Data Import:

From XLS:

> library(gdata)                   # load gdata package    
> mydata = read.xls("mydata.xls")  # read from first sheet

**Table File**

A data table can resides in a **text file**. The cells inside the table are separated by blank characters. Here is an example of a table with 4 rows and 3 columns.

100   a1   b1   
200   a2   b2   
300   a3   b3   
400   a4   b4

Now copy and paste the table above in a file named "mydata.txt" with a text editor. Then load the data into the workspace with the function read.table.

> mydata = read.table("mydata.txt")  # read text file   
> mydata                             # print data frame   
   V1 V2 V3   
1 100 a1 b1   
2 200 a2 b2   
3 300 a3 b3   
4 400 a4 b4

**CSV File**

The sample data can also be in **comma separated values**(CSV) format. Each cell inside such data file is separated by a special character, which usually is a comma, although other characters can be used as well.

The first row of the data file should contain the column names instead of the actual data. Here is a sample of the expected format.

Col1,Col2,Col3   
100,a1,b1   
200,a2,b2   
300,a3,b3

After we copy and paste the data above in a file named "mydata.csv" with a text editor, we can read the data with the function read.csv.

> mydata = read.csv("mydata.csv")  # read csv file   
> mydata   
  Col1 Col2 Col3   
1  100   a1   b1   
2  200   a2   b2   
3  300   a3   b3

getwd()

setwd("C:/MyDoc")

install.packages() to install new packages

For Help:

?c or help(c)

**Not sure about the function**; try apropos("nova")

To check the datatype use class()

**Two character values can be concatenated with the paste function**.

> fname = "Joe"; lname ="Smith"   
> paste(fname, lname)   
[1] "Joe Smith"

To extract a substring, we apply the **substr** function. Here is an example showing how to extract the substring between the third and twelfth positions in a string.

> substr("Mary has a little lamb.", start=3, stop=12)   
[1] "ry has a l"

And to replace the first occurrence of the word "little" by another word "big" in the string, we apply the sub function.

> sub("little", "big", "Mary has a little lamb.")   
[1] "Mary has a big lamb."

# Vector

A **vector**is a sequence of data elements of the same basic type. Members in a vector are officially called **components**. Nevertheless, we will just call them **members**in this site.

Here is a vector containing three numeric values 2, 3 and 5.

> c(2, 3, 5)   
[1] 2 3 5

And here is a vector of logical values.

> c(TRUE, FALSE, TRUE, FALSE, FALSE)   
[1]  TRUE FALSE  TRUE FALSE FALSE

A vector can contain character strings.

> c("aa", "bb", "cc", "dd", "ee")   
[1] "aa" "bb" "cc" "dd" "ee"

Incidentally, the number of members in a vector is given by the length function.

> length(c("aa", "bb", "cc", "dd", "ee"))   
[1] 5

# Combining Vectors

Vectors can be combined via the function c. For examples, the following two vectors n and sare combined into a new vector containing elements from both vectors.

> n = c(2, 3, 5)   
> s = c("aa", "bb", "cc", "dd", "ee")   
> c(n, s)   
[1] "2"  "3"  "5"  "aa" "bb" "cc" "dd" "ee"

**Recycling Rule**

If two vectors are of unequal length, the shorter one will be recycled in order to match the longer vector. For example, the following vectors u and v have different lengths, and their sum is computed by recycling values of the shorter vector u.

> u = c(10, 20, 30)   
> v = c(1, 2, 3, 4, 5, 6, 7, 8, 9)   
> u + v   
[1] 11 22 33 14 25 36 17 28 39

# Vector Index

We retrieve values in a vector by declaring an index inside a *single square bracket*"[]"operator.

For example, the following shows how to retrieve a vector member. Since the vector index is 1-based, we use the index position 3 for retrieving the third member.

> s = c("aa", "bb", "cc", "dd", "ee")   
> s[3]   
[1] "cc"

**Negative Index**

If the index is negative, it would strip the member whose position has the same absolute value as the negative index. For example, the following creates a vector slice with the third member removed.

> s[-3]   
[1] "aa" "bb" "dd" "ee"

**Out-of-Range Index**

If an index is out-of-range, a missing value will be reported via the symbol NA.

> s[10]   
[1] NA

# Numeric Index Vector

A new vector can be sliced from a given vector with a **numeric index vector**, which consists of member positions of the original vector to be retrieved.

