Confident Spatial Analysis and Statistics in R & GeoDa

Nick Bearman - Geospatial Training Solutions

Learning Outcomes:	R Functions & Libraries:
Understand Linked Displays in GeoDa	*brushing* (pg. 1)
Perform Local Indicators of Spatial Association in GeoDa	*LISA* (pg. 3)
Understand how to read in a variety of formats	`readOGR()` (pg. 5)
Know how to reorder data	`order()` (pg. 12)
Understand creating and using functions within R	`function ()` (pg. 13)
Know how to use buffers within R	`gBuffer()` (pg. 14)

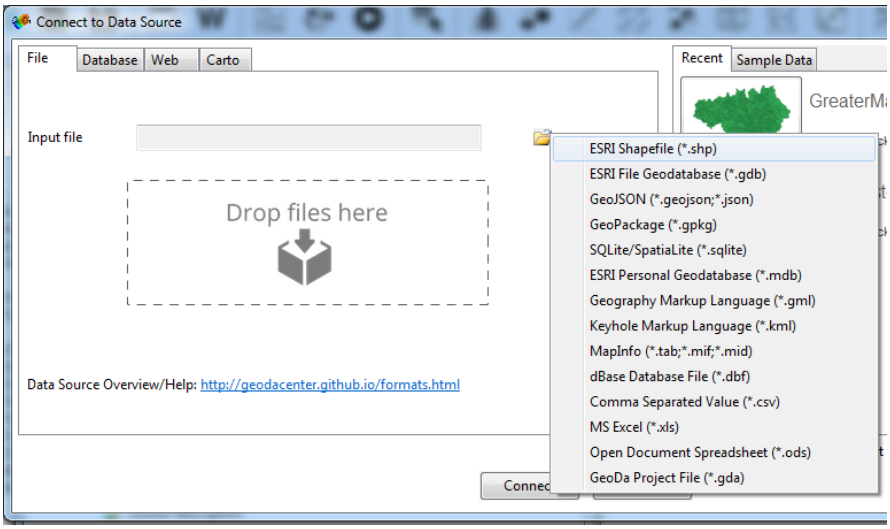
Practical 1: Spatial Analysis

We are going to be performing some Exploratory Data Analysis using a program called GeoDa. The data we are using covers Manchester, UK.

For the analysis, we are interested in the deprivation levels. First of all, let's have a look at the data in a program called GeoDa.

Download the data from http://bit.ly/csaprac1 and extract the zip file.

- Click **Start > GeoDa**
- When GeoDa opens, click on the folder icon
- Select **ESRI Shapefile (.shp)**
- Browse to and open **GreaterManchester_lsoa_2011_imd.shp**



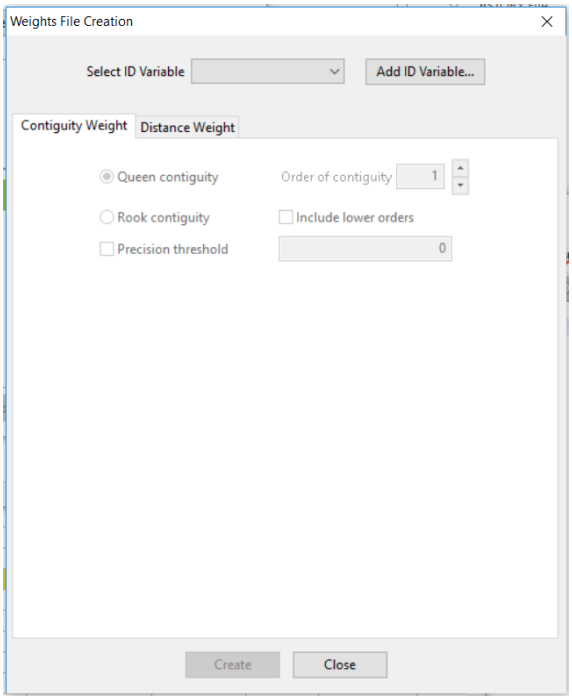
GeoDa should open a map display of the data automatically. We can create a simple choropleth map in here by right clicking on the map and then select **Change Current Map Type**. Select **Natural Breaks > 5** and choose the variable to map (we want **IMDscore**). Then a classified map will appear, similar to one we have created in RStudio.

Also open the attribute table and a histogram for `IMDscore`. GeoDa allows us to use brushing or linked displays. Select one of the larger bars in the histogram and you will see that the same entries are highlighted on the map and in the attribute table. You can also draw a box on the map (left-click and drag) and that will highlight the entries as well (you can drag the box around).

How is this different to interacting with the data in RStudio, or QGIS/ArcGIS if you have used those packages? Is it better / easier to use? Or harder to use?

We will now perform some spatial analysis within GeoDa. There is some preparatory work to do beforehand, which is creating the spatial weights. This is how GeoDa works out which polygons are next to each other and which ones to include when running local analysis (these are how we define 'neighbours', as I mentioned in the presentation).

Select **Tools > Weights Manager** and click **Create**. The following screen is displayed.

