Stats with Sparrows - 01

Julia Schroeder

July 2020

Introduction to Statistics with Sparrows and Welcome!

Learning aims

Welcome to Stats with Sparrows. Make sure you attend the welcome session, or watch the recorded version, to get up to speed to what this week will be about. The learning aims of this week are to: - Compute and interpret descriptive statistics - Compute and interpret parametric tests, in particular t-test and linear models - Understand how to interpret fixed effect interactions - Get an understanding for how to match your hypothesis with an appropriate model - Know how to report methods and results in a scientific paper

If you read through this, you might have noticed that the focus is on interpreting - not mathematics. As you are, or aspire to become, biologists, it is of utmost importance to know how to choose, and interpret statistics that are fitting to your biological question. Oftentimes, either people do not understand the statistics and simply report back numbers without ever having engaged with the values, what they really mean. On the other end of the spectrum, some people do statistics because they like the complexity, and the math, but then do not link the results back to the biology. Both approaches are not right - you need to know what statistic to use for what question, why, and how to interpret the result. You will be surprised to learn that you do not need to remember, or know, the math behind it to be a good biologist.

Following this philosophy, this week will be on applied statistics, and sometimes we will brush over the mathematical details. If you are interested in more, the recommended reading for this week will get you on the way there. Other times, however, it is helpful to know the mathematical logic behind a certain test. In that case, I will try to break it down as simple as possible. Thus, in short, this course aims at providing you with the tools, knowledge, and knowledge where to ask for help, to use the appropriate statistics for your biological question.

How to work through this course

The course is based on lectures, hand-out practicals (HOs), and group learning. Watch the video before you embark on the HO, - some information is only in one but not the other, - to graduate from the course you need both. Some videos have no HO associated. Some, but not all, HOs have exercises that you are supposed to work on. These might require you to write a short methods or results section, or to write and annotate an R script, or to prepare a presentation, either every student by themselves, or in a group. Most exercises also ask for you to discuss things in your group. Many of these exercises are like model exam questions and are really good practice. Many exercises require you to upload your results and compare it with the work of your peers, present your work, mark your peers work, discuss different solutions. You will have a group already - please use these groups for group-based work. If you are not assigned to a group, please email me.

Hand-outs and synchronous help sessions

After watching a video, return to your HO and work through the rest of it. Make sure to create a new R script (.R) for each day, and do not simple copy-paste from this handout but rather try to think about what the code does. There will be projects set up in R Studio cloud for you to work in, and for GTAs and myself to be able to help you. We expect you to write such code by yourself every day.There are in-person, and also remote, synchronous help sessions where help is available from teaching staff in break out rooms on Teams. Make sure you are comfortable with sharing your screen.

Asynchronous help sessions/Q&A chat on Teams

The asynchronous help sessions are in the Q&A chat on teams – you can post your questions here. However, because often code builts on itself, it is imperative that you solve a problem before you can continue, therefore, I suggest you work with your GTA through issues during the synchronous help sessions if you feel your skills require a lot of help.

Here is a list of things you can do in order to solve problems. Please go through this list before posting your problem on the Q&A chat:

1) Most errors are simply typos, and sometimes they are hard to find. To solve it, try to re-type it and this time pay good attention. Sometimes re-typing a line makes it easier to find the typo.
2) When something does not work, run

```
rm(list=ls())
```

at least once. Often we forget cleaning the workspace before a session. Sometimes we've done this but have loaded up variables with the same name and that is what messes things up. Even if you have no clue what could be in the global workspace, do clean your environment when you run across an error.

3) Check your objects. The variable you are trying to change, or run a model with. Look at the variable with various functions - dir(), summary(), table(), ect. Sometimes we don't even notice that we are handling, e.g. an empty vector. Double and triple check all data involved in the code that doesn't work.
4) Google the function. This is the most important advice we can give you. There are tons of information and help available online. It is common procedure for anyone using R to regularly google functions one has never used before, or forgot how to use.
5) If you use complex code with more than one operation in one go - break it down into smaller parts, and run each separate part as stand-alone. (
6) For running algorithms using multiple iterations, loops ect: Use a break point or print to print out the current value.

Only when the above do not apply, ask for help in one of our Q&A sessions. And then provide information to the script that you are asking about and what it is named, so I can find it on R Studio Cloud.

Linearity

The HOs built on each other. Even if you think you know a topic, I would advise you to work through the respective HO - you can test your knowledge using the exercise section. HOs and tasks also give you good guidelines for how to write the statistics part of your thesis, how to choose what data to collect and exam practise.

BYO

At the end of this course, we might have a BYO session. This is typically where students send me data they have worked on in the past, e.g. for their undergrad degree, and that they would like to see re-analysed. We will use these data and distribute it among groups, with the biological question, and the groups then will find different ways of analyzing it. I will also provide a solution. If you think you have appropriate data that you do not mind sharing, please email me until latest Wednesday noon. In the past, students found these BYO sessions particularly helpful, so please come forward – if nobody comes forward we cannot do these.

Revisiting what R can do for you

In this practical, you will practice using the R command line interface to do some basic stats, in particular, look at the data structure. First, we revisit what you learned last week to make sure you can use these skills.

A lot of this is revisiting last week. Start up R or RStudio, or whatever GUI you wish. Remember, you can download R from the Comprehensive R Archive Network (http://cran.r-project.org/).

So let's start now!

Some good house-keeping: Cleaning your environment!

```r
rm(list=ls())
```

Checking and re-setting your working directory:

```r
getwd()
setwd("/cloud/project/SwS01Workspace")
getwd()
```

Remember to always comment on your code:

```r
getwd()                       # check which working directory we're in
setwd("/cloud/project/SwS01Workspace")# re-set workign directory to
getwd()                       # check that what we did worked
```

Remember you can use basic calculator commands in R:

```
2*2+1

## [1] 5

2*(2+1)

## [1] 6

12/2^3

## [1] 1.5

(12/2)^3

## [1] 216
```

Remember you can assign values to variables:

```
x <- 5
x

## [1] 5

y <- 2
y

## [1] 2

x2 <- x^2
x2

## [1] 25

x

## [1] 5

a<-x2+x
a

## [1] 30

y2 <- y^2
z2 <- x2 + y2
z <- sqrt(z2)
print(z)

## [1] 5.385165
```

There are logical tests:

```
3>2
```

```
## [1] TRUE

3 >= 3

## [1] TRUE

4<2

## [1] FALSE
```

You can create vectors of different formats:

```
myNumericVector <- c(1.3,2.5,1.9,3.4,5.6,1.4,3.1,2.9)
myCharacterVector <- c("low","low","low","low","high","high","high","high")
myLogicalVector <- c(TRUE,TRUE,FALSE,FALSE,TRUE,TRUE,FALSE,FALSE)
```

We will use the str() function to get the structure of any variable.

```
str(myNumericVector)

##  num [1:8] 1.3 2.5 1.9 3.4 5.6 1.4 3.1 2.9

str(myCharacterVector)

##  chr [1:8] "low" "low" "low" "low" "high" "high" "high" "high"

str(myLogicalVector)

##  logi [1:8] TRUE TRUE FALSE FALSE TRUE TRUE ...
```

It is important for statistical reasons, to always know what kind of data your variable is - numeric, vector, ect. If you add values of different categories to one variable, you will create a mixed variable and R will coerce (force) it into one of the most basic mode that covers all:

```
myMixedVector <-c(1, TRUE, FALSE, 3, "help", 1.2, TRUE, "notwhatIplanned")
str(myMixedVector)

##  chr [1:8] "1" "TRUE" "FALSE" "3" "help" "1.2" "TRUE" "notwhatIplanned"
```

We install packages, and load them. Can you find out what the difference is between the library and the require command?

```
install.packages("lme4")
library(lme4)
require(lme4)
```

When we are lost, we ask R for help:

```
help(getwd)

help(log)
```

There are some special notation in R.

```
sqrt(4); 4^0.5; log(0); log(1); log(10); log(Inf)

## [1] 2

## [1] 2

## [1] -Inf

## [1] 0

## [1] 2.302585

## [1] Inf
```

Inf means infinity. You can specify if it's negative or positive. In R, *e* is typed in using an R function exp and we should type in exp(1). For π, just type in pi.

```
exp(1)

## [1] 2.718282

pi

## [1] 3.141593
```

R shows up to 6 decimal places, the default is to show 7 digits. To what precision should you report decimal places in your reports?

Clear your workspace at the beginning of each session, and between different parts of a project.

```
rm(list=ls())
```

Entering data

We can enter data in many ways, and you will have learned last week how to do that. Here, we will mostly simply read tables with headers:

```
d<-read.table("SparrowSize.txt", header=TRUE)
str(d)

## what you see below might differ from what you will see in your r console –
## it differs by R version. Don't be distracted by these.

## 'data.frame':    1770 obs. of  8 variables:
##  $ BirdID: int  1 2 2 2 2 2 2 2 2 2 ...
##  $ Year  : int  2002 2001 2002 2003 2004 2004 2004 2004 2004 2005 ...
##  $ Tarsus: num  16.9 16.8 17.2 17.5 17.8 ...
##  $ Bill  : num  NA NA NA 13.5 13.4 ...
##  $ Wing  : num  76 76 76 76 77 78 77 77 77 77 ...
##  $ Mass  : num  23.6 27.5 28.1 27.8 26.5 ...
##  $ Sex   : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ Sex.1 : chr  "female" "male" "male" "male" ...
```

```
head(d)

##   BirdID Year Tarsus Bill Wing  Mass Sex  Sex.1
## 1      1 2002   16.9   NA   76 23.60   0 female
## 2      2 2001   16.8   NA   76 27.50   1   male
## 3      2 2002   17.2   NA   76 28.10   1   male
## 4      2 2003   17.5 13.5   76 27.75   1   male
## 5      2 2004   17.8 13.4   77 26.50   1   male
## 6      2 2004   17.7 13.1   78 26.00   1   male

summary(d)

##      BirdID          Year          Tarsus          Bill           Wing
##  Min.   :  1.0   Min.   :2000   Min.   :15.00   Min.   : 9.7   Min.
:60.0
##  1st Qu.:117.0   1st Qu.:2003   1st Qu.:18.00   1st Qu.:13.0   1st
Qu.:76.0
##  Median :259.5   Median :2004   Median :18.60   Median :13.3   Median
:77.0
##  Mean   :290.3   Mean   :2004   Mean   :18.52   Mean   :13.3   Mean
:77.4
##  3rd Qu.:477.0   3rd Qu.:2005   3rd Qu.:19.10   3rd Qu.:13.7   3rd
Qu.:79.0
##  Max.   :636.0   Max.   :2010   Max.   :21.10   Max.   :16.0   Max.
:84.0
##                                 NA's   :85      NA's   :630    NA's   :75
##       Mass            Sex           Sex.1
##  Min.   :18.60   Min.   :0.000   Length:1770
##  1st Qu.:26.40   1st Qu.:0.000   Class :character
##  Median :27.60   Median :1.000   Mode  :character
##  Mean   :27.76   Mean   :0.513
##  3rd Qu.:29.10   3rd Qu.:1.000
##  Max.   :36.20   Max.   :1.000
##  NA's   :66
```

We can also look at the data structure. Data structure is incredibly important, but people often ignore it. The way data is structured will determine how you can analyse it, so you need to know what your data structure is. The first thing you do is find out what is the identifying feature of a single row entry. With identifying feature I mean how is a single row differentiated from other rows. Let’s have a look at the sparrow dataset:

```
head(d)

##   BirdID Year Tarsus Bill Wing  Mass Sex  Sex.1
## 1      1 2002   16.9   NA   76 23.60   0 female
## 2      2 2001   16.8   NA   76 27.50   1   male
## 3      2 2002   17.2   NA   76 28.10   1   male
## 4      2 2003   17.5 13.5   76 27.75   1   male
## 5      2 2004   17.8 13.4   77 26.50   1   male
## 6      2 2004   17.7 13.1   78 26.00   1   male
```

There is a BirdID number, but it is not unique. The same bird can be caught more than once, e.g. bird 2 has been caught in multiple years. It even has been caught twice in 2004. The measurements for Tarsus, Bill, Wing, Mass however differ slightly, even between the recaptures within years. From this we can conclude that our data structure is repeated measures of birds within years. Now I want to know, how many years, and how many bird capture per year?

```
table(d$Year)

## 
## 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 
##  236   98   66  177  468  458  123  107   11   23    3
```

This gives us a nice overview of how many capture per year. Now we want to know how many captures per bird. The table() is a bit difficult to yield here:

```
table(d$BirdID)
```

We really want to summarize this. Now we can see how often each bird is captured, but we really want to go one level up - how many birds have been captured once, twice, three times ect. ? It's a bit like inception. We want to do a table of a table:

```
table(table(d$BirdID))

## 
##   1   2   3   4   5   6   7   8   9  10  11  12 
## 225 147  98  49  45  28  15  12   8   6   1   2
```

There are two birds that have been caught 12 times!

There are other ways of doing this - you can also use the dplyr version:

```
require(dplyr)

## Loading required package: dplyr

## 
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
## 
##     filter, lag

## The following objects are masked from 'package:base':
## 
##     intersect, setdiff, setequal, union

BirdIDCount <- d %>% count(BirdID,BirdID, sort=TRUE)
BirdIDCount %>% count(n)

## # A tibble: 12 x 2
##       n    nn
##   <int> <int>
##  1     1   225
##  2     2   147
##  3     3    98
##  4     4    49
##  5     5    45
##  6     6    28
##  7     7    15
##  8     8    12
##  9     9     8
## 10    10     6
## 11    11     1
## 12    12     2
```

When people talk about sample size, it is often not simply the total number of observations. While the total number is crucial and needs to be given, it is equally crucial to give sample sizes of sub-group levels - individuals per treatment group, observations per national park, per year, per family.

Excercises:

1. How many repeats are there per bird per year?

2. How many individuals did we capture per year for each sex? Compute the numbers, devise a useful table format, and fill it in.

3. Think about how you can communicate (1) and (2) best in tables, and how you can visualise (1) and (2) using plots. Produce several solutions, and discuss in the group which the pros and cons for each solution to communicate and visualize the data structure for (1) and (2).

4. Write and submit two results sections for (1) and (2). Each result section should use different means of communicating the results, visually and in a table. Submit two word documents per group.

Statistics with Sparrows - 02

Julia Schroeder

July 2020

Learning aims

Understand how we quantify distributions

Encounter z-scores and z-distribution

Describing distributions

Let's get down to the hard stuff: statistics! But first, watch the video SwS.v.02 Afterwards, we'll get our hands dirty and do the things we've seen in the video - and, of course stumble across all the hurdles that come our way when using R and heterogenous ecological data!

We will use the sparrow data as example. But before we begin, we clear our workspace. Never forget!

```
rm(list=ls())

setwd("/cloud/project/SwS02")

d<-read.table("SparrowSize.txt", header=TRUE)
str(d)

## what you see below might differ from what you will see in your r console –
## it differs by R version. Don't be distracted by these.

## 'data.frame':    1770 obs. of  8 variables:
##  $ BirdID: int  1 2 2 2 2 2 2 2 2 2 ...
##  $ Year  : int  2002 2001 2002 2003 2004 2004 2004 2004 2004 2005 ...
##  $ Tarsus: num  16.9 16.8 17.2 17.5 17.8 ...
##  $ Bill  : num  NA NA NA 13.5 13.4 ...
##  $ Wing  : num  76 76 76 76 77 78 77 77 77 77 ...
##  $ Mass  : num  23.6 27.5 28.1 27.8 26.5 ...
##  $ Sex   : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ Sex.1 : chr  "female" "male" "male" "male" ...

names(d)

## [1] "BirdID" "Year"   "Tarsus" "Bill"   "Wing"   "Mass"   "Sex"    "Sex.1"

head(d)

##   BirdID Year Tarsus Bill Wing  Mass Sex  Sex.1
## 1      1 2002   16.9   NA   76 23.60   0 female
## 2       2 2001   16.8   NA   76 27.50   1   male
## 3       2 2002   17.2   NA   76 28.10   1   male
## 4       2 2003   17.5 13.5   76 27.75   1   male
## 5       2 2004   17.8 13.4   77 26.50   1   male
## 6       2 2004   17.7 13.1   78 26.00   1   male
```

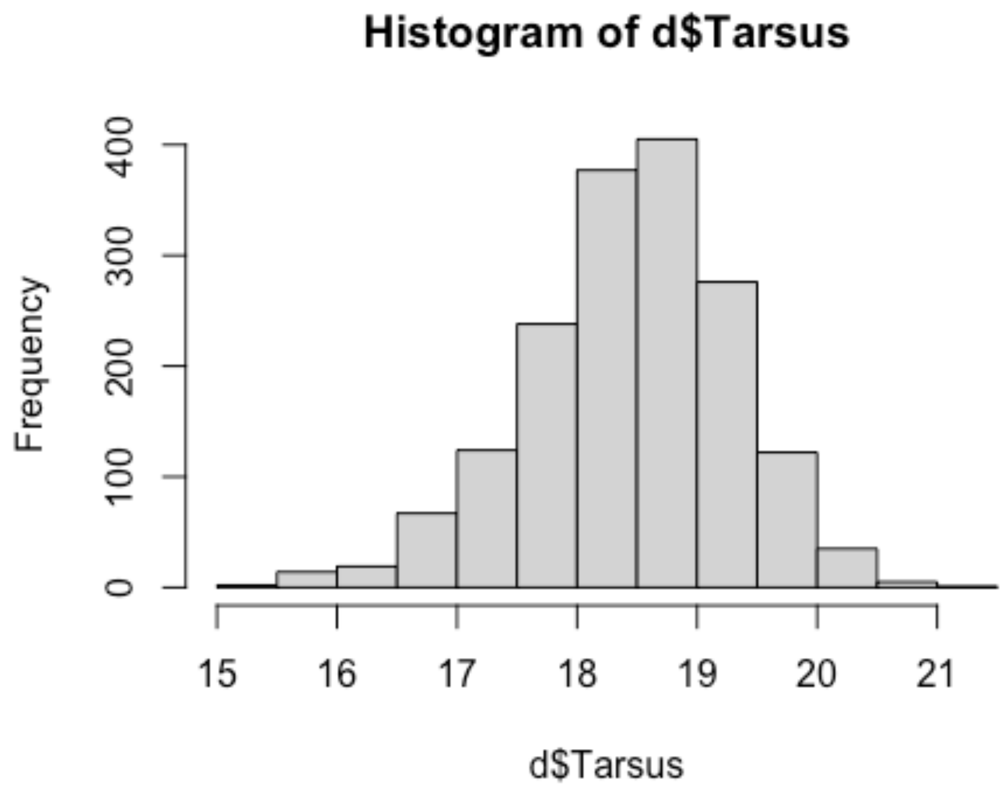
There are a few NA's - and we will deal with them soon. We already know the data structure. Now let's check the distribution of the data. We usually do this with a histogram. For now, we will work with data on the length of a bird's tarsus, that's the leg of a bird. We have a lot of data to deal with here!

```
length(d$Tarsus)

## [1] 1770
```

That's a good sample size! So let's get cracking!

Histograms, mean, median and mode

```
hist(d$Tarsus)
```



This looks like a normal distribution, doesn't it? It might lean a bit to the right, though. What is a normal distribution? Many data that we collect will be expected to approximately follow a normal distribution. Therefore, many statistics, - more explicitely, parametric statistics, rely on the data being normally distributed. So it's a good idea to google them that they do actually look somewhat normal. However, the actual expectation of many tests is that not the data themselves, but the residuals are normally distributed. Confused much? No need to be - it's less important as it sounds, and there's a few ways around it. However it's a good idea to think ahead of designing your experiment about this - using some 5-point scale (ice cold, cold, meh, warm, hot) might be less of a good idea than acutally measuring it in a unit on a continous scale (like, degree Celsius).

Now, how can we describe this distribution of values best? Usually, we use a description of centrality, and one of the spread. What is centrality?

Centrality, mean, median and mode in normally distributed data

```
mean(d$Tarsus)

## [1] NA
```

Uuups. We have missing values in our dataset (NAs). This is reflected by the error message. How to deal with this? A quick look up in help reveals that we can use an argument na.rm that is by default set to "FALSE". When set to TRUE, it means that NAs are stripped before computation. That's what we want. Let's give it a try:

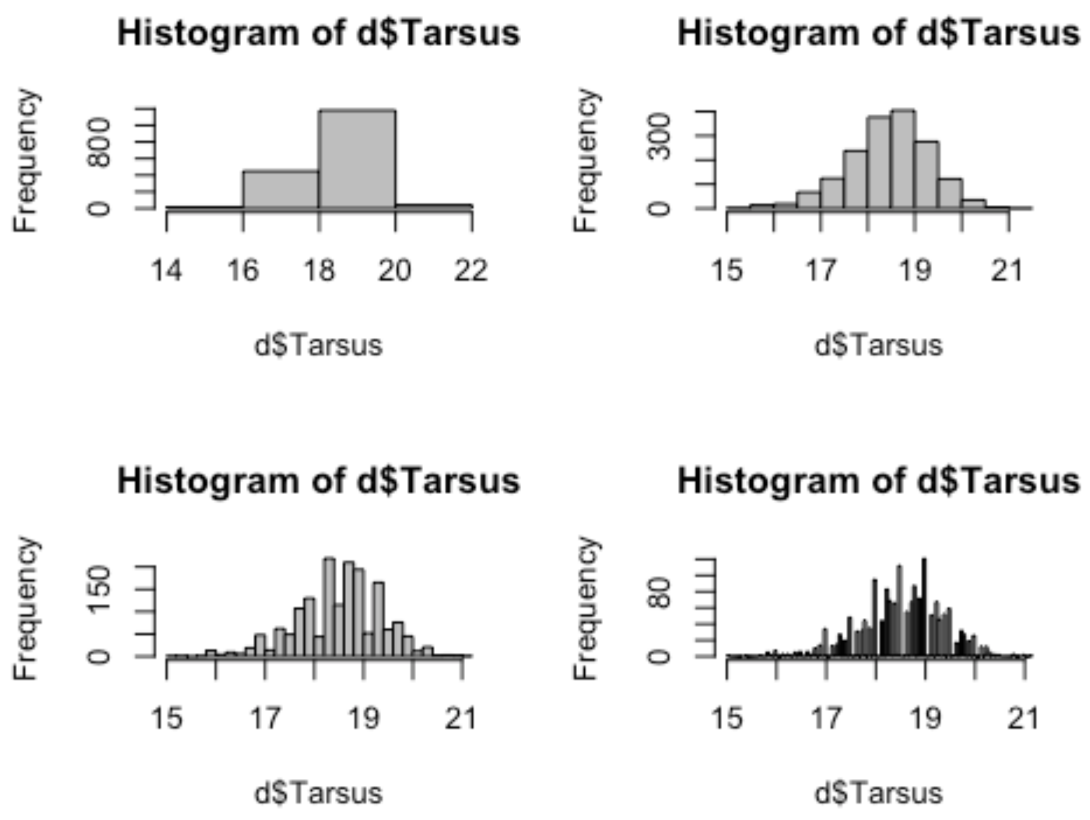
```
help(mean)
mean(d$Tarsus, na.rm = TRUE)

## [1] 18.52335

median(d$Tarsus, na.rm = TRUE)

## [1] 18.6

mode(d$Tarsus)

## [1] "numeric"
```

Now, this worked at least two of three times. What's with the odd result for mode? The mode function returns a description of the type of object. It tells us that d$Tarsus is a numerical vector. What does this mean? If we have continous data, we have a hard time estimating the mode. That is because the mode is the most frequently occuring value. Why do you think it is hard to estimate it in a continous dataset?

Because most values occur only once.

Let's play with the data a bit more and look how the distribution changes, and the (supposed) mode. Also, can you remember what par(mfrow=c(2,2)) does?

```
par(mfrow = c(2, 2))
hist(d$Tarsus, breaks = 3, col="grey")
hist(d$Tarsus, breaks = 10, col="grey")
hist(d$Tarsus, breaks = 30, col="grey")
hist(d$Tarsus, breaks = 100, col="grey")
```



Have a think:

Think about this. Why do you think there are these odd gaps?

The number of breaks determines the number of bins that is used to draw the histogram. And at some point, the resolution is larger than the resolution of the measuring device! Clearly the mode is somewhere between 18 and 19. It would be nice to get this more precisely. However, the is currently no package that calculates the mode. So we have to do it ourselves. The mode is the value that occurs most frequently in the data. Let's see - first, we need to count how often each value occurs. Seems familiar from the data strucutre session? It is:

```
head(table(d$Tarsus))

##
##          15 15.39999962 15.69999981 15.80000019 15.89999962          16
##           1           1           1           5           1           7
```

That really long table! See what I did there with head()?

Some values have been measured 7 times, others only once. Ok. Maybe it would be reasonable to round the values first, because there's clearly some odd issue at play - nobody can measure a tarsus that precisely. We'll round to one decimal.

```
?round
```

That’s what we need. Here we can see how to code it so we get 1 decimal:

```
d$Tarsus.rounded<-round(d$Tarsus, digits=1)
head(d$Tarsus.rounded)

## [1] 16.9 16.8 17.2 17.5 17.8 17.7
```

Better. But now we need to find out which one is the highest. Didn’t we sort earlier with dplyr?

```
require(dplyr)

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

TarsusTally <- d %>% count(Tarsus.rounded, sort=TRUE)
TarsusTally

## # A tibble: 55 x 2
##    Tarsus.rounded     n
##             <dbl> <int>
##  1           19     121
##  2           18.5   112
##  3           18      95
##  4           18.8    87
##  5           NA      85
##  6           18.2    83
##  7           18.9    73
##  8           18.3    70
##  9           18.4    68
## 10           18.7    68
## # … with 45 more rows
```

This works- ish. The top number (19) is the mode. But we can also see that in the 5th row, there’s a “NA” - that’s a rather frequent value. So in some datasets that might show up as the most frequent value. Better to remove the NAs first.

```
d2<-subset(d, d$Tarsus!="NA")
length(d$Tarsus)-length(d2$Tarsus)

## [1] 85
```

Fits - the difference in length between both datasets is 85- the tally for NA is the same, so this is correct.

```
TarsusTally <- d2 %>% count(Tarsus.rounded, sort=TRUE)
TarsusTally

## # A tibble: 54 x 2
##    Tarsus.rounded     n
##             <dbl> <int>
##  1           19     121
##  2           18.5   112
##  3           18      95
##  4           18.8    87
##  5           18.2    83
##  6           18.9    73
##  7           18.3    70
##  8           18.4    68
##  9           18.7    68
## 10           19.2    67
## # … with 44 more rows
```

Now all we need is to extract the first value of the first column. The result of anything dplyr is usually a tibble, and that's accessed with square brackets [] and numbers. I don't know exactly how (maybe Josh knows by heart) so I will have to play around a bit until I find the right solution:

```
TarsusTally[1]

## # A tibble: 54 x 1
##    Tarsus.rounded
##             <dbl>
##  1           19
##  2           18.5
##  3           18
##  4           18.8
##  5           18.2
##  6           18.9
##  7           18.3
##  8           18.4
##  9           18.7
## 10           19.2
## # … with 44 more rows
```

This extracts the first column

```
TarsusTally[2]

## # A tibble: 54 x 1
##        n
##    <int>
##  1   121
##  2   112
##  3    95
##  4    87
##  5    83
##  6    73
##  7    70
##  8    68
##  9    68
## 10    67
## # … with 44 more rows
```

And this the second (not so surprisingly).

```
TarsusTally[[1]]

##  [1] 19.0 18.5 18.0 18.8 18.2 18.9 18.3 18.4 18.7 19.2 19.5 18.6 19.1 19.4
17.5
## [16] 19.3 17.8 18.1 17.9 17.0 17.6 17.7 19.7 17.3 19.8 20.0 17.4 19.9 19.6
17.2
## [31] 16.9 17.1 20.1 20.2 16.8 16.0 16.5 16.6 20.3 15.8 16.4 16.2 16.1 16.7
20.4
## [46] 20.8 20.5 21.0 15.0 15.4 15.7 15.9 16.3 21.1
```

This gives us a normal vector (takes away the wrapping and naming of the column). Now I can simply access the first element by adding [1]:

```
TarsusTally[[1]][1]

## [1] 19
```

So, let's try this again:

```
mean(d$Tarsus, na.rm = TRUE)

## [1] 18.52335

median(d$Tarsus, na.rm = TRUE)

## [1] 18.6

TarsusTally[[1]][1]

## [1] 19
```

In normally distributed data, mean, median and mode should be fairly similar. If the distribution is perfectly normal, they should even be identical. As the skew of the distribution increases, these three measures diverge.

Range, variance and standard deviation

R is really good because you can guess the name of certain functions. We can just guess that "range" might give us the range of values in a vector. Either type it into the console (?range) or google it.

```
range(d$Tarsus, na.rm = TRUE)

## [1] 15.0 21.1

range(d2$Tarsus, na.rm = TRUE)

## [1] 15.0 21.1

var(d$Tarsus, na.rm = TRUE)

## [1] 0.7404059

var(d2$Tarsus, na.rm = TRUE)

## [1] 0.7404059
```

Now, range is easy - that's the minimum and maximum values. Now let's look again more closely at the variance. It's often ignored, yet it plays a very central role, and you should embrace it!

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}$$

So, the variance is described as σ^2. It should be apparent, from the video lecture, why it's a square. Let's have a closer look at the formula. It includes the mean, but also other stuff. The numerator is called the sum of squares, and is sometimes denoted by SS. This is super important. Why is it a sum of squares? Let's a have a look at how to get this. It's the sum of the squares of the *deviations* from the mean. I'll write out how to calculate this in R. We use the tarsus bit without NAs as then the length is correct.

```
sum((d2$Tarsus - mean(d2$Tarsus))^2)/(length(d2$Tarsus) - 1)

## [1] 0.7404059
```

Quick check:

```
var(d2$Tarsus)

## [1] 0.7404059
```

Now, why is it denoted as σ^2? If we square-root the variance, we get σ, and that is the standard deviation:

```
sqrt(var(d2$Tarsus))

## [1] 0.8604684
```

```
sqrt(0.74)

## [1] 0.8602325

sd(d2$Tarsus)

## [1] 0.8604684
```

Cool. Now we understand how to describe data that is about normally distributed - with measures of centrality (mean, median, mode) and measures that describe the spread (range, variance, standard deviation).

Z-scores and quantiles

Z-values, and z transformation (z-score, or standardized scores) is a super important topic that you will encounter very often and use very often. Z-scores come from a standardized normal distribution, with a mean of 0 and a standard deviation of 1.

Have a think: *What variance does this distribution have?*

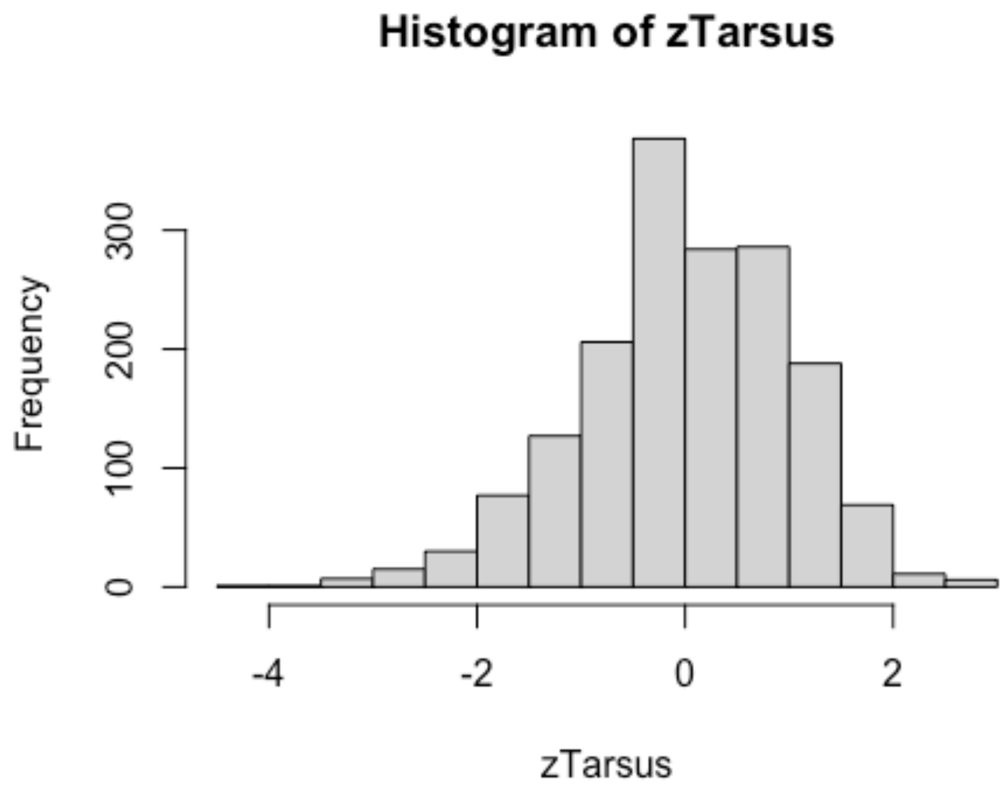
This is not a trick question. It's an easy question for someone with an intuitive understanding of maths. People who don't have that might struggle. The answer is - the variance of z-standardized data with a sd of 1 is also 1. That's because the squareroot of 1 is 1. So, in this special case, the variance equals the standard deviation.

When we do stats it is often useful to transform our data so it follows exactly these rule - z-transforming data. You can do that by dividing the deviation from the mean by the standard deviation:

$$z = \frac{y - \overline{y}}{\sigma_y}$$

Here, σ_y means standard deviation of y. Ok, let's do this, and check is all went according to plan:

```
zTarsus <- (d2$Tarsus - mean(d2$Tarsus))/sd(d2$Tarsus)
var(zTarsus)

## [1] 1

sd(zTarsus)

## [1] 1

hist(zTarsus)
```

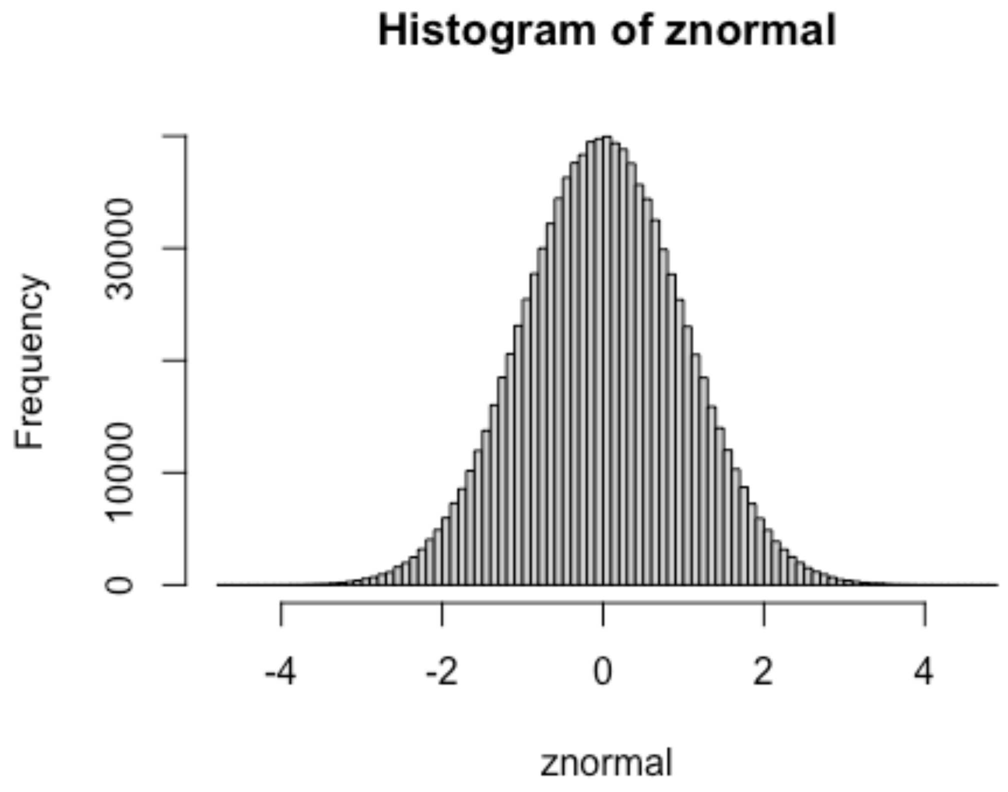


But, of course, there is a function for this in R.

