Practical R for Epidemiologists

Mark Myatt

2018-04-19

Contents

Chapter 1

Introduction

1.1 Introducing R

R is a system for data manipulation, calculation, and graphics. It provides:

- Facilities for data handling and storage
- A large collection of tools for data analysis
- Graphical facilities for data analysis and display
- A simple but powerful programming language

R is often described as an environment for working with data. This is in contrast to a *package* which is a collection of very specific tools. R is not strictly a statistics system but a system that provides many classical and modern statistical procedures as part of a broader data-analysis tool. This is an important difference between R and other statistical systems. In R a statistical analysis is usually performed as a series of steps with intermediate results being stored in objects. Systems such as SPSS and SAS provide copious output from (e.g.) a regression analysis whereas R will give minimal output and store the results of a fit for subsequent interrogation or use with other R functions. This means that R can be tailored to produce exactly the analysis and results that you want rather than produce an analysis designed to fit all situations.

R is a language based product. This means that you interact with R by typing commands such as:

```
table(SEX, LIFE)
```

rather than by using menus, dialog boxes, selection lists, and buttons. This may seem to be a drawback but it means that the system is considerably more flexible than one that relies on menus, buttons, and boxes. It also means that every stage of your data management and analysis can be recorded and edited and re-run at a later date. It also provides an audit trail for quality control purposes.

R is available under UNIX (including Linux), the Macintosh operating system OS X, and Microsoft Windows. The method used for starting R will vary from system to system. On UNIX systems you may need to issue the R command in a terminal session or click on an icon or menu option if your system has a windowing system. On Macintosh systems R will be available as an application but can also be run in a terminal session. On Microsoft Windows systems there will usually be an icon on the Start menu or the desktop.

R is an open source system and is available under the *GNU general public license* (GPL) which means that it is available for free but that there are some restrictions on how you are allowed to distribute the system and how you may charge for bespoke data analysis solutions written using the R system. Details of the general public license are available from http://www.gnu.org/copyleft/gpl.html.

R is available for download from http://www.r-project.org/.

This is also the best place to get extension packages and documentation. You may also subscribe to the R mailing lists from this site. R is supported through mailing lists. The level of support is at least as good as for commercial packages. It is typical to have queries answered in a matter of a few hours.

Even though R is a free package it is more powerful than most commercial packages. Many of the modern procedures found in commercial packages were first developed and tested using R or **S-Plus** (the commercial equivalent of R).

When you start R it will issue a prompt when it expects user input. The default prompt is:

```
>
```

This is where you type commands that call functions that instruct R to (e.g.) read a data file, recode data, produce a table, or fit a regression. For example:

```
> table(SEX, LIFE)
```

If a command you type is not complete then the prompt will change to:

```
+
```

on subsequent lines until the command is complete:

```
> table(
+ SEX, LIFE +)
```

The > and + prompts are not shown in the example commands in the rest of this material.

The example commands in this material are often broken into shorter lines and indented for ease of understanding. The code still works as lines are split in places where R knows that a line is not complete. For example:

```
table(SEX,
      LIFE)
```

could be entered on a single line as:

```
table(SEX, LIFE)
```

In this example R knows that the command is not complete until the brackets are closed. The following example could also be written on one line:

```
salex.lreg.coeffs <-
  coef(summary(salex.lreg))
```

In this case R knows that the <- operator at the end of the first line needs further input.

R maintains a history of previous commands. These can be recalled and edited using the up and down arrow keys.

Output that has scrolled off the top of the output / command window can be recalled using the window or terminal scroll bars.

Output can be saved using the `sink()` function with a file name: `sink("results.out")` to start recording output. Use the `sink()` function without a file name to stop recording output: `sink()`

You can also use clipboard functions such as copy and paste to (e.g.) copy and then paste selected chunks of output into an editor or word processor running alongside R.

All the sample data files used in the exercises in this manual are space delimited text files using the general format:

```
ID AGE IQ
1 39 94
2 41 89
```

```
3 42 83
4 30 99
5 35 94
6 44 90
7 31 94
8 39 87
```

R has facilities for working with files in different formats including (through the use of extension packages) **ODBC** (open database connectivity) and **SQL** data sources, **EpiInfo**, **EpiData**, **Minitab**, **SPSS**, **SAS**, **S-Plus**, and **Stata** format files.

