# Session 1 Practical: Exploratory spatial analysis in R

*Duncan Lee*

---

## 1. Introduction and aim of the session

This is a practical session that will show you how to do exploratory data analysis in R for spatial data.

**Aims of the session**

By the end of this session you should be able to:

- Read in shapefiles and data, and merge them to create a `SpatialPolygonsDataFrame` object.
- Create the neighbourhood matrix **W**.
- Assess the significance of the spatial correlation in data via Moran's I statistic.
- Map spatial data.

**Motivating example**

We consider part of the data analysed by Kavanagh, Lee, and Pryce (2016), which focused on the decentralisation of poverty in the four largest Scottish Cities, namely: Aberdeen, Dundee, Edinburgh, and Glasgow. Data were collected for the 2011 census, and are available at the datazone (DZ) resolution. Datazones are the smallest spatial units at which data about populations are released in Scotland, and between 500 and 1,000 people live in each datazone. These datazones lie within large administrative units called Health Boards, of which there are 14 in Scotland. The data are stored in `Scotland data.csv`, and contain the following columns.

- `DZ` - A unique code identifying each datazone.
- `HB` - The health board that the datazone lies within.
- `Easting` - The east/west coordinate (in metres) of the central point of the datazone.
- `Northing` - The north/south coordinate (in metres) of the central point of the datazone.

- `urban` - A six-level categorical variable denoting how urban or rural each datazone is. Here a 1 denotes urban while a 6 denotes rural, see http://www.gov.scot/Topics/Statistics/About/Methodology/UrbanRuralClassification.
- `JSA2011` - The number of people of working age claiming Job Seekers Allowance (JSA) in each datazone in 2011. JSA is a benefit paid to people with no job, hence is a measure of poverty.
- `workpop2011` - The total number of people of working age in each datazone in 2011.

# 2. Reading in and combining spatial data

## 2.1. Data formats

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. In the example here our data set is stored in `Scotland data.csv` and is described above.

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.

The shapefile elements for the datazones in the above motivating example are stored in `datazone.shp` and `datazone.dbf`.

Therefore the first stage in a spatial data analysis is to read in all three data elements and combine them together.

## 2.2 Reading in the data

First, ensure you have saved all data sets and shapefiles in the same location as this code file. Then set the working directory to the source file location via the `RStudio Session` menu, and then selecting `Set Working Directory`, and then `To Source File Location`. Then the `Scotland data.csv` data set can be read in and the first few rows viewed using the following code:

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

```
##          DZ       HB  Easting Northing urban JSA2011 workpop2011
## 1 S01000001 Grampian 382990.5 799562.2     3       5         616
## 2 S01000002 Grampian 394819.9 800451.8     1       3         433
## 3 S01000003 Grampian 394848.6 800789.9     1       5         615
## 4 S01000004 Grampian 394354.2 800805.5     1       5         438
## 5 S01000005 Grampian 383550.2 800851.8     3       5         427
## 6 S01000006 Grampian 384158.3 800891.2     3       5         341
```

The data set we analyse here is just the city of Glasgow. The city is set within the Greater Glasgow and Clyde health board, and is clearly urban, so we create this smaller data set by sub-setting the `dat data.frame` object based on 2 criteria:

- Each datazone must be in the Greater Glasgow and Clyde health board (`HB` column).
- Each datazone's `urban` indicator variable must equal 1.

To create this new sub-setted data set we use the following code:

```
gla <- dat[dat$HB=="Greater Glasgow & Clyde" & dat$urban==1, ]
```

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

```
library(shapefiles)
```

```
## Loading required package: foreign
```

```
##
## Attaching package: 'shapefiles'
```

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

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

The final step in this section is to combine the three elements `gla, 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 `gla`) must contain the unique identifier (in this case the variable `DZ`).
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(gla) <- gla$DZ
gla$DZ <- NULL
head(gla)
```

```
##                                HB  Easting Northing urban JSA2011
## S01001423 Greater Glasgow & Clyde 260510.2 669498.5     1      20
## S01001424 Greater Glasgow & Clyde 261587.4 669524.5     1      50
## S01001425 Greater Glasgow & Clyde 262480.9 669567.6     1      43
## S01001426 Greater Glasgow & Clyde 261050.1 669614.5     1      13
## S01001427 Greater Glasgow & Clyde 262244.8 669793.7     1      15
## S01001428 Greater Glasgow & Clyde 261720.9 669836.4     1      30
##           workpop2011
## S01001423         539
## S01001424         650
## S01001425        1006
## S01001426         429
## S01001427         537
## S01001428         569
```

The rownames have now changed to the datazone (DZ) codes, and the additional DZ 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 DZ codes, and the first few rows can again be viewed using the head() function.

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

