3 Workshop 2

3.1 Learning objectives

- To select subsets of data according to properties
- To add data to a plot
- To present data on a plot in a variety of styles
- To calculate additional fields for a table.

3.2 Recap

In the previous workshop we had loaded a dataset containing data from athletes. Repeat this step so the `ais` data is available.

You will first need to set your working directory to where you saved the data during the last workshop. Go to the menu option *Session > Set Working Directory > Choose Directory . . .* and select the correct directory (note that none of the files in the directory will show up in the browser.) When you have selected the correct directory, the files should appear in the `Files` pane in the bottom right.

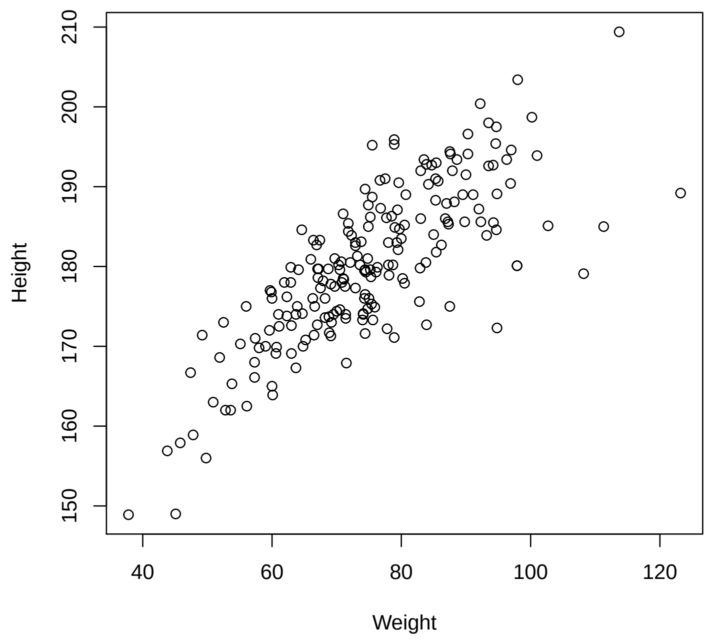
When you are in the correct directory, run the following commands to reload the data.

```
> ais <- read.table("sport.txt", sep="\t", header=T)
> attach(ais)
```

We had plotted a simple plot of the height versus the weight.

An alternative way to express this plot is as 'Y interpreted by X', or 'Y ~ X'. (~ is the 'tilde' or squiggle character, found next to the ENTER key on a PC or next to left SHIFT on a Mac). We will use this kind of expression later for linear models.

```
> plot(Height ~ Weight)
```



(Don't forget to add appropriate labels for the axes and a title)

3.3 Selecting data

The plot in the previous section plots all the data together. What if we want to plot just the data from a particular sport, or a particular gender?

3.3.1 Logical expressions

We can use logical expressions to select data. If we type an expression that will evaluate to TRUE or FALSE then this will produce a list of TRUE/FALSE values. Note that `==` is 'is equal to' and `=` means 'set equal to'. If we want to test for equality we must use `==`. A full list of logical operators is included at the end of this document.

```
> Sport=="BBall"
```

```
  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [13]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [73] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [85] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[121] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
[133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```
[145] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[157] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[181] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[193] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

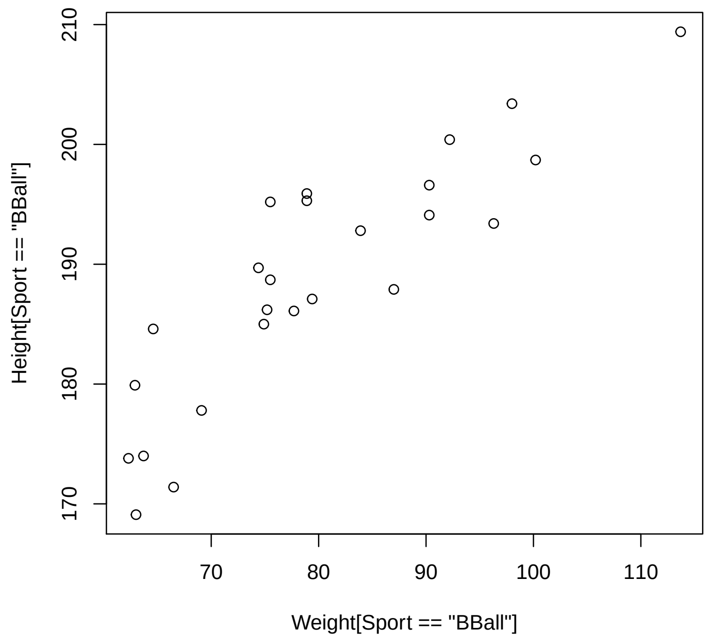
Most of the values are `FALSE` - only those corresponding to the basketball players are true. If we then use this as the index for our data we should get a subset corresponding to the basketball players. E.g. selecting their height.

