Second Edition

Michael J. Crawley

STATISTICS

An introduction using *R*

WILEY

Statistics

Statistics

An Introduction Using R

Second Edition

Michael J. Crawley
Imperial College London, UK

This edition first published 2015

Registered office
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom

For details of our global editorial offices, for customer services and for information about how to apply for permission to reuse the copyright material in this book please see our website at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The publisher is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. It is sold on the understanding that the publisher is not engaged in rendering professional services and neither the publisher nor the author shall be liable for damages arising herefrom. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Library of Congress Cataloging-in-Publication Data

Crawley, Michael J.
Statistics : an introduction using R / Michael J. Crawley. – Second edition.
pages cm
Includes bibliographical references and index.
ISBN 978-1-118-94109-6 (pbk.)
1. Mathematical statistics–Textbooks. 2. R (Computer program language) I. Title.
QA276.12.C73 2015
519.50285'5133–dc23
2014024528

A catalogue record for this book is available from the British Library.

ISBN: 9781118941096 (pbk)

Set in 10/12pt, TimesLTStd-Roman by Thomson Digital, Noida, India.

1 2015

Contents

Preface

This book is an introduction to the essentials of statistical analysis for students who have little or no background in mathematics or statistics. The audience includes first- and second-year undergraduate students in science, engineering, medicine and economics, along with post-experience and other mature students who want to relearn their statistics, or to switch to the powerful new language of R.

For many students, statistics is the least favourite course of their entire time at university. Part of this is because some students have convinced themselves that they are no good at sums, and consequently have tried to avoid contact with anything remotely quantitative in their choice of subjects. They are dismayed, therefore, when they discover that the statistics course is compulsory. Another part of the problem is that statistics is often taught by people who have absolutely no idea how difficult some of the material is for non-statisticians. As often as not, this leads to a recipe-following approach to analysis, rather than to any attempt to understand the issues involved and how to deal with them.

The approach adopted here involves virtually no statistical theory. Instead, the assumptions of the various statistical models are discussed at length, and the practice of exposing statistical models to rigorous criticism is encouraged. A philosophy of model simplification is developed in which the emphasis is placed on estimating effect sizes from data, and establishing confidence intervals for these estimates. The role of hypothesis testing at an arbitrary threshold of significance like $\alpha = 0.05$ is played down. The text starts from absolute basics and assumes absolutely no background in statistics or mathematics.

As to presentation, the idea is that background material would be covered in a series of 1-hour lectures, then this book could be used as a guide to the practical sessions and for homework, with the students working on their own at the computer. My experience is that the material can be covered in 10–30 lectures, depending on the background of the students and the depth of coverage it is hoped to achieve. The practical work is designed to be covered in 10–15 sessions of about 1½ hours each, again depending on the ambition and depth of the coverage, and on the amount of one-to-one help available to the students as they work at their computers.

The R language of statistical computing has an interesting history. It evolved from the S language, which was first developed at the AT&T Bell Laboratories by Rick Becker, John Chambers and Allan Wilks. Their idea was to provide a software tool for professional statisticians who wanted to combine state-of-the-art graphics with powerful model-fitting capability. S is made up of three components. First and foremost, it is a powerful tool for statistical modelling. It enables you to specify and fit statistical models to your data, assess the goodness of fit and display the estimates, standard errors and predicted values derived from the model. It provides you with the means to define and manipulate your data, but the way you go about the job of modelling is not predetermined, and the user is left with maximum control over the model-fitting process. Second, S can be used for data exploration, in tabulating and sorting data, in drawing scatter plots to look for trends in your data, or to check visually for the presence of outliers. Third, it can be used as a sophisticated calculator to evaluate complex arithmetic expressions, and a very flexible and general object-orientated programming language to perform more extensive data manipulation. One of its great strengths is in the way in which it deals with vectors (lists of numbers). These may be combined in general expressions, involving arithmetic, relational and transformational operators such as sums, greater-than tests, logarithms or probability integrals. The ability to combine frequently-used sequences of commands into functions makes S a powerful programming language, ideally suited for tailoring one's specific statistical requirements. S is especially useful in handling difficult or unusual data sets, because its flexibility enables it to cope with such problems as unequal replication, missing values, non-orthogonal designs, and so on. Furthermore, the open-ended style of S is particularly appropriate for following through original ideas and developing new concepts. One of the great advantages of learning S is that the simple concepts that underlie it provide a unified framework for learning about statistical ideas in general. By viewing particular models in a general context, S highlights the fundamental similarities between statistical techniques and helps play down their superficial differences. As a commercial product S evolved into S-PLUS, but the problem was that S-PLUS was very expensive. In particular, it was much too expensive to be licensed for use in universities for teaching large numbers of students. In response to this, two New Zealand-based statisticians, Ross Ihaka and Robert Gentleman from the University of Auckland, decided to write a stripped-down version of S for teaching purposes. The letter R 'comes before S', so what would be more natural than for two authors whose first initial was 'R' to christen their creation R. The code for R was released in 1995 under a General Public License, and the core team was rapidly expanded to 15 members (they are listed on the website, below). Version 1.0.0 was released on 29 February 2000. This book is written using version 3.0.1, but all the code will run under earlier releases.

There is now a vast network of R users world-wide, exchanging functions with one another, and a vast resource of packages containing data and programs. There is a useful publication called *The R Journal* (formerly *R News*) that you can read at CRAN. Make sure that you cite the R Core Team when you use R in published work; you should cite them like this:

> R Core Team (2014). *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna. Available from http://www.r-project.org/.

R is an Open Source implementation and as such can be freely downloaded. If you type CRAN into your Google window you will find the site nearest to you from which to download it. Or you can go directly to

http://cran.r-project.org

The present book has its own website at

http://www.imperial.ac.uk/bio/research/crawley/statistics

Here you will find all the data files used in the text; you can download these to your hard disk and then run all of the examples described in the text. The executable statements are shown in the text in red Courier New font. There are files containing all the commands for each chapter, so you can paste the code directly into R instead of typing it from the book. There is a series of 12 fully-worked stand-alone practical sessions covering a wide range of statistical analyses. Learning R is not easy, but you will not regret investing the effort to master the basics.

M.J. Crawley
Ascot
April 2014

1

Fundamentals

The hardest part of any statistical work is getting started. And one of the hardest things about getting started is choosing the right kind of statistical analysis. The choice depends on the nature of your data and on the particular question you are trying to answer. The truth is that there is no substitute for experience: the way to know what to do is to have done it properly lots of times before.

The key is to understand what kind of *response* variable you have got, and to know the nature of your *explanatory* variables. The response variable is the thing you are working on: it is the variable whose variation you are attempting to understand. This is the variable that goes on the y axis of the graph (the ordinate). The explanatory variable goes on the x axis of the graph (the abscissa); you are interested in the extent to which variation in the response variable is associated with variation in the explanatory variable. A continuous measurement is a variable like height or weight that can take any real numbered value. A categorical variable is a *factor* with two or more *levels*: sex is a factor with two levels (male and female), and rainbow might be a factor with seven levels (red, orange, yellow, green, blue, indigo, violet).

It is essential, therefore, that you know:

- which of your variables is the response variable?
- which are the explanatory variables?
- are the explanatory variables continuous or categorical, or a mixture of both?
- what kind of response variable have you got – is it a continuous measurement, a count, a proportion, a time-at-death, or a category?

These simple keys will then lead you to the appropriate statistical method:

1. The explanatory variables (pick one of the rows):

(a) All explanatory variables continuous	*Regression*
(b) All explanatory variables categorical	*Analysis of variance (ANOVA)*
(c) Some explanatory variables continuous some categorical	*Analysis of covariance (ANCOVA)*

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

2. The response variable (pick one of the rows):

(a) Continuous	*Regression, ANOVA or ANCOVA*
(b) Proportion	*Logistic regression*
(c) Count	*Log linear models*
(d) Binary	*Binary logistic analysis*
(e) Time at death	*Survival analysis*

There is a small core of key ideas that need to be understood from the outset. We cover these here before getting into any detail about different kinds of statistical model.

Everything Varies

If you measure the same thing twice you will get two different answers. If you measure the same thing on different occasions you will get different answers because the thing will have aged. If you measure different individuals, they will differ for both genetic and environmental reasons (nature and nurture). Heterogeneity is universal: spatial heterogeneity means that places always differ, and temporal heterogeneity means that times always differ.

Because everything varies, finding that things vary is simply not interesting. We need a way of discriminating between variation that is scientifically interesting, and variation that just reflects background heterogeneity. That is why you need statistics. It is what this whole book is about.

The key concept is the amount of variation that we would expect to occur by chance alone, when nothing scientifically interesting was going on. If we measure bigger differences than we would expect by chance, we say that the result is statistically significant. If we measure no more variation than we might reasonably expect to occur by chance alone, then we say that our result is not statistically significant. It is important to understand that this is not to say that the result is not important. Non-significant differences in human life span between two drug treatments may be massively important (especially if you are the patient involved). Non-significant is not the same as ‘not different’. The lack of significance may be due simply to the fact that our replication is too low.

On the other hand, when nothing really *is* going on, then we want to know this. It makes life much simpler if we can be reasonably sure that there is no relationship between y and x. Some students think that ‘the only good result is a significant result’. They feel that their study has somehow failed if it shows that ‘A has no significant effect on B’. This is an understandable failing of human nature, but it is not good science. The point is that we want to know the truth, one way or the other. We should try not to care too much about the way things turn out. This is not an amoral stance, it just happens to be the way that science works best. Of course, it is hopelessly idealistic to pretend that this is the way that scientists really behave. Scientists often want passionately that a particular experimental result will turn out to be statistically significant, so that they can get a *Nature* paper and get promoted. But that does not make it right.

Significance

What do we mean when we say that a result is significant? The normal dictionary definitions of significant are 'having or conveying a meaning' or 'expressive; suggesting or implying deeper or unstated meaning'. But in statistics we mean something very specific indeed. We mean that 'a result was unlikely to have occurred by chance'. In particular, we mean 'unlikely to have occurred by chance if the null hypothesis was true'. So there are two elements to it: we need to be clear about what we mean by 'unlikely', and also what exactly we mean by the 'null hypothesis'. Statisticians have an agreed convention about what constitutes 'unlikely'. They say that an event is unlikely if it occurs less than 5% of the time. In general, the null hypothesis says that 'nothing is happening' and the alternative says that 'something *is* happening'.

Good and Bad Hypotheses

Karl Popper was the first to point out that a good hypothesis was one that was capable of *rejection*. He argued that *a good hypothesis is a falsifiable hypothesis*. Consider the following two assertions:

A. there are vultures in the local park

B. there are no vultures in the local park

Both involve the same essential idea, but one is refutable and the other is not. Ask yourself how you would refute option A. You go out into the park and you look for vultures. But you do not see any. Of course, this does not mean that there are none. They could have seen you coming, and hidden behind you. No matter how long or how hard you look, you cannot refute the hypothesis. All you can say is 'I went out and I didn't see any vultures'. One of the most important scientific notions is that *absence of evidence is not evidence of absence*.

Option B is fundamentally different. You reject hypothesis B the first time you see a vulture in the park. Until the time that you *do* see your first vulture in the park, you work on the assumption that the hypothesis is true. But if you see a vulture, the hypothesis is clearly false, so you reject it.

Null Hypotheses

The null hypothesis says 'nothing is happening'. For instance, when we are comparing two sample means, the null hypothesis is that the means of the two populations are the same. Of course, the two sample means are not identical, because everything varies. Again, when working with a graph of y against x in a regression study, the null hypothesis is that the slope of the relationship is zero (i.e. y is not a function of x, or y is independent of x). The essential point is that the null hypothesis is falsifiable. We reject the null hypothesis when our data show that the null hypothesis is sufficiently unlikely.

p Values

Here we encounter a much-misunderstood topic. The p value is *not* the probability that the null hypothesis is true, although you will often hear people saying this. In fact, p values are calculated *on the assumption that the null hypothesis is true*. It is correct to say that p values have to do with the plausibility of the null hypothesis, but in a rather subtle way.

As you will see later, we typically base our hypothesis testing on what are known as *test statistics*: you may have heard of some of these already (Student's t, Fisher's F and Pearson's chi-squared, for instance): p values are about the size of the test statistic. In particular, a p value is an estimate of the probability that a value of the test statistic, or a value more extreme than this, could have occurred by chance *when the null hypothesis is true*. Big values of the test statistic indicate that the null hypothesis is unlikely to be true. For sufficiently large values of the test statistic, we reject the null hypothesis and accept the alternative hypothesis.

Note also that saying 'we do not reject the null hypothesis' and 'the null hypothesis is true' are two quite different things. For instance, we may have failed to reject a false null hypothesis because our sample size was too low, or because our measurement error was too large. Thus, p values are interesting, but they do not tell the whole story: effect sizes and sample sizes are equally important in drawing conclusions. The modern practice is to state the p value rather than just to say 'we reject the null hypothesis'. That way, the reader can form their own judgement about the effect size and its associated uncertainty.

Interpretation

It should be clear by this point that we can make two kinds of mistakes in the interpretation of our statistical models:

- we can reject the null hypothesis when it is true
- we can accept the null hypothesis when it is false

These are referred to as *Type I* and *Type II* errors, respectively. Supposing we knew the true state of affairs (which, of course, we seldom do). Then in tabular form:

Null hypothesis	Actual situation	
	True	*False*
Accept	Correct decision	Type II
Reject	Type I	Correct decision

Model Choice

There are a great many models that we could fit to our data, and selecting which model to use involves considerable skill and experience. *All models are wrong, but some models are better than others.* Model choice is one of the most frequently ignored of the big issues involved in learning statistics.

In the past, elementary statistics was taught as a series of recipes that you followed without the need for any thought. This caused two big problems. People who were taught this way never realized that model choice is a really big deal ('I'm only trying to do a t test'). And they never understood that assumptions need to be checked ('all I need is the p value').

Throughout this book you are encouraged to learn the key assumptions. In order of importance, these are

- random sampling
- constant variance
- normal errors
- independent errors
- additive effects

Crucially, because these assumptions are often *not* met with the kinds of data that we encounter in practice, we need to know what to do about it. There are some things that it is much more difficult to do anything about (e.g. non-random sampling) than others (e.g. non-additive effects).

The book also encourages users to understand that in most cases there are literally hundreds of possible models, and that choosing the best model is an essential part of the process of statistical analysis. Which explanatory variables to include in your model, what transformation to apply to each variable, whether to include interaction terms: all of these are key issues that you need to resolve.

The issues are at their simplest with designed manipulative experiments in which there was thorough randomization and good levels of replication. The issues are most difficult with observational studies where there are large numbers of (possibly correlated) explanatory variables, little or no randomization and small numbers of data points. Much of your data is likely to come from the second category.

Statistical Modelling

The object is to determine the values of the parameters in a specific model that lead to *the best fit of the model to the data*. The data are sacrosanct, and they tell us what actually happened under a given set of circumstances. It is a common mistake to say 'the data were fitted to the model' as if the data were something flexible, and we had a clear picture of the structure of the model. On the contrary, what we are looking for is the minimal adequate model to describe the data. *The model is fitted to data*, not the other way around. The best model is the model that produces the least unexplained variation (the *minimal residual deviance*), subject to the constraint that the parameters in the model should all be statistically significant.

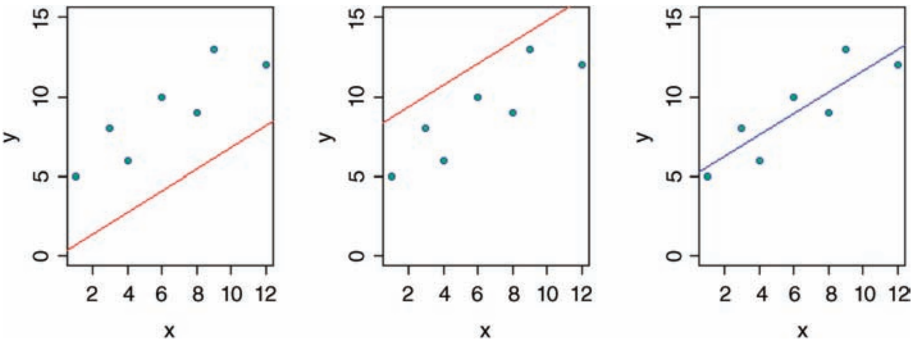
You have to specify the model. It embodies your mechanistic understanding of the factors involved, and of the way that they are related to the response variable. We want the model to be *minimal* because of the principle of parsimony, and *adequate* because there is no point in retaining an inadequate model that does not describe a significant fraction of the variation in the data. It is very important to understand that *there is not one model*; this is one of the common implicit errors involved in traditional regression and ANOVA, where the same models are used, often uncritically, over and over again. In most circumstances, there will be a large number of different, more or less plausible models that might be fitted to any given set of data. Part of the job of data analysis is to determine which, if any, of the possible models are adequate, and then, out of the set of adequate models, which is the minimal adequate model. In some cases there may be no single best model and a set of different models may all describe the data equally well (or equally poorly if the variability is great).

Maximum Likelihood

What, exactly, do we mean when we say that the parameter values should afford the 'best fit of the model to the data'? The convention we adopt is that our techniques should lead to *unbiased, variance minimizing estimators*. We define 'best' in terms of *maximum likelihood*. This notion is likely to be unfamiliar, so it is worth investing some time to get a feel for it. This is how it works:

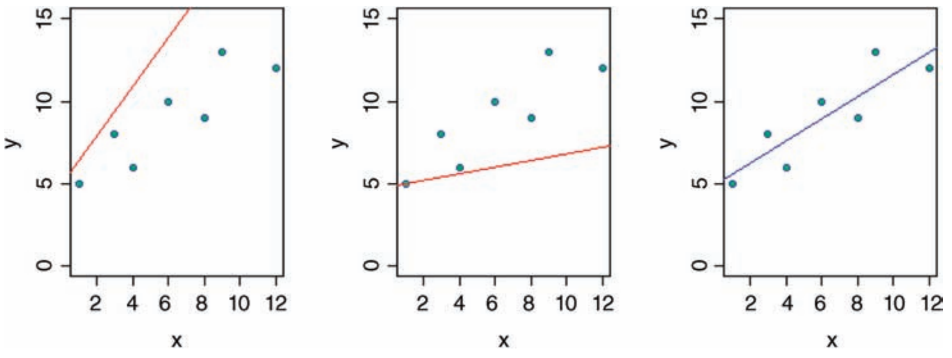
- given the data
- and given our choice of model
- what values of the parameters of that model
- make the observed data most likely?

Let us take a simple example from linear regression where the model we want to fit is $y = a + bx$ and we want the best possible estimates of the two parameters (the intercept a and the slope b) from the data in our scatterplot.



If the intercept were 0 (left-hand graph, above), would the data be likely? The answer of course, is no. If the intercept were 8 (centre graph) would the data be likely? Again, the answer is obviously no. The maximum likelihood estimate of the intercept is shown in the right-hand graph (its value turns out to be 4.827). Note that the point at which the graph cuts the y axis is *not* the intercept when (as here) you let R decide where to put the axes.

We could have a similar debate about the slope. Suppose we knew that the intercept was 4.827, then would the data be likely if the graph had a slope of 1.5 (left-hand graph, below)?



The answer, of course, is no. What about a slope of 0.2 (centre graph)? Again, the data are not at all likely if the graph has such a gentle slope. The maximum likelihood of the data given the model is obtained with a slope of 0.679 (right-hand graph).

This is not how the procedure is carried out in practice, but it makes the point that we judge the model on the basis *how likely the data would be if the model were correct*. When we do the analysis in earnest, both parameters are estimated simultaneously.

Experimental Design

There are only two key concepts:

- replication
- randomization

You replicate to increase reliability. You randomize to reduce bias. If you replicate thoroughly and randomize properly, you will not go far wrong.

There are a number of other issues whose mastery will increase the likelihood that you analyse your data the right way rather than the wrong way:

- the principle of parsimony
- the power of a statistical test
- controls
- spotting pseudoreplication and knowing what to do about it
- the difference between experimental and observational data (non-orthogonality)

It does not matter very much if you cannot do your own advanced statistical analysis. If your experiment is properly designed, you will often be able to find somebody to help you with the stats. But if your experiment is not properly designed, or not thoroughly randomized, or lacking adequate controls, then no matter how good you are at stats, some (or possibly even all) of your experimental effort will have been wasted. No amount of high-powered statistical analysis can turn a bad experiment into a good one. R is good, but not that good.

The Principle of Parsimony (Occam's Razor)

One of the most important themes running through this book concerns model simplification. The principle of parsimony is attributed to the fourteenth-century English nominalist philosopher William of Occam who insisted that, given a set of equally good explanations for a given phenomenon, then *the correct explanation is the simplest explanation*. It is called Occam's razor because he 'shaved' his explanations down to the bare minimum. In statistical modelling, the principle of parsimony means that:

- models should have as few parameters as possible
- linear models should be preferred to non-linear models
- experiments relying on few assumptions should be preferred to those relying on many
- models should be pared down until they are *minimal adequate*
- simple explanations should be preferred to complex explanations

The process of model simplification is an integral part of statistical analysis in R. In general, a variable is retained in the model only *if it causes a significant increase in deviance when it is removed from the current model*. Seek simplicity, then distrust it.

In our zeal for model simplification, we must be careful not to throw the baby out with the bathwater. Einstein made a characteristically subtle modification to Occam's razor. He said: 'A model should be as simple as possible. But no simpler.'

Observation, Theory and Experiment

There is no doubt that the best way to solve scientific problems is through a thoughtful blend of observation, theory and experiment. In most real situations, however, there are constraints on what can be done, and on the way things can be done, which mean that one or more of the trilogy has to be sacrificed. There are lots of cases, for example, where it is ethically or logistically impossible to carry out manipulative experiments. In these cases it is doubly important to ensure that the statistical analysis leads to conclusions that are as critical and as unambiguous as possible.

Controls

No controls, no conclusions.

Replication: It's the *n*s that Justify the Means

The requirement for replication arises because if we do the same thing to different individuals we are likely to get different responses. The causes of this heterogeneity in response are many and varied (genotype, age, sex, condition, history, substrate, microclimate, and so on). The object of replication is to increase the reliability of parameter estimates, and to allow us to quantify the variability that is found within the same treatment. To qualify as replicates, the repeated measurements:

- must be independent
- must not form part of a time series (data collected from the same place on successive occasions are not independent)
- must not be grouped together in one place (aggregating the replicates means that they are not spatially independent)
- must be measured at an appropriate spatial scale
- ideally, one replicate from each treatment ought to be grouped together into a block, and all treatments repeated in many different blocks.
- repeated measures (e.g. from the same individual or the same spatial location) are not replicates (this is probably the commonest cause of pseudoreplication in statistical work)

How Many Replicates?

The usual answer is 'as many as you can afford'. An alternative answer is 30. A very useful rule of thumb is this: a sample of 30 or more is a big sample, but a sample of less than 30 is a small sample. The rule doesn't always work, of course: 30 would be derisively small as a sample in an opinion poll, for instance. In other circumstances, it might be impossibly expensive to repeat an experiment as many as 30 times. Nevertheless, it is a rule of great practical utility, if only for giving you pause as you design your experiment with 300 replicates that perhaps this might really be a bit over the top. Or when you think you could get away with just five replicates this time.

There are ways of working out the replication necessary for testing a given hypothesis (these are explained below). Sometimes we know little or nothing about the variance of the response variable when we are planning an experiment. Experience is important. So are pilot studies. These should give an indication of the variance between initial units before the experimental treatments are applied, and also of the approximate magnitude of the responses to experimental treatment that are likely to occur. Sometimes it may be necessary to reduce the scope and complexity of the experiment, and to concentrate the inevitably limited resources of manpower and money on obtaining an unambiguous answer to a simpler question. It is immensely irritating to spend three years on a grand experiment, only to find at the end of it that the response is only significant at $p = 0.08$. A reduction in the number of treatments might well have allowed an increase in replication to the point where the same result would have been unambiguously significant.

Power

The power of a test is the probability of rejecting the null hypothesis when it is false. It has to do with Type II errors: β is the probability of accepting the null hypothesis when it is false. In an ideal world, we would obviously make β as small as possible. But there is a snag. The smaller we make the probability of committing a Type II error, the greater we make the probability of committing a Type I error, and rejecting the null hypothesis when, in fact, it is correct. A compromise is called for. Most statisticians work with $\alpha = 0.05$ and $\beta = 0.2$. Now the power of a test is defined as $1 - \beta = 0.8$ under the standard assumptions. This is used to calculate the sample sizes necessary to detect a specified difference when the error variance is known (or can be guessed at).

Let's think about the issues involved with power analysis in the context of a Student's *t*-test to compare two sample means. As explained on p. 91, the test statistic is $t =$ difference/ (the standard error of the difference) and we can rearrange the formula to obtain n, the sample size necessary in order that that a given difference, d, is statistically significant:

$$n = \frac{2s^2t^2}{d^2}$$

You can see that the larger the variance s^2, and the smaller the size of the difference, the bigger the sample we shall need. The value of the test statistic t depends on our decisions about Type I and Type II error rates (conventionally 0.05 and 0.2). For sample sizes of order 30, the t values associated with these probabilities are 1.96 and 0.84 respectively: these add to 2.80, and the square of 2.80 is 7.84. To the nearest whole number, the constants in the numerator evaluate to $2 \times 8 = 16$. So as a good rule of thumb, the sample size you need in each treatment is given by

$$n = \frac{16s^2}{d^2}$$

We simply need to work out 16 times the sample variance (obtained from the literature or from a small pilot experiment) and divide by the square of the difference that we want to be able to detect. So suppose that our current cereal yield is 10 t/ha with a standard deviation of sd = 2.8 t/ha (giving $s^2 = 7.84$) and we want to be able to say that a yield increase (delta) of 2 t/ha is significant at 95% with power = 80%, then we shall need to have $16 \times 7.84/4 = 31.36$ replicates in each treatment. The built in R function

```
power.t.test(delta=2,sd=2.8,power=0.8)
```

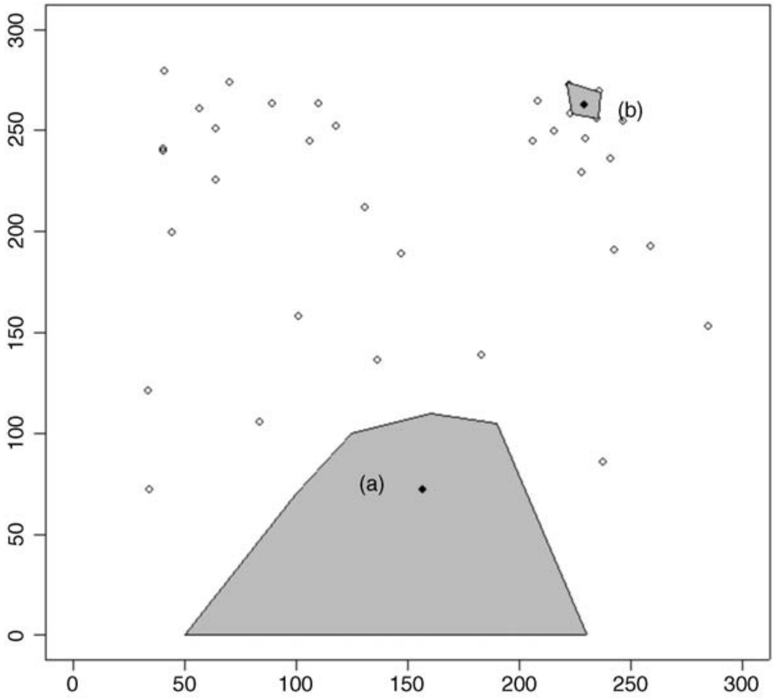
also gives $n = 32$ replicates per treatment on rounding-up.

Randomization

Randomization is something that everybody says they do, but hardly anybody does properly. Take a simple example. How do I select one tree from a forest of trees, on which to measure photosynthetic rates? I want to select the tree at random in order to avoid bias. For instance, I might be tempted to work on a tree that had accessible foliage near to the ground, or a tree that was close to the lab. Or a tree that looked healthy. Or a tree that had nice insect-free leaves. And so on. I leave it to you to list the biases that would be involved in estimating photosynthesis on any of those trees.

One common way of selecting a 'random' tree is to take a map of the forest and select a random pair of coordinates (say 157 m east of the reference point, and 228 m north). Then pace out these coordinates and, having arrived at that particular spot in the forest, select the nearest tree to those coordinates. But is this really a randomly selected tree?

If it *were* randomly selected, then it would have *exactly the same chance of being selected as every other* tree in the forest. Let us think about this. Look at the figure below, which shows a map of the distribution of trees on the ground. Even if they were originally planted out in regular rows, accidents, tree-falls, and heterogeneity in the substrate would soon lead to an aggregated spatial distribution of trees. Now ask yourself how many different random points would lead to the selection of a given tree. Start with tree (a). This will be selected by any points falling in the large shaded area.



Now consider tree (b). It will only be selected if the random point falls within the tiny area surrounding that tree. Tree (a) has a much greater chance of being selected than tree (b), and so *the nearest tree to a random point is not a randomly selected tree*. In a spatially heterogeneous woodland, isolated trees and trees on the edges of clumps will always have a higher probability of being picked than trees in the centre of clumps.

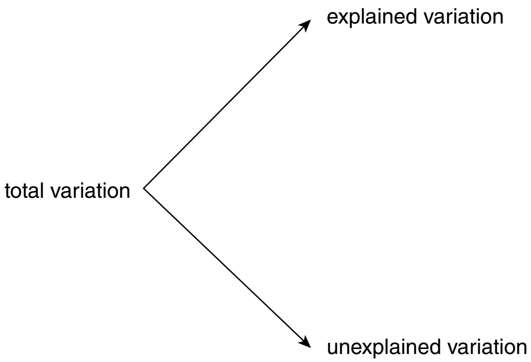
The answer is that to select a tree at random, every single tree in the forest must be numbered (all 24 683 of them, or whatever), and then a random number between 1 and 24 683 must be drawn out of a hat. There is no alternative. Anything less than that is not randomization.

Now ask yourself how often this is done in practice, and you will see what I mean when I say that randomization is a classic example of 'Do as I say, and not do as I do'. As an example of how important proper randomization can be, consider the following experiment that was designed to test the toxicity of five contact insecticides by exposing batches of flour beetles to the chemical on filter papers in Petri dishes. The animals walk about and pick up the poison on their feet. The *Tribolium* culture jar was inverted, flour and all, into a large tray, and beetles were collected as they emerged from the flour. The animals were allocated to the five chemicals in sequence; three replicate Petri dishes were treated with the first chemical, and 10 beetles were placed in each Petri dish. Do you see the source of bias in this procedure?

It is entirely plausible that flour beetles differ in their activity levels (sex differences, differences in body weight, age, etc.). The most active beetles might emerge first from the pile of flour. These beetles all end up in the treatment with the first insecticide. By the time we come to finding beetles for the last replicate of the fifth pesticide, we may be grubbing round in the centre of the pile, looking for the last remaining *Tribolium*. This matters, because the amount of pesticide picked by the beetles up will depend upon their activity levels. The more active the beetles, the more chemical they pick up on their feet, and the more likely they are to die. Thus, the failure to randomize will bias the result in favour of the first insecticide because this treatment received the most active beetles.

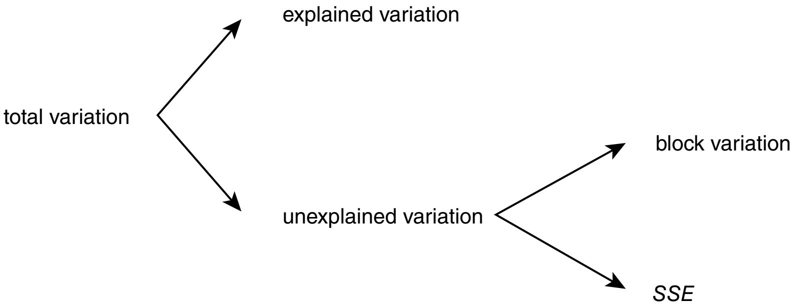
What we should have done is this. If we think that insect activity level is important in our experiment, then we should take this into account at the design stage. We might decide to have three levels of activity: active, average and sluggish. We fill the first five Petri dishes with 10 each of the active insects that emerge first from the pile. The next 50 insects we find go 10-at-a-time into five Petri dishes that are labelled average. Finally, we put last 50 insects to emerge into a set of five Petri dishes labelled sluggish. This procedure has created three *blocks* based on activity levels: we do not know precisely why the insects differed in their activity levels, but we think it might be important. Activity level is called a *random effect*: it is a factor with three levels. Next comes the randomization. We put the names of the five insecticides into a hat, shuffle them up, and draw them out one-at-a-time at random. The first Petri dish containing active beetles receives the insecticide that is first out of the hat, and so on until all five active Petri dishes have been allocated their own different pesticide. Then the five labels go back in the hat and are reshuffled. The procedure is repeated to allocate insecticide treatment at random to the five average activity Petri dishes. Finally, we put the labels back in the hat and draw the insecticide treatment for the five Petri dishes containing sluggish insects.

But why go to all this trouble? The answer is very important, and you should read it again and again until you understand it. The insects differ and the insecticides differ. But the Petri dishes may differ, too, especially if we store them in slightly different circumstances (e.g. near to the door of the controlled temperature cabinet or away at the back of the cabinet). The point is that there will be a total amount of variation in time to death across all the insects in the whole experiment (all 3 × 5 × 10 = 150 of them). We want to partition this variation into that which can be explained by differences between the insecticides and that which cannot.



If the amount of variation explained by differences between the insecticide treatments is large, then we conclude that the insecticides are significantly different from one another in their effects on mean age at death. We make this judgement on the basis of a comparison between the explained variation *SSA* and the unexplained variation *SSE*. If the unexplained variation is large, it is going to be very difficult to conclude anything about our *fixed effect* (insecticide in this case).

The great advantage of blocking is that it reduces the size of the unexplained variation. In our example, if activity level had a big effect on age at death (block variation), then the unexplained variation *SSE* would be much smaller than would have been the case if we had ignored activity and the significance of our fixed effect will be correspondingly higher:



The idea of good experimental design is to make *SSE* as small as possible, and blocking is the most effective way to bring this about.

R is very useful during the randomization stage because it has a function called `sample` which can shuffle the factor levels into a random sequence. Put the names of the five insecticides into a vector like this:

```
treatments <- c("aloprin","vitex","formixin","panto","allclear")
```

Then use `sample` to shuffle them for the active insects in dishes 1 to 5:

```
sample(treatments)
[1] "formixin" "panto"  "vitex"  "aloprin" "allclear"
```

then for the insects with average activity levels in dishes 6 to 10:

```
sample(treatments)
[1] "formixin" "allclear" "aloprin" "panto"  "vitex"
```

then finally for the sluggish ones in dishes 11 to 15:

```
sample(treatments)
[1] "panto"  "aloprin" "allclear" "vitex"  "formixin"
```

The recent trend towards ‘haphazard’ sampling is a cop-out. What it means is that ‘I admit that I didn’t randomize, but you have to take my word for it that this did not introduce any important biases’. You can draw your own conclusions.

Strong Inference

One of the most powerful means available to demonstrate the accuracy of an idea is an experimental confirmation of a prediction made by a carefully formulated hypothesis. There are two essential steps to the protocol of *strong inference* (Platt, 1964):

- formulate a clear hypothesis
- devise an acceptable test

Neither one is much good without the other. For example, the hypothesis should not lead to predictions that are likely to occur by other extrinsic means. Similarly, the test should demonstrate unequivocally whether the hypothesis is true or false.

A great many scientific experiments appear to be carried out with no particular hypothesis in mind at all, but simply to see what happens. While this approach may be commendable in the early stages of a study, such experiments tend to be weak as an end in themselves, because there will be such a large number of equally plausible explanations for the results. Without contemplation there will be no testable predictions; without testable predictions there will be no experimental ingenuity; without experimental ingenuity there is likely to be inadequate control; in short, equivocal interpretation. The results could be due to myriad plausible causes. Nature has no stake in being understood by scientists. We need to work at it. Without replication, randomization and good controls we shall make little progress.

Weak Inference

The phrase 'weak inference' is used (often disparagingly) to describe the interpretation of observational studies and the analysis of so-called 'natural experiments'. It is silly to be disparaging about these data, because they are often the only data that we have. The aim of good statistical analysis is to obtain the maximum information from a given set of data, *bearing the limitations of the data firmly in mind.*

Natural experiments arise when an event (often assumed to be an unusual event, but frequently without much justification of what constitutes unusualness) occurs that is like an experimental treatment (a hurricane blows down half of a forest block; a landslide creates a bare substrate; a stock market crash produces lots of suddenly poor people, etc.). 'The requirement of adequate knowledge of initial conditions has important implications for the validity of many natural experiments. Inasmuch as the "experiments" are recognized only when they are completed, or in progress at the earliest, it is impossible to be certain of the conditions that existed before such an "experiment" began. It then becomes necessary to make assumptions about these conditions, and any conclusions reached on the basis of natural experiments are thereby weakened to the point of being hypotheses, and they should be stated as such' (Hairston, 1989).

How Long to Go On?

Ideally, the duration of an experiment should be determined in advance, lest one falls prey to one of the twin temptations:

- to stop the experiment as soon as a pleasing result is obtained
- to keep going with the experiment until the 'right' result is achieved (the 'Gregor Mendel effect')

In practice, most experiments probably run for too short a period, because of the idiosyncrasies of scientific funding. This short-term work is particularly dangerous in medicine and the environmental sciences, because the kind of short-term dynamics exhibited after pulse experiments may be entirely different from the long-term dynamics of the same system. Only by long-term experiments of both the pulse and the press kind will the full range of dynamics be understood. The other great advantage of long-term experiments is that a wide range of patterns (e.g. 'kinds of years') is experienced.

Pseudoreplication

Pseudoreplication occurs when you analyse the data as if you had more degrees of freedom than you really have. There are two kinds of pseudoreplication:

- temporal pseudoreplication, involving repeated measurements from the same individual
- spatial pseudoreplication, involving several measurements taken from the same vicinity

Pseudoreplication is a problem because one of the most important assumptions of standard statistical analysis is *independence of errors*. Repeated measures through time on the same individual will have non-independent errors because peculiarities of the individual will be reflected in all of the measurements made on it (the repeated measures will be temporally correlated with one another). Samples taken from the same vicinity will have non-independent errors because peculiarities of the location will be common to all the samples (e.g. yields will all be high in a good patch and all be low in a bad patch).

Pseudoreplication is generally quite easy to spot. The question to ask is this. How many degrees of freedom for error does the experiment really have? If a field experiment appears to have lots of degrees of freedom, it is probably pseudoreplicated. Take an example from pest control of insects on plants. There are 20 plots, 10 sprayed and 10 unsprayed. Within each plot there are 50 plants. Each plant is measured five times during the growing season. Now this experiment generates $20 \times 50 \times 5 = 5000$ numbers. There are two spraying treatments, so there must be 1 degree of freedom for spraying and 4998 degrees of freedom for error. Or must there? Count up the replicates in this experiment. Repeated measurements on the same plants (the five sampling occasions) are certainly not replicates. The 50 individual plants within each quadrat are not replicates either. The reason for this is that conditions within each quadrat are quite likely to be unique, and so all 50 plants will experience more or less the same unique set of conditions, irrespective of the spraying treatment they receive. In fact, there are 10 replicates in this experiment. There are 10 sprayed plots and 10 unsprayed plots, and each plot will yield only one independent datum to the response variable (the proportion of leaf area consumed by insects, for example). Thus, there are 9 degrees of freedom within each treatment, and $2 \times 9 = 18$ degrees of freedom for error in the experiment as a whole. It is not difficult to find examples of pseudoreplication on this scale in the literature (Hurlbert, 1984). The problem is that it leads to the reporting of masses of spuriously significant results (with 4998 degrees of freedom for error, it is almost impossible *not* to have significant differences). The first skill to be acquired by the budding experimenter is the ability to plan an experiment that is properly replicated.

There are various things that you can do when your data are pseudoreplicated:

- average away the pseudoreplication and carry out your statistical analysis on the means
- carry out separate analyses for each time period
- use more advanced statistical techniques such as time series analysis or mixed effects models

Initial Conditions

Many otherwise excellent scientific experiments are spoiled by a lack of information about initial conditions. How can we know if something has changed if we do not know what it was like to begin with? It is often implicitly assumed that all the experimental units were alike at the beginning of the experiment, but this needs to be demonstrated rather than taken on faith. One of the most important uses of data on initial conditions is as a check on the efficiency of randomization. For example, you should be able to run your statistical analysis to demonstrate that the individual organisms were not significantly different in mean size at the beginning of a growth experiment. Without measurements of initial size, it is always possible to attribute the end result to differences in initial conditions. Another reason for measuring initial conditions is that the information can often be used to improve the resolution of the final analysis through analysis of covariance (see Chapter 9).

Orthogonal Designs and Non-Orthogonal Observational Data

The data in this book fall into two distinct categories. In the case of planned experiments, all of the treatment combinations are equally represented and, barring accidents, there will be no missing values. Such experiments are said to be *orthogonal*. In the case of observational studies, however, we have no control over the number of individuals for which we have data, or over the combinations of circumstances that are observed. Many of the explanatory variables are likely to be correlated with one another, as well as with the response variable. Missing treatment combinations will be commonplace, and such data are said to be non-orthogonal. This makes an important difference to our statistical modelling because, in orthogonal designs, the variability that is attributed to a given factor is constant, and does not depend upon the order in which that factor is removed from the model. In contrast, with non-orthogonal data, we find that the variability attributable to a given factor *does* depend upon the order in which the factor is removed from the model. We must be careful, therefore, to judge the significance of factors in non-orthogonal studies, when they are *removed from the maximal model* (i.e. from the model including all the other factors and interactions with which they might be confounded). Remember, *for non-orthogonal data, order matters.*

Aliasing

This topic causes concern because it manifests itself as one or more rows of NA appearing unexpectedly in the output of your model. Aliasing occurs when there is no information on which to base an estimate of a parameter value. Intrinsic aliasing occurs when it is due to the *structure of the model*. Extrinsic aliasing occurs when it is due to the *nature of the data*. Parameters can be aliased for one of two reasons:

- there are no data in the dataframe from which to estimate the parameter (e.g. missing values, partial designs or correlation amongst the explanatory variables)
- the model is structured in such a way that the parameter value cannot be estimated (e.g. over-specified models with more parameters than necessary)

If we had a factor with four levels (say none, light, medium and heavy use) then we could estimate four means from the data, one for each factor level. But the model looks like this:

$$y = \mu + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4$$

where the x_i are dummy variables having the value 0 or 1 for each factor level (see p. 158), the β_i are the effect sizes and μ is the overall mean. Clearly there is no point in having five parameters in the model if we can estimate only four independent terms from the data. One of the parameters must be intrinsically aliased. This topic is explained in detail in Chapter 11.

In a multiple regression analysis, if one of the continuous explanatory variables is perfectly correlated with another variable that has already been fitted to the data (perhaps because it is a constant multiple of the first variable), then the second term is aliased and adds nothing to the descriptive power of the model. Suppose that $x_2 = 0.5x_1$; then fitting a model with $x_1 + x_2$ will lead to x_2 being *intrinsically aliased* and given a parameter estimate of NA.

If all of the values of a particular explanatory variable are set to zero for a given level of a particular factor, then that level is said to have been *intentionally aliased*. This sort of aliasing is a useful programming trick during model simplification in ANCOVA when we wish a covariate to be fitted to some levels of a factor but not to others.

Finally, suppose that in a factorial experiment, all of the animals receiving level 2 of diet (factor A) and level 3 of temperature (factor B) have died accidentally as a result of attack by a fungal pathogen. This particular combination of diet and temperature contributes no data to the response variable, so the interaction term A(2):B(3) cannot be estimated. It is *extrinsically aliased*, and its parameter estimate is set to NA.

Multiple Comparisons

The thorny issue of multiple comparisons arises because when we do more than one test we are likely to find ‘false positives’ at an inflated rate (i.e. by rejecting a true null hypothesis more often than indicated by the value of α). The old fashioned approach was to use Bonferroni’s correction; in looking up a value for Student’s t, you divide your α value by the number of comparisons you have done. If the result is still significant then all is well, but it often will not be. Bonferroni’s correction is very harsh and will often throw out the baby with the bathwater. An old-fashioned alternative was to use Duncan’s multiple range tests (you may have seen these in old stats books, where lower-case letters were written at the head of each bar in a barplot: bars with different letters were significantly different, while bars with the same letter were not significantly different. The modern approach is to use contrasts wherever possible, and where it is essential to do multiple comparisons, then to use the wonderfully named Tukey's honestly significant differences (see ?TukeyHSD).

Summary of Statistical Models in R

Models are fitted to data (not the other way round), using one of the following model-fitting functions:

- lm: fits a linear model assuming normal errors and constant variance; generally this is used for regression analysis using continuous explanatory variables. The default output is summary.lm
- aov: an alternative to lm with summary.aov as the default output. Typically used only when there are complex error terms to be estimated (e.g. in split-plot designs where different treatments are applied to plots of different sizes)
- glm: fits generalized linear models to data using categorical or continuous explanatory variables, by specifying one of a family of *error structures* (e.g. Poisson for count data or binomial for proportion data) and a particular *link function*
- gam: fits generalized additive models to data with one of a family of error structures (e.g. Poisson for count data or binomial for proportion data) in which the continuous explanatory variables can (optionally) be fitted as arbitrary smoothed functions using non-parametric smoothers rather than specific parametric functions.
- lmer: fits linear mixed effects models with specified mixtures of fixed effects and random effects and allows for the specification of correlation structure amongst the explanatory variables and autocorrelation of the response variable (e.g. time series effects with repeated measures). The older lme is an alternative
- nls: fits a non-linear regression model via least squares, estimating the parameters of a specified non-linear function
- nlme: fits a specified non-linear function in a mixed effects model where the parameters of the non-linear function are assumed to be random effects; allows for the specification of correlation structure amongst the explanatory variables and autocorrelation of the response variable (e.g. time series effects with repeated measures).
- loess: fits a local regression model with one or more continuous explanatory variables using non-parametric techniques to produce a smoothed model surface
- rpart: fits a regression tree model using binary recursive partitioning whereby the data are successively split along coordinate axes of the explanatory variables so that at any node, the split is chosen that maximally distinguishes the response variable in the left and the right branches. With a categorical response variable, the tree is called a classification tree, and the model used for classification assumes that the response variable follows a multinomial distribution

For most of these models, a range of generic functions can be used to obtain information about the model. The most important and most frequently used are

`summary`	produces parameter estimates and standard errors from `lm`, and ANOVA tables from `aov`; this will often determine your choice between `lm` and `aov`. For either `lm` or `aov` you can choose `summary.aov` or `summary.lm` to get the alternative form of output (an ANOVA table or a table of parameter estimates and standard errors; see p. 158)
`plot`	produces diagnostic plots for model checking, including residuals against fitted values, influence tests, etc.
`anova`	a useful function for comparing two or more different models and producing ANOVA tables (and alternative to `AIC`)
`update`	used to modify the last model fit; it saves both typing effort and computing time

Other useful generics include:

`coef`	the coefficients (estimated parameters) from the model
`fitted`	the fitted values, predicted by the model for the values of the explanatory variables that appear in the data frame
`resid`	the residuals (the differences between measured and predicted values of *y*)
`predict`	uses information from the fitted model to produce smooth functions for plotting a curve through the scatterplot of your data. The trick is to realize that you need to provide values for all of the explanatory variables that are in the model (both continuous and categorical) as a list, and that the vectors of explanatory variables must all be exactly the same length (see p. 248 for a worked example). You can back-transform automatically using the option `type="response"`.

Organizing Your Work

There are three things that you really must keep for each separate R session:

- the *dataframe*, stored in a comma-delimited (.csv) or a tab-delimited (.txt) file
- the *script*, stored in a text file (.txt)
- the *results* obtained during this session (tables, graphs, model objects, etc.) stored in a PDF so that you can retain the graphics along with model outputs

To make sure you remember which data files and results go with which scripts, it is good practice to save the script, results and data files in the same, sensibly named folder.

Once the data are checked and edited, you are not likely ever to want to alter the data file. On the other hand, you are likely to want to keep a separate script for each working session of R. One of the great advantages of using scripts is that you can copy (potentially large) sections of code from previous successful sessions and save yourself a huge amount of typing (and wheel reinvention).

There are two sensible ways of working and you should choose the one that suits you best. The first is to write all of your code in a script editor, save it regularly, and pepper it liberally with comments (use the hatch symbol to start each comment):

```
# this is a comment
```

When you make mistakes, cut them out of the script, taking care not to delete important bits of code accidentally.

The alternative is to save the script of the whole session just before you finish. This is stored in what R calls the `history` file. At the end of the session, type

```
history(Inf)
```

and R will open a script window called 'R History' containing a full transcript of all the commands (both right and wrong) that you entered during that session. Copy all the material to a text file, edit out the mistakes (again, making sure that you do not remove any essential lines of code), add the necessary comments, then save the text file as your script for the session in its customized directory along with the data file and the results (the output of tables, models and graphics).

Whichever method you use, the saved script is a permanent record of what you did (with comments pointing out exactly why you did it). You are likely to copy and paste the code into R on future occasions when you want to do similar analyses, and you want the code to work seamlessly (which it will not do if you have unintentionally removed key lines of code).

It is a bad idea to create your scripts in a word processor because several of the symbols you will use may not be readable within R. Double quotes is a classic example of this; your word processor will have " (open quotes) and " (close quotes) but R will read only `"` (simple quotes). However, you might want to save the *results* from your R sessions in a word processor because this can include graphs as well as input and output in the same document.

Housekeeping within R

The simplest way to work is to start a new R session for each separate activity. The advantage of working this way is that things from one session do not get mixed up with things from another session.

The classic thing to go wrong is that you get two different objects with the same name, and you do not know which is which. For instance, a variable called *x* from one analysis may contain 30 numbers and a different variable called *x* from another analysis might have

50 numbers in it. At least, in that case, you can test the length of the object to see which one it is (if it is of length 50 then it must be the *x* variable from the second analysis). Worse problems arise when the two different variables called *x* are both the same length. Then you really do not know where you are.

If you insist on doing several things during the same R session, then it pays to be really well organized. In this book we `attach` dataframes, so that you can refer to your variables by name without reference to the name of the dataframe from which they come (experts generally do not use `attach`). The disadvantage of using `attach` is that you might have several dataframes attached that contain exactly the same variable name. R will warn you of this by writing

```
The following object is masked from first.frame:

    temp, wind
```

when you attach a dataframe containing variables that are already attached from a different dataframe. The message means that when you attached a new dataframe, it contained two variables, called `temp` and `wind` respectively, that were already attached from a previous dataframe called `first.frame`. This state of affairs is confusing and unsatisfactory. The way to avoid it is to make sure that you `detach` all unnecessary dataframes before attaching a new dataframe. Here is the problem situation in full:

```
first.frame <- read.csv("c:\\temp\\test.pollute.csv")
second.frame <- read.csv("c:\\temp\\ozone.data.csv")
attach(first.frame)
attach(second.frame)
```

```
The following object is masked from first.frame:

    temp, wind
```

Here is how to avoid the problem

```
first.frame <- read.csv("c:\\temp\\test.pollute.csv")
second.frame <- read.csv("c:\\temp\\ozone.data.csv")
attach(first.frame)
```

. . . this is where you work on the information from `first.frame`. Then when you are finished . . .

```
detach(first.frame)
attach(second.frame)
```

No warning message is printed because `temp` and `rain` are no longer duplicate variable names.

The other big problem arises when you create variables during a session by allocation (this typically involves calculation or the use of data-generating functions within R to produce random numbers, for instance, or sequences). So if in the first session you wanted x to be $\sqrt{2}$ then you would put:

```
x <- sqrt(2)
```

Now, in a later session you use x for the axis of a graph, and give it the sequence of values 0 to 10:

```
x <- 0:10
```

If the fact that you had done this slipped your mind, then you might later use x thinking that is was the single number $\sqrt{2}$. But R knows it to be the vector of 11 numbers 0 to 10, and this could have seriously bad consequences. The way to avoid problems like this is to remove all the variables you have calculated before you start on another project during the same session of R. The function for this is `rm` (or `remove`)

```
rm(x)
```

If you ask for a variable to be removed that does not exist, then R will warn you of this fact:

```
rm(y,z)

Warning messages:
1: In rm(y, z) : object 'y' not found
2: In rm(y, z) : object 'z' not found
```

We are now in a position to start using R in earnest. The first thing to learn is how to structure a dataframe and how to read a dataframe into R. It is immensely irritating that this first step often turns out to be so difficult for beginners to get right. Once the data are into R, the rest is plain sailing.

References

Hairston, N.G. (1989) *Ecological Experiments: Purpose, Design and Execution*, Cambridge University Press, Cambridge.

Hurlbert, S.H. (1984) Pseudoreplication and the design of ecological field experiments. *Ecological Monographs*, **54**, 187–211.

Platt, J.R. (1964) Strong inference. *Science*, **146**, 347–353.

Further Reading

Atkinson, A.C. (1985) *Plots, Transformations, and Regression*, Clarendon Press, Oxford.

Box, G.E.P., Hunter, W.G. and Hunter, J.S. (1978) *Statistics for Experimenters: An Introduction to Design, Data Analysis and Model Building*, John Wiley & Sons, New York.

Chambers, J.M., Cleveland, W.S., Kleiner, B. and Tukey, P.A. (1983) *Graphical Methods for Data Analysis*, Wadsworth, Belmont, CA.

Winer, B.J., Brown, D.R. and Michels, K.M. (1991) *Statistical Principles in Experimental Design*, McGraw-Hill, New York.

2

Dataframes

Learning how to handle your data, how to enter it into the computer, and how to read the data into R are amongst the most important topics you will need to master. R handles data in objects known as dataframes. A dataframe is an object with rows and columns (a bit like a two-dimensional matrix). The rows contain different observations from your study, or measurements from your experiment. The columns contain the values of different variables. The values in the body of the dataframe can be numbers (as they would be in as matrix), but they could also be text (e.g. the names of factor levels for categorical variables, like 'male' or 'female' in a variable called 'gender'), they could be calendar dates (like 23/5/04), or they could be logical variables (like 'TRUE' or 'FALSE'). Here is a spreadsheet in the form of a dataframe with seven variables, the leftmost of which comprises the row names, and other variables are numeric (Area, Slope, Soil pH and Worm density), categorical (Field Name and Vegetation) or logical (Damp is either true = T or false = F).

Field Name	Area	Slope	Vegetation	Soil pH	Damp	Worm density
Nash's Field	3.6	11	Grassland	4.1	F	4
Silwood Bottom	5.1	2	Arable	5.2	F	7
Nursery Field	2.8	3	Grassland	4.3	F	2
Rush Meadow	2.4	5	Meadow	4.9	T	5
Gunness' Thicket	3.8	0	Scrub	4.2	F	6
Oak Mead	3.1	2	Grassland	3.9	F	2
Church Field	3.5	3	Grassland	4.2	F	3
Ashurst	2.1	0	Arable	4.8	F	4
The Orchard	1.9	0	Orchard	5.7	F	9
Rookery Slope	1.5	4	Grassland	5	T	7
Garden Wood	2.9	10	Scrub	5.2	F	8
North Gravel	3.3	1	Grassland	4.1	F	1
South Gravel	3.7	2	Grassland	4	F	2
Observatory Ridge	1.8	6	Grassland	3.8	F	0
Pond Field	4.1	0	Meadow	5	T	6
Water Meadow	3.9	0	Meadow	4.9	T	8

(*continued*)

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

(*Continued*)

Field Name	Area	Slope	Vegetation	Soil pH	Damp	Worm density
Cheapside	2.2	8	Scrub	4.7	T	4
Pound Hill	4.4	2	Arable	4.5	F	5
Gravel Pit	2.9	1	Grassland	3.5	F	1
Farm Wood	0.8	10	Scrub	5.1	T	3

Perhaps the most important thing about analysing your own data properly is getting your dataframe absolutely right. The expectation is that you will have used a spreadsheet like Excel to enter and edit the data, and that you will have used plots to check for errors. The thing that takes some practice is learning exactly how to put your numbers into the spreadsheet. There are countless ways of doing it wrong, but only one way of doing it right. And this way is not the way that most people find intuitively to be the most obvious.

The key thing is this: *all the values of the same variable must go in the same column*. It does not sound like much, but this is what people tend to get wrong. If you had an experiment with three treatments (control, pre-heated and pre-chilled), and four measurements per treatment, it might seem like a good idea to create the spreadsheet like this:

Control	Preheated	Prechilled
6.1	6.3	7.1
5.9	6.2	8.2
5.8	5.8	7.3
5.4	6.3	6.9

But this is not a dataframe, because values of the response variable appear in three different columns, rather than all in the same column. The correct way to enter these data is to have two columns: one for the response variable and one for the levels of the experimental factor (control, pre-heated and pre-chilled). Here are the same data, entered correctly as a dataframe:

Response	Treatment
6.1	control
5.9	control
5.8	control
5.4	control
6.3	preheated
6.2	preheated
5.8	preheated
6.3	preheated
7.1	prechilled
8.2	prechilled
7.3	prechilled
6.9	prechilled

A good way to practice this layout is to use the Excel function called PivotTable (found under the Insert tab on the main menu bar) on your own data: it requires your spreadsheet to be in the form of a dataframe, with each of the explanatory variables in its own column.

Once you have made your dataframe in Excel and corrected all the inevitable data entry and spelling errors, then you need to save the dataframe in a file format that can be read by R. Much the simplest way is to save all your dataframes from Excel as comma-delimited files: File/Save As/ . . . then from the 'Save as type' options choose 'CSV (Comma delimited)'. There is no need to add a suffix, because Excel will automatically add '.csv' to your file name. This file can then be read into R directly as a dataframe, using the `read.csv` function.

Think of a name for the dataframe (say, 'worms' in this case). Now use the *gets arrow* <– which is a composite symbol made up of the two characters < (less than) and – (minus) like this

```
worms <- read.csv("c:\\temp\\worms.csv")
```

To see which variables are included in this dataframe, we use the `names` function:

```
names(worms)

[1]  "Field.Name"  "Area"  "Slope"         "Vegetation"
[5]  "Soil.pH"     "Damp"  "Worm.density"
```

In order that we can refer to the variable names directly (without prefixing them by the dataframe name) we `attach` the dataframe:

```
attach(worms)
```

To see the contents of the dataframe, just type its name:

```
worms

           Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
1         Nashs.Field  3.6    11  Grassland     4.1 FALSE            4
2      Silwood.Bottom  5.1     2     Arable     5.2 FALSE            7
3       Nursery.Field  2.8     3  Grassland     4.3 FALSE            2
4         Rush.Meadow  2.4     5     Meadow     4.9  TRUE            5
5     Gunness.Thicket  3.8     0      Scrub     4.2 FALSE            6
6            Oak.Mead  3.1     2  Grassland     3.9 FALSE            2
7        Church.Field  3.5     3  Grassland     4.2 FALSE            3
8             Ashurst  2.1     0     Arable     4.8 FALSE            4
9         The.Orchard  1.9     0    Orchard     5.7 FALSE            9
10      Rookery.Slope  1.5     4  Grassland     5.0  TRUE            7
11        Garden.Wood  2.9    10      Scrub     5.2 FALSE            8
12       North.Gravel  3.3     1  Grassland     4.1 FALSE            1
13       South.Gravel  3.7     2  Grassland     4.0 FALSE            2
14 Observatory.Ridge  1.8     6  Grassland     3.8 FALSE            0
15         Pond.Field  4.1     0     Meadow     5.0  TRUE            6
16       Water.Meadow  3.9     0     Meadow     4.9  TRUE            8
```

```
17         Cheapside  2.2      8      Scrub      4.7  TRUE           4
18        Pound.Hill  4.4      2    Arable      4.5 FALSE           5
19        Gravel.Pit  2.9      1  Grassland      3.5 FALSE           1
20         Farm.Wood  0.8     10      Scrub      5.1  TRUE           3
```

The variable names appear in row number 1. Notice that R has expanded our abbreviated T and F into TRUE and FALSE.

Selecting Parts of a Dataframe: Subscripts

We often want to extract part of a dataframe. This is a very general procedure in R, accomplished using what are called subscripts. You can think of subscripts as addresses within a vector, a matrix or a dataframe. Subscripts in R appear within square brackets, thus `y[7]` is the 7th element of the vector called *y* and `z[2,6]` is the 2nd row of the 6th column of a two-dimensional matrix called *z*. This is in contrast to arguments to functions in R, which appear in round brackets `(4,7)`.

We might want to select all the rows of a dataframe for certain specified columns. Or we might want to select all the columns for certain specified rows of the dataframe. The convention in R is that when we do not specify any subscript, then all the rows, or all the columns is assumed. This syntax is difficult to understand on first acquaintance, but `[,` ‘blank then comma’ means ‘all the rows’ and `,]` ‘comma then blank’ means all the columns. For instance, to select the first column of the dataframe, use subscript `[,1]`. To select groups of columns we provide several column numbers. Thus, to select all the rows of the first three columns of `worms`, we write:

```
worms[,1:3]
```

```
           Field.Name  Area  Slope
1         Nashs.Field   3.6     11
2      Silwood.Bottom   5.1      2
3       Nursery.Field   2.8      3
4         Rush.Meadow   2.4      5
5     Gunness.Thicket   3.8      0
6            Oak.Mead   3.1      2
7        Church.Field   3.5      3
8            Ashurst   2.1      0
9         The.Orchard   1.9      0
10      Rookery.Slope   1.5      4
11        Garden.Wood   2.9     10
12       North.Gravel   3.3      1
13       South.Gravel   3.7      2
14 Observatory.Ridge   1.8      6
15         Pond.Field   4.1      0
16       Water.Meadow   3.9      0
17          Cheapside   2.2      8
18         Pound.Hill   4.4      2
19         Gravel.Pit   2.9      1
20          Farm.Wood   0.8     10
```

To select just the middle 11 rows for all the columns of the dataframe, use subscript `[5:15,]` like this:

```
worms[5:15,]
```

```
          Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
5    Gunness.Thicket  3.8     0      Scrub     4.2 FALSE            6
6           Oak.Mead  3.1     2  Grassland     3.9 FALSE            2
7       Church.Field  3.5     3  Grassland     4.2 FALSE            3
8            Ashurst  2.1     0     Arable     4.8 FALSE            4
9        The.Orchard  1.9     0    Orchard     5.7 FALSE            9
10     Rookery.Slope  1.5     4  Grassland     5.0  TRUE            7
11       Garden.Wood  2.9    10      Scrub     5.2 FALSE            8
12      North.Gravel  3.3     1  Grassland     4.1 FALSE            1
13      South.Gravel  3.7     2  Grassland     4.0 FALSE            2
14 Observatory.Ridge  1.8     6  Grassland     3.8 FALSE            0
15        Pond.Field  4.1     0     Meadow     5.0  TRUE            6
```

It is often useful to select certain rows, based on *logical tests* on the values of one or more variables. Here is the code to select only those rows which have `Area` > `3` and `Slope` < `3` using 'comma then blank' to specify all the columns like this:

```
worms[Area>3 & Slope <3,]
```

```
        Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
2   Silwood.Bottom  5.1     2     Arable     5.2 FALSE            7
5  Gunness.Thicket  3.8     0      Scrub     4.2 FALSE            6
6         Oak.Mead  3.1     2  Grassland     3.9 FALSE            2
12    North.Gravel  3.3     1  Grassland     4.1 FALSE            1
13    South.Gravel  3.7     2  Grassland     4.0 FALSE            2
15      Pond.Field  4.1     0     Meadow     5.0  TRUE            6
16    Water.Meadow  3.9     0     Meadow     4.9  TRUE            8
18      Pound.Hill  4.4     2     Arable     4.5 FALSE            5
```

Sorting

You can sort the rows or the columns of the dataframe in any way you choose, but typically you will want to see all of the columns and to sort on the basis of the values in one or more of the columns. By default, things in R are sorted into ascending order (i.e. into alphabetical order for character data, and increasing numeric order for numbers). The simplest way to sort is to use the name of the variable. Suppose we want the whole dataframe sorted by Area:

```
worms[order(Area),]
```

```
          Field.Name Area Slope Vegetation Soil.pH  Damp Worm.density
20         Farm.Wood  0.8    10      Scrub     5.1  TRUE            3
10     Rookery.Slope  1.5     4  Grassland     5.0  TRUE            7
14 Observatory.Ridge  1.8     6  Grassland     3.8 FALSE            0
9        The.Orchard  1.9     0    Orchard     5.7 FALSE            9
8            Ashurst  2.1     0     Arable     4.8 FALSE            4
```

```
17         Cheapside  2.2     8      Scrub    4.7  TRUE            4
4        Rush.Meadow  2.4     5     Meadow    4.9  TRUE            5
3      Nursery.Field  2.8     3  Grassland    4.3 FALSE            2
11       Garden.Wood  2.9    10      Scrub    5.2 FALSE            8
19        Gravel.Pit  2.9     1  Grassland    3.5 FALSE            1
6           Oak.Mead  3.1     2  Grassland    3.9 FALSE            2
12      North.Gravel  3.3     1  Grassland    4.1 FALSE            1
7       Church.Field  3.5     3  Grassland    4.2 FALSE            3
1        Nashs.Field  3.6    11  Grassland    4.1 FALSE            4
13      South.Gravel  3.7     2  Grassland    4.0 FALSE            2
5    Gunness.Thicket  3.8     0      Scrub    4.2 FALSE            6
16      Water.Meadow  3.9     0     Meadow    4.9  TRUE            8
15        Pond.Field  4.1     0     Meadow    5.0  TRUE            6
18        Pound.Hill  4.4     2     Arable    4.5 FALSE            5
2     Silwood.Bottom  5.1     2     Arable    5.2 FALSE            7
```

The key point to note is that `order(Area)` comes before the comma, and there is a blank after the comma (which means that we want all of the columns). The original row numbers appear on the left of the sorted dataframe.

Now suppose we just want the columns containing numeric information to appear in the output; these are column numbers 2, 3, 5 and 7:

```
worms[order(Area),c(2,3,5,7)]
```

```
       Area   Slope   Soil.pH   Worm.density
20      0.8      10       5.1              3
10      1.5       4       5.0              7
14      1.8       6       3.8              0
9       1.9       0       5.7              9
8       2.1       0       4.8              4
17      2.2       8       4.7              4
4       2.4       5       4.9              5
3       2.8       3       4.3              2
11      2.9      10       5.2              8
19      2.9       1       3.5              1
6       3.1       2       3.9              2
12      3.3       1       4.1              1
7       3.5       3       4.2              3
1       3.6      11       4.1              4
13      3.7       2       4.0              2
5       3.8       0       4.2              6
16      3.9       0       4.9              8
15      4.1       0       5.0              6
18      4.4       2       4.5              5
2       5.1       2       5.2              7
```

To sort things into descending order we employ the reverse function `rev` like this:

```
worms[rev(order(worms[,5])),c(5,7)]
```

```
      Soil.pH    Worm.density
9         5.7               9
11        5.2               8
2         5.2               7
20        5.1               3
15        5.0               6
10        5.0               7
16        4.9               8
4         4.9               5
8         4.8               4
17        4.7               4
18        4.5               5
3         4.3               2
7         4.2               3
5         4.2               6
12        4.1               1
1         4.1               4
13        4.0               2
6         3.9               2
14        3.8               0
19        3.5               1
```

which sorts into descending order by Soil pH, with only Soil pH and Worm density as output (because of the `c(5,7)`). This makes the point that you can specify the sorting variable either by name (as we did with `Area` above) or by column number (as we did with `Soil.pH` here by specifying column number 5).

Summarizing the Content of Dataframes

The object called `worms` now has all the attributes of a dataframe. For example, you can summarize it, using `summary`:

```
summary(worms)
```

```
   Field.Name           Area             Slope          Vegetation
 Ashurst      : 1   Min.   :0.800    Min.   : 0.00    Arable    :3
 Cheapside    : 1   1st Qu.:2.175    1st Qu.: 0.75    Grassland:9
 Church.Field: 1    Median :3.000    Median : 2.00    Meadow    :3
 Farm.Wood    : 1   Mean   :2.990    Mean   : 3.50    Orchard   :1
 Garden.Wood  : 1   3rd Qu.:3.725    3rd Qu.: 5.25    Scrub     :4
 Gravel.Pit   : 1   Max.   :5.100    Max.   :11.00
 (Other)      :14

    Soil.pH            Damp          Worm.density
 Min.   :3.500    Mode  :logical    Min.   :0.00
 1st Qu.:4.100    FALSE:14          1st Qu.:2.00
 Median :4.600    TRUE  :6          Median :4.00
 Mean   :4.555    NA's  :0          Mean   :4.35
 3rd Qu.:5.000                      3rd Qu.:6.25
 Max.   :5.700                      Max.   :9.00
```

Values of continuous variables are summarized under six headings: one parametric (the arithmetic mean) and five non-parametric (maximum, minimum, median, 25th percentile or first quartile, and 75th percentile or third quartile). Levels of categorical variables are counted.

Summarizing by Explanatory Variables

The function you need to master for summarizing quantitative information in dataframes is called `aggregate`. You will often want to know the average values of continuous variables within the dataframe summarized by factor levels from one or more of the categorical variables. For instance, we may want to know the mean number of worms under different plant communities. For a single response variable like `Worm.density`, you can use `tapply` and `with` like this:

```
with(worms,tapply(Worm.density,Vegetation,mean))

   Arable    Grassland   Meadow    Orchard    Scrub
     5.33         2.44     6.33       9.00     5.25
```

In the early stages of analysis, however, we often want to see the mean values of all the continuous variables summarized at the same time, and this is where `aggregate` comes into its own. We need to do a little bit of work in advance, by noting down the column numbers that contain the variables for which it would be useful and sensible to calculate the mean values (i.e. the columns containing real numbers). These are `Area` and `Slope` in columns 2 and 3 respectively, `Soil.pH` in column 5 and `Worm.density` in column 7. To get their mean values classified by plant communities we need only type:

```
aggregate(worms[,c(2,3,5,7)],list(Vegetation),mean)

    Group.1  Area  Slope  Soil.pH  Worm.density
1    Arable  3.87   1.33     4.83          5.33
2 Grassland  2.91   3.67     4.10          2.44
3    Meadow  3.47   1.67     4.93          6.33
4   Orchard  1.90   0.00     5.70          9.00
5     Scrub  2.42   7.00     4.80          5.25
```

which causes all of the mean values to be printed. Do you know why there is a comma after the left hand square bracket on worms? Note that the column containing the levels of Vegetation is headed `Group.1`. This is the default used by `aggregate`. Within the column, the levels of Vegetation appear in alphabetical order. To get the column headed by ‘Community’ instead of `Group.1` we annotate the list like this:

```
aggregate(worms[,c(2,3,5,7)],list(Community=Vegetation),mean)

  Community  Area  Slope  Soil.pH  Worm.density
1    Arable  3.87   1.33     4.83          5.33
2 Grassland  2.91   3.67     4.10          2.44
3    Meadow  3.47   1.67     4.93          6.33
4   Orchard  1.90   0.00     5.70          9.00
5     Scrub  2.42   7.00     4.80          5.25
```

You can do multiple classifications using two or more categorical explanatory variables: here is a summary that asks for the mean values separately for each level of soil moisture within each vegetation type

```
aggregate(worms[,c(2,3,5,7)],
                  list(Moisture=Damp,Community=Vegetation),mean)
```

```
   Moisture   Community    Area   Slope   Soil.pH   Worm.density
1     FALSE      Arable    3.87    1.33      4.83           5.33
2     FALSE   Grassland    3.09    3.62      3.99           1.88
3      TRUE   Grassland    1.50    4.00      5.00           7.00
4      TRUE      Meadow    3.47    1.67      4.93           6.33
5     FALSE     Orchard    1.90    0.00      5.70           9.00
6     FALSE       Scrub    3.35    5.00      4.70           7.00
7      TRUE       Scrub    1.50    9.00      4.90           3.50
```

You will notice that aggregate produces only those rows for which there is output (there are no dry meadows, and no wet arable or wet orchards, for instance). This is in contrast to tapply, which produces NA for missing combinations:

```
with(worms,tapply(Slope,list(Damp,Vegetation),mean))
```

```
       Arable  Grassland  Meadow  Orchard  Scrub
FALSE    1.33       3.62      NA        0      5
TRUE       NA       4.00    1.67       NA      9
```

You choose between aggregate and tapply on the basis of which of them serves your needs best in a particular case, bearing in mind that tapply can summarize only one variable at a time.

The answer to the quiz (above) is that the comma after the square bracket means 'select all of the rows in the dataframe'. If we wanted only some of the rows, then we would need to specify *which* rows, just like we specified which columns we wanted using c(2,3,5,7).

First Things First: Get to Know Your Data

Once the data are in the computer, the temptation is to rush straight into statistical analysis. This is exactly the wrong thing to do. You need to get to know your data first. This is particularly important early on in a project, because there is a very high chance that the data contain mistakes. Obviously, these need to be put right before anything sensible can be done.

Just as important, if you do not know what your data look like, then you will not know what model to select to fit to the data (e.g. a straight line or a curved line), or whether the assumptions of your intended model are met by the data (e.g. constancy of variance and normality of errors).

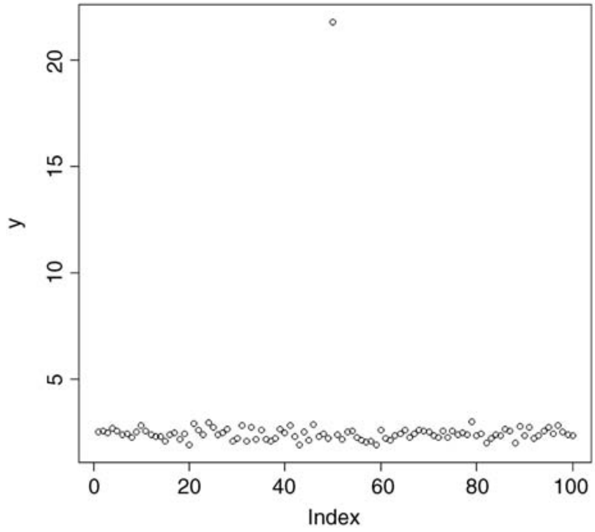
The recommended procedure is this. First, just plot the response variable on its own. This is called an index plot. It will show if there are any glaring errors in the data and whether there are trends or cycles in the values of the response as they appear in the dataframe.

```
data <- read.csv("c:\\temp\\das.csv")
attach(data)
head(data)
```

```
         y
1  2.514542
2  2.559668
3  2.460061
4  2.702720
5  2.571997
6  2.412833
```

Data inspection could not be simpler. Just plot the values of *y* using plot(y). This produces a scatterplot with the values of *y* appearing from left to right in the order in which they appear in the dataframe:

```
plot(y)
```



One data point sticks out like a sore thumb. We need to go back to the lab notebook and check what has happened. It is useful to know what line of the dataframe contains the unusually large value of *y*. This is easy to work out using the which function. Inspection of the plot shows that our questionable data point is the only one larger than 10, so we can use the which function like this:

```
which(y > 10)
[1] 50
```

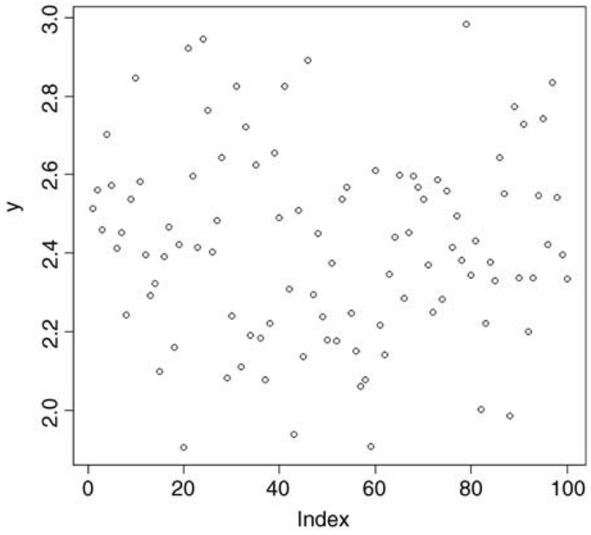
so the outlying data point is on line 50 of the spreadsheet. Now we need to go back to the lab book and try to work out what happened. What is the exact value of the outlier? We use subscripts [square brackets] for this. We want the 50th value of *y*:

```
y[50]
```

```
[1] 21.79386
```

The lab book shows that the value should be 2.179386. What has happened is that a typing error has put the decimal point in the wrong place. Clearly we should change the entry in the spreadsheet and start again. Now when we use `plot` to check the data, it looks like this

```
plot(y)
```



There is no reason to suspect any of the data points. The data are not trended with the order of entry, and the degree of scatter looks to be consistent from left to right. That is good news.

It is important to appreciate that outliers are not necessarily mistakes. Most outliers will be genuine values of the response. The key point about outliers is that you need to be aware of their existence (the plots will show this) and to understand how influential the outliers are in determining effect sizes and standard errors in your chosen model. This crucial point is discussed on p. 135. Now do the same thing for each of the continuous explanatory variables in turn.

To check for mistakes in the factor levels of categorical explanatory variables, you use the `table` function. This shows how many times each factor level appears in a particular column of the data frame. Mistakes will be highlighted because there will be more factor levels than there should be. Here is a variable called `treatment` from an experiment on fertilizer application and plant growth. There are four levels of the factor: control, nitrogen, phosphorus and both N and P. This is how we check the data:

```
yields <- read.csv("c:\\temp\\fertyield.csv")
attach(yields)
head(yields)
```

```
   treatment      yield
1    control  0.8274156
2    control  3.6126275
3    control  2.6192581
4    control  1.7412190
5    control  0.6590589
6    control  0.4891107
```

As you can see, the variable called treatment is a factor with text (‘control’ on these six lines) entries to describe which of the four treatments was applied to obtain the yield in column 2. This is how we check the factor levels:

```
table(treatment)
```

```
variable
both N and P    control    nitogen    nitrogen    phosphorus
          10         10          1           9            10
```

There are five factor levels printed rather than the four we expected, so it is clear that something is wrong. What has happened is that one of the nitrogen values has been misspelled as `nitogen` with the result that there are only 9 nitrogen values (not the 10 we expect) and an extra column in the table with one entry for the spelling mistake. The next step is to find out which row the mistake occurs in, then armed with this information, go back to the original spreadsheet and correct the error. This is easy using the `which` function. Note the use of ‘double equals’ to check for equality):

```
which(treatment == "nitogen")
```

```
[1] 11
```

The mistake is in line number 11. We need to correct that line in the spreadsheet. Make the change then start again, by reading the new corrected dataframe into R. Check all of your variables, continuous and categorical, one at a time and correct all of the mistakes you find. Now it is time to look at relationships between variables.

```
detach(yields)
```

Relationships

The place to start is with pairwise relationships. When both variables are continuous, the appropriate graph is a scatterplot:

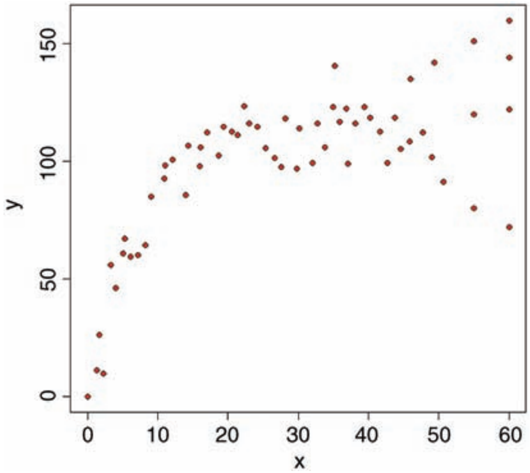
```
data <- read.csv("c:\\temp\\scatter.csv")
attach(data)
head(data)
```

```
          x          y
1  0.000000    0.00000
2  5.112000   61.04000
3  1.320000   11.11130
```

```
4  35.240000  140.65000
5   1.632931   26.15218
6   2.297635   10.00100
```

The response variable is *y* and the explanatory variable is *x*, so we write `plot(x,y)` or `plot(y~x)` to see the relationship (the two forms of `plot` produce the same graph; the choice of which to use is entirely up to you). We introduce two new features to customize the plot: changing the plotting symbol (`pch` stands for 'plotting character') from the default open circles we have been using so far to coloured discs with a black edge (`pch = 21`) and select a bright colour for the fill of the disc (the 'background' as it is called in R, using `bg="red"`):

```
plot(x,y,pch=21,bg="red")
```



This plot immediately alerts us to two important issues: (1) the relationship between the response and the explanatory variable is curved, not a straight line; and (2) the degree of scatter of the response variable increases from left to right (this is what non-constant variance (heteroscedasticity) looks like). These two features of the data are not mistakes, but they are very important elements in model selection (despite the positive correlation between *x* and *y*, we would not do a linear regression on these data, for instance).

When the explanatory variable is categorical the `plot` function produces a box-and-whisker plot. This is very useful for error-checking, as the following example illustrates:

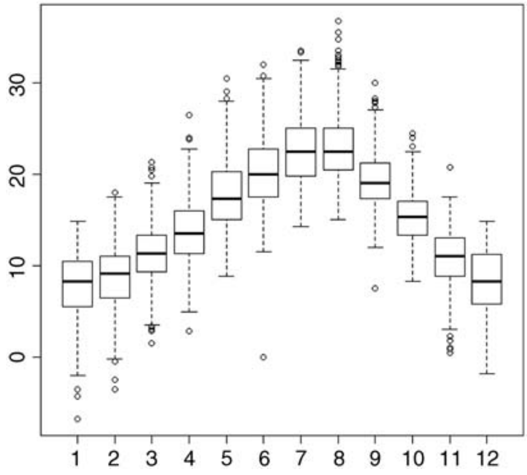
```
data <- read.csv("c:\\temp\\weather.data.csv")
attach(data)
head(data)
```

```
   upper  lower   rain  month    yr
1   10.8    6.5   12.2      1  1987
2   10.5    4.5    1.3      1  1987
3    7.5   -1.0    0.1      1  1987
```

```
4    6.5    -3.3    1.1       1   1987
5   10.0     5.0    3.5       1   1987
6    8.0     3.0    0.1       1   1987
```

There are three potential continuous response variables in this dataframe (daily maximum temperature 'upper' in degrees Celsius, daily minimum temperature 'lower' in degrees Celsius, and daily total precipitation 'rain' in millimetres) and two quantitative variables that we might choose to define as categorical (month and year). Here, we carry out an initial inspection of the maximum temperature data. Note that when the `plot` function is invoked with a categorical explanatory variable – `factor(month)` in this case – then R produces a box-and-whisker plot rather than a scatterplot (see p. 67 for details).

```
plot(factor(month),upper)
```



The box-and- whisker plot shows the very clear seasonal pattern of median temperature, peaking in July and August and reaching a minimum in January. The details of what the boxes and whiskers mean are explained on p. 161. For our present purposes we concentrate on error-checking.

The plot shows a freezing day (0 maximum) in June (month = 6), which is unheard of at this location. It turns out that the thermometer broke on this day and was replaced by a new one. The missing value for the day of the breakdown was foolishly entered as a zero (it should have been NA). Again, we go back to the spreadsheet and replace the erroneous 0 by the correct NA (this stands for not available; notice that NA is not enclosed in quotation marks).