Here it shows how to retrieve a vector slice containing the second and third members of a given vector s.

> s = c("aa", "bb", "cc", "dd", "ee")   
> s[c(2, 3)]   
[1] "bb" "cc"

#### Duplicate Indexes

The index vector allows duplicate values. Hence the following retrieves a member twice in one operation.

> s[c(2, 3, 3)]   
[1] "bb" "cc" "cc"

#### Out-of-Order Indexes

The index vector can even be out-of-order. Here is a vector slice with the order of first and second members reversed.

> s[c(2, 1, 3)]   
[1] "bb" "aa" "cc"

#### Range Index

To produce a vector slice between two indexes, we can use the colon operator ":". This can be convenient for situations involving large vectors.

> s[2:4]   
[1] "bb" "cc" "dd"

# Logical Index Vector

A new vector can be sliced from a given vector with a **logical index vector**, which has the same length as the original vector. Its members are TRUE if the corresponding members in the original vector are to be included in the slice, and FALSE if otherwise.

For example, consider the following vector s of length 5.

> s = c("aa", "bb", "cc", "dd", "ee")

To retrieve the the second and fourth members of s, we define a logical vector L of the same length, and have its second and fourth members set as TRUE.

> L = c(FALSE, TRUE, FALSE, TRUE, FALSE)   
> s[L]   
[1] "bb" "dd"

The code can be abbreviated into a single line.

> s[c(FALSE, TRUE, FALSE, TRUE, FALSE)]   
[1] "bb" "dd"

# Named Vector Members

We can assign names to vector members.

For example, the following variable v is a character string vector with two members.

> v = c("Mary", "Sue")   
> v   
[1] "Mary" "Sue"

We now name the first member as First, and the second as Last.

> names(v) = c("First", "Last")   
> v   
 First   Last   
"Mary"  "Sue"

Then we can retrieve the first member by its name.

> v["First"]   
[1] "Mary"

Furthermore, we can reverse the order with a character string index vector.

> v[c("Last", "First")]   
  Last  First   
 "Sue" "Mary"

A **matrix**is a collection of data elements arranged in a two-dimensional rectangular layout. The following is an example of a matrix with 2 rows and 3 columns.

    [         ]
      2  4  3
A =   1  5  7


We reproduce a memory representation of the matrix in R with the matrix function. The data elements must be of the same basic type.

> A = matrix(   
+   c(2, 4, 3, 1, 5, 7), # the data elements   
+   nrow=2,              # number of rows   
+   ncol=3,              # number of columns   
+   byrow = TRUE)        # fill matrix by rows   
   
> A                      # print the matrix   
     [,1] [,2] [,3]   
[1,]    2    4    3   
[2,]    1    5    7

An element at the *mth* row, *nth* column of A can be accessed by the expression A[m, n].

> A[2, 3]      # element at 2nd row, 3rd column   
[1] 7

The entire *mth* row A can be extracted as A[m, ].

> A[2, ]       # the 2nd row   
[1] 1 5 7

Similarly, the entire *nth* column A can be extracted as A[ ,n].

> A[ ,3]       # the 3rd column   
[1] 3 7

We can also extract more than one rows or columns at a time.

> A[ ,c(1,3)]  # the 1st and 3rd columns   
     [,1] [,2]   
[1,]    2    3   
[2,]    1    7

If we assign names to the rows and columns of the matrix, than we can access the elements by names.

> dimnames(A) = list(   
+   c("row1", "row2"),         # row names   
+   c("col1", "col2", "col3")) # column names   
   
> A                 # print A   
     col1 col2 col3   
row1    2    4    3   
row2    1    5    7   
   
> A["row2", "col3"] # element at 2nd row, 3rd column   
[1] 7

# Matrix Construction

There are various ways to construct a matrix. When we construct a matrix directly with data elements, the matrix content is filled along the column orientation by default. For example, in the following code snippet, the content of B is filled along the columns consecutively.

> B = matrix(   
+   c(2, 4, 3, 1, 5, 7),   
+   nrow=3,   
+   ncol=2)   
   
> B             # B has 3 rows and 2 columns   
     [,1] [,2]   
[1,]    2    1   
[2,]    4    5   
[3,]    3    7

#### Transpose

We construct the **transpose**of a matrix by interchanging its columns and rows with the function t .