```
> Height[Sport=="BBall"]
```

```
 [1] 195.9 189.7 177.8 185.0 184.6 174.0 186.2 173.8 171.4 179.9 193.4 188.7
[13] 169.1 200.4 195.3 194.1 187.9 209.4 203.4 198.7 187.1 196.6 186.1 192.8
[25] 195.2
```

Let's replot the data for just the basketball players

```
> plot(Height[Sport=="BBall"] ~ Weight[Sport=="BBall"])
```



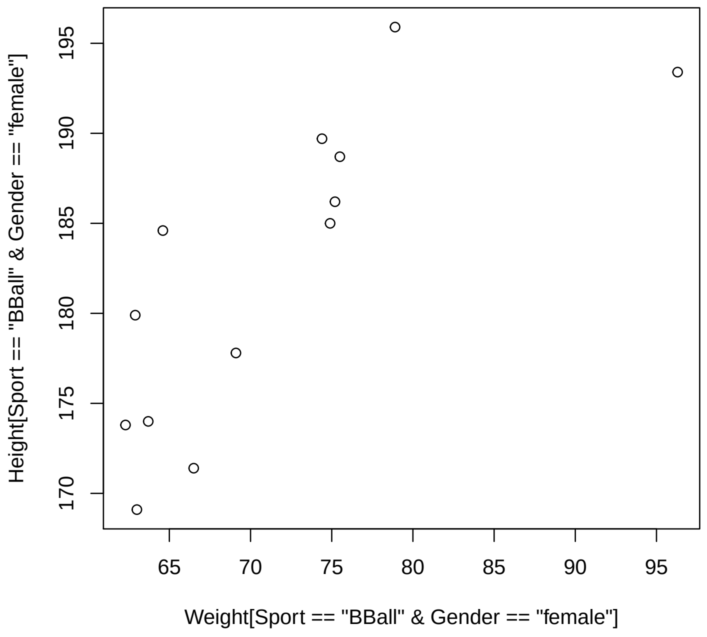
Note that we use the same selector on both X and Y data to ensure they are the same length. If the numbers of X and Y data differ then an error will occur.

3.3.2 Combining Logical Selectors

We can combine selectors using the *boolean operators* & (AND) | (OR) and ! (NOT). For AND, both criteria must be met. For OR, either criteria must be met. NOT is used in combination with the others as it inverts a particular selection. `! Sport=='Row'` would be true for all cases where the sport is NOT 'Row'.

Lets plot a graph of the female basketball players. For this we need to select on *Gender* AND on *Sport*. Please note that where the line starts with a + that this is a continuation of the command and that you can include that line (or lines) on the end of the previous line.

```
> plot(Height[Sport=="BBall" & Gender == "female"] ~
+       Weight[Sport=="BBall" & Gender == "female"])
```



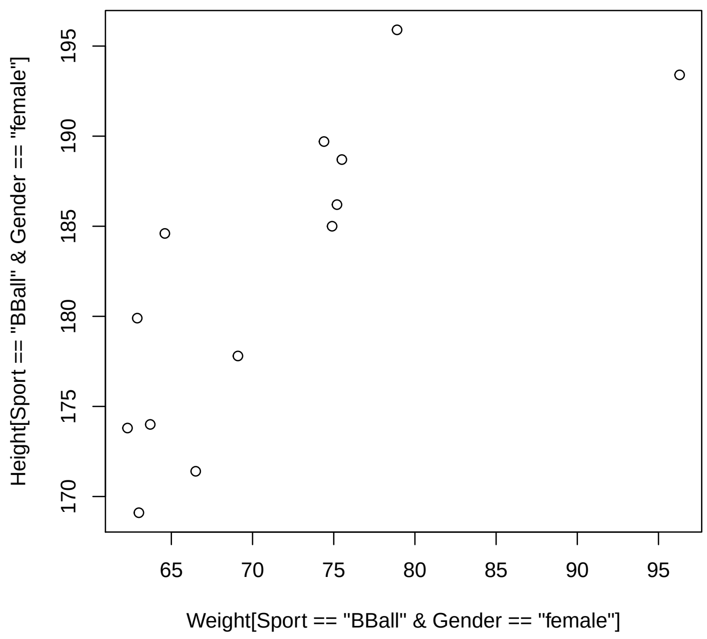
Note how in each plot the axes are rescaled to fit the data.

Challenge: *Plot the body fat percentage (Y axis) by the height (X axis) for male tennis players*

3.3.3 Adding points to a plot

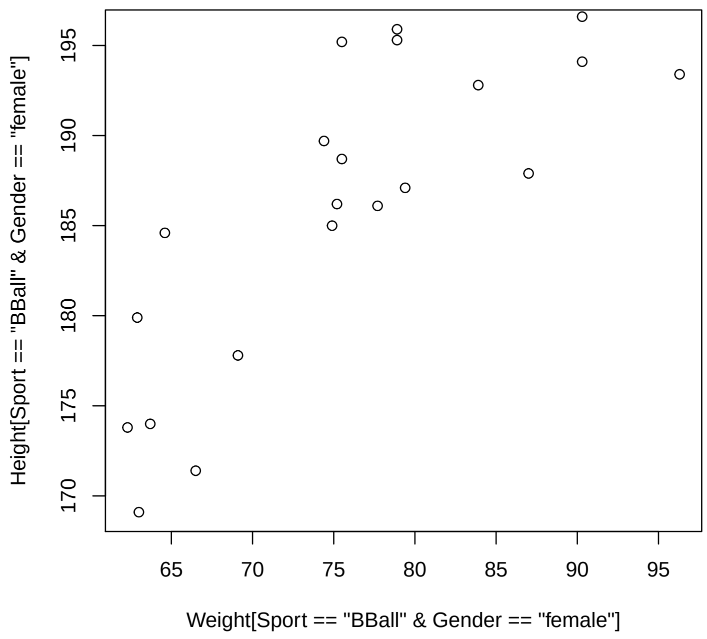
In this section we will plot two sets of data on the same graph, and learn how to change the character and colour we use for plotting. In addition we will need to find out how to set the size of our plot area.

First start off by plotting the height vs weight of the female basketball players.



Now we will add the data for the male basketball players

