PREVIEW

USING R AS A GIS

WHAT IS R?

DR. NICK BEARMAN

LOCATE PRESS

Using R as a GIS

Dr. Nick Bearman

LOCATE PRESS

Credits & Copyright

Using R as a GIS

Preview: What Is R?

by Dr. Nick Bearman

Published by Locate Press

ISBN: 978-0-9868052-6-4

Direct permission requests to info@locatepress.com or mail:
Locate Press, Suite 433, 113-437 Martin St.
Penticton, BC, Canada, V2A 5L1

Publisher Tyler Mitchell
Editor Amy Mitchell
Cover & Production Design Nathan Watson & Julie Springer
Interior Design Based on Memoir-LaTeXdocument class
Publisher Website http://locatepress.com
Book Website http://locatepress.com/book/rgis1

Contents

Introduction

Welcome to Using R as a GIS!

This book is all about the scripting language R, and how you can use it as a Geographic Information System (GIS) to manage, display, symbolise, and analyse spatial data. While initially starting out as a statistics package, now R can also create maps and perform spatial analysis - much like the tools you would find in QGIS or ArcGIS Pro.

In this book, we will cover the basics of GIS and R, as well as some more advanced visualisation and analysis techniques. We assume no prior knowledge of GIS or scripting. This book is also suitable if you have previously used a graphical desktop GIS but want to move to a script-based GIS. Two of the key benefits for adopting a script-based approach is code reusability and sharing of knowledge with others, e.g., sharing code from a research project.

The first chapter is free!

Chapter 1 - *What is R?* is available for free online and is included in this PDF. This chapter introduces some basic R commands. We have also included some bonus material showing you how to start making maps in R. Please have a look and if you like these, buy the complete book when available and enjoy the rest of the content! Sign-up to the newsletter to be notified when the final version is available.[1]

When the full book is launched, each of the five parts of the book will cover different sets of topics:

- **Part 1: What is GIS & Spatial Data?** Covers some of the key bits of theory you need to know to work with a GIS successfully. Chapter 1 (included below), gives an overview of the book and introduces some basic, core techniques in R. You will also start creating maps in Chapter 3, and customising them so they show what you want them to.

- **Part 2: Spatial Data** Covers building more advanced maps, and making use of raster and vector data. It also covers creating interactive maps in R and the differences between the SF and SP libraries.
- **Part 3: Advanced Concepts** Applies some of the GIS techniques we have learned to more advanced programming concepts in R including loops (repeating code multiple times with different data sets), if/else functions (running different bits of code depending on the data), and spatial data wrangling (getting your data into a usable format in R).
- **Part 4: Project Management** Covers some really important file management techniques, projects, version control, and R markdown. This will help you develop from running a series of scripts to do your analysis, to making full use of R and RStudio tools in your overall workflow.
- **Part 5: Next Steps** Covers some key hints and tips on where to find spatial data when you are working on your own project, what your next steps are after finishing the book, and some really useful hints and tips for finding information and help on the web.

Each chapter covers a specific topic, and many chapters have questions or an exercise at the end. It is highly recommended to do the exercises - the more practice you can do, the better! You can check your answers against the answer script available on the website.

This book is designed for people who are new to R or even new to scripting and programming. No prior knowledge is needed! Ideally, you will know a little bit of GIS already, and maybe have opened QGIS or ArcGIS. However don't worry if you haven't, we will cover the key GIS terminology that you need to know.

You can work through this book chapter by chapter or simply use it as a reference. Use the book in whatever way works for you.

[1]Register for the Locate Press newsletter: https://locatepress.com/register

1. What is R?

1.1 Learning Outcomes

After reading this chapter you will:

- Understand what R is and how it works
- Be aware of how R enables reproducible research
- Understand how to use R & RStudio
- Know some R basics

In this chapter, we discuss what R is and why it is different to many other Geographic Information System (GIS) programs.[2] While R has some limitations, it also has many benefits and advantages compared to other GIS. We will discuss some of these advantages and set the scene for the rest of the book.

We will also cover some basic concepts of using R, and link to further resources if you need more of a foundation in R, before moving on to spatial data in R in Chapter 3.

For this sample chapter we will also include some demo code showing you how to make maps in R.

1.2 R the Software

R works in a fundamentally different way to many other programs you have used. Rather than having a Graphical User Interface (GUI) with menus, buttons, and wizards to do things, R uses a command line interface. You write a command, press `Enter`, and then R does something.

This might be quite different to what you are used to, but don't be put off by this! If you are new to scripting or programming it takes a bit

[2] The R Project for Statistical Computing: r-project.org

of getting used to, but I hope I can show you the many benefits of this approach. If you have done any scripting or programming before, you will find your experience very useful.

The command line approach does have some disadvantages. One of the things I found the hardest when I started to learn R was remembering the different commands. There is a glossary at the end of this book to help with this - all of the R commands are listed, with examples of how they work.

R is also case sensitive, which means that the `read_st()` function is different to the `read_ST()` function!

However, there are many advantages to this command line system as well. R is very easily scriptable, which means because you use written commands to do your analysis, it is very easy to track exactly what you did and then share this code. For example, if your colleague needs to do the same analysis that you did, you can send them the script. They can run the script and it will repeat the commands exactly as you performed.

You can also share code with your future self. By this I mean that you can repeat the same analysis you did 6 months ago, without needing to remember exactly what you did or making copious notes. It is all there in the script.

