Introduction to R programming

Boby Mathew,
University of Bonn

Introduction

R is a scripting language mainly focused on statistical analysis. R is a modern implementation of S (S for statistics) programming language. However R become more popular because R is open source and more people are contributing to its development. R can be downloaded from the link http://www.r-project.org/. CRAN provides R in recompiled binaries files. When you start R you will get R prompt $'>'$ Here you can execute R commands. In my note I used symbol $'>'$ to represent the R command prompt and $'\#'$ is used for the explanation of those commands.
R can be used as a normal calculator for various arithmetic operations like $+, -, *, /$.
For a simple calculation

```
> 1+1
```

```
[1] 2
```

Here we did not assign the value of the expression to some variable, so R print the value of the expression. Here, our result [1] 2 means that the result is a vector with one component and the value is 2.
Let us assign the value of the expression to a variable.

```
> x=1+1
```

If you want to see the result then type 'x'.

```
> x
```

```
[1] 2
```

Changing language in R

Locate the Rconsole file in the "etc" folder under R program files folder. Find the line "language = ", change it to "language = en" if you want to run R in English. But you need to have the administrator privilege to change the language.

Increase the memory for R

Sometimes when you run a program with big dataset you will get an error "out of memory". Then you can increase the memory manually. Close R window, then right-click on your R icon then select Properties", and then goto the Shortcut" tab. Search for the "Target" field and after the closing quotes around the location of the R executable, add '–max-mem-size=7500M'. Then restart R. But you need to have the administrator privilege. Then you can check the memory using function *memory.limit*().

Functions in R

In programming languages function is a named piece of code that can complete a specific task. There are two types of functions user defined and built in functions. Built in functions are functions that are already included in R Example mean(), var() User defined functions are functions written by users to perform specific task. General syntax of the user defined function is: function_name(argument...)

R data types

In programming language a data type is a classification to identify one of various types of data. R supports a wide variety of data types including scalars, vectors (numerical, character, logical), matrices, data frames, and lists.

Example
$> x = c(4, 5, 6, 7)$ # create a numeric vector
$> y = c("one","two","three")$ # create a character vector
$> z = c(TRUE, TRUE, TRUE, FALSE)$ # create logical vector

Here 'x,y,z' are called variables. the variable name can start with one of the letters A-Z or a-z. However variable name cannot start with a number. Also variable name cannot contain white space. Here 'c' is a function that combines values into a vector. The elements of the vector can be accessed using '[ ]'. Example
$> x[1]$
output
[1]4
$length(x)$ # print the length of the vector
Arithmetic operations on vectors are performed by element by element wise. Function $seq()$ is used to generate

Indexing of vector	
$x[n]$	nth element
$x[-n]$	all elements except the nth element
$x[a:b]$	all elements from a^{th} position to b^{th} position
$x[-(a:b)]$	all elements from a^{th} position to b^{th} position except a and b
$x[x > a]$	all elements greater than a

Important functions for vector	
$prod(x)$	Gives the product of all elements in x
$sum(x)$	Gives the sum of all elements in x
$sort(x)$	Elements sorted into ascending order
$mean(x)$	Gives the sample mean
$var(x)$	Gives the sample variance
$sd(x)$	Gives the sample standard deviation

regular sequences.
Example
$seq(1:5)$
[1]1 2 3 4 5
Creating a vector of zeros and ones
$> zero = numeric(5)$
[1]0 0 0 0 0

$> zero[] = 1$ # assign 1 to all elements in the vector zero
[1]1 1 1 1 1
Function $rep()$ is used to create repeating values
Example
$> rep(2, 5)$
[1]2 2 2 2 2

"To goto the previous command use 'up arrow' "

Warnings and errors

When R find a command that R cannot interpret, it will respond with an error.
$> suareroot(9)$
Error: could not find function "suareroot"
$> sqrt9$
Error: unexpected numeric constant in "sqrt 9"

R workspace

R workspace is your current working environment and it includes all user-defined objects like vector, data frame, list and functions. User can save the workspace as '.RData' and can be reloaded when R is started next time.

work space functions	
$getwd()$	print the current working directory
$ls()$	list all the objects
$dir()$	list files in a directory
$setwd(yourdirectory)$	change to your directory(mainly for Linux)
$history()$	display last 25 commands
$savehistory(file = "myfile")$	Save the history into the disk
$save(objectlist, file = "myfile.RData")$	Save the workspace object as binary file
$load("myfile.RData")$	Load the saved objects into the current session

Getting help in R

When R is installed it offers a built in help system.

How to get help.	
$help.start()$	For general help
$RSiteSearch('mean')$	Search the keyword 'mean' in CRAN web site
$help('mean')$	Will provide help for the function 'mean'
$?mean$	Same as help('mean')
$apropos('mean')$	List all the functions contains the string 'mean'
$example('mean')$	Will give an example for the function 'mean'
$data()$	List all datasets that you can experiment with

Matrices in R

Function $matrix(data, nrow, ncol, byrow)$ is used to generate matrices in R. Here *data* is an object that has to be converted into matrix, *nrow* is the number of rows *ncol* is the number of columns and *byrow* to arrange the elements in row order. Example
$> sq = seq(1:6)$
$> mat = matrix(sq, 2)$
Function $dim()$ will provide the dimensionality of the matrix.
$> dim(mat)$
[1]2 3 (2 rows and 3 columns)
Function $t()$ is used to get the transpose of a matrix
Example
$> t(mat)$ # will provide the transpose of a matrix

In order to create a zero matrix of order 3×3: $matrix(0, 3, 3)$
To create 3×3 matrix with all elements equal to one : $matrix(1, 3, 3)$
To create an identity matrix of order 3×3: $diag(3)$

Indexing to the elements in the matrix:
$> matrix[1, 3]$ is 5. Will pick the element at the 2^{nd} row and 3^{rd} column.

Referring to the rows and columns. $> mat[2,]$ # will return the second row of the matrix.
$> mat[, 2]$ # will return the second column of the matrix.
$> mat[, -2]$ # will return a matrix after removing the second column
You can generate matrices combining vectors using function like rbind and cbind. Function *rbind* combine by rows whereas *cbind* combine by columns. Example
$> a = 1:5$
$> b = 5:9$
$> c = rbind(a, b)$

Applying a function to all rows or columns of a matrix
$> apply(mat, 1, sum)$ # will return the sum of each row
$> apply(mat, 2, sum)$ # will return the sum of each column

The third argument can be any function with vectors as arguments. Examples: $prod, mean, var$

List in R

A list is an ordered collection of objects. Usually they are name value pairs and the components can be accessed by name.
Example
$> w = list(name = "INRES", matrix = mat, age = 10)$; # is a list containing string, matrix and integer.

Each element of the List can be accessed using the name or numbers.
e.g $> w\$name$ or $> w[1]$;.
Function $str()$ display all the data types used in the list.*e.g* $> str(w)$;

Data frame in R

A data frame is used to store data and it is a list of vectors of equal length.

$> line = c(1, 2, 3)$
$> year = c(2001, 2002, 2003)$
$> obs = c(45, 55, 36)$
$> dat = data.frame(line, year, obs)$

To get more details about the data frame you can use functions like $summary(), str(), dim()$.

A data frame can be indexed like a matrix
Example
$> dat[1,]$
$> dat[, 1]$

$names(dat)$ will display all the column names and the data can be accessed using the column name: $> dat\$year$
You can use $attach()$ function to refer to the variables in the data frame. The $attach()$ function will add the data frame into the R search path.
Example:
$> attach(dat)$
Now the $> year$ is same as $> dat\$year$

$> detach()$ # function will remove the data frame from the R search path.
$> merge()$ # function can be used to merge two data frames into a single frame.

Reading and writing data in R

R offers various functions to read different formats of data into the R work space and the external files should be saved in a specific data format. $read.table()$ is the one of the commonly used function to read external data files. By default read.table function reads numeric items as numeric variables and non-numeric variables as factors.

In order to read the table or file you need to change the R working directory to the place where you stored your data file. Or you can copy the file into your R working directory.
$getwd()$ function can be used to see the working directory.
$> getwd()$
"$C : /Users/boby/Documents$"

One option is that you can copy your file to "$C : /Users/boby/Documents$".
Suppose if your data is stored in "$C : /Users/boby/Desktop$". As a second option you can change the working directory in R to the place where you stored your data using the command $setwd()$.
$setwd($"$C : /Users/boby/Desktop$"$)$
Reading a table

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

Here the *read.table*() function upload data into R and stored in a table called *dat* and $header = TRUE$ option specifies that the first row is a line of headings.
In order to examine whether the data is read properly we can use the following functions
Function *dim*() will return the number of rows and columns.

```
> dim(dat)
```

```
[1] 100   6
```

Functions like *rownames*() and *colnames*()/*names*() will return the row names and column names of the table.

```
> names(dat)
```

```
[1] "Number"      "Genotype"    "Location"    "Height"      "Root_length"
[6] "Yield"
```

Functions like *summary*() and *str*() will provide a short summary of the table.

```
> summary(dat)
```

```
     Number      Genotype     Location      Height       Root_length
 Line1  : 1   GE_1   : 2   Bonn   :50   Min.   :34.06   Min.   : 2.34
 Line10 : 1   GE_10  : 2   Cologne:50   1st Qu.:47.12   1st Qu.:16.11
 Line100: 1   GE_11  : 2                Median :50.37   Median :19.22
 Line11 : 1   GE_12  : 2                Mean   :50.24   Mean   :19.36
 Line12 : 1   GE_13  : 2                3rd Qu.:53.45   3rd Qu.:22.83
 Line13 : 1   GE_14  : 2                Max.   :65.84   Max.   :34.31
 (Other):94   (Other):88
     Yield
 Min.   :135.6
 1st Qu.:147.3
 Median :150.6
 Mean   :150.4
 3rd Qu.:153.7
 Max.   :165.7
```

head() and *tail*() functions can be used to print the first or last few lines of a table.

```
> head(dat,n=3)
```

```
  Number Genotype Location   Height Root_length    Yield
1  Line1     GE_1     Bonn 52.27573    18.91135 149.6148
2  Line2     GE_1  Cologne 53.35362    20.92301 151.0124
3  Line3     GE_2     Bonn 45.08251    14.31079 145.3708
```

```
> tail(dat,3)
```

```
     Number Genotype Location   Height Root_length    Yield
98   Line98    GE_49  Cologne 49.04374    18.31187 149.5071
99   Line99    GE_50     Bonn 43.16764    13.84799 143.1218
100 Line100    GE_50  Cologne 51.47421    20.57106 150.9898
```

Here $n = 3$ option is used to chose the number of lines to print.
Function t() can be used to transpose a data frame.

```
> dat_transpose=t(dat)
```

Here the data table is transposed and stored into a new table called *dat_transpose*.

Accessing individual columns and rows in the table

You can use $'\$'$ sign to access individual columns in the dataset.

```
> dat$Location
```

```
  [1] Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn
 [10] Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne
 [19] Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn
 [28] Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne
 [37] Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn
 [46] Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne
 [55] Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn
 [64] Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne
 [73] Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn
 [82] Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne
 [91] Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn    Cologne Bonn
[100] Cologne
Levels: Bonn Cologne
```

If you want to access the individual elements of table you can use dat[row,col]. Here row is used to set the row and col is used to set which column. As an example let us take few element of the table.

```
> dat[1,2]
```

```
[1] GE_1
50 Levels: GE_1 GE_10 GE_11 GE_12 GE_13 GE_14 GE_15 GE_16 GE_17 GE_18 ... GE_9
```

```
> dat[1,3]
```

```
[1] Bonn
Levels: Bonn Cologne
```

```
> dat[2,4]
```

```
[1] 53.35362
```

If you want to access a specific row of a table use square bracket [row.number,]. Suppose if you want to see the second row.

```
> dat[2,]

  Number Genotype Location   Height Root_length    Yield
2  Line2     GE_1  Cologne 53.35362    20.92301 151.0124
```

Suppose if you wan to see the correlation between plant height and root length:

```
> cor(dat$Height,dat$Root_length)

[1] 0.9623674
```

The same way you can use functions like *var*(), *sd*() and *mean*().

```
> var(dat$Height)

[1] 30.9677

> mean(dat$Root_length)

[1] 19.36018
```

Sub setting the data table.

