Introduction to R

R is a free statistics and graphics software that has become popular in a variety of academic fields. Users can contribute to R functions and it is at the forefront of many statistical fields. There are a good number of blogs and help lists to guide you through the R-jungle.

For example:

R-bloggers	http://www.r-bloggers.com
Talk stats	www.talkstats.com
Stack overflow	www.stackoverflow.com

This lab is aimed to introduce you to R. If you are already used to working in R, just have a go through this exercise and make sure that everything seems familiar.

Open R

R is available on all the university computers. If you are using your own laptop R can be freely downloaded from http://www.r-project.org. There are now easy options available for data import supplied in a user friendly extension platform for R – Rstudio, which can be downloaded from: http://rstudio.org/ . This extension is described further below.

When working in R command line, it is highly recommended to use a text editor to save and manage your code.

Explore R

To explore what R can do, start with the command

```
plot(rnorm(100))
```

which causes a graphics window to open with 100 random normal numbers plotted along the *y*-axis (with the numbering 1 to 100 of the successive random numbers along the *x*-axis).

R can be used as a calculator, for instance

```
2 + 2

[1] 4
```

produces the sum of the two numbers and

```
exp(-2)

[1] 0.1353353
```

computes an exponential. The `[1]` in front of the result is R's way of printing numbers and vectors (a vector is just a sequence of some particular length); a number is regarded as a vector of length 1. When printing a vector, R gives the number of the first element on the printed line. For instance, we can make a vector `x` of length 25, containing the numbers 1 to 25, as follows

```
x <- 1:25
```

If we display the vector, we might get (depending on the width of the R console window)

```
x

[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
20

[21] 21 22 23 24 25
```

The construct `1:25` is used in R to denote the sequence (vector) of numbers from 1 to 25. `<-` means “assign to”. Here: assign a vector of 1 to 25 to `x`.

Vectorized arithmetic

A convenient feature of R is that it handles data vectors as single objects. We can construct a data vector like this

```
weight <- c(60, 72, 57, 90, 95, 72)

weight

[1] 60 72 57 90 95 72
```

which might represent body weights. The construct `c(...)` is used to define vectors; the letter `c` stands for "concatenate"). It can be used to enter very small amounts of data into R (it is often better to read data from a file; se below how this is done). Suppose we also have heights the correspond to the weights

```
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
```

We could then compute a data vector of body mass indices

```
bmi <- weight/height^2

bmi

[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

Notice that the operation is carried out element-wise and the operator `^` is used to raise a value to a power. For instance, the last of the `bmi` values is computed as

```
weight[6]/(height[6]*height[6])

[1] 19.7363
```

where [ ] functions as an index to the given vector or data frame. [6] is the 6th value in that vector. Many things that we might want to compute from data vectors are already available as functions in R. For instance, the mean, variance and standard deviation of the body weights are

```
mean(weight)

[1] 74.33333

var(weight)

[1] 237.8667

sd(weight)
```

```
[1] 15.42293
```

The variance and standard deviation computed by these functions are those appropriate for a sample (i.e. "using *n*-1 in the denominator"). To get help on a function like `var`, we can write

```
?var
```

which causes a new window to open (it might take some time to get used to the terse and somewhat technical style of the help texts).

Working in R: Working directory and work space

The working directory is where R will get and save files. The workspace is everything you have loaded into R, such as data frames or functions.

To import a file into the R system, first make sure that the current directory for R is set to where the data files are located (in Windows, use the File menu (then *change dir*) of the R console program). On Macs you can simply pull the folder you want as a working directory to the R symbol on your dock, or go into *Misc* > *Change working directory*. To see where your working directory is write:

```
getwd()
```

You can also change it manually by typing:

```
setwd(your/path/here)
```

You can see what is in your workspace by typing:

```
ls()
```

You can remove items by specifying:

```
rm(x)
```

Check your workspace again to see that x has been removed. To remove all items in your workspace type:

```
rm(list=ls())
```

Reading data into R

One of the most common question from people starting to use R is how they can read data from an Excel file into the R system. First export the data from an Excel sheet to some form of text file, and then import the text file into R. Start by saving an Excel sheet either as a tab delimited text file (.txt) or as a comma delimited text file (.csv), using "Save As" from the File menu in Excel. Either of these work fine; your teacher typically uses the tab delimited option. An important issue in this context is the kind of symbol that is used by the operating system when displaying decimal numbers (this is set in the Control Panel in Windows and in the System Preferences in OS X). For international collaboration, it is an advantage to use decimal points instead of decimal commas, but operating systems are often "localized" to use the standard symbol in the country; either of them work fine in principle. However, you have to know which you have when you import the data into R. When you have made sure you have the correct working directory to reach the files you can read the text data file into R using a command like

```
dat <- read.delim("BodySize.txt")
```

which constructs a new data frame `dat` with the data from the file. `read.delim` is a short cut to read text files that are tab delimited, use decimal points and has headers for the variables. For a tab delimited file with decimal commas, you should use the command

```
 dat <- read.delim2("BodySize.txt")
```

instead (you can use the command `?read.delim` to get some info on the various possibilities for reading a spreadsheet-like text file into R).

You can use the command

```
fix(dat)
```

to look at the imported data (note we don't recommend this!).

You can use the command

```
str(dat)
```

to look at the structure of the dataframe.

Finally, note the professions were recorded as text strings (`biol` and `stat`) in the data files. When R imports this kind of data, it is automatically made into a factor. The verify this, write

```
  > dat$Profession

  [1] biol stat biol stat biol stat

  Levels: biol stat
```

So, if we use a command like

```
  > plot(dat$Profession, dat$Height)
```

R makes a classic box and whisker plot instead of a scatter plot. To see another plot of this type, load the R source file provided by your teacher

```
  > source("PlotGroups.R")
```

and give the command

```
  > plot.groups.se(dat$Height, dat$Profession)
```

to display means and standard errors for the groups. Which of the two professions is taller on average?

Another example

Read the data in **RepWeightHeight.txt** into R, using the command

```
  > dat2 <- read.delim("RepWeightHeight.txt")
```

(If you use the previous data frame name `dat`, the previous content will be overwritten.) The data consists of self-reported and measured weights and heights of male and female participants in an exercise program. You can use the command `head(dat)` to look at the first five rows of the data, or `head(dat,12)` to look at the first twelve rows. There are five variables in the data frame, `sex` (the sex of participants, coded as `M` and `F`), `weight` and `repwt` (actual and self-reported weight in kg), and `height` and `repht` (actual and self-reported height in cm). Because the sexes were coded as text in the imported data file, R constructs `sex` as a factor with `F` and `M` as levels (the factor levels are arranged in alphabetical order by default). To get an overview of the datafile, you may also try:

```
> str(dat)
```

Let us now look at a plot of the data. First load the R source file provided by your teacher

```
source("PlotAncova.R")
```

and give the command

```
plot.ancova.sep(dat$repwt, dat$weight, dat$sex)
```

to display a plot with data points and separate regression lines for the two sexes (this kind of analysis will come later in the course). Repeat the same with the heights.

Using R studio

R-studio integrates the standard console with a graphical interface for some of the common operations as loading data, saving and restoring the workspace (your saved session with data, commands and plots etc.) and a nice overview of your current session. Download: http://rstudio.org/