Introduction to R and Basic Statistical Analysis for MATH513 Big Data and Social Network Visualization

Lecturer: Luciana Dalla Valle

{*Notes*: Julian Stander}

January 2022

Contents

1 Obtaining R and RStudio for Personal Computer Use

R is the computational engine that we will work with. In addition to doing all the necessary calculations for us, R allows us to produce profession data presentations.

We can use R through RStudio, a powerful editor and work management tool.

R and RStudio are available on the University system.

For personal computer use, R can be downloaded free of charge from the nearest CRAN (The Comprehensive R Archive Network) mirror (Bristol):

http://www.stats.bris.ac.uk/R/.

RStudio can be obtained free of charge from:

https://www.rstudio.com/products/RStudio/.

We can help you to download and install these programs in case of difficult, but it is generally not difficult.

2 Working with Data in R: Data Frames, Packages and ggplot2

The data that we're going to use to get us started are the ages in months and the heights in cms of six children sampled at random from a school class:

$$\text{age } x = \begin{pmatrix} 53 \\ 43 \\ 58 \\ 38 \\ 49 \\ 55 \end{pmatrix}_{6\times 1}$$

and

$$\text{height } y = \begin{pmatrix} 98 \\ 91 \\ 104 \\ 89 \\ 97 \\ 99 \end{pmatrix}_{6\times 1} .$$

Age and height are examples of **continuous random variables** as they can take any value in a range.

To input the data into R, we assign them to R objects.

R objects can take almost any name that we wish. We should, however, avoid using the names of existing R functions; this will come with experience.

Here we choose `x` and `y` as they are sensible general names.

To see the contents of an R object, just type its name:

```
x <- c(53, 43, 58, 38, 49, 55)
x
```

```
## [1] 53 43 58 38 49 55
```

```
y <- c(98, 91, 104, 89, 97, 99)
y
```

```
## [1]  98  91 104  89  97  99
```

We could have chosen `age` and `height`:

```
age <- c(53, 43, 58, 38, 49, 55)
age
```

```
## [1] 53 43 58 38 49 55
```

```
height <- c(98, 91, 104, 89, 97, 99)
height
```

```
## [1]  98  91 104  89  97  99
```

With R and all computer languages, it's a good idea to include comment lines (using the hash #) to remind the "future you" or what you have done:

```
#
# Save the age data in the R object x
# and the height data in the R object y
#
x <- c(53, 43, 58, 38, 49, 55)
x
```

```
## [1] 53 43 58 38 49 55
```

```r
y <- c(98, 91, 104, 89, 97, 99)
y
```

```
## [1]  98  91 104  89  97  99
```

Here, output is hashed out using ##.

`x` and `y` are examples of vectors. To access particular elements of vectors use the square brackets []:

```r
x[2]
```

```
## [1] 43
```

```r
2:4 # A integer sequence from 2 to 4
```

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

```r
x[2:4]
```

```
## [1] 43 58 38
```

```r
x[c(1, 3, 5)]
```

```
## [1] 53 58 49
```

```r
x[-c(1, 3, 5)]
```

```
## [1] 43 38 55
```

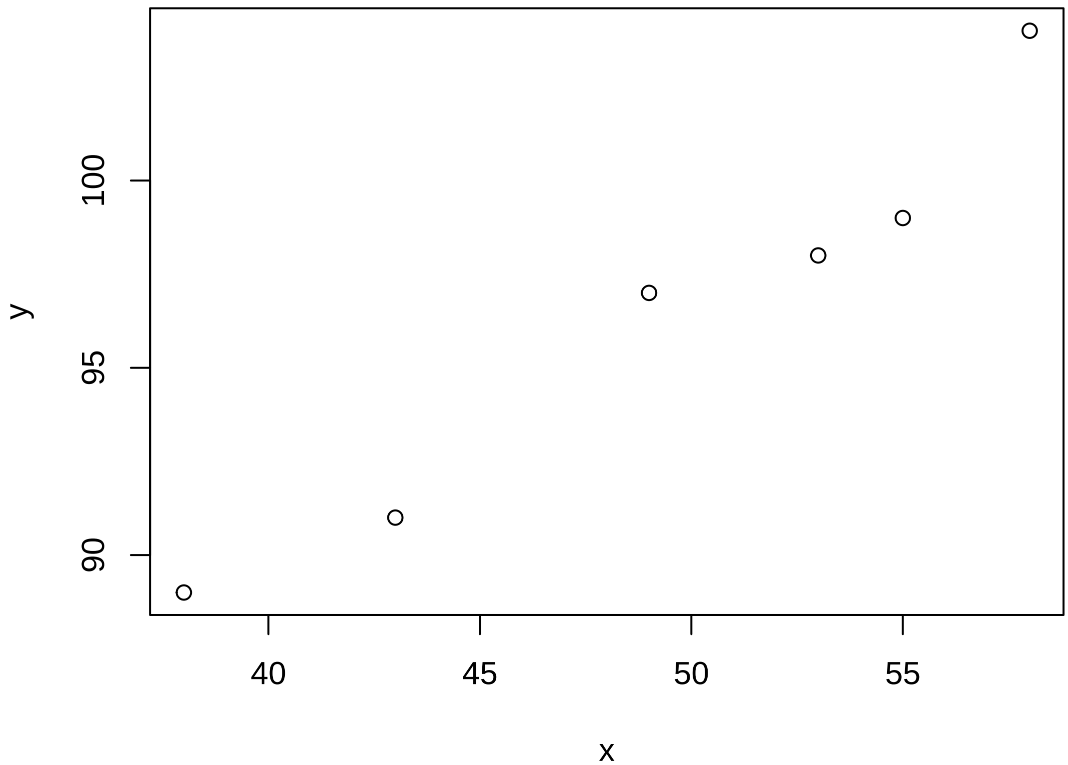
Do you understand what is happening?

2.1 A Basic Plot

The way to success in R is to build up commands slowly. Here's an example of how to produce a scatter plot:

- Simple scatter plot:

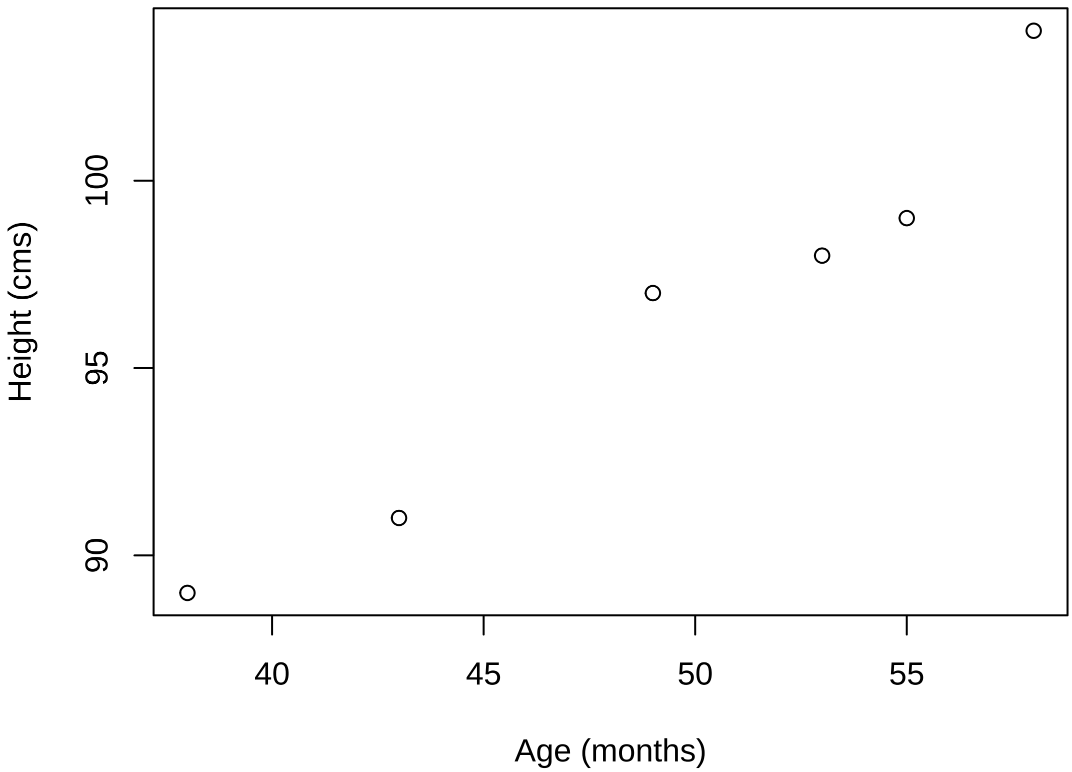
```r
plot(x, y)
```



All R functions have a name followed by **arguments** in round brackets ( ). Here, the basic arguments of the `plot` function are `x` and `y`, corresponding to what is to be plotted on the x- and y- axes. Other function arguments are possible. The = sign is used to assign values to them.

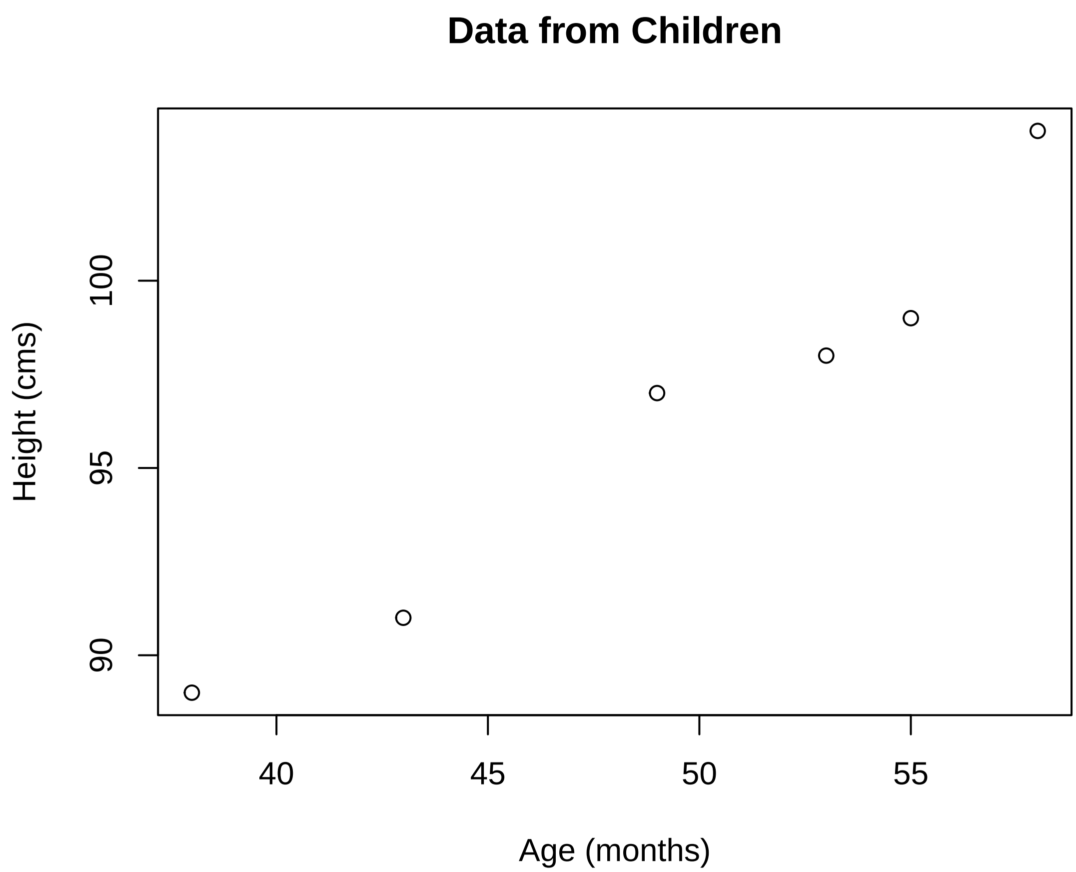
- Add axis labels using the arguments **xlab** and **ylab**:

```
plot(x, y, xlab = "Age (months)", ylab = "Height (cms)")
```



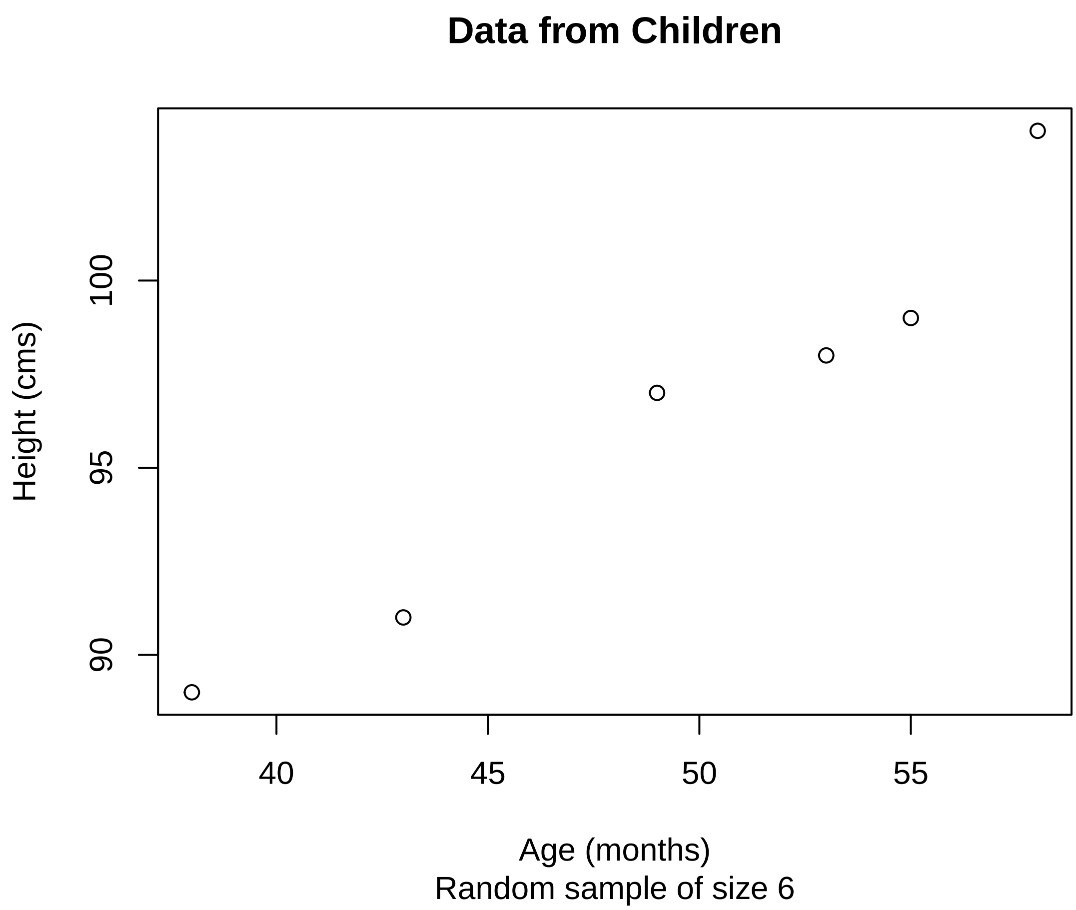
- Add a main title using the argument **main**:

```
plot(x, y, xlab = "Age (months)", ylab = "Height (cms)", main = "Data from Children")
```



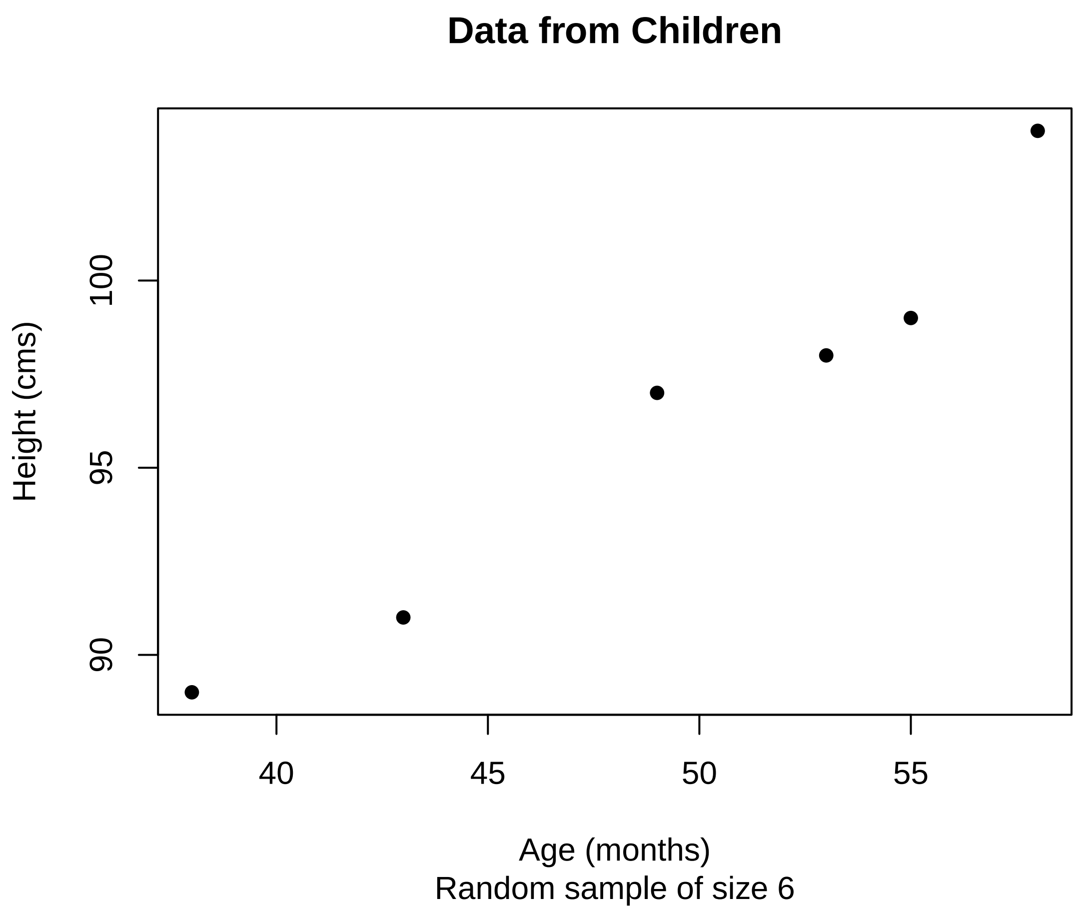
- Add a subtitle using the argument **sub**:

```
plot(x, y, xlab = "Age (months)", ylab = "Height (cms)", main = "Data from Children",
     sub = "Random sample of size 6")
```



- Fill the dots by specifying the number of a **plotting character** using **pch**:

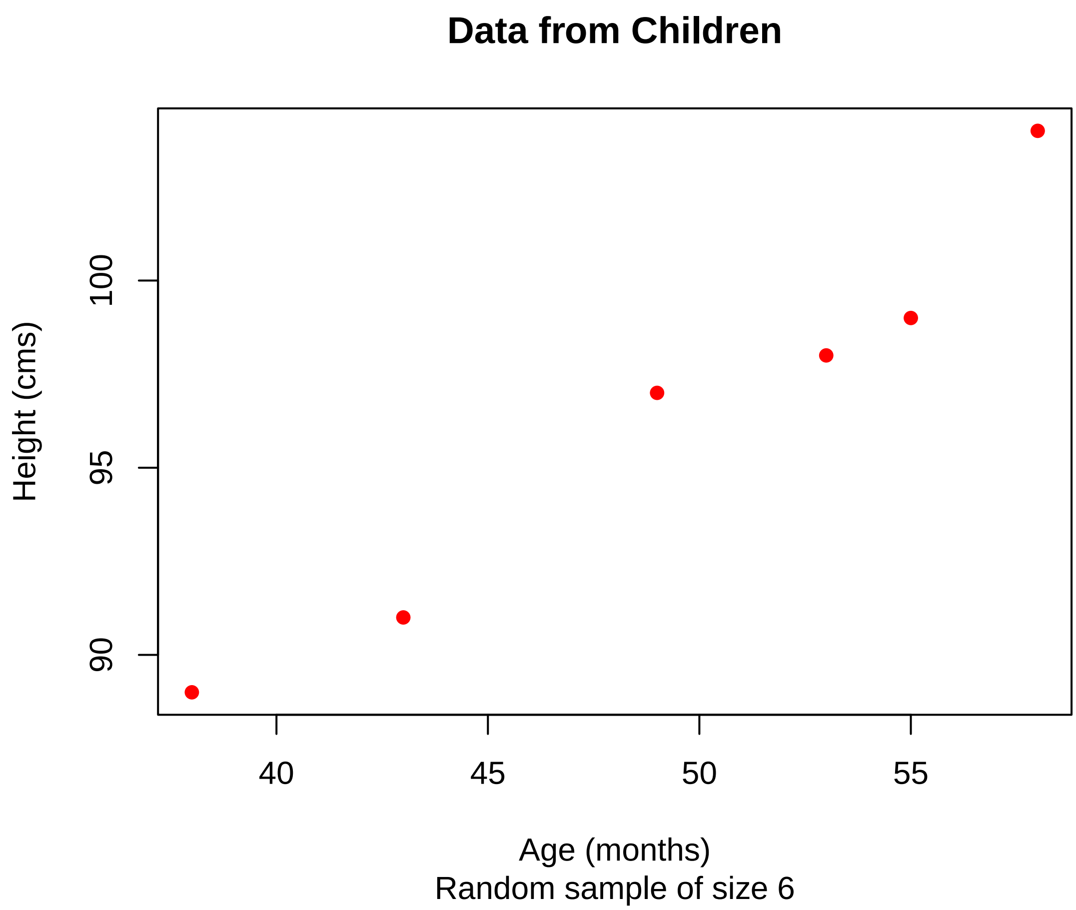
```
plot(x, y, xlab = "Age (months)", ylab = "Height (cms)", main = "Data from Children",
     sub = "Random sample of size 6", pch = 16)
```



Exercise: What happens if **pch = 17**?

- Make the dots red in **colour**:

```
plot(x, y, xlab = "Age (months)", ylab = "Height (cms)", main = "Data from Children",
     sub = "Random sample of size 6", pch = 16, col = "red")
```



The **sample mean** (or just mean) of the values $x_1, \ldots, x_n$ is denoted $\bar{x}$ and is defined as

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n};$$

here the sample size $n = 6$ and $x_1 = 53$, $x_2 = 43$, $x_3 = 58$, $x_4 = 38$, $x_5 = 49$ and $x_6 = 55$. To obtain the sample mean, we add up all the values and divide by the number of values.

In R we can calculate the sample mean using the function `mean`:

```
mean(x)
```

```
## [1] 49.33333
```

To confirm this we can work out $\sum_{i=1}^{n} x_i$ and n:

```
sum(x) # sums the elements of the argument x
```

```
## [1] 296
```

```
length(x)
```

```
## [1] 6
```

Dividing these we obtain the sample mean:

```
sum(x) /length(x)
```

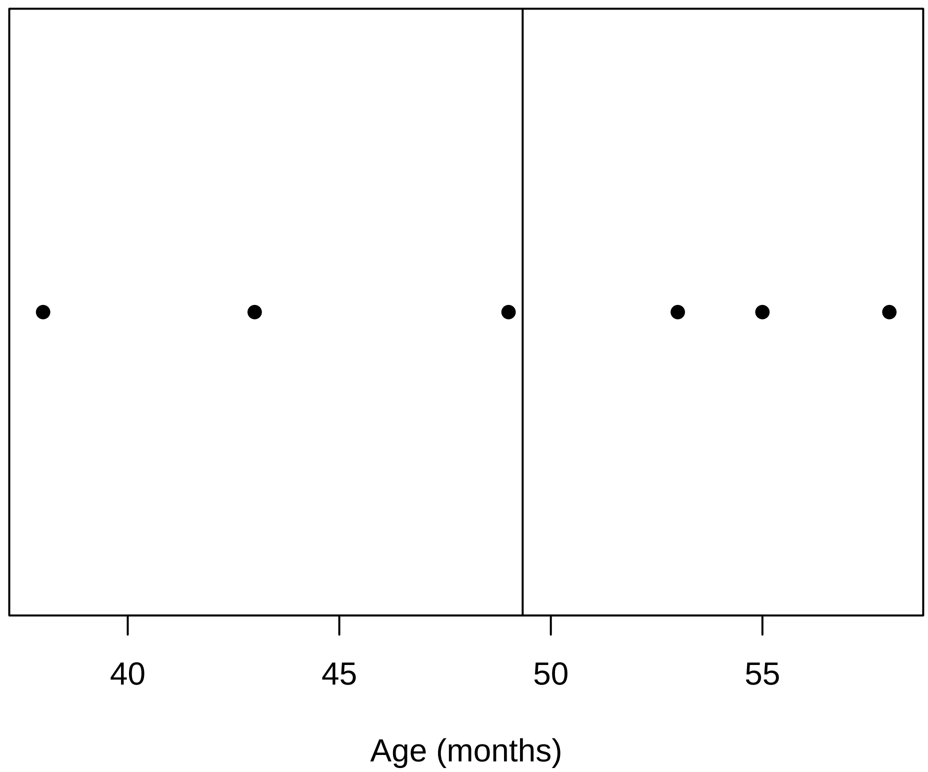
```
## [1] 49.33333
```

```
mean(x)
```

```
## [1] 49.33333
```

Let us plot the age data using a stripchart together with the sample mean as a vertical line. The `abline` function to draw a `line` (from `a` to `b`):

```
stripchart(x, pch = 16, xlab = "Age (months)")
abline(v = mean(x)) # adds a line (from a to b)
```



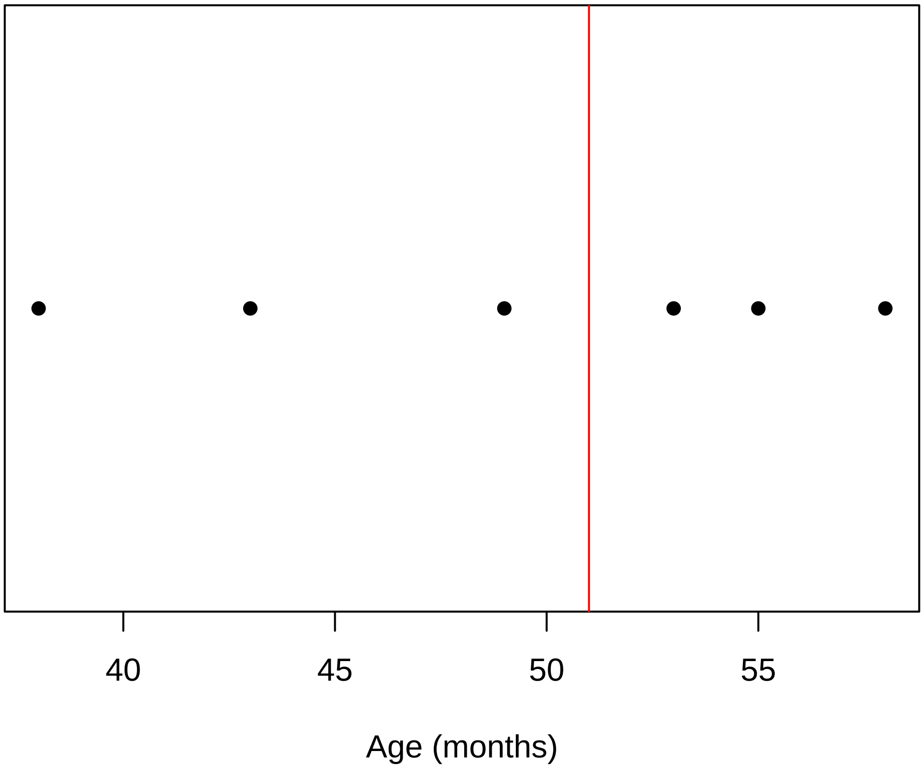
The sample mean is seen to be a **measure of location**. It is a one number summary of the location of the data. Some values are higher and some are lower. The sample mean is a typical value.

The **sample median**, calculated by the function `median`, divides the data into two parts containing the same number of points:

```
median(x)
```

```
## [1] 51
```

```
stripchart(x, pch = 16, xlab = "Age (months)")
abline(v = median(x), col = "red")
```



When the sample size $n = 6$ is even, the sample median is the mean of the third $(n/2)$ and fourth $(n/2 + 1)$ data points when `x` is `sort`ed:

```
sort(x) # x values in ascending order
```

```
## [1] 38 43 49 53 55 58
```

```
n <- length(x)
n
```

```
## [1] 6
```

```
n/2
```

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

```
n/2 + 1
```

```
## [1] 4
```

```
sort_x <- sort(x)
sort_x
```

```
## [1] 38 43 49 53 55 58
```

```
sort_x[n/2]
```

```
## [1] 49
```

```
sort_x[n/2 + 1]
```

```
## [1] 53
```

```
(sort_x[n/2] + sort_x[n/2 + 1]) / 2
```

```
## [1] 51
```

```r
median(x)
```

```
## [1] 51
```

```r
(n/2):(n/2 + 1)
```

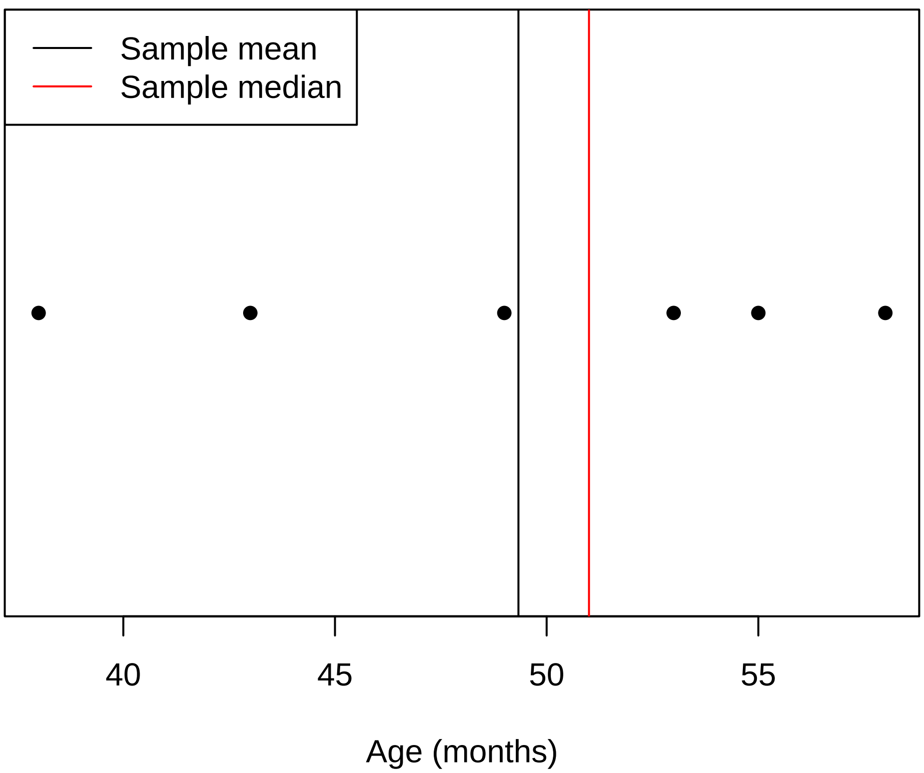
```
## [1] 3 4
```

```r
mean(sort_x[(n/2):(n/2 + 1)])
```

```
## [1] 51
```

Here we show the sample mean and the sample median together. We also add a legend:

```r
stripchart(x, pch = 16, xlab = "Age (months)")
abline(v = mean(x))
abline(v = median(x), col = "red")
legend("topleft",
       legend = c("Sample mean", "Sample median"),
       # The legend argument provides the words for the legend
       lty = 1, # We need line type 1...
       col = c("black", "red")) #...using specific colours
```



The sample median is said to be **robust to outliers**. This means that, if there were a data point that was a long way from the other points, the sample median would not be affected very much, while the sample mean would be:

```r
x_with_outlier <- c(x, 1000) # Add an outlier
mean(x_with_outlier)
```

```
## [1] 185.1429
```

```r
mean(x) # Big difference
```

```
## [1] 49.33333
```

```r
median(x_with_outlier)
```

```
## [1] 53
```

```r
median(x) # Small difference
```

```
## [1] 51
```

Now, the sample size $n = 7$ is odd and the median is the $(n + 1)/2$ data point of the sorted data:

```r
n_with_outlier <- length(x_with_outlier)
n_with_outlier
```

```
## [1] 7
```

```r
(n_with_outlier + 1) / 2
```

```
## [1] 4
```

```r
sort(x_with_outlier)[(n_with_outlier + 1) / 2]
```

```
## [1] 53
```

```r
median(x_with_outlier)
```

```
## [1] 53
```

- Now we add vertical and horizontal lines at the sample means of `x` and of `y` to our original scatter plot

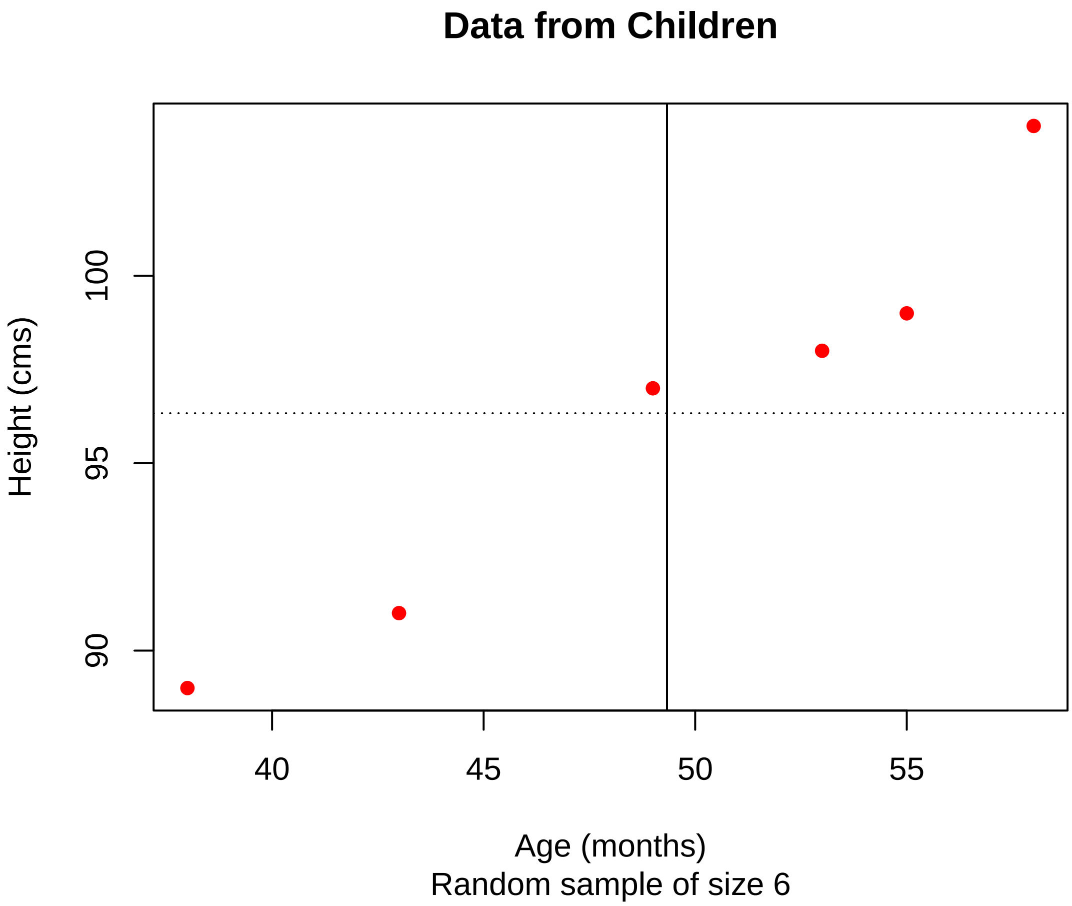
```r
mean(x)
```

```
## [1] 49.33333
```

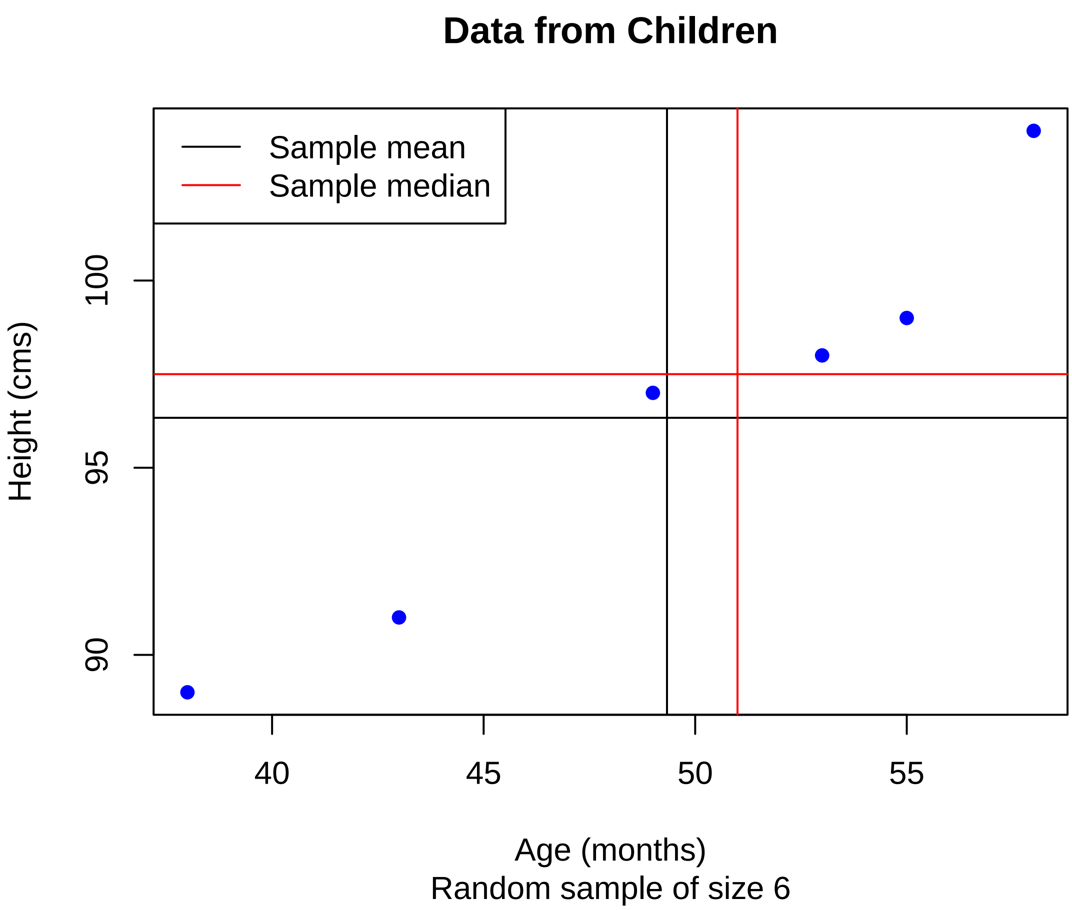
```r
mean(y) # The sample mean of the height data y
```

```
## [1] 96.33333
```

```r
plot(x, y, xlab = "Age (months)", ylab = "Height (cms)", main = "Data from Children",
     sub = "Random sample of size 6", pch = 16, col = "red")
abline(v = mean(x))
abline(h = mean(y), lty = "dotted")
```



Exercise: Add to this plot the sample medians of `x` and of `y`:



2.2 Getting Help

It's easy to read the help file of a function:

```
?abline
help(abline) # More to type
```

The help file appears in an RStudio window; alternatively a web browser may open.

- To **search** the help system for information on a topic:

```
help.search("median")
```

- To **start** the hypertext version of R's online documentation:

```
help.start()
```

- To search the R Project web site

```
RSiteSearch("median")
```

You can always find an enormous amount of information about an R topic by performing a Google search.

2.3 Data Frames

Many data sets are stored in the form of a **data frame** as we will see. These collect together all the data. We can collect together our **x** and **y** data into a data frame as follows:

```
df <- data.frame(x, y)
df
```

```
##    x   y
## 1 53  98
## 2 43  91
## 3 58 104
## 4 38  89
## 5 49  97
## 6 55  99
```

It's easier to deal with one data frame than with more than one separate variables.

- The **columns** of a data frame correspond to **variables**.
- The **rows** of a data frame correspond to **individual units**. Here the units are **children**, but they can take many forms such as patients or experiments.

If you are working on a machine that does have the package `readr` installed, you will have to install it; please see below in the section on packages.

It can be useful to think of how these data would be stored in Excel:

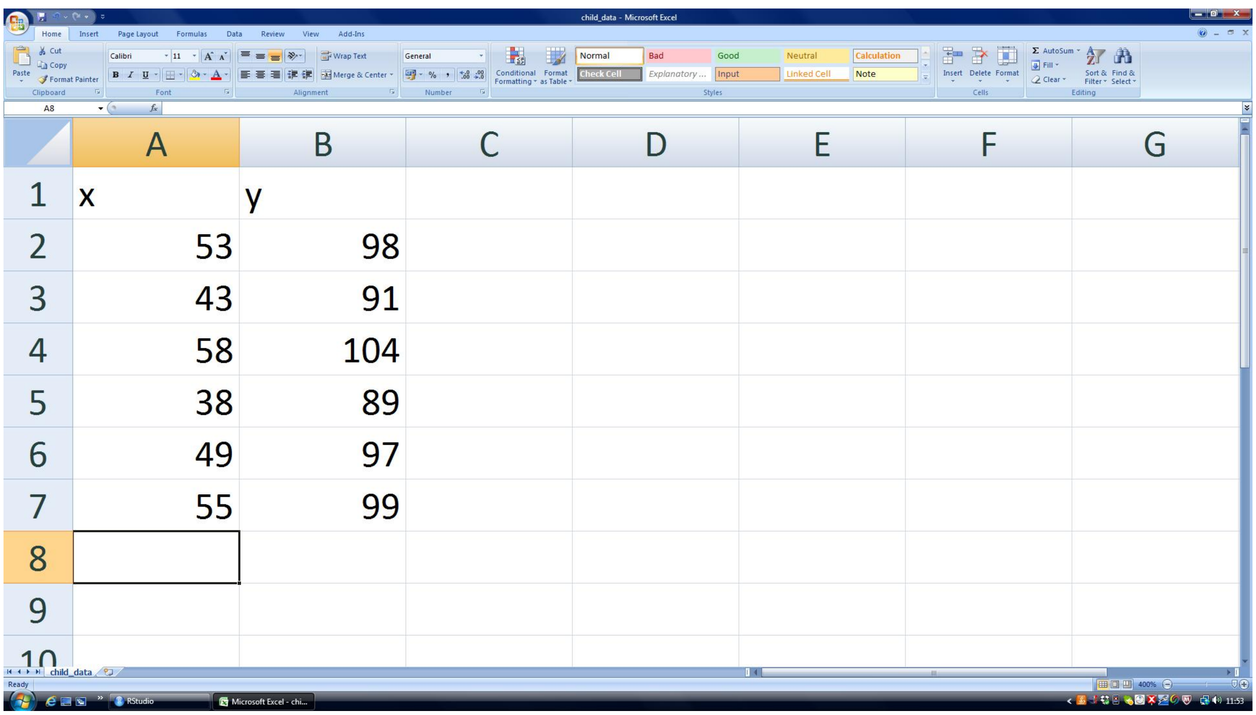


Figure 1: How these data would be stored in Excel

Once more we see that we have one column for the `x` values and one column for the `y` values. The first row of the spreadsheet provides the column "headers" or variable names. After that row, there is one row for the data from each child.

To see the column or variable **names** in a data frame use the `names` function:

```
df # To remind us of the data frame
```

```
##    x   y
## 1 53  98
```

```
## 2 43  91
## 3 58 104
## 4 38  89
## 5 49  97
## 6 55  99
```

```
names(df)
```

```
## [1] "x" "y"
```

To access a column or variable of a data frame use the dollar $:

```
df$x
```

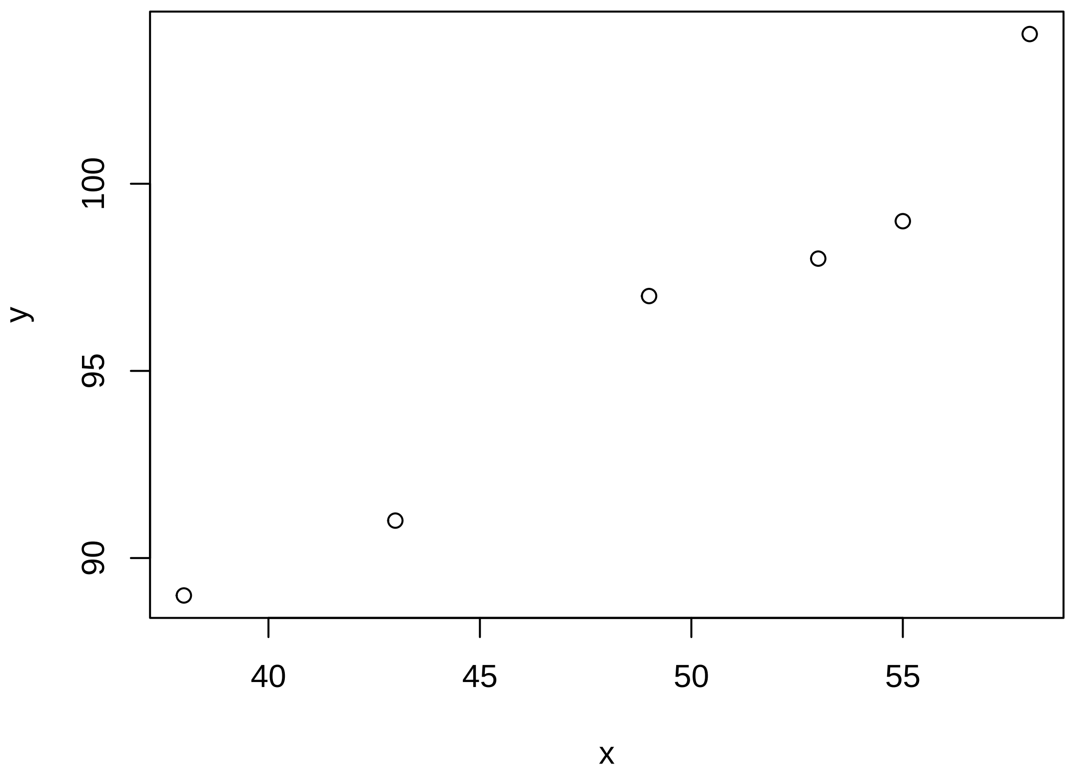
```
## [1] 53 43 58 38 49 55
```

```
df$y
```

```
## [1]  98  91 104  89  97  99
```

The **plot** function can be told to look into a data frame as follows:

```
plot(y ~ x, data = df)
```



Note that **y** ~ **x** means "how does **y** depend on **x**?".

2.4 Packages

Many useful functions and interesting data sets can be found in packages that have been contributed by generous individuals or companies. One such package is `ggplot2` by Hadley Wickham. This package provides elegant graphics for data analysis.

Many packages are already installed on the university system. These do not need to be reinstalled. Packages will not have been installed on personal computers. To install them from the command line use `install.packages`:

```
install.packages("ggplot2", repos = "http://www.stats.bris.ac.uk/R/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/n7/fj8q0jwx65g6mdk5d2phfg7wmxt52w/T//Rtmpj6DJIO/downloaded_packages
```

More information on packages, including their manuals, can be obtained from:

http://www.stats.bris.ac.uk/R/web/packages/,

for example.

For R to use a package it has to be installed. This can be achieved by using the functions `library` or (more usual these days) `require`, as we will see.

2.5 Using ggplot2

`gg` stands for grammar of graphics, and `ggplot2` provides tools for drawing graphs that follow the grammar of graphics scheme laid down by

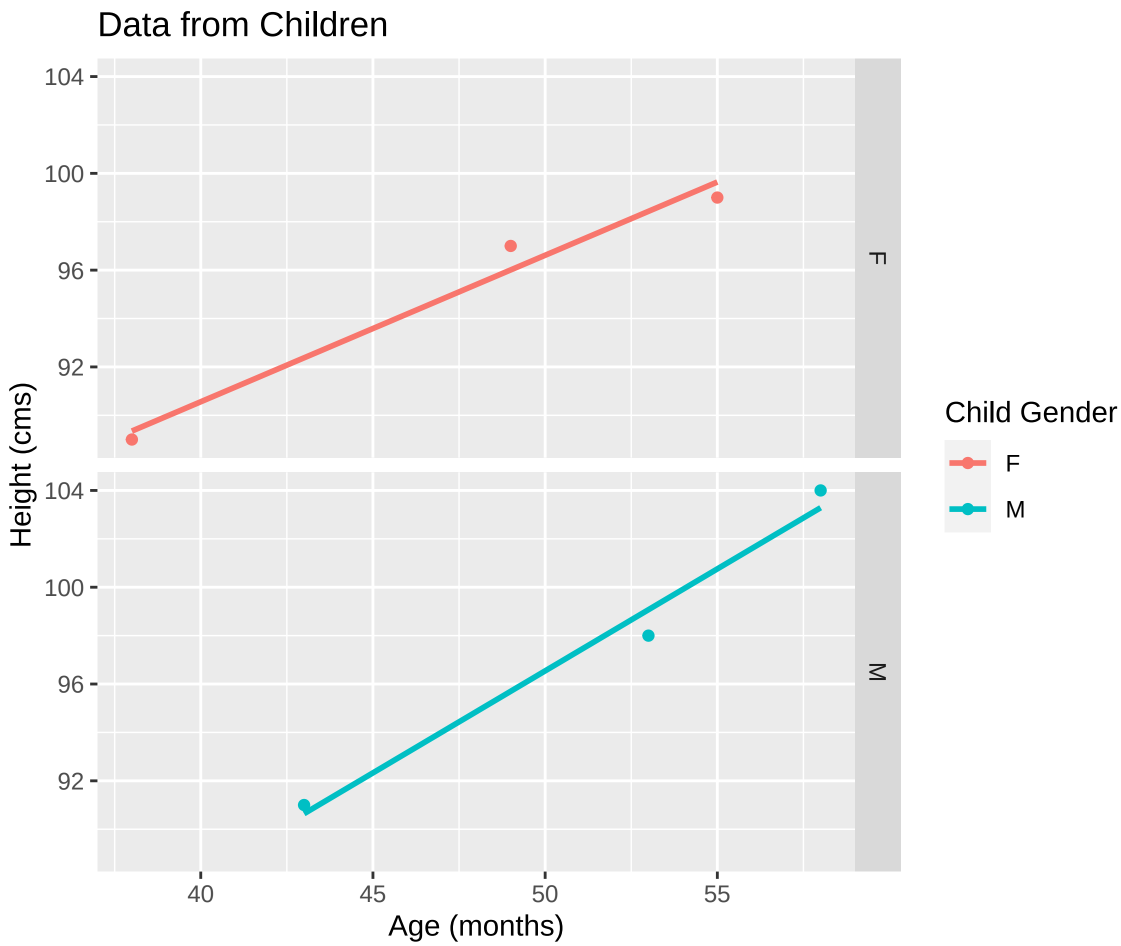
Wilkinson (2005) *The Grammar of Graphics.* Statistics and Computing. Second Edition. Springer.

To load the `ggplot2` package and to read its citation use `require` and `citation`:

```
require(ggplot2)
citation("ggplot2")
```

```
##
## To cite ggplot2 in publications, please use:
##
##   H. Wickham. ggplot2: Elegant Graphics for Data Analysis.
##   Springer-Verlag New York, 2016.
##
## A BibTeX entry for LaTeX users is
##
##   @Book{,
##     author = {Hadley Wickham},
##     title = {ggplot2: Elegant Graphics for Data Analysis},
##     publisher = {Springer-Verlag New York},
##     year = {2016},
##     isbn = {978-3-319-24277-4},
##     url = {https://ggplot2.tidyverse.org},
##   }
```

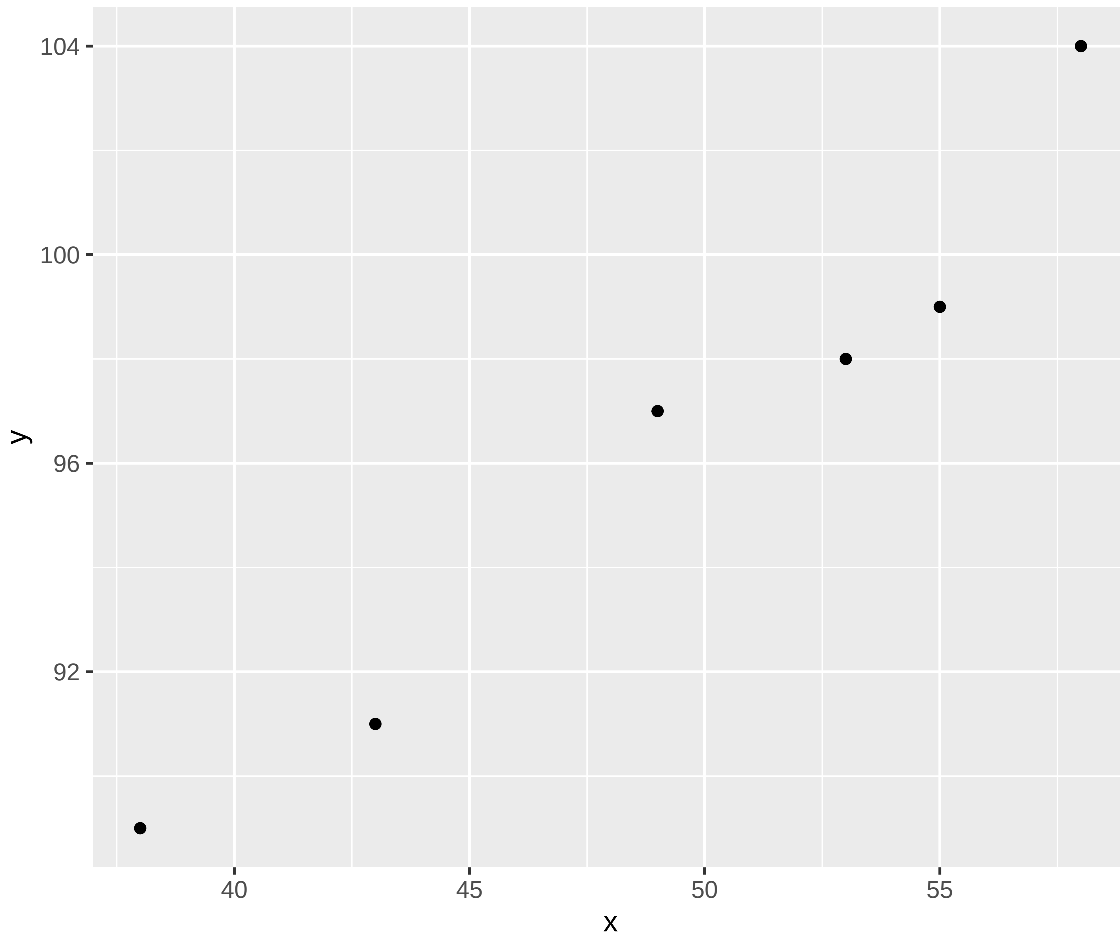
Producing plots in `ggplot2` is more difficult than producing plots in base R as we have just done, but the results can be very impressive. Moreover, once we have mastered the basics of `ggplot2`, it is possible to produce very sophisticated plots such as this one, which we will build up to:



In `ggplot2`, we begin by defining the *features* of a plot, visual properties that affect the way observations are displayed. Examples of aestetics are what is to be plotted on the **x** or **y** axis, the colour or the shape of the plotting points.

