An Introduction to R

This document is designed to get you started with using `R`

We will learn about

- what `R` is and its advantages over other statistics packages
- the basics of `R`
- plotting data and graphs

What is R ?

`R` is one of many computer programs designed to carry out statistical analyses.

Some of the advantages of `R` are

- It's free!
- It is developed by an international team of statistical computing experts.
- It is becoming the computer program of choice for statistical research.
- All standard statistical analyses are implemented
- It is actually a complete programming language and so we can extend it in any way we choose.
- It allows us to produce publication quality graphics.
- Add-on packages are available in a diverse range of specialized fields, e.g. Micro-array analysis, brain imaging etc.
- There is an extensive help system and an active email help list.
- It is available on Windows, Linux, Unix and Macintosh operating systems.

Books

In addition to the extensive documentation and help system that is included in `R` there are two main books that we recommend.

*`Introductory Statistics with* `R` *' by Peter Dalgaard, ISBN 0-387-95475-9*

- a very good introduction to `R` that includes many biostatistical examples and covers most of the basic statistics covered in this course.

Downloading and Installing R

You will also need to download and install R from this website

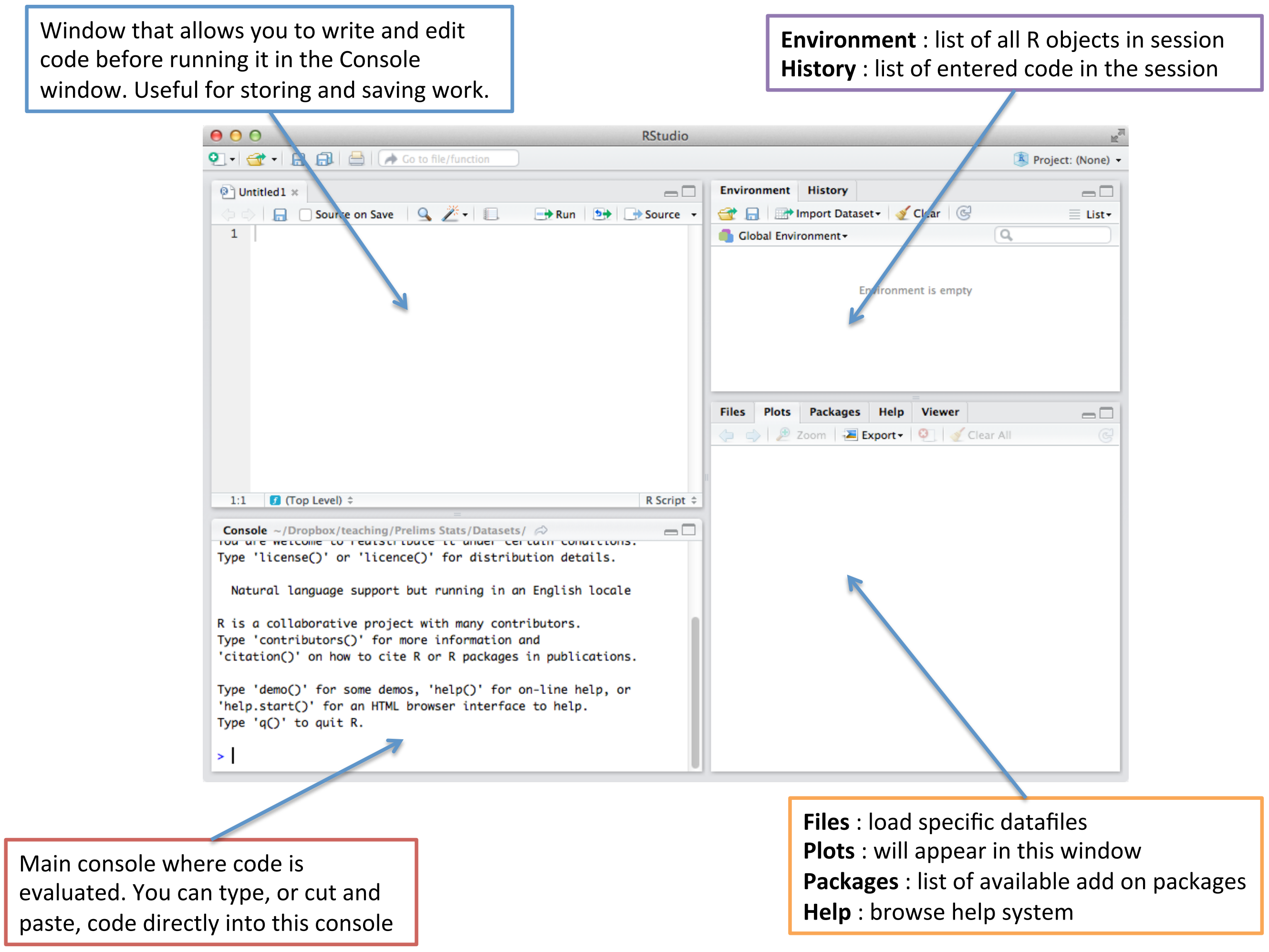
https://cran.rstudio.com/

We recommend that you use R Studio to work with R. This is an Integrated Development Environment (IDE) that allows you to enter code, read in datasets, store your work and make plots, all within a single environment.

You can download R Studio from this webpage for Windows, Mac or Linux

https://www.rstudio.com/products/rstudio/download/

The R Studio console looks like this.



Window that allows you to write and edit code before running it in the Console window. Useful for storing and saving work.

Environment : list of all R objects in session
History : list of entered code in the session

```
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.

> 
```

Main console where code is evaluated. You can type, or cut and paste, code directly into this console

Files : load specific datafiles
Plots : will appear in this window
Packages : list of available add on packages
Help : browse help system

Your first R session

R works by a question and answer model: you enter a command, press Enter.

R carries out the command and prints the results to the screen if required.

Q. For example, if we want to know the answer of 2 + 2 we would simply enter 2 + 2 into the Console and press Enter. Try this

