Part IV

Graphical Presentation of Data

4.1 Introduction

- Graphical displays of data can be very useful in showing the main features of a data set.

4.1 Introduction

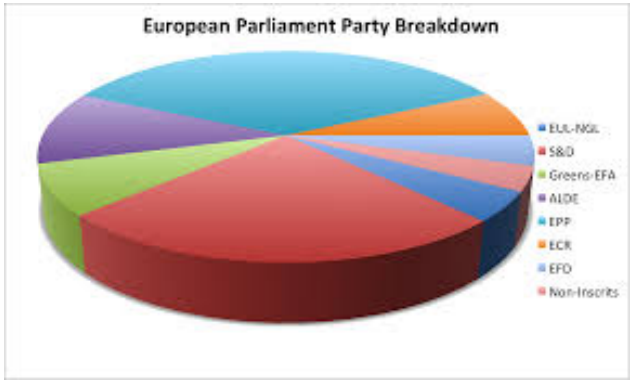
- Graphical displays of data can be very useful in showing the main features of a data set.
- The appropriate form of graph depends on the nature of the variables being displayed and what aspects are to be shown.

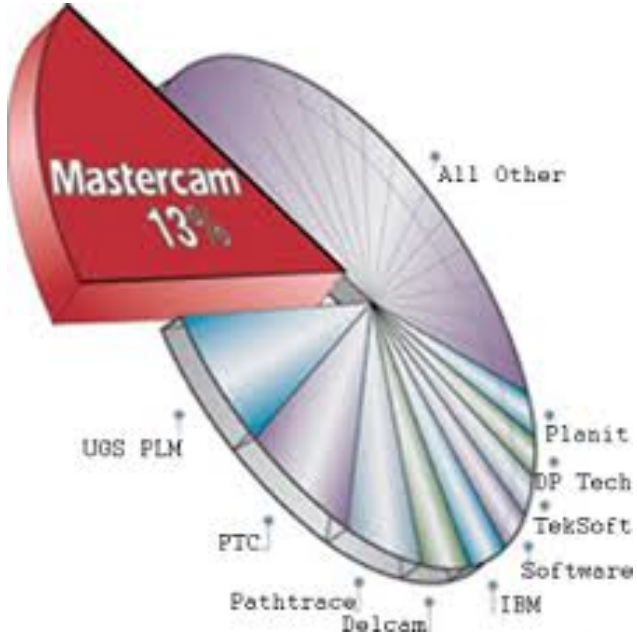
4.1 Introduction

- Graphical displays of data can be very useful in showing the main features of a data set.
- The appropriate form of graph depends on the nature of the variables being displayed and what aspects are to be shown.
- However it should always be borne in mind that the object is to provide a clear and truthful representation of the data, not to distort and not to impress with unnecessary “fancy” features.

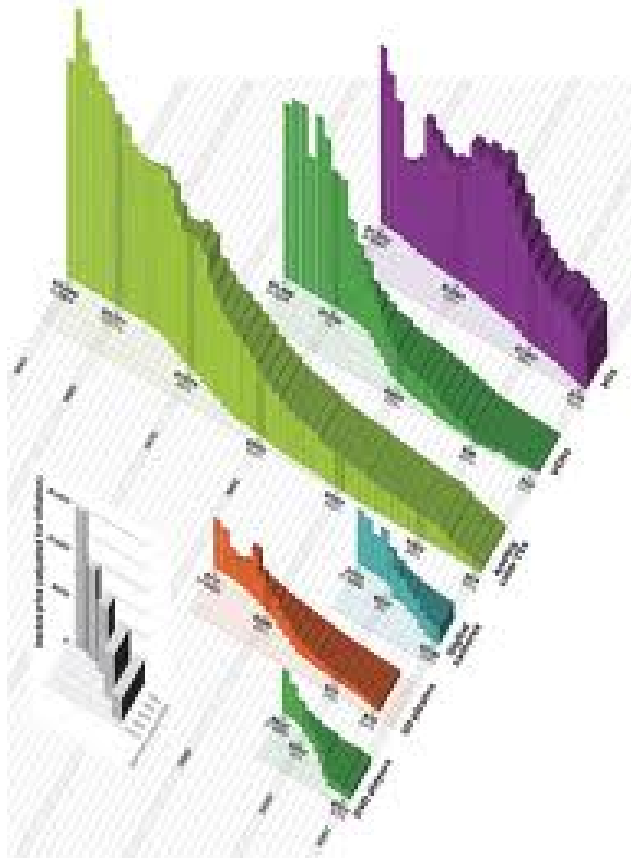
4.1 Introduction

- Graphical displays of data can be very useful in showing the main features of a data set.
- The appropriate form of graph depends on the nature of the variables being displayed and what aspects are to be shown.
- However it should always be borne in mind that the object is to provide a clear and truthful representation of the data, not to distort and not to impress with unnecessary “fancy” features.

