Applied Research Methods

Workshop 2 - Accessing, loading and manipulating data

As with any statistical software, you need to load or enter data into R to analyse it. Data can be entered directly into R. For example, if you want to create a variable –let's call it *x* – that contains the numbers 1,2,3,4,5,6,7,8,9 you can type the following command:

```
x<-c(1,2,3,4,5,6,7,8,9)
```

The "<-" symbols instructs R to assign values to a variable. R doesn't use the "=" symbol for this purpose as the equals symbol is reserved for testing equality (e.g. does A=B?). The above command creates a variable with the values shown in parentheses. A variable is essentially just a list of values. If you type "x", R will then list the values stored in that variable:

```
>x
```

```
[1] 1 2 3 4 5 6 7 8 9
```

(this is the list of values)

For any variable, you can work with the whole set of values or you can select particular ones. One simple way of doing this would be to select the (say) the 1st or the 6th value. For our variable x, to select the 1st value, we just type:

```
> x[1]
```

```
[1] 1
```

(this is the value of value one in variable x)

For the 6th value, we type:

```
> x[6]
```

```
[1] 6
```

In our example, the 1st number is one and the 6th is 6, but usually the two will not match. If we want to know the length of a variable, we can use the *length* command:

```
> length(x)
```

```
[1] 9
```

NOTE: when we use a command like *length* we use round brackets to specify which variable we want to know the length of. However, when we query what a particular value is, we use square brackets.

You might also want to look at the first 5 values of a variable. To do this, type:

```
> x[1:5]
```

```
[1] 1 2 3 4 5
```

The text in brackets "1:5" tells R that you want to list values 1 to 5. You can vary what these numbers are and also the order. That is, you could ask for values 5 to 7, or to have the values listed from 7 to 5. While this is all a bit trivial for a variable with 9 values, this can be very helpful when you are working with large data sets.

You can also store text in variables. To do this, we just have to add single quote marks around the values to be added. For example, let's say we want to store the following types of cyber crime in a variable called "types": "Online shopping fraud", "Denial of service attack", "Dating scams", we type:

```
> types<-c('Online shopping fraud','Denial of service attack','Dating scams')
```

Again, we can check that the variable contains the data we think it does by typing its name:

```
> types
```

```
[1] "Online shopping fraud"    "Denial of service attack" "Dating scams"
```

Again we can use the length command (and so on):

```
> length(types)
```

```
[1] 3
```

The two example variables above represent different types of variable. The first is numerical, while the second is text. We can do different things with different types of variable. For example, you can calculate the average or mean value of a numeric variable, but you can't do this for a text or "string" variable. If the data you are using is not in the right format, this may result in R generating error messages. This can happen (for example) when you load data that you did not input yourself and that was misclassified (e.g. a numerical variable was accidentally stored as text).

While entering data in the way described above can be useful, it can be a little time consuming and it is much more common to obtain data that is already stored in a file. This is true for both *primary* and *secondary* data: even with primary data that you may be collecting yourself, the data may well be packaged as a file (e.g. downloaded responses to an online survey), or you may want to input the data using Microsoft excel or a similar package.

This is also the case for secondary data, which often plays an important role in research. Data held in secondary datasets could be used to complement other data you may have to make your analysis more meaningful: for instance, you could associate crime rates to the sociodemographic characteristics of the areas they relate to. In some cases, secondary sources can themselves be sufficient to explore important questions, such as when examining trends in victimisation based on data from the Crime Survey of England & Wales, or exploring terrorist events via the Global Terrorism Database. Possibilities in this area are constantly increasing as more data becomes openly available.

Once you have obtained data, you want to do something with it... As well as loading it in software, this typically means rearranging it in some way before you can get to the interesting part. This is not particularly glamorous or exciting, but it is an important and unavoidable part of the analysis process - it's frequently found that data scientists spend far more of their time doing these tasks than doing exciting modelling.

The reason for this is quite simple - real world data is often messy and ugly. Parts of it can be missing, it can be provided in hundreds of different (maybe conflicting) formats, and a convenient structure for one task may not be anything like what is needed for another. Because of this, we often need to spend time tidying or preparing data - sometimes referred to as 'munging' or 'wrangling'.

We won't spend too much time on this, but we will demonstrate part of the data acquisition process and a couple of typical data manipulation tasks, using data sources commonly used in crime research.