```
> plot(Height[Sport=="BBall" & Gender == "female"] ~
+        Weight[Sport=="BBall" & Gender == "female"])
> points(Height[Sport=="BBall" & Gender == "male"] ~
+          Weight[Sport=="BBall" & Gender == "male"])
```



Spot the problems? The points all look the same and the men are not all on the plot as the axes are scaled to fit the ladies.

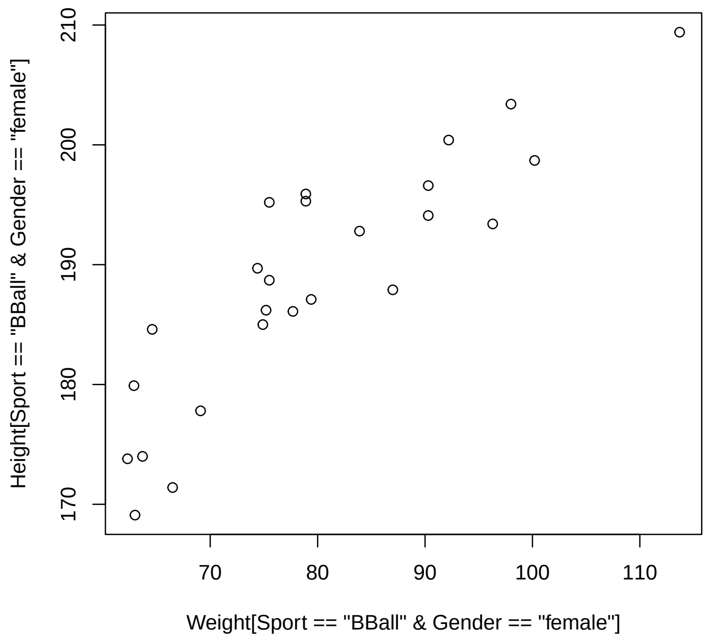
Lets deal with the axis problems first. We can explicitly set the x and y axis limits with the parameters xlim and ylim These take a list of the start and end points on the axis, e.g. `xlim=c(`*`start,end`*`)`.

First we need to establish what the limits are for our data. We can calculate the minimum and maximum with the functions `min()` and `max()`. The values will be assigned to the variables `minh`, `maxh`, `minw`and `maxw`. Check that these are the right values by looking for the minimum and maximum values in the lists returned by `Height[Sport=="BBall"]` and `Weight[Sport=="BBall"]`.

```
> minh <- min(Height[Sport=="BBall"] )
> maxh <- max(Height[Sport=="BBall"] )
> minw <- min(Weight[Sport=="BBall"] )
> maxw <- max(Weight[Sport=="BBall"] )
```

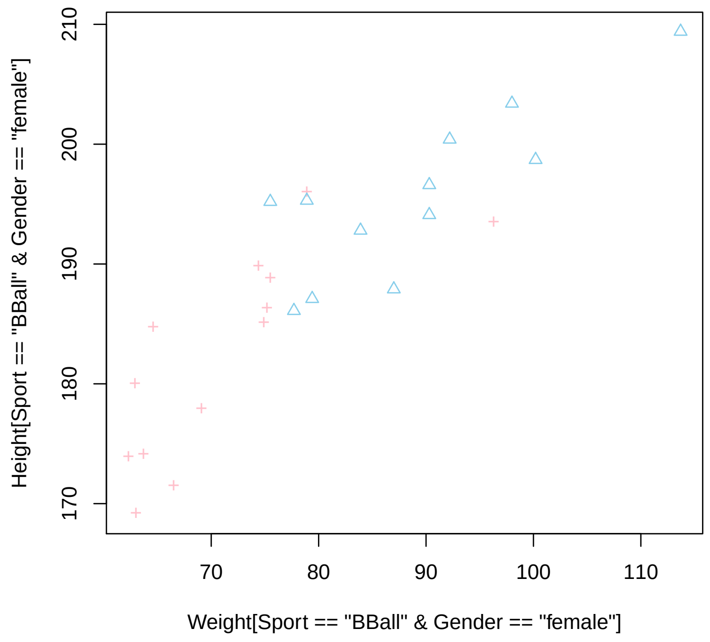
We can now use these in the plot.

```
> plot(Height[Sport=="BBall" & Gender == "female"] ~
+        Weight[Sport=="BBall" & Gender == "female"],
+      xlim=c(minw, maxw), ylim=c(minh, maxh))
> points(Height[Sport=="BBall" & Gender == "male"] ~
+          Weight[Sport=="BBall" & Gender == "male"])
```



All the points now appear in the plot but we can't tell male from female. We can use a different plotting symbol and/or a different colour for each dataset. The plot characater is set with the `pch` option and colour is set with the `col` option. We will be gender stereotypical and plot the females in pink and the men in blue. We will also set the ladies to a '+' symbol and the gentlemen to a triangle

```
> plot(Height[Sport=="BBall" & Gender == "female"] ~
+        Weight[Sport=="BBall" & Gender == "female"],
+      xlim=c(minw, maxw), ylim=c(minh, maxh), pch='+', col='pink')
> points(Height[Sport=="BBall" & Gender == "male"] ~
+          Weight[Sport=="BBall" & Gender == "male"],pch=2, col='skyblue')
```



So now we can plot different datasets on the same plot with different symbols and/or colours. We can add as many `points()` commands as we like to add data to the current plot.

Challenge: *Plot several more sports with different symbols on the same plot. Remember to set the x and y limits correctly and to include appropriate axis labels and a title*

There are many more options for plots listed in the help pages. Try `help(plot)` and `help(par)` for more information.

3.3.4 Calculating new parameters

So far we have just been using the parameters provided with the table. One commonly (mis) used metric is the Body Mass Index. This is calculated as the weight in kilos divided by the square of the height (in meters). We can calculate this directly:

```
> BMI <- Weight / ((Height/100) ** 2)
> BMI
```

```
 [1] 20.55929 20.67466 21.85821 21.88459 18.95698 21.03977 21.68995 20.62474
 [9] 22.63602 19.43517 25.74621 21.20329 22.03197 25.43572 22.63043 21.85629
[17] 22.26976 21.27456 23.47004 23.19050 23.17372 24.53621 22.95548 19.76256
[25] 23.36278 22.66565 24.23624 24.21229 20.46351 20.81005 20.16818 23.06003
[33] 24.40228 23.97437 22.61747 19.16295 21.14750 21.40367 21.02819 21.76834
[41] 21.38230 21.47268 24.44672 22.63002 22.80283 23.58011 20.06019 23.01469
[49] 24.63965 18.25796 24.47483 23.98631 26.23695 20.04237 25.72110 25.64337
```

```
 [57] 19.87217 23.35411 22.42215 20.41522 22.12997 25.17240 23.71916 21.27530
 [65] 20.86531 18.99863 22.03857 20.11651 21.34552 28.57143 26.95113 28.13049
 [73] 26.85229 25.26611 31.93286 16.74725 19.53667 20.42372 22.75871 20.11888
 [81] 22.34971 19.15766 20.76899 19.36983 22.37261 17.54151 19.05583 20.30187
 [89] 20.14602 25.36324 22.12259 21.24497 20.53350 17.05718 18.28571 18.36966
 [97] 18.93129 17.79214 17.04914 20.31440 22.46416 23.88271 23.67942 23.15344
[105] 22.32408 24.02067 23.29123 25.10633 22.80972 26.25283 21.37838 22.51519
[113] 26.72944 23.56565 25.83867 24.05738 23.84961 25.08961 23.84440 25.31399
[121] 19.68959 26.06876 25.50321 23.68758 26.79451 25.61451 25.05529 24.92820
[129] 22.95808 20.68581 23.96827 24.64142 25.93027 23.68777 25.37885 22.68156
[137] 23.36258 22.43511 22.57085 19.81469 21.19266 20.39084 21.11556 21.89201
[145] 29.97489 27.39009 23.10752 21.74694 20.88839 22.83203 22.01705 20.06671
[153] 20.14797 21.24394 19.62997 23.58304 21.64939 25.17085 23.25062 32.52009
[161] 22.59175 30.18250 34.41664 21.86382 23.99121 24.81202 21.67624 21.03583
[169] 23.12139 20.76360 23.12601 22.34566 22.28342 23.54850 19.85229 26.51095
[177] 24.77662 33.73153 30.18250 23.31338 24.50845 25.36807 23.67141 24.27816
[185] 25.82440 21.93427 23.38203 23.06536 25.20572 23.25161 22.92679 26.86370
[193] 21.25951 25.42514 24.54172 27.78987 23.57557 27.55830 23.75844 22.01302
[201] 22.33983 21.06874
```

R automatically matches the right values from each column together - we don't need to do any specific looping or programming to make sure it works.

Challenge: *What are the minimum, maximum, mean and standard deviation for BMI*

Check your results - does the mean look reasonable (between the minimum and maximum values)? Is the SD about right given a visual inspection of the data? Do the minimum and maximum correspond to what you would expect on inspecting the data?

3.3.5 Drawing lines on plots

Our athlete data has an obvious shape. There appears to be a correlation between Height and Weight. We can calculate a linear least squares fit using the linear model function, `lm`. The details of regression fitting like this will be covered in the next workshop.

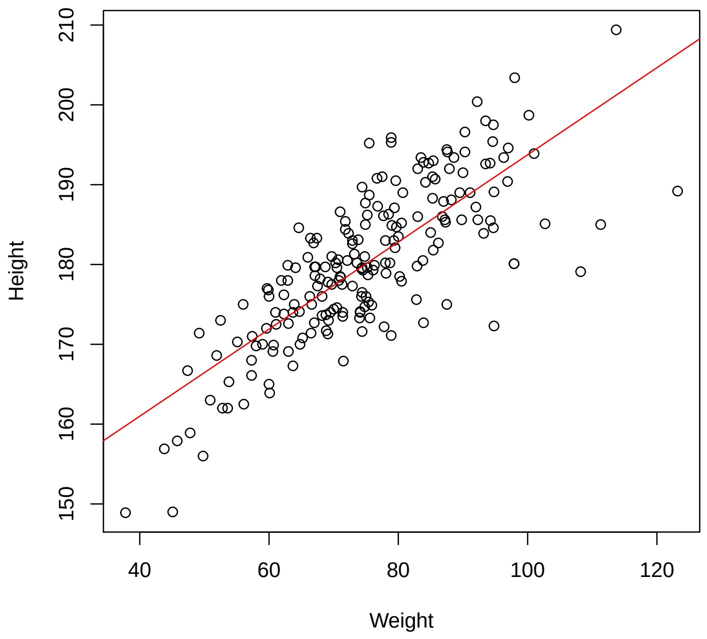
```
> lm(Height~Weight)
```

```
Call:
lm(formula = Height ~ Weight)

Coefficients:
(Intercept)       Weight
   139.1583       0.5459
```

This gives us the intecept and the gradient for the line of the form $y = mx + c$. These can be passed directly to the `abline` function to draw a regression fit on the plot.