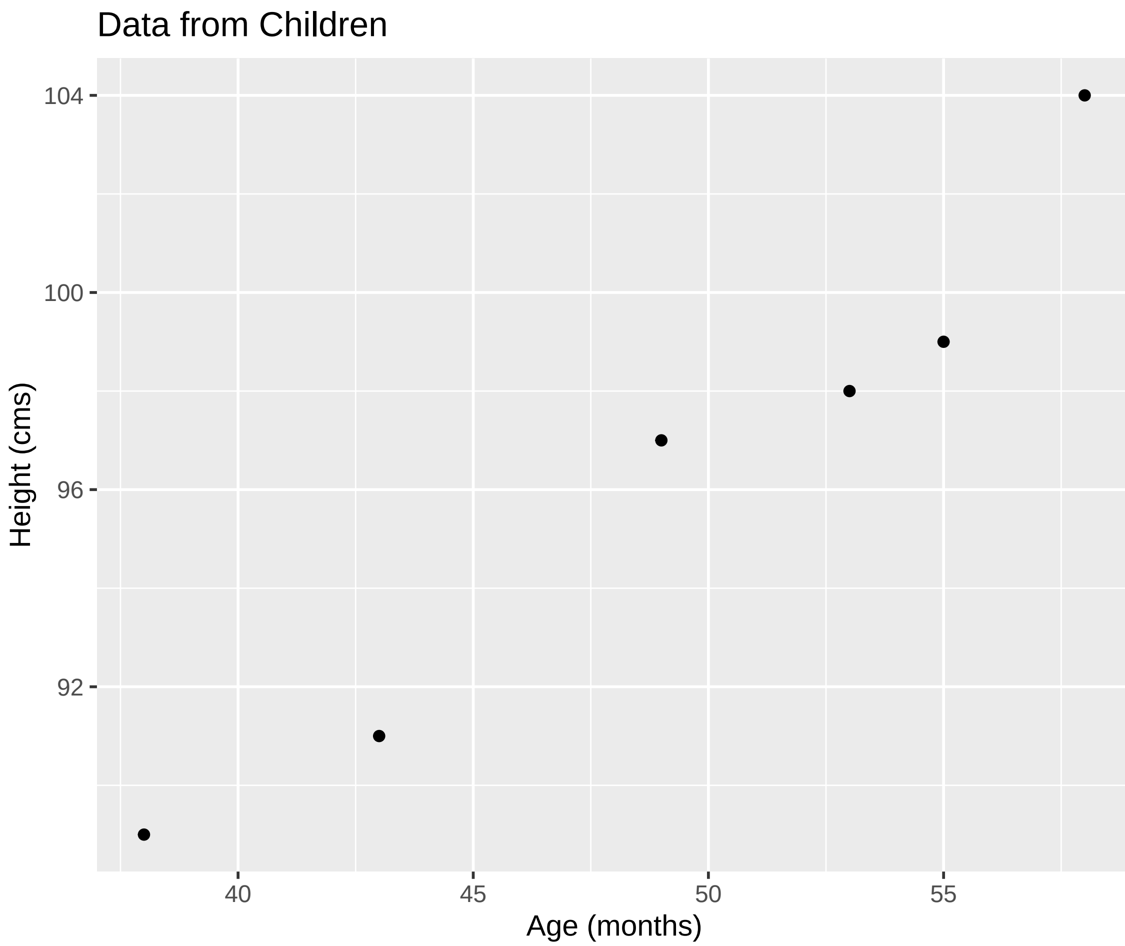
- To get started let's define the plot aestetics with the **x** values from the **df** data frame on the **x** axis and the **y** values from the **df** data frame on the **y** axis.
- We then add to this a **layer** of **points**. This is done using a so called **point geometry** as follows:

```
ggplot(df, aes(x = x, y = y)) + geom_point()
```



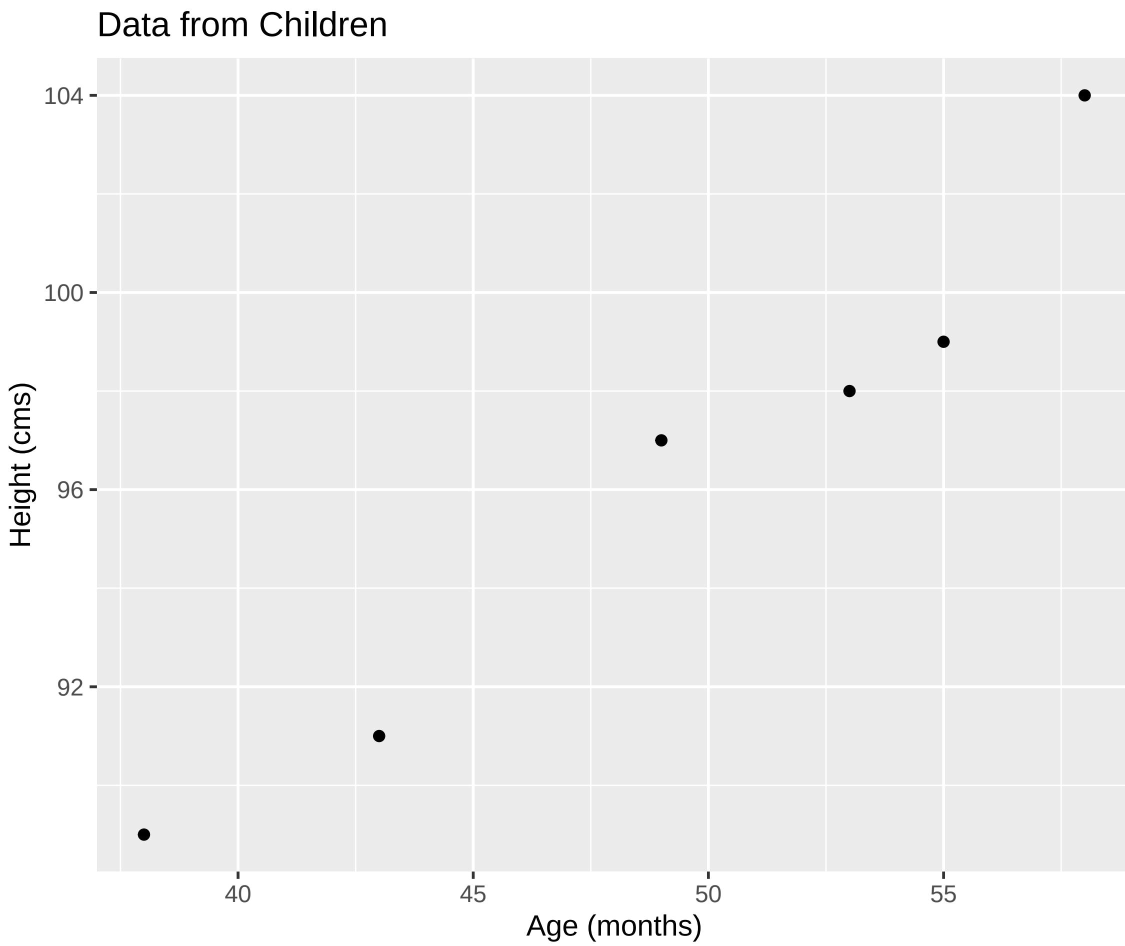
- We can add labels and titles

```
ggplot(df, aes(x = x, y = y)) + geom_point() +
  ggtitle("Data from Children") +
  xlab("Age (months)") + ylab("Height (cms)")
```



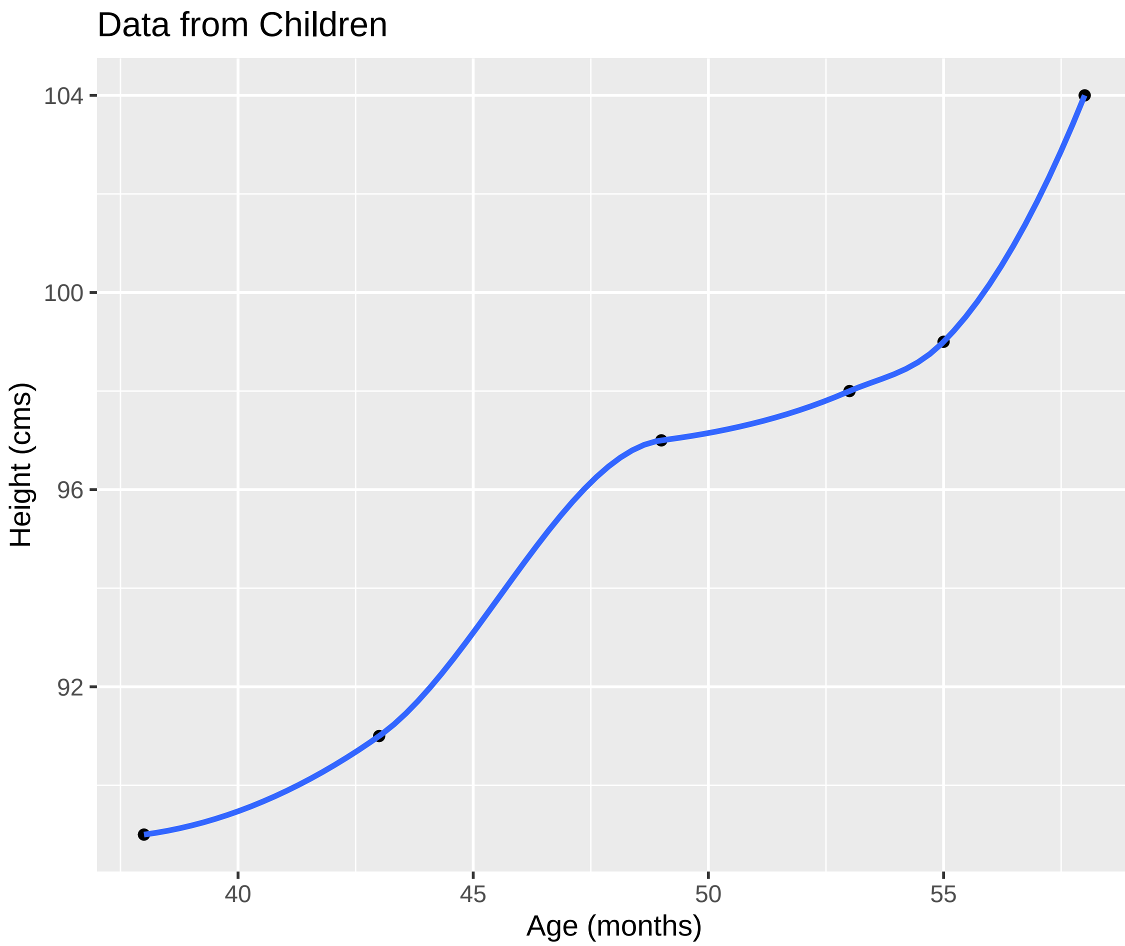
Alternatively, the `labs` function relates directly to the aesthetics:

```
ggplot(df, aes(x = x, y = y)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)")
```



- We can even add a **smooth curve geometry**:

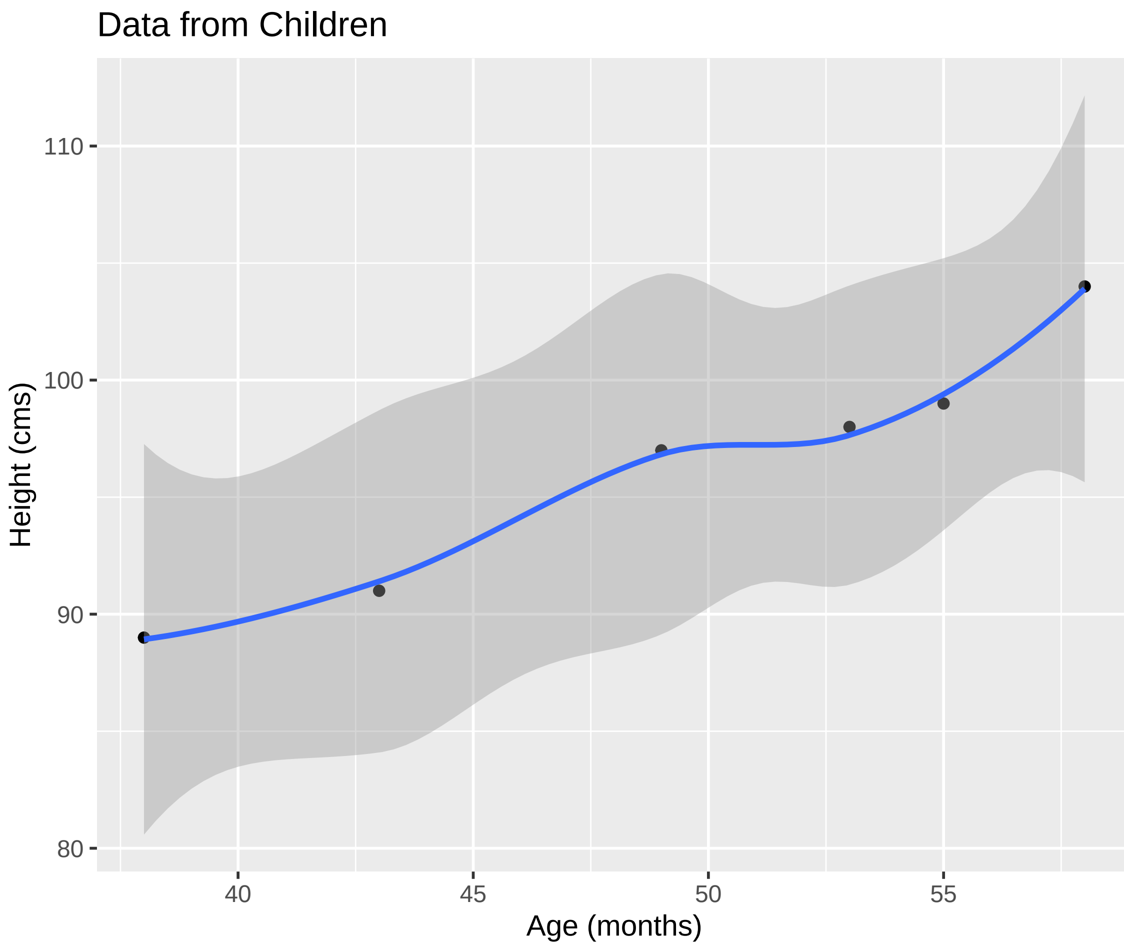
```
ggplot(df, aes(x = x, y = y)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)")  +
  geom_smooth()
```



Please don't concern yourself about the information message.

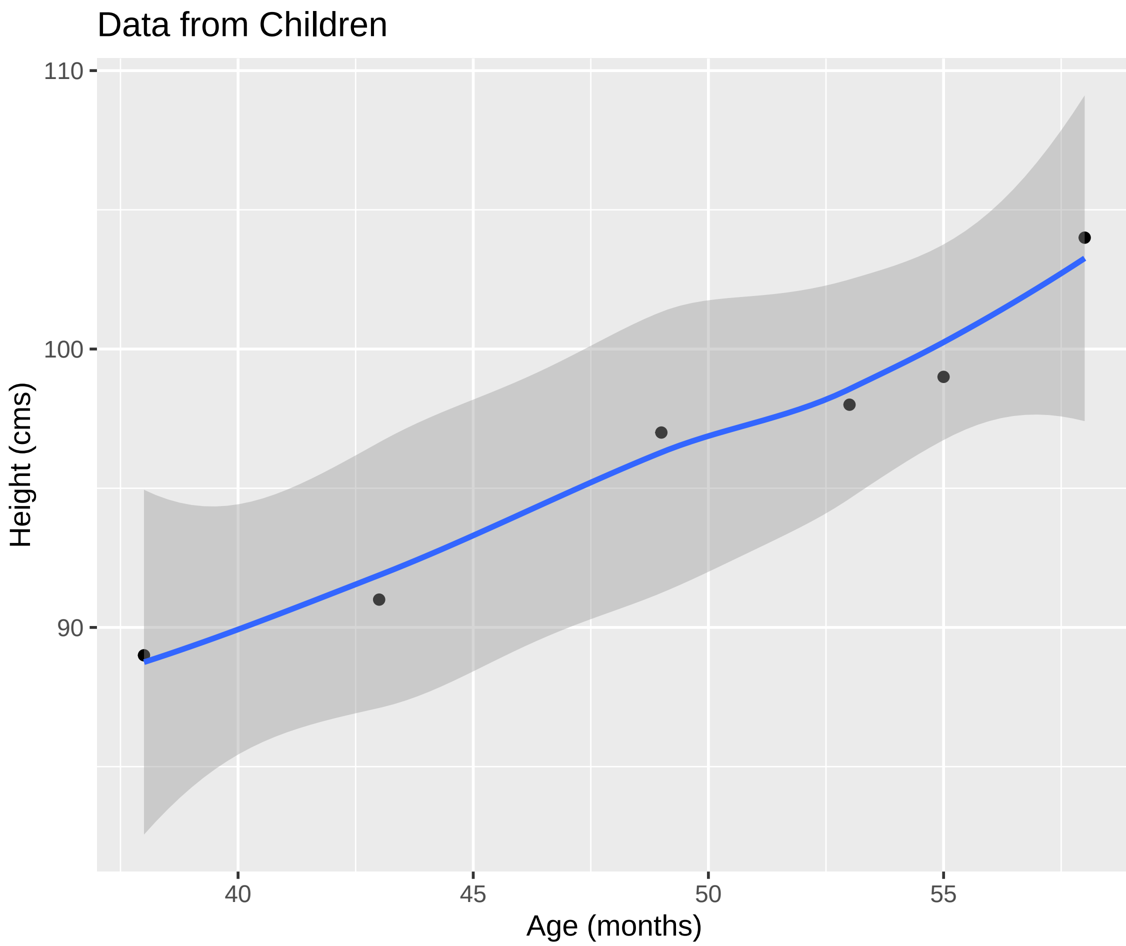
The smooth curve just added doesn't help us very much because it "joins the dots" and is, in a sense, not smooth enough. We can make it smoother using `span`:

```
ggplot(df, aes(x = x, y = y)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)")  +
  geom_smooth(span = 0.9)
```



The shaded area gives an indication of the reliability of our smooth curve. Here is an even smoother curve:

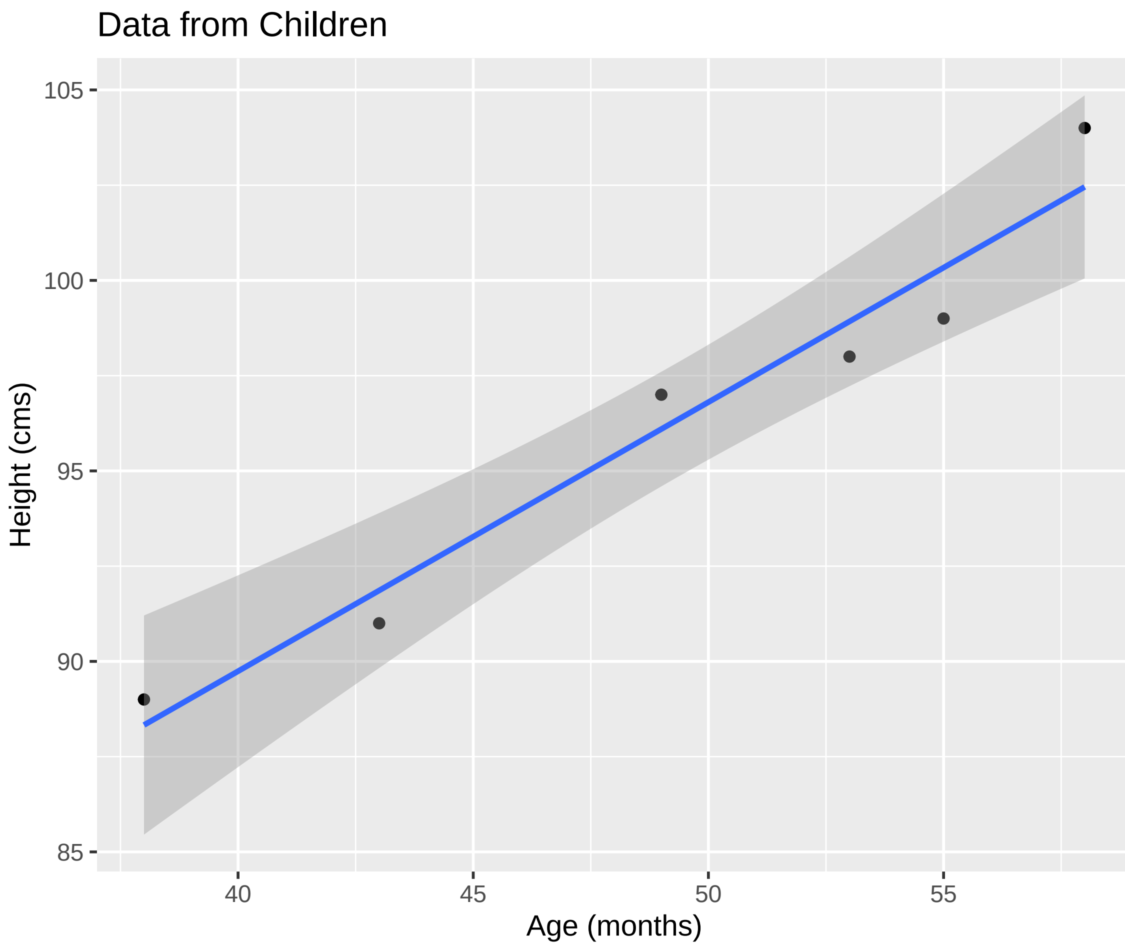
```
ggplot(df, aes(x = x, y = y)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)")  +
  geom_smooth(span = 1)
```



The fact that a straight line could be drawn within the shaded area suggests that a straight line model may be appropriate.

- or a **linear model** (simple linear regression model) with confidence bands

```
ggplot(df, aes(x = x, y = y)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)")   +
  geom_smooth(method = "lm")
```



We will study the straight line or simple linear regression model later.

2.6 An Important Aside on Factors

Let us say that we know that the first three children are male and the second three are female. Let's include this information in our data frame

```
sex <- c("M","M","M","F","F","F")
df <- data.frame(x, y, sex)
df
```

```
##    x   y sex
## 1 53  98   M
## 2 43  91   M
## 3 58 104   M
## 4 38  89   F
## 5 49  97   F
## 6 55  99   F
```

The variable `sex` comprises descriptions, not numbers. It is therefore an example of a **categorical variable**. It is good practice to define categorical variables as **factors**. The R structure `factor` allows flexible handling and labelling of categorical variables. Please consider the following:

```
sex_f <- factor(sex)
sex_f # Levels (i.e. categories) in alphabetical order
```

```
## [1] M M M F F F
## Levels: F M
```

```r
sex_f2 <- factor(sex, levels = c("M", "F"))
sex_f2 # User defined level order
```

```
## [1] M M M F F F
## Levels: M F
```

```r
sex_f <- factor(sex, labels = c("Female", "Male"))
# Standard level order, but with user defined labels
sex_f
```

```
## [1] Male   Male   Male   Female Female Female
## Levels: Female Male
```

It's often useful to look at the **str**ucture of an R object:

```r
sex_f
```

```
## [1] Male   Male   Male   Female Female Female
## Levels: Female Male
```

```r
str(sex_f)
```

```
##  Factor w/ 2 levels "Female","Male": 2 2 2 1 1 1
```

Note that internally "Female" is coded `1` and "Male" is coded `2`.

Let's redefine our data frame with `sex_f` and let's look at the structure of the resulting data frame:

```r
df <- data.frame(x, y, sex_f)
df
```

```
##    x   y  sex_f
## 1 53  98   Male
## 2 43  91   Male
## 3 58 104   Male
## 4 38  89 Female
## 5 49  97 Female
## 6 55  99 Female
```

```r
str(df)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ x    : num  53 43 58 38 49 55
##  $ y    : num  98 91 104 89 97 99
##  $ sex_f: Factor w/ 2 levels "Female","Male": 2 2 2 1 1 1
```

As factors are very important in what we will see later, here are some more examples.

- Data on the agreement of a group of people are recorded as 0 for "No" and 1 for "Yes":

```r
agreement <- c(0, 1, 1, 0, 0, 0, 0, 0, 0, 0)
agreement
```

```
##  [1] 0 1 1 0 0 0 0 0 0 0
```

- Let's turn this into a factor with appropriate labels:

```r
agreement_f <- factor(agreement, levels = c(0,1), labels = c("No", "Yes"))
agreement_f
```

```
##  [1] No  Yes Yes No  No  No  No  No  No  No 
## Levels: No Yes
```

or, slightly shorter

```
agreement_f <- factor(agreement, levels = 0:1, labels = c("No", "Yes"))
agreement_f
```

```
##  [1] No  Yes Yes No  No  No  No  No  No  No 
## Levels: No Yes
```

We can tabulate using `table`:

```
table(agreement_f)
```

```
## agreement_f
##  No Yes 
##   8   2
```

- With a different order

```
agreement_f <- factor(agreement, levels = c(1,0), labels = c("Yes", "No"))
agreement_f
```

```
##  [1] No  Yes Yes No  No  No  No  No  No  No 
## Levels: Yes No
```

```
table(agreement_f)
```

```
## agreement_f
## Yes  No 
##   2   8
```

2.7 Back to Using ggplot2

Let's recall the contents and structure of our data frame `df`:

```
df
```

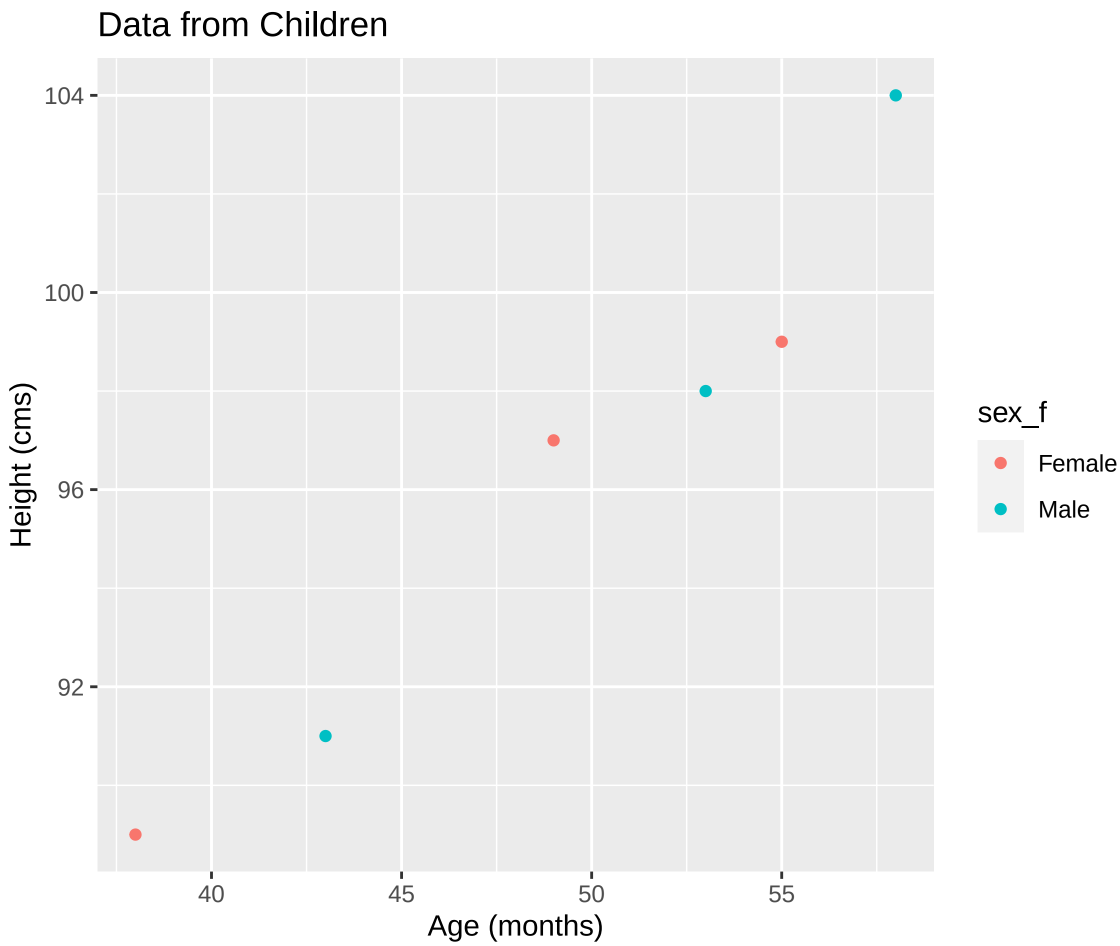
```
##    x   y  sex_f
## 1 53  98   Male
## 2 43  91   Male
## 3 58 104   Male
## 4 38  89 Female
## 5 49  97 Female
## 6 55  99 Female
```

```
str(df)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ x    : num  53 43 58 38 49 55
##  $ y    : num  98 91 104 89 97 99
##  $ sex_f: Factor w/ 2 levels "Female","Male": 2 2 2 1 1 1
```

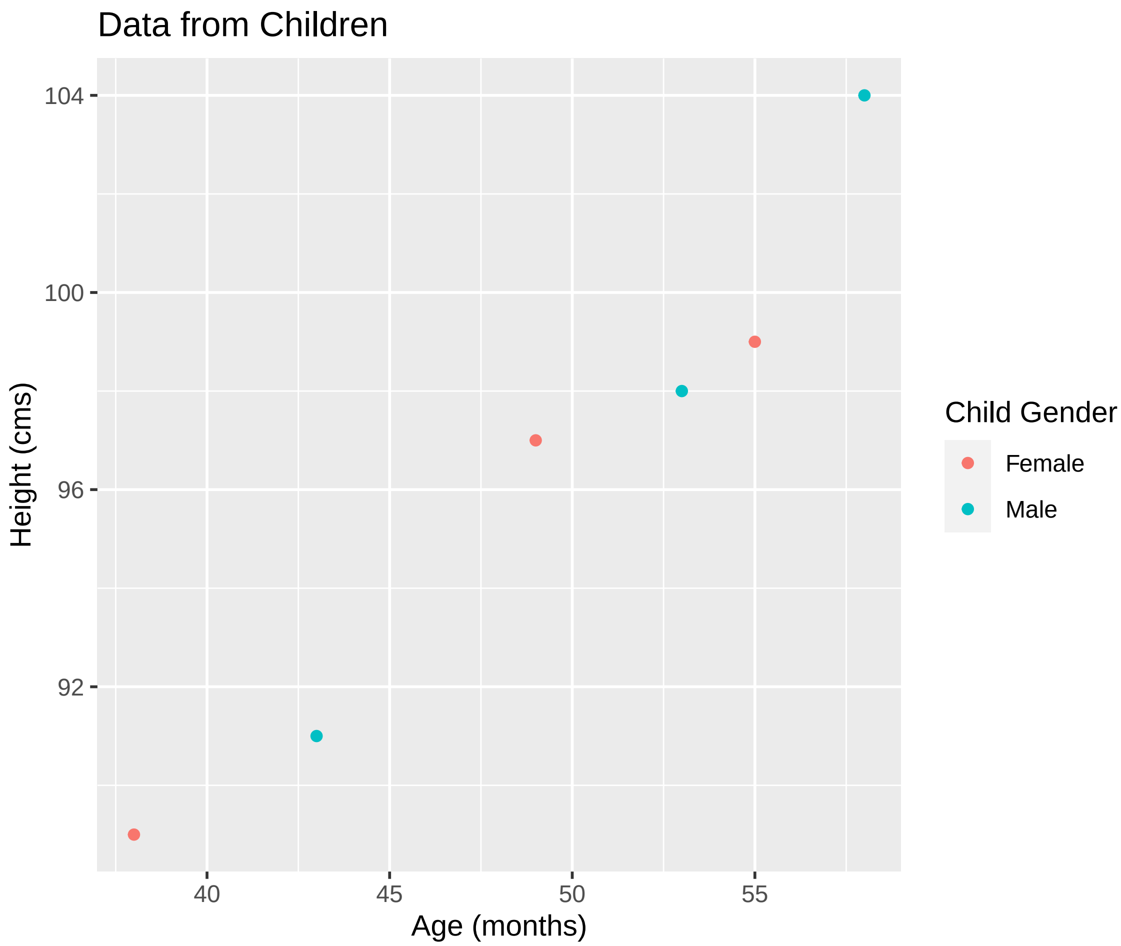
- We can use a different plotting **colour** for each sex (now defined as a factor):

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)")
```



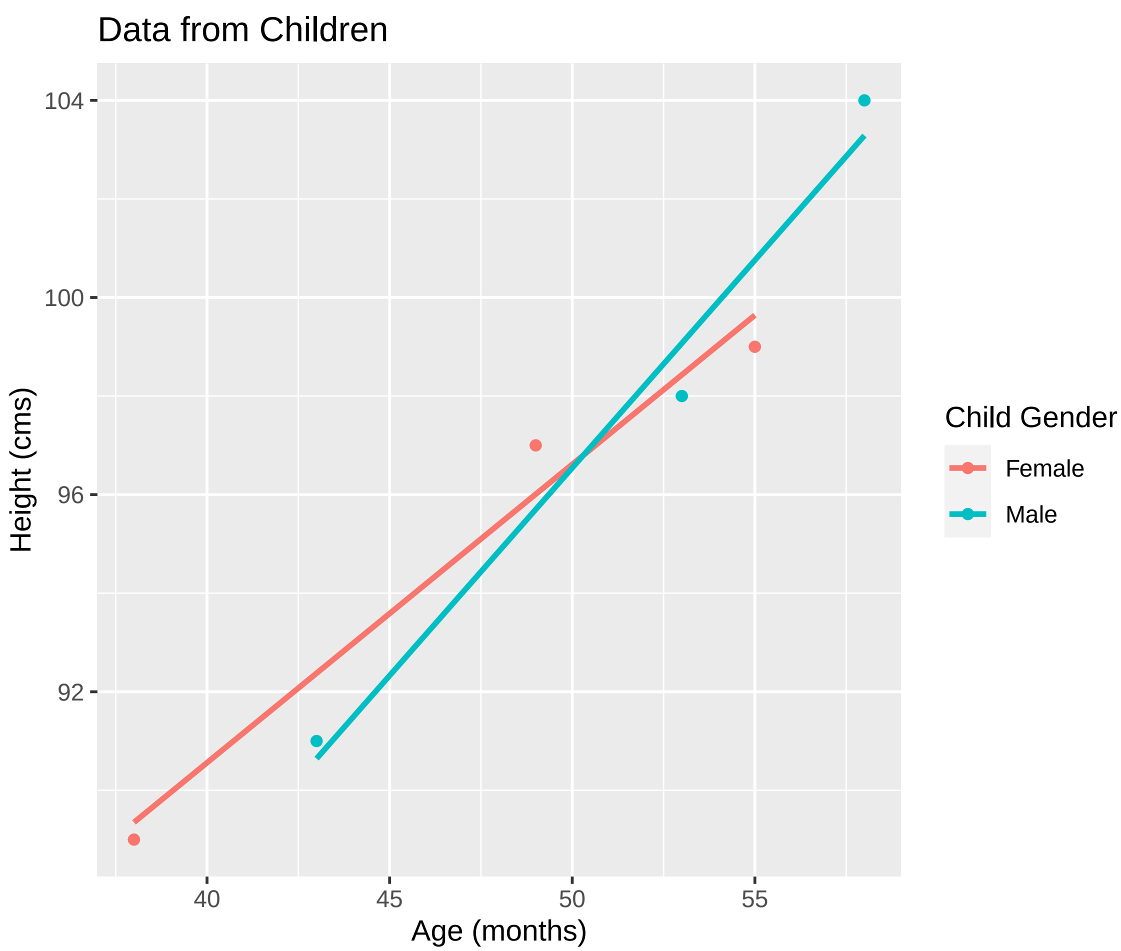
We can specify the legend title by labelling the aesthetic the defines it:

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") # Legend title
```



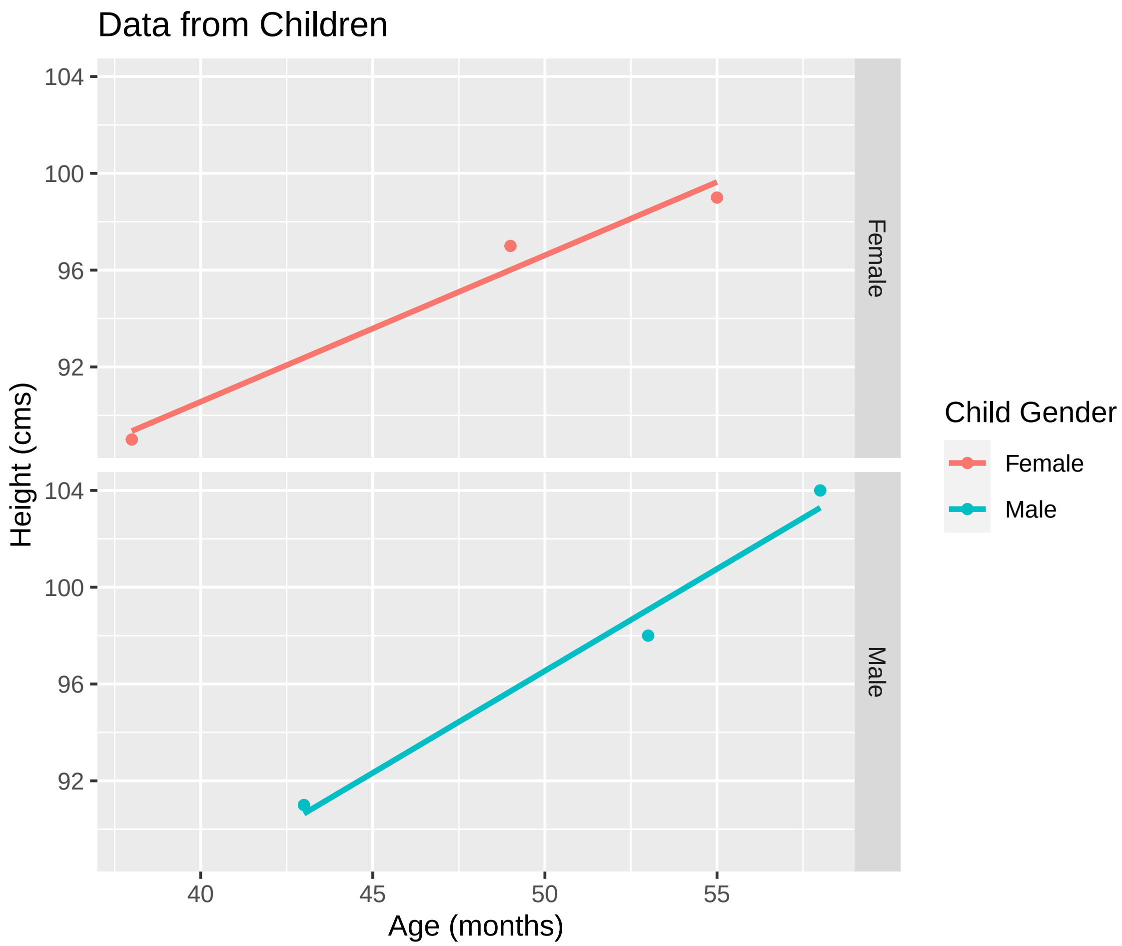
- We can include a different straight **line** for each sex (here without the confidence bands)

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender")  +
  geom_smooth(method = "lm", se = FALSE) # No standard errors or confidence bands
```



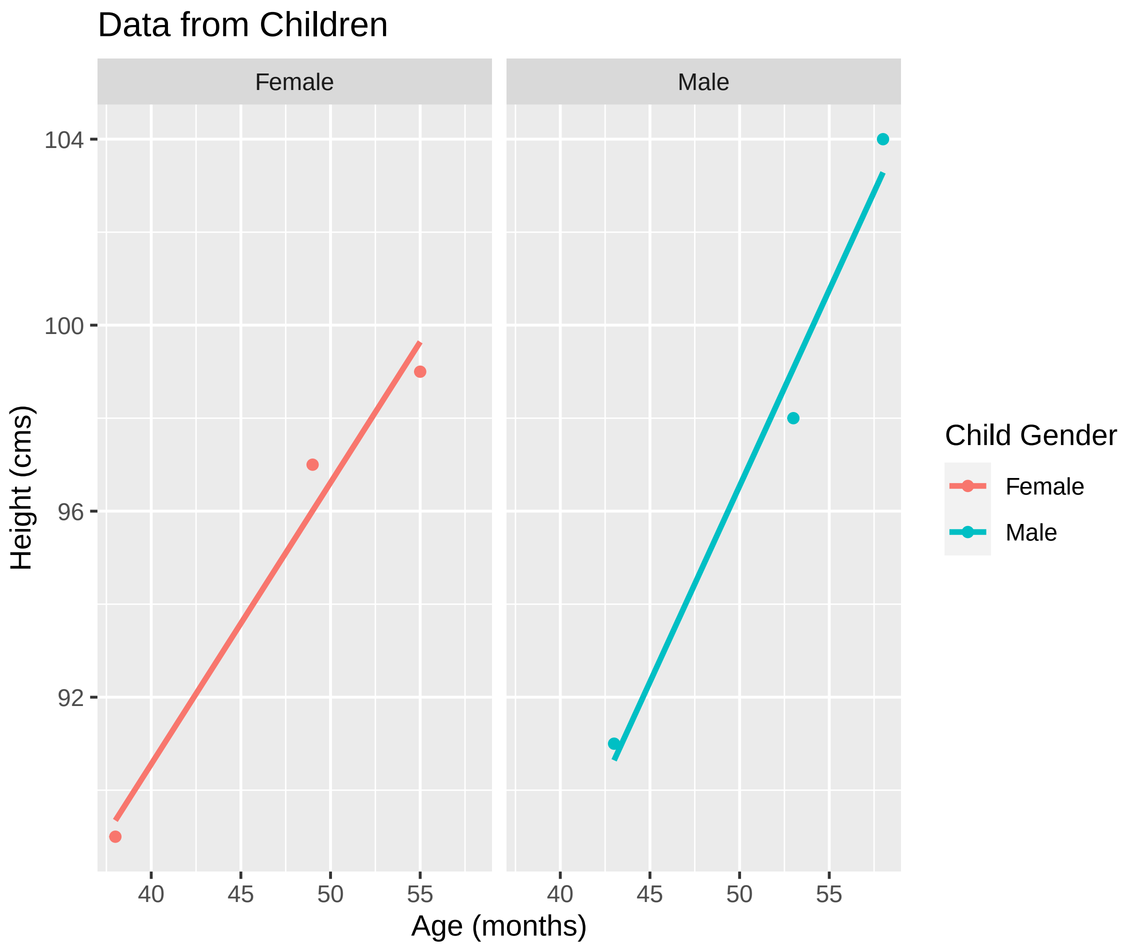
- We can even produce a separate plot for each sex using a so called **facet_grid**

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_grid(sex_f ~ .)
```



Can you see the difference between the above plot and this one?

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_grid(. ~ sex_f)
```



To push these ideas further, let's say that the children come from three towns, Town 1, Town 2 and Town 3:

```
town <- c("Town 1", "Town 1", "Town 2", "Town 2", "Town 3", "Town 3")
town
```

```
## [1] "Town 1" "Town 1" "Town 2" "Town 2" "Town 3" "Town 3"
```

```
town_f <- factor(town)
town_f
```

```
## [1] Town 1 Town 1 Town 2 Town 2 Town 3 Town 3
## Levels: Town 1 Town 2 Town 3
```

```
df <- data.frame(x, y, sex_f, town_f)
df
```

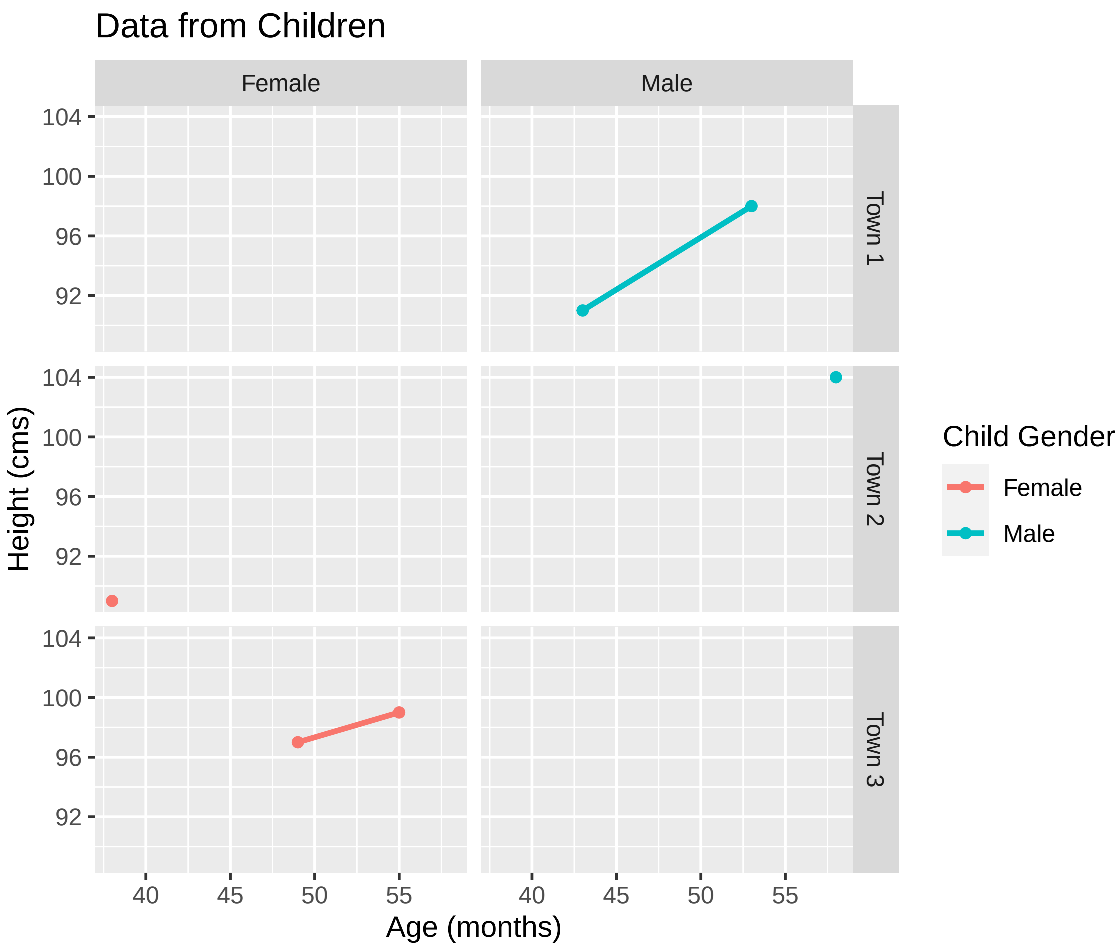
```
##    x   y  sex_f town_f
## 1 53  98   Male Town 1
## 2 43  91   Male Town 1
## 3 58 104   Male Town 2
## 4 38  89 Female Town 2
## 5 49  97 Female Town 3
## 6 55  99 Female Town 3
```

```
str(df)
```

```
## 'data.frame':    6 obs. of  4 variables:
```

```
##  $ x      : num  53 43 58 38 49 55
##  $ y      : num  98 91 104 89 97 99
##  $ sex_f : Factor w/ 2 levels "Female","Male": 2 2 2 1 1 1
##  $ town_f: Factor w/ 3 levels "Town 1","Town 2",..: 1 1 2 2 3 3
```

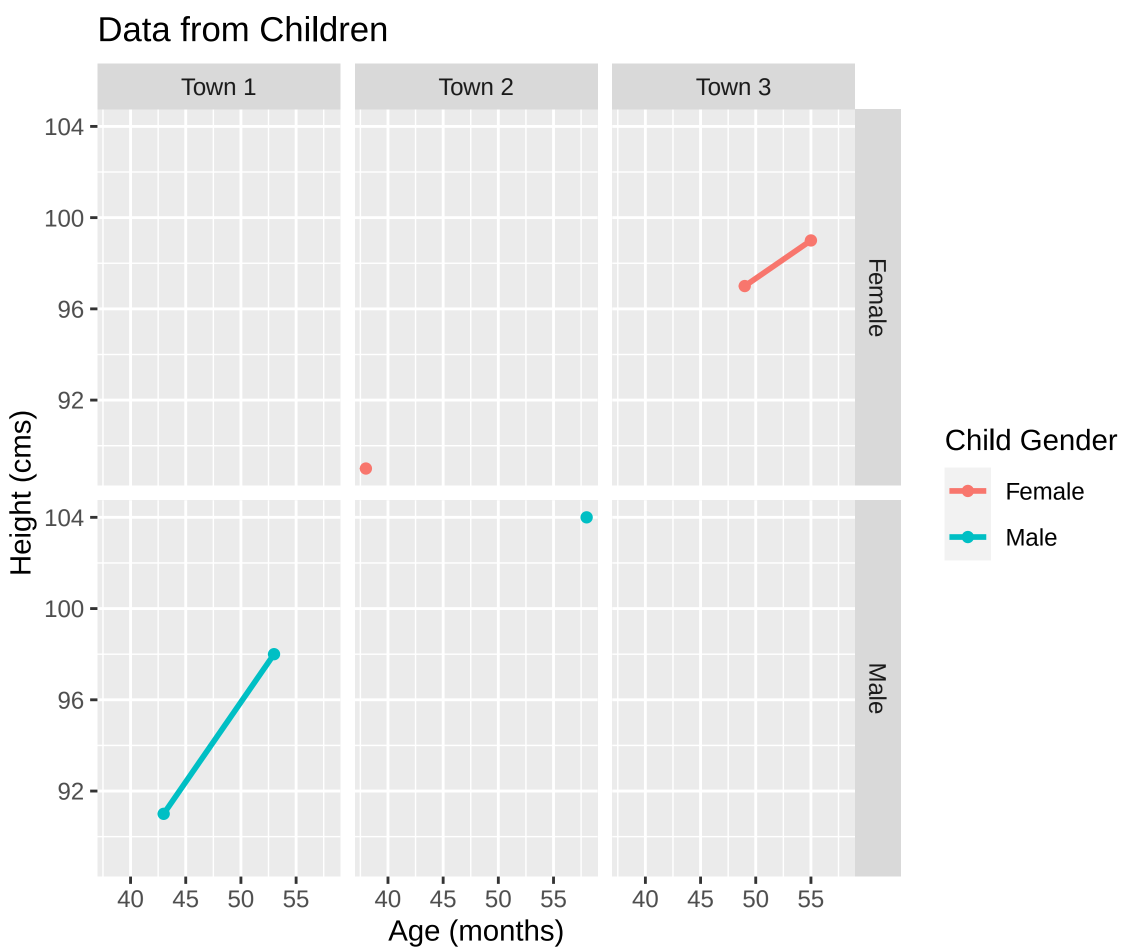
```
#
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_grid(town_f ~ sex_f)
```



Such a plot is a little silly for such a small data set, but the idea of its construction is clear.

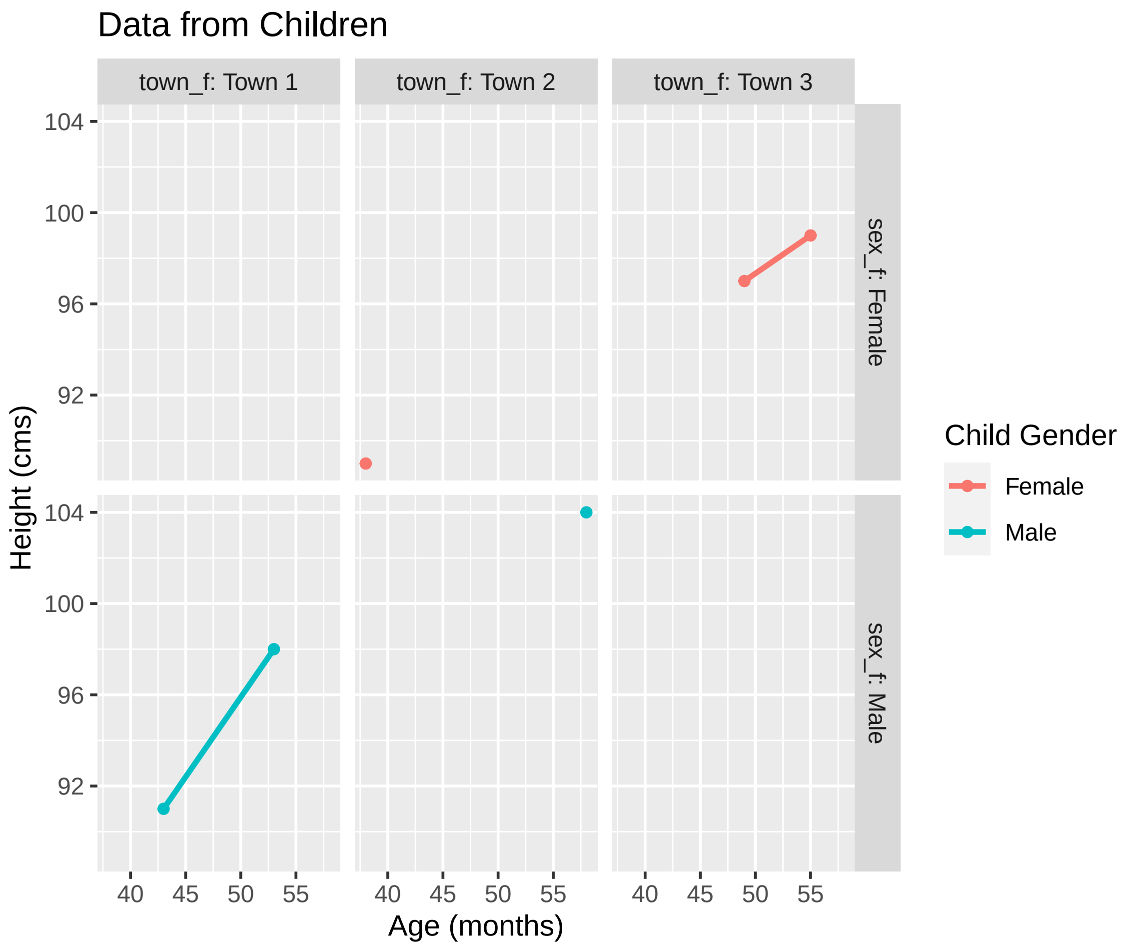
Please also consider:

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_grid(sex_f ~ town_f)
```



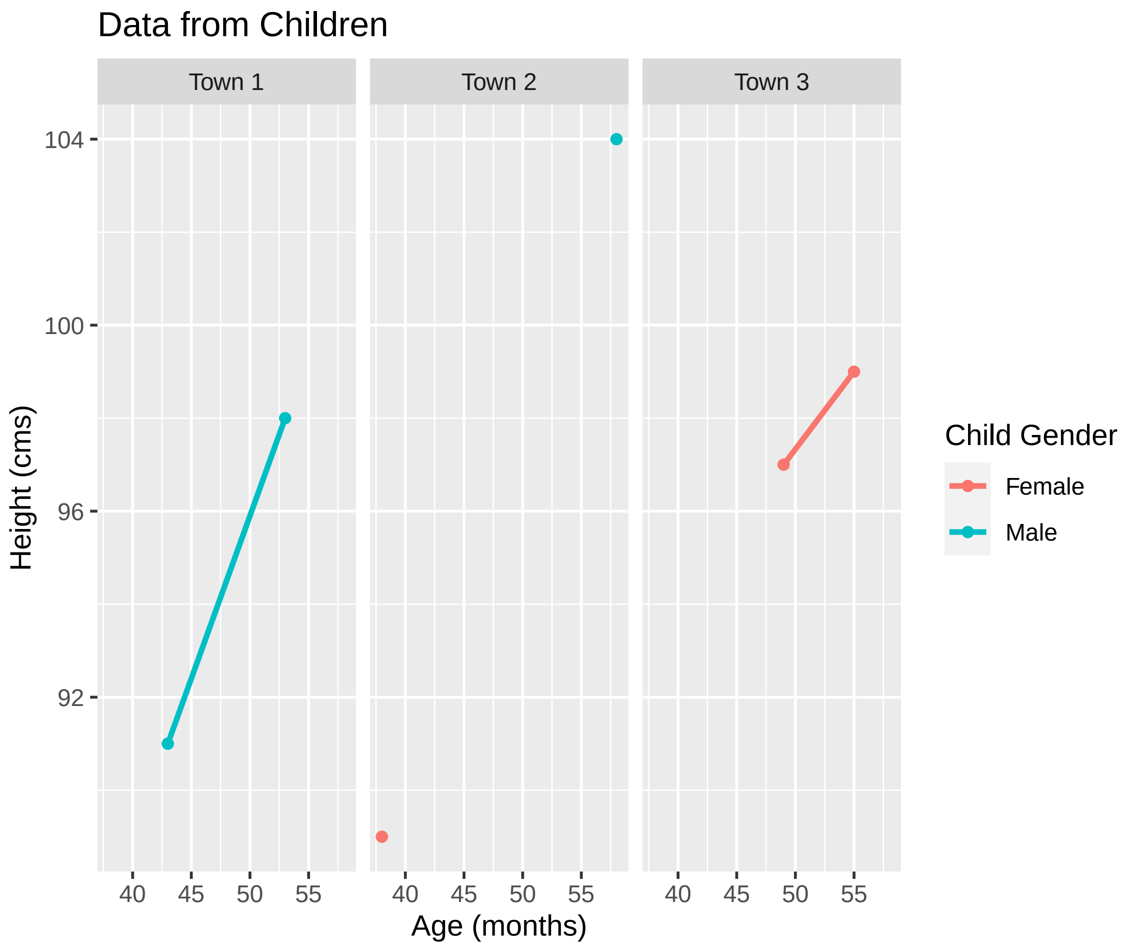
We can also make the facet labels more explicit by specifying a `labeller` in `facet_grid`:

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_grid(sex_f ~ town_f, labeller = label_both) # Make facet labels more explicit
```



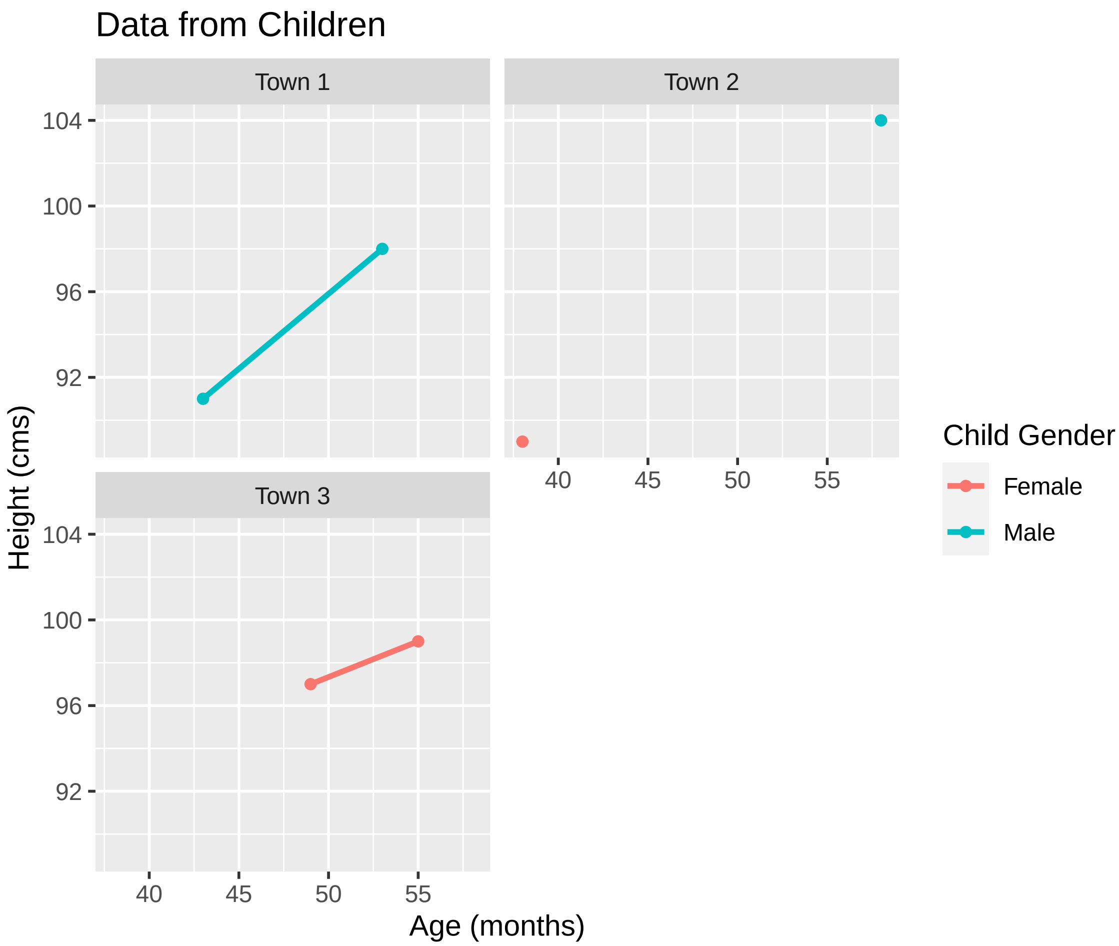
We can also use `facet_wrap` as these small scale examples shows:

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f)
```

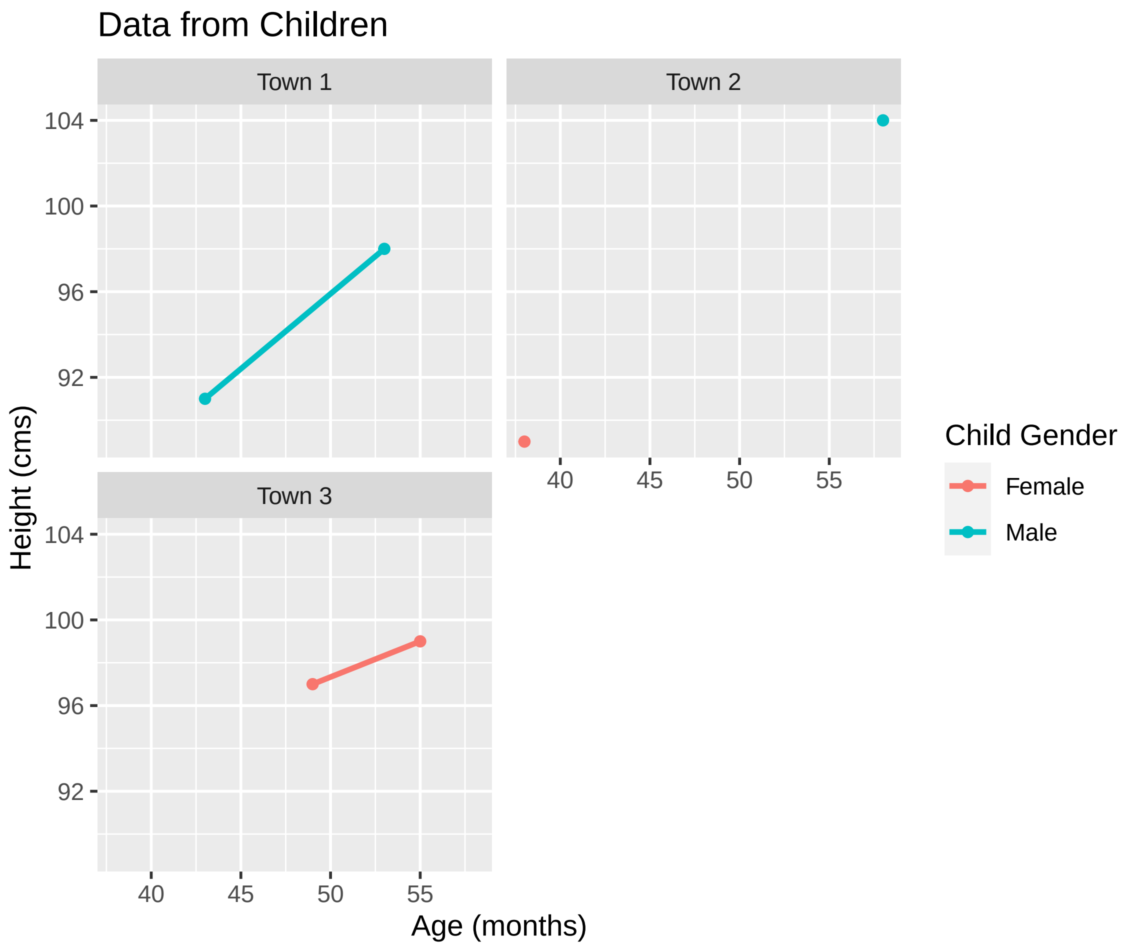


As always, please ask if you don't understand what is going on.