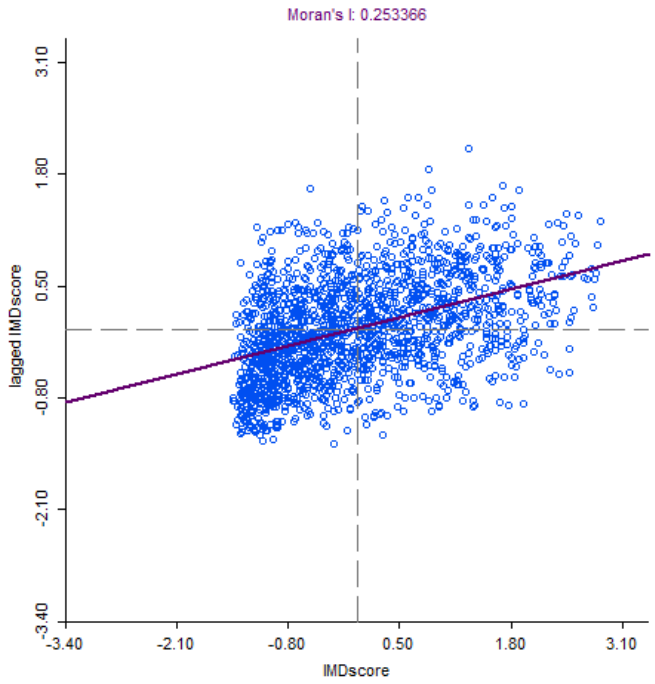


- Select **Add ID Variable...** and accept the default name.
- You now need to choose the neighbourhood weight method. Select **Queen Contiguity** and click **Create**.
- The neighbourhood weight matrix is saved in a file. A suggested name is given. This is the LSOA shape file name with a suffix of .gal. Select **Save** and after a few seconds you should see a message **Weights file `GreaterManchester_lsoa_2011_IMD.gal` created successfully**.
- Click **OK** on this window and **Close** on the Weights File Creation window. The Weights Manager window should show the details of the neighbours you have just selected.

Click **Histogram** and you can see how many neighbours each polygon has. In my data, five is the most common number of neighbours. Close this and open the **Connectivity Map**, which shows the neighbours for each polygon. Close this and the Weights Manager when you are happy with it.

Moran's I

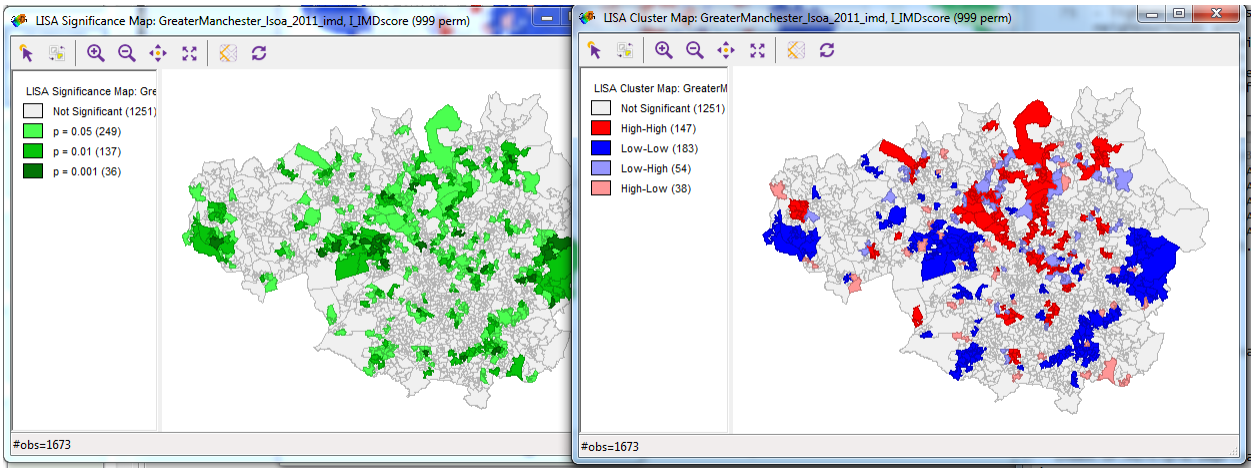
We can now start to explore the spatial patterns of the model we want to develop. Click on the **Space** item on the GeoDa menu. First we are going to look at the global indicator of spatial autocorrelation. Select the first menu item **Univariate Moran's I**. A list of variables is displayed. Choose the `IMDscore` variable. Select the weights file we created earlier.



A new window opens with a scatterplot of the IMDScore variable and the lagged IMDscore variable. The lagged IMDscore variable is the value of the IMD Score for the selected neighbourhood and its neighbours as defined when we set the spatial weights. The scatter-plot suggests a positive correlation between the variables. This means that there is a weak positive spatial autocorrelation. It has a value of 0.2534 confirming this positive spatial autocorrelation. We will now look at the local indicators of spatial association.

Local Indicators of Spatial Association (LISA)

- Select **Space > Univariate Local Moran's I**.
- Select the `IMDscore` variable from the list displayed and select the weights file used earlier (it should be selected automatically).
- A screen with three checkboxes is shown. Check all three and click **OK**. A number of windows are produced.
- The scatterplot is the same as we produced in the previous section. The two maps shown below are produced. You will probably need to expand them to full screen to see the legends and patterns displayed.



The first map shows the extent to which neighbouring areas are clustered:

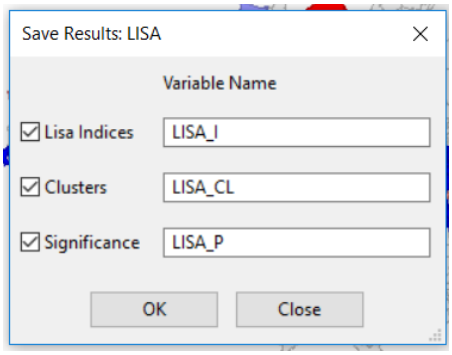
- red areas are neighbourhoods with a high proportion of deprived areas surrounded by similar neighbourhoods
- darker blue areas are neighbourhoods with a low proportion of deprived areas surrounded by similar neighbourhoods
- lighter blue areas are neighbourhoods with a low proportion of deprived areas surrounded by neighbourhoods with a high proportion of deprived areas
- pink areas are neighbourhoods with a high proportion of deprived areas surrounded by neighbourhoods with a low proportion of deprived areas
- white areas are neighbourhoods where the surrounding neighbourhoods do not generate a statistically significant relationship.

The second map shows the level of statistical significance. The values are 0.05, 0.01, 0.001 and 0.0001.

We can see that areas of high and low deprivation cluster together.

Now that we have the clustering output, we can save these to the shapefile and then load them into R to map or to do other analysis. *Moving data from one GIS application to another is something that happens regularly with many of the projects I am working on!*

Right click on the map and choose **Save Results**. Tick all three boxes (to save all of the results). This will create a new variable in the shapefile for each of the results we are saving. It should look like this:



Open the attribute table within GeoDa and you can see the three new variables on the right hand side of the table. Click the Save button to save the updated shapefile.