Inputting data in Microsoft Excel

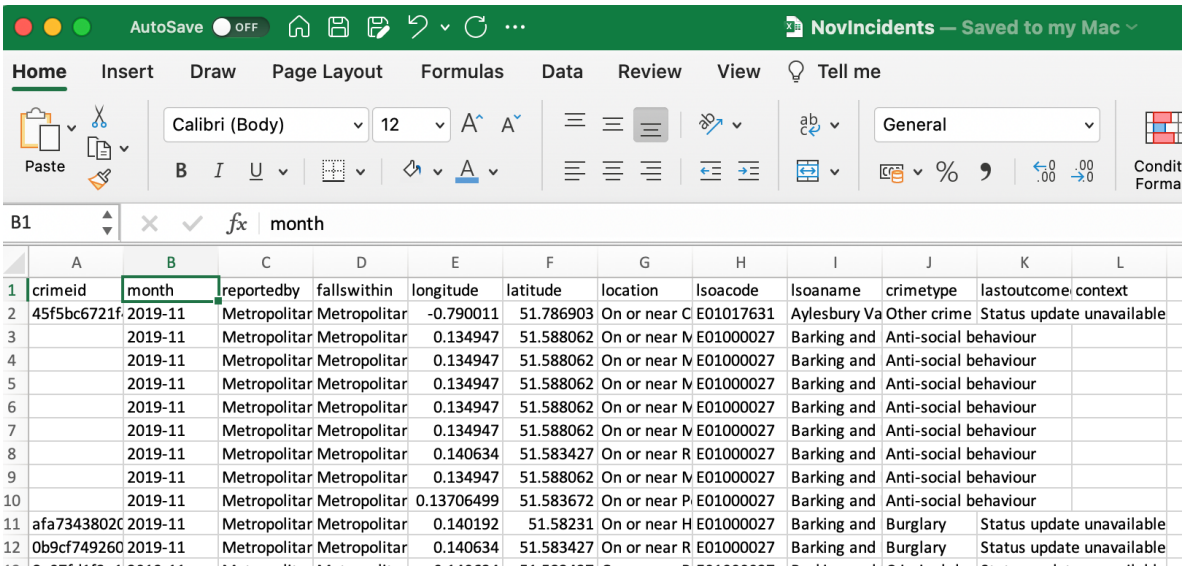
There are a variety of packages you can use to input data yourself. If you are familiar with excel, you may want to use this. Such packages are also useful for looking at existing datasets. You can examine data in R, but it's functionality more limited in this respect than other packages. If you want to create a dataset, you should organise the variables in columns. In row 1 of each column, you should add the variable name. You

then add the data beneath it. We could create a file from scratch in this way, but let's work with an example dataset of openly available crime data.

Obtaining crime data

In the UK, records of individual crime incidents have been openly available since 2011 via the data.police.uk website. The data have been through an anonymisation process to prevent the identification of victims or offenders - several characteristics are aggregated quite broadly, and locations are only approximate - and you can read about the data preparation process in detail here. While this means that it is not a *perfect* dataset, it is a great data source for understanding high-level patterns of crime.

You could download data from the website yourself, but to make things easier we have done this for you for a sample of data. The data downloaded are for the **Metropolitan Police Service** for **November 2019**. Let's start by opening the data in excel. The filename is "NovIncidents.xlsx", and it looks like the screenshot below.



