Intro to R and basic stats

Alastair Kerr

11/10/2018

Background

- R-language a statistical programming language

Other Useful Resources (Not covered today)

- ggplot2 Advanced Graphics Library
- Bioconductor Biology libraries for R
- RStudio an integrated development environment (IDE) for R.
- Shiny enables interactive graphs and tables in R

Advantages of R

- Free!
- Powerful, many libraries have been created to perform application specific tasks. e.g. analysis of microarray experiments and Next-Gen sequencing (bioconductor: including Bioseq group).
- Presentation quality graphics
  - Save as a png, pdf or svg
- History
  - What you do can be saved for the next time you use R.
  - Ability to turn it into an automated script to perform again and again on different data (the Bioinformatics Core facility can do this for you and even integrate it into Galaxy)
- [RStudio] Notebooks to combine annotation and code.

Disadvantages

- Lack of a comprehensive Graphical user interface
  - However some packges have their own GUI

Preparation

- (Optional) Download and save the tutorial data set into a directory that you will use for the practical
- Start R-Studio from computers menu (use search box)
- OR if you have a command line bifx-* account, you can access RStudio on our servers via a web browser
  - http://bifx-cli.bio.ed.ac.uk:8787
  - http://bifx-rta.bio.ed.ac.uk:8787
  - http://bifx-core2.bio.ed.ac.uk:8787

Getting More Help

- Project Home page http://www.r-project.org/
  - Check out the 'introduction to R', which is a much more in depth guide.
- Also R has a built-in help system (see later)

- Bioconductor has “Vignettes” that should help teach the topic

RStudio Panes

The are 4 main panes, each may have several tabs

- Bottom left : Console Pane
  - Here you can type commands into R
  - Additional tabs may include a terminal and script outputs
- Top Left : Open files and scripts
  - Can include raw scripts or markdown
- Top Right : Environment
  - Objects you have stored
  - Commands you have typed
  - May also have connection to databases
- Bottom right
  - System files
  - Output from plots
  - Packages available
  - Help pages

Setting up a new project

In the top right hand side is a drop down menu. Here you can start a new project. For now we will create a new one in a new directory (or in an existing directory in which you have downloaded the table). Make sure you have write permission for the directory you choose.

Once you have done this, this will be your **working directory**. Files will be saved (or loaded from) here by default unless you specify a full path.
You can change your working directory under the session menu at the top.

Using Rstudio has the advantage that everything you do will be remembered between RStudio sessions. If changing to another project using the top right menu, be sure to select yes in the pop up menu to save any changes.

R – terms used in workshop

- Working directory
  - This is the directory used to store your data and results.
  - It is useful if it is also the directory where your input data is stored.
- Vector
  - a list of numbers, (equivalent to a column in a table)
- Data Frame
  - a group of Vectors, a table where the rows are not necessarily related
- Matrix
  - a table where columns and rows are related

R Syntax

R is a **functional** based language, the inputs to a function, including options, are in brackets.

Typical command:

```
Function(data, options, moreOptions)
```

We can store the output of a function.

```
result = Function(data, options, moreOptions)
```

Even quit is a function (you can quit using the RStudio interface). Note I have commented this out using '#' so I do not accidently run this command!

```
#q()
```

So is help

```
help(read.table)
```

Provides the help page for the *function* **read.table()**

```
help.search('t test')
```

Searches for help pages that might relate to the phrase 't test'

NOTE quotes are needed for search strings, they are not needed when referring to data objects or function names.

There is a short cut for help, "?" which shows the help page on a function name, same as help(function)

```
?read.table
```

?? searches for help pages on functions, same as help.search('phrase')

```
??'t test'
```

Information is usually returned from a function, by default this is printed to screen

```
read.table('http://bifx-core.bio.ed.ac.uk/data.tsv')
```

This can always be stored, we call what it is stored in an 'object'

```
mydata  <- read.table('http://bifx-core.bio.ed.ac.uk/data.tsv')
```

here **mydata** is an object of type dataframe and the syntax **<-** assigns to the left.

