

**Statistical Analysis Using R**  
A Shortcourse at the SETAC Annual Meeting

Stephen B. Cox  
*stephen.cox@ttu.edu*

November 2009

# Chapter 1

## Introduction to R

In this exercise, you will

- familiarize yourself with working in R
- illustrate some very basic attributes of working with data in R.

Obviously, we will only have time to give a tremendously brief introduction to the basics of working in R. It is similar to the first chapter of Dalgaard (2002). For a more technical and broad introduction, see Venables and Ripley (2002). There are also a whole suite of very good introductions and tutorials available at [www.r-project.org](http://www.r-project.org). In particular, the pdf 'An Introduction to R', also available from the help menu, is a great place to spend some time familiarizing yourself with working in R. (Keep in mind that R is itself a very powerful programming language!)

### 1.1 The Command Line

The command line is the main interface to R. Enter a command, you get a response. So, for example, you can use R as a big fancy calculator.

```
> options(width = 85)
> 2 + 2
```

```
[1] 4
```

```
> 2 + 2 * 10
```

```
[1] 22
```

```
> (2 + 2) * 10
```

```
[1] 40
```

```
> log(10)
```

```
[1] 2.302585
```

```
> log10(10)
```

```
[1] 1
```

(Notice that `log()` is by default, the natural log.) Almost all of the commands that you will use in R are in the form of a function. The general way that a function works is:

```
out = function(input1, input2, ...)
```

So, whatever you do, from getting a list of variables you currently have in memory to creating a linear model, it will be done using functions. This also means that you will always need to use the parentheses. Compare the output from `ls` to what you get from `ls()`. To get help for a function, just type `help('function')`, or just `?function`.

Now, let's consider the function for combining elements into a *vector*.

```
> out = c(2, 2)
> help("c")
```

Information in R is primarily stored as one of three basic variable types (what R calls *vectors*). These are:

- numeric vectors
- character vectors
- and logical vectors.

Let's look at some examples.

```
> y = c(1, 2, 3, 4)
> y
```

```
[1] 1 2 3 4
```

```
> attributes(y)
```

```
NULL
```

```
> f = c("control", "control", "treatment", "treatment")
> f
```

```
[1] "control" "control" "treatment" "treatment"
```

```
> l = y > 2
> l
```

```
[1] FALSE FALSE TRUE TRUE
```

We used the `attributes` function to see what attributes R associates with a numeric vector. We find that there basically are no attributes. In fact, these vectors have no special attributes. However, there are other useful objects that do have attributes. Two especially useful ones are:

- factors
- data frames (what R calls a dataset)

A factor is a vector that is intended to be a discrete classification of other vectors (i.e., variables), and it can be created using the function `factor()`. The data frame is the primary way in which we group a series of vectors together that have some conceptual association (i.e., they are part of the same 'dataset').

```
> f = factor(f)
> attributes(f)
```

```
$levels
[1] "control" "treatment"
```

```
$class
[1] "factor"
```

```
> mydata = data.frame(dep = y, indep = f)
> attributes(mydata)
```

```
$names
[1] "dep" "indep"
```

```
$row.names
[1] 1 2 3 4
```

```
$class
[1] "data.frame"
```

In fact, a data.frame is a specific type of R object called a *list*. It is a list of *class* 'data.frame'. To get access to specific variable that resides in a data frame, you need to use the `$`.

```
> mydata$indep
```

```
[1] control control treatment treatment
Levels: control treatment
```

## 1.2 Importing Data

Obviously, rather than entering data at the command line, we would like a better way to get data into R. R has very good libraries for interfacing with a variety of binary data formats. However, let's focus on importing data that are stored as ASCII (i.e., text only) flat files. For the vast majority of studies we normally deal with in the biological sciences, ASCII is the best choice for storing your data. The basic function for reading in text-only files is `read.table()`. Take a look at `?read.table`. My own preference is to store data as comma-separated, text only files (i.e., .csv files). So, I tend to use `read.csv()`.

Let's import a simple example data file from Zar (1999). I have called the file `zar_example10_1.csv`. By default, R will look for files in its 'working directory', which can be found using `getwd()`. To set the working directory, use `setwd()`. Alternatively, you can just use a command from the menu. (You could also just specify the full path when giving the filename.)

```
> zar = read.csv("zar_example10_1.csv")
> summary(zar)
```

feed	weight
Min. :1.000	Min. : 57.00
1st Qu.:1.500	1st Qu.: 65.65
Median :2.000	Median : 74.00
Mean :2.474	Mean : 78.01
3rd Qu.:3.500	3rd Qu.: 89.10
Max. :4.000	Max. :102.60

```
> plot(zar)
```

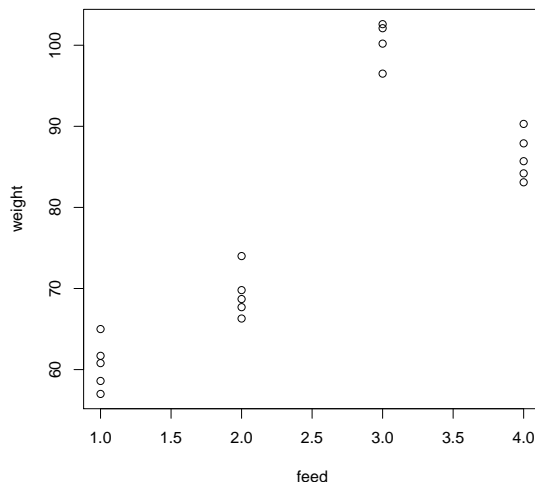


Figure 1.1: The default plot of the dataset from Zar (1999). Because there were only two variables, and they were both imported as numeric vectors, the default plot is a simple scatterplot. The behavior of `plot()` (and most R functions) will depend on what sort of input you give it!

### 1.3 Plotting

The function `plot()` is very powerful - here, we will only cover some basics (take a look at `demo(graphics)` for a demonstration of some plots). Now, take a look at `?plot`. Basically, we provide two variables - the x and y coordinates of a plot. To get this scatterplot, we could have used:

- `plot(zar$feed, zar$weight)`, or
- `plot(zar$weight ~ zar$feed)`

The  $\sim$  can be roughly translated as meaning 'as a function of'. There are other arguments that we can pass on to `plot()`. To do so, simply type "`, argument =`" in the plot function. Some useful ones are:

- `main`
- `xlab`
- `ylab`
- `pch`
- `col`

Finally, also notice that R is interpreting 'feed' to be a continuous variable. Let's convert it to a factor and look at the plot.

```
> zar$feed = factor(zar$feed)
> plot(zar$weight ~ zar$feed, xlab = "Feed Type", ylab = "Final Weight",
+      pch = 16, col = "blue")
```

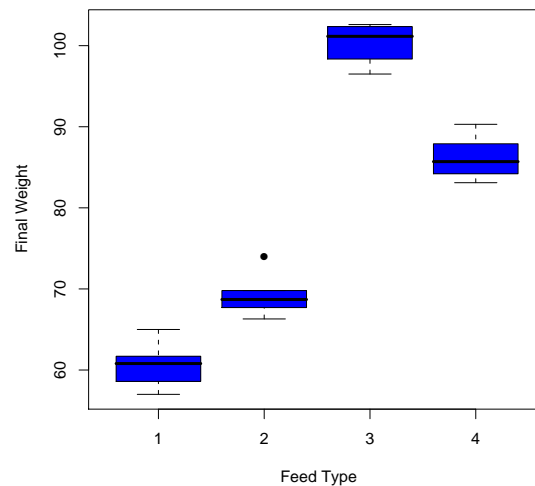


Figure 1.2: Now that our x-variable is a factor, the default plot is a boxplot.

## 1.4 Working With Your Data

### 1.4.1 Extracting

Now that your data is in R, you are going to want to manage it in a variety of ways. First, let's look at various ways to index specific elements of your dataset. The basic way to index a dataframe is by using the square brackets, `[r,c]`. Look at the behaviour of the following commands.

```
> zar[2, ]
> zar[2, ]
> zar[5, 2]
```

Each of these commands extracts a piece of the data.frame and results in a numeric vector. However, this is very different than:

```
> zar[2]
```

By leaving out an explicit row and column index, we have told R to extract the “second element of the dataframe called `zar`.” The elements of a dataframe are the variables; however, the result is still a dataframe.

Logical vectors are also useful for indexing data. For example:

```
> zar$weight > 80
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE
[14] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
> zar$feed == "1"
```

```
[1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[14] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
> zar[zar$weight > 80, ]
```

```
> zar[zar$feed == "1", ]
```

## 1.4.2 Sorting

We can also sort the data based on weight.

```
> o = order(zar$weight)
```

```
> zar[o, ]
```

## 1.4.3 Merging

Another really useful ability is merging. Let’s assume you had a dataset that gave some other useful characteristics of the feed types. The function `merge()` will merge two data.frames based on a common “key” found in both dataframes.

```
> chars = read.csv("feed_characteristics.csv")
```

```
> newdata = merge(zar, chars)
```

```
> newdata
```

	feed	weight	char1	char2
1	1	60.8	meat	low
2	1	57.0	meat	low
3	1	65.0	meat	low
4	1	58.6	meat	low
5	1	61.7	meat	low
6	2	68.7	meat	high
7	2	67.7	meat	high

8	2	74.0	meat	high
9	2	66.3	meat	high
10	2	69.8	meat	high
11	3	102.6	vegetable	low
12	3	102.1	vegetable	low
13	3	100.2	vegetable	low
14	3	96.5	vegetable	low
15	4	87.9	vegetable	high
16	4	84.2	vegetable	high
17	4	83.1	vegetable	high
18	4	85.7	vegetable	high
19	4	90.3	vegetable	high

## Chapter 2

# Hypothesis Tests

In this exercise, you will

- continue to familiarize yourself with working in R,
- test some of the assumptions of hypothesis tests using real data,
- examine some functions in R for conducting one- and two-sample hypothesis tests.

### 2.1 The Data

Lets take a look at some data on soil physico-chemical characteristics. These data are soil nutrient concentrations at 40 sampling sites within the Luquillo Experimental Forest of Puerto Rico.

```
> d = read.csv("luq_soil_1999.csv")
> attributes(d)

$names
[1] "Sample"   "Ca.mg.g." "Na.mg.g." "Mg.mg.g." "P.mg.g."  "Fe.mg.g." "Mn.mg.g."
[8] "K.mg.g."  "group"

$class
[1] "data.frame"

$row.names
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
[28] 28 29 30 31 32 33 34 35 36 37 38 39 40

> str(d)

'data.frame':      40 obs. of  9 variables:
 $ Sample  : int  37 40 43 46 49 88 91 94 97 100 ...
 $ Ca.mg.g.: num  2.79 1.02 0.21 2.31 1.04 ...
 $ Na.mg.g.: num  1.818 0.21 0.083 0.152 0.183 ...
 $ Mg.mg.g.: num  1.1 0.392 0.18 0.401 0.48 0.755 0.973 0.271 0.591 0.292 ...
 $ P.mg.g. : num  0.012 0.012 0.005 0.009 0.014 0.009 0.009 0.008 0.01 0.011 ...
```

```

$ Fe.mg.g.: num  0.352 0.71 0.6 1.158 2.478 ...
$ Mn.mg.g.: num  0.36 0.101 0.064 0.123 0.027 0.131 0.094 0.038 0.063 0.047 ...
$ K.mg.g.  : num  0.319 0.206 0.079 0.168 0.191 0.191 0.156 0.112 0.169 0.175 ...
$ group    : int   1 1 1 1 1 1 1 1 1 1 ...