> t(B)          # transpose of B   
     [,1] [,2] [,3]   
[1,]    2    4    3   
[2,]    1    5    7

#### Combining Matrices

The columns of two matrices having the same number of rows can be combined into a larger matrix. For example, suppose we have another matrix C also with 3 rows.

> C = matrix(   
+   c(7, 4, 2),   
+   nrow=3,   
+   ncol=1)   
   
> C             # C has 3 rows   
     [,1]   
[1,]    7   
[2,]    4   
[3,]    2

Then we can combine the columns of B and C with cbind.

> cbind(B, C)   
     [,1] [,2] [,3]   
[1,]    2    1    7   
[2,]    4    5    4   
[3,]    3    7    2

Similarly, we can combine the rows of two matrices if they have the same number of columns with the rbind function.

> D = matrix(   
+   c(6, 2),   
+   nrow=1,   
+   ncol=2)   
   
> D             # D has 2 columns   
     [,1] [,2]   
[1,]    6    2   
   
> rbind(B, D)   
     [,1] [,2]   
[1,]    2    1   
[2,]    4    5   
[3,]    3    7   
[4,]    6    2

#### Deconstruction

We can deconstruct a matrix by applying the c function, which combines all column vectors into one.

> c(B)   
[1] 2 4 3 1 5 7

# List

A **list**is a generic vector containing other objects.

For example, the following variable x is a list containing copies of three vectors n, s, b, and a numeric value 3.

> n = c(2, 3, 5)   
> s = c("aa", "bb", "cc", "dd", "ee")   
> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)   
> x = list(n, s, b, 3)   # x contains copies of n, s, b

#### List Slicing

We retrieve a list slice with the *single square bracket*"[]" operator. The following is a slice containing the second member of x, which is a copy of s.

> x[2]   
[[1]]   
[1] "aa" "bb" "cc" "dd" "ee"

With an index vector, we can retrieve a slice with multiple members. Here a slice containing the second and fourth members of x.

> x[c(2, 4)]   
[[1]]   
[1] "aa" "bb" "cc" "dd" "ee"   
   
[[2]]   
[1] 3

#### Member Reference

In order to reference a list member directly, we have to use the *double square bracket* "[[]]"operator. The following object x[[2]] is the second member of x. In other words, x[[2]] is a copy of s, but is *not*a slice containing s or its copy.

> x[[2]]   
[1] "aa" "bb" "cc" "dd" "ee"

We can modify its content directly.

> x[[2]][1] = "ta"   
> x[[2]]   
[1] "ta" "bb" "cc" "dd" "ee"   
> s   
[1] "aa" "bb" "cc" "dd" "ee"   # s is unaffected

# Named List Members

We can assign names to list members, and reference them by names instead of numeric indexes.

For example, in the following, v is a list of two members, named "bob" and "john".

> v = list(bob=c(2, 3, 5), john=c("aa", "bb"))   
> v   
$bob   
[1] 2 3 5   
   
$john   
[1] "aa" "bb"

#### List Slicing

We retrieve a list slice with the *single square bracket*"[]" operator. Here is a list slice containing a member of v named "bob".

> v["bob"]   
$bob   
[1] 2 3 5

With an index vector, we can retrieve a slice with multiple members. Here is a list slice with both members of v. Notice how they are reversed from their original positions in v.

> v[c("john", "bob")]   
$john   
[1] "aa" "bb"   
   
$bob   
[1] 2 3 5

#### Member Reference

In order to reference a list member directly, we have to use the *double square* *bracket*"[[]]"operator. The following references a member of v by name.

> v[["bob"]]   
[1] 2 3 5

A named list member can also be referenced directly with the "$" operator in lieu of the double square bracket operator.

> v$bob   
[1] 2 3 5

#### Search Path Attachment

We can *attach*a list to the R search path and access its members without explicitly mentioning the list. It should to be *detached*for cleanup.

> attach(v)   
> bob   
[1] 2 3 5   
> detach(v)

# Data Frame

A **data frame**is used for storing data tables. It is a list of vectors of equal length. For example, the following variable df is a data frame containing three vectors n, s, b.

> n = c(2, 3, 5)   
> s = c("aa", "bb", "cc")   
> b = c(TRUE, FALSE, TRUE)   
> df = data.frame(n, s, b)       # df is a data frame

#### Build-in Data Frame

We use built-in data frames in R for our tutorials. For example, here is a built-in data frame in R, called **mtcars**.