Looking for Interactions between Continuous Variables

Once the obvious errors have been corrected, the next questions concern model choice. For instance, does the response to one variable depend upon the level of another variable? (in the jargon, this is known as a statistical interaction). With continuous explanatory variables, we can look for interaction effects using *conditioning plots* (typically known as coplots).

With categorical explanatory variables, we can look for interaction effects using barplots. Here we have one response variable (*y*) and two continuous explanatory variables (*x* and *z*):

```
data <- read.csv("c:\\temp\\coplot.csv")
attach(data)
head(data)
```

```
          x         y          z
1  95.73429  107.8087  14.324408
2  36.20660  223.9257  10.190577
3  28.71378  245.2523  12.566815
4  78.36956  132.7344  13.084384
5  38.37717  222.2966   9.960033
6  57.18078  184.8372  10.035677
```

Two scatterplots side by side look better if we change the shape of the plotting window from the default square (7 × 7 inches) to a rectangle (7 × 4 inches) like this:

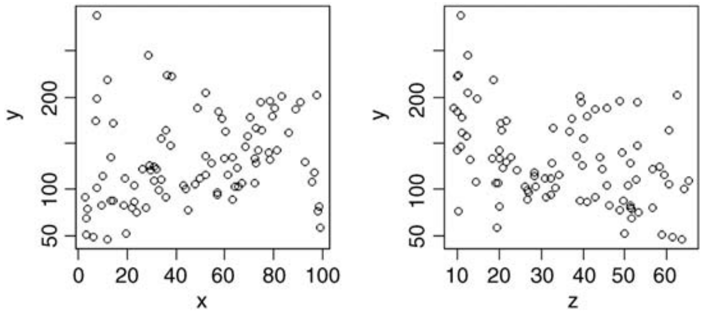
```
windows(7,4)
```

then alter the graphics parameter to specify two sets of axis on the same row (see p. 134 for details):

```
par(mfrow=c(1,2))
```

There is no clear relationship between the response and either of the two continuous explanatory variables when they are fitted on their own:

```
plot(x,y)
plot(z,y)
```

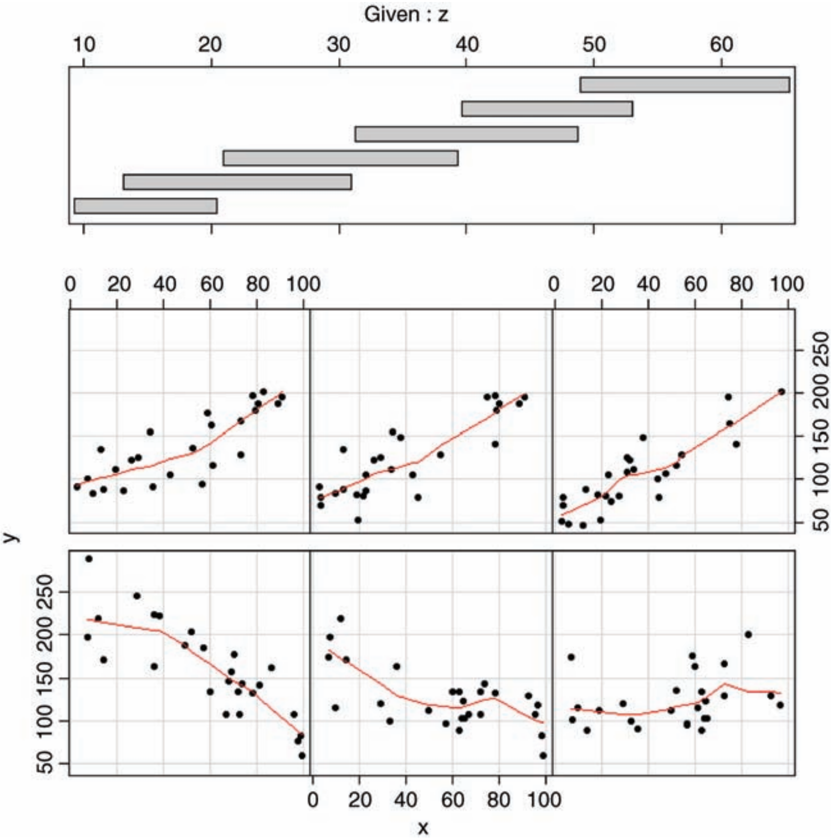


To look for interactions between continuous explanatory variables (like *x* and *z* in this example) we employ a superb graphical function called `coplot`. The function plots *y* against *x* conditional on the value of *z*. The function is very simple to use. The symbol that might be new to you is the vertical bar | which is read as ‘given’. The syntax says ‘plot *y* against *x* given the value of *z*’ and is written as `plot(y~x|z)`. The default is to split the data into six graphs (but you can change this if you want to). The sixth of the data with the lowest values of z appear in the bottom left-hand panel as a plot of y against x. To improve the appearance of the plot we can use solid black plotting symbols (`pch =16`) and fit a trend line through the scatterplot in red (this is called a non-parametric smoother, and is invoked by the function `panel.smooth`). First we return to the default window shape (7 × 7 inches):

```
windows(7,7)
```

then we draw the conditioning plot:

```
coplot(y~x|z,pch=16,panel=panel.smooth)
```



This shows a really clear relationship between the response variable and x, but the form of the relationship depends on the value of z (the second of our continuous explanatory variables). For low values of z (in the bottom left-hand panel) the relationship between y and x is strongly negative. For high values of z (top right) the relationship is strongly positive. As z increases (from bottom left to bottom right, then from top left to top right), the slope of the relationship between y and x increases rather smoothly from large negative values to zero then to increasingly positive values. Only `coplot` can show this kind of interaction so simply and clearly.

The top part of the figure can be confusing for beginners. The shaded horizontal bars are called shingles (after the American word for roof tiles) and they show the range of values of the variable (*z* in this example) used to produce each of the six panel plots. The bottom (left-hand) shingle shows that the bottom left-hand panel (with the strong negative relationship between *y* and *x*) was based on data selected to have *z* values between 10 and 20 (look at the scale on the top axis beneath ‘Given: z’). The next panel used data with *z* values between 13 and 30, the next between 20 and 40, and so on. The shingles overlap because that is the default setting (see `?coplot` for details of `overlap=0.5`): for a non-edge plot, half of the data points are shared with the panel to its left, and half are shared with the panel on its right. You can specify non-overlapping panels if that is what you want (`overlap = 0`).

Graphics to Help with Multiple Regression

The problems become most acute when we have many continuous explanatory variables and the data come from observational studies where we have no control over replication or randomization. In data sets like this, the explanatory variables are often correlated with each other (most simple models assume that the explanatory variables are independent – orthogonal in the jargon). We discuss these issues in detail in Chapter 10, but at this stage we simply observe that there are no easy cures for this. Two very useful tools for preliminary investigation of multiple regression data are *tree models* and *generalized additive models*, as illustrated on p. 197.

Interactions Involving Categorical Variables

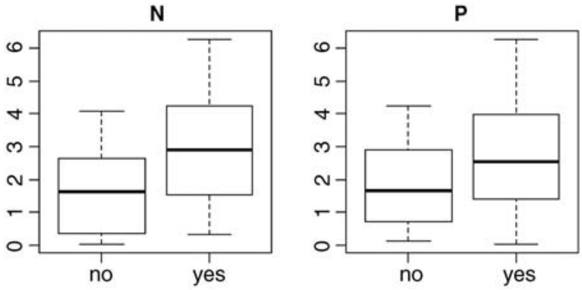
The following data are from a factorial experiment involving nitrogen and phosphorus fertilizers applied separately and in combination:

```
data <- read.csv("c:\\temp\\np.csv")
attach(data)
head(data)
```

```
      yield  nitrogen  phosphorus
1  0.8274156        no          no
2  3.6126275        no          no
3  2.6192581        no          no
4  1.7412190        no          no
5  0.6590589        no          no
6  0.4891107        no          no
```

There is one continuous response variable (yield) and two categorical explanatory variables (nitrogen and phosphorus) each with two levels (yes and no, meaning the fertilizer was or was not applied to the plot in question). First we look at the effects of the two nutrients separately:

```
windows(7,4)
par(mfrow=c(1,2))
plot(nitrogen,yield,main="N")
plot(phosphorus,yield,main="P")
```



These plots show what are called the ‘main effects’ of nitrogen and phosphorus: it looks as if nitrogen increases yield slightly more than does phosphorus. The median for plus nitrogen (‘yes’ in the left-hand plot) is above the top of the box for no nitrogen, whereas the median for plus phosphorus is below the top of the box for no phosphorus (right-hand plot). What these main effects fail to show us is whether the response to phosphorus depends on the level of nitrogen. What we need is an *interaction plot* showing effect sizes for four levels of response: neither nutrient, just nitrogen, just phosphorus, or both N and P. We use the function tapply to do this:

```
tapply(yield,list(nitrogen,phosphorus),mean)

           no        yes
no    1.47384   1.875928
yes   2.28999   3.480184
```

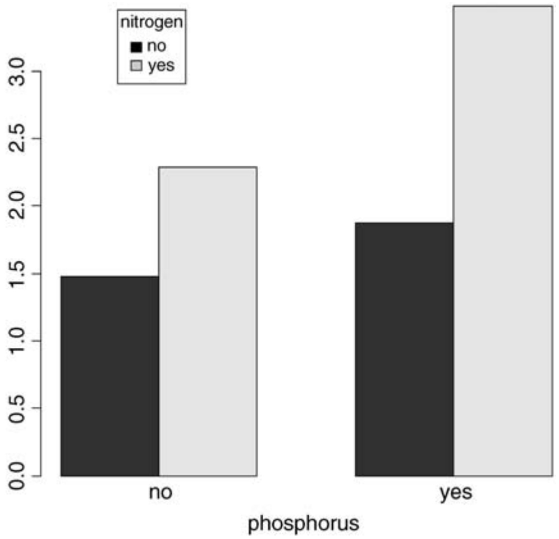
The rows refer to nitrogen and the columns to phosphorus. With no P (left column), the effect size for N is 2.290/1.474 = 1.55, but with P the effect size of N is 3.480/1.876 = 1.86. The effect size for nitrogen depends on the level of phosphorus (55% increase in yield without P, but 86% increase with P). That is an example of a statistical interaction: *the response to one factor depends upon the level of another factor.*

We need a way of showing interactions graphically. There are many way of doing this, but perhaps the most visually effective is to use the barplot function. We can use the output from tapply directly for this, but it is a good idea to add a legend to show the nitrogen treatments associated using two colours of shading:

```
barplot(tapply(yield,list(nitrogen,phosphorus),mean),
                                     beside=TRUE,xlab="phosphorus")
```

The locator function allows you to put the legend in a place where it does not interfere with any of the bars. Put the cursor where you want *the top left corner* of the legend box to appear, then left click:

```
legend(locator(1),legend=c("no","yes"),title="nitrogen",
                                       fill=c("black","lightgrey"))
```



For your final presentation, you would want to add error bars to the plot, but we shall deal with this later, once we have discussed how to measure the unreliability of effects (see p. 162).

It is essential to spend time on understanding the patterns in your data *before* you embark on the statistics. These preliminary plots are not optional extras. They are absolutely vital for deciding what kind of statistical modelling will be appropriate, and what kinds of assumptions about the data (linearity of response, constancy of variance, normality of errors) are likely to be justified. As you will see, we return to check the assumptions again, once the statistical modelling has been carried out (see p. 134).

That concludes the initial data inspection. Now we can begin to think about statistical analysis of the data. We shall concentrate on measuring effect sizes and their unreliabilities (the modern approach) and pay relatively little attention to hypothesis testing (the old-fashioned approach).

Further Reading

Chambers, J.M. and Hastie, T.J. (1992) *Statistical Models in S*, Wadsworth & Brooks/Cole, Pacific Grove, CA.

Crawley, M.J. (2013) *The R Book*, 2nd edn, John Wiley & Sons, Chichester.

3

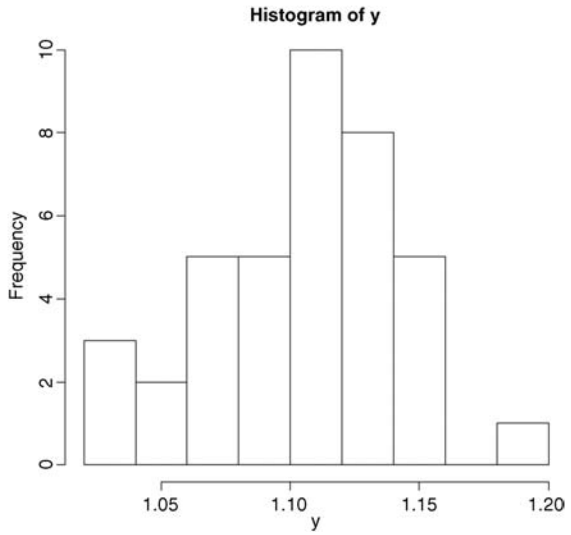
Central Tendency

Despite the fact that everything varies, measurements often cluster around certain intermediate values; this attribute is called *central tendency*. Even if the data themselves do not show much tendency to cluster round some central value, the parameters derived from repeated experiments (e.g. replicated sample means) almost inevitably do (this is called the central limit theorem; see p. 70). We need some data to work with. The data are in a vector called y stored in a text file called yvalues

```
yvals <- read.csv("c:\\temp\\yvalues.csv")
attach(yvals)
```

So how should we quantify central tendency? Perhaps the most obvious way is just by looking at the data, without doing any calculations at all. The data values that occur most frequently are called the *mode*, and we discover the value of the mode simply by drawing a histogram of the data like this:

```
hist(y)
```



Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

So we would say that the modal class of *y* was between 1.10 and 1.12 (we will see how to control the location of the break points in a histogram later).

The most straightforward quantitative measure of central tendency is the *arithmetic mean* of the data. This is the sum of all the data values $\sum y$ divided by the number of data values, *n*. The capital Greek sigma $\sum$ just means 'add up all the values' of what follows; in this case, a set of *y* values. So if we call the arithmetic mean '*y* bar', $\bar{y}$, we can write

$$\bar{y} = \frac{\sum y}{n}$$

The formula shows how we would write a general function to calculate arithmetic means for any vector of *y* values. First, we need to add them up. We could do it like this:

```
y[1] + y[2] + y[3] + . . . + y[n]
```

but that is very long-winded, and it supposes that we know the value of *n* in advance. Fortunately, there is a built-in function called `sum` that works out the total for any length of vector, so

```
total <- sum(y)
```

gives us the value for the numerator. But what about the number of data values? This is likely to vary from application to application. We could print out the *y* values and count them, but that is tedious and error-prone. There is a very important, general function in R to work this out for us. The function is called `length(y)` and it returns *the number of numbers* in the vector called *y*:

```
n <- length(y)
```

So our recipe for calculating the arithmetic mean would be `ybar <- total/n`.

```
ybar <- total/n
[1] 1.103464
```

There is no need to calculate the intermediate values, *total* and *n*, so it would be more efficient to write `ybar <- sum(y)/length(y)`. To put this logic into a general function we need to pick a name for the function, say `arithmetic.mean,` then define it as follows:

```
arithmetic.mean <- function(x) sum(x)/length(x)
```

Notice two things: we do not assign the answer `sum(x)/length(x)` to a variable name like `ybar`; and the name of the vector `x` used inside the `function(x)` may be different from the names on which we might want to use the function in future (`y,` for instance). If you type the name of a function on its own, you get a listing of the contents:

```
arithmetic.mean
function(x) sum(x)/length(x)
```

Now we can test the function on some data. First we use a simple data set where we know the answer already, so that we can check that the function works properly: say

```
data <- c(3,4,6,7)
```

where we can see immediately that the arithmetic mean is 5.

```
arithmetic.mean(data)
[1] 5
```

So that's OK. Now we can try it on a realistically big data set:

```
arithmetic.mean(y)
[1] 1.103464
```

You will not be surprised to learn that R has a built-in function for calculating arithmetic means directly, and again, not surprisingly, it is called `mean`. It works in the same way as did our home-made function:

```
mean(y)
[1] 1.103464
```

Arithmetic mean is not the only quantitative measure of central tendency, and in fact it has some rather unfortunate properties. Perhaps the most serious failing of the arithmetic mean is that it is *highly sensitive to outliers*. Just a single extremely large or extremely small value in the data set will have a big effect on the value of the arithmetic mean. We shall return to this issue later, but our next measure of central tendency does not suffer from being sensitive to outliers. It is called the *median*, and is the 'middle value' in the data set. To write a function to work out the median, the first thing we need to do is sort the data into ascending order:

```
sorted <- sort(y)
```

Now we just need to find the middle value. There is slight hitch here, because if the vector contains an even number of numbers, then there *is* no middle value. Let us start with the easy case where the vector contains an odd number of numbers. The number of numbers in the vector is given by `length(y)`and the middle value is half of this:

```
length(y)/2
[1] 19.5
```

So the median value is the 20th value in the sorted data set. To extract the median value of *y* we need to use 20 as a subscript, not 19.5, so we should convert the value of `length(y)/2` into an integer. We use `ceiling` ('the smallest integer greater than') for this:

```
ceiling(length(y)/2)
[1] 20
```

So now we can extract the median value of *y*:

```
sorted[20]
```

```
[1] 1.108847
```

or, more generally:

```
sorted[ceiling(length(y)/2)]
```

```
[1] 1.108847
```

or even more generally, omitting the intermediate variable called `sorted`:

```
sort(y)[ceiling(length(y)/2)]
```

```
[1] 1.108847
```

But what about the case where the vector contains an even number of numbers? Let us manufacture such a vector, by dropping the first element from our vector called *y* using negative subscripts like this:

```
y.even <- y[-1]
length(y.even)
```

```
[1] 38
```

The logic is that we shall work out the arithmetic average of the two values of *y* on either side of the middle; in this case, the average of the 19th and 20th sorted values:

```
sort(y.even)[19]
```

```
[1] 1.108847
```

```
sort(y.even)[20]
```

```
[1] 1.108853
```

So in this case, the median would be:

```
(sort(y.even)[19]+sort(y.even)[20])/2
```

```
[1] 1.10885
```

But to make it general we need to replace the 19 and 20 by

```
ceiling(length(y.even)/2)
```

and

```
ceiling(1+length(y.even)/2)
```

respectively.

The question now arises as to how we know, in general, whether the vector *y* contains an odd or an even number of numbers, so that we can decide which of the two methods to use. The trick here is to use ‘modulo’. This is the remainder (the amount ‘left over’) when one integer is divided by another. An even number has modulo 0 when divided by 2, and an odd number has modulo 1. The modulo function in R is `%%` (two successive percent symbols) and it is used where you would use slash `/` to carry out a regular division. You can see this in action with an even number, 38, and odd number, 39:

```
38%%2

[1] 0

39%%2

[1] 1
```

Now we have all the tools we need to write a general function to calculate medians. Let us call the function `med` and define it like this:

```
med <- function(x) {
    modulo <- length(x)%%2
    if (modulo == 0) (sort(x)[ceiling(length(x)/2)]+sort(x)[ceiling
(1+length(x)/2)])/2
    else sort(x)[ceiling(length(x)/2)]
}
```

Notice that when the `if` statement is true (i.e. we have an even number of numbers) then the expression immediately following the `if` statement is evaluated (this is the code for calculating the median with an even number of numbers). When the `if` statement is false (i.e. we have an odd number of numbers, and `modulo == 1`) then the expression following the `else` statement is evaluated (this is the code for calculating the median with an odd number of numbers). Let us try it out, first with the odd-numbered vector *y*, then with the even-numbered vector *y.even*, to check against the values we obtained earlier.

```
med(y)

[1] 1.108847

med(y.even)

[1] 1.10885
```

Both of these check out. Again, you will not be surprised that there is a built-in function for calculating medians, and helpfully it is called `median`:

```
median(y)

[1] 1.108847

median(y.even)

[1] 1.10885
```

For processes that change multiplicatively rather than additively, then neither the arithmetic mean nor the median is an ideal measure of central tendency. Under these conditions, the appropriate measure is the *geometric mean*. The formal definition of this is somewhat abstract: the geometric mean is the *n*th root of the product of the data. If we use capital Greek pi ($\prod$) to represent multiplication, and *y* 'hat' ($\hat{y}$) to represent the geometric mean, then

$$\hat{y} = \sqrt[n]{\prod y}$$

Let us take a simple example we can work out by hand: the numbers of insects on five different plants were as follows: 10, 1, 1000, 1, 10. Multiplying the numbers together gives 100 000. There are five numbers, so we want the fifth root of this. Roots are hard to do in your head, so we will use R as a calculator. Remember that *roots are fractional powers*, so the fifth root is a number raised to the power $1/5 = 0.2$. In R, powers are denoted by the `^` symbol (the caret), which is found above number 6 on the keyboard:

```
100000^0.2
```

```
[1] 10
```

So the geometric mean of these insect numbers is 10 insects per stem. Note that two of the five counts were exactly this number, so it seems a reasonable estimate of central tendency. The arithmetic mean, on the other hand, is hopeless for these data, because the large value (1000) is so influential: $10 + 1 + 1000 + 1 + 10 = 1022$ and $1022/5 = 204.4$. Note that none of the counts were close to 204.4, so the arithmetic mean is a poor estimate of central tendency in this case.

```
insects <- c(1,10,1000,10,1)
mean(insects)
```

```
[1] 204.4
```

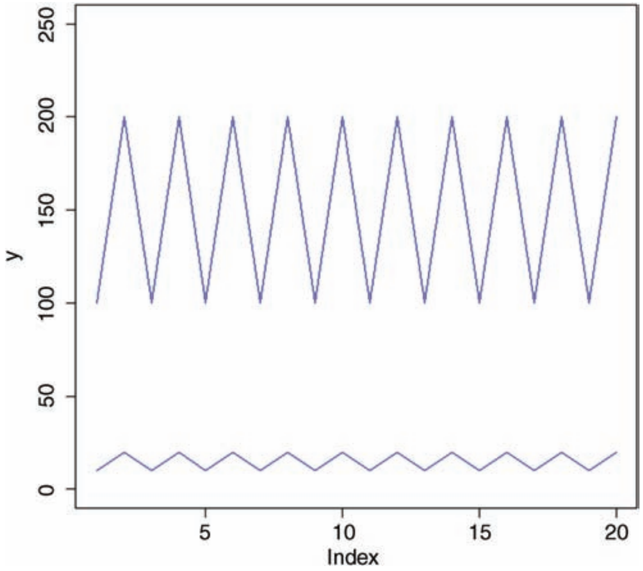
Another way to calculate geometric mean involves the use of logarithms. Recall that to multiply numbers together we add up their logarithms. And to take roots, we divide the logarithm by the root. So we should be able to calculate a geometric mean by finding the *antilog* (`exp`) *of the average of the logarithms* (`log`) of the data:

```
exp(mean(log(insects)))
```

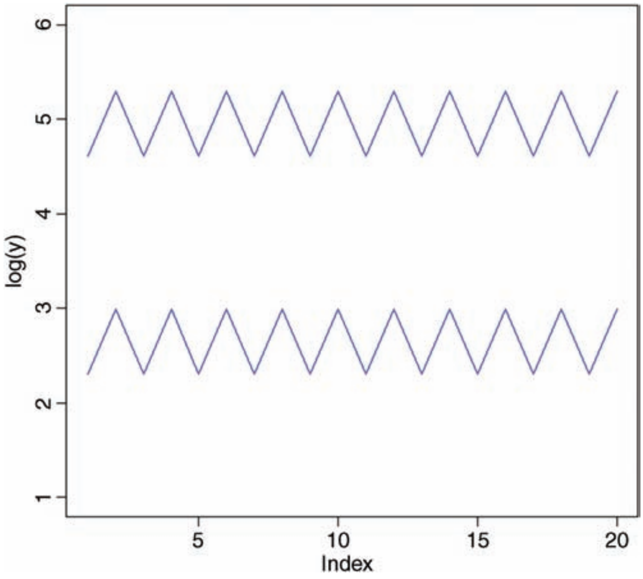
```
[1] 10
```

Writing a general function to compute geometric means is left to you as an exercise.

The use of geometric means draws attention to a general scientific issue. Look at the figure below, which shows numbers varying through time in two populations. Now ask yourself 'Which population is the more variable?' Chances are, you will pick the upper line:



But now look at the scale on the *y* axis. The upper population is fluctuating 100, 200, 100, 200 and so on. In other words, it is doubling and halving, doubling and halving. The lower curve is fluctuating 10, 20, 10, 20, 10, 20 and so on. It, too, is doubling and halving, doubling and halving. So the answer to the question is: 'They are equally variable'. It is just that one population has a higher mean value than the other (150 vs. 15 in this case). In order not to fall into the trap of saying that the upper curve is more variable than the lower curve, it is good practice to graph the logarithms rather than the raw values of things like population sizes that change multiplicatively.



Now it is clear that both populations are equally variable.

Finally, we should deal with a rather different measure of central tendency. Consider the following problem. An elephant has a territory which is a square of side 2 km. Each morning, the elephant walks the boundary of this territory. It begins the day at a sedate pace, walking the first side of the territory at a speed of 1 km/hr. On the second side, he has sped up to 2 km/hr. By the third side he has accelerated to an impressive 4 km/hr, but this so wears him out, that he has to return on the final side at a sluggish 1 km/hr. So what is his average speed over the ground? You might say he travelled at 1, 2, 4 and 1 km/hr so the average speed is $(1+2+4+1)/4=8/4=2$ km/hr. But that is wrong. Can you see how to work out the right answer? Recall that velocity is defined as distance travelled divided by time taken. The distance travelled is easy: it is just $4\times2=8$ km. The time taken is a bit harder. The first edge was 2 km long and, travelling at 1 km/hr, this must have taken 2 hr. The second edge was 2 km long and, travelling at 2 km/hr, this must have taken 1 hr. The third edge was 2 km long and, travelling at 4 km/hr, this must have taken 0.5 hr. The final edge was 2 km long and, travelling at 1 km/hr, this must have taken 2 hr. So the total time taken was $2+1+0.5+2=5.5$ hr. So the average speed is not 2 km/hr but $8/5.5=1.4545$ km/hr.

The way to solve this problem is to use the HARMONIC MEAN. This is *the reciprocal of the average of the reciprocals*. Remember that a reciprocal is 'one over'. So the average of our reciprocals is $\left(\frac{1}{1}+\frac{1}{2}+\frac{1}{4}+\frac{1}{1}\right)/4 = 2.75 \div 4 = 0.6875$. The reciprocal of this average is the harmonic mean $1/0.6875 = 1.4545$. In symbols, therefore, the harmonic mean, $\tilde{y}$, (y 'curl') is given by

$$\tilde{y} = \frac{1}{\frac{\sum \frac{1}{y}}{n}} = \frac{n}{\sum \frac{1}{y}}$$

In R, we would write either

```
v <- c(1,2,4,1)
length(v)/sum(1/v)
```

```
[1] 1.454545
```

or

```
1/mean(1/v)
```

```
[1] 1.454545
```

Further Reading

Zar, J.H. (2009) *Biostatistical Analysis*, *5th edn*, Pearson, New York.

4

Variance

A measure of variability is perhaps the most important quantity in statistical analysis. The greater the variability in the data, the greater will be our uncertainty in the values of parameters estimated from the data, and the lower will be our ability to distinguish between competing hypotheses about the data.

Consider the following data, *y*, which are plotted simply in the order in which they were measured:

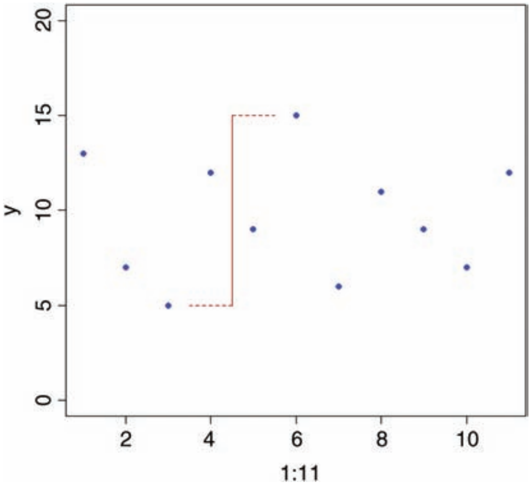
```
y <- c(13,7,5,12,9,15,6,11,9,7,12)
plot(y,ylim=c(0,20))
```

How can we quantify the variation (the scatter) in *y* that we can see here? Perhaps the simplest measure is the `range` of *y* values:

```
range(y)
```

```
[1] 5 15
```

```
plot(1:11,y,ylim=c(0,20),pch=16,col="blue")
lines(c(4.5,4.5),c(5,15),col="brown")
lines(c(4.5,3.5),c(5,5),col="brown",lty=2)
lines(c(4.5,5.5),c(15,15),col="brown",lty=2)
```

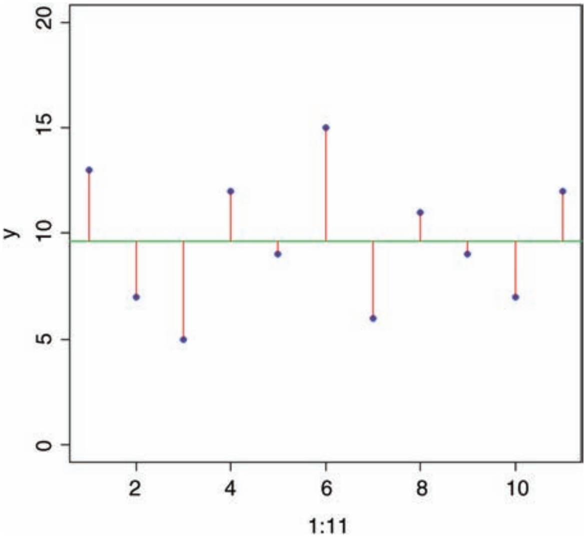


Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

This is a reasonable measure of variability, but it is too dependent on outlying values for most purposes. Also, we want *all* of our data to contribute to the measure of variability, not just the maximum and minimum values.

How about estimating the mean value, and looking at the departures from the mean (known as ‘residuals’ or ‘deviations’)?

```
plot(1:11,y,ylim=c(0,20),pch=16,col="blue")
abline(h=mean(y),col="green")
for (i in 1:11) lines(c(i,i),c(mean(y),y[i]),col="red")
```



The longer the red lines, the more variable the data. So this looks promising. How about adding up the lengths of the red lines: $\sum(y-\bar{y})$? A moment’s thought will show that this is no good, because the negative residuals (from the points below the mean) will cancel out the positive residuals (from the points above the line). In fact, it is easy to prove that this quantity $\sum(y-\bar{y})$ is zero, no matter what the variation in the data, so that’s no good (see Box 4.1).

But the only problem was the minus signs. How about ignoring the minus signs and adding up the absolute values of the residuals: $\sum(|y-\bar{y}|)$. This is a very good measure of variability, and is used in some modern, computationally intensive methods. The problem is that it makes the sums hard, and we do not want that. A much more straightforward way of getting rid of the problem of the minus signs is to square the residuals before we add them up: $\sum(y-\bar{y})^2$. *This is perhaps the most important single quantity in all of statistics*. It is called, somewhat unimaginatively, the *sum of squares*. So, in our figure (above) imagine squaring the lengths of each of the vertical red lines:

```
y - mean(y)
```

```
[1]  3.3636364 -2.6363636 -4.6363636  2.3636364 -0.6363636 5.3636364
[7] -3.6363636  1.3636364 -0.6363636 -2.6363636  2.3636364
```

Box 4.1. The sum of the differences $\sum(y-\bar{y})$ is zero

Start by writing down the differences explicitly:

$$\sum d = \sum (y - \bar{y})$$

Take $\sum$ through the brackets:

$$\sum d = \sum y - \sum \bar{y}$$

The important point is that $\sum \bar{y}$ is the same as $n.\bar{y}$ so

$$\sum d = \sum y - n\bar{y}$$

and we know that $\bar{y} = \sum y/n$, so

$$\sum d = \sum y - \frac{n\sum y}{n}$$

The ns cancel, leaving

$$\sum d = \sum y - \sum y = 0$$

```
(y - mean(y))^2
```

```
[1] 11.3140496  6.9504132 21.4958678 5.5867769 0.4049587 28.7685950
[7] 13.2231405  1.8595041  0.4049587 6.9504132 5.5867769
```

then adding up all these squared differences:

```
sum((y - mean(y))^2)
```

```
[1] 102.5455
```

So the sum of squares for our data is 102.5455. But what are its units? Well that depends on the units in which y is measured. Suppose the y values were lengths in mm. So the units of the sum of squares are mm^2 (like an area).

Now what would happen to the sum of squares if we added a 12th data point? It would get bigger, of course. And it would get bigger for every extra data point we added (except in the unlikely event that our new data point was exactly equal to the mean value, in which case we would add zero squared = 0). We do not want our measure of variability to depend on sample size in this way, so the obvious solution is to divide by the number of samples, to get the *mean squared deviation*.

At this point we need to make a brief, but important, diversion. Before we can progress, we need to understand the concept of degrees of freedom.

Degrees of Freedom

Suppose we had a sample of five numbers and their average was 4. What was the sum of the five numbers? It must have been 20, otherwise the mean would not have been 4. So now let us think about each of the five numbers in turn.

We will put a number in each of the five boxes. If we allow that the numbers could be positive or negative real numbers, we ask how many values could the first number take? Once you see what I'm doing, you will realize it could take any value. Suppose it was a 2.

2				

How many values could the next number take? It could be anything. Say, it was a 7.

2	7			

And the third number? Anything. Suppose it was a 4.

2	7	4		

The fourth number could be anything at all. Say it was 0.

2	7	4	0	

Now then. How many values could the last number take? Just one. It *has* to be another 7 because the numbers have to add up to 20 because the mean of the five numbers is 4.

2	7	4	0	7

To recap. We have total freedom in selecting the first number. And the second, third and fourth numbers. But we have no choice at all in selecting the fifth number. We have 4 degrees of freedom when we have five numbers. In general we have $n-1$ degrees of freedom if we estimated the mean from a sample of size n. More generally still, we can propose a formal definition of degrees of freedom: *degrees of freedom is the sample size, n, minus the number of parameters, p, estimated from the data.*

This is so important, you should memorize it. In the example we plotted earlier we had $n=11$ and we estimated just one parameter from the data: the sample mean, $\bar{y}$. So we have $n-1=10$ d.f.

Variance

We return to developing our quantitative measure of variability. We have come to the conclusion that the sum of squares $\sum(y-\bar{y})^2$ is a good basis for assessing variability, but we have the problem that the sum of squares increases with every new data point we add to the sample. The intuitive thing to do would be to divide by the number of numbers, n, to get the mean squared deviation. But look at the formula for the sum of squares: $\sum(y-\bar{y})^2$. We cannot begin to calculate it until we know the value of the sample mean, $\bar{y}$. And where do we get the value of $\bar{y}$ from? Do we know it in advance? Can we look it up in tables? No, we need to calculate it from the data. The mean value $\bar{y}$ is *a parameter estimated from the data*, so we lose one degree of freedom as a result. Thus, in calculating the mean squared deviation we divide by the degrees of freedom, $n-1$, rather than by the sample size, n. In the jargon, this provides us with an unbiased estimate of the variance, because we have taken account of the fact that one parameter was estimated from the data prior to computation.

Now we can formalize our definition of the measure that we shall use throughout the book for quantifying variability. It is called **variance** and it is represented conventionally by s^2:

$$\text{variance} = \frac{\text{sum of squares}}{\text{degrees of freedom}}$$

This is one of the most important definitions in the book, and you should commit it to memory. We can put it into a more mathematical form, by spelling out what we mean by each of the phrases in the numerator and the denominator:

$$\text{variance} = s^2 = \frac{\sum(y-\bar{y})^2}{n-1}$$

Let’s write an R function to do this. We have most of the necessary components already (see above); the sum of squares is obtained as `sum((y-mean(y))^2)`. For the degrees of freedom, we need to know the number of numbers in the vector, y. This is obtained by the function `length(y)`. Let us call the function `variance` and write it like this:

```
variance <- function (x)  sum((x-mean(x))^2)/(length(x)-1)
```

Now we can try out the function on our data, like this:

```
variance(y)
[1] 10.25455
```

So there we have it. Our quantification of the variation we saw in the first plot is the sample variance, $s^2 = 10.25455$. You will not be surprised that R provides its own, built-in function for calculating variance, and it has an even simpler name than the function we just wrote: `var`

```
var(y)
[1] 10.25455
```

Variance is used in countless ways in statistical analysis, so this section is probably the most important in the whole book, and you should reread it until you are sure that you know exactly what variance is, and precisely what it measures (Box 4.2).

Box 4.2. Shortcut formula for the sum of squares $\sum(y-\bar{y})^2$

The main problem with the formula defining variance is that it involves all those subtractions, $y-\bar{y}$. It would be good to find a simpler way of calculating the sum of squares. Let us expand the bracketed term $(y-\bar{y})^2$ to see if we can make any progress towards a subtraction-free solution:

$$(y-\bar{y})^2 = (y-\bar{y})(y-\bar{y}) = y^2 - 2y\bar{y} + \bar{y}^2$$

So far, so good. Now we apply the summation

$$\sum y^2 - 2\bar{y}\sum y + n\bar{y}^2 = \sum y^2 - 2\frac{\sum y}{n}\sum y + n\left[\frac{\sum y}{n}\right]^2$$

Note that only the ys take the summation sign. This is because we can replace $\sum \bar{y}$ by $n\bar{y}$. Now replace $\bar{y}$ with $\sum y/n$ on the right-hand side, then cancel the ns and collect the terms:

$$\sum y^2 - 2\frac{\left[\sum y\right]^2}{n} + n\frac{\left[\sum y\right]^2}{n^2} = \sum y^2 - \frac{\left[\sum y\right]^2}{n}$$

This is the shortcut formula for computing the sum of squares. It requires only two quantities to be estimated from the data. The sum of the squared y values $\sum y^2$ and the square of the sum of the y values $\left[\sum y\right]^2$.

Variance: A Worked Example

The data in the following table come from three market gardens. The data show the ozone concentrations in parts per hundred million (pphm) on 10 summer days.

```
ozone <- read.csv("c:\\temp\\gardens.csv")
attach(ozone)
ozone
```

```
   gardenA  gardenB  gardenC
1        3        5        3
2        4        5        3
3        4        6        2
4        3        7        1
5        2        4       10
6        3        4        4
7        1        3        3
8        3        5       11
9        5        6        3
10       2        5       10
```

The first step in calculating variance is to work out the mean:

```
mean(gardenA)
[1] 3
```

Now we subtract the mean value (3) from each of the data points:

```
gardenA - mean(gardenA)
 [1]  0  1  1  0 -1  0 -2  0  2 -1
```

This produces a vector of differences (of `length` = 10). We need to square these differences:

```
(gardenA - mean(gardenA))^2
 [1] 0 1 1 0 1 0 4 0 4 1
```

then add up the squared differences:

```
sum((gardenA - mean(gardenA))^2)
[1] 12
```

This important quantity is called ‘the sum of squares’. Variance is the sum of squares divided by degrees of freedom. We have 10 numbers, and have estimated one parameter from the data (the mean) in calculating the sum of squares, so we have 10 – 1 = 9 d.f.

```
sum((gardenA - mean(gardenA))^2)/9
[1] 1.333333
```

So the mean ozone concentration in garden A is 3.0 and the variance in ozone concentration is 1.33. We now do the same for garden B:

```
mean(gardenB)
[1] 5
```

It has a much higher mean ozone concentration than garden A. But what about its variance?

```
gardenB - mean(gardenB)
 [1]  0  0  1  2 -1 -1 -2  0  10
(gardenB - mean(gardenB))^2
 [1] 0 0 1 4 1 1 4 0 1 0
sum((gardenB - mean(gardenB))^2)
[1] 12
```

```
sum((gardenB - mean(gardenB))^2)/9
```

```
[1] 1.333333
```

This is interesting: so although the mean values are quite different, the variances are exactly the same (both have $s^2 = 1.33333$). What about garden C?

```
mean(gardenC)
```

```
[1] 5
```

Its mean ozone concentration is exactly the same as in garden B.

```
gardenC - mean(gardenC)
```

```
 [1] -2 -2 -3 -4  5 -1 -2  6 -2  5
```

```
(gardenC - mean(gardenC))^2
```

```
 [1]  4  4  9 16 25  1  4 36  4 25
```

```
sum((gardenC - mean(gardenC))^2)
```

```
[1] 128
```

```
sum((gardenC - mean(gardenC))^2)/9
```

```
[1] 14.22222
```

So, although the means in gardens B and C are identical, the variances are quite different (1.33 and 14.22, respectively). Are the variances significantly different? We do an *F* test for this, dividing the larger variance by the smaller variance:

```
var(gardenC)/var(gardenB)
```

```
[1] 10.66667
```

Then look up the probability of getting an *F* ratio as big as this by chance alone if the two variances were really the same. We need the cumulative probability of the *F* distribution, which is a function called `pf` that we need to supply with three *arguments*: the size of the variance ratio (10.667), the number of degrees of freedom in the numerator (9) and the number of degrees of freedom in the denominator (also 9). We did not know in advance which garden was going to have the higher variance, so we do what is called a two-tailed test (we simply multiply the probability by 2):

```
2*(1 - pf(10.667,9,9))
```

```
[1] 0.001624002
```

This probability is much less than 5%, so we conclude that there is a highly significant difference between these two variances. We could do this even more simply by using the built-in *F* test:

```
var.test(gardenB,gardenC)
```

```
    F test to compare two variances
data: gardenB and gardenC
F = 0.0938, num df = 9, denom df = 9, p-value = 0.001624
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.02328617 0.37743695
sample estimates:
ratio of variances
           0.09375
```

So the two variances are significantly different. But why does this matter?

What follows is one of the most important lessons so far, so keep rereading it until you are sure that you understand it. Comparing gardens A and B, we can see that two samples can have different means, but the same variance. This is assumed to be the case when we carry out standard tests (like Student's *t* test) to compare two means, or an analysis of variance to compare three or more means. *Constancy of variance* is the most important assumption we make in most statistical analyses.

Comparing gardens B and C, we can see that two samples can have the same mean but different variances. Samples with the same mean are identical, right? Wrong! Let us look into the science in a bit more detail. The damage threshold for lettuces is 8 pphm ozone, so looking at the means shows that both gardens are free of ozone damage on their lettuces (the mean of 5 for both B and C is well below the threshold of 8). Let us look at the raw data for garden B. How many of the days had ozone > 8? Look at the dataframe and you will see that none of the days exceeded the threshold. What about garden C?

```
gardenC
```

```
 [1]  3  3  2  1 10  4  3 11  3 10
```

In garden C ozone reached damaging concentrations on three days out of 10, so 30% of the time the lettuce plants would be suffering ozone damage. This is the key point: when the variances are different, we should not make inferences by comparing the means. When we compare the means, we conclude that garden C is like garden B, and that there will be no ozone damage to the lettuces. When we look at the data, we see that this is completely wrong: there is ozone damage 30% of the time in garden C and none of the time in garden B.

The moral of this story is: *when the variances are different, don't compare the means*. If you do, you run the risk of coming to entirely the wrong conclusion.

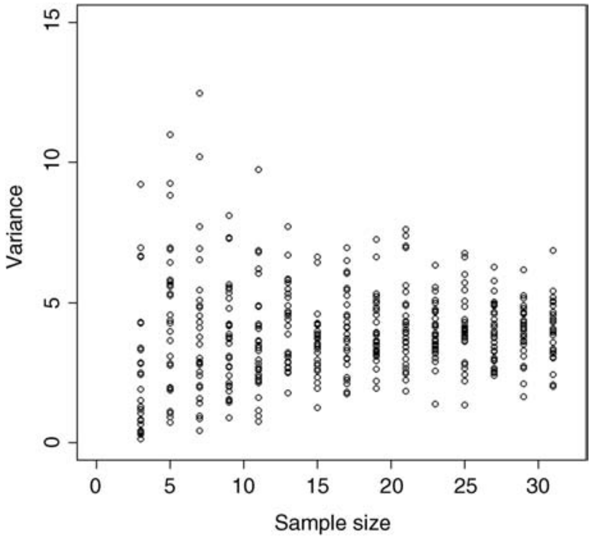
Variance and Sample Size

It is important to understand the relationship between the size of a sample (the replication, *n*) and the value of variance that is estimated. We can do a simple simulation experiment to investigate this:

```
plot(c(0,32),c(0,15),type="n",xlab="Sample size",ylab="Variance")
```

The plan is to select random numbers from a normal distribution using the function `rnorm`. Our distribution is defined as having a mean of 10 and a standard deviation of 2 (this is the square root of the variance, so $s^2 = 4$). We shall work out the variance for sample sizes between $n = 3$ and $n = 31$, and plot 30 independent instances of variance at each of the selected sample sizes:

```
for (n in seq(3,31,2)) {
for( i in 1:30){
x <- rnorm(n,mean=10,sd=2)
points(n,var(x)) }}
```



You see that as sample size declines, the range of the estimates of sample variance increases dramatically (remember that the population variance is constant at $s^2 = 4$ throughout). The problem becomes severe below samples of 13 or so, and is very serious for samples of 7 or fewer. Even for reasonably large samples (like $n = 31$) the variance varies more than threefold in just 30 trials (you can see that the rightmost group of points vary from about 2 to about 6). This means that for small samples, the estimated variance is badly behaved, and this has serious consequences for estimation and hypothesis testing.

When people ask 'How many samples do I need?' a statistician will often answer with another question, 'How many can you afford?' Other things being equal, what we have learned in this chapter is that 30 is a reasonably good sample. Anything less than this is a small sample, and anything less than 10 is a very small sample. Anything more than 30 may be an unnecessary luxury (i.e. a waste of resources). We shall see later when we study power analysis how the question of sample size can be addressed more objectively. But for the time being, take $n = 30$ samples if you can afford it, and you will not go far wrong.

Using Variance

Variance is used in two main ways:

- for establishing measures of unreliability (e.g. confidence intervals)
- for testing hypotheses (e.g. Student's t test)

A Measure of Unreliability

Consider the properties that you would like a measure of unreliability to possess. As the variance of the data increases, what would happen to the unreliability of estimated parameters? Would it go up or down? Unreliability would go *up* as variance increased, so we would want to have the variance on the top (the numerator) of any divisions in our formula for unreliability:

$$\text{unreliability} \propto s^2$$

What about sample size? Would you want your estimate of unreliability to go up or down as sample size, n, increased? You would want unreliability to go *down* as sample size went up, so you would put sample size on the bottom of the formula for unreliability (i.e. in the denominator):

$$\text{unreliability} \propto \frac{s^2}{n}$$

Finally, consider the units in which you would want unreliability to be expressed. What are the units in which our current measure is expressed? Sample size is dimensionless, but variance is based on the sum of squared differences, so it has dimensions of *squared* values of y. So if the mean was a length in cm, the variance would be an area in cm^2. This is an unfortunate state of affairs. It would make good sense to have the dimensions of the unreliability measure the same as the dimensions of the parameter whose unreliability is being measured. That is why all unreliability measures are enclosed inside a big square-root term. Unreliability measures are called *standard errors*. What we have just worked out is the *standard error of the mean*:

$$SE_{\bar{y}} = \sqrt{\frac{s^2}{n}}$$

This is a very important equation and should be memorized. Let us calculate the standard errors of each of our market garden means:

```
sqrt(var(gardenA)/10)

 [1] 0.3651484

sqrt(var(gardenB)/10)

 [1] 0.3651484

sqrt(var(gardenC)/10)

 [1] 1.19257
```

In written work one shows the unreliability of any estimated parameter in a formal, structured way like this:

'The mean ozone concentration in garden A was 3.0 ± 0.365 pphm (1 s.e., $n = 10$)'

You write plus or minus, then the unreliability measure, the units (parts per hundred million in this case) then, in brackets, tell the reader what the unreliability measure is (in this case one *standard error*) and the size of the sample on which the parameter estimate was based (in this case 10). This may seem rather stilted, unnecessary even. But the problem is that unless you do this, the reader will not know what kind of unreliability measure you have used. For example, you might have used a 95% confidence interval or a 99% confidence interval instead of one standard error.

Confidence Intervals

A confidence interval shows the likely range in which the mean would fall if the sampling exercise were to be repeated. It is a very important concept that people always find difficult to grasp at first. It is pretty clear that the confidence interval will get wider as the unreliability goes up, so:

$$\text{confidence interval} \propto \text{unreliability measure} \propto \sqrt{\frac{s^2}{n}}$$

But what do we mean by 'confidence'? This is the hard thing to grasp. Ask yourself this question. Would the interval be wider or narrower if we wanted to be *more* confident that out repeat sample mean will fall inside the interval? It may take some thought, but you should be able to convince yourself that the more confident you want to be, the *wider* the interval will need to be. You can see this clearly by considering the limiting case of complete and absolute certainty. Nothing is certain in statistical science, so the interval would have to be infinitely wide.

We can produce confidence intervals of different widths by specifying different levels of confidence. The higher the confidence, the wider the interval. How exactly does this work? How do we turn the proportionality ($\propto$) in the equation above into equality? The answer is by resorting to an appropriate theoretical distribution (as explained below). Suppose our sample size is too small to use the normal distribution ($n < 30$, as here); then we traditionally use Student's t distribution. The values of Student's t associated with different levels of confidence are available in the function `qt`, which gives the quantiles of the t distribution. Confidence intervals are always two-tailed because the parameter may be larger or smaller than our estimate of it. Thus, if we want to establish a 95% confidence interval we need to calculate the value of Student's t associated with $\alpha = 0.025$ (i.e. with 0.01(100%– 95%)/2). The value is found like this for the left-hand (0.025) and right-hand (0.975) tails:

```
qt(.025,9)

[1] -2.262157

qt(.975,9)

[1] 2.262157
```

The first argument in `qt` is the probability and the second is the degrees of freedom. This says that values as small as −2.262 standard errors below the mean are to be expected in 2.5% of cases ($p = 0.025$), and values as large as +2.262 standard errors above the mean with similar probability ($p = 0.975$). *Values of Student's t are numbers of standard errors to be expected with specified probability and for a given number of degrees of freedom.* The values of t for 99% are bigger than these (0.005 in each tail):

```
qt(.995,9)
```

```
[1] 3.249836
```

and the value for 99.5% confidence are bigger still (0.0025 in each tail):

```
qt(.9975,9)
```

```
[1] 3.689662
```

Values of Student's t like these appear in the formula for calculating the width of the confidence interval, and their inclusion is the reason why the width of the confidence interval goes up as our degree of confidence is increased. The other component of the formula, the standard error, is not affected by our choice of confidence level. So, finally, we can write down the formula for the confidence interval of a mean based on a small sample ($n < 30$):

$$\text{confidence interval} = t\text{-value} \times \text{standard error}$$

$$\mathrm{CI}_{95\%} = t_{(\alpha=0.025,\mathrm{d.f.}=9)}\sqrt{\frac{s^2}{n}}$$

For garden B, therefore, we calculate

```
qt(.975,9)*sqrt(1.33333/10)
```

```
[1] 0.826022
```

and we would present the result in written work like this:

'The mean ozone concentration in garden B was 5.0 ± 0.826 (95% CI, $n = 10$).'

Bootstrap

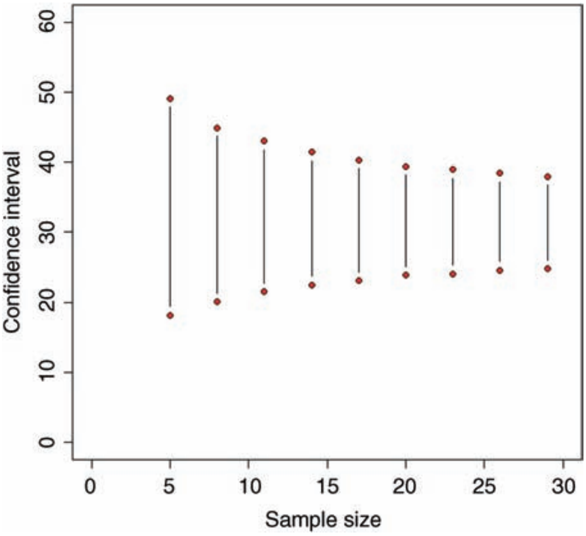
A completely different way of calculating confidence intervals is called bootstrapping. You have probably heard the old phrase about 'pulling yourself up by your own bootlaces'. That is where the term comes from. It is used in the sense of getting 'something for nothing'. The idea is very simple. You have a single sample of n measurements, but you can sample from this in very many ways, so long as you allow some values appear more than once, and other samples to be left out (i.e. *sampling with replacement*). All you do is calculate the sample mean lots of times, once for each sampling from your data, then obtain the confidence interval by looking at the extreme highs and lows of the estimated means using a function called `quantile` to extract the interval you want (e.g. a 95% interval is specified using `c(0.0275, 0.975)` to locate the lower and upper bounds). Here are the data:

```
data <- read.csv("c:\\temp\\skewdata.csv")
attach(data)
names(data)
```

```
[1] "values"
```

We shall simulate sample sizes (k) between 5 and 30, and for each sample size we shall take 10 000 independent samples from our data (the vector called values), using the function called sample with replacement (replace=T):

```
plot(c(0,30),c(0,60),type="n",xlab="Sample size",
                                      ylab="Confidence interval")
for (k in seq(5,30,3)){
       a <- numeric(10000)
       for (i in 1:10000){
            a[i] <- mean(sample(values,k,replace=T))
       }
points(c(k,k),quantile(a,c(.025,.975)),type="b",pch=21,bg="red")
}
```



The confidence interval narrows rapidly over the range of sample sizes up to about 20, but more slowly thereafter. At $n = 30$, the bootstrapped CI based on 10 00 simulations was

```
quantile(a,c(0.025,0.975))
2.5%        97.5%
24.86843    37.6895
```

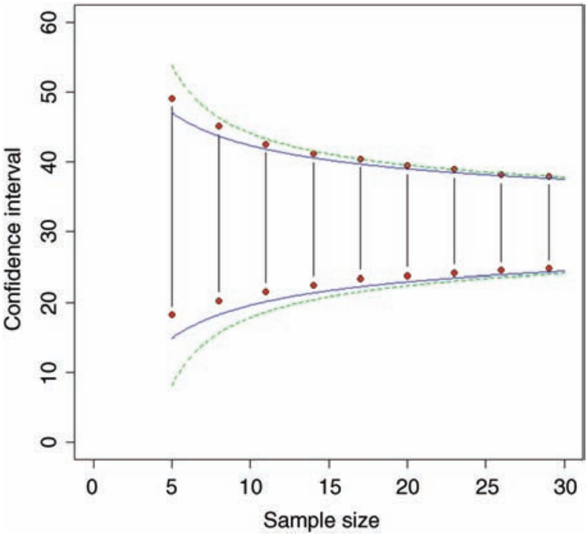
(you will get slightly different values because of the randomization). It is interesting to see how this compares with the normal theory confidence interval: $1.96\sqrt{\frac{s^2}{n}} = 1.96\sqrt{\frac{337.065}{30}} =$ 6.5698 implying that a repeat of the sample is likely to have a mean value in the range 24.39885 to 37.53846. As you see, the estimates from the bootstrap and normal theory are reassuringly close. But they are not identical.

Here are the bootstrapped intervals compared with the intervals calculated from the normal (blue solid line):

```
xv <- seq(5,30,0.1)
yv <- mean(values)+1.96*sqrt(var(values)/xv)
lines(xv,yv,col="blue")
yv <- mean(values)-1.96*sqrt(var(values)/xv)
lines(xv,yv,col="blue")
```

and Student’s *t* distribution (green dotted line):

```
yv <- mean(values)-qt(.975,xv-1)*sqrt(var(values)/xv)
lines(xv,yv,lty=2,col="green")
yv <- mean(values)+qt(.975,xv-1)*sqrt(var(values)/xv)
lines(xv,yv,lty=2,col="green")
```

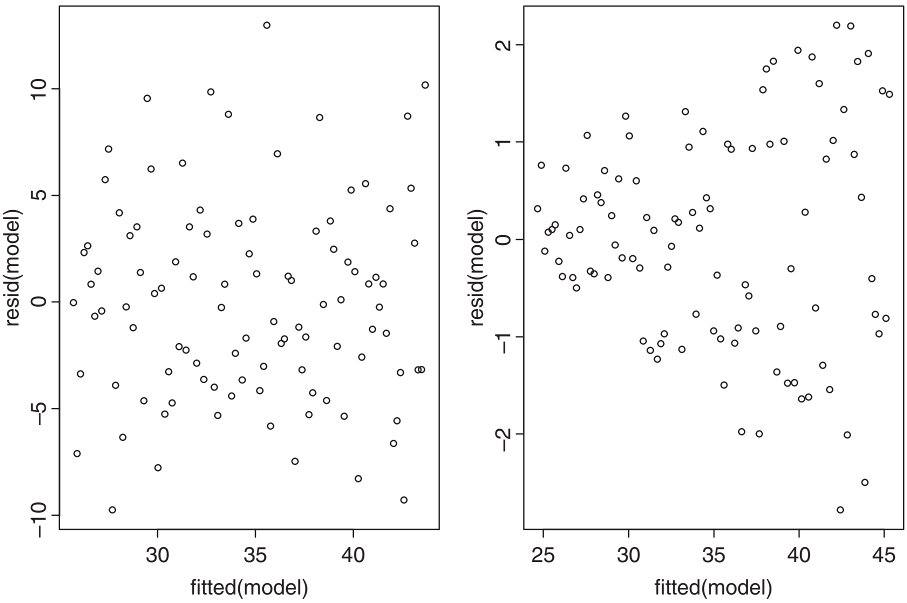


For the upper interval, you see that the bootstrapped intervals (vertical lines and red symbols) fall between the normal (the lower, blue line) and the Student’s *t* distribution (the greater, dotted green line). For the lower interval, however, the bootstrapped intervals are quite different. This is because of the skewness exhibited by these data. Very small values of the response are substantially less likely than predicted by the symmetrical normal (blue line) or Student’s *t* (dotted green line) distributions. Recall that for small-sample confidence intervals using Students *t* distribution, the sample size, *n*, enters the equation twice: once as the denominator in the formula for the standard error of the mean, then again as a determinant of the quantile of the *t* distribution `qt(0.975,n-1)`. That is why the difference between the normal and the Student’s *t* confidence intervals gets bigger as sample size gets smaller.

So which kind of confidence interval should you choose? I prefer the bootstrapped estimate because it makes fewer assumptions. If, as in our example, the data are skew, then this is reflected in the asymmetry of the confidence intervals above and below the mean (6.7 above the mean, and 6.1 below it, at $n = 30$). Both normal and Student's t assume that there is no skew, and so their confidence intervals are symmetrical, whatever the data actually show.

Non-constant Variance: Heteroscedasticity

One of the most important assumptions in classical statistical analysis is that the variance does not change with the mean value of the response variable. A good model must account adequately for the observed variance-mean relationship. A plot of standardised residuals against fitted values should look like the sky at night (points scattered at random over the whole plotting region as in the left hand panel), with no trend in the size or degree of scatter of the residuals. A common problem is that the variance increases with the mean, so that we obtain an expanding, fan-shaped pattern of residuals (right hand panel).



The plot on the left is what we want to see: no trend in the residuals with the fitted values. The plot on the right is a problem. There is a clear pattern of increasing residuals as the fitted values get larger. This is a picture of what *heteroscedasticity* looks like.

Further Reading

Rowntree, D. (1981) *Statistics without Tears: An Introduction for Non-Mathematicians*, Penguin, London.

5

Single Samples

Suppose we have a single sample. The questions we might want to answer are these:

- what is the mean value?
- is the mean value significantly different from current expectation or theory?
- what is the level of uncertainty associated with our estimate of the mean value?

In order to be reasonably confident that our inferences are correct, we need to establish some facts about the distribution of the data:

- are the values normally distributed or not?
- are there outliers in the data?
- if data were collected over a period of time, is there evidence for serial correlation?

Non-normality, outliers and serial correlation can all invalidate inferences made by standard parametric such as like Student's *t* test. It is much better in cases with non-normality and/or outliers to use a non-parametric technique such as Wilcoxon's signed-rank test. If there is serial correlation in the data, then you need to use time series analysis or mixed effects models.

Data Summary in the One-Sample Case

To see what is involved, read the data called *y* from the file called `example.csv`:

```
data <- read.csv("c:\\temp\\example.csv")
attach(data)
names(data)
```

```
[1] "y"
```

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

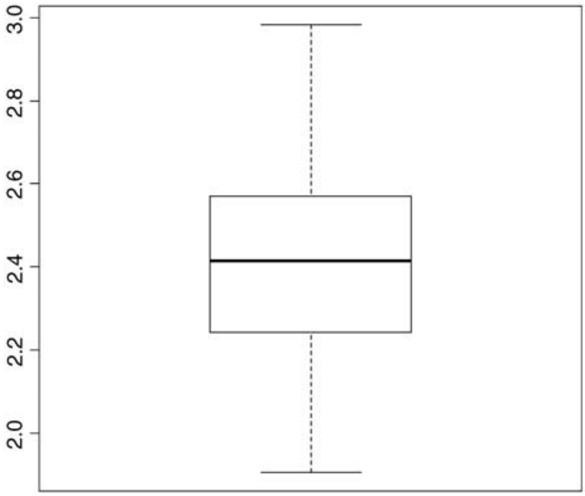
Summarizing the data could not be simpler. We use the built-in function called summary like this:

```
summary(y)
```

```
   Min.  1st Qu.  Median   Mean  3rd Qu.    Max.
  1.904   2.241    2.414  2.419   2.568   2.984
```

This gives us six pieces of information about the vector called *y*. The smallest value is 1.904 (labelled Min. for minimum) and the largest value is 2.984 (labelled Max. for maximum). There are two measures of central tendency: the median is 2.414 and the arithmetic mean in 2.419. What you may be unfamiliar with are the figures labelled '1st Qu.' and '3rd Qu.' The 'Qu.' is an abbreviation of quartile, which means one quarter of the data. The first quartile is the value of the data below which lie the smallest 25% of the data. The median is the second quartile by definition (half the data are smaller than the median). The third quartile is the value of the data above which lie the largest 25% of the data (it is sometimes called the 75th percentile, because 75% of the values of *y* are smaller than this value). The graphical equivalent of this summary table is known as a box-and-whisker plot:

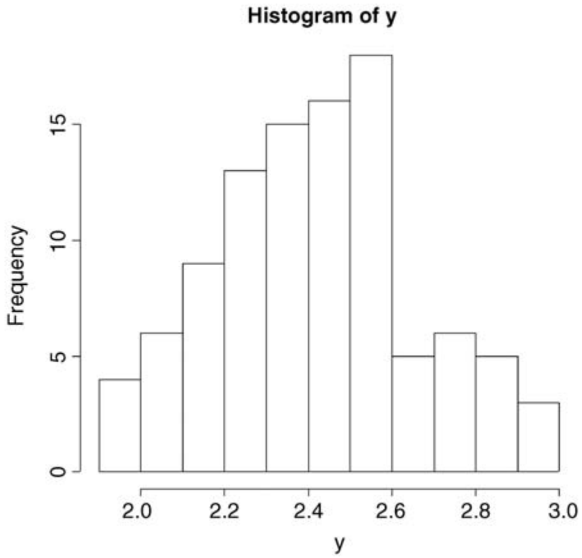
```
boxplot(y)
```



There is a lot of information here. The bold horizontal bar in the middle of the box shows the median value of *y*. The top of the box shows the 75th percentile, and the bottom of the box shows the 25th percentile. The box as a whole shows where the middle 50% of the data lie (this is called the 'interquartile range'; we can see that this is between about 2.25 and 2.55). If the boxes above and below the median are different sizes, then this is indicative of skew in the data. The whiskers show the maximum and minimum values of *y* (later on we shall see what happens when the data contain 'outliers').

Another sort of plot that we might want to use for a single sample is the histogram:

```
hist(y)
```



Histograms are fundamentally different from the graphs that we have encountered so far, because in all cases to date the response variable has been on the *y* axis (the ordinate). With a histogram, the response variable is on the *x* axis (the abscissa). The ordinate of a histogram shows the frequency with which different values of the response were observed. We can see that rather few values of *y* were less that 2.0 or greater than 2.8. The most frequently observed values of *y* were between 2.4 and 2.6. Histograms are related to probability density functions. We shall come across several very important statistical distributions in various parts of the book; we have met the normal and Student's *t* distributions already, and later we shall meet the Poisson, binomial and negative binomial distributions. What they all have in common is that *y* is on the abscissa and the ordinate shows the probability density associated with each value of *y*. You need to be careful not to fall into the trap of confusing graphs and probability distributions (see p. 1).

Our histogram (above) is clearly not symmetrical about its mode (2.5 to 2.6). There are six bars below the mode but only four above the mode. Data like this are said to be 'skew to the left' because the longer tail is on the left of the distribution.

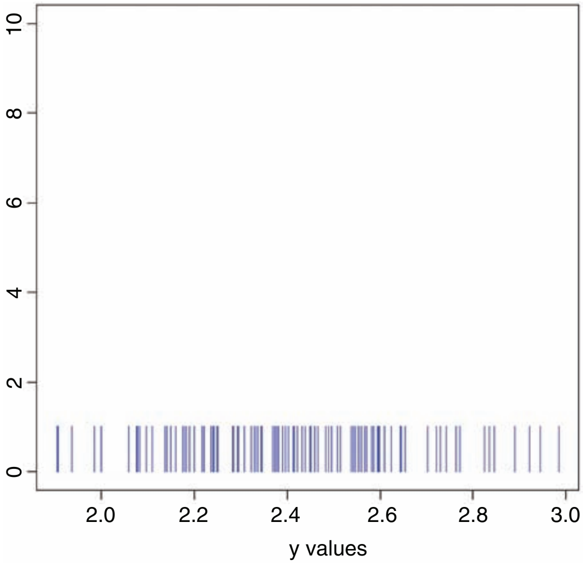
Simple as they seem at first sight, there are actually lots of issues about histograms. Perhaps the most important issue is where exactly to draw the lines between the bars (the 'bin widths' in the jargon). For whole-number (integer) data this is often an easy decision (we could draw a bar of the histogram for each of the integer values of *y*). But for continuous (real number) data like we have here, that approach is a non-starter. How many different values of *y* do we have in our vector of 100 numbers? The appropriate function to answer questions like this is `table`: we do not want to see all the values of *y*, we just want to know how many different values of *y* there are. That is to say, we want to know the length of the `table` of different *y* values:

```
length(table(y))
```

```
[1] 100
```

This shows us that there are no repeats of any of the *y* values, and a histogram of unique values would be completely uninformative (a plot like this is called a 'rug plot', and has short vertical bars placed at each value of *y*).

```
plot(range(y),c(0,10),type="n",xlab="y values",ylab="")
for (i in 1:100) lines(c(y[i],y[i]),c(0,1),col="blue")
```



Let us look more closely to see what R has chosen on our behalf in designing the histogram. The *x* axis is labelled every 0.2 units, in each of which there are two bars. So the chosen bin width is 0.1. R uses simple rules to select what it thinks will make a 'pretty' histogram. It wants to have a reasonable number of bars (too few bars looks dumpy, while too many makes the shape too rough); there are 11 bars in this case. The next criterion is to have 'sensible' widths for the bins. It makes more sense, for instance to have the bins exactly 0.1 units wide (as here) than to use one tenth of the range of *y* values, or one eleventh of the range (note the use of the `diff` and `range` functions) which are close to 0.1 but not equal to 0.1:

```
(max(y)-min(y))/10
```

```
[1] 0.1080075
```

```
diff(range(y))/11
```

```
[1] 0.09818864
```

So a width of 0.1 is a 'pretty' compromise. As we shall see later, you can specify the width of the bins if you do not like the choice that R has made for you, or if you want to draw two histograms that are exactly comparable.

The really important thing to understand is how R decides where to put values that fall exactly on one of the edges of a bin. It could decide to put them in the lower bin (the one to the left) or the higher bin (to the right) or it could toss a coin (heads in the left hand bin, tails in the right). This is hard to understand at first. Suppose that `a` is the value of the lower break and `b` is the value of the higher break for a given bar of the histogram. The convention about what to do is indicated by the use of round brackets and square brackets: `(a,b]` or `[a,b)`. The number next to the square bracket is *included* in the bar, while the number next to the round bracket is *excluded* from this bar. The first convention `(a,b]` is the default in R, and means include the right-hand endpoint `b`, but not the left-hand one `a` in this bar (in the function definition, this is written as `right = TRUE`). In our histogram (above) the modal bin is between 2.5 and 2.6. This would be written as `(2.5,2.6]` and it means that a value of exactly 2.60 would be included in this bin, but a value of exactly 2.50 would not (it would be included in the bin to the left). You will meet this convention again later, when we learn about the `cut` function for converting a continuous variable into a categorical variable (p. 276).

The main problem with histograms is the arbitrariness about where the breaks are put and how wide the bins are defined to be. For instance, distributions that look bimodal with narrow bins can look unimodal with wider bins. The moral about histograms is 'take care; all may not be as it seems'.

The Normal Distribution

This famous distribution has a central place is statistical analysis. If you take repeated samples from a population and calculate their averages, then these averages will be normally distributed. This is called the *central limit theorem*. Let us demonstrate it for ourselves.

You may be familiar with the ancient game of 'craps'. It involves the use of two 6-sided dice. In its simplest form the two dice are thrown and the two scores added together. The lowest score you can get is $1 + 1 = 2$ and the highest is $6 + 6 = 12$. There is only one way of getting each of these scores so they both have the same low probability ($1/6 \times 1/6 = 1/36$). You can score 3 by throwing 1 and 2 or 2 and 1 (so the probability of getting 3 is $2 \times 1/36 = 1/18$; the same as scoring 11 by getting 5 and 6 or 6 and 5). The most likely score is 7 because there are so many ways of getting this: 1 and 6, 2 and 5, 3 and 4, 4 and 3, 5 and 2 or 6 and 1). Let us simulate 10 000 plays of the game and produce a histogram of the results. The possible scores are the 11 numbers from 2 to 12:

```
score <- 2:12
```

The number of ways of getting each score are:

```
ways <- c(1,2,3,4,5,6,5,4,3,2,1)
```

We can use the rep function to produce a vector of all the 36 possible outcomes:

```
game <- rep(score,ways)
game
```

```
 [1] 2 3 3 4 4 4 5 5 5  5  6  6  6  6  6 7 7 7 7 7 7
[22] 8 8 8 8 8 9 9 9 9 10 10 10 11 11 12
```

Now we draw a single random sample from this vector to represent the outcome of one throw (this game produced a score of 5):

```
sample(game,1)
```

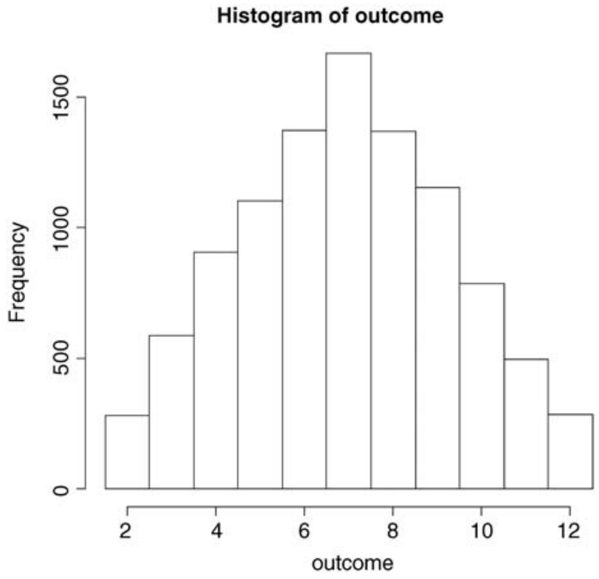
```
[1] 5
```

and we record this score in a vector called outcome. The game is repeated 10 000 times:

```
outcome <- numeric(10000)
for (i in 1:10000) outcome[i] <- sample(game,1)
```

This is what the distribution of outcomes looks like:

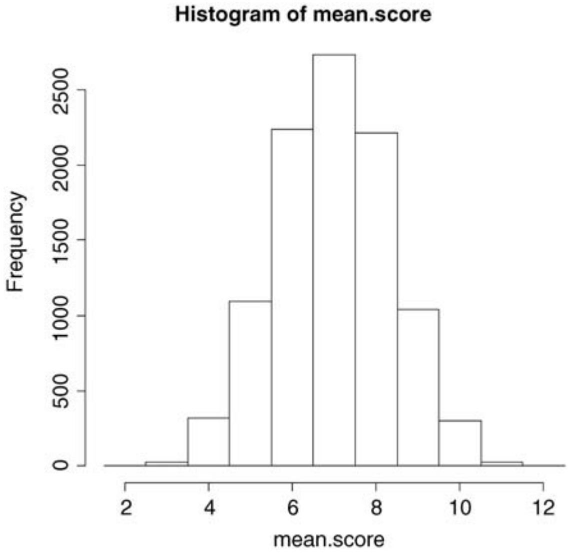
```
hist(outcome,breaks=(1.5:12.5))
```



Note the trick of specifying the breakpoints to be offset by 0.5 in order to get the correct labels in the centre of the relevant bars.

The distribution is very well behaved, but it is clearly triangular, not bell-shaped like the normal. What if we work out the mean score over, say, three games? Presumably, the mean score is still going to be 7 (as above) but what will the distribution of mean scores look like? We shall try it and see:

```
mean.score <- numeric(10000)
for (i in 1:10000) mean.score[i] <- mean(sample(game,3))
hist(mean.score,breaks=(1.5:12.5))
```



That was a demonstration of the central limit theorem in action. The triangular distribution of scores has become a normal distribution of mean scores, even though we were averaging across only three games. To demonstrate the goodness of fit to the normal distribution, we can overlay the histogram with a smooth probability density function generated from a normal distribution (`dnorm`) with the same mean and standard deviation of our actual sample of games:

```
mean(mean.score)
[1] 6.9821
sd(mean.score)
[1] 1.366118
```

To accommodate the top of the smooth density function, we need to make the *y* axis a little longer: `ylim=c(0,3000)`. To generate a smooth curve, we need a series of values for the *x* axis ranging between 2 and 12 (as a rule of thumb, you need 100 or so values to make a smooth-looking curve in R):

```
xv <- seq(2,12,0.1)
```

Now calculate the height of the curve. The standard normal has an integral of 1.0 but our histogram has an integral of 10 000 so we calculate the height of the curve like this

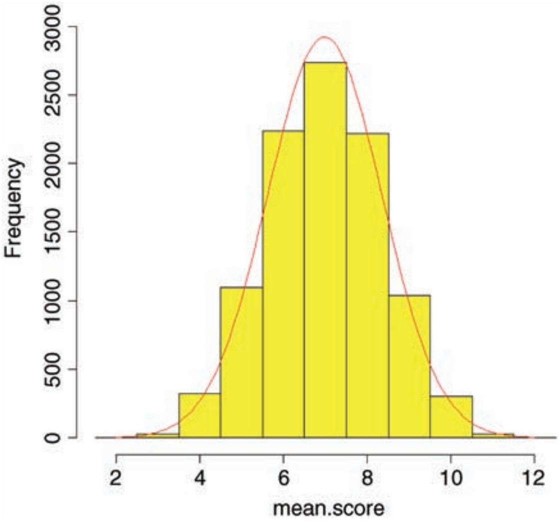
```
yv <- 10000*dnorm(xv,mean(mean.score),sd(mean.score))
```

We shall make a few minor embellishments by removing the heading from the plot (`main=""`), colouring the bars in yellow (`col="yellow"`):

```
hist(mean.score,breaks=(1.5:12.5),ylim=c(0,3000),
                                        col="yellow", main="")
```

and overlaying the normal probability density in red:

```
lines(xv,yv,col="red")
```



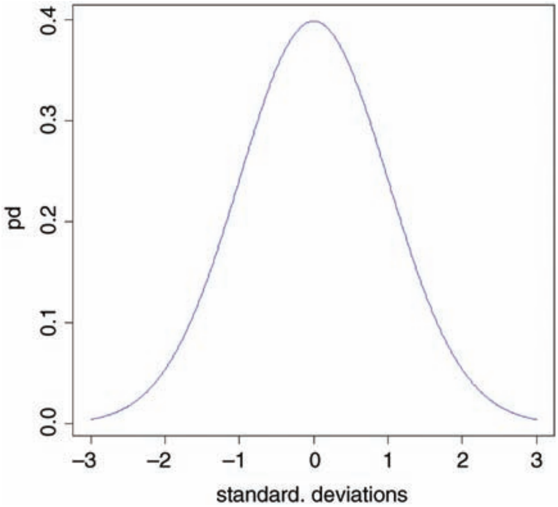
As you can see, the fit to the normal distribution is excellent, even though we were averaging across just three throws of the dice. The central limit theorem really works. Almost any distribution, even a ‘badly behaved’ one like the negative binomial (p. 251), will produce a normal distribution of sample means taken from it.

The great thing about the normal distribution is that we know so much about its shape. Obviously, all values must lie between minus infinity and plus infinity, so the area under the whole curve is 1.0. The distribution is symmetrical, so half of our samples will fall below the mean, and half will be above it (i.e. the area beneath the curve to the left of the mean is 0.5). The important thing is that we can predict the distribution of samples in various parts of the curve. For example, *c*.16% of samples will be more than 1 standard deviation above the mean, and *c*.2.5% of samples will be more than 2 standard deviations below the mean. But how do I know this?

There is an infinity of different possible normal distributions: the mean can be anything at all, and so can the standard deviation. For convenience, it is useful to have a standard normal distribution, whose properties we can tabulate. But what would be a sensible choice for the mean of such a standard normal distribution? 12.7? Obviously not. 1? Not bad, but the distribution is symmetrical, so it would be good to have the left and right halves with similar scales (not 1 to 4 on the right, but −2 to 1 on the left). The only really sensible choice is to have the mean equal to 0. What about the standard deviation? Should that be 0 as well? Hardly, since that would be a distribution with no spread at all. Not very useful. It could be any positive number, but in practice the most sensible choice is 1. So there you have it. The standard normal distribution is one specific case of the normal with mean = 0 and standard deviation = 1. So how does this help?

It helps a lot, because now we can work out the area below the curve up to any number of standard deviations (these are the values on the *x* axis):

```
standard.deviations <- seq(-3,3,0.01)
pd <- dnorm(standard.deviations)
plot(standard.deviations,pd,type="l",col="blue")
```



You can see that almost all values fall within 3 standard deviations of the mean, one way or the other. It is easy to find the area beneath the curve for any value on the *x* axis (i.e. for any specified value of the standard deviation). Let us start with standard deviation = −2. What is the area beneath the curve to the left of −2? It is obviously a small number, but the curvature makes it hard to estimate the area accurately from the plot. R provides the answer with a function called `pnorm` (‘probability for a normal distribution’; strictly ‘cumulative probability’, as we shall see). Because we are dealing with a standard normal (mean = 0, sd = 1) we need only specify the value of the normal deviate, which is −2 in our case:

```
pnorm(-2)
```

```
[1] 0.02275013
```

This tells us that just a bit less than 2.5% of values will be lower than −2. What about 1 standard deviation below the mean?

```
pnorm(-1)
[1] 0.1586553
```

In this case, about 16% of random samples will be smaller than 1 standard deviation below the mean. What about big values of the normal deviate? The density function shows a maximum of +3. What is the probability of getting a sample from a normal distribution that is more than 3 standard deviations above the mean? The only point to note here is that `pnorm` gives the probability of getting a value *less* than the value specified (not more, as we want here). The trick is simply to subtract the value given by `pnorm` from 1 to get the answer we want:

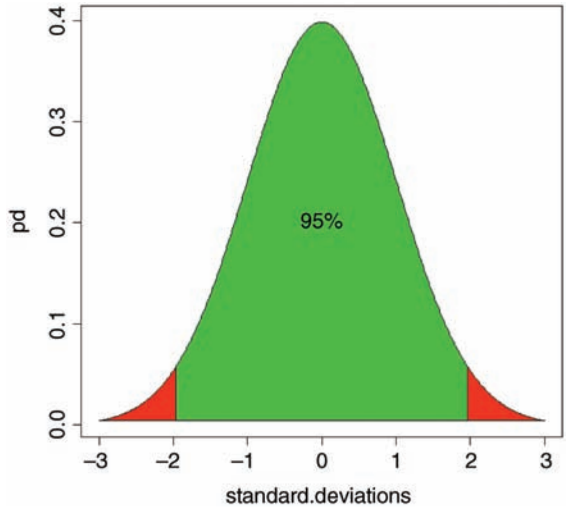
```
1-pnorm(3)
[1] 0.001349898
```

This tells us that a value as large as 3 or more is very unlikely indeed: less than 0.2%, in fact.

Probably the most frequent use of the standard normal distribution is in working out the values of the normal deviate that can be expected by chance alone. This, if you like, is the opposite kind of problem to the ones we have just been dealing with. There, we provided a value of the normal deviate (such as −1, or −2 or +3) and asked what probability was associated with such a value. Now, we want to provide a probability and find out what value of the normal deviate is associated with that probability. Let us take an important example. Suppose we want to know the upper and lower values of the normal deviate *between which 95% of samples are expected to lie*. This means that 5% of samples will lie outside this range, and because the normal is a symmetrical distribution, this means that 2.5% of values will be expected to be smaller than the lower bound (i.e. lie to the left of the lower bound) and 2.5% of values will be expected to be greater than the upper bound (i.e. lie to the right of the upper bound). The function we need is called `qnorm` ('quantiles of the normal distribution') and it is used by specifying our two probabilities 0.025 and 0.975 in a vector like this `c(0.025,0.975)`:

```
qnorm(c(0.025,0.975))
[1] -1.959964 1.959964
```

These are two very important numbers in statistics. They tell us that with a normal distribution, 95% of randomly selected values will fall between −1.96 and +1.96 standard deviations of the mean. Let us shade in these areas under the normal probability density function to see what is involved:



In the green area between the two vertical lines, we can expect 95% of all random samples to fall; we expect 2.5% of samples to be more than 1.96 standard deviations below the mean (the left-hand red area), and we expect 2.5% of samples to be greater than 1.96 standard deviations above the mean (the right-hand red area). If we discover that this is *not* the case, then our sample is not normally distributed. It might, for instance, follow a Student's *t* distribution (see p. 82).

To sum up: if we want to provide values of the normal deviate and work out probabilities, we use `pnorm`; if we want to provide probabilities and work out values of the normal deviate, we use `qnorm`. You should try and remember this important distinction.

Calculations Using *z* of the Normal Distribution

Suppose we have measured the heights of 100 people. The mean height was 170 cm and the standard deviation was 8 cm. The normal distribution looks like this:

```
ht <- seq(150,190,0.01)
plot(ht,dnorm(ht,170,8),type="l",col="brown",
                  ylab="Probability density",xlab="Height")
```

(the top left-hand panel in the plots below). We can ask three sorts of questions about data like these. What is the probability that a randomly selected individual will be:

- shorter than a particular height?
- taller than a particular height?
- between one specified height and another?

