Geocoding general practice addresses using R

Objective:

The aim of this activity is to provide a brief introduction to explain how data can be geocoded in R. R provides a number of geocoding utilities and we will explore one of these here, namely the geocode command in the ggmap library.

Data:

For this exercise, we will use a sample of postcodes of general practice locations in a comma separated values (.csv) file called **QOFpostcodes.csv**, originally derived from here: http://systems.digital.nhs.uk/data/ods/datadownloads/gppractice. It has two fields:

Input_row: a unique number for each postcode

Input_postocde: the postcode of the doctor's surgery (general practice).

If you are not sure what a UK postcode looks like or what it does, you may find this guide helpful: http://www.restore.ac.uk/geo-refer/resources.php particularly this section: http://www.restore.ac.uk/geo-refer/38330ctuks00y00000000.php

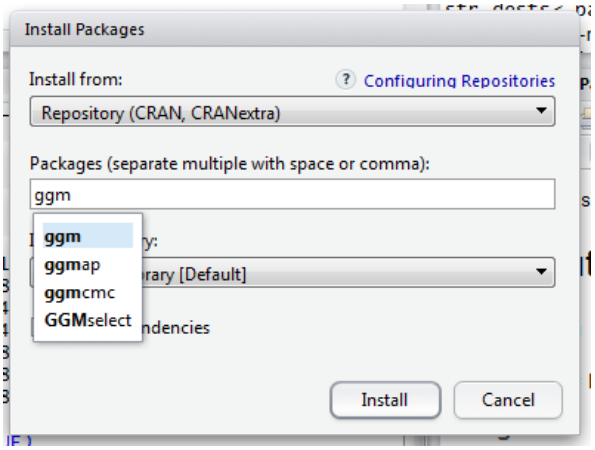
Practical Instructions:

Note: we assume that you already have a functioning copy of R Studio installed. If you do not, please see the accompanying instructions to this exercise.

Using ggmap

Let us begin by opening up the data for this exercise, which are held in a .csv file. You may wish to look at the data file in Excel before starting work with R, for example by choosing *file, open* and then next to *file name* selecting *text files.*

After starting up R Studio, we should now install the ggmap library, which contains the geocoding routine that we will use for this exercise. To do this, head for the *tools* menu and then select *install packages*. If you start typing in **ggmap**, the relevant package should appear on a drop-down list (note: the *install to* setting may look different on your machine):



Having installed the ggmap package, we now need to load it into R so that we can use all of the functionality that it contains. In the bottom-left R Console window, at the prompt, type:

```
> library(ggmap)
```

Preparing the data for geocoding

Now we need to import our comma separated values (csv) file of postcodes into a new data frame, which we can do using R's read.csv utility:

```
> QOFpostcodes <- read.csv("C:/temp/QOFpostcodes.csv", header=TRUE)
```

For newcomers to R: Note that **sample** is a data frame (essentially a table of information, consisting of columns each containing an equal number of values). Note also that we have to use a / rather than the Windows-based \ in the folder pathway to the .csv file. Note too that header=TRUE indicates that the first line of the .csv file contains names for the fields, rather than actual data. We could have used single rather than double quotations, and we could have used '=' rather than '<-' to assign the output from read.csv to sample, though most R coding style guides recommend '<-'.

We can quickly display the first few values in the resultant, imported data frame in R as follows:

```
> head(QOFpostcodes)
```

This will show the content of the first few rows in the data frame or we could type:

```
>QOFpostcodes
```

….to see everything and the many rows the data frame contains.

For our geocoding exercise, rather than the entire data frame, what we will actually need is just the second field (the one named **input_postocde** with the misspelling for added realism), which has the unit postcodes in it. We can strip this out as follows:

```
> postcodes1 <- QOFpostcodes$input_postocde
```

The $ enables us to refer to a specific field within a data frame, and we can quickly assign this to a new data frame with a single column.

Before we go further, it would be helpful to make it clear that these postcodes are in the UK. We can do this by appending an additional piece of text to each of our unit postcodes, indicating which country they lie in:

```
> postcodes2 <- paste(postcodes1, ", UK", sep="")
```

This code adds the additional text ", UK" to the end of each unit postcode string and places this in a new data frame **postcodes2**. By default, this paste utility adds an additional blank space as a separator between the postcode and the ", UK". The `sep=""` part prevents this happening by indicating that we would like no separator, rather than a blank space separator (which would be `sep="")`. We can check out the first few rows of our new data frame as we did before, to make sure this is all working:

```
> head(postcodes2)
```

