Introduction to R

Richard Wilkinson

What is R?

R is a free open-source programming language that is primarily used for statistical analysis of data. Because almost all practical statistical analysis is carried out on a computer (as is most applied maths), becoming familiar with programming languages is an essential task for any statistician.

Why learn R?

- R is the most popular statistical programming language in academia, and will soon be in industry (currently its second most popular).
- It is open source – free!
- It is available on Windows, Mac and Linux.
- It is incredibly powerful.
- See http://www.nature.com/news/programming-tools-adventures-with-r-1.16609 for a recent article about the rise of R.

However, R requires practice in order to learn how to use it. Please go through this document in your own time or at the drop-in computer help session, and attempt the exercises. If you are using your own computer, download and install R from http://cran.r-project.org/

There are several good online R tutorials. One of best I've found is at http://tryr.codeschool.com/ which allows you to type commands into your web-browser.

Exercise 1

If you feel you've forgotten the R basics from G11STA, work your way through the R tutorial above.

Rstudio

Rstudio is an integrated development environment for R. It allows you to type your code into one window and run it in another, without having to copy and paste all the time. It also allows you to see the variables you've created, keep track of figures, and manually edit matrices. I **highly recommend** you download Rstudio and do all your R work in Rstudio.

You can download Rstudio from http://www.rstudio.com/. Note you will still need to download R first.

There is no reason not to use Rstudio, as it will make it much easier to save all your files and to write your coursework.

R markdown

One way to write your coursework is to copy and paste R output and figures into a Word file. Besides from the nuisance of having to continually switch between windows, this approach also makes it difficult to correct errors, as if you find an error near the beginning of your work, you then have to redo all the later results and figures, repeating the laborious task of copying and pasting.

There is a *better* way. **R markdown** is a way of combining R code and output (including figures) with text. You write everything into a single .Rmd file in Rstudio (click the new file button, and select a R markdown

file). Then when you click the **knit** button, Rstudio will run all your code, produce figures, and produce a beautiful looking document that includes all your text. You can produce html files (files that look good on web-browsers), pdfs, or Word documents this way. This removes the need for copying and pasting. Moreover, if you change any code in the document, and press the knit button again, it will then recreate all your figures and output using the new code.

Rather than provide a tutorial here, it is easiest if you see http://rmarkdown.rstudio.com for details.

R markdown Basics

This document and all the case studies are created using the rmarkdown package in Rstudio. You can download the .Rmd files from moodle to see how simple it is. To create a .html file for viewing in a web-browser, type the following into an R console:

```
library(rmarkdown)
render('Rinto.Rmd', "html_document")
```

or simply press the Knit html button in Rstudio.

To create a .pdf file which is more suitable for printing, type

```
render('Rinto.Rmd', "pdf_document")
```

or press the Knit PDF button in Rstudio. You can also create MS Word documents which you can edit later. You can extract just the R code, by typing

```
library(knitr)
purl('Rinto.Rmd')
```

This creates the file Rintro.R containing only the R commands.

The beauty of using R markdown is that all the figures are saved automatically for you. If you choose not to use R markdown, then you will have to manually save all of your figures using a command such as

```
dev.print(pdf, file="FuelScatterPlots.pdf", width=12, height=12) # save the file
```

Note that you can include equations

$$\pi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

by using latex. This is the type-setting program used by nearly all mathematicians. You will need to learn latex in the third year if you do a project. For your coursework for this module, you can write equations in by hand if necessary.

Exercise 2

- Create an R markdown document in Rstudio, selecting the pdf option. Create a vector of 1000 random variables, and plot a histogram. Now repeat this and create a Word file.

Starting Rstudio on the University of Nottingham PC Network

First log into the computer system. You can start Rstudio by going to the start menu and selecting

- Start: All programs: UoN Software: Statistical and mathematical: Rstudio

R packages

One of the most powerful aspects of R is that it allows other people to contribute their own code for performing certain tasks. These are provided in R packages.