```

Note the use of the function `str()`. This function gives a summary of the structure of whatever object you pass to it. Now, let's take a look at the data.

```
> plot(d[2:8])
```

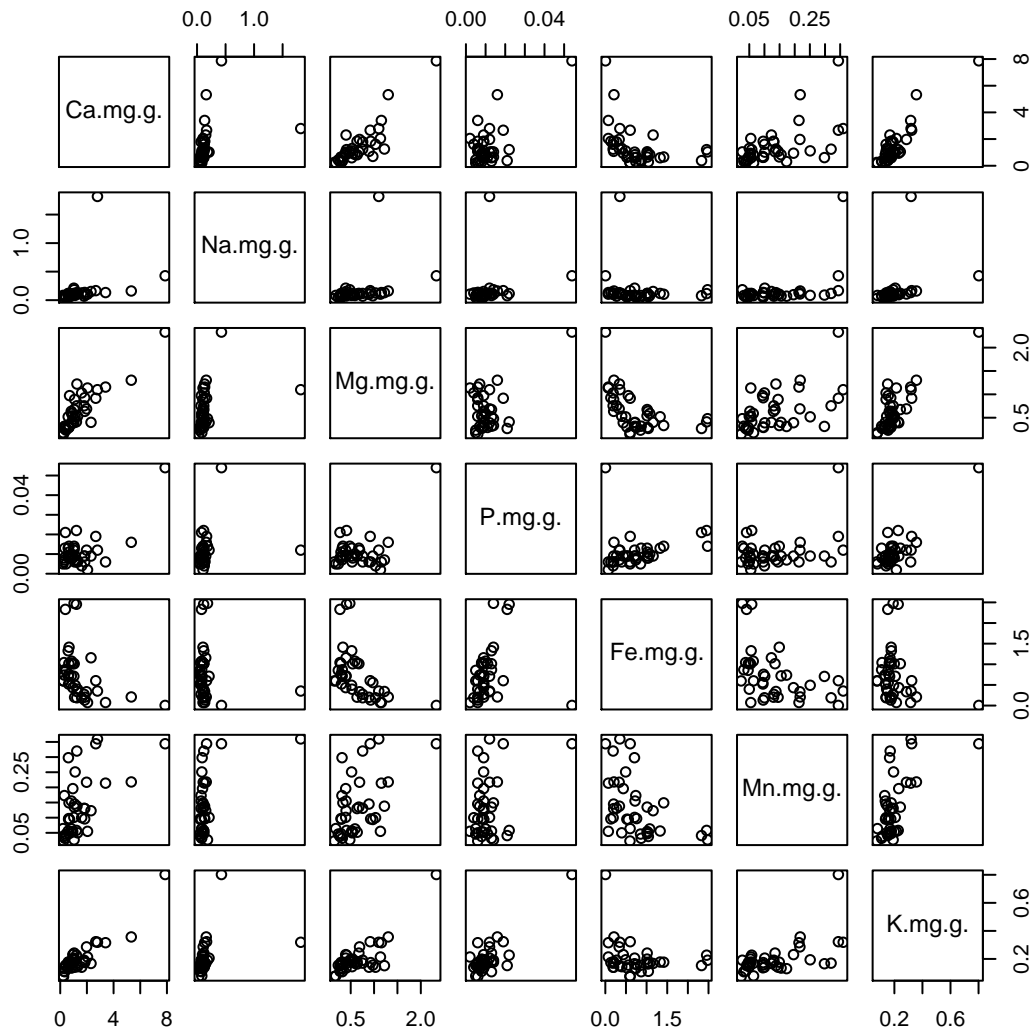


Figure 2.1: The default plot of the soil concentration data. Because I plotted a dataset with more than two variables, the default behavior was to plot a matrix of all possible scatterplots. See if you notice any striking patterns.

Let's take a look at the data on Na concentrations in a little more detail.

```
> boxplot(d$Na.mg.g)
> hist(d$Na.mg.g)
```

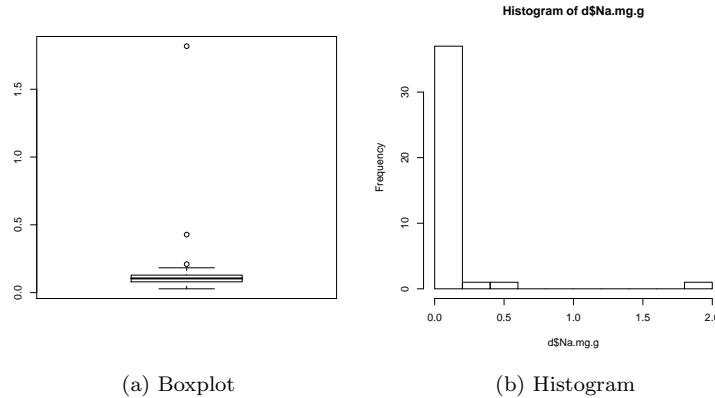


Figure 2.2: The Na concentration data

Both plots are giving a strong indication that there is at least one outlier in the data. What do we do with this outlier? Well, this is not so much a statistical question, as it is a question about your confidence in the data. How confident are you that the instrumentation was working properly? How confident are you that there was no contamination introduced into the original sample? For now, let's leave it in.

## 2.2 Testing Normality

We will discuss two primary ways to test the normality assumption.

- quantile-quantile plots
- hypothesis tests (e.g., Shapiro-Wilk Test).

### 2.2.1 Q-Q plots

One of the easiest, and most robust ways to test for deviations from normality is to plot quantiles one would expect, given a normal pdf, against the quantiles observed in the data. This is the idea behind the quantile-quantile plot. If the data are approximately normal, then the points on such a plot should follow a straight line. There are two functions that are useful here. `qqnorm()` plots the normal quantile-quantile (i.e., Q-Q) plot. `qqline()` adds the expected linear relationship to the Q-Q plot.

```
> qqnorm(d$Na.mg.g)
> qqline(d$Na.mg.g)
```

Looking at the result (Fig. 2.3), you can see that there are deviations from what we would expect. However, it really appears that the deviations are due mainly to some really large values. Let's remove some of these large values and see what happens.

```
> qqnorm(sort(d$Na.mg.g)[1:39])
> qqline(sort(d$Na.mg.g)[1:39])
```

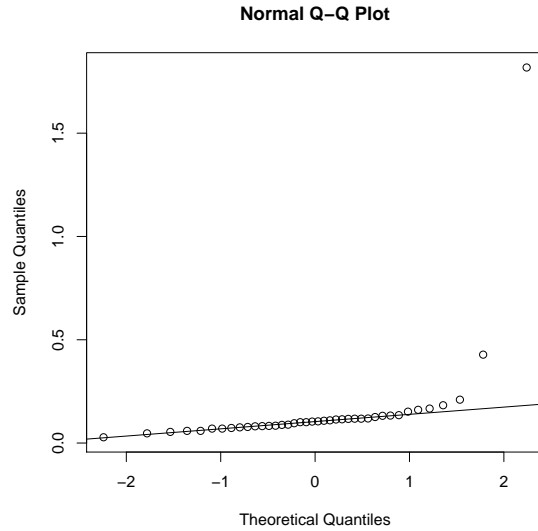
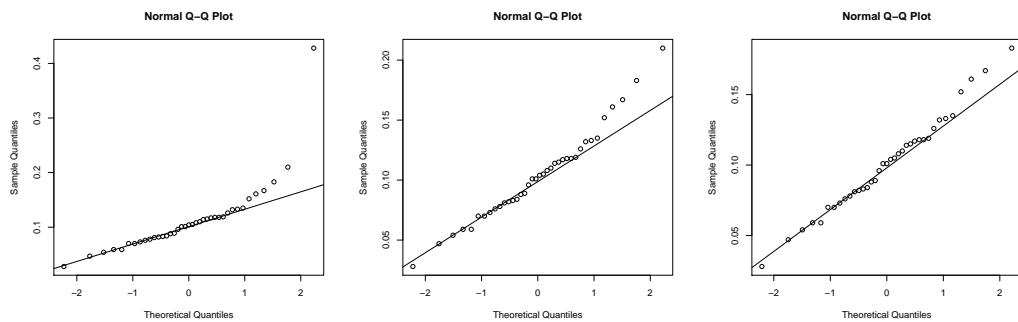


Figure 2.3: Normal Q-Q plot of the Na concentration data. Note the way that the outliers stand out, indicating we have some values that are larger than expected.

```
> qqnorm(sort(d$Na.mg.g)[1:38])
> qqline(sort(d$Na.mg.g)[1:38])

> qqnorm(sort(d$Na.mg.g)[1:37])
> qqline(sort(d$Na.mg.g)[1:37])
```



(a) Dropping the largest value (b) Dropping the largest two values (c) Dropping the largest three values

Figure 2.4: Q-Q plots of the Na concentration data

Note how we used `sort()` as a quick and dirty way of identifying the largest values. (In section 2.4.1 we will see a more objective way of identifying outliers.) Deleting the large values tends to increase the fit between our observed data and what we would expect assuming a normal pdf. How many should be deleted? Should any?

## 2.2.2 Normality hypothesis tests

There are a variety of hypothesis tests available to test for deviations from what would be expected given a normally distributed population. All test:

$$\begin{aligned}H_o: & \text{ data are normally distributed} \\H_a: & \text{ not so}\end{aligned}$$

The Shapiro-Wilk test is one such hypothesis test. Like any other hypothesis test, the test statistic (called  $W$ ) is compared to its sampling distribution - assuming that  $H_o$  is true.

```
> shapiro.test(d$Na.mg.g)

      Shapiro-Wilk normality test

data:  d$Na.mg.g
W = 0.293, p-value = 1.205e-12

> shapiro.test(sort(d$Na.mg.g)[1:39])

      Shapiro-Wilk normality test

data:  sort(d$Na.mg.g)[1:39]
W = 0.7059, p-value = 1.580e-07

> shapiro.test(sort(d$Na.mg.g)[1:38])

      Shapiro-Wilk normality test

data:  sort(d$Na.mg.g)[1:38]
W = 0.9728, p-value = 0.4716

> shapiro.test(sort(d$Na.mg.g)[1:37])

      Shapiro-Wilk normality test

data:  sort(d$Na.mg.g)[1:37]
W = 0.9874, p-value = 0.9431
```

So, the data are non-normal. Once the outliers are removed, however, we have no indication of deviations from normality. Although the Shapiro-Wilk test has been shown to have excellent power, one weakness is that it is sensitive to tied observations. In particular, tied observations tend to reduce the power of the test.

## 2.3 One Sample Tests

The function `t.test()` handles a variety of t-tests.

- One sample

- Two samples
  - Classical t-test
  - Welch's approximation for heteroscedastic data
  - Paired t-test

Take a look at `?t.test`. For the Na data, let's test  $H_o : \mu = 0.1$ .

```
> t.test(d$Na.mg.g, mu = 0.1)
```

One Sample t-test

```
data: d$Na.mg.g
t = 1.251, df = 39, p-value = 0.2184
alternative hypothesis: true mean is not equal to 0.1
95 percent confidence interval:
 0.06619675 0.24340325
sample estimates:
mean of x
 0.1548
```

In fact, given the variability in our observed data, what is the probability that we would have rejected  $H_o : \mu = 0.1$ , assuming that  $\mu = 0.3$ ? Or, more simply stated, given the level of variability in our data, and given our pre-determined  $\alpha$ , what was our power to detect an effect size of 0.2?

```
> power.t.test(n = 40, delta = 0.2, sd = sd(d$Na.mg.g.), sig.level = 0.05,
+   power = NULL, type = "one.sample", alternative = "two.sided")
```

One-sample t test power calculation

```
      n = 40
  delta = 0.2
    sd = 0.2770449
sig.level = 0.05
  power = 0.9936264
alternative = two.sided
```

So, if an effect size of 0.2 was biologically meaningful, these results indicate that we had sufficient power to detect meaningful effects. In addition, it might be tempting to conduct another sort of power analysis.

```
> delta = mean(d$Na.mg.g) - 0.1
> delta
```

```
[1] 0.0548
```

```
> power.t.test(n = 40, delta = delta, sd = sd(d$Na.mg.g.), sig.level = 0.05,
+   power = NULL, type = "one.sample", alternative = "two.sided")
```

One-sample t test power calculation

```
      n = 40
```

```

delta = 0.0548
sd = 0.2770449
sig.level = 0.05
power = 0.2297612
alternative = two.sided

```

Here, we are trying to ask, 'Given our data, what was our power to detect the *observed* effect size?' Although both of these power analyses were conducted after we had actually done the hypothesis test of interest (i.e., they are both *post-hoc* power analyses), they are very different! In the first case, we chose an effect size based on biological significance. In the second case, we let our data determine the effect size. **In general, the second case is NOT recommended (and some would argue it is absolutely meaningless).** See p.168 of Quinn and Keough for a good discussion of this.

