Introduction to Mapping in R

NOTE: You can download the source files for this book from here. The source files are in the format of R Notebooks. Notebooks are pretty neat, because the allow you execute code within the notebook, so that you can work interactively with the notes.

Spatial statistics is a sub-field of spatial analysis that has grown in relevance in recent years as a result of 1) the availability of information that is geo-coded, in other words, that has geographical references; and 2) the availability of software to analyze such information.

A key technology fuelling this trend is that of Geographical Information Systems (GIS). GIS are, in simplest terms, digital mapping for the 21st century. In most cases, however, GIS go beyond cartographic functions to also enable and enhance our ability to analyze data.

There are many available packages for geographical information analysis. Some are very user friendly, and widely available in many institutional contexts, such as ESRI's Arc software. Others are fairly specialized, such as Caliper's TransCAD, which implements many operations of interest for transportation engineering and planning.

Others packages have the advantage of being more flexible and/or free.

Such is the case of the `R` statistial computing language. `R` has been adopted by many in the spatial analysis community, and a number of specialized libraries have been developed to support mapping and spatial data analysis functions.

The objective of this note is to provide an introduction to mapping in `R`. Maps are one of the fundamental tools of spatial statistics and spatial analysis, and `R` allows for many GIS-like functions.

If you wish to work interactively with this chapter you will need the following:

- An R markdown notebook version of this document (the source file).
- A package called `geog4ga3`.

In the previous reading/practice you created a simple proportional symbols map. In this reading/practice you will learn how to create more sophisticated maps in `R`.

Learning Objectives

In this reading, you will:

1. Revisit how to install and load a package.
2. Learn how to invoke a data and view the data structure.
3. Learn how to easily create maps using `R`.
4. Think about how statistical maps help us understand patterns.

Suggested Readings

- Bivand RS, Pebesma E, Gomez-Rubio V [-@Bivand2008] Applied Spatial Data Analysis with R, Chapters 2-3. Springer: New York
- Brunsdon C and Comber L [-@Brunsdon2015R] An Introduction to R for Spatial Analysis and Mapping, Chapter 3. Sage: Los Angeles

Preliminaries

It is good practice to clear the working space to make sure that you do not have extraneous items there when you begin your work. The command in `R` to clear the workspace is `rm` (for "remove"), followed by a list of items to be removed. To clear the workspace from *all* objects, do the following:

```r
# The function `ls()` lists all objects in the Environment, that is,
# your current workspace; `rm()` removes all objects listed in the argument `list = `
rm(list = ls())
```

Packages

According to Wickham [-@wickham2015rpackages] packages are the basic units of reproducible code in the `R` multiverse.

Now that your workspace is clear, you can proceed to load a package. In this case, the package is the one used for this book/course, called `geog4ga3`:

```r
# The function 'library' is used to load the data we want to work with.
# In this case, it is the geog4ga3 master package that we want to work with
library(geog4ga3)
```

```
## Warning: replacing previous import 'plotly::filter' by 'stats::filter' when
## loading 'geog4ga3'
```

```
## Warning: replacing previous import 'dplyr::lag' by 'stats::lag' when loading
## 'geog4ga3'
```

The package includes a few datasets that will be used throughout the book:

```r
# The function 'data' is used to check if a dataset is present
# within any loaded packages. In this case, we are looking for 'snow_deaths'
data("snow_deaths")
```

Exploring Dataframes and a Simple Proportional Symbols Map

If you correctly loaded the library, you can now access the dataframes in the package `geog4ga3`. For this section, you will need two dataframes, namely `snow_pumps` and `snow_deaths`:

```r
# The function 'head' will display the first few rows of the dataframe,
# snow_deaths
head(snow_deaths)
```

```
##         long      lat Id Count
## 0 -0.1379301 51.51342  1     3
## 1 -0.1378831 51.51336  2     2
## 2 -0.1378529 51.51332  3     1
## 3 -0.1378120 51.51326  4     1
## 4 -0.1377668 51.51320  5     4
## 5 -0.1375369 51.51318  6     2
```

These data are from the famous London cholera example by John Snow (not the one from Game of Thrones, but the British physician). John Snow is considered the father of spatial epidemiology, and his study mapping the outbreak is credited with helping find its cause.This study investigates the cholera outbreak of Soho, London, in 1854.

The dataframe `snow_deaths` includes the geocoded addresses of cholera deaths in `long` and `lat`, and the number of deaths (the `Count`) recorded at each address, as well as unique identifiers for the addresses (`Id`).

