6. Basic Spatial Analysis

This helpsheet will explore a variety of basic spatial analysis techniques, including *clipping*, *point in polygon* and *buffering*.

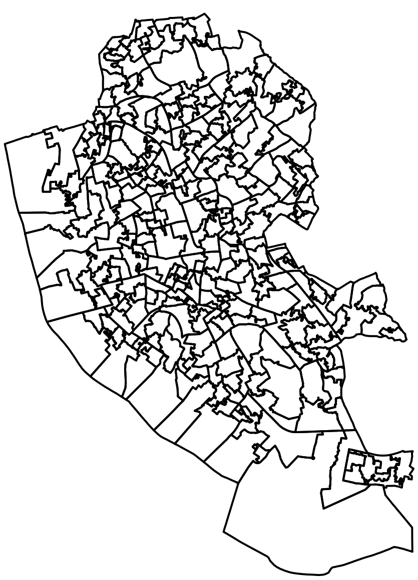
Clipping

Clipping allows us to use one set of boundaries to cut another, a bit like using a cookie cutter.

```
# Load the Libaries
library(rgdal)
library(maptools)
library(rgeos)
library(stringr)

# Set working directory
setwd("M:/R work")
# Download data.zip from the web
download.file("http://data.alex-singleton.com/r-helpsheets/6/data.zip", "data.zip")
# Unzip file
unzip("data.zip")

# Read in both shape files
LSOA <- readOGR(".", "england_LSOA_2011")
outline <- readOGR(".", "England-outline")
```

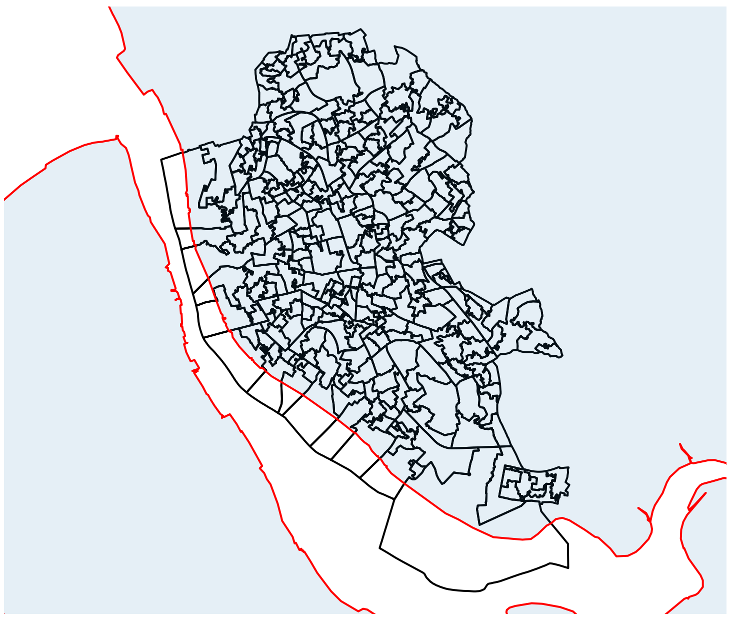
First of all, we can plot the LSOA zones in Liverpool.

```
# Plot the LSOA Map
plot(LSOA)
```



We can also plot the England outline, but if we just run `plot(outline)` it will replace the LSOA plot on the display. To add the `outline` layer to the existing plot window, we can run the code below which will plot the outline with a red border and we can also adjust the colour of the border to a red colour (`border="red"`), and the fill colour (`col="#2C7FB820"`) a shade of blue. These represent two ways of specifying colours. The second contains eight alphanumerics, the first six relate to a HEX colour code. To view various colours that can be used in R, have a look at the website http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf. The final two characters are the level of transparency (in this case 20%). *Sometimes when running R in Windows, the transparency option will not work - it will just fill it with a solid colour. In this case, just remove the* `col = "#2C7FB820"` *section from the plot command to generate a red outline.*