You could also use the equals sign **=**

```
mydata = read.table('http://bifx-core.bio.ed.ac.uk/data.tsv')
```

or the right assign sign **->**

```
read.table('http://bifx-core.bio.ed.ac.uk/data.tsv') -> mydata
```

Reminder:

- Vector: a list of numbers. Equivalent to a column in a table
- Data Frame: a collection of vectors. Equivalent to a table

Hint: When using the console, the Up/Down arrow keys can be use to cycle through previous commands

Getting data into R

For a beginner this can be is the hardest part, it is also the most important to get right. **Please if I am unclear, say so!**

It is possible to create a vector by typing data directly into R using the combine function 'c'

```
x   <-  c(1,2,3,4,5)
```

same as

```
x  <-  c(1:5)
```

creates the vector x with the numbers between 1 and 5.

You can see what is in an object at any time by typing its name;

```
x
```

will produce the output

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

Note that names need to be quoted

```
daysofweek  <- c('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday')
```

Usually however you want to input from a file. We have touched on the 'read.table' function already.

```
mydata  <- read.table('http://bifx-core.bio.ed.ac.uk/data.tsv')
```

Now **mydata** is a data frame with multiple vectors. Each vector can be identified by the default syntax **mydata$V1 mydata$V2 mydata$V3**

If any of these are typed it will print to screen

```
mydata$V1
```

```
##  [1] A 1 2 2 3 3 3 3 3 4 4 4 4 4 4 4 4 4 5 5 5 5 6 6 6 7 7 8
## Levels: 1 2 3 4 5 6 7 8 A
```

By default the function assumes certain things from the file

- The file is a plain text file (there are functions to read excel files)
- columns are separated by any number of tabs or spaces
- there is the same number of data points in each column
- there is no header row (labels for the columns)
- there is no column with names for the rows [I'll explain].

If any of these are false, we need to tell that to the function.

If it has a header column

```
mydata  <- read.table('http://bifx-core.bio.ed.ac.uk/data.tsv', header=TRUE)  #  header=T also works
```

Note that there is a comma between different parts of the functions arguments. If there is one less column in the header row, then R assumes that the 1st column of data after the header are the row names.

Now the vectors (columns) are identified by their name mydata$A mydata$B mydata$C if any of these are typed it will print to screen

```
summary(mydata) # Summary about the whole data frame
```

```
##        A               B               C               D
##  Min.   :1.000   Min.   : 4.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:3.000   1st Qu.: 6.000   1st Qu.:1.000   1st Qu.:2.000
##  Median :4.000   Median : 7.000   Median :1.000   Median :3.000
##  Mean   :4.296   Mean   : 7.296   Mean   :1.778   Mean   :3.333
##  3rd Qu.:5.000   3rd Qu.: 8.000   3rd Qu.:2.000   3rd Qu.:4.000
##  Max.   :8.000   Max.   :11.000   Max.   :5.000   Max.   :9.000
##        E
```

```
##  Min.   :1.000
##  1st Qu.:3.500
##  Median :4.000
##  Mean   :4.407
##  3rd Qu.:5.000
##  Max.   :8.000
```

```
summary(mydata$A)  # Summary information of column A
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.000   4.000   4.296   5.000   8.000
```

We can shortcut having to type the data frame each time by attaching it

```
attach(mydata)
```

```
summary(B) #  summary of column B as 'mydata' is attached
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.000   6.000   7.000   7.296   8.000  11.000
```

Two other important options for read.table

If it is separated only by tabs and has a header

```
mydata  <- read.table('http://bifx-core.bio.ed.ac.uk/data.tsv', header=T, sep='\t')
```

Really useful if you have spaces in the contents of some columns, so R does not mess up reading the columns. However if the columns are of an uneven length it will tell you.

If you know that the file has uneven columns

```
mydata  <- read.table('http://bifx-core.bio.ed.ac.uk/data.tsv', header=T, sep='\t', fill=TRUE)
```

This causes R to fill empty spaces in columns with 'NA'.