1.2 Retrieving data

All of the exercises in this manual assume that the necessary data files are located in the current working directory. All of the data files that you require to follow this material are in a ZIP archive that can be downloaded from:

http://www.brixtonhealth.com/prfe/prfe.zip

A command such as:

```
read.table("data/fem.dat", header = TRUE)
```

retrieves the data stored in the file named `fem.dat` which is stored in the `data` folder.

To retrieve data that is stored in files outside a different directory you need to specify the full path to the file. For example:

```
read.table("~/example/fem.dat", header = TRUE)
```

will retrieve the data stored in the file named `fem.dat` stored in the `example` directory under the user's home directory on UNIX, Linux, and OS X systems.

R follows many UNIX operating and naming conventions including the use of the backslash (\) character to specify special characters in strings (e.g. using `\n` to specify a new line in printed output). Windows uses the backslash (\) character to separate directory and file names in paths. This means that Windows users need to escape any backslashes in file paths using an additional backslash character. For example:

```
read.table("c:\\example\\fem.dat", header = TRUE)
```

will retrieve the data that is stored in the file named `fem.dat` which is stored in the `example` directory off the root directory of the `C:` drive. The Windows version of R also allows you to specify UNIX-style path names (i.e. using the forward slash (`/`) character as a separator in file paths). For example:

```
read.table("c:/example/fem.dat", header = TRUE)
```

Path names may include shortcut characters such as:

`.`	The current working directory
`..`	Up one level in the directory tree
`~`	The user's home directory (on UNIX-based systems)

R also allows you to retrieve files from any location that may be represented by a standard `uniform resource locator (URL)` string. For example:

```
read.table("file://~/example/fem.dat", header = TRUE)
```

will retrieve the data stored in the file named `fem.dat` stored in the `example` directory under the users home

directory on UNIX-based systems.

All of the data files used in this section are stored in the `/data` directory of this guide's GitLab repository (https://git.validmeasures.org/datahub/datahubguide/tree/master/data). This means, for example, that you can use the `read.table()` function specifying

"https://git.validmeasures.org/datahub/datahubguide/tree/master/data/fem.dat"

as the `URL` to retrieve the data that is stored in the file named `fem.dat` which is stored in the `/data` directory of this guide's GitLab repository.

Chapter 2

Getting acquainted with R

In this exercise we will use R to read a dataset and produce some descriptive statistics, produce some charts, and perform some simple statistical inference. The aim of the exercise is for you to become familiar with R and some basic R functions and objects.

The first thing we will do, after starting R, is issue a command to retrieve an example dataset:

```
fem <- read.table("fem.dat", header = TRUE)
```

This command illustrates some key things about the way R works.

We are instructing R to assign (using the <- operator) the output of the `read.table()` function to an object called `fem`.

The `fem` object will contain the data held in the file `fem.dat` as an R data.frame object:

```
class(fem)
```

```
## [1] "data.frame"
```

You can inspect the contents of the `fem` data.frame (or any other R object) just by typing its name:

```
fem
```

```
##    ID AGE IQ ANX DEP SLP SEX LIFE    WT
## 1   1  39 94   2   2   2   1    1  2.23
## 2   2  41 89   2   2   2   1    1  1.00
## 3   3  42 83   3   3   2   1    1  1.82
## 4   4  30 99   2   2   2   1    1 -1.18
## 5   5  35 94   2   1   1   1    2 -0.14
## 6   6  44 90  NA   1   2   2    2  0.41
```

Note that the `fem` object is built from other objects. These are the named vectors (columns) in the dataset:

```
names(fem)
```

```
## [1] "ID"   "AGE"  "IQ"   "ANX"  "DEP"  "SLP"  "SEX"  "LIFE" "WT"
```

The [1] displayed before the column names refers to the numbered position of the first name in the output. These positions are known as indexes and can be used to refer to individual items. For example:

```
names(fem)[1]
```

```
## [1] "ID"
```

```
names(fem)[8]
```

```
## [1] "LIFE"
```

```
names(fem)[2:4]
```

```
## [1] "AGE" "IQ"  "ANX"
```

The data consist of 118 records:

```
nrow(fem)
```

```
## [1] 118
```

each with nine variables:

```
ncol(fem)
```

```
## [1] 9
```

for female psychiatric patients.

