Section 2 – Statistical tests in R

This second section of the R manual will provide you with the skills to conduct a range of different statistical tests in R. Section 2 relates directly to chapters 1, 2, 3, 5, 6, and 8 of the textbook, and will allow you to conduct the analyses used by the original researchers of the experiments described in the Ecology in Action textbook, and produce many of the figures you see. This section will therefore equip you with many of the skills you need to analyse your own data in the future. Many of the sections require you to use the skills you have learned in section 1 (such as how to bring data in to R), so you may need to refer back if you get stuck. Don't worry though, R is just like learning a language, the more you use it the more natural it becomes; what feels difficult now will become second nature with practice.

The statistical tests you learn in this section should allow you to analyse the vast majority of data you encounter during your undergraduate degree, and will form the basis of post-graduate learning. By the time you complete this section, you should also be able to look up more complex statistical tests online, install the packages to run them, and analyse more difficult data.

CONTENTS

2.1 – Numerical or continuous variables (corresponds to Chapter 1 of the textbook)

2.1.1 – Correlation analysis

2.1.2 – Variables, regression analyses, and P-values

2.1.3 – (Advanced option) Data transformations

2.2 – Means and measures of variation (corresponds to Chapter 2 of the textbook)

2.2.1 – Calculating means, variances, standard deviations and standard errors

2.2.2 – (Advanced option) Building your own standard error function

2.3 – Confidence intervals (corresponds to Chapter 3 of the textbook)

2.4 – Categorical variables, frequencies, and contingency tables (Chapter 5 of the textbook)

2.5 – Testing for differences between treatments (Chapter 6 of the textbook)

2.5.1 – Analysis of Variance (ANOVA)

2.5.2 – Calculating treatment mean and standard errors with "tapply()"

2.5.3 – Post-hoc testing, Tukey tests

2.5.4 – T-tests

2.6 – Logarithms, exponents and their uses

2.6.1 – Logarithmic axes

2.6.2 – (Advanced option) – Using transformations and "lm()" to quantify power relationships

2.1 – Numerical or continuous variables (corresponds to Chapter 1 of the textbook)

2.1.1 – Correlation analysis (see Dealing with data 1.1).

Recall from chapter 1 of the textbook that correlation looks at relationships between numerical or continuous variables (i.e. both variables can be counted or measured). In this section we shall first do a worked example of a correlation analysis, then you will do a rainfall vs. grass example that we will compare to the results in the textbook. In both cases we shall look at the **correlation coefficient (r)**, the parameter that describes the strength of the correlation, and produce a graph.

Worked example – scorpion venom

While working in a field station in Costa Rica, myself and colleagues regularly encountered worryingly large scorpions on an almost daily basis. Despite continuous assurances that they weren't dangerous, it started me thinking about whether there was a relationship between scorpion size and the amount of venom it contained. Below I've shown some example data we can use to understand the relationship between scorpion size and weight. In the example we shall try to find out if larger scorpions have more venom than smaller ones. Bring the data set in to R and call it "weight.venom" using the code below (For Mac users, you will have to slightly modify the code, refer to section 1.6).

```
weight.venom<-read.csv("weight vs. venom.csv")
### looking at the data
weight.venom
     weight   venom.vol
1     4.6      0.19920
2     6.0      0.26840
3     4.9      0.19098
4     4.9      0.22098
5     5.1      0.24200
6     5.3      0.21060
7     5.3      0.21060
8     5.5      0.24100
9     5.6      0.27000
10    6.1      0.26000
```

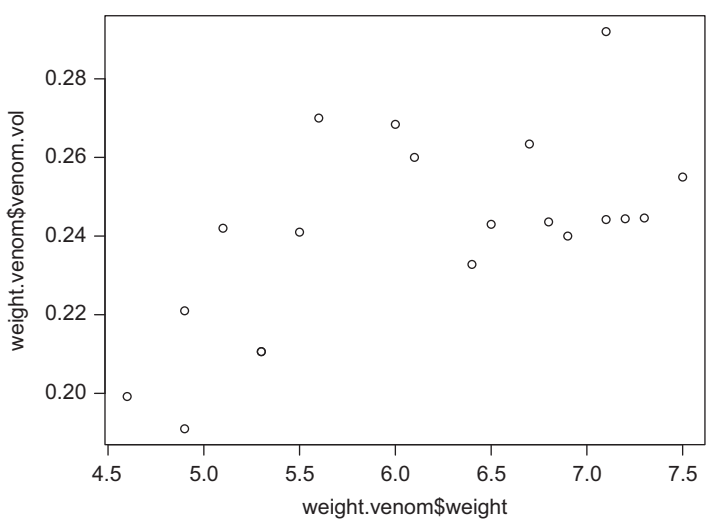
```
11     6.4       0.23280
12     6.5       0.24300
13     6.7       0.26340
14     6.8       0.24360
15     6.9       0.24000
16     7.1       0.29200
17     7.1       0.24420
18     7.2       0.24440
19     7.3       0.24460
20     7.5       0.25500
```

First, we'll plot the data to see what they look like. We know what the two columns in the dataset "weight.venom" are called ("weight" and "venom.vol"). Recall that to identify a column within a data set, we use "$". So to look at "weight" within "weight.venom" we use:

```
weight.venom$weight
[1]  4.6 6.0 4.9 4.9 5.1 5.3 5.3 5.5 5.6 6.1 6.4 6.5 6.7 6.8 6.9 7.1 7.1 7.2 7.3 7.5
```

The line of numbers underneath are the weights of the different scorpions. Now we can use the "plot" command to look at the relationship between weight and venom volume

```
plot(weight.venom$weight,weight.venom$venom.vol)
```



There appears to be a bit of a correlation, so let's have a look using the "cor" function to give us our r-value. Remember, r=1 is a perfect positive correlation, r=-1 is a perfect negative correlation, and r=0 is no correlation at all.

```
cor(weight.venom$weight,weight.venom$venom.vol)
[1]  0.6174748
```

So we have found an r-value of 0.62, indicating that there does appear to be some positive correlation between weight of an individual and the volume of venom it carries.

A little revision on graphics

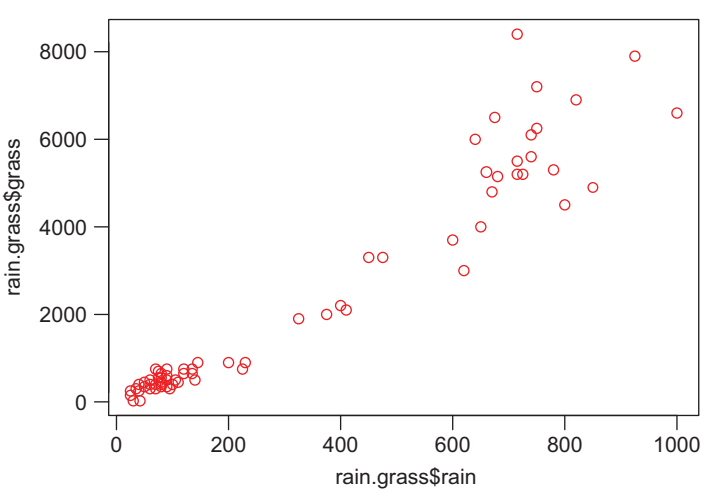
Remember from section 1.8 that it's possible to customise graphics in R a great deal. We shall not go through them all here, but two very useful commands are "xlab" and "ylab". Using these you can label the axis on a graph anything you like. For example, using the line below you can make the weight vs venom plot with the correct labels on the axis. Be careful with the placement of commas and inverted commas, as if they will generate error messages if not put in correctly and the plot won't be drawn.

```
plot(weight.venom$weight,weight.venom$venom.vol,xlab="Scorpion Weight",
ylab="Venom volume")
```