Suppose if you want to create another table where the Yield is less than 150. You can use *subset*() function.

```
> dat_150=subset(dat, dat$Yield<150)
> head(dat_150,3)

  Number Genotype Location   Height Root_length    Yield
1  Line1     GE_1     Bonn 52.27573    18.91135 149.6148
3  Line3     GE_2     Bonn 45.08251    14.31079 145.3708
6  Line6     GE_3  Cologne 47.48507    15.34137 147.4639
```

Another example if you want to create a table where our Yield is less than 150 and the root length is less than 14.

```
> dat_14=subset(dat, dat$Yield<150 &  dat$Root_length<14)
> head(dat_14,3)

   Number Genotype Location   Height Root_length    Yield
9   Line9     GE_5     Bonn 42.24373    10.96346 142.6255
10 Line10     GE_5  Cologne 45.69879    13.22416 143.3145
25 Line25    GE_13     Bonn 38.00213    11.09641 139.6112
```

If you want to get a table where Yield is less than 150 or the root length is less than 20

```
> sub=subset(dat, dat$Yield<150 | dat$Root_length<20)
> head(sub,3)

  Number Genotype Location   Height Root_length    Yield
1  Line1     GE_1     Bonn 52.27573    18.91135 149.6148
3  Line3     GE_2     Bonn 45.08251    14.31079 145.3708
4  Line4     GE_2  Cologne 52.47572    19.16425 152.5497
```

If you want to create another table for the data coming only from bonn you can use $\%in\%$ operator.

```
> bonn=dat[dat$Location %in% "Bonn",]
> head(bonn,3)
```

```
  Number Genotype Location   Height Root_length    Yield
1  Line1     GE_1     Bonn 52.27573    18.91135 149.6148
3  Line3     GE_2     Bonn 45.08251    14.31079 145.3708
5  Line5     GE_3     Bonn 51.08301    21.76464 150.1298
```

Suppose if you want to add a new column called "Diff" to the table which is the difference between the "Height" and the "Root length"

```
> dat$Diff=dat$Height-dat$Root_length
> head(dat,3)
```

```
  Number Genotype Location   Height Root_length    Yield     Diff
1  Line1     GE_1     Bonn 52.27573    18.91135 149.6148 33.36438
2  Line2     GE_1  Cologne 53.35362    20.92301 151.0124 32.43061
3  Line3     GE_2     Bonn 45.08251    14.31079 145.3708 30.77172
```

And if you what to delete a column you can use

```
> dat$Diff=NULL
> head(dat,3)
```

```
  Number Genotype Location   Height Root_length    Yield
1  Line1     GE_1     Bonn 52.27573    18.91135 149.6148
2  Line2     GE_1  Cologne 53.35362    20.92301 151.0124
3  Line3     GE_2     Bonn 45.08251    14.31079 145.3708
```

Same way if you want to delete a row.

```
> dat=dat[-1,]
> head(dat,3)
```

```
  Number Genotype Location   Height Root_length    Yield
2  Line2     GE_1  Cologne 53.35362    20.92301 151.0124
3  Line3     GE_2     Bonn 45.08251    14.31079 145.3708
4  Line4     GE_2  Cologne 52.47572    19.16425 152.5497
```

If you want to delete more than one row you have to use $c()$ function and specify the rows like $[-c(1,3...),]$.

```
> dat=dat[-c(1,3),]
> head(dat,3)
```

```
  Number Genotype Location   Height Root_length    Yield
3  Line3     GE_2     Bonn 45.08251    14.31079 145.3708
5  Line5     GE_3     Bonn 51.08301    21.76464 150.1298
6  Line6     GE_3  Cologne 47.48507    15.34137 147.4639
```

Replacing values in a data frame.

In order to replace values fist you need to choose the column. As an example let us replace all the values less than 20 in the column Root_length to 0.

```
> dat$Root_length[dat$Root_length <18]=0
> head(dat,3)
```

```
  Number Genotype Location   Height Root_length    Yield
3  Line3     GE_2     Bonn 45.08251     0.00000 145.3708
5  Line5     GE_3     Bonn 51.08301    21.76464 150.1298
6  Line6     GE_3  Cologne 47.48507     0.00000 147.4639
```

Suppose if you want to replace Location Bonn to Hamburg, Then first you need to change the column Location into character.

```
> dat$Location=as.character(dat$Location)
> dat$Location[dat$Location=="Bonn"]="Hamburg"
> head(dat,3)
```

```
  Number Genotype Location   Height Root_length    Yield
3  Line3     GE_2  Hamburg 45.08251     0.00000 145.3708
5  Line5     GE_3  Hamburg 51.08301    21.76464 150.1298
6  Line6     GE_3  Cologne 47.48507     0.00000 147.4639
```

edit() function can be used to edit a data frame:
$> new\_file = edit(dat)$
edit() function brings up a separate spreadsheet-like environment for editing. However *edit*() function will not change the original file, but it will make a copy of the original file to save the changes.

Writing data in R

write.table() function can be used to write a data frame into a file.
Example

```
> write.table(dat,'test_one.txt')
```

Here the function *write.table*() will write the table *dat* into a file called *test.txt* and will be stored in the hard disk. And the file names should be specified inside quotes. If you want to append the data to an existing file use $'append = T'$ option.

```
> write.table(dat,'test_one.txt',append=T)
```

```
> write.table(dat,"test_one.txt",row.names=F,quote=F,sep="\t")
```

In order to remove the row names use $'row.names = F'$ option. Option $'quote = F'$ is used to remove the quotes and option $sep =' \backslash t'$ is used to write the data in tab-separated value file .
To read csv files coming from Excel you can use the function *read.csv*().

Reading Excel files into R

In order to read Excel files into R, first we need to install the package *gdata*.
You can install the package using *install.pacakges*() function.
$>$ *install.packages*($'gdata'$)
Then you have to call the package into R using function *library*()

```
> library(gdata)
> dat = read.xls ("course_data.xlsx",sheet = 1, header = TRUE)
> head(dat,3)
```

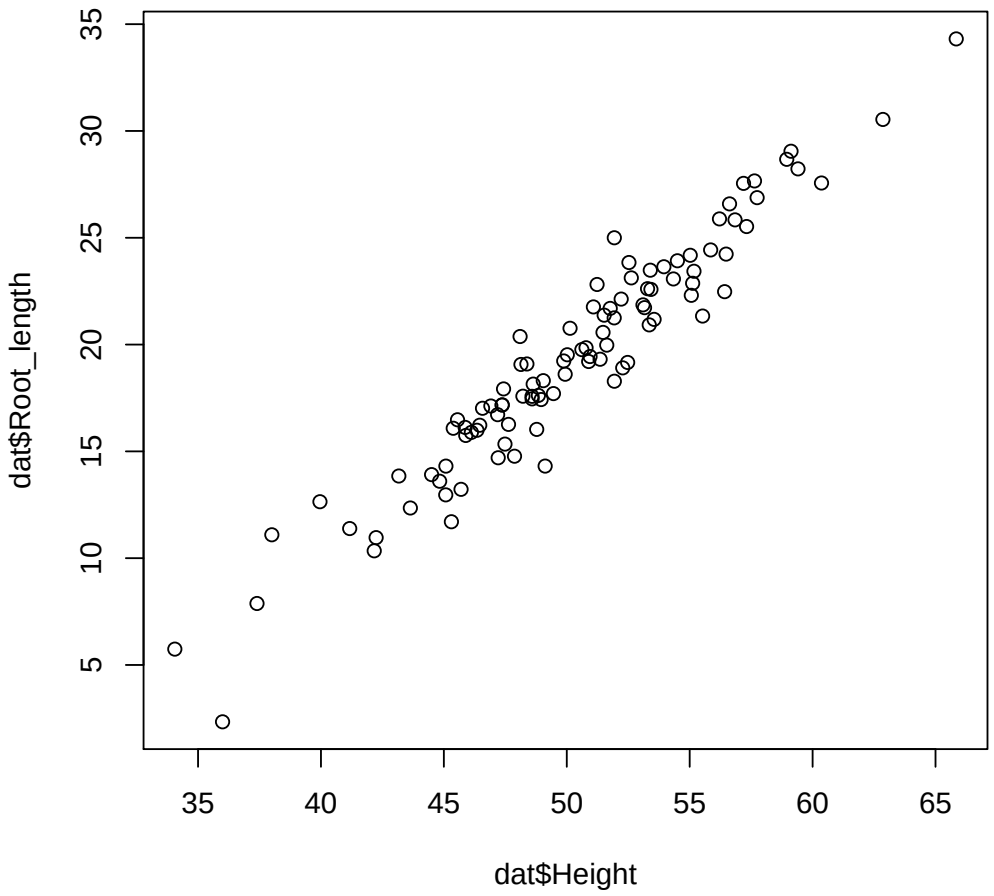
```
  Number Genotype Location   Height Root_length    Yield
1      1     GE_1     Bonn 52.27573    18.91135 149.6148
2      2     GE_1  Cologne 53.35362    20.92301 151.0124
3      3     GE_2     Bonn 45.08251    14.31079 145.3708
```

Here $sheet = 1$ option is used to specify the excel sheet, that you want to read into R.

Basic graphic functions in R

R provides various basic and more advance graphics functions.
plot() is one of the basic graphic function and *plot*() will draw the scatter plot. Scatter plot is a good way to see the relationship between two numerical variables
Let us plot a scatter plot for the plat height and root length

```
> plot(dat$Height,dat$Root_length,type="p")
```

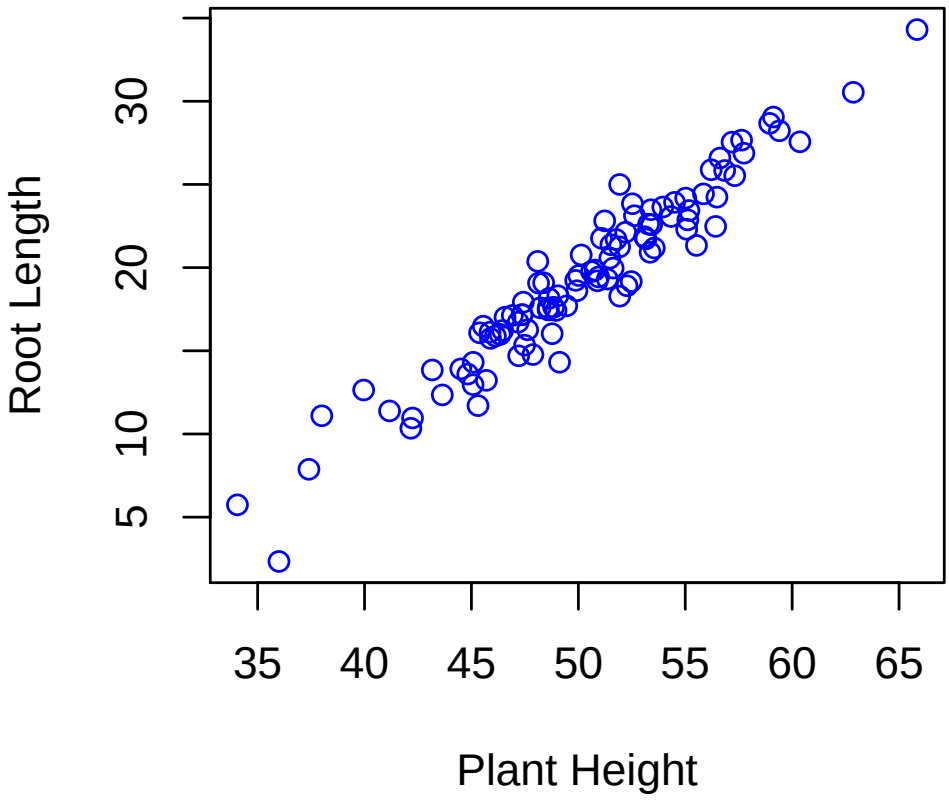


Here the *type* = "*p*" option is used to decide the type of the plot.
Different options to draw the image are.