Moran's I in R (optional exercise)

We can also perform the same analysis in R. Load this shapefile back into RStudio. We need to use the `SP` library for these commands.

```
#load library
  library(rgdal)
#read in file
  GreaterManchester_lsoa_2011_imd <- readOGR(".", "GreaterManchester_lsoa_2011_imd")
```

Your data should look like this:

```
head(GreaterManchester_lsoa_2011_imd@data)
```

```
       CODE        NAME    LSOAname   LADcode LADname IMDscore IMDrank
0 E01004766 Bolton 005A Bolton 005A E08000001  Bolton   29.998    8241
1 E01004767 Bolton 005B Bolton 005B E08000001  Bolton   21.420   13147
2 E01004768 Bolton 001A Bolton 001A E08000001  Bolton    5.837   29371
3 E01004769 Bolton 003A Bolton 003A E08000001  Bolton    8.667   25890
```

```
4 E01004770 Bolton 003B Bolton 003B E08000001  Bolton    8.189   26504
5 E01004771 Bolton 003C Bolton 003C E08000001  Bolton    5.415   29855
  IMDdecile
0         3
1         5
2         9
3         8
4         9
5        10
```

To do the Moran's I analysis, we need another library:

```
#load the library
  library(spdep)
#work out the neighbours
  GreaterManchester_lsoa_nb <- poly2nb(GreaterManchester_lsoa_2011_imd)
#reformat the list of neighbours to a weight matrix
  GreaterManchester_lsoa_lw <- nb2listw(GreaterManchester_lsoa_nb)
#run the test
  moran.test(GreaterManchester_lsoa_2011_imd@data$IMDscore, GreaterManchester_lsoa_lw)
```

```
    Moran I test under randomisation

data:  GreaterManchester_lsoa_2011_imd@data$IMDscore
weights: GreaterManchester_lsoa_lw

Moran I statistic standard deviate = 17.07, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     0.2533629100     -0.0005980861      0.0002213410
```

This gives us the Moran's I statistic, as well as some associated variables. It is possible to run a whole range of analysis, which goes beyond the scope of this course.

Practical 2: Mapping and R/RStudio Recap

We are going to be looking at a range of data for Greater Manchester and we'll be using both RStudio and GeoDa.

This is a good time to remind you about using scripts (i.e. make sure you use one!). It's also a good idea to use the project facility within RStudio. To do this, open RStudio, click **File > New Project...**. Then click **New Directory > Empty Project**, select a folder (e.g. Documents) and then type a new folder name where the project will be saved. Finally click **Create Project**. This will be your working directory.

First, let's load one of the libraries we need.

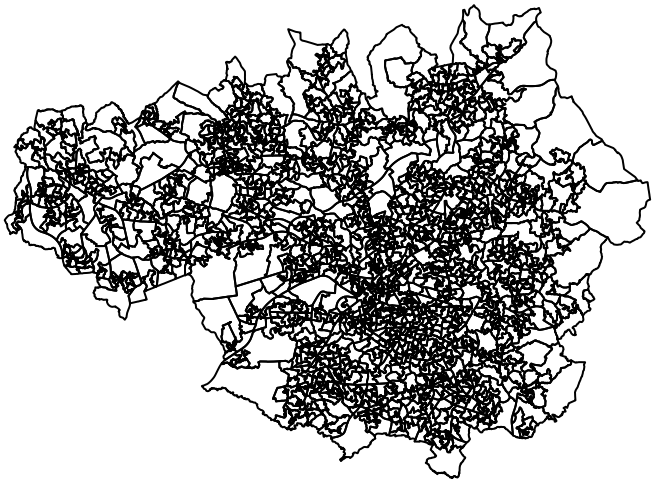
```
library(rgdal)
```

Next, read in some data and plot the data.

```
#download data
  download.file("http://www.nickbearman.me.uk/data/r/GreaterManchester_lsoa_2011.zip",
                "GreaterManchester_lsoa_2011.zip")
#unzip
  unzip("GreaterManchester_lsoa_2011.zip")
```

```
#read in data
  manchester_lsoa <- readOGR(".", "GreaterManchester_lsoa_2011")
```

```
#plot data
  plot(manchester_lsoa)
```



Each LSOA has a code and a name, we are looking at a range of IMD values; we need to join the spatial data to the attribute data. Download the IMD data from **http://bit.ly/2kFAlex** (or https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/467774/File_7_ID_2015_All_ranks__deciles_and_scores_for_the_Indices_of_Deprivation___and_population_denominators.csv). This file contains the IMD data for 2015 for all LSOAs in England.

There are 3 measures in this data - the IMD score, rank and decile. We will primarily be using the score, where the area with the highest score is the most deprived (with a rank of 1 and in the first decile).

Download the CSV file to your working directory, rename it to something more sensible and read it in:

Renaming files with long names to a much shorter file name is a good way of avoiding typos.

```
imd <- read.csv("imd.csv", header = TRUE)
```

Using `head(imd)` we can see some of the field names are rather long. Shorten them in RStudio (using the code below). For the moment we only need the IMD Score, Rank and Decile fields. Delete the others and we can always add them back in later if we need them *(this is an advantage of having a well-commented script!)*.

```
#delete columns we don't need
  imd <- imd[,1:7]
#rename columns
colnames(imd) <- c("LSOAcode","LSOAname","LADcode","LADname","IMDscore","IMDrank","IMDdecile")
```

Next, we need to join the data together.

```
manchester_lsoa@data <- merge(manchester_lsoa@data,imd,by.x="CODE",by.y="LSOAcode", all.x=TRUE)
```

