Lab 3

Univariate Statistics with R

Introduction

In this lab, you are going to begin to become accustomed to randomly generating data of different types in R, generating appropriate plots for different types of data, reading in real data and checking data sets, summarising variables, and plotting real data. The last sections of this lab are intended to be completed as homework, however if you get it all done in the 2 hours - good for you. So let's get started.

Before we begin

Set up a project for today's lab.

In Rstudio, go to the top-right menu and create a new project in an empty folder called "Lab2" (see last week's lab for more information). Unless otherwise stated, you are advised to create a new R script and type/edit commands for this session into the Editor, sending them to the Console using Ctrl-Return (or Cmd-Return on a Mac).

Simple Plots

Binary & Nominal Multiple Category Variables

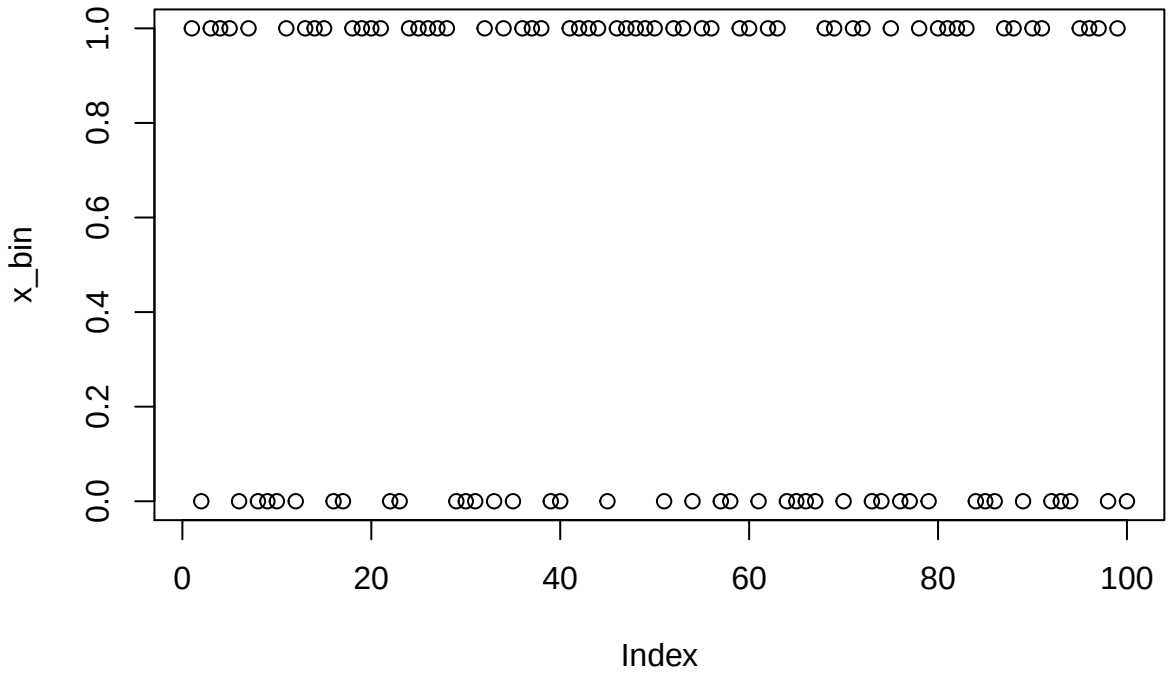
Let's start with the visualization of binary variables. First, let's randomly generate a single binary variable.

```
# Randomly Generate a Single Binary Variable
x_bin <- rbinom(100, 1, .6)
```

Take a look at `?rbinom` and try to work out what frequency of 1s we might expect to see in `x_bin`? We will come back to this shortly.

Now let's look to make a plot of this variable.

```
plot(x_bin)
```



What has gone wrong here?

`plot()` is generally quite a clever function in that it will produce appropriate plots for the type of data we give it. But we need to make sure the variables we give `plot()` are of the type we would like. Let's check whether the variable we created is being treated as a factor (R language for categorical variables):

```
is.factor(x_bin) #Asking R whether x_bin is being treated as a factor.
```

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

So R is not recognising our variable as a categorical variable. What is R treating the `xbin` as;

```
class(x_bin)
```

```
## [1] "integer"
```

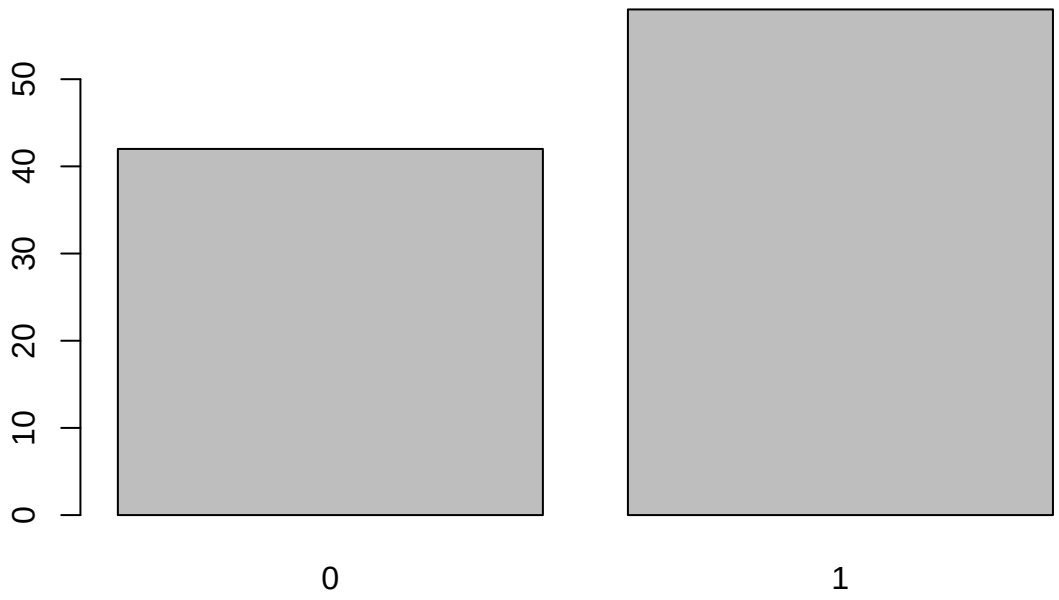
We want this to be a factor (nominal categories), so We can change the status of this variable as follows:

```
x_bin <- as.factor(x_bin)
is.factor(x_bin)
```

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

Excellent! R now recognises the variable as a categorical variable. Now let's see what `plot()` does:

```
plot(x_bin)
```



`plot()` has now given us a very basic frequency histogram of 0s and 1s, for the variable `x_bin`.

Let's briefly return to the question above - what is the frequency of 1s in `x_bin`? From looking at the plot above, we can see there are more 1s than 0s, but as it stands the plot is not much use to us in seeing the exact frequency (we will deal with editing plots shortly). Let's ask R to give us the actual frequencies of 0s and 1s:

```
table(x_bin) # this produces a frequency table for our variable
```

```
## x_bin
##  0  1
## 42 58
```

Time to Think: So, thinking back to how we specified the creation of our randon variable (`rbinom(100, 1, .6)`), why is it we do not have 60 1s? Also, why is it that you may have a different number of 1s to the person sat next to you?

Hopefully you can see that this is your first taste of probability theory and random variables (although you may not have quite phrased it that way), which we will discuss lots more next week. For now, lets continue looking at plots. Note from this point on your plots will look different to mine (and your neighbours) because you have different random variables.

Now let's create a second binary variable and create a mini data set with both variables:

```
y_bin <- rbinom(100, 1, .75)
# look at the frequencies of 0 and 1
table(y_bin)
```

```
## y_bin
##  0  1
## 32 68
```

```r
# tell R y_bin is a binary categorical variable
y_bin <- as.factor(y_bin)
# cbind() combines two columns into a single object, here called data
data <- cbind(x_bin, y_bin)
# shows the column and row names and the first few lines of data
head(data)
```

```
##      x_bin y_bin
## [1,]     2     1
## [2,]     1     1
## [3,]     2     2
## [4,]     2     1
## [5,]     2     1
## [6,]     1     1
```

Time to Think: Look at the first few lines carefully. Why is R showing the values of `x_bin` and `y_bin` to be 1 and 2 instead of 0 and 1? More importantly, if we assume this is a nominal variable, does this matter?

Looking at our data also provides us with a chance to talk briefly indexing. Note in the left column we have `[1, ]` for the first row. R uses the `[ ]` when indexing rows, columns and individual cells in a data frame or matrix. The numbers appear in the order [row, column]. So;

- `[1, ]` = the first row
- `[, 1]` = the first column
- `[1, 1]` = the value in the first row, first column

Indexing is a very useful tool. We can use it to highlight individual elements, perform tasks on single rows or columns, subset data etc. Indexing will appear in lots of the code that follows. You can use the basic rules here to help you "read R". As an example:

```r
data[3, 2] #calls the value in the third row, second column of our data set
```

```
## y_bin
##     2
```

```r
head(data) # which we can check by looking the top few rows of data
```

```
##      x_bin y_bin
## [1,]     2     1
## [2,]     1     1
## [3,]     2     2
## [4,]     2     1
## [5,]     2     1
## [6,]     1     1
```

You can play around with this a little if you are not yet convinced. For example, we could use indexing to give us the total number of 1s in the first column;

```r
sum(data[, 1]==1)
```

```
## [1] 42
```

We can then check this is correct using the `table()` function again.

```r
table(data[, 1]) #then we can check the answer using table()
```

```
##
##  1  2
## 42 58
```

Time to Think: OK, so this one is tricky, but have a go at describing what the code line `sum(data[, 1]==1)` is doing. Although in principle this is quite simple, how R does something of this sort takes a bit of getting used to. **HINT**: Look up logical vectors.

One last thing before we get back to plots. I want to give our variables names so we can pretend we have collected real data (instead of just simulating). So, here we use `colnames()` to change the names of the variables. We create a vector of new names, here "Sex"" and "Over 18"", and then we assign this to data;

```
colnames(data) <- c("Sex", "Over18")
head(data) # we can then check the names have changed
```

```
##      Sex Over18
## [1,]   2      1
## [2,]   1      1
## [3,]   2      2
## [4,]   2      1
## [5,]   2      1
## [6,]   1      1
```

This is good. But now we need our numerical labels to mean something consistent with the variable names. So we need to assign verbal labels to the numerical values.

```
# To manipulate our data, we first make it a data frame
data <- as.data.frame(data)

# Now we make sure our variable is recognised as a factor
data$Sex <- as.factor(data$Sex)
is.factor(data$Sex)
```

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

OK, all looking good so far. Remember you looked up logical vectors earlier, well you should recognise that when we use is.factor (asking R a direct question of whether the variable is a factor or not) it returns a logical vector.