> mtcars   
               mpg cyl disp  hp drat   wt ...   
Mazda RX4     21.0   6  160 110 3.90 2.62 ...   
Mazda RX4 Wag 21.0   6  160 110 3.90 2.88 ...   
Datsun 710    22.8   4  108  93 3.85 2.32 ...   
               ............

The top line of the table, called the **header**, contains the column names. Each horizontal line afterward denotes a **data row**, which begins with the name of the row, and then followed by the actual data. Each data member of a row is called a **cell**.

To retrieve data in a cell, we would enter its row and column coordinates in the *single square bracket*"[]" operator. The two coordinates are separated by a comma. In other words, the coordinates begins with row position, then followed by a comma, and ends with the column position. The order is important.

Here is the cell value from the first row, second column of mtcars.

> mtcars[1, 2]   
[1] 6

Moreover, we can use the row and column names instead of the numeric coordinates.

> mtcars["Mazda RX4", "cyl"]   
[1] 6

Lastly, the number of data rows in the data frame is given by the nrow function.

> nrow(mtcars)    # number of data rows   
[1] 32

And the number of columns of a data frame is given by the ncol function.

> ncol(mtcars)    # number of columns   
[1] 11

Further details of the mtcars data set is available in the R documentation.

> help(mtcars)

#### Preview

Instead of printing out the entire data frame, it is often desirable to preview it with the headfunction beforehand.

> head(mtcars)   
               mpg cyl disp  hp drat   wt ...   
Mazda RX4     21.0   6  160 110 3.90 2.62 ...   
               ............

# Data Frame Column Vector

We reference a data frame column with the *double square bracket*"[[]]" operator.

For example, to retrieve the ninth column vector of the built-in data set [mtcars](http://www.r-tutor.com/node/10), we write mtcars[[9]].

> mtcars[[9]]   
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...

We can retrieve the same column vector by its name.

> mtcars[["am"]]   
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...

We can also retrieve with the "$" operator in lieu of the double square bracket operator.

> mtcars$am   
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...

Yet another way to retrieve the same column vector is to use the *single square* *bracket*"[]"operator. We prepend the column name with a comma character, which signals a wildcard match for the row position.

> mtcars[,"am"]   
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...

# Data Frame Column Slice

We retrieve a data frame column slice with the *single square bracket*"[]" operator.

#### Numeric Indexing

The following is a slice containing the first column of the built-in data set [mtcars](http://www.r-tutor.com/node/10).

> mtcars[1]   
                   mpg   
Mazda RX4         21.0   
Mazda RX4 Wag     21.0   
Datsun 710        22.8   
                   ............

#### Name Indexing

We can retrieve the same column slice by its name.

> mtcars["mpg"]   
                   mpg   
Mazda RX4         21.0   
Mazda RX4 Wag     21.0   
Datsun 710        22.8   
                   ............

To retrieve a data frame slice with the two columns mpg and hp, we pack the column names in an index vector inside the single square bracket operator.

> mtcars[c("mpg", "hp")]   
                   mpg  hp   
Mazda RX4         21.0 110   
Mazda RX4 Wag     21.0 110   
Datsun 710        22.8  93   
                   ............

# Data Frame Row Slice

We retrieve rows from a data frame with the single square bracket operator, just like what we did with columns. However, in additional to an index vector of row positions, we append an extra comma character. This is important, as the extra comma signals a wildcard match for the second coordinate for column positions.

#### Numeric Indexing

For example, the following retrieves a row record of the built-in data set [mtcars](http://www.r-tutor.com/node/10). Please notice the extra comma in the square bracket operator, and it is *not*a typo. It states that the 1974 Camaro Z28 has a gas mileage of 13.3 miles per gallon, and an eight cylinder 245 horse power engine, ..., *etc*.

> mtcars[24,]   
            mpg cyl disp  hp drat   wt  ...   
Camaro Z28 13.3   8  350 245 3.73 3.84  ...

To retrieve more than one rows, we use a numeric index vector.

> mtcars[c(3, 24),]   
            mpg cyl disp  hp drat   wt  ...   
Datsun 710 22.8   4  108  93 3.85 2.32  ...   
Camaro Z28 13.3   8  350 245 3.73 3.84  ...

#### Name Indexing

We can retrieve a row by its name.

