Additional Lecture Notes for Statistics for Biology

Stephan Peischl

Contents

Introduction

These notes should help you get started with the statistics software R, guide you through the practical part of the course and provide additional information on selected topics. Before you continue reading, make sure you have R and R Studio installed on your computer. Try to use R as an calculator as shown in the lecture slides and try to get yourself accustomed to the different windows in the R Studio environment. No prior knowledge about R is necessary and once everything is up and running you can dive right in.

We start by loading some relevant packages for this course. An R package is a collection of functions, data, and documentation that extends the capabilities of base R.

```
# make sure you have th packages installed using
# install.packages(packagename)

library(tidyverse)
library(Rmisc)
library(tibble)
library(dplyr)


# here i define a color palette to use as standard when plotting with ggplot
# this is purely for aesthetic reasons

cbp2 <- c("#000000", "#E69F00", "#56B4E9", "#009E73",
          "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

scale_colour_discrete <- function(...) {
  scale_colour_manual(..., values = cbp2)
}
```

Visualizations with ggplot2 and base R

GGplot 2

We start with some examples of how to visulaize data quickly and easily. This should show you how to produce high quality visulaizations using R and motivate you to learn more about programming and data analysis with R.

R has several systems for making graphs, and I have chosen to start with ggplot2 beacuse it is elegant, versatile and is easy to leanr if you have no prior knowledge about programming. For those of you who are already familiar wiht programming langauges, doing graphics in "base R" may seem more intuitive initially but I hope I can convince you that the grammar of plotting used by ggplot has lots of advantages and is very readable once you get the hang of it.

If you like to learn more about the theoretical underpinnings of ggplot2 before you start, I would recommend reading "The Layered Grammar of Graphics", http://vita.had.co.nz/papers/layered-grammar.pdf.

The MPG dataset

Let us use our first graph to answer a question: Do cars with big engines use more fuel than cars with small engines? What does the relationship between engine size and fuel efficiency look like? Is it positive? Negative? Linear? Nonlinear?

We can find an answer with the mpg data set found in ggplot2 (aka ggplot2::mpg). A data frame is the R data format to store data in a spreadsheet: a rectangular collection of variables (in the columns) and observations (in the rows). The data frame mpg contains observations collected by the US Environmental Protection Agency on 38 models of car.

```
# the command head() shos us the first few rows of the data set to
# get an overview of what is stored in our dataset

head(mpg)
```

```
## # A tibble: 6 x 11
##   manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
##   <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
## 1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
## 2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
## 3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
## 4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
## 5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
## 6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

Among the variables in mpg are:

displ, a car's engine size, in litres.

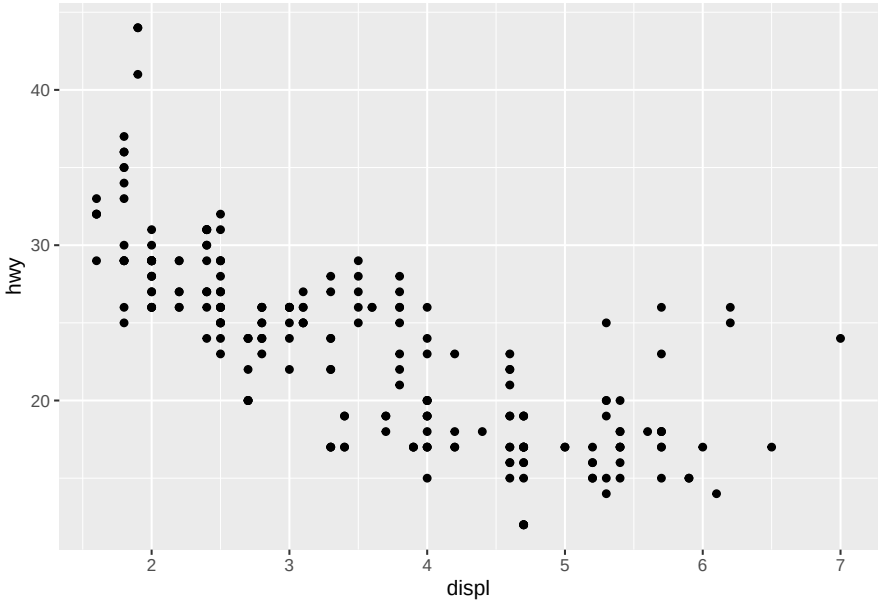
hwy, a car's fuel efficiency on the highway, in miles per gallon (mpg). A car with a low fuel efficiency consumes more fuel than a car with a high fuel efficiency when they travel the same distance.

To learn more about mpg, open its help page by running ?mpg. in the next seciton we learn how to visualize such data.

The grammar of ggplot2

In general you begin a plot with the function *ggplot()*. ggplot() creates a coordinate system that you can add layers to. The first argument of ggplot() is the dataset to use in the graph. For instance, typing ggplot(data = mpg) into the console creates an empty graph (which is boring so it is not shown here). You can now add layers to your plot. For example, the function geom_point() adds a layer of points to your plot, thus creating a scatterplot.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

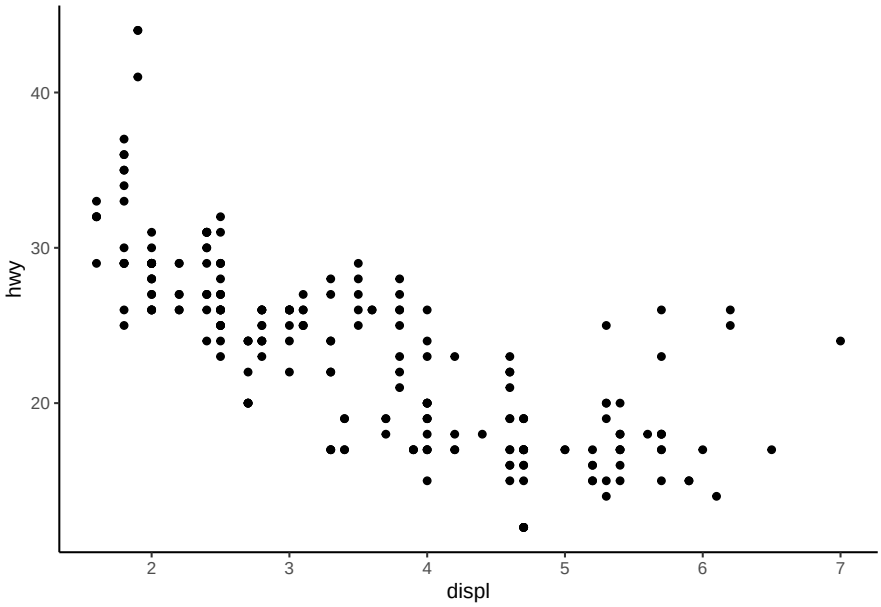


There are many other such *geometric functions* withing the ggplot2 universe and you can combine them in a single plot. You will learn a whole bunch of them throughout this course, but be aware that we will only scratch the surface of what is available. After this course, you should be ready to equip your self with new tools for data analysis whenever you need them.

Each geom function in ggplot2 takes a mapping argument. This defines how variables in your dataset are mapped to visual properties. The mapping argument is always paired with aes(), and the x and y arguments of aes() specify which variables to map to the x and y axes. ggplot2 looks for the mapped variables in the data argument, in this case, mpg.

Next, we use a different theme beacuse this gray background in the standard design is ugly.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  theme_classic()
```



The general grammar of ggplot is

ggplot(data = <DATA>) +

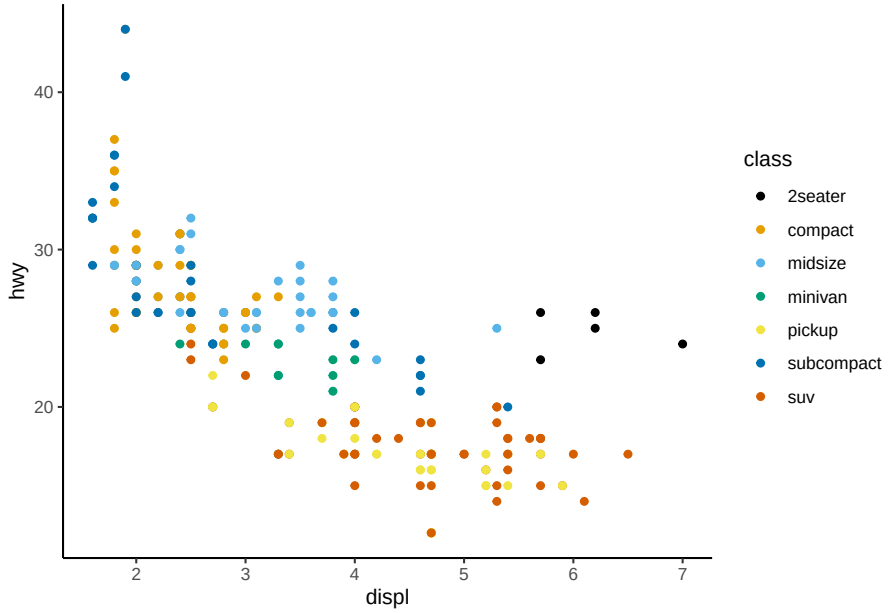
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))

We will explain each part step by step in examples. The DATA argument should be quite clear: here you specify the dataframe (we will learn more about that alter) that you want to work with.

The second part, GEOM_FUNCTION, specifies what kind of plot you want. As mentioned above *geom_point* gives you a classic scatter plot. Many other such funtions exist and we will encounter some of them.

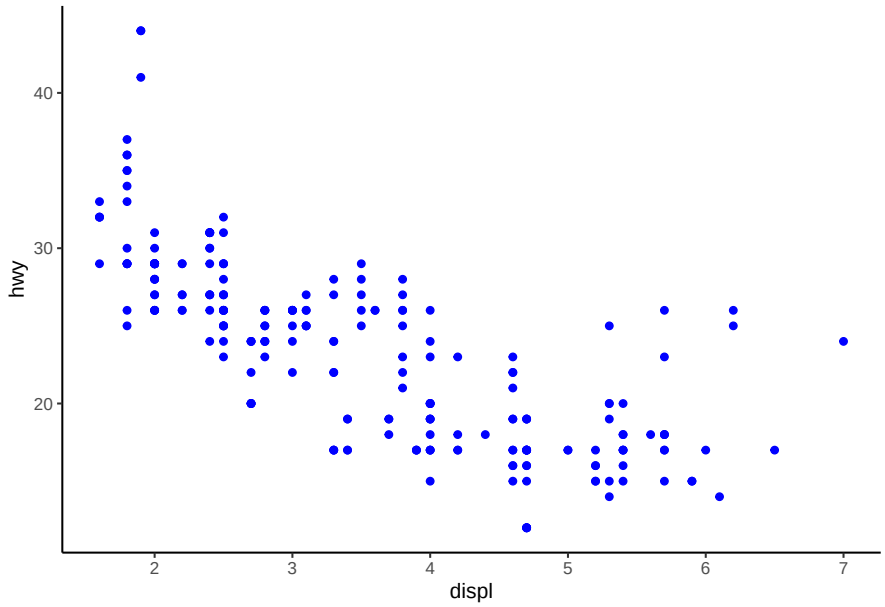
Finally, MAPPING specifies which variables hould be plotted. We ussually specify this with an aesthetics mapping of the form *mapping = aes(x = . . . , y =. . .)* to deterimne the variables that give us the x- and y-coordinates. We can however also change the aesthetics of our plot in various way, for instance by giving data points different colors that indicate the value of a third variable:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy,color = class)) +
  theme_classic()
```



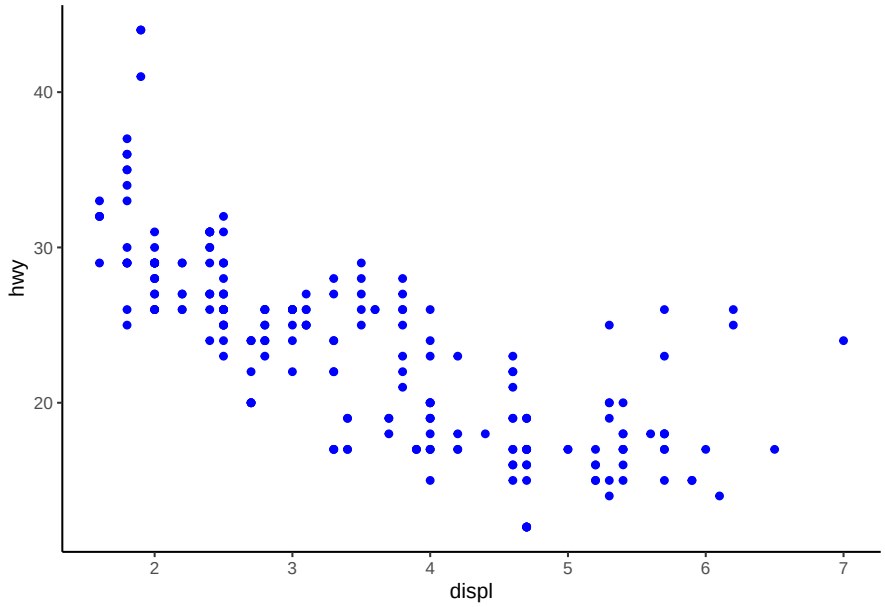
Another important option is to set a clor manually:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue") +
  theme_classic()
```

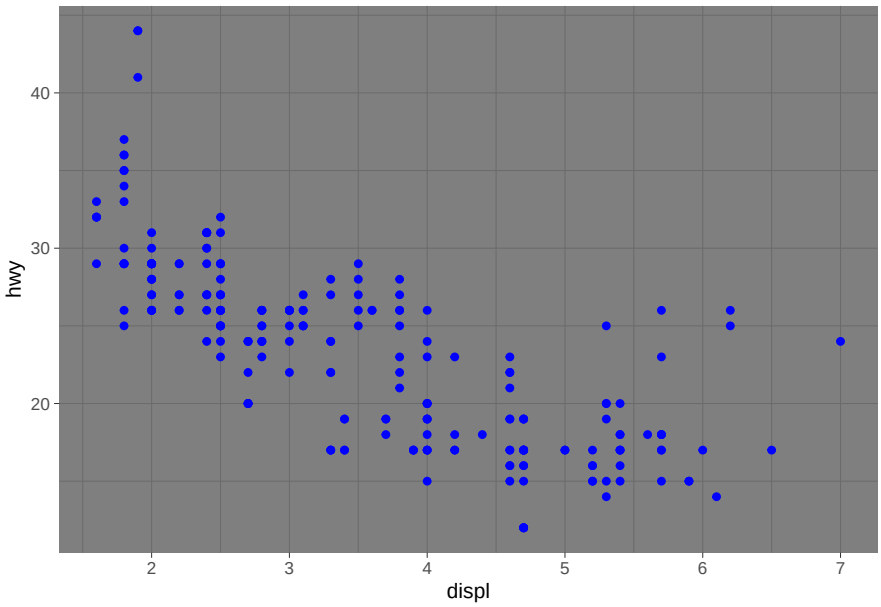


As a final remark, you can also create a plot, store it in a variable and then explore themes after that:

```
p = ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue")

p+theme_classic()
```




```
p+theme_dark()
```



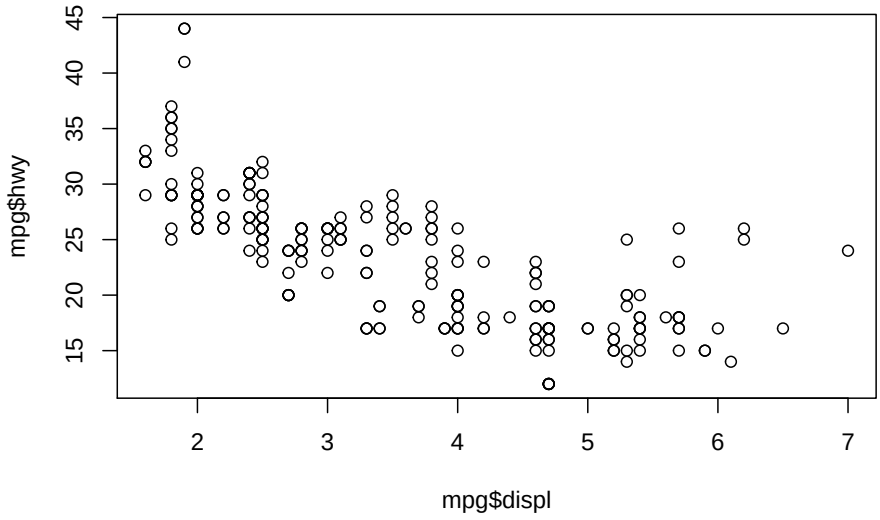
We now set our prefered them globally, so we do not need to specifiy it all the time:

```
theme_set(theme_classic())
```

Base R graphics

Base R refers to the standard R functions. While ggplot allows us to make sophisticated graphs very quickly, I find that in some situations base R is just as good or even better. Let us recreate the same plot as before:

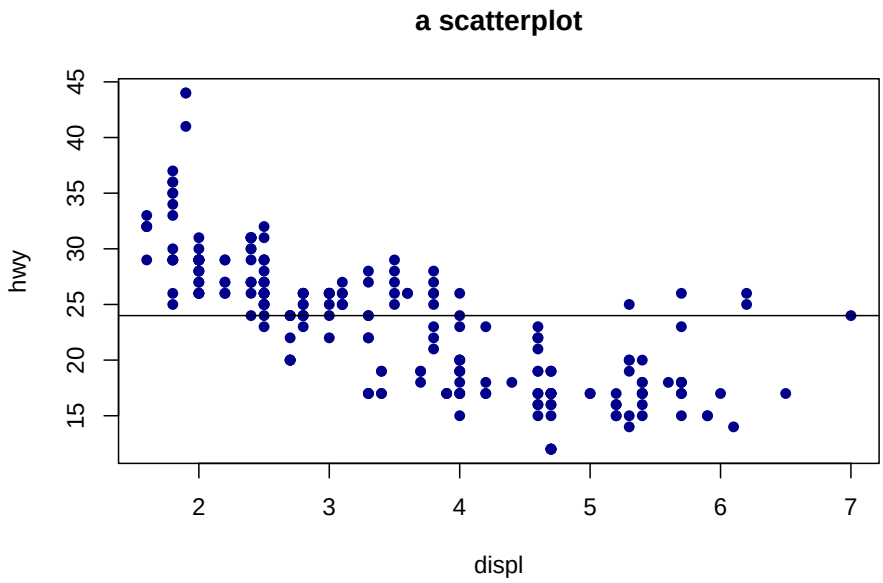
```
plot(mpg$displ,mpg$hwy)
```



Note how we use the $ symbol to access the variables stored in a data frame here; it allows you to access a specific column of a data frame. Try typing the name of a dataframe, followed by the $ operator in RStudio; it should automatically suggest you the availbale variable names in your data frame.

