STATS 2107
Statistical Modelling and Inference II
First year R recap

Sharon Lee, Matt Ryan

Semester 2 2022

Data

Entering data

If you want to enter data you can use the `c()` command. The `c` is short for concatenation, *i.e.,* it forces the values together. For example imagine that I would like to store the numbers:

```
1, 2, 5, 10
```

We use the command

```
my_numbers  <- c(1,2,5,10)
```

Variables

Lets break this down:

- `c(1,2,5,10)` puts the numbers together in a vector,
- `<-` the gets symbol lets us add a label or variable name to the vector, think of it as pointing to the label,
- `my_number` is the label or variable name.

I can now get this number by typing at the command line the label:

```
my_numbers
```

```
## [1]  1  2  5 10
```

This storing of data with a label is called a **variable**.

Functions

The command `c()` is called a function. Functions are bits of code stored in R that let us perform actions on data, they are made up of three parts:

- **name**: the name identifies the function, in this case just `c`,
- **arguments**: we can pass stuff into the function, this is the stuff in the brackets, in our case `1,2,5,10`, and
- **return value**: the function will pass back (return) stuff to you, in our case it returns a vector of numbers we stored.

Summary statistics

The first functions that we will look at is those that give us summary statistics. Lets start with the sample mean:

Sample mean

The function to do this is `mean()`. It is built into R so we do not need to load it. We need some data. R has some data built in so lets use this:

```
data("iris")
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

So I loaded the data with `data("iris")`, and then had a look at the first few rows with `head(iris)`. This data is stored in a dataframe.

Dataframes

Data in R is most often stored as a dataframe, which is a spreadsheet-like arrangement of data. The form is

- each row is a subject, and
- each column is a measurement (variable) of the subjects.

In our case, the subjects are Irises, which are a type of flower. The variable names are given at the top (header) and we have the Sepal Length, Sepal Width, Petal Length, Petal Width, and the Species. We can get the values of individual variables with

```
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

This is read as:

- **iris$**: inside the dataframe `iris`
- **Sepal.Length**: there is a column called `Sepal.Length` - go get it.

Lets get back to our first summary statistic and calculate the mean Sepal Length with

```
mean(iris$Sepal.Length)
```

```
## [1] 5.843333
```

The argument was the lengths (`Sepal.Length`), the function `mean()`, and the value returned was 5.8433333.

More summary statistics

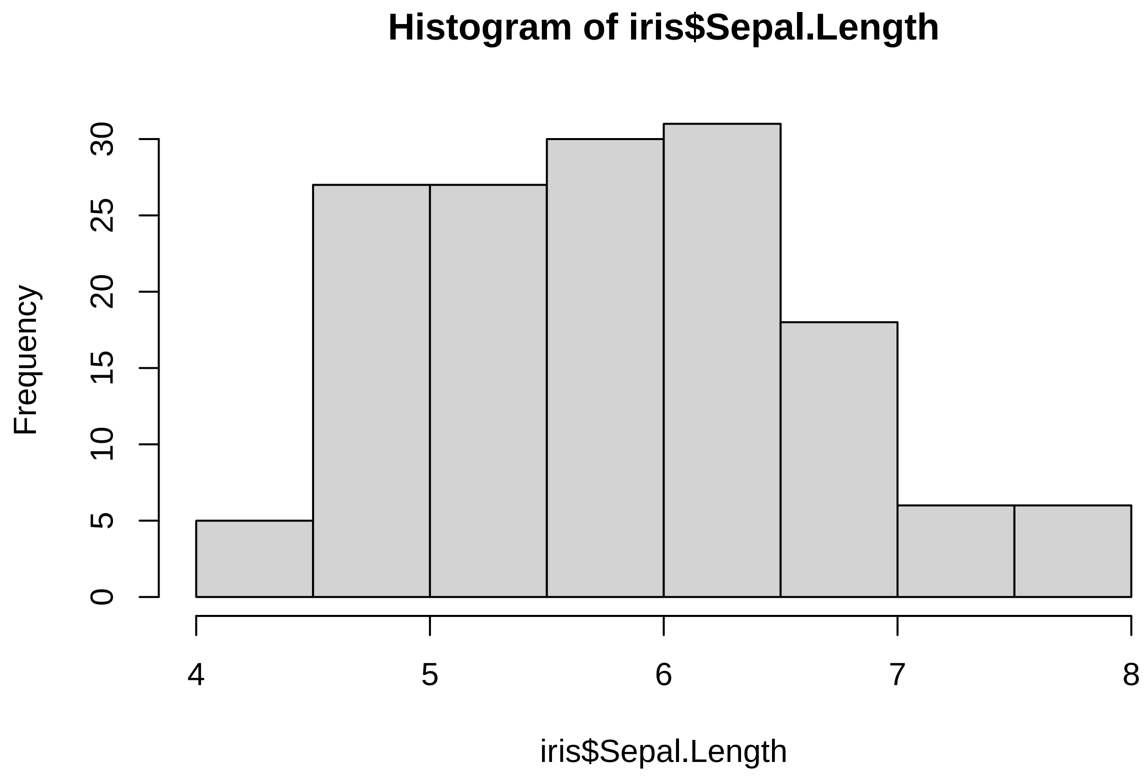
Other summary statistics that you might need are

Summary stat	function
Median	`median()`
Variance	`var()`
Standard deviation	`sd()`
Five number summary	`fivenum()`
General summary	`summary()`