R Basics

Q. R can be thought of as an overgrown calculator and as such we do all of things we can do on a standard calculator. Try typing in the following expressions

Multiplication
```
5 * 4
```

Exponentiation
```
exp(-2)
```

Square root
```
sqrt(12)
```

Raising to a power
```
3^5
```

Often we will want to store the results of a command. To do so we `assign' the result to a `variable' with a name of our choice.

Q. Type in the following command and press Enter.

```
a = (3 * 7) + 1
```

Q. The above command has stored the result in a variable called a. To examine the value stored in a variable we simply enter its name. Enter the command

```
a
```

Q. Variables can be manipulated in any way we wish. For example, to raise the value stored in the variable a to the power 3 enter the command

```
a^3
```

Q. Calculate 41 * 13 – 67 and store the answer as a variable called x. Also, calculate 12^3 - e^7 and store the answer in a variable called y. Calculate y – x (**ans.** 165.3668).

Vectors

Often we need to work with vectors of numbers. We can assign vectors in several ways.

Q. A vectors of consecutive integers can be assigned using the command like

```
v1 = 11:15
```

Q. Look at the vector by entering its name

```
v1
```

Q. More generally, sequences with set intervals can be assigned using the seq() function. For example, to create a vector of numbers starting with 11 and increasing to 15 in units of 0.5 enter the command

```
v2 = seq(11, 15, by = 0.5)
```

Q. Look at the result by entering its name

```
v2
```

Q. Alternatively, we can assign the vector directly using the command

```
v3 = c(11, 2, 73, 24, 35)
```

Q. Look at the result

Q. We can manipulate vectors as if they were numbers. For example, if we want to square all the elements of the vector and add 6 we could use

```
v3^2 + 6
```

Q. If we want to count how many elements of the vector are greater than 13.5 we can use

```
sum(v3 > 13.5)
```

Q. Specific elements or subsets of elements of the vector can be extracted. For example we can extract the 4th element of vector v3 using

```
v3[4]
```

Q. We can extract the first three elements of v3 using

```
v3[1:3]
```

Q. If we want elements 1, 2 and 5 we can use

```
v3[c(1, 2, 5)]
```

Q. Vectors can also contain strings

```
v4 = c("tree", "apple", "pear", "ball", "sky")
```

Q. Look at the result

Q. Create a vector (called x1) of integers that increases from 1 to 19 in units of 2. Square all the elements in the vector and subtract 100 and store the answer in another vector (called x2). Use an R command to calculate how many elements of x2 are greater or equal to zero. (**ans.** 5)