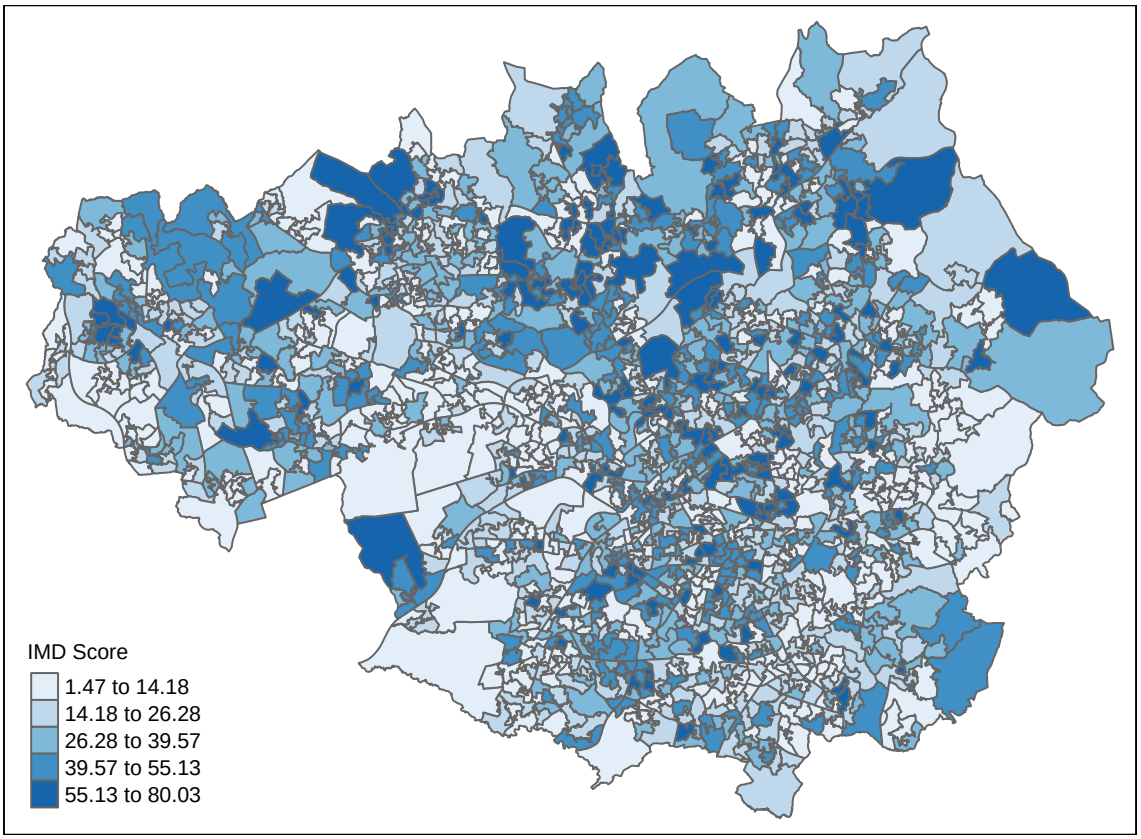
You should have something like this:

```
head(manchester_lsoa@data)
```

```
       CODE        NAME    LSOAname   LADcode LADname IMDscore IMDrank
1 E01004766 Bolton 005A Bolton 005A E08000001  Bolton   29.998    8241
2 E01004767 Bolton 005B Bolton 005B E08000001  Bolton   21.420   13147
3 E01004768 Bolton 001A Bolton 001A E08000001  Bolton    5.837   29371
4 E01004769 Bolton 003A Bolton 003A E08000001  Bolton    8.667   25890
5 E01004770 Bolton 003B Bolton 003B E08000001  Bolton    8.189   26504
6 E01004771 Bolton 003C Bolton 003C E08000001  Bolton    5.415   29855
  IMDdecile
1         3
2         5
3         9
4         8
5         9
6        10
```

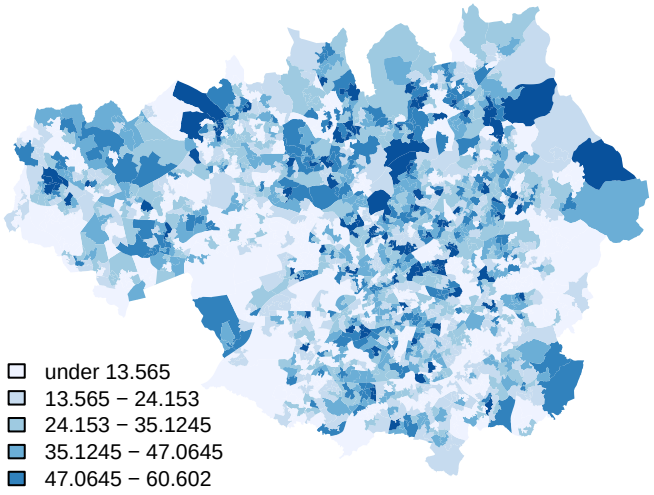
We can also now plot the IMD score across Greater Manchester, using `tmap` (this code should be familiar to you!):

```
#load library
  library(tmap)
#using tmap
  tm_shape(manchester_lsoa) +
    tm_polygons("IMDscore", title = "IMD Score", palette = "Blues", style = "jenks") +
    tm_layout(legend.title.size = 0.8)
```



We can also plot the data with the code for SP, using the base `plot()` command, not making use of the `tmap` library:

```
#load libraries
  library(maptools)
  library(classInt)
  library(RColorBrewer)
#select variable to map
  var <- manchester_lsoa@data[,"IMDscore"]
#set colours & breaks
  breaks <- classIntervals(var, n = 6, style = "fisher")
  my_colours <- brewer.pal(6, "Blues")
#plot map
  plot(manchester_lsoa, col = my_colours[findInterval(var, breaks$brks, all.inside = TRUE)],
       axes = FALSE, border = rgb(0.8,0.8,0.8,0))
  #the 4th parameter in the RGB value is transparency and
  #0 means transparent (so we don't get a border at all).
#draw legend
  legend(x = 350382, y = 392310, legend = leglabs(breaks$brks), fill = my_colours,
         bty = "n", cex = 0.7)
#add a title
  title("Greater Manchester: IMD 2015 Score by LSOA")
```



Now we have a map of IMD by LSOA in Greater Manchester.

Make sure you are happy with the code you have used. Check the glossary, help in RStudio (`?functionname`) or Google for any functions you have not come across before.

Practical 3: Spatial Decision Making

Public transport has an important role to play in tackling deprivation. We can use RStudio to see where the trams in Manchester run and how they relate to deprived areas.