```
> plot(Height~Weight)
> abline(lm(Height~Weight), col="red")
```



`abline` has many options. It can be used to draw any line on the plot given a gradient and intercept. Vertical or horizontal lines can be drawn with simple options. `help(abline)` will list the relevant parameters.

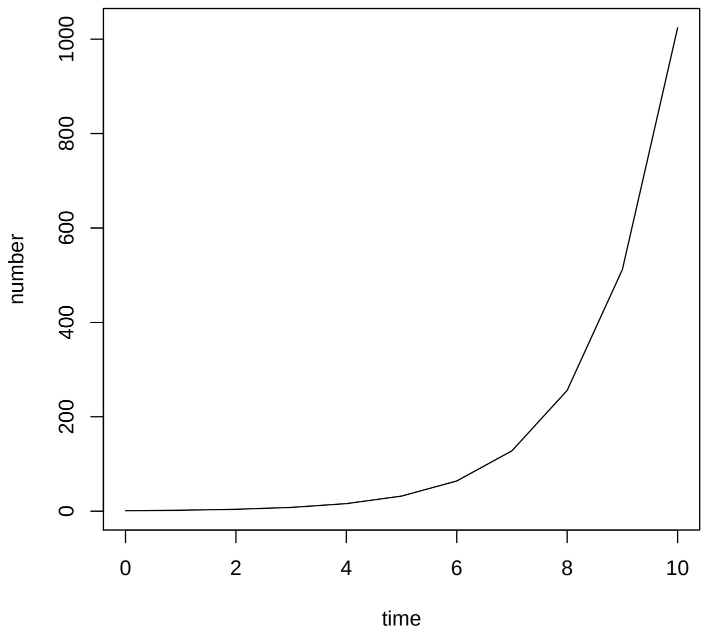
3.3.6 Dealing with logarithmic data

Let us assume that we have a bacterial culture. This doubles in number every time period t. We can model this growth as 2^t.

```
> time <- 0:10
> number <- 2 ** time
```

And by using the `type="l"` option we can tell plot to draw lines between points rather than symbols.

```
> plot(number~time, type="l")
```

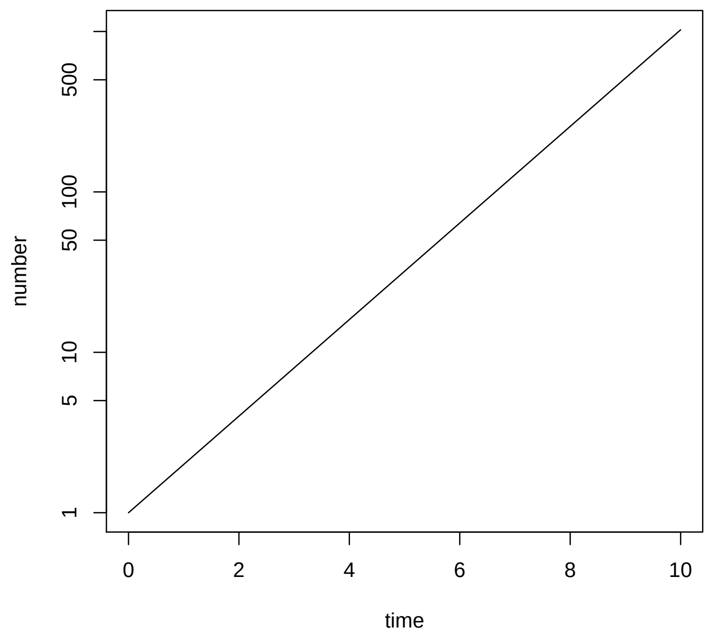


This is nice but is not terribly useful. It is more helpful to be able to plot the logistic growth with a log scale for the Y axis. We con do this on two ways. One is to plot the data on a log axis, and the other is to plot the logarithm of the data.

Note the difference in the y-axis scale between these two plots, even though they are functionally identical.

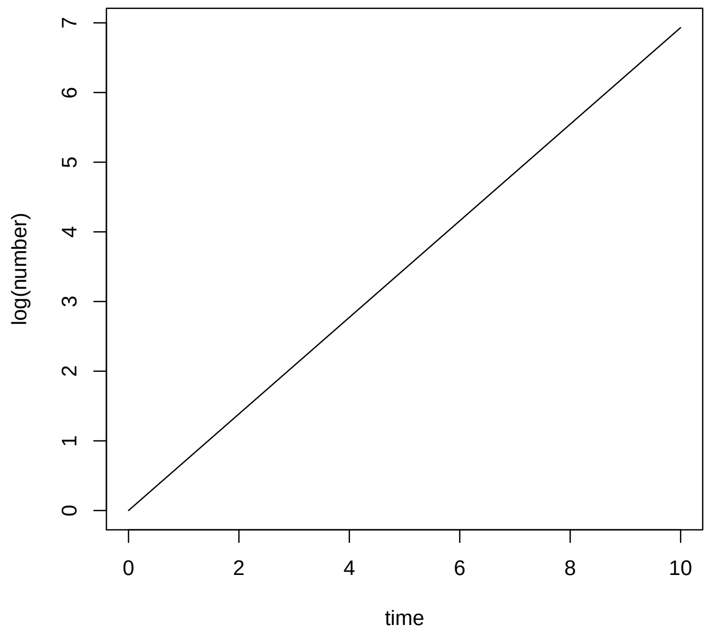
Method 1:

```
> plot(number~time, type="l", log="y" )
```



Method 2:

```
> plot(log(number)~time, type="l" )
```



3.3.7 Applying functions across factors

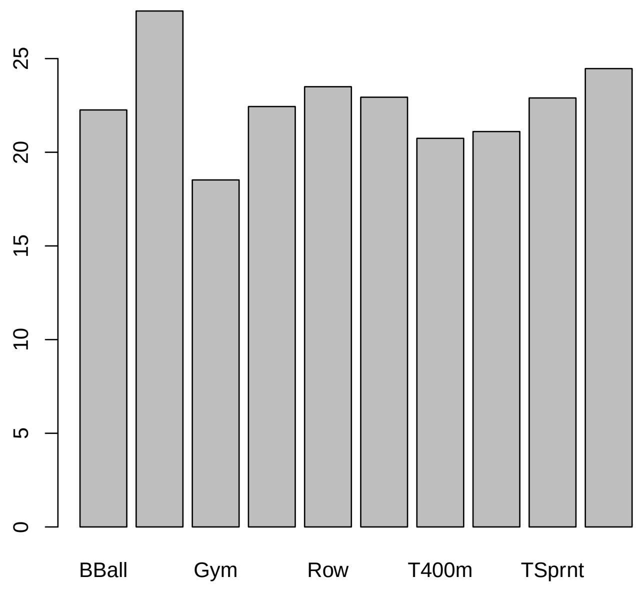
Categorical parameters are known as factors. They can be used (as we have seen above) to separate data into groups for boxplots. We can also calculate values across factors by using the `tapply()` function. `tapply()` takes a minimum of three arguments; The parameter to which the function should be applied; The grouping variable that describes the different groups that the data should be divided into; and the function that should be applied. For example, to calculate the mean of the BMI for each sport we would use `tapply()` as:

```
> tapply(BMI, Sport, mean)
```

```
   BBall    Field      Gym  Netball      Row     Swim    T400m   Tennis
22.25824 27.53994 18.52174 22.44039 23.49808 22.93838 20.74257 21.10517
  TSprnt    WPolo
22.89781 24.46598
```

This provides a table of mean counts. These can be plotted with the `barplot()` function.

```
> barplot(tapply(BMI,Sport, mean))
```



`tapply()` can also split by multiple factors, but in this case they need to be specified as a list rather than as a formula.

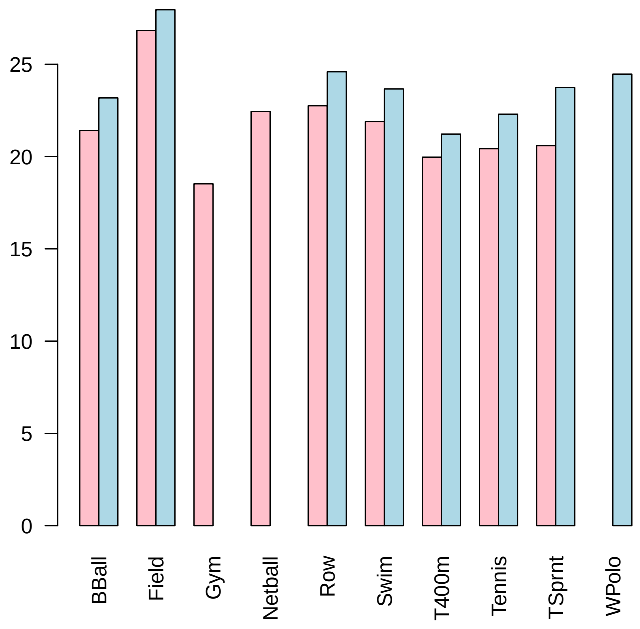
```
> tapply(BMI, list(Gender,Sport), mean)
```

```
          BBall    Field      Gym  Netball      Row     Swim    T400m   Tennis
female 21.41083 26.83154 18.52174 22.44039 22.75128 21.89297 19.96706 20.42527
male   23.17627 27.95318       NA       NA 24.59340 23.66212 21.21649 22.29501
```

```
         TSprnt    WPolo
female 20.58929       NA
male   23.73726 24.46598
```

and `barplot()` can handle multidimensional tables as well...