Matrices

Q. Matrices can be constructed from vectors using the matrix() function. Type in the commands and look at the result

```
v1 = 1:12
m1 = matrix(v1, nrow = 3, ncol = 4, byrow = TRUE)
```

You should have got

```
> m1
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Q. In the above command we set the argument byrow = TRUE which specifies that the elements of the vector v1 are to be placed into the matrix starting with the first row, then the second row etc. The default value of this argument is FALSE, thus if we simply omit this part of the command we get a matrix which has been filled by column. Type in the command to try this

```
m2 = matrix(v1, nrow = 3, ncol = 4)
```

Q. Like vectors, matrices can be manipulated as if they were variables. To square all the elements and add 3 enter the command and check this is what has happened

```
m1^2 + 3
```

Q. We can also extract elements in a similar way to vectors. For example, to extract the element of m2 in the 2nd row and 3rd column we can use

```
m1[2, 3]
```

Q. You can transpose matrices using the t() operator. For example,

```
t(m1)
```

Q. Create a 10x10 matrix of integers from 1 to 100, filled one row at a time. Square all the elements of the matrix, subtract 400 and extract the element in the 5th row and 8th column. (**ans.** 1904)

Q. You can multiply matrices together using the %*% operator. For example,

```
m1 %*% t(m1)
```

Lists

Often we would like to store data of several different types and sizes in one object. This can be achieved using a list.

Q. To create a list that include the vectors v1 and v4 and the matrix m1 enter the command and look at the result

```
list1 = list(v1 = v1, v4 = v4, m1 = m1)
```

Q. Components of a list can be accessed by name using the $ symbol. For example, the component v1 can be extracted by entering the command

```
list1$v1
```

Q. To access the 2nd component of the list (without reference to its name) enter the command

```
list1[[2]]
```

Obtaining a list of the stored objects

Q. The variables, vectors, matrices and lists we have assigned so far will be stored until the end of the `R` session. We can look at which objects we have created using the function ls(). Type in the command

```
ls()
```

Or you can use the Environment window in the top right of the R Studio window to browse the different objects.

Functions

So far we have used the `functions' exp, sqrt, seq, c, matrix, and ls. The statistical functionality of `R` is accessed through functions so we need to be familiar with their use. The format of a function call is the function name followed by a set of parentheses containing one or more arguments. The ? symbol can be used to get a description of any given function.

Q. Enter the following command to produce a description of matrix() in the Help window on the bottom right of R Studio

```
?matrix
```

The most precise way to specify the arguments to a function is by name.

Q. For example, enter the following command to create a 2x2 matrix of integers from 1 to 4, filled one row at a time. Look at the result to make sure this is what you get

```
matrix(data = 1:4, nrow = 2, ncol = 2, byrow = FALSE)
```

Q. What happens if you change the argument byrow from FALSE to TRUE?

Q. Alternatively arguments can be specified by position if we know the form of the function. For example, create a matrix equivalent to the first one above by entering

```
matrix(1:4, 2, 2)
```

Many function arguments have sensible default settings and thus can be ignored in standard function calls. For example, the second call to matrix() above did not include the byrow argument as the default is FALSE.

Often we will want to do something R cannot do directly so we can write our own functions. For example, we might want to evaluate the quadratic x2 - 2x + 4 many times so we can write a function that evaluates the quadratic for a specific value of x.

Q. To do this enter the command