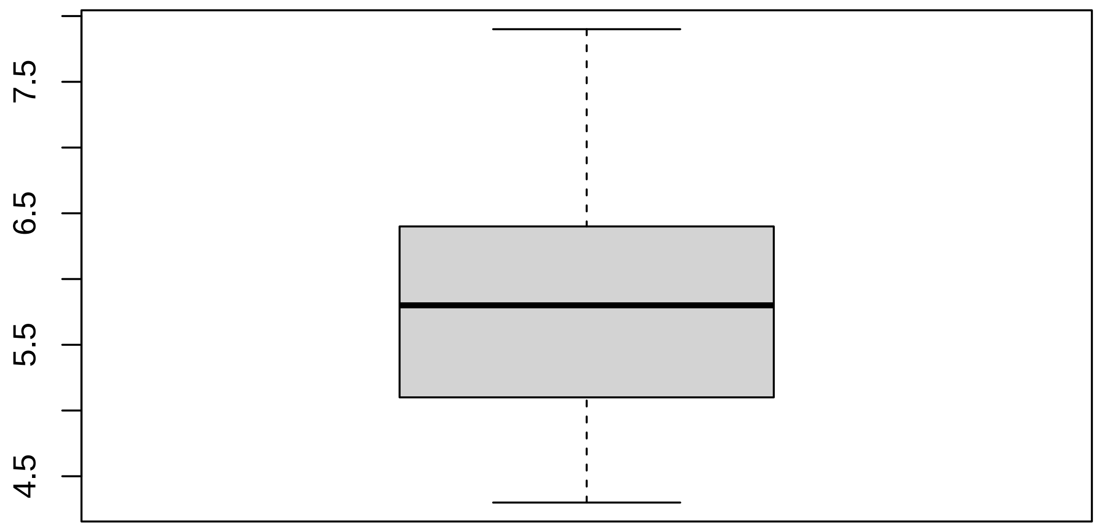
Basic plots

We can get histograms and box-plots with `hist()` and `boxplot()` respectively.

```
hist(iris$Sepal.Length)
```

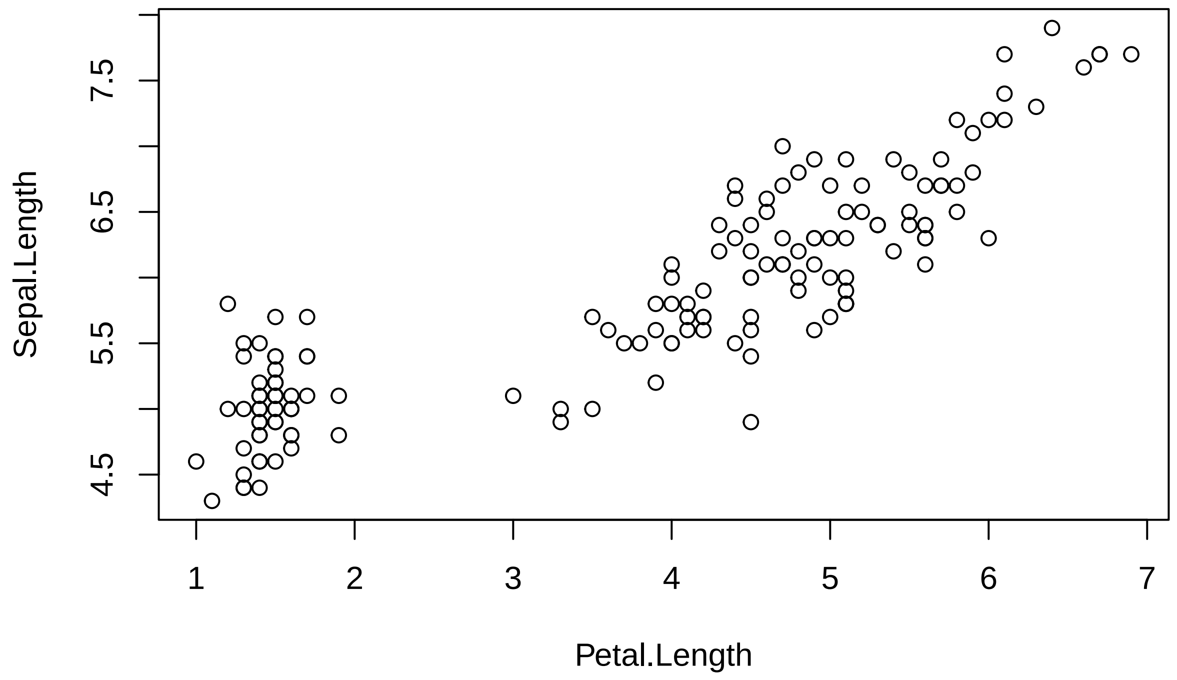


```
boxplot(iris$Sepal.Length)
```



Also you can get a scatter-plot

```r
plot(Sepal.Length ~ Petal.Length, data = iris)
```



There is better ways to do this - ggplot2 - see SMI.

Probability commands

R has built in commands to do probability calculations.