Geocoding the postcodes

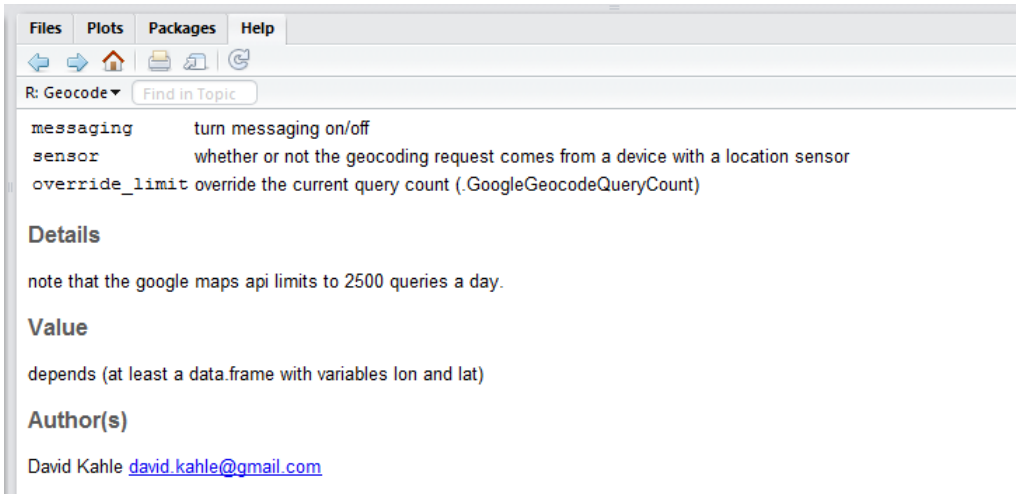
Now that we have put in all of this work, we could now try and geocode these postcodes. This code uses the geocode utility within ggmap, takes our **postcodes2** field, and outputs a new data frame called **postcode_coords**:

```
> postcode_coords <- geocode(postcodes2)
```

However, this will generate an error message. If you type:

```
> ?geocode
```

…and look in the bottom right panel on the help tab, you can see that geocode is limited to 2500 calls per day under the terms of use of the Google Maps API:



Our data file has more than 2,500 records, so we have exceeded this limit and hence we have an error message. As it will take us a while to geocode these locations, let us limit ourselves to the first 100 records only for this exercise:

```
> postcodes3 <- postcodes2[1:100]
```

The [] notation enables us to subset our data frame and choose only a certain number of records within it – in this case the first 100. Let us try again with our geocoding using the subset:

```
> postcode_coords <- geocode(postcodes3)
```

Be patient here (you may see some …. appearing on your screen as the postcodes are processed)! This time, we are within the limits on daily use for free use of the Google Maps API, so after a delay, the geocoding should be successful. Let us see what the output data frame looks like:

```
> head(postcode_coords)
```

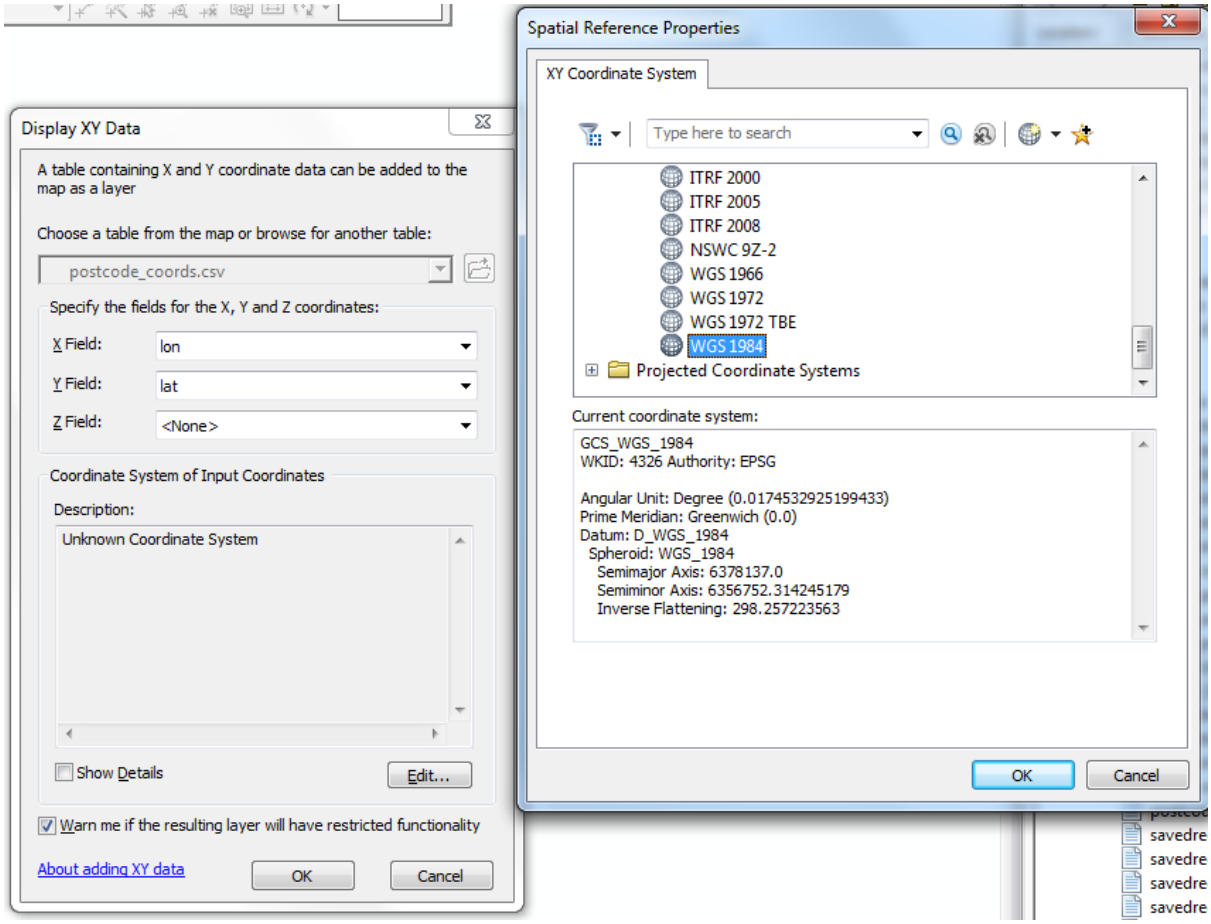
We now have plausible latitudes and longitudes for each of our input postcodes. If we want to plot these postcodes in ArcGIS, we could now export them out of R in .csv format as follows:

```
>  write.csv(postcode_coords, file="C:/temp/postcode_coords.csv")
```

