Chapter 4

Loading data

4.1 Introduction

The main purpose of R is analysis of data. Therefore, it is vitally important to be able to load data. Fortunately, R is very flexible about importing data. Certain functions can be used to read and write data stored in a variety of text formats, as well as data saved by statistical packages such as Minitab, S, SAS, SPSS, Stata, Systat, and for reading and writing `.dbf` (dBase) files. Data can also be entered directly into R.

4.2 Entering data directly

Data can be entered directly into R using the function `c()` (for concatenate):

```
> attendance <- c(9, 10, 15, 13, 12)
> attendance

[1]  9 10 15 13 12
```

The function `scan()` can also be used to read the data line-by-line from the console (input is terminated by entering a *blank line*):

```
> attendance <- scan()
9
10
15
13
12

Read 5 items
> attendance

[1]  9 10 15 13 12
```

4.3 Loading text data files

Most users load their data into R from text files. Consider this (artificial) data set, named `datafile1.dat` say, where tabs separate the data within each row:

```
Gender   Age   Height   Income
Male     34    173      34556
Male     45    184      67780
Female   NA    175      56449
Male     24             22089
Female   45    169      34402
Female   21    172      26709
Female   26    166      45221
Male     NA    189      56223
```

The categorical variable `Gender` takes the value `Male` or `Female`. Age, Height and Income are quantitative. There are three missing values, two coded as `NA` and one simply missing altogether. The first line also has a header row containing the variable names.

The data could be loaded as follows, directly from the web page where I have this file:

```
> read.table("http://www.sci.usq.edu.au/staff/dunn/Datasets/datafile1.dat",
+     header = TRUE, sep = "\t")

  Gender Age Height Income
1   Male  34    173  34556
2   Male  45    184  67780
3 Female  NA    175  56449
4   Male  24     NA  22089
5 Female  45    169  34402
6 Female  21    172  26709
7 Female  26    166  45221
8   Male  NA    189  56223
```

The input `\sep="\t"` means the columns are separated with tab characters. If the data file is 'nice ' (no missing values, no textstrings, etc.), you can probably omit the `sep` input. Note that R can load textual data as well as numerical data. Also, note that the missing values posed no problem.

If the columns are not separated with tabs, other separators can be used.

Almost always, when data is loaded it is allocated to an object name:

```
> df <- read.table("http://www.sci.usq.edu.au/staff/dunn/Datasets/datafile1.dat",
+     header = TRUE, sep = "\t")
```

Soon we will look at the structure of these types of objects.

4.4 More on loading data

- R includes many data sets by default (for example, for use in the R help). A list of these is shown by typing

  ```
  > data()
  ```

  These data can be loaded as follows:

  ```
  > data(USArrests)
  ```

  In most instances, however, data will need to be loaded from a data file.

- If the first row of your data file does not have variable names, omit the argument `header=TRUE`. R then names the variables `V1`, `V2`, and so on, by default.

- If data are separated by commas (for example, `.csv` files stands for comma separated variables), you can used `read.table` with `sep=","` or `read.csv` which is specifically designed for comma-separated data files.

- Almost any separator can be specified using `sep=`; see `?read.table`.

- If you omit the `sep` argument, R thinks variable are separated by white space (spaces, tabs, new lines or carriage returns). This generally works well, but sometimes you need to explicitly specify the separator.

- Data are often missing. Sometime, many different codes are used to denote this in one data file. For example, `na.strings=c("NA","99999","-1")` means that `NA`, `99999` and `-1` all denote missing values.

- R copes if missing values are simply absent and nothing appears; for example, if a comma-separated file contained this line:
  `6, -1, "Male",, 173,12,,0`
  R would see two missing values.

- Often, text is quoted in quotation marks. By default, R looks for double quote marks: `"`. Other quote characters can be defined also. The default is `quote="\""`. (Note the quotation character is entered between double quotes (so we could also say `quote="!"` if the quotation marks are `!` wich is unlikely); so to indicate the quotation character itself is a double quote, the double quote must be *escaped*.)

- There are many other options; see `?read.table`.

- For more powerful handling of data importing, see `?scan`.

- See `read.fwf` specifically designed for reading data in `fixed width format`.

- Type `library(help=foreign)` to learn about how R reads data from other file formats such as SPSS, SAS and Minitab.

- As already seen, the data file can be a `url` so that data can be loaded from the web.

- Unless told otherwise, R loads data from the current working directory/folder, which can be obtained from

  ```
  > getwd()
  ```

  ```
  [1] "/home/mcsci/dunn/pkd/Admin/Webpages/Homepage/LearnR/Docs"
  ```

  This directory/folder can be *set* as follows in Linux:

  ```
  setwd("~/datasets/thiscourse/datatfiles")
  ```

  In Windows, use one of:

  ```
  setwd("c:/datasets/thiscourse/datasets")
  ```

  or

  ```
  set.wd("c:\\datasets\\thiscourse\\datasets")
  ```

  or use the menu to set the current working directory.

