Modelling spatial data in `R` with `CARBayes`

Part 1: Introduction and exploratory analysis

Duncan Lee and Eilidh Jack

---

1. Introduction

This is a practical session that will show you how to do exploratory data analysis in `R` for spatial data. Specifically, this session will cover:

- Reading in shapefiles and data, and merging them to create a `SpatialPolygonsDataFrame` object.
- Mapping spatial data.
- Creating the neighbourhood matrix **W**.
- Assessing the significance of the spatial correlation in data via Moran's I statistic.

1.1. Working example

We illustrate this process by means of a worked example, which relates to the spatial pattern in respiratory hospitalisations in the $K = 271$ intermediate zones that comprise the Greater Glasgow and Clyde health board. The data are available from *Scottish Statistics* (http://statistics.gov.scot), and relate to the number of admissions to non-psychiatric and non-obstetric hospitals due to respiratory disease in 2011. Also available are the expected numbers of hospital admissions, which were computed using indirect standardisation. Additionally a number of covariates are also available, and the dataset `Scotland spatial data.csv` contains the following variables.

- `IZ -` The code for each intermediate zone, which is the unique identifier for this dataset.
- `Y -` The observed number of hospital admissions for respiratory disease in 2011 in each IZ.
- `E -` The estimated expected number of hospital admissions for respiratory disease in 2011 in each IZ computed using indirect standardisation.

- `jsa` - The percentage of the working age population in 2011 in each IZ that are in receipt of Job Seekers Allowance (JSA), which is a measure of socio-economic deprivation in an IZ.
- `ethnic` - The percentage of school children who are non-white in 2011 in each IZ, which is a proxy measure of the non-white make up of an IZ.
- `no2` - The estimated average nitrogen dioxide (NO_2) air pollution concentration in each IZ.

In addition to this dataset the shapefile elements are contained in `ScotlandIZ.shp` and `ScotlandIZ.dbf`. Code for this example is stored in Code to do this is in the file `Scotland exploratory analysis.R`.

2. Reading in and combining spatial data

Spatial data come in two main parts.

1. A **dataset** containing the variables to be mapped and modelled. The dataset can be of different types (e.g. .csv, .dat, .txt, etc) but must have a column containing a set of unique identifiers for each area.
2. A **shapefile** containing the polygon boundaries for the set of areas that the data relate to. A shapefile is a collection of many different types (i.e. different file extensions) of files with a common name. In `R` we need the following parts:

- `.shp` contains the polygon information for each area (e.g. the set of vertices that make up each polygon).
- `.dbf` contains a look up table which links the polygons in the `.shp` file with the unique identifier in the dataset.

Therefore the first stage in a spatial data analysis is to read in all three data elements and combine them together. The dataset `Scotland spatial data.csv` is a comma separated variable (.csv) file and thus can be read in using the `R` command `read.csv()` as follows:

```
dat <- read.csv(file="Scotland spatial data.csv")
```

Note, the above command only told `R` the name of the data file and not where it is stored. Thus before this step we need to put all data files in the same directory as the `R` code file we are doing the analysis in, and then set the working directory to the current directory via the `Rstudio Session` menu and then selecting `Set Working Directory` and then `To Source File Location`. Once the data are read in the first few rows of this dataset can be viewed using the `head()` function as follows:

```r
head(dat)
```

```
##          IZ  Y         E   jsa ethnic      no2
## 1 S02000260 90  93.19477 4.600   7.54 16.13495
## 2 S02000261 20  43.78443 1.775   6.27 15.26339
## 3 S02000262 58  92.03014 1.800   9.73 15.26339
## 4 S02000263 43  81.48188 1.200  16.12 17.25486
## 5 S02000264 52 122.64095 2.150   5.83 16.00148
## 6 S02000265 24  56.49576 2.000   7.44 15.42137
```