```
##     DZ_CODE DZ_NAME DZ_GAELIC STDAREA_HA Shape_Leng Shape_Area
## 1 S01000001    <NA>      <NA>  446.59357  12834.265  4465935.9
## 2 S01000002    <NA>      <NA>   30.89500   2765.763   308949.9
## 3 S01000003    <NA>      <NA>   14.03399   2339.842   133045.8
## 4 S01000004    <NA>      <NA>   19.97742   2490.040   199774.2
## 5 S01000005    <NA>      <NA>   16.01258   3248.682   158273.8
## 6 S01000006    <NA>      <NA>   18.04853   3049.431   180485.3
```

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

```
library(CARBayes)
```

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

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

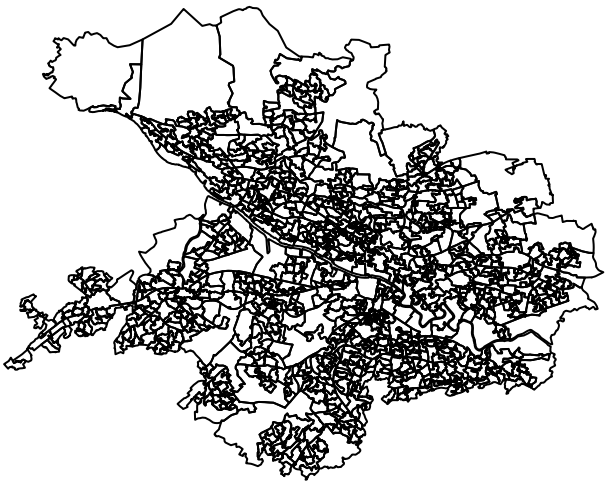
```
library(sp)
sp.gla <- combine.data.shapefile(data=gla, shp=shp, dbf=dbf)
class(sp.gla)
```

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

The sp.gla object is a spatialPolygonsDataFrame object (from the sp package), and

contains both the dataset and the spatial polygon information. The spatial layout of the areas can be viewed by using the `plot()` function on `sp.gla` as follows.

```
plot(sp.gla)
```



**Tip -** The set of data points in the dataset can relate to a subset or all of the areas in the shapefile. In this example the shapefile elements `(shp, dbf)` contain the polygon information for all datazones in Scotland, where as the dataset `gla` only contains data on the datazones in Glasgow. Thus the combined spatial data object `sp.gla` only contains the set of datazones in Glasgow.

The dataset can be accessed in this spatial object via the data element, that is `sp.gla@data`. Thus we can view the first few lines of this element via the `head()` function as usual.

```
head(sp.gla@data)
```

```
##                                HB  Easting Northing urban JSA2011
## S01001423 Greater Glasgow & Clyde 260510.2 669498.5     1      20
## S01001424 Greater Glasgow & Clyde 261587.4 669524.5     1      50
## S01001425 Greater Glasgow & Clyde 262480.9 669567.6     1      43
## S01001426 Greater Glasgow & Clyde 261050.1 669614.5     1      13
## S01001427 Greater Glasgow & Clyde 262244.8 669793.7     1      15
## S01001428 Greater Glasgow & Clyde 261720.9 669836.4     1      30
##           workpop2011
## S01001423         539
## S01001424         650
## S01001425        1006
## S01001426         429
## S01001427         537
## S01001428         569
```

It is easy to add variables to `sp.gla`, and in this example we wish to measure poverty in a datazone by the proportion of the working age population that are in receipt of JSA in 2011. This can be done using the code below.

```
sp.gla@data$propJSA2011 <- sp.gla@data$JSA2011 / sp.gla@data$workpop2011
```

# 3. Creating the neighbourhood matrix W

A binary neighbourhood matrix **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.gla`, 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.gla, row.names = rownames(sp.gla@data))
W <- nb2mat(W.nb, style = "B")
class(W)
```

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

```
dim(W)
```

```
## [1] 1161 1161
```

As you can see from the results above the element `W` is a $1161 \times 1161$ matrix, which is because there are 1161 datazones in the Glasgow study region. This code has created a binary neighbourhood matrix based on sharing a common border, but other options for creating **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).

