

Advanced Quantitative methods

Duncan Golicher

2019-03-27

Contents

1	Introduction	5
1.1	The new statistics	5
1.2	Statistical models vs statistical tests	5
1.3	Bayesian vs frequentist approaches to inference	6
1.4	Using p-values with discretion	6
1.5	Common pitfalls	7
2	R programming	9
2.1	Using R to simulate data	9
2.2	Simple R programming	9
2.3	Exercise	20

Chapter 1

Introduction

In this unit you will learn to apply a range of statistical methods that are considered to be more advanced than are typically taught on an introductory course in data analysis. However the unit aims to provide more than a simple set of recipes for applying methods. Advances in computing power have resulted in an explosion of new methods for analysing ecological data. It would be impossible to design a course that includes examples of all the possible methods that might be relevant to the analysis of a data set that forms the subject of a master's dissertation. Data analysis does not simply involve applying statistical methods. Good data management and effective data manipulation are just as important as good analysis. So an important element of the course will focus on understanding the nature of data and the ways data can be manipulated.

1.1 The new statistics

The underlying philosophy of the course is based on a contemporary concept of good statistical practice. This has sometimes been called “the new statistics” (Hobbs and Hilborn, 2006)

The aim of the new statistics is to evaluate the relative strength of evidence in data for hypotheses represented as models. Traditionally, models used by ecologists for statistical inference have been limited to a relatively small set of linear forms. The functional forms and definitions of parameters in these models were chosen for statistical reasons; that is, they were not constructed to explicitly symbolize biological states and processes. Consequently, composing models to represent hypotheses has traditionally played a relatively minor role in developing testable statements by most ecological researchers. Statistical models were used to represent verbal hypotheses, and little thought was applied to the model building that ultimately supported inference.

The new statistics require deliberate, thoughtful specification of models to represent competing ecological hypotheses

1.2 Statistical models vs statistical tests

“New statistics” has developed not just as a response to advances in computing power, although that has played an important role. A fundamental element of contemporary thinking with regard to the use of statistics is the avoidance, or at the least the downplaying, of null hypothesis tests (NHST). This can be confusing for students who have taken introductory courses in statistics in which the words “test” and “p-value” constantly were emphasised. Many students (and many researchers) may be unaware that the whole basis of null hypothesis testing was in fact controversial from the outset. Many prominent statisticians have argued against NHST (Nickerson et al., 2000). Some of the criticisms that can be found in Nickerson's excellent review have been severe. For example “Null hypothesis testing provides researchers with no incentive

to specify either their own research hypotheses or competing hypotheses.... it is surely the most bone-headedly misguided procedure ever institutionalized in the rote training of science students” (Gigerenzer, 1998). The biggest problem with NHST is that it does not actually test a hypothesis of genuine interest. Some of the other arguments against NHST made by statisticians are quite technical in nature, although they are well worth trying to understand. A reasonably non-technical review is provided by Goodman (2008) in which 12 common misconceptions are laid out clearly. A very influential and highly cited paper that restates many of the criticisms of NHST made by statisticians in the context of ecology is Johnson (1999). I won’t repeat all the arguments made by Johnson. It is well worth reading the paper.

1.3 Bayesian vs frequentist approaches to inference

It is my *belief* (and the word belief is important in this context) that all students should be aware that a vigorous debate arose between two schools of thought regarding the nature of statistical inference at the time that many of the conventional, text book, statistical tests were being developed. It is fascinating to read the words of R A Fisher writing in the “Design of experiments”, first published in 1935 . Fisher stated that “*I shall not assume the truth of Bayes’ axiom*”. What is most remarkable about that particular statement is that Bayes’ axiom is a simple mathematical consequence of applying the rules of probability. There is no controversy whatsoever regarding the axiom’s mathematical validity. Fisher was aware that implementing what was known as “inverse probability” would be mathematically unfeasible in the context of the sort of problems he was working on at the time. Integration formulas to calculate marginal likelihoods are horrendously complex in even very simple cases involving continuous distributions, and are totally intractable for most designs. However his main objection to the use of Bayes’ theorem was that to him it appeared to introduce an element of subjectivity, and thus did not allow researchers to reach clear conclusions. It is unfortunate that this led to what some studies of the history of scientific thought have described as a “holy war” between two schools of thought. The unfortunate consequence of this was that efforts to remove the incorporation of subjective prior beliefs into analyses led to the conventional analytical device of NHST in which **all** prior information was ignored. This was never intended by Fisher himself. It occurred almost by accident, as a result of the development of NHST by Pearson and Neyman. Fisher originally intended p-values to be indications that further research was worth conducting, and not as decision rules regarding the truth of either a null hypothesis nor any alternative hypothesis. However for a while this got overlooked in statistical courses that emphasised the logic and mathematical basis of NHST without placing the null in the context of any field of study. Students being told to state null hypotheses in the form H_0 “There is no difference in species richness between heathland and grassland” would be quite correct in pointing out the illogical and unhelpful nature of H_0 . We already know *a priori* that there must be a difference, so why on earth would we ever test this? The answer is that we clearly should not. In an applied setting we would aim to develop a much richer and more informative analysis of the difference between the two vegetation types, conditional on the data that we have available. There is no need to place subjective prior probabilities on the size of the difference in order to do this, but to ignore all relevant knowledge regarding the two systems would clearly be quite wrong. Finding evidence against H_0 is only worth stating as an objective of a study such if H_0 is inherently credible. This can be reasonable in the context of a well planned experiment, but In all other cases there will be much more information to be obtained from the data when H_0 is not considered to be an objective of the analysis.