Have a think: *Use Google to find this function, remember it, and use it often! Share the function you found in your group to see if you all found the same one!*

BTW, the reason we call them z-scores is because the normal distribution is also called the z-distribution. The neat thing about R is that it allows you to make us datasets from scratch that follow this distribution:

```r
znormal <- rnorm(1e+06)
hist(znormal, breaks = 100)
```



That's a beautiful normal distribution. Let's examine it a bit closer:

```
summary(znormal)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -4.799191 -0.674387 -0.000260 -0.000521  0.673333  4.850767
```

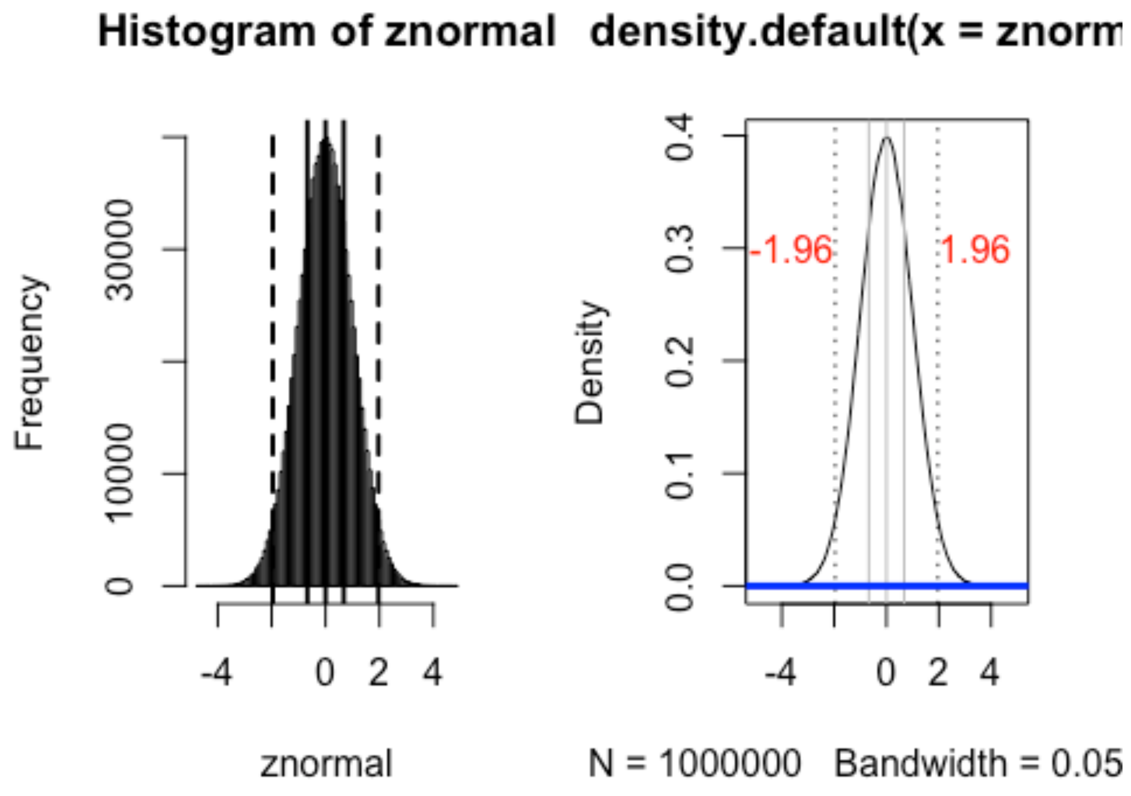
Very helpful. We even get the central tendencies in one go. The quantiles refer to the respective cut-offs of the data distribution, think back to the beginning of this lecture! The median is the second quantile where 50% of data points are included. R can also give us other information on the normal distribution such as the value at a given quantile (qnorm) or the probability at a given value (not data randomly sampled from it, which we did above):

```
qnorm(c(0.025, 0.975))
pnorm(.Last.value)

## [1] -1.959964  1.959964

## [1] 0.025 0.975
```

`qnorm(c(0.025,0.975))` gives us the 2.5% and 97.5% quantiles from the corresponding probability distribution. Both bracket 95% of all values in the distribution. And pnorm gets us the corresponding probabilities.

These last quantiles are important when we get to hypothesis testing, so remember them!

```r
par(mfrow = c(1, 2))
hist(znormal, breaks = 100)
abline(v = qnorm(c(0.25, 0.5, 0.75)), lwd = 2)
abline(v = qnorm(c(0.025, 0.975)), lwd = 2, lty = "dashed")
plot(density(znormal))
abline(v = qnorm(c(0.25, 0.5, 0.75)), col = "gray")
abline(v = qnorm(c(0.025, 0.975)), lty = "dotted", col = "black")
abline(h = 0, lwd = 3, col = "blue")
text(2, 0.3, "1.96", col = "red", adj = 0)
text(-2, 0.3, "-1.96", col = "red", adj = 1)
```



```
## null device
##           1
```

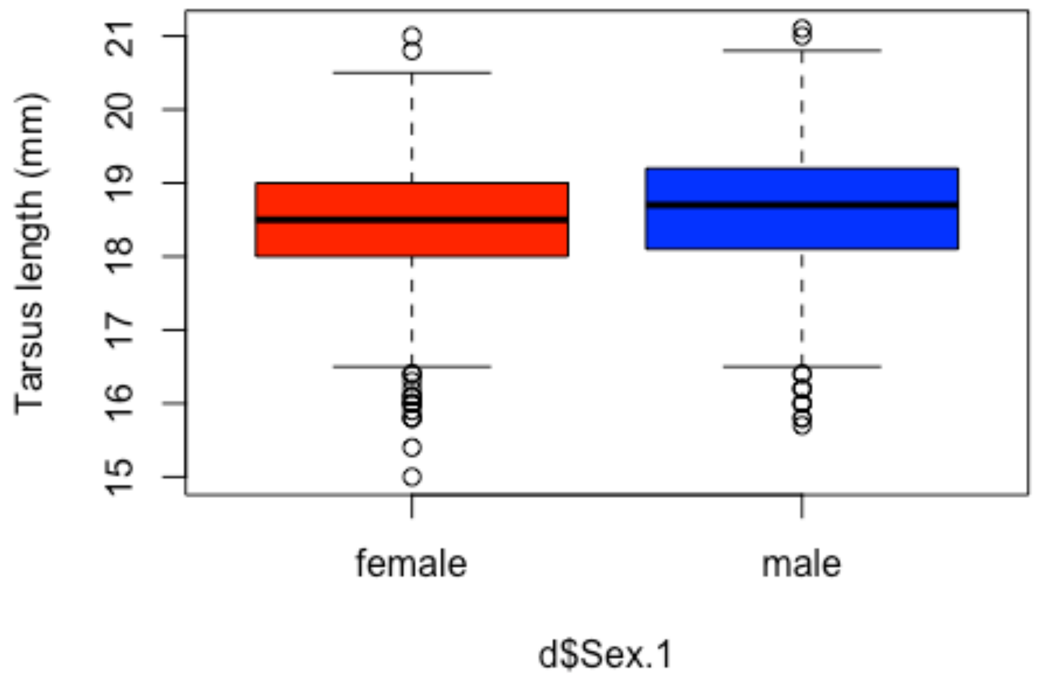
The 95% Confidence Interval (95% CI) is a very important property. It is the range of values the encompass the population true value with 95% probability. That means we will make an error in 5% of the times. Here it means that the true mean of the population that we sampled, and then used the data to calculate the 95CI, lies within this confidence interval 95 times out of 100 sampling events.

Now, let's a have a look at a sparrow example. I'll plot the tarsus length between the two sexes.

Exercises:

(1) What do you think why did I use dSex.1 here and not dSex? Discuss in your group.

```
boxplot(d$Tarsus~d$Sex.1, col = c("red", "blue"), ylab="Tarsus length (mm)")
```



What do you think these boxes and lines represent? You can find that our using.

```
?boxplot
```

Use the help function in R often and frequently. Many of them even detail examples when you scroll to the bottom.

(2) Discuss when a median or mode, might be useful over a mean.

(3) Discuss how the precision of a variable affects which bin size is good for histogramms Z-scores: What is the variance of z-standardized data?

(4) Which R function did you find to compute z-scores? How does it work?

(5) Why did I prefer d$Sex.1over d$Sex? What other solution could have have done to achieve the same goal?

(6) *What is the difference between a distribution with mean = 0, var = 2, and mean = 0, var = 200?*

Statistics with Sparrows - 03

Julia Schroeder

July 2020

HO 03

Data types

Learning aims

- Understand different data types
- Understand why continuous data are the most valuable data
- Understand that non-continuous data has limited value

```r
rm(list=ls())
d<-read.table("SparrowSize.txt", header=T)
str(d)
```

Here we see the difference in data typees - BirdID is an integer, that means there are no decimals. Year, is an integer, too. Tarsus, Bill, Wing, Mass are numerical - that means they are continuous variables. Sex is an integer - here I denoted female with 0, and male with 1. Sex.1 is a character, that means, text. These data types are similar, but not exactly, the data types we're dealing with now.

So, which of these are continous, and what is categorical? The first thing to do is, naturally, have a good think. Clearly `Sex.1` is categorical, there is no question about that. And, while a number, `Sex` is, too. In `Sex`, females are denominated as 0, and males as 1, but that still does not mean males are somewhat *more* than females. As such, the 0 1 coding for sex is a hack, too, and we need to keep that into account when we look at results for this. Make a mental note!

Clearly, `Tarsus`, `Bill`, `Wing`, `Mass` are numerical - these are continuous measurements of length or mass, in milimeter and gram.

`BirdID` is a number, an integer. That means there's no decimals. There can also be no decimals, because between bird 1 and bird 2, there's no option for a bird 1.5. Also, the differences between these numbers are meaningless - we cannot substract bird 3245 from bird 12. In that sense, the `BirdID`s are really more equivalent to names, or nick-names, for individual birds. We use numbers because that's easiest in our database - we can automate it - but really, these should be a proper non-numerical categorical variable. We can either keep that in mind (dangerous, because we might forget, pass the varible on to some statistical analysis, and R will interpret it as continous, because it's denoted as integer… so unless we can be 100% sure we'll never use it as a fixed covariate or so, we better tell R it's a categorical factor:

```
d$BirdIDFact<-as.factor(d$BirdID)
str(d$BirdIDFact)

##  Factor w/ 636 levels "1","2","3","4",..: 1 2 2 2 2 2 2 2 2 2 ...
```

And now, looking at the structure, we can see the difference this line makes - `BirdID` is an integer, and our new variable, `BirdIDFact`, is a factor, with 636 levels (that means, individual birds in this case), that are called names, and the names are numbers. R indicates that with "" around the number - these aren't interpreted as values, but rather, as text. We can check this by attempting to compute the mean of each:

```
mean(d$BirdID)

## [1] 290.2983
```

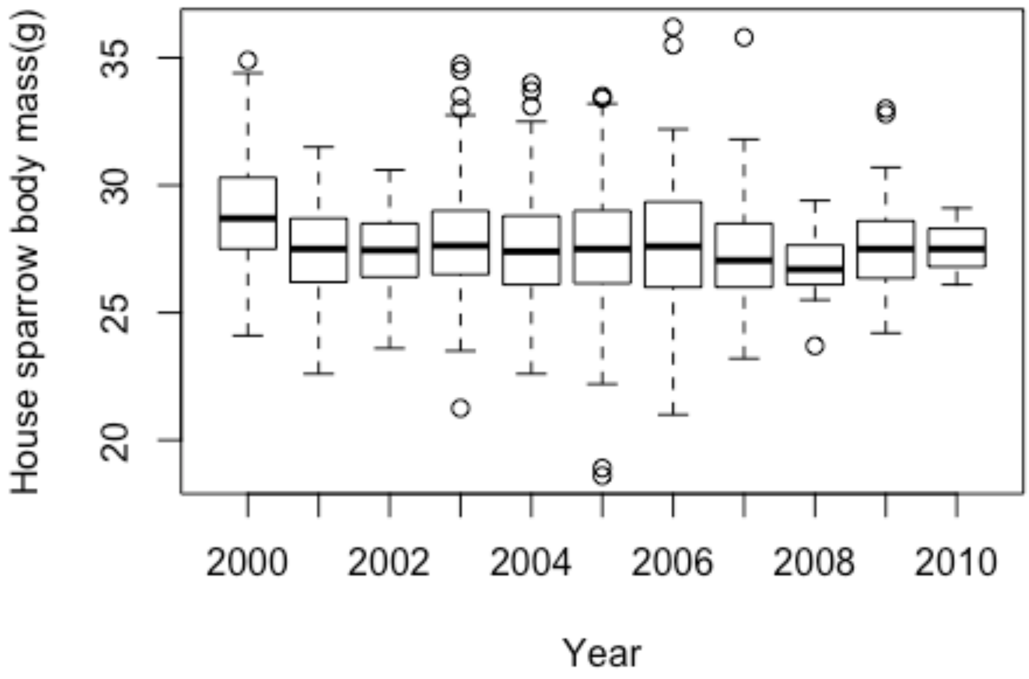
This is the mean of the old variable `BirdID` - that's clearly a completely insane value to compute, yet R does it - because it has no brains - it just follows the orders you give it. That's where your biologist skills are needed - to tell R what variables mean what biologically.

```
mean(d$BirdIDFact)

## Warning in mean.default(d$BirdIDFact): argument is not numeric or logical:
## returning NA

## [1] NA
```

And we get an error message when we try to take the mean of our new, factorial, and categorical, variable. That makes sense. We can't take a mean of a bunch of names.

Finally, we'll move on to the last variable - `Year`. Year is a tricky variable. It can be completely categorical - we can assume, that every year we go out to Lundy to measure sparrows, something is different. Some years, sparrows will be fat. Other years, the weather might be worse and they might be skinny. So, we can postulate that there will be annual variation in body mass of house sparrows. But we do not predict a directional change in this. We can thus plot `Year` as a factor:

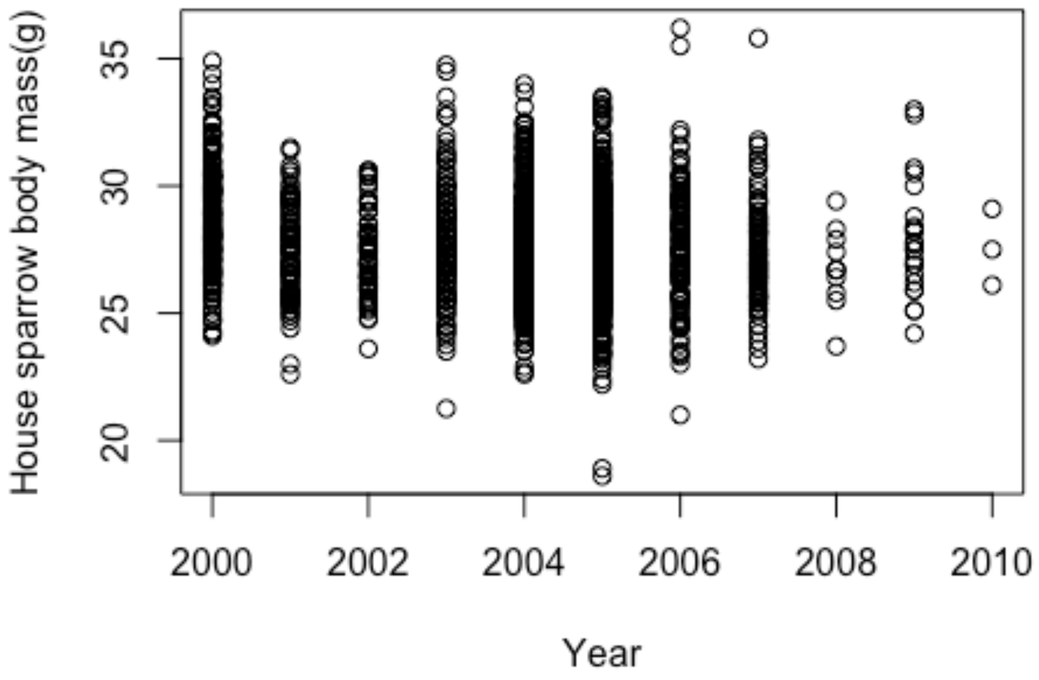
```
plot(d$Mass~as.factor(d$Year), xlab="Year", ylab="House sparrow body
mass(g)")
```



And we get a plot that shows quite some variation between years in body mass. R clearly interprets the Year variable as a categorical variable, because, even though we did not tell R to, it made a box plot. This will happen if your explanatory variable is categorical: because there is no possibility for decimals. If `Year` is interpreted as categorical, then there is no option for 2017.5. It also means there is no tangible difference between the years 2000 and 2010, and 2015, even though the difference between the first two is 10, and the latter two is only 5. `as.factor()` shapeshifts a variable into one where numbers are interpreted as text, and have no value per se attached. Note that, though, R is not completely stupid. It orders the years from early to late. However, that's not a particularly clever move by R. It is just that any alphanumerical text (anything with letters and numbers that's not a value) in R is by default ordered - you guessed it - alphanumerically. So, a year called y2020 would show up before a year called z2012. Sometimes, we can use this fact to our advantage - it is nicer to see the graph above with the years ordered, even though we know that it's meant to be categorical.

What would happen if we did not tell R that `Year` is a categorical variable (remember, we achieved that by adding `as.factor()` to `d$Year`)? Let's try!

```
plot(d$Mass~d$Year, xlab="Year", ylab="House sparrow body mass(g)")
```



Umm. That looks very different indeed! And that is because R interprets `Year` as a continuous variable, and it expects there to be at least the possibility for decimals, and because the differences between years are important, and the values mean something. 2020 is 10 years later than 2010. Now, when would that be of interest? For instance, when we think about climate change. In birds, there is the issue that the food for the birds young is dependent on temperature. Blue tits *Cyanistes caeruleus*, for example, rely on caterpillars that emerge with tree budburst. As the budburst occurs ever earlier in the year, due to spring warming, over the years, blue tits will start breeding too late, and have fewer chicks. Therefore, we can postulate, there is strong selection on early laying dates. And evolution should take care of this - over the years, blue tits should shift their laying dates to ever earlier dates. In this case, we are interpreting year as continuous variable.

In our sparrow data there is no laying date. So let's have a look at some blue tit data:

```
rm(list=ls())
```

We need to clean our workspace - if we are to read in another dataset with also a variable called `Year` there is mighty opportunity for confusion on the R front!

```
b<-read.table("BTLD.txt", header=T)
str(b)
```

Again, let's investigate the variables. `LD.in_AprilDays` is the laying date in days from the 1st of April onwards. As such, 3 is 3. April, and 32 is the first of May. Ok, that's, while an integer, a continous variable because the later they lay their eggs, the worse for their chicks. We also get information on the clutch size when the chicks are 7 days old - the number of offspring in the nest seven days after hatching. While there can hardly be a fraction of a chick, it is still a continuous variable because the differences mean something. And a mean clutch size of

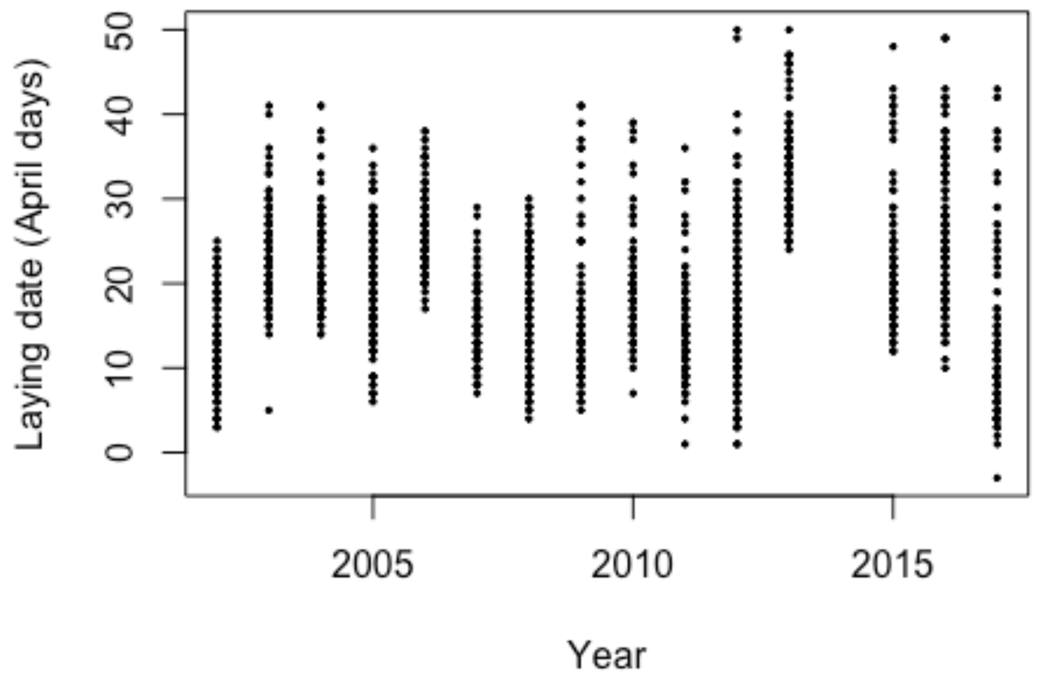
```
mean(b$ClutchsizeAge7, na.rm = TRUE)

## [1] 9.168243
```

is meaningful, so `ClutchsizeAge7` is a continuous variable.

Then there is an `IDFemale` - no males because males lay no eggs. This time, we get a number with a letter attached, and some versions of R automatically identifies that as a `Factor` - a categorical variable. If the version you're working with identifies it as a character, you may have to set it to be a factor – as we've done above. There's 1463 different females.

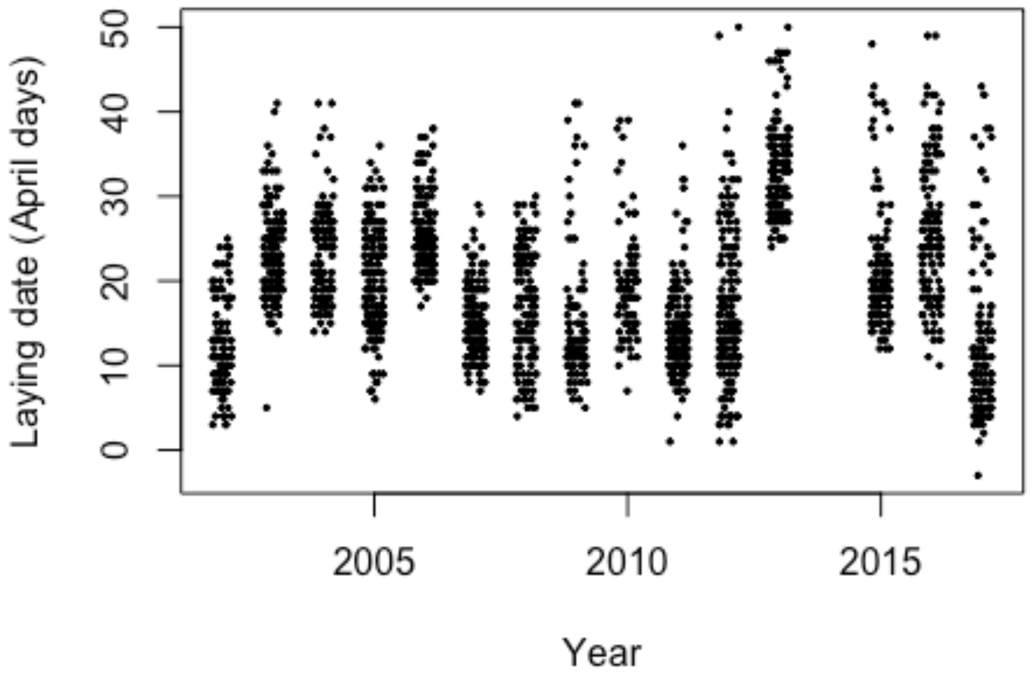
So we're left with `Year`, which is, as with the sparrows, considered by R as an integer. We, however, want it to be considered numerical - because we want to see whether there is a change from early years towards late years in laying date. So let's plot this:

```
plot(b$LD.in_AprilDays.~b$Year, ylab="Laying date (April days)", xlab="Year",
pch=19, cex=0.3)
```



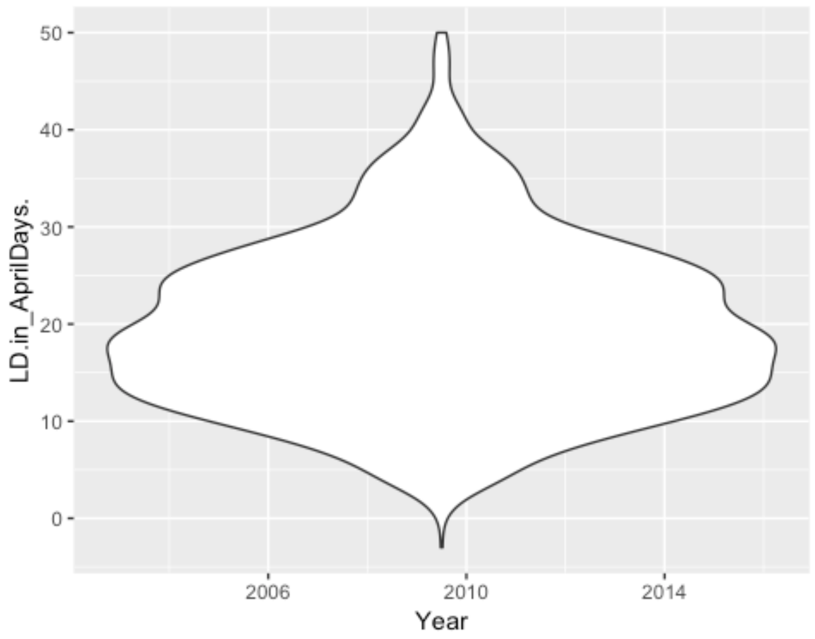
I find this really hard to see. That's becasue while both are continuous variables in the statistical sense, they are measured at a precision unit that gives us many measurements of the same value. We could change the plot to get a better view of the single points. There are multiple ways to achieve this. One, that I like particularly, is to add some random noise around each point in x-axis direction, so that we can see better. Adding that noise is done using the function `jitter()` in R:

```
plot(b$LD.in_AprilDays.~jitter(b$Year), ylab="Laying date (April days)",
xlab="Year", pch=19, cex=0.3)
```



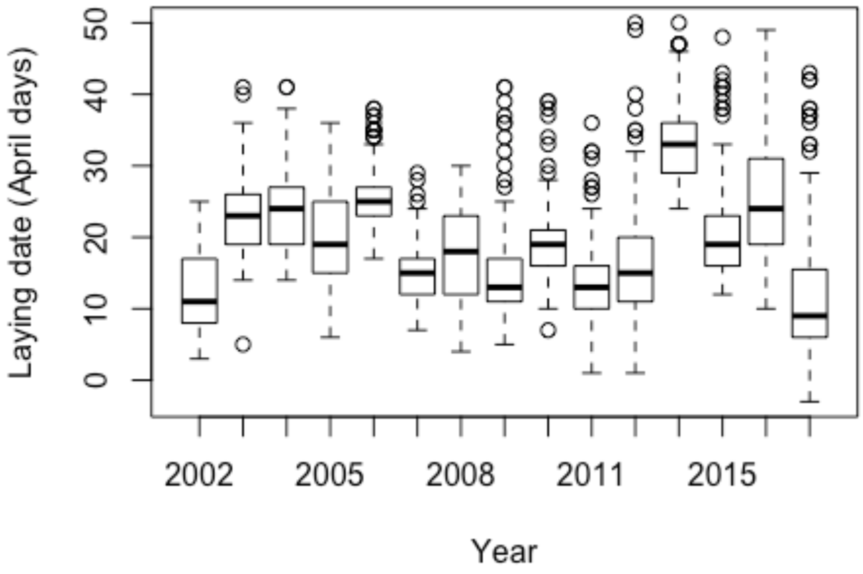
This is much better. Now we can see, at least approximately, how many points are there for each laying date in each year. What we have done here is visualise the distribution. Of course, you need to say so in the figure legend - otherwise people might think that the noise is meaningful biological data! There are other methods for this - another one that I like a lot is the violin plot. It's part of the ggplot2 package:

```
require(ggplot2)

## Loading required package: ggplot2

p <- ggplot(b, aes(x=Year, y=LD.in_AprilDays.)) +
  geom_violin()
p
```



Ummm. This doesn't look like we intended it to. It looks like R interpreted `Year` as a continuous variable - but that's what we want! Yes, it is what we want. We want to interpret `Year` as continuous variable, where later years lead to earlier laying dates and the other way around. But - we want to see the data still visualized categorically - but interpreted continuously. So, in a way, our old boxplot would do for an assessment of the biological interpretation:

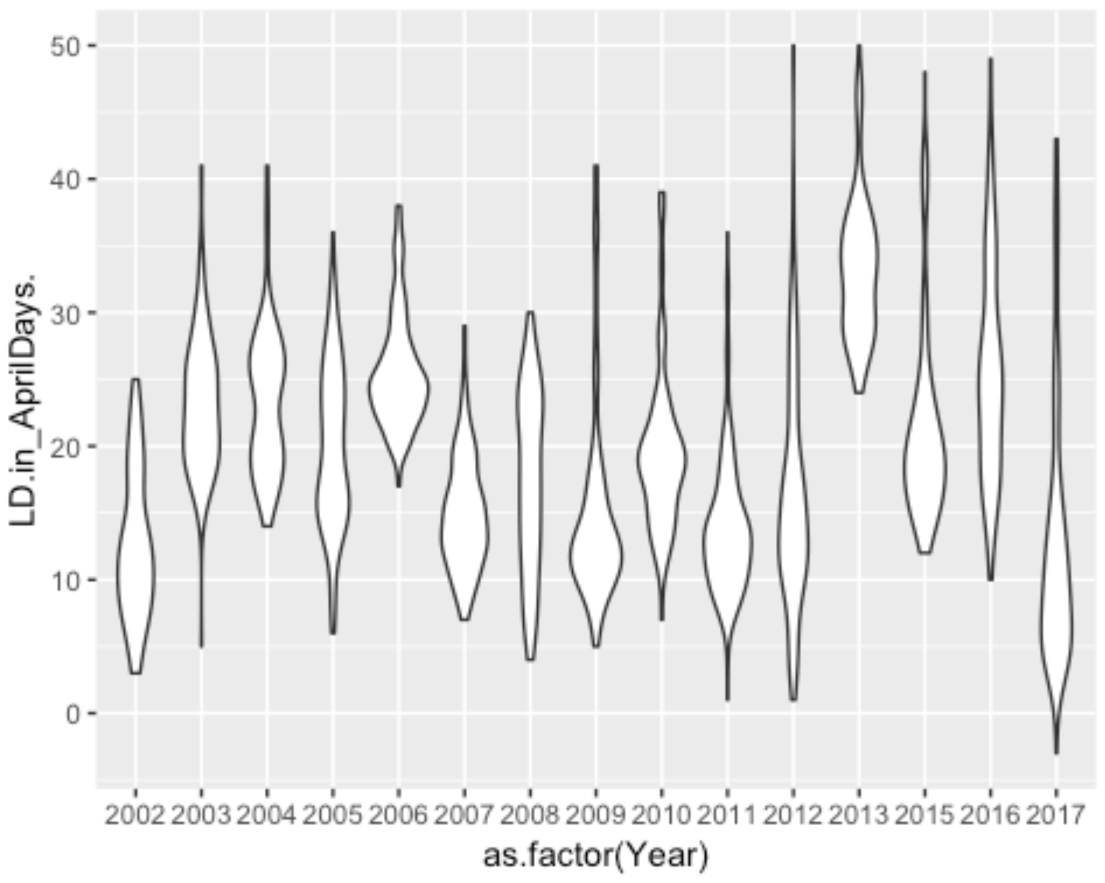
```
boxplot(b$LD.in_AprilDays.~b$Year, ylab="Laying date (April days)",
xlab="Year")
```



That's confusing? Well, yes, it is. Sometimes, you want to interpret a variable as continuous, and you also want to run the statistics on the variable as a continuous variable - for instance, to calculate the mean. But for visualization, sometimes it can be best to plot the variable as categorical. Both, calculating the statistic, and making a visualization, have different purposes. And only a biologist can make sense of this – someone who only knows statistics couldn't. So, yes, you can have a variable in your dataset that R treats as integer, but that is biologically a factor. And, dependent on your research question and hypothesis, you might have to run the variable in your analysis (mean, t-test, linear model) as continuous variable (e.g. `Year` in the climate change context), and sometimes, you have to run it as a categorical variable, e.g, if you want to look at annual variation. Oddly enough, when you plot it for visualisation, your thinking has to be different - it has to go around the idea of how can you best visualize your idea, and how you can achieve that. And how will the respective plotting package interpret your variable. It's a lot of fiddling, and that's normal. When I do this, there's typically a lot of trial and error, a lot of looking up help functions in R, and googling it – that is completely normal for any coding activity. So, don't be demotivated by many red error lines in your R console - that's part of the process!

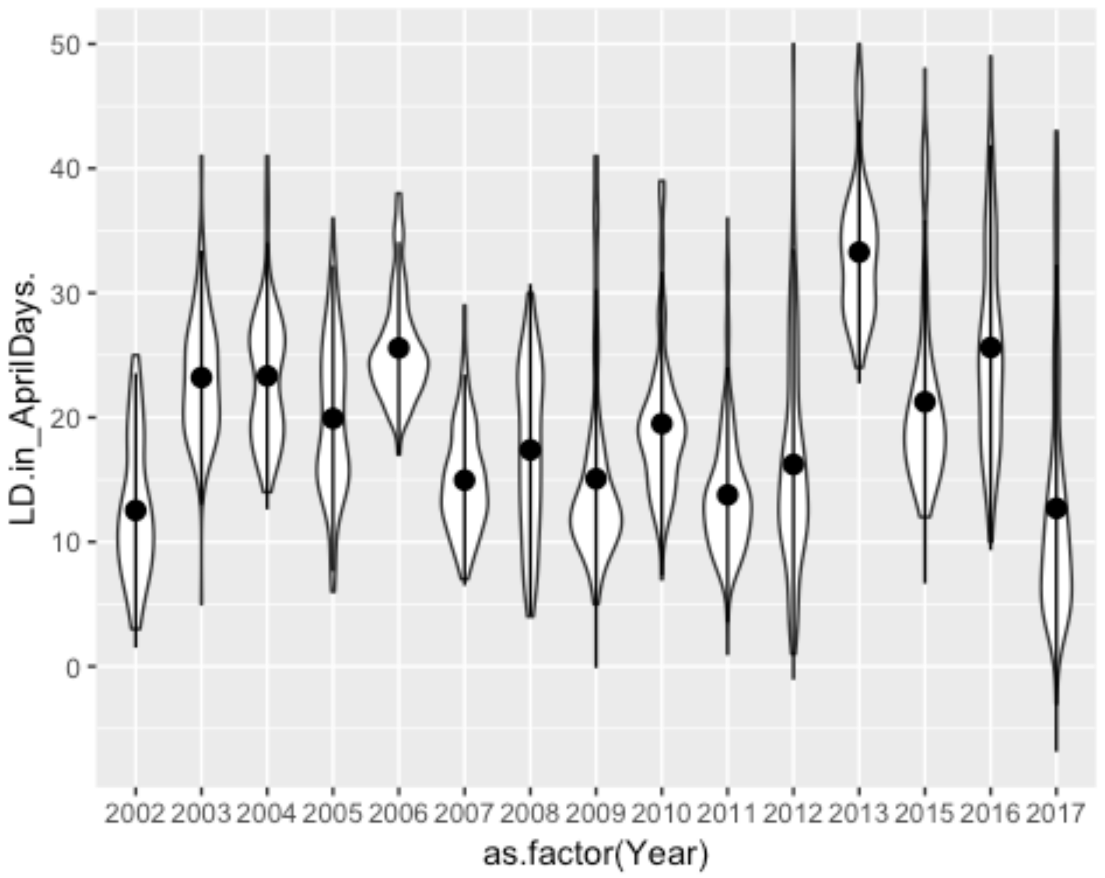
But I digress. Back to our visualisation with ggplot2! We wanted a nice violin plot. We can use `Year` as a factor (=categorical variable) here and then we get a nice plot:

```
p <- ggplot(b, aes(x=as.factor(Year), y=LD.in_AprilDays.)) +
    geom_violin()
p
```



That’s much better! Now we can add some descriptive statistics (mean, and standard deviation):

```
p + stat_summary(fun.data="mean_sdl",
                 geom="pointrange")
```



Now. That’s a nicer plot. We get descriptive statistics in it - we see the range of laying date in each year, we see how the data is distributed across days, and we can see where the mean is.

