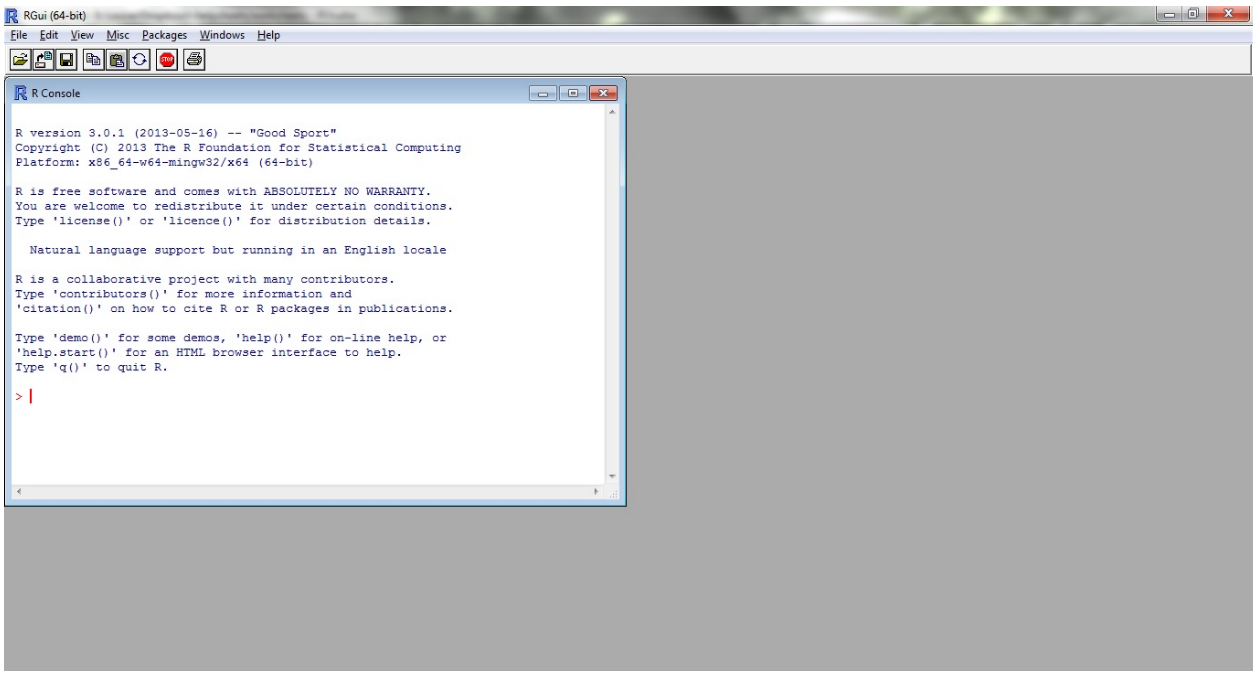
1. R Basics - How do I use R?

R is not a traditional program like Word, Excel or Chrome - instead of using the mouse to click on menus, with R you type in commands, which it then runs. Initially this might be is a bit harder to learn, but it does mean that you can easily rerun the same set of commands without having to remember which menus you have clicked. You also have a record of the work you have done, and both of these are very useful as you will see later on.

To get started, just click the R icon, and a new window (called the R console) will appear:



```
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.

  Natural language support but running in an English locale

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.

> 
```

One key concept you need to know when using R is the working directory. This is the folder where you keep your R files and any other files you happen to be using. Usually this will be set to `M:/R work` in these helpsheets.

*If you want to save files elsewhere, this is fine (just put in the appropriate file path). To find out where your current working directory is, run **`getwd()`**. To set your working directory, run **`setwd("M:/R work")`**. Make sure that the directory exists - otherwise R may start to give error messages.*

R as a calculator

At the most basic level R can be used as a calculator. Try typing the following and then press enter/return.

```
6 + 8
```

This should output:

```
[1] 14
```

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. These helpsheets will contain a mix of things you should type in (such as `6 + 8` above) and things R will output (`14` above). They will always use the same style.