- To specify the location of a file directly, use, for example,
  `read.table("~/datasets/thiscourse/tutorial1.dat")`.
  In Windows, use
  `read.table("c:/datasets/thiscourse/tutorial1.dat")`
  or
  `read.table("c:\\datasets\\thiscourse\\tutorial1.dat")`

4.5 Data frames

Data files may contain different types of variables, such as

- Quantitative variables (such as heights);
- Categorical variables (such as gender);
- Text (or string) variables (such as people's names);

and many others. R also handles missing values well.

When data is loaded into R, it is stored as an object called a *data frame*. A data frame is like a matrix, but it can contain numerical as well as textual variables. For example,

```
> dummy.data <- read.table("http://www.sci.usq.edu.au/staff/dunn/Datasets/datafile1.dat",
+     header = TRUE, sep = "\t")
> dummy.data

  Gender Age Height Income
1   Male  34    173  34556
2   Male  45    184  67780
3 Female  NA    175  56449
4   Male  24     NA  22089
5 Female  45    169  34402
6 Female  21    172  26709
7 Female  26    166  45221
8   Male  NA    189  56223
```

So we have a data frame called `dummy.data`. How do we just access *one* variable in that data frame? As follows[1]:

```
> dummy.data$Gender

[1] Male   Male   Female Male   Female Female Female Male
Levels: Female Male

> summary(dummy.data$Gender)

Female   Male
     4      4

> summary(dummy.data$Income)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  22090   32480   39890   42930   56280   67780
```

[1]Incidentally, this example demonstrates that R is *object oriented*: the `summary` command produces different output depending of what type of input it is asked to summarize.

In this way, you can have several data sets ("data frames") loaded at once. Try typing `Gender`, and see what happens: R doesn't know any variable by the name `Gender`, only `dummy.data$Gender`. It gets pretty annoying after a while typing `dummy.data$Gender` all the time. . . so consider this:

```
> attach(dummy.data)
> Gender

[1] Male   Male   Female Male   Female Female Female Male
Levels: Female Male

> Income

[1] 34556 67780 56449 22089 34402 26709 45221 56223
```

The command `attach` makes the variables in the specified data frame available without having to go through the cumbersome `data.frame$variable.name` process.

When you have finished with a data frame, remember to use

```
> detach(dummy.data)
```

If you have two data frames open that share variable names, anything could happen. . .

4.6 Examining data

4.6.1 Initial examination of the data frame

The data file `jelly-R.dat` is available for loading in to R, and comes from Hand et al. [1]. Proceed as follows: First ensure you are in the correct directory where you saved the data file, load it from a URL (you'll have to be connected to the internet!), or give the full path:

```
> jf <- read.table("http://www.sci.usq.edu.au/staff/dunn/Datasets/Books/Hand/Hand-R/jelly-
+     header = TRUE)
> attach(jf)
> names(jf)

[1] "Breadth" "Length"  "Site"
```

Once data is loaded, the first task is to quickly examine the data. The following commands are useful for quick looks, especially to ensure the data loaded correctly:

```
> head(jf)

  Breadth Length Site
1     6.5    8.0    1
2     6.0    9.0    1
3     6.5    9.0    1
4     7.0    9.0    1
5     8.0    9.5    1
6     7.0   10.0    1

> tail(jf)

   Breadth Length Site
41      19     20    2
42      15     21    2
43      16     21    2
44      21     21    2
45      19     22    2
46      20     22    2

> summary(jf)

    Breadth          Length            Site      
 Min.   : 6.00   Min.   : 8.00   Min.   :1.000  
 1st Qu.:10.25   1st Qu.:13.00   1st Qu.:1.000  
 Median :15.00   Median :16.25   Median :2.000  
 Mean   :13.34   Mean   :15.80   Mean   :1.522  
 3rd Qu.:16.00   3rd Qu.:19.00   3rd Qu.:2.000  
 Max.   :21.00   Max.   :22.00   Max.   :2.000  
```

Notice that summary produces different information for different sort of variables: again, this is the object-oriented aspect of R.

4.6.2 Initial examination of the variables

All the usual statistical summaries of data are available in R; for example, with quantitative variables:

- measures of centre, like the mean and median:

  ```
  > mean(Length)

  [1] 15.80435

  > median(Length)

  [1] 16.25
  ```

- measures of spread, like the standard deviation, variance and interquartile range:

  ```
  > sd(Length)

  [1] 4.167971

  > var(Length)

  [1] 17.37198

  > IQR(Length)

  [1] 6
  ```

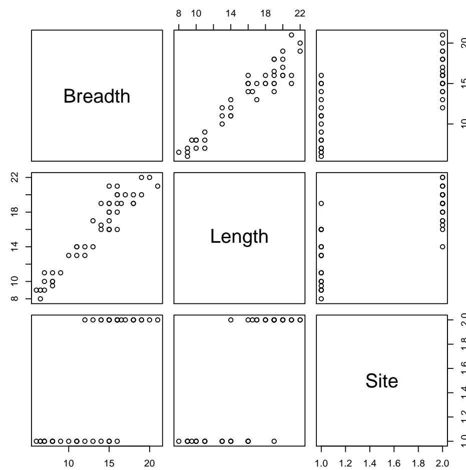
A simpler overview of a variable is provided using `summary`, much like we saw above:

```
> summary(Length)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   8.00   13.00   16.25   15.80   19.00   22.00

> summary(Site)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   2.000   1.522   2.000   2.000
```