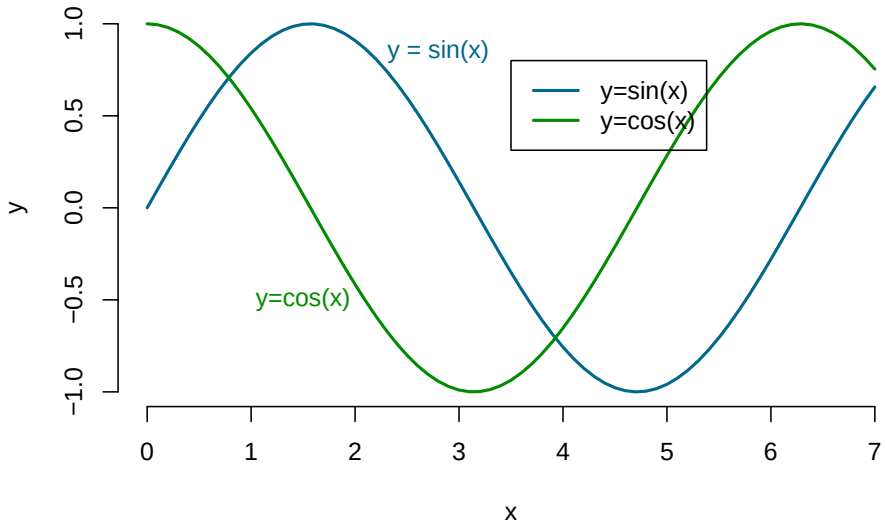
We can alos add stuff to this plot, change colors, and pretty much everything else we would like to. A quick look at the help function should give you an overview about the scope of the function plot and it's parameters. For instance, have a look at the following plot, where I make a scatterplot and modifiy the appearance of the symbols, add some labels and a title, and then add a horizontal line at the value of the median fuel efficiency:

```
plot(mpg$displ,mpg$hwy,pch=16,col="darkblue",xlab="displ",ylab="hwy", main ="a scatterplot")
abline(h = median(mpg$hwy))
```



When working with data frames, ggplot may seem more attractive and easy to use than base R functions, simply because many steps are automated. You may want to try color-coding the different points according to the varibale class, just as we did before. Then try to add a legend (look up the function *legend* in the R help). You will find out that it may be a bit tedious to put all this things together. However, it also gives you a lot of control about the final looks of your figure once you have figured out what you want and how you get it. Another advantge of base R functions is that you are not restircted to the data frame format, which often makes things easier. For instance, consider this plot:

```
x=seq(0,7,by=0.1)
# x = (0,0.1,0.2,0.3, ... ,6.9,7)
y=sin(x)


plot(x,y,lwd=2,type="l",col="deepskyblue4",axes=FALSE)
#plot without axes

axis(1) #add axes to plot
axis(2)

text(2.8,0.85,"y = sin(x)",col="deepskyblue4")
# add some text at coordinates x = 2.8, y = 0.85

lines(x,cos(x),lwd=2,col="green4")  # add a cosine plot
text(1.5,-0.5,"y=cos(x)",col="green4")   # and label it

legend(3.5,0.8,legend=c("y=sin(x)","y=cos(x)"),
col=c("deepskyblue4","green4"),lwd=2)
```




```
# add a legend to the plot
```

Instead of using a data frame, we have created vectors of variables and used them in several different ways. You will explore this more in the exercise sessions.

By now you should see that R is very versatile and can be used for much more than simply doing statistics. This course will teach you the basics of R so that you can start using it for your research projects. An essential part is that you should also learn how to find and use additional resources, such as the help function in R, books, the internet, etc. Nobody knows all of R - it is an ever growing community endevaour and looking up how things work, which functions or packages are available, or how other people solved a problem is an essential part of learning to be proficient in R.

Working with data frames

Loading datasets

How to we feed R our data? Of course, we can just type it into the console, a bit like in the previous base R graphics example. For large data sets (or rather anythign that is not an eytremely small data set) this is not practical. We want to load our data from a file, which makes our analysis easily reproducible if we share data and R scripts with someone. First we need to make sure R knows where our working directory is, using the function *setwd* (set working directory):

```
setwd("/Users/stephan/Dropbox/Teaching/BlogdownPages/StatisticsForBiology")
```

If you want to check your current working directory, you can do this with *getwd()*.

I use a dataset about the number of plates on sticklebacks. You can find this dataset (and many others) on ilias. We can load a dataset using the function *read.csv* and store it in the variable stickleback

```
stickleback = read.csv("chap03e3SticklebackPlates.csv")
```

The variable *stickleback* now contains a so called data frame. We can have a quick summary of the content of this data frame using the command *str* (short for *STR*ucture):

```
str(stickleback)
```

```
## 'data.frame':    344 obs. of  3 variables:
##  $ id      : Factor w/ 344 levels "4-1","4-10","4-100",..: 1 102 282 293 2 22 42 131 212 280 ...
##  $ plates  : int  11 63 22 10 14 11 58 36 31 61 ...
```

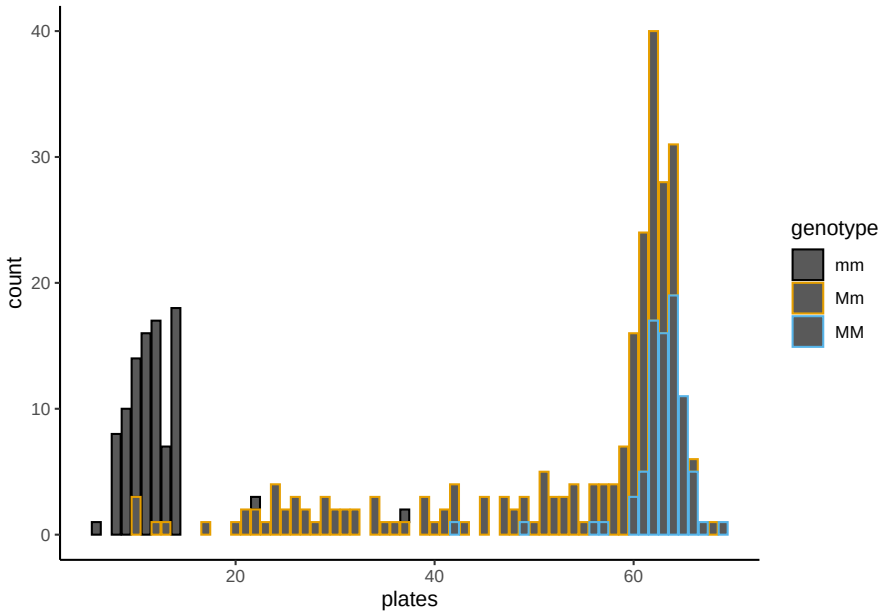
```
##  $ genotype: Factor w/ 3 levels "mm","Mm","MM": 1 2 2 2 1 1 2 2 2 2 ...
```

We can see form the output that we have 344 observations (= sample size) and for each individual we have measured 3 variables (id, plates and genotypes). ID and genotpye are so-called factors, that is, a catergorical variable and plates is of type integer.

We used the function *read.csv* because our data is stored in the CSV format (short for *c*omma *s*eperated *v*alues - open the file in a text editor to see what that means). CSV is a common format and you can export your data from software like Excel in that format. Of course, R provides functions for other formats as well or you an use the read.table() function where you can specify how the data should be read and transformed into a data frame.

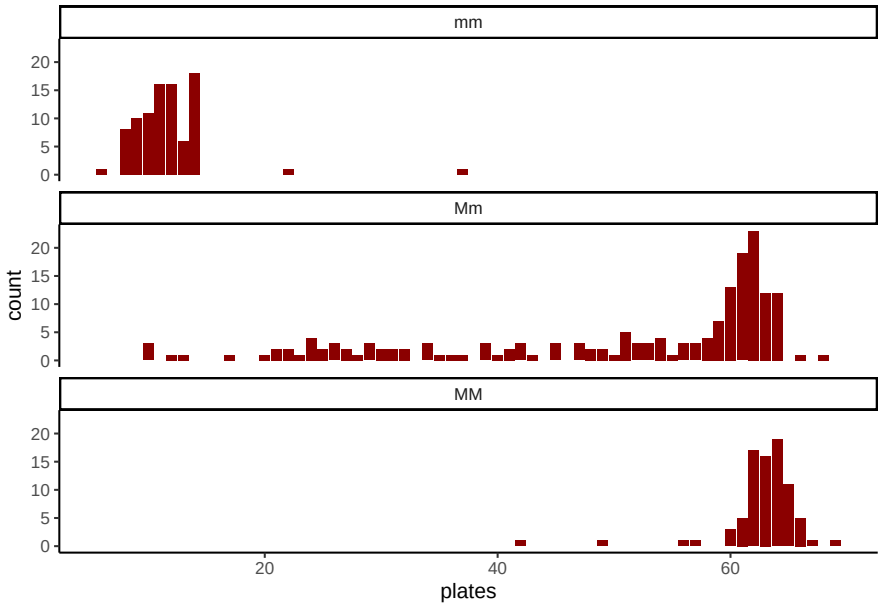
Let us use our ggplot skills to create a nice figure that visualizes our data set. For this we introduce the fucntion *geom_bar()* which allows us to make a bar plot of the number of plates per indivudal, colored by genotype:

```
ggplot(data = stickleback) +
  geom_bar(mapping = aes(x = plates,color=genotype)) +
  theme_classic()
```



This figure is a bit messy. A better way would be to show the distribution for each genotype in its own window. This can be done using the function *facet_wrap*. As before, we can simply add this to our ggplot command:

```
ggplot(data = stickleback) +
  geom_bar(mapping = aes(x = plates),fill="darkred") +
  facet_wrap(~genotype,nrow = 3) +
  theme_classic()
```



We used a new symbol here: the ~. This is part of an R object called formula and it will alter become clear why we used it here like this. For now, just remember to use it in front of the varibale name in facet wrap. The second aprameter is simply the number of rows in the plot. Try changing it and see what happens. Note: the variable that you pass to facet_wrap shoudl **always** be categorical!

Manipulating data frames

A data frame is essentially a spreadsheet or a table (or rather a matrix). Therfore, we acess each column or each row, if we want to. This can be done using the *[,]* symbols. The element in row 3 and column 5 is given by:

```
stickleback[3,5]
```

```
## NULL
```

the whole 3rd column can be accesses by

```
stickleback[,3]
```

```
##   [1] mm Mm Mm Mm mm mm Mm Mm Mm Mm Mm mm mm mm Mm Mm mm Mm Mm MM Mm Mm mm Mm MM
##  [26] Mm mm MM Mm Mm MM mm Mm Mm mm Mm mm MM Mm mm Mm MM mm Mm Mm MM Mm mm mm Mm
##  [51] Mm MM mm mm mm Mm MM MM Mm MM MM Mm MM Mm mm mm MM Mm Mm MM MM Mm MM Mm MM
##  [76] MM Mm Mm Mm mm Mm Mm Mm Mm MM Mm Mm Mm Mm MM mm Mm Mm mm MM Mm mm MM mm mm
## [101] Mm mm mm Mm MM Mm mm mm Mm Mm MM MM Mm mm Mm MM MM Mm mm Mm Mm MM mm MM mm
## [126] mm Mm Mm Mm mm Mm mm Mm mm Mm MM Mm Mm Mm Mm MM Mm MM Mm Mm Mm mm MM Mm MM
## [151] MM MM Mm Mm mm mm Mm MM MM Mm Mm mm Mm Mm mm Mm mm Mm Mm MM Mm MM Mm MM MM
## [176] Mm mm MM mm Mm Mm Mm Mm Mm Mm Mm mm Mm MM Mm Mm MM mm MM Mm mm Mm Mm Mm Mm
## [201] Mm Mm Mm Mm Mm Mm MM MM Mm Mm mm Mm mm mm Mm MM Mm Mm Mm Mm mm mm Mm Mm Mm
## [226] mm MM MM MM Mm Mm mm MM MM MM MM Mm Mm MM Mm MM mm mm Mm Mm Mm MM mm mm MM
## [251] mm Mm MM Mm Mm mm mm Mm Mm MM mm Mm Mm mm Mm Mm Mm Mm mm MM Mm mm Mm MM Mm
## [276] mm Mm Mm MM mm mm Mm MM Mm Mm mm MM mm Mm Mm Mm MM mm MM Mm Mm mm MM MM mm
## [301] MM MM MM Mm Mm MM mm mm MM MM mm Mm Mm Mm Mm MM mm Mm Mm Mm Mm Mm MM Mm mm Mm
## [326] Mm Mm Mm Mm Mm mm MM mm mm Mm Mm Mm mm Mm mm MM mm Mm mm
## Levels: mm Mm MM
```

and the 5th row by

```
stickleback[5,]
```

```
##     id plates genotype
## 5 4-10     14       mm
```

Note that stickleback[[,3]] gives you all measurment of the 3rd variable in our dataframe and is hence equivalent to stickleback$genotype. The vector stickleback[5,] on the other hand gives you a vector with the 3 measured values of the 5 th individual.

We can also use this way of accessing elements to find elements of a certain type, for instance, fi we want to know the genotype of all individuals with more then 30 plates, we can get this in the following way:

```
stickleback[stickleback$plates>20,3]
```

Try to understand what is happening here by teasing apart and investigating the differnt parts of this (nested) command. What would happen if you removed the *stickleback$plates>20* from the command, or the *3*.

A slightly more elegant way to get this is by using the function *filter* (make sure you have the package dyplr loaded!):

```
filter(stickleback,plates>20)
```

which gives ou a data frame with only the individuals that satisfy the condition set in the argument.

Adding variables to data frames

Sometimes we wish to extend our dataframe with calcualtions we did during our analysis or with additional measurements. This can be done with the function *mutate()*. It simply adds new columns at the end of your dataset. Let's say we want to identify which indivduals have fewer or more plates as comapred to the global mean:

```
diff.to.mean = stickleback$plates - mean(stickleback$plates)
stickleback2 = mutate(stickleback,difference = diff.to.mean)
head(stickleback2)
```

```
##     id plates genotype difference
## 1  4-1     11       mm  -32.43314
## 2  4-2     63       Mm   19.56686
## 3  4-4     22       Mm  -21.43314
## 4  4-5     10       Mm  -33.43314
## 5 4-10     14       mm  -29.43314
## 6 4-12     11       mm  -32.43314
```

Subsetting a data frame

We have alreaddy seen how to look at a subset of your data frame using indexing. Here is another slightly more elegant way of doing this, especially when many conditions are combined over multiple variables. For instance, what if we want to consider only stickleback with more than 20 plates:

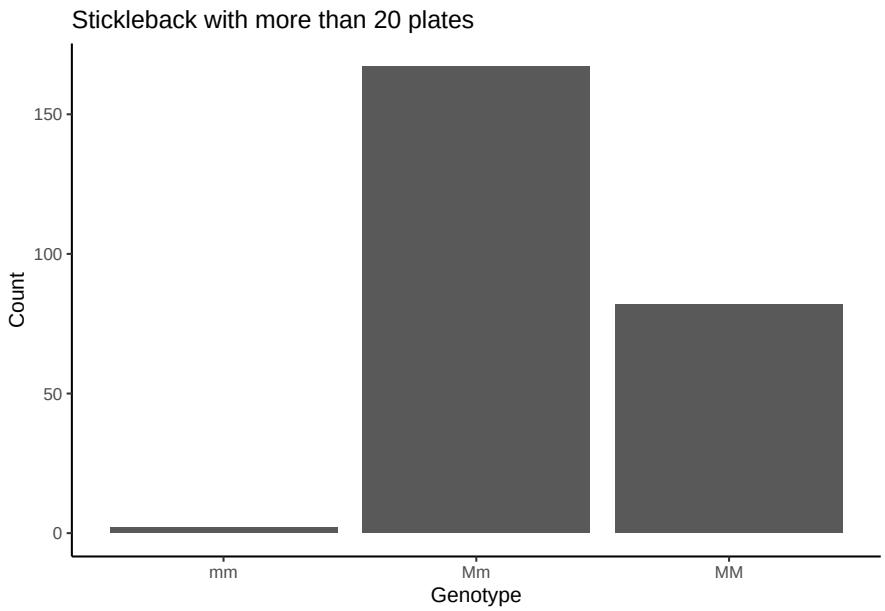
```
head(subset(stickleback, plates>20))
```

```
##      id plates genotype
## 2   4-2     63       Mm
## 3   4-4     22       Mm
## 7  4-14     58       Mm
## 8  4-23     36       Mm
## 9  4-31     31       Mm
```

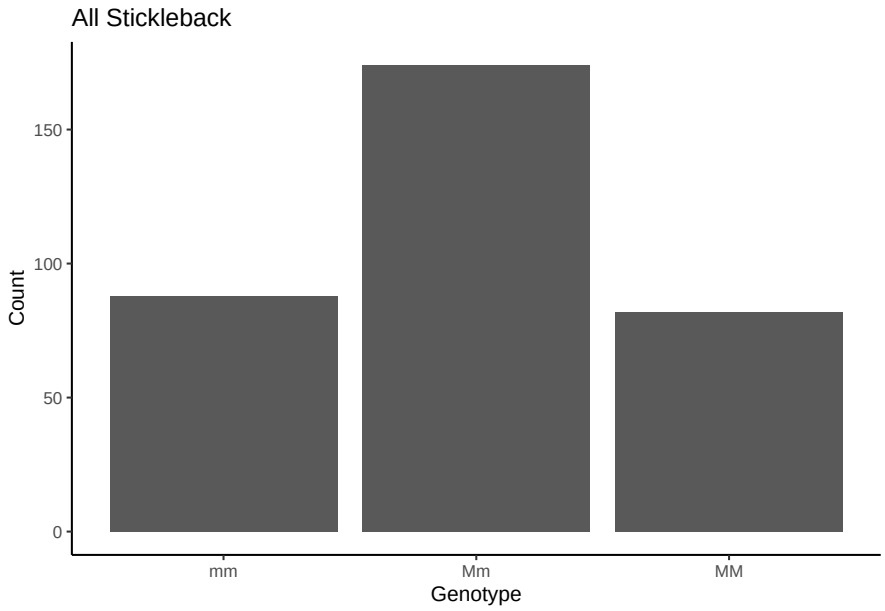
```
## 10 4-38     61       Mm
```

We could now compare how the disitrbution of genotypes changes if we condition on more than 10 plates:

```
ggplot(data = subset(stickleback, plates>20)) +
  geom_bar(mapping= aes(x = genotype)) +
  labs(title="Stickleback with more than 20 plates",
        x ="Genotype", y = "Count")
```




```
ggplot(data = stickleback) +
  geom_bar(mapping= aes(x = genotype)) +
  labs(title="All Stickleback",
        x ="Genotype", y = "Count")
```



There are a few *mm* genotypes with more than 20 plates. But how many?

```
subset(stickleback,plates > 20 & genotype == "mm")
```

```
##       id plates genotype
## 100 4-25     37       mm
## 132 4-87     22       mm
```

It turns out there are exactly two: one with 37 plates and one with 22. We have used logical operators here: "&" means that both conditions need to be satisfied. In contrast, try to see what the symbol " | " means:

```
head(subset(stickleback,plates > 20  |  genotype == "mm"))
```

```
##     id plates genotype
## 1  4-1     11       mm
## 2  4-2     63       Mm
## 3  4-4     22       Mm
## 5 4-10     14       mm
## 6 4-12     11       mm
## 7 4-14     58       Mm
```

You may have guessed it already, the " | " represetns the logical "or" command, indicating that either of the two conditions needs to be satisfied. Note that for logical comparions we use the "==" and not the "=" symbol, which is reserved for assignments. What is the difference? Try it:

```
a = 5

a == 5
```

```
## [1] TRUE
```

```
a == 6
```

```
## [1] FALSE
```

We can also use the symbols < (smaller), > (larger), <= (smaller or equal), >= (larger or equal), != (not equal) and combine them into logical statements.