1.4 Using p-values with discretion

In the quantitative and spatial analysis unit I introduced a pragmatic approach towards NHST that will continue to be used on this unit. This approach emphasises confidence intervals as the most important result of applying technique based on frequentist inference. It accepts the use of p-values associated with null hypothesis tests, but only as the “statistic of last resort”. The actual numerical values of confidence intervals match those produced by Bayesian methods with non-informative priors in many cases. So, confidence intervals can informally be interpreted as measures of uncertainty regarding the parameters of interest, even

if formally they are not. When taking this approach, when running any regression analysis the most important result would be the slope of the line, and associated confidence interval. The next most valuable output is the coefficient of determination (R squared) as a measure of the signal to scatter ratio. If all else fails, then a p-value can be reported, with due discretion and care with regard to the wording of the interpretation, as a “test” of any **detectable** association between variables conditional on the data obtained. Although power analysis is highly advisable when planning any study in order to assess whether the level of replication is likely to be sufficient to provide statistical significance, when adopting this pragmatic approach you should not set out with the initial intention of “testing” a null hypothesis. Instead you should aim to estimate effect sizes. The most important element of any study is the scientific question that is of interest. Many questions can only be answered through estimating the size of effects and/or the shape and form of any response. Simply detecting a difference, an association or a correlation does not usually answer the question fully. We should, and usually can, do much better than that. This implies that non parametric methods will not be taught on this course, although it is still worth being aware of them as potential fall back methods of last resort when all else fails. Non parametric tests are easy to run and understand in R. The R code to run them is provided in this crib sheet for reference.



http://r.bournemouth.ac.uk:82/AQM/AQM_2018/Crib_sheets/Classical_statistical_tests.html



Always follow the advice of your supervisor with regards to the presentation of p-values. Some supervisors will insist on p-values being reported for all analyses. As they are included in the output of all inferential methods it is simple enough to include them in a report, even when they are not particularly informative. Just make sure that confidence intervals are also provided and used when answering questions regarding effect sizes and meaningful comparisons.

1.5 Common pitfalls

Some of the advice provided in the literature regarding the validity of statistical tests when assumptions are not met is potentially misleading. In non experimental studies the assumptions of almost any inferential method are rarely met in full. So technically almost all statistical methods might be considered to be “invalid”. In reality it is not the violation of an assumption that is important, it is the degree to which an assumption is violated and the influence that any violation may have on the validity of the substantive findings that is really important. If an analysis produces a p-value of < 0.0001 when using a method that leads to minor violations of assumptions, almost any correction will still lead to rejection of the null hypothesis. So a conclusion based only on NHST will not change at all. On the other hand confidence intervals may have to be broadened when the violations are corrected for, so a modification as a result of making a correction for unmet assumptions will take place. This is a powerful additional argument against emphasising NHST alone (naked p-values) that is not made clear in all the papers cited here. Many students have learned to test “the data” for normality. This is poor advice, for many reasons, not least of which is that students often test the wrong element in the data. The assumption of normality in regression analysis applies to the residuals, not to the raw data (Zuur et al., 2010). Rigorous diagnostics are a very important part of building statistical models, but these are best conducted through careful inspection of the patterns shown in the residuals rather than through automated statistical tests. A statistical test of normality does not actually test the data anyway. It is, in reality, testing whether the data could have been obtained from an underlying normal distribution. The only way to “test” the data themselves is to look at them carefully. As suggested by Steel et al. (2013), plot the data early and often. When looking at observational data that were not derived from a conventional experimental design, you should adopt an incremental approach to choosing an analysis. There is nothing to prevent you using the “wrong” models as initial tools for understanding the data and (hopefully) finally choosing a better, more appropriate model as the analysis progresses.

