



Chapter 1

Introduction

Statistics is fundamentally concerned with the understanding of structure in data. One of the effects of the information-technology era has been to make it much easier to collect extensive datasets with minimal human intervention. Fortunately, the same technological advances allow the users of statistics access to much more powerful ‘calculators’ to manipulate and display data. This book is about the modern developments in applied statistics that have been made possible by the widespread availability of workstations with high-resolution graphics and ample computational power. Workstations need software, and the S^1 system developed at Bell Laboratories (Lucent Technologies, formerly AT&T) provides a very flexible and powerful environment in which to implement new statistical ideas. Lucent’s current implementation of S is exclusively licensed to the Insightful Corporation², which distributes an enhanced system called S -PLUS.

An Open Source system called R^3 has emerged that provides an independent implementation of the S language. It is similar enough that almost all the examples in this book can be run under R .

An S environment is an integrated suite of software facilities for data analysis and graphical display. Among other things it offers

- an extensive and coherent collection of tools for statistics and data analysis,
- a language for expressing statistical models and tools for using linear and non-linear statistical models,
- graphical facilities for data analysis and display either at a workstation or as hardcopy,
- an effective object-oriented programming language that can easily be extended by the user community.

The term *environment* is intended to characterize it as a planned and coherent system built around a language and a collection of low-level facilities, rather than the ‘package’ model of an incremental accretion of very specific, high-level and

¹The name S arose long ago as a compromise name (Becker, 1994), in the spirit of the programming language C (also from Bell Laboratories).

²<http://www.insightful.com>

³<http://www.r-project.org>

sometimes inflexible tools. Its great strength is that functions implementing new statistical methods can be built on top of the low-level facilities.

Furthermore, most of the environment is open enough that users can explore and, if they wish, change the design decisions made by the original implementors. Suppose you do not like the output given by the regression facility (as we have frequently felt about statistics packages). In **S** you can write your own summary routine, and the system one can be used as a template from which to start. In many cases sufficiently persistent users can find out the exact algorithm used by listing the **S** functions invoked. As **R** is Open Source, *all* the details are open to exploration.

Both **S-PLUS** and **R** can be used under Windows, many versions of UNIX and under Linux; **R** also runs under MacOS (versions 8, 9 and X), FreeBSD and other operating systems.

We have made extensive use of the ability to extend the environment to implement (or re-implement) statistical ideas within **S**. All the **S** functions that are used and our datasets are available in machine-readable form and come with all versions of **R** and Windows versions of **S-PLUS**; see Appendix C for details of what is available and how to install it if necessary.

System dependencies

We have tried as far as is practicable to make our descriptions independent of the computing environment and the exact version of **S-PLUS** or **R** in use. We confine attention to versions 6 and later of **S-PLUS**, and 1.5.0 or later of **R**.

Clearly some of the details must depend on the environment; we used **S-PLUS** 6.0 on Solaris to compute the examples, but have also tested them under **S-PLUS** for Windows version 6.0 release 2, and using **S-PLUS** 6.0 on Linux. The output will differ in small respects, for the Windows run-time system uses scientific notation of the form $4.17\text{e-}005$ rather than $4.17\text{e-}05$.

Where timings are given they refer to **S-PLUS** 6.0 running under Linux on one processor of a dual 1 GHz Pentium III PC.

One system dependency is the mouse buttons; we refer to buttons 1 and 2, usually the left and right buttons on Windows but the left and middle buttons on UNIX / Linux (or perhaps both together of two). Macintoshes only have one mouse button.

Reference manuals

The basic **S** references are Becker, Chambers and Wilks (1988) for the basic environment, Chambers and Hastie (1992) for the statistical modelling and first-generation object-oriented programming and Chambers (1998); these should be supplemented by checking the on-line help pages for changes and corrections as **S-PLUS** and **R** have evolved considerably since these books were written. Our aim is not to be comprehensive nor to replace these manuals, but rather to explore much further the use of **S** to perform statistical analyses. Our companion book, Venables and Ripley (2000), covers many more technical aspects.