Finally, R also allows you to use computer programming concepts like loops. This allows you to easily repeat your code, but with different data. For example, if you have a table of 20 variables for the same geographic area, you can create a map in R for one of those variables. You can then use a loop to run that code 20 times to create 20 different maps, one for each variable. If you wanted to do that in QGIS or ArcGIS, it would be a lot more work. You'll be learning how to use loops to create multiple maps within this book - see Chapter 10 for details.

1.3 Reproducible Research

Reproducible Research is a term that has become increasingly popular in academic circles since 2010, with the first term used in 1992 [3] and the concept referred to as early as 1968 [4]. It has recently come to the fore in GIScience with the developments of data science and more

advanced computational techniques [2]. However, the concept of reproducible research is key to all academic research. It has been around for a long time, underpinning scientific research from the beginning of its history.

The concept is that anyone reading an academic paper, with a suitable level of knowledge, should be able to recreate that research. Usually, this refers to experimentation but now also refers to software and statistical analysis.

R is great at reproducible research because the script records everything you do. By running the script, you can reproduce the data and results. Ensuring your research is reproducible shows that it is good science. Some conferences and journals even ask you to submit your code and data along with your article, so the reviewers can reproduce your results. R is a great language to do this in - although the principle works equally well with any programming language.

ArcGIS: Reproducible Research

If you are moving to R from ArcGIS, you might wonder whether ArcGIS can be used to create reproducible research. Fundamentally, I would say no. A key element of reproducible research is open source software, which ArcGIS is not. ArcGIS is a commercial piece of software, which is often available to academics at a heavily discounted price.

For a piece of research to be reproducible, someone needs to have the knowledge, data, process, and software to replicate your results. ArcGIS is primarily a graphical interface. This makes scripting your work more difficult although not impossible. You can use Python, but this is trickier to do - see also the box on the next page about QGIS.

More importantly, if a user doesn't have access to the software you used, they cannot replicate your work. Since ArcGIS is not an open source product they have to buy the software to use it. If they do not have access to ArcGIS, then they can't reproduce the research. Therefore, I would say that ArcGIS (or any closed source software) cannot be used in truly reproducible work.

QGIS: Reproducible Research

If you are moving to R from QGIS, you might wonder whether QGIS can be used to create reproducible research. My answer is, "yes and no." It is possible to use QGIS for reproducible research, but making your work truly reproducible is quite tricky. QGIS is open source software, which is great for reproducible research. Anyone can download and install it to replicate your work, so this is not a problem.

Some may argue that access to technology could be an issue here, and if people can't access computer technology (such as a laptop or internet connection) then the work is not really reproducible. I would say a standard laptop capable of running QGIS is not much of a requirement, and we have to draw the line somewhere. Of course, it may be slightly different if considering resource intensive analysis like high performance computing. However this is getting beyond the scope of this book!

The other element is the work itself; how easy is it to replicate? If you have a script then this process is very easy. You run the script, which repeats exactly what you did by repeating the same commands, and you get the result out at the end. QGIS is not primarily a scripting program, it uses a graphical user interface. It is much more difficult to record exactly what you did, and for another user to replicate it. You can do scripting (using Python) within QGIS, but it is not the core focus of the product. So in my view, you cannot do truly reproducible work in QGIS. QGIS of course does have many other advantages, but this is a book about R. See Chapter 7: GIS software in Bearman (2021) [1] for a comparison of different GIS software.

Another key aspect of reproducible research is version control. This allows multiple people to contribute to a project, and also allows you to 'snapshot' a specific version of your code so that you can always go back to it. See Chapter 16: Version Control for more information.

1.4 What is R & RStudio?

R has been around in some form since 1993.[3] It was originally designed as a statistics programming language, allowing users to perform all kinds of statistical analysis. One of the key features was its flexibility. R has libraries that allow users to develop custom commands. Writing these libraries is not much more difficult than writing R code. Many, many users have written their own libraries, with 18,410 available at

the time of writing.[4] This means that many types of statistical test can be run in R, including very niche applications.

The R system also allows you to pick and choose which libraries are loaded at any one time. This means that R is still relatively quick to run and doesn't get overloaded with all the additional code. Particularly relevant to us, some of these libraries allow R to handle spatial data and create maps. We will pick this up more in Chapter 3.

For the moment, just know that you can load libraries into R. These libraries provide R with many different functions, including GIS data handling, mapping, and analysis.

Since 2010, any publication talking about R will include a discussion of *RStudio*. R and RStudio are two separate programs but are closely interrelated, so much so that when people talk about 'using R' it is almost implied that they are also using RStudio.

R is the actual language itself, and the program R understands the code we write and runs it. RStudio is a program that runs on top of R and acts as an interface. It provides a nice environment in which to work with R scripts, and makes it easier to work with them. It is officially called an Integrated Development Environment (IDE).

If you do any programming in other languages, you will find that they often have an IDE as well. One programming language can have many different IDEs. Python has several IDEs, for example. There are several IDEs for R, however RStudio is by far the most popular. Figure 1.1, on the following page shows the differences between the R and RStudio interfaces.

RStudio has many more options, tools, and windows than R, which is quite barebones by comparison. We will be working with RStudio when we do the exercises in this book, but it will be R that is actually doing the processing. RStudio is just the interface to make it easier for us to work with R.

[3] R programming language: en.wikipedia.org/wiki/R_(programming_language)

[4] 18,410 libraries listed at `https://cran.r-project.org/web/packages/index.html` in May 2022.

