Introduction to R Programming

Edited by Ozancan Ozdemir

Contents

Introduction to R Programming

What is R?

. R is a high-level computer language and environment for statistics and graphics

. Performs a variety of simple and advanced statistical methods

. Produces high quality graphics

. R is a computer language so we can write new functions that extends R's uses

. R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand (hence the name).

. R is a free open source software maintained by several contributors. Including an R Core Team" of 17 programmers who are responsible for modifying the R source code.

. Most universities and jobs require or at least prefer that you know R

. The official R home page is http://www.R-project.org

Downloading and Installing R

To download R for the first time(to update the version, see bellows)

. Go to http://www.R-project.org

. Click on Download CRAN(CRAN: Comprehensive R Archive Network)

. Select your country (If your country is not in the list, then select a country geographically nearby yours)

. Click on Download R for Windows, select base

. Click on Download R 3.x.x for Windows, select "Save"

To install R

. Go to the folder at which the exe file is downloaded.

. Click on the .exe file, then follow the instructions.

Downloading and Installing R Studio

RStudio is an integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

To download R for the first time(to update the version, see bellows)

. Go to https://www.rstudio.com/

. Click on Download

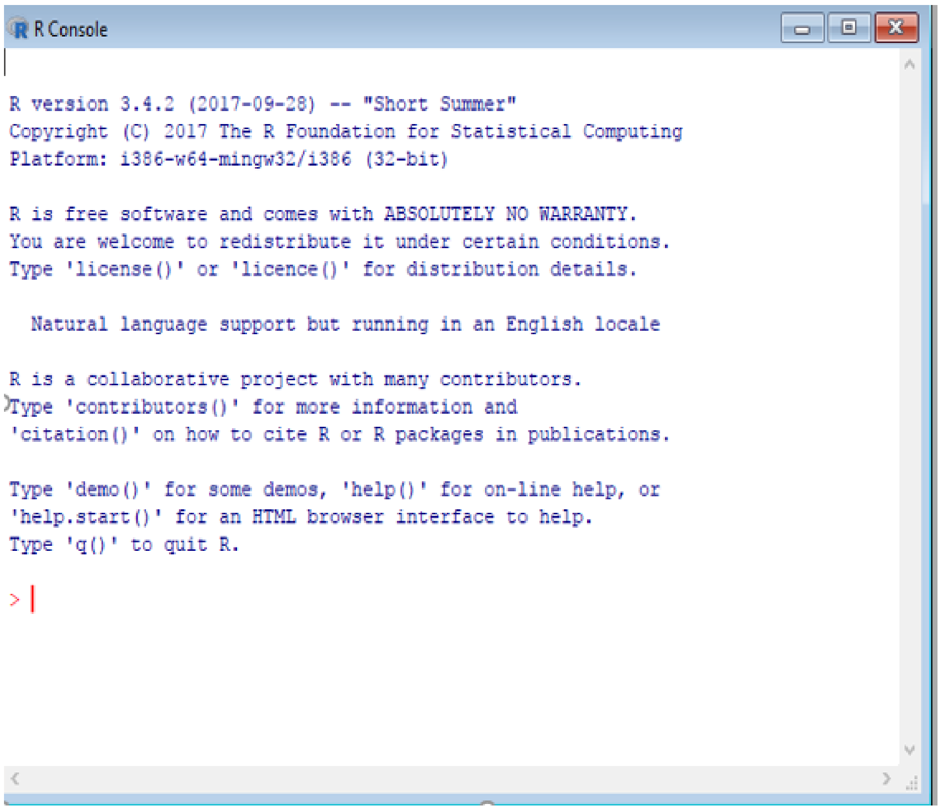
. Click on Download button under R Studio(Open Source) title

In order to use R Studio, you must install R first.

Preliminaries

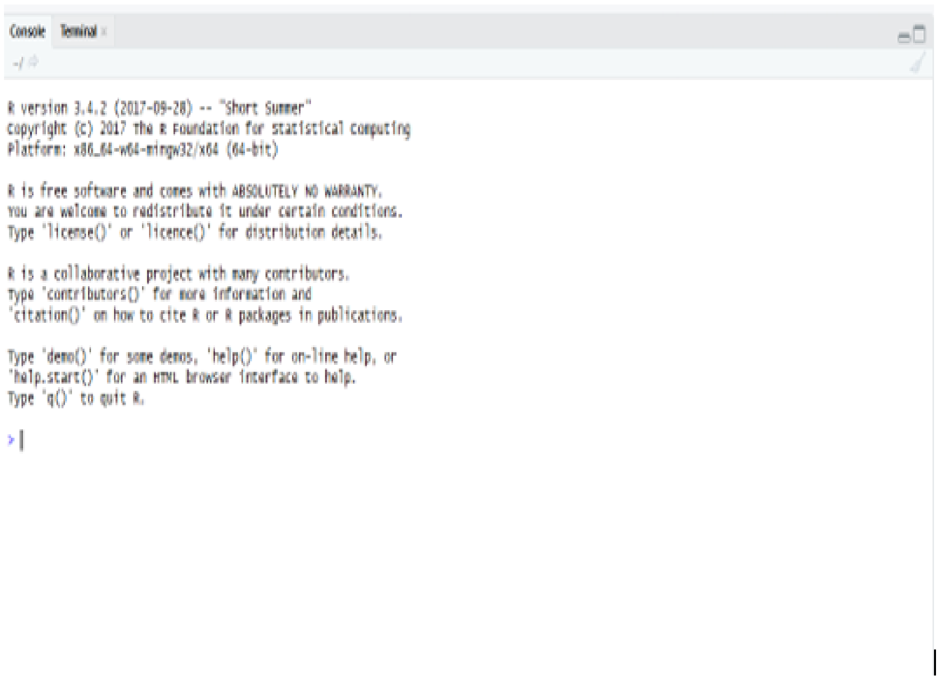
R console

That's the window where you type your commands. There is a sign on the left hand side of each line. Next to it, you type your command.



```
R version 3.4.2 (2017-09-28) -- "Short Summer"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

R Programming



```
R version 3.4.2 (2017-09-28) -- "Short Summer"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

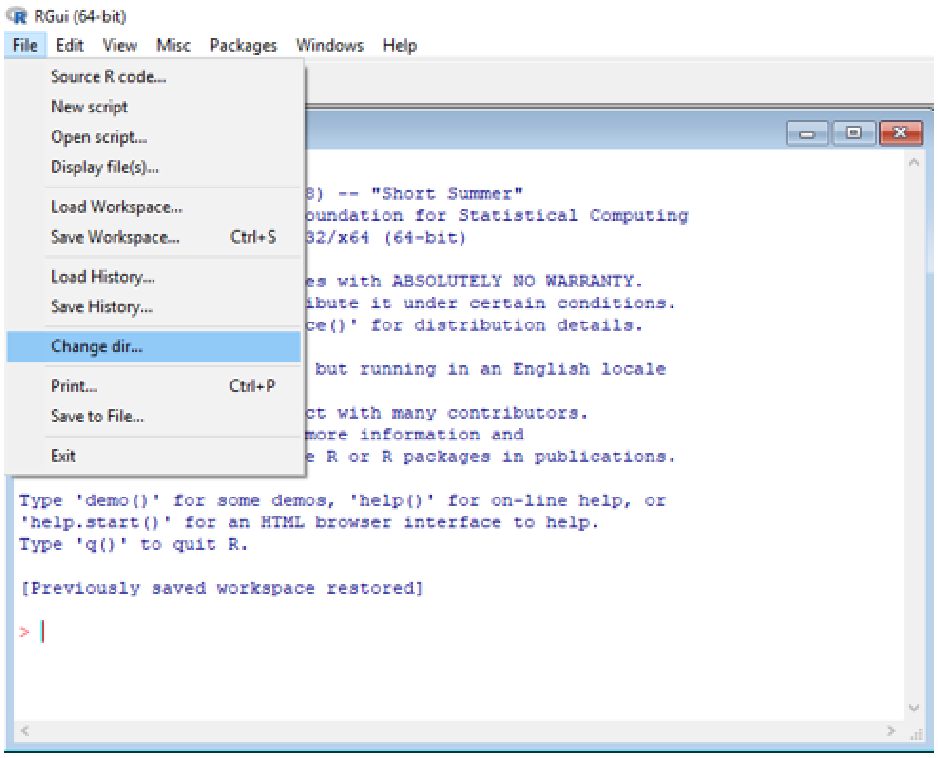
R Studio

Working Directory

Working Directory is the place that contains all your necessary files and documents e.g datasets you work on etc. There are two ways to set your working directory. First way is using `getwd` and `setwd` functions.

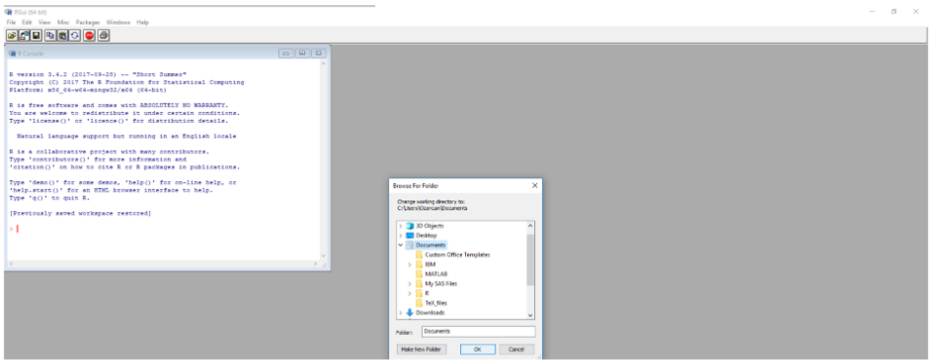
```
> getwd() #You get the working directory
[1] "C:/Users/Ozancan/Documents"
> setwd("C:/Users/Ozancan/Desktop")#you set your working directory as desktop
```

Another way is usage of menu bar. For R Programming, you can follow way shown in the following pictures.



```
8) -- "Short Summer"
oundation for Statistical Computing
32/x64 (64-bit)

es with ABSOLUTELY NO WARRANTY.
ibute it under certain conditions.
ce()' for distribution details.

 but running in an English locale

ct with many contributors.
more information and
e R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> 
```

First, select file and click on change directory option. After clicking on change directory option, the following screen is shown.



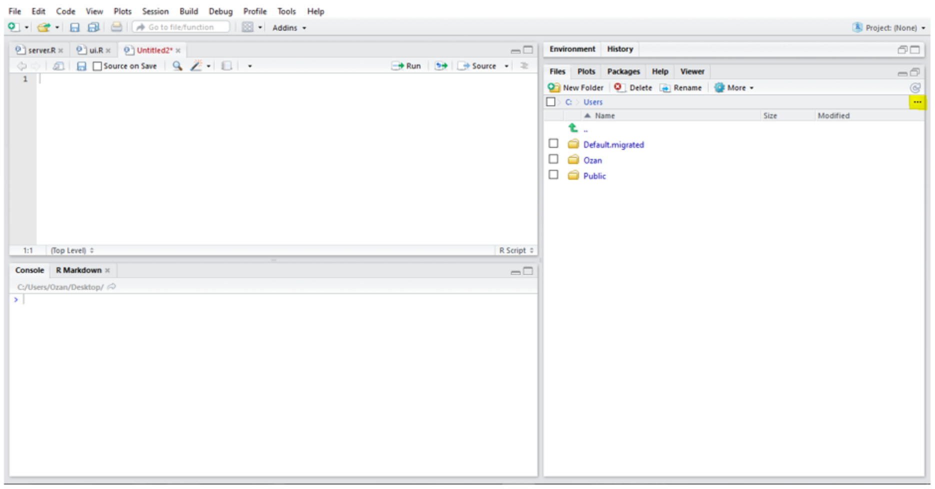
```
R version 3.4.2 (2017-09-28) -- "Short Summer"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> 
```

Change working directory to:
C:\Users\Ozancan\Documents

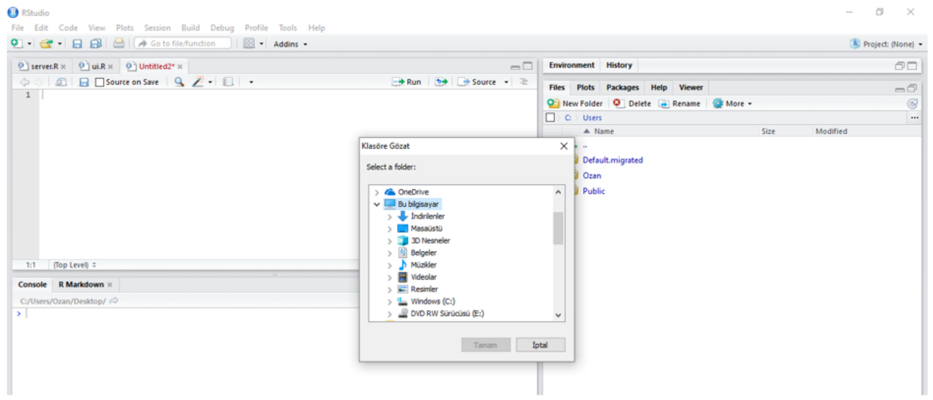
Folder: Documents

Then, you can select your working directory.

Another way for R Studio is shown in the following pictures.



First, click on the three dots marked with yellow.



Then, you can select your working directory.

Getting help and Libraries

. To get help on a specific command, e.g. plot, type on R `?plot`

. If you are not sure of the exact form of the command

Either go to help menu on R console or type on R console `??plot`

. Either way, a help window will pop up including a list of all the commands having the word plot. E.g one of them is `boot::glm.diag.plots`. This expression consists of two parts: `boot` and `glm.diag.plots`. Every command is contained in a library. The first term, `boot`, is the name of the library. The second term, `glm.diag.plots` , is the name of plot.

. To get help on `glm.diag.plots:` type on R window `library(boot)`. Then type `?glm.diag.plots`

. If the library(i.e package) is not preloaded, you will get an error message when you type `library(boot)`. To load, go to packages on the menu in the R console. You need your computer to be connected to the Internet for loading up the packages.

. Alternatively, you can use `install.packages` function. E.g

```
>install.packages('boot')
```

Basic Commands

. `ls()` to see which objects and functions are saved in the workspace

. `rm('a')` deletes the object a from the workspace

. `rm(list=ls())` deletes everything in the workspace

. `dir()` to see the contents of the folder you are in

. `q()` to exit the R console

. # you can add your explanation

. Ctrl+L deletes codes on the console window

. Ctrl+R or Ctrl+Enter run the codes on the editor

. You can repeat the previously written comands by using arrow keys.

. How to assign values to variables

```
> x<-2
```

```
> x=2
```

. What to name variables

Name should start with a letter

Can continue with letters, numbers, dot or underline characters

Case is important. That is a and A are different names

. Where to write your commands

You can write your commands on the R console window or in a simple editor such as Notepad or R script.

Basic Operators

Mathematical Operators

R and R Studio can be considered as a powerful calculator. Some basic examples are given below.

```
> 3+5
[1] 8
> 7*8
[1] 56
> 88/2
[1] 44
> 5^2 #taking the square
[1] 25
> a = 3
> b = a^2
> print(b)#used to print your variable
[1] 9
> log(15) #ln15
[1] 2.70805
> log10(1000) #log10 is logarithm base 10
[1] 3
> exp(12) #Taking the exponential power of the number
```

```
[1] 162754.8
```

Logical Operators

These operators are frequently used for data subseting, loops and writing functions.

```
>     greater than
<   less than
>=  greater than or equals
<=  less than or equals
= = equality
~ = inequality
|   Or
&     And
```

Example;

```
> 3>5
[1] F
> 3<5&6>7
[1] F
> 3<5|6>7
[1] TRUE
```

Creating a sequence in R

A sequence is an enumerated collection of objects in which repetitions are allowed. Like a set, it contains members The number of elements (possibly infinite) is called the length of the sequence.

. To form a sequence use the command `seq` function or ':'. e.g

```
> a=1:20
> a
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> a=seq(5,25)
> a
 [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
> z=seq(1,20,2) #the last number in paranthesis shows the amount of increment
> z
 [1]  1  3  5  7  9 11 13 15 17 19

.   To repeat the sequence of numbers use the command rep. e.g
> rep(z,2)
 [1]  1  3  5  7  9 11 13 15 17 19  1  3  5  7  9 11 13 15 17 19
> rep(seq(1,9,3),4)
 [1] 1 4 7 1 4 7 1 4 7 1 4
> rep(seq(1,9,3),each=4)
 [1] 1 1 1 1 4 4 4 4 7 7 7 7
```

Arrays and Matrices

An array object (or simply array) contains a collection of elements of the same type, each of which is indexed (i.e., identified) by a number.

A matrix is a collection of elements of the same type, organized in the form of a table. Each element is indexed by a pair of numbers that identify the row and the column of the element.

. To construct an array in R and R Studio we use `c` (combine) function. For example, define an object named v

```
> v=c(1.2,3.5,.79,25)
> v
[1]  1.20  3.50  0.79 25.00
```

. To extract the rth entry of the vector use v[r] where r is any number

```
> v[2]
[1] 3.5
> v[c(1,2)]# to get more than one entry from vector
[1] 1.2 3.5
```

. `length` function shows the length of an object.

```
> length(v)
[1] 4
```

. The logical operators are used to extract elements from array based on some criteria. e.g

```
> v=c(1.2,3.5,.79,25)
> v[v>2.3]
[1]  3.5 25.0
> v[v>20 & v<30]
[1] 25
> v[v==1.2 | v<1]
[1] 1.20 0.79
```

. By using c (combine function), you can create an array that contains only characters.

```
> odtu=c("odtu","1956","yilinda","kuruldu")
> odtu
[1] "odtu"    "1956"    "yilinda" "kuruldu"
```

. To extract entry from such an array

```
> odtu[odtu=="yilinda"]
[1] "yilinda"
```

. You can give names to elements in R by using names command.

```
> names(v)=c("1stentry","2ndentry","3rdentry","4thentry")
> v
1stentry 2ndentry 3rdentry 4thentry
    1.20     3.50     0.79    25.00
.   To extract elements from arrays with element names.
> v["1stentry"]
1stentry
     1.2
> v[c("1stentry","2ndentry")]
1stentry 2ndentry
     1.2      3.5
```

. To construct a matrix of size I by J and define as an object and called mat.

```
> mat=matrix(c(1,2,3,4,5,6),ncol=2)
> mat
     [,1] [,2]
```

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

. Another way to construct a matrix;

```
> mat2=matrix(c(1,2,3,4,5,6),3,2)#3 represents number of row, 2 represents   number of column
> mat2
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

. To extract the e.g 2,1 th entry of matrix of mat use mat[2,1]

```
> mat[2,1]
[1] 2
```

. We can assign names to rows and columns of a matrix. e.g Consider the matrix defined above.

```
> rownames(mat)=c("istanbul","beyoglu","taksim")
> colnames(mat)=c("galata","saray")
> mat
         galata saray
istanbul      1     4
beyoglu       2     5
taksim        3     6
> mat["istanbul","galata"]
[1] 1
```

. You can find out the dimension of any matrix with dim command.

```
> mat
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> dim(mat)#dimension of the matrix
[1] 3 2
```

. If you wish to obtain only row number of any matrix, you have to use nrow command. On the other hand, if you want to see only column number of any matrix, it is enough to type ncol command.

```
> nrow(mat)#number of rows for mat that is the name of matrix you defined.
[1] 3
> ncol(mat)#number of columns for mat that is the name of matrix you defined.
[1] 2
```

More details about matrices

```
> A <- matrix(c( 6, 1,+ 0, -3,-1, 2),3, 2, byrow = TRUE)
> B <- matrix(c( 4, 2,0, 1,-5, -1),3, 2, byrow = TRUE)
> A
     [,1] [,2]
[1,]    6    1
[2,]    0   -3
[3,]   -1    2
```

```
> B
     [,1] [,2]
[1,]    4    2
[2,]    0    1
[3,]   -5   -1

> A + B #summation of two matrices
     [,1] [,2]
[1,]   10    3
[2,]    0   -2
[3,]   -6    1
> A - B #subtraction of two matrices
     [,1] [,2]
[1,]    2   -1
[2,]    0   -4
[3,]    4    3


> A * B # this is component-by-component multiplication, not matrix multiplication
     [,1] [,2]
[1,]   24    2
[2,]    0   -3
[3,]    5   -2
> t(A) #take the transpose of the matrix
     [,1] [,2] [,3]
[1,]    6    0   -1
[2,]    1   -3    2
> C<-matrix(c(2,4,5,6),nrow=2)
> C
     [,1] [,2]
[1,]    2    5
[2,]    4    6
> A%*%C #the matrix multiplication
     [,1] [,2]
[1,]   16   36
[2,]  -12  -18
[3,]    6    7
> solve (C) #take the inverse of the matrix
      [,1]   [,2]
[1,] -0.75  0.625
[2,]  0.50 -0.250
```

Adding/Deleting Elements of Vectors and Matrices

Technically, vectors and matrices are of fixed length and dimensions. However, they can be reassigned.

```
> x <- c(12,5,13,16,8)
> x <- c(x,20) # append 20
> x
[1] 12  5 13 16  8 20
> x <- c(x[1:3],20,x[4:6]) # insert 20
> x
[1] 12  5 13 20 16  8 20
> x <- x[-2:-4]# delete elements 2 through 4
> x
[1] 12 16  8 20
```

. The `rbind()` and `cbind()` functions enable one to add rows or columns to a matrix.

```
> one=c(1,1,1,1)
> one
[1] 1 1 1 1
> z=matrix(c(1,2,3,4,1,1,0,0,1,0,1,0),ncol=3)
> z
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    1    0
[3,]    3    0    1
[4,]    4    0    0
> cbind(one,z) #"combine vector and matrix as column
     one
[1,]   1 1 1 1
[2,]   1 2 1 0
[3,]   1 3 0 1
[4,]   1 4 0 0
> q=rbind(c(1,2),c(3,4))#combine vector and matrix as row
> q
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

. To delete entry from matrices or vectors - (minus) sign is used.

```
> z=matrix(c(1,2,3,4,1,1,0,0,1,0,1,0),ncol=3)
> z
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    1    0
[3,]    3    0    1
[4,]    4    0    0
> z[-1,] #deleting first row
     [,1] [,2] [,3]
[1,]    2    1    0
[2,]    3    0    1
[3,]    4    0    0
> z[-c(1,2),]#deleting first and second row. To delete more than one row,
# use c (combine) command
     [,1] [,2] [,3]
[1,]    3    0    1
[2,]    4    0    0
> z[,-1]#deleting first column
     [,1] [,2]
[1,]    1    1
[2,]    1    0
[3,]    0    1
[4,]    0    0
> z[,-c(1,2)]#deleting first and second column. To delete more than one row, use
# c (combine) command
[1] 1 0 1 0
```

Swirl library in R - Learn R in R

The swirl R package makes it fun and easy to learn R programming and data science. If you are new to R, have no fear. On this page, we'll walk you through each of the steps required to begin using swirl. In order to run swirl, you must have R 3.1.0 or later installed on your computer. Step 1: Get R and R Studio Step 2: Install Swirl:

```
> install.packages("swirl")
```

Step 3: Start Swirl:

```
>library(swirl)#call the swirl

>swirl() #start swirl
```

Step 4: Install an interactive course

The first time you start swirl, you'll be prompted to install a course. You can either install one of the recommended courses or visit http://swirlstats.com/students.html for more options. There are even more courses available from the Swirl Course Network.
If you'd like to install a course that is not part of our course repository, type ?InstallCourses at the R prompt for a list of functions that will help you do so.

Reading Data Sets in R

Before we move on and discover how to load your data into R, it might be useful to go over following checklist that will make it easier to import the data correctly into R:

. If you work with spreadsheets, the first row is usually reserved for the header, while the first column is used to identify the sampling unit;

. Avoid names, values or fields with blank spaces, otherwise each word will be interpreted as a separate variable, resulting in errors that are related to the number of elements per line in your data set;

. If you want to concatenate words, inserting a . in between to words instead of a space;

. Short names are preferred over longer names;

. Try to avoid using names that contain symbols such as ?, $,%, ^, &, *, (,),-,#, ?„,<,>, /, |, , [,] ,{, and };

. Delete any comments that you have made in your Excel file to avoid extra columns or NA's to be added to your file; and make sure that any missing values in your data set are indicated with NA.

Preparing your R workspace

In general, when you read the data set, your data file ***MUST BE*** in your working directory. If you remember how to do it, OK. Otherwise, please look at the previous recitation or reminder at the beginning of the recitation.

Loading data set being available in R

R and R Studio includes many data sets. In order to load such a data set, you have to type the name of data set on R console. To see the list of the data sets, you can visit the following page.

https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html

```
> cars
> AirPassengers
> mtcar
```

Read TXT files with read.table()

If you have a .txt or a tab-delimited text file, you can easily import it with the basic R function read.table().

. If the variables in your data set have column names.

```
> read.table("drug.txt",header=T)
  drug  math
1 1.17 78.93
2 2.97 58.20
3 3.26 67.47
4 4.69 37.47
5 5.83 45.65
6 6.00 32.92
7 6.41 29.97
```

. If the variables in your data set do not have column names.

```
> read.table("drug1.txt",header=F)
    V1    V2
1 1.17 78.93
2 2.97 58.20
3 3.26 67.47
4 4.69 37.47
5 5.83 45.65
6 6.00 32.92
7 6.41 29.97
```

Read CSV Excel Files into R

If you have a file that separates the values with a , or ; you usually are dealing with a .csv file. To successfully load this file into R, you can

use the read.table() function in which you specify the separator character, or you can use the `read.csv()` or `read.csv2()` functions. The

former function is used if the separator is a „ the latter if ; is used to separate the values in your data file.

Remember that the `read.csv()` as well as the `read.csv2()` function are almost identical to the `read.table()` function.

```
> yates1=read.table("yates.csv",sep=",",header=T)
> head(yates1) #shows some first observation of the data set, you can use it any time
  Block Trt Yield
1     1   2 51.50
2     1   7 36.75
3     1  11 21.00
4     1  15 41.56
5     1  19 22.25
6     2   1 50.75
```

```
> yates2=read.csv("yates.csv",header=T)
> head(yates2)
```

```
  Block Trt Yield
1     1   2 51.50
2     1   7 36.75
3     1  11 21.00
4     1  15 41.56
5     1  19 22.25
6     2   1 50.75
```

`read.delim()` for Delimited Files

The `read.delim` function is typically used to read in delimited text files, where data is organized in a data matrix with rows representing cases and columns representing variables.

```
> d=read.delim("annual.txt", header=TRUE, sep="\t")
> head(d)
  month avgHigh season schoolIn
1   Jan      38 Winter      yes
2   Feb      41 Winter      yes
3   Mar      47 Spring      yes
4   Apr      56 Spring      yes
5   May      69 Spring      yes
6   Jun      81 Summer       no
```

`sep="\t"` tells R that the file is tab-delimited (use `" "` for space delimited and “,” for comma delimited.

In addition to them, there are many type of data set and there are many ways to read these data set into R and R Studio. You can learn these by searching on the internet.