The area under the whole curve is exactly 1; everybody has a height between minus infinity and plus infinity. True, but not particularly helpful. Suppose we want to know the probability that one of our people, selected at random from the group, will be less than

160 cm tall. We need to convert this height into a value of z; that is to say, we need to convert 160 cm into *a number of standard deviations from the mean*. What do we know about the standard normal distribution? It has a mean of 0 and a standard deviation of 1. So we can convert any value y, from a distribution with mean $\bar{y}$ and standard deviation s to a standard normal very simply by calculating:

$$z = \frac{y - \bar{y}}{s}$$

So we convert 160 cm into a number of standard deviations. It is less than the mean height (170 cm) so its value will be negative:

$$z = \frac{160 - 170}{8} = -1.25$$

Now we need to find the probability of a value of the standard normal taking a value of −1.25 or smaller. This is the area under the left-hand tail of the distribution. The function we need for this is `pnorm`: we provide it with a value of z (or, more generally, with a quantile) and it provides us with the probability we want:

```
pnorm(-1.25)
[1] 0.1056498
```

So the answer to our first question is just over 10% (the orange shaded area, below).

The second question is: What is the probability of selecting one of our people and finding that they are taller than 185 cm? The first two parts of the exercise are exactly the same as before. First we convert our value of 185 cm into a number of standard deviations:

$$z = \frac{185 - 170}{8} = 1.875$$

Then we ask what probability is associated with this, using `pnorm`:

```
pnorm(1.875)
[1] 0.9696036
```

But this is the answer to a different question. This is the probability that someone will be *less* than 185 cm tall (that is what the function `pnorm` has been written to provide). All we need to do is to work out the complement of this:

```
1 - pnorm(1.875)
[1] 0.03039636
```

So the answer to the second question is about 3% (the blue shaded area, below).

Finally, we might want to know the probability of selecting a person between 165 cm and 180 cm. We have a bit more work to do here, because we need to calculate two z values:

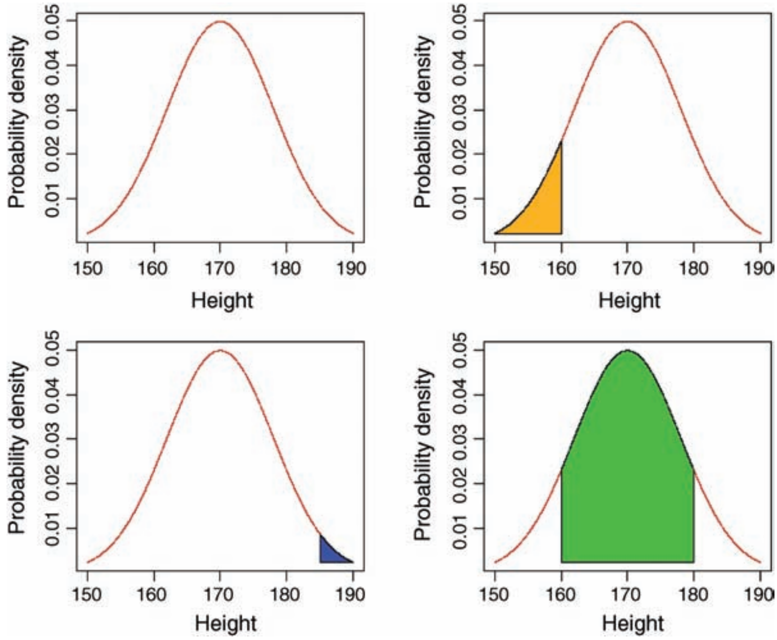
$$z_1 = \frac{165 - 170}{8} = -0.625 \quad \text{and} \quad z_2 = \frac{180 - 170}{8} = 1.25$$

The important point to grasp is this: we want the probability of selecting a person between these two z values, so we *subtract the smaller probability from the larger probability*. It might help to sketch the normal curve and shade in the area you are interested in:

```
pnorm(1.25) - pnorm(-0.625)
```

```
[1] 0.6283647
```

Thus we have a 63% chance of selecting a medium-sized person (taller than 165 cm and shorter than 180 cm) from this sample with a mean height of 170 cm and a standard deviation of 8 cm (the green shaded area, below).



The function called `polygon` is used for colouring in different shaped areas under the curve: to see how it is used, type `?polygon`

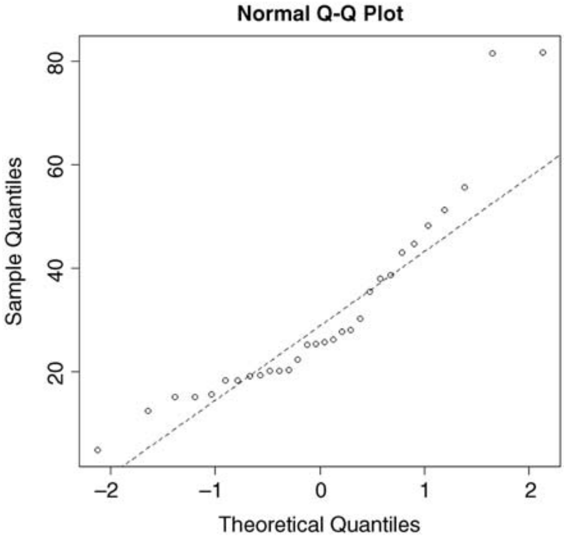
```
par(mfrow=c(2,2))

ht <- seq(150,190,0.01)
pd <- dnorm(ht,170,8)

plot(ht,dnorm(ht,170,8),type="l",col="brown",
ylab="Probability density",xlab="Height")

plot(ht,dnorm(ht,170,8),type="l",col="brown",
ylab="Probability density",xlab="Height")
yv <- pd[ht<=160]
```

```
xv <- ht[ht<=160]
xv <- c(xv,160,150)
yv <- c(yv,yv[1],yv[1])
polygon(xv,yv,col="orange")

plot(ht,dnorm(ht,170,8),type="l",col="brown",
ylab="Probability density",xlab="Height")
xv <- ht[ht>=185]
yv <- pd[ht>=185]
xv <- c(xv,190,185)
yv <- c(yv,yv[501],yv[501])
polygon(xv,yv,col="blue")

plot(ht,dnorm(ht,170,8),type="l",col="brown",
ylab="Probability density",xlab="Height")
xv <- ht[ht>=160 & ht <= 180]
yv <- pd[ht>=160 & ht <= 180]
xv <- c(xv,180,160)
yv <- c(yv,pd[1],pd[1])
polygon(xv,yv,col="green")
```

Plots for Testing Normality of Single Samples

The simplest test of normality (and in many ways the best) is the ‘quantile–quantile plot’; it plots the ranked samples from our distribution against a similar number of ranked quantiles taken from a normal distribution. If the sample is normally distributed then the line will be straight. Departures from normality show up as various sorts of non-linearity (e.g. S-shapes or banana shapes). The functions you need are `qqnorm` and `qqline` (quantile–quantile plot against a normal distribution):

```
data <- read.csv("c:\\temp\\skewdata.csv")
attach(data)
qqnorm(values)
qqline(values,lty=2)
```



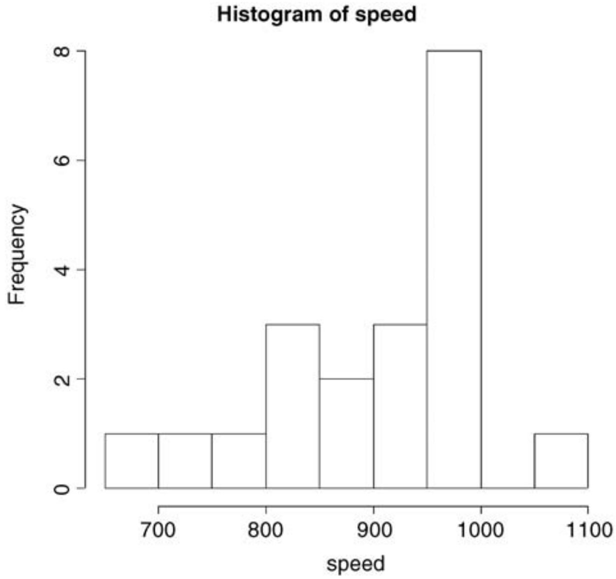
This shows a marked S-shape, indicative of non-normality (as we already know, our distribution is non-normal because it is skew to the left; see p. 68).

We can investigate the issues involved with Michelson's (1880) famous data on estimating the speed of light. The actual speed is 299 000 km s^{-1} plus the values in our dataframe called light:

```
light <- read.csv("c:\\temp\\light.csv")
attach(light)
names(light)
```

```
[1] "speed"
```

```
hist(speed)
```



We get a summary of the non-parametric descriptors of the sample like this:

```
summary(speed)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    650     850     940     909     980    1070
```

From this, you see at once that the median (940) is substantially bigger than the mean (909), as a consequence of the strong negative skew in the data seen in the histogram. The interquartile range, the difference between the first and third quartiles, is $980 - 850 = 130$. This is useful in the detection of outliers: a good rule of thumb is this

> an *outlier* is a value more than 1.5 times the interquartile range above the third quartile, or below the first quartile.

($130 \times 1.5 = 195$). In this case, therefore, outliers would be measurements of speed that were less than $850 - 195 = 655$ or greater than $980 + 195 = 1175$. You will see that there are no large outliers in this data set, but one or more small outliers (the minimum is 650).

Inference in the One-Sample Case

We want to test the hypothesis that Michelson's estimate of the speed of light is significantly different from the value of 299 990 thought to prevail at the time. The data have all had 299 000 subtracted from them, so the test value is 990. Because of the non-normality, the use of Student's *t* test in this case is ill advised. The correct test is Wilcoxon's signed-rank test:

```
wilcox.test(speed,mu=990)

        Wilcoxon signed rank test with continuity correction

data: speed

V = 22.5, p-value = 0.00213

alternative hypothesis: true location is not equal to 990

Warning message:
In wilcox.test.default(speed, mu = 990) :
  cannot compute exact p-value with ties
```

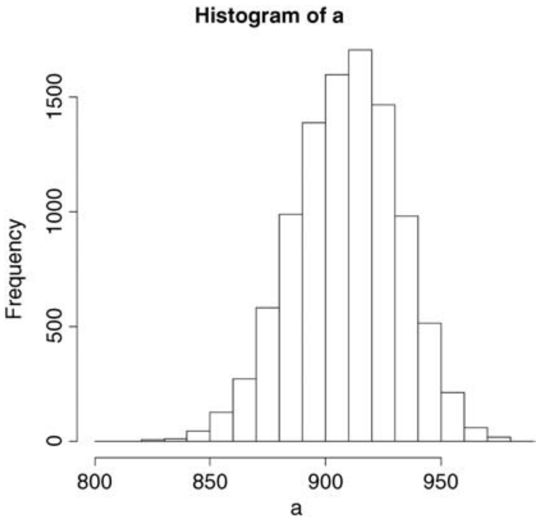
We reject the null hypothesis and accept the alternative hypothesis because $p = 0.00213$ (i.e. much less than 0.05). The speed of light is significantly less than 990.

Bootstrap in Hypothesis Testing with Single Samples

We shall meet parametric methods for hypothesis testing later. Here we use bootstrapping to illustrate another non-parametric method of hypothesis testing. Our sample mean value of *y* is 909. The question we have been asked to address is this: 'How likely is it that the population mean that we are trying to estimate with our random sample of 100 values is as big as 990?'

We take 10 000 random samples with replacement using $n = 100$ from the 100 values of light and calculate 10 000 values of the mean. Then we ask: what is the probability of obtaining a mean as large as 990 by inspecting the right-hand tail of the cumulative probability distribution of our 10 000 bootstrapped mean values? This is not as hard as it sounds:

```
a <- numeric(10000)
for(i in 1:10000) a[i] <- mean(sample(speed,replace=T))
hist(a)
```



The test value of 990 is off the scale to the right. A mean of 990 is clearly most unlikely, given the data:

```
max(a)
```

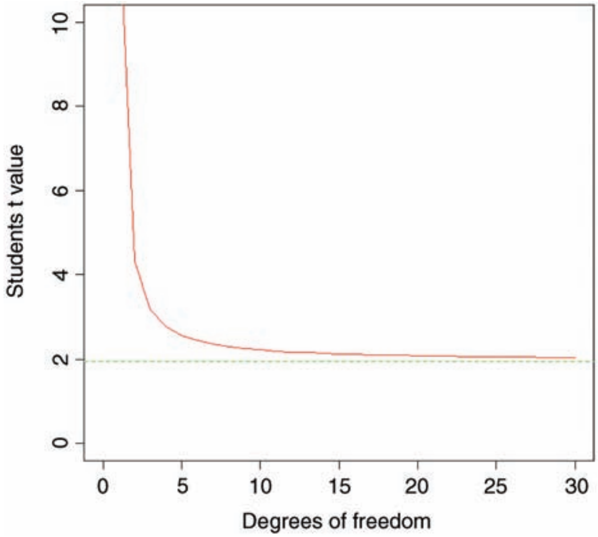
```
[1] 983
```

In our 10 000 samples of the data, we never obtained a mean value greater than 983, so the probability that the mean is 990 is clearly $p < 0.0001$.

Student's *t* Distribution

Student's *t* distribution is used instead of the normal distribution when sample sizes are small ($n < 30$). Recall that the 95% intervals of the standard normal were −1.96 to +1.96 standard deviations. Student's *t* distribution produces bigger intervals than this. The smaller the sample, the bigger the interval. Let us see this in action. The equivalents of `pnorm` and `qnorm` are `pt` and `qt`. We are going to plot a graph to show how the upper interval (equivalent to the normal's 1.96) varies with sample size in a *t* distribution. This is a deviate, so the appropriate function is `qt`. We need to supply it with the probability (in this case $p = 0.975$) and the degrees of freedom (we shall vary this from 1 to 30 to produce the graph)

```
plot(c(0,30),c(0,10),type="n",
                              xlab="Degrees of freedom",ylab="Students t value")
lines(1:30,qt(0.975,df=1:30),col="red")
abline(h=1.96,lty=2,col="green")
```



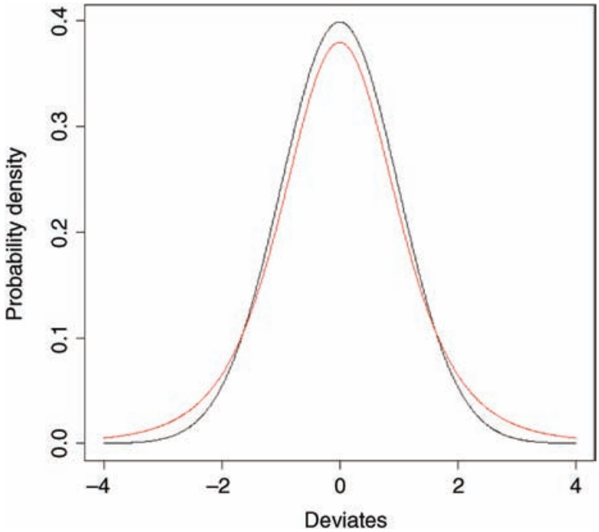
The importance of using Student's *t* rather than the normal is relatively slight until the degrees of freedom fall below about 10 (above which the critical value is roughly 2), and then it increases dramatically below about 5 degrees of freedom. For samples with more than 30 degrees of freedom, Student's *t* produces an asymptotic value of 1.96, just like the normal (this is the horizontal green dotted line). The graph demonstrates that Student's $t = 2$ is a reasonable rule of thumb; memorizing this will save you lots of time in looking up critical values in later life.

So what does the *t* distribution look like, compared to a normal? Let us redraw the standard normal in black:

```
xvs <- seq(-4,4,0.01)
plot(xvs,dnorm(xvs),type="l",
                     ylab="Probability density",xlab="Deviates")
```

Now we can overlay Student's *t* with d.f. = 5 as a red line to see the difference:

```
lines(xvs,dt(xvs,df=5),col="red")
```



The difference between the normal (black line) and Student's *t* distributions (red line) is that the *t* distribution has 'fatter tails'. This means that extreme values are more likely with a *t* distribution than with a normal, and the confidence intervals are correspondingly broader. So instead of a 95% interval of ±1.96 with a normal distribution we should have a 95% interval of ±2.57 for a Student's *t* distribution with just 5 degrees of freedom:

```
qt(0.975,5)
[1] 2.570582
```

Higher-Order Moments of a Distribution

So far, and without saying so explicitly, we have encountered the first two moments of a sample distribution. The quantity $\sum y$ was used in the context of defining the arithmetic mean of a single sample: this is the first moment, $\bar{y} = \sum y/n$. The quantity $\sum(y - \bar{y})^2$, the sum of squares, was used in calculating sample variance, and this is the second moment of the distribution, $s^2 = \sum (y - \bar{y})^2/(n-1)$. Higher-order moments involve powers of the difference greater than 2, such as $\sum (y - \bar{y})^3$ and $\sum (y - \bar{y})^4$.

Skew

Skew (or skewness) is the dimensionless version of the third moment about the mean

$$m_3 = \frac{\sum (y - \bar{y})^3}{n}$$

which is rendered dimensionless by dividing by the cube of the standard deviation of y (because this is also measured in units of y^3):

$$s_3 = \text{sd}(y)^3 = \left(\sqrt{s^2}\right)^3$$

The skew is then given by

$$\text{skew} = \gamma_1 = \frac{m_3}{s_3}$$

It measures the extent to which a distribution has long, drawn-out *tails* on one side or the other. A normal distribution is symmetrical and has skew = 0. Negative values of γ_1 mean skew to the left (negative skew) and positive values mean skew to the right. To test whether a particular value of skew is significantly different from 0 (and hence the distribution from which it was calculated is significantly non-normal) we divide the estimate of skew by its approximate standard error:

$$SE_{\gamma_1} = \sqrt{\frac{6}{n}}$$

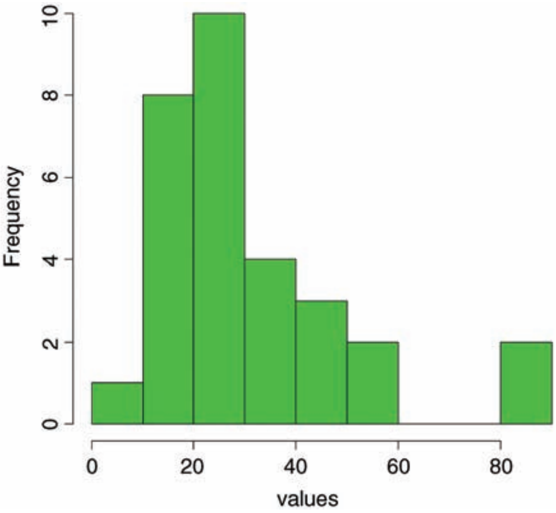
It is straightforward to write an R function to calculate the degree of skew for any vector of numbers, x, like this:

```
skew <- function(x){
m3 <- sum((x-mean(x))^3)/length(x)
s3 <- sqrt(var(x))^3
m3/s3 }
```

Note the use of the `length(x)` function to work out the sample size, n, whatever the size of the vector x. The last expression inside a function is not assigned to a variable name, and is returned as the value of `skew(x)` when this is executed from the command line.

We use the data from the file called `skewdata.txt` that we read on p. 79. To illustrate the skew, we plot a histogram of `values`, taking the opportunity to introduce two useful options: `main=""` to suppress the production of a title and `col="green"` to fill in the histogram bars in a chosen colour:

```
hist(values,main="",col="green")
```



The data appear to be positively skew (i.e. to have a longer tail on the right than on the left). We use the new function `skew` to quantify the degree of skewness:

```
skew(values)
[1] 1.318905
```

Now we need to know whether a skew of 1.319 is significantly different from zero. We do a t test, dividing the observed value of skew by its standard error $\sqrt{6/n}$:

```
skew(values)/sqrt(6/length(values))
[1] 2.949161
```

Finally, we ask: what is the probability of getting a t value of 2.949 by chance alone, given that we have 28 degrees of freedom, when the skew value really is zero?

```
1 - pt(2.949,28)
[1] 0.003185136
```

We conclude that these data show significant non-normality ($p < 0.0032$). Note that we have $n - 2 = 28$ degrees of freedom, because in order to calculate skew we needed to know the values of two parameters that were estimated from the data: the mean and the variance.

The next step might be to look for a transformation that normalizes the data by reducing the skewness. One way of drawing in the larger values is to take square roots, so let us try this to begin with:

```
skew(sqrt(values))/sqrt(6/length(values))
[1] 1.474851
```

This is not significantly skew. Alternatively, we might take the logs of the values:

```
skew(log(values))/sqrt(6/length(values))
[1] -0.6600605
```

This is now slightly skew to the left (negative skew), but the value of Student's *t* is smaller than with a square-root transformation, so we might prefer a log transformation in this case.

Kurtosis

This is a measure of non-normality that has to do with the peakyness, or flat-toppedness, of a distribution. The normal distribution is bell-shaped, whereas a kurtotic distribution is other than bell-shaped. In particular, a more flat-topped distribution is said to be platykurtic, and a more pointy distribution is said to be leptokurtic. Kurtosis is the dimensionless version of the fourth moment about the mean

$$m_4 = \frac{\sum (y - \bar{y})^4}{n}$$

which is rendered dimensionless by dividing by the square of the variance of y (because this is also measured in units of y^4):

$$s_4 = \text{var}(y)^2 = (s^2)^2$$

Kurtosis is then given by

$$\text{kurtosis} = \gamma_2 = \frac{m_4}{s_4} - 3$$

The minus 3 is included because a normal distribution has $m_4/s_4 = 3$. This formulation therefore has the desirable property of giving zero kurtosis for a normal distribution, while a flat-topped (platykurtic) distribution has a negative value of kurtosis, and a pointy (leptokurtic) distribution has a positive value of kurtosis. The approximate standard error of kurtosis is

$$SE_{\gamma_2} = \sqrt{\frac{24}{n}}$$

An R function to calculate kurtosis might look like this:

```
kurtosis <- function(x) {
m4 <- sum((x-mean(x))^4)/length(x)
s4 <- var(x)^2
m4/s4 - 3 }
```

For our present data, we find that kurtosis is not significantly different from normal:

```
kurtosis(values)
```

```
[1] 1.297751
kurtosis(values)/sqrt(24/length(values))
[1] 1.450930
```

because the t value (1.45) is substantially less than the rule of thumb (2.0).

Reference

Michelson, A.A. (1880) Experimental determination of the velocity of light made at the U.S. Naval Academy, Annapolis. *Astronomical Papers*, **1**, 109–145.

Further Reading

Field, A., Miles, J. and Field, Z. (2012) *Discovering Statistics Using R*, Sage, London.

Williams, D. (2001) *Weighing the Odds. A Course in Probability and Statistics*, Cambridge University Press, Cambridge.

6

Two Samples

There is absolutely no point in carrying out an analysis that is more complicated than it needs to be. Occam's razor applies to the choice of statistical model just as strongly as to anything else: simplest is best. The so-called classical tests deal with some of the most frequently-used kinds of analysis, and they are the models of choice for:

- comparing two variances (Fisher's *F* test, `var.test`)
- comparing two sample means with normal errors (Student's *t* test, `t.test`)
- comparing two means with non-normal errors (Wilcoxon's test, `wilcox.test`)
- comparing two proportions (the binomial test, `prop.test`)
- correlating two variables (Pearson's or Spearman's rank correlation, `cor.test`)
- testing for independence in contingency tables using chi-squared (`chisq.test`)
- testing small samples for correlation with Fisher's exact test (`fisher.test`)

Comparing Two Variances

Before we can carry out a test to compare two sample means, we need to test whether the sample variances are significantly different (see p. 55). The test could not be simpler. It is called Fisher's *F* test after the famous statistician and geneticist R. A. Fisher, who worked at Rothamsted in south-east England. To compare two variances, all you do is *divide the larger variance by the smaller variance*.

Obviously, if the variances are the same, the ratio will be 1. In order to be significantly different, the ratio will need to be significantly bigger than 1 (because the larger variance goes on top, in the numerator). How will we know a significant value of the variance ratio from a non-significant one? The answer, as always, is to look up the *critical value* of the variance ratio. In this case, we want critical values of Fisher's *F*. The R function for this is `qf` which stands for 'quantiles of the *F* distribution'. For our example of ozone levels in market gardens (see Chapter 4) there were 10 replicates in each garden, so there were $10 - 1 = 9$ degrees of freedom for each garden. In comparing two gardens, therefore, we have 9 d.f. in the numerator and 9 d.f. in the denominator. Although *F* tests in analysis of variance are

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

typically one-tailed (the treatment variance is expected to be larger than the error variance if the means are significantly different; see p. 153), in this case, we had no expectation as to which garden was likely to have the higher variance, so we carry out a two-tailed test ($p = 1 - \alpha/2$). Suppose we work at the traditional $\alpha = 0.05$, then we find the critical value of F like this:

```
qf(0.975,9,9)
```

```
4.025994
```

This means that a calculated variance ratio will need to be greater than or equal to 4.026 in order for us to conclude that the two variances are significantly different at $\alpha = 0.05$. To see the test in action, we can compare the variances in ozone concentration for market gardens B and C.

```
f.test.data <- read.csv("c:\\temp\\f.test.data.csv")
attach(f.test.data)
names(f.test.data)
```

```
[1] "gardenB" "gardenC"
```

First, we compute the two variances:

```
var(gardenB)
```

```
[1] 1.333333
```

```
var(gardenC)
```

```
[1] 14.22222
```

The larger variance is clearly in garden C, so we compute the F ratio like this:

```
F.ratio <- var(gardenC)/var(gardenB)
F.ratio
```

```
[1] 10.66667
```

The *test statistic* shows us that the variance in garden C is more than 10 times as big as the variance in garden B. The *critical value* of F for this test (with 9 d.f. in both the numerator and the denominator) is 4.026 (see `qf`, above), so we conclude: *since the test statistic is larger than the critical value, we reject the null hypothesis*.

The null hypothesis was that the two variances were not significantly different, so we accept the alternative hypothesis that the two variances are significantly different. In fact, it is better practice to present the p value associated with the calculated F ratio rather than just to reject the null hypothesis. To do this we use `pf` rather than `qf`. We double the resulting probability to allow for the two-tailed nature of the test:

```
2*(1 - pf(F.ratio,9,9))
```

```
[1] 0.001624199
```

so the probability of obtaining an F ratio as large as this or larger, if the variances were the same (as assumed by the null hypothesis), is less than 0.002. It is important to note that the p value is *not* the probability that the null hypothesis is true (this is a common mistake amongst beginners). The null hypothesis is *assumed to be true* in carrying out the test. Keep rereading this paragraph until you are sure that you understand this distinction between what p values *are* and what they *are not*.

Because the variances are significantly different, it would be wrong to compare the two sample means using Student's t test. In this case the reason is very obvious; the means are exactly the same (5.0 pphm) but the two gardens are clearly different in their daily levels of ozone pollution (see p. 55).

There is a built-in function called `var.test` for speeding up the procedure. All we provide are the names of the two variables containing the raw data whose variances are to be compared (we do not need to work out the variances first):

```
var.test(gardenB,gardenC)

        F test to compare two variances

data: gardenB and gardenC
F = 0.0938, num df = 9, denom df = 9, p-value = 0.001624
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.02328617 0.37743695
sample estimates:
ratio of variances
           0.09375
```

Note that the variance ratio, F, is given as roughly 1/10 rather than roughly 10 because `var.test` put the variable name that came first in the alphabet (`gardenB`) on top (i.e. in the numerator) instead of the bigger of the two variances. But the p value of 0.0016 is exactly the same, and we reject the null hypothesis. These two variances are highly significantly different.

```
detach(f.test.data)
```

Comparing Two Means

Given what we know about the variation from replicate to replicate within each sample (the within-sample variance), how likely is it that our two sample means were drawn from populations with the same average? If this is highly likely, then we shall say that our two sample means are not significantly different. If it is rather unlikely, then we shall say that our sample means are significantly different. As with all of the classical tests, we do this by calculating a *test statistic* and then asking the question: how likely are we to obtain a test statistic this big (or bigger) if the null hypothesis is true? We judge the probability by comparing the test statistic with a *critical value*. The critical value is calculated on the assumption that the null hypothesis is true. It is a useful feature of R that it has built-in statistical tables for all of the important probability distributions, so that if we provide R with the relevant degrees of freedom, it can tell us the critical value for any particular case.

There are two simple tests for comparing two sample means:

- *Student's t test* when the samples are independent, the variances constant, and the errors are normally distributed
- *Wilcoxon rank-sum test* when the samples are independent but the errors are *not* normally distributed (e.g. they are ranks or scores or some sort)

What to do when these assumptions are violated (e.g. when the variances are different) is discussed later on.

Student's *t* Test

Student was the pseudonym of W.S. Gosset who published his influential paper in *Biometrika* in 1908. He was prevented from publishing under his own name by dint of the archaic employment laws in place at the time, which allowed his employer, the Guinness Brewing Company, to prevent him publishing independent work. Student's t distribution, later perfected by R. A. Fisher, revolutionised the study of small-sample statistics where inferences need to be made on the basis of the sample variance s^2 with the population variance σ^2 unknown (indeed, usually unknowable).

The *test statistic* is the number of standard errors by which the two sample means are separated:

$$t = \frac{\text{difference between the two means}}{\text{SE of the difference}} = \frac{\bar{y}_\text{A} - \bar{y}_\text{B}}{SE_\text{diff}}$$

We already know the standard error of the mean (see p. 60) but we have not yet met the standard error of the difference between two means. For two independent (i.e. non-correlated) variables, *the variance of a difference is the sum of the separate variances* (see Box 6.1).

Box 6.1. The variance of a difference between two independent samples

We want to work out the sum of squares of a difference between samples A and B. First we express each y variable as a departure from its own mean, μ:

$$\sum \left[(y_\text{A} - \mu_\text{A}) - (y_\text{B} - \mu_\text{B})\right]^2$$

If we were to divide by the degrees of freedom, we would get the variance of the difference, $\sigma^2_{\bar{y}_\text{A} - \bar{y}_\text{B}}$. Start by calculating the square of the difference:

$$(y_\text{A} - \mu_\text{A})^2 + (y_\text{B} - \mu_\text{B})^2 - 2(y_\text{A} - \mu_\text{A})(y_\text{B} - \mu_\text{B})$$

Then apply summation:

$$\sum (y_A - \mu_A)^2 + \sum (y_B - \mu_B)^2 - 2\sum (y_A - \mu_A)(y_B - \mu_B)$$

We already know that the average of $\sum (y_A - \mu_A)^2$is the variance of population A and the average of $\sum (y_B - \mu_B)^2$ is the variance of population B (Box 4.2). So the variance of the *difference* between the two sample means is the *sum* of the variances of the two samples, minus a term $2\sum (y_A - \mu_A)(y_B - \mu_B)$, i.e. minus 2 times the covariance of samples A and B (see Box 6.2). But because (by assumption) the samples from A and B are independently drawn they are uncorrelated, the covariance is zero, so $2\sum (y_A - \mu_A)(y_B - \mu_B) = 0$. This important result needs to be stated separately:

$$\sigma^2_{\bar{y}_A - \bar{y}_B} = \sigma^2_A + \sigma^2_B$$

If two samples are independent, *the variance of the difference is the sum of the two sample variances*. This is *not* true, of course, if the samples are positively or negatively correlated (see p. 108).

This important result allows us to write down the formula for the standard error of the difference between two sample means:

$$SE_{\text{diff}} = \sqrt{\frac{s^2_A}{n_A} + \frac{s^2_B}{n_B}}$$

At this stage we have everything we need to carry out Student's *t* test. Our null hypothesis is that the two sample means are the same, and we shall accept this unless the value of Student's *t* is sufficiently large that it is unlikely that such a difference arose by chance alone. We shall estimate this probability by comparing our test statistic with the critical value from Student's *t* distribution with the appropriate number of degrees of freedom.

For our ozone example, each sample has 9 degrees of freedom, so we have 18 d.f. in total. Another way of thinking of this is to reason that the complete sample size as 20, and we have estimated two parameters from the data, $\bar{y}_A$ and $\bar{y}_B$, so we have $20 - 2 = 18$ d.f. We typically use 5% as the chance of rejecting the null hypothesis when it is true (this is the Type I error rate). Because we did not know in advance which of the two gardens was going to have the higher mean ozone concentration (and we usually do not), this is a two-tailed test, so the *critical value* of Student's *t* is:

```
qt(0.975,18)
[1] 2.100922
```

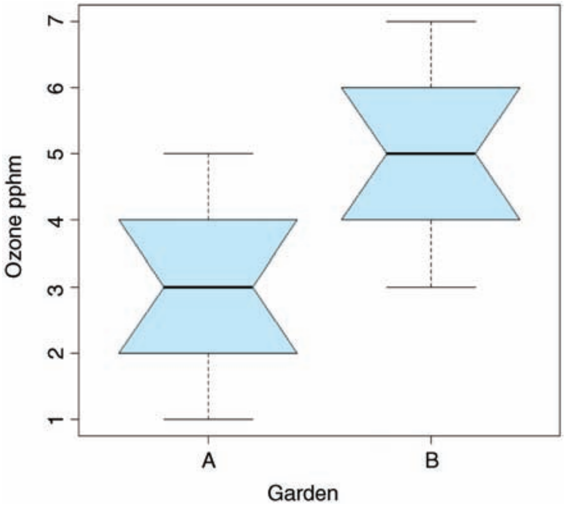
This means that our test statistic needs to be bigger than 2.1 in order to reject the null hypothesis, and hence to conclude that the two means are significantly different at $\alpha = 0.05$. The data frame is attached like this:

```
t.test.data <- read.csv("c:\\temp\\t.test.data.csv")
attach(t.test.data)
names(t.test.data)
```

```
[1] "gardenA" "gardenB"
```

A useful graphical test for two samples employs the 'notches' option of `boxplot`:

```
ozone <- c(gardenA,gardenB)
label <- factor(c(rep("A",10),rep("B",10)))
boxplot(ozone~label,notch=T,xlab="Garden",
                              ylab="Ozone pphm",col="lightblue")
```



Because *the notches of two plots do not overlap*, we conclude that the medians are significantly different at the 5% level. Note that the variability is similar in both gardens (both in terms of the range – the length of the whiskers – and the interquartile range – the size of the boxes).

To carry out a *t* test long-hand, we begin by calculating the variances of the two samples, `s2A` and `s2B`:

```
s2A <- var(gardenA)
s2B <- var(gardenB)
```

We need to check that the two variances are not significantly different:

```
s2A/s2B
```

```
[1] 1
```

They are identical, which is excellent. Constancy of variance is the most important assumption of the *t* test. In general, the value of the test statistic for Student's *t* is *the difference divided by the standard error of the difference*.

In our case, the numerator is the difference between the two means ($3-5=-2$), and the denominator is the square root of the sum of the variances (1.333333) divided by their sample sizes (10):

```
(mean(gardenA)-mean(gardenB))/sqrt(s2A/10+s2B/10)
```

Note that in calculating the standard errors we divide by the sample size (10) not the degrees of freedom (9); degrees of freedom were used in calculating the variances (see p. 53).

This gives the value of Student's *t* as

```
[1] -3.872983
```

With *t* tests you can ignore the minus sign; it is only the absolute value of the difference between the two sample means that concerns us. So the calculated value of the test statistic is 3.87 and the critical value is 2.10 (`qt(0.975,18)`, above). Since the test statistic is larger than the critical value, we reject the null hypothesis.

Notice that the wording of that previous sentence is exactly the same as it was for the *F* test (above). Indeed, the wording is always the same for all kinds of tests, and you should try to memorize it. The abbreviated form is easier to remember: *larger reject, smaller accept*. The null hypothesis was that the two means are not significantly different, so we reject this and accept the alternative hypothesis that *the two means are significantly different*. Again, rather than merely rejecting the null hypothesis, it is better to state the probability that a test statistic as extreme as this (or more extreme) would be observed if the null hypothesis was true (i.e. the mean values were not significantly different). For this we use `pt` rather than `qt`, and 2 × `pt` because we are doing a two-tailed test:

```
2*pt(-3.872983,18)
[1] 0.001114540
```

so $p < 0.0015$. You will not be surprised to learn that there is a built-in function to do all the work for us. It is called, helpfully, `t.test` and is used simply by providing the names of the two vectors containing the samples on which the test is to be carried out (`gardenA` and `gardenB` in our case).

```
t.test(gardenA,gardenB)
```

There is rather a lot of output. You often find this: the simpler the statistical test, the more voluminous the output.

```
        Welch Two Sample t-test
data:  gardenA and gardenB
```

```
t = -3.873, df = 18, p-value = 0.001115
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.0849115 -0.9150885
sample estimates:
mean of x mean of y
        3         5
```

The result is exactly the same as we obtained long-hand. The value of t is −3.873 and since *the sign is irrelevant in a t test* we reject the null hypothesis because the test statistic is larger than the critical value of 2.1. The mean ozone concentration is significantly higher in garden B than in garden A. The output also gives a p value and a confidence interval. Note that, because the means are significantly different, *the confidence interval on the difference does not include zero* (in fact, it goes from −3.085 up to −0.915). You might present the result like this:

> Ozone concentration was significantly higher in garden B (mean = 5.0 pphm) than in garden A (mean = 3.0 pphm; $t = 3.873$, $p = 0.0011$ (two-tailed), d.f. = 18).

which gives the reader all of the information they need to come to a conclusion about the size of the effect and the unreliability of the estimate of the size of the effect.

Wilcoxon Rank-Sum Test

This is a non-parametric alternative to Student's t test, which we could use if the errors were non-normal. The Wilcoxon rank-sum test statistic, W, is calculated as follows. Both samples are put into a single array with their sample names clearly attached (A and B in this case, as explained below). Then the aggregate list is sorted, taking care to keep the sample labels with their respective values. A rank is then assigned to each value, with ties getting the appropriate average rank (two-way ties get (rank i + (rank i + 1))/2, three-way ties get (rank i + (rank i + 1) + (rank i + 3))/3, and so on). Finally, the ranks are added up for each of the two samples, and significance is assessed on size of the smaller sum of ranks.

First we make a combined vector of the samples:

```
ozone <- c(gardenA,gardenB)
ozone
```

```
[1] 3 4 4 3 2 3 1 3 5 2 5 5 6 7 4 4 3 5 6 5
```

Then we make a list of the sample names, A and B:

```
label <- c(rep("A",10),rep("B",10))
label
```

```
[1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B"
```

Now we use the built-in function rank to get a vector containing the ranks, smallest to largest, within the combined vector:

```
combined.ranks <- rank(ozone)
combined.ranks
```

```
 [1]  6.0 10.5 10.5  6.0  2.5  6.0  1.0  6.0 15.0  2.5 15.0 15.0 18.5 20.0 10.5
[16] 10.5  6.0 15.0 18.5 15.0
```

Notice that the ties have been dealt with by averaging the appropriate ranks. Now all we need to do is calculate the sum of the ranks for each garden. We use tapply with sum as the required operation:

```
tapply(combined.ranks,label,sum)
```

```
  A   B
 66 144
```

Finally, we compare the smaller of the two values (66) with values in tables of Wilcoxon rank sums (e.g. Snedecor and Cochran, 1980, p. 555), and reject the null hypothesis if our value of 66 is *smaller* than the value in tables. For samples of size 10 and 10 like ours, the 5% value in tables is 78. Our value is smaller than this, so we reject the null hypothesis. The two sample means are significantly different (in agreement with our earlier *t* test on the same data).

We can carry out the whole procedure automatically, and avoid the need to use tables of critical values of Wilcoxon rank sums, by using the built-in function wilcox.test:

```
wilcox.test(gardenA,gardenB)
```

This produces the following output:

```
        Wilcoxon rank sum test with continuity correction

data:  gardenA and gardenB
W = 11, p-value = 0.002988
alternative hypothesis: true location shift is not equal to 0

Warning message:
In wilcox.test.default(gardenA, gardenB) :
  cannot compute exact p-value with ties
```

The function uses a normal approximation algorithm to work out a *z* value, and from this a *p* value to assess the hypothesis that the two means are the same. This *p* value of 0.002988 is much less than 0.05, so we reject the null hypothesis, and conclude that the mean ozone concentrations in gardens A and B are significantly different. The warning message at the end draws attention to the fact that there are ties in the data (repeats of the same ozone measurement), and this means that the *p* value cannot be calculated exactly (this is seldom a real worry).

It is interesting to compare the *p* values of the *t* test and the Wilcoxon test with the same data: $p = 0.001115$ and 0.002988, respectively. The non-parametric test is much more appropriate than the *t*-test when the errors are not normal, and the non-parametric test is about 95% as powerful with normal errors, and can be *more* powerful than the *t* test if the distribution is strongly skewed by the presence of outliers. Typically, as here, the *t* test will give the lower *p* value, so the Wilcoxon test is said to be conservative: if a difference is significant under a Wilcoxon test it would be even more significant under a *t* test.

Tests on Paired Samples

Sometimes, two-sample data come from paired observations. In this case, we might expect a correlation between the two measurements, either because they were made on the same individual, or because they were taken from the same location. You might recall that earlier (Box 6.1) we found that the variance of a difference was the average of

$$(y_A - \mu_A)^2 + (y_B - \mu_B)^2 - 2(y_A - \mu_A)(y_B - \mu_B)$$

which is the variance of sample A, plus the variance of sample B, minus 2 times the covariance of A and B. When the covariance of A and B is *positive*, this is a great help because it reduces the variance of the difference, which makes it easier to detect significant differences between the means. Pairing is not always effective, because the correlation between y_A and y_B may be weak.

The following data are a composite biodiversity score based on a kick sample of aquatic invertebrates from 16 rivers:

```
streams <- read.csv("c:\\temp\\streams.csv")
attach(streams)
names(streams)
```

```
[1] "down" "up"
```

The elements are paired because the two samples were taken on the same river, one upstream and one downstream from the same sewage outfall. If we ignore the fact that the samples are paired, it appears that the sewage outfall has no impact on biodiversity score ($p = 0.6856$):

```
t.test(down,up)
```

```
        Welch Two Sample t-test

data: down and up
t = -0.4088, df = 29.755, p-value = 0.6856
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -5.248256 3.498256
sample estimates:
mean of x mean of y
   12.500  13.375
```

However, if we allow that the samples are paired (simply by specifying the option `paired=T`), the picture is completely different:

```
t.test(down,up,paired=T)

        Paired t-test

data: down and up
t = -3.0502, df = 15, p-value = 0.0081
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.4864388 -0.2635612
sample estimates:
mean of the differences
                -0.875
```

Now the difference between the means is highly significant ($p = 0.0081$). The moral is clear. If you can do a paired *t* test, then you should always do the paired test. It can never do any harm, and sometimes (as here) it can do a huge amount of good. In general, if you have information on *blocking* or *spatial correlation* (in this case, the fact that the two samples came from the same river), then you should always use it in the analysis.

Here is the same paired test carried out as a one-sample *t*-test on the *differences* between the pairs:

```
d <- up-down
t.test(d)

        One Sample t-test

data: d
t = 3.0502, df = 15, p-value = 0.0081
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.2635612 1.4864388
sample estimates:
mean of x
    0.875
```

As you see, the result is identical to the two-sample *t* test with `paired=T` ($p = 0.0081$).

The upstream values of the biodiversity score were greater by 0.875 on average, and this difference is highly significant. Working with the differences has halved the number of degrees of freedom (from 30 to 15), but it has more than compensated for this by reducing the error variance, because there is such a strong positive correlation between y_A and y_B. The moral is simple: *blocking always helps*. The individual rivers are the blocks in this case.

The Binomial Test

This is one of the simplest of all statistical tests. Suppose that you cannot *measure* a difference, but you can *see* it (e.g. in judging a diving contest). For example, nine springboard divers were scored as better or worse, having trained under a new regime and under the conventional regime (the regimes were allocated in a randomized sequence to each athlete: new then conventional, or conventional then new). Divers were judged twice: one diver was worse on the new regime, and eight were better. What is the evidence that the new regime produces significantly better scores in competition? The answer comes from a two-tailed binomial test. How likely is a response of 1/9 (or 8/9 or more extreme than this, i.e. 0/9 or 9/9) if the populations are actually the same (i.e. there was no difference between the two training regimes)? We use `binom.test` for this, specifying the number of 'failures' (1) and the total sample size (9):

```
binom.test(1,9)
```

This produces the output:

```
        Exact binomial test

data: 1 and 9
number of successes = 1, number of trials = 9, p-value = 0.03906
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.002809137 0.482496515
sample estimates:
probability of success
             0.1111111
```

From this we would conclude that the new training regime is significantly better than the traditional method, because $p < 0.05$. The p value of 0.03906 is the exact probability of obtaining the observed result (1 of 9) or a more extreme result (0 of 9) if the two training regimes were the same (so the probability of any one of the judging outcomes was 0.5). If the two regimes had identical effects on the judges, then a score of either 'better' or 'worse' has a probability of 0.5. So eight successes has a probability of $0.5^8 = 0.0039$ and one failure has a probability of 0.5. The whole outcome has a probability of

$$0.00390625 \times 0.5 = 0.001953125$$

Note, however, that there are 9 ways of getting this result, so the probability of 1 out of 9 is

$$9 \times 0.001953125 = 0.01757812$$

This is not the answer we want, because there is a more extreme case when all 9 of the judges thought that the new regime produced improved scores. There is only one way of obtaining this outcome (9 successes and no failures) so it has a probability of $0.5^9 = 0.001953125$. This means that the probability the observed result or a more extreme outcome is

$$0.001953125 + 0.01757812 = 0.01953124$$

Even this is not the answer we want, because the whole process might have worked in the opposite direction (the new regime might have produced worse scores (8 and 1 or 9 and 0) so what we need is a two-tailed test). The result we want is obtained simply by doubling the last result:

$$2 \times 0.01953124 = 0.03906248$$

which is the figure produced by the built in function `binom.test` (above)

Binomial Tests to Compare Two Proportions

Suppose that in your institution only four females were promoted, compared with 196 men. Is this an example of blatant sexism, as it might appear at first glance? Before we can judge, of course, we need to know the number of male and female candidates. It turns out that 196 men were promoted out of 3270 candidates, compared with 4 promotions out of only 40 candidates for the women. Now, if anything, it looks like the females did better than males in the promotion round (10% success for women versus 6% success for men).

The question then arises as to whether the apparent positive discrimination in favour of women is statistically significant, or whether this sort of difference could arise through chance alone. This is easy in R using the built-in binomial proportions test `prop.test` in which we specify two vectors, the first containing the number of successes for females and males `c(4,196)` and second containing the *total* number of female and male candidates `c(40,3270)`:

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

        2-sample test for equality of proportions with continuity correction

data: c(4, 196) out of c(40, 3270)
X-squared = 0.5229, df = 1, p-value = 0.4696
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.06591631 0.14603864
sample estimates:
    prop 1      prop 2
0.10000000 0.05993884

Warning message:
In prop.test(c(4, 196), c(40, 3270)) :
  Chi-squared approximation may be incorrect
```

There is no evidence in favour of positive discrimination ($p = 0.4696$). A result like this will occur more than 45% of the time by chance alone. Just think what would have happened if one of the successful female candidates had not applied. Then the same promotion system would have produced a female success rate of 3/39 instead of 4/40 (7.7% instead of 10%).

The moral is very important: *in small samples, small changes have big effects*.

Chi-Squared Contingency Tables

A great deal of statistical information comes in the form of *counts* (whole numbers or integers); the number of animals that died, the number of branches on a tree, the number of days of frost, the number of companies that failed, the number of patients that died. With count data, the number 0 is often the value of a response variable (consider, for example, what a 0 would mean in the context of the examples just listed).

The dictionary definition of contingency is ‘a thing dependent on an uncertain event’ (*Oxford English Dictionary*). In statistics, however, the contingencies are *all the events that could possibly happen*. A contingency table shows the counts of how many times each of the contingencies actually happened in a particular sample. Consider the following example that has to do with the relationship between hair colour and eye colour in white people.

For simplicity, we just chose two contingencies for hair colour: ‘fair’ and ‘dark’. Likewise we just chose two contingencies for eye colour: ‘blue’ and ‘brown’. Each of these two categorical variables, eye colour and hair colour, has two levels (‘blue’ and ‘brown’ and ‘fair’ and ‘dark’, respectively). Between them, they define four possible outcomes (the contingencies): fair hair and blue eyes, fair hair and brown eyes, dark hair and blue eyes, and dark hair and brown eyes. We take a sample of people and count how many of them fall into each of these four categories. Then we fill in the 2 × 2 contingency table like this:

	Blue eyes	Brown eyes
Fair hair	38	11
Dark hair	14	51

These are our observed frequencies (or counts). The next step is very important. In order to make any progress in the analysis of these data we need a *model* which predicts the expected frequencies. What would be a sensible model in a case like this? There are all sorts of complicated models that you might select, but the simplest model (Occam’s razor, or the principle of parsimony) is that hair colour and eye colour are *independent*. We may not believe that this is actually true, but the hypothesis has the great virtue of being falsifiable. It is also a very sensible model to choose because it makes it easy to predict the expected frequencies based on the assumption that the model is true. We need to do some simple probability work. What is the probability of getting a random individual from this sample whose hair was fair? A total of 49 people (38 + 11) had fair hair out of a total sample of 114 people. So the probability of fair hair is 49/114 and the probability of dark hair is 65/114. Notice that because we have only two levels of hair colour, these two probabilities add up to 1 ((49 + 65)/114). What about eye colour? What is the probability of selecting someone at random from this sample with blue eyes? A total of 52 people had blue eyes (38 + 14) out of the sample of 114, so the probability of blue eyes is 52/114 and the probability of brown eyes is 62/114. As before, these sum to 1 ((52 + 62)/114). It helps to append the subtotals to the margins of the contingency table like this:

	Blue eyes	Brown eyes	Row totals
Fair hair	38	11	49
Dark hair	14	51	65
Column totals	52	62	114

Now comes the important bit. We want to know the expected frequency of people with fair hair *and* blue eyes, to compare with our observed frequency of 38. Our model says that the two are independent. This is essential information, because it allows us to calculate the expected probability of fair hair and blue eyes. *If, and only if, the two traits are independent, then the probability of having fair hair and blue eyes is the product of the two probabilities.* So, following our earlier calculations, the probability of fair hair and blue eyes is 49/114 × 52/114. We can do exactly equivalent things for the other three cells of the contingency table:

	Blue eyes	Brown eyes	Row totals
Fair hair	$\frac{49}{114} \times \frac{52}{114}$	$\frac{49}{114} \times \frac{62}{114}$	49
Dark hair	$\frac{65}{114} \times \frac{52}{114}$	$\frac{65}{114} \times \frac{62}{114}$	65
Column totals	52	62	114

Now we need to know how to calculate the expected frequency. It could not be simpler. It is just the probability multiplied by the total sample ($n = 114$). So the expected frequency of blue eyes and fair hair is $\frac{49}{114} \times \frac{52}{114} \times 114 = 22.35$ which is much less than our observed frequency of 38. It is beginning to look as if our hypothesis of independence of hair and eye colour is false.

You might have noticed something useful in the last calculation: two of the sample sizes cancel out. Therefore, the expected frequency in each cell is just the row total (R) times the column total (C) divided by the grand total (G):

$$E = \frac{R \times C}{G}$$

We can now work out the four expected frequencies.

	Blue eyes	Brown eyes	Row totals
Fair hair	22.35	26.65	49
Dark hair	29.65	35.35	65
Column totals	52	62	114

Notice that the row and column totals (the so-called ‘marginal totals’) are retained under the model. It is clear that the observed frequencies and the expected frequencies are different. But in sampling, everything always varies, so this is no surprise. The important question is whether the expected frequencies are *significantly* different from the observed frequencies.

We assess the significance of the differences between using a chi-squared test. We calculate a test statistic χ^2 (Pearson’s chi-squared) as follows:

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

where O is the observed frequency and E is the expected frequency. Capital Greek sigma $\sum$ just means ‘add up all the values of’. It makes the calculations easier if we write the observed and expected frequencies in parallel columns, so that we can work out the corrected squared differences more easily.

	O	E	$(O-E)^2$	$\frac{(O-E)^2}{E}$
Fair hair and blue eyes	38	22.35	244.92	10.96
Fair hair and brown eyes	11	26.65	244.92	9.19
Dark hair and blue eyes	14	29.65	244.92	8.26
Dark hair and brown eyes	51	35.35	244.92	6.93

All we need to do now is to add up the four components of chi-squared to get the test statistic $\chi^2 = 35.33$. The question now arises: is this a big value of chi-squared or not? This is important, because if it *is* a bigger value of chi-squared than we would expect by chance, then we should reject the null hypothesis. If, on the other hand, it is within the range of values that we would expect by chance alone, then we should accept the null hypothesis.

We always proceed in the same way at this stage. We have a calculated value of the test statistic: $\chi^2 = 35.33$. We *compare this value of the test statistic with the relevant critical value.* To work out the critical value of chi-squared we need two things:

- the number of degrees of freedom
- the degree of certainty with which to work

In general, a contingency table has a number of rows (r) and a number of columns (c), and the degrees of freedom is given by

$$\text{d.f.} = (r-1) \times (c-1)$$

So we have $(2-1)\times(2-1) = 1$ degree of freedom for a 2×2 contingency table. You can see why there is only one degree of freedom by working through our example. Take the 'fair hair and brown eyes' box (the top right in the table) and ask how many values this could possibly take. The first thing to note is that the count could not be more than 49, otherwise the row total would be wrong. But in principle, the number in this box is free to be any value between 0 and 49. We have one degree of freedom for this box. But when we have fixed this box to be 11, you will see that we have no freedom at all for any of the other three boxes.

	Blue eyes	Brown eyes	Row totals
Fair hair		11	49
Dark hair			65
Column totals	52	62	114

The top left box has to be $49 - 11 = 38$ because the row total is fixed at 49. Once the top left box is defined as 38 then the bottom left box has to be $52 - 38 = 14$ because the column total is fixed (the total number of people with blue eyes was 52). This means that the bottom right box has to be $65 - 14 = 51$. Thus, *because the marginal totals are constrained*, a 2×2 contingency table has just 1 degree of freedom.

The next thing we need to do is say how certain we want to be about the falseness of the null hypothesis. The more certain we want to be, the larger the value of chi-squared we would need to reject the null hypothesis. It is conventional to work at the 95% level. That is our certainty level, so our uncertainty level is $100 - 95 = 5\%$. Expressed as a fraction, this is called alpha ($\alpha = 0.05$). Technically, alpha is the probability of *rejecting* the null hypothesis when it is *true*. This is called a Type I error. A Type II error is *accepting* the null hypothesis when it is *false* (see p. 4).

Critical values in R are obtained by use of *quantiles* (q) of the appropriate statistical distribution. For the chi-squared distribution, this function is called qchisq. The function has two arguments: the certainty level ($1 - \alpha = 0.95$), and the degrees of freedom (d.f. = 1):

```
qchisq(0.95,1)
```

```
[1] 3.841459
```

The critical value of chi-squared is 3.841.The logic goes like this: since the calculated value of the test statistic is *greater* than the critical value, we *reject* the null hypothesis. You should memorize this sentence: put the emphasis on ‘greater’ and ‘reject’.

What have we learned so far? We have rejected the null hypothesis that eye colour and hair colour are independent. But that is not the end of the story, because we have not established the *way* in which they are related (e.g. is the correlation between them positive or negative?). To do this we need to look carefully at the data, and compare the observed and expected frequencies. If fair hair and blue eyes were positively correlated, would the observed frequency of getting this combination be greater or less than the expected frequency? A moment’s thought should convince you that the observed frequency will be greater than the expected frequency when the traits are positively correlated (and less when they are negatively correlated). In our case we expected only 22.35 but we observed 38 people (nearly twice as many) to have both fair hair and blue eyes. So it is clear that fair hair and blue eyes are *positively* associated.

In R the procedure is very straightforward. We start by defining the counts as a 2×2 matrix like this:

```
count <- matrix(c(38,14,11,51),nrow=2)
count
```

```
     [,1] [,2]
[1,]   38   11
[2,]   14   51
```

Notice that you enter the data *column-wise* (not row-wise) into the matrix. Then the test uses the chisq.test function, with the matrix of counts as its only argument:

```
chisq.test(count)

        Pearson's Chi-squared test with Yates' continuity correction

data: count
X-squared = 33.112, df = 1, p-value = 8.7e-09
```

The calculated value of chi-squared is slightly different from ours, because Yates's correction has been applied as the default (see `?chisq.test`). If you switch the correction off (`correct=F`), you get the exact value we calculated by hand:

```
chisq.test(count,correct=F)

        Pearson's Chi-squared test

data: count
X-squared = 35.3338, df = 1, p-value = 2.778e-09
```

It makes no difference at all to the interpretation: there is a highly significant positive association between fair hair and blue eyes for this group of people.

Fisher's Exact Test

This test is used for the analysis of contingency tables based on small samples in which *one or more of the expected frequencies are less than 5*. The individual counts are *a*, *b*, *c* and *d* like this:

2 × 2 table	Column 1	Column 2	Row totals
Row 1	a	b	$a+b$
Row 2	c	d	$c+d$
Column totals	$a+c$	$b+d$	n

The probability of any one particular outcome is given by

$$p = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{a!b!c!d!n!}$$

where n is the grand total, and ! means 'factorial' (the product of all the numbers from n down to 1; 0! is defined as being 1).

Our data concern the distribution of 8 ants nests over 10 trees of each of two species (A and B). There are two categorical explanatory variables (ants and trees), and four contingencies, ants (present or absent) and trees (A or B). The response variable is the vector of four counts `c(6,4,2,8)`.

	Tree A	Tree B	Row totals
With ants	6	2	8
Without ants	4	8	12
Column totals	10	10	20

Now we can calculate the probability for this particular outcome:

```
factorial(8)*factorial(12)*factorial(10)*factorial(10)/
  (factorial(6)*factorial(2)*factorial(4)*factorial(8)*factorial(20))
```

```
[1] 0.07501786
```

But this is only part of the story. We need to compute the probability of outcomes that are *more extreme* than this. There are two of them. Suppose only one ant colony was found on tree B. Then the table values would be 7, 1, 3, 9 but the row and column totals would be exactly the same (*the marginal totals are constrained*). The numerator always stays the same, so this case has probability

```
factorial(8)*factorial(12)*factorial(10)*factorial(10)/
  (factorial(7)*factorial(3)*factorial(1)*factorial(9)*factorial(20))
```

```
[1] 0.009526078
```

There is an even more extreme case if no ant colonies at all were found on tree B. Now the table elements become 8, 0, 2, 10 with probability

```
factorial(8)*factorial(12)*factorial(10)*factorial(10)/
  (factorial(8)*factorial(2)*factorial(0)*factorial(10)*factorial(20))
```

```
[1] 0.0003572279
```

We need to add these three probabilities together:

```
0.07501786 + 0.009526078 + 0.000352279
```

```
[1] 0.08489622
```

But there was no *a priori* reason for expecting the result to be in this direction. It might have been tree A that had relatively few ant colonies. We need to allow for extreme counts in the opposite direction by doubling this probability (all Fisher's exact tests are two-tailed):

```
2*(0.07501786+0.009526078+0.000352279)
```

```
[1] 0.1697924
```

This shows that there is no strong evidence of a correlation between tree and ant colonies. The observed pattern, or a more extreme one, could have arisen by chance alone with probability $p = 0.17$.

There is a built in function called fisher.test, which saves us all this tedious computation. It takes as its argument a 2×2 matrix containing the counts of the four contingencies. We make the matrix like this (compare with the alternative method of making a matrix, above):

```
x <- as.matrix(c(6,4,2,8))
dim(x) <- c(2,2)
x

        [,1]  [,2]
[1,]      6     2
[2,]      4     8
```

Then we run the test like this:

```
fisher.test(x)

        Fisher's Exact Test for Count Data

data: x
p-value = 0.1698
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
  0.6026805 79.8309210
sample estimates:
odds ratio
  5.430473
```

The fisher.test can be used with matrices much bigger than 2×2. Alternatively, the function may be provided with two vectors containing factor levels, instead of a matrix of counts, as here; this saves you the trouble of counting up how many combinations of each factor level there are:

```
table <- read.csv("c:\\temp\\fisher.csv")
attach(table)
head(table)

  tree nests
1    A  ants
2    B  ants
3    A  none
4    A  ants
5    B  none
6    A  none
```

The function is invoked by providing the two vector names as arguments:

```
fisher.test(tree,nests)
```

Correlation and Covariance

With two continuous variables, x and y, the question naturally arises as to whether their values are correlated with each other (remembering, of course, that *correlation does not imply causation*). Correlation is defined in terms of the variance of x, the variance of y, and the covariance of x and y (the way the two vary together; the way they co-vary) on the assumption that both variables are normally distributed. We have symbols already for the two variances, s_x^2 and s_y^2. We denote the covariance of x and y by cov(x, y), after which the correlation coefficient r is defined as

$$r = \frac{\operatorname{cov}(x, y)}{\sqrt{s_x^2 s_y^2}}$$

We know how to calculate variances (p. 53), so it remains only to work out the value of the covariance of x and y. Covariance is defined as *the expectation of the vector product* $x \times y$, which sounds difficult, but is not (Box 6.2). The covariance of x and y is *the expectation of the product minus the product of the two expectations*.

Box 6.2. Correlation and covariance

The correlation coefficient is defined in terms of the covariance of x and y, and the geometric mean of the variances of x and y:

$$\rho(x, y) = \frac{\operatorname{cov}(x, y)}{\sqrt{\operatorname{var}(x) \times \operatorname{var}(y)}}$$

The covariance of x and y is defined as the expectation of the vector product: $(x - \bar{x})(y - \bar{y})$:

$$\operatorname{cov}(x, y) = \mathrm{E}[(x - \bar{x})(y - \bar{y})]$$

We start by multiplying through the brackets:

$$(x - \bar{x})(y - \bar{y}) = xy - \bar{x}y - x\bar{y} + \overline{xy}$$

Now applying expectations, and remembering that the expectation of x is $\bar{x}$ and the expectation of y is $\bar{y}$, we get

$$\operatorname{cov}(x, y) = \mathrm{E}(xy) - \bar{x}\mathrm{E}(y) - \mathrm{E}(x)\bar{y} + \overline{xy} = \mathrm{E}(xy) - \overline{xy} - \overline{xy} + \overline{xy}$$

Then $-\overline{xy}+\overline{xy}$ cancels out, leaving $-\overline{xy}$ which is $-\mathrm{E}(x)\mathrm{E}(y)$ so

$$\mathrm{cov}(x, y) = \mathrm{E}(xy) - \mathrm{E}(x)\mathrm{E}(y)$$

Notice that when x and y are uncorrelated, $\mathrm{E}(xy) = \mathrm{E}(x)\mathrm{E}(y)$, so the covariance is 0 in this case. The corrected sum of products $SSXY$ (see p. 123) is given by

$$SSXY = \sum xy - \frac{\sum x \sum y}{n}$$

so covariance is computed as

$$\mathrm{cov}(x, y) = SSXY\sqrt{\frac{1}{(n-1)^2}}$$

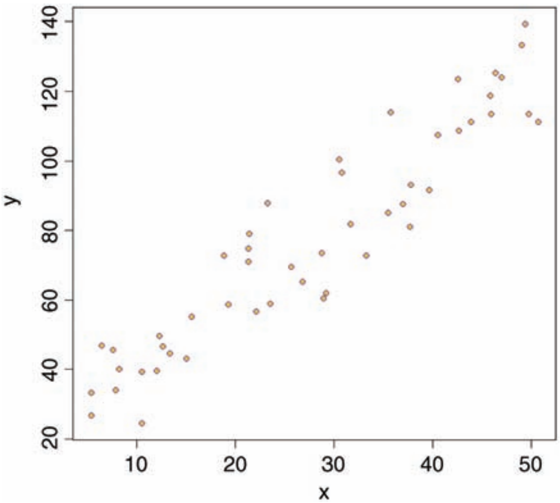
which provides a shortcut formula for the correlation coefficient

$$r = \frac{SSXY}{\sqrt{SSX.SSY}}$$

because the degrees of freedom $(n - 1)$ cancel out. The sign of r takes the sign of $SSXY$: positive for positive correlations and negative for negative correlations.

Let us work through a numerical example:

```
data <- read.csv("c:\\temp\\twosample.csv")
attach(data)
plot(x,y,pch=21,col="blue",bg="orange")
```



First we need the variance of x and the variance of y:

```
var(x)
[1] 199.9837
var(y)
[1] 977.0153
```

The covariance of x and y, cov(x,y), is given by the `var` function when we supply it with two vectors like this:

```
var(x,y)
[1] 414.9603
```

Thus, the correlation coefficient should be $414.96/\sqrt{199.98 \times 977.02}$:

```
var(x,y)/sqrt(var(x)*var(y))
[1] 0.9387684
```

Let us see if this checks out:

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

Great relief! So now you know the definition of the correlation coefficient: it is the covariance divided by the geometric mean of the two variances.

Correlation and the Variance of Differences between Variables

Samples often exhibit positive correlations that result from the pairing, as in the upstream and downstream invertebrate biodiversity data that we investigated earlier (p. 97). There is an important general question about the effect of correlation on the variance of differences between variables. In the extreme, when two variables are so perfectly correlated that they are identical, the difference between one variable and the other is zero. So it is clear that the variance of a difference will decline as the strength of positive correlation increases.

These data show the depth of the water table (m below the surface) in winter and summer at nine locations:

```
paired <- read.csv("c:\\temp\\water.table.csv ")
attach(paired)
names(paired)
[1] "Location" "Summer"  "Winter"
```

We begin by asking whether there is a correlation between summer and winter water table depths across locations:

```
cor(Summer, Winter)
```

```
 [1] 0.8820102
```

There is a strong positive correlation. Not surprisingly, places where the water table is high in summer tend to have a high water table in winter as well. If you want to determine the significance of a correlation (i.e. the p value associated with the calculated value of r) then use `cor.test` rather than `cor`. This test has non-parametric options for Kendall's tau or Spearman's rank depending on the method you specify (`method="k"` or `method="s"`), but the default method is Pearson's product-moment correlation (`method="p"`):

```
cor.test(Summer, Winter)
```

```
        Pearson's product-moment correlation

data: Summer and Winter
t = 4.9521, df = 7, p-value = 0.001652
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.5259984 0.9750087
sample estimates:
     cor
0.8820102
```

The correlation is highly significant ($p = 0.00165$). Now, let us investigate the relationship between the correlation coefficient and the three variances: the summer variance, the winter variance, and *the variance of the differences* (summer–winter):

```
varS <- var(Summer)
varW <- var(Winter)
varD <- var(Summer-Winter)
```

The correlation coefficient ρ is related to these three variances by:

$$\rho = \frac{\sigma_y^2 + \sigma_z^2 - \sigma_{y-z}^2}{2\sigma_y\sigma_z}$$

So, using the values we have just calculated, we get a correlation coefficient of

```
(varS+varW-varD)/(2*sqrt(varS)*sqrt(varW))
```

```
[1] 0.8820102
```

which checks out. We can also see whether the variance of the difference is equal to the sum of the component variances (as we saw for independent variables on p. 97):

```
varD
```

```
[1] 0.01015
```

```
varS + varW
```

```
[1] 0.07821389
```

No, it is not. They would be equal only if the two samples were independent. In fact, we know that the two variables are positively correlated, so the variance of the difference should be *less* than the sum of the variances by an amount equal to $2 \times r \times s_1 \times s_2$:

```
varS + varW - 2 * 0.8820102 * sqrt(varS) * sqrt(varW)
```

```
 [1] 0.01015
```

That's more like it.

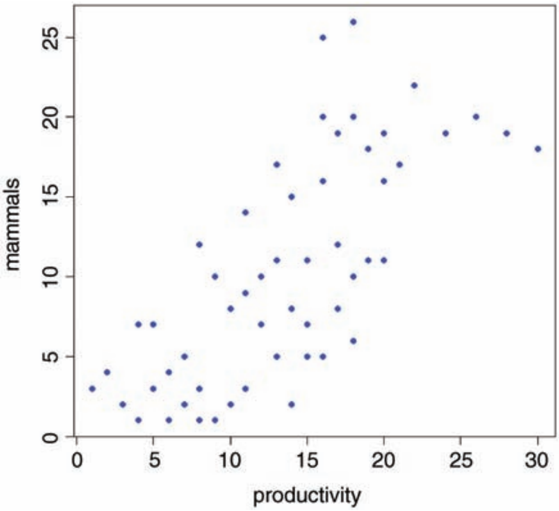
Scale-Dependent Correlations

Another major difficulty with correlations is that scatterplots can give a highly misleading impression of what is going on. The moral of this exercise is very important: *things are not always as they seem*. The following data show the number of species of mammals in forests of differing productivity:

```
data <- read.csv("c:\\temp\\productivity.csv")
attach(data)
names(data)
```

```
[1] "productivity" "mammals"   "region"
```

```
plot(productivity,mammals,pch=16,col="blue")
```

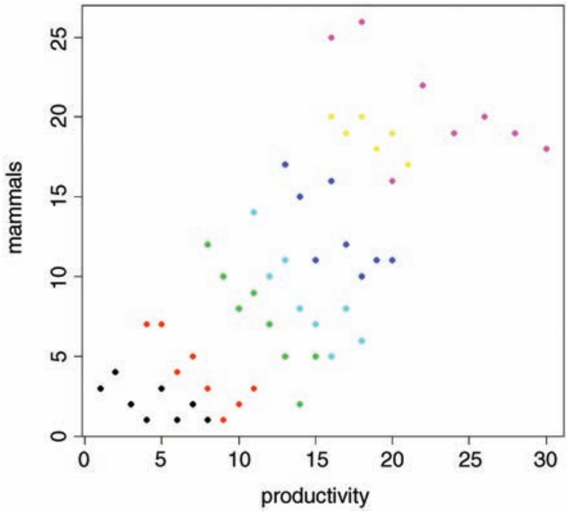


There is a very clear positive correlation: increasing productivity is associated with increasing species richness. The correlation is highly significant:

```
cor.test(productivity,mammals,method="spearman")

        Spearman's rank correlation rho

data: productivity and mammals
S = 6515.754, p-value = 5.775e-11
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.7516389

Warning message:
In cor.test.default(productivity, mammals, method = "spearman") :
  Cannot compute exact p-value with ties
```

But what if we look at the relationship for each region separately, using a different colour for each region?

```
plot(productivity,mammals,pch=16,col=as.numeric(region))
```



The pattern is obvious. In every single case, increasing productivity is associated with *reduced* mammal species richness within each region. The lesson is clear: you need to be extremely careful when looking at *correlations across different scales*. Things that are positively correlated over short time scales may turn out to be negatively correlated in the long term. Things that appear to be positively correlated at large spatial scales may turn out (as in this example) to be negatively correlated at small scales.

Reference

Snedecor, G.W. and Cochran, W.G. (1980) *Statistical Methods*, Iowa State University Press, Ames.

Further Reading

Dalgaard, P. (2002) *Introductory Statistics with R*, Springer-Verlag, New York.

7

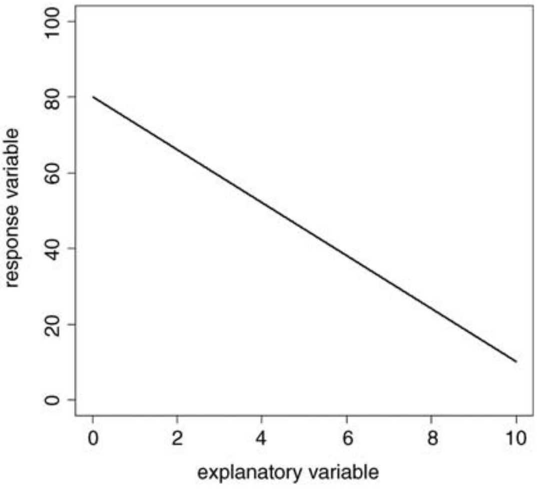
Regression

Regression analysis is the statistical method you use when both the response variable and the explanatory variable are continuous variables (i.e. real numbers with decimal places – things like heights, weights, volumes, or temperatures). Perhaps the easiest way of knowing when regression is the appropriate analysis is to see that a scatterplot is the appropriate graphic (in contrast to analysis of variance, say, when the appropriate plot would have been a box-and-whisker or a bar chart).

The essence of regression analysis is using sample data to estimate parameter values and their standard errors. First, however, we need to select a model which describes the relationship between the response variable and the explanatory variable(s). There are literally hundreds of models from which we might choose. Perhaps the most important thing to learn about regression is that *model choice is a really big deal*. The simplest model of all is the linear model:

$$y = a + bx$$

The response variable is y, and x is a continuous explanatory variable. There are two parameters, a and b: the intercept is a (the value of y when $x = 0$); and the slope is b (the slope, or gradient, is the change in y divided by the change in x which brought it about). The slope is so important that it is worth drawing a picture to make clear what is involved.



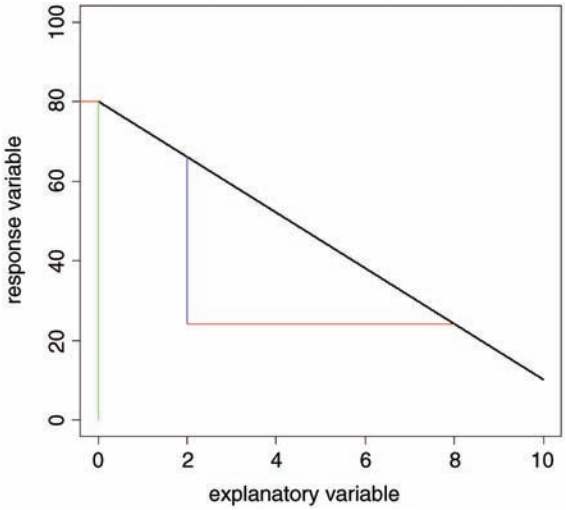
Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

The task is to work out the slope and intercept of this negative linear relationship between the response variable and the explanatory variable. It is easiest to start with the intercept in this case, because the value of $x = 0$ appears on the graph (it does not always). The intercept is simply the value of y when $x = 0$. We can get this by inspection. Draw a vertical line from $x = 0$ until it intercephs the black regression line (the green line) then a horizontal line (red) from the regression line until it cuts the y axis. Read off the value directly. It is 80 in this case.

Estimating the slope is slightly more involved because we need to calculate

$$\frac{\text{change in } y}{\text{change in } x \text{ that brought it about}}$$

In practice, it is a good idea for precision to select a large change in x. Let us take it from 2 to 8. Because the slope of the graph is negative, the value of y is lower when $x = 8$ than it is when $x = 2$. At $x = 2$, we draw a blue line vertically downwards from the regression line to the value of y when $x = 8$. The length of this blue line is the change in y (often denoted as 'delta y', or Δy in symbols). Now we draw a horizontal brown line showing the change in x from 2 to 8. The length of this brown line is Δx. When $x = 2$ we can read off the value of y (approximately) from the graph: it is roughly 66. Similarly, when $x = 8$ we can read off the value of y as 24.



So the change in x is +6 (from 2 up to 8) and the change in y is −42 (from 66 down to 24). Finally, we can calculate the slope of the straight line, b:

$$b = \frac{\text{change in } y}{\text{change in } x} = \frac{24 - 66}{8 - 2} = \frac{-42}{6} = -7.0$$

We now know both of the parameter values of the line: the intercept $a = 80$ and the slope $b = -7.0$. We can write the parameterised equation like this:

$$y = 80 - 7x$$

We can predict values of y at x values we have not measured (say, at $x = 10.5$):

$$y = 80 - 7.0 \times 10.5 = 6.5$$

We can also ask what x values are associated with particular values of y (say, $y = 40$). This is a bit more work, because we have to rearrange the equation

$$40 = 80 - 7.0x$$

First, subtract 80 from both sides

$$40 - 80 = -7.0x$$

then divide both sides by −7 to find the value of x:

$$x = \frac{40 - 80}{-7} = \frac{-40}{-7} = +5.714286$$

You can get a rough check on this value by inspection of the graph. You should work through this example repeatedly until you understand it completely.

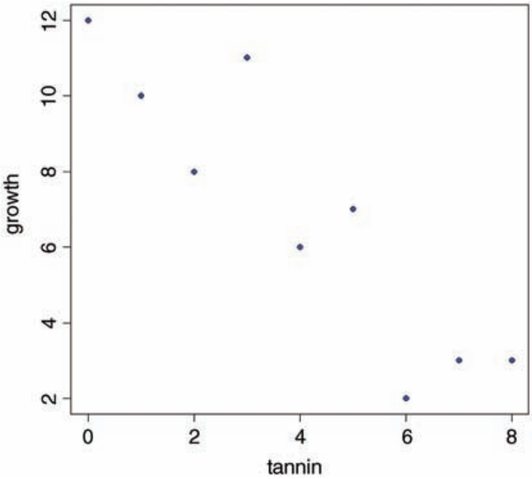
Linear Regression

Let us start with an example. The thing to understand is that there is nothing difficult, or mysterious about estimating the regression parameters. We can do a reasonable job, just by eye.

