Introduction to Spatial Data & Using R as a GIS

Nick Bearman - Geospatial Training Solutions & Robin Lovelace - University of Leeds

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.

- 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
```

R stores data in a data frame, which is a key type of variable in R. We can read some data in from the internet.

```
pop2011 <- read.csv("http://nickbearman.me.uk/data/r/pop2011.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(pop2011)
```

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(pop2011)
```

to open a new tab showing the data.

We can use square brackets to look at specific sections of the data frame, for example `pop2011[1,]` or `pop2011[,1]`. We can also delete columns and create new columns using the code below.

```
#create a new column in pop2011 dataframe call avgincome, storing the value NA
pop2011$avgincome <- NA
#see what has happened
head(pop2011)

#delete a column
pop2011$avgincome <- NULL
#see what has happened
head(pop2011)

#rename a column
colnames(pop2011)[1] <- "year"
#see what has happened
head(pop2011)
```

Geographical Information

R has developed into a GIS as a result of user contributed packages, or libraries, as R refers to them. To work with spatial data, we need to load some new *libraries*.

```
library(sf)
library(tmap)
```

If you get an error message, you may need to install these libaries. Run `install.packages("sf")` for the `sf` library.

This 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**.

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.

If you are using RStudio Server, then your working directory is on the server, so you don't need to do anything!

There is a set of shapefiles for Liverpool in a zip file. You can download, extract and load in these files:

```
download.file("http://www.nickbearman.me.uk/data/r/england_lsoa_2011.zip","england_lsoa_2011.zip")
unzip("england_lsoa_2011.zip")
```

```
LSOA <- st_read("england_lsoa_2011.shp")
```

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

```
plot(LSOA$geometry)
```

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

```
head(LSOA)
```

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 `code` field, as this matched the `code` field in the `pop2011` file. We can use this to join the two data sets together.

```
LSOA <- merge(LSOA, pop2011, by.x="code", by.y="code")
```

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

```
head(LSOA)
```

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

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

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

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.

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.

We can change the `fill` parameter to show different data sets. Try using it to show data from the different age catagories in the attribute table.

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:

```
library(RColorBrewer)
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

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

Histograms (optional exercise)

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)
```

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:

```
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!

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)`.

```
#create map
m <- tm_shape(LSOA) +
  tm_polygons("AllUsualResidents", title = "All Usual Residents", palette = "Greens",
```

```
              style = "equal") +
  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.

```
#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 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.

Exporting Shapefiles

We used `st_read` to read in a shape file, and we can use `st_write` to export a shape file:

```
st_write(LSOA, "LSOA-population.shp")
```

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).