# Introducing the basics of R

Rolf Bänziger, rbanziger@westminster.ac.uk

25/1/2021

This tutorial introduces the basic prinicples of R and its built-in plotting functions.

## Vectors

Vectors are the fundamental data structures of R. A vector stores a ordered set of values, called elements. All elements of the vector must be of the same type, e.g. numeric (any real number), integer (any whole number), logical (true or false), or character (text).

Vectors are most often defined using the function `c()`.

```r
c(2,4,6)
```

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

```r
c("Alice", "Bob", "Charlie", "Dan", "Fiona", "Gab")
```

```
## [1] "Alice"   "Bob"     "Charlie" "Dan"     "Fiona"   "Gab"
```

Another function that is useful to create a vector is `seq()`, which creates a sequence. For example, the following command creates a vector containing all odd numbers between 1 and 100.

```r
# Odd numbers < 100
seq(1, 100, 2)
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
## [26] 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Vectors can be used in arithmetic operations.

```r
# Add 5 to each element
c(10, 20, 30, 40) + 5
```

```
## [1] 15 25 35 45
```

```r
# Divide each element by 5
c(10, 20, 30, 40) / 5
```

```
## [1] 2 4 6 8
```

## Variables

Any value can be stored in a variable, using the assignment operator `<-`. You will sometimes also see `=` used to assign values to an operator.

```
subjects <- c("Alice", "Bob", "Charlie", "Dan", "Fiona", "Gab")
weight <- c(60, 72, 57, 90, 72, 95)
height <- c(1.75, 1.80, 1.65, 1.90, 1.91, 1.74 )
```

Just the name of the variable will output its value.

```
subjects
```

```
## [1] "Alice"   "Bob"     "Charlie" "Dan"     "Fiona"   "Gab"
```

The code below calculates the BMI of the subjects using the variables we just defined.

```
weight / (height ^ 2)
```

```
## [1] 19.59184 22.22222 20.93664 24.93075 19.73630 31.37799
```

Of course, we can assign the output of this calculation to another variable.

```
bmi  <- weight / (height ^ 2)
```

## Logical values

Logical values can either be true or false. They can be either assigned by using the keywords `TRUE` and `FALSE` (alternatively `T` and `F`), or they can be computed, e.g. by using comparison operators (visit Operators for an overview of all operators.

```
# A simple vector containing logical values
c(TRUE, FALSE, TRUE, TRUE)
```

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

```
# Find BMIs over 25, indicating an obese person
bmi > 25
```

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

Logical vectors can be used to filter other vectors. For example, we can use the code below to retrieve all obese subjects.

```
subjects[bmi > 25]
```

```
## [1] "Gab"
```

### *Exercise*

Create a vector with the elements "Alex", "Aria", "Addison" and "Aurora" and save it in the variable `students`.

Create a vector exam1 containing the values 50, 65, 45 and 35. Create a vector exam2 containing the values 85, 72, 81 and 62. Create a vector exam3 containing the values 62, 55, 85 and 42.

Compute the average grades. Add exam1, exam2 and exam3 and divide the result by 3. Use parentheses if necessary.

The pass mark is 50%. List all students that have an average of at least 50.

```
students <- c("Alex", "Aria", "Addison", "Aurora")
exam1 <- c(50, 65, 45, 35)
exam2 <- c(85, 72, 81, 62)
exam3 <- c(62, 55, 85, 42)

average_grade <- (exam1 + exam2 + exam3) / 3
average_grade
```

```
## [1] 65.66667 64.00000 70.33333 46.33333
```

```
students[average_grade >= 50]
```

```
## [1] "Alex"    "Aria"    "Addison"
```

# Data frames

A data frame is a structure similar to a table. Most data that you will analyse is a data frame. A data frame is created using the data.frame function.

```
bmi_data <- data.frame(subjects, weight, height)
bmi_data
```

```
##   subjects weight height
## 1    Alice     60   1.75
## 2      Bob     72   1.80
## 3  Charlie     57   1.65
## 4      Dan     90   1.90
## 5    Fiona     72   1.91
## 6      Gab     95   1.74
```

Single columns are accessed using the $-sign and the column name.

```
bmi_data$subjects
```

```
## [1] "Alice"   "Bob"     "Charlie" "Dan"     "Fiona"   "Gab"
```

Several columns can be extracted with a vector containing the column names.

```r
bmi_data[c("subjects", "height")]
```

```
##   subjects height
## 1    Alice   1.75
## 2      Bob   1.80
## 3  Charlie   1.65
## 4      Dan   1.90
## 5    Fiona   1.91
## 6      Gab   1.74
```

Alternatively, we can use indices to access the columns.

```r
# First and fourth column
bmi_data[c(1, 3)]
```

```
##   subjects height
## 1    Alice   1.75
## 2      Bob   1.80
## 3  Charlie   1.65
## 4      Dan   1.90
## 5    Fiona   1.91
## 6      Gab   1.74
```

To extract single cells, we need to pass two indices. Rows are specified first, then columns.

```r
# Get second row, first column
bmi_data[2,1]
```

```
## [1] "Bob"
```

We can also specify ranges.

```r
# Get second and fourth row, columns 1 to 3
bmi_data[c(2, 3),1:3]
```

```
##   subjects weight height
## 2      Bob     72   1.80
## 3  Charlie     57   1.65
```

We don't need to specify columns and rows, we can just specify the rows…

```r
# First three rows, all columns
bmi_data[1:3,]
```

```
##   subjects weight height
## 1    Alice     60   1.75
## 2      Bob     72   1.80
## 3  Charlie     57   1.65
```

…or columns.

```
# Second and third columns, all rows
bmi_data[,c(2,3)]
```

```
##   weight height
## 1     60   1.75
## 2     72   1.80
## 3     57   1.65
## 4     90   1.90
## 5     72   1.91
## 6     95   1.74
```

And we can also specify logical values:

```
# Get second and third columns and first three rows
cols <- c(F, T, T)
rows <- c(T, T, T, F, F, F)
bmi_data[rows, cols]
```

```
##   weight height
## 1     60   1.75
## 2     72   1.80
## 3     57   1.65
```

## Creating new columns

Data frame columns can be used in calculations like vectors, and the vectors can be assigned to new columns.

```
bmi_data$bmi <- bmi_data$weight / (bmi_data$height ^ 2)
bmi_data
```

```
##   subjects weight height      bmi
## 1    Alice     60   1.75 19.59184
## 2      Bob     72   1.80 22.22222
## 3  Charlie     57   1.65 20.93664
## 4      Dan     90   1.90 24.93075
## 5    Fiona     72   1.91 19.73630
## 6      Gab     95   1.74 31.37799
```

Let' find obese subjects.

```
bmi_data[bmi_data$bmi>25,]
```

```
##   subjects weight height      bmi
## 6      Gab     95   1.74 31.37799
```

### *Exercise*

1. Create a data frame students_data using the vectors students, exam1, exam2 and exam3.
2. Add a column to students_data called average_grade, which contains the average of all exams.
3. Show all the names and average grade of all students who passed.

```r
students_data <- data.frame(students, exam1, exam2, exam3)
students_data$average_grade <-
  (students_data$exam1 + students_data$exam2 + students_data$exam3) / 3
students_data[students_data$average_grade >= 50, c("students", "average_grade")]
```

```
##   students average_grade
## 1     Alex      65.66667
## 2     Aria      64.00000
## 3  Addison      70.33333
```

## Built-in data sets

R contains a number of built in data sets. Use `data()` to get a list of all loaded data sets.

