Lab 2

Univariate Statistics with R

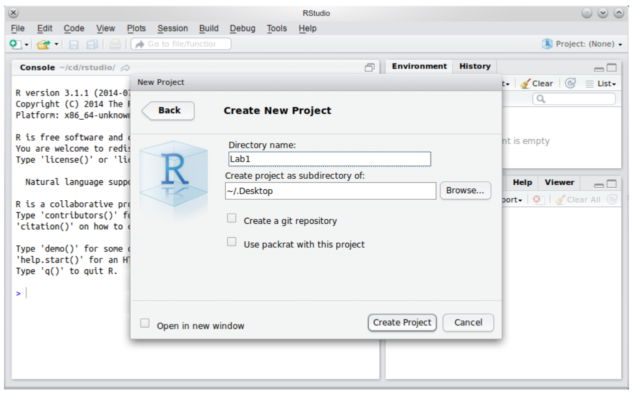
The aims of this lab are to *familiarise you with Rstudio* and to *give you a taster of* `R`.

RStudio

Task 1: Start RStudio. Select the *Project* dropdown (top right) and create a new empty project in a new directory called `Lab1` (see figure 1).

Tip: If you're not familiar with the idea of a directory (also sometimes called a folder), read Navarro, pp. 81ff., for an explanation. If you want the folder to be on your Desktop, create the project as a subdirectory of ~/Desktop (Mac) or ~/../Desktop (PC).

Pro tip: Before you do anything else, go to the *View* menu and zoom in or out to your satisfaction. RStudio will remember this setting when it next starts.



```
R version 3.1.1 (2014-07
Copyright (C) 2014 The F
Platform: x86_64-unknown

R is free software and c
You are welcome to redis
Type 'license()' or 'lic

  Natural language suppo

R is a collaborative pro
Type 'contributors()' fo
'citation()' on how to c

Type 'demo()' for some c
'help.start()' for an HT
Type 'q()' to quit R.

>
```

Directory name: Lab1

Create project as subdirectory of: ~/.Desktop

☐ Create a git repository

☐ Use packrat with this project

☐ Open in new window

Figure 1: Creating a new project in a directory called *Lab1*

Task 2: Select *File→New File→R Script*. In the window which appears, click on the disk icon just below *Untitled1* and save the (blank) script as `myscript.R`

Your RStudio window should now look like the one in figure 2 (we've added some names to the various windows for ease of reference).

Console

The 'heart of `R`' is the *Console* window. This is where instructions are sent to `R`, and its responses are given.

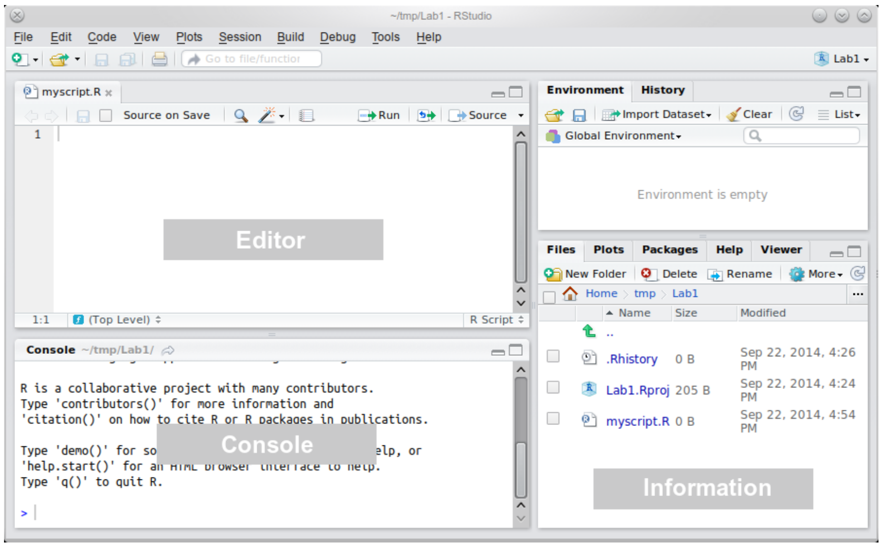
Task 3: click on the console and type `1 + 1` (then hit Enter).