Before we assign verbal labels to the numerical values, we should first check how many levels (in R speak levels are the number of unique numerical labels) we have. So;

```
levels(data$Sex)
```

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

And we can see we have 2 levels with numerical labels 1 & 2. So now we assign new informative verbal labels:

```
levels(data$Sex) <- c("Male", "Female")
```

All done. Now let's do the same for the 'Over 18' variable, but this time using indexing in our code;

```
data[, 2] <- as.factor(data[, 2])
is.factor(data[, 2])
```

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

```
levels(data[, 2])
```

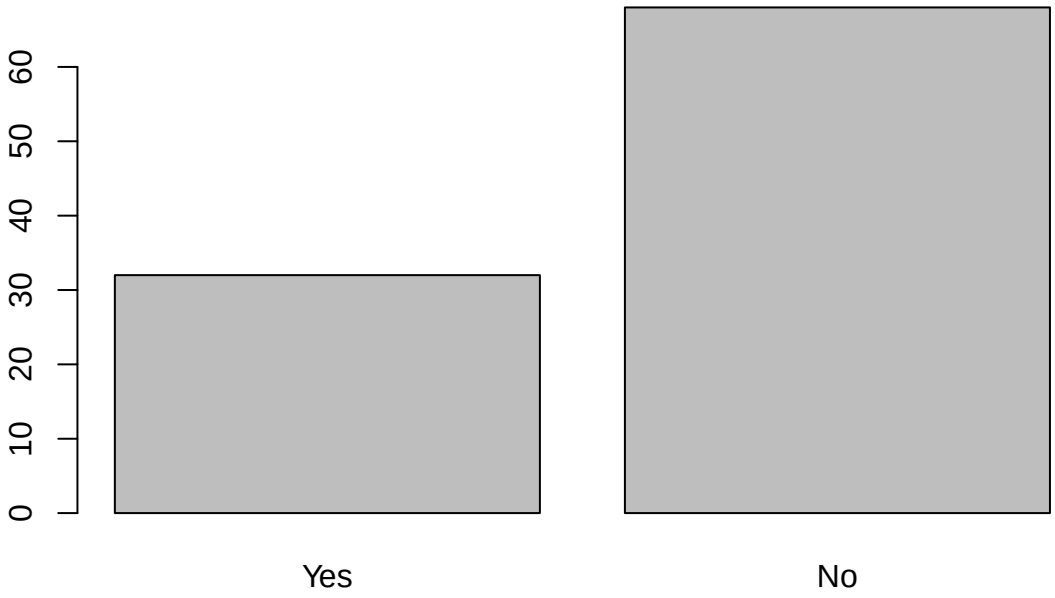
```
## [1] "1" "2"
```

```
levels(data[, 2]) <- c("Yes", "No")
```

Now (finally), back to plotting. Let's look at the histograms of the frequencies for both variables:

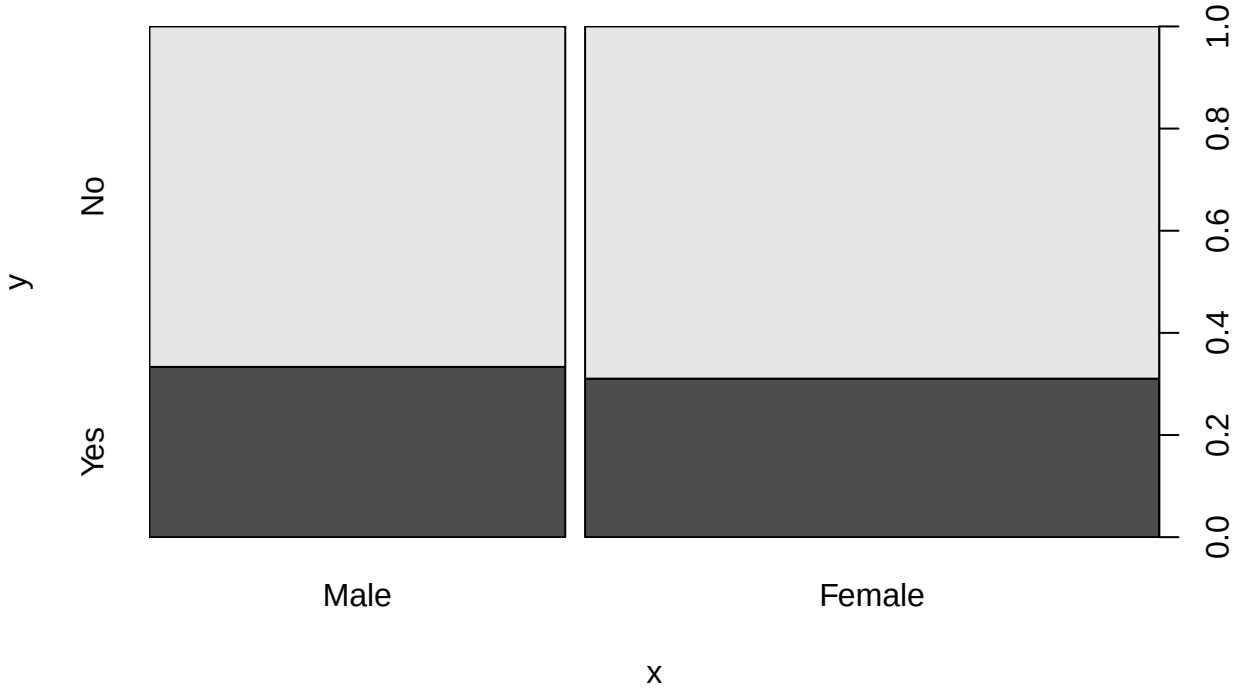
```r
plot(data$Sex)
```


```r
plot(data[, 2])
```



Now, if we want to consider the ratio of men and women who are over 18, we can compute a mosaic plot. To do this we simply give R two categorical variables:

```
plot(data$Sex, data[, 2])
```



Time to Think: What do you think would happen if we used `data[, 1]` instead of `data$sex`? What can we conclude from this plot? Are there proportionally more males than females 18+? Again, why does your plot look different to mine and your neighbour?

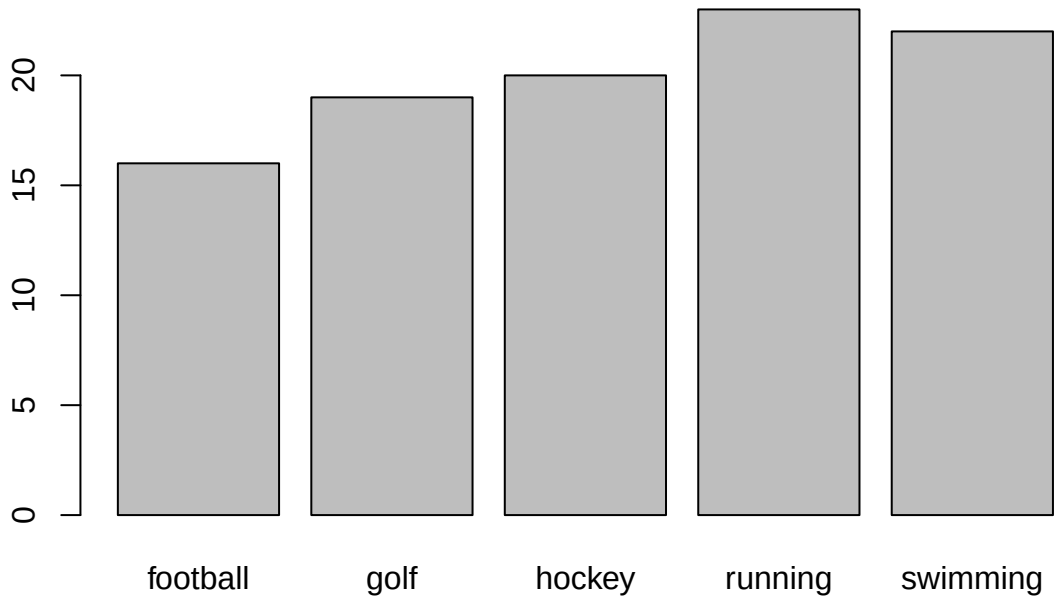
We can extend all the principles above to considering multiple category nominal variables. Let's create add a variable to our data called Primary Sport. Let's suppose our sample of 100 people play five sports (football, running, hockey, golf & swimming).

So let's create this variable. Here is one way to do it:

```
# First we create a list of the names of our sports
choose <- c("football", "running", "hockey", "golf", "swimming")

# next we create a variable by sampling from this list 100 times with replacement
sport <- sample(choose, 100, replace = T)

# sport is now a vector 1x100 with randomly selected sports.
# We can then add this to our data.
data <- cbind(data, sport)
```

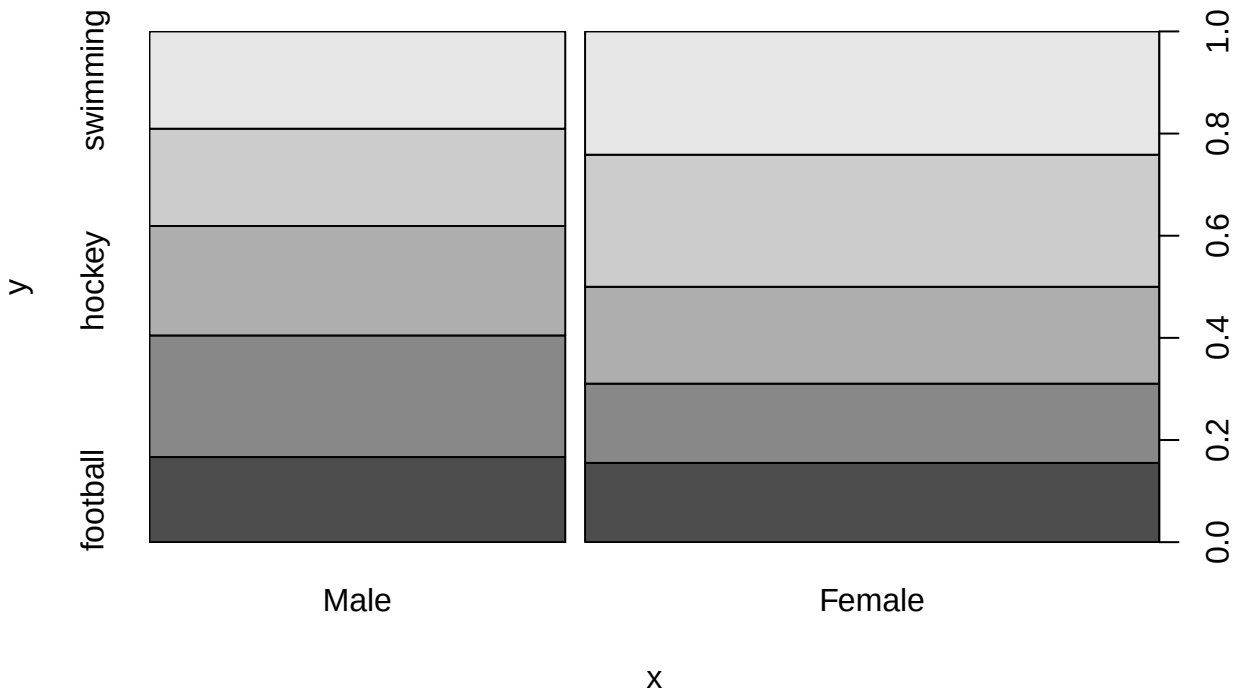
We can then create the histogram for this `sport`:

```
plot(data[, 3])
```



And compute mosaic plots for the favourite sport amongst men and women:

```
plot(data[, 1], data[, 3])
```



Time to think: Look at the help for `?sample`. Look at the third argument, `prob` in the help. We will get on to this next week, but inutively, what do you think probability weights are? Can you work out from the help what the default settings for the weights are? How do you think the mosaic plot may change if we changed these weights? If you want to play with the weights you can. Add the argument `prob = c()` to the code for creating the sport variable. Inside the parentheses, list 5 numbers which sum to 1. For example: `prob = c(.1, .5, .2, .1, .1)`.

Note: You could try to make these meaningful by estimating the actual popularities of these sports.

IMPORTANT

With all of the plots in this section, it really doesn't matter what order we plot the variables in as they are nominal variables. There is no order of presentation for gender which makes more logical sense than any other. However, the same is not true for ordered categorical variables.

Ordinal Categorical Variables

For ordered categorical data, our primary tool for visualization is the same as for the binary and nominal variables - the histogram. However, here, order matters for reasons that should be clear from the lecture. Unlike nominal categorical variables, order does matter and we want to preserve this in our visualizations.