```r
data()
```

Review a description of a built in data set with `?` (or the `help()` function).

```r
? cars
```

Use the data set like any data frame variable.

```r
# Retrieve first 10 rows
cars[1:10,]
```

```
##    speed dist
## 1      4    2
## 2      4   10
## 3      7    4
## 4      7   22
## 5      8   16
## 6      9   10
## 7     10   18
## 8     10   26
## 9     10   34
## 10    11   17
```

R provides a number of functions to quickly explore data frames.

```r
# Column names
colnames(cars)
```

```
## [1] "speed" "dist"
```

```r
# list first 6 rows
head(cars)
```

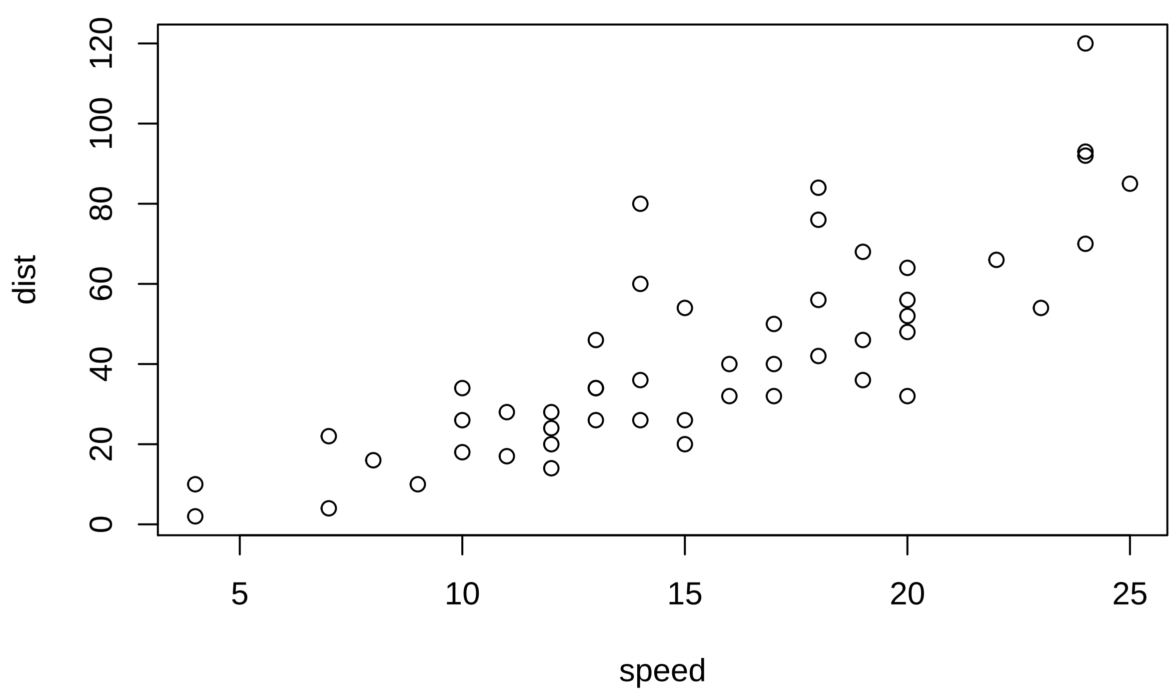
```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```r
# statistical summary
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

And of course, we can plot the data.

```r
plot(cars)
```



### *Exercise*

Familiarise yourself with the following data sets:

- airquality
- mtcars
- pressure

What does the data represent? What are the dimensions (columns)?

```
? airquality
plot(airquality)
```

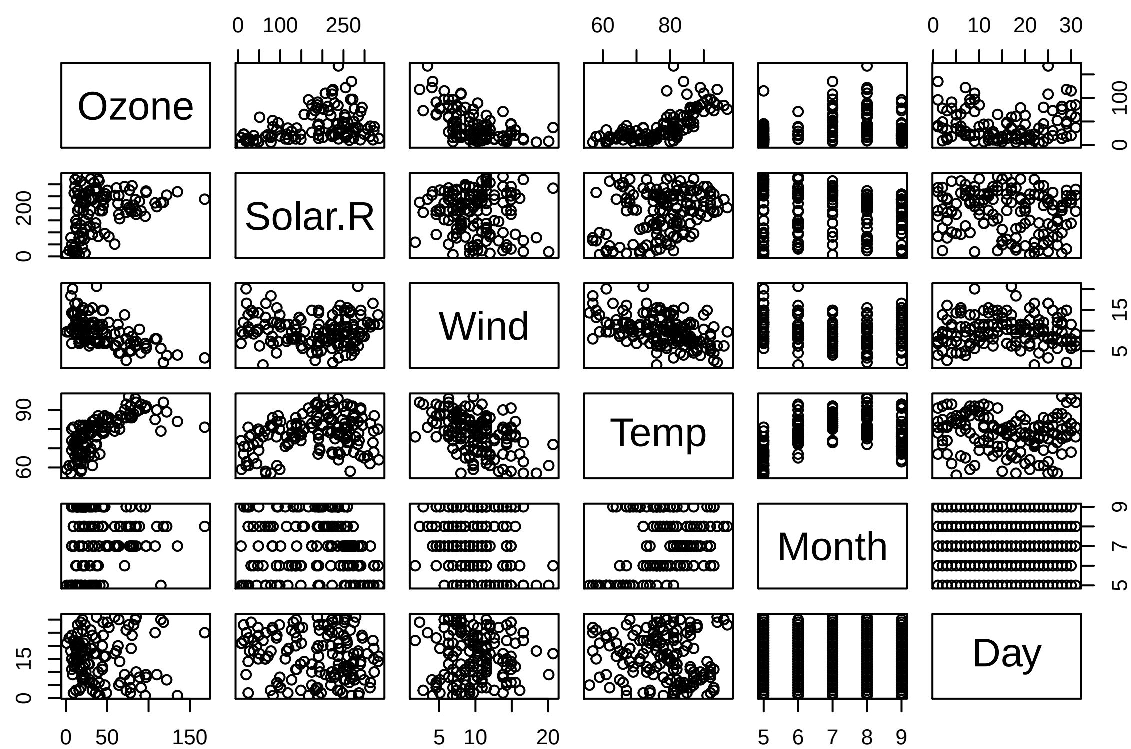
```
? mtcars
plot(mtcars)
```

```
? pressure
plot(pressure)
```