Graphical user interfaces (GUIs)

S-PLUS for Windows comes with a GUI shown in Figure B.1 on page 458. This has menus and dialogs for many simple statistical and graphical operations, and there is a **Standard Edition** that only provides the GUI interface. We do not discuss that interface here as it does not provide enough power for our material. For a detailed description see the system manuals or Krause and Olson (2000) or Lam (2001).

The UNIX / Linux versions of S-PLUS 6 have a similar GUI written in Java, obtained by starting with `Splus -g`: this too has menus and dialogs for many simple statistical operations.

The Windows, Classic MacOS and GNOME versions of R have a much simpler console.

Command line editing

All of these environments provide command-line editing using the arrow keys, including recall of previous commands. However, it is not enabled by default in S-PLUS on UNIX / Linux: see page 447.

1.1 A Quick Overview of S

Most things done in S are permanent; in particular, data, results and functions are all stored in operating system files.⁴ These are referred to as *objects*.

Variables can be used as scalars, matrices or arrays, and S provides extensive matrix manipulation facilities. Furthermore, objects can be made up of collections of such variables, allowing complex objects such as the result of a regression calculation. This means that the result of a statistical procedure can be saved for further analysis in a future session. Typically the calculation is separated from the output of results, so one can perform a regression and then print various summaries and compute residuals and leverage plots from the saved regression object.

Technically S is a function language. Elementary commands consist of either *expressions* or *assignments*. If an expression is given as a command, it is evaluated, printed and the value is discarded. An assignment evaluates an expression and passes the value to a variable but the result is not printed automatically. An expression can be as simple as `2 + 3` or a complex function call. Assignments are indicated by the *assignment operator* `<-`. For example,

```
> 2 + 3
[1] 5
> sqrt(3/4)/(1/3 - 2/pi^2)
[1] 6.6265
> library(MASS)
```

⁴These should not be manipulated directly, however. Also, R works with an in-memory workspace containing copies of many of these objects.

```

> data(chem) # needed in R only
> mean(chem)
[1] 4.2804
> m <- mean(chem); v <- var(chem)/length(chem)
> m/sqrt(v)
[1] 3.9585

```

Here `>` is the **S** prompt, and the `[1]` states that the answer is starting at the first element of a vector.

More complex objects will have printed a short summary instead of full details. This is achieved by an object-oriented programming mechanism; complex objects have *classes* assigned to them that determine how they are printed, summarized and plotted. This process is taken further in **S-PLUS** in which *all* objects have classes.

S can be extended by writing new functions, which then can be used in the same way as built-in functions (and can even replace them). This is very easy; for example, to define functions to compute the standard deviation⁵ and the two-tailed *P* value of a *t* statistic, we can write

```

std.dev <- function(x) sqrt(var(x))
t.test.p <- function(x, mu = 0) {
  n <- length(x)
  t <- sqrt(n) * (mean(x) - mu) / std.dev(x)
  2 * (1 - pt(abs(t), n - 1)) # last value is returned
}

```

It would be useful to give both the *t* statistic and its *P* value, and the most common way of doing this is by returning a list; for example, we could use

```

t.stat <- function(x, mu = 0) {
  n <- length(x)
  t <- sqrt(n) * (mean(x) - mu) / std.dev(x)
  list(t = t, p = 2 * (1 - pt(abs(t), n - 1)))
}

z <- rnorm(300, 1, 2) # generate 300 N(1, 4) variables.
t.stat(z)
$t:
[1] 8.2906
$p:
[1] 3.9968e-15

unlist(t.stat(z, 1)) # test mu=1, compact result
      t      p
-0.56308 0.5738

```

The first call to `t.stat` prints the result as a list; the second tests the non-default hypothesis $\mu = 1$ and using `unlist` prints the result as a numeric vector with named components.

Linear statistical models can be specified by a version of the commonly used notation of Wilkinson and Rogers (1973), so that

⁵**S-PLUS** and **R** have functions `stdev` and `sd`, respectively.

```
time ~ dist + climb
time ~ transplant/year + age + prior.surgery
```

refer to a regression of `time` on both `dist` and `climb`, and of `time` on `year` within each transplant group and on `age`, with a different intercept for each type of prior surgery. This notation has been extended in many ways, for example to survival and tree models and to allow smooth non-linear terms.