```
my.f = function(x) { x^2 - 2*x + 4}
```

Q. Calculate the value of the function at x = 3 using the command

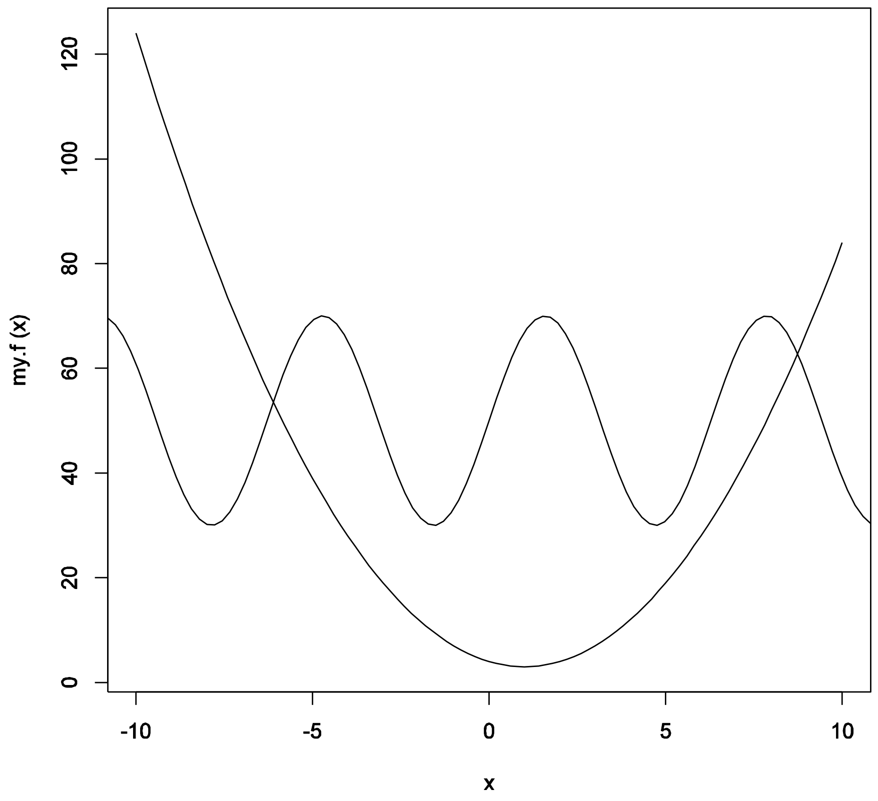
```
my.f(3)
```

Q. The R function curve() can be used to plot a given function or expression over a given range. To plot the function we have created use the command

```
curve(my.f(x), from = -10, to = 10)
```

Q. If we have another function that we want to plot then we can add to the existing plot using the argument add = TRUE. To try this out enter the commands

```
myg = function(x) { 20*sin(x) + 50}
curve(myg, add = TRUE)
```



Q. Write an R function to evaluate the quadratic f(x) = $x^2 + 4x - 7$ and evaluate it at x=5 (**ans.** 38)

The R Help System

In addition to the ? command R has extensive documentation and a user friendly help system. These can be accessed by clicking the Home symbol in the Help window, or by entering

```
help.start()
```

in the Console window. From this window you will be able to browse the manuals and search for functions.

Reading in your own data

R has a very useful function called read.table for reading data into a session.

Q. Use a text editor to create a file called data.txt which should look something like this

```
subject weight length
1       34.6   87.20
2       65.3   76.20
3       76.6   71.90
4       42.0   9.01
5       45.3   87.10
```

Q. Read the data into R using the command

```
a = read.table(file = "data.txt", header = TRUE)
```

Q. Look at the result to check you read in the data correctly

Q. Many R statistical functions work with data in the form of a `data frame'. We can turn our data into a data frame using

```
d1 = data.frame(a)
```

We can refer to individual variables e.g. using

```
d1$length
```

We can also `attach' to this data frame using

```
attach(d1)
```

and then we can refer to the individual variables in the data frame without referring to the data frame itself:

```
length
[1] 87.20 76.20 71.90  9.01 87.10
```

Plotting and summarizing data

One of the most important parts of any statistical analysis is the graphical exploration and presentation of the data and results. R has excellent graphics functionality that allows us to produce publication quality plots.

First we need some data to plot! There are many useful biostatistical datasets contained in the package ISwR which accompanies the book `Introductory Statistics with R ' by Peter Dalgaard.

[NB. A package is set of functions and datasets that can be loaded into an R session to provide extra functionality or access to datasets]

You will probably need to Install the package ISwR. This can be done by selecting

Tools->Install Packages

from the main R Studio menu and typing in the name of the package you want to install.

Q. Load the package by typing the following into the Console, or use the Packages tab in the bottom right R Studio window

```
library(ISwR)
```

Q. The dataset we will use consists of lung function data for 25 cystic fibrosis patients. The dataset can be loaded into the session using

```
data(cystfibr)
```

Q. Once loaded the dataset can be viewed by entering its name. [NB. the columns represent the variables and rows represent individuals/observations.]