## 2.4 Two Sample Tests

### 2.4.1 Independent samples

The soil samples we have been discussing are from two different groups, denoted by the `group` variable. Let's test  $H_o : \mu_1 = \mu_2$  vs.  $H_a : \mu_1 \neq \mu_2$ . To do so, we may want to first test for normality. You might be tempted to look at Fig. 2.3; however, remember that the assumption of the Two-sample t-test is that the data are normal, *within* each group. To make things easier, let's create two new variables that represent the two groups.

```

> Na1 = d$Na.mg.g[d$group == 1]
> Na2 = d$Na.mg.g[d$group == 2]
> qqnorm(Na1)
> qqline(Na1)

> qqnorm(Na2)
> qqline(Na2)

```

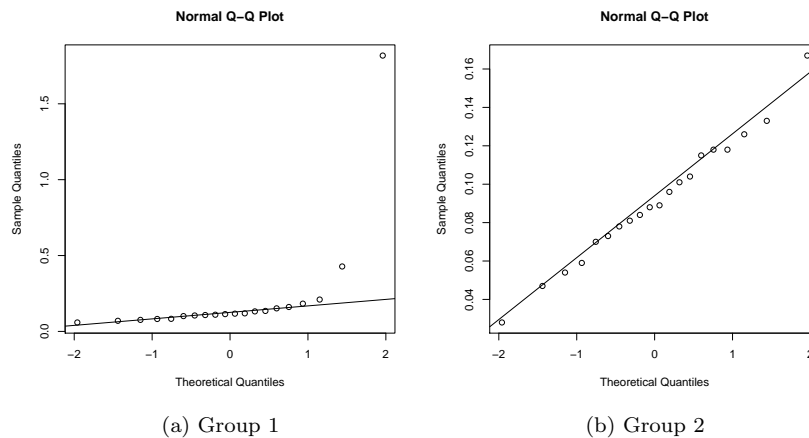


Figure 2.5: Q-Q plots of the Na concentration data in each group

Group 1 seems to have deviations from normality. Again, it appears to be a couple of extremely large values that are a problem. Should these observations be thrown out? What do you think?

We also might want to test for differences in variance. To do so, we will use `var.test()`.

```
> var.test(Na1, Na2)
```

```
F test to compare two variances
```

```
data: Na1 and Na2
F = 137.3039, num df = 19, denom df = 19, p-value = 4.441e-16
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 54.34655 346.89152
sample estimates:
ratio of variances
 137.3039
```

Here, we gave the function `var.test()` two arguments - `Na1` and `Na2`. Another way to specify this test is to use the 'model formula' notation. That is, let's use the argument `variable ~ group` to say, 'we want to look at our variable as a function of the grouping variable.' This notation will be really useful in the future. As a result, we will tend to use it.

```
> var.test(d$Na.mg.g ~ d$group)
```

```
F test to compare two variances
```

```
data: d$Na.mg.g by d$group
F = 137.3039, num df = 19, denom df = 19, p-value = 4.441e-16
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 54.34655 346.89152
sample estimates:
ratio of variances
 137.3039
```

Remember, this test of variances is very sensitive to deviations from normality. Hence, we won't put much stock in it at this point. Finally, let's test  $H_o : \mu_1 = \mu_2$  vs.  $H_a : \mu_1 \neq \mu_2$ . Take a look at `?t.test` again.

```
> t.test(d$Na.mg.g ~ d$group)
```

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

```
data: d$Na.mg.g by d$group
t = 1.4674, df = 19.277, p-value = 0.1584
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-0.05384025  0.30724025
sample estimates:
mean in group 1 mean in group 2
 0.21815          0.09145
```

Examine this output carefully. (*You should be able to interpret everything you see!*) Well, no significant differences. Why could this be? Let's take a look at the variances in the two groups. To do so, I am going to use a really handy function called `tapply()`.

```
> tapply(d$Na.mg.g, d$group, var)
```

```
      1      2
0.14802045 0.00107805
```

So - the variance in group 1 is much, much larger than the variance in group 2. This, in turn, increases uncertainty in our estimate of  $\mu_1$ , and makes it difficult to distinguish from  $\mu_2$ . But, remember, this is really due to a couple of outliers. Let's remove them and see what happens. I am going to use `boxplot()` to help me find outliers.

```
> bp = boxplot(d$Na.mg.g)
> str(bp)
```

```
List of 6
```

```
$ stats: num [1:5, 1] 0.028 0.0795 0.1045 0.129 0.183
$ n      : num 40
$ conf  : num [1:2, 1] 0.0921 0.1169
$ out   : num [1:3] 1.818 0.21 0.428
$ group: num [1:3] 1 1 1
$ names: chr "1"
```

```
> address = d$Na.mg.g %in% bp$out
> address
```

```
[1] TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[14] FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[27] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[40] FALSE
```

```
> d$Na.mg.g[!address]
```

```
[1] 0.083 0.152 0.183 0.070 0.076 0.059 0.082 0.117 0.108 0.119 0.135 0.114 0.110
[14] 0.161 0.132 0.105 0.101 0.118 0.167 0.126 0.101 0.115 0.084 0.133 0.118 0.054
[27] 0.096 0.073 0.059 0.028 0.078 0.081 0.088 0.047 0.104 0.089 0.070
```

```
> t.test(d$Na.mg.g[!address] ~ d$group[!address])
```

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

```
data: d$Na.mg.g[!address] by d$group[!address]
t = 1.8958, df = 33.849, p-value = 0.06655
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.001494916  0.042947857
sample estimates:
mean in group 1 mean in group 2
      0.1121765      0.0914500
```

So, removing the outliers changed the overall result of the test. Fortunately, non-parametric tests are not as sensitive to outliers. Why? Because the process of ranking the observations minimizes the effect of outliers.

```
> wilcox.test(d$Na.mg.g ~ d$group)

Wilcoxon rank sum test with continuity correction

data: d$Na.mg.g by d$group
W = 285.5, p-value = 0.02147
alternative hypothesis: true location shift is not equal to 0
```

Note the way the statement of the hypothesis has changed. Here,

$$H_0: \text{ the true location shift } = 0$$

$$H_a: \text{ the true location shift } \neq 0$$

Finally, the Wilcoxon rank sum test is conceptually the same thing as conducting a t-test on the ranked data. Let's use `rank()` to rank the data and then conduct the t-test.

```
> t.test(rank(d$Na.mg.g) ~ d$group)

Welch Two Sample t-test

data: rank(d$Na.mg.g) by d$group
t = 2.4582, df = 37.867, p-value = 0.01865
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 1.508106 15.591894
sample estimates:
mean in group 1 mean in group 2
      24.775      16.225
```

We get very similar results to the Wilcoxon test. In fact, some texts argue that the results should be identical. See `?wilcox.test` for some discussion of the fact that the exact details of what constitutes the Wilcoxon rank sum test is still debated.

## 2.4.2 Paired t-test

As a final example, let's consider a dataset on deer measurements.

```
> d2 = read.csv("zar_example9_1.csv")
> d2
```

	Deer	hindleg	foreleg
1	1	142	138
2	2	140	136
3	3	144	147
4	4	144	139
5	5	142	143
6	6	146	141
7	7	149	143
8	8	150	145
9	9	142	136
10	10	148	146

Note the structure of the data. Lengths of hindlegs and forelegs were measured on ten individual deer. Thus, the hindleg observations are not independent of the foreleg measurements. Let's see what will happen if we ignore this lack of independence.

```
> t.test(d2$hindleg, d2$foreleg)
```

Welch Two Sample t-test

```
data: d2$hindleg and d2$foreleg
t = 1.978, df = 17.501, p-value = 0.06389
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.2122142  6.8122142
sample estimates:
mean of x mean of y
  144.7    141.4
```

```
> boxplot(d2$hindleg, d2$foreleg)
```

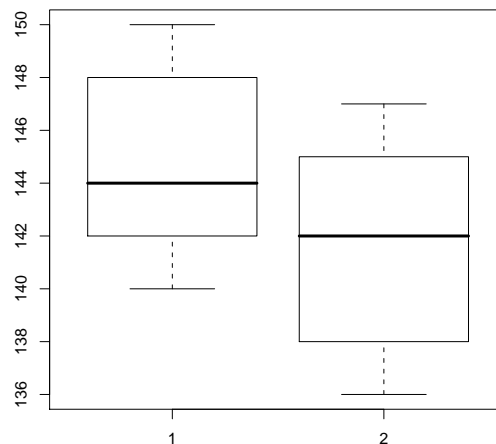


Figure 2.6: Boxplot of the data on deer measurements. Note the degree of overlap in the distributions of the two variables. The variability within each variable makes it difficult to detect a difference.

Now, let's conduct the appropriate test.

```
> t.test(d2$hindleg, d2$foreleg, paired = T)
```

Paired t-test

```
data: d2$hindleg and d2$foreleg
t = 3.4138, df = 9, p-value = 0.007703
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  1.113248  5.486752
```

```

sample estimates:
mean of the differences
          3.3

```

We are now able to detect a difference. In fact, for most individuals in the data, hindlegs are longer than forelegs. But, the fact that there is variability among deers in terms of overall size would mask these differences if we did not account for it. `interaction.plot()` is a useful function for visualizing this effect.

```

> g = gl(2, 10, labels = c("h", "f"))
> g

[1] h h h h h h h h h h f f f f f f f f f f
Levels: h f

> deer2 = c(d2$Deer, d2$Deer)
> deer2

[1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10

> interaction.plot(g, deer2, c(d2$hindleg, d2$foreleg), ylab = "length")

```

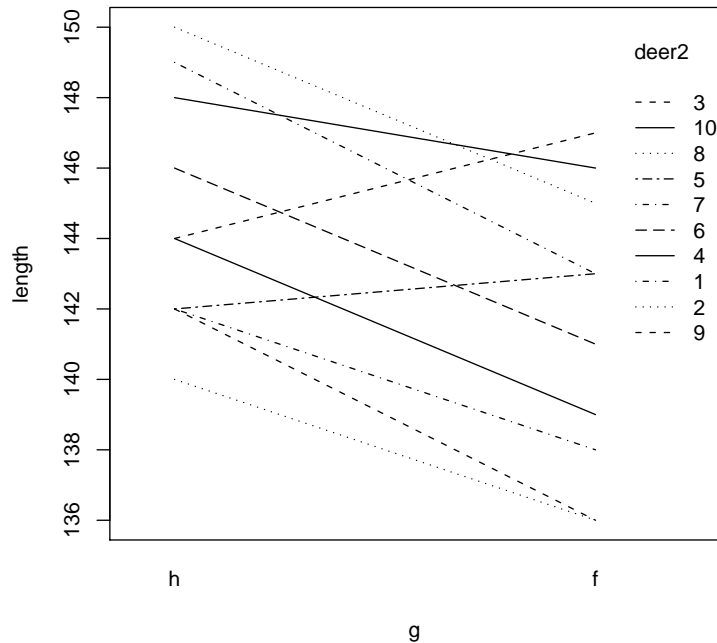


Figure 2.7: An interaction plot of the data on deer measurements. Each individual is denoted by a different line on the plot.

# Chapter 3

## Analysis of Variance

In this exercise, you will

- continue to familiarize yourself with working in R,
- examine some functions in R for conducting a Model I (Fixed Effects) Analysis of Variance (ANOVA) with one factor,
- and examine some functions in R for conducting a Model II (Random Effects) ANOVA with one factor.

### 3.1 The Data

As an example, let's use the data from Quinn and Keough (2002) found in Box 8.1. The variable of interest, diatom diversity, has been measured in a variety of streams. Zn concentrations in the water also have been measured.

```
> d = read.csv("medley.csv")
> attributes(d)

$names
[1] "STREAM" "ZINC" "DIVERSTY" "ZNGROUP" "RESID1" "PREDICT1" "RESID2"
[8] "PREDICT2"

$class
[1] "data.frame"

$row.names
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
[28] 28 29 30 31 32 33 34

> str(d)

'data.frame': 34 obs. of 8 variables:
 $ STREAM : Factor w/ 6 levels "Arkan","Blue",...: 4 4 4 4 2 2 2 2 2 2 ...
 $ ZINC : Factor w/ 4 levels "BACK","HIGH",...: 1 2 2 4 1 2 1 1 2 4 ...
```

```

$ DIVERSTY: num 2.27 1.25 1.15 1.62 1.7 0.63 2.05 1.98 1.04 2.19 ...
$ ZNGROUP : int 1 4 4 3 1 4 1 1 4 3 ...
$ RESID1 : num 0.4725 -0.0278 -0.1278 -0.0978 -0.0975 ...
$ PREDICT1: num 1.8 1.28 1.28 1.72 1.8 ...
$ RESID2 : num 0.6975 -0.3225 -0.4225 0.0475 0.03 ...
$ PREDICT2: num 1.57 1.57 1.57 1.57 1.67 ...