Practice Questions:

- Add a new varaible to the stickleback data frame that contains the difference between plate number and the mean plate number for that individuals genotype.
- Load the flights dataset:

```
library(nycflights13)
```

Find all flights that

- Had an arrival delay of two or more hours
- Flew to Houston (IAH or HOU)
- Were operated by United, American, or Delta
- Departed in summer (July, August, and September)
- Arrived more than two hours late, but didn't leave late
- Were delayed by at least an hour, but made up over 30 minutes in flight
- Departed between midnight and 6am

Calculating Descriptive Statistics

We continue using the stickleback data set and calcualte some statistics on it. A simple way to get a quick overview about the properties of your data frame is using the funciton *summary*:

```
summary(stickleback)
```

```
##        id          plates      genotype
##  4-1    :  1   Min.   : 6.00   mm: 88
##  4-10   :  1   1st Qu.:14.00   Mm:174
##  4-100  :  1   Median :57.00   MM: 82
##  4-101  :  1   Mean   :43.43
##  4-103  :  1   3rd Qu.:62.00
##  4-105  :  1   Max.   :69.00
##  (Other):338
```

If you want to calcualte a specific statistic, like the mean of a variable, this can be done easily:

```
mean(stickleback$plates)
```

```
## [1] 43.43314
```

You may have noticed that we have no calcualted the mean number of plates for *ALL* fish in our data set. What if we want to calcualte the number of plates for a specific genotype? We can use the function filter:

```
stickleback.mm = filter(stickleback,genotype=="mm")
mean(stickleback.mm$plates)
```

```
## [1] 11.67045
```

You could now calcualte the mean plate numbers for each genotpye in this way:

```
mean.mm = mean((filter(stickleback,genotype=="mm"))$plates)
mean.mM = mean((filter(stickleback,genotype=="mM"))$plates)
mean.MM = mean((filter(stickleback,genotype=="MM"))$plates)
```

Note: The tidyverse offers an elegant way to write a long series of commands in a very readable format. To show you how to combine multiple commands quickly, we first need two more functions, *group_by* and *summarize*, and a new operator called the pipe: %>% (because we create a pipeline thorugh which our data "flows"). The above code to calculate the mean for each genotype would become:

```
stickleback %>%
group_by(genotype) %>%
summarise(avg = mean(plates))
```

```
##        avg
## 1 43.43314
```

This is very readable: you take the dataframe stickleback, group it by genotye and summarize it by calcualting the mean number of plates. However, errors might be more difficutl to spot when the code is written in that way, as intermediate steps cannot be checked.

In the exercises you will learn a few more useful functions to rearrange, filter, extend and edit data frames.

Practice Question: Can you come up with anotehr way to do this whithout using filter?

Two possible answers (there are many ways to do this): Straightforward "indexing":

```
mean.mm = mean(stickleback$plates[stickleback$genotype=="mm"])
mean.mM = mean(stickleback$plates[stickleback$genotype=="mM"])
mean.MM = mean(stickleback$plates[stickleback$genotype=="MM"])
```

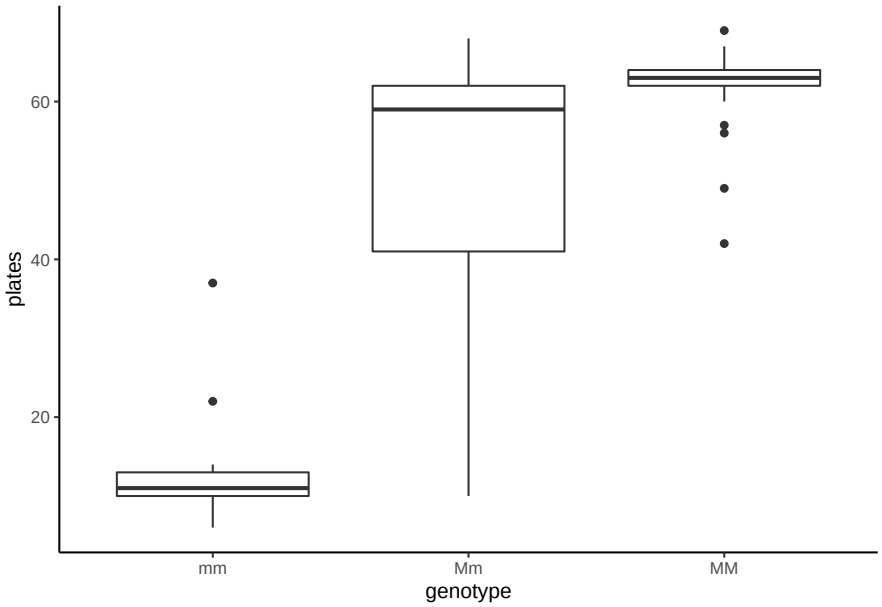
or by using the function *tapply* (look up the help page for this function by typing "?tapply"):

```
mean.plates = tapply(stickleback$plates,stickleback$genotyp,mean)
```

Plots

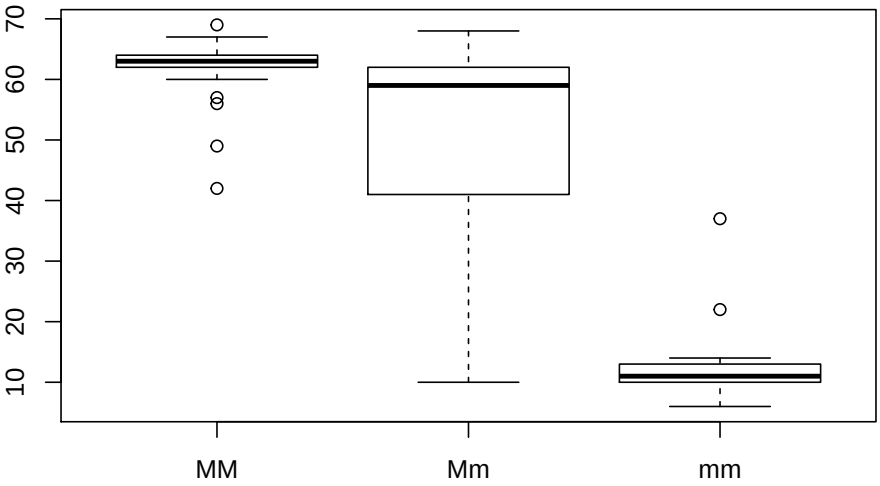
We can do a boxplot easily using ggplot:

```
ggplot(data = stickleback) +
  geom_boxplot(mapping = aes(x = genotype, y = plates))
```



In base R, this can be done as well:

```
boxplot(stickleback$plates[stickleback$genotype=="MM"],
        stickleback$plates[stickleback$genotype=="Mm"],
        stickleback$plates[stickleback$genotype=="mm"],
names = c("MM","Mm","mm")
)
```

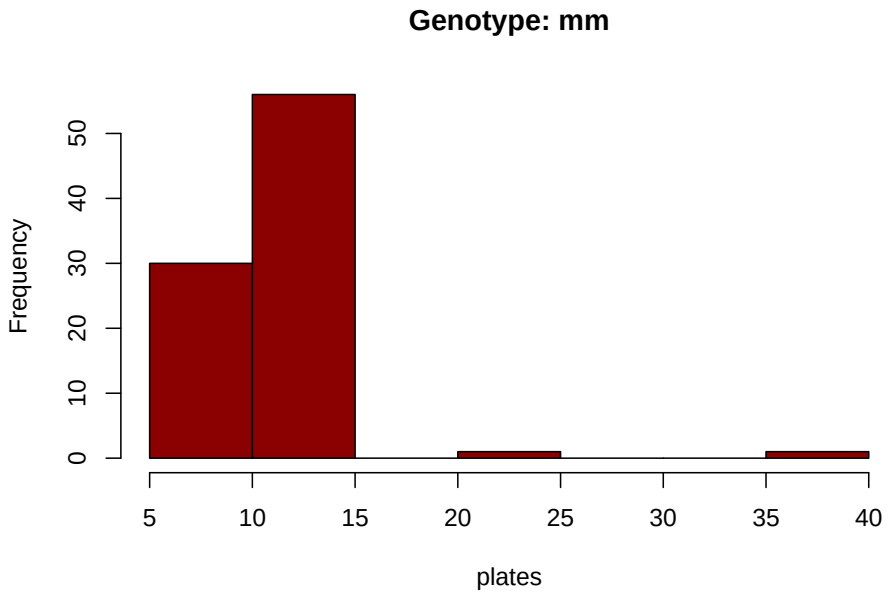


Practice Question: Can you plot a histogram of plate numbers of individuals with genotpye mm using base R

plotting functions? Hint: Use what you have learned here and look up the function hist(). Alternatively, use ggplot and geom_histogram.

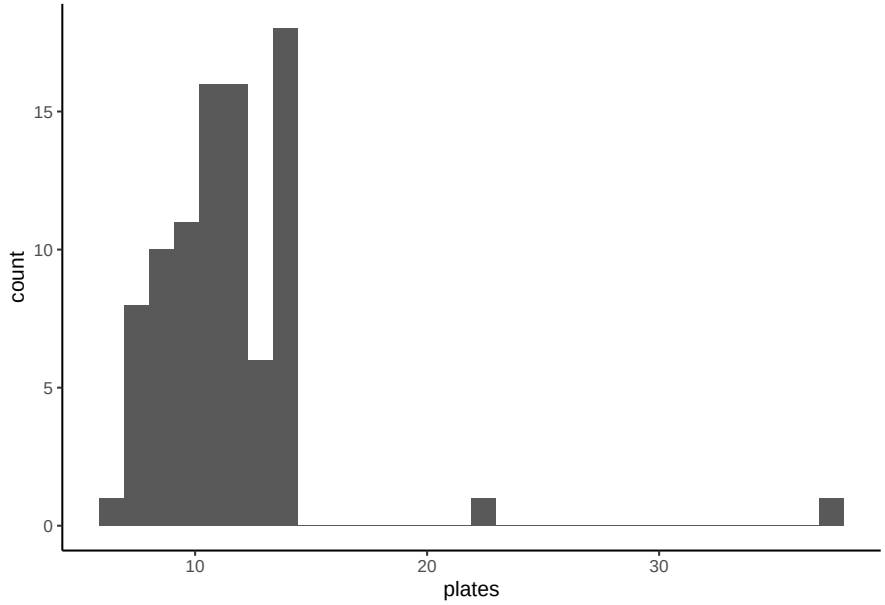
Answer:

```
hist(stickleback$plates[stickleback$genotype=="mm"],col="darkred",xlab="plates",main="Genotype: mm")
```




```
ggplot(data=filter(stickleback,genotype == "mm")) +
  geom_histogram(mapping = aes(x = plates))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



A bit of workflow

In this part you will see how to explore a data set and store your analysis in a reproducible and reusable way.

What is an R script

So far we only used the console of R. We enter commands and R executes them. Most of the time we want to have a whole bunch of commands stored in a single space: loading the file, preparing and cleaning the data,make figures, etc. The solution is easy, we jsut store our commands in a text file with the extension *.R* and tell R do execute this script. Evene better, RStudio has its own window panel for writing and executing scripts! Try to open a new R script (via File -> New -> R script) and just write your code in the newly opened window. When you have written your code, you can eitehr execute it step by step by putting the mouse cursor in the line you want to execute and then hit then *Run* button. If you want to eecute the whole script from beginning to end, just hit the *Source* button. This allows you to make reproducible scripts than you can modify and reuse whenever you need to. This is probably the biggest advantage of R. Once you have solved a problem and wrote a script for it, you can always reuse it. The more you use R, the less work will be necessary for each new project!

In the next part I will introduce the basic tools for creating an automated analysis that consists of several steps.

Defining and using Variables

Defining variables is really easy in R. Unlike in other programming languages we do not need to precisly define what kind of variable we want. We can just werite down a name and assign a value, R figures out the rest. For instance:

```
x = 5
```

automatically creates a *numeric* variable and assigns the value 5 to it. This command

```
vec = c(1,2,3,4,5,6)
```

creates a vectors that contains the values 1 to 6. The *c* here stands for *concatenate* and tells us to combine all the numbers into a single vector. You can access the elements of the vector by their index, e.g., we can set the 5th element of the vector to 0 in the follwoing way:

```
vec[5] = 0
```

In general it is advised to define the type of a varible before using it, and also tell R how big this variable will be (that is, how much memory we need to reserve to store it). For instance, if we want to define a 3x4 matrix we can use the command *matrix*:

```
mat=matrix(ncol=3,nrow=4)
```

You can then enter numbers in the rows and columns using the [] operator:

```
mat
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
## [3,]   NA   NA   NA
## [4,]   NA   NA   NA
```

```
mat[2,3]= 5
mat
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA    5
## [3,]   NA   NA   NA
## [4,]   NA   NA   NA
```

You can see that the matrix is intially filled with the object *NA* which stands for *not available* and ist the term R uses for missing data. Just like in dataframes (after all a data frame is more or less a matrix with benefits), you can also name columns and rows and use them to access the elements:

```
mat = matrix(1:25,nrow=5,ncol=5)
colnames(mat) = c("var1","var2","var3","var4","var5")
rownames(mat) = c("obs1","obs2","obs3","obs4","obs5")
mat
```

```
##      var1 var2 var3 var4 var5
## obs1    1    6   11   16   21
## obs2    2    7   12   17   22
## obs3    3    8   13   18   23
## obs4    4    9   14   19   24
## obs5    5   10   15   20   25
```

```
mat["obs2",]
```

```
## var1 var2 var3 var4 var5
##    2    7   12   17   22
```

```
mat[,"var3"]
```

```
## obs1 obs2 obs3 obs4 obs5
##   11   12   13   14   15
```

```
mat["obs2","var3"]
```

```
## [1] 12
```

If we want to store some text, we can write

```
word = "hello"
```

and R will automatically recognize that the string of letters "hello" is not numeric and will creat a variable of type *character*. Try to see how this is different from

```
letters = c("h","e","l","l","o")
```

Next try to see what happens here:

```
hello = 10
word1 = "hello"
word2 = hello
print(word1)
```

```
## [1] "hello"
```

```
print(word2)
```

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

Loops

Before we have seen how to calcualte the mean number of plates for each genotype. Essentially we have done the same thing 3 times. Imagine we woul dhave hundreeds of genotypes. In such situations loops come in handy. They allow us to automate a process that needs to be repeated many times. We only introduce the so-called *for-loop* here, which will allow you to do a lot of things. Once you have understood how it works, it will be easy to figure out how a *while-loop* works and what the differences are.

Here is a simple for loop that calcualtes the mean number of plates for each genotpye in our stickleback data set:

```
# first we create a vector that contains all genotypes in our dataset
# we can do this manually:
# genotypes = c("mm","mM","MM")
# or by using the function levels()
# that gives us a vector with all the different entries found in a
# vector of type factor

genotypes = levels(stickleback$genotype)

# next, we define a vector that has the same length as the number
# of different genotypes and name the elements accordingly

plate.numbers = vector("numeric",length(genotypes))
names(plate.numbers)=genotypes

# no comes the actual loop:
for(g in genotypes)
{
  plate.numbers[g]=mean(stickleback$plates[stickleback$genotype==g])
}
```

We can see what is happening here by showing the intermediate steps more explicitly:

```
i = 1
for(g in genotypes)
{
  print(paste("This is iteration ",i," of our for-loop"))
  print(paste("Genotype ",i," = ",g))
  mean.temp = mean(stickleback$plates[stickleback$genotype==g])
  print(paste("The mean number of plates of genotype ",g," is ",mean.temp))
  plate.numbers[g]=mean.temp
  i = i+1
}
```

```
## [1] "This is iteration  1  of our for-loop"
## [1] "Genotype  1  =  mm"
## [1] "The mean number of plates of genotype  mm  is  11.6704545454545"
## [1] "This is iteration  2  of our for-loop"
## [1] "Genotype  2  =  Mm"
## [1] "The mean number of plates of genotype  Mm  is  50.3793103448276"
## [1] "This is iteration  3  of our for-loop"
## [1] "Genotype  3  =  MM"
## [1] "The mean number of plates of genotype  MM  is  62.780487804878"
```

There are 3 basics steps here:

- Reserve memory for the output
- Set up how often the loop should be iterated
- The code that should be repeatedly calcualted

Question: Could you come up with an alterantive way to solve this problem using a loop?

If / else

Another useful tool are if/else constructs. They allow you to make choices during your script. Let us again consider the sticklbeack data set. Let say we want to calcualte the difference between the number of plates of plates of an individual and the mean of that indivudal's genotype. We can do this by combining loops with if/else statements (admittedly, there are better ways to achieve this but it is a good instuctive example):