> mtcars["Camaro Z28",]   
            mpg cyl disp  hp drat   wt  ...   
Camaro Z28 13.3   8  350 245 3.73 3.84  ...

And we can pack the row names in an index vector in order to retrieve multiple rows.

> mtcars[c("Datsun 710", "Camaro Z28"),]   
            mpg cyl disp  hp drat   wt  ...   
Datsun 710 22.8   4  108  93 3.85 2.32  ...   
Camaro Z28 13.3   8  350 245 3.73 3.84  ...

#### Logical Indexing

Lastly, we can retrieve rows with a logical index vector. In the following vector L, the member value is TRUE if the car has automatic transmission, and FALSE if otherwise.

> L = mtcars$am == 0   
> L   
 [1]   FALSE FALSE FALSE  TRUE ...

Here is the list of vehicles with automatic transmission.

> mtcars[L,]   
                     mpg cyl  disp  hp drat    wt  ...   
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215  ...   
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440  ...   
                 ............

And here is the gas mileage data for automatic transmission.

> mtcars[L,]$mpg   
 [1] 21.4 18.7 18.1 14.3 24.4 ...

# Frequency Distribution of Qualitative Data

The **frequency distribution**of a data variable is a summary of the data occurrence in a collection of non-overlapping categories.

#### Example

In the data set [painters](http://www.r-tutor.com/node/19), the frequency distribution of the School variable is a summary of the number of painters in each school.

#### Problem

Find the frequency distribution of the painter schools in the data set painters.

#### Solution

We apply the table function to compute the frequency distribution of the School variable.

> library(MASS)                 # load the MASS package   
> school = painters$School      # the painter schools   
> school.freq = table(school)   # apply the table function

#### Answer

The frequency distribution of the schools is:

> school.freq   
school   
 A  B  C  D  E  F  G  H   
10  6  6 10  7  4  7  4

#### Enhanced Solution

We apply the cbind function to print the result in column format.

> cbind(school.freq)   
  school.freq   
A          10   
B           6   
C           6   
D          10   
E           7   
F           4   
G           7   
H           4

# Relative Frequency Distribution of Qualitative Data

The **relative frequency distribution**of a data variable is a summary of the frequency proportion in a collection of non-overlapping categories.

The relationship of frequency and relative frequency is:

Relative F requency =-Frequency-
                    Sample Size


#### Example

In the data set [painters](http://www.r-tutor.com/node/19), the relative frequency distribution of the School variable is a summary of the proportion of painters in each school.

#### Problem

Find the relative frequency distribution of the painter schools in the data set painters.

#### Solution

We first apply the table function to compute the frequency distribution of the School variable.

> library(MASS)                 # load the MASS package   
> school = painters$School      # the painter schools   
> school.freq = table(school)   # apply the table function

Then we find the sample size of painters with the nrow function, and divide the frequency distribution with it. Therefore the relative frequency distribution is:

> school.relfreq = school.freq / nrow(painters)

#### Answer

The relative frequency distribution of the schools is:

> school.relfreq   
school   
       A        B        C        D        E        F   
0.185185 0.111111 0.111111 0.185185 0.129630 0.074074   
       G        H   
0.129630 0.074074

#### Enhanced Solution

We can print with fewer digits and make it more readable by setting the digits option.

> old = options(digits=1)   
> school.relfreq   
school   
   A    B    C    D    E    F    G    H   
0.19 0.11 0.11 0.19 0.13 0.07 0.13 0.07   
> options(old)

In addition, we can apply the cbind function to print the result in column format.

> old = options(digits=1)   
> cbind(school.relfreq)   
  school.relfreq   
A           0.19   
B           0.11   
C           0.11   
D           0.19   
E           0.13   
F           0.07   
G           0.13   
H           0.07   
> options(old)    # restore the old option

# Bar Graph

A **bar graph**of a qualitative data sample consists of vertical parallel bars that shows the frequency distribution graphically.

#### Example

In the data set [painters](http://www.r-tutor.com/node/19), the bar graph of the School variable is a collection of vertical bars showing the number of painters in each school.

#### Problem

Find the bar graph of the painter schools in the data set painters.

#### Solution

We first apply the table function to compute the frequency distribution of the School variable.