Moving forward, investigating new data

Tony Sinclair, the author of the rainfall study detailed in the textbook has kindly provided data he collected. The data are contained in the data set "rain vs grass.csv". Load in this data, find the names of the columns, plot the data using the plot command, and then calculate the correlation coefficient. These data were collected from open woodland in the Serengeti and are slightly different from the original example in the textbook. How does the plot, and the correlation coefficient compare with the book example and what does this suggest?

```
rain.grass<-read.csv("~/Ecology in action/rain vs grass.csv")

names(rain.grass)

plot(rain.grass$rain,rain.grass$grass)
```




```
cor(rain.grass$rain,rain.grass$grass)
[1] 0.9604657
```

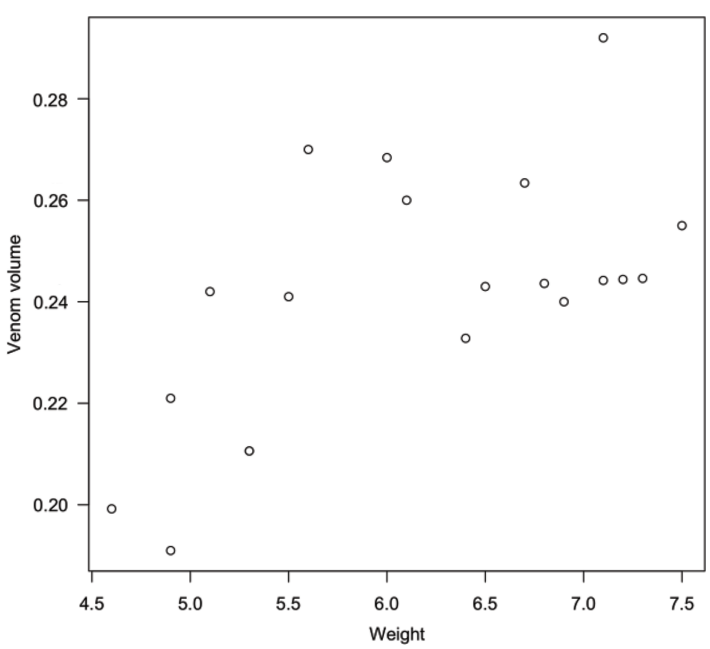
The correlation coefficient is very similar, showing that rain and grass cover are very tightly related to each other regardless of whether you're looking at woodland or open areas. However, the amount of grass that grows in woodland is slightly less, possibly due to shading by trees

2.1.2 – Variables, regression analysis and P-values (see Dealing with data 1.2)

While correlation coefficients are useful, they don't actually tell us whether or not a relationship is *statistically significant*. The formal test to understand whether a relationship between an independent and dependent variable is **statistically significant** is known as regression. In R, regression is implemented using **linear modelling.** With linear modelling we can now ask "by how much does Y change with a one unit change of X?, and is this relationship statistically significant?".

We shall initially go through a regression example by re-visiting the scorpion venom data from Section 2.1.1. of the R manual. However, now we will now q*uantify* the relationship between weight (the **independent** variable) and venom volume (the **dependent** variable) and formally test its statistical significance. In doing so, we are testing our research hypothesis (H_R) that heavier scorpions contain more venom. As before, bring the data set "weight vs. venom.csv" into R, call it "weight.venom".

Make a plot with scorpion weight on the x-axis, and venom on the y-axis, you should have the same graph as in section 2.1.1



Remember, by convention, the independent variable is always on the x-axis, and the dependent variable is always on the y-axis.

We will now test the significance of the relationship between weight and venom using regression. We shall produce a linear model to describe the relationship, to do this we use the "lm" (for "linear model) command in R.

```
model.1<-lm(weight.venom$venom.vol~weight.venom$weight)
```

If you don't get an error message, it worked! You have now built a model in R named "model.1". The "~" inside the model literally translates as "by", so with the command line we're saying "do a regression looking at how venom volume changes with weight". This method of using the "~" (called a "tilda") is very common in R.

We can now get the results of our model using the "summary" command. . .

```
summary(model.1) ### Getting the results

lm(formula = weight.venom$venom.vol ~ weight.venom$weight)

Residuals:
      Min          1Q      Median          3Q         Max
-0.029231   -0.014436  -0.008339  0.014551   0.038145

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.138703    0.030997   4.475    0.000293 ***
weight        0.016634    0.004995   3.330    0.003722 **
-
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.02019 on 18 degrees of freedom
Multiple R-squared: 0.3813, Adjusted R-squared: 0.3469
F-statistic: 11.09 on 1 and 18 DF, p-value: 0.003722
```

What R has actually done with the "lm" command is to fit a straight line to our data, using the normal equation for a straight line. . .

$$Y = B_0 + B_1X$$

Recall from the textbook that this is called the regression formula. The Y represents the value of the dependent variable (venom volume) and X represents the value of the independent variable (scorpion weight). In the R output given above, the values in the "estimates" column give the values for B_0 and B_1 in the regression formula. The estimate of the "intercept" from the output is above is B_0 and the estimate for "weight" is B_1. Recall that this estimate for B_1 is the amount that our Y variable (the dependent variable "venom volume") will change with a change of 1 unit in our X variable (the independent variable "weight"). R has essentially built us a model describing the relationship between scorpion weight and the volume of venom it contains. This model can be written out in the same form as the regression equation by using the values in the "estimates" column.

$$\text{venom volume} = 0.00029 + 0.0037^{*}\text{weight}$$

We can then use this to predict the amount of venom contained in different scorpions. For example, if we have a scorpion that weighs 2 grams, we would expect the volume of venom it contains to be 0.00029+0.0037*2, which equals 0.00769. If we had a scorpion weighing 5.5 grams, we would expect its venom volume to be 0.00029 +0.0037*5.5, which equals 0.02064.

Within our R output, we also have a column "Pr(>|t|)", giving us our **P-values**. This column tells us whether or not each of the intercept (B_0 in the regression equation) and the estimate for weight (B_1 in the regression equation) are statistically significant. By convention, we say that any P-value less than 0.05 is statistically significant. You will see this written in scientific papers in the form "Venon volume significantly increased with weight (P = 0.004)". However, remember that although by convention we use 0.05 as the critical value of significance, you should always interpret P-values with caution, is P = 0.06 really indicating there's no relationship? **Dealing with Data 1.2** in the textbook raises some very interesting and important points relating to P-values and statistical significance. Read through this section of the book again if you need to refresh your memory.

In addition to P-values, the "Multiple R-squared" value at the bottom of the R output tells us what proportion of the total variation in the data our model has covered, in other words how good our model is. The R^2 is calculated by dividing the amount of variation the model explains by the total variation in the data. A higher value is clearly better, but unlike a P-value where we say that a value less than 0.05 is "significant", there is no hard-and-fast rule of what constitutes a "good" R^2 value. In our above example, the adjusted R^2 is 0.3467, meaning our model explains almost 35% of the data, which I suppose isn't too bad.