However, the take home from this is - when you run statistics, when you plot graphs, when you collect data - always have in mind whether your data is categorical or continuous, and what you need from it. Some continuous data can be treated as categorical - but rarely the other direction.

No excercises.

Statistics with Sparrows - 04

Julia Schroeder

July 2020

HO 04

Precision and standard error

Learning aims

- To understand the link between the standard error of the mean, and the sample size
- To understand that standard errors belong to a statistic (typically a mean)
- To understand that standard errors are a measure of precision

Standard errors

Standard errors (se) are a good way to display uncertainty. You can caluclate them using this equation:

$$se = \sqrt{\frac{s^2}{n}}$$

Let's do this for Tarsus in our dataset. Some basic housekeeping to start with:

```r
rm(list=ls())

d<-read.table("SparrowSize.txt", header=TRUE)
d1<-subset(d, d$Tarsus!="NA")
seTarsus<-sqrt(var(d1$Tarsus)/length(d1$Tarsus))
seTarsus

## [1] 0.02096211
```

and now only for 2001:

```r
d12001<-subset(d1, d1$Year==2001)
seTarsus2001<-sqrt(var(d12001$Tarsus)/length(d12001$Tarsus))
seTarsus2001

## [1] 0.1030623
```

The SE of 2001 is about five times the one for the total population. That's shocking! So, what determines how larger or small our measurement of precision, the SE is? Let's have a closer look at it:

$$se = \sqrt{\frac{s^2}{n}}$$

with a bit of algebra we find out this is the same as

$$se = \frac{s}{\sqrt{n}}$$

The important bit in this part is to have a good look at the denominator. N represents sample size. s^2 is, as we know, the variance. Thus, if se is an indicator of the uncertainty, then what do we have to do to make this number the smallest possible? We have to increase the sample size! We can do some basic math to find out how much larger our sample size needs to be to half our standard error:

$$\frac{se}{2} = \frac{s}{2\sqrt{n}} = \frac{s}{\sqrt{4n}}$$

This is the square root law of sample size: to improve your precision by doubling it, you need to increase your sample size by it's squared term: - 4!

Another reason why I showed you these equations for the standart error is that it is not only related to variance, but also to our 95% confidence interval. Remember that? Here is the equation to calculate the 95% confidence interval. It is also a measure of precision. Can you spot the standard error?

$$CI_{95\%} = \pm 1.96 \frac{s}{\sqrt{n}}$$

or

$$CI_{95\%} = \pm 1.96 se$$

Note that this works best for large sample sizes (approx. N>50).

Visualizing a change in precision

Let's see if we can visualise this in an appealing way, as I've shown briefly in the lecture. The grand idea now is to produce a plot that shows how the mean varies with increasing sample size, and how the accompanying standard errors shrink with sample size. Many times in statistics, it is useful to first generate a dataset with known quantities, and then run the analyses on it as a test of how it would turn out. To do that, we have to make up some data:

I will use a dataset that I create myself - that way I know what the "true" mean is. I will simulate a dataset of golden glamour dragons' tail lengths. But first I will clear the workspace:

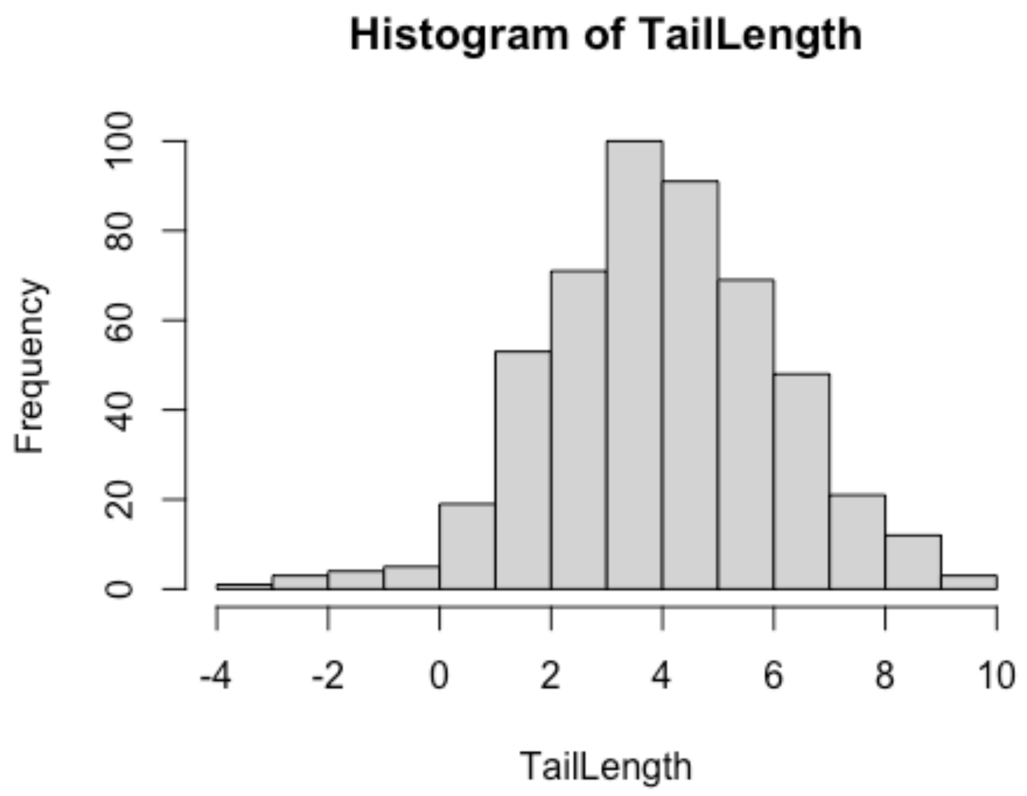
```
rm(list=ls())
```

Then I will create a vector with 500 entries of measurements of dragon tail lengths. I want a mean of 3.8 meter, and a standard deviation of 0.75:

```
TailLength<-rnorm(500,mean=3.8, sd=2)
```

Check out the function `rnorm()` if you haven't done that yet. It's very useful in many situations. This function creates randomly drawn values from a normal distribution, and you can say what mean and sd you want. Next, I'll check out the data, see if it fits my expectations, and plot it:

```
summary(TailLength)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##  -3.403   2.606   3.939   3.948   5.350   9.964

length(TailLength)

## [1] 500

var(TailLength)

## [1] 4.284017

sd(TailLength)

## [1] 2.069787

hist(TailLength)
```

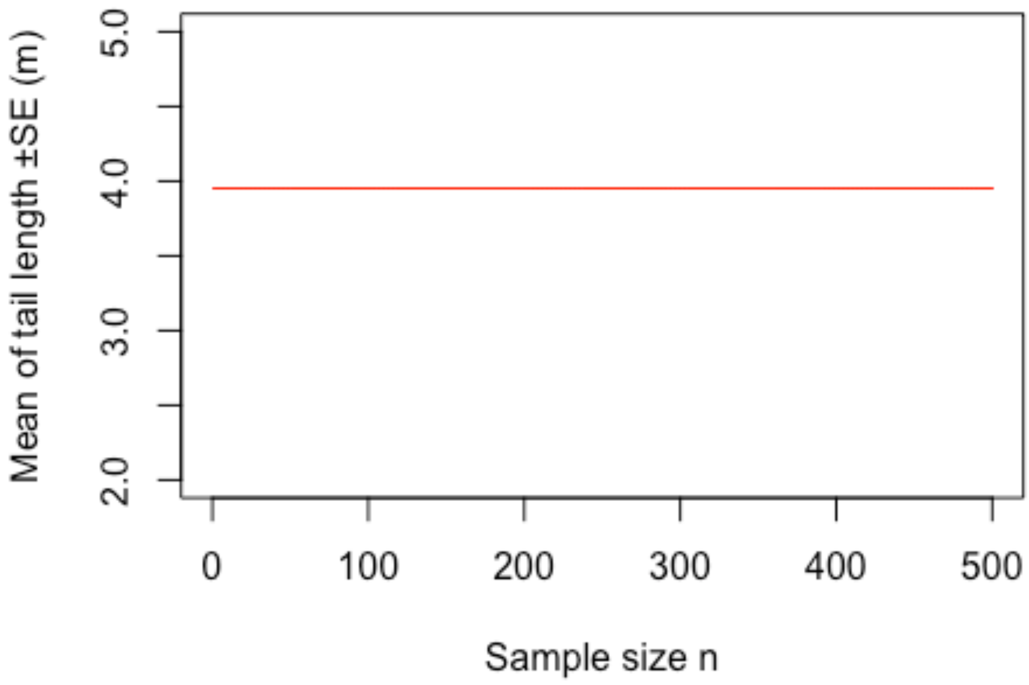


That's reasonable - the mean is 3.79, and the standard deviation is 0.71. *Note that these numbers may look different in your console - that is because `rnorm()` draws random numbers, so of course it will not be exactly the same - that's the whole point of it.*

Now to the real exercise. I want to randomly draw a specified number of observations from this dataset, and calculate the mean, and standard error, and plot it. I actually want to do that for sample sizes from one, all the way up to 400. To do that I have multiple options. I chose here a classical, traditional, `for` loop, and plot the points while I'm looping. I first prepare the canvas of the plot. To do that, I need to know the maximum and minimum values that need to be displayed. Obviously, the x-axis will run to 400. The y-axis will run from the minimum to the maximum mean, and I will give it some space for the standard error bars. I want to plot the grand total mean - as a black line so I can compare the other means to it. To do that I need to create a dataset with a vector for x that runs from 1 to 400, and a y vector that holds 500 times the grand total mean. I can create y in many ways, but by multiplying it with x I make sure they are both of the same length.

```
x<-1:length(TailLength)
y<-mean(TailLength)+0*x
min(TailLength)

## [1] -3.403074

max(TailLength)
```

```
## [1] 9.964022

plot(x,y, cex=0.03, ylim=c(2,5),xlim=c(0,500), xlab="Sample size n", ylab="Me
an of tail length ±SE (m)", col="red")
```



Now I need to populate my graph with means of samples of this data. To do this, I run the `for` loop. But before I do that I make vectors for the means (`mu`) and the standard errors (`SE`):

```
SE<-c(1)
SE

## [1] 1

mu<-c(1)
mu

## [1] 1
```

These two vectors will be filled with the data. Having a vector allows me to use the elements of it - if I want to fill the first element with the mean of a sample size of n=1, I can say `SE[1]<-`. Now when I use a placeholder for the sample size, in this case, `n`, I can loop through it and fill up the vectors one by one:

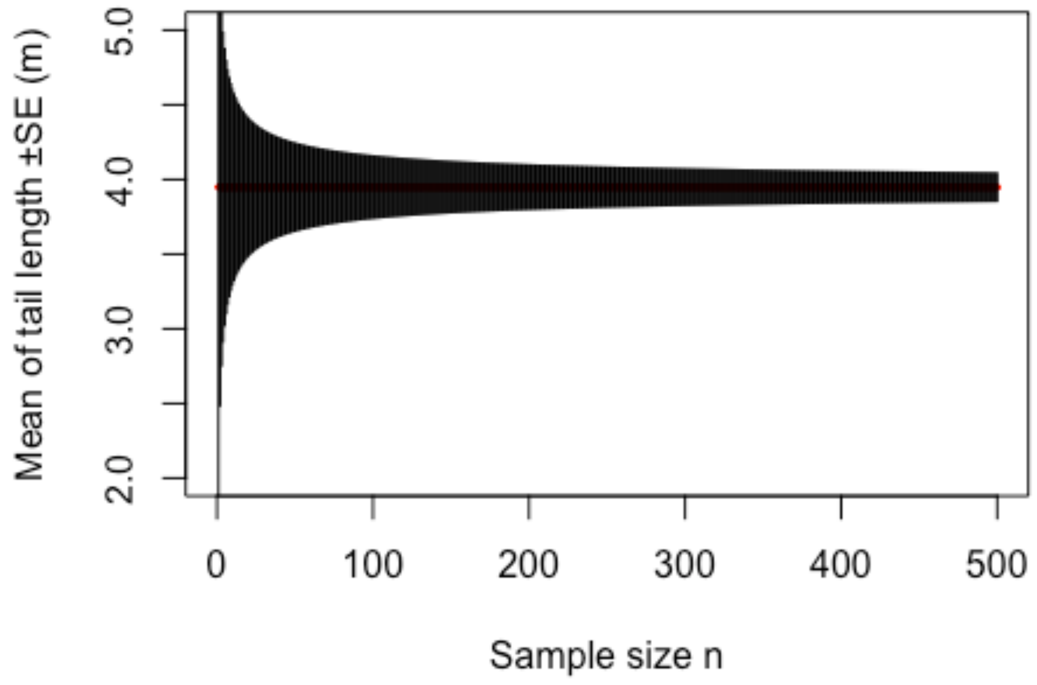
```
for (n in 1:length(TailLength)) {
  d<-sample(TailLength, n, replace=FALSE)
  mu[n]<-mean(TailLength)
  SE[n]<-sd(TailLength)/sqrt(n)
}
```

Now I want to first see if what I've done did what I wanted it to do:

```
head(SE)

## [1] 2.0697867 1.4635602 1.1949919 1.0348933 0.9256367 0.8449869

head(mu)

## [1] 3.948466 3.948466 3.948466 3.948466 3.948466 3.948466

length(SE)

## [1] 500

length(mu)

## [1] 500
```

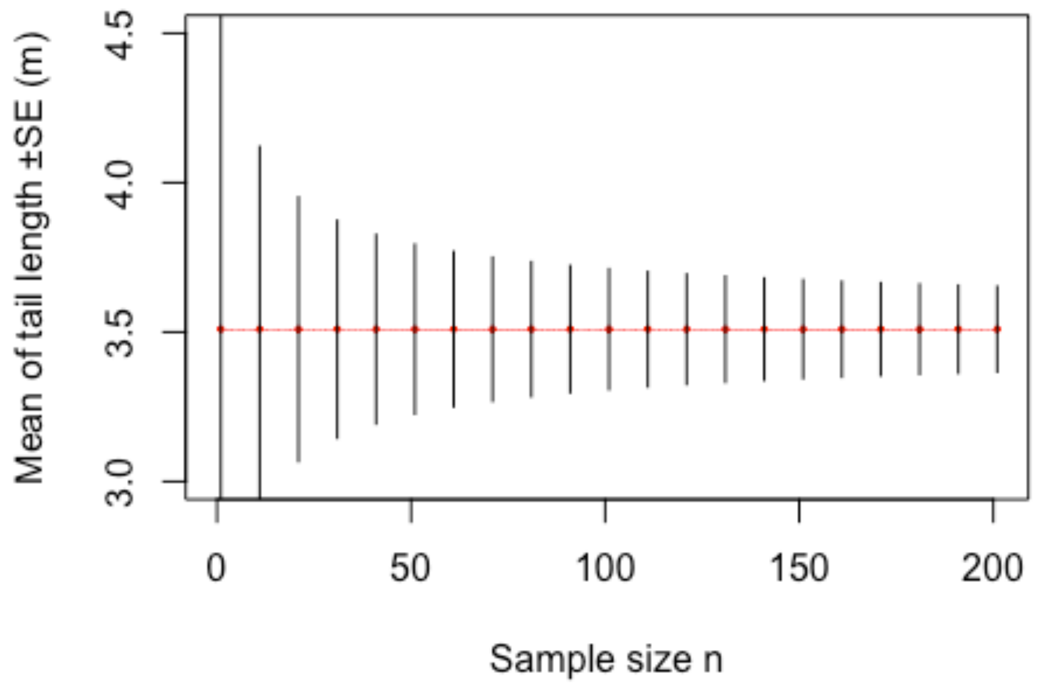
That looks fabulous. Now I can add to the previous plot. I want to plot the new means on the y-axis, and the sample sizes (n) on the x-axis. Then I want to plot error bars that go from the means to mean±SE.

```
up<-mu+SE
down<-mu-SE
x<-1:length(SE)
segments(x, up, x1=x, y1=down, lty=1)
```



While this seems to be doing what it's supposed to do, it's not so nice on the eye. Maybe I need to thin it out - also, 500 seems to be a bit too long - 200 will do. I'll do 201 because then I can sample starting from 1, by 10:

```
rm(list=ls())
TailLength<-rnorm(201,mean=3.8, sd=2)
length(TailLength)

## [1] 201

x<-1:201
y<-mean(TailLength)+0*x
plot(x,y, cex=0.03, ylim=c(3,4.5),xlim=c(0,201), xlab="Sample size n", ylab="
Mean of tail length ±SE (m)", col="red")
n<-seq(from=1, to=201, by=10)
n

##  [1]   1  11  21  31  41  51  61  71  81  91 101 111 121 131 141 151 161 1
71 181
## [20] 191 201

SE<-c(1)
mu<-c(1)
for (i in 1:length(n)) {
  d<-sample(TailLength, n[i], replace=FALSE)
  mu[i]<-mean(TailLength)
  SE[i]<-sd(TailLength)/sqrt(n[i])
}
up<-mu+SE
down<-mu-SE
length(up)

## [1] 21

length(n)

## [1] 21

plot(x,y, cex=0.03, ylim=c(3,4.5),xlim=c(0,201), xlab="Sample size n", ylab="
Mean of tail length ±SE (m)", col="red")
points(n,mu,cex=0.3, col="red")
segments(n, up, x1=n, y1=down, lty=1)
```



And it becomes clear how the standard error shrinks with increasing sample size.

Exercises:

Note: nothing needs to be uploaded. These exercises are intended to improve your understanding of the statistics, the functions, and improve your ability to problem solve and code.

1) Calculate the standard error of Tarsus, Mass, Wing and Bill length of the complete population sample (as opposed to all sparrows in this world) Note N of each. Then, subset the dataset to only 2001 data `d1<-subset(d, d$Year==2001)`, as we did in the lecture. Calculate SE for Tarsus, Mass, Wing and Bill length for the 2001 sample. Calculate the 95% CI for each mean.

2) Play around with the last plot we made. Change the values of the simulation. Change the mean, the standard deviation to other values, and see how it affects your plot. Just try out various combinations! This should help you to get better at

- Understanding how mean and SE and sample size are linked
- Understanding how to plot things in R
- Understanding how to use for loops in R
- Understanding how to make, access and use variables, vectors and data frames in R

Note: it is ok if you find this difficult. This is something that you need to do, and do again, and redo again, and the more often you practice this the better you will become at understanding it. So use this time to repeat and practice.

Non-compulsory:

3) If you found the above easy and want to test out your coding and stats skills, this is a task for you.

Sample mass of the whole sparrow dataset. Then produce a plot to see at what sample size the standard error become so small that you'd be confident to get a reasonably precise mean - precise with respect to the grand total mean of the whole dataset. (The next part is rather philosophical). Then look at sample sizes in different years and think about what sort of biological questions one could answer with one year's data, and what not.

Statistics with Sparrows - 05

Julia Schroeder

July 2020

HO 05

Comparing means

Learning aims

- Understand that effect size is most important when comparing means estimates
- Get insight into why the p-value is only secondary
- Learn how to execute a t-test
- Learn how to judge the biological significance of the results
- Apply the effect of sample size on mean estimate precision

Hypothesis testing

What do you remember about hypothesis testing? I am sure there are some concepts in your brains about p-values and 0.05?

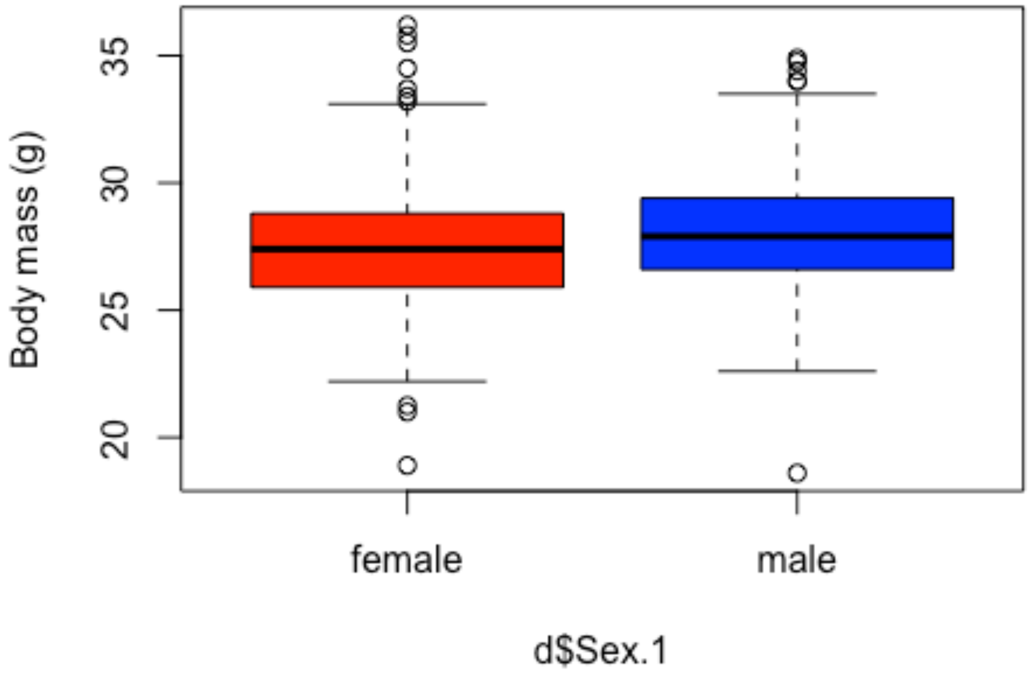
p-value: — the probability of observing a particular result (and more extreme results) when the null hypothesis is true (i.e. there is no effect).

Ok, let's try this out. But first, - you know, housekeeping:

```
rm(list=ls())

d<-read.table("SparrowSize.txt", header=TRUE)
```

We will use it to test for a difference in female and male body mass in house sparrows. Let's see what we can gleam from a boxplot:

```
boxplot(d$Mass~d$Sex.1, col = c("red", "blue"), ylab="Body mass (g)")
```



It looks like males are slightly heavier than females, but how can we tell if that difference means something? We first have to find out our hypothesis. In this case, it is that the DIFFERENCE between males and females in body mass is different from zero. Our null hypothesis is that the difference is zero. If that is not supported, then there are two options: If this difference is positive, it means males have larger body mass, if it is negative, it means males have lower body mass than females.

```
t.test1 <- t.test(d$Mass~d$Sex.1)
t.test1

##
##  Welch Two Sample t-test
##
## data:  d$Mass by d$Sex.1
## t = -5.5654, df = 1682.9, p-value = 3.039e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.7669117 -0.3672162
## sample estimates:
## mean in group female   mean in group male
##             27.46852             28.03558
```

R is quite helpful here in that it tells us the alternative hypothesis: that the difference between the two groups is not equal to 0. That is what we found, because the t-value is large, given the degrees of freedom (1683). This means that with a very high probability, males are heavier than females. Should we accept the alternative hypothesis then?

Yes, but wait, does it actually make biological sense? The p-value is super small, most people would be very excited. But I want to teach you to not focus too much on the p-values, instead look at 95%CIs and the parameter estimates. The male mean is 28.0g and the female mean is 27.5g. The difference is about half a gram. However, that's the mean difference. But can we say something about the precision of this effect? Look at the output, it gives us a 95%CI of the difference between males and females: 95% of the differences between males and females fall between -0.77g and -0.37g (males heavier). This gives us a good indication about how important this difference is in biology.

5% of the times, however, the difference will be outside of this interval. That's a type 1 error. There is a 5% chance that this data is actually not representing the real world, and that the difference between the sexes is actually 0.

Large datasets are more likely to pick up on small effect sizes (remember the square root law). Let's see if we would reduce our dataset to the 50 first rows, could we still detect a difference between male and female body mass?

```
d1<-as.data.frame(head(d, 50))
length(d1$Mass)

## [1] 50

t.test2 <- t.test(d1$Mass~d1$Sex)
t.test2

##
##  Welch Two Sample t-test
##
## data:  d1$Mass by d1$Sex
## t = 0.33484, df = 26.84, p-value = 0.7403
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.9361255  1.3011254
## sample estimates:
## mean in group 0 mean in group 1
##         27.5200         27.3375
```

Oh. All of the sudden, we find no difference! Just because we took a smaller dataset. The lesson here is that with large datasets, you are more likely to encounter a statistically significant effect, but whether or not this effect is actually meaningful, is not something you can understand by looking at the p-value. In the first t.test, the p-value was very significant, however the real difference (effect size) was very small (0.3g more or less is not a lot, it is actually

Exercises:

1) Test if wing length in 2001 differs from the grand-total mean - Test if male and female wing length differ in 2001 - Test if male and female wing length differ in the full dataset - Test if male and female tarsus differs in the full dataset - Report all your results in a table and share with your group. Discuss what and how to present. Find at least two different ways of presenting your resutls in a table. Explore different table options.

2) Now, run a batch of tests - check for each year whether any of the measures differs from the grand total mean.

Upload one word document per group for (2). In this word document, present the results from (2) using what you've learned in (1), in a delightful and easy to understand way, as you would in a scientific publication. One word document per group to upload.

3) Then, see if you understand how to employ the t.test properly in R. There are eleven cohorts (years) in this dataset. Test whether the first five years differ in terms of Tarsus, Mass, Wing, and Bill from the latter six years. To do this, you will first need to create a two-level variable that denotes to wich group a certain observation belongs. Then you can test against this. Do it in two ways - once with **,** and once with ~.

To do that you will have to re-arrange the data, which might be tricky and take quite some of your time, a lot of brain power, and a lot of de-bugging! All this is something you will need to do quite often over the course of this year. So do try to wrap your head around it. Use all your mighty plyr functions for this. There will be various, vastly different solutions for this one - as long as the end result is the same, that's perfectly fine.

Call on your group for help. Discuss your possible solutions in your group, and later on the online discussion board.

Statistics with Sparrows - 06

Julia Schroeder

July 2020

HO 06

Statistical power

Learning aims

- Conduct a statistical power test

Hypothetically ...

After your graduation, you are approached by the Ministry of Magic (MoM) to investigate how the harvesting of horns from Romanian Longhorn dragons could be made more economical. If we'd knew whether male or female Romanian Longhorns had longer horns, we could systematically capture the sex with the longer horns. While not a particularly pleasant job, it is still the year of the pandemic and jobs are rare. Also, the MoM is a powerful institution and maybe, if you do this job, you can get a foot in the door for a proper biologist's job, so you accept. When you get started, you desperately hope, you don't have to measure too many individuals to get the answer, so a statistical power analysis to find out the reasonable number of dragons to measure is a good idea. You know that in a very old book you've read once, it has been mentioned that while horns are generally around 1m in length, females have longer horns by about 30cm. You look up that reference, but it remains vague. It does say it only measured 5 dragons and the standard deviation was 1.2. That seems a lot to you, but, alas. To work!

First, we google for R packages that run power analyses. There's a couple: `pwr()` is one that seems really nice, but at the time of writing, it's not been updated for the most recent version of R. If it has, do try it out! There's also a number of websites that directly compute the statistical power. But you want to do it in R! So on we look. Today, we're using `WebPower()`. If it's not installed on your machine, do that now. And don't ask a question online why the code doesn't run if you haven't installed the package yet - I assume you know how to install packages by now!

```r
rm(list=ls())
# we never forget this one!

require(WebPower)

?WebPower
```

Ok, that reads useful. Please click on the blue link “Index” in the help viewer and have a read through.

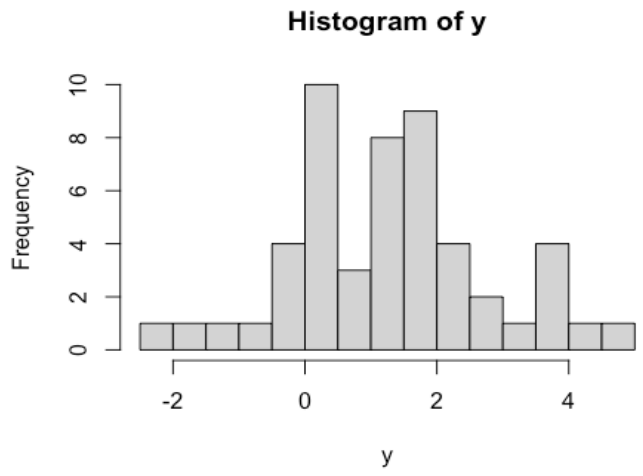
Of course, it’s always the last entry we’re interested in. Find the entry for “Statistical Power Analysis for t-Tests”, click on it, and have a read through.

The aim is to sample as many males as female dragons. Then we need our effect size d - also called *Cohen’s d*. It’s calculated as I said in the lecture as the difference between two means, divided by the standard deviation. What it does, in practice, is to provide us with a ratio - effect size to standard deviation. We take the values from the brief:

```
0.3/1.2

## [1] 0.25
```

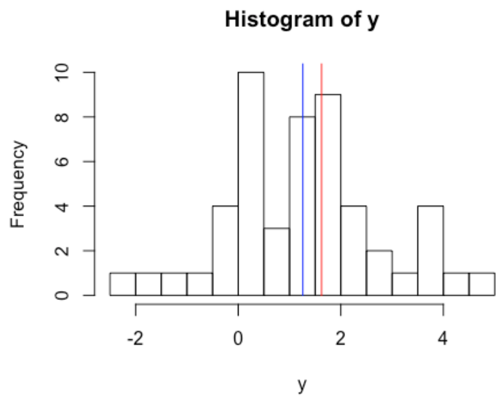
So d is 0.25. That means we want to detect an effect size that is 1/4 of a standard deviation. If we visualize that in a standard deviation density plot, it would look like the following. I simulate data with a mean of 1m, and a standard deviation of 1.3. Horns can’t be shorter than zero, so that’s our cut-off.

```
y<-rnorm(51, mean=1, sd=1.3)
x<-seq(from=0, to=5, by=0.1)

length(x)

## [1] 51

plot(hist(y, breaks=10))
```



```
mean(y)

## [1] 1.260218

sd(y)
```

```
## [1] 1.458004

segments(x0=(mean(y)), y0=(0), x1=(mean(y)), y1=40, lty=1, col="blue")

# and now 0.25 sd left of the mean (because females are larger)
segments(x0=(mean(y)+0.25*sd(y)), y0=(0), x1=(mean(y)+0.25*sd(y)), y1=40, lty
=1, col="red")
```



It becomes quite quickly obvious that that's a fairly small effect size, given the variability of the data. You are a bit disheartened, because this is ominous - you guess you will likely need a larger sample size. But you can't be sure unless you've done the math, so let's have a go.

```
?wp.t
```

Ok, we need two sample sizes n1 and n2 (actually, that's what we want to figure out). We'll assume they are the same, because it's the easiest - you can hardly predict now how many females and males you'll capture so going with chance is your best bet. Next, your effect size is 0.25. Alpha is 0.05 - that's set to default anyways - in 5% of all cases you won't detect an effect that's actually there. As for the power - you want your statistical power to be 0.80 or higher - because you've heard in my lecture that's conventionally agreed upon reasonably statistical power - a 20% chance of a false negative. Type is confusing but you go with two sample because you will have two samples - males and females. You will not compare one sample against a fixed mean. Alternative -direction of the alternative hypothesis. You *think* females may be larger but the reference is cryptic, the data is poor, and the study has been done a long time ago before they knew how to properly sex dragons (it's not as easy as one might think!). So you leave it with the default - two-sided. You have no clue what the tolerance is - it seems to have something to do with taking roots but it seems safe enough to ignore that for now. So you follow one of the examples below:

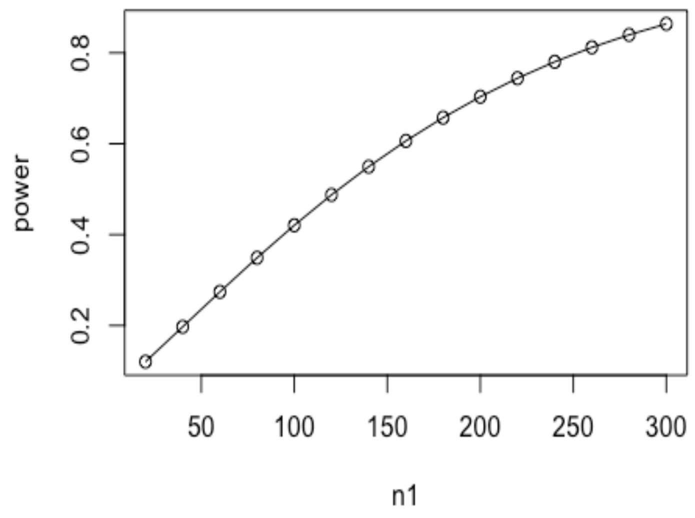
```
wp.t(d=0.25, power=0.8, type="two.sample", alternative="two.sided")

## Two-sample t-test
##
##             n    d alpha power
##     252.1275 0.25  0.05   0.8
```

```
##
## NOTE: n is number in *each* group
## URL: http://psychstat.org/ttest
```

Ah. The result is very clear, if disheartening. You need to sample 252 dragons in each group to get a clear answer. Yikes. You better go back to the MoM to ask for a raise - with that budget you'd never be able to capture that many Romanian Longhorns - if there even are that many alive! But at least you have hard data to show the MoM why you need more cash - if you sample fewer than 504 individuals it's unlikely you'll get a satisfactory answer!

As you look further through the examples in the help file, you see that it is possible to produce a power curve. Maybe that would help to convince the MoM to give you more money? If you could show them how low the power will be with a smaller sample size:

```
res.1<-wp.t(n1=seq(20,300,20), n2=seq(20,300,20), d=0.25, type="two.sample.2n
", alternative="two.sided")
res.1

## Unbalanced two-sample t-test
##
##      n1  n2    d alpha     power
##      20  20 0.25  0.05 0.1203354
##      40  40 0.25  0.05 0.1971831
##      60  60 0.25  0.05 0.2741094
##      80  80 0.25  0.05 0.3490542
##     100 100 0.25  0.05 0.4205383
##     120 120 0.25  0.05 0.4875700
##     140 140 0.25  0.05 0.5495495
##     160 160 0.25  0.05 0.6061828
##     180 180 0.25  0.05 0.6574078
##     200 200 0.25  0.05 0.7033333
##     220 220 0.25  0.05 0.7441884
##     240 240 0.25  0.05 0.7802824
##     260 260 0.25  0.05 0.8119726
##     280 280 0.25  0.05 0.8396409
##     300 300 0.25  0.05 0.8636746
##
## NOTE: n1 and n2 are number in *each* group
## URL: http://psychstat.org/ttest2n

plot(res.1, xvar='n1', yvar='power')
```



This is nice - it shows how much the power increases with increasing sample size - of one group! So, the only hope you have in getting away with a smaller sample size is if the standard deviation is somewhat smaller than 1.2m, or, if the effect size is actually larger than 0.3m. But you won't find out until you measure the dragons and as there will be a danger of false positives and negatives, there's no other way than measuring 500 Longhorns!

