Introduction to QGIS, Spatial Data and Cartography

Day 2: Wednesday 25th November 2020

Learning Outcomes:	R Functions & Libraries:
Be able to use R to read in CSV data	`read.csv()` (pg. 3)
Be able to use R to read in spatial data	`st_read()` (pg. 4)
Know how to plot spatial data using R	`qtm()` (pg. 5) & `tm_shape()` (pg. 8)
Know how to customize colour & classifications	`style` (pg. 9)
Understand how to use loops for multiple maps	`for(){}` (pg. 12)
Know how to re-project spatial data	`st_transform()` (pg. 15)
Be able to perform point in polygon analysis	`poly.counts()` (pg. 15)
Know how to save shape files	`st_write()` (pg. 17)

Practical 1: Intro to R & GIS

R Basics

R began as a statistics program and is still used as one by many users. We are going to use a program called RStudio, which works on top of R and provides a good user interface. I'll talk a little bit about RStudio in the presentation, and the key areas of the window are highlighted overleaf.

- Open up RStudio (click **Start** and type in `RStudio` or double-click the icon on the desktop).

R can initially be used as a calculator - enter the following into the left-hand side of the window - the section labelled **Console**:

```
6 + 8
```

Don't worry about the `[1]` for the moment - just note that R printed out `14` since this is the answer to the sum you typed in. In these worksheets, sometimes I show the results of what you have typed in. This is in the format shown below:

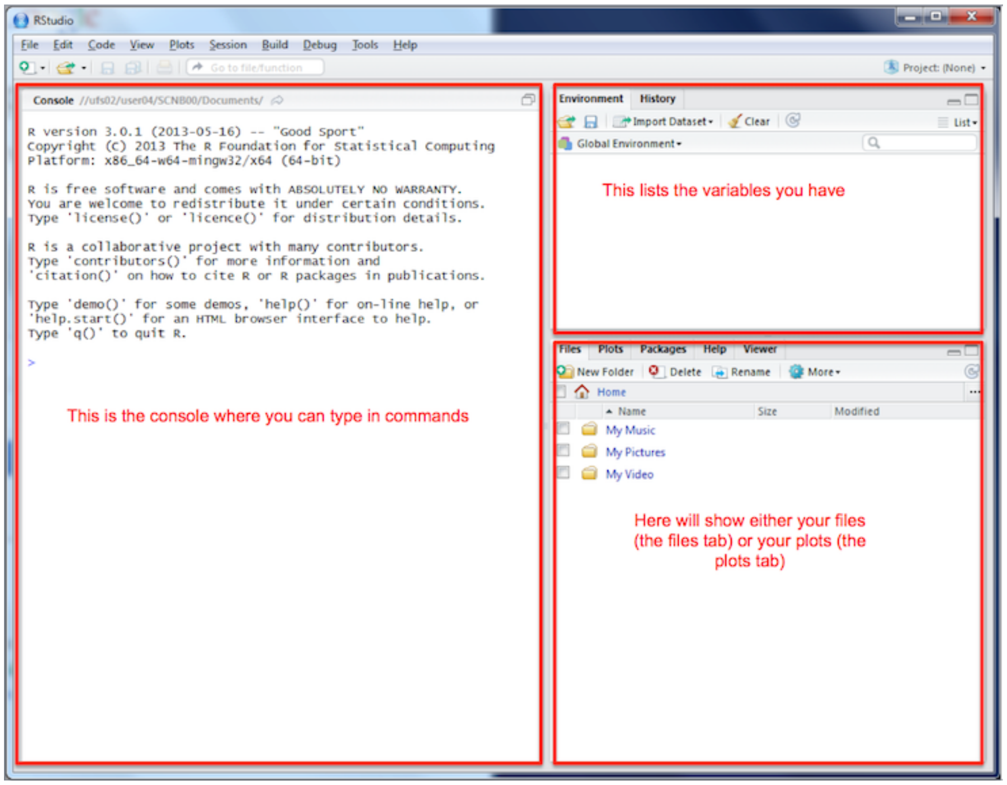
```
5 * 4
```

```
[1] 20
```

Also note that `*` is the symbol for multiplication here - the last command asked R to perform the calculation '5 times 4'. Other symbols are `-` for subtraction and `/` for division:

```
12 - 14
```

```
[1] -2
```


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Project: (None)
Console //ufs02/user04/SCNB00/Documents/
Environment History
Import Dataset
Clear
List
Global Environment
This lists the variables you have
Files Plots Packages Help Viewer
New Folder Delete Rename More
Home
Name Size Modified
My Music
My Pictures
My Video
This is the console where you can type in commands
Here will show either your files (the files tab) or your plots (the plots tab)

```
R version 3.0.1 (2013-05-16) -- "Good Sport"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 1: Screenshot of RStudio

```
6 / 17
```

```
[1] 0.3529412
```

You can also assign the answers of the calculations to variables and use them in calculations.