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, ncol = 2) # Number of columns specified
```

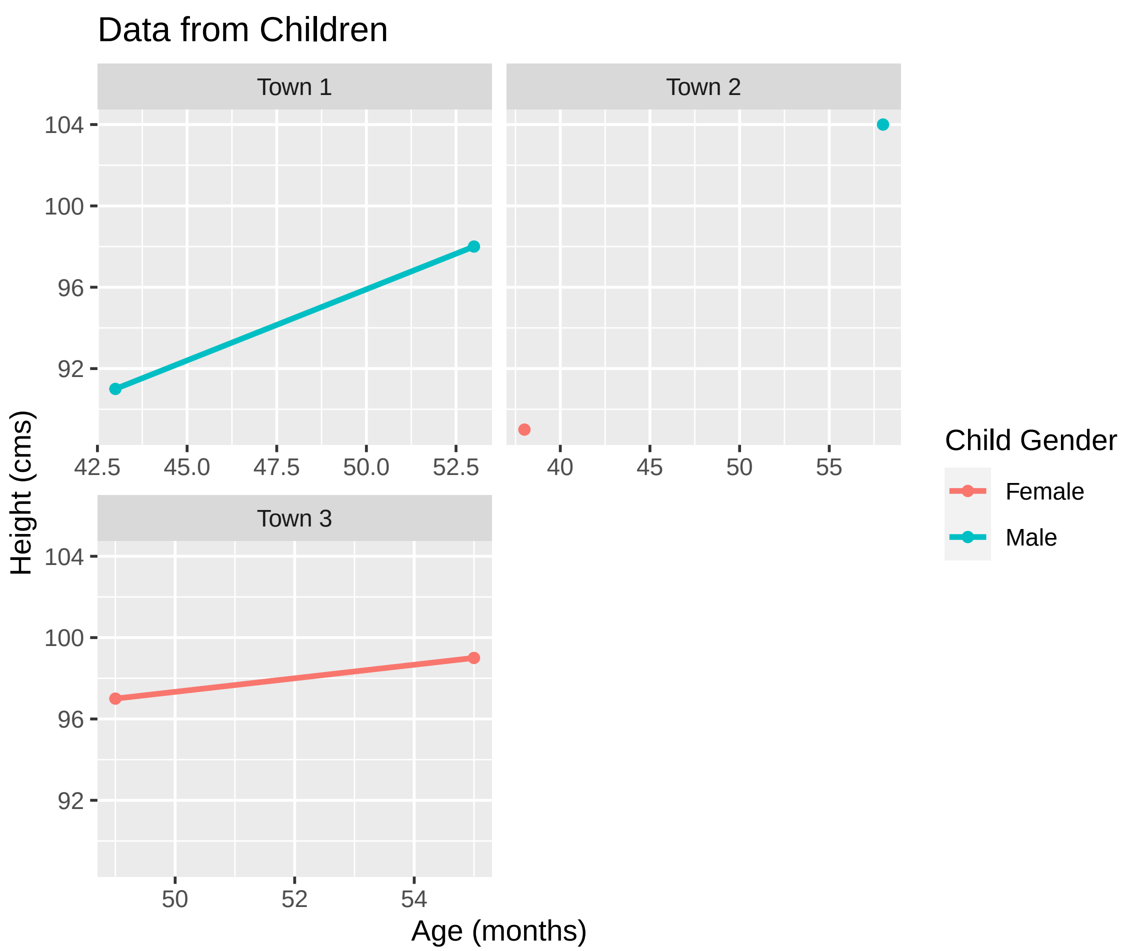



```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, nrow = 2) # Number of rows specified; here the same plot
```



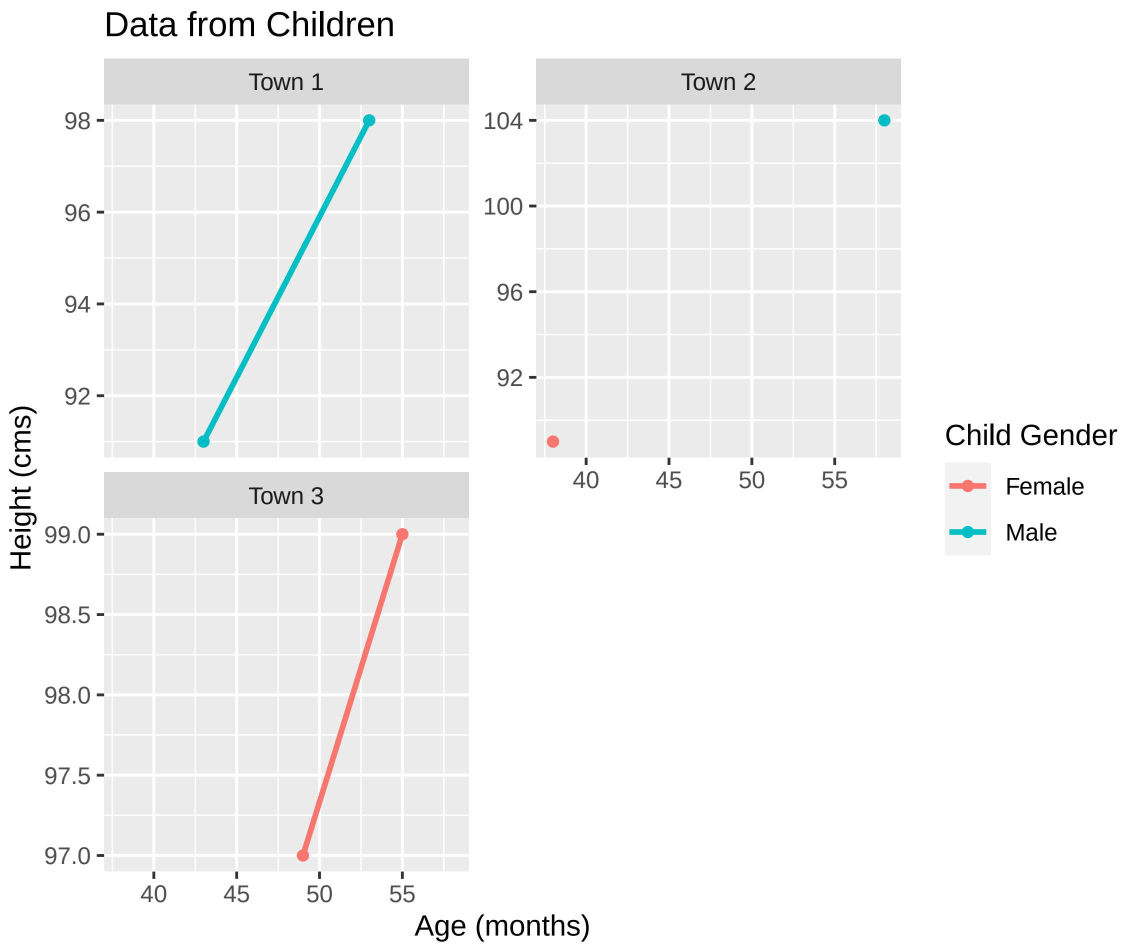
We don't have to use the same scales for each plot:

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, nrow = 2, scales = "free_x")
```



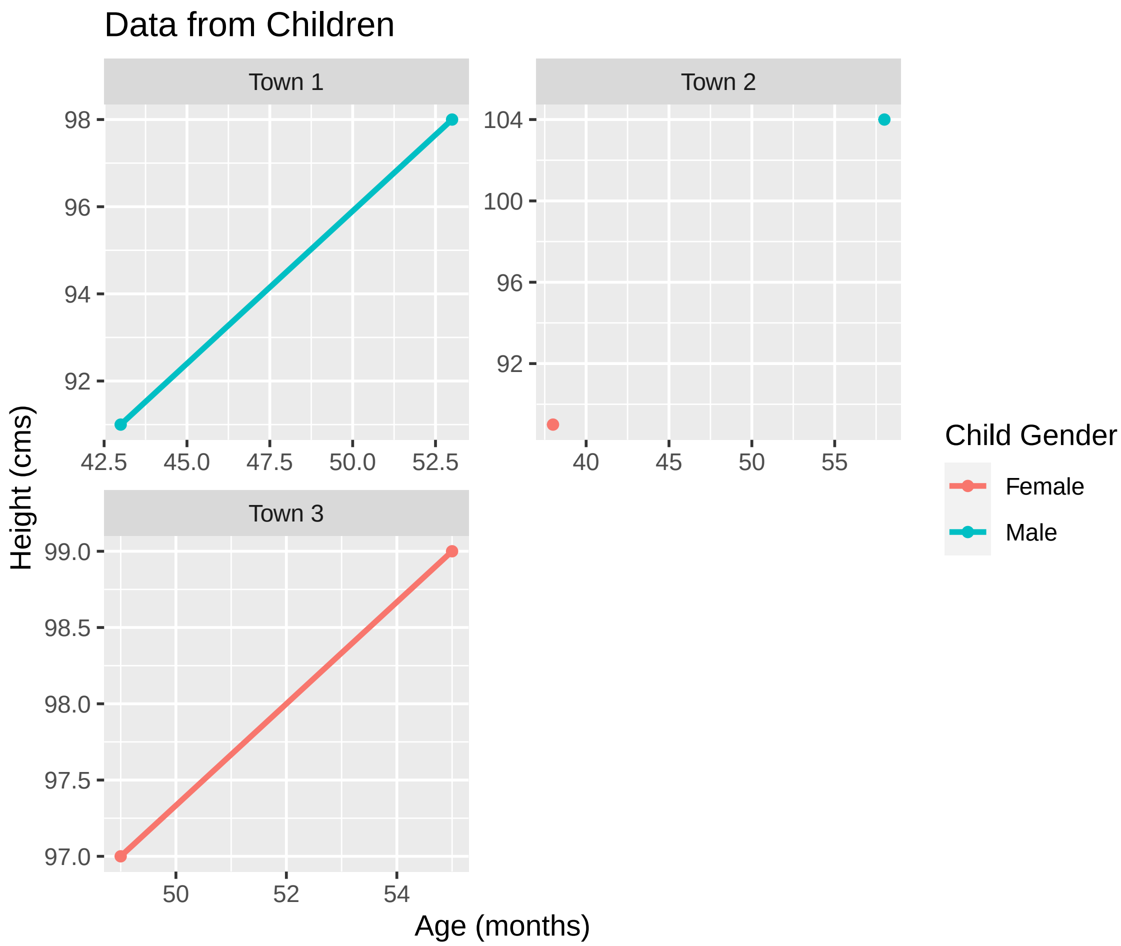

```
# Free (i.e. different) scales on the x-axis
```

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, nrow = 2, scales = "free_y")
```




```
# Free (i.e. different) scales on the y-axis
```

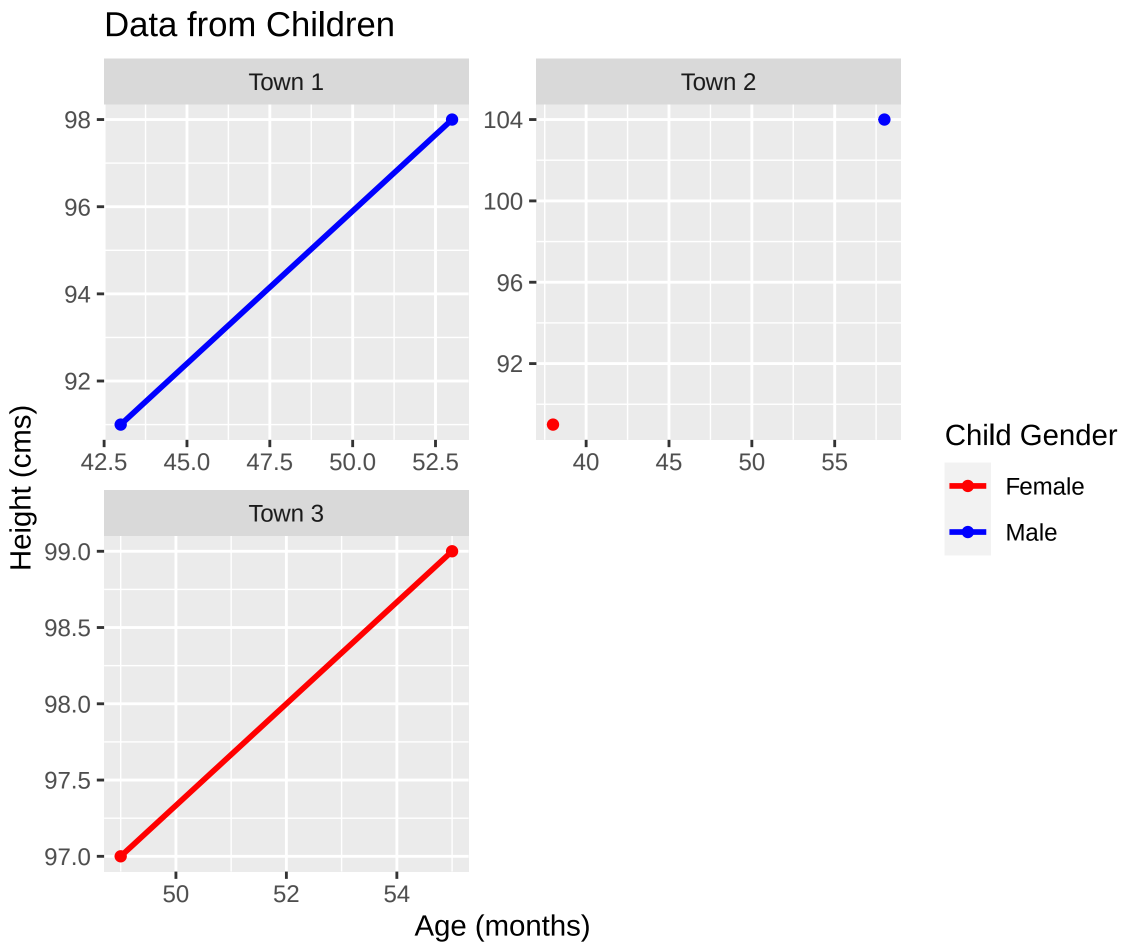
```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, nrow = 2, scales = "free")
```




```
# Free (i.e. different) scales on both axes
```

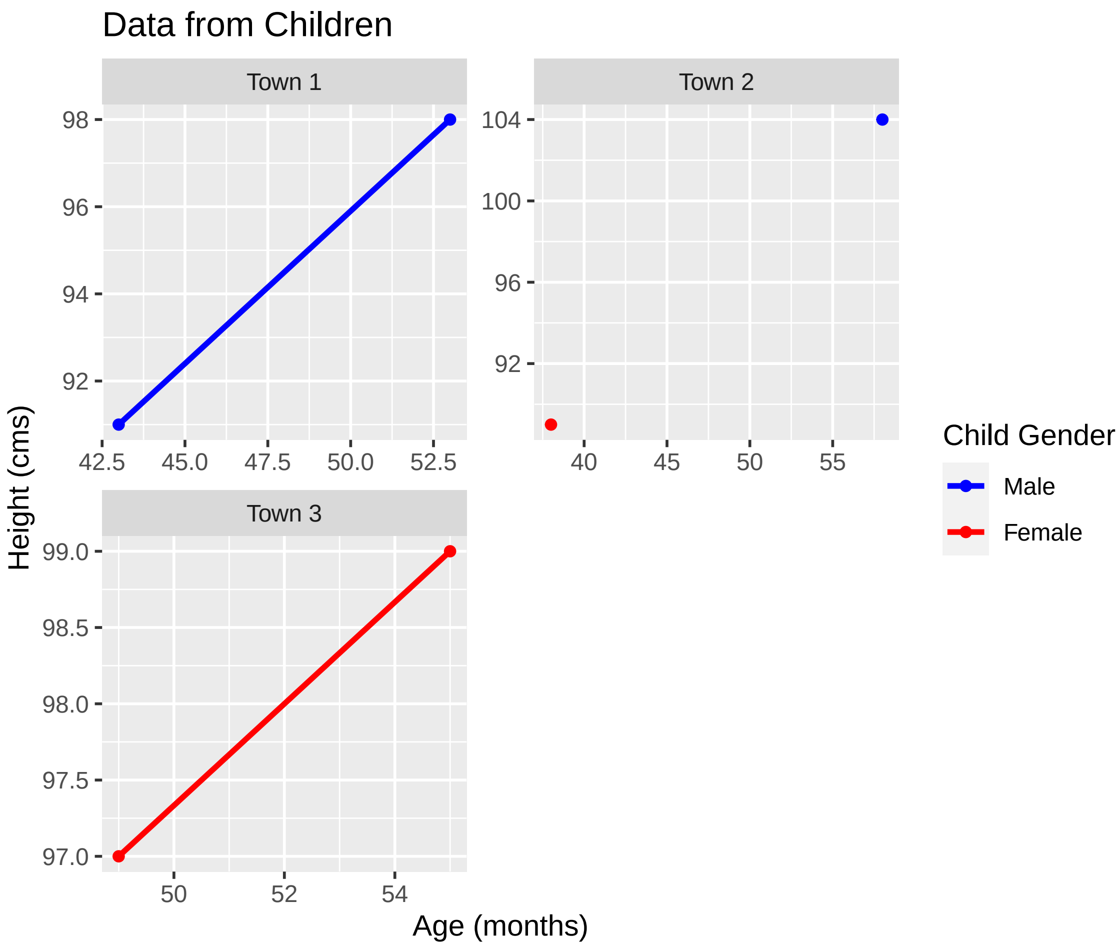
It's also possible to change the default colours used for each sex:

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, nrow = 2, scales = "free") +
  scale_colour_manual(values = c("Female" = "red", "Male" = "blue"))  # Assign the colours manually
```



and to reverse the order of the legend scale:

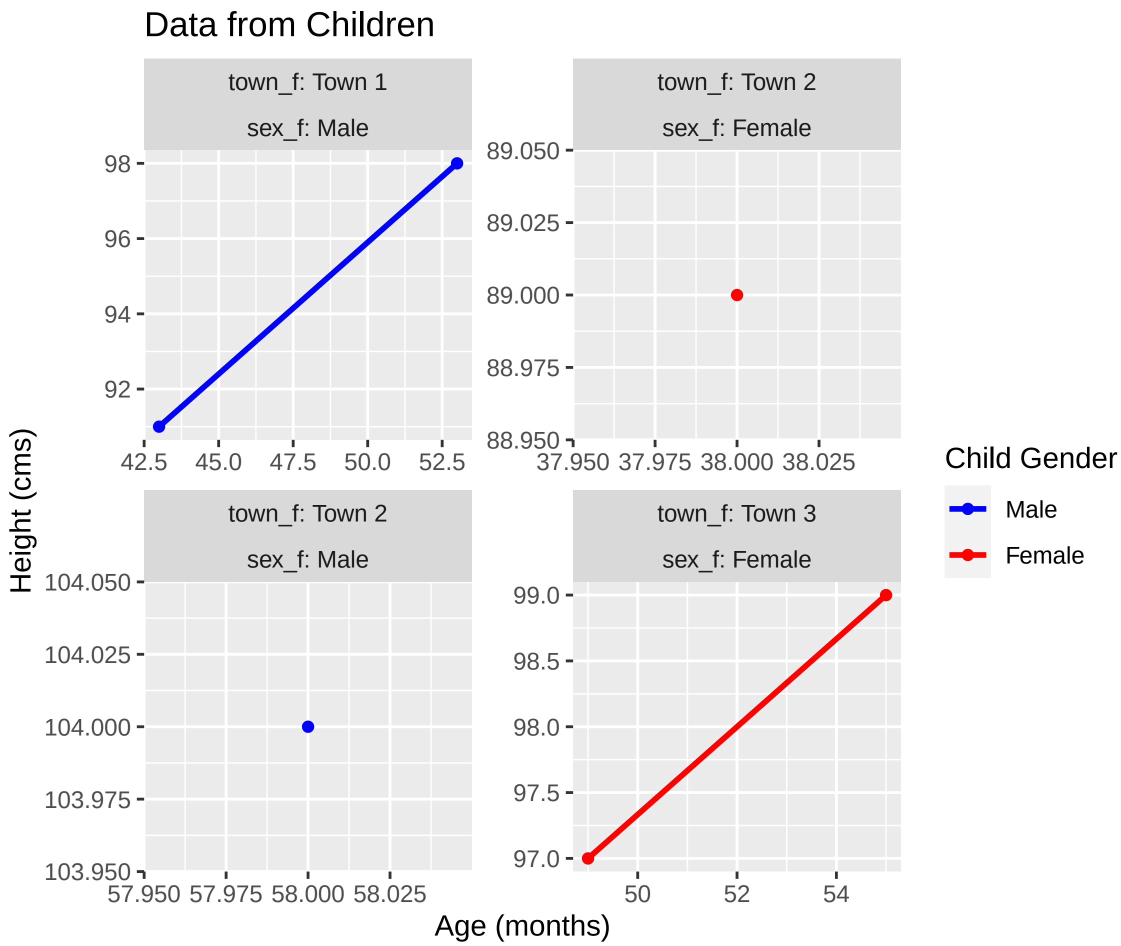
```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, nrow = 2, scales = "free") +
  scale_colour_manual(guide = guide_legend(reverse=TRUE), # Reverse the scale
                      values = c("Female" = "red", "Male" = "blue"))
```



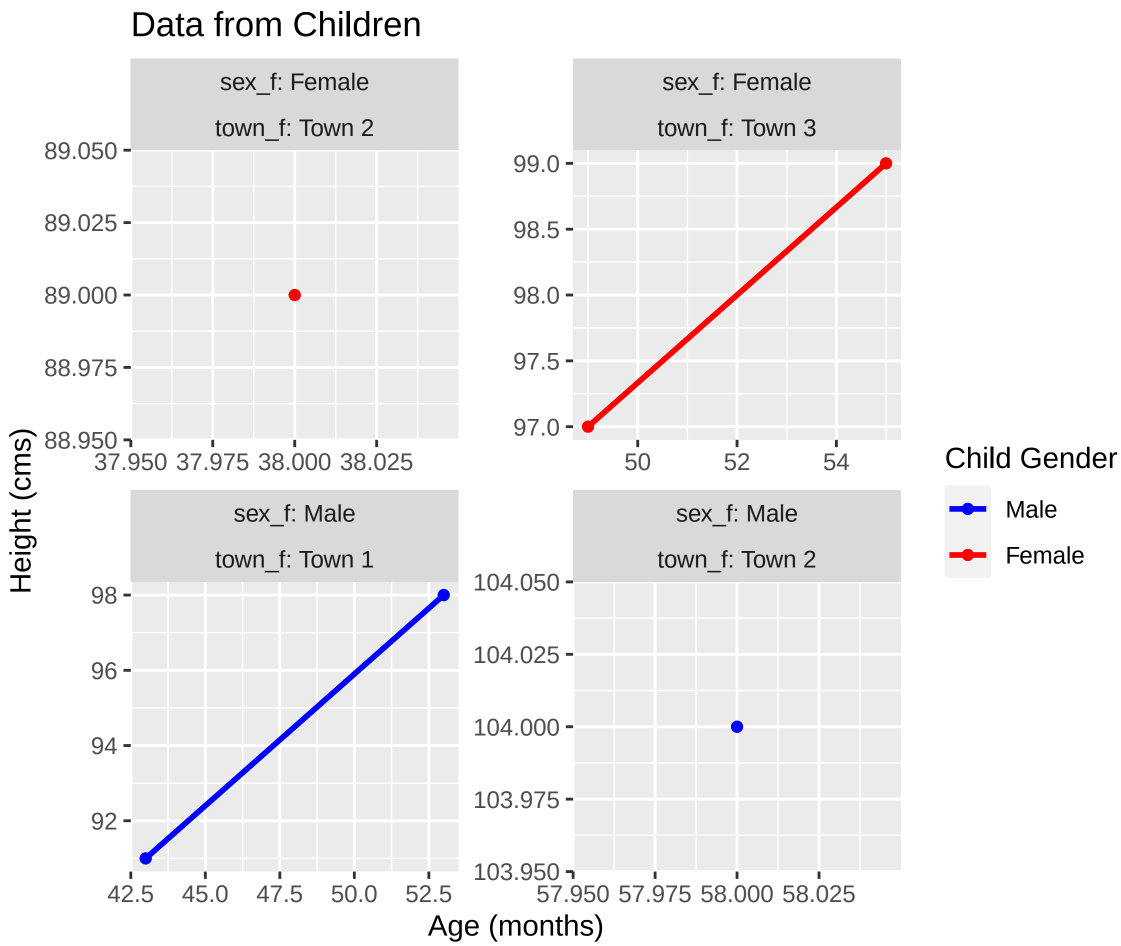
Equivalently, we could have redefined the levels of the factor `sex_f`.

Here are a couple of other examples, illustrating various uses of `facet_wrap`:

```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f + sex_f, nrow = 2, scales = "free", labeller = label_both) +
    # Wrap the plots around according to town_f and then sex_f
    # Make facet labels more explicit
  scale_colour_manual(guide = guide_legend(reverse=TRUE), # Reverse the scale
                      values = c("Female" = "red", "Male" = "blue"))
```




```r
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(c("sex_f", "town_f"), nrow = 2, scales = "free", labeller = label_both) +
    # Another way of specifying the facetting variables
    # Make facet labels more explicit
  scale_colour_manual(guide = guide_legend(reverse=TRUE), # Reverse the scale
                      values = c("Female" = "red", "Male" = "blue"))
```

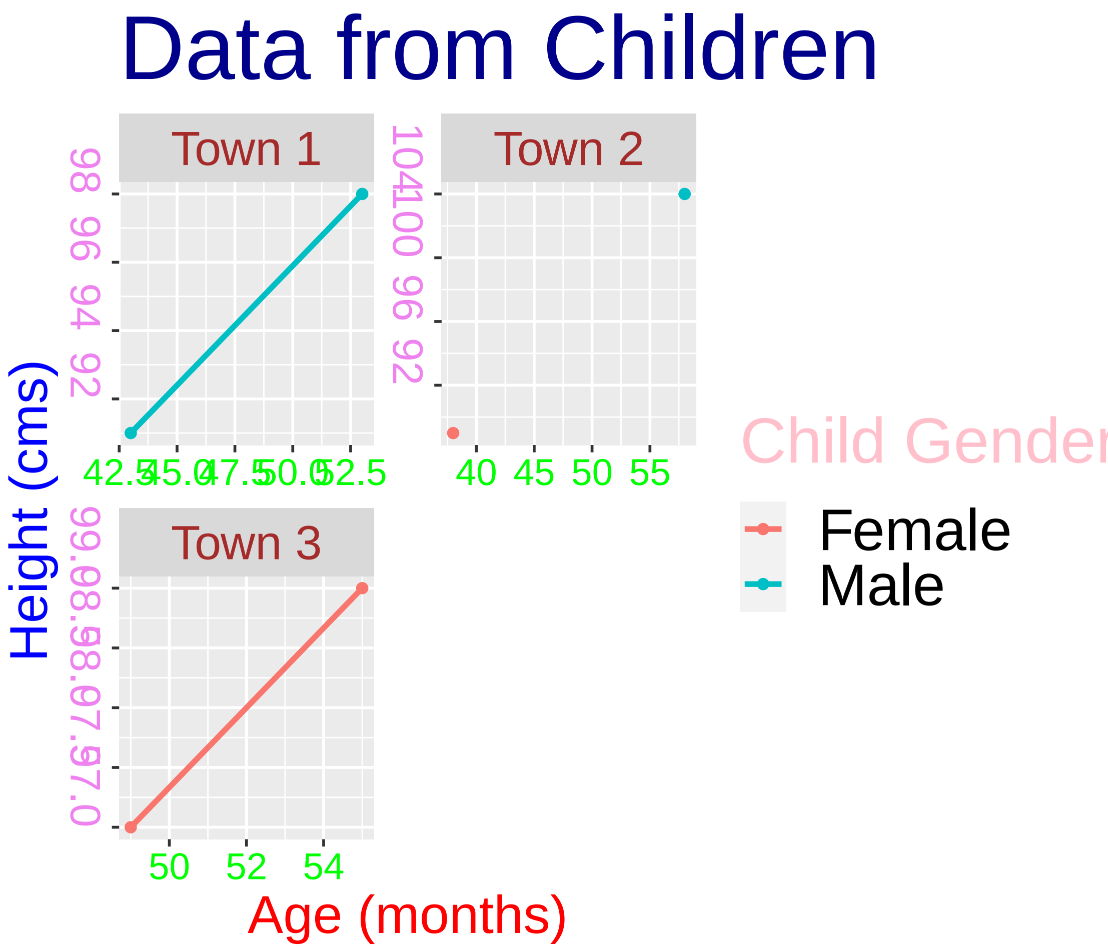


We hope that this shows some of the power of **ggplot2**. We'll see more of that power on bigger data soon.

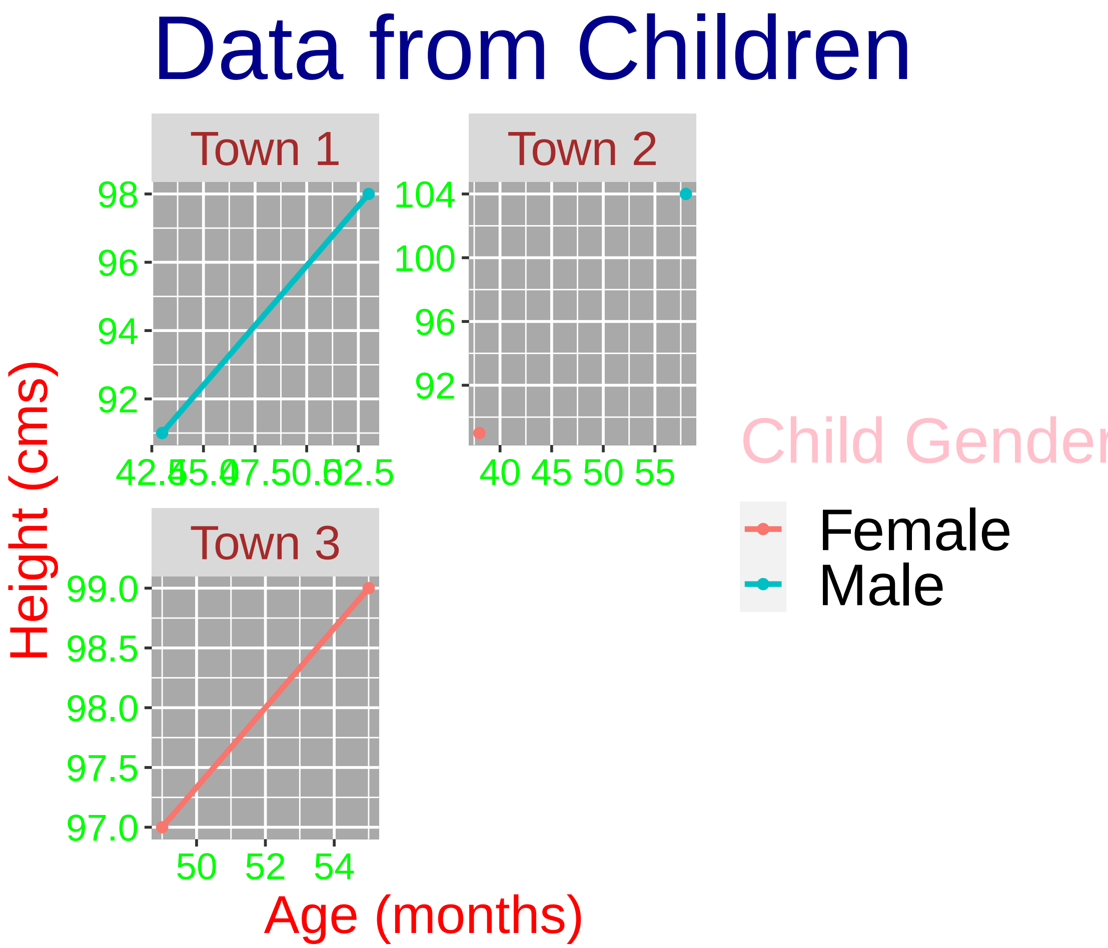
2.8 Customizing ggplot2 Plots Using Themes

We can easily customize `ggplot2` plots by adding a specified theme. We illustrate this using examples, which hopefully are self-explanatory. Please experiment and ask if anything is not clear. Please note that we are not suggesting that you should customize your plots in this way. Our aim is to show you part of what is possible.

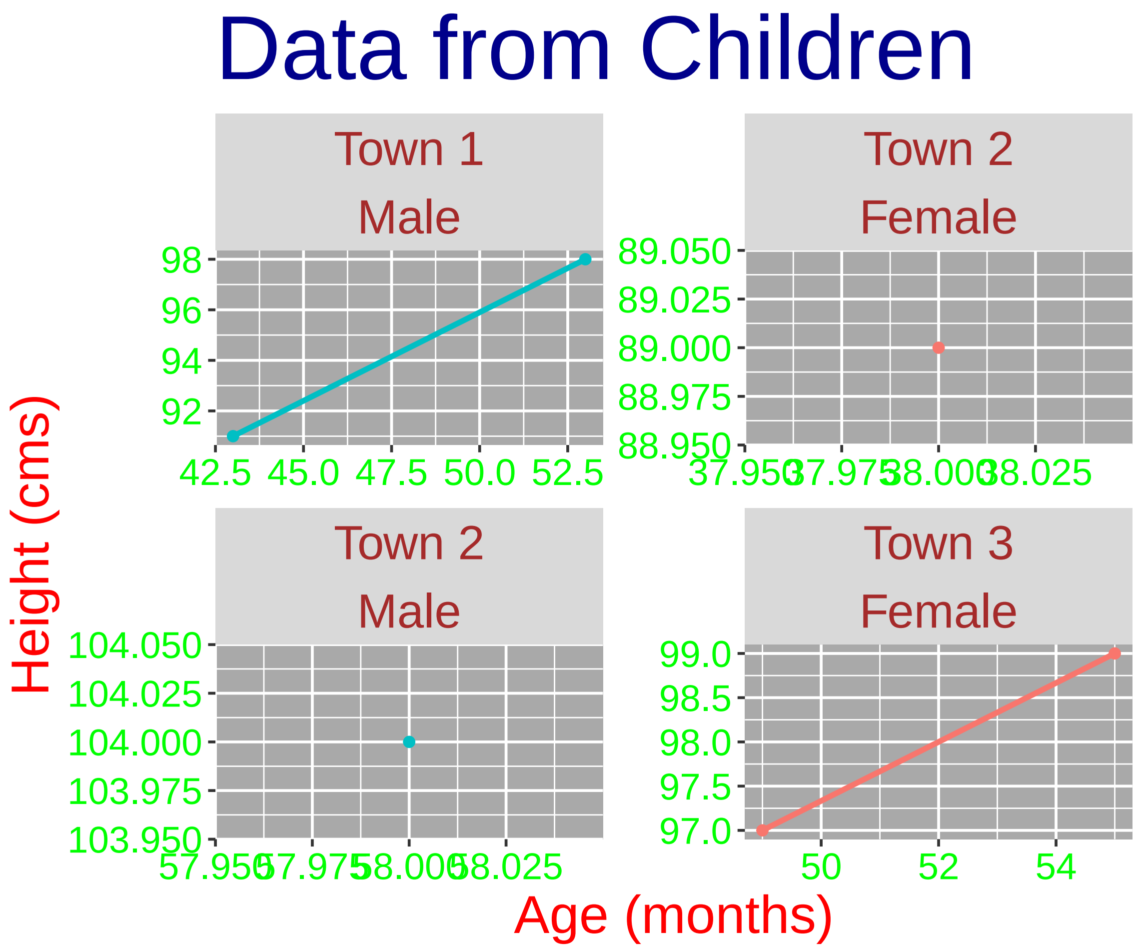
```
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, nrow = 2, scales = "free") +
  theme(title = element_text(size = 28, color = "darkblue"), # Customize the main title
      axis.title.x = element_text(size = 20, color = "red"), # Customize the x-axis title
        axis.text.x = element_text(size = 14, color = "green"), # Customize the x-axis text
        axis.title.y = element_text(size = 20, color = "blue"), # Customize the y-axis title
        axis.text.y = element_text(size = 16, color = "violet", angle = 270),
                                                        # Customize the y-axis text
        legend.title = element_text(size = 24, color = "pink"), # Customize the legend title
        legend.text = element_text(size = 22), # Customize the legend text
        strip.text.x = element_text(size = 18, color = "brown") # Customize the facet text
        )
```




```r
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ town_f, nrow = 2, scales = "free") +
  theme(title = element_text(size = 28, color = "darkblue"), # Customize the main title
        axis.title = element_text(size = 20, color = "red"), # Customize the titles of both axes
        axis.text = element_text(size = 14, color = "green"), # Customize the text of both axes
        legend.title = element_text(size = 24, color = "pink"), # Customize the legend title
        legend.text = element_text(size = 22), # Customize the legend text
        strip.text.x = element_text(size = 18, color = "brown"), # Customize the facet text
        panel.background = element_rect(fill = 'darkgrey') # Customize the panel background
        )
```




```r
ggplot(df, aes(x = x, y = y, col = sex_f)) + geom_point() +
  ggtitle("Data from Children") +
  labs(x = "Age (months)", y = "Height (cms)", col = "Child Gender") +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(town_f ~ sex_f, nrow = 2, scales = "free") +
  theme(title = element_text(size = 28, color = "darkblue"), # Customize the main title
        axis.title = element_text(size = 20, color = "red"), # Customize the titles of both axes
        axis.text = element_text(size = 14, color = "green"), # Customize the text of both axes
        strip.text.x = element_text(size = 18, color = "brown"), # Customize the facet text
        panel.background = element_rect(fill = 'darkgrey'), # Customize the panel background
        legend.position = "none") # No legend
```



2.9 More Details

The Data Visualization Cheat Sheet provides full coverage of `ggplot2` and is available from

https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

There is an enormous amount of material on the web about `ggplot2`.
`ggplot2` is accompanied by the book

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis.* Springer New York.

which will soon have a second edition.

2.10 The `ggrepel` Package and Bulit in Themes

The `ggplot2` function `geom_text` allows you to add text to a plot. Here is an illustration based on the `mtcars` data supplied with `ggplot2`. The `mtcars` data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models); please see the help file. The data frame `mtcars` has row names which correspond to the names of the cars:

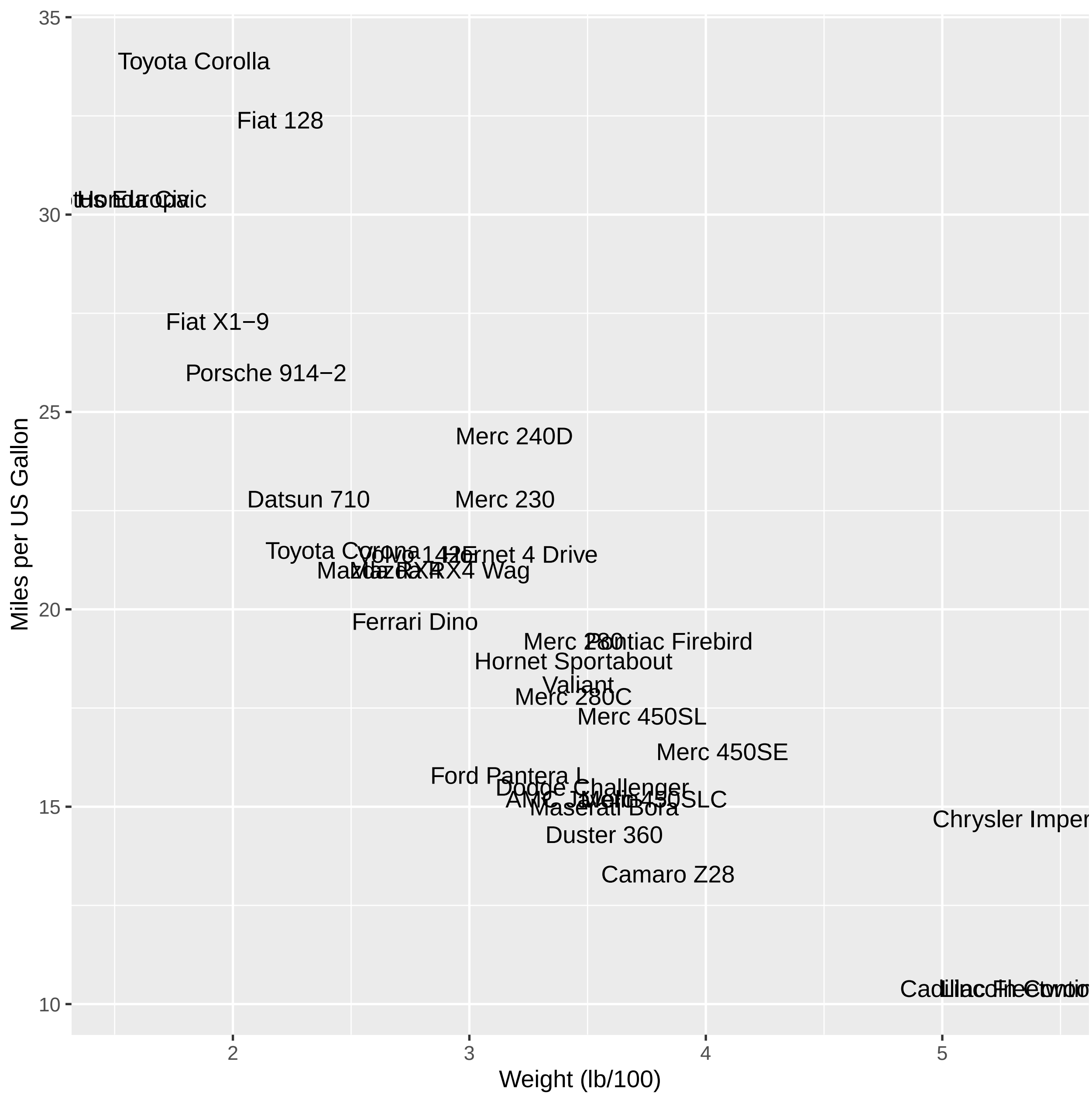
```
rownames(mtcars)
```

```
##  [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
##  [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
##  [7] "Duster 360"          "Merc 240D"           "Merc 230"
## [10] "Merc 280"            "Merc 280C"           "Merc 450SE"
```

```
## [13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
## [22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
## [25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

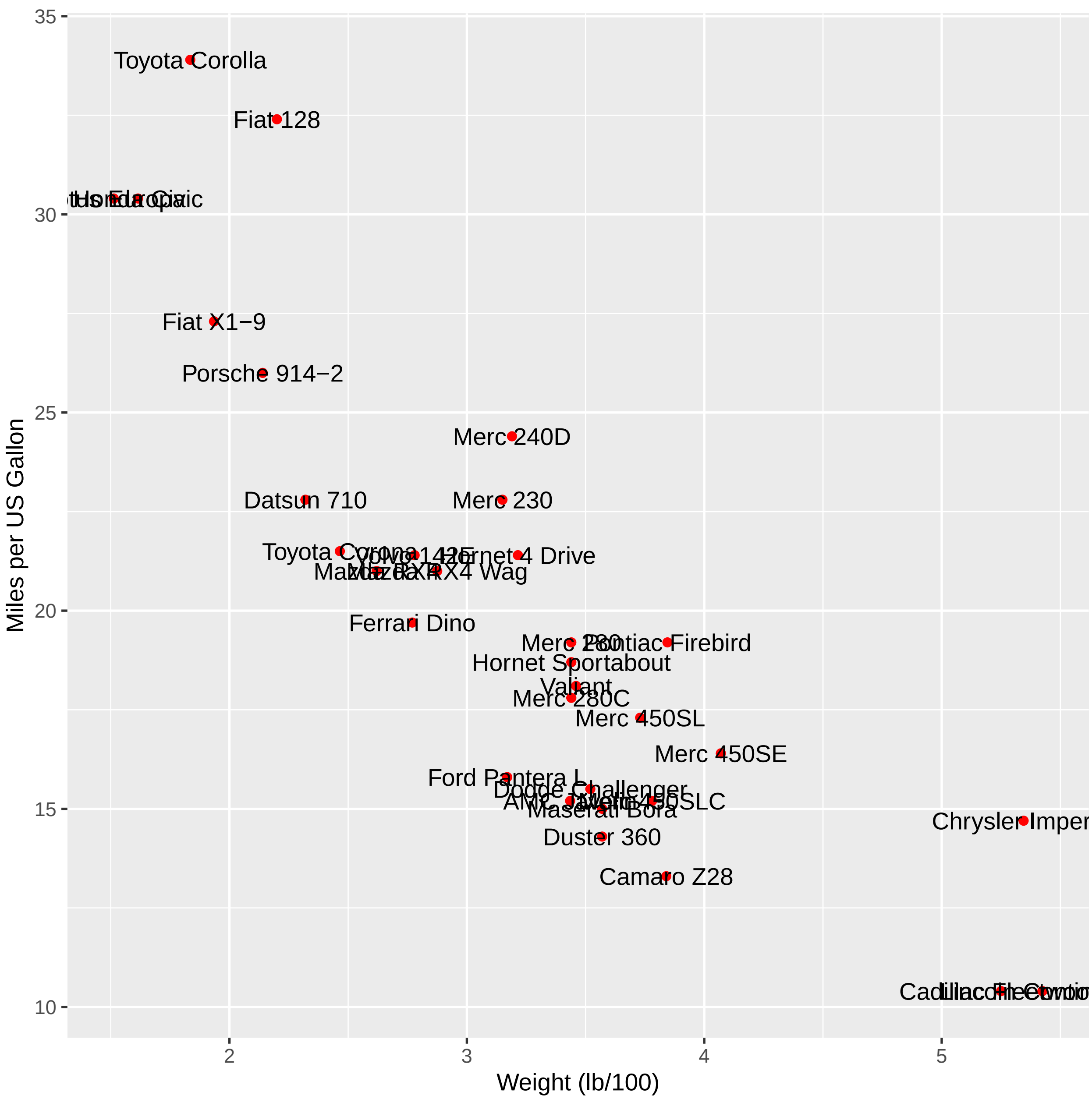
Now let's plot miles per US gallon against weight (units: lb/1000), showing the car name:

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_text(aes(label = rownames(mtcars))) +
  labs(x = "Weight (lb/100)", y = "Miles per US Gallon")
```



We may even add the data points using `geom_point`:

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(color = 'red') +
  geom_text(aes(label = rownames(mtcars))) +
  labs(x = "Weight (lb/100)", y = "Miles per US Gallon")
```



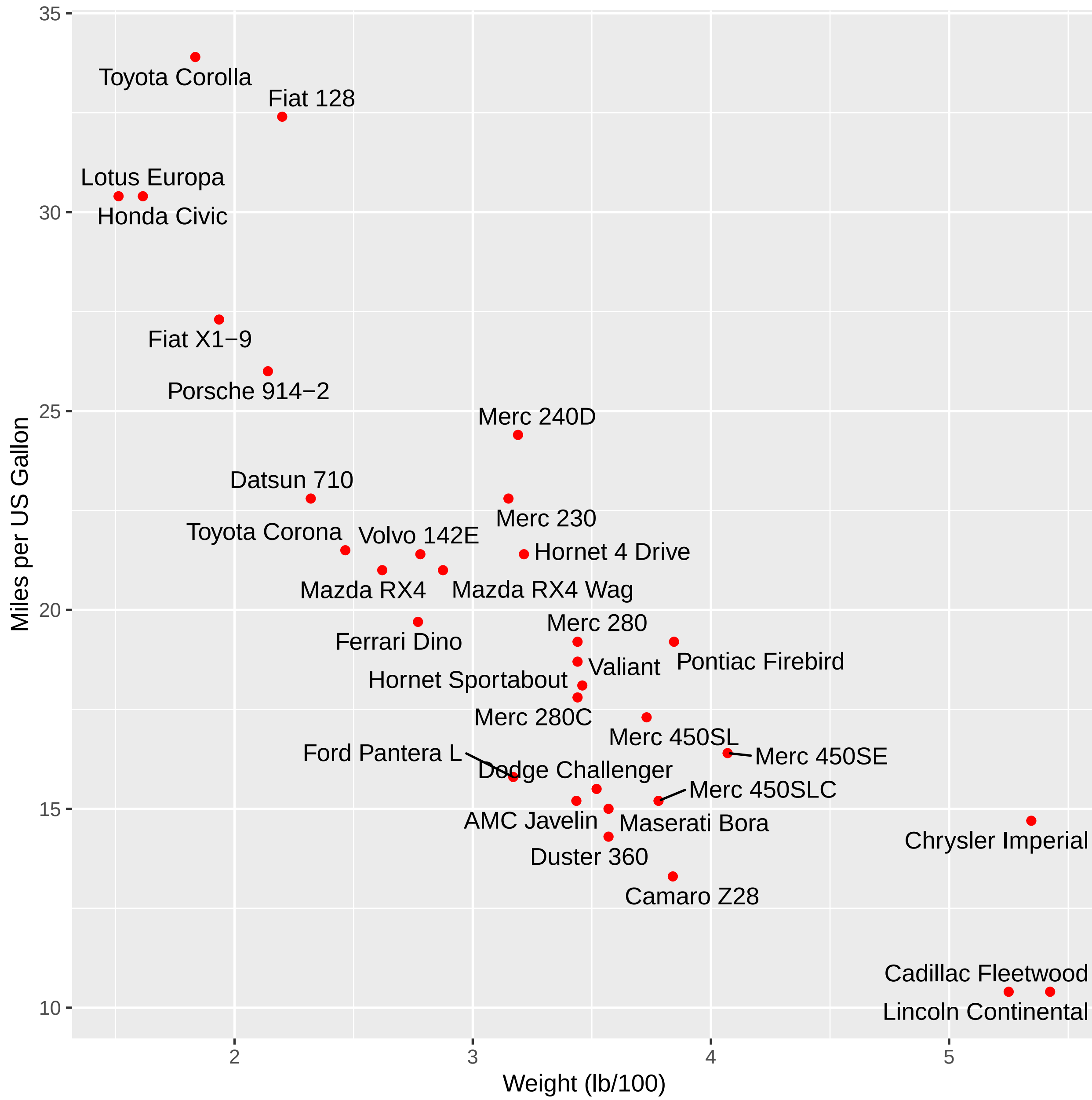
Unfortunately, the plot looks cluttered as many of the names are plotted over each other. We can resolve this using `geom_text_repel` instead of `geom_text` from the `ggrepel` package (which may have to be loaded):

```
# install.packages("ggrepel", repos = "http://www.stats.bris.ac.uk/R/")
require(ggrepel)
citation("ggrepel")
```

```
##
```

```
## To cite package 'ggrepel' in publications use:
##
##   Kamil Slowikowski (2021). ggrepel: Automatically Position
##   Non-Overlapping Text Labels with 'ggplot2'. R package version 0.9.1.
##   https://CRAN.R-project.org/package=ggrepel
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {ggrepel: Automatically Position Non-Overlapping Text Labels with
## 'ggplot2'},
##     author = {Kamil Slowikowski},
##     year = {2021},
##     note = {R package version 0.9.1},
##     url = {https://CRAN.R-project.org/package=ggrepel},
##   }
```

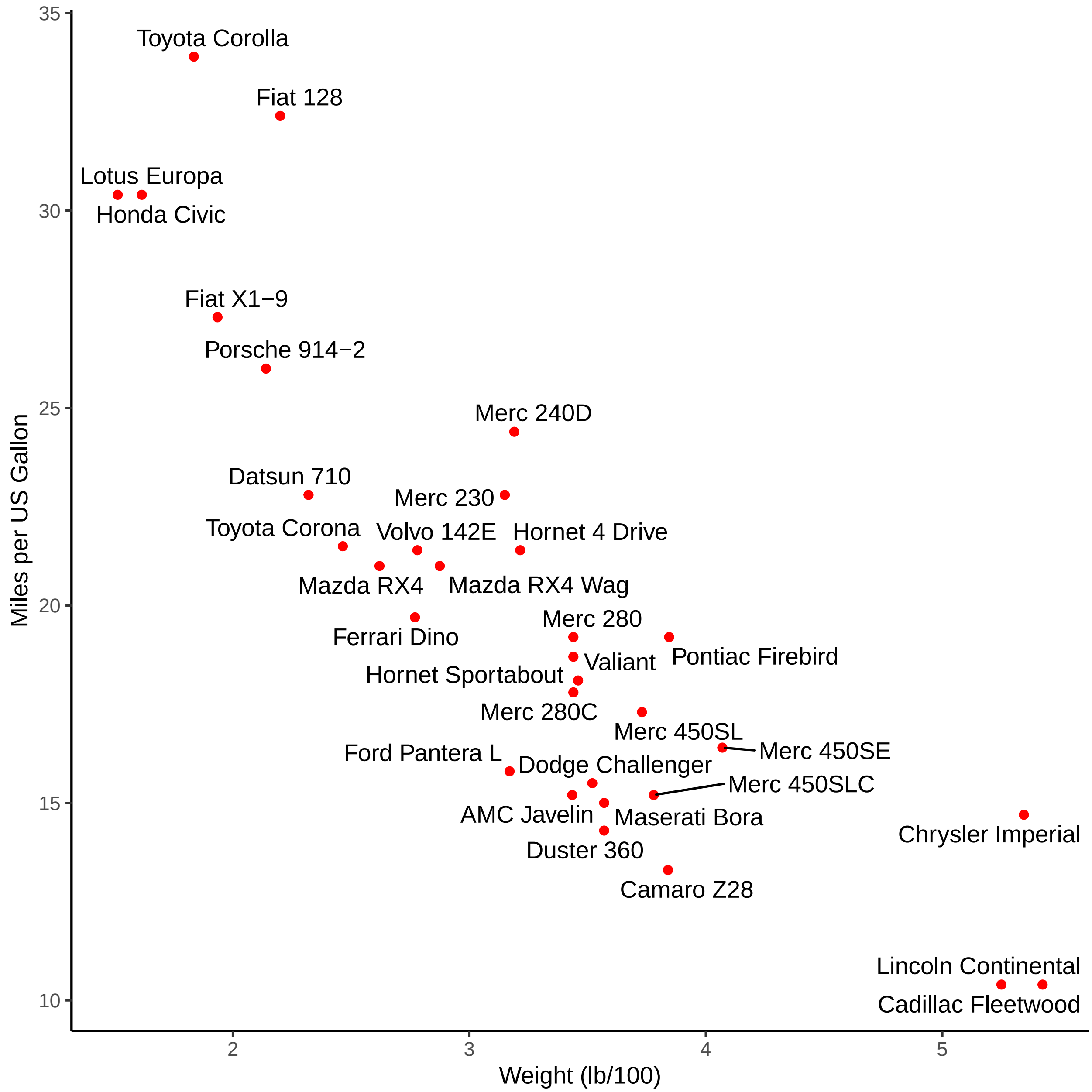
```r
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(color = 'red') +
  geom_text_repel(aes(label = rownames(mtcars))) +
  labs(x = "Weight (lb/100)", y = "Miles per US Gallon")
```



This is a considerable improvement.

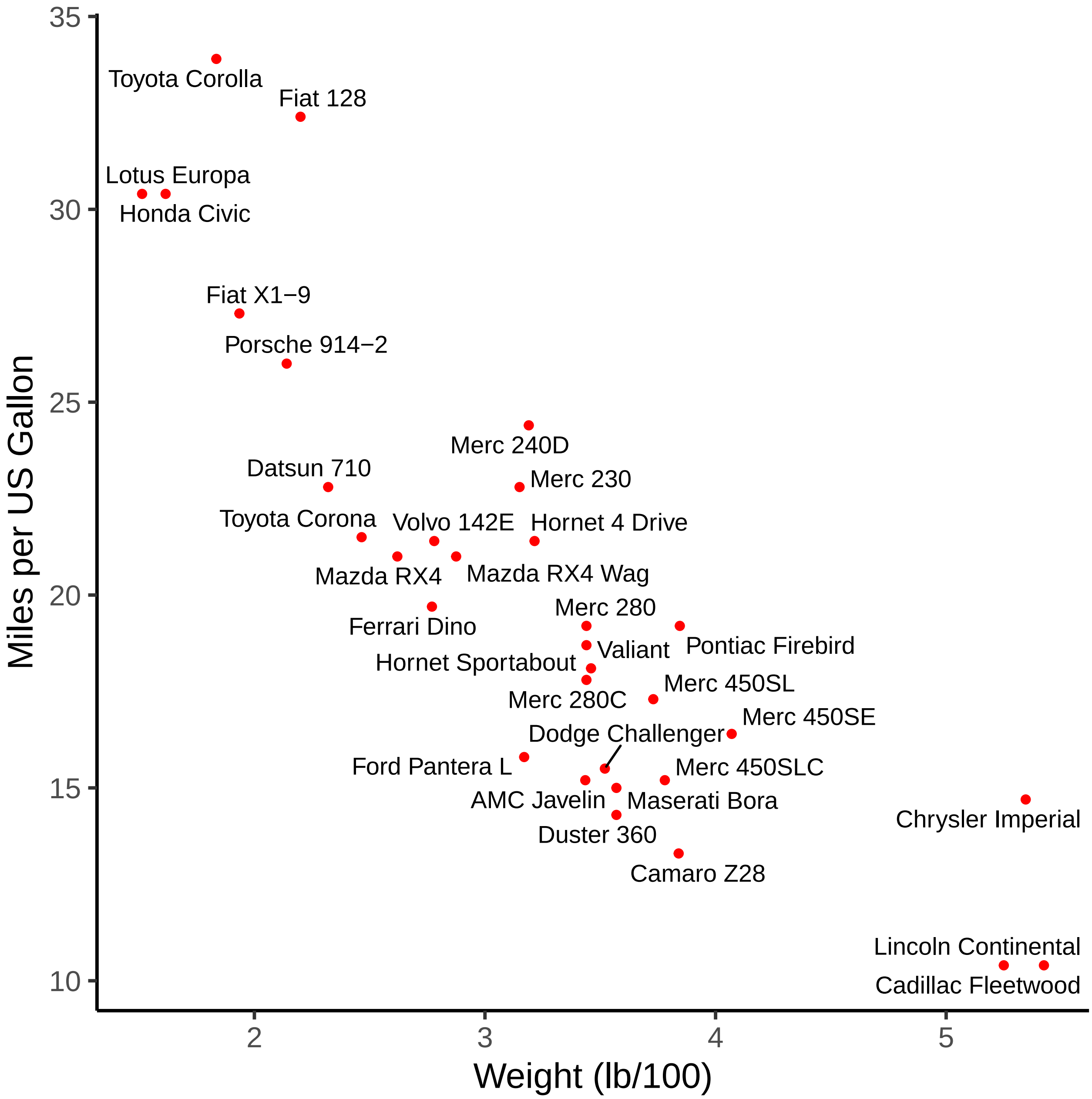
`ggplot2` supplies a range of built in themes; please see the help file for `ggtheme`. Here we illustrate the use of `theme_classic`:

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(color = 'red') +
  geom_text_repel(aes(label = rownames(mtcars))) +
  labs(x = "Weight (lb/100)", y = "Miles per US Gallon") +
  theme_classic()
```