A summary of each variable can be computed using the `summary()` function as follows:

```r
summary(dat)
```

```
##          IZ            Y                E                jsa
##  S02000260:  1   Min.   : 20.0   Min.   : 42.52   Min.   : 0.700
##  S02000261:  1   1st Qu.: 57.5   1st Qu.: 66.79   1st Qu.: 2.825
##  S02000262:  1   Median : 80.0   Median : 79.91   Median : 4.800
##  S02000263:  1   Mean   : 83.2   Mean   : 83.68   Mean   : 5.107
##  S02000264:  1   3rd Qu.:103.0   3rd Qu.:100.20   3rd Qu.: 7.263
##  S02000265:  1   Max.   :189.0   Max.   :155.86   Max.   :13.100
##  (Other)  :265
##      ethnic            no2
##  Min.   : 0.140   Min.   : 4.543
##  1st Qu.: 2.835   1st Qu.:12.702
##  Median : 5.670   Median :16.531
##  Mean   :11.146   Mean   :16.955
##  3rd Qu.:14.220   3rd Qu.:20.363
##  Max.   :83.720   Max.   :39.379
##
```

The shapefile elements `ScotlandIZ.shp` and `ScotlandIZ.dbf` can be read in using functionality from the `shapefiles` library as follows.

```r
library(shapefiles)
```

```
## Warning: package 'shapefiles' was built under R version 3.3.2

## Loading required package: foreign

##
## Attaching package: 'shapefiles'

## The following objects are masked from 'package:foreign':
##
##     read.dbf, write.dbf
```

```r
shp <- read.shp(shp.name = "ScotlandIZ.shp")
dbf <- read.dbf(dbf.name = "ScotlandIZ.dbf")
```

The final step in this section is to combine the three elements `dat, shp, dbf` together, which can be done using the `combine.data.shapefile()` function from the `CARBayes` package. However, for this to work two things are required.

1. The rownames of the dataset (in this case `dat`) must contain the unique identifier (in this case the variable `IZ`).
2. The first column of the `data.frame` element in the `.dbf` file (in this case `dbf`) must also contain the unique identifier

The first of these can be achieved via the code below, and the `head()` function reveals how the data set has changed.

```
rownames(dat) <- dat$IZ
dat$IZ <- NULL
head(dat)
```

```
##            Y         E   jsa ethnic      no2
## S02000260 90  93.19477 4.600   7.54 16.13495
## S02000261 20  43.78443 1.775   6.27 15.26339
## S02000262 58  92.03014 1.800   9.73 15.26339
## S02000263 43  81.48188 1.200  16.12 17.25486
## S02000264 52 122.64095 2.150   5.83 16.00148
## S02000265 24  56.49576 2.000   7.44 15.42137
```

The rownames have now changed to the IZ codes, and the additional IZ column is no longer required and has thus been removed. Next, we need to check that the first column of the `data.frame` in the `dbf` element contains the IZ codes, and the first few rows can again be viewed using the `head()` function.

```
head(dbf$dbf)
```

```
##   INTGEOCODE                          INTGEONAME
## 1       <NA>                                <NA>
## 2  S02000001                          Cove South
## 3  S02000002     Kincorth, Leggart and Nigg South
## 4  S02000003                              Culter
## 5  S02000004 Cults, Bieldside and Milltimber East
## 6  S02000005                          Cove North
```

From this you can see that the first column does include the IZ codes, which is what is required. Now, the 3 data elements `dat, shp, dbf` can be combined using the following code.

```
library(CARBayes)
```

```
## Warning: package 'CARBayes' was built under R version 3.3.3
```

```
## Loading required package: MASS
```

```
## Loading required package: Rcpp
```

```r
library(sp)
sp.dat <- combine.data.shapefile(data=dat, shp=shp, dbf=dbf)
class(sp.dat)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

The `sp.dat` object is a `spatialPolygonsDataFrame` object (from the `sp` package), and contains both the dataset and the spatial polygon information. The dataset can be accessed in this spatial object via the `data` element, that is `sp.dat@data`. Thus we can view the first few lines of this element via the `head()` function as usual.

```r
head(sp.dat@data)
```

```
##            Y         E   jsa ethnic      no2
## S02000260 90  93.19477 4.600   7.54 16.13495
## S02000261 20  43.78443 1.775   6.27 15.26339
## S02000262 58  92.03014 1.800   9.73 15.26339
## S02000263 43  81.48188 1.200  16.12 17.25486
## S02000264 52 122.64095 2.150   5.83 16.00148
## S02000265 24  56.49576 2.000   7.44 15.42137
```

The spatial regions can be observed via the `plot()` function.