The columns in the dataset are:

ID	Patient ID
AGE	Age in years
IQ	IQ score
ANX	Anxiety (1=none, 2=mild, 3=moderate, 4=severe)
DEP	Depression (1=none, 2=mild, 3=moderate or severe)
SLP	Sleeping normally (1=yes, 2=no)
SEX	Lost interest in sex (1=yes, 2=no)
LIFE	Considered suicide (1=yes, 2=no)
WT	Weight change (kg) in previous 6 months

The first ten records of the `fem` data.frame are:

```
##    ID AGE  IQ ANX DEP SLP SEX LIFE    WT
## 1   1  39  94   2   2   2   1    1  2.23
## 2   2  41  89   2   2   2   1    1  1.00
## 3   3  42  83   3   3   2   1    1  1.82
## 4   4  30  99   2   2   2   1    1 -1.18
## 5   5  35  94   2   1   1   1    2 -0.14
## 6   6  44  90  NA   1   2   2    2  0.41
## 7   7  31  94   2   2  NA   1    1 -0.68
## 8   8  39  87   3   2   2   1    2  1.59
## 9   9  35 -99   3   2   2   1    1 -0.55
## 10 10  33  92   2   2   2   1    1  0.36
```

You may check this by asking `R` to display all columns of the first ten records in the `fem` data.frame:

```
fem[1:10, ]
```

```
##    ID AGE  IQ ANX DEP SLP SEX LIFE    WT
## 1   1  39  94   2   2   2   1    1  2.23
## 2   2  41  89   2   2   2   1    1  1.00
## 3   3  42  83   3   3   2   1    1  1.82
## 4   4  30  99   2   2   2   1    1 -1.18
## 5   5  35  94   2   1   1   1    2 -0.14
## 6   6  44  90  NA   1   2   2    2  0.41
```

```
## 7   7  31  94   2   2  NA   1    1 -0.68
## 8   8  39  87   3   2   2   1    2  1.59
## 9   9  35 -99   3   2   2   1    1 -0.55
## 10 10  33  92   2   2   2   1    1  0.36
```

The space after the comma is optional. You can think of it as a *placeholder* for where you would specify the indexes for columns you wanted to display. For example:

```
fem[1:10,2:4]
```

displays the first ten rows and the second, third and fourth columns of the `fem` data.frame:

```
##    AGE  IQ ANX
## 1   39  94   2
## 2   41  89   2
## 3   42  83   3
## 4   30  99   2
## 5   35  94   2
## 6   44  90  NA
## 7   31  94   2
## 8   39  87   3
## 9   35 -99   3
## 10  33  92   2
```

`NA` is a special value meaning *not available* or *missing*.

You can access the contents of a single column by name:

```
fem$IQ
```

```
##   [1]  94  89  83  99  94  90  94  87 -99  92  92  94  91  86  90 -99  91
##  [18]  82  86  88  97  96  95  87 103 -99  91  87  91  89  92  84  94  92
##  [35]  96  96  86  92 102  82  92  90  92  88  98  93  90  91 -99  92  92
##  [52]  91  91  86  95  91  96 100  99  89  89  98  98 103  91  91  94  91
##  [69]  85  92  96  90  87  95  95  87  95  88  94 -99 -99  87  92  86  93
##  [86]  92 106  93  95  95  92  98  92  88  85  92  84  92  91  86  92  89
## [103] -99  96  97  92  92  98  91  91  89  94  90  96  87  86  89 -99
```

```
fem$IQ[1:10]
```

```
##  [1]  94  89  83  99  94  90  94  87 -99  92
```

The `$` sign is used to separate the name of the data.frame and the name of the column of interest. Note that `R` is case-sensitive so that `IQ` and `iq` are ***not*** the same.

You can also access rows, columns, and individual cells by specifying row and column positions. For example, the `IQ` column is the third column in the `fem` data.frame:

```
fem[ ,3]
```

```
##   [1]  94  89  83  99  94  90  94  87 -99  92  92  94  91  86  90 -99  91
##  [18]  82  86  88  97  96  95  87 103 -99  91  87  91  89  92  84  94  92
##  [35]  96  96  86  92 102  82  92  90  92  88  98  93  90  91 -99  92  92
##  [52]  91  91  86  95  91  96 100  99  89  89  98  98 103  91  91  94  91
##  [69]  85  92  96  90  87  95  95  87  95  88  94 -99 -99  87  92  86  93
##  [86]  92 106  93  95  95  92  98  92  88  85  92  84  92  91  86  92  89
## [103] -99  96  97  92  92  98  91  91  89  94  90  96  87  86  89 -99
```

```
fem[9, ]
```

```
##   ID AGE  IQ ANX DEP SLP SEX LIFE    WT
```

```
## 9  9  35 -99   3   2   2   1    1 -0.55
```

```
fem[9,3]
```

```
## [1] -99
```

There are missing values in the `IQ` column which are all coded as **-99**. Before proceeding we must set these to the special `NA` value:

```
fem$IQ[fem$IQ == -99] <- NA
```

The term inside the square brackets is also an index. This type of index is used to refer to subsets of data held in an object that meet a particular condition. In this case we are instructing `R` to set the contents of the `IQ` variable to `NA` if the contents of the `IQ` variable is **-99**.