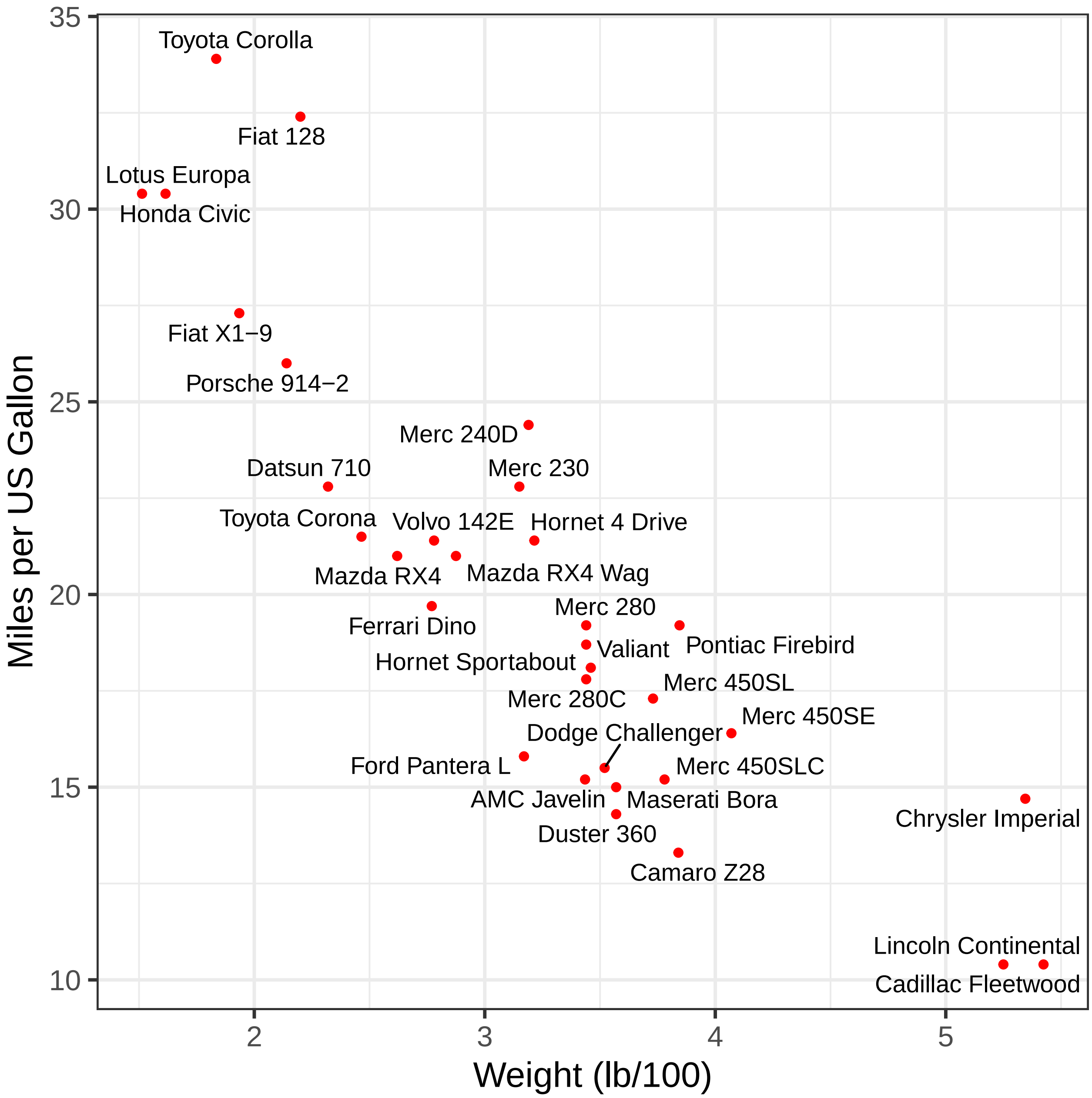
We can change the base font size:

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(color = 'red') +
  geom_text_repel(aes(label = rownames(mtcars))) +
  labs(x = "Weight (lb/100)", y = "Miles per US Gallon") +
  theme_classic(base_size = 16)
```



Please also consider:

```
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(color = 'red') +
  geom_text_repel(aes(label = rownames(mtcars))) +
  labs(x = "Weight (lb/100)", y = "Miles per US Gallon") +
  theme_bw(base_size = 16) # A different built in theme
```



3 Summary Statistics

3.1 Measures of Location

We have already met the sample mean and the sample median. Here are the sample mean and the sample median of the age data, shown also on a `stripchart`:

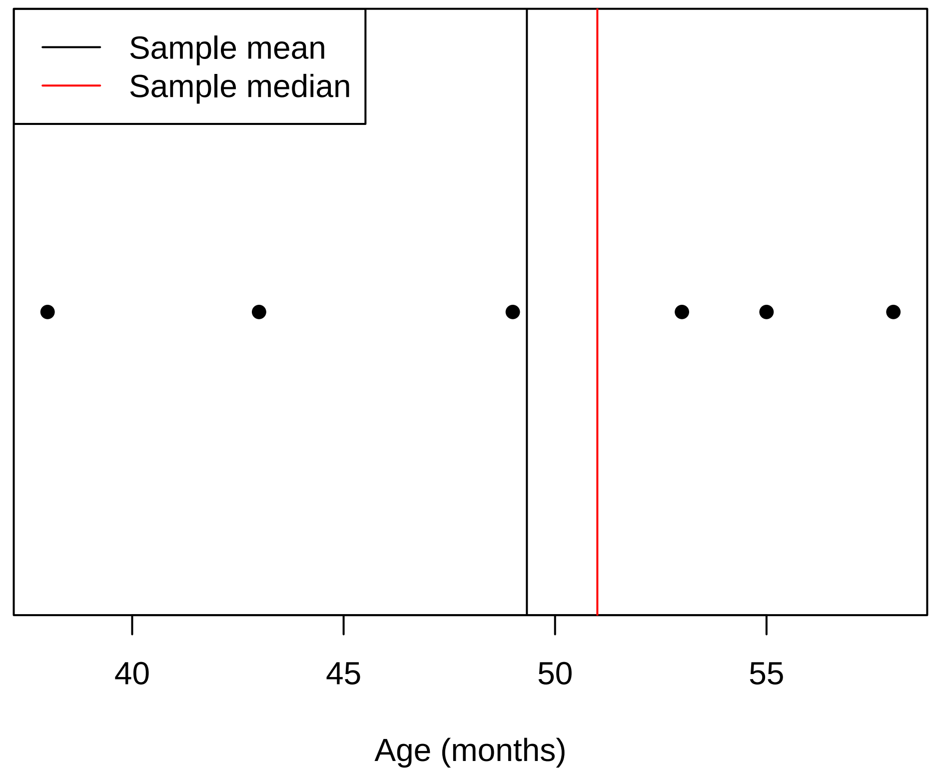
```
mean(x)
```

```
## [1] 49.33333
```

```
median(x)
```

```
## [1] 51
```

```
stripchart(x, pch = 16, xlab = "Age (months)")
abline(v = mean(x))
abline(v = median(x), col = "red")
legend("topleft",
       legend = c("Sample mean", "Sample median"), lty = 1, col = c("black", "red"))
```



The sample mean and sample median are one number summary of the location of the data and are therefore referred to as **measures of location**.

The sample median is **robust to outliers**.

3.2 Measures of Spread

The **sample variance** provides a **measure of spread**, that is a measure of how spread out the data are.

```
var(x)
```

```
## [1] 57.86667
```

Unfortunately, the units of the sample variance are the square of the original units. In this case the units of the sample variance would be months^2. This is not very helpful.

For this reason, it is more usual to work with the **sample standard deviation**. This is the square root of the sample variance and has the original units, in this case months:

```
sd(x)
```

```
## [1] 7.607014
```

This value makes sense in the context of the stripchart as a measure of spread of the data.

Another measure of spread, which is especially useful when there are outliers in the data, is the **sample interquartile range**:

```
IQR(x)
```

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

The sampler interquartile range is more robust to outliers than the sample standard deviation:

```
sd(x_with_outlier)
```

```
## [1] 359.3853
```

```
sd(x) # Big difference
```

```
## [1] 7.607014
```

```
IQR(x_with_outlier)
```

```
## [1] 10.5
```

```
IQR(x) # Small difference
```

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

We will meet the sample interquartile range below when we discuss the boxplot.

4 Statistical Modelling: The Simple Linear Regression Model and Correlation

4.1 Data Revisited

Let us revisit the ages in months and the heights in cms from six children:

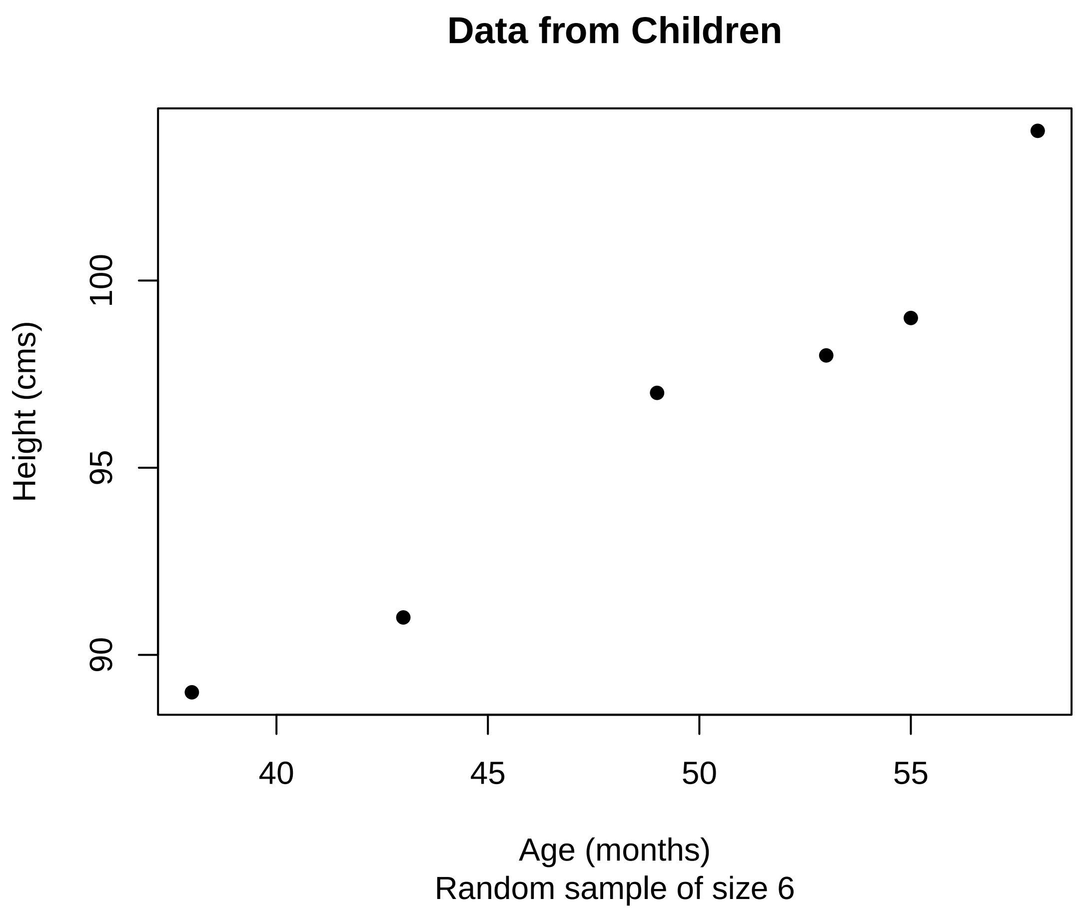
$$\text{age } x = \begin{pmatrix} 53 \\ 43 \\ 58 \\ 38 \\ 49 \\ 55 \end{pmatrix}_{6\times 1}$$

and

$$\text{height } y = \begin{pmatrix} 98 \\ 91 \\ 104 \\ 89 \\ 97 \\ 99 \end{pmatrix}_{6\times 1}.$$

We plotted the data as follows:

```
x <- c(53, 43, 58, 38, 49, 55)
y <- c(98, 91, 104, 89, 97, 99)
plot(x, y, xlab = "Age (months)", ylab = "Height (cms)", main = "Data from Children",
     sub = "Random sample of size 6", pch = 16)
```



4.2 A Fundamental Question: How Does One Variable Depend on Another?

The plot leads us to ask the question

- how does child height depend on child age?

The plot suggests that child height depends linearly (like a straight line) on child age.

4.3 Mathematical Formulation of the Simple Linear Regression Model

In mathematics, a linear model can be written as

$$y = \alpha + \beta x,$$

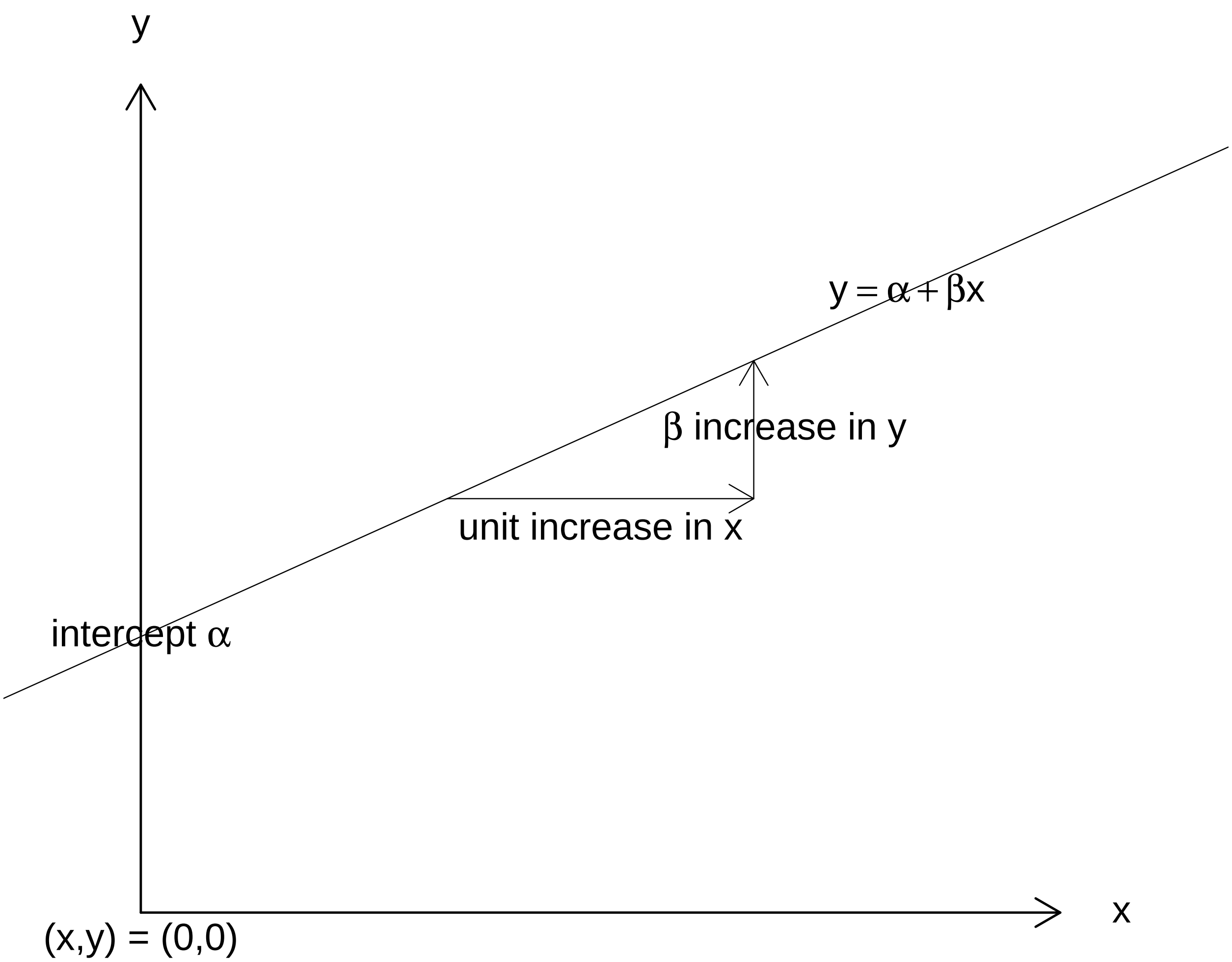
in which

- α is the intercept, that is the value of y when $x = 0$

and

- β is the slope, that is the increase in y for a unit increase in x.
  In other words if x increases by 1, y will increase by β.

Here is an example for positive β.



If β is negative, then the line slopes downwards.

We can see from the plot of the data that, although child height depend linearly on child age, that dependence is not perfect. Some errors are involved. To take account of these we propose a **statistical model**

$$y = \alpha + \beta x + \text{error},$$

in which y is child height, x is child age and the error term is random.

This model is referred to as a **simple linear regression model**.

- α and β are referred to as parameters of the model.

4.4 Fitting the Simple Linear Regression Model

We can get R to estimate the values of α and β for us. This is done using the linear model function `lm`:

```
m <- lm(y ~ x) # Fit a linear model and save its results
m # Look at the results
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##     61.4931       0.7062
```

```r
coef(m) # Extract the coefficients
```

```
## (Intercept)           x
##  61.4930876   0.7062212
```

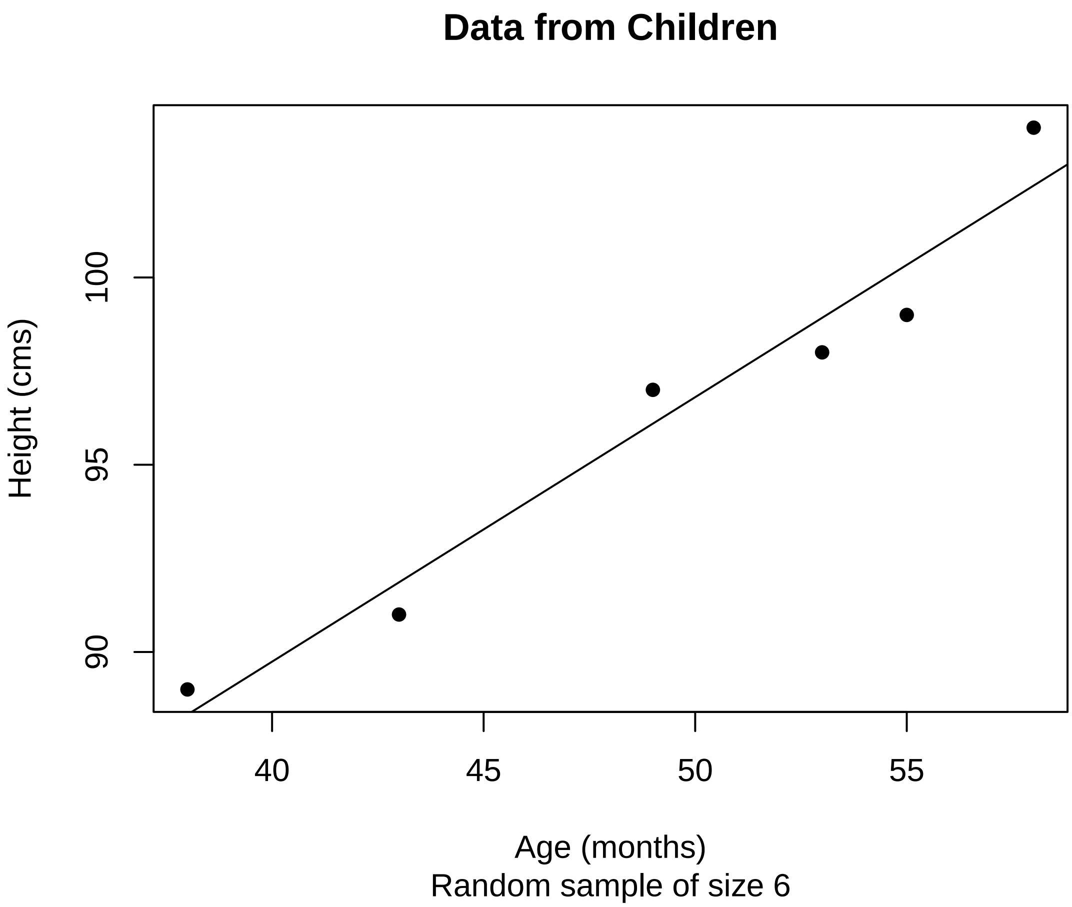
We see that the estimate of α is $\hat{\alpha} =$ 61.49 and that the estimate of β is $\hat{\beta} =$ 0.7062.

Hence, the height of a child of zero age is estimated to be 61.49. This makes no real sense because there is no child of zero age in the data set. It's an example of **extrapolation**, that is making statements beyond the range of the data.

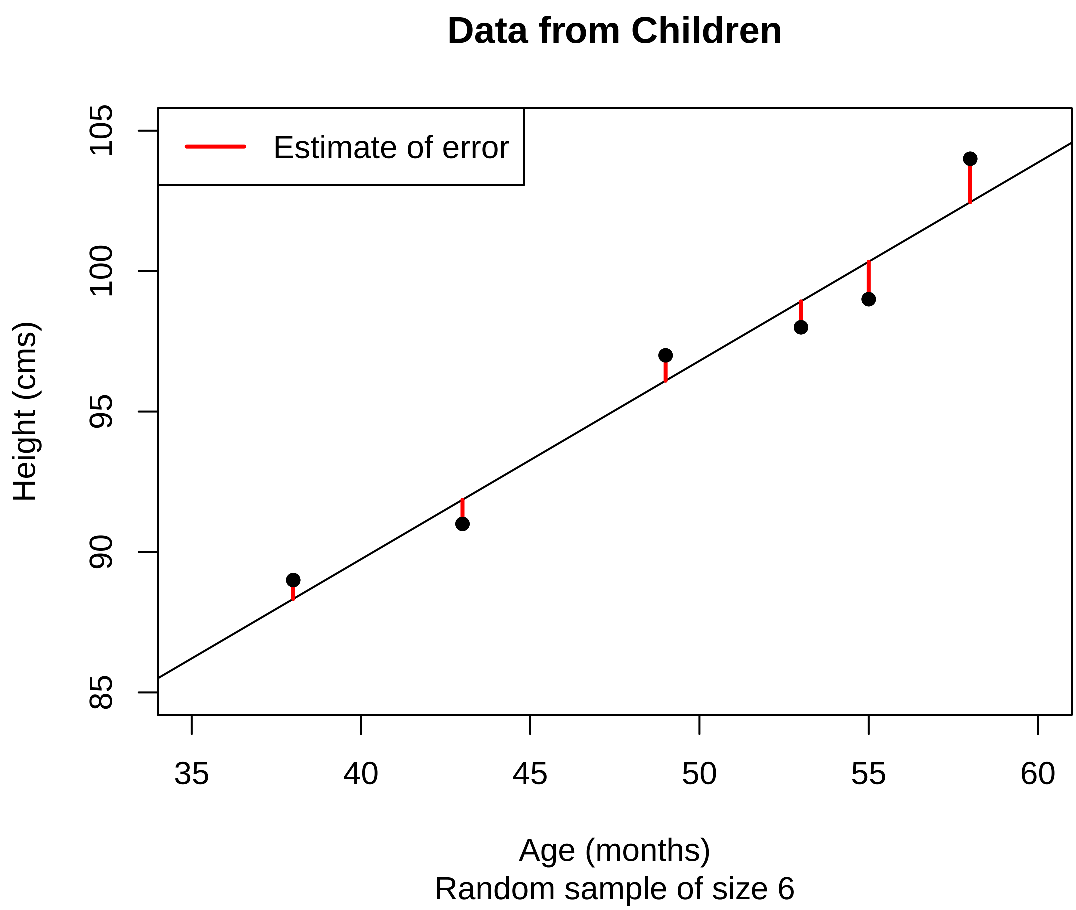
For every one month increase in child age, we estimate that child height increases by 0.7062 cms.

We can add the fitted line $y = \hat{\alpha} + \hat{\beta}x =$ 61.49 + 0.7062 x to the plot using `abline`:

```r
plot(x, y, xlab = "Age (months)", ylab = "Height (cms)", main = "Data from Children",
     sub = "Random sample of size 6", pch = 16)
abline(m) # Add a fitted straight line
```



This plot makes the errors at each point clear to us. Here the estimates of these errors are shown in red:



The estimates of the errors are called **residuals**.

4.5 More Information the Fitted Model with Interpretation

We can obtain more information about the model that we have fitted using the `summary` function:

```
summary(m)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##       1       2       3       4       5       6
## -0.9228 -0.8606  1.5461  0.6705  0.9021 -1.3353
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 61.49309    3.88251  15.838 9.29e-05 ***
## x            0.70622    0.07793   9.062 0.000822 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.326 on 4 degrees of freedom
## Multiple R-squared:  0.9536, Adjusted R-squared:  0.9419
## F-statistic: 82.12 on 1 and 4 DF,  p-value: 0.0008218
```

We will now explore some of this information.

- The R^2 value or `Multiple R-squared` value

First, the R^2 value 0.9536 tells us the proportion of information in the y data that is explained by x in the model. Here, 95.4% of the information in height is explained by age.

- The Second p-Value

The values in the `Pr(>|t|)` column of the summary table are referred to as p-values. The second of these 0.000822 is important to us. It allows us to answer the question:

- **Is there an underlying linear relationship between height (y) and age (x)?**

This is a profound question. We are not asking whether there is a relationship between height and age in the data collected. We are asking, more generally, whether there is a relationship between height and age in a much larger population of children.

For reasons that we will not go into, we would conclude that

- **there is an underlying linear relationship between height (y) and age (x) if the second p-value in the summary table were less than** 0.05.

The second p-value is 0.000822 and so is certainly less than 0.05. We therefore conclude that there is an underlying linear relationship between height (y) and age (x).

4.6 Prediction

It's a natural question to ask what value of y the model would predict at a given value of x. This is quite easy to answer.

- We can obtain the values of y at the x data values using the function `fitted`, as these values are the fitted y values:

```
fitted(m)
```

```
##         1         2         3         4         5         6
##  98.92281  91.86060 102.45392  88.32949  96.09793 100.33525
```

- We can obtain the value of y at any given value of x, such as $x = 40$, using `predict`:

```
predict(m, newdata = data.frame(x = 40)) # Predict at x = 40
```

```
##        1
## 89.74194
```

Note that the new data has to be specified as a data frame.

- We can extend this to more than one x value:

```
predict(m, newdata = data.frame(x = c(40,45))) # Predict at x = 40 and x = 45
```

```
##        1        2
## 89.74194 93.27304
```

- We can ask how reliable is the estimate of the underlying linear relationship $\alpha + \beta x$:

```
predict(m, newdata = data.frame(x = c(40,45)), interval = "confidence")
```

```
##        fit      lwr      upr
## 1 89.74194 87.22482 92.25905
## 2 93.27304 91.50196 95.04412
```

```
   # Predictions with confidence intervals
```

The intervals defined by the columns `lwr` (for lower value of the interval) and `upr` (for upper value of the interval) are referred to as a **confidence intervals**. The larger these intervals, the less reliable are the estimates of $\alpha + \beta x$.

- We can also ask how reliable is the estimate of a new data value $y^{\text{new}} = \alpha + \beta x + \text{error}$:

```
predict(m, newdata = data.frame(x = c(40,45)), interval = "prediction")
```

```
##        fit      lwr      upr
## 1 89.74194 85.28307 94.20080
## 2 93.27304 89.18864 97.35745
```

```
   # Predictions with prediction intervals
```

The intervals defined by the columns `lwr` and `upr` are referred to as a **prediction intervals**. The larger these intervals, the less reliable are the estimates of a new data value $y^{\text{new}} = \alpha + \beta x + \text{error}$.

- We can also produce confidence intervals for the parameters α and β:

```
confint(m)
```

```
##                  2.5 %     97.5 %
## (Intercept) 50.7135185 72.2726566
## x            0.4898495  0.9225929
```

Again, the wider the interval, the less well the parameter is estimated.

4.7 Displaying Confidence and Prediction Intervals Graphically

We can display confidence intervals and prediction intervals along the range of the x values as we will now explain step by step:

- We will calculate the quantities that we wish to plot along a **seq**uence of x values of length N from the minumum value of x to the maximum value:

```
min(x)
```

```
## [1] 38
```

```
min_x <- min(x)
max(x)
```

```
## [1] 58
```

```
max_x <- max(x)
N <- 25 # Number of x points for future calculations
x_seq <- seq(from = min_x, to = max_x, length = N)
x_seq
```

```
##  [1] 38.00000 38.83333 39.66667 40.50000 41.33333 42.16667 43.00000 43.83333
##  [9] 44.66667 45.50000 46.33333 47.16667 48.00000 48.83333 49.66667 50.50000
## [17] 51.33333 52.16667 53.00000 53.83333 54.66667 55.50000 56.33333 57.16667
## [25] 58.00000
```

- Now predict y at this sequence of x values, asking for confidence intervals:

```
y_conf <- predict(m, newdata = data.frame(x = x_seq), interval = "confidence")
y_conf
```

```
##          fit      lwr      upr
## 1   88.32949 85.45357 91.20542
## 2   88.91801 86.19420 91.64182
```

```
## 3   89.50653  86.93119  92.08187
## 4   90.09505  87.66387  92.52622
## 5   90.68356  88.39143  92.97570
## 6   91.27208  89.11288  93.43129
## 7   91.86060  89.82701  93.89419
## 8   92.44912  90.53240  94.36584
## 9   93.03763  91.22734  94.84793
## 10  93.62615  91.90989  95.34241
## 11  94.21467  92.57792  95.85142
## 12  94.80319  93.22922  96.37716
## 13  95.39171  93.86173  96.92168
## 14  95.98022  94.47380  97.48665
## 15  96.56874  95.06448  98.07300
## 16  97.15726  95.63367  98.68085
## 17  97.74578  96.18217  99.30938
## 18  98.33429  96.71151  99.95708
## 19  98.92281  97.22368 100.62194
## 20  99.51133  97.72090 101.30176
## 21 100.09985  98.20531 101.99439
## 22 100.68836  98.67890 102.69783
## 23 101.27688  99.14343 103.41034
## 24 101.86540  99.60038 104.13042
## 25 102.45392 100.05099 104.85684
```

We obtain a **matrix** of values.

- Now predict y at this sequence of x values, asking for prediction intervals:

```r
y_pred <- predict(m, newdata = data.frame(x = x_seq), interval = "prediction")
y_pred
```

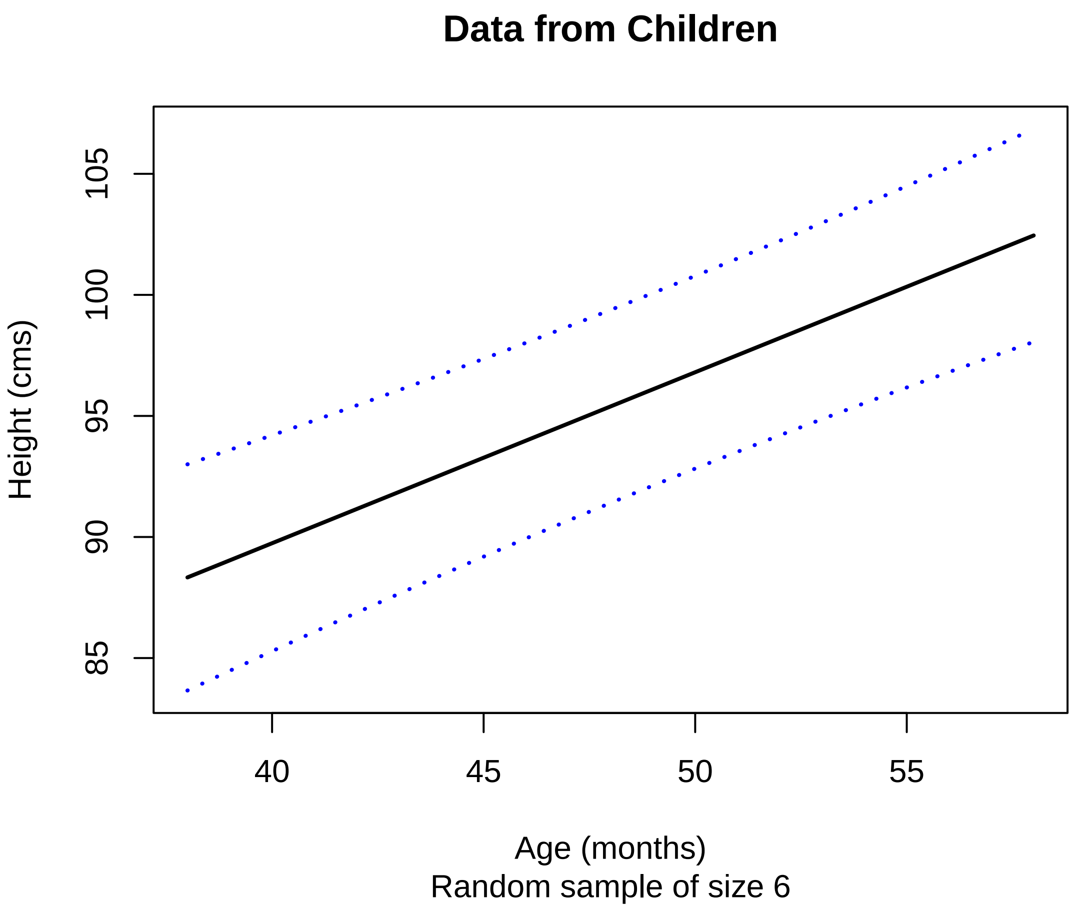
```
##          fit      lwr       upr
## 1   88.32949 83.65867  93.00032
## 2   88.91801 84.33928  93.49674
## 3   89.50653 85.01453  93.99852
## 4   90.09505 85.68412  94.50597
## 5   90.68356 86.34772  95.01940
## 6   91.27208 87.00502  95.53914
## 7   91.86060 87.65571  96.06549
## 8   92.44912 88.29948  96.59875
## 9   93.03763 88.93607  97.13920
## 10  93.62615 89.56522  97.68709
## 11  94.21467 90.18669  98.24265
## 12  94.80319 90.80031  98.80606
## 13  95.39171 91.40592  99.37749
## 14  95.98022 92.00342  99.95702
## 15  96.56874 92.59276 100.54472
## 16  97.15726 93.17392 101.14059
## 17  97.74578 93.74696 101.74459
## 18  98.33429 94.31197 102.35662
## 19  98.92281 94.86909 102.97653
## 20  99.51133 95.41850 103.60416
## 21 100.09985 95.96041 104.23928
## 22 100.68836 96.49509 104.88164
## 23 101.27688 97.02279 105.53097
## 24 101.86540 97.54383 106.18697
```

```
## 25 102.45392 98.05850 106.84934
```

Again, we obtain a **mat**rix of values.

- Plot the matrix of the wider prediction intervals using `matplot`:

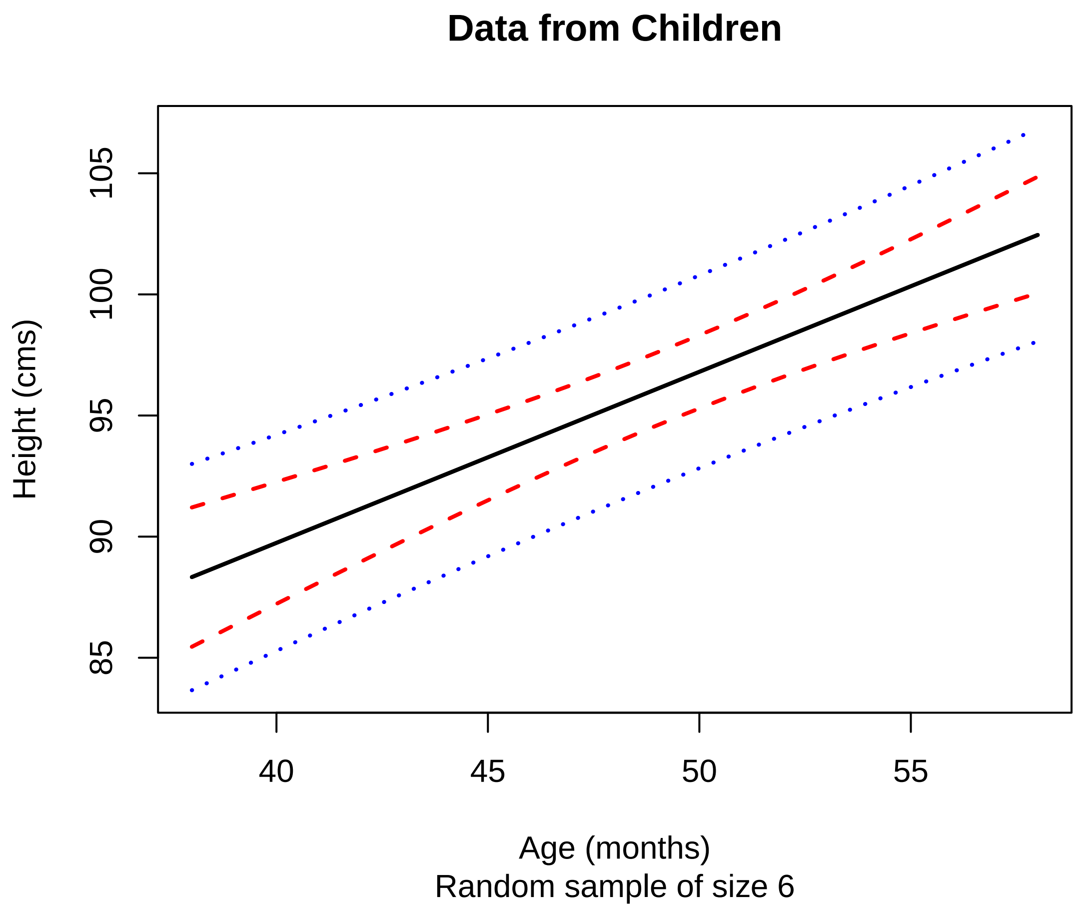
```
matplot(x_seq, y_pred,
        type = "l", # lines ...
        lty = c(1, 3, 3), # ... of these types ...
        col = c("black", "blue", "blue"), # ... of these colours ...
        lwd = 2, # ... of this width
        xlab = "Age (months)", ylab = "Height (cms)",
        main = "Data from Children",
        sub = "Random sample of size 6")
```



- Now add **lines** corresponding to the confidence intervals using `matlines`:

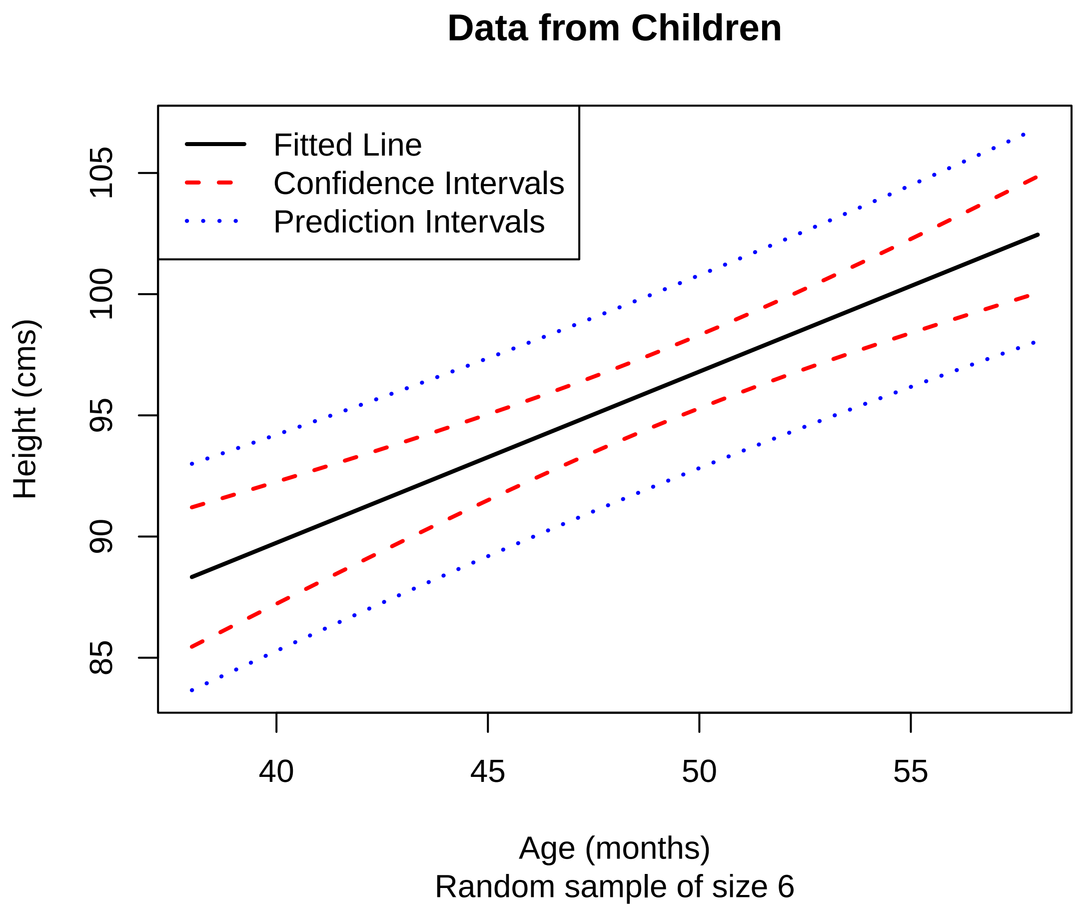
```
matplot(x_seq, y_pred,
        type = "l", # lines ...
        lty = c(1, 3, 3), # ... of these types ...
        col = c("black", "blue", "blue"), # ... of these colours ...
        lwd = 2, # ... of this width
        xlab = "Age (months)", ylab = "Height (cms)",
        main = "Data from Children",
        sub = "Random sample of size 6")
matlines(x_seq, y_conf,
        lty = c(1, 2, 2),
```

```
        lwd = 2,
        col = c("black", "red", "red")
       )
```



- Add a legend of explanation:

```
matplot(x_seq, y_pred,
        type = "l", # lines ...
        lty = c(1, 3, 3), # ... of these types ...
        col = c("black", "blue", "blue"), # ... of these colours ...
        lwd = 2, # ... of this width
        xlab = "Age (months)", ylab = "Height (cms)",
        main = "Data from Children",
        sub = "Random sample of size 6")
matlines(x_seq, y_conf,
        lty = c(1, 2, 2),
        lwd = 2,
        col = c("black", "red", "red")
       )
legend("topleft",
       legend = c("Fitted Line",
                  "Confidence Intervals",
                  "Prediction Intervals"),
       lty = c(1, 2, 3),
       lwd = 2,
       col = c("black", "red", "blue"))
```



The confidence intervals tell us about the reliability of the estimation of the underlying linear relationship $\alpha + \beta x$.

The prediction intervals tell us about the reliability of the estimation of a new data value $y^{\text{new}} = \alpha + \beta x + \text{error}$.

4.8 Tests of correlations: What is the Dependence Between Variables?

The simple linear regression model is used to understand how one variable y depends linearly on another x.

The **correlaton coefficient** provides a measure of linear association or dependence between the variables x and y.

- The correlation coefficient takes values between -1 and 1.
- Values of the correlation coefficient near 1 suggest a strong positive linear relationship between the two variables x and y: as one increases, so does the other.
- Values of the correlation coefficient near -1 suggest a strong negtive linear relationship between the two variables x and y: as one increases, the other decreases.
- We can estimate the correlation coefficient from data using Pearson's product moment correlation coefficient. Pearson's product moment correlation coefficient is usually appropriate for data from continuous random variables, that is random variables that can take any value in a range such as height and age.
- Before doing so, we should plot our data to see whether there seems to be a linear relationship between x and y. If there isn't, it may not be appropriate to estimate correlation in this way.

- Here is the estimated correlation:

```
cor(x, y)
```

```
## [1] 0.976501
```

This suggests a very strong positive linear dependency between height and age in children.

We can also estimate the correlations of the variables in a data frame as follows:

```
data_children <- data.frame(Age = x, Height = y)
data_children
```

```
##   Age Height
## 1  53     98
## 2  43     91
## 3  58    104
## 4  38     89
## 5  49     97
## 6  55     99
```

```
cor(data_children)
```

```
##             Age   Height
## Age    1.000000 0.976501
## Height 0.976501 1.000000
```

The correlation of a variable with itself is automatically 1.

We can also test to see

- whether there is, more general, linear dependency between age and height in a much larger population of children.

As before, we need a p-value. We would conclude that

- **there is an underlying linear dependency between age (x) and height (y) if the p-value is less than 0.05**.

We can calculate the appropriate p-value using `cor.test`:

```
cor.test(x, y)
```

```
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 9.0621, df = 4, p-value = 0.0008218
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7948533 0.9975296
## sample estimates:
##      cor
## 0.976501
```

We can extract the estimated correlation and the p-value:

```
cor.test(x, y)$estimate
```

```
##      cor
## 0.976501
```

```r
cor.test(x, y)$p.value
```

```
## [1] 0.0008218175
```

Since the p-value of 0.0008218 is less than 0.05, we conclude that there is an underlying linear relationship between age (x) and height (y) in the more general population of children.

Behind the computation of every p-value there is considerable mathematical theory based on modelling assumptions. We will not discuss these assumptions here. However, tests based on other estimates of correlation, such as Spearman's rank correlation coefficient or Kendall's τ (tau) make less strong assumptions and may be more appropriate for other types of data, such as discrete data (data from variables that take a limited number of values) and ranked data (data based on ranks, such as first, second,. . .).

To compute Spearman's rank correlation and Kendall's τ is simple:

```r
cor(x, y, method = "spearman")
```

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

```r
cor(x, y, method = "kendall")
```

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

These two estimates of correlation are based on the ranks of the data:

```r
x
```

```
## [1] 53 43 58 38 49 55
```

```r
rank(x)
```

```
## [1] 4 2 6 1 3 5
```

```r
   # Rank 1 corresponds to the lowest value
   # Rank 6 corresponds to the highest
y
```

```
## [1]  98  91 104  89  97  99
```

```r
rank(y)
```

```
## [1] 4 2 6 1 3 5
```

We can see that the ranks of the age and height data are the same. For this reason, these estimates report a perfect correlation of 1.

With these estimates of correlation, we reach the same conclusions about the underlying linear dependency between age (x) and height (y). The p-values are zero:

```r
cor.test(x, y, method = "spearman")
```

```
## 
##  Spearman's rank correlation rho
## 
## data:  x and y
## S = 0, p-value = 0.002778
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
## rho
##   1
```

```r
cor.test(x, y, method = "kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  x and y
## T = 15, p-value = 0.002778
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
## tau
##   1
```

We end this section by introducing and working with a famous data set. The data consist of 17 pairs of numbers corresponding to observed boiling point and corrected barometric pressure at locations in the Alps.

The data are available from the `alr4` package:

```
# install.packages("alr4", repos = "http://www.stats.bris.ac.uk/R/") # Needed on your own machine
require(alr4)
citation("alr4")
```

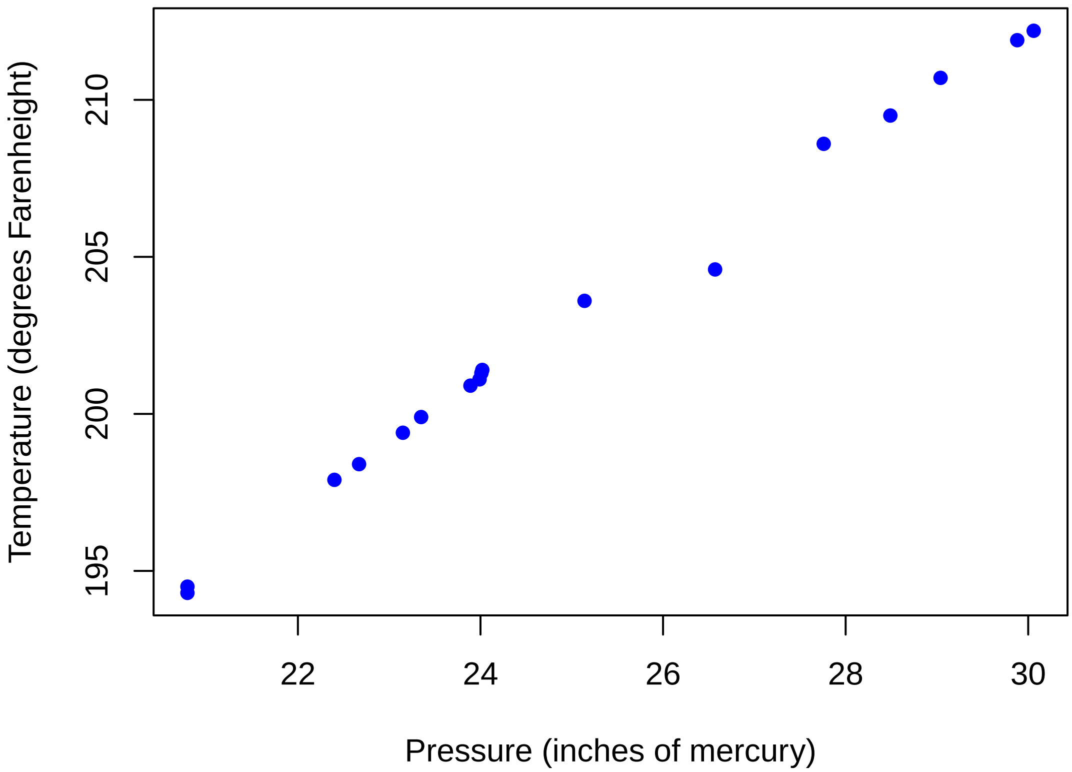
```
##
## To cite the alr4 package in publications use:
##
##   Sanford Weisberg (2014). Applied Linear Regression, Fourth Edition.
##   Hoboken NJ: Wiley. URL: http://z.umn.edu/alr4ed
##
## A BibTeX entry for LaTeX users is
##
##   @Book{,
##     title = {Applied Linear Regression},
##     edition = {Fourth},
##     author = {Sanford Weisberg},
##     year = {2014},
##     publisher = {Wiley},
##     address = {Hoboken {NJ}},
##     url = {http://z.umn.edu/alr4ed},
##   }
```

Look at the help file for the data in the alr4 package:

```
?Forbes
```

Let's plot the data:

```
plot(bp ~ pres, data = Forbes,
     pch = 16, col = "blue",
     xlab = "Pressure (inches of mercury)",
     ylab = "Temperature (degrees Farenheight)")
```



Note that the variables to use in this scatter plot are defined using the formula `bp ~ pres` (how does temperature depend on pressure?). The `data` argument tells the `plot` function to look into the data frame `Forbes` for the variables `bp` and `pres`.

We can compute the above correlations as follows:

```
with(Forbes, cor(bp, pres))
```

```
## [1] 0.9972102
```

```
with(Forbes, cor(bp, pres, method = "spearman"))
```

```
## [1] 0.9993871
```

```
with(Forbes, cor(bp, pres, method = "kendall"))
```

```
## [1] 0.9963167
```

The correlation between pressure and temperature is very high, suggesting that there is a strong positive dependence between pressure and temperature. As pressure increases, so does temperature.

The function `with` tells `cor` to work **with** the data in `Forbes`, that is to look in `Forbes` for `bp` and `pres`.

Note that

```
with(Forbes, cor(bp, pres, method = "spearman"))
```

```
## [1] 0.9993871
```

gives the same answer as

```
with(Forbes, cor(rank(bp), rank(pres)))
```

```
## [1] 0.9993871
```

That is, Spearman's rank correlation coefficient is the same as Pearson's product moment correlation coefficient applied to the ranks of the data.

Exercise: Perform a simple linear regression analysis of the data in the `Forbes` data frame.

5 Introduction to Programming: Writing a Simple R Function

R is also a powerful programming language. The basic programming building block in R is the function. We have met many of R's own functions such as `mean`. We will now briefly illustrate how we can write and use our own R functions. This will

- provide us with further insight into R functions;
- put us on the path to becoming an R programmer.

Programming is a very valuable and important transferable skill. Once we have learnt to programme in one language, it becomes much simpler to programme in other languages.

5.1 Aims of Our Function

The aims of our function will be

- To calculate the distance of the point (x, y) from the origin $(0, 0)$. This is given by Pythagoras' Theorem as

$$\sqrt{x^2 + y^2}. \tag{1}$$

- Then, more generally to calculate the distance of the point (x, y) from the point (x_0, y_0). This is given by Pythagoras' Theorem as

$$\sqrt{(x - x_0)^2 + (y - y_0)^2}. \tag{2}$$

- Then, to calculate the gradient ratio

$$\frac{y - y_0}{x - x_0}. \tag{3}$$

5.2 Getting Started with Function Writing

- Each function has a name, such as `my_dist` and takes the following basic structure:

```
my_dist <- function(){
}
```

- The arguments of the function (the named values that are passed to the function) will go between the round brackets `(`and `)`.
- The main body of the function will go between the curly brackets `{` and `}`. The main body of the function can contain any R code.
- Hence, we can extend our basic R function to compute (1):

```
my_dist <- function(x, y){
sqrt(x^2 + y^2)
}
```

- We can use the function in the usual way to calculate the distance of the point $(3, 4)$ from the origin $(0, 0)$, expecting the answer $\sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$:

```
my_dist(3, 4)
```

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

Here $x = 3$ and $y = 4$, while in

```
my_dist(4, 3)
```

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

$x = 4$ and $y = 3$.

- If we cannot remember the order of the arguments, we can refer to them by name:

```
my_dist(y = 3, x = 4)
```

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

(In this case, (1) is symmetric in x and y (i.e. if x is replaced with y and y is replaced with x the function is the same) so the order of the arguments does not matter. However, for a general function the order of the arguments does matter.)

5.3 Default Values

It's easy to extend our R function to compute (2) by introducing additional arguments:

```
my_dist <- function(x, y, x0, y0){
    sqrt((x - x0)^2 + (y - y0)^2)
}
```

Let's calculate the distance of the point $(6, 14)$ from the point $(1, 2)$

```
my_dist(6, 14, 1, 2)
```

```
## [1] 13
```

Now, let us say that most of the time we are interested in calculating the distance between the point (x, y) and the origin $(x_0, y_0) = (0, 0)$. We can introduce the specific values $x_0 = 0$ and $y_0 = 0$ as arguments of the function:

```
my_dist <- function(x, y, x0 = 0,y0 = 0){
sqrt((x - x0)^2 + (y - y0)^2)
}
```

The values $x_0 = 0$ and $y_0 = 0$ are referred to as **default** values. These values are used unless other values are specified, as these examples show:

- The distance of the point $(6, 14)$ from the point $(1, 2)$ as before:

```
my_dist(6, 14, 1, 2)
```

```
## [1] 13
```

- The distance between the point $(x, y) = (3, 4)$ and the origin $(x_0, y_0) = (0, 0)$:

```
my_dist(3, 4)
```

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

or

```
my_dist(3, 4, 0, 0)
```

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

or

```
my_dist(x0 = 0, y0 = 0, x = 3, y = 4)
```

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

5.4 Returning Values

The function returns the value on its last line and this can be used in a future calculation:

```
d <- my_dist(3, 4)
d
```

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

```
2*d
```

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

Now let us get the function to calculate and report the **gradient** (3) as well as the distance (2):

```
my_dist <- function(x, y, x0 = 0,y0 = 0){
d <- sqrt((x - x0)^2 + (y - y0)^2)
g <- (y - y0)/ (x - x0)
print(d)
print(g)
}
my_dist(3,4)
```

```
## [1] 5
## [1] 1.333333
```

Note that we need to **print** values within a function to make them visible on the screen.

To return more than one value for future use we need to include them in a list, as the following example illustrates:

```
my_dist <- function(x, y, x0 = 0,y0 = 0){
d <- sqrt((x - x0)^2 + (y - y0)^2)
g <- (y - y0)/ (x - x0)
return(list(D = d, G = g))
}
my_dist(3,4)
```

```
## $D
## [1] 5
##
## $G
## [1] 1.333333
```

```
results <- my_dist(3,4)
results
```

```
## $D
## [1] 5
##
## $G
## [1] 1.333333
```

```
results$D
```

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

```
results$G
```

```
## [1] 1.333333
```

Note that:

- Values are returned from the function using **return**.
- We can specify the name of the returned values in the `list`: `list(D = d, G = g)`, for example.
- We can access these values for future use by means of the dollar `$`.

Lists are very flexible constructions in R. The can contain different types of information:

```
list(some_numbers = c(2, 5, 6),
     some_animals = c("cat", "dog", "elephant"),
     a_matrix = matrix(c(2,3,4,6,7,8), nrow = 2, byrow = TRUE))
```

```
## $some_numbers
## [1] 2 5 6
##
## $some_animals
## [1] "cat"      "dog"      "elephant"
##
## $a_matrix
##      [,1] [,2] [,3]
## [1,]    2    3    4
## [2,]    6    7    8
```

A list can include other lists

```
list(some_numbers = c(2, 5, 6),
     some_animals = list(mammals = c("cat", "dog"), reptile = c("turtle", "snake", "lizard")) )
```

```
## $some_numbers
## [1] 2 5 6
##
## $some_animals
## $some_animals$mammals
## [1] "cat" "dog"
##
## $some_animals$reptile
## [1] "turtle" "snake"  "lizard"
```

Note that

```
matrix(c(2,3,4,6,7,8), nrow = 2, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    4
## [2,]    6    7    8
```

takes the values in `c(2,3,4,6,7,8)` and arranges them in a matrix comprising two rows, assigning them to the matrix row by row.

5.5 Additional Comment

We will not discuss R function writing in any further detail here, except to say that there are several useful R constructions such as

- The **for** loop for repeating instructions:

```r
for(i in 2:5){
print(i^2)
}
```

```
## [1] 4
## [1] 9
## [1] 16
## [1] 25
```

This example is for illustration only; there are easier ways of achieving the same output such as

```r
(2:5)^2
```

```
## [1]  4  9 16 25
```

- The **if** and **else** construction:

```r
z <- -2
if(z > 0){
print("positive")
} else
{
print("negative or zero")
}
```

```
## [1] "negative or zero"
```

Later we will meet `ifelse` which is used when working with vectors.

6 Working with Data from a File

6.1 Questionnaire Data

Questionnaire data are available in the form of a comma separated variable or csv file called `MATH513_Questionnaire_Data.csv`.

- Open this file using Excel (just click on it) and note its structure.
- To read this file into R, we can use the function `read.csv` (see the help file) or the more powerful function `read_csv` from the `readr` package:

First, we will set a working directory to tell R where to look for files:

```r
setwd("~/Documents/backup_22_11_2021/Plym_teaching/MATH513/2019_20/Introduction_to_R")
```

Your working directory will probably be different from mine.

Sometimes the double back slash \\ is used instead of /.