Binomial distribution

Let

$$X \sim Bin(n, p),$$

then to calculate the CDF, *i.e*,

$$P(X \leq x)$$

we use

$$\texttt{pbinom}(\texttt{x}, \texttt{n}, \texttt{p}).$$

For the PMF, *i.e.*

$$P(X = x)$$

we use

$$\texttt{dbinom}(\texttt{x}, \texttt{n}, \texttt{p}).$$

If we would like to simulate N random binomial variables we use

```
rbinom(N, n, p)
```

Finally if we would like to do an inverse probability calculation, *i.e*,

Find c such that

$$P(X \leq c) = q,$$

we use

$$\texttt{qbinom}(\texttt{q}, \texttt{n}, \texttt{p}).$$

Example

We toss a fair coin 10 times and count the number of heads X, so we have

$$X \sim Bin(10, 1/2)$$

The probability that we toss exactly 3 heads is

```
dbinom(3, 10, 0.5)
```

```
## [1] 0.1171875
```

The probability that we toss 3 or less heads is

```
pbinom(3, 10, 0.5)
```

```
## [1] 0.171875
```

To simulate repeating this experiment 100 times

```
rbinom(100, 10, 0.5)
```

```
##   [1] 4 3 3 4 6 6 3 6 4 5 6 6 2 7 5 6 9 4 6 5 7 4 4 7 8 5 5 2 5 6 5 7 4 1 6 5 3
##  [38] 2 8 4 6 4 7 4 4 6 5 3 6 5 2 0 7 4 4 6 6 4 7 6 5 3 8 5 8 4 4 6 3 4 4 4 4 5
##  [75] 6 5 7 5 4 4 4 7 5 4 3 6 6 5 5 5 2 6 3 3 5 5 5 8 4 7
```

The value c such that the probability the number of head is c or less is 0.1 is

```
qbinom(0.1, 10, 0.5)
```

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

Normal distribution

The equivalent commands for the normal are

Calculation	command
CDF	`pnorm(x, mean, sd)`
PDF	`dnorm(x, mean, sd)`
Inverse CDF	`qnorm(p, mean, sd)`
Simulate	`rnorm(N, mean, sd)`

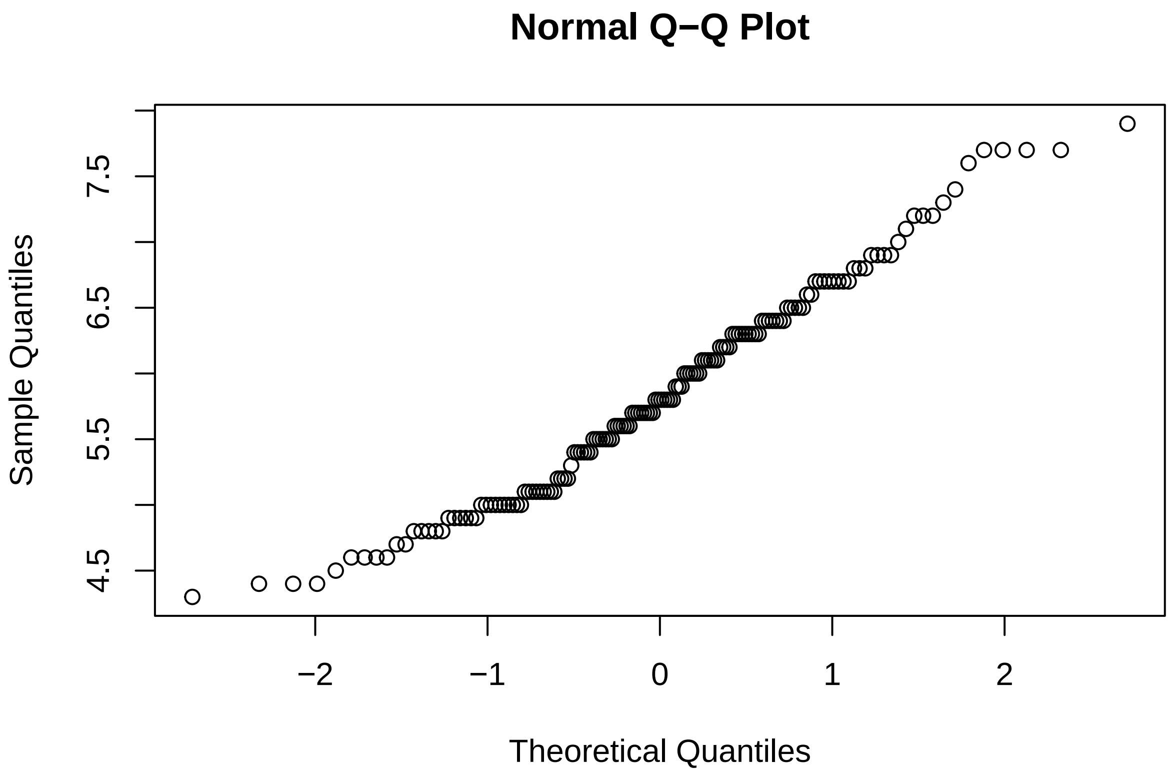
These commands generalise to all sorts of distributions. In general, you have

Calculation	command
CDF	`p...`, *i.e.* `pchisq(x, df)`
PDF	`d...`, *i.e.* `dpois(x, lambda)`
Inverse CDF	`q...`, *i.e* `qt(x, df)`
Simulate	`r...`, *i.e.* `rbeta(n, shape1, shape2)`