Visually presenting the model

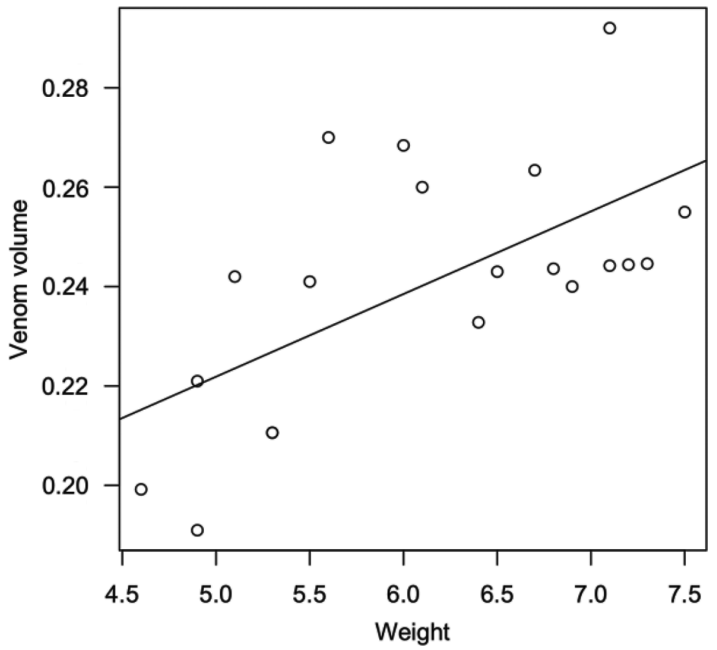
There are multiple ways of achieving the same goal of representing your regression line on the plot of your data. The easiest way is to use the function "abline". This is a very versatile function that can be used in multiple ways. To put our line on the graph, first we have to re-make the plot itself, do this using the same "plot" command line you used before in section 2.1.1. We can then either manually input the values of our line from the output of our model, putting the intercept as the first term and the estimate given for weight (the slope of the line) as the second, e.g.

```
abline(0.138703,0.016634)
```

Alternatively, we could just tell abline to take these values directly out of the model output..

```
abline(model.1)
```

You should now have something that looks like this..



The look of the line (e.g. colour) can be adjusted using the normal "plot" arguments (e.g. "col").

We now have a way of formally testing the statistical significance of a relationship between two variables (using the P-values), and can show the relationship by plotting our data and regression line.

Moving forward, investigating new data

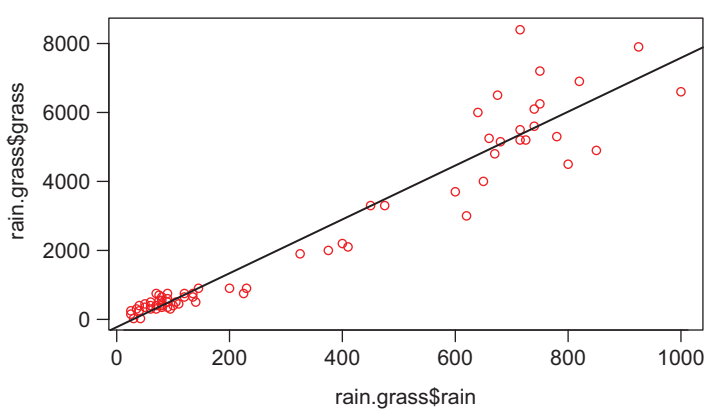
Now we have done the worked example for the scorpion data, let's investigate whether the relationship between rainfall and grass data provided by Tony Sinclair for 2.1.1. is statistically significant, and produce a plot (with regression line) that illustrates the model. The data are contained in the data set "rain vs grass.csv". Produce a research and a null hypothesis, load in this data, find the names of the columns, plot the data using the plot command, then perform the test. Given your results, how would you describe the relationship between rainfall and grass?

Research hypothesis, there is a significant relationship between rainfall and grass, as rainfall increases, grass biomass will also increase.

Null hypothesis, there is no relationship between rainfall and grass.

```
mod.1<-lm(rain.grass$grass~rain.grass$rain)
summary(mod.1)

### A COMMON MISTAKE HERE IS TO GET THE "dependent" and "independent" mixed
around between the plot() and lm() commands. If this is done the abline
below won't work. Remember, the name of the model itself is not important.
```

```
Call:
lm(formula = rain.grass$grass ~ rain.grass$rain)

Residuals:
      Min        1Q    Median        3Q       Max
 -1614.98  -210.92    15.89   251.21   3043.40

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     -225.0854   126.2918  -1.782     0.0794.
rain.grass$rain    7.8066     0.2807   27.815    <2e-16 ***
—
Signif. codes:   0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 701.2 on 65 degrees of freedom
Multiple R-squared: 0.9225, Adjusted R-squared: 0.9213
F-statistic: 773.6 on 1 and 65 DF, p-value: < 2.2e-16

### putting on the model
plot(rain.grass$rain,rain.grass$grass)
abline(mod.1)
```



Grass significantly increases with rainfall, we find support for our research hypothesis.

2.1.3 – (Advanced option) Data transformations (see Dealing with data 1.3)

As described in the Ecology in Action textbook, often we may want to represent data in different ways. As the book illustrated, representing the data as a proportion of the maximum number of elephants (maximum number = 1.0, all others are less than this) made understanding patterns in the two populations easier. Many people like to do their data transformations in excel before bringing the data into R as they are more familiar with how excel operated. However, these types of transformations in R are possible, and I thought it might be useful to describe the process here. Whether you do your transformations in R or excel will be ultimately down to your personal preference.

Using the "read.csv" command as before, bring the datasets "mara elephants.csv" and "serengeti elephants.csv" into R. Within these data sets "date" describes when a survey was conducted, "elephants" describes the number of

elephants, and "location" describes where the survey was conducted. Have a look at the data to check it's come across correctly, your first 5 rows of the mara data should look like this

```
> mara.data

   date  elephants location
1  1961        477     mara
2  1967        480     mara
3  1970        726     mara
4  1977        717     mara
5  1984        879     mara
```

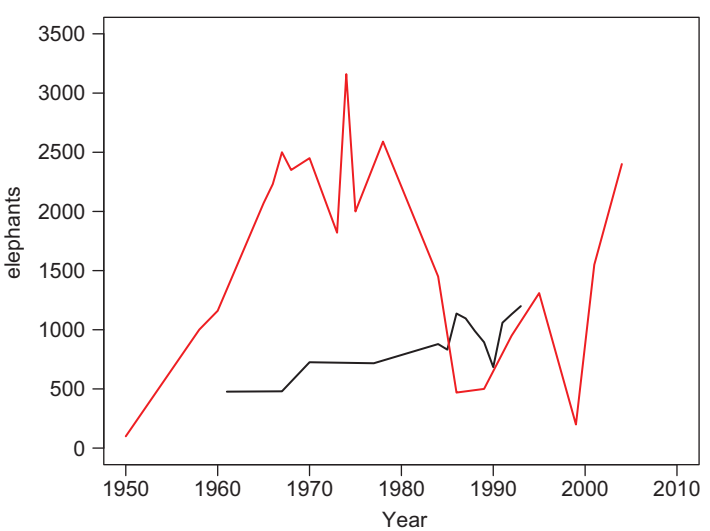
We can now plot data from both these data sets on the same graph in the same way as figure 1.12. As we only want the years 1950 -2010, we can use the function "xlim" to constrain the length of the axis. Also, as we want the y-axis to contain the full range of the data, we shall use "ylim" to determine the limits of the axis. By inserting limits for the axes into the plot command, we can determine their size, they are highlighted in the line below. Also, as we want the graph to be a line, we shall use type="1" as before. Remember, the "$" symbol is used to identify column within a data set. We have again also used the "xlab" and "ylab" commands to label the axes.

```
plot(mara.data$date,mara.data$elephants,xlim=c(1950,2010),ylim=c(0,3500),
type="l",xlab="Year",ylab="elephants")
```