Firstly, load the tramline data, available as a GeoJSON file. This is a different type of file to a shape file and is an XML based format (open it in Notepad if you are interested in how it is structured). `readOGR()` can read all sorts of file formats, so we can adapt the code easily.

```
#download data
  download.file("http://www.nickbearman.me.uk/data/r/tram.zip","tram.zip")
#unzip
  unzip("tram.zip")
#read in tramline data
  tramlines <- readOGR(dsn = "tramlines.geojson")
#if this doesn't work, try:
# tramlines <- readOGR(dsn = "tramlines.geojson", layer="OGRGeoJSON")
```

We also need the tram stations, which are only available as a CSV file with Eastings and Northings.

```
#read in CSV with tram station locations
  tram_stations_CSV <- read.csv("metrolink-stations.csv", header = TRUE)
```

Read the CSV file in (metrolink-stations.csv) and it should look like this:

```
head(tram_stations_CSV)
```

```
         X        Y  Label_Text Phase  Condition TIF Object_Des Object_Typ
1 382360.6 403798.8 Heaton Park 1 & 2 functional  NA                Station
2 383530.3 403163.7 Bowker Vale 1 & 2 functional  NA                Station
3 384114.2 402316.2   Crumpsall 1 & 2 functional  NA                Station
4 380431.9 410525.9        Bury 1 & 2 functional  NA                Station
5 378833.2 407354.4   Radcliffe 1 & 2 functional  NA                Station
6 380523.0 406114.1  Whitefield 1 & 2 functional  NA                Station
  Comments
1
2
3
4
5
6
```

We now need to convert this to a `SpatialPointsDataFrame`, like with the crime data in the Introduction course.

From `head(tram_stations_CSV)`, we can see that the data consists of a number of columns, each with a heading. Two of these are called **X** and **Y**. These are the columns that give the coordinates (Eastings and Northings, as they are in BNG) of each station in the data you have just downloaded. There are also other columns, with various information. At the moment, the data is just in a data frame object - not any kind of spatial object. To turn it into a SpatialPointsDataFrame, enter the following:

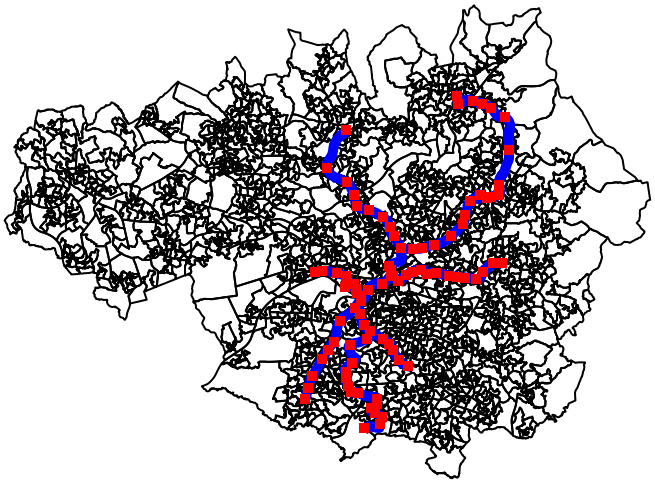
```
#extract coordinates
  coords <- cbind(Easting = tram_stations_CSV$X, Northing = tram_stations_CSV$Y)
#create spatialPointsDataFrame, removing coordinates from the @data element
#we can use the same projection information as manchester_lsoa as we know it is identical
  tram_stations <- SpatialPointsDataFrame(coords, tram_stations_CSV[, -(1:2)],
                                     proj4string = manchester_lsoa@proj4string)
```

This creates a `SpatialPointsDataFrame` object. This second line (starting `coords`) prepares the coordinates into a form that the `SpatialPointsDataFrame` can use. The `SpatialPointsDataFrame` function on the forth line takes three arguments - the first is coordinates, created in the line above. The second argument is the data frame **minus** (i.e. not including) columns 1 and 2 - this is what `-(1:2)` indicates. These columns provide all the non-geographical data from the data frame. The third is the coordinate system that the data is currently in (British National Grid). The resulting object `tram_stations` is a spatial points geographical shape object, whose points are each recorded crime in the data set you download.

Try `head(tram_stations@data)` to see the attribute table, similar to earlier.

We can also plot the data, on top of the Greater Manchester LSOAs. Note the change in colour (`col`) and size (`cex` for points and `lwd` for lines), to make the lines and stations more visible. We can also use different symbols, by substituting in `pch = 1` for `pch="."`. Try searching in Google "R plot pch" and see what the different options are.

```
#plot LSOAs
plot(manchester_lsoa)
#plot tram lines, in blue
plot(tramlines, col = "blue", lwd = 5, add = TRUE)
#plot stations, in red
plot(tram_stations, pch = ".", col = "red", cex = 5, add = TRUE)
```



We can see what the deprivation level is at each tram station, using a point in polygon analysis. The points are the tram stations and the polygons are the LSOAs.

```
# This is another R package, allowing GIS overlay operations
  library(rgeos)
# this extracts the tram stations that are over the LSOAs (all of them in our case)
  z <- tram_stations[!is.na(over(tram_stations, geometry(manchester_lsoa))),]
# this takes the attribute data from the LSOA that contain the tram_stations,
  #and adds it to the data frame of z
    z@data <- data.frame(z@data, na.omit(over(tram_stations, manchester_lsoa)))
#this copies the project from tram_stations to z
    z@proj4string <- tram_stations@proj4string
#copy z to tram_stations_joined
    tram_stations_joined <- z
#preview data
    head(tram_stations_joined)
```

```
   Label_Text Phase  Condition TIF Object_Des Object_Typ Comments
1 Heaton Park 1 & 2 functional  NA               Station
2 Bowker Vale 1 & 2 functional  NA               Station
3   Crumpsall 1 & 2 functional  NA               Station
4        Bury 1 & 2 functional  NA               Station
5   Radcliffe 1 & 2 functional  NA               Station
6  Whitefield 1 & 2 functional  NA               Station
       CODE            NAME        LSOAname   LADcode    LADname IMDscore
1 E01005150 Manchester 029A Manchester 029A E08000003 Manchester   23.782
2 E01005263 Manchester 046A Manchester 046A E08000003 Manchester   39.483
```

```
3 E01005267 Manchester 046B Manchester 046B E08000003 Manchester   27.593
4 E01005135 Manchester 002B Manchester 002B E08000003 Manchester   58.037
5 E01005097 Manchester 012C Manchester 012C E08000003 Manchester   65.478
6 E01004827     Bolton 021A     Bolton 021A E08000001     Bolton   49.477
  IMDrank IMDdecile
1   11557         4
2    4664         2
3    9405         3
4    1113         1
5     508         1
6    2316         1
```