QQ-plot

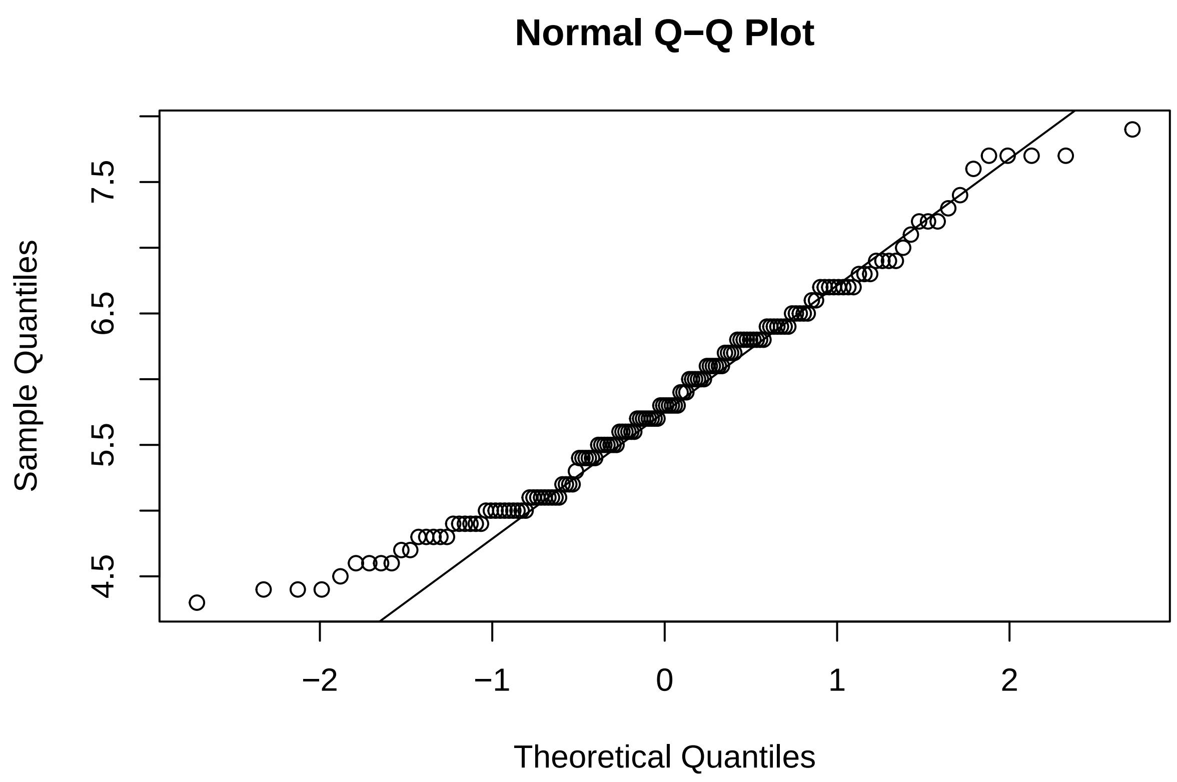
We can test that data is normally distributed with the QQ-plot.

```
qqnorm(iris$Sepal.Length)
```



We can add a reference line:

```
qqnorm(iris$Sepal.Length)
qqline(iris$Sepal.Length)
```



Inference for means.

Most of these are done with the t-test `t.test()`.

One sample t-test

Let's test the null hypothesis that the population mean Sepal length is 5, *i.e.*:

$$H_0 : \mu = 5,$$

where μ is the mean Sepal length.

```
t.test(x = iris$Sepal.Length, mu = 5)
```

```
##
##  One Sample t-test
##
## data:  iris$Sepal.Length
## t = 12.473, df = 149, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 5
## 95 percent confidence interval:
##  5.709732 5.976934
## sample estimates:
## mean of x
##  5.843333
```

So the arguments are

- the data `x` is the iris sepal length,
- the null value `mu` is 5.

Notice that I have named the arguments in the t-test function. If the arguments are not labelled, R uses the order they are in to assign them.

Two-sample t-test

As an example of a two-sample t-test we will use the `sleep` data

```
data(sleep)
```

This looks at two sleep-drugs contained in the `group` variable, and the extra sleep that the students got, contained in the `extra` variable.

The null and alternative hypotheses are

$$H_0 : \mu_1 = \mu_2$$
$$H_a : \mu_1 \neq \mu_2,$$

where μ_1 is the mean extra sleep on Drug 1, and μ_2 is the mean extra sleep on Drug 2.

The box-plot is given in Figure 1. It looks like Drug 2 gives more sleep.