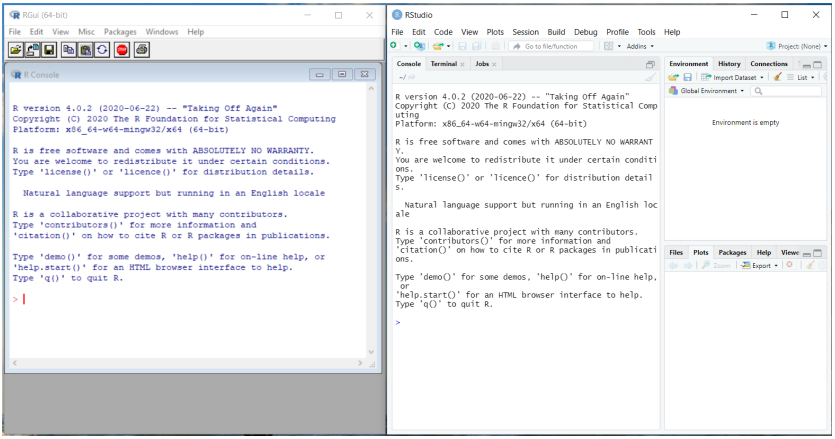


Figure 1.1: The R (left) and RStudio (right) interfaces. When people talk about working in R, most of the time they are using RStudio to write and run their code.

1.5 Using R & RStudio

This next section will contain some code snippets as well as bits of text. Feel free to use this in whatever way works best for you. You can read through the text and try out the code in RStudio as you go, or read through all the text first and then try out the code. The code is also available in `chapter1-script.R`.

When you first open RStudio there are three key sections: *Console*, *Environment*, and *Files*. See their locations in figure 1.2, on the next page.

The *Console* is where you type in your commands, and R produces feedback. For example, if you type:

```
3+4
```

in the *Console* and press `enter`, R will respond:

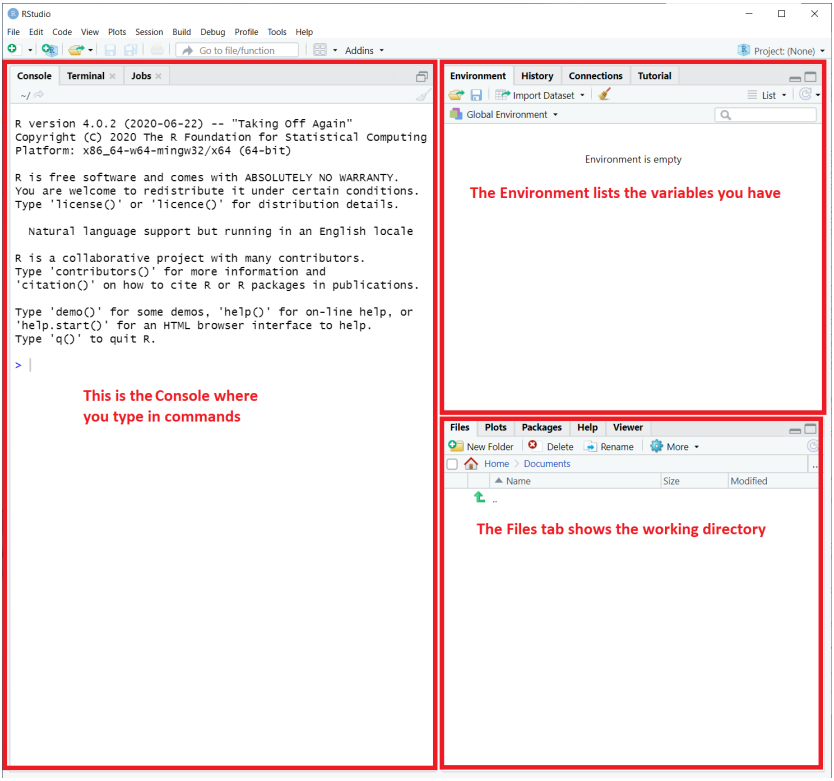


Figure 1.2: RStudio, with the Console (left), Environment (top-right) and Files (bottom-right) tabs labelled

```
[1] 7
```

You can type individual commands in R, but you can also collect lots of commands together in a script. This will be covered further in Chapter 3.

The *Environment* (top-right, figure 1.2) lists the variables R is currently using. R uses the concept of variables to store data. This could be anything from a single number, to a series of numbers, to words, to a whole spatial data layer. The same concept is used to set-up or read in variables, the <- "assign to" symbol.

The *Files* tab (bottom-right, figure 1.2, on the previous page) shows the working directory. The working directory is a key concept in R - it is where R loads files and saves files. You might have a different working directory for each project or topic you are working on. You can choose what the working directory is. Directory and folder mean the same thing and are used interchangeably.[5] When you load data into R, it will look for files in the working directory and only in the working directory. R will also save files into the working directory.

1.5.1 Installing R & RStudio

Install R & RStudio before moving further on in this book. Instructions are available.[6] If you have issues installing R, RStudio, or the `sf` and `tmap` libraries, you can use *RStudio.Cloud* instead. RStudio.Cloud is a website that allows you to run RStudio from within a web browser tab. Details are in the instructions listed above.

1.5.2 R Basics Worksheet

This worksheet takes you though some basic R commands and basic concepts in R, with no prior knowledge of R needed. However, if you want some more information on R, how it works, and basic commands in R, check it out.[7]

Type what you see below into the *Console* section of RStudio.

Any line prefixed with a `#` is a comment - a note for you, something the computer ignores. You don't need to type in these lines. However, commenting your code is really important. We will talk more about this when we get to scripts.