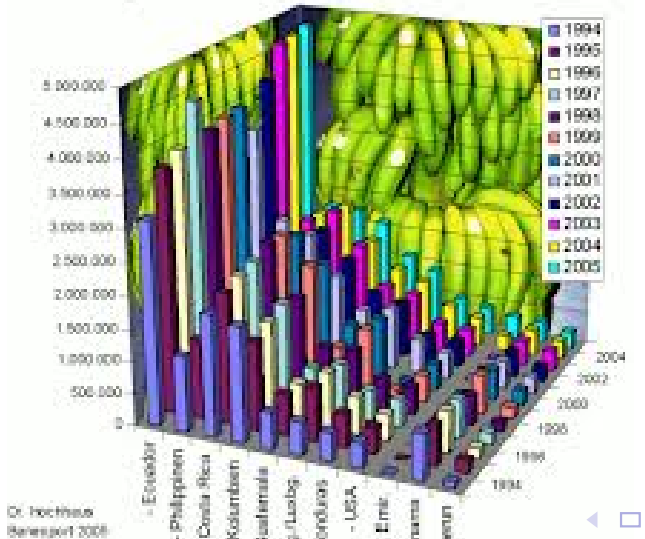




http://www.fusioncharts.com/explore/pie-doughnut-charts

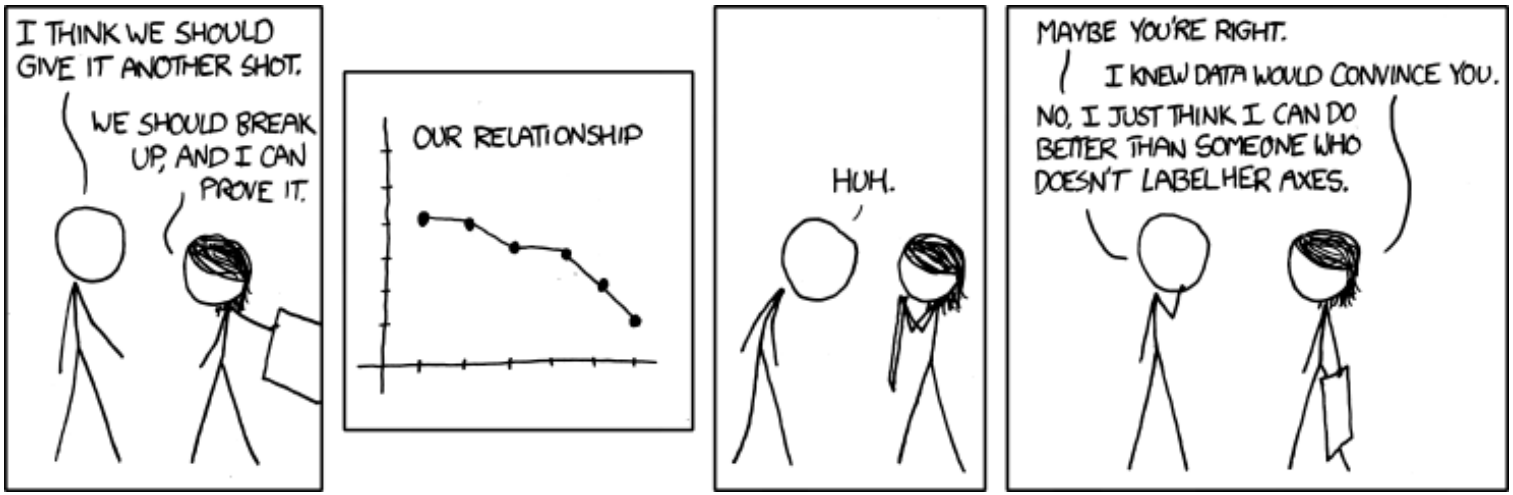


Export von Bananen in Tonnen von 1994-2005

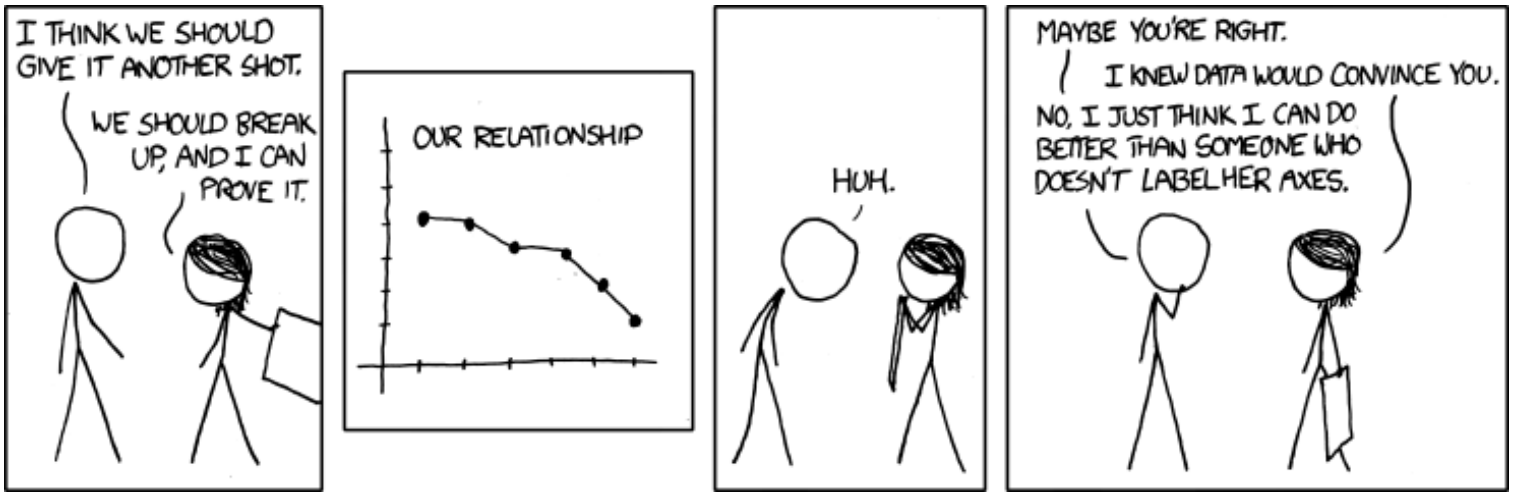


Dr. Hochhaus
Bananenexport 2005

4.1 Label your %#?* axes!



4.1 Label your %#?* axes!



Rant over!

4.2 Qualitative data: bar charts

- The most useful way to display qualitative data is usually with a bar chart.
- The length of each bar is proportional to the frequency of the corresponding value of the variable in the sample of data.
- Note that the widths of the bars should be equal to avoid giving a false impression.

MPAA breakdown

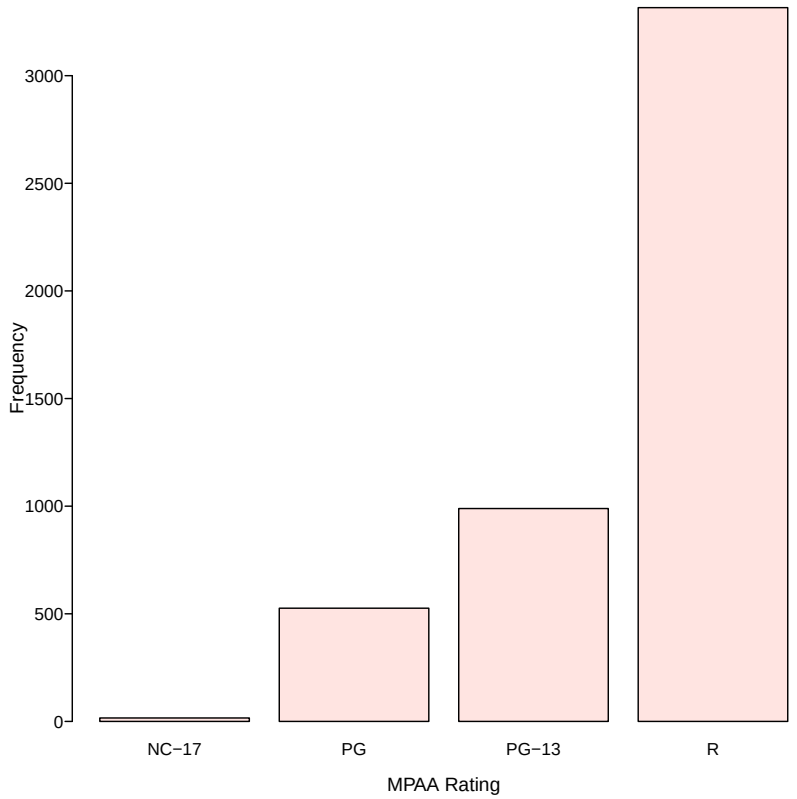


Figure: Barchart of the mpaa ratings for 4847 films.

MPAA breakdown

```
> table(movies$mpaa)
NC-17    PG PG-13     R
   16   526   989  3316
```

MPAA breakdown

```
> table(movies$mpaa)
NC-17    PG PG-13     R
   16   526   989  3316
```

Inside the `barplot` function:

```
> barplot(table(movies$mpaa), xlab="MPAA Rating",
+         ylab="Frequency", border = "black",
+         col="mistyrose")
```

MPAA breakdown

```
> table(movies$mpaa)
NC-17     PG PG-13      R
   16    526    989   3316
```

Inside the `barplot` function:

```
> barplot(table(movies$mpaa), xlab="MPAA Rating",
+         ylab="Frequency", border = "black",
+         col="mistyrose")
```

Remember to load the data first!

```
> library(mas1343)
> data(movies)
```

4.3 Histograms

- To represent the distribution of a sample of values of a continuous variable we can use a histogram.
- The range of values of the variable is divided into intervals, known as *classes*, and the frequencies in classes are represented by columns.
- As the variable is continuous, there are no gaps between neighbouring columns — unlike a bar chart.

4.3 Histograms

- To represent the distribution of a sample of values of a continuous variable we can use a histogram.
- The range of values of the variable is divided into intervals, known as *classes*, and the frequencies in classes are represented by columns.
- As the variable is continuous, there are no gaps between neighbouring columns — unlike a bar chart.
- Note also that, strictly speaking, it is the *area* of the column which is proportional to the frequency, not the height.
- The reason for this is that columns need not be of the same width.
- Computer software tends to use columns of the same width.
- However this default can be overridden in R if you really want to.

4.3 Histograms

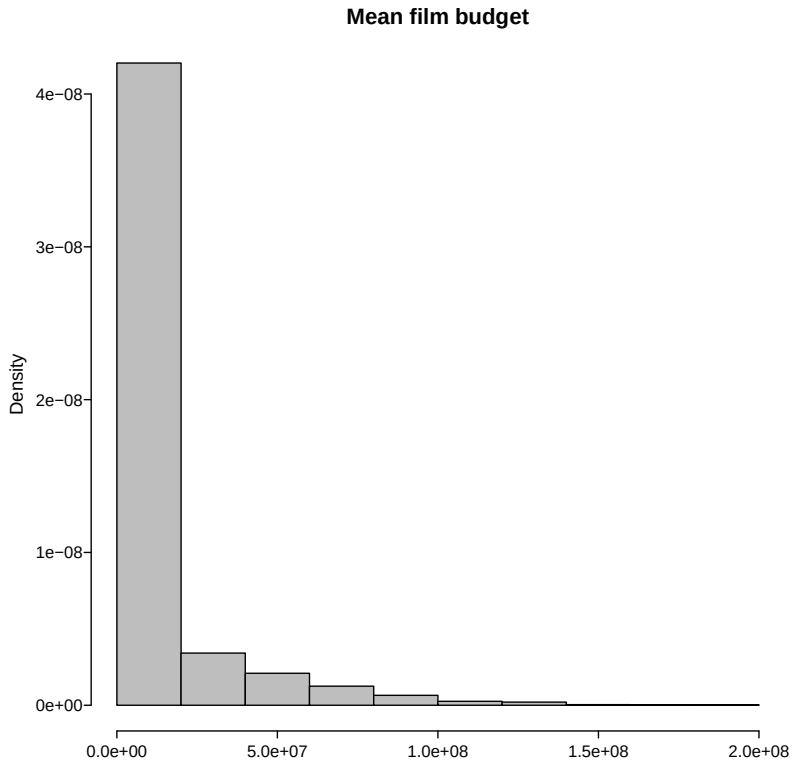


Figure: Histogram of film ratings and budgets.

4.3 Histograms

- Figure 4.2 shows histograms of the film budgets.
- When dealing with densities (relative frequency), we can easily work out the height using this formula:

$$\text{Height} = \frac{\text{frequency}}{n \times \text{ Bin-width}} .$$

When the y-axis is labelled with density or relative frequencies, the area under the histogram is one.

4.3 Histograms

- Figure 4.2 shows histograms of the film budgets.
- When dealing with densities (relative frequency), we can easily work out the height using this formula:

$$\text{Height} = \frac{\text{frequency}}{n \times \text{ Bin-width}} .$$

  When the y-axis is labelled with density or relative frequencies, the area under the histogram is one.
- Bin widths should be chosen so that you get a good idea of the distribution of the data, without being swamped by random variation.

4.3 Histograms

To generate Figure 4.2 in R we use the following commands:

```
> hist(movies$Budget, col="grey",
+      main="Mean film budget", freq=FALSE,
+      xlab="Budget ($)")
```

4.3.1 How many bins should we have?

First we will define the notation we will use:

- n: the sample size;
- k: the number of bins in the histogram;
- h: the bin-width.