The last two examples will still work with our file and give the same result as with only headers=T

As this is such a common task there are functions identical to read.table but with different default settings. e.g. **read.delim** and **read.csv**. Check out the help pages for each.

Import Dataset

In the top right panel there is a button called "import dataset". This can make importing data much easier. It can call the read.table set of functions or use alternative libraries. The command used will be displayed on the console. Note that you need to have the file on the computer to use this button.

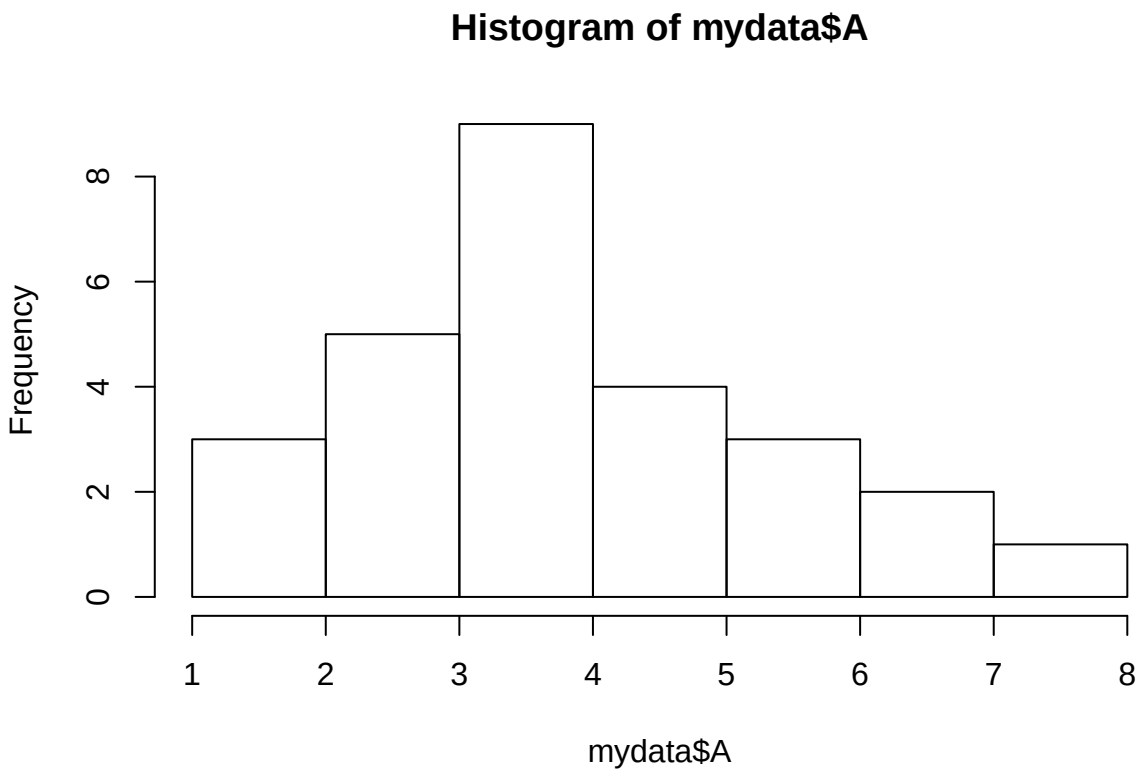
Graphs

I recommend learning ggplot2 for graphics but we will cover basic plots in "base" R.