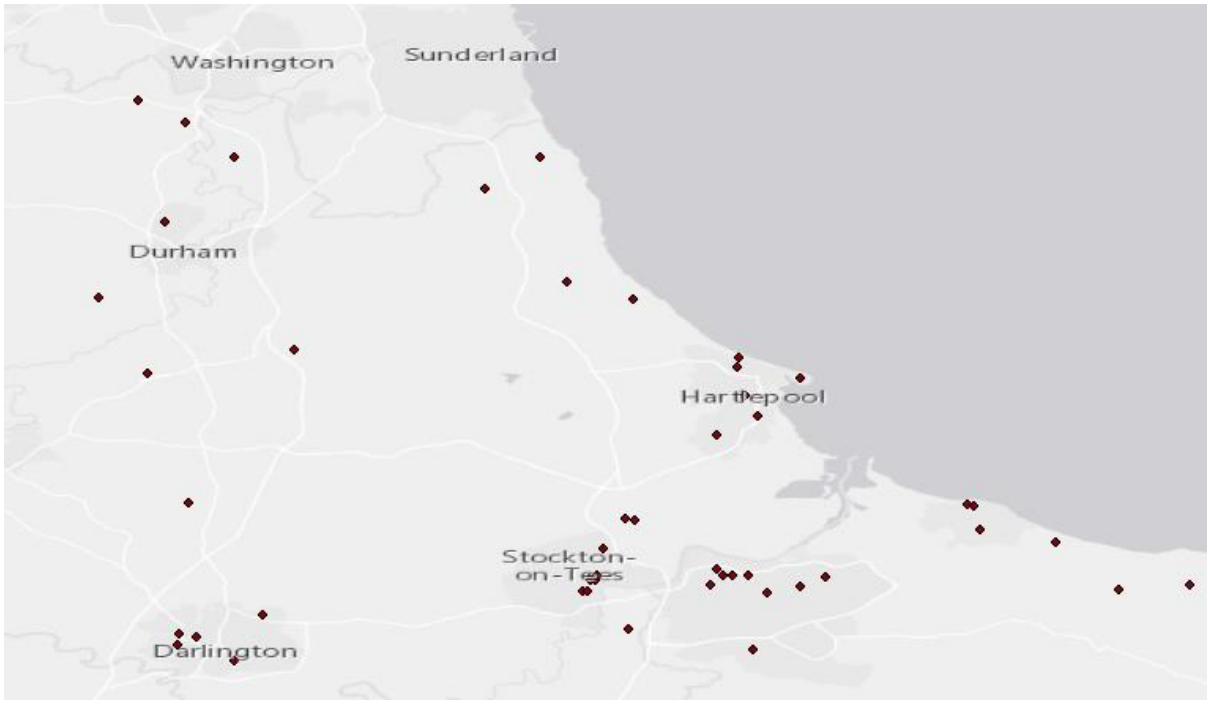
Mapping the practice locations in ArcMap

If you open up ArcMap, you should be able to open up this .csv file (for example by dragging it to the left-hand table of contents window from the right-hand 'catalog' window. If you cannot see the 'catalog' window, head for the *windows* menu and select *catalog*.

A right-click on this table in the left-hand panel, then *display xy data…* should enable you to plot these points. By default, ArcMap should realise that the longitudes in the field **lon** are x-coordinates (the x field) and the latitudes in the field **lat** are y-coordinates (the y field). As these coordinates are latitudes and longitudes in the World Geodetic System 1984 datum, you will need to document this by clicking on edit, then heading to geographic coordinate systems (meaning unprojected coordinates in latitude and longitude rather than projected coordinates in metres), then world, and finally WGS 1984:



Finally, heading for the *file* menu, then choosing *data…* and add *basemap…* will let you see where exactly these various practices are. Here are some of them:


Washington
Sunderland
Durham
Hartlepool
Stockton-
on-Tees
Darlington