GEOG0013 Statistical Analysis Worksheet 1: A gentle intro to R

Overview

This afternoon you will be exploring how *secondary data* and *statistical analysis* can be used to understand Barcelona. Our theme for this workshop is 'dwelling' and more specifically the geography of rental prices. In recent years Barcelona has become a very expensive city (e.g. http://www.catalannews.com/business/item/rent-in-catalonia-increases-by-10-3-in-2017) and there is much concern that spiraling housing costs create hardship amongst less advantaged groups or force them to move out of the city altogether. Your task today is therefore to examine (i) how rents vary across Barcelona and (ii) what types of people live in more and less expensive places.

This afternoon you will also be learning how to work with a very flexible software package called R (see https://www.r-project.org/). You will work extensively with R during your degree and once you master the basics, you will be able to use it to explore almost any data you encounter.

R is a free software environment for statistical computing and graphics. It is extremely powerful and is widely used by the academic and business communities. Unlike software such as Excel, the user has to manually type commands to get R to execute tasks such as loading in a dataset or performing a calculation. Although this seems convoluted, the biggest advantage of writing code is that you can build up a document, or script, which provides a complete record of what you have done. This makes it easy to repeat or alter things while ensuring that your work can be reproduced later.

Other reasons to use R include:

- It's intuitive and produces superb graphics and maps. It's 'intelligent' and offers in-built good practice – it tends to stick to statistical conventions and present data in sensible ways.
- It's cross-platform, customisable and extendable with a vast swathe of libraries. Libraries are basically add-ons that users contribute to do certain things.
- It is well known, respected and used at the world's largest tech companies (including Google, Microsoft and Facebook), pharmaceutical companies (including Johnson & Johnson, Merck, and Pfizer), and in many government and third sector offices.
- It offers a transferable skill showing experience of statistics and computing. Put it on your CV!

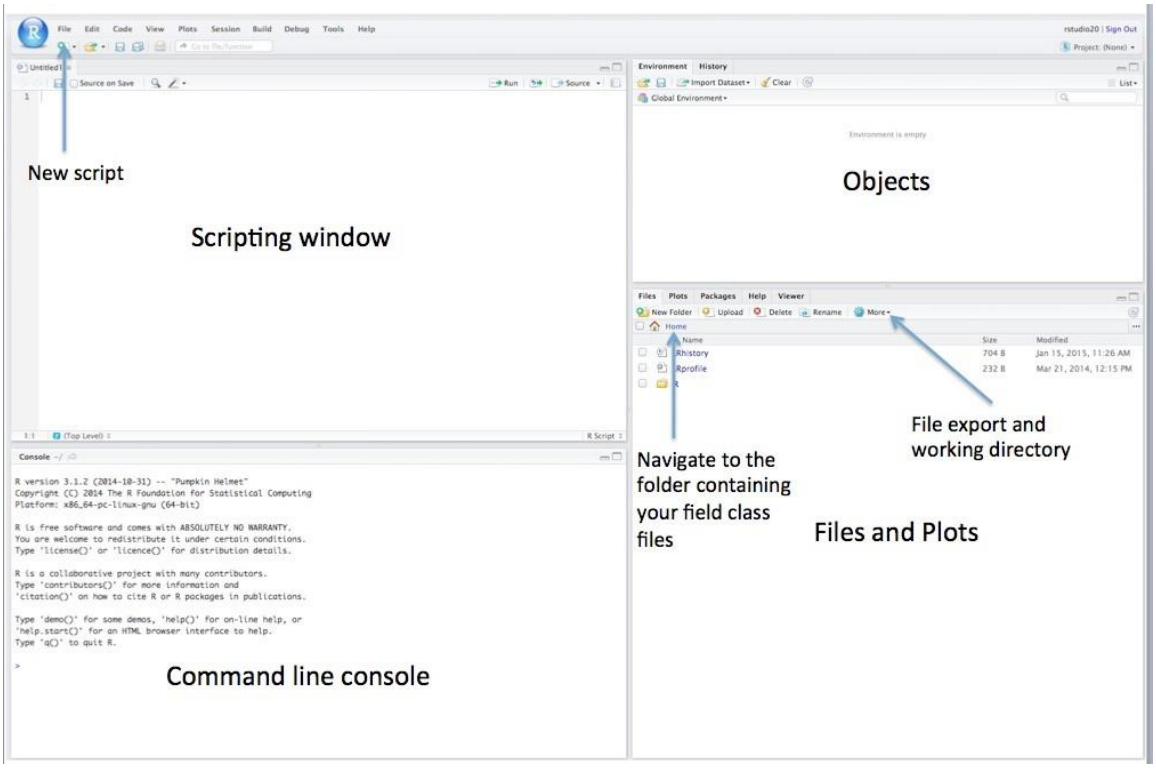
As you are not computer scientists our emphasis is on showing you how R can be used to do geographical research. Do not worry if you do not find writing code to be intrinsically interesting – treat it like learning a language or how to drive. Take your time and think through every piece of code you type as you go. The best way to learn R is to take the basic code we provide and experiment with changing parameters - such as the colour of points in a graph - to really get "under the hood" of the software. Take lots of notes as you go and if you are getting frustrated take a break!