You should see something like the following:

```
> 1 + 1
```

```
## [1] 2
```

The *Information* area (all of the right-hand side of Rstudio) shows you useful information about the state of your project. At the moment, you can see some relevant files in the bottom pane, and an empty 'global environment' at the top.



Name	Size	Modified
..		
.Rhistory	0 B	Sep 22, 2014, 4:26 PM
Lab1.Rproj	205 B	Sep 22, 2014, 4:24 PM
myscript.R	0 B	Sep 22, 2014, 4:54 PM

```
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 so                              elp, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 2: `myscript.R` ready to edit

Task 4: click on the console and type `x <- 1+1` and see what happens. Look at the global environment, in particular. What do you think `<-` does?

Task 5: hit ctrl-uparrow (funny-squiggly-uparrow on Mac). Highlight the `x <- 1+1` line and press Enter (this will rewrite it to the console). Edit it so that it reads `x <- x + 1`. What will this do? Press Enter to confirm your guess.

Task 6: try `y <- c(1, 2, 3, 4, 5)`. What happens? What kind of thing is `y`?

Task 7: try `y <- y * 2`. Try and guess what will happen before you press Enter.

Task 8: try `y` followed by Enter. This is one way you can get `R` to 'print out' a value on the console.

Task 9: try `plot(y, y+3)`. You should see a simple scatterplot appear bottom right, where information is shown. Note that RStudio has automatically *switched tab* to show the *Plots* tab, since you've just created a plot. Previously is showed the *Files* tab: You've already encountered the *Packages* tab last week.

Tip: click on *Zoom* to see your plot at a larger scale.

Task 10: type `?plot` into the Console. What happens? This is the generic way of getting help on any function (such as `plot()`). `R` help can range from the very useful to the quite esoteric; once you're familiar with `R`, the help gets more useful. In the meantime, the Web is often your friend.

The next exercise is a little harder: Imagine you like your scatterplot but you don't like the open circles it uses to mark each point. Instead, you want to use filled squares.

Task 11: open a web browser, and go to <www.rseek.org>. Search for `plot change points`. Using the links returned, see if you can repeat the `plot` in Task 9 but make the points come out as filled squares. If that turns out to be easy, try red filled squares. Then try each square a different colour.

Tip: there are a lot of ways to do this! The simplest is to look for a *graphical parameter* to do with plot characters and add the appropriate argument to your `plot()` command.

Editor

So far you've been entering simple instructions into the Console. But this can be a pain, for two reasons: (1) as you change commands like `plot()`, it would be nice to be able to edit what you have already typed in;

and (2) it would be good to keep a record of what you've done, so that you can do it again at a later stage, perhaps with changes.

Instead of using the console, you can use the *Editor* to type commands. The Editor creates a file which records all of the commands you type in, so you can run them again later, or build up complex commands and functions over several lines of text.

Task 12: in the Editor, type `x <- seq(-3.5, 3.5)` and hit Enter. What happens?

Task 13: click on the line you just typed in, and hit control-Enter (squiggly-Enter if you're on a Mac). What happens now?

Task 14: what does `x` become? What does `seq()` do?

Task 15: add the following lines in the Editor:

```
y <- dnorm(x)
# plot normal distribution
plot(x, y, type = 'l')
```

Task 16: click on `y <- dnorm(x)`, and hit control-Enter. Press control-Enter twice more to pass the following lines to the console. What does `#` do? What is the end result?

You might not be very satisfied with your end result (not a very curvy curve!). This is probably because we calculated values of the normal distribution (in `y`) for too few values of `x`. Maybe we can fix that...

Task 17: change the first line of your script in the editor so that it reads:

```
x <- seq(-3.5, 3.5, length.out = 49)
```

Task 18: select your entire script using the mouse, and press control-Enter. What happens? Look both at the plot and at the Global Environment above it.

Task 19: change the last line of your script so that it reads:

```
plot(x, y, type = 'l', lwd = 3, lty = 2, col = 'red')
```

Task 20: create a new plot. Do you need to run just the `plot()` line, or the whole script? Why? What does the plot look like now?

Task 21: advanced task: Using <www.rseek.org> and/or `R` help, try and edit your plot command to produce a plot similar to that in figure 3, including the title and axis labels

Task 22: save `myscript.R` (click on the disk icon).

A simple analysis

The point of this section is to get you used to playing with `R`, including loading, creating, plotting, and analysing data. What follows isn't necessarily meaningful or the 'best' way of doing things; all of the numbers are made up, and the techniques used are a bit cumbersome, but they should give a flavour of some of the things you can do. Don't worry if you're not familiar with all of the statistical concepts yet (although many of you will be); these will all be explained as the course progresses.

Task 23: open a new blank script; save it as `iq.R` (see Task 2)

Task 24: type in the following and execute the commands:

```
## compare iqs
library(foreign)
df <- read.spss('http://is.gd/iqs_spss', to.data.frame = TRUE)
```

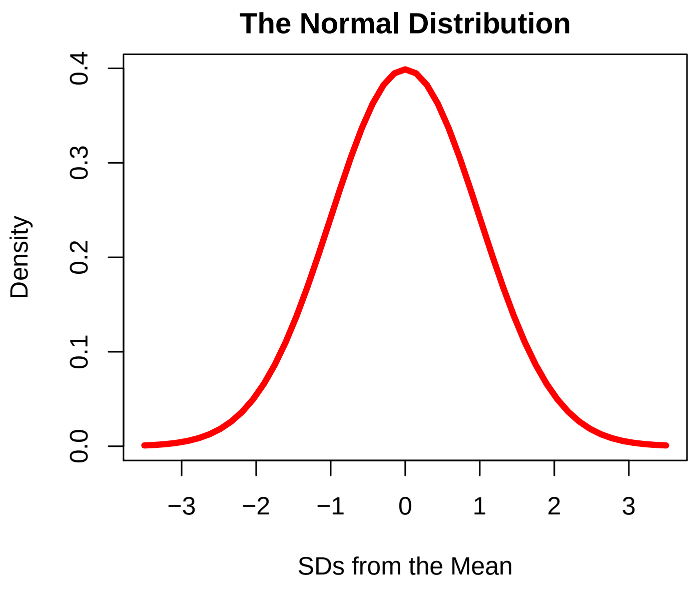


Figure 3: Normal Distribution produced using `seq()`, `dnorm()`, `plot()`.

Task 25: you should see a new variable `df` in the Global environment. Click on this. You should get something like figure 4

`df` is some data in a *data frame*, roughly equivalent to a worksheet in SPSS, or a spreadsheet. As you've seen, you can inspect it by clicking it in the Global environment, but it's not editable (through point and click); we'll get on to editing *etc.*, later in the course.

The two lines of script you've run have done a huge amount of work, in fact. They've connected to the internet and downloaded an SPSS `.sav` data file, and converted it to `R` format. If you look at the Console, you will see some (harmless) warnings reflecting the fact that the data was created in a recent version of SPSS.

The data is supposed to represent the IQs of a particular football team, Fulchester~United (11 players plus one substitute). We'd like to know whether Fulchester's mean IQ is higher than what you might expect from a totally 'average' football team, drawn indiscriminately from the general population.

Task 26: type the following in to `iq.R` (you might need to click on the tab first):