"p" for points
"l" for lines
"b" for both
"c" for the lines part alone of"b"
"o" for both over plotted image
"h" for histogram like vertical lines

Option *main* is used to set title for the graph.
Options *xlab* and *ylab* can be used to set labels for the x and y axis respectively.
Option *col* is used to set different colors for the graph.

```
> plot(dat$Height,dat$Root_length,type="p",
+ xlab="Plant Height",ylab="Root Length",col="blue")
```



You can save the images in different formats like 'pdf','png' ,'jpeg','postscripts' and 'bmp'.
You can either use the functions or from the menu: $File- > SaveAs..$
Functions to save the image in different format.

$> pdf("example.pdf")$ # will save the image in 'pdf' file
$> win.metafile("example.wmf")$ # will save the image as 'windows metafile'
$> png("example.png")$ # will save the image as 'png' file
$> jpeg("example.jpg")$ # will save the image as 'jpeg' file
$> bmp("example.bmp")$ # will save the image as 'bmp' file
$> postscript("example.ps")$ # will save the image as 'postscript' file
It is important to call $dev.off()$ function to shuts down the graphics once the graphic is saved otherwise you cannot open the saved graphic file.
As an example let us save the plot with the name $example.pdf$. First crest the file with the file type.

```
> pdf("example.pdf")
```

Then plot the graph.

```
> plot(dat$Height,dat$Root_length,type="p",xlab="Plant Height",ylab="Root Length",col="blue")
```

Finally close the graphic device using $dev.off()$ function

```
> dev.off()

null device
          1
```

>?*devices* # will display all the graphics formats supported by R.

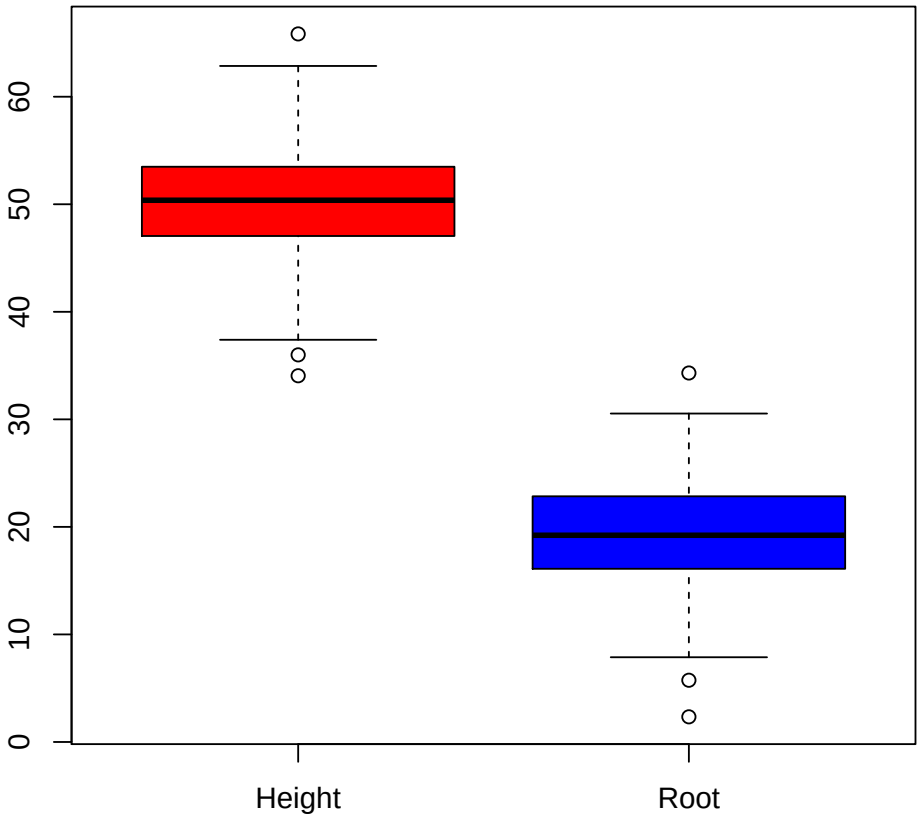
Box plots in R

Box plots are good at showing the distribution of data points around the median.

```
> boxplot(dat$Height,dat$Root_length,col=c("red","blue"))
```

Here "*col*" option is used to change the color
Example
Suppose if you want to give names for each box plot then

```
> boxplot(dat$Height,dat$Root_length,col=c("red","blue"),
+ names=c("Height","Root"))
```



Box plot three variables.

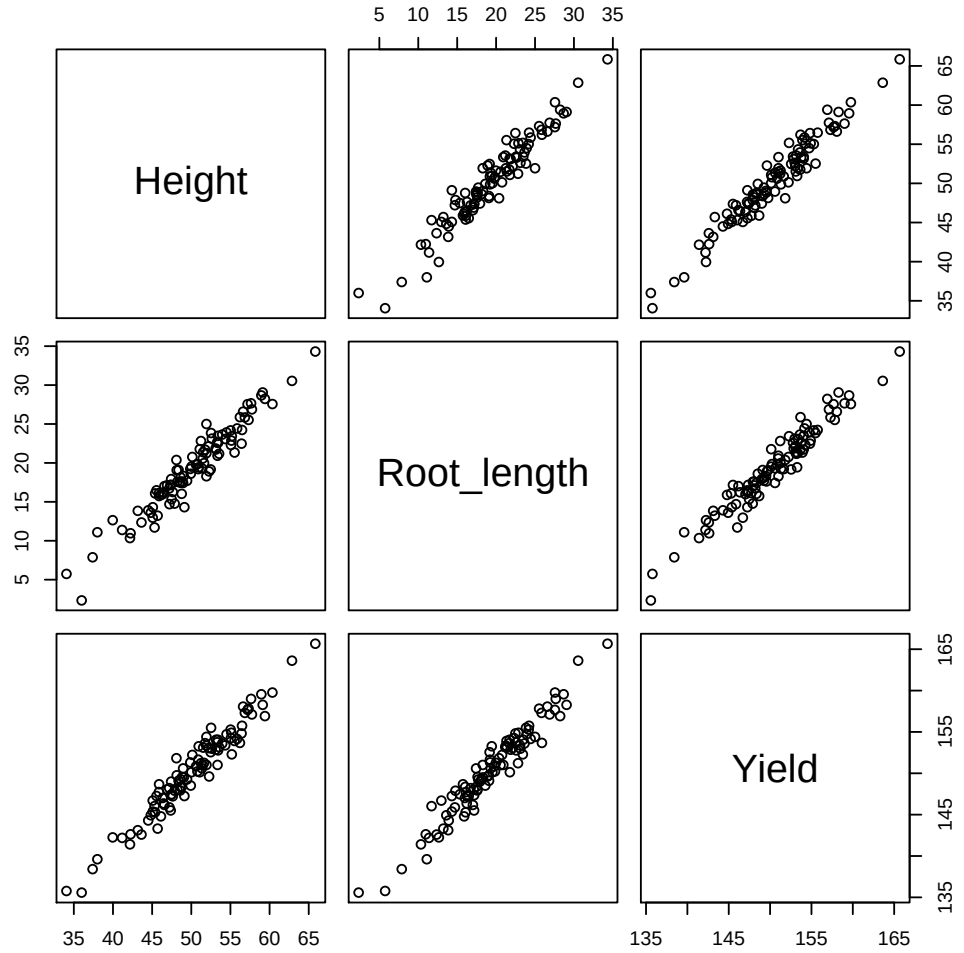
```
> boxplot(dat$Height,dat$Root_length,dat$Yield,
+ col=c("red","blue","green"),
+ names=c("Height","Root","Yield"))
```

If you wan to plot the location specific box plots for Height then

```
> boxplot(Height~Location,data=dat)
```

When you have many observation *pairs*() function is an useful way to check the relationship between those observation. Let us check the dependencies between the variables in our data. First create a new table with the observations Height, Root Length and Yield.

```
> dat_obs=dat[,4:6]
> pairs(dat_obs)
```



Bagplot

Bagplot is way to display two dimensional data. Bagplots are used to visualize the location, spread and outliers in a data set

In order to draw bag plot you need to install library $'library(aplpack)'$

```
> library(aplpack)
> bagplot(dat$Height,dat$Root_length)
```

$windows()$ can be used to open more than one graphics window at a time.
You can use $width$ and $height$ arguments to resize the window.
Example
$> windows(width = 2, height = 3)$

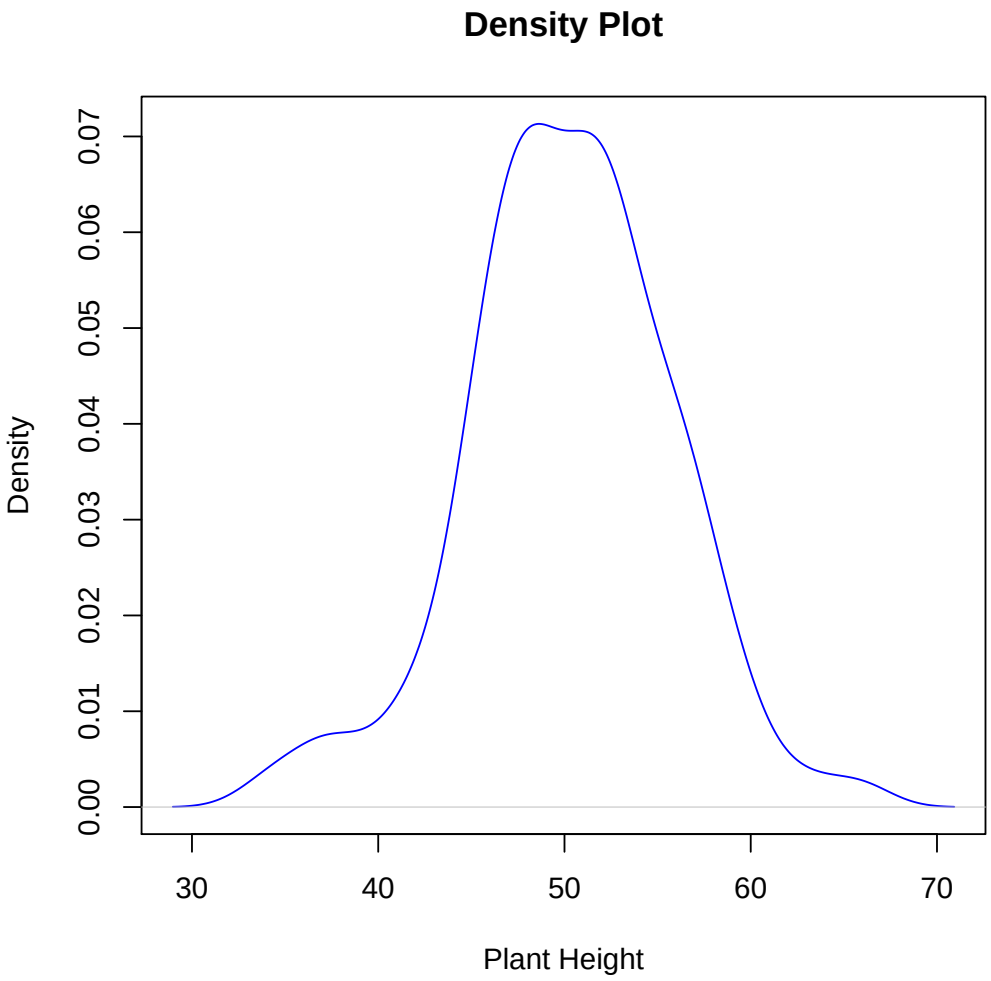
Histogram and density plots

Histograms are used to plot the distribution of numerical data.

```
> hist(dat$Height,col="red", breaks=20)
```

Here $breaks$ option decides the number of break points (no of bins) in the histogram.
However, histogram are not good in determining the shape of the distribution. Density plots are more useful to show the distribution of the data. First estimate the density of the data using $density()$ function and plot it.