	A	B	C	D	E	F	G	H	I	J	K	L
1	crimeid	month	reportedby	fallswithin	longitude	latitude	location	lsoacode	lsoaname	crimetype	lastoutcome	context
2	45f5bc6721f	2019-11	Metropolitar	Metropolitar	-0.790011	51.786903	On or near C	E01017631	Aylesbury Va	Other crime	Status update unavailable	
3		2019-11	Metropolitar	Metropolitar	0.134947	51.588062	On or near M	E01000027	Barking and	Anti-social behaviour		
4		2019-11	Metropolitar	Metropolitar	0.134947	51.588062	On or near M	E01000027	Barking and	Anti-social behaviour		
5		2019-11	Metropolitar	Metropolitar	0.134947	51.588062	On or near M	E01000027	Barking and	Anti-social behaviour		
6		2019-11	Metropolitar	Metropolitar	0.134947	51.588062	On or near M	E01000027	Barking and	Anti-social behaviour		
7		2019-11	Metropolitar	Metropolitar	0.134947	51.588062	On or near M	E01000027	Barking and	Anti-social behaviour		
8		2019-11	Metropolitar	Metropolitar	0.140634	51.583427	On or near R	E01000027	Barking and	Anti-social behaviour		
9		2019-11	Metropolitar	Metropolitar	0.134947	51.588062	On or near M	E01000027	Barking and	Anti-social behaviour		
10		2019-11	Metropolitar	Metropolitar	0.13706499	51.583672	On or near P	E01000027	Barking and	Anti-social behaviour		
11	afa7343802(	2019-11	Metropolitar	Metropolitar	0.140192	51.58231	On or near H	E01000027	Barking and	Burglary	Status update unavailable	
12	0b9cf749260	2019-11	Metropolitar	Metropolitar	0.140634	51.583427	On or near R	E01000027	Barking and	Burglary	Status update unavailable	

To explain the content, each row concerns a crime recorded by the metropolitan police. The first variable is a unique identifier, while the next specifies the month the crime occurred. The other fields provide other details about the location of the offence, the type of crime and so on. Of particular interest to us in what follows is the lsoaname. This specified the census "lower super output area" or lsoa within which the crime occurred. As a unit of geography, LSOAs have about 700 homes in them. Take a look at the data. You will see that many crimes occur in some LSOAs while only a few happen in others.

To use the data in R let's convert the file to a comma separated variable file. These types of file can easily be read by most packages. To do this, just select "Save As" in excel and save the file with the ".csv" extension (you will find this option in the File Format box, see below).

Loading crime data in R

Make sure you have set the working directory to the one in which you saved the .csv file, then import the data to R with the following command:

```
data<-read.csv("NovIncidents.csv")
```

Assuming everything is as it should be, R will load the data . However, you won't see the data as R does not have a "browser". To take a look at the first few rows of data, type:

```
head(data)
```

This will show the data in the console window. It will not be pretty but you should be able to check that the data look about right. You might also want to check that R loaded all rows of data. The excel file contained 91,204 rows of data, so let's check how many rows of data you have using the *length* command. The above command reads the file into a dataframe called "data" (you could call it something more original). This is essentially just a table of data with variable or column names. However, to use this dataframe, you have to tell R to do so. There are a number of ways of doing this. One way is to refer to the dataframe and the specific variable each time you use a variable. For our data and the variable "lsoacode", to list the values, you would type:

```
data$lsoacode
```

Here, the command is broken up into two parts. The first specifices the dataframe we are using, the second the variable of interest. The "$" separates the two. To count the number of rows of data, type:

```
length(data$lsoaname)
```

You should get the answer 91,204. This is a long-hand way of referring to variables but I often do this, particularly when I'm using a number of datasets at the same time. This is because it makes things very explicit and avoids the possibility of accidently referring to a variable with the same name in a different data set. Another approach is to use the "attach" command, as follows:

```
attach(data)
```

You can now refer to any of the variable names in the "data" dataframe using only the variable name (e.g. you could type lsoacode). This can lead to errors if you have multiple dataframes open with the same variable names so do be careful if you do this. For this exercise, let's say that we want to count how many crimes occur in each LSOA. Given that this is all we want to do, we could just get rid of the other variables. To do this, we can copy the data into a new dataframe that only contains the variable of interest. One way of doing this is by copying the data over and specifying which columns to drop. To do this, type:

```
MetNov<-data[ -c(1:7, 9:12) ] # extract lsoacodes only
```

The values in parentheses above indicate which columns we wish to drop. In this example, these are columns 1 to 7 and columns 9 to 12. The data for the variable "lsoaname" (which is column 8) are copied into a new dataframe that I've called "MetNov". Use the "head" command to take a look at the first few rows of data.

```
> head(MetNov)
   lsoacode
1 E01017631
2 E01000027
3 E01000027
4 E01000027
5 E01000027
6 E01000027
```

Aggregating data

Now we have the variable of interest, we can aggregate this to produce a new variable that provides a count of how many crimes occurred in each LSOA. To do this, type:

```
> aggdata<-aggregate(MetNov, by=list(unique.values=lsoacode), FUN=length)
```