Remember, with an ordered categorical variable we often assume that the data we observe come from a truly normally distributed variable, but that our measurements are coarsely categorized. For the purpose of our generated data, let's create a variable, `fitness`, that is individuals self-rated fitness level on a scale of 1-8:

```
norm <- rnorm(100, 0, 1) #start by creating a random normal variable
initial_fitness <- cut(norm, 8)
initial_fitness <- as.numeric(initial_fitness)
```

Here we first cut our normally distributed variable into 8 categories, and then told R that this new variable `initial_fitness` was numeric. However, in R, we can easily put functions within other functions, so we could have written the same thing as;

```
initial_fitness <- as.numeric(cut(norm, 8))
```

Look out for this as we go through the practical.

In the second line, we have used the cut function to create 8 equally sized groups from our normally distributed variable. These represent our coarse divisions of an underlying normal variable. We use `as.numeric` to tell R we want to treat this variable as if it were numeric, not as a factor.

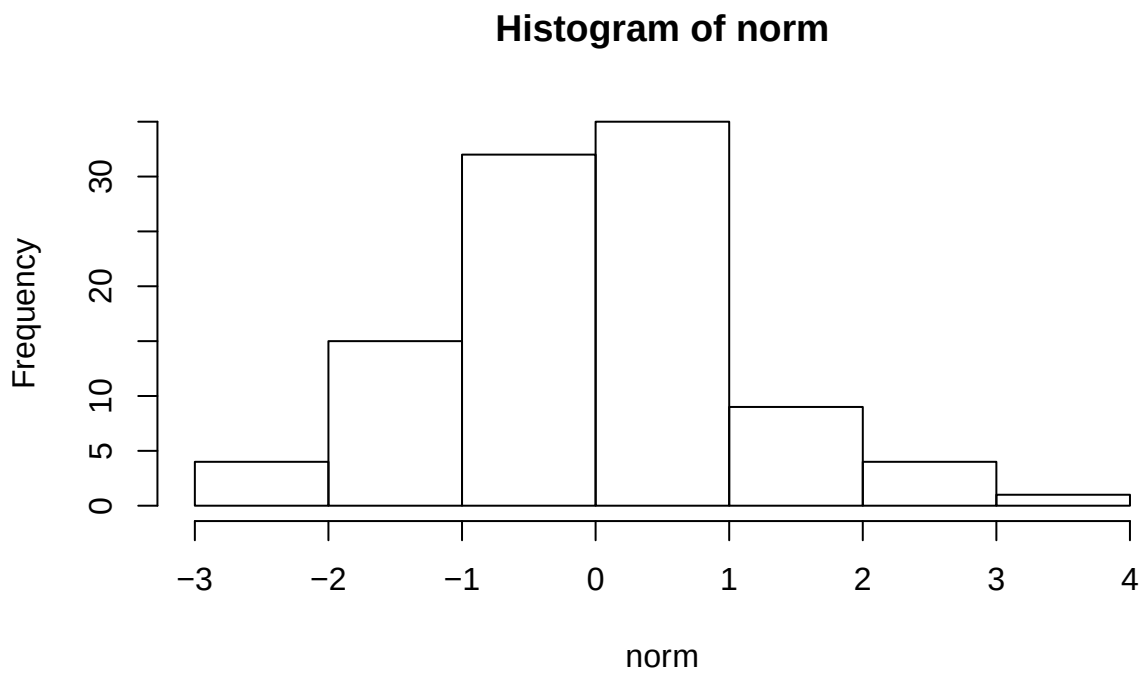
Remember: We discussed in class that there is much debate about whether coarsely categorized variables should be treated as continuous or not. This is an important decision you will have to make in your own analyses and you will need to make sure you have told R what type of variable each variable in your data set is. As an aside, if you want to check what R thinks each variable is, you can use:

```
class(initial_fitness)
```

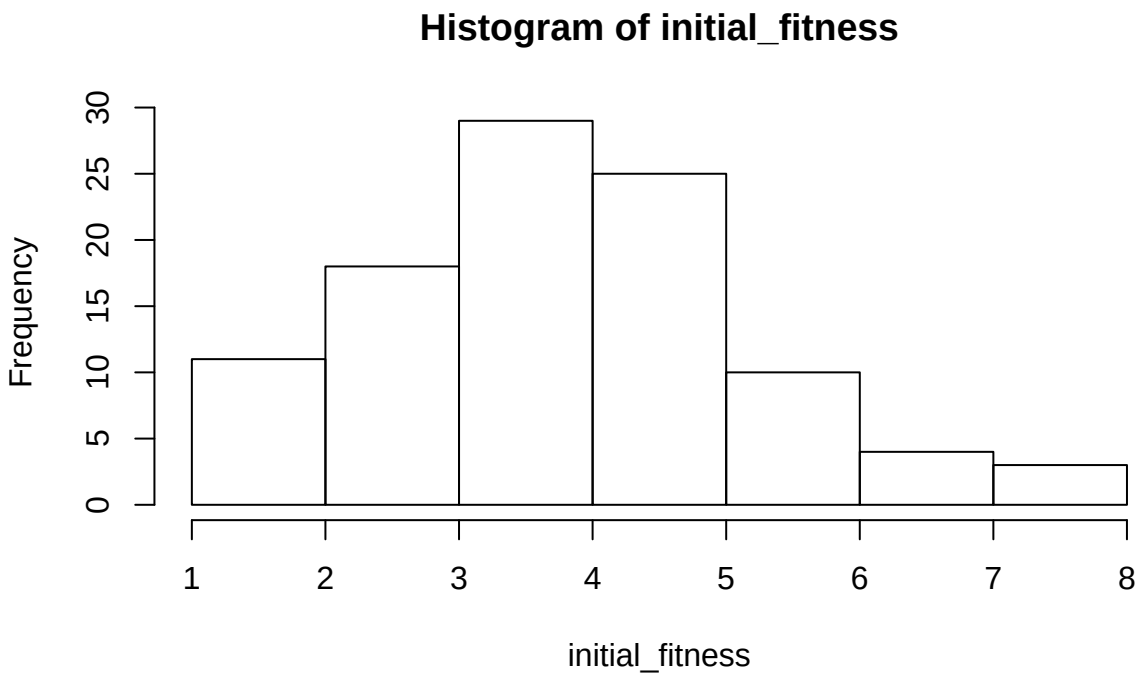
```
## [1] "numeric"
```

We can see how the distribution of our variables degrades when we coarsely categorize `norm` by using histograms;

```
hist(norm)
```




```
hist(initial_fitness)
```



Time to Think: With 8 categories, this does not look too bad, but what would it look like if we used say 4 categories? If you want to check this out, create a new variable with a different name, and use the `cut()` function and the continuous variable `norm` to categorize it. Then plot it.

Now let's attach the new variable to the data set. We could just attach it as it is, but to show you another method to manipulate your data, let's jumble up the order a bit;

```
fitness <- sample(initial_fitness, 100, replace = F)
```

Time to Think: So you have seen `sample()` used before (above). Have a go at describing what we are doing with sample here. **HINT**: use `?sample` and focus on size we have requested (given sample size), and what it might mean to not replace values. (again dont worry - this is a hard question and we will go over it properly next week - but give it a go).

We can then compare the first few lines to see we have jumbled it up, and the cell frequencies to show this is all we have done;

```
head(initial_fitness)
```

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

```
head(fitness)
```

```
## [1] 3 1 6 4 2 4
```

```
table(initial_fitness)
```

```
## initial_fitness
##  1  2  3  4  5  6  7  8
##  3  8 18 29 25 10  4  3
```

```
table(fitness)
```

```
## fitness
```

```
##  1  2  3  4  5  6  7  8
##  3  8 18 29 25 10  4  3
```

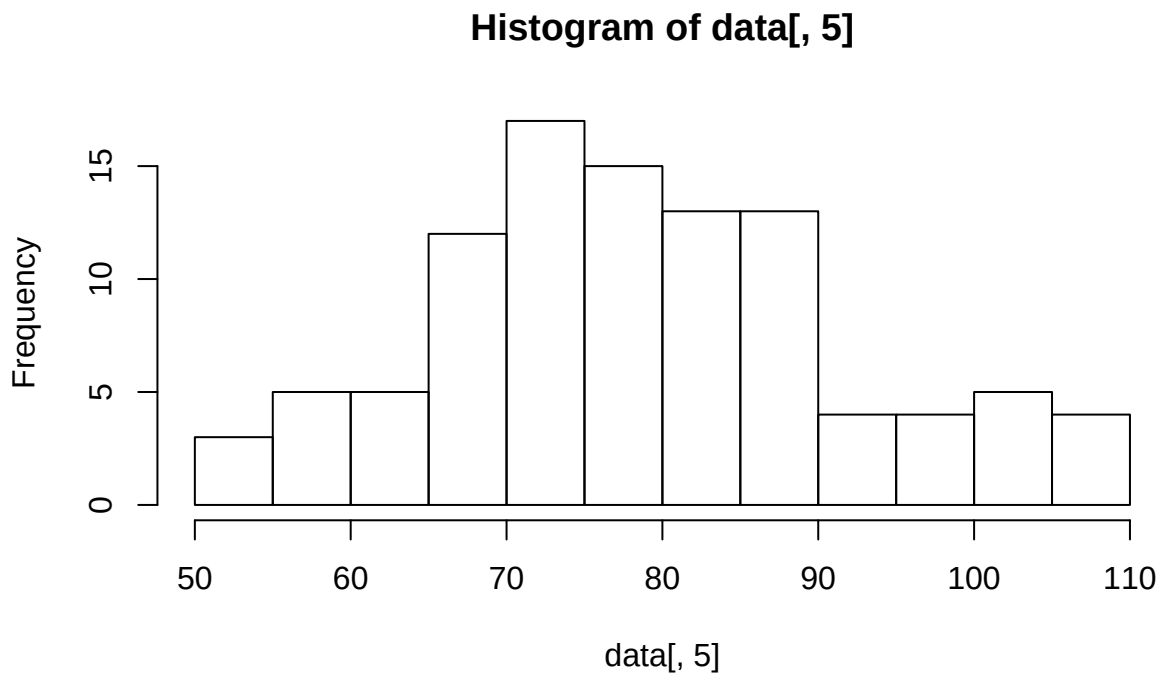
So now let's bind it to our other data;

```
data <- cbind(data, fitness) #add fitness
colnames(data)[4] <- "Fitness" #rename the 4th column in data "Fitness"
```

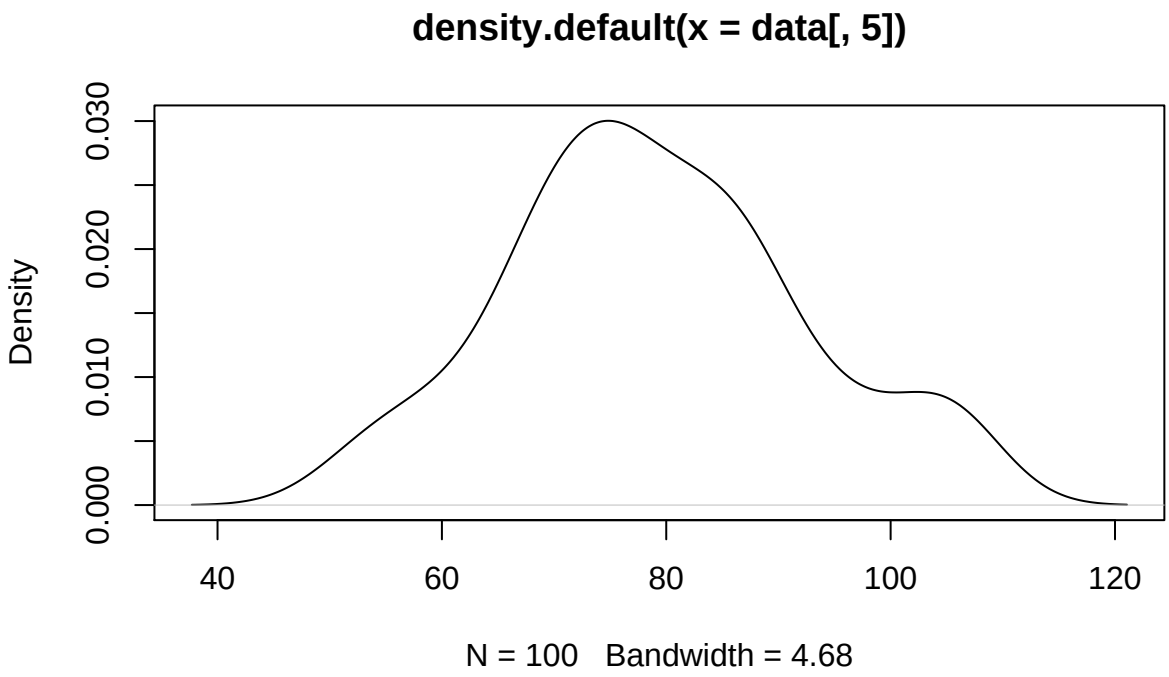
Continuous Variables

In some cases our variables will truly be continuously measured on a ratio scale. When this is the case, we can visualize the distribution of the variable in a number of ways. We can use a histogram, but in doing so we need to decide how many "bins" to use. Alternatively, we can use a kernel density plot. We have gone over a number of examples of this slowly, so let's do this in one quick hit of code. Let's create a normally distributed variable for weight (kg), attach it and plot it;

```
a <- rnorm(10000, 81, 12)
weight <- sample(a, 100, replace = F)
data <- cbind(data, weight)
hist(data[, 5])
```




```
plot(density(data[, 5]))
```



Time to Think: As a test, can you verbally state what each of the lines of code are doing when we make the variable weight and plot it?