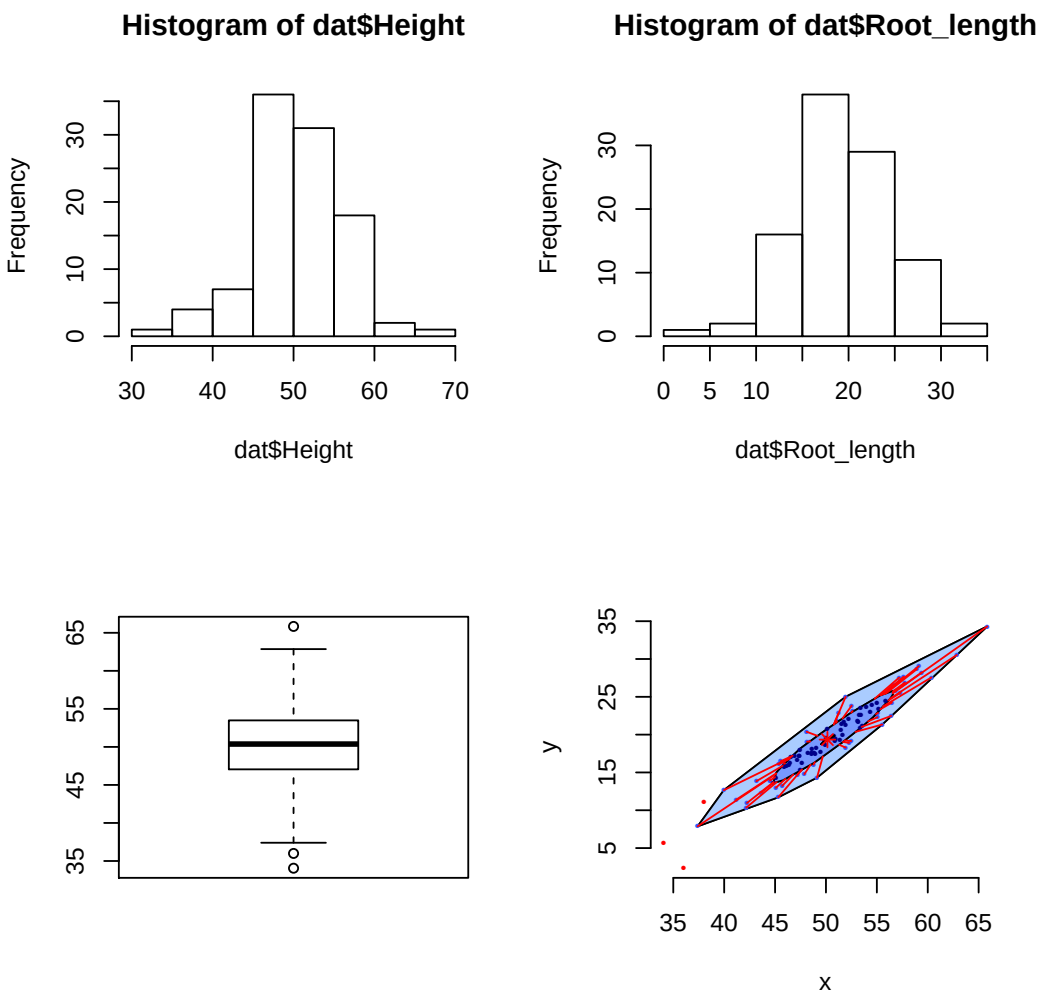
```
> den=density(dat$Height)
> plot(den,main="Density Plot",xlab="Plant Height",col="blue")
```



Combining Plots

In R it is easy to combine multiple plots into on plot. If you want to plot many panels in the same plot you can use the *par*() function.

```
> gp=par(mfrow = c(2,2)) # mfrow create 2 columns and two rows
> hist(dat$Height)
> hist(dat$Root_length)
> boxplot(dat$Height)
> bagplot(dat$Height,dat$Root_length)
```



Grammar of graphics plot (ggplot)

Grammar of graphics plot (ggplot) is based on concepts from The Grammar of Graphics by Leland Wilkinson, 2005. Nowadays *ggplot* is one of the most widely used multivariate data visualization tools in R. It provide many high level functions to visualize the multivariate data. *ggplot*2 contain 6 main components.
Data-is what you want to visualize and must be stored in R data frame.

geoms- represent how you want to draw the plot: points,lines,polygons...

aesthetics- represent the visual properties of the graph like color, shape, transparency, fill...

Scale - describe the scale for each aesthetics example like log scale .

Statistical transformations(stat)- is optional but useful to summarize the data.

Facet-describe how to break the data into subset and how to display those subsets.

The main function in *ggplot* is *qplot*(**q**uick plot). *qplot* is very similar to *plot*() function.

```
> library('ggplot2')
> dat=read.table("real_data.txt",header=T)# reading the data.
> attach(dat)
> head(dat,10)
```

```
     Line Treat     Place Year Hea Height    Yield
1  Sca001     T      Diko 2003  72  100.0 41.63330
2  Sca001     N      Diko 2003  72   95.0 37.50000
3  Sca001     T     Gudow 2003  71   84.0 43.00000
4  Sca001     N     Gudow 2003  71   90.0 34.00000
5  Sca001     T   Irlbach 2003  62   55.0 28.20513
6  Sca001     N   Irlbach 2003  63   50.0 33.05861
7  Sca001     T Morgenrot 2003  64  100.0 33.78400
8  Sca001     N Morgenrot 2003  65  108.0 27.35000
9  Sca001     T      Diko 2004  73  113.7 57.96500
10 Sca001     N      Diko 2004  74  113.3 55.59330
```

```
> str(dat)
```

```
'data.frame':        1006 obs. of  7 variables:
 $ Line  : Factor w/ 63 levels "Sca001","Sca002",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Treat : Factor w/ 2 levels "N","T": 2 1 2 1 2 1 2 1 2 1 ...
 $ Place : Factor w/ 4 levels "Diko","Gudow",..: 1 1 2 2 3 3 4 4 1 1 ...
 $ Year  : int  2003 2003 2003 2003 2003 2003 2003 2003 2004 2004 ...
 $ Hea   : int  72 72 71 71 62 63 64 65 73 74 ...
 $ Height: num  100 95 84 90 55 ...
 $ Yield : num  41.6 37.5 43 34 28.2 ...
```

Continuous variable and categorical variables

Continuous variables can take many possible values examples like measures of a person age, height, plant height or the population in a town and the continuous variables will show a continuous variation in the measurements. Whereas, the categorical variable defines a group to which the measurements belongs and normally they do not many possible values . Example like city, a person affected with disease, gender..
R will handle the categorical variables as factors and in our dataset we have year with two possible values (2003 and 2004), place with four values and treatment with two (1 and 0). So we need to define these values as factor as follows.

```
> Treat=as.factor(Treat)
> Year=as.factor(Year)
> Place=as.factor(Place)
```

Some simple plots

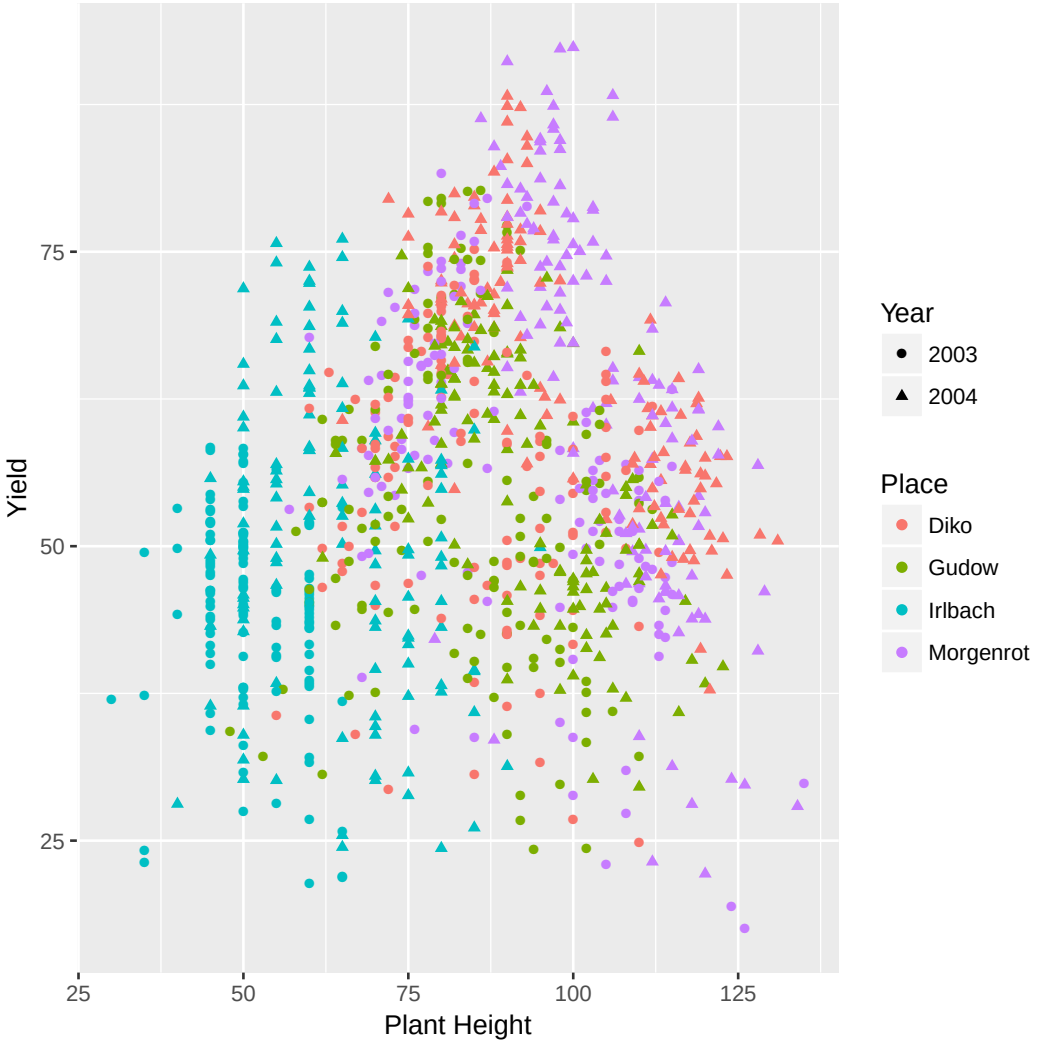
```
> qplot(Height,Yield,color=Year,xlab="Plant Height",ylab="Yield")
```

Give different shapes for different years

```
> qplot(Height,Yield,shape=Year,xlab="Plant Height",ylab="Yield")
```

Give different shapes for different years and different colors for different regions

```
> qplot(Height,Yield,shape=Year,color=Place,xlab="Plant Height",ylab="Yield")
```



If you want to draw a simple density plot and histogram

```
> qplot(Yield,data=dat,geom="density")
> qplot(Yield,data=dat,geom="histogram")
```

Here *geom* define how to display the points on the plot.
Here *colour* is an aesthetic attributes that will draw the density plot for Yield according to different places(we have four locations in our data).