```
price <- 300
```

Here, the value `300` is stored in the variable `price`. The `<-` symbol means put the value on the right into the variable on the left, it is typed with a `<` followed by a `-`. The variables are shown in the window labelled **Environment**, in the top right. Variables can be used in subsequent calculations. For example, to apply a 20% discount to this price, you could enter the following:

```
price - price * 0.2
```

```
[1] 240
```

or use intermediate variables:

```
discount <- price * 0.2
price - discount
```

```
[1] 240
```

R can also work with lists of numbers, as well as individual ones. Lists are specified using the `c` function. Suppose you have a list of house prices specified in thousands of pounds. You could store them in a variable called `house.prices` like this:

```
house.prices <- c(120,150,212,99,199,299,159)
house.prices
```

```
[1] 120 150 212  99 199 299 159
```

Note that there is no problem with full stops in the middle of variable names.

You can then apply functions to the lists.

```
mean(house.prices)
```

```
[1] 176.8571
```

If the house prices are in thousands of pounds, then this tells us that the mean house price is 176,900 GBP. Note that on your display, the answer may be displayed to more significant digits, so you may have something like 176.8571429 as the mean value.

The Data Frame

R has a way of storing data in an object called a **data frame**. This is rather like an internal spreadsheet where all the relevant data items are stored together as a set of columns.

We have a CSV file of house prices and burglary rates, which we can load into R. We can use a function called `read.csv` which, as you might guess, reads CSV files. Run the line of code below, which loads the CSV file into a variable called `hp.data`.

```
hp.data <- read.csv("http://nickbearman.me.uk/data/r/hpdata.csv")
```

When we read in data, it is always a good idea to check it came in ok. To do this, we can preview the data set. The `head` command shows the first 6 rows of the data.

```
head(hp.data)
```

```
  ID Burglary Price
1 21        0   200
2 24        7   130
3 31        0   200
4 32        0   200
5 78        6   180
6 80       19   140
```

You can also click on the variable listed in the Environment window, which will show the data in a new tab. You can also enter:

```
View(hp.data)
```

to open a new tab showing the data.

You can also describe each column in the data set using the `summary` function:

```r
summary(hp.data)
```

For each column, a number of values are listed:

Item	Description
Min.	The smallest value in the column
1st. Qu.	The first quartile (the value 1/4 of the way along a sorted list of values)
Median	The median (the value 1/2 of the way along a sorted list of values)
Mean	The average of the column
3rd. Qu.	The third quartile (the value 3/4 of the way along a sorted list of values)
Max.	The largest value in the column

Based on these numbers, an impression of the spread of values of each variable can be obtained. In particular it is possible to see that the median house price in St. Helens by neighbourhood ranges from 65,000 GBP to 260,000 GBP and that half of the prices lie between 152,500 GBP and 210,000 GBP. Also it can be seen that since the median measured burglary rate is zero, then at least half of areas had no burglaries in the month when counts were compiled.

We can use square brackets to look at specific sections of the data frame, for example `hp.data[1,]` or `hp.data[,1]`. We can also delete columns and create new columns using the code below. Remember to use the `head()` command as we did earlier to look at the data frame.

```r
#create a new column in hp.data dataframe call counciltax, storing the value NA
hp.data$counciltax <- NA
#see what has happened
head(hp.data)

#delete a column
hp.data$counciltax <- NULL
#see what has happened
head(hp.data)
```

```r
#rename a column
colnames(hp.data)[3] <- "Price-thousands"
#see what has happened
head(hp.data)
```

```
  ID Burglary Price-thousands
1 21        0             200
2 24        7             130
3 31        0             200
4 32        0             200
5 78        6             180
6 80       19             140
```

Geographical Information

R has developed into a GIS as a result of user contributed packages, or libraries, as R refers to them. We will be using several libraries in this practical, and will load them as necessary.

If you are using your personal computer, you will need to install the R libraries, as well as loading them. To do this, run `install.packages("library_name")`.

To work with spatial data, we need to load some new *libraries*.