```
number.obs = dim(stickleback)[1]
diff.to.mean = vector("numeric",length=number.obs)
for (i in 1:number.obs)
{
  if(stickleback$genotype[i] == "mm")
    diff.to.mean[i] = stickleback$plate[i] - mean.mm

  if(stickleback$genotype[i] == "mM")
    diff.to.mean = stickleback$plate[i] - mean.mM

  if(stickleback$genotype[i] == "MM")
    diff.to.mean = stickleback$plate[i] - mean.MM
}


stickleback3= mutate(stickleback2,diff.per.genotype = diff.to.mean)

head(stickleback3)
```

```
##     id plates genotype difference diff.per.genotype
## 1  4-1     11       mm  -32.43314        -0.7804878
## 2  4-2     63       Mm   19.56686                NA
## 3  4-4     22       Mm  -21.43314                NA
## 4  4-5     10       Mm  -33.43314                NA
## 5 4-10     14       mm  -29.43314                NA
## 6 4-12     11       mm  -32.43314                NA
```

Working with Distributions and random variables

Functions to work with probaility distributions

R has a set of functions to work with distirbutions and random numbres. Have a look at

```
help(Distributions)
```

if you want to know more. Let us focus on the normal distirbution. It's density is given by

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The function *dnorm* returns the value of the probability density function for the normal distribution for x, given parameters μ and σ. Some examples of using dnorm are below:

```
# We use the pdf of the normal with x = 0,
# mu = 0 and sigma = 0.
# The dnorm function takes three main arguments,
# as do all of the *norm functions in R.
```

```r
dnorm(0, mean = 0, sd = 1)
```

```
## [1] 0.3989423
```

```r
# Another exmaple of dnorm where parameters have been changed.

dnorm(2, mean = 5, sd = 3)
```

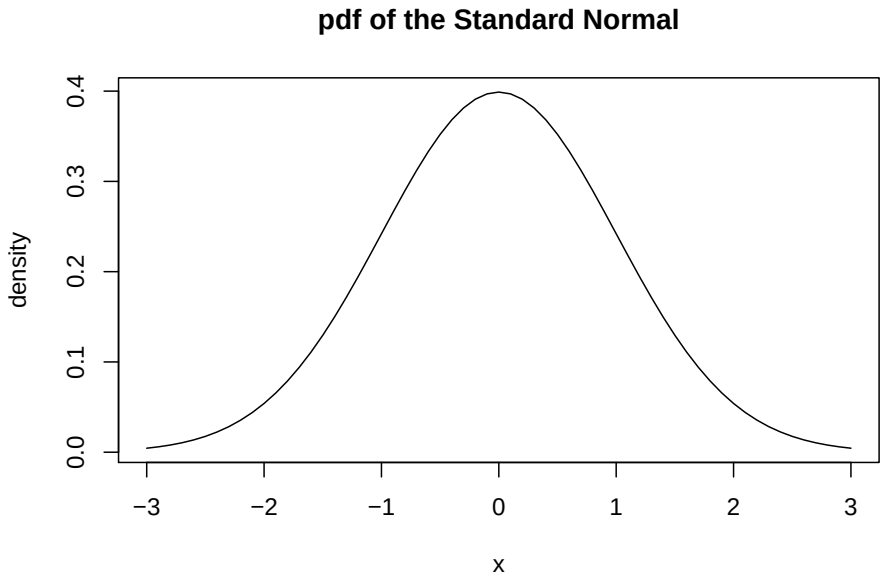
```
## [1] 0.08065691
```

Let us do a plot showing the density fucntion of the normal distirbution. We use base R as ggplot works fine with data frames and here we prefer to wrok with vectors instead.

```r
# The plot should show x values on the x-axis and the dnorm(x,...)
# values on the y-axis

# First I'll make a vector of x values
x_vals <- seq(-3, 3, by = .1)

# Let's have a look
x_vals
```

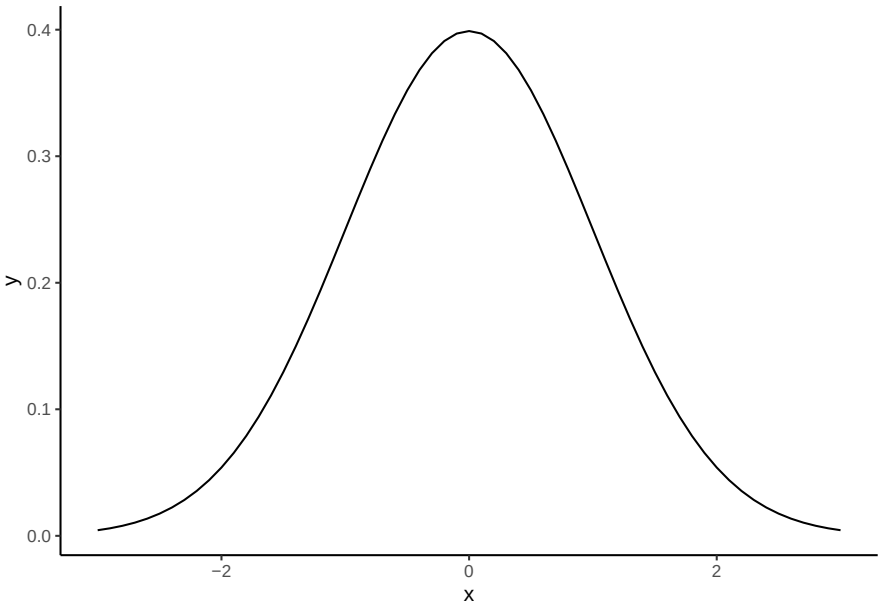
```
##  [1] -3.0 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2.0 -1.9 -1.8 -1.7 -1.6
## [16] -1.5 -1.4 -1.3 -1.2 -1.1 -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1
## [31]  0.0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.0  1.1  1.2  1.3  1.4
## [46]  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9
## [61]  3.0
```

```r
# The y values are then given by dnorm
y_vals <- dnorm(x_vals,0,1)

# Now we'll plot these values
plot(x_vals,y_vals,
     type = "l", # Make it a line plot
     main = "pdf of the Standard Normal",
     xlab= "x",ylab="density")
```



Of course, you could create a data frame with these values and then use ggplot if you want to:

```
std_norm = data.frame(x= x_vals,y=y_vals)

ggplot(data = std_norm) +
  geom_line(mapping = aes(x = x,y=y))
```



The other three most relevant functions are: *pnorm* (cumulative distribution function), *qnorm* (quantiles) and *rnorm* (draw random numbers). The ue of *pnorm* is completely analogous to *dnorm*, so we skip it here and focus on themore interesting *qnorm* and *rnorm*.

Quantiles

The function qnorm gives us the quantiles of the normal distirbution:

```
# Let's make a vector of :
# from 0 to 1 by increments of .05
quantiles <- seq(0, 1, by = .05)
quantiles
```

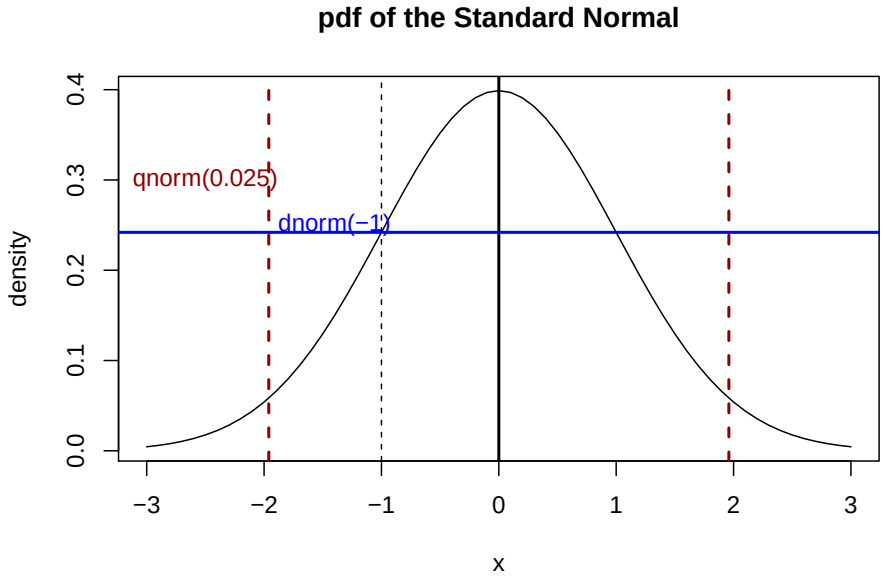
```
##  [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70
## [16] 0.75 0.80 0.85 0.90 0.95 1.00
```

```
# Now we'll find the value for each of those quantiles
# Remeber that the q-quantile is the value Q such that
# P(X>Q) = q
# In other words, the quantile is the inverse of the CDF

qvalues <- qnorm(quantiles)
qvalues
```

```
##  [1]       -Inf -1.6448536 -1.2815516 -1.0364334 -0.8416212 -0.6744898
##  [7] -0.5244005 -0.3853205 -0.2533471 -0.1256613  0.0000000  0.1256613
## [13]  0.2533471  0.3853205  0.5244005  0.6744898  0.8416212  1.0364334
## [19]  1.2815516  1.6448536        Inf
```

We could now plot the PDF of a normal distribution and illustrate a few of the concepts:

```r
plot(x_vals,y_vals,
     type = "l", # Make it a line plot
     main = "pdf of the Standard Normal",
     xlab= "x",ylab="density")

# the 50 % quantile is just the median:
abline(v = qnorm(0.5),col="black",lwd=2,lty=1)

# the 2.5 %  and the 97.5% quantile indicated the area where 95% of the most
# common data lies:
abline(v = qnorm(c(0.025,0.975)),col="darkred",lwd=2,lty=2)
text(-2.5,0.3,"qnorm(0.025)",col="darkred")

# In contrast, the function dnorm gives us the value of the PDF
# for a specfific x value, that is, the height of the function f(x)
# at the point x

abline(h = dnorm(-1),col="blue",lwd=2,lty=1)
abline(v = -1,lty=2)
text(-1.4,0.25,"dnorm(-1)",col="blue")
```



Drawing random numbers

R also allows us to draw random numbers. This is very handy in many situations. The parameters of rnorm are

- the number of observations
- the mean of the distirbution
- the standard deviation of the distribution

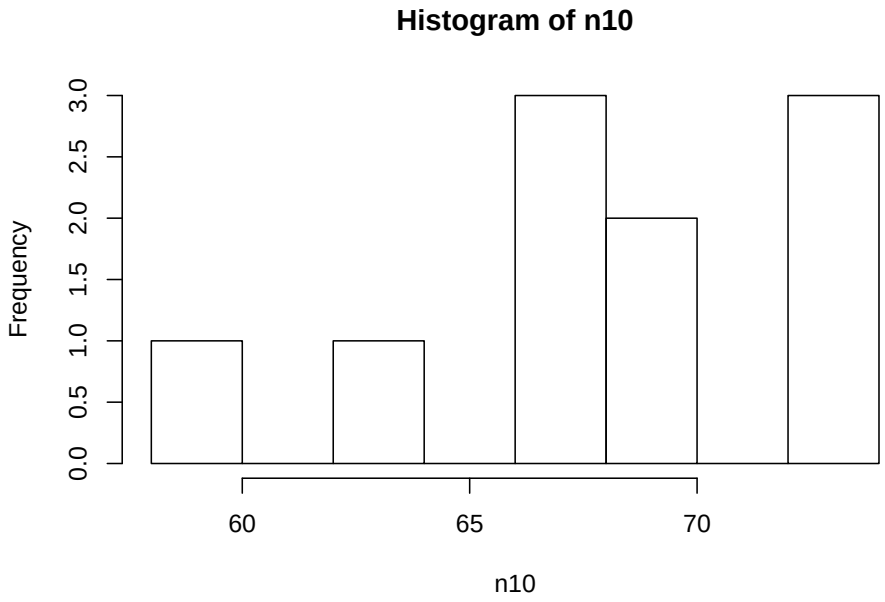
Here is an example, where we calculate a sample of 0, 100 and 1000 observations of the same distirbution.

```r
# Let's generate three different vectors of random numbers
# from a normal  distribution
```

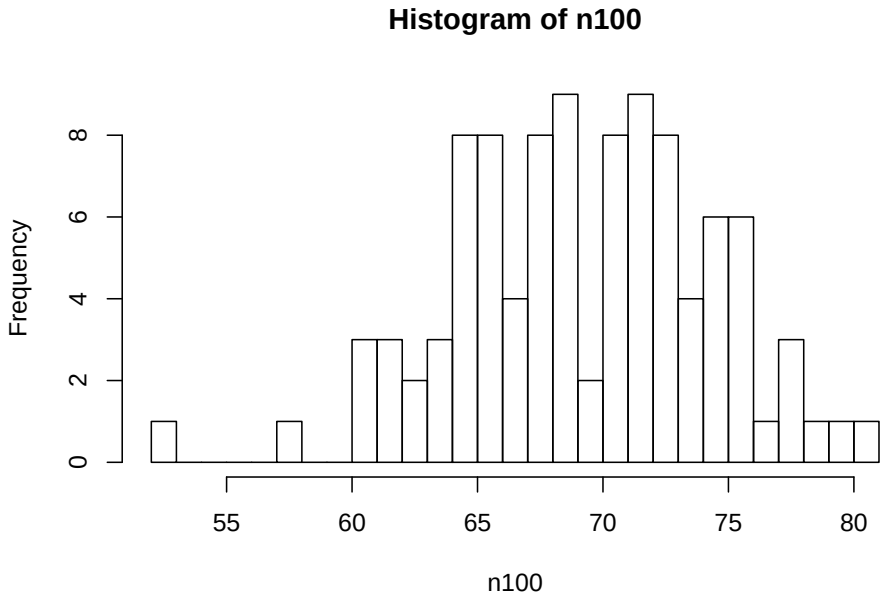
```r
n10 <- rnorm(10, mean = 70, sd = 5)
n100 <- rnorm(100, mean = 70, sd = 5)
n10000 <-  rnorm(10000, mean = 70, sd = 5)

# Let's just look at one of the vectors
n10
```

```
##  [1] 67.27264 73.27253 59.89174 73.59033 69.84460 66.47077 69.40316 63.89786
##  [9] 73.37192 66.36807
```

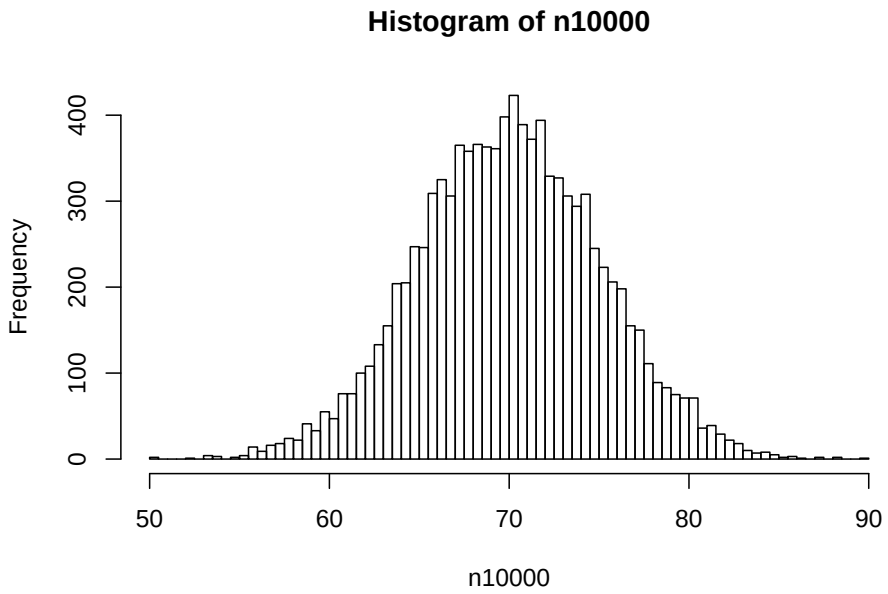
```r
# The breaks argument specifies how many bars are in the histogram
hist(n10, breaks = 5)
```




```r
hist(n100, breaks = 20)
```




```r
hist(n10000, breaks = 100)
```



Note: If we set a *seed* for our random number generator (the algorithm that is used to calcualte a radnom number), we can make our analysis replicable:

```
set.seed(100)
rnorm(1,0,1)
```

```
## [1] -0.5021924
```

```
rnorm(1,0,1)
```

```
## [1] 0.1315312
```

```
set.seed(100)
rnorm(1,0,1)
```

```
## [1] -0.5021924
```

```
rnorm(1,0,1)
```

```
## [1] 0.1315312
```

Plotting Confidence intervalls

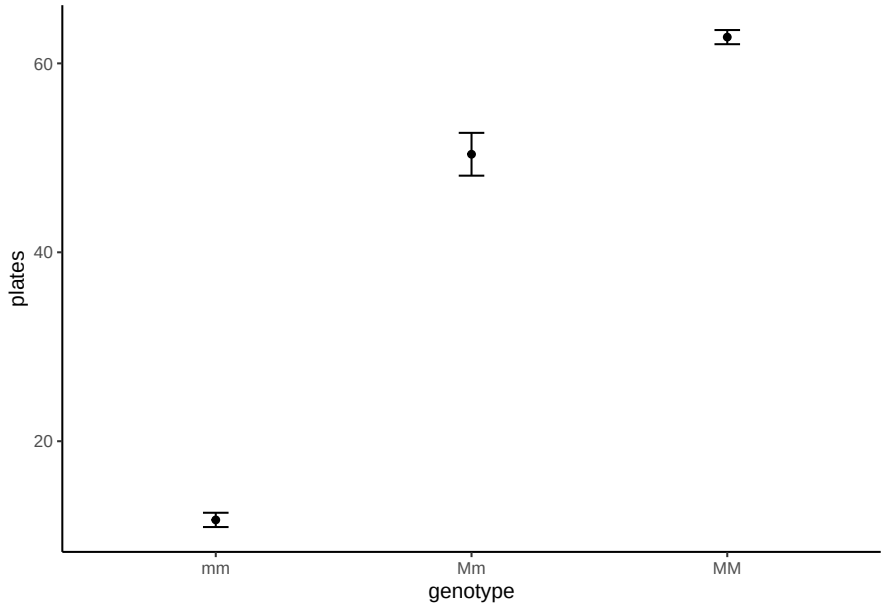
We first generate a data frame with a summary of our data (this can be done in many ways, this is a concise one):

```
CIs <- summarySE(stickleback, measurevar="plates", groupvars=c("genotype"))