Check that this has worked:

```
fem$IQ
```

```
##   [1]  94  89  83  99  94  90  94  87  NA  92  92  94  91  86  90  NA  91
##  [18]  82  86  88  97  96  95  87 103  NA  91  87  91  89  92  84  94  92
##  [35]  96  96  86  92 102  82  92  90  92  88  98  93  90  91  NA  92  92
##  [52]  91  91  86  95  91  96 100  99  89  89  98  98 103  91  91  94  91
##  [69]  85  92  96  90  87  95  95  87  95  88  94  NA  NA  87  92  86  93
##  [86]  92 106  93  95  95  92  98  92  88  85  92  84  92  91  86  92  89
## [103]  NA  96  97  92  92  98  91  91  89  94  90  96  87  86  89  NA
```

We can now compare the groups who have and have not considered suicide. For example:

```
by(fem$IQ, fem$LIFE, summary)
```

Look at the help for the `by()` function:

```
help(by)
```

Note that you may use `?by` as a shortcut for `help(by)`.

The `by()` function applies another function (in this case the `summary()` function) to a column in a data.frame (in this case `fem$IQ`) split by the value of another variable (in this case `fem$LIFE`).

It can be tedious to always have to specify a data.frame each time we want to use a particular variable. We can fix this problem by 'attaching' the data.frame:

```
attach(fem)
```

```
## The following objects are masked from fem (pos = 3):
##
##     AGE, ANX, DEP, ID, IQ, LIFE, SEX, SLP, WT
```

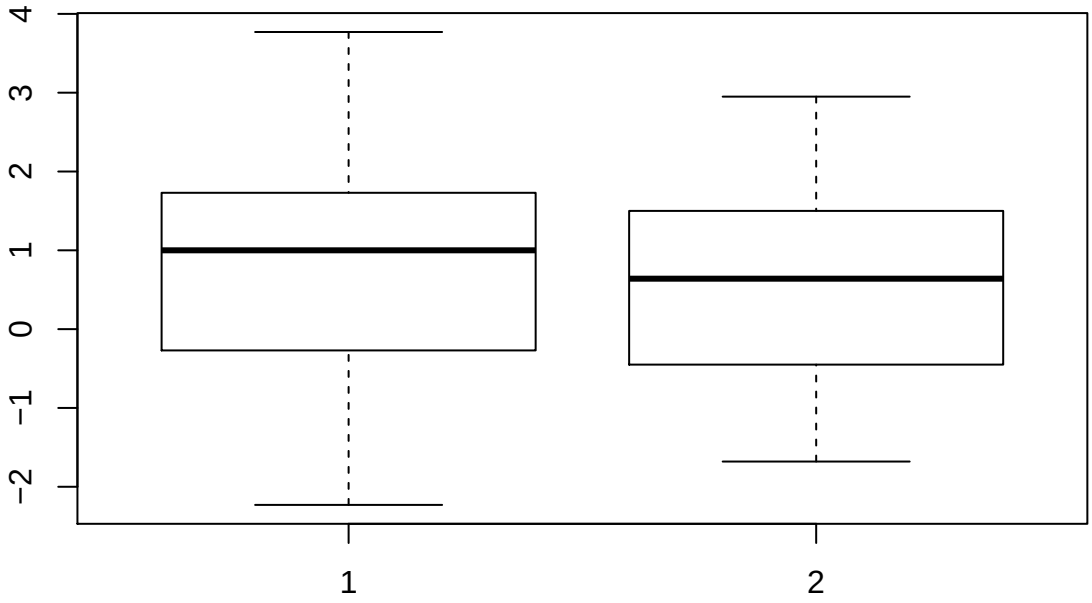
We can now refer to the columns in the `fem` data.frame without having to specify the name of the data.frame. This time we will produce summary statistics for `WT` by `LIFE`:

```
by(WT, LIFE, summary)
```

```
## LIFE: 1
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -2.2300 -0.2700  1.0000  0.7867  1.7300  3.7700       4
## ------------------------------------------------------------
## LIFE: 2
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -1.6800 -0.4500  0.6400  0.6404  1.5000  2.9500       7
```