```r
plot(sp.dat)
```

It is easy to add variables to `sp.dat`, so for example, the standardised morbidity ratio (SMR = observed admissions / expected admissions) can be computed and stored in the dataset with the name `smr` using the following code:

```
sp.dat@data$smr  <- sp.dat@data$Y / sp.dat@data$E
```

We now show how to assess the presence of spatial correlation in the SMR and draw a map of it.

3. Creating the neighbourhood matrix W and computing Moran's I statistic.

A binary neighbourhood matrix $\mathbf{W}$ based on sharing a common border can be constructed using functionality from the `spdep` package. This is done in two steps, the first creates a neighbourhood (`nb`) object from the `spatialPolygonsDataFrame` object `sp.dat`, and the second creates a neighbourhood matrix from the `nb` object. Code to achieve this is below.

```
library(spdep)
```

```
## Loading required package: Matrix
```

```
W.nb <- poly2nb(sp.dat, row.names = rownames(sp.dat@data))
W <- nb2mat(W.nb, style = "B")
class(W)
```

```
## [1] "matrix"
```

```
dim(W)
```

```
## [1] 271 271
```

As you can see from the results above the element `W` is a 271×271 matrix, which is because there are $K = 271$ IZ in the Greater Glasgow and Clyde health board study region. This code has created a binary neighbourhood matrix based on sharing a common border, but other options for creating $\mathbf{W}$, such as being one of the k nearest neighbours, can be created using the `knearneigh()` function. For more details on creating different types of neighbourhood matrices see Bivand, Pebesma, and Gomez-Rubio (2013).

Moran's I statistic for measuring the amount of spatial correlation can be calculated using functionality from the `spdep` package. Specifically, the statistic is computed and a hypothesis test (where the null hypothesis H_0 is independence) conducted using the `moran.mc()` function. This function conducts the Monte Carlo permutation test and it is computed for the SMR variable using the code below.

```
W.list <- nb2listw(W.nb, style = "B")
moran.mc(x = sp.dat@data$smr, listw = W.list, nsim = 10000)
```

```
##
##  Monte-Carlo simulation of Moran I
##
## data:  sp.dat@data$smr
## weights: W.list
## number of simulations + 1: 10001
##
## statistic = 0.41831, observed rank = 10001, p-value = 9.999e-05
## alternative hypothesis: greater
```

The first line of the above code creates a spatial list (`listw`) object, and the second line computes Moran's I and conducts the hypothesis test. In the code above the p-value for the hypothesis test is based on 10,000 random permutations (`nsim`), which should be large enough to give a reasonable p-value. The output of the `moran.mc()` function includes the Moran's I statistic (`statistic`), how extreme the observed value of Moran's I was compared to the values computed for the 10,000 random permutations (`observed rank`), and the p-value against the null hypothesis of independence (`p-value`). In this example Moran's I statistic equals 0.41831 and is significantly different from independence, which provides evidence for spatial correlation in the `smr` variable.

4. Mapping spatial data

Once the data have been read into `R` the natural first step is to draw a map, and in this example the SMR is the natural variable to visualise. `R` has a number of different packages for drawing maps, including `sp` and `ggplot2`, and we illustrate the `ggplot2` package here. First load the required packages using the code:

```
library(ggplot2)
library(rgeos)
```

```
## Warning: package 'rgeos' was built under R version 3.3.2

## rgeos version: 0.3-22, (SVN revision 544)
##  GEOS runtime version: 3.5.0-CAPI-1.9.0 r4084
##  Linking to sp version: 1.2-4
##  Polygon checking: TRUE
```