CIs
```

```
##   genotype   N   plates        sd        se        ci
## 1       mm  88 11.67045  3.567805 0.3803293 0.7559456
## 2       Mm 174 50.37931 15.146866 1.1482809 2.2664440
## 3       MM  82 62.78049  3.410313 0.3766060 0.7493279
```

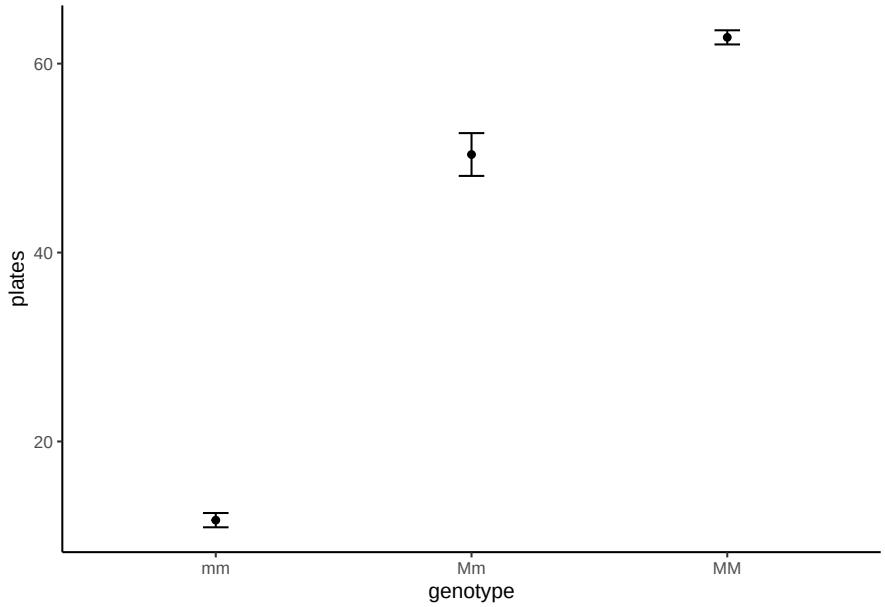
and then we use the data frame with our summary to add error bars to our plot:

```
ggplot(data = CIs) +
  geom_point(mapping = aes(x = genotype,y=plates)) +
  geom_errorbar(mapping = aes(x = genotype,y=plates,ymin=plates-ci, ymax=plates+ci), width=.1)
```



Note: One can shorten things by specifying the aesthetics in the ggplot command - this simply means that all geom_functions use the same x and y variables (which is very often the case, but not always!):

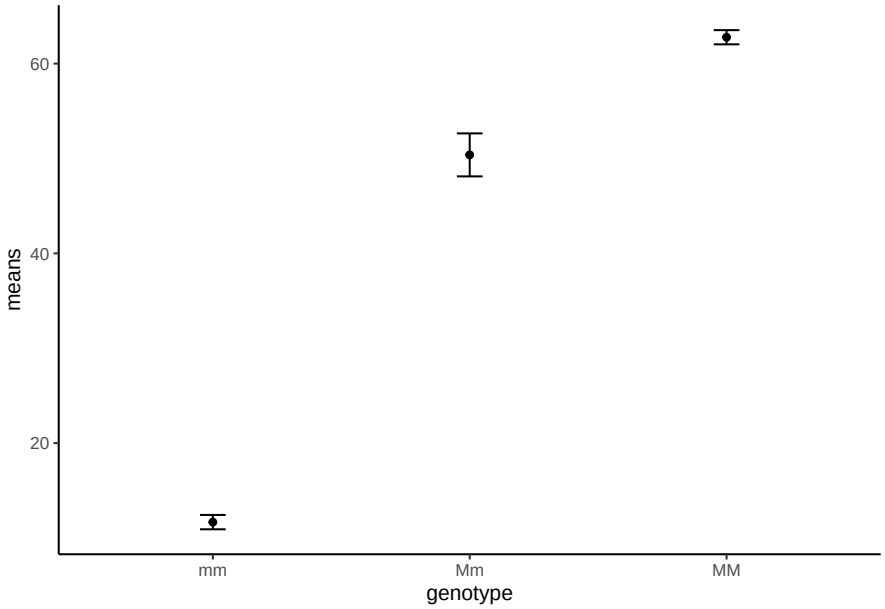
```
ggplot(data = CIs, aes(x = genotype,y=plates)) +
  geom_point() +
  geom_errorbar(mapping = aes(ymin=plates-ci, ymax=plates+ci), width=.1)
```



Note: The function *t.test()* can also be used to calcualte confidence intervalls assuming a normal distribution.

```
genotypes = c("MM","Mm","mm")
CI.upper = vector("numeric",3)
CI.lower = vector("numeric",3)
```

```
means = vector("numeric",3)

i = 1

for(g in genotypes)
{

  results = t.test(stickleback$plates[stickleback$genotype == g])

  CI.lower[i] = results$conf.int[1]
  CI.upper[i]= results$conf.int[2]
  means[i] = results$estimate
  i = i + 1
}

CI.data = data.frame(genotype = genotypes,mean = means,CI.lower = CI.lower,CI.upper = CI.upper)

ggplot(data = CI.data, aes(x = genotype,y=means)) +
  geom_point() +
  geom_errorbar(mapping = aes(ymin=CI.lower, ymax=CI.upper), width=.1)
```



Putting everything together: A simple simulation

Copy and paste this R code into a script. Before you run it, try to figure out what it does.

```
reps = 1000
sample_size = 10
mu = 10
sigma = 5
means = vector("numeric",reps)

for (i in 1:reps)
{
```

```
  sample = rnorm(sample_size,mu,sigma)
  means[i] = mean(sample)
}

hist(means,freq=F,main="A distribution")

x = seq(mu-2*sigma,mu+2*sigma,by=0.01)

SE = sigma/sqrt(sample_size)

y = dnorm(x,mu,SE)

lines(x,y)
```

Hypothesis Tests

In this chpater we will learn how to do hypotehsis tests in R. This is very simple, as you will see - often a single line of code is sufficient for conducting a test.

Binomial-test

We use the binomial test to test whether spermatogenesis genes in the mouse genome occur with unusual frequency on the X chromosome. First, we load the dat set and inspect it:

```
mouseGenes <- read_csv("~/Dropbox/Teaching/BlogdownPages/StatisticsForBiology/chap07e2SexAndX.csv")

head(mouseGenes)
```

```
## # A tibble: 6 x 2
##   chromosome onX
##   <chr>      <chr>
## 1 4          no
## 2 4          no
## 3 6          no
## 4 6          no
## 5 6          no
## 6 7          no
```

```
# Tabulate the number of spermatogenesis genes on the
# X-chromosome and the number not on the X-chromosome.

table(mouseGenes$onX)
```

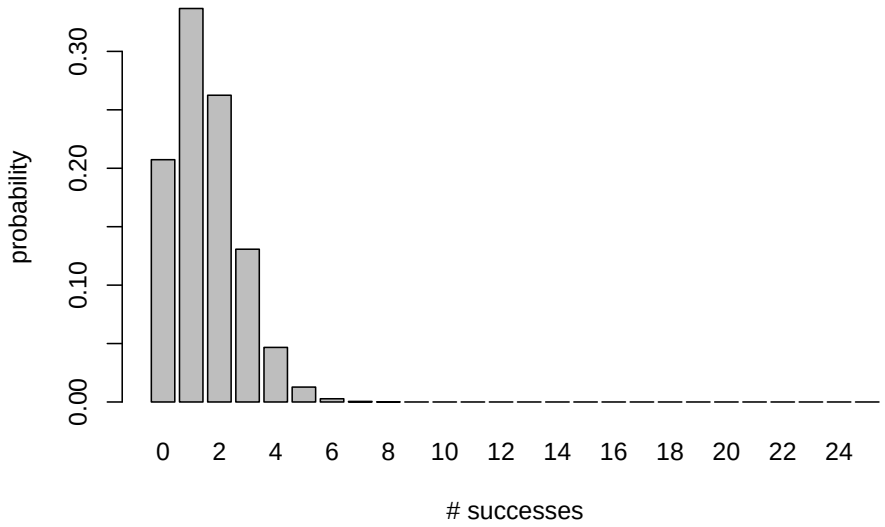
```
##
##  no yes
##  15  10
```

```
# Calculate the binomial probabilities of all possible outcomes
# under the null hypothesis. Under the binomial
# distribution with n = 25 and p = 0.061, the number of successes
# can be any integer between 0 and 25.

xsuccesses <- 0:25
```

```r
probx <- dbinom(xsuccesses, size = 25, prob = 0.061)
data.frame(xsuccesses, probx)
```

```
##    xsuccesses        probx
## 1           0 2.073193e-01
## 2           1 3.367007e-01
## 3           2 2.624760e-01
## 4           3 1.307255e-01
## 5           4 4.670757e-02
## 6           5 1.274386e-02
## 7           6 2.759585e-03
## 8           7 4.865905e-04
## 9           8 7.112305e-05
## 10          9 8.727323e-06
## 11         10 9.071211e-07
## 12         11 8.035781e-08
## 13         12 6.090306e-09
## 14         13 3.956429e-10
## 15         14 2.203032e-11
## 16         15 1.049510e-12
## 17         16 4.261188e-14
## 18         17 1.465509e-15
## 19         18 4.231266e-17
## 20         19 1.012696e-18
## 21         20 1.973624e-20
## 22         21 3.052667e-22
## 23         22 3.605629e-24
## 24         23 3.055193e-26
## 25         24 1.653947e-28
## 26         25 4.297797e-31
```

```r
barplot(probx,names.arg=xsuccesses,xlab="# successes",ylab="probability")
```




```r
# Use these probabilities to calculate the P-value corresponding to
# an observed 10 spermatogenesis genes on the X chromosome.
# Remember to multiply the probability of 10 or more successes
# by 2 for the two-tailed test result.
```

```r
2 * sum(probx[xsuccesses >= 10])
```

```
## [1] 1.987976e-06
```

```r
# For a faster result, try R's built-in binomial test.
# The resulting P-value is slightly different from our calculation. # The output of binom.test includes
# proportion using the Clopper-Pearson method, which is more
# conservative than the Agresti-Coull method.

binom.test(10, n = 25, p = 0.061)
```

```
##
##  Exact binomial test
##
## data:  10 and 25
## number of successes = 10, number of trials = 25, p-value = 9.94e-07
## alternative hypothesis: true probability of success is not equal to 0.061
## 95 percent confidence interval:
##  0.2112548 0.6133465
## sample estimates:
## probability of success
##                    0.4
```

```r
# Agresti-Coull 95% confidence interval for the proportion
# using the binom package.

library(binom)
binom.confint(30, n = 87, method = "ac")
```

```
##          method  x  n      mean     lower     upper
## 1 agresti-coull 30 87 0.3448276 0.2532164 0.4495625
```

χ^2 test

We perform a goodness-of-fit test. We use a Poisson probability model and fit it to frequency data on the number of marine invertebrate extinctions per time block in the fossil record. We first read and inspect the data. Each row is a time block, with the observed number of extinctions listed.

The first step is to create a frequency table for the number of time blocks in each number-of-extinctions category. The command table does what's needed, but note that some extinction categories are not represented (e.g., 0, 12 and 13 extinctions).

```r
extinctTable <- table(extinctData$numberOfExtinctions)
data.frame(Frequency = addmargins(extinctTable))
```

```
##    Frequency.Var1 Frequency.Freq
## 1               1             13
## 2               2             15
## 3               3             16
## 4               4              7
## 5               5             10
## 6               6              4
## 7               7              2
## 8               8              1
## 9               9              2
```

```
## 10              10               1
## 11              11               1
## 12              14               1
## 13              16               2
## 14              20               1
## 15             Sum              76
```

To remedy the problem of missing categories, we transform the original variable into a factor (that is, a categorical variable) with all counts between 0 and 20 as levels (note that 20 is not the maximum possible number of extinctions, but it is a convenient cutoff for this table).

```
extinctData = mutate(extinctData,nExtinctFactor = factor(numberOfExtinctions, levels = c(0:20)))

extinctTable2 <- table(extinctData$nExtinctFactor)
```

Estimate the mean number of extinctions per time block from the data. The estimate is needed for the goodness-of-fit test.

```
meanExtinctions <- mean(extinctData$numberOfExtinctions)
meanExtinctions
```

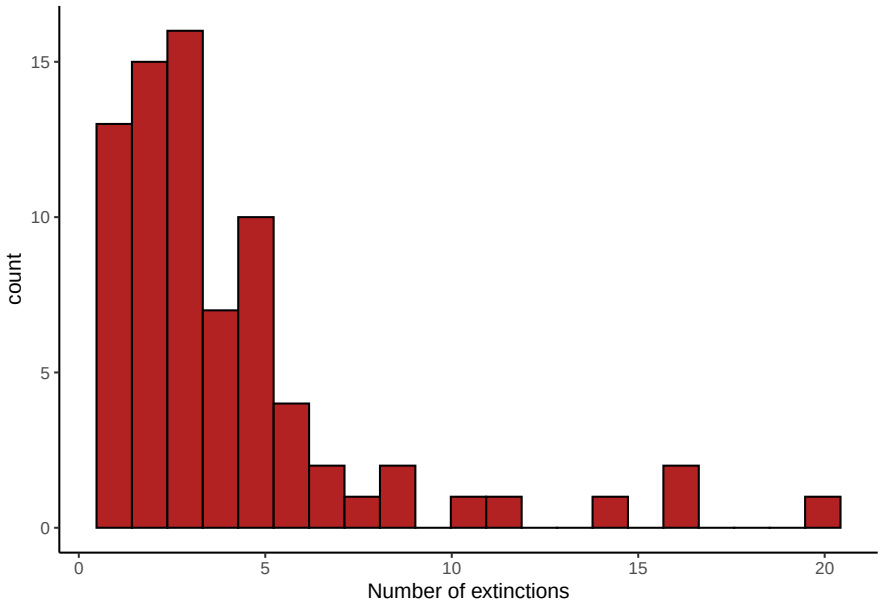
```
## [1] 4.210526
```

Calculate expected frequencies under the Poisson distribution using the estimated mean. (For now, continue to use 20 extinctions as the cutoff, but don't forget that the Poisson distribution includes the number-of-extinctions categories 21, 22, 23, and so on.)

```
expectedProportion <- dpois(0:20, lambda = meanExtinctions)
expectedFrequency <- expectedProportion * 76
```

Show the frequency distribution in a histogram.

```
ggplot(extinctData) +
  geom_histogram(mapping= aes(x =numberOfExtinctions),fill = "firebrick",color="black",bins=21)+
  labs (x="Number of extinctions")
```



Make a table of observed and expected frequencies, saving as a data frame.

```
extinctFreq <- data.frame(nExtinct = 0:20, obsFreq = as.vector(extinctTable2),
    expFreq = expectedFrequency)
extinctFreq
```

```
##    nExtinct obsFreq      expFreq
## 1         0       0 1.127730e+00
## 2         1      13 4.748338e+00
## 3         2      15 9.996501e+00
## 4         3      16 1.403018e+01
## 5         4       7 1.476861e+01
## 6         5      10 1.243672e+01
## 7         6       4 8.727524e+00
## 8         7       2 5.249639e+00
## 9         8       1 2.762968e+00
## 10        9       2 1.292616e+00
## 11       10       1 5.442596e-01
## 12       11       1 2.083290e-01
## 13       12       0 7.309790e-02
## 14       13       0 2.367543e-02
## 15       14       1 7.120431e-03
## 16       15       0 1.998718e-03
## 17       16       2 5.259783e-04
## 18       17       0 1.302733e-04
## 19       18       0 3.047328e-05
## 20       19       0 6.753081e-06
## 21       20       1 1.421701e-06
```

The low expected frequencies will violate the assumptions of the χ^2 test, so we will need to group categories. Create a new variable that groups the extinctions into fewer categories (again, there are many ways to do this, here I use the function *cut*).

```
extinctFreq$groups <- cut(extinctFreq$nExtinct,
    breaks = c(0, 2:8, 21), right = FALSE,
    labels = c("0 or 1","2","3","4","5","6","7","8 or more"))
extinctFreq
```

```
##    nExtinct obsFreq      expFreq    groups
## 1         0       0 1.127730e+00    0 or 1
## 2         1      13 4.748338e+00    0 or 1
## 3         2      15 9.996501e+00         2
## 4         3      16 1.403018e+01         3
## 5         4       7 1.476861e+01         4
## 6         5      10 1.243672e+01         5
## 7         6       4 8.727524e+00         6
## 8         7       2 5.249639e+00         7
## 9         8       1 2.762968e+00 8 or more
## 10        9       2 1.292616e+00 8 or more
## 11       10       1 5.442596e-01 8 or more
## 12       11       1 2.083290e-01 8 or more
## 13       12       0 7.309790e-02 8 or more
## 14       13       0 2.367543e-02 8 or more
## 15       14       1 7.120431e-03 8 or more
## 16       15       0 1.998718e-03 8 or more
## 17       16       2 5.259783e-04 8 or more
## 18       17       0 1.302733e-04 8 or more
```

```
## 19       18       0 3.047328e-05 8 or more
## 20       19       0 6.753081e-06 8 or more
## 21       20       1 1.421701e-06 8 or more
```

Then sum up the observed and expected frequencies within the new categories.

```
obsFreqGroup <- tapply(extinctFreq$obsFreq, extinctFreq$groups, sum)
expFreqGroup <- tapply(extinctFreq$expFreq, extinctFreq$groups, sum)
ObsVsExp = data.frame(obs = obsFreqGroup, exp = expFreqGroup,group= c("0 or 1","2","3","4","5","6","7",
ObsVsExp
```

```
##           obs       exp     group
## 0 or 1     13  5.876068    0 or 1
## 2          15  9.996501         2
## 3          16 14.030177         3
## 4           7 14.768608         4
## 5          10 12.436722         5
## 6           4  8.727524         6
## 7           2  5.249639         7
## 8 or more   9  4.914760 8 or more
```

The *expected* frequency for the last category, "8 or more", doesn't yet include the expected frequencies for the categories 21, 22, 23, and so on (remeber that we used a cut-off of 20 before!). However, the expected frequencies must sum to 76. In the following, we recalculate the expected frequency for the last group, expFreqGroup[length(expFreqGroup)], as 76 minus the sum of the expected frequencies for all the other groups.

```
expFreqGroup[length(expFreqGroup)] = 76 - sum(expFreqGroup[1:(length(expFreqGroup)-1)])
data.frame(obs = obsFreqGroup, exp = expFreqGroup)
```

```
##           obs       exp
## 0 or 1     13  5.876068
## 2          15  9.996501
## 3          16 14.030177
## 4           7 14.768608
## 5          10 12.436722
## 6           4  8.727524
## 7           2  5.249639
## 8 or more   9  4.914761
```

Finally, we are ready to carry out the χ^2 goodness-of-fit test. R gives us a warning here because one of the expected frequencies is less than 5. However, we have been careful to meet the assumptions of the χ^2 test, so let's persevere. Once again, R doesn't know that we have estimated a parameter from the data (the mean), so it won't use the correct degrees of freedom when calculating the P-value. As before, we need to grab the χ^2 value calculated by chisq.test and recalculate P using the correct degrees of freedom. Since the number of categories is now 8, the correct degrees of freedom is 8 - 1 - 1 = 6.