## Plots in R

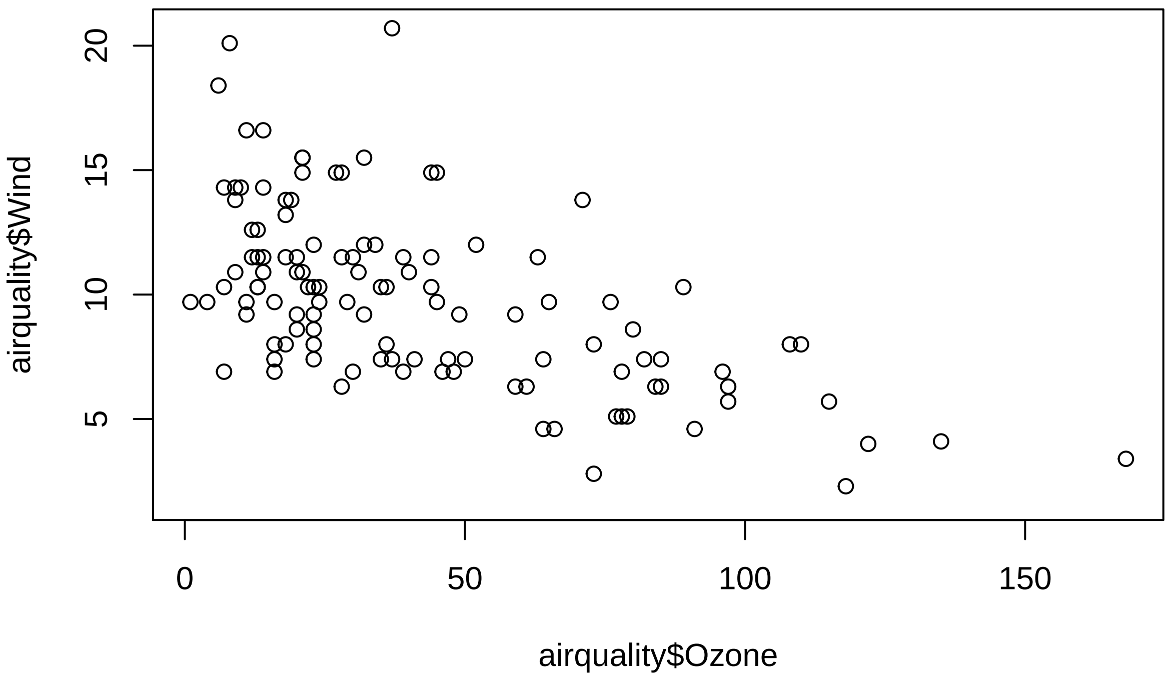
`plot()` is a generic function and produces an output based on the type of the data passed to it. We will use a number of built-in data sets to demonstrate the functionality of `plot()`.

```
plot(airquality)
```



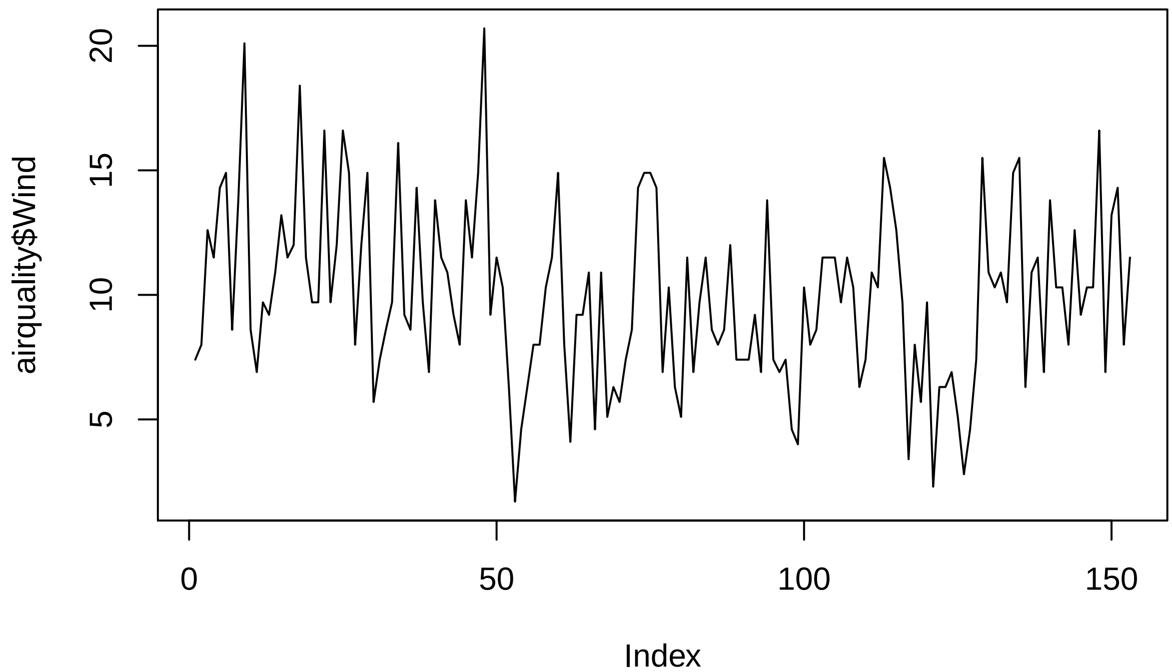
Two parameters are treated as x and y axis and produces a scatter plot.

```
plot(airquality$Ozone, airquality$Wind)
```




By default the plot() function produces a scatter plot with dots. To make a line graph, pass it the vector of x and y values, and specify type = "l" for line (or p for points, b: both, etc.):

```
plot(airquality$Wind , type = "l")
```



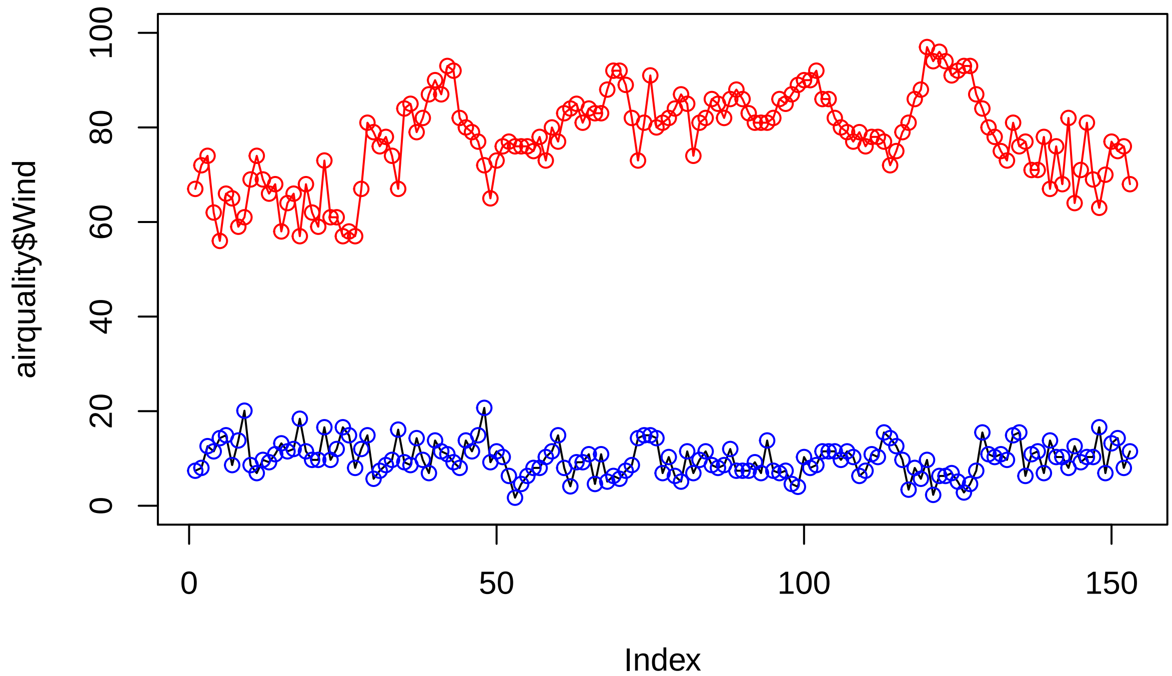

It is possible to include multiple data series in the same plot. First, call `plot()`, then add additional series with `lines()` and `points()`. Note that we need to specify `ylim =` to make sure the plotting area is big enough to plot the second series.

```
# base graphic
plot(airquality$Wind, type = "l", ylim = c(0, 100))

# add points
points(airquality$Wind, col= "blue")

# add second line in red color
lines(airquality$Temp, col = "red")

# add points to second line
points(airquality$Temp, col = "red")
```

