Chapter 1

Software, data handling, descriptive statistics and graphical methods

1.1 Objectives

1. To use R and its basic data handling facilities.
2. To calculate appropriate descriptive statistics to summarise data.
3. To use a range of graphical methods to explore univariate and bivariate data.

Whenever a study is carried out, data are obtained. These data need to be organised, displayed, summarised and analysed in order to address the question(s) being asked. This lab presents methods for organising, displaying and summarising data. Many methods have been developed (and continue to be developed) in order to cope with the different types of data collected, differently-sized data sets and the different types of questions that are asked.

We start with a brief introduction to the software for the course, R. Please note that the first six chapters of the icebreakeR document (available online) are recommended reading.

1.2 Using R

This course uses R, which is a free, open-source statistical environment.

R is a computer language for data analysis and modeling. R can be used as an object-oriented programming language, or as a statistical environment within which lists of instructions can be performed sequentially without user input.

We can interact with R by typing or pasting commands into a command line in the console. This is what the prompt looks like:

```
>
```

Or we can write them in a text editor[1] and submitting them to the console, using the `source()` command. More about that later.

Examples of the kinds of commands we might use are:

```
> 1 + 1
```

```
[1] 2
```

[1] *E.g.*, WinEDT, vi, or one of the emacs family, or R's own text editor. We recommend that you do not use a word processor! The specific problem with using a word-processor to *write* scripts is that they make all sorts of assumptions about grammar and spelling that are not true for scripts.

This exclusive use of a command-line interface (CLI) does make our learning curve steeper. However, there are many advantages:

- It allows us to keep a record of the steps undertaken;
- It allows us to easily rerun commands without intervention;
- It allows us to make templates of often-used graphics or reports;
- It aids in the communication of the analysis to others;
- It protects us against change: when software changes, menus change but syntax rarely does.

Some interaction with R can be done through menus for some operating systems, but this interaction is mainly administrative. Some users have written GUIs for R, but we do not consider them here.[2]

You can find and freely download the executables and source code at: `http://www.r-project.org`.

▷ **Example. Getting Fired Up**

Start R on your computer, and try the command that you saw above. Verify that R can do basic arithmetic.

1.2.1 Using this Document

This document is in `pdf` format, which gives you the ability to copy any text to the clipboard. You can then paste it to the R console (in MS Windows) by right-clicking in the console and selecting the "Paste commands only" option. This will save you a lot of tedious typing.

The R commands are printed in a slanting typewriter font. It can be a little misleading, for example when the vertical bar | is used and appears to be a slash. Copying and pasting is safer than typing for yourself. Also, commands that span more than one line are connected by means of a `+` sign. Do not type this in if you are transcribing the code; it will create errors. The "Paste commands only" option takes care of this detail; it also cleverly ignores any non-commands, for example, this paragraph.

If you do choose to type the commands in to the console, you must always conclude your input using the `Enter` key.

1.2.2 Planning Ahead

In order to proceed we need to establish some protocols. These class notes will assume that you will adopt the following directory structure for labs and for exercises.

- The top-level directory in your personal disk space (e.g. on a thumb drive) will be associated with the course. Call it, for example, `MAST90044 Labs`.
- It will contain 11 directories, one for each lab class. Label these directories `lab01`, `lab02`, etc.
- Each lab directory will contain the following directories, which we will refer to as being *siblings* of each other:
  - `scripts` will be your *working directory* for each lab (see Section 1.2.3),

[2] More can be learned about these undertakings at `http://www.r-project.org/GUI`

- `data` for storing data sets that you need,
- `notes` for storing these notes,
- `graphics` where pdfs will be created, and
- `images` where you will save R objects if you wish.

We will assume that, when you start R, *you immediately tell R to use the scripts directory for the appropriate lab as the working directory.* We will also assume that any data sets that you have downloaded are stored in the data directory of the appropriate lab.

We will refer to these combinations in the following way: `MAST90044/lab01/data` refers to the data directory for lab 1 in this course.

The advantage of this setup is that, from a common working directory, we always know where in the computer to find the data. It will be in `../data`. The `../` tells R to go up one level in the directory hierarchy before finding the `data` directory. Being able to find the data is essential: the manual importation of exercise data is training for the importation of your own data into R.

▷ **Example. Making your World**

Make sure that your directory structure reflects the above.

1.2.3 Data Handling

In order to become proficient at data analysis with R, it is essential to be able to read in data. To read data, we need to know something about how R interacts with the files on the computer.

Working Directory

The working directory is the default location to and from which R writes and reads files.

If we want to read or write to a different location, we must explicitly say so. Life is therefore much easier if all the data and scripts for an analysis are kept in a single (frequently backed up!) location.

In the Command Line Interface versions of R one can use only the command line to check or select the working directory, as follows.

```
> getwd()                  # Prints the working directory.
> setwd("C:/Temp")         # Set "C:/Temp" to be the working directory.
```

This latter command has a few points worth discussing.