A second dataframe `snow_pumps` includes the geocoded locations of water pumps in Soho:

```r
head(snow_pumps)
```

```
##           long      lat  Id Count
## 01   -0.1366679 51.51334 251     1
```

```
## 1100 -0.1395862 51.51388 252     1
## 250  -0.1396710 51.51491 253     1
## 310  -0.1316299 51.51235 254     1
## 410  -0.1335944 51.51214 255     1
## 510  -0.1359191 51.51154 256     1
```

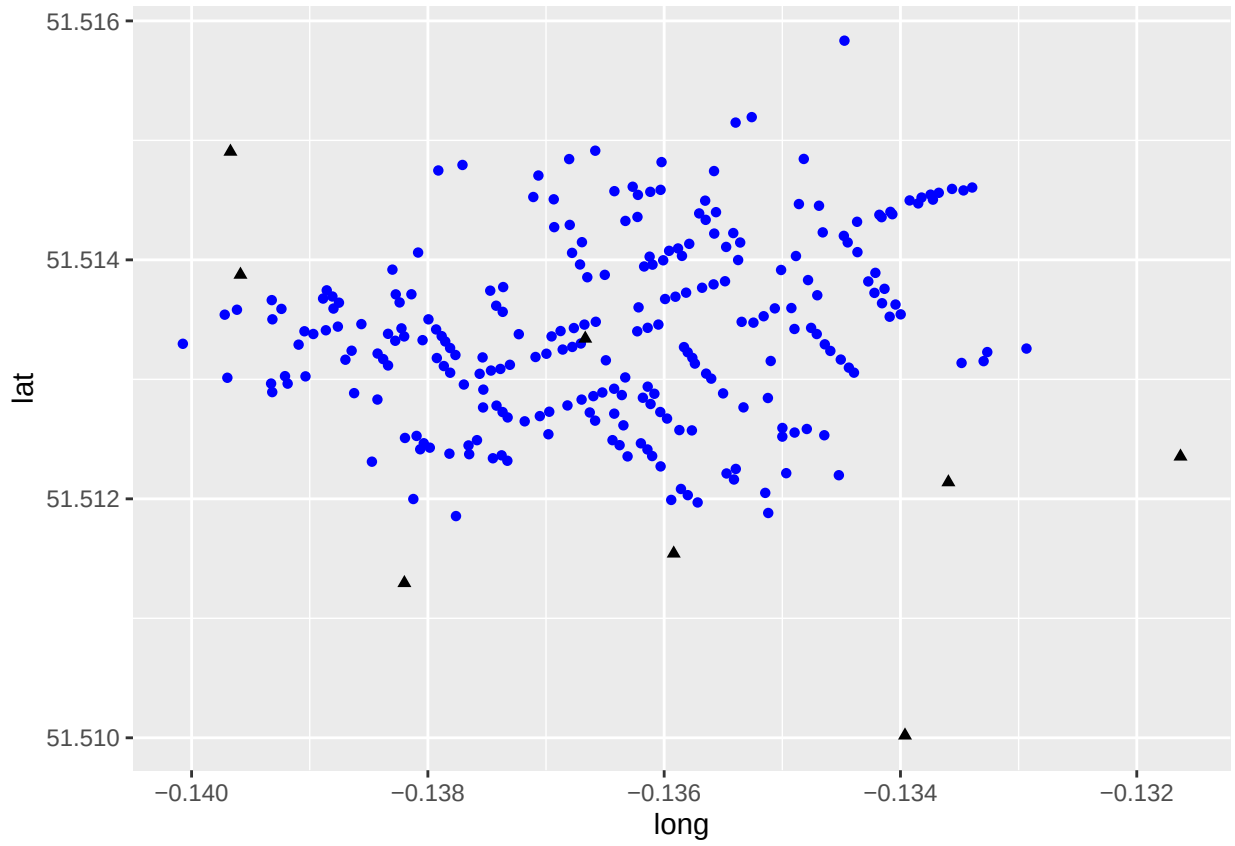
As in your previous reading, it is possible to map the cases using `ggplot2`. Begin by loading the package `tidyverse`:

```
#'Tidyverse' is a collection of R packages designed for data science used in everyday data analyses
library(tidyverse)
```