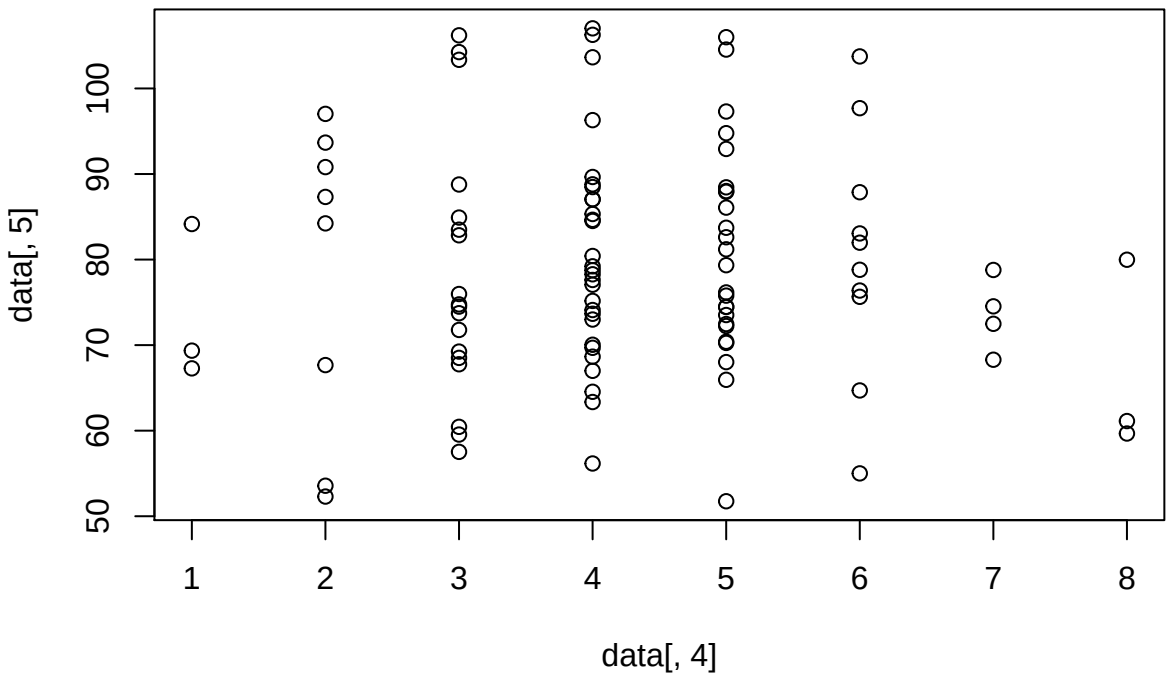
HINT: Use the `?` help options to look at the different arguments and look back in this lab.

Scatterplots: Visualizing associations

Sometimes we might want to use plots to look at the relations between variables. We will cover lots of different types of plots to do this as the course progresses (particularly plots which help check assumptions in GLM). As a simple case, consider looking at the relationship between two continuous variables. We already have two variables we are treating as continuous in our data, Fitness and Weight.

This time, instead of giving plot a single variable, we give it two. Again, it is smart enough to know what to do with these data:

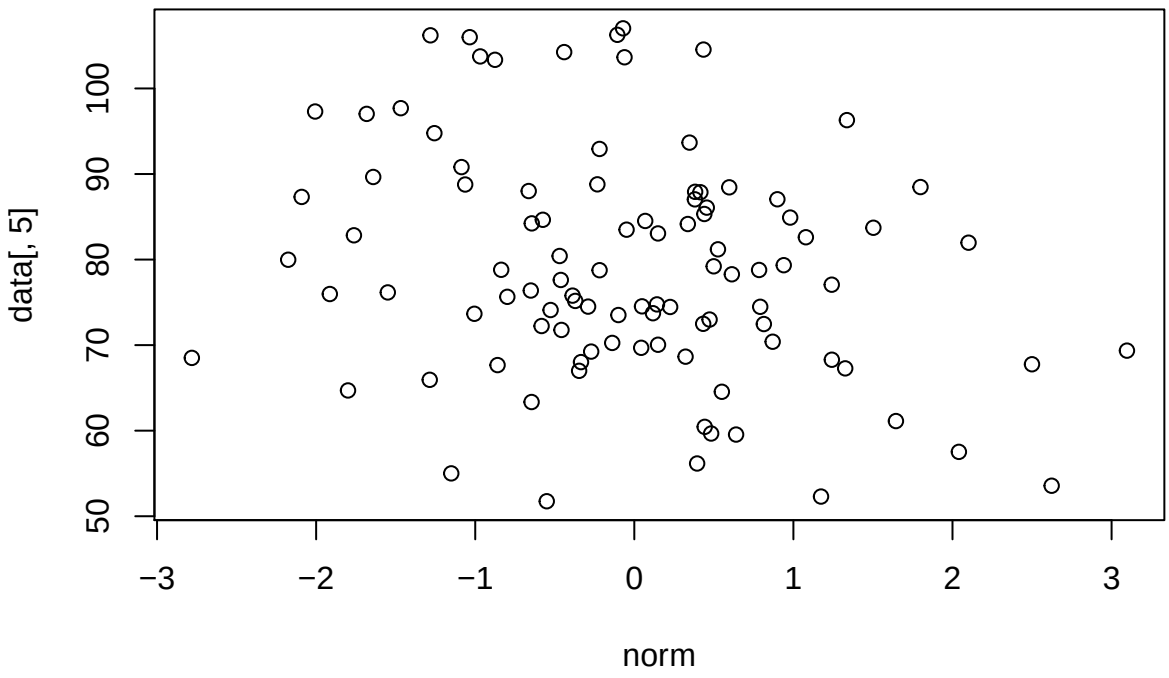
```
plot(data[, 4], data[, 5])
```



Time to Think: Although we are treating Fitness as continuous, this scatterplot looks odd. Why do you think this is?

Instead of using the categorized variable fitness, why dont we use the `norm` instead. Though not in your data set, this is still in your R environment (look down the values list in the top right).

```
plot(norm, data[, 5])
```



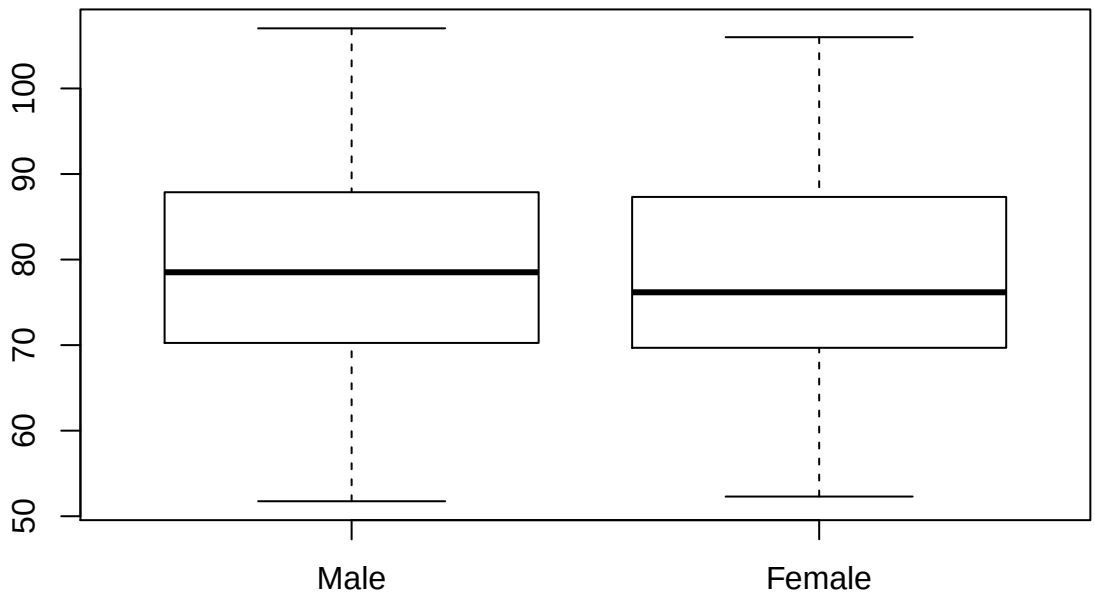
Boxplots: Visualizing mean differences

So, the scatterplot gives us the association between two variables, but what about comparing the mean score on a continuous variable given a categorical variable. This type of plot is very useful if you are conducting a t-test (see week 4). You have in fact already done this in the first lab, so this code should seem familiar and we wont linger on it:

```
# I am using this here to remind myself how each of the variables is named in data
names(data)
```

```
## [1] "Sex"     "Over18"  "sport"   "Fitness" "weight"
```

```
boxplot(weight~Sex, data = data)
```



Descriptive Statistics

As well as the basics of plotting, it is also useful to be able to produce basic descriptive statistics for the variables in your data set. So let's look at the options for this. My preference for descriptive statistics is to use the `describe()` function in the `psych` package. This is simple to use;

```
library(psych)
describe(data)
```

```
##         vars   n  mean    sd median trimmed   mad   min    max range  skew
## Sex*       1 100  1.58  0.50   2.00    1.60  0.00  1.00   2.00  1.00 -0.32
## Over18*    2 100  1.68  0.47   2.00    1.73  0.00  1.00   2.00  1.00 -0.76
## sport*     3 100  3.16  1.39   3.00    3.20  1.48  1.00   5.00  4.00 -0.15
## Fitness    4 100  4.26  1.49   4.00    4.24  1.48  1.00   8.00  7.00  0.21
## weight     5 100 79.08 13.25  77.95   78.73 13.19 51.75 107.03 55.27  0.22
##         kurtosis   se
## Sex*       -1.92 0.05
## Over18*    -1.44 0.05
## sport*     -1.27 0.14
## Fitness     0.07 0.15
## weight     -0.40 1.32
```

However, as you can see, `psych` reports the arthemtic mean (column 3) for all variables, even those which are factors. We can tell which `psych` and R are treating as factors as a * appears after the name of the variable on the left.