Then the number of bins we will use to construct a histogram is:

$$k = \left\lceil \frac{\max(x) - \min(x)}{h} \right\rceil \tag{4.1}$$

where $\lceil \cdot \rceil$ is the ceiling function.

Table 4.1: Sturges' rule

- $k_{ST} = \lceil \log_2 n + 1 \rceil$
- Default
- Tends not to be very good for $n > 30$.

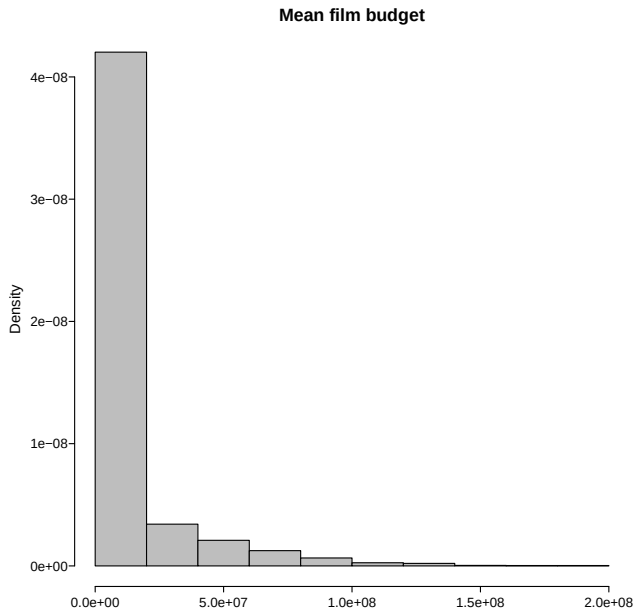


Table 4.1: Scotts' rule

- $h_{SC} = 3.49 \times s \times n^{-1/3}$
- breaks = "Scott"

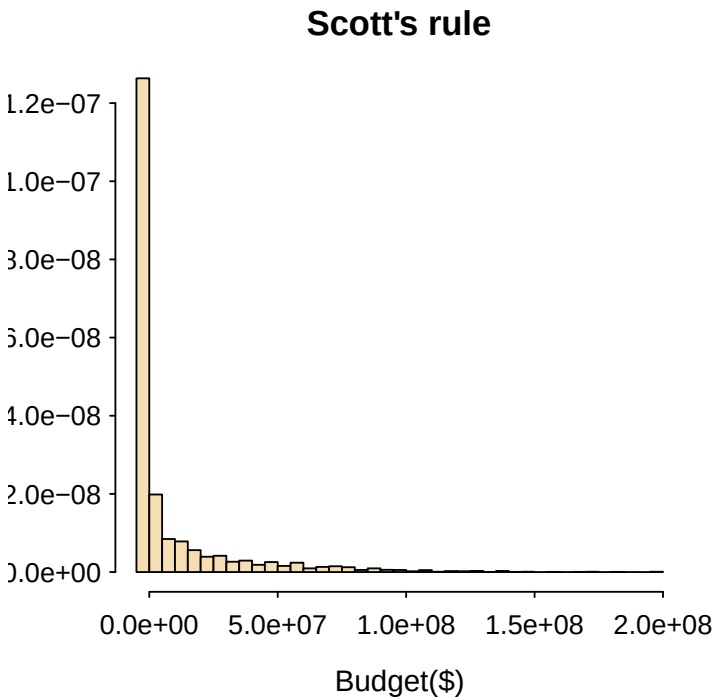
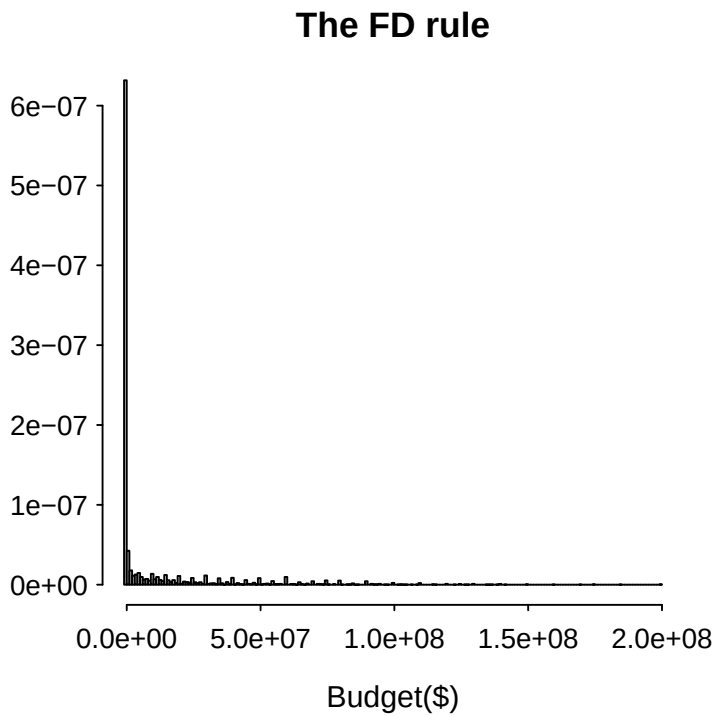


Table 4.1: Freedman-Diaconis

- $h_{FR} = 2 \times IQR(x) \times n^{-1/3}$
- breaks = “FD"
- When the distribution is symmetric, this is very similar to Scott’s rule.



4.3.2 Example: Movie data

Suppose we want to calculate the number of bins for the budget movie variable.

4.3.2 Example: Movie data

Suppose we want to calculate the number of bins for the budget movie variable. For Sturges' rule we have:

4.3.2 Example: Movie data

Suppose we want to calculate the number of bins for the budget movie variable. For Sturges' rule we have:

$$k_{ST} =$$

4.3.2 Example: Movie data

Suppose we want to calculate the number of bins for the budget movie variable. For Sturges' rule we have:

$$k_{ST} = \lceil \log_2(n) \rceil + 1 =$$

4.3.2 Example: Movie data

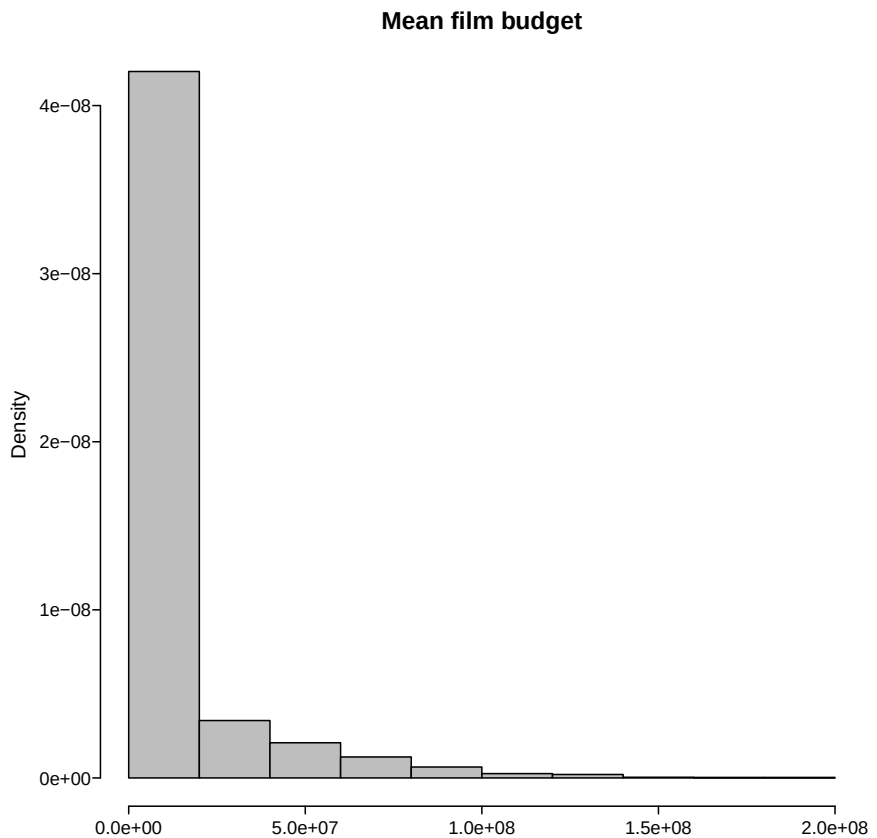
Suppose we want to calculate the number of bins for the budget movie variable. For Sturges' rule we have:

$$k_{ST} = \lceil \log_2(n) \rceil + 1 = \lceil \log_2(4847) \rceil + 1 =$$

4.3.2 Example: Movie data

Suppose we want to calculate the number of bins for the budget movie variable. For Sturges' rule we have:

$$k_{ST} = \lceil \log_2(n) \rceil + 1 = \lceil \log_2(4847) \rceil + 1 = 14 \ .$$



Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} =$$

Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} = \frac{3.49 \times s}{n^{1/3}} =$$

Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} = \frac{3.49 \times s}{n^{1/3}} = \frac{3.49 \times 23039711}{4847^{1/3}} \simeq$$

Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} = \frac{3.49 \times s}{n^{1/3}} = \frac{3.49 \times 23039711}{4847^{1/3}} \simeq 4,751,289$$

Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} = \frac{3.49 \times s}{n^{1/3}} = \frac{3.49 \times 23039711}{4847^{1/3}} \simeq 4,751,289$$

then the number of bins is:

Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} = \frac{3.49 \times s}{n^{1/3}} = \frac{3.49 \times 23039711}{4847^{1/3}} \simeq 4,751,289$$

then the number of bins is:

$$k_{SC} =$$

Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} = \frac{3.49 \times s}{n^{1/3}} = \frac{3.49 \times 23039711}{4847^{1/3}} \simeq 4,751,289$$

then the number of bins is:

$$k_{SC} = \left\lceil \frac{\max(x) - \min(x)}{h_{sc}} \right\rceil =$$

Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} = \frac{3.49 \times s}{n^{1/3}} = \frac{3.49 \times 23039711}{4847^{1/3}} \simeq 4,751,289$$

then the number of bins is:

$$k_{SC} = \left\lceil \frac{\max(x) - \min(x)}{h_{sc}} \right\rceil = \left\lceil \frac{2 \times 10^8 - (-1)}{4751289} \right\rceil =$$

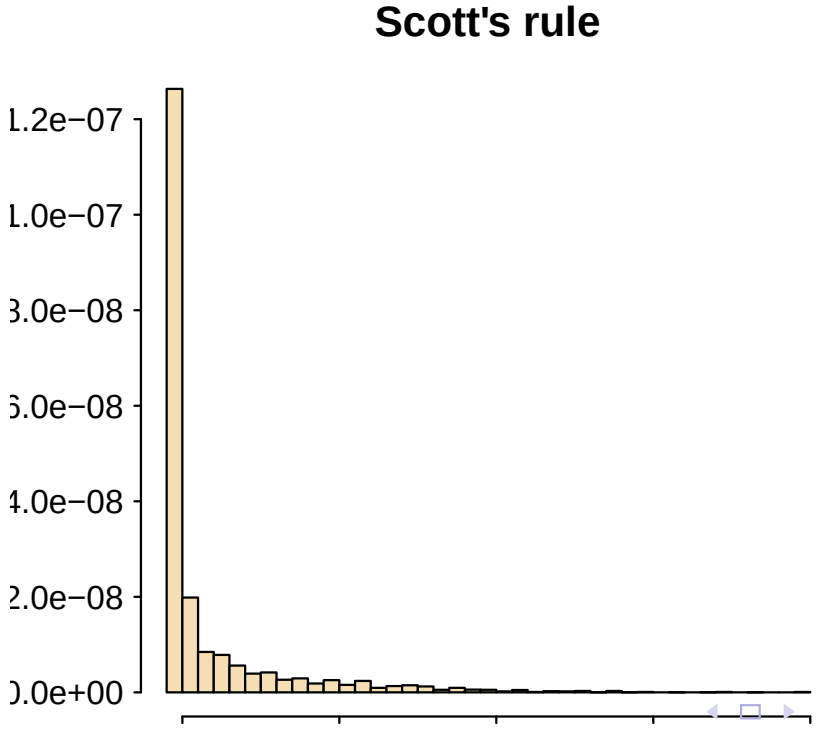
Movie data: Scotts' rule

For Scotts' rule, we first calculate the bin width

$$h_{SC} = \frac{3.49 \times s}{n^{1/3}} = \frac{3.49 \times 23039711}{4847^{1/3}} \simeq 4,751,289$$

then the number of bins is:

$$k_{SC} = \left\lceil \frac{\max(x) - \min(x)}{h_{sc}} \right\rceil = \left\lceil \frac{2 \times 10^8 - (-1)}{4751289} \right\rceil = 43 \ .$$



Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} =$$

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} = \frac{2 \times IQR(x)}{n^{1/3}} =$$

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} = \frac{2 \times IQR(x)}{n^{1/3}} = \frac{2 \times (8 \times 10^6 - (-1))}{4847^{1/3}} \simeq$$

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} = \frac{2 \times IQR(x)}{n^{1/3}} = \frac{2 \times (8 \times 10^6 - (-1))}{4847^{1/3}} \simeq 945,429 \ .$$

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} = \frac{2 \times IQR(x)}{n^{1/3}} = \frac{2 \times (8 \times 10^6 - (-1))}{4847^{1/3}} \simeq 945,429 \ .$$

Hence,

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} = \frac{2 \times IQR(x)}{n^{1/3}} = \frac{2 \times (8 \times 10^6 - (-1))}{4847^{1/3}} \simeq 945,429 \ .$$

Hence,

$$k_{FD} =$$

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} = \frac{2 \times IQR(x)}{n^{1/3}} = \frac{2 \times (8 \times 10^6 - (-1))}{4847^{1/3}} \simeq 945,429 \ .$$

Hence,

$$k_{FD} = \left\lceil \frac{\max(x) - \min(x)}{h_{sc}} \right\rceil$$

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} = \frac{2 \times IQR(x)}{n^{1/3}} = \frac{2 \times (8 \times 10^6 - (-1))}{4847^{1/3}} \simeq 945,429 \ .$$

Hence,

$$k_{FD} = \left\lceil \frac{\max(x) - \min(x)}{h_{sc}} \right\rceil = \left\lceil \frac{2 \times 10^8 - (-1)}{945429} \right\rceil =$$

Movie data: Freedman-Diaconis rule

For the FD rule, the bin width is:

$$h_{FD} = \frac{2 \times IQR(x)}{n^{1/3}} = \frac{2 \times (8 \times 10^6 - (-1))}{4847^{1/3}} \simeq 945,429 \ .$$

Hence,

$$k_{FD} = \left\lceil \frac{\max(x) - \min(x)}{h_{sc}} \right\rceil = \left\lceil \frac{2 \times 10^8 - (-1)}{945429} \right\rceil = 212 \ .$$

4.4 Box and whisker plots

- A box and whisker plot, sometimes simply called a ‘boxplot’, is another way to represent continuous data.
- This kind of plot is particularly useful for comparing two or more groups, by placing the boxplots side-by-side.