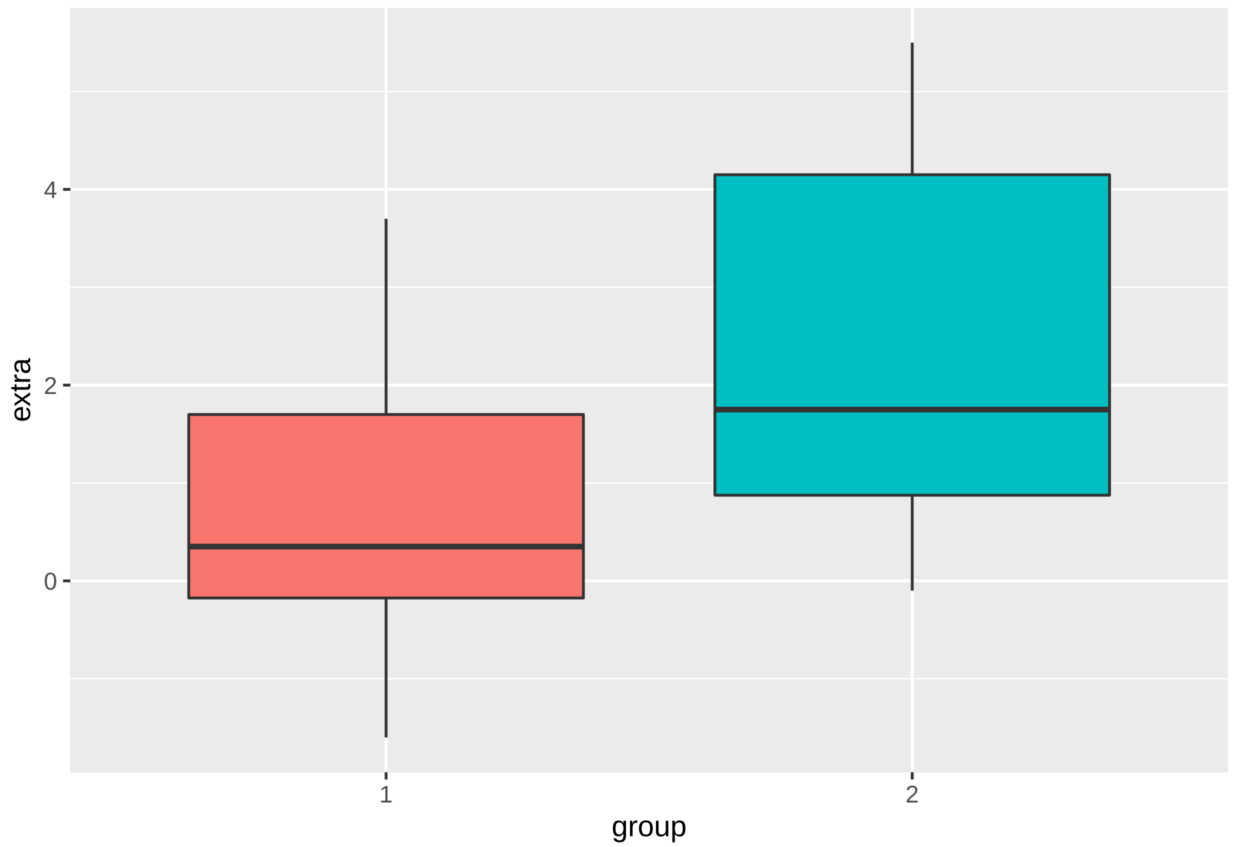


Figure 1: Boxplots of extra sleep for drug 1 and drug 2.

```
t.test(extra ~ group, data = sleep)
```

```
##
##  Welch Two Sample t-test
##
## data:  extra by group
## t = -1.8608, df = 17.776, p-value = 0.07939
## alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
## 95 percent confidence interval:
##  -3.3654832  0.2054832
## sample estimates:
## mean in group 1 mean in group 2
##            0.75            2.33
```

Notice how we let R know which is the response variable and which is the treatment variable using the ~ symbol. Variables to the left of ~ (`extra`) are the response, while variables to the right of ~ (`group`) are the treatments.

Matched pairs t-test

Matched pairs can also be used using `t.test()`. The data set is the `immer` dataset that has a Barley field trial. This is in an extra add-on which we must load.

```
library(MASS)
data("immer")
head(immer)
```

```
##   Loc Var    Y1    Y2
## 1  UF   M  81.0  80.7
## 2  UF   S 105.4  82.3
## 3  UF   V 119.7  80.4
## 4  UF   T 109.7  87.2
## 5  UF   P  98.3  84.2
## 6   W   M 146.6 100.4
```

Now we can do a matched-pairs t-test.

```
t.test(x = immer$Y1, y = immer$Y2, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  immer$Y1 and immer$Y2
## t = 3.324, df = 29, p-value = 0.002413
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   6.121954 25.704713
## sample estimates:
## mean of the differences
##                15.91333
```

Note that I added the argument `paired = TRUE` to let R know that it is a matched-paired t-test.

Wilcoxin rank sum

The non-parametric version of the two-sample t-test built into R is the Wilcoxon rank sum test. We will repeat the `sleep` data analysis with a Wilcoxon test.

```
wilcox.test(extra ~ group, data = sleep)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  extra by group
## W = 25.5, p-value = 0.06933
## alternative hypothesis: true location shift is not equal to 0
```

Linear regression

To illustrate linear regression, I will use the FVC dataset. I have loaded this in and stored it as a modern type of dataframe - the tibble. Basically, it just gives a nicer appearance. For more about loading in excel data and tribbles see SMI.

```
fvc  <- readxl::read_excel("FVC.xlsx")
```