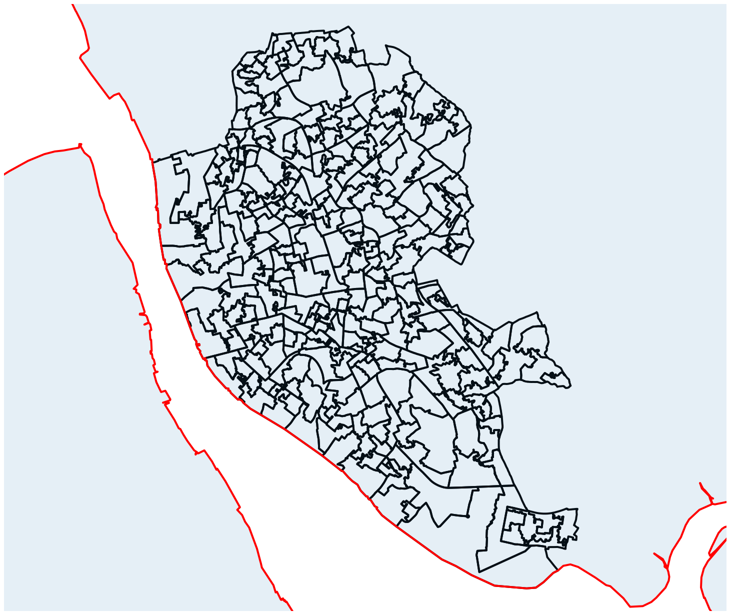
```
# Overplot the outline map
plot(outline, add = TRUE, border = "red", col = "#2C7FB820")
```



As you will notice, the LSOA boundaries cross the River Mersey and stop at the river centre line. This doesn't look very nice, so we can tidy this up by getting R to clip the LSOA boundaries where they cross the England outline border.

We do this using the `gIntersection` command, passing it the two layer variables (`outline` and `LSOA`). We can also tell R we just want it to use the area covering the LSOAs by specifying `byid = TRUE` and `id = my_area_id`. Be aware that the `gIntersection` command may take up to 90 seconds to run - do not worry if your computer appears to freeze. Just wait for the command to complete.

```
# set the area we want to cut
my_area_id <- as.character(LSOA@data$ZONECODE)
# run the Intersection command, saving output to clipLSOA, this may take
# anywhere up to 90 seconds to run
clipLSOA <- gIntersection(LSOA, outline, byid = TRUE, id = my_area_id)
# replot the map as above to see what we have done
plot(clipLSOA)
plot(outline, add = TRUE, border = "red", col = "#2C7FB820")
```



We have now removed the parts of the LSOAs that overlap the coastline, and the map looks much more attractive.

Point in Polygon Analysis

Point in polygon analysis is useful when you want to create a subset of points from a larger set based on their spatial location. In this example we will load a list of locations that relate to all doctors surgeries in England, and use the polygons of ward boundaries in Leeds to create a subset of the Leeds doctors surgeries. To begin with, we need to load the libraries and get the GP and Wards data.