Exercises:

1. *You are given a dataset by your PI, where an experiment has been performed where the growth of two groups of bacterial colonies (n each group = 300) has been compared. Your PI has found an interesting difference between both groups - to an effect size of 0.11, with a p = 0.044. The result is rather ground breaking, but the PI is worried that it might be a misleading result. The PI didn't do a statistical power analysis, so they are uncertain about whether they had sufficient power. The bacterial colonies have since died, because nobody could enter the lab for months due to a pandemic. Hence, to repeat the experiment would be very expensive and time consuming. Calculate the statistical power of this test. Then discuss in your group what the result means and how the PI should proceed.*

2. *Write up a methods section that details what you have done and justifies your decisions. Then write a results section that details your results, and use tables and/or graphs to support your points. Then write up a conclusion section, where you must make one suggestion of how, and why, the PI should proceed.* ***Submit one word document per group.***

3. *Philosophical discourse: In your groups, discuss what the consequence of Type I and Type II errors are for the body of published literatures. If one in twenty results may be*

a false positive or negative, then what does that mean for the whole body of literature? Discuss repercussions and potential solutions.

Statistics with Sparrows - 07

Julia Schroeder

July 2020

HO 07

Simple linear functions

Linear functions come in the form:

$$y = a + bx$$

where b and m are single numbers (the intercept a, and slope b, respectively), and y and x are vectors. The latter means lists of multiple numbers, like in r. It's important the both, y and x have the same number of elements - both vectors need to be of the same length. To indicate that x and y aren't single numbers we sometimes use indices:

$$y_i$$

$$x_i$$

This is shorthand for: There are many y's - from y_1 to y_n, where n is the length of the vector and we want the ith element. The nice thing about this notation is it is also how R keeps track of its vectors, - but it uses double brackets [[]] to indicate an element:

```
rm(list=ls())
x<-seq(from = -5, to = 5, by = 1)
x

## [1] -5 -4 -3 -2 -1  0  1  2  3  4  5

x[[1]]

## [1] -5

x[[2]]

## [1] -4

x[[9]]

## [1] 3

x[[length(x)]]

## [1] 5
```

Neat, huh? We can even use a function inside the double brackets. This can be useful when we don't know the absolute length, or when it changes dynamically within our code. We can even use an i:

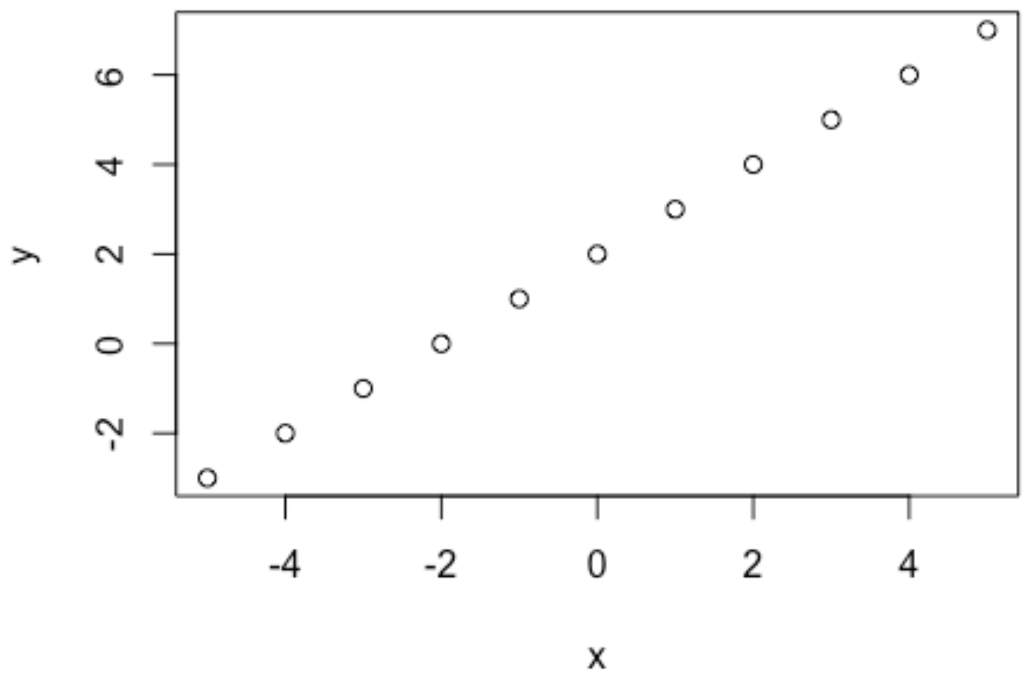
```
i<-1
x[[i]]

## [1] -5

i<- seq(0,10,1)
i

##  [1]  0  1  2  3  4  5  6  7  8  9 10

x[[i[[2]]]]

## [1] -5
```

Brain freeze much? The last example wasn't particularly applicable, but it will be helpful for your understanding if you can wrap your head around it. But we digress. Back to:

$$y \sim a + bx$$

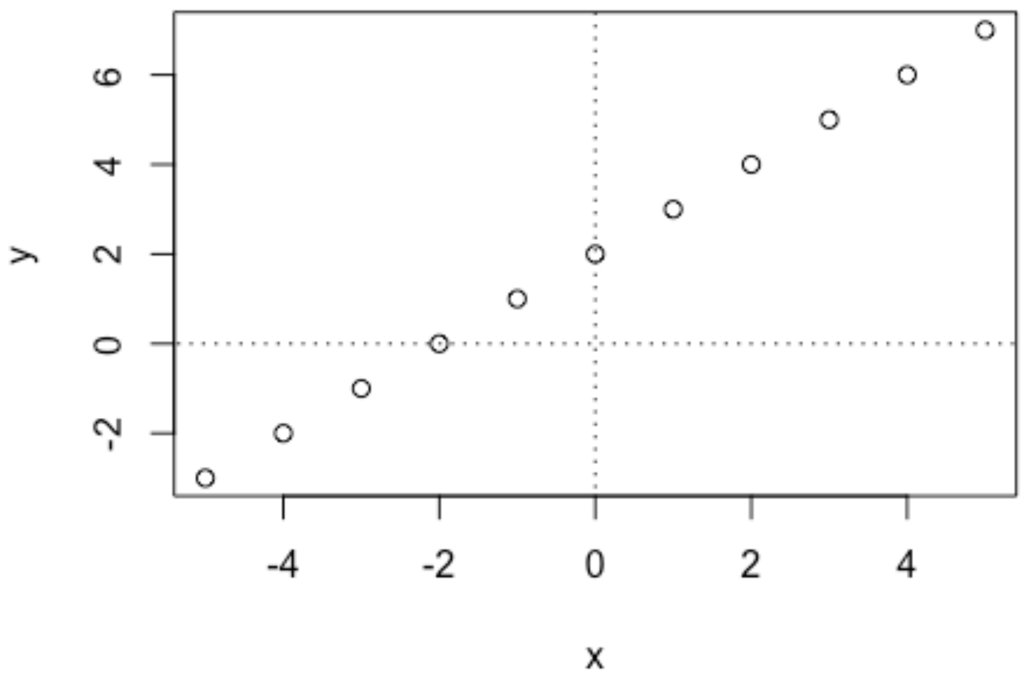
Now, we can come up with any sort of line. Say we want a line with an intercept of 2 (a=2 - meaning the line intercepts the y axis at 2), and a slope of 1 (b=1, meaning the line increases 1 y for each x), we can r to calculate the y's for us:

```
a<-2
b<-1
y<-a+b*x
plot(x,y)
```



This looks good, doesn't it? We'll add the cartesian axes so we can see the intercept better:

```
plot(x,y)
segments(0,-10,0,10, lty=3)
segments(-10,0,10,0,lty=3)
```

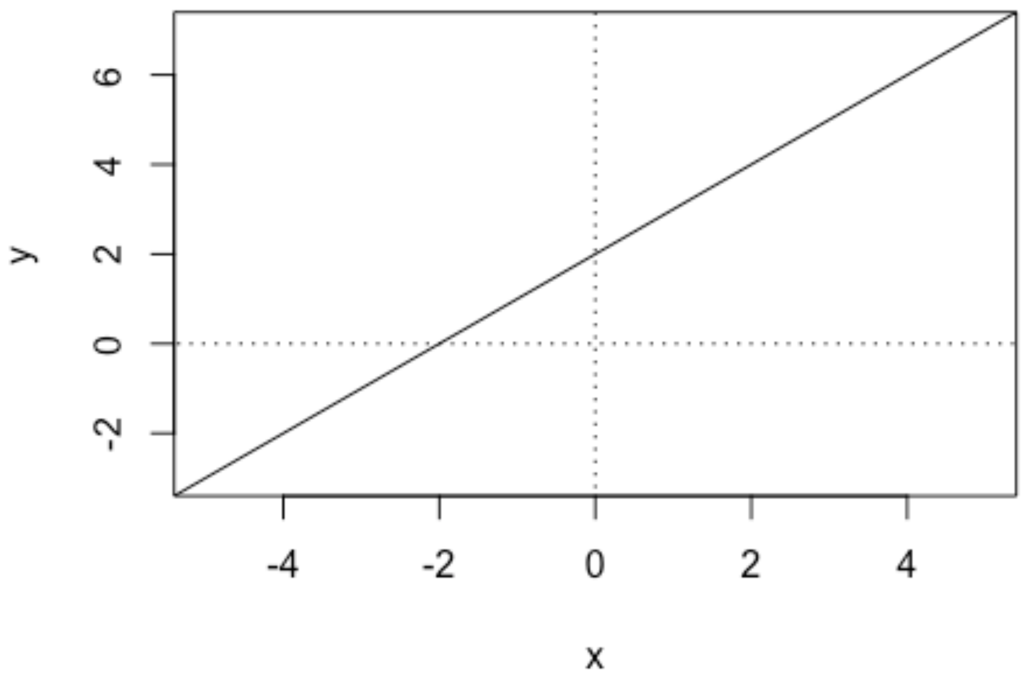


Better. But it's not a line, it's a number of dots. That's because x, and therefore also y, are not continuous - remember we told r to make the x-vector to only hold 11 data points. R plots lines using linear functions using `abline()`:

```
?abline
```

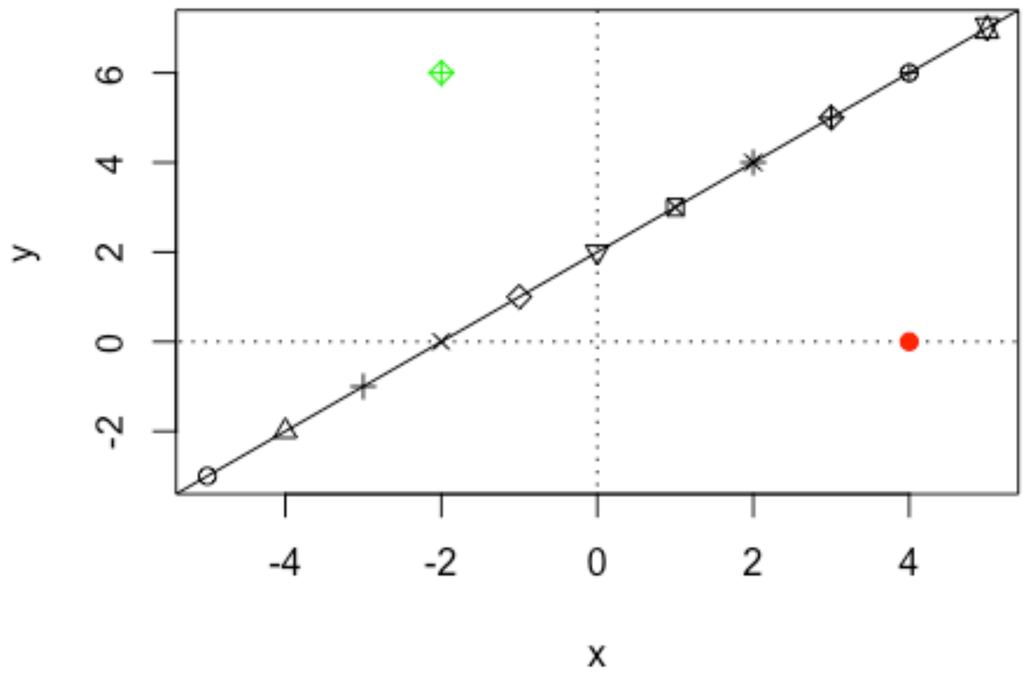
Read through the help text. It clearly tells us how to specify the intercept and the slope. Before we give it a go, we need to notice a small detail under the description of abline. It says *This function* **adds** *one or more straight lines through the current plot.* The **adds** is the key word here - it adds a line to an existing plot. So we first have to make a plot, then add to it. If we do not want the dots, we have a number of options. Today, we plot it but tell R to plot the dots in white, so we can't see them (it's a bit like cheating, but in coding ends often justify the means).

```
plot(x,y, col="white")
segments(0,-10,0,10, lty=3)
segments(-10,0,10,0,lty=3)
abline(a = 2, b=1)
```



Neat, huh? So we only need two values (intercept and slope) to plot any line. It's rather nice. Now you can plot points into this cartesian coordinate system anywhere you want, using the function `points`. Check it out in R:

```
plot(x,y, col="white")
segments(0,-10,0,10, lty=3)
segments(-10,0,10,0,lty=3)
abline(a = 2, b=1)

points(4,0, col="red", pch=19)
points(-2,6, col="green", pch=9)
points(x,y, pch=c(1,2,3,4,5,6,7,8,9,10,11))
```

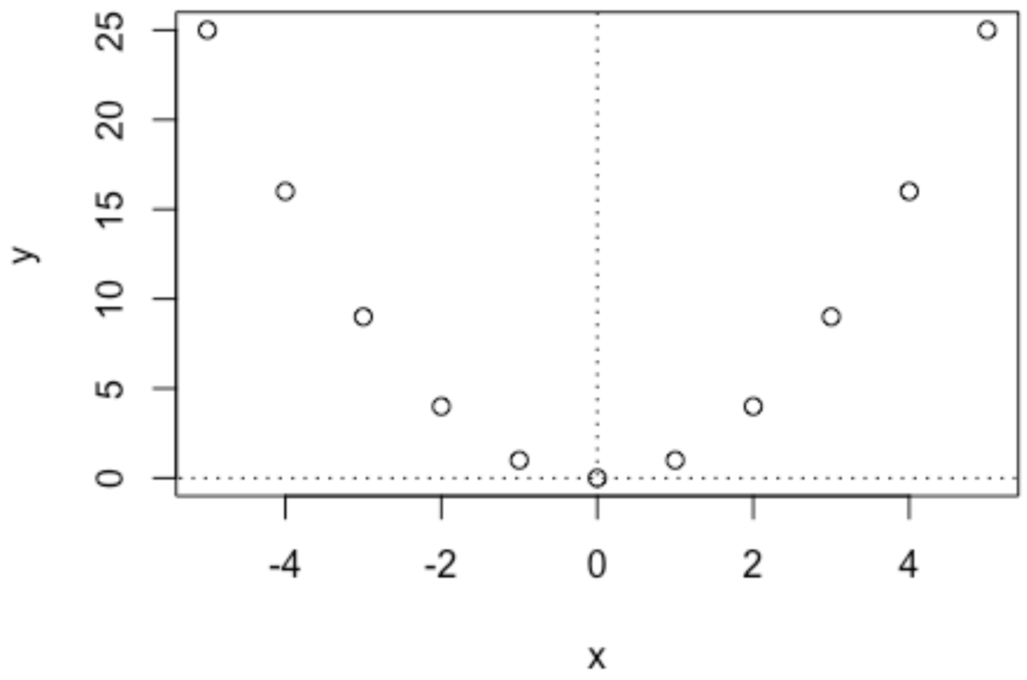


You can even plot a quadratic function. That's where x is squared. If you recall the quadratic function from school, it might have looked something like:

$$y = x^2$$

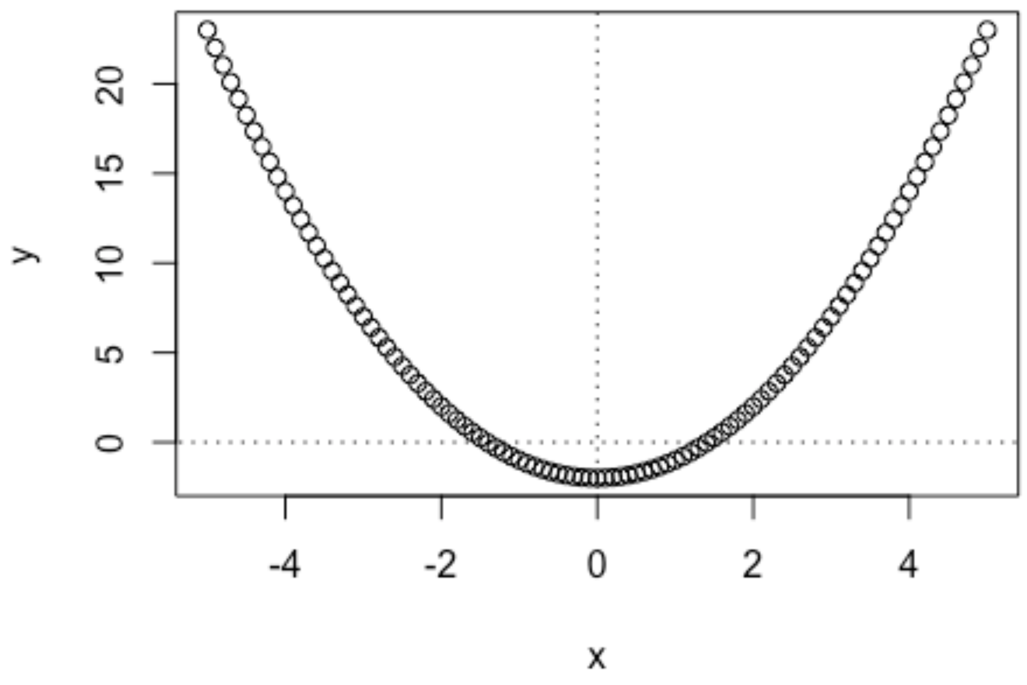
Let's see

```
y<-x^2
plot(x,y)
segments(0,-30,0,30, lty=3)
segments(-30,0,30,0,lty=3)
```



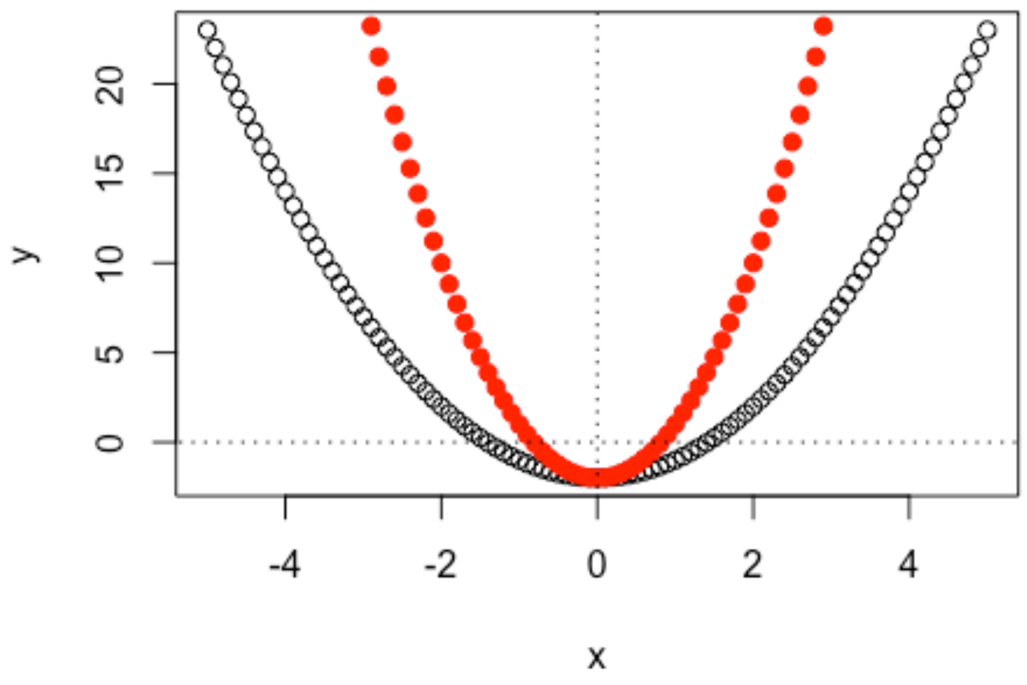
This is a quadratic curve, but it's still called a linear function. It goes through the origin - meaning the intercept is zero. Ok, now we can add an intercept *a* to the function. Let's set it to minus 2! We also make more x points (like, many!) so it is easier to see:

```
x<-seq(from = -5, to = 5, by = 0.1)
a<- -2
y<-a+x^2
plot(x,y)
segments(0,-30,0,30, lty=3)
segments(-30,0,30,0,lty=3)
```



What happens when we give the quadratic part a slope b? To see better what happens, we'll add it to the previous plot (we have to replot it, though).

```
plot(x,y)
a<- -2
b<-3
y<-a+b*x^2
points(x,y, pch=19, col="red")
segments(0,-30,0,30, lty=3)
segments(-30,0,30,0,lty=3)
```

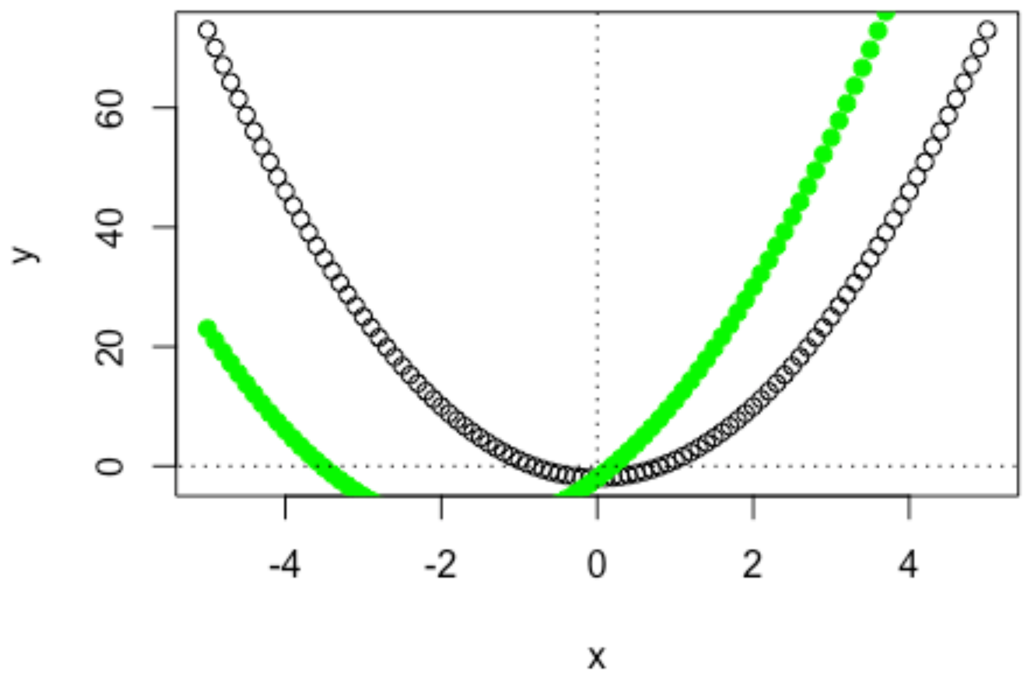


It gets steeper. So, now you can guess what happens when you make the slope smaller than 1, or even smaller than 0. It's an exercise, actually.

I want to take this one step further. What happens when we add a non-quadradic effect?

$$y = a + (b1 * x) + (b2 * x^2)$$

```
plot(x,y)
a<- -2
b1<- 10
b2<-3
y<-a+b1*x+b2*x^2
points(x,y, pch=19, col="green")
segments(0,-100,0,100, lty=3)
segments(-100,0,100,0,lty=3)
```



It moves the whole curve! How cool is that?

Exercises

1) The response variable (y) is body mass in kg of Romanian Longhorn dragons. The explanatory variable (x) is the amount of sheep they've eaten in the last hour. The slope is 12. You don't know the intercept (but you don't need to know it to answer this question). How much kg does a Longhorn dragon gain on average per sheep eaten in the last hour?

2) The slope is 8, and the intercept is 0. This relationship describes the number of new moth species in my garden per day of catching in July. How many species can I expect to have seen after catching 10 days in July?

3) The quadratic relationship y~-1+2x-0.15x^2 describes how a phoenix's' reproductive success first improves with age (in years), and then declines as they senesce. At what age are these birds in their prime year? How many offspring do they produce at their prime, before reproductive output starts declining again?

4) You tested for the relationship between nature reserve area (explanatory variable, in ha), and species richness (in number of species). You find a linear slope b1 of 2, a quadratic slope b2 of -0.08, and an intercept of -1. Describe the change of the number of species seen as reserve area size changes.

5) Finish the Linear Functions – A Primer – Exit Quiz before proceeding to the next lecture. It’s on blackboard: PG Life Science Core Skills Modules 2020-2021, ho.07.SwS.Linear Functions

Statistics with Sparrows - 08

Julia Schroeder

July 2020

HO 08

Starting off with linear models

Learning aims

- Getting a feel for the code used for linear models
- Exploring the structure of models, and the structure of model objects, in R
- Understanding how to access the different statistical parameters of models
- Starting to explore correlations and covariances linked with simple linear models

Let's give this a go:

First, we will try to re-do what I've shown you in the lecture. We will re-create the vectors x and y, and then run the model. I want you to use the output (in your R) to see if you can find similar estimates to those we "guesstimated" during the lecture. It is really important now that you not only run the code in your own machine, but also, that you inspect the output, be it red (errors or warning messages) or not red. You need to look at all the numbers, put them into perspective of each other, and think about what each number means. I'll talk you through it but you need to start deducing this by yourself at this point:

```
rm(list=ls())
x<-c(1,2,3,4,8)
y<-c(4,3,5,7,9)
x

## [1] 1 2 3 4 8

mean(x)

## [1] 3.6

var(x)

## [1] 7.3

y

## [1] 4 3 5 7 9
```

```
mean(y)

## [1] 5.6

var(y)

## [1] 5.8
```

So this is how we made the data for our model. Now we run the linear model. As linear models are the most important function for any sort of statistic, it's a really important function to know it by heart. It's `lm()`. Accessing the help function with this gives:

```
?lm
```

It seems we need to pass a sort of formula to R, that's the model equation! Remember:

$$y = b_0 + b_1 * x$$

Where b_0 is the intercept, b_1 is the slope for the explanatory variable x, and y is the response variable. In R, we specify that in a very specific way. First, we give the response variable, y, then we use this sign "~" (called the tilde). Make sure you find it on your keyboard - you'll be using it **a lot** from here on in. If you can't find it, google where it is on your keyboard. Then, after the tilde comes our explanatory variable. We don't have to specify the intercept b_0 (at least not for most models), it is estimated automatically by default. For the slope b_1 we need to tell R what explanatory variable it is connected to. So we just say `y~x` in this case. If we have a data frame, we have to specify the data frame using the `data=` argument. There's an argument for `weights` that we ignore for now, as the one for `subset=`. There is one for the `na.action=`,this is self-explanatory and I want you to play around with that a bit later on. There will be little if any need to use any of the other arguments, but don't let that stop your curiosity. Yet, it won't be part of this course. It's all explained in the "Details" section of the help text. Then, under the "Details" section (you need to scroll down), there's a "Value" section, that explains what the function returns. It returns an object of class "lm", and then it goes on to say what sort of values you can access from that. There's `coefficients` - that sounds like the parameter estimates we're after. The next one, `residuals` should be clear to you. `fitted values` is interesting, and we'll look at that further below. Then we can glance of the rest. Scroll further down, until you reach "See Also". There's the `summary.lm` and `anova.lm` command, also further generic functions, and `predict.lm`. And lots more if you want to dive right in, but I'd say reserve that for later. The examples are interesting, and I urge you to run them. Very much like this hand-out, you can copy-paste the code and it should run, then inspect the output and see if you understand it. I recommend doing this after you've done this hand out.

Let's put this to work:

```
model1 <- (lm(y~x))
model1

##
## Call:
## lm(formula = y ~ x)
```

```
##
## Coefficients:
## (Intercept)            x
##      2.6164       0.8288
```

```
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       1       2       3       4       5
##  0.5548 -1.2740 -0.1027  1.0685 -0.2466
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6164     0.8214   3.185   0.0499 *
## x             0.8288     0.1894   4.375   0.0221 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.024 on 3 degrees of freedom
## Multiple R-squared:  0.8645, Adjusted R-squared:  0.8193
## F-statistic: 19.14 on 1 and 3 DF,  p-value: 0.0221
```

```
coefficients(model1)
```

```
## (Intercept)           x
##   2.6164384   0.8287671
```

```
resid(model1)
```

```
##          1          2          3          4          5
##  0.5547945 -1.2739726 -0.1027397  1.0684932 -0.2465753
```

```
mean(resid(model1))
```

```
## [1] 1.111578e-17
```

```
var(resid(model1))
```

```
## [1] 0.7859589
```

```
length(resid(model1))
```

```
## [1] 5
```

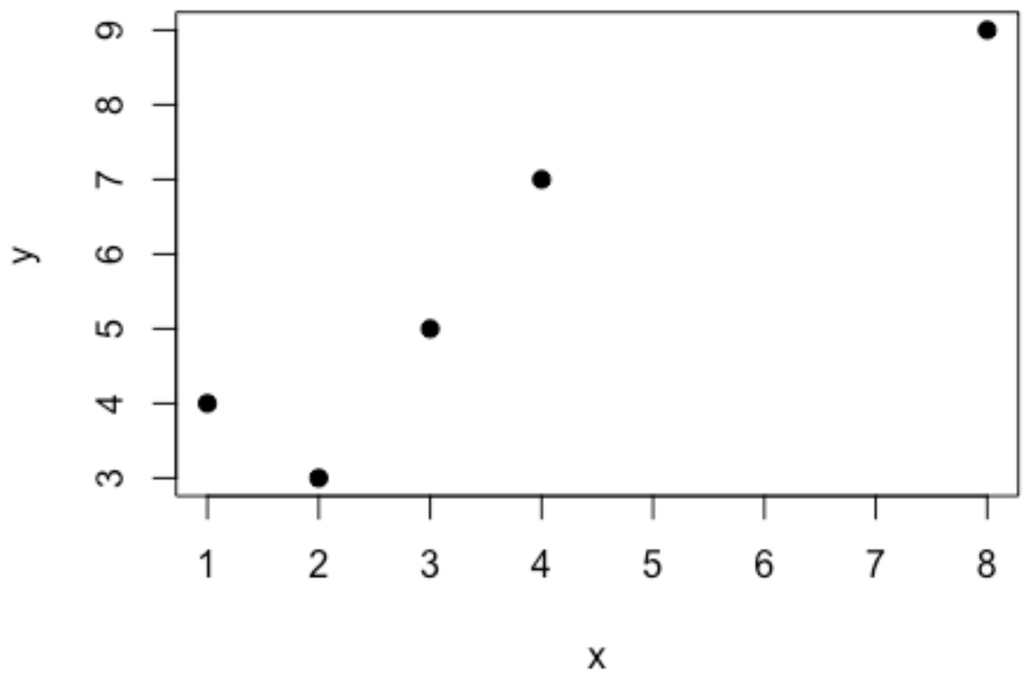
I think, give you’ve been through the lecture, you will now see how we can access our primary statistics, the intercept and the slope for x.

However, there is clearly more - the most comprehensive thing came through with the `summary()` function:

```
summary(model1)

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       1       2       3       4       5
##  0.5548 -1.2740 -0.1027  1.0685 -0.2466
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.6164     0.8214   3.185   0.0499 *
## x             0.8288     0.1894   4.375   0.0221 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.024 on 3 degrees of freedom
## Multiple R-squared:  0.8645, Adjusted R-squared:  0.8193
## F-statistic: 19.14 on 1 and 3 DF,  p-value: 0.0221
```

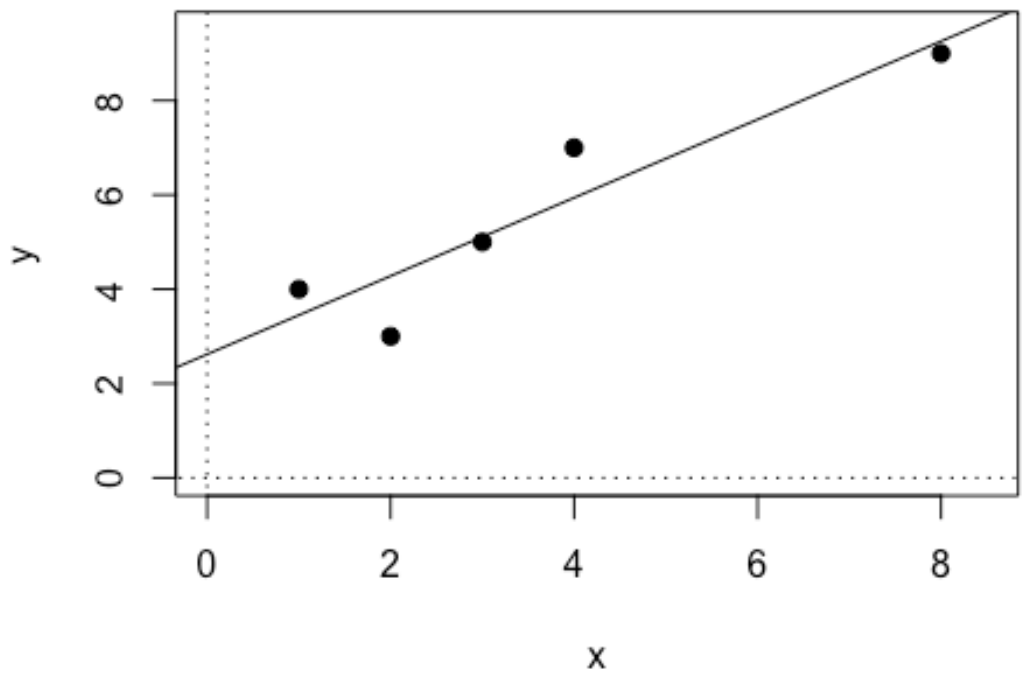
It tells us the residuals. Then the coefficients, and their standard errors, and associated t-values and p-values. Then there's a row of significance codes, a row about the residual SE, the dfs, and some stuff on R^2 and F statistics that we haven't spoken about just yet. But we will. For now, we stick to the Coefficents. As they are estimated from data, we can calculate a standard error around them. And if their 95%CI (or, as we know, 2xSE) does not span 0, we can assume they are not statistically significant. For example, the slope is 0.829 and its SE of 0.19. Twice the SE plus minus the estimate (2x0.19+0.829 and 2x0,19-0.089) does not span zero thus the slope is statistically significant. That's confirmed by the t-value, and the p-value. Later more about this, but it's good to see. Also, there are 3 degrees of freedom. That makes sense because we have 5 data points, minus 2 for the two coefficients (intercept and slope) that we estimated. So far, I hope you have understood everything. Now, let's have a look at the line that this model predicts. We can plot it with abline into the plot with the datapoints. Note, that when I plot a model, I use the code y ~ x to denotate that y is the response variable, and x is the explanatory variable. That makes r plot y on the y axis (where the response belongs), and x on the x-axis (again, where it belongs).

```
plot(y~x, pch=19)
```



If I plot it with a comma (,), then it will be plotted such that the first variable will be interpreted as x, and the second one as y. It's a bit like with the t-test. Try it! To make sure you're not confused it is really good to stick to the tilde as the response vs explanatory variable stuff is always clear - response is always to the left of the tilde. I plot the y and x graph, and I change the x- and y-axes length (xlim and ylim) so that they include zero, because I want to see the intercept when I plot the line through my data. I do that with `abline()`. I know now the coefficients, so I can put that into the abline function. I bring the coefficients up again so I don't need to scroll I also add the cartesian coordinate axes explicitely so I'll be better able to assess the intercept:

```
plot(y~x, pch=19, xlim=c(0,8.5), ylim=c(0,9.5))
segments(0,-30,0,30, lty=3)
segments(-30,0,30,0,lty=3)
coefficients(model1)

## (Intercept)           x
##   2.6164384   0.8287671

abline(2.62, 0.83)
```



That looks like a neatly fitted line indeed. The intercept looks like somewhere between 2 and 3, so 6.6 seems like a good enough guess. And when you squint, you can see about what the slope would be (remember: one to the right, and then the slope is how much you have to go up until you hit the line).

