

Introduction to R

Dr. Norma Coffey
Norma.Coffey@ul.ie

University of Limerick

Course Outline

The R language:

1. **Language essentials:** Objects; functions, vectors, missing values, matrices and arrays, factors, lists, data frames. Indexing, sorting and implicit loops. Logical operators. Packages and libraries.
2. **Flow control:** for, while, if/else, repeat, break.
3. **Probability distributions:** Built-in distributions in R; densities, cumulatives, quantiles, random numbers.
4. **Statistical graphics:** Graphical devices. High level plots. Low level graphics functions.
5. **Statistical functions:** One and two-sample inference, regression and correlation, tabular data, power, sample size calculations.

Course Outline

Applications of R:

The applications of R will be explored by considering several case studies in statistics. Each case study is motivated by a scientific question that needs to be answered, and full background material is presented. The cases are grouped by broad statistical topics: data analysis, applied probability, statistical inference, regression methods.

Prime Texts:

1. Peter Dalgaard. Introductory Statistics with R. Springer, 2002. ISBN 0-387-95475-9. 519.5/DAL
2. W. John Braun and Duncan J. Murdoch. A first course in statistical programming with R. Cambridge University Press, 2007. ISBN 978-0-521-69424-7.

Assessment

Continuous assessment - 3 assessments weighted as 10%, 15% and 25%.

No end of semester exam.

Notes etc. available on <http://jkcray.maths.ul.ie/ms4024.html>

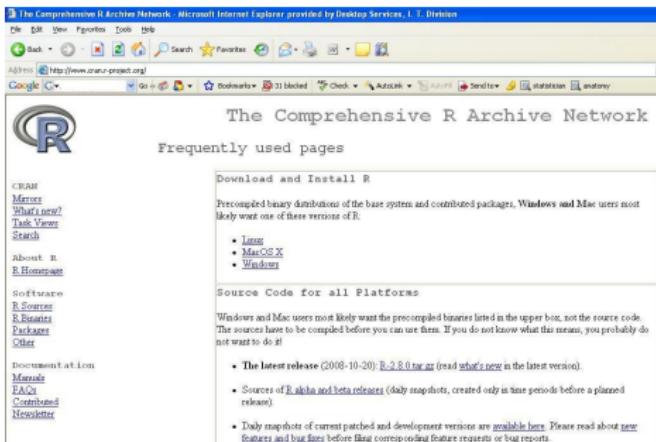
What is R?

R is

- a suite of software facilities for:
 - ▶ reading and manipulating data
 - ▶ computation
 - ▶ conducting statistical analyses
 - ▶ displaying the results
- open-source version (i.e. freely available version - no license fee) of the S programming language, a language for manipulating *objects*
- a programming environment for data analysis and graphics
- a platform for development and implementation of new algorithms
- Software and packages can be downloaded from
www.cran.r-project.org

Installing R

R must be installed on your system! If it is not, go to
www.cran.r-project.org



Click on

Windows > base > R-version-win32.exe > Run

and follow the instructions to install the programme.

Starting R

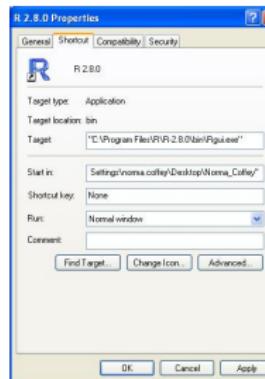
R can be started in the usual way by double-clicking on the R icon on the desktop.

R works best if you have a dedicated folder for each separate project - called the working folder.

- Create the directory/folder that will be used as the working folder, e.g. create a folder on your desktop titled `Your_name` by right-clicking, then clicking `New > Folder`.
- Right-click on an existing R icon and click `Copy`.
- In the working folder, right-click and click `Paste`. The R icon will appear in the folder.

Starting R

- Right-click on the R icon and click Properties.
- In the **Start in** box type the location of the working directory,
e.g.
"C:\Documents and Settings\norma.coffey\Desktop\Your_name"



- Click Apply, then Ok.

Starting R

There are 3 ways to start R in the working folder:

- double-click on the R shortcut
- double-click on the .RData file in the folder, when it exists
- double-click any R shortcut and use `setwd(filepath)` command

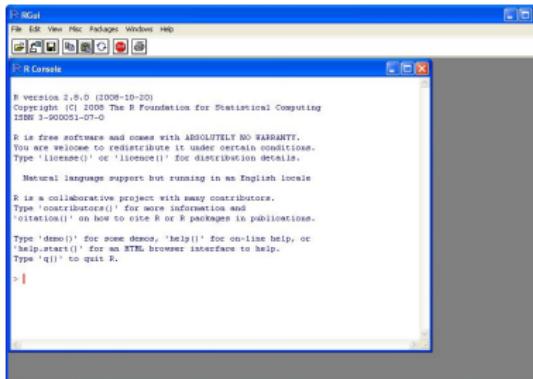


Figure: The R console (command line) window.

How does R work?

- R creates its objects in memory and saves them in a single file called .RData (by default)
- Commands are recorded in an .Rhistory file
- Commands may be recalled and reissued using up- and down-arrow
- Recalled commands may be edited
- Flawed commands may be abandoned by pressing <Esc>
- Copy-and-paste from a “script” file
- Copy-and-paste from the history window used for recalling several commands at once
- To end your session type q() or just kill the window.

How does R work?

There are a number of drop-down menus in the R Gui (File, Edit, View, Packages, Help).

Users are expected to type input (*commands*) into R in the console window. When R is ready for input, it prints out its prompt, a ">".

Commands:

- consist of *expressions* or *assignments*
- are separated by a semi-colon (;) or by a newline
- can be grouped together using braces ({ and })

Comments can be included and are indicated with a hash (#).

How does R work?

Users enter a line with a command after the prompt and press Enter.

The programme carries out the command and prints the result if relevant. For example, if the expression $2 + 2$ is typed in, the following is printed in the R console:

```
> 2 + 2  
[1] 4  
>
```

The prompt `>` indicates that R is ready for another command. If a command is incomplete at the end of a line, the prompt `+` is displayed on subsequent lines until the command is syntactically complete.

Calculator

R can also evaluate other standard calculations:

```
> exp(-2)
[1] 0.1353353

> 2*3*4*5
[1] 120

> pi    # R knows about pi
[1] 3.141593

> 1000*(1 + 0.075)^5 - 1000
[1] 435.6293
```

Assignments

It is often required to store intermediate results so that they do not need to be re-typed over and over again. To assign a value of 10 to the variable `x` type:

```
> x <- 10
```

and press Enter.

Can also use the command

```
> assign("x", 10)
```

There is no visible result, however `x` now has the value 10 and can be used in subsequent expressions.

```
> x  
[1] 10
```

```
> x + x  
[1] 20
```

```
> sqrt(x)  
[1] 3.162278
```

Case sensitivity and variable names

R is a case-sensitive language, e.g. x and X do not refer to the same variable.

Variable names:

- can be created using letters, digits and the . (dot) symbol, e.g. weight, wt.male
- must not start with a digit or a . followed by a digit.
- Some names are used by the system, e.g.
c, q, t, C, D, F, I, T, diff, df, pt - AVOID!

Objects

You cannot perform much statistics on single numbers. R works by creating different *objects* and using various function calls that create and use those objects.

- Vectors of
 - ▶ numbers
 - ▶ logical values
 - ▶ character strings
 - ▶ complex numbers
- Matrices and general n -way arrays
- Lists - arbitrary collections of objects of any type, e.g. list of vectors, list of matrices, etc.
- Data frames - lists with a rectangular structure
- Connections - connection to files and similar things
- Functions

Objects

During an R session, objects are created and stored by name. The command

```
> ls()
```

displays all currently-stored objects (*workspace*). Objects can be removed using

```
> rm(x, a, temp, wt.males)
```

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

removes all of the objects in the workspace.

At the end of each R session, you are prompted to save your workspace. If you click Yes, all objects are written to the `.RData` file. When R is re-started, it reloads the workspace from this file and the command history stored in `.Rhistory` is also reloaded.

Getting help in R

R has a built-in help facility. To get more information on any specific function, e.g. `sqrt()`, the command is

```
> help(sqrt)
```

An alternative is

```
> ? sqrt
```

Can also obtain help on features specified by special characters.

Must enclose in single or double quotes (e.g. "`["`")

```
> help("[")
```

Help is also available in HTML format by running

```
> help.start()
```

For more information use

```
> ? help
```

Packages

"R" contains one or more libraries of packages. Packages contain various functions and data sets for numerous purposes, e.g. survival package, genetics package, fda package, etc.

Some packages are part of the basic installation. Others can be downloaded from CRAN.

To access all of the functions and data sets in a particular package, it must be loaded into the workspace. For example, to load the fda package:

```
> library(fda)
```

One important thing to note is that if you terminate your session and start a new session with the saved workspace, you must load the packages again.

Packages

To check what packages are currently loaded into the workspace

```
> search()
[1] ".GlobalEnv"      "package:MASS"      "package:stats"
[4] "package:graphics" "package:grDevices" "package:utils"
[7] "package:datasets" "package:methods"   "Autoloads"
[10] "package:base"
```

Can remove a package you have loaded use:

```
> detach("package:fda")
```

An interactive session

Create a folder called Session 1 and copy an R shortcut into this folder. Right-click on this shortcut and go to Properties. Change the address in the **Start In** box to the location of your folder.

For the purposes of this session, a data set already stored in R will be used. To access this data, must first load the package containing the data. (R has many packages containing various functions that can be used to analyse data, e.g. if you want to analyse your data using splines, need to load the `splines` package). In this example, the data is stored in the MASS package. This is loaded with the command

```
> library(MASS)
```

Now have access to all functions and data sets stored in this package.

An interactive session

We will work with the data set titled “whiteside”. To display the data:

```
> whiteside
   Insul  Temp  Gas
1 Before -0.8  7.2
2 Before -0.7  6.9
3 Before  0.4  6.4
4 Before  2.5  6.0
5 Before  2.9  5.8
6 Before  3.2  5.8
7 Before  3.6  5.6
8 Before  3.9  4.7
9 Before  4.2  5.8
10 Before 4.3  5.2
```

This is a particular type of object called a data frame.

A full description of these data is found using

```
> ? whiteside
```

An interactive session

To remind ourselves of the names of the columns:

```
> names(whiteside)
[1] "Insul" "Temp" "Gas"
```

Summary statistics for each column are determined using

```
> summary(whiteside)
   Insul          Temp          Gas
  Before:26    Min.   :-0.800   Min.   :1.300
  After :30   1st Qu.: 3.050   1st Qu.:3.500
              Median : 4.900   Median :3.950
              Mean   : 4.875   Mean   :4.071
              3rd Qu.: 7.125   3rd Qu.:4.625
              Max.   :10.200   Max.   :7.200
```

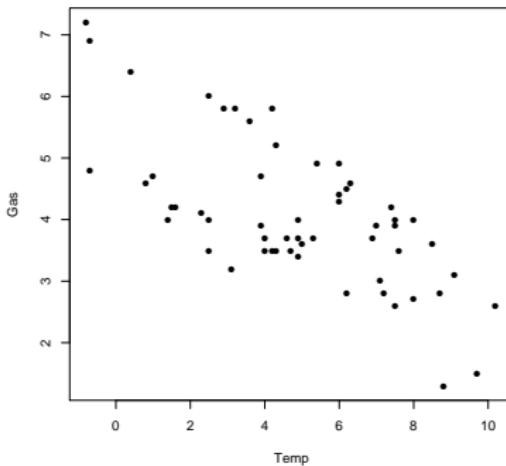
Access the data in a particular column

```
> whiteside$Temp
```

An interactive session

A plot of gas consumption versus temperature is now created.

```
> plot(Gas ~ Temp, data=whiteside, pch=16)
```

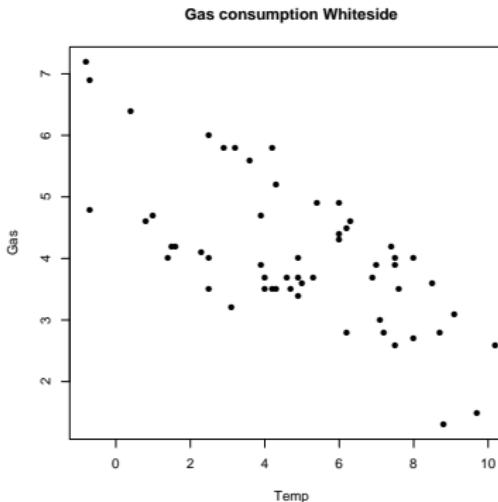


An interactive session

A title can be put on the graph

```
> plot(Gas ~ Temp, data=whiteside, pch=16, main="Gas consumption Whiteside")
```

Note: Do not need to re-type entire command. Press the up-arrow key to recall the last command. Edit this command to include `main="Gas consumption Whiteside"`, as above.



An interactive session

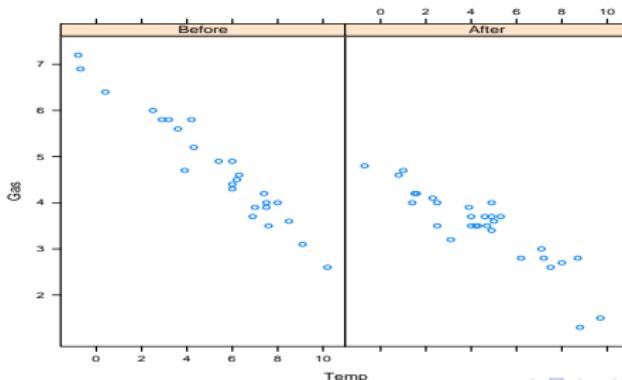
Can produce separate graphs for gas consumption versus temperature before insulation used and after insulation used.

Requires the use of `xyplot()` available in the lattice package.

Need to load this package into R before function can be used.

Then use `xyplot()`.

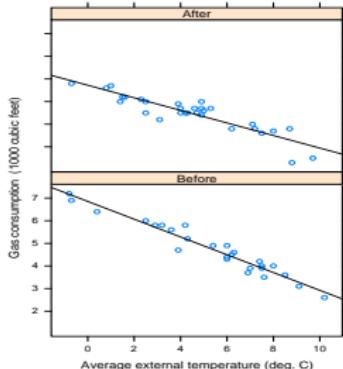
```
> library(lattice) # Loads the lattice package  
> ? xyplot # Gives more information on xyplot()  
> xyplot(Gas ~ Temp | Insul, whiteside)
```



An interactive session

More complex plot (can copy code from examples section of help file on whiteside data set obtained earlier)

```
> xyplot(Gas ~ Temp | Insul, whiteside, panel =  
+   function(x, y, ...) {  
+     panel.xyplot(x, y, ...)  
+     panel.lmline(x, y, ...)  
+   }, xlab = "Average external temperature (deg. C)",  
+   ylab = "Gas consumption (1000 cubic feet)", aspect = "xy",  
+   strip = function(...) strip.default(..., style = 1))  
>
```



An interactive session

Entry of data at the command line

Will now create a data frame with 2 columns. The following data gives, for each amount by which an elastic band is stretched over the end of a ruler, the distance that the band moved when released:

Stretch (mm)	Distance (cm)
46	148
54	182
48	173
50	166
44	109
42	141
52	166

An interactive session

Entry of data at the command line

Use the `data.frame()` command.

Name the data frame **elasticband**.

```
> elasticband <- data.frame(stretch = c(46,54,48,50,44,42,52),  
+ distance=c(148,182,173,166,109,141,166))  
>  
> elasticband  
  stretch  distance  
1       46        148  
2       54        182  
3       48        173  
4       50        166  
5       44        109  
6       42        141  
7       52        166
```

Exercises

1. Create summary statistics for the elastic band data.
2. Create a plot of distance versus stretch.
3. Use the `help()` command to find more information about the `hist()` command.
4. Create a histogram of the distance using `hist()`.
5. The following data are on snow cover for Eurasia in the years 1970-1979.

year	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979
snow.cover	6.5	12.0	14.9	10.0	10.7	7.9	21.9	12.5	14.5	9.2

- (i) Enter the data into R. To save keystrokes, enter the successive years as `1970:1979`.
- (ii) Take the logarithm of snow cover.
- (iii) Plot snow cover versus year.
6. Display all objects in the workspace. Remove the data frame **elasticband**.

Objects and simple manipulations

Vectors

Vectors are the simplest type of object in R. There are 3 main types of vectors:

- Numeric vectors
- Character vectors
- Logical vectors
- (Complex number vectors)

To set up a numeric vector x consisting of 5 numbers, 10.4, 5.6, 3.1, 6.4, 21.7, use

```
> x <- c(10.4, 5.6, 3.1, 6.4, 21.7)
```

or

```
> assign("x", c(10.4, 5.6, 3.1, 6.4, 21.7))
```

Numeric Vectors

To print the contents of x:

```
> x  
[1] 10.4 5.6 3.1 6.4 21.7
```

The [1] in front of the result is the index of the first element in the vector x.

To access a particular element of x

```
> x[1]  
[1] 10.4
```

```
> x[5]  
[1] 21.7
```

Can also do further assignments:

```
> y <- c(x, 0, x)
```

Creates a vector y with 11 entries (two copies of x with a 0 in the middle)

Numeric Vectors

Computations

- Computations are performed element-wise, e.g.

```
> 1/x  
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
```

- Short vectors are “recycled” to match long ones

```
> v <- x + y
```

Warning message:

In x + y : longer object length is not a multiple
of shorter object length

- Some functions take vectors of values and produce results of the same length:

sin, cos, tan, asin, acos, atan, log, exp, Arith, ...

```
> cos(x)
```

```
[1] -0.5609843 0.7755659 -0.9991352 0.9931849 -0.9579148
```

Numeric Vectors

Computations

- Some functions return a single value:

sum, mean, max, min, prod, ...

```
> sum(x)
[1] 47.2
>
> length(x)
[1] 5
>
> sum(x)/length(x)
[1] 9.44
>
> mean(x)
[1] 9.44
```

- Some functions are a bit special:

cumsum, sort, range, pmax, pmin, ...

Numeric Vectors

Complex Numbers

Care must be taken when working with complex numbers. The expression

```
> sqrt(-17)
[1] NaN
Warning message:
In sqrt(-17) : NaNs produced
```

gives NaN (i.e. Not a Number) and a warning but

```
> sqrt(-17+0i)
[1] 0+4.123106i
```

performs the calculations as complex numbers.

Generating Sequences I

R has a number of ways to generate sequences of numbers. These include:

- the colon ":", e.g.

```
> 1:10  
[1] 1 2 3 4 5 6 7 8 9 10
```

This operator has the highest priority within an expression, e.g. $2*1:10$ is equivalent to $2*(1:10)$.

- the `seq()` function. (Use `> ? seq` to find out more about this function).

```
> seq(1,10)
```

```
> seq(from=1, to=10)
```

```
> seq(to=10, from=1)
```

are all equivalent to `1:10`.

Generating Sequences II

Can also specify a step size (using `by=value`) or a length (using `length=value`) for the sequence.

```
> s1 <- seq(1,10, by=0.5)
> s1
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
[12] 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0
```

```
> s2 <- seq(1,10, length=5)
> s2
[1] 1.00 3.25 5.50 7.75 10.00
```

- the `rep()` function - replicates objects in various ways.

```
> s3 <- rep(x, 2)
> s3
[1] 10.4 5.6 3.1 6.4 21.7 10.4 5.6 3.1 6.4 21.7
[11] 10.4 5.6 3.1 6.4 21.7
```

```
> s4 <- rep(c(1,4),c(10,15))
> s4
[1] 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

Character Vectors

- To set up a character/string vector `z` consisting of 4 place names use

```
> z <- c("Canberra", "Sydney", "Newcastle", "Darwin")
```

```
> z <- c('Canberra', 'Sydney', 'Newcastle', 'Darwin')
```

- Can be concatenated using `c()`

```
> c(z, "Mary", "John")
```

```
[1] "Canberra"  "Sydney"   "Newcastle"  "Darwin"   "Mary"    "John"
```

- Lots of in-built functions in R to manipulate character vectors.

Logical Vectors

- A logical vector is a vector whose elements are TRUE, FALSE or NA.
- Are generated by *conditions*, e.g.
`> temp <- x > 13`

Takes each element of the vector `x` and compares it to 13.
Returns a vector the same length as `x`, with a value TRUE when the condition is met and FALSE when it is not.
- The logical operators are `>`, `>=`, `<`, `<=`, `==` for exact equality and `!=` for inequality.
- `&` and `|`
- `&&` and `||`

Missing Values

In some cases the entire contents of a vector may not be known.
For example, missing data from a particular data set.

A place can be reserved for this by assigning it the special value NA.

Can check for NA values in a vector x using the command

```
> is.na(x)
```

Returns a logical vector the same length as x with a value TRUE if
that particular element is NA.

```
> w <- c(1:10, rep(NA,4), 22)  
> is.na(w)
```

Indexing Vectors

Have already seen how to access single elements of a vector.

Subsets of a vector may also be selected using a similar approach.

- `> ex1 <- w[!is.na(w)]`

Stores the elements of the vector `w` that do NOT have the value `NA`, into `ex1`.

- `> ex2 <- w[1:3]`

Selects the first 3 elements of the vector `w` and stores them in the new vector `ex2`.

- `> ex3 <- w[-(1:4)]`

Using the `-` sign indicates that these elements should be *excluded*. This command excludes the first 4 elements of `w`.

```
> ex4 <- w[-c(1,4)]
```

In this case only the 1st and 4th elements of `w` are excluded.

Modifying Vectors

To alter the contents of a vector, similar methods can be used.

- Remember `x` has contents

```
> x  
[1] 10.4 5.6 3.1 6.4 21.7
```

For example, to modify the 1st element of `x` and assign it a value 5 use

```
> x[1] <- 5  
> x  
[1] 5.0 5.6 3.1 6.4 21.7
```

- The following command replaces any NA (missing) values in the vector `w` with the value 0

```
> w[is.na(w)] <- 0
```

Modifying Vectors

- Let

```
> y <- c(-1, -2, rep(0, 3), 7, 8, 9)
> y
[1] -1 -2  0  0  0  7  8  9
```

The following replaces any elements of y with a negative value with the corresponding positive value. (Note this is equivalent to using the in-built `abs()` function).

```
> y[y < 0] <- -y[y < 0]
> y
[1] 1 2 0 0 0 7 8 9
```

Factors

A factor is a special type of vector used to represent categorical data, e.g. gender, social class, etc.

- Stored internally as a numeric vector with values $1, 2, \dots, k$, where k is the number of levels.
- Can have either *ordered* and *unordered* factors.
- A factor with k levels is stored internally consisting of 2 items
 - (a) a vector of k integers
 - (b) a character vector containing strings describing what the k levels are.

Factors

Example

Consider a survey that has data on 200 females and 300 males. If the first 200 values are from females and the next 300 values are from males, one way of representing this is to create a vector

```
> gender <- c(rep("female", 200), rep("male", 300))
```

To change this into a factor

```
> gender <- factor(gender)
```

The factor gender is stored internally as

1	female
2	male

Each category, i.e. female and male, is called a level of the factor.

To determine the levels of a factor the function `levels()` can be used:

```
> levels(gender)
[1] "female" "male"
```

Factors

Example

Five people are asked to rate the performance of a product on a scale of 1-5, with 1 representing very poor performance and 5 representing very good performance. The following data were collected.

```
> satisfaction <- c(1, 3, 4, 2, 2)
> fsatisfaction <- factor(satisfaction, levels=1:5)
> levels(fsatisfaction) <- c("very poor", "poor", "average",
  "good", "very good")
```

The first line creates a numeric vector containing the satisfaction levels of the 5 people. Want to treat this as a categorical variable and so the second line creates a factor. The `levels=1:5` argument indicates that there are 5 levels of the factor. Finally the last line sets the names of the levels to the specified character strings.

Exercises I

Vectors and Factors

1. Create a vector x with the following entries

```
3 4 1 1 2 1 4 2 1 1 5 3 1 1 1 2 4 5 5 3
```

Check which elements of x are equal to 1 (Hint use == operator). Modify x so that all of the 1's are changed to 0's.

2. Create a vector y containing the elements of x that are greater than 1.
3. Create a sequence of numbers from 1 to 20 in steps of 0.2 and store.
4. Concatenate x and y into a vector called `newVec`.
5. Display all objects in the workspace and then remove `newVec` (see Lecture 1).

Exercises II

Vectors and Factors

6. Six patients were asked to rate their pain from 0 to 3, with 0 representing ‘no pain’, 1 representing ‘mild’ pain, 2 representing ‘medium’ pain and 3 representing ‘severe’ pain. The following results were obtained:

Patient	1	2	3	4	5	6
Pain level	0	3	1	2	1	2

Create a factor `fpain` to represent the above data.

Matrices

A matrix

- is a two-dimensional array of numbers;
- has rows and columns;
- is used for many purposes in statistics.

In R matrices are represented as vectors with dimensions.