```
library(maptools)
```

```
## Warning: package 'maptools' was built under R version 3.3.2

## Checking rgeos availability: TRUE
```

Before you can draw a map, ggplot2 requires you to turn the sp.dat SpatialPolygonsDataFrame object into a data.frame. This can be done using the following code:

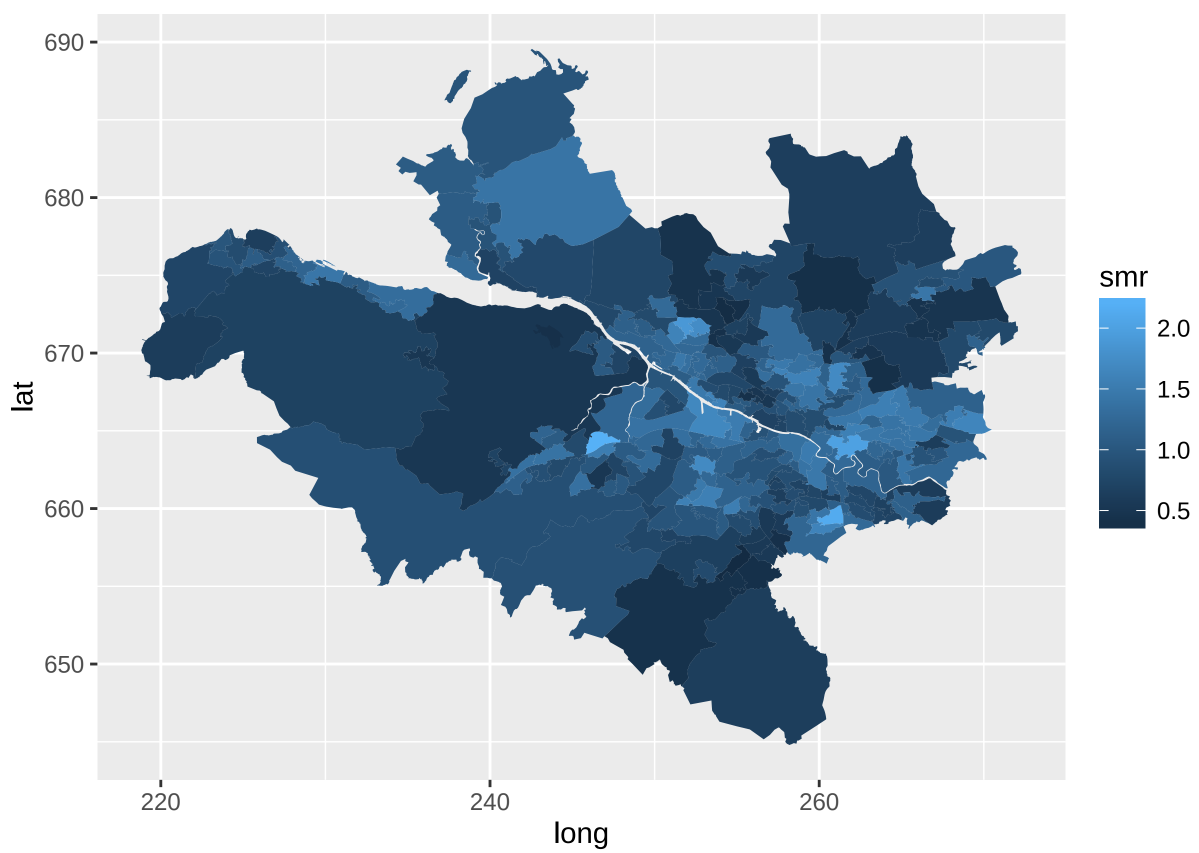
```
sp.dat@data$id <- rownames(sp.dat@data)
temp1 <- fortify(sp.dat, region = "id")
sp.dat2 <- merge(temp1, sp.dat@data, by = "id")
```

These lines transform the sp.dat SpatialPolygonsDataFrame object into a data.frame called sp.dat2. Next we transform the spatial scale from metres to Kilometres by dividing the spatial scales by 1000 using the code below.

```
sp.dat2$long <- sp.dat2$long / 1000
sp.dat2$lat <- sp.dat2$lat / 1000
```

Then a basic map of the SMR can be created using the following code:

```
ggplot(data = sp.dat2, aes(x=long, y=lat, goup=group, fill = smr)) +
    geom_polygon()
```

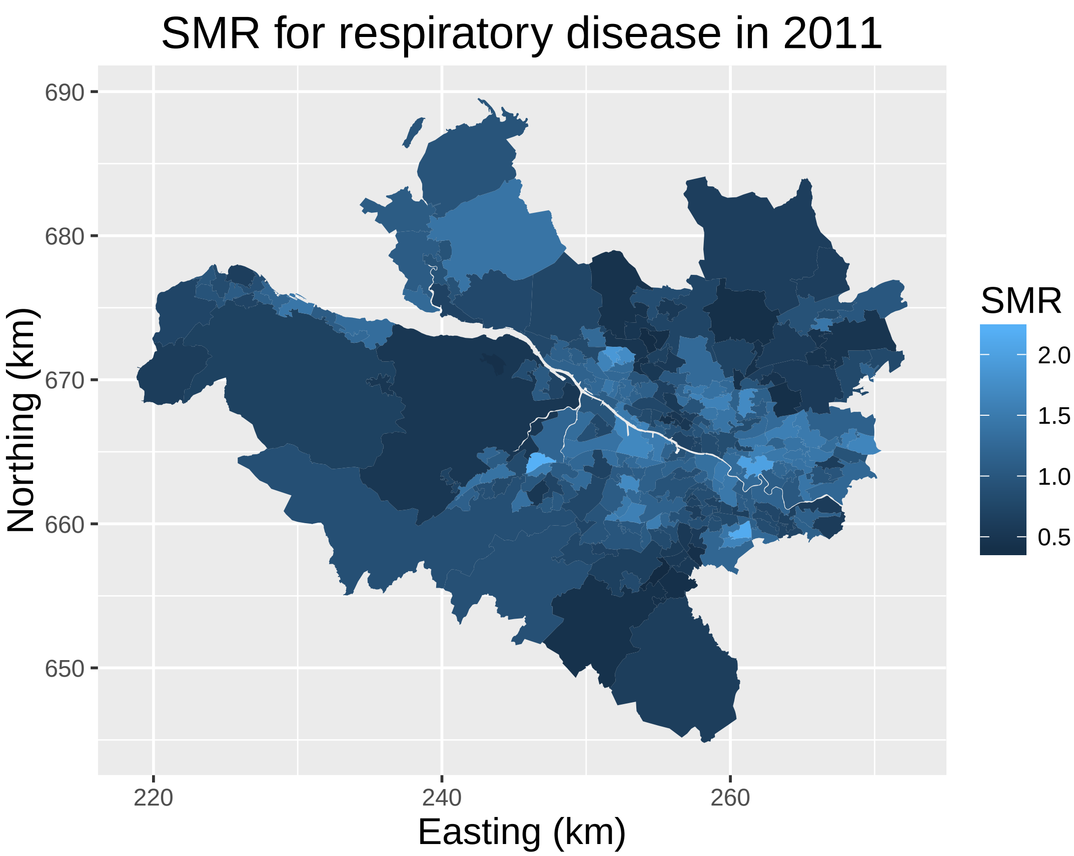


Here:

- ggplot() - specifies the data frame, the two variables (columns) to be used to create the plotting area, and the variable to be mapped.
- geom_poylgon() - adds the shaded areal units to the plot.

However, this map is unsatisfactory in a number of ways, and can be improved by adding additional commands to the `ggplot()` function separated by the `+` sign as shown below.