1.2 Using S

How to initialize and start up your S environment is discussed in Appendix A.

Bailing out

One of the first things we like to know with a new program is how to get out of trouble. S environments are generally very tolerant, and can be interrupted by Ctrl-C.⁶ (Use Esc on GUI versions under Windows.) This will interrupt the current operation, back out gracefully (so, with rare exceptions, it is as if it had not been started) and return to the prompt.

You can terminate your S session by typing

```
q()
```

at the command line or from Exit on the File menu in a GUI environment.

On-line help

There is a help facility that can be invoked from the command line. For example, to get information on the function `var` the command is

```
> help(var)
```

A faster alternative (to type) is

```
> ?var
```

For a feature specified by special characters and in a few other cases (one is "function"), the argument must be enclosed in double or single quotes, making it an entity known in S as a character string. For example, two alternative ways of getting help on the list component extraction function, `[`, are

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

Many S commands have additional help for `name.object` describing their result: for example, `lm` under S-PLUS has a help page for `lm.object`.

Further help facilities for some versions of S-PLUS and R are discussed in Appendix A. Many versions can have their manuals on-line in PDF format; look under the Help menu in the Windows versions.

⁶This means hold down the key marked Control or Ctrl and hit the second key.

1.3 An Introductory Session

The best way to learn **S** is by using it. We invite readers to work through the following familiarization session and see what happens. First-time users may not yet understand every detail, but the best plan is to type what you see and observe what happens as a result.

Consult Appendix A, and start your **S** environment.

The whole session takes most first-time users one to two hours at the appropriate leisurely pace. The left column gives commands; the right column gives brief explanations and suggestions.

A few commands differ between environments, and these are prefixed by **# R:** or **# S:**. Choose the appropriate one(s) and omit the prefix.

<code>library(MASS)</code>	A command to make our datasets available. Your local advisor can tell you the correct form for your system.
<code>?help</code>	Read the help page about how to use help.
<code># S: trellis.device()</code>	Start up a suitable device.

<code>x <- rnorm(1000)</code> <code>y <- rnorm(1000)</code>	Generate 1 000 pairs of normal variates
<code>truehist(c(x,y+3), nbins=25)</code>	Histogram of a mixture of normal distributions. Experiment with the number of bins (25) and the shift (3) of the second component.
<code>?truehist</code>	Read about the optional arguments.
<code>contour(dd <- kde2d(x,y))</code>	2D density plot.
<code>image(dd)</code>	Greyscale or pseudo-colour plot.

<code>x <- seq(1, 20, 0.5)</code> <code>x</code>	Make $x = (1, 1.5, 2, \dots, 19.5, 20)$ and list it.
<code>w <- 1 + x/2</code> <code>y <- x + w*rnorm(x)</code>	<code>w</code> will be used as a ‘weight’ vector and to give the standard deviations of the errors.
<code>dum <- data.frame(x, y, w)</code> <code>dum</code> <code>rm(x, y, w)</code>	Make a <i>data frame</i> of three columns named <code>x</code> , <code>y</code> and <code>w</code> , and look at it. Remove the original <code>x</code> , <code>y</code> and <code>w</code> .
<code>fm <- lm(y ~ x, data = dum)</code> <code>summary(fm)</code>	Fit a simple linear regression of <code>y</code> on <code>x</code> and look at the analysis.