```
> m <- rnorm(12) # Creates a vector of 12 random numbers
> m
[1] -0.32902981  0.64425299  0.91911540 -0.40675505  0.60745737 -0.06756709
[7]  0.38349915 -2.35291296  0.37126362 -1.33875464 -0.49121615 -1.55876372

> dim(m) <- c(3,4)
> m
      [,1]      [,2]      [,3]      [,4]
[1,] -0.3290298 -0.40675505  0.3834992 -1.3387546
[2,]  0.6442530  0.60745737 -2.3529130 -0.4912161
[3,]  0.9191154 -0.06756709  0.3712636 -1.5587637
```

Matrices

The `dim` function sets the *dimension* of `m`.

Causes R to treat the vector of 12 numbers as a 3×4 matrix.

Note that the storage is carried out by filling in the columns first, then the rows.

Another way to create a matrix is to use the `matrix()` function.

```
> n <- rnorm(10) # Generates a new vector of 10 random numbers
> matrix(n, nrow=5, ncol=2, byrow=T)
      [,1]      [,2]
[1,] 1.1202412 -0.6622302
[2,] 1.1286009  0.8751449
[3,] 1.2719938 -0.6243375
[4,] 0.7223669  0.8414961
[5,] 0.6330745  0.8950885
```

Matrices

The `byrow=T` command causes the matrix to be filled in row by row rather than column by column.

Re-call the last command and change `byrow=T` to `byrow="F"`. Notice the difference between the two outputs. This time the matrix is filled in column by column.

Useful functions for matrices include

`nrow()`, `ncol()`, `t()`, `rownames()`, `colnames()`.

```
> nrow(m)
[1] 3

> rownames(m) <- c("A", "B", "C")
> m
     [,1]      [,2]      [,3]      [,4]
A -0.3290298 -0.40675505  0.3834992 -1.3387546
B  0.6442530  0.60745737 -2.3529130 -0.4912161
C  0.9191154 -0.06756709  0.3712636 -1.5587637
```

Matrices

The `t()` function is the transposition function (rows become columns and vice versa).

```
> t(m)
      A          B          C
[1,] -0.3290298  0.6442530  0.91911540
[2,] -0.4067550  0.6074574 -0.06756709
[3,]  0.3834992 -2.3529130  0.37126362
[4,] -1.3387546 -0.4912161 -1.55876372
```

Can merge vectors and matrices together, column-wise or row-wise using `rbind()` (add on rows) or `cbind()` (add on columns).

When using `rbind()` - if combining matrices with other matrices, the matrices must have the same number of columns. If combining vectors with other vectors or vectors with matrices the vectors can have any length but will be lengthened/shortened accordingly if of differing lengths.

Matrices

When using `cbind()` - if combining matrices with other matrices, the matrices must have the same number of rows. If combining vectors with other vectors or vectors with matrices, the vectors can have any length but will be lengthened/shortened accordingly if of differing lengths.

A warning message is printed.

```
> X1 <- matrix(1:12, nrow=3, ncol=4, byrow=T)
> X2 <- matrix(20:27, nrow=2, ncol=4)
> rbind(X1,X2,n)
      [,1]      [,2]      [,3]      [,4]
1.000000  2.0000000  3.000000  4.0000000
5.000000  6.0000000  7.000000  8.0000000
9.000000 10.0000000 11.000000 12.0000000
20.000000 22.0000000 24.000000 26.0000000
21.000000 23.0000000 25.000000 27.0000000
n  1.120241 -0.6622302  1.128601  0.8751449
Warning message:
In rbind(X1, X2, n) :
  number of columns of result is not a multiple of vector length (arg 3)
```

Matrices

If the vector is too short, the values are re-cycled

```
> rbind(X1,X2,c(1,2))
      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   20   22   24   26
[5,]   21   23   25   27
[6,]    1    2    1    2
```

- Generate 2 new matrices X3 and X4 that have the same number of rows.

Use cbind() to combine the matrices column-wise.

Need to be careful when working with matrices. For example, if A and B are square matrices of the same size then

```
> A <- matrix(1:9, nrow=3, ncol=3)
> B <- matrix(10:18, nrow=3, ncol=3)
> C <- A %*% B # Calculates the product of two matrices, C = AB
> C <- A * B   # Calculates element by element products.
```

Matrices

Other functions to work on matrices include:

```
crossprod(A, B)    # = t(A) %*% B

diag(n)            # Creates a diagonal matrix with
                  # the values in the vector n on
                  # the diagonal

solve(C)           # Calculates the inverse of A

C^(-1)             # Calculates 1/c_ij

eigen(C)           # Calculates the eigenvalues and
                  # eigenvectors of C
```

Indexing Matrices

Say we have a 5×6 matrix

```
> X <- matrix(rnorm(30), nrow=5)
> dimnames(X) <- list(letters[1:5], LETTERS[1:6])
> X
```

	A	B	C	D	E	F
a	0.4233750	-0.6569585	0.67730663	0.4258022	-0.2003732	0.5934342
b	0.8045364	0.3983394	1.11778619	-0.2736120	0.5258172	-0.8595252
c	0.7479229	1.5591540	1.04614639	1.0419031	2.6454571	-0.2541819
d	0.6153824	-1.0451224	-0.03475772	0.9428584	0.3023099	0.2933696
e	0.8011791	-0.5668138	1.45136460	-0.3036993	0.2801525	-1.4701663

Can access the value in row 3, column 2 using

```
> X[3,2]                                > X["c", "B"]
[1] 1.559154                            [1] 1.559154
```

Indexing Matrices

Can also access multiple elements, e.g. we wish to extract

- elements in columns 2 and 4

```
> X[,c(2,4)]  
      B           D  
a -0.6569585  0.4258022  
b  0.3983394 -0.2736120  
c  1.5591540  1.0419031  
d -1.0451224  0.9428584  
e -0.5668138 -0.3036993
```

- elements in rows 2 to 4

```
> X[2:4,]  
      A           B           C           D           E           F  
b  0.8045364  0.3983394  1.11778619 -0.2736120  0.5258172 -0.8595252  
c  0.7479229  1.5591540  1.04614639  1.0419031  2.6454571 -0.2541819  
d  0.6153824 -1.0451224 -0.03475772  0.9428584  0.3023099  0.2933696
```

Indexing Matrices

- elements $X[1,3]$, $X[2,2]$ and $X[3,1]$ - easiest to create an index array

```
> index <- array(c(1:3, 3:1), dim=c(3,2))
> index
 [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1

> X[index]
[1] 0.6773066 0.3983394 0.7479229
```

- want to replace these elements by zero

```
> X[index] <- 0
```

	A	B	C	D	E	F
a	0.4233750	-0.6569585	0.00000000	0.4258022	-0.2003732	0.5934342
b	0.8045364	0.0000000	1.11778619	-0.2736120	0.5258172	-0.8595252
c	0.0000000	1.5591540	1.04614639	1.0419031	2.6454571	-0.2541819
d	0.6153824	-1.0451224	-0.03475772	0.9428584	0.3023099	0.2933696
e	0.8011791	-0.5668138	1.45136460	-0.3036993	0.2801525	-1.4701663

Arrays

An array can have multiple dimensions.

A matrix is a special case of an array (a 2-d array).

Can construct an array from a vector z containing 300 elements using the `dim()` function (as for matrices).

```
> z <- rnorm(300)  
> dim(z) <- c(10, 6, 5)
```

Creates a 3-d array with dimensions $10 \times 6 \times 5$ (like storing 5 matrices, each with 10 rows and 6 columns).

Can also use the `array()` function.

```
> A1 <- array(0, c(2, 2, 3))      # Creates an array of zeros  
  
> a <- rnorm(50)  
> A2 <- array(a, c(5, 5, 2))      # Creates an array from vector a
```

Use `? array` to find out more about arrays.

Indexing Arrays

Elements of multi-dimensional arrays can be extracted using similar techniques. For example

```
> arr.1 <- array(1:24, dim=c(4,2,3))
> arr.1[2,,]      # Extracts the data in row 2 of the 3 'matrices'.
>
> arr.1[,2,]      # Extracts the data in column 2 of the 3 'matrices'.
>
> arr.1[,,1]      # Extracts the data in the first 'matrix'.
>
> arr.1[1,2,3]    # Extracts the data in the row 1, column 2 of the
# third 'matrix'.
```

Exercises I

Matrices and Arrays

1. Construct a matrix A with values 10, 20, 30, 50 in column 1, values 1, 4, 2, 3 in column 2 and values 15, 11, 19, 5 in column 3, i.e. a 4×3 matrix. Also construct a vector B with values 2.5, 3.5, 1.75. Check your results to ensure that they are correct.
2. Combine A and B into a new matrix C using `cbind()`.
3. Combine A and B into a new matrix H using `rbind()`.
4. Determine the dimensions of C and H using `dim()` function.
5. Calculate the following:

$$\begin{pmatrix} 1 & 4 & 3 \\ 0 & -2 & 8 \end{pmatrix} \times \begin{pmatrix} 1 & 9 \\ 2 & 17 \\ -6 & 3 \end{pmatrix}$$

Exercises II

Matrices and Arrays

6. Create a $4 \times 4 \times 2$ array `arr` using the values 1 to 32.
7. Print out the value in row 1, column 3 of the first ‘matrix’.
8. Print out the value in row 2, column 4 of the second ‘matrix’.
9. Add these two values together.

Lists

Lists

- are an ordered collection of components;
- components may be arbitrary R objects (data frames, vectors, lists, etc.);
- single bracket notation for sublists;
- double bracket notation for individual components;
- construct using the function `list()`.

A simple example of a list is as follows:

```
> L1 <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
```

Each component of the list is given a name (i.e. `name`, `wife`, `no.children`, `child.ages`).

Lists

Construct a second list omitting the component names:

```
> L2 <- list("Fred", "Mary", 3, c(4,7,9))
```

What is the difference between the two?

Three equivalent ways of accessing the first component

```
> L1[["name"]]  
[1] "Fred"
```

```
> L1$name  
[1] "Fred"
```

```
> L1[[1]]  
[1] "Fred"
```

A sublist consisting of the first component only

```
> L1[1]  
$name  
[1] "Fred"
```

Lists

The names of each component of the list can be accessed using

```
> names(L1)
[1] "name"      "wife"       "no.children"    "child.ages"
```

```
> names(L2)
NULL
```

Can set the names for the list components after the list has been created.

```
> names(L2) <- c("name.hus", "name.wife", "no.child", "child.age")
> names(L2)
[1] "name.hus"      "name.wife"      "no.child"      "child.age"
```

Can also concatenate lists:

```
> L3 <- c(L1, L2)
```

Exercises

Lists

1. Create 4 vectors Year, mean_weight, Gender and mean_height with the following entries:

Year	1980	1988	1996	1998	2000	2002
mean_weight	71.5	72.1	73.7	74.3	75.2	74.7
Gender	M	M	F	F	M	M
mean_height	179.3	179.9	180.5	180.1	180.3	180.4

2. Create a list called myList consisting of the above vectors.
Give each component of the list a name.
3. Use 3 different ways to access the 4th element of the list.

Data Frames

A data frame

- can be thought of as a data matrix or data set;
- is a generalisation of a matrix;
- is a list of vectors and/or factors of the same length;
- has a unique set of row names.
- Data in the same position across columns come from the same experimental unit.

Can create data frames from pre-existing variables:

```
> d <- data.frame(mean_weight, Gender)
```

This is the same as re-typing

```
> d <- data.frame(mean_weight=c(71.5,72.1,73.7,74.3,75.2,74.7),  
+ Gender=c("M", "M", "F", "F", "M", "M"))
```

Data Frames

Can also convert other objects (e.g. lists, matrices) into a data frame.

In the previous exercises you created a list called `mylist`.

To convert this to a data frame:

```
> new.data <- as.data.frame(mylist)
```

Note that the data in each row are related, that is the same person is male, has a birth date of 1980, a weight of 71.5 kg and a height of 179.3 cm.

As with lists, individual components (columns) can be accessed using the `$` notation.

```
> new.data$year  
[1] 1980 1988 1996 1998 2000 2002
```

Indexing Data Frames

Have already seen how to access individual/sets of values in vectors, matrices and arrays.

It is possible to use the same methods to access values of a data frame. Makes use of the matrix-like structure.

```
> new.data
  year mean_weight gender mean_height
1 1980      71.5       M      179.3
2 1988      72.1       M      179.9
3 1996      73.7       F      180.5
4 1998      74.3       F      180.1
5 2000      75.2       M      180.3
6 2002      74.7       M      180.4

> new.data[3,2]
[1] 73.7
```

gives the value in the 3rd row and 2nd column of new.data.

Indexing Data Frames

```
> new.data[,2]  
[1] 71.5 72.1 73.7 74.3 75.2 74.7
```

returns all the measurements in the 2nd column (same as using new.data\$mean_weight).

```
> new.data[3,]  
  year mean_weight gender mean_height  
3 1996       73.7      F        180.5
```

returns all measurements for the 3rd individual.

NB - Comma!!

Indexing Data Frames

Other indexing techniques also apply. For example, selecting all data for cases that satisfy some criterion, such as the data for all males.

```
> new.data[new.data$gender == "M",]
```

Selects the rows of the data frame where gender is male. Note that the row names are the same as those in the original data frame.

To select only the weight and height of females born after 1996 use:

```
> new.data[new.data$gender == "F" & new.data$year > 1996, c(2,4)]  
    mean_weight  mean_height  
4           74.3       180.1
```

Indexing Data Frames

The first two logical commands

`new.data$gender == "F" & new.data$year > 1996` dictate which rows to select from the data frame (the `&` tells R that BOTH conditions must be satisfied).

The `c(2,4)` dictates which columns to select (in this case columns 2 and 4).

Replacing the `&` with a `|` selects the rows that satisfy EITHER condition.

Re-type the last command and replace the `&` with a `|`. Notice how the results differ.

```
> new.data[new.data$gender == "F" | new.data$year > 1996, c(2,4)]  
  mean_weight  mean_height  
3       73.7      180.5  
4       74.3      180.1  
5       75.2      180.3  
6       74.7      180.4
```

attach() and detach()

It can be time consuming to access the variables in a data frame if you need to repeatedly use long commands like

```
> new.data[new.data$gender == "F" | new.data$year > 1996, c(2,4)]
```

The attach() command

```
> attach(new.data)
> search()
```

places the data frame new.data onto the search path and forces R to look for objects among the variables in new.data.

No need to use the \$ notation.

```
> new.data[gender == "F" | year > 1996, c(2,4)]
```

attach() and detach()

Must be careful that there are no other objects in the workspace with the same names as the names of the columns in the data frame - R will get confused.

If you want to make changes to the data frame, must still use the \$ notation.

To detach the data frame

```
> detach(new.data)  
> search()
```

Exercises I

Data Frames

1. Create a data frame called `club.points` with the following data.

Firstname	Lastname	Age	Gender	Points
Alice	Ryan	37	F	278
Paul	Collins	34	M	242
Jerry	Burke	26	M	312
Thomas	Dolan	72	M	740
Marguerite	Black	18	F	177
Linda	McGrath	24	F	195

2. Store the points for every person into a vector called `pts`, then calculate the average number of points received. (Hint use `mean()` function).
3. Store the data for the females only into a data frame called `fpoints`.

Exercises II

Data Frames

4. The age for Jerry Burke was entered incorrectly. Change his age to 28.
5. Determine the maximum age of the males.
6. Extract the data for people with more than 100 points and are over the age of 30.

The apply Family

Sometimes want to apply a function to each element of a vector/data frame/list/array.

Four members: `lapply`, `sapply`, `tapply`, `apply`

- `lapply`: takes any structure and gives a list of results (hence the 'l')
- `sapply`: like `lapply`, but tries to simplify the result to a vector or matrix if possible (hence the 's')
- `apply`: only used for arrays/matrices
- `tapply`: allows you to create tables (hence the 't') of values from subgroups defined by one or more factors.

Used for

- efficiency relative to explicit loops
- convenience

The apply Family

lapply and sapply

Will use an in-built data set called `trees`. Gives girth, height and volume measurements for 31 trees.

To calculate the mean of each variable in `trees`

```
> lapply(trees, mean)
```

```
$Girth
```

```
[1] 13.24839
```

```
$Height
```

```
[1] 76
```

```
$Volume
```

```
[1] 30.17097
```

```
> sapply(trees, mean)
```

Girth	Height	Volume
-------	--------	--------

13.24839	76.00000	30.17097
----------	----------	----------

The apply Family

apply

The second argument is the function that we want to compute (can also write your own functions) and we can also specify other arguments, e.g. `na.rm=T` to remove NA values from the calculations.

To apply a function to either the row/columns of a matrix

```
> X1 <- matrix(1:12, nrow=3)
> apply(X1, 1, sum)
[1] 22 26 30
```

Calculates the sum of the values in each row. The second argument is an index (or vector of indices) dictating the dimension to apply the function to.

```
> apply(X1, 2, mean)
[1]  2  5  8 11
```

Calculates the mean of the values in each column.

The apply Family

tapply

Type the following code into R

```
> library(MASS)          # Allows R to use stored functions and data in this
                           # library
> Cars93                 # Data set stored in this library
   Manufacturer      Model     Type Min.Price Price Max.Price MPG.city
1       Acura        Integra  Small     12.9  15.9    18.8     25
2       Acura      Legend Midsize    29.2  33.9    38.7     18
3      Audi           90 Compact    25.9  29.1    32.3     20
4      Audi          100 Midsize    30.8  37.7    44.6     19
5       BMW        535i Midsize    23.7  30.0    36.2     22
```

Manufacturer is a factor

```
> is.factor(Cars93$Manufacturer)
[1] TRUE
```

and we want to calculate the average price of a car for each manufacturer.

```
> tapply(Cars93$Price, Cars93$Manufacturer, mean)
  Acura        Audi        BMW       Buick      Cadillac
  24.90000    33.40000   30.00000  21.62500   37.40000
```

Reading Data from Files

Sometimes data can be stored in external files like text files, Excel files etc. R provides several functions to read data in from such files.

- `scan()` - offers a low-level reading facility
- `read.table()` - can be used to read data frames from formatted text files
- `read.csv()` can be used to read data frames from comma separated variable files.
- When reading from Excel files, the simplest method is to save each worksheet separately as a .csv file and use `read.csv()` on each. A better way is to open a data connection to the excel file directly and use the ODBC facilities.

Reading Data from Files

General rules for storing data in external files:

- Use tabs as separators
- Each row has to have the same number of columns
- Missing data is NA, not empty
- As .txt: The ideal format!
- As .xls: save as tabbed text

Reading Data from Files

`read.table()`

Save the file `example.txt` into the same directory/folder as your R session (`.RData` file).

```
> example.data <- read.table("example.txt", header=TRUE)
```

Note that the variable names, “`x1`”, “`x2`”, and “`y`”, were in the first line of the data file, hence the `header=TRUE` command. This tells R that the column names in the text file should be used as the variable names in `example.data`.

Check whether `example.data` is a matrix, data frame, list...
(Hint: use `is.matrix()`, `is.data.frame()`, etc.)

If the file is in a different directory/folder than your R session, specify a full file path

```
> example.data <- read.table("C:/.../Desktop/example.txt", header=TRUE)
```

Writing Data to Files

May also want to save your output in an external file. Use `write.table()` or `write.csv()`.

```
> write.table(example.data, "Ex1.txt", row.names=FALSE, sep=" ")
```

Writes the data in `example.data` to a text file called “`Ex1.txt`” which is in the same folder as your R session. (Can also specify a `filepath`)

The `row.names=FALSE` command ensures that the row numbers are not saved in the file. (Exercise: re-call the last command and change `row.names=FALSE` to `row.names=TRUE`.)

The `sep=" "` command ensures that the output is separated by a space. Can change this using

```
> write.table(example.data, "Ex1.txt", row.names=FALSE, sep=", ")
```

Output is now separated by commas.

Writing Data to Files

To write data to an Excel file

```
> write.csv(example.data, "Ex2.csv", row.names=FALSE)
```

For more information on importing and exporting data in R refer to the *R Data Import/Export* manual available online.

Exercises

Reading Data from Files

1. Download the example2 data and save.
2. Read this data into R.
3. Print out the data for cases 10 to 18.
4. Print out the data for column 2, cases 23, 2, and 5 (in that order).
5. Find the mean, standard deviation, minimum and maximum for each variable using the smallest number of commands possible.

Scripting

So far we have entered commands into R on a line by line basis. This is often not very practical, e.g. if you enter a command over 5 lines you need to use the up-arrow 5 times to re-enter it.

As a result, R scripts are used. These are essentially text files containing collections of R commands that can be copied and pasted into the R console.

Can use Notepad, Wordpad, etc. but there are some specialist text editors that work specifically for R, e.g. TinnR (can be downloaded from <http://sourceforge.net/projects/tinn-r>) and RWinEdt (used with LaTex and WinEdt - can be downloaded from CRAN).

Scripting

We will use RWinEdt since WinEdt is available to us.

Go to <http://cran.r-project.org/>.

Click on the Packages link on the left hand side of the page. This gives a list of all packages available for download.

Find RWinEdt package and click on the link. Then click on RWinEdt_1.8-0.zip and click Save. Choose a place to save the zip file.

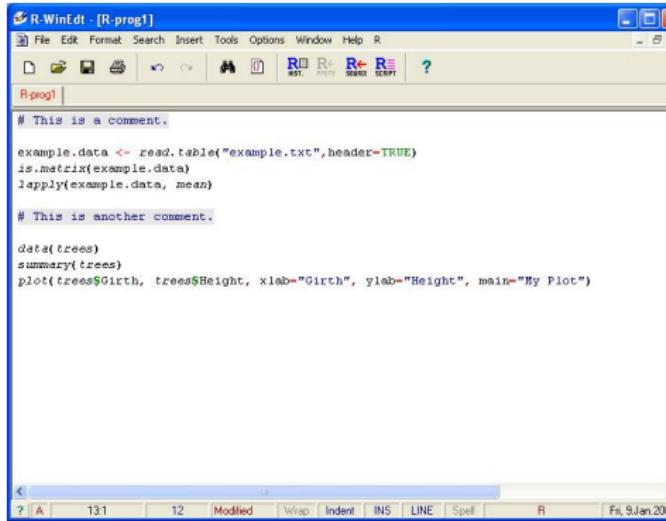
Go into the R console and click on
Packages > Install package(s) from local zip files..
and navigate to where you saved the zip file. Click on it and click Open.