R can initially be used as a calculator. Enter the following into the left-hand side of the window, the section labelled *Console*:

[5] The term directory was used exclusively prior to 1995. When Microsoft released Windows 95, in 1995, they introduced the term folder. The two have been used interchangeably ever since.

[6] Installing R and RStudio: `https://nickbearman.github.io/installing-software/r-rstudio`

[7] Getting Started with R: `https://rpubs.com/nickbearman/gettingstartedwithr`

```
6 + 8
```

You will see R outputs this:

```
[1] 14
```

Don't worry about the `[1]` for the moment. Just note that R printed out `14` since this is the answer to the sum you typed in.

We can also do multiplication:

```
5 * 4
[1] 20
```

Also note that `*` is the symbol for multiplication here. The last command asked R to perform the calculation '5 multiplied by 4'. Other symbols are `-` for subtraction and `/` for division:

```
12 - 14
[1] -2

6 / 17
[1] 0.3529412
```

You can also assign the answers of the calculations to variables and use them in calculations:

```
price <- 300
```

Here, the value `300` is stored in the variable `price`. The `<-` symbol means put the value on the right into the variable on the left. It is typed with a `<` followed by a `-`. The variables are shown in the window labelled *Environment*, in the top right (see figure 1.3, on the following page).

Variables can be used in subsequent calculations. For example, to apply a 20% discount to this price you could enter the following:

```
price - price * 0.2
[1] 240
```

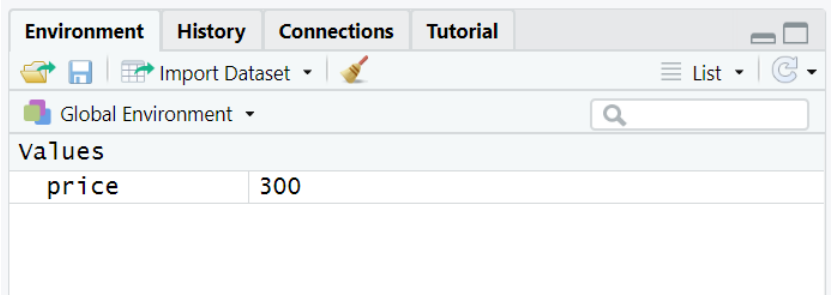


Figure 1.3: The 'price' variable, listed in the Environment

or use intermediate variables:

```
discount <- price * 0.2
price - discount
[1] 240
```

R can also work with lists of numbers, as well as individual ones. Lists are specified using the *c* function. Suppose you have a list of house prices specified in thousands of pounds. You could store them in a variable called house.prices like this:

```
house.prices <- c(120,150,212,99,199,299,159)
```

Running this code will then show us what is stored in the house.prices variable:

```
house.prices
[1] 120 150 212  99 199 299 159
```

Note that there is no problem with full stops or a period in the middle of variable names. You can then apply functions to the lists:

```
mean(house.prices)
[1] 176.8571
```

If the house prices are in thousands of pounds, then this tells us that the mean house price is 176,900 GBP.

1.5.3 The Data Frame

R has a way of storing data in an object called a *data frame*. This is rather like a spreadsheet, where all the relevant data items are stored together as a set of rows and columns.

Below is a CSV file of house prices and burglary rates, which you can load into R. You can use a function called `read.csv()` which, as you might guess, reads CSV files. Run the line of code below, which loads the CSV file into a variable called hp.data.:

```
hp.data <- read.csv("https://locatepress.com/files/rgis/hpdata.csv")
```

When you read in data, it is always a good idea to check it came in okay. To do this, you can preview the data set. The `head` command shows the first six rows of the data:

```
head(hp.data)
```

```
  ID Burglary Price
1 21        0   200
2 24        7   130
3 31        0   200
4 32        0   200
5 78        6   180
6 80       19   140
```

You can click on the variable listed in the *Environment* window, which will show the data in a new tab. You can also enter the code below to open a new tab showing the data (figure 1.4, on the next page):

```
View(hp.data)
```

You can also describe each column in the data set using the `summary` function:

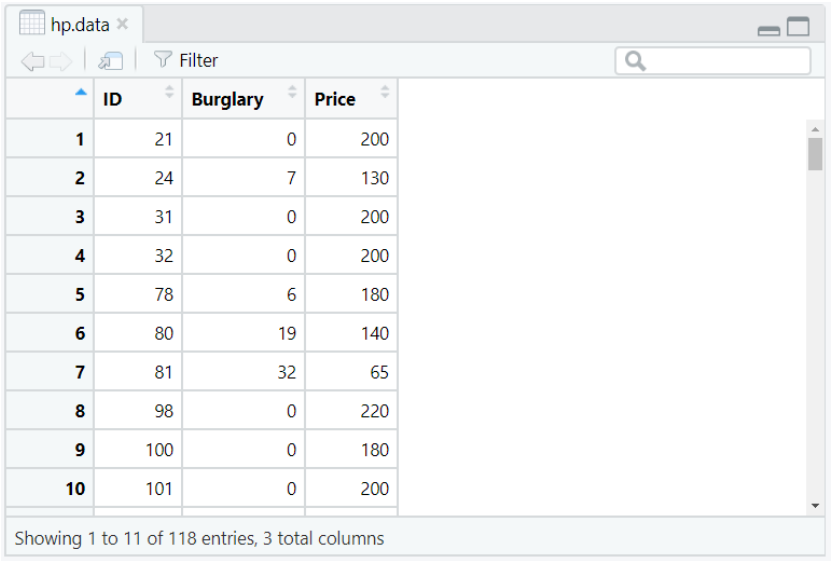


Figure 1.4: The *hp.data* variable, shown as a table, using the *View()* function

```
summary(hp.data)
```

```
      ID           Burglary          Price      
Min.   : 21.0   Min.   : 0.000   Min.   : 65.0  
1st Qu.:615.5   1st Qu.: 0.000   1st Qu.:152.5  
Median :846.5   Median : 0.000   Median :185.0  
Mean   :654.0   Mean   : 5.644   Mean   :179.0  
3rd Qu.:875.8   3rd Qu.: 7.000   3rd Qu.:210.0  
Max.   :905.0   Max.   :37.000   Max.   :260.0  `
```