`psych` does give us the median, minumum and maximum values, range, skew and kurtosis - so it is doing

pretty well in describing our data. But we may want a couple of other things.

```
# as we have seen, this gives frequencies
table(data$Sex)
```

```
##
##   Male Female
##     42     58
```

```
# we can use quantile to give us the values at a given percentile of our variable
# here we select the 25th and 75th
quantile(data$weight, c(.25, .75))
```

```
##      25%      75%
## 69.95359 87.45793
```

```
# we could also use this to get the inter-quartile range
lower <- as.numeric(quantile(data$weight, c(.25)))
upper <- as.numeric(quantile(data$weight, c(.75)))
IQrange <- upper-lower
IQrange
```

```
## [1] 17.50434
```

```
# we can also get the mode. For multiple category variables,
# we just want the largest value from the 2nd row of table()
table(data[, 4])
```

```
##
##  1  2  3  4  5  6  7  8
##  3  8 18 29 25 10  4  3
```

Sometimes we may want to create our own table of results that includes the appropriate statistics given our variables. Let's have a go.

```
# first job, lets create an empty 2x2 matrix
empty <- matrix(nrow = 2, ncol = 2)
colnames(empty) <- c("Mean", "Skew") # and then give the columns names
rownames(empty) <- c("Weight", "Fitness") # and the rows names
empty # and lets take a look
```

```
##         Mean Skew
## Weight    NA   NA
## Fitness   NA   NA
```

Now let's put some values in. First we need to compute the statistics (here I will use `describe()`, save the output as an object, and then use indexing to isolate the values of interest and enter them into our empty table:

```
descriptive <- describe(data)
descriptive # so lets take the mean of weight.
```

```
##         vars   n  mean    sd median trimmed   mad   min    max range  skew
## Sex*       1 100  1.58  0.50   2.00    1.60  0.00  1.00   2.00  1.00 -0.32
## Over18*    2 100  1.68  0.47   2.00    1.73  0.00  1.00   2.00  1.00 -0.76
## sport*     3 100  3.16  1.39   3.00    3.20  1.48  1.00   5.00  4.00 -0.15
## Fitness    4 100  4.26  1.49   4.00    4.24  1.48  1.00   8.00  7.00  0.21
## weight     5 100 79.08 13.25  77.95   78.73 13.19 51.75 107.03 55.27  0.22
##         kurtosis   se
## Sex*       -1.92 0.05
```

```
## Over18*    -1.44 0.05
## sport*     -1.27 0.14
## Fitness     0.07 0.15
## weight     -0.40 1.32
```

```
descriptive[5, 3] # which is in the fifth row, third column.
```

```
## [1] 79.08437
```

```
# now we can add this to our matrix
empty[1, 1] = descriptive[5, 3]
empty
```

```
##             Mean Skew
## Weight  79.08437   NA
## Fitness       NA   NA
```

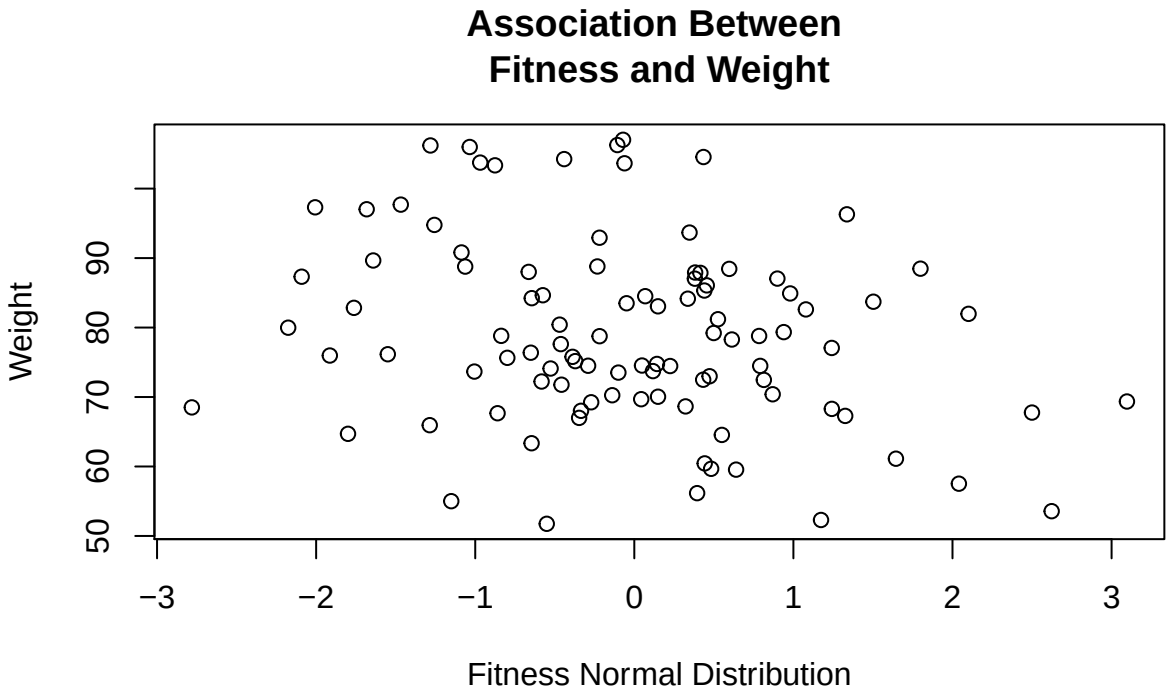
Time to Think: Finish this table by adding the other three values. If you want an extra challenge, edit the code to add a third column for the standard deviation of the mean (sd column in the describe output). Put this column between "Mean" and "Skew"

More advanced plotting

OK, now we have, at some length in both labs 1 and 2, looked at the basic forms of plots in R and some basic skills of manipulating and using objects in R. Now let's start adding a bit of finesse to our plots and some additional features. The examples below are in no way exhaustive of the HUGE variety of things you can do with plots in R, but they should give you a feel for what is possible.

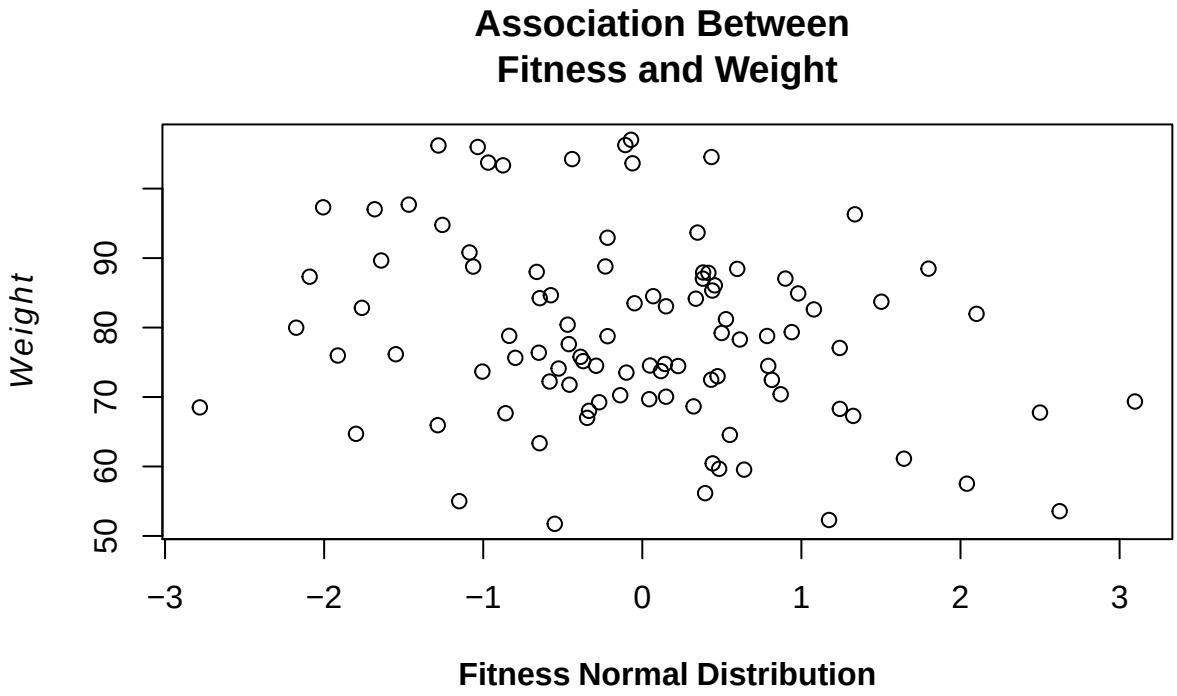
Let's start with our scatterplot from above and add some basic things which you did in lab 1.

```
# Let's start with labels
plot(norm, data[, 5], main = "Association Between \nFitness and Weight",
     xlab = "Fitness Normal Distribution", ylab = "Weight")
```



As something new, we can use the `\n` to split titles across multiple lines. Now how about adding bold and italics to labels? We can do this with `expression`;

```
plot(norm, data[, 5], main = "Association Between \nFitness and Weight",
     xlab = expression(bold(Fitness~Normal~Distribution)),
     ylab = expression(italic(Weight)))
```

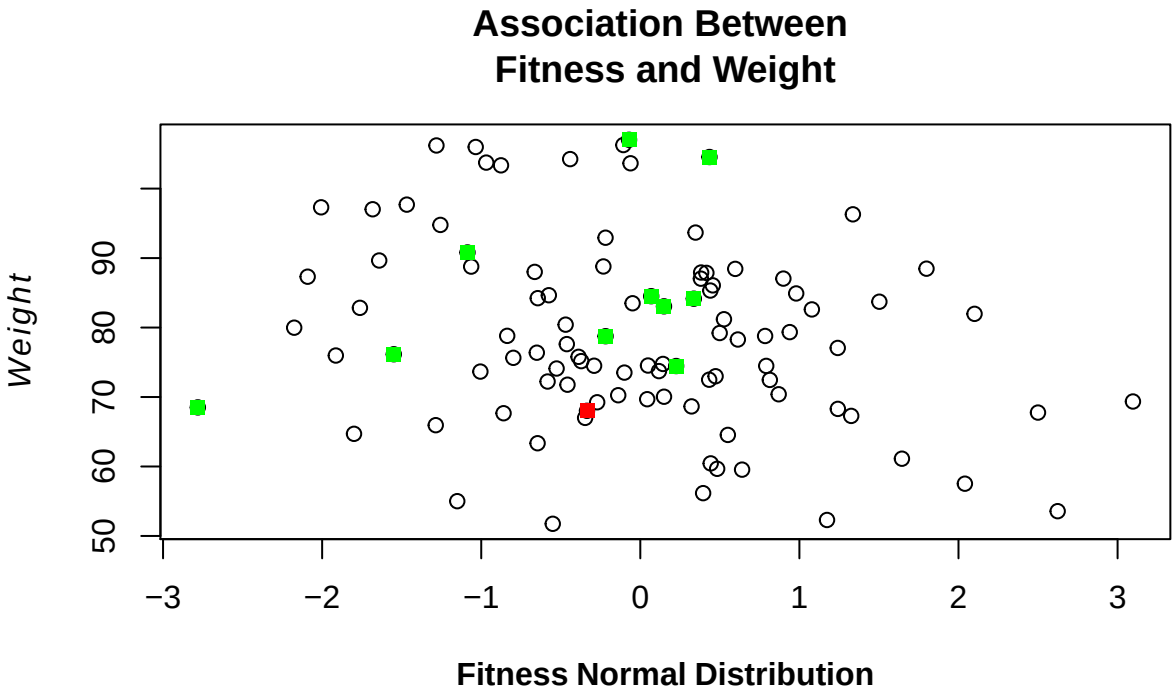


Note, when using `expression()`, we need to use a `~` to conect words. Given the plot above, use what you learnt last week to change the colour, size and symbol used for all the points.