```
> utils:::menuInstallLocal()  
package 'RWinEdt' successfully unpacked and MD5 sums checked  
updating HTML package descriptions
```

Scripting

To load RWinEdt

```
> library(RWinEdt)
```



The screenshot shows the R-WinEdt application window titled "R-WinEdt - [R-prog1]". The menu bar includes File, Edit, Format, Search, Insert, Tools, Options, Window, Help, and R. Below the menu is a toolbar with icons for file operations and R-related functions. The main editor area contains the following R code:

```
# This is a comment.  
example.data <- read.table("example.txt", header=TRUE)  
is.matrix(example.data)  
lapply(example.data, mean)  
  
# This is another comment.  
  
data(trees)  
summary(trees)  
plot(trees$Girth, trees$Height, xlab="Girth", ylab="Height", main="My Plot")
```

The status bar at the bottom shows the file name "R-prog1.R", the date "Fri, 9 Jan 2007", and various text editing options like wrap, indent, and spell.

To begin typing in commands click on File > New and type in the commands you want to use.

Scripting

The file can be saved to some location on your computer for use later. No need to re-type all commands you used during your session. Gets a .R extension.

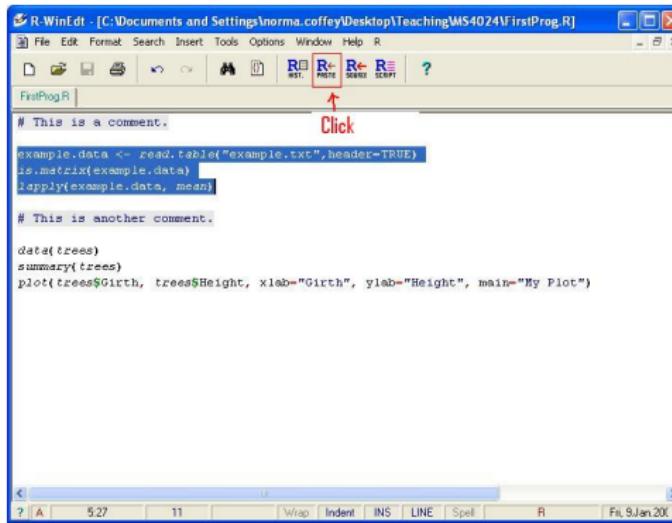
Can copy and paste commands from the script file to the R console.
OR can get R to carry out commands directly from RWinEdt.

To send all commands in the script file, click on



Scripting

To send only some selected commands, highlight the lines you want to use and then click the R Paste button.



Only the highlighted commands are evaluated by R.

Simulation and Probability Distributions

Save the file ProbDist.R to your current working folder.

Load RWinEdt if it is not already open (`library(RWinEdt)`).
Open the file ProbDist.R.

It is possible in R to create random samples from a particular population.

To pick a random sample from a set of numbers use the `sample()` command. See ProbDist.R.

Can also evaluate the density function, the cumulative distribution, the quantile function and create a random sample from a particular distribution distribution.

Simulation and Probability Distributions

Distribution	R name	Additional Args
beta	beta	shape1, shape2, ncp
binomial	binom	size, prob
Cauchy	cauchy	location, scale
chi-squared	chisq	df, ncp
exponential	exp	rate
F	f	df1, df2, ncp
...
normal	norm	mean, sd
Poisson	pois	lambda
Student's t	t	df, ncp
uniform	unif	min, max
Weibull	weibull	shape, scale
Wilcoxon	wilcox	m, n

Simulation and Probability Distributions

- The prefix:
 - ▶ p: probabilities (cumulative distribution)
 - ▶ q: quantiles (percentage points)
 - ▶ d: density functions (probability for discrete RVs)
 - ▶ r: random (or simulated) values
- The stems:
 - ▶ norm: Normal (Gaussian)
 - ▶ t, chisq, f: Normal test distributions
 - ▶ unif: Uniform distribution, by default [0, 1] range
 - ▶ gamma, cauchy, etc. : various specials
 - ▶ binom, pois, negbin, etc. : various discrete

Simulation and Probability Distributions

One problem when generating random numbers using `sample` or the prefix `r`, is that each time you take a sample, different numbers will be produced. If you want to sample the same numbers, use a command called `set.seed`.

For example,

```
set.seed(9)
sample(1:10, 4)
[1] 3 1 2 8
```

If we want to draw the same numbers again, need to set the seed and then re-draw the sample.

```
set.seed(9)
sample(1:10, 4)
[1] 3 1 2 8
```

If you do not use the `set.seed` function before re-drawing the sample, different numbers will be produced.

```
sample(1:10, 4)
[1] 5 2 4 3
```

Exercises

Simulation and Probability Distributions

1. Draw a random sample of size 100 from the interval [0,2] which contains 200 values. Sample without replacement.
2. Use dt to evaluate the density function of the t distribution with 13 degrees of freedom at 20 values in the range -1 to 1.
3. Find $P[X \leq x] = 0.01$ for a t distribution with 9 degrees of freedom.
4. IQ scores are known to have a normal distribution with mean 100 and standard deviation 15. What IQ would you have if you were in the 80th percentile?
5. What IQ would you have if you were in the top 10 percent?
6. What is the probability of having an IQ above 142?
7. Set the seed to "0" and create two samples of size 20 from the standard normal distribution with the same values. Repeat the process but set the seed to your ID number.

Plotting

Plotting

- Simple plotting:
 - ▶ `plot`, `hist`, `pairs`, `boxplot`, ...
- Adding to existing plots:
 - ▶ `points`, `lines`, `abline`, `legend`, `title`, `mtext`, ...
- Interacting with graphics:
 - ▶ `locator`, `identify`
- Three dimensional data:
 - ▶ `contour`, `image`, `persp`, ...
- To see the many possibilities that R offers
 - > `demo(graphics)`

Plotting

Basic plotting function is `plot()`. Possible arguments to `plot()` include:

- `x, y` – basic arguments (`y` may be omitted)
- `xlim = c(lo, hi), ylim = c(lo, hi)` – make sure the plotted axes ranges include these values
- `xlab = "x", ylab = "y"` – labels for x- and y-axes respectively
- `type = "c"` – type of plot (`p, l, b, h, S, ...`)
- `lty = n` – line type (if lines used)
- `lwd = n` – line width
- `pch = v` – plotting character(s)
- `col = v` – colour to be used for everything.

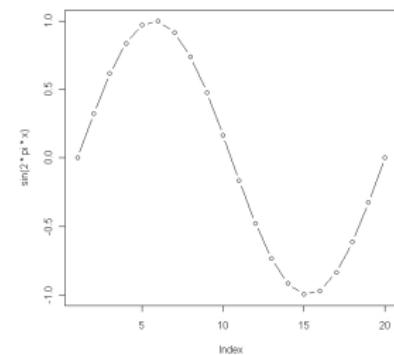
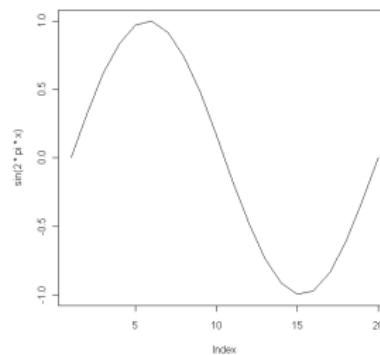
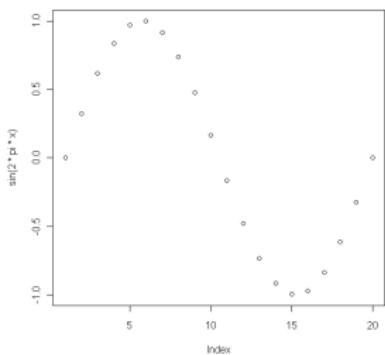
Various examples can be found in the file `Plots.R` in RWinEdt.

Plotting

```
x <- seq(0,1,length=20)
plot(sin(2*pi*x))          # Points

plot(sin(2*pi*x), type="l")    # Lines

plot(sin(2*pi*x), type="b")    # Points and lines
```

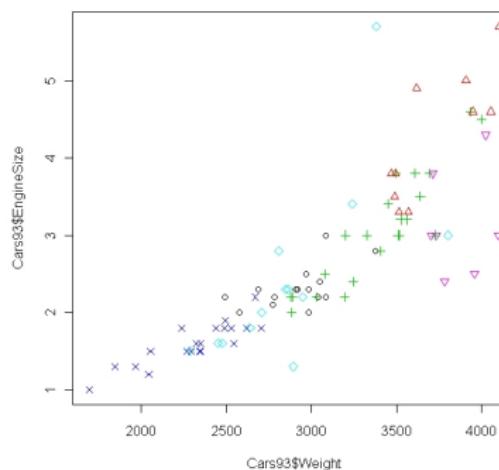


Plotting

Adding titles, lines, points

```
library(MASS)

# Colour points and choose plotting symbols according to a
# levels of a factor
plot(Cars93$Weight, Cars93$EngineSize, col=as.numeric(Cars93>Type),
      pch=as.numeric(Cars93>Type))
```



Plotting

Adding titles, lines, points

```
# Adds x and y axes labels and a title.  
plot(Cars93$Weight, Cars93$EngineSize, ylab="Engine Size",  
      xlab="Weight", main="My plot")  
  
# Add lines to the plot.  
lines(x=c(min(Cars93$Weight), max(Cars93$Weight)), y=c(min(Cars93$EngineSize),  
      max(Cars93$EngineSize)), lwd=4, lty=3, col="green")  
  
abline(h=3, lty=2)  
abline(v=1999, lty=4)  
  
# Add points to the plot.  
points(x=min(Cars93$Weight), y=min(Cars93$EngineSize), pch=16, col="red")
```

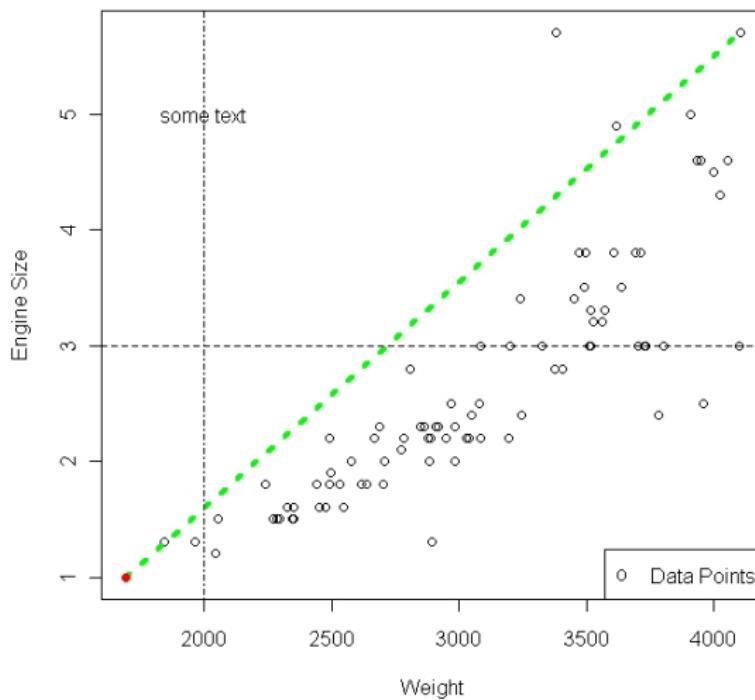
Plotting

Adding titles, lines, points

```
# Add text to the plot.  
text(x=2000, y=5, "some text")  
  
# Add text under main title.  
mtext(side=3, "sub-title", line=0.45)  
  
# Add a legend  
legend("bottomright", legend=c("Data Points"), pch="o")
```

My plot

sub-title



Plotting

Adding regression lines

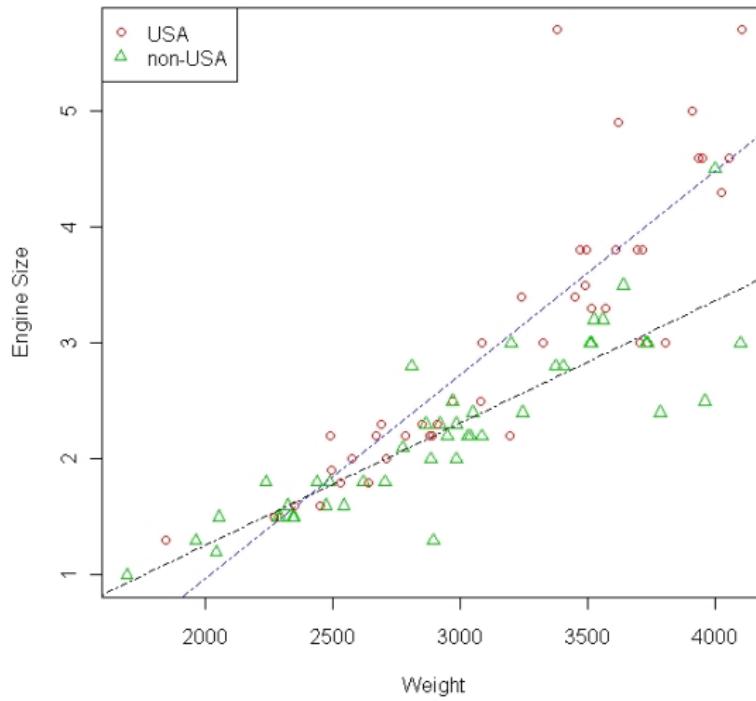
```
levels(Cars93$Origin)
[1] "USA"      "non-USA"

plot(Cars93$Weight, Cars93$EngineSize, pch = (1:2)[Cars93$Origin],
col = (2:3)[Cars93$Origin], xlab="Weight", ylab="Engine Size")
legend("topleft", legend=levels(Cars93$Origin), pch=1:2, col=2:3)

fm1 <- lm(EngineSize ~ Weight, Cars93, subset = Origin == "USA")
abline(coef(fm1), lty=4, col="blue")

fm2 <- lm(EngineSize ~ Weight, Cars93, subset = Origin == "non-USA")
abline(coef(fm2), lty=4, col="black")
```

Plotting



Plotting

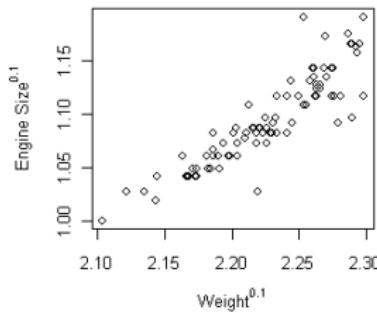
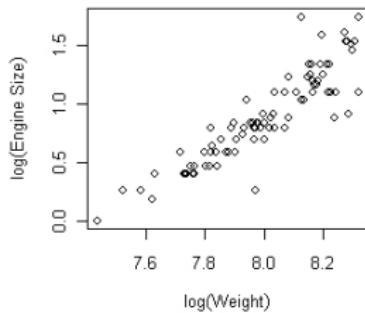
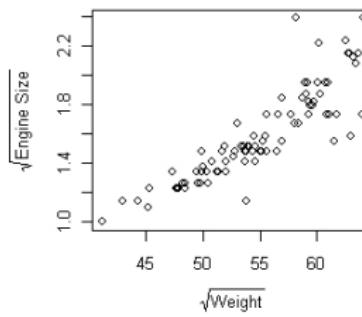
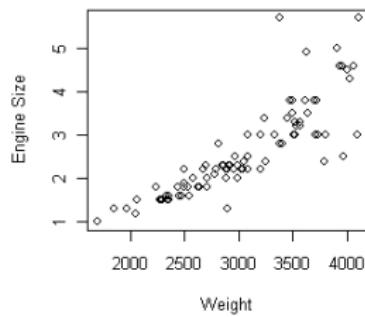
Multiple graphs

```
par(mfrow=c(2,2))      # Will create 4 plots on the same page.  
                      # Two in each row and two in each column.  
  
plot(Cars93$Weight, Cars93$EngineSize, xlab="Weight", ylab="Engine Size")  
  
plot(sqrt(Cars93$Weight), sqrt(Cars93$EngineSize),  
     xlab=expression(sqrt(Weight)), ylab=expression(sqrt("Engine Size")))  
  
plot(log(Cars93$Weight), log(Cars93$EngineSize),  
     xlab=expression(log(Weight)), ylab=expression(log("Engine Size")))  
  
plot(Cars93$Weight^0.1, Cars93$EngineSize^0.1,  
     xlab=expression(Weight^0.1), ylab=expression("Engine Size"^-0.1) )  
  
par(mfrow=c(1,1))      # Resets to create a single plot per page.
```

The `expression` command plots mathematical symbols on the x and y axes. For more information on `expression`

? `plotmath`

Plotting



Plotting

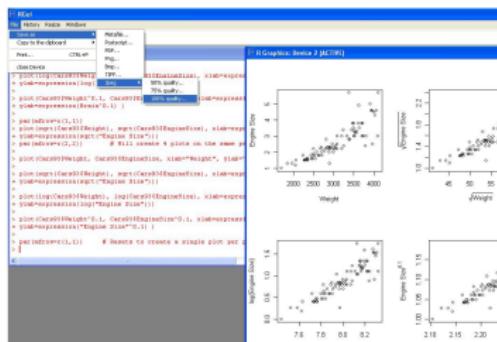
Saving plots

First need to make sure the graphics device is active by clicking on it.

Then click File > Save As > .

Get a number of options...

Mostly use Postscript, PDF and Jpeg.



Plotting

Histograms

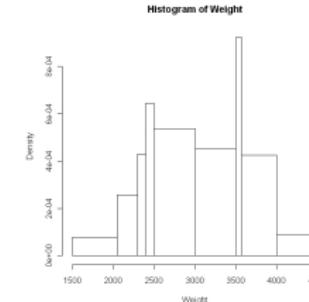
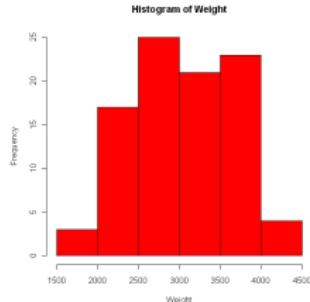
Histograms can be created using the `hist` command.

To create a histogram of the car weights from the Cars93 data set

```
hist(Cars93$Weight, xlab="Weight", main="Histogram of Weight", col="red")
```

R automatically chooses the number and width of the bars. Can change this by specifying the location of the break points.

```
hist(Cars93$Weight, breaks=c(1500, 2050, 2300, 2350, 2400,  
2500, 3000, 3500, 3570, 4000, 4500), xlab="Weight",  
main="Histogram of Weight")
```



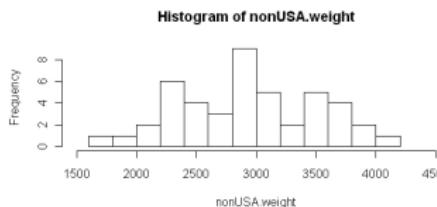
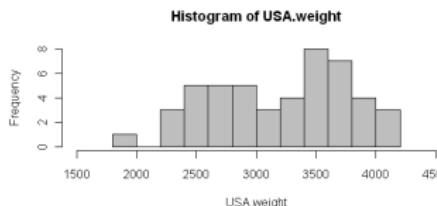
Plotting

Histograms

Histograms for multiple groups.

```
USA.weight <- Cars93$Weight[Cars93$Origin == "USA"]
nonUSA.weight <- Cars93$Weight[Cars93$Origin == "non-USA"]

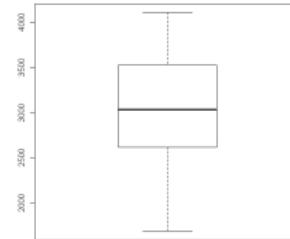
par(mfrow=c(2,1))
hist(USA.weight, breaks=10, xlim=c(1500,4500), col="grey")
hist(nonUSA.weight, breaks=10, xlim=c(1500,4500))
par(mfrow=c(1,1))
```



Plotting

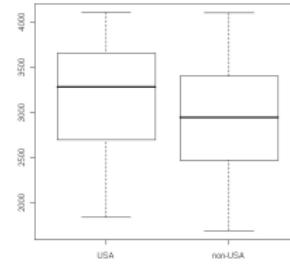
Boxplots

```
boxplot(Cars93$Weight)
```



```
boxplot(Cars93$Weight ~ Cars93$Origin)
```

```
boxplot(USA.weight, nonUSA.weight,  
names=c("USA", "non-USA"))
```

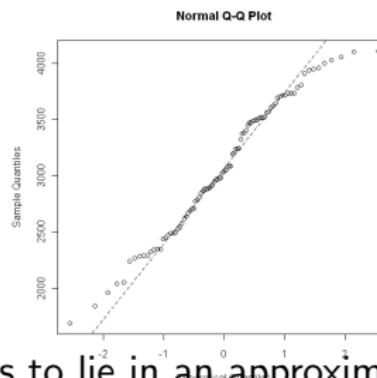


Plotting

Normal probability (Q-Q) plots

To check that data are normally distributed:

```
qqnorm(Cars93$Weight)  
qqline(Cars93$Weight)
```



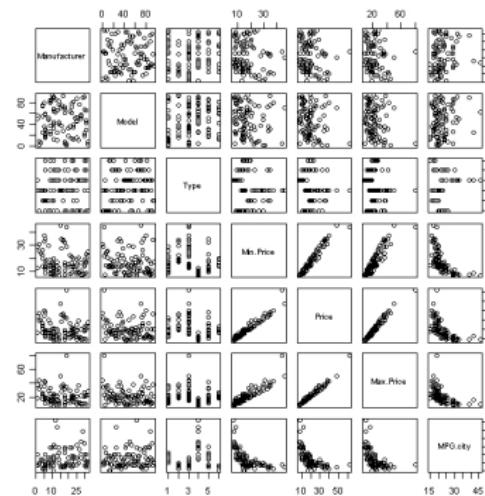
Want all of the points to lie in an approximate straight line (along the dotted line) for a normal distribution.

Plotting

Plots for multivariate data

If your data are stored in a data frame with several columns, the `pairs` command produces pairwise plots of the data in each column, i.e. the data in column 1 vs the data in column 2, column 1 vs column 3, and so on.

```
pairs(Cars93[,1:7])
```

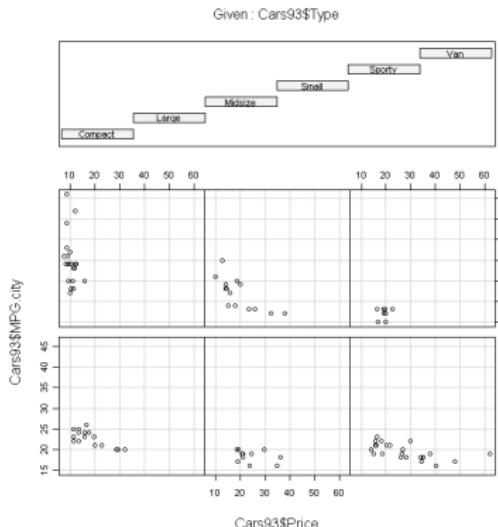


Plotting

Plots for multivariate data

coplot is also useful if you have 3 or 4 variables. For example if a and b are numeric vectors and c is a numeric vector or factor, the command `coplot(a ~ b | c)` produces plots of the values of a versus b for every level of c.