For each column, a number of values are listed:

```
=======  ================================
Item     Description
=======  ================================
Min.     The smallest value in the column
1st. Qu. The first quartile
Median   The median
Mean     The average of the column
3rd. Qu. The third quartile
Max.     The largest value in the column
```

Based on these numbers, an impression of the spread of values of each variable can be obtained. In particular, it is possible to see that the median house price in St. Helens by neighbourhood ranges from 65,000 GBP to 260,000 GBP. Half of the prices lie between 152,500 GBP and 210,000 GBP. Also, it can be seen that since the median measured burglary rate is zero, then at least half of areas had no burglaries in the month when counts were compiled.

Square brackets can be used to look at specific sections of the data frame, for example `hp.data[1,]` or `hp.data[,1]`. You can also delete columns and create new columns using the code below. Remember to use the `head()` command as we did earlier to look at the data frame. Try the code below, and see what it does. The output has not been included here for brevity:

```
#create a new column in hp.data call counciltax, storing the value NA
hp.data$counciltax <- NA

#see what has happened
head(hp.data)

#delete a column
hp.data$counciltax <- NULL

#see what has happened
head(hp.data)

#rename a column
colnames(hp.data)[3] <- "Price-thousands"

#see what has happened
head(hp.data)
```

1.6 Summary

In this chapter we have covered:

- How R works
- Reproducible research
- R & RStudio
- R basics

There is much more we can do in R, but these are the building blocks you will base the rest of your learning on. Check out the bonus material below!

1.7 Bonus Material

Here is some bonus material for this sample chapter, from subsequent chapters in the book.

We are going to use two libraries in this section. Type the code below into the console to install them. Make sure you have installed them both. `tmap` is quite a big library and may take a while to install.

```
install.packages("sf")
install.packages("tmap")
```

If you have problems installing the libraries, check the error messages. Sometimes it will mention a dependent library. If you install the dependent library and then install the main library, it will work. It is also worth trying either restarting RStudio or your computer to see if that solves the problem. You can also find answers on https://nickbearman.github.io/installing-software/r-rstudio under Troubleshooting.

1.7.1 Load a shapefile

Once we have set our working directory, we can load in a file from it. Download the `sthelens.zip` file from the book website, and save it into your working directory.

This is a zip file - a series of files combined together to make it easier to download. This is very common with shapefiles, as they are made up of 3 (or more) separate files that need to be kept together. You need to extract the files - do this by double clicking and choose *Extract All*. You may not need to click *Extract All* if you are on a Mac.

This will extract the files into a folder, but make sure you check if they have extracted into a sub-folder. If they have, you will need to move them to your working directory. If they are in a sub-folder of your working directory, R will not be able to find them.

Once this is done, run the following command to read them in:

```
library(sf)
sthelens <- st_read("sthelens.shp")
```

This command uses the function `st_read()`. It reads in the file `sthelens.shp` (from the working directory) and saves it as a variable called `sthelens`.

> The file name and the variable are not related. We could have used this code: `banana <- st_read("sthelens.shp")` which would call our variable *banana*. But it's important to have clear variable names, and often makes sense for them to be the same as the file name - which should explain what information is stored in that variable.

It should give you the output below:

```
Reading layer 'sthelens' from data source

'C:\Users\nick\Documents\r-gis\data\sthelens.shp'
  using driver 'ESRI Shapefile'
Simple feature collection with 118 features and 5 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 345350.2 ymin: 387836.6 xmax: 361798.8 ymax: 404145.6
Projected CRS: Transverse_Mercator
```

You can also see that `sthelens` is listed in your *Environment* tab (top right of the RStudio window).

St. Helens is now stored as a Simple Features (or `sf`) object. You can use the `qtm()` function to draw the polygons (for example, the map of the LSOA - a type of administrative area in the UK).

```
library(tmap)
qtm(sthelens)
```

`qtm()` stands for *Quick Thematic Map*, and is a quick way of getting a map output. It shows us that we have read the data in properly, and what the data looks like (figure 1.5, on the following page).