```
cystfibr
```

Q. To use the variables in the dataset without reference to the dataset itself we attach to the dataset using the command

```
attach(cystfibr)
```

Q. Now we can just enter the name of a variable to view it. Try entering

```
tlc
```

Summary measures

R has functions that calculate various simple summary measures. Try out the following commands (use the help to find out what the function do)

```
mean(tlc)
```

```
median(tlc)
```

```
sd(tlc)
```

```
var(tlc)
```

```
range(tlc)

quantile(tlc)

summary(cystfibr)
```

hist

Q. The function hist plots histograms. The dataset has data on both males and females and it would be nice to view the data separately for each sex. First, we split the graphical display into two parts (one above the other using the command)

```
par(mfrow = c(2, 1))
```

Then we can plot a histogram of the variable tlc for just males using

```
hist(tlc[sex == 0], xlim = c(80, 150))
```

Finally, we can plot the tlc histogram from females using

```
hist(tlc[sex == 1], xlim = c(80, 150))
```

density

The function density can be used in conjunction with plot to produce a density estimate from a given dataset. A density estimate is effectively a smooth version of a histogram. The function lines can be used to overlay a density estimate on a histogram.

Q. A histogram (normalized so that the total area under the bars is 1.0) can be produced by setting the argument freq = FALSE in the hist function, i.e.

```
hist(tlc, freq = FALSE)
```

Q. The density plot can be added using the command

```
lines(density(tlc))
```

boxplot

The function boxplot can be used to plot a boxplot of a given dataset.

Q. For example, we can plot boxplots of the variable tlc for each value of sex using the commands

```
par(mfrow = c(1, 1))
boxplot(tlc ~ sex)
```

stripchart

The function stripchart is another useful way of plotting data that occur in groups. Using the argument method = ``jitter" avoids the problem of multiple points being overlaid on top of each other.

Q. Try this type of plot using

```
par(mfrow = c(1, 2))
stripchart(tlc ~ sex)
stripchart(tlc ~ sex, method = "jitter")
```

barplot

Barplots are useful for plotting categorical data that occur in groups.

Q. To demonstrate the barplot load the juul dataset in the ISwR library.

```
detach(cystfibr)
data(juul)
attach(juul)
```

Q. After attaching to the dataset we use the function table to tabulate two categorical variables from the dataset using the command

```
tab1 = table(sex, tanner)
```

Q. Look at the table by entering its name

```
tab1
```

Q. There are two main forms of a barplot, specified by the argument beside

```
par(mfrow = c(1, 2))
barplot(tab1)
barplot(tab1, beside = TRUE)
```

Q. Barplots of the table the other way round are produced by `transposing' the table using the function t(). To do this enter

```
par(mfrow = c(1, 2))
barplot(t(tab1))
barplot(t(tab1), beside = TRUE)
```

Q. By changing the rownames of the table we clarify which barplot refers to which group

```
rownames(tab1) = c(``boy", ``girl")
barplot(t(tab1), beside = TRUE)
```

Q. Add legends to the barplots using the argument legend.text

```
par(mfrow = c(1, 2))
barplot(tab1, beside = TRUE, legend.text = rownames(tab1))
barplot(t(tab1), beside = TRUE, legend.text = rownames(t(tab1)))
```

dotchart and pie

Q. Dotcharts and piecharts can also be produced in R. Try these out using

```
par(mfrow = c(1, 2))
dotchart(tab1)
dotchart(t(tab1))

par(mfrow = c(1, 2))
pie(tab1[1,], main = ``boy")
pie(tab1[2,], main = ``girl")
```

Creating images from plots

Often you will want to save a copy of a plot produced by a statistical analysis.

This is easy to do using R Studio.

When a plot is displayed in the plot window in the bottom right of the R Studio window you click the ‘Export’ button which will give you the option to “Save as Image…” or “Save as PDF…” or “Copy to Clipboard”.

Once you have saved your plot in a file you can include it in other documents or reports.

Calculating probabilities in R

Q. R has functions that can calculate probabilities from many widely used probability models. As an example lets suppose birth weights have a N(3350, 500^2) distribution and we want to calculate the probability that a baby is born who weighs less than 3000g. We can do this is R using

```
pnorm(3000, mean = 3350, sd = 500)
```

Q. What is the probability that a baby is born who weighs more than 3000g?

Q. What is the probability that a baby is born who weighs between 3000g and 4000g?