Isolating Points & Different Groups on a Plot

OK, now let's see how we can pick out single points, and include plots for multiple groups. So, to identify a single point (or multiple points), we want to create an object which gives the x and y coordinates of the points;

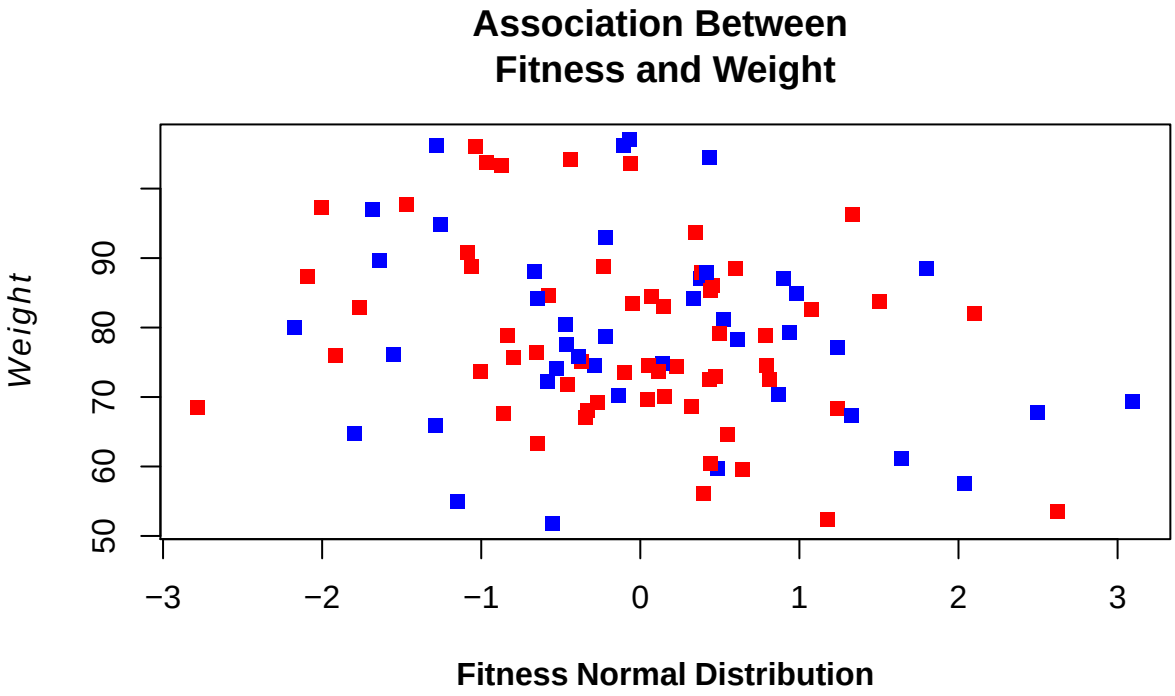
```
y_coord <- data$weight[90] #take 90th element in weight
x_coord <- norm[90] # take 90th element in norm

# create a vector of weight values for elements 1 to 10
y2_coord <- data$weight[1:10]
x2_coord <- norm[1:10] # and the same for norm

plot(norm, data[, 5], main = "Association Between \nFitness and Weight",
     xlab = expression(bold(Fitness~Normal~Distribution)),
     ylab = expression(italic(Weight)))
# we can then use points to adjust particular points give the co-ordinates above
points(x_coord, y_coord, pch = 15, col = "red")
points(x2_coord, y2_coord, pch = 15, col = "green")
```



Time to Think: Using xy.coords can be a little tricky at times. Look up the function `identify()` and see if you can do the same thing.

So that is individual points, now how about 2 groups on the same plot. Let's have the points for men and women coloured differently on our scatterplot.

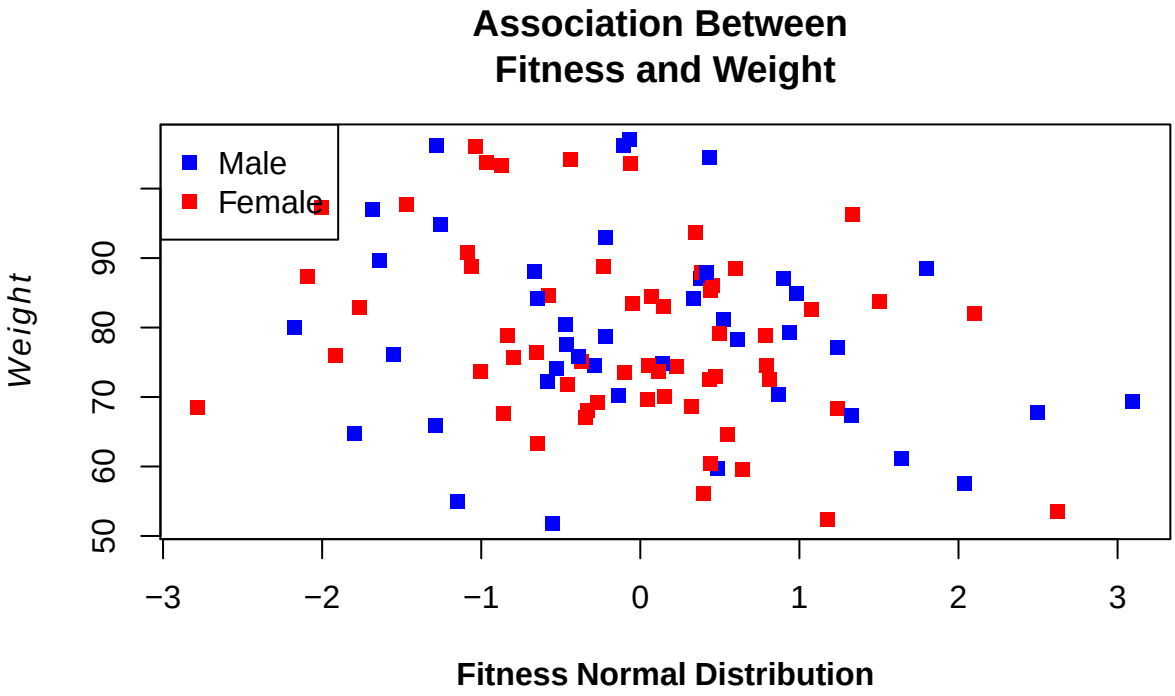
```
plot(norm, data[, 5], main = "Association Between \nFitness and Weight",
     xlab = expression(bold(Fitness~Normal~Distribution)),
     ylab = expression(italic(Weight)),
     pch = 15, col = c("blue", "red")[data$Sex])
```



Here we use the `[data$Sex]` to index the colour vector. Provided the variable we use is a factor, and the length of the `col` vector matches the number of levels, this will work for us.

Now we have different coloured points on the plot, we had better add a legend to explain what they are. So;

```
plot(norm, data[, 5], main = "Association Between \nFitness and Weight",
     xlab = expression(bold(Fitness~Normal~Distribution)),
     ylab = expression(italic(Weight)),
     pch = 15, col = c("blue", "red")[data$Sex])
legend("topleft", legend = c("Male", "Female"),
       col = c("blue", "red"), pch = c(15, 15))
```

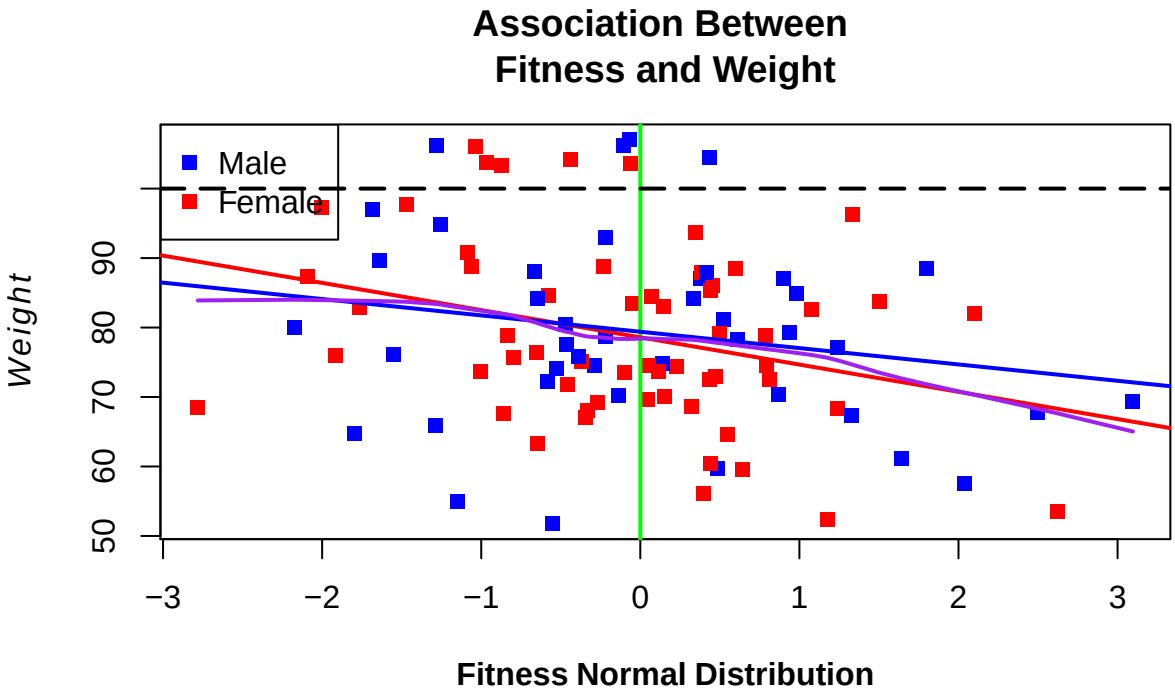


Time to Think: Look at `?legend`, and think through why we have put in all the elements we have. Particularly think about why we have used `c()` to create the different column vectors. As a practice task, change the points symbols and colours in the plot and legend, and move the legend to a different location.

HINT: Look up the help for the function `locator()`. This has multiple uses, much like `identify()`, and may be of use when producing your own plots.