To add the line for the Serengeti data, we use the "lines" command. This draws a new line on an existing plot, we shall use the "col" command to make the line a different colour. As we're now adding data from a different data set, we'll have to make a few changes to the command line. As we're adding lines to a plot, we don't need to label the axes.

```
lines(serengeti.data$date,serengeti.data$elephants,xlim=c(1950,2010),
ylim=c(0,3500),type="l",col="red")
```

This should produce the following figure, representing the same data as in figure 1.12



Now however, we need to transform these data to proportions. To do this, we make a new column within each data set, that contains each data point as a proportion. The function "max" can be used to find the highest value within an object. Dividing each of the data points by this maximum will give us their value as a proportion. We shall call our new column "prop" for proportion, and make the column for the mara data set using the line below.

```
mara.data$prop<-(mara.data$elephants/max(mara.data$elephants))
```

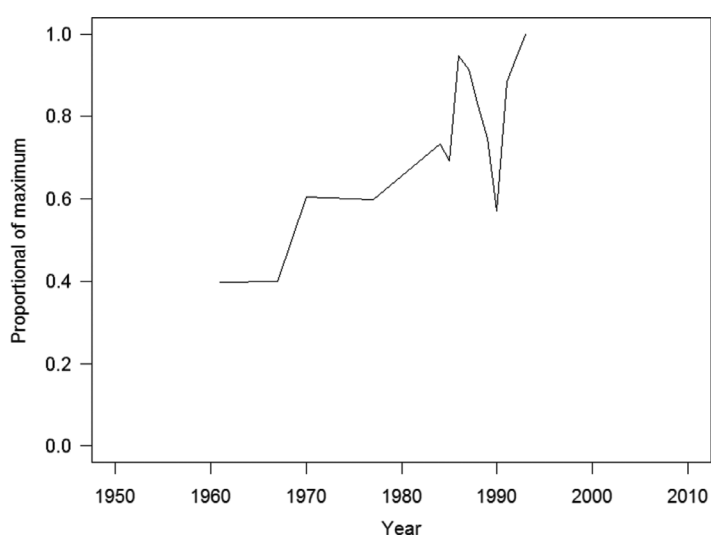
This line translates as "Make a new column in mara.data called "prop". This column is equal to the column "elephants" divided by the maximum value of the column "elephants". Have a look at the data set to check it's worked, you should see the column on the end. The first 5 rows should look like this

```
> mara.data
    date  elephants     location        prop
1   1961        477         mara   0.3975000
2   1967        480         mara   0.4000000
3   1970        726         mara   0.6050000
4   1977        717         mara   0.5975000
5   1984        879         mara   0.7325000
```

We can now plot "prop" and date, to look at how the number of elephants as a proportion of the maximum has changed over time. We can use the same "xlim" and "ylim" commands as before, but remember, our y-values have now changed and are now all between 0 and 1.

```
plot(mara.data$date,mara.data$prop,xlim=c(1950,2010),ylim=c(0,1),type="l",
xlab="Year",ylab="Proportion of maximum")
```

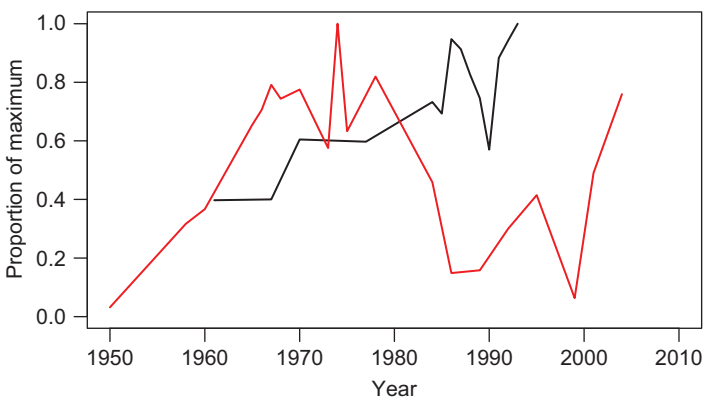
The graph should now look like this.



Notice how the y-axis is now between 0 and 1, rather than 0 and 3500.

Moving forwards, investigating new data

Now repeat the transformation for the Serengeti data. Make sure to represent the data points as proportions of the maximum value of the Serengeti data. After you have made the proportions, draw the lines for both locations on the same graph, but make the lines different colours. How has representing the data as proportions altered your perceptions of the population changes through time?



When using the actual number of elephants, it looks like the Serengeti population is in better condition. When using the proportions it's clear that the Mara population is increasing and doing well, while the Serengeti population took a big crash in the 80s and is only just now recovering.

2.2 – Means and measures of variation (see Dealing with data 2.2 and 2.3)

2.2.1 – Calculating means, variances, standard deviations and standard errors

Functions to calculate the mean, variance, and standard deviation already exist in R and are intuitive to use. The data set "temperatures.csv" contains the temperature data from chapter 2 of the book with the means removed. Bring the data set into R and call it temp.data, remember the syntax will differ slightly whether you're using a Mac or a PC, and depending on where you've saved the data (refer to chapter 0).

```
temp.data<-read.csv("temperatures.csv")

### Checking it's in
temp.data
  City         Country  Jan   Feb   Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov   Dec
1 Volgograd    Russia   -11.0  -8.0 -2.5  8.5 15.0 20.0 23.0 20.0 15.0  6.5   0.5  -2.5
2 Ulaanbaatar  Mongolia -21.1 -17.8 -8.9  1.4  9.8 15.0 16.9 15.6  8.9  0.3 -10.3 -18.9
3 Debrecen     Hungary   -2.5   0.6  5.6 11.1 15.8 18.9 20.1 20.0 16.4 11.1   4.7   0.0
4 Edmonton     Canada   -14.4 -10.8 -5.6  3.6 10.3 14.2 15.8 15.0 10.0  4.4  -5.8 -12.2
5 Topeka       USA       -2.8   0.8  6.7 12.5 17.8 23.3 25.8 24.7 20.0 14.2   5.8  -0.3
```

We can then have a look at the mean and variation for each month. The functions "mean()", "var()", and "sd()" calculate the mean, variance and standard deviation respectively. For example, to calculate each of these metrics for Jan we would use:

```
mean(temp.data$Jan)
[1]  -10.36
var(temp.data$Jan)
[1]  62.753
sd(temp.data$Jan)
[1]  7.921679
```

Use these functions to have a look at how the mean and variation in temperature changes during the course of the year as described in the book.

Calculating standard error

Strangely, R doesn't have a built in function to calculate the standard error. However, we can easily obtain the standard error using a combination of other functions. Recall that the standard error is estimated by the dividing the standard deviation by the square root of the sample size.