```
coplot(Cars93$MPG.city~Cars93$Price|Cars93>Type)
```



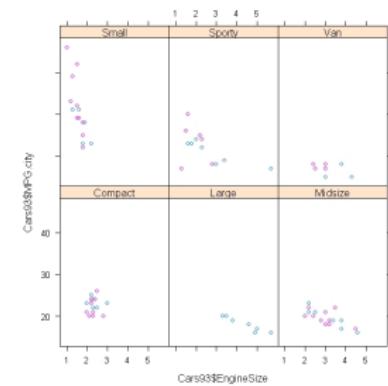
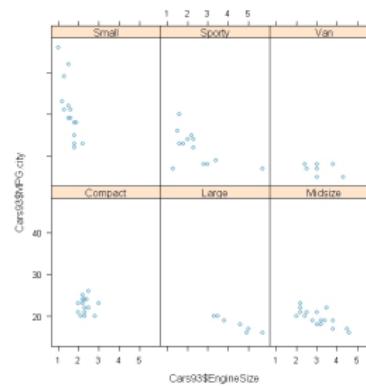
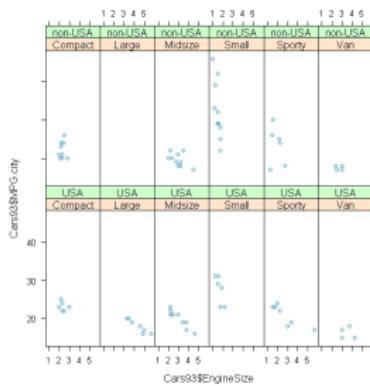
Plotting

Lattice graphs

```
xyplot(Cars93$MPG.city~Cars93$EngineSize|Cars93>Type)
```

```
xyplot(Cars93$MPG.city~Cars93$EngineSize|Cars93>Type*Cars93$Origin)
```

```
xyplot(Cars93$MPG.city~Cars93$EngineSize|Cars93>Type,
panel=panel.superpose, groups=Cars93$Origin)
```



Plotting

Lattice graphs

Can examine the plotting parameters in `xyplot`.

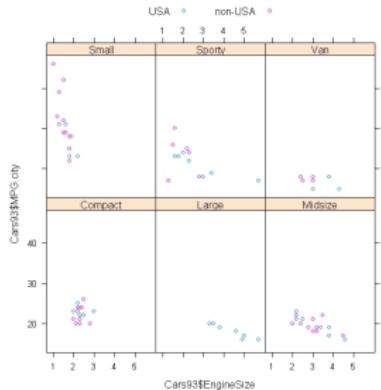
```
pars <- trellis.par.get("superpose.symbol")
pars
$alpha
[1] 1 1 1 1 1 1 1
$cex
[1] 0.8 0.8 0.8 0.8 0.8 0.8 0.8
$col
[1] "#0080ff"   "#ff00ff"   "darkgreen" "#ff0000"   "orange"    "#00ff00"
[7] "brown"
$fill
[1] "#CCFFFF" "#FFCCFF" "#CCFFCC" "#FFE5CC" "#CCE6FF" "#FFFFCC" "#FFCCCC"
$font
[1] 1 1 1 1 1 1 1
$pch
[1] 1 1 1 1 1 1 1
```

Plotting

Lattice graphs

Can use these to add a key to the plot:

```
# Adds a key  
xyplot(Cars93$MPG.city~Cars93$EngineSize|Cars93$Type,  
panel=panel.superpose, groups=Cars93$Origin, key =  
list(columns = 2, text = list(levels(Cars93$Origin)),  
points = Rows(pars,1:2)))
```



Plotting

Other lattice plots

```
splom( ~ data.frame) # Scatterplot matrix  
bwplot(factor ~ numeric, ...) # Boxplot  
qqmath(factor ~ numeric, ...) # Q-Q plot  
dotplot(factor ~ numeric, ...) # 1-D display  
stripplot(factor ~ numeric, ...)  
barchar(character ~ numeric, ...)  
histogram( ~ numeric, ...)  
densityplot( ~ numeric, ...) # Smoothed version of histogram
```

Plotting

2-D and 3-D plots

```
? volcano
data(volcano)
x <- 10*(1:nrow(volcano))
y <- 10*(1:ncol(volcano))

# Creates a 2-D image of x and y co-ordinates.
image(x, y, volcano, col = terrain.colors(100),
axes = FALSE)

# Adds contour lines to the current plot.
contour(x, y, volcano, levels = seq(90, 200, by=5),
add = TRUE, col = "peru")

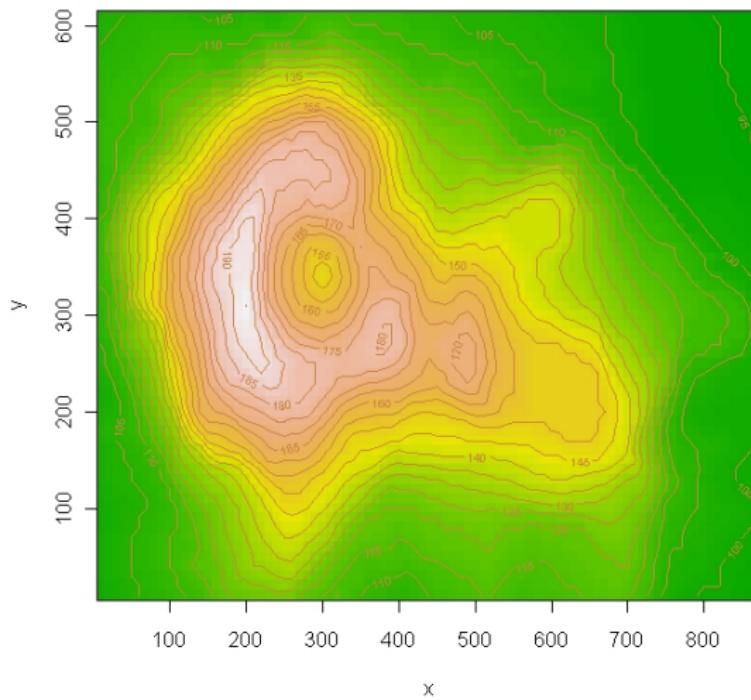
# Adds x and y axes to the plot.
axis(1, at = seq(100, 800, by = 100))
axis(2, at = seq(100, 600, by = 100))

# Draws a box around the plot.
box()

# Adds a title.
title(main = "Maunga Whau Volcano", font.main = 4)
```

Plotting

Maunga Whau Volcano



Plotting

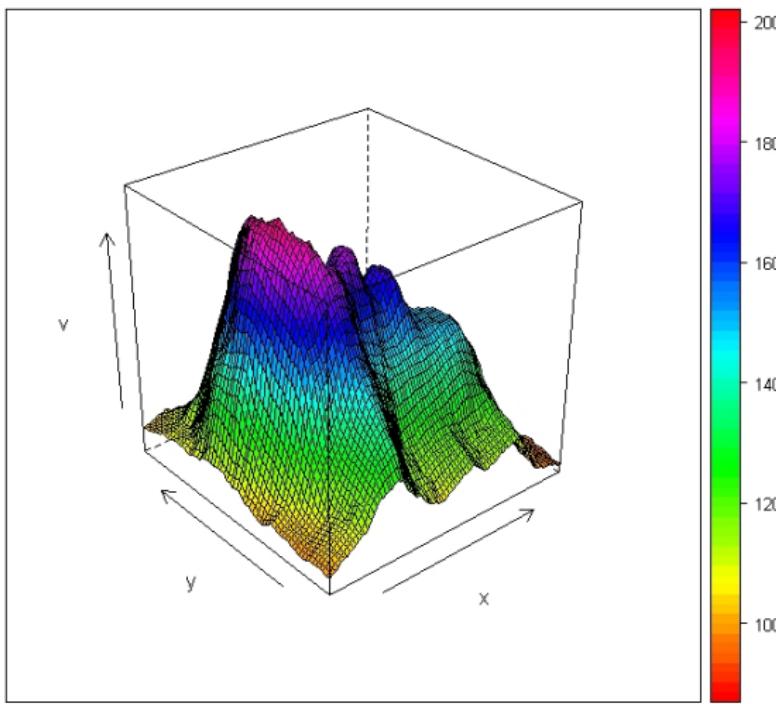
2-D and 3-D plots

```
dim(volcano)
[1] 87 61

# Creates a data frame from all combinations of the
# supplied vectors or factors.
vdat <- expand.grid(x = x, y = y)
vdat$v <- as.vector(volcano)

wireframe(v ~ x*y, vdat, drape=TRUE, col.regions = rainbow(100))
```

Plotting



Exercises I

Plotting

Type the answers to the following exercises into a script file and run the commands from there.

1. Create a vector x of the values from 1 to 20.
2. Create a vector $w \leftarrow 1 + \text{sqrt}(x)/2$.
3. Create a data frame called `dummy`, with columns $x = x$ and $y = x + \text{rnorm}(x)*w$. To ensure we all get the same values, set the seed to 0.
4. Create a histogram and a boxplot of y and plot them side-by-side on the same graphing region. Label the axes accordingly. Save your results as a Jpeg file.
5. Plot y versus x using an appropriate plotting command. Put a title on the graph and labels on the axes.

Exercises II

Plotting

6. Enter the command `fm <- lm(y ~ x, data=dummy)` to fit a linear regression model. Add the estimated regression line to the current plot and make it the colour blue.
7. Extract the values of the residuals using `resids <- resid(fm)`. Check that the residuals are normally distributed by creating a Q-Q plot.
8. The airquality data set in the base library has columns Ozone, Solar.R, Wind, Temp, Month and Day. Plot Ozone against Solar.R for each of THREE temperature ranges and each of THREE wind ranges. (Hint: Use coplot.)
9. Construct a histogram of Wind and overlay the density curve. (Hint: Need `hist`, `lines` and `density`.)

Looping and Functions

Functions

- Have already seen functions in R, e.g.

```
mean(x)  
sd(x)  
plot(x, y, ...)  
lm(y ~ x, ...)
```

- Functions have a name and a list of *arguments* or input objects. For example, the argument to the function `mean()` is the vector `x`.
- Functions also have a list of output objects, i.e. objects that are returned once the function has been run (called).
- A function must be written and loaded into R before it can be used.

Functions are typically written if we need to compute the same thing for several data sets and what we want to calculate is not already implemented in the commercial software yet.

Writing Basic Functions

A simple function can be constructed as follows:

```
function_name <- function(arg1, arg2, ...){  
  commands  
  
  output  
}
```

You decide on the name of the function.

The `function` command shows R that you are writing a function.

Inside the parenthesis you outline the input objects required and decide what to call them.

The commands occur inside the `{ }`.

The name of whatever output you want goes at the end of the function.

Comments lines (usually a description of what the function does is placed at the beginning) are denoted by `#`.

Writing Basic Functions

Example

```
sf1 <- function(x){  
  x^2  
}
```

This function is called sf1.

It has one argument, called x.

Whatever value is inputted for x will be squared and the result outputted to the screen.

This function must be loaded into R and can then be called.

Writing Basic Functions

Example

Load the function into R by highlighting the code and clicking the Paste button in RWinEdt.

Type ls() into the console. Note that the function now appears.

Can call the function using:

```
sf1(x = 3)           sf1(3)  
[1] 9               [1] 9
```

To store the result into a variable x.sq

```
x.sq <- sf1(x = 3)      x.sq <- sf1(3)  
> x.sq  
[1] 9
```

Writing Basic Functions

Example

```
sf2 <- function(a1, a2, a3){  
  x <- sqrt(a1^2 + a2^2 + a3^2)  
  return(x)  
}
```

This function is called `sf2` with 3 arguments.

The values inputted for `a1`, `a2`, `a3` will be squared, summed and the square root of the sum calculated and stored in `x`. (There will be no output to the screen as in the last example.)

The `return` command specifies what the function returns, here the value of `x`.

Will not be able to view the result of the function unless you store it.

```
sf2(a1=2, a2=3, a3=4)           sf2(2, 3, 4) # Can't see result.
```

```
res <- sf2(a1=2, a2=3, a3=4)      res <- sf2(2, 3, 4) # Need to use this.  
res  
[1] 5.385165
```

Argument Matching

How does R know which arguments are which? Uses *argument matching*.

Argument matching is done in a few different ways.

- The arguments are matched by their positions. The first supplied argument is matched to the first formal argument and so on, e.g. when writing `sf2` we specified that `a1` comes first, `a2` second and `a3` third. Using `sf2(2, 3, 4)` assigns 2 to `a1`, 3 to `a2` and 4 to `a3`.
- The arguments are matched by name. A named argument is matched to the formal argument with the same name, e.g. `sf2` arguments have names `a1`, `a2` and `a3`. Can do things like `sf2(a1=2, a3=3, a2=4)`, `sf2(a3=2, a1=3, a2=4)`, etc.
- Name matching happens first, then positional matching is used for any unmatched arguments.

Argument Matching

Can also give some/all arguments *default* values.

```
mypower <- function(x, pow=2){  
  x^pow  
}
```

If a value for the argument `pow` is not specified in the function call, a value of 2 is used.

```
mypower(4)  
[1] 16
```

If a value for `pow` is specified, that value is used.

```
mypower(4, 3)  
[1] 64
```

```
mypower(pow=5, x=2)  
[1] 32
```

More Complex Functions

The following function returns several values in the form of a list:

```
myfunc <- function(x)
{
  # x is expected to be a numeric vector
  # function returns the mean, sd, min, and max of the vector x

  the.mean <- mean(x)
  the.sd <- sd(x)
  the.min <- min(x)
  the.max <- max(x)

  return(list(average=the.mean, stand.dev=the.sd, minimum=the.min,
             maximum=the.max))
}
```

More Complex Functions

```
x <- rnorm(40)
res <- myfunc(x)
res
$average
[1] 0.29713
```

```
$stand.dev
[1] 1.019685
```

```
$minimum
[1] -1.725289
```

```
$maximum
[1] 2.373015
```

To access any particular value use:

```
res$average
[1] 0.29713
```

```
res$stand.dev
[1] 1.019685
```

Exercises

Functions

1. Write a function that when passed a number, returns the number squared, the number cubed, and the square root of the number.
2. Write a function that when passed a numeric vector, prints the value of the mean and standard deviation to the screen (Hint: use the cat() function in R.) and creates a histogram of the data.
3. Management requires a function that calculates the sum of the lengths of 3 vectors. Write the function.

if Statement

Have already encountered *implicit looping* when using the apply family of functions.

Conditional execution: the if statement has the form

```
if (condition){           # Brackets can be omitted if only one command
    expr_1                 # to be carried out.
}
else {
    expr_2
}
```

The condition must evaluate to a logical value, i.e. TRUE or FALSE. If the condition == TRUE, expr_1 is carried out, which can consist of a single command or multiple commands. If the condition == FALSE, expr_2 is carried out.

if Statement

Can also have longer if statements:

```
if (condition1){  
    expr_1  
}  
else if (condition2){  
    expr_2  
}  
...  
else {  
    expr_n  
}
```

If condition1 == TRUE, expr_1 is executed and the checking stops. If condition1 == FALSE, moves on to condition2 and checks if that condition is met. If condition2 == TRUE, expr_2 is executed and checking stops. If condition2 == FALSE, moves on to the next condition and so on until all conditions have been checked.

The final else is executed if none of the previous conditions have returned a value of TRUE.

if Statement

Usually the logical operators `&&`, `||`, `==`, `!=`, `>`, `<`, `>=`, `<=` are used as the conditions in the if statement.

The following function gives a demonstration of the use of
`if... else`.

```
comparisons1 <- function(number)
{
  # if ... else
  if (number != 1)
  {
    cat(number,"is not one\n")
  }
  else
  {
    cat(number,"is one\n")
  }
}

> comparisons1(1)      > comparisons1(20)
1 is one               20 is not one
```

if Statement

The following demonstrates the use of

```
if ... else if ... else  
comparisons2 <- function(number)  
{  
  if (number == 0)  
  {  
    cat(number,"equals 0\n")  
  }  
  else if (number > 0)  
  {  
    cat(number,"is positive\n")  
  }  
  else  
  {  
    cat(number,"is negative\n")  
  }  
}
```

```
> comparisons2(0)  
0 equals 0
```

```
> comparisons2(-15)  
-15 is negative
```

```
> comparisons2(1)  
1 is positive
```

if Statement

This function demonstrates the use of `&&` in the condition. This means that both conditions must be met before a value of TRUE is returned.

```
comparisons3 <- function(number)
{
  if ( (number > 0) && (number < 10) )
  {
    cat(number,"is between 0 and 10\n")
  }
}
```

```
> comparisons3(-1)      > comparisons3(9)          > comparisons3(10)
                           9 is between 0 and 10
```

ifelse Statement

A vectorised version of the if statement is ifelse. This is useful if you want to perform some action on every element of a vector that satisfies some condition.

The syntax is

```
ifelse( condition, true expr, false expr )  
If condition == TRUE, the true expr is carried out. If  
condition == FALSE, the false expr is carried out.
```

```
x <- rnorm(20, mean=15, sd=5)  
x  
[1] 23.608513 14.424667 12.306040 14.291568 18.522846 14.514071 22.004400  
[8] 24.658249 11.697999 16.344976 22.110389 8.455789 19.672274 22.393680  
[15] 11.449034 17.288859 14.839597 14.484774 18.636589 22.670548  
  
ifelse(x >= 17, sqrt(x), NA)  
[1] 4.858859 NA NA NA 4.303818 NA 4.690885  
[8] 4.965707 NA NA 4.702169 NA 4.435344 4.732196  
[15] NA 4.157987 NA NA 4.317012 4.761360
```

for Loops

Repetitive execution: for loops, while loops and repeat loops.

To loop/iterate through a certain number of repetitions a **for** loop is used. The basic syntax is

```
for(variable_name in sequence) {  
  command  
  command  
  command  
}  
}
```

A simple example of a **for** loop is:

```
for(i in 1:5){  
  print(sqrt(i))  
}  
[1] 1  
[1] 1.414214  
[1] 1.732051  
[1] 2  
[1] 2.236068
```

for Loops

Another example is:

```
n <- 20
p <- 5
value <- vector(mode="numeric", length=n)
rand.nums <- matrix(rnorm(n*p), nrow=n)
for(i in 1:length(value)){
  value[i] <- max(rand.nums[i,])
  print(sum(value))
}
```

The first four lines create variables `n` and `p` with values 20 and 5 respectively, a numeric vector called `value` with length 20 and a matrix of $20 \times 5 = 100$ random numbers, called `rand.nums`, with 20 rows.

The `for` loop performs 20 loops and stores the maximum value from each row of `rand.nums` into position `i` of the vector `value`.

The sum of the current numbers in `value` is also printed to the screen.

for Loops

Can also have *nested* for loops. Indenting your code can be useful when trying to “match” brackets.

```
for(variable_name1 in sequence) {  
    command  
    command  
    for(variable_name2 in sequence) {  
        command  
        command  
        command  
    } # ends inner for loop  
} # ends outer for loop
```

It should be noted that variable_name2 should be different from variable_name1, e.g. use i and j. Using the same name will reset the counter each time and result in an infinite loop!!

for Loops

Load the function `simple.nesting` from `Loops.R` and call the function using

```
simple.nesting(num.fam=5, num.child=3).
```

The file `nest.dat` will be created in your current working directory. Open this file and explore the contents.

`for` loops and multiply nested `for` loops are generally avoided when possible in R as they can be quite slow. We will use in simulation examples later in the course.

while Loops

The while loop can be used if the number of iterations required is not known beforehand. For example, if we want to continue looping until a certain condition is met, a while loop is useful.

The following is the syntax for a while loop:

```
while (condition){  
    command  
    command  
}
```

The loop continues while condition == TRUE.

```
niter <- 0  
num <- sample(1:100, 1)  
while(num != 20) {  
    num <- sample(1:100, 1)  
    niter <- niter + 1  
}  
niter
```

next, break, repeat **Statements**

The `next` statement can be used to discontinue one particular iteration of any loop, i.e. this iteration is ended and the loop “skips” to the next iteration. Useful if you want a loop to continue even if an error is found (error checking).

The `break` statement completely terminates a loop. Useful if you want a loop to end if an error is found. See the `Loops.R` script file for code to exhibit the difference between the `next` and `break` statements.

The `repeat` loop uses `next` and `break`. The only way to end this type of loop is to use the `break` statement. For an example, see the `Loops.R` script file.

Exercises

Looping and Functions

1. For each of the following code sequences, predict the result.

Then do the computation:

(a) `answer <- 0
for(j in 3:5) { answer <- j + answer }`

(b) `answer <- 10
j <- 0
while(j < 5) {
 j <- j + 1
 if(j == 3)
 next
 else
 answer <- answer + j*answer
}`

2. Add up all the numbers for 1 to 100 in two different ways:
using a `for` loop and using `sum`.
3. Create a vector `x <- seq(0, 1, 0.05)`. Plot `x` versus `x` and
use `type="l"`. Label the y-axis "y". Add the lines `x` versus
`x^j` where `j` can have values 3 to 5 using either a `for` loop or
a `while` loop.

One Sample Inference

One Sample Tests

If we have a single sample we might want to answer several questions:

- What is the mean value?
- Is the mean value significantly different from current theory? (Hypothesis test)
- What is the level of uncertainty associated with our estimate of the mean value? (Confidence interval)

To ensure that our analysis is correct we need to check for outliers in the data and we also need to check whether the data are normally distributed or not.

Checking Normality

Graphical methods are often used to check that the data being analysed are normally distributed. Can use

- Histogram
- Boxplot
- Normal probability (Q-Q) plot

Read the file dat.txt into a data frame called data:

```
data <- read.table("C:/.../das.txt", header=TRUE)
```

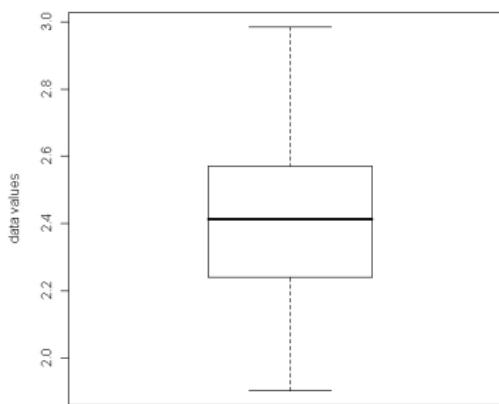
There is only one column of data, y. Can create a summary of y:

summary(data)	summary(data\$y)
Min. :1.904	Min. 1st Qu. Median Mean 3rd Qu. Max.
1st Qu.:2.241	1.904 2.241 2.414 2.419 2.568 2.984
Median :2.414	
Mean :2.419	
3rd Qu.:2.568	
Max. :2.984	

Checking Normality

Create a boxplot, histogram and Q-Q plot of y:

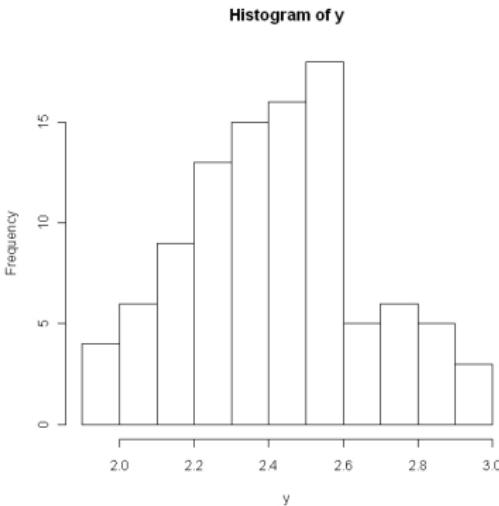
```
boxplot(data$y, ylab="data values")
```



The median line is approximately in the center of the boxplot and the whiskers are approximately the same length. This indicates that the data are normally distributed. There are no obvious outliers.

Checking Normality

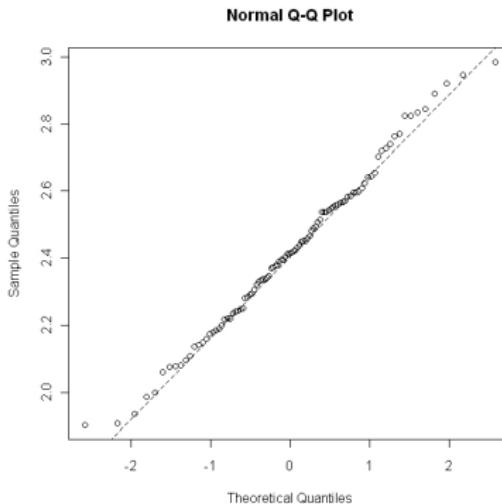
```
histogram(data$y, xlab="y", main="Histogram of y")
```



The histogram is approximately symmetric and bell-shaped indicating a normal distribution.

Checking Normality

```
qqnorm(data$y)  
qqline(data$y, lty=2)
```



All of the points lie in an approximate straight line along the diagonal of the plot indicating a normal distribution. There are no obvious outliers.

Checking Normality

An official test, e.g. the Anderson-Darling test, Kolmogorov-Smirnov test etc. can also be carried out in R. This requires you to download the `nortest` package from CRAN.

Go to <http://cran.r-project.org/> > Packages > `nortest`. Click on the `nortest_1.0.zip` link and click Save. Save the zip file to a suitable location.

The package must be loaded into R by clicking Packages > Install packages from local zip files and locating `nortest`. Once the package has been loaded, can then use `library(nortest)` to access the functions.

Checking Normality

The Anderson-Darling test is carried out using:

```
ad.test(data$y)
```

```
Anderson-Darling normality test
```

```
data: data$y  
A = 0.1634, p-value = 0.9421
```

Remember what this test is doing!!

H_0 : The data are normally distributed.

H_1 : The data are not normally distributed.

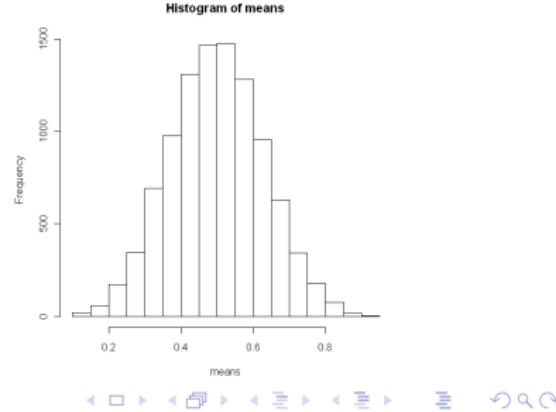
Since the p-value is $0.9421 > 0.05 \Rightarrow$ not enough evidence to reject $H_0 \Rightarrow$ the data are normally distributed.

Central Limit Theorem

Hypothesis testing and confidence interval construction are based on the Central Limit Theorem. CLT - see *Introductory Data Analysis* notes by Dr. Ailish Hannigan.

Can check the CLT using a small simulation example. We will take 10000 samples of size 5 from data with a uniform distribution and record the means. When we plot a histogram of the means, should have a normal distribution.

```
means <- numeric(10000)
for(i in 1:10000){
  means[i] <- mean(runif(5))
}
hist(means)
```



Hypothesis Testing for the Mean

For large samples ($n \geq 30$) and/or if the population standard deviation (σ) is known, the usual test statistic is given by:

$$Z = \frac{\bar{x} - \mu}{SE(\bar{x})}$$

where $SE(\bar{x}) = \sigma/\sqrt{n}$ OR $SE(\bar{x}) = s/\sqrt{n}$ and the critical values are looked up in the normal tables.

For the data stored in das.txt, we want to test the hypothesis that these data are consistent with a long-run average of 3.

$$H_0 : \mu = 3$$

$$H_1 : \mu \neq 3$$

Hypothesis Testing for the Mean

Implemented directly in R using:

```
xbar <- mean(data$y)          # sample mean
s <- sd(data$y)                # sample std. deviation
n <- length(data$y)           # sample size
mu <- 3                        # test value

Z <- (xbar - mu)/(s/sqrt(n))
Z
[1] -24.03721
```

Now need to determine the p-value, i.e. the probability of getting a value less than -24.03721 and compare with α or the critical value to compare with the TS.

Hypothesis Testing for the Mean

Have already seen how to do this using `pnorm` and `qnorm`.