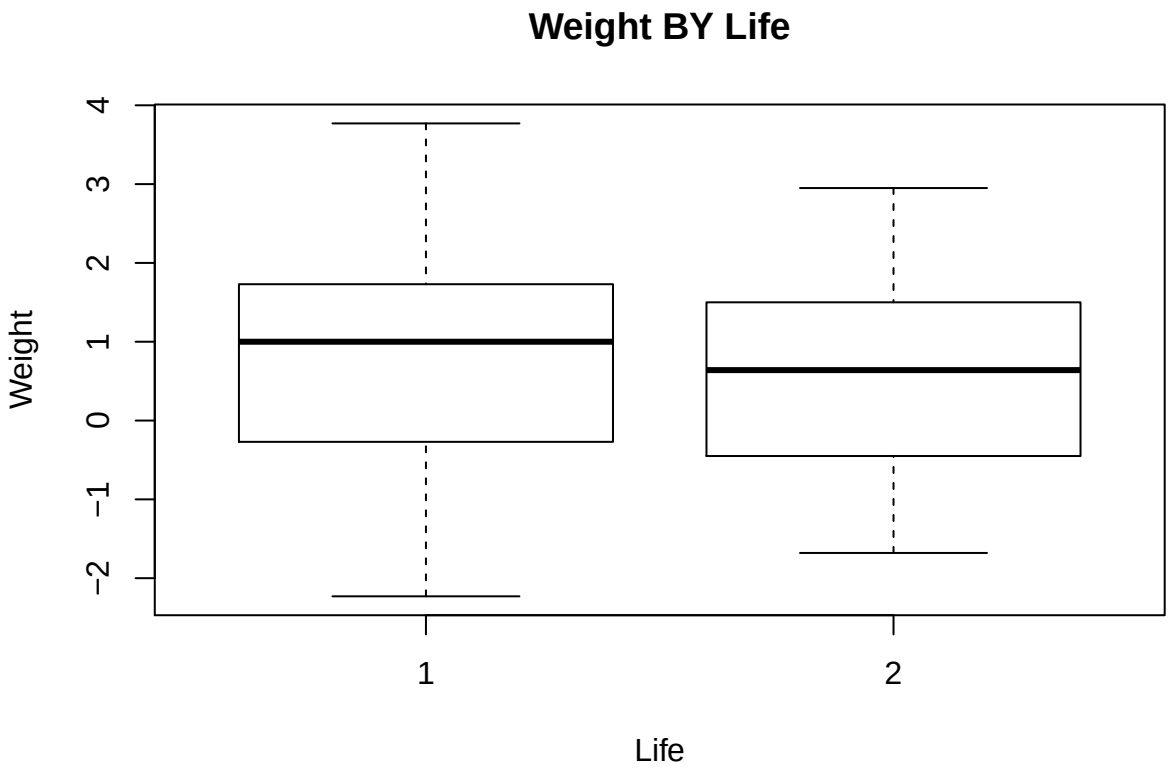
We can view the same data as a box and whisker plot:

```
boxplot(WT ~ LIFE)
```



We can add axis labels and a title to the graph:

```
boxplot(WT ~ LIFE,
        xlab = "Life",
        ylab = "Weight",
        main = "Weight BY Life")
```