Now, you can create a blank `ggplot2` object from which you can render the points for deaths and the pumps.

```
# The function 'ggplot' is used for data visualization - it creates a graph.
# The function 'geom_point' tells R you want to create a plot of points.
# 'data = snow_deaths' tells R you want to use the 'snow_deaths' dataframe.
# 'aes' stands for aesthetics of your graph where 'x = long' sets the x axis
# to 'long', where 'y = lat' sets the y axis to 'lat', where 'color = blue'
# colours the points blue and 'shape = 16' assigns the shape of the points
# - in this case, '16' are circles and '17' are triangles

ggplot() +
  geom_point(data = snow_deaths, aes(x = long, y = lat), color = "blue", shape = 16) +
  geom_point(data = snow_pumps, aes(x = long, y = lat), color = "black", shape = 17)
```



This map is a decent example of how to represent visually some contents in the dataframe. Here, information is displayed using different colours and symbols to represent pumps and deaths from the London Cholera

Example. Though this map provides useful insights, it is not of the greatest quality. We will illustrate other ways of creating maps below, including interactive maps.

Improving on the Proportional Symbols Map

A package that extends the functionality of mapping in `R` is `leaflet`. A key feature of the `leaflet` package is the ability to make maps interactive for the user. We will see next how to enhance our proportional symbol map using this package. First you need to load the package (you need to install it first if you have not already):

```
# 'Leaflet' is a package used for visualizing data on a map in R.
# 'Magrittr' is a package used for creating pipe operators
# install.packages('leaflet') # Run only if you have not yet installed `leaflet`!
# install.packages('magrittr') # Run only if you have not yet installed `magrittr`!
library(leaflet)
library(magrittr)
```

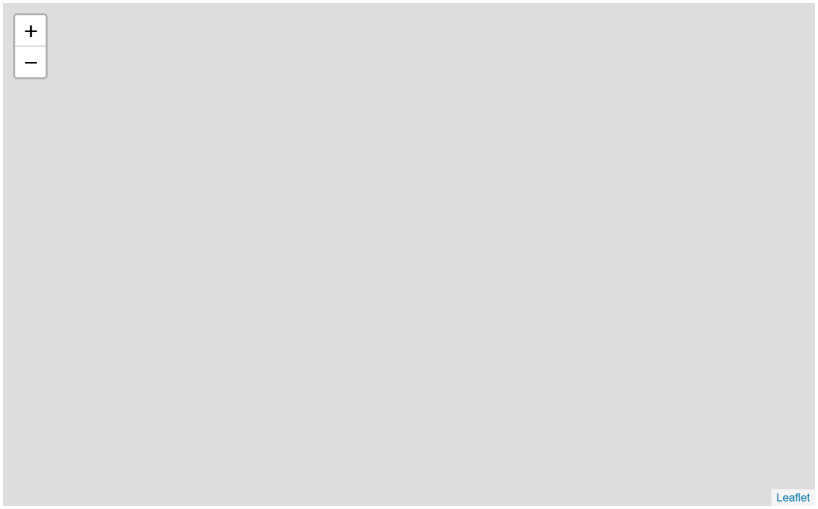
The first step is to create a `leaflet` object, which will be saved in `m` :

```
# Here, we create a `leaflet` object and assign it to the variable, 'm'.
# The 'setView' function sets the view of the map where 'lng = -0.136' sets the longitutde,
# 'lat = 51.513' sets the latitude and the map zoom is set to 16. The '%>%' is a pipe operator
# that passes the output from the left hand side of the operator to the first argument of the
# right hand side of the operator. In this case we are telling `R` that we want to center the
# map on the set longitude and latitude, with a zoom level of 16, which corresponds roughly
# to a neighborhood

m <- leaflet(data = snow_deaths) %>%
  setView(lng = -0.136, lat = 51.513, zoom = 16)
```

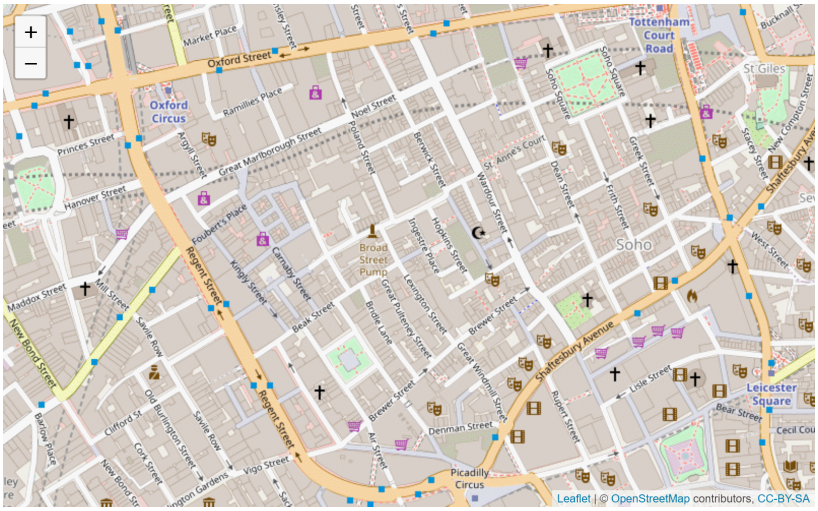
This map looks like this at this point:

```
m
```