Now, we can move on and use a somewhat larger dataset. We can simulate that, and even make up the data that we want. Let's see how that works. We will first create an x variable that goes from -10 to +10. Then we decide on a random value for the slope, maybe -0.2. We also choose an intercept: 7.1. Then we simulate numbers using the normal number generator in R:

```
x<-seq(from=-10, to=10, by=0.2)
x

##   [1] -10.0  -9.8  -9.6  -9.4  -9.2  -9.0  -8.8  -8.6  -8.4  -8.2  -8.0  -
7.8
##  [13]  -7.6  -7.4  -7.2  -7.0  -6.8  -6.6  -6.4  -6.2  -6.0  -5.8  -5.6  -
5.4
##  [25]  -5.2  -5.0  -4.8  -4.6  -4.4  -4.2  -4.0  -3.8  -3.6  -3.4  -3.2  -
3.0
##  [37]  -2.8  -2.6  -2.4  -2.2  -2.0  -1.8  -1.6  -1.4  -1.2  -1.0  -0.8  -
0.6
##  [49]  -0.4  -0.2   0.0   0.2   0.4   0.6   0.8   1.0   1.2   1.4   1.6
```

```
1.8
##  [61]   2.0   2.2   2.4   2.6   2.8   3.0   3.2   3.4   3.6   3.8   4.0
4.2
##  [73]   4.4   4.6   4.8   5.0   5.2   5.4   5.6   5.8   6.0   6.2   6.4
6.6
##  [85]   6.8   7.0   7.2   7.4   7.6   7.8   8.0   8.2   8.4   8.6   8.8
9.0
##  [97]   9.2   9.4   9.6   9.8  10.0
```

```
y<- 7.1-0.2 * x
y
```

```
##   [1] 9.10 9.06 9.02 8.98 8.94 8.90 8.86 8.82 8.78 8.74 8.70 8.66 8.62 8.5
8 8.54
##  [16] 8.50 8.46 8.42 8.38 8.34 8.30 8.26 8.22 8.18 8.14 8.10 8.06 8.02 7.9
8 7.94
##  [31] 7.90 7.86 7.82 7.78 7.74 7.70 7.66 7.62 7.58 7.54 7.50 7.46 7.42 7.3
8 7.34
##  [46] 7.30 7.26 7.22 7.18 7.14 7.10 7.06 7.02 6.98 6.94 6.90 6.86 6.82 6.7
8 6.74
##  [61] 6.70 6.66 6.62 6.58 6.54 6.50 6.46 6.42 6.38 6.34 6.30 6.26 6.22 6.1
8 6.14
##  [76] 6.10 6.06 6.02 5.98 5.94 5.90 5.86 5.82 5.78 5.74 5.70 5.66 5.62 5.5
8 5.54
##  [91] 5.50 5.46 5.42 5.38 5.34 5.30 5.26 5.22 5.18 5.14 5.10
```

Now we can use that data to run a linear model. We expect the slope to be -0.2, and the intercept to be 7.1, of course:

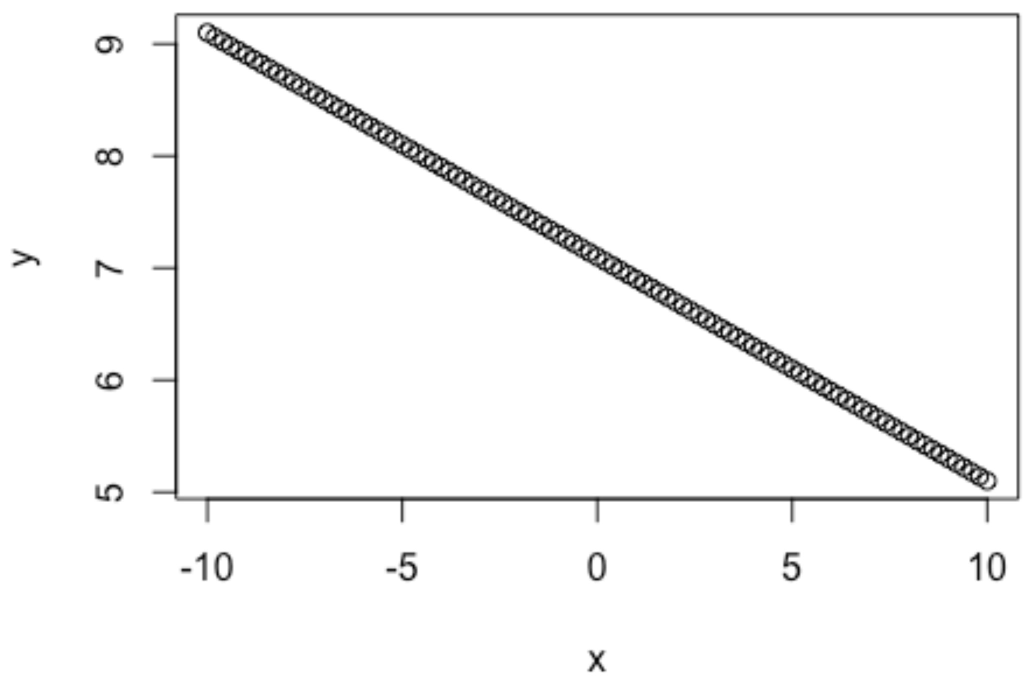
```
summary(lm(y~x))
```

```
## Warning in summary.lm(lm(y ~ x)): essentially perfect fit: summary may be
## unreliable
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.385e-15 -4.873e-16 -1.301e-16  1.431e-16  1.882e-14
##
## Coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept)  7.100e+00  1.955e-16  3.632e+16   <2e-16 ***
## x           -2.000e-01  3.352e-17 -5.966e+15   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.964e-15 on 99 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1 
## F-statistic: 3.56e+31 on 1 and 99 DF,  p-value: < 2.2e-16
```

Uuups. While the estimates are excellent, we get an odd error message. It says the fit is essentially perfect. What does that mean? Maybe it will clear up when we plot the data (plotting data when in trouble is always a good way to find the mistake):

```
plot(y~x)
```



Now it's clear - the data is too perfect. There is no uncertainty. So the standard error is super small, as are the residuals. Both are practically zero.

Well, we can try to simulate some uncertainty to our data. We'll use a random number generator to add the residual error when we create our y values. The function `runif()` does that - try `?runif` for more information. Basically, it creates random numbers between 0 and 1 by default, but you can set that interval to one of your choice.You also must specify the length of the vector you want to create. As you remember our residuals are the same number of values as x and y, so that's what we need to do.

Now, this time, we expect out coefficients to not be completely perfect, but still in the ballpark. Note: your own values should be different from those in the hand out because it's a random number generator!

```r
y<- 7.1-0.2 * x + runif(length(x))
summary(lm(y~x))

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56360 -0.21841 -0.03266  0.23890  0.52249
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.615939   0.029828  255.32   <2e-16 ***
## x           -0.211210   0.005116  -41.29   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2998 on 99 degrees of freedom
## Multiple R-squared:  0.9451, Adjusted R-squared:  0.9446
## F-statistic:  1705 on 1 and 99 DF,  p-value: < 2.2e-16
```

That looks much better.

Exercises

(no uploads)

1) Run the code you find under "examples" in the help file for `lm`
2) Look through the helpfile for `runif` and see if you understand it.
3) Increase and decrease the size of the random noise you add to the data, and see what it does to your linear model estimates! This will help you better understand the concepts behind it, and also help you to better interpret results. Further, I hope this will help you to use this tool (making up data where you know what the result should be like, and then analysing it) to better understand statistics.
4) Run the example code from the helpfile for `runif` and see if you can understand it.

Statistics with Sparrows - 09

Julia Schroeder

August 2020

HO 09

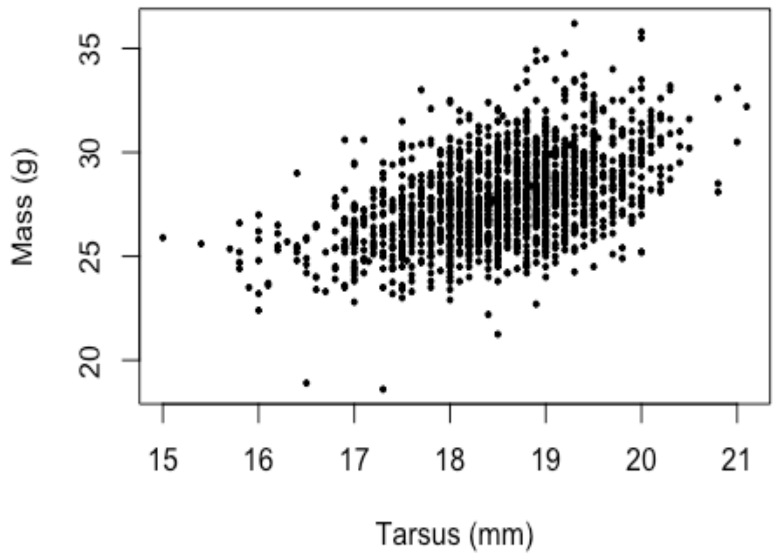
Linear models with one predictor

Again: housekeeping!

```
rm(list=ls())

d<-read.table("SparrowSize.txt", header=TRUE)
```

Linear models include things like correlations. Correlations are associations between two variables. For instance, sparrows that are larger, may also be heavier:

```
plot(d$Mass~d$Tarsus, ylab="Mass (g)", xlab="Tarsus (mm)", pch=19, cex=0.4)
```

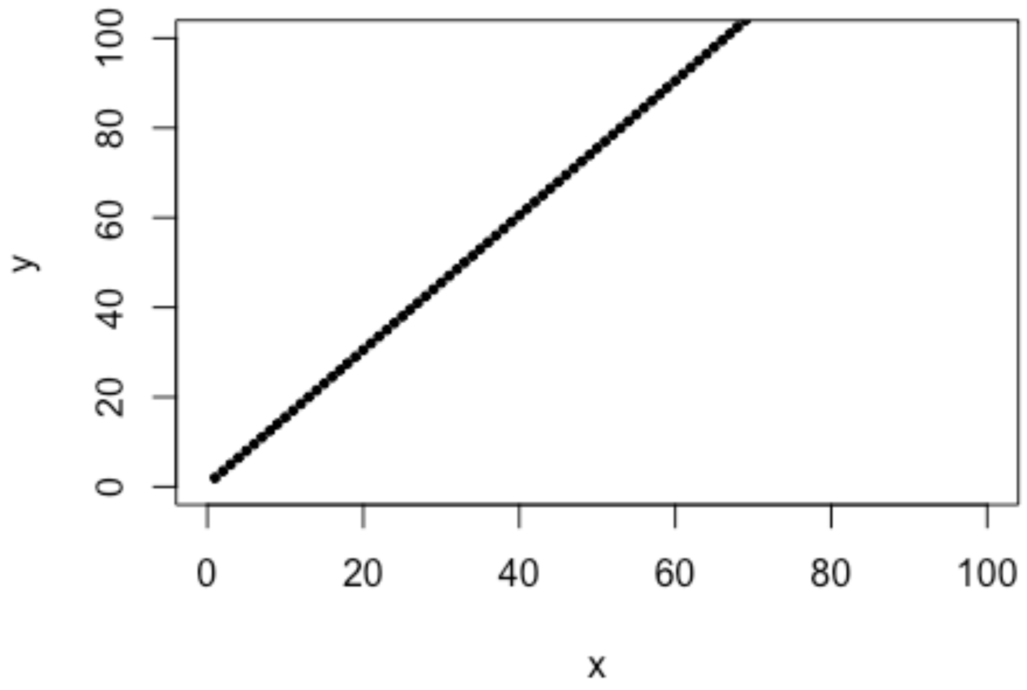


So, there clearly seems to be a linear relationship here. A quick brush-up on things related to linear relationships. How can we describe a line in mathematical terms?

$$y = a + bx$$

Y are the y-coordinates, x the x-coordinates. a is the intercept, that's where the line crosses the y-axis. b is the slope, that defines how steep the line is, and whether it is negative (decrease) or positive (increase). The slope can be interpreted as increasing by b for each 1 x. You can then find all y and x that fit those criteria, and they will all fall on a line. I can demonstrate that easily in R:

```
x<-c(1:100)
a<-0.5
b<-1.5
y<-b*x+a
plot(x,y, xlim=c(0,100), ylim=c(0,100), pch=19, cex=0.5)
```



In statistics, we use such a line equation to describe the linear association between two variables. We use a slightly different equation, though:

$$y_i = b_0 + b_1 x_i + \varepsilon_i$$

First, we note the *i*'s at the right hand bottom of *y* and *x*. This informs us denote that there is a population of data running from 1 to i observations.

In our case, y1 is

```
head(d$Mass)
```

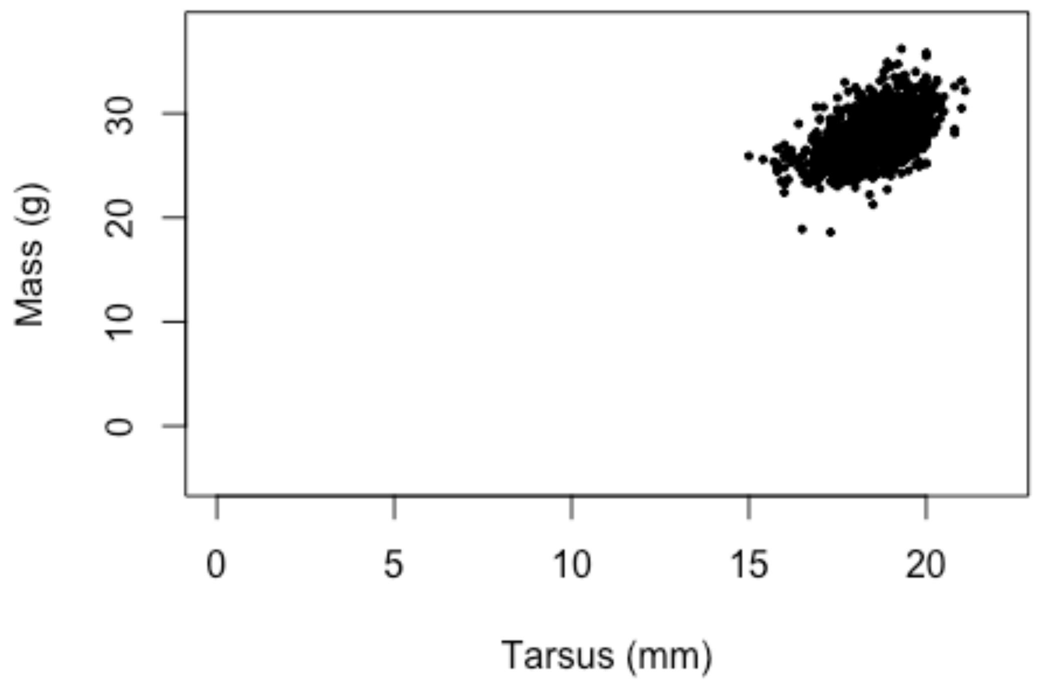
```
## [1] 29.4 31.6 29.9 31.6 31.0 28.1

d$Mass[1]

## [1] 29.4
```

the maximum n, and respective y_n is

```
length(d$Mass)

## [1] 1770

d$Mass[1770]

## [1] 33

tail(d$Mass)

## [1] 24.5 25.9 27.1 25.1 26.1 33.0
```

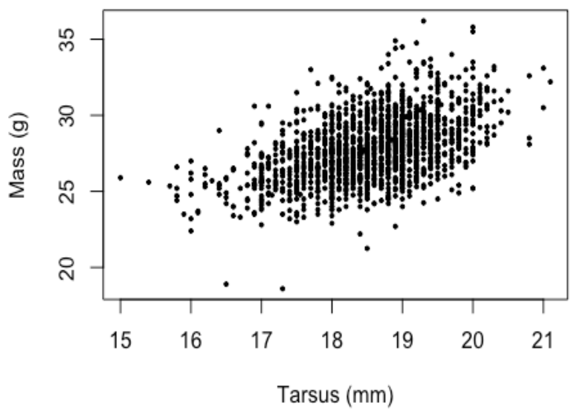
Then, there are b_0, which is the intercept, and b_1, which is the slope. These stay the same for all *i*s. These two are the parameters we want to estimate, to get a grip at the association between mass and tarsus. From looking at the plot, we can give it a guess (after we've plotted it such that we can see the 0 of the *x*axis, which is where the intercept is evaluated):

```
plot(d$Mass~d$Tarsus, ylab="Mass (g)", xlab="Tarsus (mm)", pch=19, cex=0.4, y
lim=c(-5,38), xlim=c(0,22))
```



Squinting a bit at it makes me think that the intercept is somewhere between -5 and 10. The slope is more difficult. We can look at the difference it makes for 5mm - that's about the difference between 20 and 30g, maybe a bit less. Very roughly. It might be easier to estimate on a plot that's zoomed in a bit:

```r
plot(d$Mass~d$Tarsus, ylab="Mass (g)", xlab="Tarsus (mm)", pch=19, cex=0.4)
```



Yes, so with a squint at the plot I would approximate 10g difference for each 5mm in Tarsus. Scaling down to 1mm Tarsus, that gives a change in mass of 10/5 = 2g. So, we can predict that our equation should look like this, where b_0 = 5-ish and b_1=2. We insert that into our equation:

$$y_i = b_0 + b_1 x_i + \varepsilon_i$$

$$y_i = 5 + 1.6x_i + \varepsilon_i$$

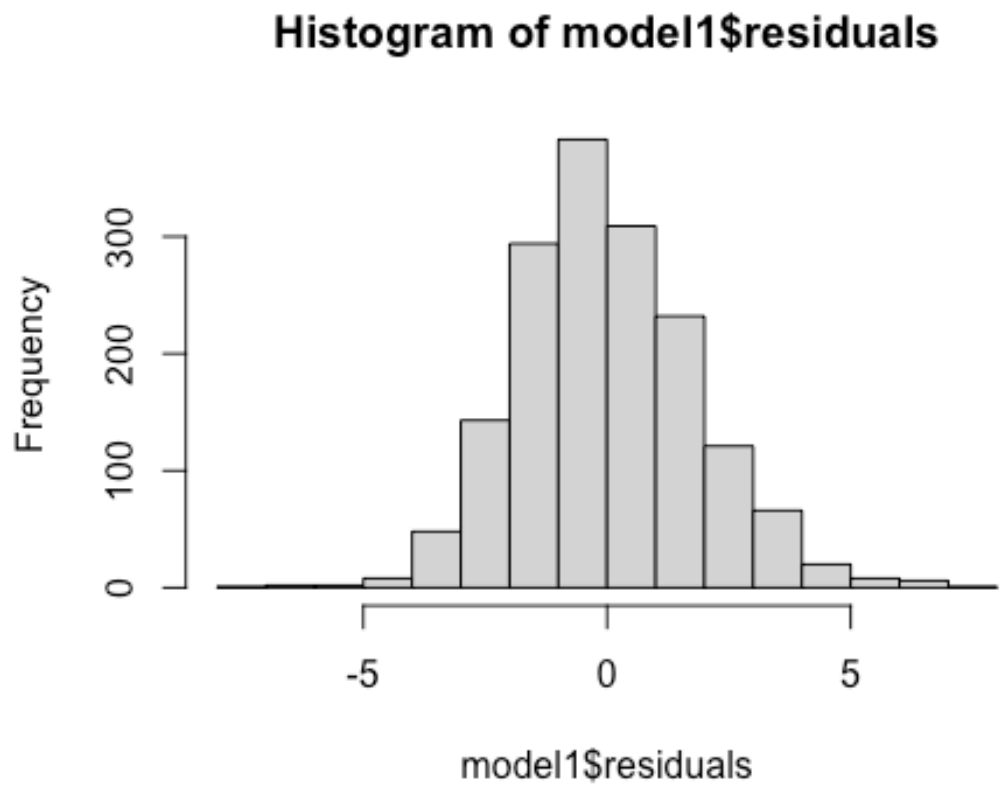
Umm, there's this odd epsilon that we've conveniently ignored until now. What's with this? Looking back at our plot, it is clear that our data does also do other things. We not only want to quantify the direction and steepness of the association, but also the spread. THis spread is the ERROR - the noise around the straight line. That's the epsilon - there is one error term, or *residual* for each observation. This residual describes how far the point is away from the actual line. The line is fitted such that these residuals take on the smallest possible value. To be completely honest, we don't take the simple distance from the line to the point. No, we use the distance that is vertical (parralel to y), and then, we take the sum of the squares of them, - the sum of the squared residuals. Then, we wiggle the line so that the sum of squares (sounds familiar?), takes on the smallest value possible for these data points. We call this the *least square* method. $\sum(y_i - \hat{y}_i)$, where $\hat{y}_i$ is the *i-th* y point that one calculates from the equation of the fitted line, with the respective x_i data point. This is all tremendously important. If you struggle understanding this, do ask, or use the Wednesday to ask your question and revisit this.

Now, we'll harness the powers of R to see how this looks like in real life. First, of course, we strip the dataset to one that does not contain missing values:

```
d1<-subset(d, d$Mass!="NA")
d2<-subset(d1, d1$Tarsus!="NA")
length(d2$Tarsus)

## [1] 1644

model1<-lm(Mass~Tarsus, data=d2)
summary(model1)

##
## Call:
## lm(formula = Mass ~ Tarsus, data = d2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.7271 -1.2202 -0.1302  1.1592  7.5036
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.83246    0.98195    5.94 3.48e-09 ***
## Tarsus       1.18466    0.05295   22.37  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.841 on 1642 degrees of freedom
## Multiple R-squared:  0.2336, Adjusted R-squared:  0.2332
## F-statistic: 500.6 on 1 and 1642 DF,  p-value: < 2.2e-16
```

Let's examine this result. First, R tells us what we asked it to do, the call. That's easy. Then, it tells us about the residuals - you should know what this second report means. It tells us about the distribution of the residuals. We can actually access the residuals:

```
hist(model1$residuals)
```



```
head(model1$residuals)

##         1         2         3         4         5         6
## 1.1774836 3.4959507 1.4405508 3.2590180 2.5405512 0.9436773
```

Remember this for later.

Now on the the coefficients of the equation. There are *Parameter estimate*, *Standard Errors*, *t-values*, and *p-values* of each of the *b*s - the intercept (b_0) and the slope for Tarsus ($b1$).

We've been somewhat wrong with our slope guesstimate, it was even smaller than we thought. But the intercept was well guessed. We can look at the standard errors, and t-values (do you know now where these come from?). The data for the intercept tells us it's statistically significantly different from zero. That's not really interesting, because we are not interested in the body mass of a bird with a tarsus of 0 length. Most of the time, when you do statistics, the intercept is not interesting. However, it pays if you recall what it really is. Later, we will do this analysis with z-standardized tarsus, and then the intercept is all of the sudden quite interesting. The slope is statistically significantly different from zero, with p<0.001. That's cool. It's in the ballpark of 1.2 - for 1 mm increase in tarsus, birds are about 1g heavier. Good. What's the rest of the stuff reported here?

There are the degrees of freedom. Let's check:

So, there are 1644 observations, and the *df*s are 1642. That means, R estimated 2 parameters, one for the intercept, and one for the slope of tarsus. The standard error of the residuals is also given.

But, there is more. What's the R-squared stuff? We just look at the first one - that' s 0.23. That means, and here we finally get to the spread, that 23% of the variance in mass is explained by variation in tarsus. If this value is 1 (100%), we'd see a straight line. We can test this using the x/y data we generated before, with an intercept of 0.5, and a slope of 1.5, and a perfect fit - the R-squared thus there should be 1:

```
model2<-lm(y~x)
summary(model2)

## Warning in summary.lm(model2): essentially perfect fit: summary may be
## unreliable

##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.372e-13 -1.237e-15  1.230e-15  3.407e-15  2.160e-14
##
## Coefficients:
##              Estimate Std. Error   t value Pr(>|t|)
## (Intercept) 5.000e-01  2.891e-15 1.729e+14   <2e-16 ***
## x           1.500e+00  4.971e-17 3.018e+16   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.435e-14 on 98 degrees of freedom
## Multiple R-squared:      1,	Adjusted R-squared:      1
## F-statistic: 9.106e+32 on 1 and 98 DF,  p-value: < 2.2e-16
```

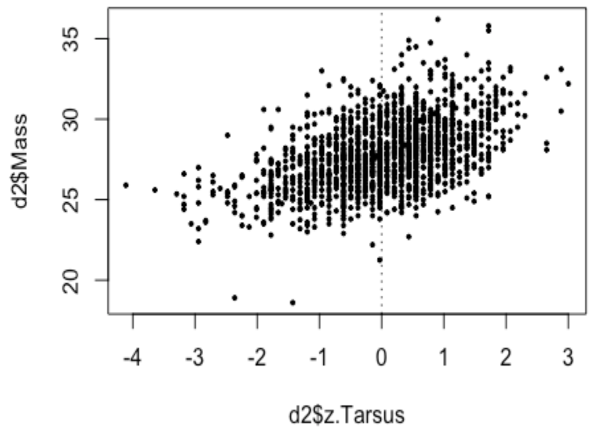
Back to the sparrow data. We learned before about z-scores, and standardization. And I've said it is commonplace in statistics to z-standardize the covariate - that is, the continous predictor variable. Why? Let's run the model with z-scores of Tarsus instead of Mass:

```
d2$z.Tarsus<-scale(d2$Tarsus)
model3<-lm(Mass~z.Tarsus, data=d2)
summary(model3)
```

```
##
## Call:
## lm(formula = Mass ~ z.Tarsus, data = d2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.7271 -1.2202 -0.1302  1.1592  7.5036
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 27.77895    0.04539  611.94   <2e-16 ***
## z.Tarsus     1.01596    0.04541   22.37   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.841 on 1642 degrees of freedom
## Multiple R-squared:  0.2336, Adjusted R-squared:  0.2332
## F-statistic: 500.6 on 1 and 1642 DF,  p-value: < 2.2e-16
```

When we now look at the estimates, what does the Intercept reflect? Remember, it is where the regression line crosses 0 on the x-axis. It becomes more apparent when we plot it:

```
plot(d2$Mass~d2$z.Tarsus, pch=19, cex=0.4)
abline(v = 0, lty = "dotted")
```

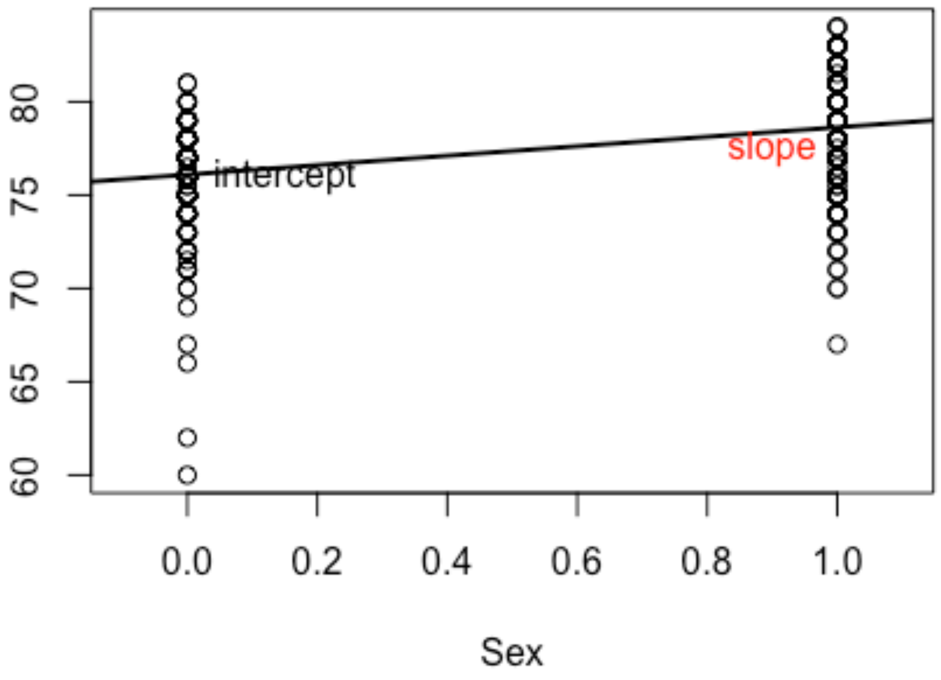


And what does the slope reflect now? Refer back to the z-scores we discussed a while ago.

Ok, let’s see…

```
head(d)

##   BirdID Cohort CaptureDate CaptureTime Year Tarsus Bill Wing Mass Sex  Se
x.1
## 1   4401   1991   21-Jun-00        <NA> 2000   18.9   NA   82 29.4   1    m
ale
## 2   4401   1991   02-Oct-00        <NA> 2000   18.8   NA   79 31.6   1    m
ale
```

```
## 3   4405   1994   20-Jun-00        <NA> 2000   19.1   NA   77 29.9   0 fem
ale
## 4   4405   1994   04-Oct-00        <NA> 2000   19.0   NA   78 31.6   0 fem
ale
## 5   4405   1994   07-Oct-00        <NA> 2000   19.1   NA   77 31.0   0 fem
ale
## 6   4409   1994   23-Mar-00        <NA> 2000   18.0   NA   76 28.1   1   m
ale

str(d)

## 'data.frame':    1770 obs. of  11 variables:
##  $ BirdID     : int  4401 4401 4405 4405 4405 4409 4409 4409 4409 4409 ...
##  $ Cohort     : int  1991 1991 1994 1994 1994 1994 1994 1994 1994 1994 ...
##  $ CaptureDate: chr  "21-Jun-00" "02-Oct-00" "20-Jun-00" "04-Oct-00" ...
##  $ CaptureTime: chr  NA NA NA NA ...
##  $ Year       : int  2000 2000 2000 2000 2000 2000 2000 2000 2001 2001 ...
##  $ Tarsus     : num  18.9 18.8 19.1 19 19.1 ...
##  $ Bill       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Wing       : num  82 79 77 78 77 76 76 73 79 77 ...
##  $ Mass       : num  29.4 31.6 29.9 31.6 31 ...
##  $ Sex        : int  1 1 0 0 0 1 1 1 1 1 ...
##  $ Sex.1      : chr  "male" "male" "female" "female" ...

d$Sex<-as.numeric(d$Sex)
plot(d$Wing ~ d$Sex, xlab="Sex", xlim=c(-0.1,1.1), ylab="")
abline(lm(d$Wing ~ d$Sex), lwd = 2)
text(0.15, 76, "intercept")
text(0.9, 77.5, "slope", col = "red")
```



Can you explain how the t-test, that as we know, tests for a statistically different from zero difference between two means, can be used to test for linear models?

```
d4<-subset(d, d$Wing!="NA")
m4<-lm(Wing~Sex, data=d4)
t4<-t.test(d4$Wing~d4$Sex, var.equal=TRUE)
summary(m4)

## 
## Call:
## lm(formula = Wing ~ Sex, data = d4)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -16.0961  -1.0961  -0.0961   1.3683   5.3683 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 76.09611    0.07175 1060.50   <2e-16 ***
## Sex          2.53562    0.09998   25.36   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
```

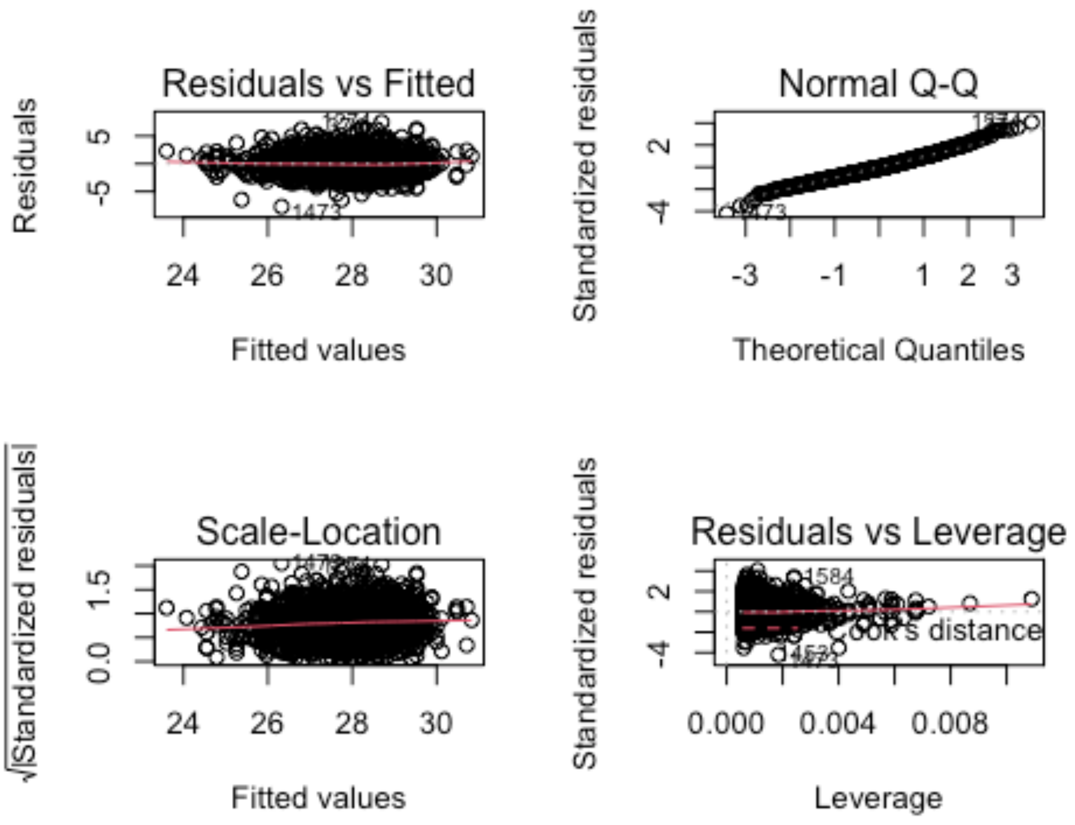
```
## Residual standard error: 2.057 on 1693 degrees of freedom
## Multiple R-squared:  0.2753, Adjusted R-squared:  0.2749
## F-statistic: 643.1 on 1 and 1693 DF,  p-value: < 2.2e-16
```

```
t4
```

```
##
##  Two Sample t-test
##
## data:  d4$Wing by d4$Sex
## t = -25.36, df = 1693, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.731727 -2.339518
## sample estimates:
## mean in group 0 mean in group 1
##        76.09611        78.63173
```

Before we end this, we have to do some linear model diagnostics. This is to make sure the assumptions that one has to make for a linear model to hold, are actually met. The most important assumption of linear models is that the residuals (!) are normally distributed. We can only test for that after we've fitted the model, because that's how we get the residuals.

```
par(mfrow=c(2,2))
plot(model3)
```

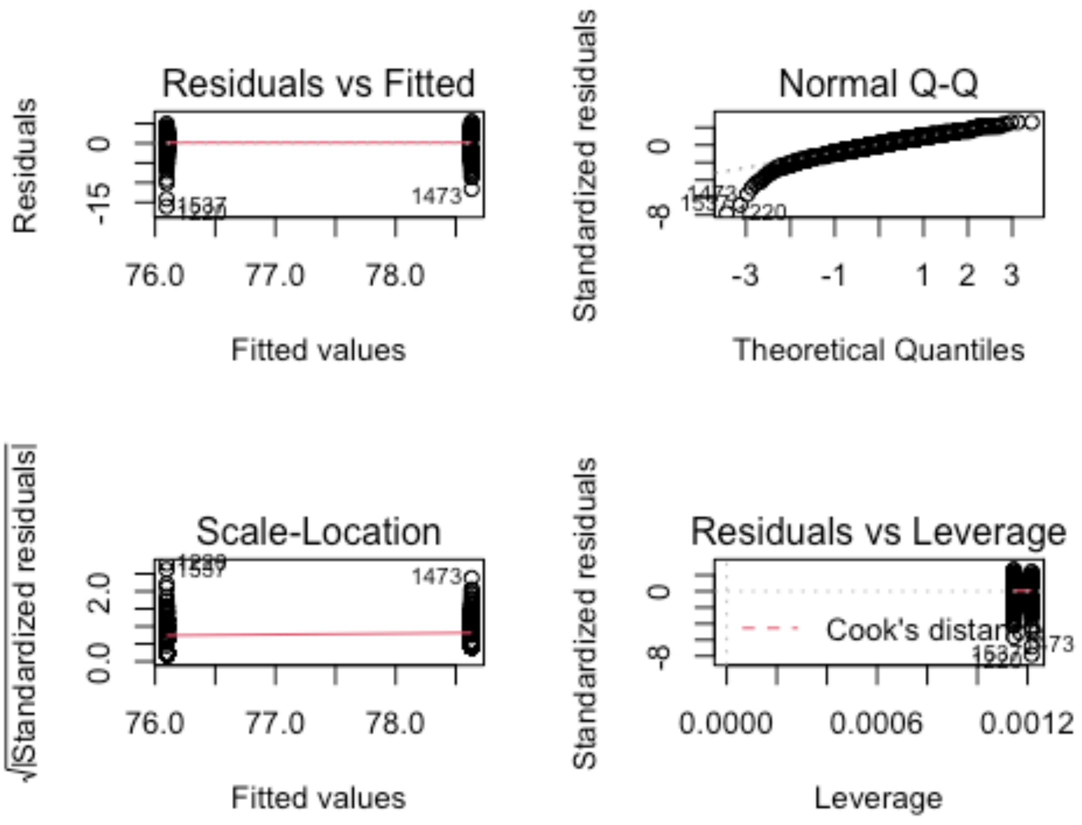


Here, we want to look at the top left plot. It shows the residuals (the distances between the values and the calculated regression line) on the y-axis, and the fitted values - those y's that you get when you calculate using the regression equation with b_0 and b_1. We want these to be distributed roughly randomly. Like "stars in the sky". This is an ok plot for this. We do not want to see patterns such as residuals getting larger with increasing fitted values. That's why R provides us with a nice red line, that shows some sort of deviations from y=0 (which is a super small residual).