```
> barplot(tapply(BMI, list(Gender,Sport), mean), beside=T,
+         las=2, col=c('pink','lightblue'))
```

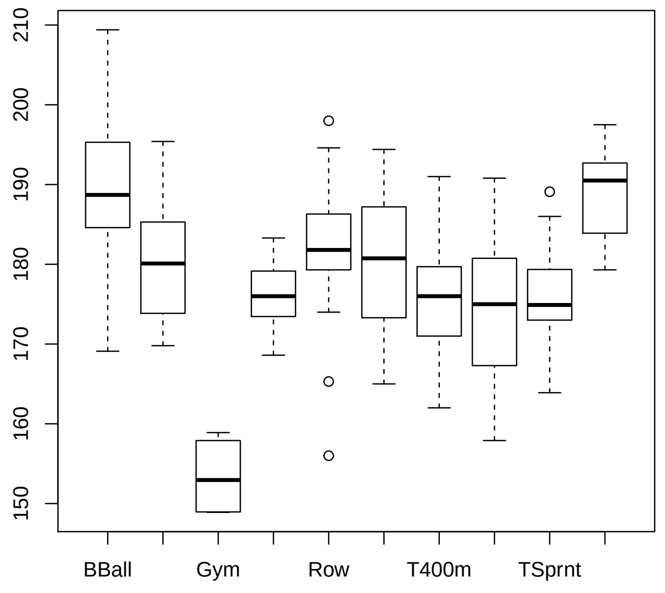


Look at `help(barplot)` to find out what the 'beside' parameter does.

3.4 Grouping data - the boxplot

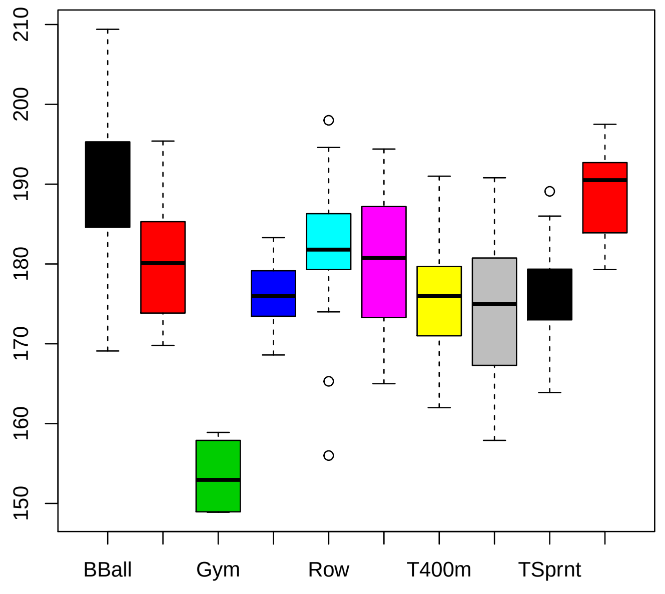
Sometimes we want to compare multiple groups. In this case we will compare between sports. One way to compare is to use a boxplot. This shows the spread of data for each group arranged in adjacent columns.

```
> boxplot(Height~Sport)
```



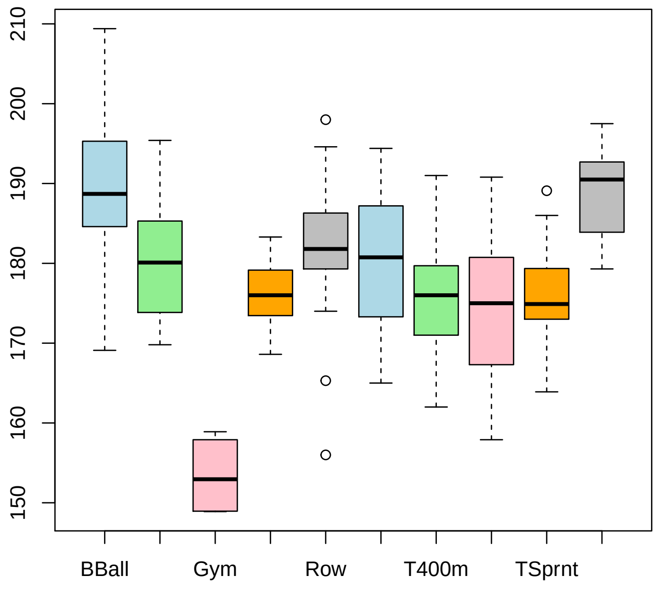
`boxplot()` takes many of the same options as `plot()`. In particular we can set the fill colour with `col`. Previously we used a single colour, but we can set a list of colours which will be used in order and recycled as necessary.

```
> boxplot(Height~Sport, col=1:10)
```



This uses the built in colours numbered 1-10. We can also specify the colours directly as a list.

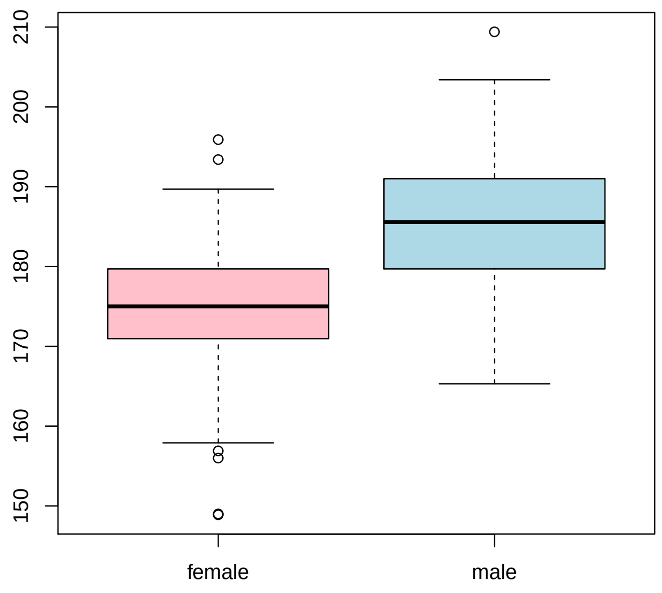
```
> boxplot(Height~Sport,
+         col=c('lightblue','lightgreen','pink','orange', 'grey'))
```



3.4.1 Combining factors in box plots

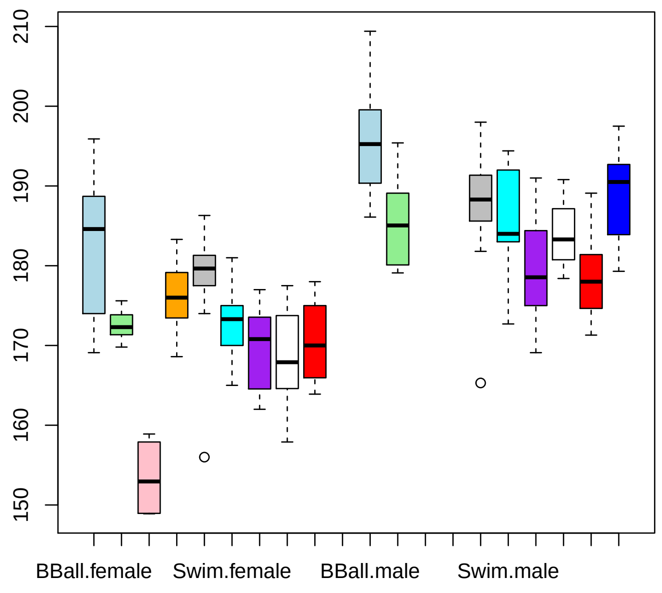
If we compare the heights of male and female athletes, there is a difference. Is this due to the inclusion of the gymnasts (exclusively female) who are much smaller than most other athletes?