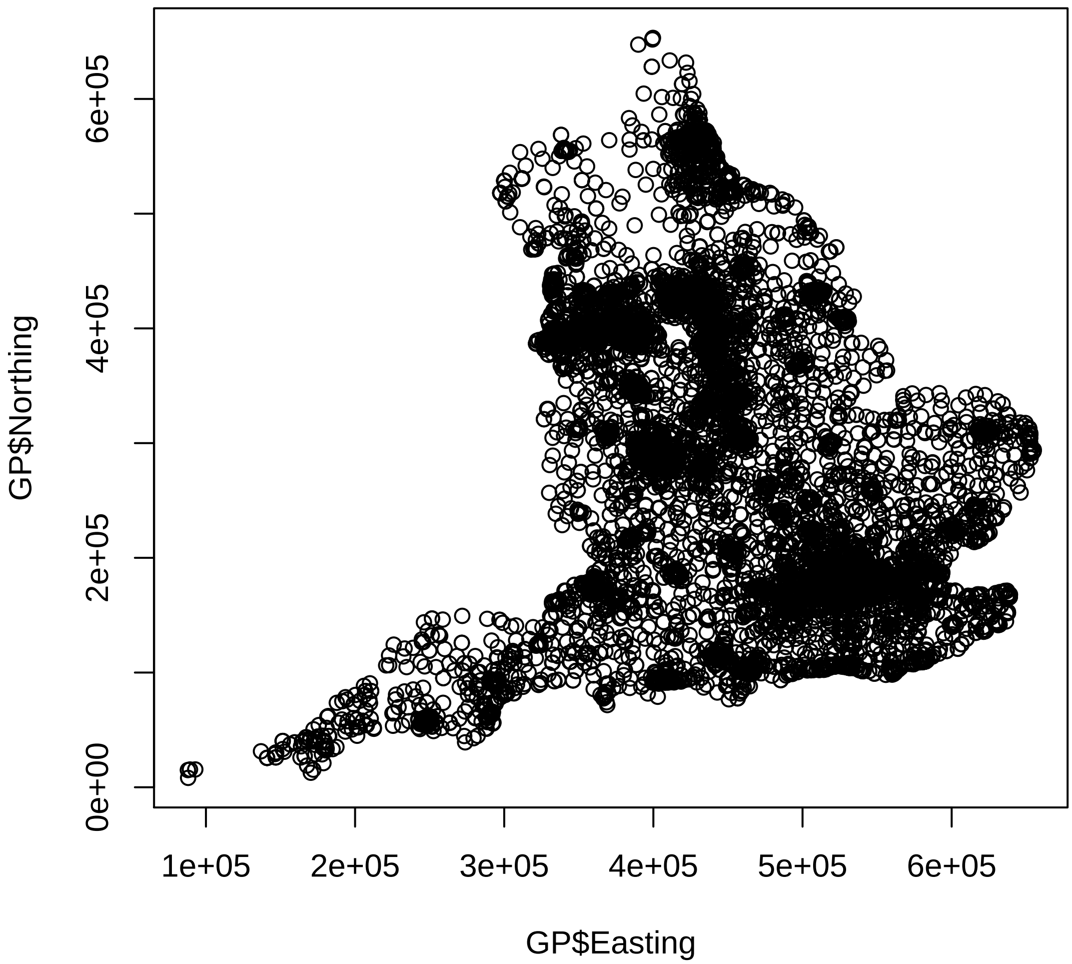
```
# Load the Libaries
library(maptools)
library(rgeos)

# Set working directory
setwd("M:/R work")
# Download data.zip from the web
download.file("http://data.alex-singleton.com/r-helpsheets/6/data.zip", "data.zip")
# Unzip file
unzip("data.zip")

# Read in shapefile
Wards <- readShapeSpatial("CAS-leeds", proj4string = CRS("+init=epsg:27700"))
```

It's worth having a quick look at the Leeds data so we know what it looks like:

```
# Plot Wards to check it has been read in correctly
plot(Wards)
```

The doctors surgeries data is quite untidy - once we've read it in, we need to remove some extra columns that we don't need, and rename the ones we do.

```
# Get Data
GP <- read.csv("General Practices 2006.csv", header = TRUE, skip = 3)

# Extract the columns we want
GP <- subset(GP, select = c("Practice.Doctor.s.Name", "Easting", "Northing"))

# Rename the columns to something more helpful
colnames(GP) <- c("Surgery", "Easting", "Northing")

# Do a plot to check what the data look like
plot(GP$Easting, GP$Northing)
```



This should look like the above. The next stage is to convert the data into a SpatialPointsDataFrame.

```
# Remove those GP without Easting or Northing
GP <- subset(GP, Easting != "" & Northing != "")
# Create a unique ID for each GP
GP$GP_ID <- 1:nrow(GP)
# Create the SpatialPointsDataFrame
GP_SP <- SpatialPointsDataFrame(coords = c(GP[2], GP[3]), data = data.frame(GP$Surgery,
    GP$GP_ID), proj4string = CRS("+init=epsg:27700"))
```