Colour size and shape are main aesthetic attributes in *ggplot*2. Unlike *plot*() function *qplot*() will convert a categorical variable in your data.

```
> qplot(Yield,data=dat,geom="histogram",facets=Year~.)
```

Here the *facets* argument will split the data according to different year and will draw separate histograms for the yield in different years.

If you want to draw the histogram for different locations then:

```
> qplot(Yield,data=dat,geom="histogram",facets=Place~.)
```

If you want to draw the box plot for two different years then:

```
> qplot(Hea,Yield,data=dat,geom="boxplot",colour=factor(Year))
```

If you want to draw the scatter plot for two different years:

```
> qplot(Yield,Hea,data=dat,facets=Year~ .)+geom_smooth()
```

```
> qplot(Yield,Hea,data=dat,facets=~ Year)+geom_smooth()
```

smooth will fit a line through the middle of the data. While smoothing the plot you can use different method like *lm* or *glm*.

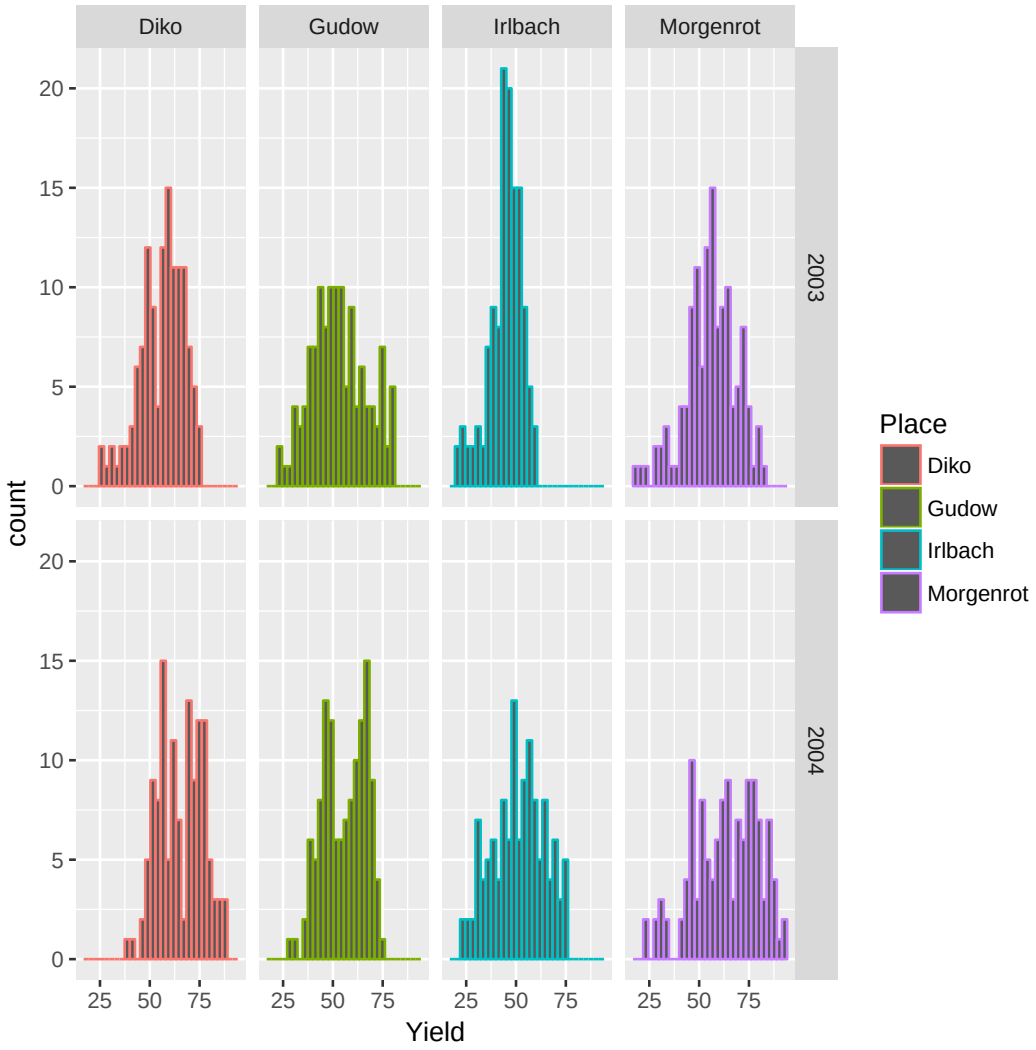
```
> qplot(Yield,Hea,data=dat,geom=c("point","smooth"))
```

Jittering is adding random noise to data in order to prevent over plotting in graphs. So if you have two equal values then *jitter* plot will add some random noise to one value and prevent over plotting.

```
> qplot(Yield,Hea,data=dat,geom="jitter")
```

If you want to down the scatter plot for heading date with the Yield for each locations and two different years in one single plot then.

```
> p=qplot(Yield,data=dat,geom="histogram",color=Place)
> p+facet_grid(Year~Place)
> p
```



Saving the image

In order to save the image first draw the picture

```
> p= qplot(Yield,Hea,data=dat)
```

Then save to the disk using function *ggsave*.

```
> ggsave("example.png",width=5,height=5)
```

You can save in different format like $jpeg, png, tif, pdf$.

```
> ggsave("example.jpeg",width=5,height=5)
```

Layers

Using *ggplot* function you can develop a plot by adding layers.
Aesthetics attributes are the properties that you can perceive on a graphic. Examples are points, size, shape and color.

```
> c=ggplot(dat, aes(y=Yield, x=Hea)) # will create a plot object\\
```

However in the above object there is no layer. In order to visualize the object we need to add a layer.
Add a layer to the object:

```
> c=c+geom_point()
> plot(c)
```

Suppose if you want to create a histogram for the trait Yield.

```
> c=ggplot(dat, aes(x=Yield)) # will create a plot object\\
> c=c+geom_histogram()
> plot(c)
```

Remember for histogram you need only one column and here we defined that as $aes(x = Yield)$. Then we add histogram as a layer using function $geom_histogram()$. In order to change the color

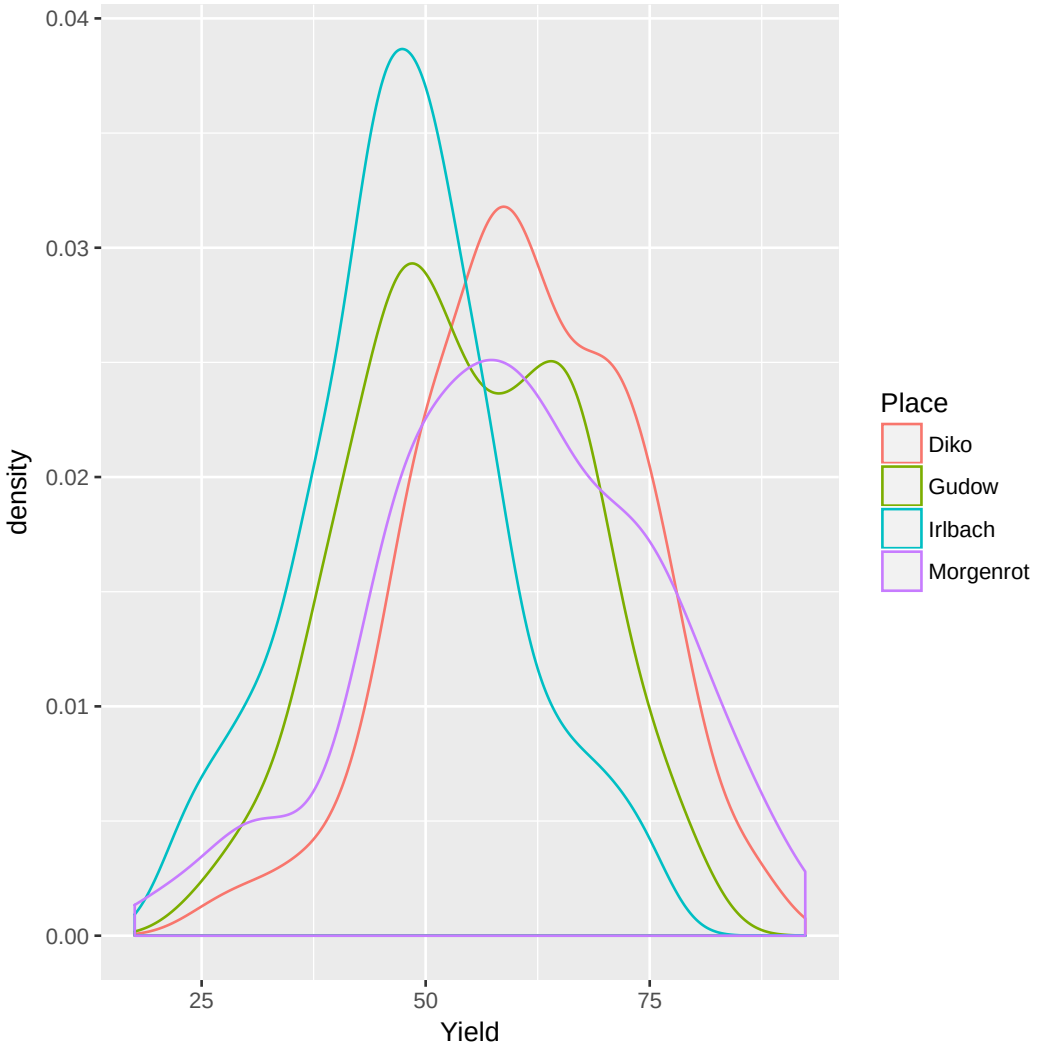
```
> c=ggplot(dat, aes(x=Yield)) # will create a plot object\\
> c=c+geom_histogram(color="red",fill="blue")
> plot(c)
```

Here red will be the color of the border and blue will the color for the bins. Bar plot for different regions

```
> ggplot(dat, aes(x=Yield, fill=Place)) + geom_histogram(position="dodge",binwidth=2)
```

Density plot for different regions

```
> ggplot(dat, aes(x=Yield,color=Place,group=Place)) + geom_density()
```



To change the back ground.

```
> m=ggplot(dat, aes(x=Yield, group=Place,color=Place)) +  geom_density()
> m+theme(panel.background = element_rect(fill = 'white', colour = 'red'))
```

Adding title you can use the *ggtitle*() function

```
> m+ggtitle("Density plot")
```

Changing the x and y axes names.