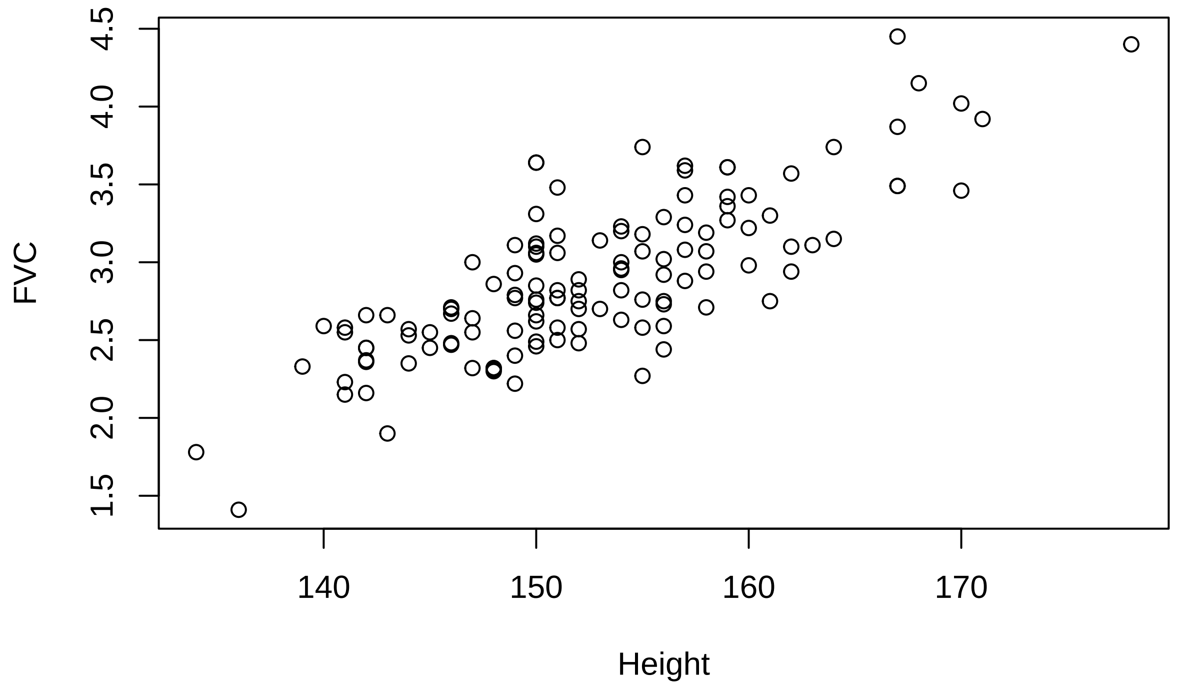
```
fvc
```

```
## # A tibble: 127 x 3
##      FVC Height Weight
##    <dbl>  <dbl>  <dbl>
##  1  2.75    156     51
##  2  2.66    142     37
##  3  2.32    148     35
##  4  4.4     178     58
##  5  2.7     146     38
##  6  2.35    144     35
##  7  3.42    159     47
##  8  3.12    150     52
##  9  2.76    150     43
## 10  3.02    156     46
## # ... with 117 more rows
```

Scatter plot

Let start with a scatter-plot of FVC on Height.

```
plot(FVC~Height, data = fvc)
```



Notice again the use of the `~` to denote the response variable `FVC` and the predictor variable `Height`.

Linear regression.

To fit the model:

$$FVC_i = \beta_0 + \beta_1 Height_i + \varepsilon_i, i = 1, \ldots, n$$
$$\varepsilon_i \sim \text{i.i.d. } N(0, \sigma^2),$$

we use

```
fvc.lm  <- lm(FVC ~ Height, data = fvc)
```

lets break this down

- `lm()`: **L**inear **M**odel function
- `FVC ~ Height`: FVC is the response variable, Height is the predictor variable.
- `fvc.lm  <-`: save the results for later and label it `fvc.lm`

We can get the estimates and other information about the linear model with:

```
summary(fvc.lm)
```

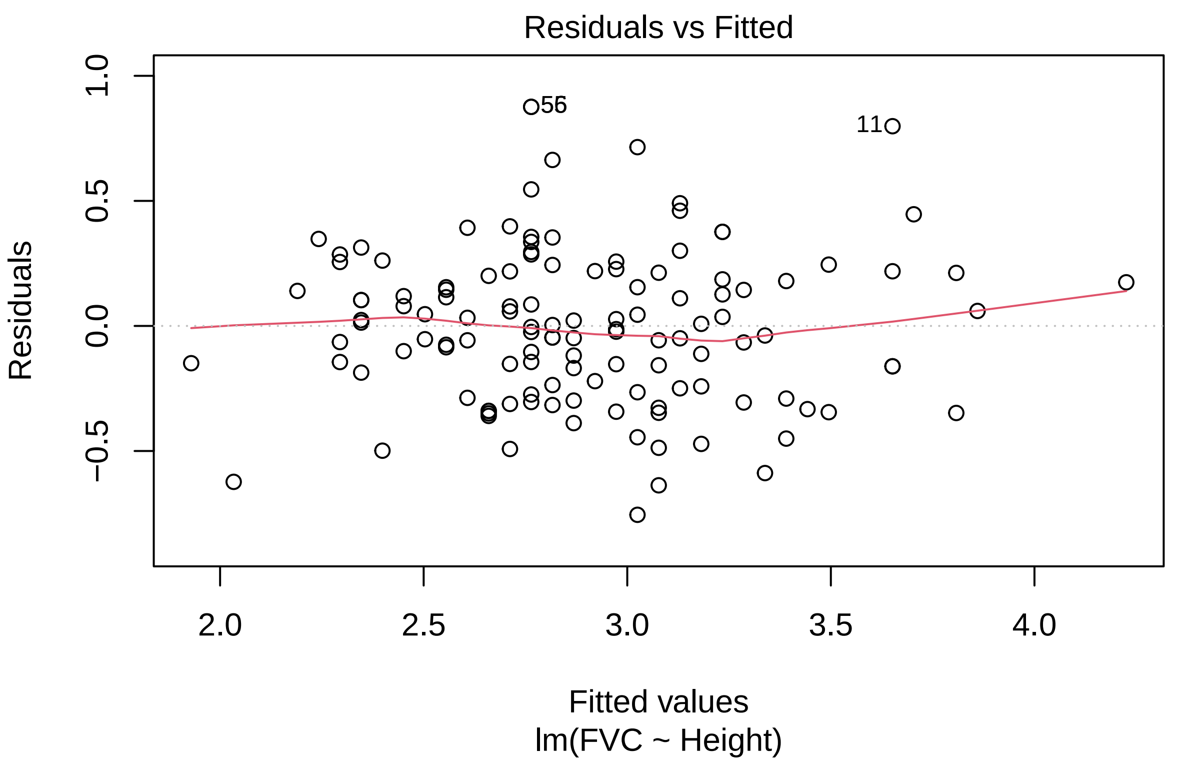
```
##
## Call:
## lm(formula = FVC ~ Height, data = fvc)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.75507 -0.23898 -0.00411  0.21238  0.87589
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.064961   0.552593  -9.166 1.24e-15 ***
## Height       0.052194   0.003618  14.426  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3137 on 125 degrees of freedom
## Multiple R-squared:  0.6248, Adjusted R-squared:  0.6218
## F-statistic: 208.1 on 1 and 125 DF,  p-value: < 2.2e-16
```