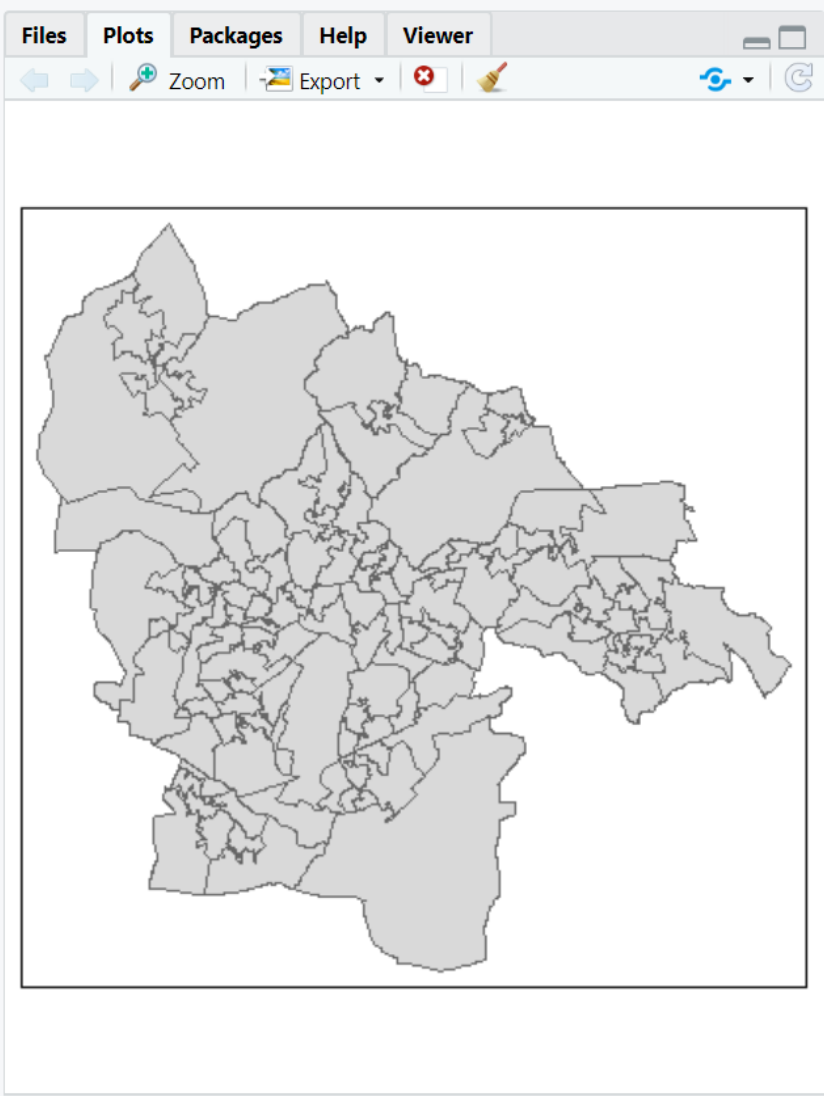


Figure 1.5: Output in the Plots window from *qtm(sthelens)*

Along with the spatial element of vector data, each file has an attribute table. This is a table of data, with each row linked to a specific polygon. We can use the `head()` command to view the first six rows of the attribute table.

```
head(sthelens)
```

```
Simple feature collection with 6 features and 5 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 345350.2 ymin: 387836.6 xmax: 356013.9 ymax: 403054
Projected CRS: Transverse_Mercator
ID  GID            NAME         LABEL  ZONECODE
1 21 2049 St. Helens 009D 04BZE01006825 E01006825
2 24 2052 St. Helens 001B 04BZE01006883 E01006883
3 31 3564 St. Helens 022A 04BZE01006898 E01006898
4 32 3565 St. Helens 001D 04BZE01006885 E01006885
5 78 4389 St. Helens 023C 04BZE01006891 E01006891
6 80 4391 St. Helens 018C 04BZE01006828 E01006828
                        geometry
1 POLYGON ((348074.3 395549.9...
2 POLYGON ((347824.8 401169.7...
3 POLYGON ((351561.1 390368, ...
4 POLYGON ((351662.3 402905.2...
5 POLYGON ((348260 391439, 34...
6 POLYGON ((348188.1 392976.1...
```

This is the same as the attribute table in programs like ArcGIS, QGIS, or MapInfo. If you want to open the shapefile in QGIS or ArcGIS to have a look, feel free to.

As well as the attribute table, we can see quite a range of other information. The header info says what type of vector file it is:

```
Geometry type: POLYGON
```

and what projection and coordinate system the data are in:

```
Projected CRS: Transverse_Mercator
```

We also then have the columns themselves: `geometry` is the coordinates of the polygons. There are a few others, including `ID` which is the one we are interested in.

We are going to join this data to the `hp.data` variable. Both data sets have the `ID` column in them, and this is what we are going to use to join them (Figure 1.6)

```
> head(sthelens)
Simple feature collection with 6 features and 5 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 345350.2 ymin: 387836.6 xmax: 356013.9 ymax: 403054
Projected CRS: Transverse_Mercator
  ID  GID            NAME         LABEL  ZONECODE                      geometry
1 21 2049 St. Helens 009D 04BZE01006825 E01006825 POLYGON ((348074.3 395549.9...
2 24 2052 St. Helens 001B 04BZE01006883 E01006883 POLYGON ((347824.8 401169.7...
3 31 3564 St. Helens 022A 04BZE01006898 E01006898 POLYGON ((351561.1 390368, ...
4 32 3565 St. Helens 001D 04BZE01006885 E01006885 POLYGON ((351662.3 402905.2...
5 78 4389 St. Helens 023C 04BZE01006891 E01006891 POLYGON ((348260 391439, 34...
6 80 4391 St. Helens 018C 04BZE01006828 E01006828 POLYGON ((348188.1 392976.1...

> head(hp.data)
  ID Burglary Price-thousands
1 21        0             200
2 24        7             130
3 31        0             200
4 32        0             200
5 78        6             180
6 80       19             140
```

Figure 1.6: The *head()* of *sthelens* and *hp.data*, showing the *ID* field which we are going to use for the join