# 4. Calculating Moran's I statistic

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 described earlier, and it is computed for the JSA proportion in 2011 variable using the code below.

```
W.list <- nb2listw(W.nb, style = "B")
moran.mc(x = sp.gla@data$propJSA2011, listw = W.list, nsim = 10000)
```

```
##
##  Monte-Carlo simulation of Moran I
##
```

```
## data:  sp.gla@data$propJSA2011
## weights: W.list
## number of simulations + 1: 10001
##
## statistic = 0.45265, 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 is 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 0.45265 and is significantly different from independence, which provides evidence of spatial correlation in the `propJSA2011` variable.

# 5. Mapping spatial data using ggplot2()

Once the data have been read into `R` the natural first step is to draw a map, and in this example the proportion of people claiming JSA in 2011 is a natural variable to visualise. `R` has a number of different packages for drawing maps, including `sp` and `ggplot2`, and here we illustrate the `ggplot2` package. However before you can draw a map, `ggplot2` requires you to turn the `sp.gla` `SpatialPolygonsDataFrame` object into a `data.frame`. This can be done using the following code:

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

```
## rgeos version: 0.3-23, (SVN revision 546)
##  GEOS runtime version: 3.6.1-CAPI-1.10.1 r0
##  Linking to sp version: 1.2-4
##  Polygon checking: TRUE
```

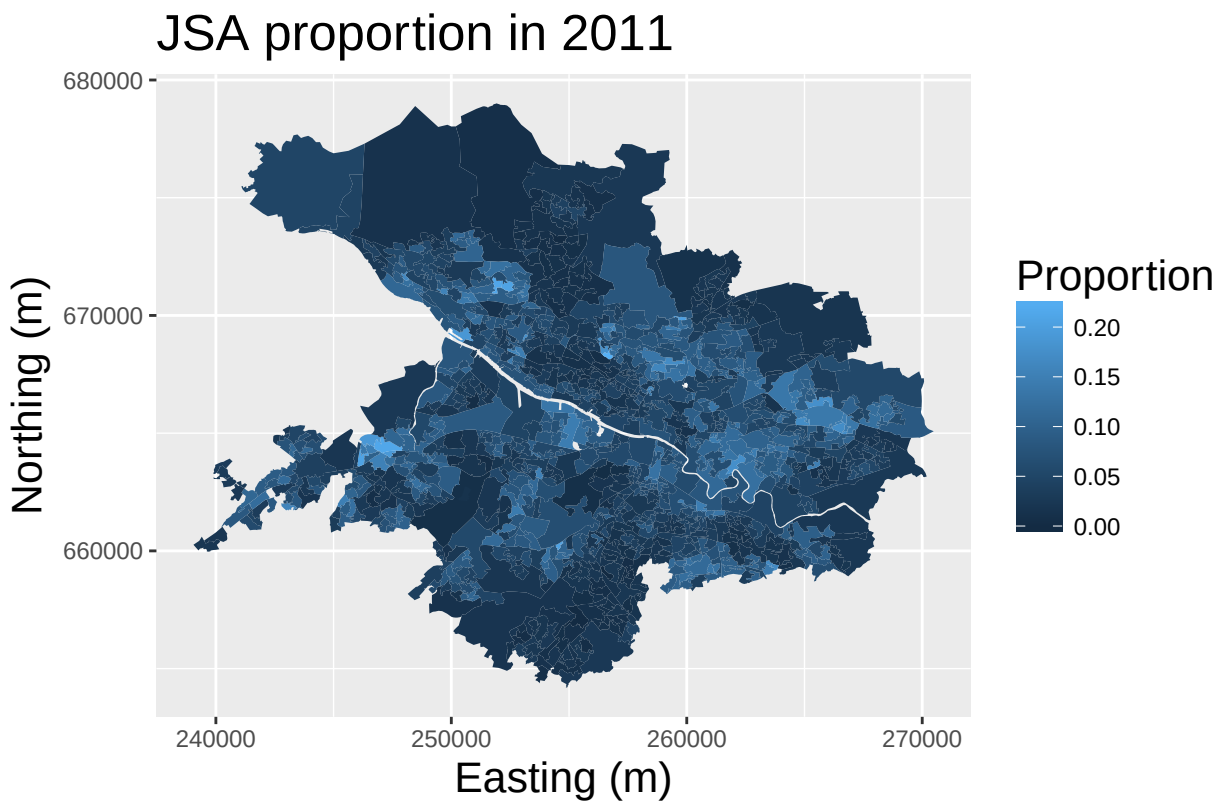
```
library(maptools)
```