```r
#load libraries
library(sf)
library(tmap)
```

However, this just makes R *able* to handle geographical data, it doesn't actually load any specific data sets. To do this, we need to read in some data. We are going to use **shapefiles** - a well known GIS data format. We are going to be using LSOA (Lower layer Super Output Areas) data for St. Helens in Merseyside.

R uses working folders to store information relevant to the current project you are working on. I suggest you make a folder called **R work** somewhere sensible. Then we need to tell R where this is, so click **Session > Set Working Directory > Choose Directory...** and select the folder that you created.

As with most programs, there are multiple ways to do things. For instance, to set the working directory we could type: `setwd("M:/R work")`. Your version might well have a longer title, depending on what you called the folder. Also note that slashes are indicated with a '/' not '\'.

There is a set of shapefiles for the St. Helens neighbourhoods at the same location as the data set you read in earlier. Since several files are needed, I have bundled these together in a single zip file. You will now download this to your local folder and subsequently unzip it. This can all be done via R functions:

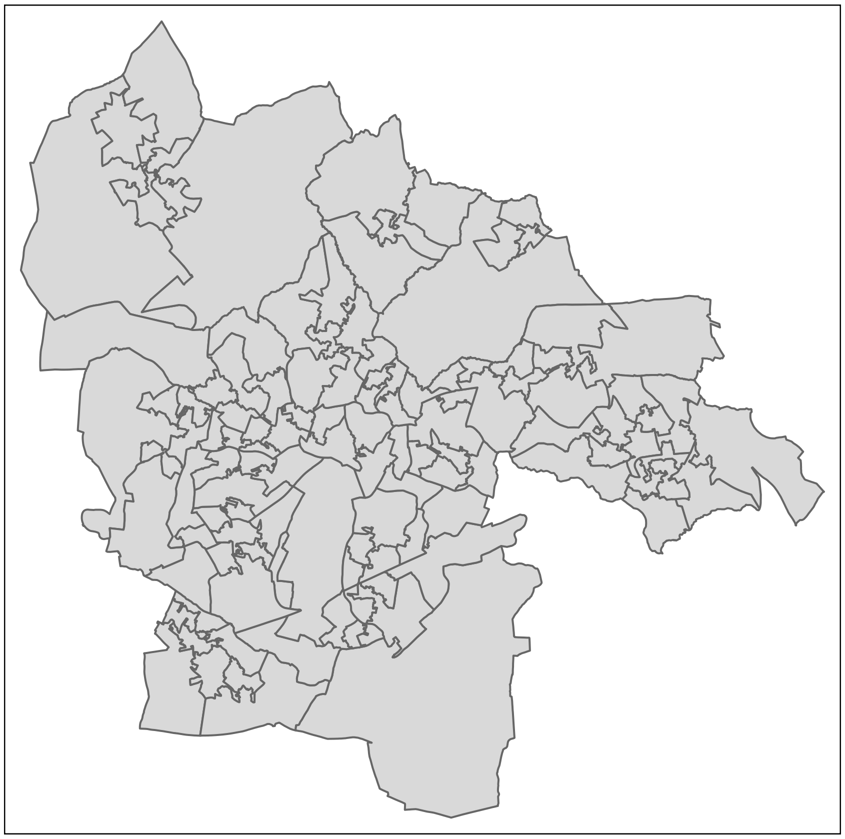
```r
download.file("http://www.nickbearman.me.uk/data/r/sthelens.zip","sthelens.zip")
unzip("sthelens.zip")
```

The first function actually downloads the zip file into your working folder, the second one unzips it. Now, we can read these into R.

```r
sthelens <- st_read("sthelens.shp")
```

The `st_read` function does this and stores them as a Simple Features (or `sf`) object. You can use the `qtm` function to draw the polygons (i.e. the map of the LSOA).

```r
qtm(sthelens)
```



We can also use the `head()` command to show the first six rows, exactly the same as with a data frame.

```r
head(sthelens)
```

```
Simple feature collection with 6 features and 5 fields
geometry type:  POLYGON
dimension:      XY
bbox:           xmin: 345350.2 ymin: 387836.6 xmax: 356013.9 ymax: 403054
projected CRS:  Transverse_Mercator
  ID  GID            NAME         LABEL  ZONECODE
1 21 2049 St. Helens 009D 04BZE01006825 E01006825
2 24 2052 St. Helens 001B 04BZE01006883 E01006883
3 31 3564 St. Helens 022A 04BZE01006898 E01006898
4 32 3565 St. Helens 001D 04BZE01006885 E01006885
5 78 4389 St. Helens 023C 04BZE01006891 E01006891
6 80 4391 St. Helens 018C 04BZE01006828 E01006828
                        geometry
1 POLYGON ((348074.3 395549.9...
2 POLYGON ((347824.8 401169.7...
3 POLYGON ((351561.1 390368, ...
4 POLYGON ((351662.3 402905.2...
5 POLYGON ((348260 391439, 34...
6 POLYGON ((348188.1 392976.1...
```

This is the same as the attribute table in programs like ArcGIS, QGIS or MapInfo. If you want to open the shapefile in QGIS or ArcGIS to have a look, feel free to.

You can see there is a lot of information there, including the geometry. The useful bit for us is the `ID` field, as this matched the ID field in the `hp.data` file. We can use this to join the two data sets together, and then show the Burglary rates on the map.

The idea is that there is a field in each data set that we can use to join the two together; in this case we have the `ID` field in `sthelens` and the `ID` field in `hp.data`.

```
sthelens <- merge(sthelens, hp.data)
```

And use the `head` function to check the data have been joined correctly.

```
head(sthelens)
```

Now that we have joined the data together, we can draw a choropleth map of these house prices.