- `setwd()` is a *function*; we use functions to tell R what we want it to do, and
- `"C:/Temp"` is an *argument* for the function. Arguments are the main way that we tell R what to apply the function to or specify other features of the function. In this case, we use the argument to tell R that we would like to use `C:/Temp` as the working directory.

The forward slashes are used regardless of the underlying operating system. This is unfamiliar and distracting if you are accustomed to working in Windows, where backward slashes are conventionally used.

In Windows, there is a menu item that allows for selecting the working directory:

Select `File > Change dir` and the working directory will be highlighted. You can browse to change it.

▷ **Example. Setting the Working Directory**

Make the `scripts` directory in `lab01` your working directory. Verify that you have got it right using the `getwd` function.

Reading Data

Importing data to R is a bit tricky. One has to know about the structure of the data in order to import it. In this subject, we will mainly be using comma-delimited files.

In comma-delimited files, each row corresponds to an observation, and each column corresponds to a variable. The columns are separated by commas, hence the name. The file names are almost invariably suffixed by "`.csv`".

▷ **Example.** Forest inventory is used to determine the quantity and quality of timber resources. The comma-delimited dataset named `ufc.csv` contains the tree measurements from a forest inventory carried out in the Upper Flat Creek stand of the University of Idaho Experimental Forest.

Download the `ufc.csv` dataset from the class website and save it to the data directory for the lab class. Make sure that your working directory is correctly set, using the `getwd` function. Then, import the `ufc` data using the following command.

```
> ufc <- read.csv("../data/ufc.csv")
```

This command also has a few points worth discussing. Firstly, if the command is executed correctly, then all you will see is another prompt. *You won't see any other output.* This can be confusing to the novice!

Secondly, `read.csv` creates a new object in the random-access computer memory. The object is called `ufc`, and it contains the result of reading the comma-delimited data file called `"ufc.csv"`, which is stored in the sibling directory called `data`.

- `ufc` is the name of the new object; hereafter, if we wish to manipulate this object we can do so by the name "ufc";
- `<-` is the assignment operator. This is how we tell R to create (or recreate) an object;
- `read.csv()` tells R to find a comma-delimited file that holds the data; and
- `"../data/ufc.csv"` is the name of the file, *relative to the working directory.*

Having input the data, and called it `ufc`, we can examine it using the `str` function (`str` is short for "structure").

```
> str(ufc)
```

```
'data.frame':        336 obs. of  5 variables:
 $ plot    : int  2 2 3 3 3 4 4 5 5 6 ...
 $ tree    : int  1 2 2 5 8 1 2 2 4 1 ...
 $ species : Factor w/ 4 levels "DF","GF","WC",..: 1 4 2 3 3 3 1 1 2 1 ...
 $ dbh.cm  : num  39 48 52 36 38 46 25 54.9 51.8 40.9 ...
 $ height.m: num  20.5 33 30 20.7 22.5 18 17 29.3 29 26 ...
```

We now know that `ufc` is a data frame, which is a special kind of R object that can be thought of as a dataset, with a row for each observation and a column for each variable. See the icebreakeR for more information on data frames.

We can also read files in with arbitrary column separators (using `read.table`), in fixed-width format (`read.fwf`), or one line at a time (`scan`).

1.2.4 Work space

The work space is distinct from the working directory. The work space is R's operating table, as it were. All the objects that we create are in the work space (this is not entirely true, but will suffice for the moment).

We can find out what objects we have created in the workspace using `ls()`. Try it.

We delete them using the `rm` function. For example, `rm(ufc)` deletes the `ufc` object.

It is often useful to *look carefully at the data* using displays and/or summaries. Many interesting things can often be noted, indeed there is often no need for a formal analysis in order to answer the question(s) being asked. Used in this way, descriptives statistics and graphical methods are referred to as exploratory data analysis (EDA).

1.3 Descriptive Statistics

When we have read the data we need to examine them, partially to learn more about their structure, and partially to be sure that we know what we are looking at.

There are two main types of variable, **categorical** (qualitative) and **numerical** (quantitative), each of which can be further divided: categorical into **nominal** (e.g. gender) and **ordered** (e.g. severity of disease — mild/moderate/severe) and numerical into **discrete** (e.g. number of children) and **continuous** (e.g. temperature).

1.3.1 Categorical variables

Tables of counts by category usually provide the best summaries of categorical variables.

▷ **Example.** One key aspect of forest inventory in native forests is to determine the species balance, that is, the number of individuals of each species. A good starting point in summarising categorical variables is to construct tables of their counts.

In order to focus on a particular variable within the dataframe, we extract it by means of the $ operator. For example, to know more about the `species` variable in the `ufc` data frame, we use the expression `ufc$species`. A table of counts is produced by

```
> table(ufc$species)

 DF  GF  WC  WL
 57 118 139  22
```

FYI, the four species codes are: `DF` – Douglas fir; `GF` – grand fir; `WC` – western red cedar; `WL` – western larch.

From the frequency table above, we see that the most common tree species is western red cedar and the least common is western larch.