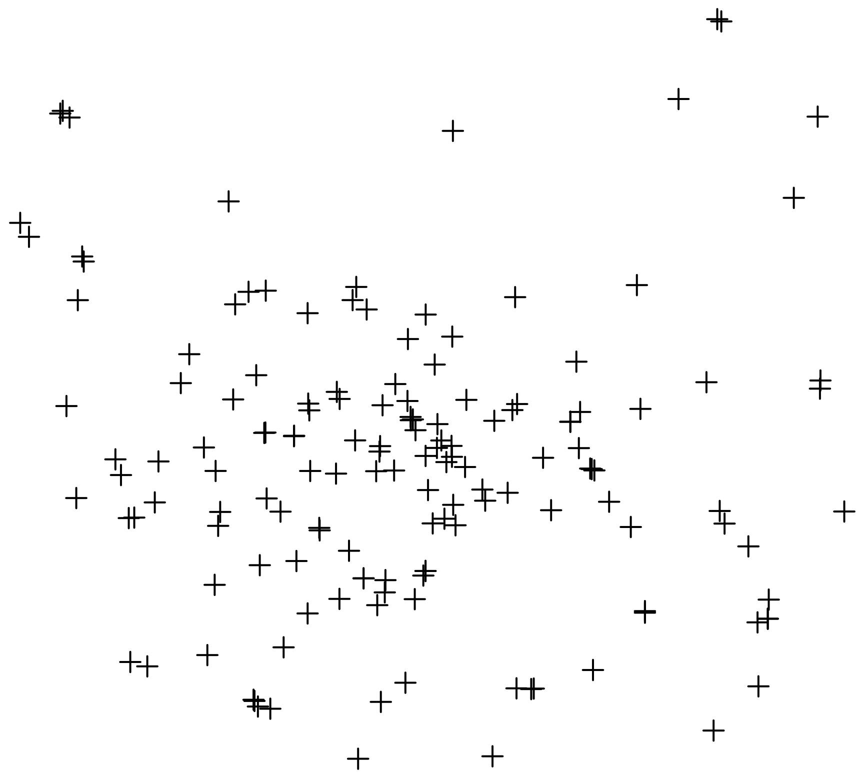
The first line contains a `subset` command which removes any of the entries which have a blank value for Northings or Eastings. `!=` means 'not equal to' and `&` means 'AND' so in "English" the command reads "overwrite the GP data frame with a subset of the GP data frame where the Easting field is not blank and the Northing field is not blank".

In a SpatialPointsDataFrame each entry must have a unique ID, so the second line creates an ID in the column `GP_ID`. The third line brings together the different elements to create the SpatialPointsDataFrame, `GP-SP`. `GP[2]` and `GP[3]` are the `Easting` and `Northing` columns respectively, and the `data =` section tells R which bits of the data frame to include. In this case we only want the surgery name (`GP$Surgery`) and the ID number (`GP$GP_ID`). The final term (`proj4string`) specifies which projection the data set is in `-` in this case, British National Grid (`epsg:27700`).

```
# Show the results
plot(GP_SP)
```