The **first** time you use a R package you will need to install it (they should be pre-installed on University machines). This is very easy to do as long as you have an internet connection. For example, to install the car package, which we will use extensively in the case studies, simply type

```
install.packages('car')
```

When you need to use a command in a package, you must first load it.

```
library(car)
```

You only need to load each pacakge once each session.

Some basics

Rather than giving an extensive introduction to R (which was introduced in G11STA), I will just point out some useful aspects here. The tutorial at http://tryr.codeschool.com/ can be used as a refresher if you need it. The best way to learn the aspects of R needed for this module, is to type the commands in the case-studies into R for yourself. This will teach you everything you need to know. Another useful source is google. There are many useful webpages answering nearly every R related question you can think of.

Exercise 3

- Work through Case Study 3 on basic model fitting in R.

Some useful reminders

- Use the text editing facility in Rstudio for storing your code. Click the new file button, and select a new R script (a .R file). Type all of your R commands into this file, and then run them from here. The source button in rstudio is useful, as it runs all the commands in the current file. If you just wish to run a single line, you can press the run button (or ctrl-enter if you prefer keyboard shortcuts), which runs the line the cursor is currently on. You never need to copy and paste. You can then convert this .R file into a .Rmd markdown file later if you wish to turn it into a readable document.
- You can comment your code (useful for turning bits off so they don't run), by typing #. Then everything on that line after the # is treated as a comment

```
rnorm(2) # everything from here is ignored by R.
```

```
## [1] -0.1070293  0.2732360
```

- The help pages are also useful. Type ? before any command to see the help pages

```
?lm
```

- If you wish to remove variables from memory then use rm, e.g.

```
x <- c(21,3,5)
rm(x)
```

- If you press the up arrow when the cursor is in the R console, then you can scroll through your command history. This is particularly useful if you had a small typo in a command that you wish to correct. Tab is also useful as it auto-completes commands.