The `merge` function joins the data. In this case, R is clever enough to look at both data sets, see there is a field called `ID` in both datasets, and join them automatically. If the fields weren't called the same thing we would need to specify the fields - and we will do this in a later example:

```
sthelens <- merge(sthelens, hp.data)
```

We can use the `head` function to check the data have been joined correctly:

```
head(sthelens)
```

```
Simple feature collection with 6 features and 7 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 346795.9 ymin: 389514.8 xmax: 351909.3 ymax: 404145.6
Projected CRS: Transverse_Mercator
```

```
ID  GID            NAME         LABEL     ZONECODE  Burglary Price-thousands
1  100 4411 St. Helens 001C 04BZE01006884 E01006884        0             180
2  101 4412 St. Helens 001F 04BZE01006887 E01006887        0             200
3  102 4413 St. Helens 001E 04BZE01006886 E01006886       15             210
4  111 6849 St. Helens 023B 04BZE01006889 E01006889        6             170
5  112 6850 St. Helens 022E 04BZE01006910 E01006910       12             180
6  113 6851 St. Helens 019H 04BZE01006913 E01006913        8             160
```

Now we can see how the data have been joined.

Finally, for this chapter, we can use `qtm()` again to draw a quick map to show the burglary data (Figure 1.7, on the following page):

```
qtm(sthelens,  fill="Burglary")
```

This shows the data for us, and is a quick way to generate a map. We don't have any way of customising it, but we can use a different function for this - `tm_shape()` - which we will look at in the next chapter.

You can export the map if you want to:

- click on the *Export* button
- choose *Copy to Clipboard...*
- choose *Copy Plot*

If you also have Word up and running, you can then paste the map into your document. You can also save the map as an image or PDF. There are also other ways of exporting maps from R, which we will come to in due course.

Here is a list of useful commands in R: https://github.com/nickbearman/intro-r-spatial-analysis/raw/master/glossary-helpsheet.pdf.

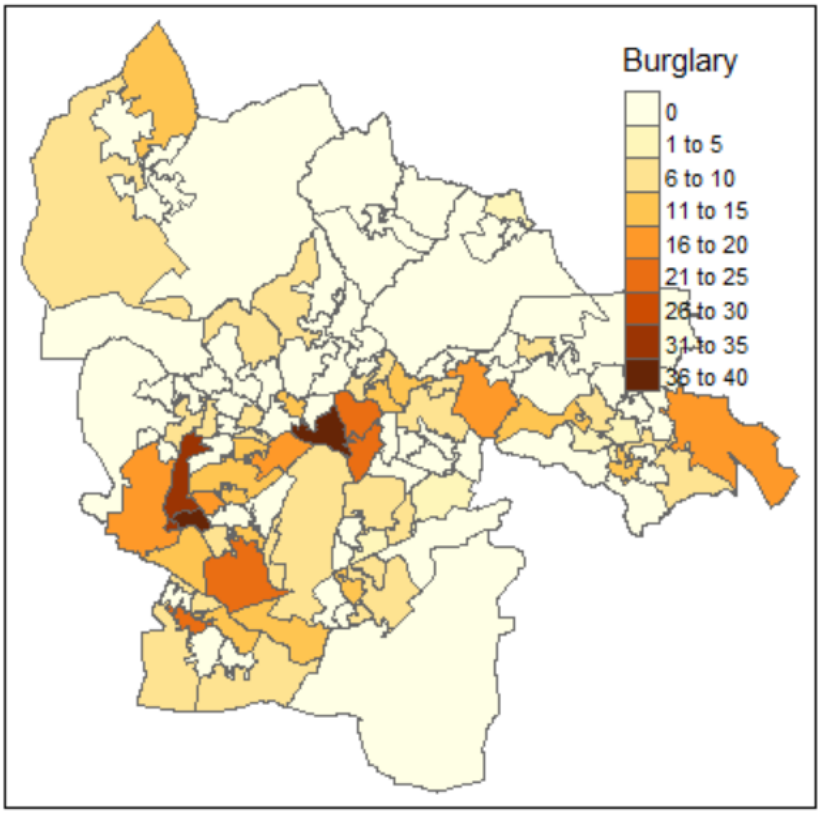


Figure 1.7: Burglary data for St Helens

1.7.2 What Next?

This e-book is a just short taster of how we go about making maps in R. There are many more things we can do. For more details, see the full book when it is available! To receive notice when it is launched, subscribe to our newsletter and you will be one of the first to know.[8]

[8]Subscribe to Locate Press newsletter: https://locatepress.com/subscribe

Bibliography

[1] N. Bearman. GIS: Research methods. Available at: https://www.bloomsbury.com/uk/gis-9781350129559/, 2021. Accessed: 31 May 2021.

[2] C. Brunsdon and A. Comber. Opening practice: supporting reproducibility and critical spatial data science. *Journal of Geographical Systems*, 23(4):477–496, 2021. doi:10.1007/s10109-020-00334-2.

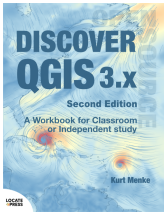
[3] J.F. Claerbout and M. Karrenbach. Electronic documents give reproducible research a new meaning. In *SEG Technical Program Expanded Abstracts 1992*, pages 601—604. Society of Exploration Geophysicists (SEG Technical Program Expanded Abstracts), 1992. doi:10.1190/1.1822162.

[4] D.T. Lykken. Statistical significance in psychological research. *Psychological Bulletin*, 70(3):151—159, 1968.