```
ggplot(data = sp.dat2, aes(x=long, y=lat, goup=group, fill = smr)) +
    geom_polygon() +
    coord_equal() +
    xlab("Easting (km)") +
    ylab("Northing (km)") +
    labs(title = "SMR for respiratory disease in 2011", fill = "SMR") +
    theme(title = element_text(size=14))
```

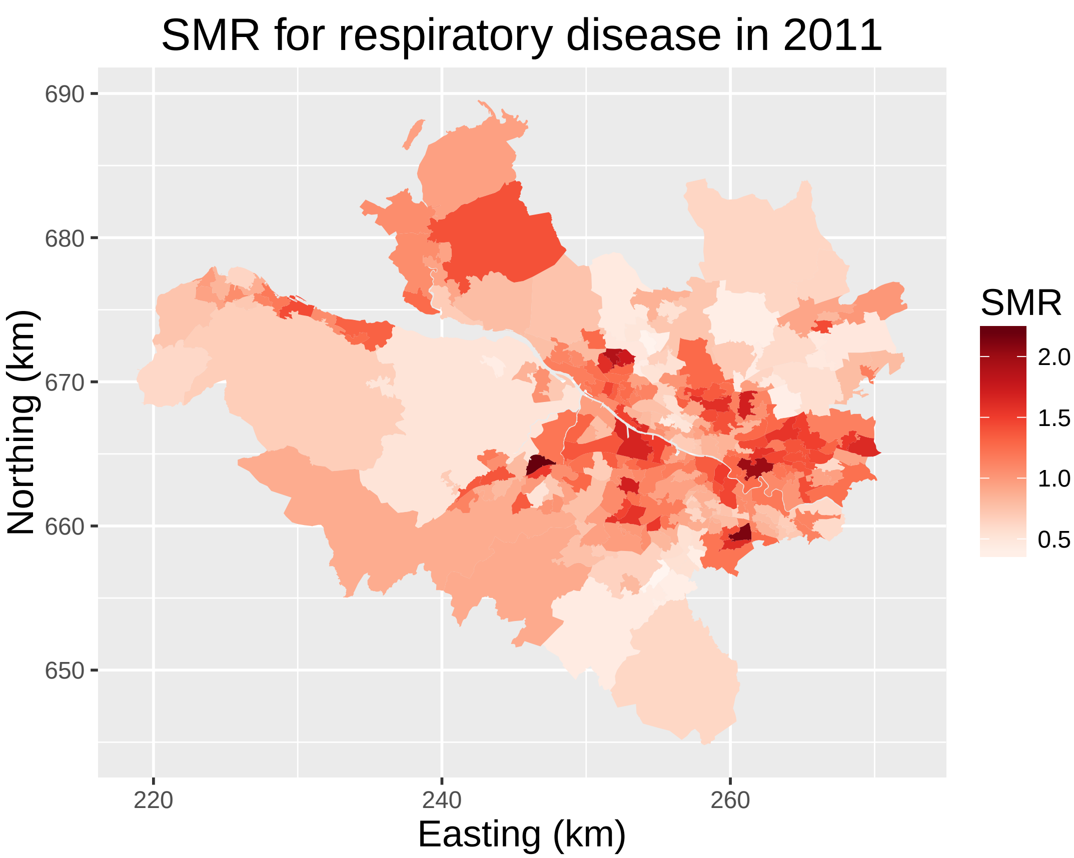


Here the new lines make the following changes:

- `coord_equal()` - makes sure that 1 unit in the x axis direction is the same size as 1 unit in the y axis direction.
- `xlab()` - specify the horizontal axis label for the plot.
- `ylab()` - specify the vertical axis label for the plot.
- `labs()` - add titles to the plot and to the colour key.
- `theme()` - change the typeface and size of the text on the plot.

There are lots of different ways to change the colour scale on the map, and one such way is through the `scale_fill_gradientn()` function. An example is shown below.

```r
library(RColorBrewer)
ggplot(data = sp.dat2, aes(x=long, y=lat, goup=group, fill = smr)) +
    geom_polygon() +
    coord_equal() +
    xlab("Easting (km)") +
    ylab("Northing (km)") +
    labs(title = "SMR for respiratory disease in 2011", fill = "SMR") +
    theme(title = element_text(size=14)) +
    scale_fill_gradientn(colors=brewer.pal(n=9, name="Reds"))
```



Here, the `colors` argument in the `scale_fill_gradientn()` function is specified by a colour palette from the `RColorBrewer` library, where the `n` argument specifies how many colour shades are included (here `n=9`). Possible colour schemes are available from http://colorbrewer2.org/.

5. Overlaying a Google map

Finally, we illustrate how to overlay the previous map on a Google map, so that the places within the study region can be observed. The first challenge in doing this is that Google maps can only be downloaded in longitude and latitude coordinates in degrees, where as the coordinate system for the data is in easting and northing in metres. Thus we first create a copy of the spatial data object `sp.dat` (which we call `sp.dat3`) and change its coordinates from metres to degrees. This is done using the following code:

```
library(rgdal)
```

```
## Warning: package 'rgdal' was built under R version 3.3.2

## rgdal: version: 1.2-5, (SVN revision 648)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.0.1, released 2015/09/15
##  Path to GDAL shared files: C:/Users/Duncan Lee/Documents/R/win-library/3.3/rgdal/gda
##  Loaded PROJ.4 runtime: Rel. 4.9.2, 08 September 2015, [PJ_VERSION: 492]
##  Path to PROJ.4 shared files: C:/Users/Duncan Lee/Documents/R/win-library/3.3/rgdal/p
##  Linking to sp version: 1.2-4
```

```
sp.dat3 <- sp.dat
proj4string(sp.dat3) <- CRS("+init=epsg:27700")
sp.dat3 <- spTransform(sp.dat3, CRS("+init=epsg:4326"))
```

The original data set did not have any coordinate system specified, so this is first assigned to be in metres (3rd line), and then transformed to be in degrees (4th line). Once the coordinate system is consistent with that used by Google maps, the `spatialPolygonsDataFrame` object `sp.dat3` is transformed to a `data.frame` object using the code below (which is similar to code presented earlier).