R uses * for multiplication, so for example 5 times 4 is:

```
5 * 4
```

Which outputs:

```
[1] 20
```

You also have '`-`' for subtraction and '`/`' for division:

```
15 - 8
```

```
[1] 7
```

```
125/5
```

```
[1] 25
```

R also has functions like square root, sine, cosine and so on. For example:

```
sqrt(25)
```

```
[1] 5
```

These expressions can also be combined all in one line:

```
sqrt(5 + 6 * 10/4)
```

```
[1] 4.472
```

Note: It is always important to remember the order of combined sums like the one above. Remember that `5 + 6 * 10 / 4` is not the same as `(5 + 6) * 10 / 4`. Remember to put in brackets if they are required (see http://www.mathsisfun.com/operation-order-bodmas.html or http://en.wikipedia.org/wiki/Order_of_operations).

Variables

You can also assign numbers and results from calculations to variables as follows:

```
price <- 300
```

This has stored the value `300` in the variable `price`. The `<-` symbol put the value on the right into the variable on the left. It is typed with a `<` followed by a `-`.

We can then do calculations using this variable in the same way as the numbers above. For example if we wanted to reduce the `price` by 20% we could do this:

```
price - price * 0.2
```

```
[1] 240
```

Or use multiple variables in one line:

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

```
[1] 240
```

Remember in R that variables are case-sensitive (a bit like passwords). This means that `price` is not the same variable as `Price`. Try it with `discount <- Price * 0.2`. What happens? It gives you an error message like this:

```
Error:  object 'Price' not found
```

This means it can't find the object / variable `Price`.

If you ever want to check which variables are defined in your "workspace", just run `ls()` and R will print a list of the variables you have. `rm(x)` where `x` is the variable will remove the variable - note there is no undo! Try `rm(price)` which will remove the variable price.

R can also work with lists of numbers as well as individual ones. You can specify a list of numbers 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
```

Variable names can contain full stops in them, like the `house.prices` example above; they still work in the same way.

You can apply functions to the list. For example, to take the average of a list, enter:

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

```
[1] 176.9
```

If the house prices are in thousands of pounds, then this tells us that the mean house price is `176.9` thousand pounds. Note here that on your display, the answer may be displayed with more significant digits, so you may have something like `176.8571` as the mean value.

Data Frames

Data frames are an important component of R and worth spending some time on. They are like a spreadsheet, in that they can have columns of related information. We are going to create something like this:

House Price	Burglary Rate
200	0
130	7
200	0
200	0
...	...

Add the following two lists into R, by copying and pasting the code:

```
house.prices <- c(200, 130, 200, 200, 180, 140, 65, 220, 180, 200,
    210, 170, 180, 160, 180, 130, 240, 180, 170, 230, 150, 200, 200,
    210, 220, 180, 200, 210, 150, 200, 230, 120, 180, 180, 190, 72,
    80, 190, 220, 150, 200, 170, 170, 230, 200, 160, 140, 100, 140,
    170, 180, 260, 170, 230, 190, 220, 140, 220, 120, 96, 210, 170,
    180, 140, 150, 67, 200, 230, 140, 230, 83, 170, 200, 210, 240,
    180, 200, 210, 250, 140, 130, 190, 110, 160, 150, 230, 160, 210,
    200, 230, 210, 190, 120, 180, 87, 160, 190, 190, 230, 180, 110,
    200, 250, 180, 200, 130, 180, 190, 190, 230, 210, 210, 150, 190,
    210, 200, 210, 170)

burg.rates <- c(0, 7, 0, 0, 6, 19, 32, 0, 0, 0, 15, 6, 12, 8, 7, 6,
    0, 0, 6, 0, 7, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 21, 7, 12, 7, 36,
    18, 0, 0, 7, 6, 0, 0, 0, 0, 0, 13, 22, 0, 0, 0, 7, 12, 7, 5, 11,
    0, 0, 13, 13, 0, 6, 15, 6, 17, 37, 0, 6, 6, 5, 24, 0, 0, 0, 0,
    0, 0, 0, 5, 15, 0, 5, 6, 0, 0, 0, 13, 0, 6, 0, 0, 0, 23, 6, 13,
    15, 6, 0, 0, 7, 7, 0, 0, 0, 0, 19, 13, 0, 0, 0, 6, 9, 0, 0, 0,
    0, 0, 5)
```

Now, before we go any further we need to make sure that all the data have been entered into R correctly. We can see how many items are in each variable using `length(x)` where `x` is the variable name.