```
## Checking rgeos availability: TRUE
```

```
sp.gla@data$id <- rownames(sp.gla@data)
temp1 <- fortify(sp.gla, region = "id")
sp.gla2 <- merge(temp1, sp.gla@data, by = "id")
```

The first three lines load libraries that you need, while the next three transform the `sp.gla` `SpatialPolygonsDataFrame` object into a `data.frame` called `sp.gla2`. Then a map of the proportion of people claiming JSA in 2011 can be created using the following code:

```
ggplot(data = sp.gla2, aes(x=long, y=lat, goup=group, fill = c(propJSA2011))) +
    geom_polygon() +
    coord_equal() +
    xlab("Easting (m)") +
    ylab("Northing (m)") +
    labs(title = "JSA proportion in 2011", fill = "Proportion") +
    theme(title = element_text(size=16))
```



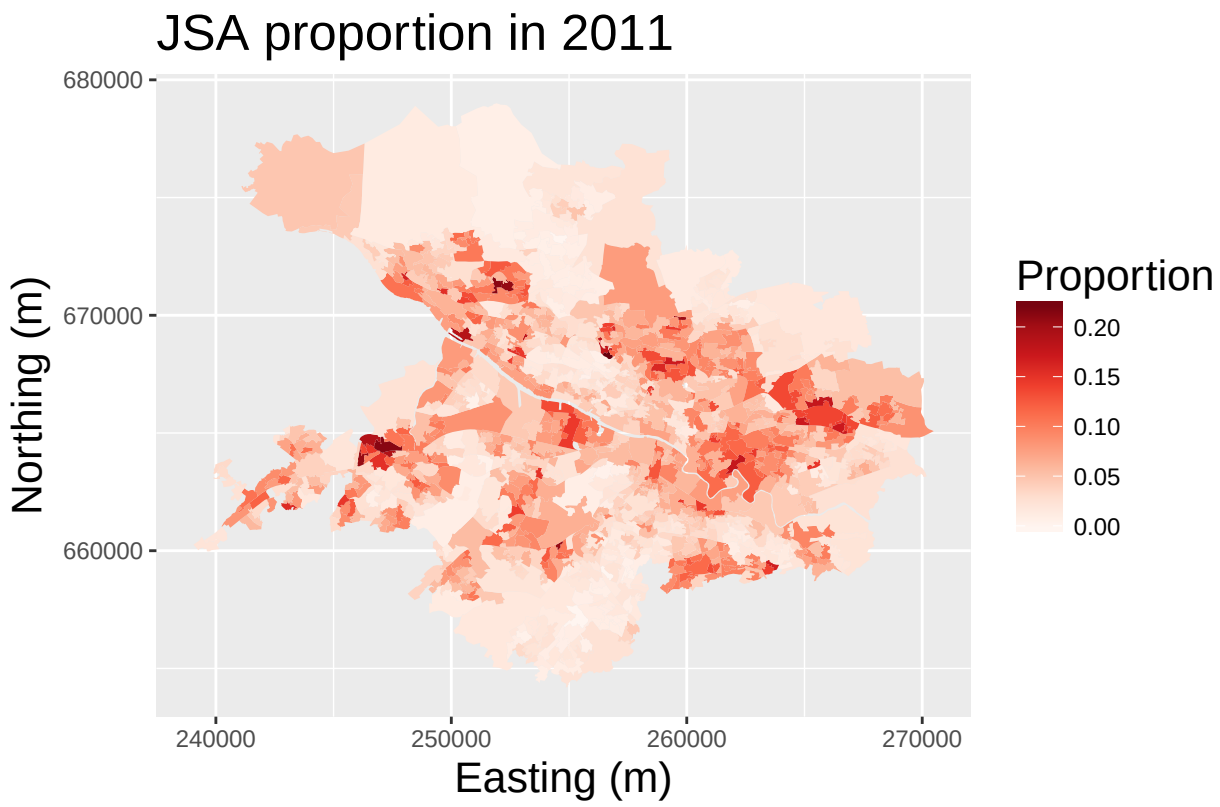

When using the `ggplot2` package each line adds something to the plot as discussed below.

- `ggplot()` - specifies the data frame and the two variables (columns) to be used to create the plotting area.
- `geom_poylgon()` - adds the shaded areal units to the plot.
- `coord_equal()` - makes sure that a 1 unit in the x axis direction is the same as a 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 a 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.

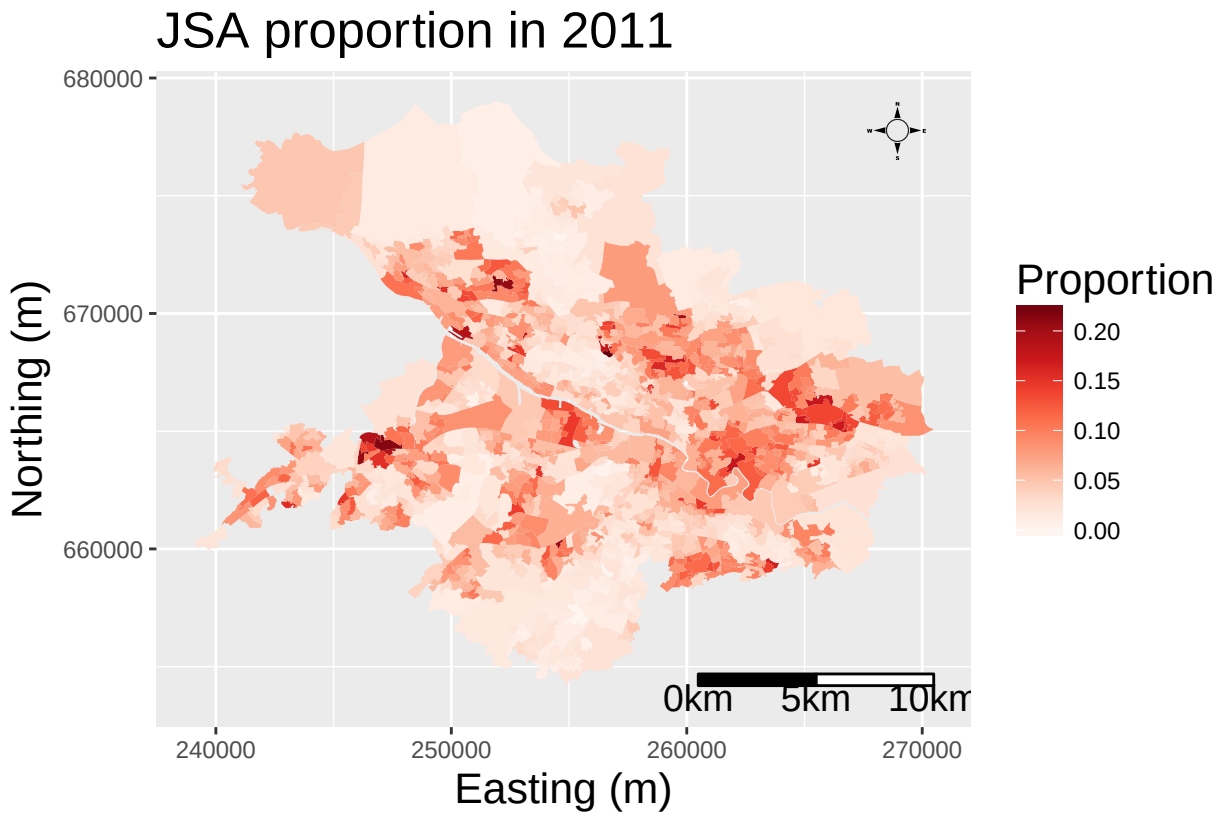
```
library(RColorBrewer)
ggplot(data = sp.gla2, aes(x=long, y=lat, goup=group, fill = c(propJSA2011))) +
```

