the argument(s) 'style', 'n', 'palette' (rename to 'values') to
  'tm_scale_intervals(<HERE>)'
[cols4all] color palettes: use palettes from the R package cols4all. Run
`cols4all::c4a_gui()` to explore them. The old palette name "Reds" is named
"brewer.reds"
Multiple palettes called "reds" found: "brewer.reds", "matplotlib.reds". The first one, "brew
```




```
sf2$NoCentralHeating_rate = sf2$No.central.heating / sf2$Households * 100
tm_shape(sf2) + tm_polygons("NoCentralHeating_rate",style = "jenks",n=3,palette="Greens") + t
```

```
-- tmap v3 code detected -------------------------------------------------

[v3->v4] `tm_polygons()`: instead of `style = "jenks"`, use fill.scale =
`tm_scale_intervals()`.
i Migrate the argument(s) 'style', 'n', 'palette' (rename to 'values') to
  'tm_scale_intervals(<HERE>)'
[cols4all] color palettes: use palettes from the R package cols4all. Run
`cols4all::c4a_gui()` to explore them. The old palette name "Greens" is named
"brewer.greens"
Multiple palettes called "greens" found: "brewer.greens", "matplotlib.greens". The first one
```



1.3.2 Merseyside neighbourhoods

Finally you can save your R script, `week1.R` it should look something like the below:

```
# Week 1 script

# assignment
3+5
y <- 3+5
# have a look at y
y

# make matrices
x <- matrix(c(1,2,3,4,5,6,7,8), nrow = 4)
y = matrix(1:8, nrow = 4, byrow = T)
# have a look at these
x
y
# x is a matrix
x

# operations
```

```r
# multiplication
x*2
# sum of x
sum(x)
# mean of x
mean(x)

# load some inbuilt data
data(mtcars)
# inspect the class of mtcars
class(mtcars)

# list all objects in my working environment
ls()

# the structure of mtcars
str(mtcars)
# the first six rows (or head) of mtcars
head(mtcars)

# print out all of mtcars
mtcars

# plot mpg against disp
plot(disp ~ mpg,  data = mtcars, pch=16)

# the help for points
?points

# an enhanced plot using a different notation
plot(x = mtcars$mpg, y = mtcars$disp,   pch = 1, col = "dodgerblue",
     cex = 1.5, xlab = "Miles per Gallon", ylab = "Displacement", main = "Hello World!")

# summaries fo all the variables in mtcars
summary(mtcars)

# return the names of the mtcars variables
names(mtcars)
# return the 3rd to 7th names
names(mtcars)[c(3:7)]
# check what this does
c(3:7)
```

```r
# plot the 3rd to 7th variables in mtcars
plot(mtcars[, c(3:7)], cex = 0.5,
     col = "red", upper.panel=panel.smooth)

# 1st row
mtcars[1,]
# 3rd column
mtcars[,3]
# a selection of rows
mtcars[c(3:5,8),]

# assign 3:7 to x
x = c(3:7)
# get the 3rd to 7th names in mtcars using x
names(mtcars)[x]
# recreate the plot
plot(mtcars[,x], cex = 0.5, col = "red")

# some tasks
elasticband <- data.frame(stretch=c(46,54,48,50,44,42,52),
                          distance=c(148,182,173,166,109,141,166))
# have a look
elasticband

# don't run this!
# install.packages("tidyverse", dep = TRUE)

# packages only need to be loaded once
# install the packages in one go and THEN comment out
# install.packages(c("sf", "tidyverse"), dep = TRUE)

# load a package
library(sf)

## Answers to tasks
# Task 1
plot(stretch~distance, data = elasticband)
# or
plot(elasticband$stretch, elasticband$distance)

# Task 2
hist(mtcars$mpg)
```

```r
hist(mtcars$mpg, xlab='Miles per Gallon',
     main='Histogram of MPG',
     breaks = 15,
     col = 'DarkRed')

hist(mtcars$mpg, prob = T,
     xlab='Miles per Gallon',
     main='Histogram of MPG',
     breaks = 15,
     col = 'DarkRed',
     border = "#FFFFBF")
# add the probability density trend
lines(density(mtcars$mpg, na.rm=T),col='salmon',lwd=2)
# show the frequencies at the bottom - like a rug!
rug(mtcars$mpg)

# Task 3
hist(log(mtcars$mpg))
```

1.4 Summary

The aim of this session has been to familiarise you with the R environment if you have not used R before. If you have but not for a while, then hopefully this has acted as a refresher. Some key things to take away are:

- R is a learning curve, and like driving the more your practice the better you become.
- Your job is to try to **understand** what the code is doing and **not** to remember the code.
- To help with this, you should add your own comments to the script to help you understand what is going on when you return to them. Comments are prefaced by a hash (`#`) that is ignored by R.
- Always set your working directory to the sub-folder containing your R script.
- Always run your code from an R script… **always**!

The **reading** for this week is Harris (2016) Chapter 12 up to page 282. You do not have to install any packages (Section 12.2), packages will be introduced in Week 20, but you should try some of the code. Go through the illustrations in Section 12.3 (The Basics of R, starting p253), entering commands **with your comments** in the script (`prep.R`) that you created above.

Optionally you could also **briefly read** or skim Section 12.3 - the sections are mis-numbered (*A Geographical Introduction to R*, starting p261), as we will cover these in more detail in subsequent weeks and modules. Go through the Section 12.3 (*A Little More about the Workings of R*, starting on p268), again entering commands in the script that you created above. Don't worry about regression (top of p273) we will cover this later, but pay attention to *Data Frames* (p274), Referencing rows and columns (p275) and Subsetting (p279). Stop at *Reading Data* (p282).

Other good on-line *get started in R* guides include:

- The Owen guide (only up to page 28) : https://cran.r-project.org/doc/contrib/Owen-TheRGuide.pdf
- An Introduction to R - https://cran.r-project.org/doc/contrib/Lam-IntroductionToR_LHL.pdf
- R for beginners https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

1.4.1 Formative Tasks

Recall that a `data.frame` is a rectangular array of columns of data. Here you will create a data frame of two columns containing numeric values. The following data gives the distance that an elastic band moves when released for each amount it is stretched over the end of a ruler:

```
elasticband <- data.frame(stretch=c(46,54,48,50,44,42,52),
                          distance=c(148,182,173,166,109,141,166))
# have a look
elasticband
```

```
  stretch distance
1      46      148
2      54      182
3      48      173
4      50      166
5      44      109
6      42      141
7      52      166
```

The function `data.frame()` can be used to input these (or other) data directly into `data.frame` objects.

Task 1 Plot `distance` against `stretch` from the `elasticband` data frame.

Task 2 Use the `hist()` command to plot a histogram of the `mpg` values in the `mtcars` data frame. **Hints**: a) think about how the Hello World plot was parameterised and the fact that histograms are constructed from a single variable, and b) examine the help for `hist` by entering `?hist` at the console.

Task 3 Repeat 2 after taking logarithms of `disp` cover using the `log()` function - i.e. do a histogram of 'log(mtcars$mpg)

1.4.2 References

Brunsdon, Chris, and Lex Comber. 2018. *An Introduction to r for Spatial Analysis and Mapping (2e)*. Sage.

Comber, Lex, and Chris Brunsdon. 2021. *Geographical Data Science and Spatial Data Analysis: An Introduction in r*. Sage.

Harris, Richard. 2016. *Quantitative Geography: The Basics*. Sage.