Chapter 2

R programming

During the course of this unit you will develop the skills to be able to apply R code to the analysis of a range of data sets. Although crib sheets will be provided with model code for all the analytical techniques shown on the course, in order to manipulate data effectively and apply novel techniques it is important to begin to become more comfortable with basic R programming techniques and concepts. This week we will revise some of the fundamentals of R programming.

2.1 Using R to simulate data

If you understand data structures you can usually choose and run the right analysis. If you do not consider the basic properties of your data then not only will you not know which analysis to use, but you will probably not even be able to import the data into any software in order to run it!

2.2 Simple R programming

Try running some simple commands by stepping through them in code chunks. When working in RStudio notice that the results are shown under the code chunk when

```
1+1
```

```
## [1] 2
```

Experiment some more. For example ..

```
5*10
```

```
## [1] 50
```

```
12/4
```

```
## [1] 3
```

```
3^2
```

```
## [1] 9
```

There is a lot more maths functionality built into the R language that you can find out about as you go on. However to follow the primer you do not need to learn any more commands than are shown in this document.

Note that when you type maths in the console the output is just printed to the screen. This is not stored anywhere in the computer's memory. So we need to bring in the concept of data objects. A data object "holds" the numbers and can be manipulated by commands. This is how R manages to run analyses on your data. When you import a file containing data it will be stored into a data object.

The simplest data object is a variable. If a variable only contains one number it is known formally as a scalar. A variable with many numbers is a vector. So let's assign a single value to a variable to form a scalar. We do that in R using the `<-` operator which is typed as a combination of the less than sign `<` with a horizontal bar `-`.

```
x<-5
```

Nothing appears to have happened! However you now have your first data object, a scalar called `x`. This is stored in memory. So try this.

```
x*2
```

```
## [1] 10
```

Notice that this would not have worked if you had typed `X*2` as R is case sensitive.

So, how do we form a vector with multiple values? When you are analysing your own data the answer is that you usually won't need to. You import all the values from a file. But if you wish to form a vector in the console you must use the concatenation operator `"c"`. So this gives the variable `x` a series of values.

```
x<-c(1,4,5,9,10,11,12,34,56,67,100,123,45)
```

Now see what happens if you type

```
x*2
```

```
## [1] 2 8 10 18 20 22 24 68 112 134 200 246 90
```

You can carry out any sort of mathematical operation that involves `x` and all the values in the vector will be used. Notice that the results are just printed out to the console and lost.

If you want to assign the results to a new variable you use the `"<-"` operator again. This is a very common practice when analysing data. So, say you are intending to work with the natural logarithm of `x` you might write.

```
logx<-log(x)
```

You can see that this has worked by writing the new variable name so that R prints out the contents to the console.

```
logx
```

```
## [1] 0.000000 1.386294 1.609438 2.197225 2.302585 2.397895 2.484907
## [8] 3.526361 4.025352 4.204693 4.605170 4.812184 3.806662
```

This time you can see more clearly the purpose of the indices in the output. The second line starts with the 12th number in the vector.

You can find the names of the data objects that are held in memory by typing `ls()`.

```
ls()
```

```
## [1] "logx" "x"
```

2.2.1 Data structures

Now we can begin looking at data structures. You can ask R to tell you about the structure of any object that it has in memory by typing `str()`.

```
str(x)

## num [1:13] 1 4 5 9 10 11 12 34 56 67 ...
```

```
str(logx)

## num [1:13] 0 1.39 1.61 2.2 2.3 ...
```

So R has told us that both `x` and `logx` are numerical vectors. If you have a lot of data you probably do not want to look at all of it at once.

How does this relate to choosing an analysis? We have seen that this particular variable contains numbers. However in statistical analysis we also use “variables” that don’t consist of numbers. They vary in another respect. If you look at any introductory statistical textbook you will see a chapter that defines types of variables in terms of interval, ordinal and scale variables and nominal and ordinal variables.

The sort of analysis that you will be able to run is determined by the sort of variable, or combination of variables, that you are working with.