```

There are two separate variables that code for the Zn group. ZINC is a factor with 4 levels.

```
> levels(d$ZINC)
```

```
[1] "BACK" "HIGH" "LOW" "MED"
```

R interprets factors in alphabetical order, and, in most cases for ANOVA, this is an unimportant detail. However, there may be cases where we want to retain a certain order among the factors. For example, here we would like to maintain the relationship BACK < LOW < MED < HIGH. *Later on, we will see that such ordered factors can be analyzed in a specific way that gives more insight into the data.* For now, we just want to retain this order for the sake of looking at plots, etc. ZNGROUP codes for the Zn group using the integers 1-4 to help retain this ordering. Let's convert it to a factor, look at some useful summary information, and then plot the data.

```
> d$ZNGROUP = factor(d$ZNGROUP)
> attributes(d$ZNGROUP)
```

```
$levels
[1] "1" "2" "3" "4"
```

```
$class
[1] "factor"
```

```
> tapply(d$DIVERSTY, factor(d$ZNGROUP), summary)
```

```
$`1`
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.760  1.658   1.935   1.797  2.088   2.270
```

```
$`2`
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1.400  1.788   1.990   2.033  2.230   2.830
```

```
$`3`
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.800  1.620   1.940   1.718  2.060   2.190
```

```
$`4`
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.630  1.040   1.250   1.278  1.430   1.900
```

```
> tapply(d$DIVERSTY, factor(d$ZNGROUP), var)
```

```

      1      2      3      4
0.2354786 0.1980214 0.2530194 0.1822194

```

```
> boxplot(d$DIVERSTY ~ (d$ZNGROUP), xlab = "Zn Group", ylab = "Diatom Diversity",
+         names = c("B", "L", "M", "H"))
```

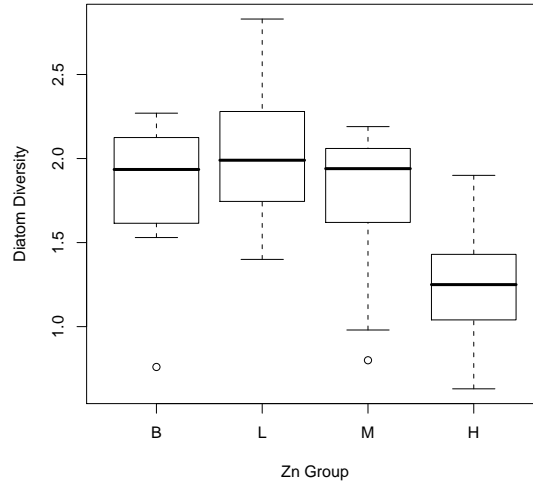


Figure 3.1: Boxplot of the Diatom data. There does appear to be outliers in the B and M groups.

In looking at the variances, we know that they will never be exactly equal. However, it is worth taking a look to see if there are any groups that appear to have a variance that is orders of magnitude different than the others. Notice how all of this summary information (and more) is nicely illustrated in the boxplot.

## 3.2 ANOVA

### 3.2.1 Syntax for ANOVA in in R

Conducting an Analysis of Variance in R consists of two steps. First, the function `lm` is used to to construct a linear model of the relationship between  $y$ , the dependent variable, and  $x$ , the categorical independent variable (must be defined as a factor within R using `factor()`). The second step is to construct the ANOVA table from this model using the function `anova()`. For example,

```
> out = lm(y~x, data = name)
> anova(out)
```

where `out` is the name of a variable in which to save the output from `lm()`, and `name` is the name of the dataframe containing  $x$  and  $y$ . This conducts the ANOVA, testing for differences in the mean values of  $y$  within the groups defined by  $x$ .

Alternatively, the function `aov()` can be used to do both steps simultaneously. For example,

```
> out = aov(y~x, data = name)
```

would perform the same ANOVA; however the output is saved a little differently. This comes in handy for certain other functions. `summary(out)` would print a nice table of the results.

**Key Point to Remember:** The approach to conducting an ANOVA in R is based on the fact that an ANOVA is a *linear model*. This may not make a lot of sense right now, but we will discuss this in detail later on.

### 3.2.2 The Diatom Analysis

First, we may want to check for heteroscedasticity. Bartlett's test is the multiple group extension of the two-sample F-test. It is conducted using `bartlett.test()`, and tests:

$$\begin{aligned} H_o: & \sigma_1 = \sigma_2 = \dots = \sigma_k \\ H_a: & \text{at least one inequality} \end{aligned}$$

```
> bartlett.test(d$DIVERSITY ~ d$ZNGROUP)
```

```
    Bartlett test of homogeneity of variances
```

```
data:  d$DIVERSITY by d$ZNGROUP
Bartlett's K-squared = 0.2529, df = 3, p-value = 0.9686
```

Like the F-ratio test for two variances, Bartlett's test is sensitive to an assumption that the data within each group are normally distributed. Another good way to examine how well our data adhere to the assumptions of the ANOVA model is to look at the residuals from that model. Remember, the ANOVA model is:

$$Y_{ij} = \mu + \alpha_i + \epsilon_{ij} \tag{3.1}$$

The assumptions about normality and equal variances really apply to those  $\epsilon_{ij}$ , which can be extracted from the model using `resid()`. In particular, a plot of the residuals vs. the predicted values (i.e., the means within each group) is a useful tool for examining assumptions. (`predict()` will extract the group means). Let's fit the model, and look at this plot.

```
> m = lm(DIVERSITY ~ ZNGROUP, data = d)
> plot(resid(m) ~ predict(m), ylab = "Residuals", xlab = "Group mean diversity")
```

In fact, if we give a linear model fit to `plot()`, it will give us a series of four diagnostic plots. The first of these is this exact plot. Take a look at `plot(m, which = 1)`. Here, residuals look pretty good. Let's proceed with the ANOVA.

```
> anova(m)
```

Analysis of Variance Table

```
Response: DIVERSITY
      Df Sum Sq Mean Sq F value Pr(>F)
ZNGROUP   3  2.5666   0.8555   3.9387 0.01756 *
Residuals 30  6.5164   0.2172
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Because  $p < 0.05$ , we would conclude that there is at least one difference among the group means. Note that `aov()` is basically a wrapper function that fits the linear model and then extracts the ANOVA table. It will give the same result:

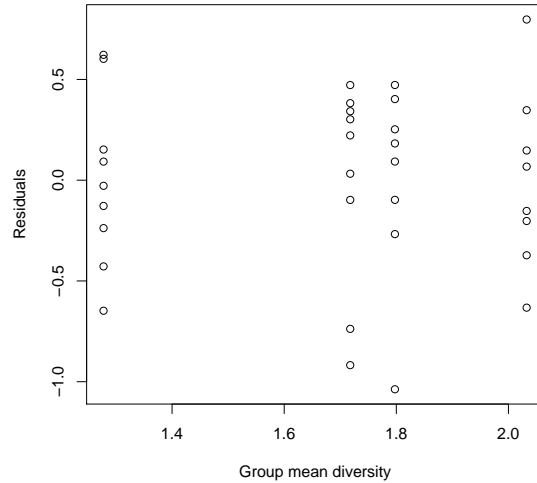


Figure 3.2: Residual plot of the Diatom data. There does not appear to be any striking outliers, and the variability within each group is relatively constant.

```
> out = aov(DIVERSITY ~ ZNGROUP, data = d)
> summary(out)
```

```
          Df Sum Sq Mean Sq F value Pr(>F)
ZNGROUP    3  2.5666  0.8555   3.9387 0.01756 *
Residuals  30  6.5164  0.2172
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 3.3 Post-hoc tests

#### 3.3.1 Adjusted t-tests

This is great - but where do the differences occur? First, let's look at all pairwise t-tests, using Bonferroni's adjustment. Frequently, this approach is described as a way to adjust your CWER (i.e., your  $\alpha$ ) for each pairwise comparison to ensure a constant EWER. Instead of adjusting  $\alpha$ , the following R functions adjust the p-values themselves. Therefore, to make our decisions for each comparison, we can just compare the reported p-values to our desired EWER.

```
> pairwise.t.test(d$DIVERSITY, d$ZNGROUP, p.adj = "bonf")
```

Pairwise comparisons using t tests with pooled SD

data: d\$DIVERSITY and d\$ZNGROUP

```
  1     2     3
2 1.000 -     -
3 1.000 1.000 -
```

```
4 0.173 0.014 0.326
```

```
P value adjustment method: bonferroni
```

Bonferroni tends to be too conservative, so let's look at Holm's adjustment.

```
> pairwise.t.test(d$DIVERSTY, d$ZNGROUP, p.adj = "holm")
```

```
Pairwise comparisons using t tests with pooled SD
```

```
data: d$DIVERSTY and d$ZNGROUP
```

```
  1      2      3
2 0.643 -      -
3 0.727 0.525 -
4 0.145 0.014 0.217
```

```
P value adjustment method: holm
```

### 3.3.2 Tukey's test

Tukey's test is good choice for post-hoc comparisons. Specifically, Tukey's Honestly Significant Difference (HSD) test is implemented using `TukeyHSD()`. The input should be the output from `aov()`. In our example, this is our variable `out`.

```
> TukeyHSD(out, ordered = T)
```

```
Tukey multiple comparisons of means
 95% family-wise confidence level
factor levels have been ordered
```

```
Fit: aov(formula = DIVERSTY ~ ZNGROUP, data = d)
```

```
$ZNGROUP
      diff      lwr      upr    p adj
3-4 0.4400000 -0.15739837 1.0373984 0.2095597
1-4 0.51972222 -0.09606192 1.1355064 0.1218677
2-4 0.75472222  0.13893808 1.3705064 0.0116543
1-3 0.07972222 -0.53606192 0.6955064 0.9847376
2-3 0.31472222 -0.30106192 0.9305064 0.5153456
2-1 0.23500000 -0.39863665 0.8686367 0.7457444
```

The additional argument `ordered = T` just ensures the the differences are positive. (Compare this result to `TukeyHSD(out)`).

In all cases, we see that the data indicate that the difference really lies between groups 2 and 4. To adequately interpret and summarize this information, we have to follow the rules for interpreting post-hoc tests. We found:

$$\mu_4 = \mu_3 = \mu_1 \text{ and } \mu_3 = \mu_1 = \mu_2.$$

We could also summarize this result on the boxplot using letters. Here, I will have to expand the y-axis limits using the `ylim` argument to give room for the letters. I will also use `tapply(d$DIVERSITY, d$ZNGROUP, max)` to extract the maximum diversity values in each group. These (plus a little extra space) become the y-coordinates for the new letters.

```
> bp = boxplot(d$DIVERSITY ~ (d$ZNGROUP), ylim = c(0, 3), xlab = "Zn Group",
+   ylab = "Diatom Diversity", names = c("B", "L", "M", "H"))
> text(1:4, tapply(d$DIVERSITY, d$ZNGROUP, max) + 0.15, c("ab", "a",
+   "ab", "b"))
```

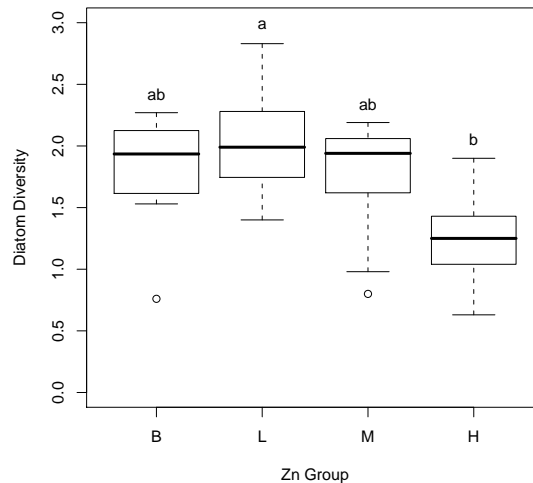


Figure 3.3: Boxplot of the diatom diversity data. Here, letters represent the results of Tukey's post-hoc comparisons of group means.

### 3.3.3 Dunnett's test

Finally, instead of comparing all possible combinations, you might instead want to just compare each group to a reference. In this case, you might want to use the "Background" group as a reference. Dunnett's test will do this, and can be implemented in the `multcomp` library. The main function in `multcomp` for doing post hoc tests is `glht` (which stands for 'general linear hypothesis tests'). At this point, the syntax may seem a little strange, but the `multcomp` library is extremely powerful. We will only scratch the surface by using it to conduct the Dunnett's test. (Also, be aware that when you install `multcomp`, there are a large number of other libraries it depends on.)

```
> library(multcomp)
> test.out = glht(out, linfct = mcp(ZNGROUP = "Dunnett"))
> summary(test.out)
```

Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts

```
Fit: aov(formula = DIVERSTY ~ ZNGROUP, data = d)
```

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t )
2 - 1 == 0	0.23500	0.23303	1.008	0.6195
3 - 1 == 0	-0.07972	0.22647	-0.352	0.9702
4 - 1 == 0	-0.51972	0.22647	-2.295	0.0728 .

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
(Adjusted p values reported -- single-step method)

Notice that the first group (whatever that happens to be) is considered the reference group. Be careful to make sure this is what you want!