4.4 Box and whisker plots

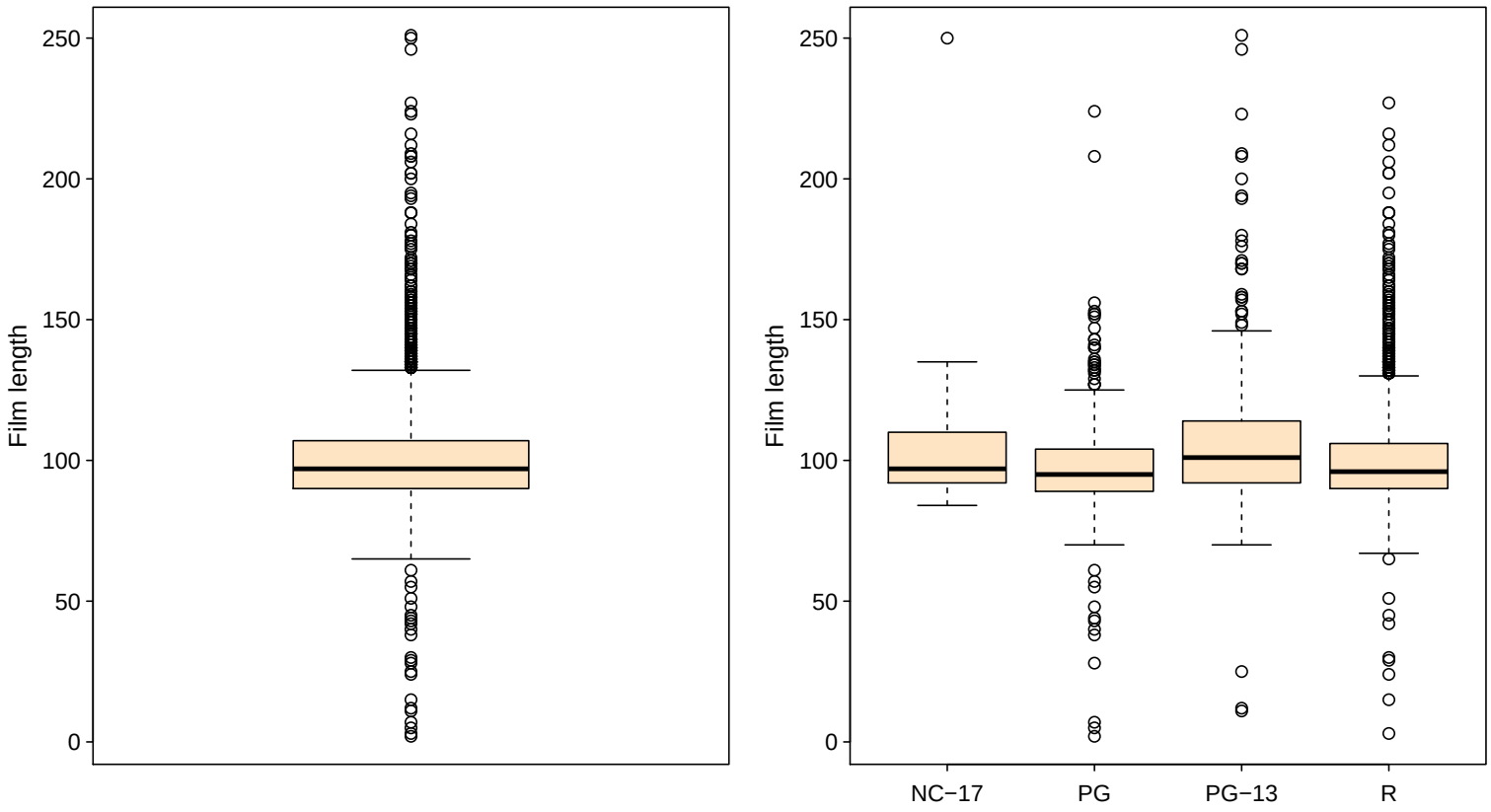


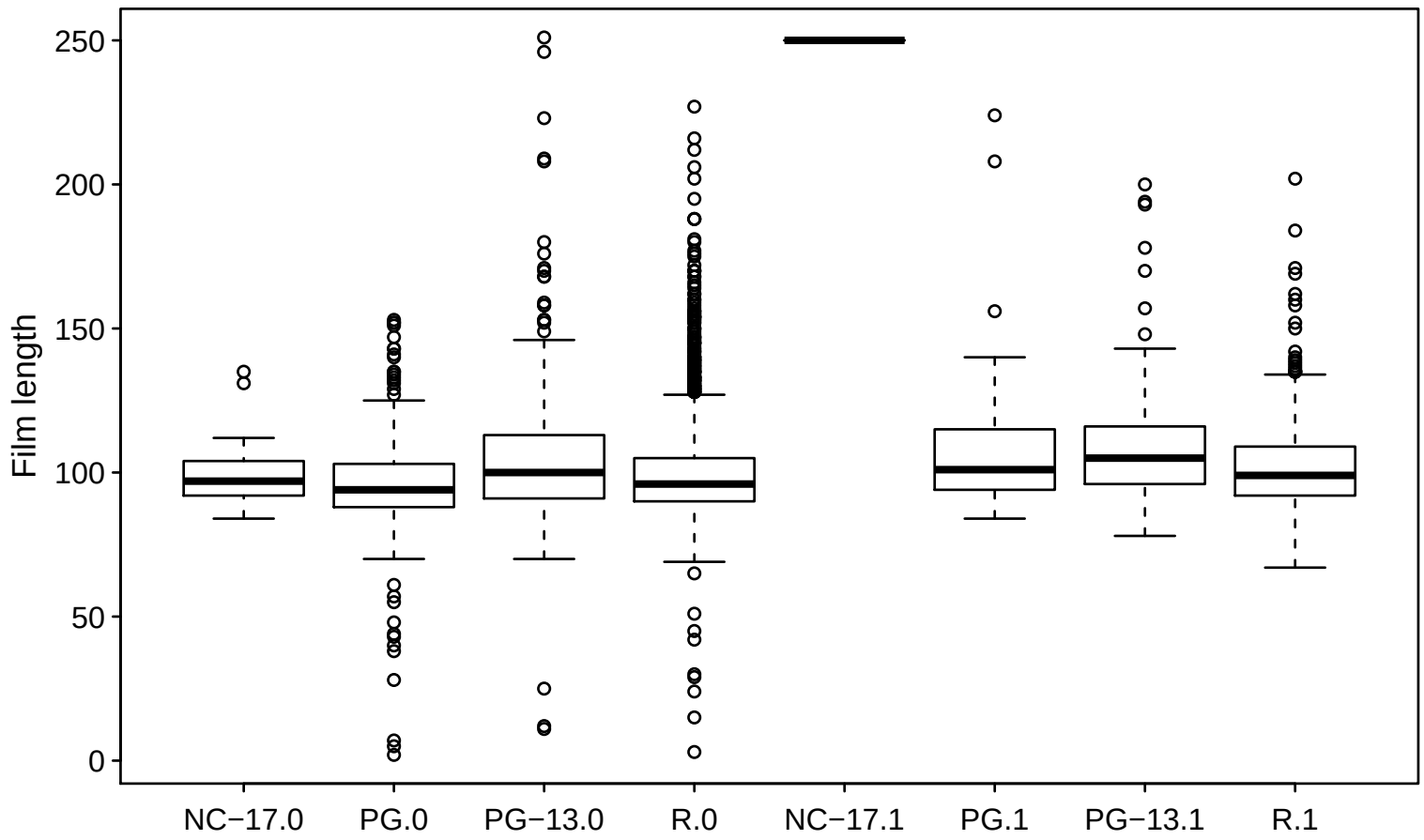
Figure: Box and whisker plots of (a) film length (b) film length split according to the mpaa rating.

4.4 Box and whisker plots

To do this in R we use the following commands:

```
> par(mfrow=c(1, 2))
> boxplot(movies$Length, ylab="Film length",
+         col="bisque")
> boxplot(movies$Length ~ movies$mpaa,
+         ylab="Film length",
+         col="bisque")
```

4.4 Box and whisker plots




```
> boxplot(movies$Length ~ movies$mpaa + movies$Romance)
```

4.4.1 Boxplot example 1

For the data set

0.1	0.1	0.2	0.4	0.4	0.8	0.8	0.8
0.9	0.9	1.0	1.4	1.6	2.0	2.4	3.5

Table: An example data set.

construct a boxplot.

4.4.1 Solution

First we calculate the median and quartiles

Median	1^{st} quartile	3^{rd} quartile	IQR
0.85	0.4	1.55	1.15

Table: Summary statistics for the first example data in Table 4.3.

Solution

To calculate the outliers and whiskers, we first calculate:

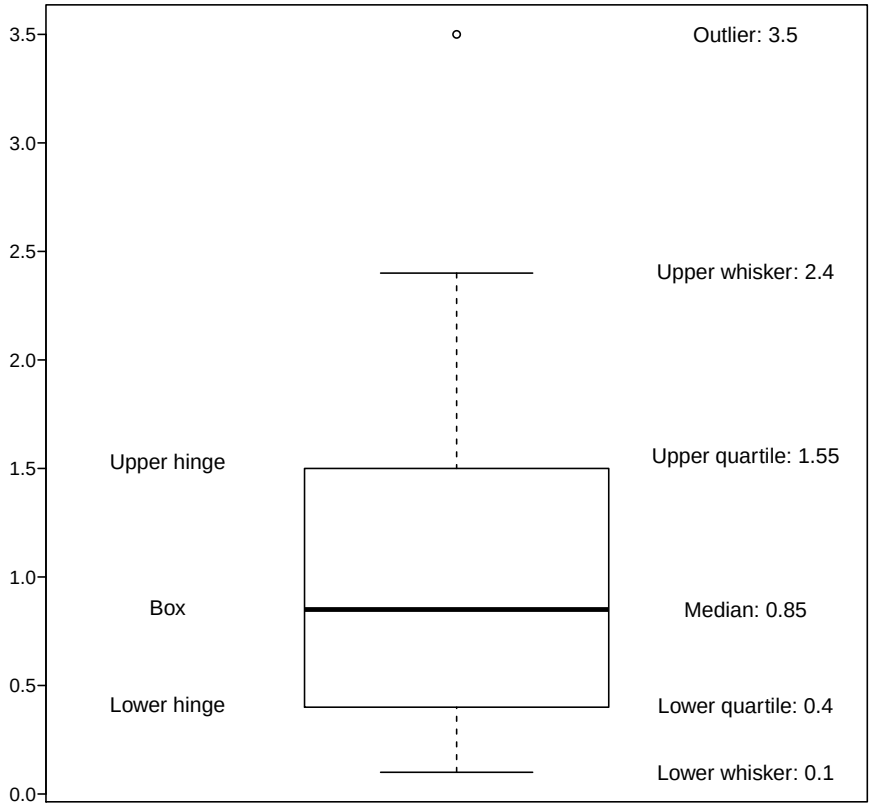
$$W_L = \text{lower quartile} - 1.5IQR = -1.325$$

and

$$W_U = \text{upper quartile} + 1.5IQR = 3.275$$

Since $3.5 > W_U = 3.275$, this means that 3.5 is an outlying point. Since we have no points less than W_L, the lower whisker is the smallest data point, i.e. 0.1. The upper whisker is $\max(x^*)$ where x^* does not include any outlying points, i.e. 2.4.

4.4.1 Boxplot example 1



4.4.2 Boxplot example 2

For the following data set, construct a boxplot

9.0	32.8	33.0	34.9	35.4	39.7	41.6	42.0
42.3	43.2	46.9	49.2	51.6	51.7	55.0	81.0

Table: An example data set.