Let’s set up a categorical variable. Experienced users of R often want to replicate values in order to set up some test data as Dytham suggests. So R has a range of functions for making data. The `rep` function replicates the values.

```
gender<-rep(c("Male","Female"),each=10)
gender

## [1] "Male" "Male" "Male" "Male" "Male" "Male" "Male"
## [8] "Male" "Male" "Male" "Female" "Female" "Female" "Female"
## [15] "Female" "Female" "Female" "Female" "Female" "Female"
```

Now if we ask R about the data structure it will tell us that we have a character vector.

```
str(gender)

## chr [1:20] "Male" "Male" "Male" "Male" "Male" "Male" "Male" "Male" ...
```

In statistics categorical variables are referred to as factors. Factors are special character vectors with numbered levels. R automatically assumes that any column in a data file that contains non numeric values is a factor and converts the data to that form. In this case we need to tell R to turn the character vector into a factor.

```
gender<-as.factor(gender)
str(gender)

## Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 ...
```

Now R tells us that `gender` is a factor with two levels. There is no intrinsic order to the levels. R places them in alphabetical order by default. The important element to be aware of is that `gender` is a variable. It varies in a known and specific way (it has two levels), but it is a variable all the same. You cannot calculate means, medians, standard deviations or any similar summary statistics using a factor. However you can, and will, use them together with numerical variables in many types of analysis.

Let’s produce a numerical variable to go alongside this. In the UK the mean height of men is around 176 cm and women 164 cm. So we could produce a vector with these values using this code. What we need to do is to replicate the “expected value” for each gender to be placed alongside the factor level.

```
height<-rep(c(176,164),each=10)
height

## [1] 176 176 176 176 176 176 176 176 176 176 164 164 164 164 164 164 164
## [18] 164 164 164
```

```
str(height)
```

```
## num [1:20] 176 176 176 176 176 176 176 176 176 176 ...
```

So now we have another numerical variable. However if we really carried out a survey of people's heights we would be absolutely amazed if we got data like this. Although the numbers represent an estimate of the expected value for each of the ten men and women we know from experience that people's heights vary around this value. In fact they vary quite a lot. The standard deviation is around 6cm. So we need to add some variability. This demonstrates clearly how statistical procedures work. Data consists of two components. It has an underlying structure that we typically want to find out about. In this case there is a difference in heights which is related to gender. There is also random variability, sometimes called the stochastic component. We are usually less interested in this directly, but we need to be very careful to make justifiable assumptions about this variability in order to correctly analyse the data.

Knowing this we can now make our variable more realistic by adding in some simulated values taken from a normal distribution with this standard deviation.

```
set.seed(1)
height<-height+rnorm(20,sd=6)
height
```

```
## [1] 172.2413 177.1019 170.9862 185.5717 177.9770 171.0772 178.9246
## [8] 180.4299 179.4547 174.1677 173.0707 166.3391 160.2726 150.7118
## [15] 170.7496 163.7304 163.9029 169.6630 168.9273 167.5634
```

We may want to round the values to one decimal place to make them equivalent to the sort of measurements we might make in practice.

```
height<-round(height,1)
```

Now we have two variables that are held in R's memory. We are assuming that they both form part of a simulated data set that we could have obtained if we had measured a stratified random sample of twenty students consisting of ten men and ten women. Let's call the survey "hsurvey" and make what is known as a data frame to hold the results.

```
hsurvey<-data.frame(gender,height)
str(hsurvey)
```

```
## 'data.frame': 20 obs. of 2 variables:
## $ gender: Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 2 2 ...
## $ height: num 172 177 171 186 178 ...
```

So we have now made up data that has the same structure and similar properties to the results we would get from carrying out a simple survey of people's heights. The basic concepts used to do this can be extended to more complex situations and used to help the design of experiments.

A data frame is the basis of almost all statistical analysis. It consists of two or more columns that line up in such a way that the measurements or observations recorded have been taken from the same individual member of a sample. This is often equivalent to a single sheet in a spreadsheet. R tells us that we have one factor and one numerical variable in this case. We might have measured many other variables at the same time and produced a much wider set of data. These may have been either categorical or numerical. Providing that they all "line up" and correspond to the same individual we have our data frame. Many standard statistical techniques use two variables together. Later on in the course you will see how we can analyse data consisting of more than two variables using multivariate analysis.

The complete data frame is shown below.

```
hsurvey
```

```
## gender height
```

```
## 1   Male  172.2
## 2   Male  177.1
## 3   Male  171.0
## 4   Male  185.6
## 5   Male  178.0
## 6   Male  171.1
## 7   Male  178.9
## 8   Male  180.4
## 9   Male  179.5
## 10  Male  174.2
## 11  Female 173.1
## 12  Female 166.3
## 13  Female 160.3
## 14  Female 150.7
## 15  Female 170.7
## 16  Female 163.7
## 17  Female 163.9
## 18  Female 169.7
## 19  Female 168.9
## 20  Female 167.6
```

If you want to remove all the other variables you set up from memory and leave just the data frame you could type.

```
remove(x,logx,height,gender)
ls()
```

```
## [1] "hsurvey"
```

Now if we ask R to print height or gender it will not find them. They have been placed within the data frame.

```
height
```

```
## Error in eval(expr, envir, enclos): object 'height' not found
```

```
gender
```

```
## Error in eval(expr, envir, enclos): object 'gender' not found
```