**Key Point to Remember:** The steps to analyzing the data using ANOVA were:

- plot the response as a function of the grouping variable,
- fit the linear model using `lm()`,
- consider the assumptions (e.g., by examining the residuals of the linear model),
- interpret the ANOVA results, which were obtained using `anova()`,
- consider post-hoc comparisons.

### 3.4 Non-parametric alternatives

If we had wished to use a non-parametric test, we have a couple of different options. The Kruskal-Wallis test is the non-parametric extension to the Wilcoxon (Mann-Whitney) test we used for two groups. It is conducted as follows.

```
> kruskal.test(DIVERSTY ~ ZNGROUP, data = d)
```

```
Kruskal-Wallis rank sum test
```

```
data: DIVERSTY by ZNGROUP
```

```
Kruskal-Wallis chi-squared = 8.7367, df = 3, p-value = 0.03300
```

Alternatively, we could also just do an ANOVA on the rank-transformed data. For one-way ANOVA, this is basically the same thing as the Kruskal-Wallis.

```
> summary(aov(rank(DIVERSTY) ~ ZNGROUP, data = d))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
ZNGROUP	3	866.12	288.71	3.6008	0.02471 *
Residuals	30	2405.37	80.18		

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Finally - to conduct post-hoc tests for the Kruskal-Wallis, we can just rely on adjusted p-values from multiple Wilcoxon two-sample tests.

```
> pairwise.wilcox.test(d$DIVERSITY, d$ZNGROUP, p.adj = "holm")
```

Pairwise comparisons using Wilcoxon rank sum test

data: d\$DIVERSITY and d\$ZNGROUP

```
 1      2      3
2 1.000 -      -
3 1.000 1.000 -
4 0.137 0.056 0.250
```

P value adjustment method: holm

Here, we have lost some power to detect differences!

### 3.5 Trend Analysis

Up to now, we have not considered the fact that our treatment groups actually fall in a specified order. That is, we may want to consider the fact that our groups follow the order: *BACK < LOW < MED < HIGH*. If this is true, in addition to asking about mean differences, we may want to know if there is any evidence that there is a linear, or perhaps quadratic, relationship between diatom diversity and Zn concentrations. Because we are using the framework of a linear model for our analysis, we can address this question! To do so, all we need to do is specify that our factor is *ordered*.

```
> d$ZNGROUP2 = factor(d$ZNGROUP, ordered = T)
> attributes(d$ZNGROUP2)
```

```
$levels
[1] "1" "2" "3" "4"
```

```
$class
[1] "ordered" "factor"
```

```
> m = lm(DIVERSITY ~ ZNGROUP2, data = d)
> anova(m)
```

Analysis of Variance Table

```
Response: DIVERSITY
      Df Sum Sq Mean Sq F value Pr(>F)
ZNGROUP2  3  2.5666  0.8555  3.9387 0.01756 *
Residuals 30  6.5164  0.2172
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice - the overall ANOVA is exactly the same as before. Let's look at a summary of the model.

```
> summary(m)
```

```

Call:
lm(formula = DIVERSTY ~ ZNGROUP2, data = d)

Residuals:
    Min       1Q   Median       3Q      Max
-1.03750 -0.22896  0.07986  0.33222  0.79750

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.70639    0.08007  21.312 <2e-16 ***
ZNGROUP2.L  -0.41901    0.16014  -2.617  0.0138 *
ZNGROUP2.Q  -0.33750    0.16014  -2.108  0.0435 *
ZNGROUP2.C   0.09491    0.16014   0.593  0.5578
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4661 on 30 degrees of freedom
Multiple R-squared:  0.2826,    Adjusted R-squared:  0.2108
F-statistic: 3.939 on 3 and 30 DF,  p-value: 0.01756

```

Now, these represent hypotheses about a linear, quadratic, or even a cubic relationship between diatom diversity and Zn concentrations. We see that the data suggest that there is a linear, and even a quadratic (i.e., curvilinear) response. This definitely is consistent with what we see in Fig. 3.1. For now, we will not go into detail about what all of the output means. Instead, we will just use this information to help us qualitatively describe the relationship between our response variable and the treatment factor.

### 3.6 Model II ANOVA

Up to this point, everything we have considered is in the context of Model I ANOVA. That is, we have assumed that our treatment factor of interest (ZNGROUP) was a well-defined, repeatable, *fixed* treatment factor. There is another factor in the dataset we may be interested in. The observations came from a variety of different streams. We may want to know if differences among streams contributes to a significant amount of variation in our response variable. Here, we consider the streams we measured to be a *random* sub-sample of all possible streams. Thus, we want to test:

$$\begin{aligned}
 H_o: & \text{ there is no added variance in diversity due to streams} \\
 H_a: & \text{ there is added variance in diversity due to streams}
 \end{aligned}$$

To do so, we use a slightly different model,

$$Y_{ij} = \mu + A_i + \epsilon_{ij}, \tag{3.2}$$

where the  $A_i$  represent the variation in diversity that is due to variability among streams. Now, let's first do the Model II ANOVA using the same functions we just learned.

```

> m2 = lm(DIVERSTY ~ STREAM, data = d)
> anova(m2)

```

Analysis of Variance Table

Response: DIVERSTY

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
STREAM	5	1.8278	0.3656	1.4108	0.2508
Residuals	28	7.2552	0.2591		

Because we only have one factor, there are no differences in the calculations between a Model I and Model II analysis. The  $MS_{Residuals}$  represents the variance in diversity due to the sampling locations within a stream. We would then use the formulas in Table 8.5 to calculate variance that could be attributed to streams. Alternatively, let's take a look at an alternative approach to dealing with random effects factors. The `nlme` library is used to evaluate mixed-effects models. A mixed-effects model has both fixed and random factors, and, although we only have a random factor here, we can use the `nlme` library. In fact, the `nlme` library can be used any time you have at least one random factor (it does not handle cases with no random factors).

```
> library(nlme)
> m3 = lme(DIVERSITY ~ 1, data = d, random = ~1 | STREAM)
```

Now, the syntax for creating the model is a little different than we have seen before. We use `lme()`, which is the mixed-effects equivalent to `lm()`. The first argument is a formula specification of the fixed effects component of the model. Because we have no fixed effects factors, we use a 1 to represent an intercept (i.e., mean values). The last argument is a specification of the random effects component of the model. What it is literally saying is, 'consider the mean diversity within streams a random parameter'. The syntax will seem a little clunky and weird at first, but `lme()` is a very powerful tool for examining more complicated mixed effects models.

To get the variance estimates, we will use `summary()`. `intervals()` can even be used to put confidence intervals on these estimates.<sup>1</sup>

```
> summary(m3)
```

Linear mixed-effects model fit by REML

```
Data: d
      AIC      BIC    logLik
60.25615 64.74568 -27.12808
```

Random effects:

```
Formula: ~1 | STREAM
      (Intercept) Residual
StdDev:  0.1433068 0.5075011
```

Fixed effects: DIVERSITY ~ 1

```
      Value Std.Error DF  t-value p-value
(Intercept) 1.693271  0.105306 28 16.07953      0
```

Standardized Within-Group Residuals:

```
      Min      Q1      Med      Q3      Max
-2.0786852 -0.7689358  0.2106570  0.6706556  2.0945380
```

Number of Observations: 34

Number of Groups: 6

```
> intervals(m3)
```

<sup>1</sup>see section 1.3.2 in Pinheiro and Bates (2000) for a discussion of the validity of these CI's with unbalanced data. This book is pretty much *the* standard reference for mixed-effects models, which are very rapidly gaining in popularity.

Approximate 95% confidence intervals

```
Fixed effects:
      lower      est.      upper
(Intercept) 1.477561 1.693271 1.908981
attr("label")
[1] "Fixed effects:"

Random Effects:
Level: STREAM
      lower      est.      upper
sd((Intercept)) 0.01900776 0.1433068 1.080444

Within-group standard error:
      lower      est.      upper
0.3911509 0.5075011 0.6584602
```

One advantage in using the `nlme` library is that we can specify other methods for calculating the variance components. The default method, called REML, is most often the best choice. Note that the estimates are given as standard deviations. (Square them and you will get values equal to those on p.175 of Quinn and Keough (2002).) So, the variance in diatom diversity due to variation among sampling stations within a stream is 0.258 (i.e.,  $0.5075011^2$ ). On the other hand, the variance that can be attributed to variation among streams is only 0.021 (i.e.,  $0.1433068^2$ ). Variation among streams does not appear to be all that important.

# Chapter 4

## General Linear Model

In this exercise, you will

- continue to gain familiarity with R
- practice the use of `lm()`, the function for creating and examining linear models
- illustrate the use of dummy variables to demonstrate the relationship between ANOVA and regression
- illustrate ANCOVA

### 4.1 The Data

To begin, let's look at a very simple dataset from Zar (1999). It contains data from an evaluation of the systolic blood pressure from patients of a variety of ages. (Remember, you must have the working directory set!)

```
> options(width = 85)
> blood = read.csv("zar_example17_8.csv")
> blood
```

```
   age sbp
1   30 108
2   30 110
3   30 106
4   40 125
5   40 120
6   40 118
7   40 119
8   50 132
9   50 137
10  50 134
11  60 148
12  60 151
13  60 146
14  60 147
```

```

15 60 144
16 70 162
17 70 156
18 70 164
19 70 158
20 70 159

```

```
> summary(blood)
```

```

      age      sbp
Min.   :30.0  Min.   :106.0
1st Qu.:40.0  1st Qu.:119.8
Median :55.0  Median :140.5
Mean   :52.5  Mean   :137.2
3rd Qu.:62.5  3rd Qu.:152.2
Max.   :70.0  Max.   :164.0

```

```
> plot(blood$sbp ~ blood$age, xlab = "Age", ylab = "Systolic Blood Pressure",
+      pch = 16, cex.lab = 1.5, cex.axis = 1.5)
```

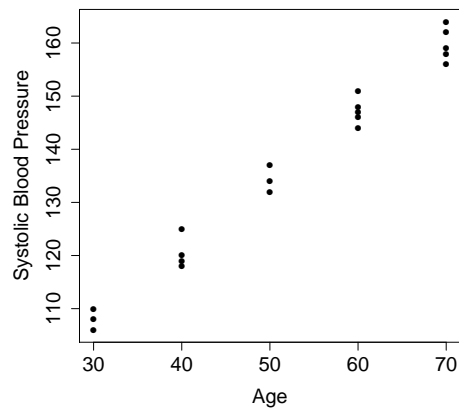


Figure 4.1: Scatterplot showing the blood pressure data.

## 4.2 Creating Linear Models

As you will recall, the primary function for developing a linear model is `lm()`. Take a look at `?lm`.

### 4.2.1 Simple Least Squares Regression

Let's start by assuming that the ages are actual measured values, and that we would like to test for a linear response of systolic blood pressure with age.

```
> m1 = lm(sbp ~ age, data = blood)
> attributes(m1)
```

```

$names
 [1] "coefficients" "residuals"      "effects"      "rank"        "fitted.values"
 [6] "assign"       "qr"            "df.residual"  "xlevels"     "call"
[11] "terms"       "model"

$class
 [1] "lm"

```

The output from the linear model fit is saved in an object of class 'lm'. There are a variety of other functions that are useful for working with such objects. Some of these (there are many others!) are:

- `plot()` - get a series of diagnostic plots
- `summary()` - summarize the linear model results
- `predict()` - get predicted values from the model
- `resid()` - get the residuals from the model
- `anova()` - summarize the model results in an ANOVA table.

Let's take a look at some of these.

```

> par(mfrow = c(2, 2))
> plot(m1)

```

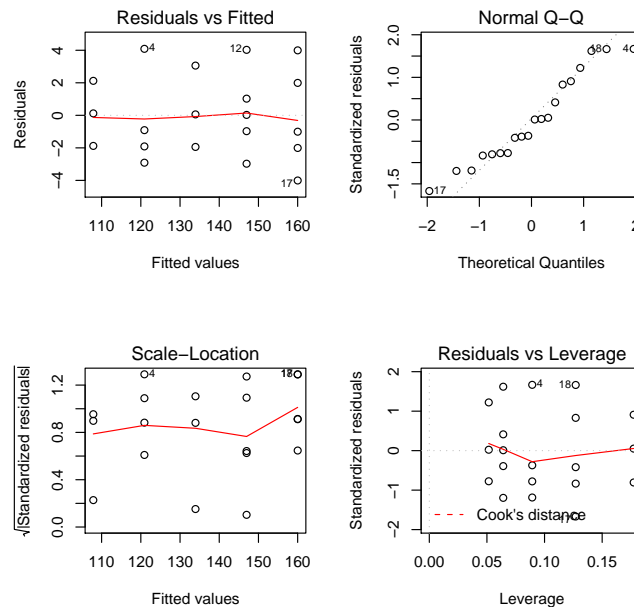


Figure 4.2: Diagnostic plots of a model fit using `lm()`. Notice that the default action of `plot`, when given an object of class 'lm', is to create 4 different plots. We used `par()` to put all of these in one plot window.