```r
require(readr)
citation("readr")
```

```
##
## To cite package 'readr' in publications use:
##
##   Hadley Wickham, Jim Hester and Jennifer Bryan (2021). readr: Read
##   Rectangular Text Data. R package version 2.1.1.
##   https://CRAN.R-project.org/package=readr
##
## A BibTeX entry for LaTeX users is
```

```
##
##   @Manual{,
##     title = {readr: Read Rectangular Text Data},
##     author = {Hadley Wickham and Jim Hester and Jennifer Bryan},
##     year = {2021},
##     note = {R package version 2.1.1},
##     url = {https://CRAN.R-project.org/package=readr},
##   }
```

```
qd <- read_csv("MATH513_Questionnaire_Data.csv")
head(qd) # See the first few rows
```

```
## # A tibble: 6 x 19
##   Height   Age Sex    BirthPlace SiblingsNo EatMeat DrinkCoffee LikeBeer Sports
##    <dbl> <dbl> <chr>  <chr>           <dbl> <chr>   <chr>       <chr>    <chr>
## 1    170  23   Female essex               1 Yes     Yes         No       Yes
## 2    188  22.4 Male   London              1 Yes     Yes         No       No
## 3    180  30.1 Male   Athens              0 Yes     Yes         Yes      Yes
## 4    185  21   Male   China               0 Yes     Yes         Yes      Yes
## 5    170  22.1 Female Plymouth            2 Yes     Yes         No       No
## 6    182  25   Male   Nigeria             4 Yes     No          No       Yes
## # ... with 10 more variables: Driver <chr>, LeftHanded <chr>, Abroad <chr>,
## #   Sleep <dbl>, Rent <dbl>, Happy_accommodation <chr>, Distance <dbl>,
## #   Travel_time <dbl>, Mode_of_transport <chr>, Safe <chr>
```

```
names(qd) # Variable or column names
```

```
##  [1] "Height"              "Age"                 "Sex"
##  [4] "BirthPlace"          "SiblingsNo"          "EatMeat"
##  [7] "DrinkCoffee"         "LikeBeer"            "Sports"
## [10] "Driver"              "LeftHanded"          "Abroad"
## [13] "Sleep"               "Rent"                "Happy_accommodation"
## [16] "Distance"            "Travel_time"         "Mode_of_transport"
## [19] "Safe"
```

```
dim(qd) # Dimension
```

```
## [1] 18 19
```

```
nrow(qd)
```

```
## [1] 18
```

```
ncol(qd)
```

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

We see that the dataframe `qd` comprises 18 rows and 19 columns.

An Excel file, such as `MATH513_Questionnaire_Data.xlsx` in the same working directory can be read in using the function `read_excel` from the package `readxl`:

```
require(readxl)
citation("readxl")
```

```
##
## To cite package 'readxl' in publications use:
##
##   Hadley Wickham and Jennifer Bryan (2019). readxl: Read Excel Files. R
```

```
##   package version 1.3.1. https://CRAN.R-project.org/package=readxl
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {readxl: Read Excel Files},
##     author = {Hadley Wickham and Jennifer Bryan},
##     year = {2019},
##     note = {R package version 1.3.1},
##     url = {https://CRAN.R-project.org/package=readxl},
##   }
```

```
setwd("~/Documents/backup_22_11_2021/Plym_teaching/MATH513/2019_20/Introduction_to_R")
qd_excel <- read_excel("MATH513_Questionnaire_Data.xlsx",
                      sheet = "MATH513_Questionnaire_Data")
head(qd_excel) # See the first few rows
```

```
## # A tibble: 6 x 19
##   Height   Age Sex    BirthPlace SiblingsNo EatMeat DrinkCoffee LikeBeer Sports
##    <dbl> <dbl> <chr>  <chr>           <dbl> <chr>   <chr>       <chr>    <chr>
## 1    170  23   Female essex               1 Yes     Yes         No       Yes
## 2    188  22.4 Male   London              1 Yes     Yes         No       No
## 3    180  30.1 Male   Athens              0 Yes     Yes         Yes      Yes
## 4    185  21   Male   China               0 Yes     Yes         Yes      Yes
## 5    170  22.1 Female Plymouth            2 Yes     Yes         No       No
## 6    182  25   Male   Nigeria             4 Yes     No          No       Yes
## # ... with 10 more variables: Driver <chr>, LeftHanded <chr>, Abroad <chr>,
## #   Sleep <dbl>, Rent <dbl>, Happy_accommodation <chr>, Distance <dbl>,
## #   Travel_time <dbl>, Mode_of_transport <chr>, Safe <chr>
```

```
names(qd_excel)
```

```
##  [1] "Height"              "Age"                 "Sex"
##  [4] "BirthPlace"          "SiblingsNo"          "EatMeat"
##  [7] "DrinkCoffee"         "LikeBeer"            "Sports"
## [10] "Driver"              "LeftHanded"          "Abroad"
## [13] "Sleep"               "Rent"                "Happy_accommodation"
## [16] "Distance"            "Travel_time"         "Mode_of_transport"
## [19] "Safe"
```

We will make use of these data below.

7 Working with Data from a Database

This topic is the subject of a separate set of notes.

8 Data Wrangling with `dplyr`

The excellent `dplyr` package allows us to perform a wide range of data manipulation or wrangling operations. It also allows us to interact in a quick and efficient way with databases, as we will see in a separate set of notes.

Let's load `dplyr` and study its sitation:

```
require(dplyr)
citation("dplyr")
```

```
##
## To cite package 'dplyr' in publications use:
##
##   Hadley Wickham, Romain François, Lionel Henry and Kirill Müller
##   (2021). dplyr: A Grammar of Data Manipulation. R package version
##   1.0.7. https://CRAN.R-project.org/package=dplyr
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {dplyr: A Grammar of Data Manipulation},
##     author = {Hadley Wickham and Romain François and Lionel Henry and Kirill Müller},
##     year = {2021},
##     note = {R package version 1.0.7},
##     url = {https://CRAN.R-project.org/package=dplyr},
##   }
```

We will now see examples of the basic **verbs** supplied by `dplyr`. We will work with the above `qd` questionnaire data frame. If you didn't read in these data above, here is the code:

```
require(readr)
qd <- read_csv("MATH513_Questionnaire_Data.csv")
```

8.1 Selecting Columns

It's easy to select out (and then work with) particular columns/variables of a data frame using the `select` verb. Here we select the columns `Height` and `Travel_time`:

```
select(qd, Height, Travel_time)
```

```
## # A tibble: 18 x 2
##    Height Travel_time
##     <dbl>       <dbl>
##  1    170          20
##  2    188           5
##  3    180           3
##  4    185          20
##  5    170          40
##  6    182          15
##  7    175          90
##  8    187          15
##  9    170          15
## 10    168           5
## 11    162          25
## 12    168          12
## 13    165          10
## 14    158           6
## 15    171          20
## 16    163          10
## 17    186          23
## 18    177           6
```

Note that only the part of the data that fits into the available screen space is displayed.

Warning: Sometimes we have experienced a conflict between the `select` function from `dplyr` and a `select` in another package.

- To make it explicit that we want to use `select` from `dplyr`, we can use the following code:

```
dplyr::select(qd, Height, Travel_time)
```

```
## # A tibble: 18 x 2
##    Height Travel_time
##     <dbl>       <dbl>
##  1    170          20
##  2    188           5
##  3    180           3
##  4    185          20
##  5    170          40
##  6    182          15
##  7    175          90
##  8    187          15
##  9    170          15
## 10    168           5
## 11    162          25
## 12    168          12
## 13    165          10
## 14    158           6
## 15    171          20
## 16    163          10
## 17    186          23
## 18    177           6
```

8.2 Filering Rows

We can look at particular rows using the `filter` verb. Let's look at all the rows corresponding to males (`M`):

```
filter(qd, Sex == "M")
```

```
## # A tibble: 0 x 19
## # ... with 19 variables: Height <dbl>, Age <dbl>, Sex <chr>, BirthPlace <chr>,
## #   SiblingsNo <dbl>, EatMeat <chr>, DrinkCoffee <chr>, LikeBeer <chr>,
## #   Sports <chr>, Driver <chr>, LeftHanded <chr>, Abroad <chr>, Sleep <dbl>,
## #   Rent <dbl>, Happy_accommodation <chr>, Distance <dbl>, Travel_time <dbl>,
## #   Mode_of_transport <chr>, Safe <chr>
```

Again, note that only the part of the data that fits into the available screen space is displayed, with the other variables being listed by name.

Let's select people who are male and (`&`) who are taller than 170 cms:

```
filter(qd, Sex == "M" & Height > 170)
```

```
## # A tibble: 0 x 19
## # ... with 19 variables: Height <dbl>, Age <dbl>, Sex <chr>, BirthPlace <chr>,
## #   SiblingsNo <dbl>, EatMeat <chr>, DrinkCoffee <chr>, LikeBeer <chr>,
## #   Sports <chr>, Driver <chr>, LeftHanded <chr>, Abroad <chr>, Sleep <dbl>,
## #   Rent <dbl>, Happy_accommodation <chr>, Distance <dbl>, Travel_time <dbl>,
## #   Mode_of_transport <chr>, Safe <chr>
```

8.3 Arranging Rows

We can order the rows of a data frame using the `arrange` verb. Let's do this on people's height:

```
arrange(qd, Height)
```

```
## # A tibble: 18 x 19
##    Height   Age Sex    BirthPlace SiblingsNo EatMeat DrinkCoffee LikeBeer Sports
##     <dbl> <dbl> <chr>  <chr>           <dbl> <chr>   <chr>       <chr>    <chr>
##  1    158  24.2 Female China               0 Yes     Yes         Yes      Yes
##  2    162  24   Female Home Kong           1 Yes     No          No       No
##  3    163  21   Female CHINA               0 Yes     Yes         No       Yes
##  4    165  28   Female india               0 Yes     Yes         Yes      No
##  5    168  23   Male   Surrey              2 Yes     Yes         No       No
##  6    168  22   Female Malaysia            2 Yes     Yes         No       Yes
##  7    170  23   Female essex               1 Yes     Yes         No       Yes
##  8    170  22.1 Female Plymouth            2 Yes     Yes         No       No
##  9    170  12   Male   Hong Kong           1 Yes     No          No       Yes
## 10    171  35.5 Male   Plymouth            6 Yes     Yes         No       Yes
## 11    175  22.4 Male   Exeter              1 Yes     No          No       No
## 12    177  22.2 Male   Bournemou~          1 Yes     Yes         Yes      No
## 13    180  30.1 Male   Athens              0 Yes     Yes         Yes      Yes
## 14    182  25   Male   Nigeria             4 Yes     No          No       Yes
## 15    185  21   Male   China               0 Yes     Yes         Yes      Yes
## 16    186  24.8 Male   Dusseldorf          3 Yes     No          Yes      Yes
## 17    187  24.8 Male   USA                 4 Yes     Yes         Yes      Yes
## 18    188  22.4 Male   London              1 Yes     Yes         No       No
## # ... with 10 more variables: Driver <chr>, LeftHanded <chr>, Abroad <chr>,
## #   Sleep <dbl>, Rent <dbl>, Happy_accommodation <chr>, Distance <dbl>,
## #   Travel_time <dbl>, Mode_of_transport <chr>, Safe <chr>
```

Height is arranged in ascending order. We can also arrange in descending order:

```
arrange(qd, desc(Height))
```

```
## # A tibble: 18 x 19
##    Height   Age Sex    BirthPlace SiblingsNo EatMeat DrinkCoffee LikeBeer Sports
##     <dbl> <dbl> <chr>  <chr>           <dbl> <chr>   <chr>       <chr>    <chr>
##  1    188  22.4 Male   London              1 Yes     Yes         No       No
##  2    187  24.8 Male   USA                 4 Yes     Yes         Yes      Yes
##  3    186  24.8 Male   Dusseldorf          3 Yes     No          Yes      Yes
##  4    185  21   Male   China               0 Yes     Yes         Yes      Yes
##  5    182  25   Male   Nigeria             4 Yes     No          No       Yes
##  6    180  30.1 Male   Athens              0 Yes     Yes         Yes      Yes
##  7    177  22.2 Male   Bournemou~          1 Yes     Yes         Yes      No
##  8    175  22.4 Male   Exeter              1 Yes     No          No       No
##  9    171  35.5 Male   Plymouth            6 Yes     Yes         No       Yes
## 10    170  23   Female essex               1 Yes     Yes         No       Yes
## 11    170  22.1 Female Plymouth            2 Yes     Yes         No       No
## 12    170  12   Male   Hong Kong           1 Yes     No          No       Yes
## 13    168  23   Male   Surrey              2 Yes     Yes         No       No
## 14    168  22   Female Malaysia            2 Yes     Yes         No       Yes
## 15    165  28   Female india               0 Yes     Yes         Yes      No
## 16    163  21   Female CHINA               0 Yes     Yes         No       Yes
## 17    162  24   Female Home Kong           1 Yes     No          No       No
## 18    158  24.2 Female China               0 Yes     Yes         Yes      Yes
## # ... with 10 more variables: Driver <chr>, LeftHanded <chr>, Abroad <chr>,
## #   Sleep <dbl>, Rent <dbl>, Happy_accommodation <chr>, Distance <dbl>,
## #   Travel_time <dbl>, Mode_of_transport <chr>, Safe <chr>
```

We can arrange on height and then on travel time (for example):

```
qd2 <- arrange(qd, Height, Travel_time)
qd2 # Not all variables shown
```

```
## # A tibble: 18 x 19
##    Height   Age Sex    BirthPlace SiblingsNo EatMeat DrinkCoffee LikeBeer Sports
##     <dbl> <dbl> <chr>  <chr>           <dbl> <chr>   <chr>       <chr>    <chr>
##  1    158  24.2 Female China               0 Yes     Yes         Yes      Yes
##  2    162  24   Female Home Kong           1 Yes     No          No       No
##  3    163  21   Female CHINA               0 Yes     Yes         No       Yes
##  4    165  28   Female india               0 Yes     Yes         Yes      No
##  5    168  23   Male   Surrey              2 Yes     Yes         No       No
##  6    168  22   Female Malaysia            2 Yes     Yes         No       Yes
##  7    170  12   Male   Hong Kong           1 Yes     No          No       Yes
##  8    170  23   Female essex               1 Yes     Yes         No       Yes
##  9    170  22.1 Female Plymouth            2 Yes     Yes         No       No
## 10    171  35.5 Male   Plymouth            6 Yes     Yes         No       Yes
## 11    175  22.4 Male   Exeter              1 Yes     No          No       No
## 12    177  22.2 Male   Bournemou~          1 Yes     Yes         Yes      No
## 13    180  30.1 Male   Athens              0 Yes     Yes         Yes      Yes
## 14    182  25   Male   Nigeria             4 Yes     No          No       Yes
## 15    185  21   Male   China               0 Yes     Yes         Yes      Yes
## 16    186  24.8 Male   Dusseldorf          3 Yes     No          Yes      Yes
## 17    187  24.8 Male   USA                 4 Yes     Yes         Yes      Yes
## 18    188  22.4 Male   London              1 Yes     Yes         No       No
## # ... with 10 more variables: Driver <chr>, LeftHanded <chr>, Abroad <chr>,
## #   Sleep <dbl>, Rent <dbl>, Happy_accommodation <chr>, Distance <dbl>,
## #   Travel_time <dbl>, Mode_of_transport <chr>, Safe <chr>
```

```
select(qd2, Height, Travel_time)
```

```
## # A tibble: 18 x 2
##    Height Travel_time
##     <dbl>       <dbl>
##  1    158           6
##  2    162          25
##  3    163          10
##  4    165          10
##  5    168           5
##  6    168          12
##  7    170          15
##  8    170          20
##  9    170          40
## 10    171          20
## 11    175          90
## 12    177           6
## 13    180           3
## 14    182          15
## 15    185          20
## 16    186          23
## 17    187          15
## 18    188           5
```

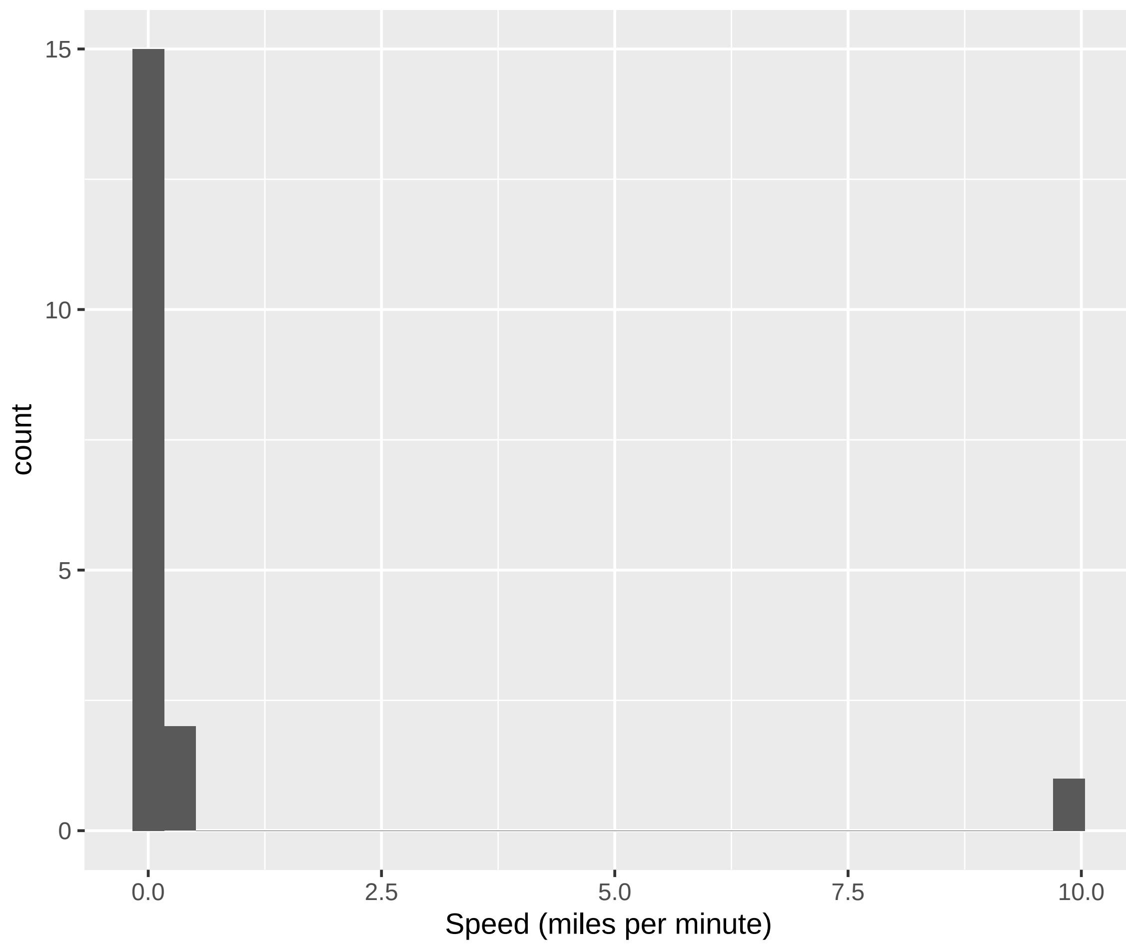
8.4 Making New Variables

We can add a new column to a data frame using the `mutate` verb. For example, we can work out travel speed as `Distance` divided by `Travel_time`. The code also illustrates the production of a **histogram** using `ggplot2`:

```
qd3 <- mutate(qd, Speed = Distance / Travel_time)
qd3$Speed
```

```
##  [1] 0.05000000 0.16000000 0.03333333 0.10000000 0.50000000 0.04000000
##  [7] 0.50000000 0.10000000 0.03333333 0.10000000 0.04400000 0.04166667
## [13] 0.05000000 0.06666667 0.07500000 9.90000000 0.06086957 0.05000000
```

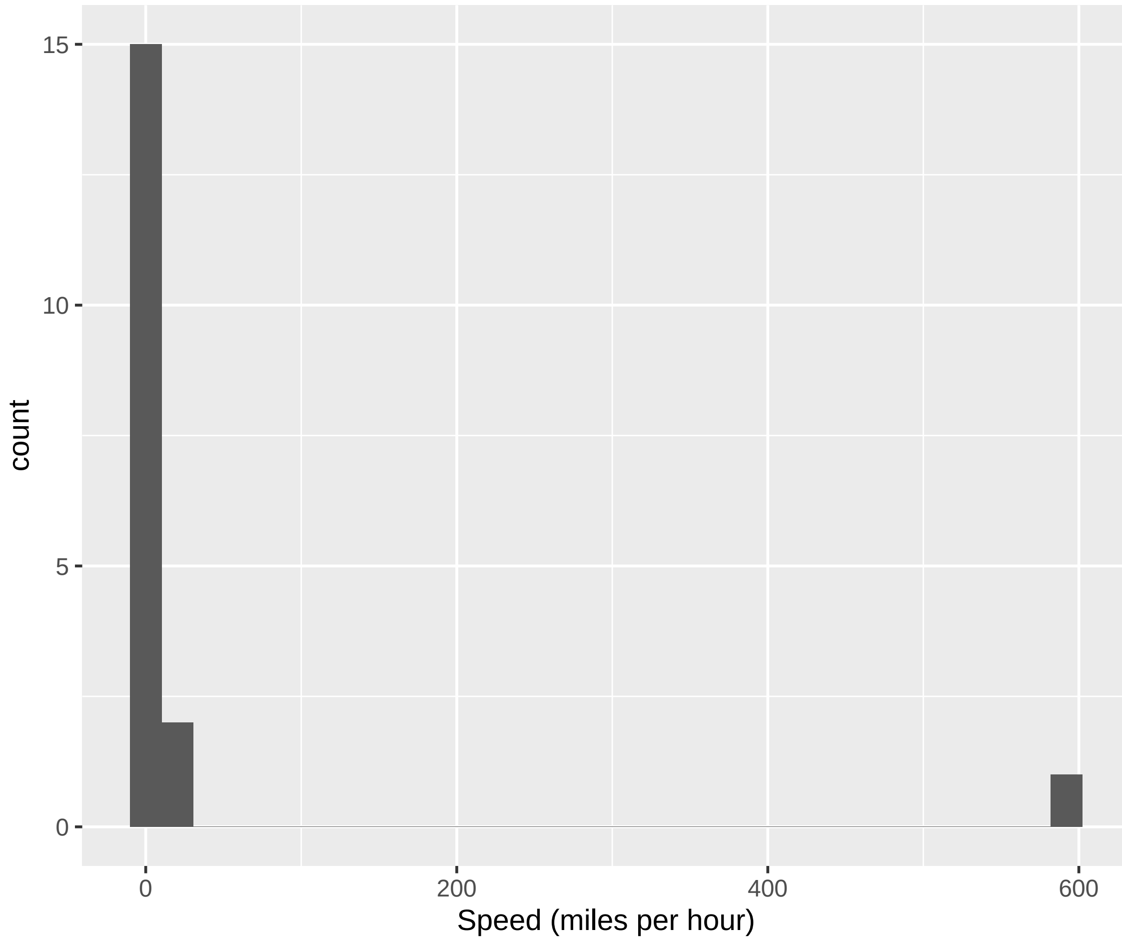
```
#
# Produce a histogram
#
require(ggplot2)
ggplot(qd3, aes(x = Speed)) +
    geom_histogram() +
    labs(x = "Speed (miles per minute)")
```




```
#
# Miles per hour
#
qd3 <- mutate(qd, Speed = Distance / Travel_time, Speed_mph = 60 * Speed)
qd3$Speed_mph
```

```
## [1]   3.000000   9.600000   2.000000   6.000000  30.000000   2.400000
## [7]  30.000000   6.000000   2.000000   6.000000   2.640000   2.500000
## [13]   3.000000   4.000000   4.500000 594.000000   3.652174   3.000000
```

```
ggplot(qd3, aes(x = Speed_mph)) +
    geom_histogram() +
    labs(x = "Speed (miles per hour)")
```



The height of bars of a histogram represent the number of data points that fall into the interval (or bin) at the base of the bar.

Notice how in `mutate` we can refer to variables that have just been created.

Let us examine the command

`ggplot(qd3, aes(x = Speed)) + geom_histogram() + labs(x = "Speed (miles per minute)")`

Fist the basic plot aesthetic is set up by specifying that `Speed` from the data frame `qd3` will be plotted on the `x` axis. Then a histogram geometry or layer is specified by `geom_histogram`. Finally, we label the axis.

We can renames a variable:

```
qd4 <- rename(qd3, s = Speed_mph)
qd3$Speed_mph
```

```
## [1]   3.000000   9.600000   2.000000   6.000000  30.000000   2.400000
## [7]  30.000000   6.000000   2.000000   6.000000   2.640000   2.500000
## [13]   3.000000   4.000000   4.500000 594.000000   3.652174   3.000000
```

```r
qd4$s
```

```
##  [1]   3.000000   9.600000   2.000000   6.000000  30.000000   2.400000
##  [7]  30.000000   6.000000   2.000000   6.000000   2.640000   2.500000
## [13]   3.000000   4.000000   4.500000 594.000000   3.652174   3.000000
```

```r
qd4$Speed_mph # Shouldn't exist.  Why?
```

```
## NULL
```

8.5 Summarizing Data

We can summarie data using the `summarise` verb:

```r
summarise(qd, ave = mean(Height))
```

```
## # A tibble: 1 x 1
##     ave
##   <dbl>
## 1  174.
```

```r
summarise(qd, sd = sd(Height))
```

```
## # A tibble: 1 x 1
##      sd
##   <dbl>
## 1  9.29
```

```r
summarise(qd, ave = mean(Height), sd = sd(Height))
```

```
## # A tibble: 1 x 2
##     ave    sd
##   <dbl> <dbl>
## 1  174.  9.29
```

`count` counts the number of rows with each unique value of a variable:

```r
count(qd, Sex)
```

```
## # A tibble: 2 x 2
##   Sex        n
##   <chr>  <int>
## 1 Female     7
## 2 Male      11
```

8.6 Grouping Data

The function `group_by` allows us to produce summary statistics broken down by groups:

```r
qd_by_Sex <- group_by(qd, Sex)
summarise(qd_by_Sex, ave = mean(Height), sd = sd(Height))
```

```
## # A tibble: 2 x 3
##   Sex      ave    sd
##   <chr>  <dbl> <dbl>
## 1 Female  165.  4.49
## 2 Male    179   7.25
```

8.7 Chaining Commands using a Pipe

The above summary statistics for each group were produced in two stages. First, we grouped the data by the variable `Sex`. Then we worked out the summary statistics.

The "pipe" `%>%`, like an arrow, allows us to chain commands together, reflecting better our though processes and workflow. Essentially,

`x %>% f(y)`

is the same as

`f(x, y)`

Here is the above example done using `%>%`:

```
qd %>% group_by(Sex) %>% summarise(ave = mean(Height), sd = sd(Height))
```

```
## # A tibble: 2 x 3
##   Sex      ave    sd
##   <chr>  <dbl> <dbl>
## 1 Female  165.  4.49
## 2 Male    179   7.25
```

Here is another example:

```
qd %>% group_by(Sex) %>%
   summarise(ave = mean(Height), sd = sd(Height), n = n()) %>%
   arrange(desc(sd))
```

```
## # A tibble: 2 x 4
##   Sex      ave    sd     n
##   <chr>  <dbl> <dbl> <int>
## 1 Male    179   7.25    11
## 2 Female  165.  4.49     7
```

The function `n` finds the number of values in a vector.

8.8 Combining Data Sets

Sometimes it is important to be able to combine data that are supplied in different data sets. Here is a very small example.

Let's define two simple data frames `a` and `b` with one variable `x1` (holding a person identifier) in common:

```
a <- data_frame(x1 = c("A", "B", "C"), x2 = c(1, 2, 3))
# This using dplyr's data_frame function which is more efficient than
# the base R function data.frame
a
```

```
## # A tibble: 3 x 2
##   x1       x2
##   <chr> <dbl>
## 1 A         1
## 2 B         2
## 3 C         3
```

```
b <- data_frame(x1 = c("A", "B", "D"), x3 = c(4, 5, 6))
b
```

```
## # A tibble: 3 x 2
##   x1       x3
```

```
##   <chr> <dbl>
## 1 A         4
## 2 B         5
## 3 D         6
```

We can combine these in four different ways. Do you understand what is happening?

- Join matching rows from `b` to `a`:

```
left_join(a, b, by = "x1")
```

```
## # A tibble: 3 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
## 3 C         3    NA
```

- Join matching rows from `a` to `b`:

```
right_join(a, b, by = "x1")
```

```
## # A tibble: 3 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
## 3 D        NA     6
```

- Retain only the rows in both data frames:

```
inner_join(a, b, by = "x1")
```

```
## # A tibble: 2 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
```

- Retain all values and all rows:

```
full_join(a, b, by = "x1")
```

```
## # A tibble: 4 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
## 3 C         3    NA
## 4 D        NA     6
```

Note that it's possible to join on variables with different names in the two data.frames:

```
a <- data_frame(x1 = c("A", "B", "C"), x2 = c(1, 2, 3))
b <- data_frame(x1 = c("A", "B", "D"), x3 = c(4, 5, 6))
#
left_join(a, b, by = "x1")
```

```
## # A tibble: 3 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
```

```
## 1 A          1     4
## 2 B          2     5
## 3 C          3    NA
```

```
right_join(a, b, by = "x1")
```

```
## # A tibble: 3 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
## 3 D        NA     6
```

```
inner_join(a, b, by = "x1")
```

```
## # A tibble: 2 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
```

```
full_join(a, b, by = "x1")
```

```
## # A tibble: 4 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
## 3 C         3    NA
## 4 D        NA     6
```

```
#
# Different variable name
#
b_var <- data_frame(x1_var = c("A", "B", "D"), x3 = c(4, 5, 6))
#
left_join(a, b_var, by = c("x1" = "x1_var"))
```

```
## # A tibble: 3 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
## 3 C         3    NA
```

```
right_join(a, b_var, by = c("x1" = "x1_var"))
```

```
## # A tibble: 3 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
## 3 D        NA     6
```

```
inner_join(a, b_var, by = c("x1" = "x1_var"))
```

```
## # A tibble: 2 x 3
##   x1       x2    x3
```

```
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
```

```r
full_join(a, b_var, by = c("x1" = "x1_var"))
```

```
## # A tibble: 4 x 3
##   x1       x2    x3
##   <chr> <dbl> <dbl>
## 1 A         1     4
## 2 B         2     5
## 3 C         3    NA
## 4 D        NA     6
```

Of course, it's always possible to rename a variable so that they have a common name. As another example of renaming a variable, please consider:

```r
b_var
```

```
## # A tibble: 3 x 2
##   x1_var    x3
##   <chr>  <dbl>
## 1 A          4
## 2 B          5
## 3 D          6
```

```r
b_var %>% rename(x1.var = x1_var)
```

```
## # A tibble: 3 x 2
##   x1.var    x3
##   <chr>  <dbl>
## 1 A          4
## 2 B          5
## 3 D          6
```

8.9 Reshaping Data

The illustrative data in the file `wide_data.csv` comprises observations taken at three time periods on four patients.

- Open the file `wide_data.csv` using Excel and look at the data structure.
- Read in the data. You may have to specify a working directory first:

```r
require(readr)
wide <- read_csv("wide_data.csv")
wide
```

```
## # A tibble: 4 x 4
##   Patient Observation_1 Observation_2 Observation_3
##   <chr>           <dbl>         <dbl>         <dbl>
## 1 A                  20            43            70
## 2 B                  12            32            61
## 3 C                  27            55            77
## 4 D                  24            49            68
```

```r
names(wide)
```

```
## [1] "Patient"       "Observation_1" "Observation_2" "Observation_3"
```

It is very often much more convenient to have one variable containing the observation times and another containing the values. The function `gather` from `tidyr` allows us to reformat the data in this way:

```
require(tidyr)
citation("tidyr")
```

```
##
## To cite package 'tidyr' in publications use:
##
##   Hadley Wickham (2021). tidyr: Tidy Messy Data. R package version
##   1.1.4. https://CRAN.R-project.org/package=tidyr
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {tidyr: Tidy Messy Data},
##     author = {Hadley Wickham},
##     year = {2021},
##     note = {R package version 1.1.4},
##     url = {https://CRAN.R-project.org/package=tidyr},
##   }
```

```
long <- gather(wide, "Observation", "Value", 2:4)
# Place the information in columns 2 to 4 into Value,
# with a column Observation recording the observation number
long
```

```
## # A tibble: 12 x 3
##    Patient Observation   Value
##    <chr>   <chr>         <dbl>
##  1 A       Observation_1    20
##  2 B       Observation_1    12
##  3 C       Observation_1    27
##  4 D       Observation_1    24
##  5 A       Observation_2    43
##  6 B       Observation_2    32
##  7 C       Observation_2    55
##  8 D       Observation_2    49
##  9 A       Observation_3    70
## 10 B       Observation_3    61
## 11 C       Observation_3    77
## 12 D       Observation_3    68
```

We can go the other way using `spread`:

```
spread(long, "Observation", "Value")
```

```
## # A tibble: 4 x 4
##   Patient Observation_1 Observation_2 Observation_3
##   <chr>           <dbl>         <dbl>         <dbl>
## 1 A                  20            43            70
## 2 B                  12            32            61
## 3 C                  27            55            77
## 4 D                  24            49            68
```

Once data is in the above “long” format

```r
long
```

```
## # A tibble: 12 x 3
##    Patient Observation   Value
##    <chr>   <chr>         <dbl>
##  1 A       Observation_1    20
##  2 B       Observation_1    12
##  3 C       Observation_1    27
##  4 D       Observation_1    24
##  5 A       Observation_2    43
##  6 B       Observation_2    32
##  7 C       Observation_2    55
##  8 D       Observation_2    49
##  9 A       Observation_3    70
## 10 B       Observation_3    61
## 11 C       Observation_3    77
## 12 D       Observation_3    68
```

it can be summarised and plotted easily:

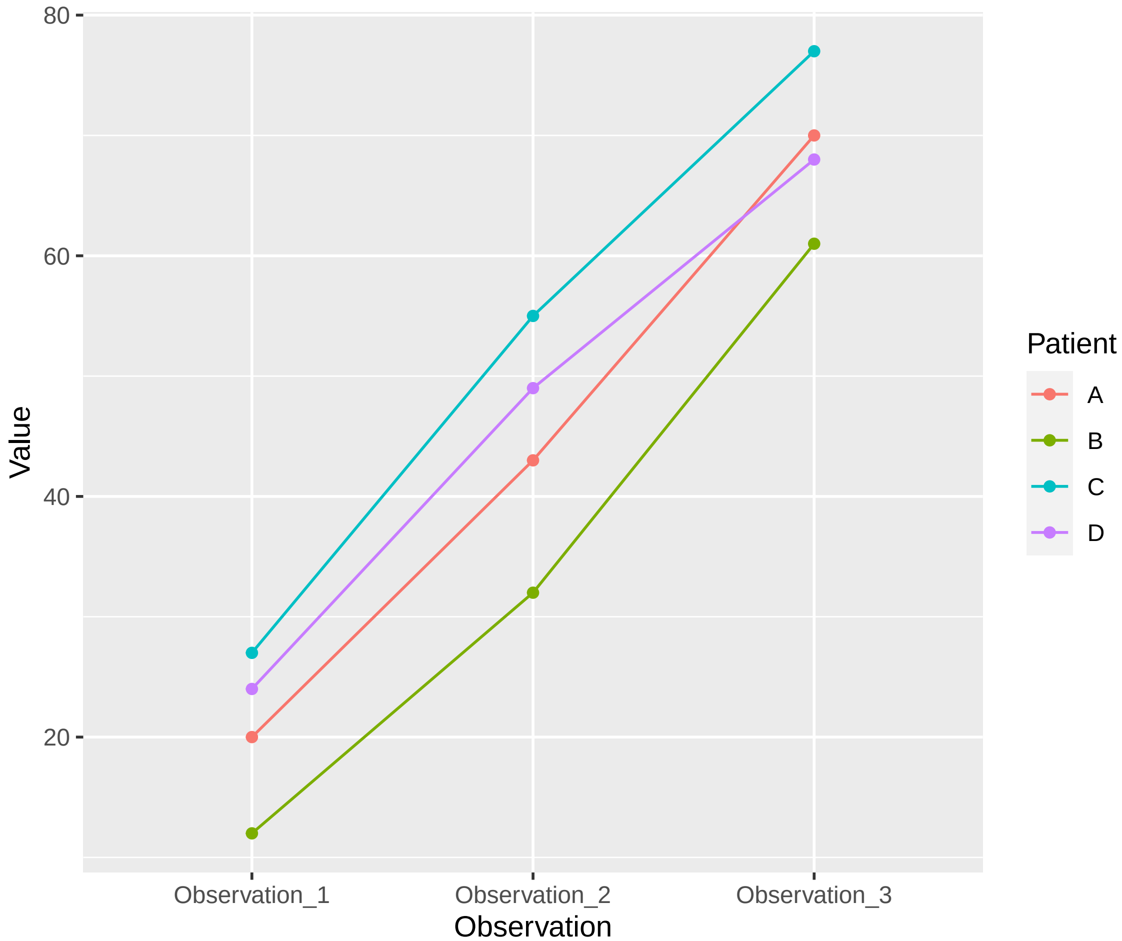
```r
long %>% group_by(Patient) %>% summarise(maximum = max(Value))
```

```
## # A tibble: 4 x 2
##   Patient maximum
##   <chr>     <dbl>
## 1 A            70
## 2 B            61
## 3 C            77
## 4 D            68
```

```r
long %>% group_by(Observation) %>% summarise(spread = IQR(Value))
```

```
## # A tibble: 3 x 2
##   Observation   spread
##   <chr>          <dbl>
## 1 Observation_1   6.75
## 2 Observation_2  10.2
## 3 Observation_3   5.5
```

```r
#
ggplot(long, aes(x = Observation, y = Value, group = Patient, colour = Patient)) +
    geom_point() +
    geom_line()
```



In `ggplot` the coloured lines join the observations from each patient. To achieve this we said that the data were **group**ed by `Patient` and that the **colour**s should correspond to `Patient`.

8.10 The useful `ifelse` function

Say that `Observation_1` was observed at time 5 months, `Observation_2` was observed at time 8 months and `Observation_3` was observed at time 12 months.

We could define a variable containing this information using the very useful `ifelse` function. In general `ifelse(test, yes, no)` considers every element of `test` in turn and return `yes` if the `test` element is `TRUE` and `no` otherwise. For example,

```
ifelse(qd$Height > 170, "tall", "less tall")
```

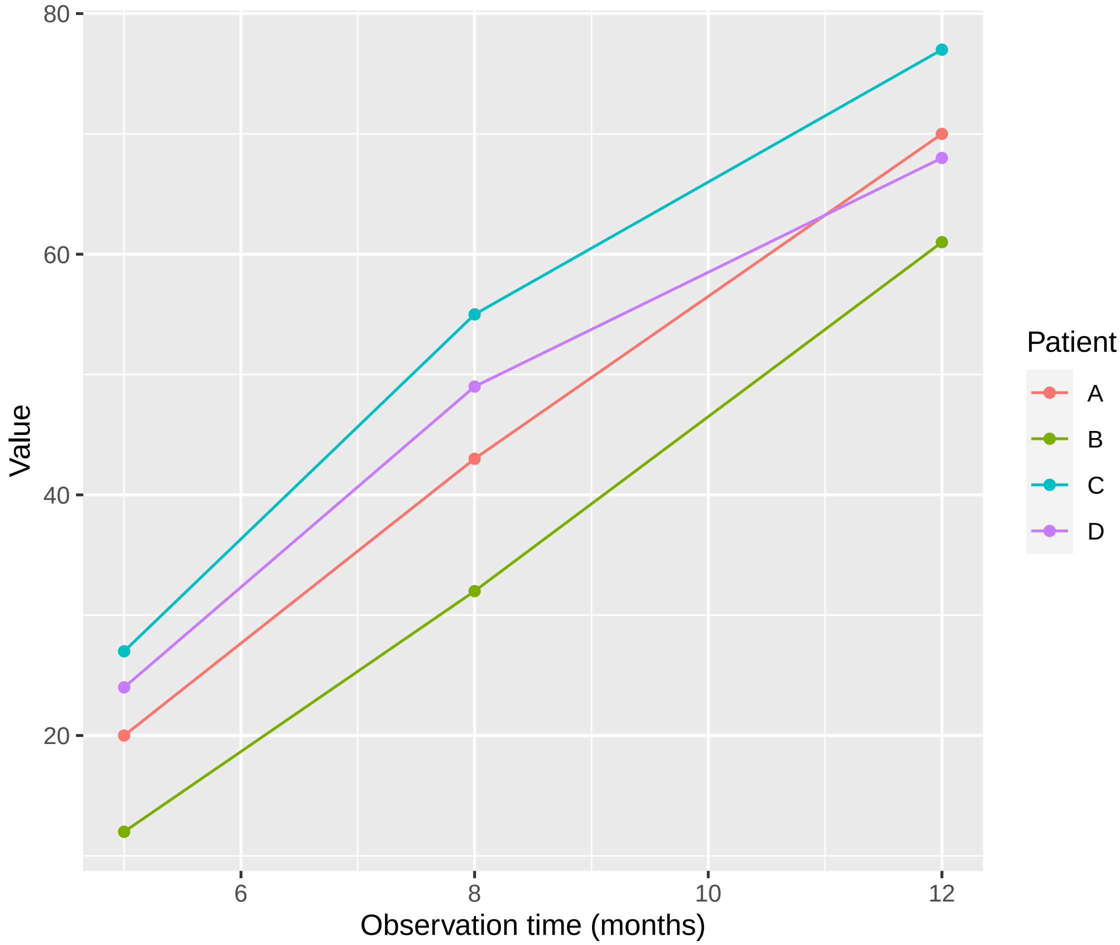
```
##  [1] "less tall" "tall"      "tall"      "tall"      "less tall" "tall"
##  [7] "tall"      "tall"      "less tall" "less tall" "less tall" "less tall"
## [13] "less tall" "less tall" "tall"      "less tall" "tall"      "tall"
```

The beauty of `ifelse` is that it can be applied to many values, such as each value of a vector.

To add the observation times in months to the data frame, we need to use a nested `ifelse`:

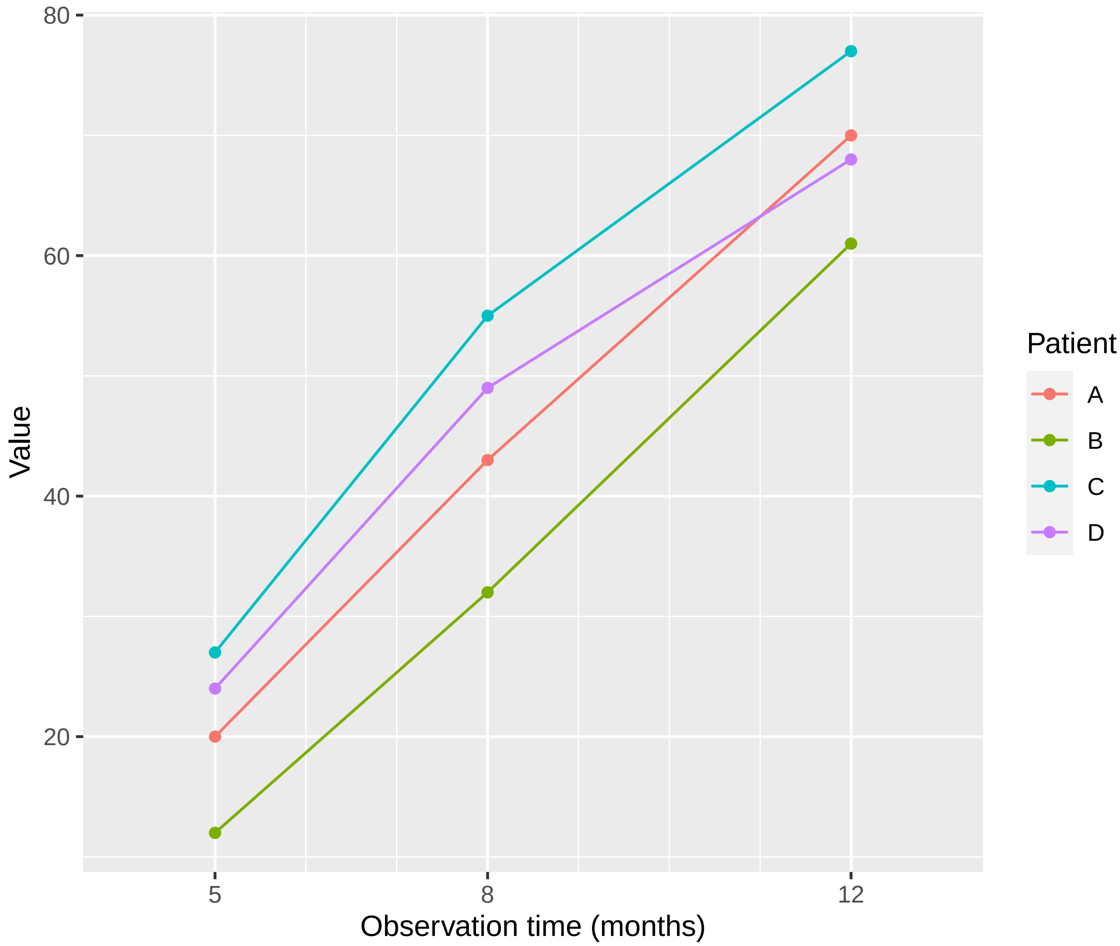
```
long_with_times <- long %>% mutate(Observation_time =
                                     ifelse(Observation == "Observation_1", 5,
                                        ifelse(Observation == "Observation_2", 8, 12)))
#
ggplot(long_with_times,
```

```
       aes(x = Observation_time, y = Value, group = Patient, colour = Patient)) +
    geom_point() +
    geom_line() +
    labs(x = "Observation time (months)")
```



We can customize the x-axis, for example:

```
ggplot(long_with_times,
       aes(x = Observation_time, y = Value, group = Patient, colour = Patient)) +
    geom_point() +
    geom_line() +
    labs(x = "Observation time (months)") +
    scale_x_continuous(breaks = c(5, 8, 12), minor_breaks = 5:12, limits = c(4,13))
```



8.11 More Details

The Data Wrangling Cheat Sheet provides full coverage of `dplyr` and `tidyr` and is available from

https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

9 Other Graphical Displays

9.1 Plots with Logatithmic Scales

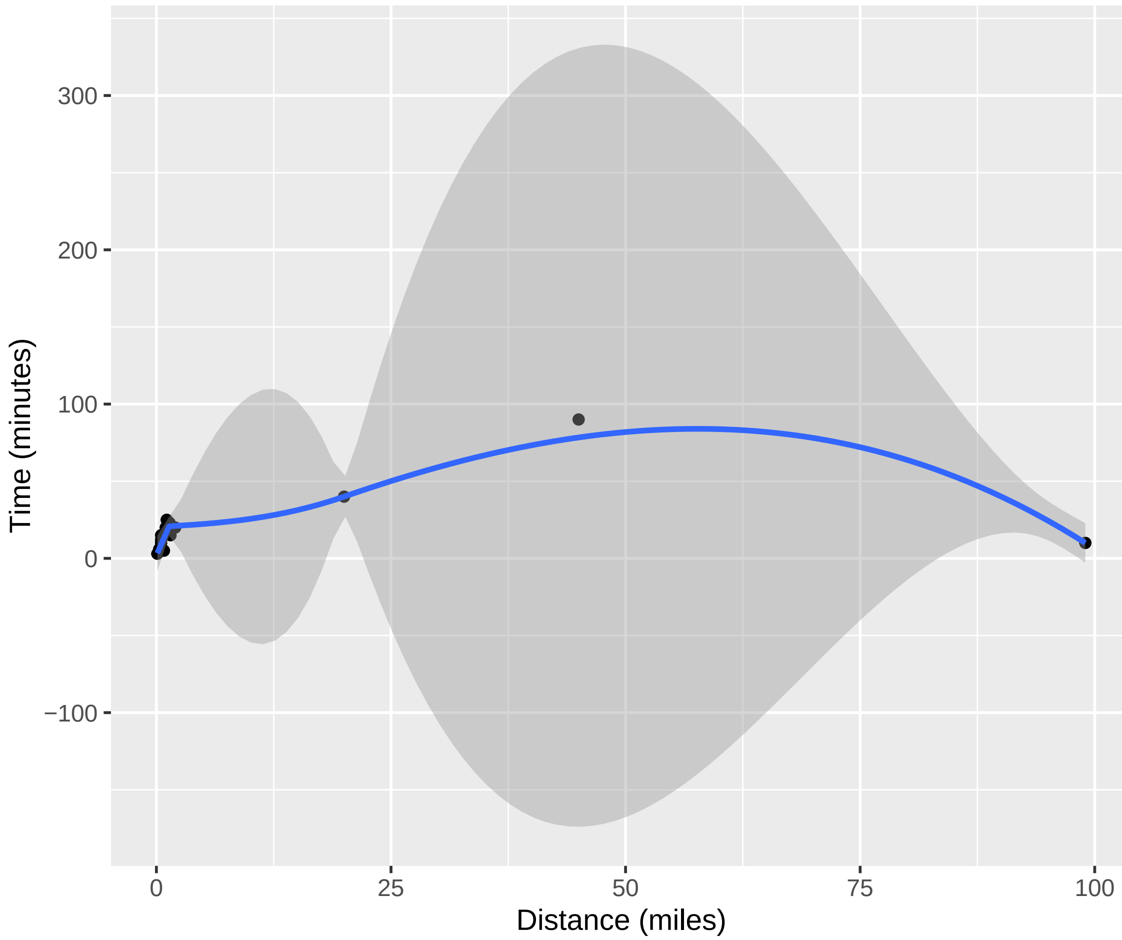
If data comprise a lot of small values together with large values, it is often useful to plot them using a logarithmic scale, so that detail is not lost. Here is an example.