If we provide two arguments to the `table` function, then we get two dimensions, i.e. a crosstable of frequencies. For the `ufc` data, trees were numbered within each plot. If the tree numbers represented different positions in the plot, and we were interested in how often each species took up the different positions, we would construct the following crosstable:

```
> table(ufc$species,ufc$tree)

     1  2  3  4  5  6  7  8  9 10
  DF 14 10 15  8  5  4  0  1  0  0
```

```
GF 39 41 18 11  5  1  3  0  0  0
WC 51 34 28  8  2  7  6  1  1  1
WL 11  4  1  2  2  2  0  0  0  0
```

If we want one of the margins to sum to 1, then we use the `prop.table` function instead, as it gives proportions rather than frequencies.

1.3.2 Numerical variables

It is often useful to examine a few summary statistics that show the main features of the distribution of data values. The two most obvious features of the data are (i) the typical or central value, considered as a measure of the location of the data, and (ii) the spread or variability. Each of the features can be described in a number of different ways. Here we focus on those descriptions that are most popular in statistical work.

Measures of location

The two most commonly used measures of location are the *sample mean* and the *sample median*.

- *Sample mean* or *average*. The sample mean is the sum of the observations divided by the number of measurements. Suppose we have n observations and let x_1 denote the first observation, x_2 the second, and so on up to x_n. Then the sample mean, $\bar{x}$, is given by:

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

  ▷ **Example.** One way of measuring the size of a tree is to measure the diameter of the trunk, outside the bark, at a height of 1.2 metres from the ground (this is called diameter at breast height, or dbh). The mean of a sample of such numbers provides information about the location of the diameters. In R, the function is

```
> mean(ufc$dbh.cm)

[1] 37.41369
```

  We also often need to compute a measure of location conditioned on a categorical variable, i.e separately for each category.

  ▷ **Example.** The mean diameter for each species is found by the `tapply` function, which tells R to perform the calculation on the variable named in the first argument, using categories defined by the variable named in the second argument, and apply the function named in the third argument to the data in each category.

```
> tapply(ufc$dbh.cm, ufc$species, mean)

      DF       GF       WC       WL
39.90526 35.21186 38.84460 33.72727
```

- *Sample median*. The sample median is the middle measurement when the measurements are arranged in order. If there is an even number of measurements, it is the average of the middle two. Let $x_{(1)}$ denote the smallest observation, $x_{(2)}$ the second smallest and so on up to the largest, $x_{(n)}$. The sample median, m, can be defined in terms of these order statistics as:

$$m = x_{\left(\frac{n+1}{2}\right)}$$

When n is even, $x_{\left(\frac{n+1}{2}\right)}$ is defined to be the average of $x_{\left(\frac{n}{2}\right)}$ and $x_{\left(\frac{n}{2}+1\right)}$.

▷ **Example.** For dbh:

```
> median(ufc$dbh.cm)

[1] 35

> tapply(ufc$dbh.cm, ufc$species, median)

   DF    GF    WC    WL
40.00 32.60 37.70 29.05
```

Since it is not affected by a few wild values (like the mean is) the median is said to be *robust* to outliers. On the other hand, it is a less efficient estimator, meaning that, all other things equal, the uncertainty about the true median given an estimate of the median is larger than the uncertainty about the true mean given an estimate of the mean.

Another measure of location is the *Winsorized mean*, which is the mean of the sample with extreme values removed. The Winsorized mean is used because it is robust to outliers, although not as robust as the median.

Measures of spread

Several measures of spread are described in most statistics books. Most statistical work uses the *sample standard deviation*, and so shall we.

- *Sample standard deviation.* The sample standard deviation, s, is calculated as:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}.$$

That is, we calculate the differences between each observation and the average and square these differences to make them positive. Then we add them up and divide by $n-1$. Finally we take the square root of this.

A useful rule of thumb is that for many data sets, *approximately 68% of the data lie within one standard deviation of the mean, and about 95% of the data lie within two standard deviations of the mean.*

So for small data sets (less than 50 points), *the standard deviation is about one quarter of the range.* We will see later why this is true – it is based on the normal distribution.

The square of the standard deviation is the variance. It is more difficult to interpret than the standard deviation since it is not in the same units as the data. Its main advantage is that it is easier to work with mathematically.

▷ **Example.** The standard deviation of the diameters of the trees is:

```
> sd(ufc$dbh.cm)

[1] 17.31454
```

We can find the standard deviation by species as follows.

```
> tapply(ufc$dbh.cm, ufc$species, sd)

      DF       GF       WC       WL
16.41194 16.66866 18.44310 14.45746
```

Distribution of the data

An overview of the distribution of the data can be obtained by the so-called five-number summary, which comprises the median, the first and third quartiles, and the minimum and maximum. The first quartile is the value below which approximately a quarter of the data lie; the third quartile is the value above which approximately a quarter of the data lie.