Z is just a temporary variable to hold the updated tram_stations information. If we wanted to be tidy, we should really delete the z variable when we have finished: `rm(z)`

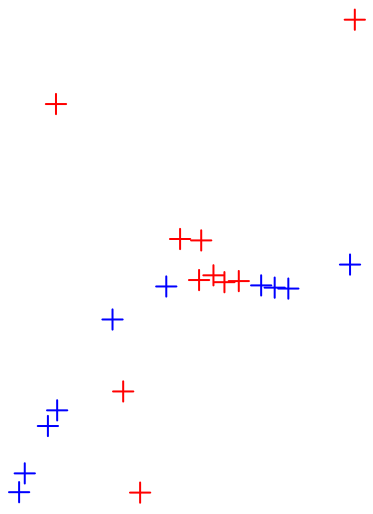
Reordering data

There is an `order()` function in R that allows us to reorder data. This is particularly useful if we want to display the 10 most and 10 least deprived stations! This code below creates a new variable with the data in a different order.

```
tram_stations_joined_reordered <-
  tram_stations_joined[order(tram_stations_joined@data$IMDscore, decreasing = TRUE),]
```

We can then plot the top and bottom 10 stations:

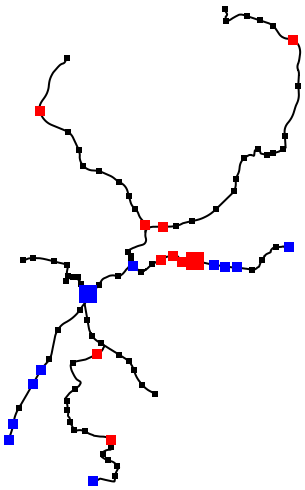
```
plot(tram_stations_joined_reordered[1:10,], col = "red")
plot(tram_stations_joined_reordered[81:91,], col = "blue", add = TRUE)
```



So, which tram stop is the most deprived? and which is the least deprived?

```
#most deprived
tram_stations_joined_reordered[1,]
#least deprived
tram_stations_joined_reordered[91,]
```

Can you create a map similar to this (see over), with the 10 most deprived stations in red and the top 10 least deprived stations in blue as well as identifying the most and least deprived stations? As an optional extra, see if you can include an appropriate legend and title.



Creating a Function (optional exercise)

The code for doing a spatial join can easily be turned into a function within R. A function allows us to repeat a section of code with different data, without having to type out all of the lines again. This also hides some of the complexity from the user. For example, if we wanted to do a spatial join between railway stations and LSOAs, we could just substitute railway stations for trams. A function we have written ourselves works exactly the same way as many other functions we have used (`read.csv()`, `readOGR()`, etc.). Running a `SpatialJoin()` function we have written would look a bit like this:

```
tram_stations_joined <- SpatialJoin(tram_stations, manchester_lsoa)
```

Where `tram_stations` and `manchester_lsoa` are the parameters, `SpatialJoin` the function name and `tram_stations_joined` the output. Currently, if you ran the line above, R would give an error message, as it doesn't know what that function is. To create the function (and it will appear in the environment list on the right) run this code:

```r
#function to join points with the attribute data of the polygons they overlay
SpatialJoin <- function(pts, polys) {
  #error checking
    if (!inherits(polys, "SpatialPolygonsDataFrame"))
        stop("MUST BE SP SpatialPolygonsDataFrame OBJECT")
    if ((inherits(pts, "SpatialPointsDataFrame") | inherits(pts, "SpatialPoints")) == FALSE)
        stop("Must be sp SpatialPointsDataFrame object")
  #extract points in overlay
    z <- pts[!is.na(over(pts, geometry(polys))),]
  #join attribute data
    z@data <- data.frame(z@data, na.omit(over(pts,polys)) )
  #update projection
    z@proj4string <- pts@proj4string
  #return z
    z
}
```

In a function it is common to use generic terms to refer to the inputs and outputs (e.g. pts, polys, z), because the function doesn't know (or care) whether we are dealing with tram stations or railway stations - it just cares that it is a point (or polygon) layer.

This is where error checking comes in, which is what takes up the first half of the function. These check whether the poly variable is a polygon and the pts variable a points layer. If they are not, it will stop running and generate an error message. Try creating the function and running it.

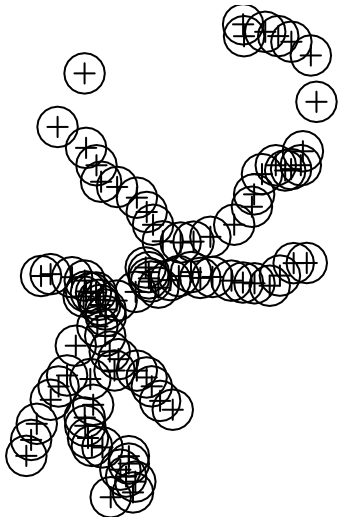
We can then repeat the process of working out which train station is the most and least deprived. The train stations for Manchester are included in the tram.zip file as `rail-stations.geojson` (originally from https://data.gov.uk/dataset/metrolink-and-rail-stations). Try writing the code to read the data in, join it, reorder it and display it on the map.

Tram Stop Buffers

Currently we have been looking at just the LSOA each tram stop is within. However we could look at the LSOAs that are within a set distance (for example a 15 minute walk) of each tram stop. Why do you think we might we want to do this?