4.4.2 Boxplot example 2

Median	1^{st} quartile	3^{rd} quartile	IQR	W_L	W_U
42.15	35.025	51.00	15.975		

Table: Summary statistics for the second example data in table 4.5

4.4.2 Boxplot example 2

Median	1^{st} quartile	3^{rd} quartile	IQR	W_L	W_U
42.15	35.025	51.00	15.975		

Table: Summary statistics for the second example data in table 4.5

$$W_L \quad =$$

4.4.2 Boxplot example 2

Median	1^{st} quartile	3^{rd} quartile	IQR	W_L	W_U
42.15	35.025	51.00	15.975		

Table: Summary statistics for the second example data in table 4.5

$$W_L = \text{lower quartile} - 1.5IQR =$$

4.4.2 Boxplot example 2

Median	1^{st} quartile	3^{rd} quartile	IQR	W_L	W_U
42.15	35.025	51.00	15.975	11.0625	

Table: Summary statistics for the second example data in table 4.5

$$W_L = \text{lower quartile} - 1.5IQR = 11.0625$$

4.4.2 Boxplot example 2

Median	1^{st} quartile	3^{rd} quartile	IQR	W_L	W_U
42.15	35.025	51.00	15.975	11.0625	

Table: Summary statistics for the second example data in table 4.5

$$W_L = \text{lower quartile} - 1.5IQR = 11.0625$$
$$W_U =$$

4.4.2 Boxplot example 2

Median	1^{st} quartile	3^{rd} quartile	IQR	W_L	W_U
42.15	35.025	51.00	15.975	11.0625	

Table: Summary statistics for the second example data in table 4.5

$$W_L = \text{lower quartile} - 1.5IQR = 11.0625$$
$$W_U = \text{Upper quartile} + 1.5IQR =$$

4.4.2 Boxplot example 2

Median	1^{st} quartile	3^{rd} quartile	IQR	W_L	W_U
42.15	35.025	51.00	15.975	11.0625	74.9625

Table: Summary statistics for the second example data in table 4.5

$$
\begin{aligned}
W_L &= \text{lower quartile} - 1.5IQR = 11.0625 \\
W_U &= \text{Upper quartile} + 1.5IQR = 74.9625
\end{aligned}
$$

4.4.2 Boxplot example 2

9.0	32.8	33.0	34.9	35.4	39.7	41.6	42.0
42.3	43.2	46.9	49.2	51.6	51.7	55.0	81.0

- From table 4.5, we see that $9.0 < W_L = 11.0625$.

4.4.2 Boxplot example 2

9.0	32.8	33.0	34.9	35.4	39.7	41.6	42.0
42.3	43.2	46.9	49.2	51.6	51.7	55.0	81.0

- From table 4.5, we see that $9.0 < W_L = 11.0625$. So the lower whisker extends to the point 32.8 and 9.0 is an outlier.

4.4.2 Boxplot example 2

9.0	32.8	33.0	34.9	35.4	39.7	41.6	42.0
42.3	43.2	46.9	49.2	51.6	51.7	55.0	81.0

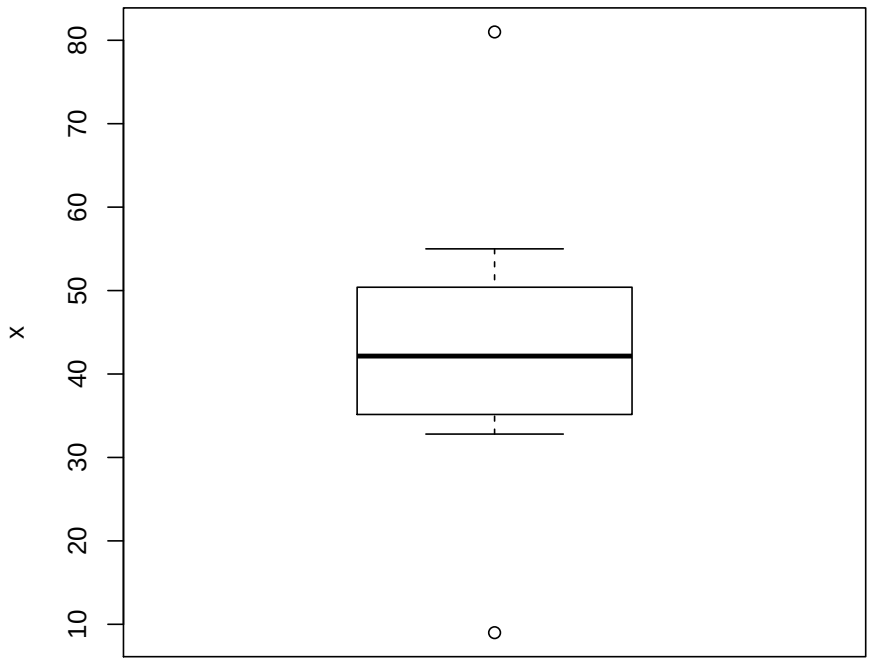
- From table 4.5, we see that $9.0 < W_L = 11.0625$. So the lower whisker extends to the point 32.8 and 9.0 is an outlier.
- From table 4.5, we see that $81 > W_U = 74.9625$.

4.4.2 Boxplot example 2

9.0	32.8	33.0	34.9	35.4	39.7	41.6	42.0
42.3	43.2	46.9	49.2	51.6	51.7	55.0	81.0

- From table 4.5, we see that $9.0 < W_L = 11.0625$. So the lower whisker extends to the point 32.8 and 9.0 is an outlier.
- From table 4.5, we see that $81 > W_U = 74.9625$. So the upper whisker extends to the point 55.0 and 81 is an outlier.

4.4.2 Boxplot example 2



Part V

Control statements and functions

5.1.1 Basic functions

This function takes in a single argument x and returns x^2:

```
> Fun1 = function(x) {
+   return (x*x)
+ }
```

5.1.1 Basic functions

The key elements in the function call are:

- The word `function`;

5.1.1 Basic functions

The key elements in the function call are:

- The word `function`;
- The brackets $()$ which enclose the **argument** list. This list may be empty.

5.1.1 Basic functions

The key elements in the function call are:

- The word `function`;
- The brackets $()$ which enclose the **argument** list. This list may be empty.
- A sequence of statements in curly braces $\{\ \}$.

5.1.1 Basic functions

The key elements in the function call are:

- The word `function`;
- The brackets $()$ which enclose the **argument** list. This list may be empty.
- A sequence of statements in curly braces $\{\ \}$.
- A `return` statement.

5.1.1 Basic functions

We ‘call’ Fun1 in the following manner:

```
> Fun1(5)
[1] 25
```

5.1.1 Basic functions

We ‘call’ Fun1 in the following manner:

```
> Fun1(5)
[1] 25
```

```
> y = Fun1(10)
```

5.1.1 Basic functions

We ‘call’ Fun1 in the following manner:

```
> Fun1(5)
[1] 25
```

```
> y = Fun1(10)
```

```
> y
[1] 100
```

5.1.1 Basic functions

We ‘call’ Fun1 in the following manner:

```
> Fun1(5)
[1] 25
```

```
> y = Fun1(10)
```

```
> y
[1] 100
```

```
> z = c(1, 2, 3, 4)
> Fun1(z)
[1]  1  4  9 16
```

5.1.1 Basic functions

Of course, the old saying ‘Garbage in, Garbage out’ is true:

```
#Incorrect function calls.
> Fun1()
Error in Fun1() : argument "x" is missing, with no default
> Fun1("5")
Error in x * x : non-numeric argument to binary operator
```

The error messages do give you an idea of what went wrong.

Question 1

What is the value of y?

```
> fun1 = function(x) {
+          return(2*x)
+ }
> y = fun1(2)
```

- 5
- 10
- 2
- 4

Question 1

What is the value of y?

```
> x = 5
> fun1 = function(x) {
+          return(2*x)
+ }
> (y= fun1(2))
[1] 4
```

- 5
- 10
- 2
- 4

Question 2

What is the value of y?

```
> fun1 = function(x) {
+    z = 3
+    return(x*z)
+ }
> y = fun1(2)
```

- 2
- 3
- 4
- 6

Question 2

What is the value of y?

```
> fun1 = function(x) {
+   z = 3
+   return(x*z)
+ }
> (y = fun1(2))
[1] 6
```

- 2
- 3
- 4
- 6

Question 3

What is the value of y?

```
> fun1 = function(x) {
+    z = 3
+    return(x*z)
+ }
> k = 5
> y = fun1(k)
```

Question 3

What is the value of y?

```
> fun1 = function(x) {
+   z = 3
+   return(x*z)
+ }
> k = 5
> (y = fun1(k))
[1] 15
```

5.1.1 Basic functions

Other variations to this simple function are:

```
> Fun2 = function(x=1) {
+   return (x*x)
+ }
```

5.1.1 Basic functions

Other variations to this simple function are:

```
> Fun2 = function(x=1) {
+   return (x*x)
+ }
```

```
> Fun2()
[1] 1
```

5.1.1 Basic functions

Other variations to this simple function are:

```
> Fun2 = function(x=1) {
+   return (x*x)
+ }
```

```
> Fun2()
[1] 1
```

```
> Fun2(4)
[1] 16
```

5.1.1 Basic functions

```
> Fun3 = function(x, y) {
+   return (x*y)
+ }
```

5.1.1 Basic functions

```
> Fun3 = function(x, y) {
+    return (x*y)
+ }
```

```
> Fun3(3, 4)
[1] 12
```

Question 4

What is the value of y?

```
> fun1 = function(x=3) {
+          return(2*x)
+ }
> y = fun1(2)
```