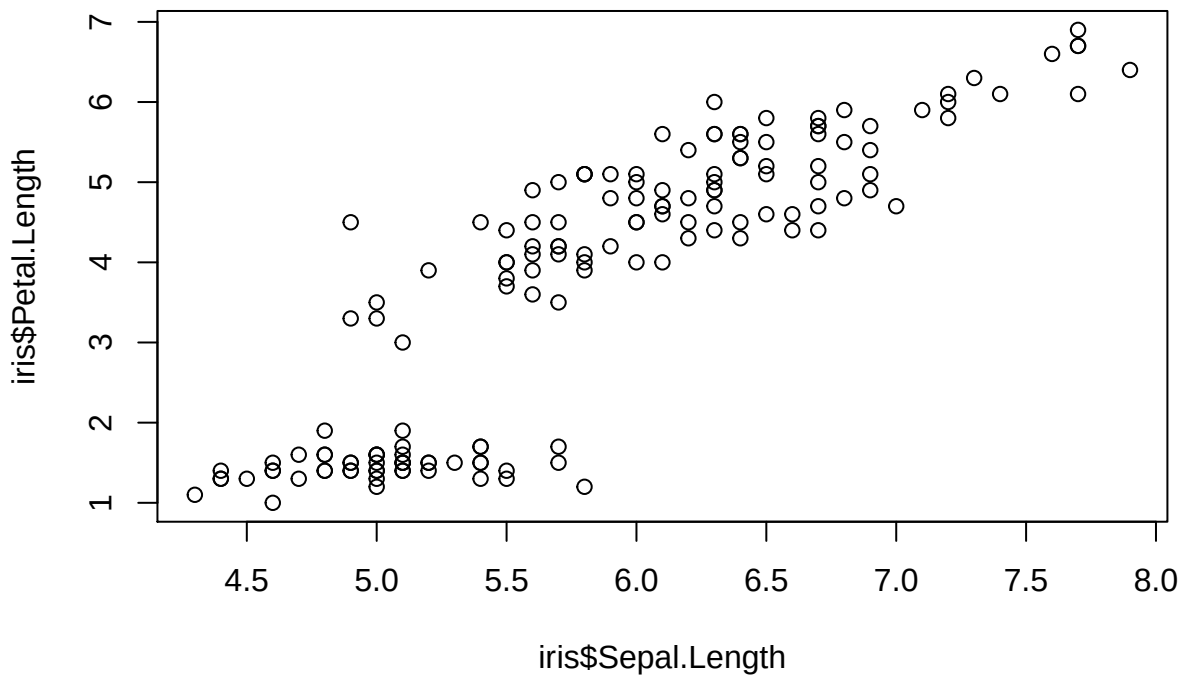
Loading Data into R

Loading data into R can be challenging at first. The key is to first save the data file in a directory of your choice, and then to change the working directory of R to that directory. The working directory is where R will look for any data, and where it will save any figures you create. To change the working directory in Rstudio, click on the Files tab in the bottom right hand window, navigate to where you want to be your home directory, and then click More -> Set as working directory.

Simple graphics

Simple plots in R are easy to create.

```
data(iris)
plot(iris$Sepal.Length, iris$Petal.Length)
```

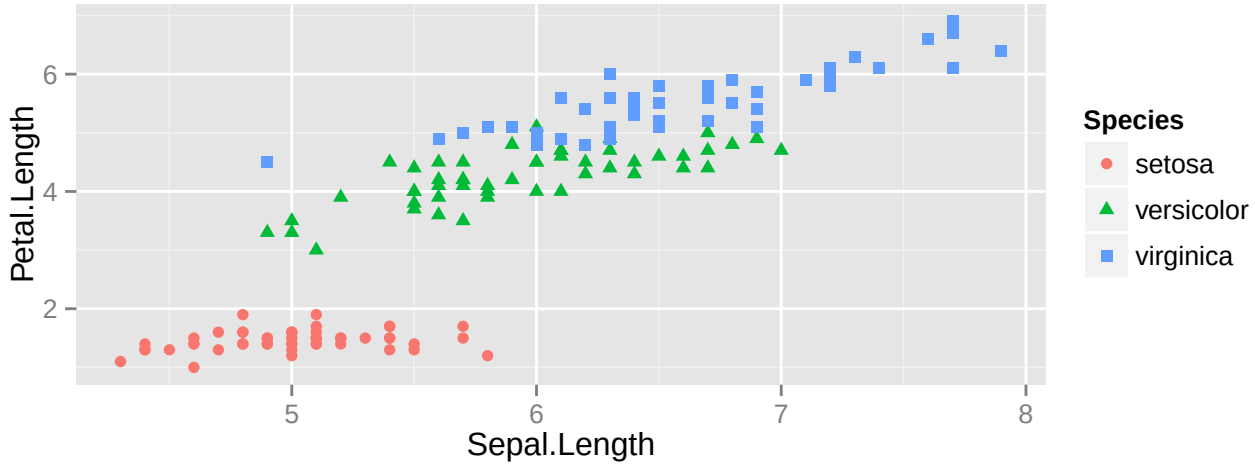


as are histograms.