One last thing, lets add some lines to our scatterplot. There are multiple reasons why we may want to add lines. Below I add regression lines for males and females (more about this in the GLM lectures), vertical and horizontal lines, and curved loess fit lines. Clearly, we would never want all these on one plot, but this is simply to show what can be done. So;

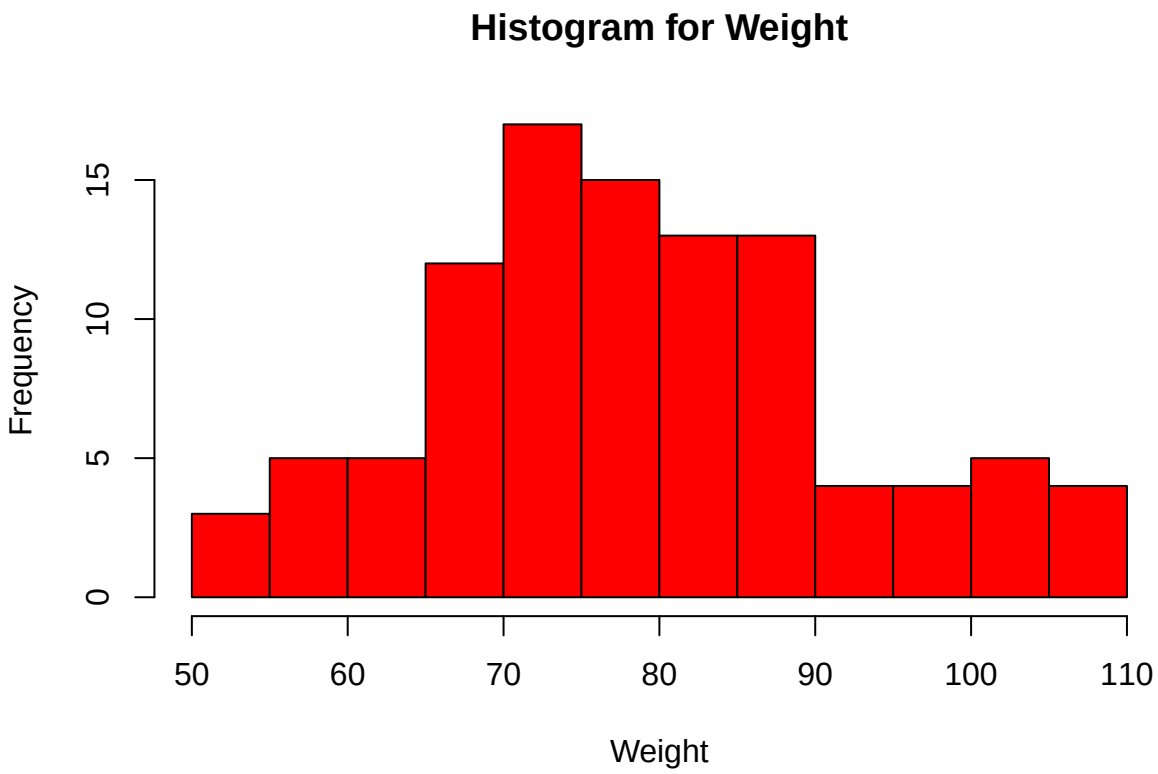
```
plot(norm, data[, 5], main = "Association Between \nFitness and Weight",
     xlab = expression(bold(Fitness~Normal~Distribution)),
     ylab = expression(italic(Weight)),
     pch = 15, col = c("blue", "red")[data$Sex])
legend("topleft", legend = c("Male", "Female"),
       col = c("blue", "red"), pch = c(15, 15))
abline(v = 0, col = "green", lwd = 2) # vertical line at norm = 0
# horizontal line at weight = 100
abline(h = 100, col = "black", lty = 5, lwd = 2)
# regression line women
abline(lm(data[, 5]~norm, subset = data$Sex=="Female"),
       col = "red", lwd = 2)
# regression line men
abline(lm(data[, 5]~norm, subset = data$Sex=="Male"),
       col = "blue", lwd = 2)
# lowess line of best fit
lines(lowess(norm, data[, 5]), col = "purple", lwd = 2)
```



Applying some features to histograms

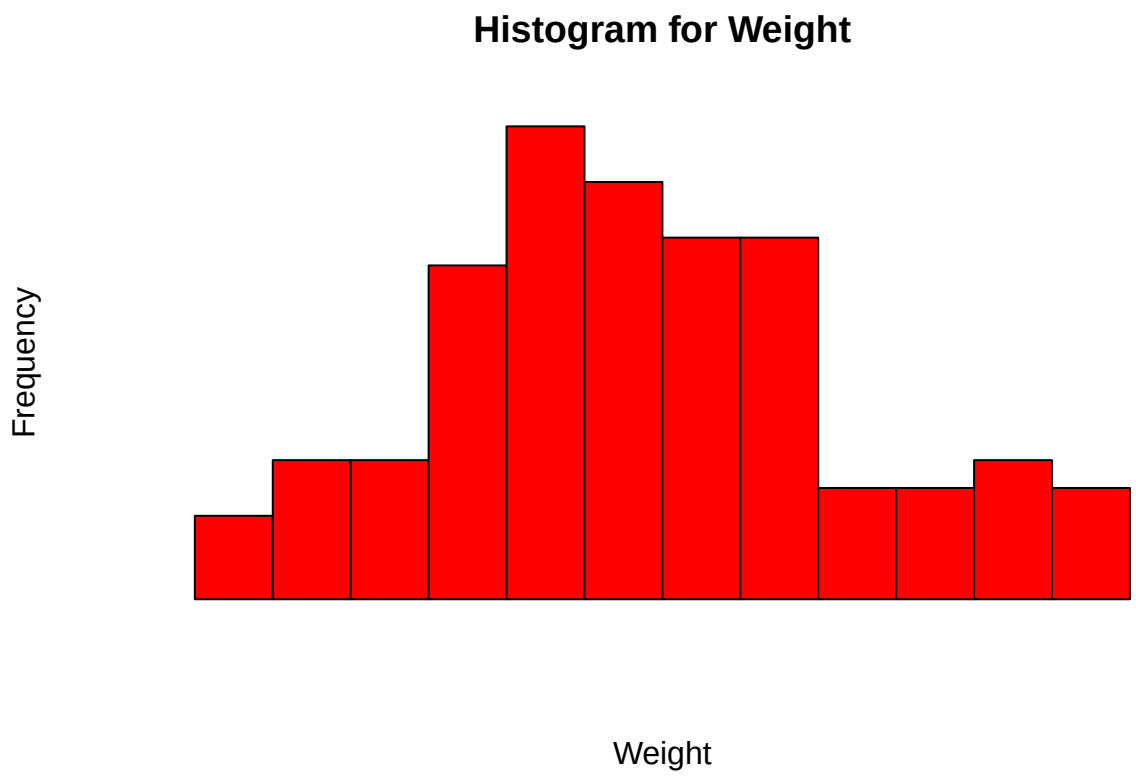
As noted above, many of the plotting features are general across types of plot. One of the last things we should look at is changing the axes on a plot. We will do more of this next week but we can do the basics here with a histogram;

```
hist(data$weight, main = "Histogram for Weight",
     xlab = "Weight", ylab = "Frequency", col = "red")
```



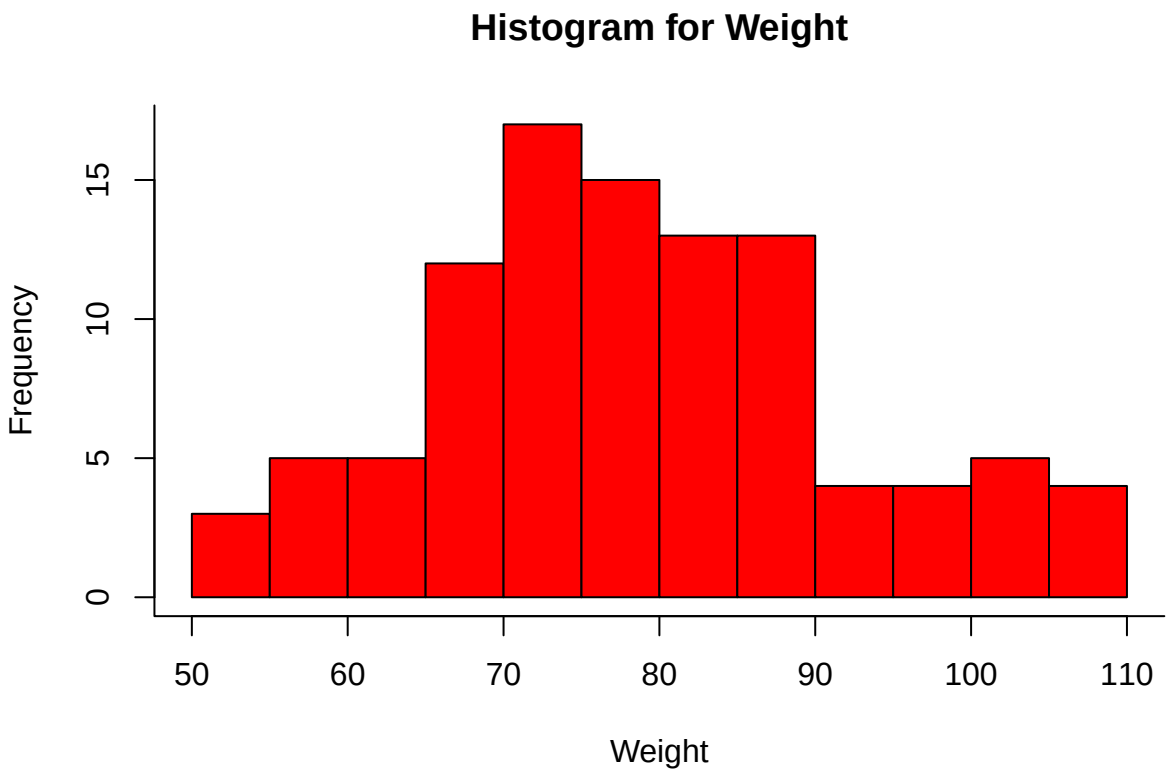
So this plot now has some pretty colours, but our axes don't look very good. The x and y axes dont join for a start. So let's get rid of these and draw them ourself;

```
hist(data$weight, main = "Histogram for Weight",
     xlab = "Weight", ylab = "Frequency", col = "red",
     axes = F) # this line removes the axes from the plot
```



And now we can add some new ones using `axis()` ;

```
hist(data$weight, main = "Histogram for Weight",
     xlab = "Weight", ylab = "Frequency", col = "red",
     axes = F) # this puts an L shaped box (2 sides) onto our plot
box(bty = "L")
# we can use axis for each side to put the default check marks back
axis(side = 1, xpd = T)
axis(side = 2, xpd = T)
```



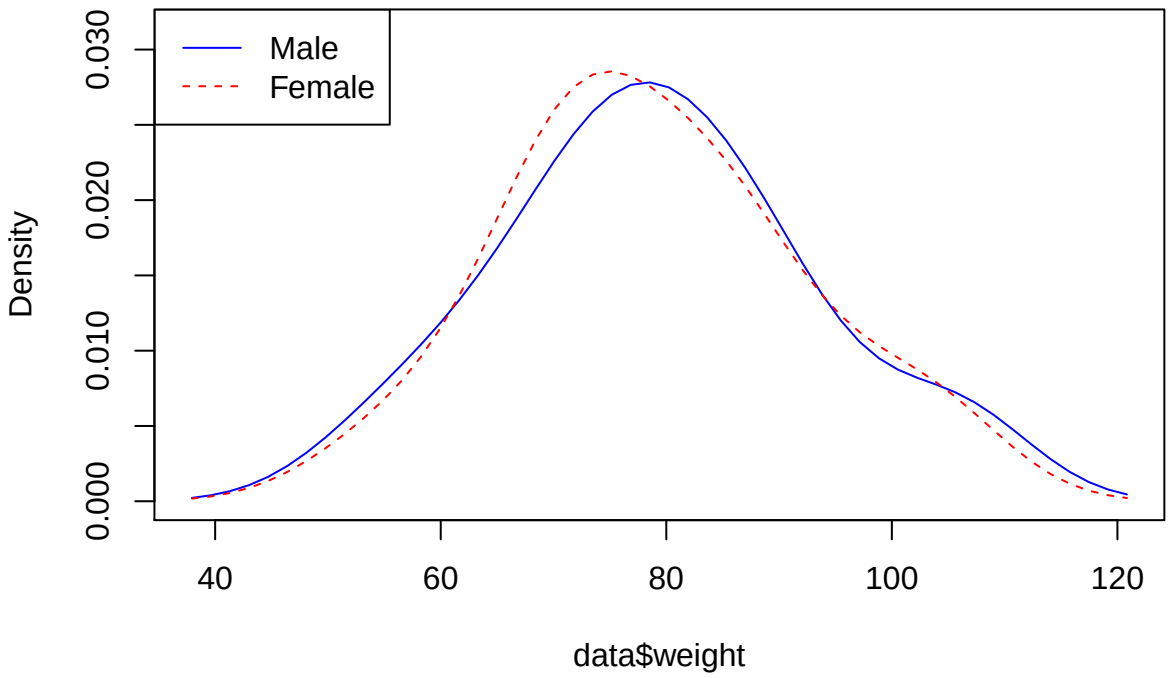
If we want to plot two density plots for two groups on the same plot, there is a useful function to help us (we can use all the normal graphical parameters here:

```
#install.packages("sm")
library(sm)
```

```
## Warning: package 'sm' was built under R version 3.4.2
```

```
## Package 'sm', version 2.2-5.4: type help(sm) for summary information
```