Some Built-in Functions

There are many built-in functions in R. Here are some of them. Assume we have the following x vector.

```
> x=c(1,9,0,5,1,9,5,6)
> which(x==6)#shows the location of 6 in the vector of x
[1] 8
> which.max(x)#shows the location of the maximum element of vector of x
[1] 2
> length(x) #to see the length of the vector
[1] 8
> max(x) #maximum value of a vector
[1] 9
> min(x) #minimum value of a vector
[1] 0
> range(x)#minimum and maximum values of vectors
[1] 0 9
> sum(x)#the summation of all elements of a vector
[1] 36
> cumsum(x)#cumulative sum of vector
[1]  1 10 10 15 16 25 30 36
> mean(x)#mean of the vector
[1] 4.5
> median(x)#median of the vector
[1] 5
> var(x)#variance of vector
```

```
[1] 12.57143
> sd(x)#standard deviation of the vector
[1] 3.545621
> sort(x)#sort in increasing order
[1] 0 1 1 5 5 6 9 9
> sort(x,decreasing = T)#sort in decreasing order
[1] 9 9 6 5 5 1 1 0
> diff(x)#take the difference of ith and (i+1)th element
[1]  8 -9  5 -4  8 -4  1
If we have a matrix,
> x=matrix(c(2,4,6,1,2,3,3,6,9),3,3)
> x
     [,1] [,2] [,3]
[1,]    2    1    3
[2,]    4    2    6
[3,]    6    3    9
> which(x==6)#shows the location of 6 in the matrix as vector x
[1] 3 8
> which.max(x)#shows the location of the maximum element of the matrix as vector x
[1] 9
> length(x) #to see the length of the matrix as vector
[1] 9
> max(x) #maximum value of the matrix as vector
[1] 9
> min(x) #minimum value of the matrix as vector
[1] 1
> range(x)#minimum and maximum values of the matrix as vector
[1] 1 9
> sum(x)#the summation of all elements of the matrix as vector
[1] 36
> cumsum(x)#cumulative sum of the matrix as vector
[1]  2  6 12 13 15 18 21 27 36
> mean(x)#mean of the matrix as vector
[1] 4
> median(x)#median of the matrix as vector
[1] 3
> sd(x)#standard deviation of the matrix as vector
[1] 2.54951
> sort(x)#sort in increasing the matrix as vector
[1] 1 2 2 3 3 4 6 6 9
> sort(x,decreasing = T)#sort in decreasing order
[1] 9 6 6 4 3 3 2 2 1
> diff(x)#take the difference of i,j th and (i+1,j)th element of the matrix
     [,1] [,2] [,3]
[1,]    2    1    3
[2,]    2    1    3
```

Data Frame

A data frame in R combines features of vectors, matrices, and lists. Like vectors, data frames must have the same kind of data in each column. Like matrices, data frames have both rows and columns. And like lists, data frames allow the user to have a combination of numeric, character, and logical data. You can think of a data frame in the same way you would think of a data set in a statistics program or a worksheet in Excel or some other spreadsheet program.

Creating A Data Frame from Vectors

```
people <-c("Kim","Bob","Ted","Sue","Liz","Amanada","Tricia","Johnathan","Luis","Isabel")
gender<-c("m","m","m","f","f","f","f","f","m","f")
scores <-c(17,19,24,25,16,15,23,24,29,17)
quiz_scores <- data.frame(people,gender,scores) #to create a data frame
quiz_scores
```

```
##       people gender scores
## 1        Kim      m     17
## 2        Bob      m     19
## 3        Ted      m     24
## 4        Sue      f     25
## 5        Liz      f     16
## 6    Amanada      f     15
## 7     Tricia      f     23
## 8  Johnathan      f     24
## 9       Luis      m     29
## 10    Isabel      f     17
```

We can obtain individual columns by using the column index in square brackets. We can also employ the data frame name followed by a $ sign and the column name.

```
quiz_scores[2]
```

```
##    gender
## 1       m
## 2       m
## 3       m
## 4       f
## 5       f
## 6       f
## 7       f
## 8       f
## 9       m
## 10      f
```

```
quiz_scores$scores #mostly used version
```

```
##  [1] 17 19 24 25 16 15 23 24 29 17
```

```
attach(quiz_scores)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     gender, people, scores
```

```
scores
```

```
##  [1] 17 19 24 25 16 15 23 24 29 17
```

Changing class of the object

Suppose you read your data set into R or you have an object created in R, and you wish to change the class of it.

```
x=1:5

y=6:10

z=c(x,y)

class(z)#to find out the class of the object
```

```
## [1] "integer"
```

```
z=as.data.frame(z)

class(z)
```

```
## [1] "data.frame"
```

```
z1=as.matrix(z)

class(z1)
```

```
## [1] "matrix"
```

Accessing data from Data Frame

```
quiz_scores$people
```

```
##  [1] Kim       Bob       Ted       Sue       Liz       Amanada   Tricia
##  [8] Johnathan Luis      Isabel
## Levels: Amanada Bob Isabel Johnathan Kim Liz Luis Sue Ted Tricia
```

```
 quiz_scores$scores>15
```

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

```
quiz_scores$scores[quiz_scores$scores>15] #to see grades greater than 15
```

```
## [1] 17 19 24 25 16 23 24 29 17
```

```
quiz_scores[quiz_scores[,"scores"]>15,] #to see grades greater than 15 and corresponding other variable
```

```
##       people gender scores
## 1        Kim      m     17
## 2        Bob      m     19
## 3        Ted      m     24
## 4        Sue      f     25
## 5        Liz      f     16
## 7     Tricia      f     23
## 8  Johnathan      f     24
## 9       Luis      m     29
## 10    Isabel      f     17
```

```
quiz_scores$scores[quiz_scores$gender=="f"]#to see grades for female students
```

```
## [1] 25 16 15 23 24 17
```

```
 quiz_scores[quiz_scores[,"gender"]=="f",]#to see grades for female students and corresponding other va
```

```
##       people gender scores
## 4        Sue      f     25
## 5        Liz      f     16
## 6    Amanada      f     15
## 7     Tricia      f     23
## 8  Johnathan      f     24
## 10    Isabel      f     17
```

Data Subsetting

Data subsetting is the first, but one of the most important part of the data analysis. Assume that we read the dataset that contains some demographic information related to some nations in the world.

```
> data=read.table("data.txt",header=T)
```

```
> head(data)
```

```
      nation birth_rate pci  pop mortality_rate
1  Venezuela       46.4 392 0.40           68.5
2     Mexico       45.7 118 0.61           87.8
3    Ecuador       45.3  44 0.53          115.8
4   Colombia       38.6 158 0.53          106.8
5     Ceylon       37.2  81 0.53           71.6
6 PuertoRico       35.0 374 0.37           60.2
```

```
> class(data) #check the class of object
[1] "data.frame"
> dim(data)
[1] 29  5
```

. Assume we want to construct a new matrix with the first 10 rows of data

```
data[1:10,]
```

. Assume we want to construct a new matrix with the 1 st , 3 nd ,12 th , and 15 th rows of data

```
data[c(1,3,12,15),]
```

. Assume we want to construct a new matrix same as data but without the 4th row of data

```
data[-4,]
```

. Assume we want to construct a new matrix same as data but without the 3rd,5th,12nd and 27th row of the data

. Assume we want to construct a new matrix same as data but without pci information

```
data[,-3]
```

. Assume we want to construct a new matrix same as data but without pci and mortality rate information

```
data[,-c(3,5)]
```

. Assume we want to confine the analysis only to birth rate which are at least 25

`data[data$birth_rate>=25,] OR data[data[,2]>=25,]`

. Assume we want to include only the nations whose mortality rate given is between 80 and 120

`data[data$mortality_rate>=80&data$mortality_rate<=120,]`

or

`data[data[,5]>=80&data[,5]<=120,]`

. Assume we want to include only the nations whose population per income equals to either 0.53 or 0.3

Graphics in R

Graphics is a great strength of R. The graphics package is part of the standard distribution and contains many useful functions for creating a variety of graphic displays.

Scatter Plot

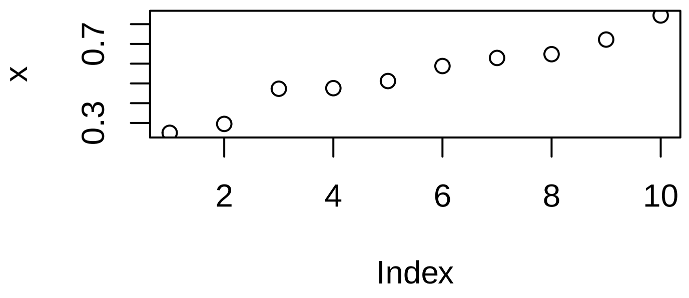
Scatter plots can help you identify the relationship between two data samples. A scatter plot is a simple plot of one variable against another. In order to draw a scatter plot, the following command is used.

```
> plot(x)#draw the scatter plot of random variable x
```

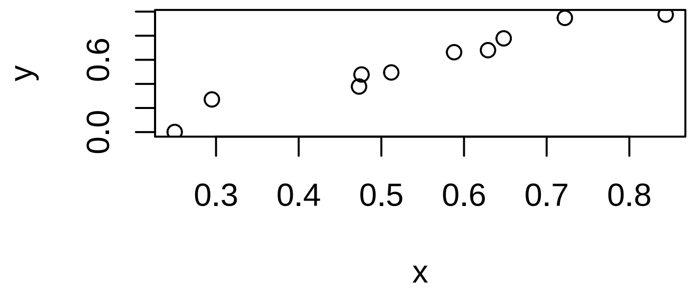
```
> plot(x,y)##draw the scatter plot of random variables x and y
```

Here is the basic example of scatter plot.

```r
x=c(0.25, 0.295, 0.473, 0.476, 0.512,0.588, 0.629, 0.648, 0.722, 0.844)
y=c(0.00102, 0.271, 0.378, 0.478, 0.495, 0.663, 0.68, 0.778, 0.948, 0.975)
plot(x)#Draw scatter plot of x
```

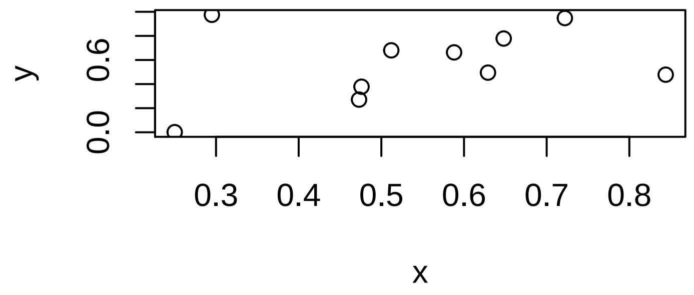


```r
plot(x,y)#Draw scatter plot of x and y
```

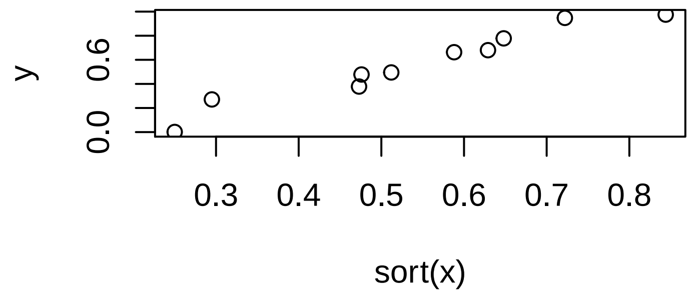


If you draw a scatter or line plot, note that your x variable should be arranged in an order. In previous example, the elements in the vector x is arranged in an order. If they are not ordered, it is possible to use sort command to obtain an ordered vector.

```
x=c(0.25, 0.473, 0.476, 0.844 ,0.629, 0.588, 0.512, 0.648, 0.722,0.295)
y=c(0.00102, 0.271, 0.378, 0.478, 0.495, 0.663, 0.68, 0.778, 0.948, 0.975)
plot(x,y) #if you do not use sort
```



```
plot(sort(x),y) # if you use sort
```



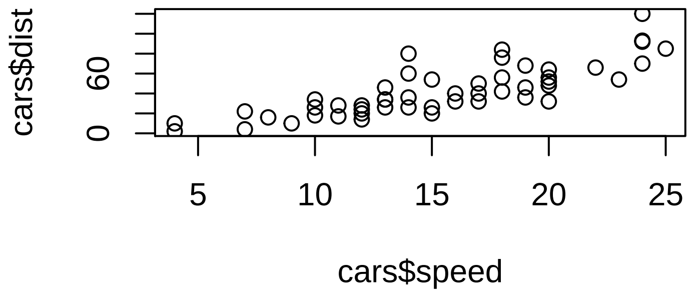
Adding Title and Labels and Other Manupulations

R enables us to add title, axis name to our plot. Also, we can arrange the color of items in the plot with appropriate commands. To show an example, cars dataset in R is used.

```
head(cars)
```

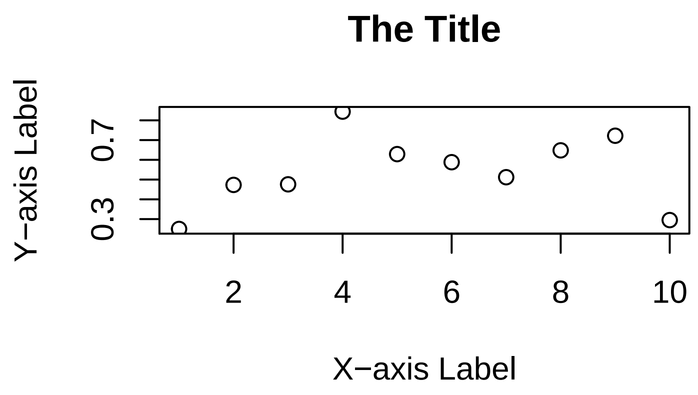
```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```
plot(cars$speed,cars$dist)
```

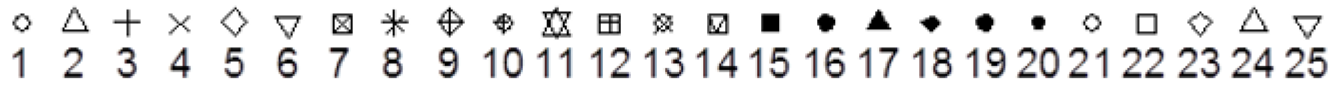


```
> #main command is used to add title into your plot
> #xlab command is used to change the name of x label
> #ylab command is used to change the name of y label
```

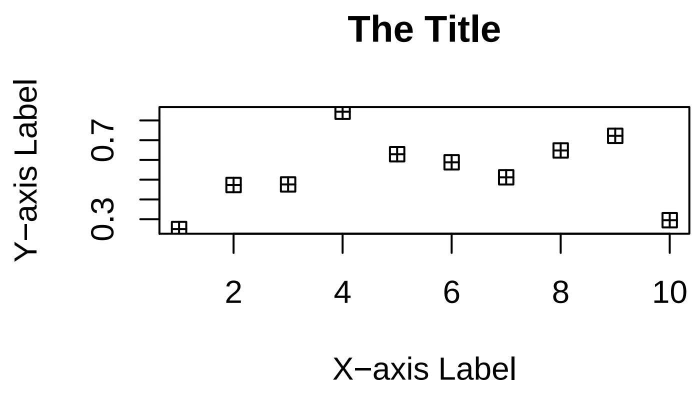
```
plot(x, main="The Title", xlab="X-axis Label", ylab="Y-axis Label")
```



Changing Plot Character



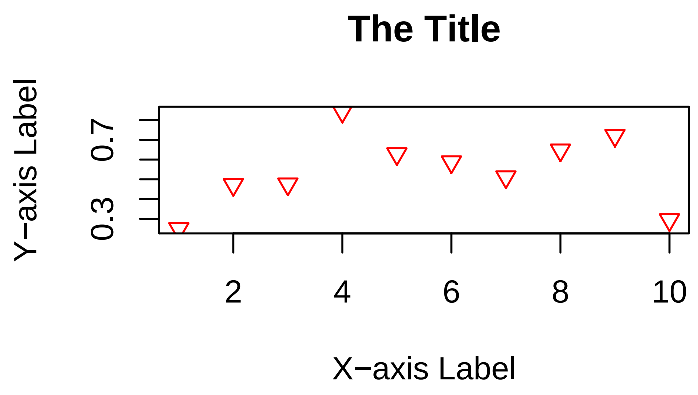
```
plot(x, main="The Title", xlab="X-axis Label", ylab="Y-axis Label",pch=12)
```



Changing the color of points or figures in the plot

. In order to see the color index, you can visit, https://www.statmethods.net/advgraphs/parameters.html

```
plot(x, main="The Title", xlab="X-axis Label", ylab="Y-axis Label",pch=6,  col=18)#col command help us
```



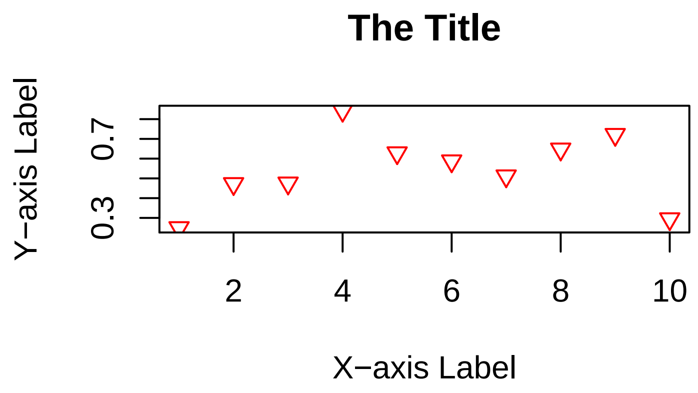
```
grid(0,0.8) #to add grid
```

Add Lines to Your Plot

```
abline(a = NULL, b = NULL, h = NULL, v = NULL...)
a, b    the intercept and slope, single values.
h   the y-value(s) for horizontal line(s).
v   the x-value(s) for vertical line(s).
```

```
abline(h=0.6,v=4)
```

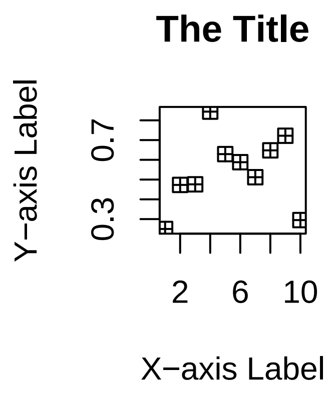
```
plot(x, main="The Title", xlab="X-axis Label", ylab="Y-axis Label",pch=6,  col=18)#col command help us
```



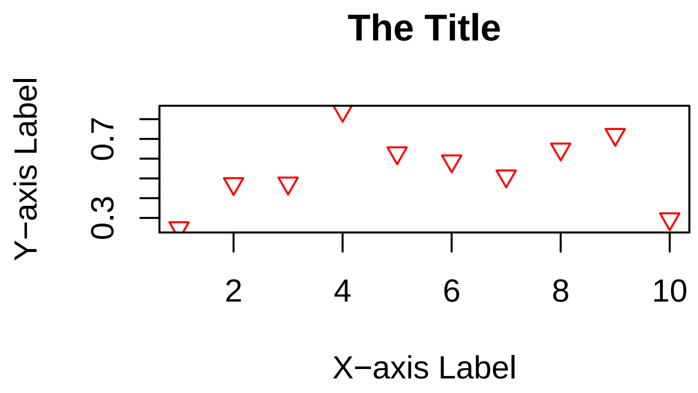
```
lines(y) #you can add y variables as line in the plot
```

Showing more than one plot in one window

```
par(mfrow=c(1,2)) #where c(nofrow,nofcolumn)
plot(x, main="The Title", xlab="X-axis Label", ylab="Y-axis Label",pch=12)
```



```
plot(x, main="The Title", xlab="X-axis Label", ylab="Y-axis Label",pch=6,col=18)
```



```
#col command help us to arrange the color of the points
#after using par() command, when you draw a new plot please do not forget to close  your window
```

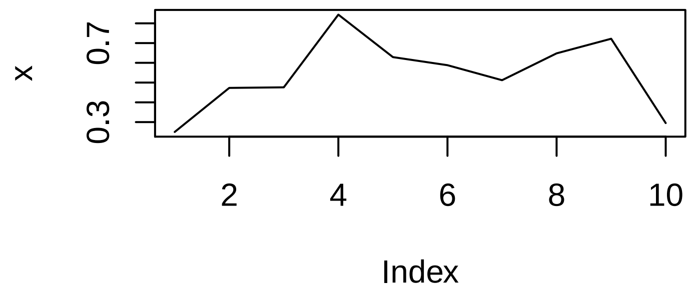
Line plots

Line graph is used to look at the changes in variable over time or look at the relationship between two variable. In both cases, x axis corresponds to the independent variable(time, days etc), y axis corresponds to the dependent variable(temperature, income etc)

```
x
```

```
##  [1] 0.250 0.473 0.476 0.844 0.629 0.588 0.512 0.648 0.722 0.295
```

```
plot(x,type="l") #type command specifies the type of your plot
```



```
#plot command helps you to draw a line plot
```

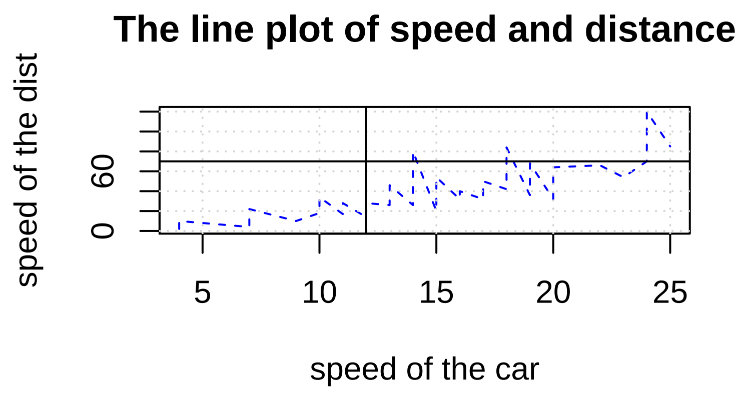
In order to add label names, titles and change the line color, you can use the previous commands which are xlab, ylab, main etc.

```
head(cars)
```

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

```
attach(cars) #in order to not to use $ sign
```

```
plot(speed,dist,type="l",main="The line plot of speed and distance",xlab="speed of the car",ylab="speed
#main->add title
 #xlab and ylab -> add label name
#col-> change the color of the line
#More examples on lty:
#lty= "solid"  or lty=1
#lty= "dashed"  or lty=2
#lty= "dotted"  or lty=3
#lty= "dotdash"  or lty=4
#lty= "longdash"  or lty=5
#lty= "twodash"  or lty=6
grid()#you can add grid
abline(h=70,v=12)
```



Bar Plot

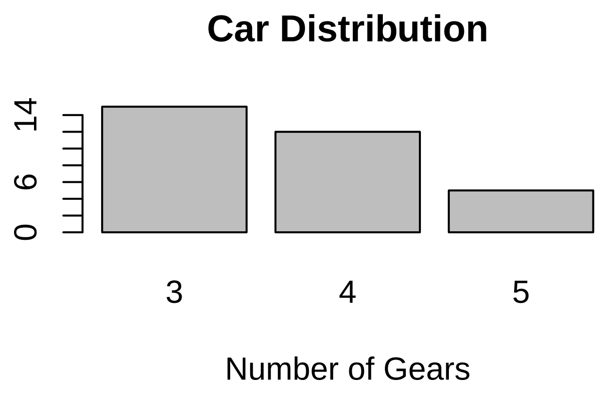
A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The `barplot` function produces a simple bar chart. It assumes that the heights of your bars are conveniently stored in a vector.

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

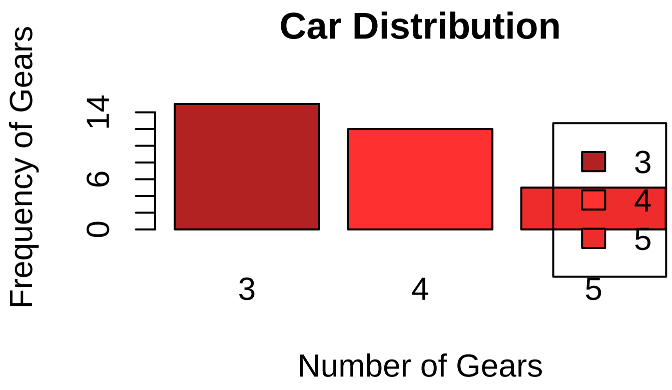
```
counts <- table(mtcars$gear)#table commands helps you to create the  frequency table
```

```
barplot(counts, main="Car Distribution", xlab="Number of Gears")
```



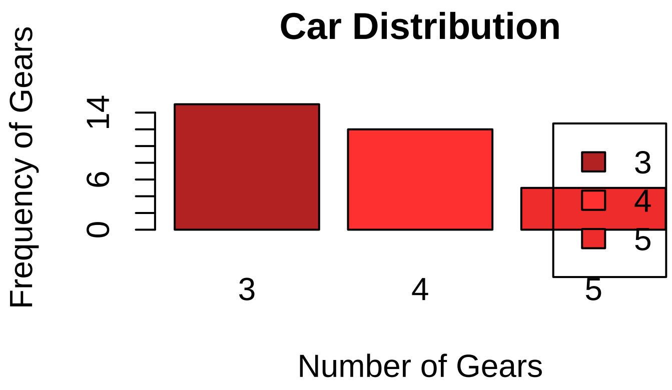
. You can see also list of colors in R from this pdf file. http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf

```
barplot(counts, main="Car Distribution", xlab="Number of Gears",ylab="Frequency of Gears",col=c("firebr
```



```
#legend command helps you to add small info box
#also we can use legend=c("3","4","5")
#to color your bars, you can use col command
```

```
barplot(counts, main="Car Distribution", xlab="Number of Gears",ylab="Frequency of Gears",col=c("firebr
```



Histogram

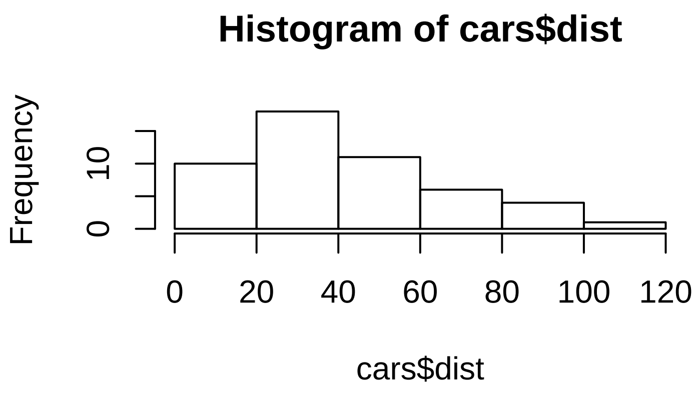
Quantitative variables often take so many values that a graph of the distribution is clearer if nearby values are group together. The most common graph of the distribution of one quantitative variable is a histogram. ***(Used for continious type of data)***

```
> hist(x)
```

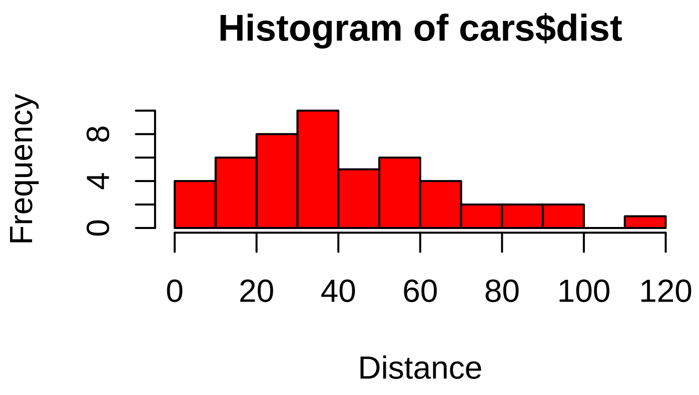
```
#histogram
#simple histogram
head(cars)
```

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

```
hist(cars$dist)
```

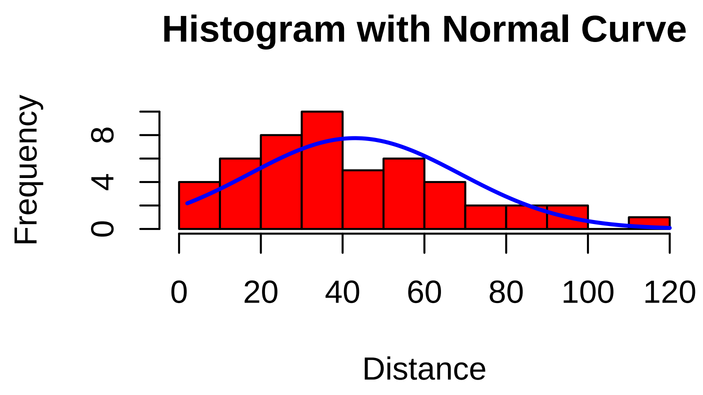


```
# Colored Histogram with Different Number of Bins
hist(cars$dist, breaks=12, col="red",xlab="Distance") #breaks used to arrange number of bin
```



Adding Normal Curve to Histogram

```
# Adding a Normal Curve
x <- cars$dist
h<-hist(x, breaks=10, col="red", xlab="Distance",  main="Histogram with Normal Curve")
xfit<-seq(min(x),max(x),length=150)
yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))  #dnorm->density of normal dist
yfit <- yfit*diff(h$mids[1:2])*length(x) #mids ->the n cell midpoints.
lines(xfit, yfit, col="blue", lwd=2)
```



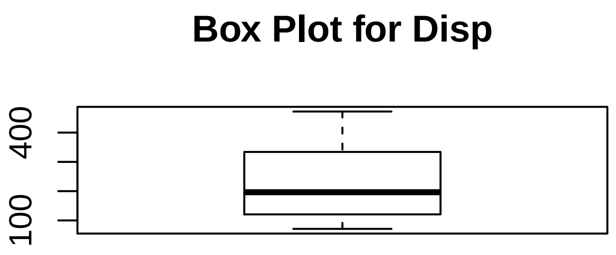
Box Plot

We use five number summary which are minimum, 1st quartile, median, 3rd quartile and maximum values of data to draw a box plot. On each box, the central mark indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points not considered outliers, and the outliers are plotted individually using the 'o' symbol.