```
#qtm
qtm(sthelens,  fill="Burglary")
```

This is a very quick way of getting a map out of R. To use the map, click on the **Export** button, and then choose **Copy to Clipboard...**. Then choose Copy Plot. If you also have Word up and running, you can then paste the map into your document. You can also save the map as an Image or PDF.

Practical 2: Making a Map with Census Data

Working with R often requires several lines of code to get an output. Rather than typing code into the **Console**, we can use a **script** instead. This allows us to go back and edit the code very easily, to correct mistakes!

Create a new script (**File > New File > R Script**) and enter the code in there. Then you can select the lines you want to run by highlighting them, and then pressing `Ctrl+Enter`, or using the **Run** button.

Now we are going to use the same principle as we used before to create a map of some data from the 2011 Census. We need to download the data, and although there are other sources of these data, in this example we will use the https://www.nomisweb.co.uk/ website.

- Navigate to https://www.nomisweb.co.uk/.
- Under **Census Statistics** click **2011 Data catalogue**.
- Select **Key Statistics (KS)**.
- Choose **KS102EW - Age Structure**.
- There are various different ways of downloading the data - explore the different options. We are going to download the table as a CSV file.
- Under **Download (.csv)**, set **Type of area** to **super output areas - lower layer 2011** from the drop down list.
- Click **Download**.
- A file called `bulk.csv` will be downloaded - save this in your working directory.

Open this file up in Excel, and you can see there are a number of different columns, covering different data. We are interested in the age data - scroll across and see the different values we have.

Add the command below to your script, and run it to read in the CSV file. The `header = TRUE` tells R to assign the first row as variable names.

```
pop2011 <- read.csv("bulk.csv", header = TRUE)
```

Then run **head** to see that the data has been read in correctly. *R will show all 23 variables, where as I've only shown the first 5 in the handout.*

```
head(pop2011)
```

```
  date       geography geography.code Rural.Urban
1 2011 Darlington 001B      E01012334       Total
2 2011 Darlington 001C      E01012335       Total
3 2011 Darlington 001D      E01012366       Total
4 2011 Darlington 001E      E01033481       Total
5 2011 Darlington 001F      E01033482       Total
6 2011 Darlington 002C      E01012323       Total
  Age..All.usual.residents..measures..Value
1                                      2466
2                                      1383
3                                      2008
4                                      1364
5                                      1621
6                                      1563
```

Some of the variable names are not displayed very clearly. We can rename the columns, so that when we run the **head** command, R lists the correct names. This will also help us refer to the columns later on.

Run the code below, which creates a new variable which contains the names (**newcolnames**) and then applies it to the `pop2011` data frame.

*It's also worth noting here that any line of code that starts with a **#** is a comment - i.e. R will ignore that line and move onto the next. I've included them here so you can see what is going on, but you don't need to type them in.*

```
#create a new variable which contains the new variable names
newcolnames <- c("AllUsualResidents","Age00to04","Age05to07",
                 "Age08to09","Age10to14","Age15","Age16to17",
                 "Age18to19","Age20to24","Age25to29",
                 "Age30to44","Age45to59","Age60to64",
                 "Age65to74","Age75to84","Age85to89",
                 "Age90andOver","MeanAge","MedianAge")

#apply these to pop2011 data frame
colnames(pop2011)[5:23] <- newcolnames
```

The final line of code (starting **colnames**) actually updates the variable names. The square brackets are used to refer to specific elements- in this case, columns 5 to 23. *For example, `pop2011[1,]` will show the first row and `pop2011[,1]` will show the first column.*

Now we have the correct column names for the data frame. It would also be good to check they have been applied to the `pop2011` dataframe correctly. **What code would you use to do this?**

```
  date       geography geography.code Rural.Urban AllUsualResidents Age00to04
1 2011 Darlington 001B      E01012334       Total              2466       161
2 2011 Darlington 001C      E01012335       Total              1383        58
```

```
3 2011 Darlington 001D      E01012366       Total               2008        72
4 2011 Darlington 001E      E01033481       Total               1364       154
5 2011 Darlington 001F      E01033482       Total               1621       153
6 2011 Darlington 002C      E01012323       Total               1563       110
  Age05to07 Age08to09 Age10to14
1       134        80       169
2        46        19        81
3        61        40        87
4        68        33        61
5        82        38        96
6        60        31       110
```

Now we have the attribute data (the number of people in each age group in each LSOA in this case) we need to join this attribute data to the spatial data. Therefore, first, we need to download the spatial data.

- Go to http://census.edina.ac.uk/ and select **Boundary Data Selector**.
- Then set **Country** to **England**, **Geography** to **Statistical Building Block**, **dates** to **2011 and later**, and click **Find**.
- Select **English Lower Layer Super Output Areas, 2011** and click **List Areas**.
- Select **Liverpool** from the list and click **Extract Boundary Data**.
- After a 5 to 20 second wait, click `BoundaryData.zip` to download the files.

Extract the files, and move all the files starting with the name `england_lsoa_2011` to your working folder. Then read in the data:

```r
#read in shapefile
LSOA <- st_read("england_lsoa_2011.shp")
```

Like earlier, we can use the `qtm()` command to preview the map. We can also look at the attribute table with `head()`. Try these both now.

The next stage is to join the attribute data to the spatial data, like we did in the exercise earlier. See if you can see how and why I have changed the code from earlier.

```r
#join attribute data to LSOA
LSOA <- merge(LSOA, pop2011, by.x="code", by.y="geography.code")
```

And use the `head` command to check it has joined correctly. Your data should contain correctly labelled Age data in the 7th to 26th columns.