```
    geom_polygon() +
    coord_equal() +
    xlab("Easting (m)") +
    ylab("Northing (m)") +
    labs(title = "JSA proportion in 2011", fill = "Proportion") +
    theme(title = element_text(size=16)) +
    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. Possible colour schemes are available from http://colorbrewer2.org/. Next, adding a scalebar and a north arrow is straightforward using the `north()` and `scalebar` functions from the `ggsn` package using the code below:

```
library(ggsn)
ggplot(data = sp.gla2, aes(x=long, y=lat, goup=group, fill = c(propJSA2011))) +
    geom_polygon() +
    coord_equal() +
    xlab("Easting (m)") +
    ylab("Northing (m)") +
    labs(title = "JSA proportion in 2011", fill = "Proportion") +
    theme(title = element_text(size=16)) +
    scale_fill_gradientn(colors=brewer.pal(n=9, name="Reds")) +
    north(sp.gla2, location="topright", symbol=16) +
    scalebar(sp.gla2, dist=5)
```




## 6. Overlaying spatial data on a Google map

Finally in terms of mapping data, we illustrate briefly how to overlay your data on a Google map. We note that this is a relatively new feature, and is currently not all that flexible. The first problem to overcome is that the spatial coordinates of the data are (easting, northing) in metres, where as Google maps can only be downloaded in (longitude, latitude) in degrees. To overcome this we create a new `spatialPolygonsDataFrame` object and change its coordinates to the (longitude, latitude) system using the code below.

```
library(rgdal)
```

```
## rgdal: version: 1.2-7, (SVN revision 660)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.1.3, released 2017/20/01
##  Path to GDAL shared files: /Library/Frameworks/R.framework/Versions/3.4/Resources/li
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: /Library/Frameworks/R.framework/Versions/3.4/Resources/
##  Linking to sp version: 1.2-4
```

```
sp.gla3 <- sp.gla
proj4string(sp.gla3) <- CRS("+init=epsg:27700")
```

```
sp.gla3 <- spTransform(sp.gla3, CRS("+init=epsg:4326"))
```

The package required to do this is `rgdal`, and the new `spatialPolygonsDataFrame` object `sp.gla3` is first assigned its current coordinate system using the `proj4string()` function. Then the coordinate system is changed to (longitude, latitude) using the `spTransform()` function. Next, this new `spatialPolygonsDataFrame` object `sp.gla3` is transformed into a `data.frame` object `sp.gla4` using the code below which is similar to that outlined above.

```
sp.gla3@data$id <- rownames(sp.gla3@data)
temp1 <- fortify(sp.gla3, region = "id")
sp.gla4 <- merge(temp1, sp.gla3@data, by = "id")
```

Next we load the `ggmap` library, and obtain a Google map using the following code.

```
library(ggmap)
extent <- bbox(sp.gla3)
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=55.870183,-4.3203
```

Here, the `extent` object is the geographical extent of the data, while `centre` is the middle point of data and is needed by the `get_map()` function. Finally, the `zoom` argument specifies the size of the region in the map, and is an integer from 3 (continent) to 21 (building). You will need to try different zoom levels before you decide which is right for you. Sadly, you cannot get zoom levels between those specified by integers. Then you can draw the final map using the code below.

```
ggmap(myMap) +
    geom_polygon(data=sp.gla4, aes(x=long, y=lat, group=group, fill=c(propJSA2011)),
                 alpha=0.8) +
    xlab("Longitude") +
    ylab("Latitude") +
    labs(title = "JSA proportion in 2011", fill = "Proportion") +
    theme(title = element_text(size=16)) +
    scale_fill_gradientn(colors=brewer.pal(n=9, name="Blues"))  +
    scale_x_continuous(limits = extent[1, ]) +
    scale_y_continuous(limits = extent[2, ])
```