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

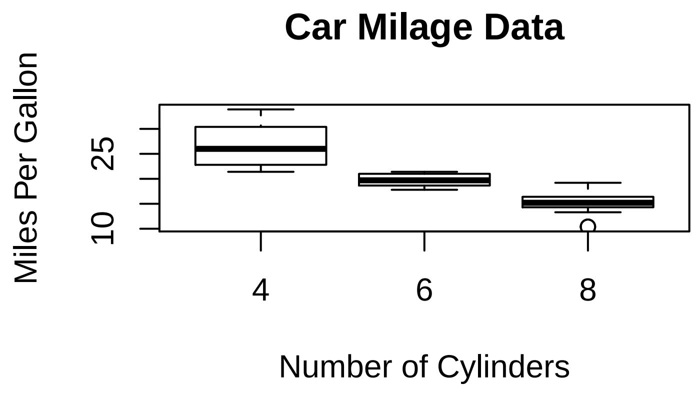
```
boxplot(mtcars$disp,data=mtcars,main="Box Plot for Disp")
```



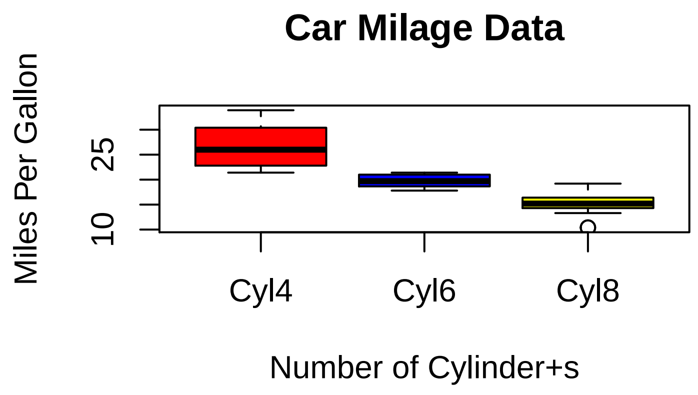
```
attach(mtcars)
table(cyl)
```

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

```
boxplot(mpg~cyl,data=mtcars, main="Car Milage Data", xlab="Number of Cylinders", ylab="Miles Per Gallon
```



```
boxplot(mpg~cyl,data=mtcars,main="Car Milage Data",xlab="Number of Cylinder+s",ylab="Miles Per Gallon",
```

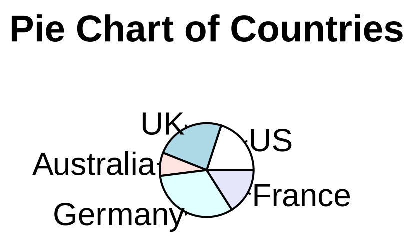


```
#names helps you to change the category names
```

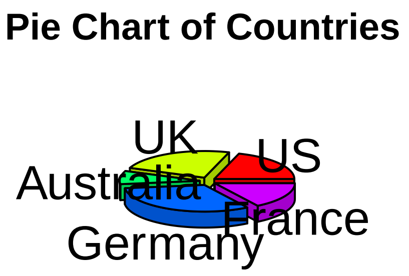
Pie Chart

Pie charts are created with the function pie(x, labels=) where x is a non-negative numeric vector indicating the area of each slice and labels= notes a character vector of names for the slices.

```
# Simple Pie Chart
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries") #label shows the label names
```



```
# 3D Exploded Pie Chart
library(plotrix)
slices <- c(10, 12, 4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie3D(slices,labels=lbls,explode=0.1, main="Pie Chart of Countries ")
```

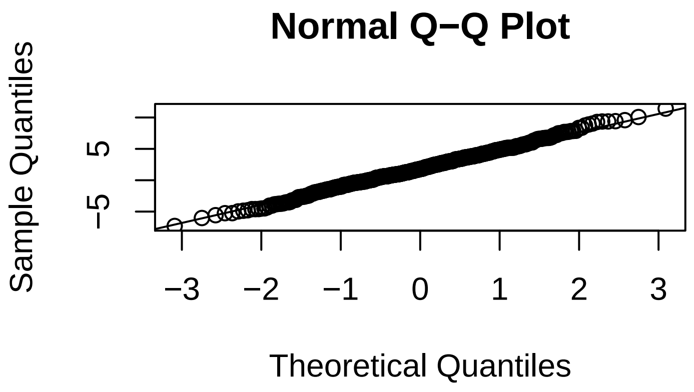


```
#explode helps you to divide your plot
#if you do not use it, you have only one big slice
```

Quantile-Quantile Plot (Q-Q Plot)

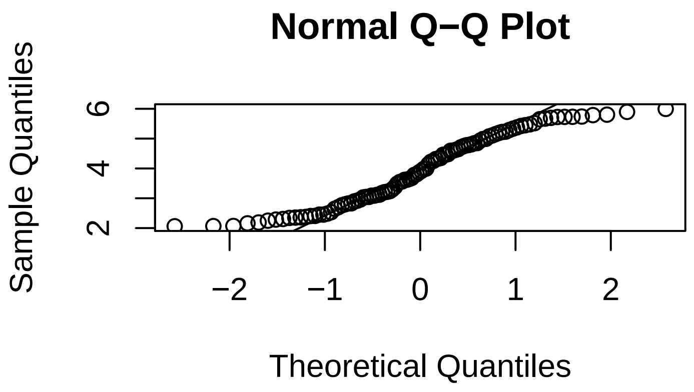
The quantile-quantile (q-q) plot is a graphical technique for determining whether the variable of interest follows the normal distribution or not.It is also used to detect outliers in some cases such as regression models.

```
x=rnorm(500,2,3) #generate random  number from normal distribution with  mean 2 and sd 3
qqnorm(x)#draw a qq plot
qqline(x) #to add normality line
```



To be normal, we expect to see most of the points should be on the normality line. If this is not the case, this may indicate non-normality.

```r
y=runif(100,2,6)#generate random     number from uniform distribution with a=2,b=6
qqnorm(y) #draw a qq plot
qqline(y)#to add normality line
```



Handling with Missing Data in R

Missing Data: In statistics, missing data, or missing values, occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data.

There are many types of missing data and different reasons for data being missing. Both issues affect the analysis. Some examples are:

. In a postal questionnaire survey not all the selected individuals respond;

. In a randomised trial, some patients are lost to follow up before the end of the study;

. In a multicentre study, some centres do not measure a particular variable;

. In a study in which patients are assessed frequently some data are missing at some time points for unknown reasons;

. Occasional data values for a variable are missing because some equipment failed;

. Some laboratory samples are lost in transit or technically unsatisfactory;

. In a magnetic resonance imaging study some very obese patients are excluded as they are too large for the machine;

. In a study assessing quality of life some patients die during the follow up period.

Data often are missing in research in economics, sociology, and political science because governments choose not to, or fail to, report critical statistics. Sometimes missing values are caused by the researcher-for example, when data collection is done improperly or mistakes are made in data entry.

How to deal with it in R?

In general, the missing values are recorded as `NA`, `-99`,0,`*`, or nonsense numbers such as `354564`. When you start your analysis, you should be aware of having missing value or not. If you have, then you have to fix this problem with appropriate methods, then you can go further.

```
ex <- c(4, 6, 2, NA, 9, 2)  # NA is used to show an object is a missing value in R.
#Note that, we don't need to use quotation marks to use NA even though it seems like a character
```

```
ex
```

```
## [1]  4  6  2 NA  9  2
```

Testing Missing Values

```
> is.na(x) # returns TRUE of x is missing
```

```
is.na(ex)
```

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

```
#TRUE indicates that the value is missing
```

```
a=c(2,6,5,4)
na.fail(a)
```

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

```
ex
```

```
## [1]  4  6  2 NA  9  2
```

```
na.fail(ex)
Error in na.fail.default(ex) : missing values in object
```

Recording values to Missing

R reads only `NA` as missing value. Other representations such as "-99","*" are not considered as NA.

```
> data=read.table("data.txt",header=T)
> data
   V1  V2  V3
1   9   4   3
2   5 -99   1
3 -99   2   2
4 -99 -99 -99
```

```
5  17   9   8
6   4   2 -99
> data[data==-99]=NA
> data
  V1 V2 V3
1  9  4  3
2  5 NA  1
3 NA  2  2
4 NA NA NA
5 17  9  8
6  4  2 NA
```

Finding out the number of missing values

```
> length(which(is.na(ex)))
[1] 1
> #We use length, which and is.na command together to find out the number of missing values
```

When you have missing values, you cannot do any analysis.

```
sum(ex)# Can not gives a result because of the missing case in the vector.
```

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

Excluding missing values from analysis

```
sum(ex, na.rm = TRUE)   # Now, we inform R that there is a missing case in the vector and want R to ign
```

```
## [1] 23
```

```
#na.rm command help me to remove NA term from the vector.
```

Removing missing values from data

```
ex
```

```
## [1]  4  6  2 NA  9  2
```

```
ex1=na.omit(ex)
#na.omit deletes NA values in your data.
ex1
```

```
## [1] 4 6 2 9 2
## attr(,"na.action")
## [1] 4
## attr(,"class")
## [1] "omit"
```

```
 sum(ex1)
```

```
## [1] 23
```

Missing Imputation .

. Fill-in or impute the missing values. Use the rest of the data to predict the missing values. Simply replacing the missing value of a predictor with the average value of that predictor is one easy method. Using regression on the other predictors is another possibility. It's not clear how much the diagnostics and inference on the filled-in dataset is affected. Some additional uncertainty is caused by the imputation which needs to be allowed for.

. Missing observation correlation. Consider just xi yi pairs with some observations missing. The means and SDs of x and y can be used in the estimate even when a member of a pair is missing. An analogous method is available for regression problems.

. Maximum likelihood methods can be used assuming the multivariate normality of the data. The EM algorithm is often used here. We will not explain the details but the idea is essentially to treat missing values as nuisance parameters. Most modeling functions in R offer options for dealing with missing values. You can go beyond pairwise of listwise deletion of missing values through methods such as multiple imputation. Good implementations that can be accessed through R include Amelia II, Mice, and mitools.

Graphics with `ggplot2`

What is ggplot2?

ggplot2 is a plotting system for R, based on the grammar of graphics, which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics.

*To get more information about this special packages, you can visit http://ggplot2.org/

*Also, there are many videos, books and pages related to this packages.

The ggplot2 implies "Grammar of Graphics" which believes in the principle that a plot can be split into the following basic parts -

Plot = data + Aesthetics + Geometry

- **data** refers to a data frame (dataset).
- **Aesthetics** indicates x and y variables. It is also used to tell R how data are displayed in a plot, e.g. color, size and shape of points etc.
- **Geometry** refers to the type of graphics (bar chart, histogram, box plot, line plot, density plot, dot plot etc.)

Why ggplot2 is better?

- Excellent themes can be created with a single command.
- Its colors are nicer and more pretty than the usual graphics.
- Easy to visualize data with multiple variables.
- Provides a platform to create simple graphs providing plethora of information.

Before starting class, please load the following libraries

```
install.packages(c("ggplot2","gcookbook")) #to install more than one library at the same time.
#ggplot2 is for the graphical representation
#gcookbook is for some special data sets
```

After installing, please do no forget to call your libraries for the next steps.

```
library(ggplot2) #calling ggplot2 library
```

```
## Warning: package 'ggplot2' was built under R version 3.4.3
```

```
library(gcookbook)#calling gcookbook library
```

```
## Warning: package 'gcookbook' was built under R version 3.4.3
```

Scatter Plot

Scatter plots can help you identify the relationship between two data samples. A scatter plot is a simple plot of one variable against another.

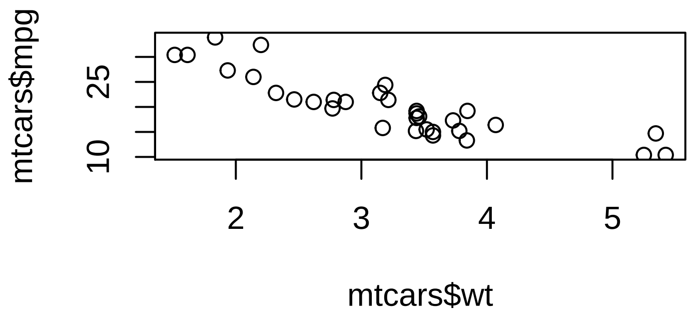
Short Reminder- How to draw a scatter plot using general commands?

Let's consider *mtcars* data set being available in R.

```
head(mtcars) #the first six observation of mtcars being an available in R is shown
```

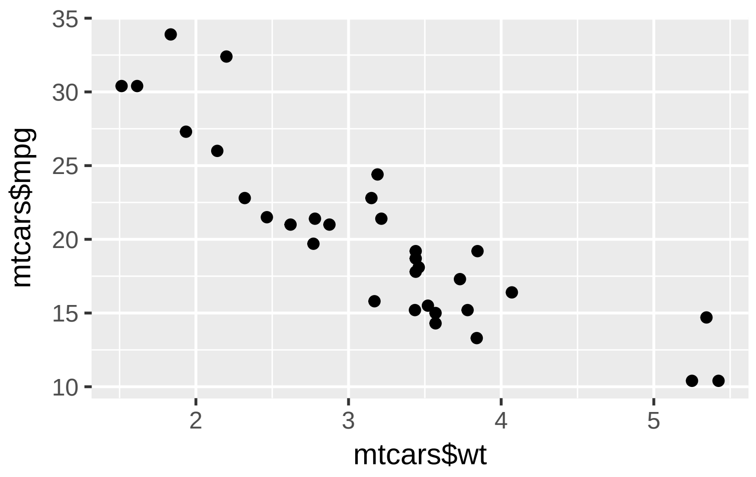
```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
plot(mtcars$wt, mtcars$mpg) #to draw a scatter plot, use plot command
```



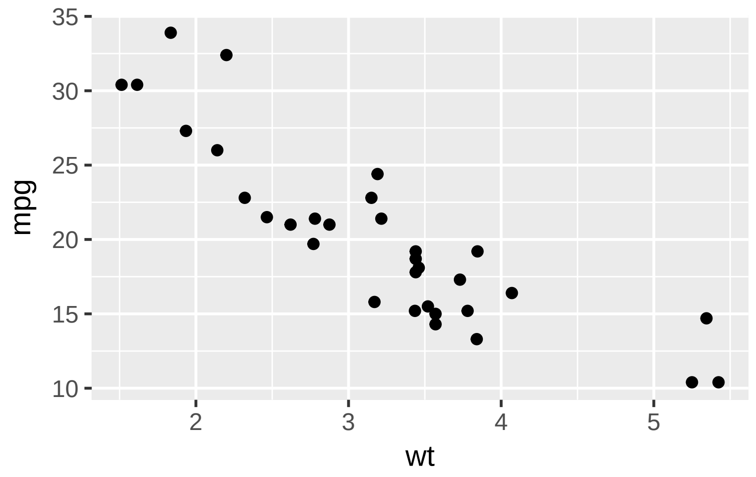
How can we draw same plot by using ggplot2?

```
qplot(mtcars$wt, mtcars$mpg) #first way
```



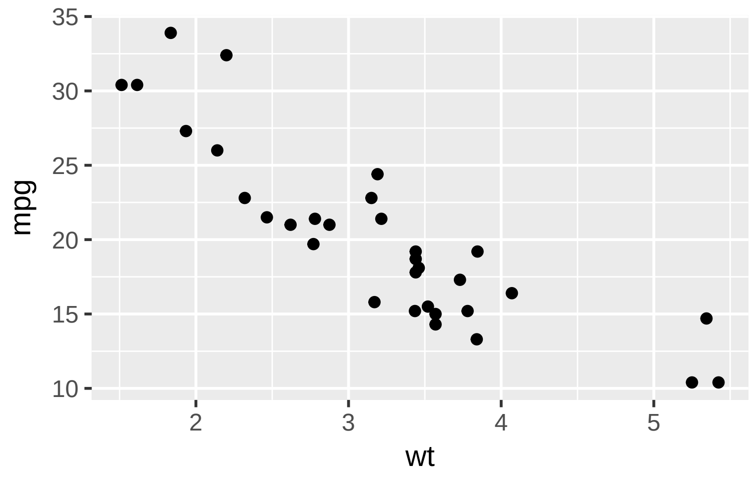
If the two vectors are already in the same data frame, you can use the following syntax:

```
qplot(wt, mpg, data=mtcars)
```



Another way

```
ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point() #suggested way
```

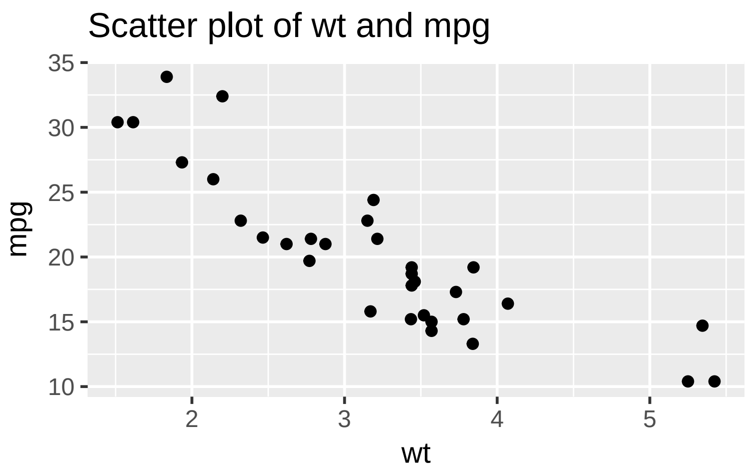


```
#geom_point function creates your plot as scatter plot
```

Adding Label Names and Titles

To add title and label names into your plot, `labs` command is used. Consider the previous plot.

```
ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point()+labs(title="Scatter plot of wt and mpg",x="wt", y = "mpg
```



```
#title is used to add title
#x change the name of x axes
#y change the name of y axes
```

The `labs` command can be used not only scatter plot, but also other plots to add title etc.

Drawing scatter plot of two continious variables conditioned on one categorical variables

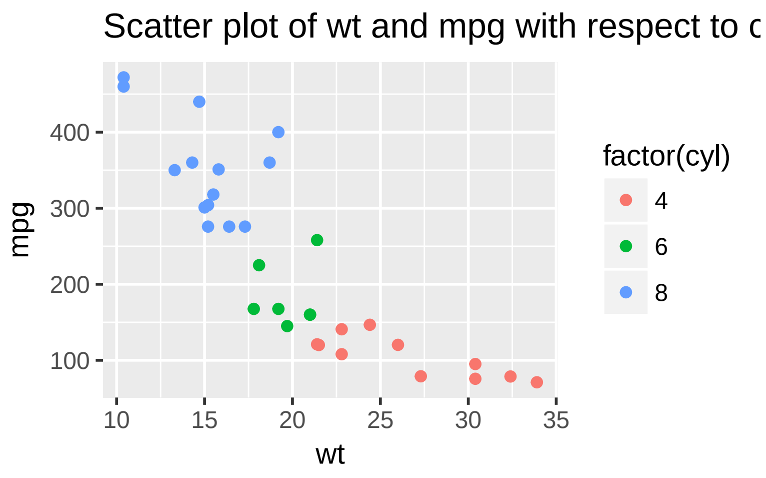
```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
table(mtcars$cyl) #produces frequency table
```

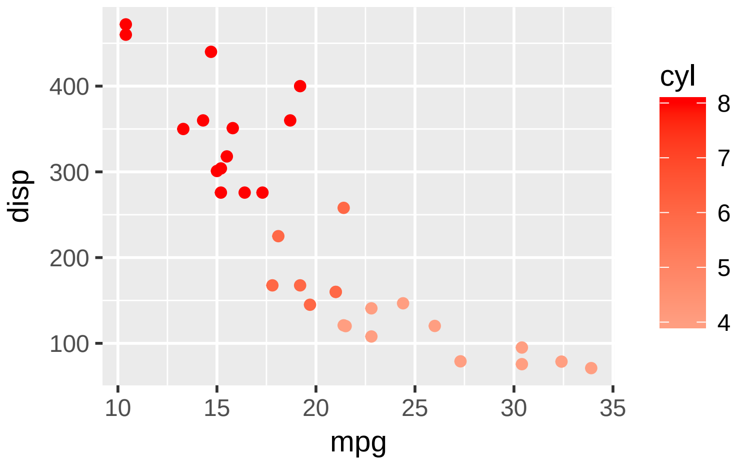
```
## 
##  4  6  8 
## 11  7 14
```

```
ggplot(data = mtcars,aes(x = mpg,y = disp,colour = factor(cyl))) + geom_point()+labs(title="Scatter plo
```



Changing color scale in legend

```r
library(ggplot2)
c <- ggplot(mtcars,aes(x = mpg, y = disp, color = cyl))  + geom_point()
c + scale_color_gradient2(low = "yellow", high = "red")
```

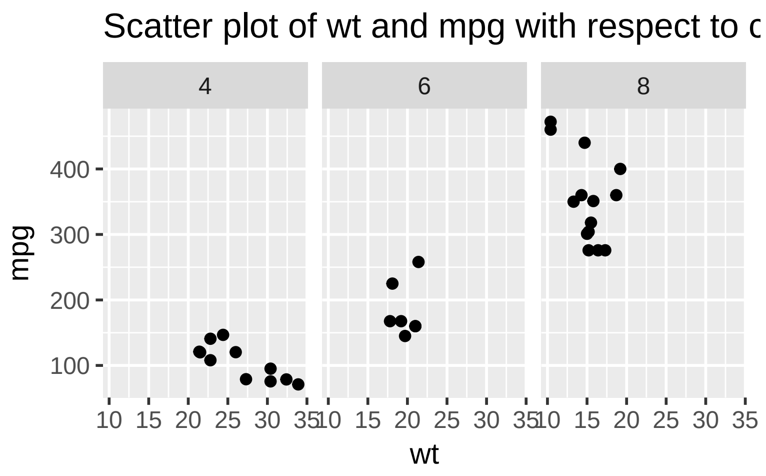


When you use `scale_color_gradient2` command, do not use `factor()` for coloring your plot.

Drawing multiple scatter plot

By using `facet_wrap` command, you can easily produce multiple scatter plot.

```r
ggplot(data = mtcars,aes(x = mpg,y = disp)) + geom_point()+labs(title="Scatter plot of wt and mpg with r
```



Line Plot

Line graph is used to look at the changes in variable over time or look at the relationship between two variable. In both cases, x axis corresponds to the independent variable(time, days etc), y axis corresponds to the dependent variable(temperature, income etc)

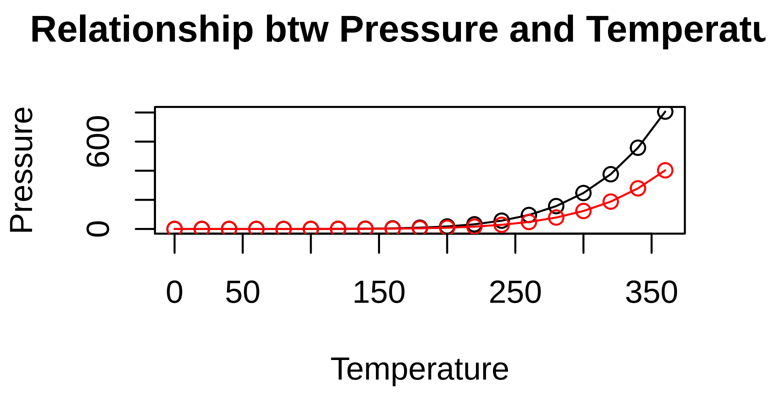
Short Reminder- How to draw a line plot using general commands?