```r
head(LSOA)
```

Making Maps

Now we have all the data setup, we can actually create the map. We can use the `qtm()` code like we did earlier.

We can use the `fill` parameter like we did earlier with the `Burglary` data. Try working out the code yourself to show the first set of age data.

This works well, and we can choose which variable to show. However we don't get many options with this. We can use a different function `tm_shape()`, which will give us more options.

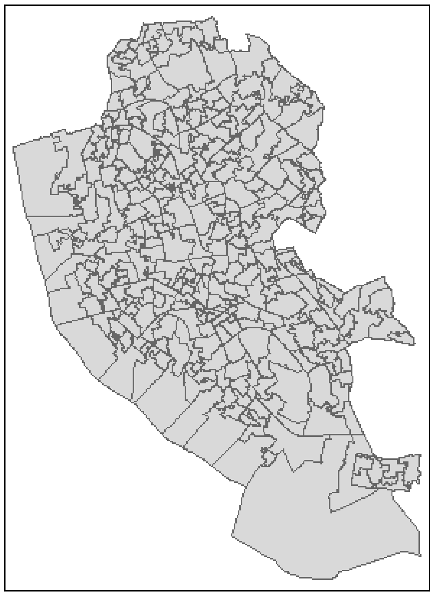
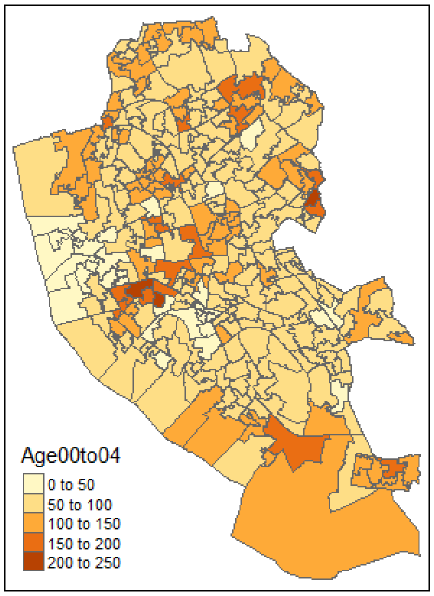



Figure 2: Output from `qtm(LSOA)` (left) and output with `tm_shape(LSOA) + tm_polygons("Age00to04")` (right).

```r
tm_shape(LSOA) +
  tm_polygons("Age00to04")
```

```r
tm_shape(LSOA) +
  tm_polygons("Age00to04", title = "Aged 0 to 4", palette = "Greens", style = "jenks") +
  tm_layout(legend.title.size = 0.8)
```

This allows us to change the title, colours and legend title size. Try substituting in `Blues` and adjusting the title.

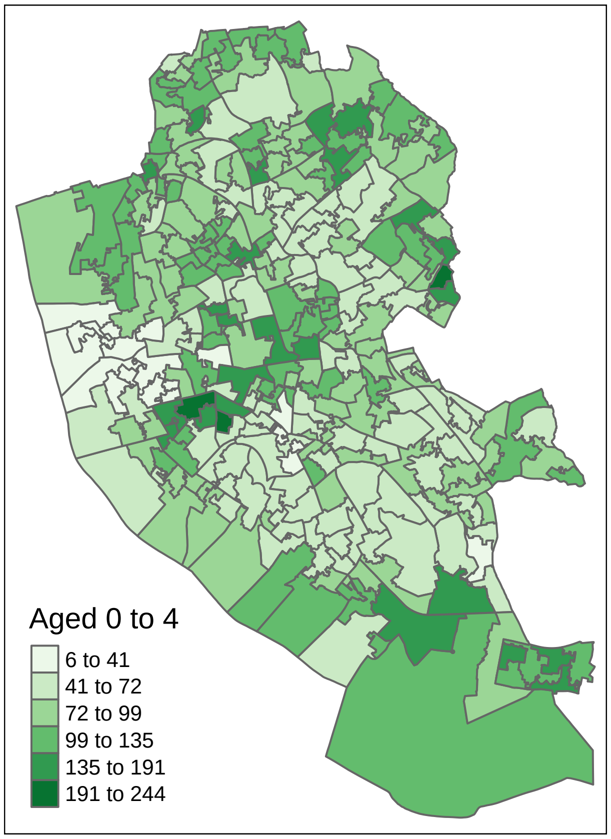
Colours and Categories

We can choose lots of different colours from ColorBrewer, and different classification methods as well. To show all of the different colour palettes we could use, run this code:

```r
#load the R Color Brewer library
  library(RColorBrewer)
#display the palette
  display.brewer.all()
```

We can also choose which classification method to use and how many classes. We can set `n = 6` to set the number of classes