```
2*pnorm(-24.03721)
[1] 1.135936e-127  (approx. 0)
```

Since p-value = 0 \Rightarrow enough evidence to reject H_0 and conclude that $\mu \neq 3$.

ALTERNATIVELY – this is a two-tailed test and $\alpha = 0.05$

```
qnorm(0.975)           qnorm(0.025)   OR  qnorm(0.975, lower.tail=FALSE)
[1] 1.959964          [1] -1.959964
```

Since $TS = -24.03721 < -1.96 \Rightarrow$ enough evidence to reject H_0 and conclude that $\mu \neq 3$.

Hypothesis Testing for the Mean

If this was a one-tailed test:

$$H_0 : \mu \leq 3$$

$$H_1 : \mu > 3$$

$$H_0 : \mu \geq 3$$

$$H_1 : \mu < 3$$

The p-value would be calculated as (note no multiplication by 2)

```
pnorm(-24.03721)  
[1] 5.679679e-128
```

and only one critical value is required. When there is a $>$ sign in H_1 and $\alpha = 0.05$:

```
qnorm(0.95)  
[1] 1.644854
```

When there is a $<$ sign in H_1 and $\alpha = 0.05$:

```
qnorm(0.05)          OR qnorm(0.05, lower.tail=FALSE)  
[1] -1.644854
```

Confidence Intervals for the Mean

A confidence interval for the sample mean of a large sample or when σ is known is given by

$$\bar{x} - Z_{1-\alpha/2}SE(\bar{x}) < \mu < \bar{x} + Z_{1-\alpha/2}SE(\bar{x})$$

Need to find $Z_{1-\alpha/2}$ from the normal tables. Use `qnorm` which gives the value of x for a specified value of k such that $P[X \leq x] = k$.

A 95% CI: $\bar{x} - Z_{0.975}SE(\bar{x}) < \mu < \bar{x} + Z_{0.975}SE(\bar{x})$.

This implies that $k = 0.975$ and $Z_{0.975} = \text{qnorm}(0.975) = 1.96$.
For the das.txt data:

```
lower.tail <- xbar - qnorm(0.975)*s/sqrt(n)
upper.tail <- xbar + qnorm(0.975)*s/sqrt(n)
```

95% CI: $2.372119 < \mu < 2.466793$.

Confidence Intervals for the Mean

A 90% CI: $\bar{x} - Z_{0.95}SE(\bar{x}) < \mu < \bar{x} + Z_{0.95}SE(\bar{x})$.

This implies that $k = 0.95$ and $Z_{0.95} = \text{qnorm}(0.95) = 1.645$.

```
lower.tail <- xbar - qnorm(0.95)*s/sqrt(n)
upper.tail <- xbar + qnorm(0.95)*s/sqrt(n)
```

90% CI: $2.379730 < \mu < 2.459182$.

Hypothesis Testing for the Mean

"R" does not have an in-built function to carry out a Z-test but does have the function `t.test`. A t-test is usually carried out when the sample size is small ($n < 30$) and the sample standard deviation is used. The test statistic is

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

and the critical values are looked up in the t-tables.

However, remember that as the sample size increases, the t distribution becomes more and more like a normal distribution. Carrying out a t-test for a large sample is equivalent to carrying out the Z-test as outlined above when the sample standard deviation is used.

Hypothesis testing

Testing the previous hypothesis using the t.test function

```
t.test(data$y, mu=3, alternative="two.sided", conf.level=0.95)
```

```
One Sample t-test
```

```
data: data$y
```

```
t = -24.0372, df = 99, p-value < 2.2e-16
```

```
alternative hypothesis: true mean is not equal to 3
```

```
95 percent confidence interval:
```

```
2.371533 2.467379
```

```
sample estimates:
```

```
mean of x
```

```
2.419456
```

Hypothesis Testing for the Mean

The line

One Sample t-test

is self-explanatory.

```
data: data$y
```

tells us what data has been used for the test.

```
t = -24.0372, df = 99, p-value < 2.2e-16
```

tells us the value of the test statistic (t), the degrees of freedom ($n - 1$) and the p-value for the test. If the $p\text{-value} < \alpha \Rightarrow$ enough evidence to reject H_0 , which is the case here.

Hypothesis Testing for the Mean

alternative hypothesis: true mean is not equal to 3

tells us (a) that this is a two-tailed test (as we have not equal to in the alternative hypothesis) and (b) the value that we wanted to test whether the mean could be equal to.

95 percent confidence interval:

2.371533 2.467379

is the 95% confidence interval for these data calculated using

$$\bar{x} - t_{(0.975, df=99)} SE(\bar{x}) < \mu < \bar{x} + t_{(0.975, df=99)} SE(\bar{x}).$$

Hypothesis Testing for the Mean

There are a number of things that we may need to change when using `t.test`.

One is the value of μ - in the last example we tested $\mu = 3$.

Another is to change from a two-tailed to a one-tailed test. For example to test:

$$H_0 : \mu \leq 3$$

$$H_1 : \mu > 3$$

the call to `t.test` is

```
t.test(data$y, mu=3, alternative="greater", conf.level=0.95)
```

NOTE: Need to be careful with the CI estimate - is a one-sided CI.

95 percent confidence interval:
2.379354 Inf

Hypothesis Testing for the Mean

To test:

$$H_0 : \mu \geq 3$$

$$H_1 : \mu < 3$$

the call to t.test is

```
t.test(data$y, mu=3, alternative="less", conf.level=0.95)
```

Finally, to change from a 95% CI to a 90% CI, change "conf.level=0.95" to "conf.level=0.90" (or to a 99% CI - "conf.level=0.99").

If you save the output from t.test

```
t.res <- t.test(data$y, mu=3, alternative="less", conf.level=0.95)
```

the individual components, i.e. confidence interval, p-value, etc. can all be accessed using the \$ notation.

Hypothesis Testing for the Mean

```
names(t.res)
[1] "statistic"    "parameter"    "p.value"      "conf.int"      "estimate"
[6] "null.value"   "alternative"  "method"       "data.name"

t.res$statistic           t.res$p.value          t.res$conf.int
t                         [1] 2.051910e-43      [1] -Inf 2.459557
-24.03721                  attr(,"conf.level")
                           [1] 0.95
```

Hypothesis Tests for a Proportion

We can also perform a hypothesis test for a population proportion, p . The R function to carry out such a test is `prop.test`.

A manufacturer claims that the proportion of defective items produced is approximately 4%. A random sample of size 50 is taken, 3 of which are defective. Is the manufacturer's claim justified?

```
prop.test(x=3, n=50, p = 0.04, alternative = "two.sided", conf.level = 0.95)
```

```
1-sample proportions test with continuity correction
```

```
data: 3 out of 50, null probability 0.04
X-squared = 0.1302, df = 1, p-value = 0.7182
```

```
alternative hypothesis: true p is not equal to 0.04
```

```
95 percent confidence interval:
0.01562459 0.17541874
```

```
sample estimates:
```

```
      p
0.06
```

Hypothesis Tests for a Proportion

The inputs to `prop.test` are x – the number of defective items in the sample, n – the sample size, p – the probability being tested in the hypothesis (if this is `NULL`, $p=0.5$ is used, `alternative` – specifies the alternative hypothesis and `conf.level`.

The outputs are essentially the same as those for `t.test`:

- test statistic value,
- p-value,
- 95% CI,
- sample estimate, etc.

These can be accessed individually using:

```
res.p <- prop.test(x=3, n=50, p=0.04, alternative="two.sided", conf.level=0.95)
```

```
res.p$statistic      res.p$p-value      res.p$conf.int
```

Exercises I

One Sample Inference

1. The following are measurements (in mm) of a critical dimension on a sample of engine crankshafts:

224.120	224.001	224.017	223.982	223.989	223.961
223.960	224.089	223.987	223.976	223.902	223.980
224.098	224.057	223.913	223.999		

- (a) Calculate the mean and standard deviation for these data.
- (b) The process mean is supposed to be $\mu = 224\text{mm}$. Is this the case? Give reasons for your answer.
- (c) Construct a 99% confidence interval for these data and interpret.
- (d) Check that the normality assumption is valid using 2 suitable plots.

Exercises II

One Sample Inference

2. A poll on social issues interviewed 1025 people randomly selected from the United States. 47% of people said that they do not get enough time to themselves. A report claims that over 40% of the population are not satisfied with personal time. Is this the case?
3. Construct a 98% confidence interval for the average weight of male runners. A random sample of 35 runners was taken with a mean weight of $\bar{x} = 60\text{kg}$. Suppose that the standard deviation of the population is known to be $\sigma = 5\text{kg}$.

Two Sample Inference

Two Sample Tests

All of the previous hypothesis tests and confidence intervals can be extended to the two-sample case.

The same assumptions apply, i.e. data are normally distributed in each population and we may want to test if the mean in one population is the same as the mean in the other population, etc.

Normality can be checked using histograms, boxplots and Q-Q plots as before. Or by carrying out the Anderson-Darling test on each group of data.

Hypothesis Tests for Two Means

Independent Groups

If the population standard deviations σ_1 and σ_2 are known, the TS is of the form:

$$Z = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}.$$

The critical value and p-value are looked up in the normal tables.

Example 6.1

A survey of study habits wishes to determine whether the mean study hours completed by women at a particular college is higher than for men at the same college. A sample of $n_1 = 10$ women and $n_2 = 12$ men were taken, with mean hours of study $\bar{x}_1 = 120$ and $\bar{x}_2 = 105$ respectively. The standard deviations were known to be $\sigma_1 = 28$ and $\sigma_2 = 35$.

Hypothesis Tests for Two Means

Independent Groups

The hypothesis being tested is:

$$\begin{aligned} H_0 : \mu_1 &\leq \mu_2 & (\mu_1 - \mu_2 \leq 0) \\ H_1 : \mu_1 &> \mu_2 & (\mu_1 - \mu_2 > 0) \end{aligned}$$

In R, the TS is calculated using:

```
xbar1 <- 120
xbar2 <- 105
sd1 <- 28
sd2 <- 35
n1 <- 10
n2 <- 12

TS <- ( (xbar1 - xbar2) - (0) )/sqrt( (sd1^2/n1) + (sd2^2/n2) )
TS
[1] 1.116536
```

Now need to calculate the critical value or the p-value.

Hypothesis Tests for Two Means

Independent Groups

The critical value can be looked up using `qnorm`. Since this is a one-tailed test and there is a $>$ sign in H_1 :

```
qnorm(0.95)  
[1] 1.644854
```

Since $TS = 1.116536 < 1.645 \Rightarrow$ not enough evidence to reject H_0 and conclude that the population mean hours study for women is not higher than the population mean hours study for men.

The p-value is determined using `pnorm`. Careful! Remember `pnorm` gives the probability of getting a value LESS than the value specified. We want the probability of getting a value greater than the TS.

```
1-pnorm(1.116536)           OR pnorm(1.116536, lower.tail=FALSE)  
[1] 0.1320964
```

Again, since $p\text{-value} > 0.05 \Rightarrow$ not enough evidence to reject H_0 and conclude that the population mean hours study for women is not higher than the population mean hours study for men.

Confidence Intervals for Two Means

Independent Groups

A confidence interval can be constructed using

$$(\bar{x}_1 - \bar{x}_2) \pm Z_{1-\alpha/2} \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}.$$

A 95% CI:

```
lower.tail <- (xbar1 - xbar2) - qnorm(0.975)*sqrt( (sd1^2/n1) + (sd2^2/n2) )
upper.tail <- (xbar1 - xbar2) + qnorm(0.975)*sqrt( (sd1^2/n1) + (sd2^2/n2) )
```

A 99% CI:

```
lower.tail <- (xbar1 - xbar2) - qnorm(0.995)*sqrt( (sd1^2/n1) + (sd2^2/n2) )
upper.tail <- (xbar1 - xbar2) + qnorm(0.995)*sqrt( (sd1^2/n1) + (sd2^2/n2) )
```

Sample Standard Deviations Used

Independent Groups

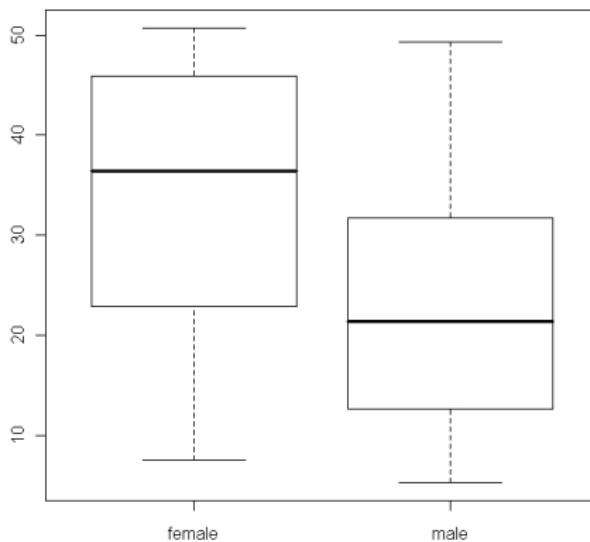
When the sample standard deviations are used we can use `t.test` to carry out the hypothesis tests and construct the confidence intervals.

Example 6.2

Read the data from the file `twosample.txt` into R. These data show the ages of a random sample of 49 people from a particular town. Also recorded is the gender of each person. The town council would like to determine if there is a difference between the average age of females and average age of males in the town. Carry out an appropriate hypothesis test using $\alpha = 0.01$.

Looking at side-by-side boxplots:

```
boxplot(age$x ~ age$gender)
```



Can see that the median age for females is higher than that for males and the boxes overlap only slightly indicating there may be a significant difference between the population mean age for females and the population mean age for males.

Testing:

$$\begin{array}{ll} H_0 : \mu_{fem} = \mu_{male} & (\mu_{fem} - \mu_{male} = 0) \\ H_1 : \mu_{fem} \neq \mu_{male} & (\mu_{fem} - \mu_{male} \neq 0) \end{array}$$

There are two ways to carry out this test:

```
age <- read.table("C:/.../twosample.txt", header=TRUE)

fem.age <- age$x[age$gender=="female"]
male.age <- age$x[age$gender=="male"]

t.test(fem.age, male.age, mu=0, alternative="two.sided", conf.level=0.99)

Welch Two Sample t-test

data: fem.age and male.age
t = 2.6532, df = 45.511, p-value = 0.01094
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
-0.1340292 20.3957737
sample estimates:
mean of x mean of y
32.85932 22.72845
```

Can see that the p-value is 0.01094 which is just slightly larger than $\alpha = 0.01 \Rightarrow$ there is not enough evidence to reject H_0 and conclude that there is no difference between the population mean age for females and males.

The second approach supplies a formula to t.test:

```
t.test(age$x ~ age$gender, mu=0, alternative="two.sided", conf.level=0.99)
```

Welch Two Sample t-test

```
data: age$x by age$gender
t = 2.6532, df = 45.511, p-value = 0.01094
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
-0.1340292 20.3957737
sample estimates:
mean in group female   mean in group male
            32.85932                22.72845
```

The \sim specifies that age\$x is described by age\$gender.

Confidence Intervals

When both sample sizes are large ($n_1, n_2 \geq 30$), a confidence interval can be constructed using

$$(\bar{x}_1 - \bar{x}_2) \pm Z_{1-\alpha/2} \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}.$$

However, if one or both sample sizes are small we use the t-distribution. There are two approaches that can be carried out. The first assumes that the variances in the two populations are different.

In this case the Welch Two Sample t-test is used (as seen in the previous examples). This adjusts the degrees of freedom of the t-distribution used to look up the critical values, calculate the p-value and construct confidence intervals. The CI can be written as:

$$(\bar{x}_1 - \bar{x}_2) \pm t_{df, 1-\alpha/2} \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}.$$

Assuming Equal Variances

The usual approach is to assume that the variances in each population are identical (this must be first be verified using an F test) and to use a *pooled* estimate of the standard deviation based on the standard deviations in each group. The critical values and p-value then come from a t-distribution with $n_1 + n_2 - 2$ degrees of freedom. The TS then becomes

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_p^2}{n_1} + \frac{s_p^2}{n_2}}},$$

where $s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$ is the pooled estimate of the standard deviation.

The CI can be written as:

$$(\bar{x}_1 - \bar{x}_2) \pm t_{(df=n_1+n_2-2, 1-\alpha/2)} \sqrt{\frac{s_p^2}{n_1} + \frac{s_p^2}{n_2}}.$$

Assuming Equal Variances

For the previous example, if we assume that the variances are equal:

```
t.test(age$x ~ age$gender, mu=0, conf.level=0.99, var.equal=T)
```

Two Sample t-test

```
data: age$x by age$gender
t = 2.6609, df = 47, p-value = 0.01063
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
-0.09012371 20.35186813
sample estimates:
mean in group female   mean in group male
            32.85932           22.72845
```

Can see that there is little difference between the results. Slight change in value for the TS, the degrees of freedom is now a whole number and the p-value has changed slightly.

In general both approaches yield very similar results, unless both the group sizes and standard deviations are very different.

Comparing Variances

Can check that our assumption of equal variances is correct by carrying out an F test.

$$H_0 : \sigma_1^2 = \sigma_2^2 \quad (\sigma_1^2 / \sigma_2^2 = 1)$$

$$H_1 : \sigma_1^2 \neq \sigma_2^2 \quad (\sigma_1^2 / \sigma_2^2 \neq 1)$$

There is a simple command in R to do this:

`var.test(fem.age, male.age)` OR `var.test(age$x ~ age$gender)`

F test to compare two variances

```
data: age$x by age$gender
F = 1.3244, num df = 23, denom df = 24, p-value = 0.4989
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
0.5803683 3.0447669
sample estimates:
ratio of variances
1.324441
```

The p-value is $> 0.05 \Rightarrow$ not enough evidence to reject H_0 so conclude that the variances are equal. OK to use `var.equal=T`.



Hypothesis Tests Two Means

Paired Data

Sometimes two-sample data comes from paired observations. This occurs when two measurements are made on the same experimental unit. For example, a new drug designed to reduce blood pressure is being tested. Patient 1 is given the new drug, and a BP measurement is taken. A week later Patient 1 is given the current drug on the market, and a second BP measurement is taken. Then want to compare if there is a significant difference between the new drug and the current drug used.

Example 6.3

The file `matchedpairs.txt` contains the weights of 10 people before and after an exercise regime was undertaken. Want to determine if this regime resulted in weight loss.

Hypothesis Tests Two Means

Paired Data

Read the data in this file into R and store in a dataframe called `wgts`.

```
wgts <- read.table("C:/.../matchedpairs.txt", header=TRUE)
```

```
wgts
  Before After
1     168   156
2     215   200
3     225   223
4     192   190
5     207   203
6     224   201
7     246   211
8     198   176
9     201   194
10    195   203
```

Hypothesis Tests Two Means

Paired Data

To carry out this test, must first calculate the *differences* between the before and after measurements. Calculate the vector of differences Before - After and store into `diffs`.

```
diffs <- wgts$Before - wgts$After
```

```
diffs  
[1] 12 15  2  2  4 23 35 22  7 -8
```

Important to note that if the after measurement is less than the before measurement (i.e. the person LOST weight) the difference will be positive. The more positive the difference, the more weight the person lost.

Hypothesis Tests Two Means

Paired Data

If exercising “works”, would expect the average difference to be greater than 0 (since differences are calculated as Before – After).

$$H_0 : \mu_d \leq 0$$

$$H_1 : \mu_d > 0$$

The TS is:

$$t = \frac{\bar{x}_d - \mu_d}{s_d / \sqrt{n}}.$$

(NOTE: if the number of differences is $\geq 30 \Rightarrow$ use the normal distribution.)

```
n <- length(diffs)  
  
TS <- ( mean(diffs) - 0 )/( sd(diffs)/sqrt(n) )  
TS  
[1] 2.845656
```

Hypothesis Tests Two Means

Paired Data

Look up the CV or the p-value.

```
qt(0.95, df=9)      # CV  
[1] 1.833113
```

```
1 - pt(TS, df=9)    # p-value  
[1] 0.009612471
```

The CI is calculated as:

$$\bar{x}_d \pm t_{(1-\alpha/2, df)} \frac{s_d}{\sqrt{n}}.$$

```
lower.tail <- mean(diffs) - qt(0.975, df=9)*sd(diffs)/sqrt(n)  
lower.tail  
[1] 2.337558
```

```
upper.tail <- mean(diffs) + qt(0.975, df=9)*sd(diffs)/sqrt(n)  
upper.tail  
[1] 20.46244
```

Hypothesis Tests Two Means

Paired Data

Instead of doing this by hand, could use `t.test`. Can do a one-sample t-test on `diffs`

```
t.test(x=diffs, mu=0, alternative="greater", conf.level=0.95)
```

One Sample t-test

```
data:  diffs
t = 2.8457, df = 9, p-value = 0.009612
alternative hypothesis: true mean is greater than 0
95 percent confidence interval:
 4.056354      Inf
sample estimates:
mean of x
 11.4
```

Can see that the results are the same as we calculated earlier.

Hypothesis Tests Two Means

Paired Data

R also allows us to carry out a paired t-test directly (i.e. without calculating the differences):

```
t.test(x=wgts$Before, y=wgts$After, alternative="greater", paired=TRUE)
```

Paired t-test

```
data: wgts$Before and wgts$After
t = 2.8457, df = 9, p-value = 0.009612
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 4.056354      Inf
sample estimates:
mean of the differences
                  11.4
```

R calculates the differences internally when `paired=TRUE`. The differences are calculated as $x - y$.

Hypothesis Tests Two Proportions

Can also extend the hypothesis test for a proportion to the two-sample case. For example, in manufacturing plant, might want to compare that the proportion of defective items from Line 1 is the same as the proportion defective from Line 2.

Example 6.4

A company wants to investigate the proportion of males and females promoted in the last year. 4 out of 40 female candidates were promoted, while 196 out of 3270 male candidates were promoted. Is there evidence of sexism in the company?

Testing:

$$H_0 : p_1 = p_2 \quad (p_1 - p_2 = 0)$$

$$H_1 : p_1 \neq p_2 \quad (p_1 - p_2 \neq 0)$$

Hypothesis Tests Two Proportions

```
prop.test(c(4,196), c(40,3270))
```

```
2-sample test for equality of proportions with continuity correction
```

```
data: c(4, 196) out of c(40, 3270)
```

```
X-squared = 0.5229, df = 1, p-value = 0.4696
```

```
alternative hypothesis: two.sided
```

```
95 percent confidence interval:
```

```
-0.06591631 0.14603864
```

```
sample estimates:
```

```
prop 1     prop 2
```

```
0.10000000 0.05993884
```

There is no evidence of sexism in the company since the p-value is $0.4696 > 0.05 \Rightarrow$ not enough evidence to reject H_0 . Conclude that $p_1 = p_2$, the proportion of females promoted is equal to the proportion of males promoted.

Chi-squared Test

A χ^2 test is carried out on tabular data containing counts, e.g. the number of animals that died, the number of days of rain, the number of stocks that grew in value, etc.

Usually have two qualitative variables, each with a number of levels, and want to determine if there is a relationship between the two variables, e.g. hair colour and eye colour, social status and crime rates, house price and house size, gender and left/right handedness.

The data are presented in a contingency table:

	right-handed	left-handed	TOTAL
male	43	9	52
female	44	4	48
TOTAL	87	13	100

Chi-squared Test

The hypothesis to be tested is

H_0 : There is no relationship between gender and left/right-handedness

H_1 : There is a relationship between gender and left/right-handedness

The values that we collect from our sample are called the observed (O) frequencies (counts). Now need to calculate the expected (E) frequencies, i.e. the values we would expect to see in the table, if H_0 was true.

$$E = \frac{\text{Row Total} \times \text{Column Total}}{\text{Grand Total}},$$

		right-handed	left-handed	TOTAL
male	O	43	9	52
	E	$\frac{52 \times 87}{100}$	$\frac{52 \times 13}{100}$	
female	O	44	4	48
	E	$\frac{48 \times 87}{100}$	$\frac{48 \times 13}{100}$	
TOTAL		87	13	100

Chi-squared Test

The TS is then:

$$\chi^2 = \sum \frac{(O - E)^2}{E}.$$

This has a χ^2 distribution with $(r - 1) \times (c - 1)$ degrees of freedom.

This can be carried out in R by hand:

```
obs.vals <- matrix(c(43,9,44,4), nrow=2, byrow=T)
row.tots <- apply(obs.vals, 1, sum)
col.tots <- apply(obs.vals, 2, sum)
exp.vals <- row.tots%o%col.tots/sum(obs.vals)

TS <- sum((obs.vals-exp.vals)^2/exp.vals)
TS
[1] 1.777415
```

Chi-squared Test

Look up the critical value and p-value in a χ^2 distribution with df = $(2 - 1) \times (2 - 1) = 1$.