<code>fm1 <- lm(y ~ x, data = dum, weight = 1/w^2) summary(fm1)</code>	Since we know the standard deviations, we can do a weighted regression.
<code># R: library(modreg)</code>	R only
<code>lrf <- loess(y ~ x, dum)</code>	Fit a smooth regression curve using a modern regression function.
<code>attach(dum)</code>	Make the columns in the data frame visible as variables.
<code>plot(x, y)</code>	Make a standard scatterplot. To this plot we will add the three regression lines (or curves) as well as the known true line.
<code>lines(spline(x, fitted(lrf)), col = 2)</code>	First add in the local regression curve using a spline interpolation between the calculated points.
<code>abline(0, 1, lty = 3, col = 3)</code>	Add in the true regression line (intercept 0, slope 1) with a different line type and colour.
<code>abline(fm, col = 4)</code>	Add in the unweighted regression line. <code>abline()</code> is able to extract the information it needs from the fitted regression object.
<code>abline(fm1, lty = 4, col = 5)</code>	Finally add in the weighted regression line, in line type 4. This one should be the most accurate estimate, but may not be, of course. One such outcome is shown in Figure 1.1.
<code>plot(fitted(fm), resid(fm), xlab = "Fitted Values", ylab = "Residuals")</code>	You may be able to make a hardcopy of the graphics window by selecting the <code>Print</code> option from a menu. A standard regression diagnostic plot to check for heteroscedasticity, that is, for unequal variances. The data are generated from a heteroscedastic process, so can you see this from this plot?
<code>qqnorm(resid(fm)) qqline(resid(fm))</code>	A normal scores plot to check for skewness, kurtosis and outliers. (Note that the heteroscedasticity may show as apparent non-normality.)

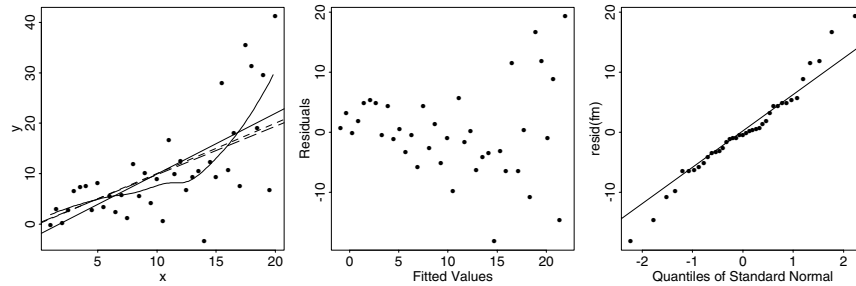


Figure 1.1: Four fits and two residual plots for the artificial heteroscedastic regression data.

<code>detach()</code>	Remove the data frame from the search
<code>rm(fm, fm1, lrf, dum)</code>	path and clean up again.

We look next at a set of data on record times of Scottish hill races against distance and total height climbed.

<code># R: data(hills)</code>	
<code>hills</code>	List the data.
<code># S: splom(~ hills)</code>	Show a matrix of pairwise scatterplots
<code># R: pairs(hills)</code>	(Figure 1.2).
<code># S: brush(hills)</code>	Try highlighting points and see how
	they are linked in the scatterplots (Fig-
	ure 1.3). Also try rotating the points in
	3D.
<code>Click on the Quit button in the</code> <code>graphics window to continue.</code>	
<code>attach(hills)</code>	Make columns available by name.
<code>plot(dist, time)</code>	Use mouse button 1 to identify outlying
<code>identify(dist, time,</code> <code>row.names(hills))</code>	points, and button 2 to quit. Their row
	numbers are returned. On a Macintosh
	click outside the plot to quit.
<code>abline(lm(time ~ dist))</code>	Show least-squares regression line.
<code># R: library(lqs)</code>	Fit a very resistant line. See Figure 1.4.
<code>abline(lqs(time ~ dist),</code> <code>lty = 3, col = 4)</code>	
<code>detach()</code>	Clean up again.

We can explore further the effect of outliers on a linear regression by designing our own examples interactively. Try this several times.

<code>plot(c(0,1), c(0,1), type="n")</code>	Make our own dataset by clicking with
<code>xy <- locator(type = "p")</code>	button 1, then with button 2 (outside the
	plot on a Macintosh) to finish.