The next plot right of the residual plot, the Q-Q plot, shows the standardised residuals (you should know by now what that means) plotted against the quantiles they are supposed to fall into, assuming the residuals are normally distributed. You should know what this sentence means by now. This plot should look like a straight line, and we're happy with what we see here.

Compare it with our model 4:

```r
par(mfrow=c(2,2))
plot(m4)
```



Can you explain why these two top plots test for the normality of residuals?

The bottom right plot shows the square root of the residuals - it's one you can look at, too. And the bottom left one is an interesting one - it shows the residuals in relationship to their leverage. How important some data points are in relationship to others.

Excercises:

1) Discuss the questions above with your group, until you find satisfactory answers.

2) Run a linear model, where you test the hypothesis that sparrows with bigger bills can eat more. The prediction is that the larger the bill, the heavier the sparrow.

Detail what your explanatory and what your response variable is. Write a short (1A4) report on methods and results per group. Before you go into the linear model, you should first describe your data, say how many sparrows, how many females and males, whether there is a difference in your response between the sexes. If that difference is meaningful, you should test the sexes separately. Write this section as you would write it for a scientific article. Discuss in the group what should go into the report, and what not. Submit one report per group.

Stats with Sparrows - 11

Julia Schroeder

August 2020

HO 11

This practical will look at fitting models in R that use multiple variables to explain the response variable.

Housekeeping!

```
rm(list=ls())
```

Daphnia growth

We will stray from the sparrows and use a dataset on the growth of Daphnia populations, looking at the rate of growth in water containing four different detergents and using individuals of three different clones. There is also a variable on the water source used but we will not be adding it to the analysis here. We will load the data and then look at the data distribution using two box and whisker plots (Figure 1)

```
daphnia <- read.delim("daphnia.txt")
summary(daphnia)

##   Growth.rate       Water              Detergent          Daphnia
##  Min.   :1.762   Length:72          Length:72          Length:72
##  1st Qu.:2.797   Class :character   Class :character   Class :character
##  Median :3.788   Mode  :character   Mode  :character   Mode  :character
##  Mean   :3.852
##  3rd Qu.:4.807
##  Max.   :6.918

head(daphnia)

##   Growth.rate Water Detergent Daphnia
## 1    2.919086  Tyne    BrandA  Clone1
## 2    2.492904  Tyne    BrandA  Clone1
## 3    3.021804  Tyne    BrandA  Clone1
## 4    2.350874  Tyne    BrandA  Clone2
## 5    3.148174  Tyne    BrandA  Clone2
## 6    4.423853  Tyne    BrandA  Clone2

str(daphnia)
```

```
## 'data.frame':	72 obs. of  4 variables:
##  $ Growth.rate: num  2.92 2.49 3.02 2.35 3.15 ...
##  $ Water      : chr  "Tyne" "Tyne" "Tyne" "Tyne" ...
##  $ Detergent  : chr  "BrandA" "BrandA" "BrandA" "BrandA" ...
##  $ Daphnia    : chr  "Clone1" "Clone1" "Clone1" "Clone2" ...
```

We go through the list we've gone through in the lecture. First things first!

1: Outliers

We check for potential outliers in x and y. From the summary data, we can see that the categories have sufficient samples size - this is a homogeneous dataset!

```r
par(mfrow = c(1, 2))
plot(Growth.rate ~ as.factor(Detergent), data=daphnia)
plot(Growth.rate ~ as.factor(Daphnia), data = daphnia)
```

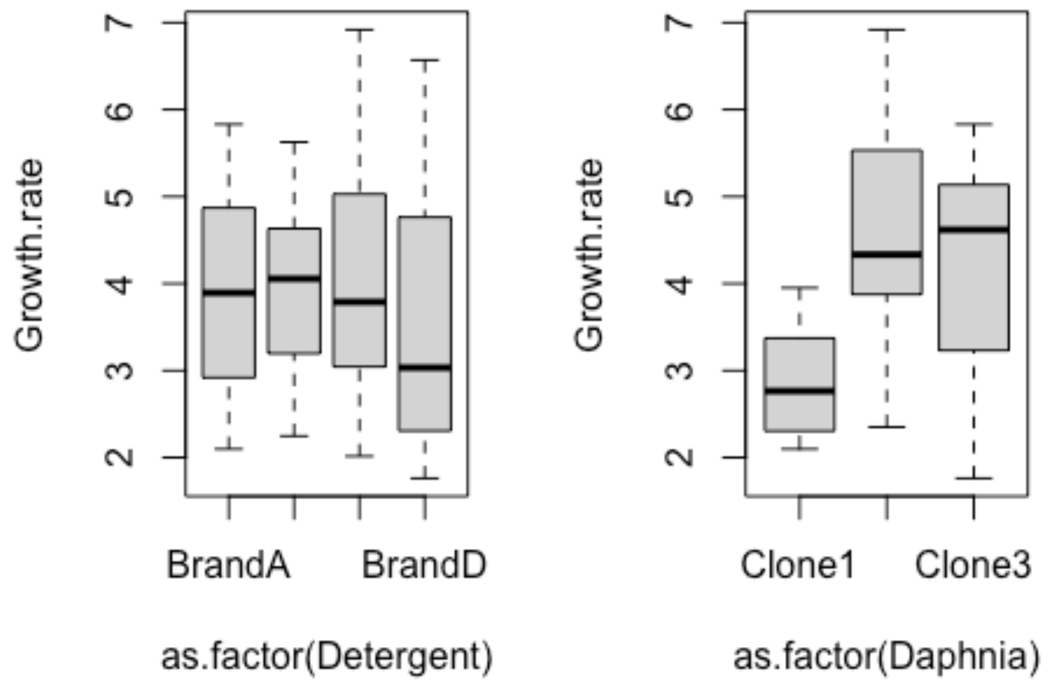


Figure 1. box and whisker plots

Outliers in boxplots come up as circles. So, no there are none.

2: Homogeneity of variances

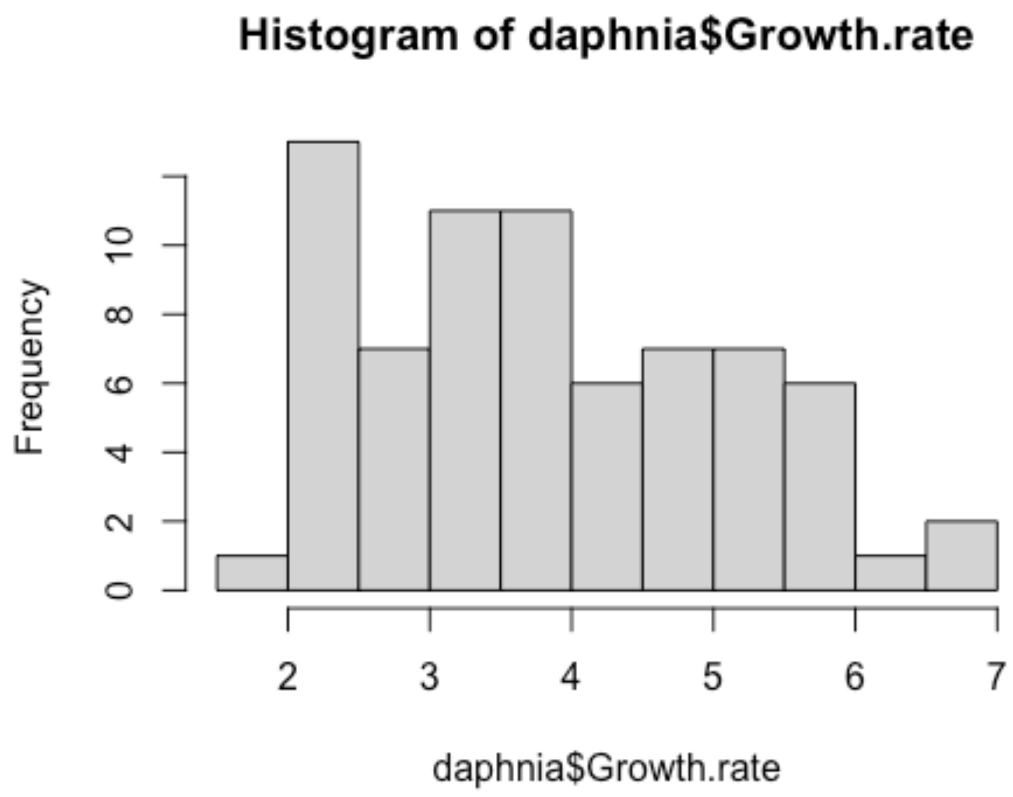
This is an important assumption for ANOVAS and regression analysis in general. To run the model explaining growth rate with Detergent brand and genotype, we have to assume that the variances within each brand and within each genotype are similar. Looking at the plot, they are *sort of* similar. A rule of thumb (for ANOVA) is that the ratio between the largest and smallest variance should not be much more than 4 (this is a conservative estimate).

```
require(dplyr)

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

daphnia %>%
  group_by(Detergent) %>%
  summarise (variance=var(Growth.rate))

## # A tibble: 4 x 2
##   Detergent variance
##   <chr>        <dbl>
## 1 BrandA        1.51
## 2 BrandB        1.09
## 3 BrandC        1.78
## 4 BrandD        2.38

daphnia %>%
  group_by(Daphnia) %>%
  summarise (variance=var(Growth.rate))

## # A tibble: 3 x 2
##   Daphnia variance
##   <chr>      <dbl>
## 1 Clone1     0.331
## 2 Clone2     1.53
## 3 Clone3     1.53
```

The ratio of variances for Clone 1 with the other two is larger than 4, it's about 5 actually. Well, this is not *much* more. This is for ANOVA, but for regression we really want to look at the variances of the residuals. Hmm. It's borderline. Our goal is a regression analysis and not the ANOVA. Also, note that ecological data is always very messy, and you can get stuck if you try to follow all these rules - they can stop you from doing anything! Thus, instead of being intimidated, we'll plough ahead, but we will keep this in mind when we *interpret our model and draw conclusions*! That's the important bit: you can do all statistical analysis you like, you can validate the assumptions, and that's all ok but *be explicitly clear about it in your report* ("the assumption of normality was violated - the ratio between the largest and smallest variance was 5, which is slightly too much and might bias the least square estimators") and *consider the consequences of this for when you draw your conclusions* (also explicitly in your report).

3: Are the data normally distributed?

```
dev.off()
#This is to tell R to not plot again in the same multi-plot frame as we set u
p before.

hist(daphnia$Growth.rate)
```



Errr. Well. This is a good one. WHAT exactly needs to be normally distributed? And how close to normal should it be? Linear regression assumes normality, but it *is reasonably robust against violations*. However, it assumes that the observations for each x are normal. So, if you measure something at x-2 10 times, you'd expect the resulting y to be normally distributed. Zuur et al. 2010 nicely shows this. Really, we are interested in the residuals. So we'll look at that later in more detail and for now hope that the growth rate is ok-ish normally distributed.

4: Are there excessively many zeroes?

Looking at the histogram, no.

5: Is there collinearity among the covariates?

Well, we only have categories here. So it doesn't apply. What we could do is check if all combinations are represented and if so, if there is any correlation. But as we know the dataset if homogeneous, that will suffice.

6: Visually inspect relationships

Done that before with the boxplots. No continous covariates. It looks like Clone 1 has an effect, but that's about it.

7: Consider interactions?

Not this time around - we want to avoid that as we've learned it is often difficult to interpret.

Ok, let's get to the meat:

Model daphnia:

We can also use our superior plotting skills to create barplots showing the means and standard errors of the mean for both clonal genotype and detergent presence. We will first use the function tapply to get the means and standard deviations for each of the two explanatory variables. tapply is about as useful as dplyr. Often, you can reach the same goal in R with different means.

```r
seFun <- function(x) {
    sqrt(var(x)/length(x))
}
detergentMean <- with(daphnia, tapply(Growth.rate, INDEX = Detergent,
    FUN = mean))
detergentSEM <- with(daphnia, tapply(Growth.rate, INDEX = Detergent,
    FUN = seFun))
cloneMean <- with(daphnia, tapply(Growth.rate, INDEX = Daphnia, FUN = mean))
cloneSEM <- with(daphnia, tapply(Growth.rate, INDEX = Daphnia, FUN = seFun))
```

Now we can use par(mfrow=(2,1), mar=c(4,4,1,1)) to plot them one above the other on the same graphics device and to reduce the size of the margins (Figure 2).

```r
par(mfrow=c(2,1),mar=c(4,4,1,1))
barMids <- barplot(detergentMean, xlab = "Detergent type", ylab = "Population
growth rate",
    ylim = c(0, 5))
arrows(barMids, detergentMean - detergentSEM, barMids, detergentMean +
    detergentSEM, code = 3, angle = 90)
```

```
barMids <- barplot(cloneMean, xlab = "Daphnia clone", ylab = "Population grow
th rate",
    ylim = c(0, 5))
arrows(barMids, cloneMean - cloneSEM, barMids, cloneMean + cloneSEM,
    code = 3, angle = 90)
```

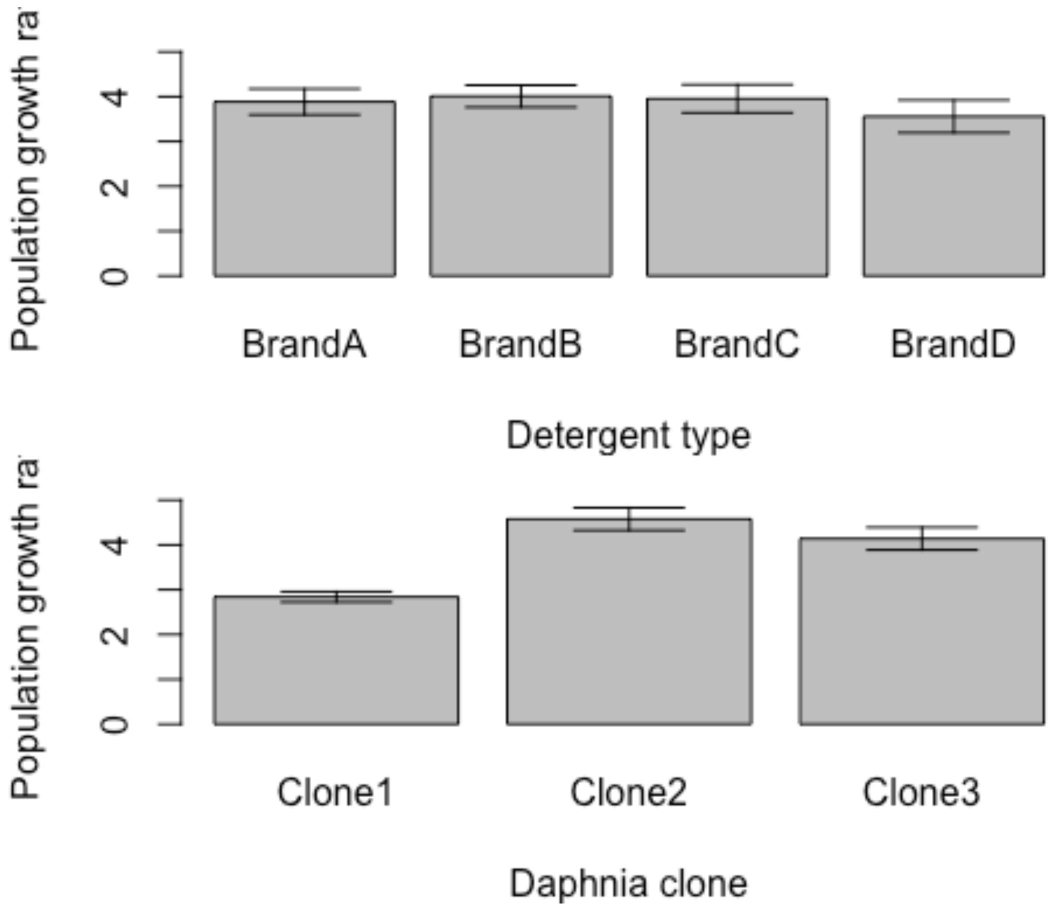


Figure 2. Boxplots of detergent type and daphnia clone against populations growth rates

The differences in the means for the detergents don't look like they matter but we should test whether they have any explanatory power. We can do this by adding both variables into the formula describing the model. So far we have only seen this in simple situations where there is one variable describing another (y ~ x). We can use the + sign to add extra variables into the right hand side of the formula: y ~ x + z means model y using both the x and z variables. So now we can fit the model and look at the analysis of variance table:

```
daphniaMod <- lm(Growth.rate ~ Detergent + Daphnia, data = daphnia)
summary(daphniaMod)

##
## Call:
## lm(formula = Growth.rate ~ Detergent + Daphnia, data = daphnia)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -2.25917 -0.72208 -0.06135  0.71041  2.28597 
## 
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)    
## (Intercept)     2.87280    0.30930   9.288 1.34e-13 ***
## DetergentBrandB  0.12521    0.35715   0.351    0.727    
## DetergentBrandC  0.06968    0.35715   0.195    0.846    
## DetergentBrandD -0.32660    0.35715  -0.914    0.364    
## DaphniaClone2    1.73725    0.30930   5.617 4.21e-07 ***
## DaphniaClone3    1.29884    0.30930   4.199 8.19e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 1.071 on 66 degrees of freedom
## Multiple R-squared:  0.3533, Adjusted R-squared:  0.3043 
## F-statistic: 7.211 on 5 and 66 DF,  p-value: 1.944e-05
```

So, from this, we can get the means for clone and detergents. Let's dissect this. The Intercept is the value for DetergentBrandA and DaphniaClone1. So, the estimate for DetergentBrandB is the difference between DetergentBrandA Clone1, and BrandB Clone 1. BrandC is the differentce between BrandA Clone1 and BrandC Clone1. And so forth. Clone 2 is the difference between BrandA Clone1, and BrandA Clone2. And so forth.

Or, in other words: The coefficient table tests whether one mean is different from zero and then tests whether the other means are different from the first. The first value it uses is the mean of the data from the first detergent brand for the first clonal genotype. The next lines are then the difference from this 'Brand A, Clone 1' reference to each of the other means. We know these differences already because they are the same as the differences in the means we calculated for the barplot:

```
detergentMean - detergentMean[1]

##      BrandA      BrandB      BrandC      BrandD 
##  0.00000000  0.12521198  0.06968013 -0.32660105

cloneMean - cloneMean[1]

##   Clone1   Clone2   Clone3 
## 0.000000 1.737246 1.298845
```

So to get the mean for 'Brand A, Clone 2', we add 1.74 on to the 'Brand A, Clone 1' reference value of 2.87 (1.74 + 2.87 = 4.61). If we want to get 'Brand B, Clone 3', we need to combine coefficients, that is, add the 0.125 to go from Brand A to Brand B and the 1.30 to go from Clone 1 to Clone 3 (0.125 + 1.30 + 2.87 = 4.29). We can use the Tukey HSD test to test all the pairwise differences. This is a very useful test to test across multiple categories. But to do that we first need to run the model (slightly) differently (because it doesn't accept the lm input).

```
daphniaANOVAMod <- aov(Growth.rate ~ Detergent + Daphnia, data = daphnia)
summary(daphniaANOVAMod)

##             Df Sum Sq Mean Sq F value   Pr(>F)    
## Detergent    3   2.21   0.737   0.642    0.591    
## Daphnia      2  39.18  19.589  17.063 1.06e-06 ***
## Residuals   66  75.77   1.148                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

?aov
```

You can see that the aov method is similar to lm, but less powerful as it only fits an ANOVA looking at the variances (mean squares). Yet, when we want to do multiple comparisons, it can be quite useful.

```
daphniaModHSD <- TukeyHSD(daphniaANOVAMod)
daphniaModHSD

##   Tukey multiple comparisons of means
##     95% family-wise confidence level
## 
## Fit: aov(formula = Growth.rate ~ Detergent + Daphnia, data = daphnia)
## 
## $Detergent
##                    diff        lwr       upr     p adj
## BrandB-BrandA  0.12521198 -0.8161307 1.0665547 0.9850797
## BrandC-BrandA  0.06968013 -0.8716625 1.0110228 0.9973423
## BrandD-BrandA -0.32660105 -1.2679437 0.6147416 0.7972087
## BrandC-BrandB -0.05553185 -0.9968745 0.8858108 0.9986474
## BrandD-BrandB -0.45181303 -1.3931557 0.4895296 0.5881893
## BrandD-BrandC -0.39628118 -1.3376239 0.5450615 0.6849619
## 
## $Daphnia
##                   diff        lwr       upr     p adj
## Clone2-Clone1  1.737246  0.9956362 2.4788555 0.0000013
## Clone3-Clone1  1.298845  0.5572351 2.0404544 0.0002393
## Clone3-Clone2 -0.438401 -1.1800107 0.3032086 0.3378930
```

Cool! This Tukey test even gives us the upper and lower 95 confidence intervals. That's brilliant!

The Tukey test now gives us two tables: the first shows which pairs of detergents differ in their effect on population growth rate (spoiler: none!) and the second shows which of the differences in genotype matter. It shows that the extra comparison between Clone 2 and 3 is not significant. Plotting daphniaModHSD (Figure 3) will also give us two plots, so we will use par(mfrow=c(1,2)) again to put them on the same page. We are also going to need a wide margin on the left for all the long pairwise labels.

```
par(mfrow=c(2,1),mar=c(4,4,1,1))
plot(daphniaModHSD)
```

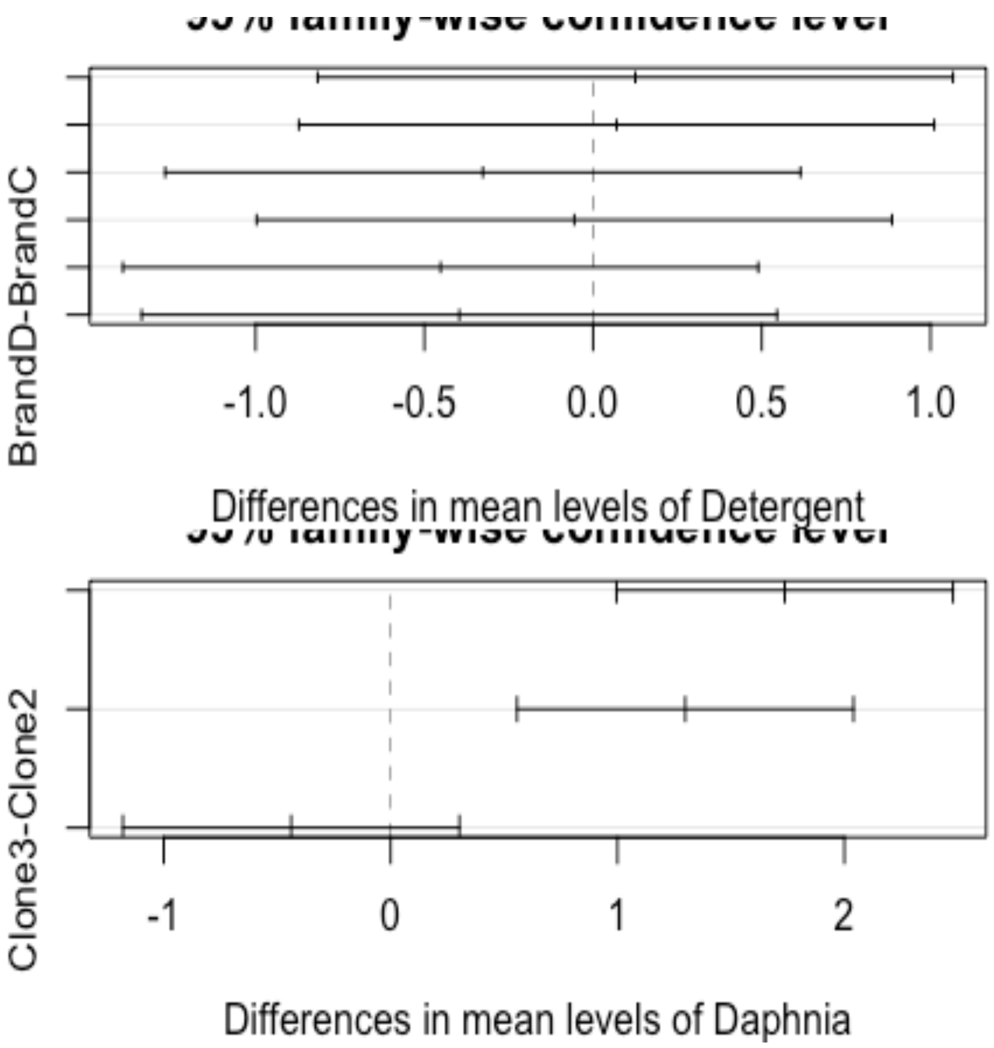
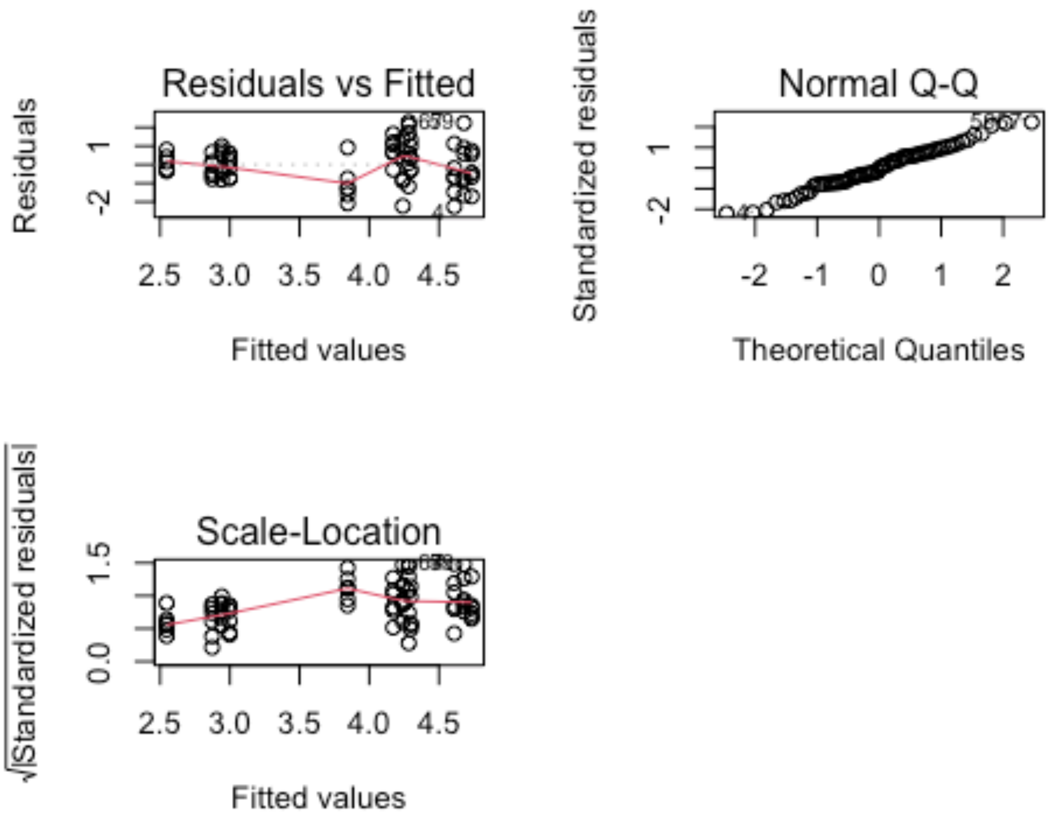


Figure 3. Tukey-test

Now some model validation:

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

## hat values (leverages) are all = 0.08333333
##  and there are no factor predictors; no plot no. 5
```



Ok. So much for stars in the sky. I'm sure this left bit comes from clone 3. The QQ plot at least looks good, and there is no outliner indeed. While this is not nice, it's also not the end of the world. This is publishable - given that you openly explain all the things that may affect how one interprets the data. With experience, you'll get a better "feel" for what is ok and what not. But you need to do it to get that feel, so don't be frustrated when your data doesn't fit perfectly! It never will.

Good. Now we move on to some more real bits!

Multiple regression

We will use an example dataset that looks at the volume of usable timber harvested from trees of known height and girth (diameter). We want to see whether both height and girth are important in predicting the yield from a tree. We will use the built in pairs() function to look at the distribution of the data: this generates a neat layout from a data frame that plots each variable against each other variable in the data frame (Figure 4).