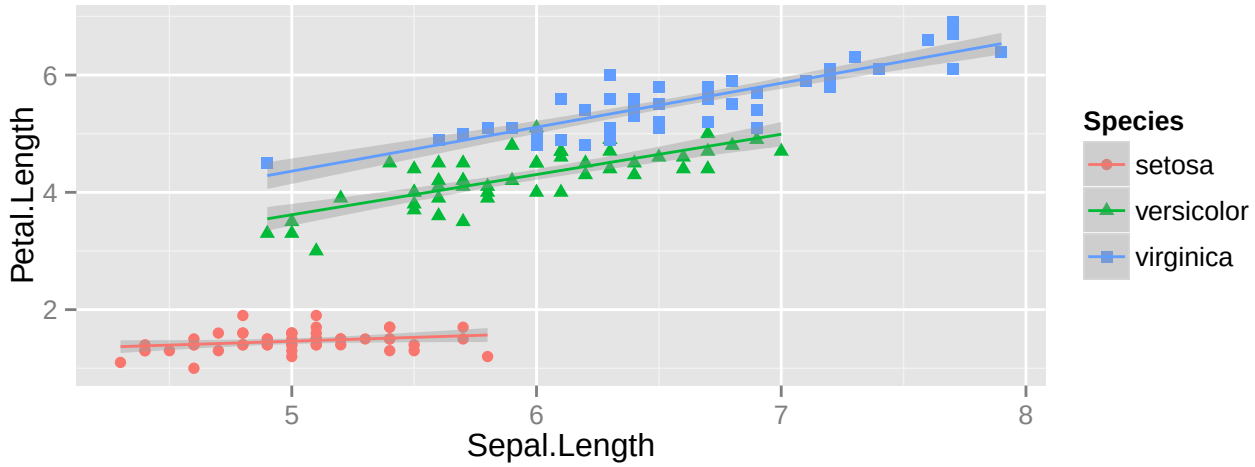
```
hist(iris$Sepal.Length)
```

The ggplot2 package is a way of easily creating beautiful plots, and which makes it easy to add colour and regression lines.

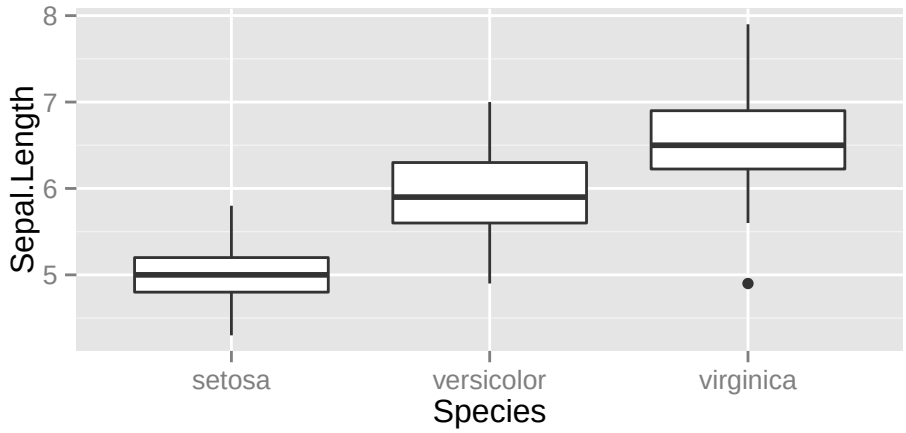
```
library(ggplot2)
qplot(x=Sepal.Length, y=Petal.Length, data=iris, colour=Species, shape=Species)
```




```
qplot(x=Sepal.Length, y=Petal.Length, data=iris, colour=Species, shape=Species,
      geom=c('point', 'smooth'), method='lm')
```




```
qplot(x=Species, y=Sepal.Length, data=iris, geom="boxplot")
```



See the exploratory data analysis case study for examples. There are many websites which offer help with ggplot2, many of which contain more detail than we need. A good site which shows output from many different function calls is http://www.ceb-institute.org/bbs/wp-content/uploads/2011/09/handout_ggplot2.pdf

Data frames

Data frames are the preferred way to store data in R. They are essentially like matrices and can be indexed by their row or column in the same way. Each row corresponds to a particular case or observation, and each column corresponds to a particular measurement (covariate or response).

```
iris$Species ## access the Species column as a list
iris[,5] ## access  the same information by indexing the matrix
```

If you want to select several columns, it is often easier to use the commands in the dplyr package. Try the following

```
library(dplyr)
select(iris, Species, Sepal.Length)
select(iris, starts_with("Petal"))
select(iris, -Species)
```

to see what they do.