▷ **Example.** Here is a numerical summary for ufc:

```
> summary(ufc$dbh.cm)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.10   24.80   35.00   37.41   47.15  101.50
```

We can find the summary by species as follows.

```
> tapply(ufc$dbh.cm, ufc$species, summary)

$DF
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  14.00   28.20   40.00   39.91   50.20   99.80

$GF
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.20   23.52   32.60   35.21   44.40   86.10

$WC
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.10   26.00   37.70   38.84   49.20  101.50

$WL
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  12.50   23.12   29.05   33.73   43.85   70.80
```

R can use different algorithms to determine the first and third quantiles; the details are not important for our purposes[3].

The five-number summary is the basis of the boxplot, which is its most common graphical representation (see Section 1.4.1).

Correlation coefficient

The correlation coefficient, denoted by r, is a measure of the strength of the *linear* relationship between two (numerical) variables. For variables x and y:

[3]There are nine different algorithms for quantiles. You can learn more about it by typing `?quantile` at the command line if you are interested.

$$r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s_x}\right)\left(\frac{y_i - \bar{y}}{s_y}\right)$$

- The value of r always lies between –1 and 1. Positive r indicates positive association between the variables (as x increases, so does y) and negative r indicates negative association.
- The extreme values $r = -1$ and $r = 1$ only occur when the data lie exactly on a straight line, a perfect linear relationship.
- A correlation of 0 indicates no *linear* relation between x and y.

▷ **Example.** The correlation between height and diameter for the ufc trees is:

```
> cor(ufc$dbh.cm, ufc$height.m)

[1] 0.7699552
```

1.4 Graphical Methods

The things to look for in a data display include:

- shape — symmetric or skewed, unimodal or multimodal,
- location — where to find the 'centre' of the distribution,
- spread — how spread out the values are,
- outliers — any very unusual values, and
- association — between (pairs of) variables.

It is possible to obtain numerical summaries of the above features (e.g. mean, standard deviation and correlation).

1.4.1 Univariate Data

The first kind of graphical methods we will consider are those which present individual variables, or *univariate data*.

- A *bar chart* is often used to show the distribution of a categorical variable or an ordinal variable.

  ▷ **Example.** The number of trees of each species is plotted in Figure 1.1.

  ```
  > barplot(table(ufc$species), horiz=TRUE)
  ```

  Note the horizontal orientation of the bars, caused by the `horiz = T` argument. The default is for vertical bars.

- A *histogram* is a bar chart representing frequencies in different ranges of the values of a numerical variable.

  To construct the classes:

  1. divide the **range** of the data by k, the number of desired intervals;
  2. round the answer to a sensible unit;
  3. choose the first class to contain the smallest observation, and make sure intervals don't overlap.

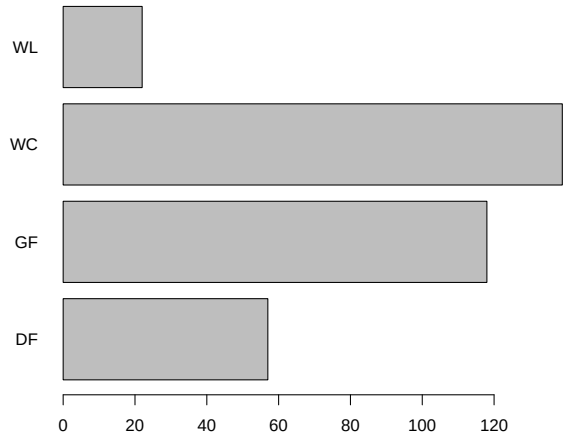

Figure 1.1: The number of trees of each species from the ufc data.

▷ **Example.** A histogram of the number of trees of each species is plotted in Figure 1.2.

```
> hist(ufc$dbh.cm)
```

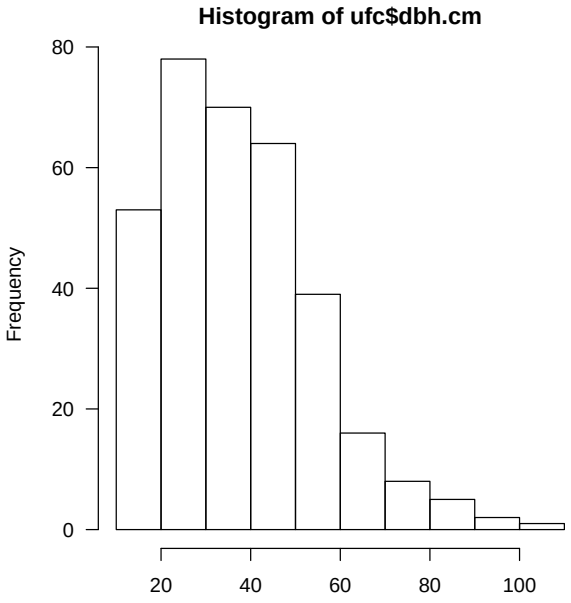

Figure 1.2: The number of trees of each diameter class from the ufc data.

- A *density plot* is a smoothed histogram. The subjectiveness of histograms (e.g. the choice of bin width and starting point) is potentially unsatisfactory. One strategy is to try to smooth them out using kernel density estimators, something we will not discuss in detail here. Instead of choosing the bin width we have to choose the bandwidth, but there are algorithms for that too ...