Let's consider *pressure* data set being available in R.

```
head(pressure) #shows first 6 observations
```

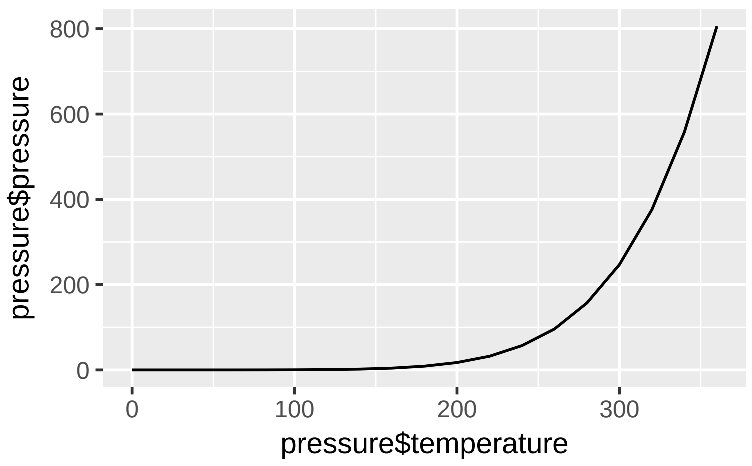
```
##   temperature pressure
## 1           0   0.0002
## 2          20   0.0012
## 3          40   0.0060
## 4          60   0.0300
## 5          80   0.0900
## 6         100   0.2700
```

```
plot(pressure$temperature, pressure$pressure, type="l",xlab="Temperature",ylab="Pressure",main="Relatio
#xlab and ylab options arrange axes name.
# To add points and/or multiple lines (Figure 2-3, right), first call plot() for the first line,
#then add points with points() and additional lines with lines():
points(pressure$temperature, pressure$pressure) #add black points
lines(pressure$temperature, pressure$pressure/2, col="red") #add red lines
points(pressure$temperature, pressure$pressure/2, col="red") #add points to the lines
```



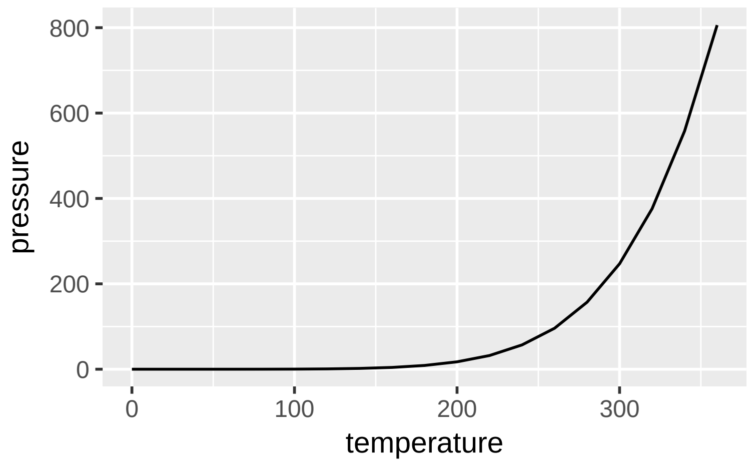
How can we draw same plot by using ggplot2?

```
qplot(pressure$temperature, pressure$pressure, geom="line")
```



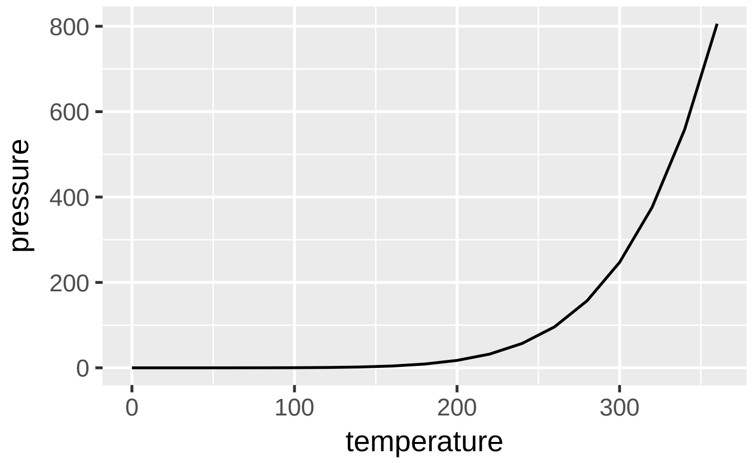
If the two vectors are already in the same data frame, you can use the following syntax:

```
qplot(temperature, pressure, data=pressure, geom="line") #with label name
```

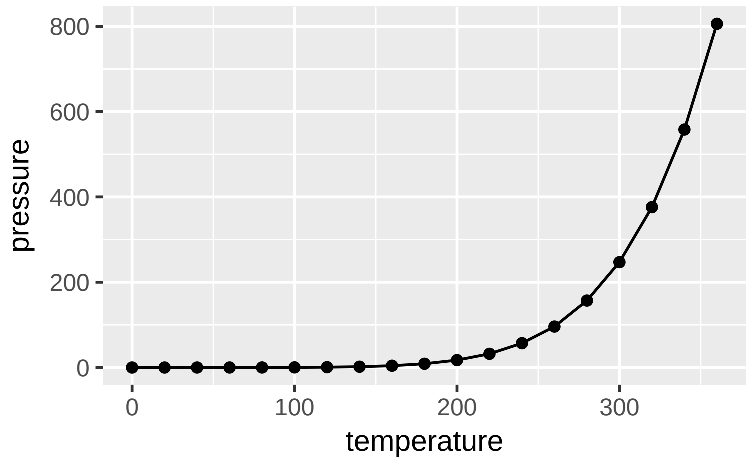


Another way

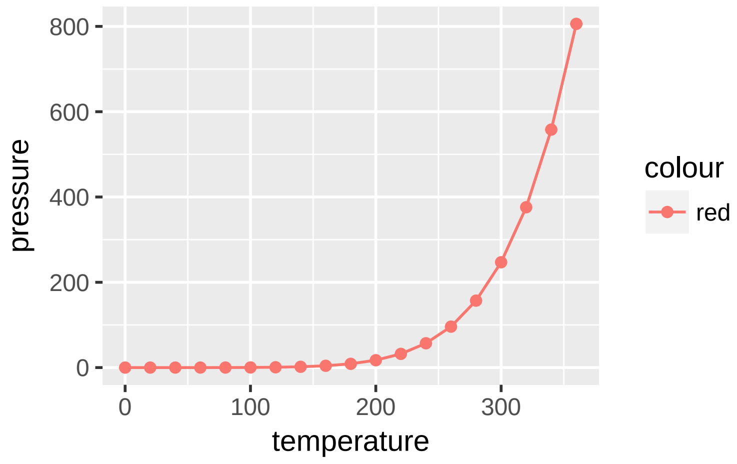
```
ggplot(pressure, aes(x=temperature, y=pressure)) + geom_line() #geom_line converts your plot into line 
```



```
ggplot(pressure, aes(x=temperature, y=pressure)) + geom_line() + geom_point() #to add points
```



```
qplot(temperature, pressure, data=pressure, geom=c("line", "point"),col="red") #to change colors of poi
```



Histogram

Quantitative variables often take so many values that a graph of the distribution is clearer if nearby values are group together. The most common graph of the distribution of one quantitative variable is a histogram. (Used for continious type of data)

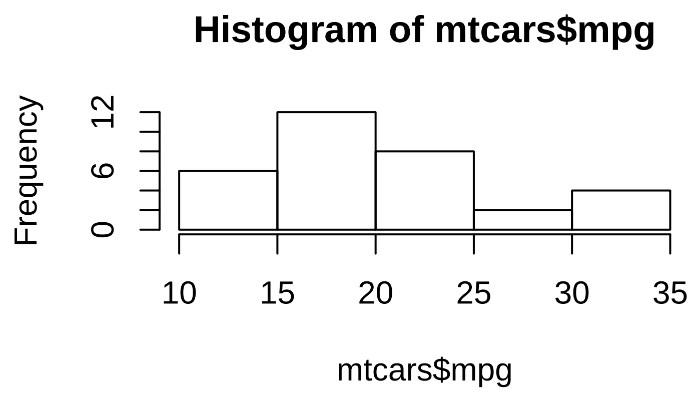
Histogram can be used for **continious** type of random variables

to find out the shape of the distribution of the variable of interest to detect the outlier

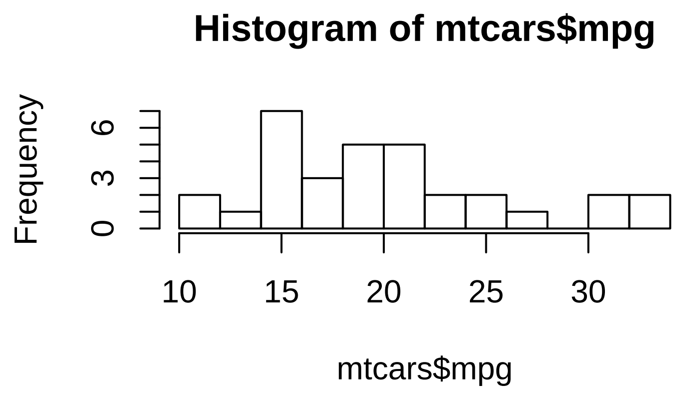
Short Reminder- How to draw a histogram using general commands?

Let's consider *mtcars* data set being available in R.

```r
hist(mtcars$mpg)
```



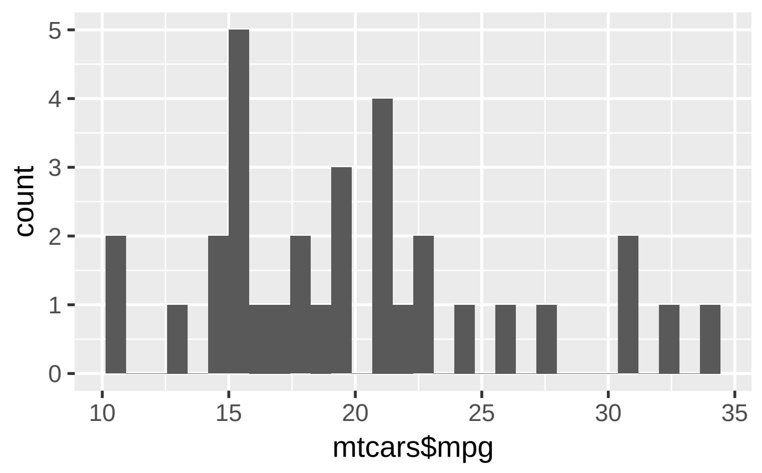
```r
hist(mtcars$mpg, breaks=10)#Specify approximate number of bins with breaks
```



How can we draw same plot by using ggplot2?

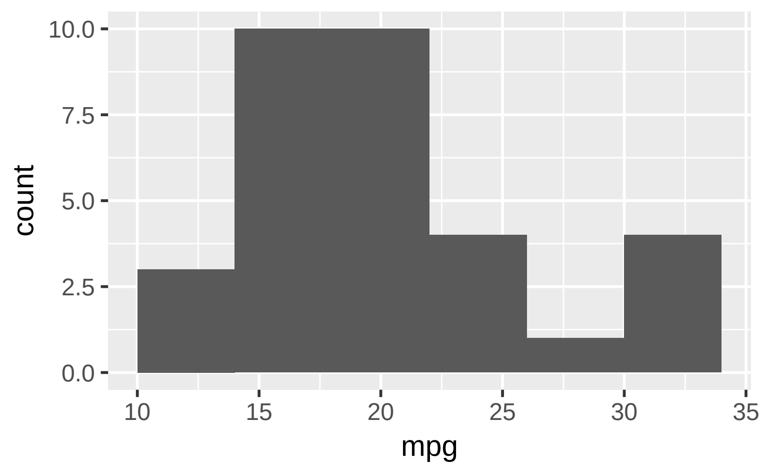
```r
qplot(mtcars$mpg)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



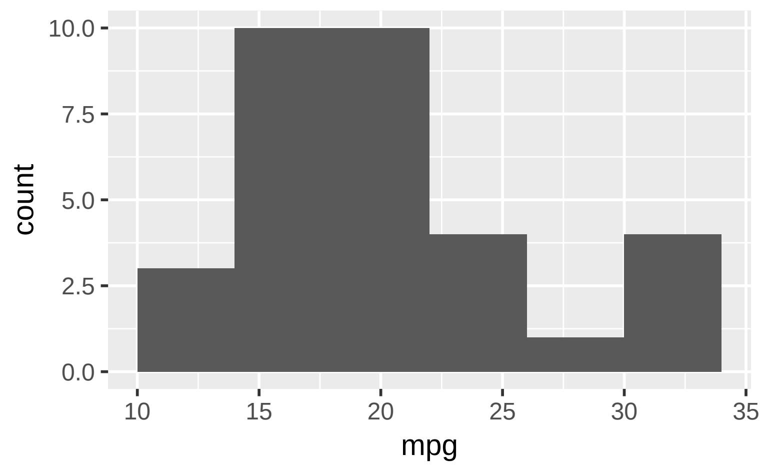
If the vector is in a data frame, you can use the following syntax:

```
library(ggplot2)
qplot(mpg, data=mtcars, binwidth=4)
```

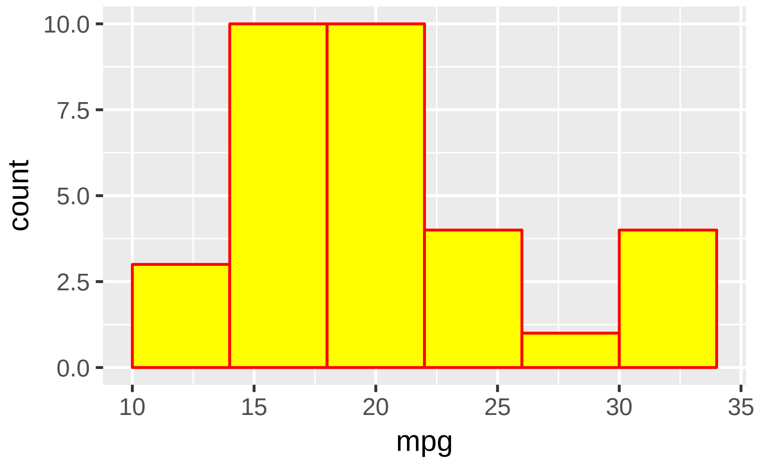


This is equivalent to:

```
ggplot(mtcars, aes(x=mpg)) + geom_histogram(binwidth=4)#geom_histogram converts your plot into histogra
```



```r
ggplot(mtcars,aes(x=mpg))+geom_histogram(binwidth = 4,colour="red",fill="yellow")#with color
```

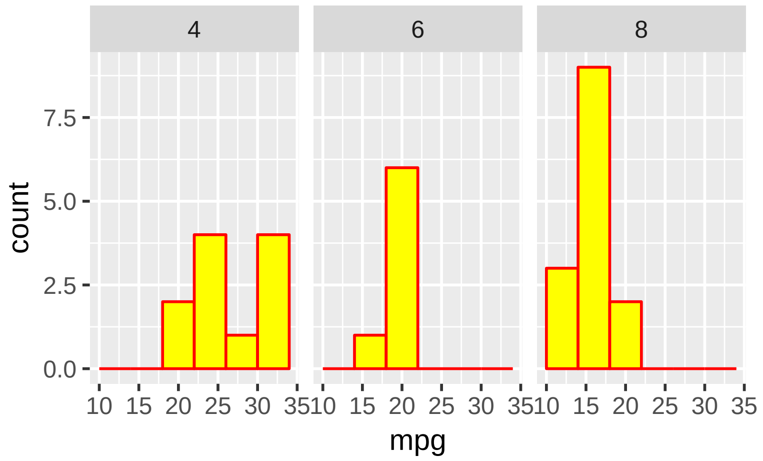


```r
#fill fills inside of histogram
#colour defines the color of frame
```

Drawing Multiple Histogram

By using `facet_wrap` command, you can easily produce multiple histogram.

```r
ggplot(mtcars,aes(x=mpg))+geom_histogram(binwidth = 4,colour="red",fill="yellow")+facet_wrap("cyl")
```



Box Plot

We use five number summary which are minimum, 1st quartile, median, 3rd quartile and maximum values of data to draw a box plot. On each box, the central mark indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points not considered outliers, and the outliers are plotted individually using the '*' symbol.

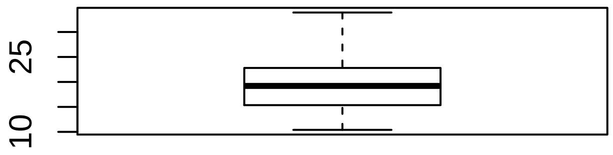
Box Plot can be used for **continious** type of random variables

to find out the shape of the distribution of the variable of interest to detect the outlier *to compare the variable of interest with respect to categorical variable.

Short Reminder- How to draw a box plot using general commands?

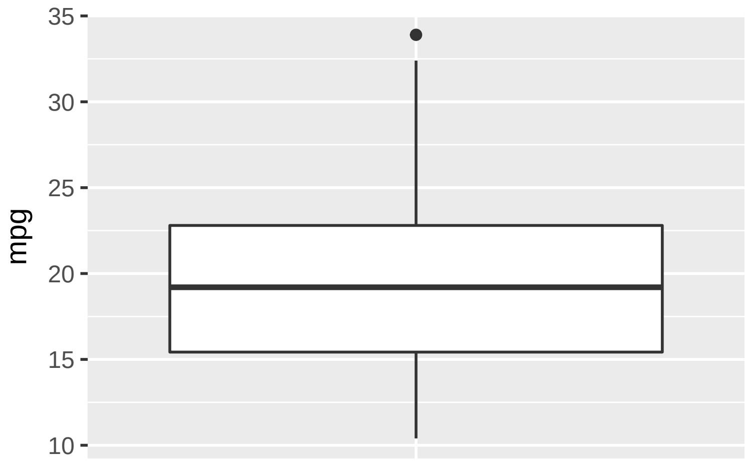
Let's consider *mtcars* data set being available in R.

```
boxplot(mtcars$mpg)
```

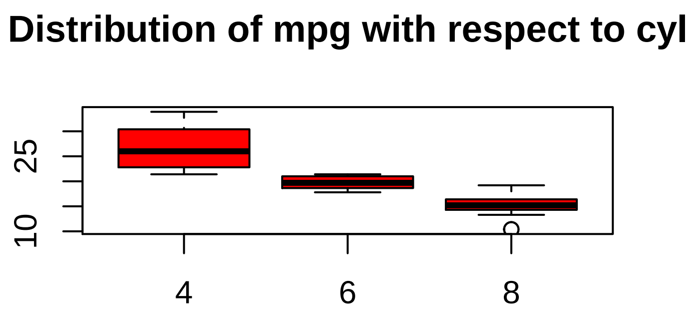


How can we draw same plot by using ggplot2?

```
ggplot(mtcars,aes(x=factor(0),mpg))+geom_boxplot()+
   theme(axis.title.x=element_blank(),
    axis.text.x=element_blank(),
    axis.ticks.x=element_blank())
```

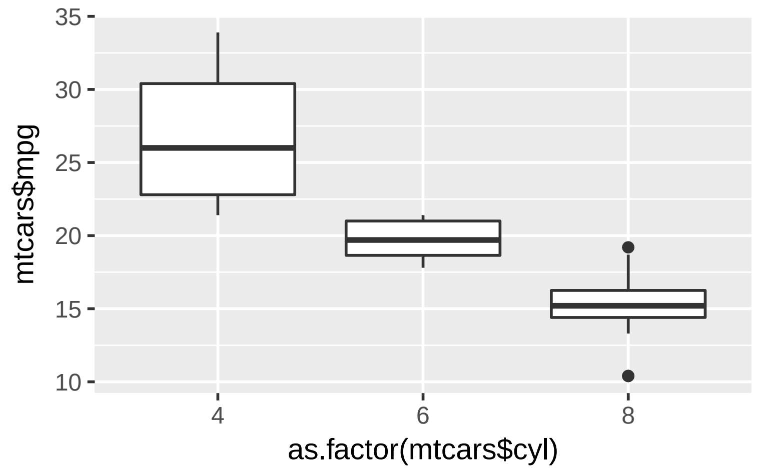


```
boxplot(mtcars$mpg~mtcars$cyl,main="Distribution of mpg with respect to cyl",col="red")#a simple box pl
```



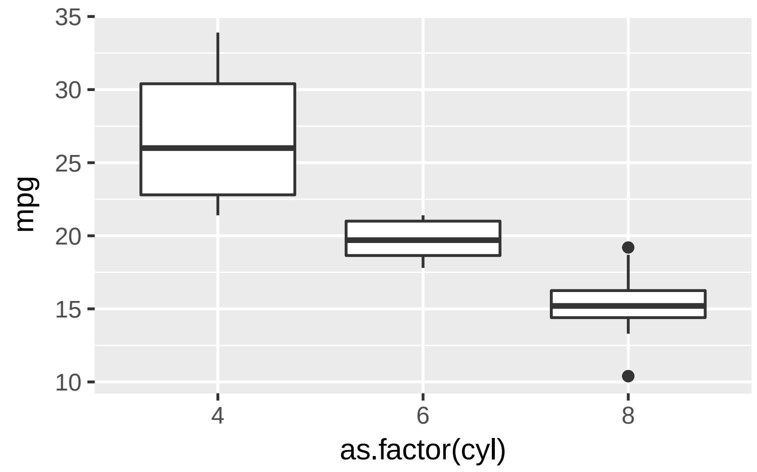
How can we draw same plot by using ggplot2?

```
qplot(as.factor(mtcars$cyl),mtcars$mpg,geom = "boxplot")
```



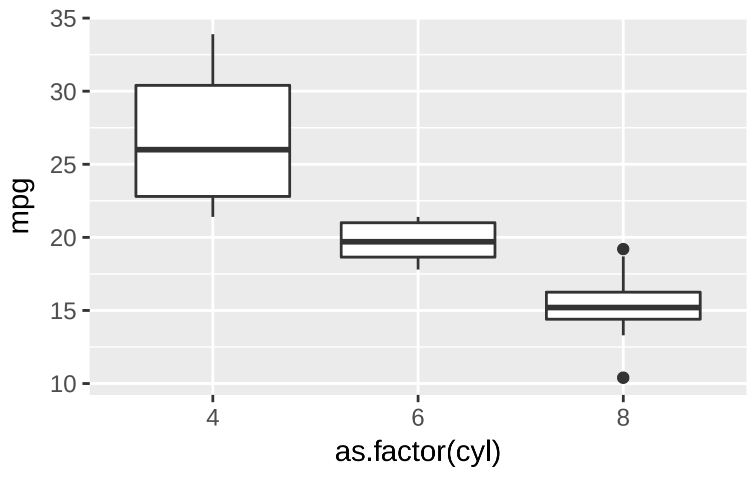
If the variables are in the same data frame

```
qplot(as.factor(cyl),mpg,data=mtcars,geom="boxplot")
```



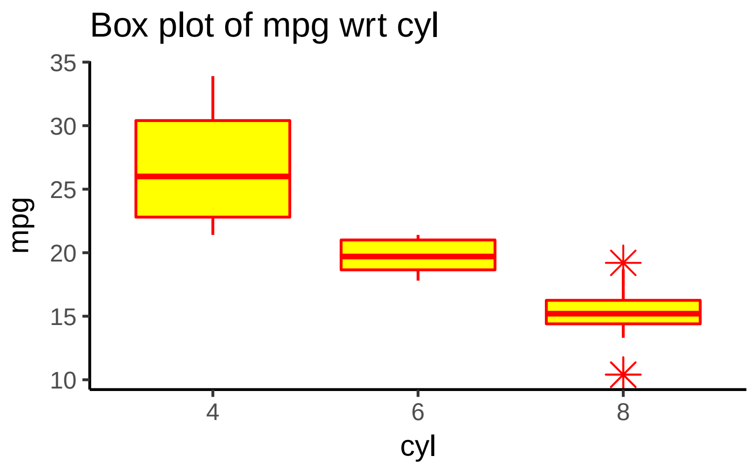
Another way

```
ggplot(mtcars, aes(x=as.factor(cyl), y=mpg)) + geom_boxplot( )
```



Customizing boxplot

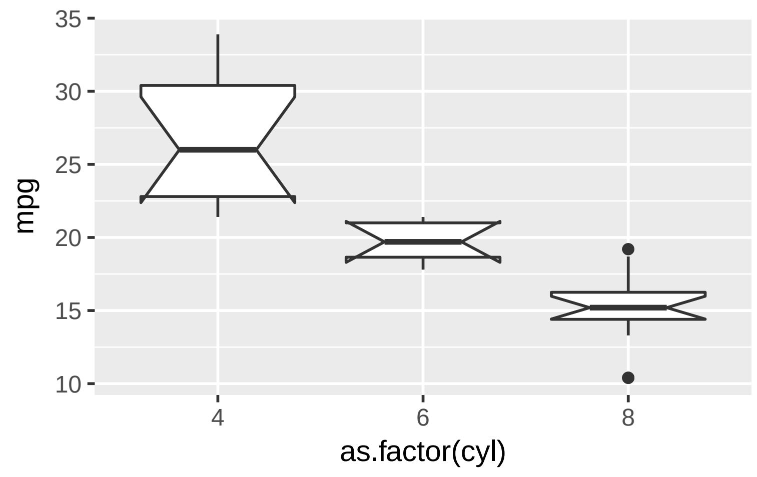
```
ggplot(mtcars, aes(x=as.factor(cyl), y=mpg)) + geom_boxplot(outlier.colour="red", outlier.shape=8,outli
labs(title="Box plot of mpg wrt cyl",x="cyl", y = "mpg")+ theme_classic()
```



Notched Boxplot

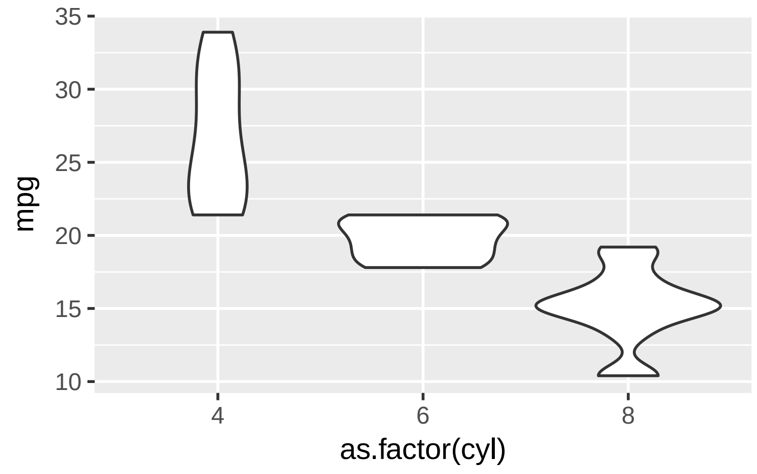
```
ggplot(mtcars, aes(x=as.factor(cyl), y=mpg)) + geom_boxplot(notch=TRUE)
```

```
## notch went outside hinges. Try setting notch=FALSE.
## notch went outside hinges. Try setting notch=FALSE.
```

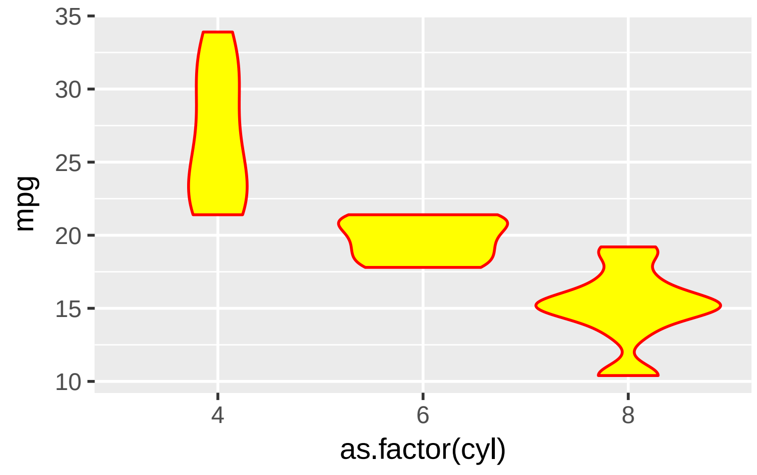


Violin Boxplot

```
ggplot(mtcars, aes(x=as.factor(cyl), y=mpg)) + geom_violin()
```



```
ggplot(mtcars, aes(x=as.factor(cyl), y=mpg)) + geom_violin(fill="yellow",colour="red")
```

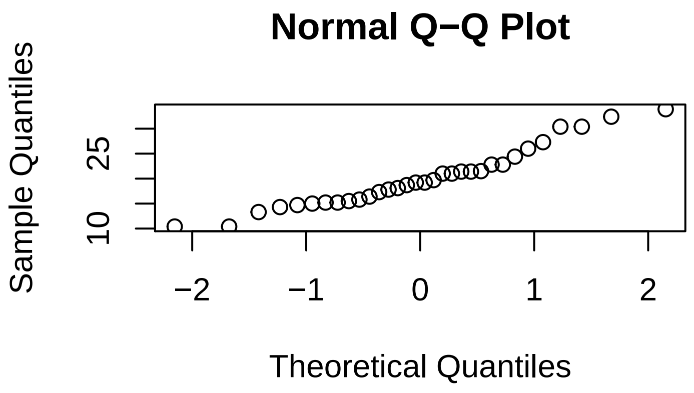


Quantile-Quantile Plot

The quantile-quantile (q-q) plot is a graphical technique for determining whether the variable of interest follows **the normal distribution or not.**

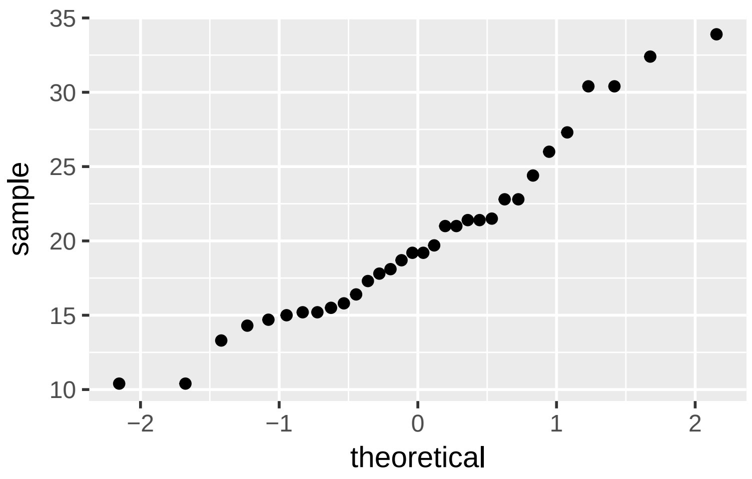
Short Reminder- How to draw a box plot using general commands?