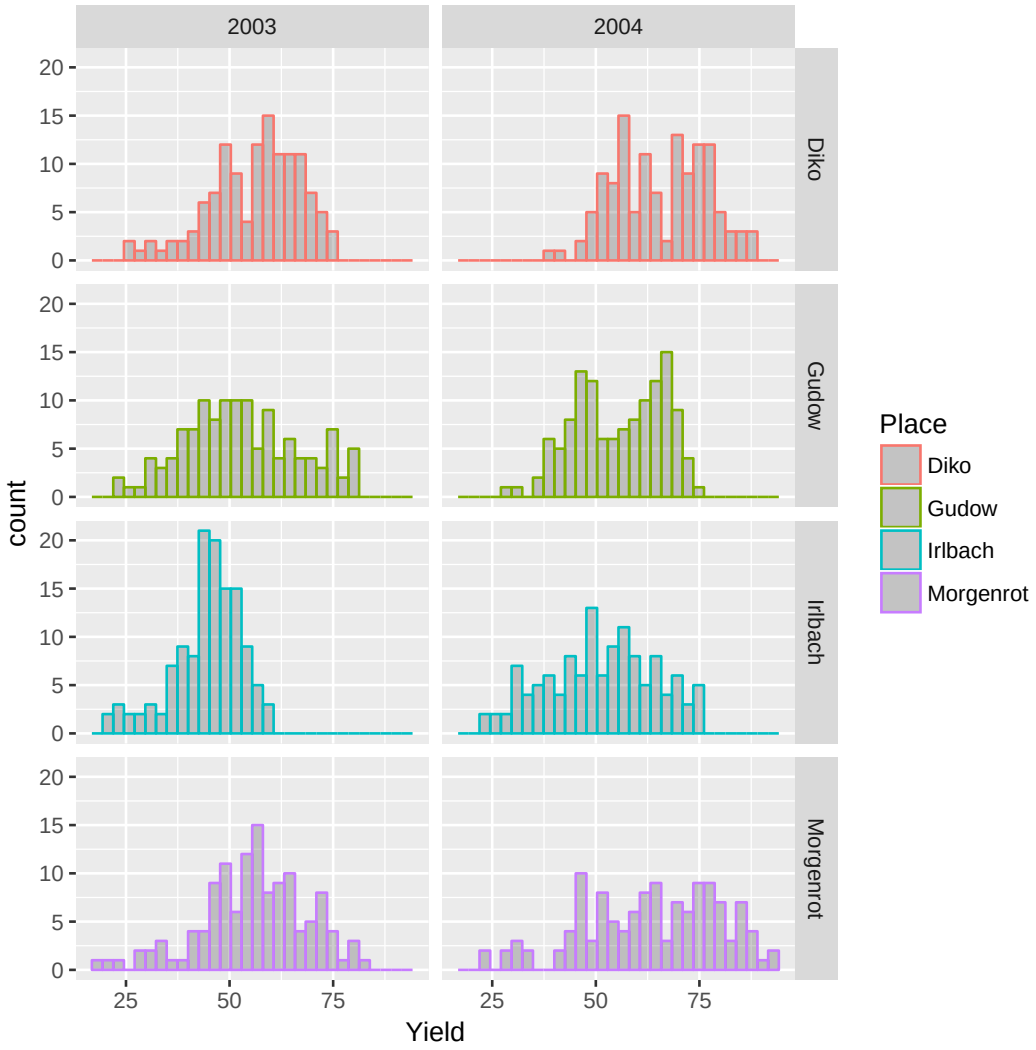
```
> m+xlab("Yield")+ylab("Count")
```

Density plot for different location in a single panel

```
> p=ggplot(dat,aes(Yield,color=Place))+geom_density()
> p+facet_grid(Place~.)
```

Histogram for different location with different years in a single panel

```
> p=ggplot(dat,aes(Yield,color=Place))+geom_histogram(alpha=0.3)
> p+facet_grid(Place~Year)
```



Customizing the legends and color.

```
> p=ggplot(dat, aes(Yield,fill = factor(Place, levels=unique(Place))))+
+ scale_fill_manual(name="Place",
+ values = c("brown3", "grey69", "forestgreen","skyblue1"))
> p+geom_histogram()+facet_grid(Place~ Year)
```

In order to draw a box plot you can use the option *geom_boxplot*()

```
> p=ggplot(dat,aes(factor(Year),Yield,
+ fill = factor(Place, levels=unique(Place))))+
+ scale_fill_manual(name="Place",
+ values = c("brown3", "grey69", "forestgreen","skyblue1"))
> p+theme(legend.position="none")+geom_boxplot()+facet_grid(Place~.)
```

Loops in R

In programming languages loops are used execute codes repeatedly. *for* and *while* loops are the commonly used loops. It is important to handle the loop properly because loops can be too slow. Loops should start with { symbol

and ed with }. Let us create a loop which prints the numbers from 1 to 5.

```
> for(i in 1:5){
+ print("hai")
+ }
```

```
[1] "hai"
[1] "hai"
[1] "hai"
[1] "hai"
[1] "hai"
```

Here i is called a counter and the value of i will be increase by 1 each time the code is execute. Hence everything inside the curly brackets {..} is done 5 times.

Consider another example to print the numbers between 1 to 5

```
> for(i in 1:5){
+ print(i)
+ }
```

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

Let us consider another example to print the members of an array using a loop.
First create an array with name 'num'.

```
> num=c("One","Two","Three","Four","five")
```

```
> for(i in 1:5){
+ print(num[i])
+ }
```

```
[1] "One"
[1] "Two"
[1] "Three"
[1] "Four"
[1] "five"
```

Functions in R

In programming languages function is a named piece of code that can complete a specific task. There are two types of functions user defined and built in functions. Built in functions are functions that are already included in R Example $mean(), var()$ User defined functions are functions written by users to perform specific task.
General syntax of the user defined function is:
$myfunction = function(arg1, arg2, ...)$

{
statements
return(*object*)
}

Like loops curly brackets { } are used to enclose the function statements.
Consider a simple function to calculate mean plant height in our example dataset.

```
> val=mean(dat$Height)
> val

[1] 84.06233
```

Here *mean*() is the function and our table column *dat*$*Height* is the argument. Parenthesis ( ) is used to group arguments and in function calls and you need the parenthesis even in function calls without no arguments.
Let us create a function to calculate the square of a number (just multiply it by itself).
First create the function with name *user_mean*.

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

Here 'x' is the argument need to be passed. Now just call our used defined function as follows.

```
> res=user_square(3)
> res

[1] 9
```

Even you can pass a vector and get the square as follows

```
> val=seq(1:5)
> val

[1] 1 2 3 4 5

> res=user_square(val)
> res

[1]  1  4  9 16 25
```

Probability distributions and simulation

R supports a large class for probability distributions. For every distribution there are four commands starting with the following letters.
> "*d*"′ # for calculating the relative likelihood of a random variable
> "*p*"′ # for evaluating the (cumulative) distribution function
> "*q*" # for evaluating the quantile function
> "*r*" # generating random values from the distribution

Normal distribution

```
> dnorm(1)

[1] 0.2419707
```

dnorm() function provide the relative likelihood of 1 with mean 0 and standard deviation 1.

```
> dnorm(1,mean=4,sd=10)

[1] 0.03813878
```

Calculate the the relative likelihood of 1 with mean 4 and standard deviation 10.
The second function is *pnrom* which calculate the probability that a normally distributed random number will be less than the given number.

```
> pnorm(0.5,mean=0,sd=1)

[1] 0.6914625
```

Returns the probability of a normally distributed random number with mean zero and sd 1 being less than 0.5.
The third function is *qnorm*() for a given probability the function returns the boundary value that determines the give probability.

```
> qnorm(0.95,mean=10,sd=5)

[1] 18.22427
```

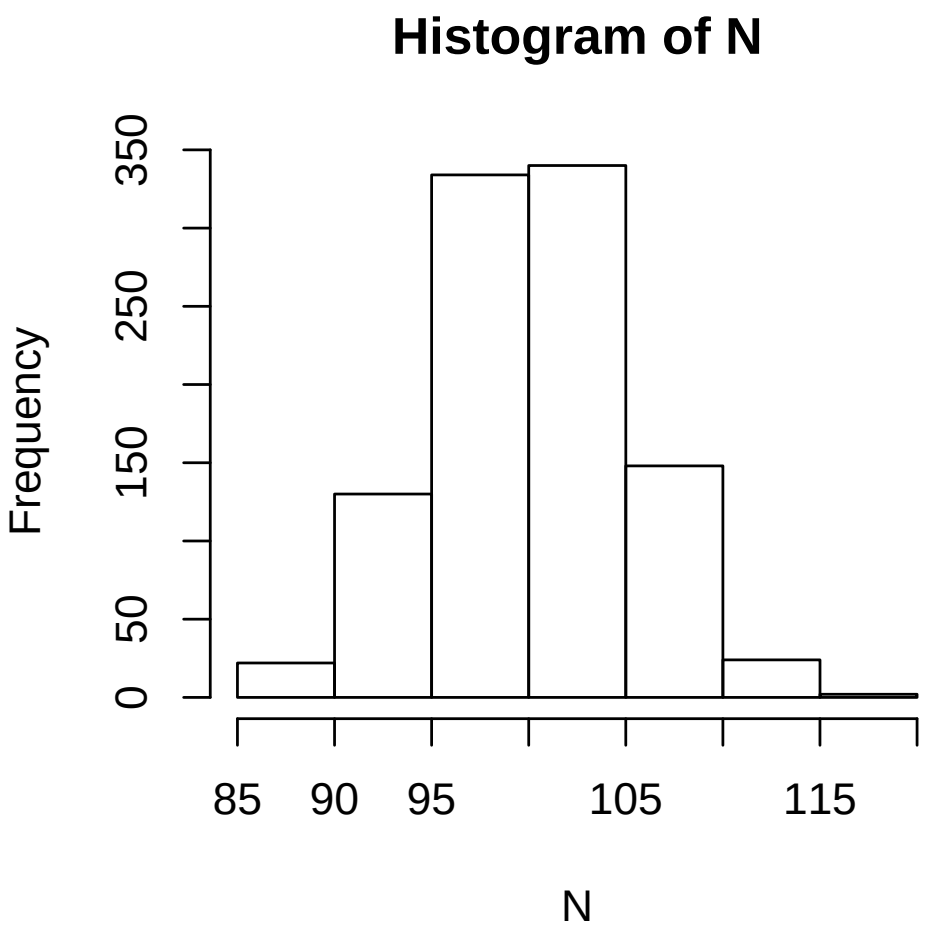
Here 95 % of the values will be under 18.22.
Last function is *rnorm*(). This function will generate random variable from normal distribution

```
> N=rnorm(1000,mean=100,sd=5)
```

Here the function will create 1000 normally distributed random variables with mean 100 and standard deviation 5.

Let us plot the figure.

```
> hist(N)
```



T test

Suppose if we draw subset of samples from a normal population, then each subset will be t-distributed. t-distributions describe samples drawn from a full population; accordingly, the t-distribution for each sample size is different. 't-test' can be used when you have independent observations from a normal distribution and both the population mean μ and the population variance σ^2 are unknown. Then you can use function $t.test()$ for calculating a confidence interval for μ, or for testing the (null) hypothesis R offers various functions for each distributions.

```
> dat=read.table("real_data.txt",header=T)
> dat_T=dat[dat$Treat %in% 'T',]
> dat_N=dat[dat$Treat %in% 'N',]
> t.test(dat_T$Yield,dat_N$Yield,paired=T)

        Paired t-test

data:  dat_T$Yield and dat_N$Yield
t = 3.0104, df = 502, p-value = 0.002741
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
```

```
 0.4999003 2.3784137
sample estimates:
mean of the differences
               1.439157
```

Here, our 'null hypothesis' is that there is no effect of treatment. But our p-values are really small so we reject the null hypothesis and conclude that there is significant difference in the 'Yield' due to the treatment.

```
> bonn = c(175, 165, 165, 190, 156, 152, 152, 175, 174)
> cologne= c(155, 169, 173, 173, 155, 156, 175, 174, 179)
> t.test(bonn,cologne)

        Welch Two Sample t-test

data:  bonn and cologne
t = -0.10481, df = 14.91, p-value = 0.9179
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.85931  10.74820
sample estimates:
mean of x mean of y
 167.1111  167.6667
```

Statistical conclusion: Hight P value hence we cannot reject the null hypothesis. Conclusion: we can conclude that the averages of two groups are significantly similar.
Note the here we did not use the option $paired = T$ because the samples are not the same.