▷ **Example.** The number of trees of each species is plotted in Figure 1.3.

```
> plot(density(ufc$dbh.cm))
```

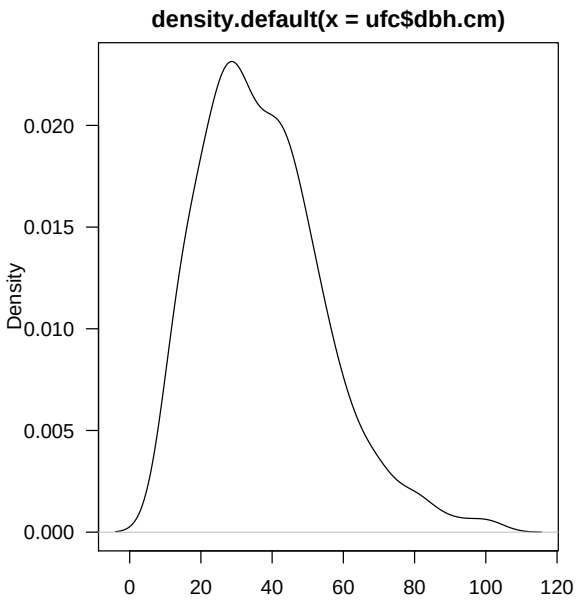

Figure 1.3: The number of trees of each diameter class from the ufc data.

- A *boxplot* is a visual display of the five-number summary. It provides a useful method for summarising a large data set into a few values which show its location, spread and general shape. It is also 'robust', in that the general features are not much affected by a few wild values.

  1. The 'box' is drawn between the lower and upper quartiles, with a line at the median.
  2. 'Whiskers' are drawn to the largest and smallest values that are not 'outliers'; outliers are often marked specifically.
  3. The quantitative variable can go on the horizontal or the vertical axis. Each way has its merits.
  4. The 'width' of the boxes (the dimension perpendicular to the whiskers) can be made proportional to the square root of the sample size.

▷ **Example.** The dbh of trees of each species is plotted *by species* in Figure 1.4. This is sometimes called a conditioning plot.

```
> boxplot(dbh.cm ~ species, varwidth = TRUE, data = ufc, horizontal=TRUE)
```

The last example has introduced a variation that is worth paying some attention to.

- `boxplot()` is the function name, as before,
- `data = ufc` tells the function to look inside the `ufc` data for the objects that it needs,
- `dbh.cm ~ species` tells the function to plot the boxplot of `dbh.cm` by the categories defined by the `species` variable, and
- `varwidth = TRUE` tells R that it should make the box width vary depending on the number of objects in each category.

The `dbh.cm ~ species` is a useful kind of object called a *formula*. We will use it much more in modelling.

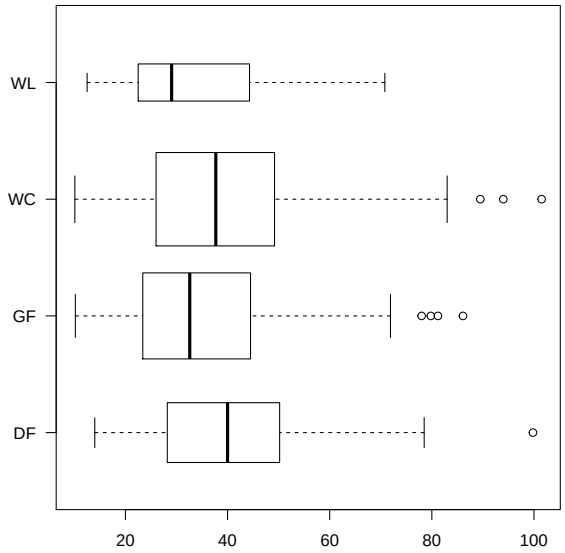

Figure 1.4: The dbh of trees of each species from the ufc data.

1.4.2 Bivariate Data

We may also wish to use graphical methods to represent the way two variables relate.

- *Scatterplots* are used to graphically present the relationship between pairs of variables.

  ▷ **Example.** The dbh of trees of each species is plotted against their height in Figure 1.5, together with a line of best fit (fitted using least squares).

```
> plot(height.m ~ dbh.cm, data = ufc)
> abline(lm(height.m ~ dbh.cm, data = ufc))
```

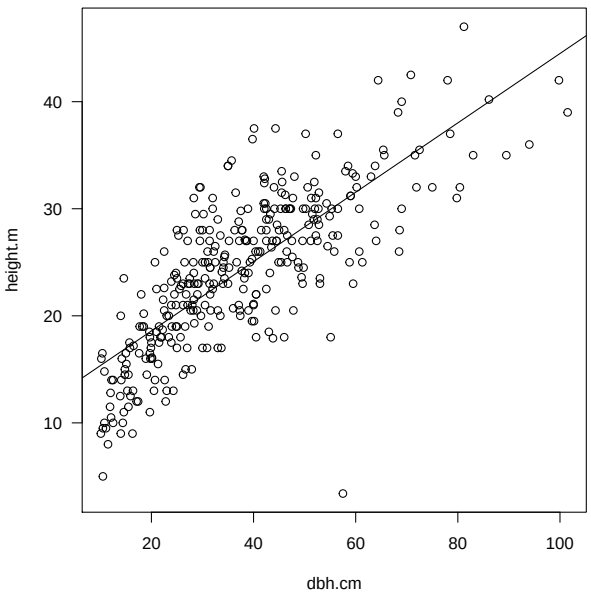

Figure 1.5: The dbh and height of trees from the ufc data.