```
reg.data <- read.csv("c:\\temp\\tannin.csv")
attach(reg.data)
names(reg.data)
```

```
[1] "growth" "tannin"
```

```
plot(tannin,growth,pch=21,bg="blue")
```



This is how we do regression 'by eye'. We ask: what has happened to the y value? It decreased from about 12 to about 2, so the change in y is -10 (the minus sign is important). How did the x value change? It increased from 0 to 8, so the change in x is +8 (when working out regressions by eye, it is a good idea to take as big a range of x values as possible, so here we took the complete range of x). What is the value of y when $x = 0$? It is about 12, so the intercept is roughly $a \approx 12$. Finally, what is the value of b? It is the change in y (-10) divided by the change in x which brought it about (8), so $b \approx -10/8 = -1.25$. So our rough guess at the regression equation is

$$y = 12.0 - 1.25x$$

That's all there is to it. Obviously, we want to make the procedure more objective than this. And we also want to estimate the unreliability of the two estimated parameters (i.e. the standard errors of the slope and intercept). But the basics are just as straightforward as that.

Linear Regression in R

How close did we get to the maximum likelihood estimates of a and b with our guesstimates of 12 and -1.25? It is easy to find out using the R function `lm` which stands for 'linear model' (note that the first letter of the function name `lm` is a lower case L, not a number one). All we need do is tell R which of the variables is the response variable (growth in this case) and which is the explanatory variable (tannin concentration in the diet). The response variable goes on the left of the tilde ~ and the explanatory variable goes on the right, like this: `growth ~ tannin`. This is read 'growth is modelled as a function of tannin'. Now we write:

```
lm(growth~tannin)
```

```
Coefficients:
(Intercept)   tannin
     11.756   -1.217
```

The two parameters are called `Coefficients` in R: the intercept is 11.756 (compared with out guesstimate of 12), and the slope is -1.217 (compared with our guesstimate of -1.25). Not bad at all.

So where does R get its coefficients from? We need to do some calculations to find this out. If you are more mathematically inclined, you might like to work through Box 7.1, but this is not essential to understand what is going on. Remember that what we want are the maximum likelihood estimates of the parameters. That is to say that, given the data, and having selected a linear model, we want *to find the values of the slope and intercept that make the data most likely*. Keep rereading this sentence until you understand what it is saying.

The best way to see what is going on is to do it graphically. Let us cheat a bit by fitting the best-fit straight line through our scatterplot, using `abline` like this:

```
abline(lm(growth~tannin),col="green")
```

Box 7.1. The least-squares estimate of the regression slope, *b*

The *best-fit* slope is found by rotating the line until the *error sum of squares*, *SSE*, is minimized, so we want to find the minimum of $\sum (y - a - bx)^2$. We start by finding the derivative of *SSE* with respect to *b*:

$$\frac{dSSE}{db} = -2\sum x(y - a - bx)$$

Now, multiplying through the bracketed term by *x* gives:

$$\frac{dSSE}{db} = -2\sum xy - ax - bx^2$$

Apply summation to each term separately, set the derivative to zero, and divide both sides by −2 to remove the unnecessary constant:

$$\sum xy - \sum ax - \sum bx^2 = 0$$

We cannot solve the equation as it stands because there are 2 unknowns, *a* and *b*. However, we know the value of *a* is $\bar{y} - b\bar{x}$. Also, note that $\sum ax$ can be written as $a\sum x$, so, replacing *a* and taking both *a* and *b* outside their summations gives:

$$\sum xy - \left[\frac{\sum y}{n} - b\frac{\sum x}{n}\right]\sum x - b\sum x^2 = 0$$

Now multiply out the central bracketed term by $\sum x$ to get

$$\sum xy - \frac{\sum x \sum y}{n} + b\frac{\left(\sum x\right)^2}{n} - b\sum x^2 = 0$$

Finally, take the two terms containing *b* to the right-hand side, and note their change of sign:

$$\sum xy - \frac{\sum x \sum y}{n} = b\sum x^2 - b\frac{\left(\sum x\right)^2}{n}$$

Then divide both sides by $\sum x^2 - \left(\sum x\right)^2/n$ to obtain the required estimate *b*:

$$b = \frac{\sum xy - \frac{\sum x \sum y}{n}}{\sum x^2 - \frac{\left(\sum x\right)^2}{n}}$$

Thus, the value of *b* that minimizes the sum of squares of the departures is given simply by (see Box 7.3 for more details):

$$b = \frac{SSXY}{SSX}$$

This is the *maximum likelihood estimate of the slope* of the linear regression.

The fit is reasonably good, but it is not perfect. The data points do not lie on the fitted line. The difference between each data point and the value predicted by the model at the same value of x is called a *residual*. Some residuals are positive (above the line) and others are negative (below the line). Let us draw vertical lines to indicate the size of the residuals. The first x point is at tannin = 0. The y value measured at this point was growth = 12. But what is the growth predicted by the model at tannin = 0? There is a built-in function called predict to work this out:

```
fitted <- predict(lm(growth~tannin))
fitted
```

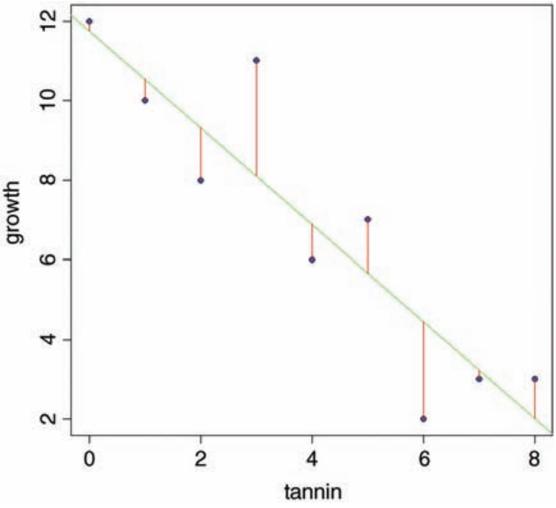
```
        1         2         3         4         5         6         7
11.755556 10.538889 9.322222 8.105556 6.888889 5.672222 4.455556
       8         9
3.238889 2.022222
```

So the first predicted value of growth is 11.755556 when tannin = 0. To draw the first residual, both x coordinates will be 0. The first y coordinate will be 12 (the observed value) and the second will be 11.755556 (the fitted (or predicted) value). We use lines, like this:

```
lines(c(0,0),c(12,11.755556))
```

We could go through, laboriously, and draw each residual like this. But it is much quicker to automate the procedure, using a loop to deal with each residual in turn:

```
for (i in 1:9)
     lines (c(tannin[i],tannin[i]),c(growth[i],fitted[i]),col="red")
```



These residuals describe the goodness of fit of the regression line. Our maximum likelihood model is defined as *the model that minimizes the sum of the squares of these residuals*. It is useful, therefore, to write down exactly what any one of the residuals, d, is: it is the measured value, y, minus the fitted value called $\hat{y}$ (y 'hat'):

$$d = y - \hat{y}$$

We can improve on this, because we know that $\hat{y}$ is on the straight line $a + bx$, so

$$d = y - (a + bx) = y - a - bx$$

The equation includes $a - bx$ because of the minus sign outside the bracket. Now our best-fit line, by definition, is given by the values of a and b that minimize the sums of the squares of the ds (see Box 7.1). Note, also, that just as $\sum(y - \bar{y}) = 0$ (Box 4.1), so the sum of the residuals $\sum d = \sum(y - a - bx) = 0$ (Box 7.2).

Box 7.2. The sum of the residuals in a linear regression is zero

Each point on the graph, y, is located at x. The model predicts a value $\hat{y} = a + bx$ where a is the intercept and b is the slope. Each residual is the distance between y and the fitted value $\hat{y}$ at location x and we are interested in the sum of the residuals:

$$\sum (y - a - bx)$$

which we shall prove is equal to zero. First, we note that the best-fit regression line passes through the point $(\bar{x}, \bar{y})$ so that we know that $\bar{y} = a + b\bar{x}$. We can rearrange this to find the value of the intercept:

$$a = \bar{y} - b\bar{x}$$

We can substitute this value in the equation for the sum of the residuals (above):

$$\sum (y - \bar{y} + b\bar{x} - bx)$$

Now we apply the summation, remembering that $\sum \bar{y} = n\bar{y}$ and $\sum \bar{x} = n\bar{x}$

$$\sum y - n\bar{y} + bn\bar{x} - b\sum x$$

Recall that $\bar{y} = \sum y/n$ and that $\bar{x} = \sum x/n$ so we can replace the means:

$$\sum y - n\frac{\sum y}{n} + bn\frac{\sum x}{n} - b\sum x$$

Cancel the ns to get

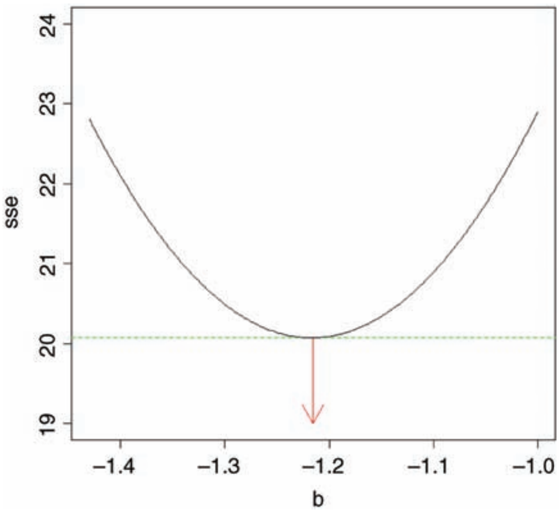
$$\sum y - \sum y + b\sum x - b\sum x = 0$$

which completes the proof.

To get an overview of what is involved, it is useful to plot the sum of the squares of the residuals against the value of the parameter we are trying to estimate. Let us take the slope as our example. We know one thing for certain about our straight-line model; it will pass through the point in the centre of the cloud of data whose coordinates are $(\bar{x}, \bar{y})$. The best-fit line will be pivoted about the mean values of x and y and our job is to find the best value for this slope – the one that minimizes the sum of squares of the red lines in the graph above. It should be reasonably clear that if our estimate of the slope is too steep, then the fit will be poor and the sum of squares will be large. Likewise, if our estimate of the slope is too shallow, then the fit will be poor and the sum of squares will be large. Somewhere between these two extremes, there will be a value for the slope that minimizes the sum of squares. This is the best-fit value that we want to discover. First we need to loop thorough an appropriate range of values which will include the best-fit value (say $-1.4 < b < -1.0$) and work out the sum of the squared residuals: let us call this quantity `sse` (you will see why later):

- change the value of the slope b
- work out the new intercept $a = \bar{y} - b\bar{x}$
- predict the fitted values of growth for each level of tannin $a + bx$
- work out the residuals $y - a - bx$
- square them and add them up, $\sum (y - a - bx)^2$
- associate this value of `sse[i]` with the current estimate of the slope `b[i]`

Once this process is complete, we can produce a U-shaped graph with the squared residuals on the y axis and the estimate of the slope on the x axis. Now we find the minimum value of `sse` (it turns out to be 20.072) and draw a horizontal dashed green line. At the point where this minimum touches the graph, we read down to the x axis to find the best value of the slope (the red arrow). This is the value ($b = -1.217$) that R provided for us earlier.



Here is the R code that produces the figure and extracts the best estimate of *b*:

```
b <- seq(-1.43,-1,0.002)
sse <- numeric(length(b))
for (i in 1:length(b)) {
     a <- mean(growth)-b[i]*mean(tannin)
     residual <- growth - a - b[i]*tannin
     sse[i] <- sum(residual^2)
}
```

```
plot(b,sse,type="l",ylim=c(19,24))
     arrows(-1.216,20.07225,-1.216,19,col="red")
     abline(h=20.07225,col="green",lty=2)
     lines(b,sse)
```

```
b[which(sse==min(sse))]
```

Calculations Involved in Linear Regression

We want to find the minimum of $\sum d^2 = \sum (y - a - bx)^2$. To work this out we need the 'famous five': these are $\sum y^2$ and $\sum y$, $\sum x^2$ and $\sum x$ and a new quantity, $\sum xy$, the sum of products. The sum of products is worked out pointwise, so for our data, it is:

```
tannin
```

```
[1]  0  1  2  3  4  5  6  7  8
```

```
growth
```

```
[1]  12  10  8  11  6  7  2  3  3
```

```
tannin*growth
```

```
[1]  0  10  16  33  24  35  12  21  24
```

We have $0 \times 12 = 0$, plus $1 \times 10 = 10$, plus $2 \times 8 = 16$, and so on:

```
sum(tannin*growth)
```

```
[1] 175
```

The next thing is to use the famous five to work out three essential 'corrected sums': the corrected sum of squares of *x*, the corrected sum of squares of *y* and the corrected sum of products, *xy*. The corrected sums of squares of *y* and *x* should be familiar to you:

$$SSY = \sum y^2 - \frac{\left(\sum y\right)^2}{n}$$

$$SSX = \sum x^2 - \frac{\left(\sum x\right)^2}{n}$$

because if we wanted the variance in *y*, we would just divide *SSY* by its degrees of freedom (and likewise for the variance in *x*; see p. 55). It is only the corrected sum of products that is novel, but its structure is directly analogous. Think about the formula for *SSY*, above. It is 'the sum of *y* times *y*' $\sum y^2$, 'minus the sum of *y* times the sum of *y*' $\left(\sum y\right)^2$ 'divided by the sample size', n. The formula for SSX is similar. It is 'the sum of x times x' $\sum x^2$, 'minus the sum of x times the sum of x' $\left(\sum x\right)^2$ 'divided by the sample size', n. Now the corrected sum of products is:

$$SSXY = \sum xy - \frac{\left(\sum x\right)\left(\sum y\right)}{n}$$

If you look carefully you will see that this has exactly the same kind of structure. It is 'the sum of x times y' $\sum xy$, 'minus the sum of x times the sum of y ' $\left(\sum x\right)\left(\sum y\right)$ 'divided by the sample size', n.

These three corrected sums of squares are absolutely central to everything that follows about regression and analysis of variance, so it is a good idea to reread this section as often as necessary, until you are confident that you understand what SSX, SSY and $SSXY$ represent (Box 7.3).

Box 7.3. Corrected sums of squares and products in regression

The total sum of squares is SSY, the sum of squares of x is SSX, and the corrected sum of products is $SSXY$:

$$SSY = \sum y^2 - \frac{\left(\sum y\right)^2}{n}$$

$$SSX = \sum x^2 - \frac{\left(\sum x\right)^2}{n}$$

$$SSXY = \sum xy - \frac{\sum x \sum y}{n}$$

The explained variation is the regression sum of squares, SSR:

$$SSR = \frac{SSXY^2}{SSX}$$

The unexplained variation is the error sum of squares, SSE, can be obtained by difference

$$SSE = SSY - SSR$$

but SSE is defined as the sum of the squares of the residuals, which is

$$SSE = \sum (y - a - bx)^2$$

The correlation coefficient, r, is given by

$$r = \frac{SSXY}{\sqrt{SSX \times SSY}}$$

The next question is how we use *SSX*, *SSY* and *SSXY* to find the maximum likelihood estimates of the parameters and their associated standard errors. It turns out that this step is much simpler than what has gone before. The maximum likelihood estimate of the slope, b, that we extracted from the graph (above) is just:

$$b = \frac{SSXY}{SSX}$$

(the detailed derivation of this is in Box 7.1).

Now that we know the value of the slope, we can use any point on the fitted straight line to work out the maximum likelihood estimate of the intercept, a. One part of the definition of the best-fit straight line is that it passes through the point $(\bar{x}, \bar{y})$ determined by the mean values of x and y. Since we know that $y = a + bx$, it must be the case that $\bar{y} = a + b\bar{x}$, and so

$$a = \bar{y} - b\bar{x} = \frac{\sum y}{n} - b\frac{\sum x}{n}$$

We can work out the parameter values for our example. To keep things as simple as possible, we can call the variables `SSX,` `SSY` and `SSXY` (note that R is 'case sensitive' so the variable `SSX` is different from `ssx`):

Box 7.4. The shortcut formula for the sum of products, *SSXY*

SSXY is based on the expectation of the product $(x - \bar{x})(y - \bar{y})$. Start by multiplying out the brackets:

$$(x - \bar{x})(y - \bar{y}) = xy - x\bar{y} - y\bar{x} + \bar{x}\bar{y}$$

Now apply the summation, remembering that $\sum x\bar{y} = \bar{y}\sum x$ and $\sum y\bar{x} = \bar{x}\sum y$:

$$\sum xy - \bar{y}\sum x - \bar{x}\sum y + n\bar{x}\bar{y}$$

We know that $\sum x = n\bar{x}$ and that $\sum y = n\bar{y}$, so

$$\sum xy - n\bar{y}\bar{x} - n\bar{x}\bar{y} + n\bar{x}\bar{y} = \sum xy - n\bar{x}\bar{y}$$

Now replace the product of the two means by $\sum x/n \times \sum y/n$

$$\sum xy - n\frac{\sum x}{n}\frac{\sum y}{n}$$

which, on cancelling the ns, gives the corrected sum of products as

$$SSXY = \sum xy - \frac{\sum x \sum y}{n}$$

```
SSX <- sum(tannin^2)-sum(tannin)^2/length(tannin)
SSX
```

```
[1] 60
```

```
SSY <- sum(growth^2)-sum(growth)^2/length(growth)
SSY
```

```
[1] 108.8889
```

```
SSXY <- sum(tannin*growth)-sum(tannin)*sum(growth)/length(tannin)
SSXY
```

```
[1] -73
```

That is all we need. So the slope is

$$b = \frac{SSXY}{SSX} = \frac{-73}{60} = -1.2166667$$

and the intercept is given by

$$a = \frac{\sum y}{n} - b.\frac{\sum x}{n} = \frac{62}{9} + 1.2166667\frac{36}{9} = 6.8889 + 4.86667 = 11.755556$$

Now we can write the maximum likelihood regression equation in full:

$$y = 11.75556 - 1.216667x$$

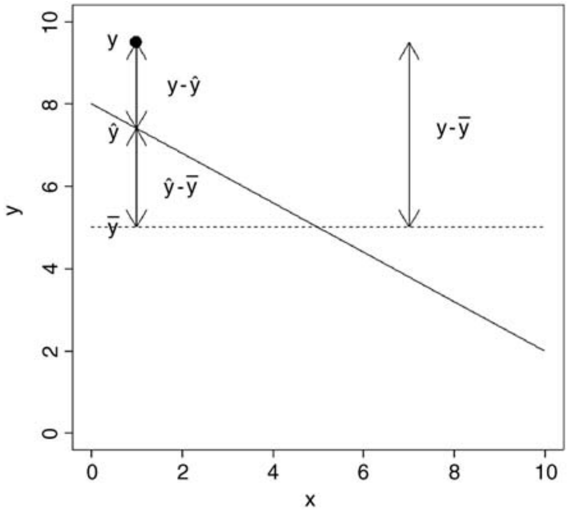
This, however, is only half of the story. In addition to the parameter estimates, $a = 11.756$ and $b = -1.2167$, we need to measure the unreliability associated with each of the estimated parameters. In other words, we need to calculate the standard error of the intercept and the standard error of the slope. We have already met the standard error of a mean, and we used it in calculating confidence intervals (p. 60) and in doing Student's t test (p. 91). Standard errors of regression parameters are similar in so far as they are enclosed inside a big square-root term (so that the units of the standard error are the same as the units of the parameter), and they have the error variance, s^2, in the numerator. There are extra components, however, which are specific to the unreliability of a slope or an intercept (see Boxes 7.6 and 7.7 for details). Before we can work out the standard errors, however, we need to know the value of the error variance s^2, and for this we need to carry out an analysis of variance.

Partitioning Sums of Squares in Regression: *SSY* = *SSR* + *SSE*

The idea is simple: we take the total variation in y, SSY, and partition it into components that tell us about the explanatory power of our model. The variation that is explained by the model is called the regression sum of squares (denoted by SSR), and the unexplained variation is called the error sum of squares (denoted by SSE; this is the sum of the squares of the lengths of the red lines that we drew on the scatterplot on p. 119). Then $SSY = SSR + SSE$ (the proof is given in Box 7.5).

Box 7.5. Proof that $SSY = SSR + SSE$

Let us start with what we know. The difference between y and $\bar{y}$ is the sum of the differences $y - \hat{y}$ and $\hat{y} - \bar{y}$, as you can see from the figure:



Inspection of the figure shows clearly that $(y - \bar{y}) = (y - \hat{y}) + (\hat{y} - \bar{y})$. It is not obvious, however, that the sums of the squares of these three quantities should be equal. We need to prove that.

First, square the terms on both sides of the equation

$$(y - \bar{y})^2 = (y - \hat{y})^2 + (\hat{y} - \bar{y})^2$$

to get

$$y^2 - 2y\bar{y} + \bar{y}^2 = y^2 - 2y\hat{y} + \hat{y}^2 + \hat{y}^2 - 2\hat{y}\bar{y} + \bar{y}^2$$

From both sides, subtract $y^2 + \bar{y}^2$ to leave

$$-2y\bar{y} = -2y\hat{y} + 2\hat{y}^2 - 2\hat{y}\bar{y}$$

Apply summation then set to zero:

$$0 = \sum 2y\bar{y} - \sum 2y\hat{y} + \sum 2\hat{y}^2 - \sum 2\hat{y}\bar{y}$$

Now group the terms with and without $\bar{y}$:

$$0 = \sum 2y\bar{y} - \sum 2\hat{y}\bar{y} + \sum 2\hat{y}^2 - \sum 2y\hat{y}$$

Because $\bar{y}$ is a constant, we can take it outside the summation along with the 2:

$$0 = 2\bar{y}\sum(y-\hat{y}) + 2\sum\hat{y}(\hat{y}-y)$$

We already know that $\sum(y-\hat{y}) = 0$ from Box 7.2, so all that remains is to prove is that $2\sum\hat{y}(\hat{y}-y) = 0$. This requires more algebra, because we have to replace $\hat{y}$ with $a + bx = \bar{y} + b(x-\bar{x})$. First multiply through the bracket to get $\hat{y}^2 - y\hat{y}$, then replace the $\hat{y}$:

$$(\bar{y} + b(x-\bar{x}))(\bar{y} + b(x-\bar{x})) - y(\bar{y} + b(x-\bar{x}))$$

This is the gory bit. Multiply out the brackets:

$$\bar{y}^2 + 2b\bar{y}(x-\bar{x}) + b^2(x-\bar{x})^2 - y\bar{y} - by(x-\bar{x})$$

Next, apply the summation, recalling that $\sum\bar{y}^2 = n\bar{y}^2$:

$$n\bar{y}^2 + 2b\bar{y}\sum(x-\bar{x}) + b^2\sum(x-\bar{x})^2 - \bar{y}\sum y - b\sum y(x-\bar{x})$$

Now replace $\bar{y}$ with $\frac{\sum y}{n}$, cancel the n and note that $n\bar{y}^2 = \bar{y}\sum y$ to obtain

$$2b\frac{\sum y}{n}\sum(x-\bar{x}) + b^2\sum(x-\bar{x})^2 - b\sum y(x-\bar{x})$$

Notice that the first of the three terms involves $\sum(x-\bar{x})$ so this must be zero, leaving

$$b^2\sum(x-\bar{x})^2 - b\sum y(x-\bar{x})$$

Take $-b$ outside the bracket and note that $\sum(x-\bar{x})^2 = SSX$ and $\sum y(x-\bar{x}) = SSXY$, so

$$-b(SSXY \;-\; b\,SSX)$$

Finally, remember that the definition of the slope, $b = SSXY/SSX$ so $b.SSX = SSXY$, giving

$$-b(SSXY \;- SSXY) = 0$$

to complete the proof.

Now, in principle, we could compute *SSE* because we know that it is the sum of the squares of the deviations of the data points from the fitted model, $\sum d^2 = \sum(y-a-bx)^2$. Since we know the values of a and b, we are in a position to work this out. The formula is fiddly, however, because of all those subtractions, squarings and addings up. Fortunately, there is a very simple shortcut that involves computing *SSR*, the explained variation, rather than *SSE*. This is because

$$SSR = b.SSXY$$

so we can immediately work out $SSR = -1.21667 \times -73 = 88.81667$. And since $SSY = SSR + SSE$ we can get SSE by subtraction:

$$SSE = SSY - SSR = 108.8889 - 88.81667 = 20.07222$$

These components are now drawn together in what is known as the 'ANOVA table'. Strictly, we have analysed sums of squares rather than variances up to this point, but you will see why it is called analysis of variance shortly. The leftmost column of the ANOVA table lists the sources of variation: regression, error and total in our example. The next column contains the sums of squares, *SSR, SSE* and *SSY*. The third column is in many ways the most important to understand; it contains the degrees of freedom. There are n points on the graph ($n = 9$ in this example). So far, our table looks like this:

Source	Sum of squares	Degrees of freedom	Mean squares	*F* ratio
Regression	88.817			
Error	20.072			
Total	108.889			

We shall work out the degrees of freedom associated with each of the sums of squares in turn. The easiest to deal with is the total sum of squares, because it always has the same formula for its degrees of freedom. The definition is $SSY = \sum (y - \bar{y})^2$, and you can see that there is just one parameter estimated from the data: the mean value, $\bar{y}$. Because we have estimated one parameter from the data, we have $n - 1$ degrees of freedom (where n is the total number of points on the graph; 9 in our example). The next easiest to work out is the error sum of squares. Let us look at its formula to see how many parameters need to be estimated from the data before we can work out $SSE = \sum (y - a - bx)^2$. We need to know the values of both a and b before we can calculate *SSE*. These are estimated from the data, so the degrees of freedom for error are $n - 2$. This is important, so reread the last sentence if you do not see it yet. The most difficult of the three is the regression degrees of freedom, because you need to think about this one in a different way. The question is this: how many extra parameters, over and above the mean value of y, did you estimate when fitting the regression model to the data? The answer is 1. The extra parameter you estimated was the slope, b. So the regression degrees of freedom in this simple model, with just one explanatory variable, is 1. This will only become clear with practice.

To complete the ANOVA table, we need to understand the fourth column, headed 'Mean squares'. This column contains the variances, on which analysis of variance is based. The key point to recall is that

$$\text{variance} = \frac{\text{sum of squares}}{\text{degrees of freedom}}$$

This is very easy to calculate in the context of the ANOVA table, because the relevant sums of squares and degrees of freedom are in adjacent columns. Thus the regression variance is just $SSR/1 = SSR$, and the error variance is $s^2 = SSE/(n - 2)$. Traditionally, one does not fill in the bottom box (it would be the overall variance in y, $SSY/(n - 1)$). Finally, the ANOVA table is completed by working out the F ratio, which is a ratio between two variances. In most simple ANOVA tables, you divide the treatment variance in the numerator (the regression variance in this case) by the error variance s^2 in the denominator. The null hypothesis under test in a linear regression is that the slope of the regression line is zero (i.e. no dependence of y on x). The two-tailed alternative hypothesis is that the slope is significantly different from zero (either positive or negative). In many applications it is not particularly interesting to reject the null hypothesis, because we are interested in the effect sizes (estimates of the slope and intercept) and their standard errors. We often know from the outset that the null hypothesis is false. Nevertheless, to test whether the F ratio is sufficiently large to reject the null hypothesis, we compare our *test statistic* (the calculated value of F in the final column of the ANOVA table) with the *critical value* of F. Recall that the test statistic is the value of F that is expected by chance alone when the null hypothesis is true. We find the critical value of F from quantiles of the F distribution `qf`, with 1 d.f. in the numerator and $n-2$ d.f. in the denominator (as described below). Here is the completed ANOVA table:

Source	Sum of squares	Degrees of freedom	Mean squares	F ratio
Regression	88.817	1	88.817	30.974
Error	20.072	7	$s^2 = 2.86746$	
Total	108.889	8		

Notice that the component degrees of freedom add up to the total degrees of freedom (this is always true, in any ANOVA table, and is a good check on your understanding of the design of the experiment). The last question concerns the magnitude of the F ratio $= 30.974$: is it big enough to justify rejection of the null hypothesis? The critical value of the F ratio is the value of F that would arise due to chance alone when the null hypothesis was true, given that we have 1 d.f. in the numerator and 7 d.f. in the denominator. We have to decide on the level of uncertainty that we are willing to put up with; the traditional value for work like this is 5%, so our certainty is 0.95. Now we can use quantiles of the F distribution, `qf`, to find the critical value:

```
qf(0.95,1,7)
[1] 5.591448
```

Because our calculated value of F (the test statistic $= 30.974$) is much larger than the critical value (5.591), we can be confident in rejecting the null hypothesis. Perhaps a better thing to do, rather than working rigidly at the 5% uncertainty level, is to ask what is the probability of getting a value for F as big as 30.974 or larger if the null hypothesis is true. For this we use `1-pf` rather than `qf`:

```
1-pf(30.974,1,7)
[1] 0.0008460725
```

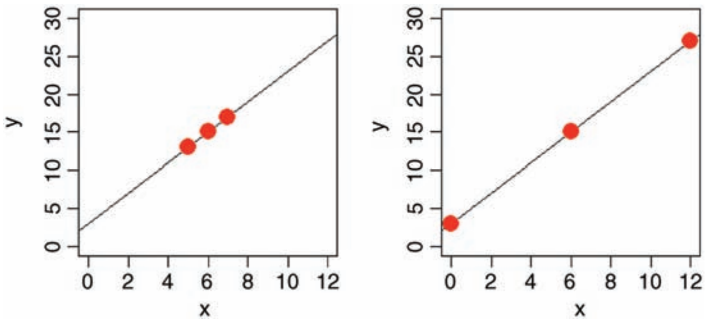
It is very unlikely indeed ($p < 0.001$).

Box 7.6. The standard error of the regression slope

This quantity is given by

$$SE_b = \sqrt{\frac{s^2}{SSX}}$$

The error variance s^2 comes from the ANOVA table and is the quantity used in calculating standard errors and confidence intervals for the parameters, and in carrying out hypothesis testing. *SSX* measures the spread of the x values along the x axis. Recall that standard errors are *unreliability estimates*. Unreliability increases with the error variance, so it makes sense to have s^2 in the numerator (on top of the division). It is less obvious why unreliability should depend on the range of x values. Look at these two graphs that have exactly the same slopes ($b = 2$) and intercepts ($a = 3$).



The difference is that the left-hand graph has all of its x values close to the mean value of x, while the graph on the right has a broad span of x values. Which of these do you think would give the most reliable estimate of the slope? It is pretty clear that it is the graph on the right, with the wider range of x values. Increasing the spread of the x values reduces unreliability of the estimated slope and hence appears in the denominator (on the bottom of the equation). What is the purpose of the big square root term? This is there to make sure that the units of the unreliability estimate are the same as the units of the parameter whose unreliability is being assessed. The error variance is in units of *y squared*, but the slope is in units of y per unit change in x.

Next, we can use the calculated error variance $s^2 = 2.867$ to work out the standard errors of the slope (Box 7.6) and the intercept (Box 7.7). First the standard error of the slope:

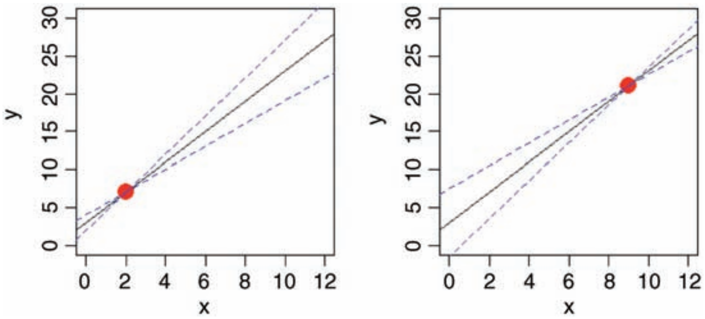
$$SE_b = \sqrt{\frac{s^2}{SSX}} = \sqrt{\frac{2.867}{60}} = 0.2186$$

Box 7.7. The standard error of the intercept

This quantity is given by

$$SE_a = \sqrt{\frac{s^2 \sum x^2}{n \times SSX}}$$

which is like the formula for the standard error of the slope, but with two additional terms. Uncertainty declines with increasing sample size n. It is less clear why uncertainty should increase with $\sum x^2$. The reason for this is that uncertainty in the estimate of the intercept increases, the further away from the intercept that the mean value of x lies. You can see this from the following graphs of $y = 3 + 2x$ (solid black lines). On the left is a graph with a low value of $\bar{x} = 2$ and on the right an identical graph (same slope and intercept) but estimated from a data set with a higher value of $\bar{x} = 9$. In both cases there is a variation in the slope from $b = 1.5$ to $b = 2.5$ (dotted blue lines). Compare the difference in the prediction of the intercept in the two cases.



Confidence in predictions made with linear regression declines with the square of the distance between the mean value of x and the value of x at which the prediction is to be made (i.e. with $(x - \bar{x})^2$). Thus, when the origin of the graph is a long way from the mean value of x, the standard error of the intercept will be large, and vice versa.

In general, the *standard error for a predicted value* $\hat{y}$ is given by:

$$SE_{\hat{y}} = \sqrt{s^2 \left[\frac{1}{n} + \frac{(x - \bar{x})^2}{SSX} \right]}$$

Note that the formula for the standard error of the intercept is just the special case of this for $x = 0$.

The formula for the standard error of the intercept is a little more involved (Box 7.7):

$$SE_a = \sqrt{\frac{s^2 \sum x^2}{n \times SSX}} = \sqrt{\frac{2.867 \times 204}{9 \times 60}} = 1.0408$$

Now that we know where all the numbers come from, we can repeat the analysis in R and see just how straightforward it is. It is good practice to give the statistical model a name: `model` is as good as any.

```
model <- lm(growth~tannin)
```

Then, you can do a variety of things with the model. The most important, perhaps, is to see the details of the estimated effects, which you get from the `summary` function:

```
summary(model)

Coefficients:
             Estimate  Std. Error  t value   Pr(>|t|)
(Intercept)   11.7556      1.0408   11.295  9.54e-06 ***
tannin        -1.2167      0.2186   -5.565  0.000846 ***

Residual standard error: 1.693 on 7 degrees of freedom
Multiple R-squared: 0.8157,   Adjusted R-squared: 0.7893
F-statistic: 30.97 on 1 and 7 DF, p-value: 0.0008461
```

This shows everything you need to know about the parameters and their standard errors (compare the values for SE_a and SE_b with those you calculated long-hand, above). We shall meet the other terms shortly (residual standard error, multiple R-squared and adjusted R-squared). The p value and the F statistic are familiar from the ANOVA table.

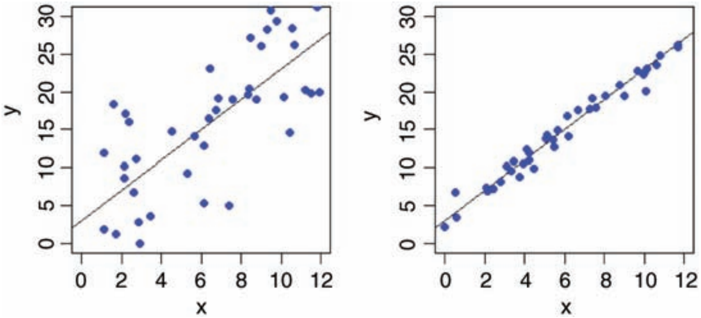
If you want to see the ANOVA table rather than the parameter estimates, then the appropriate function is `summary.aov`:

```
summary.aov(model)

           Df   Sum Sq   Mean Sq   F value     Pr(>F)
tannin      1    88.82     88.82     30.97  0.000846 ***
Residuals   7    20.07      2.87
```

This shows the error variance ($s^2 = 2.87$) along with SSR (88.82) and SSE (20.07), and the p value we just computed using `1-pf`. Of the two sorts of summary table, `summary.lm` is vastly the more informative, because it shows the effect sizes (in this case the slope of the graph and the intercept) and their unreliability estimates (the standard errors of the slope and intercept). Generally, you should resist the temptation to put ANOVA tables in your written work. The important information like the p value and the error variance can be put in the text, or in figure legends, much more efficiently. ANOVA tables put far too much emphasis on hypothesis testing, and show nothing directly about effect sizes or their unreliabilities.

Measuring the Degree of Fit, r^2

There is a very important issue that remains to be considered. Two regression lines can have exactly the same slopes and intercepts, and yet be derived from completely different relationships:

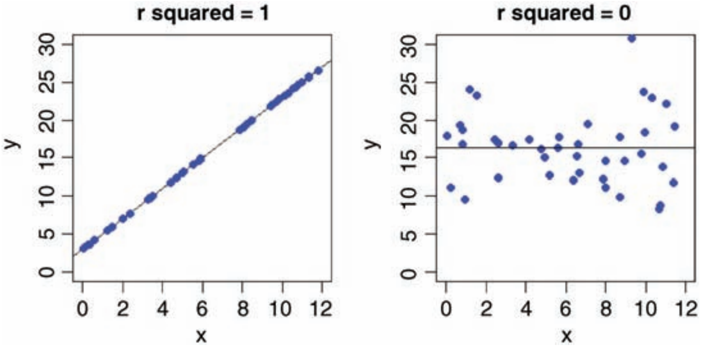


We need to be able to quantify the degree of fit, which is low in the graph on the left and high in the one on the right. In the limit, all the data points might fall exactly on the line. The degree of scatter in that case would be zero and the fit would be perfect (we might define a perfect fit as 1). At the other extreme, x might explain none of the variation in y at all; in this case, fit would be 0 and the degree of scatter would be 100%.

Can we combine what we have learned about *SSY, SSR* and *SSE* into a measure of fit that has these properties? Our proposed metric is *the fraction of the total variation in y that is explained by the regression*. The total variation is *SSY* and the explained variation is *SSR*, so our measure – let us call it r^2 – is given by

$$r^2 = \frac{SSR}{SSY}$$

This varies from 1, when the regression explains all of the variation in y ($SSR = SSY$), to 0 when the regression explains none of the variation in y ($SSE = SSY$).

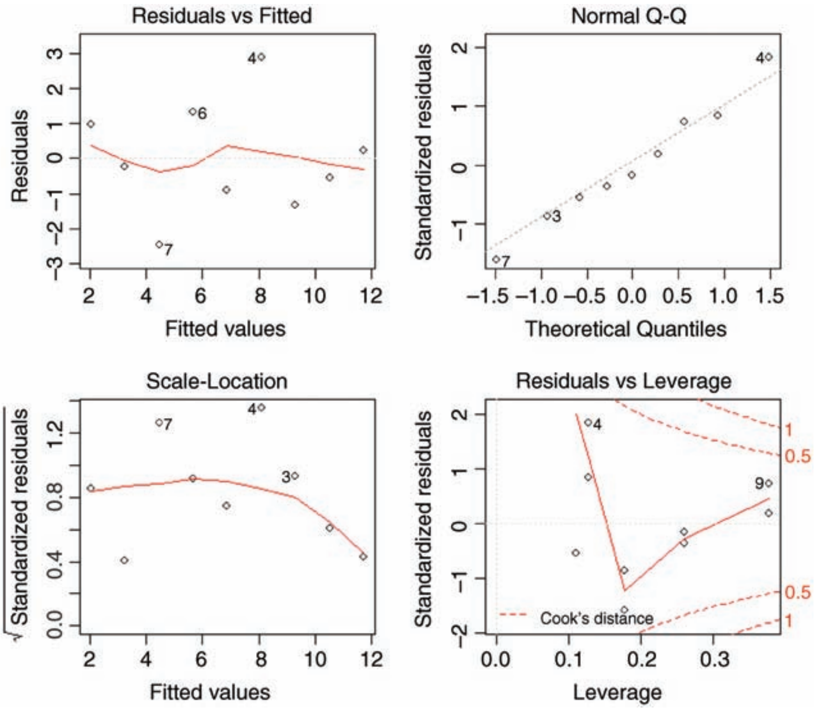


The formal name of this quantity is the coefficient of determination, but these days most people just refer to it as ‘*r-squared*’. We have already met the square root of this quantity (r or ρ), as the correlation coefficient (p. 108).

Model Checking

The final thing you will want to do is to expose the model to critical appraisal. The assumptions we really want to be sure about are *constancy of variance* and *normality of errors*. The simplest way to do this is with four built-in model-checking plots:

```
par(mfrow=c(2,2))
plot(model)
```



The first graph (top left) shows residuals on the y axis against fitted values on the x axis. It takes experience to interpret these plots, but what you *do not* want to see is lots of structure or pattern in the plot. Ideally, as here, the points should look like the sky at night. It is a major problem if the scatter increases as the fitted values get bigger;

this would show up like a wedge of cheese on its side (like this ◀ or less commonly like this ▶; see p. 65). But in our present case, everything is OK on the constancy of variance front.

The next plot (top right) is the normal quantile–quantile plot (`qqnorm`, p. 79) which should be a straight line if the errors are normally distributed. Again, the present example looks fine. If the pattern were S-shaped or banana-shaped, we would need to fit a different model to the data.

The third plot (bottom left) is like the first, but on a different scale; it shows the square root of the standardized residuals (where all the values are positive) against the fitted values; if there was a problem, the points would be distributed inside a triangular shape, with the scatter of the residuals increasing as the fitted values increase. But there is no such pattern here, which is good.

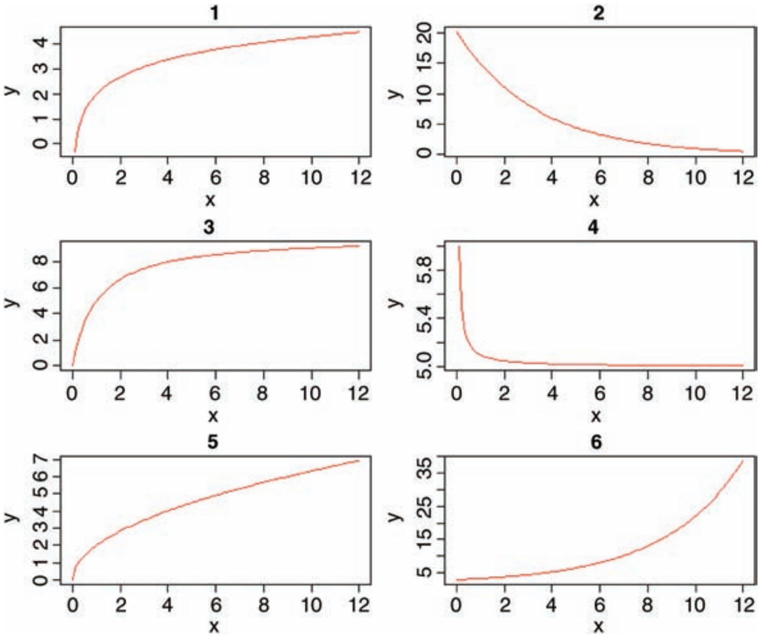
The fourth and final plot (lower right) is all about highlighting influential points (p. 148); these are the points on the graph that have the biggest effects on the parameter estimates. It shows Cook's distance as red contours on a plane defined by leverage and standardized residuals, with each point on the graph represented by an open circle, and selected points numbered (based on the order in which they appear in the dataframe). You can see that point number 9 has the highest leverage, point number 7 (all but hidden behind the label) has the highest Cook's distance, and point number 4 has the largest residual. In my opinion, this information is more clearly displayed in tabular form; try `influence.measures(model)`.

It is the top two graphs of `plot(model)` that are most important, and you should concentrate on these. The important point is that we *always* do model-checking; the `summary(model)` table is not the end of the process of regression analysis.

Transformation

You must not fall into the trap of thinking that $y = a + bx$ is the only two-parameter model for describing the relationship between the response variable and a single continuous explanatory variable. Model choice is a vitally important part of statistical analysis. Here are some other useful two-parameter models:

log X	$y\ = a + b\ln(x)$	1.
log Y	$y = \exp(a + bx)$	2.
asymptotic	$y = \dfrac{ax}{1 + bx}$	3.
reciprocal	$y = a + \dfrac{b}{x}$	4.
power law	$y = ax^b$	5.
exponential	$y = ae^{bx}$	6.



It is straightforward to estimate the parameters of such models if the equations can be transformed so that they become linear in their parameters. An example should make this clear. The following data show the relationship between radioactive emissions and time:

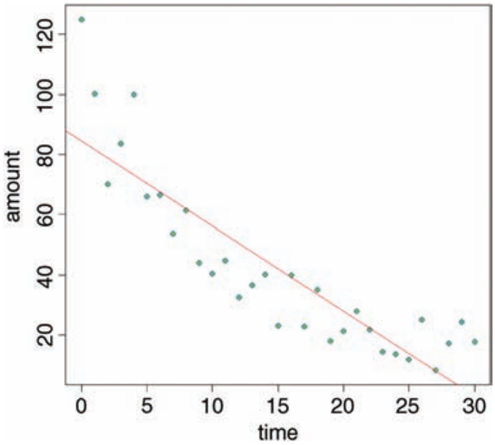
```
par(mfrow=c(1,1))
data <- read.csv("c:\\temp\\decay.csv")
attach(data)
names(data)
```

```
[1] "time"  "amount"
```

```
plot(time,amount,pch=21,col="blue",bg="green")
```

We start by fitting a straight line through the scatterplot, using `abline` with a linear model:

```
abline(lm(amount~time),col="red")
```



This draws attention to the pronounced curvature in the data. Most of the residuals at low values of `time` are positive, most of the residuals for intermediate values of `time` are negative, and most of the residuals at high values of `time` are positive. This is clearly *not* a good model for these data.

There is a very important point here. If, instead of looking at the fit of the model to the data using `plot`, we had simply done the statistics, then we might easily have come to the opposite conclusion. Here is a summary of the linear model applied to these data:

```
summary(lm(amount~time))
```

```
Coefficients:
               Estimate  Std. Error  t value    Pr(>|t|)
(Intercept)     84.5534     5.0277    16.82    < 2e-16 ***
time            -2.8272     0.2879    -9.82   9.94e-11 ***

Residual standard error: 14.34 on 29 degrees of freedom
Multiple R-squared: 0.7688,   Adjusted R-squared: 0.7608
F-statistic: 96.44 on 1 and 29 DF, p-value: 9.939e-11
```

The model explains more than 76% of the variation in the response (a very high value of *r*-squared) and the *p* value is vanishingly small. The moral is that *p* values and *r*-squared are *not* good measures of model adequacy.

Because the data relate to a decay process, it might be that an exponential function $y = ae^{-bx}$ describes the data better. If we can linearize this equation, then we can estimate the parameter values using a linear model. Let us try taking logs of both sides

$$y = ae^{-bx}$$

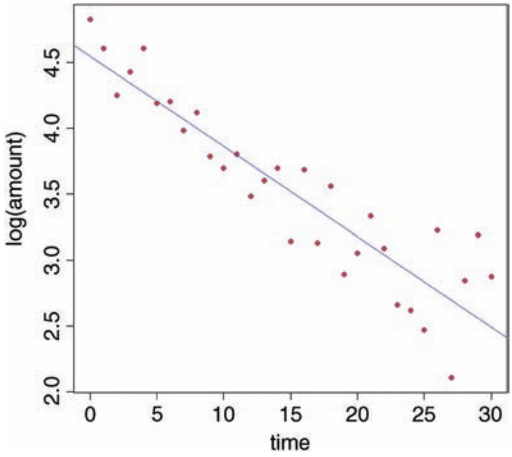
$$\log(y) = \log(a) - bx$$

If we replace log(y) by Y and log(a) by A, you can see that we have a linear model:

$$Y = A - bx$$

The intercept of this linear model is A and the slope is $-b$. To fit the model we have the *untransformed values* of time on the x axis and the *log* of amount on the y axis:

```
plot(time,log(amount),pch=21,col="blue",bg="red")
abline(lm(log(amount)~time),col="blue")
```



The fit to the model is greatly improved. There is a new issue, however, in that the variance appears to increase with time and, as you will recall, non-constant variance is a potentially serious problem. Let us estimate the parameter values of this exponential model and then check its assumptions using `plot(model)`.

```
model <- lm(log(amount)~time)
summary(model)
```

```
Coefficients:
               Estimate  Std. Error  t value    Pr(>|t|)
(Intercept)    4.547386    0.100295    45.34    < 2e-16 ***
time          -0.068528    0.005743   -11.93   1.04e-12 ***

Residual standard error: 0.286 on 29 degrees of freedom
Multiple R-squared: 0.8308,  Adjusted R-squared: 0.825
F-statistic: 142.4 on 1 and 29 DF, p-value: 1.038e-12
```

The slope of the straight line is −0.068528 and its standard error is 0.005743. The value of r^2 is even higher following transformation (83%) and the p value is even lower. The intercept of 4.547386 with its standard error of 0.100295 is for A, not the value we need for our exponential equation, but a is the antilog of A. When we back-transform, the standard errors become asymmetric up and down. It may take a moment for you to see why this is the case. Let us add one standard error to the intercept and subtract one standard error from it to get upper and lower intervals.

```
upper <- 4.547386 + 0.100295
lower <- 4.547386 - 0.100295
```

Now we return to the original scale of measurement by taking antilogs using `exp`:

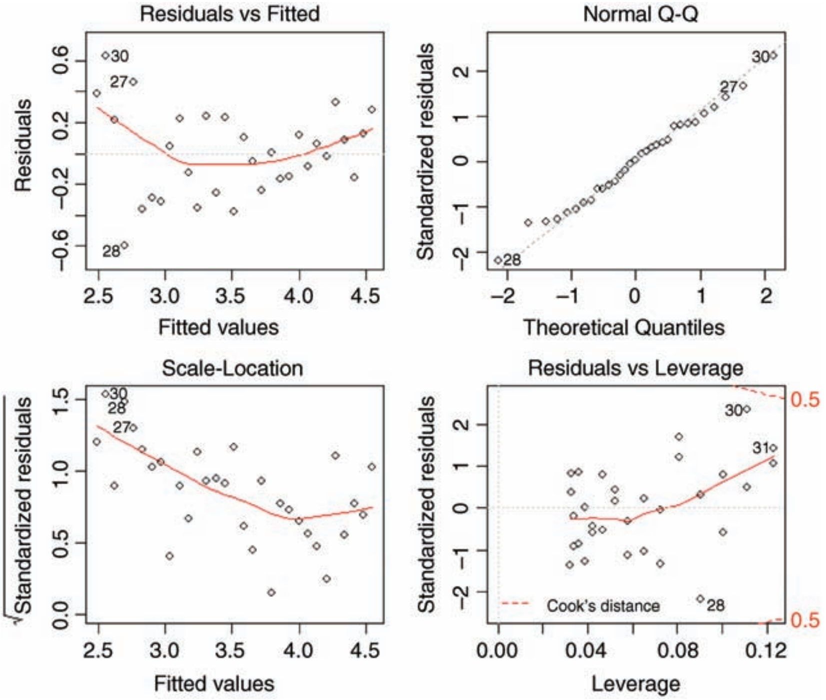
```
exp(upper)

[1] 104.3427

exp(lower)

[1] 85.37822
```

so the intercept on the original axis is between 85.38 and 104.34, but the best estimate for the intercept is

```
exp(4.547386)

[1] 94.38536
```

which means that the interval above the intercept is 9.957 but the interval below it is 9.007. Beginners often find it disconcerting that the two unreliability measures are different sizes.

Now we check the assumptions of the model using `plot(model)`:



The good news is that the normality of errors assumption looks good (the top right plot is reasonably straight). As we guessed by looking at the transformed data, however, the variance does show strong signs of non-constancy (the top left and bottom left plots). The bottom right plot shows that data points 30 and 31 have high leverage and point number 28 has a large residual. We shall see how deal with these issues later, but for the moment, we want to plot the curved line through the scatterplot on the original scale of measurement.

```
par(mfrow=c(1,1))
plot(time,amount,pch=21,col="blue",bg="green")
```

The key thing to understand about drawing curved lines in R is that curves are made up of lots of small straight-line sections. Once you get more than about 100 sections in the width of a graph, the curve typically looks quite smooth. Looking at the scatterplot, you can see that we want the values of time to go from 0 up to 30. To get more than 100 segments to the curve we therefore want more than three steps per unit time; let us take four steps, which makes the sequence interval 0.25. We call the variable `xv`, to stand for 'x values':

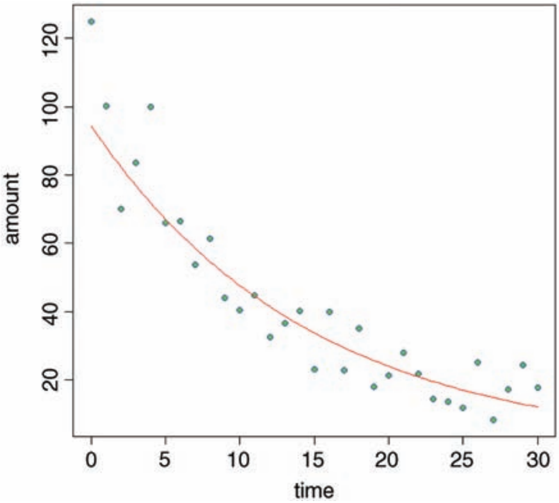
```
xv <- seq(0,30,0.25)
```

This gives us 121 values (`length(xv)`). We know the equation for the exponential curve is 94.38536 exp(−0.068528 x), so now we can calculate the y values (amounts) associated with each x value:

```
yv <- 94.38536 * exp(-0.068528 * xv)
```

Now we use the lines function to add the curve to the scatterplot:

```
lines(xv,yv,col="red")
```



As you can see, our model is a good description of the data for intermediate values of time, but the model is poor at predicting amount for time = 0 and for time > 28. Clearly, more work is required to understand what is going on at the extremes, but exponential decay describes the central part of the data reasonably well.

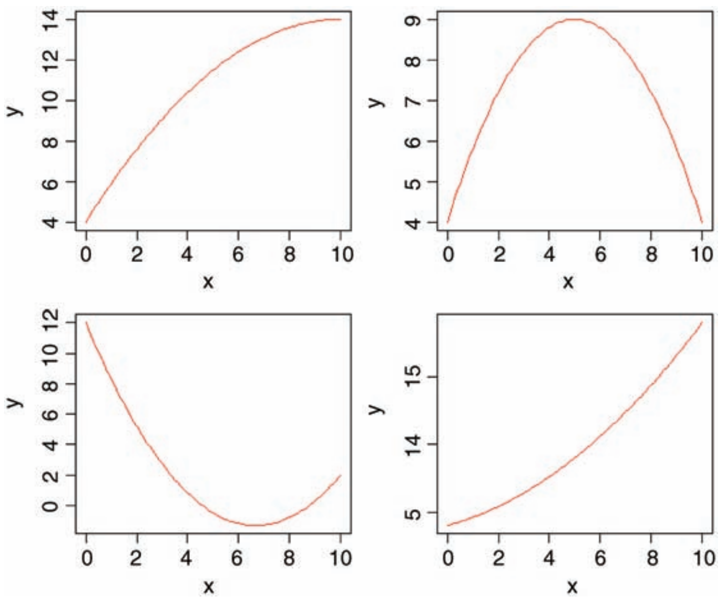
Polynomial Regression

The relationship between y and x often turns out not to be a straight line. But Occam's razor requires that we fit a linear model unless a non-linear relationship is significantly better at describing the data. So this begs the question: how do we assess the significance of departures from linearity? One of the simplest ways is to use polynomial regression

$$y = a + bx + cx^2 + dx^3 + \dots$$

The idea of polynomial regression is straightforward. As before, we have just one continuous explanatory variable, x, but we can fit higher powers of x, such as x^2 and x^3, to the model in addition to x to describe curvature in the relationship between y and x. It is useful to experiment with the kinds of graphs that can be generated with very simple models. Even if we restrict ourselves to the inclusion of a quadratic term, x^2, there are many curves we can describe, depending upon the signs of the linear and quadratic terms:

```
par(mfrow=c(2,2))
curve(4+2*x-0.1*x^2,0,10,col="red",ylab="y")
curve(4+2*x-0.2*x^2,0,10,col="red",ylab="y")
curve(12-4*x+0.3*x^2,0,10,col="red",ylab="y")
curve(4+0.5*x+0.1*x^2,0,10,col="red",ylab="y")
```



In the top left panel, there is a curve with positive but declining slope, with no hint of a hump ($y = 4 + 2x - 0.1x^2$). At top right we have a curve with a clear maximum ($y = 4 + 2x - 0.2x^2$), and at bottom left a curve with a clear minimum ($y = 12 - 4x + 0.35x^2$). The bottom right curve shows a positive association between y and x with the slope increasing as x increases ($y = 4 + 0.5x + 0.1x^2$). So you can see that a simple quadratic model with just three parameters (an intercept, a slope for x, and a slope for x^2) is capable of describing a wide range of functional relationships between y and x. It is very important to understand that the quadratic model *describes* the relationship between y and x; it does not pretend to *explain* the mechanistic (or causal) relationship between y and x.

We can use the decay data as an example of model comparison. How much better than a linear model with two parameters (call it `model2`) is a quadratic with three parameters (`model3`)? The function `I` stands for 'as is' and allows you to use arithmetic operators like caret (`^` for calculating powers) in a model formula where the same symbol would otherwise mean something different (in a model formula, caret means the order of interaction terms to be fitted).

```
model2 <- lm(amount~time)
model3 <- lm(amount~time+I(time^2))
summary(model3)
```

```
Coefficients:
                Estimate  Std. Error  t value    Pr(>|t|)
(Intercept)    106.38880     4.65627   22.849    < 2e-16 ***
time            -7.34485     0.71844  -10.223   5.90e-11 ***
I(time^2)        0.15059     0.02314    6.507   4.73e-07 ***

Residual standard error: 9.205 on 28 degrees of freedom
Multiple R-squared: 0.908,   Adjusted R-squared: 0.9014
F-statistic: 138.1 on 2 and 28 DF, p-value: 3.122e-15
```

You can see that the slope for the quadratic term (0.15059) is highly significant, which indicates important curvature in the data. To see how much better the quadratic model is when compared to the simpler linear model we can use `AIC` (see p. 232) or `anova` (see p. 172):

```
AIC(model2,model3)
```

```
       df      AIC
model2  3 257.0016
model3  4 230.4445
```

The much lower AIC of the quadratic `model3` means that it is preferred (see p. 232 for details). Alternatively, if you like p values, then comparison of the two models by `anova` shows that the curvature is highly significant ($p < 0.000001$):

```
anova(model2,model3)
```

```
Analysis of Variance Table

Model 1:  amount ~ time
Model 2:  amount ~ time + I(time^2)
 Res.Df     RSS Df Sum of Sq     F    Pr(>F)
1     29 5960.6
2     28 2372.6  1    3588.1 42.344 4.727e-07 ***
```

Non-Linear Regression

Sometimes we have a mechanistic model for the relationship between y and x, and we want to estimate the parameters and standard errors of the parameters of a specific non-linear equation from data. There are a number of frequently-used non-linear models to choose from. What we mean in this case by non-linear is not that the relationship is curved (it was curved in the case of polynomial regressions, but these were linear models), but that the relationship cannot be linearized by transformation of the response variable or the explanatory variable (or both). Here is an example: it shows jaw bone length as a function of age in deer. Theory indicates that the relationship is an ‘asymptotic exponential’ with three parameters:

$$y = a - be^{-cx}$$

In R, the main difference between linear models and non-linear models is that we have to tell R the exact nature of the equation as part of the model formula when we use non-linear modelling. In place of `lm` we write `nls` (this stands for ‘nonlinear least squares’). Then we write `y~a-b*exp(-c*x)` to spell out the precise nonlinear model we want R to fit to the data. The slightly tedious thing is that R requires us to specify initial guesses at the values of the parameters a, b and c (note, however, that some common non-linear models have ‘self-starting’ versions in R which bypass this step). Let us plot the data and work out sensible starting values. It always helps in cases like this to evaluate the equation’s ‘behaviour at the limits’. That is to say, the values of y when $x = 0$ and when $x =$ infinity. For $x = 0$, we have `exp(-0)` which is 1, and $1 \times b = b$ so $y = a - b$. For $x =$ infinity, we have `exp(-infinity)` which is 0, and $0 \times b = 0$ so $y = a$. That is to say, the asymptotic value of y is a, and the intercept is $a - b$. If you need to check your maths, you can do calculations with infinity and zero in R like this:

```
exp(-Inf)
```

```
[1] 0
```

```
exp(-0)
```

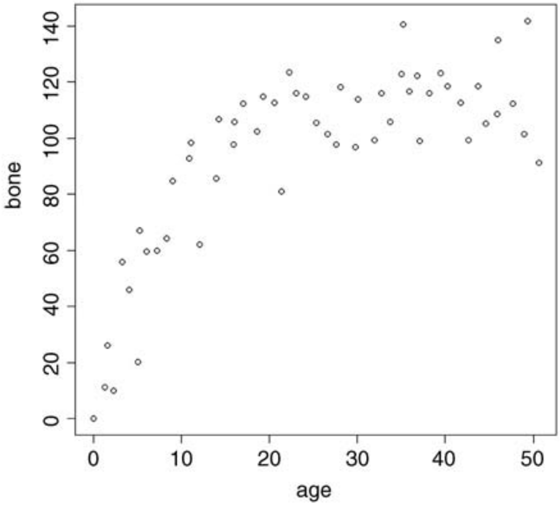
```
[1] 1
```

Here are the data on bone length as a function of age:

```
deer <- read.csv("c:\\temp\\jaws.csv")
attach(deer)
names(deer)
```

```
[1] "age" "bone"
```

```
par(mfrow=c(1,1))
plot(age,bone,pch=21,bg="lightgrey")
```



Inspection suggests that a reasonable estimate of the asymptote is $a \approx 120$ and intercept ≈ 10, so $b = 120 - 10 = 110$. Our guess of the value of c is slightly harder. Where the curve is rising most steeply, jaw length is about 40 where age is 5; rearranging the equation gives

$$c = -\frac{\log((a-y)/b)}{x} = -\frac{\log(120-40)/110)}{5} = 0.06369075$$

Now that we have the three parameter estimates, we can provide them to R as the starting conditions as part of the `nls` call like this: `list(a = 120, b = 110, c = 0.064)`

```
model <- nls(bone~a-b*exp(-c*age),start=list(a=120,b=110,c=0.064))
summary(model)
```

```
Formula: bone ~ a - b * exp(-c * age)

Parameters:
   Estimate  Std. Error  t value    Pr(>|t|)
a  115.2528      2.9139    39.55    < 2e-16 ***
b  118.6875      7.8925    15.04    < 2e-16 ***
c    0.1235      0.0171     7.22   2.44e-09 ***

Residual standard error: 13.21 on 51 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 2.381e-06
```

All the parameters appear to be significant at $p < 0.001$. Beware, however. This does not necessarily mean that all the parameters need to be retained in the model. In this case, $a = 115.2528$ with $SE = 2.9139$ is clearly not significantly different from $b = 118.6875$ with $SE = 7.8925$ (they would need to differ by more than 2 standard errors to be significant). So we should try fitting the simpler two-parameter model

$$y = a(1 - e^{-cx})$$

```
model2 <- nls(bone~a*(1-exp(-c*age)),start=list(a=120,c=0.064))
anova(model,model2)
```

```
Analysis of Variance Table

Model 1: bone ~ a - b * exp(-c * age)
Model 2: bone ~ a * (1 - exp(-c * age))
 Res.Df Res.Sum Sq Df Sum Sq F value Pr(>F)
1   51   8897.3
2   52   8929.1 -1 -31.843 0.1825 0.671
```

Model simplification was clearly justified ($p = 0.671$), so we accept the two-parameter version, `model2`, as our minimal adequate model. We finish by plotting the curve through the scatterplot. The age variable needs to go from 0 to 50:

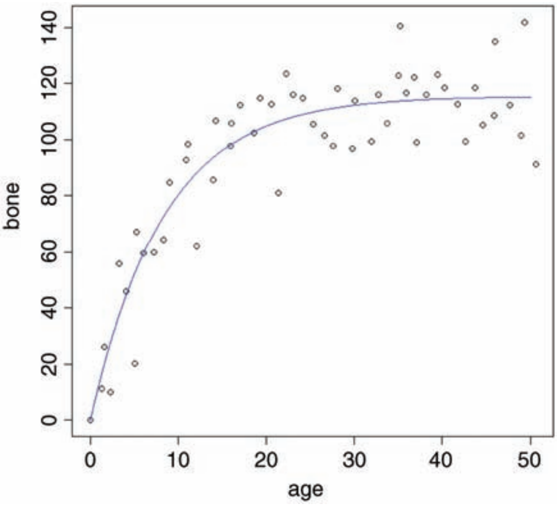
```
av <- seq(0,50,0.1)
```

and we use `predict` with `model2` to generate the predicted bone lengths:

```
bv <- predict(model2,list(age=av))
```

Note the use of `list` to assign our steps along the x axis (called `av`) to the variable used for the x axis in `model2` (called `age`).

```
lines(av,bv,col="blue")
```



The parameters of this curve are obtained from `model2`:

```
summary(model2)

Formula: bone ~ a * (1 - exp(-c * age))

Parameters:
      Estimate   Std. Error  t value    Pr(>|t|)
a    115.58056     2.84365   40.645    < 2e-16 ***
c      0.11882     0.01233    9.635   3.69e-13 ***

Residual standard error: 13.1 on 52 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 1.356e-06
```

which we could write like this $y = 115.58(1 - e^{-0.1188x})$ or like this $y = 115.58(1 - \exp(-0.1188\ x))$ according to taste or journal style. If you want to present the standard errors as well as the parameter estimates, you could write:

> The model $y = a\,(1 - \exp(-b\,x))$ had $a = 115.58 \pm 2.84$ (1 s.e., $n = 54$) and $b = 0.1188 \pm 0.0123$ (1 s.e.) and explained 84.9% of the total variation in bone length.

Note that because there are only two parameters in the minimal adequate model, we have called them a and b (rather than a and c as in the original formulation).

You may be puzzled as to how we know that `model2` explained 84.9% of the total variation in bone length, because the `summary` does not give an r-squared figure. We need to do a little more work to find SSY and SSR (see p. 123). The easiest way to find SSY is to fit a null model, estimating only the intercept. In R, the intercept is parameter 1 and is fitted like this: `y~1`. The sum of squares associated with this model is SSY:

```
null.model <- lm(bone ~ 1)
summary.aov(null.model)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
Residuals   53  59008    1113
```

The key figure to extract from this is the total sum of squares $SSY = 59\,008$. The non-linear output (above) did not give us either *SSE* or *SSR* but it did print:

```
Residual standard error: 13.1 on 52 degrees of freedom
```

This is useful because we can get the residual variance by squaring the residual standard error ($13.1^2 = 171.61$) and convert this into the residual sum of squares *SSE* by multiplying it by the degrees of freedom ($52 \times 13.1^2 = 8923.72$). Recall that *r*-squared is *SSR*/*SST* expressed as a percentage, and that $SSR = SSY - SSE$. Thus, the fraction of the variance in bone length explained by our model is

```
100*(59008-8923.72)/59008
```

```
[1] 84.8771
```

Generalized Additive Models

Sometimes we can see that the relationship between *y* and *x* is non-linear but we do not have any theory or any mechanistic model to suggest a particular functional form (mathematical equation) to describe the relationship. In such circumstances, generalized additive models are particularly useful, because they fit non-parametric smoothers to the data without requiring us to specify any particular mathematical model to describe the non-linearity. This will become clear with an example.

```
library(mgcv)
hump <- read.csv("c:\\temp\\hump.csv")
attach(hump)
names(hump)
```

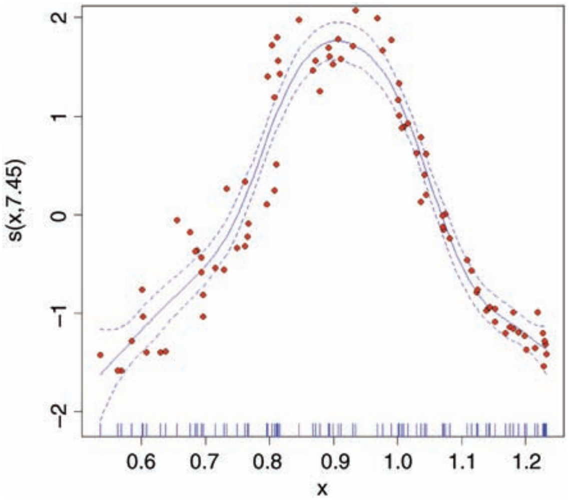
```
[1] "y" "x"
```

We start by fitting the generalized additive model as a smoothed function of *x*, `s(x)`:

```
model <- gam(y~s(x))
```

Then we plot the model, and overlay the scatterplot of data points:

```
plot(model,col="blue")
points(x,y-mean(y),pch=21,bg="red")
```



The *y* axis is labelled `s(x,7.45)` which is interpreted as saying that the smoothed function of *x* shown by the solid blue curve ('the non-parametric smoother') involves the equivalent of 7.45 degrees of freedom (remember that a straight line would use 2 d.f.: the intercept and the slope). The dotted lines show the confidence interval for the location of the smoothed function of *x*. On the *x* axis you see a rug plot, showing where the *x* points on the graph are located.

The model summary is obtained in the usual way:

```
summary(model)
```

```
Family: gaussian
Link function: identity

Formula:
y ~ s(x)

Parametric coefficients:
             Estimate  Std. Error  t value  Pr(>|t|)
(Intercept)   1.95737     0.03446     56.8   <2e-16 ***

Approximate significance of smooth terms:
       edf   Ref.df      F   p-value
s(x) 7.452   8.403   116.9   <2e-16 ***

R-sq.(adj) = 0.919  Deviance explained = 92.6%
GCV score = 0.1156 Scale est. = 0.1045  n = 88
```

This shows that the humped relationship between *y* and *x* is highly significant (the *p* value of the smooth term `s(x)` is less than 0.0000001). The fitted function explains 91.9% of the variance in y ($r^2 = 0.919$). The intercept (1.95737) is just the mean value of *y*.

Note that because of the strong hump in the relationship, a linear model `lm(y~x)` indicates no significant relationship between the two variables ($p = 0.346$). This is an object lesson in always plotting the data before you come to conclusions from the statistical analysis; in this case, if you had started with a linear model you would have thrown out the baby with the bathwater by concluding that nothing was happening. In fact, something very significant is happening but it is producing a humped, rather than a trended relationship.

Influence

One of the commonest reasons for a lack of fit is through the existence of outliers in the data. It is important to understand, however, that a point may *appear* to be an outlier because of misspecification of the model, and not because there is anything wrong with the data.

Take this circle of data that shows absolutely no relationship between y and x:

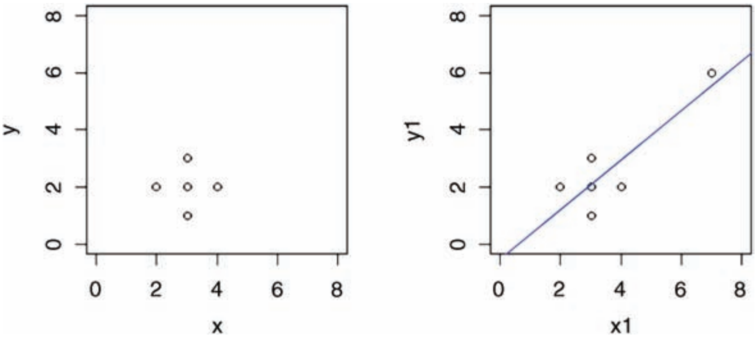
```
x <- c(2,3,3,3,4)
y <- c(2,3,2,1,2)
```

We want to draw two graphs side by side, and we want them to have the same axis scales:

```
windows(7,4)
par(mfrow=c(1,2))
plot(x,y,xlim=c(0,8),ylim=c(0,8))
```

Obviously, there is no relationship between y and x in the original data. But let’s add an outlier at the point (7,6) using concatenation `c` and see what happens.

```
x1 <- c(x,7)
y1 <- c(y,6)
plot(x1,y1,xlim=c(0,8),ylim=c(0,8))
abline(lm(y1~x1),col="blue")
```



Now, there is a significant regression of y on x. The outlier is said to be highly *influential*.

Testing for the presence of influential points is an important part of statistical modelling. You cannot rely on analysis of the residuals, because by their very influence, these points force the regression line close to them.

Measures of leverage for a given data point y are proportional to $(x-\bar{x})^2$. The commonest measure of leverage is

$$h_i = \frac{1}{n} + \frac{(x_i-\bar{x})^2}{\sum (x_i-\bar{x})^2}$$

where the denominator is *SSX*. A good rule of thumb is that a point is highly influential if its

$$h_i > \frac{2p}{n},$$

where p is the number of parameters in the model. There is a useful function called `influence.measures` which highlights influential points in a given model

```
reg <- lm(y1~x1)
influence.measures(reg)
```

```
Influence measures of
         lm(formula = y1 ~ x1) :

  dfb.1_   dfb.x1  dffit  cov.r   cook.d   hat  inf
1  0.687 -0.5287 0.7326  1.529 0.26791 0.348
2  0.382 -0.2036 0.5290  1.155 0.13485 0.196
3 -0.031  0.0165 -0.0429 2.199 0.00122 0.196
4 -0.496  0.2645 -0.6871 0.815 0.19111 0.196
5 -0.105 -0.1052 -0.5156 1.066 0.12472 0.174
6 -3.023  4.1703 4.6251  4.679 7.62791 0.891    *
```

You can see point #6 is highlighted by an asterisk, drawing attention to its high influence.

Further Reading

Cook, R.D. and Weisberg, S. (1982) *Residuals and Influence in Regression*, Chapman & Hall, New York.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models*, Chapman & Hall, London.

Wetherill, G.B., Duncombe, P., Kenward, M. *et al.* (1986) *Regression Analysis with Applications*, Chapman & Hall, London.

8

Analysis of Variance

Analysis of variance or ANOVA is the technique we use when all the explanatory variables are categorical. The explanatory variables are called *factors*, and each factor has two or more *levels*. When there is a single factor with three or more levels we use one-way ANOVA. If we had a single factor with just two levels, we would use Student's t test (see p. 97), and this would give us exactly the same answer that we would have obtained by ANOVA (remember the rule that with just two levels then $F = t^2$). Where there are two or more factors, then we use two-way or three-way ANOVA, depending on the number of explanatory variables. When there is replication at each level in a multi-way ANOVA, the experiment is called a *factorial* design, and this allows us to study *interactions* between variables, in which we test whether *the response to one factor depends on the level of another factor*.

One-Way ANOVA

There is a real paradox about analysis of variance, which often stands in the way of a clear understanding of exactly what is going on. The idea of ANOVA is to compare two or more means. But it does this by comparing variances. How can that work?

The best way to see what is happening is to work through a graphical example. To keep things as simple as possible, we shall use a factor with just two levels at this stage, but the argument extends to any number of levels. Suppose that we have atmospheric ozone concentrations measured in parts per hundred million (pphm) in two commercial lettuce-growing gardens (we shall call the gardens A and B for simplicity).

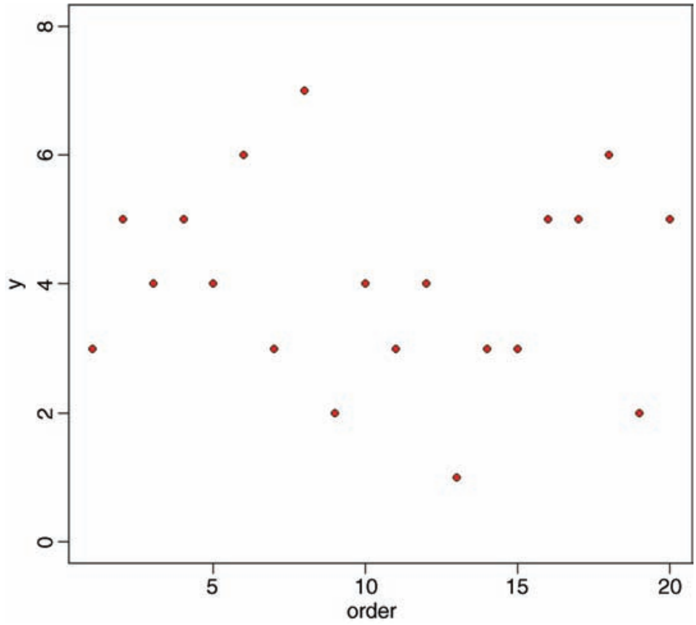
```
oneway <- read.csv("c:\\temp\\oneway.csv")
attach(oneway)
names(oneway)
```

```
[1] "ozone" "garden"
```

As usual, we begin by plotting the data, but here we plot the y values (ozone concentrations) against the order in which they were measured:

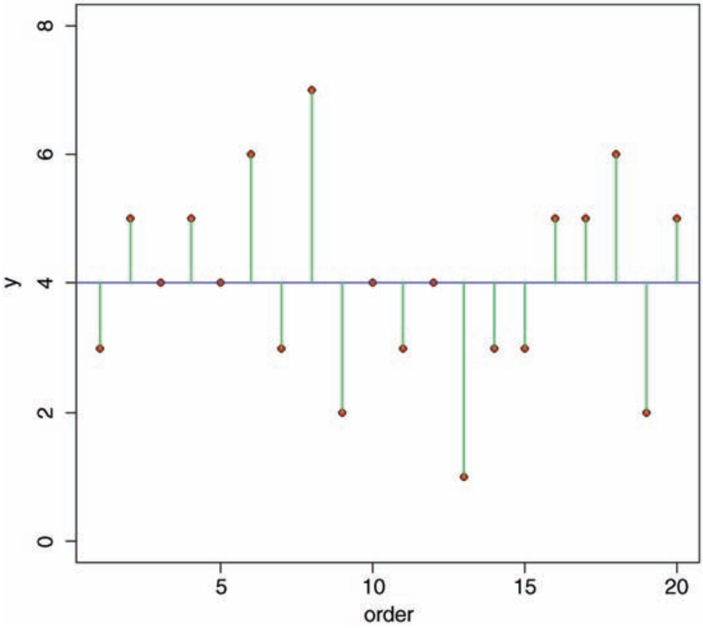
```
plot(1:20,ozone,ylim=c(0,8),ylab="y",xlab="order",pch=21,bg="red")
```

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.



There is lots of scatter, indicating that the variance in *y* is large. To get a feel for the overall variance, we can plot the mean value of *y*, and indicate each of the residuals by a vertical line from `mean(y)` to the value of `y`, like this:

```
abline(h=mean(ozone),col="blue")
for(i in 1:20) lines(c(i,i),c(mean(ozone),ozone[i]),col="green")
```



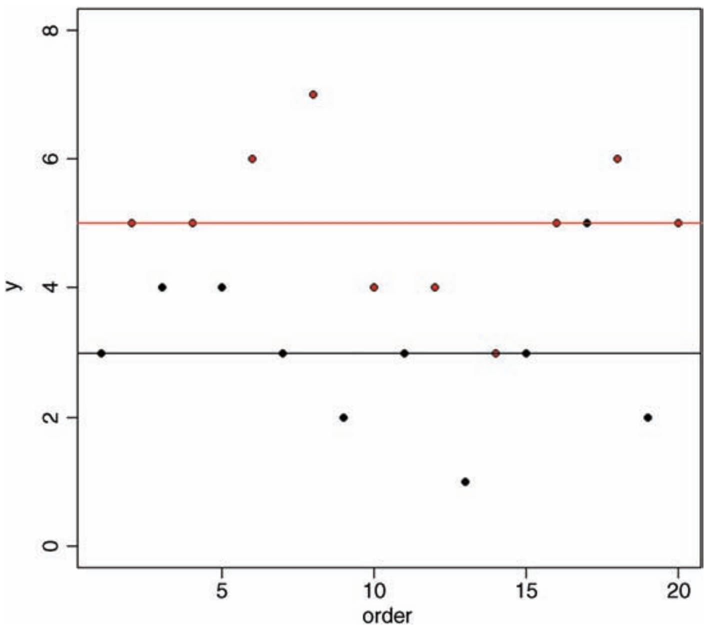
We refer to this overall variation as the *total sum of squares*, *SSY*, which is the sum of the squares of the lengths of the green lines. More formally, this is given by:

$$SSY = \sum (y - \bar{y})^2$$

which should look familiar, because it is the formula used for defining the variance of y ($s^2 =$ sum of squares/degrees of freedom; see p. 51).

This next step is the key to understanding how ANOVA works. Instead of fitting the overall mean value of y thorough the data, and looking at the departures of all the data points from the overall mean, let us fit the individual treatment means (the mean for garden A and the mean for garden B in this case), and look at the departures of data points from the appropriate treatment mean. It will be useful if we have different colours for the different gardens; say, black for garden A (`col= 1`) and red for garden B (`col = 2`). Note the use of `as.numeric` to turn the factor levels A and B into the numbers 1 and 2 for the purposes of selecting the colours (`bg`) for the plotting symbols.

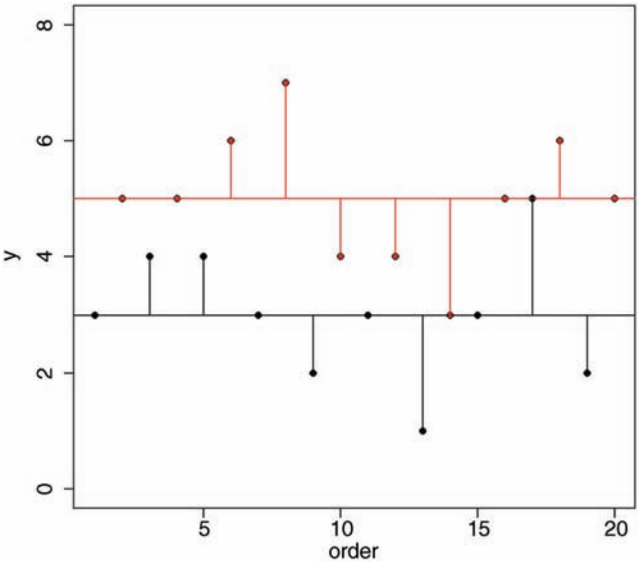
```
plot(1:20,ozone,ylim=c(0,8),ylab="y",xlab="order",
                                 pch=21,bg=as.numeric(garden))
abline(h=mean(ozone[garden=="A"]))
abline(h=mean(ozone[garden=="B"]),col="red")
```



Now it is clear that the mean ozone concentration in garden B (red) is substantially higher than in garden A. The aim of ANOVA is to determine whether it is significantly higher, or whether this kind of difference could come about by chance alone, when the mean ozone concentrations in the two gardens were really the same.

Now we draw the residuals – the differences between the measured ozone concentrations, and the mean for the garden involved:

```
index <- 1:length(ozone)
for (i in 1:length(index)){
if (garden[i] == "A")
    lines(c(index[i],index[i]),c(mean(ozone[garden=="A"]),ozone
[i]))
else
    lines(c(index[i],index[i]),c(mean(ozone[garden=="B"]),ozone
[i]), col="red")
}
```



Now here's the quiz. If the means in the two gardens are not significantly different, what should be the difference in the lengths of the red and black residual lines in this figure and the green lines in the first figure we drew? After a bit of thought, you should see that if the means were the same, then the red and black horizontal lines in this figure would be in the same place, and hence the lengths of the residual lines would be the same as in the previous figure.

We are half way there. Now, suppose that mean ozone concentration is *different* in the two gardens. Would the residual lines be bigger or smaller when we compute them from the individual treatment means (the red and black lines, as above), or from the overall mean (the green lines in the previous figure)? They would be *smaller* when computed from the individual treatment means *if the individual treatment means were different*.

So there it is. That is how ANOVA works. *When the means are significantly different, then the sum of squares computed from the individual treatment means will be smaller than the sum of squares computed from the overall mean*. We judge the significance of the difference between the two sums of squares using analysis of variance.

The analysis is formalized by defining this new sum of squares: it is the sum of the squares of the differences between the individual y values and the relevant treatment mean.

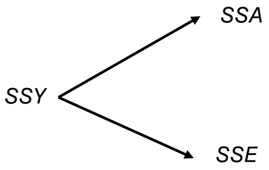
We shall call this *SSE*, the *error sum of squares* (there has been no error in the sense of a mistake; 'error' is used here as a synonym of 'residual'). It is the sum of the squares of the lengths of the vertical red lines added to the sum of the squares of the lengths of the vertical black lines in the graphic above:

$$SSE = \sum_{j=1}^{k} \sum (y - \bar{y}_j)^2$$

We compute the mean for the *j*th level of the factor in advance, and then add up the squares of the differences. Given that we worked it out this way, can you see how many degrees of freedom should be associated with *SSE*? Suppose that there were *n* replicates in each treatment ($n = 10$ in our example). And suppose that there are *k* levels of the factor ($k = 2$ in our example). If you estimate *k* parameters from the data before you can work out *SSE*, then you must have lost *k* degrees of freedom in the process. Since each of the *k* levels of the factor has *n* replicates, there must be $k \times n$ numbers in the whole experiment ($2 \times 10 = 20$ in our example). So the degrees of freedom associated with *SSE* is $kn - k = k(n - 1)$. Another way of seeing this is to say that there are *n* replicates in each treatment, and hence $n - 1$ degrees of freedom for error in each treatment (because 1 d.f. is lost in estimating each treatment mean). There are *k* treatments (i.e. *k* levels of the factor) and hence there are $k(n - 1)$ d.f. for error in the experiment as a whole.

Now we come to the 'analysis' part of the analysis of variance. The total sum of squares in *y*, *SSY*, is broken up (analysed) into components. The unexplained part of the variation is called the error sum of squares, *SSE*. The component of the variation that is explained by differences between the treatment means is called the treatment sum of squares, and is traditionally denoted by *SSA*. This is because in two-way ANOVA, with two different categorical explanatory variables, we shall use *SSB* to denote the sum of squares attributable to differences between the means of the second factor. And *SSC* to denote the sum of squares attributable to differences between the means of the third factor. And so on.

Analysis of variance, therefore, is based on the notion that we break down the total sum of squares, *SSY*, into useful and informative components.



Typically, we compute all but one of the components, then find the value of the last component by subtraction of the others from *SSY*. We already have a formula for *SSE*, so we could obtain *SSA* by difference: $SSA = SSY - SSE$. Starting with *SSY*, we calculate the sum of the squares of the differences between the *y* values and the overall mean:

```
SSY <- sum((ozone-mean(ozone))^2)
SSY
```

```
[1] 44
```

The question now is: 'How much of this 44 is attributable to differences between the means of gardens A and B (*SSA* = explained variation) and how much is sampling error

(*SSE* = unexplained variation)?’ We have a formula defining *SSE*; it is the sum of the squares of the residuals calculated separately for each garden, using the appropriate mean value. For garden A, we get

```
sum((ozone[garden=="A"]-mean(ozone[garden=="A"]))^2)
```

```
[1] 12
```

and for garden B

```
sum((ozone[garden=="B"]-mean(ozone[garden=="B"]))^2)
```

```
[1] 12
```

so the error sum of squares is the total of these components $SSE = 12 + 12 = 24$. Finally, we can obtain the treatment sum of squares, *SSA*, by difference:

$$SSA = 44 - 24 = 20$$

At this point, we can fill in the ANOVA table (see p. 128):

Source	Sum of squares	Degrees of freedom	Mean square	*F* ratio
Garden	20.0	1	20.0	15.0
Error	24.0	18	$s^2 = 1.3333$	
Total	44.0	19		

We need to test whether an *F* ratio of 15.0 is large or small. To do this we compare it with the critical value of *F* from quantiles of the *F* distribution, `qf`. We have 1 degree of freedom in the numerator, and 18 degrees of freedom in the denominator, and we want to work at 95% certainty ($\alpha = 0.05$):

```
qf(0.95,1,18)
```

```
[1] 4.413873
```

The calculated value of the *test statistic* of 15.0 is much greater than the *critical value* of $F = 4.41$, so we can reject the null hypothesis (equality of the means) and accept the alternative hypothesis (the two means are significantly different). We used a one-tailed *F* test (0.95 rather than 0.975 in the `qf` function) because we are only interested in the case where the treatment variance is large relative to the error variance. But this approach is rather old-fashioned; the modern view is to calculate the *effect size* (the difference between the means is 2.0 pphm ozone) and to state the probability that such a difference would arise by chance alone when the null hypothesis was true and the mean ozone concentration was the same in the two gardens. For this we use cumulative probabilities of the *F* distribution, rather than quantiles, like this:

```
1-pf(15.0,1,18)
```

```
 [1] 0.001114539
```

So the probability of obtaining data as extreme as ours (or more extreme) if the two means really were the same is roughly 0.1%.

That was quite a lot of work. Here is the whole analysis in R in a single line:

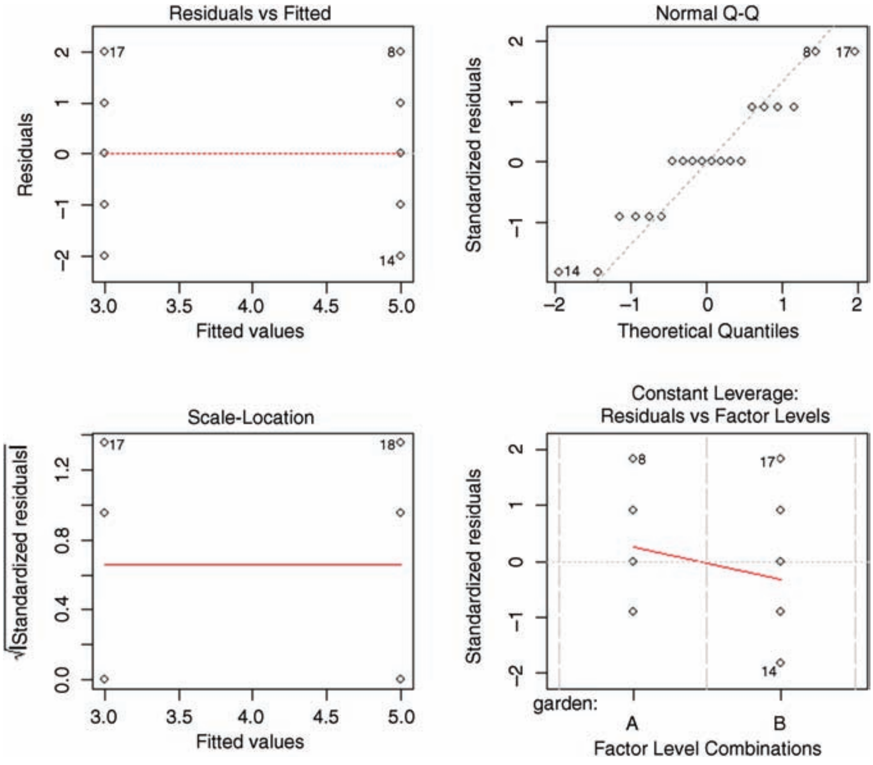
```
summary(aov(ozone~garden))
```

```
            Df Sum  Sq  Mean Sq   F value   Pr(>F)
garden       1       20  20.000        15  0.00111 **
Residuals   18       24   1.333
```

The first column shows the sources of variation (*SSA* and *SSE*, respectively); note that R leaves off the bottom row that we included for total variation, *SSY*. The next column shows degrees of freedom: there are two levels of garden (A and B) so there is $2-1=1$ d.f. for garden, and there are 10 replicates per garden, so $10-1=9$ d.f. per garden and two gardens, so error d.f. $=2\times 9=18$. The next column shows the sums of squares: $SSA=20$ and $SSE=24$. The fourth column gives the mean squares (sums of squares divided by degrees of freedom); the treatment mean square is 20.0 and the error variance, s^2 (synonym of the residual mean square) is $24/18=1.3333$. The F ratio is $20/1.333=15$, and the probability that this (or a result more extreme than this) would arise by chance alone if the two means really were the same is 0.001115 (just as we calculated long-hand, above).

We finish by carrying out graphical checks of the assumptions of the model: namely, constancy of variance and normality of errors.