Analysis of variance(ANOVA)

One-way ANOVA is a generalization of $t-test$ for two independent samples. One-way ANOVA allows us to compare means of several independed samples assuming that the population is normally distributed. Moreover the data is assumed to be normally distributed with common variance and may be different means.
We can formulate the test statistics like this
$H_0 : \mu_1 = \mu_2 = ...\mu_n$.
Let $\bar{x}$ be the grand mean(mean of all samples) and $\bar{x}_i$ the mean of i^{th} sample.
the the total sum of squares is:

$$SST = \sum\sum(x_{ij} - \bar{x})^2$$

This tell us total variation form the grand mean and we can split this variance into two:

$SST = SSE + SSTr$

The first term SSE is the error sum of squares, which measures the variation with in the group. The second term $SSTr$ called the treatment sum of squares, which measures the variability among the means of the samples.
Example:

```
> dat_anova=read.table("anova.txt",header=T)
> head(dat,7)
```

```
    Line Treat     Place Year Hea Height    Yield
1 Sca001     T      Diko 2003  72    100 41.63330
2 Sca001     N      Diko 2003  72     95 37.50000
3 Sca001     T     Gudow 2003  71     84 43.00000
4 Sca001     N     Gudow 2003  71     90 34.00000
5 Sca001     T   Irlbach 2003  62     55 28.20513
6 Sca001     N   Irlbach 2003  63     50 33.05861
7 Sca001     T Morgenrot 2003  64    100 33.78400
```

You can use *aov*() function also for the analysis.
One way anova
The order of the variable are important for anova. First variable is the depended variable and after the $\sim$ sign independent variable(s) are given.

```
> res=aov(Height~Treatment,data=dat_anova)
> summary(res)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
Treatment    1   2238  2237.9   14.87 0.000206 ***
Residuals   98  14753   150.5
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Two way anova

Suppose if you want to include location as independent variable

```
> res=aov(Height~Treatment+Location,data=dat_anova)
> summary(res)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
Treatment    1   2238  2237.9   18.16 4.71e-05 ***
Location     1   2797  2797.4   22.70 6.65e-06 ***
Residuals   97  11955   123.3
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Two way anova with interaction

If you want to check the interaction effect use sign $*$

```
> res=aov(Height~Treatment*Location,data=dat_anova)
> summary(res)
```

```
                   Df Sum Sq Mean Sq F value   Pr(>F)
Treatment           1   2238  2237.9   22.39 7.65e-06 ***
Location            1   2797  2797.4   27.99 7.68e-07 ***
Treatment:Location  1   2361  2360.6   23.62 4.57e-06 ***
Residuals          96   9595    99.9
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Based on th $p - value$ you can decide whether to reject or accept the null hypothesis.

F-test for the variance

In order to evaluate the sample variances of the two groups, using a Fisher's F-test, R provided *var.test*() function.

```
> dat_T=dat[dat$Treat %in% 'T',]
> dat_N=dat[dat$Treat %in% 'N',]
> var.test(dat_T$Yield,dat_N$Yield)

        F test to compare two variances

data:  dat_T$Yield and dat_N$Yield
F = 1.2244, num df = 502, denom df = 502, p-value = 0.02352
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 1.027666 1.458776
sample estimates:
ratio of variances
          1.224392
```

Here, our 'null hypothesis' is that there is no difference in variance due to treatment. But our p-values are really small so we reject the null hypothesis and conclude that there is significant variation in 'Yield' due to the treatment.

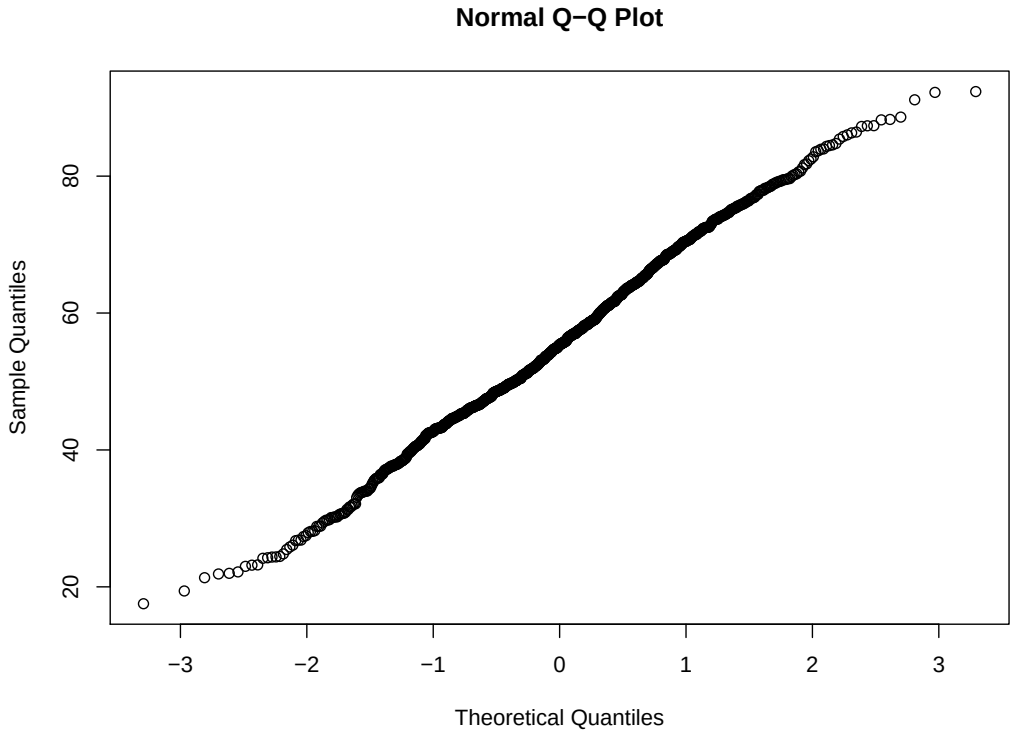
```
> var.test(bonn,cologne)

        F test to compare two variances

data:  bonn and cologne
F = 1.741, num df = 8, denom df = 8, p-value = 0.45
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.3927227 7.7184910
sample estimates:
ratio of variances
          1.741042
```

Statistical conclusion: Hight P value hence we cannot reject the null hypothesis. Conclusion: we can conclude that the variances of two groups are significantly similar (homogeneous)

Test for normality

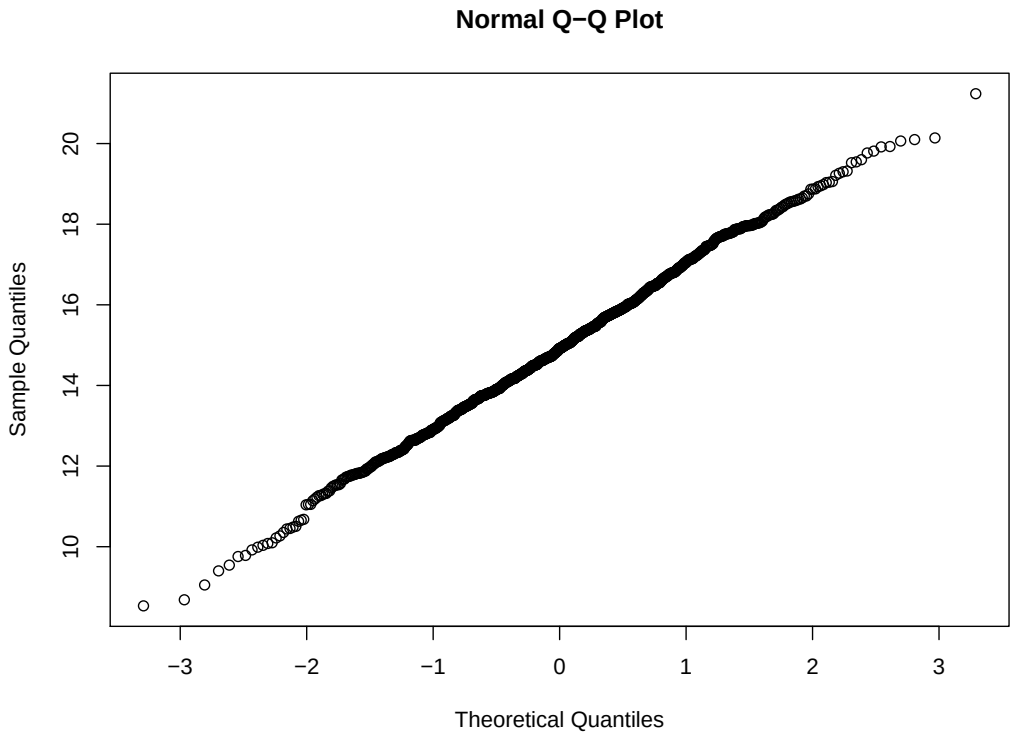
Many of the statistical analysis needs the data to be normally distributed so it is important to test for the normality of the dataset before the analysis.
The best way to address this questions is to visualize the data using some simple graphical functions like *his*() and *summary*()
In order to check the normality you can use the function *qqnorm*().

```
> qqnorm(dat$Yield)
```



If your sample is normally distributed then the line will be straight.
Let us generate few random numbers from normal distribution and check for the normality using $Q-Q$ plot

```
> x=rnorm(1000,15,2)
> qqnorm(x)
```



Shapiro test is another method to check for the normality.
Example:

```
> x=rnorm(1000,15,2)
> shapiro.test(x)

        Shapiro-Wilk normality test

data:  x
W = 0.99719, p-value = 0.07862
```

The null-hypothesis of *shapiro.test*() test is that the population is normally distributed
A p-value <= 0.05 would reject the null hypothesis that the samples come from normal distribution
Let us check for the normality of our real dataset using Shapiro test.

```
> shapiro.test(dat_T$Yield)

        Shapiro-Wilk normality test

data:  dat_T$Yield
W = 0.99147, p-value = 0.005424

> shapiro.test(dat_N$Yield)

        Shapiro-Wilk normality test

data:  dat_N$Yield
W = 0.99638, p-value = 0.3125
```

Here the results says the 'Yield' is not normally distributed for the treated plants, whereas for the non treated plants 'Yield' is normally distributed.

Linear regression

When both the response variable and the explanatory variable are continuous then you can use regression analysis. Before starting the regression analysis you can draw the scatter plot to see there is some patterns in the data. The are different kinds of regression but we are going to cover only linear regression and non-linear regression. First we need to select a model which describes the relationship between the response variable and the explanatory variable. We also make some important assumptions that the variance in y is constant, the explanatory variable ,x is measured without error and the errors are normally distributed. We can consider a simple model:

$y = a + bx$
Here a is called the intercept and b is the slop which is defined by the change in y divided by the change in x.
Example:

```
> dat=read.table("lm_example.txt",header=T)
> head(dat,5)
```

```
  Number Genotype Location Treatment   Height Root_length    Yield
1      1     GE_1     Bonn         T 22.27573    18.91135 149.6148
2      2     GE_1     Bonn         N 53.35362    20.92301 151.0124
3      3     GE_1  Cologne         T 45.08251    14.31079 145.3708
4      4     GE_1  Cologne         N 52.47572    19.16425 152.5497
5      5     GE_2     Bonn         T 21.08301    21.76464 150.1298
```

First draw the scatter plot for the two variable.