We can refer to the variables by writing out the full name of the data frame and the variable we want, separated by “\$”

```
hsurvey$height
```

```
## [1] 172.2 177.1 171.0 185.6 178.0 171.1 178.9 180.4 179.5 174.2 173.1
```

```
## [12] 166.3 160.3 150.7 170.7 163.7 163.9 169.7 168.9 167.6
```

```
hsurvey$gender
```

```
## [1] Male   Male   Male   Male   Male   Male   Male   Male   Male   Male
## [11] Female Female Female Female Female Female Female Female Female Female
## Levels: Female Male
```

Or we can attach the data frame. This makes the variables available directly. Notice that if we have several data frames containing variables with the same name this could cause confusion (so I don’t usually do this)

```
attach(hsurvey)
height
```

```
## [1] 172.2 177.1 171.0 185.6 178.0 171.1 178.9 180.4 179.5 174.2 173.1
```

```
## [12] 166.3 160.3 150.7 170.7 163.7 163.9 169.7 168.9 167.6
gender
## [1] Male Male Male Male Male Male Male Male Male Male
## [11] Female Female Female Female Female Female Female Female Female Female
## Levels: Female Male
```

2.2.2 Saving and loading data frames

Now that we have made up some data we might want to save it in the format that we will eventually use to capture our real data. The simplest, most portable data format is a CSV (Comma Separated Variable) file. Such a file can be easily read by all software.

First we must find a place to store the data. The standard practice is to make a separate data directory (folder) for each analysis that you carry out. You then set this as the working directory using the menu options in the R console. Once you have set your working directory (which could be on a USB stick for portability) you can save the data that is held in memory using an R command.

```
write.csv(hsurvey, file="hsurvey.csv", row.names=FALSE)
```

The command has three elements. The first is the name of the data frame that you wish to save. This is not quoted. The second is the name of the file into which you are going to save the data. The third tells R not to add row names to the file (these are not usually necessary).

Data frames are the standard input to all statistical analysis and are of a consistent form. Many different sorts of data summaries and tables can be generated from a data frame quickly by statistical software.

You can check the working directory using this command.

```
getwd()
## [1] "/home/rstudio/webpages/books/tidy_data"
##In my case I am running R on Linux so the paths look different to Windows
```

To see a list of files in the directory type `dir()`

```
dir()
##[1] hsurvey.csv
```

