AN INTRODUCTION TO R

R has become a worldwide language for statistics, predictive analysis, and data visualisation. It offers a wide range of methodologies for understanding data and is useful if you want to collect, summarise, transform, explore, visualise or present data.

R is probably best known for producing sophisticated graphics, the base installation provides hundreds of data-management, statistical and graphical functions. However, there are thousands of extension packages that can be installed.

R is an open source project and is freely available for a range of platforms.

The examples and information to be used here have been taken from the following book:

"R in Action: Data Analysis and Graphics with R", Robert Kabacoff, Manning Pubs Co Series, 2011.

1. WORKING WITH R

R is a **case sensitive, interpreted language**. It is also an **interactive language** which indicates readiness for the next line of user input with a prompt (> by default).

You can either enter commands one at a time at the command prompt (>) (which we will consider here) or run a set of commands from a source file.

There are a wide variety of data types, including vectors, matrices, data frames (similar to datasets), and lists (collections of objects).

Most functionality is provided through built-in and user-created functions and the creation and manipulation of objects.

An **object** is basically anything that can be assigned a value. For R, that is just about everything (data, functions, graphs, analytic results, and more). All objects are kept in memory during an interactive session.

Statements consist of functions and assignments.

R uses the symbol <- for assignments, rather than the typical = sign.

```
height<-3

weight<-6

area<-height*weight

area
```

Example (Assignment statement)

```
> x<-rnorm(5)
> x<-2+2
> y<-2*x
> print(x)
[1] 4
> print(y)
[1] 8
```

This creates a vector object named x containing five random derivatives from a standard normal distribution.

Comments are preceded by the # symbol, therefore any text appearing after the # is ignored by the R interpreter.

R variables and data types

First, we introduce the common variable types and data types that you'll be working with in R. Commonly, errors involve using the wrong variable or data type

Variable type	Type	Example
integer	Whole numbers	1, 100, -9
numeric	Decimals	0.1, -0.09, 234.567
character	Text	"A", "hello", "welcome"
logical	Booleans	TRUE or FALSE
factor	Categorical	"green", "blue", "red", "purple"
missing	Logical	NA
empty	-	NULL

Data types in R can be examined as follows.

```
class(TRUE)

class(NA)

class(2)

class(2L)

class(2.5)

is.integer(2.5)

is.integer(2)

is.integer(2L)

as.integer(2.5)

as.integer(TRUE)

as.integer(FALSE)

class("Hello students")
```

Vectors

A vector is the simplest type of data structure in R. Simply put, a vector is a sequence of data elements of the same basic type. Members of a vector are called Components.

Here is a vector containing three numeric values 2, 3 and 5.

```
> c(2, 3, 5)
[1] 2 3 5
```

And here is a vector of logical values.

```
> c(TRUE, FALSE, TRUE, FALSE, FALSE)
[1]  TRUE FALSE  TRUE FALSE FALSE
```

A vector can contain character strings.

```
> c("aa", "bb", "cc", "dd", "ee")
[1] "aa" "bb" "cc" "dd" "ee"
```

Incidentally, the number of members in a vector is given by the length function.

```
> length(c("aa", "bb", "cc", "dd", "ee"))
[1] 5
```

```
#Create and name vectors
drawn_suits<-c("hearts","spades","diamonds","diamonds","spades")
length(drawn_suits)
length(area)
is.vector(area)
remain<-c(11,12,11,13)
remain

suits<-c("spades","hearts","diamonds","clubs")
#Give names to the vector entries
names(remain)<-suits
remain

remainn<-c(spades=11,hearts=12,diamonds=11,clubs=13)
remain
```

Other vector manipulations

```
#Other special vectors
LETTERS
letters
1:1000
drawn_ranks<-c(7,5,"K","Q",4)
as.integer(drawn_ranks)


#Vector arithmetic
earnings<-c(50,100,30)
earnings+10
```

Vector subsetting

R's subsetting operators are powerful and fast. Mastery of subsetting allows you to succinctly express complex operations in a way that few other languages can match.

```
#Subsetting
remain
remain[1]
remain["spades"]
remain[c("spades","hearts")]


remain[-1]
remain[-c(1,2)]
remain[c(TRUE,FALSE,FALSE,FALSE)]
```

Example 1

To get a feel for the interface, we will work through a simple example.

Say that you are studying physical development and you've collected the ages and weights of 10 infants in the first year of their life. You are interested in the distribution of the weights and their relationship to age.

Table 1. The ages and weights of 10 infants.