- 3
- 4
- 6

Question 4

What is the value of y?

```
> fun1 = function(x=3) {
+          return(2*x)
+ }
> (y = fun1(2))
[1] 4
```

- 3
- 4
- 6

Question 5

What is the value of y?

```
> fun1 = function(x=3) {
+   return(2*x)
+ }
> y = fun1()
```

- 3
- 4
- 6

Question 5

What is the value of y?

```
> fun1 = function(x=3) {
+   return(2*x)
+ }
> (y = fun1())
[1] 6
```

- 3
- 4
- 6

Question 6

What is the value of y?

```
> fun1 = function(x=3) {
+    x = 2
+    return(2*x)
+ }
> y = fun1()
```

- 3
- 4
- 6

Question 6

What is the value of y?

```
> fun1 = function(x=3) {
+   x = 2
+   return(2*x)
+ }
> (y = fun1())
[1] 4
```

- 3
- 4
- 6

5.1.2 A more useful function

Here the function below takes in a vector, plots a histogram and returns a vector containing the mean and standard deviation:

```
> Investigate = function(values) {
+   hist(values)
+   m_std = c(mean(values), sd(values))
+   return(m_std)
+ }
```

5.1.2 A more useful function

Once we have created our function, we can put it to good use:

```
> Investigate(movies$Rating)
[1] 5.522715 1.451864
```

5.1.2 A more useful function

Once we have created our function, we can put it to good use:

```
> Investigate(movies$Rating)
[1] 5.522715 1.451864
```

```
> Investigate(movies$Length)
[1] 100.87807  17.34152
```

5.1.2 A more useful function

Once we have created our function, we can put it to good use:

```
> Investigate(movies$Rating)
[1] 5.522715 1.451864
```

```
> Investigate(movies$Length)
[1] 100.87807  17.34152
```

```
> Investigate(movies$Budget)
[1] 10286893 23039711
```

5.1.3 Variable scope

5.1.3 Variable scope

- When we call a function, R first looks for *local* variables, then *global* variables.
- For example, Fun4 uses a global variable:

```
> blob = 5
> Fun4 = function() {
+   return(blob)
+ }
```

5.1.3 Variable scope

- When we call a function, R first looks for *local* variables, then *global* variables.
- For example, Fun4 uses a global variable:

```
> blob = 5
> Fun4 = function() {
+   return(blob)
+ }
```

```
> Fun4()
[1] 5
```

5.1.3 Variable scope

But Fun5 uses a local variable:

```
> blob = 5
> Fun5 = function() {
+   blob = 6
+   return(blob)
+ }
```

5.1.3 Variable scope

But Fun5 uses a local variable:

```
> blob = 5
> Fun5 = function() {
+   blob = 6
+   return(blob)
+ }
```

```
> Fun5()
[1] 6
```

5.1.3 Variable scope

But Fun5 uses a local variable:

```
> blob = 5
> Fun5 = function() {
+   blob = 6
+   return(blob)
+ }
```

```
> Fun5()
[1] 6
```

```
> blob
[1] 5
```

Question 7

What is the value of y?

```
> x = 1
> fun1 = function(x) {
+   x = 2
+   return(x)
+ }
> y= fun1(3)
```

- 1
- 2
- 3

Question 7

What is the value of y?

```
> x = 1
> fun1 = function(x) {
+   x = 2
+   return(x)
+ }
> (y= fun1(3))
[1] 2
```

- 1
- 2
- 3

Question 8

What is the value of y?

```
> x = 1
> fun1 = function(x=3) {
+    x = 2
+    return(x)
+ }
> y= fun1()
```

- 1
- 2
- 3

Question 8

What is the value of y?

```
> x = 1
> fun1 = function(x=3) {
+   x = 2
+   return(x)
+ }
> (y= fun1())
[1] 2
```

- 1
- 2
- 3

Question 9

What is the value of y?

```
> x = 1
> fun1 = function(x=3) {
+   return(2*x)
+ }
> x = fun1(x)
> y = fun1(x)
```

Question 9

What is the value of y?

```
> x = 1
> fun1 = function(x=3) {
+   return(2*x)
+ }
> x = fun1(x)
> (y = fun1(x))
[1] 4
```

5.2 The `cat` command

- A useful function to help debugging is the `cat` function.
- This function is used to print messages to the screen.

5.2 The cat command

- A useful function to help debugging is the cat function.
- This function is used to print messages to the screen.
- For example,

```
> x = 5
> cat(x, "\n")
5
> (y = cat(x, "\n"))
5
NULL
```

- We will use the cat function in the next section.

5.3 Conditionals

- Conditional statements are features of a programming language which perform different computations or actions depending on whether a condition evaluates to TRUE or FALSE.
- They are used in almost all computer programs.

5.3.1 If statements

The basic structure of an `if` statement is:

```
> if(expr) {
+   ##do something
+ }
```

where `expr` is evaluated to be either `TRUE` or `FALSE`.

5.3.1 `If` statements

The following example illustrates `if` statements in R:

```
> x = 5
> y = 5
> if(x<5) {
+   y = 0
+ }
```

5.3.1 `If` statements

The following example illustrates `if` statements in R:

```
> x = 5
> y = 5
> if(x<5) {
+   y = 0
+ }
```

```
> y
[1] 5
```

In this code chunk, x $<$ 5 evaluates to be FALSE so the following brackets are not evaluated.

5.3.1 If statements

We test for greater than in a similar manner:

```
> x = 5
> y = 5
> if(x > 0) {
+   y = 0
+ }
> y
[1] 0
```

Here x $>$ 0 evaluates to be TRUE so, y is set equal to 0. If we wanted to test for equality with zero, then we would use $=>$.

5.3.2 If else statements

We can link together a number of if statements

```
> x = 0
> if(x > 0) {
+   cat("x is greater than zero")
+ } else if(x < 0) {
+   cat("x is less than zero")
+ } else {
+   cat("x must be zero!")
+   cat("\n")
+ }
x must be zero!
```

The final else is optional.

5.3.2 If else statements

```
> IsNegative = function(value) {
+   is_pos = FALSE
+   if(value < 0) {
+     is_pos = TRUE
+   }
+   return(is_pos)
+ }
```

5.3.2 If else statements

```
> IsNegative = function(value) {
+   is_pos = FALSE
+   if(value < 0) {
+     is_pos = TRUE
+   }
+   return(is_pos)
+ }
```

```
> IsNegative(1)
[1] FALSE
```

5.3.2 If else statements

```
> IsNegative = function(value) {
+   is_pos = FALSE
+   if(value < 0) {
+     is_pos = TRUE
+   }
+   return(is_pos)
+ }
```

```
> IsNegative(1)
[1] FALSE
```

```
> IsNegative(-5.6)
[1] TRUE
```

5.3.2 If else statements

A more sophisticated function could be

```
> IsGreaterThan = function(value1, value2) {
+   is_greater_than = FALSE
+   if(value1 > value2) {
+     is_greater_than = TRUE
+   }
+   return(is_greater_than)
+ }
```

5.3.2 If else statements

```
> IsGreaterThan(-5, -6)
[1] TRUE
```

5.3.2 If else statements

```
> IsGreaterThan(-5, -6)
[1] TRUE
```

```
> IsGreaterThan(10, 10)
[1] FALSE
```

Question 1

What is the value of y?

```
> x = 5
> if(x==6)  {
+    y = FALSE
+ } else {
+    y = TRUE
+ }
```

- TRUE
- FALSE

Question 1

What is the value of y?

```
> x = 5
> if(x==6)  {
+    y = FALSE
+ } else {
+    y = TRUE
+ }
> y
[1] TRUE
```

- TRUE
- FALSE

Question 2

What is the value of y?

```
> x = 5
> if(x > 6){
+    y = 0
+ } else if(x >= 5) {
+    y = 1
+ } else {
+ y = 2
+ }
```

- 0
- 1
- 2

Question 2

What is the value of y?

```
> x = 5
> if(x > 6){
+    y = 0
+ }else if(x >= 5) {
+    y = 1
+ } else {
+   y = 2
+ }
> y
[1] 1
```

- 0
- 1
- 2

Question 3

What is the value of z?

```
> x = 5
> y = "male"
> if(x > 6 & y == "Female"){
+   z = 0
+ }else if(x < 5 & y == "Male") {
+   z = 1
+ } else {
+   z = 2
+ }
```

- 0
- 1
- 2

Question 3

What is the value of z?

```
> x = 5
> y = "male"
> if(x > 6 & y == "Female"){
+   z = 0
+ }else if(x < 5 & y == "Male") {
+   z = 1
+ } else {
+   z = 2
+ }
> z
[1] 2
```

- 0
- 1
- 2

5.4 Control statements

- At times we would like to perform some operation on a vector or a data frame.
- Often R has built-in functions that will do this for you, e.g. `mean`, `sd`, other times we have to write our own functions.
- For example, suppose we want calculate $\sum_{i=1}^{10} i^2$.
- In R we can use a `for` loop:

```
> x = 0
> for(i in 1:10) {
+   x = x + i^2
+ }
```