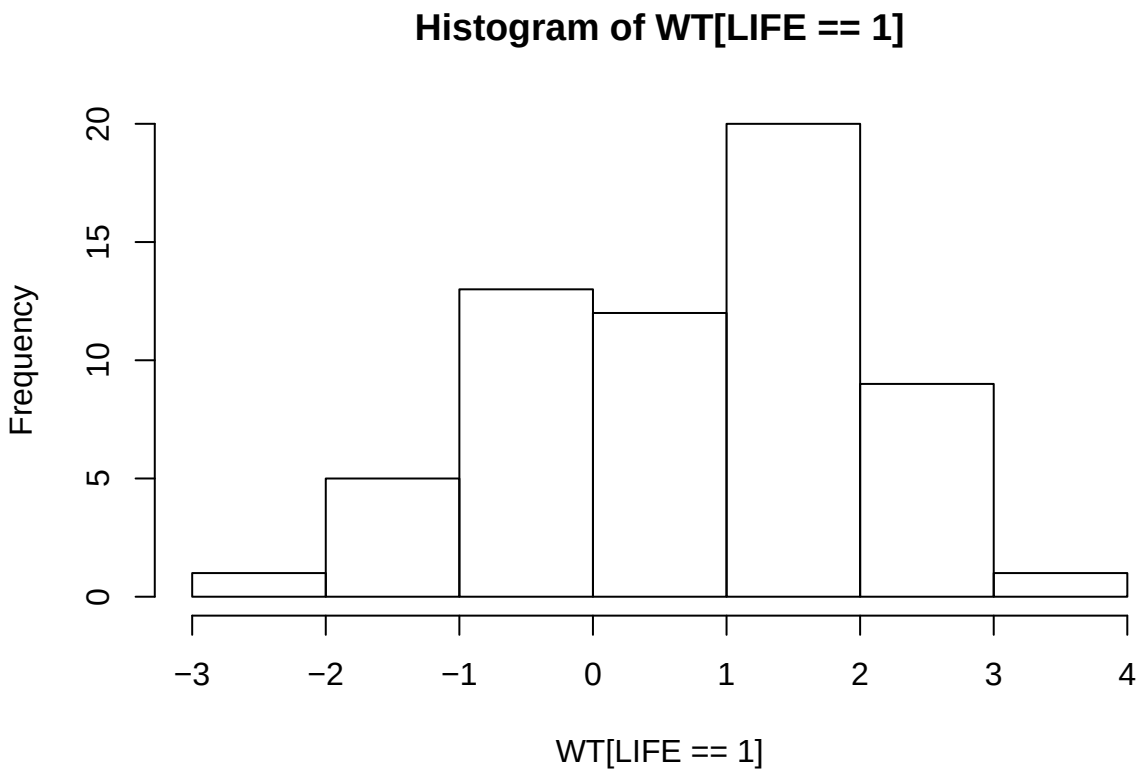
A more descriptive title might be “Weight Change BY Considered Suicide”.

The groups do not seem to differ much in their medians and the distributions appear to be reasonably

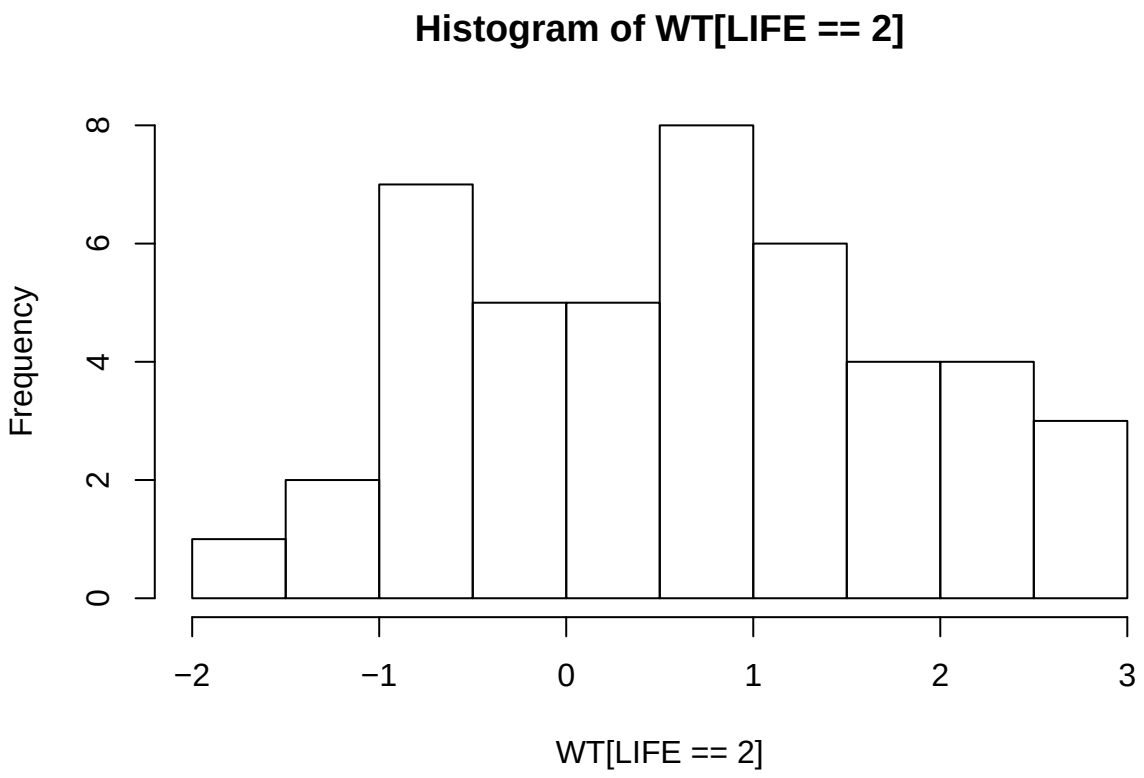
symmetrical about their medians with a similar spread of values.