```r
qqnorm(mtcars$mpg) #a simple qqplot in R
```

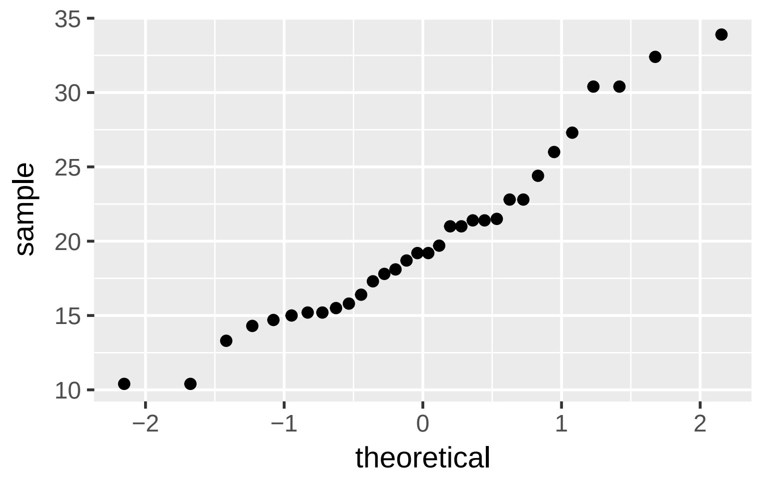


How can we draw same plot by using ggplot2?

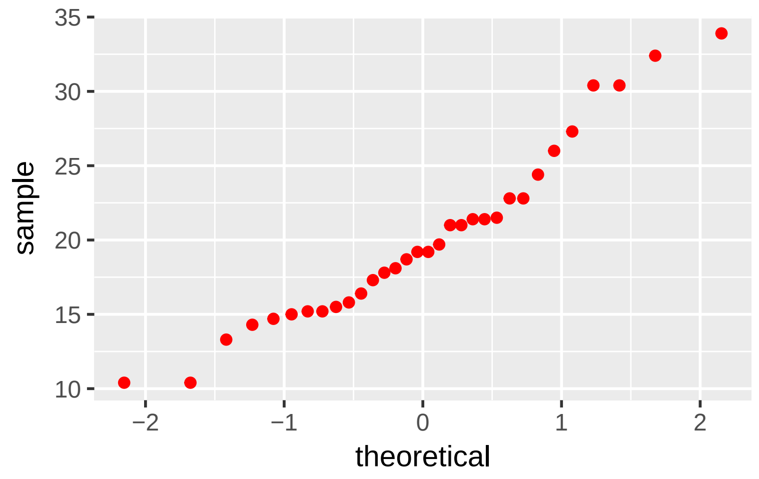
```r
qplot(sample = mpg, data = mtcars)
```



```r
ggplot(mtcars, aes(sample=mpg))+stat_qq()
```

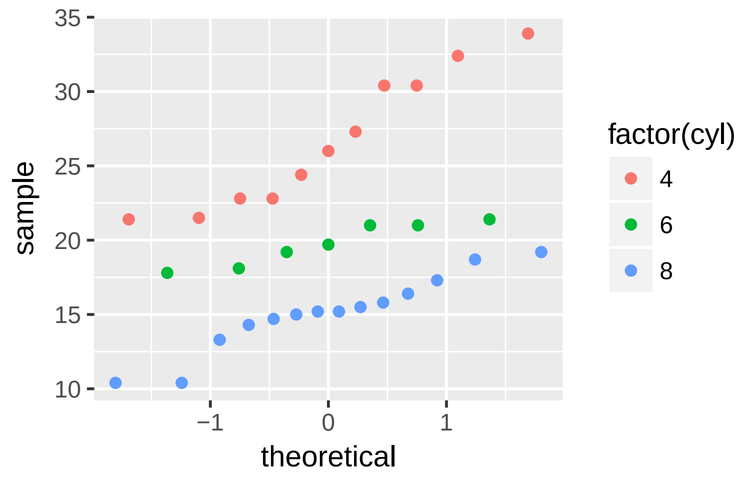


```r
ggplot(mtcars, aes(sample=mpg))+stat_qq(col="red") #with red color
```



Change qq plot colors by groups

```r
ggplot(mtcars) +stat_qq(aes(sample = mpg, colour = factor(cyl)))
```



In ggplot, we cannot directly add normality line, however there are many functions written for this purpose on the internet

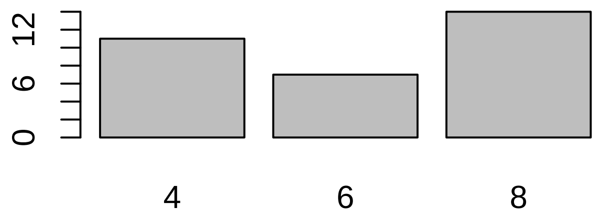
Bar Plot

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. It assumes that the heights of your bars are conveniently stored in a vector.

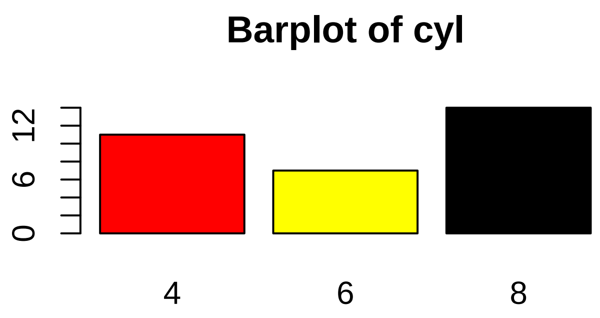
Short Reminder- How to draw a box plot using general commands?

Let's consider *mtcars* data set being available in R.

```r
barplot(table(mtcars$cyl))
```

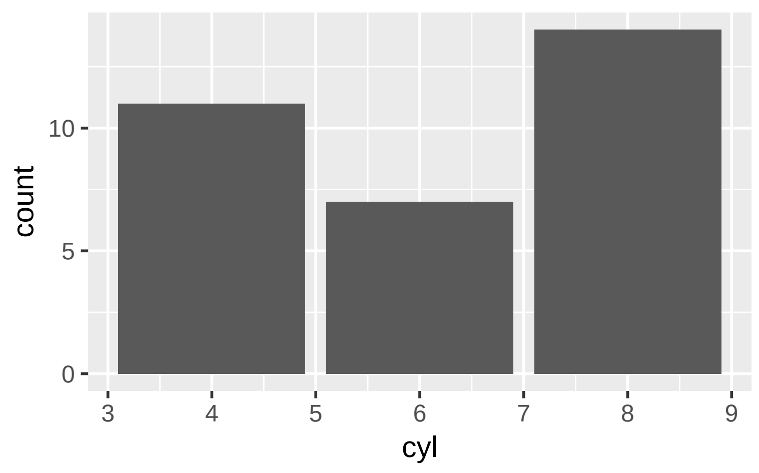


```r
barplot(table(mtcars$cyl),col=c("red","yellow","black"),main="Barplot of cyl")
```

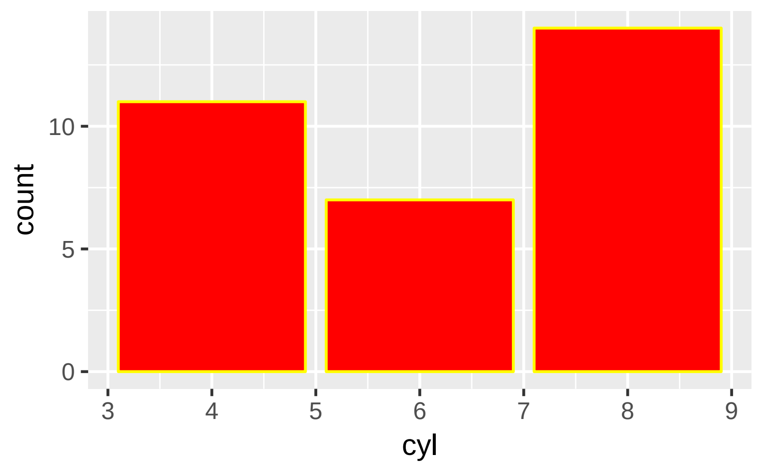


How can we draw same plot by using ggplot2?

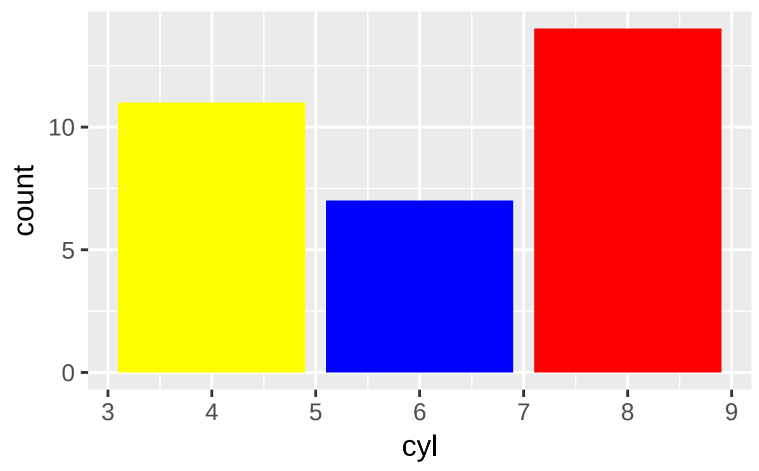
```r
ggplot(mtcars, aes(x=cyl)) + geom_bar()
```



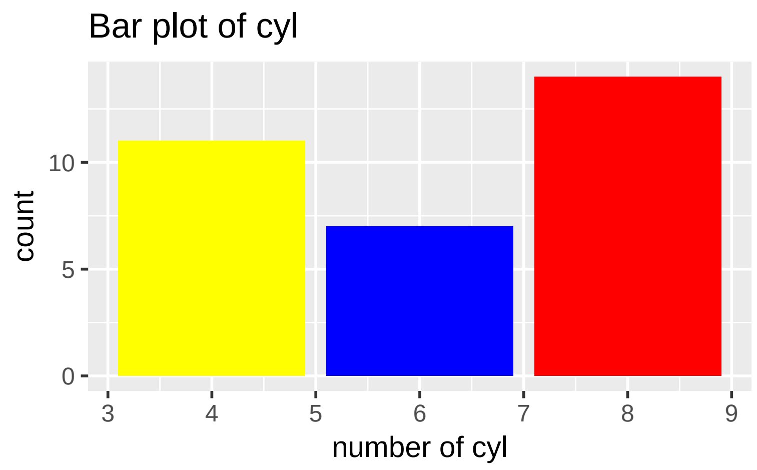
```
ggplot(mtcars, aes(x=cyl)) + geom_bar(fill="red",col="yellow")
```



```
ggplot(mtcars,aes(x=cyl))+geom_bar(fill=c("yellow","blue","red"))
```



```
ggplot(mtcars,aes(x=cyl))+geom_bar(fill=c("yellow","blue","red"))+labs(title="Bar plot of cyl",x="numbe
```

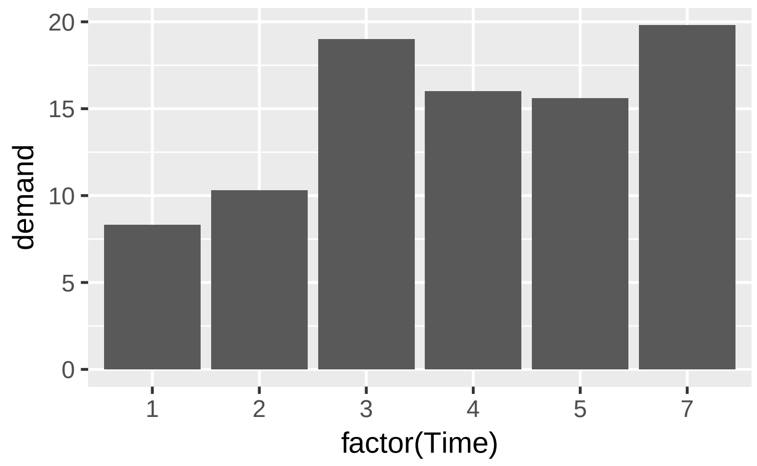


Drawing bar plot of continious variable with respect to one categorical variable

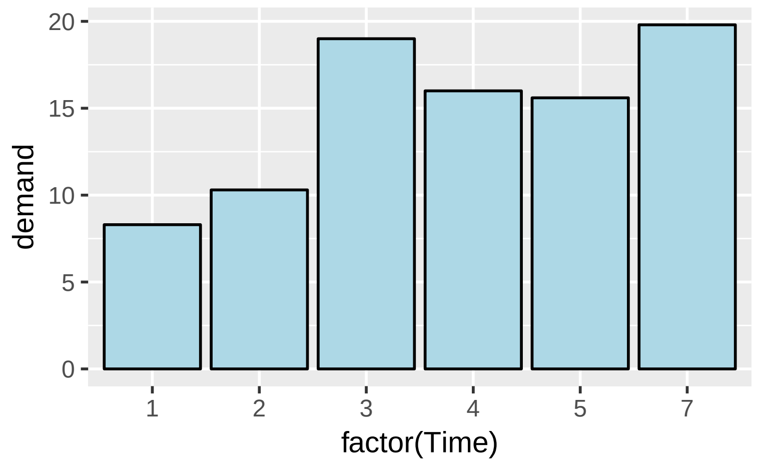
```
library(gcookbook)
BOD
```

```
##   Time demand
## 1    1    8.3
## 2    2   10.3
## 3    3   19.0
## 4    4   16.0
## 5    5   15.6
## 6    7   19.8
```

```
ggplot(BOD, aes(x=factor(Time), y=demand)) + geom_bar(stat="identity")# Convert Time to a discrete (cat
```



```
ggplot(BOD, aes(x=factor(Time), y=demand))+geom_bar(stat="identity", fill="lightblue", colour="black")
```

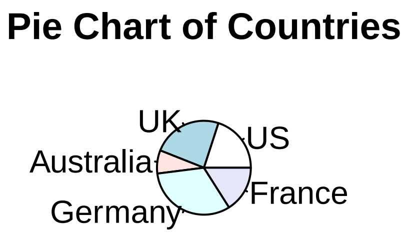


Pie Chart

A pie chart (or a circle chart) is a circular statistical graphic which is divided into slices to illustrate numerical proportion. It is mainly used to represent **categorical** variables

Short Reminder- How to draw a box plot using general commands?

```
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries") #label shows the label names
```

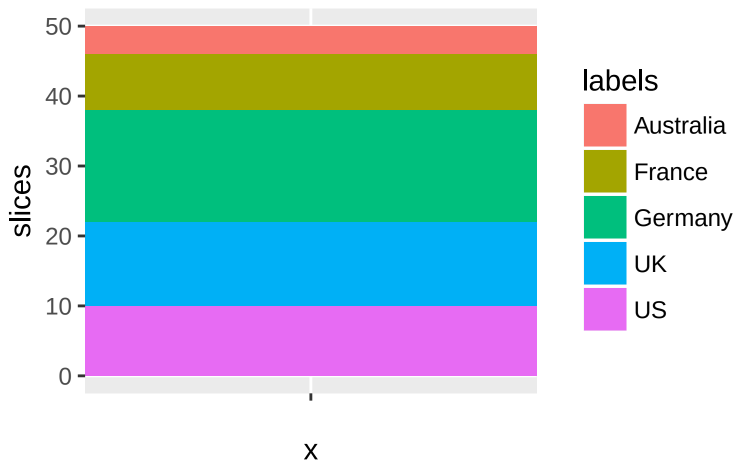


To draw a pie chart in ggplot2, you have to create a bar plot at first. Then, you should convert your bar plot into pie chart.

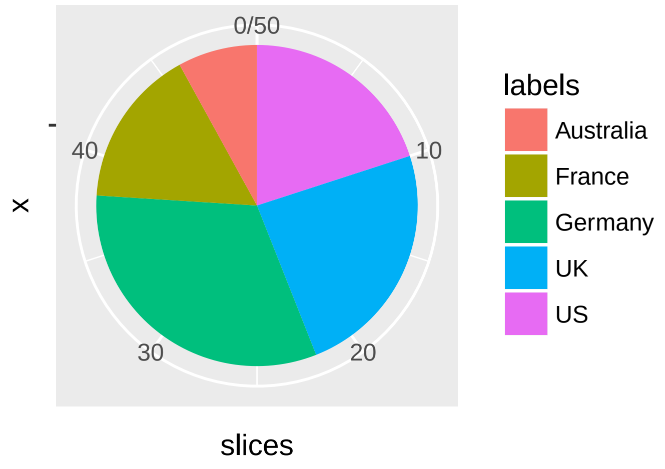
```
df<-data.frame(slices=c(10, 12,4, 16, 8),labels=c("US", "UK", "Australia", "Germany", "France"))
df
```

```
##   slices    labels
## 1     10        US
## 2     12        UK
## 3      4 Australia
## 4     16   Germany
## 5      8    France
```

```r
#Use a barplot to visualize the data :
library(ggplot2)
# Barplot
bp<- ggplot(df, aes(x="", y=slices, fill=labels))+geom_bar(width = 1, stat = "identity")
bp#ggplot2 pie chart for data visualization in R software
```



```r
#Create a pie chart :
bp + coord_polar("y", start=0)
```



Descriptive Statistics in R

Descriptive statistics are used to summarize data in a way that provides insight into the information contained in the data. This might include examining the mean or median of numeric data or the frequency of observations for nominal data. Plots can be created that show the data and indicating summary statistics.

Choosing which summary statistics are appropriate depend on the type of variable being examined. Different statistics should be used for interval/ratio, ordinal, and nominal data.

In describing or examining data, you will typically be concerned with measures of *location*, *variation*, and *shape*.

Measure of Central Tendency

Mean

The mean is the arithmetic average, and is a common statistic used with interval/ratio data. It is simply the sum of the values divided by the number of values. The `mean` function in R will return the mean. Please call beaver1 data being built-in R by typing `beaver1`.

Data description is as follows;

time=Time of observation, in the form 0330 for 3:30am

temp=Measured body temperature in degrees Celsius.

activ=Indicator of activity outside the retreat.

```
head(beaver1)
```

```
##   day time  temp activ
## 1 346  840 36.33     0
## 2 346  850 36.34     0
## 3 346  900 36.35     0
## 4 346  910 36.42     0
## 5 346  920 36.55     0
## 6 346  930 36.69     0
```

```
mean(beaver1$temp)
```

```
## [1] 36.86219
```

The average body temperature of participants is 36 Celsius.

What if we have `NA` term?

```
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```
sum(is.na(airquality$Solar.R)) #there are 7 missing observations.
```

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

```
mean(airquality$Solar.R,na.rm=T)
```

```
## [1] 185.9315
```

```
#the average value of solar radiation in Langleys is 185.913 Angstroms.
#*Angstroms=Measure of Solar Radiaton
```

Median

The median is defined as the value below which are 50% of the observations. To find this value manually, you would order the observations, and separate the lowest 50% from the highest 50%. For data sets with an odd number of observations, the median is the middle value. For data sets with an even number of observations, the median falls half-way between the two middle values. `median()` command is used to calculate the sample median.

```r
median(beaver1$temp)
```

```
## [1] 36.87
```

The half of the body temperature of participants is 36.87 Celsius or below . OR The half of the body temperature of participants is 36.87 Celsius or above .

What if we have `NA` term?

```r
median(airquality$Solar.R,na.rm=T)
```

```
## [1] 205
```

```r
#the half of the value of solar radiation in Langleys is  205 Angstroms or below.
#*Angstroms=Measure of Solar Radiaton
```

Note that median is more robust to outlier than mean.

Mode

The mode is a summary statistic that is used rarely in practice, but is normally included in any discussion of mean and medians. When there are discreet values for a variable, the mode is simply the value which occurs most frequently

```r
library(DescTools)
Mode(beaver1$temp) #with capital M
```

```
## [1] 36.89
```

The most of the body temperature of participants is 36.89 Celsius.

What if we have `NA` term?

```r
Mode(airquality$Solar.R,na.rm=T)
```

```
## [1] 238 259
```

```r
#the most of the value of solar radiation in Langleys is around 238 and 259 Angstroms.
#*Angstroms=Measure of Solar Radiaton
```

Measure of Dispersion

Range

The range of a set of data is the difference between the largest and smallest values. However, in descriptive statistics, this concept of range has a more complex meaning. The range is the size of the smallest interval which contains all the data and provides an indication of statistical dispersion. It is measured in the same units as the data. Since it only depends on two of the observations, it is most useful in representing the dispersion of **small data sets**.

```r
range<-range(beaver1$temp)
range[2]-range[1] #Shows the difference between minimum and maximum values
```

```
## [1] 1.2
```

The difference between minimum temperature and maximum temperature is 1.2 Celsius

Calculating Quantiles

Quantiles are cut points dividing the range of a probability distribution into contiguous intervals with equal probabilities, or dividing the observations in a sample in the same way.

```
quantile(beaver1$temp,0.75) #3rd quantile
```

```
##     75%
## 36.9575
```

The 75% of the body temperature of participants is 36.96 Celsius or below . OR The 25% of the body temperature of participants is 36.96 Celsius or above

What if we have `NA` term?

```
quantile(airquality$Solar.R,0.75,na.rm=T)
```

```
##    75%
## 258.75
```

```
#The 75% of the value of solar radiation in Langleys is  258.75 Angstroms or below.
#*Angstroms=Measure of Solar Radiaton
```

Median is the 2nd quantile of the data.

Interquartile Range

It is the difference between 3rd quantile and 1st quantile.An interquartile range is a measure of where the bulk of the values lie. It refers to spread of 50% of the data.

```
IQR(beaver1$temp)
```

```
## [1] 0.1975
```

What if we have `NA` term?

```
IQR(airquality$Solar.R,0.75,na.rm=T)
```

```
## [1] 146
```

Variance and Standard Deviation

The variance is a measure of how far each value in the data set is from the mean.Standard deviation is the measure of spread most commonly used in statistical practice when the mean is used to calculate central tendency. Thus, it measures spread around the mean.

Standard deviation is a square root of variance.

```
var(beaver1$temp)
```

```
## [1] 0.03741196
```

The interpretation of variance is not easy because the value you obtain is in the squared unit.

```
sd(beaver1$temp)
```

```
## [1] 0.1934217
```

Note that you can easily calculated mean,median,25th and 75th quartiles,min,max of any data by using `summary(mydata)`

*** You can easily calculated Tukey min,lower-hinge, median,upper-hinge,max of any data by using `fivenum(mydata)` ***

Please try these by yourself!

Measure of Association

They measure the statistical strength of the relationship on the variable of interests. Although there are many types tool to measure the relationship, covariance and Pearson correlation coefficients are the most known and common tools for "numerical" type of data. The main difference between covariance and correlation is, covariance shows the direction of the relationship depending on the sign of value (+'ve or -'ve). However, the correlation shows the strength of the "LINEAR" relationship between variables.

For categorical data, we use chi-square test. There are also Spearman rho and Kendall Tau correlation coefficients but they are non-parametric test and not commonly used.

We can investigate the relationship between variables by drawing line or scatter plot. However, you cannot be sure by looking these plots. In statistics, tests are always powerful than visual tools. Visual tools give idea, tests confirm the ideas.

Covariance

Covariance is a measure of how much two random variables vary together.

Please call iris data set.

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
cov(iris$Sepal.Length,iris$Petal.Length)
```

```
## [1] 1.274315
```

There are positive relationship between Sepal Length and Petal Length

```
cov(iris$Sepal.Length,iris$Sepal.Length)
```

```
## [1] 0.6856935
```

What is this value?

Correlation

Correlation is a statistical technique that can show whether and how strongly pairs of variables are related.

```
cor(iris$Sepal.Length,iris$Petal.Length)
```

```
## [1] 0.8717538
```

They are highly correlated.

Chi-Square Test

The Chi-Square test of independence is used to determine if there is a significant relationship between two nominal (categorical) variables. The frequency of each category for one nominal variable is compared across the categories of the second nominal variable. The data can be displayed in a contingency table where each row represents a category for one variable and each column represents a category for the other variable. Ho: The two categorical variables are independent. H1 The two categorical variables are dependent.

```
library(MASS)       # load the MASS package
head(survey)
```

```
##      Sex Wr.Hnd NW.Hnd W.Hnd    Fold Pulse    Clap Exer Smoke Height
## 1 Female   18.5   18.0 Right  R on L    92    Left Some Never 173.00
## 2   Male   19.5   20.5  Left  R on L   104    Left None Regul 177.80
## 3   Male   18.0   13.3 Right  L on R    87 Neither None Occas     NA
## 4   Male   18.8   18.9 Right  R on L    NA Neither None Never 160.00
## 5   Male   20.0   20.0 Right Neither    35   Right Some Never 165.00
## 6 Female   18.0   17.7 Right  L on R    64   Right Some Never 172.72
##        M.I    Age
## 1   Metric 18.250
## 2 Imperial 17.583
## 3     <NA> 16.917
## 4   Metric 20.333
## 5   Metric 23.667
## 6 Imperial 21.000
```

```
tbl = table(survey$Smoke, survey$Exer)
tbl                 # the contingency table
```

```
##
##         Freq None Some
##   Heavy    7    1    3
##   Never   87   18   84
##   Occas   12    3    4
##   Regul    9    1    7
```

```
chisq.test(survey$Smoke, survey$Exer) # to conduct chi-square test
```

```
## Warning in chisq.test(survey$Smoke, survey$Exer): Chi-squared approximation
## may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  survey$Smoke and survey$Exer
## X-squared = 5.4885, df = 6, p-value = 0.4828
```

```
#since p value is greater than 0.05, It can be said that smoke and exercise are independent variables.
```

Measure of Skewness and Kurtosis

Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point. Zero indicates symmetry. The larger its absolute value the more asymmetric the distribution. Positive values indicate a long right tail, and negative values indicate a long left tail.

When two or more symmetrical distributions are compared, the difference in them are studied with 'Kurtosis'. Measure of the relative peakedness of a distribution.

K = 3 indicates a normal "bellshaped" distribution (mesokurtic).

K < 3 indicates a platykurtic distribution (flatter than a normal distribution with shorter tails).

K > 3 indicates a leptokurtic distribution (more peaked than a normal distribution with longer tails).

Please visit http://personal.cityu.edu.hk/~meachan/Online%20Anthropometry/Chapter5/Ch5-3.htm

To calculate them in R, You should install `moments` packages. `install.packages("moments")`

```r
library(moments)
skewness(beaver1$temp) #left skewed
```

```
## [1] -0.02782567
```

```r
kurtosis(beaver1$temp) #leptokurtic
```

```
## [1] 4.351118
```

What if we have `NA` term?

```r
skewness(airquality$Solar.R,na.rm=T)
```

```
## [1] -0.4236342
```

```r
kurtosis(airquality$Solar.R,na.rm=T)
```

```
## [1] 2.023567
```

Control Structures

Control structures allow you to put some "logic" into your R code, rather than just always executing the same R code everytime. Commonly used control strucures are

`if` and `else` : testing a condition and acting on it

`for`: execute a loop a fixed number of times

`while`: execute a loop while a condition is true

`repeat`: execute an infinite loop (must break out of it to stop)

`break`: break the execution of a loop

`next`: skip an interation of a loop

*** if-else***

This combination is probably is the most commonly used control structre in R. In this structure, you are able to test a condition and act on it depending on whether it's true or f. There are three structres in if-else combinations.