The `plot` command in R is a versatile command which chooses an appropriate plot type

for the class of the data given. It usually creates a sensible plot without needing further options (though there are many such options available).

- We can also obtain *conditional scatterplots*. To do so we need to access some portions of R that are not loaded by default.

▷ **Example.** The dbh of trees is plotted against the height in Figure 1.6, conditioned on species.

```
> library(lattice)
```

This command tells R to load the `lattice` library. Doing so adds extra functionality to R that is not enabled when R starts.

Now we can easily plot the height and diameter of the trees by species, where we put each species into a separate panel within the plot (Figure 1.6).

```
> xyplot(height.m ~ dbh.cm | species, data = ufc)
```

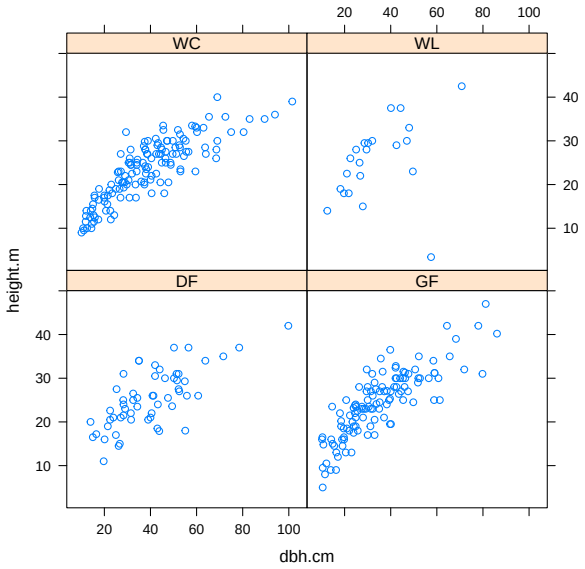

Figure 1.6: The dbh and height of trees from the ufc data, conditioned upon species in separate panels.

Alternatively, we can plot the height and diameter of the trees by species, where we put each species into distinct groups within the panel (Figure 1.7).

```
> xyplot(height.m ~ dbh.cm, groups = species,
+        auto.key = list(space="right"), data = ufc)
```

The `auto.key` option determines where the legend is placed, in this case to the right of the plot.

These configurations can be combined to plot complex data structures. For example, the panel row might represent the level of one factor, the panel column might represent another factor, and a third factor might be represented by the symbols or colours in the plot.

- Often imposing a line of best fit upon the scatterplot provides visual cues as to the nature of the pattern, which we do in the form of *scatterplots with smoothers*.

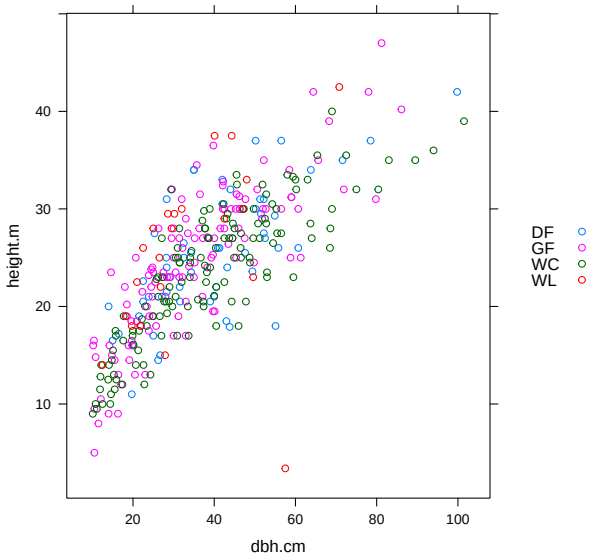

Figure 1.7: The dbh and height of trees from the ufc data, conditioned upon species with separate colours.

▷ **Example.** The dbh of trees is plotted against the height in Figure 1.8, conditioned on species, with a smooth line added. We won't worry about the mathematics that underpins the selection of the line or how it is constructed.

```
> xyplot(height.m ~ dbh.cm | species, type = c("p", "smooth"), data=ufc)
```

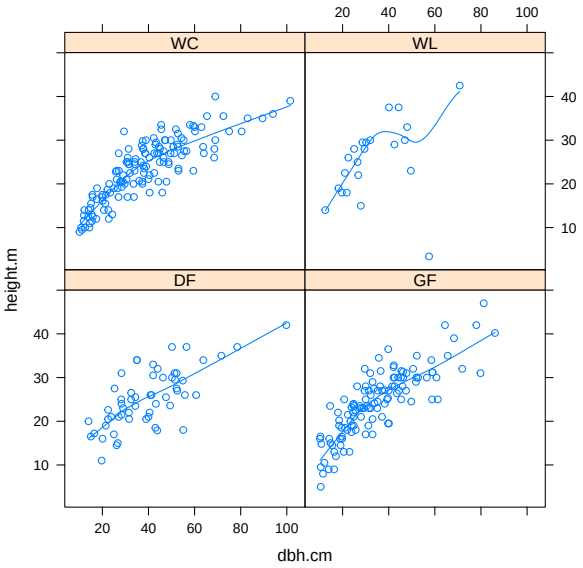

Figure 1.8: The dbh and height of trees from the ufc data, conditioned upon species, with a smooth line added.