We can look at the distribution as histograms:

```
hist(WT[LIFE == 1])
```

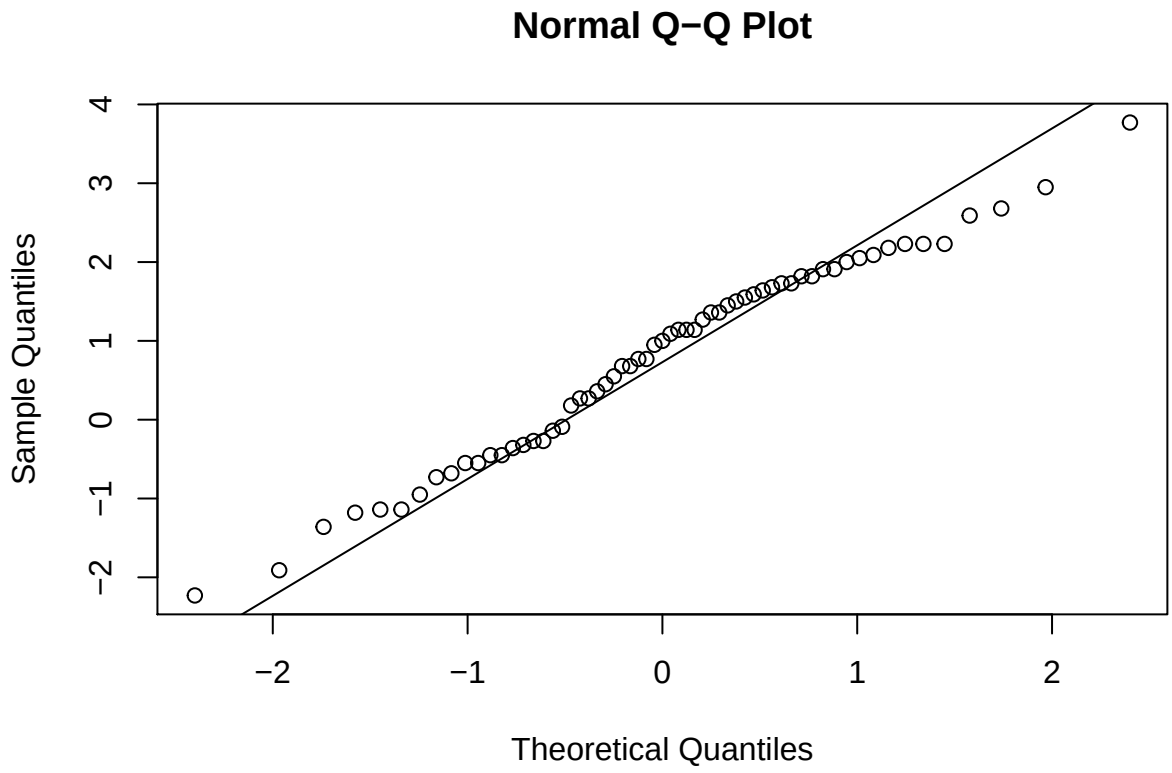



```
hist(WT[LIFE == 2])
```

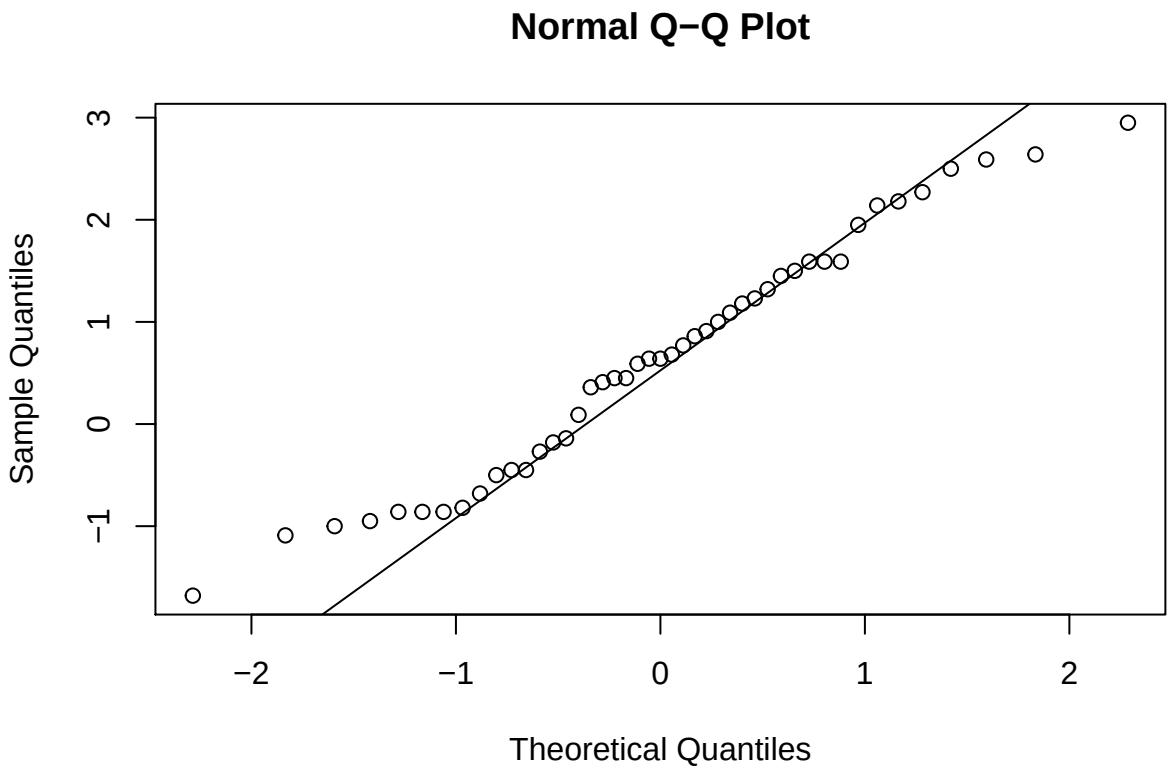


and check the assumption of normality using quantile-quantile plots:

```
qqnorm(WT[LIFE == 1])
qqline(WT[LIFE == 1])
```



```
qqnorm(WT[LIFE == 2])
qqline(WT[LIFE == 2])
```



or by using a formal test:

```
shapiro.test(WT[LIFE == 1])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  WT[LIFE == 1]
## W = 0.98038, p-value = 0.4336
```

```
shapiro.test(WT[LIFE == 2])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  WT[LIFE == 2]
## W = 0.97155, p-value = 0.3292
```

Remember that we can use the `by()` function to apply a function to a data.frame, including statistical functions such as `shapiro.test()`:

```
by(WT, LIFE, shapiro.test)
```

```
## LIFE: 1
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.98038, p-value = 0.4336
```

```
##
## ------------------------------------------------------------
## LIFE: 2
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.97155, p-value = 0.3292
```

We can also test whether the variances differ significantly using *Bartlett's test* for the homogeneity of variances:

```
bartlett.test(WT, LIFE)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  WT and LIFE
## Bartlett's K-squared = 0.32408, df = 1, p-value = 0.5692
```

There is no significant difference between the two variances.

Many functions in `R` have a *formula interface* that may be used to specify multiple variables and the relations between multiple variables. We could have used the formula interface with the `bartlett.test()` function:

```
bartlett.test(WT ~ LIFE)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  WT by LIFE
## Bartlett's K-squared = 0.32408, df = 1, p-value = 0.5692
```

Having checked the normality and homogeneity of variance assumptions we can proceed to carry out a `t-test`:

```
t.test(WT ~ LIFE, var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  WT by LIFE
## t = 0.59869, df = 104, p-value = 0.5507
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3382365  0.6307902
## sample estimates:
## mean in group 1 mean in group 2
##       0.7867213       0.6404444
```

There is no evidence that the two groups differ in weight change in the previous six months.

We could still have performed a `t-test` if the variances were not homogenous by setting the **var.equal** parameter of the `t.test()` function to **FALSE**:

```
t.test(WT ~ LIFE, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
```

```
## data:  WT by LIFE
## t = 0.60608, df = 98.866, p-value = 0.5459
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3326225  0.6251763
## sample estimates:
## mean in group 1 mean in group 2
##       0.7867213       0.6404444
```

or performed a non-parametric test:

```
wilcox.test(WT ~ LIFE)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  WT by LIFE
## W = 1488, p-value = 0.4622
## alternative hypothesis: true location shift is not equal to 0
```

An alternative, and more general, non-parametric test is:

```
kruskal.test(WT ~ LIFE)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  WT by LIFE
## Kruskal-Wallis chi-squared = 0.54521, df = 1, p-value = 0.4603
```