Remember to get more information about the options to a function type '?function'

Histogram of A

```
hist(mydata$A)
```

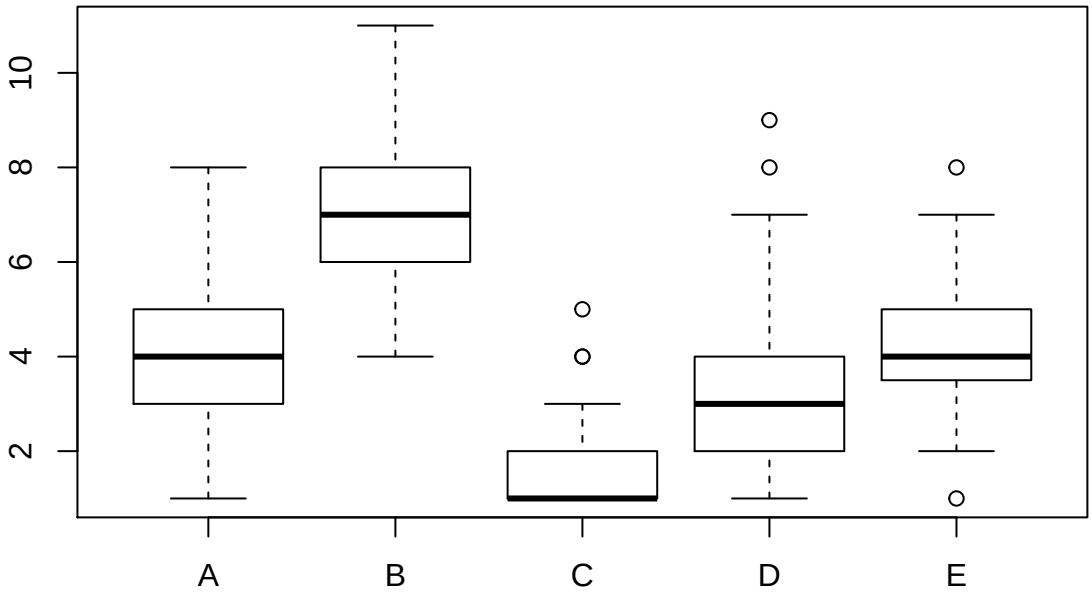


If there was more data we could increase the number of vertical columns with the option, breaks=50 (or another relevant number).

```
hist(mydata$A, breaks=5)
```

Boxplots

```
boxplot(mydata)
```



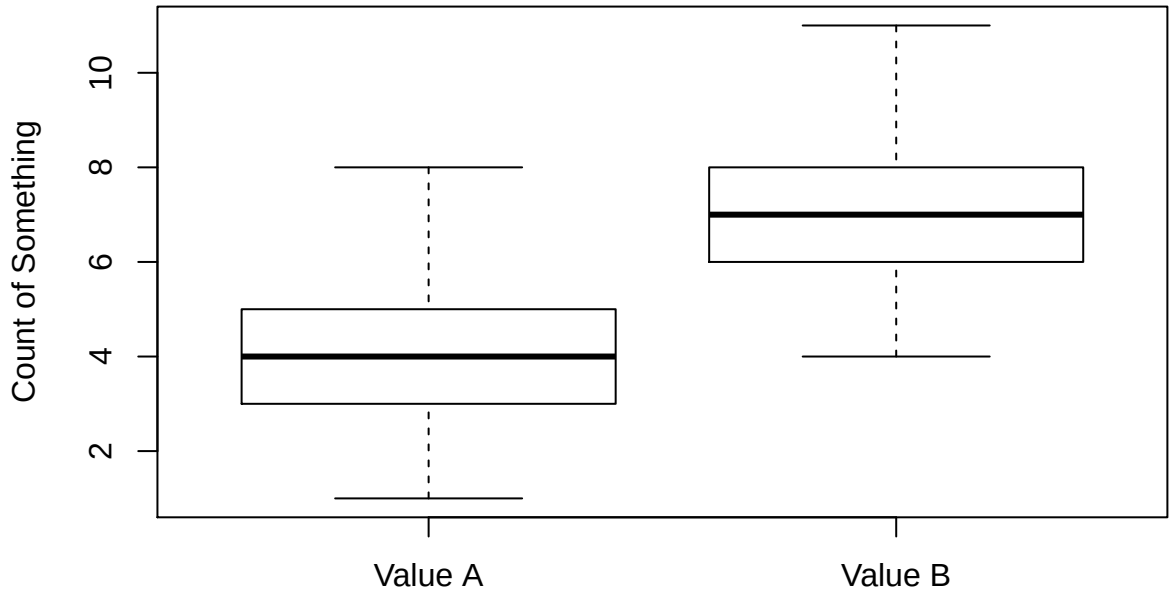
We can get rid of the need to type the data frame each time by using the attach function

```r
attach(mydata) # if not already done so
```

```r
boxplot(mydata$A, mydata$B, names=c("Value A", "Value B") , ylab="Count of Something")
```