1.4.3 Saving Graphics

There are two main ways to save graphics using R for Windows.

1. Right-click the graph window, select "Copy to clipboard as windows meta-file." and then paste into a Word document, for example. The graphic will appear as a scaleable object in the document.

2. Surround the code that you use to make the graph with the following function calls: one to create a device to receive the graphic, and one to close the device. This is illustrated by the following:

▷ **Example.** To create a pdf with the scatterplot of height against diameter, contained in the current working directory, we would use:

```
> pdf("../graphics/dbh-height.pdf")
> plot(height.m ~ dbh.cm, data = ufc)
> dev.off()
```

1.5 Exercises

At this point, you have finished the formal material of the lab. Following are a series of exercises for you to work on. They have decreasing amounts of detail. As you attempt each one, reflect upon what you have learned during the previous exercise. As you proceed we will provide fewer and fewer hints.

If at any time you would like to stop, and restart later, then you have two choices. Firstly, you can just quit and start again from where you left off. Most of the exercises are self-contained. If, however, you would like to stop in the middle of an exercise (and suddenly go for coffee, for example), then you can save your workspace. You would do this as follows.

```
> save.image("lab01.RData")
```

at some later time when you wish to return, use

```
> load("lab01.RData")
```

Note that only the objects in your workspace have been saved. Your graphs must be recreated but you can save the code used to produce them in a script file.

1. **Effect of 'the pill' on blood pressure.** (From *Freedman, Pisani and Purves,* 1978.)

   In a study of the side effects of the oral contraceptive 'the pill' 17,500 women aged 17-58 took part in a study by the Kaiser Clinic in California during 1969-1971. The study compared the blood pressures (BP) of users and nonusers of the pill. The results turned out to be similar for different age groups, and here the data on women aged 25 to 34 are given. There were 1747 users and 3040 nonusers in that age group. The table gives the distribution of systolic blood pressures (in mm.Hg) tabulated by pill use: for nonusers (the first column) and for users (the second column). Class intervals include the left point, but not the right.

   Paste the following code into R, to obtain the frequency table below. Don't forget to use `Paste commands only`.

```
> bp <- data.frame(class = c("085-090", "090-095", "095-100",
+ "100-105", "105-110", "110-115", "115-120", "120-125", "125-130",
+ "130-135", "135-140", "140-145", "145-150", "150-155", "155-160"),
+                  nonusers = c(1,1,5,11,11,17,18,11,9,7,4,2,2,1,0),
+                  users = c(0,0,4,5,10,15,17,14,12,10,5,4,2,1,1))
> bp
```

```
     class nonusers users
1  085-090        1     0
2  090-095        1     0
3  095-100        5     4
4  100-105       11     5
5  105-110       11    10
6  110-115       17    15
7  115-120       18    17
8  120-125       11    14
9  125-130        9    12
10 130-135        7    10
11 135-140        4     5
12 140-145        2     4
13 145-150        2     2
14 150-155        1     1
15 155-160        0     1
```

   Note that we populate the data frame by constructing and naming vectors. The `c` function joins all of its arguments into a vector.

   The use of "085-090" rather than "85-90" is a bit ugly, but it is necessary when plotting (see (d)) to keep the classes in the correct order.

   (a) What is the question being asked here?

   (b) What type of study is it?

(c) What are the variables and what types of variables are they?

(d) What do you conclude from these data? You might like to try the following commands to produce a graphical display of these data:

```
par(mfrow = c(2,1))
plot(nonusers~class,data=bp)
plot(users~class,data=bp)
```

The R `plot` function as usual manages to produce a plot, though it is not very attractive on this occasion. We won't worry about this for the moment. The `par` command allows multiple figures in the one graphical object, in this case two in each row and one in each column (hence the (2,1)).

(e) How might the study be improved?

2. **Pulse rate and exercise.** (from OzDASL[4])

```
> pulse <- read.table("../data/ms212.txt", header=TRUE)
```

Here it is important to tell R that the first row of the table contained the variable names. We do so via the argument `header=TRUE`.