```
qchisq(0.95, df=1)      # Critical value.  
[1] 3.841459
```

```
1-pchisq(TS, df=1)      # p-value.  
[1] 0.1824671
```

Since $TS < CV \Rightarrow$ not enough evidence to reject H_0 , conclude that there is no relationship between gender and right/left-handedness. (Same conclusion for p-value.)

Chi-squared Test

R has an in-built function chisq.test to carry out this test.

```
chisq.test(obs.vals, correct=F)
```

Pearson's Chi-squared test

```
data: obs.vals  
X-squared = 1.7774, df = 1, p-value = 0.1825
```

Storing the results from this test:

```
chi.res <- chisq.test(obs.vals, correct=F)
```

```
names(chi.res)  
[1] "statistic" "parameter" "p.value"     "method"      "data.name" "observed"  
[7] "expected"   "residuals"
```

allows us to access the above calculated values using the \$ notation as with t.test.

Sample Size and Power Calculations

Each time a hypothesis test is carried out, there is a chance that the wrong conclusion will be reached. There are two types of error:

- Type I error (α) – reject H_0 when it is true.
- Type II error (β) – accept H_0 when it is false.

	H_0 true	H_0 false
Reject H_0	Type I error α	Correct decision
Accept H_0	Correct decision	Type II error β

Sample Size and Power Calculations

Ideally want both α and β to be as small as possible. However, they are not independent and making β smaller, increases α and vice versa.

The *power* of the test, defined as $1 - \beta$, is the probability of accepting H_0 when it is true, i.e. make the correct decision. Want this to be as close as possible to 1.

The risk of a Type II error depends on the size and nature of the deviation you are trying to detect, e.g. if there is very little difference, there is not much chance of detecting it, i.e. the power will be small. The power calculations can be used in two ways:

- Given the sample size used, what is the chance of detecting a difference of a particular size?
- What size sample is required to detect a difference of a particular size with specified power?

The R function `power.t.test` or `power.prop.test`.

Sample Size and Power Calculations

Single sample: to detect a difference $\delta = \pm 2$ from a distribution with a mean of 20 and variance 10 with probability (power) 0.8, the sample size required is:

```
power.t.test(type="one.sample", power=0.8, sd=sqrt(10), delta=2)
```

One-sample t test power calculation

```
    n = 21.62146
    delta = 2
    sd = 3.162278
    sig.level = 0.05
    power = 0.8
    alternative = two.sided
```

The sample size required is a decimal – round up!! Hence, need a sample of size 22.

Sample Size and Power Calculations

Conversely, the probability of detecting a difference of $\delta = \pm 2$ from a distribution with mean of 20 and variance 10 using a sample of size 20 is:

```
power.t.test(type="one.sample", n=20, sd=sqrt(10), delta=2)
```

One-sample t test power calculation

```
n = 20
delta = 2
sd = 3.162278
sig.level = 0.05
power = 0.765219
alternative = two.sided
```

Can see that the power of the test is 0.765219.

Sample Size and Power Calculations

Two Samples: To determine the sample size required for two groups such that at the 1% level, we can find a difference of 0.5cm in a distribution with a standard deviation of 2cm with power 0.9:

```
power.t.test(delta=0.5, power=0.9, sd=2, sig.level=0.01)
```

Two-sample t test power calculation

```
    n = 477.8021
    delta = 0.5
    sd = 2
    sig.level = 0.01
    power = 0.9
    alternative = two.sided
```

NOTE: n is number in *each* group

NOTE: the "type=one.sample" is removed and the default "type=two.sample" is used. Samples of size 488 should be taken from each group.

Sample Size and Power Calculations

Power calculations and sample sizes can also be determined for one-tailed tests. An example of this using the previous data is:

```
power.t.test(delta=0.5, power=0.9, sd=2, sig.level=0.01, alt="one.sided")
```

Two-sample t test power calculation

```
    n = 417.898
    delta = 0.5
    sd = 2
    sig.level = 0.01
    power = 0.9
    alternative = one.sided
```

NOTE: n is number in *each* group

Note the slight reduction in sample size required.

Sample Size and Power Calculations

Paired Data: To determine the power of the test to detect a difference of size 10 from a population of paired data such that the standard deviation of the differences is approximately 14 and using samples of size 21:

```
power.t.test(type="paired", delta=10, sd=14, n=21)
```

Paired t test power calculation

```
n = 21
delta = 10
sd = 14
sig.level = 0.05
power = 0.8754649
alternative = two.sided
```

NOTE: n is number of *pairs*, sd is std.dev. of *differences* within pairs

The power of this test is 0.8754649.

Sample Size and Power Calculations

Proportions: Currently can only consider calculations for two samples.

The R command is `power.prop.test` and is essentially the same as `power.t.test`, except `delta` and `sd` are replaced by `p1` and `p2`. These are the hypothesised probabilities in the two groups.

What sample sizes are required to obtain a power of 0.85 with $p_1 = 0.15$ and $p_2 = 0.30$?

```
power.prop.test(power=0.85, p1=0.15, p2=0.30)
```

Two-sample comparison of proportions power calculation

```
    n = 137.6040
    p1 = 0.15
    p2 = 0.3
sig.level = 0.05
    power = 0.85
alternative = two.sided
```

NOTE: `n` is number in *each* group

Exercises I

Two Sample Inference

1. The following data were collected by researchers measuring the number of beetle larva per stem in plots of oats after applying one of two treatments; no pesticide (control) or Malathion. The following data were collected.

Group	Treatment	n	\bar{x}	s
1	Control	13	3.47	1.21
2	Malathion	14	1.36	0.52

Is there significant evidence that the mean number of larvae per stem is reduced by malathion? Use $\alpha = 0.01$. You may assume that the variances are equal.

Exercises II

Two Sample Inference

2. To test the protein content of a new variety of corn with increased amounts of lysine, 20 male chicks were fed a ration containing the new corn and another 20 chicks were fed a ration of normal corn. The file `twosampleex2.txt` contain the weight gains (in grams) of the chicks after 21 days. Read these data into R and store in a data frame called `chicks`.
 - (a) Present the data graphically and describe the distribution of the data.
 - (b) Test whether the variances are the same in each group.
 - (c) Based on the results of part (b), test the claim that the chicks fed the new corn gain more weight than chicks fed the normal corn.
 - (d) Construct a 98% CI for the mean weight gain in chicks fed the new corn. Do this in two ways - one using the CI formula, and one using `t.test`.

Exercises III

Two Sample Inference

3. The data in `twosampleex3.txt` contains the pretest and posttest scores on a listening exam in French for 40 French teachers who attended and intensive summer course in French. Read this data into R and store in a data frame called `french.res`.
 - (a) What type of design is this (i.e. independent groups or matched pairs)?
 - (b) We want to show that attending the institute improves listening skills. State the appropriate H_0 and H_1 and carry out the appropriate test in R.
 - (c) Calculate and store the differences between the pretest and posttest scores ($\text{Post} - \text{Pre}$) and check that these are normally distributed.

Exercises IV

Two Sample Inference

4. When comparing variances, the F test statistic is calculated using $F = \frac{s_1^2}{s_2^2}$ and the critical value is looked up in the F tables with degrees of freedom $n_1 - 1$ and $n_2 - 1$. What is the upper 2.5% critical value when F is calculated using samples of size $n_1 = 10$ and $n_2 = 8$? (Hint: use `qf`)
5. A study of chromosome abnormalities and criminality examined data on 4124 Danish males. Each man was classified as having a criminal record or not and as having the normal chromosome pair or one of the abnormalities. 381 out of the 4096 men with normal chromosomes had a criminal record, while 8 out of the 28 men with chromosome abnormalities had a criminal record. Some experts believe that chromosome abnormalities are linked with increased criminality. Do these data lend support to this belief?

Exercises V

Two Sample Inference

6. Three suppliers provide the following data on defective parts.

Supplier	Part Quality		
	Good	Minor Defect	Major Defect
A	90	3	7
B	170	18	7
C	135	6	9

- (a) Test for independence between supplier and part quality. Use $\alpha = 0.05$.
- (b) Show how the test statistic value and p-value from the test were calculated using appropriate R commands.

Exercises VI

Two Sample Inference

7. A telephone company bases charges to a client on the assumption that telephone surveys can be completed in 15 minutes or less. With a sample of 35 surveys, a population standard deviation of 4 minutes and a level of significance of 0.01, the sample mean will be used to test the hypothesis $H_0 : \mu \leq 15$. What is the power of this test to detect a change of $\delta = \pm 2$? What sample size is required to make the power = 0.9 if a two-tailed test is used?

Regression and Correlation

Simple Linear Regression

Simple linear regression is used to describe the relationship between two variables. For example, you may want to describe the relationship between age and blood pressure or the relationship between scores in a midterm exam and scores in the final exam, etc.

The simple linear regression model is of the form:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

where the ε_i are i.i.d. random variables such that $\varepsilon_i \sim N(0, \sigma^2)$. The intercept β_0 describes the point at which the line intersects the y axis while the slope β_1 , describes the change in y (response/dependent variable) for every unit increase in x (explanatory/independent variable).

Need to use our sample data to estimate β_0 , β_1 and σ^2 - use the method of least squares.

Scatterplot

One of the first steps in a regression analysis is to determine if any kind of relationship exists between the two variables.

A scatterplot is created and can initially be used to get an idea about the nature of the relationship between the variables, e.g. if the relationship is linear, curvilinear, or no relationship exists.

Read the data from the file SLRex1.txt into a data frame called SLR1.

```
SLR1 <- read.table("C:/.../SLRex1.txt", header=T)  
SLR1
```

	x	y
1	5.366516	26.76595
2	6.435778	46.89376
3	7.831232	34.11415
4	7.587142	45.49667
5	5.380939	33.22162
6	8.254098	39.98920

...

Scatterplot

This file contains readings for a response variable y and an explanatory variable x .

To create a scatterplot in R, simply use the `plot` command. Remember y is the response variable and x is the explanatory variable:

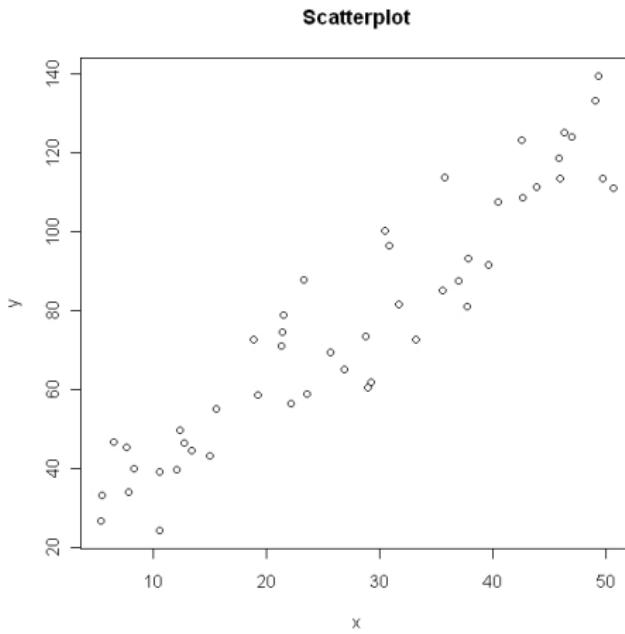
```
plot(x=SLR1$x, y=SLR1$y, main="Scatterplot")
```

OR

```
attach(SLR1)
plot(x=x, y=y, main="Scatterplot")
```

(The `attach` command makes the variables in the data frame `SLR1` accessible by name rather than using the `$` notation). Need to be careful as if there are other variables in use with the same names as those in the data frame, R will get confused. When finished working with `SLR1`, need to use `detach(SLR1)`.

Scatterplot



There is clearly a positive linear relationship between x and y. As x increases y also increases. No obvious outliers. SLR is appropriate here.

Correlation

Can see from the scatterplot that there is a linear relationship between x and y . To measure the strength of this relationship, calculate the correlation coefficient

$$r = \frac{\text{Cov}(x, y)}{\sqrt{\text{Var}(x) \times \text{Var}(y)}}.$$

Calculate the variances of x and y and covariance between x and y

```
var(x)  
[1] 199.9837
```

```
var(y)  
[1] 977.0153
```

```
var(x,y) # Gives the covariance between x and y.  
[1] 414.9603
```

```
cor.res <- var(x,y)/sqrt(var(x)*var(y))  
cor.res  
[1] 0.9387684
```

Correlation

"R" also has an in-built function to calculate the correlation coefficient:

```
cor(x,y)  
[1] 0.9387684
```

Remember $-1 \leq r \leq 1$ and the closer r is to ± 1 , the stronger the linear relationship. $r = -1 \Rightarrow$ perfect negative linear relationship. $r = +1 \Rightarrow$ perfect positive linear relationship. $r = 0 \Rightarrow$ either no relationship or no LINEAR relationship.

Here, there is a strong positive linear relationship since $r = +0.9387684$. (This reinforces the results from the scatterplot, where we saw a positive linear relationship.)

Hypothesis Test Correlation Coefficient

Can also test whether the correlation between the two variables is statistically significant. NOTE: ρ is the population correlation coefficient.

$H_0 : \rho = 0$ (There is no linear relationship)

$H_1 : \rho \neq 0$ (There is a significant linear relationship)

This test is carried out in R using:

```
cor.test(x,y)
Pearson's product-moment correlation
```

```
data: x and y
t = 18.6791, df = 47, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.8934139 0.9651786
sample estimates:
cor
0.9387684
```

Hypothesis Test Correlation Coefficient

The correlation coefficient is highly significant. The p-value is virtually $0 < 0.05 \Rightarrow$ enough evidence to reject H_0 and conclude that there is a significant linear relationship between y and x .

Model Fitting

To fit a SLR model

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

in R we use:

```
model <- lm(y ~ x)
names(model)
[1] "coefficients"   "residuals"      "effects"       "rank"
[5] "fitted.values"  "assign"        "qr"            "df.residual"
[9] "xlevels"        "call"          "terms"         "model"
```

which is read: `model` is a linear model where `y` is modelled as a function of `x`. Can see there are various output values returned.

Model Fitting

To access the values of the estimated coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$:

```
model$coefficients
(Intercept)           x
19.94379      2.07497
```

Can also access the fitted values \hat{y} , i.e. the y values you would expect to see for a particular value of x according to the best-fitting model. The model is

$$\hat{y} = 19.94379 + 2.07497x.$$

```
SLR1$x[1]
[1] 5.366516
```

```
19.94379 + 2.07497*SLR1$x[1]
[1] 31.07915
```

```
model$fitted
 1          2          3          4          5          6          7
31.07915 33.29783 36.19336 35.68688 31.10907 37.07079 41.84819
...

```

Model Fitting

To determine the residuals $y - \hat{y}$ from the fitted model:

```
SLR1$y[1]  
[1] 26.76595
```

```
SLR1$y[1] - model$fitted[1]  
-4.313199
```

```
model$residuals  
1 2 3 4 5 6  
-4.3131987 13.5959306 -2.0792120 9.8097867 2.1125467 2.9184105  
...  
...
```

R also has some inbuilt functions to access the coefficients, fitted values and residuals of a particular model. Enter the following commands and verify that they give the same results as using the \$ notation:

```
coef(model)
```

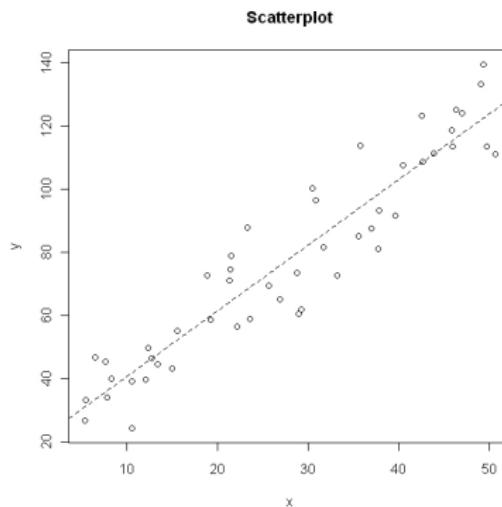
```
fitted(model)
```

```
resid(model)
```

Model Fitting

Can add the fitted regression line to the scatterplot:

```
plot(x,y,main="Scatterplot")
abline(coef(model), lty=2)          OR abline(model$coef, lty=2)
```



Model Fitting

Can also do lots of different things with the object called `model`.
The first thing we might do is summarise it:

```
summary(model)
```

Call:

```
lm(formula = y ~ x)
```

Residuals:

Min	1Q	Median	3Q	Max
-19.433	-8.580	-1.188	9.064	19.787

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	19.9438	3.4466	5.786	5.66e-07 ***
x	2.0750	0.1111	18.679	< 2e-16 ***

Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 1

Residual standard error: 10.88 on 47 degrees of freedom

Multiple R-squared: 0.8813, Adjusted R-squared: 0.8788

F-statistic: 348.9 on 1 and 47 DF, p-value: < 2.2e-16

Model Fitting

The Call: displays the model you fitted.

Next comes a summary of the residual values. Will never be able to fit a straight line through all of the data points. There will be some error (residual) = value observed in the sample - value predicted by the line = $y - \hat{y}$ associated with using the line to describe your data. This summary gives the min, max, 1st, 2nd (median) and 3rd quartile values of these residuals.

The Coefficients: section gives the values of the estimated coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$. The fitted regression line is

$$\hat{y} = 19.9438 + 2.075x.$$

Also provided are the standard error for the estimated coefficients, t-values and p-values used to test the hypotheses

$$H_0 : \beta_0 = 0$$

$$H_1 : \beta_0 \neq 0$$

$$H_0 : \beta_1 = 0$$

$$H_1 : \beta_1 \neq 0$$

Model Fitting

The TS are

$$t = \frac{\text{coef}}{\text{SE}(\text{coef})} \sim t_{(df=n-2)}.$$

The p-values for both of these tests are 0 and so there is enough evidence to reject H_0 and conclude that both β_0 and β_1 are not 0, i.e. there is a significant linear relationship between x and y.

Also given are the R^2 and R^2 adjusted values. Here $R^2 = \text{SSR}/\text{SST} = 0.8813$ and so 88.13% of the variation in y is being explained by x.

The final line gives the result of using the ANOVA table to assess the model fit.

Source	DF	SS	MS	F
Regression	$p - 1$	SSR	MSR	MSR/MSE
Error	$n - p$	SSE	$s^2 = \text{MSE}$	
Total	$n - 1$	SST		

Model Fitting

In SLR, the ANOVA table tests

$$H_0 : \beta_1 = 0$$

$$H_1 : \beta_1 \neq 0$$

The TS is the F value and the critical value and p-values are found in the F tables with $(p - 1)$ and $(n - p)$ degrees of freedom.

This output gives the p-value = 0, therefore there is enough evidence to reject H_0 and conclude that there is a significant linear relationship between y and x. The full ANOVA table can be accessed using :

```
anova(model)
```

Analysis of Variance Table

Response: y

```

          Df Sum Sq Mean Sq F value    Pr(>F)
x           1 41329   41329  348.91 < 2.2e-16 ***
Residuals 47  5567     118
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1

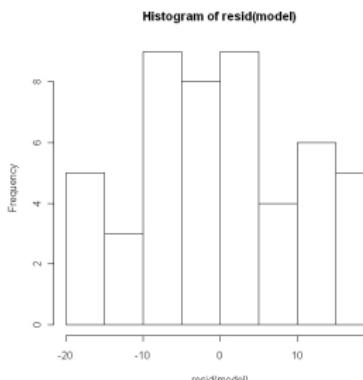
```

Diagnostics

Once the model has been fitted, must then check the residuals.
The residuals should be independent and normally distributed with mean of 0 and constant variance.

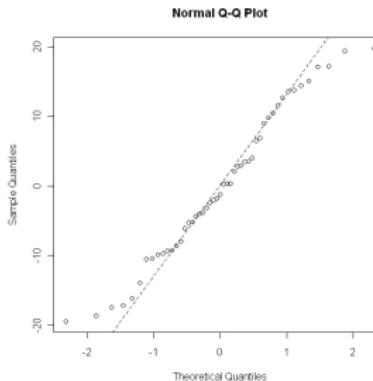
A Q-Q plot checks the assumption of normality (can also use a histogram as in MINITAB) while a, plot of the residuals versus fitted values gives an indication as to whether the assumption of constant variance holds.

```
hist(resid(model))          OR  hist(model$resid)
```



Diagnostics

```
qqnorm(resid(model))  
qqline(resid(model), lty=2)
```

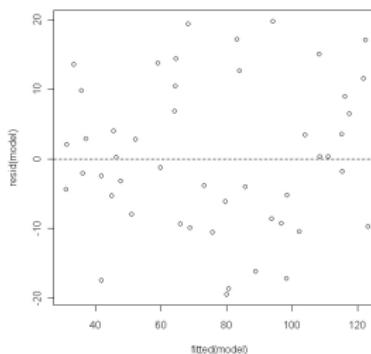


The histogram is approximately the symmetric and bell-shaped indicating a normal distribution. The Q-Q plot shows that the majority of the residuals lie in an approximate straight line, i.e. have a normal distribution. However, there is some evidence of heavy tails.

Diagnostics

To check for constant variance:

```
plot(fitted(model), resid(model))
abline(h=0, lty=2) # Adds a horizontal line at y=0.
```



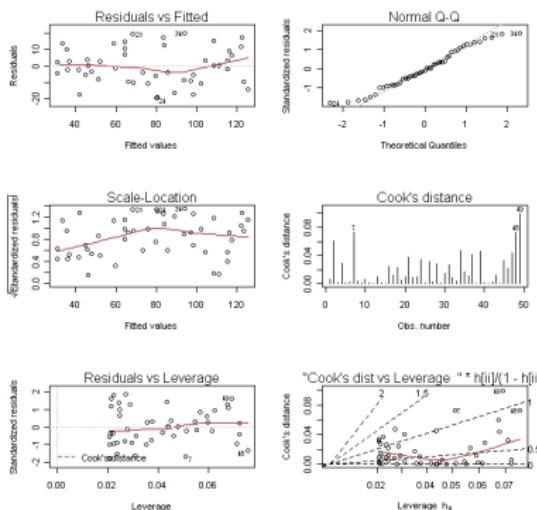
This plot should be randomly scattered above and below the zero line, no evidence of curvature and no evidence of increasing variance for larger values of \hat{y} . Can see that this is the case here - residuals do have constant variance.

Overall - happy with the model.

Diagnostics

When you supply a model object to the plot function in R, there are 6 possible diagnostic plots of the residuals produced. Four of the plots are shown by default, but can specify to show any subset (or all) of the 6 choices using which.