Model checking

For model checking we need to following plots

Residual versus fitted plot

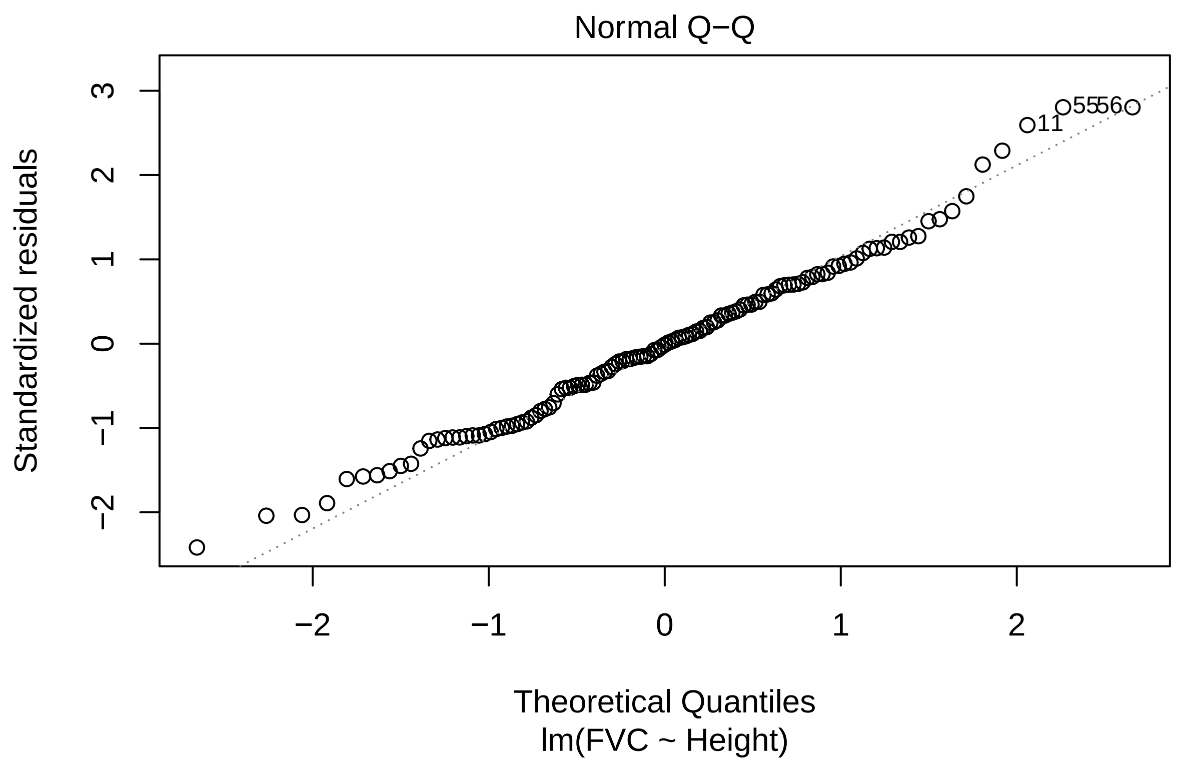
```
plot(fvc.lm, which = 1)
```



Notice that when I use plot with a linear model, R will produce lots of different plots. The argument `which = 1` tells R which one I want.

QQ-plot of residuals

```
plot(fvc.lm, which = 2)
```



Prediction

We can get predicted values, prediction intervals and confidence intervals using the `predict()` function. For a height of 150cm, we have the following:

```
predict(fvc.lm, newdata = data_frame(Height = 150))
```

```
##         1
## 2.764106
```

```
predict(fvc.lm, newdata = data_frame(Height = 150), interval = "confidence")
```

```
##        fit      lwr      upr
## 1 2.764106 2.706084 2.822127
```

```
predict(fvc.lm, newdata = data_frame(Height = 150), interval = "prediction")
```

```
##        fit      lwr      upr
## 1 2.764106 2.140573 3.387638
```