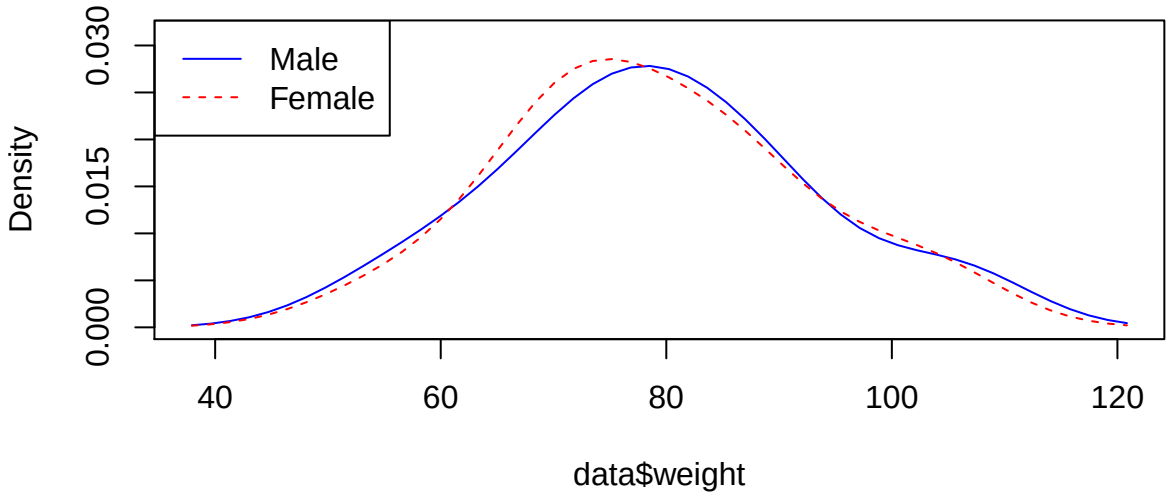
```
sm.density.compare(data$weight, data$Sex, col = c("blue", "red"), lty = c(1, 2))
legend("topleft", levels(data$Sex), lty = c(1, 2), col = c("blue", "red"))
```

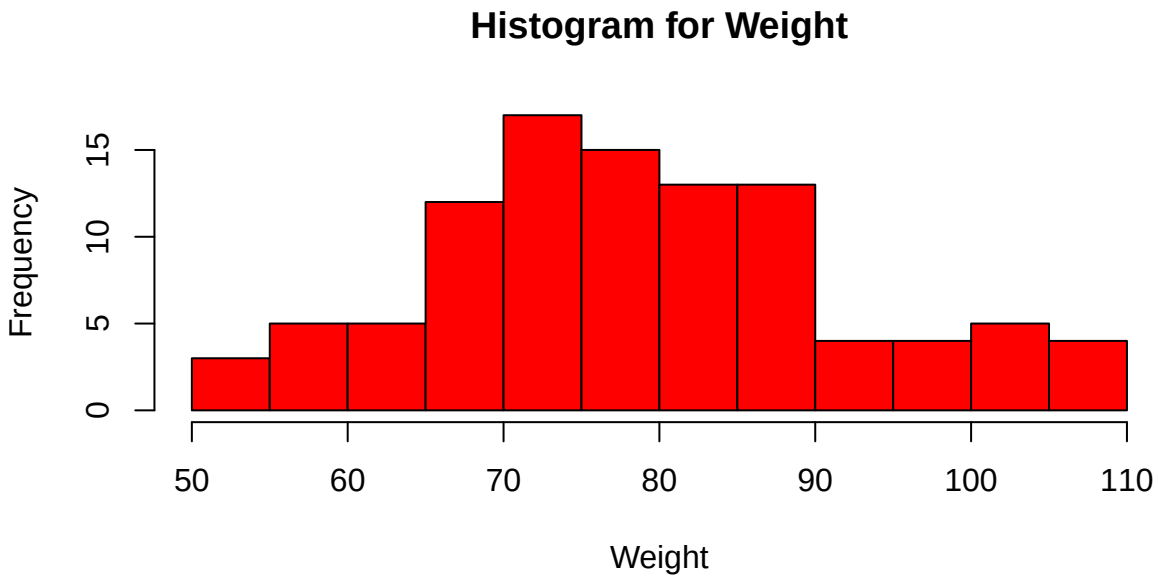


Controlling the Plotting Space: Multiple Plots in 1

So we have now done some fancy things with the basic plots for different types of data. There is one more step that is important to highlight to you with respect to plotting in R. Not only can we have quite fine control over the plots themselves, we can also control the space that the plots are drawn in. The way in which we do this is by adjusting the basic graphical parameters. So let's follow through a simple example of plotting two plots on top of each other. We will use the code developed above for the plots, so all we need to worry about here is adjusting the space.

```
# first we use the mfrow command to tell R we want to divide the plotting space
par(mfrow = c(2, 1))
# then we give it the two plots.
# Automatically the first set of code plots the top plot,
# the second the lower plot
sm.density.compare(data$weight, data$Sex, col = c("blue", "red"), lty = c(1, 2))
legend("topleft", levels(data$Sex), lty = c(1, 2), col = c("blue", "red"))

hist(data$weight, main = "Histogram for Weight",
     xlab = "Weight", ylab = "Frequency", col = "red")
```





Time to Think: From the code given above, create a figure which has four plots in it.

Sometimes we may want more space on one side of our plot. If this is the case, we need to change the margins in the plotting space. We do this within `par()` as well. First though, it can be useful to see what our actual margins are. We can see this by simply typing `par()`. This produces quite a lot of output as it lists all the graphical parameters settings. But we can just look at the margins using;

```
check <- par()
check$mar
```

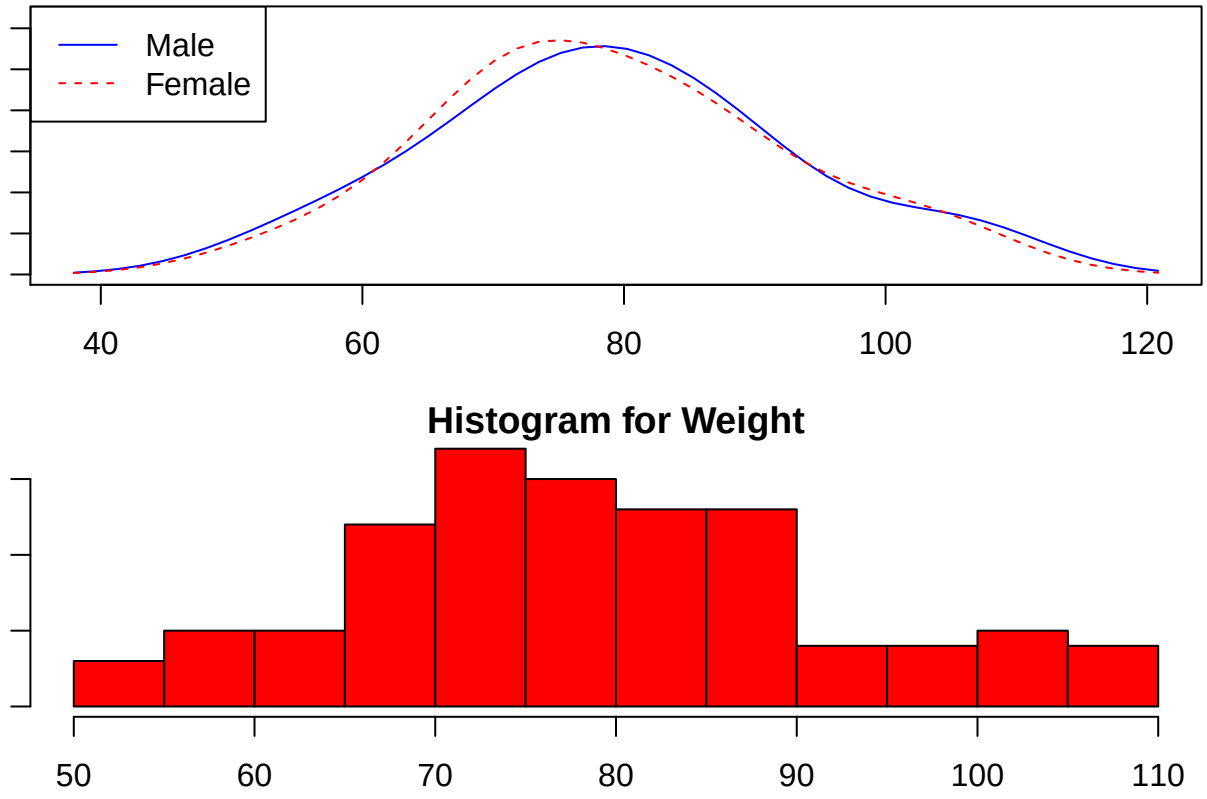
```
## [1] 5.1 4.1 4.1 2.1
```

Here you have four numbers corresponding to the sides of the plot

- 1 = bottom = 5.1
- 2 = left = 4.1
- 3 = top = 4.1
- 4 = right = 2.1

To change the margins we simple add;

```
# first we use the mfrow command to tell R we want to divide the plotting space
par(mfrow = c(2, 1),
## change margins
mar = c(3, 1, 1, 1))

# then provide the two plots
sm.density.compare(data$weight, data$Sex,
                   col = c("blue", "red"), lty = c(1, 2))
legend("topleft", levels(data$Sex), lty = c(1, 2), col = c("blue", "red"))

hist(data$weight, main = "Histogram for Weight",
     xlab = "Weight", ylab = "Frequency", col = "red")
```



You can see that these are not very good settings. We have lost the title on the top plot, and we have lost the axes labels on the left hand side. However, this does shows you the basics of how to adjust the margins. We will want to set out plotting space back to the default. We do this by simply clicking the "Clear All" option on the plotting screen.

That wraps up the tutorial part of the lab for now. We will do more with plotting in the course of future labs, but for now, this will do us.

Let's put all this into practice

Start a new project called something like "Lab 3 Real Data". This will make it easier to reference back to your project code (above) when answering the questions.

In all the examples above, we have been using randomly generated data. Now let's (assume!!) we have got some real data (i.e. you have not seen me simulate it).

So, let's get practicing, describing and visualizing our data. First, read the data into R. Look back at the instructions from week 2 if you can not remember how to do this. The data is a .csv file, so you can also look at `?read.csv`. The data set is saved on LEARN in the Lecture 3 folder.

OK, so now I have a list of tasks. All of these should be achievable by looking at the code above and adjusting it for the new names of variables, size of the data set, etc.

First, here is some information about the variables. You will notice there is very litte useful labelling in the data set.

Code Book

Variables in the order they appear in the data set:

1. Daily energy expenditure (vigorous activity)
2. Daily energy expenditure (light activity)
3. Sex (0 = female; 1 = male)
4. BDNF-alpha genotype (0 = Risk Allele Not Present ; 1 = Risk Allele Present)
5. Attitude to exercise (1-8; 1 = lowest, 8 = highest)

Questions

1. Add an ID to the data set. **HINT**: see week 2 lab
2. Name all variables.
3. Add labels to the sex and BDNF-alpha variables.
4. Produce a descriptives table with appropriate measures of central tendency and a column for sample size.
5. Produce a frequency plot for Attitude to Exercise and a density plot for Daily energy expenditure (vigorous).
6. Plot the association between Daily energy expenditure vigorous activity and light activity.
7. Produce a plot to show the relative frequencies of BDNF-alpha risk alleles among men and women.
8. Produce a plot comparing the densities of daily energy expenditure (light activity) in men and women.
9. Combine these four plots into one figure.

Good luck!