5.4 Control statements

- At times we would like to perform some operation on a vector or a data frame.
- Often R has built-in functions that will do this for you, e.g. `mean`, `sd`, other times we have to write our own functions.
- For example, suppose we want calculate $\sum_{i=1}^{10} i^2$.
- In R we can use a `for` loop:

```
> x = 0
> for(i in 1:10) {
+   x = x + i^2
+ }
```

```
> x
[1] 385
```

5.4 Control statements

$\sum_{j=-5}^{-1} e^j / j^2$,

```
> total = 0
> for(j in -5:-1) {
+   total = total + exp(j)/j^2
+ }
```

5.4 Control statements

$\sum_{j=-5}^{-1} e^j / j^2$,

```
> total = 0
> for(j in -5:-1) {
+   total = total + exp(j)/j^2
+ }
```

```
> total
[1] 0.4086594
```

5.4 Control statements

A more tricky example is, calculate $\sum e^k / k^2$, for $k = 3, 6, 9, \ldots, 21$:

```
> total = 0
> for(i in 1:7) {
+   k = i*3
+   total = total + exp(k)/k^2
+ }
> total
[1] 3208939
```

Question 4

What it the value of x?

```
> total = 0
> for(blob in 1:4) {
+   total = total + 1
+ }
> x = total
```

Question 4

What it the value of x?

```
> total = 0
> for(blob in 1:4) {
+   total = total + 1
+ }
> (x = total)
[1] 4
```

Question 5

What it the value of x?

```
> total = 0
> for(blob in 1:10) {
+   if(blob > 9) {
+     total = total + 1
+   }
+ }
> x = total
```

Question 5

What it the value of x?

```
> total = 0
> for(blob in 1:10) {
+   if(blob > 9) {
+     total = total + 1
+   }
+ }
> (x = total)
[1] 1
```

5.5 Putting it all together

Rather than have to constantly write R code to solve the summations in §5.4 we can create a function to solve the general form:

$$\sum_{i=i_s}^{i_e} \frac{e^i}{i^2} \quad \text{for } i = i_s,\ i_s + j,\ i_s + 2j,\ \ldots,\ i_e\ .$$

5.5 Putting it all together

So in R we have

```
> Summation1 = function(i_s, i_e, j) {
+   total = 0
+   for(i in i_s:(i_e/j)) {
+     k = i*j
+     total = total + exp(k)/k^2
+   }
+   return(total)
+ }
```

5.5 Putting it all together

So in R we have

```
> Summation1 = function(i_s, i_e, j) {
+   total = 0
+   for(i in i_s:(i_e/j)) {
+     k = i*j
+     total = total + exp(k)/k^2
+   }
+   return(total)
+ }
```

```
> Summation1(-5, -1, 1)
[1] 0.4086594
```

5.5 Putting it all together

So in R we have

```
> Summation1 = function(i_s, i_e, j) {
+   total = 0
+   for(i in i_s:(i_e/j)) {
+     k = i*j
+     total = total + exp(k)/k^2
+   }
+   return(total)
+ }
```

```
> Summation1(-5, -1, 1)
[1] 0.4086594
```

```
> Summation1(3, 21, 3)
[1] 3208925
```

5.6.1 The apply function

- We use the apply function when we want to *apply* the same function to every row or column of a data frame.
- For example, suppose we have a data frame with three columns:

```
> (df4 = data.frame(c1 = 1:4, c2 = 4:7, c3 = 2:5))
  c1 c2 c3
1  1  4  2
2  2  5  3
3  3  6  4
4  4  7  5
```

5.6.1 The apply function

- The apply function takes (at least) three arguments.
- The first argument is the data frame, the second the number 1 or 2 indicating row or column and the third a function to apply to each row or column.
- So

```
> apply(df4, 1, mean)
[1] 2.333333 3.333333 4.333333 5.333333
```

calculates the mean value of every row,

Standard deviation of every column

```
> apply(df4, 2, sd)
       c1        c2        c3
1.290994 1.290994 1.290994
```

Suppose one of the columns was non-numeric

```
> (df5 = data.frame(c1 = 1:3, c2 = 4:6,
+     c3 = LETTERS[1:3]))
  c1 c2 c3
1  1  4  A
2  2  5  B
3  3  6  C
```

then taking the mean doesn't really make sense:

```
> apply(df5, 1, mean)
[1] NA NA NA
```

Suppose one of the columns was non-numeric

```
> (df5 = data.frame(c1 = 1:3, c2 = 4:6,
+     c3 = LETTERS[1:3]))
  c1 c2 c3
1  1  4  A
2  2  5  B
3  3  6  C
```

then taking the mean doesn't really make sense:

```
> apply(df5, 1, mean)
[1] NA NA NA
```

Instead, we remove the column, then calculate the mean:

```
> apply(df5[ ,1:2], 1, mean)
[1] 2.5 3.5 4.5
```

Question 6

What type of variable is z?

```
> apply(z, 1, mean)
```

- Data frame
- Double
- String
- logical

Question 6

What type of variable is z?

```
> apply(z, 1, mean)
```

- Data frame
- Double
- String
- logical

Question 7

Which of these statements is correct?

```
> apply(z, 2, median)
```

- We calculate the median of z
- We calculate the median for each row of z
- We calculate the median for each column of z
- We calculate the mean of z

Question 2

Which of these statements is correct?

```
> apply(z, 2, median)
```

- We calculate the median of z
- We calculate the median for each row of z
- We calculate the median for each column of z
- We calculate the mean of z

5.6.2 The `tapply` function

- The function `tapply` is very useful, but at first glance can be tricky to understand.
- It's best described using an example:

```
> tapply(movies$Length, movies$mpaa, mean)
    NC-17         PG      PG-13          R
110.18750  97.23384 104.97877 100.18818
```

5.6.2 The `tapply` function

- In the above code, we have calculated the average movie length conditional on its MPAA rating.
- So the average length of a PG movie is 97 minutes and the average NC-17 movie length is 110mins.
- With `tapply` we can do we very interesting things.

5.6.2 The `tapply` function

- For example, in the next piece of code, we plot the average movie rating conditional on its length:

```
> tapply(movies$Length, movies$Rating, mean)[1:6]
   1  1.2  1.3  1.4  1.5  1.6
85.5 93.0 87.5 85.0 67.0 86.0
```

5.6.2 The `tapply` function

- For example, in the next piece of code, we plot the average movie rating conditional on its length:

```
> tapply(movies$Length, movies$Rating, mean)[1:6]
   1  1.2  1.3  1.4  1.5  1.6
85.5 93.0 87.5 85.0 67.0 86.0
```

```
> rating_by_len = tapply(movies$Length, movies$Rating,
+   mean)
> plot(names(rating_by_len), rating_by_len)
```

5.6.2 The `tapply` function

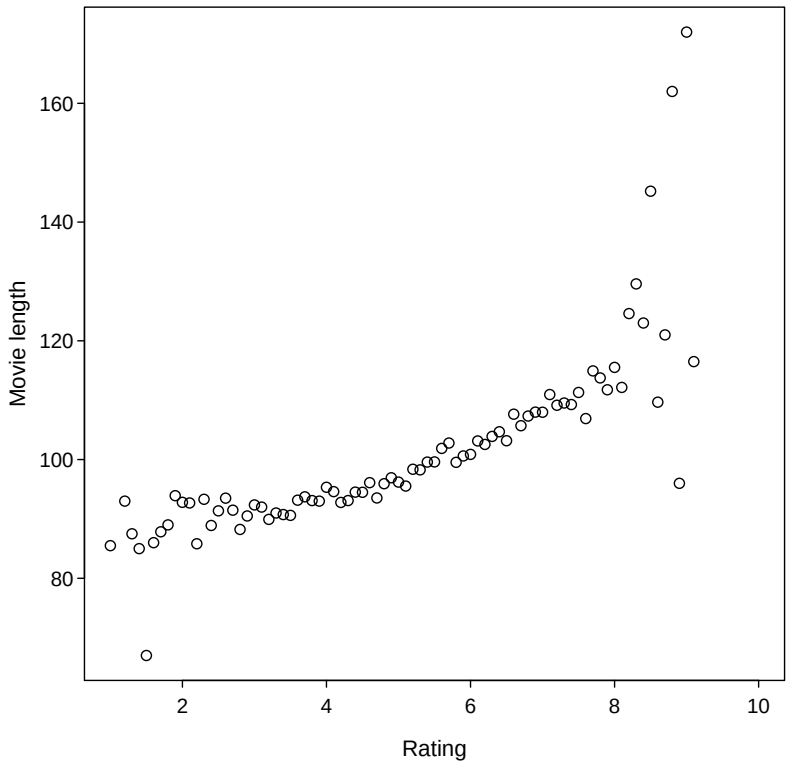


Figure: Plot of movie length conditional on it's rating.

5.7 Help

- R has a very good help system.
- If you need information about a particular function, say `plot`.
- Then typing `?plot` in a R terminal will bring up the associated help page.
- The internet is another very good source of R help.
- Unfortunately, using Google isn't particularly useful since the letter "R" appears on most web pages!
- However, you can use

  `http://www.rseek.org/`

  Using this search engine limits searches to R web-pages.