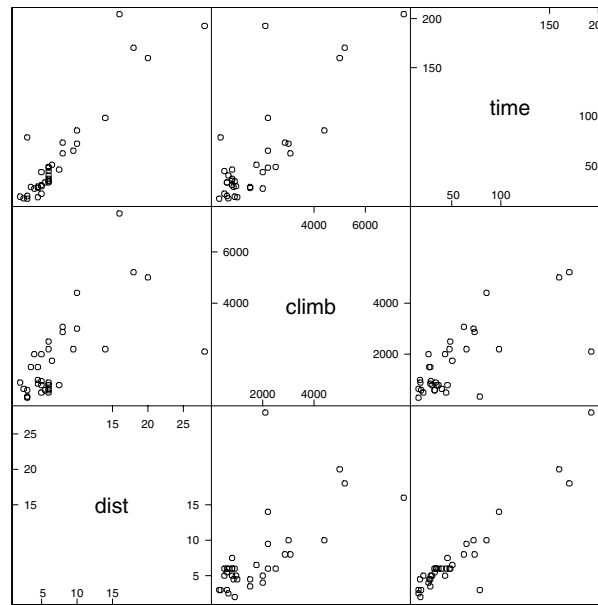
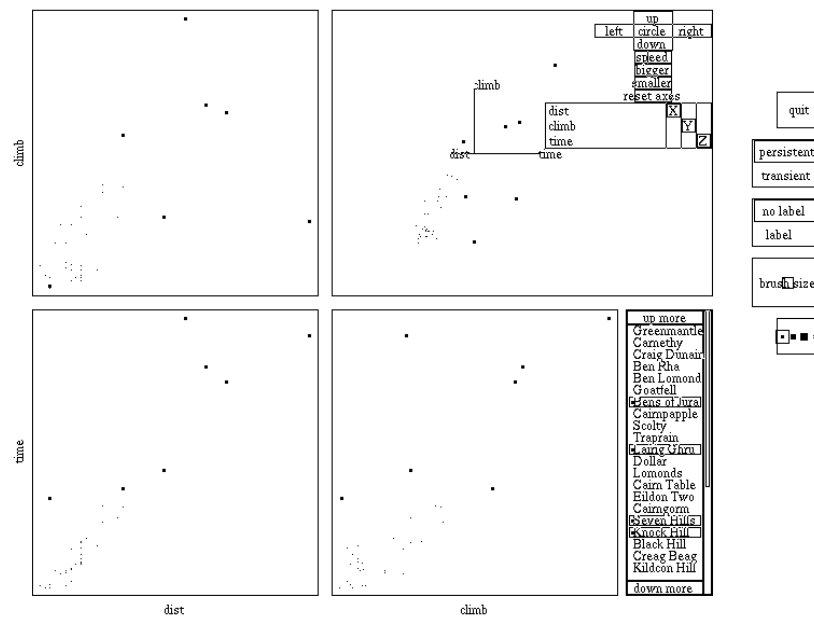


Figure 1.2: Scatterplot matrix for data on Scottish hill races.



press Button 1 to highlight, Button 2 to downlight

Figure 1.3: Screenshot of a brush plot of dataset hills (UNIX).

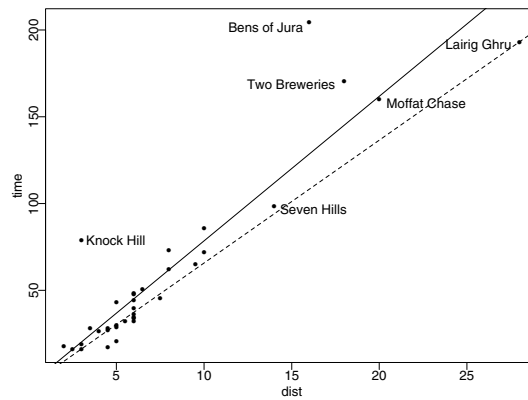


Figure 1.4: Annotated plot of time versus distance for hills with regression line and resistant line (dashed).

<code>abline(lm(y ~ x, xy), col = 4)</code>	Fit least-squares, a robust regression
<code>abline(rlm(y ~ x, xy,</code>	and a resistant regression line. Repeat
<code>method = "MM"),</code>	to try the effect of outliers, both verti-
<code>lty = 3, col = 3)</code>	cally and horizontally.
<code>abline(lqs(y ~ x, xy),</code>	
<code>lty = 2, col = 2)</code>	
<code>rm(xy)</code>	Clean up again.

We now look at data from the 1879 experiment of Michelson to measure the speed of light. There are five experiments (column `Expt`); each has 20 runs (column `Run`) and `Speed` is the recorded speed of light, in km/sec, less 299 000. (The currently accepted value on this scale is 734.5.)