```
par(mfrow=c(3,2))
plot(model, which=1:6)
```



Diagnostics

Can suppress the added smooth (red line) using:

```
plot(model, which=1:6, add.smooth=F)
```

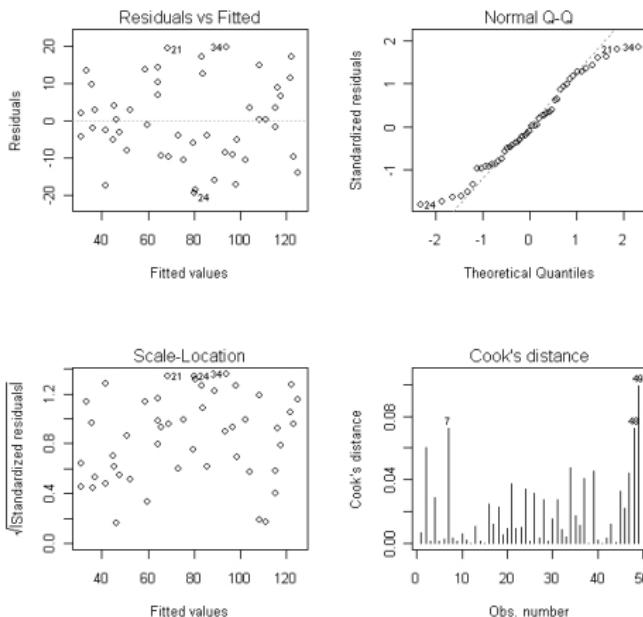
To plot only the first 4 plots:

```
par(mfrow=c(2,2))
plot(model, which=1:4, add.smooth=F)
par(mfrow=c(1,1))
```

To plot the first, second and fourth plots:

```
par(mfrow=c(2,2))
plot(model, which=c(1,2,4), add.smooth=F)
par(mfrow=c(1,1))
```

Diagnostics



Any possible outliers are identified in each of the first 3 plots. The Cook's distance plot identifies high leverage points

Prediction

Suppose we want to know the predicted value \hat{y} at $x = 30$. Could write out the equation using the parameter estimates and fill in the required value for x :

```
19.94379 + 2.07497*30  
[1] 82.19289
```

Alternatively, could use the predict function:

```
predict(model, list(x=30))  
1  
82.19289
```

For linear plots use abline for superimposing the model on a scatterplot of the data points. For curved responses, use the predict function to generate the lines.

Confidence and Prediction Intervals

Fitted lines are often presented with uncertainty bands around them. There are two types of bands:

- Confidence bands - refer to the POPULATION.
- Prediction bands - refer to an INDIVIDUAL.

One way to get intervals:

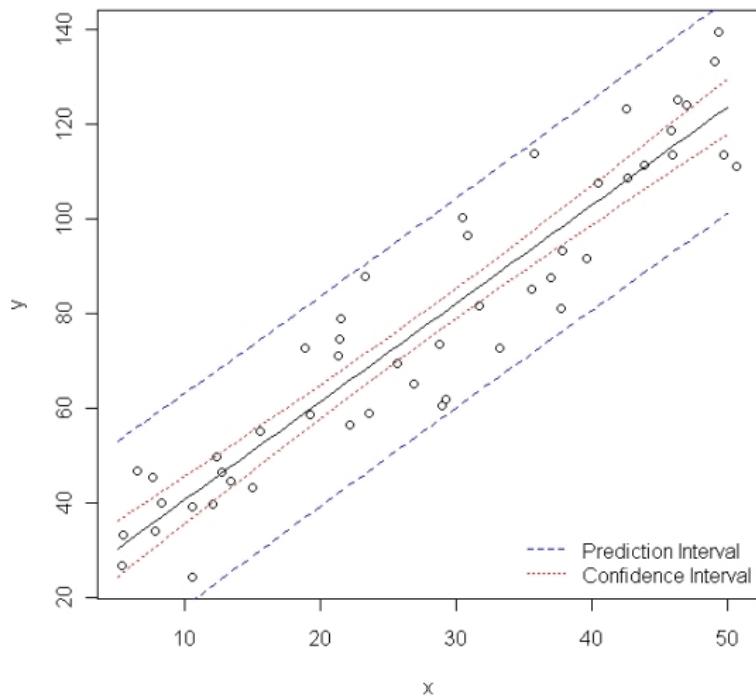
```
predict(model, interval="confidence")
```

```
predict(model, interval="prediction")
```

In order plot the bands on the same plot as the fitted line, use the following:

```
grid <- seq(5,50)
pi <- predict(model, list(x=grid), interval="prediction")
ci <- predict(model, list(x=grid), interval="confidence")
plot(SLR1$x, SLR1$y, xlab="x", ylab="y")
matlines(grid, pi, lty=c(1,2,2), col=c("black", "blue","blue"))
matlines(grid, ci, lty=c(1,3,3), col=c("black", "red","red"))
legend("bottomright", legend=c("Prediction Interval", "Confidence Interval"),
lty=2:3, col=c("blue","red"), bty="n")
```

Confidence and Prediction Intervals



Polynomial Regression

The relationship between two variables can often turn out not to be a straight line. One way of addressing the problem is to use polynomial regression, where additional powers of the explanatory variable x (x^2 , x^3 , etc.) are added to the model to improve the fit.

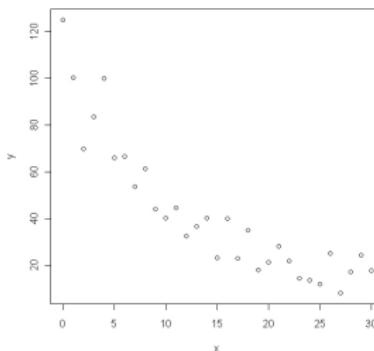
$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_p x_i^p + \varepsilon_i$$

The data in `decay.txt` contain measurements of radioactive emissions (y) over time (x). Read this data into a data frame called `decay`.

```
decay <- read.table("C:/.../decay.txt", header=T)
decay
  x      y
1 0 125.00000
2 1 100.24886
3 2  70.00000
4 3  83.47080
5 4 100.00000
6 5  65.90787
...
```

The first step is to create a scatterplot of these data. The SLR fit is also added:

```
attach(decay)  
names(decay)  
[1] "x" "y"  
  
plot(x,y)
```



Can see that the relationship is non-linear.

First fit the SLR model:

```
lm.decay <- lm(y~x)
abline(coef(lm.decay), lty=2)
```

```
summary(lm.decay)
```

Call:

```
lm(formula = y ~ x)
```

Residuals:

Min	1Q	Median	3Q	Max
-19.065	-10.029	-2.058	5.107	40.447

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	84.5534	5.0277	16.82	< 2e-16 ***
x	-2.8272	0.2879	-9.82	9.94e-11 ***

Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 1

Residual standard error: 14.34 on 29 degrees of freedom

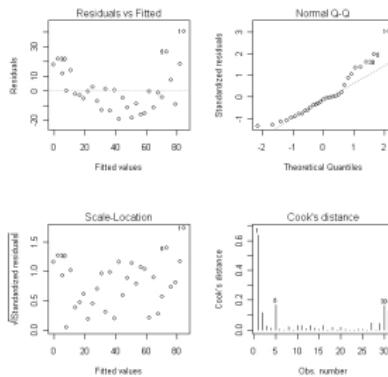
Multiple R-squared: 0.7688, Adjusted R-squared: 0.7608

F-statistic: 96.44 on 1 and 29 DF, p-value: 9.94e-11

Model looks ok, everything is significant. BUT must check the residuals to get full picture.

Plots of the residuals:

```
par(mfrow=c(2,2))
plot(lm.decay, which=1:4, add.smooth=F)
par(mfrow=c(1,1))
```



Clearly, assumptions do not hold. Need to look at other models...

Quadratic Model

Try a quadratic model: $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$. Must include $I(x^2)$ term in model specification.

```
qm.decay <- lm(y ~ x + I(x^2))
```

OR

```
qm.decay <- update(lm.decay, ~.+I(x^2))
```

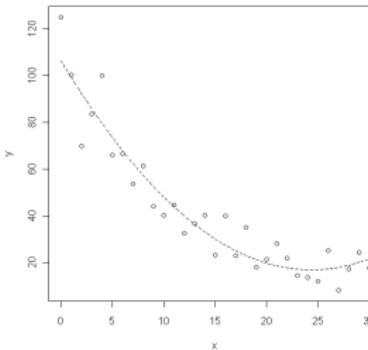
The update command can be used to update the current model `lm.decay`. The `~ .` tells R to re-fit the model using all of the terms in the current model. The `+I(x^2)` term then adds the extra quadratic term. (See `? update` for more information.)

Quadratic Model

Can plot the fitted quadratic model on the scatterplot. Easiest to use predict.

First need to create a sequence of x values at which to evaluate. Then plot the corresponding fitted values.

```
plot(x, y)
newx <- seq(0,30, 0.1)
pred.y <- predict(qm.decay, list(x=newx, x2=newx^2))
lines(newx, pred.y, lty=2)
```



Quadratic Model

```
summary(qm.decay)
```

Call:

```
lm(formula = y ~ x + I(x^2))
```

Residuals:

Min	1Q	Median	3Q	Max
-22.301	-6.044	-1.604	4.224	20.581

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	106.38880	4.65627	22.849	< 2e-16 ***
x	-7.34485	0.71844	-10.223	5.90e-11 ***
I(x^2)	0.15059	0.02314	6.507	4.73e-07 ***

Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 1

Residual standard error: 9.205 on 28 degrees of freedom

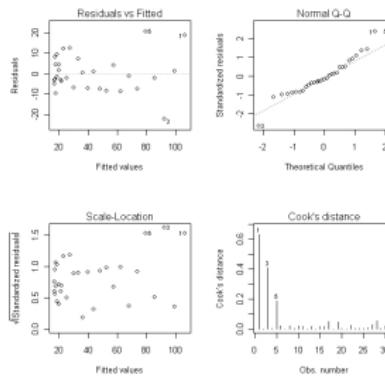
Multiple R-squared: 0.908, Adjusted R-squared: 0.9014

F-statistic: 138.1 on 2 and 28 DF, p-value: 3.122e-15

The new term is also significant.

Again carry out model diagnostics using plot:

```
par(mfrow=c(2,2))
plot(qm.decay, which=1:4, add.smooth=F)
par(mfrow=c(1,1))
```



Residuals have improved, though still some evidence of non-constant variance and non-normality. Can we improve further??

Comparing Models

Now have two different models describing the same data. Is the simpler model better than the more complicated model or vice versa? Remember concept of *parsimony*. Can compare the models using ANOVA:

```
anova(lm.decay, qm.decay)
Analysis of Variance Table
```

```
Model 1: y ~ x
Model 2: y ~ x + I(x^2)
Res.Df   RSS Df Sum of Sq    F    Pr(>F)
1     29 5960.6
2     28 2372.6  1    3588.1 42.344 4.727e-07 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1    1
```

Since the p-value = 0, conclude that the more complex model is preferred.

Summary SLR

SLR analysis involves:

- Creating a scatterplot to determine the nature of the relationship between x and y
- If the relationship is linear, measuring the strength of the relationship using the correlation coefficient
- Fitting the best model by estimating parameter values from data
- There are always lots of different possible models to describe a given data set
- Using diagnostic plots of the residuals to check the adequacy of the fitted model. Must check for non-constant variance and non-normal errors.
- If the relationship is non-linear, fit e.g. polynomial, exponential, non-linear model and use predict to generate the fitted curve for plotting.

Multiple Regression

Multiple regression is carried out when there is more than one explanatory variable x . The multiple linear regression model has the form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \varepsilon.$$

x_1, x_2, \dots, x_p are the explanatory variables and we need to use the sample data to find estimates of $\beta_0, \beta_1, \dots, \beta_p$.

The file GPA.txt contains the college GPA values (College- y) for 100 final year college students. Also provided are these students' high school GPA values (High.School - x_1), SAT scores (SAT - x_2) and the number of letters of recommendation received by each student (Letters - x_3).

Want to perform a regression analysis for these data.

Scatterplots

Read the data from GPA.txt into a data frame called GPA.

```
GPA <- read.table("C:/.../GPA.txt", header=T)
```

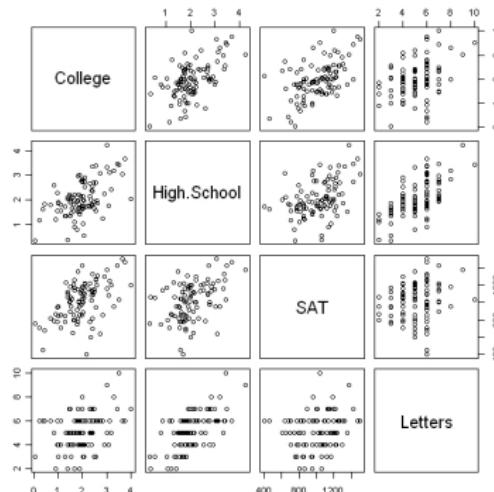
The first step is to create scatterplots for the data, to determine the nature of the relationship(s) between y and the explanatory variables.

Could create individual scatterplots for y vs x_1 , x_2 and x_3 as in SLR. However the `pairs` command does this for us.

Also creates scatterplots of x_1 vs x_2 , x_1 vs x_3 and x_2 vs x_3 . These additional plots will give us an indication as to whether *multicollinearity* exists between the explanatory variables.

Scatterplots

`pairs(GPA)`



Looking along the top row, there seems to be a relatively strong linear relationship between college GPA and high school GPA, and college GPA and SAT scores. There is also a linear relationship between college GPA and letters of recommendation, though not as strong. There appears to be evidence of multicollinearity between all 3 explanatory variables.

Correlation

Can use `cor` to calculate the correlation between y and all of the xs and between the xs . This results in a correlation matrix:

```
cor(GPA)
```

	College	High.School	SAT	Letters
College	1.0000000	0.5451980	0.5227546	0.3500768
High.School	0.5451980	1.0000000	0.4326248	0.6265836
SAT	0.5227546	0.4326248	1.0000000	0.2175928
Letters	0.3500768	0.6265836	0.2175928	1.0000000

Look at the values either above or below the diagonal (symmetric matrix).

College GPA has the highest correlation with high school GPA, followed by SAT scores and finally letters of recommendation.

There is quite high correlation between letters of recommendation and high school GPA ($r = 0.63$). There is also evidence of some correlation between high school GPA and SAT scores ($r = 0.43$) and letters of recommendation and SAT scores ($r = 0.22$).

Hypothesis Test Correlation Matrix

"cor.test" can only perform a hypothesis test for two variables at a time. Will write a function to work on the correlation matrix and store the p-values:

```
cormat.test <- function(dat=GPA){  
    p.mat <- matrix(0, nrow=ncol(dat), ncol=ncol(dat))  
    colnames(p.mat) <- colnames(dat)  
    rownames(p.mat) <- colnames(dat)  
    for(i in 1:ncol(dat)){  
        for(j in 1:ncol(dat)){  
            p.mat[i,j] <- cor.test(dat[,i], dat[,j])$p.value  
        }  
    }  
    return(p.mat)  
}  
cormat.test(GPA)
```

	College	High.School	SAT	Letters
College	0.000000e+00	4.492104e-09	2.416862e-08	3.563919e-04
High.School	4.492104e-09	0.000000e+00	6.951243e-06	3.103073e-12
SAT	2.416862e-08	6.951243e-06	0.000000e+00	2.965269e-02
Letters	3.563919e-04	3.103073e-12	2.965269e-02	0.000000e+00

Hypothesis Test Correlation Matrix

In each case testing:

$$H_0 : \rho_{yx_i} = 0 \quad \text{OR}$$

$$H_1 : \rho_{yx_i} \neq 0$$

$$H_0 : \rho_{x_i x_j} = 0$$

$$H_1 : \rho_{x_i x_j} \neq 0$$

All of the p-values = 0 \Rightarrow enough evidence to reject H_0 in all cases and conclude that there is a significant linear relationship between y and each of x_1 , x_2 and x_3 .

There are also significant relationships between the explanatory variables: x_1 and x_2 , x_1 and x_3 , x_2 and x_3 (i.e. multicollinearity). All explanatory variables may not belong (or be required) in the model.

Model Fitting

In the simplest case, fit a multiple linear regression model

$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon$. This is set up by putting + between the explanatory variables:

```
attach(GPA)
GPA.mod <- lm(College ~ High.School + SAT + Letters)
summary(GPA.mod)
Call:
lm(formula = College ~ High.School + SAT + Letters)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.097945	-0.440695	-0.009399	0.385926	1.760638

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)		
(Intercept)	-0.1532639	0.3229381	-0.475	0.636156		
High.School	0.3763511	0.1142615	3.294	0.001385 **		
SAT	0.0012269	0.0003032	4.046	0.000105 ***		
Letters	0.0226843	0.0509817	0.445	0.657358		

Signif. codes:	0 ***	0.001 **	0.01 *	0.05 .	0.1	1

Model Fitting

Examine the ANOVA results first. This time the ANOVA table tests

$$H_0 : \beta_1 = \beta_2 = \beta_3 = 0$$

$$H_1 : \text{at least one } \beta_j \neq 0$$

Residual standard error: 0.5895 on 96 degrees of freedom

Multiple R-squared: 0.3997, Adjusted R-squared: 0.381

F-statistic: 21.31 on 3 and 96 DF, p-value: 1.160e-10

p-value = 0 \Rightarrow enough evidence to reject H_0 and conclude that there is a significant linear relationship between y and at least one x .

Model Fitting

The Coefficients: section performs t-tests: given that the other variables are in the model, is this variable significant?

$$H_0 : \beta_j = 0$$

$$H_1 : \beta_j \neq 0$$

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-0.1532639	0.3229381	-0.475	0.636156
High.School	0.3763511	0.1142615	3.294	0.001385 **
SAT	0.0012269	0.0003032	4.046	0.000105 ***
Letters	0.0226843	0.0509817	0.445	0.657358

Can see that High.School (x_1) and SAT (x_2) are significant and should stay in the model. However, Letters is no longer significant and can be removed.

Model Fitting

The ANOVA table for these data is found using:

```
anova(GPA.mod)
```

```
Analysis of Variance Table
```

```
Response: College
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
High.School	1	16.518	16.518	47.538	5.699e-10 ***
SAT	1	5.627	5.627	16.194	0.0001141 ***
Letters	1	0.069	0.069	0.198	0.6573582
Residuals	96	33.358	0.347		

This table represents successive tests resulting in (from the bottom upwards) a stepwise removal of each term in the model until only High.School is left.

Model Fitting

Can compare the various models possible using anova:

```
GPA.mod1 <- lm(College ~ High.School)
GPA.mod2 <- lm(College ~ High.School + SAT)
```

```
anova(GPA.mod1, GPA.mod2)
Analysis of Variance Table
```

```
Model 1: College ~ High.School
Model 2: College ~ High.School + SAT
Res.Df   RSS Df Sum of Sq    F    Pr(>F)
1      98 39.054
2      97 33.427  1      5.627 16.329 0.0001067 ***
```

There is a significant improvement in the model by adding SAT scores. Model 2 is preferred.

Model Fitting

Compare Model 2 and the full model:

```
anova(GPA.mod2, GPA.mod)
```

Analysis of Variance Table

Model 1: College ~ High.School + SAT

Model 2: College ~ High.School + SAT + Letters

	Res.Df	RSS	Df	Sum of Sq	F	Pr(>F)
1	97	33.427				
2	96	33.358	1	0.069	0.198	0.6574

There is no improvement in explanatory power of the model by adding Letters. Again Model 2 is preferred.

OK to do this by hand when the number of variables is small.
Otherwise, use the step command is useful.

Stepwise Regression

Stepwise regression is a model selection tool that is useful for determining the “best” model from a set of variables. The main approaches are:

- Forward selection - start with no variables in the model and then including them one by one to determine if they are statistically significant.
- Backward elimination - start with all variables in the model and test them one by one for statistical significance, deleting any that are not significant.
- A combination of the above - test at each stage for variables to be included or excluded.

The `step` command carry carry out any of the three options outlined above.

The model selected by the stepwise regression analysis is returned.

Stepwise Regression

Must give step a model to begin with. Start with max model (Use backward or both).

```
GPA.max <- lm(College ~ . , data=GPA)
final.mod <- step(GPA.max, direction="both")
summary(final.mod)
Call:
lm(formula = College ~ High.School + SAT, data = GPA)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.121526	-0.441197	0.009536	0.381976	1.803556

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)		
(Intercept)	-0.0881312	0.2866638	-0.307	0.759169		
High.School	0.4071133	0.0905946	4.494	1.94e-05 ***		
SAT	0.0012167	0.0003011	4.041	0.000107 ***		

Signif. codes:	0 ***	0.001 **	0.01 *	0.05 .	0.1	1

Residual standard error: 0.587 on 97 degrees of freedom

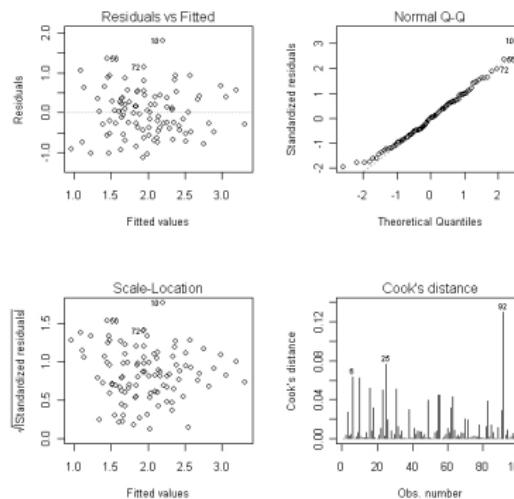
Multiple R-squared: 0.3985, Adjusted R-squared: 0.3861

F-statistic: 32.13 on 2 and 97 DF, p-value: 1.963e-11

Diagnostics

Have found that the optimal model is to use High.School and SAT only. Must check the residuals:

```
par(mfrow=c(2,2))
plot(fin.mod, which=1:4, add.smooth=F)
par(mfrow=c(1,1))
```



Further Modelling

More complex models can be fitted by updating the current model.

- Polynomial terms - `update(fin.mod, ~. + I(High.School^2))` .
- Interaction terms - `update(fin.mod, ~. + High.School:SAT)`.
- Nested terms - `lm(y ~ x1 + x2 + x1/x2)` .
- Dummy variables, etc.

See “An Introduction to R” available on CRAN for full details on model specification.

R can also perform:

- ANOVA analysis;
- ANCOVA analysis;
- Logistic regression;
- Generalised linear modelling;
- Generalised additive modelling;
- etc, etc, etc

Exercises I

Regression Analysis

1. `alps.dat` describes the boiling point of water (y) at different barometric pressures (x).
 - (a) Construct a scatterplot and comment. Is SLR appropriate for these data? Why?
 - (b) Calculate the correlation coefficient and test its significance.
 - (c) Fit a SLR model. Add the fitted line to the scatterplot.
 - (d) Examine the residuals using diagnostic plots. Are there any outliers? Are there any high leverage points?
2. The data in `stopping.dat` give the stopping distances of a car travelling at various speeds. Want to model the relationship between Distance (y) and Speed (x).
 - (a) Create a suitable plot of these data and describe the relationship between the two variables. Is SLR appropriate here?
 - (b) Fit a SLR model, modelling y versus x and examine the residuals.

Exercises II

Regression Analysis

- (c) Fit an appropriate polynomial model (e.g. quadratic, cubic) and compare with the SLR model. Does including the extra polynomial term significantly improve the model?
- (d) Add the chosen model to the scatterplot using predict.
- (e) Examine the residuals of the polynomial model. Do the assumptions of normality and constant variance hold?
- (f) Try fitting the model

$$\log(\text{Distance}) = \beta_0 + \beta_1 \text{Speed} + \beta_2 \text{Speed}^2. \text{ Use}$$

`lm(log(Distance) ~ Speed + I(Speed^2))`

Examine the residuals. Do they improve?

Exercises III

Regression Analysis

3. The data in `bodyfat.dat` give the body fat, triceps skinfold thickness, thigh circumference and midarm circumference for 20 healthy females aged 20 to 34. The body fat percentage was obtained by a cumbersome and expensive procedure requiring the immersion of the person in water. It would therefore be very helpful if a regression model with some or all of these predictor variables could provide reliable predictions of the amount of body fat, since the measurements needed for the predictor variables are easy to obtain.
 - (a) Using a suitable plotting command, determine if there is a relationship between body fat and any of the 3 predictor variables. Is there evidence of multicollinearity?
 - (b) Determine the correlation matrix and test the significance of the correlation coefficients.
 - (c) Fit a MLR model using all 3 variables and determine which are significant.

Exercises IV

Regression Analysis

- (d) Find the “best” model using the step function. Fit this model and examine the residuals.
4. The file `body.dat` contains 21 body dimension measurements as well as age, weight, height, and gender on 507 individuals. Read this data into a data frame called `body`. `weight` is the response variable and the remaining 24 variables are the explanatory variables.
- (a) Fit a model for `weight` versus all of the remaining variables except `gender`.
 - (b) From the output, determine what variables are significant in the model.
 - (c) Based on these results, fit the reduced model omitting any non-significant terms.
 - (d) Examine the residuals from the new model and comment on the validity of the assumptions.

Exercises V

Regression Analysis

- (e) Perform a backward stepwise regression analysis on the original model containing all variables except gender, and compare the resulting model to the model you chose in part (b).

Simulation

Why Simulate?

Simulation methods enable us to learn important information about a process we are interested in. For example, we might like to know how well a student would do by random guessing on a multiple choice exam. Can get some idea using simulation.

Statistical simulation methods use random numbers to simulate the process of interest. Often we are interested in studying the accuracy of our approximations or the effect of violating any assumptions we may be used.

We will use many of the skills already learned on this course, i.e. random number generation, looping, functions, etc. to carry out the simulation studies.