The map is empty! This is because we have not yet added any geographical information to plot. We can begin by adding a basemap as follows:

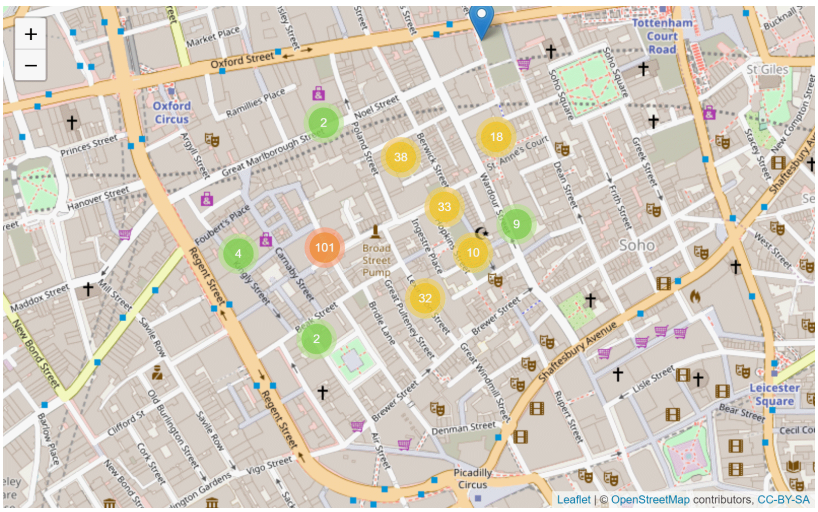
```
# We are adding a basemap or background map of the study location by means of
# the `addTiles` function to the 'm' variable
m <- m %>% addTiles()
m
```



The map now shows the neighborhood in Soho where the cholera outbreak happened. Now, at long last, we can add the cases of cholera deaths to the map. For this, we indicate the coordinates (preceded by `~`), and set an option for clustering by means of the `clusterOptions` in the following fashion:

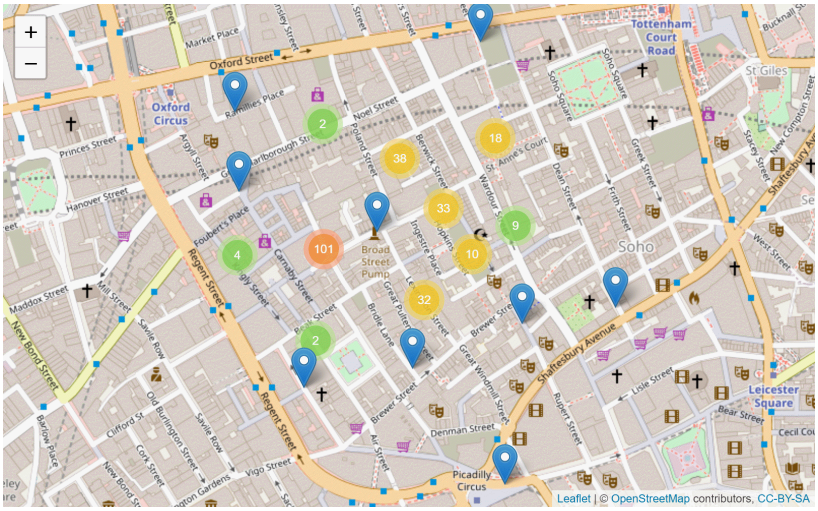
```
# We are adding the cholera deaths to the map using 'group = Deaths'.
# The '~' symbol tells R to use the same longitude and latitude values
# used in the previous block of code and the 'clusterOptions = markerClusterOptions()'
# clusters a large number of markers on the map - in this case it is clusturing
# number of deaths into icons with numbers

m <- m %>% addMarkers(~long, ~lat, clusterOptions = markerClusterOptions(), group = "Deaths")
m
```