A number of students measured their pulse (`Pulse1`). Each tossed a coin to determine whether to run on the spot for 1 minute. All then re-measured their pulse (`Pulse2`). Also recorded were gender (1 = male, 2 = female), whether they ran (1 = yes, 2 = no), level of other exercise (1 = high, 2 = moderate, 3 = low), smoking status (1 = regularly, 2 = not regularly), height (in cm) and weight (in kilograms). Data from the first ten students are shown below.

```
> head(pulse, n=10)
```

	Height	Weight	Age	Gender	Smokes	Alcohol	Exercise	Ran	Pulse1	Pulse2	Year
1	173	57	18	2	2	1	2	2	86	88	93
2	179	58	19	2	2	1	2	1	82	150	93
3	167	62	18	2	2	1	1	1	96	176	93
4	195	84	18	1	2	1	1	2	71	73	93
5	173	64	18	2	2	1	3	2	90	88	93
6	184	74	22	1	2	1	3	1	78	141	93
7	162	57	20	2	2	1	2	2	68	72	93
8	169	55	18	2	2	1	2	2	71	77	93
9	164	56	19	2	2	1	1	2	68	68	93
10	168	60	23	1	2	1	2	1	88	150	93

(a) Use the `tapply` and `plot` functions in R to examine the effect of each variable on Pulse1.

(b) What is the effect of running on pulse?

3. **Test results.** A class of 122 students obtained the following scores on a test.

```
> scores <- c(16, 14, 15, 16, 14, 20, 18, 16, 13, 17, 16, 13, 18,
+             16, 22, 20, 24, 18, 16, 20, 26, 19, 14, 17, 14, 15,
+             19, 12, 11, 16, 19, 13, 13, 18, 8, 16, 28, 16, 27,
+             17, 12, 15, 7, 12, 22, 20, 16, 10, 8, 16, 13, 14,
```

[4] http://www.statsci.org/data/index.html

```
+             14, 16, 18, 15, 21, 23, 16, 5, 14, 23, 17, 15, 14,
+             22, 20, 22, 13, 20, 18, 13, 14, 21, 14, 18, 10, 20,
+             24, 17, 21, 15, 18, 12, 23, 17, 10, 15, 11, 5, 16,
+             19, 22, 10, 15, 17, 13, 23, 20, 3, 18, 15, 22, 12,
+             9, 20, 16, 17, 17, 16, 21, 18, 11, 14, 6, 21, 25,
+             18, 26, 18, 18, 15)
```

Produce appropriate displays and summaries of these data.

4. **Soil water evaporation.** To investigate how evaporation of water from soil is affected by air flow above it, a single soil sample was used, the speed of airflow was varied and the rate of evaporation was measured.

Air speed	0.5	0.5	0.5	1.0	1.0	1.0	1.5	1.5	1.5
Evaporation	5.39	4.43	5.50	7.70	6.20	6.14	5.62	6.12	7.20

Air speed	2.0	2.0	2.0	2.5	2.5	2.5
Evaporation	6.88	7.73	6.01	5.10	7.29	7.28

Make a graphical study of the effect of air speed on evaporation.

```
> soil <- data.frame(air.speed = c(0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.5,
+                       1.5, 1.5, 2.0, 2.0, 2.0, 2.5, 2.5, 2.5),
+                     evaporation = c(5.39, 4.43, 5.50, 7.70, 6.20, 6.14,
+                       5.62, 6.12, 7.20, 6.88, 7.73, 6.01, 5.10, 7.29, 7.28))
```

5. **Melon yields.** Six plots of each of four varieties of melons gave the following yields (kg/m^2); the data are presented in Table 1.1.

Table 1.1: Melon yields (kg/m^2) by variety.

Variety			
A	B	C	D
25	40	18	28
17	35	23	29
26	32	26	33
16	37	15	32
22	43	11	30
16	37	24	28

Make a graphical study of the effect of variety on melon yields.

```
> melons <- data.frame(variety = rep(c("A","B","C","D"), 6),
+                      yield = c(
+                        25, 40, 18, 28,
+                        17, 35, 23, 29,
+                        26, 32, 26, 33,
+                        16, 37, 15, 32,
+                        22, 43, 11, 30,
+                        16, 37, 24, 28))
>
```

Note the use of the `rep` function to `rep`eat the first argument—the four letters—the number of times noted by the second argument. Experiment with `rep` to become familiar with it.

6. **Tree inventory data.** A forest inventory was carried out in the East Hatter Creek stand of the University of Idaho experimental forest. The data are on the class website in the `ehc.csv` dataset. Download these data, and load them into R. Use the graphical tools presented in this chapter to explore the relationship between height and dbh.

7. **Lymphocyte counts.** In an experiment to compare the effects of four drugs (A, B, C and a placebo D) on the lymphocyte counts in mice, a randomised block design with four mice from each of five litters was used, the litters being regarded as blocks. The lymphocyte counts (thousands per mm^3 blood) were :

	Litter				
	1	2	3	4	5
A	7.1	6.1	6.9	5.6	6.4
B	6.7	5.1	5.9	5.1	5.8
C	7.1	5.8	6.2	5.0	6.2
D	6.7	5.4	5.7	5.2	5.3

Create a dataframe for these data. You will need the `rep` function in R.

Plot lymphocyte counts against drug and then against litter. Think about which plot is more informative in helping to achieve the aim of the experiment.