```
names(qd)
```

```
##  [1] "Height"                 "Age"                    "Sex"
##  [4] "BirthPlace"             "SiblingsNo"             "EatMeat"
##  [7] "DrinkCoffee"            "LikeBeer"               "Sports"
## [10] "Driver"                 "LeftHanded"             "Abroad"
## [13] "Sleep"                  "Rent"                   "Happy_accommodation"
## [16] "Distance"               "Travel_time"            "Mode_of_transport"
## [19] "Safe"
```

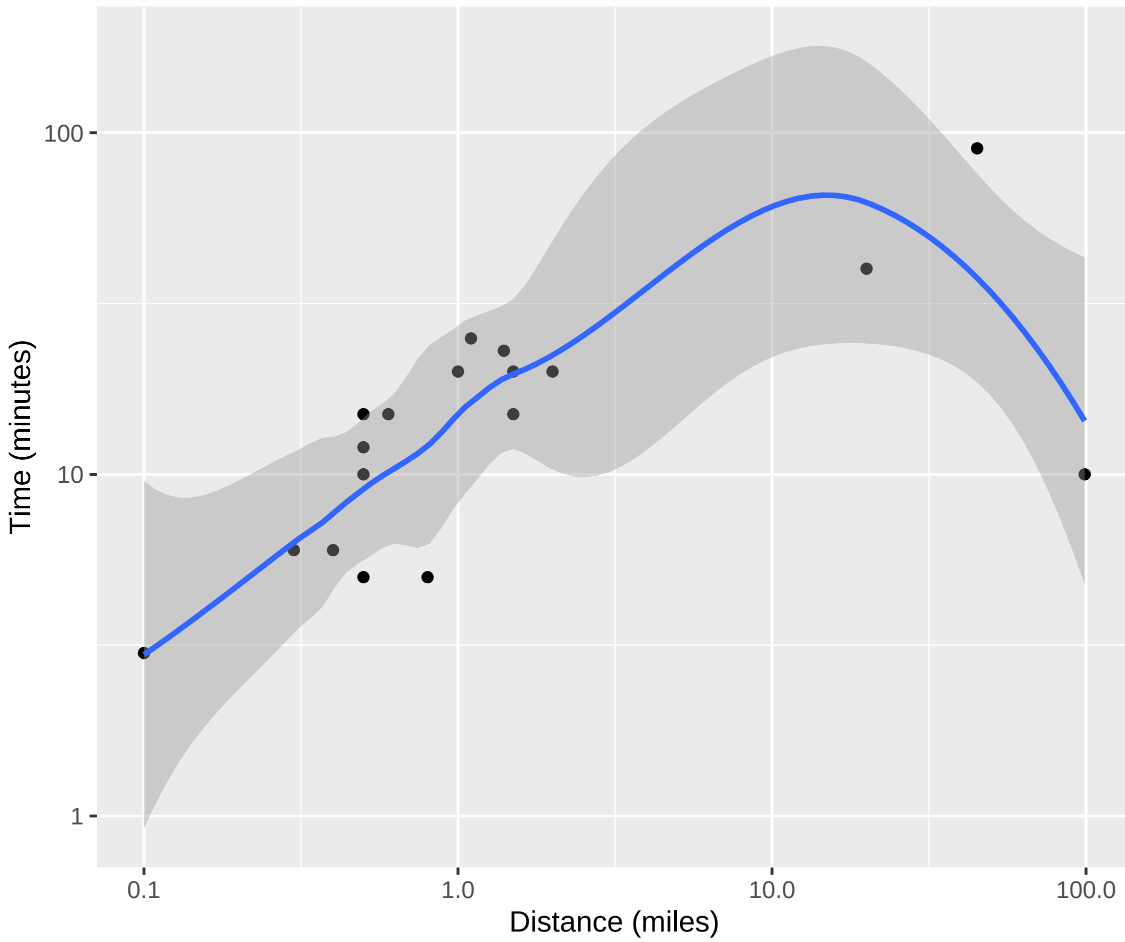
```
require(ggplot2)
ggplot(qd, aes(x = Distance, y = Travel_time)) +
```

```
    geom_point() +
    geom_smooth() +
    labs(x = "Distance (miles)", y = "Time (minutes)")
```



There are quite a few people who have short journeys. The detail of these journeys is lost due to the very long journeys. We will therefore use a logarithmic scale on both axes:

```
ggplot(qd, aes(x = Distance, y = Travel_time)) +
    geom_point() +
    geom_smooth() +
    scale_x_log10() +
    scale_y_log10() +
    labs(x = "Distance (miles)", y = "Time (minutes)")
```



Please note that the units on the scales remain the same. It's just the scales themselves that change. They go up in powers!

The detail is now visible across the whole range of distance and time values.

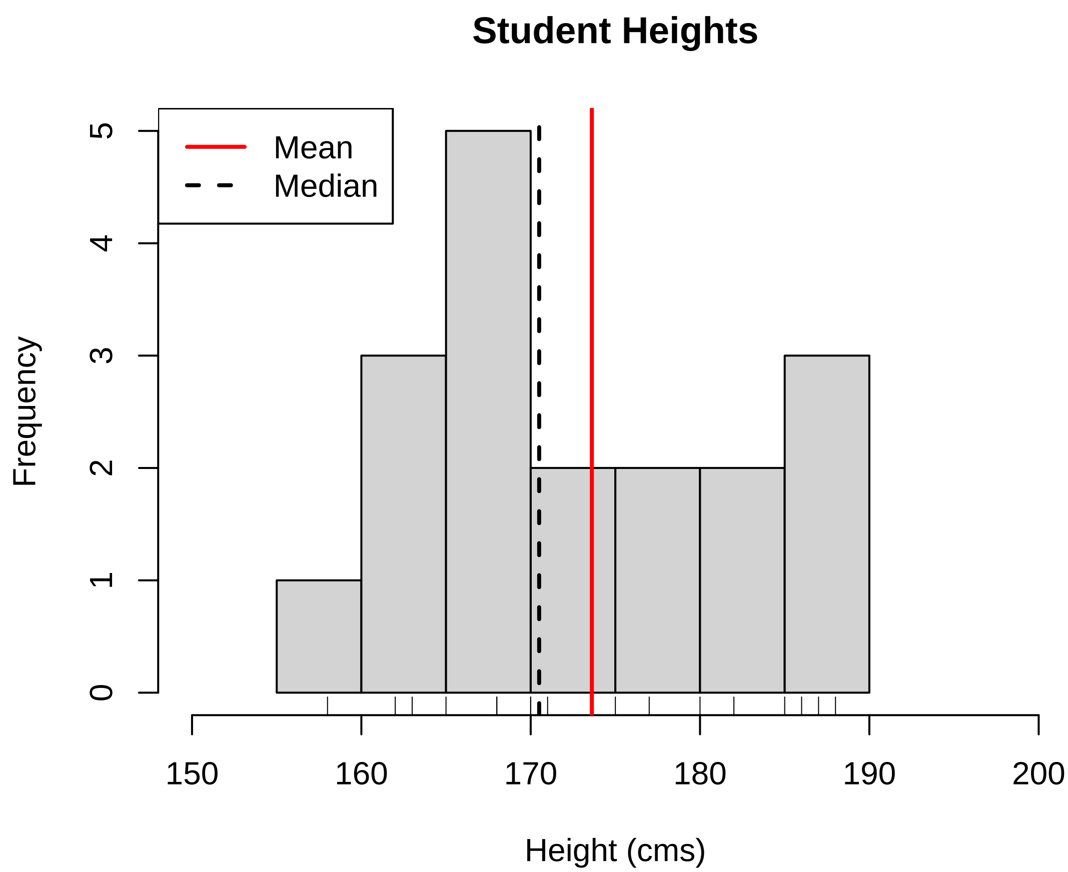
Comment on this graph. Why is it interesting?

Other useful graphical displays are the histogram, which we have met briefly, and the box plot.

9.2 Histograms

To produce a histogram in base R, we can use the function `hist`:

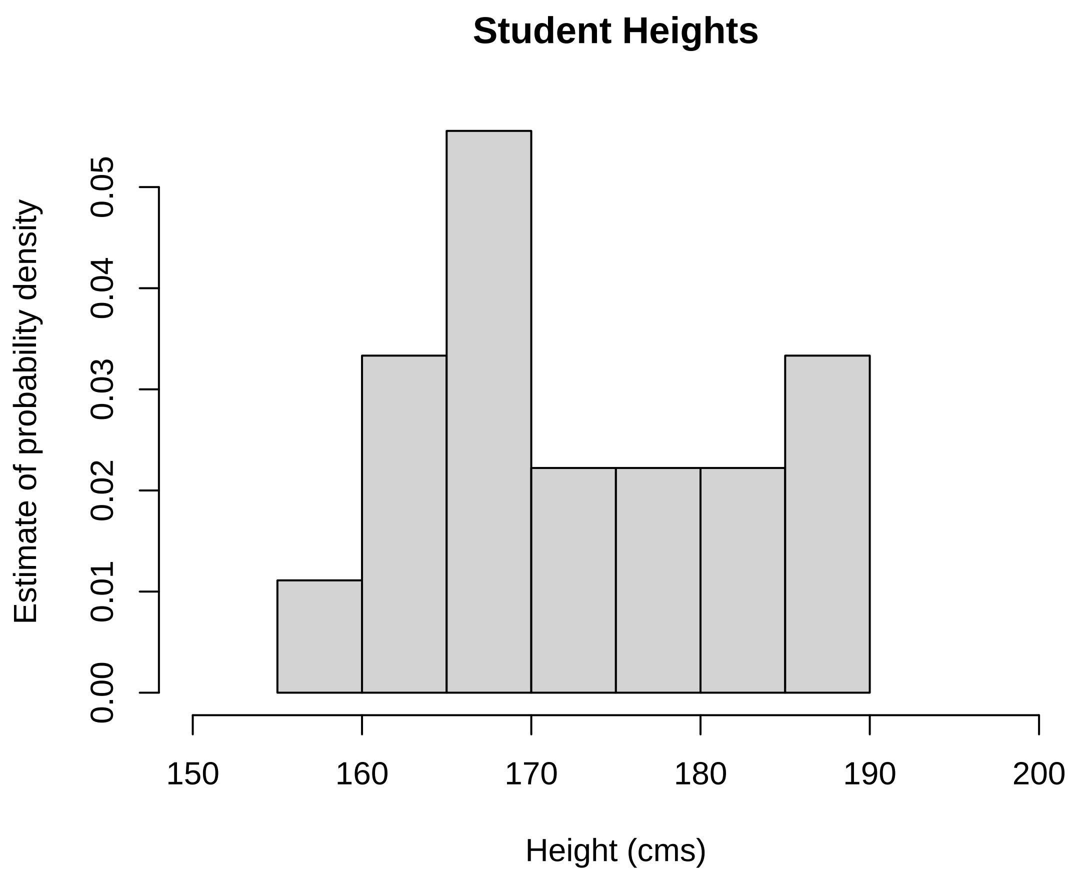
```
hist(qd$Height, xlab = "Height (cms)", main = "Student Heights", xlim = c(150, 200))
rug(qd$Height)
# We can see which data point lies in each interval or bin
abline(v = mean(qd$Height), col = "red", lwd = 2)
abline(v = median(qd$Height), lty = 2, lwd = 2)
legend("topleft", legend = c("Mean", "Median"),
       lty = c(1,2), col = c("red", "black"), lwd = 2)
```



`rug` adds a carpet of tick marks of the data to the plot.

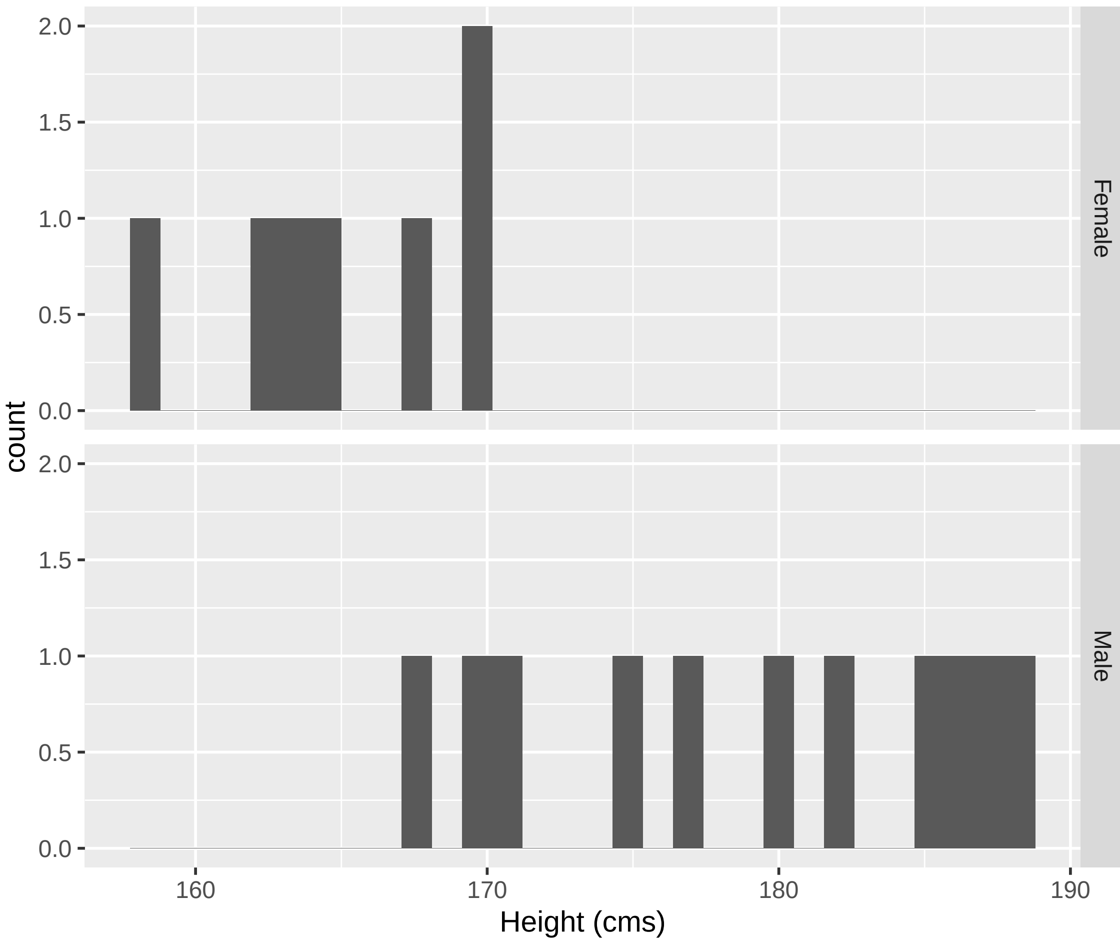
A histogram the area under which is one can be produced by setting `freq = FALSE`

```
hist(qd$Height, xlab = "Height (cms)", main = "Student Heights",
     ylab = "Estimate of probability density",
     xlim = c(150, 200),
     freq = FALSE)
```



We have already seen how to produce a histogram in `ggplot2`. Here we produce a histogram for each gender:

```
require(ggplot2)
ggplot(qd, aes(x = Height)) +
    geom_histogram() +
    labs(x = "Height (cms)") +
    facet_grid(Sex ~ .)
```



In general, males are taller than females, of whom there are just a few.

9.3 Boxplots

Before discussing the boxplot we need some definitions.

The upper quatile is a value such that 25% of the data lie above it and 75% lie below it. It is sometimes called the 0.75-quantile. The lower quatile is a value such that 25% of the data lie below it and 75% lie above it. It is sometimes called the 0.25-quantile. The sample interquartile range is the difference between the upper quartile and the lower quartile of the sample. We illustrate these values below when we discuss the boxplot. Here we calculate them:

```
sort(qd$Height) # The sorted data so that results can be checked
```

```
##  [1] 158 162 163 165 168 168 170 170 170 171 175 177 180 182 185 186 187 188
```

```
n_height <- length(qd$Height)
n_height
```

```
## [1] 18
```

```
n_height / 4
```

```
## [1] 4.5
```

```
3*n_height / 4
```

```
## [1] 13.5
```

```r
quantile(qd$Height, probs = 0.25) # Lower quartile
```

```
## 25%
## 168
```

```r
quantile(qd$Height, probs = 0.75) # Upper quartile
```

```
##   75%
## 181.5
```

```r
quantile(qd$Height, probs = c(0.25,0.75)) # Both lower and upper quartile
```

```
##   25%   75%
## 168.0 181.5
```

```r
diff(quantile(qd$Height, probs = c(0.25,0.75)))  # The difference between these values
```

```
##  75%
## 13.5
```

```r
IQR(qd$Height) # The interquartile range
```

```
## [1] 13.5
```

The median is the 0.5-quantile:

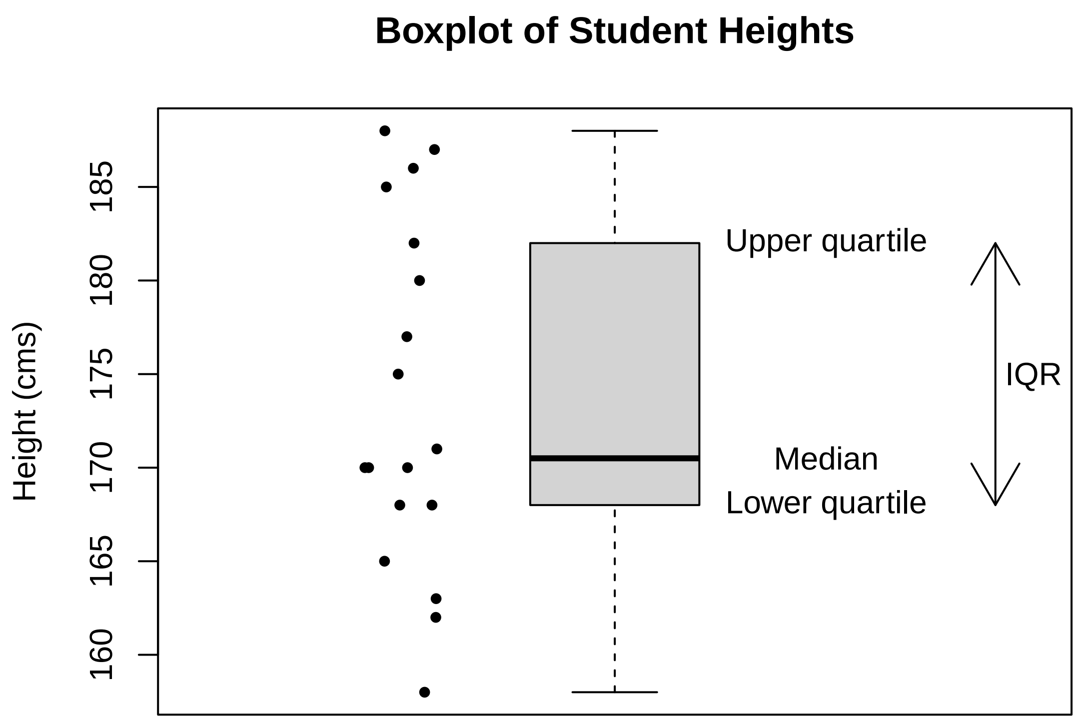
```r
quantile(qd$Height, probs = 0.5)
```

```
##   50%
## 170.5
```

```r
median(qd$Height)
```

```
## [1] 170.5
```

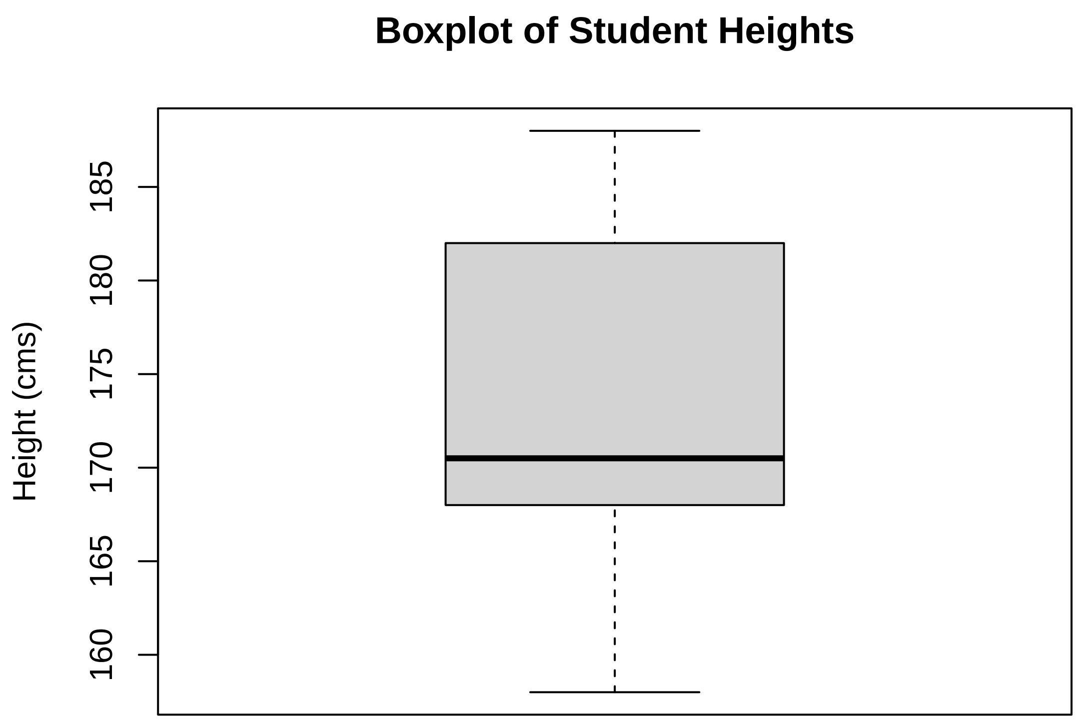
This figure and the following text provides an explanation of a boxplot (sometimes called a box and whisker plot):



The box itself shows the lower quartile, median and upper quartile. The difference between these two quartiles provides us with a measure of spread of the data and its value is known as the sample interquartile range (IQR). The data points have been added using filled points to illustrate how the parts of the boxplot correspond to the data. The data points have been jittered horizontally so that all points are visible. These data points will **not** appear on your boxplot. The whiskers indicate the highest/lowest value with distance from the upper/lower quartile no more that 1.5 $\times$ the sample interquartile range. Values beyond these whiskers (here there are none) are indicated separately using dots.

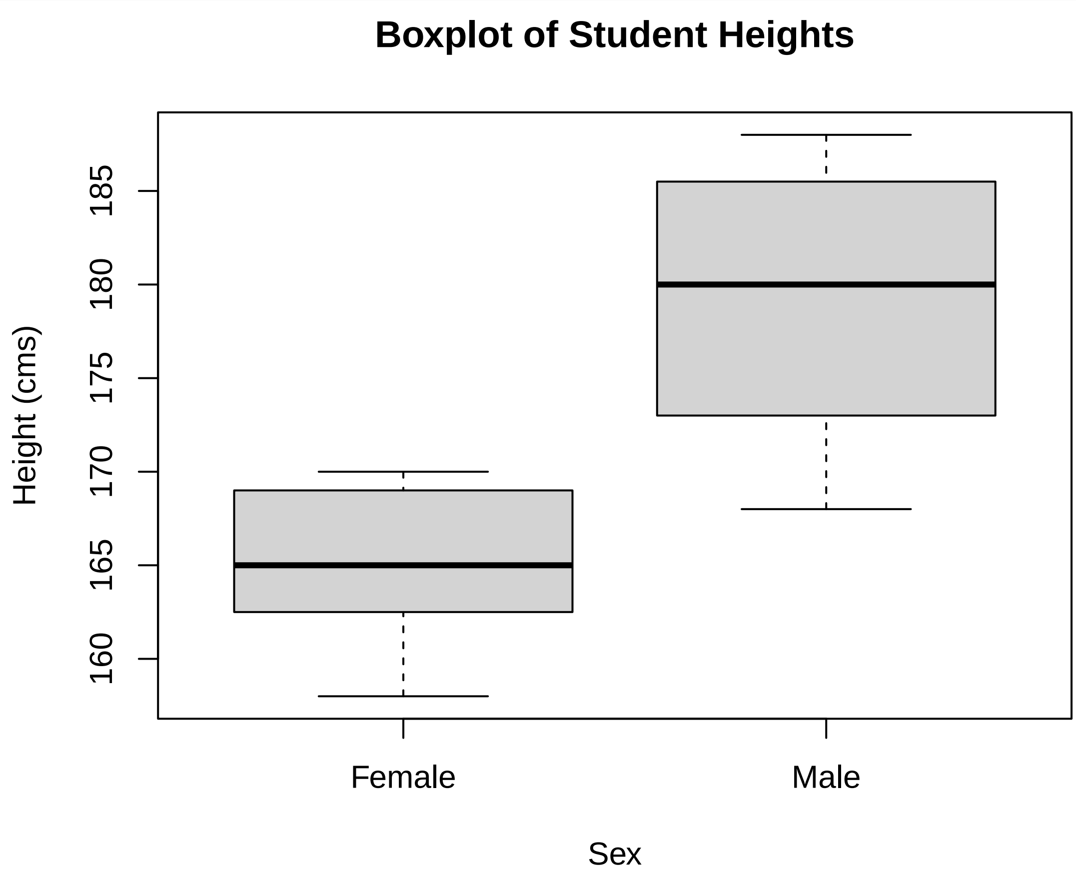
To produce the boxplot in base R

```
boxplot(qd$Height, ylab = "Height (cms)", main = "Boxplot of Student Heights")
```



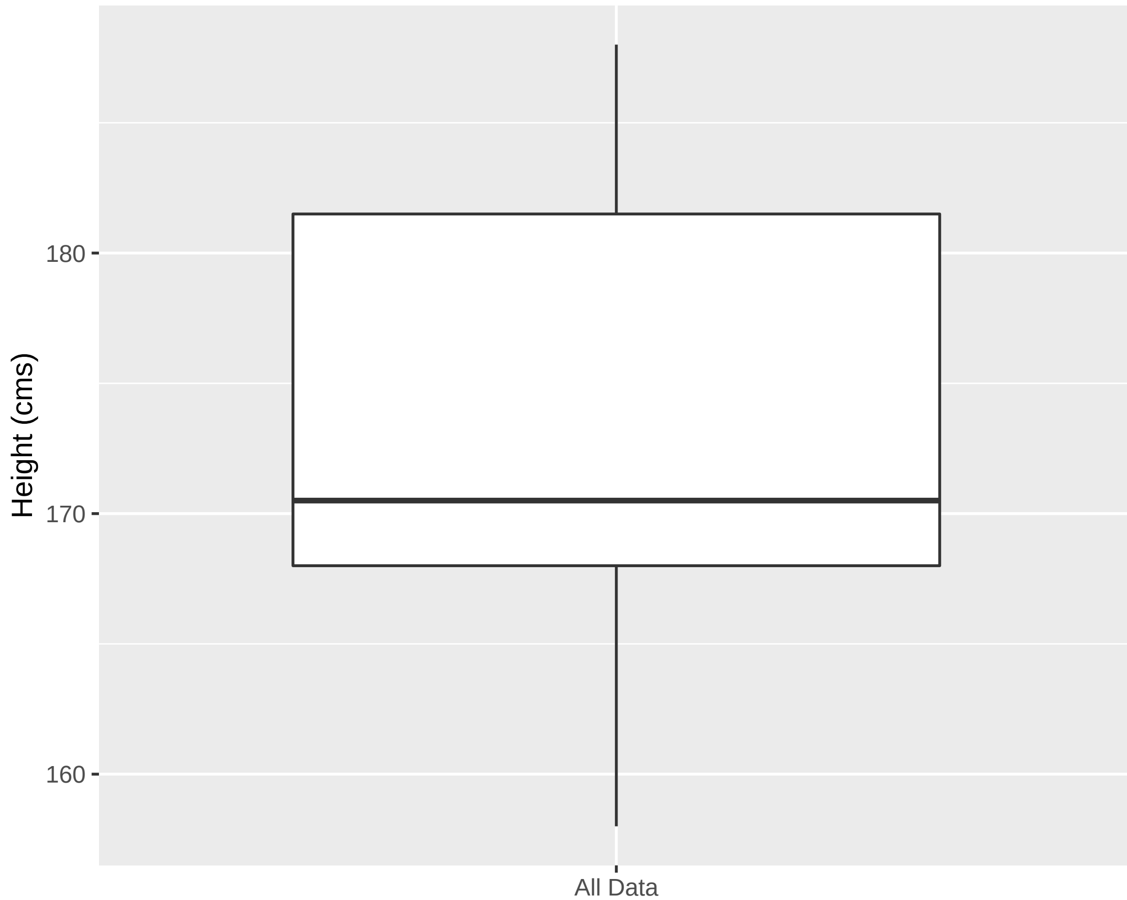
We can produce separate boxplots for males and females:

```
boxplot(Height ~ Sex, data = qd, ylab = "Height (cms)", main = "Boxplot of Student Heights")
```



In `ggplot2`:

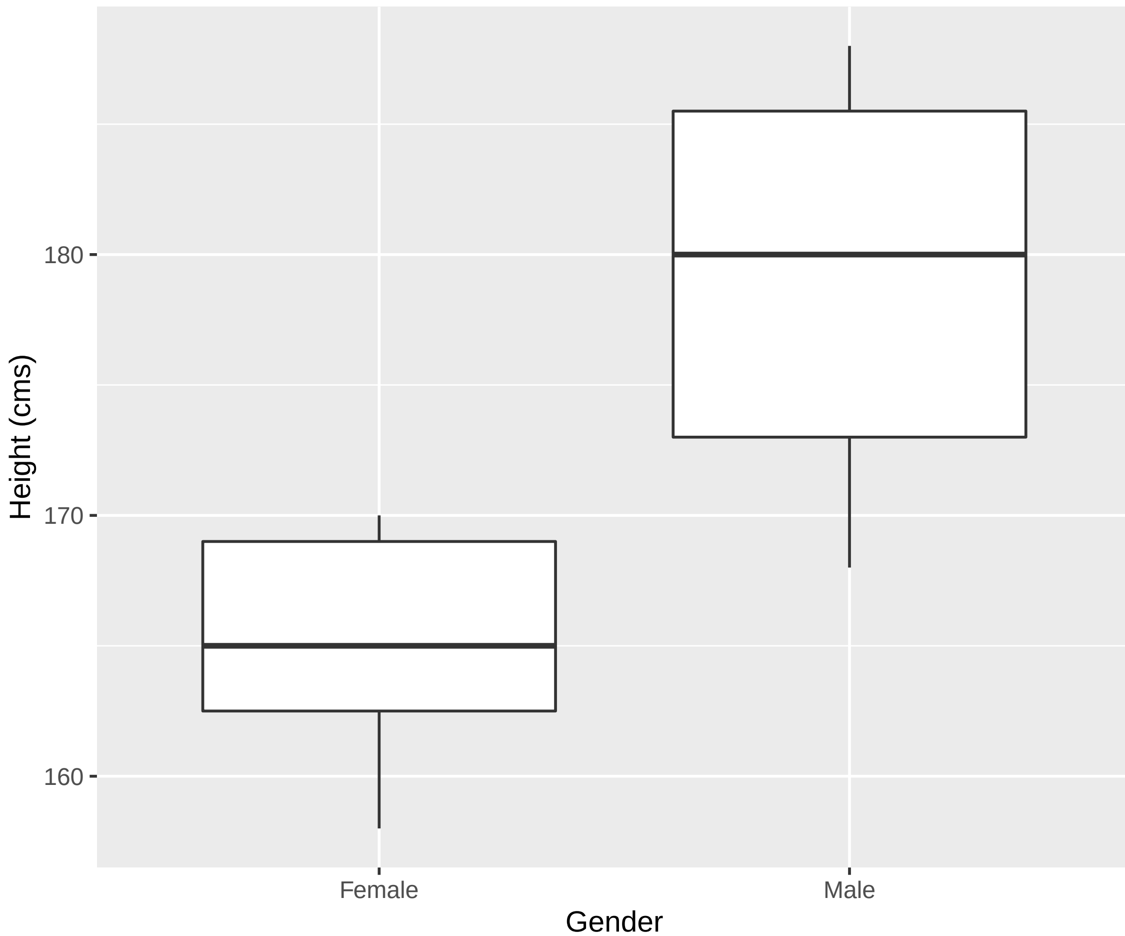
```
ggplot(qd, aes(x = factor(1, label = "All Data"), y = Height)) +
    geom_boxplot() +
    labs(x = "", y = "Height (cms)")
```



Note that we need to specify something (here, something essentially meaningless) to be plotted on the x-axis.

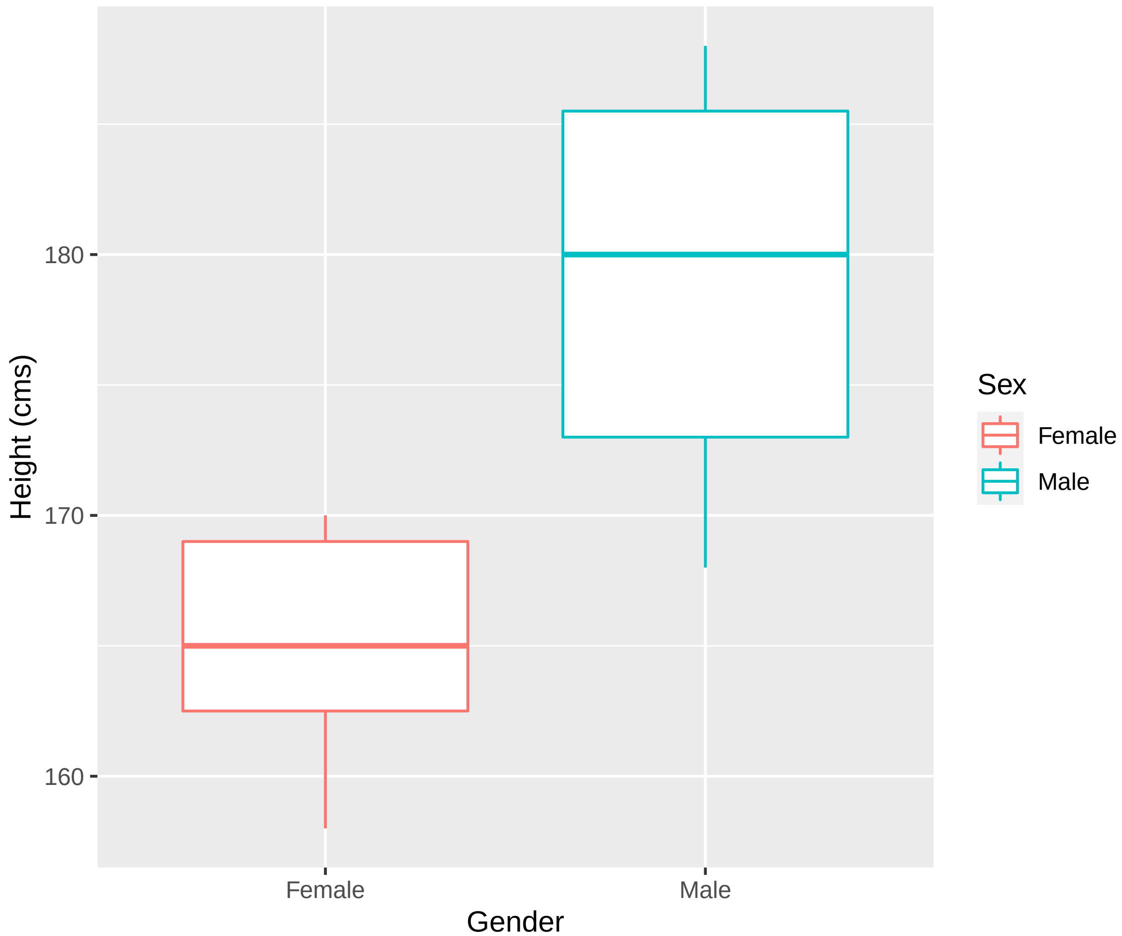
- Separate plots for males and females:

```
ggplot(qd, aes(x = Sex, y = Height)) +
    geom_boxplot() +
    labs(x = "Gender", y = "Height (cms)")
```



or, with colour,

```
ggplot(qd, aes(x = Sex, y = Height, col = Sex)) +
    geom_boxplot() +
    labs(x = "Gender", y = "Height (cms)")
```



9.4 Converting a Continuous Variable into a Factor

Sometimes it is useful to convert a continuous variable (that is, a variable that can take any value in a range) such as `Age` into a factor. The function `cut` helps us to do this; please read its help file. Please remember that an interval $[a, b)$ is said to be closed on the left and open on the right; it contains a, but not b. The interval $(a, b]$ is open on the left and closed on the right; it contains b, but not a.

```
range(qd$Age) # Data range
```

```
## [1] 12.0 35.5
```

```
#
# Split age into age groups
#
cut(qd$Age, breaks = c(12, 20, 22, 24, 30, 40), right = FALSE)
```

```
##  [1] [22,24) [22,24) [30,40) [20,22) [22,24) [24,30) [22,24) [24,30) [12,20)
## [10] [22,24) [24,30) [22,24) [24,30) [24,30) [30,40) [20,22) [24,30) [22,24)
## Levels: [12,20) [20,22) [22,24) [24,30) [30,40)
```

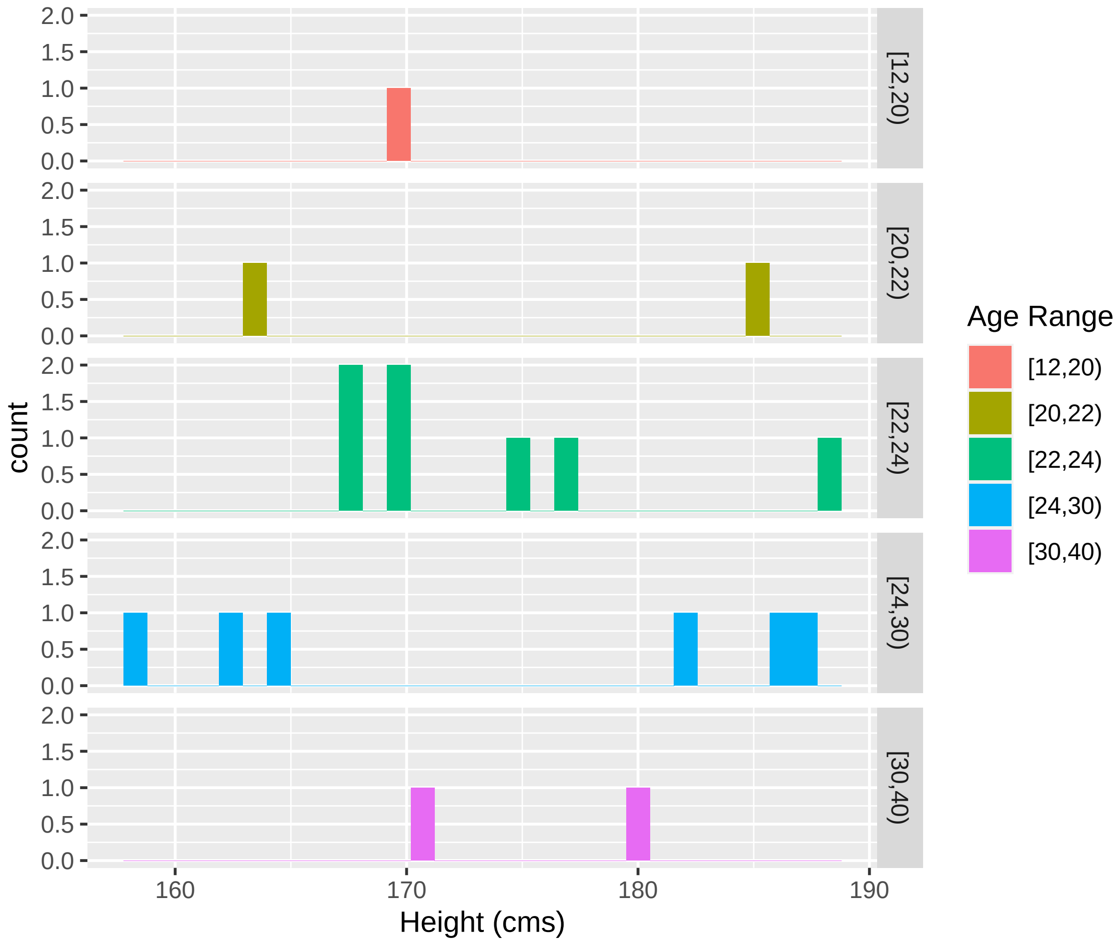
```
#
# Add a variable with this information
#
qd <- qd %>% mutate(Age_f =
                     cut(Age,
                         breaks = c(12, 20, 22, 24, 30, 40),
```

```
                          right = FALSE))
table(qd$Age_f)
```

```
##
## [12,20) [20,22) [22,24) [24,30) [30,40)
##       1       2       7       6       2
```

We can now produce a histogram of `Height` stratified by `Age_f`, the factor version of `Age`:

```
ggplot(qd, aes(x = Height, fill = Age_f)) +
             # fill is used to colour the histogram according to Age_f
    geom_histogram() +
    labs(x = "Height (cms)", fill = "Age Range") +
    facet_grid(Age_f ~ .)
```



Please note the use of the aesthetic `fill` to colour the histogram according to `Age_f`.

The age ranges may be too narrow and may need modification.

10 Statistical Tests

10.1 Comparing the Underlying Means of Two Groups

We can ask:

- Is there an underlying difference in mean height between females and males?

This is a profound question. We are not asking whether there is a difference between the mean height of females and the mean height of males in the data collected. We are asking, more generally, whether there is a difference between the mean height of females and the mean height of males in a much bigger population.

To answer this question, we can use a *t*-test:

```
t.test(Height ~ Sex, data = qd, var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Height by Sex
## t = -4.5075, df = 16, p-value = 0.0003579
## alternative hypothesis: true difference in means between group Female and group Male is not equal to
## 95 percent confidence interval:
##  -20.374200  -7.340085
## sample estimates:
## mean in group Female   mean in group Male
##             165.1429             179.0000
```

We need to focus on the *p*-value

```
## [1] 0.0003579246
```

If the *p*-value is less that 0.05, which it is here, we conclude that

- **there is an underlying difference in mean height between females and males**.

Note that in the above `t.test` code we told R where to look for the variables `Height` and `Sex` using the `data` argument. More precisely, we specified that it should look in the data frame `qd` for the required variables.

In the above `t.test` code we specified `var.equal = TRUE`. This was because we saw from the graphs that the spreads of the female heights and the male heights were quite similar. (We could also test for this.)

If the spreads of the two groups seem very different, we can do a slightly different test called the Welch test, specifying `var.equal = FALSE`:

```
t.test(Height ~ Sex, data = qd, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  Height by Sex
## t = -5.007, df = 16, p-value = 0.000129
## alternative hypothesis: true difference in means between group Female and group Male is not equal to
## 95 percent confidence interval:
##  -19.724102  -7.990184
## sample estimates:
## mean in group Female   mean in group Male
##             165.1429             179.0000
```

Here the conclusion is the same.

10.2 Comparing the Underlying Means of More Than Two Groups

Here are the weekly hours worked by accountants, GPs, lecturers and plumbers[1]

```
accountants <- c(45, 38, 40, 42, 48, 37, 44, 40, 39, 42, 41, 40, 36, 40, 48)
gps <- c(60, 57, 44, 52, 57, 45, 42, 56, 53, 42, 44, 54, 51, 58)
```

[1] Data kindly suppied by Dr John Eales, who has also provided very helpful comments on these notes

```
lecturers <- c(52, 45, 40, 48, 36, 50, 56, 42, 37, 43, 47)
plumbers <- c(44, 39, 50, 37, 45, 39, 52, 45, 39, 48, 44, 43, 53)
```

The question of interest is:

Is the underlying mean number of hours different between these **four** groups?

To answer this question, we first need to transform the data so that there is one column for the hours worked and another for the employment group:

```
n_a <- length(accountants)
n_g <- length(gps)
n_l <- length(lecturers)
n_p <- length(plumbers)
hours_worked <- c(accountants, gps, lecturers, plumbers)
hours_worked
```

```
##  [1] 45 38 40 42 48 37 44 40 39 42 41 40 36 40 48 60 57 44 52 57 45 42 56 53 42
## [26] 44 54 51 58 52 45 40 48 36 50 56 42 37 43 47 44 39 50 37 45 39 52 45 39 48
## [51] 44 43 53
```

```
group <- factor(c(rep("Accountants", n_a),
                  rep("GPS", n_g),
                  rep("Lecturers", n_l),
                  rep("Plumbers", n_p)))
group
```

```
##  [1] Accountants Accountants Accountants Accountants Accountants Accountants
##  [7] Accountants Accountants Accountants Accountants Accountants Accountants
## [13] Accountants Accountants Accountants GPS         GPS         GPS
## [19] GPS         GPS         GPS         GPS         GPS         GPS
## [25] GPS         GPS         GPS         GPS         GPS         Lecturers
## [31] Lecturers   Lecturers   Lecturers   Lecturers   Lecturers   Lecturers
## [37] Lecturers   Lecturers   Lecturers   Lecturers   Plumbers    Plumbers
## [43] Plumbers    Plumbers    Plumbers    Plumbers    Plumbers    Plumbers
## [49] Plumbers    Plumbers    Plumbers    Plumbers    Plumbers
## Levels: Accountants GPS Lecturers Plumbers
```

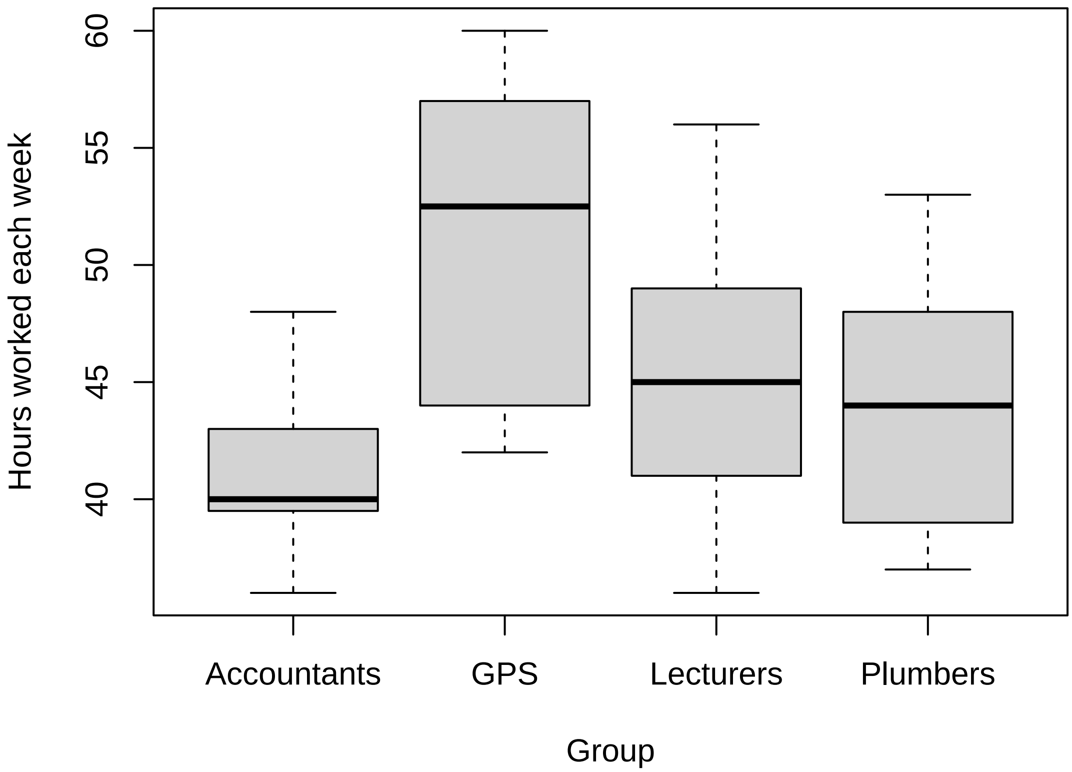
- Factors

Note that group is a **factor**; this means that it comprises labels (Accountant, GP, Lecturer and Plumber) that identify the group to which each person belongs.

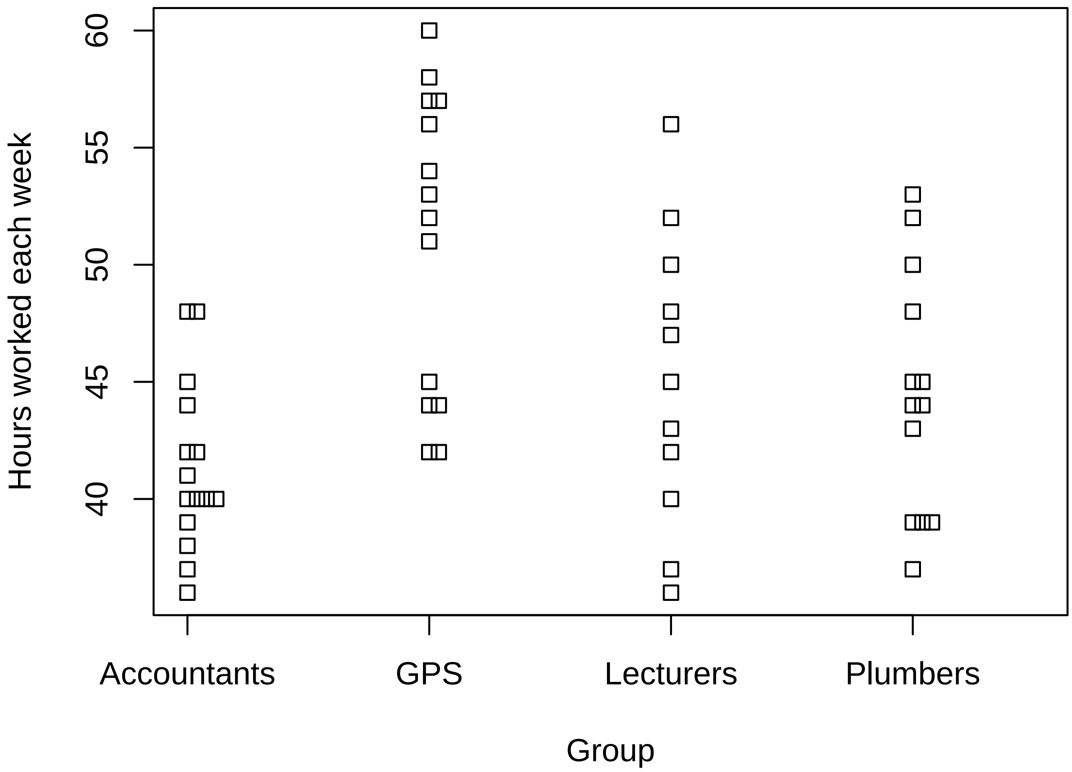
- Numerical and Graphical Summaries

Let's see a simple plot and some summary statistics:

```
boxplot(hours_worked ~ group,
        xlab = "Group", ylab = "Hours worked each week")
```




```r
stripchart(hours_worked ~ group,
           vertical = TRUE, method = "stack",
           xlab = "Group", ylab = "Hours worked each week")
```



```r
mean(hours_worked) # Overall mean
```

```
## [1] 45.45283
```

```r
by(hours_worked, group, mean) # Group means (alternative way to compute them)
```

```
## group: Accountants
## [1] 41.33333
## ------------------------------------------------------------
## group: GPS
## [1] 51.07143
## ------------------------------------------------------------
## group: Lecturers
## [1] 45.09091
## ------------------------------------------------------------
## group: Plumbers
## [1] 44.46154
```

```r
by(hours_worked, group, sd) # Group standard deviations (alternative way to compute them)
```

```
## group: Accountants
## [1] 3.598942
## ------------------------------------------------------------
## group: GPS
## [1] 6.426679
## ------------------------------------------------------------
## group: Lecturers
## [1] 6.252272
## ------------------------------------------------------------
## group: Plumbers
## [1] 5.173899
```

`Exercise:` Can you produce the boxplots using `ggplot2` and the summary statistics using `dplyr`?

- One-way Analysis of Variance

To answer the question

Is the underlying mean number of hours different between these **four** groups?

we perform a **one-way analysis of variance**. In fact, we are fitting a linear model, as met above. This can be done using the `lm` function.

```r
m <- lm(hours_worked ~ group) # Fit a linear model
anova(m)
```

```
## Analysis of Variance Table
##
## Response: hours_worked
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## group      3  710.73 236.910  8.1156 0.0001731 ***
## Residuals 49 1430.40  29.192
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, the p-value is

```
## [1] 0.0001731113
```

If the p-value is less that 0.05, which it is here, we conclude that

- **there is an difference between the underlying mean number of hours different between these four groups**.

A similar result can be obtained using the `aov` function

```
m2 <- aov(hours_worked ~ group)
summary(m2)
```

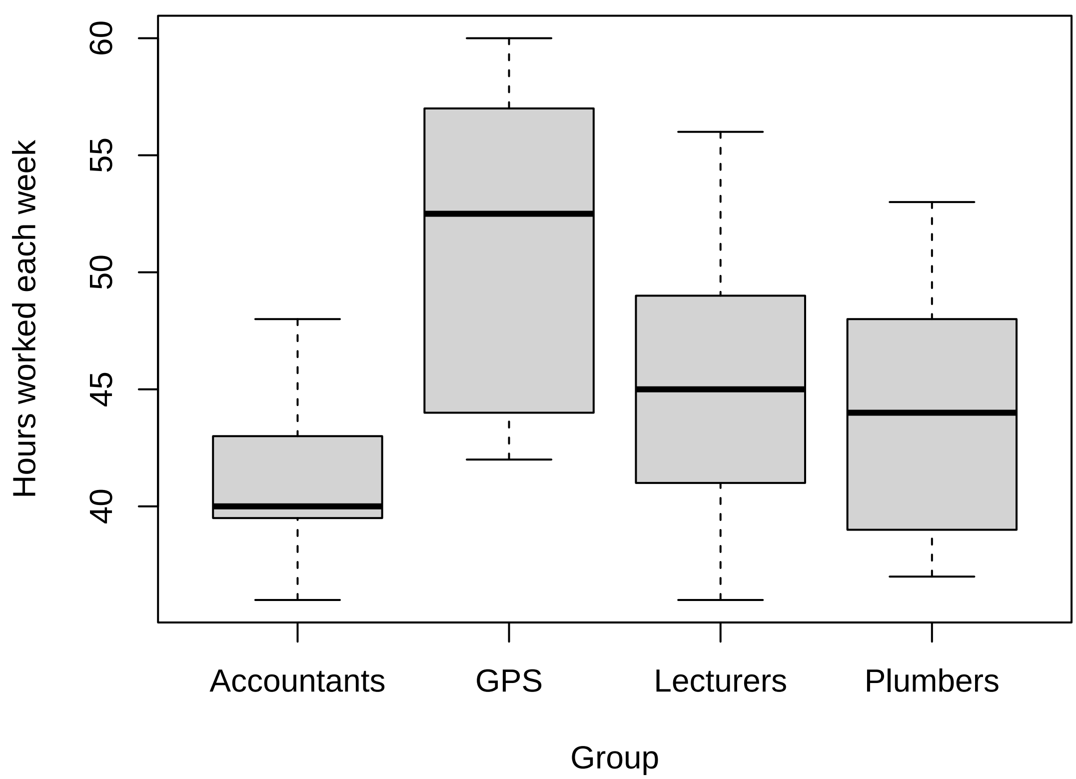
```
##             Df Sum Sq Mean Sq F value   Pr(>F)    
## group        3  710.7  236.91   8.116 0.000173 ***
## Residuals   49 1430.4   29.19                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that `boxplot`, `lm` and `aov` have `data` arguments just like `t.test`. This allows us to tell R to look into a data frame for the required variables. Here are some examples,

```
df_working_hours <- data.frame(G = group, H = hours_worked)
head(df_working_hours)
```

```
##             G  H
## 1 Accountants 45
## 2 Accountants 38
## 3 Accountants 40
## 4 Accountants 42
## 5 Accountants 48
## 6 Accountants 37
```

```
boxplot(H ~ G, data = df_working_hours, xlab = "Group", ylab = "Hours worked each week")
```




```
m3 <- lm(H ~ G, data = df_working_hours)
anova(m3)
```

```
## Analysis of Variance Table
##
## Response: H
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## G          3  710.73 236.910  8.1156 0.0001731 ***
## Residuals 49 1430.40  29.192
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
m4 <- aov(H ~ G, data = df_working_hours)
summary(m4)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## G            3  710.7  236.91   8.116 0.000173 ***
## Residuals   49 1430.4   29.19
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- Follow-up Analysis

If we find that there is an difference between the underlying means of the groups, we should proceed by performing a follow-up analysis to see where the group differences are:

```
TukeyHSD(m2) # Follow-up: pair-wise comparisons
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = hours_worked ~ group)
##
## $group
##                             diff        lwr        upr     p adj
## GPS-Accountants        9.7380952   4.398473 15.0777170 0.0000745
## Lecturers-Accountants  3.7575758  -1.946245  9.4613962 0.3086324
## Plumbers-Accountants   3.1282051  -2.316606  8.5730167 0.4289920
## Lecturers-GPS         -5.9805195 -11.769883 -0.1911564 0.0404480
## Plumbers-GPS          -6.6098901 -12.144249 -1.0755310 0.0133213
## Plumbers-Lecturers    -0.6293706  -6.515892  5.2571510 0.9918817
```

We should look at the p-values in the `p adj` column. When the p-value is less that 0.05, we should conclude that there is an underlying differnce.
So here there are differences between GPs and accountants, between lecturers and GPs and between plumbers and GPs. GPs work a lot!!!

11 Working with Likert Scale Data

11.1 What is Likert Scale Data

For illustration purposes, we will work with a small data set from a simple questionnaire to gain some experience with dealing with data that are recorded on a Likert scale (Strongly Disagree, Disagree, Neutral, Agree and Strongly Agree).