If we remove the dataframe we formed we will be left with nothing in R's memory

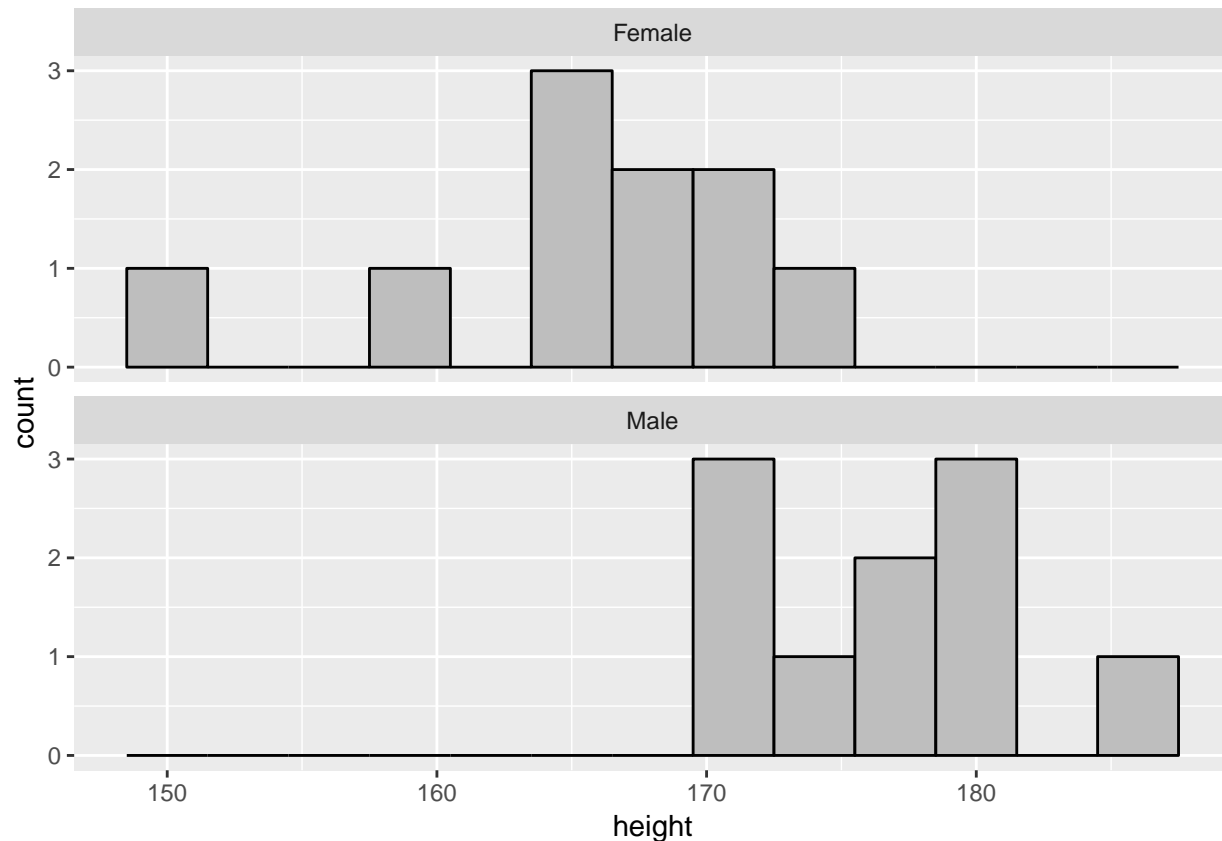
```
remove(hsurvey)
ls()
```

To load it back from the file type

```
hsurvey<-read.csv("hsurvey.csv")
str(hsurvey)
```

2.2.3 Histogram of simulated data

```
library(ggplot2)
g0<-ggplot(hsurvey, aes(x=height))
g0+geom_histogram(fill="grey", colour="black", binwidth=3)+facet_wrap(~gender, nrow=2)
```



So we have simulated some data, saved it, and reloaded the data into R. We have then looked at it as a histogram.

2.2.4 Saving a workspace

You can save the

```
save(list=ls(),file="height_study.rda")
```

2.2.4.1 Installing packages

Many of the more advanced features of R have been implemented within add on packages. These are held online at numerous mirror sites throughout the world. Packages are what makes R the important research tool that it is. Notice that the top articles from *Methods in Ecology and Evolution* present recently built R packages.

In order to use packages you have to install them to your hard disk. You do this once, and need to be online in order to download them. Once installed you make a package available at the start of a session using the command “library”. All the most useful packages have been installed already on the PCs in the lab. You may want to add some packages to your laptop. This can be done either using the menu options at the top of the console or by writing a command (if you know the names of the packages you need).

For example the following line will install the package `vegan`. This provides access to a large range of useful methods for analysing biodiversity and community data.

```
install.packages("vegan")
```

Once installed you can use `vegan` by typing

```
library(vegan)
```

```
## Loading required package: permute
```

```
## Loading required package: lattice
```

```
## This is vegan 2.5-4
```

During the course I will tell you which packages are most useful, and what they are useful for. I will also show you how to get help on functions in order to work independently. For now, just be aware that if R tells you that “there is no package called `vegan`” if you type “`library(vegan)`” it means that the package has not yet been downloaded. You would then need to install it first.

2.2.4.2 More vector functions

As we have seen functions act on the whole vector at once. This applies also to functions that return only one number as a result.

```
x<-c(1,3,5,7,4,2,7)
sum(x)
```

```
## [1] 29
```

```
mean(x)
```

```
## [1] 4.142857
```

```
length(x)
```

```
## [1] 7
```

```
var(x)
```

```
## [1] 5.47619
```

```
sd(x)
```

```
## [1] 2.340126
```

It is very important to realise that a vector with NA values can cause problems for these sorts of functions. You need to tell R explicitly to remove the NAs. If not the result itself will be an NA. For example.

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

```
## [1] 3.5
```

```
x<-c(1,2,3,4,5,NA)
mean(x)
```

```
## [1] NA
```

```
mean(x,na.rm=T)
```

```
## [1] 3
```

This is a very common pitfall for beginners. In general, if something does not work as you expect look for NAs!

2.2.5 Generating sequences of numbers in R

One of the most useful features of R is the ease with which you can generate sequences of numbers and simulated data sets. To produce a sequence you can use the `:` syntax.

```
x<-0:100
0:100
```

```
## [1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
## [18] 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
## [35] 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## [52] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
## [69] 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
## [86] 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

```
x<-30:10
x
```

```
## [1] 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10
```

A longer but more flexible way of producing a sequence is with `seq`. For example to produce even numbers between 0 and 100.

```
x<-seq(0,100,by=2)
x
```

```
## [1] 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
## [18] 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66
## [35] 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100
```

Say we know the length of the sequence we want but have not worked out the intervals.