```r
tm_shape(LSOA) +
  tm_polygons("Age00to04", title = "Aged 0 to 4", palette = "Greens", n = 6, style = "jenks")
```

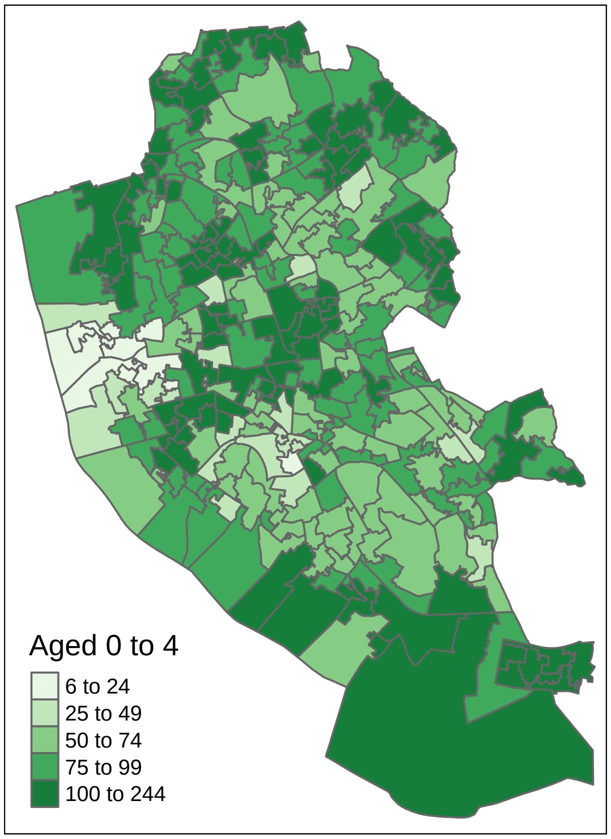


We can also set the `style` which is the classification method. Standard options are:

Classification Name	Code	Details or Example
Equal Interval	`equal`	*Regular intervals e.g. 0-5, 5-10, 10-15, 15-20*
Quantiles	`quantile`	*Split the data into 5 equal categories, with the same number of data points in each category*
Natural Breaks	`jenks`	*Algorithm based to create data driven categories*
Standard Deviation	`sd`	*Bases classes on data's standard deviation e.g. -2SD to -1SD, -1SD to 0, 0 to 1SD, 1SD to 2SD*
Fixed Breaks	`fixed`	*You choose the breaks - see below*

Fixed Breaks example:

```
tm_shape(LSOA) +
  tm_polygons("Age00to04", title = "Aged 0 to 4", palette = "Greens", n = 6, style = "fixed",
    breaks=c(6, 25, 50, 75, 100, 244))
```



Classification on a Histogram (optional exercise)

You can also show a histogram with the classification breaks using this code:

```
#select the variable
  var <- LSOA$Age00to04
#calculate the breaks
  library(classInt)
  breaks <- classIntervals(var, n = 6, style = "fisher")
#draw histogram
  hist(var)
#add breaks to histogram
  abline(v = breaks$brks, col = "red")
```

Histograms

We can also add a histogram of the data to the map:

```
tm_shape(LSOA) +
  tm_polygons("AllUsualResidents", title = "All Usual Residents", palette = "Greens",
      style = "equal", legend.hist = T)
```

Layout Options and Margins (optional exercise)

`tmap` has a huge number of options to customise the layout - have a look at https://rdrr.io/cran/tmap/man/tm_layout.html.

You may often have to adjust the margins to make your map look good. Adjusting the `inner.margins` is a good starting point. It is a vector (list) of four values specifying the bottom, left, top, and right margins, in that order. For example:

```r
tm_shape(LSOA) +
  tm_polygons("Age00to04", title = "Aged 0 to 4", palette = "Greens", style = "jenks") +
  tm_layout(legend.title.size = 0.8, inner.margins = c(.04, .03, .02, .01))
```

Scale Bar and North Arrow (optional exercise)

It is also good practice to add a scale bar and a north arrow to each of the maps you produce. Running this code will add these to the map:

```r
tm_shape(LSOA) +
  #Set colours and classification methods
  tm_polygons("AllUsualResidents", title = "All Usual Residents", palette = "Greens",
      style = "equal") +
  #Add scale bar
  tm_scale_bar(width = 0.22, position = c(0.05, 0.18)) +
  #Add compass
  tm_compass(position = c(0.3, 0.07)) +
  #Set layout details
  tm_layout(frame = F, title = "Liverpool", title.size = 2,
            title.position = c(0.7, "top"))
```

You may well need to adjust the position of the items on the map. Try Googling `"tm_scale_bar position"` for information on how to do this.

Remember that any line starting with a `#` is a comment, and will be ignored by R. Comments are very useful for us to note what the code is doing, particularly when you come back to it 6 months later and can't remember what it is supposed to do!

Interactive Maps (optional exercise)

The `tmap` library has two different viewing options - `plot` which is what we have been using so far, and `view` which provides a basemap and the ability to zoom in and out. Try this code:

```r
#set tmap to view mode
  tmap_mode("view")
#plot using qtm
  qtm(sthelens)
#plot using tm_shape
  tm_shape(LSOA) +
  #Set colours and classification methods
  tm_polygons("AllUsualResidents", title = "All Usual Residents", palette = "Greens",
      style = "equal")
#return tmap to plot mode
  tmap_mode("plot")
```

Shiny is a tool developed by RStudio we can use to create interactive maps on the web, with many different options. Have a look at https://shiny.rstudio.com/.

Exporting and Creating Multiple Maps

We can automatically save the map as a file by creating the map object as a new variable (`m`) and then save it using `tmap_save(m)`.