Age (months)	Weight (Kg)
01	4.4
03	5.3
05	7.2
02	5.2
11	8.5
09	7.3
03	6.0
09	10.4
12	10.2
03	6.1

The age and weight data needs to be entered as vectors using the function c(), which **combines** its arguments into a vector list.

The mean and standard deviations of the weights, along with the correlation between age and weight also need to be calculated using the in-built R functions. These are: mean(), sd() and cor() respectively.

Age also needs to be plotted against weight to allow us to visually inspect the trend. The function here is plot().

Example (Analysing age and weights of infants)

```
> age <-c(1,3,5,2,11,9,3,9,12,3)
> weight <-c(4.4,5.3,7.2,5.2,8.5,7.3,6.0,10.4,10.2,6.1)
> mean(weight)
[1] 7.06
> sd(weight)
[1] 2.077498
> cor(age,weight)
[1] 0.9075655
> plot(age,weight)
```

What were the results for the mean weight and the standard deviation? Does there appear to be any relationship between age in months and weight in kilograms?

```
> cor.test(age,weight)
```

R Tips

- To get a sense of what R can do graphically, enter demo() at the command prompt to see a complete list of demonstrations. Try also selecting demo(graphics).
- Play around with some of the other demos and take a look at the code produced.
- R provides extensive help facilities, for general help try typing help.start() and RSiteSearch("function-name").

The workspace is your current R working environment and includes any user-defined objects. At the end of an R session you can save an image of the current workspace that's automatically reloaded the next time R starts.

The current working directory is the directory from which R will read files and to which it will save results by default.

To find out the current working directory type getwd(). To set the current use setwd().

Save the current workspace to a folder of your choice. Then start a new workspace and attempt to load this file.

```
> setwd("     ")

> getwd()

>load(“ ”)

>mean(weight)

>source("Introduction.RData")
```

2. CREATING DATASETS

The first step in any data analysis is the creation of a dataset that needs to be analysed. In R this involves:

- Selecting a data structure to hold the data
- Entering or importing the data into the data structure

Data can be entered manually or imported from an external source. These data sources can include text files, spreadsheets, statistical packages, and database-management systems (includes SAS and SQL databases).

Once a dataset is created you will typically annotate it, adding descriptive labels for variables and code. A data set is typically a rectangular array of data with rows representing observations and columns representing variables.

```
#Data frames

names<-c("Ann","Ben","David")

age<-c(28,30,9)

child<-c(FALSE,FALSE,TRUE)

people<-data.frame(names,age,child)

people

names(people)<-c("Names","Age","Child")

str(people)
```

Example 2

Table 2. A patient dataset.

PatientID	AdmDate	Age	Diabetes	Status
1	15/10/2014	25	Type1	Poor
2	01/11/2014	34	Type2	Improved
3	21/10/2014	28	Type1	Excellent
4	28/10/2014	52	Type1	Poor

In the dataset shown in Table 2, PatientID is a row or case identifier, AdmDate is a date variable, Age is a continuous variable, Diabetes is a nominal variable, and Status is an ordinal variable.

What is the difference between ordinal and nominal variables, give examples of each.

R contains a wide variety of structures for holding data including, scalars, vectors, arrays, data frames and lists.

Table 2 corresponds to a data frame in R, this is the most common structure. A data frame is more general than a matrix in that different columns can contain different modes of data (numeric, character and so on). It's similar to the dataset you'd typically see in SAS and SPSS.

A data frame is created with the data.frame() function:

```
> mydata <-data.frame(col1,col2,col3,...)
```

where col1, col2, col3, and so on are column vectors of any type. Names for columns can be provided with the names function.

Example (Creating a data frame)

```
> patientID<-c(1,2,3,4)
> age<- c(25,34,28,52)
> diabetes<-c("Type1","Type2", "Type1", "Type1")
> status <-c("Poor", "Improved", "Excellent", "Poor")
> patientdata<- data.frame(patientID,age,diabetes,status)
> patientdata
```

Example (Specify elements of a data frame)

```
> patientdata[1:2]

> patientdata[c("diabetes","status")]

> patientdata$age ( the $ indicates a particular variable)

> table(patientdata$diabetes,patientdata$status)
```

In this example patientID is used to identify individuals in the dataset. In R, case identifiers can be specified with a **rowname** option in the data-frame function.

For example:

> patientdata<- data.frame(patientID,age,diabetes,status, row.names=patientID)

Categorical (nominal) and ordered categorical (ordinal) variables in R are called **factors**. Factors are crucial in R because they determine how data is analysed and presented visually.