```
x<-seq(0,100,length=23)
x
```

```
## [1] 0.000000 4.545455 9.090909 13.636364 18.181818 22.727273
## [7] 27.272727 31.818182 36.363636 40.909091 45.454545 50.000000
## [13] 54.545455 59.090909 63.636364 68.181818 72.727273 77.272727
## [19] 81.818182 86.363636 90.909091 95.454545 100.000000
```

2.2.5.1 Using `rep` to replicate a vector

If we want ten copies of the same vector one after the other we can use.

```
x<-rep(c(1,4,9,23),times=10)
x
```

```
## [1] 1 4 9 23 1 4 9 23 1 4 9 23 1 4 9 23 1 4 9 23 1 4 9
## [24] 23 1 4 9 23 1 4 9 23 1 4 9 23 1 4 9 23
```

However we might want each number in the vector replicated ten times before moving to the next. In this case we use `each` instead of `times`.

```
x<-rep(c(1,4,9,23),each=10)
x
```

```
## [1] 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 9 9 9
## [24] 9 9 9 9 9 9 9 9 23 23 23 23 23 23 23 23 23
```

2.2.5.2 Replicating text

When designing a format to hold the results from a planned experiment or field survey it can be very useful to generate replicated sequences of text for the factor levels or grouping variables. This is also very easy.

```
x<-rep(c("Control", "Treatment1", "Treatment2"), each=10)
x

## [1] "Control" "Control" "Control" "Control" "Control"
## [6] "Control" "Control" "Control" "Control" "Control"
## [11] "Treatment1" "Treatment1" "Treatment1" "Treatment1" "Treatment1"
## [16] "Treatment1" "Treatment1" "Treatment1" "Treatment1" "Treatment1"
## [21] "Treatment2" "Treatment2" "Treatment2" "Treatment2" "Treatment2"
## [26] "Treatment2" "Treatment2" "Treatment2" "Treatment2" "Treatment2"
```

Or of course

```
x<-rep(c("Control", "Treatment1", "Treatment2"), times=10)
x

## [1] "Control" "Treatment1" "Treatment2" "Control" "Treatment1"
## [6] "Treatment2" "Control" "Treatment1" "Treatment2" "Control"
## [11] "Treatment1" "Treatment2" "Control" "Treatment1" "Treatment2"
## [16] "Control" "Treatment1" "Treatment2" "Control" "Treatment1"
## [21] "Treatment2" "Control" "Treatment1" "Treatment2" "Control"
## [26] "Treatment1" "Treatment2" "Control" "Treatment1" "Treatment2"
```

2.2.6 Logical vectors and subsetting

One of the keys to using R efficiently is the concept of logical vectors, indices and subsetting. However the concept does take a little effort to get used to. Let's take it a step at a time. Say we have a vector x which we have setup like this.

```
x<-seq(-4, 10, by=2)
x<-rep(x, times=3)
x

## [1] -4 -2 0 2 4 6 8 10 -4 -2 0 2 4 6 8 10 -4 -2 0 2 4 6 8
## [24] 10
```

Now we can ask a question that will be answered as true or false for each of the numbers. Is the element of x greater than zero?

```
x>0

## [1] FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
## [12] TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE TRUE TRUE TRUE
## [23] TRUE TRUE
```

R replies with a vector stating whether the result is true or false. We can also ask which of the numbers is greater than zero.

```
which(x>0)

## [1] 4 5 6 7 8 12 13 14 15 16 20 21 22 23 24
```

Now R replies with the indices of the elements of the vector.

Subsetting the vector involves the use of the square brackets `{}` and `[]`. If you include either a logical vector with TRUE and FALSE values or numeric indices within the square brackets R will subset the vector.

Either way works.

```
x[x>0]
## [1] 2 4 6 8 10 2 4 6 8 10 2 4 6 8 10
x[which(x>0)]
```

```
## [1] 2 4 6 8 10 2 4 6 8 10 2 4 6 8 10
```

When we move on to handling more than one variable at a time using data frames we will see that the same concept can be used to subset whole blocks of data. It is a very powerful and fundamentally simple way of manipulating data.

A more complex example is given here. This takes every second member of `x` using a sequence of indices as a subset.