```
saveChiTest <- chisq.test(obsFreqGroup, p = expFreqGroup/76)
```

```
## Warning in chisq.test(obsFreqGroup, p = expFreqGroup/76): Chi-squared
## approximation may be incorrect
```

```
saveChiTest
```

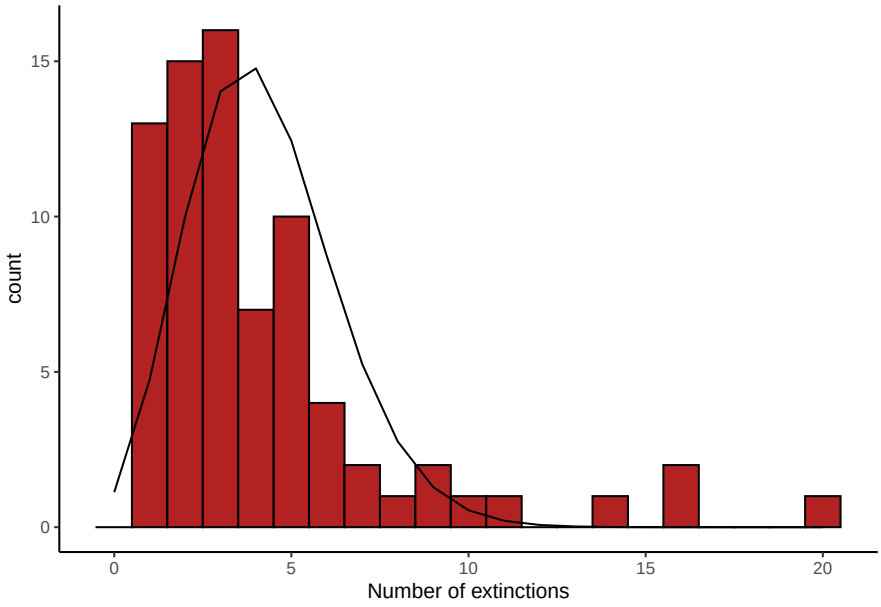
```
##
##  Chi-squared test for given probabilities
##
## data:  obsFreqGroup
## X-squared = 23.95, df = 7, p-value = 0.001163
```

```
# Wrong degrees of freedom, so wrong P-value!

pValue <- 1 - pchisq(saveChiTest$statistic, df = 6)

# correct P-value!
pValue
```

```
##    X-squared
## 0.0005334919
```

Finally, we can compare the expected and observed data in a histogramm. One can clearly see that the fit is not very good.

```
df = data.frame(x = 0:20,y = 76*dpois(0:20,meanExtinctions))

ggplot(extinctData) +
  geom_histogram(mapping= aes(x =numberOfExtinctions),fill = "firebrick",color="black",bins=21)+
  labs (x="Number of extinctions") +
  geom_line(data = df, mapping = aes(x = x,y=y))
```



t-test

One-sample t-test

We use a one-sample t-test to compare body temperature in a random sample of people with the "expected" temperature 98.6 F.

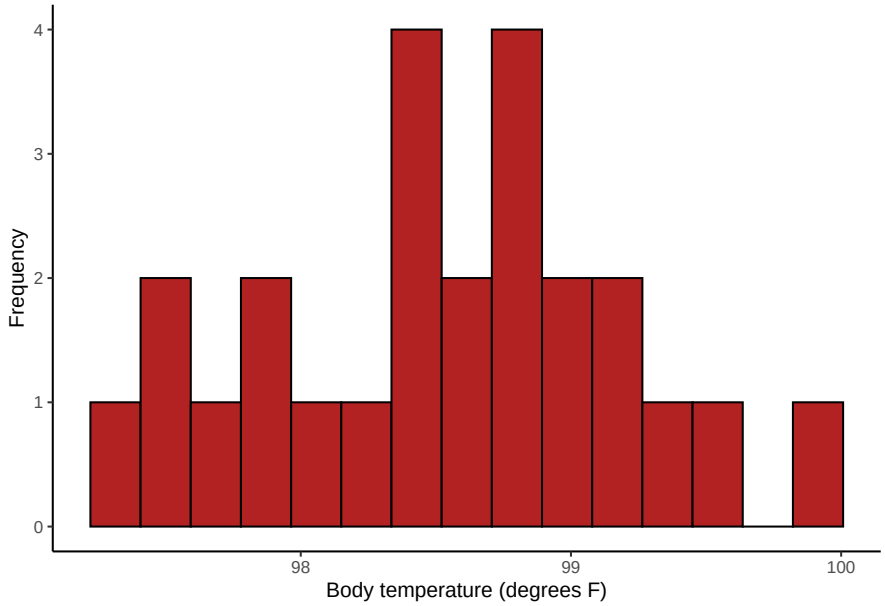
Read and inspect the data:

```
heat <- read.csv(url("http://www.zoology.ubc.ca/~schluter/WhitlockSchluter/wp-content/data/chapter11/cha
head(heat)
```

```
##   individual temperature
## 1          1        98.4
## 2          2        98.6
```

```
## 3          3        97.8
## 4          4        98.8
## 5          5        97.9
## 6          6        99.0
```

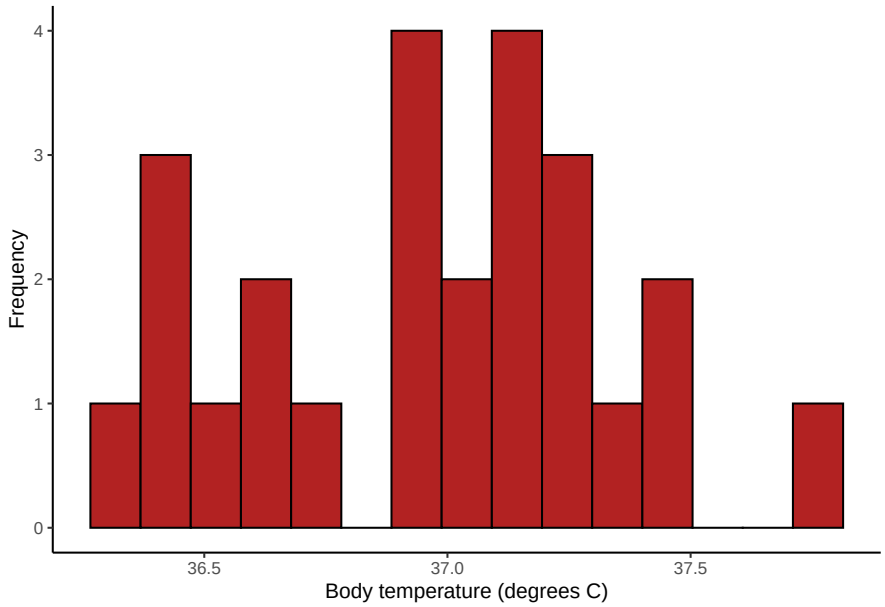
```
ggplot(data = heat) +
  geom_histogram(mapping = aes(x = temperature),fill = "firebrick",color="black",bins=15) +
  labs(x = "Body temperature (degrees F)",y = "Frequency", main = "")
```



A one-sample t-test can be calculate using the function t.test. The mu argument gives the value stated in the null hypothesis. Let us first transform the data into celsius using the formula to turn:

$$(X??F - 32)??5/9 = Y??C$$

```
mu0 = (98.6-32)*5/9

heat = mutate(heat,temperatureC = (temperature-32) * 5/9)

ggplot(data = heat) +
  geom_histogram(mapping = aes(x = temperatureC),fill = "firebrick",color="black",bins=15) +
  labs(x = "Body temperature (degrees C)",y = "Frequency", main = "")
```




```
t.test(heat$temperatureC, mu = mu0)
```

```
##
##  One Sample t-test
##
## data:  heat$temperatureC
## t = -0.56065, df = 24, p-value = 0.5802
## alternative hypothesis: true mean is not equal to 37
## 95 percent confidence interval:
##  36.80235 37.11321
## sample estimates:
## mean of x
##  36.95778
```

Two-sample t-test

We compare horn length of live and dead (spiked) horned lizards.

```
lizard <- read.csv(url("http://www.zoology.ubc.ca/~schluter/WhitlockSchluter/wp-content/data/chapter12/
head(lizard)
```

```
##   squamosalHornLength Survival
## 1                25.2   living
## 2                26.9   living
## 3                26.6   living
## 4                25.6   living
## 5                25.7   living
## 6                25.9   living
```

Note that there is one missing value for the variable "squamosalHornLength". Everything is easier if we eliminate the row with missing data.

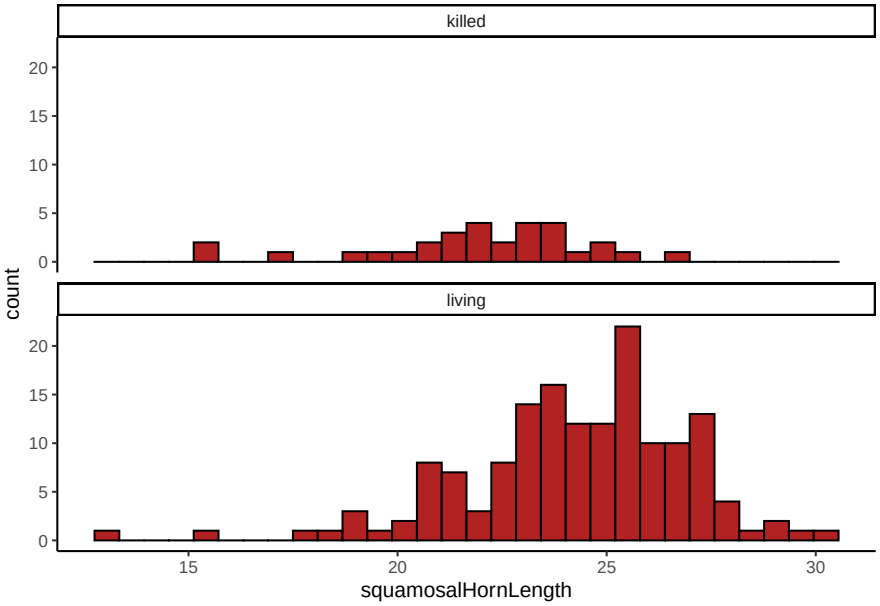
```
lizard2 <- na.omit(lizard)
head(lizard2)
```

```
##   squamosalHornLength Survival
```

```
## 1                  25.2   living
## 2                  26.9   living
## 3                  26.6   living
## 4                  25.6   living
## 5                  25.7   living
## 6                  25.9   living
```

```
ggplot(data = lizard2) +
  geom_histogram(mapping = aes(x = squamosalHornLength ),fill = "firebrick",color="black") +
  facet_wrap( ~Survival,nrow=2) +
  theme_classic()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



A two-sample t-test of the difference between two means can be carried out with t.test by using a formula, asking if squamosalHornLength is predicted by Survival, and specifying that the variables are in the data frame lizard.

```
t.test(squamosalHornLength ~ Survival, data = lizard)
```

```
##
##  Welch Two Sample t-test
##
## data:  squamosalHornLength by Survival
## t = -4.2634, df = 40.372, p-value = 0.0001178
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.381912 -1.207092
## sample estimates:
## mean in group killed mean in group living
##             21.98667              24.28117
```

The same reuslt could be achieved with

```
t.test(lizard$squamosalHornLength[lizard$Survival=="killed"], lizard$squamosalHornLength[lizard$Survival
```

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

```
##
## data:  lizard$squamosalHornLength[lizard$Survival == "killed"] and lizard$squamosalHornLength[lizard$
## t = -4.2634, df = 40.372, p-value = 0.0001178
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.381912 -1.207092
## sample estimates:
## mean of x mean of y
##  21.98667  24.28117
```

The output of t.test includes the 95% confidence interval for the difference between means. Add $confint after calling the function to get R to report only the confidence interval. The formula in the following command tells R to compare squamosalHornLength between the two groups indicated by Survival.

```
t.test(squamosalHornLength ~ Survival, data = lizard)$conf.int
```

```
## [1] -3.381912 -1.207092
## attr(,"conf.level")
## [1] 0.95
```

*Note: R has used the Welch two sample t-test here. If we want to force R to use the standrad t.test we set a parameter specifiying this:

```
t.test(squamosalHornLength ~ Survival, data = lizard,var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  squamosalHornLength by Survival
## t = -4.3494, df = 182, p-value = 2.27e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.335402 -1.253602
## sample estimates:
## mean in group killed mean in group living
##             21.98667             24.28117
```

Deviations from Normailty

QQ Plots and Transformations

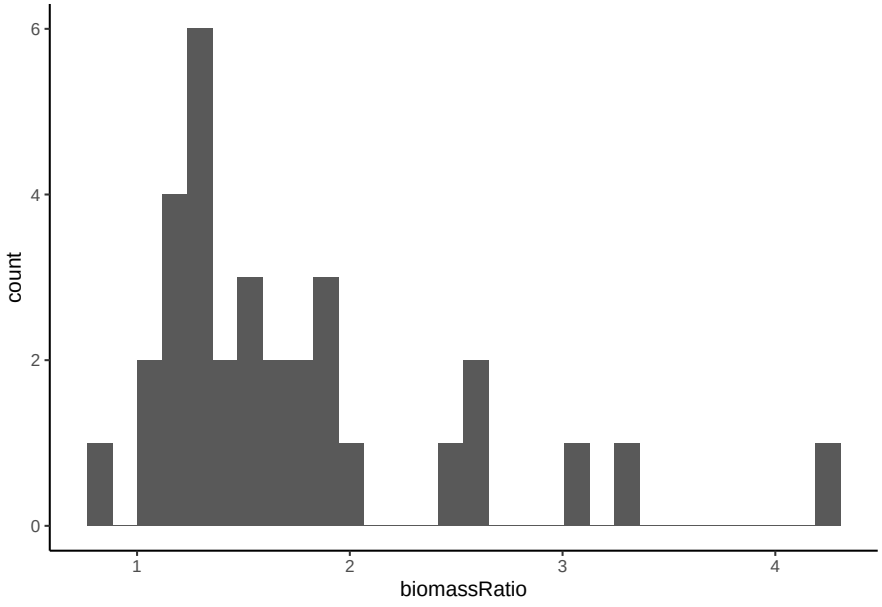
We investigate the ratio of biomass between marine reserves and non-reserve control areas.

```
marine <- read.csv(url("http://www.zoology.ubc.ca/~schluter/WhitlockSchluter/wp-content/data/chapter13/
head(marine)
```

```
##   biomassRatio
## 1         1.34
## 2         1.96
## 3         2.49
## 4         1.27
## 5         1.19
## 6         1.15
```

```
ggplot(data = marine) +
  geom_histogram(mapping = aes(x = biomassRatio))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Normal quantile plot of biomass ratio data.

```
library(car)
```

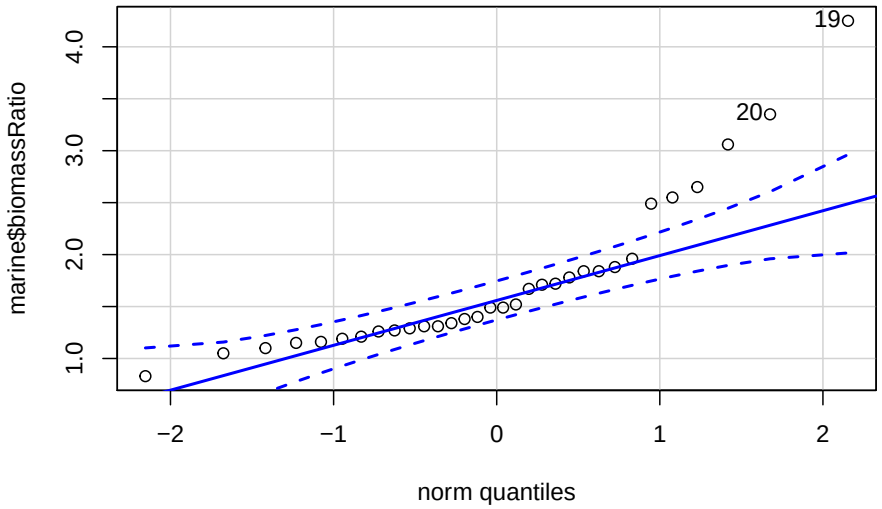
```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following object is masked from 'package:purrr':
##
##     some
```

```
qqPlot(marine$biomassRatio, distribution = "norm")
```




```
## [1] 19 20
```

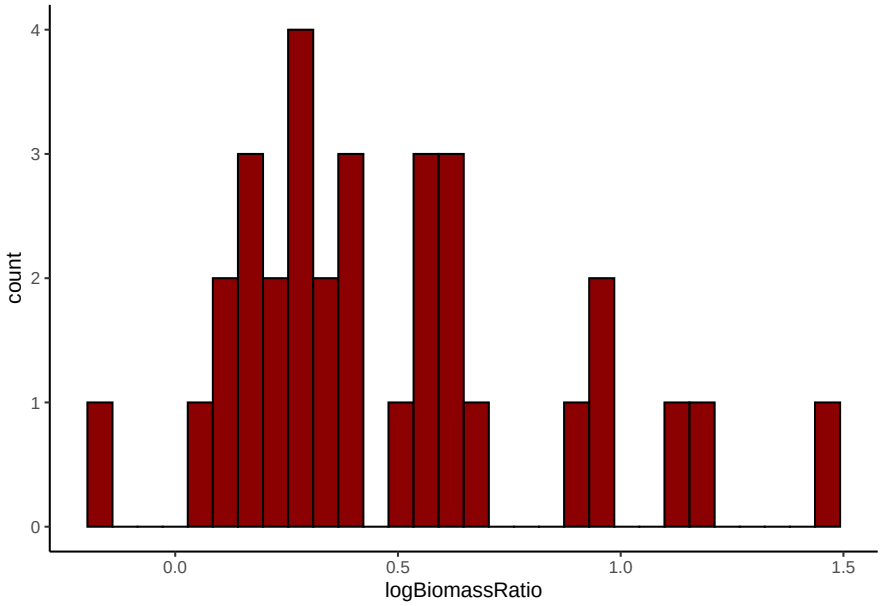
The function log() takes the natural logarithm of all the elements of a vector or variable. The following command puts the results into a new variable in the same data frame, marine.