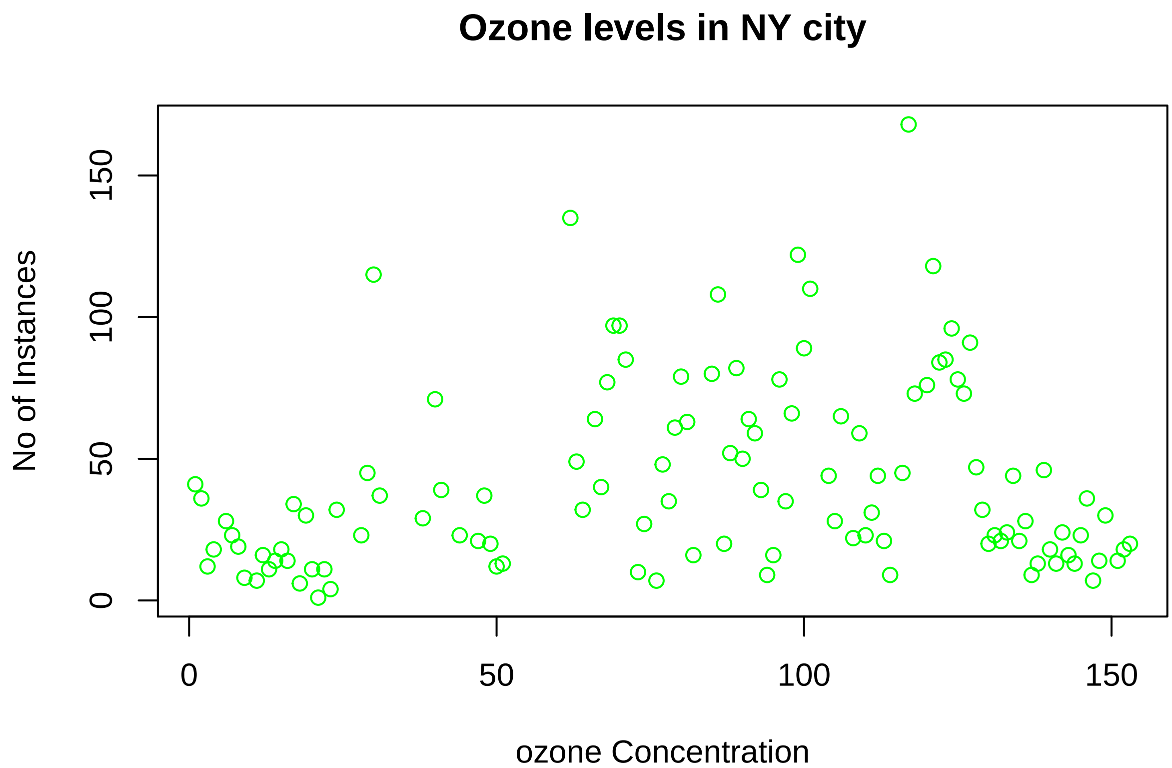



NB: we specified the colour of the elements with the `col =` parameter. Use `colors()` to get a list of all supported colors.

## Labels and Titles

We can also label the X and the Y axis and give a title to our plot. Additionally, we also have the option of giving color to the plot.

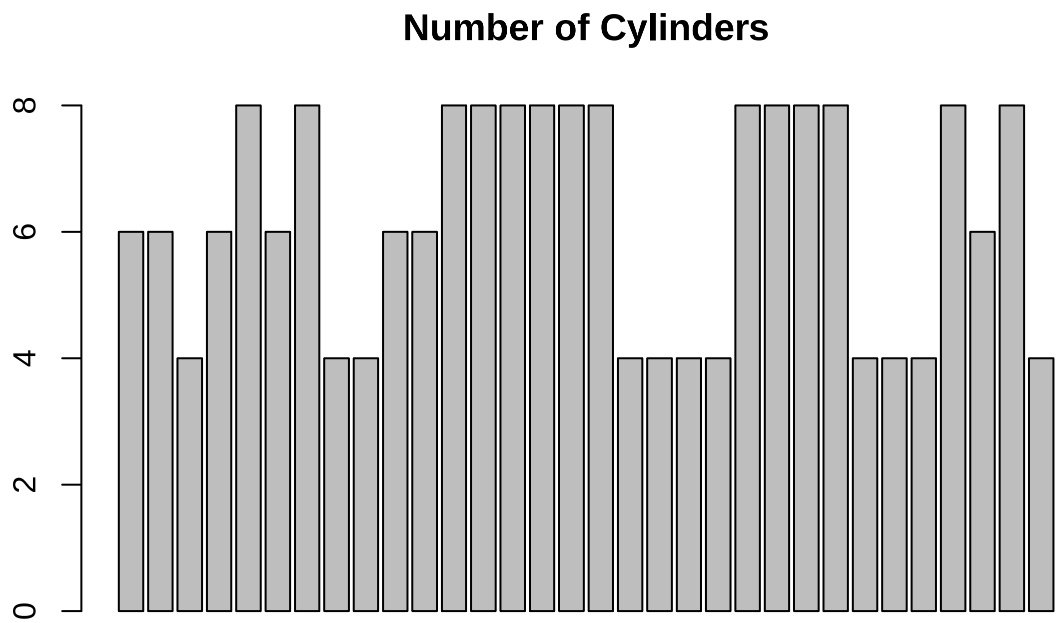
```
plot(airquality$Ozone, xlab = 'ozone Concentration', ylab = 'No of Instances',
     main = 'Ozone levels in NY city', col = 'green')
```




### Bar plot

The function `barplot()` creates bar plots.

```
barplot(mtcars$cyl, main = "Number of Cylinders")
```



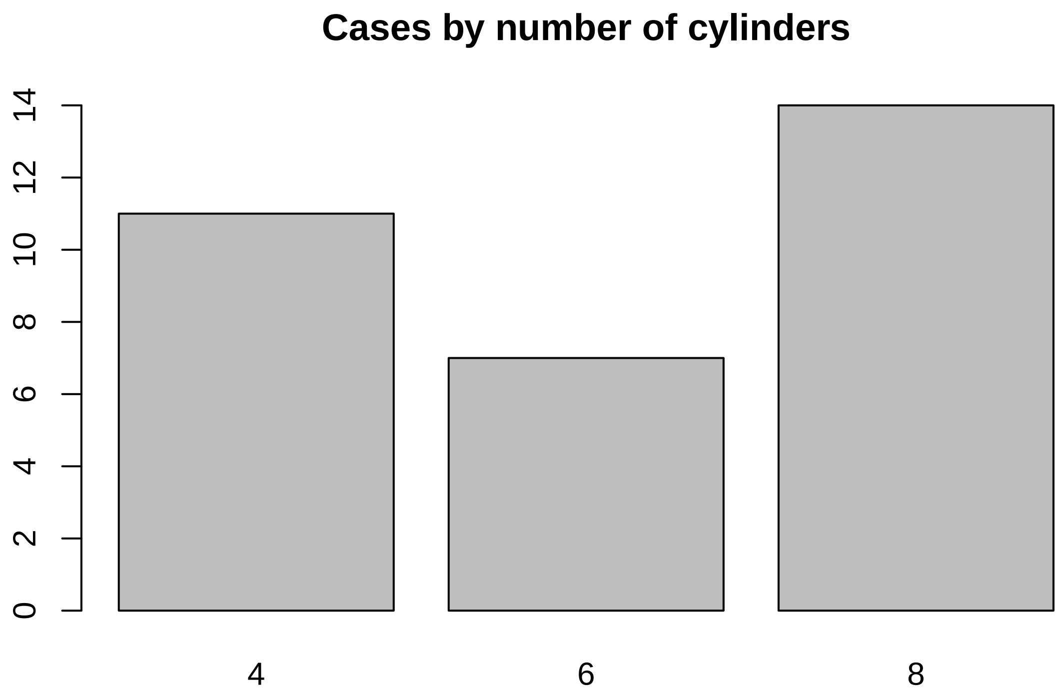

We see that the mtcars data set contains cases with 4, 6 or 8 cylinders. If you want the bar plot to show the number of records in each category, use the table function to count the number of cases per category.

```
table(mtcars$cyl)
```

```
##
##  4  6  8
## 11  7 14
```

Pass the result to barplot.