For this, we need to use a buffer - a circle of a set distance around each tram stop. At a speed of 3 mph, this is equal to about 1.2km. We can use the `gBuffer` function, from the `rgeos` library.

```r
#plot the tram stations
  plot(tram_stations_joined)
#calculate the buffer (distance is 1200 meters)
  tram_stations_joined_buffer <- gBuffer(tram_stations_joined, width = 1200, byid = TRUE)
#plot the buffer (add to existing plot)
  plot(tram_stations_joined_buffer, add = TRUE)
```

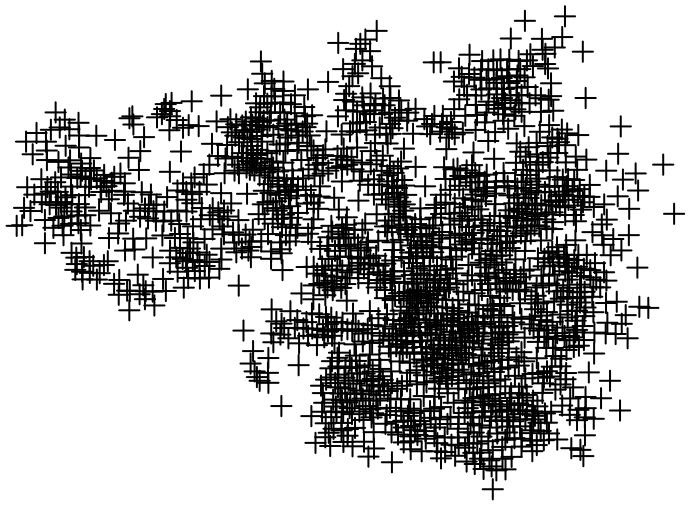


We now have the buffers (the circles) for each tram station. There are a number of different ways we can look at how the buffers and LSOAs relate to one another. We are going to look at a quite simple approach - convert the LSOAs to points and for each buffer take the average deprivation level of the LSOAs within the buffer.

There are a number of limitations with this method and this is something to think about as you work through this process: how could you do the analysis differently in the future?

First, we need to convert the LSOA polygons into points. Fortunately, there is a function (`gCentroid()`) which is part of the `rgeos` package which will do this for us. This calculates the centre point (centroid) for each LSOA. The function creates a `SpatialPoints` variable, which doesn't have an attribute table (`@data`), so we need to copy this over manually.

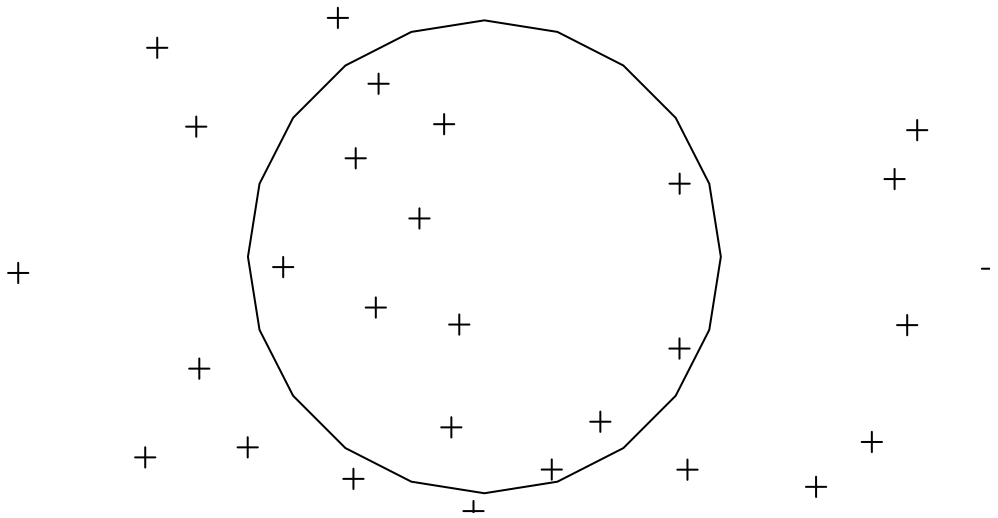
```
#convert polygons to points
  manchester_lsoa_points <- gCentroid(manchester_lsoa, byid = TRUE)
  plot(manchester_lsoa_points)
#create coordinates
  coords = data.frame(manchester_lsoa_points@coords[,1:2])
#create SpatialPointsDataFrame using coords and data
  manchester_lsoa_points <- SpatialPointsDataFrame(coords, data=data.frame(
    manchester_lsoa@data$IMDscore), proj4string= manchester_lsoa@proj4string)
#plot points
  plot(manchester_lsoa_points)
```



```
#show head
  head(manchester_lsoa_points@data)
```

```
  manchester_lsoa.data.IMDscore
1                        29.998
2                        21.420
3                         5.837
4                         8.667
5                         8.189
6                         5.415
```

```
#plot first tram station
  plot(tram_stations_joined_buffer[1,])
#plot lsoa points on top
  plot(manchester_lsoa_points, add = TRUE)
```



Now we have the train stations buffers as polygons and the LSOAs as points (as shown in the plot above). We can do the overlay manually by plotting the different data sets on top of one another (as we did above). However, R is unable to link each tramstop with the LSOAs within the buffer. For this we need to get R to do the overlay and save the associated information - i.e. the average IMD score for each tram station, based on the buffer.

```
#Create point in polygon list
  o <- over(tram_stations_joined_buffer,manchester_lsoa_points,returnList=TRUE)
#View length of the list
  length(o)
```

```
[1] 91
```

```
#If we examine the object o, we will see also see that this comprises a list of dataframes.
#The summary function tells you about an object - head, is used to wrap around the function
#so only the first six elements are shown
  head(summary(o))
```