Books from Locate Press

Be sure to visit http://locatepress.com for information on new and upcoming titles.

Discover QGIS 3.x - Second Edition



EXPLORE THE LATEST LONG TERM RELEASE (LTR) OF QGIS!

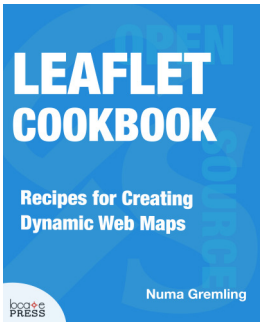
Discover QGIS 3.x - Second Edition is a comprehensive update to the original workbook built for both the classroom and professionals looking to build their skills.

Designed to take advantage of the latest QGIS features, this book will guide you in improving your maps and analysis.

Discover QGIS 3.x covers Spatial analysis, Data management, and Cartography. The book includes new exercises, and several new sections.

The book is a complete resource and includes: lab exercises, challenge exercises, all data, discussion questions, and solutions.

Leaflet Cookbook



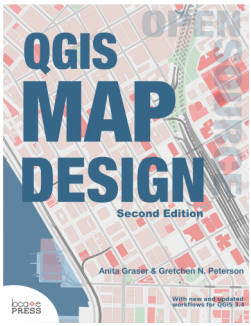
COOK UP DYNAMIC WEB MAPS USING THE RECIPES IN THE LEAFLET COOKBOOK.

Leaflet Cookbook will guide you in getting started with Leaflet, the leading open-source JavaScript library for creating interactive maps. You'll move swiftly along from the basics to creating interesting and dynamic web maps.

Even if you aren't an HTML/CSS wizard, this book will get you up to speed in creating dynamic and sophisticated web maps. With sample code and complete examples, you'll find it easy to create your own maps in no time.

A download package containing all the code and data used in the book is available so you can follow along as well as use the code as a starting point for your own web maps.

QGIS Map Design - 2nd Edition



LEARN HOW TO USE QGIS 3 TO TAKE YOUR CARTOGRAPHIC PRODUCTS TO THE HIGHEST LEVEL.

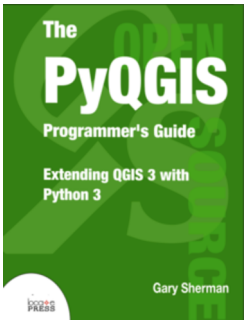
QGIS 3.4 opens up exciting new possibilities for creating beautiful and compelling maps!

Building on the first edition, the authors take you step-by-step through the process of using the latest map design tools and techniques in QGIS 3. With numerous new map designs and completely overhauled workflows, this second edition brings you up to speed with current cartographic technology and trends.

See how QGIS continues to surpass the cartographic capabilities of other geoware available today with its data-driven overrides, flexible expression functions, multitudinous color tools, blend modes, and atlasing capabilities. A prior familiarity with basic QGIS capabilities is assumed. All example data and project files are included.

Get ready to launch into the next generation of map design!

The PyQGIS Programmer's Guide



WELCOME TO THE WORLD OF PYQGIS, THE BLENDING OF QGIS AND PYTHON TO EXTEND AND ENHANCE YOUR OPEN SOURCE GIS TOOLBOX.

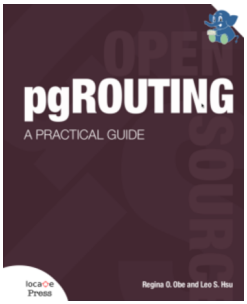
With PyQGIS you can write scripts and plugins to implement new features and perform automated tasks.

This book is updated to work with the next generation of QGIS—version 3.x. After a brief introduction to Python 3, you'll learn how to understand the QGIS Application Programmer Interface (API), write scripts, and build a plugin.

The book is designed to allow you to work through the examples as you go along. At the end of each chapter you will find a set of exercises you can do to enhance your learning experience.

The PyQGIS Programmer's Guide is compatible with the version 3.0 API released with QGIS 3.x and will work for the entire 3.x series of releases.

pgRouting: A Practical Guide



WHAT IS PGROUTING?

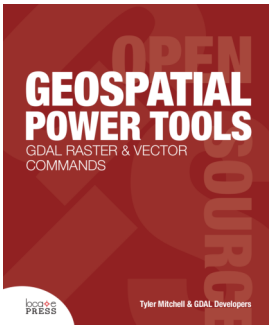
It's a PostgreSQL extension for developing network routing applications and doing graph analysis.

Interested in pgRouting? If so, chances are you already use PostGIS, the spatial extender for the PostgreSQL database management system.

So when you've got PostGIS, why do you need pgRouting? PostGIS is a great tool for molding geometries and doing proximity analysis, however it falls short when your proximity analysis involves constrained paths such as driving along a road or biking along defined paths.

This book will both get you started with pgRouting and guide you into routing, data fixing and costs, as well as using with QGIS and web applications.

Geospatial Power Tools



EVERYONE LOVES POWER TOOLS!

The GDAL and OGR apps are the power tools of the GIS world—best of all, they're free.

The utilities include tools for examining, converting, transforming, building, and analysing data. This book is a collection of the GDAL and OGR documentation, but also includes new content designed to help guide you in using the utilities to solve your current data problems.

Inside you'll find a quick reference for looking up the right syntax and example usage quickly. The book is divided into three parts: *Workflows and examples*, *GDAL raster utilities*, and *OGR vector utilities*.

Once you get a taste of the power the GDAL/OGR suite provides, you'll wonder how you ever got along without them.