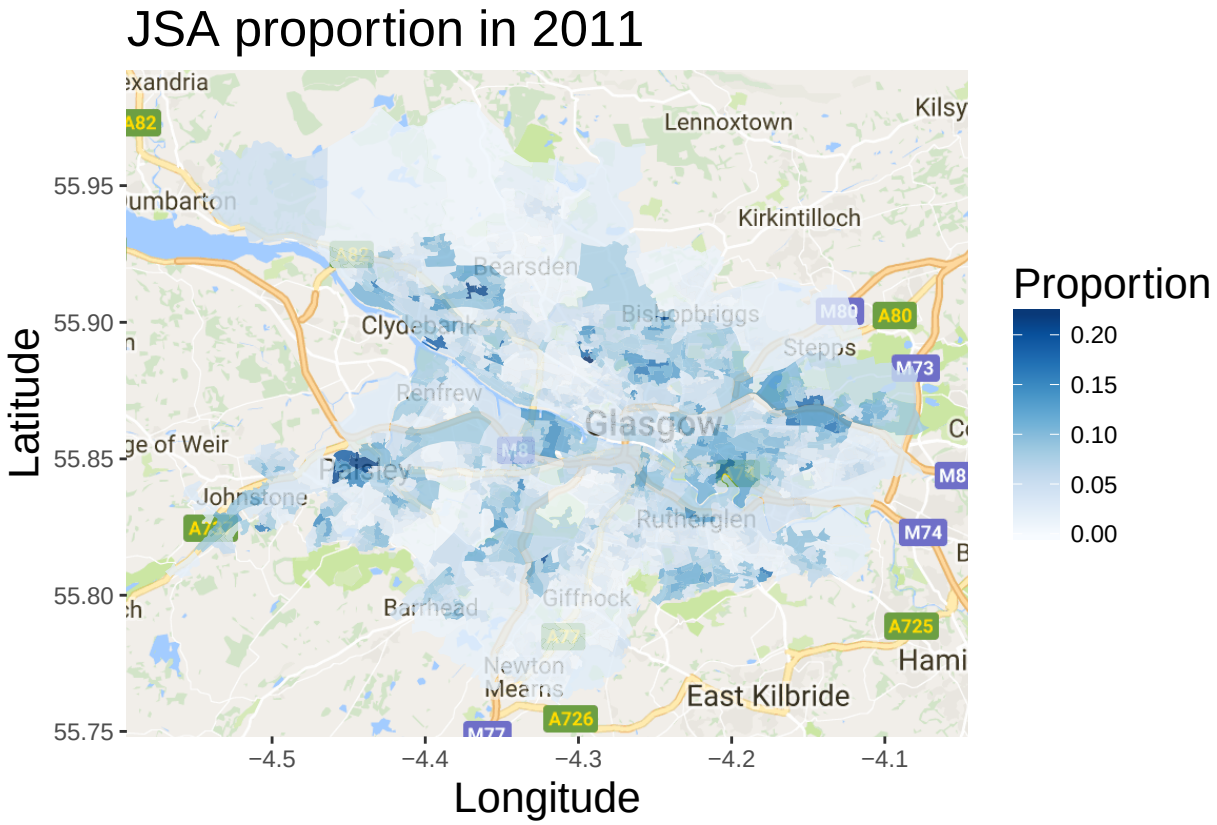
```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which
## will replace the existing scale.
```

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

```
## Warning in data.frame(x = x.major, y = y.range[1]): row names were found
## from a short variable and have been discarded
```

```
## Warning in data.frame(x = x.range[1], y = y.major): row names were found
## from a short variable and have been discarded
```




Here the `ggmap()` function plots the map. We have made the map too large via the zoom argument (zoom=11 was too small), and the last two lines with `scale_x_continuous()` and `scale_y_continuous()` shrink the scale to the size of the data. Finally, the `alpha` argument in the `geom_polygon()` function controls the level of transparency of the plotted data, and is between 0 and 1.

# 7. Further example

The second example focuses on the spatial pattern in average property prices across Greater Glasgow, Scotland, in 2008. This is an ecological regression analysis, whose aim is to identify the factors that affect property prices and quantify their effects. The data come from the Scottish Statistics database (*http://statistics.gov.scot*), and the study region is the Greater Glasgow and Clyde health board (GGHB), which is split into 271 intermediate geographies (IG). These IGs are small areas that have a median area of 124 hectares and a median population of 4,239. The data are stored in `housedata.csv`, and contain the following columns.

- `IG` - A unique code identifying each intermediate geography.
- `price` - The average property price.

- `crime` - The crime rate in terms of the number of crimes per 10,000 people.
- `rooms` - The average number of rooms in a property.
- `sales` - The number of property sales in 2008.
- `driveshop` - The average time taken to drive to the nearest shopping centre.
- `type` - The most prevalent property type.

Modelling here will be done on the natural log of the `price` variable, because the data on the original scale are skewed and not appropriate for modelling with a Gaussian distribution. The shapefile elements for the IGs are stored in `IG.shp` and `IG.dbf`.

**Task** - Read in the data and map and compute Moran's I statistic for the natural log of the average property price.

# References

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

Kavanagh, L, D Lee, and G Pryce. 2016. "Is Poverty Decentralizing? Quantifying Uncertainty in the Decentralization of Urban Poverty." *Annals of the American Association of Geographers* 106: 1286–98.