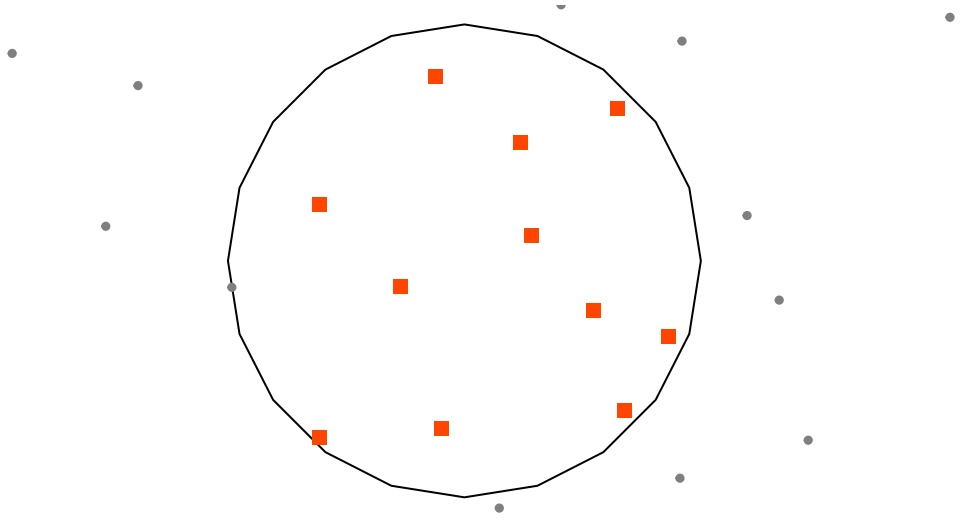
```
  Length Class      Mode
1 1      data.frame list
2 1      data.frame list
3 1      data.frame list
4 1      data.frame list
5 1      data.frame list
6 1      data.frame list
```

```
#View an item from the list (in this case sixth) (i.e. the sixth tram station and all of
#the LSOAs that are linked with it)
```

```
  o[[6]]
```

```
    manchester_lsoa.data.IMDscore
17                         23.549
18                         10.729
19                          5.851
59                         53.526
61                         49.477
62                         42.027
63                         44.206
64                         42.786
339                        46.694
343                        26.536
390                        10.093
```

```
#get a list of those LSOAs that are within this buffer using the rownames
#make them numeric
  row_o <- as.numeric(rownames(o[[6]]))
#plot tram station buffer number 6
  plot(tram_stations_joined_buffer[6,])
#Plot all the LSOA centroids
  plot(manchester_lsoa_points,pch=19,cex=.5,col="#7F7F7F",add=TRUE)
#Plot those LSOA centroids which are inside the buffer - i.e. we use the row_o to select the rows
  plot(manchester_lsoa_points[row_o,],pch=15,cex=1,col="#FF4500",add=TRUE)
```



This shows the tram station buffer (the circle), all of the LSOA centroids (grey dots) and those LSOA centroids within the buffer (red squares). For each tram buffer, we need to calculate the average IMD score,

as we are likely to have a range of deprivation levels within the tram station buffer. The `colMeans()` function will calculate the mean value for each column (as you might have guessed!). We want to do this for each tram station buffer and the `lapply()` function allows us to do this - to apply a function (`colMeans()`) across the list of tram station buffers. We can then join this back onto the tram_stations data frame.

```
  average_deprivation <- lapply(o, colMeans)
#View the first six items from the list
  average_deprivation[1:6]
```

```
$`1`
manchester_lsoa.data.IMDscore
                     31.29583

$`2`
manchester_lsoa.data.IMDscore
                      36.3762

$`3`
manchester_lsoa.data.IMDscore
                     38.92367

$`4`
manchester_lsoa.data.IMDscore
                     55.79322

$`5`
manchester_lsoa.data.IMDscore
                      44.827

$`6`
manchester_lsoa.data.IMDscore
                     32.31582
```

```
#collapse list back into a normal data frame
  tram_deprivation <- data.frame(matrix(unlist(average_deprivation),
                                        nrow=length(average_deprivation), byrow=T))
#Change the column names to something sensible
  colnames(tram_deprivation) <- c("average_deprivation")
#This should look like
  head(tram_deprivation)
```

```
  average_deprivation
1            31.29583
2            36.37620
3            38.92367
4            55.79322
5            44.82700
6            32.31582
```

```
#Join - the ordering has not been changed so this is valid
  tram_stations@data <- cbind(tram_stations@data,tram_deprivation)
#show data
  head(tram_stations@data)
```

```
   Label_Text Phase  Condition TIF Object_Des Object_Typ Comments
1 Heaton Park 1 & 2 functional  NA               Station
```

```
2 Bowker Vale 1 & 2 functional  NA               Station
3   Crumpsall 1 & 2 functional  NA               Station
4        Bury 1 & 2 functional  NA               Station
5   Radcliffe 1 & 2 functional  NA               Station
6  Whitefield 1 & 2 functional  NA               Station
  average_deprivation
1            31.29583
2            36.37620
3            38.92367
4            55.79322
5            44.82700
6            32.31582
```

We now have the average deprivation for each tram stop. **How is this different to our previous measure? What are the limitations?** Plot a map of your results.

- Discuss your results with a neighbour. How could the deprivation be calculated differently?
- Try running the analysis with a different buffer. How easy is this within R compared to another GIS?
- How about doing a polygon polygon overlay? How easy is this in R compared to another GIS? *See http://gis.stackexchange.com/questions/140504/extracting-intersection-areas-in-r for some interesting suggestions.*

If you want to install additional libraries, you can do this, but on centrally managed machines, it will only install onto that machine, so if you use a different machine you may need to rerun the installation command.

Future Developments (optional exercise)

We can extend this principle to look at groups of tram stops (either using just the stop location, or the buffer approach. For example, let's say there are some prospective new tram routes which could be built (see `future_tramlines` and `future_tramstops`).

We need to decide which route is the most important. For this exercise we are going to say that the route that gives access to the most deprived area(s) will be our recommended route (obviously there are many other factors that could be involved!).

Analyse the prospective new routes to see which reaches the areas with highest deprivation levels.

This practical was written using R 3.5.1 (2018-07-02) and RStudio 1.1.463 by Dr. Nick Bearman (nick@geospatialtrainingsolutions.co.uk).