The data are in a comma separated variable file `likert_example.csv` in the usual working directory. Let's read in that file:

```
require(readr)
setwd("~/Documents/backup_22_11_2021/Plym_teaching/MATH513/2019_20/Introduction_to_R")
# Your working directory will probably be different
```

```r
simple_questionnaire <- read_csv("likert_example.csv")
simple_questionnaire
```

```
## # A tibble: 10 x 5
##    Person Q1                Q2                Q3             Gender
##    <chr>  <chr>             <chr>             <chr>          <chr>
##  1 P1     Neutral           Strongly Agree    Neutral        M
##  2 P2     Strongly Disagree Strongly Agree    Neutral        M
##  3 P3     Disagree          Strongly Agree    Neutral        F
##  4 P4     Neutral           Neutral           Agree          F
##  5 P5     Agree             Strongly Agree    Disagree       F
##  6 P6     Strongly Agree    Strongly Agree    Neutral        M
##  7 P7     Strongly Agree    Strongly Agree    Disagree       M
##  8 P8     Disagree          Strongly Disagree Neutral        M
##  9 P9     Disagree          Agree             Agree          M
## 10 P10    Agree             Disagree          Strongly Agree F
```

```r
str(simple_questionnaire)
```

```
## spec_tbl_df [10 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Person: chr [1:10] "P1" "P2" "P3" "P4" ...
##  $ Q1    : chr [1:10] "Neutral" "Strongly Disagree" "Disagree" "Neutral" ...
##  $ Q2    : chr [1:10] "Strongly Agree" "Strongly Agree" "Strongly Agree" "Neutral" ...
##  $ Q3    : chr [1:10] "Neutral" "Neutral" "Neutral" "Agree" ...
##  $ Gender: chr [1:10] "M" "M" "F" "F" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Person = col_character(),
##   ..   Q1 = col_character(),
##   ..   Q2 = col_character(),
##   ..   Q3 = col_character(),
##   ..   Gender = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

The data record the responses to three questions (Question 1/2/3 or Q1/2/3) of ten people. The gender of each person is also recorded.

11.2 Expressing the Question Results and Other Variables as Factors

As the variables/columns `Q1`, `Q2` and `Q3` are characters, it makes sense to redefine them as factors, with the five levels Strongly Disagree, Disagree, Neutral, Agree and Strongly Agree. We will also define Gender as a factor.

We will try to do this in a way that reminds us of the structure of an R function and the use of `mutate_each` from `dplyr`.

The following function converts a vector to a factor with the required five levels:

```r
factor_5 <- function(x){
    factor(x, levels = c("Strongly Disagree", "Disagree", "Neutral", "Agree", "Strongly Agree"))
}
factor_5
```

```
## function(x){
##     factor(x, levels = c("Strongly Disagree", "Disagree", "Neutral", "Agree", "Strongly Agree"))
## }
```

Our function is called `factor_5`, a name that we choose. Its argument, in round brackets `()`, is a vector `x`. This vector is then transformed into a factor with the required five levels.

Let's use this function to transform the variables/columns `Q1`, `Q2` and `Q3` simultaneously into factors:

```
require(dplyr)
simple_questionnaire_f <- simple_questionnaire %>% mutate_each(funs(factor_5), Q1:Q3)
# This applies the function function_5 to the columns Q1 through to Q3
simple_questionnaire_f
```

```
## # A tibble: 10 x 5
##    Person Q1                Q2                Q3             Gender
##    <chr>  <fct>             <fct>             <fct>          <chr>
##  1 P1     Neutral           Strongly Agree    Neutral        M
##  2 P2     Strongly Disagree Strongly Agree    Neutral        M
##  3 P3     Disagree          Strongly Agree    Neutral        F
##  4 P4     Neutral           Neutral           Agree          F
##  5 P5     Agree             Strongly Agree    Disagree       F
##  6 P6     Strongly Agree    Strongly Agree    Neutral        M
##  7 P7     Strongly Agree    Strongly Agree    Disagree       M
##  8 P8     Disagree          Strongly Disagree Neutral        M
##  9 P9     Disagree          Agree             Agree          M
## 10 P10    Agree             Disagree          Strongly Agree F
```

```
str(simple_questionnaire_f)
```

```
## spec_tbl_df [10 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Person: chr [1:10] "P1" "P2" "P3" "P4" ...
##  $ Q1    : Factor w/ 5 levels "Strongly Disagree",..: 3 1 2 3 4 5 5 2 2 4
##  $ Q2    : Factor w/ 5 levels "Strongly Disagree",..: 5 5 5 3 5 5 5 1 4 2
##  $ Q3    : Factor w/ 5 levels "Strongly Disagree",..: 3 3 3 4 2 3 2 3 4 5
##  $ Gender: chr [1:10] "M" "M" "F" "F" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Person = col_character(),
##   ..   Q1 = col_character(),
##   ..   Q2 = col_character(),
##   ..   Q3 = col_character(),
##   ..   Gender = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Please note that more generally we can refer to columns by their number:

```
simple_questionnaire %>% mutate_each(funs(factor_5), 2:4)
```

```
## # A tibble: 10 x 5
##    Person Q1                Q2                Q3             Gender
##    <chr>  <fct>             <fct>             <fct>          <chr>
##  1 P1     Neutral           Strongly Agree    Neutral        M
##  2 P2     Strongly Disagree Strongly Agree    Neutral        M
##  3 P3     Disagree          Strongly Agree    Neutral        F
##  4 P4     Neutral           Neutral           Agree          F
##  5 P5     Agree             Strongly Agree    Disagree       F
##  6 P6     Strongly Agree    Strongly Agree    Neutral        M
##  7 P7     Strongly Agree    Strongly Agree    Disagree       M
##  8 P8     Disagree          Strongly Disagree Neutral        M
```

```
##  9 P9     Disagree          Agree             Agree          M
## 10 P10    Agree             Disagree          Strongly Agree F
```

Compare this with:

```
simple_questionnaire %>% mutate_each(funs(factor_5), c(2,4))
```

```
## # A tibble: 10 x 5
##    Person Q1                Q2                Q3             Gender
##    <chr>  <fct>             <chr>             <fct>          <chr>
##  1 P1     Neutral           Strongly Agree    Neutral        M
##  2 P2     Strongly Disagree Strongly Agree    Neutral        M
##  3 P3     Disagree          Strongly Agree    Neutral        F
##  4 P4     Neutral           Neutral           Agree          F
##  5 P5     Agree             Strongly Agree    Disagree       F
##  6 P6     Strongly Agree    Strongly Agree    Neutral        M
##  7 P7     Strongly Agree    Strongly Agree    Disagree       M
##  8 P8     Disagree          Strongly Disagree Neutral        M
##  9 P9     Disagree          Agree             Agree          M
## 10 P10    Agree             Disagree          Strongly Agree F
```

Now for `Gender`:

```
simple_questionnaire_f2 <- simple_questionnaire_f %>%
    mutate(Gender = factor(Gender, level = c("F", "M")))
# F and M would be the default order
simple_questionnaire_f2
```

```
## # A tibble: 10 x 5
##    Person Q1                Q2                Q3             Gender
##    <chr>  <fct>             <fct>             <fct>          <fct>
##  1 P1     Neutral           Strongly Agree    Neutral        M
##  2 P2     Strongly Disagree Strongly Agree    Neutral        M
##  3 P3     Disagree          Strongly Agree    Neutral        F
##  4 P4     Neutral           Neutral           Agree          F
##  5 P5     Agree             Strongly Agree    Disagree       F
##  6 P6     Strongly Agree    Strongly Agree    Neutral        M
##  7 P7     Strongly Agree    Strongly Agree    Disagree       M
##  8 P8     Disagree          Strongly Disagree Neutral        M
##  9 P9     Disagree          Agree             Agree          M
## 10 P10    Agree             Disagree          Strongly Agree F
```

```
str(simple_questionnaire_f2)
```

```
## spec_tbl_df [10 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Person: chr [1:10] "P1" "P2" "P3" "P4" ...
##  $ Q1    : Factor w/ 5 levels "Strongly Disagree",..: 3 1 2 3 4 5 5 2 2 4
##  $ Q2    : Factor w/ 5 levels "Strongly Disagree",..: 5 5 5 3 5 5 5 1 4 2
##  $ Q3    : Factor w/ 5 levels "Strongly Disagree",..: 3 3 3 4 2 3 2 3 4 5
##  $ Gender: Factor w/ 2 levels "F","M": 2 2 1 1 1 2 2 2 2 1
##  - attr(*, "spec")=
##   .. cols(
##   ..   Person = col_character(),
##   ..   Q1 = col_character(),
##   ..   Q2 = col_character(),
##   ..   Q3 = col_character(),
##   ..   Gender = col_character()
```

```
##   .. )
##  - attr(*, "problems")=<externalptr>
```

We could transform the questionnaire responses into numerical values for use with data dimension reduction and cluster identification procedures. It should, however, be pointed out that the choice of the numbers 1, 2, 3, 4 and 5 to represent "Strongly Disagree", "Disagree", "Neutral", "Agree" and "Strongly Agree" is an arbitrary one. Other people may choose other numbers:

```
simple_questionnaire_numeric <- simple_questionnaire_f2  %>%
    mutate_each(funs(as.numeric), Q1:Q3) # The function as.numeric does the work
simple_questionnaire_numeric
```

```
## # A tibble: 10 x 5
##    Person    Q1    Q2    Q3 Gender
##    <chr>  <dbl> <dbl> <dbl> <fct>
##  1 P1         3     5     3 M
##  2 P2         1     5     3 M
##  3 P3         2     5     3 F
##  4 P4         3     3     4 F
##  5 P5         4     5     2 F
##  6 P6         5     5     3 M
##  7 P7         5     5     2 M
##  8 P8         2     1     3 M
##  9 P9         2     4     4 M
## 10 P10        4     2     5 F
```

```
str(simple_questionnaire_numeric)
```

```
## spec_tbl_df [10 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Person: chr [1:10] "P1" "P2" "P3" "P4" ...
##  $ Q1    : num [1:10] 3 1 2 3 4 5 5 2 2 4
##  $ Q2    : num [1:10] 5 5 5 3 5 5 5 1 4 2
##  $ Q3    : num [1:10] 3 3 3 4 2 3 2 3 4 5
##  $ Gender: Factor w/ 2 levels "F","M": 2 2 1 1 1 2 2 2 2 1
##  - attr(*, "spec")=
##   .. cols(
##   ..   Person = col_character(),
##   ..   Q1 = col_character(),
##   ..   Q2 = col_character(),
##   ..   Q3 = col_character(),
##   ..   Gender = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
#
# Compare
#
simple_questionnaire_f2
```

```
## # A tibble: 10 x 5
##    Person Q1                Q2                Q3             Gender
##    <chr>  <fct>             <fct>             <fct>          <fct>
##  1 P1     Neutral           Strongly Agree    Neutral        M
##  2 P2     Strongly Disagree Strongly Agree    Neutral        M
##  3 P3     Disagree          Strongly Agree    Neutral        F
##  4 P4     Neutral           Neutral           Agree          F
##  5 P5     Agree             Strongly Agree    Disagree       F
```

```
##  6 P6     Strongly Agree    Strongly Agree    Neutral        M
##  7 P7     Strongly Agree    Strongly Agree    Disagree       M
##  8 P8     Disagree          Strongly Disagree Neutral        M
##  9 P9     Disagree          Agree             Agree          M
## 10 P10    Agree             Disagree          Strongly Agree F
```

11.3 Simple Tabulation

We can tabulate the `Q1`, `Q2` and `Q3` results as follows:

```
table(simple_questionnaire_f2$Q1)
```

```
##
## Strongly Disagree          Disagree           Neutral             Agree
##                 1                 3                 2                 2
##    Strongly Agree
##                 2
```

```
table(simple_questionnaire_f2$Q2)
```

```
##
## Strongly Disagree          Disagree           Neutral             Agree
##                 1                 1                 1                 1
##    Strongly Agree
##                 6
```

```
table(simple_questionnaire_f2$Q3)
```

```
##
## Strongly Disagree          Disagree           Neutral             Agree
##                 0                 2                 5                 2
##    Strongly Agree
##                 1
```

11.4 Graphical Displays: Customizing a ggplot Barplot

We can produce nice displays of these data quite simply using `ggplot2`.
The first step is to construct a column containing the main variable of interest, which here would be `Question_Response`. This can be done using the function `gather` from `tidyr`:

```
require(tidyr)
simple_questionnaire_2 <- simple_questionnaire_f2 %>%
    gather(Question, Question_Response, Q1:Q3)
simple_questionnaire_2
```

```
## # A tibble: 30 x 4
##    Person Gender Question Question_Response
##    <chr>  <fct>  <chr>    <chr>
##  1 P1     M      Q1       Neutral
##  2 P2     M      Q1       Strongly Disagree
##  3 P3     F      Q1       Disagree
##  4 P4     F      Q1       Neutral
##  5 P5     F      Q1       Agree
##  6 P6     M      Q1       Strongly Agree
##  7 P7     M      Q1       Strongly Agree
##  8 P8     M      Q1       Disagree
##  9 P9     M      Q1       Disagree
```

```
## 10 P10    F      Q1       Agree
## # ... with 20 more rows
```

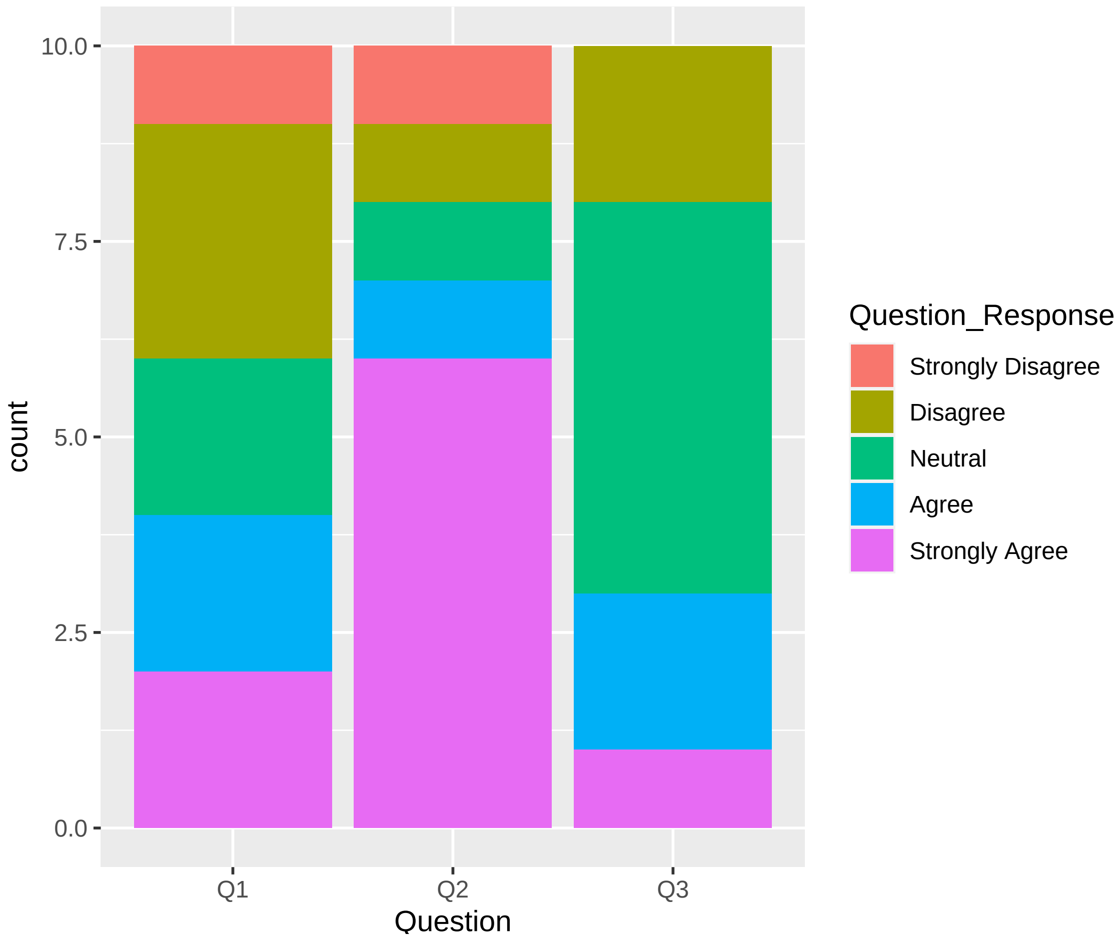
```r
#
# So we create one column that records the Question,
# another that records the Question_Responses, and
# these are filled with the data in columns Q1 through Q3
#
str(simple_questionnaire_2)
```

```
## tibble [30 x 4] (S3: tbl_df/tbl/data.frame)
##  $ Person           : chr [1:30] "P1" "P2" "P3" "P4" ...
##  $ Gender           : Factor w/ 2 levels "F","M": 2 2 1 1 1 2 2 2 2 1 ...
##  $ Question         : chr [1:30] "Q1" "Q1" "Q1" "Q1" ...
##  $ Question_Response: chr [1:30] "Neutral" "Strongly Disagree" "Disagree" "Neutral" ...
```

```r
#
# Define Question_Response as a factor
#
simple_questionnaire_2_f <- simple_questionnaire_2 %>%
    mutate(Question_Response =
               factor(Question_Response,
                      levels = c("Strongly Disagree", "Disagree", "Neutral",
                                 "Agree", "Strongly Agree")))
```

Now we can use the power of ggplot2. Please remember that we are working with a very small, illustrative example.

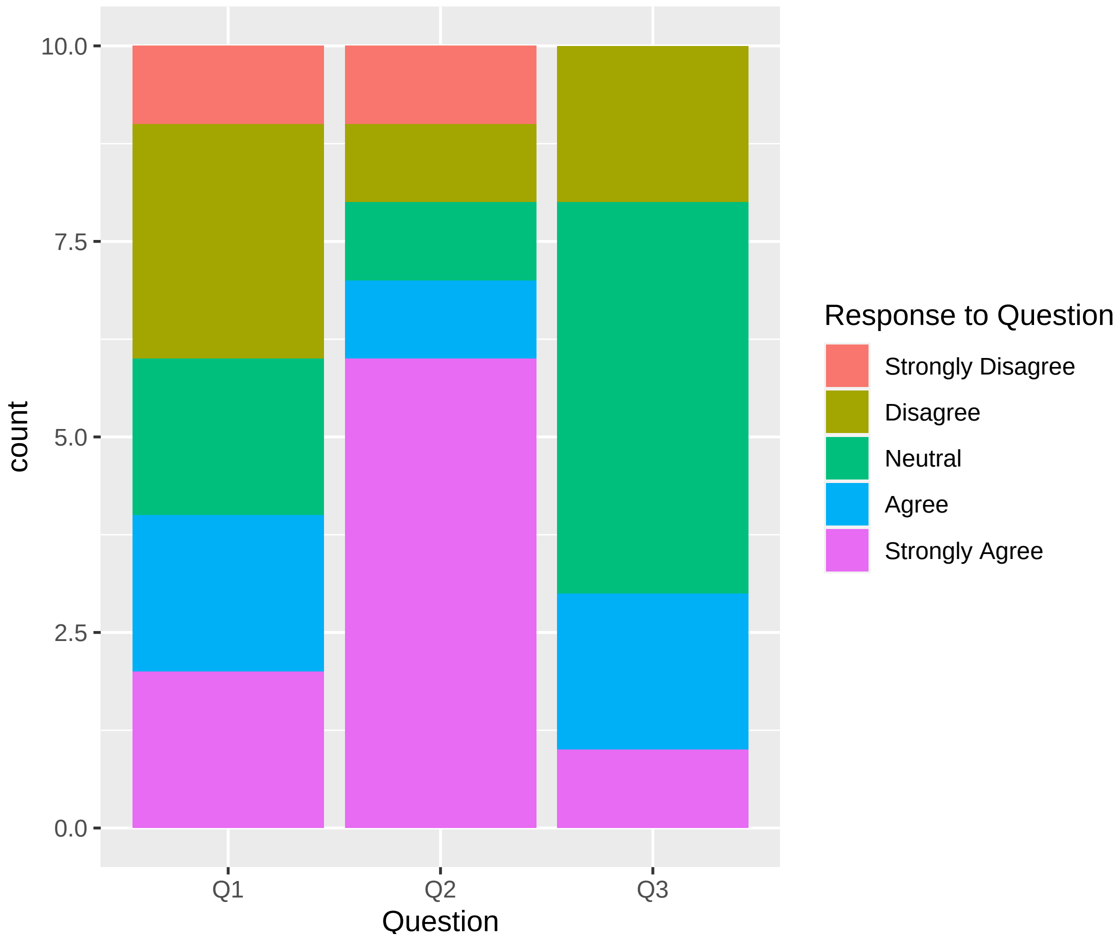
```r
require(ggplot2)
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar()
```



Please note that `geom_bar` performs the tabulation for us. So it's very easy to produce a nice barplot of the data. We may wish to make some changes.

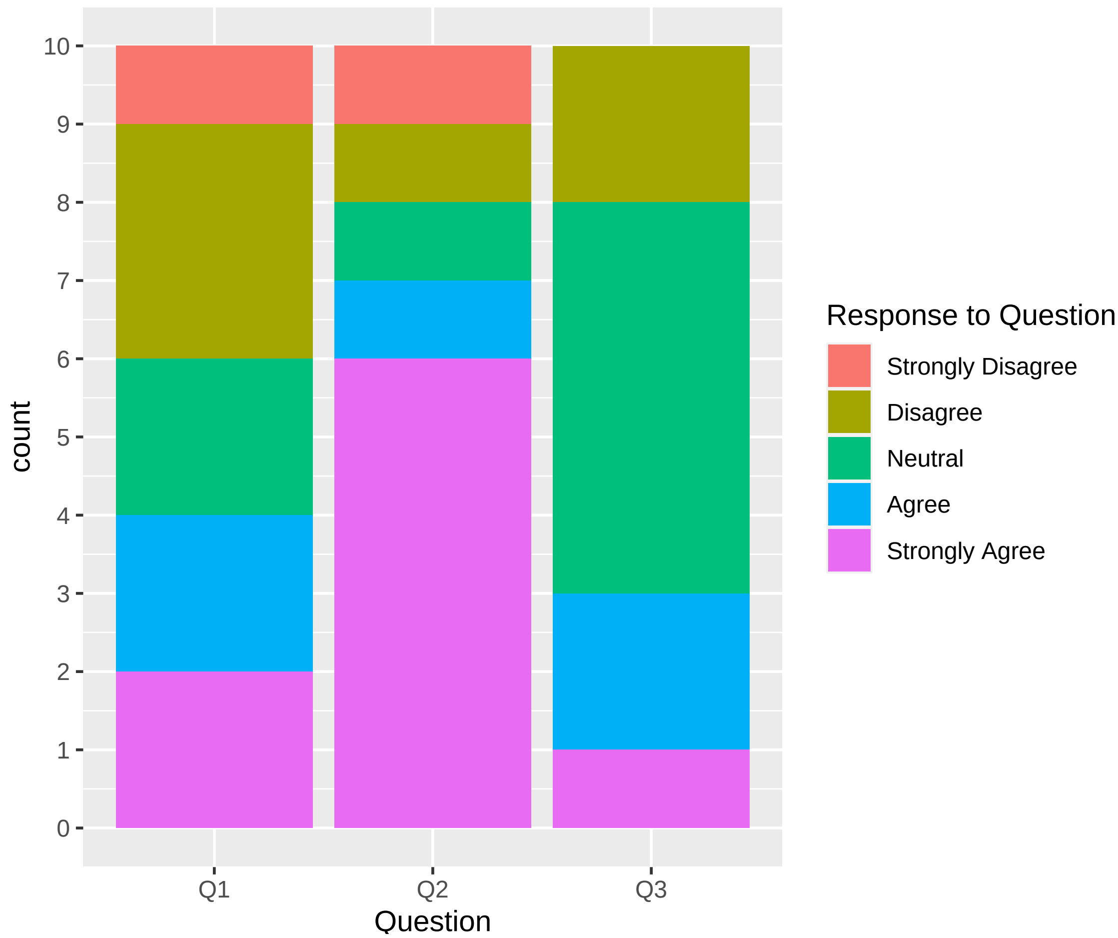
First, let's provide a better legend title:

```
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar() +
    labs(fill = "Response to Question")
```



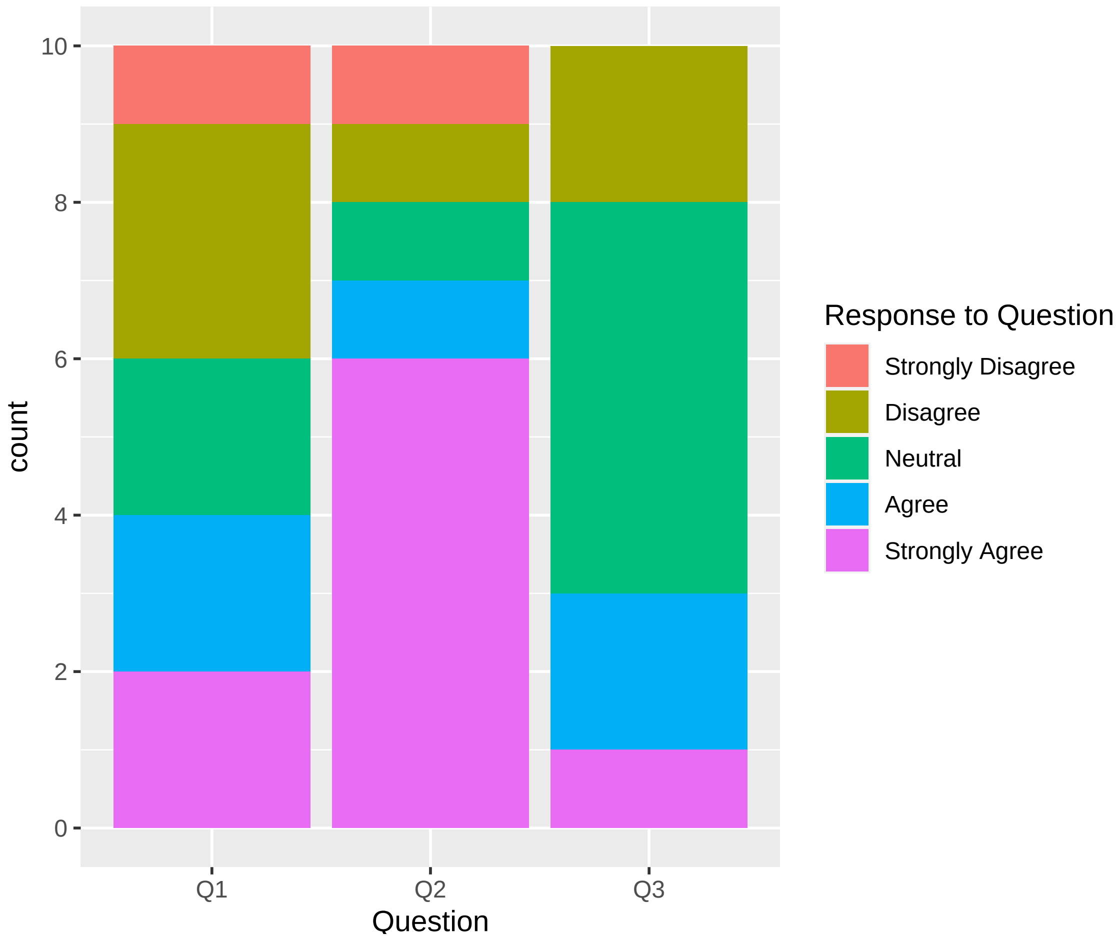
Now let's specify the y-axis scale:

```
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar() +
    labs(fill = "Response to Question") +
    scale_y_continuous(breaks = 0:10)
```



or more generally

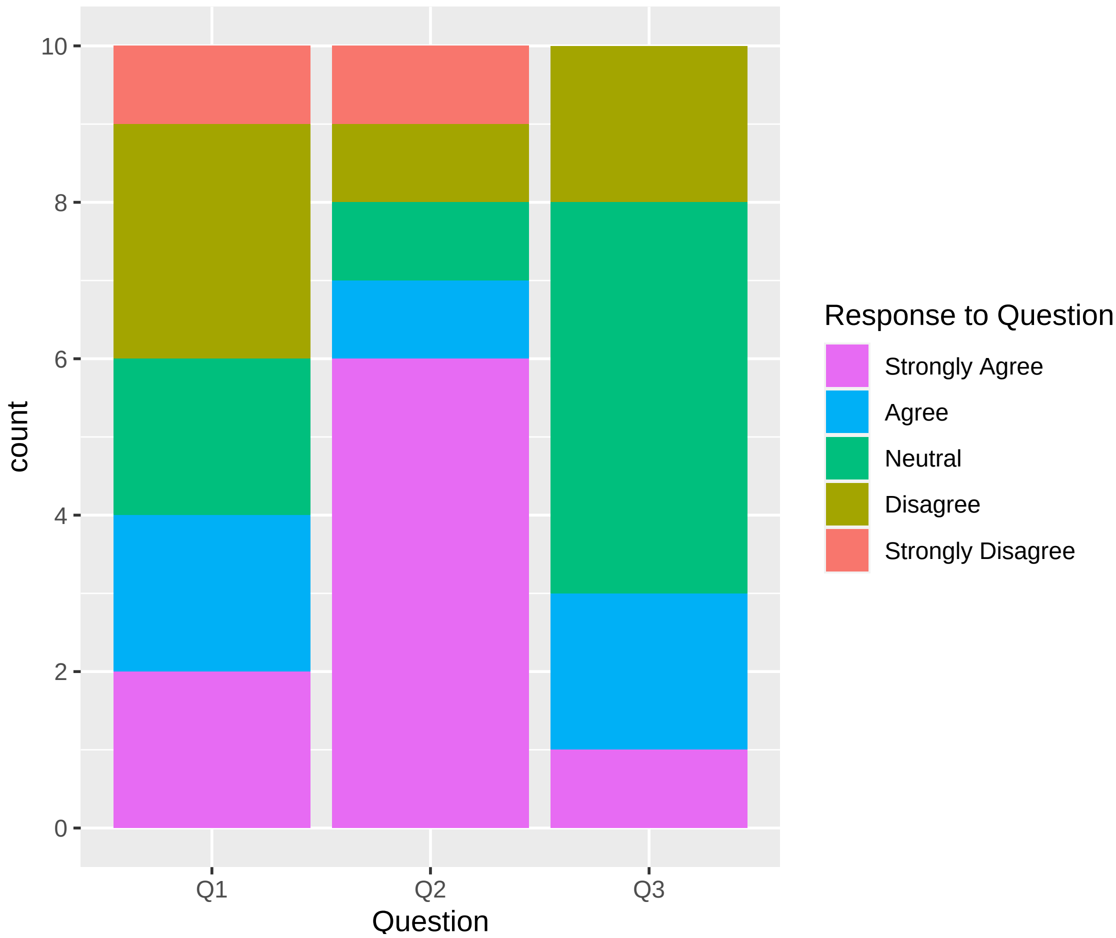
```
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar() +
    labs(fill = "Response to Question") +
    scale_y_continuous(breaks =
        seq(from = 0, to = nrow(simple_questionnaire), by = 2))    # Scale goes up by 2 units
```



for example.

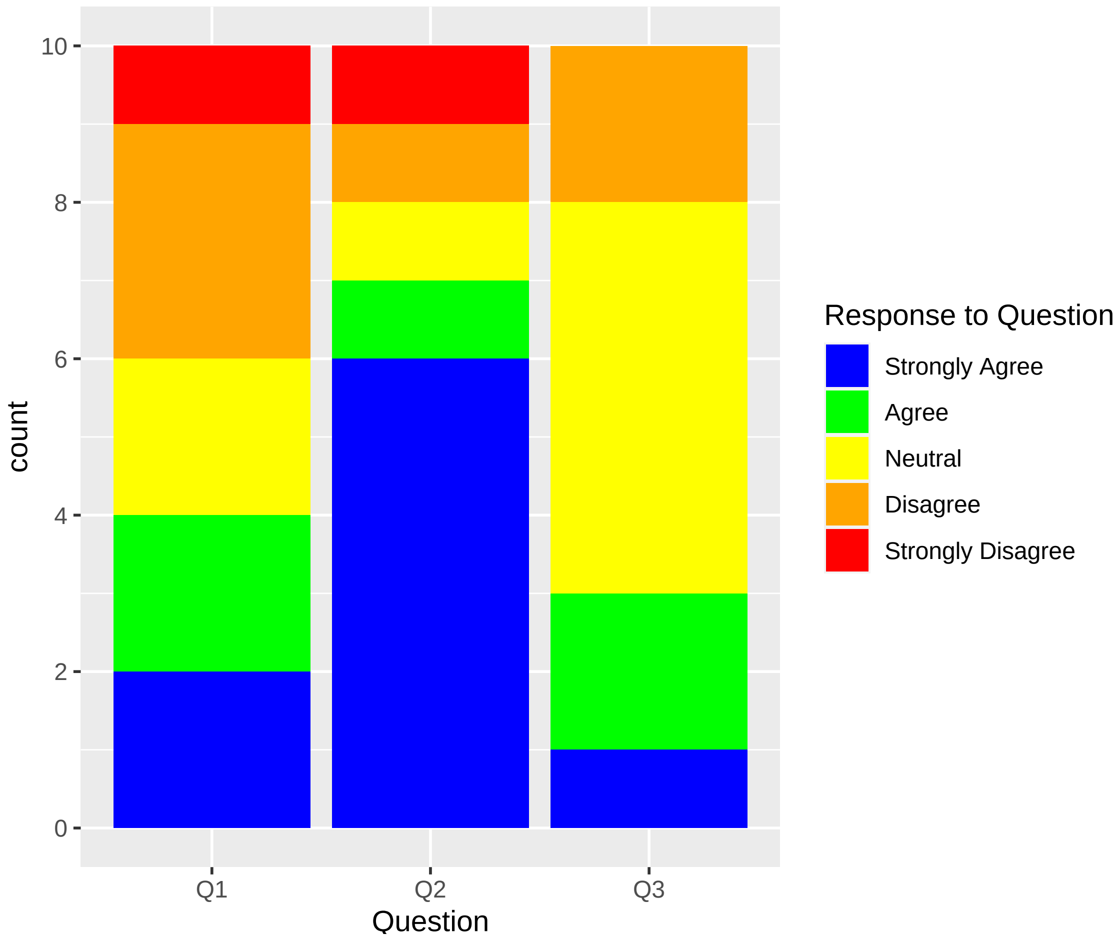
Now let's reverse the order of the legend to correspond better to the plot:

```
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar() +
    labs(fill = "Response to Question") +
    scale_y_continuous(breaks = seq(from = 0, to = nrow(simple_questionnaire), by = 2)) +
    scale_fill_discrete(guide = guide_legend(reverse=TRUE))
```



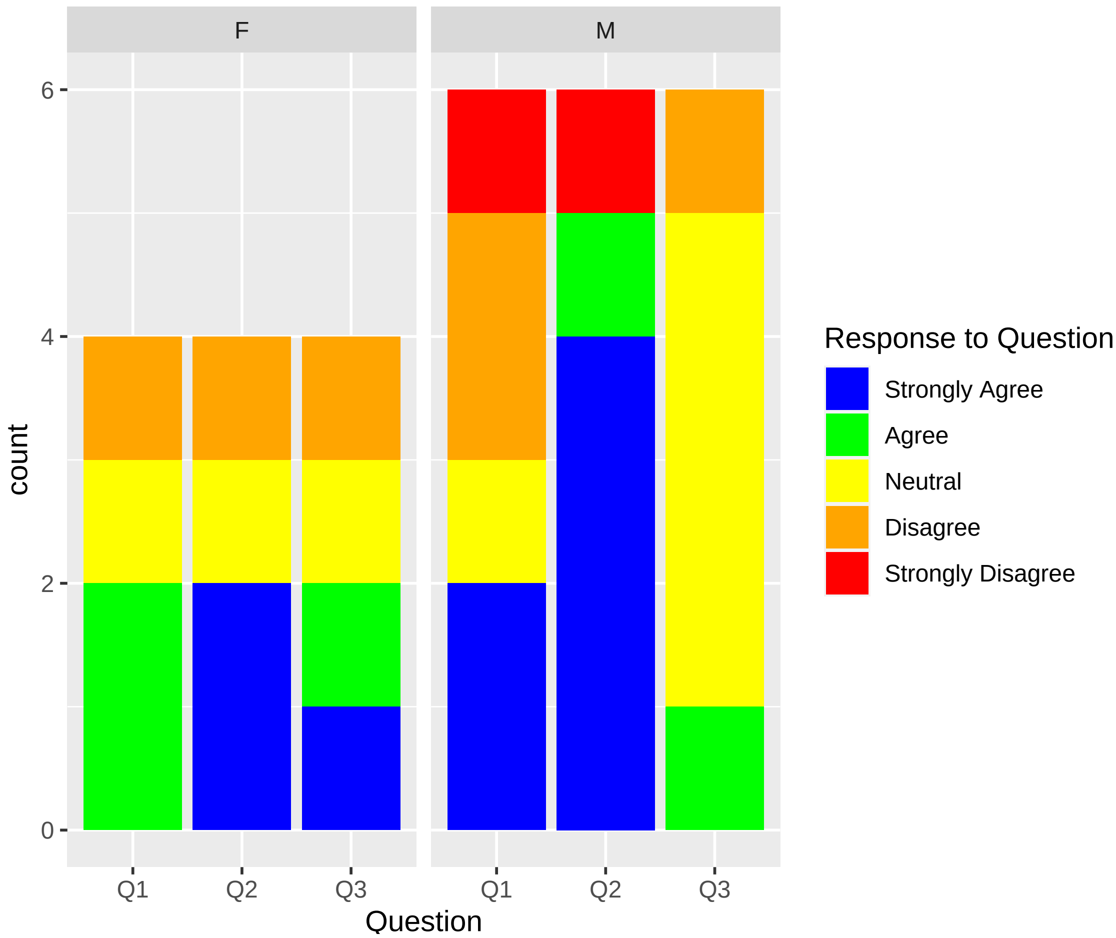
Finally, let's choose our own colours.
Please note that when we specify values as well we need to use `scale_fill_manual` and not `scale_fill_discrete`:

```
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar() +
    labs(fill = "Response to Question") +
    scale_y_continuous(breaks = seq(from = 0, to = nrow(simple_questionnaire), by = 2)) +
    scale_fill_manual(guide = guide_legend(reverse=TRUE),
                        values =  c("Strongly Disagree" = "red",
                            "Disagree" = "orange",
                            "Neutral" = "yellow",
                            "Agree" = "green",
                            "Strongly Agree" = "blue"))
```



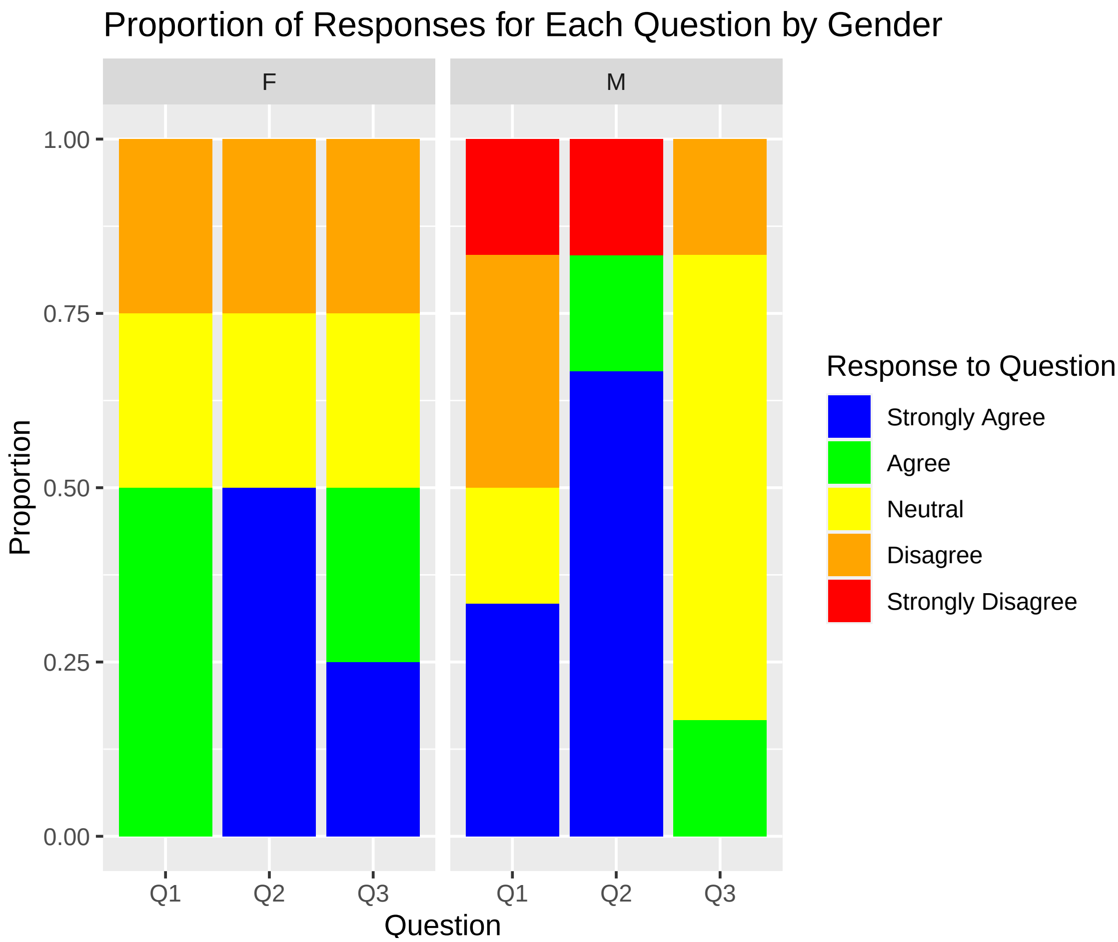
Let's produce such a plot for each gender. This illustrates one of the great advantages of using `ggplot2`:

```
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar() +
    labs(fill = "Response to Question") +
    scale_y_continuous(breaks = seq(from = 0, to = nrow(simple_questionnaire), by = 2)) +
    scale_fill_manual(guide = guide_legend(reverse=TRUE),
                        values =  c("Strongly Disagree" = "red",
                            "Disagree" = "orange",
                            "Neutral" = "yellow",
                            "Agree" = "green",
                            "Strongly Agree" = "blue")) +
    facet_grid(. ~ Gender) # Gender defines the columns
```



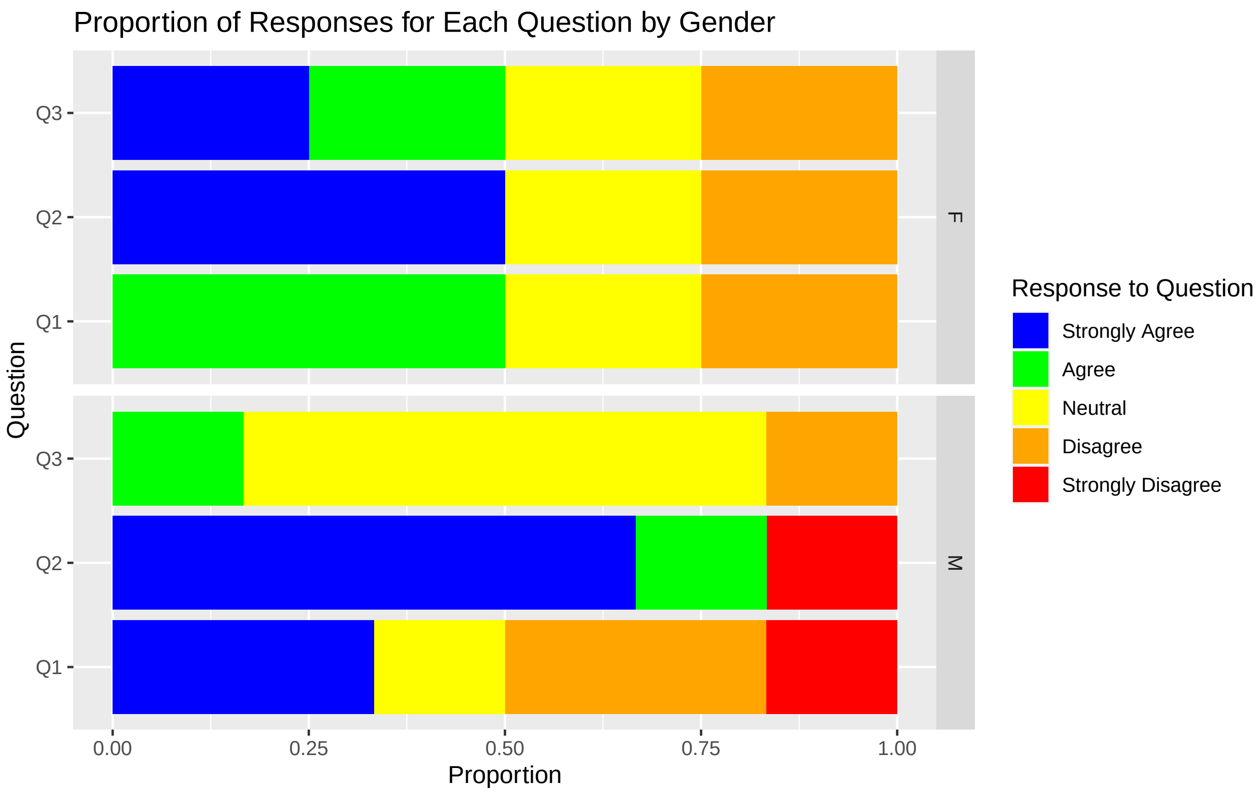
Now let's show proportions (we no longer need to specify the y-axis scale):

```
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar(position = "fill") + # To show proportions
    labs(fill = "Response to Question", y = "Proportion") +
    scale_fill_manual(guide = guide_legend(reverse=TRUE),
                      values =  c("Strongly Disagree" = "red",
                          "Disagree" = "orange",
                          "Neutral" = "yellow",
                          "Agree" = "green",
                          "Strongly Agree" = "blue")) +
    facet_grid(. ~ Gender) +
    ggtitle("Proportion of Responses for Each Question by Gender")
```



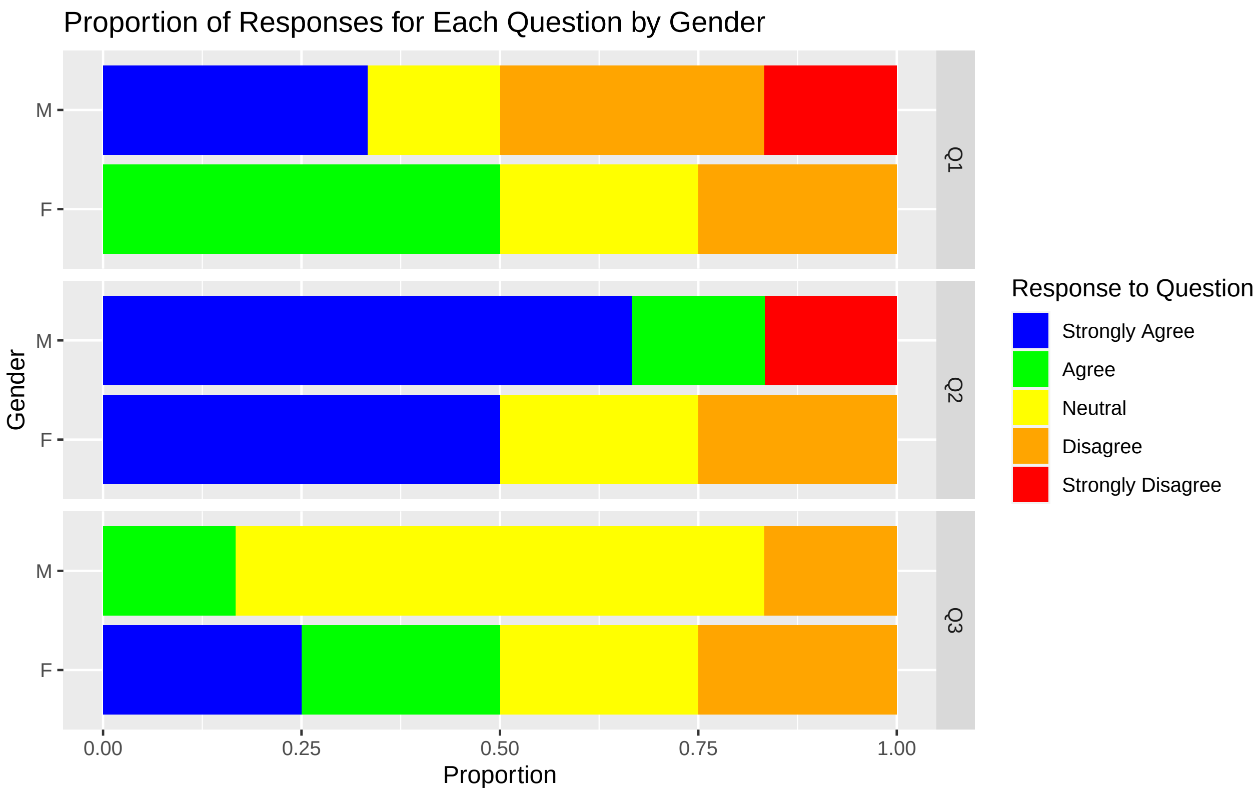
Our eyes may find it easier to make comparisons between the genders if the bars run horizontally and if we assign each gender to a row, rather than a column:

```
ggplot(simple_questionnaire_2_f, aes(x = Question, fill = Question_Response)) +
    geom_bar(position = "fill") + # To show proportions
    labs(fill = "Response to Question", y = "Proportion") +
    scale_fill_manual(guide = guide_legend(reverse=TRUE),
                        values =  c("Strongly Disagree" = "red",
                            "Disagree" = "orange",
                            "Neutral" = "yellow",
                            "Agree" = "green",
                            "Strongly Agree" = "blue")) +
    coord_flip() + # Horizontal bars
    facet_grid(Gender ~ .) +
    ggtitle("Proportion of Responses for Each Question by Gender")
```



It's not hard to modify this so that we can compare the results of each question across gender.

```
ggplot(simple_questionnaire_2_f, aes(x = Gender, fill = Question_Response)) +
    geom_bar(position = "fill") + # To show proportions
    labs(fill = "Response to Question", y = "Proportion") +
    scale_fill_manual(guide = guide_legend(reverse=TRUE),
                        values =  c("Strongly Disagree" = "red",
                            "Disagree" = "orange",
                            "Neutral" = "yellow",
                            "Agree" = "green",
                            "Strongly Agree" = "blue")) +
    coord_flip() + # Horizontal bars
    facet_grid(Question ~ .) +
    ggtitle("Proportion of Responses for Each Question by Gender")
```



11.5 The `likert` Package

We can also use the `likert` package to obtain informative and attractive representations of the data. Here are some examples.

```
#
# You should use the latest version of the likert package, available from GitHub
#
# If you don't have the latest version of the likert package from GitHub,
# you'll need to do the following
#
# To install from GitHub, you'll need the devtools package
#
install.packages("devtools", repos = "http://www.stats.bris.ac.uk/R/")
#
require(devtools)
#
# Now install the latest version of the likert package
#
devtools::install_github('jbryer/likert')
```

Now we should be able to use the `likert` package to produce summaries and a plot of the data:

```
require(likert)
citation("likert")
```

```
##
## To cite package 'likert' in publications use:
##
##   Jason Bryer and Kimberly Speerschneider (2016). likert: Analysis and
```

```
##   Visualization Likert Items. R package version 1.3.5.
##   https://CRAN.R-project.org/package=likert
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {likert: Analysis and Visualization Likert Items},
##     author = {Jason Bryer and Kimberly Speerschneider},
##     year = {2016},
##     note = {R package version 1.3.5},
##     url = {https://CRAN.R-project.org/package=likert},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
```

```
#
# We just need the responses to the questions
# Select columns Q1 through to Q3.  These need to be factors,
# perhaps with the same number of levels
#
Answers_to_questions <- simple_questionnaire_f2 %>% select(Q1:Q3)
Answers_to_questions
```

```
## # A tibble: 10 x 3
##    Q1                Q2                Q3
##    <fct>             <fct>             <fct>
##  1 Neutral           Strongly Agree    Neutral
##  2 Strongly Disagree Strongly Agree    Neutral
##  3 Disagree          Strongly Agree    Neutral
##  4 Neutral           Neutral           Agree
##  5 Agree             Strongly Agree    Disagree
##  6 Strongly Agree    Strongly Agree    Neutral
##  7 Strongly Agree    Strongly Agree    Disagree
##  8 Disagree          Strongly Disagree Neutral
##  9 Disagree          Agree             Agree
## 10 Agree             Disagree          Strongly Agree
```

```
str(Answers_to_questions)
```

```
## tibble [10 x 3] (S3: tbl_df/tbl/data.frame)
##  $ Q1: Factor w/ 5 levels "Strongly Disagree",..: 3 1 2 3 4 5 5 2 2 4
##  $ Q2: Factor w/ 5 levels "Strongly Disagree",..: 5 5 5 3 5 5 5 1 4 2
##  $ Q3: Factor w/ 5 levels "Strongly Disagree",..: 3 3 3 4 2 3 2 3 4 5
```

```
#
# This is more than just a dataframe, which seems to confuse likert
# So first, we ensure it's just a dataframe
#
Answers_to_questions_df <- data.frame(Answers_to_questions)
str(Answers_to_questions_df)
```

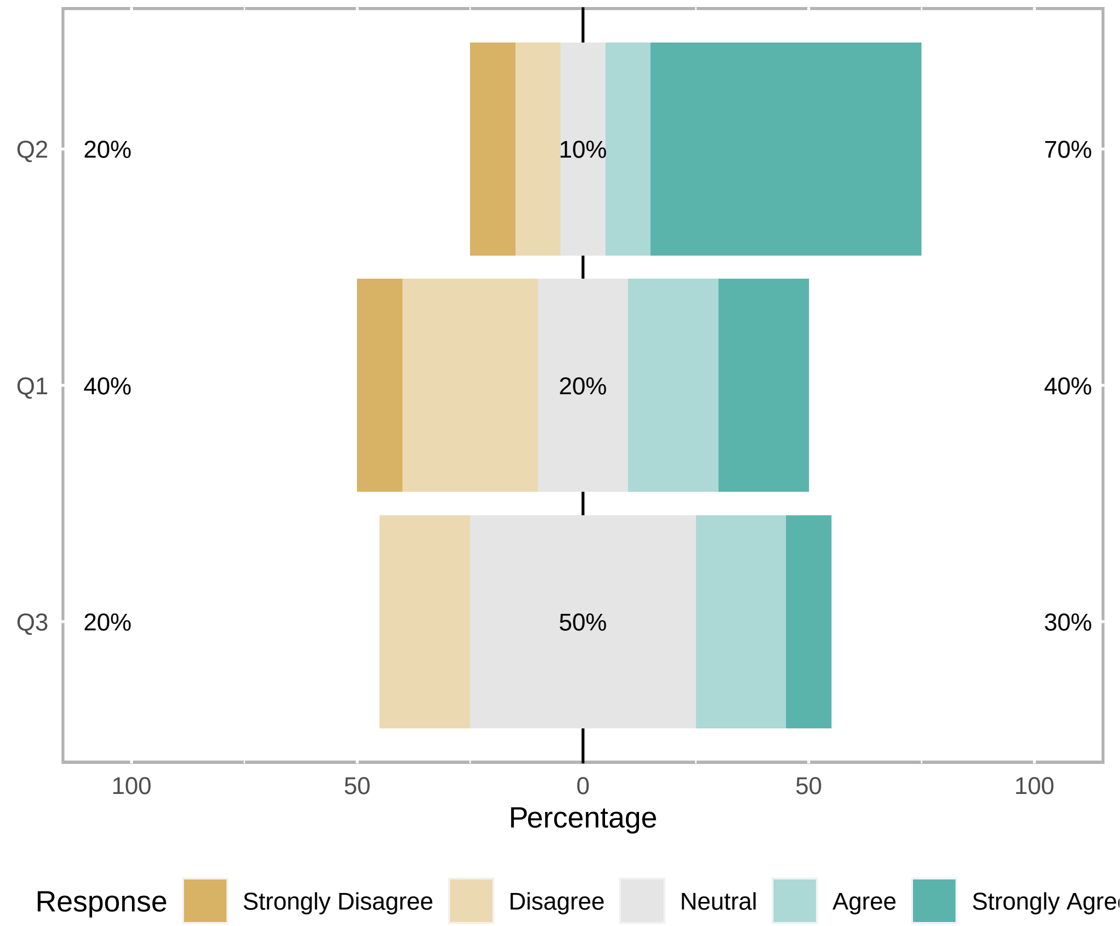
```
## 'data.frame':    10 obs. of  3 variables:
##  $ Q1: Factor w/ 5 levels "Strongly Disagree",..: 3 1 2 3 4 5 5 2 2 4
##  $ Q2: Factor w/ 5 levels "Strongly Disagree",..: 5 5 5 3 5 5 5 1 4 2
```

```
##  $ Q3: Factor w/ 5 levels "Strongly Disagree",..: 3 3 3 4 2 3 2 3 4 5
```

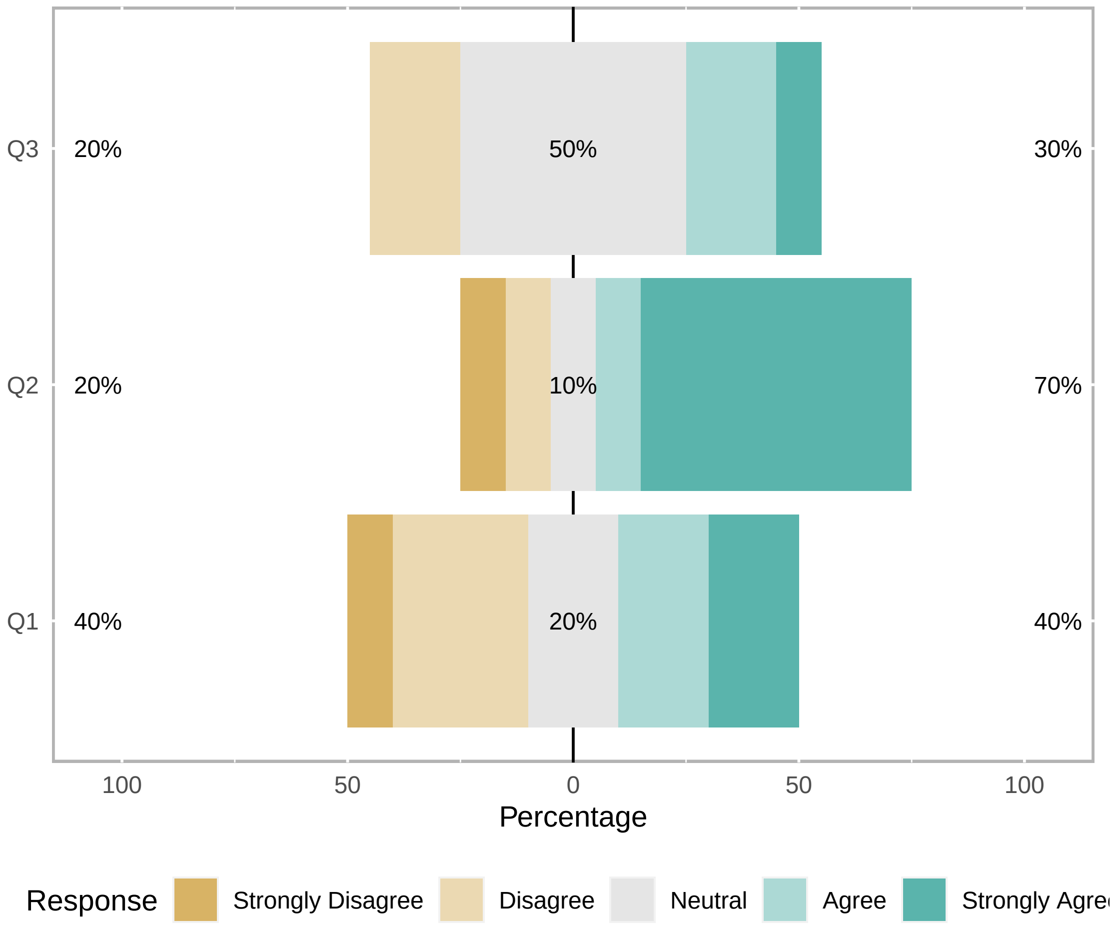
```r
#
# Now convert to a likert class, summarise and plot
#
Answers_likert <- likert(Answers_to_questions_df)
summary(Answers_likert)
```

```
##   Item low neutral high mean        sd
## 2   Q2  20      10   70  4.0 1.4907120
## 1   Q1  40      20   40  3.1 1.3703203
## 3   Q3  20      50   30  3.2 0.9189366
```

```r
# Low/high corresponds to categories below/above Neutral
plot(Answers_likert)
```

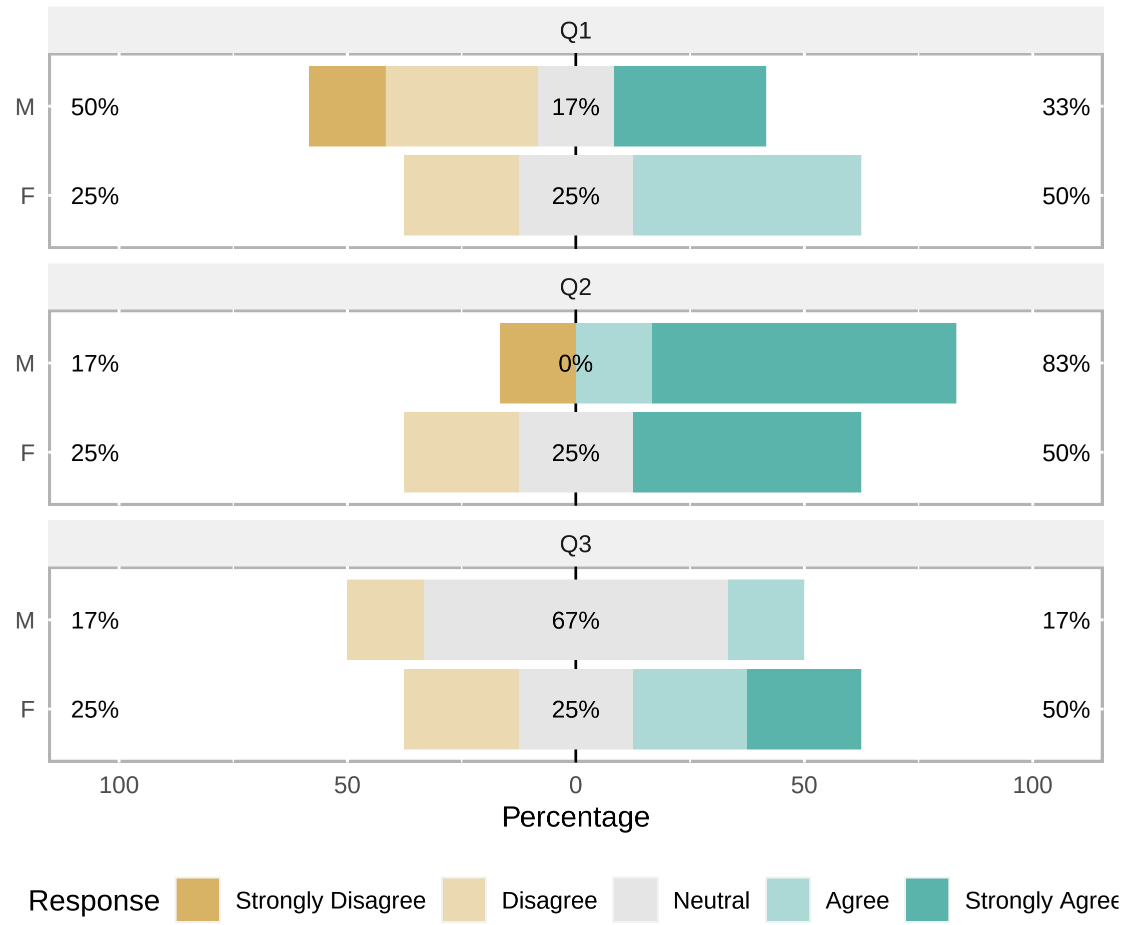



```r
#
# The summary and plot are ordered on "high"
# To show the questions in their natural order
#
plot(Answers_likert, ordered = FALSE)
```



Finally, we can group by gender, which must be a factor:

```
Gender_f <- simple_questionnaire_f2$Gender
Answers_likert_gender <- likert(items = Answers_to_questions_df,
                                grouping = Gender_f)
plot(Answers_likert_gender)
```



These plots contain essentially the same information as out `ggplot2` bar charts. However, the way they are presented makes it much easier for us to understand what is going on in questionnaire data. Make sure that you understand them.