```
> boxplot(Height~Gender, col=c( 'pink','lightblue'))
```



We can compare the parameters by both Gender and Sport by combining them in the 'interpreted by' section of our plot command. To do this we use the expression `Height Sport+Gender`

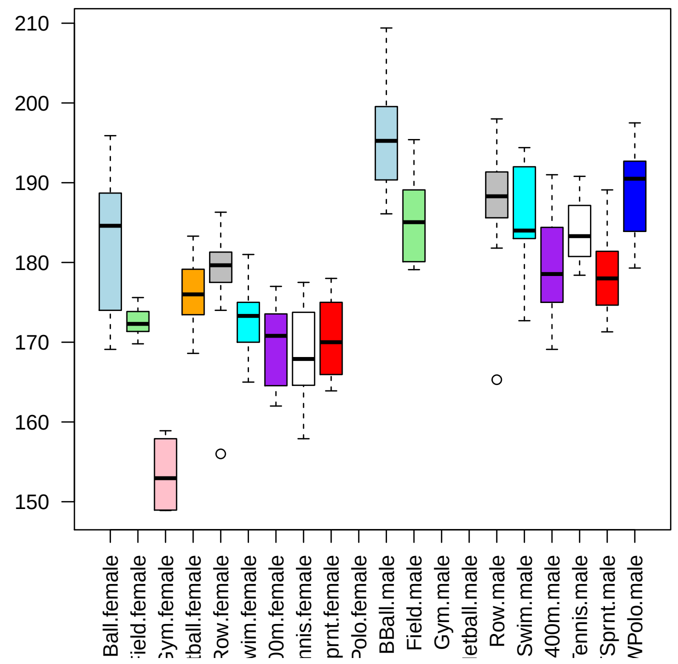
```
> boxplot(Height~Sport+Gender,
+          col=c('lightblue','lightgreen','pink','orange', 'grey',
+                'cyan', 'purple', 'white', 'red','blue'))
```



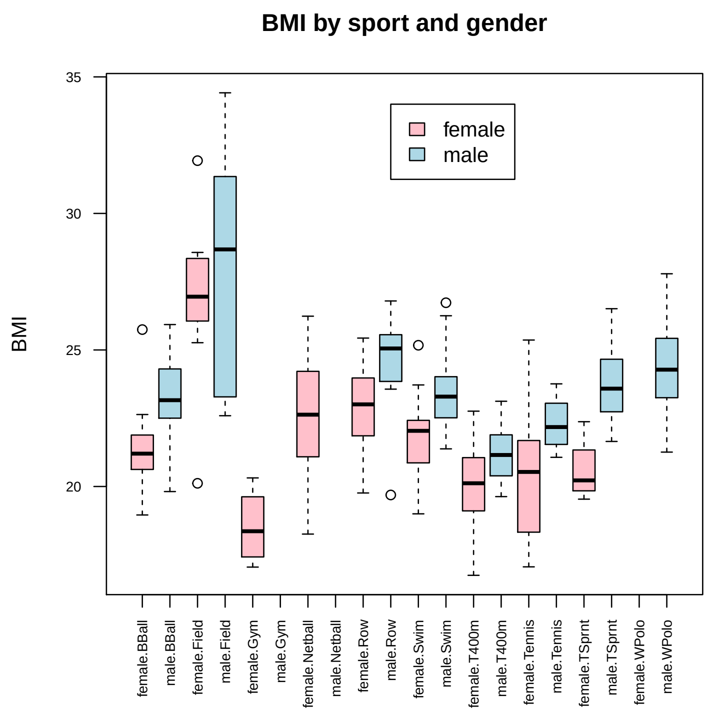
This splits the data into sections, first by Sport then secondly by Gender. *What happens if you use* ***`Gender+Sport`*** *instead of* ***`Sport+Gender`****?*

One problem with this plot is that the labels on the X axis clash with eachother so it is not always possible to see which group is which. By using `help(par)` we can see that axis label styles are set with the `las` option. Option 2 will set them perpendicular to the axis, and hence more readable.

```
> boxplot(Height~Sport+Gender,
+         col=c('lightblue','lightgreen','pink','orange', 'grey',
+               'cyan', 'purple', 'white', 'red','blue'), las=2)
```



Challenge: *Reproduce the plot shown below:*



Hint: *You may need to see what the* `legend()` *command can do ...*