Properties of Sample Mean

The sample mean \bar{x} is known to be an *unbiased* estimator of the population mean μ . To investigate that this is in fact the case, we can perform a simulation study.

Say we know that the variable of interest X (e.g. IQ, weight, age) has a known distribution such that $X \sim N(100, 15^2)$. Need to generate a random sample of size n from this distribution and calculate the mean. This is considered one simulation step. We then repeat this process a number of times and examine the results.

Properties of Sample Mean

```
mu <- 100
sigma <- 15
n.sim <- 10  # Number of simulations to be carried out.
n <- 20      # Sample size of each simulated data set.
xbar <- rep(NA, n.sim)  # Vector to store the means of each
                        # simulated data set.

for(i in 1:n.sim){
  set.seed(i)      # Allows us to repeat our simulation study
                    # using exactly the same numbers.

  x <- rnorm(n, mu, sigma)
  xbar[i] <- mean(x)
}
```

Properties of Sample Mean

The sample mean for the 10 samples each of size 20 are:

```
xbar  
[1] 102.85786 102.93192 97.49214 105.64444 95.79163 102.90547 106.64870  
[8] 96.98678 97.68591 99.09201
```

If \bar{x} is an unbiased estimator of μ then the mean of all 10 sample means should be the same as μ .

```
mean(xbar)  
[1] 100.8037
```

Exercise 1:

Repeat the process using 100 simulations and 1000 simulations.
Compare the results.

```
mean(xbar)      # 100 simulations  
[1] 100.0034
```

```
mean(xbar)      # 1000 simulations  
[1] 99.99943
```

Properties of Sample Mean

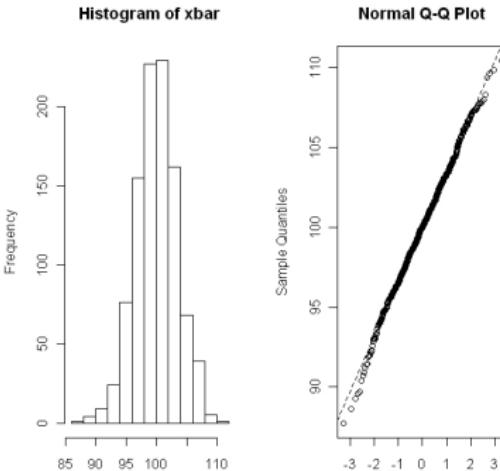
After performing 1000 simulations, can examine the distribution of the sample means. Remember CLT - should have a normal distribution such that $\bar{x} \sim N(\mu, \sigma^2/n)$.

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

```
hist(xbar)
```

```
qqnorm(xbar)
```

```
qqline(xbar, lty=2)
```



Properties of Sample Variance

Is the sample variance s^2 an unbiased estimate of the population variance σ^2 ?

Exercise 2:

Repeat the process but calculate the variance for each sample instead of the mean. $\sigma^2 = 15^2 = 225$.

```
mean(xvar)      # 10 simulations.
```

```
[1] 210.6519
```

```
mean(xvar)      # 100 simulations.
```

```
[1] 222.2472
```

```
mean(xvar)      # 1000 simulations.
```

```
[1] 222.6586
```

Exercises 3

1. Assume that X has a Poisson distribution with $\lambda = 4$. At each simulation step generate a random sample of size $n = 15$ and calculate the mean and variance. Repeat this 20 and 2000 times. What are the results? (NOTE: $E[X] = \text{Var}(X) = \lambda$)
2. Repeat the process assuming that X has a χ^2 distribution with 9 degrees of freedom. (NOTE: $E[X] = df$ and $\text{Var}(X) = 2 * df$.)

Confidence Intervals

To investigate the coverage of confidence intervals we also use simulation.

Assume our data is such that $X \sim N(\mu = 100, \sigma^2 = 15^2)$. If 1000 95% CIs are constructed using the same sample size etc., the true population mean μ should be contained in 95% (950) of them. If 1000 99% CIs are constructed, μ should be contained in 99% (990) of them. To investigate if this is the case, perform the following simulation study. Assume that σ is known, i.e. use the normal distribution to look up the values used in the CI.

CI has the form:

$$\bar{x} - Z_{1-\alpha/2} \times \frac{\sigma}{\sqrt{n}} < \mu < \bar{x} + Z_{1-\alpha/2} \times \frac{\sigma}{\sqrt{n}}.$$

Confidence Intervals

```
mu <- 100
sigma <- 15
n <- 100
n.sim <- 1000
l.ci <- rep(NA, n.sim)
u.ci <- rep(NA, n.sim)
for(i in 1:n.sim){
  set.seed(i)
  x <- rnorm(n, mu, sigma)
  l.ci[i] <- mean(x) - qnorm(0.975)*sigma/sqrt(n)
  u.ci[i] <- mean(x) + qnorm(0.975)*sigma/sqrt(n)
}
cont.mu <- (l.ci <= mu) & (u.ci >= mu)
length(cont.mu[cont.mu==TRUE])
[1] 958
```

958 out of the 1000 CIs contain the true mean - slightly higher than 950.

Repeat the process for a 99% CI. How many of the CIs contain the true mean? Answer: 991

Confidence Intervals

Now assume that σ is unknown and must be estimated from each of the samples created during the simulation. For large samples can use values from the normal or t-distribution in the CI calculations.

```
mu <- 100
sigma <- 15
n <- 100
n.sim <- 1000
l.ci <- rep(NA, n.sim)
u.ci <- rep(NA, n.sim)
l.cit <- rep(NA, n.sim)
u.cit <- rep(NA, n.sim)
for(i in 1:n.sim){
  set.seed(i)
  x <- rnorm(n, mu, sigma)
  l.ci[i] <- mean(x) - qnorm(0.975)*sd(x)/sqrt(n)
  u.ci[i] <- mean(x) + qnorm(0.975)*sd(x)/sqrt(n)
  l.cit[i] <- mean(x) - qt(0.975, df=n-1)*sd(x)/sqrt(n)
  u.cit[i] <- mean(x) + qt(0.975, df=n-1)*sd(x)/sqrt(n)
}
```

Confidence Intervals

The coverages are:

```
cont.mu <- (l.ci <= mu) & (u.ci >= mu)
cont.mut <- (l.cit <= mu) & (u.cit >= mu)
```

```
length(cont.mu[cont.mu==TRUE])
[1] 955
```

```
length(cont.mut[cont.mut==TRUE])
[1] 959
```

Exercise 4

Is the same true for small samples?

Repeat the previous exercise using samples of size 20. Create 1000 simulated data sets each of size 20.

First construct a 95% CI for each sample using values from the normal distribution.

Next construct a 95% CI for each sample using values from the t-distribution. Compare the coverage results.

Hypothesis Tests

Assume now that $X \sim N(\mu, \sigma^2 = 2^2)$. Want to check the probability of making a Type I error when carrying out a hypothesis test for μ using a particular significance level α .

Say we are interested in testing the hypothesis

$$H_0 : \mu = 3$$

$$H_1 : \mu \neq 3$$

using $\alpha = 0.05$.

Will perform 1000 simulations and use samples of size $n = 100$.

Assume that our data comes from a $N(\mu = 3, \sigma^2 = 2^2)$ distribution, i.e. H_0 is true. When $\alpha = 0.05$, approximately 5% of the p-values generated from the t-test will be < 0.05 , i.e. will reject H_0 when it is actually true (make a Type I error).

Hypothesis Tests

```
mu <- 3
sigma <- 2
n <- 100
n.sim <- 1000
p.value <- rep(NA, n.sim)
for(i in 1:n.sim){
  set.seed(i)
  x <- rnorm(n, mu, sigma)
  p.value[i] <- t.test(x, mu=3, conf.level=0.95, alt="two.sided")$p.value
}
mean(p.value < 0.05)
[1] 0.041
```

Check the probability of getting a Type I error when $\alpha = 0.01$.

Hypothesis Tests

The following example shows how to calculate the power of a test for a given shift in the data mean.

Say our data now comes from a $N(\mu = 4, \sigma^2 = 2^2)$, i.e. the mean has shifted from 3 to 4 and H_0 is no longer true.

The power of the test is the probability of making the correct decision, i.e. the probability of rejecting H_0 when it is in fact false.
Still testing the hypothesis

$$H_0 : \mu = 3$$

$$H_1 : \mu \neq 3$$

Hypothesis Tests

```
mu <- 4
sigma <- 2
n <- 100
n.sim <- 1000
p.value <- rep(NA, n.sim)
for(i in 1:n.sim){
  set.seed(i)
  x <- rnorm(n, mu, sigma)
  p.value[i] <- t.test(x, mu=3, conf.level=0.95, alt="two.sided")$p.value
}
mean(p.value < 0.05)
[1] 0.997
```

The power of the test for detecting a shift $\delta = \pm 1$ using a sample of size 100 is 0.997.

Exercises 5

1. What is the power of the test for detecting a shift $\delta = \pm 0.2$ using a sample of size 50?
2. What is the power of the test for detecting a shift of $\delta = \pm 0.2$ using a sample of size 100?
3. What is the power of the test for detecting a shift of $\delta = \pm 0.2$ using a sample of size 500? What do you notice?
4. Let $\delta = \pm 0.5$ and repeat the previous 3 exercises.
5. Change α to 0.01 and repeat the previous 6 exercises.

Bootstrapping

Bootstrapping is a resampling technique used in statistics to estimate the properties of an estimator (such as its variance). For example, if we are interested in the height of people world wide. As we can not measure all the population, we sample only a small part of it. From that sample only one value of a statistic can be obtained, i.e one mean, or one standard deviation etc., and hence we don't see how variable that statistic is. The bootstrap allows us to create many samples (from our single sample) and thus calculate many values of the statistic of interest. Can then get a measure of how variable that estimate is.

Bootstrapping

The bootstrap is typically applied when the assumptions required for a particular statistical method are in doubt, or where very complicated formulas are required to calculate standard errors.

The advantage of bootstrapping is its simplicity - it is straightforward to apply the bootstrap to derive estimates of standard errors and confidence intervals for complex statistics such as percentile points, proportions, odds ratios, and correlation coefficients.

How does it work?

- A random sample of size n is drawn from the population of interest and stored.
- Create a new sample of size n from this original data by sampling WITH REPLACEMENT from it.
- Calculate the value of interest from this new sample. For example, calculate the mean, calculate the odds ratio, fit a particular model, etc.
- Repeat the previous 2 steps a large number of times, usually denoted by $B = 10000, 100000$, etc.
- Use the sampling distribution of the estimates thus computed to be an approximation to the “true” population sampling distribution. Confidence intervals, hypothesis tests, etc. can then be performed using this sampling distribution.

How does it work?

One way to estimate confidence intervals from bootstrap samples is to take the α and $1 - \alpha$ quantiles of the estimated values. These are called bootstrap percentile intervals.

Take the B estimates and order them from smallest to largest. The lower confidence limit is given by the value at position $B * (\alpha/2)$ in this ordered list, while the upper confidence limit is given by the value at position $B * (1 - (\alpha/2))$ in the ordered list.

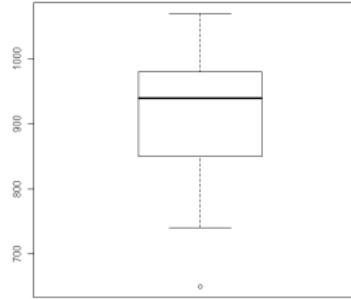
For example, let $B = 1000$ and say we are constructing a 95% CI. The lower limit is given by the value at position $1000 * (0.025) = 25$ in the ordered list and the upper limit is given by the value at position $1000 * (1 - 0.025) = 975$ in the ordered list.

Example

The data in `light.txt` contains Michelson's (1879) data on estimating the speed of light. Read this data into a data frame called `light`. The speed is $299\,000 \text{ km s}^{-1}$ plus the value recorded in the file.

This is a small sample $n = 20$ and the true variance is unknown, and so we would usually use the t-test and t-distribution to analyse these data. However, create a boxplot of these data:

```
attach(light)  
boxplot(speed)
```



Example

Create a Q-Q plot

```
qqnorm(speed)
qqline(speed, lty=2)

library(nortest)
ad.test(speed)
```

These data are clearly skewed (i.e. non-normal) and so the assumption that the data are normally distributed underlying the use of the t-distribution is violated.

Standard testing procedures and confidence interval estimates will not be suitable for these data. One solution is to use the bootstrap.

Example

Say we want to test how likely it is that the speed of light is $299,950 \text{ km s}^{-1}$. The data have 299,000 subtracted from them and so the test value is 950.

```
B <- 10000      # No. bootstrap samples to take
xbar <- rep(NA, B)
for(i in 1:B){
  xbar[i] <- mean(sample(speed, replace=TRUE))
}
hist(xbar)
```

To determine the p-value:

```
ord.xbar <- sort(xbar)
p.value <- length(ord.xbar[ord.xbar >= 950])/B
[1] 0.0323
```

What if we wanted to test how likely it is that the speed of light is $299,978 \text{ km s}^{-1}$?

```
length(ord.xbar[ord.xbar >= 978])/B
[1] 2e-04
```

Example

To construct a 95% CI:

```
lower.ci <- ord.xbar[B*0.025]  
lower.ci  
[1] 861.5
```

```
upper.ci <- ord.xbar[B*0.975]  
upper.ci  
[1] 952.5
```

Can also just use the quantile function in R:

```
ci <- quantile(xbar, c(0.025, 0.975))  
ci  
2.5% 97.5%  
861.5 952.5
```

Example

If you compare this to the confidence interval obtained using normal theory:

```
lower.cin <- mean(speed) - qnorm(0.975)*sd(speed)/sqrt(20)
lower.cin
[1] 863.015
```

```
upper.cin <- mean(speed) + qnorm(0.975)*sd(speed)/sqrt(20)
upper.cin
[1] 954.985
```

OR the t-distribution:

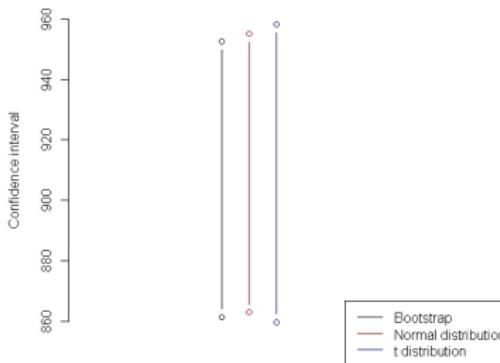
```
lower.cit <- mean(speed) - qt(0.975, df=19)*sd(speed)/sqrt(20)
lower.cit
[1] 859.8931
```

```
upper.cit <- mean(speed) + qt(0.975, df=19)*sd(speed)/sqrt(20)
upper.cit
[1] 958.1069
```

Values are quite close. Construct a plot to examine further.

Example

```
plot(c(5,20), c(850, 970), type="n", ylab="Confidence interval",
xlab="", axes=F)
axis(1, labels=F, col.ticks="white")
axis(2)
points(c(10, 10), ci, type="b")
points(c(11,11), c(lower.cin, upper.cin), type="b", col="red")
points(c(12,12), c(lower.cit, upper.cit), type="b", col="blue")
legend("bottomright", legend=c("Bootstrap", "Normal distribution",
"t distribution"), col=c("black", "red", "blue"), lty=1)
```



Exercise 1

Store the data from `skewdata.txt` into a data frame called `skew`. Attach the data frame to the search path in R.

1. Create a histogram, a boxplot and a Q-Q plot of these data and describe the distribution.
2. Set up a bootstrap simulation consisting of $B = 10,000$ bootstrap samples from these data and calculate the mean for each sample. Construct a 98% bootstrap CI for the mean and compare with the corresponding CIs constructed using values from the normal distribution and t-distribution.
3. Repeat the exercise for samples of size between 5 and 30 and plot the resulting bootstrap CIs on the same plot. This can be done using nested for loops.

Exercise II

```
plot(c(0,30), c(0,60), type="n", xlab="Sample size",
ylab="Confidence interval")
size <- seq(5,30, by=5)
for(k in size){
  ...
  for(i in 1:B){
    ...
  }
  points(..., type="b")
}
```

4. Add confidence bands to the plot for the normal distribution and the t-distribution. Use different line types and colours for the lines. Add a legend indicating which lines are which. Hint:

Exercise III

```
x.vals <- seq(5,30,0.1)
ci.low <- mean(values) - qnorm(0.975)*sd(values)/sqrt(x.vals)
ci.high <- ...

cit.low <- mean(values) - qt(0.975, df=xv-1)*sd(values)/sqrt(x.vals)
cit.high <- ...

lines(x=..., y=..., lty=1, col="red")
lines(x=..., y=..., lty=1, col="red")

lines(x=..., y=..., lty=2, col="blue")
lines(x=..., y=..., lty=2, col="blue")

legend(..., legend=..., col=..., lty=...)
```

Project 1 - 10%

Write an R script file to:

- (a) Create a vector called **Height** with the following entries:

181 160 174 170 172 165 161 167 164 166

- (b) Create a vector called **Shoe.size** with the following entries:

44 38 42 43 43 39 38 38 39 38

- (c) Create a vector called **Gender** with the following entries:

"M" "F" "F" "M" "M" "F" "F" "F" "F" "F"

- (d) Using the vectors created above, create a data frame called **students** with variable names **height**, **shoesize** and **gender**.

The following questions require using the data frame you created in part (d):

- (e) Calculate the average height for all students.
- (f) Calculate the average height for male students versus female students.
- (g) Create side-by-side boxplots of the heights of male students and heights of female students. Put the title “Height Males Vs Height Females” on the graph and label the boxplots appropriately, e.g. under the boxplot of male heights have the label “Males” and under the boxplot of female heights have the label “Females”. Save this graph as BoxplotHgt.png.
- (h) Extract the data for female students who have shoe size 38.

You are asked to submit a single Adobe PDF file prepared using LATEX containing:

- a listing of your R script file (see
<http://jkcray.maths.ul.ie/ms4024/LaTeX-Files/HowToIncludeRCode.tex>)
- the results obtained for parts (e), (f) and (h)
- the plot created by your code
- a (very) short summary of your work.

Project 2 - 15%

- The following is the formula for converting $^{\circ}\text{F}$ into $^{\circ}\text{C}$:

$$\text{degC} = \frac{5}{9}(\text{degF} - 32).$$

Create an R function called `FtoC` which converts a temperature given in $^{\circ}\text{F}$ into $^{\circ}\text{C}$. The function should have

- ▶ a single input argument (the $^{\circ}\text{F}$) and
- ▶ a single output value (the result of converting the input value into $^{\circ}\text{C}$).
- Check your results using 78 $^{\circ}\text{F}$ - you should get approx. 25.56 $^{\circ}\text{C}$.

- You are also asked to create a second R function called `createSamp` which generates a random sample of size $n = 20$ from a Normal distribution with $\mu = 35$ and $\sigma = 2$. This function should
 - ▶ store the generated values in a vector called `mySamp`;
 - ▶ create a histogram of `mySamp` with an appropriate title of your choice and x-axis label “Degrees Fahrenheit”;
 - ▶ return the vector `mySamp`.

- Finally you are asked to write an R function called Run which:
 - ▶ uses the function `createSamp` to generate a random sample of numbers and stores the result in a vector called `degFar`;
 - ▶ uses an appropriate looping structure (`apply`, `for`, `while`, etc.) to apply the function `FtoC` to each element of the vector `degFar`, converting each element to $^{\circ}\text{C}$. The results should be stored in a vector called `degCen`;
 - ▶ creates a boxplot of `degCen` with an appropriate title of your choice and y-axis label “Degrees Celcius”.
 - ▶ generates a 98% CI for the values in `degCen` and prints the resulting interval to the screen.

You are asked to submit a single Adobe PDF file prepared using LATEX containing:

- listings of your R functions
 - ▶ FtoC;
 - ▶ createSamp;
 - ▶ Run
- the result(s) printed in the R console after running the function Run;
- the plots created by your code;
- a short summary of your work (approximately 1 page).

Project 3 - 25%

The data generated in this project contains the time taken to write the following message

the quick brown fox jumps over the lazy dog

on a mobile phone. The message had to be typed with no errors, no abbreviations and no use of the phone dictionary. The time taken to type the message was measured using a stop watch and each subject used both their own phone and a control phone. 15 teenagers and 15 people over the age of 30 took part in the study.

Four variables (plus an ID) were measured:

- ID – An identifier
- Age – The person's age (in years)
- AgeGroup – their age group; one of Teens or Over30
- Time – The time (in seconds) to SMS the above message
- Phone – the type of phone used; either their Own phone or a Control phone.

These data consist of both independent data (Teens vs Over30s) and matched pairs data (Own vs Control phones).

Each of you have your own data file to analyse. The files `student_id.txt` contain the data collected as described above, where `student_id` is your own UL student ID number, e.g. if your ID number is 1234567, the file `1234567.txt` is your data file to be used for this project.

Download your own file to your working folder from

<http://jkcray.maths.ul.ie/ms4024/R-Files/RFilesProj3>

and read the data into R using

```
sms <- read.table("student_id.txt", header=T)  
  
names(sms)  
[1] "ID"          "Age"         "AgeGroup"    "Time"        "Phone"
```

Marks will be awarded for the R code used to answer Tasks 1-3 below and also the correct interpretations of the results obtained for your particular data set.

Task 1

Initially compare the times taken by the Teens group with the times taken by the Over30s group.

- (a) Perform an exploratory data analysis of these data by:
 - (i) calculating the mean and standard deviation for the times taken by Teens and the times taken by Over30s;
 - (ii) constructing side-by-side boxplots of the times for each group.
- (b) From these initial results, do you think there is a difference between the average time taken by teenagers to send the message and the average time taken by over 30s to send the message? Justify your answer.
- (c) Perform a hypothesis test to determine if the variances in both groups are equal. Use $\alpha = 0.05$.

- (d) Based on the results of part (c), perform a suitable hypothesis test to determine if there is a significant difference between the average time taken by teenagers to send the message and the average time taken by people over 30 to send the message. Use $\alpha = 0.05$.
- (e) Do the results of the hypothesis test agree with your initial conclusion?

Task 2

Now compare the time taken by each subject when using their own phone versus the time taken by each subject when using the control phone.

- (a) Extract the times taken by subjects using their own phone and store in a vector called `Own`.
- (b) Extract the times taken by subjects using the control phone and store in a vector called `Control`.
- (c) Calculate the vector of differences
`diffs <- Own - Control.`
- (d) Create a histogram of `diffs`. From this plot, do you think the time taken by each individual using their own phone differs significantly from the time taken using the control phone?

- (e) Construct a 95% confidence interval for the differences and use this to test whether on average, the time taken by each individual using their own phone differs significantly from the time taken using the control phone.

Task 3

The following questions refer to regression analysis:

- (a) Extract all of the data for subjects using their own phone and store in a data frame called newdat.
- (b) Create a scatterplot of newdat\$Age (x) versus newdat\$Time (y) and describe the relationship between them. Label the axes appropriately and give the plot a suitable title.
- (c) Is there a significant correlation between Age and Time?
- (d) Fit a SLR model called my.model to these data using the lm command and add the fitted line to your scatterplot.

- (e) Extract the estimated coefficients for the intercept and slope from `my.model` and write down the fitted regression line.
- (f) Determine if the assumptions of normality and constant variance hold by creating appropriate plots of the residuals.
Justify your answer.

You are asked to submit a single Adobe PDF file prepared using LATEX containing a summary for each task.

For Task 1, include the following:

- a listing of your R commands for Task 1;
- the result(s) printed in the R console for part (a)(i);
- the result(s) printed in the R console for part (c);
- the result(s) printed in the R console for part (d);
- your answers/conclusions for parts (b), (c), (d) and (e);
- the plot created by your code.

For Task 2, include the following:

- a listing of your R commands for Task 2;
- the result printed in the R console for part (e);
- your answers/conclusions for parts (d) and (e);
- the plot created by your code.

For Task 3, include the following:

- a listing of your R commands for Task 3;
- the result printed in the R console for parts (c) and (e);
- your answers/conclusions for parts (b), (c) and (f);
- the plots created by your code.

- The LaTeX file downloadable from

<http://jkcray.maths.ul.ie/ms4024/LaTeX-Files/HowToIncludeRCode.tex>
has been updated to include a separate section for each Task.
Use this file to submit your answers.

- There are 5 items required when uploading your work:
 - ▶ your R workspace;
 - ▶ the text file containing your R commands;
 - ▶ the LaTeX document you have created;
 - ▶ the PDF file created;
 - ▶ the 4 plots created.