```
x[seq(2,length(x),by=2)]
```

```
## [1] -2 2 6 10 -2 2 6 10 -2 2 6 10
```

Could you follow how this worked? Working outwards

- `length(x)` gives the number of elements in `x`. Lets call this `n`
- `seq(0,length(x),by=2)` gives the vector of indices 2,4,6,...n.
- A subset of `x` is found using the square brackets.

With experience it is common to wrap up several steps in a single line. There is nothing wrong with explicitly writing

```
n<-length(x)
ind<-seq(2,n,by=2)
x[ind]
```

```
## [1] -2 2 6 10 -2 2 6 10 -2 2 6 10
```

2.2.6.1 Sorting and ordering

We can also sort vectors. There are two ways of doing this. The first is very direct, but I really do not recommend it. It uses the `sort` command with the argument `decreasing=TRUE` or `FALSE`

```
sort(x,decreasing=T)
## [1] 10 10 10 8 8 8 6 6 6 4 4 4 2 2 2 0 0 0 -2 -2 -2 -4 -4
## [24] -4
```

That's simple enough. However it is much better practice in the long run to use `order`. This needs a little explaining, again we will take it step by step.

```
order(x,decreasing=T)
## [1] 8 16 24 7 15 23 6 14 22 5 13 21 4 12 20 3 11 19 2 10 18 1 9
## [24] 17
```

`Order` has not given us the numbers in order! But of course it should not, as `sort` does that. Instead `order` has given us the the indices in order. Notice that if there are ties, as in this case, R respects the original order for the tied numbers. So how can we use `order` to sort the vector? If you have followed the logic of using indices to refer to elements of a vector you might have guessed.

```
x[order(x,decreasing=T)]
```

```
## [1] 10 10 10 8 8 8 6 6 6 4 4 4 2 2 2 0 0 0 -2 -2 -2 -4 -4
## [24] -4
```

Although this involves more typing than simply writing `sort`, it is more powerful. The power comes from the way indices can be used with many variables at once. When we move on to see data frames this will be clearer. For the moment look at this simple example to see the logic.

```
x<-c(1,3,2,4)
y<-c("A","B","C","D")
y[order(x,decreasing=F)]
```

```
## [1] "A" "C" "B" "D"
```

2.2.6.2 Ranks

Finding ranks is a very common procedure in statistics. This is a slightly different problem. We need a way to deal with ties, and there is more than one solution to the problem.

```
x<-c(1,1,2,3,4,5,5)
rank(x,ties="average")
```

```
## [1] 1.5 1.5 3.0 4.0 5.0 6.5 6.5
```

```
rank(x,ties="max")
```

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

```
rank(x,ties="min")
```

```
## [1] 1 1 3 4 5 6 6
```

The last example coincides with the familiar ranks given in sports (joint gold medal followed by the bronze). Notice that there is no decreasing argument to `rank`. The lowest numbers take the lowest ranks. If you really want to rank performance scores you must reverse them first by, say, subtracting all the scores from the maximum possible.

```
x<-c(1,1,2,3,4,5,5)
rank(max(x)-x,ties="min")
```

```
## [1] 6 6 5 4 3 1 1
```

2.3 Exercise

In order to practice putting together these commands in order to simulate some data try this.

A researcher is interested in looking at the differences in diameters of trees in three different woodland sites in the New Forest. At each site there are several different species. In order to simplify the task we will consider only two types of trees ... conifers and broadleaves. We will also simplify the exercise by assuming that the same number of trees (50) are sampled in each woodland.

Set up a dataframe with three columns. One column represents the site. The second represents the type of tree (i.e. conifer or broadleaf). The third represents the diameters. So there will be 150 observations (rows) in all. Try to produce data in which there is a difference in mean diameter that is affected by the site from which the measurements are taken and the type of tree being measured. You can assume that the random variation in diameters is normally distributed and that measurements are taken to the nearest cm.

Bibliography

- Gigerenzer, G. (1998). Surrogates for theories. *THEORY & PSYCHOLOGY*, 8(2):195–204.
- Goodman, S. (2008). A dirty dozen: Twelve p-value misconceptions. *Seminars in Hematology*, 45(3):135–140.
- Hobbs, N. T. and Hilborn, R. (2006). Alternatives to statistical hypothesis testing in ecology: A guide to self teaching. *ECOLOGICAL APPLICATIONS*, 16(1):5–19.
- Johnson, D. H. (1999). The insignificance of statistical significance testing. *The Journal of Wildlife Management*, 63(3):763.
- Nickerson, R. S., Baron, J., Chechile, R., and Es, W. (2000). Null hypothesis significance testing : A review of an old and continuing controversy. 5(2):241–301.
- Steel, E. A., Kennedy, M. C., Cunningham, P. G., and Stanovick, J. S. (2013). Applied statistics in ecology: Common pitfalls and simple solutions. *Ecosphere*, 4(9):1–13.
- Zuur, A. F., Ieno, E. N., and Elphick, C. S. (2010). A protocol for data exploration to avoid common statistical problems. *Methods in Ecology and Evolution*, 1(1):3–14.