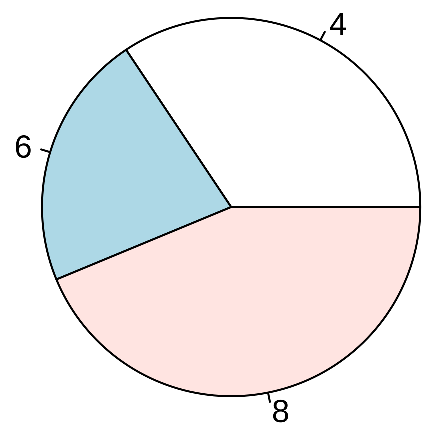
```
carsPerCyl <- table(mtcars$cyl)
barplot(carsPerCyl, main = "Cases by number of cylinders")
```




### Pie chart

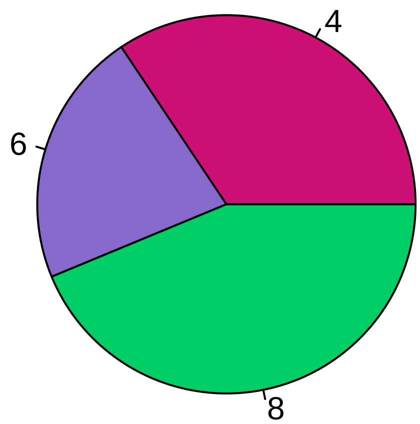
Use `pie()` to create pie charts. We can create the same chart as above as a pie chart.

```
pie(carsPerCyl)
```



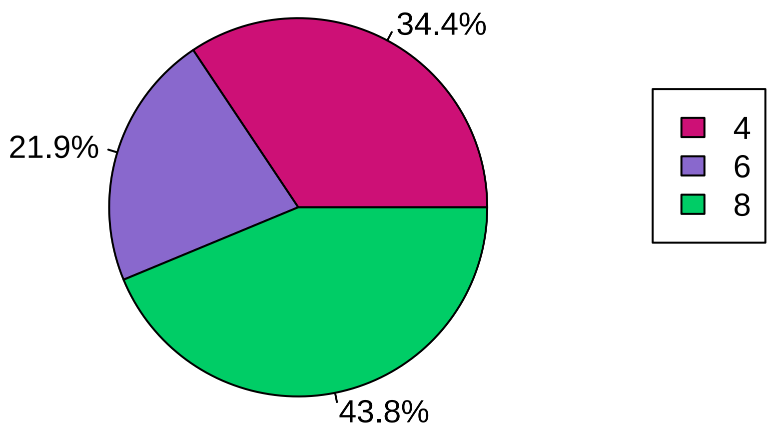

Let's add a heading and define our own colors.

```
colours <- c("deeppink3", "mediumpurple3", "springgreen3")
pie(carsPerCyl, col = colours)
```




The `legend()` function adds a legend to the chart.

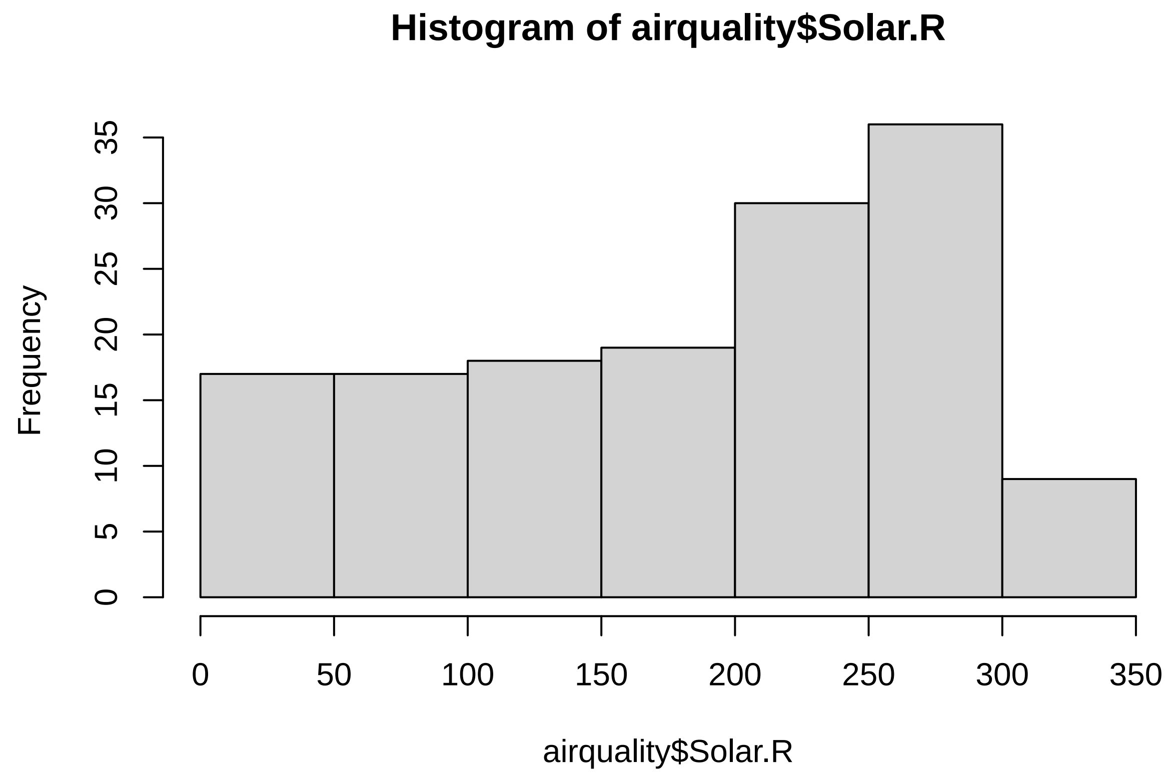
```
labels <- round(carsPerCyl/sum(carsPerCyl) * 100, 1)
labels <- paste0(labels, "%")
pie(carsPerCyl, col = colours, labels = labels)
legend(1.5, 0.5, names(carsPerCyl), fill = colours)
```