The diagnostic plots in Figure 4.2 are useful for evaluating the fit of the model to the data. In particular, the residual plots will reveal potential heteroscedasticity. The Q-Q plot reveals potential deviations from normality.

```

> summary(m1)

Call:
lm(formula = sbp ~ age, data = blood)

Residuals:
    Min       1Q   Median       3Q      Max
-4.0050 -1.9186 -0.4421  2.0264  4.0893

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 68.78491    2.21607   31.04  <2e-16 ***
age          1.30314    0.04077   31.97  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.57 on 18 degrees of freedom
Multiple R-squared:  0.9827,    Adjusted R-squared:  0.9817
F-statistic: 1022 on 1 and 18 DF,  p-value: < 2.2e-16

```

The summary contains the results of the regression analysis, including the slope and intercept estimates. For each coefficient in the model, the appropriate standard error, t-statistic, and p-value are given. These tests evaluate the hypothesis:

$$\begin{aligned}
 H_o: & \beta_i = 0 \\
 H_a: & \beta_i \neq 0
 \end{aligned}$$

```

> predict(m1)

     1      2      3      4      5      6      7      8      9
107.8792 107.8792 107.8792 120.9107 120.9107 120.9107 120.9107 133.9421 133.9421
    10     11     12     13     14     15     16     17     18
133.9421 146.9736 146.9736 146.9736 146.9736 146.9736 160.0050 160.0050 160.0050
    19     20
160.0050 160.0050

```

The function `predict()` is useful for getting predicted values for any values of the independent variable. These predicted values can be used to add the regression line (and even confidence and prediction bounds) to our previous plot. Alternatively, we can add the regression line to our plot using `abline()`.

```

> plot(blood$sbp ~ blood$age, xlab = "Age", ylab = "Systolic Blood Pressure",
+      pch = 16, cex.lab = 1.5, cex.axis = 1.5)
> abline(m1)

```

## 4.2.2 Analysis of Variance

Now - let's go back and look at the way that ANOVA is conducted within the regression framework. To do so, let's assume that the ages are really just categories. We will use `factor()` to tell R to interpret the variable `age` as a factor. Then, we can use the function `anova()` to create the ANOVA table from the linear model.

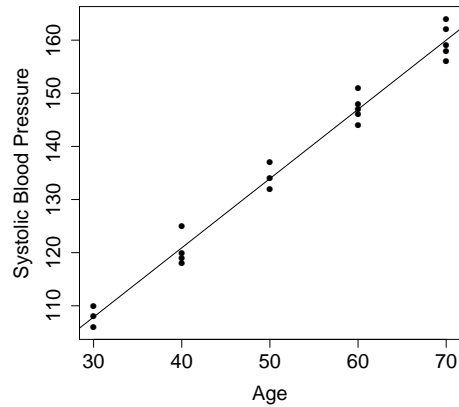


Figure 4.3: Scatterplot showing the blood pressure data with the regression line.

```
> blood$ageF = factor(blood$age)
> attributes(blood$ageF)
```

```
$levels
[1] "30" "40" "50" "60" "70"
```

```
$class
[1] "factor"
```

```
> m2 = lm(sbp ~ ageF, data = blood)
> anova(m2)
```

#### Analysis of Variance Table

```
Response: sbp
      Df Sum Sq Mean Sq F value    Pr(>F)
ageF    4  6751.9  1688.0  215.92 4.622e-13 ***
Residuals 15  117.3    7.8
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

#### Treatment Contrasts

Now - let's look at how the linear model is parameterized.

```
> summary(m2)
```

```
Call:
lm(formula = sbp ~ ageF, data = blood)

Residuals:
    Min       1Q   Median       3Q      Max
-3.8000 -1.8500 -0.4167  2.0500  4.5000
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )	
(Intercept)	108.000	1.614	66.903	< 2e-16	***
ageF40	12.500	2.136	5.853	3.17e-05	***
ageF50	26.333	2.283	11.535	7.41e-09	***
ageF60	39.200	2.042	19.198	5.71e-12	***
ageF70	51.800	2.042	25.368	9.83e-14	***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.796 on 15 degrees of freedom  
Multiple R-squared: 0.9829, Adjusted R-squared: 0.9784  
F-statistic: 215.9 on 4 and 15 DF, p-value: 4.622e-13

What do these coefficients represent? By default, R will convert the factor `age` to a series of dummy variables, using 'treatment' contrasts. You can look at how this is accomplished using `contrasts()`

```
> contrasts(blood$ageF)
```

```
      40 50 60 70
30  0  0  0  0
40  1  0  0  0
50  0  1  0  0
60  0  0  1  0
70  0  0  0  1
```

This is the 'contrast matrix', and it tells us that the estimate of the intercept in the summary table is the mean sbp in the group labeled 30. The other 'slopes' are actually estimates of the change in mean as you go from the first group (30) to the other groups.

To get a better idea of what these dummy variables look like. Let's read in another dataset.

```
> d = read.csv("zar_example17_8_dummy2.csv")
> d
```

```
   age sbp d1 d2 d3 d4
1   30 108  0  0  0  0
2   30 110  0  0  0  0
3   30 106  0  0  0  0
4   40 125  1  0  0  0
5   40 120  1  0  0  0
6   40 118  1  0  0  0
7   40 119  1  0  0  0
8   50 132  0  1  0  0
9   50 137  0  1  0  0
10  50 134  0  1  0  0
11  60 148  0  0  1  0
12  60 151  0  0  1  0
13  60 146  0  0  1  0
14  60 147  0  0  1  0
15  60 144  0  0  1  0
16  70 162  0  0  0  1
```

```

17 70 156 0 0 0 1
18 70 164 0 0 0 1
19 70 158 0 0 0 1
20 70 159 0 0 0 1

```

Here, four dummy variables, d1 to d4, have been created. We can use these to create the linear model.

```

> m3 = lm(sbp ~ d1 + d2 + d3 + d4, data = d)
> summary(m3)

```

Call:

```
lm(formula = sbp ~ d1 + d2 + d3 + d4, data = d)
```

Residuals:

```

      Min       1Q   Median       3Q      Max
-3.8000 -1.8500 -0.4167  2.0500  4.5000

```

Coefficients:

```

              Estimate Std. Error t value Pr(>|t|)
(Intercept)  108.000     1.614   66.903 < 2e-16 ***
d1             12.500     2.136    5.853 3.17e-05 ***
d2             26.333     2.283   11.535 7.41e-09 ***
d3             39.200     2.042   19.198 5.71e-12 ***
d4             51.800     2.042   25.368 9.83e-14 ***
---

```

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

```

Residual standard error: 2.796 on 15 degrees of freedom
Multiple R-squared:  0.9829,    Adjusted R-squared:  0.9784
F-statistic: 215.9 on 4 and 15 DF,  p-value: 4.622e-13

```

So, this is exactly the same summary that we had using `age` as a factor. This is exactly what R does for factors.

## Polynomial Contrasts

Because there is actually a well-defined, quantitative order to the levels of the factor `age`, we could tell R to take this ordering into consideration. Here, instead of creating a new variable, I will just use `factor()` in the model statement within `lm()`.

```

> m3 = lm(sbp ~ factor(age, ordered = T), data = blood)
> anova(m3)

```

Analysis of Variance Table

Response: sbp

```

              Df Sum Sq Mean Sq F value    Pr(>F)
factor(age, ordered = T)  4  6751.9  1688.0  215.92 4.622e-13 ***
Residuals                15   117.3    7.8
---

```

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

So, by specifying that `age` is an 'ordered factor,' we get the exact same ANOVA table. However, the parameterization of the linear model is different. In R, the default for an ordered factor is 'polynomial contrasts'.

```
> contrasts(factor(blood$age, ordered = T))
```

```
      .L      .Q      .C      ^4
30 -0.6324555  0.5345225 -3.162278e-01  0.1195229
40 -0.3162278 -0.2672612  6.324555e-01 -0.4780914
50  0.0000000 -0.5345225 -4.095972e-16  0.7171372
60  0.3162278 -0.2672612 -6.324555e-01 -0.4780914
70  0.6324555  0.5345225  3.162278e-01  0.1195229
```

```
> summary(m3)
```

Call:

```
lm(formula = sbp ~ factor(age, ordered = T), data = blood)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-3.8000 -1.8500 -0.4167  2.0500  4.5000
```

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      133.9667     0.6417  208.779 < 2e-16 ***
factor(age, ordered = T).L  41.2045     1.4211  28.994 1.38e-14 ***
factor(age, ordered = T).Q  -0.2049     1.4789  -0.139  0.892
factor(age, ordered = T).C  -0.5060     1.3506  -0.375  0.713
factor(age, ordered = T)^4   0.3586     1.4845   0.242  0.812
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 2.796 on 15 degrees of freedom

Multiple R-squared: 0.9829, Adjusted R-squared: 0.9784

F-statistic: 215.9 on 4 and 15 DF, p-value: 4.622e-13

Now, the contrast matrix is a bit more complicated. However, from it, we can look for evidence of a linear, quadratic, cubic, or even quartic response. (I'll let you decide how to interpret a quartic response!)

### 4.2.3 Analysis of Covariance

Analysis of covariance is appropriate when you have both a categorical and a continuous independent variable. Consider the following example from Sokal and Rohlf (1995). These data represent the membrane potential ( $y$ ) for four different cation systems ( $g$ ) as a function of the log of activity ratio ( $x$ ). We wish to test whether the mean membrane potential is different for the different cation systems, taking into account the covariate (activity ratio).  $y$  is the dependent variable,  $g$  is the categorical independent variable, and  $x$  is the continuous covariate. Note the use of `attach()` and `detach()`. These allow you to get access to all of the variables stored in the dataframe titled 'membrane' without having to constantly type `membrane$` before every variable name. These should only be used with caution! It is very easy to confuse yourself if you overuse `attach()`.

```

> membrane = read.csv("sokal_example14_9.csv")
> attach(membrane)
> plot(y ~ x, pch = as.numeric(system))
> legend(-1.7, 40, legend = levels(system), pch = 1:4, bty = "n")
> detach(membrane)

```

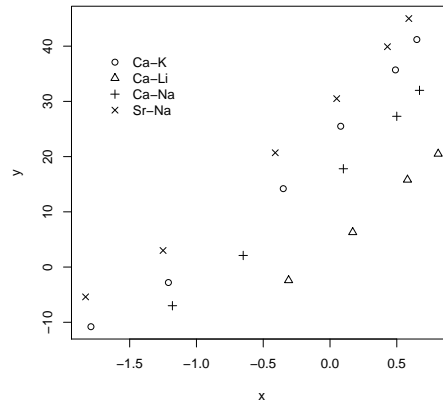


Figure 4.4: Data on cation systems from Sokal and Rohlf (1995). Note the use of `legend()` to get the legend placed on the plot.

Let's first look at the parameters we get when we fit a model with both a categorical and a continuous independent variable. Again, the categorical variable is really treated as a series of dummy variables using treatment contrasts.

```

> m4 = lm(y ~ x * system, data = membrane)
> summary(m4)

```

Call:

```
lm(formula = y ~ x * system, data = membrane)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-3.0736	-1.1065	0.1687	1.0612	2.7338

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )	
(Intercept)	24.7616	0.8393	29.504	2.69e-13	***
x	21.3941	0.8798	24.316	3.19e-12	***
systemCa-Li	-21.1696	1.4504	-14.595	1.93e-09	***
systemCa-Na	-7.9619	1.2046	-6.609	1.69e-05	***
systemSr-Na	5.9383	1.1995	4.950	0.000265	***
x:systemCa-Li	-0.7282	2.4048	-0.303	0.766830	
x:systemCa-Na	-0.3255	1.5012	-0.217	0.831707	
x:systemSr-Na	-0.5266	1.2486	-0.422	0.680085	

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.908 on 13 degrees of freedom

Multiple R-squared: 0.9921, Adjusted R-squared: 0.9879  
 F-statistic: 234.1 on 7 and 13 DF, p-value: 1.210e-12

The first two coefficients are estimates of the intercept and slope of the relationship between y and x within the first cation system. The next three ('systemCa-Li', 'systemCa-Na', and 'systemSr-Na') represent the change in intercept as you go from the first system (Ca-K) to the others. The interaction terms are the changes in slope. Thus, we have four slopes and four intercepts.