```
marine = mutate(marine,logBiomassRatio =  log(biomassRatio))
```

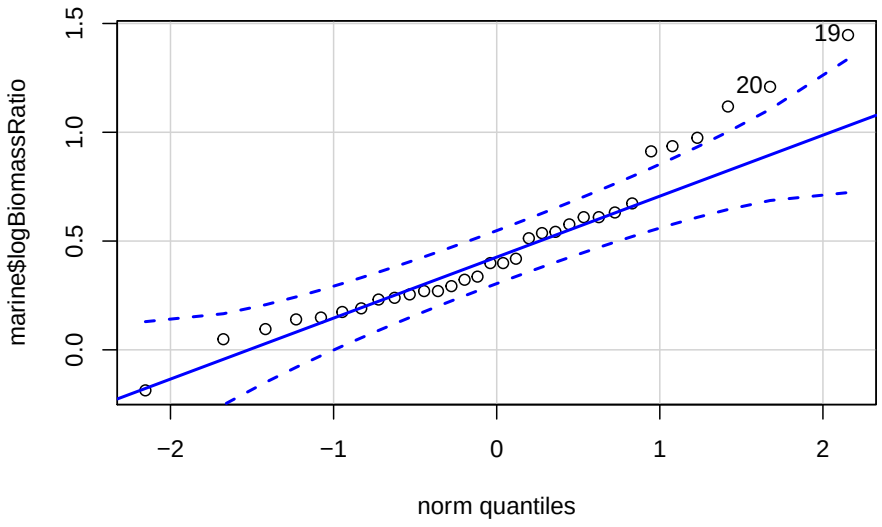
Histogram and QQ-Plot of the log-transformed marine biomass ratio.

```
ggplot(data = marine) +
  geom_histogram(mapping = aes(x = logBiomassRatio),col="black",fill="darkred")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
qqPlot(marine$logBiomassRatio, distribution = "norm")
```




```
## [1] 19 20
```

95% confidence interval of the mean using the log-transformed data.

```
t.test(marine$logBiomassRatio)$conf.int
```

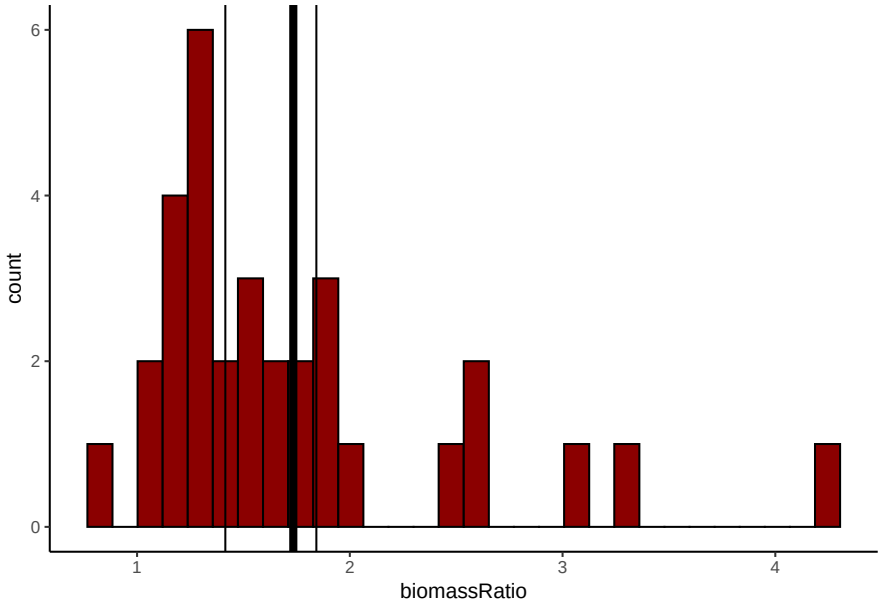
```
## [1] 0.3470180 0.6112365
```

```
## attr(,"conf.level")
## [1] 0.95
```

Back-transform lower and upper limits of confidence interval (exp is the inverse of the log function).

```
conf.int = exp(t.test(marine$logBiomassRatio)$conf.int )
meanBiomassRatio = mean(marine$biomassRatio)

ggplot(data = marine) +
  geom_histogram(mapping= aes(x = biomassRatio),col="black",fill="darkred") +
  geom_vline(xintercept = conf.int)+
  geom_vline(xintercept = meanBiomassRatio,lwd=2)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Non-Parametric Alternatives

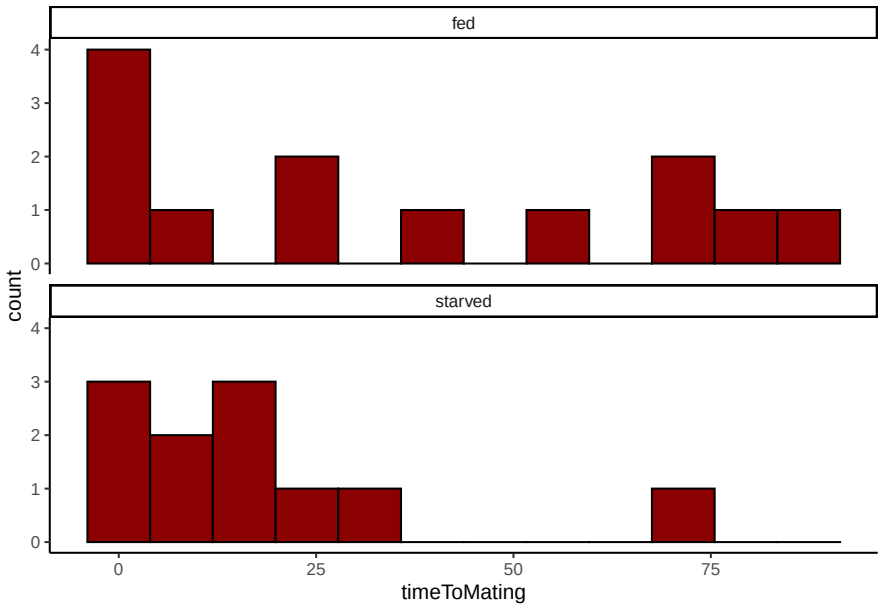
We use the Wilcoxon rank-sum test (equivalent to the Mann-Whitney U-test) comparing times to mating (in hours) of starved and fed female sagebrush crickets. We also apply the permutation test to the same data.

Read and inspect data:

```
cannibalism <- read.csv(url("http://www.zoology.ubc.ca/~schluter/WhitlockSchluter/wp-content/data/chapt
head(cannibalism)
```

```
##   feedingStatus timeToMating
## 1       starved          1.9
## 2       starved          2.1
## 3       starved          3.8
## 4       starved          9.0
## 5       starved          9.6
## 6       starved         13.0
```

```
ggplot(data = cannibalism) +
  geom_histogram(mapping = aes(x = timeToMating ),bins=12,color="black",fill="darkred") +
  facet_wrap( ~feedingStatus,nrow=2) +
  theme_classic()
```



We perfrom the Wilcoxon rank-sum test (equivalent to Mann-Whitney U-test)

```
wilcox.test(timeToMating ~ feedingStatus, data = cannibalism)
```

```
##
##  Wilcoxon rank sum test
##
## data:  timeToMating by feedingStatus
## W = 88, p-value = 0.3607
## alternative hypothesis: true location shift is not equal to 0
```

Permutation Test

We use a permutation test of the difference between mean time to mating of starved and fed crickets.We begin by calculating the observed difference between means (starved minus fed). The difference is -18.25734 in this data set.

```
# tapply is another way to calcualte the means of time to mating
# for each feeding status seperatly

cricketMeans <- tapply(cannibalism$timeToMating, cannibalism$feedingStatus, mean)

cricketMeans
```

```
##      fed  starved
## 35.98462 17.72727
```

```
diffMeans <- cricketMeans[2] - cricketMeans[1]
diffMeans
```

```
##   starved
## -18.25734
```

We choose to perform 1000 permutations and set a seed to make the analysis reproducible:

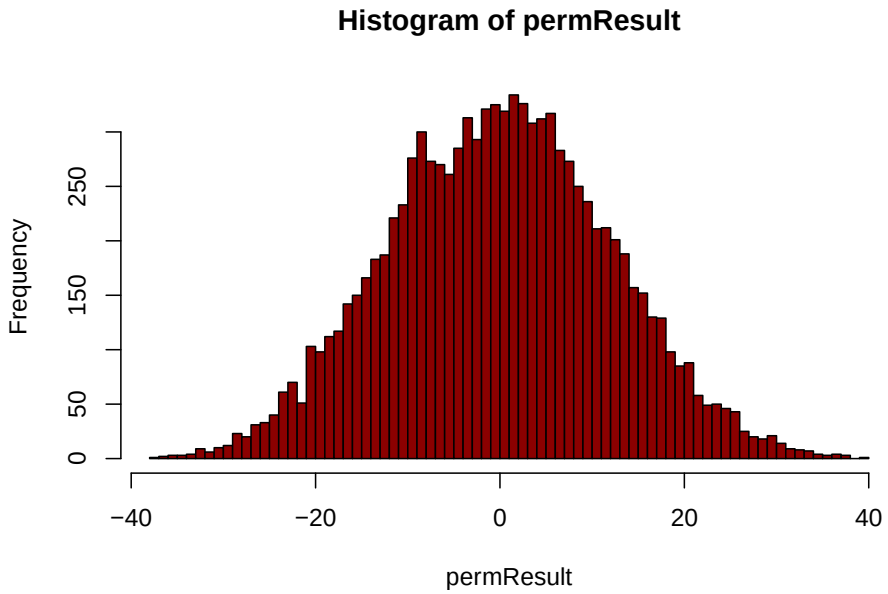
```r
nPerm <- 10000
set.seed(2048)
```

The following code implements aloop to permute the data many times (determined by nperm, investigate the function sample to see what it does). In the loop, i is just a counter that goes from 1 to nPerm by 1; each permuted difference is saved in the permResult.

```r
permResult <- vector() # initializes
for(i in 1:nPerm){
    # step 1: permute the times to mating
    permSample <- sample(cannibalism$timeToMating, replace = FALSE)
    # step 2: calculate difference betweeen means
    permMeans <- tapply(permSample, cannibalism$feedingStatus, mean)
    permResult[i] <- permMeans[2] - permMeans[1]
}
```

Plot null distribution based on the permuted differences.

```r
hist(permResult, right = FALSE, breaks = 100,col="darkred")
```



Use the null distribution to calculate an approximate P-value. This is the twice the proportion of the permuted means that fall below the observed difference in means, diffMeans (-18.25734 in this example). The following code calculates the number of permuted means falling below diffMeans.

```r
sum(as.numeric(permResult <= diffMeans))
```

```
## [1] 657
```

These commands obtain the fraction of permuted means falling below diffMeans.

```r
sum(as.numeric(permResult <= diffMeans)) / nPerm
```

```
## [1] 0.0657
```

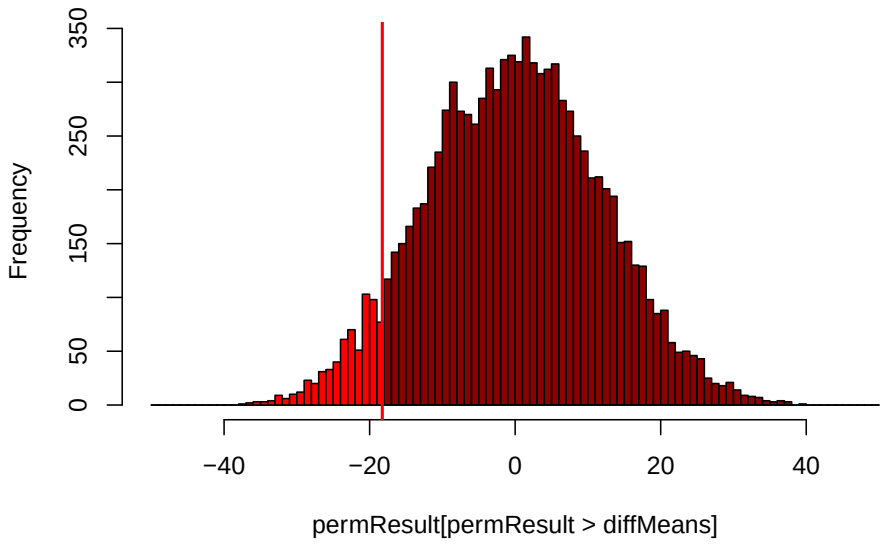
Finally, multiply by 2 to get the P-value for a two-sided test.

```r
2 * ( sum(as.numeric(permResult <= diffMeans)) / nPerm )
```

```
## [1] 0.1314
```

We can illustrate the permutaed differences and add the observed difference on to show the outcome of our test: the birght red colored bars indicate the proportion of permutation results that yielded a more exterme results as the on observed.

```
hist(permResult[permResult>diffMeans],breaks=seq(-50,50,by=1),col="darkred",main="")
hist(permResult[permResult<=diffMeans],breaks=seq(-50,50,by=1),add=T,col="red")
abline(v = diffMeans,col="red",lwd=2)
```



ANOVA

One way ANOVA

We compare the phase shift in the circadian rhythm of melatonin production in participants given alternative light treatments.

Read and inspect the data.

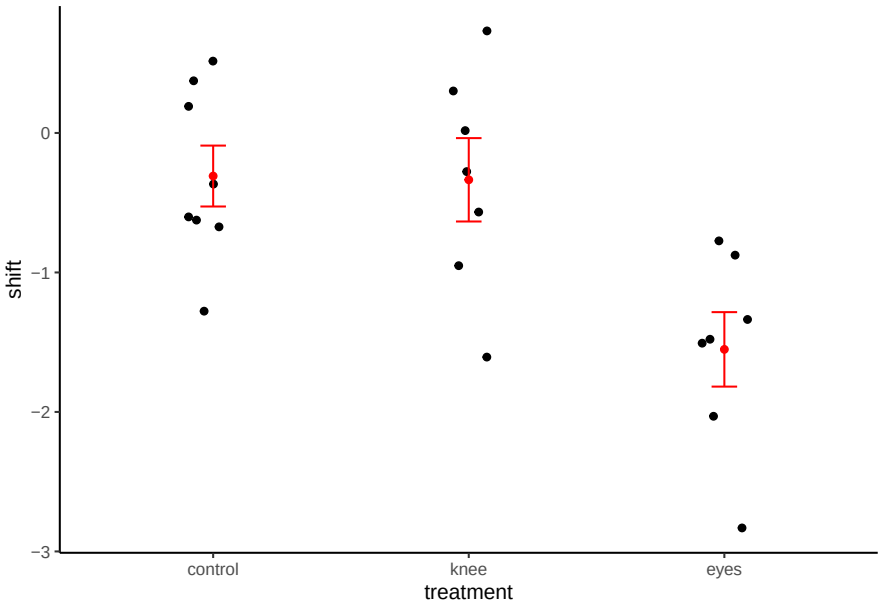
```
circadian <- read.csv(url("http://www.zoology.ubc.ca/~schluter/WhitlockSchluter/wp-content/data/chapter
```

Set the preferred ordering of groups in tables and graphs.

```
circadian$treatment <- factor(circadian$treatment,
    levels = c("control", "knee", "eyes"))



meanShift <- tapply(circadian$shift, circadian$treatment, mean)
sdevShift <- tapply(circadian$shift, circadian$treatment, sd)
n         <- tapply(circadian$shift, circadian$treatment, length)
mean_SD =data.frame(mean = meanShift, std.dev = sdevShift, n = n,treatment = c("control", "knee", "eyes
```

We plot the data showing the raw data points as well as the mean (including error bars shwoing one standard error in each direction).

```
ggplot(data = circadian) +
  geom_jitter(mapping = aes(x = treatment,y=shift),width=0.1) +
  geom_errorbar(data = mean_SD,mapping = aes(x = treatment,ymin=meanShift-sdevShift/sqrt(n), ymax=meanSh
  geom_point(data = mean_SD,mapping = aes(x = treatment,y=meanShift),color="red")
```



The first step involves fitting the ANOVA model to the data using lm ("lm" stands for "linear model", of which ANOVA is one type). Then we use the command anova to assemble the ANOVA table.

```
circadianAnova <- lm(shift ~ treatment, data = circadian)
anova(circadianAnova)
```

```
## Analysis of Variance Table
##
## Response: shift
##           Df Sum Sq Mean Sq F value   Pr(>F)
## treatment  2 7.2245  3.6122  7.2894 0.004472 **
## Residuals 19 9.4153  0.4955
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

R2 indicating the fraction of variation in the response variable "explained" by treatment. This is again done in two steps. The first step calculates a bunch of useful quantities from the ANOVA model object previously created with a lm command. The second step shows the R2 value.