### Histogram

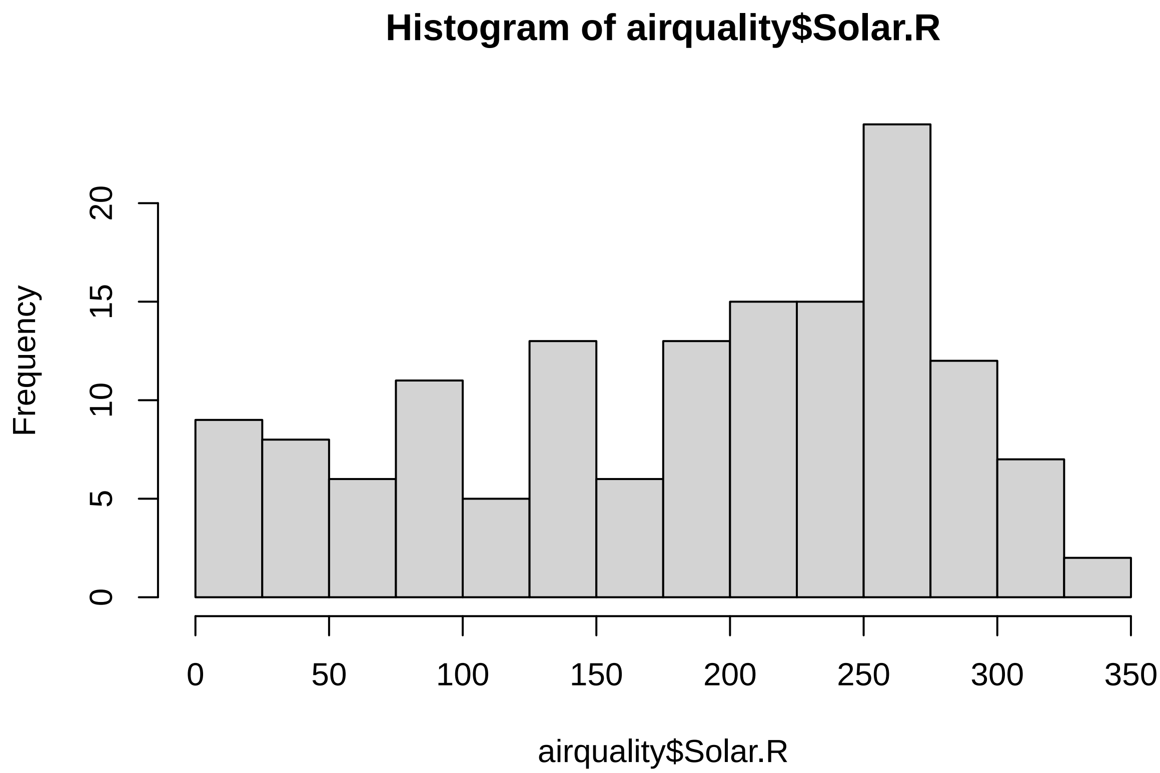
A histogram show the data distribution. It shows the frequencies of values in so called buckets (ranges).

```
hist(airquality$Solar.R)
```




It is often desirable to specify your own values for the number of buckets.

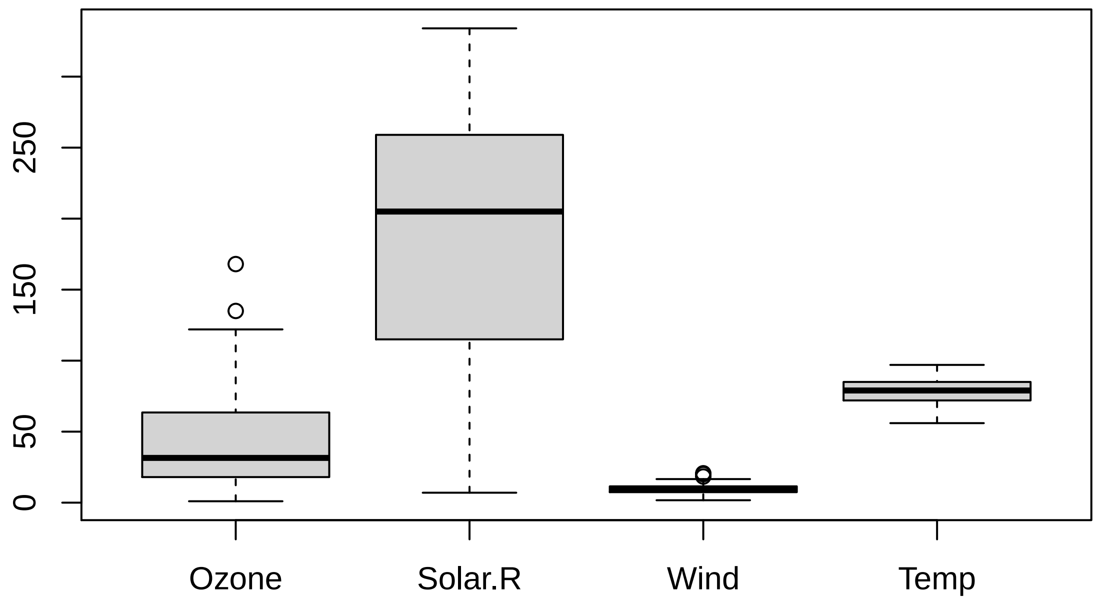
```
hist(airquality$Solar.R, breaks = seq(from = 0, to = 350, by = 25))
```




### Boxplot

Another way to display the distribution of data is the box plot. A box plot displays quartiles (25 percentile, median, 75 percentile), minimum, maximum and outliers.