```
> attach(membrane)
> plot(y ~ x, pch = as.numeric(system))
> legend(-1.7, 40, legend = levels(system), pch = 1:4, bty = "n")
> lines(x[system == "Ca-K"], predict(m4)[system == "Ca-K"])
> lines(x[system == "Ca-Li"], predict(m4)[system == "Ca-Li"])
> lines(x[system == "Ca-Na"], predict(m4)[system == "Ca-Na"])
> lines(x[system == "Sr-Na"], predict(m4)[system == "Sr-Na"])
> detach(membrane)
```

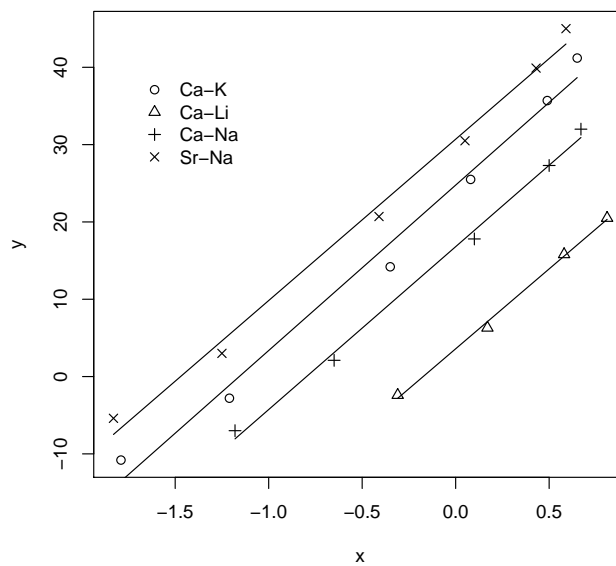


Figure 4.5: Data on cation systems from Sokal and Rohlf (1995). Here, regression lines are illustrated.

In any ANCOVA, the first thing to do is to see if the relationship between the dependent variable and the covariate is consistent among the levels of the categorical independent variable. By consistent, we mean that the slopes within each of the groups (as defined by the categorical variable) are the same. This can be done by the following:

```
> anova(m4)
```

Analysis of Variance Table

```
Response: y
      Df Sum Sq Mean Sq  F value    Pr(>F)
```

```

x          1 4197.0 4197.0 1152.7173 4.431e-14 ***
system     3 1768.6 589.5 161.9151 1.521e-10 ***
x:system   3 0.8 0.3 0.0729 0.9735
Residuals 13 47.3 3.6

```

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

In the ANOVA table above, the F test for the interaction term (x:system) tests the null hypothesis that the slopes of the relationships between membrane potential and activity ratio are the same within each cation system.

Because we have no indication of different slopes (i.e., the interaction term from above was non-significant), we can use a common slope for each cation system. We recalculate the linear model and the resulting ANOVA table.

```

> m5 = lm(y ~ x + system, data = membrane)
> summary(m5)

```

Call:

```
lm(formula = y ~ x + system, data = membrane)
```

Residuals:

```

      Min       1Q   Median       3Q      Max
-3.07213 -1.11170 -0.04373  1.06533  2.83495

```

Coefficients:

```

              Estimate Std. Error t value Pr(>|t|)
(Intercept)  24.6547     0.7291  33.815 2.60e-16 ***
x             21.0929     0.4900  43.050 < 2e-16 ***
systemCa-Li -21.1962     1.1663 -18.173 4.17e-12 ***
systemCa-Na  -7.8522     1.0569  -7.429 1.43e-06 ***
systemSr-Na   6.1362     1.0016   6.126 1.46e-05 ***

```

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

```

Residual standard error: 1.734 on 16 degrees of freedom
Multiple R-squared: 0.992, Adjusted R-squared: 0.99
F-statistic: 495.8 on 4 and 16 DF, p-value: < 2.2e-16

```

Notice that now, the model includes one slope that is used for each of the cation systems.

```
> anova(m5)
```

Analysis of Variance Table

Response: y

```

      Df Sum Sq Mean Sq F value    Pr(>F)
x          1 4197.0 4197.0 1395.25 < 2.2e-16 ***
system     3 1768.6 589.5 195.98 8.005e-13 ***
Residuals 16 48.1 3.0

```

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Now, the F test for the categorical variable (cation system) tests the null hypothesis that the intercepts of the relationships between membrane potential and activity ratio are the same within each cation system. This is the test of mean differences in membrane potential among cation systems, after removing the effects of activity ratio. Let's take a look at the model itself.

```
> attach(membrane)
> plot(y ~ x, pch = as.numeric(system))
> legend(-1.7, 40, legend = levels(system), pch = 1:4, bty = "n")
> lines(x[system == "Ca-K"], predict(m5)[system == "Ca-K"])
> lines(x[system == "Ca-Li"], predict(m5)[system == "Ca-Li"])
> lines(x[system == "Ca-Na"], predict(m5)[system == "Ca-Na"])
> lines(x[system == "Sr-Na"], predict(m5)[system == "Sr-Na"])
> detach(membrane)
```

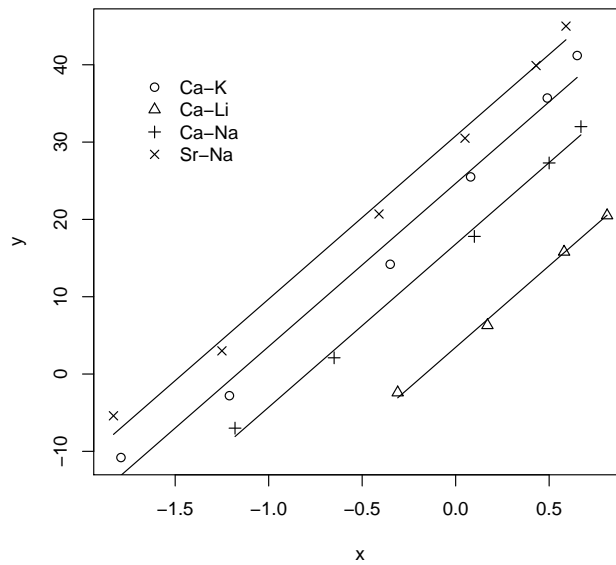


Figure 4.6: Data on cation systems from Sokal and Rohlf (1995). Here, the regression lines (each with a slope of 21.09) are illustrated.

## Chapter 5

# Generalized Linear Models

In this exercise, you will

- continue to gain familiarity with R
- practice the use of `glm()`, a function for creating generalized linear models
- create a logistic regression model for a dose response relationship.

### 5.1 The Data

The following data represent the response of tobacco budworms (death) that were exposed to varying concentrations of trans-cypermethrin. They come from an example on pg. 190 of Venables and Ripley (2002).

```
> options(width = 85)
> conc = c(1, 2, 4, 8, 16, 32)
> y = c(1, 4, 9, 13, 18, 20)
```

where `y` represents the number of individuals that died at each concentration. Obviously, this number does not mean much without knowing how many individuals were actually exposed in each group. It turns out, 20 individuals were exposed in each group.

```
> n = c(20, 20, 20, 20, 20, 20)
```

Now, let's look at the proportion of individuals that died in each group.

```
> p = y/n
> plot(conc, p, xlab = "Concentration(ppb)", ylab = "Proportion",
+      pch = 16, cex.lab = 1.5, cex.axis = 1.5)
```

The concentrations are obviously on a log scale. Let's log-transform the concentration values.

```
> logconc = log2(conc)
> plot(logconc, p, xlab = "log(Concentration(ppb))", ylab = "Proportion",
+      pch = 16, cex.lab = 1.5, cex.axis = 1.5)
```

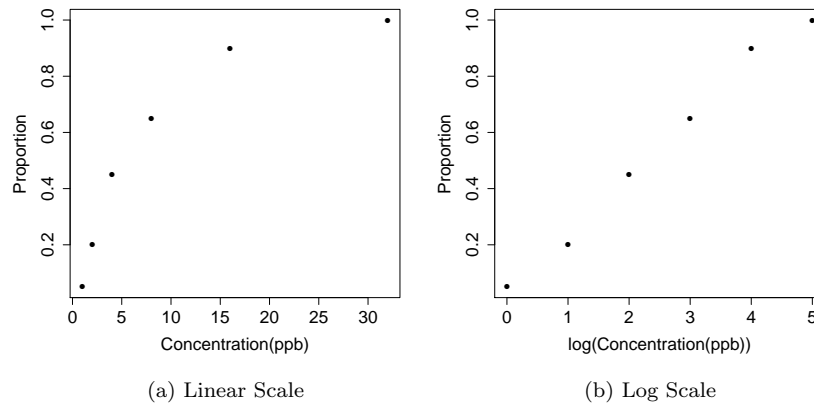


Figure 5.1: Scatterplots of the data.

Based on an examination of Fig. 5.1b, we could choose to just fit a linear regression using  $\log(\text{Concentration})$  as the independent variable.

```
> m = lm(p ~ logconc)
> plot(logconc, p, xlab = "log(Concentration(ppb))", ylab = "Proportion",
+      pch = 16, cex.lab = 1.5, cex.axis = 1.5)
> abline(m)
```

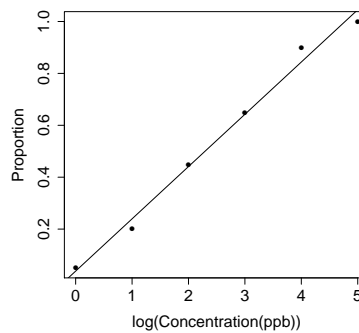


Figure 5.2: Scatterplot showing the linear model fit.

However, there are a couple of problems with this approach. First of all, we know that the proportions (our response variable) are not normally distributed. Hence, linear regression is not really appropriate. More importantly, the model makes predictions that do not make sense. For example, the model predicts proportions above 1 and below 0. This is unrealistic.

## 5.2 Generalizing the Linear Model

The GLM will allow us to model our response variable more appropriately. Let's fit a logistic GLM model to the data. This is the same thing as doing a 'logit' regression. The function for fitting a generalized linear model is `glm()`. Take a look at `?glm`. A main difference from `lm()` is the `family`

argument that tells R what error distribution to use for the response variable. The available error distributions can be seen using `?family`. Each of these error distributions also has an associated default link function.

### 5.2.1 The response variable

For a binomial response, the default link function is the logit, where:

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) \quad (5.1)$$

Thus, we are going to create a linear model with the natural log of  $\frac{p}{1-p}$  as a response variable.  $\frac{p}{1-p}$  is really just the 'odds' of dying. For an individual in this experiment, what are the odds that it dies? Well, it is just the number that died divided by the number that lived, or:

$$\frac{1+4+9+13+18+20}{19+16+11+7+2+0} = 1.18$$

So, if you get thrown into this experiment, odds are 65:55, or 1.18 that you will die! For the logistic model, we want to model the natural log of the odds of death (i.e., the log-odds of death) as a function of dose.

### 5.2.2 Fitting the model

We have a couple of different ways we can specify the response variable. First, we can just use the proportions. However, we would still need to specify the sample size within each group, and we could do this with the `weights` argument. Alternatively, we can just use a matrix with two columns (the number that died, and the number that lived) as the response variable.

```
> m1 = glm(p ~ logconc, family = binomial(link = "logit"), weights = n)
```

or

```
> raw = cbind(y, n - y)
> m1 = glm(raw ~ logconc, family = binomial("logit"))
> summary(m1)
```

Call:

```
glm(formula = raw ~ logconc, family = binomial("logit"))
```

Deviance Residuals:

1	2	3	4	5	6
-0.12505	0.30463	0.22204	-0.71011	-0.02679	1.10375

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )
(Intercept)	-2.8186	0.5480	-5.143	2.70e-07 ***
logconc	1.2589	0.2121	5.937	2.91e-09 ***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

```
Null deviance: 71.1376 on 5 degrees of freedom
Residual deviance: 1.8810 on 4 degrees of freedom
AIC: 20.228
```

Number of Fisher Scoring iterations: 4

One thing we may be interested in reporting is the dose that is required to kill 50% of the individuals (i.e., the LC50). `dose.p()` in the MASS library is useful for this. Check out `?dose.p`.

```
> library(MASS)
> lc50 = dose.p(m1, p = 0.5)
> lc50
```

```
          Dose          SE
p = 0.5: 2.238815 0.2039871
```

In addition to the LC50, we get an estimate of its standard error. We can approximate the 95% CI around this by using  $LC50 \pm 1.96 * se$ . This approximation (sometimes called a Wald interval) is based on a normal sampling distribution.