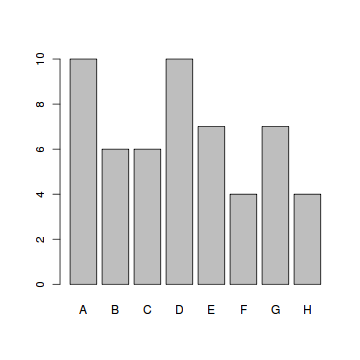
> library(MASS)                 # load the MASS package   
> school = painters$School      # the painter schools   
> school.freq = table(school)   # apply the table function

Then we apply the barplot function to produce its bar graph.

> barplot(school.freq)         # apply the barplot function

#### Answer

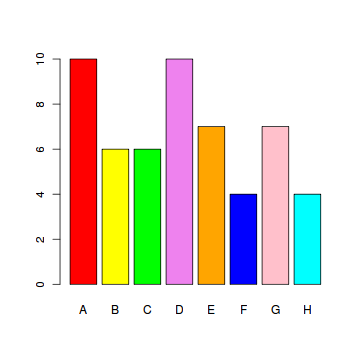
The bar graph of the school variable is:



#### Enhanced Solution

To colorize the bar graph, we select a color palette and set it in the col argument of barplot.

> colors = c("red", "yellow", "green", "violet",   
+   "orange", "blue", "pink", "cyan")   
> barplot(school.freq,         # apply the barplot function   
+   col=colors)                # set the color palette



# Category Statistics

In the built-in data set [painters](http://www.r-tutor.com/node/19), the painters are classified according to the schools they belong. Each school can be characterized by its various statistics, such as [mean](http://www.r-tutor.com/node/35) composition, drawing, coloring and expression scores.

Suppose we would like to know which school has the highest mean composition score. We would have to first find out the mean composition score of each school. The following shows how to find the mean composition score of an arbitrarily chosen school.

#### Problem

Find out the mean composition score of school C in the data set painters.

#### Solution

The solution consists of a few steps:

1. Create a logical index vector for school C.

> library(MASS)                 # load the MASS package   
> school = painters$School      # the painter schools   
> c\_school = school == "C"      # the logical index vector

1. Find the child data set of painters for school C. For explanation, please consult the tutorial of [*Data Frame Row Slice*](http://www.r-tutor.com/node/17).

> c\_painters = painters[c\_school, ]  # child data set

1. Find the mean composition score of school C.

> mean(c\_painters$Composition)   
[1] 13.167

#### Answer

The mean composition score of school C is 13.167.

#### Alternative Solution

Instead of computing the mean composition score manually for each school, use the tapplyfunction to compute them all at once.

> tapply(painters$Composition, painters$School, mean)   
     A      B      C      D      E      F      G      H   
10.400 12.167 13.167  9.100 13.571  7.250 13.857 14.000

#### Exercise

1. Find programmatically the school with the highest composition scores.
2. Find the percentage of painters whose color score is equal to or above 14.

# Scatter Plot

A **scatter plot**pairs up values of two quantitative variables in a data set and display them as geometric points inside a Cartesian diagram.

#### Example

In the data set [faithful](http://www.r-tutor.com/node/25), we pair up the eruptions and waiting values in the same observation as (*x,*y) coordinates. Then we plot the points in the Cartesian plane. Here is a preview of the eruption data value pairs with the help of the cbind function.

> duration = faithful$eruptions      # the eruption durations   
> waiting = faithful$waiting         # the waiting interval   
> head(cbind(duration, waiting))   
     duration waiting   
[1,]    3.600      79   
[2,]    1.800      54   
[3,]    3.333      74   
[4,]    2.283      62   
[5,]    4.533      85   
[6,]    2.883      55

#### Problem

Find the scatter plot of the eruption durations and waiting intervals in faithful. Does it reveal any relationship between the variables?

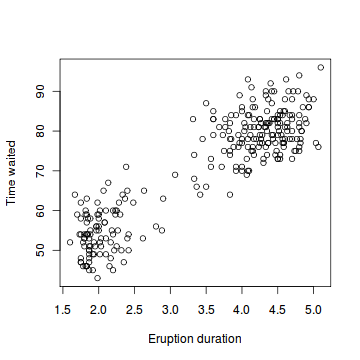
#### Solution

We apply the plot function to compute the scatter plot of eruptions and waiting.