```
plot(aov(ozone~garden))
```



The first plot shows that the variances are identical in the two treatments (this is exactly what we want to see). The second plot shows a reasonably straight line relationship on the normal quantile–quantile plot (especially since, in this example, the *y* values are whole numbers), so we can be confident that non-normality of errors is not a major problem. The third plot shows the residuals against the fitted values on a different scale (constant variance again), and the fourth plot shows Cook's statistics, drawing attention to the fact that points 8, 17 and 14 have large residuals.

Shortcut Formulas

In the unlikely event that you ever need to do analysis of variance using a calculator, then it is useful to know the short-cut formula for calculating *SSA*. We calculated it by difference, above, having worked out *SSE* long-hand. To work out *SSA* from first principles, the thing you need to understand is what we mean by a 'treatment total'. The treatment total is simply the sum of the *y* values within a particular factor level. For our two gardens we have:

```
cbind(ozone[garden=="A"],ozone[garden=="B"])
```

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

```
tapply(ozone,garden,sum)
```

```
 A  B
30 50
```

The totals for gardens A and B are 30 and 50 respectively, and we will call these T_1 and T_2. The shortcut formula for *SSA* (Box 8.1) is then:

$$SSA = \frac{\sum T_i^2}{n} - \frac{\left(\sum y\right)^2}{kn}$$

We should confirm that this really does give *SSA*:

$$SSA = \frac{30^2 + 50^2}{10} - \frac{80^2}{2 \times 10} = \frac{3400}{10} - \frac{6400}{20} = 340 - 320 = 20$$

which checks out. In all sorts of analysis of variance, the key point to realize is that the sum of the subtotals squared is always *divided by the number of numbers that were added together to get each subtotal*. That sounds complicated, but the idea is simple. In our case we squared the subtotals T_1 and T_2 and added the results together. We divided by 10 because T_1 and T_2 were each the sum of 10 numbers.

Box 8.1. Corrected sums of squares in one-way ANOVA

The total sum of squares, *SSY*, is defined as the sum of the squares of the differences between all the data points, *y*, and the overall mean, $\bar{\bar{y}}$:

$$SSY = \sum_{i=1}^{k} \sum (y - \bar{\bar{y}})^2$$

where the inner $\sum$ means the sum over the *n* replicates within each of the *k* factor levels and the outer $\sum$ means add the subtotals across each of the factor levels. The error sum of squares, *SSE*, is the sum of the squares of the differences between the data points, *y*, and their individual treatment means, $\bar{y}_i$:

$$SSE = \sum_{i=1}^{k} \sum (y - \bar{y}_i)^2$$

The treatment sum of squares, *SSA*, is the sum of the squares of the differences between the individual treatment means, $\bar{y}_i$, and the overall mean, $\bar{\bar{y}}$:

$$SSA = \sum_{i=1}^{k} \sum_{j=1}^{n} (\bar{y}_i - \bar{\bar{y}})^2 = n \sum_{i=1}^{k} (\bar{y}_i - \bar{\bar{y}})^2$$

Squaring the bracketed term and applying summation gives

$$\sum \bar{y}_i^2 - 2\bar{\bar{y}} \sum \bar{y}_i + k.\bar{\bar{y}}^2$$

Now replace $\bar{y}_i$ by T_i/n (where T_i is our conventional name for the *k* individual treatment totals) and replace $\bar{\bar{y}}$ by $\sum y/k.n$ to get

$$\frac{\sum_{i=1}^{k} T_i^2}{n^2} - 2\frac{\sum y \sum_{i=1}^{k} T_i}{n.k.n} + k\frac{\sum y \sum y}{k.n.k.n}$$

Note that $\sum_{i=1}^{k} T_i = \sum_{i=1}^{j} \sum_{j=1}^{n} y_{ij}$ so the right-hand positive and negative terms both have the form $\left(\sum y\right)^2/k.n^2$. Finally, multiplying through by *n* gives

$$SSA = \frac{\sum T^2}{n} - \frac{\left(\sum y\right)^2}{k.n}$$

As an exercise, you should prove that $SSY = SSA + SSE$ (for a hint, see Box 7.4).

Effect Sizes

So far we have concentrated on hypothesis testing, using summary.aov. It is usually more informative to investigate the effects of the different factor levels, using summary.lm like this:

```
summary.lm(aov(ozone~garden))
```

It was easy to interpret this kind of output in the context of a regression, where the rows represent parameters that are intuitive: namely, the intercept and the slope. In the context of analysis of variance, it takes a fair bit of practice before the meaning of this kind of output is transparent.

```
Coefficients:
            Estimate Std.  Error  t value   Pr(>|t|)
(Intercept)   3.0000     0.3651     8.216  1.67e-07 ***
gardenB       2.0000     0.5164     3.873   0.00111 **

Residual standard error: 1.155 on 18 degrees of freedom
Multiple R-squared: 0.4545,  Adjusted R-squared: 0.4242
F-statistic:  15 on 1 and 18 DF, p-value: 0.001115
```

The rows are labelled (Intercept) and gardenB. But what do the parameter estimates 3.0 and 2.0 actually mean? And why are the standard errors different in the two rows (0.3651 and 0.5164)? After all, the variances in the two gardens were identical.

To understand the answers to these questions, we need to know how the equation for the explanatory variables is structured when the explanatory variable, as here, is categorical. To recap, the linear regression model is written as

```
lm(y~x)
```

which R interprets as the two-parameter linear equation

$$y = a + bx$$

in which the values of the parameters a and b are to be estimated from the data. But what about our analysis of variance? We have one explanatory variable, $x =$ ‘garden’, with two levels, A and B. The aov model is exactly analogous to the regression model

```
aov(y~x)
```

But what is the associated equation? Let us look at the equation first, then try to understand it:

$$y = a + bx_1 + cx_2$$

This looks just like a multiple regression, with two explanatory variables, x_1 and x_2. The key point to understand is that x_1 and x_2 are *the levels of the factor called* x. If ‘garden’ were a four-level factor, then the equation would have four explanatory variables in it, $x_1, \ldots, x_4$. With a categorical explanatory variable, the levels are all coded as 0 except for the level associated with the y value in question, which is coded as 1. You will find this hard to understand without a good deal of practice. Let us look at the first row of data in our dataframe:

```
garden[1]
```

```
[1] A
```

So the first ozone value in the data frame comes from garden A. This means that $x_1 = 1$ and $x_2 = 0$. The equation for the first row therefore looks like this:

$$y = a + b \times 1 + c \times 0 = a + b \times 1 = a + b$$

What about the second row of the dataframe?

```
garden[2]
```

```
[1] B
```

Because this row refers to garden B, x_1 is coded as 0 and x_2 is coded as 1 so the equation for the second row is:

$$y = a + b \times 0 + c \times 1 = a + c \times 1 = a + c$$

So what does this tell us about the parameters a, b and c? And why do we have three parameters, when the experiment generates only two means ? These are the crucial questions for understanding the `summary.lm` output from an analysis of variance. The simplest interpretation of the three-parameter case that we have dealt with so far is that the intercept a is the overall mean from the experiment:

```
mean(ozone)
```

```
[1] 4
```

Then, if a is the overall mean, so $a + b$ must be the mean for garden A and $a + c$ must be the mean for garden B (see the equations above). If that is true, then b must be *the difference between the mean of garden A and the overall mean*. And c must be *the difference between the mean of garden B and the overall mean*. Thus, the intercept is a *mean*, and the other parameters are *differences between means*. This explains why the standard errors are different in the different rows of the table: the standard error of the intercept is the standard error of a mean

$$SE_{\bar{y}} = \sqrt{\frac{s_A^2}{n_A}}$$

whereas the standard errors on the other rows are standard errors of the difference between two means:

$$SE_{\text{diff}} = \sqrt{\frac{s_{\text{A}}^2}{n_{\text{A}}} + \frac{s_{\text{B}}^2}{n_{\text{B}}}}$$

which is a bigger number (bigger by a factor of $1.4142 = \sqrt{2}$ if, as here, the sample sizes and variances are equal).

With three parameters, then, we should have b equal to the mean ozone concentration in garden A minus 4 and c equal to the mean ozone concentration in garden B minus 4.

```
mean(ozone[garden=="A"])-mean(ozone)
[1] -1
mean(ozone[garden=="B"])-mean(ozone)
[1] 1
```

That would be a perfectly reasonable way to parameterize the model for this analysis of variance. But it suffers from the fact that there is a redundant parameter. The experiment produces only two means (one for each garden), and so there is no point in having three parameters to represent the output of the experiment. One of the three parameters is said to be ‘aliased’ (see p. 16). There are lots of ways round this dilemma, as explained in detail in Chapter 12 on contrasts. Here we adopt the convention that is used as the default in R using so-called *treatment contrasts*. Under this convention, we dispense with the overall mean, a. So now we are left with the right number of parameters (b and c). In treatment contrasts, *the factor level that comes first in the alphabet is set equal to the intercept*. The other parameters are expressed as differences between this mean and the other relevant means. So, in our case, the mean of garden A becomes the intercept

```
mean(ozone[garden=="A"])
[1] 3
```

and the difference between the means of garden B and garden A is the second parameter:

```
mean(ozone[garden=="B"])-mean(ozone[garden=="A"])
[1] 2
```

Let us revisit our `summary.lm` table and see if it now makes sense:

```
Coefficients:
             Estimate Std. Error t value   Pr(>|t|)
 (Intercept)   3.0000     0.3651   8.216 1.67e-07 ***
gardenB        2.0000     0.5164   3.873  0.00111 **
```

The intercept is 3.0 which is the mean for garden A (because the factor level A comes before level B in the alphabet). The estimate for garden B is 2.0. This tells us that the mean ozone concentration in garden B is 2 pphm greater than in garden A (greater because there is no minus sign). We would compute the mean for garden B as 3.0 + 2.0 = 5.0. In practice, we would not obtain the means like this, but by using `tapply`, instead:

```
tapply(ozone, garden, mean)
```

```
A       B
3       5
```

There is more about these issues in Chapter 12.

Plots for Interpreting One-Way ANOVA

There are two traditional ways of plotting the results of ANOVA:

- box-and-whisker plots
- barplots with error bars

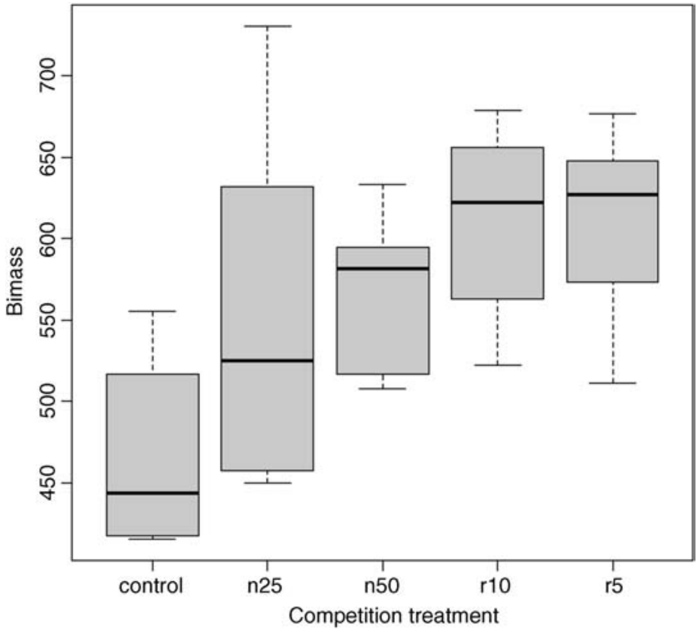
Here is an example to compare the two approaches. We have an experiment on plant competition where the response variable is `biomass` and we have one factor with five levels. The factor is called `clipping` and the levels are control (i.e. unclipped), two intensities of shoot pruning and two intensities of root pruning:

```
comp <- read.csv("c:\\temp\\competition.csv")
attach(comp)
names(comp)
```

```
[1] "biomass" "clipping"
```

This is what the data look like:

```
plot(clipping,biomass,xlab="Competition treatment",
                                  ylab="Biomass",col="lightgrey")
```



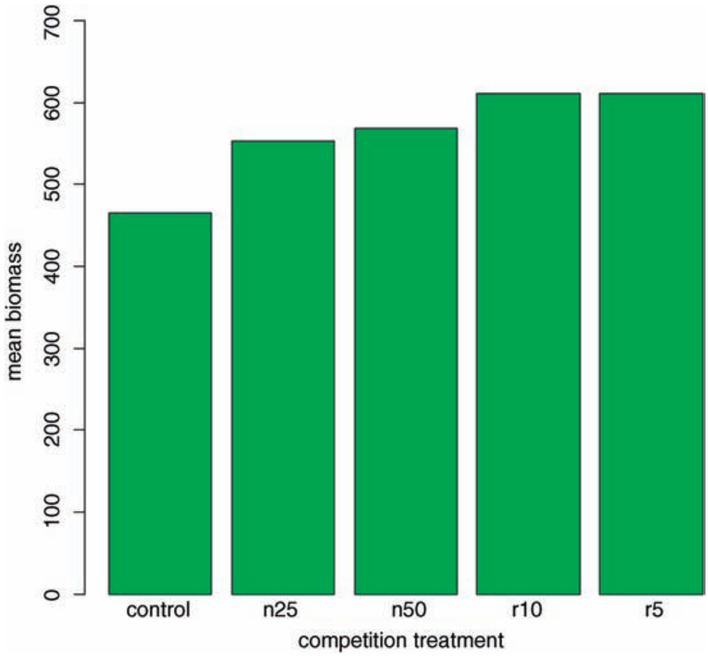
The box-and-whisker plot is good at showing the nature of the variation within each treatment, and also whether there is skew within each treatment (e.g. for the control plots, there is a wider range of values between the median and upper quartile than between the lower quartile and median). No outliers are shown above the whiskers, so the tops and bottoms of the bars are the maxima and minima within each treatment. The medians for the competition treatments are all higher than the upper quartile of the controls, suggesting that they may be significantly different from the controls, but there is little to suggest that any of the competition treatments are significantly different from one another (see below for the analysis).

Barplots with error bars are the style preferred by many journal editors, and some people think that they make hypothesis testing easier. We shall see. Unlike S-PLUS, R does not have a built-in function called `error.bar` so we shall have to write our own. First, the `barplot` on its own. We need to calculate the heights of the bars to represent the mean biomass from each of the five treatments, and the simplest way to do this is to use `tapply`:

```
heights <- tapply(biomass,clipping,mean)
```

Now draw the `barplot`, making sure that the *y* axis is long enough to accommodate the tops of the error bars that we intend to add later:

```
barplot(heights,col="green",ylim=c(0,700),
                           ylab="mean biomass",xlab="competition treatment")
```



This is fine as far as it goes, but it gives us no impression of the uncertainty associated with the estimated heights of the bars.

Here is a very simple function without any bells or whistles. We shall call it error.bars to distinguish it from the much more general S-PLUS function. We shall pass two arguments to the function; the heights of the bars *y* and the lengths of the error bars, *z* like this:

```
error.bars <- function(y,z) {
x <- barplot(y,plot=F)
n <- length(y)
for (i in 1:n)
     arrows(x[i],y[i]-z,x[i],y[i]+z,code=3,angle=90,length=0.15)
}
```

The second line works out the *x* coordinates of the centres of the bars without redrawing the barplot (plot = FALSE). The next line works out how many error bars are to be drawn (n <- length(y), which is 5 in this example). The trick here is to modify the arrows function to draw the errors bars with heads at right angles (angle=90) and heads at both ends of the arrow (code=3). The default head length looks a bit clunky, so we reduce the width of the head to 0.15 inches. We use a for loop to draw the n error bars separately.

To use this function we need to decide what kind of values (z) to use for the lengths of the bars. Let us use *one standard error of the mean* based on the pooled error variance from the ANOVA, then return to a discussion of the pros and cons of different kinds of error bars later. Here is the one-way ANOVA:

```
model <- aov(biomass~clipping)
summary(model)
```

```
            Df   Sum Sq   Mean Sq  F value    Pr(>F)
clipping     4    85356    21339   4.3015  0.008752 **
Residuals   25   124020     4961
```

From the ANOVA table we can see that the pooled error variance $s^2 = 4961$. Now we need to know how many numbers were used in the calculation of each of the five means:

```
table(clipping)
```

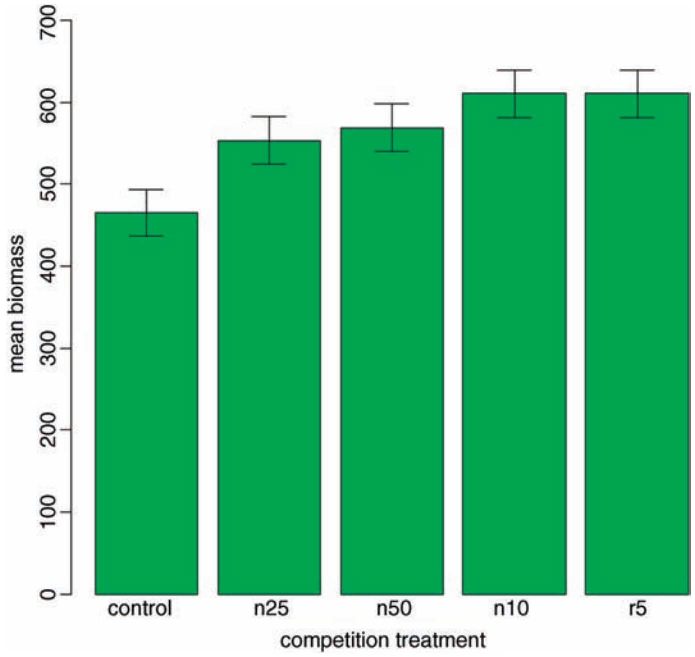
```
clipping
control  n25  n50  r10  r5
      6    6    6    6   6
```

There was equal replication (which makes life easier), and each mean was based on six replicates, so the standard error of the mean is $\sqrt{s^2/n} = \sqrt{4961/6} = 28.75$. We shall draw an error bar up 28.75 from each mean and down by the same distance, so we need five values, one for each bar, each of 28.75:

```
se <- rep(28.75,5)
```

Now we can use the new function to add the error bars to the plot:

```
error.bars(heights,se)
```

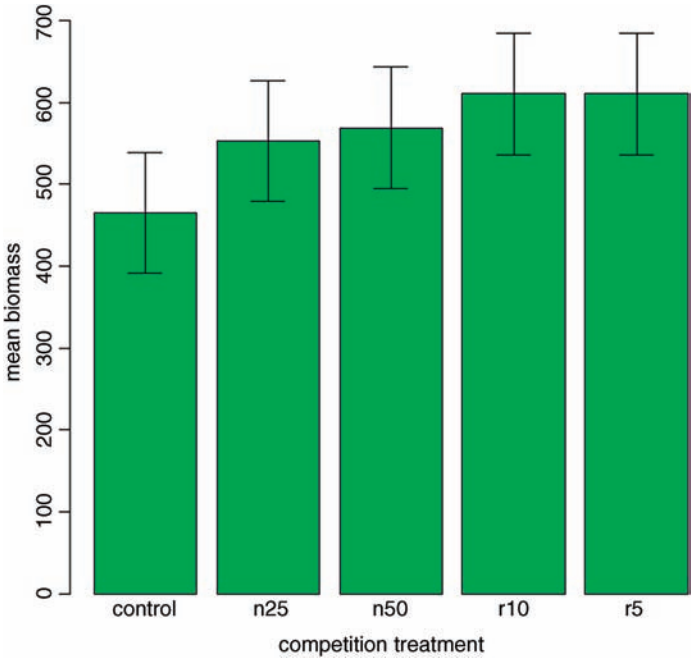


We do not get the same feel for the distribution of the values within each treatment as was obtained by the box-and-whisker plot, but we can certainly see clearly which means are *not* significantly different. If, as here, we use ± 1 standard error as the length of the error bars, then *when the bars overlap this implies that the two means are not significantly different*. Remember the rule of thumb for *t*: significance requires 2 or more standard errors, and if the bars overlap it means that the difference between the means is less than 2 standard errors. This shows clearly that none of the means for the clipped plants (n25, n50, r10 or r5) is significantly different from any other (the top of the bar for n25 overlaps the bottom of the bar for r10).

There is another issue, too. For comparing means, we should be using the standard error of the difference between two means (not the standard error of one mean) in our tests (see p. 91); these bars would be about 1.4 times as long as the bars we have drawn here. So while we can be sure that the pruning treatments are not significantly different from one another, we cannot conclude from this plot that the controls have significantly lower biomass than the rest (because the error bars are not the correct length for testing differences between means).

An alternative graphical method is to use 95% confidence intervals for the lengths of the bars, rather than standard errors of means. This is easy to do: we multiply our standard errors by Student's *t*, qt(.975,5) = 2.570582, to get the lengths of the confidence intervals:

```
ci <- se*qt(.975,5)
barplot(heights,col="green",ylim=c(0,700),
                    ylab="mean biomass",xlab="competition treatment")
error.bars(heights,ci)
```



Now, all of the error bars overlap, implying visually that there are no significant differences between the means. But we know that this is not true from our analysis of variance, in which we rejected the null hypothesis that all the means were the same at $p = 0.00875$. If it were the case that the bars did not overlap when we are using confidence intervals (as here), then that would imply that the means differed by more than 4 standard errors, and this is much greater than the difference required for significance. So this is not perfect either. With standard errors we could be sure that the means were *not* significantly different when the bars did overlap. And with confidence intervals we can be sure that the means *are* significantly different when the bars do not overlap. But the alternative cases are not clear-cut for either type of bar. Can we somehow get the best of both worlds, so that the means are significantly different when the bars do not overlap, and the means are not significantly different when the bars do overlap?

The answer is yes, we can, if we use LSD bars (LSD stands for 'least significant difference'). Let us revisit the formula for Student's t test:

$$t = \frac{\text{a difference}}{\text{standard error of the difference}}$$

We say that the difference is significant when $t > 2$ (by the rule of thumb, or $t >$ `qt(0.975,df)` if we want to be more precise). We can rearrange this formula to find the smallest difference that we would regard as being significant. We can call this the least significant difference:

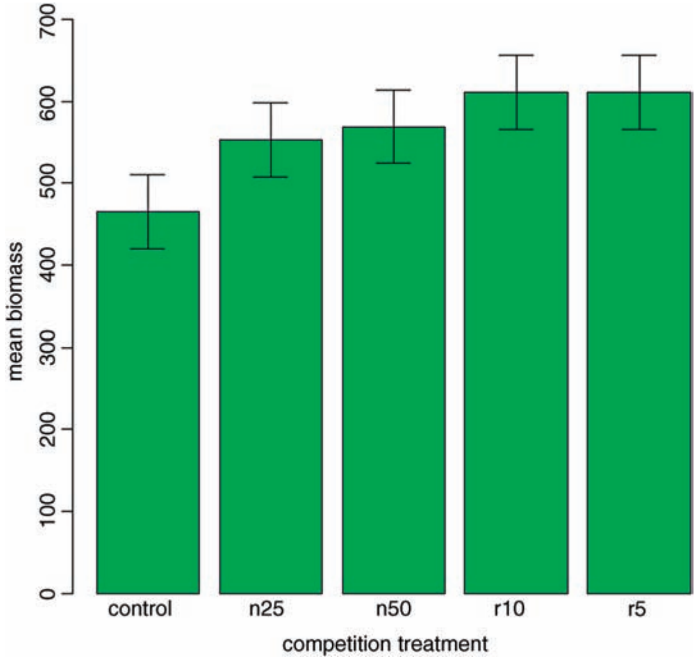
$$\text{LSD} = \texttt{qt(0.975,df)} \times \text{standard error of a difference} \approx 2 \times SE_{\text{diff}}$$

In our present example this is

```
qt(0.975,10)*sqrt(2*4961/6)
[1] 90.60794
```

because a difference is based on 12 – 2 = 10 degrees of freedom. What we are saying is that the two means would be significantly different if they differed by 90.61 or more. How can we show this graphically? We want overlapping bars to indicate a difference less than 90.61, and non-overlapping bars to represent a difference greater than 90.61. With a bit of thought you will realize that we need to draw bars that are LSD/2 in length, up and down from each mean. Let us try it with our current example:

```
lsd <- qt(0.975,10)*sqrt(2*4961/6)
lsdbars <- rep(lsd,5)/2

barplot(heights,col="green",ylim=c(0,700),
                  ylab="mean biomass",xlab="competition treatment")
error.bars(heights,lsdbars)
```



Now we can interpret the significant differences visually. The control biomass is significantly lower than any of the four treatments, but none of the four treatments is significantly different from any other. The statistical analysis of this contrast is explained in detail in Chapter 12. Sadly, most journal editors insist on error bars of 1 standard error. It is true that there are complicating issues to do with LSD bars (not least the vexed question of multiple comparisons; see p. 17), but at least LSD/2 bars do what was intended by the error plot (i.e. overlapping bars means non-significance and non-overlapping bars means significance). Neither standard errors nor confidence intervals can say that. A better option might be to use box-and-whisker plots with the `notch=T` option to indicate significance (see p. 93).

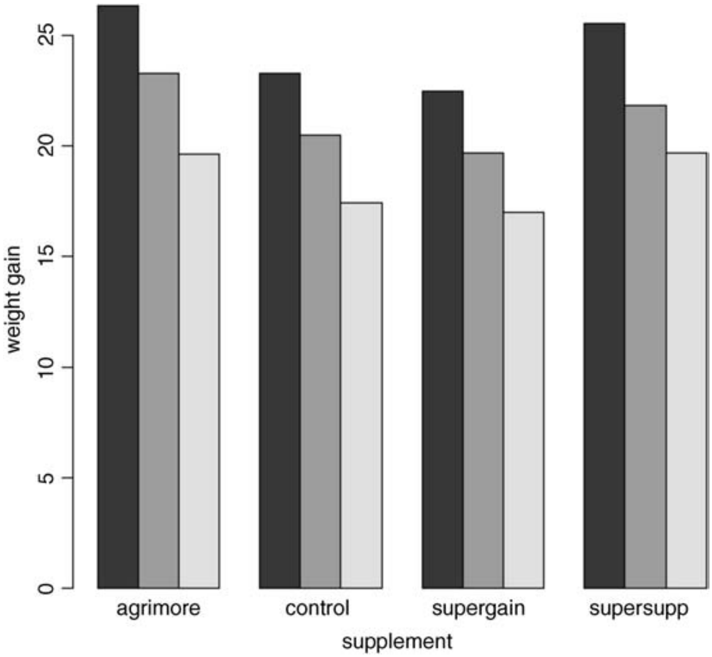
Factorial Experiments

A factorial experiment has two or more factors, each with two or more levels, plus replication for each combination of factor levels. This means that we can investigate statistical interactions, in which *the response to one factor depends on the level of another factor*. Our example comes from a farm-scale trial of animal diets. There are two factors: diet and supplement. Diet is a factor with three levels: barley, oats and wheat. Supplement is a factor with four levels: agrimore, control, supergain and supersupp. The response variable is weight gain after 6 weeks.

```
weights <- read.csv("c:\\temp\\growth.csv")
attach(weights)
```

Data inspection is carried out using `barplot` (note the use of `beside=T` to get the bars in adjacent clusters rather than vertical stacks; we shall add the error bars later):

```
barplot(tapply(gain,list(diet,supplement),mean),beside=T)
```



Note that the second factor in the list (supplement) appears as groups of bars from left to right in alphabetical order by factor level, from ‘agrimore’ to ‘supersupp’. The first factor (diet) appears as three levels within each group of bars: dark = barley, mid = oats, light = wheat, again in alphabetical order by factor level. We should really add a `legend` to explain the levels of diet. We use `levels` to extract the names of the diets to use as labels in the key:

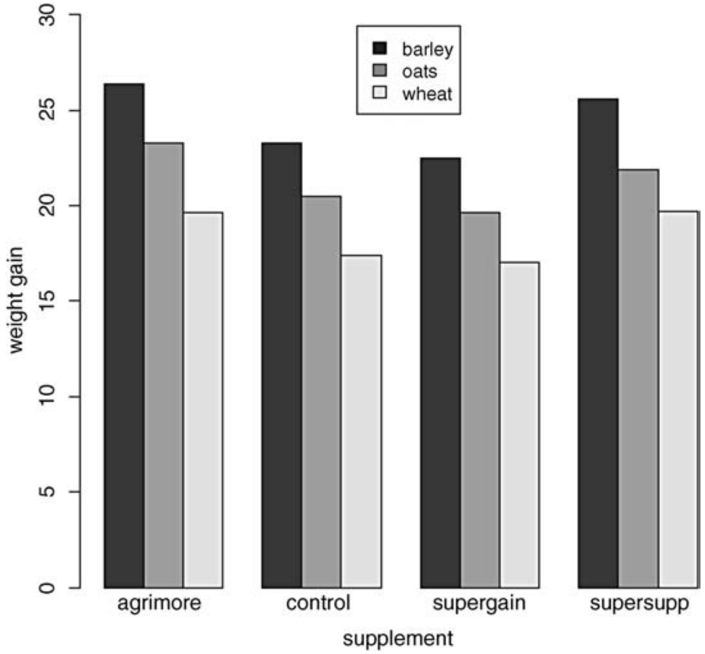
```
labels <- levels(diet)
```

We have used colours so far, but it is easy to use gray scales in R. The continuum goes from 0 = black to 1 = white, so we could use 0.2, 0.6 and 0.9 here:

```
shade <- c(0.2,0.6,0.9)
```

The tricky bit is locating the key on the plot so that it does not overlap any of the bars. You need to understand that R locates the legend on the plot using the coordinates of *the top left-hand corner of the box* that surrounds the key. So you need to find some space below and to the right of a suitable point. Then move the cursor to this point and click. R will draw the legend at that point.

```
barplot(tapply(gain,list(diet,supplement),mean),beside=T,
                 ylab="weight gain",xlab="supplement",ylim=c(0,30))

legend(locator(1),labels,gray(shade))
```



There are clearly substantial differences between the three diets, but the effects of supplement are less obvious. We inspect the mean values using tapply as usual:

```
tapply(gain,list(diet,supplement),mean)

         agrimore  control supergain  supersupp
barley   26.34848 23.29665  22.46612   25.57530
oats     23.29838 20.49366  19.66300   21.86023
wheat    19.63907 17.40552  17.01243   19.66834
```

Now we use aov or lm to fit a factorial ANOVA (the choice affects only whether we get an ANOVA table or a list of parameters estimates as the default output from summary). We estimate parameters for the main effects of each level of diet and each level of supplement, plus terms for the interaction between diet and supplement. Interaction degrees of freedom are the product of the degrees of freedom of the component terms (i.e. $(3-1)\times(4-1)=6$). The model is

```
gain ~ diet + supplement + diet:supplement
```

but this can be simplified using the asterisk notation like this:

```
model <- aov(gain~diet*supplement)
summary(model)
```

```
                 Df  Sum Sq  Mean Sq F value   Pr(>F)
diet              2  287.17   143.59   83.52 3.00e-14 ***
supplement        3   91.88    30.63   17.82 2.95e-07 ***
diet:supplement   6    3.41     0.57    0.33    0.917
Residuals        36   61.89     1.72
```

The ANOVA table shows that there is no hint of any interaction between the two explanatory variables ($p=0.9166$); evidently the effects of diet and supplement are additive but both are highly significant. Now that we know that the error variance $s^2=1.72$ we can add the error bars to the plot. For bars showing standard errors, we need to know the replication for each combination of factor levels:

```
tapply(gain,list(diet,supplement),length)
```

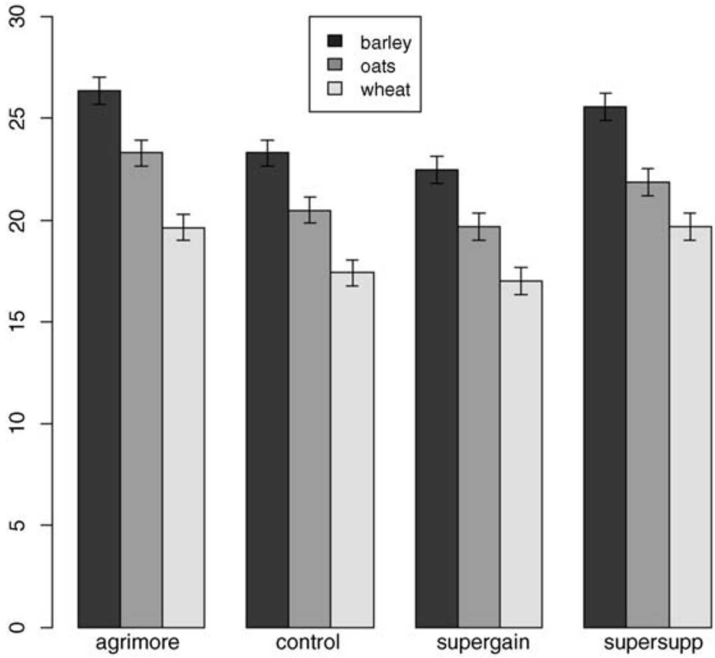
```
        agrimore  control  supergain  supersupp
barley         4        4          4          4
oats           4        4          4          4
wheat          4        4          4          4
```

so the appropriate standard error is $\sqrt{1.72/4}=0.656$. The error.bars function that we wrote earlier will not work for grouped bars, so we need to modify the code like this:

```
x <- as.vector(barplot(tapply(gain,list(diet,supplement),mean),
beside=T,ylim=c(0,30)))
y <- as.vector(tapply(gain,list(diet,supplement),mean))
z <- rep(0.656,length(x))
for( i in 1:length(x))
arrows(x[i],y[i]-z[i],x[i],y[i]+z[i],length=0.05,code=3,angle=90)
```

As usual, we need to make room on the y axis for the top of the highest error bars. Note the use of as.vector to convert the tabular output of tapply into a form suitable for plotting. Don’t forget to add the legend:

```
legend(locator(1),labels,gray(shade))
```



The disadvantage of the ANOVA table is that it does not show us the effect sizes, and does not allow us to work out how many levels of each of the two factors are significantly different. As a preliminary to model simplification, `summary.lm` is often more useful than `summary.aov`:

```
summary.lm(model)

Coefficients:

                           Estimate Std. Error t value Pr(>|t|)
(Intercept)                 26.3485     0.6556  40.191  < 2e-16 ***
dietoats                    -3.0501     0.9271  -3.290 0.002248 **
dietwheat                   -6.7094     0.9271  -7.237 1.61e-08 ***
supplementcontrol           -3.0518     0.9271  -3.292 0.002237 **
supplementsupergain         -3.8824     0.9271  -4.187 0.000174 ***
supplementsupersupp         -0.7732     0.9271  -0.834 0.409816
dietoats:supplementcontrol   0.2471     1.3112   0.188 0.851571
dietwheat:supplementcontrol  0.8183     1.3112   0.624 0.536512
dietoats:supplementsupergain 0.2470     1.3112   0.188 0.851652
dietwheat:supplementsupergain 1.2557    1.3112   0.958 0.344601
dietoats:supplementsupersupp -0.6650    1.3112  -0.507 0.615135
dietwheat:supplementsupersupp 0.8024    1.3112   0.612 0.544381

Residual standard error: 1.311 on 36 degrees of freedom
Multiple R-squared: 0.8607,  Adjusted R-squared: 0.8182
F-statistic: 20.22 on 11 and 36 DF, p-value: 3.295e-12
```

This is a rather complex model, because there are 12 estimated parameters (the number of rows in the table): 6 main effects and 6 interactions. The output re-emphasizes that none of the interaction terms is significant, but it suggests that the minimal adequate model will require five parameters: an intercept, a difference due to oats, a difference due to wheat, a difference due to control and difference due to supergain (these are the five rows with significance stars). This draws attention to the main shortcoming of using treatment contrasts as the default. If you look carefully at the table, you will see that the effect sizes of two of the supplements, control and supergain, are not significantly different from one another. You need lots of practice at doing *t* tests in your head, to be able to do this quickly. Ignoring the signs (because the signs are negative for both of them) we have 3.05 versus 3.88, a difference of 0.83. But look at the associated standard errors (both 0.927); the difference is only about 1 standard error of the difference between two means. For significance, we would need roughly 2 standard errors (remember the rule of thumb, in which $t \geq 2$ is significant; see p. 83). The rows get starred in the significance column because treatment contrasts compare all the main effects in the rows with the intercept (where each factor is set to its first level in the alphabet, namely `agrimore` and `barley` in this case). When, as here, several factor levels are different from the intercept, but not different from one another, they all get significance stars. This means that you cannot count up the number of rows with stars in order to determine the number of significantly different factor levels.

We begin model simplification by leaving out the interaction terms:

```
model <- lm(gain~diet+supplement)
summary(model)
```

```
Coefficients:
                      Estimate  Std. Error  t  value   Pr(>|t|)
(Intercept)            26.1230     0.4408     59.258   < 2e-16 ***
dietoats               -3.0928     0.4408     -7.016  1.38e-08 ***
dietwheat              -5.9903     0.4408    -13.589   < 2e-16 ***
supplementcontrol      -2.6967     0.5090     -5.298  4.03e-06 ***
supplementsupergain    -3.3815     0.5090     -6.643  4.72e-08 ***
supplementsupersupp    -0.7274     0.5090     -1.429      0.16

Residual standard error: 1.247 on 42 degrees of freedom
Multiple R-squared: 0.8531,  Adjusted R-squared: 0.8356
F-statistic: 48.76 on 5 and 42 DF, p-value: < 2.2e-16
```

It is clear that we need to retain all three levels of diet because oats differ from wheat by $5.99 - 3.10 = 2.89$ with a standard error of 0.44 ($t \gg 2$). It is not clear that we need all four levels of supplement, however. Supersupp is not obviously different from the agrimore (−0.727 with standard error 0.509). Nor is supergain obviously different from the unsupplemented control animals ($3.38 - 2.70 = 0.68$). We shall try a new two-level factor to replace the four-level supplement, and see if this significantly reduces the model's explanatory power. Agrimore and supersupp are recoded as 'best' and control and supergain as 'worst':

```
supp2 <- factor(supplement)
levels(supp2)
```

```
[1] "agrimore" "control"  "supergain" "supersupp"
```

```
levels(supp2)[c(1,4)] <- "best"
levels(supp2)[c(2,3)] <- "worst"
levels(supp2)
```

```
[1] "best" "worst"
```

Now we fit the simpler model, then compare the two models:

```
model2 <- lm(gain~diet+supp2)
anova(model,model2)
```

```
Analysis of Variance Table
Model 1:  gain ~ diet + supplement
Model 2:  gain ~ diet + supp2
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     42 65.296
2     44 71.284 -2  -5.9876 1.9257 0.1584
```

The simpler model2 has saved 2 degrees of freedom and is not significantly worse than the more complex model ($p = 0.158$). This is the minimal adequate model: now all of the parameters are significantly different from zero and from one another:

```
summary(model2)
```

```
Coefficients:
             Estimate  Std. Error  t value    Pr(>|t|)
(Intercept)   25.7593      0.3674   70.106    < 2e-16 ***
dietoats      -3.0928      0.4500   -6.873   1.76e-08 ***
dietwheat     -5.9903      0.4500  -13.311    < 2e-16 ***
supp2worst    -2.6754      0.3674   -7.281   4.43e-09 ***

Residual standard error: 1.273 on 44 degrees of freedom
Multiple R-squared: 0.8396,  Adjusted R-squared: 0.8286
F-statistic: 76.76 on 3 and 44 DF, p-value: < 2.2e-16
```

Model simplification has reduced our initial 12-parameter model to a much more tractable four-parameter model that is far easier to communicate to readers. If maximum weight gain is your objective, then a diet of barley and a supplement of either agrimore or supersupp is indicated. Note that the recommendations of the model might be different if profit were the response variable rather than weight gain because we have not taken the costs into account.

Pseudoreplication: Nested Designs and Split Plots

The model-fitting functions aov, lme and lmer have the facility to deal with complicated error structures. Detailed analysis of these topics is beyond the scope of this book (see *The R Book* (Crawley, 2013) for worked examples), but it is important that you can recognize them, and hence avoid the pitfalls of pseudoreplication. There are two general cases:

- nested sampling, as when repeated measurements are taken from the same individual, or observational studies are conducted at several different spatial scales (most or all of the factors are *random effects*)

- split-plot analysis, as when designed experiments are carried out with different treatments applied to plots of different sizes (most of the factors are *fixed effects*)

Split-Plot Experiments

In a split-plot experiment, different treatments are applied to plots of different sizes. Each different plot size is associated with its own error variance, so instead of having one error variance (as in all the ANOVA tables up to this point), we have as many error terms as there are different plot sizes. The analysis is presented as a series of component ANOVA tables, one for each plot size, in a hierarchy from the largest plot size with the lowest replication at the top, down to the smallest plot size with the greatest replication at the bottom.

The example refers to a designed field experiment on crop yield with three treatments: irrigation (with two levels, irrigated and not), sowing density (with three levels, low, medium and high), and fertilizer application (with three levels, low, medium and high):

```
yields <- read.csv("c:\\temp\\splityield.csv")
attach(yields)
names(yields)
```

```
[1] "yield"      "block"      "irrigation"  "density"     "fertilizer"
```

The largest plots were the four whole fields (`block`), each of which was split in half, and irrigation was allocated at random to one half of the field. Each irrigation plot was split into three, and one of three different seed-sowing densities (low, medium or high) was allocated at random (independently for each level of irrigation and each block). Finally, each density plot was divided into three and one of three fertilizer nutrient treatments (N, P, or N and P together) was allocated at random. The model formula is specified as a factorial, using the asterisk notation. The error structure is defined in the `Error()` term, with the plot sizes listed from left to right, from largest to smallest, with each variable separated by the slash operator/. Note that the smallest plot size, fertilizer, does not need to appear in the `Error()` term:

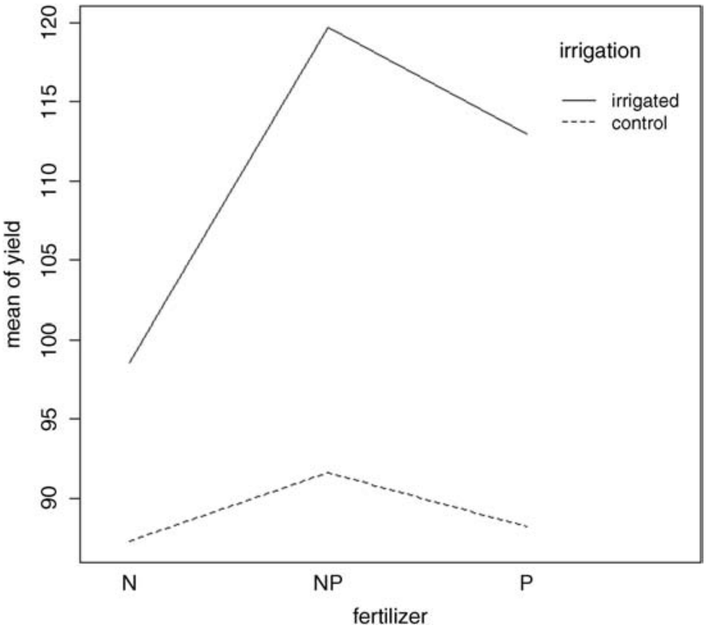
```
model <-
aov(yield~irrigation*density*fertilizer+Error(block/irrigation/
density))
summary(model)
```

```
Error: block
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  3  194.4   64.81

Error: block:irrigation
           Df  Sum Sq  Mean Sq  F value  Pr(>F)
irrigation  1    8278     8278    17.59  0.0247 *
Residuals   3    1412      471

Error: block:irrigation:density
                   Df  Sum Sq  Mean Sq  F value  Pr(>F)
density             2    1758    879.2    3.784  0.0532 .
irrigation:density  2    2747   1373.5    5.912  0.0163 *
Residuals          12    2788    232.3
```

```
Error: Within
                              Df  Sum Sq  Mean Sq  F value    Pr(>F)
fertilizer                     2  1977.4    988.7   11.449  0.000142 ***
irrigation:fertilizer          2   953.4    476.7    5.520  0.008108 **
density:fertilizer             4   304.9     76.2    0.883  0.484053
irrigation:density:fertilizer  4   234.7     58.7    0.680  0.610667
Residuals                     36  3108.8     86.4
```

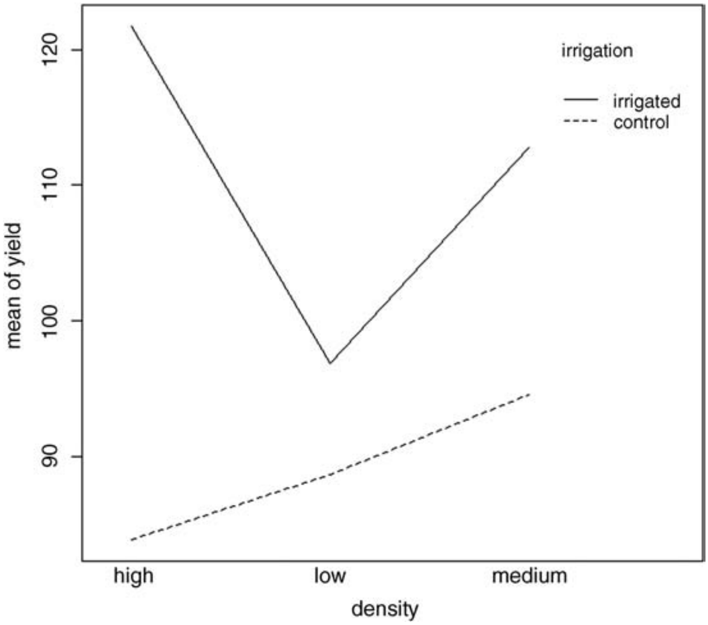
Here you see the four ANOVA tables, one for each plot size: blocks are the biggest plots, half blocks get the irrigation treatment, one third of each half block gets a sowing density treatment, and one third of a sowing density treatment gets each fertilizer treatment. Each plot size has a different error variance. Note that the non-significant main effect for density ($p = 0.053$) does *not* mean that density is unimportant, because density appears in a significant interaction with irrigation (the density terms cancel out when averaged over the two irrigation treatments; see below). The best way to understand the two significant interaction terms is to plot them using `interaction.plot` like this:

```
interaction.plot(fertilizer,irrigation,yield)
```



Irrigation increases yield proportionately more on the P-fertilized plots than on the N-fertilized plots. The `irrigation:density` interaction is more complicated:

```
interaction.plot(density,irrigation,yield)
```



On the irrigated plots, yield is minimal on the low-density plots, but on control plots yield is minimal on the high-density plots.

Random Effects and Nested Designs

Mixed effects models are so called because the explanatory variables are a mixture of fixed effects and random effects:

- fixed effects influence only the *mean* of *y*
- random effects influence only the *variance* of *y*

A random effect should be thought of as coming from a population of effects: the existence of this population is an extra assumption. We speak of *prediction of random effects*, rather than estimation: we *estimate* fixed effects from data, but we intend to make predictions about the population from which our random effects were sampled. Fixed effects are unknown constants to be estimated from the data. Random effects govern the variance–covariance structure of the response variable. The fixed effects are often experimental treatments that were applied under our direction, and the random effects are either categorical or continuous variables that are distinguished by the fact that we are typically not interested in the parameter values, but only in the variance they explain.

One or more of the explanatory variables represents *grouping* in time or in space. Random effects that come from the same group will be correlated, and this contravenes one of the fundamental assumptions of standard statistical models: *independence of errors*. Mixed effects models take care of this non-independence of errors by modelling the covariance structure introduced by the grouping of the data. A major benefit of random effects models is that they economize on the number of degrees of freedom used up by the factor levels. Instead of estimating a mean for every single factor level, the random effects model estimates the distribution of the means (usually as the standard deviation of the differences of the factor-level means around an overall mean). Mixed effects models are particularly useful in cases where there is temporal pseudoreplication (repeated measurements) and/or spatial pseudoreplication (e.g. nested designs or split-plot experiments). These models can allow for:

- spatial autocorrelation between neighbours
- temporal autocorrelation across repeated measures on the same individuals
- differences in the mean response between blocks in a field experiment
- differences between subjects in a medical trial involving repeated measures

The point is that we really do not want to waste precious degrees of freedom in estimating parameters for each of the separate levels of the categorical random effects. On the other hand, we do want to make use of the all measurements we have taken, but because of the pseudoreplication we want to take account of both the:

- correlation structure, used to model within-group correlation associated with temporal and spatial dependencies, using *correlation*
- variance function, used to model non-constant variance in the within-group errors using *weights*

Fixed or Random Effects?

It is difficult without lots of experience to know when to use categorical explanatory variables as fixed effects and when as random effects. Some guidelines are:

- Am I interested in the effect sizes? Yes means fixed effects
- Is it reasonable to suppose that the factor levels come from a population of levels? Yes means random effects
- Are there enough levels of the factor in the data on which to base an estimate of the variance of the population of effects? No means fixed effects
- Are the factor levels informative? Yes means fixed effects
- Are the factor levels just numeric labels? Yes means random effects
- Am I mostly interested in making inferences about the distribution of effects, based on the random sample of effects represented in the dataframe? Yes means random effects.
- Is there hierarchical structure? Yes means you need to ask whether the data are experimental or observations
- Is it a hierarchical experiment, where the factor levels are experimental manipulations? Yes means fixed effects in a split-plot design (see p. 173)

- Is it a hierarchical observational study? Yes means random effects, perhaps in a variance components analysis (see p. 183)
- When your model contains both fixed and random effects, use mixed effects models
- If your model structure is linear, use linear mixed effects, `lmer`
- Otherwise, specify the model equation and use non-linear mixed effects, `nlme`

Removing the Pseudoreplication

The extreme response to pseudoreplication in a data set is simply to eliminate it. Spatial pseudoreplication can be averaged away and temporal pseudoreplication can be dealt with by carrying out carrying out separate ANOVAs, one for each time period. This approach has two major weaknesses:

- it cannot address questions about treatment effects that relate to the longitudinal development of the mean response profiles (e.g. differences in growth rates between successive times)
- inferences made with each of the separate analyses are not independent, and it is not always clear how they should be combined

Analysis of Longitudinal Data

The key feature of longitudinal data is that the same individuals are measured repeatedly through time. This would represent temporal pseudoreplication if the data were used uncritically in regression or ANOVA. The set of observations on one individual subject will tend to be positively correlated, and this correlation needs to be taken into account in carrying out the analysis. The alternative is a cross-sectional study, with all the data gathered at a single point in time, in which each individual contributes a single data point. The advantage of longitudinal studies is that they are capable of separating *age effects* from *cohort effects*; these are inextricably confounded in cross-sectional studies. This is particularly important when differences between years mean that cohorts originating at different times experience different conditions, so that individuals of the same age in different cohorts would be expected to differ. There are two extreme cases in longitudinal studies:

- a few measurements on a large number of individuals
- a large number of measurements on a few individuals

In the first case it is difficult to fit an accurate model for change within individuals, but treatment effects are likely to be tested effectively. In the second case, it is possible to get an accurate model of the way individuals change though time, but there is less power for testing the significance of treatment effects, especially if variation from individual to individual is large. In the first case, less attention will be paid to estimating the correlation structure, while in the second case the covariance model will be the principal focus of attention. The aims are:

- to estimate the average time course of a process
- to characterize the degree of heterogeneity from individual to individual in the rate of the process
- identify the factors associated with both of these, including possible cohort effects

The response is not the individual measurement, but the *sequence of measurements* on an individual subject. This enables us to distinguish between age effects and year effects (see Diggle, Liang and Zeger, 1994, for details).

Derived Variable Analysis

The idea here is to get rid of the pseudoreplication by reducing the repeated measures into a set of summary statistics (slopes, intercepts or means), and then *analyse these summary statistics* using standard parametric techniques such as ANOVA or regression. The technique is weak when the values of the explanatory variables change through time. Derived variable analysis makes most sense when it is based on the parameters of scientifically interpretable non-linear models from each time sequence. However, the best model from a theoretical perspective may not be the best model from the statistical point of view.

There are three qualitatively different sources of random variation:

- *random effects*: experimental units differ (e.g. genotype, history, size, physiological condition) so that there are intrinsically high responders and other low responders
- *serial correlation*: there may be time-varying stochastic variation within a unit (e.g. market forces, physiology, ecological succession, immunity) so that correlation depends on the time separation of pairs of measurements on the same individual, with correlation weakening with the passage of time
- *measurement error*: the assay technique may introduce an element of correlation (e.g. shared bioassay of closely spaced samples; different assay of later specimens)

Dealing with Pseudoreplication

For random effects we are often more interested in the question of how much of the variation in the response variable can be attributed to a given factor than we are in estimating means or assessing the significance of differences between means. This procedure is called variance components analysis.

```
rats <- read.csv("c:\\temp\\rats.csv")
attach(rats)
names(rats)
```

```
[1] "Glycogen" "Treatment" "Rat"    "Liver"
```

This classic example of pseudoreplication comes from Snedecor and Cochran (1980). Three experimental treatments were administered to rats, and the glycogen contents of the rats' livers were analysed as the response variable. This was the set-up: there were two rats per treatment, so the total sample was $n = 3 \times 2 = 6$. The tricky bit was this: after each rat was killed, its liver was cut up into three pieces: a left-hand bit, a central bit and a right-hand bit. So now there are six rats each producing three bits of liver, for a total of $6 \times 3 = 18$ numbers. Finally, two separate preparations were made from each macerated bit of liver, to assess the measurement error associated with the analytical machinery. At this point there are $2 \times 18 = 36$ numbers in the dataframe as a whole. The factor levels are numbers, so we need to declare the explanatory variables to be categorical before we begin:

```
Treatment <- factor(Treatment)
Rat <- factor(Rat)
Liver <- factor(Liver)
```

Here is the analysis done the *wrong* way:

```
model <- aov(Glycogen~Treatment)
summary(model)
```

```
           Df  Sum Sq  Mean  Sq  F  value    Pr(>F)
Treatment   2   1558    778.8     14.5  3.03e-05 ***
Residuals  33   1773     53.7
```

Treatment has a highly significant effect on liver glycogen content ($p = 0.00003$). Wrong! We have committed a classic error of pseudoreplication. Look at the error line in the ANOVA table: it says the residuals have 33 degrees of freedom. But there were only six rats in the whole experiment, so the error degrees of freedom must be $6 - 1 - 2 = 3$ (not 33)!

Here is the analysis of variance done properly, averaging away the pseudoreplication. First reduce the dataframe from 36 numbers to just six numbers: one mean value per rat:

```
yv <- tapply(Glycogen,list(Treatment,Rat),mean)
yv
```

```
          1         2
1  132.5000  148.5000
2  149.6667  152.3333
3  134.3333  136.0000
```

Turn this object into a vector to act as our new response variable:

```
( yv <- as.vector(yv))
```

```
[1]  132.5000  149.6667  134.3333  148.5000  152.3333  136.0000
```

From the `tapply` output you can see that the data appear columnwise. The columns (1 and 2) are the replicate rats, and the rows (1, 2 and 3) are the treatments (control, supplement and supplement plus sugar). We need to produce a new factor of length 6 to contain the `treatment` in the correct order, 1, 2, 3 then 1, 2, 3 (make sure you understand this):

```
treatment <- factor(c(1,2,3,1,2,3))
```

Now we can fit the non-pseudoreplicated model:

```
model <- aov(yv~treatment)
summary(model)

            Df  Sum Sq  Mean Sq  F  value Pr(>F)
treatment    2   259.6   129.80    2.929 0.197
Residuals    3   132.9    44.31
```

As you can see, the error degrees of freedom are correct (d.f. = 3, not 33), and the interpretation is completely different: there are no significant differences in liver glycogen under the three experimental treatments ($p = 0.197$).

There are two different ways of doing the analysis properly in R: ANOVA with multiple error terms (`aov`) or linear mixed effects models (`lmer`). The problem is that the bits of the same liver are pseudoreplicates because they are spatially correlated (they come from the same rat); they are not independent, as required if they are to be true replicates. Likewise, the two preparations from each liver bit are very highly correlated (the livers were macerated before the preparations were taken, so they are essentially the same sample, and certainly not independent replicates of the experimental treatments).

Here is the correct analysis using `aov` with multiple error terms. In the error term we start with the largest scale (treatment), then rats within treatments, then liver bits within rats within treatments. Finally, there were replicated measurements (two preparations) made for each bit of liver (see Box 8.2):

```
model2 <- aov(Glycogen~Treatment+Error(Treatment/Rat/Liver))
summary(model2)

Error: Treatment
           Df   Sum Sq  Mean  Sq
Treatment   2     1558     778.8

Error: Treatment:Rat
           Df   Sum  Sq  Mean  Sq  F value Pr(>F)
Residuals   3    797.7     265.9

Error: Treatment:Rat:Liver
          Df   Sum Sq  Mean  Sq  F value Pr(>F)
Residuals 12     594      49.5

Error: Within
          Df    Sum  Sq  Mean  Sq  F  value Pr(>F)
Residuals 18      381    21.17
```

You can do the correct, non-pseudoreplicated analysis of variance from this output. The F test involves the treatment variance (778.8) divided by *the error variance at the spatial scale immediately below* (i.e. rats within treatments 265.9). This means that the test statistic $F = 778.8/265.9 = 2.928921$ is not significant (compare with the wrong analysis on p. 180).

Note that the current version of R does not print the F values or p values. The critical value of F would need to be

```
qf(0.95,2,3)
[1] 9.552094
```

Box 8.2. Sums of squares in hierarchical designs

The trick to understanding these sums of squares is to appreciate that with nested categorical explanatory variables (random effects) the correction factor, which is subtracted from the sum of squared subtotals, is *not* the conventional $\left(\sum y\right)^2/kn$. Instead, the correction factor is the uncorrected sum of squared subtotals from the level in the hierarchy immediately above the level in question. This is very hard to see without lots of practice. The total sum of squares, *SSY*, and the treatment sum of squares, *SSA*, are computed in the usual way (see Box 8.1):

$$SSY = \sum y^2 - \frac{\left(\sum y\right)^2}{n}$$

$$SSA = \frac{\sum_{i=1}^{k} C_i^2}{n} - \frac{\left(\sum y\right)^2}{kn}$$

The analysis is easiest to understand in the context of an example. For the rats data, the treatment totals were based on 12 numbers (2 rats, 3 liver bits per rat and 2 preparations per liver bit). In this case, in the formula for *SSA*, above, has $n = 12$ and $kn = 36$. We need to calculate sums of squares for rats within treatments, SS_{Rats}, liver bits within rats within treatments, $SS_{\text{Liverbits}}$, and preparations within liver bits within rats within treatments, $SS_{\text{Preparations}}$:

$$SS_{\text{Rats}} = \frac{\sum R^2}{6} - \frac{\sum C^2}{12}$$

$$SS_{\text{Liverbits}} = \frac{\sum L^2}{2} - \frac{\sum R^2}{6}$$

$$SS_{\text{Preparations}} = \frac{\sum y^2}{1} - \frac{\sum L^2}{2}$$

The correction factor at any level is the *uncorrected sum of squares from the level above*. The last sum of squares could have been computed by difference:

$$SS_{\text{Preparations}} = SSY - SSA - SS_{\text{Rats}} - SS_{\text{Liverbits}}$$

Variance Components Analysis (VCA)

To turn this nested ANOVA table into a variance components analysis we need to do a little work. The thing to understand is that each row of the table includes the variability from the level below plus the new variability introduced at that level. So at the bottom, the variance of 21.17 represents measurement error (differences between readings produced by similar samples put through the same machine). The next level up reflects heterogeneity within individual rats' livers (e.g. if the middle of the liver was different in glycogen content than the left- or right-hand ends of the liver; physiological differences, if you like). The next level up reflects differences between rats; these will include gender effects, genetic effects, nutritional effects and such like (although we would hope that these had been controlled in the experimental design).

We are interested in discovering the variance entering the system at each level in the hierarchy. To find this out we begin by calculating the difference in variance between adjacent levels:

Level 2 versus level 1: $49.5 - 21.17 = 28.33$ reflecting liver bit difference

Level 3 versus level 2: $265.9 - 49.5 = 216.4$ reflecting rat differences

We stop at this point, because the next level up is a fixed effect (treatment) not a random effect (like rat, or liver bit within rat).

The next step is to divide these differences by the number of pseudoreplicates in the lower of the two levels (2 preparations per liver bit; 6 preparations per rat, 2 from each of 3 liver bits). So now we have:

Residuals = preparations within liver bits: unchanged = 21.17

Liver bits within rats within treatments: $(49.5 - 21.17)/2 = 14.165$

Rats within treatments: $(265.89 - 49.5)/6 = 36.065$

These numbers are the *variance components*. You can see straight away that most variation enters this experiment as a result of differences between individual rats in their mean liver glycogen contents (36.065). Least variation is introduced by cutting each liver into three pieces (14.165).

The results of a variance components analysis are traditionally expressed as percentages. We add the three components together to do this:

```
vc <- c(21.17,14.165,36.065)
100*vc/sum(vc)
```

```
[1] 29.64986 19.83894 50.51120
```

That's all there is to it. An analysis like this can be very useful in planning future experiments. Since more than 50% of the variance is attributable to differences between the rats, then increasing the number of rats will have the biggest positive effects on the power of the next experiment (you might also consider controlling the rats more carefully by using the same age and same gender of the same genotype, for instance). There is no real point in splitting up each liver: it makes for three times the work and explains less than 20% of the variance.

References

Crawley, M.J. (2013) *The R Book*, 2nd edn, John Wiley & Sons, Chichester.

Diggle, P.J., Liang, K.-Y. and Zeger, S.L. (1994) *Analysis of Longitudinal Data*, Clarendon Press, Oxford.

Snedecor, G.W. and Cochran, W.G. (1980) *Statistical Methods*, Ames, Iowa State University Press.

Further Reading

Pinheiro, J.C. and Bates, D.M. (2000) *Mixed-Effects Models in S and S-PLUS*, Springer-Verlag, New York.

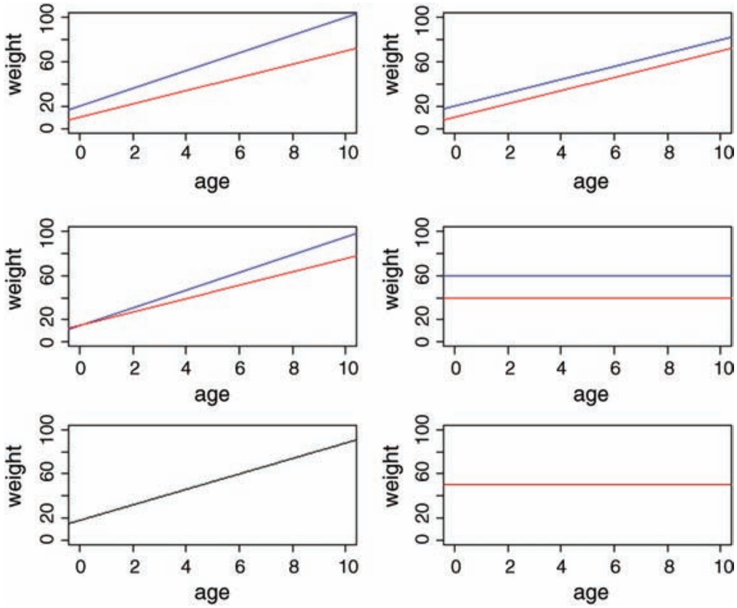
9

Analysis of Covariance

Analysis of covariance (ANCOVA) involves a combination of regression and analysis of variance. The response variable is continuous, and there is at least one continuous explanatory variable and at least one categorical explanatory variable. Typically, the maximal model involves estimating a slope and an intercept (the regression part of the exercise) for each level of the categorical variable(s) (the ANOVA part of the exercise). Let us take a concrete example. Suppose we are modelling weight (the response variable) as a function of sex and age. Sex is a factor with two levels (male and female) and age is a continuous variable. The maximal model therefore has four parameters: two slopes (a slope for males and a slope for females) and two intercepts (one for males and one for females) like this:

$$weight_{\text{male}} = a_{\text{male}} + b_{\text{male}} \times age$$
$$weight_{\text{female}} = a_{\text{female}} + b_{\text{female}} \times age$$

Model simplification is an essential part of ANCOVA, because the principle of parsimony requires that we keep as few parameters in the model as possible.



Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.
© 2015 John Wiley & Sons, Ltd. Published 2015 by John Wiley & Sons, Ltd.

There are at least six possible models in this case, and the process of model simplification begins by asking whether we need all four parameters (top left). Perhaps we could make do with two intercepts and a common slope (top right). Or a common intercept and two different slopes (centre left). There again, age may have no significant effect on the response, so we only need two parameters to describe the main effects of sex on weight; this would show up as two separated, horizontal lines in the plot (one mean weight for each sex; centre right). Alternatively, there may be no effect of sex at all, in which case we only need two parameters (one slope and one intercept) to describe the effect of age on weight (bottom left). In the limit, neither the continuous nor the categorical explanatory variables might have any significant effect on the response, in which case model simplification will lead to the one-parameter null model $\hat{y} = \bar{y}$ (a single, horizontal line; bottom right).

Decisions about model simplification are based on the explanatory power of the model: if the simpler model does not explain significantly less of the variation in the response, then the simpler model is preferred. Tests of explanatory power are carried out using `anova` or `AIC` to compare two models: when using `anova` we shall only retain the more complicated model if the p value from comparing the two models is less than 0.05. When using `AIC` we simply prefer the model with the lower value.

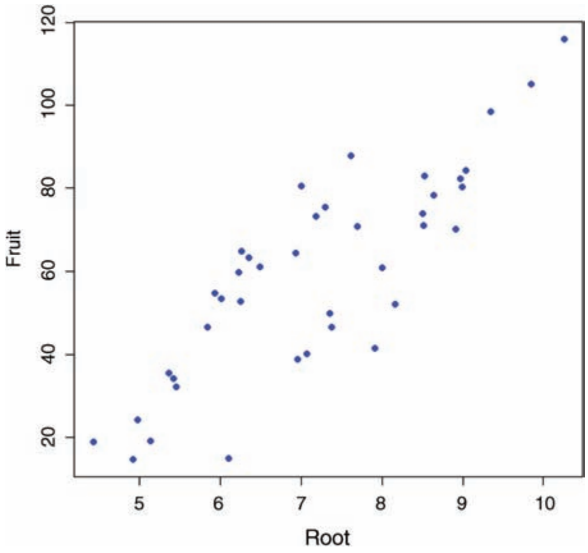
Let us see how this all works by investigating a realistic example. The dataframe concerns an experiment on a plant's ability to regrow and produce seeds following grazing. The initial, pre-grazing size of the plant is recorded as the diameter of the top of its rootstock. Grazing is a two-level factor: grazed or ungrazed (protected by fences). The response is the weight of seeds produced per plant at the end of the growing season. Our expectation is that big plants will produce more seeds than small plants and that grazed plants will produce fewer seeds than ungrazed plants. Let us see what actually happened:

```
compensation <- read.csv("c:\\temp\\ipomopsis.csv")
attach(compensation)
names(compensation)
```

```
[1] "Root"  "Fruit" "Grazing"
```

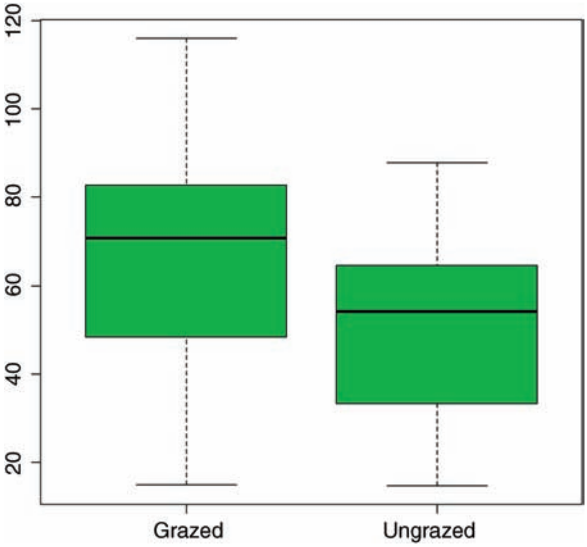
We begin with data inspection. First: did initial plant size matter?

```
plot(Root,Fruit,pch=16,col="blue")
```



Yes, indeed. Bigger plants produced more seeds at the end of the growing season. What about grazing?

```
plot(Grazing,Fruit,col="lightgreen")
```



This is not at all what we expected to see. Apparently, the grazed plants produced *more* seeds, not less than the ungrazed plants. Taken at face value, the effect looks to be statistically significant ($p < 0.03$):

```
summary(aov(Fruit~Grazing))

            Df  Sum Sq  Mean Sq  F value  Pr(>F)
Grazing      1    2910   2910.4    5.309  0.0268 *
Residuals   38   20833    548.2
```

We shall return to this after we have carried out the statistical modelling properly.

Analysis of covariance is done in the familiar way: it is just that the explanatory variables are a mixture of continuous and categorical variables. We start by fitting the most complicated model, with different slopes and intercepts for the grazed and ungrazed plants. For this, we use the asterisk operator:

```
model <- lm(Fruit~Root*Grazing)
```

An important thing to realize about analysis of covariance is that *order matters*. Look at the regression sum of squares in the ANOVA table when we fit root first:

```
summary.aov(model)

              Df  Sum Sq  Mean Sq   F value     Pr(>F)
Root           1   16795    16795   359.968   < 2e-16 ***
Grazing        1    5264     5264   112.832  1.21e-12 ***
```

```
Root:Grazing    1      5       5     0.103      0.75
Residuals      36   1680      47
```

and when we fit root second:

```
model <- lm(Fruit~Grazing*Root)
summary.aov(model)
```

```
Grazing         1   2910   2910    62.380   2.26e-09 ***
Root            1  19149  19149   410.420    < 2e-16 ***
Grazing:Root    1      5      5     0.103       0.75
Residuals      36   1680     47
```

In both cases, the error sum of squares (1680) and the interaction sum of squares (5) are the same, but the regression sum of squares (labelled `Root`) is much greater when root is fitted to the model after grazing (19,149), than when it is fitted first (16,795). This is because the data for ANCOVA are typically non-orthogonal. Remember, *with non-orthogonal data, order matters* (Box 9.1).

Box 9.1. Corrected sums of squares in analysis of covariance

The total sum of squares, SSY, and the treatment sums of squares, SSA, are calculated in the same way as in a straightforward analysis of variance (Box 8.1). The sums of squares for the separate regressions within the individual factor levels, i, are calculated as shown in Box 7.3: $SSXY_i$, SSX_i, SSR_i, and SSE_i are then added up across the factor levels:

$$SSXY_{\text{total}} = \sum SSXY_i$$

$$SSX_{\text{total}} = \sum SSX_i$$

$$SSR_{\text{total}} = \sum SSR_i$$

Then the overall regression sum of squares, SSR, is calculated from the total corrected sums of products and the total corrected sums of squares of x:

$$SSR = \frac{(SSXY_{\text{total}})^2}{SSX_{\text{total}}}$$

The difference in the two estimates, SSR and SSR_{total}, is called SSR_{diff} and is a measure of the significance of the differences between the regression slopes. Now we can compute SSE by difference:

$$SSE = SSY - SSA - SSR - SSR_{\text{diff}}$$

But *SSE* is *defined* for the *k* levels in which the regressions were computed as

$$SSE = \sum_{i=1}^{k} \sum (y - a_i - b_i x)^2$$

Of course, both methods give the same answer.

Back to the analysis. The interaction, SSR_{diff}, representing differences in slope between the grazed and ungrazed treatments, appears to be insignificant, so we remove it:

```
model2 <- lm(Fruit~Grazing+Root)
```

Notice the use of + rather than * in the model formula. This says 'fit different intercepts for grazed and ungrazed plants, but fit the same slope to both graphs'. Does this simpler model have significantly lower explanatory power? We use `anova` to find out:

```
anova(model,model2)
Analysis of Variance Table

Model 1: Fruit ~ Grazing * Root
Model 2: Fruit ~ Grazing + Root
  Res.Df    RSS  Df Sum of Sq      F Pr(>F)
1     36 1679.7
2     37 1684.5  -1   -4.8122 0.1031  0.75
```

The simpler model does not have significantly lower explanatory power ($p = 0.75$), so we adopt it. Note that we did not have to do the `anova` in this case: the *p* value given in the `summary.aov(model)` table gave the correct, deletion *p* value. Here are the parameter estimates from our minimal adequate model:

```
summary.lm(model2)
Coefficients:
                 Estimate Std. Error  t value  Pr (>|t|)
(Intercept)      -127.829      9.664   -13.23  1.35e-15 ***
GrazingUngrazed    36.103      3.357    10.75  6.11e-13 ***
Root               23.560      1.149    20.51   < 2e-16 ***

Residual standard error: 6.747 on 37 degrees of freedom
Multiple R-squared: 0.9291,  Adjusted R-squared: 0.9252
F-statistic: 242.3 on 2 and 37 DF, p-value: < 2.2e-16
```

The model has high explanatory power, accounting for more than 90% of the variation in seed production (multiple r^2). The hard thing about ANCOVA is understanding what the parameter estimates mean. Starting at the top, the first row, as labelled, contains an intercept. It is the intercept for the graph of seed production against initial rootstock size for the grazing treatment *whose factor level comes first in the alphabet*. To see which one this is, we can use `levels`:

```
levels(Grazing)
```

```
[1] "Grazed"  "Ungrazed"
```

So the intercept is for the grazed plants. The second row, labelled `GrazingUngrazed`, is *a difference between two intercepts*. To get the intercept for the ungrazed plants, we need to add 36.103 to the intercept for the grazed plants ($-127.829 + 36.103 = -91.726$). The third row, labelled `Root`, is *a slope*: it is the gradient of the graph of seed production against initial rootstock size, and it is the same for both grazed and ungrazed plants. If there had been a significant interaction term, this would have appeared in row 4 as *a difference between two slopes*.

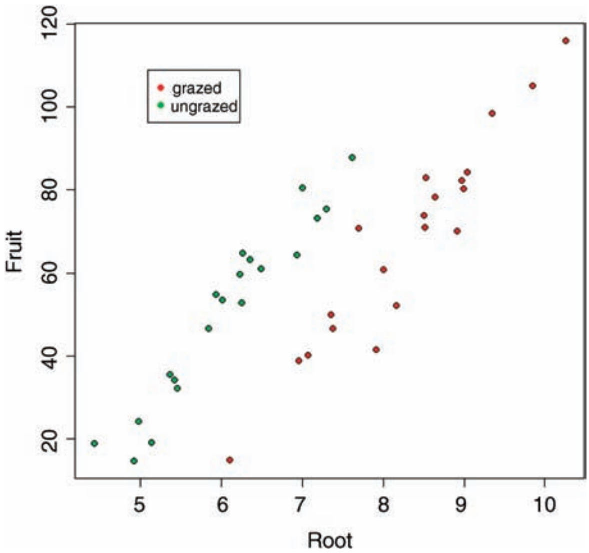
We can now plot the fitted model through the scatterplot. It will be useful to have different colours for the grazed (red) and ungrazed plants (green)

```
plot(Root,Fruit,pch=21,bg=(1+as.numeric(Grazing)))
```

Note the use of `1+as.numeric(Grazing)` to produce the different colours: 2 (red) for Grazed and 3 (green) for ungrazed. Let us add a legend to make this clear:

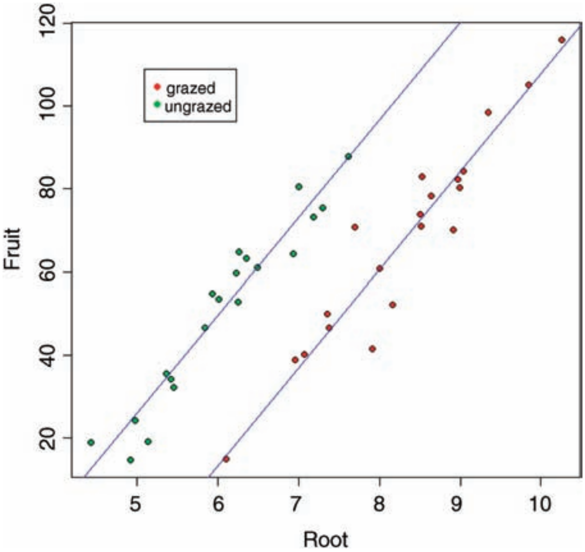
```
legend(locator(1),c("grazed","ungrazed"),col=c(2,3),pch=16)
```

Just position the cursor where you want the top left-hand corner of the key to appear, then click:



Now it becomes clear why we got the curious result at the beginning, in which grazing appeared to increase seed production. Clearly what has happened is that the majority of big plants ended up in the grazed treatment (red symbols). If you compare like with like (e.g. plants at 7 mm initial root diameter), it is clear that the ungrazed plants (green symbols) produced more seed than the grazed plants (36.103 more, to be precise). This will become clearer when we fit the lines predicted by model2:

```
abline(-127.829,23.56,col="blue")
abline(-127.829+36.103,23.56,col="blue")
```



This example shows the great strength of analysis of covariance. By controlling for initial plant size, we have completely reversed the interpretation. The naïve first impression was that grazing increased seed production:

```
tapply(Fruit,Grazing,mean)
```

```
  Grazed   Ungrazed
 67.9405   50.8805
```

and this was significant if we were rash enough to fit grazing on its own ($p = 0.0268$ as we saw earlier). But when we do the correct ANCOVA, we find the opposite result: grazing significantly *reduces* seed production for plants of comparable initial size; for example from 77.46 to 41.36 at mean rootstock size:

```
-127.829+36.103+23.56*mean(Root)
[1] 77.4619
-127.829+23.56*mean(Root)
[1] 41.35889
```

The moral is clear. When you have covariates (like initial size in this example), then use them. This can do no harm, because if the covariates are not significant, they will drop out during model simplification. Also remember that in ANCOVA, *order matters*. So always start model simplification by removing the highest-order interaction terms first. In ANCOVA, these interaction terms are *differences between slopes* for different factor levels (recall that in multi-way ANOVA, the interaction terms were differences between means). Other ANCOVAs are described in Chapters 13, 14 and 15 in the context of count data, proportion data and binary response variables.

Further Reading

Huitema, B.E. (1980) *The Analysis of Covariance and Alternatives*, John Wiley & Sons, New York.

10

Multiple Regression

In multiple regression we have a continuous response variable and two or more continuous explanatory variables (i.e. there are no categorical explanatory variables). In many applications, multiple regression is the most difficult of all the statistical models to do well. There are several things that make multiple regression so challenging:

- the studies are often observational (rather than controlled experiments)
- we often have a great many explanatory variables
- we often have rather few data points
- missing combinations of explanatory variables are commonplace

There are several important statistical issues, too:

- the explanatory variables are often correlated with one another (non-orthogonal)
- there are major issues about which explanatory variables to include
- there could be curvature in the response to the explanatory variables
- there might be interactions between explanatory variables
- the last three issues all tend to lead to parameter proliferation

There is a temptation to become personally attached to a particular model. Statisticians call this 'falling in love with your model'. It is as well to remember the following truths about models:

- all models are wrong
- some models are better than others
- the correct model can never be known with certainty
- the simpler the model, the better it is

Fitting models to data is the central function of R. The process is essentially one of exploration; there are no fixed rules and no absolutes. The object is to determine a minimal adequate model from the large set of potential models that might be used to describe the given set of data. In this book we discuss five types of model:

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

- the null model
- the minimal adequate model
- the current model
- the maximal model
- the saturated model

The stepwise progression from the saturated model (or the maximal model, whichever is appropriate) through a series of simplifications to the minimal adequate model is made on the basis of *deletion tests*; these are *F* tests, AIC, *t* tests or chi-squared tests that assess the significance of the increase in deviance that results when a given term is removed from the current model.

Models are representations of reality that should be both accurate and convenient. However, it is impossible to maximize a model's realism, generality and holism simultaneously, and the principle of parsimony (Occam's razor; see p. 8) is a vital tool in helping to choose one model over another. Thus, we would only include an explanatory variable in a model if it significantly improved the fit of a model. Just because we went to the trouble of measuring something, that does not mean we have to have it in our model. Parsimony says that, other things being equal, we prefer:

- a model with $n - 1$ parameters to a model with n parameters
- a model with $k - 1$ explanatory variables to a model with k explanatory variables
- a linear model to a model which is curved
- a model without a hump to a model with a hump
- a model without interactions to a model containing interactions between variables

Other considerations include a preference for models containing explanatory variables that are easy to measure over variables that are difficult or expensive to measure. Also, we prefer models that are based on a sound mechanistic understanding of the process over purely empirical functions.

Parsimony requires that the model should be as simple as possible. This means that the model should not contain any redundant parameters or factor levels. We achieve this by fitting a maximal model then simplifying it by following one or more of these steps:

- remove non-significant interaction terms
- remove non-significant quadratic or other non-linear terms
- remove non-significant explanatory variables
- group together factor levels that do not differ from one another
- in ANCOVA, set non-significant slopes of continuous explanatory variables to zero

subject, of course, to the caveats that the simplifications make good scientific sense, and do not lead to significant reductions in explanatory power.

Just as there is no perfect model, so there may be no optimal scale of measurement for a model. Suppose, for example, we had a process that had Poisson errors with multiplicative effects amongst the explanatory variables. Then, one must chose between three different scales, each of which optimizes one of three different properties:

1. the scale of $\sqrt{y}$ would give constancy of variance
2. the scale of $y^{2/3}$ would give approximately normal errors
3. the scale of $\ln(y)$ would give additivity

Thus, any measurement scale is always going to be a compromise, and you should choose the scale that gives the best overall performance of the model.

Model	Interpretation
Saturated model	One parameter for every data point Fit: perfect Degrees of freedom: none Explanatory power of the model: none
Maximal model	Contains all (p) factors, interactions and covariates that might be of any interest. Many of the model's terms are likely to be insignificant Degrees of freedom: $n-p-1$ Explanatory power of the model: it depends
Minimal adequate model	A simplified model with $0 \le p' \le p$ parameters Fit: less than the maximal model, but not significantly so Degrees of freedom: $n-p'-1$ Explanatory power of the model: $r^2 = SSR/SSY$
Null model	Just one parameter, the overall mean $\bar{y}$ Fit: none; $SSE = SSY$ Degrees of freedom: $n-1$ Explanatory power of the model: none

The Steps Involved in Model Simplification

There are no hard and fast rules, but the procedure laid out below works well in practice. With large numbers of explanatory variables, and many interactions and non-linear terms, the process of model simplification can take a very long time. But this is time well spent because it reduces the risk of overlooking an important aspect of the data. It is important to realize that *there is no guaranteed way of finding all the important structures in a complex dataframe.*

Step	Procedure	Explanation
1	Fit the maximal model	Fit all the factors, interactions and covariates of interest. Note the residual deviance. If you are using Poisson or binomial errors, check for overdispersion and rescale if necessary

(*continued*)

(*Continued*)

Step	Procedure	Explanation
2	Begin model simplification	Inspect the parameter estimates using `summary`. Remove the least significant terms first, using `update -`, starting with the highest-order interactions
3	If the deletion causes an insignificant increase in deviance	Leave that term out of the model. Inspect the parameter values again. Remove the least significant term remaining
4	If the deletion causes a significant increase in deviance	Put the term back in the model using `update +`. These are the statistically significant terms as assessed by deletion from the maximal model
5	Keep removing terms from the model	Repeat steps 3 or 4 until the model contains nothing but significant terms. This is the minimal adequate model. If none of the parameters is significant, then the minimal adequate model is the null model

Caveats

Model simplification is an important process, but it should not be taken to extremes. For example, the interpretation of deviances and standard errors produced with fixed parameters that have been estimated from the data should be undertaken with caution. Again, the search for 'nice numbers' should not be pursued uncritically. Sometimes there are good scientific reasons for using a particular number (e.g. a power of 0.66 in an allometric relationship between respiration and body mass). Again, it is much more straightforward, for example, to say that yield increases by 2 kg per hectare for every extra unit of fertilizer, than to say that it increases by 1.947 kg. Similarly, it may be preferable to say that the odds of infection increase 10-fold under a given treatment, rather than to say that the logits increase by 2.321 (without model simplification this is equivalent to saying that there is a 10.186-fold increase in the odds). It would be absurd, however, to fix on an estimate of 6 rather than 6.1 just because 6 is a whole number.

Order of Deletion

Remember that when explanatory variables are correlated (as they almost always are in multiple regression), *order matters*. If your explanatory variables are correlated with each other, then the significance you attach to a given explanatory variable will depend upon whether you delete it from a maximal model or add it to the null model. If you always test by model simplification then you won't fall into this trap.

The fact that you have laboured long and hard to include a particular experimental treatment does not justify the retention of that factor in the model if the analysis shows it to have no explanatory power. ANOVA tables are often published containing a mixture of significant and non-significant effects. This is not a problem in orthogonal designs, because sums of squares can be unequivocally attributed to each factor and interaction term. But as soon as there are missing values or unequal weights, then it is impossible to tell how the parameter estimates and standard errors of the significant terms would have been altered if the non-significant terms had been deleted. The best practice is this

- say whether your data are orthogonal or not
- present a minimal adequate model
- give a list of the non-significant terms that were omitted, and the deviance changes that resulted from their deletion

Readers can then judge for themselves the relative magnitude of the non-significant factors, and the importance of correlations between the explanatory variables.

The temptation to retain terms in the model that are 'close to significance' should be resisted. The best way to proceed is this. If a result would have been *important* if it had been statistically significant, then it is worth repeating the experiment with higher replication and/or more efficient blocking, in order to demonstrate the importance of the factor in a convincing and statistically acceptable way.

Carrying Out a Multiple Regression

For multiple regression, the approach recommended here is that before you begin modelling in earnest you do two things:

- use tree models to investigate whether there are complicated interactions
- use generalized additive models to investigate curvature

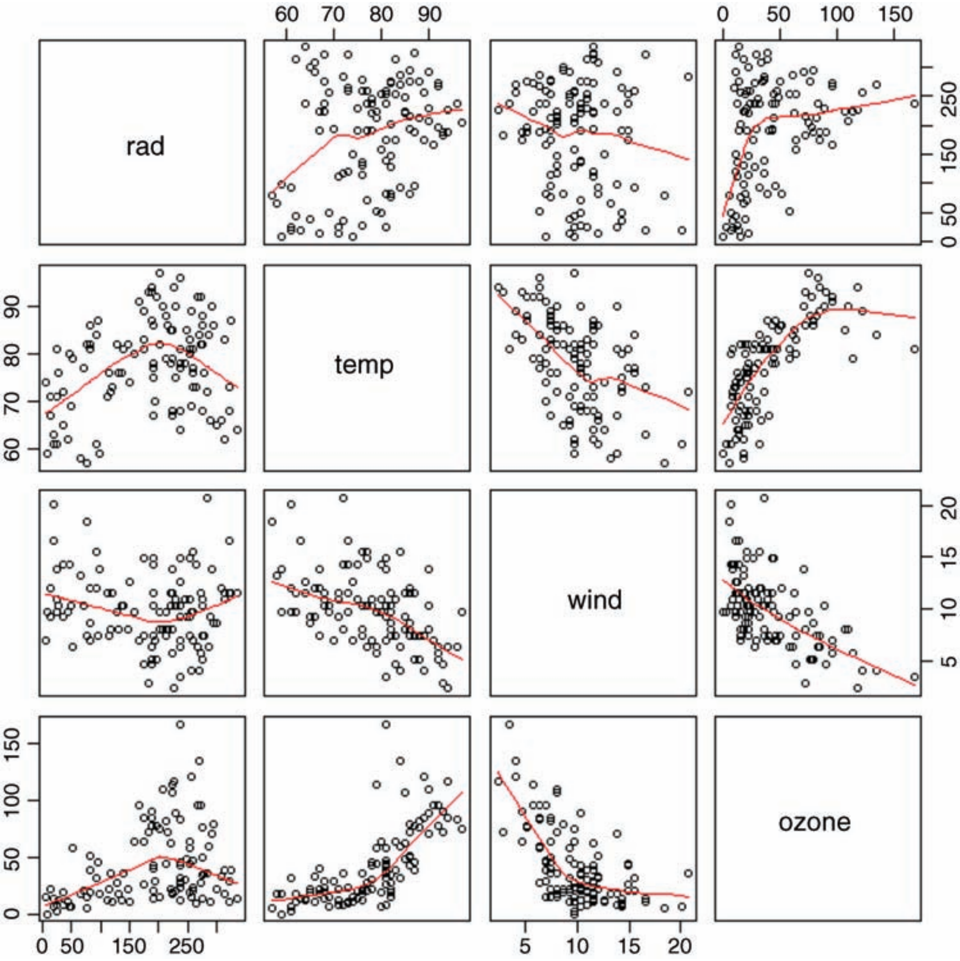
Let us begin with an example from air pollution studies. How is ozone concentration related to wind speed, air temperature and the intensity of solar radiation?