First one,

```
if (condition){
#do something if condition is true
}
```

Second one,

```
if (condition){
#do something if condition is true
}
else{
#do someting if condition is not true
}
```

Third one,

```
if (condition){
#do something if condition is true
} else if (condition2) {
#do someting if condition2 is  true
} else {
#do something if neither condition 1 nor condition 2 is true
}
```

ifelse(test, yes, no)

It is a function where "test" is the logical condition, "yes" is what iftest will return if the logical condition "test"is true, and no" is what iftest returns when f.

```
> x<- c(0.8289285, -0.9927791, -0.6685806, -0.6130534, -0.5632220,0.2899354, 0.9422919, -0.9862554, 0.8
> y <- ifelse(x>0, 1, -1)
> y
[1] 1 -1 -1 -1 -1 1 1 -1 1 -1
```

Example

Write a structure to calculate the median of given vector x where x is 1,5,9,7,2,10. Hint: use `sort` and `length` commands.

```
x<-c(1,5,9,7,2,10)
sortx<-sort(x)
n=length(sortx)#you can also write sort(x)
  if(n%%2==0) {
median <- (sortx[n/2]+sortx[1+n/2])/2
} else {
  median <- sortx[(n+1)/2]
}
median
```

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

Example

For a given vector numbers where numbers are 4,5,9,6,2,1,3. Please classify them as odd and even by using `ifelse`.

```
numbers<-c(4,5,9,6,2,1,3)
ifelse(numbers%%2==0, "even","odd")
```

```
## [1] "even" "odd"  "odd"  "even" "even" "odd"  "odd"
```

`for` loops

In R , for loops take an interator variable and assign it successive values from a sequence or vector. It is most commonly used for *iterating over the elements of an object.*

Let's start with a simple example

```
for (i in 1:10) {
    print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
x <- c("apples", "oranges", "bananas", "strawberries")

for (i in length(x)) {
    print(x[i])
}
```

```
## [1] "strawberries"
```

```
for (i in  1:length(x)) {
    print(x[i])
}
```

```
## [1] "apples"
## [1] "oranges"
## [1] "bananas"
## [1] "strawberries"
```

Example

Write a structure for a given vector v=4,5,-6,2,0,-2,4 that decides which number is negative which number is positive. If a number from the vector is positive, print "it is a positive number", if it is negative, print "it is a negative number". If it is equals to zero, print "it equals to zero"

```
v=c(4,5,-6,2,0,-2,4)
for (i in 1:length(v))
if(v[i]>0){
  print("it is a positive number")
```

```r
} else if (v[i]<0){
  print("it is a negative number")
} else{
  print("it equals to zero")
}
```

```
## [1] "it is a positive number"
## [1] "it is a positive number"
## [1] "it is a negative number"
## [1] "it is a positive number"
## [1] "it equals to zero"
## [1] "it is a negative number"
## [1] "it is a positive number"
```

Nested `for` loops

for loops can be nested inside of each other.

```r
m <- matrix(1:10, 2)
for (i in 1:nrow(m)) {
    for (j in 1:ncol(m)) {
        print(m[i, j])
    }
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
## [1] 2
## [1] 4
## [1] 6
## [1] 8
## [1] 10
```

They are commonly used for multidimensiional or hiearchical data structures. Be careful with nesting though. Nesting beyond 2 or 3 levels often makes it difficult to read/understand the code. If you find yourself in a need of a large number of nested loops, you may want to break up the loops by using functions.

Example

a. Create a matrix of size 6 by 6, where the entries are equal to the summation of the row and column number.

```r
n<-6
mymat<-matrix(0,n,n)


for ( i in 1:nrow(mymat))   {
for ( ii in 1:ncol(mymat))  {

mymat[i,ii]<-i+ii

}
}
mymat
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    2    3    4    5    6    7
## [2,]    3    4    5    6    7    8
## [3,]    4    5    6    7    8    9
## [4,]    5    6    7    8    9   10
## [5,]    6    7    8    9   10   11
## [6,]    7    8    9   10   11   12
```

Create the following square matrix of size 6 by 6.

```
n<-6
mymat<-matrix(0,n,n)

for ( i in 1:nrow(mymat))   {
for ( j in 1:ncol(mymat))  {
if ( i==j)  {mymat[i,j]<-1}
else { mymat[i,j]<-i+j }

}
}
mymat
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    4    5    6    7
## [2,]    3    1    5    6    7    8
## [3,]    4    5    1    7    8    9
## [4,]    5    6    7    1    9   10
## [5,]    6    7    8    9    1   11
## [6,]    7    8    9   10   11    1
```

c. Create the following square matrix of size 6.

```
n<-6
mymat<-matrix(0,n,n)

for ( i in 1:nrow(mymat))   {
for ( j in 1:ncol(mymat))  {

if ( i==j)  {mymat[i,j]<-1}
else if ( i> j)   { mymat[i,j]<-i+j }
else { mymat[i,j]<-100}

}
}
mymat
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1  100  100  100  100  100
## [2,]    3    1  100  100  100  100
## [3,]    4    5    1  100  100  100
## [4,]    5    6    7    1  100  100
## [5,]    6    7    8    9    1  100
## [6,]    7    8    9   10   11    1
```

Example

Write a "sort" algorithm for the given vector v where v=4,5,-6,2,0,-2,4 by using for loop.

```
v=c(4,5,-6,2,0,-2,4)
vsort<-c()
for(i in 1:length(v)){
  vsort[i]<-min(v)
  v=v[-which.min(v)]
}
vsort
```

```
## [1] -6 -2  0  2  4  4  5
```

while

While loops begin by testing a condition. If it is true, then they execute the loop body. Once the loop body is executed, the condition is tested again, and so forth, until the condition is f, after which the loop exists. Let's write a simple while.

```
count<-0
while(count<10){
  count<-count+1
  print(count)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

Example

For a given x equals to 2 and n equals to 8, write an R program to calculate using a while loop.

```
x<-2
n<-8
i<-0
sum<-0
while(i<=n) {
sum<-sum+(x^i)
i<-i+1
}
sum
```

```
## [1] 511
```

`repeat` loops

repeat initiates an infinite loop right from the start. These are not commonly used in statistical or data analysis applications but they do have their uses. The only way to exit a repeat loop is to call `break`.

`next` and `break`

“next“‘ is used to skip an iteration of a loop.

```
for(i in 1:100) {
if(i <= 20) {
## Skip the first 20 iterations
next
}
## Do something here
}
```

break is used to exit a loop immediately, regardless of what iteration the loop may be on.

```
for(i in 1:100) {
print(i)
if(i > 20) {
## Stop loop after 20 iterations
break
}
}
```

Functions

Although there are many built-in functions in R, you have to create your own functions in R. You create an R function using the `function` keyword. For example, the following function computes the square of any given number.

```
square<-function(x){
x^2
}
> square(2)
[1] 4
> square(-2)
[1] 4
>
```

In this example, the defined function uses only one argument. However, when you define a function, R allows you to use more than one arguments.

```
> sum.of.square<-function(x,y){
+   (x^2)+(y^2)
+   }
> sum.of.square(1,2)
[1] 5
```

The following function does this by including three statements: one for computing the mean of its input, one for getting the standard deviation, and a final expression that returns the input scaled to be centered on the mean and having one standard deviation:

```
rescale <- function(x) {
m <- mean(x)
s <- sd(x)
(x - m) / s
}
> x<-c(5,6,3,9,6)
> rescale(x)
[1] -0.36901248  0.09225312 -1.29154369  1.47604993  0.09225312
```

In defining functions, it is possibe to use control structures such as `if-else`, `for` etc.

```
> Identity<-function(n){
+  mat<-matrix(0,n,n)
+ for (i in 1:n){
+ for (j in 1:n){
+  if (i==j){
+  mat[i,j]=1
+     }
+     }
+ }
+  mat
+ }
> Identity(2)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

It is also possible to use users defined function while defining a new function. For example,

```
square<-function(x){
x^2
}
equation<-function(x){
square(x)+(3*x)+5
}
> equation (2)
[1] 15
```

`return` in functions

If you want to return a value from a function before its last expression, you can use the return function. It might look like a keyword, but it is a function, and you need to include the parentheses when you use it.

```
return(expression)
```

`Return`is usually used to exit a function early and isn't used that much in most R code. It is easier to return a value by just making it the last expression in a function rather than explicitly using return. Here, there are some cases where `return` is used.

```
> rescale(x)
> rescale <- function(x) {
  + m <- mean(x)
  + s <- sd(x)
  + (x - m) / s
  + (x + m)/s
  + }
> rescale(x)
[1] 4.981669 5.442934 4.059137 6.826731 5.442934

> rescale <- function(x) {
  +   m <- mean(x)
  +   s <- sd(x)
  +   (x - m) / s
  +   (x + m)/s
  +   return((x - m) / s)
  + }
> rescale(x)
[1] -0.36901248  0.09225312 -1.29154369  1.47604993  0.09225312

> rescale <- function(x) {
  +   m <- mean(x)
  +   s <- sd(x)
  +   scale<-(x - m) / s
  +   notscale<-(x+m)/s
  + }
> rescale(x)
> rescale <- function(x) {
  + m <- mean(x)
  + s <- sd(x)
  + scale<-(x - m) / s
  + notscale<-(x+m)/s
  + return(scale)
  +   }
> rescale(x)
[1] -0.36901248  0.09225312 -1.29154369  1.47604993  0.09225312
```

Named Parameters and Default Parameters

```
rescale <- function(x, ozancan) {
m <- mean(x)
translated <- x - m
if (ozancan) return(translated)
s <- sd(x)
translated / s
}
```

In this example, "ozancan" is the named paramater. If it is true, function runs one expression, if it is f, function runs another expression.

```
> rescale(x,ozancan=T)
[1] -0.8  0.2 -2.8  3.2  0.2
> rescale(x,ozancan=F)
[1] -0.36901248  0.09225312 -1.29154369  1.47604993  0.09225312
```

Suggestion

Note that writing function in R is not that much easy. If you want to learn more, you can look at the following book; *Thomas Mailund-Functional Programming in R. Advanced Statistical Programming for Data Science, Analysis and Finance-Apress (2017)*

Example

Write a function that shows the absolute value of any given number.

```
abs<-function(x){
  ifelse(x<0,-x,x)
}
abs(-5)
```

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

Example

Create the function unique, which given a vector will return a new vector with the elements of the first vector with duplicated elements removed. (Please try your functions for any vector.)

```
f.uniq <- function (v) {
  s <- c()
 for(i in 1:length(v)) {
    if(sum(v[i] == s) == 0) {
      s <- c(s, v[i])
    }
  }
  s
}
f.uniq(c(9, 9, 1, 1, 1, 0))
```

```
## [1] 9 1 0
```

Example Write a function to convert temperature from one scale to another (Fahrenheit <-> Celsius). The function should have two arguments: the temperature to convert, and an argument `to.celsius`, which has a default value of `TRUE`. If the function is passed `to.celsius = TRUE`, it should convert the passed temperature from Fahrenheit to Celsius and return the value in Celsius degrees. If `to.celsius` is `F`, it should return the Fahrenheit value of the passes Celsius temperature. A reminder, the two conversion formulas are:

ºF = ºC * 9/5 + 32

ºC = (ºF - 32) * 5/9

(Please try your functions for any vector.)

```
# Returns Fahrenheit conversion of passed c.temp (in Celsius)

convert.to.far <- function(c.temp) {

f.temp <- c.temp * 9/5 + 32

return(f.temp)
}

# Returns Celsius conversion of passed  f.temp (in Fahrenheit)

convert.to.cel <- function(f.temp)
{
c.temp <- (f.temp - 32) * 5/9

return(c.temp)
}
call
```

```
## function (name, ...)  .Primitive("call")
```

```
convert.temp <- function(temp, to.celsius = TRUE) {

if (to.celsius) {

converted <- convert.to.cel(temp)
} else {
```

```r
converted <- convert.to.far(temp)

}

return(converted)

}
temp<-c(15,26,35,37,26,4,-2)
convert.temp(temp,to.celsius = T)
```

```
## [1]  -9.444444  -3.333333   1.666667   2.777778  -3.333333 -15.555556
## [7] -18.888889
```

```r
convert.temp(temp,to.celsius = F)
```

```
## [1] 59.0 78.8 95.0 98.6 78.8 39.2 28.4
```

Example Suppose an angle a is given as a positive real number of degrees. If 0 < a < 90 then it is quadrant 1.

If 90 < a < 180 then it is quadrant 2.

If 180 < a < 270 then it is quadrant 3.

If 270 < a < 360 then it is quadrant 4.

Write a function quadrant(alpha) which returns the quadrant of the angle a. Try your functions for a equals to 50.

```r
quadrant <- function(alpha)
{
1 + (alpha%%360)%/%90
}
quadrant(50)
```

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

Example

Consider the continuous function

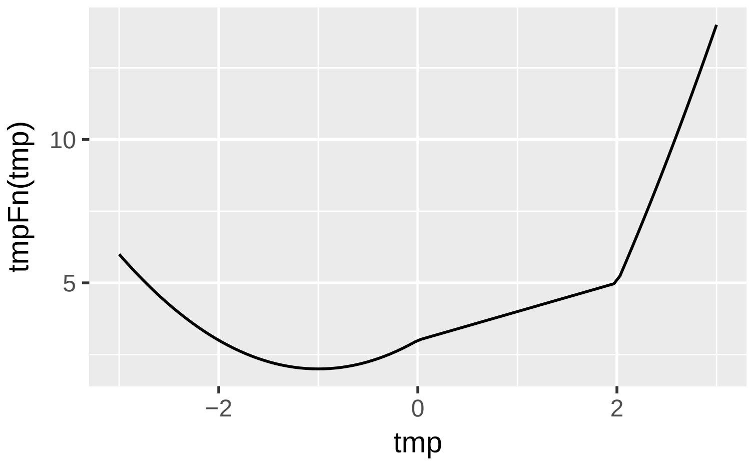
$$f(x) = \begin{cases} x^2 + 2x + 3 & \text{if } x < 0 \\ x + 3 & \text{if } 0 \le x < 2 \\ x^2 + 4x - 7 & \text{if } 2 \le x. \end{cases}$$

Write a function `tmpFn` which takes a single argument `xVec`. The function should return the vector of values of the function f(x) evaluated at the values in `xVec`.

Hence plot the function f(x) for -3 < x < 3.

```r
tmpFn <- function(x)
{
ifelse(x < 0, x^2 + 2*x + 3, ifelse(x < 2, x+3, x^2 + 4*x - 7))
}
tmp <- seq(-3, 3, len=100)
data<-data.frame(tmp,tmpFn(tmp))
```

```r
library(ggplot2)
ggplot(data,aes(x=tmp,y=tmpFn(tmp)))+geom_line()
```



Example

Write a function to compute the number of nonmissing and missing observations in a given vector. (Please try your functions for any vector.)

```
samp.size <- function(x) {
n_na <- sum(is.na(x))
n_obs <- length(x) - n_na
out <- cbind(n_obs, n_na)
return(out)
}
nums <- c(1:23,rep(NA,2))
samp.size(nums)
```

```
##      n_obs n_na
## [1,]    23    2
```

Probability Distributions in R

The R programming includes many statistical distributions and their functions. There are 4 types of functions for every distribution.

`p` for "probability", the cumulative distribution function (c.d.f.)

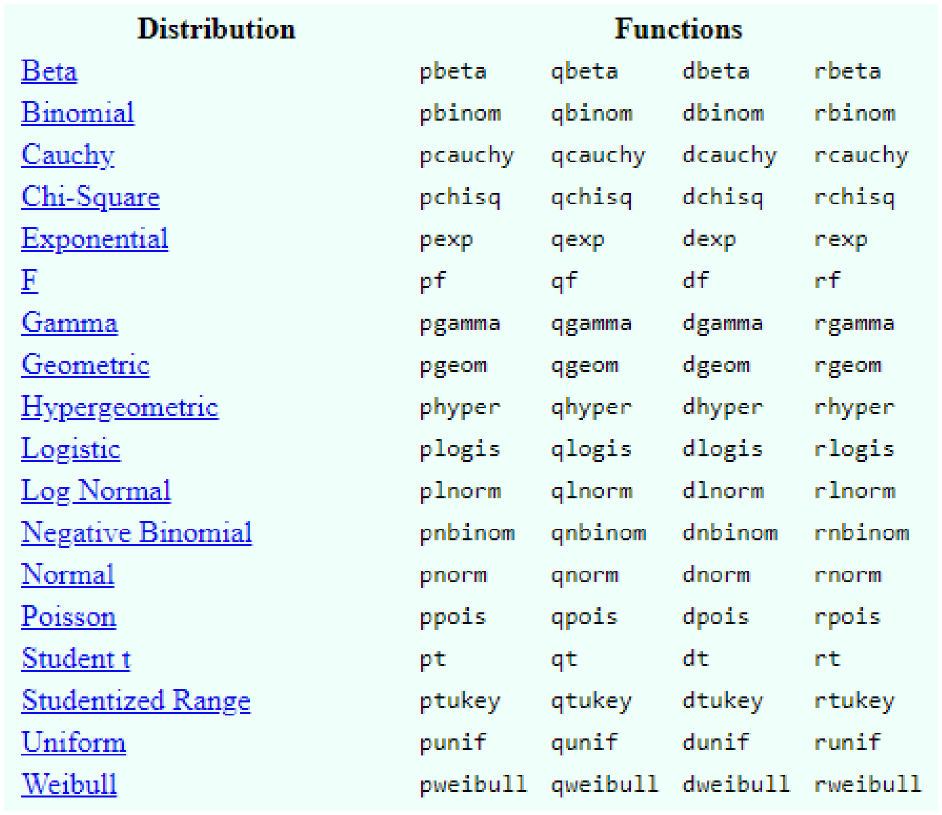
`q` for "quantile", the inverse c.d.f.

`d` for "density", the density function (p.f. or p.d.f.)

`r` for "random", a random variable having the specified distribution.

How to use these functions

For the normal distribution, these functions are `pnorm`, `qnorm`, `dnorm`, and `rnorm`. For the binomial distribution, these functions are `pbinom`, `qbinom`, `dbinom`, and `rbinom`. And so forth.



Distribution	Functions			
Beta	pbeta	qbeta	dbeta	rbeta
Binomial	pbinom	qbinom	dbinom	rbinom
Cauchy	pcauchy	qcauchy	dcauchy	rcauchy
Chi-Square	pchisq	qchisq	dchisq	rchisq
Exponential	pexp	qexp	dexp	rexp
F	pf	qf	df	rf
Gamma	pgamma	qgamma	dgamma	rgamma
Geometric	pgeom	qgeom	dgeom	rgeom
Hypergeometric	phyper	qhyper	dhyper	rhyper
Logistic	plogis	qlogis	dlogis	rlogis
Log Normal	plnorm	qlnorm	dlnorm	rlnorm
Negative Binomial	pnbinom	qnbinom	dnbinom	rnbinom
Normal	pnorm	qnorm	dnorm	rnorm
Poisson	ppois	qpois	dpois	rpois
Student t	pt	qt	dt	rt
Studentized Range	ptukey	qtukey	dtukey	rtukey
Uniform	punif	qunif	dunif	runif
Weibull	pweibull	qweibull	dweibull	rweibull

In order to find out how to use these, you can visit https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Distributions.html

Generating random numbers from distributions

By **random number**, we mean the numbers supposedly coming from a specific probability distribution. To generate number, rfunction is used.

```
> z=runif(5)       # generates 5 numbers from U(0,1)
> z
[1] 0.3486722 0.5405154 0.7346775 0.7533214 0.2895739

> bg = rnorm(3,2,1)  # generates 3 numbers from N(2,1)
> bg
[1] 3.6665352 0.6932574 1.2995387

> e1 = rbinom(20,10,0.4) # generates 20 numbers from Binomial(10,0.4)
> e1
 [1] 4 2 4 4 2 3 8 2 3 2 6 3 6 2 6 7 5 3 4 6

> e1 = rbinom(20,10,0.4) # generates 20 numbers from Binomial(10,0.4)
> e1
 [1] 2 3 2 4 1 3 5 2 4 5 2 4 3 3 4 4 5 2 5 3
```

As you can see, although we write the same code we obtain the different numbers. How to we fix it?

`set.seed(94302)` : you can type any number in it. It will be the starting point in the internal counter of the computer. Specifying a number with the `set.seed` command enables you to generate exact same number each time you run your code.

```
> set.seed(1234)
> rnorm(5,0,1)
[1] -1.2070657  0.2774292  1.0844412 -2.3456977  0.4291247
> set.seed(1234)
> rnorm(5,0,1)
[1] -1.2070657  0.2774292  1.0844412 -2.3456977  0.4291247
> set.seed(123)
> rnorm(5,0,1)
[1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

Calculating Probability

You can compute probabilities easily by using R. In order to compute probabilities, you can use `p` -probability- and `d`-density- functions.

How to use it?

For example, you want to calculate P(X=7) where X~Bin(n=10,p=0.3)

```
> dbinom(7,10,0.3)
[1] 0.009001692
```

If you want to calculate the cumulative probability of X~Bin(n=10,p=0.3)

```
> pbinom(7,10,0.3,lower.tail = TRUE)
[1] 0.9984096
```

If you want to calculate P(X>7) where X~Bin(n=10,p=0.3)

```
> pbinom(7,10,0.3,lower.tail = F)
[1] 0.001590386
```

`lower.tail`logical; if `TRUE (default)`, probabilities are P[X ??? x], otherwise, P[X > x].

If you try to calculate the probability from continuous distribution,

```
> pexp(1,10,lower.tail = T)
[1] 0.9999546
```

Example

Set your seed to 1 and generate 10 random numbers using `runif` and save it in an object called ‘`random_numbers`.

```
set.seed(1)
random_numbers <- runif(10)
```

Example

Using the function `ifelse` and the object `random_numbers` simulate `coin tosses`. Hint: If `random_numbers` is bigger than .5 then the result is head, otherwise is tail.

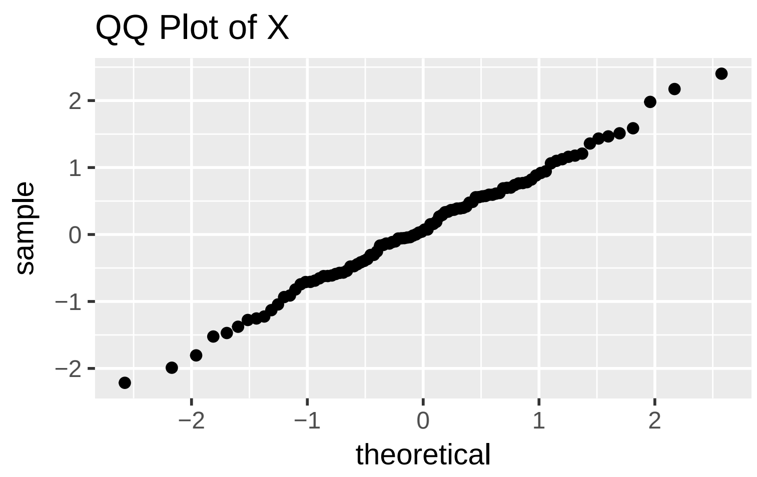
```
coin_tosses_1 <- ifelse(random_numbers>.5, 'head', 'tail')
```

Example

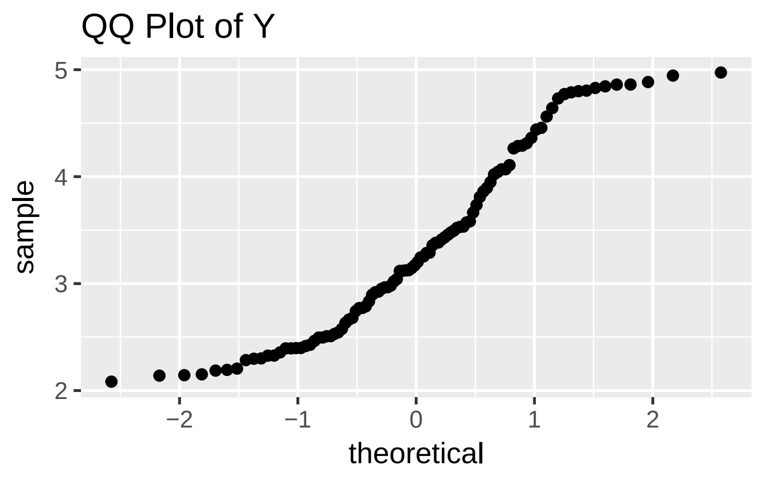
The simulated data are given as the following: X~norm(100,0,1) ; Y~Unif(100,2,5)

According to this, a) Check normality assumption using both qqnorm. b) Compare two variances.

```r
x<-rnorm(100,0,1)
y<-runif(100,2,5)
#to use ggplot packages, I need to combine them in one data frame.
data<-data.frame(x,y)
library(ggplot2)
ggplot(data, aes(sample=x))+stat_qq()+labs(title="QQ Plot of X")
```



```r
ggplot(data, aes(sample=y))+stat_qq()+labs(title="QQ Plot of Y")
```



```r
#Compare the variance values
var_x<-var(x)
var_y<-var(y)
cbind(var_x,var_y)
```

```
##          var_x     var_y
## [1,] 0.7905585 0.7378776
```

Example

Simulate normal distribution values. Imagine a population in which the average height is 1.70 m with an standard deviation of 0.1, using `rnorm` simulate the height of 100 people and save it in an object called `heights`.

To get an idea of the values of heights applying the function `summary` to it.

```
set.seed(1)
heights <- rnorm(n = 100, mean = 1.70, sd = .1)
summary(heights)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.479   1.651   1.711   1.711   1.769   1.940
```

Example By using the simulated data set in the previous question calculate the following probabilites. a) What's the probability that a person will be smaller or equal to 1.90 m ? b) What's the probability that a person will be taller or equal to 1.60 m?

```
#a
pnorm(1.90, mean = 1.70, sd = .1)
```

```
## [1] 0.9772499
```

```
#b
1 - pnorm(1.60, mean = 1.70, sd = .1)
```

```
## [1] 0.8413447
```

```
#OR
pnorm(1.60, mean = 1.70, sd = .1,lower.tail = F)
```

```
## [1] 0.8413447
```

Example The suicide rate in a certain state is 1 suicide per 100000 inhabitants per month. Find the probability that, in a city of 400000 inhabitants within this state, there will be 8 suicides in a given month.

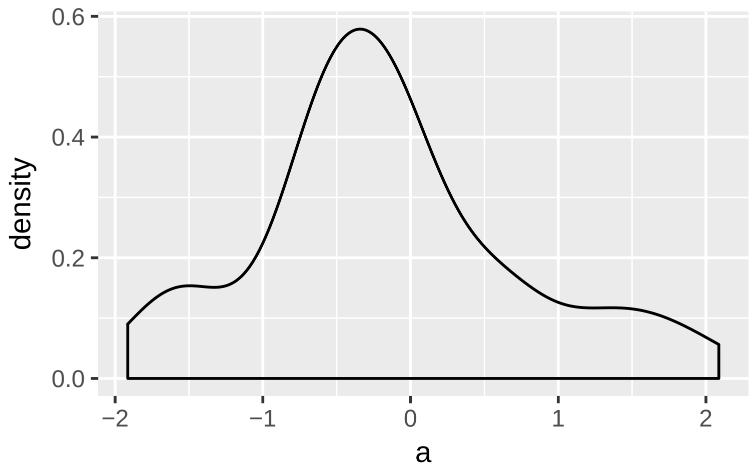
```
prob<-dbinom(8,400000,1/100000)
prob
```

```
## [1] 0.02976988
```

Plotting Probability Distributions

You can draw the density plot of your generated numbers by using `geom_density()` function in ggplot2.