Worksheet aims

1. Learn how to enter commands into R and create an R script
2. Create and manipulate objects
3. Calculate basic descriptive statistics

R basics

To open R click on the start menu and open RStudio. You should see a screen resembling the screenshot below. If an update prompt appears just ignore it.



Anything that appears as red in the command line (where you can write code) implies an error or warning, so you will most likely need to correct that portion of your code. If you see a > on the left it means you can type your next line into the command window, whereas + means that you haven't finished the previous line of code. + signs often appear if you don't close brackets etc.

At its absolute simplest R is a calculator. Type the following command into the command line console window. Once you have typed in the numbers you need to hit Enter to execute the code:

```
5 + 10
```

Often we want to store the results of commands we type so that we can retrieve them later. To do this we need to create R *objects* to store numbers or other things that we either type in or obtain from a command. We can create objects using the <- symbol. The <- symbol creates the object named on the left hand side of the arrow from the inputs shown on the right hand side. Try creating two objects by entering the following code:

```
a <- 5
b <- 10
```

The first line creates an object called 'a' with the value 5, while the second creates an object called 'b' with the value 10. To see what each object contains you can just type **print**(), placing the name of the object into the brackets:

```
print(a)
```

Objects can then be treated in the same way as the numbers they contain. This means you can use them in calculations. For example, to multiply a by b you type:

```
a*b
```

You can then create a new object called 'ab' to hold the result of the a x b calculation:

```
ab<- a*b

print(ab)
```

You can list all the objects that are currently active using the `ls()` command. R stores objects in your computer's RAM so they can be processed quickly. Without saving, these objects will be lost if you close R or it crashes – which can happen, particularly when using remote UCL desktop facilities…

To show the active objects in R type:

```
ls()
```

You may wish to delete an object. This can be done using `rm()` with the name of the object in the brackets. For example to delete the 'ab' object you can issue this command:

```
rm(ab)
```

Use the `ls()` command again to check that `ab` is no longer listed:

```
ls()
```

On your own now remove the 'a' and 'b' objects. This may all seem a bit dull but you will soon see how these types of simple commands can be used to explore real data from Barcelona.

More complex operations

The real power of R comes when we can begin to execute *functions* on objects. Until now our objects have been extremely simple integers (whole numbers). Now we will use functions to build up some more complex objects and create a test dataset.

In the first instance we will use the `c()` function. "`c`" means concatenate and it essentially just groups the things in the brackets together. Use the code below to create a new object called DOB that groups together the bracketed dates:

```
DOB <- c(1993,1993,1994,1991,1993)
```

Type `print(DOB)` to see the result. You can also use the `View()` function to display the DOB object in a new window if you want (close the window by clicking the x on the tab that appears). You can see that the dates are stored as an untitled column of 5 rows.

We can now execute some simple statistical functions (mean, median and range) on the DOB object:

```
mean(DOB)

median(DOB)

range(DOB)
```

Every *function* need a series of *arguments* to be passed to it in order to work. These arguments are typed within the brackets and typically comprise the name of the object (in the examples above this is DOB) that contains the data followed by some parameters. The exact parameters required are listed in each function's help files. To find the help file for the function type `?` followed by the function name, for example to pull up the helpfile for the `mean()` function you should type:

```
?mean
```

All helpfiles will have a "Usage" heading detailing the parameters required. In the case of the mean you can see it simply says `mean(x, ...)`. In function helpfiles 'x' will always refer to the object the function requires and, in the case of the `mean`, the "..." refers to some optional arguments that we don't need to worry about.