The function factor() stores the categorical values as a vector of integers in the range [1…k], where k is the number of unique values in the nominal variable where the original values (character strings) are mapped to these values.

For vectors representing ordinal variables, you add the parameter ordered=TRUE to the factor() function.

Example (Declaring factors)

```
> diabetes <-factor(diabetes)

> status <-factor(status, order=TRUE, levels=c("Poor", "Improved", "Excellent"))

> str(patientdata)

> summary(patientdata)
```

3. DATA INPUT

Perhaps the simplest way to enter data is from the keyboard. There are two common methods: entering data through R's build-in text editor and embedding data directly into your code.

The edit() function in R invokes a text editor that lets you enter data manually.

The steps are:

- Create an empty data frame (or matrix) with the variable names and modes you want to have in the final data set.
- Invoke the text editor on this data object, enter your data, and save the results to the data object.

Example (R text editor)

```
> mydata <-data.frame(age=numeric(0), gender=character(0),weight=numeric(0))
> mydata <-edit(mydata)
```

You can import data from delimited text files using read.table(), a function that reads a file in table format and saves it as a data frame. Each row of the table appears as one line in the file.

The syntax is

Mydataframe <-read.table(file,options)

where file is a delimited ASCII file and the options are parameters controlling how data is processed.

Optional arguments include:

`header`	a logical value indicating whether the file contains the names of the variables as its first line.
`sep`	the field separator character.
`dec`	the character used in the file for decimal points.
`row.names`	a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names.
`col.names`	a vector of optional names for the variables. The default is to use "V" followed by the column number.

`na.strings`	a character vector of strings which are to be interpreted as NA values (=c("-9","?"). Blank fields are also considered to be missing values in logical, integer, numeric and complex fields.
`colClasses`	character. A vector of classes to be assumed for the columns.
`nrows`	integer: the maximum number of rows to read in. Negative and other invalid values are ignored.
`skip`	integer: the number of lines of the data file to skip before beginning to read data.
`stringsAsFactors`	logical: should character vectors be converted to factors? Note that this is overridden by as.is and colClasses, both of which allow finer control.

Consider the text file named studentgrades.txt containing students' grades in maths, science, and social studies.

Each line of the file represents a student. The first line contains the variable names, separated with commas. Each subsequent line contains a new student's information, also separated with commas as below.

StudentID, First, Last, Math, Science, Social Studies
011, Bob, Smith, 90, 80, 67
012, Jane, Weary, 75, , 80
010, Dan, "Thornton, III", 65, 75, 70
040, Mary, "O'Leary", 90, 95, 92

Example (Importing text file)

```
> grades <- read.table("studentgrades.txt", header=TRUE, row.names="StudentID", sep=",")

>grades

>str(grades)
```

What do you notice about how the data is imported?

Social studies is renamed.

Student ID number is now row name.

Missing grade is read as NA.

First and last names are converted as factors, this is by default, which may not be desirable.

Example (Dealing with character variables)

```
> grades <- read.table("studentgrades.txt", header=TRUE, row.names="StudentID", sep=",",
stringsAsFactors=FALSE)>grades

>str(grades)
```

Example (Using colClasses option)

```
grades <- read.table("studentgrades.txt", header=TRUE, row.names="StudentID", sep=",",
colClasses=c("character", "character", "character", "numeric", "numeric","numeric"))

> str(grades)
```

The best way to read in an excel file is to export it to a comma-delimited file from Excel and import it into R using the method above. There is also a package name “xlsx” which allows you to import excel worksheets directly.

4. DATA MANAGEMENT

Once you have your data in R, the next step is to prepare it for analysis.

Example 3

Consider the topic of how men and women differ in the ways they lead their organisations.

Typical questions might be:

- Do men and women in management positions differ in the degree to which they defer to supervisors?
- Does this vary from country to country, or are these gender differences universal?

One way to answer these questions is to have bosses in multiple countries rate their managers on deferential behaviour, using questions like:

- My manager asks my opinion before making personnel decisions.

These questions could be rated on a scale of 1 to 5, with 1 being “strongly disagree” and 5 being “strongly agree”.

Table 3. Example of responses to differences in leadership.