```
a<-rnorm(50,0,1)
data<-data.frame(a)
ggplot(data,aes(x=a))+geom_density()
```



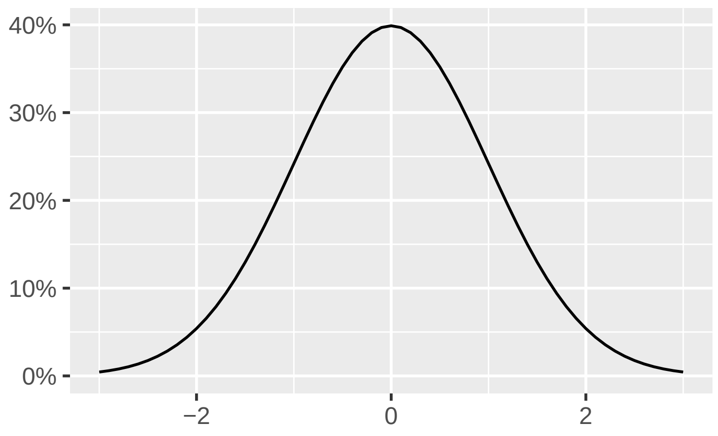
Also, `ggdistribution` is a helper function to plot Distributions in the stats package easier using `ggfortify`.

For example, plot standard normal distribution from -3 to 3

```r
library(ggfortify)
```

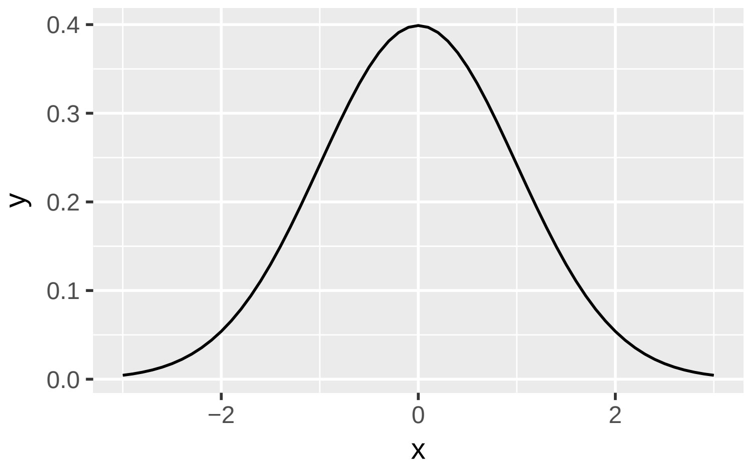
```
## Warning: namespace 'DBI' is not available and has been replaced
## by .GlobalEnv when processing object 'plot.index'

## Warning: namespace 'DBI' is not available and has been replaced
## by .GlobalEnv when processing object 'plot.index'
```

```r
ggdistribution(dnorm, seq(-3, 3, 0.1), mean = 0, sd = 1)
```



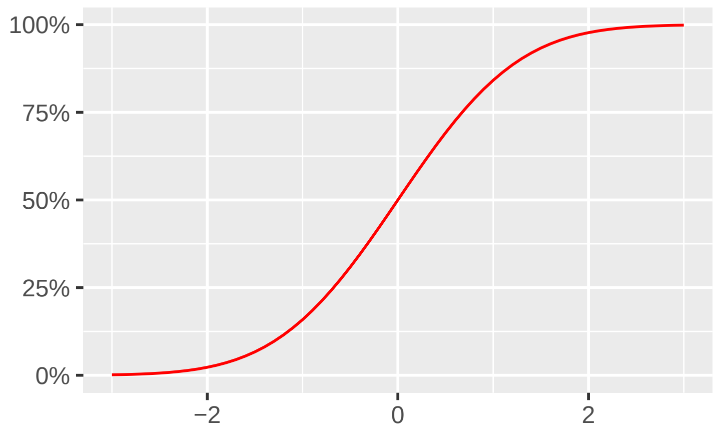
You can draw the same plot in this way,

```r
x<-seq(-3, 3, 0.1)
y<-dnorm(x,0,1)
data<-data.frame(x,y) #Since I use ggplot
ggplot(data,aes(x=x,y=y))+geom_line()
```



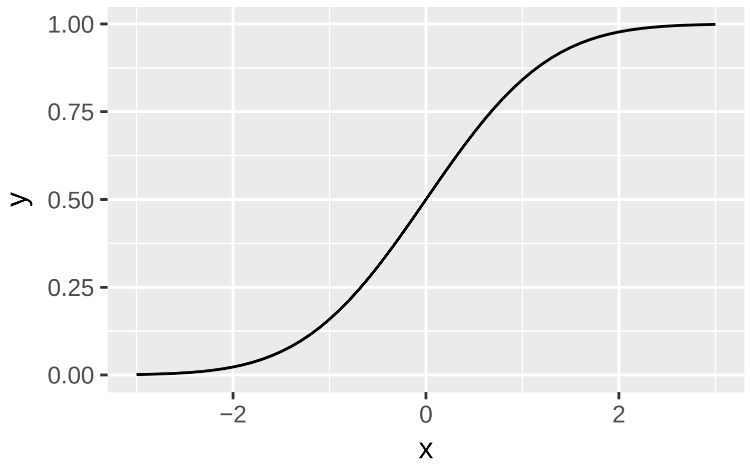
`ggdistribution` accepts PDF/CDF function, sequence, and options passed to PDF-CDF function. Also, it has some options to configure how plot looks. Use `help(ggdistribution)` to check available options.

```r
ggdistribution(pnorm, seq(-3, 3, 0.1), mean = 0, sd = 1, colour = 'red')
```

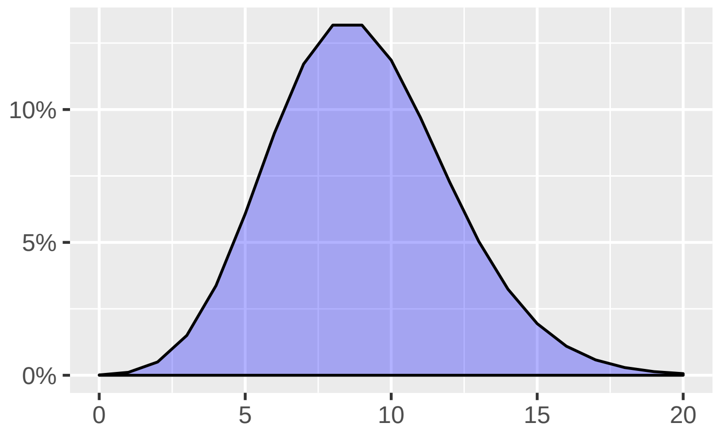


You can also draw the same plot by using `ggplot` function.

```
x<-seq(-3, 3, 0.1)
y<-pnorm(x,0,1)
data<-data.frame(x,y) #Since I use ggplot
ggplot(data,aes(x=x,y=y))+geom_line()
```

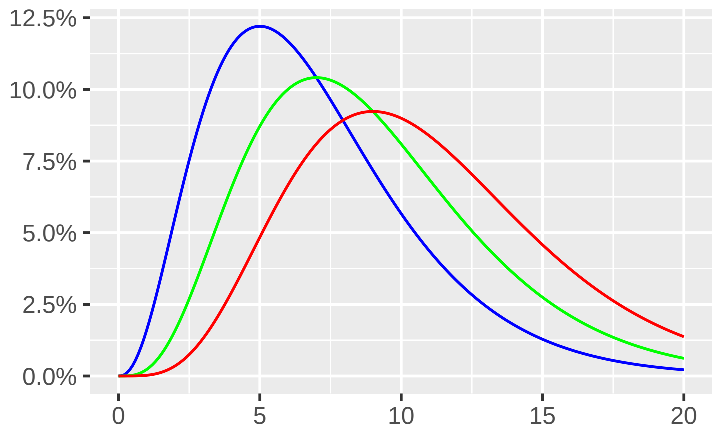


```
ggdistribution(dpois, seq(0, 20), lambda = 9, fill = 'blue')
```



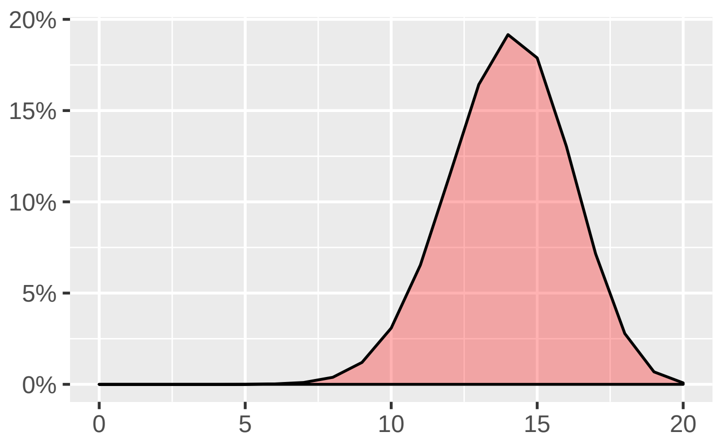
If you want to plot some distributions overwrapped, use p keyword to pass ggplot instance.

```
p <- ggdistribution(dchisq, seq(0, 20, 0.1), df = 7, colour = 'blue')
p <- ggdistribution(dchisq, seq(0, 20, 0.1), df = 9, colour = 'green', p = p)
ggdistribution(dchisq, seq(0, 20, 0.1), df = 11, colour = 'red', p = p)
```



Example Assume that 20 biased coin flips with probability 0.7. and plot the mass function of X.

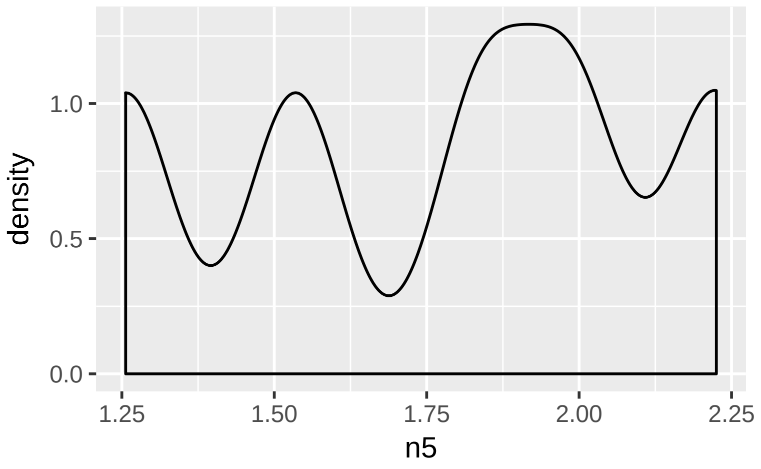
```
ggdistribution(dbinom, seq(0, 20),size=20, prob=0.7, fill = "red")
```



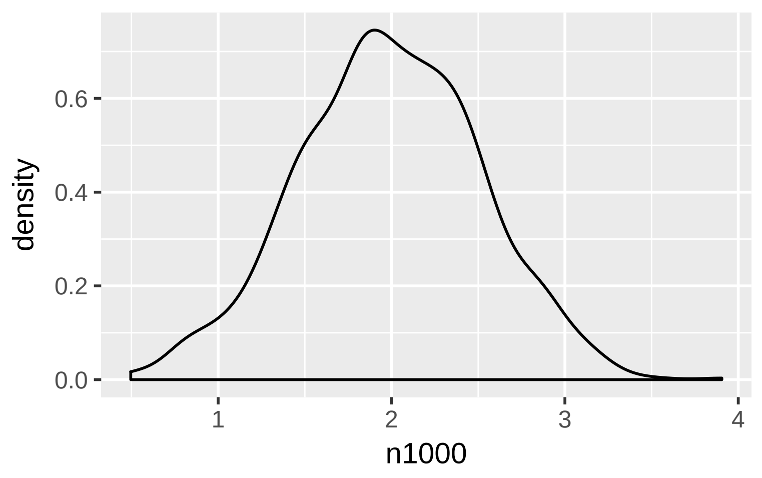
Example

Generate 5 random numbers from normal distribution with mean 2, standard deviaton 0.5 Then, generate 1000 random numbers from normal distribution with same parameter values. Then, draw them their density plot. What can we say about this plot?

```
n5<-rnorm(5,2,0.5)
n1000<-rnorm(1000,2,0.5)
data<-data.frame(n5,n1000)
ggplot(data,aes(x=n5))+geom_density()
```



```
ggplot(data,aes(x=n1000))+geom_density()
```



apply Family in R

The `apply()` family belongs to the R base package and is populated with functions to manipulate slices of data from matrices, arrays, lists and dataframes in a repetitive way. These functions allow crossing the data in a number of ways and avoid explicit use of loop constructs. They act on an input list, matrix or array and apply a named function with one or several optional arguments.

The called function could be:

+An aggregating function, like for example the mean, or the sum (that return a number or scalar);

+Other transforming or subsetting functions; and

+Other vectorized functions, which return more complex structures like lists, vectors, matrices and arrays.

The `apply()` functions are useful for performing operartions with very few lines of code instead of constructing complex loops. This family contains `apply()`, `lapply()` , `sapply()`, `vapply()`, `mapply()`, `rapply()`, and `tapply()` functions.

How and when should we use these?

This depends on the structre of the dataset and the output that you need.

apply() in R

This function is "godfather" of this family. It operates on arrays. For simplicity, the tutorial limits itself to 2D arrays, which are also known as matrices.

`apply(X, MARGIN, FUN, ...)`

Consider `cars` data set in R.

```
head(cars)  #R built-in dataset
```

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

```
#Sum up for each row:
apply(cars,1,sum)
```

```
##  [1]   6  14  11  29  24  19  28  36  44  28  39  26  32  36  40  39  47
## [18]  47  59  40  50  74  94  35  41  69  48  56  49  57  67  60  74  94
## [35] 102  55  65  87  52  68  72  76  84  88  77  94 116 117 144 110
```

```
#Sum up for each column:
apply(cars,2,sum)
```

```
## speed  dist
##   770  2149
```

It is also possibe to write a user defined function in R by using `apply`

For example, the following function is used to multiply each all values by 10.

```
> ex<-apply(cars,1:2,function(x) 10 * x)
> head(ex)
     speed dist
[1,]    40   20
[2,]    40  100
[3,]    70   40
[4,]    70  220
[5,]    80  160
[6,]    90  100
```

The `lapply()` Function

If you want to apply a function to every entry in a list or in a vector, we can use `lapply()` function. When you execute `?lapply`, you see that the syntax looks like the `apply()` function.

`>lapply(X, FUN, ...)`

The difference is that:

+It can be used for other objects like dataframes, lists or vectors; and

+The output returned is a list (which explains the "l" in the function name), which has the same number of elements as the object passed to it.

```
lapply(cars,sum)
```

```
## $speed
## [1] 770
##
## $dist
## [1] 2149
```

```
lapply(cars,mean)
```

```
## $speed
## [1] 15.4
##
## $dist
## [1] 42.98
```

Another example,

```
data <- list(x = 1:5, y = 6:10, z = 11:15)
data
```

```
## $x
## [1] 1 2 3 4 5
##
## $y
## [1]  6  7  8  9 10
##
## $z
## [1] 11 12 13 14 15
```

```
lapply(data,median)
```

```
## $x
## [1] 3
##
## $y
## [1] 8
##
## $z
## [1] 13
```

The `sapply()` function

The `sapply()` function works like `lapply()`, but it tries to simplify the output to the most elementary data structure that is possible. In other words, `sapply` is the same as `lapply`, but returns a vector instead of a list.

```
sapply(X, FUN, ...)
```

Consider the data created in the previous part.

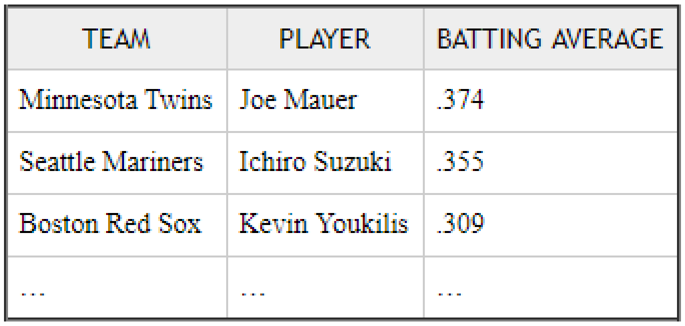
```
sapply(data, median)
```

```
##  x  y  z
##  3  8 13
```

The `tapply()` function

In statistics, one of the most basic activities we do is computing summaries of variables. These summaries might be as simple as an average, or more complex. Let's look at some simple examples. When you read the results of a medical trial, you will see things such as "The average age of subjects in this trial was 55 years in the treatment group, and 54 years in the control group." As another example, let's look at one from the world of baseball.

Batting Leaders per Team



TEAM	PLAYER	BATTING AVERAGE
Minnesota Twins	Joe Mauer	.374
Seattle Mariners	Ichiro Suzuki	.355
Boston Red Sox	Kevin Youkilis	.309
...	...	...

These two examples have a lot in common, even if they don't appear to when first reading. In the first example, we have a dataset from a medical trial. We want to break up the dataset into two groups, treatment and control, and then compute the sample average for age within each group.

In the second example, we want to break up the dataset into 30 groups, one for each MLB team, and then compute the maximum batting average within each group.

So what is in common?

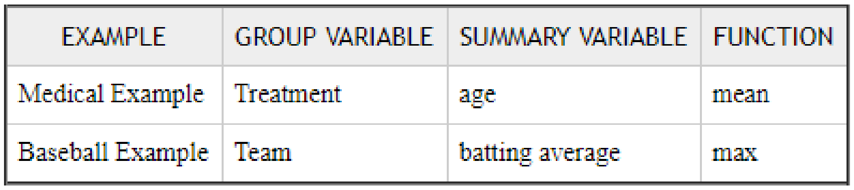
In each case we have

+A dataset that can be broken up into groups

+We want to break it up into groups

+Within each group, we want to apply a function

The following table summarizes the situation.



EXAMPLE	GROUP VARIABLE	SUMMARY VARIABLE	FUNCTION
Medical Example	Treatment	age	mean
Baseball Example	Team	batting average	max

The `tapply()` function can solve both of these problems for us!

Logic behind the usage of this function is as follows, `tapply(Summary Variable, Group Variable, Function)`

```
## generate data for medical example
medical.example <-data.frame(patient = 1:100,age = rnorm(100, mean = 60, sd = 12), treatment = gl(2, 50
summary(medical.example)
```

```
##     patient            age             treatment
##  Min.   :  1.00   Min.   :21.44   Treatment:50
##  1st Qu.: 25.75   1st Qu.:53.40   Control  :50
##  Median : 50.50   Median :59.24
##  Mean   : 50.50   Mean   :60.60
##  3rd Qu.: 75.25   3rd Qu.:69.52
##  Max.   :100.00   Max.   :91.28
```

`gl()` generates factors by specifying the pattern of their levels.

```
## generate data for baseball example
## 5 teams with 5 players per team
baseball.example <-data.frame(team = gl(5, 5,labels = paste("Team", LETTERS[1:5])), player = sample(let
summary(baseball.example)
```

```
##      team       player   batting.average
##  Team A:5   a      : 1   Min.   :0.2288
##  Team B:5   b      : 1   1st Qu.:0.2549
##  Team C:5   c      : 1   Median :0.3260
##  Team D:5   d      : 1   Mean   :0.3157
##  Team E:5   e      : 1   3rd Qu.:0.3680
##             f      : 1   Max.   :0.3960
##             (Other):19
```

```
## tapply(Summary Variable, Group Variable, Function)
## Medical Example
tapply(medical.example$age, medical.example$treatment, mean)
```

```
## Treatment   Control
##  58.55391  62.64686
```

```
## Baseball Example
tapply(baseball.example$batting.average, baseball.example$team,max)
```

```
##    Team A    Team B    Team C    Team D    Team E
## 0.3893942 0.3910864 0.3959520 0.3779970 0.3503441
```

Statistical Inference

Hypothesis Testing

A hypothesis is a statement on the population paramater. The aim of the hypoythesis test is to decide which one of the complementary hypothess is true based on sample values from the population.

One sample Z-test

It is a test that used to test the hypothesis about the sample mean. It is only used to test sample mean. Here is the type of hypotheses

Ho: µ1=µ vs H1= µ1 != µ (Two Tail)

Ho: µ1<=µ vs H1= µ1> µ (Right Tail)

Ho: µ1>=µ vs H1= µ1=µ vs H1= µ1

One Sample T-test

The one-sample t-test is used to determine whether a sample comes from a population with a specific mean. This population mean is not always known, but is sometimes hypothesized. It is used when sample mean and variance is unknown.

When we use student t- test

-Data points should be independent from each other.

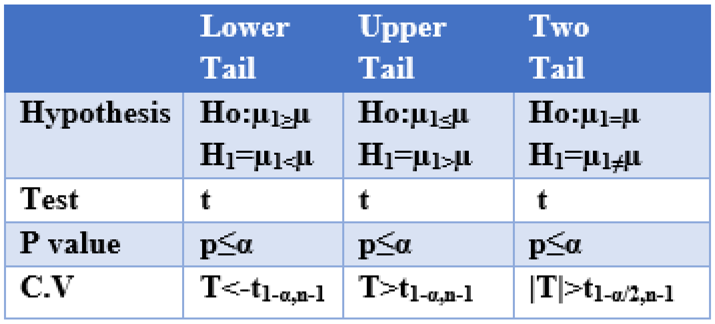
-Your data should be normally distributed.

-Your data should be randomly selected from a population, where each item has an equal chance of being selected.

t= $\frac{\bar{x}-\mu}{s/n}$

In this equation, $\bar{x}$ is sample mean, μ is the population mean given in the question, s is the standard deviation of the sample.

The rejection rules for t-test are as follows;



	Lower Tail	Upper Tail	Two Tail
Hypothesis	$H_o: \mu_1 \geq \mu$ $H_1 = \mu_1 < \mu$	$H_o: \mu_1 \leq \mu$ $H_1 = \mu_1 > \mu$	$H_o: \mu_1 = \mu$ $H_1 = \mu_1 \neq \mu$
Test	t	t	t
P value	$p \leq \alpha$	$p \leq \alpha$	$p \leq \alpha$
C.V	$T < -t_{1-\alpha,n-1}$	$T > t_{1-\alpha,n-1}$	$\lvert T \rvert > t_{1-\alpha/2,n-1}$

Consider mtcars data set. We want to test whether mean value of mpg is different than 15 or not.

```r
#Checking normality at first
shapiro.test(mtcars$mpg)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mtcars$mpg
## W = 0.94756, p-value = 0.1229
```

```r
t.test(mtcars$mpg,alternative ="two.sided",mu=15)
```

```
##
##  One Sample t-test
##
## data:  mtcars$mpg
## t = 4.778, df = 31, p-value = 4.054e-05
## alternative hypothesis: true mean is not equal to 15
## 95 percent confidence interval:
##  17.91768 22.26357
## sample estimates:
## mean of x
##  20.09062
```

We want to test whether mean value of mpg is less than 15 or not.

```r
t.test(mtcars$mpg,alternative ="less",mu=15)
```

```
##
##  One Sample t-test
##
## data:  mtcars$mpg
## t = 4.778, df = 31, p-value = 1
## alternative hypothesis: true mean is less than 15
## 95 percent confidence interval:
##      -Inf 21.89707
## sample estimates:
## mean of x
##  20.09062
```

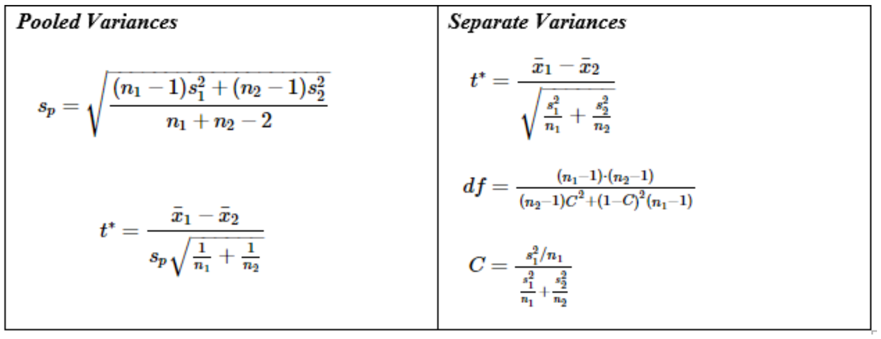
We want to test whether mean value of mpg is greater than 15 or not.

```r
t.test(mtcars$mpg,alternative ="greater",mu=15)
```

```
##
##  One Sample t-test
##
## data:  mtcars$mpg
## t = 4.778, df = 31, p-value = 2.027e-05
## alternative hypothesis: true mean is greater than 15
## 95 percent confidence interval:
##  18.28418      Inf
## sample estimates:
## mean of x
##  20.09062
```

Two-Sample-T-test

A two-sample t-test is used when you want to compare two independent groups to see if their means are different. There are two options for estimating the variances for the 2-sample t-test with independent samples; using pooled variances or using separate variances.



Pooled Variances	***Separate Variances***
$s_p = \sqrt{\frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-2}}$	$t^* = \frac{\bar{x}_1-\bar{x}_2}{\sqrt{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}}$
$t^* = \frac{\bar{x}_1-\bar{x}_2}{s_p\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}}$	$df = \frac{(n_1-1)\cdot(n_2-1)}{(n_2-1)C^2+(1-C)^2(n_1-1)}$
	$C = \frac{s_1^2/n_1}{\frac{s_1^2}{n_1}+\frac{s_2^2}{n_2}}$

Assumptions

-Samples should be taken from Normal Distribution randomly with unknown variances.

-Two samples must be independent of each other.

Example

subjects were given a drug (treatment group) and an additional 6 subjects a placebo (control group). Their reaction time to a stimulus was measured (in ms). We want to perform a two-sample t-test for comparing the means of the treatment and control groups.

Control = 91, 87, 99, 77, 88, 91

Treat = 101, 110, 103, 93, 99, 104

Ho: µ1 ≤ µ2 vs H1= µ1

It is seen that the average values of samples represented by median are different for each cases. It is an indication of to reject the null hypothesis.

Then, check assumption!

Normality Assumption,

```r
#Check Normality for each sample
shapiro.test(cholesterol$response[cholesterol$trt=="1time"])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  cholesterol$response[cholesterol$trt == "1time"]
## W = 0.93063, p-value = 0.4541
```

```
#Since p value is greater than our significance level taken 0.05,
#the response variable follows normal distribution.
```

```
shapiro.test(cholesterol$response[cholesterol$trt=="2times"])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  cholesterol$response[cholesterol$trt == "2times"]
## W = 0.9432, p-value = 0.5892
```

```
#Since p value is greater than our significance level taken 0.05,
#the response variable follows normal distribution.
```

```
shapiro.test(cholesterol$response[cholesterol$trt=="4times"])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  cholesterol$response[cholesterol$trt == "4times"]
## W = 0.95487, p-value = 0.7262
```

```
#Since p value is greater than our significance level taken 0.05,
#the response variable follows normal distribution.
```

```
shapiro.test(cholesterol$response[cholesterol$trt=="drugD"])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  cholesterol$response[cholesterol$trt == "drugD"]
## W = 0.93406, p-value = 0.489
```

```
#Since p value is greater than our significance level taken 0.05,
#the response variable follows normal distribution.
```

```
shapiro.test(cholesterol$response[cholesterol$trt=="drugE"])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  cholesterol$response[cholesterol$trt == "drugE"]
## W = 0.98085, p-value = 0.9696
```

```
#Since p value is greater than our significance level taken 0.05,
#the response variable follows normal distribution.
```

Homogenity of Variance,

In this assumption, our null hypothesis is that all samples have equal variances and alternative hypothesis suggests that at least one of them has different variance. Our aim is to be failed to reject the null hypothesis.

```
bartlett.test(response~trt,data=cholesterol) #Since p value is greater than 0.05, it can be said that
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  response by trt
## Bartlett's K-squared = 0.57975, df = 4, p-value = 0.9653
```

```r
#there is no evidence to conclude that the samples have different variance values.
```

After checking assumptions, the One-Way Anova is conducted as follows,

```r
fit<-aov(response~trt,data=cholesterol) #fitting anova model
summary(fit)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)    
## trt          4 1351.4   337.8   32.43 9.82e-13 ***
## Residuals   45  468.8    10.4                     
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Studying the output of the ANOVA table above we see that the F-statistic is 32.43 with a p-value equal to 9.82e-13 . We clearly reject the null hypothesis of equal means for all five drug groups.