```
timber <- read.delim("timber.txt")
summary(timber)
```

```
##      volume          girth            height
##  Min.   :0.7386   Min.   : 66.23   Min.   :18.9
##  1st Qu.:1.4048   1st Qu.: 88.17   1st Qu.:21.6
##  Median :1.7524   Median :102.94   Median :22.8  
##  Mean   :2.1847   Mean   :105.72   Mean   :22.8  
##  3rd Qu.:2.7010   3rd Qu.:121.69   3rd Qu.:24.0  
##  Max.   :5.5757   Max.   :164.38   Max.   :26.1
```

```
str(timber)
```

```
## 'data.frame':    31 obs. of  3 variables:
##  $ volume: num  0.746 0.746 0.739 1.188 1.361 ...
##  $ girth : num  66.2 68.6 70.2 83.8 85.4 ...
##  $ height: num  21 19.5 18.9 21.6 24.3 24.9 19.8 22.5 24 22.5 ...
```

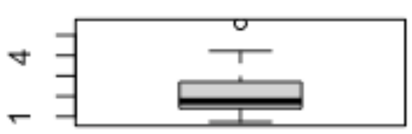
```
head(timber)
```

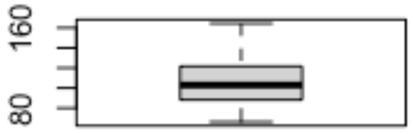
```
##   volume girth height
## 1 0.7458 66.23   21.0
## 2 0.7458 68.62   19.5
## 3 0.7386 70.22   18.9
## 4 1.1875 83.79   21.6
## 5 1.3613 85.38   24.3
## 6 1.4265 86.18   24.9
```

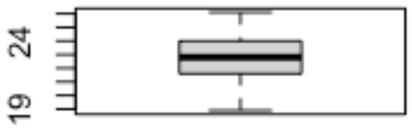
Let's get through our list:

1: Outliers

```
par(mfrow = c(2, 2))
boxplot(timber$volume)
boxplot(timber$girth)
boxplot(timber$height)
```







```
## null device
##           1
```

Ok. So there is one outlier in volume. It's rather large, but not excessively so - nothing suggestst that it's a measurement error or typo or so. It seem biologically true, so we keep it in but remember it's there for when we look at leverage. If this point would, say, be very important in determining some relationship we might want to have a look whether the model would turn out differently if we'd take it out. Generally, it's always better to keep stuff in! If you decide to take outliers out, *always disclose that in your report and JUSTIFY why you did it on biological grounds*. Stating "the analysis was odd with this data point in" is not a good justification. Saying "I think it's a typo because there is no tree in this world with a diameter of 5km" is biologically reasonable.

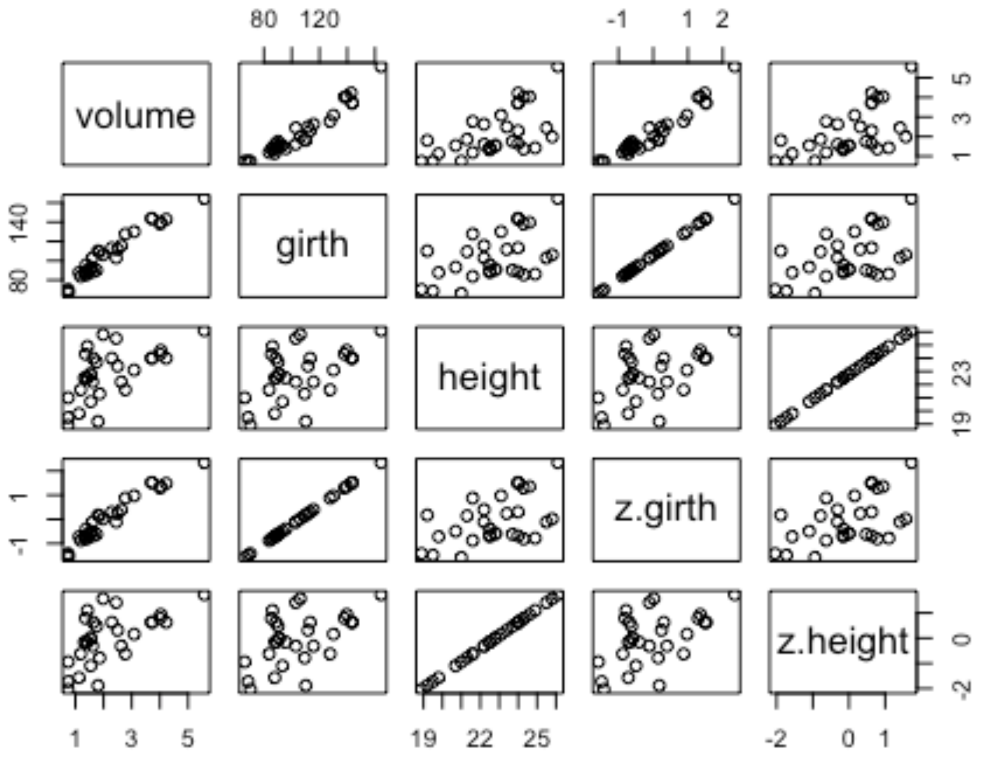
2: Homogeneity of variances

```
var(timber$volume)

## [1] 1.416803

var(timber$girth)

## [1] 627.0461

var(timber$height)
```

```
## [1] 3.654
```

Ouf. Wait a moment. Since we're interested in volume (y), we could standardize our x'es for the What does this thing "homogeneity of variances" really mean? It means that the level of variance of the explanatory variables are constant across the sample. Therefore, the violation of homogeneity is called heterogeneity or heteroscedasticy. The easiest option to deal with heterogeneity is data transformation. analysis.

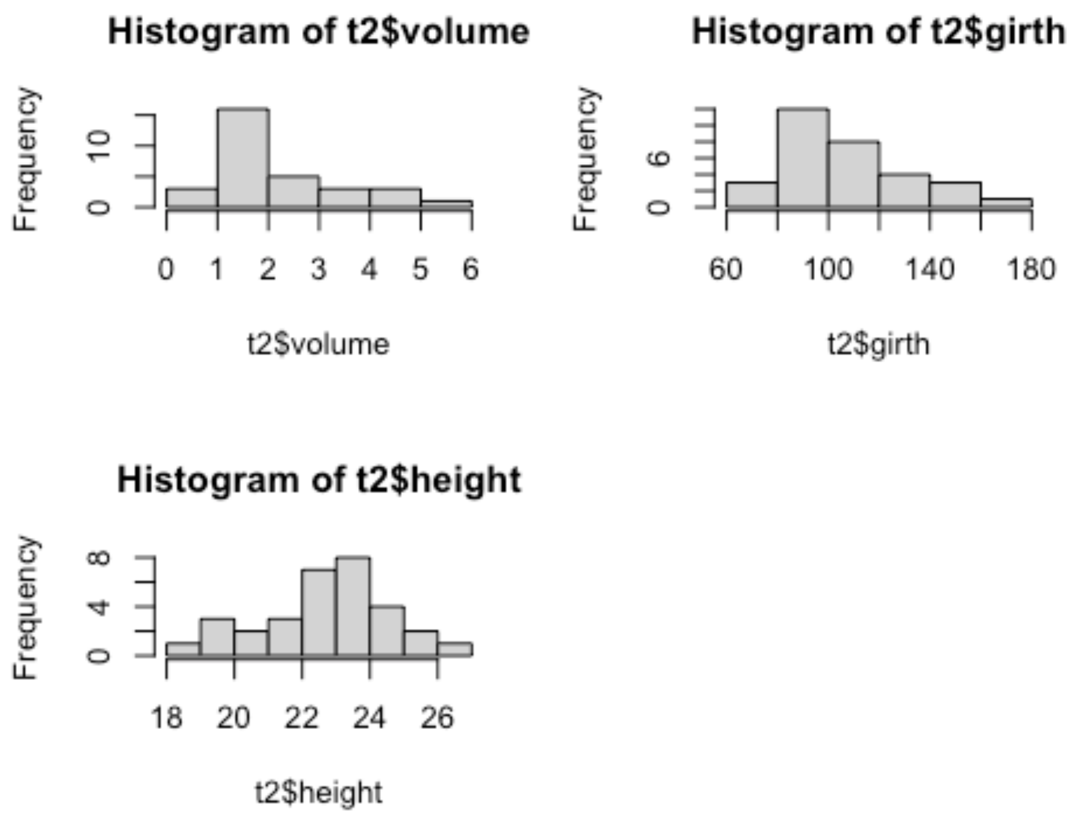
```
t2<-as.data.frame(subset(timber, timber$volume!="NA"))
t2$z.girth<-scale(timber$girth)
t2$z.height<-scale(timber$height)
var(t2$z.girth)

##      [,1]
## [1,]    1

var(t2$z.height)

##      [,1]
## [1,]    1

plot(t2)
```



Ok. It's probably a good idea to run the analysis with z-scores.

3: Are the data normally distributed?

```
par(mfrow = c(2, 2))
hist(t2$volume)
hist(t2$girth)
hist(t2$height)
```



```
## null device
##           1
```

We’ve learned and won’t freak out this time. This is as good as it gets in ecology, evolution and conservation. What this really gets at is whether the residuals are normally distributed – but we dealt with this in a previous ho.

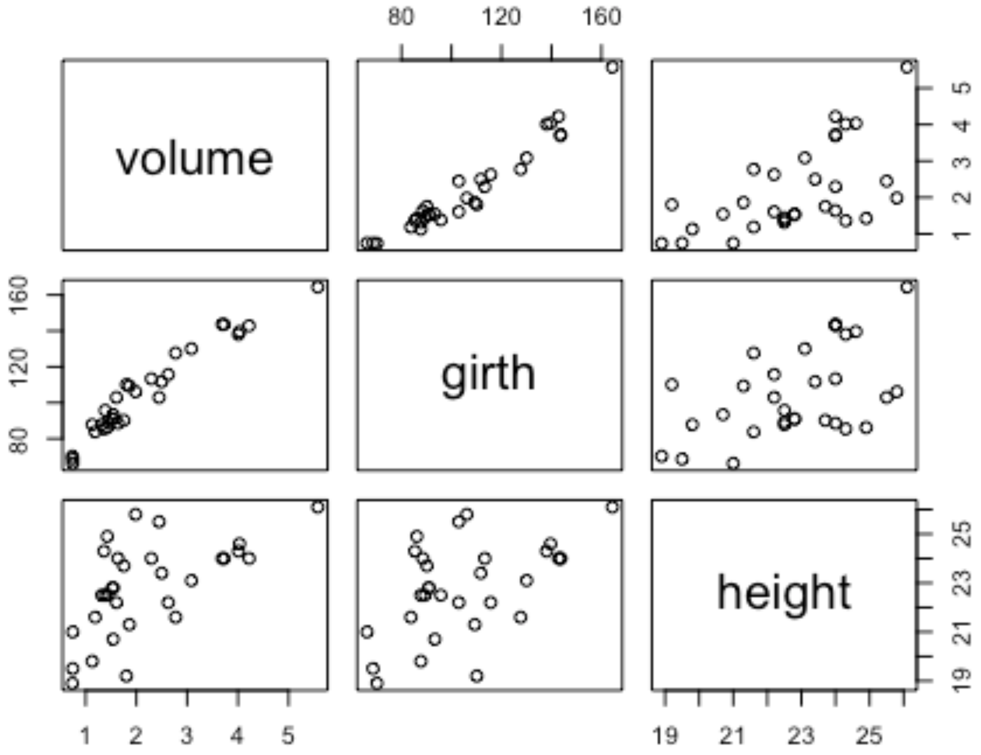
4: Are there excessively many zeroes?

Nope.

5: Is there collinearity among the covariates?

Uh. This is a nice one.

```
pairs(timber)
```



```
cor(timber)

##           volume     girth    height
## volume 1.0000000 0.9671176 0.5982517
## girth  0.9671176 1.0000000 0.5192873
## height 0.5982517 0.5192873 1.0000000
```

So these variables are all positively correlated, with tree diameter being a better predictor of timber yield than height. Too much correlation among the predictors (girth and height) is not a good thing. This is called “collinearity”. What it does is it inflates variation. So, when you have a lot of collinearity in your covariates, you’ll get larger standart errors of those correlated variables than you’d get if there was no collinearity. That means that it is more difficult to detect an effect, that *you are likely to not get a significant result even though there might be one*. This means that *if there is lots of collinearity any normal evaluation of a model is super conservative*. Also, dropping covariates can affect the estimates of other covariates if there is collinearity around. That can be super confusing. The standard errors are inflated with the square root of the *Variance Inflation Factor*. This VIF we can use to find out what amoung of collinearity is too much. VIF can be calculated by running an extra linear model in which the covariate of focus (here, girth) is y, and all other covariates of the model (here only one, height) are the covariates. Then you can calculate the VIF as follows:

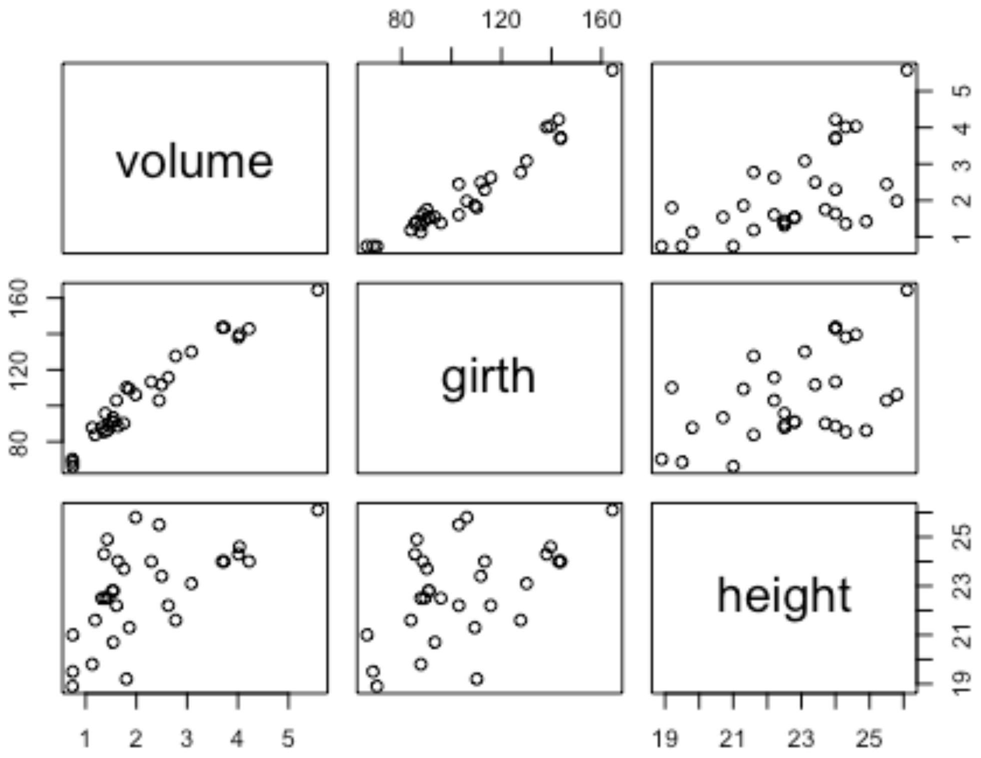
$$VIF = \frac{1}{1 - R^2}$$

```
summary(lm(girth ~ height, data = timber))

##
## Call:
## lm(formula = girth ~ height, data = timber)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -33.82 -15.32  -0.57  21.90  36.21
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -49.382     47.559  -1.038  0.30770
## height         6.803      2.079   3.272  0.00276 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.77 on 29 degrees of freedom
## Multiple R-squared:  0.2697, Adjusted R-squared:  0.2445
## F-statistic: 10.71 on 1 and 29 DF,  p-value: 0.002757

VIF<- 1/(1-0.27)
VIF

## [1] 1.369863

sqrt(VIF)

## [1] 1.170411
```

The standard errors of girth are thus inflated by 1.17, which is not a lot. A VIF of 1.4 is also ok. Some people are super strict and say throw out all covariates with VIF more than 3. Others say VIF more than 10. I say, it depends. I also say, test for it, and keep it in mind when you do your interpretation, and disclose it in your report! Think biology, always.

Back to the pairs plot:

```
pairs(timber)
```

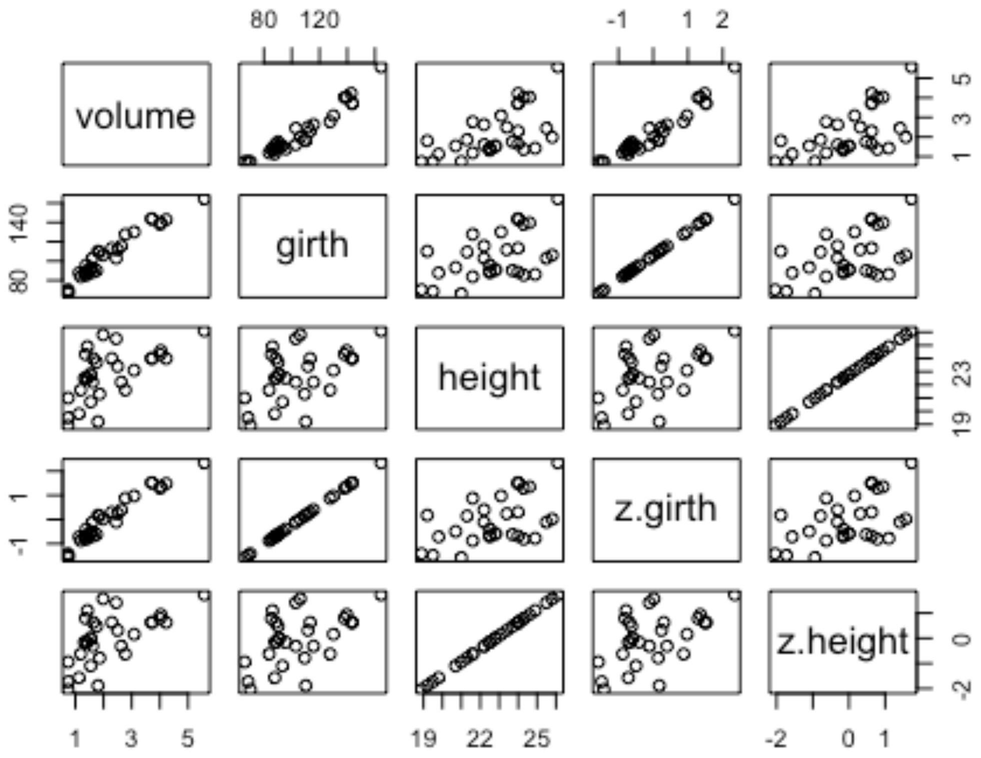


```r
cor(timber)
```

```
##           volume     girth    height
## volume 1.0000000 0.9671176 0.5982517
## girth  0.9671176 1.0000000 0.5192873
## height 0.5982517 0.5192873 1.0000000
```

We also see the outlier in volume. It behaves as all other points behave, and doesn't influence the correlation much. But we'll see that better in the leverage plots later, but for now, we're happy. We can do the same thing with your scaled predictors, but that won't be much different:

```r
pairs(t2)
```



```
cor(t2)

##             volume     girth    height   z.girth  z.height
## volume   1.0000000 0.9671176 0.5982517 0.9671176 0.5982517
## girth    0.9671176 1.0000000 0.5192873 1.0000000 0.5192873
## height   0.5982517 0.5192873 1.0000000 0.5192873 1.0000000
## z.girth  0.9671176 1.0000000 0.5192873 1.0000000 0.5192873
## z.height 0.5982517 0.5192873 1.0000000 0.5192873 1.0000000
```

You can see the correlations are the same (they should be).

6: Visually inspect relationships

Well, we've done that for the covariates, but the pair plot also shows the relationships with the response, volume. There seem to be some relationships.

7: Consider interactions?

Not now because we're still learning. We'll look at interactions with the next dataset.

Now on to our model:

If girth has such a high correlation with volume, do we actually need both variables?

```r
timberMod <- lm(volume ~ girth + height, data = timber)
anova(timberMod)
```

```
## Analysis of Variance Table
##
## Response: volume
##           Df Sum Sq Mean Sq  F value Pr(>F)
## girth      1 39.755  39.755 503.1070 <2e-16 ***
## height     1  0.537   0.537   6.7933 0.0145 *
## Residuals 28  2.213   0.079
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So yes, it looks like height is needed as well as the girth of the tree. If we look at the coefficients in the linear model summary, then we can get the estimates of the slopes and intercept.

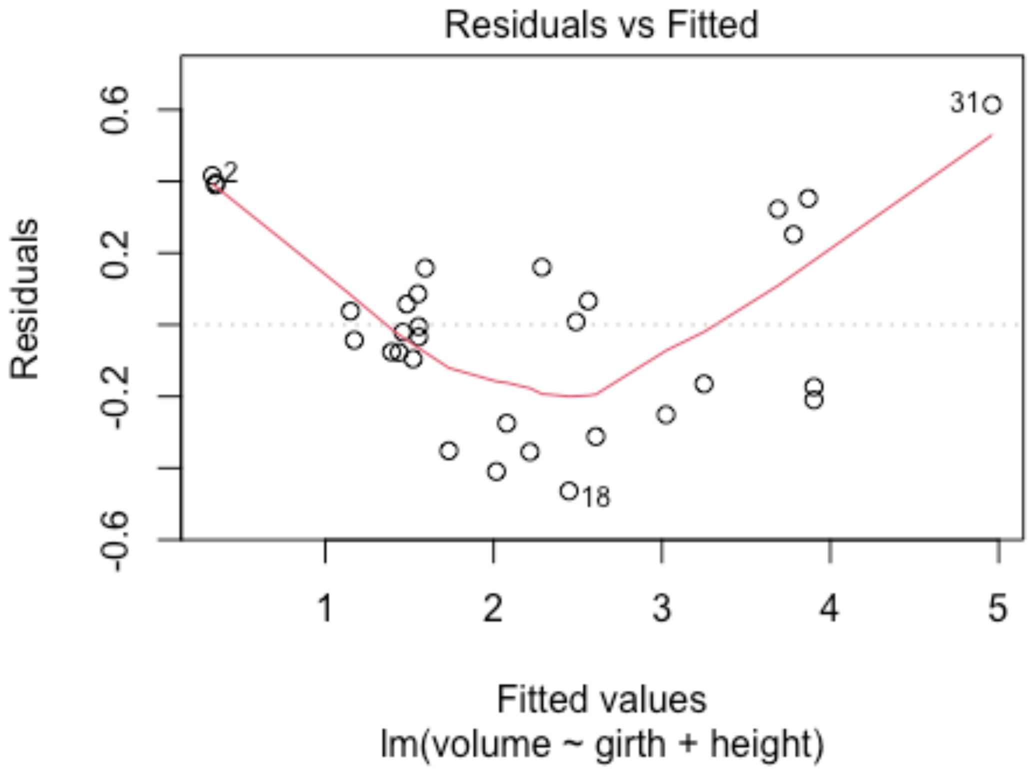
```r
summary(timberMod)
```

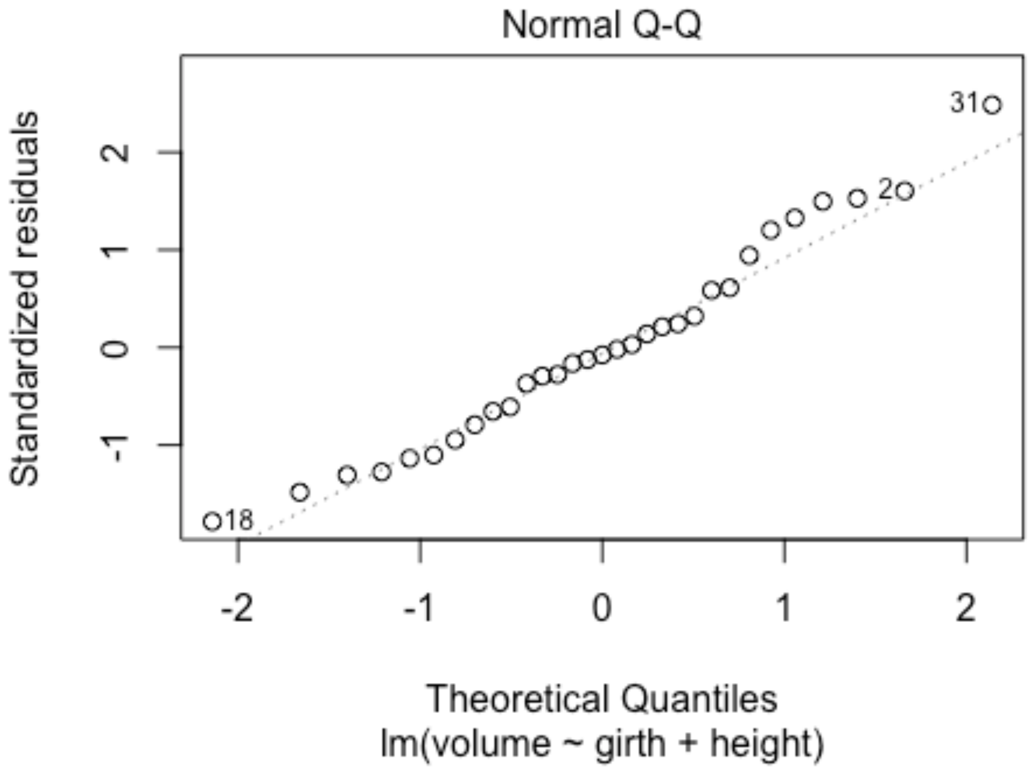
```
##
## Call:
## lm(formula = volume ~ girth + height, data = timber)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46391 -0.19171 -0.02072  0.15929  0.61439
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.198997   0.625537  -6.713 2.75e-07 ***
## girth        0.042725   0.002398  17.815  < 2e-16 ***
## height       0.081883   0.031416   2.606   0.0145 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2811 on 28 degrees of freedom
## Multiple R-squared:  0.9479, Adjusted R-squared:  0.9442
## F-statistic:   255 on 2 and 28 DF,  p-value: < 2.2e-16
```

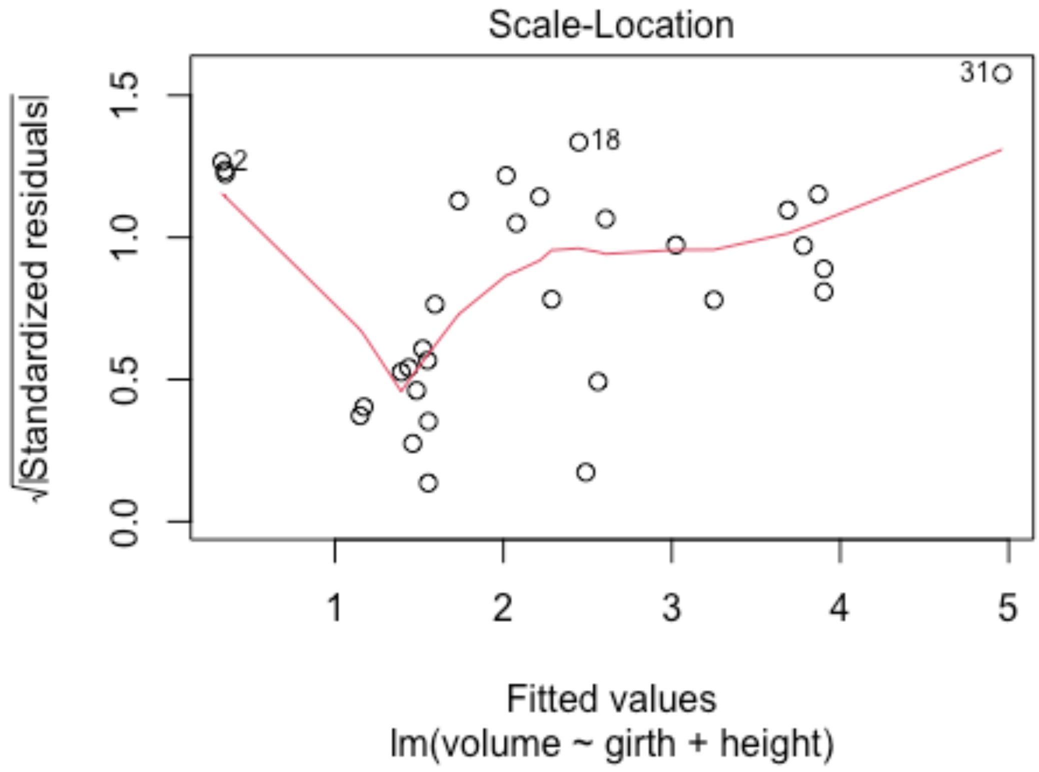
So, looking at the r^2, more than 90 percent of the variation in timber volume is explained by the following equation: *volume = −4.2 + 0.08 × height + 0.04 × girth*. One important thing to note is that this model makes stupid predictions if the tree is very small — a one metre sampling of diameter 10 cm will contain -3.72 tonnes of timber. It is not at all sensible to expect a statistical model to make good predictions outside of the range of the data used to fit it.

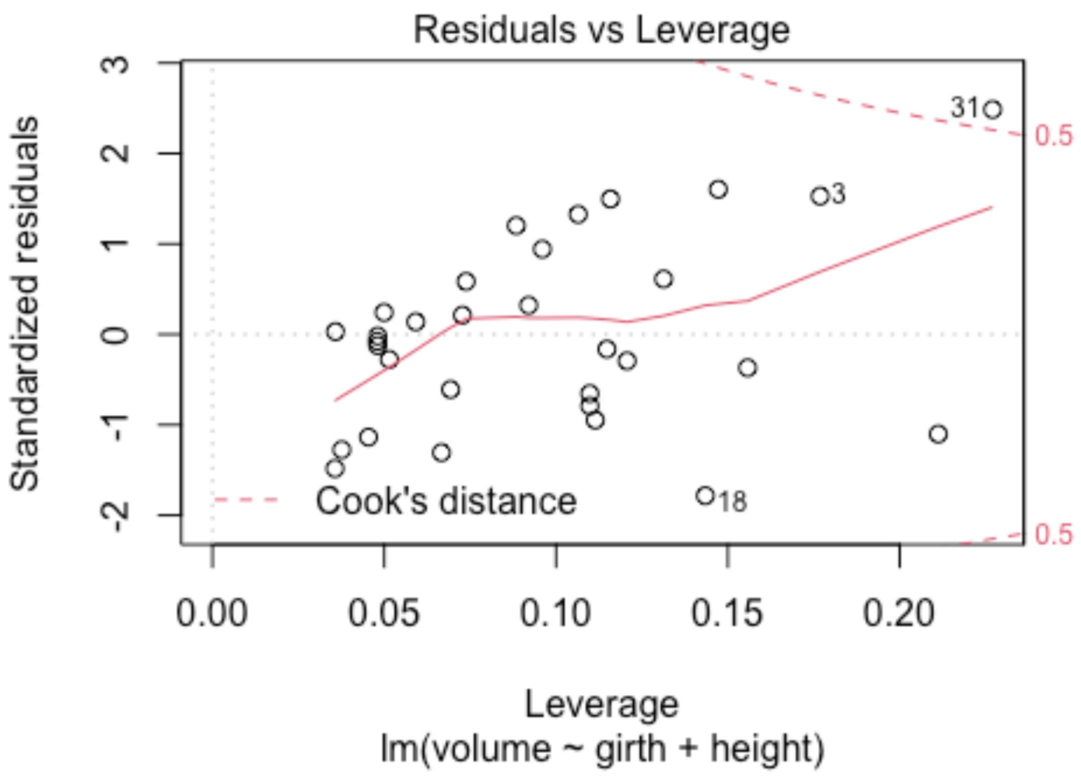
Time for model validation:

```r
plot(timberMod)
```


Residuals vs Fitted
Residuals
0.6
0.2
-0.2
-0.6
1
2
3
4
5
2
31
18
Fitted values
lm(volume ~ girth + height)


Normal Q-Q
Standardized residuals
Theoretical Quantiles
lm(volume ~ girth + height)
-2
-1
0
1
2
18
2
31


Scale-Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
1
2
3
4
5
Fitted values
lm(volume ~ girth + height)
2
18
31



Ouch. Again, not so much starry sky. QQ is ok, because most of the points fall within the expected line. Leverage is not nice, and there seems to be one point especially that stands out (31). If we'd want to publish this we'd run the whole thing without this point and see if we'd come to the same conclusions. If so, we could leave it in. If not, we'd have to do some thinking into why 31 stands out so much.

Exercises:

1) Run the timber model without the previously found outlier. See what your conclusions are. If you'd publish it, would you do it with outlier or without? Why?

2) Another example: The previous two examples looked at predicting a variable using two categorical variables and then two continuous variables. What if you have both continuous and categorical variables like sex and tarsus in the model? We will use the Ipopmopsis dataset, which records plant fruit production as a function of root stock diameter for plants that have been grazed by rabbit and those that have been protected from grazing. We want to know whether fruit production (response) is associated with both grazing (fixed factor) and root stock size (continous covariate). The model we are fitting here is describing fruit production as a function of root stock size — this is like in a normal regression, so we are after a slope and an intercept. In addition to this, we want to fit fruit production as a function of the two levels of

grazing — we are testing whether there is a difference in intercept between the two lines.

```
plantGrowth <- read.delim("ipomopsis.txt")
summary(plantGrowth)
```

```
##       Root            Fruit           Grazing         
##  Min.   : 4.426   Min.   : 14.73   Length:40         
##  1st Qu.: 6.083   1st Qu.: 41.15   Class :character  
##  Median : 7.123   Median : 60.88   Mode  :character  
##  Mean   : 7.181   Mean   : 59.41                     
##  3rd Qu.: 8.510   3rd Qu.: 76.19                     
##  Max.   :10.253   Max.   :116.05
```

Use the checklist (1-6) on the plant growth dataset. Fruit is the response, Root the covariate and Grazing the fixed factor.You want to specify your null model, the maximum model, and find the best model that includes and interaction between Root and Grazing (indicated with a * in your model formula). You need to run through the checklist as we've done above first, this time at point 6 to also plot potential interactions. You can do that best by seperately plotting xy plots and/or cor.tests for each category. Then you want to do model validation.

3) Use the sparrow dataset to find out how much each structural measurement (tarsus, wing, bill) and sex affects body mass. Use multiple linear models to do that. Run the checklist first. Write down the null-model, the maximum model, and the final model that you select to be the best fit. Explain what it means.

Stats with Sparrows - 12

Julia Schroeder

12 ANOVA and repeatability

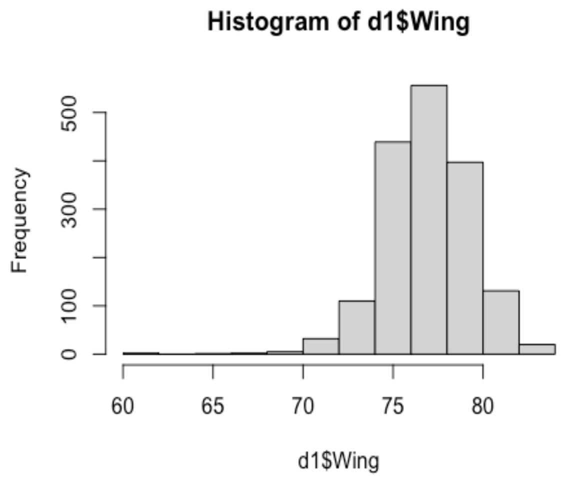
This handout is long, and there are two excercises, with group submissions each. It is recommended to do the first excercise before you begin with the repeatability part. I'll add a note at that bit in the handout to remind you.

Again: housekeeping!

```
rm(list=ls())
setwd("H:/StatsWithSparrows")

d<-read.table("SparrowSize.txt", header=TRUE)
```

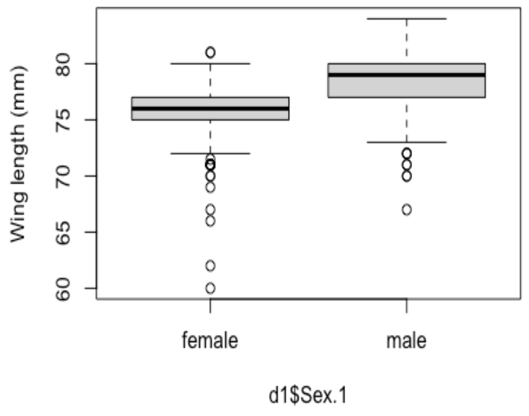
We now examine ANOVA - an analysis of variance that requires a continuous response, and a categorical predictor (explanatory) variable. ANOVAs test for differencs in variances, between and within groups (the groups are determined by the levels of the factor). They are a certain kind of linear models. To make things more exciting, we use a new variable: wing length. Let's check it out first:

```
d1<-subset(d, d$Wing!="NA")
summary(d1$Wing)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    60.0    76.0    77.0    77.4    79.0    84.0

hist(d1$Wing)
```



Ok, there are some missing values, and some “outliers”. These likely come from young birds, or from birds that are moulting. We leave them in for now, but keep this in mind for when we interpret our results.

```
model1<-lm(Wing~Sex.1,data=d1)
summary(model1)

##
## Call:
## lm(formula = Wing ~ Sex.1, data = d1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.0961  -1.0961  -0.0961   1.3683   5.3683
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 76.09611    0.07175 1060.50   <2e-16 ***
## Sex.1male    2.53562    0.09998   25.36   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.057 on 1693 degrees of freedom
## Multiple R-squared:  0.2753, Adjusted R-squared:  0.2749
## F-statistic: 643.1 on 1 and 1693 DF,  p-value: < 2.2e-16

boxplot(d1$Wing~d1$Sex.1, ylab="Wing length (mm)")
```



Good, we understand most of these output. The F-statistic comes from an ANOVA.

```
anova(model1)

## Analysis of Variance Table
##
## Response: Wing
##               Df Sum Sq Mean Sq F value    Pr(>F)    
## Sex.1          1 2722.0 2721.98  643.15 < 2.2e-16 ***
## Residuals 1693 7165.3    4.23                      
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

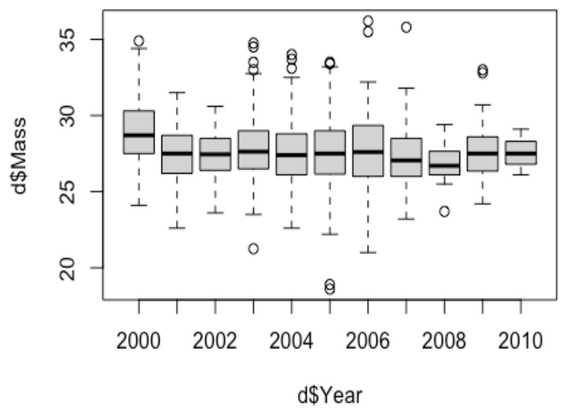
All the sums of squares. Let's have a closer look. There are the sums of squares of the between-group variance (Sex.1), and those for the within-group variances (the residual variance). Then, there are the Mean squares for each of these two (within- and between-groups). The F-value, that is calculated by dividing the mean squares of the between-group estimate by the residual mean squares. Then we ask R to look up the p-value of the F-distribtuion. Thank you Ronald Fisher!

Ok, now we know that wing length differs between the two sexes. But, how much does it differ? To find that out, we actually have to do a t-test. Such a test that one runs after the main analysis, to see which group differs, and how much it differs, from another group, are called "post-hoc" tests.