```r
#create map
m <- tm_shape(LSOA) +
  tm_polygons("AllUsualResidents", title = "All Usual Residents", palette = "Greens",
              style = "equal") +
  tm_scale_bar(width = 0.22, position = c(0.05, 0.18)) +
  tm_compass(position = c(0.3, 0.07)) +
  tm_layout(frame = F, title = "Liverpool", title.size = 2,
            title.position = c(0.7, "top"))
#save map
tmap_save(m)
```

Saving the map using code allows us to create multiple maps very easily. A variable (`mapvariables`) is used to list which variables should be mapped, and then the line starting `for` starts a loop. Try running the code, and then change the variables it maps.

```r
#set which variables will be mapped
  mapvariables <- c("AllUsualResidents", "Age00to04", "Age05to07")

#loop through for each map
  for (i in 1:length(mapvariables)) {
  #setup map
    m <- tm_shape(LSOA) +
      #set variable, colours and classes
      tm_polygons(mapvariables[i], palette = "Greens", style = "equal") +
      #set scale bar
      tm_scale_bar(width = 0.22, position = c(0.05, 0.18)) +
      #set compass
      tm_compass(position = c(0.3, 0.07)) +
      #set layout
      tm_layout(frame = F, title = "Liverpool", title.size = 2,
                title.position = c(0.7, "top"))
    #save map
    tmap_save(m, filename = paste0("map-",mapvariables[i],".png"))
  #end loop
  }
```

You can replace `.png` with `.jpg` or `.pdf` for different file formats.

Creating a Tidy Map Title (optional exercise)

As you may have spotted, the map and legend titles aren't great. Just using the field name does give us the information we require, but it doesn't make a good looking map.

How can we make a better map title?

Have an experiment and see if you can work it out. There is some code available (in `script-examples/script-multiple-maps-title.R`) but try not to look straight away!

Practical 3: Clustering of Crime Points

In this section we will look at some point data, and how we can analyse it in R. We need to read in the crime data, and because the crime data is just a CSV file, we need to do some processing in order to get it as spatial data in R, including some re-projecting of the data (converting the coordinate system from WGS 1984 to BNG).

```
#Read the data into a variable called crimes
crimes <- read.csv("http://nickbearman.me.uk/data/r/police-uk-2020-04-merseyside-street.csv")
```

Now the CSV data has been read in, take a quick look at it using the `head()` function. You will see that the data consists of a number of columns, each with a heading. Two of these are called *Longitude* and *Latitude* - these are the column headers that give the coordinates of each incident in the data you have just downloaded. Another is headed *Crime.Type* and tells you which type of crime occurred.

At the moment, the data is just in a data frame object - not any kind of spatial object. To create the spatial object, enter the following:

```
#create crimes data
crimes_sf <- st_as_sf(crimes, coords = c('Longitude', 'Latitude'), crs = "+init=epsg:4326")
```

We are taking the `crimes` data, telling R which columns have coordinates, and saying that they are in WGS 1984 (`+init=epsg:4326`).

If you get an `Error: invalid crs: +init=espg:4326, reason: no system list, errno: 2` *or similar, double check for typos. Typos can often sneak in to* **`epsg`***!*

Use the `head()` function to check the data have been process properly. We can also use `qtm()` to plot a map.

We also need to reproject the data, from Latitude/Longitude (`4326`), to British National Grid (`27700`):

```r
#reproject to British National Grid, from Latitude/Longitude
crimes_sf_bng <- st_transform(crimes_sf, crs = "+init=epsg:27700")
```

To see the geographical pattern of these crimes, enter:

```r
tm_shape(crimes_sf_bng) +
  tm_dots(size = 0.1, shape = 19, col = "darkred", alpha = 0.5)
```

We can see the rough outline of the Merseyside Police area, as well as the path of the River Mersey.

You can also examine the kinds of different crimes recorded.

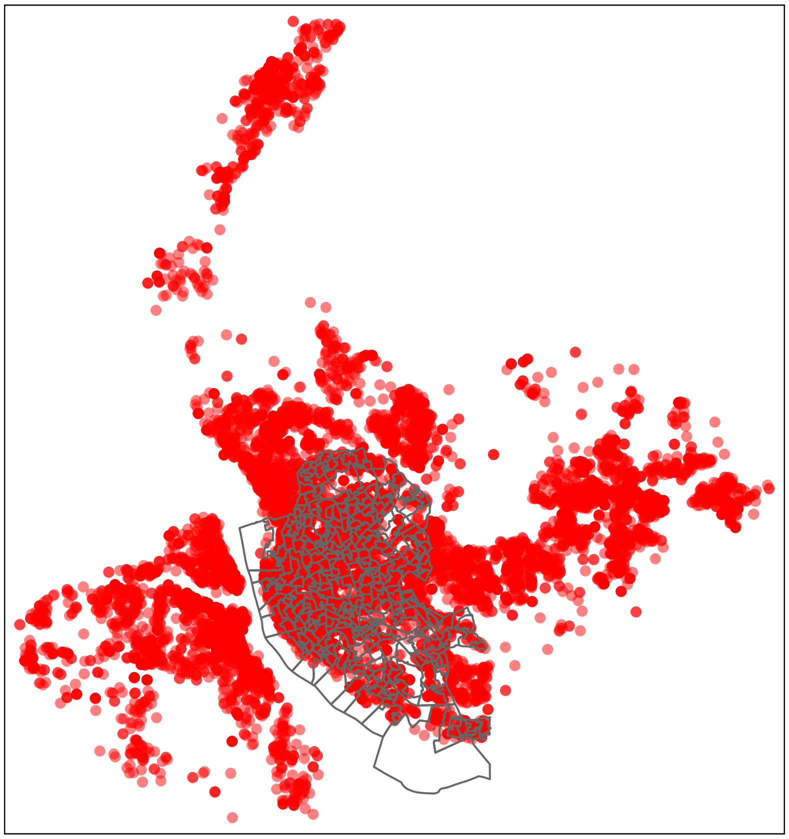
```r
table(crimes_sf_bng$Crime.type)
```

Point in Polygon

Having the spatial location of the crimes is great, but we need to combine this with other data to begin to really understand it.

To start with, we can overlay the LSOA we used earlier on the crimes. We can plot the crimes like we did earlier. What we can also do is plot the LSOA data on top.