```
> lc50[1] + 1.96 * attr(lc50, "SE")
```

```
          [,1]
p = 0.5: 2.63863
```

```
> lc50[1] - 1.96 * attr(lc50, "SE")
```

```
          [,1]
p = 0.5: 1.839000
```

The use of `attr()` is necessary only because of the particular way in which `dose.p` stores its result. The 'SE' is an 'attribute' of the result. Remember, generalized linear models cannot rely on ordinary least squares techniques to estimate coefficients. In fact, they use maximum likelihood estimators. As a result, we cannot use F-tests in an ANOVA table. However, we can rely on an approximate  $\chi^2$  distribution.

```
> anova(m1, test = "Chisq")
```

Analysis of Deviance Table

Model: binomial, link: logit

Response: raw

Terms added sequentially (first to last)

	Df	Deviance	Resid. Df	Resid. Dev	P(> Chi )
NULL			5	71.138	
logconc	1	69.257	4	1.881	8.645e-17

### 5.2.3 Plotting the results

Now, let's add the prediction line to the plot. To do this, we must first create a new data frame to hold new values of all independent variables from which we want to predict the new values. We then use `predict()` to get the predicted values.

```
> pred.frame = data.frame(logconc = seq(0, 5, 0.1))
> phat = predict(m1, pred.frame, type = "response")
> plot(logconc, p, xlab = "log(Concentration(ppb))", ylab = "Proportion",
+      pch = 16, cex.lab = 1.5, cex.axis = 1.5)
> lines(pred.frame$logconc, phat)
```

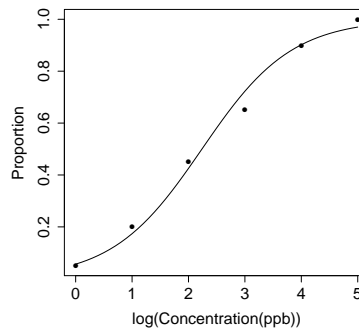


Figure 5.3: Scatterplot showing the logit model fit.

## 5.3 Analysis of Covariance

OK, let's complicate things a little bit. The previous data were for males only, but we are interested in the possible effects of sex on response - so, let's add in the females. To make this easier, you can just read in a new dataset called `glmexample.csv`. (Make sure you are in the correct working directory - or, give a full directory specification.)

```
> d = read.csv("glmexample.csv")
> d$logconc = log2(d$conc)
> d
```

	conc	sex	dead	alive	logconc
1	1	m	1	19	0
2	2	m	4	16	1
3	4	m	9	11	2
4	8	m	13	7	3
5	16	m	18	2	4
6	32	m	20	0	5
7	1	f	0	20	0
8	2	f	2	18	1
9	4	f	6	14	2
10	8	f	10	10	3
11	16	f	12	8	4
12	32	f	16	4	5

To start, let's just look at how the odds of dying may depend on sex (ignoring concentration for the time being). Note that now we have to use the `data` argument to tell R that the variables `dead`, `alive`, and `sex` are within the dataframe titled `d`.

```
> m2 = glm(cbind(dead, alive) ~ sex, family = binomial("logit"), data = d)
> summary(m2)
```

Call:

```
glm(formula = cbind(dead, alive) ~ sex, family = binomial("logit"),
     data = d)
```

Deviance Residuals:

```
      Min       1Q   Median       3Q      Max
-4.7887 -2.9371  0.1015  2.3400  4.9522
```

Coefficients:

```
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.4754      0.1878  -2.532  0.0113 *
sexm         0.6425      0.2623   2.449  0.0143 *
```

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

```
Null deviance: 124.88 on 11 degrees of freedom
Residual deviance: 118.80 on 10 degrees of freedom
AIC: 152.91
```

Number of Fisher Scoring iterations: 4

### 5.3.1 Odds Ratios

Like in the ANOVA models we have seen before, R is using treatment contrasts to develop a dummy variable that codes for sex. Thus, the intercept is the  $\ln\left(\frac{p}{1-p}\right)$ , or the log-odds of death, for females (i.e.,  $\ln(\text{Odds}_{females})$ ). The estimate for `sexm` represents the change in the log-odds as you go from being a female to being a male. In fact, it is:

$$\beta_1 = \ln\left(\frac{\text{Odds}_{males}}{\text{Odds}_{females}}\right) \quad (5.2)$$

We can use `exp()` to get rid of the log (remember that 'log' in R refers to the natural log).

```
> exp(coef(m2)[2])
```

```
sexm
1.901186
```

Thus, males are 1.901 times as likely to die as are females. This number is referred to as the *odds ratio* (OR). It is:

$$OR = e^{\beta_i} = \frac{\text{Odds}_{males}}{\text{Odds}_{females}} \quad (5.3)$$

(See if you can't calculate this by hand using the raw data.) The interpretation of coefficients from logistic regression as odds ratios is an important concept!

### 5.3.2 Examining the model

Now, we can just fit the full model that includes both dose and sex.

```
> m3 = glm(cbind(dead, alive) ~ sex * logconc, family = binomial("logit"),
+ data = d)
> summary(m3)
```

Call:

```
glm(formula = cbind(dead, alive) ~ sex * logconc, family = binomial("logit"),
    data = d)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.39849	-0.32094	-0.07592	0.38220	1.10375

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )	
(Intercept)	-2.9935	0.5527	-5.416	6.09e-08	***
sexm	0.1750	0.7783	0.225	0.822	
logconc	0.9060	0.1671	5.422	5.89e-08	***
sexm:logconc	0.3529	0.2700	1.307	0.191	

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 124.8756 on 11 degrees of freedom  
Residual deviance: 4.9937 on 8 degrees of freedom  
AIC: 43.104

Number of Fisher Scoring iterations: 4

```
> anova(m3, test = "Chisq")
```

Analysis of Deviance Table

Model: binomial, link: logit

Response: cbind(dead, alive)

Terms added sequentially (first to last)

	Df	Deviance	Resid. Df	Resid. Dev	P(> Chi )
NULL			11	124.876	
sex	1	6.077	10	118.799	0.014
logconc	1	112.042	9	6.757	3.499e-26
sex:logconc	1	1.763	8	4.994	0.184

Because the interaction between `sex` and `logconc` is not significant, we have no evidence that the slope of the relationship between  $\text{logit}(p)$  and dose differs between males and females. Hence, we can simplify the model.

```
> m4 = glm(cbind(dead, alive) ~ logconc + sex, family = binomial("logit"),
+ data = d)
> summary(m4)
```

Call:

```
glm(formula = cbind(dead, alive) ~ logconc + sex, family = binomial("logit"),
    data = d)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.10540	-0.65343	-0.02225	0.48471	1.42944

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )	
(Intercept)	-3.4732	0.4685	-7.413	1.23e-13	***
logconc	1.0642	0.1311	8.119	4.70e-16	***
sexm	1.1007	0.3558	3.093	0.00198	**

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 124.876 on 11 degrees of freedom  
 Residual deviance: 6.757 on 9 degrees of freedom  
 AIC: 42.867

Number of Fisher Scoring iterations: 4

And we can get the odds ratios.

```
> exp(coef(m4))
```

(Intercept)	logconc	sexm
0.031019	2.898560	3.006400

Now we see that if you increase the  $\log_2$ (Concentration) by one unit, you are 2.9 times as likely to die. In addition, after *controlling* for logconc, males are 3.01 times as likely to die as are females. This is the same thing as saying, for a given concentration, males are 3.01 times as likely to die as are females.

To extract the LC50 values, we need to have a slope and intercept for each group. However, as it is currently parameterized, the model coefficients contain a common slope, but the estimate of the intercept for males is really the *change* in intercept as you go from females to males. We can get the actual intercept for males by re-parameterizing the model. By adding `-1` to the model formula, we tell R to drop the overall intercept from the model. Now, look at the coefficients.

```
> m4 = glm(cbind(dead, alive) ~ logconc + sex - 1, family = binomial("logit"),
+ data = d)
> summary(m4)
```

```
Call:
glm(formula = cbind(dead, alive) ~ logconc + sex - 1, family = binomial("logit"),
    data = d)
```

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.10540  -0.65343  -0.02225   0.48471   1.42944
```

```
Coefficients:
      Estimate Std. Error z value Pr(>|z|)
logconc  1.0642    0.1311   8.119 4.70e-16 ***
sexf    -3.4732    0.4685  -7.413 1.23e-13 ***
sexm    -2.3724    0.3855  -6.154 7.56e-10 ***
---

```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)
```

```
Null deviance: 126.227 on 12 degrees of freedom
Residual deviance: 6.757 on 9 degrees of freedom
AIC: 42.867
```

```
Number of Fisher Scoring iterations: 4
```

Then, we pass these coefficients to `dose.p()`. For males, we have:

```
> dose.p(m4, c(3, 1), 0.5)
```

```
          Dose      SE
p = 0.5: 2.229262 0.2259649
```

And for females, we have:

```
> dose.p(m4, c(2, 1), 0.5)
```

```
          Dose      SE
p = 0.5: 3.263587 0.2297539
```

We can also use ANOVA to test for overall effects.

```
> anova(m4, test = "Chisq")
```

```
Analysis of Deviance Table
```

```
Model: binomial, link: logit
```

```
Response: cbind(dead, alive)
```

```
Terms added sequentially (first to last)
```

	Df	Deviance	Resid. Df	Resid. Dev	P(> Chi )
NULL			12	126.227	
logconc	1	20.621	11	105.606	5.598e-06
sex	2	98.849	9	6.757	3.429e-22

### 5.3.3 Plotting the results

There are a number of ways we could plot the model predictions. For example, we could do it using code similar to that used in Fig.5.3 for males and females separately. Here, let's use `curve()`. This will take any function that is passed to it, evaluate it along a continuum, and plot the results as a curve. So, before we can use it, we must create a function that will give model predictions. We will do this for the males (call it `pm`), and females (`pf`). We will also add a legend using `legend()`.

```
> d$p = d$dead/(d$dead + d$alive)
> par(mar = c(5, 5, 4, 2))
> plot(d$logconc, d$p, col = as.numeric(d$sex), xlab = "log(Concentration(ppb))",
+      ylab = "Proportion", pch = 16, cex.lab = 1.5, cex.axis = 1.5)
> pm = function(x) predict(m4, newdata = data.frame(logconc = x, sex = as.factor("m")),
+      type = "response")
> pf = function(x) predict(m4, newdata = data.frame(logconc = x, sex = as.factor("f")),
+      type = "response")
> curve(pm, 0, 5, add = T, col = "red")
> curve(pf, 0, 5, add = T)
> legend(0, 1, legend = c("Males", "Females"), col = c("red", "black"),
+       lty = 1, pch = 16, bty = "n")
```

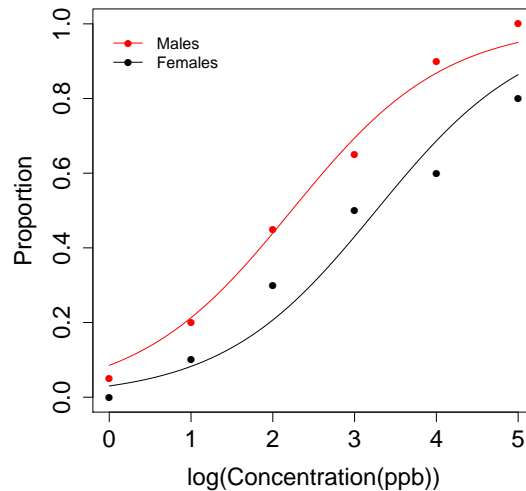


Figure 5.4: Scatterplot showing the logit model fit of males and females.

# Bibliography

Dalgaard, P. 2002. Introductory statistics with R. Springer, New York.

Pinheiro, J. C. and D. M. Bates. 2000. Mixed-effects models in S and S-PLUS. Springer, New York.  
URL <http://www.loc.gov/catdir/enhancements/fy0816/99053566-d.html>.

Quinn, G. P. and M. J. Keough. 2002. Experimental design and data analysis for biologists. Cambridge University Press, Cambridge, UK. URL  
<http://www.loc.gov/catdir/samples/cam033/2001037845.html>.

Sokal, R. R. and F. J. Rohlf. 1995. Biometry: the principles and practice of statistics in biological research. W.H. Freeman, New York, 3rd ed edition.

Venables, W. N. and B. D. Ripley. 2002. Modern applied statistics with S. Springer, New York, 4th ed edition.

Zar, J. H. 1999. Biostatistical analysis. Prentice Hall, Upper Saddle River, N.J., 4th ed edition.