4.6.3 Initial plotting of the data

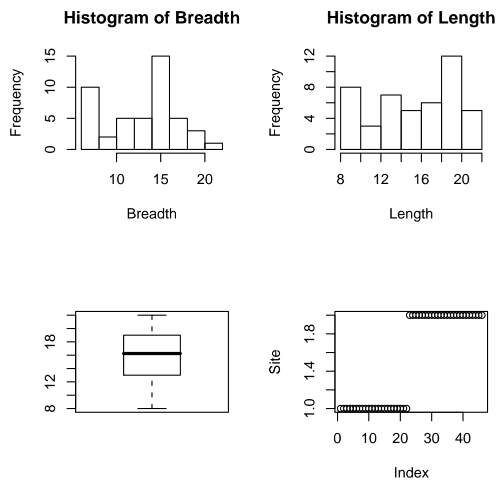
We should also do some plots of the data:

```
> plot(jf)
```



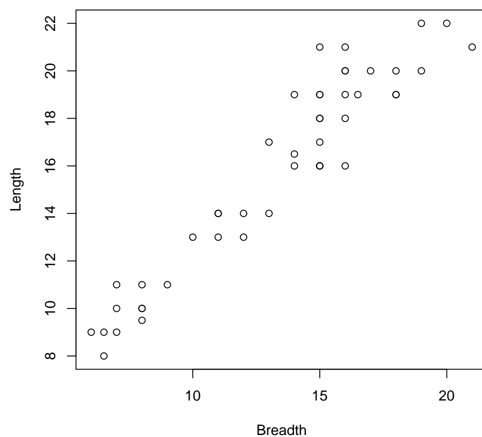
Even better, we can examine each variable separately[2]:

```
> par(mfrow = c(2, 2))
> hist(Breadth)
> hist(Length)
> boxplot(Length)
> plot(Site)
> par(mfrow = c(1, 1))
```



Relationships may also be examined:

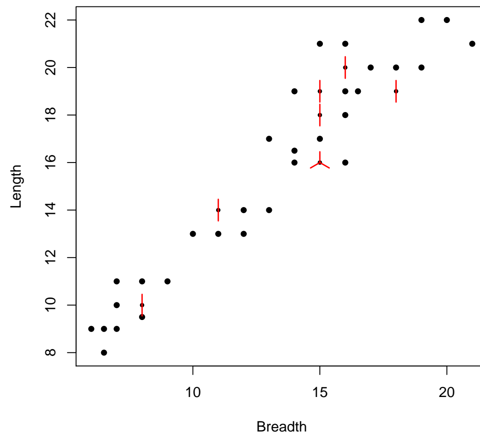
```
> plot(Breadth, Length)
```



Some markers represent more than one observation; here is an alternative plot:

[2]The command `par( mfrow=c(r,c) )` splits the graphics windows into `r` rows and `c` columns. Also see the help (or run the `example`) for `layout` and `split.screen`.

```
> sunflowerplot(Breadth, Length)
```



In either case, there appears to be a linear relationship between Length and Breadth. Perhaps some kind of linear model is appropriate.

We can also be fancier. First, plot the data but don't display the points (that is, use `type="n"`):

```
> plot(Breadth ~ Length, type = "n")
```

Then plot Breadth against Length differently for each Site, using a different plotting character (`pch`) and colour (`col`):

```
> points(Breadth[Site == "DangarIsland"] ~ Length[Site ==
+     "DangarIsland"], pch = 19, col = "red")
> points(Breadth[Site == "SalamanderBay"] ~ Length[Site ==
+     "SalamanderBay"], pch = 15, col = "blue")
```

If you want to be really clever, add a legend:

```
> legend("topleft", pch = c(19, 15), col = c("red", "blue"),
+     legend = c("Salamander Bay", "Salamander Bay"))
```

The final plot is shown in Figure 4.1.

This shows us that the data for each Site is different, even though the linear relationship looks quite similar: Salamander Bay jellyfish are generally *larger*.

And remember:

```
> detach(jf)
```

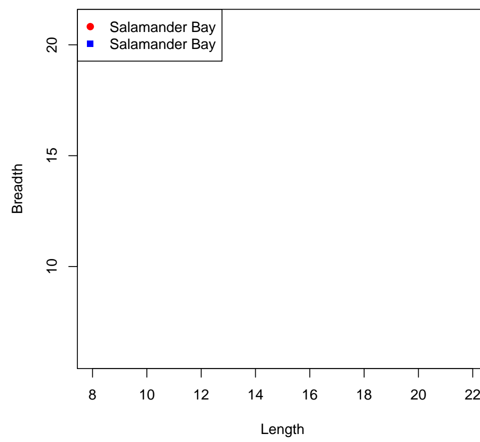


Figure 4.1: Plotting the jellyfish data, using different plotting characters for each location