```
ozone.pollution <- read.csv("c:\\temp\\ozone.data.csv")
attach(ozone.pollution)
names(ozone.pollution)
```

```
[1] "rad"  "temp" "wind" "ozone"
```

In multiple regression, it is always a good idea to use the `pairs` function to look at all the correlations:

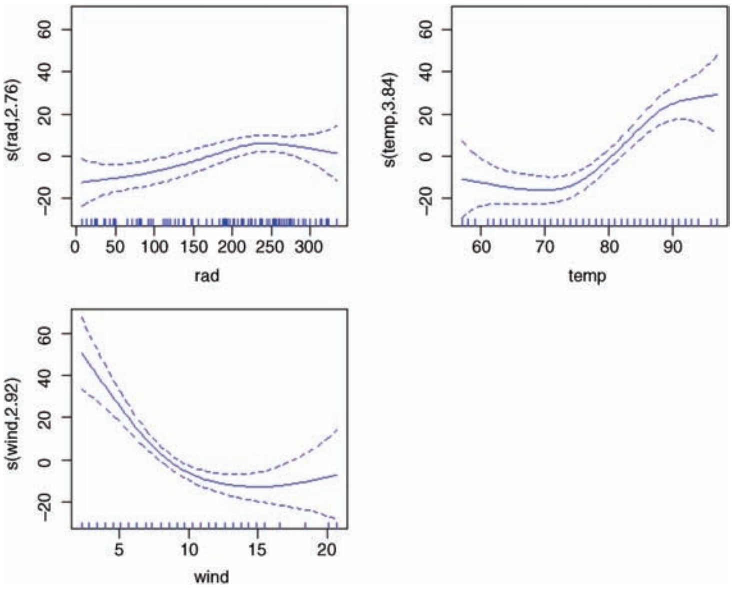
```
pairs(ozone.pollution,panel=panel.smooth)
```



The response variable, ozone concentration, is shown on the *y* axis of the bottom row of panels: there is a strong negative relationship with wind speed, a positive correlation with temperature and a rather unclear, but possibly humped relationship with radiation.

A good way to start a multiple regression problem is using non-parametric smoothers in a generalized additive model (gam) like this:

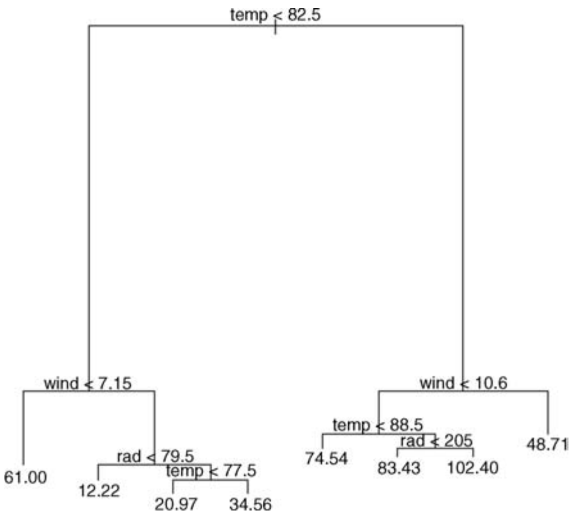
```
library(mgcv)
par(mfrow=c(2,2))
model <- gam(ozone~s(rad)+s(temp)+s(wind))
plot(model,col= "blue")
```



The confidence intervals are sufficiently narrow to suggest that the curvature in all three relationships might be real.

The next step might be to fit a tree model to see whether complex interactions between the explanatory variables are indicated. The older `tree` function is better for this graphical search for interactions than is the more modern `rpart` (but the latter is much better for statistical modelling):

```
par(mfrow=c(1,1))
library(tree)
model<-tree(ozone~.,data=ozone.pollution)
plot(model)
text(model)
```



This shows that temperature is far and away the most important factor affecting ozone concentration (the longer the branches in the tree, the greater the deviance explained). Wind speed is important at both high and low temperatures, with still air being associated with higher mean ozone levels (the figures at the ends of the branches are mean ozone concentrations). Radiation shows an interesting, but subtle effect. At low temperatures, radiation matters at relatively high wind speeds (>7.15), whereas at high temperatures, radiation matters at relatively low wind speeds (<10.6); in both cases, however, higher radiation is associated with higher mean ozone concentration. The tree model therefore indicates that the interaction structure of the data is not particularly complex (this is a reassuring finding).

Armed with this background information (likely curvature of responses and a relatively uncomplicated interaction structure), we can begin the linear modelling. We start with the most complicated model: this includes interactions between all three explanatory variables plus quadratic terms to test for curvature in response to each of the three explanatory variables:

```
model1 <- lm(ozone~temp*wind*rad+I(rad^2)+I(temp^2)+I(wind^2))
summary(model1)
```

```
Coefficients:
                Estimate  Std. Error  t value  Pr (>|t|)
(Intercept)    5.683e+02  2.073e+02    2.741   0.00725 **
temp          -1.076e+01  4.303e+00   -2.501   0.01401 *
wind          -3.237e+01  1.173e+01   -2.760   0.00687 **
rad           -3.117e-01  5.585e-01   -0.558   0.57799
I(rad^2)      -3.619e-04  2.573e-04   -1.407   0.16265
I(temp^2)      5.833e-02  2.396e-02    2.435   0.01668 *
I(wind^2)      6.106e-01  1.469e-01    4.157  6.81e-05 ***
temp:wind      2.377e-01  1.367e-01    1.739   0.08519 .
temp:rad       8.403e-03  7.512e-03    1.119   0.26602
wind:rad       2.054e-02  4.892e-02    0.420   0.67552
temp:wind:rad -4.324e-04  6.595e-04   -0.656   0.51358

Residual standard error: 17.82 on 100 degrees of freedom
Multiple R-squared: 0.7394,  Adjusted R-squared: 0.7133
F-statistic: 28.37 on 10 and 100 DF, p-value: < 2.2e-16
```

The three-way interaction is clearly not significant, so we remove it to begin the process of model simplification:

```
model2 <- update(model1,~. – temp:wind:rad)
summary(model2)
```

Next, we remove the least significant two-way interaction term, in this case `wind:rad`:

```
model3 <- update(model2,~. - wind:rad)
summary(model3)
```

Then we try removing the temperature by wind interaction:

```
model4 <- update(model3,~. - temp:wind)
summary(model4)
```

We shall retain the marginally significant interaction between `temp` and `rad` ($p = 0.04578$) for the time being, but leave out all other interactions. In `model4`, the least significant quadratic term is for `rad`, so we delete this:

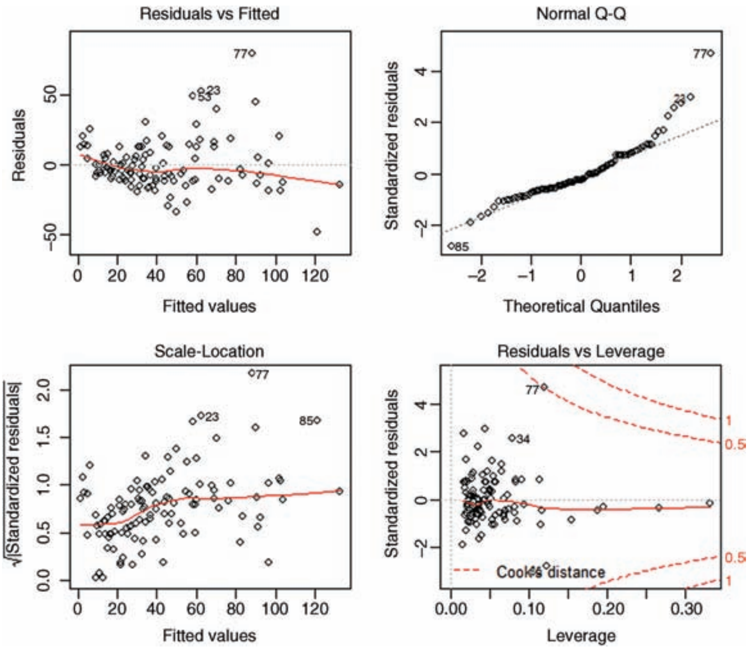
```
model5 <- update(model4,~. - I(rad^2))
summary(model5)
```

This deletion has rendered the `temp:rad` interaction insignificant, and caused the main effect of radiation to become insignificant. We should try removing the `temp:rad` interaction:

```
model6 <- update(model5,~. - temp:rad)
summary(model6)
```

```
Coefficients:
               Estimate   Std. Error t value  Pr (>|t|)
(Intercept) 291.16758   100.87723    2.886   0.00473 **
temp          -6.33955     2.71627   -2.334   0.02150 *
wind         -13.39674     2.29623   -5.834  6.05e-08 ***
rad            0.06586     0.02005    3.285   0.00139 **
I(temp^2)      0.05102     0.01774    2.876   0.00488 **
I(wind^2)      0.46464     0.10060    4.619   1.10e-05 ***

Residual  standard error: 18.25  on  105  degrees of  freedom
Multiple R-Squared: 0.713,      Adjusted R-squared: 0.6994
F-statistic:  52.18  on 5  and  105 DF,  p-value:      0
```

Now we are making progress. All the terms in `model6` are significant. At this stage, we should check the assumptions, using `plot(model6)`:



There is a clear pattern of variance increasing with the mean of the fitted values. This is bad news (heteroscedasticity). Also, the normality plot is distinctly curved; again, this is bad news. Let us try transformation of the response variable. There are no zeros in the response, so a log transformation is worth trying. We need to repeat the entire process of model simplification from the very beginning, because transformation alters the variance structure and the linearity of all the relationships.

```
model7 <- lm(log(ozone)~temp*wind*rad+I(rad^2)+I(temp^2)+I(wind^2))
```

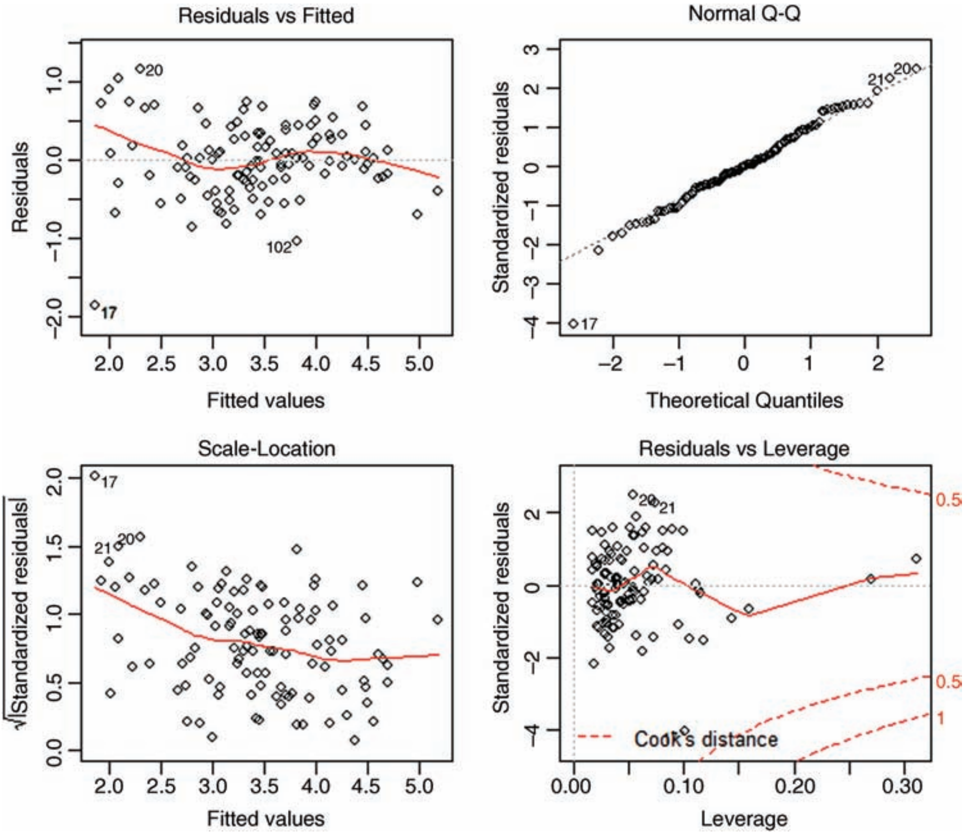
We can speed up the model simplification using the `step` function:

```
model8 <- step(model7)
summary(model8)
```

```
Coefficients:
             Estimate  Std. Error  t value Pr(>|t|)
(Intercept) 7.724e-01  6.350e-01   1.216 0.226543
temp        4.193e-02  6.237e-03   6.723 9.52e-10 ***
wind       -2.211e-01  5.874e-02  -3.765 0.000275 ***
rad         7.466e-03  2.323e-03   3.215 0.001736 **
I(rad^2)   -1.470e-05  6.734e-06  -2.183 0.031246 *
I(wind^2)   7.390e-03  2.585e-03   2.859 0.005126 **

Residual standard error: 0.4851 on 105 degrees of freedom
Multiple R-squared:   0.7004,     Adjusted R-squared:   0.6861
F-statistic:  49.1 on 5 and 105 DF,  p-value:  < 2.2e-16
```

This simplified model following transformation has a different structure than before: all three main effects are significant and there are still no interactions between variables, but the quadratic term for temperature has gone and a quadratic term for radiation remains in the model. We need to check that transformation has improved the problems that we had with non-normality and non-constant variance:

```
plot(model8)
```



This shows that the variance and normality are now reasonably well behaved, so we can stop at this point. We have found the minimal adequate model. Our initial impressions from the tree model (no interactions) and the generalized additive model (a good deal of curvature) have been borne out by the statistical modelling.

A Trickier Example

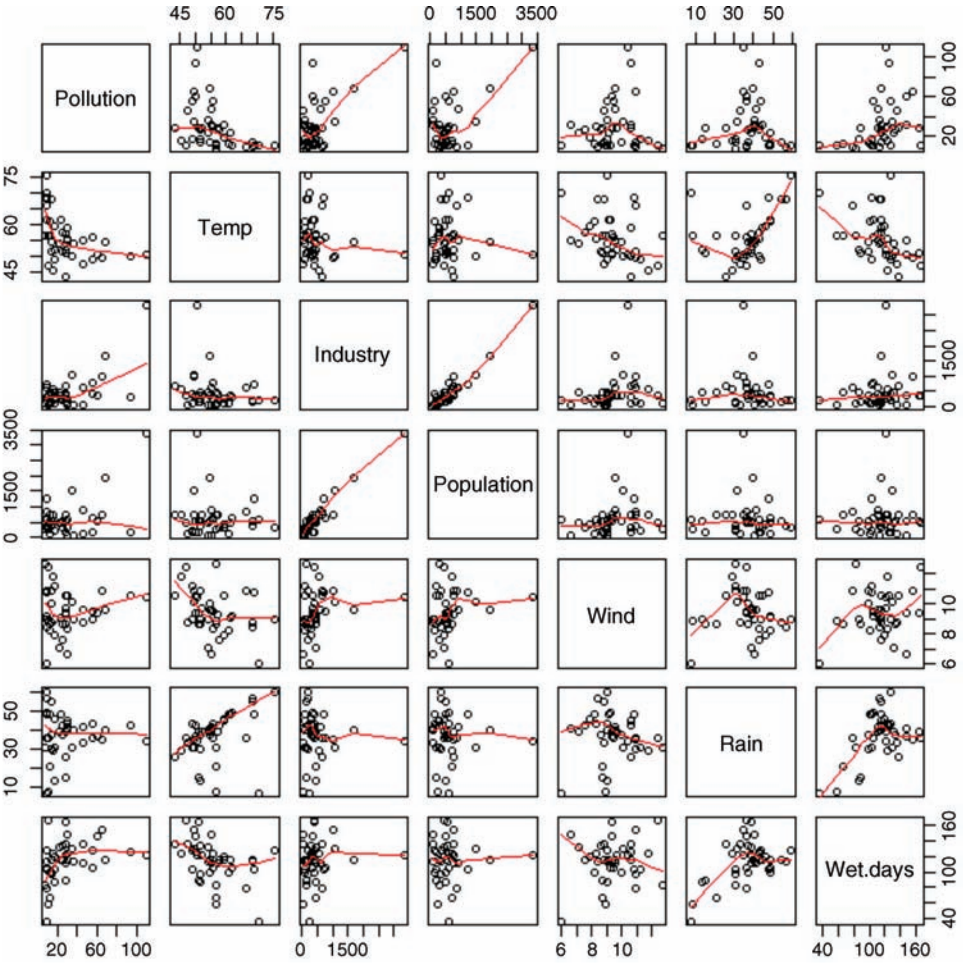
In this example we introduce two new but highly realistic difficulties: more explanatory variables, and fewer data points. This is another air pollution dataframe, but the response variable this time is sulphur dioxide concentration. There are six continuous explanatory variables:

```
pollute <- read.csv("c:\\temp\\sulphur.dioxide.csv")
attach(pollute)
names(pollute)
```

```
[1] "Pollution" "Temp"    "Industry"  "Population" "Wind"
[6] "Rain"    "Wet.days"
```

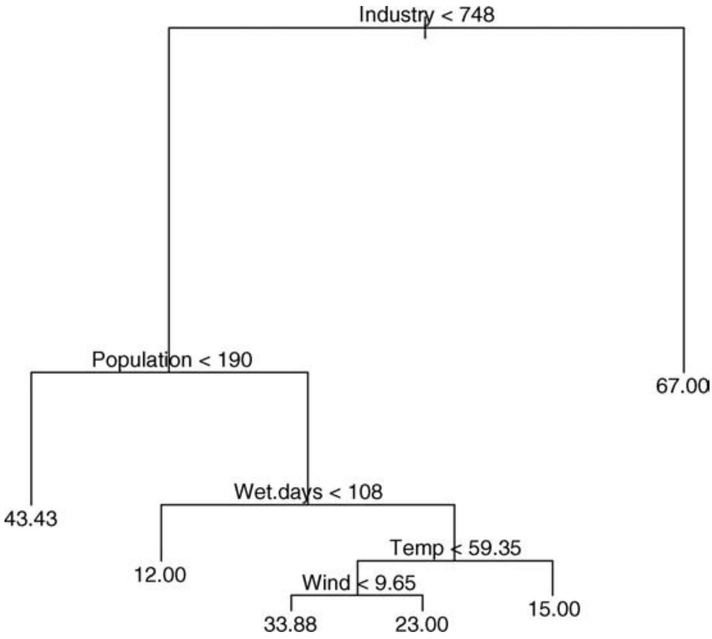
Here are the 36 paired scatterplots:

```
pairs(pollute,panel=panel.smooth)
```



This time, let us begin with the tree model rather than the generalized additive model.

```
par(mfrow=c(1,1))
library(tree)
model <- tree(Pollution~.,data=pollute)
plot(model)
text(model)
```



This tree model is much more complicated than we saw in the previous ozone example. It is interpreted as follows. The most important explanatory variable is industry, and the threshold value separating low and high values of industry is 748. The right hand branch of the tree indicates the mean value of air pollution for high levels of industry (67.00). The fact that this limb is unbranched means that no other variables explain a significant amount of the variation in pollution levels for high values of industry. The left-hand limb does not show the mean values of pollution for low values of industry, because there are other significant explanatory variables. Mean values of pollution are only shown at the extreme ends of branches. For low values of industry, the tree shows us that population has a significant impact on air pollution. At low values of population (< 190) the mean level of air pollution was 43.43. For high values of population, the number of wet days is significant. Low numbers of wet days (< 108) have mean pollution levels of 12.00, while temperature has a significant impact on pollution for places where the number of wet days is large. At high temperatures (> 59.35 °F) the mean pollution level was 15.00, while at lower temperatures the run of wind is important. For still air (wind < 9.65) pollution was higher (33.88) than for higher wind speeds (23.00).

The virtues of tree-based models are numerous:

- they are easy to appreciate and to describe to other people
- the most important variables stand out
- interactions are clearly displayed
- non-linear effects are captured effectively
- the complexity of the behaviour of the explanatory variables is plain to see

We conclude that the interaction structure is highly complex. We shall need to carry out the linear modelling with considerable care.

Start with some elementary calculations. With six explanatory variables, how many interactions might we fit? Well, there are $5+4+3+2+1=15$ two-way interactions for a start. Plus 20 three-way, 15 four-way and 6 five-way interactions, and 1 six-way interaction for good luck. Then there are quadratic terms for each of the six explanatory variables. So we are looking at about 70 parameters that might be estimated from the data with a complete maximal model. But how many data points do we have?

```
length(Pollution)
```

```
[1] 41
```

Oops! We are planning to estimate almost twice as many parameters as there are data points. That's taking over-parameterization to new heights. This leads us to a very general and extremely important question: *how many data points does one need per parameter to be estimated?*

Of course there can be no hard and fast answer to this, but perhaps there are some useful rules of thumb? To put the question another way, given that I have 41 data points, how many parameters is it reasonable to expect that I shall be able to estimate from the data?

As an absolute minimum, one would need at least three data points per parameter (remember that any two points can always be joined perfectly by a straight line). Applying this to our current example means that the absolute maximum we should try to estimate would be $41/3=13$ parameters (one for the intercept plus 12 slopes). A more conservative rule of thumb might suggest 10 data points per parameter, under which more stringent criteria we would be allowed to estimate an intercept and just three slopes.

It is at this stage that the reality bites. If I cannot fit all of the explanatory variables I want, then how do I choose which ones to fit? This is a problem because:

- main effects can appear to be non-significant when a relationship is strongly curved (see p. 148) so if I don't fit a quadratic term (using up an extra parameter in the process) I shan't be confident about leaving a variable out of the model
- main effects can appear to be non-significant when interactions between two or more variables are pronounced (see p. 174)
- interactions can only be investigated when variables appear together in the same model (interactions between two continuous explanatory variables are typically included in the model as a function of the product of the two variables, and both main effects should be included when we do this)

The ideal strategy is to fit a maximal model containing all the explanatory variables, with curvature terms for each variable, along with all possible interactions terms, and then simplify this maximal model, perhaps using `step` to speed up the initial stages. In our present example, however, this is completely out of the question. We have far too many variables and far too few data points.

In this example, it is impossible to fit all combinations of variables simultaneously and hence it is highly likely that important interaction terms could be overlooked. We know from the tree model that the interaction structure is going to be complicated, so we need to concentrate on that. Perhaps a good place to start is by looking for curvature, to see if we can eliminate this as a major cause of variation. Fitting all the variables and their quadratic terms requires an intercept and 12 slopes to be estimated, and this is right at the extreme limit of three data points per parameter:

```
model1 <- lm(Pollution~Temp+I(Temp^2)+Industry+I(Industry^2)+
Population+I(Population^2)+Wind+I(Wind^2)+Rain+I(Rain^2)+Wet.
days+I(Wet.days^2))
summary(model1)
```

```
Coefficients:
                  Estimate  Std. Error  t value  Pr (>|t|)
(Intercept)     -6.641e+01   2.234e+02   -0.297   0.76844
Temp             5.814e-01   6.295e+00    0.092   0.92708
I(Temp^2)       -1.297e-02   5.188e-02   -0.250   0.80445
Industry         8.123e-02   2.868e-02    2.832   0.00847 **
I(Industry^2)   -1.969e-05   1.899e-05   -1.037   0.30862
Population      -7.844e-02   3.573e-02   -2.195   0.03662 *
I(Population^2)  2.551e-05   2.158e-05    1.182   0.24714
Wind             3.172e+01   2.067e+01    1.535   0.13606
I(Wind^2)       -1.784e+00   1.078e+00   -1.655   0.10912
Rain             1.155e+00   1.636e+00    0.706   0.48575
I(Rain^2)       -9.714e-03   2.538e-02   -0.383   0.70476
Wet.days        -1.048e+00   1.049e+00   -0.999   0.32615
I(Wet.days^2)    4.555e-03   3.996e-03    1.140   0.26398

Residual standard error: 14.98 on 28 degrees of freedom
Multiple R-squared: 0.7148,  Adjusted R-squared: 0.5925
F-statistic: 5.848 on 12 and 28 DF, p-value: 5.868e-05
```

So that's our first bit of good news. In this unsimplified model, there is no evidence of curvature for any of the six explanatory variables. Only the main effects of industry and population are significant in this (highly over-parameterized) model. Let us see what `step` makes of this:

```
model2 <- step(model1)
summary(model2)
```

```
Coefficients:
              Estimate Std. Error t value   Pr (>|t|)
(Intercept)  54.468341  14.448336   3.770   0.000604   ***
I(Temp^2)    -0.009525   0.003395  -2.805   0.008150   **
Industry      0.065719   0.015246   4.310   0.000126   ***
Population   -0.040189   0.014635  -2.746   0.009457   **
I(Wind^2)    -0.165965   0.089946  -1.845   0.073488   .
Rain          0.405113   0.211787   1.913   0.063980   .

Residual standard error: 14.25 on 35 degrees of freedom
Multiple R-squared: 0.6773,  Adjusted R-squared: 0.6312
F-statistic: 14.69 on 5 and 35 DF, p-value: 8.951e-08
```

The simpler model shows significant curvature of temperature and marginally significant curvature for wind. Notice that `step` has removed the linear terms for `Temp` and `Wind`; normally when we have a quadratic term we retain the linear term even if its slope is not significantly different from zero. Let us remove the non-significant terms from `model2` to see what happens:

```
model3 <- update(model2, ~.- Rain-I(Wind^2))
summary(model3)
Coefficients:
              Estimate Std. Error t value  Pr (>|t|)
(Intercept)  42.068701  9.993087   4.210  0.000157   ***
I(Temp^2)    -0.005234  0.003100  -1.688  0.099752   .
Industry      0.071489  0.015871   4.504  6.45e-05   ***
Population   -0.046880  0.015199  -3.084  0.003846   **

Residual standard error: 15.08 on 37 degrees of freedom
Multiple R-squared: 0.6183,  Adjusted R-squared: 0.5874
F-statistic: 19.98 on 3 and 37 DF, p-value: 7.209e-08
```

Simplification reduces the significance of the quadratic term for temperature. We shall need to return to this. The model supports out interpretation of the initial tree model: a main effect for industry is very important, as is a main effect for population.

Now we need to consider the interaction terms. We shall not fit interaction terms without both the component main effects, so we cannot fit all the two-way interaction terms at the same time (that would be $15+6=21$ parameters; well above the rule of thumb maximum value of 13). One approach is to fit the interaction terms in randomly selected sets. With all six main effects, we can afford to assess $13-6=7$ interaction terms at a time. Let us try this. Make a vector containing the names of the 15 two-way interactions:

```
interactions <- c("ti","tp","tw","tr","td","ip","iw",
"ir","id","pw","pr","pd","wr","wd","rd")
```

Now shuffle the interactions into random order using `sample` without replacement:

```
sample(interactions)

[1] "wr" "wd" "id" "ir" "rd" "pr" "tp" "pw" "ti"
[10]"iw" "tw" "pd" "tr" "td" "ip"
```

It would be pragmatic to test the two-way interactions in three models each containing all the main effects, plus two-way interaction terms five-at-a-time:

```
model4 <- lm(Pollution~Temp+Industry+Population+Wind+Rain+Wet.
days+Wind:Rain+Wind:Wet.days+Industry:Wet.days+Industry:Rain+Rain:
Wet.days)
model5 <- lm(Pollution~Temp+Industry+Population+Wind+Rain+Wet.
days+Population:Rain+Temp:Population+Population:Wind+Temp:
Industry+Industry:Wind)
```

```
model6 <- lm(Pollution~Temp+Industry+Population+Wind+Rain+Wet.
days+Temp:Wind+Population:Wet.days+Temp:Rain+Temp:Wet.
days+Industry:Population)
```

Extracting only the interaction terms from the three models, we see:

```
Industry:Rain        -1.616e-04  9.207e-04  -0.176   0.861891
Industry:Wet.days     2.311e-04  3.680e-04   0.628   0.534949
Wind:Rain             9.049e-01  2.383e-01   3.798   0.000690   ***
Wind:Wet.days        -1.662e-01  5.991e-02  -2.774   0.009593   **
Rain:Wet.days         1.814e-02  1.293e-02   1.403   0.171318

Temp:Industry        -1.643e-04  3.208e-03  -0.051   0.9595
Temp:Population       1.125e-03  2.382e-03   0.472   0.6402
Industry:Wind         2.668e-02  1.697e-02   1.572   0.1267
Population:Wind      -2.753e-02  1.333e-02  -2.066   0.0479      *
Population:Rain       6.898e-04  1.063e-03   0.649   0.5214

Temp:Wind             1.261e-01  2.848e-01   0.443   0.66117
Temp:Rain            -7.819e-02  4.126e-02  -1.895   0.06811     .
Temp:Wet.days         1.934e-02  2.522e-02   0.767   0.44949
Industry:Population   1.441e-06  4.178e-06   0.345   0.73277
Population:Wet.days   1.979e-05  4.674e-04   0.042   0.96652
```

The next step might be to put all of the significant or close-to-significant interactions into the same model, and see which survive:

```
model7 <- lm(Pollution~Temp+Industry+Population+Wind+Rain+Wet.
days+Wind:Rain+Wind:Wet.days+Population:Wind+Temp:Rain)
summary(model7)
Coefficients:
                       Estimate  Std.Error  t value   Pr (>|t|)
(Intercept)          323.054546 151.458618    2.133   0.041226   *
Temp                  -2.792238   1.481312   -1.885   0.069153   .
Industry               0.073744   0.013646    5.404   7.44e-06   ***
Population             0.008314   0.056406    0.147   0.883810
Wind                 -19.447031   8.670820   -2.243   0.032450   *
Rain                  -9.162020   3.381100   -2.710   0.011022   *
Wet.days               1.290201   0.561599    2.297   0.028750   *
Temp:Rain              0.017644   0.027311    0.646   0.523171
Population:Wind       -0.005684   0.005845   -0.972   0.338660
Wind:Rain              0.997374   0.258447    3.859   0.000562   ***
Wind:Wet.days         -0.140606   0.053582   -2.624   0.013530   *
```

We certainly don't need `Temp:Rain`

```
model8 <- update(model7,~.-Temp:Rain)
summary(model8)
```

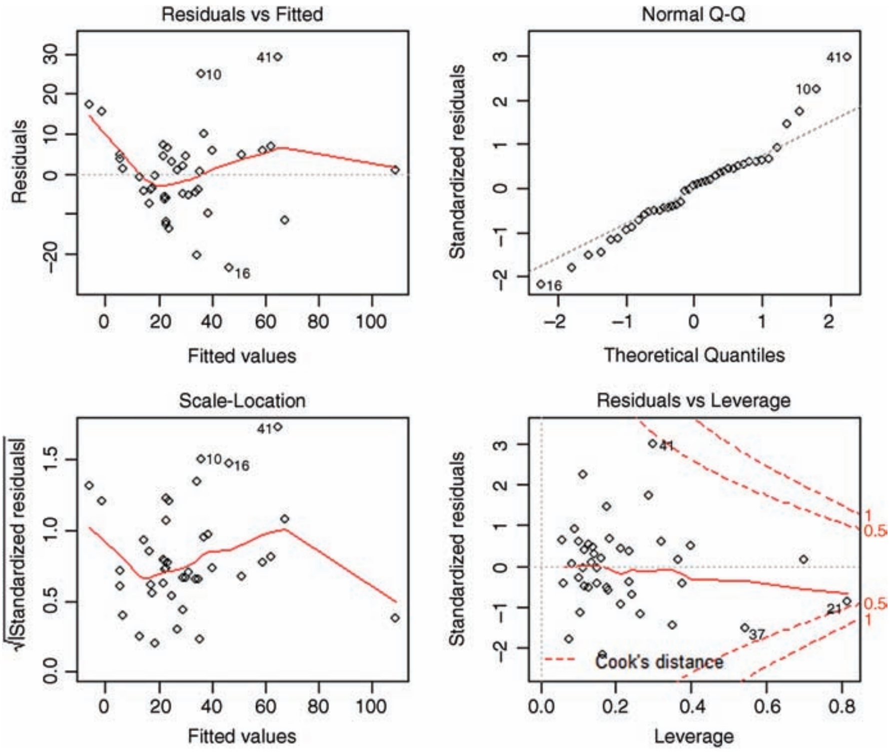
or `Population:Wind`

```
model9 <- update(model8,~.-Population:Wind)
summary(model9)
```

```
Coefficients:
                     Estimate  Std. Error  t  value   Pr (>|t|)
(Intercept)         290.12137    71.14345    4.078   0.000281   ***
Temp                 -2.04741     0.55359   -3.698   0.000811   ***
Industry              0.06926     0.01268    5.461   5.19e-06   ***
Population           -0.04525     0.01221   -3.707   0.000793   ***
Wind                -20.17138     5.61123   -3.595   0.001076   **
Rain                 -7.48116     1.84412   -4.057   0.000299   ***
Wet.days              1.17593     0.54137    2.172   0.037363   *
Wind:Rain             0.92518     0.20739    4.461   9.44e-05    ***
Wind:Wet.days        -0.12925     0.05200   -2.486   0.018346   *

Residual standard error: 11.75 on 32 degrees of freedom
Multiple R-squared: 0.7996,   Adjusted R-squared: 0.7495
F-statistic: 15.96 on 8 and 32 DF,  p-value: 3.51e-09All the terms in
model7 are significant.  Time for a check on the behaviour of the model:
```

There are two significant two-way interactions, `Wind:Rain` and `Wind:Wet.days`. It is time to check the assumptions:

```
plot(model9)
```



The variance is OK, but there is a strong indication of non-normality of errors. But what about the higher-order interactions? One way to proceed is to specify the interaction level using ^3 in the model formula, but if we do this, we'll run out of degrees of freedom straight away. A sensible option is to fit three-way terms for the variables that already appear in two-way interactions: in our case, that is just one term, `Wind:Rain:Wet.days`:

```
model10 <- update(model9,~. + Wind:Rain:Wet.days)
summary(model10)
```

```
Coefficients:
                      Estimate  Std. Error  t value   Pr (>|t|)
(Intercept)         278.464474   68.041497    4.093   0.000282   ***
Temp                 -2.710981    0.618472   -4.383   0.000125   ***
Industry              0.064988    0.012264    5.299   9.11e-06   ***
Population           -0.039430    0.011976   -3.293   0.002485   **
Wind                 -7.519344    8.151943   -0.922   0.363444
Rain                 -6.760530    1.792173   -3.772   0.000685   ***
Wet.days              1.266742    0.517850    2.446   0.020311   *
Wind:Rain             0.631457    0.243866    2.589   0.014516   *
Wind:Wet.days        -0.230452    0.069843   -3.300   0.002440   **
Wind:Rain:Wet.days    0.002497    0.001214    2.056   0.048247   *

Residual standard error: 11.2 on 31 degrees of freedom
Multiple R-squared: 0.8236,  Adjusted R-squared: 0.7724
F-statistic: 16.09 on 9 and 31 DF, p-value: 2.231e-09
```

There is indeed a marginally significant three-way interaction. You should confirm that there is no place for quadratic terms for either `Temp` or `Wind` in this model.

That's enough for now. I'm sure you get the idea. Multiple regression is difficult, time-consuming, and always vulnerable to subjective decisions about what to include and what to leave out. The linear modelling confirms the early impression from the tree model: for low levels of industry, the SO_2 level depends in a simple way on population (people tend to want to live where the air is clean) and in a complicated way on daily weather (as reflected by the three-way interaction between wind, total rainfall and the number of wet days (i.e. on rainfall intensity)).

Further Reading

Claeskens, G. and Hjort, N.L. (2008) *Model Selection and Model Averaging*, Cambridge University Press, Cambridge.

Draper, N.R. and Smith, H. (1981) *Applied Regression Analysis*, John Wiley & Sons, New York.

Fox, J. (2002) *An R and S-Plus Companion to Applied Regression*, Sage, Thousand Oaks, CA.

Mosteller, F. and Tukey, J.W. (1977) *Data Analysis and Regression*, Addison-Wesley, Reading, MA.

11

Contrasts

One of the hardest things about learning to do statistics is knowing how to interpret the output produced by the model that you have fitted to data. The reason why the output is hard to understand is that it is based on *contrasts*, and the idea of contrasts may be unfamiliar to you.

Contrasts are the essence of hypothesis testing and model simplification. They are used to compare means or groups of means with other means or groups of means, in what are known as *single degree of freedom comparisons*. There are two sorts of contrasts we might want to carry out:

- contrasts we had planned to carry out at the experimental design stage (these are referred to as *a priori* contrasts)
- contrasts that look interesting after we have seen the results (these are referred to as *a posteriori* contrasts)

Some people are very snooty about a posteriori contrasts, on the grounds that they were *unplanned*. You are not supposed to decide what comparisons to make *after* you have seen the analysis, but scientists do this all the time. The key point is that you should only do contrasts *after* the ANOVA has established that there really are significant differences to be investigated. It is bad practice to carry out tests to compare the largest mean with the smallest mean, if the ANOVA fails to reject the null hypothesis (tempting though this may be).

There are two important points to understand about contrasts:

- there is a huge number of *possible* contrasts
- there are only $k-1$ *orthogonal* contrasts

where k is the number of factor levels. Two contrasts are said to be orthogonal to one another if the comparisons are statistically independent. Technically, two contrasts are orthogonal if *the products of their contrast coefficients sum to zero* (we shall see what this means in a moment).

Let's take a simple example. Suppose we have one factor with five levels and the factor levels are called a, b, c, d and e. Let us start writing down the possible contrasts. Obviously we could compare each mean singly with every other:

$$a \text{ vs. } b, a \text{ vs. } c, a \text{ vs. } d, a \text{ vs. } e, b \text{ vs. } c, b \text{ vs. } d, b \text{ vs. } e, c \text{ vs. } d, c \text{ vs. } e, d \text{ vs. } e$$

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

but we could also compare pairs of means:

$$\{a,b\} \text{ vs. } \{c,d\}, \{a,b\} \text{ vs. } \{c,e\}, \{a,b\} \text{ vs. } \{d,e\}, \{a,c\} \text{ vs. } \{b,d\}, \{a,c\} \text{ vs. } \{b,e\}, \ldots$$

or triplets of means:

$$\{a,b,c\} \text{ vs. } d, \{a,b,c\} \text{ vs. } e, \{a,b,d\} \text{ vs. } c, \{a,b,d\} \text{ vs. } e, \{a,c,d\} \text{ vs. } b, \ldots$$

or groups of four means:

$$\{a,b,c,d\} \text{ vs. } e, \{a,b,c,e\} \text{ vs. } d, \{b,c,d,e\} \text{ vs. } a, \{a,b,d,e\} \text{ vs. } c, \{a,b,c,e\} \text{ vs. } d$$

I think you get the idea. There are absolutely masses of possible contrasts. In practice, however, we should only compare things once, either directly or implicitly. So the two contrasts a vs. b and a vs. c implicitly contrast b vs. c. This means that if we have carried out the two contrasts a vs. b and a vs. c then the third contrast b vs. c is *not* an orthogonal contrast because you have already carried it out, implicitly. Which particular contrasts are orthogonal depends very much on your choice of the first contrast to make. Suppose there were good reasons for comparing $\{a, b, c, e\}$ vs. d. For example, d might be the placebo and the other four might be different kinds of drug treatment, so we make this our first contrast. Because $k-1=4$ we only have three possible contrasts that are orthogonal to this. There may be a priori reasons to group $\{a, b\}$ and $\{c, e\}$, so we make this our second orthogonal contrast. This means that we have no degrees of freedom in choosing the last two orthogonal contrasts: they have to be a vs. b and c vs. e. Just remember that *with orthogonal contrasts you compare things only once.*

Contrast Coefficients

Contrast coefficients are a numerical way of embodying the hypothesis we want to test. The rules for constructing contrast coefficients are straightforward:

- treatments to be lumped together get the same sign (plus or minus)
- groups of means to be contrasted get opposite sign
- factor levels to be excluded get a contrast coefficient of 0
- the contrast coefficients must add up to 0

Suppose that with our five-level factor $\{a, b, c, d, e\}$ we want to begin by comparing the four levels $\{a, b, c, e\}$ with the single level d. All levels enter the contrast, so none of the coefficients is 0. The four terms $\{a, b, c, e\}$ are grouped together so they all get the same sign (minus, for example, although it makes no difference which sign is chosen). They are to be compared to d, so it gets the opposite sign (plus, in this case). The choice of what numeric values to give the contrast coefficients is entirely up to you. Most people use whole numbers rather than fractions, but it really doesn't matter. All that matters is that the values chosen sum to 0. The positive and negative coefficients have to add up to the same value. In our example, comparing four means with one mean, a natural choice of coefficients would be

-1 for each of $\{a, b, c, e\}$ and +4 for d. Alternatively, we could have selected +0.25 for each of $\{a, b, c, e\}$ and -1 for d.

```
factor level:                 a      b      c      d      e

contrast 1 coefficients, c: -1     -1     -1      4     -1
```

Suppose the second contrast is to compare $\{a, b\}$ with $\{c, e\}$. Because this contrast excludes d, we set its contrast coefficient to 0. $\{a, b\}$ get the same sign (say, plus) and $\{c, e\}$ get the opposite sign. Because the number of levels on each side of the contrast is equal (two in both cases) we can use the same numeric value for all the coefficients. The value 1 is the most obvious choice (but you could use 13.7 if you wanted to be perverse):

```
factor level:                a       b      c      d      e

contrast 2 coefficients, c:  1       1     -1      0     -1
```

There are only two possibilities for the remaining orthogonal contrasts: a vs. b and c vs. e:

```
factor level:                a      b      c      d      e
contrast 3 coefficients, c:  1     -1      0      0      0
contrast 4 coefficients, c:  0      0      1      0     -1
```

An Example of Contrasts in R

The example comes from the competition experiment we analysed in Chapter 9 in which the biomass of control plants is compared to the biomass of plants grown in conditions where competition was reduced in one of four different ways. There are two treatments in which the roots of neighbouring plants were cut (to 5 cm depth or 10 cm) and two treatments in which the shoots of neighbouring plants were clipped (25% or 50% of the neighbours cut back to ground level; see p. 162).

```
comp <- read.csv("c:\\temp\\competition.csv")
attach(comp)
names(comp)
```

```
[1] "biomass" "clipping"
```

We start with the one-way ANOVA:

```
model1 <- aov(biomass~clipping)
summary(model1)
```

```
          Df  Sum Sq  Mean Sq  F  value   Pr(>F)
clipping   4   85356   21339      4.302  0.00875 **
Residuals 25 124020     4961
```

Clipping treatment has a highly significant effect on biomass. But have we fully understood the result of this experiment? I don't think so. For example, which factor levels had the biggest effect on biomass? Were all of the competition treatments significantly different from the controls ? To answer these questions, we need to use `summary.lm`:

```
summary.lm(model1)
```

```
Coefficients:
              Estimate Std. Error t value   Pr(>|t|)
(Intercept)   465.17     28.75   16.177   9.4e-15    ***
clippingn25    88.17     40.66    2.168   0.03987    *
clippingn50   104.17     40.66    2.562   0.01683    *
clippingr10   145.50     40.66    3.578   0.00145    **
clippingr5    145.33     40.66    3.574   0.00147    **

Residual standard error: 70.43 on 25 degrees of freedom
Multiple R-squared: 0.4077,  Adjusted R-squared: 0.3129
F-statistic: 4.302 on 4 and 25 DF,  p-value: 0.008752
```

It looks as if we need to keep all five parameters, because all five rows of the summary table have one or more significance stars. If fact, this is not the case. This example highlights the major shortcoming of *treatment contrasts* (the default contrast method in R): they do not show how many significant factor levels we need to retain in the minimal adequate model.

A Priori Contrasts

In this experiment, there are several planned comparisons we should like to make. The obvious place to start is by comparing the control plants that were exposed to the full rigours of competition, with all of the other treatments.

```
levels(clipping)
```

```
[1]  "control"  "n25"    "n50"    "r10"    "r5"
```

That is to say, we want to contrast the first level of clipping with the other four levels. The contrast coefficients, therefore, would be 4, −1, −1, −1, −1. The next planned comparison might contrast the shoot-pruned treatments (n25 and n50) with the root-pruned treatments (r10 and r5). Suitable contrast coefficients for this would be 0, 1, 1, −1, −1 (because we are ignoring the control in this contrast). A third contrast might compare the two depths of root-pruning; 0, 0, 0, 1, −1. The last orthogonal contrast would therefore have to compare the two intensities of shoot-pruning: 0, 1, −1, 0, 0. Because the clipping factor has five levels there are only $5 - 1 = 4$ orthogonal contrasts.

R is outstandingly good at dealing with contrasts, and we can associate these four user-specified a priori contrasts with the categorical variable called clipping like this:

```
contrasts(clipping) <-
cbind(c(4,-1,-1,-1,-1),c(0,1,1,-1,-1),c(0,0,0,1,-1),c(0,1,-1,0,0))
```

We can check that this has done what we wanted by typing

```
contrasts(clipping)

        [,1]  [,2] [,3] [,4]
control    4     0    0    0
n25       -1     1    0    1
n50       -1     1    0   -1
r10       -1    -1    1    0
r5        -1    -1   -1    0
```

which produces the matrix of contrast coefficients that we specified. Note that all the columns add to zero (i.e. each set of contrast coefficients is correctly specified). Note also that the products of any two of the columns sum to zero (this shows that all the contrasts are orthogonal, as intended): for example comparing contrasts 1 and 2 gives products $0+(-1)+(-1)+1+1=0$.

Now we can refit the model and inspect the results of our specified contrasts, rather than the default treatment contrasts:

```
model2 <- aov(biomass~clipping)
summary.lm(model2)

Coefficients:
             Estimate Std. Error t value   Pr (>|t|)
(Intercept) 561.80000  12.85926  43.688   < 2e-16 ***
clipping1   -24.15833   6.42963  -3.757  0.000921 ***
clipping2   -24.62500  14.37708  -1.713  0.099128 .
clipping3     0.08333  20.33227   0.004  0.996762
clipping4    -8.00000  20.33227  -0.393  0.697313

Residual standard error: 70.43 on 25 degrees of freedom
Multiple R-squared: 0.4077,  Adjusted R-squared: 0.3129
F-statistic: 4.302 on 4 and 25 DF, p-value: 0.008752
```

Instead of requiring five parameters (as suggested by our initial treatment contrasts), this analysis shows that we need only two parameters: the overall mean (561.8) and the contrast between the controls and the four competition treatments ($p=0.000921$). All the other contrasts are non-significant. Notice than the rows are labelled by the variable name `clipping` pasted to the contrast number (`1` to `4`).

Treatment Contrasts

We have seen how to specify our own customized contrasts (above) but we need to understand the default behaviour of R. This behaviour is called *treatment contrasts* and it is set like this:

```
options(contrasts=c("contr.treatment","contr.poly"))
```

The first argument, `"contr.treatment"`, says we want to use treatment contrasts (rather than Helmert contrasts or sum contrasts), while the second, `"contr.poly"`, sets the default method for comparing ordered factors (where the levels are semi-quantitative like ‘low’, ‘medium’ and ‘high’). We shall see what these unfamiliar terms mean in a moment. Because we had associated our explanatory variable `clipping` with a set of user-defined contrasts (above), we need to remove these to get back to the default treatment contrasts, like this:

```
contrasts(clipping) <- NULL
```

Now let us refit the model and work out exactly what the `summary` is showing:

```
model3 <- lm(biomass~clipping)
summary(model3)
```

```
Coefficients:
             Estimate Std. Error  t value   Pr(>|t|)
(Intercept)  465.17      28.75    16.177    9.4e-15    ***
clippingn25   88.17      40.66     2.168    0.03987    *
clippingn50  104.17      40.66     2.562    0.01683    *
clippingr10  145.50      40.66     3.578    0.00145    **
clippingr5   145.33      40.66     3.574    0.00147    **

Residual standard error: 70.43 on 25 degrees of freedom
Multiple R-squared: 0.4077,  Adjusted R-squared: 0.3129
F-statistic: 4.302 on 4 and 25 DF, p-value: 0.008752
```

First things first. There are five rows in the summary table, labelled `(Intercept)`, `clippingn25`, `clippingn50`, `clippingr10` and `clippingr55`. Why are there five rows? There are five rows in the summary table because this model has estimated five parameters from the data (one for each factor level). This is a very important point. Different models will produce summary tables with different numbers of rows. You should always be aware of how many parameters your model is estimating from the data, so you should never be taken by surprise by the number of rows in the summary table.

Next we need to think what the individual rows are telling us. Let us work out the overall mean value of biomass, to see if that appears anywhere in the table:

```
mean(biomass)
```

```
[1] 561.8
```

No. I can’t see that number anywhere. What about the five treatment means? We obtain these with the extremely useful `tapply` function:

```
tapply(biomass,clipping, mean)
```

```
 control      n25      n50      r10       r5
465.1667 553.3333 569.3333 610.6667 610.5000
```

This is interesting. The first treatment mean (465.1667 for the controls) *does* appear (in row number 1), but none of the other means are shown. So what does the value of 88.17 in row number 2 of the table mean? Inspection of the individual treatment means lets you see that 88.17 is the *difference* between the mean biomass for n25 and the mean biomass for the control. What about 104.17 ? That turns out to be the difference between the mean biomass for n50 and the mean biomass for the control. Treatment contrasts are expressed as *the difference between the mean for a particular factor level and the mean for the factor level that comes first in the alphabet*. No wonder they are hard to understand on first encounter.

Why does it all have to be so complicated? What's wrong with just showing the five mean values? The answer has to do with parsimony and model simplification. We want to know whether the differences between the means that we observe (e.g. using tapply) are statistically significant. To assess this, we need to know the difference between the means and (crucially) the standard error of the difference between two means (see p. 91 for a refresher on this). The summary.lm table is designed to make this process as simple as possible. It does this by showing us just one mean (the intercept) and all the other rows are differences between means. Likewise, in the second column, it shows us just one standard error (the standard error of the mean for the intercept – the control treatment in this example), and all of the other values are the standard error of the difference between two means. So the value in the top row (28.75) is much smaller than the values in the other four rows (40.66) because the standard error of a mean is

$$\sqrt{\frac{s^2}{n}}$$

while the standard error of the difference between two means is

$$\sqrt{\frac{s_\mathrm{A}^2}{n_\mathrm{A}}+\frac{s_\mathrm{B}^2}{n_\mathrm{B}}}=\sqrt{\frac{2s^2}{n}}$$

You can see that for the simplest case where (as here) we have equal replication in all five factor levels ($n = 6$) and we intend to use the pooled error variance s^2, the standard error of the difference between two means is $\sqrt{2} = 1.414$ times the size of the standard error of a mean. Let us check this out for our example

```
28.75*sqrt(2)
[1] 40.65864
```

So that checks out. What if it is not convenient to compare the means with the factor level that happens to come first in the alphabet ? Then all we need to do is change the order of the levels with a new factor declaration, in which we specify the order in which the factor levels are to be considered. You can see an example of this on p. 242. Alternatively, you can define the contrasts attributes of the factor, as we saw earlier in this chapter.

Model Simplification by Stepwise Deletion

Model simplification with treatment contrasts requires that we aggregate non-significant factor levels in a stepwise a posteriori procedure. The results we get by doing this are *exactly* *the same* as those obtained by defining the contrast attributes of the explanatory variable as we did earlier.

Looking down the list of parameter estimates, we see that the most similar effect sizes are for r5 and r10 (the effects of root pruning to 5 and 10 cm): 145.33 and 145.5, respectively. We shall begin by simplifying these to a single root-pruning treatment called root. The trick is to use 'levels gets' to change the names of the appropriate factor levels. Start by copying the original factor name:

```
clip2 <- clipping
```

Now inspect the level numbers of the various factor level names:

```
levels(clip2)
[1] "control" "n25"   "n50"   "r10"   "r5"
```

The plan is to lump together r10 and r5 under the same name, 'root'. These are the fourth and fifth levels of clip2 (count them from the left), so we write:

```
levels(clip2)[4:5] <- "root"
```

and to see what has happened we type:

```
levels(clip2)
[1] "control" "n25"   "n50"   "root"
```

We see that "r10" and "r5" have been merged and replaced by "root". The next step is to fit a new model with clip2 in place of clipping, and to use anova to test whether the new simpler model with only four factor levels is significantly worse as a description of the data:

```
model4 <- lm(biomass~clip2)
anova(model3,model4)

Analysis of Variance Table

Model 1: biomass ~ clipping
Model 2: biomass ~ clip2
  Res.Df   RSS Df Sum of Sq     F Pr(>F)
1     25 124020
2     26 124020 -1 -0.0833333 0.0000168 0.9968
```

As we expected, this model simplification was completely justified. The next step is to investigate the effects using summary:

```
summary(model4)
```

```
Coefficients:
            Estimate Std. Error  t value    Pr(>|t|)
(Intercept)   465.17     28.20   16.498   2.72e-15  ***
clip2n25       88.17     39.87    2.211   0.036029  *
clip2n50      104.17     39.87    2.612   0.014744  *
clip2root     145.42     34.53    4.211   0.000269  ***

Residual standard error: 69.07 on 26 degrees of freedom
Multiple R-squared: 0.4077,  Adjusted R-squared: 0.3393
F-statistic: 5.965 on 3 and 26 DF, p-value: 0.003099
```

Next, we want to compare the two shoot-pruning treatments with their effect sizes of 88.17 and 104.17. They differ by 16.0 (much more than the root-pruning treatments did), but you will notice that the standard error of the difference if 39.87. You will soon get used to doing *t* tests in your head and using the rule of thumb that *t* needs to be bigger than 2 for significance. In this case, the difference would need to be at least $2 \times 39.87 = 79.74$ for significance, so we predict that merging the two shoot pruning treatments will not cause a significant reduction in the explanatory power of the model. We can lump these together into a single shoot-pruning treatment as follows:

```
clip3 <- clip2
levels(clip3)[2:3] <- "shoot"
levels(clip3)
```

```
[1] "control" "shoot"  "root"
```

Then we fit a new model with clip3 in place of clip2:

```
model5 <- lm(biomass~clip3)
anova(model4,model5)
```

```
Analysis of Variance Table

Model 1: biomass ~ clip2
Model 2: biomass ~ clip3
  Res.Df    RSS Df Sum of Sq     F Pr(>F)
1     26 124020
2     27 124788 -1      -768 0.161 0.6915
```

Again, this simplification was fully justified. Do the root and shoot competition treatments differ?

```
summary(model5)
```

```
Coefficients:
             Estimate Std. Error  t value    Pr(>|t|)
(Intercept)   465.17      27.75    16.760    8.52e-16   ***
clip3shoot     96.17      33.99     2.829    0.008697   **
clip3root     145.42      33.99     4.278    0.000211   ***

Residual standard error: 67.98 on 27 degrees of freedom
Multiple R-squared: 0.404,   Adjusted R-squared: 0.3599
F-statistic: 9.151 on 2 and 27 DF, p-value: 0.0009243
```

The two effect sizes differ by $145.42-96.17=49.25$ with a standard error of 33.99 (giving $t<2$) so we shall be surprised if merging the two factor levels causes a significant reduction in the model's explanatory power:

```
clip4 <- clip3
levels(clip4)[2:3] <- "pruned"
levels(clip4)
```

```
[1] "control" "pruned"
```

Now we fit a new model with `clip4` in place of `clip3`:

```
model6 <- lm(biomass~clip4)
anova(model5,model6)
```

```
Analysis of Variance Table

Model 1: biomass ~ clip3
Model 2: biomass ~ clip4
  Res.Df      RSS Df Sum of Sq      F Pr(>F)
1     27 124788
2     28 139342 -1   -14553 3.1489 0.08726 .
```

This simplification was close to significant, but we are ruthless (p > 0.05), so we accept the simplification. Now we have the minimal adequate model:

```
summary(model6)
```

```
Coefficients:
             Estimate Std. Error  t value    Pr (>|t|)
(Intercept)   465.2       28.8     16.152    1.01e-15   ***
clip4pruned   120.8       32.2      3.751    0.000815   ***

Residual standard error: 70.54 on 28 degrees of freedom
Multiple R-squared: 0.3345,   Adjusted R-squared: 0.3107
F-statistic: 14.07 on 1 and 28 DF, p-value: 0.0008149
```

It has just two parameters: the mean for the controls (465.2) and the difference between the control mean and the four other treatment means ($465.2+120.8=586.0$):

```
tapply(biomass,clip4,mean)
```

```
 control  pruned
465.1667 585.9583
```

We know from the p value $= 0.000815$ (above) that these two means are significantly different, but just to show how it is done, we can make a final `model7` that has no explanatory variable at all (it fits only the overall mean). This is achieved by writing `y~1` in the model formula (parameter `1`, you will recall, is the intercept in R):

```
model7 <- lm(biomass~1)
anova(model6,model7)
```

```
Analysis of Variance Table

Model 1: biomass ~ clip4
Model 2: biomass ~ 1
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
1     28 139342
2     29 209377 -1    -70035 14.073 0.0008149 ***
```

Note that the p value is exactly the same as in `model6`. The p values in R are calculated such that they avoid the need for this final step in model simplification: they are 'p on deletion' values.

It is informative to compare what we have just done by stepwise factor-level reduction with what we did earlier by specifying the `contrast` attribute of the factor.

Contrast Sums of Squares by Hand

The key point to understand is that *the treatment sum of squares SSA is the sum of all k* − 1 *orthogonal sums of squares*. It is very useful to know which of the contrasts contributes most to *SSA*, and to work this out, we compute a set of contrast sums of squares *SSC* as follows:

$$SSC = \frac{\left(\sum \frac{c_i T_i}{n_i}\right)^2}{\sum \frac{c_i^2}{n_i}}$$

or, when all the sample sizes are the same for each factor level:

$$SSC = \frac{\left(\frac{1}{n}\sum c_i T_i\right)^2}{\frac{1}{n}\sum c_i^2}$$

The significance of a contrast is judged in the usual way by carrying out an F test to compare the contrast variance with the pooled error variance, s^2 from the ANOVA table.

Since all contrasts have a single degree of freedom, the contrast variance is equal to *SSC*, so the *F* test is just

$$F = \frac{SSC}{s^2}$$

The contrast is significant (i.e. the two contrasted groups have significantly different means) if the calculated value of the test statistic is larger than the critical value of *F* with 1 and $k(n-1)$ degrees of freedom. We demonstrate these ideas by continuing our example. When we did factor-level reduction in the previous section, we obtained the following sums of squares for the four orthogonal contrasts that we carried out (the figures come from the model-comparison ANOVA tables, above):

Contrast	Sum of squares
Control vs. the rest	70 035
Shoot pruning vs. root pruning	14 553
Intense vs. light shoot pruning	768
Deep vs. shallow root pruning	0.083
Total	

What we shall do now is carry out the calculations involved in the a priori contrasts in order to demonstrate that the results are identical. The new quantities we need to compute the contrast sums of squares *SSC* are the treatment totals, T_i, in the formula above and the contrast coefficients c_i. It is worth writing down the contrast coefficients again (see p. 214, above):

Contrast	Contrast coefficients				
Control vs. the rest	4	−1	−1	−1	−1
Shoot pruning vs. root pruning	0	1	1	−1	−1
Intense vs. light shoot pruning	0	1	−1	0	0
Deep vs. shallow root pruning	0	0	0	1	−1

We use tapply to obtain the five treatment totals:

```
tapply(biomass,clipping,sum)
control     n25     n50     r10      r5
   2791    3320    3416    3664    3663
```

For the first contrast, the contrast coefficients are 4, −1, −1, −1, and −1. The formula for *SSC* is therefore

$$\frac{\left[\frac{1}{6}\times\{(4\times 2791)+(-1\times 3320)+(-1\times 3416)+(-1\times 3664)+(-1\times 3663)\}\right]^2}{\frac{1}{6}\times\ [4^2+(-1)^2+(-1)^2+(-1)^2+(-1)^2]}$$

$$=\frac{\left(\frac{1}{6}\times -2899\right)^2}{\frac{20}{6}}=\frac{233\,450}{3.33333}=70\,035$$

In the second contrast, the coefficient for the control is zero, so we leave this out:

$$\frac{\left[\frac{1}{6}\times\{(1\times 3320)+(1\times 3416)+(-1\times 3664)+(-1\times 3663)\}\right]^2}{\frac{1}{6}\times\ [(-1)^2+(-1)^2+(-1)^2+(-1)^2]}=\frac{\left(\frac{1}{6}\times -591\right)^2}{\frac{4}{6}}$$

$$=\frac{9702.25}{0.666666}=14\,553$$

The third contrast ignores the root treatments, so:

$$\frac{\left[\frac{1}{6}\times\{(1\times 3320)+(-1\times 3416)\}\right]^2}{\frac{1}{6}\times\ [(-1)^2+(1)^2]}=\frac{\left(\frac{1}{6}\times -96\right)^2}{\frac{2}{6}}=\frac{256}{0.33333}=768$$

Finally, we compare the two root pruning treatments:

$$\frac{\left[\frac{1}{6}\times\{(1\times 3664)+(-1\times 3663)\}\right]^2}{\frac{1}{6}\times\ [(-1)^2+(1)^2]}=\frac{\left(\frac{1}{6}\times 1\right)^2}{\frac{2}{6}}=\frac{0.027778}{0.333333}=0.083333$$

You will see that these sums of squares are exactly the same as we obtained by stepwise factor-level reduction. So don’t let anyone tell you that these procedures are different or that one of them is superior to the other.

The Three Kinds of Contrasts Compared

You are very unlikely to need to use Helmert contrasts or sum contrasts in your work, but if you need to find out about them, they are explained in detail in *The R Book* (Crawley, 2013).

Reference

Crawley, M.J. (2013) *The R Book*, 2nd edn, John Wiley & Sons, Chichester.

Further Reading

Ellis, P.D. (2010) *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*, Cambridge University Press, Cambridge.

Miller, R.G. (1997) *Beyond ANOVA: Basics of Applied Statistics*, Chapman & Hall, London.

12

Other Response Variables

Up to now, the response variable has been a continuous real number such as a weight, length or concentration. We made several important assumptions about the behaviour of the response variable, and it is worth reiterating those assumptions here, ranked in order of importance:

- random sampling
- constant variance
- normal errors
- independent errors
- additive effects

So far, when we found that one or more of the assumptions was wrong, our typical resort was transformation of the response variable, coupled perhaps with transformation of one or more of the explanatory variables.

In our line of work, we often meet with response variables where the key assumptions are rarely if ever met. In these cases, it is sensible to look for alternatives to transformation that might improve our ability to model these systems effectively. In this book we cover four new kinds of response variable that are very common in practice:

- count data
- proportion data
- binary response data
- age-at-death data

all of which routinely fail the assumptions about constancy of variance and normality of errors. It turns out that these different kinds of response variables have more in common than you might first imagine, and that they can all be dealt with in the framework of generalized linear models (GLMs). These models allow variance to be non-constant and errors to be non-normally distributed. It is worth noting, however, that they still assume random

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.
© 2015 John Wiley & Sons, Ltd. Published 2015 by John Wiley & Sons, Ltd.

sampling and independence of errors. Effects in GLMs may be additive or multiplicative, depending on the circumstances.

We begin by thinking about variance. Count data are whole numbers (integers) so you can see immediately that when the mean is low, the data are likely to consist only of zeros, ones and twos, with the odd three of four thrown in. This being the case, the variance of count data is bound to be low when the mean is low (recall that variance is the sum of the squares of the departures of the counts from the mean count, divided by the degrees of freedom). However, when the mean of count data is high, the range of individual counts can be from zero to potentially very large numbers, and when we work out the residuals and square them we can expect to obtain very large numbers, and a consequently high variance. For count data, therefore, the variance is expected to increase with the mean, rather than being constant as assumed in linear models. For the kind of count data we are talking about here, we know the number of times that something happened (lightning strikes, cells on a microscope slide, insects on a leaf), but not the number of times it did *not* happen.

For proportion data based we typically have a count of the number of individuals doing one thing and another count of the number not doing that thing. Both these counts are free to vary and both counts are important in analysing the data properly. In general the numbers doing the thing in question are known as *successes* and the numbers not doing it are known as *failures*. This can be rather macabre, as when the response variable is the number of people dying in a medical trail. Examples of proportion data based on counts are:

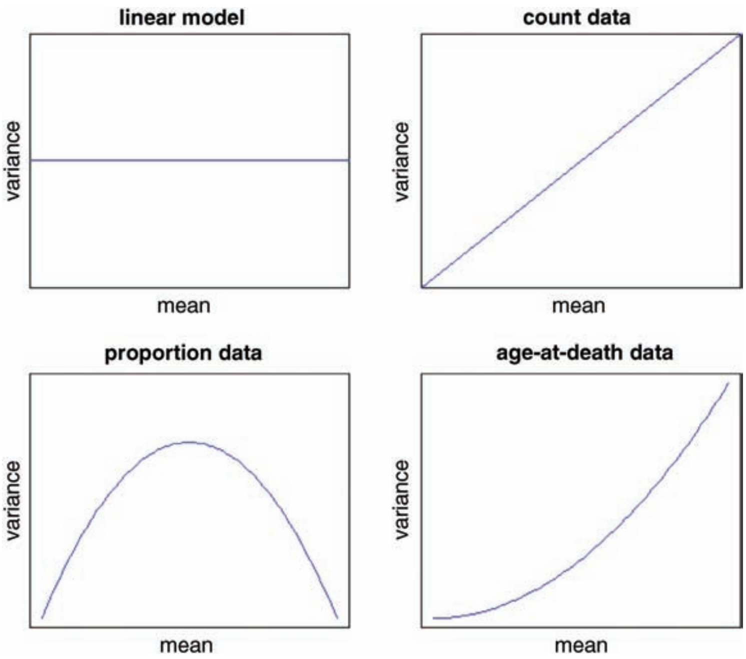
Successes	Failures
dead	alive
female	male
diseased	healthy
occupied	unoccupied
pollinated	not pollinated
adult	juvenile

You can see that this kind of proportion data based on counts arises in a great many kinds of circumstances. Let us think about the variance of proportion data like this. If the success rate is 100% then all the individuals are alike and the variance is zero. Again, if the success rate is zero, then all the individuals are alike and the variance is zero. If, however, the success rate is intermediate (50%, say) then some of the individuals are in one class and some are in the other, so variance is high. This means that unlike count data (above) where the variance increased monotonically with the mean, for proportion data the variance is a humped function of the mean. The binomial distribution is an important example of the kind of distribution used in the analysis of proportion data: if p is the probability of success and n is the number of trails, then the mean number of successes is np and the variance in the number of successes is $np(1-p)$. As you can see, the variance is 0 when $p=1$ and 0 when $p=0$, reaching a peak when $p=0.5$.

Another sort of proportion data (such as percentage cover data from plant ecology), is not based on counts, but involves continuous numbers that are bounded both above and below (for instance, you cannot have negative percentage cover, or cover values that are greater than 100%). These kinds of proportion data are arcsine transformed prior to analysis, then analysed using linear models (see p. 257).

A very particular kind of response variable is analysed by medical researchers: this is the age at death. The research questions centre on the effect of a particular treatment on (hopefully) increasing the age at death of the patients receiving this treatment compared to patients receiving the placebo (the controls). Age-at-death data are notorious for their non-constant variance. We saw that for count data the variance increases with the mean, but for age-at-death data the situation is even more extreme than this: the variance increases with the square of the mean.

Here are graphs of the variance as a function of the mean for four contrasting kinds of response variable: data suitable for analysis using linear models (constant variance, top left); count data (linearly increasing variance, top right); proportion data (humped variance mean relationship, bottom left) and age-at-death data (quadratically increasing variance, bottom right).



Introduction to Generalized Linear Models

A generalized linear model has three important properties:

- the error structure
- the linear predictor
- the link function

These are all likely to be unfamiliar concepts. The ideas behind them are straightforward, however, and it is worth learning what each of the concepts involves.

The Error Structure

Up to this point, we have dealt with the statistical analysis of data with normal errors. In practice, however, many kinds of data have non-normal errors:

- errors that are strongly skewed
- errors that are kurtotic
- errors that are strictly bounded (as in proportions)
- errors that cannot lead to negative fitted values (as in counts)

In the past, the only tools available to deal with these problems were transformation of the response variable or the adoption of non-parametric methods. A GLM allows the specification of a variety of different error distributions:

- Poisson errors, useful with count data
- binomial errors, useful with data on proportions
- gamma errors, useful with data showing a constant coefficient of variation
- exponential errors, useful with data on time to death (survival analysis)

The error structure is defined by means of the `family` directive, used as part of the model formula like this:

```
glm(y ~ z, family = poisson)
```

which means that the response variable y has Poisson errors. Or

```
glm(y ~ z, family = binomial)
```

which means that the response is binary, and the model has binomial errors. As with previous models, the explanatory variable z can be continuous (leading to a regression analysis) or categorical (leading to an ANOVA-like procedure called analysis of deviance, as described below).

The Linear Predictor

The structure of the model relates each observed y value to a predicted value. The predicted value is obtained *by transformation of the value emerging from the linear predictor*. The linear predictor, η (eta), is a linear sum of the effects of one or more explanatory variables, x_j, and is what we see when we ask for `summary.lm`:

$$\eta_i = \sum_{j=1}^{p} x_{ib}\beta_j$$

where the xs are the values of the p different explanatory variables, and the βs are the (usually) unknown parameters to be estimated from the data. The right-hand side of the equation is called the *linear structure*.

There are as many terms in the linear predictor as there are parameters, p, to be estimated from the data. Thus with a simple regression, the linear predictor is the sum of two terms whose parameters are the intercept and the slope. With a one-way ANOVA with four treatments, the linear predictor is the sum of four terms leading to the estimation of the mean for each level of the factor. If there are covariates in the model, they add one term each to the linear predictor (the slope of each relationship). Interaction terms in a factorial ANOVA add one or more parameters to the linear predictor, depending upon the degrees of freedom of each factor (e.g. there would be three extra parameters for the interaction between a two-level factor and a four-level factor, because $(2-1)\times(4-1)=3$).

The linear predictor can be inspected by typing `summary.lm`: there are as many rows as there are parameters in your model, and for every parameter you get the effect size and the standard error of the effect in the first two columns. The other columns are less important because you could easily work them out yourself: the t value, the p value and the significance stars.

What you will find difficult at first is knowing exactly what effects are being shown on each row. In an analysis of covariance, for example, the top row will contain an intercept and the second row might contain a slope, but the other rows will all be differences between intercepts and/or differences between slopes (i.e. there is only *one* slope in the table and only *one* intercept, no matter how many slopes or intercepts there are in the fitted model).

Fitted Values

One of the great strengths of GLMs is that different models can be compared on the same scale of measurement as the response variable. Up to now, we have used *variance* to measure the lack of fit between the model and the data: this was the sum of the squares of the difference between the response variable y and the fitted values predicted by the model $\hat{y}$, which is $\sum(y-\hat{y})^2$ divided by the degrees of freedom. Now if you fit a different model, say $\log(y)=a+bx$, then obviously the variance is completely different because it is based on $\sum(\log(y)-\hat{y})^2$. This makes model comparison difficult, because there is no common currency for measuring the fit of the two models to the data. In a GLM, however, we always compare y and $\hat{y}$ on the same scale on which the response was measured (as a count, for instance, or as a proportion based on two counts). This makes model comparison much more straightforward.

A General Measure of Variability

The difference is that the measure of lack of fit of the model to the data depends on the context. We give it the new, more general name of *deviance*. The technical definition won't mean much to you at this stage, but here it is:

$$\text{deviance} = -2 \times \text{log likelihood}$$

where the log likelihood depends on *the model given the data*. We don't need to unlearn anything, because deviance is the same as variance when we have constant variance and normal errors (as in linear regression, ANOVA or ANCOVA). But for count data we *do* need a different measure of lack of fit (it is based on $y\log(y/\hat{y})$ rather than on $(y-\hat{y})^2$) and we need a different definition of deviance for proportion data, and so on. But the point is that for measuring the fit, we compare y and $\hat{y}$ on the same original untransformed scale. We shall discuss the various deviance measures in more detail in the following chapters, but so that you can compare them with one another, here are the main measures of lack of fit side-by-side:

Model	Deviance	Error	Link
linear	$\sum(y-\hat{y})^2$	Gaussian	identity
log linear	$2\sum y\log\left(\frac{y}{\hat{y}}\right)$	Poisson	log
logistic	$2\sum y\log\left(\frac{y}{\hat{y}}\right)+(n-y)\log\left(\frac{n-y}{n-\hat{y}}\right)$	binomial	logit
gamma	$2\sum\frac{(y-\hat{y})}{y}-\log\left(\frac{y}{\hat{y}}\right)$	gamma	reciprocal

To determine the fit of a given model, a GLM evaluates the linear predictor for each value of the response variable, then back-transforms the predicted value to compare it with the observed value of y. The parameters are then adjusted, and the model refitted on the transformed scale in an iterative procedure until the fit stops improving. It will take some time before you understand what is going on here, and why it is so revolutionary. Don't worry; it will come with practice.

The Link Function

One of the difficult things to grasp about GLMs is the relationship between the values of the response variable (as measured in the data and predicted by the model as fitted values) and the linear predictor.

The transformation to be employed is specified in the link function. The fitted value is computed by applying the reciprocal of the link function, in order to get back to the original scale of measurement of the response variable. Thus, with a log link, the fitted value is the antilog of the linear predictor, and with the reciprocal link, the fitted value is the reciprocal of the linear predictor.

The thing to remember is that the *link function relates the mean value of y to its linear predictor*. In symbols, this means that:

$$\eta = g(\mu)$$

which is simple, but needs thinking about. The linear predictor, η, emerges from the linear model as a sum of the terms for each of the p parameters. *This is not a value of y* (except in the special case of the *identity link* that we have been using (implicitly) up to now). The value of η is obtained by transforming the value of y by the link function, and the predicted value of y is obtained by applying the inverse link function to η.

The most frequently used link functions are shown below. An important criterion in the choice of link function is to ensure that the fitted values stay within reasonable bounds. We would want to ensure, for example, that counts were all greater than or equal to zero (negative count data would be nonsense). Similarly, if the response variable was the proportion of individuals who died, then the fitted values would have to lie between 0 and 1 (fitted values greater than 1 or less than 0 would be meaningless). In the first case, a log link is appropriate because the fitted values are antilogs of the linear predictor, and all antilogs are greater than or equal to zero. In the second case, the logit link is appropriate because the fitted values are calculated as the antilogs of the log odds, log(*p*/*q*).

By using different link functions, the performance of a variety of models can be compared directly. The total deviance is the same in each case, and we can investigate the consequences of altering our assumptions about precisely how a given change in the linear predictor brings about a response in the fitted value of *y*. The most appropriate link function is the one which produces the minimum residual deviance.

Canonical Link Functions

The canonical link functions are the default options employed when a particular error structure is specified in the `family` directive in the model formula. Omission of a `link` directive means that the following settings are used:

Error	Canonical link
`gaussian`	identity
`poisson`	log
`binomial`	logit
`Gamma`	reciprocal

You should try to memorize these canonical links and to understand why each is appropriate to its associated error distribution. Note that only gamma errors have a capital initial letter in R.

Choosing between using a link function (e.g. log link) in a GLM and transforming the response variable (i.e. having log(*y*) as the response variable rather than *y*) and using a linear model takes a certain amount of experience.

As you read the following four chapters you will gradually become more familiar with these new concepts. Of course they are hard to understand at first, and it would be wrong to pretend otherwise. But they do become easier with practice. The key is to understand that when variance is *not* constant and when the errors are *not* normally distributed, we have to do something about it. And using a generalized linear model instead of a linear model is often the best solution. Learning about deviance, link functions and linear predictors is a small price to pay.

Akaike's Information Criterion (AIC) as a Measure of the Fit of a Model

Unexplained variation is bound to go down with every parameter added to the model. The more parameters that there are in the model, the better will be the fit. You could obtain a perfect fit if you had a separate parameter for every data point (the saturated model; p. 195), but this model would have absolutely no explanatory power. There is always going to be a trade-off between the goodness of fit and the number of parameters required by parsimony.

What we would like to do is to judge the usefulness of each parameter. A simple way of doing this is to penalize each parameter, and only allow the parameter to stay in the model if it more than pays for itself (i.e. if the unexplained variation does down by more than the penalty). When comparing two models, the smaller the AIC, the better the fit. This is the basis of automated model simplification using `step`. AIC uses a penalty of 2 per parameter, so for a given model AIC is calculated as

$$\text{AIC} = \text{deviance} + 2\text{p}$$

and when the deviance goes down by less than 2, then inclusion of the extra parameter is not justified. Other systems (like the Bayesian Information Criterion, BIC) employ stiffer penalties and so typically lead to fewer parameters being included in the minimal adequate model (e.g. BIC penalizes each parameter by log(n) where n is the number of observations).

Further Reading

Aitkin, M., Francis, B., Hinde, J. and Darnell, R. (2009) *Statistical Modelling in R*, Clarendon Press, Oxford.

McCullagh, P. and Nelder, J.A. (1989) *Generalized Linear Models*, 2nd edn, Chapman & Hall, London.

McCulloch, C.E. and Searle, S.R. (2001) *Generalized, Linear and Mixed Models*, John Wiley & Sons, New York.

13

Count Data

Up to this point, the response variables have all been continuous measurements such as weights, heights, lengths, temperatures and growth rates. A great deal of the data collected by scientists, medical statisticians and economists, however, is in the form of *counts* (whole numbers or integers). The number of individuals who died, the number of firms going bankrupt, the number of days of frost, the number of red blood cells on a microscope slide, or the number of craters in a sector of lunar landscape are all potentially interesting variables for study. With count data, the number 0 often appears as a value of the response variable (consider, for example, what a 0 would mean in the context of the examples just listed). In this chapter we deal with data on *frequencies*, where we count how many times something happened, but we have no way of knowing how often it did *not* happen (e.g. lightening strikes, bankruptcies, deaths, births). This is in contrast with count data on *proportions*, where we know the number doing a particular thing, but also the number not doing that thing (e.g. the proportion dying, sex ratios at birth, proportions of different groups responding to a questionnaire).

Straightforward linear regression methods (assuming constant variance and normal errors) are not appropriate for count data for four main reasons:

- the linear model might lead to the prediction of negative counts
- the variance of the response variable is likely to increase with the mean
- the errors will not be normally distributed
- zeros are difficult to handle in transformations

In R, count data are handled very elegantly in a generalized linear model by specifying `family=poisson` which uses Poisson errors and the log link (see Chapter 12). The log link ensures that all the fitted values are positive, while the Poisson errors take account of the fact that the data are integer and have variances that are equal to their means.

A Regression with Poisson Errors

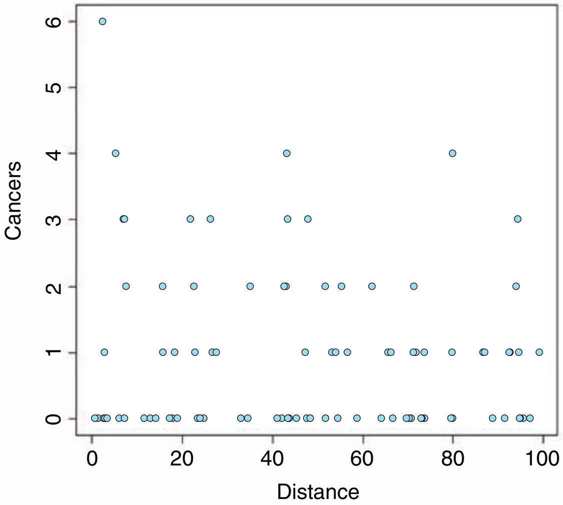
This example has a count (the number of reported cancer cases per year per clinic) as the response variable, and a single continuous explanatory variable (the distance from a nuclear plant to the clinic in kilometres). The question is whether or not proximity to the reactor affects the number of cancer cases.

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

```
clusters <- read.csv("c:\\temp\\clusters.csv")
attach(clusters)
names(clusters)
```

```
[1] "Cancers" "Distance"
```

```
plot(Distance,Cancers,pch=21,bg="lightblue")
```



As is typical of count data, the values of the response don’t cluster around a regression line, but they are dispersed in discrete rows within a roughly triangular area. As you can see there are lots of zeros all the way from distances of 0 to 100 km. The only really large count (6) was very close to the source. There seems to be a downward trend in cancer cases with distance. But is the trend significant? To find out, we do a regression of cases against distance, using a GLM with Poisson errors:

```
model1 <- glm(Cancers~Distance,poisson)
summary(model1)
```

```
Coefficients:
              Estimate   Std. Error  z value   Pr(>|z|)
(Intercept)   0.186865    0.188728    0.990    0.3221
Distance     -0.006138    0.003667   -1.674    0.0941 .