This creates a new dataframe called "aggdata". Again, we could have called this anything. Considering the "aggregate" command, the first thing we specify in parentheses is the dataframe to be aggregated (in our case, "MetNov"), we then specify which variable we are using for the purposes of aggregation. In this case, we want to count how many crimes occurred in each LSOA. Put differently, we want to count how many rows contain each LSOA code. To do this, we specify which variable we are using for the purposes of aggregation. This is "lsoacode" and it is the "by=list" parameter that we use to specify this. The final part of the command specifies what we wish to do. In this case, we want to count the frequency with which each LSOA occurs. We specify "length" to do this, assigning this to the parameter "FUN". Having executed this command, use the "head" command to look at the data, so type:

```
> head(aggdata)
```

Which should generate the following output:

```
  unique.values lsoacode

1                    805
2     E01000001        2
3     E01000002        1
4     E01000003        2
5     E01000005        7
6     E01000006       16
```

From this, you can see that for 805 crime records, no LSOA code was provided. Two crimes occurred in LSOA `E01000001`, and so on.

You can also get a summary of the data using the "summary" command:

```
> summary(aggdata)
```

This should produce the following output:

```
unique.values         lsoacode
 Length:5043        Min.   :  1.00
 Class :character   1st Qu.:  7.00
 Mode  :character   Median : 12.00
                    Mean   : 18.09
                    3rd Qu.: 20.00
                    Max.   :887.00
```

From this we can see that the minimum count was 1, the maximum was 887 and the average (or mean) was 18.09.

Adding other data and merging datasets

Often, it is necessary to import and combine datasets. We will work through a very simple example here. The file "CensusHHLDs.xlsx" contains census LSOA data regarding the number of households in census areas across London. Open up the file in excel and take a look at it. You should see that there are 4,836 rows of data, including the variable names (of which there are only two). Save the file as a ".csv" file and open it in R. Next, use the "head" command to look at the first few rows of data.

```
census<-read.csv("CensusHHLDs.csv")
```

```
head(census)
```

We are now ready to merge the two datasets so that for each LSOA we have a count of crime and the number of households. To do this, we use the "merge" command, as follows:

```
M<-merge(aggdata, census, by.x="unique.values", by.y="lsoa")
```

As usual, we specify a new dataframe that will contain the data, in this case "M". Inside the parentheses we first specify the two dataframes we wish to merge (aggdata and census). We then specify which variable is to be used to merge the data for each dataframe. Here, we need to identify the dataframe variables that specify the LSOA codes. For the aggdata dataframe, this is the "unique.values" variable (this variable was created during the aggregation step above). For the census dataframe, the respective variable is "lsoa". Having ran the command, use the head command to take a look at the data:

```
head(M)
```

```
  unique.values lsoacode HHLDS
1     E01000001        2  1012
2     E01000002        1   976
3     E01000003        2   887
4     E01000005        7   528
5     E01000006       16   558
6     E01000007       40   627
```

Run a summary too:

```
> summary(M)
 unique.values          lsoacode          HHLDS      
 Length:4825        Min.   :  1.00   Min.   : 410.0  
 Class :character   1st Qu.:  8.00   1st Qu.: 607.0  
 Mode  :character   Median : 13.00   Median : 672.0  
                    Mean   : 18.68   Mean   : 700.7  
                    3rd Qu.: 21.00   3rd Qu.: 776.0  
                    Max.   :887.00   Max.   :1747.0
```

This indicates that there were 4825 LSOAs that appeared in *both* dataframes. The mean number of crimes is slightly different to before (18.68 versus 18.09) but this is because some of the LSOAs in the crime data were not merged due to the fact that there was no census data for them. We see that the average number of households per LSOA is 700.7.

The last thing that you might want to do after merging data is to save the file. To do this in R, use the "write" command. For our example, type:

```
write.csv(M, "MergedData.csv")
```

This should store the dataframe "M" as a .csv file in the working directory you specified. There is much more that you can do to manipulate data in R – the aim of this session was to just highlight some of the common procedures.

One final point, when you close R it will ask you if you wish to save your workspace image. If you do, this saves all of your data and functions. This can be useful, but I would advise against it. If you are working with large datasets this will use up a lot of space and can lead to errors for a variety of reasons. I find that it is better to save your R scripts and data as described previously and start from fresh each time you use R.