As these helpfiles can be a bit intimidating Google is always another useful way to find hints and tips. When people are having problems they often post their code online for the R community to correct. A good way to figure out R problems is therefore to paste this corrected code into your R command window and then gradually adapt it to solve your problem.

The structure of the `DOB` object - essentially a group of numbers - is known as a *vector* object in R. A vector object just looks like a column of numbers. To build more complex objects that, for example, resemble a spreadsheet with multiple columns of data, it is possible to create a class of objects known as a 'data frame'. This is probably the most commonly used class of object in R and you will shortly be importing data about Barcelona into R data frames.

We can create a data frame here by combining two vectors (two columns):

```
Singers <- c("Zayn", "Liam", "Harry", "Louis", "Niall")

One.Direction<- data.frame(Singers, DOB)
```

If you type `print(One.Direction)` you will now see the data frame called One.Direction:

```
print(One.Direction)
```

You can also visually examine One.Direction by clicking on the icons in the 'Data' section of the Global Environment Window, or alternatively by using **View**(One.Direction).

Sadly, as some (many?) of you will know, one of the singers subsequently left the band. We are therefore going to have to remove him from our table. The below code tells R to load only the second to fifth columns of the Singers and DOB objects. This removes Zayn, who is the first case in the data.

```
Singers <- Singers[2:5]

DOB <- DOB[2:5]
```

Now you can very simply run the following code again to view the updated dataframe, minus Zayn.

```
One.Direction<- data.frame(Singers, DOB)

print(One.Direction)
```

Using scripts

In the previous section R felt cumbersome. We had to enter data manually and each line of code had to be written into the command line. Fortunately this isn't normal practice! In RStudio, look to the top left corner and you will see a plus symbol, click on it and select "R Script".

This should open a blank document that looks a bit like the command line. The difference is that anything you type here can be saved as a script text file which can be re-run at a later date.

When writing a script it is important to keep notes about what each step is doing. To do this the hash # symbol is put before any code. The # symbol *comments out* that particular line so that R ignores it when the script is run. Type the following into the script:

```
#This is my first R script
myData<- data.frame(0:10, 20:30)

print(myData)
```

This code creates a new object called myData. This is achieved using the **data.frame()** function which contains two parameters: the numbers 0 to 10 in the first column, and the numbers 20 to 30 in the second (a colon tells R to count from one number to another). In the scripting window if you highlight all the code you have written and press the "Run" button on the top on the scripting window, you will see that the code is sent to the command line and the text on the line after the # is ignored.

From now on you should always type your code in the scripting window and then use the Run button to execute it. If you have an error then edit the line in the script and hit run again. Instead of using Run you can also highlight the code and hit cntrl-enter (Windows) or cmd-enter (Mac).

The columns in the myData data frame do not yet have sensible headings. You can add these by typing:

```
#Add column names
names(myData)<- c("X", "Y")

#print myData object to check names were added successfully.
print(myData)
```

In this code the **names()** function assigns the names “X” and “Y” to the first and second columns (vectors) in the myData object. On your own now try using another names function to rename these columns again to be “Alpha” and “Beta”.

That wraps up this gentle intro to R. If you finish early, try generating another data frame called ‘family’ with three vectors that contain the (1) names of some family members, their (2) DOBs and (3) their ages. Try calculating their mean and median ages and the range of years they were born in.

Hints and tips

1. R is case sensitive so you need to make sure that you capitalise everything correctly. Making a mistake with capitals is one of the most common problems people encounter.

2. The spaces between the words don't matter but the positions of the commas and brackets do. Remember, if you find the command line prompt > is replaced with +, it is because the command you issued previously is incomplete. Hit the escape (esc) key and try again.

3. It is important to come up with good names for your objects. In the case of the One.Direction

object I used a full-stop to separate the words as well as capitalization. It is good practice to keep object names short so I could have gone for OneDirection or one.dir. You cannot start an object name with a number so 1D won't work.

4. If you press the 'up' arrow on the keyboard with the command line selected you will be able to edit the previous lines of code. This helps when correcting (debugging) code.

5. The Teetor (2011) book on the reading list as well as the other R resources give lots of useful extra pointers about R and how to write particular types of function.

6. It can be quite helpful to keep a record of all the functions you have learnt as you go. Consider creating a spreadsheet or other document to do this.