> duration = faithful$eruptions      # the eruption durations   
> waiting = faithful$waiting         # the waiting interval   
> plot(duration, waiting,            # plot the variables   
+   xlab="Eruption duration",        # x−axis label   
+   ylab="Time waited")              # y−axis label

#### Answer

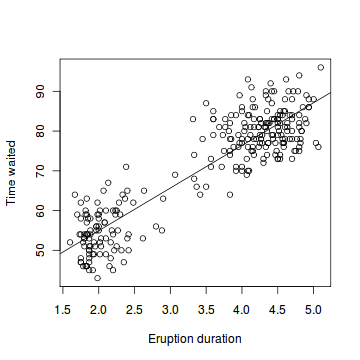
The scatter plot of the eruption durations and waiting intervals is as follows. It reveals a *positive linear relationship*between them.



#### Enhanced Solution

We can generate a linear regression model of the two variables with the lm function, and then draw a trend line with abline.

> abline(lm(waiting ~ duration))



# Quartile

There are several **quartiles**of an observation variable. The **first quartile**, or **lower** **quartile**, is the value that cuts off the first 25% of the data when it is sorted in ascending order. The **second quartile**, or **median**, is the value that cuts off the first 50%. The **third quartile**, or **upper quartile**, is the value that cuts off the first 75%.

#### Problem

Find the quartiles of the eruption durations in the data set [faithful](http://www.r-tutor.com/node/25).

#### Solution

We apply the quantile function to compute the quartiles of eruptions.

> duration = faithful$eruptions     # the eruption durations   
> quantile(duration)                # apply the quantile function   
    0%    25%    50%    75%   100%   
1.6000 2.1627 4.0000 4.4543 5.1000

#### Answer

The first, second and third quartiles of the eruption duration are 2.1627, 4.0000 and 4.4543 minutes respectively.

#### Exercise

Find the quartiles of the eruption waiting periods in faithful.

#### Note

# Percentile

The nth **percentile**of an observation variable is the value that cuts off the first *n* percent of the data values when it is sorted in ascending order.

#### Problem

Find the 32nd, 57th and 98th percentiles of the eruption durations in the data set [faithful](http://www.r-tutor.com/node/25).

#### Solution

We apply the quantile function to compute the percentiles of eruptions with the desired percentage ratios.

> duration = faithful$eruptions     # the eruption durations   
> quantile(duration, c(.32, .57, .98))   
   32%    57%    98%   
2.3952 4.1330 4.9330

#### Answer

The 32nd, 57th and 98th percentiles of the eruption duration are 2.3952, 4.1330 and 4.9330 minutes respectively.

#### Exercise

Find the 17th, 43rd, 67th and 85th percentiles of the eruption waiting periods in faithful.

#### Note

There are several algorithms for the computation of percentiles. Details can be found in the R documentation via help(quantile).

# Range

The **range**of an observation variable is the difference of its largest and smallest data values. It is a measure of how far apart the entire data spreads in value.

Range = Largest Value− Smallest Value


#### Problem

Find the range of the eruption duration in the data set [faithful](http://www.r-tutor.com/node/25).

#### Solution

We apply the max and min function to compute the largest and smallest values of eruptions, then take the difference.

> duration = faithful$eruptions     # the eruption durations   
> max(duration) − min(duration)     # apply the max and min functions   
[1] 3.5

#### Answer

The range of the eruption duration is 3.5 minutes.

#### Exercise

Find the range of the eruption waiting periods in faithful.

# Interquartile Range

The **interquartile range**of an observation variable is the difference of its upper and lower quartiles. It is a measure of how far apart the middle portion of data spreads in value.

Interquartile Range = U pper Quartile − Lower Quartile


#### Problem

Find the interquartile range of eruption duration in the data set [faithful](http://www.r-tutor.com/node/25).

#### Solution

We apply the IQR function to compute the interquartile range of eruptions.

> duration = faithful$eruptions     # the eruption durations   
> IQR(duration)                     # apply the IQR function   
[1] 2.2915

#### Answer

The interquartile range of eruption duration is 2.2915 minutes.

# Box Plot

The **box plot**of an observation variable is a graphical representation based on its quartiles, as well as its smallest and largest values. It attempts to provide a visual shape of the data distribution.

#### Problem

Find the box plot of the eruption duration in the data set [faithful](http://www.r-tutor.com/node/25).