The ANOVA F-test answers the question whether there are significant differences in the K population means. However, it does not provide us with any information about how they differ. Therefore, when you reject H0 in ANOVA, additional analyses are required to determine what is driving the difference in means. There are several tests in the literature such as LSD test, Tukey Test, pairwise t-test..

```r
TukeyHSD(fit)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = response ~ trt, data = cholesterol)
##
## $trt
##                   diff        lwr       upr     p adj
## 2times-1time   3.44300 -0.6582817  7.544282 0.1380949
## 4times-1time   6.59281  2.4915283 10.694092 0.0003542
## drugD-1time    9.57920  5.4779183 13.680482 0.0000003
## drugE-1time   15.16555 11.0642683 19.266832 0.0000000
## 4times-2times  3.14981 -0.9514717  7.251092 0.2050382
## drugD-2times   6.13620  2.0349183 10.237482 0.0009611
## drugE-2times  11.72255  7.6212683 15.823832 0.0000000
## drugD-4times   2.98639 -1.1148917  7.087672 0.2512446
## drugE-4times   8.57274  4.4714583 12.674022 0.0000037
## drugE-drugD    5.58635  1.4850683  9.687632 0.0030633
```

In this test, each output is obtained from indivual tests. For example, the first test is about the difference between drugs 1time and drugs 2times. The null hypothesis suggests that there is no difference between these samples while the alternative hypothesis suggests that there is a difference between them. To make a decision, the p value is observed and since p value is greater than our significance value, it can be said that there is no significant difference between these two samples. The other tests are interpreted in the same way.

In addition to Tukey Test, pairwise t test is also helpful for this purpose. The syntax of this test is as follows,

```
pairwise.t.test(reponse, factor, p.adjust = method, alternative = c("two.sided", "less", "greater"))
```

```r
pairwise.t.test(cholesterol$response, cholesterol$trt, p.adjust = "bonferroni")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  cholesterol$response and cholesterol$trt
##
##        1time   2times  4times  drugD
## 2times 0.21333 -       -       -
## 4times 0.00038 0.34352 -       -
## drugD  3.5e-07 0.00106 0.44316 -
## drugE  1.1e-12 2.3e-09 3.8e-06 0.00348
##
## P value adjustment method: bonferroni
```

```
#We prefer bonferroni method, because it gives the narrowest confidence interval.
```

Each enrty in this matrix shows the p-value of the tests where the null hypothesis that there is no difference between samples. It is seen that the results are matched with one coming from Tukey HSD test.

Linear Regression

Note: This part is created using notes edited by Selva Prabhakaran

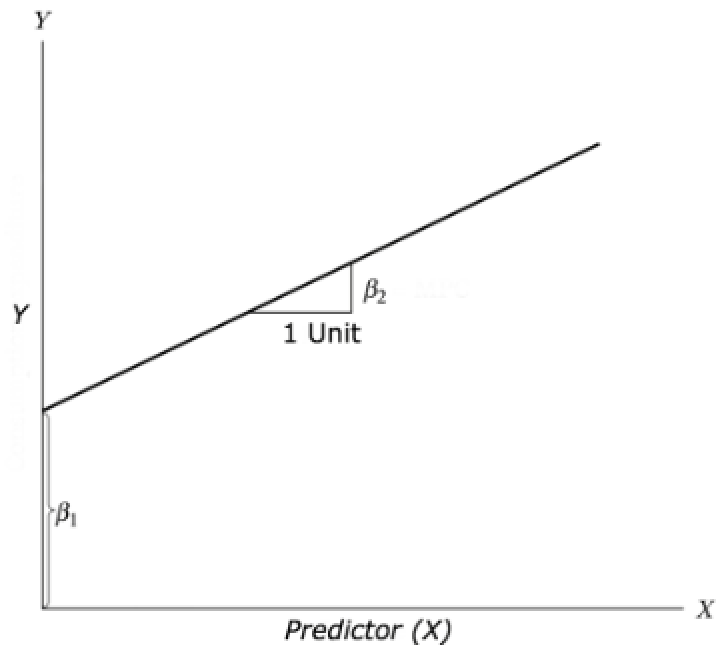
Linear regression is used to predict the value of an outcome variable Y based on one or more input predictor variables X. The aim is to establish a linear relationship (a mathematical formula) between the predictor variable(s) and the response variable, so that, we can use this formula to estimate the value of the response Y, when only the predictors (Xs) values are known.

Introduction

The aim of linear regression is to model a continuous variable Y as a mathematical function of one or more X variable(s), so that we can use this regression model to predict the Y when only the X is known. This mathematical equation can be generalized as follows:

$Y = \beta 1 + \beta 2X + ??$

where, ??1 is the intercept and ??2 is the slope. Collectively, they are called regression coefficients. ?? is the error term, the part of Y the regression model is unable to explain.



Simple Linear Regression

When the regression model contains one dependent variable and one independent variable, we call the approach simple linear regression.

$y=\beta o+\beta 1^* \ x1+\varepsilon$

where βo is the y-intercept,$\beta 1$is the slope (or regression coefficient), and ε is the error term.

Main question of Interest: Are the response and independent variable related linearly? Can we model the relationship between them using a linear regression?

Object: To explore the effect of independent variable X on the response variable Y.

Example

For this analysis, we will use the cars dataset that comes with R by default. `cars` is a standard built-in dataset, that makes it convenient to demonstrate linear regression in a simple and easy to understand fashion. You can access this dataset simply by typing in cars in your R console. You will find that it consists of 50 observations(rows) and 2 variables (columns) - dist and speed. Lets print out the first six observations here..

```r
head(cars)
```

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

Our aim is to predict distance value by using speed information. For this purpose, a linear model is constructed with `lm()` function. This function is used to construct a simple linear or multiple linear model. However, before we begin building the regression model, it is a good practice to analyze and understand the variables. The graphical analysis and correlation study below will help with this.

Graphical Analysis

The aim of this example is to build a simple regression model that we can use to predict Distance (dist) by establishing a statistically significant linear relationship with Speed (speed). But before jumping in to the syntax, lets try to understand these variables graphically. Typically, for each of the independent variables (predictors), the following plots are drawn to visualize the following behavior:

Scatter plot: Visualize the linear relationship between the predictor and response

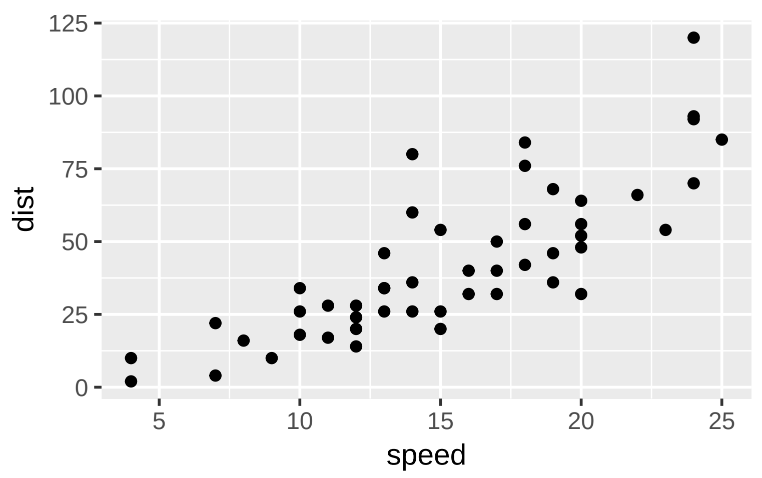
Box plot: To spot any outlier observations in the variable. Having outliers in your predictor can drastically affect the predictions as they can easily affect the direction/slope of the line of best fit.

Density plot: To see the distribution of the predictor variable. Ideally, a close to normal distribution (a bell shaped curve), without being skewed to the left or right is preferred. Let us see how to make each one of them.

Scatter Plot

Scatter plots can help visualize any linear relationships between the dependent (response) variable and independent (predictor) variables. Ideally, if you are having multiple predictor variables, a scatter plot is drawn for each one of them against the response, along with the line of best as seen below.

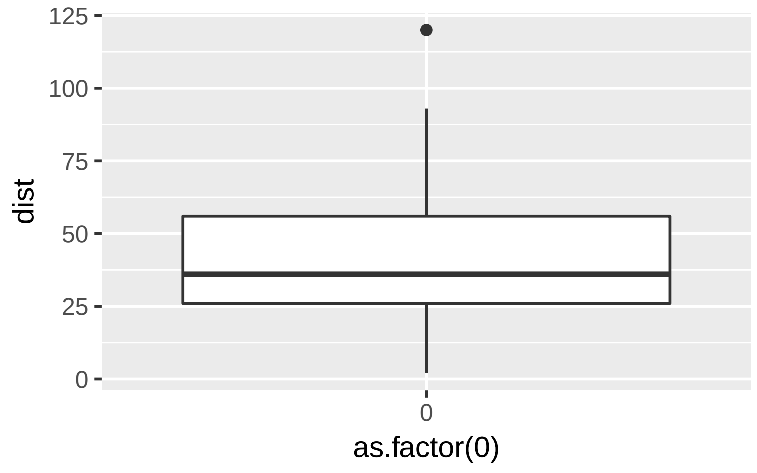
```r
ggplot(cars, aes(x=speed, y=dist)) + geom_point()
```



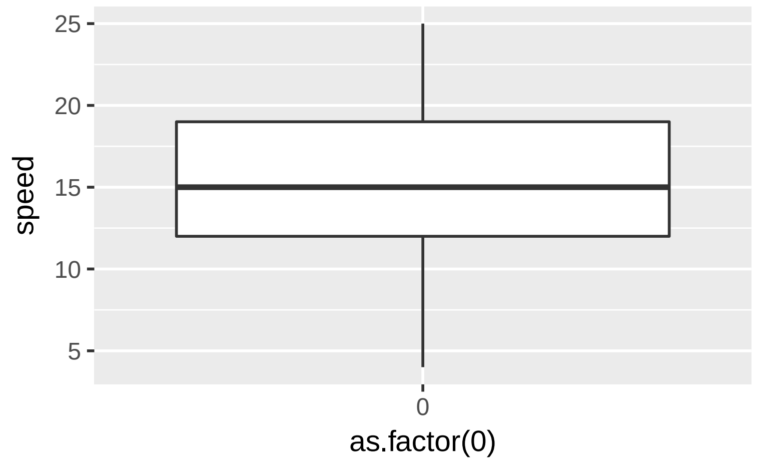
The scatter plot along with the smoothing line above suggests a linearly increasing relationship between the 'dist' and 'speed' variables. This is a good thing, because, one of the underlying assumptions in linear regression is that the relationship between the response and predictor variables is linear and additive.

BoxPlot *Check for outliers* Generally, any datapoint that lies outside the $1.5 * interquartile - range(1.5 * IQR)$ is considered an outlier, where, IQR is calculated as the distance between the 25th percentile and 75th percentile values for that variable.

```r
ggplot(cars,aes(x=as.factor(0),y=dist))+geom_boxplot()
```



```
ggplot(cars,aes(x=as.factor(0),y=speed))+geom_boxplot()
```

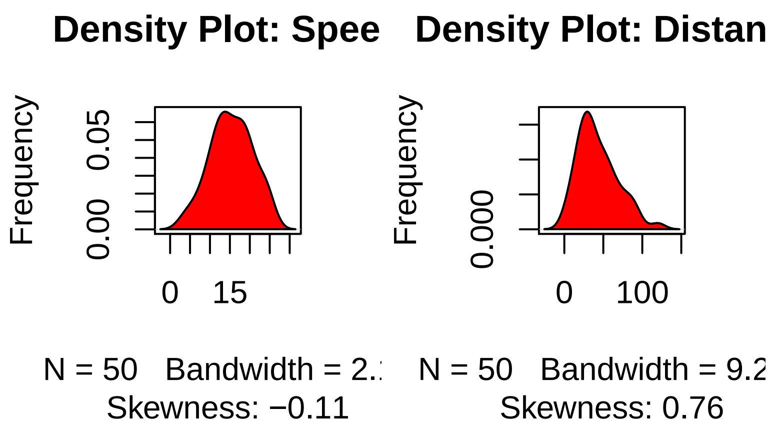


Density plot *Check if the response variable is close to normality*

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.4.3
```

```
par(mfrow=c(1, 2))  # divide graph area in 2 columns
plot(density(cars$speed), main="Density Plot: Speed", ylab="Frequency", sub=paste("Skewness:", round(e1
polygon(density(cars$speed), col="red")
plot(density(cars$dist), main="Density Plot: Distance", ylab="Frequency", sub=paste("Skewness:", round(
polygon(density(cars$dist), col="red")
```



Correlation

Correlation is a statistical measure that suggests the level of linear dependence between two variables, that occur in pair - just like what we have here in speed and dist. Correlation can take values between -1 to +1. If we observe for every instance where speed increases, the distance also increases along with it, then there is a high positive correlation between them and therefore the correlation between them will be closer to 1. The opposite is true for an inverse relationship, in which case, the correlation between the variables will be close to -1.

A value closer to 0 suggests a weak relationship between the variables. A low correlation ($-0.2 < x < 0.2$) probably suggests that much of variation of the response variable (Y) is unexplained by the predictor (X), in which case, we should probably look for better explanatory variables.

```
cor(cars$speed, cars$dist)  # calculate correlation between speed and distance
```

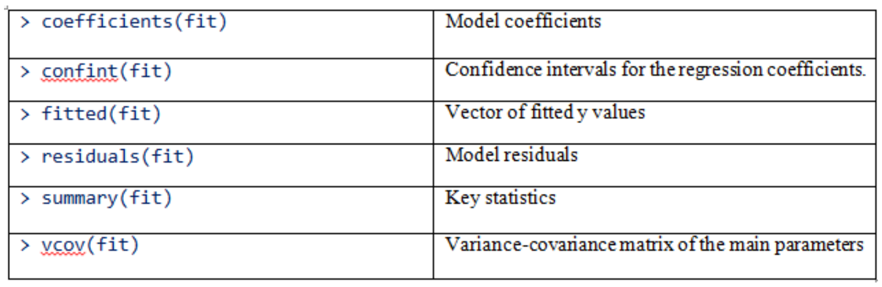
```
## [1] 0.8068949
```

Build a Model

Now that we have seen the linear relationship pictorially in the scatter plot and by computing the correlation, lets see the syntax for building the linear model. The function used for building linear models is lm(). The lm() function takes in two main arguments, namely: 1. Formula 2. Data. The data is typically a data.frame and the formula is a object of class formula. But the most common convention is to write out the formula directly in place of the argument as written below.

```
fit<-lm(dist~speed,data=cars)
```

Please try these following codes.



`> coefficients(fit)`	Model coefficients
`> confint(fit)`	Confidence intervals for the regression coefficients.
`> fitted(fit)`	Vector of fitted y values
`> residuals(fit)`	Model residuals
`> summary(fit)`	Key statistics
`> vcov(fit)`	Variance-covariance matrix of the main parameters

```
print(fit)
```

```
##
## Call:
```

```
## lm(formula = dist ~ speed, data = cars)
##
## Coefficients:
## (Intercept)        speed
##     -17.579        3.932
```

Therefore, the model is

dist= -17.579+3.932* **speed**

The average distance value is -17.579 if the range of speed covers 0. If not, it does not have a physical interpretation. On the other hand, the one unit increment in speed results in 3.932 unit increment in distance. It has a positive effect on the response variable which is distance.

```r
summary(fit)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

The p-value of the model is 1.49e-12 which is less than 0.05. Therefore, it is said that the model is significant. Also, adjusted R-squared value is 0.64. It means 64% variability in distane -response variable- is explained by speed. Besides, It is seen that both intercept and the effect of speed on the distance are significant due to p values.

The p Value: Checking for statistical significance

The summary statistics above tells us a number of things. One of them is the model p-Value (bottom last line) and the p-Value of individual predictor variables (extreme right column under 'Coefficients'). The p-Values are very important because, We can consider a linear model to be statistically significant only when both these p-Values are less that the pre-determined statistical significance level, which is ideally 0.05. This is visually interpreted by the significance stars at the end of the row. The more the stars beside the variable's p-Value, the more significant the variable.

Null and alternate hypothesis When there is a p-value, there is a hull and alternative hypothesis associated with it. In Linear Regression, the Null Hypothesis is that the coefficients associated with the variables is equal to zero. The alternate hypothesis is that the coefficients are not equal to zero (i.e. there exists a relationship between the independent variable in question and the dependent variable).

t-value We can interpret the t-value something like this. A larger t-value indicates that it is less likely that the coefficient is not equal to zero purely by chance. So, higher the t-value, the better.

Pr(>|t|) or p-value is the probability that you get a t-value as high or higher than the observed value when the Null Hypothesis (the ?? coefficient is equal to zero or that there is no relationship) is true. So if the

Pr(>|t|) is low, the coefficients are significant (significantly different from zero). If the Pr(>|t|) is high, the coefficients are not significant.

What this means to us? when p Value is less than significance level (< 0.05), we can safely reject the null hypothesis that the co-efficient ?? of the predictor is zero. In our case, linearMod, both these p-Values are well below the 0.05 threshold, so we can conclude our model is indeed statistically significant.

It is absolutely important for the model to be statistically significant before we can go ahead and use it to predict (or estimate) the dependent variable, otherwise, the confidence in predicted values from that model reduces and may be construed as an event of chance.

How to calculate the t Statistic and p-Values?

When the model co-efficients and standard error are known, the formula for calculating t Statistic and p-Value is as follows:

$t = (?? - coefficient) / std.error$

```
modelSummary <- summary(fit)  # capture model summary as an object
modelCoeffs <- modelSummary$coefficients  # model coefficients
beta.estimate <- modelCoeffs["speed", "Estimate"]  # get beta estimate for speed
std.error <- modelCoeffs["speed", "Std. Error"]  # get std.error for speed
t_value <- beta.estimate/std.error  # calc t statistic
p_value <- 2*pt(-abs(t_value), df=nrow(cars)-ncol(cars))  # calc p Value
f_statistic <- fit$fstatistic[1]  # fstatistic
f <- summary(fit)$fstatistic  # parameters for model p-value calc
model_p <- pf(f[1], f[2], f[3], lower=FALSE)
```

```
values=list(t_value,p_value,model_p)
values
```

```
## [[1]]
## [1] 9.46399
##
## [[2]]
## [1] 1.489836e-12
##
## [[3]]
##        value
## 1.489836e-12
```

R-Squared and Adj R-Squared

The actual information in a data is the total variation it contains. What R-Squared tells us is the proportion of variation in the dependent (response) variable that has been explained by this model.

$$R^2 = 1 - \frac{SSE}{SST}$$

where, SSE is the sum of squared errors given by $SSE = \sum_i^n (y_i - \hat{y_i})^2$ and $SST = \sum_i^n (y_i - \bar{y_i})^2$ is the sum of squared total. Here, $\hat{y_i}$ is the fitted value for observation i and $\bar{y}$ is the mean of Y.

We don't necessarily discard a model based on a low R-Squared value. Its a better practice to look at the AIC and prediction accuracy on validation sample when deciding on the efficacy of a model.

What about adjusted R-Squared?

As you add more X variables to your model, the R-Squared value of the new bigger model will always be greater than that of the smaller subset. This is because, since all the variables in the original model is also present, their contribution to explain the dependent variable will be present in the super-set as well, therefore, whatever new variable we add can only add (if not significantly) to the variation that was already explained. It is here, the adjusted R-Squared value comes to help. Adj R-Squared penalizes total value for the number of terms (read predictors) in your model. Therefore when comparing nested models, it is a good practice to look at adj-R-squared value over R-squared.

$$R^2_{adj} = 1 - \frac{MSE}{MST}$$

where, MSE is the mean squared error given by $MSE = \frac{SSE}{(n-q)}$ and $MST = \frac{SST}{(n-1)}$ squared total, where n is the number of observations and q is the number of coefficients in the model.

Therefore, by moving around the numerators and denominators, the relationship between R^2 and R^2_{adj} becomes:

$$R^2_{adj} = 1 - \left(\frac{\left(1 - R^2\right)(n-1)}{n-q} \right)$$

*** Standard Error and F-Statistic***

Both standard errors and F-statistic are measures of goodness of fit.

$$Std.Error = \sqrt{MSE} = \sqrt{\frac{SSE}{n-q}}$$

$$F - statistic = \frac{MSR}{MSE}$$

where, n is the number of observations, q is the number of coefficients and MSR is the mean square regression, calculated as,

$$MSR = \frac{\sum_i^n \left(y_i \hat{-} \bar{y}\right)}{q-1} = \frac{SST - SSE}{q-1}$$

AIC and BIC

The Akaike's information criterion - AIC (Akaike, 1974) and the Bayesian information criterion - BIC (Schwarz, 1978) are measures of the goodness of fit of an estimated statistical model and can also be used for model selection. Both criteria depend on the maximized value of the likelihood function L for the estimated model.

The AIC is defined as:

AIC=(???2)×ln(L)+(2×k)

where, k is the number of model parameters and the BIC is defined as:

BIC=(???2)×ln(L)+kxln(n)

where, n is the sample size.

For model comparison, the model with the lowest AIC and BIC score is preferred.

```r
AIC(fit)  # AIC => 419.1569
```

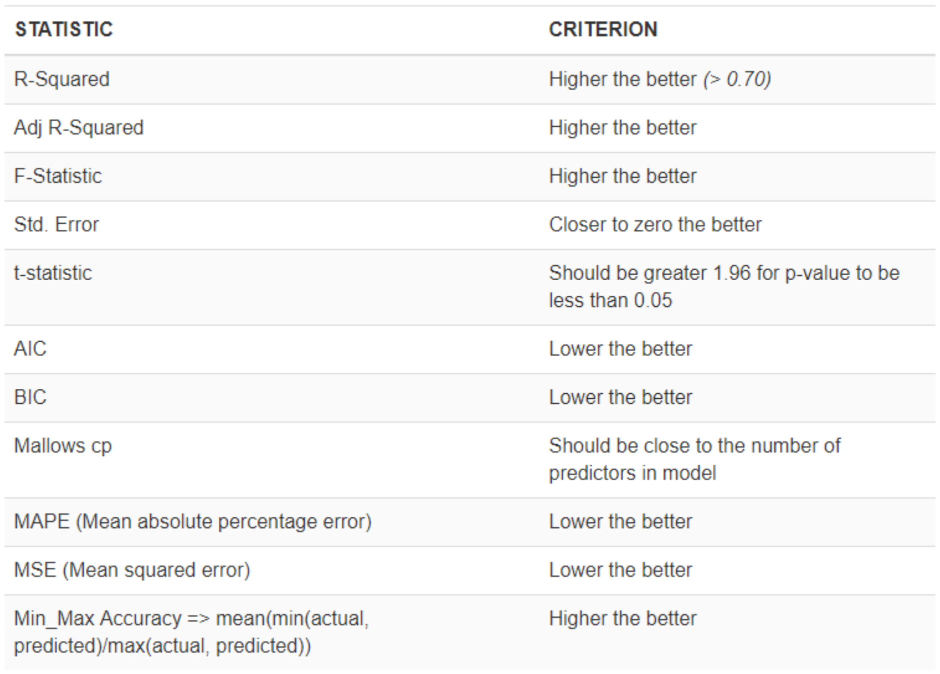
```
## [1] 419.1569
```

```r
BIC(fit)  # BIC => 424.8929
```

```
## [1] 424.8929
```

How to know if the model is best fit for your data?

The most common metrics to look at while selecting the model are:



STATISTIC	CRITERION
R-Squared	Higher the better *(> 0.70)*
Adj R-Squared	Higher the better
F-Statistic	Higher the better
Std. Error	Closer to zero the better
t-statistic	Should be greater 1.96 for p-value to be less than 0.05
AIC	Lower the better
BIC	Lower the better
Mallows cp	Should be close to the number of predictors in model
MAPE (Mean absolute percentage error)	Lower the better
MSE (Mean squared error)	Lower the better
Min_Max Accuracy => mean(min(actual, predicted)/max(actual, predicted))	Higher the better

Predicting Linear Models

So far we have seen how to build a linear regression model using the whole dataset. If we build it that way, there is no way to tell how the model will perform with new data. So the preferred practice is to split your dataset into a 80:20 sample (training:test), then, build the model on the 80% sample and then use the model thus built to predict the dependent variable on test data.

Doing it this way, we will have the model predicted values for the 20% data (test) as well as the actuals (from the original dataset). By calculating accuracy measures (like min_max accuracy) and error rates (MAPE or MSE), we can find out the prediction accuracy of the model. Now, lets see how to actually do this.

Step 1: Create the training (development) and test (validation) data samples from original data.

```r
# Create Training and Test data -
set.seed(100)  # setting seed to reproduce results of random sampling
training <- sample(1:nrow(cars), 0.8*nrow(cars))  # row indices for training data
train<- cars[training, ]  # model training data
test<- cars[-training, ]   # test data
```

Step 2: Develop the model on the training data and use it to predict the distance on test data

```r
# Build the model on training data -
model <- lm(dist ~ speed, data=train)  # build the model
distPred <- predict(model, test)  # predict distance
```

Step 3: Review diagnostic measures.

```r
summary(model) # model summary
```

```
##
## Call:
## lm(formula = dist ~ speed, data = train)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.350 -10.771  -2.137   9.255  42.231
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -22.657      7.999  -2.833  0.00735 **
## speed          4.316      0.487   8.863 8.73e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.84 on 38 degrees of freedom
## Multiple R-squared:  0.674,  Adjusted R-squared:  0.6654
## F-statistic: 78.56 on 1 and 38 DF,  p-value: 8.734e-11
```

```r
AIC (model)  # Calculate akaike information criterion
```

```
## [1] 338.4489
```

From the model summary, the model p value and predictor's p value are less than the significance level, so we know we have a statistically significant model. Also, the R-Sq and Adj R-Sq are comparative to the original model built on full data.

Step 4: Calculate prediction accuracy and error rates

A simple correlation between the actuals and predicted values can be used as a form of accuracy measure. A higher correlation accuracy implies that the actuals and predicted values have similar directional movement, i.e. when the actuals values increase the predicteds also increase and vice-versa.

```r
actuals_preds <- data.frame(cbind(actuals=test$dist, predicteds=distPred))  # make actuals_predicteds d
correlation_accuracy <- cor(actuals_preds)  # 82.7%
head(actuals_preds)
```

```
##    actuals predicteds
## 1        2  -5.392776
## 4       22   7.555787
## 8       26  20.504349
## 20      26  37.769100
## 26      54  42.085287
## 31      50  50.717663
```

Now lets calculate the Min Max accuracy and MAPE:

$$MinMaxAccuracy = mean\left(\frac{min\left(actuals, predicteds\right)}{max\left(actuals, predicteds\right)}\right)$$

$$MeanAbsolutePercentageError\ (MAPE) = mean\left(\frac{abs\left(predicteds???actuals\right)}{actuals}\right)$$

```r
min_max_accuracy <- mean(apply(actuals_preds, 1, min) / apply(actuals_preds, 1, max))
# => 58.42%, min_max accuracy
mape <- mean(abs((actuals_preds$predicteds - actuals_preds$actuals))/actuals_preds$actuals)
# => 48.38%, mean absolute percentage deviation
```

k- Fold Cross validation

Suppose, the model predicts satisfactorily on the 20% split (test data), is that enough to believe that your model will perform equally well all the time? It is important to rigorously test the model's performance as much as possible. One way is to ensure that the model equation you have will perform well, when it is ‘built’ on a different subset of training data and predicted on the remaining data.

How to do this is? Split your data into ‘k’ mutually exclusive random sample portions. Keeping each portion as test data, we build the model on the remaining (k-1 portion) data and calculate the mean squared error of the predictions. This is done for each of the ‘k’ random sample portions. Then finally, the average of these mean squared errors (for ‘k’ portions) is computed. We can use this metric to compare different linear models.

By doing this, we need to check two things:

If the model’s prediction accuracy isn’t varying too much for any one particular sample, and If the lines of best fit don’t vary too much with respect the the slope and level. In other words, they should be parallel and as close to each other as possible. You can find a more detailed explanation for interpreting the cross validation charts when you learn about advanced linear model building.