Unfortunately, we have periodically had problems with the `likert` package, so do not be surprised if your code does not work. If it doesn't, do not hesitate to ask us for assistance. We'll try our best to resolve the problems.

11.6 Cross-Tabulation and Testing for Dependence

Now let's perform some cross-tabulation. For example, we may want to investigate the dependence between the responses to Question 1 and Question 2.

```
table(simple_questionnaire$Q1, simple_questionnaire$Q2)
```

```
##
##                    Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                0        1       0              1                 0
##   Disagree             1        0       0              1                 1
##   Neutral              0        0       1              1                 0
##   Strongly Agree       0        0       0              2                 0
##   Strongly Disagree    0        0       0              1                 0
```

We can save typing using `with` which tells R to work with a particular object:

```
with(simple_questionnaire, table(Q1, Q2))
```

```
##                     Q2
## Q1                   Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                 0        1       0              1                 0
##   Disagree              1        0       0              1                 1
##   Neutral               0        0       1              1                 0
##   Strongly Agree        0        0       0              2                 0
##   Strongly Disagree     0        0       0              1                 0
```

We could actually test to see whether there is an underlying dependence between the responses to Question 1 and Question 2. By an "underlying" dependence we mean in a larger population from which the ten people are drawn.

To do this we use a statistical test called the chi-squared test, or χ^2 test. Please note that with such a small data set, it makes little sense to perform such a test. We include this for illustrative purposes only. To perform the test:

```
table_Q1_Q2 <- with(simple_questionnaire, table(Q1, Q2))
table_Q1_Q2
```

```
##                     Q2
## Q1                   Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                 0        1       0              1                 0
##   Disagree              1        0       0              1                 1
##   Neutral               0        0       1              1                 0
##   Strongly Agree        0        0       0              2                 0
##   Strongly Disagree     0        0       0              1                 0
```

```
chisq.test(table_Q1_Q2)
```

```
##
##  Pearson's Chi-squared test
##
## data:  table_Q1_Q2
## X-squared = 13.889, df = 16, p-value = 0.607
```

If the p-value (here, it's 0.607) is less than 0.05 we would conclude that there was a statistically significant underlying dependence between the responses to Question 1 and Question 2. Here, possibly due to small sample size, the p-value is very much bigger than 0.05 and so the data provide no evidence for an underlying dependence.

Another way to cross-tabulate data is using the `xtabs` function:

```
xtabs(~ Q1 + Q2, data = simple_questionnaire)
```

```
##                     Q2
## Q1                   Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                 0        1       0              1                 0
##   Disagree              1        0       0              1                 1
##   Neutral               0        0       1              1                 0
##   Strongly Agree        0        0       0              2                 0
##   Strongly Disagree     0        0       0              1                 0
```

This can be extended:

```
table_Q1_Q2_Q3 <- xtabs(~ Q1 + Q2 + Q3, data = simple_questionnaire)
table_Q1_Q2_Q3
```

```
## , , Q3 = Agree
##
```

```
##                     Q2
## Q1                   Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                 0        0       0              0                 0
##   Disagree              1        0       0              0                 0
##   Neutral               0        0       1              0                 0
##   Strongly Agree        0        0       0              0                 0
##   Strongly Disagree     0        0       0              0                 0
##
## , , Q3 = Disagree
##
##                     Q2
## Q1                   Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                 0        0       0              1                 0
##   Disagree              0        0       0              0                 0
##   Neutral               0        0       0              0                 0
##   Strongly Agree        0        0       0              1                 0
##   Strongly Disagree     0        0       0              0                 0
##
## , , Q3 = Neutral
##
##                     Q2
## Q1                   Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                 0        0       0              0                 0
##   Disagree              0        0       0              1                 1
##   Neutral               0        0       0              1                 0
##   Strongly Agree        0        0       0              1                 0
##   Strongly Disagree     0        0       0              1                 0
##
## , , Q3 = Strongly Agree
##
##                     Q2
## Q1                   Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                 0        1       0              0                 0
##   Disagree              0        0       0              0                 0
##   Neutral               0        0       0              0                 0
##   Strongly Agree        0        0       0              0                 0
##   Strongly Disagree     0        0       0              0                 0
```

```r
ftable(table_Q1_Q2_Q3) # A better display
```

```
##                                   Q3 Agree Disagree Neutral Strongly Agree
## Q1                Q2
## Agree             Agree                0        0       0              0
##                   Disagree             0        0       0              1
##                   Neutral              0        0       0              0
##                   Strongly Agree       0        1       0              0
##                   Strongly Disagree    0        0       0              0
## Disagree          Agree                1        0       0              0
##                   Disagree             0        0       0              0
##                   Neutral              0        0       0              0
##                   Strongly Agree       0        0       1              0
##                   Strongly Disagree    0        0       1              0
## Neutral           Agree                0        0       0              0
##                   Disagree             0        0       0              0
##                   Neutral              1        0       0              0
```

```
##                    Strongly Agree          0         0        1               0
##                    Strongly Disagree       0         0        0               0
## Strongly Agree     Agree                   0         0        0               0
##                    Disagree                0         0        0               0
##                    Neutral                 0         0        0               0
##                    Strongly Agree          0         1        1               0
##                    Strongly Disagree       0         0        0               0
## Strongly Disagree  Agree                   0         0        0               0
##                    Disagree                0         0        0               0
##                    Neutral                 0         0        0               0
##                    Strongly Agree          0         0        1               0
##                    Strongly Disagree       0         0        0               0
```

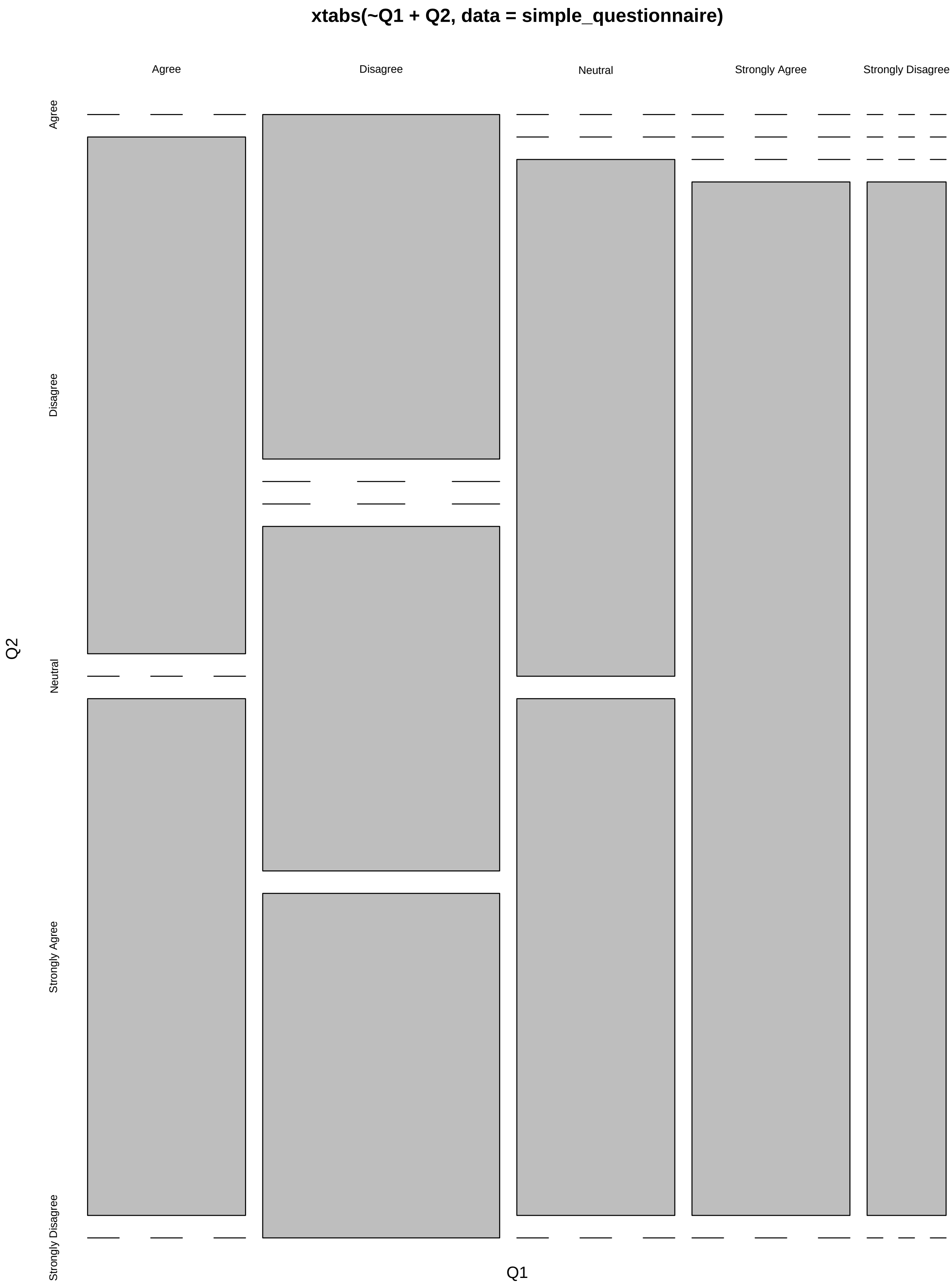
One graphical display of such tables is a mosaic plot. Here the mosaic plot is not particularly effective as there are many zeros (shown by a dash) in the table.

- Mosaic Plot Display of the Cross-tabulation of Questions 1 and 2

```
t(xtabs(~ Q1 + Q2, data = simple_questionnaire)) # Transposed to correspond to the mosaic plot
```

```
##                      Q1
## Q2                    Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                   0        1       0              0                 0
##   Disagree                1        0       0              0                 0
##   Neutral                 0        0       1              0                 0
##   Strongly Agree          1        1       1              2                 1
##   Strongly Disagree       0        1       0              0                 0
```

```
mosaicplot(xtabs(~ Q1 + Q2, data = simple_questionnaire))
```


xtabs(~Q1 + Q2, data = simple_questionnaire)
Agree
Disagree
Neutral
Strongly Agree
Strongly Disagree
Q2
Agree
Disagree
Neutral
Strongly Agree
Strongly Disagree
Q1

In complicated cases the mosaic plot may need careful consideration before it is fully understood.

For a better example of a mosaic plot, we need to return to the larger questionnaire data:

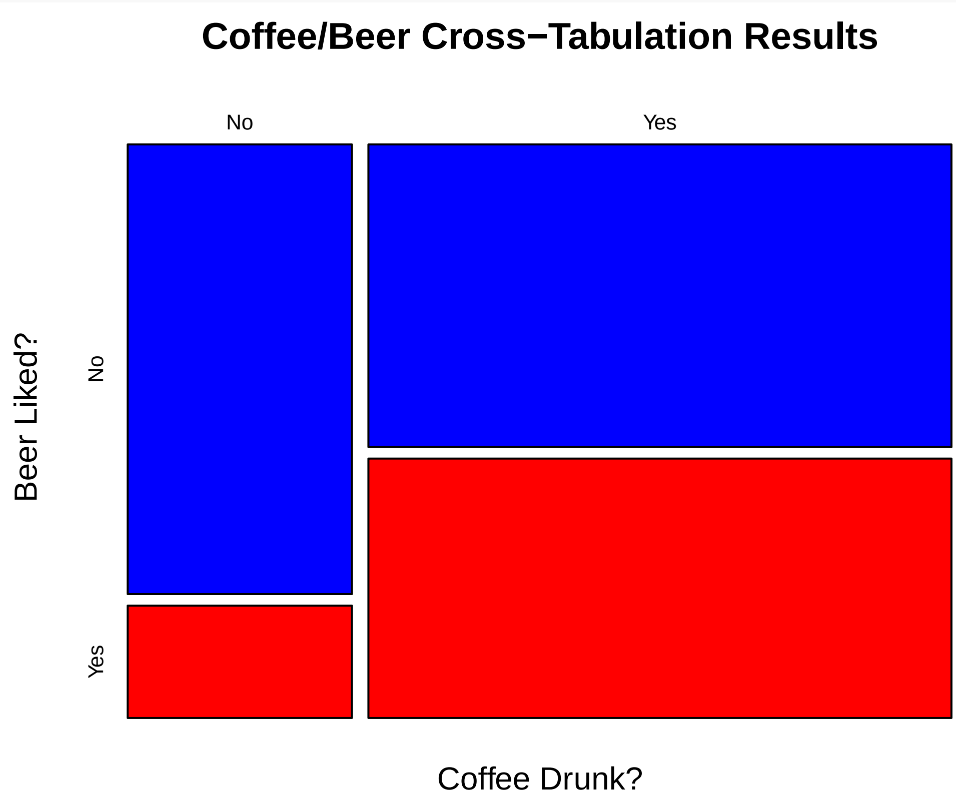
```
setwd("~/Documents/backup_22_11_2021/Plym_teaching/MATH513/2019_20/Introduction_to_R") # As appropriate
require(readr)
qd <- read_csv("MATH513_Questionnaire_Data.csv")
names(qd) # Variable or column names
```

```
##  [1] "Height"              "Age"                 "Sex"
##  [4] "BirthPlace"          "SiblingsNo"          "EatMeat"
##  [7] "DrinkCoffee"         "LikeBeer"            "Sports"
## [10] "Driver"              "LeftHanded"          "Abroad"
## [13] "Sleep"               "Rent"                "Happy_accommodation"
## [16] "Distance"            "Travel_time"         "Mode_of_transport"
## [19] "Safe"
```

```
t(xtabs(~ DrinkCoffee + LikeBeer, data = qd))
```

```
##         DrinkCoffee
## LikeBeer No Yes
##      No   4   7
##      Yes  1   6
```

```
mosaicplot(xtabs(~ DrinkCoffee + LikeBeer, data = qd),
           color = c("blue", "red"),
           xlab = "Coffee Drunk?",
           ylab = "Beer Liked?",
           main = "Coffee/Beer Cross-Tabulation Results")
```



We can also perform in `dplyr` cross-tabulations such as the above for the simple questionnaire:

```r
table_Q1_Q2 <- with(simple_questionnaire, table(Q1, Q2))
table_Q1_Q2
```

```
##                    Q2
## Q1                   Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                  0        1       0              1                 0
##   Disagree               1        0       0              1                 1
##   Neutral                0        0       1              1                 0
##   Strongly Agree         0        0       0              2                 0
##   Strongly Disagree      0        0       0              1                 0
```

The `dplyr` approach is more flexible and yields output with which we can more easily work:

```r
simple_questionnaire %>% group_by(Q1, Q2) %>%
    summarise(n = n())
```

```
## # A tibble: 9 x 3
## # Groups:   Q1 [5]
##   Q1                Q2                    n
##   <chr>             <chr>             <int>
## 1 Agree             Disagree              1
## 2 Agree             Strongly Agree        1
## 3 Disagree          Agree                 1
## 4 Disagree          Strongly Agree        1
## 5 Disagree          Strongly Disagree     1
## 6 Neutral           Neutral               1
## 7 Neutral           Strongly Agree        1
## 8 Strongly Agree    Strongly Agree        2
## 9 Strongly Disagree Strongly Agree        1
```

```r
# The function n() enumerates the number of observations in the current group
```

This looks at all possible combinations of `Q1` and `Q2` and counts the number of occurrences. It does not report zero occurrences.

We can turn this into tabular format using `spread` from `tidyr`:

```r
require(tidyr)
simple_questionnaire %>% group_by(Q1, Q2) %>%
    summarise(n = n()) %>%
    spread(Q2, n)
```

```
## # A tibble: 5 x 6
## # Groups:   Q1 [5]
##   Q1                Agree Disagree Neutral `Strongly Agree` `Strongly Disagree`
##   <chr>             <int>    <int>   <int>            <int>               <int>
## 1 Agree                NA        1      NA                1                  NA
## 2 Disagree              1       NA      NA                1                   1
## 3 Neutral              NA       NA       1                1                  NA
## 4 Strongly Agree       NA       NA      NA                2                  NA
## 5 Strongly Disagree    NA       NA      NA                1                  NA
```

```r
# The columns are provided by Q2 and the values by n
```

We can `fill` using zeros:

```r
table_Q1_Q2_dplyr <- simple_questionnaire %>% group_by(Q1, Q2) %>%
    summarise(n = n()) %>%
```

```r
    spread(Q2, n, fill = 0)
table_Q1_Q2_dplyr
```

```
## # A tibble: 5 x 6
## # Groups:   Q1 [5]
##   Q1                Agree Disagree Neutral `Strongly Agree` `Strongly Disagree`
##   <chr>             <dbl>    <dbl>   <dbl>            <dbl>               <dbl>
## 1 Agree                 0        1       0                1                   0
## 2 Disagree              1        0       0                1                   1
## 3 Neutral               0        0       1                1                   0
## 4 Strongly Agree        0        0       0                2                   0
## 5 Strongly Disagree     0        0       0                1                   0
```

```r
table_Q1_Q2
```

```
##                    Q2
## Q1                  Agree Disagree Neutral Strongly Agree Strongly Disagree
##   Agree                 0        1       0              1                 0
##   Disagree              1        0       0              1                 1
##   Neutral               0        0       1              1                 0
##   Strongly Agree        0        0       0              2                 0
##   Strongly Disagree     0        0       0              1                 0
```

```r
str(table_Q1_Q2_dplyr)
```

```
## grouped_df [5 x 6] (S3: grouped_df/tbl_df/tbl/data.frame)
##  $ Q1               : chr [1:5] "Agree" "Disagree" "Neutral" "Strongly Agree" ...
##  $ Agree            : num [1:5] 0 1 0 0 0
##  $ Disagree         : num [1:5] 1 0 0 0 0
##  $ Neutral          : num [1:5] 0 0 1 0 0
##  $ Strongly Agree   : num [1:5] 1 1 1 2 1
##  $ Strongly Disagree: num [1:5] 0 1 0 0 0
##  - attr(*, "groups")= tibble [5 x 2] (S3: tbl_df/tbl/data.frame)
##   ..$ Q1   : chr [1:5] "Agree" "Disagree" "Neutral" "Strongly Agree" ...
##   ..$ .rows: list<int> [1:5]
##   .. ..$ : int 1
##   .. ..$ : int 2
##   .. ..$ : int 3
##   .. ..$ : int 4
##   .. ..$ : int 5
##   .. ..@ ptype: int(0)
##   ..- attr(*, ".drop")= logi TRUE
```

```r
str(table_Q1_Q2)
```

```
##  'table' int [1:5, 1:5] 0 1 0 0 0 1 0 0 0 0 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ Q1: chr [1:5] "Agree" "Disagree" "Neutral" "Strongly Agree" ...
##   ..$ Q2: chr [1:5] "Agree" "Disagree" "Neutral" "Strongly Agree" ...
```

Note that the `dplyr` approach yields a dataframe.

11.7 A Summary Example, with Some Extensions

We finish this section by considering a different example in order to summarize some of the techniques that we have met.

The data in the file `likert_example_numeric.csv` contain questionnaire results from twenty people. The gender of each person and whether he or she is a graduate is also recorded. The responses to each question are recorded as numerical values 1, 2, 3, 4 and 5 representing "Strongly Disagree", "Disagree", "Neutral", "Agree" and "Strongly Agree". As we have seen, the assignment of numers to categories is rather arbitrary, so we should be careful if we work with the numbers themselves.

- Read in the Data

```
require(readr)
setwd("~/Documents/backup_22_11_2021/Plym_teaching/MATH513/2019_20/Introduction_to_R")
# Your working directory will probably be different
Questionnaire_original <- read_csv("likert_example_numeric.csv")
Questionnaire_original
```

```
## # A tibble: 20 x 7
##    Gender Graduate    Q1    Q2    Q3    Q4    Q5
##    <chr>  <chr>    <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 M      Yes          4     2     4     5     4
##  2 M      Yes          5     1     4     4     2
##  3 M      Yes          3     3     5     5     1
##  4 M      Yes          4     2     5     4     2
##  5 M      Yes          4     2     5     4     1
##  6 F      Yes          3     4     2     2     2
##  7 F      Yes          4     4     2     4     1
##  8 F      Yes          3     5     4     4     4
##  9 F      Yes          4     5     4     5     4
## 10 F      Yes          4     4     2     5     1
## 11 M      No           3     3     5     3     2
## 12 M      No           3     2     4     2     3
## 13 M      No           4     2     4     4     3
## 14 M      No           4     1     4     1     3
## 15 M      No           3     1     3     4     3
## 16 F      No           3     5     4     3     2
## 17 F      No           2     5     4     3     2
## 18 F      No           2     3     5     2     3
## 19 F      No           3     3     4     2     2
## 20 F      No           3     4     5     4     2
```

```
str(Questionnaire_original)
```

```
## spec_tbl_df [20 x 7] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Gender  : chr [1:20] "M" "M" "M" "M" ...
##  $ Graduate: chr [1:20] "Yes" "Yes" "Yes" "Yes" ...
##  $ Q1      : num [1:20] 4 5 3 4 4 3 4 3 4 4 ...
##  $ Q2      : num [1:20] 2 1 3 2 2 4 4 5 5 4 ...
##  $ Q3      : num [1:20] 4 4 5 5 5 2 2 4 4 2 ...
##  $ Q4      : num [1:20] 5 4 5 4 4 2 4 4 5 5 ...
##  $ Q5      : num [1:20] 4 2 1 2 1 2 1 4 4 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Gender = col_character(),
##   ..   Graduate = col_character(),
##   ..   Q1 = col_double(),
##   ..   Q2 = col_double(),
##   ..   Q3 = col_double(),
##   ..   Q4 = col_double(),
```

```
##   ..   Q5 = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

- Convert Variables to Factors, as appropriate

We need a slightly different factor conversion function:

```
#
# Question Results
#
factor_5_numeric <- function(x){
    factor(x,
           levels = 1:5,
           labels = c("Strongly Disagree", "Disagree", "Neutral", "Agree", "Strongly Agree"))
}
#
require(dplyr)
#
Questionnaire_2 <- Questionnaire_original  %>% mutate_each(funs(factor_5_numeric), Q1:Q5)
Questionnaire_2
```

```
## # A tibble: 20 x 7
##    Gender Graduate Q1             Q2                Q3             Q4     Q5
##    <chr>  <chr>    <fct>          <fct>             <fct>          <fct>  <fct>
##  1 M      Yes      Agree          Disagree          Agree          Stron~ Agree
##  2 M      Yes      Strongly Agree Strongly Disagree Agree          Agree  Disag~
##  3 M      Yes      Neutral        Neutral           Strongly Agree Stron~ Stron~
##  4 M      Yes      Agree          Disagree          Strongly Agree Agree  Disag~
##  5 M      Yes      Agree          Disagree          Strongly Agree Agree  Stron~
##  6 F      Yes      Neutral        Agree             Disagree       Disag~ Disag~
##  7 F      Yes      Agree          Agree             Disagree       Agree  Stron~
##  8 F      Yes      Neutral        Strongly Agree    Agree          Agree  Agree
##  9 F      Yes      Agree          Strongly Agree    Agree          Stron~ Agree
## 10 F      Yes      Agree          Agree             Disagree       Stron~ Stron~
## 11 M      No       Neutral        Neutral           Strongly Agree Neutr~ Disag~
## 12 M      No       Neutral        Disagree          Agree          Disag~ Neutr~
## 13 M      No       Agree          Disagree          Agree          Agree  Neutr~
## 14 M      No       Agree          Strongly Disagree Agree          Stron~ Neutr~
## 15 M      No       Neutral        Strongly Disagree Neutral        Agree  Neutr~
## 16 F      No       Neutral        Strongly Agree    Agree          Neutr~ Disag~
## 17 F      No       Disagree       Strongly Agree    Agree          Neutr~ Disag~
## 18 F      No       Disagree       Neutral           Strongly Agree Disag~ Neutr~
## 19 F      No       Neutral        Neutral           Agree          Disag~ Disag~
## 20 F      No       Neutral        Agree             Strongly Agree Agree  Disag~
```

```
#
# Gender
#
Questionnaire_2 <- Questionnaire_2 %>% mutate(Gender = factor(Gender, level = c("F","M")))
#
# Graduate
#
Questionnaire_2 <- Questionnaire_2 %>% mutate(Graduate = factor(Graduate, level = c("Yes","No")))
Questionnaire_2
```

```
## # A tibble: 20 x 7
```

```
##    Gender Graduate Q1             Q2                Q3             Q4     Q5
##    <fct>  <fct>    <fct>          <fct>             <fct>          <fct>  <fct>
##  1 M      Yes      Agree          Disagree          Agree          Stron~ Agree
##  2 M      Yes      Strongly Agree Strongly Disagree Agree          Agree  Disag~
##  3 M      Yes      Neutral        Neutral           Strongly Agree Stron~ Stron~
##  4 M      Yes      Agree          Disagree          Strongly Agree Agree  Disag~
##  5 M      Yes      Agree          Disagree          Strongly Agree Agree  Stron~
##  6 F      Yes      Neutral        Agree             Disagree       Disag~ Disag~
##  7 F      Yes      Agree          Agree             Disagree       Agree  Stron~
##  8 F      Yes      Neutral        Strongly Agree    Agree          Agree  Agree
##  9 F      Yes      Agree          Strongly Agree    Agree          Stron~ Agree
## 10 F      Yes      Agree          Agree             Disagree       Stron~ Stron~
## 11 M      No       Neutral        Neutral           Strongly Agree Neutr~ Disag~
## 12 M      No       Neutral        Disagree          Agree          Disag~ Neutr~
## 13 M      No       Agree          Disagree          Agree          Agree  Neutr~
## 14 M      No       Agree          Strongly Disagree Agree          Stron~ Neutr~
## 15 M      No       Neutral        Strongly Disagree Neutral        Agree  Neutr~
## 16 F      No       Neutral        Strongly Agree    Agree          Neutr~ Disag~
## 17 F      No       Disagree       Strongly Agree    Agree          Neutr~ Disag~
## 18 F      No       Disagree       Neutral           Strongly Agree Disag~ Neutr~
## 19 F      No       Neutral        Neutral           Agree          Disag~ Disag~
## 20 F      No       Neutral        Agree             Strongly Agree Agree  Disag~
```

```
str(Questionnaire_2)
```

```
## spec_tbl_df [20 x 7] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Gender  : Factor w/ 2 levels "F","M": 2 2 2 2 2 1 1 1 1 1 ...
##  $ Graduate: Factor w/ 2 levels "Yes","No": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Q1      : Factor w/ 5 levels "Strongly Disagree",..: 4 5 3 4 4 3 4 3 4 4 ...
##  $ Q2      : Factor w/ 5 levels "Strongly Disagree",..: 2 1 3 2 2 4 4 5 5 4 ...
##  $ Q3      : Factor w/ 5 levels "Strongly Disagree",..: 4 4 5 5 5 2 2 4 4 2 ...
##  $ Q4      : Factor w/ 5 levels "Strongly Disagree",..: 5 4 5 4 4 2 4 4 5 5 ...
##  $ Q5      : Factor w/ 5 levels "Strongly Disagree",..: 4 2 1 2 1 2 1 4 4 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Gender = col_character(),
##   ..   Graduate = col_character(),
##   ..   Q1 = col_double(),
##   ..   Q2 = col_double(),
##   ..   Q3 = col_double(),
##   ..   Q4 = col_double(),
##   ..   Q5 = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

- Simple Tabulation

```
table(Questionnaire_2$Q3)
```

```
##
## Strongly Disagree          Disagree           Neutral             Agree
##                 0                 3                 1                10
##    Strongly Agree
##                 6
```

```r
table(Questionnaire_2$Q5)
```

```
##
## Strongly Disagree          Disagree           Neutral             Agree
##                 4                 8                 5                 3
##    Strongly Agree
##                 0
```

```r
#
# Slightly more advanced tabulation
#
Questionnaire_2 %>% group_by(Gender, Q3) %>% summarise(n = n())
```

```
## # A tibble: 6 x 3
## # Groups:   Gender [2]
##   Gender Q3                 n
##   <fct>  <fct>          <int>
## 1 F      Disagree           3
## 2 F      Agree              5
## 3 F      Strongly Agree     2
## 4 M      Neutral            1
## 5 M      Agree              5
## 6 M      Strongly Agree     4
```

```r
Questionnaire_2 %>% group_by(Gender, Graduate,  Q5) %>% summarise(n = n())
```

```
## # A tibble: 10 x 4
## # Groups:   Gender, Graduate [4]
##    Gender Graduate Q5                    n
##    <fct>  <fct>    <fct>             <int>
##  1 F      Yes      Strongly Disagree     2
##  2 F      Yes      Disagree              1
##  3 F      Yes      Agree                 2
##  4 F      No       Disagree              4
##  5 F      No       Neutral               1
##  6 M      Yes      Strongly Disagree     2
##  7 M      Yes      Disagree              2
##  8 M      Yes      Agree                 1
##  9 M      No       Disagree              1
## 10 M      No       Neutral               4
```

- Bar Plots

Manipulate data into long form:

```r
require(tidyr)
#
Questionnaire_2_long <- Questionnaire_2 %>% gather(Question, Question_Response, Q1:Q5)
Questionnaire_2_long
```

```
## # A tibble: 100 x 4
##    Gender Graduate Question Question_Response
##    <fct>  <fct>    <chr>    <chr>
##  1 M      Yes      Q1       Agree
##  2 M      Yes      Q1       Strongly Agree
##  3 M      Yes      Q1       Neutral
##  4 M      Yes      Q1       Agree
```

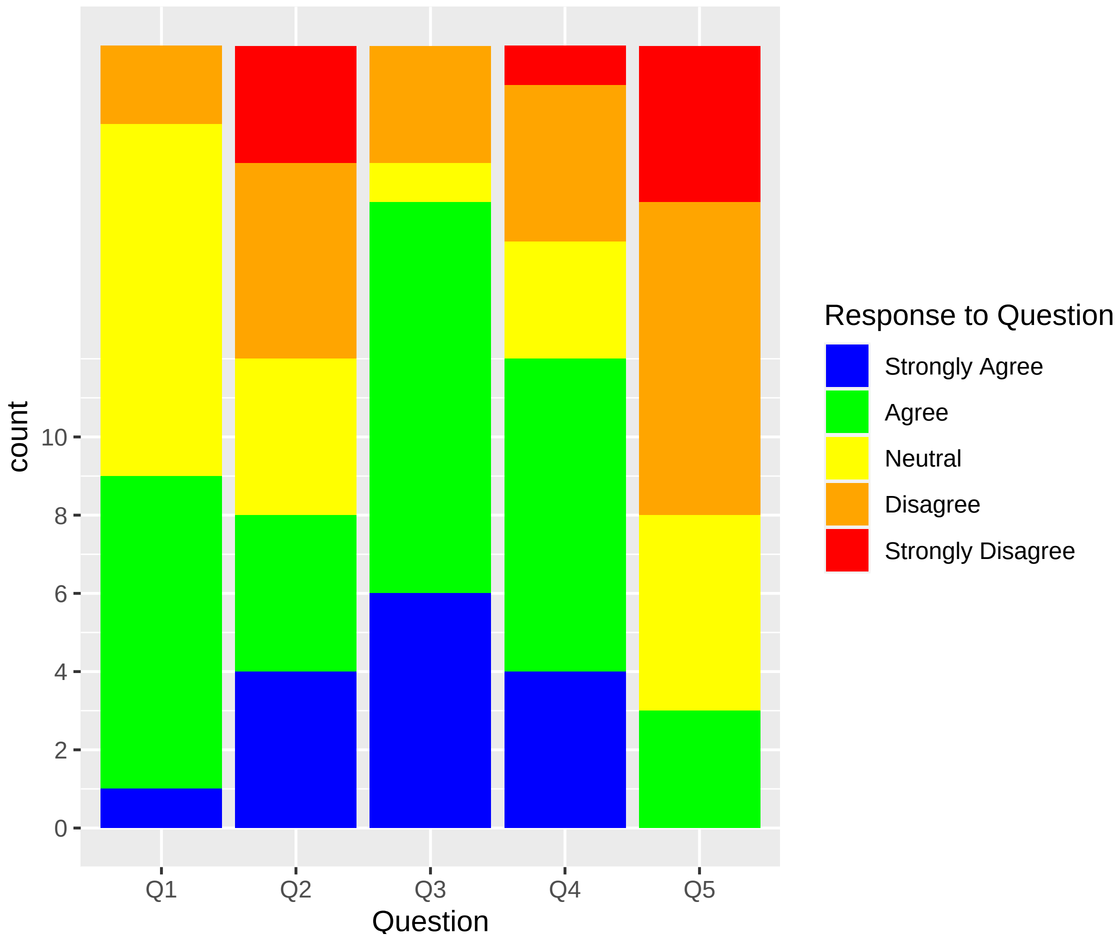
```
##   5 M      Yes      Q1       Agree
##   6 F      Yes      Q1       Neutral
##   7 F      Yes      Q1       Agree
##   8 F      Yes      Q1       Neutral
##   9 F      Yes      Q1       Agree
##  10 F      Yes      Q1       Agree
## # ... with 90 more rows
```

```
Questionnaire_2_long <- Questionnaire_2_long %>%
    mutate(Question_Response = factor(Question_Response,
  levels = c("Strongly Disagree", "Disagree", "Neutral","Agree", "Strongly Agree")))
str(Questionnaire_2_long)
```

```
## tibble [100 x 4] (S3: tbl_df/tbl/data.frame)
##  $ Gender           : Factor w/ 2 levels "F","M": 2 2 2 2 2 1 1 1 1 1 ...
##  $ Graduate         : Factor w/ 2 levels "Yes","No": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Question         : chr [1:100] "Q1" "Q1" "Q1" "Q1" ...
##  $ Question_Response: Factor w/ 5 levels "Strongly Disagree",..: 4 5 3 4 4 3 4 3 4 4 ...
```

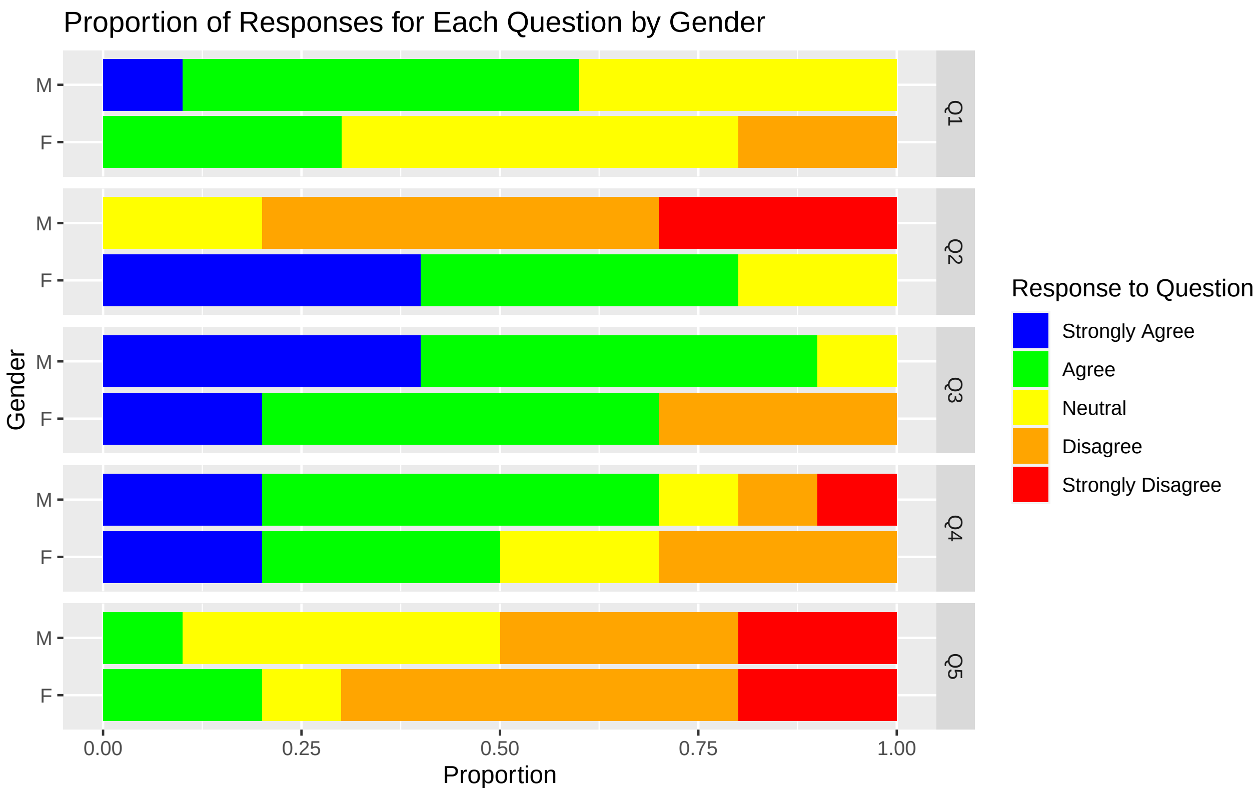
Simple bar plot:

```
require(ggplot2)
#
ggplot(Questionnaire_2_long, aes(x = Question, fill = Question_Response)) +
geom_bar() +
labs(fill = "Response to Question") +
scale_y_continuous(breaks = seq(from = 0, to = nrow(simple_questionnaire), by = 2)) +
scale_fill_manual(guide = guide_legend(reverse=TRUE),
     values = c("Strongly Disagree" = "red",
                "Disagree" = "orange",
                "Neutral" = "yellow",
                "Agree" = "green",
                "Strongly Agree" = "blue"))
```



Faceted bar plots:

```
ggplot(Questionnaire_2_long, aes(x = Gender, fill = Question_Response)) +
geom_bar(position = "fill") + # To show proportions
labs(fill = "Response to Question", y = "Proportion") +
scale_fill_manual(guide = guide_legend(reverse=TRUE),
values = c("Strongly Disagree" = "red",
           "Disagree" = "orange",
           "Neutral" = "yellow",
           "Agree" = "green",
           "Strongly Agree" = "blue")) +
coord_flip() + # Horizontal bars
facet_grid(Question ~ .) +
ggtitle("Proportion of Responses for Each Question by Gender")
```



- Some Numerical Summaries

Work with the original numeric data

```
Questionnaire_original_long <- Questionnaire_original %>%
                                 gather(Question, Question_Response, Q1:Q5)


Questionnaire_original_long <- Questionnaire_original_long %>%
                                 mutate(Gender = factor(Gender, level = c("F","M")))
Questionnaire_original_long <- Questionnaire_original_long %>%
                                 mutate(Graduate = factor(Graduate, level = c("Yes","No")))
#
# Summarise
#
Questionnaire_original_long %>%
    group_by(Question, Gender) %>%
    summarise(m = mean(Question_Response))
```

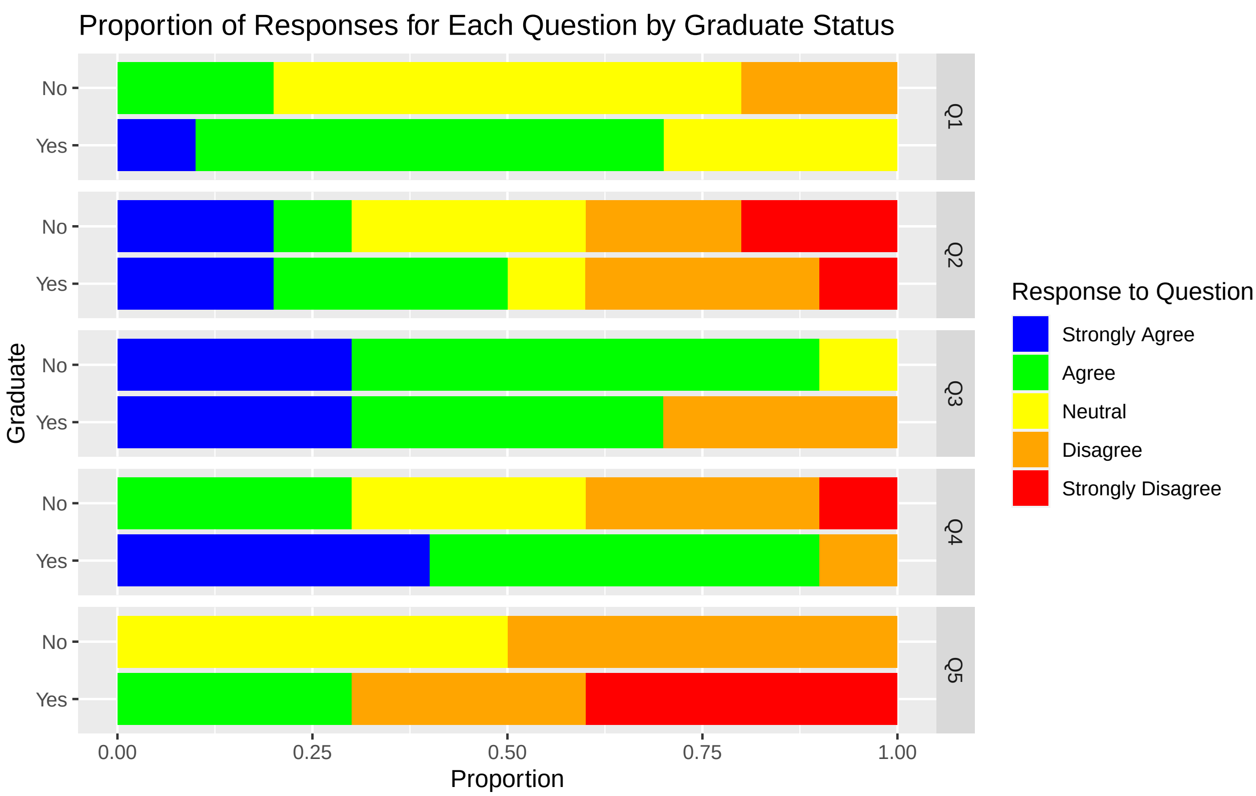
```
## # A tibble: 10 x 3
## # Groups:   Question [5]
##    Question Gender     m
##    <chr>    <fct>  <dbl>
##  1 Q1       F        3.1
##  2 Q1       M        3.7
##  3 Q2       F        4.2
##  4 Q2       M        1.9
##  5 Q3       F        3.6
##  6 Q3       M        4.3
##  7 Q4       F        3.4
```

```
##   8 Q4        M         3.6
##   9 Q5        F         2.3
##  10 Q5        M         2.4
```

Do these agree with the graphs?

- Again

```
ggplot(Questionnaire_2_long, aes(x = Graduate, fill = Question_Response)) +
geom_bar(position = "fill") + # To show proportions
labs(fill = "Response to Question", y = "Proportion") +
scale_fill_manual(guide = guide_legend(reverse=TRUE),
values = c("Strongly Disagree" = "red",
           "Disagree" = "orange",
           "Neutral" = "yellow",
           "Agree" = "green",
           "Strongly Agree" = "blue")) +
coord_flip() + # Horizontal bars
facet_grid(Question ~ .) +
ggtitle("Proportion of Responses for Each Question by Graduate Status")
```




```
#
Questionnaire_original_long %>%
    group_by(Question, Graduate) %>%
    summarise(m = mean(Question_Response))
```

```
## # A tibble: 10 x 3
## # Groups:   Question [5]
##    Question Graduate     m
##    <chr>    <fct>    <dbl>
##  1 Q1       Yes        3.8
```

```
##   2 Q1        No          3
##   3 Q2        Yes         3.2
##   4 Q2        No          2.9
##   5 Q3        Yes         3.7
##   6 Q3        No          4.2
##   7 Q4        Yes         4.2
##   8 Q4        No          2.8
##   9 Q5        Yes         2.2
##  10 Q5        No          2.5
```

- Cross-tabulation and Testing

```
with(Questionnaire_2, table(Q1, Q2))
```

```
##                      Q2
## Q1                    Strongly Disagree Disagree Neutral Agree Strongly Agree
##   Strongly Disagree                   0        0       0     0              0
##   Disagree                            0        0       1     0              1
##   Neutral                             1        1       3     2              2
##   Agree                               1        4       0     2              1
##   Strongly Agree                      1        0       0     0              0
```

```
#
# Test
#
table_Q1_Q2 <- with(Questionnaire_2, table(Q1, Q2))
table_Q1_Q2
```

```
##                      Q2
## Q1                    Strongly Disagree Disagree Neutral Agree Strongly Agree
##   Strongly Disagree                   0        0       0     0              0
##   Disagree                            0        0       1     0              1
##   Neutral                             1        1       3     2              2
##   Agree                               1        4       0     2              1
##   Strongly Agree                      1        0       0     0              0
```

```
#
# Drop the first row as it contains only zeros which invalidates the chi-square test
#
table_Q1_Q2[-1,]
```

```
##                   Q2
## Q1                 Strongly Disagree Disagree Neutral Agree Strongly Agree
##   Disagree                         0        0       1     0              1
##   Neutral                          1        1       3     2              2
##   Agree                            1        4       0     2              1
##   Strongly Agree                   1        0       0     0              0
```

```
#
# Now the test
#
chisq.test(table_Q1_Q2[-1,])
```

```
## 
##  Pearson's Chi-squared test
## 
## data:  table_Q1_Q2[-1, ]
## X-squared = 14.255, df = 12, p-value = 0.2847
```

What do you conclude?

We can get row and column proportions.

```
table_Q1_Q2
```

```
##                     Q2
## Q1                   Strongly Disagree Disagree Neutral Agree Strongly Agree
##   Strongly Disagree                  0        0       0     0              0
##   Disagree                           0        0       1     0              1
##   Neutral                            1        1       3     2              2
##   Agree                              1        4       0     2              1
##   Strongly Agree                     1        0       0     0              0
```

```
prop.table(table_Q1_Q2, margin = 1) # Row proportions
```

```
##                     Q2
## Q1                   Strongly Disagree  Disagree   Neutral     Agree
##   Strongly Disagree
##   Disagree                   0.0000000 0.0000000 0.5000000 0.0000000
##   Neutral                    0.1111111 0.1111111 0.3333333 0.2222222
##   Agree                      0.1250000 0.5000000 0.0000000 0.2500000
##   Strongly Agree             1.0000000 0.0000000 0.0000000 0.0000000
##                     Q2
## Q1                   Strongly Agree
##   Strongly Disagree
##   Disagree                0.5000000
##   Neutral                 0.2222222
##   Agree                   0.1250000
##   Strongly Agree          0.0000000
```

```
prop.table(table_Q1_Q2, margin = 2) # Column proportions
```

```
##                     Q2
## Q1                   Strongly Disagree  Disagree   Neutral     Agree
##   Strongly Disagree          0.0000000 0.0000000 0.0000000 0.0000000
##   Disagree                   0.0000000 0.0000000 0.2500000 0.0000000
##   Neutral                    0.3333333 0.2000000 0.7500000 0.5000000
##   Agree                      0.3333333 0.8000000 0.0000000 0.5000000
##   Strongly Agree             0.3333333 0.0000000 0.0000000 0.0000000
##                     Q2
## Q1                   Strongly Agree
##   Strongly Disagree       0.0000000
##   Disagree                0.2500000
##   Neutral                 0.5000000
##   Agree                   0.2500000
##   Strongly Agree          0.0000000
```

The same results using `dplyr`:

```
table_Q1_Q2_dplyr <- Questionnaire_2 %>% group_by(Q1, Q2) %>%
                       summarise(n = n()) %>%
                       spread(Q2, n, fill = 0)
table_Q1_Q2_dplyr
```

```
## # A tibble: 4 x 6
## # Groups:   Q1 [4]
##   Q1             `Strongly Disagree` Disagree Neutral Agree `Strongly Agree`
```

```
##   <fct>                        <dbl>    <dbl>   <dbl> <dbl>            <dbl>
## 1 Disagree                         0        0       1     0                1
## 2 Neutral                          1        1       3     2                2
## 3 Agree                            1        4       0     2                1
## 4 Strongly Agree                   1        0       0     0                0
```

```
#
Questionnaire_2 %>% group_by(Q1, Q2) %>%
                       summarise(n = n()) %>%
                       mutate(p = n / sum(n)) %>%
                       select(-n) %>%
                       spread(Q2, p, fill = 0)
```

```
## # A tibble: 4 x 6
## # Groups:   Q1 [4]
##   Q1             `Strongly Disagree` Disagree Neutral Agree `Strongly Agree`
##   <fct>                        <dbl>    <dbl>   <dbl> <dbl>            <dbl>
## 1 Disagree                     0        0       0.5   0                0.5
## 2 Neutral                      0.111    0.111   0.333 0.222            0.222
## 3 Agree                        0.125    0.5     0     0.25             0.125
## 4 Strongly Agree               1        0       0     0                0
```

```
#
# Compare
#
prop.table(table_Q1_Q2, margin = 1)
```

```
##                    Q2
## Q1                   Strongly Disagree  Disagree   Neutral     Agree
##   Strongly Disagree
##   Disagree                  0.0000000 0.0000000 0.5000000 0.0000000
##   Neutral                   0.1111111 0.1111111 0.3333333 0.2222222
##   Agree                     0.1250000 0.5000000 0.0000000 0.2500000
##   Strongly Agree            1.0000000 0.0000000 0.0000000 0.0000000
##                    Q2
## Q1                   Strongly Agree
##   Strongly Disagree
##   Disagree               0.5000000
##   Neutral                0.2222222
##   Agree                  0.1250000
##   Strongly Agree         0.0000000
```

```
#
Questionnaire_2 %>% group_by(Q2, Q1) %>%
                       summarise(n = n()) %>%
                       mutate(p = n / sum(n)) %>%
                       select(-n) %>%
                       spread(Q2, p, fill = 0)
```

```
## # A tibble: 4 x 6
##   Q1             `Strongly Disagree` Disagree Neutral Agree `Strongly Agree`
##   <fct>                        <dbl>    <dbl>   <dbl> <dbl>            <dbl>
## 1 Disagree                     0          0      0.25   0                0.25
## 2 Neutral                      0.333      0.2    0.75   0.5              0.5
## 3 Agree                        0.333      0.8    0      0.5              0.25
## 4 Strongly Agree               0.333      0      0      0                0
```

```r
#
# Compare
#
prop.table(table_Q1_Q2, margin = 2)
```

```
##                      Q2
## Q1                    Strongly Disagree  Disagree   Neutral     Agree
##   Strongly Disagree           0.0000000 0.0000000 0.0000000 0.0000000
##   Disagree                    0.0000000 0.0000000 0.2500000 0.0000000
##   Neutral                     0.3333333 0.2000000 0.7500000 0.5000000
##   Agree                       0.3333333 0.8000000 0.0000000 0.5000000
##   Strongly Agree              0.3333333 0.0000000 0.0000000 0.0000000
##                      Q2
## Q1                    Strongly Agree
##   Strongly Disagree        0.0000000
##   Disagree                 0.2500000
##   Neutral                  0.5000000
##   Agree                    0.2500000
##   Strongly Agree           0.0000000
```

```r
#
```

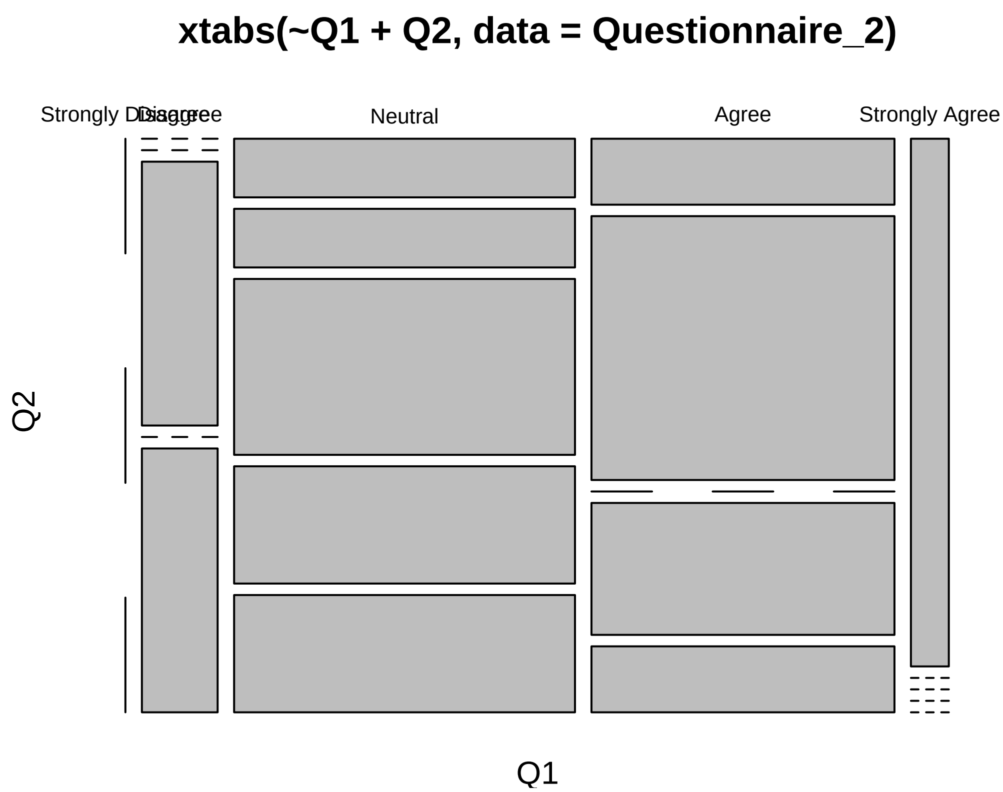
Note that the first row has been removed automatically.

- Mosaic Plot

```r
t(xtabs(~ Q1 + Q2, data = Questionnaire_2))
```

```
##                      Q1
## Q2                    Strongly Disagree Disagree Neutral Agree Strongly Agree
##   Strongly Disagree                   0        0       1     1              1
##   Disagree                            0        0       1     4              0
##   Neutral                             0        1       3     0              0
##   Agree                               0        0       2     2              0
##   Strongly Agree                      0        1       2     1              0
```

```r
mosaicplot(xtabs(~ Q1 + Q2, data = Questionnaire_2))
```



12 General Exercise on Questionnaire Data

Consider again the questionnaire data:

```
require(readr)
qd <- read_csv("MATH513_Questionnaire_Data.csv")
head(qd) # See the first few rows
```

```
## # A tibble: 6 x 19
##   Height   Age Sex    BirthPlace SiblingsNo EatMeat DrinkCoffee LikeBeer Sports
##    <dbl> <dbl> <chr>  <chr>           <dbl> <chr>   <chr>       <chr>    <chr>
## 1    170  23   Female essex               1 Yes     Yes         No       Yes
## 2    188  22.4 Male   London              1 Yes     Yes         No       No
## 3    180  30.1 Male   Athens              0 Yes     Yes         Yes      Yes
## 4    185  21   Male   China               0 Yes     Yes         Yes      Yes
## 5    170  22.1 Female Plymouth            2 Yes     Yes         No       No
## 6    182  25   Male   Nigeria             4 Yes     No          No       Yes
## # ... with 10 more variables: Driver <chr>, LeftHanded <chr>, Abroad <chr>,
## #   Sleep <dbl>, Rent <dbl>, Happy_accommodation <chr>, Distance <dbl>,
## #   Travel_time <dbl>, Mode_of_transport <chr>, Safe <chr>
```

```
names(qd) # Variable or column names
```

```
##  [1] "Height"              "Age"                 "Sex"
##  [4] "BirthPlace"          "SiblingsNo"          "EatMeat"
##  [7] "DrinkCoffee"         "LikeBeer"            "Sports"
## [10] "Driver"              "LeftHanded"          "Abroad"
## [13] "Sleep"               "Rent"                "Happy_accommodation"
## [16] "Distance"            "Travel_time"         "Mode_of_transport"
```

```
## [19] "Safe"
```

Perform further explanatory analyses of these data to try to answer questions that may interest people trying to understand students and the lives that they lead.

For example:

- Do members of the group feel safe when returning to their term time accommodation at night?
- Are people generally happy with their accommodation? Does this depend on how much rent they are paying?
- Do members of the group take part in sport? How does this vary between the genders?
- How does the amount of sleep depend on some of the other variables?
- How do people get to the university?
- Are most people in the group right-handed?
- Etc!