The map now displays the locations of cholera deaths on the map. If you zoom in, the clusters will rearrange accordingly. Try it! The other information that we have available is the location of the water pumps, which we can add to the map above (notice that the Broad Street Pump is already shown in the basemap!):

```
m %>% addMarkers(data = snow_pumps, ~long, ~lat, group = "Pumps")
```



An alternative and quicker way to run the same bit of code is by means of pipe operators (%>%). These operators make writing code a lot faster, easier to read, and more intuitive! Recall that a pipe operator will take the output of the preceding function, and pass it on as the first argument of the next:

```
m_test <- leaflet () %>%
  setView(lng = -0.136, lat = 51.513, zoom = 16) %>%
  addTiles() %>%
  addMarkers(data = snow_deaths, ~long, ~lat, clusterOptions = markerClusterOptions(), group = "Deaths"
  addMarkers(data = snow_pumps, ~long, ~lat, group = "Pumps")
m_test
```

The above results in a much nicer map. Is this map informative? What does it tell you about the incidence of cholera and the location of the pumps?

Some Simple Spatial Analysis

Despite the simplicity of this map, we can begin to do some spatial analysis. For instance, we could create a *heatmap*. You have probably seen heatmaps in many different situations before, as they are a popular visualization tool. Heatmaps are created based on a spatial analytical technique called *kernel analysis*. We will cover this technique in more detail later on. For the time being, it can be illustrated by taking advantage of the `leaflet.extras` package, which contains a heatmap function. Load the package as follows:

```
#install.packages("leaflet.extras") # Run only if you have not installed `leaflet.extras` yet!
library(leaflet.extras)
```

Next, create a second leaflet object for this example, and call it `m2`. Notice that we are using the same `setView` parameters:

```
m2 <- leaflet(data = snow_deaths) %>%
  setView(lng = -0.136, lat = 51.513, zoom = 16) %>%
  addTiles()
```

Then, add the heatmap. The function used to do this is `addHeatmap`. We specify the coordinates and the variable for the intensity (i.e., each case in the dataframe is representative of `Count` deaths at the address). Two parameters are important here, the `blur` and the `radius`. If you are working with the R notebook version of the book, experiment changing these parameters:

```
# The 'addHeatmap' function is making a heat map. We specify the coordinates,
# same as the block of code above. The 'intensity' function sets a numeric value,
# the 'blur' specifies the amount of blur to apply and the 'radius' function
# sets the radius of each point on the heatmap
m2 %>% addHeatmap(lng = ~long, lat = ~lat, intensity = ~Count,
                  blur = 40, max = 1, radius = 25)
```

Lastly, you can also add markers for the pumps as follows:

```
m2 %>% addHeatmap(lng = ~long, lat = ~lat, intensity = ~Count,
                  blur = 40, max = 1, radius = 25) %>%
  addMarkers(data = snow_pumps, ~long, ~lat, group = "Pumps")
```

And everything together:

```
m2_test <- leaflet(data = snow_deaths) %>%
  setView(lng = -0.136, lat = 51.513, zoom = 16) %>%
  addTiles() %>%
  addHeatmap(lng = ~long, lat = ~lat,
             intensity = ~Count,
             blur = 40, max = 1, radius = 25) %>%
  addMarkers(data = snow_deaths,
             ~long, ~lat,
             clusterOptions = markerClusterOptions(),
             group = "Deaths") %>%
  addMarkers(data = snow_pumps, ~long, ~lat, group = "Pumps")
m2_test
```

A heatmap (essentially a kernel density of spatial points; more on this in a later chapter) makes it very clear that most cases of cholera happend in the neighborhood of one (possibly contaminated) water pump! At the time, Snow noted with respect to this geographical pattern that:

> "It will be observed that the deaths either very much diminished, or ceased altogether, at every point where it becomes decidedly nearer to send to another pump than to the one in Broad street. It may also be noticed that the deaths are most numerous near to the pump where the water

could be more readily obtained."

Snow's analysis helped to convince officials to close the pump, after which the cholera outbreak subsided. This illustrates how even some relatively simple spatial analysis can help to inform public policy and even save lives. You can read more about this case here.

In this practice you have learned how to implement some simple mapping and spatial statistical analysis using `R`. In future readings we will further explore the potential of `R` for both.

Other Resources

For more information on the functionality of `leaflet`, please check Leaflet for R