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 149.48 on 93 degrees of freedom
Residual deviance: 146.64 on 92 degrees of freedom
AIC: 262.41

Number of Fisher Scoring iterations: 5
```

The trend does not look to be significant, but we need first to look at the residual deviance. It is assumed that this is the same as the residual degrees of freedom. The fact that residual deviance is larger than residual degrees of freedom indicates that we have *overdispersion* (extra, unexplained variation in the response). We compensate for the overdispersion by refitting the model using quasipoisson rather than Poisson errors:

```
model2 <- glm(Cancers~Distance,quasipoisson)
summary(model2)
```

```
Coefficients:
              Estimate   Std.Error  t  value  Pr(>|t|)
(Intercept)   0.186865    0.235364     0.794    0.429
Distance     -0.006138    0.004573    -1.342    0.183

(Dispersion parameter for quasipoisson family taken to be 1.555271)

    Null deviance: 149.48 on 93 degrees of freedom
Residual deviance: 146.64 on 92 degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 5
```

Compensating for the overdispersion has increased the *p* value to 0.183, so there is no compelling evidence to support the existence of a trend in cancer incidence with distance from the nuclear plant. To draw the fitted model through the data, you need to understand that the GLM with Poisson errors uses the log link, so the parameter estimates and the predictions from the model (the ‘linear predictor’ above) are in logs, and need to be antilogged before the (non-significant) fitted line is drawn.

We want the *x* values to go from 0 to 100, so we can put:

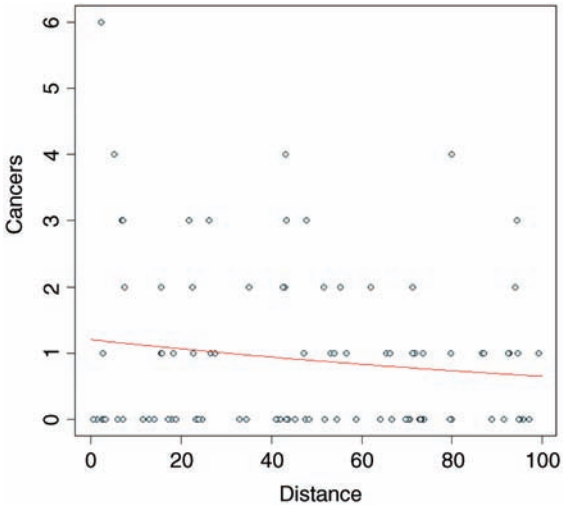
```
xv <- 0:100
```

To draw attention to the relationship between the linear predictor and the values of *y* we want to draw on our scatterplot, we shall work out the *y* values long-hand. The intercept is 0.186865 and the slope is −0.006138 (see above), so the linear predictor gives yv as:

```
yv <- 0.186865-0.006138*xv
```

The important point to bear in mind is that yv is on a logarithmic scale. We want to plot y (the raw numbers of cases, not their logarithms) so we need to take antilogs (to back-transform, in other words):

```
y <- exp(yv)
lines(xv,y,col="red")
```



The red fitted line is curved, but very shallow. The trend is not significant ($p > 0.18$). Once you have understood the relationship between the linear predictor, the link function and the predicted count data, `y`, you can speed up the curve drawing with the `predict` function, because you can specify `type="response"`, which carries out the back-transformation automatically:

```
y <- predict(model2,list(Distance=xv), type="response")
lines(xv,y,col="red")
```

Analysis of Deviance with Count Data

The response variable is a count of infected blood cells per mm^2 on microscope slides prepared from randomly selected individuals. The explanatory variables are smoker (a logical variable with values ‘yes’ or ‘no’), age (three levels: under 20, 21 to 59, 60 and over), sex (male or female) and a body mass score (three levels: normal, overweight, obese).

```
count <- read.csv("c:\\temp\\cells.csv")
attach(count)
names(count)

[1] "cells"  "smoker"  "age"     "sex"    "weight"
```

It is always a good idea with count data to get a feel for the overall frequency distribution of counts using table:

```
table(cells)

  0    1    2    3    4    5    6    7
314   75   50   32   18   13    7    2
```

Most subjects (314 of them) showed no damaged cells, and the maximum of 7 was observed in just two patients. We begin data inspection by tabulating the main effect means:

```
tapply(cells,smoker,mean)

    FALSE      TRUE
0.5478723 1.9111111

tapply(cells,weight,mean)

   normal     obese      over
0.5833333 1.2814371 0.9357143

tapply(cells,sex,mean)

   female      male
0.6584507 1.2202643

tapply(cells,age,mean)

      mid       old     young
0.8676471 0.7835821 1.2710280
```

It looks as if smokers have a substantially higher mean count than non-smokers, that overweight and obese subjects had higher counts than those of normal weight, males had a higher count that females, and young subjects had a higher mean count than middle-aged or older people. We need to test whether any of these differences are significant and to assess whether there are any interactions between the explanatory variables:

```
model1 <- glm(cells~smoker*sex*age*weight,poisson)
summary(model1)
```

You should scroll down to the bottom of the (voluminous) output to find the residual deviance and residual degrees of freedom, because we need to test for overdispersion:

```
Null deviance: 1052.95 on 510 degrees of freedom
Residual deviance: 736.33 on 477 degrees of freedom
AIC: 1318

Number of Fisher Scoring iterations: 6
```

The residual deviance (736.33) is much greater than the residual degrees of freedom (477), indicating substantial overdispersion, so before interpreting any of the effects, we should refit the model using quasipoisson errors:

```
model2 <- glm(cells~smoker*sex*age*weight,quasipoisson)
summary(model2)
```

```
Coefficients: (2 not defined because of singularities)
                                        Estimate Std. Error t value Pr(>|t|)
(Intercept)                              -0.8329     0.4307  -1.934   0.0537 .
smokerTRUE                               -0.1787     0.8057  -0.222   0.8246
sexmale                                   0.1823     0.5831   0.313   0.7547
ageold                                   -0.1830     0.5233  -0.350   0.7267
ageyoung                                  0.1398     0.6712   0.208   0.8351
weightobese                               1.2384     0.8965   1.381   0.1678
weightover                               -0.5534     1.4284  -0.387   0.6986
smokerTRUE:sexmale                        0.8293     0.9630   0.861   0.3896
smokerTRUE:ageold                        -1.7227     2.4243  -0.711   0.4777
smokerTRUE:ageyoung                       1.1232     1.0584   1.061   0.2892
sexmale:ageold                           -0.2650     0.9445  -0.281   0.7791
sexmale:ageyoung                         -0.2776     0.9879  -0.281   0.7788
smokerTRUE:weightobese                    3.5689     1.9053   1.873   0.0617 .
smokerTRUE:weightover                     2.2581     1.8524   1.219   0.2234
sexmale:weightobese                      -1.1583     1.0493  -1.104   0.2702
sexmale:weightover                        0.7985     1.5256   0.523   0.6009
ageold:weightobese                       -0.9280     0.9687  -0.958   0.3386
ageyoung:weightobese                     -1.2384     1.7098  -0.724   0.4693
ageold:weightover                         1.0013     1.4776   0.678   0.4983
ageyoung:weightover                       0.5534     1.7980   0.308   0.7584
smokerTRUE:sexmale:ageold                 1.8342     2.1827   0.840   0.4011
smokerTRUE:sexmale:ageyoung              -0.8249     1.3558  -0.608   0.5432
smokerTRUE:sexmale:weightobese           -2.2379     1.7788  -1.258   0.2090
smokerTRUE:sexmale:weightover            -2.5033     2.1120  -1.185   0.2365
smokerTRUE:ageold:weightobese             0.8298     3.3269   0.249   0.8031
smokerTRUE:ageyoung:weightobese          -2.2108     1.0865  -2.035   0.0424 *
smokerTRUE:ageold:weightover              1.1275     1.6897   0.667   0.5049
smokerTRUE:ageyoung:weightover           -1.6156     2.2168  -0.729   0.4665
sexmale:ageold:weightobese                2.2210     1.3318   1.668   0.0960 .
sexmale:ageyoung:weightobese              2.5346     1.9488   1.301   0.1940
sexmale:ageold:weightover                -1.0641     1.9650  -0.542   0.5884
sexmale:ageyoung:weightover              -1.1087     2.1234  -0.522   0.6018
smokerTRUE:sexmale:ageold:weightobese    -1.6169     3.0561  -0.529   0.5970
smokerTRUE:sexmale:ageyoung:weightobese       NA         NA      NA       NA
smokerTRUE:sexmale:ageold:weightover          NA         NA      NA       NA
smokerTRUE:sexmale:ageyoung:weightover    2.4160     2.6846   0.900   0.3686

(Dispersion parameter for quasipoisson family taken to be 1.854815)

    Null deviance: 1052.95 on 510 degrees of freedom
Residual deviance:  736.33 on 477 degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 6
```

The first thing to notice is that there are NA (missing value) symbols in the table of the linear predictor (the message reads Coefficients: (2 not defined because of singularities). This is the first example we have met of aliasing

(p. 16): these symbols indicate that there are no data in the dataframe from which to estimate two of the terms in the four-way interaction between smoking, sex, age and weight.

It does look as if there might be three-way interaction between smoking, age and obesity ($p = 0.0424$). With a complicated model like this, it is a good idea to speed up the early stages of model simplification by using the `step` function, but this is not available with `quasipoisson` errors, so we need to work through the analysis longland. Start by removing the aliased four-way interaction, then try removing what looks (from the p values) to be the least significant three-way interaction, the sex–age–weight interaction:

```
model3 <- update(model2, ~. -smoker:sex:age:weight)
model4 <- update(model3, ~. -sex:age:weight)
anova(model4,model3,test="F")
```

```
Analysis of Deviance Table

Resid. Df  Resid. Dev  Df  Deviance      F Pr(>F)
1       483     745.31
2       479     737.87    4   7.4416  1.0067 0.4035
```

That model simplification was not significant ($p = 0.4035$) so we leave it out and try deleting the next least significant interaction

```
model5 <- update(model4, ~. -smoker:sex:age)
anova(model5,model4,test="F")
```

Again, that was not significant, so we leave it out.

```
model6 <- update(model5, ~. -smoker:age:weight)
anova(model6,model5,test="F")
```

Despite one-star significance for one of the interaction terms, this was not significant either, so we leave it out.

```
model7 <- update(model6, ~. -smoker:sex:weight)
anova(model7,model6,test="F")
```

That is the last of the three-way interactions, so we can start removing the two-way interactions, starting, as usual, with the least significant:

```
model8 <- update(model7, ~. -smoker:age)
anova(model8,model7,test="F")
```

Not significant. Next:

```
model9 <- update(model8, ~. -sex:weight)
anova(model9,model8,test="F")
```

Not significant. Next:

```
model10 <- update(model9, ~. -age:weight)
anova(model10,model9,test="F")
```

Not significant. Next:

```
model11 <- update(model10, ~. -smoker:sex)
anova(model11,model10,test="F")
```

Not significant. Next:

```
model12 <- update(model11, ~. -sex:age)
anova(model12,model11,test="F")
```

```
Analysis of Deviance Table

  Resid. Df Resid. Dev Df Deviance       F  Pr(>F)
1       502     791.59
2       500     778.69  2   12.899  3.4805 0.03154 *
```

That is significant so the sex–age interaction needs to stay int the model. What about the smoker–weight interaction? We need to compare it with `model11` (not `model12`):

```
model13 <- update(model11, ~. -smoker:weight)
anova(model13,model11,test="F")
```

```
Analysis of Deviance Table

  Resid. Df  Resid. Dev  Df  Deviance       F    Pr(>F)
1       502      790.08
2       500      778.69   2    11.395  3.0747   0.04708 *
```

That is significant too, so we retain it. It appears that `model11` is the minimal adequate model. There are two-way interactions between smoking and weight and between sex and age, so all four of the explanatory variables need to remain in the model as main effects (you must not delete main effects from a model, even if they look to be non-significant (like sex and age in this case) when they appear in significant interaction terms. The biggest main effect (to judge by the p values) is smoking. The significance of the intercept is not interesting in this case (it just says the mean number of cells for non-smoking, middle-aged females of normal weight is greater than zero – but it is a count, so it would be).

```
Coefficients:
                      Estimate  Std. Error  t  value   Pr(>|t|)
(Intercept)           -1.09888     0.33330   -3.297   0.00105  **
smokerTRUE             0.79483     0.26062    3.050   0.00241  **
sexmale                0.32917     0.34468    0.955   0.34004
ageold                 0.12274     0.34991    0.351   0.72590
ageyoung               0.54004     0.36558    1.477   0.14025
weightobese            0.49447     0.23376    2.115   0.03490   *
```

```
weightover               0.28517    0.25790    1.106   0.26937
sexmale:ageold           0.06898    0.40297    0.171   0.86414
sexmale:ageyoung        -0.75914    0.41819   -1.815   0.07007  .
smokerTRUE:weightobese   0.75913    0.31421    2.416   0.01605  *
smokerTRUE:weightover    0.32172    0.35561    0.905   0.36606

(Dispersion parameter for quasipoisson family taken to be 1.853039)

    Null deviance: 1052.95 on 510 degrees of freedom
Residual deviance:  778.69 on 500 degrees of freedom
```

This model shows a significant interaction between smoking and weight in determining the number of damaged cells ($p = 0.05$):

```
tapply(cells,list(smoker,weight),mean)

         normal     obese      over
FALSE 0.4184397  0.689394 0.5436893
TRUE  0.9523810  3.514286 2.0270270
```

and an interaction between sex and age ($p = 0.03$):

```
tapply(cells,list(sex,age),mean)

             mid         old     young
female 0.4878049  0.5441176  1.435897
male   1.0315789  1.5468750  1.176471
```

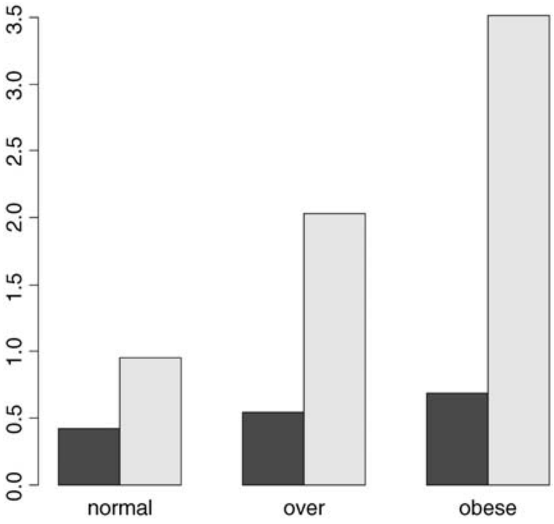
The summary table shows that the gender effect is much smaller for young people (where the count was slightly higher in females) than for middle-aged or older people (where the count was much higher in males).

With complicated interactions like this, it is often useful to produce graphs of the effects. The relationship between smoking and weight is shown like this:

```
barplot(tapply(cells,list(smoker,weight),mean),beside=T)
```

This is OK, but the bars are not in the best order; the obese category should be on the right of the figure. To correct this we need to change the order of the factor levels for `weight` away from the default (alphabetical order) to a user-specified order, with `normal` of the left and `obese` on the right. We do this using the function called `factor`:

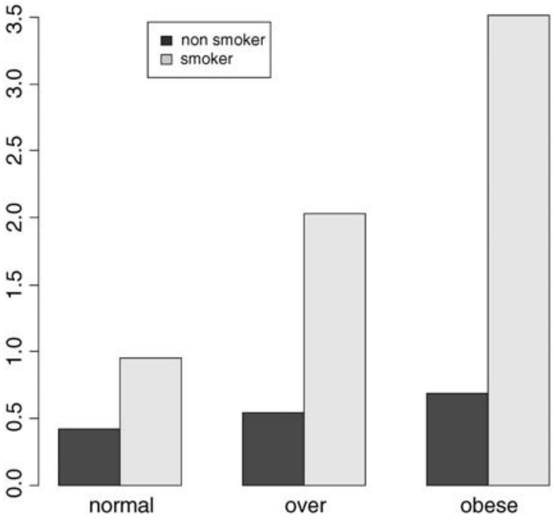
```
weight <- factor(weight,c("normal","over","obese"))
barplot(tapply(cells,list(smoker,weight),mean),beside=T)
```



That’s more like it. Finally, we need to add a legend showing which bars are the smokers:

```
barplot(tapply(cells,list(smoker,weight),mean),beside=T)
legend(locator(1),c("non smoker","smoker"),fill=gray(c(0.2,0.8)))
```

Just click when the cursor is over the spot on the graph where you want the top left-hand corner of the legend to appear:



The Danger of Contingency Tables

We have already dealt with simple contingency tables and their analysis using Fisher's exact test or Pearson's chi-squared test (see Chapter 6). But there is an important further issue to be dealt with. In observational studies we quantify only a limited number of explanatory variables. It is inevitable that we shall fail to measure a number of factors that have an important influence on the behaviour of the system in question. That's life, and given that we make every effort to note the important factors, there's little we can do about it. The problem comes when we ignore factors that have an important influence on the response variable. This difficulty can be particularly acute if we *aggregate data over important explanatory variables*. An example should make this clear.

Suppose we are carrying out a study of induced defences in trees. A preliminary trial has suggested that early feeding on a leaf by aphids may cause chemical changes in the leaf which reduce the probability of that leaf being attacked later in the season by hole-making caterpillars. To this end we mark a large cohort of leaves, then score whether they were infested by aphids early in the season and whether they were holed by insects later in the year. The work was carried out on two different trees and the results were as follows:

Tree	Aphids	Holed	Intact	Total leaves	Proportion holed
Tree1	Without	35	1750	1785	0.0196
	With	23	1146	1169	0.0197
Tree2	Without	146	1642	1788	0.0817
	With	30	333	363	0.0826

There are four variables: the response variable, count, with eight values (highlighted above), a two-level factor for late season feeding by caterpillars (holed or intact), a two-level factor for early season aphid feeding (with or without aphids) and a two-level factor for tree (the observations come from two separate trees, imaginatively named Tree1 and Tree2).

```
induced <- read.csv("c:\\temp\\induced.csv")
attach(induced)
names(induced)
```

```
[1] "Tree"    "Aphid"    "Caterpillar"   "Count"
```

We begin by fitting what is known as a *saturated model*. This is a curious thing, which has as many parameters as there are values of the response variable. The fit of the model is perfect, so there are no residual degrees of freedom and no residual deviance. The reason why we fit a saturated model is that it is always the best place to start modelling complex contingency tables. If we fit the saturated model, then there is no risk that we inadvertently leave out important interactions between the so-called ‘nuisance variables’. These are the parameters that need to be in the model to ensure that the marginal totals are properly constrained.

```
model <- glm(Count~Tree*Aphid*Caterpillar,family=poisson)
```

The asterisk notation ensures that the saturated model is fitted, because all of the main effects and two-way interactions are fitted, along with the three-way, tree–aphid–caterpillar interaction. The model fit involves the estimation of $2\times2\times2=8$ parameters, and exactly matches the eight values of the response variable, *Count*. There is no point looking at the saturated model in any detail, because the reams of information it contains are all superfluous. The first real step in the modelling is to use `update` to remove the three-way interaction from the saturated model, and then to use `anova` to test whether the three-way interaction is significant or not:

```
model2 <- update(model, ~ . - Tree:Aphid:Caterpillar)
```

The punctuation here is very important (it is comma, tilde, dot, minus) and note the use of colons rather than asterisks to denote interaction terms rather than main effects plus interaction terms. Now we can see whether the three-way interaction was significant by specifying `test="Chi"` like this:

```
anova(model,model2,test="Chi")

Analysis of Deviance Table

Model 1: Count ~ Tree * Aphid * Caterpillar
Model 2: Count ~ Tree + Aphid + Caterpillar + Tree:Aphid +
Tree:Caterpillar + Aphid:Caterpillar

  Resid. Df  Resid. Dev  Df     Deviance  Pr(>Chi)
1         0  0.00000000
2         1  0.00079137  -1  -0.00079137    0.9776
```

This shows clearly that the interaction between caterpillar attack and leaf holing does not differ from tree to tree ($p=0.9776$). Note that if this interaction had been significant, then we would have stopped the modelling at this stage. But it wasn't, so we leave it out and continue. What about the main question? Is there an interaction between caterpillar attack and leaf holing? To test this we delete the aphid–caterpillar interaction from the model, and assess the results using `anova`:

```
model3 <- update(model2, ~ . - Aphid:Caterpillar)
anova(model3,model2,test="Chi")

Analysis of Deviance Table

Model 1: Count ~ Tree + Aphid + Caterpillar + Tree:Aphid +
Tree:Caterpillar
Model 2: Count ~ Tree + Aphid + Caterpillar + Tree:Aphid +
Tree:Caterpillar +  Aphid:Caterpillar

  Resid. Df  Resid. Dev  Df Deviance  Pr(>Chi)
1         2  0.0040853
2         1  0.0007914   1 0.003294    0.9542
```

There is absolutely no hint of an interaction ($p = 0.954$). The interpretation is clear: this work provides *no* evidence for induced defences caused by early season caterpillar feeding.

But look what happens when we do the modelling the *wrong* way. Suppose we went straight for the aphid–caterpillar interaction of interest. We might proceed like this:

```
wrong <- glm(Count~Aphid*Caterpillar,family=poisson)
wrong1 <- update(wrong,~. - Aphid:Caterpillar)
anova(wrong,wrong1,test="Chi")
```

```
Analysis of Deviance Table

Model 1: Count ~ Aphid * Caterpillar
Model 2: Count ~ Aphid + Caterpillar

  Resid. Df  Resid. Dev  Df  Deviance  Pr(>Chi)
1         4     550.19
2         5     556.85  -1   -6.6594 0.009864 **
```

The aphid–caterpillar interaction is highly significant ($p < 0.01$) providing strong evidence for induced defences. Wrong! By failing to include the tree variable in the model we have omitted an important explanatory variable. As it turns out, and we should really have determined by more thorough preliminary analysis, the trees differ enormously in their average levels of leaf holing:

```
tapply(Count,list(Tree,Caterpillar),sum)
```

```
        holed   not
Tree1      58  2896
Tree2     176  1975
```

The proportion of leaves with holes on Tree1 was $58/(58 + 2896) = 0.0196$, but on Tree2 was $176/(176 + 1975) = 0.0818$. Tree2 has more than four times the proportion of its leaves holed by caterpillars. If we had been paying more attention when we did the modelling the wrong way, we should have noticed that the model containing only aphid and caterpillar had massive overdispersion, and this should have alerted us that all was not well.

The moral is simple and clear. Always fit a saturated model first, containing all the variables of interest and all the interactions involving the nuisance variables (tree in this case). Only delete from the model those interactions that involve the variables of interest (aphid and caterpillar in this case). Main effects are meaningless in contingency tables, as are the model summaries. Always test for overdispersion. It will never be a problem if you follow the advice of simplifying down from a saturated model, because you only ever leave out non-significant terms, and you never delete terms involving any of the nuisance variables, except for the highest-order interaction at the first step in model simplification (Tree:Aphid:Caterpillar in this example).

Analysis of Covariance with Count Data

In this example the response is a count of the number of plant species on plots that have different biomass (a continuous explanatory variable) and different soil pH (a categorical variable with three levels: high, mid and low):

```
species <- read.csv("c:\\temp\\species.csv")
attach(species)
names(species)

[1] "pH"   "Biomass" "Species"
```

We start by plotting the data, using different colours for each of the three pH classes:

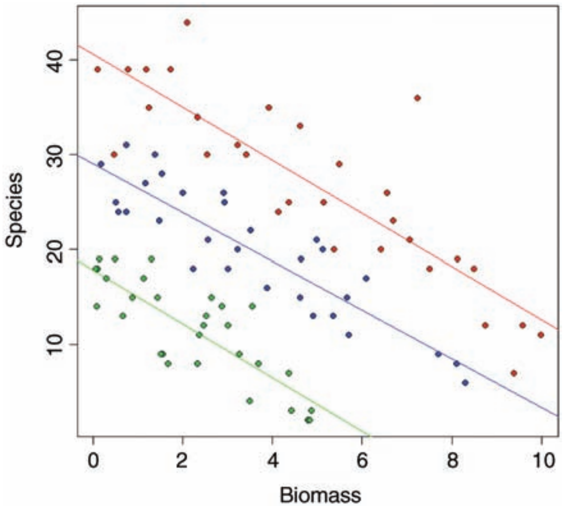
```
plot(Biomass,Species,pch=21,bg=(1+as.numeric(pH)))
```

Now we fit a straightforward analysis of covariance and use `abline` to draw lines of the appropriate colour through the scatterplot:

```
model <- lm(Species~Biomass*pH)
summary(model)

Call:
lm(formula = Species ~ Biomass * pH)

Residuals:
   Min      1Q  Median     3Q     Max
-9.290  -2.554  -0.124  2.208  15.677

Coefficients:
               Estimate Std. Error t value    Pr(>|t|)
(Intercept)    40.60407    1.36701  29.703    < 2e-16 ***
Biomass        -2.80045    0.23856 -11.739    < 2e-16 ***
pHlow         -22.75667    1.83564 -12.397    < 2e-16 ***
pHmid         -11.57307    1.86926  -6.191   2.1e-08 ***
Biomass:pHlow  -0.02733    0.51248  -0.053     0.958
Biomass:pHmid   0.23535    0.38579   0.610     0.543

Residual standard error: 3.818 on 84 degrees of freedom
Multiple R-squared: 0.8531,  Adjusted R-squared: 0.8444
F-statistic: 97.58 on 5 and 84 DF, p-value: < 2.2e-16
```

There is no evidence in the linear model for any difference between the slopes at different levels of pH. Make sure you understand how I got the intercepts and slopes for `abline` out of the model summary:

```
abline(40.60407,-2.80045,col="red")
abline(40.60407-22.75667,-2.80045-0.02733,col="green")
abline(40.60407-11.57307,-2.80045+0.23535,col="blue")
```



It is clear that species declines with biomass, and that soil pH has a big effect on species, but does the slope of the relationship between species and biomass depend on pH? The biggest problem with this linear model is that it predicts negative values for species richness above biomass values of 6 for the low pH plots. Count data are strictly bounded below, and our model should really take account of this.

Let us refit the model using a GLM with Poisson errors instead of a linear model:

```
model <- glm(Species~Biomass*pH,poisson)
summary(model)
```

```
Coefficients:
                Estimate   Std. Error  z value      Pr(>|z|)
(Intercept)      3.76812     0.06153   61.240      < 2e-16      ***
Biomass         -0.10713     0.01249   -8.577      < 2e-16      ***
pHlow           -0.81557     0.10284   -7.931     2.18e-15      ***
pHmid           -0.33146     0.09217   -3.596     0.000323      ***
Biomass:pHlow   -0.15503     0.04003   -3.873     0.000108      ***
Biomass:pHmid   -0.03189     0.02308   -1.382     0.166954

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 452.346 on 89 degrees of freedom
Residual deviance:  83.201 on 84 degrees of freedom
AIC: 514.39

Number of Fisher Scoring iterations: 4
```

The residual deviance is not larger than the residual degrees of freedom, so we don’t need to correct for overdispersion. Do we need to retain the interaction term? We test this by deletion:

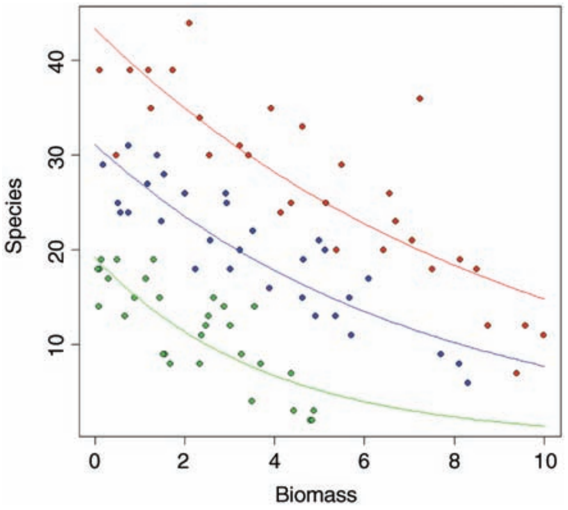
```
model2 <- glm(Species~Biomass+pH,poisson)
anova(model,model2,test="Chi")
```

```
Analysis of Deviance Table

Model 1: Species ~ Biomass * pH
Model 2: Species ~ Biomass + pH

  Resid. Df Resid. Dev  Df  Deviance    Pr(>Chi)
1        84      83.201
2        86      99.242  -2    -16.04  0.0003288 ***
```

Yes, we do. There is a highly significant difference between the slopes at different levels of pH. So our first model is minimal adequate.

Finally, we draw the fitted curves through the scatterplot, using `predict`. The trick is that we need to draw three separate curves; one for each level of soil pH.

```
plot(Biomass,Species,pch=21,bg=(1+as.numeric(pH)))
xv <- seq(0,10,0.1)
length(xv)

[1] 101

acidity <- rep("low",101)

yv <- predict(model,list(Biomass=xv,pH=acidity),type="response")
lines(xv,yv,col="green")

acidity <- rep("mid",101)

yv <- predict(model,list(Biomass=xv,pH=acidity),type="response")
lines(xv,yv,col="blue")

acidity <- rep("high",101)

yv <- predict(model,list(Biomass=xv,pH=acidity),type="response")
lines(xv,yv,col="red")
```



Note the use of `type="response"` in the `predict` function. This ensures that the response variable is calculated as species rather than log(species), and means we do not need to back-transform using antilogs before drawing the lines. You could make the R code more elegant by writing a function to plot any number of lines, depending on the number of levels of the factor (three levels of pH in this case).

Frequency Distributions

Given data on the numbers of bankruptcies in 80 districts, we want to know whether there is any evidence that some districts show greater than expected numbers of cases.

What would we expect? Of course we should expect some variation. But how much, exactly? Well, that depends on our model of the process. Perhaps the simplest model is that absolutely nothing is going on, and that every single bankruptcy case is absolutely independent of every other. That leads to the prediction that the numbers of cases per district will follow a Poisson process: a distribution in which the variance is equal to the mean (Box 13.1).

Box 13.1. The Poisson distribution

The Poisson distribution is widely used for the description of count data. We know how many times something happened (e.g. kicks from cavalry horses, lightening strikes, bomb hits), but we have no way of knowing how many times it did not happen. The Poisson is a one-parameter distribution, specified entirely by the mean. The variance is identical to the mean (λ), so the variance–mean ratio is equal to 1. The probability of observing a count of x is given by

$$P(x) = \frac{e^{-\lambda}\lambda^x}{x!}$$

This can be calculated very simply on a hand calculator because:

$$P(x) = P(x-1)\frac{\lambda}{x}$$

which means that if you start with the *zero term*

$$P(0) = e^{-\lambda}$$

then each successive probability is obtained simply by multiplying by the mean and dividing by x.

Let us see what the data show:

```
case.book <- read.csv("c:\\temp\\cases.csv")
attach(case.book)
names(case.book)
```

```
[1] "cases"
```

First we need to count the numbers of districts with no cases, one case, two cases, and so on. The R function that does this is called `table`:

```
frequencies <- table(cases)
frequencies
```

```
cases
 0   1   2  3  4  5  6  7  8  9 10
34  14  10  7  4  5  2  1  1  1  1
```

There were no cases at all in 34 districts, but one district had 10 cases. A good way to proceed is to compare our distribution (called frequencies) with the distribution that would be observed if the data really did come from a Poisson distribution as postulated by our model. We can use the R function `dpois` to compute the probability density of each of the 11 frequencies from 0 to 10 (we multiply the probability produced by `dpois` by the total sample of 80 to obtain the predicted frequencies). We need to calculate the mean number of cases per district – this is the Poisson distribution's only parameter:

```
mean(cases)
```

```
[1] 1.775
```

The plan is to draw two distributions side by side, so we set up the plotting region:

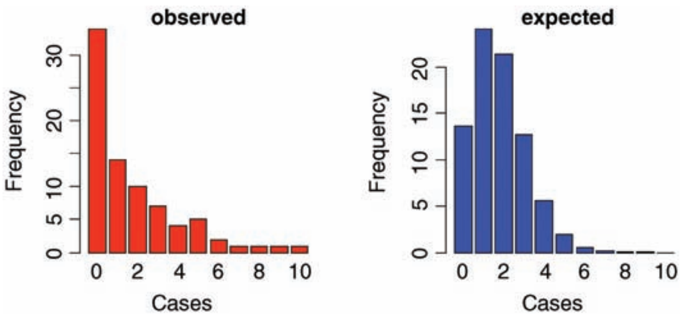
```
windows(7,4)
par(mfrow=c(1,2))
```

Now we plot the observed frequencies in the left-hand panel:

```
barplot(frequencies,ylab="Frequency",xlab="Cases",
col="red",main="observed")
```

and the predicted, Poisson frequencies in the right-hand panel:

```
barplot(dpois(0:10,1.775)*80,names=as.character(0:10),
ylab="Frequency",xlab="Cases",col="blue",main="expected")
```



The distributions are very different: the mode of the observed data is zero, but the mode of the Poisson with the same mean is 1; the observed data contained examples of 8, 9 and 10 cases, but these would be highly unlikely under a Poisson process. We would say that the observed data are highly *aggregated*; they have a variance–mean ratio of nearly 3 (the Poisson, of course, has a variance–mean ratio of 1):

```
var(cases)/mean(cases)
```

```
[1]  2.99483
```

So, if the data are not Poisson distributed, how are they distributed? A good candidate distribution where the variance–mean ratio is this big (c. 3.0) is the negative binomial distribution (Box 13.2).

Box 13.2. The negative binomial distribution

This discrete, two-parameter distribution is useful for describing the distribution of count data, where the variance is often much greater than the mean. The two parameters are the mean μ and the clumping parameter k. The smaller the value of k, the greater the degree of clumping. The density function is

$$p(x) = \left(1 + \frac{\mu}{k}\right)^{-k} \frac{\Gamma(k + x)}{x!\Gamma(k)} \left(\frac{\mu}{\mu + k}\right)^{x}$$

where Γ is the gamma function. The zero term is found by setting $x = 0$ and simplifying:

$$p(0) = \left(1 + \frac{\mu}{k}\right)^{-k}$$

Successive terms in the distribution can be computed iteratively from

$$p(x) = p(x - 1)\left(\frac{k + x - 1}{x}\right)\left(\frac{\mu}{\mu + k}\right)$$

An initial estimate of the value of k can be obtained from the sample mean and variance:

$$k \approx \frac{\mu^2}{s^2 - \mu}$$

Since k cannot be negative, it is clear that the negative binomial distribution should not be fitted to data where the variance is less than the mean (use a binomial distribution instead). The precise maximum likelihood estimate of k is found numerically, by iterating progressively more fine-tuned values of k until the left- and right-hand sides of the following equation are equal:

$$n \ln\left(1 + \frac{\mu}{k}\right) = \sum_{x=0}^{\max}\left(\frac{A_{(x)}}{k+x}\right)$$

where the vector $A(x)$ contains the total frequency of values *greater* than x.

You could create a function to work out the probability densities like this:

```
negbin <- function(x,u,k)
    (1+u/k)^(-k)*(u/(u+k))^x*gamma(k+x)/(factorial(x)*gamma(k))
```

then use the function to produce a barplot of probability densities for a range of x values (say 0 to 10, for a distribution with specified mean and aggregation parameter (say $\mu = 0.8, k = 0.2$) like this:

```
xf <- numeric(11)
for (i in 0:10) xf[i+1] <- negbin(i,0.8,0.2)
barplot(xf)
```

As described in Box 13.2, this a two-parameter distribution. We have already worked out the mean number of cases, which is 1.775. We need an estimate of the clumping parameter, k, to establish the degree of aggregation in the data (a small value of k (such as $k < 1$) would show high aggregation, while a large value (such as $k > 5$) would indicate randomness). We can get an approximate estimate of magnitude using the formula in Box 13.2:

$$k = \frac{\mu^2}{s^2 - \mu}$$

We have:

```
mean(cases)^2/(var(cases)-mean(cases))
```

```
[1] 0.8898003
```

so we shall work with $k = 0.8898$.

How do we compute the expected frequencies? The density function for the negative binomial distribution is `dnbinom` and it has three arguments: the frequency for which we want the probability (in our case 0 to 10), the clumping parameter (`size` = 0.8898), and the mean number of cases (`mu` = 1.775); we multiply by the total number of cases (80) to obtain the expected frequencies

```
expected <- dnbinom(0:10,size=0.8898,mu=1.775)*80
```

The plan is to draw a single figure in which the observed and expected frequencies are drawn side by side. The trick is to produce a new vector (called `both`) which is twice as long as the observed and expected frequency vectors ($2 \times 11 = 22$). Then we put the observed frequencies in the odd numbered elements (using modulo 2 to calculate the values of the subscripts), and the expected frequencies in the even numbered elements:

```
both <- numeric(22)
both[1:22 %% 2 != 0] <- frequencies
both[1:22 %% 2 == 0] <- expected
```

On the *x* axis, we intend to label only every other bar:

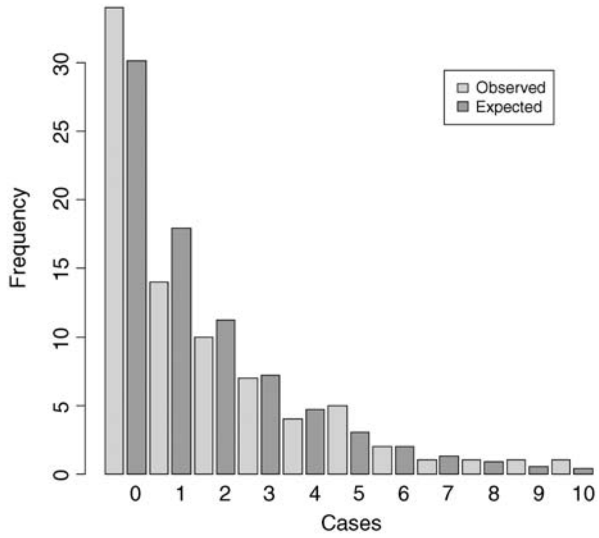
```
labels <- character(22)
labels[1:22 %% 2 == 0] <- as.character(0:10)
```

Now we can produce the barplot, using light grey for the observed frequencies and dark grey for the negative binomial frequencies:

```
barplot(both,col=rep(c("lightgray","darkgray"),11),names=labels,
ylab="Frequency",xlab="Cases")
```

We need to add a legend to show what the two colours of the bars mean. You can locate the legend by trial and error, or by left-clicking mouse when the cursor is in the correct position, using the `locator(1)` function (see p. 169):

```
legend(locator(1),c("Observed","Expected"),
fill=c("lightgray","darkgray"))
```



The fit to the negative binomial distribution is much better than it was with the Poisson distribution, especially in the right-hand tail. But the observed data have too many zeros and too few ones to be represented perfectly by a negative binomial distribution. If you want to quantify the lack of fit between the observed and expected frequency distributions, you can calculate Pearson's chi square $\sum (O-E)^2/E$ based on the number of comparisons that have expected frequency greater than 4.

```
expected
```

```
[1] 30.1449097 17.8665264 11.2450066  7.2150606  4.6734866 3.0443588
[7]  1.9905765  1.3050321  0.8572962  0.5640455  0.3715655
```

If we accumulate the rightmost six frequencies, then all the values of `expected` will be bigger than 4. The degrees of freedom are then given by the number of comparisons (6) minus the number of parameters estimated from the data (2 in our case; the mean and k) minus 1 for contingency (because the total frequency must add up to 80), so there are 3 degrees of freedom. We use 'levels gets' to reduce the lengths of the observed and expected vectors, creating an upper interval called '5+' for '5 or more':

```
cs <- factor(0:10)
levels(cs)[6:11] <- "5+"
levels(cs)
```

```
[1] "0" "1" "2" "3" "4" "5+"
```

Now make the two shorter vectors `of` and `ef` (for observed and expected frequencies):

```
ef <- as.vector(tapply(expected,cs,sum))
of <- as.vector(tapply(frequencies,cs,sum))
```

Finally, we can compute the chi-squared value measuring the difference between the observed and expected frequency distributions, and use `1-pchisq` to work out the p value:

```
sum((of-ef)^2/ef)
```

```
[1] 2.581842
```

```
1 - pchisq(2.581842,3)
```

```
[1] 0.4606818
```

We conclude that a negative binomial description of these data is reasonable (the observed and expected distributions are not significantly different; $p = 0.46$).

Further Reading

Agresti, A. (1990) *Categorical Data Analysis*, John Wiley & Sons, New York.

Santer, T.J. and Duffy, D.E. (1990) *The Statistical Analysis of Discrete Data*, Springer-Verlag, New York.

14

Proportion Data

An important class of problems involves data on proportions:

- studies on percentage mortality
- infection rates of diseases
- proportion responding to clinical treatment
- proportion admitting to particular voting intentions
- sex ratios
- data on proportional response to an experimental treatment

These are count data, but what they have in common is that we know how many of the experimental objects are in one category (dead, insolvent, male or infected) and we also know how many are in another (alive, solvent, female or uninfected). This differs from Poisson count data, where we knew how many times an event occurred, but *not* how many times it did not occur (Chapter 13).

We model processes involving proportional response variables in R by specifying a generalized linear model (GLM) with `family=binomial`. The only complication is that whereas with Poisson errors we could simply say `family=poisson`, with binomial errors we must specify the number of failures as well as the numbers of successes by creating a two-vector response variable. To do this we bind together two vectors using `cbind` into a single object, `y`, comprising the numbers of successes and the number of failures. The *binomial denominator*, *n*, is the total sample, and

```
number.of.failures <- binomial.denominator – number.of.successes
y <- cbind(number.of.successes, number.of.failures)
```

The old-fashioned way of modelling this sort of data was to use the percentage mortality as the response variable. There are four problems with this:

- the errors are not normally distributed
- the variance is not constant

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

- the response is bounded (by 100 above and by 0 below)
- by calculating the percentage, we lose information of the size of the sample, n, from which the proportion was estimated

By using a GLM, we cause R to carry out a weighted regression, with the sample sizes, `n`, as weights, and employing the logit link function to ensure linearity. There are some kinds of proportion data, such as *percentage cover*, that are best analysed using conventional linear models (with normal errors and constant variance) following *arcsine transformation*. In such cases, the response variable, measured in radians, is $\sin^{-1}\sqrt{0.01 \times p}$, where p is percentage cover. If, however, the response variable takes the form of a *percentage change* in some continuous measurement (such as the percentage change in weight on receiving a particular diet), then rather than arcsine transform the data, it is usually better treated by either

- analysis of covariance (see Chapter 9), using final weight as the response variable and initial weight as a covariate, or
- by specifying the response variable as a relative growth rate, measured as log(final weight/initial weight)

both of which can then be analysed with normal errors without further transformation.

Analyses of Data on One and Two Proportions

For comparisons of one binomial proportion with a constant, use `binom.test` (see p. 98). For comparison of two samples of proportion data, use `prop.test` (see p. 100). The methods of this chapter are required only for more complex models of proportion data, including regression and contingency tables, where GLMs are used.

Averages of Proportions

A classic beginner's mistake is to average proportions as you would with any other kind of real number. We have four estimates of the sex ratio: 0.2, 0.17, 0.2 and 0.53. They add up to 1.1, so the mean is $1.1/4 = 0.275$. Wrong! This is *not* how you average proportions. These proportions are based on count data and we should use the counts to determine the average. For proportion data, the mean success rate is given by:

$$\text{mean proportion} = \frac{\text{total number of successes}}{\text{total number of attempts}}$$

We need to go back and look at the counts on which our four proportions were calculated. It turns out they were 1 out of 5, 1 out of 6, 2 out of 10 and 53 out of 100, respectively. The total number of successes was $1+1+2+53=57$ and the total number of attempts was $5+6+10+100=121$. The correct mean proportion is $57/121=0.4711$. This is nearly double the answer we got by doing it the wrong way. So beware.

Count Data on Proportions

The traditional transformations of proportion data were arcsine and probit. The arcsine transformation took care of the error distribution, while the probit transformation was used to linearize the relationship between percentage mortality and log dose in a bioassay. There is nothing wrong with these transformations, and they are available within R, but a simpler approach is often preferable, and is likely to produce a model that is easier to interpret.

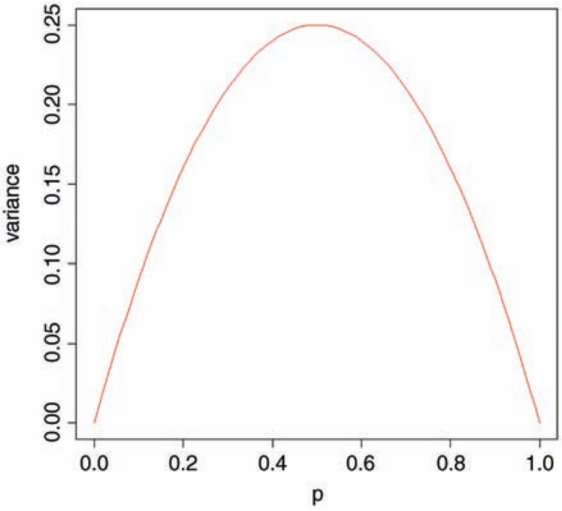
The major difficulty with modelling proportion data is that the responses are *strictly bounded*. There is no way that the percentage dying can be greater than 100% or less than 0%. But if we use simple techniques such as linear regression or analysis of covariance, then the fitted model could quite easily predict negative values or values greater than 100%, especially if the variance was high and many of the data were close to 0 or close to 100%.

The *logistic* curve is commonly used to describe data on proportions, because, unlike the straight line model, it asymptotes at 0 and 1 so that negative proportions and responses of more than 100% cannot be predicted. Throughout this discussion we shall use p to describe the proportion of individuals observed to respond in a given way. Because much of their jargon was derived from the theory of gambling, statisticians call these *successes*, although to a demographer measuring death rates this may seem a little eccentric. The individuals that respond in other ways (the statistician's *failures*) are therefore $1-p$ and we shall call the proportion of failures q (so that $p+q=1$). The third variable is the size of the sample, n, from which p was estimated (it is the binomial denominator, and the statistician's *number of attempts*).

An important point about the binomial distribution is that the variance is not constant. In fact, the variance of a binomial distribution with mean np is:

$$s^2 = npq$$

so that the variance changes with the mean like this:



The variance is low when p is very high or very low (when all or most of the outcomes are identical), and the variance is greatest when $p=q=0.5$. As p gets smaller, so the binomial distribution gets closer and closer to the Poisson distribution. You can see why this is so by considering the formula for the variance of the binomial (above). Remember that for the Poisson, the variance is equal to the mean, $s^2 = np$. Now, as p gets smaller, so q gets closer and closer to 1, so the variance of the binomial converges to the mean:

$$s^2 = npq \approx np \quad (q \approx 1)$$

Odds

The logistic model for p as a function of x looks like this:

$$p = \frac{e^{a+bx}}{1 + e^{a+bx}}$$

and there are no prizes for realizing that the model is not linear. When confronted with a new equation like this, it is a good idea to work out its *behaviour at the limits*. This involves asking what the value of y is when $x = 0$, and what the value of y is when x is very large (say, $x =$ infinity)?

When $x = 0$ then $p = \exp(a)/(1 + \exp(a))$, so this is the intercept. There are two results that you should memorize: $\exp(\infty) = \infty$ and $\exp(-\infty) = 1/\exp(\infty) = 0$. So when x is a very large positive number, we have $p = 1$ and when x is a very large negative number, we have $p = 0/1 = 0$, so the model is strictly bounded.

The trick of linearizing the logistic turns out to involve a very simple transformation. You may have come across the way in which bookmakers specify probabilities by quoting the *odds* against a particular horse winning a race (they might give odds of 2 to 1 on a reasonably good horse or 25 to 1 on an outsider). What they mean by 2 to 1 is that if there were three races, then our horse would be expected to lose two of them and win one of them (with a built-in margin of error in favour of the bookmaker, needless to say).

This is a rather different way of presenting information on probabilities than scientists are used to dealing with. Thus, where the scientist might state a proportion as 0.3333 (1 success out of 3), the bookmaker would give odds of 2 to 1. In symbols, this is the difference between the scientist stating the probability p, and the bookmaker stating the odds p/q. Now if we take the *odds* p/q and substitute this into the formula for the logistic, we get:

$$\frac{p}{q} = \frac{e^{a+bx}}{1 + e^{a+bx}} \left[1 - \frac{e^{a+bx}}{1 + e^{a+bx}}\right]^{-1}$$

which looks awful. But a little algebra shows that:

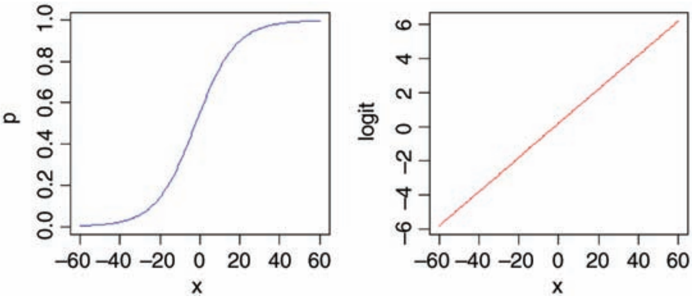
$$\frac{p}{q} = \frac{e^{a+bx}}{1 + e^{a+bx}} \left[\frac{1}{1 + e^{a+bx}}\right]^{-1} = e^{a+bx}$$

Now, taking natural logs and recalling that $\ln(e^x) = x$ will simplify matters even further, so that

$$\ln\left(\frac{p}{q}\right) = a + bx$$

This gives a *linear predictor*, $a+bx$, not for p but for the *logit* transformation of p, namely $\ln(p/q)$. In the jargon of R, the logit is the *link function* relating the linear predictor to the value of p.

Here is p as a function of x (left panel) and logit(p) as a function of x (right panel) for the logistic with $a=0.2$ and $b=0.1$:



You might ask at this stage 'why not simply do a linear regression of $\ln(p/q)$ against the explanatory x variable?' R has three great advantages here:

- it allows for the non-constant binomial variance
- it deals with the fact that logits for ps near 0 or 1 are infinite
- it allows for differences between the sample sizes by weighted regression

Overdispersion and Hypothesis Testing

All the different modelling procedures that we have met in earlier chapters can also be used with data on proportions. Factorial analysis of variance, analysis of covariance and multiple regression can all be carried out using GLMs. The only difference is that we assess the significance of terms on the basis of chi-squared; this is the increase in scaled deviance that results from removal of a term from the current model.

The important point to bear in mind is that hypothesis testing with binomial errors is less clear-cut than with normal errors. While the chi-squared approximation for changes in scaled deviance is reasonable for large samples (i.e. bigger than about 30), it is poorer with small samples. Most worrisome is the fact that the degree to which the approximation is satisfactory is itself unknown. This means that considerable care must be exercised in the interpretation of tests of hypotheses on parameters, especially when the parameters are marginally significant or when they explain a very small fraction of the total deviance. With binomial or Poisson errors we cannot hope to provide exact p values for our tests of hypotheses.

As with Poisson errors, we need to address the question of overdispersion (see Chapter 13). When we have obtained the minimal adequate model, *the residual scaled deviance should be roughly equal to the residual degrees of freedom*. When the residual deviance is larger than the residual degrees of freedom there are two possibilities: either the model is misspecified, or the probability of success, p, is not constant within a given treatment level.

The effect of randomly varying p is to increase the binomial variance from npq to

$$s^2 = npq + n(n-1)\sigma^2$$

leading to a larger-than-expected residual deviance. This occurs even for models that would fit well if the random variation were correctly specified.

One simple solution is to assume that the variance is not npq but $npqs$, where s is an unknown *scale parameter* ($s > 1$). We obtain an estimate of the scale parameter by dividing the Pearson chi-squared by the degrees of freedom, and use this estimate of s to compare the resulting scaled deviances. To accomplish this, we use `family = quasibinomial` rather than `family = binomial` when there is overdispersion.

The most important points to emphasize in modelling with binomial errors are:

- create a two-column object for the response, using `cbind` to join together the two vectors containing the counts of success and failure
- check for overdispersion (residual deviance greater than residual degrees of freedom), and correct for it by using `family=quasibinomial` rather than `family=binomial` if necessary
- remember that you do not obtain exact p values with binomial errors; the chi-squared approximations are sound for large samples, but small samples may present a problem
- the fitted values are counts, like the response variable
- because of this the raw residuals are also in counts
- the linear predictor that you see when you ask for `summary(model)` is in logits (the log of the odds, $\ln(p/q)$)
- you can back-transform from logits (z) to proportions (p) by $p = 1/(1 + 1/\exp(z))$

Applications

You can do as many kinds of modelling in a GLM as in a linear model: here we show examples of:

- regression with binomial errors (continuous explanatory variables)
- analysis of deviance with binomial errors (categorical explanatory variables)
- analysis of covariance with binomial errors (both kinds of explanatory variables)

Logistic Regression with Binomial Errors

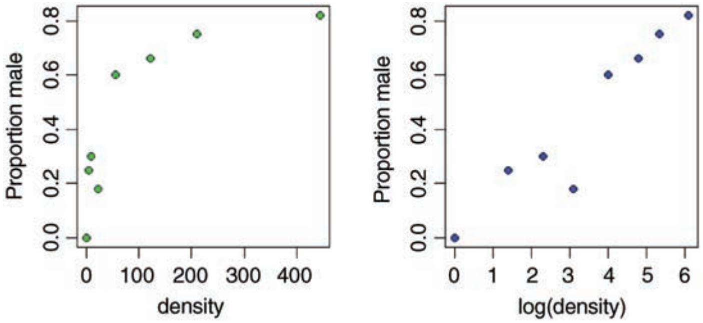
This example concerns sex ratios in insects (the proportion of all individuals that are males). In the species in question, it has been observed that the sex ratio is highly variable, and an experiment was set up to see whether population density was involved in determining the fraction of males.

```
numbers <- read.csv("c:\\temp\\sexratio.csv")
numbers
```

```
  density females  males
1       1       1      0
2       4       3      1
3      10       7      3
4      22      18      4
5      55      22     33
6     121      41     80
7     210      52    158
8     444      79    365
```

It certainly looks as if there are proportionally more males at high density, but we should plot the data as proportions to see this more clearly:

```
attach(numbers)
windows(7,4)
par(mfrow=c(1,2))
p <- males/(males+females)
plot(density,p,ylab="Proportion male")
plot(log(density),p,ylab="Proportion male")
```



Evidently, a logarithmic transformation of the explanatory variable is likely to improve the model fit. We shall see in a moment.

Does increasing population density lead to a significant increase in the proportion of males in the population? Or, more succinctly, is the sex ratio density dependent? It certainly looks from the plot as if it is.

The response variable is a matched pair of counts that we wish to analyse as proportion data using a GLM with binomial errors. First we bind together the vectors of male and female counts into a single object that will be the response in our analysis:

```
y <- cbind(males,females)
```

This means that *y* will be interpreted in the model as the proportion of all individuals that were male. The model is specified like this:

```
model <- glm(y~density,binomial)
```

This says that the object called `model` gets a generalized linear model in which *y* (the sex ratio) is modelled as a function of a single continuous explanatory variable called `density`, using an error distribution from the `family = binomial`. The output looks like this:

```
summary(model)

Coefficients:
              Estimate   Std. Error z value   Pr (>|z|)
(Intercept)  0.0807368    0.1550376   0.521      0.603
density      0.0035101    0.0005116   6.862   6.81e-12   ***

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 71.159 on 7 degrees of freedom
Residual deviance: 22.091 on 6 degrees of freedom
AIC: 54.618

Number of Fisher Scoring iterations: 4
```

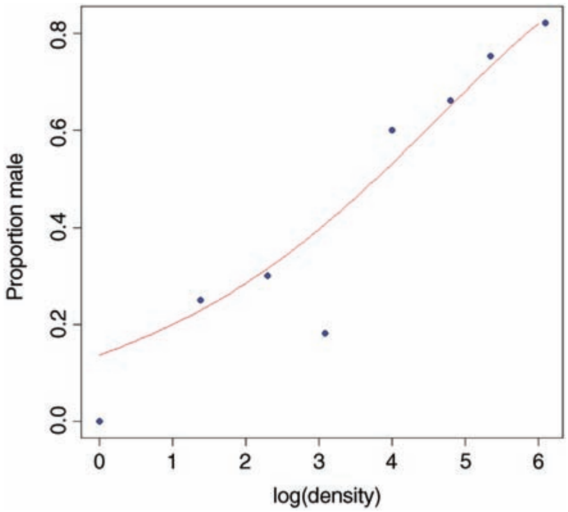
The model table looks just as it would for a straightforward regression. The first parameter is the intercept and the second is the slope of the graph of sex ratio against population density. The slope is highly significantly steeper than zero (proportionately more males at higher population density; $p = 6.81 \times 10^{-12}$). We can see if log transformation of the explanatory variable reduces the residual deviance below 22.091:

```
model <- glm(y~log(density),binomial)
summary(model)

Coefficients:
             Estimate   Std. Error  z   value   Pr (>|z|)
(Intercept) -2.65927      0.48758     -5.454   4.92e-08   ***
log(density) 0.69410      0.09056      7.665   1.80e-14   ***

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 71.1593 on 7 degrees of freedom
Residual deviance:  5.6739 on 6 degrees of freedom
AIC: 38.201

Number of Fisher Scoring iterations: 4
```

This is a big improvement, so we shall adopt it. There is a technical point here, too. In a GLM like this, it is assumed that the residual deviance is the same as the residual degrees of freedom. If the residual deviance is larger than the residual degrees of freedom, this is called overdispersion. It means that there is extra, unexplained variation, over and above the binomial variance assumed by the model specification. In the model with log(density) there is no evidence of overdispersion (residual deviance = 5.67 on 6 d.f.), whereas the lack of fit introduced by the curvature in our first model caused substantial overdispersion (residual deviance = 22.09 on 6 d.f.) when we had density instead of log(density) as the explanatory variable.

Model checking involves the use of plot(model). As you will see, there is no worryingly great pattern in the residuals against the fitted values, and the normal plot is reasonably linear. Point number 4 is highly influential (it has a big value of Cook's distance), and point number 8 has high leverage (but the model is still significant with this point omitted). We conclude that the proportion of animals that are males increases significantly with increasing density, and that the logistic model is linearized by logarithmic transformation of the explanatory variable (population density). We finish by drawing the fitted line though the scatterplot:

```
xv <- seq(0,6,0.1)
plot(log(density),p,ylab="Proportion male",pch=21,bg="blue")
lines(xv,predict(model,list(density=exp(xv)),
type="response"),col="brown")
```



Note the use of type="response" to back-transform from the logit scale to the S-shaped proportion scale and exp(xv) to back-transform the logs from the *x* axis to densities as required by the model formula (where they are then logged again). As you can see, the model is very poor fit for the lowest density and for log(density) ≈ 3, but a reasonably good fit over the rest of the range. I suspect that the replication was simply too low at very low population densities to get an accurate estimate of the sex ratio. Of course, we should not discount the possibility that the data point is not an outlier, but rather that the model is wrong.

Proportion Data with Categorical Explanatory Variables

This example concerns the germination of seeds of two genotypes of the parasitic plant *Orobanche* and two extracts from host plants (bean and cucumber) that were used to stimulate germination. It is a two-way factorial analysis of deviance.

```
germination <- read.csv("c:\\temp\\germination.csv")
attach(germination)
names(germination)
```

```
[1] "count"  "sample"  "Orobanche" "extract"
```

The response variable count is the number of seeds that germinated out of a batch of size = sample. So the number that didn't germinate is sample – count, and we construct the response vector like this:

```
y <- cbind(count, sample-count)
```

Each of the categorical explanatory variables has two levels:

```
levels(Orobanche)
```

```
[1] "a73" "a75"
```

```
levels(extract)
```

```
[1] "bean"  "cucumber"
```

We want to test the hypothesis that there is no interaction between *Orobanche* genotype (a73 or a75) and plant extract (bean or cucumber) on the germination rate of the seeds. This requires a factorial analysis using the asterisk * operator like this:

```
model <- glm(y ~ Orobanche * extract, binomial)
summary(model)
```

```
Coefficients:
                             Estimate Std. Error z value  Pr (>|z|)
(Intercept)                   -0.4122     0.1842  -2.238    0.0252  *
Orobanchea75                  -0.1459     0.2232  -0.654    0.5132
extractcucumber                0.5401     0.2498   2.162    0.0306  *
Orobanchea75:extractcucumber   0.7781     0.3064   2.539    0.0111  *

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 98.719 on 20 degrees of freedom
Residual deviance: 33.278 on 17 degrees of freedom
AIC: 117.87

Number of Fisher Scoring iterations: 4
```

At first glance, it looks as if there is a highly significant interaction ($p = 0.0111$). But we need to check that the model is sound. The first thing is to check for is overdispersion. The residual deviance is 33.278 on 17 d.f. so the model is quite badly overdispersed:

```
33.279 / 17
```

```
[1] 1.957588
```

The overdispersion factor is almost 2. The simplest way to take this into account is to use what is called an ‘empirical scale parameter’ to reflect the fact that the errors are not binomial as we assumed, but were larger than this (overdispersed) by a factor of 1.9576. We refit the model using quasibinomial to account for the overdispersion:

```
model <- glm(y ~ Orobanche * extract, quasibinomial)
```

Then we use update to remove the interaction term in the normal way:

```
model2 <- update(model, ~ . - Orobanche:extract)
```

The only difference is that we use an *F* test instead of a chi-squared test to compare the original and simplified models:

```
anova(model,model2,test="F")
Analysis of Deviance Table

Model 1:  y ~ Orobanche * extract
Model 2:  y ~ Orobanche + extract

  Resid. Df Resid. Dev  Df  Deviance      F   Pr(>F)
1        17     33.278
2        18     39.686  -1    -6.408  3.4419  0.08099  .
```

Now you see that the interaction is not significant ($p = 0.081$). There is no compelling evidence that different genotypes of *Orobanche* respond differently to the two plant extracts. The next step is to see if any further model simplification is possible:

```
anova(model2,test="F")

Analysis of Deviance Table

Model: quasibinomial, link: logit

Response: y

Terms added sequentially (first to last)


           Df  Deviance  Resid.  Df Resid.  Dev       F     Pr(>F)
NULL                             20     98.719
Orobanche   1     2.544          19     96.175   1.1954     0.2887
extract     1    56.489          18     39.686  26.5412 6.692e-05  ***
```

There is a highly significant difference between the two plant extracts on germination rate, but it is not obvious that we need to keep *Orobanche* genotype in the model. We try removing it:

```
model3 <- update(model2, ~ . - Orobanche)
anova(model2,model3,test="F")
```

```
Analysis of Deviance Table

Model 1: y ~ Orobanche + extract
Model 2: y ~ extract
  Resid. Df  Resid. Dev  Df  Deviance       F Pr(>F)
1        18      39.686
2        19      42.751  -1    -3.065  1.4401 0.2457
```

There is no justification for retaining *Orobanche* in the model. So the minimal adequate model contains just two parameters:

```
coef(model3)
```

```
    (Intercept) extractcucumber
     -0.5121761       1.0574031
```

What, exactly, do these two numbers mean? Remember that the coefficients are from the linear predictor. They are on the transformed scale, so because we are using quasi-binomial errors, they are in logits ($\ln(p/(1-p))$). To turn them into the germination rates for the two plant extracts requires a little calculation. To go from a logit x to a proportion p, you need to do the following sum:

$$p = \frac{1}{1 + \frac{1}{e^x}}$$

So our first x value is -0.5122 and we calculate

```
1/(1+1/(exp(-0.5122)))
```

```
[1] 0.3746779
```

This says that the mean germination rate of the seeds with the first plant extract was 37%. What about the parameter for cucumber extract (1.057)? Remember that with categorical explanatory variables *the parameter values are differences between means*. So to get the second germination rate we *add 1.057 to the intercept* before back-transforming:

```
1/(1+1/(exp(-0.5122+1.0574)))
```

```
[1] 0.6330212
```

This says that the germination rate was nearly twice as great (63%) with the second plant extract (cucumber). Obviously we want to generalize this process, and also to speed up the calculations of the estimated mean proportions. We can use `predict` to help here, because `type="response"` makes predictions on the back-transformed scale automatically:

```
tapply(predict(model3,type="response"),extract,mean)

     bean   cucumber
0.3746835 0.6330275
```

It is interesting to compare these figures with the averages of the raw proportions. First we need to calculate the proportion germinating, *p*, in each sample:

```
p <- count/sample
```

Then we can find the average the germination rates for each extract:

```
tapply(p,extract,mean)

     bean   cucumber
0.3487189 0.6031824
```

You see that this gives different answers. Not too different in this case, it’s true, but different nonetheless. The correct way to average proportion data is to add up the total counts for the different levels of abstract, and only then to turn them into proportions:

```
tapply(count,extract,sum)

  bean  cucumber
   148       276
```

This means that 148 seeds germinated with bean extract and 276 with cucumber. But how many seeds were involved in each case?

```
tapply(sample,extract,sum)

  bean  cucumber
   395       436
```

This means that 395 seeds were treated with bean extract and 436 seeds were treated with cucumber. So the answers we want are 148/395 and 276/436 (i.e. the correct mean proportions). We automate the calculation like this:

```
as.vector(tapply(count,extract,sum))/
as.vector(tapply(sample,extract,sum))

[1] 0.3746835  0.6330275
```

These are the correct mean proportions as produced by the GLM. The moral here is that *you calculate the average of proportions by using total counts and total samples and not by averaging the raw proportions.*

To summarize this analysis:

- make a two-column response vector containing the successes and failures
- use `glm` with `family=binomial` (you don't need to include `family=`)
- fit the maximal model (in this case it had four parameters)
- test for overdispersion
- if, as here, you find overdispersion then use `quasibinomial` errors
- begin model simplification by removing the interaction term
- this was non-significant once we had adjusted for overdispersion
- try removing main effects (we didn't need *Orobanche* genotype in the model)
- use `plot` to obtain your model-checking diagnostics
- back transform using `predict` with the option `type="response"` to obtain means

Analysis of Covariance with Binomial Data

This example concerns flowering in five varieties of perennial plants. Replicated individuals in a fully randomized design were sprayed with one of six doses of a controlled mixture of growth promoters. After 6 weeks, plants were scored as flowering or not flowering. The count of flowering individuals forms the response variable. This is an ANCOVA because we have both continuous (`dose`) and categorical (`variety`) explanatory variables. We use logistic regression because the response variable is a count (`flowered`) that can be expressed as a proportion (`flowered/number`).

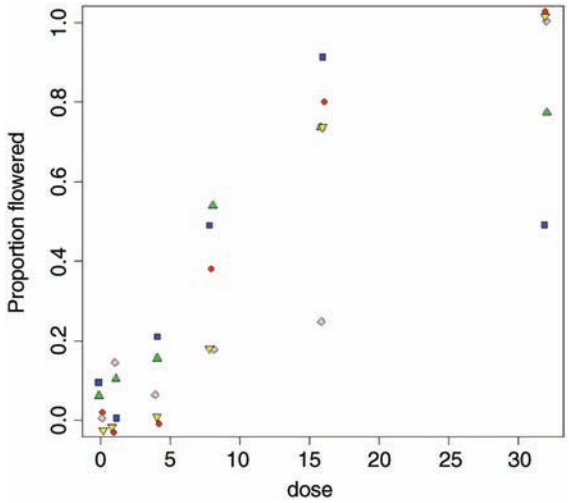
```
props <- read.csv("c:\\temp\\flowering.csv")
attach(props)
names(props)
```

```
[1] "flowered" "number"  "dose"   "variety"
```

```
y <- cbind(flowered,number-flowered)
pf <- flowered/number
pfc <- split(pf,variety)
dc <- split(dose,variety)
```

With count data like these, it is common for points from different treatments to hide one another. Note the use of `jitter` to move points left or right and up or down by a small random amount so that all of the symbols at a different point can be seen:

```
plot(dose,pf,type="n",ylab="Proportion flowered")
points(jitter(dc[[1]]),jitter(pfc[[1]]),pch=21,bg="red")
points(jitter(dc[[2]]),jitter(pfc[[2]]),pch=22,bg="blue")
points(jitter(dc[[3]]),jitter(pfc[[3]]),pch=23,bg="gray")
points(jitter(dc[[4]]),jitter(pfc[[4]]),pch=24,bg="green")
points(jitter(dc[[5]]),jitter(pfc[[5]]),pch=25,bg="yellow")
```



There is clearly a substantial difference between the plant varieties in their response to the flowering stimulant. The modelling proceeds in the normal way. We begin by fitting the maximal model with different slopes and intercepts for each variety (estimating 10 parameters in all):

```
model1 <- glm(y~dose*variety,binomial)
summary(model1)
```

The model exhibits substantial overdispersion (51.083/20 > 2), so we fit the model again with quasi-binomial errors:

```
model2 <- glm(y~dose*variety,quasibinomial)
summary(model2)
```

```
Coefficients:
              Estimate  Std. Error  t value  Pr (>|t|)
(Intercept)   -4.59165    1.56314   -2.937   0.00814   **
dose           0.41262    0.15195    2.716   0.01332   *
varietyB       3.06197    1.65555    1.850   0.07922   .
varietyC       1.23248    1.79934    0.685   0.50123
varietyD       3.17506    1.62828    1.950   0.06534   .
varietyE      -0.71466    2.34511   -0.305   0.76371
dose:varietyB -0.34282    0.15506   -2.211   0.03886   *
dose:varietyC -0.23039    0.16201   -1.422   0.17043
dose:varietyD -0.30481    0.15534   -1.962   0.06380   .
dose:varietyE -0.00649    0.20130   -0.032   0.97460

 (Dispersion parameter for quasibinomial family taken to be 2.293557)

    Null deviance: 303.350 on 29 degrees of freedom
Residual deviance:  51.083 on 20 degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 5
```

Do we need to retain the interaction term (it appears to be significant in only one case)? We leave it out and compare the two models using anova:

```
model3 <- glm(y~dose+variety,quasibinomial)
anova(model2,model3,test="F")
```

```
Analysis of Deviance Table

Model 1: y ~ dose * variety
Model 2: y ~ dose + variety
  Resid. Df  Resid. Dev  Df  Deviance       F     Pr(>F)
1        20      51.083
2        24      96.769 -4   -45.686 4.9798   0.005969 **
```

Yes, we do. The interaction term is much more significant than indicated by any one of the *t* tests ($p = 0.005969$).

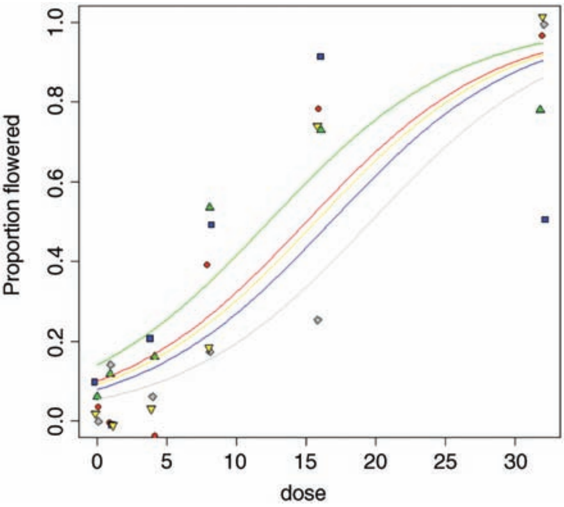
Let us draw the five fitted curves through the scatterplot. The values on the dose axis need to go from 0 to 30:

```
xv <- seq(0,32,0.25)
length(xv)
```

```
[1] 129
```

This means we shall need to provide the predict function with 129 repeats of each factor level in turn:

```
yv <- predict(model3,list(dose=xv,variety=rep("A",129)),
type="response")
lines(xv,yv,col="red")
yv <- predict(model3,list(dose=xv,variety=rep("B",129)),
type="response")
lines(xv,yv,col="blue")
yv <- predict(model3,list(dose=xv,variety=rep("C",129)),
type="response")
lines(xv,yv,col="gray")
yv <- predict(model3,list(dose=xv,variety=rep("D",129)),
type="response")
lines(xv,yv,col="green")
yv <- predict(model3,list(dose=xv,variety=rep("E",129)),
type="response")
lines(xv,yv,col="yellow")
```



As you can see, the model is a reasonable fit for two of the varieties (A and E, represented red circles and yellow down-triangles, respectively), not bad for one variety (C, grey diamonds) but very poor for two of them: B (blue squares) and D (green up-triangles). For several varieties, the model overestimates the proportion flowering at zero dose, and for variety B there seems to be some inhibition of flowering at the highest dose because the graph *falls* from 90% flowering at dose 16 to just 50% at dose 32 (the fitted model is assumed to be asymptotic). Variety D appears to be asymptoting at less than 80% flowering. These failures of the model focus attention for future work.

The moral is that just because we have proportion data, that does not mean that the data will necessarily be well described by the logistic. For instance, in order to describe the response of variety B, the model would need to have a hump, rather than to asymptote at $p = 1$ for large doses, and to model variety D the model would need an extra parameter to estimate an asymptotic value that was less than 100%.

Further Reading

Hosmer, D.W. and Lemeshow, S. (2000) *Applied Logistic Regression*, 2nd edn, John Wiley & Sons, New York.

15

Binary Response Variable

Many statistical problems involve binary response variables. For example, we often classify things as dead or alive, occupied or empty, healthy or diseased, male or female, literate or illiterate, mature or immature, solvent or insolvent, employed or unemployed, and it is interesting to understand the factors that are associated with an individual being in one class or the other. In a study of company insolvency, for instance, the data would consist of a list of measurements made on the insolvent companies (their age, size, turnover, location, management experience, workforce training, and so on) and a similar list for the solvent companies. The question then becomes which, if any, of the explanatory variables increase the probability of an individual company being insolvent.

The response variable contains only 0s or 1s; for example, 0 to represent dead individuals and 1 to represent live ones. Thus, there is only a single column of numbers for the response, in contrast to proportion data where two vectors (successes and failures) were bound together to form the response (see Chapter 14). An alternative is allowed by R in which the values of the response variable are represented by a two-level factor (like 'dead' or 'alive', 'male' or 'female', etc.).

The way that R treats binary data is to assume that the values of the response come from *a binomial trial with sample size 1*. If the probability that an individual is dead is p, then the probability of obtaining y (where y is either dead or alive, 0 or 1) is given by an abbreviated form of the binomial distribution with $n = 1$, known as the Bernoulli distribution:

$$P(y) = p^y(1-p)^{1-y}$$

The random variable y has a mean of p and a variance of $p(1-p)$, and the object is to determine how the explanatory variables influence the value of p. The trick to using binary response variables effectively is to know when it is worth using them, and when it is better to lump the successes and failures together and analyse the *total counts* of dead individuals, occupied patches, insolvent firms or whatever. The question you need to ask yourself is whether or not you have unique values of one or more explanatory variables for each and every individual case. If the answer is 'yes', then analysis with a binary response variable is likely to be fruitful. If the answer is 'no', then there is nothing to be gained, and you should reduce your data by aggregating the counts to the resolution at which each count *does* have a

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

unique set of explanatory variables. For example, suppose that all your explanatory variables were categorical (say sex (male or female), employment (employed or unemployed) and region (urban or rural)). In this case there is nothing to be gained from analysis using a binary response variable because none of the individuals in the study have *unique* values of any of the explanatory variables. It might be worthwhile if you had each individual's body weight, for example; then you could ask whether, when you control for sex and region, heavy people are more likely to be unemployed than light people. In the absence of *unique* values for any explanatory variables, there are two useful options:

- analyse the data as a contingency table using Poisson errors, with the count of the total number of individuals in each of the eight contingencies (2 × 2 × 2) as the response variable (see Chapter 13) in a dataframe with just eight rows
- decide which of your explanatory variables is the key (perhaps you are interested in gender differences), then express the data as proportions (the number of males and the number of females) and recode the binary response as a count of a two-level factor – the analysis is now of proportion data (e.g. the proportion of all individuals that are female) using binomial errors (see Chapter 14)

If you *do* have unique measurements of one or more explanatory variables for each individual, these are likely to be continuous variables such as body weight, income, medical history, distance to the nuclear reprocessing plant, geographic isolation, and so on. This being the case, successful analyses of binary response data tend to be multiple regression analyses or complex analyses of covariance, and you should consult Chapters 10 and 11 for details on model simplification and model criticism.

In order to carry out modelling on a binary response variable we take the following steps:

- create a single vector containing 0s and 1s (or one of two factor levels) as the response variable
- use `glm` with `family=binomial`
- you can change the link function from the default logit to complementary log-log
- fit the model in the usual way
- test significance by deletion of terms from the maximal model, and compare the change in deviance with chi-squared
- note that there is no such thing as overdispersion with a binary response variable, and hence no need to change to using quasi-binomial when the residual deviance is large
- `plot(model)` is rarely informative with binary response variables, so model checking is more than usually challenging

The choice of link function is generally made by trying both links and selecting the link that gives the lowest deviance. The logit link that we used earlier is symmetric in p and q, but the complementary log-log link is asymmetric.

Incidence Functions

In this example, the response variable is called incidence; a value of 1 means that an island was occupied by a particular species of bird, and 0 means that the bird did not breed there. The explanatory variables are the area of the island (km^2) and the isolation of the island (distance from the mainland, km).