```r
#plot the crime data
tm_shape(crimes_sf_bng) +
  tm_dots(size = 0.1, shape = 19, col = "red", alpha = 0.5)+
#add LSOA on top
tm_shape(LSOA) +
  tm_borders()
```



This doesn't really help much, as R sets the window to the first data set plotted. Try plotting the LSOA first, then the crimes layer.

This is still just two layers overlaid, but hopefully we can see what is going on now. A common GIS function is "point-in-polygon" which will allow us to count how many crimes have been reported in each LSOA.

This is something that doesn't yet work (easily) in `SF` so we need to do it in `SP`.

```
#load libraries
library(rgdal) #for readOGR()
library(GISTools) #for poly.counts()
#read shape file in in SP format
  LSOA_sp <- readOGR(".", "england_lsoa_2011")
#convert crimes SF to sp
#(see https://cran.r-project.org/web/packages/sf/vignettes/sf2.html#
#     conversion_to_other_formats:_wkt,_wkb,_sp for details)
  crimes_sp_bng <- as(crimes_sf_bng, "Spatial")
#calculate the number of crimes in each LSOA
  crimes.count <- poly.counts(crimes_sp_bng, LSOA_sp)
#add this as a new field to our LSOA data
  LSOA_sp@data$crimes.count <- crimes.count
```

The `LSOA_sp` data is now in a `Large SpatialPolygonsDataFrame` (see the list in the environment) which is from the `SP` library. This means we can't use `head()` in the same way. Instead we can use `head@data()`.

```
#show the first 6 rows (exactly the same as head(LSOA_sp) but for SP data)
  head(LSOA_sp@data)
```

```
                         label          name      code crimes.count
0 E08000012E02006934E01033755 Liverpool 062D E01033755           18
1 E08000012E02006932E01033758 Liverpool 060B E01033758           42
2 E08000012E02001356E01033759 Liverpool 010F E01033759           22
3 E08000012E02006932E01033762 Liverpool 060E E01033762           32
4 E08000012E02001396E01032505 Liverpool 050F E01032505           12
5 E08000012E02001396E01032506 Liverpool 050G E01032506           10
```

Fortunately, `tmap` can plot `SP` or `SF` data, so we can use the same code as earlier:

```
#tmap
tm_shape(crimes_sf_bng) +
  tm_dots(size = 0.1, shape = 19, col = "red", alpha = 0.5) +
#add LSOA on top
tm_shape(LSOA_sp) +
  tm_borders()
```

```
tm_shape(LSOA_sp) +
  tm_polygons("crimes.count", title = "Number of Crimes", palette = "Greens", style = "jenks")
```

Exporting Shapefiles

We used `st_read` to read in a shape file, and we can use `st_write` to export a shape file, but this only works for `SF` data, so we need to reformat it first:

```
#convert SP to SF
LSOA_sf_crime_count <- st_as_sf(LSOA_sp)
#save as shapefile
st_write(LSOA_sf_crime_count, "LSOA-crime-count.shp")
```

Calculating the Crime Rate (optional exercise)

This just tells us how many crimes there were in each LSOA - nothing about the rate. Try calculating the rate of the crimes per 10,000 people using the population data from earlier.

You can calculate the rate by calculating `(number of crimes / population) * 10000`. See the beginning of the handout for help on calculating values. Remember you will need to refer explicitly to the data frame and variable (`LSOA$crime.count`) each time you mention a column, i.e. `(LSOA@data$crime.count / LSOA$AllUsualResidents) * 10000`.

There is a "answer" script in `script-examples/script-crime-rates.R` but try not to look at it straight away!

Practical 4: Bring Your Own Data

This is the opportunity to use some of the techniques we have covered on your own data. If you don't have any data with you, you can download some sample data from the links in the training day handout.

This practical was written using R 4.0.2 (2020-06-22) and RStudio 1.3.1093 by Dr. Nick Bearman (nick@geospatialtrainingsolutions.co.uk).