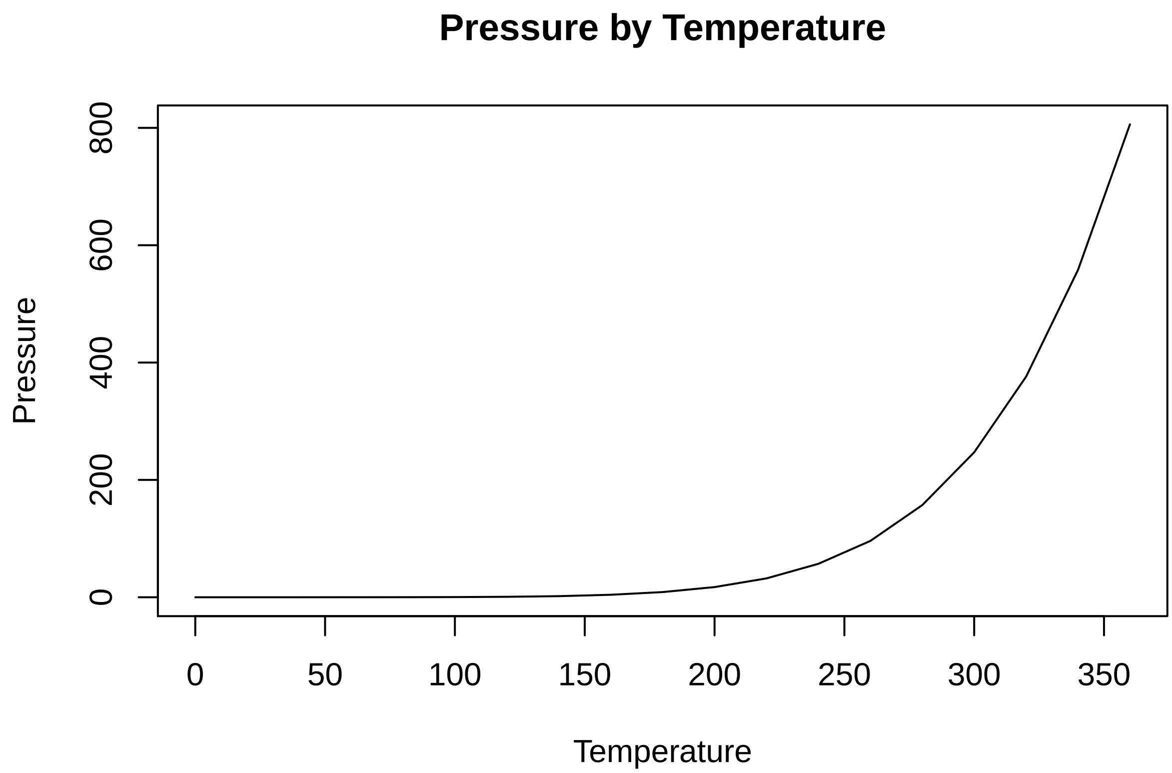
```
boxplot(airquality[,1:4])
```




### ***Exercise***

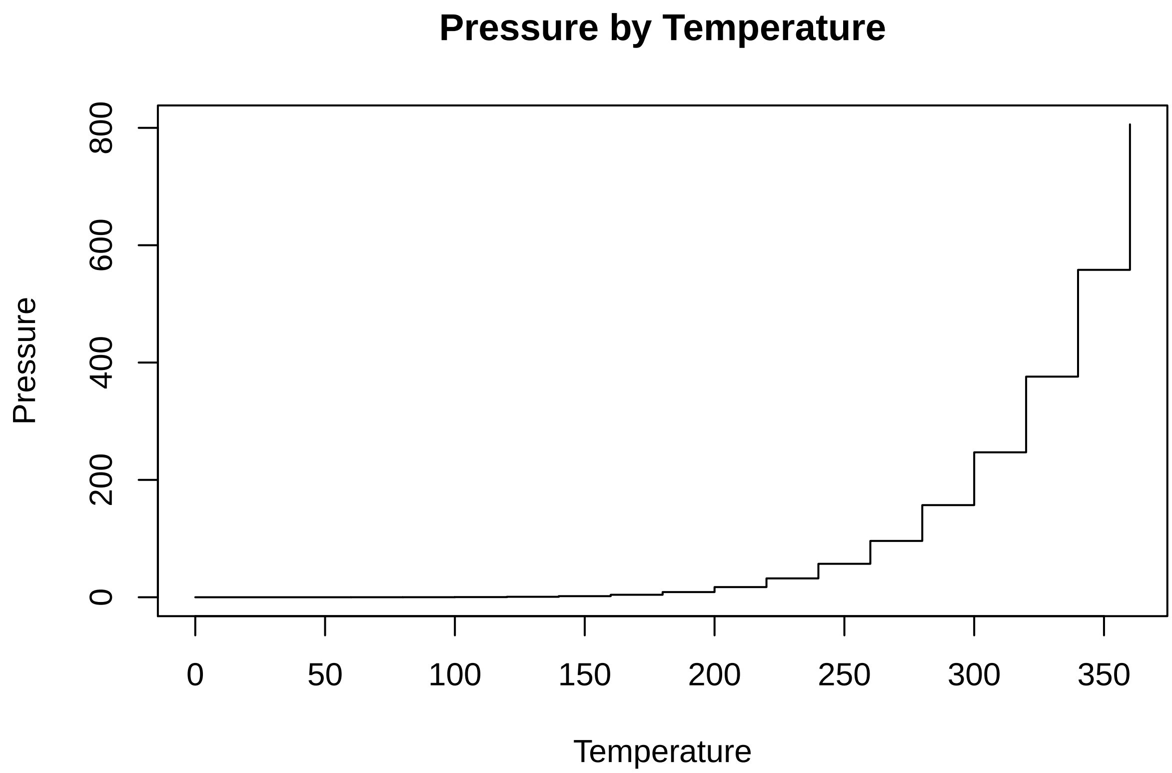
1. Using the `pressure` data set, create a line chart showing the pressure by temperature. Give the chart a title and label the axes accordingly.

```
plot(pressure$temperature, pressure$pressure, type= 'l',
     main = "Pressure by Temperature", xlab = "Temperature", ylab = "Pressure")
```



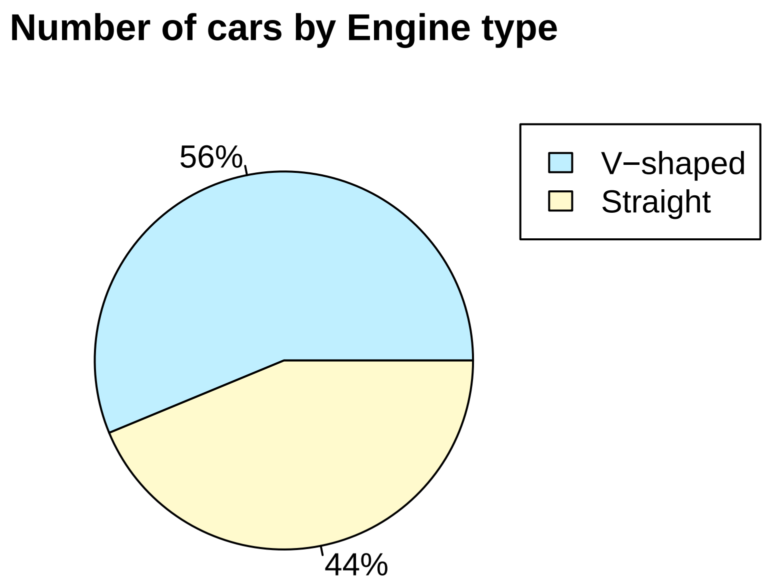

2. Create the same chart with a stepped line instead of a smooth line.

```
plot(pressure$temperature, pressure$pressure, type= 's',
      main = "Pressure by Temperature", xlab = "Temperature", ylab = "Pressure")
```