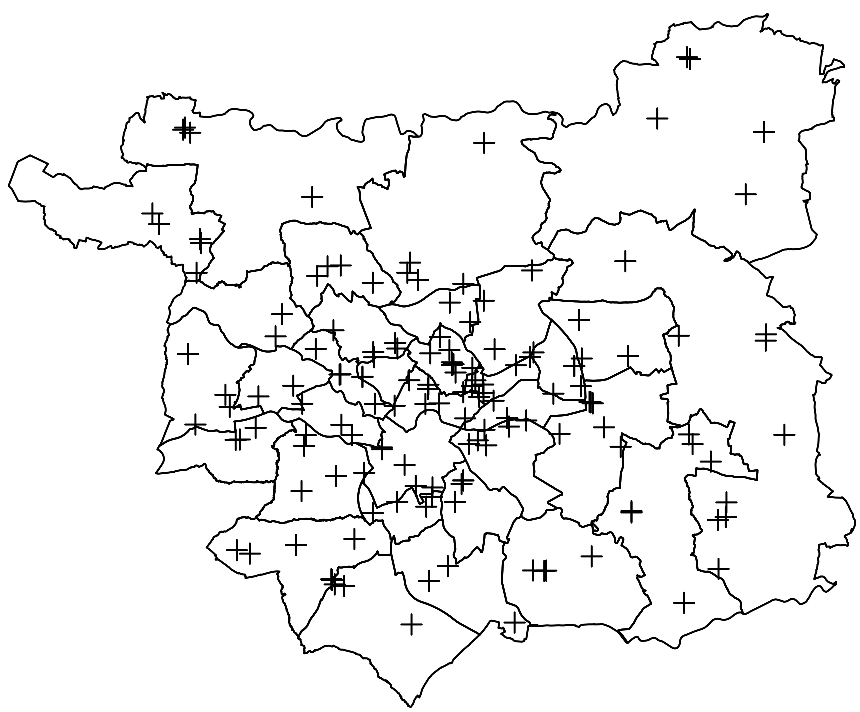
This plot will look similar to the previous one, but the data are now stored in a SpatialPointsDataFrame. We now can calculate a point in polygon, i.e. to select those points which lie within the boundary of Leeds. The forth line below uses a `!is.na` command. `is.an` is a command to test whether a value is 'NA' and `!` means the inverse, so the command is testing whether the value (of `GP_SP@data$label`) is not `NA`.

```
# point in polygon - returns a dataframe of the attributes of the polygons
# that the point is within.
o <- over(GP_SP, Wards)

# Many of these will be NA values - because most GPs are not in Leeds!
head(o)

# Add the attributes back onto the GP_SP SpatialPointsDataFrame (they are
# the same length)
GP_SP@data <- cbind(GP_SP@data, o)

# Use the NA values to remove those points not within Leeds
GP_SP_Leeds <- GP_SP[!is.na(GP_SP@data$label), ]

# Map your results
plot(GP_SP_Leeds)
```



We can also plot the points over the Leeds LSOAs:

```
plot(Wards)
plot(GP_SP_Leeds, add = TRUE)
```



We can also view the data in the GP_SP_Leeds data frame.

```r
# View the data slot of the results
head(GP_SP_Leeds@data)
```

```
                     GP.Surgery GP.GP_ID  gid ons_label         name
3591 Dr A M Marshall & Partners     3591 1664    00DAFA  Aireborough
3592    Dr Adams R J & Partners     3592 1662    00DAFU Morley North
3593    Dr Adams R J & Partners     3593 1662    00DAFU Morley North
3594    Dr Adams R J & Partners     3594 1662    00DAFU Morley North
3595                 Dr Ali S A     3595 3470    00DAFQ      Hunslet
3596   Dr Allman I G & Partners     3596 3430    00DAGG     Weetwood
      label
3591 08DAFA
3592 08DAFU
3593 08DAFU
3594 08DAFU
3595 08DAFQ
3596 08DAGG
```

Buffers

> *This section looks at buffers. It carries on from the section on points in polygon, so make sure you complete that section first.*

Buffers are often used in spatial analysis for defining context of points. In this example we will calculate a buffer from the doctors surgeries of a 10 minute walking distance, based on an average of 3 mph, which is around 1608m.

The rgeos package has a function called `gBuffer()` that can be used to create buffers around points, lines or polygon objects. In the following example we create a new SpatialPolygons object called `GP_SP_Leeds_Buffers`. This then needs to be converted into a SpatialPolygonsDataFrame object by joining the `@data` from `GP_SP_Leeds` back onto `GP_SP_Leeds_Buffers`. Spatial Polygons objects do not have the data slot.

```r
# buffers
GP_SP_Leeds_Buffers <- gBuffer(GP_SP_Leeds, width = 1608, byid = TRUE)

# Convert GP_SP_Leeds_Buffers into a SpatialPolygonsDataFrame (rather than
# SpatialPolygons) by joining the data of the GP_SP_Leeds
# SpatialPolygonsDataFrame
GP_SP_Leeds_Buffers <- SpatialPolygonsDataFrame(GP_SP_Leeds_Buffers, GP_SP_Leeds@data)
```

We can also now plot this on top of the Wards map.

```r
# Wards wards
plot(Wards, axes = FALSE, col = "#6E7B8B", border = "#CAE1FF")
# GP locations
plot(GP_SP_Leeds, pch = 19, cex = 0.4, col = "#5CACEE", add = TRUE)
# catchment buffers
plot(GP_SP_Leeds_Buffers, axes = FALSE, col = NA, border = "red", add = TRUE)
```