The **Standard Error of the Mean (SE)** = $\frac{SD}{\sqrt{N}}$

In R we can do this multiple ways. We can easily use the function "sd()" to calculate the standard deviation as above, and then manually divide this by the square root of the sample size using the function "sqrt()". For example, if we wanted to calculate the standard error for Jan. . .

```
sd(temp.data$Jan)/sqrt(5)
[1]  3.542683
```

This is the "manual" way of calculating it. However, if we didn't know our sample size, or it was too long to easily count, we can use the function "length()" to do it for us. The function "length()" simply calculates how many data points are in the object contained the parentheses, fortunately, this also happens to be the sample size *N*.

```
sd(temp.data$Jan)/sqrt(length(temp.data$Jan))
[1]  3.542683
```

2.2.2 – (Advanced option) Building your own standard error function

Recall that in section 1.1 we introduced the idea of a "function" in R, and section 1.5 highlights some important functions that you may regularly use. While it isn't too much trouble to use a combination of the functions "sd ()", "sqrt()", and "length()" to calculate the standard error each time, we could simplify it by building **our own new function** that will calculate it for us. To do this, we using the "function()" command. Here, we name a new

function, and the function acts upon whatever is in the parentheses, for convention I tend to use "x", however you can use any letter or combination of letters you like. Just try not to use a letter or word you use a lot elsewhere as R can get confused. We then use curly brackets "{}" to describe the actual function. Whenever I make a function I always insert line breaks before and after the curly brackets as I find it helps me quickly see what is in the function, but this is optional. For example, if we wanted to write a function that calculates the standard error called "se()" using the line of code from above, we can write. . .

```
se<-function(x)
{
sd(x)/sqrt(length(x))
}
```

In plain English, the above piece of code translates as "se is a function that acts upon whatever is in the parentheses, in this case x. The function takes the standard deviation of x, and divides it by the square root of the sample size of x". We can then use our new function "se()" like we would any other. For example, to calculate the standard error of the January weather from the book. . .

```
se(temp.data$Jan)
[1] 3.542683
```

Our new function works! We get the exact same result as when we calculated the standard error manually.

Although they may take a little time to initially compose, writing your own functions can seriously cut down the time it takes to complete tasks where you have a lot of repetition. To save my own functions, I have made an R script simply called "functions", whenever I'm using R I can then open this script and copy the functions into the console as needed.

2.3 – Confidence intervals (see Dealing with data 3.1)

In section 2.2 we went through how to calculate the mean, variance, and standard deviation using the functions "mean()", "var()", and "sd()" respectively. In addition, we also learned how to calculate the standard error using a combination of functions, or by writing our own function. As outlined in the textbook, the standard error holds the key to inferring statistically significant differences between treatments (although cannot always be relied on).

In this section we will learn how to quickly put data into R manually, and calculate confidence intervals. We can enter the data from Dealing with Data 3.1 as a vector using the "c()" command as before. We shall call this vector rif.rest to represent the reproductive rate of a rifampicin-resistant mutant in comparison to a wild-type mutant, when both are placed in a medium without antibiotics.

```
rif.rest<-c(0.79,0.96,0.88,1.04,0.84,0.92,0.87)
```

Check that the data have gone in okay

```
rif.rest
[1]  0.79 0.96 0.88 1.04 0.84 0.92 0.87
```

We can now calculate the mean, standard deviation and standard error as we did in section 2.2. However, this time we shall name each of these to use in subsequent calculations. The mean would therefore be

```
mean.rif.rest<-mean(rif.rest)
```

Now call "mean.rif.rest" to check it's worked

```
> mean.rif.rest
[1]  0.9
```

There is now an object in R called "mean.rif.rest", which is a single value representing the mean of the vector "rif.rest". Use this same technique of naming and calculating to produce objects of the standard deviation and standard error called "sd.rif.rest" and "se.rif.rest".

```
sd.rif.rest<-sd(rif.rest)
sd.rif.rest
[1]  0.08225975

se.rif.rest
[1]  0.03109126
```

Remember from Dealing with Data 3.1 that the 95% confidence interval is equal to the mean $\pm$ 1.96(SE). We can therefore calculate the upper and lower standard errors as follows

```
#upper
mean.rif.rest+1.96*se.rif.rest

#lower
mean.rif.rest-1.96*se.rif.rest
```

Calculate these values, does your confidence interval contain the value 1? What can you therefore conclude about the competitive abilities of the resistant bacteria based on this confidence interval?

```
### Upper
mean.rif.rest+1.96*se.rif.rest
[1]  0.9609389

### lower
> mean.rif.rest-1.96*se.rif.rest
[1]  0.8390611
```

Confidence interval does not contain 1, therefore we can conclude the resistant bacteria is a poorer competitor.

2.4 – Categorical variables, frequencies, and contingency tables (see Dealing with data 5.1)

Dealing with data 5.1 in the Ecology in Action textbook introduces the concept of the chi-squared (χ^2) test to look at differences in the frequency with which different categories are selected. The chi-squared test is very useful as it is a very robust test that makes minimal assumptions about the distribution of data, and can therefore be applied to a wide range of problems.

In this section we shall go walk through an example of how to perform a chi-squared test using some data on the survival rates of the freshwater crustacean *Daphnia* (also know as "water fleas"). You will then use the techniques you have learned to analyse Tweddle's data on seed desiccation from chapter 5 of the textbook.

Bring the data set "daphnia survival.csv") into R. The data set documents the survival rates of a freshwater zooplankton species (*Daphnia pulex*) fed two different diets, protists or algae.

```
daphnia.data<-read.csv("daphnia survival.csv")
> daphnia.data
         survival protist.diet algal.diet
1 number survived          102        210
2     number died          148         40
```

Remember, R doesn't like spaces, so the spaces in the column headings will be replaced with periods (".").

We can now very simply run a chi-squared test using the command "chisq.test". However, we have to remove the first column from the test, as it doesn't contain numbers but row names. Recall from Section 1.6 we can remove columns from a data frame by identifying them with square brackets "[ ]". With the brackets, the first number refers to the row, the second number refers to the columns. Remember that if we leave either the number denoting rows or columns blank, it will give us all the rows or columns respectively. For example to remove the first column from data frame "daphnia.data", but keep all the rows we would use. . .

```
daphnia.data[,-1]
```

We can therefore perform the chi-squared test and look at the output below.

```
> chisq.test(daphnia.data[,-1] )
Pearson's Chi-squared test with Yates' continuity correction
data: daphnia.data[,-1]
X-squared = 5.838, df = 1, p-value = 0.01568
```

In this case the P-value is less than 0.05, giving us a statistically significant result. This means that the survival rates for the *Daphnia* fed on the two different diets is not random, and that *Daphnia* fed on one diet likely survive better than *Daphnia* fed on another. However, the chi-squared test doesn't actually tell us what the pattern is in

our data, just that *the data are not random.* This is an important distinction to make. In the *Daphnia* example it's not a major deal as we only had two diets, and two possible outcomes (survived/dead). However, if we had three diets and found a statistically significant result, it would tell us that the data weren't random, but it would not tell us which diets were significantly different from each other.

Moving forwards, investigating new data

The data set "seeds.csv" contains Tweddle *et al*'s data that is described in the textbook. Bring this data set into R, and run a chi-squared test using the procedure outlined above. Given the results you achieve, would you say there is a relationship between desiccation tolerance and the production of dormant seeds? If so, what does it appear this relationship is? Support your answer.

```
chisq.test(seeds[ ,-1])

Pearson's Chi-squared test with Yates' continuity correction

data: seeds[ , -1]

X-squared = 68.504, df = 1, p-value < 2.2e-16
```

There does appear to be a relationship between desiccation tolerance and production of normal seeds. The desiccation tolerant plants produce a much greater proportion of dormant seeds, as illustrated by the tiny p-values ($p<2.2*10^{-16}$, this is very low indeed)

2.5 – Testing for differences between treatments (see Dealing with data 6.1)

2.5.1 – Analysis of Variance (ANOVA)

As described in chapter 6 of the textbook, ANOVA is a very versatile and useful test that can be used to look for significant differences between experimental treatments. In addition to be useful in its own right, the principles and outputs from an ANOVA test are at the heart of a lot of other statistical tests, so it's important to understand what the outputs mean early on.