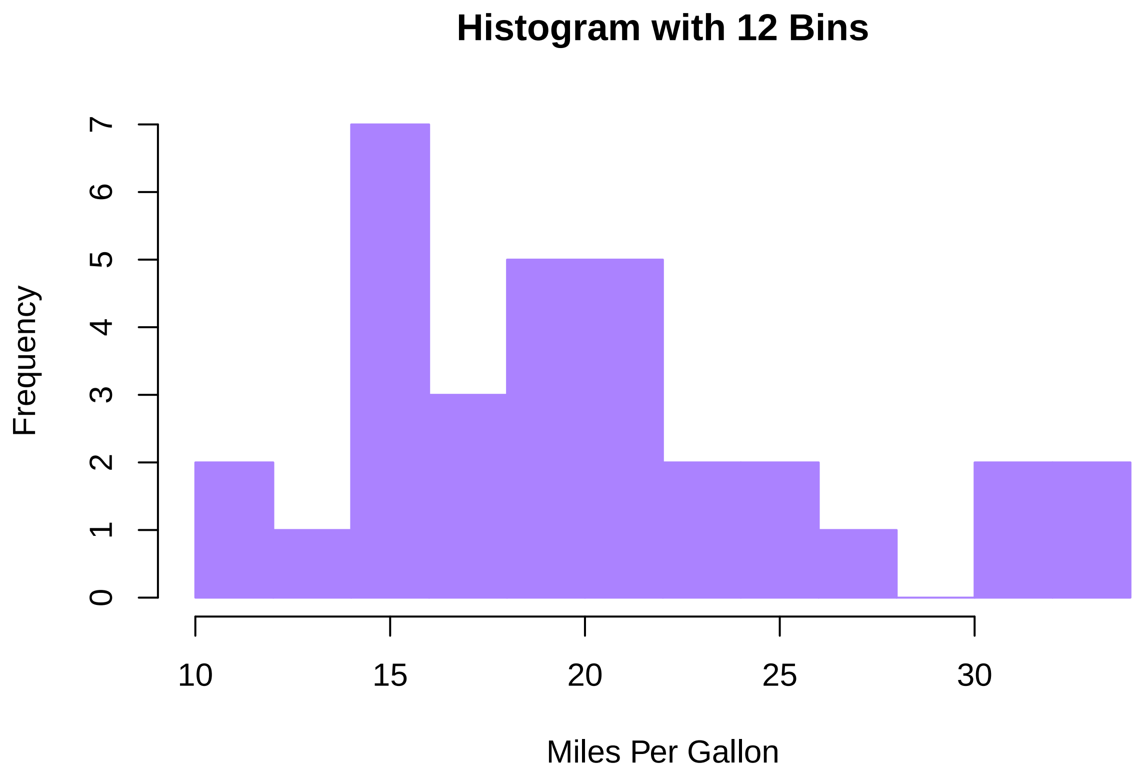

3. Create a pie chart showing the number of cars in the `mtcars` data set by engine form. Place the legend in the upper right corner.

```
carsByEngine <- table(mtcars$vs)
labels <- paste0(round(carsByEngine/sum(carsByEngine)*100), "%")
colours <- c("lightblue1", "lemonchiffon")
pie(carsByEngine, labels = labels, col = colours, main = "Number of cars by Engine type")
legend(1.0, 1.0, c("V-shaped", "Straight"), fill = colours)
```



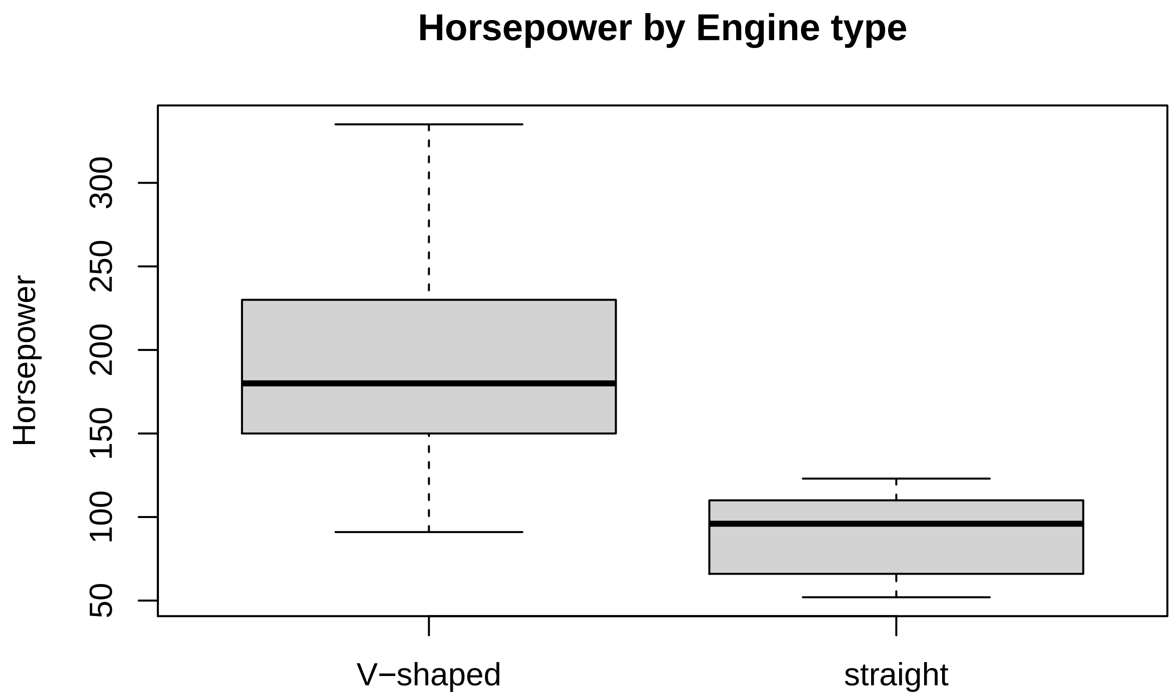

4. Create a histogram of the Miles Per Gallon values in the `mtcars` data set with 12 bins. Label the x-axis and fill the columns purple (including border the border).

```
hist(mtcars$mpg, breaks =  12, xlab = "Miles Per Gallon",
     main = "Histogram with 12 Bins", col = "mediumpurple1", border = "mediumpurple1")
```



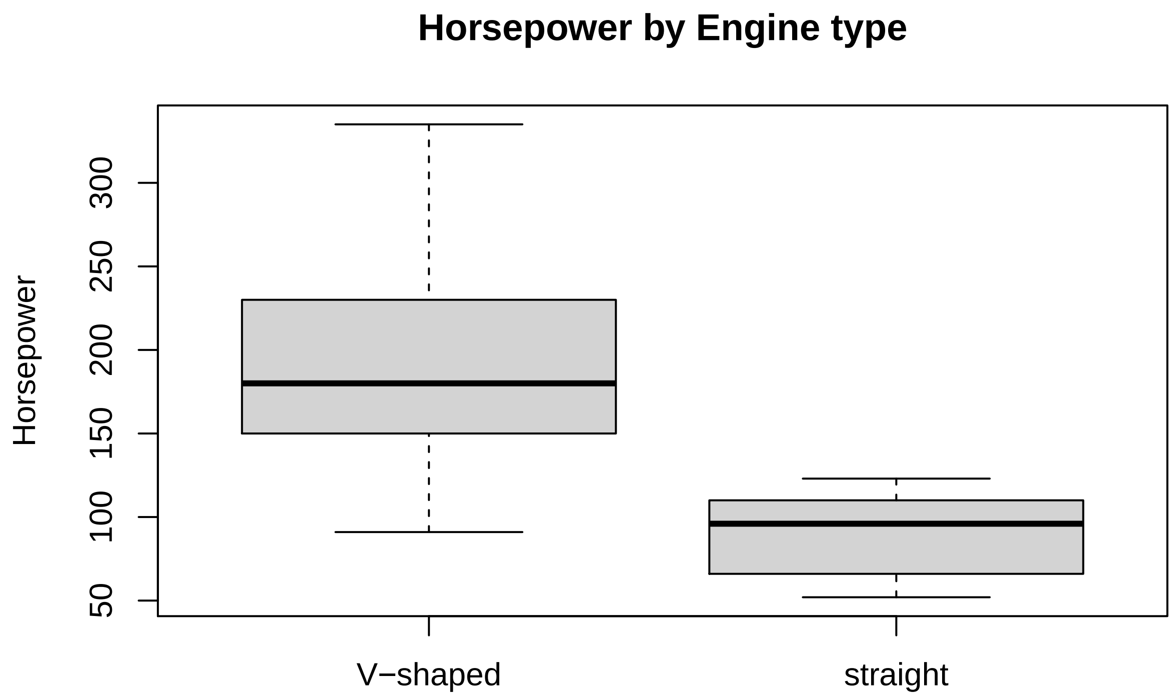

5. Create a box plot showing the distribution of the Horsepower by engine type. Label the chart reasonably.

```
vShaped <- mtcars[mtcars$vs==0,"hp"]
straight <- mtcars[mtcars$vs==1,"hp"]
boxplot(vShaped, straight, names = c("V-shaped", "straight"),
        main = "Horsepower by Engine type", xlab = NULL, ylab = "Horsepower")
```




Alternatively:

```
boxplot(mtcars$hp ~ mtcars$vs, main = "Horsepower by Engine type", xlab = NULL,
        ylab = "Horsepower", names = c("V-shaped", "straight"))
```


Horsepower by Engine type
Horsepower
50
100
150
200
250
300
V−shaped
straight