```
circadianAnovaSummary <- summary(circadianAnova)
circadianAnovaSummary$r.squared
```

```
## [1] 0.4341684
```

Two way ANOVA

Analyze data from a factorial experiment investigating the effects of herbivore presence, height above low tide, and the interaction between these factors, on abundance of a red intertidal alga using two-factor ANOVA.

Read and inspect the data.

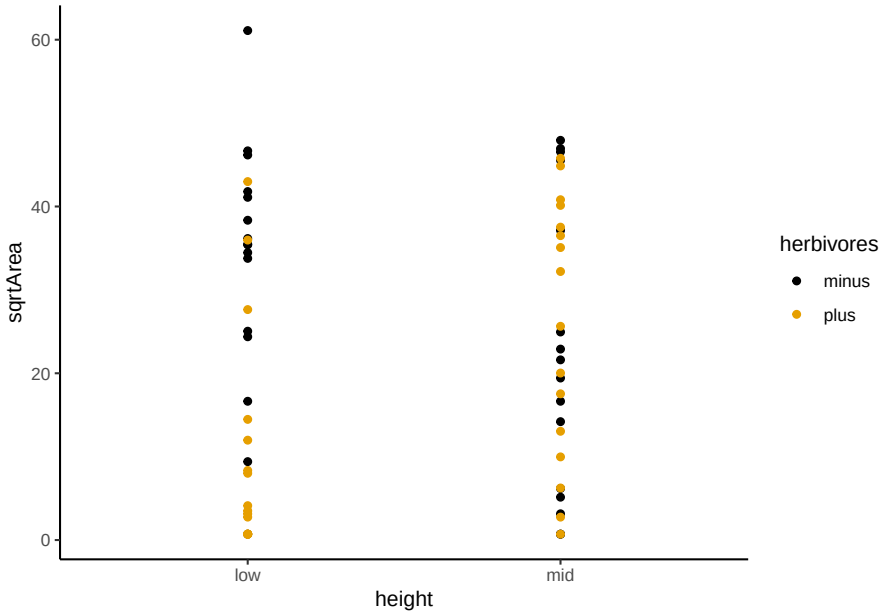
```
algae <- read.csv(url("http://www.zoology.ubc.ca/~schluter/WhitlockSchluter/wp-content/data/chapter18/c
head(algae)
```

```
##   height herbivores  sqrtArea
## 1    low      minus  9.405573
```

```
## 2    low      minus 34.467736
## 3    low      minus 46.673485
## 4    low      minus 16.642139
## 5    low      minus 24.377498
## 6    low      minus 38.350604
```

We first take a look at the data:

```
ggplot(data = algae,  aes(x = height, color = herbivores, group = herbivores, y = sqrtArea)) +
  geom_point()
```

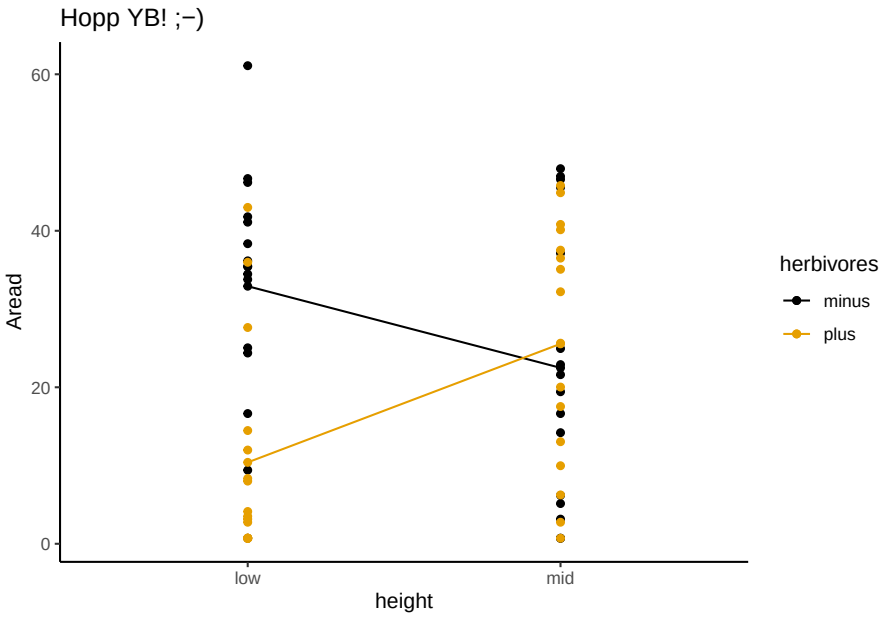


This is a mess so we next add some summary statistics of the data: the mean Area for each combination of explanatory variables.

```
ggplot(data = algae) +
  aes(x = height, color = herbivores, group = herbivores, y = sqrtArea) +
    geom_point()+
  stat_summary(fun.y = mean, geom = "point") +
  stat_summary(fun.y = mean, geom = "line")+
  labs(title="Hopp YB! ;-)",
        x ="height", y = "Aread")
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```



It looks like there is some interaction between *height* and *herbivores*, but there is also quite some noise so it is difficult to tell.

Let us first fit a null model having both main effects but no interaction term. We can do this again with *lm* (*note:* conceptually a t-test, an ANOVA and linear regression are all pretty much the same thing and hence the function lm can be used in all these cases):

```
algaeNoInteractModel <- lm(sqrtArea ~ height + herbivores, data = algae)

summary(algaeNoInteractModel)
```

```
##
## Call:
## lm(formula = sqrtArea ~ height + herbivores, data = algae)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.171 -13.748  -2.235  15.433  34.576
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)      26.520      3.602   7.362 5.53e-10 ***
## heightmid         2.358      4.160   0.567   0.5729
## herbivoresplus   -9.722      4.160  -2.337   0.0227 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.64 on 61 degrees of freedom
## Multiple R-squared:  0.0866, Adjusted R-squared:  0.05665
## F-statistic: 2.892 on 2 and 61 DF,  p-value: 0.06311
```

Now fit the full model, with interaction term included.

```
algaeFullModel <- lm(sqrtArea ~ height * herbivores, data = algae)
```

Finally we cretea na ANOVA table that compares the two models:

```
anova(algaeNoInteractModel, algaeFullModel)
```

```
## Analysis of Variance Table
##
## Model 1: sqrtArea ~ height + herbivores
## Model 2: sqrtArea ~ height * herbivores
##   Res.Df   RSS Df Sum of Sq      F   Pr(>F)
## 1     61 16888
## 2     60 14270  1      2617 11.003 0.001549 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Linear Regression

We test the null hypothesis of zero regression slope. The data are from an experiment investigating the effect of plant species diversity on the stability of plant biomass production.

Read and inspect the data.

```
prairie <- read.csv(url("http://www.zoology.ubc.ca/~schluter/WhitlockSchluter/wp-content/data/chapter17/
head(prairie)
```

```
##   nSpecies biomassStability
## 1        1             7.47
## 2        1             6.74
## 3        1             6.61
## 4        1             6.40
## 5        1             5.67
## 6        1             5.26
```

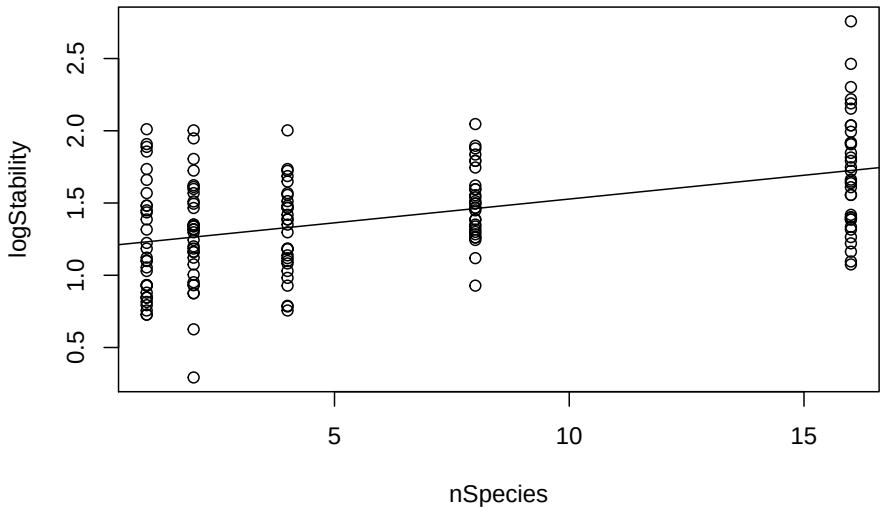
We perfrom a Log-transform on the varibale stability and include the new variable in the data frame. Inspect the data frame once again to make sure the function worked as intended.

```
prairie = mutate(prairie,logStability = log(biomassStability))
head(prairie)
```

```
##   nSpecies biomassStability logStability
## 1        1             7.47     2.010895
## 2        1             6.74     1.908060
## 3        1             6.61     1.888584
## 4        1             6.40     1.856298
## 5        1             5.67     1.735189
## 6        1             5.26     1.660131
```

Scatter plot with regression line in base R:

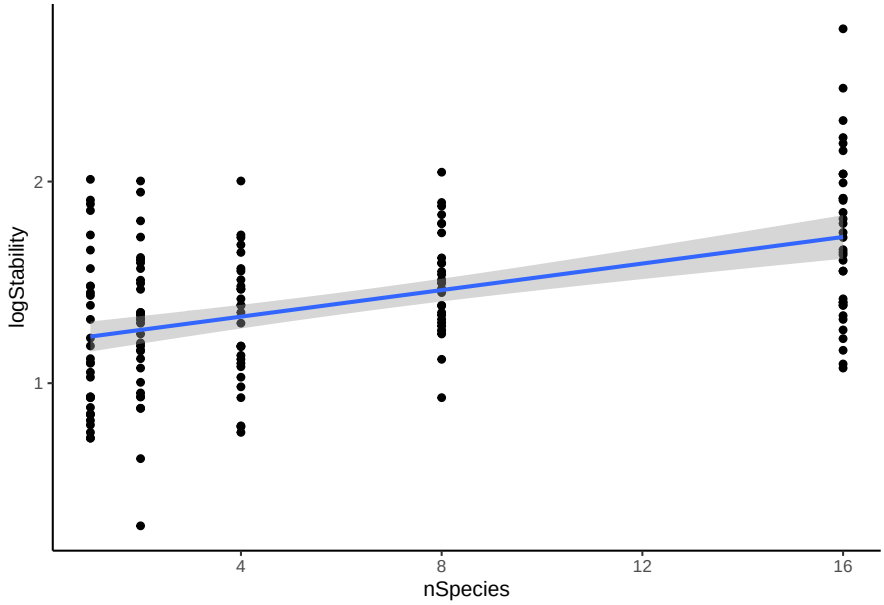
```
plot(logStability ~ nSpecies, data = prairie)
prairieRegression <- lm(logStability ~ nSpecies, data = prairie)
abline(prairieRegression)
```



Scatter plot with regression line in ggplot:

```
ggplot(data = prairie,aes(y=logStability,x=nSpecies)) +
  geom_point()+
  geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
prairieRegression <- lm(logStability ~ nSpecies, data = prairie)
summary(prairieRegression)
```

```
##
## Call:
## lm(formula = logStability ~ nSpecies, data = prairie)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.97148 -0.25984 -0.00234  0.23100  1.03237
##
```

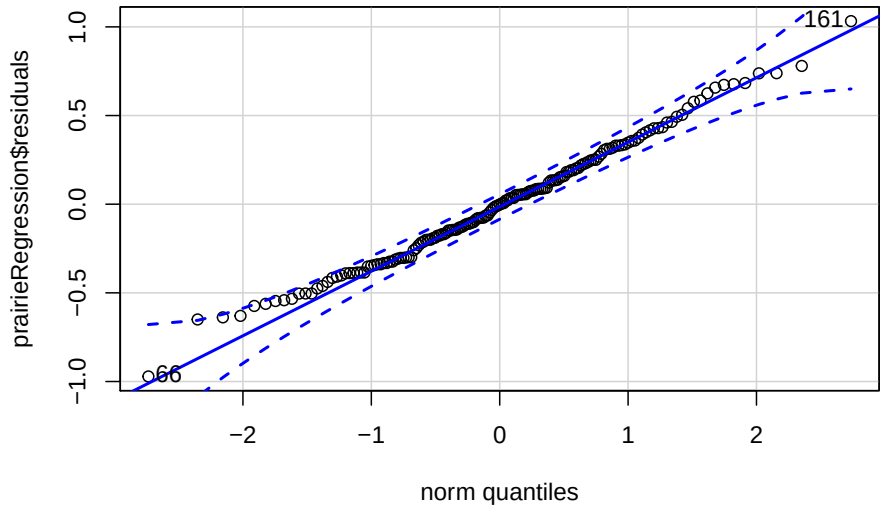
```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.198294   0.041298  29.016  < 2e-16 ***
## nSpecies    0.032926   0.004884   6.742 2.73e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3484 on 159 degrees of freedom
## Multiple R-squared:  0.2223, Adjusted R-squared:  0.2174
## F-statistic: 45.45 on 1 and 159 DF,  p-value: 2.733e-10
```

```
confint(prairieRegression)
```

```
##                  2.5 %     97.5 %
## (Intercept) 1.11673087 1.27985782
## nSpecies    0.02328063 0.04257117
```

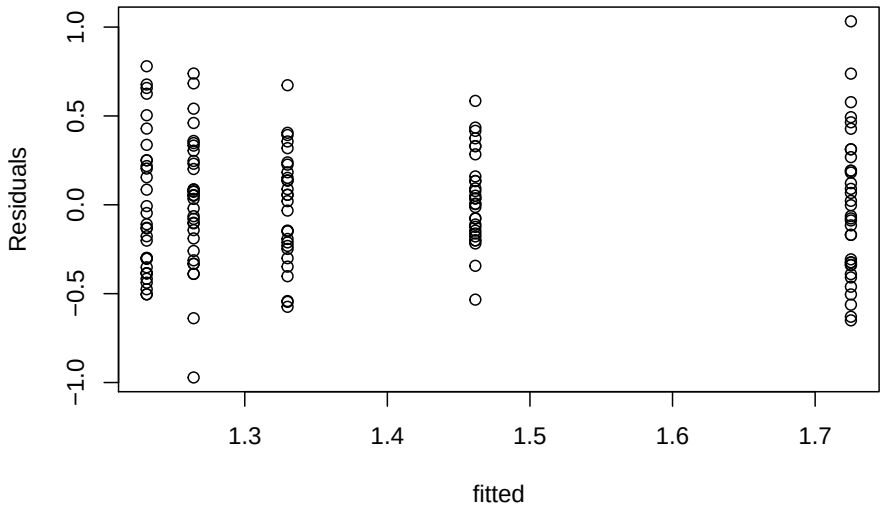
Finally, we check our modelling assumptions with a QQ plot of the residuals as well as a Tukey-Anscombe plot:

```
qqPlot(prairieRegression$residuals,dist="norm")
```




```
## [1] 161  66
```

```
plot(prairieRegression$fitted.values,prairieRegression$residuals,xlab="fitted",ylab="Residuals")
```



Additional notes to the course slides

Law of total probability

Assume that $B_i, i = 1, 2, 3, \ldots$ is a set of pairwise disjoint events whose union is the entire sample space (that is, every possible event that can happen is covered by exactly one of the B_is).

As an example: consider rolling a six-sided die. Then the event $B_i (i = 1, ..., 6)$ is the event that you roll the number i. All events are disjoint and together they cover everything that can happen.

For any event A we can then write

$$P(A) = \sum_i P(A \cap B_i)$$

or, equivalently,

$$P(A) = \sum_i P(A \mid B_i) P(B_i).$$

This figure is helpful for understanding what is going on here:

Example:

We roll two dice. We want to know the probaility that the first roll has a smaller outcome than the second roll.

We call the events $B_i, i = 1, ..., 6$ the event that the frist roll shows i dots. So B_2 is the event that you roll a 2 with the first roll. (Think about why this satisfies the conditons for the events B_i stated in the law of total probaility.)

Let us clall X_1 the outcome of the first roll and X_2 the outcome of the second roll: $P(B_2)$ is then $P(X_1 = 2)$. The event A is the event that $X_1 > X_2$. We use the formula

$$P(A) = \sum_i P(A \mid B_i) P(B_i).$$

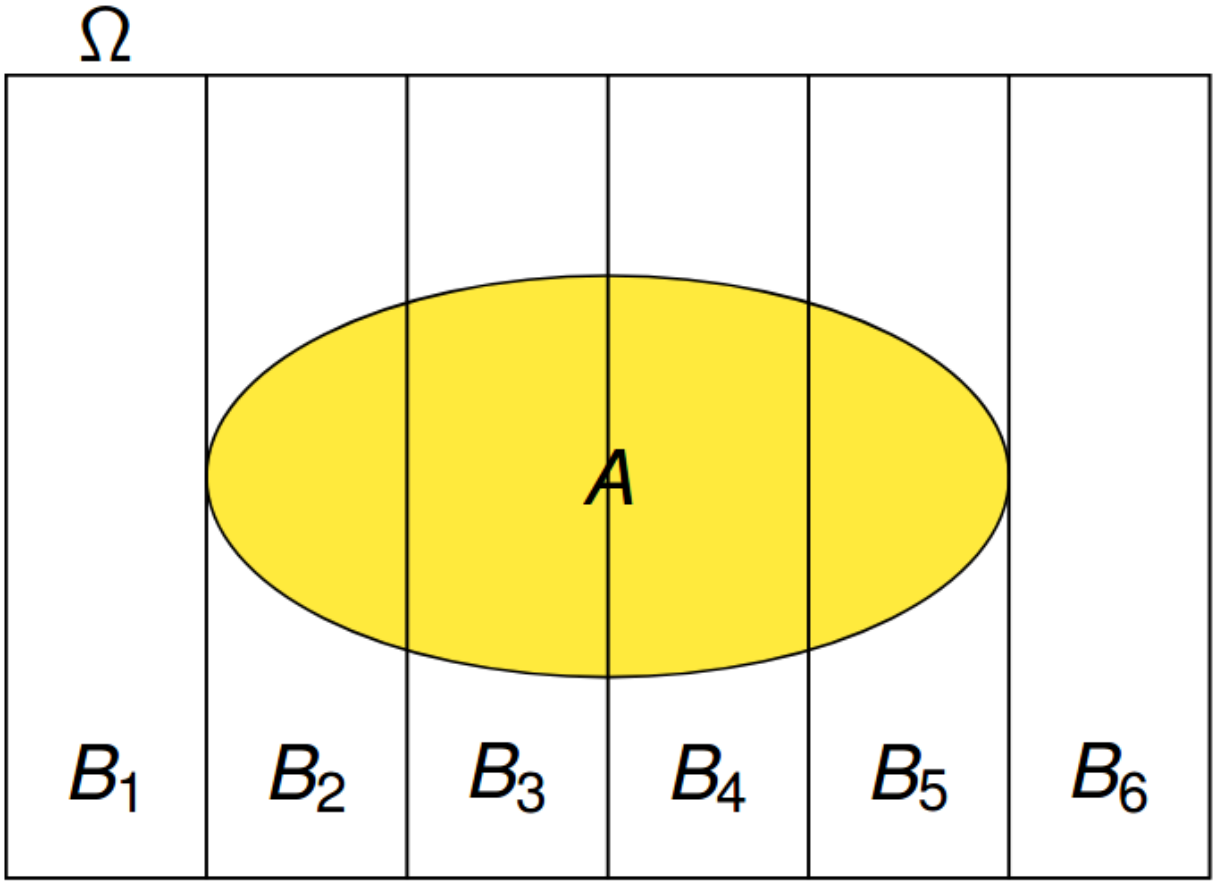


Figure 1: Illustration of the law of total probability

If you roll a i with the first die, there are 6-i possible outcomes for the second roll that are larger than i. This means: $P(A \mid B_1) = 5/6$ $P(A \mid B_2) = 4/6$ $P(A \mid B_3) = 3/6$ $P(A \mid B_4) = 2/6$ $P(A \mid B_5) = 1/6$ $P(A \mid B_6) = 0$

and furthermore

$P(B_i) = 1/6$ for each i.

Then

$P(A) = 5/6 * 1/6 + 4/6 * 1/6 + 3/6 * 1/6 + 2/6 * 1/6 + 1/6 * 1/6 + 0 * 1/6$

$= 0.417$.

Let us test this with a small R simulation:

```
# We do 10000 die rolls for each die:
die.1 = sample.int(6, size = 10000,replace=T)
die.2 = sample.int(6, size = 10000,replace=T)

# calcualte the proportion of times die 2 is larger than die 1
sum(die.2>die.1)/10000
```

```
## [1] 0.4129
```