```
t.test(d1$Wing~d1$Sex.1, var.equal=TRUE)

## 
##  Two Sample t-test
## 
## data:  d1$Wing by d1$Sex.1
## t = -25.36, df = 1693, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.731727 -2.339518
## sample estimates:
## mean in group female   mean in group male 
##             76.09611             78.63173
```

This is ok for two groups. But what if we have lots of groups? Let's see how that works by testing for differences between years:

```
boxplot(d$Mass~d$Year)
```



```
m2<-lm(Mass~as.factor(Year),data=d)
anova(m2)

## Analysis of Variance Table
##
## Response: Mass
##                   Df Sum Sq Mean Sq F value    Pr(>F)
## as.factor(Year)   10  340.2  34.020  7.8866 1.721e-12 ***
## Residuals       1693 7303.0   4.314
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There's many years, and while we could do a summary -

```
summary(m2)

##
## Call:
## lm(formula = Mass ~ as.factor(Year), data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.0051 -1.4051 -0.1089  1.2645  8.4874
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          28.8537     0.1375 209.772  < 2e-16 ***
## as.factor(Year)2001  -1.3620     0.2518  -5.410 7.22e-08 ***
## as.factor(Year)2002  -1.3871     0.2903  -4.778 1.92e-06 ***
## as.factor(Year)2003  -1.0596     0.2105  -5.035 5.30e-07 ***
## as.factor(Year)2004  -1.3182     0.1686  -7.816 9.49e-15 ***
## as.factor(Year)2005  -1.2486     0.1695  -7.367 2.71e-13 ***
## as.factor(Year)2006  -1.1411     0.2349  -4.858 1.29e-06 ***
## as.factor(Year)2007  -1.4176     0.2546  -5.568 2.99e-08 ***
## as.factor(Year)2008  -2.0810     0.6411  -3.246  0.00119 **
## as.factor(Year)2009  -0.9842     0.4544  -2.166  0.03046 *
## as.factor(Year)2010  -1.2871     1.2070  -1.066  0.28642
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.077 on 1693 degrees of freedom
##   (66 observations deleted due to missingness)
## Multiple R-squared:  0.04451,    Adjusted R-squared:  0.03887
## F-statistic: 7.887 on 10 and 1693 DF,  p-value: 1.721e-12
```

we'd only get differences between a year and the reference level (2000), but we struggle to understand the differences between other years, say, the difference between 2009 and 2006. That's something we could work out by just adding the year's level to the reference, and comparing these, but it's quite a lot of work. What we really want to do is t-tests between each and every combination to work out what is going one here. That's practically aTukey's post-hoc test:

```
?TukeyHSD
```

This says it needs a fitted model object of the aov() fit.

```
?aov
```

So there's another way to fit an ANOVA. Let's see.

```
am2<-aov(Mass~as.factor(Year),data=d)
summary(am2)

##                   Df Sum Sq Mean Sq F value   Pr(>F)
## as.factor(Year)   10    340   34.02   7.887 1.72e-12 ***
## Residuals       1693   7303    4.31
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 66 observations deleted due to missingness
```

This returns the same information than from the lm() call.

```
TukeyHSD(am2)

##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = Mass ~ as.factor(Year), data = d)
##
## $`as.factor(Year)`
##                  diff        lwr         upr     p adj
## 2001-2000 -1.36197549 -2.1734639 -0.55048712 0.0000039
## 2002-2000 -1.38706137 -2.3227432 -0.45137957 0.0001019
## 2003-2000 -1.05961042 -1.7379444 -0.38127646 0.0000284
## 2004-2000 -1.31818721 -1.8617515 -0.77462289 0.0000000
## 2005-2000 -1.24861442 -1.7948606 -0.70236828 0.0000000
## 2006-2000 -1.14112306 -1.8981612 -0.38408495 0.0000688
## 2007-2000 -1.41755784 -2.2380803 -0.59703533 0.0000016
## 2008-2000 -2.08100080 -4.1474707 -0.01453085 0.0466609
## 2009-2000 -0.98416285 -2.4486968  0.48037105 0.5296791
## 2010-2000 -1.28706115 -5.1772604  2.60313813 0.9929548
## 2002-2001 -0.02508588 -1.0932301  1.04305830 1.0000000
## 2003-2001  0.30236507 -0.5494380  1.15416816 0.9878264
## 2004-2001  0.04378829 -0.7051401  0.79271666 1.0000000
## 2005-2001  0.11336108 -0.6375160  0.86423815 0.9999935
## 2006-2001  0.22085243 -0.6948649  1.13656977 0.9995087
## 2007-2001 -0.05558234 -1.0244418  0.91327715 1.0000000
## 2008-2001 -0.71902530 -2.8487505  1.41069994 0.9917374
## 2009-2001  0.37781264 -1.1746986  1.93032386 0.9994681
## 2010-2001  0.07491434 -3.8492521  3.99908075 1.0000000
## 2003-2002  0.32745095 -0.6434021  1.29830396 0.9918009
## 2004-2002  0.06887416 -0.8131013  0.95084966 1.0000000
## 2005-2002  0.13844695 -0.7451839  1.02207779 0.9999908
## 2006-2002  0.24593831 -0.7814493  1.27332592 0.9995402
## 2007-2002 -0.03049647 -1.1055201  1.04452715 1.0000000
## 2008-2002 -0.69393943 -2.8740121  1.48613320 0.9948248
## 2009-2002  0.40289852 -1.2179896  2.02378659 0.9993590
## 2010-2002  0.10000022 -3.8517170  4.05171745 1.0000000
## 2004-2003 -0.25857679 -0.8606709  0.34351731 0.9523031
## 2005-2003 -0.18900400 -0.7935203  0.41551232 0.9955235
## 2006-2003 -0.08151264 -0.8816138  0.71858850 0.9999999
## 2007-2003 -0.35794741 -1.2183614  0.50246662 0.9615269
## 2008-2003 -1.02139037 -3.1040217  1.06124094 0.8905172
## 2009-2003  0.07544757 -1.4118031  1.56269828 1.0000000
## 2010-2003 -0.22745073 -4.1262589  3.67135750 1.0000000
## 2005-2004  0.06957279 -0.3784955  0.51764107 0.9999915
## 2006-2004  0.17706414 -0.5124915  0.86661983 0.9991483
## 2007-2004 -0.09937063 -0.8580784  0.65933717 0.9999983
## 2008-2004 -0.76281359 -2.8055271  1.27989990 0.9821835
## 2009-2004  0.33402436 -1.0967936  1.76484226 0.9996322
## 2010-2004  0.03112606 -3.8465060  3.90875817 1.0000000
## 2006-2005  0.10749135 -0.5841803  0.79916303 0.9999915
## 2007-2005 -0.16894342 -0.9295749  0.59168802 0.9997647
## 2008-2005 -0.83238638 -2.8758151  1.21104237 0.9667328
## 2009-2005  0.26445157 -1.1673873  1.69629044 0.9999567
## 2010-2005 -0.03844673 -3.9164557  3.83956222 1.0000000
## 2007-2006 -0.27643477 -1.2001674  0.64729790 0.9968732
## 2008-2006 -0.93987773 -3.0494566  1.16970112 0.9393125
## 2009-2006  0.15696021 -1.3677970  1.68171738 0.9999998
## 2010-2006 -0.14593808 -4.0592072  3.76733105 1.0000000
## 2008-2007 -0.66344296 -2.7966268  1.46974090 0.9957111
## 2009-2007  0.43339498 -1.1238574  1.99064734 0.9982981
## 2010-2007  0.13049669 -3.7955479  4.05654123 1.0000000
## 2009-2008  1.09683794 -1.3571539  3.55082975 0.9380454
## 2010-2008  0.79393965 -3.5662056  5.15408490 0.9999621
## 2010-2009 -0.30289830 -4.4120866  3.80628996 1.0000000
```

And here we get the complete table, with each of the combinations, the difference in body mass between those, the upper and lower confidence intervals, and a p-value that's adjusted for multiple comparisons. This is because - as you've learned earlier, our definition of "statistical significance" means that 1 in 20 tests will come up as a false positive. So, if you run, as here, 55 tests, 55/20 or 2.75 tests on average will wrongly come up as statistically significant. That's problematic, of course. To account for that, we can adjust the p-values by that percentage. That's what this function does automatically. It is a bit difficult to keep the overview, though, with so many levels.

This is a good time to complete excercise 1. After that, continue below.

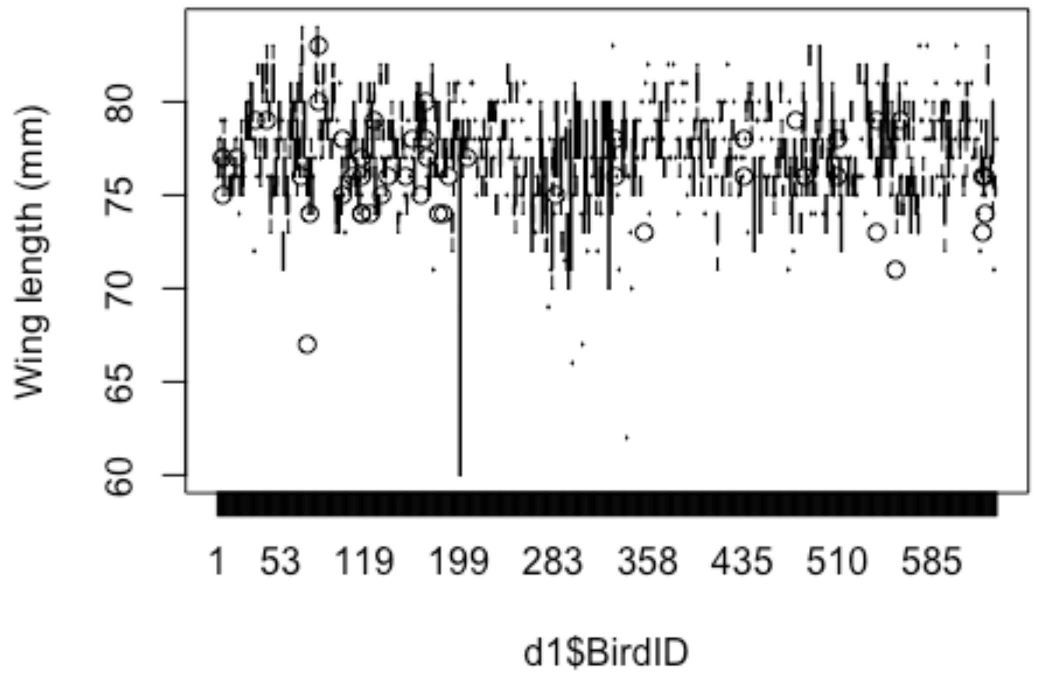
1) A researcher wants to know how the magic properties of the plant Aconite change with the environmental temperature the dried flower or leaves are stored in. She collected 100 leaves, and 100 flowers, and stored them in closed containers in a freezer (A), at room temperature (B), and in a heated room (C). She is now interested in the change in their magic properties with respect to their treatment. You are tasked with the statistical analysis. Per group, prepare two separate analyses - one using linear models and their results, and one using ANOVA. Conduct your analyses, then write a methods and results section for this experiments. Make sure to include a descriptive statistics section in addition to the results section. Per group, submit two documents (1 A4 each), both with methods and results.

What if there are many more levels?

What can we do if there's even more levels? Let's use BirdID as factorial for now.

Each sparrow is measured more than once. Now, it is interesting to know if each wing length measurement is the same for the same sparrow everytime it is caught, or whether the wing length changes over time. For instance, one could imagine that sparrows grow longer wing feathers when they get older. We can test that with an ANOVA - it tests, if we remember from earlier, whether the *variation within-groups is smaller than the variation among-groups*. In this case, group is birdID. Note that in english, the term *between* is only used for two groups, not for more, then we use *among* instead.

```r
boxplot(d1$Wing~d1$BirdID, ylab="Wing length (mm)")
```



Here, we have loads and loads of groups. A couple hundred groups, actually. This is when an ANOVA can be really helpful. Let's see where this gets us:

That's a lot! of individual birds. Before we even start with any model, we need to get a grip on the data structure. Can we get an idea of how many individual birds there are, and how often they are measured? There are several ways to do this, but here, we'll use dplyr

```
install.packages("dplyr")

require(dplyr)

## Loading required package: dplyr

## Warning: package 'dplyr' was built under R version 4.0.2

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
tbl_df(d1)

## Warning: `tbl_df()` is deprecated as of dplyr 1.0.0.
## Please use `tibble::as_tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.

## # A tibble: 1,695 x 8
##    BirdID  Year Tarsus  Bill  Wing  Mass   Sex Sex.1
##     <int> <int>  <dbl> <dbl> <dbl> <dbl> <int> <chr>
##  1      1  2002   16.9  NA      76  23.6     0 female
##  2      2  2001   16.8  NA      76  27.5     1 male
##  3      2  2002   17.2  NA      76  28.1     1 male
##  4      2  2003   17.5  13.5    76  27.8     1 male
##  5      2  2004   17.8  13.4    77  26.5     1 male
##  6      2  2004   17.7  13.1    78  26       1 male
##  7      2  2004   17.5  13.3    77  27.7     1 male
##  8      2  2004   17.8  13.1    77  26.6     1 male
##  9      2  2004   17.8  13.1    77  26.6     1 male
## 10      2  2005   17.7  13.3    77  26.6     1 male
## # … with 1,685 more rows

glimpse(d1)

## Rows: 1,695
## Columns: 8
## $ BirdID <int> 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 4, 5, 5, 5, 5, 5, 5, 5,
5,…
## $ Year   <int> 2002, 2001, 2002, 2003, 2004, 2004, 2004, 2004, 2004, 2005,
20…
## $ Tarsus <dbl> 16.9, 16.8, 17.2, 17.5, 17.8, 17.7, 17.5, 17.8, 17.8, 17.7,
18…
## $ Bill   <dbl> NA, NA, NA, 13.5, 13.4, 13.1, 13.3, 13.1, 13.1, 13.3, 13.2,
NA…
## $ Wing   <dbl> 76, 76, 76, 76, 77, 78, 77, 77, 77, 77, 78, 78, 79, 76, 76,
76…
## $ Mass   <dbl> 23.60, 27.50, 28.10, 27.75, 26.50, 26.00, 27.70, 26.60, 26.
60,…
## $ Sex    <int> 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,…
## $ Sex.1  <chr> "female", "male", "male", "male", "male", "male", "male", "
mal…
```

Well, those are two helpful functions we commit to our memory for future use. Now let's get to the main task: we want to do some serious sub-totalling. What we need to do is group the data. Now, each row has one data entry. That means, some information in the data is repeated. For instance, when a bird is caught more than once, it gets more than one lines. We want to count how often birds where caught only once, twice, thrice ect. To do this, the best thing to do would be first to sort all data by BirdID, the identifier of individual birds. Then, we'd count how many rows each BirdID occupies. Then, we'd group them again by how many double, triple ect. observations we have, until we have a good summary that fits our needs. That sounds complicated. I'm sure you could do this with your magical skills you learned last week. But we'll try another way:

With dplyr, we can "pipe" objects to another function. That is super helpful when we don't want to write lengthy code with if's and for's:

```
d$Mass %>% cor.test(d$Tarsus, na.rm=TRUE)

## 
##  Pearson's product-moment correlation
## 
## data:  . and d$Tarsus
## t = 22.374, df = 1642, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4454229 0.5195637
## sample estimates:
##       cor 
## 0.4833596
```

It takes the first object before the percentage sign (here
d
$Mass)andinsertsitasthefirstargumentintothefollowingfunction.Itevengetsinserted * before * d$Tarsus.

Let's see what we can do with this:

```
d1 %>%
  group_by(BirdID) %>%
  summarise (count=length(BirdID))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 618 x 2
##    BirdID count
##     <int> <int>
##  1      1     1
##  2      2    10
##  3      3     1
##  4      4     1
##  5      5     9
##  6      6     4
##  7      7     2
##  8      8     1
##  9      9     1
## 10     10     3
## # … with 608 more rows
```

Ok - let's slow down. Here, we take our data d, and then group it by BirdID (the identifier for each individual bird). Then we take this data and pass it on to the function summarise.

This is a super helpful function, it makes one row of many rows of data. Here, it takes each group of data on the same BirdID, and summarises it by counting how many lines are in this group.

The output is not very helpful. We see that well, bird 4 shows up 1 time in the dataset, bird 11 10 times, bird 13 only 1 time, and so forth. But it is not really helpful yet because we need to summarise this *again*, this time by count. Let's do that. I have to make a confession first: dplyr actually has a shorter way of doing this. I just didn't tell you right away because I wanted you to understand the group_by and summarise functions first. The shortcut to the above is this:

```
d1 %>%
  group_by(BirdID) %>%
  summarise (count=length(BirdID))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 618 x 2
##    BirdID count
##     <int> <int>
##  1      1     1
##  2      2    10
##  3      3     1
##  4      4     1
##  5      5     9
##  6      6     4
##  7      7     2
##  8      8     1
##  9      9     1
## 10     10     3
## # … with 608 more rows
```

On to further summarizing:

```
d1 %>%
  group_by(BirdID) %>%
  summarise (count=length(BirdID))  %>%
    count(count)

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 12 x 2
##    count     n
##    <int> <int>
##  1     1   222
##  2     2   147
##  3     3    88
##  4     4    48
##  5     5    47
##  6     6    26
##  7     7    15
##  8     8    10
##  9     9     6
## 10    10     7
## 11    11     1
## 12    12     1
```

You see, this gives us exactly the information we need. In our dataset of 1695 observations of wing length, 222 birds have been measured only once. So they are not very good data for getting within-group estimates. But 147 have been measured 2, and even more have been measured 3, 4, 5, up to 12 times! That is an amazing dataset of ***repeated measures***. Let's run the ANOVA now, and let's not forget to tell R that we use BirdID as a factor!

```r
model3<-lm(Wing~as.factor(BirdID), data=d1)
anova(model3)
```

```
## Analysis of Variance Table
##
## Response: Wing
##                     Df Sum Sq Mean Sq F value    Pr(>F)
## as.factor(BirdID)  617 8147.3 13.2047  8.1734 < 2.2e-16 ***
## Residuals         1077 1740.0  1.6156
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ok. Here we see clearly that there is statistically significantly more variation among groups (BirdID) than within-groups (residuals). The mean squares are much larger among groups than within. So here the difference between an ANOVA and a t.test becomes really clear: a t.test is about parameter estimates, the means, about the difference in size. An ANOVA is about the difference in variance among groups vs within groups. Because, clearly, any post-hoc test here would be still way too many comparisons for us to even consider interpreting. So instead of looking at means, we look at the variation within and among groups - that's a different way of describing data - instead of looking at centrality, we now look at spread. That's the important concept here to get clear with.

Now, when we look at the F statistic - 8 means the among variance divided by the within variance results in 8, that means, the amount of variance explained by differences among Birds (levels in the factorial variable) is 8 times more than the residual variance (the variance that is not explained by differences among birds). That means, individual birds are more likely to have a more similar wing length on re-measuring, than if you compare among birds. This makes sense - if you measure the same individual twice, the value will be more similar than if you measure two different individuals.

What else can we do with this sort of analysis?

We will examine the whole ANOVA bits again, from a bit of a different perspective. We will look at *repeatability*.

Another way of saying what we found here would be that individual bird have consistent wing length. They differ less between multiple measures than they differ from each other.

Another way of saying that is saying bird's wing length is *repeatable*. The statistical term is called *repeatablity*, or r, or "intraclass correlation coefficient", and it is important in several ways. It can be used to describe biological things, as we did here. It can however, also, be use to assess the quality of a method - to test for individual observer repeatability, as we did in the lecture.

The repeatability can be calculated in different ways. The simplest way is using the SS and MS of an ANOVA. A very good biologist and statistician, Kate Lessells, published a paper on this, it is called "Unrepeatable repeatabilities: a common mistake". This paper was rejected in many journals, and ultimately was published in a bird journal, "The Auk". It was published in 1987, and since has been cited about 3000 times. It has been very successful in teaching generations of biologists how to do the correct statistics. I strongly urge you to read it, maybe best today, as now your knowledge on ANOVA is fresh, and you will understand it much easier!

However, for now, I will also explain how to calculate the repeatability. It is given as

$$r = \frac{s_A^2}{(s_W^2 + s_A^2)}$$

Where s_A is the among-group variance and s_W is the within-group variance. So what we really do here is we calculate the ratio of variance of the total variance $(s_W^2 + s_A^2)$ that is explained by among-group differences. Among over total - that's the difference to F - which is the ratio of among variance explained over residual variance. And total variance explained is among + residual. Now, what this does, really is taking out the F statistic, and replacing it with something that is directly, biologically, interpretable.

We look at the % of variance explained by among-group differences. How can we do that? Remember how the ANOVA really only describes the variance partitions. We only used the sums of squares because the denominator canceled out. However, remember how calculating SS of among-groups was complicated, because we had to weigh for the sample size of each group? So far we only had balanced datasets, where each group had the same number of observations. So what can we do if that differs, and how does that influence the ANOVA results, and the resullting ratios?

That's what Kates paper is about - this group sample size is not as easy as one might think. It's easy for balanced group sizes. But if they are not balanced, you're in trouble. You can read this up in the paper, but for the record:

$$s_W^2 = MS_W$$

and

$$s_A^2 = \frac{MS_A - MS_W}{n_0}$$

The paper points out that many people used to just assume that we could just ignore n_0. That is wrong, because the sums of squares squares are not the variance unless we account properly for sample sizes of the groups. So we won't make this mistake. We know where the MS in the function come from, but what exactly is n_0 then? It is most often, actually not the sample size. It would be the sample size n if all group sizes were equal, say, we'd have 10 birds observed once, 10 twice, 10 thrice ect. That would be called a balanced dataset. However, much to our distress, in ecology and evolution, most datasets are far from balanced. Our datasets are often "dirty" and heterogeneous. As we saw earlier, we don't have balanced sample sizes for our BirdID ANOVA here. So we need to calculate n_0. It is calculated by a complicated function:

$$n_0 = [\frac{1}{a-1}[\sum_{i=1}^{a} n_i - (\frac{\sum_{i=1}^{a} n_i^2}{\sum_{i=1}^{a} n_i})]$$

This looks horribly complicated. But, as most things in life, it is not. Check this out: a is the number of groups. n_i is the sample size in each group. In our BirdID example, it's 618. 618 individual birds, and n_1 would be 1, n_2 is 10, and so forth. We have just done this when we explored the data structure:

```
d1 %>%
  group_by(BirdID) %>%
  summarise (count=length(BirdID))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 618 x 2
##    BirdID count
##     <int> <int>
##  1      1     1
##  2      2    10
##  3      3     1
##  4      4     1
##  5      5     9
##  6      6     4
##  7      7     2
##  8      8     1
##  9      9     1
## 10     10     3
## # … with 608 more rows
```

We could also find it out a in a more elegant way, using this code:

```
d1 %>%
  group_by(BirdID) %>%
  summarise (count=length(BirdID))  %>%
    count(count)

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 12 x 2
##    count     n
##    <int> <int>
##  1     1   222
##   2      2   147
##   3      3    88
##   4      4    48
##   5      5    47
##   6      6    26
##   7      7    15
##   8      8    10
##   9      9     6
##  10     10     7
##  11     11     1
##  12     12     1
```

Cool. So we know what a and n_i are. Next, we are told to do lots of sums. First, let's look at the inner most bracket, the fracture. We sum the square values of each group's n, and divide this sum by the unsquared sum of the ns of each group. That's easy enough, don't you think? Dplyr to the rescue:

```
d1 %>%
  group_by(BirdID) %>%
  summarise (count=length(BirdID)) %>%
  summarise (sum(count))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 1 x 1
##   `sum(count)`
##          <int>
## 1         1695
```

Uhh. That's the denominator = 1695. The numerator is similar, but we square the count (n) of each group first:

```
d1 %>%
  group_by(BirdID) %>%
  summarise (count=length(BirdID)) %>%
  summarise (sum(count^2))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 1 x 1
##   `sum(count^2)`
##            <dbl>
## 1           7307
```

Cool. That's 7307. Ok, R, this is easy, the fraction is:

```
7307/1695
```

```
## [1] 4.310914
```

Good. What's next? We're supposed to substract this fraction from the denominator. Clever, huh, we don't have to calculate that monster again. We certainly can do that:

```
1695-7307/1695

## [1] 1690.689
```

Even better. The last part is easy. a is 618, so it's really 1 over 617. We can do that:

```
(1/617)*(1695-7307/1695)

## [1] 2.740177
```

And that's our n_0, excellent. It is actually pretty close to 3, which you can see as a sort of a centrality measure of how many observations we have for each group. Because the differences are so extreme, we can't use a mean or so, we've got to use this monster of n_0. But since you've become so handy with sums of squares, you probably have some understanding where this comes from, and what it corrects for.

Let's finally calculate the repeatability! I already forgot the equation - I have really bad memory, I can only remember we needed to divide the part of variation that's explained by among group differences by the total variation, corrected for n_0. Here is the equation again:

$$r = \frac{s_A^2}{(s_W^2 + s_A^2)}$$

with

$$s_W^2 = MS_W$$

and

$$s_A^2 = \frac{MS_A - MS_W}{n_0}$$

and (monster)

$$n_0 = [\frac{1}{a-1}[\sum_{i=1}^{a} n_i - (\frac{\sum_{i=1}^{a} n_i^2}{\sum_{i=1}^{a} n_i})]$$

which is 2.74.

Here is out ANOVA result:

```
model3<-lm(Wing~as.factor(BirdID), data=d1)
anova(model3)

## Analysis of Variance Table
##
## Response: Wing
##                     Df Sum Sq Mean Sq F value    Pr(>F)
## as.factor(BirdID)  617 8147.3 13.2047  8.1734 < 2.2e-16 ***
## Residuals         1077 1740.0  1.6156
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So, really, the repeatability is:

```
((13.20-1.62)/2.74)/(1.62+(13.20-1.62)/2.74)

## [1] 0.7229006
```

The result of r is a fraction. We can express this as a percentage. Now we know that 72% of the variation in wing length is explained by among-individual differences. That equally means that individuals are relatively consistent in their wing length - most of the variation comes from differences among individuals, and only 100-72 - 18% of residual variance remains - within birds. Very cool. The concept of repeatability is an important one, and you should remember this for the rest of your life, as everything, really in this course. Over the time you spend here in Silwood, you will come across other methods of calculating the repeatability, but the principles stay the same: we look at the ratio of variance explained among vs between groups.

Excercises:

1) (you should have done this one already! But I'll repeat it here – don't do it a second time)

A researcher wants to know how the magic properties of the plant Aconite change with the environmental temperature the dried flower or leaves are stored in. She collected 100 leaves, and 100 flowers, and stored them in closed containers in a freezer (A), at room temperature (B), and in a heated room (C). She is now interested in the change in their magic properties with respect to their treatment. You are tasked with the statistical analysis. Per group, prepare two separate analyses - one using linear models and their results, and one using ANOVA. Conduct your analyses, then write a methods and results section for this experiment. Make sure to include a descriptive statistics section in addition to the results section. Per group, submit two documents (1 A4 each), both with methods and results.

2) We often use repeatability to assess how accurately an we can measure a trait. In that case, an reseracher measures the same trait twice or more often, on a number of individuals. Then we can calculate the repeatability of these measures. This is important to understand how much "noise" or randomness one can expect in ones data. You will find some data from someone conducting such an analysis on their measurements of fly morphometrics. The dataset[1] "Wylde_single.mounted.txt" consists of 22 morphological trait measurements in the neriid fly *Telostylinus angusticollis*, each of which has been measured twice. The fly ID is denoted in the column "ID". There is also a column "Sex" - this is important because certain traits are only measurable in females, and others only in males. For now, we ignore the columns that denote "Condition", and "Larval diet" - we can come back to these later on. In this exercise, you will calculate the repeatability for each of the 22 traits, and report them in a table. If you can, produce a code that prevents you from re-typing the same code - can you write a short function that loops through it? Submit one table with the

repeatabilities (in word or pdf) per group, make the table look professional, like in a paper.

[1]This data is from a paper on a preprint server, and courtesy of the author, Zac Wylde. If you want to know what these measurements are, check it out: DOI: 10.1101/2020.05.07.083428

Stats with Sparrows - 13

Julia Schroeder

Repeatability with linear mixed models, and beyond

I have promised you in the last hand out that there is an easier method to calculate repeatability - and that's with linear mixed models. These also work much better with unbalanced datasets - no need to figure out n naught ect. So let's have a look at these - we'll be using Zac's fly dataset for this. We need a new package - called lme4.

```
a<-read.table("Wylde_single.mounted.txt", header=T)
head(a)

library(lme4)

## Loading required package: Matrix
```

We'll investigate the repeatability of Femur length:

```
lmm1<-lmer(Femur_length~1+(1|ID), data=a)
```

Before we look at the summary statistics, we will take a closer look at the code:

The first bit makes sense - lmer is the function (instead of lm for a linear model). Then we have the response variable left of the tilde (~). Then there's a 1 - that's odd. You need to know here that if you do not want to fit a fixed effect, just estimate the intercept and nothing else - you write 1. It's something you need to know - that's how this is coded. I could have just as well put sex there or any other fixed effect - but because I want to get the repeatability without any other influence - I do not code a fixed effect. Then, we see something new - (1|ID). That is how in lmer we specify random effects - those that we want the model to use to partition the variance into. So here, we say - partition the variance into variance explained among ID, and residuals (we don't need to say we also wnat residual variance). Let's have a look at the summary statistics:

```
summary(lmm1)

## Linear mixed model fit by REML ['lmerMod']
## Formula: Femur_length ~ 1 + (1 | ID)
##    Data: a
##
## REML criterion at convergence: -247.9
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -3.09822 -0.38745 -0.00387  0.36983  3.08564
##
## Random effects:
```

```
##  Groups   Name        Variance  Std.Dev.
##  ID       (Intercept) 1.2570822 1.12120
##  Residual             0.0003399 0.01844
## Number of obs: 360, groups:  ID, 180
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  4.64978    0.08357   55.64
```

Let's examine the output. First, we're reminded of the model structure - only intercept fitted (1), and then ID as a random effect. The data is a. Then we get a REML convergence criterion - we'll ignore this for now. Then there are the residulas - and some information about their spread.

Next are the random effects - this is new. But it's fairly easy to interpret - we get the variance explained by among ID groups, and the residual variance. We will use these data later to calcluate the repeatability. For some reason that I still fail to understand, lmer also provides us with the standard deviation - we could just as well calculate this from the variance so it's a bit of a duplicate report.

We then get information about the data structure - and we can see there are 180 individuals, each measured twice give us a total sample size of 360. So this dataset is really balanced - and we could actually have used ANOVA anyways.

Next, we get information about the fixed effects - this should be familiar to you. Because we did only fit the intercept (1), this parameter estimate is the mean of the reponse variable.

Now let's calculate the repeatability. Remember - it's the among group variance explained, divided by the total variance. And the total variance is the among group variance plus the residual (or within-group) variance. :

```
Repeatability<-1.257/(1.257+0.0003)
Repeatability

## [1] 0.9997614
```

Excercises:

In the fly dataset, the repeatability is about measurement error - observers would measure the same thing twice to see how accurate they measure. However, we can also use repeatability to learn more about biology. The repeatabiliy of a trait in individuals who are captured at multiple points during their lives, tells us something about the phenotypic flexibility of these traits. For instance, if we'd capture a sparrow multiple times during their life, we would expect their body mass to change quite a bit, but a trait that doesn't change across time, like tarsus length, shouldn't change much. However, in both cases, we'd still expect some consistency within individuals, and larger differences between individuals (as we've seen in the last handout). (1) Use the linear mixed model method to calculate repeatability of tarsus, wing and mass in house sparrows individuals, and think about biological reasons for why the estimates differ. (2) Random effects in linear mixed models are not only useful for estimating the variance components. In addition to getting that information, modelling a factor that is repeated in the data also accounts for something that is called “pseudo-replication”. The problem with having a repeated-measures dataset like the sparrows is that we often enter the same individual more than once into a model. This can be problematic for simlar reasons as the ANOVA gets off with unbalanced datasets - think about it - imagine a dataset of ten indiviudals, but where one indiviudal is measured 1oo times, and the others each only twice. The one individual will inform the model overly much, and that would not give the correct result. If, however, we account for the variance within individuals by modelling ID as random effect, we account for that bias, and the resulting fixed effects will be reliable. So, that’s another reason to know your data structure - if there’s pseudo-replication in your dataset in a grouping factor (e.g. by plots, individuals, observers, years ..) it’s a good idea to model that as random effect. Your task now to is to play with this information - use the sparrow dataset and re-run models that you’ve ran before, but this time, bird ID as random effect. Compare the results between “simple” linear models without the random effect, and those with. (3) Continue your exploration from (2), but this time, add a second random effect - use also year as a factor on the random part of the model.

Stats with Sparrows - 14

Julia Schroeder

All the models

If you've made it to here, - you're done really really well. You will know by now more than many biologists do. You have all the skills, and all the tools, to run your own analyses. This is the last hand out, and all it provides is a few ideas that you can use to hone your skills - but do not stop at my suggestions! Use the datasets provided and explore them statistically. Think about relationships you can explore. Think about data structure, and run models accordingly. This sort of explorative data analysis is important to better understand your data, find patterns and practise your skills.

While you do this, discuss your finding, and approaches, with the group, GTA and professor.

Here are a few research questions as suggestions to get you started:

1) What is the observer repeatability of the house sparrow ornament measurement? datafile: "OrnamentAge.txt"
2) In Telostylinus angusticollis, are females larger than males? In which traits? Wylde dataset.
3) Does house sparrow body mass change with age? Use the SparrowSize dataset, and assume the first year they are observed is the year they are born. This requires some data wrangling - you need to create a variable age for this dataset.

Here is a past exam question for you to practice. Discuss solutions in the group, and ask your GTA and the professors for feedback.

Exam question:
You are interested in the effects of climate change on bird's timing of breeding. You spend the last 4 years collecting data on the date the birds lay their first egg of the first clutch for an individual female in a given year. Your study species is the climate-change sensitive golden phoenix (*Phoenix potterus fawkes*), whose eggs burst into flames and smoke when they have not hatched by April 20. So with ongoing climate change the hope is that more of them may be able to survive if they could only breed earlier.
You collected data on individual birds, recording the egg laying data in days from 1st March. This way, 14 is March 14, and 36 is April 6, and so forth. You collected this data over the course of four years, between 2006 and 2009. You want to analyse whether laying date changed over the course of the years, in particular, whether it decreased. You use two main approaches for data analysis, and you can find below the R command, and the R output. With the below R output, you will deduce the analysis strategy, and write a methods and results section as you would for your thesis
Use all your knowledge from the R course on how to communicate statistics!

```
> length(PhoenixData$LayingDate)
[1] 108
```

```
> var(PhoenixData$LayingDate)
[1] 539.0041

> summary(PhoenixData$LayingDate)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   6.00   31.75   43.00   48.62   65.25  114.00 

> table(PhoenixData$year)
2006 2007 2008 2009 
  46   33   10   19 

> summary(lm(LayingDate~as.factor(year), data=PhoenixData))
Call:
lm(formula = LayingDate ~ as.factor(year), data = PhoenixData)

Residuals:
    Min      1Q  Median      3Q     Max 
-29.804 -16.000  -2.452  12.397  61.697 

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)    
(Intercept)           56.804      3.076  18.467  < 2e-16 ***
as.factor(year)2007   -4.501      4.759  -0.946  0.34643    
as.factor(year)2008  -20.704      7.279  -2.844  0.00536 ** 
as.factor(year)2009  -27.804      5.689  -4.887 3.73e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 20.86 on 104 degrees of freedom
Multiple R-squared:  0.2152,	Adjusted R-squared:  0.1925 
F-statistic: 9.505 on 3 and 104 DF,  p-value: 1.325e-05

> summary(lm(LayingDate~year, data=PhoenixData))
Call:
lm(formula = LayingDate ~ year, data = PhoenixData)

Residuals:
    Min      1Q  Median      3Q     Max 
-31.247 -15.118  -4.021  11.753  65.205 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 19017.846   3635.037   5.232 8.52e-07 ***
year           -9.451      1.811  -5.218 9.03e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 20.81 on 106 degrees of freedom
Multiple R-squared:  0.2044,	Adjusted R-squared:  0.1969 
F-statistic: 27.23 on 1 and 106 DF,  p-value: 9.026e-07
```