```
island <- read.csv("c:\\temp\\isolation.csv")
attach(island)
names(island)
```

```
[1] "incidence" "area"   "isolation"
```

There are two continuous explanatory variables, so the appropriate analysis is multiple regression. The response is binary, so we shall do logistic regression with binomial errors. We begin by fitting a complex model involving an interaction between isolation and area:

```
model1 <- glm(incidence~area*isolation,binomial)
```

Then we fit a simpler model with only main effects for isolation and area:

```
model2 <- glm(incidence~area+isolation,binomial)
```

Now we compare the two models using anova:

```
anova(model1,model2,test="Chi")
```

```
Analysis of Deviance Table

Model 1: incidence ~ area * isolation
Model 2: incidence ~ area + isolation

Resid.  Df   Resid. Dev   Df   Deviance   Pr(>Chi)
1       46       28.252
2       47       28.402   -1   -0.15043     0.6981
```

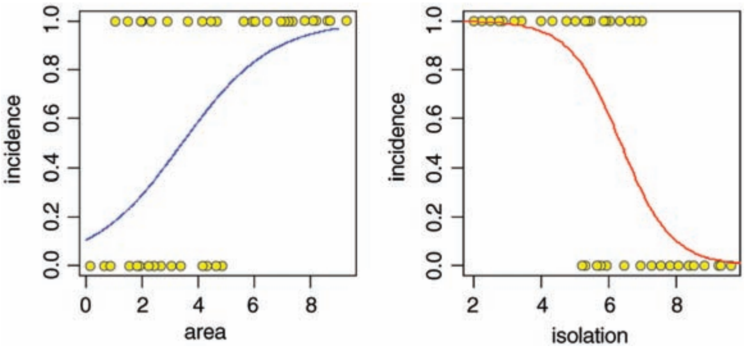
The simpler model is not significantly worse, so we accept this for the time being, and inspect the parameter estimates and standard errors:

```
summary(model2)
```

```
Coefficients:
               Estimate   Std. Error    z value     Pr(>|z|)
(Intercept)      6.6417       2.9218      2.273     0.02302 *
area             0.5807       0.2478      2.344     0.01909 *
isolation       -1.3719       0.4769     -2.877     0.00401 **

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 68.029 on 49 degrees of freedom
Residual deviance: 28.402 on 47 degrees of freedom

AIC: 34.402

Number of Fisher Scoring iterations: 6
```

The estimates and their standard errors are in logits. Area has a significant positive effect (larger islands are more likely to be occupied), but isolation has a very strong negative effect (isolated islands are much less likely to be occupied). This is the minimal adequate model. We should plot the fitted model through the scatterplot of the data. It is much easier to do this for each variable separately, like this:

```
windows(7,4)
par(mfrow=c(1,2))
xv <- seq(0,9,0.01)

modela <- glm(incidence~area,binomial)
modeli <- glm(incidence~isolation,binomial)

yv <- predict(modela,list(area=xv),type="response")
plot(area,incidence,pch=21,bg="yellow")
lines(xv,yv,col="blue")

xv2 <- seq(0,10,0.1)
yv2 <- predict(modeli,list(isolation=xv2),type="response")
plot(isolation,incidence,pch=21,bg="yellow")
lines(xv2,yv2,col="red")
```



This is well and good, but it is very difficult to know how good the fit of the model is when the data are shown only as 0s or 1s. It is sensible to compute one or more intermediate probabilities from the data, and to show these empirical estimates (ideally with their standard errors) on the plot in order to judge whether the fitted line is a reasonable description of the data.

For the purposes of demonstration, we divide the ranges of area and of isolation into (say) three, count how many successes and failures occurred in each interval, calculate the mean proportion incidence in each third, p, and add these estimates to the plot as points along with their standard error bars $\sqrt{p(1-p)/n}$. Start by using `cut` to obtain the break points on the two axes:

```
ac <- cut(area,3)
ic <- cut(isolation,3)
tapply(incidence,ac,sum)
```

```
(0.144,3.19] (3.19,6.23] (6.23,9.28]
           7           8          14
```

```
tapply(incidence,ic,sum)
```

```
(2.02,4.54] (4.54,7.06] (7.06,9.58]
         12          17           0
```

There were seven data points indicating occupation (success) in the lowest third of the area axis and 14 in the highest third. There were 12 data points indicating occupation (success) in the lowest third of range of values for isolation and none in the highest third. Note the convention for labelling intervals: `(a, b]` means include the right-hand endpoint, `b` (the square bracket), but not the left one, `a` (the round bracket).

Now count the total number of islands in each interval using `table`:

```
table(ac)
```

```
ac
(0.144,3.19] (3.19,6.23] (6.23,9.28]
          21          15          14
```

```
table(ic)
```

```
ic
(2.02,4.54] (4.54,7.06] (7.06,9.58]
         12          25          13
```

The probability of occupation is given by dividing the number of successes by the number of cases:

```
tapply(incidence,ac,sum)/ table(ac)
```

```
(0.144,3.19] (3.19,6.23] (6.23,9.28]
   0.3333333    0.5333333    1.0000000
```

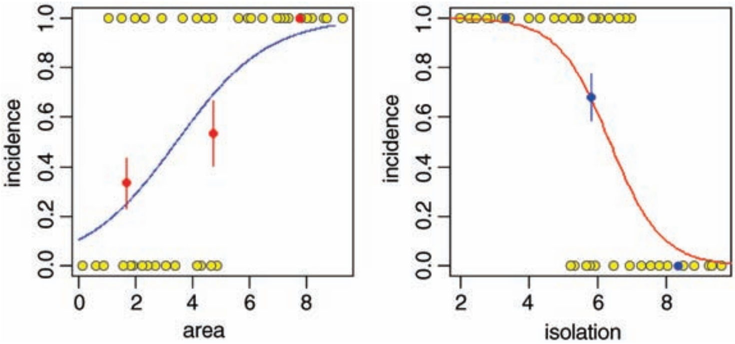
```
tapply(incidence,ic,sum)/ table(ic)
```

```
(2.02,4.54] (4.54,7.06] (7.06,9.58]
       1.00        0.68        0.00
```

The idea is to plot these mean proportions and their standard errors ($\sqrt{pq/n}$) along with the regression line from the model to see how close the model gets to the three calculated proportions:

```
xv <- seq(0,9,0.01)
yv <- predict(modela,list(area=xv),type="response")
plot(area,incidence,pch=21,bg="yellow")
lines(xv,yv,col="blue")
```

```
d <- (max(area)-min(area))/3
left <- min(area)+d/2
mid <- left+d
right <- mid+d
xva <- c(left,mid,right)
pa <- as.vector(tapply(incidence,ac,sum)/ table(ac))
se <- sqrt(pa*(1-pa)/table(ac))

xv <- seq(0,9,0.01)
yv <- predict(modela,list(area=xv),type="response")
lines(xv,yv,col="blue")

points(xva,pa,pch=16,col="red")
for (i in 1:3) lines(c(xva[i],xva[i]),
c(pa[i]+se[i],pa[i]-se[i]),col="red")

xv2 <- seq(0,10,0.1)
yv2 <- predict(modeli,list(isolation=xv2),type="response")
plot(isolation,incidence,pch=21,bg="yellow")
lines(xv2,yv2,col="red")

d <- (max(isolation)-min(isolation))/3
left <- min(isolation)+d/2
mid <- left+d
right <- mid+d
xvi <- c(left,mid,right)
pi <- as.vector(tapply(incidence,ic,sum)/ table(ic))
se <- sqrt(pi*(1-pi)/table(ic))

points(xvi,pi,pch=16,col="blue")
for (i in 1:3) lines(c(xvi[i],xvi[i]),
c(pi[i]+se[i],pi[i]-se[i]),col="blue")
```



You can see at once that the fit for the right-hand graph of incidence against isolation is excellent; the logistic is a very good model for these data. In contrast, the fit for the left-hand graph of incidence against area is poor; at low values of area the model (blue line) underestimates the observed data (red point with error bars) while for intermediate values of area the model (blue line) overestimates the observed data (red point with error bars).

This approach of plotting the two effects separately would not work, of course, if there were a significant interaction between area and isolation; then you would need to produce conditioning plots of incidence against area for different degrees of isolation.

ANCOVA with a Binary Response Variable

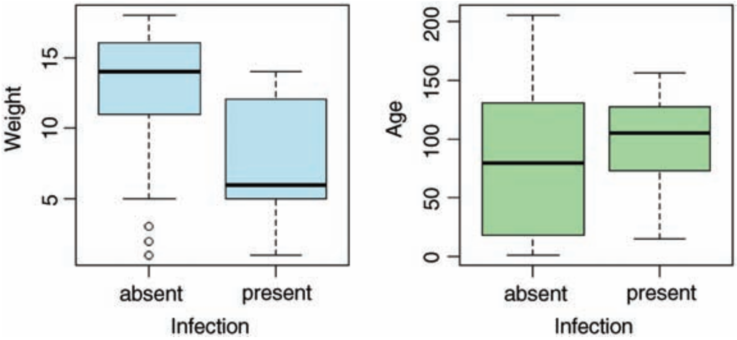
In this example the binary response variable is parasite infection (infected or not) and the explanatory variables are weight and age (continuous) and sex (categorical). We begin with data inspection:

```
infection <- read.csv("c:\\temp\\infection.csv")
attach(infection)
names(infection)
```

```
[1] "infected" "age"    "weight"   "sex"
```

For the continuous explanatory variables, weight and age, it is useful to look at box-and-whisker plots of the data:

```
windows(7,4)
par(mfrow=c(1,2))
plot(infected,weight,xlab="Infection",ylab="Weight",
col="lightblue")
plot(infected,age,xlab="Infection",ylab="Age", col="lightgreen")
```



Infected individuals are substantially lighter than uninfected individuals, and occur in a much narrower range of ages. To see the relationship between infection and gender (both categorical variables) we can use `table`:

```
table(infected,sex)
```

```
           sex
infected   female  male
  absent       17    47
  present      11     6
```

This indicates that the infection is much more prevalent in females (11/28) than in males (6/53).

We get down to business, as usual, by fitting a maximal model with different slopes for each level of the categorical variable:

```
model <- glm(infected~age*weight*sex,family=binomial)
summary(model)
```

```
Coefficients:
                     Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)         -0.109124    1.375388    -0.079     0.937
age                  0.024128    0.020874     1.156     0.248
weight              -0.074156    0.147678    -0.502     0.616
sexmale             -5.969109    4.278066    -1.395     0.163
age:weight          -0.001977    0.002006    -0.985     0.325
age:sexmale          0.038086    0.041325     0.922     0.357
weight:sexmale       0.213830    0.343265     0.623     0.533
age:weight:sexmale  -0.001651    0.003419    -0.483     0.629

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234 on 80 degrees of freedom
Residual deviance: 55.706 on 73 degrees of freedom
AIC: 71.706

Number of Fisher Scoring iterations: 6
```

It certainly does not look as if any of the high-order interactions are significant. Instead of using `update` and `anova` for model simplification, we can use `step` to compute the AIC for each term in turn:

```
model2 <- step(model)

Start: AIC= 71.71
```

First, it tests whether the three-way interaction is required:

```
                    Df   Deviance     AIC
- age:weight:sex     1     55.943   69.943
<none>                     55.706   71.706

Step: AIC= 69.94
```

This causes a reduction in AIC of just 71.7 – 69.9 = 1.8 and hence is not significant, so the three-way interaction is eliminated.

Next, it looks at the two-way interactions and decides which of the three to delete first:

```
                    Df   Deviance     AIC
- weight:sex         1     56.122   68.122
- age:sex            1     57.828   69.828
<none>                     55.943   69.943
- age:weight         1     58.674   70.674

Step: AIC= 68.12
```

Only the removal of the weight–sex interaction causes a reduction in AIC, so this interaction in deleted and the other two interactions are retained. Let us see if we would have been this lenient:

```
summary(model2)
```

```
Call:
glm(formula = infected ~ age + weight + sex + age:weight + age:sex,
    family = binomial)

Coefficients:
                Estimate   Std. Error   z value   Pr(>|z|)
(Intercept)    -0.391572    1.264850    -0.310    0.7569
age             0.025764    0.014918     1.727    0.0842  .
weight         -0.036493    0.128907    -0.283    0.7771
sexmale        -3.743698    1.786011    -2.096    0.0361  *
age:weight     -0.002221    0.001365    -1.627    0.1037
age:sexmale     0.020464    0.015199     1.346    0.1782

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234 on 80 degrees of freedom
Residual deviance: 56.122 on 75 degrees of freedom
AIC: 68.122
```

Neither of the two interactions retained by `step` would figure in our model ($p > 0.10$). We shall use `update` to simplify `model2`:

```
model3 <- update(model2,~.-age:weight)
anova(model2,model3,test="Chi")
```

```
Analysis of Deviance Table

Model 1: infected ~ age + weight + sex + age:weight + age:sex
Model 2: infected ~ age + weight + sex + age:sex

Resid.  Df   Resid. Dev   Df    Deviance  Pr(>Chi)
1       75      56.122
2       76      58.899    -1       -2.777  0.09562 .
```

So there is no really persuasive evidence of an age–weight term ($p = 0.096$).

```
model4 <- update(model2,~.-age:sex)
anova(model2,model4,test="Chi")
```

```
Analysis of Deviance Table

Model 1: infected ~ age + weight + sex + age:weight + age:sex
Model 2: infected ~ age + weight + sex + age:weight

Resid.  Df   Resid. Dev   Df   Deviance   Pr(>Chi)
1       75      56.122
2       76      58.142    -1    -2.0203     0.1552
```

Note that we are testing all the two-way interactions by deletion from the model that contains all two-way interactions (model2): $p = 0.1552$, so nothing there, then.

What about the three main effects?

```
model5 <- glm(infected~age+weight+sex,family=binomial)
summary(model5)
```

```
Coefficients:
               Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)    0.609369    0.803288     0.759     0.448096
age            0.012653    0.006772     1.868     0.061701  .
weight        -0.227912    0.068599    -3.322     0.000893  ***
sexmale       -1.543444    0.685681    -2.251     0.024388  *

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234 on 80 degrees of freedom
Residual deviance: 59.859 on 77 degrees of freedom
AIC: 67.859

Number of Fisher Scoring iterations: 5
```

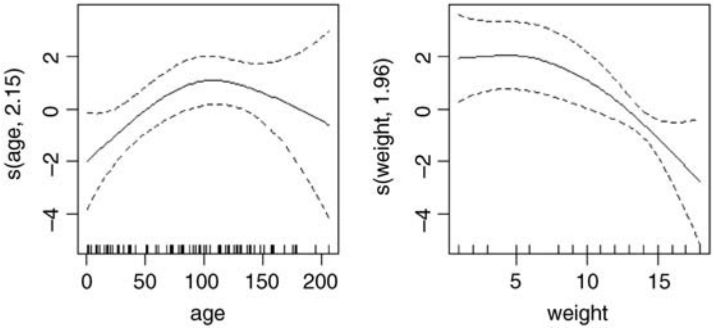
Weight is highly significant, as we expected from the initial boxplot, sex is quite significant, and age is marginally significant. It is worth establishing whether there is any evidence of non-linearity in the response of infection to weight or age. We might begin by fitting quadratic terms for the two continuous explanatory variables:

```
model6 <- glm(infected~age+weight+sex+I(weight^2)+I(age^2),
family=binomial)
summary(model6)
```

```
Coefficients:
               Estimate   Std. Error   z value    Pr(>|z|)
(Intercept)  -3.4475839   1.7978359    -1.918       0.0552  .
age           0.0829364   0.0360205     2.302       0.0213  *
weight        0.4466284   0.3372352     1.324       0.1854
sexmale      -1.2203683   0.7683288    -1.588       0.1122
I(weight^2)  -0.0415128   0.0209677    -1.980       0.0477  *
I(age^2)     -0.0004009   0.0002004    -2.000       0.0455  *

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234 on 80 degrees of freedom
Residual deviance: 48.620 on 75 degrees of freedom
AIC: 60.62

Number of Fisher Scoring iterations: 6
```

Evidently, both relationships are significantly non-linear. It is worth looking at these non-linearities in more detail, to see if we can do better with other kinds of models (e.g. non-parametric smoothers, piecewise linear models or step functions). A good start is often a generalized additive model when we have continuous covariates:

```
library(mgcv)
model7 <- gam(infected~sex+s(age)+s(weight),family=binomial)
plot.gam(model7)
```



These non-parametric smoothers are excellent at showing the humped relationship between infection and age, and at highlighting the possibility of a threshold at weight ≈ 8 in the relationship between weight and infection.

We can now return to a GLM to incorporate these ideas. We shall fit `age` and `age^2` as before, but try a piecewise linear fit for `weight`, estimating the threshold weight at a range of values (say, 8–14) and selecting the threshold that gives the lowest residual deviance; this turns out to be a threshold of 12 (a bit higher than it appears from the `gam` plot, above). The piecewise regression is specified by the term:

```
I((weight – 12) * (weight > 12))
```

The `I` ('as is') is necessary to stop the * being evaluated as an interaction term in the model formula. What this expression says is: regress infection on the value of `weight–12`, but only do this when `weight>12` is true; otherwise, assume that infection is independent of weight.

```
model8 <- glm(infected~sex+age+I(age^2)+
I((weight-12)*(weight>12)),family=binomial)
summary(model8)
```

```
Coefficients:
                                Estimate Std. Error  z value  Pr(>|z|)
(Intercept)                   -2.7511382  1.3678824  -2.011   0.0443 *
sexmale                       -1.2864683  0.7349201  -1.750   0.0800 .
age                            0.0798629  0.0348184   2.294   0.0218 *
I(age^2)                      -0.0003892  0.0001955  -1.991   0.0465 *
I((weight - 12) * (weight > 12)) -1.3547520  0.5350853  -2.532   0.0113 *

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234 on 80 degrees of freedom
Residual deviance: 48.687 on 76 degrees of freedom
AIC: 58.687

Number of Fisher Scoring iterations: 7
```

```
model9 <- update(model8,~.-sex)
anova(model8,model9,test="Chi")
model10 <- update(model8,~.-I(age^2))
anova(model8,model10,test="Chi")
```

The effect of sex on infection is not quite significant ($p = 0.071$ for a chi-squared test on deletion), so we leave it out. The quadratic term for age does not look highly significant here ($p = 0.0465$), but a deletion test gives $p = 0.011$, so we retain it. The minimal adequate model is therefore `model9`:

```
summary(model9)
```

```
Coefficients:
                             Estimate Std. Error  z value Pr(>|z|)
(Intercept)                 -3.1207552 1.2665593  -2.464  0.0137 *
age                          0.0765784 0.0323376   2.368  0.0179 *
I(age^2)                    -0.0003843 0.0001846  -2.081  0.0374 *
I((weight - 12) * (weight > 12)) -1.3511706 0.5134681  -2.631  0.0085 **

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 83.234 on 80 degrees of freedom
Residual deviance: 51.953 on 77 degrees of freedom
AIC: 59.953

Number of Fisher Scoring iterations: 7
```

We conclude there is a humped relationship between infection and age, and a threshold effect of weight on infection. The effect of sex is marginal, but might repay further investigation ($p = 0.071$).

Further Reading

Collett, D. (1991) *Modelling Binary Data*, Chapman & Hall, London.
Cox, D.R. and Snell, E.J. (1989) *Analysis of Binary Data*, Chapman & Hall, London.

16

Death and Failure Data

Time-to-death data, and data on failure times, are often encountered in statistical modelling. The main problem is that the variance in such data is almost always non-constant, and so standard methods are inappropriate. If the errors are gamma distributed, then the *variance is proportional to the square of the mean* (recall that with Poisson errors, the variance is equal to the mean). It is straightforward to deal with such data using a generalized linear model (GLM) with gamma errors.

This case study has 50 replicates in each of three treatments: an untreated control, low dosage and high dosage of a novel cancer treatment. The response is the age at death for the rats (expressed as an integer number of months):

```
mortality <- read.csv("c:\\temp\\deaths.csv")
attach(mortality)
names(mortality)
```

```
[1] "death"   "treatment"
```

```
tapply(death,treatment,mean)
```

```
control    high     low
   3.46    6.88    4.70
```

The animals receiving the high dose lived roughly twice as long as the untreated controls. The low dose increased life expectancy by more than 35%. The variance in age at death, however, is not constant:

```
tapply(death,treatment,var)
```

```
  control          high          low
0.4167347     2.4751020     0.8265306
```

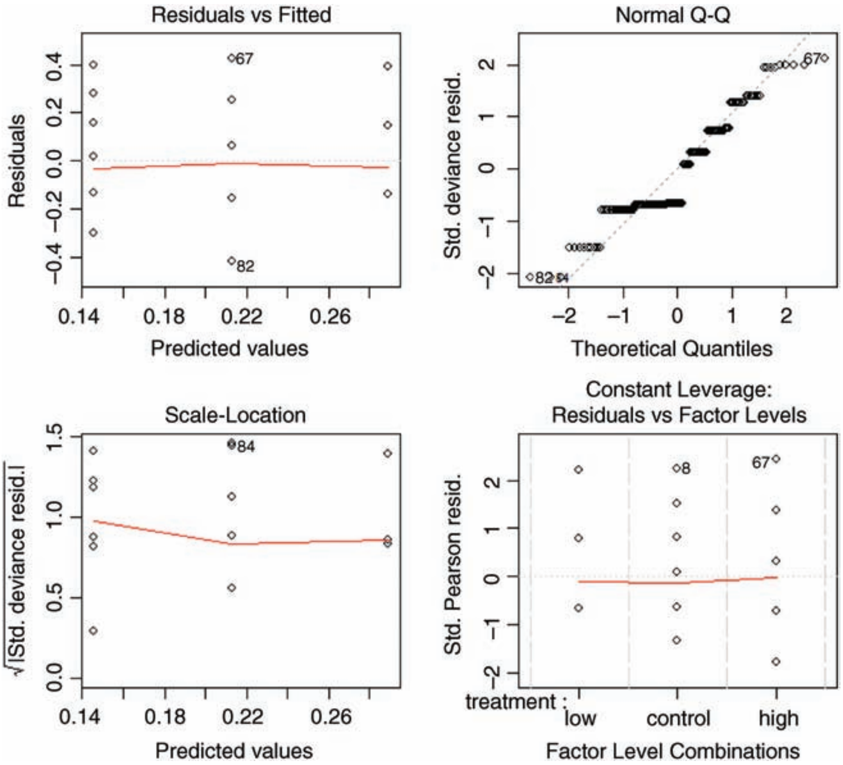
The variance is much greater for the longer-lived individuals, so we should not use standard statistical models which assume constant variance and normal errors. But we can use a GLM with gamma errors:

```
model <- glm(death~treatment,Gamma)
summary(model)
```

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

```
Coefficients:
                   Estimate    Std. Error   t value      Pr(>|t|)
(Intercept)        0.289017     0.008327    34.708      < 2e-16 ***
treatmenthigh     -0.143669     0.009321   -15.414      < 2e-16 ***
treatmentlow      -0.076251     0.010340    -7.374    1.11e-11 ***

(Dispersion parameter for Gamma family taken to be 0.04150576)

    Null deviance: 17.7190 on 149 degrees of freedom
Residual deviance:  5.8337 on 147 degrees of freedom
AIC: 413.52

Number of Fisher Scoring iterations: 4
```

The link function with gamma errors is the reciprocal: that is why the parameter for the high dose appears as a negative term in the summary table: the mean value for high dose is calculated as $0.289 - 0.1437 = 0.1453$, and $1/0.1453 = 6.882$. Checking the model using `plot(model)` shows that it is reasonably well behaved (you might like to compare the behaviour of `lm(death~treatment)`). We conclude that all three treatment levels are significantly different from one another.



A common difficulty with data on time at death is that some (or even many) of the individuals do not die during the trial, so their age at death remains unknown (they might recover, they might leave the trial, or the experiment might end before they die). These individuals are said to be *censored*. Censoring makes the analysis much more complicated, because the censored individuals provide some information (we know the age at which they were last seen alive) but the data are of a different type from the information on age at death which is the response variable in the main analysis. There is a whole field of statistical modelling for such data: it is called *survival analysis*.

```
detach(mortality)
```

Survival Analysis with Censoring

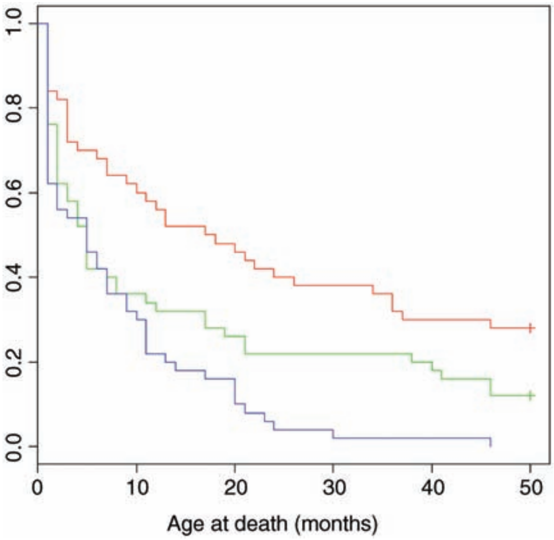
The next example comes from a study of mortality in 150 wild male sheep. There were three experimental *group*s, and the animals were followed for 50 months. The groups were treated with three different medicines against their gut parasites: group A received a bolus with a high dose of worm-killer, group B received a low dose, and group C received the placebo (a bolus with no worm-killing contents). The initial body mass of each individual (*weight*) was recorded as a covariate. The month in which each animal died (*death*) was recorded, and animals which survived up to the 50th month (the end of the study) were recorded as being censored (they thus have censoring indicator status = 0, whereas the animals that died all have status = 1).

```
library(survival)
sheep <- read.csv("c:\\temp\\sheep.deaths.csv")
attach(sheep)
names(sheep)
```

```
[1] "death" "status" "weight" "group"
```

The overall survivorship curves for the three groups of animals are obtained like this:

```
plot(survfit(Surv(death,status)~group),col=c(2,3,4),
                                    xlab="Age at death (months)")
```



The crosses + at the end of the survivorship curves for groups A (red) and B (green) indicate that there was censoring in these groups (not all of the individuals were dead at the end of the experiment). Parametric regression in survival models uses the survreg function, for which you can specify a wide range of different error distributions. Here we use the exponential distribution for the purposes of demonstration (you can chose from dist = "extreme", "logistic", "gaussian" or "exponential" and from link = "log" or "identity"). We fit the full analysis of covariance model to begin with:

```
model <- survreg(Surv(death,status)~weight*group,dist="exponential")
summary(model)
```

```
Call:
survreg(formula = Surv(death, status) ~ weight * group, dist =
"exponential")
                 Value   Std. Error        z          p
(Intercept)     3.8702       0.3854   10.041   1.00e-23
weight         -0.0803       0.0659   -1.219   2.23e-01
groupB         -0.8853       0.4508   -1.964   4.95e-02
groupC         -1.7804       0.4386   -4.059   4.92e-05
weight:groupB   0.0643       0.0674    0.954   3.40e-01
weight:groupC   0.0796       0.0674    1.180   2.38e-01

Scale fixed at 1

Exponential distribution
Loglik(model)= -480.6  Loglik(intercept only)= -502.1
        Chisq= 43.11 on 5 degrees of freedom, p= 3.5e-08
Number of Newton-Raphson Iterations: 5
n= 150
```

Model simplification proceeds in the normal way. You could use update, but here (for variety only) we refit progressively simpler models and test them using anova. First we take out the different slopes for each group:

```
model2 <- survreg(Surv(death,status)~weight+group,dist="exponential")
anova(model,model2,test="Chi")
```

```
       Terms Resid. Df    -2*LL          Test Df   Deviance    P(>|Chi|)
1   weight * group   144 961.1800                NA         NA          NA
2   weight + group   146 962.9411  -weight:group -2  -1.761142   0.4145462
```

The interaction is not significant so we leave it out and try deleting weight:

```
model3 <- survreg(Surv(death,status)~group,dist="exponential")
anova(model2,model3,test="Chi")
```

```
           Terms Resid. Df    -2*LL      Test Df     Deviance      P(>|Chi|)
1    weight + group     146 962.9411          NA           NA             NA
2             group     147 963.9393   -weight -1   -0.9981333    0.3177626
```

This is not significant, so we leave it out and try deleting group:

```
model4 <- survreg(Surv(death,status)~1,dist="exponential")
anova(model3,model4,test="Chi")
```

```
    Terms Resid. Df       -2*LL   Test Df     Deviance         P(>|Chi|)
1    group      147    963.9393        NA           NA                NA
2        1      149   1004.2865        -2    -40.34721    1.732661e-09
```

This is highly significant, so we add it back. The minimal adequate model is `model3` with the three-level factor *group*, but there is no evidence that initial body *weight* had any influence on survival.

```
summary(model3)
```

```
Call:
survreg(formula = Surv(death, status) ~ group, dist = "exponential")
               Value    Std. Error       z           p
(Intercept)    3.467         0.167   20.80    3.91e-96
groupB        -0.671         0.225   -2.99    2.83e-03
groupC        -1.386         0.219   -6.34    2.32e-10

Scale fixed at 1

Exponential distribution
Loglik(model)= -482  Loglik(intercept only)= -502.1
        Chisq= 40.35 on 2 degrees of freedom, p= 1.7e-09
Number of Newton-Raphson Iterations: 5
n= 150
```

We need to retain all three groups (group B is significantly different from both group A and group C).

It is straightforward to compare error distributions for the same model structure:

```
model3 <- survreg(Surv(death,status)~group,dist="exponential")
model4 <- survreg(Surv(death,status)~group,dist="extreme")
model5 <- survreg(Surv(death,status)~group,dist="gaussian")
model6 <- survreg(Surv(death,status)~group,dist="logistic")
anova(model3,model4,model5,model6)
```

```
    Terms Resid. Df       -2*LL  Test Df   Deviance    Pr(>Chi)
1   group       147    963.9393       NA         NA          NA
2   group       146   1225.3512     =  1 -261.411949         NA
3   group       146   1178.6582     =  0   46.692975         NA
4   group       146   1173.9478     =  0    4.710457         NA
```

Our initial choice of exponential was clearly the best, giving much the lowest residual deviance (963.94).

You can immediately see the advantage of doing proper survival analysis when you compare the predicted mean ages at death from model3 with the crude arithmetic averages of the raw data on age at death:

```
tapply(predict(model3,type="response"),group,mean)
```

```
        A          B        C
32.05555   16.38635     8.02
```

```
tapply(death,group,mean)
```

```
    A       B       C
23.08   14.42    8.02
```

If there is no censoring (as in group C, where all the individuals died) then the estimated mean ages at death are identical. But when there is censoring, the arithmetic mean underestimates the age at death, and when the censoring is substantial (as in group A) this underestimate is very large (23.08 vs. 32.06 months).

Further Reading

Cox, D.R. and Oakes, D. (1984) *Analysis of Survival Data*, Chapman & Hall, London.

Kalbfleisch, J. and Prentice, R.L. (1980) *The Statistical Analysis of Failure Time Data*, John Wiley & Sons, New York.

Appendix

Essentials of the R Language

R as a Calculator

The command line after the screen prompt > is an excellent calculator:

```
> log(42/7.3)
[1] 1.749795
```

By default, logs in R are base *e* (natural or Napierian, not base 10 logs), but you can specify any base you want, as a second argument to the `log` function. Here is log base 2 of 16:

```
> log(16,2)
[1] 4
```

Each line can have many characters, but if you want to evaluate a complicated expression, you might like to continue it on one or more further lines for clarity. The way you do this is by ending the line at a place where the line is obviously incomplete (e.g. with a trailing comma, operator, or with more left parentheses than right parentheses, implying that more right parentheses will follow). When continuation is expected, the line prompt changes from > to +

```
> 5+6+3+6+4+2+4+8+
+          3+2+7

[1] 50
```

Note that the + continuation prompt does not carry out arithmetic plus. If you have made a mistake, and you want to get rid of the + prompt and return to the > prompt, then press the Esc key and use the Up arrow to edit the last (incomplete) line.

From here onwards (and throughout the book), the prompt character > is omitted. The material that you should type on the command line is shown in `red` font in this book. Just press the Return key to see the answer. The output from R is shown in dark blue

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.

`Courier New` font, which uses absolute rather than proportional spacing, so that columns of numbers remain neatly aligned on the page and on the screen.

Two or more expressions can be placed on a single line so long as they are separated by semicolons:

```
2+3; 5*7; 3-7

[1] 5
[1] 35
[1] -4
```

All of the usual calculations can be done directly, and the standard order of precedence applies: powers and roots are evaluated first, then multiplication and division, then finally addition and subtraction. Although there is a named function for square roots, `sqrt`, roots are generally calculated as fractional powers. Exponentiation (powers) uses the caret ('hat') `^` operator (not `**` as in some computing languages like Fortran or GLIM). So the cube root of 8 is

```
8^(1/3)

[1] 2
```

The exponentiation operator `^` groups *right to left*, but all other operators group left to right. Thus, `2 ^ 2 ^ 3` is `2 ^ 8`, not `4 ^ 3`, whereas `1 - 1 - 1` is `- 1,` not `1.` Use brackets as necessary to override the precedence. For a *t* test, where the value required is $\frac{|y_A - y_B|}{SE_{\text{diff}}}$ and the numbers are 5.7, 6.8 and 0.38, you type:

```
abs(5.7-6.8)/0.38

[1] 2.894737
```

The default number of digits printed out by R is 7 (as above). You can control the number of digits printed using the `digits` option like this:

```
options(digits=3)
abs(5.7-6.8)/0.38

[1] 2.89
```

Built-in Functions

All the mathematical functions you could ever want are here (Table A.1). We have already met the `log` function; the antilog function to the base *e* is `exp:`

```
exp(1)

[1] 2.718282
```

The trigonometric functions in R measure angles in radians. A circle is 2π radians, and this is 360°, so a right-angle (90°) is $\pi/2$ radians. R knows the value of π as `pi`:

```
pi

[1] 3.141593

sin(pi/2)

[1] 1

cos(pi/2)

[1] 6.123032e-017
```

Notice that the cosine of a right-angle does not come out as exactly zero, even though the sine came out as exactly 1. The `e-017` means 'times 10^{-17} '. While this is a very small number it is clearly not exactly zero (so you need to be careful when testing for exact equality of real numbers; see p. 313).

Table A.1 Mathematical functions used in R

Function	Meaning
`log(x)`	log to base e of x
`exp(x)`	antilog of x (=2.7818^x)
`log(x,n)`	log to base n of x
`log10(x)`	log to base 10 of x
`sqrt(x)`	square root of x
`factorial(x)`	$x!$
`choose(n,x)`	binomial coefficients $n!/(x!\,(n-x)!)$
`gamma(x)`	$\Gamma(x)$ $(x-1)!$ for integer x
`lgamma(x)`	natural log of `gamma(x)`
`floor(x)`	greatest integer $< x$
`ceiling(x)`	smallest integer $> x$
`trunc(x)`	closest integer to x between x and 0: `trunc`(1.5) = 1, `trunc`(−1.5) = −1 `trunc` is like `floor` for positive values and like `ceiling` for negative values
`round(x, digits=0)`	round the value of x to an integer
`signif(x, digits=6)`	give x to six digits in scientific notation
`runif(n)`	generates n random numbers between 0 and 1 from a uniform distribution
`cos(x)`	cosine of x in radians
`sin(x)`	sine of x in radians

(*continued*)

Table A.1 (*Continued*)

Function	Meaning
`tan(x)`	tangent of *x* in radians
`acos(x), asin(x), atan(x)`	inverse trigonometric transformations of real or complex numbers.
`acosh(x), asinh(x), atanh(x)`	inverse hyperbolic trigonometric transformations on real or complex numbers
`abs(x)`	the absolute value of *x*, ignoring the minus sign if there is one

Numbers with Exponents

For very big numbers or very small numbers R uses the following scheme:

`1.2e3`	means 1200 because the e3 means 'move the decimal point 3 places to the right'
`1.2e-2`	means 0.012 because the e-2 means 'move the decimal point 2 places to the left'
`3.9+4.5i`	is a complex number with real (3.9) and imaginary (4.5) parts, and *i* is the square root of −1

Modulo and Integer Quotients

Integer quotients and remainders are obtained using the notation `%/%` (percent, divide, percent) and `%%` (percent, percent) respectively. Suppose we want to know the integer part of a division – say, how many 13s are there in 119:

```
119 %/% 13

[1] 9
```

Now suppose we wanted to know the remainder (what is left over when 119 is divided by 13), known in maths as *modulo*:

```
119 %% 13

[1] 2
```

Modulo is very useful for testing whether numbers are odd or even: odd numbers are 1 modulo 2 and even numbers are 0 modulo 2:

```
9 %% 2

[1] 1
8 %% 2

[1] 0
```

Likewise, you use modulo to test if one number is an exact multiple of some other number. For instance, to find out whether 15 421 is a multiple of 7, type:

```
15421 %% 7 == 0
[1] TRUE
```

Assignment

Objects obtain values in R by assignment ('*x gets* a value'). This is achieved by the *gets arrow* which is a composite symbol made up from 'less than' and 'minus' <- without a space between them. Thus, to create a scalar constant *x* with value 5 we type

```
x <- 5
```

and not `x = 5`. Notice that there is a potential ambiguity if you get the spacing wrong. Compare our `x <- 5` meaning '*x* gets 5' and `x < -5` which is a logical question, asking 'is *x* less than minus 5?' and producing the answer TRUE or FALSE.

Rounding

Various sorts of rounding (rounding up, rounding down, rounding to the nearest integer) can be done easily. Take 5.7 as an example. The 'greatest integer less than' function is `floor`:

```
floor(5.7)
[1] 5
```

The 'next integer' function is `ceiling`:

```
ceiling(5.7)
[1] 6
```

You can round to the nearest integer by adding 0.5 to the number then using `floor`. There is a built-in function for this, but we can easily write one of our own to introduce the notion of writing functions. Call it `rounded`, then define it as a function like this:

```
rounded <- function(x) floor(x+0.5)
```

Now we can use the new function:

```
rounded(5.7)
[1] 6
rounded(5.4)
[1] 5
```

Infinity and Things that Are Not a Number (NaN)

Calculations can lead to answers that are plus infinity, represented in R by `Inf`:

```
3/0
```

```
[1] Inf
```

or minus infinity, which is `-Inf` in R:

```
-12/0
```

```
[1] -Inf
```

Calculations involving infinity can be evaluated:

```
exp(-Inf)
```

```
[1] 0
```

```
0/Inf
```

```
[1] 0
```

```
(0:3)^Inf
```

```
[1]  0  1 Inf Inf
```

The syntax `0:3` is very useful. It generaters a series (0, 1, 2, 3 in this case) and the function is evaluated for each of the values to produce a **vector** of answers. In this case, the vector is of `length = 4`.

Other calculations, however, lead to quantities that are not numbers. These are represented in R by `NaN` ('not a number'). Here are some of the classic cases:

```
0/0
```

```
[1] NaN
```

```
Inf-Inf
```

```
[1] NaN
```

```
Inf/Inf
```

```
[1] NaN
```

You need to understand clearly the distinction between `NaN` and `NA` (this stands for 'not available' and is the missing value symbol in R; see below). There are built-in tests to check whether a number is finite or infinite:

```
is.finite(10)
```

```
[1] TRUE
```

```
is.infinite(10)
```

```
[1] FALSE
```

```
is.infinite(Inf)
```

```
[1] TRUE
```

Missing Values (NA)

Missing values in dataframes are a real source of irritation because they affect the way that model-fitting functions operate, and they can greatly reduce the power of the modelling that we would like to do.

Some functions do not work with their default settings when there are missing values in the data, and mean is a classic example of this:

```
x <- c(1:8,NA)
mean(x)
```

```
[1] NA
```

In order to calculate the mean of the non-missing values, you need to specify that the NA are to be removed, using the na.rm=TRUE argument:

```
mean(x,na.rm=T)
```

```
[1] 4.5
```

To check for the location of missing values within a vector, use the function is.na(x). Here is an example where we want to find the locations (7 and 8) of missing values within a vector called vmv:

```
(vmv <- c(1:6,NA,NA,9:12))
```

```
 [1]  1  2  3  4  5  6 NA  NA  9 10 11 12
```

Note the use of round brackets to get the answer printed as well as allocated. Making an index of the missing values in an array could use the seq function:

```
seq(along=vmv)[is.na(vmv)]
```

```
 [1] 7 8
```

However, the result is achieved more simply using which like this:

```
which(is.na(vmv))
```

```
[1] 7 8
```

If the missing values are genuine counts of zero, you might want to edit the NA to 0. Use the is.na function to generate subscripts for this:

```
(vmv[is.na(vmv)] <- 0)
```

```
 [1]  1  2  3  4  5 6  0  0  9 10 11 12
```

Alternatively, use the `ifelse` function like this:

```
vmv <- c(1:6,NA,NA,9:12)
ifelse(is.na(vmv),0,vmv)
```

```
[1]  1  2  3  4  5  6  0  0  9 10 11 12
```

Operators

R uses the following operator tokens

`+ - * / %% ^`	arithmetic
`> >= < <= == !=`	relational
`! & |`	logical
`~`	model formulae
`<- ->`	assignment
`$`	list indexing (the ‘element name’ operator)
`:`	create a sequence

Several of the operators have different meaning inside model formulas. Thus in a model formula `*` indicates the main effects plus interaction, `:` indicates the interaction between two variables and `^` means all interactions up to the indicated power.

Creating a Vector

Vectors are variables with one or more values of the same type: logical, integer, real, complex, string (or character) or raw. Values can be assigned to vectors in many different ways. They can be generated by R: here the vector called *y* gets the sequence of integer values 10 to 16 using `:` (colon), the sequence operator:

```
y <- 10:16
```

You can type the values into the command line, using the *concatenation function* `c`:

```
y <- c(10, 11, 12, 13, 14, 15, 16)
```

Alternatively, you can enter then numbers from the keyboard one at a time using `scan`:

```
y <- scan()
```

```
1: 10
2: 11
3: 12
4: 13
5: 14
6: 15
7: 16
8:
Read 7 items
```

Press the Enter key twice to indicate that data input is complete. However, the commonest way to allocate values to a vector is to read the data from an external file, using `read.csv` or `read.table` (p. 25).

Named Elements within Vectors

It is often useful to have the values in a vector labelled in some way. For instance, if our data are counts of 0, 1, 2, . . . occurrences in a vector called `counts`,

```
(counts <- c(25,12,7,4,6,2,1,0,2))

[1] 25 12  7   4   6   2   1   0   2
```

so that there were 25 zeros, 12 ones and so on, it would be useful to name each of the counts with the relevant number 0, 1, . . . , 8:

```
names(counts) <- 0:8
```

Now when we inspect the vector called counts we see both the names and the frequencies:

```
counts

 0  1  2 3 4 5 6 7 8
25 12  7 4 6 2 1 0 2
```

If you have computed a `table` of counts, and you want to *remove* the names, then use the `as.vector` function like this:

```
(st <- table(rpois(2000,2.3)))

  0   1   2   3   4   5  6  7 8 9
205 455 510 431 233 102 43 13 7 1

as.vector(st)

 [1] 205  455  510  431  233  102  43  13   7   1
```

Vector Functions

One of the great strengths of R is its ability to evaluate functions over entire vectors, thereby avoiding the need for loops and subscripts. The most important vector functions are shown in Table A.2.

Table A.2 Vector functions in R

Operation	Meaning
`max(x)`	maximum value in *x*
`min(x)`	minimum value in *x*
`sum(x)`	total of all the values in *x*
`mean(x)`	arithmetic average of the values in *x*
`median(x)`	median value in *x*
`range(x)`	vector of `min(x)` and `max(x)`
`var(x)`	sample variance of *x*, with degrees of freedom = `length(x) − 1`
`cor(x,y)`	correlation between vectors *x* and *y*
`sort(x)`	a sorted version of *x*
`rank(x)`	vector of the ranks of the values in *x*
`order(x)`	an integer vector containing the permutation to sort *x* into ascending order
`quantile(x)`	vector containing the minimum, lower quartile, median, upper quartile, and maximum of *x*
`cumsum(x)`	vector containing the sum of all of the elements up to that point
`cumprod(x)`	vector containing the product of all of the elements up to that point
`cummax(x)`	vector of non-decreasing numbers which are the cumulative maxima of the values in *x* up to that point.
`cummin(x)`	vector of non-increasing numbers which are the cumulative minima of the values in *x* up to that point.
`pmax(x,y,z)`	vector, of length equal to the longest of *x*, *y*, or *z* containing the maximum of *x*, *y* or *z* for the *i*th position in each
`pmin(x,y,z)`	vector, of length equal to the longest of *x*, *y*, or *z* containing the minimum of *x*, *y* or *z* for the *i*th position in each
`colMeans(x)`	column means of dataframe or matrix *x*
`colSums(x)`	column totals of dataframe or matrix *x*
`rowMeans(x)`	row means of dataframe or matrix *x*
`rowSums(x)`	row totals of dataframe or matrix *x*

Summary Information from Vectors by Groups

One of the most important and useful vector functions to master is `tapply`. The ‘t’ stands for ‘table’ and the idea is to `apply` a function to produce a table from the values in the vector, based on one or more grouping variables (often the grouping is by factor levels). This sounds much more complicated than it really is:

```
data <- read.csv("c:\\temp\\daphnia.csv")
attach(data)
names(data)
```

```
[1] "Growth.rate" "Water"   "Detergent"  "Daphnia"
```

The response variable is Growth.rate and the other three variables are factors. Suppose we want the mean growth rate for each Detergent:

```
tapply(Growth.rate,Detergent,mean)
```

```
BrandA  BrandB  BrandC  BrandD
  3.88    4.01    3.95    3.56
```

This produces a table with four entries, one for each level of the factor called Detergent. To produce a two-dimensional table we put the two grouping variables in a list. Here we calculate the median growth rate for Water type and Daphnia clone:

```
tapply(Growth.rate,list(Water,Daphnia),median)
```

```
     Clone1 Clone2 Clone3
Tyne   2.87   3.91   4.62
Wear   2.59   5.53   4.30
```

The first variable in the list creates the rows of the table and the second the columns.

Subscripts and Indices

While we typically aim to apply functions to vectors as a whole, there are circumstances where we want to select only some of the elements of a vector. This selection is done using *subscripts* (also known as *indices*). Subscripts have square brackets [2], while functions have round brackets (2). Subscripts on vectors, matrices, arrays and dataframes have one set of square brackets [6], [3,4] or [2,3,2,1], while subscripts on lists have double square brackets [[2]] or [[i,j]] (see p. 309). When there are two subscripts to an object like a matrix or a dataframe, the first subscript refers to the *row* number (the rows are defined as margin number 1) and the second subscript refers to the *column* number (the columns are margin number 2). There is an important and powerful convention in R such that *when a subscript appears as a blank it is understood to mean 'all of'*. Thus

- [,4] means all rows in column 4 of an object
- [2,] means all columns in row 2 of an object

There is another indexing convention in R which is used to extract named components from objects using the $ operator like this: model$coef or model$resid (p. 317). This is known as 'indexing tagged lists' using the *element names operator* $.

Working with Vectors and Logical Subscripts

Take the example of a vector containing the 11 numbers 0 to 10:

```
x <- 0:10
```

There are two quite different kinds of things we might want to do with this. We might want to *add up* the values of the elements:

```
sum(x)
[1] 55
```

Alternatively, we might want to *count* the elements that passed some logical criterion. Suppose we wanted to know how many of the values were less than 5:

```
sum(x<5)
[1] 5
```

You see the distinction. We use the vector function `sum` in both cases. But `sum(x)` adds up the values of the *x*s and `sum(x<5)` counts up the number of cases that pass the logical condition '*x* is less than 5'. This works because of *coercion* (p. 314). Logical TRUE has been coerced to numeric 1 and logical FALSE has been coerced to numeric 0.

That's all well and good, but how do you add up the values of just some of the elements of *x*? We specify a logical condition, but we don't want to count the number of cases that pass the condition, we want to add up all the values of the cases that pass. This is the final piece of the jigsaw, and involves the use of *logical subscripts*. Note that when we *counted* the number of cases, the counting was applied to the entire vector, using `sum(x<5).` To find the sum of the *x* values that are less than 5, we write:

```
sum(x[x<5])
[1] 10
```

Let us look at this in more detail. The logical condition `x<5` is either true or false:

```
x<5
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
[10] FALSE FALSE
```

You can imagine false as being numeric 0 and true as being numeric 1. Then the vector of subscripts `[x<5]` is five 1s followed by six 0s:

```
1*(x<5)
 [1] 1 1 1 1 1 0 0 0 0 0 0
```

Now imagine multiplying the values of *x* by the values of the logical vector:

```
x*(x<5)
 [1] 0 1 2 3 4 0 0 0 0 0 0
```

When the function `sum` is applied, it gives us the answer we want: the sum of the values of the numbers $0+1+2+3+4=10$.

```
sum(x*(x<5))
[1] 10
```

This produces the same answer as `sum(x[x<5]),` but is rather less elegant. Suppose we want to work out the sum of the three largest values in a vector. There are two steps: first sort the vector into descending order; then add up the values of the first three elements of the sorted array. Let us do this in stages. First, the values of *y*:

```
y <- c(8,3,5,7,6,6,8,9,2,3,9,4,10,4,11)
```

Now if you apply `sort` to this, the numbers will be in ascending sequence, and this makes life slightly harder for the present problem:

```
sort(y)
 [1] 2 3 3 4 4 5 6 6 7 8 8 9 9 10 11
```

We can use the *reverse* function `rev` like this (use the Up arrow key to save typing):

```
rev(sort(y))
 [1] 11 10 9 9 8 8 7 6 6 5 4 4 3 3 2
```

So the answer to our problem is $11+10+9=30$. But how to compute this? We can use specific subscripts to discover the contents of any element of a vector. We can see that 10 is the second element of the sorted array. To compute this we just specify the subscript [2]:

```
rev(sort(y))[2]
[1] 10
```

A range of subscripts is simply a series generated using the colon operator. We want the subscripts 1 to 3, so this is:

```
rev(sort(y))[1:3]
 [1] 11 10 9
```

So the answer to the exercise is just:

```
sum(rev(sort(y))[1:3])
[1] 30
```

Note that we have not changed the vector *y* in any way, nor have we created any new space-consuming vectors during intermediate computational steps.

Addresses within Vectors

There are two important functions for finding addresses within arrays. The function `which` is very easy to understand. The vector *y* (see above) looks like this:

```
y
[1] 8 3 5 7 6 6 8 9 2 3 9 4 10 4 11
```

Suppose we wanted to know the addresses of the elements of *y* that contained values bigger than 5:

```
which(y>5)
[1] 1 4 5 6 7 8 11 13 15
```

Notice that the answer to this enquiry is *a set of subscripts*. We *don't* use subscripts inside the `which` function itself. The function is applied to the whole array. To see the *values of y* that are larger than 5 we just type:

```
y[y>5]
[1] 8 7 6 6 8 9 9 10 11
```

Note that this is a shorter vector than *y* itself, because values of 5 or less have been left out.

```
length(y)
[1] 15
length(y[y>5])
[1] 9
```

Trimming Vectors Using Negative Subscripts

An extremely useful facility is to use *negative subscripts* to drop terms from a vector. Suppose we wanted a new vector, *z*, to contain everything but the first element of *x*:

```
x <- c(5,8,6,7,1,5,3)
z <- x[-1]
z
[1] 8 6 7 1 5 3
```

Our task is to calculate a trimmed mean of *x* which ignores both the smallest and largest values (i.e. we want to leave out the 1 and the 8 in this example). There are two steps to this. First we `sort` the vector *x*. Then we remove the first element using `x[-1]` and the last using `x[-length(x)]`. We can do both drops at the same time by concatenating both instructions like this: `-c(1,length(x))`. Then we use the built-in function `mean`:

```
trim.mean <- function (x) mean(sort(x)[-c(1,length(x))])
```

The answer should be `mean(c(5,6,7,5,3))` = 26/5 = 5.2. Let us try it out:

```
trim.mean(x)
```

```
[1] 5.2
```

Logical Arithmetic

Arithmetic involving logical expressions is very useful in programming and in selection of variables (see Table A.3). If logical arithmetic is unfamiliar to you, then persevere with it, because it will become clear how useful it is, once the penny has dropped. The key thing to understand is that logical expressions evaluate to either true or false (represented in R by TRUE or FALSE), and that R can coerce TRUE or FALSE into numerical values: 1 for TRUE and 0 for FALSE.

Table A.3 Logical operations

Symbol	Meaning
`!`	logical not
`&`	logical and
`|`	logical or
`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`==`	logical equals (double =)
`!=`	not equal
`&&`	and with if
`||`	or with if
`xor(x,y)`	exclusive or
`isTRUE(x)`	an abbreviation of `identical(TRUE,x)`

Repeats

You will often want to generate repeats of numbers or characters, for which the function is `rep`. The object that is named in the first argument is repeated a number of times as specified in the second argument. At its simplest, we would generate five 9s like this:

```
rep(9,5)
```

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

You can see the issues involved by a comparison of these three increasingly complicated uses of the `rep` function:

```
rep(1:4, 2)
```

```
[1] 1 2 3 4 1 2 3 4
```

```
rep(1:4, each=2)
```

```
[1] 1 1 2 2 3 3 4 4
```

```
rep(1:4, each = 2, times = 3)
```

```
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

When each element of the series is to be repeated a different number of times, then the second argument must be a vector of the same length as the vector comprising the first argument. Suppose that we need four 9s, one 15, four 21s and two 83s. We use the concatenation function c to create two vectors, one for the target numbers and another for the repeats of each element of the target:

```
rep(c(9,15,21,83),c(4,1,4,2))
```

```
[1] 9 9 9 9 15 21 21 21 21 83 83
```

Generate Factor Levels

The function gl ('generate levels') is useful when you want to encode long vectors of factor levels. The syntax for the three arguments is this:

gl('up to', 'with repeats of', 'to total length')

Here is the simplest case where we want factor levels up to 4 with repeats of 3 repeated only once (i.e. to total length = 12):

```
gl(4,3)
```

```
  [1] 1 1 1 2 2 2 3 3 3 4 4 4
Levels: 1 2 3 4
```

Here is the function when we want that whole pattern repeated twice:

```
gl(4,3,24)
```

```
  [1] 1 1 1 2 2 2 3 3 3 4 4 4 1 1 1 2 2 2 3 3 3 4 4 4
Levels: 1 2 3 4
```

If the total length is not a multiple of the length of the pattern, the vector is truncated:

```
gl(4,3,20)
```

```
  [1] 1 1 1 2 2 2 3 3 3 4 4 4 1 1 1 2 2 2 3 3
Levels: 1 2 3 4
```

If you want text for the factor levels, rather than numbers, use labels like this:

```
gl(3,2,24,labels=c("A","B","C"))
```

```
[1] A A B B C C A A B B C C A A B B C C A A B B C C
Levels: A B C
```

Generating Regular Sequences of Numbers

For regular series of whole numbers use the colon operator (: as on p. 296). When the interval is not 1.0 you need to use the seq function. In general, the three arguments to seq are: initial value, final value, and increment (or decrement for a declining sequence). Here we want to go from 0 up to 1.5 in steps of 0.2:

```
seq(0,1.5,0.2)
```

```
[1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4
```

Note that `seq` stops *before* it gets to the second argument (1.5) if the increment does not match exactly (our sequence stops at 1.4). If you want to `seq` downwards, the third argument needs to be negative

```
seq(1.5,0,-0.2)
```

```
[1] 1.5 1.3 1.1 0.9 0.7 0.5 0.3 0.1
```

Again, zero did not match the decrement, so was excluded and the sequence stopped at 0.1. If you want to create a sequence of the same `length` as an existing vector, then use `along` like this. Suppose that our existing vector, *x*, contains 18 random numbers from a normal distribution with a mean of 10.0 and a standard deviation of 2.0:

```
x <- rnorm(18,10,2)
```

Suppose also that we want to generate a sequence of the same length as this (18) starting at 88 and stepping down to exactly 50 for `x[18]`:

```
seq(88,50,along=x)
```

```
 [1] 88.00000 85.76471 83.52941 81.29412 79.05882 76.82353 74.58824 72.35294
 [9] 70. 11765 67.88235 65.64706 63.41176 61.17647 58.94118 56.70588 54.47059
[17] 52.23529 50.00000
```

This is useful when you do not want to go to the trouble of working out the size of the increment but you do know the starting value (88 in this case) and the final value (50).

Matrices

There are several ways of making a matrix. You can create one directly like this:

```
X <- matrix(c(1,0,0,0,1,0,0,0,1),nrow=3)
X
```

```
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

Here, by default, the numbers are entered columnwise. The `class` and `attributes` of *X* indicate that it is a matrix of 3 rows and 3 columns (these are its `dim` attributes):

```
class(X)

[1] "matrix"

attributes(X)

$dim
[1] 3 3
```

In the next example, the data in the vector appear row-wise, so we indicate this with byrow=T:

```
vector <- c(1,2,3,4,4,3,2,1)
V <- matrix(vector,byrow=T,nrow=2)
V

     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    4    3    2    1
```

Another way to convert a vector into a matrix is by providing the vector object with two dimensions (rows and columns) using the dim function like this:

```
dim(vector) <- c(4,2)
```

and we can check that vector has now become a matrix:

```
is.matrix(vector)

[1] TRUE
```

We need to be careful, however, because we have made no allowance at this stage for the fact that the data were entered row-wise into vector:

```
vector

  [,1] [,2]
[1,]  1  4
[2,]  2  3
[3,]  3  2
[4,]  4  1
```

The matrix we want is the transpose, t, of this matrix:

```
(vector <- t(vector))

  [,1] [,2] [,3] [,4]
[1,]  1  2  3  4
[2,]  4  3  2  1
```

Character Strings

In R, character strings are defined by double quotation marks. We being by defining a phrase:

```
phrase <- "the quick brown fox jumps over the lazy dog"
```

The function called `substr` is used to extract substrings of a specified number of characters from a character string. Here is the code to extract the first, the first and second, the first, second and third, and so on (up to 20) characters from our phrase:

```
q <- character(20)
for (i in 1:20) q[i] <- substr(phrase,1,i)
q
```

```
[1]  "t"                    "th"                    "the"
[4]  "the "                 "the q"                 "the qu"
[7]  "the qui"              "the quic"              "the quick"
[10] "the quick "           "the quick b"           "the quick br"
[13] "the quick bro"        "the quick brow"        "the quick brown"
[16] "the quick brown "     "the quick brown f"     "the quick brown fo"
[19] "the quick brown fox"  "the quick brown fox "
```

The second argument in `substr` is the number of the character at which extraction is to begin (in this case always the first), and the third argument is the number of the character at which extraction is to end (in this case, the *i*th). To split up a character string into individual characters, we use `strsplit` like this:

```
strsplit(phrase,split=character(0))
```

```
[[1]]
 [1]  "t" "h" "e" " " "q" "u" "i" "c" "k" " " "b" "r" "o" "w" "n" " "
[17]  "f" "o" "x" " " "j" "u" "m" "p" "s" " " "o" "v" "e" "r"
[31]  " " "t" "h" "e" " " "l" "a" "z" "y" " " "d" "o" "g"
```

The `table` function is useful for counting the number of occurrences of characters of different kinds:

```
table(strsplit(phrase,split=character(0)))
```

```
  a b c d e f g h i j k l m n o p q r s t u v w x y z
8 1 1 1 1 3 1 1 2 1 1 1 1 1 1 4 1 1 2 1 2 2 1 1 1 1 1
```

This demonstrates that all of the letters of the alphabet were used at least once within our phrase, and that there were eight blanks within phrase. This suggests a way of counting the number of words in a phrase, given that this will always be one more than the number of blanks:

```
words <- 1+table(strsplit(phrase,split=character(0)))[1]
words
```

```
9
```

It is easy to switch between upper and lower cases using the `toupper` and `tolower` functions:

```
toupper(phrase)
```

```
[1] "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG"
```

```
tolower(toupper(phrase))
```

```
[1] "the quick brown fox jumps over the lazy dog"
```

Writing Functions in R

Functions in R are objects that carry out operations on *arguments* that are supplied to them and return one or more values. The syntax for writing a function is

```
  function (argument list) body
```

The first component of the function declaration is the keyword `function` which indicates to R that you want to create a function. An argument list is a comma-separated list of formal arguments. A formal argument can be a symbol (i.e. a variable name like *x* or *y*), a statement of the form `symbol = expression` (e.g. `pch=16`) or the special formal argument `...` (dot dot dot). The body can be any valid R expression or set of R expressions. Generally, the body is a group of expressions contained in curly brackets `{ }` with each expression on a separate line. Functions are typically assigned to symbols, but they do not need to be. This will only begin to mean anything after you have seen several examples in operation.

Arithmetic Mean of a Single Sample

The mean is the sum of the numbers $\sum y$ divided by the number of numbers $n = \sum 1$ (summing over the number of numbers in the vector called *y*). The R function for *n* is `length(y)` and for $\sum y$ is `sum(y)`, so a function to compute arithmetic means is:

```
arithmetic.mean <- function(x)  sum(x)/length(x)
```

We should test the function with some data where we know the right answer:

```
y <- c(3,3,4,5,5)
arithmetic.mean(y)
```

```
[1] 4
```

Needless to say, there is a built- in function for arithmetic means called `mean`:

```
mean(y)
```

```
[1] 4
```

Median of a Single Sample

The median (or 50th percentile) is the middle value of the sorted values of a vector of numbers:

```
sort(y)[ceiling(length(y)/2)
```

There is slight hitch here, of course, because if the vector contains an even number of numbers, then there *is* no middle value. The logic is that we need to work out the arithmetic average of the two values of *y* on either side of the middle. The question now arises as to how we know, in general, whether the vector *y* contains an odd or an even number of numbers, so that we can decide which of the two methods to use. The trick here is to use modulo 2 (p. 46). Now we have all the tools we need to write a general function to calculate medians. Let us call the function `med` and define it like this:

```
med <- function(x) {
odd.even <- length(x)%%2
if (odd.even == 0) (sort(x)[length(x)/2]+sort(x)[1+ length(x)/2])/2
else sort(x)[ceiling(length(x)/2)]
}
```

Notice that when the `if` statement is true (i.e. we have an even number of numbers) then the expression immediately following the `if` function is evaluated (this is the code for calculating the median with an even number of numbers). When the `if` statement is false (i.e. we have an odd number of numbers, and `odd.even == 1`) then the expression following the `else` function is evaluated (this is the code for calculating the median with an odd number of numbers). Let us try it out, first with the odd-numbered vector `y`, then with the even-numbered vector `y[-1]`, after the first element of y (which is `y[1]` = 3) has been dropped (using the negative subscript):

```
med(y)

[1] 4

med(y[-1])

[1] 4.5
```

Again, you won't be surprised that there is a built-in function for calculating medians, and helpfully it is called `median`.

Loops and Repeats

The classic, Fortran-like loop is available in R. The syntax is a little different, but the idea is identical; you request that an index, *i*, takes on a sequence of values, and that one or more lines of commands are executed as many times as there are different values of *i*. Here is a loop executed five times with the values of *i*: we print the square of each value

```
for (i in 1:5) print(i^2)

[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

For multiple lines of code, you use curly brackets { } to enclose material over which the loop is to work. Note that the 'hard return' (the Enter key) at the end of each command line is an essential part of the structure (you can replace the hard returns by semicolons if you like, but clarity is improved if you put each command on a separate line).

Here is a function that uses the while function in converting a specified number to its binary representation. The trick is that the smallest digit (0 for even or 1 for odd numbers) is always at the right-hand side of the answer (in location 32 in this case):

```
binary <- function(x)                    {
if (x == 0) return(0)
i <- 0
     string <- numeric(32)
     while(x>0)              {
           string[32-i] <- x %% 2
           x <- x%/% 2
           i <- i+1      }
first <- match(1,string)
string[first:32]              }
```

The leading zeros (1 to first – 1) within the string are not printed. We run the function to find the binary representation of the numbers 15 through 17:

```
sapply(15:17,binary)
```

```
[[1]]
[1] 1 1 1 1

[[2]]
[1] 1 0 0 0 0

[[3]]
[1] 1 0 0 0 1
```

The ifelse Function

Sometimes you want to do one thing if a condition is true and a different thing if the condition is false (rather than do nothing, as in the last example). The ifelse function allows you to do this for entire vectors without using for loops. We might want to replace any negative values of y by -1 and any positive values and zero by $+1$:

```
z <- ifelse( y < 0, -1, 1)
```

Evaluating Functions with apply

The apply function is used for applying functions to the rows (subscript 1) or columns (subscript 2) of matrices or dataframes:

```
(X <- matrix(1:24,nrow=4))
```

```
   [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  1    5    9   13   17   21
[2,]  2    6   10   14   18   22
[3,]  3    7   11   15   19   23
[4,]  4    8   12   16   20   24
```

Here are the row totals (four of them) across margin 1 of the matrix:

```
apply(X,1,sum)
```

```
[1] 66 72 78 84
```

Here are the column totals (six of them) across margin 2 of the matrix:

```
apply(X,2,sum)
```

```
[1] 10 26 42 58 74 90
```

Note that in both cases, the answer produced by `apply` is a vector rather than a matrix. You can apply functions to the individual elements of the matrix rather than to the margins. The margin you specify influences only the shape of the resulting matrix.

```
apply(X,1,sqrt)
```

```
         [,1]     [,2]     [,3]     [,4]
[1,] 1.000000 1.414214 1.732051 2.000000
[2,] 2.236068 2.449490 2.645751 2.828427
[3,] 3.000000 3.162278 3.316625 3.464102
[4,] 3.605551 3.741657 3.872983 4.000000
[5,] 4.123106 4.242641 4.358899 4.472136
[6,] 4.582576 4.690416 4.795832 4.898979
```

```
apply(X,2,sqrt)
```

```
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 1.000000 2.236068 3.000000 3.605551 4.123106 4.582576
[2,] 1.414214 2.449490 3.162278 3.741657 4.242641 4.690416
[3,] 1.732051 2.645751 3.316625 3.872983 4.358899 4.795832
[4,] 2.000000 2.828427 3.464102 4.000000 4.472136 4.898979
```

Testing for Equality

You need to be careful in programming when you want to test whether or not two computed numbers are equal. R will assume that you mean ‘exactly equal’ and what *that* means depends upon machine precision. Most numbers are rounded to 53 binary digits accuracy. Typically therefore, two floating point numbers will *not* reliably be equal unless they were computed by the same algorithm, and not always even then. You can see this by squaring the square root of 2: surely these values are the same?

```
x <- sqrt(2)
x * x == 2
```

```
[1] FALSE
```

We can see by how much the two values differ by subtraction:

```
x * x – 2
```

```
[1] 4.440892e-16
```

Testing and Coercing in R

Objects have a type, and you can test the type of an object using an is.type function (see Table A.4). For instance, mathematical functions expect numeric input and text-processing functions expect character input. Some types of objects can be coerced into other types (again see Table A.4). A familiar type of coercion occurs when we interpret the TRUE and FALSE of logical variables as numeric 1 and 0, respectively. Factor levels can be coerced to numbers. Numbers can be coerced into characters, but non-numeric characters cannot be coerced into numbers.

```
as.numeric(factor(c("a","b","c")))

[1] 1 2 3
as.numeric(c("a","b","c"))

[1] NA NA NA
Warning message:
NAs introduced by coercion
as.numeric(c("a","4","c"))

[1] NA 4 NA
Warning message:
NAs introduced by coercion
```

If you try to coerce complex numbers to numeric the imaginary part will be discarded. Note that is.complex and is.numeric are never both TRUE.

Table A.4 Testing and coercing

Type	Testing	Coercing
Array	is.array	as.array
Character	is.character	as.character
Complex	is.complex	as.complex
Dataframe	is.data.frame	as.data.frame
Double	is.double	as.double
Factor	is.factor	as.factor
Raw	is.raw	as.raw
List	is.list	as.list
Logical	is.logical	as.logical
Matrix	is.matrix	as.matrix
Numeric	is.numeric	as.numeric
Time series (ts)	is.ts	as.ts
Vector	is.vector	as.vector

We often want to coerce tables into the form of vectors as a simple way of stripping off their dimnames (using as.vector), and to turn matrices into dataframes (as.data.frame). A lot of testing involves the ‘not’ operator ! in functions to return an error message if the wrong type is supplied. For instance, if you were writing a function to calculate geometric means you might want to test to ensure that the input was numeric using the !is.numeric function

```
geometric <- function(x){
if(!is.numeric(x)) stop ("Input must be numeric")
exp(mean(log(x))) }
```

Here is what happens when you try to work out the geometric mean of character data:

```
geometric(c("a","b","c"))
```

```
Error in geometric(c("a", "b", "c")) : Input must be numeric
```

You might also want to check that there are no zeros or negative numbers in the input, because it would make no sense to try to calculate a geometric mean of such data:

```
geometric <- function(x){
if(!is.numeric(x)) stop ("Input must be numeric")
if(min(x)<=0) stop ("Input must be greater than zero")
exp(mean(log(x))) }
```

Testing this:

```
geometric(c(2,3,0,4))
```

```
Error in geometric(c(2, 3, 0, 4)) : Input must be greater than zero
```

But when the data are OK there will be no messages, just the numeric answer:

```
geometric(c(10,1000,10,1,1))
```

```
[1] 10
```

Dates and Times in R

The measurement of time is highly idiosyncratic. Successive years start on different days of the week. There are months with different numbers of days. Leap years have an extra day in February. Americans and Britons put the day and the month in different places: 3/4/2006 is 4 March in one case and 3 April in another. Occasional years have an additional ‘leap second’ added to them because friction from the tides is slowing down the rotation of the earth from when the standard time was set on the basis of the tropical year in 1900. The cumulative effect of having set the atomic clock too slow accounts for the continual need to insert leap seconds (32 of them since 1958). There is currently a debate about abandoning leap seconds and introducing a ‘leap minute’ every century or so, instead. Calculations involving times are complicated by the operation of time zones and daylight saving schemes in different countries. All these things mean that working with dates and times is excruciatingly complicated. Fortunately, R has a robust system for dealing with this complexity. To see how R handles dates and times, have a look at `Sys.time()`:

```
Sys.time()
```

```
[1] "2015-03-23 08:56:26 GMT"
```

The answer is strictly hierarchical from left to right: the longest time scale (year = 2015) comes first, then month (March = 03) then day (the 23rd) separated by hyphens, then there is a blank space and the time hours first (08 in the 24-hour clock) then minutes (56), then seconds (26) separated by colons. Finally, there is a character string explaining the time zone (GMT = Greenwich Mean Time). Now obviously, there is a lot going on here. To accommodate this, R uses the Portable Operating System Interface system (POSIX) for representing times and dates:

```
class(Sys.time())
```

```
[1] "POSIXct" "POSIXt"
```

There are two quite different ways of expressing time, each useful in a different context. For doing graphs and regression analyses where time is the explanatory variable, you want time to be a continuous number. But where you want to do summary statistics (say, monthly means across a long run of years), you want month to be a factor with 12 levels. In R this distinction is handled by the two classes `POSIXct` and `POSIXlt`. The first class, with suffix `ct`, you can think of as *continuous time* (i.e. a number of seconds). The other class, with suffix `lt`, you can think of as *list time* (i.e. a named list of vectors, representing all of the various categorical descriptions of the time, including day of the week and so forth). It is hard to remember these acronyms, but it is well worth making the effort. You can see (above) that `Sys.time` is of both class `ct` and class `t`. Times that are of class `lt` are also of class `t`. What this means is that class `POSIXt` tells you that an object is a time, but not what *sort* of time it is; it does not distinguish between continuous time `ct` and list times `lt`. Naturally, you can easily convert from one representation to the other:

```
time.list <- as.POSIXlt(Sys.time())
class(time.list)
```

```
[1] "POSIXlt" "POSIXt"
```

```
unlist(time.list)
```

```
 sec  min hour mday  mon year wday yday isdst
  26   56    8   23    2  115    1   81     0
```

Here you see the nine components of the list of class `POSIXlt`. The time is represented by the number of seconds (`sec`), minutes (`min`) and hours (`hour` on the 24-hour clock). Next comes the day of the month (`mday`, starting from 1), then the month of the year (`mon`, starting (non-intuitively, perhaps) at January = 0; you can think of it as ‘months completed so far’), then the year (starting at 0 = 1900). The day of the week (`wday`) is coded from Sunday = 0 to Saturday = 6. The day within the year (`yday`) is coded from 0 = January 1 (you can think of it as ‘days completed so far’). Finally, there is a logical variable `isdst` which asks whether daylight saving time is in operation (0 = FALSE in this case because we are on Greenwich Mean Time). The ones you are most likely to use include `year` (to get yearly mean values), `mon` (to get monthly means) and `wday` (to get means for the different days of the week, useful for answering questions like ‘are Fridays different from Mondays?’).

You can use the element name operator `$` to extract parts of the date and time from this object using the names: `sec, min, hour, mday, mon, year, wday, yday` and `isdst`. Here we extract the day of the week (`date$wday=0` meaning Sunday) and the Julian date (day of the year after 1 January as `date$yday`):

```
time.list$wday
```

```
[1] 1
```

```
time.list$yday
```

```
[1] 81
```

meaning that today is a Monday and the 82nd day of the year (81 days are completed so far).

It is important to know how to read dates and times into R. Here we illustrate two of the commonest methods:

- Excel dates
- dates stored in separate variables for year, month, day, hour, minute, second

The Excel date convention uses forward slashes to separate the numerical components. The English way to give a date is day/month/year and the American way is month/day/year. Single-figure days and months can have a leading zero. So 03/05/2015 means 3 May in England but 5 March in America. Because of the appearance of the slashes these are character strings rather than numbers, and R will interpret them as factors when they are read from a file into a dataframe. You have to convert this factor into a date–time object using a function called `strptime` (you can remember this as ‘stripping the time’ out of a character string). We start by reading the Excel dates into a dataframe:

```
data <- read.table("c:\\temp\\date.txt",header=T)
head(data)
```

```
   x       date
1  3  15/06/2014
2  1  16/06/2014
3  6  17/06/2014
4  7  18/06/2014
5  8  19/06/2014
6  9  20/06/2014
```

As things stand, the date is not recognized by R:

```
class(date)
```

```
[1] "factor"
```

The strptime function takes the name of the factor (date) and a format statement showing exactly what each element of the character string represents and what symbols are used as separators (forward slashes in this case). In our example, the day of the month comes first (%d) then a slash, then the month (%m) then another slash, then the year in full (%Y; if the year was just the last two digits '15' then the appropriate code would be %y in lower case). We give the translated date a new name like this:

```
Rdate <- strptime(date,"%d/%m/%Y")
class(Rdate)
```

```
[1] "POSIXlt" "POSIXt"
```

That's more like it. Now we can do date-related things. For instance, what is the mean value of *x* for each day of the week? You need to know that the name of the day of the week within the list called Rdate is wday, then

```
tapply(x,Rdate$wday,mean)
```

```
    0     1     2     3     4     5     6
5.660 2.892 5.092 7.692 8.692 9.692 8.892
```

The value was lowest on Mondays (day 1) and highest on Fridays (day 5).

For cases where your data file has several variables representing the components of date or a time (say, hours, minutes and seconds separately), you use the paste function to join the components up into a single character string, using the appropriate separators (hyphens for dates, colons for times). Here is the data file:

```
time <- read.csv("c:\\temp\\times.csv")
attach(time)
head(time)
```

```
  hrs min sec  experiment
1   2  23   6           A
2   3  16  17           A
3   3   2  56           A
4   2  45   0           A
5   3   4  42           A
6   2  56  25           A
```

Create a vector of times *y* using paste with a colon separator:

```
y <- paste(hrs,min,sec,sep=":")
y
```

```
 [1] "2:23:6"  "3:16:17" "3:2:56"  "2:45:0"  "3:4:42"   "2:56:25"
 [7] "3:12:28" "1:57:12" "2:22:22" "1:42:7"  "2:31:17"  "3:15:16"
[13] "2:28:4"  "1:55:34" "2:17:7"  "1:48:48"
```

You then need to convert these character strings into something that R will recognize as a date and/or a time. If you use strptime then R will automatically add today's date to the

POSIXct object. Note that "%T" is a shortcut for "%H:%M:%S". For more details and a full list of codes, see *The R Book* (Crawley, 2013).

```
strptime(y,"%T")
```

```
 [1] "2014-01-22 02:23:06" "2014-01-22 03:16:17" "2014-01-22 03:02:56"
 [4] "2014-01-22 02:45:00" "2014-01-22 03:04:42" "2014-01-22 02:56:25"
     ....
```

When you only have times rather than dates and times, then you may want to use the as.difftime function rather than as.POSIXct to create the variables to work with:

```
( Rtime <- as.difftime(y))
```

```
Time differences in hours
 [1] 2.385000 3.271389 3.048889 2.750000 3.078333 2.940278 3.207778
 [8] 1.953333 2.372778 1.701944 2.521389 3.254444 2.467778 1.926111
[15] 2.285278 1.813333
```

This converts all of the times into decimal hours in this case (it would convert to minutes if there were no hours in the examples). Here is the mean time for each of the two experiments (A and B):

```
tapply(Rtime,experiment,mean)
```

```
       A          B
2.829375  2.292882
```

Calculations with Dates and Times

You can subtract one date from another, but you can*not* add two dates together. The following calculations are allowed for dates and times:

- time + number
- time − number
- time1 − time2
- time1 ‘logical operation’ time2

where the logical operations are one of ==, !=, <, <=, > or >=.

The thing you need to grasp is that you should convert your dates and times into POSIXlt objects *before* starting to do any calculations. Once they are POSIXlt objects, it is straightforward to calculate means, differences and so on. Here we want to calculate the number of days between two dates, 22 October 2003 and 22 October 2005:

```
y2 <- as.POSIXlt("2018-10-22")
y1 <- as.POSIXlt("2015-10-22")
```

Now you can do calculations with the two dates:

```
y2-y1
```

```
Time difference of 1096 days
```

Note that you cannot *add* two dates. It is easy to calculate differences between times using this system. Note that the dates are separated by hyphens whereas the times are separated by colons:

```
y3 <- as.POSIXlt("2018-10-22 09:30:59")
y4 <- as.POSIXlt("2018-10-22 12:45:06")
y4-y3
```

```
Time difference of 3.235278 hours
```

Alternatively, you can do these calculations using the difftime function:

```
difftime("2018-10-22 12:45:06","2018-10-22 09:30:59")
```

```
Time difference of 3.235278 hours
```

Here is an example of logical operation on date time objects. We want to know whether y4 was later than y3:

```
y4 > y3
```

```
[1] TRUE
```

Understanding the Structure of an R Object Using str

We finish with a simple but important function for understanding what kind of object is represented by a given name in the current R session. We look at three objects of increasing complexity: a vector of numbers, a list of various classes and a linear model.

```
x <- runif(23)
str(x)
```

```
 num [1:23] 0.971 0.23 0.645 0.697 0.537 ...
```

We see that x is a number (num) in a vector of length 23 ([1:23]) with the first five values as shown (0.971, . . .).

In this example of intermediate complexity the variable called basket is a list:

```
basket <- list(rep("a",4),c("b0","b1","b2"),9:4,gl(5,3))
basket
```

```
[[1]]
[1] "a" "a" "a" "a"

[[2]]
[1] "b0" "b1" "b2"

[[3]]
[1] 9 8 7 6 5 4

[[4]]
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
Levels: 1 2 3 4 5
```

The first element is a character vector of length 4 (all "a"), while the second has three different character strings. The third is a numeric vector and the fourth is a factor of length 12 with four levels. Here is what you get from str:

```
str(basket)
```

```
List of 4
 $ : chr [1:4] "a" "a" "a" "a"
 $ : chr [1:3] "b0" "b1" "b2"
 $ : int [1:6] 9 8 7 6 5 4
 $ : Factor w/ 5 levels "1","2","3","4",..: 1 1 1 2 2 2 3 3 3 4 ...
```

First you see the length of the list (4) then the attributes of each of the four components: the first two are character strings (chr), the third contains integers (int) and the fourth is a factor with five levels.

Finally, we look at an example of a complex structure – a quadratic linear model:

```
xv <- seq(0,30)
yv <- 2+0.5*xv+rnorm(31,0,2)
model <- lm(yv~xv+I(xv^2))
str(model)
```

```
List of 12
 $ coefficients : Named num [1:3] 2.51317 0.38809 0.00269
  ..- attr(*, "names")= chr [1:3] "(Intercept)" "xv" "I(xv^2)"
 $ residuals   : Named num [1:31] -1.712 -1.869 4.511 0.436 -0.207 ...
  ..- attr(*, "names")= chr [1:31] "1" "2" "3" "4" ...
 $ effects     : Named num [1:31] -50.965 23.339 -1.068 0.646 0.218 ...
  ..- attr(*, "names")= chr [1:31] "(Intercept)" "xv" "I(xv^2)" "" ...
 $ rank        : int 3
 $ fitted.values: Named num [1:31] 2.51 2.9 3.3 3.7 4.11 ...
   ..- attr(*, "names")= chr [1:31] "1" "2" "3" "4" ...
 $ assign      : int [1:3] 0 1 2
 $ qr          :List of 5
  ..$ qr  : num [1:31, 1:3] -5.57 0.18 0.18 0.18 0.18 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:31] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:3] "(Intercept)" "xv" "I(xv^2)"
  .. ..- attr(*, "assign")= int [1:3] 0 1 2
  ..$ qraux: num [1:3] 1.18 1.24 1.13
  ..$ pivot: int [1:3] 1 2 3
  ..$ tol  : num 1e-07
  ..$ rank : int 3
  ..- attr(*, "class")= chr "qr"
 $ df.residual : int 28
 $ xlevels     : Named list()
 $ call        : language lm(formula = yv ~ xv + I(xv^2))
 $ terms       :Classes 'terms', 'formula' length 3 yv ~ xv + I(xv^2)
  .. ..- attr(*, "variables")= language list(yv, xv, I(xv^2))
  . . . .- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
  . . . . . .- attr(*, "dimnames")=List of 2
  . . . . . . . .$ : chr [1:3] "yv" "xv" "I(xv^2)"
  . . . . . . . .$ : chr [1:2] "xv" "I(xv^2)"
  . . . .- attr(*, "term.labels")= chr [1:2] "xv" "I(xv^2)"
  . . . .- attr(*, "order")= int [1:2] 1 1
  . . . .- attr(*, "intercept")= int 1
  . . . .- attr(*, "response")= int 1
  . . . .- attr(*, ".Environment")=<environment: R_GlobalEnv>
  . . . .- attr(*, "predvars")= language list(yv, xv, I(xv^2))
  . . . .- attr(*, "dataClasses")= Named chr [1:3] "numeric" "numeric"
"numeric"
  . . . . . .- attr(*, "names")= chr [1:3] "yv" "xv" "I(xv^2)"
 $ model    :'data.frame': 31 obs. of 3 variables:
  ..$ yv    : num [1:31] 0.802 1.035 7.811 4.138 3.901 ...
  ..$ xv    : int [1:31] 0 1 2 3 4 5 6 7 8 9 ...
  ..$ I(xv^2):Class 'AsIs' num [1:31] 0 1 4 9 16 25 36 49 64 81 ...
  ..- attr(*, "terms")=Classes 'terms', 'formula' length 3 yv ~ xv +
I(xv^2)
  . . . . . .- attr(*, "variables")= language list(yv, xv, I(xv^2))
  . . . . . .- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
  . . . . . . . .- attr(*, "dimnames")=List of 2
  . . . . . . . . . .$ : chr [1:3] "yv" "xv" "I(xv^2)"
  . . . . . . . . . .$ : chr [1:2] "xv" "I(xv^2)"
  . . . . . .- attr(*, "term.labels")= chr [1:2] "xv" "I(xv^2)"
  . . . . . .- attr(*, "order")= int [1:2] 1 1
  . . . . . .- attr(*, "intercept")= int 1
  . . . . . .- attr(*, "response")= int 1
  . . . . . .- attr(*, ".Environment")=<environment: R_GlobalEnv>
  . . . . . .- attr(*, "predvars")= language list(yv, xv, I(xv^2))
  . . . . . .- attr(*, "dataClasses")= Named chr [1:3] "numeric" "numeric"
"numeric"
  . . . . . . . .- attr(*, "names")= chr [1:3] "yv" "xv" "I(xv^2)"
 - attr(*, "class")= chr "lm"
```

This complex structure consists of a list of 12 objects, covering every detail of the model from the variables, their values, the coefficients, the residuals, the effects, and so on.

Reference

Crawley, M.J. (2013) *The R Book*, 2nd edn, John Wiley & Sons, Chichester.

Further Reading

Chambers, J.M. and Hastie, T.J. (1992) *Statistical Models in S*, Wadsworth & Brooks/Cole, Pacific Grove, CA.

R Development Core Team (2014) *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Avaialble from http://www.R-project.org.

Venables, W.N. and Ripley, B.D. (2002) *Modern Applied Statistics with S-PLUS*, 4th edn, Springer-Verlag, New York.

Index

Entries in bold are R functions

Statistics: An Introduction Using R, Second Edition. Michael J. Crawley.