same as

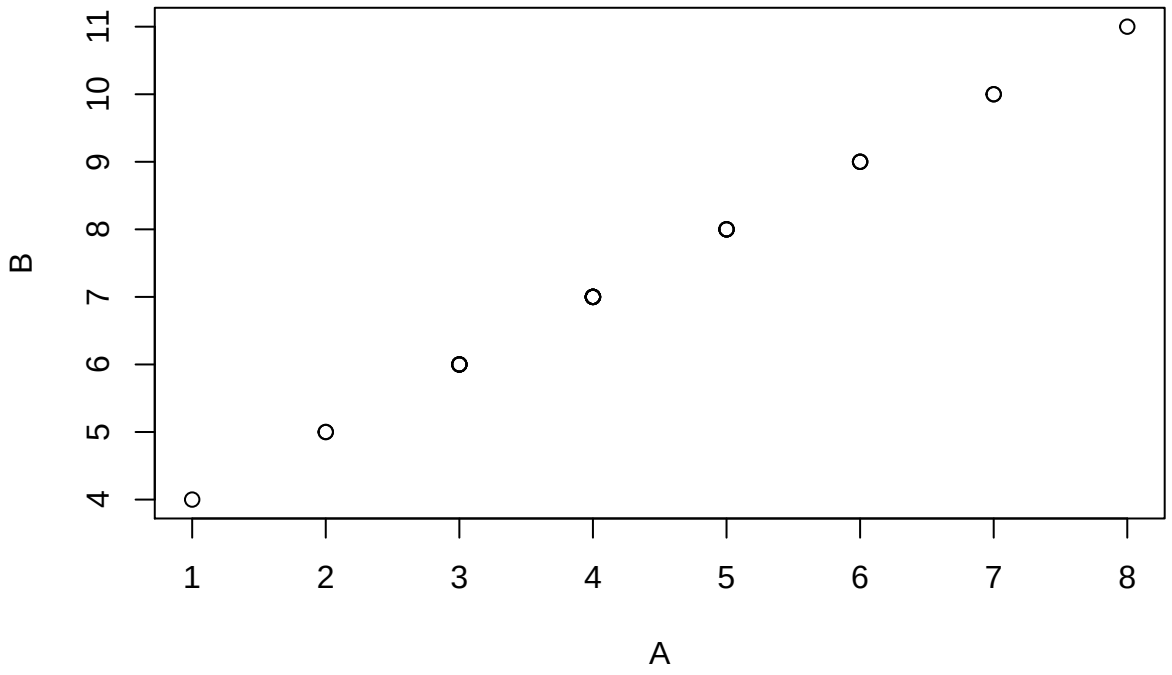
```r
boxplot(A, B, names=c("Value A", "Value B") , ylab="Count of Something")
```



Note that the opposite function of attach is detach

Scatter plot

```r
attach(mydata) # if not already done so
plot(A,B)  #  or plot(mydata$A, mydata$B)
```



Saving images

Use the export button under Plots in the bottom right plane.

Alternatively these instructions work for any device and can be used in scripts.

You need to create a new device of the type of file you need, then send the data to that device

to save as a png file (easy to load into the likes of powerpoint, also great for web applications).

```
png('filename.png')
boxplot(A, B, name=c("Value A", "Value B") , ylab="Count of Something")
```

OR to save as a pdf

```
pdf('filename.pdf')
boxplot(A, B, name=c("Value A", "Value B") , ylab="Count of Something")
```

Note that nothing will appear on screen, the output is going to the file. Also it may not be saved immediately but will once the device (or R) is quit.