Multiple regression

Want more predictor variable - your wish is my command - the `lm()` command

```
fvc.lm2  <- lm(FVC ~ Height + Weight, data = fvc)
```

```
summary(fvc.lm2)
```

```
##
## Call:
## lm(formula = FVC ~ Height + Weight, data = fvc)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.65960 -0.21612 -0.00273  0.17748  0.88240
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.799689   0.664114  -5.721 7.48e-08 ***
## Height       0.039651   0.005252   7.549 8.23e-12 ***
## Weight       0.014871   0.004652   3.197  0.00176 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3027 on 124 degrees of freedom
## Multiple R-squared:  0.6533, Adjusted R-squared:  0.6477
## F-statistic: 116.8 on 2 and 124 DF,  p-value: < 2.2e-16
```

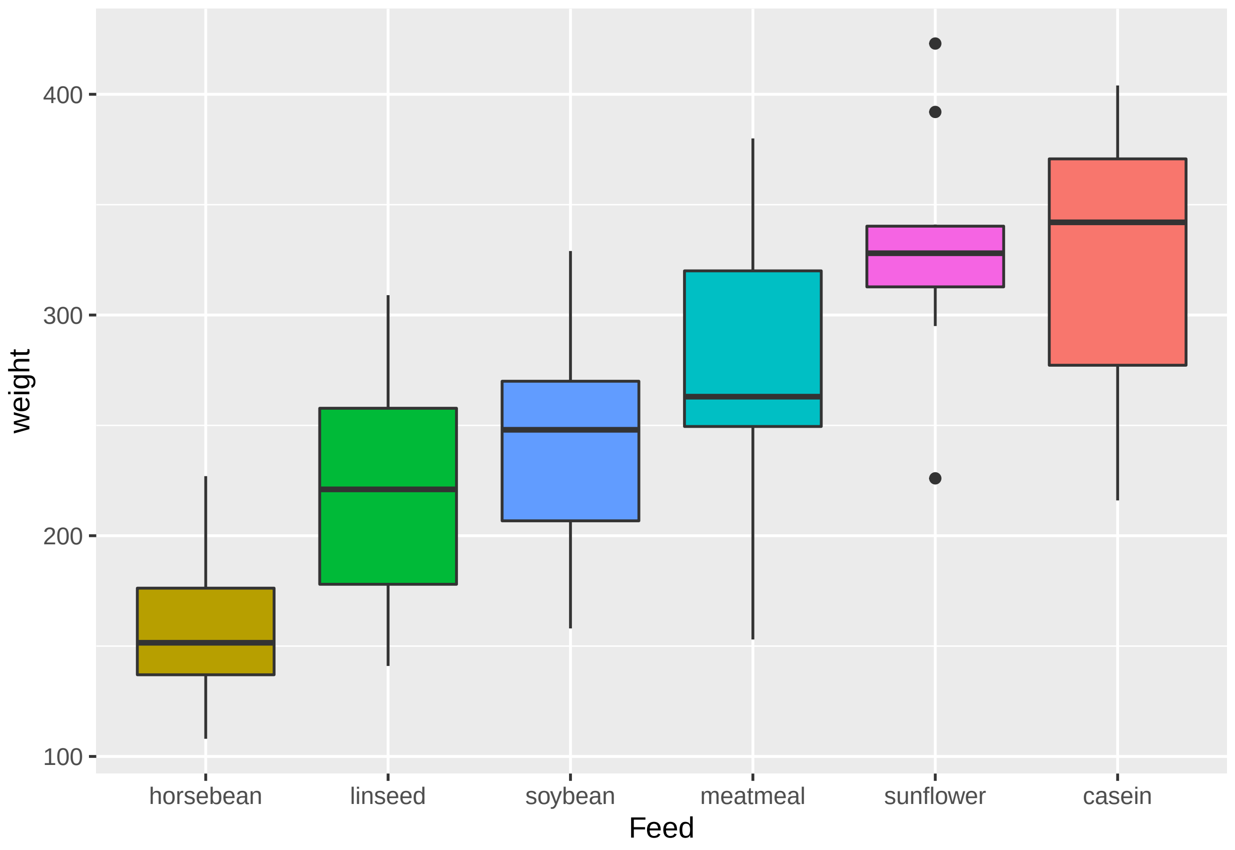
One-way ANOVA

One-way ANOVA is really just a linear model and so we still use the `lm()` function first and then the `anova()` command.

Lets get some data:

```
data("chickwts")
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```



First we fit a linear model:

```
chickwts.lm  <- lm(weight ~ feed, data = chickwts)
```

Perform a one-way ANOVA:

```
anova(chickwts.lm)
```

```
## Analysis of Variance Table
##
## Response: weight
##           Df Sum Sq Mean Sq F value    Pr(>F)
## feed       5 231129   46226  15.365 5.936e-10 ***
## Residuals 65 195556    3009
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Finally, we can find where the differences are:

```
pairwise.t.test(x = chickwts$weight, g = chickwts$feed, p.adjust.method = "bonferroni")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  chickwts$weight and chickwts$feed
##
##           casein  horsebean linseed meatmeal soybean
## horsebean 3.1e-08 -         -       -        -
## linseed   0.00022 0.22833   -       -        -
## meatmeal  0.68350 0.00011   0.20218 -        -
## soybean   0.00998 0.00487   1.00000 1.00000  -
## sunflower 1.00000 1.2e-08   9.3e-05 0.39653  0.00447
##
## P value adjustment method: bonferroni
```

Videos

Want more? Of course you do! Visit the following link to see videos by Dr Jono Tuke:

https://www.youtube.com/playlist?list=PLWqU-yL9fgWbRUM8biwsTxaUQ7ShIDkWy