However, you must be clear on exactly what ANOVA is testing. . ..

Null hypothesis (H_0) = There are no differences among treatment means
Research hypothesis (H_R) = at least one treatment mean is statistically different from one other

It is important to understand that the research hypothesis we are testing with ANOVA ***is not saying ALL treatments are different from ALL others***, just that at least ***one is different from one other***.

The actual ANOVA test calculates an *F*-statistic, and then looks at whether this value is greater than the critical value of *F* given our experimental conditions.

It's best demonstrated with an example. Bring the dataset "ANOVA mosquitoes.csv" into R. The data set details the number of mosquitoes that were present in 1000ml beakers placed around 3 different field stations in Costa Rica. Each row in the data set represents a single beaker. The column "mosquitoes" is our dependent variable, and "location" is our independent variable (the three field stations, either "Pitilla", "Monte.verde", or "De.salva")

```
Mosquito.data<-read.csv("ANOVA mosquitoes.csv",header=TRUE)
Mosquito.data ### Taking a look at it...
    mosquitoes      location
1            3       pitilla
2            3       pitilla
3            6       pitilla
4           12       pitilla
5           12       pitilla
6           15       pitilla
7           15       pitilla
8           16       pitilla
9           24   Monte.verde
10          32   Monte.verde
11          34      De.salva
12          35   Monte.verde
13          36   Monte.verde
14          38   Monte.verde
15          52   Monte.verde
16          67   Monte.verde
17          83   Monte.verde
18          88   Monte.verde
19          95   Monte.verde
20         100      De.salva
21         100      De.salva
22         102      De.salva
23         102      De.salva
24         105      De.salva
25         110      De.salva
26         113      De.salva
27         120      De.salva
28         121      De.salva
29         148       pitilla
30         152       pitilla
```

Based on these 10 samples from each location, we want to know whether there are statistically significant differences in mosquito abundance among the three locations using ANOVA. We do this using the "aov()" command in R. Unlike the chi-squared test described in 2.5, for the aov() command we have to actually name the test something, we'll call it "anova.1". We use the "$" to identify columns within the data frame as we've done before, and "~" to denote "by".

```
anova.1<-aov(Mosquito.data$mosquitoes~Mosquito.data$location)
```

The above line of codes translates as “make a new object called “anova.1”, this object is an ANOVA test of our dependent variable (the column “mosquitoes”), by our independent variable (the column “location”)”. Make sure you can understand what each part of the code relates to.

When you run the test, it doesn’t automatically give you the output, this is because you’ve just produced the test within R, to look at the results, you need to use the “summary()” command..

```
summary(anova.1)

This should then give you the following ANOVA table...

                          Df Sum Sq Mean Sq F value  Pr(>F)
Mosquito.data$location     2  20923   10462    6.55  0.0048 **
Residuals                 27  43124    1597
- - -
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘’ 1
```

The above output contains everything you need to report your result, R has calculated the *F*-statistic and the P-value. In addition the degrees of freedom have been calculated in the DF column. We will not go through what they are in detail, but they relate to the number of independent samples within the data.

To formally report the result from an ANOVA, we would write *“Our results support the research hypothesis that there are statistically significant differences between mosquito abundances among the three field stations ($F = 6.55$, $p = 0.0048$, ANOVA)”.* This one sentence conveys all the information we want to get across. Remember that by convention a significant difference is denoted by $p < 0.05$.

However, rather than just know whether there are differences among the field station, we may wish to know which stations differ from each other, so we can identify which has the most mosquitoes (and thus avoid spending too much time there)

2.5.2 – Calculating treatment mean and standard errors using “tapply()”

The first step in finding which treatments are statistically different from each other is to calculate the means and standard errors of the different treatments. To calculate the means and standard errors for each treatment, we could manually input the data on mosquito abundances into different vectors, and then calculate the means and standard errors. However, inputting the data manually like this is time consuming and there’s a chance that you may input errors. Instead, we can use the very useful function “tapply()”. This function applies a command to each level within a treatment. We tell tapply the variable that we’re interested in, how we would like that variable split (or indexed), and finally the function we’d like to apply. For example, using our “Mosquito.data” data frame, we’re interested in the abundance of mosquitoes, split by each location, and want to calculate the mean. We therefore use the following command line

```
tapply(Mosquito.data$mosquitoes, Mosquito.data$location,mean)

that should produce...

De.salva        Monte.verde   pitilla
 100.7             55.0         38.2
```

We now have the mean number of mosquitoes at each of the different field stations. The standard error can be calculated in various ways. Recall from 2.2 that the standard error is equal to the standard deviation divided by the square root of the sample size (in this case 10). We can use the tapply() function to calculate our standard deviation by simply replacing the "mean" command in the above line of code with sd(). After you have calculated the standard deviations, simply dividing each by the square root of 10 will give you the standard errors. Alternatively, reload the standard error function you built in 2.2, and then replace "mean" with "se" (or whatever you called your standard error function) in the above line of code.

Your eventual standard errors should be...

```
De.salva   Monte.verde   pitilla
7.813166    8.277144    18.697475
```

Given values of the means and standard errors, which stations do you expect have statistically different numbers of mosquitoes?

2.5.3 – Post-hoc testing, Tukey tests

Although calculating the means and standard errors is informative, it doesn't formally tell us which groups are statistically different from each other. As knowing which treatments are different from each other when we've obtained a statistically significant ANOVA result is useful and interesting, we need a way to do it. Fortunately, a range of "post-hoc" tests exist to achieve exactly that. However, post-hoc tests must only be used when the result of our ANOVA (or any other test) is statistically significant.

The Tukey test

The Tukey test is among the most popularly used post-hoc tests. The beauty of this test is that it compares each of our different treatment means to each, and tells us which are significantly different. In R, it's very easy to run, we just use the "TukeyHSD()" command on whatever we named our ANOVA test, in our case (from 2.5.1), "anova.1"

```
TukeyHSD(anova.1)  ### Running a post-hoc Tukey test

This gives the following results table...

Tukey multiple comparisons of means
   95% family-wise confidence level
Fit: aov(formula = data$mosquitoes ~ data$location)
```

```
$`data$location`
                       diff     lwr          upr          p adj
Monte.verde-De.salva  -45.7   -90.01399   -1.386014   0.0422571
pitilla-De.salva      -62.5   -106.81399  -18.186014   0.0045414
pitilla-Monte.verde   -16.8   -61.11399   27.513986   0.6202390
```

Beautiful, just look at it. The TukeyHSD() command has ran the post-hoc test on our original ANOVA (in the Fit: "aov. . .." Line) and then given us P-values for all of our location comparisons in the last column of the table. We can now just read these off and report them. We see Monte verde differed from De salva, pitilla differed from De salva, but Pitilla didn't differ from Monte verde. Is this the same as what you would have expected given your treatment means and standard errors?