Or if you want to remain in R

```
dev.off() #turns off the png (or pdf etc) device, thus forces the data to save
```

Testing Data

IMPORTANT use non-parametric tests for non-parametric data

Always best to check Shapiro Wilk test for normality

```
shapiro.test(vector)
```

If significant then NON-parametric i.e. if $p < 0.05$

Do this for each data point

Do A and E differ by random chance?

As A and E are both normally distributed we a use a parametric test (test the means)

```
t.test(A,E)
```

```
##
##  Welch Two Sample t-test
##
## data:  A and E
## t = -0.25047, df = 51.997, p-value = 0.8032
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.0012865  0.7790643
## sample estimates:
## mean of x mean of y
##  4.296296  4.407407
```

If the data is paired you should use the option **"paired=true"**

`PAIRED DATA: sets of data from the same row that came from the same set (e.g. time-frame)`

What about A and D?

As D is NOT normally distributed we a use a non-parametric test

I prefer the Kolmogorov-Smirnov Test (aka KS test)

ks.test(A,D)

Multple Testing

If you run multiple tests on the same data you need to adjust the p-values for **multiple testing correction**. The easiest method is to use the function p.adjust(x), where x is a list of p-values. e.g.

```
x = c(0.0001, 0.05, 0.001, 0.1, 0.1, 0.1, 0.1,0.5,0.5,0.5)
p.adjust(x)
```

```
##  [1] 0.001 0.400 0.009 0.700 0.700 0.700 0.700 1.000 1.000 1.000
```

Matrix Data and tests

A matrix is simply a data frame in which the row positions are important.

You can turn a data frame into a matrix using the as.matrix() functional

```
mymatrix <- as.matrix(mydata)
```

Or create one from a vector using the **matrix()** function (See **?matrix**)

```
lst <- c(54,66,80,20)
```

```
matrix(lst, nrow=2)
```

Which gives:

```
##      [,1] [,2]
## [1,]   54   80
## [2,]   66   20
```

Note: nrow specifies the number of rows (alternatively you can specify the number of columns by **ncol**). The default parameters assume numbers in the list fill the first column, then fill the second column and so on. This can be changed by the **byrow** argument.

Same as

```
twoBtwo <- matrix(c(54,66,80,20), nrow=2)
```

Now the matrix is saved and is called twoBtwo.

You can change the default column and row names with the colnames and rownames function

```
colnames(twoBtwo) <- c('A1', 'B1')
```

```
rownames(twoBtwo) <- c('A2', 'B2')
```

Chi-squared and Fisher's Exact Test (Count based data)

Observing a fluorescent marker in normal or wild mutant cells. Is there a significant difference between the 2 cell types?

```
Wild type cells with marker: 54
Wild type cells no marker: 66
Mutant cells with marker: 80
Mutant cells no marker: 20
```

Fortunately we have this already in the **twoBtwo** matrix, and the values in **lst**!

Chi-squared test, R function **chisq.test()**, works on matrixes. It is a type of likelihood ratio test

All these give the same result:

```
chisq.test(twoBtwo)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  twoBtwo
## X-squared = 26.612, df = 1, p-value = 2.486e-07
```

```
chisq.test(matrix(lst, nrow=2))
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  matrix(lst, nrow = 2)
## X-squared = 26.612, df = 1, p-value = 2.486e-07
```

```
chisq.test(matrix(c(54,66,80,20), nrow=2))
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  matrix(c(54, 66, 80, 20), nrow = 2)
## X-squared = 26.612, df = 1, p-value = 2.486e-07
```

Note if values in any box were small (e.g. <=5) then Fisher's exact test should be used

```
fisher.test(twoBtwo)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  twoBtwo
## p-value = 1.029e-07
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.1055182 0.3899188
## sample estimates:
## odds ratio
##  0.2061251
```

I recommend using Fisher's exact test over the chi square in general as the extra compute is trivial on a computer and the test is more robust.