```
sp.dat3@data$id <- rownames(sp.dat3@data)
temp1 <- fortify(sp.dat3, region = "id")
sp.dat4 <- merge(temp1, sp.dat3@data, by = "id")
```

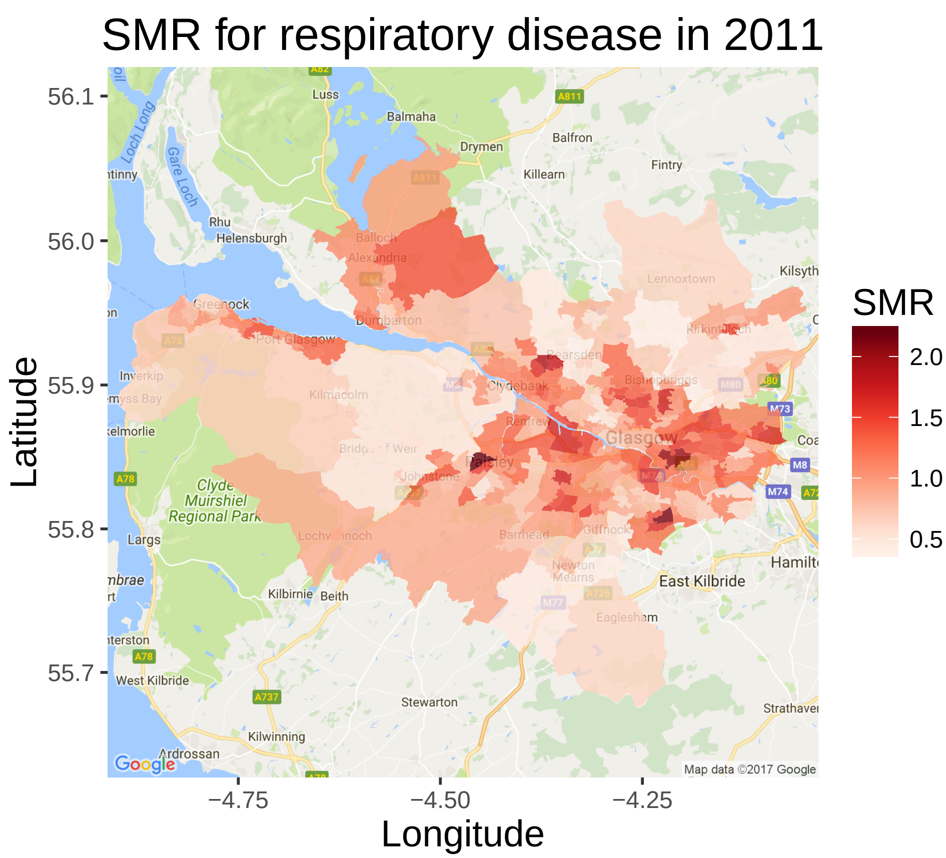
Then the mapping is conducted in two stages, first downloading a Google map, and then second overlaying it on the data map. Downloading the Google map is achieved via the following code using the `ggmap` package:

```
library(ggmap)
extent <- bbox(sp.dat3)
centre <- apply(extent, 1,mean)
myMap <- get_map(location=centre, maptype="roadmap", zoom=10)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=55.874374,-4.4728
```

Lines 2 and 3 define the centre of the map, while line 4 downloads the map. The `zoom` argument takes in an integer between 3 and 21, which defines the scale of the map (the smaller the number the bigger the area the map covers). Then the map is overlaid on the previous map via the following code.

```r
ggmap(myMap) +
    geom_polygon(data=sp.dat4, aes(x=long, y=lat, group=group, fill=smr),
    alpha=0.8) +
    xlab("Longitude") +
    ylab("Latitude") +
    labs(title = "SMR for respiratory disease in 2011", fill = "SMR") +
    theme(title = element_text(size=14)) +
    scale_fill_gradientn(colors=brewer.pal(n=9, name="Reds"))
```



References

Bivand, R, E Pebesma, and V Gomez-Rubio. 2013. *Applied Spatial Data Analysis with R*. 2nd ed. Springer-Verlag, New York.