```r
# draw 12 random IQs from the `general population'
# assuming mean population IQ is 100 and SD is 15
iqs <- rnorm(12, mean = 100, sd = 15)
```

Task 27: what does `iqs` contain? (You might want to type `iqs` into the Console to see all of the values without affecting your script)

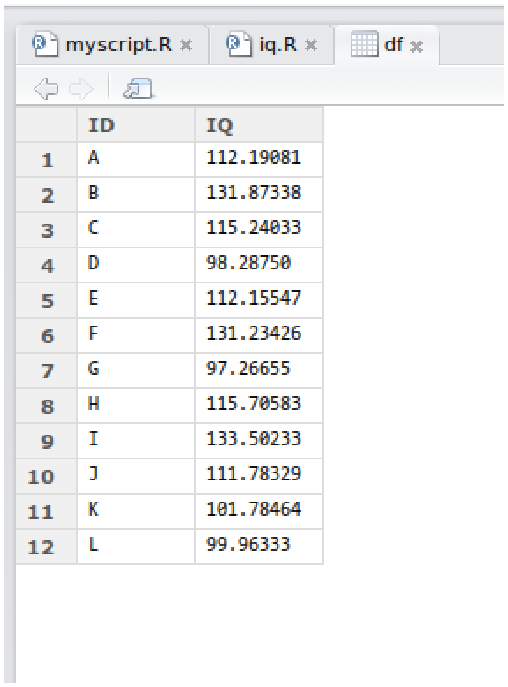
Task 28: type `summary(iqs)` into the Console for some useful further information

Task 29: look at your neighbour's Console, if they're at roughly the same stage as you. Why is their output for `summary(iqs)` different?

Task 30: now add the following lines to your script and run them

```r
# create new data frame with labels
df2 <- data.frame(ID = LETTERS[13:24], IQ = iqs)
```

Task 31: look at `df2` and see if you can work out what you've created



	ID	IQ
1	A	112.19081
2	B	131.87338
3	C	115.24033
4	D	98.28750
5	E	112.15547
6	F	131.23426
7	G	97.26655
8	H	115.70583
9	I	133.50233
10	J	111.78329
11	K	101.78464
12	L	99.96333

Figure 4: Viewing the data in `df`

Pro tip: `LETTERS[13:24]` is an example of *subscripting*. Go the console and type `LETTERS`: what do you get? On the next line, type `13:24` and see what that returns. Now run `LETTERS[13:24]`. Can you see what it's doing? Try `LETTERS[c(13, 1, 18, 20, 9, 14)]`

Task 32: type the following in to the `iq.R` script (`rbind()` binds by rows) and run it:

```
# bind the two data frames together
df <- rbind(df, df2)
```

Task 33: what has happened to `df`? (If you want to click on `df` but the tab is already open, close the tab first)

Note that `df` contains all of the IQs in one column. We need to add another column to *index* which sample the IQs come from (called a *factor* in `R`).

Task 34: add the following to your script and run it:

```
# add a factor
df$source <- gl(2, 12, labels = c('Fulchester', 'Random'))
```

`gl()` stands for 'generate levels' (of a factor). The first number gives the number of levels; the second, how many times each level is repeated; the `labels =` argument allows us to call the levels something more useful than `c(1, 2)` (which would be the default).

Task 35: look at `df` again (you might need to close the tab and reopen it). What have you just added?

It's time to do some statistics! This is a very simple test, and the results will vary depending on the random sample you created earlier, but let's compare the Fulchester sample with our random sample:

Task 36: add the following to your script and run it:

Tip: `IQ ~ source` can be read as "`IQ` is predicted by `source`". There are other ways of specifying a t-test but this way makes it clear what you are expecting to affect what

```r
# run a t-test
t.test(IQ ~ source, data = df)
```

Task 37: the output of the t-test should be in the Console below. What can you say about your totally fake data?

Last Things

Task 38: exit the project (*Lab1*→*Close Project*, save anything when asked)

You should get an empty RStudio. You can create a different project for different work (using the *Project* dropdown), or you can re-open *Lab1* and you should be exactly where you were before.

When you restart RStudio next time, you can go straight to the *Projects* tab to open an existing project (RStudio will remember where the relevant files are), or you can create a brand new project in a new directory (and switch between projects at will at any time).

That's it for today! Feedback on this, and whether you feel you learnt anything, very welcome.