<code># R: data(michelson)</code>	
<code>attach(michelson)</code>	Make the columns visible by name.
<code>search()</code>	The <i>search path</i> is a sequence of places, either directories or data frames, where S-PLUS looks for objects required for calculations.
<code>plot(Expt, Speed,</code>	Compare the five experiments with simple boxplots. The result is shown in Figure 1.5.
<code>main="Speed of Light Data",</code>	
<code>xlab="Experiment No.")</code>	
<code>fm <- aov(Speed ~ Run + Expt)</code>	Analyse as a randomized block design, with <i>runs</i> and <i>experiments</i> as factors.
<code>summary(fm)</code>	

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
Run	19	113344	5965	1.1053	0.36321

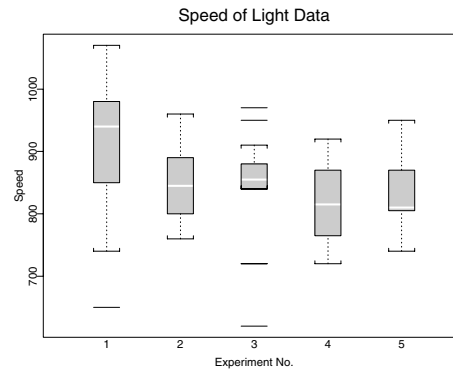


Figure 1.5: Boxplots for the speed of light data.

```
Expt      4      94514    23629    4.3781 0.00307
Residuals 76     410166    5397
```

```
fm0 <- update(fm, .~ . - Run)  Fit the sub-model omitting the non-
anova(fm0, fm)                 sense factor, runs, and compare using
                                a formal analysis of variance.
```

```
Analysis of Variance Table
Response: Speed
```

	Terms	Resid.	Df	RSS	Test	Df	Sum of Sq	F Value	Pr(F)
1	Expt		95	523510					
2	Run + Expt	76	410166	+Run	19	113344	1.1053	0.36321	

```
detach()                      Clean up before moving on.
rm(fm, fm0)
```

The S environment includes the equivalent of a comprehensive set of statistical tables; one can work out P values or critical values for a wide range of distributions (see Table 5.1 on page 108).

```
1 - pf(4.3781, 4, 76)         P value from the ANOVA table above.
qf(0.95, 4, 76)               corresponding 5% critical point.
```

```
q()
```

Quit your S environment. R will ask if you want to save the workspace: for this session you probably do not.

1.4 What Next?

We hope that you now have a flavour of **S** and are inspired to delve more deeply. We suggest that you read Chapter 2, perhaps cursorily at first, and then Sections 3.1–7 and 4.1–3. Thereafter, tackle the statistical topics that are of interest to you. Chapters 5 to 16 are fairly independent, and contain cross-references where they do interact. Chapters 7 and 8 build on Chapter 6, especially its first two sections.

Chapters 3 and 4 come early, because they are about **S** not about statistics, but are most useful to advanced users who are trying to find out what the system is really doing. On the other hand, those programming in the **S** language will need the material in our companion volume on **S** programming, Venables and Ripley (2000).

Note to R users

The **S** code in the following chapters is written to work with **S-PLUS 6**. The changes needed to use it with **R** are small and are given in the scripts available on-line in the `scripts` directory of the `MASS` package for **R** (which should be part of every **R** installation).

Two issues arise frequently:

- Datasets need to be loaded explicitly into **R**, as in the

```
data(hills)
data(michelson)
```

lines in the introductory session. So if dataset `foo` appears to be missing, make sure that you have run `library(MASS)` and then try `data(foo)`. We generally do not mention this unless something different has to be done to get the data in **R**.

- Many of the packages are not attached by default, so **R** (currently) needs far more use of the `library` function.

Note too that **R** has a different random number stream and so results depending on random partitions of the data may be quite different from those shown here.



<http://www.springer.com/978-0-387-95457-8>

Modern Applied Statistics with S

Venables, W.N.; Ripley, B.D.

2002, XII, 498 p., Hardcover

ISBN: 978-0-387-95457-8