Picking out particular observations corresponding to their properties is called filtering. This can be done either with the filter command from dplyr or explicitly. For example, the following two commands do the same thing.

```
iris[iris['Species']=='virginica',] ## explicitly accessing the matrix elements
filter(iris, Species=='virignica') ## using dplyr
```

What do the following commands do?

```
filter(iris, Sepal.Length<5)
filter(iris, Sepal.Length<5, Petal.Width==0.2)
filter(iris, Sepal.Length<5, Petal.Width!=0.2)
```

Linear model syntax

lm is the key function used to fit models. Its usage is illustrated in the 9 case studies.

To fit a linear model using R, it is first necessary to understand the syntax for defining models. Let's assume that the response variable being modeled is Y and that A, B and C are covariates that might affect Y. The table below provides some useful examples. Note that the mathematical symbols used to define models do not have their normal meanings!

Syntax	Model
$Y \sim A$	$Y = \beta_o + \beta_1 A$
$Y \sim -1 + A$	$Y = \beta_1 A$
$Y \sim A + A^2$	$Y = \beta_o + \beta_1 (A + A^2)$
$Y \sim A + I(A^2)$	$Y = \beta_o + \beta_1 A + \beta_2 A^2$
$Y \sim A + B$	$Y = \beta_o + \beta_1 A + \beta_2 B$
$Y \sim A : B$	$Y = \beta_o + \beta_1 AB$
$Y \sim A * B$	$Y = \beta_o + \beta_1 A + \beta_2 B + \beta_3 AB$
$Y \sim (A + B + C)^2$	$Y = \beta_o + \beta_1 A + \beta_2 B + \beta_3 C + \beta_4 AB + \beta_5 AC + \beta_6 BC$

Notice the difference between the third and fourth lines of the table. The function I() is used to seperate terms, so that we get A and A^2 included as independent variables in the regression.

You can also include transformations. For example

```
lm(log(Y)~ A + B^2+exp(C))
```

fits the model

$$\log(Y) = \beta_0 + \beta_1 A + \beta_2 B^2 + \beta_3 \exp(C) + \epsilon$$

Trying things for yourself

You should now try fitting a linear model.

Exercise 4

- Load the hills dataset from the MASS library with the command

```
library(MASS)
data(hills)
?hills
```

- Perform some initial exploratory data analysis by producing scatter plots of each of the pairs of variables.
- Fit a linear model to predict the winning race time from the distance and the total amount of climbing.
- What are the estimated coefficients?
- What are the error estimates (standard errors) on these estimates?
- What is the deviance of the model?
- What is the R^2 and adjusted R^2 value?
- Use visreg to visualise the fitted model.
- Now fit the model that includes an interaction term between distance and the amount of climbing. Is this a significant improvement over the simpler model?
- Fit the model

$$time = a + b \times dist + c \times dist^2 + \epsilon$$

The Case Study handout gives you many different commands for doing this, so I shall not repeat them here. Now try downloading one of the datasets from the module webpage, and try fitting various linear models.

Programming with R

Functions

A function is a series of commands that can be called by using a simple name and arguments. Consider the following function:

```
testfun<-function(n=100){
    x<-rnorm(n)
    out<-list()
    out$sum <- sum(x)
    out$sumsq <- sum(x^2)
    out$name <- 'Normal'
    return(out)
}
```

In order to see the details of the stored function, just type the function name.

Loops

Loops are useful for repeating calculations. The syntax is illustrated below

```
for (i in 1:5){
  print(i^2)
}
```

If statements

If statements can be used to check logical statements, and then executing code only if the statement is true. The loop below does the same thing as above, but only for integers that are divisible by 3:

```
for (i in 1:10){
  if (i/3==trunc(i/3)){
    print(i^2)
  }
}
```

While statements

WHILE loops can be used to carry out a statement repeatedly while the condition is TRUE. If the condition is FALSE then the command is not carried out, and the program then exits the WHILE loop.

```
sum<-1
while (sum <= 100){
  sum<-sum+1
}
```

Some useful online resources

- Another brief introduction to R http://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf
- R markdown help http://rmarkdown.rstudio.com