Moving forward, investigating new data

Now bring the data set "lizard data.csv" into R. Run an ANOVA and Tukey test as before. These data are a subset of the data analyzed by William Cooper and his colleagues (2006) as described in Ecology in Action. Use the R output to decide whether or not you find support for the research hypothesis (H_R) that there would be a relation between food abundance and approach distance. Calculate the treatment means and standard errors, and describe the results. Do your results match up with the data described in the Ecology in Action textbook?

```
lizard.data<-read.csv("~/Ecology in action/lizard data.csv")

names(lizard.data)
[1] "replicate" "maggot" "distance"

#### "maggot" is the independent variable and "distance" is the dependent

anova.lizard<-aov(lizard.data$distance~lizard.data$maggot)

summary(anova.lizard)
                   Df Sum Sq Mean Sq F value Pr(>F)
lizard.data$maggot  3   2.662  0.8873   3.83  0.013 *
Residuals          77 17.836  0.2316  -
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Our results show support for the research hypothesis that there is a relationship
between the number of maggots offered, and the distance they'll let you approach.

Post-hoc

TukeyHSD(anova.lizard)

Tukey multiple comparisons of means
   95% family-wise confidence level

Fit: aov(formula = lizard.data$distance ~ lizard.data$maggot)

$`lizard.data$maggot`
                diff          lwr         upr       p adj
four-eight  0.1007841  -0.307352531  0.5089208  0.9157779
one-eight   0.2483489  -0.152306434  0.6490043  0.3693033
```

```
zero-eight  0.5096202   0.082173103    0.9370674    0.0128923
one-four    0.1475648   -0.225488646   0.5206182    0.7272973
zero-four   0.4088361   0.007145672    0.8105266    0.0444938
zero-one    0.2612713   -0.132815458   0.6553581    0.3098779
```

Looks like zero is significantly different from four and eight, nothing else is significantly different.

```
#### means
tapply(lizard.data$distance,lizard.data$maggot,mean)
    eight        four         one          zero
0.9390059   1.0397900   1.1873548     1.4486261

#### standard error (using our own "se" function)
tapply(lizard.data$distance,lizard.data$maggot,se)
     eight        four         one          zero
0.10957310  0.12555995  0.09060716    0.09489299
```

Can see the biggest difference is between zero and eight, the mean approach distance decreases as more maggots are offered, just like the book

2.5.4 – T-tests

Recall from the textbook that a t-test is a similar to an ANOVA but used when our independent variable has only two treatments. It is used in roughly the same way as an ANOVA, but uses the command "t-test" instead of "aov".

Bring the dataset "fish diet.csv" into R. This data set describes the results of 18 fish feeding trials. Each fish was placed in a tank with 20 prey items, either copepods or daphnia, and the number they ate was recorded. What is therefore our independent variable? And what is our dependent variable?

Unlike an ANOVA, for a t-test we don't need to name the test and create an object. Instead we can just run the command straight in R..

```
t-test(fish.data$eaten~fish.data$diet)

      Welch Two Sample t-test

data: fish.data$eaten by fish.data$diet
t = -3.4543, df = 10.79, p-value = 0.005538
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-8.011379 -1.766398
sample estimates:
mean in group copepod mean in group daphnia
           3.666667          8.555556
```

Within the t-test, the t-value can be either positive or negative depending on the order that you enter the data, although the interpretation is the same. In our case, the copepod replicates were entered into the test first and as they were smaller, the t-value is negative. However, by convention we tend to present the t-value as a positive. We see from

the P-value in the above R output that we have a statistically significant result. Recall from the textbook that the t-test uses a *t*-statistic rather than an *F*-statistic, however we would report the result in a similar way. ***"We find support for the research hypothesis that fish have higher consumption rates of daphnia than they do of copepods".***

2.6 – Logarithms, exponents and their uses

2.6.1 – Logarithmic axes (see Dealing with data 8.1)

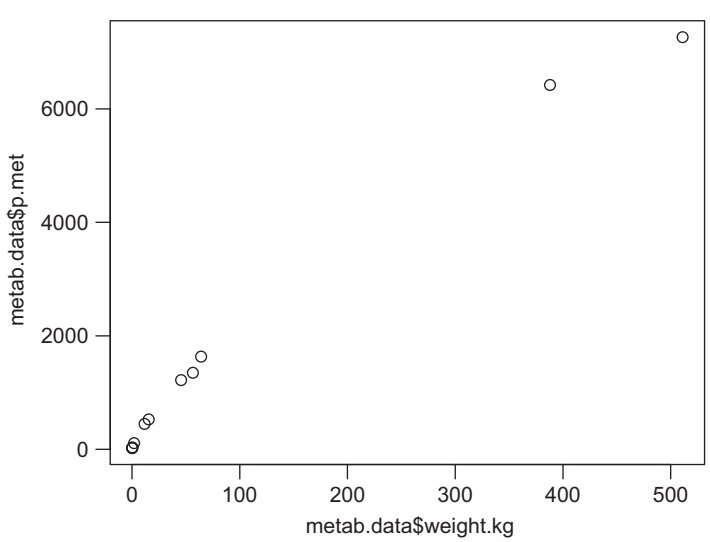
Chapter 8 of the textbook discusses how data can be transformed using $\log_{10}$ and the natural log (ln) to find power laws between relationships. Using $\log_{10}$, ln and exponents in R is relatively simple, however it is easy to make mistakes with the syntax so they must be used with caution.

The data "metabolic rate.csv" contains the data on weights and metabolic rates used by Kleiber. Bring the data in to R, name it "metab.data" for convenience.

```
metab.data<-read.csv("metabolic rate.csv")
metab.data
       Animal  weight.kg      p.met
1       steer    511.000     7265.0
2         cow    388.000     6421.0
3         man     64.100     1632.0
4       woman     56.500     1349.0
5       sheep     45.600     1219.9
6    dog.male     15.500      525.0
7  dog.female     11.600      448.0
8         hen      1.960      106.0
9      pigeon      0.300       30.8
10   rat.male      0.226       25.5
11 rat.female      0.173       20.2
12  ring.dove      0.150       19.5
```

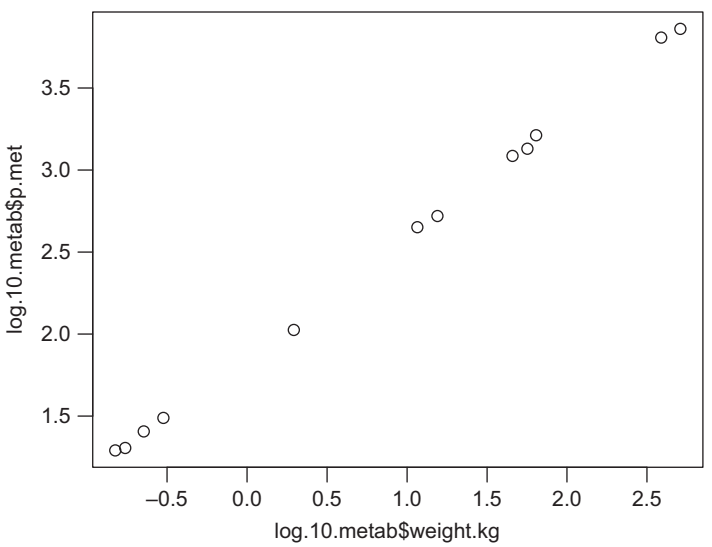
Initially, lets try plotting the untransformed data as is to have a look at the shape

```
plot(metab.data$weight.kg,metab.data$p.met)
```



As discussed in the textbook, patterns are difficult to describe as the data on animal weights span three orders of magnitude, and there are many data points at the lower end of the x-axis that overlap each other. By plotting the data on logarithmic axes, it becomes easier to see that the data are a straight line, and prevent overlap. In R, plotting data on log axes is easy using the "log" argument in the "plot" command. As we want both axes to be log, we insert log="xy" into the command, I've highlighted it in the line below.

```
plot(metab.data$weight.kg,metab.data$p.met,log="xy")
```



It's now much easier to visualise the relationship. If you wanted only the x or y axis to be on a log scale, you would use "log=x" or "log=y" respectively.