```
length(house.prices)
length(burg.rates)
```

You should get `118` for both. If you don't, try adding the numbers into the variables again.

For any variable, if you just type it's name (e.g. `brug.rates`) R will list all of the values contained within it:

```
  [1]   0   7   0   0   6  19  32   0   0   0  15   6  12   8   7   6   0   0   6   0
 [21]   7   0   0   0   0   0   0   0  17   0   0  21   7  12   7  36  18   0   0   7
 [41]   6   0   0   0   0   0  13  22   0   0   0   7  12   7   5  11   0   0  13  13
 [61]   0   6  15   6  17  37   0   6   6   5  24   0   0   0   0   0   0   0   5  15
 [81]   0   5   6   0   0   0  13   0   6   0   0   0  23   6  13  15   6   0   0   7
[101]   7   0   0   0   0  19  13   0   0   0   6   9   0   0   0   0   0   5
```

This command shows all of the values, and some numbers in square brackets - these relate to the position in the list of the first number of each row. For the example above, the second row begins with `[21]` which means that the first number in this row (a `7` in this case) is the 21st number in the list. The main idea is to allow you to find positions in the list of higher numbers more easily.

A handy hint to remember is that pressing up on the keyboard will get R to show the previous command you typed - handy if you want to repeat something, or make a small correction. Pressing up again will take you to further previous commands, and so on. Try this now.

You can also use the "`history()`" command, which will open a new window with the history of the commands that you have typed in R. "`history()`" will only work when you are running R on Windows - it doesn't work on OS X or Ubuntu.

We can now merge these two lists together (`house.prices` and `burg.rates`) using a data frame. You can think of it as a bit like a spreadsheet where all relevant data are stored together as a set of columns. This is similar to the data set storage in SPSS where each variable corresponds to a column and each case (or observation) corresponds to a row. However, while SPSS can only have one data set active at a time, in R you can have several of them, similar to multiple sheets in an Excel workbook. These are stored in your workspace.

To create a data frame containing the two lists enter:

```
hp.data <- data.frame(Burglary = burg.rates, Price = house.prices)
```

Then type in its name to list it:

```
hp.data
```

A little bit of explanation: *read this to understand what has just happened!*

The function `data.frame` takes all of the variables that you wish to have as columns. The `Burglary=burg.rates` creates a column in the data frame called `Burglary` containing the values in the variable `burg.rates` in the last section. Similarly, it has a column called `Price` containing the values from `house.prices`. This new data frame is called `hp.data` (an object in R is similar to a variable, although it can be more complex - so it can contain more sophisticated things like data frames, not just a list of values). Typing in the name of a data frame object (once it has been created) lists the values in the columns.

Summarising Data

With a data frame, we can use the `ncol()` and `nrow()` commands to see how many rows and columns are in the data frame. Try this now:

```
ncol(hp.data)
```

```
[1] 2
```

```
nrow(hp.data)
```

```
[1] 118
```

We can also get R to give summary values of the data frame by typing:

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

```
    Burglary         Price    
 Min.   : 0.00   Min.   : 65  
 1st Qu.: 0.00   1st Qu.:152  
 Median : 0.00   Median :185  
 Mean   : 5.64   Mean   :179  
 3rd Qu.: 7.00   3rd Qu.:210  
 Max.   :37.00   Max.   :260  
```

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 ¼ of the way along a sorted list of values)
Median	The median (the value ½ of the way along a sorted list of values)
Mean	The average of the column
3rd. Qu.	The third quartile (the value ¾ of the way along a sorted list of values)
Max.	The largest value in the column

Between 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 this area by neighborhood ranges from £65,000 to £260,000, and that half of the prices lie between £152,500 and £210,000. Also, it can be seen that since the median measured burglary rate is zero, then at least half of the areas had no burglaries in the month when counts were compiled.

Data Subsets

As we saw above, you can get a summary of a data set using `summary()`, which will give some summary statistics for the data set specified between the brackets. As well as printing out the whole data set by typing the objects name `hp.data` we can get R to output the first 6 rows using the `head()` command.

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

We can also specify a specific row and/or column using either numbers in square brackets `[]` or column/row names. For example:

```
hp.data[15, 2]
```

This prints data in the 15th row and the 2nd column, `180` in our case. You can also print using column/row names as well using quote marks:

```
hp.data[15, "Price"]
```

You can also get a range of rows, using a colon:

```
hp.data[10:15, "Price"]
```

This lists the burglary rates in rows 10-15 in the data set.

If you want to specify a full column (i.e. all of the burglary rates), just leave the part where you would write the column range empty:

```
hp.data[, "Burglary"]
```

You can use a similar approach to select a row of the data -

```
hp.data[12, ]
```

This gives Burglary and Price (i.e. house price) values for the 12th row in the dataframe.

Another way of selecting columns is to use the $ (dollar) approach.

```
hp.data$Price
```

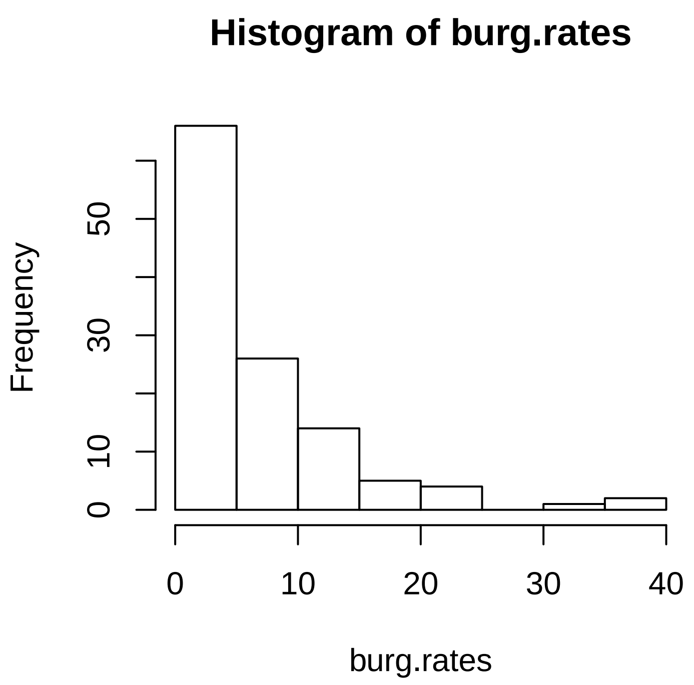
This prints the column called `Price`.

Graphics

There are a range of simple graphics that R can do to help you understand your data.

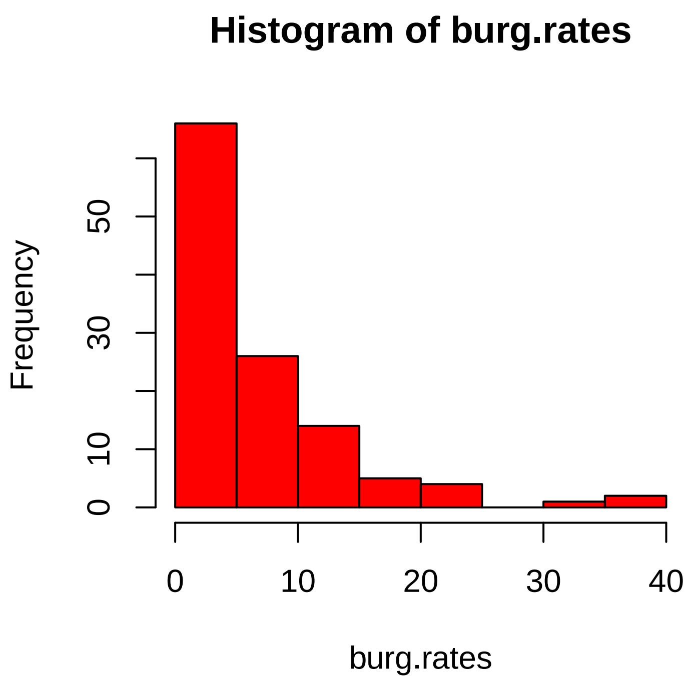
Firstly a histogram:

```
hist(burg.rates)
```



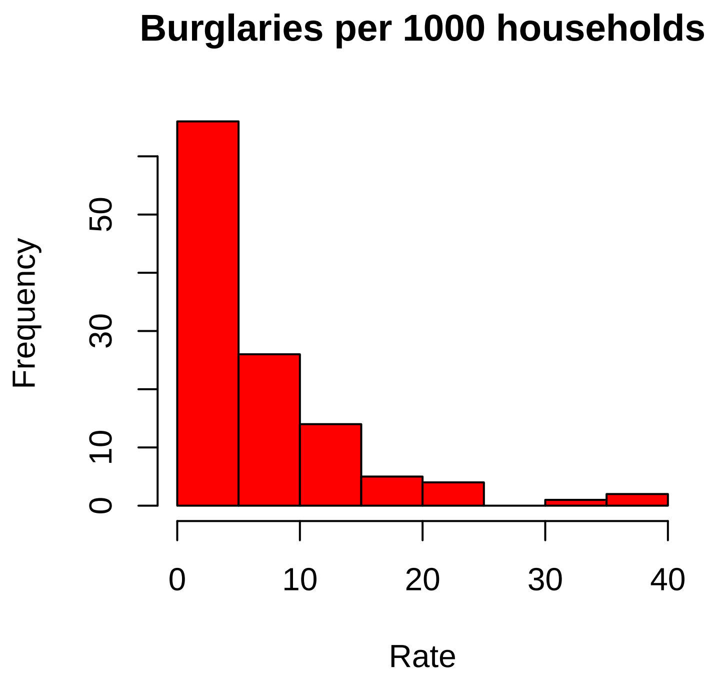
A new window will appear with the histogram in, and you can copy and paste this into Word, PowerPoint or elsewhere. R will generally give basic plots unless you tell it otherwise. To get a histogram with red bars, enter:

```
hist(burg.rates, col = "red")
```



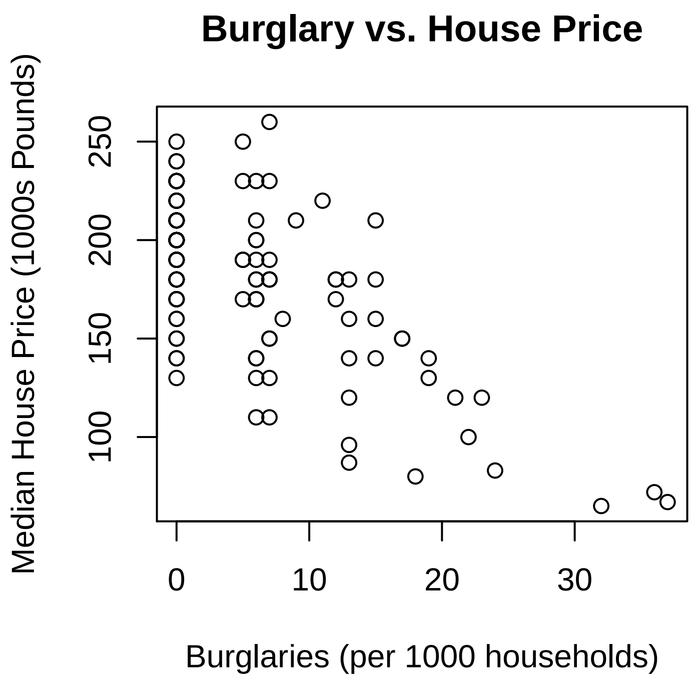
And to add a title, x-axis label (xlab) and y-axis label (ylab) use:

```
hist(burg.rates, col = "red", main = "Burglaries per 1000 households",
    xlab = "Rate", ylab = "Frequency")
```



You can also see the relationship between the two variables (median house price and burglary rates) by creating a scatter plot:

```
plot(burg.rates, house.prices, main = "Burglary vs. House Price",
    xlab = "Burglaries (per 1000 households)", ylab = "Median House Price (1000s Pounds)")
```



This shows that there is a relationship between the two quantities, although there is still a fair amount of randomness as well. The points show there is a general tendency for house prices to fall as burglary rate increases, but that there are other factors affecting house prices as well.