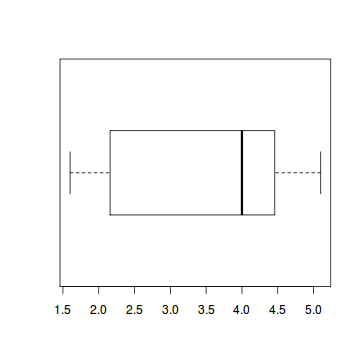
#### Solution

We apply the boxplot function to produce the box plot of eruptions.

> duration = faithful$eruptions       # the eruption durations   
> boxplot(duration, horizontal=TRUE)  # horizontal box plot

#### Answer

The box plot of the eruption duration is:



# Variance

The **variance**is a numerical measure of how the data values is dispersed around the [mean](http://www.r-tutor.com/node/35). In particular, the **sample variance**is defined as:

          n
s2 =--1--∑  (x - ¯x)2
    n - 1i=1  i


Similarly, the **population variance**is defined in terms of the population mean *μ* and population size *N*:

 2   1-∑N       2
σ  = N    (xi - μ)
       i=1


#### Problem

Find the variance of the eruption duration in the data set [faithful](http://www.r-tutor.com/node/25).

#### Solution

We apply the var function to compute the variance of eruptions.

> duration = faithful$eruptions    # the eruption durations   
> var(duration)                    # apply the var function   
[1] 1.3027

#### Answer

The variance of the eruption duration is 1.3027.

# Standard Deviation

The **standard deviation**of an observation variable is the square root of its [variance](http://www.r-tutor.com/node/42).

#### Problem

Find the standard deviation of the eruption duration in the data set [faithful](http://www.r-tutor.com/node/25).

#### Solution

We apply the sd function to compute the standard deviation of eruptions.

> duration = faithful$eruptions    # the eruption durations   
> sd(duration)                     # apply the sd function   
[1] 1.1414

#### Answer

The standard deviation of the eruption duration is 1.1414.

# Covariance

The **covariance**of two variables *x*and *y*in a data set measures how the two are linearly related. A positive covariance would indicate a positive linear relationship between the variables, and a negative covariance would indicate the opposite.

The **sample covariance**is defined in terms of the [sample means](http://www.r-tutor.com/elementary-statistics/numerical-measures/mean) as:

           n
s  = --1--∑  (x  - ¯x)(y − ¯y)
xy   n - 1 i=1 i     i


Similarly, the **population covariance**is defined in terms of the [population mean](http://www.r-tutor.com/elementary-statistics/numerical-measures/mean) *μx*, *μy* as:

     -1 N∑
σxy = N   (xi - μx)(yi − μy)
        i=1


#### Problem

Find the covariance of eruption duration and waiting time in the data set [faithful](http://www.r-tutor.com/elementary-statistics/quantitative-data). Observe if there is any linear relationship between the two variables.

#### Solution

We apply the cov function to compute the covariance of eruptions and waiting.

> duration = faithful$eruptions   # eruption durations   
> waiting = faithful$waiting      # the waiting period   
> cov(duration, waiting)          # apply the cov function   
[1] 13.978

#### Answer

The covariance of eruption duration and waiting time is about 14. It indicates a positive linear relationship between the two variables.

# Skewness

The **skewness**of a data population is defined by the following formula, where *μ*2 and *μ*3 are the second and third [central moments](http://www.r-tutor.com/elementary-statistics/numerical-measures/moment).

γ1 = μ3∕μ3∕22


Intuitively, the skewness is a measure of symmetry. As a rule, negative skewness indicates that the [mean](http://www.r-tutor.com/elementary-statistics/numerical-measures/mean) of the data values is less than the [median](http://www.r-tutor.com/elementary-statistics/numerical-measures/median), and the data distribution is *left-skewed*. Positive skewness would indicate that the mean of the data values is larger than the median, and the data distribution is *right-skewed*.

#### Problem

Find the skewness of eruption duration in the data set [faithful](http://www.r-tutor.com/elementary-statistics/quantitative-data).

#### Solution

We apply the function skewness from the e1071 package to compute the skewness coefficient of eruptions. As the package is not in the core R library, it has to be installed and loaded into the R workspace.

> library(e1071)                    # load e1071   
> duration = faithful$eruptions     # eruption durations   
> skewness(duration)                # apply the skewness function   
[1] -0.41355

#### Answer

The skewness of eruption duration is -0.41355. It indicates that the eruption duration distribution is skewed towards the left.

#### Exercise

Find the skewness of eruption waiting period in faithful.

**To remove duplicate values use unique()**