2.6.2 – (Advanced option) – Using transformations and "lm()" to quantify power relationships

We showed above that on a $\log_{10}$ scale, the relationship between weight and metabolic rate was a straight line. However, we can go a step further to quantify the relationship between weight and metabolic rate by transforming the data, using the "lm()" function from section 2.1 to find the slope of the line, and then back-transforming the parameters using exponents.

Recall from the textbook that if we $\log_{10}$ an equation of the form $Y = kX^c$ we end up with $\log_{10}(Y) = \log_{10}(k) + c*\log_{10}(X)$. As this equation is in the form of $Y = b_0 + b_1X$, we can use regression just as we did in chapter 1 to find the values for k and c.

First, we need to transform our data on weight and p_{met}, the easiest way to do this is to add two extra columns to the data frame, one containing the $\log_{10}$ of weight and the other containing the $\log_{10}$ of metabolic rate. We can do this by naming extra columns, and using the command "log10()". E.g.

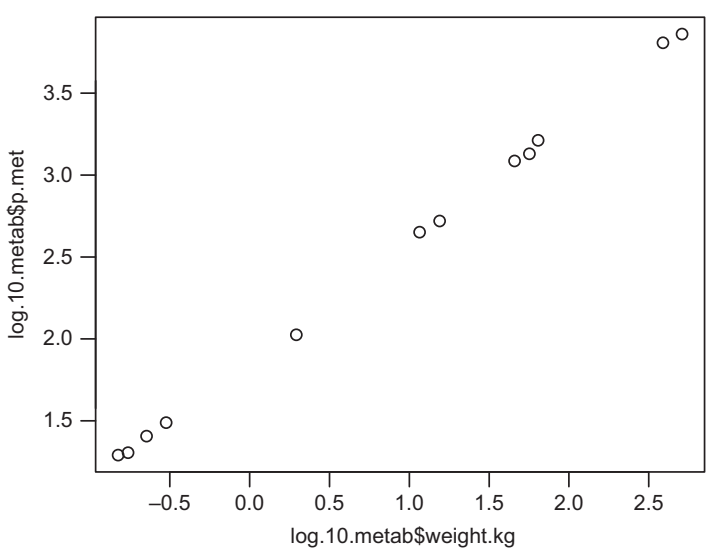
```
metab.data$log10.weight<-log10(metab.data$weight)
```

The above line of code translates as "Make a new column in data frame "metab.data" called "log10.weight". This column equals the $\log_{10}$ of the column "weight" in the data set "metab.data". Edit the above line of code to make another column in the same dataframe that gives the $\log_{10}$ of metabolic rate, call it "log10.p.met". When you've done it, have a look at the dataframe to check the new columns have been added.

```
metab.data
        Animal weight.kg  p.met log10.weight log10.p.met
1        steer   511.000 7265.0    2.7084209    3.861236
2          cow   388.000 6421.0    2.5888317    3.807603
3          man    64.100 1632.0    1.8068580    3.212720
4        woman    56.500 1349.0    1.7520484    3.130012
5        sheep    45.600 1219.9    1.6589648    3.086324
6     dog.male    15.500  525.0    1.1903317    2.720159
7   dog.female    11.600  448.0    1.0644580    2.651278
8          hen     1.960  106.0    0.2922561    2.025306
9       pigeon     0.300   30.8   -0.5228787    1.488551
10    rat.male     0.226   25.5   -0.6458916    1.406540
11  rat.female     0.173   20.2   -0.7619539    1.305351
12   ring.dove     0.150   19.5   -0.8239087    1.290035
```

The new columns are on the end. Try plotting these two new columns against each other, the graph should resemble the earlier plot with logged axis, although the axis labels will be different.

```
plot(log.10.metab$weight.kg,log.10.metab$p.met)
```

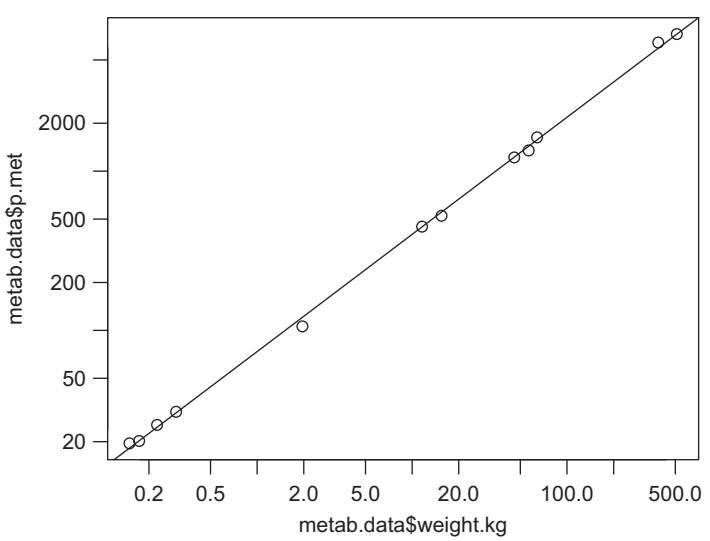


From here we can again see the relationship is a straight line. Lets now find this relation ship using "lm()" to do a regression as we did in section 2.1.2. We shall call the regression (or linear model) "metab.mod", and then look at the summary using the "summary()" command.

```
metab.mod<-
lm(metab.data$log10.p.met~metab.data$log10.weight)
summary(metab.mod)

Call:
lm(formula = metab.data$log10.p.met ~
metab.data$log10.weight)

Residuals:
      Min         1Q      Median          3Q         Max
-0.056423  -0.006473    0.001390    0.015490   0.035855
Coefficients:
                         Estimate    Std. Error t value     Pr(>|t|)
(Intercept)              1.866662      0.009310   200.5    <2e-16 ***
metab.data$log10.weight 0.735886      0.006127   120.1    <2e-16 ***
- - -
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
1

Residual standard error: 0.0266 on 10 degrees of freedom
Multiple R-squared: 0.9993, Adjusted R-squared: 0.9992
F-statistic: 1.443e+04 on 1 and 10 DF, p-value: < 2.2e-16
```

We can see from the output that our intercept (on the log scale) is 1.866662, and the slope of the line (on the log scale) 0.74. We can add the line given by these values to our plot using "abline()"

```
abline(metab.mod)
```



The equation of our line is..

$\log_{10}(\text{p.met})=1.86662+0.735*\log_{10}(\text{weight})$

This equation is now in the form $\log_{10}(Y) = \log_{10}(K)+c^*\log_{10}(X)$. We can rearrange this to the form $Y = KX^c$. At present, we have a $\log_{10}$ value for K of 1.86662, to get the arithmetic value of K, we raise 10 to the power of 1.866662. We can do this using "^" in R.

```
10^1.866662
[1] 73.56343
```

By then rearranging, and rounding, we can then find that the relationship between p_{met} (our Y-value) and weight (our x-value) is $p_{met} = 73.6^*weight^{0.74}$. This value is very close to Kleiber's ($P_{met} = 73.3M^{0.74}$)