Manager	Date	Country	Gender	Age	Q1	Q2	Q3	Q4	Q5
1	24/10/14	US	M	32	5	4	5	5	5
2	28/10/14	US	F	45	3	5	2	5	5
3	01/10/14	UK	F	25	3	5	5	5	2
4	12/10/14	UK	M	39	3	3	4		
5	01/05/14	UK	F	99	2	2	1	2	1

The data management issues we need to address before performing analyses are:

1. Combine five rating (q1 to q5) to a single mean score.
2. Handle missing values.
3. Create new dataset with only variables of interest.

Example (Creating leadership data)

```
,2,3,4,5)
> date<-c("24/10/14", "28/10/14","01/10/14","12/10/14","01/05/14")
> country<-c("US", "US", "UK", "UK", "UK")
> gender<-c("M","F","F","M", "F")
> age<-c(32,45,25,39,99)
> q1<-c(5,3,3,3,2)
> q2<-c(4,5,5,3,2)
> q3<-c(5,2,5,4,1)
> q4<-c(5,5,5,NA,2)
> q5<-c(5,5,2,NA,1)
> leadership<-
data.frame(manager,date,country,gender,age,q1,q2,q3,q4,q5,stringsAsFactors=FALSE)
> leadership
```

Example (Creating new variable)

This is accomplished with statements of the form:

Variable <- expressionattach

```
> sumq <- q1+q2+q3+q4+q5


> attach(leadership)
> leadership$sumq <-q1+q2+q3+q4+q5
> detach(leadership)
> leadership
```

Example (Missing values)

```
> leadership$age[leadership$age ==99] <-NA

> leadership
```

Example (Renaming variables)

```
> names (leadership)[2]<-"testDate"

>leadership
```

5. Statistical tests in R

t-tests

The t.test( ) function produces a variety of t-tests in R. Unlike most statistical packages, the default assumes unequal variance and applies the Welsh df modification.

To carry out independent 2-sample t-test:

t.test(y~x) # where y is numeric and x is a binary factor

Or independent 2-group t-test

t.test(y1,y2) # where y1 and y2 are numeric

For a paired samples t-test:

t.test(y1,y2,paired=TRUE) # where y1 & y2 are numeric

Finally for a one sample t-test:

t.test(y,mu=3) # Ho: mu=3

You can use the var.equal = TRUE option to specify equal variances and a pooled variance estimate. You can use the alternative="less" or alternative="greater" option to specify a one tailed test.

Nonparametric tests

For independent 2-group Mann-Whitney U Test

wilcox.test(y~A) # where y is numeric and A is A binary factor

For independent 2-group Mann-Whitney U Test

wilcox.test(y,x) # where y and x are numeric

For dependent 2-group Wilcoxon Signed Rank Test

wilcox.test(y1,y2,paired=TRUE) # where y1 and y2 are numeric

For Kruskal Wallis Test One Way Anova by Ranks

kruskal.test(y~A) # where y1 is numeric and A is a factor

For the wilcox.test you can use the alternative="less" or alternative="greater" option to specify a one tailed test.

Multiple Linear Regression Example

```
fit <- lm(y ~ x1 + x2 + x3, data=mydata)
summary(fit) # show results
```

Other useful functions:

```
coefficients(fit) # model coefficients
confint(fit, level=0.95) # CIs for model parameters
fitted(fit) # predicted values
residuals(fit) # residuals
anova(fit) # anova table
vcov(fit) # covariance matrix for model parameters
influence(fit) # regression diagnostics
```

For Diagnostic Plots

Diagnostic plots provide checks for heteroscedasticity, normality, and influential observations.

```
# diagnostic plots
layout(matrix(c(1,2,3,4),2,2)) # optional 4 graphs/page
plot(fit)
```

ANOVA

If you have been analyzing ANOVA designs in traditional statistical packages, you are likely to find R's approach less coherent and user-friendly.

In the following examples lower case letters are numeric variables and upper case letters are factors.

One Way Anova (Completely Randomized Design)

```
fit <- aov(y ~ A, data=mydataframe)
```

Multiple Comparisons

You can get Tukey HSD tests using the function below. By default, it calculates post hoc comparisons on each factor in the model. You can specify specific factors as an option.

```
# Tukey Honestly Significant Differences
TukeyHSD(fit) # where fit comes from aov()
```

Chi-Square Test

For 2-way tables you can use **chisq.test(***mytable***)** to test independence of the row and column variable.

By default, the p-value is calculated from the asymptotic chi-squared distribution of the test statistic. Optionally, the p-value can be derived via Monte Carlo simultation.

Fisher Exact Test

fisher.test(*x***)** provides an exact test of independence. *x* is a two dimensional contingency table in matrix form.