```
> plot(dat$Height,dat$Root_length)
```

Use $lm()$ function to fit the data.

```
> mod=lm(dat$Yield~ dat$Height,data=dat)
> summary(mod)
```

```
Call:
lm(formula = dat$Yield ~ dat$Height, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-13.5364  -3.1900   0.1194   2.8269  14.1848

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 144.15166    1.84039   78.33  < 2e-16 ***
```

```
dat$Height    0.13790    0.03917    3.52 0.000656 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.106 on 98 degrees of freedom
Multiple R-squared:  0.1123,        Adjusted R-squared:  0.1032
F-statistic: 12.39 on 1 and 98 DF,  p-value: 0.0006559
```

In this example our Yield can be represented as follows.

$$Yeild = (Intercept)144.15166 + 0.13790(slope) * Height + residul.$$

The summary of model can be explained as follows Residuals:
Here residual can be checked for the linearity assumption of the model. If the residual plot indicate some pattern or curve, our model may not be linear. You can draw a histogram to check whether the residuals are distributed normally.

```
> hist(resid(mod))
```

Coefficients:
First is the ”(Intercept)” which is the average yield of a plant and often the interpretation of the intercept do not make sense in the real world. Second is slope in our example slope is 0.137 and we can say that one unit increase of Height will result in 0.137 unit increase in Yield
Coefficient - Standard Error
Which measures how much the coefficient estimates can vary from the actual average value. In our example our Yield may vary 1.840. Lower values of standard Error are preferred. t value
t-value measure how many standard deviations our estimate is away from 0 and if they are far way from zero we reject the null hypothesis and we conclude that there is a relationship exist between Yield and Plant Height
Residual Standard Error
It is a measure of the quality of a linear regression fit. A value of zero means model fits the data perfectly.
F-statistic
F-statistic is a measure of whether there is a relationship between our predictor and the response variables.
Let us look at the Root Length

```
> mod=lm(dat$Yield~ dat$Root_length,data=dat)
> summary(mod)

Call:
lm(formula = dat$Yield ~ dat$Root_length, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-3.05262 -0.81147  0.02502  0.90268  3.10143

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)     131.53329    0.49369   266.4   <2e-16 ***
dat$Root_length   0.97327    0.02458    39.6   <2e-16 ***
```

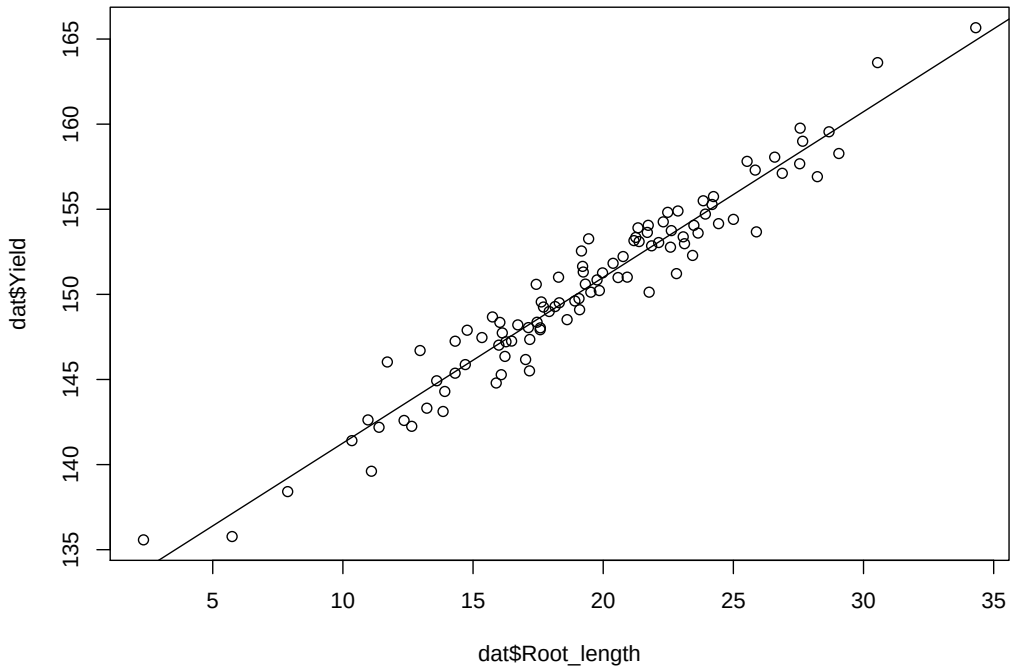
```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.314 on 98 degrees of freedom
Multiple R-squared:  0.9412,        Adjusted R-squared:  0.9406
F-statistic:  1568 on 1 and 98 DF,  p-value: < 2.2e-16
```

The slope of the depended variable Root length is 0.97327 and the corresponding $Std.Error$ is 0.02458. Hence the t value can be calculated as 0.96397/0.02589 and the likelihood of this values can be calcaulted using a t table. Note that the lowest p value returned by $lm()$ function is $2e-16$. Here the $'adjustedR-squared'$ indicates that 95% of the variation in Yield can be explained by the depended varaible Root Length. Here you can see that Root length has more influence on the Yield also you can see that the data is more fitted in to the model. In some cases people do not want to include the intercept in the model.

```
Call:
lm(formula = dat$Yield ~ dat$Root_length - 1, data = dat)

Residuals:
   Min     1Q Median     3Q    Max
-84.32 -12.51  11.00  28.80 118.53

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
dat$Root_length   7.2857     0.1754   41.55   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 35.22 on 99 degrees of freedom
Multiple R-squared:  0.9458,        Adjusted R-squared:  0.9452
F-statistic:  1726 on 1 and 99 DF,  p-value: < 2.2e-16
```

Plotting the Graph. You can use the $abline()$ function to add the regression line to the plot.

```
> plot(dat$Yield~ dat$Root_length)
> mod=lm(Yield~ Root_length,data=dat)
> abline(mod)
```



Function *fitted* will display all the fitted value using the model. The residual plot against the fitted value is a useful to diagnose our model.

```
> plot(fitted(mod),resid(mod))
```

Multiple linear regression

When y is depended on more than one variables you have to use multiple linear regression. The model for the multiple liner regression is:
$Y_i = \beta_0 + \beta_1 X_{1i} + ... + \beta_p X_{1i} + \varepsilon_i$. Example:

```
> res=lm(Yield ~ Height+Root_length,data=dat)
> summary(res)

Call:
lm(formula = Yield ~ Height + Root_length, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-3.12465 -0.80596 -0.02665  0.92563  3.02824

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 131.17244    0.58768 223.202   <2e-16 ***
Height        0.01198    0.01062   1.128    0.262
```

```
Root_length   0.96397    0.02589  37.229   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.313 on 97 degrees of freedom
Multiple R-squared:  0.9419,        Adjusted R-squared:  0.9407
F-statistic: 786.7 on 2 and 97 DF,  p-value: < 2.2e-16
```

$'+'$ is used to add an explanatory variable into the model.

$'-'$ is used remove an explanatory variable from the model

$'*'$ is used to include explanatory variables and interactions

$'/'$ used for nesting explanatory variables in the model

Let us check the effect of location on the Yield.

```
> res=lm(Yield ~ Height+Root_length+Location,data=dat)
```

Also if you want to compare different models you can add and subtract interesting terms and refit the model. R provide *update*() function for refitting. The general format is: $update(model.object, formula = . \sim . + the\ new\ term)$ Example:

```
> res=lm(Yield ~ Height,data=dat)
> res_new=update(res,formula=Yield~ Height+Root_length)
> summary(res_new)

Call:
lm(formula = Yield ~ Height + Root_length, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-3.12465 -0.80596 -0.02665  0.92563  3.02824

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 131.17244    0.58768 223.202   <2e-16 ***
Height        0.01198    0.01062   1.128    0.262
Root_length   0.96397    0.02589  37.229   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.313 on 97 degrees of freedom
Multiple R-squared:  0.9419,        Adjusted R-squared:  0.9407
F-statistic: 786.7 on 2 and 97 DF,  p-value: < 2.2e-16
```

Akaike information criterion(AIC) can be used as a criteria for comparing different models. The model with smaller AIC fits the data better. R provide $AIC()$ function to calculate the AIC between models. General form: $AIC(first\_model\_object, second\_model\_object)$

```
> AIC(res,res_new)
```

```
        df      AIC
res      3 613.8512
res_new  4 343.1420
```

General Linear models (GLM)

GLM is a generalization of liner regression models. In simple linear regression we are trying model the relationship between dependent variable(Y) and one or more explanatory variables (X). Linear regression is appropriate when the response variable has a normal distribution. GLM can be considered when the variance in not constant and error is not normally distributed.
Three important properties of GLM are:

The error structure
The linear predictor
the link function

GLM allow different error distributions like Poisson binomial or gamma.
The error structure is defined by the $family$ option in $glm()$ function.
Example
$> glm(Y \sim X, family = binomial)$
Here $glm()$ is the function for general linear models in R. Here the response variable Y is binary hence the error term follows binomial distribution.
The linear predictor is a liner combination of coefficients(which we are interested to estimate) and explanatory variables (X).
The linear predictor is expressed as:
$\eta_i = \sum_{j=1}^{p} X_{ij}\beta_j$
Link function relate the mean value of the the response variable(observations, y) to its linear predictor.
$\eta = g(\mu)$ Here g is the link function. Comparing $glm()$ and $lm()$:
Example:

```
> mod_lm =lm(Yield~ Height+Root_length, data = dat)
```

We can calculate the coefficients for the same data set using $glm()$.

```
> mod_glm =glm(Yield~ Height+Root_length, data = dat,family=gaussian)
> summary(mod_glm)

Call:
glm(formula = Yield ~ Height + Root_length, family = gaussian,
```

```
    data = dat)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.12465  -0.80596  -0.02665   0.92563   3.02824

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 131.17244    0.58768 223.202   <2e-16 ***
Height        0.01198    0.01062   1.128    0.262
Root_length   0.96397    0.02589  37.229   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 1.722888)

    Null deviance: 2878.07  on 99  degrees of freedom
Residual deviance:  167.12  on 97  degrees of freedom
AIC: 343.14

Number of Fisher Scoring iterations: 2
```

We used different methods but we got the same coefficients. $glm()$ is normally used to model binary data(0 or 1) and count data(in which the observations are non-negative integers like (1,2,3,4)
Let us consider an example taken from UCLA web site. In the dataset we three predictor variables: gre, gpa and rank.
These scores are important to get admission for higher eduction in a university.

```
> score=read.table("score.txt",header=T)
> head(score,4)

  admit gre  gpa rank
1     0 380 3.61    3
2     1 660 3.67    3
3     1 800 4.00    1
4     1 640 3.19    4

> res = glm(admit~ gre+gpa+as.factor(rank), family=binomial, data=score)
> summary(res)

Call:
glm(formula = admit ~ gre + gpa + as.factor(rank), family = binomial,
    data = score)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6268  -0.8662  -0.6388   1.1490   2.0790
```

```
Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)      -3.989979   1.139951  -3.500 0.000465 ***
gre               0.002264   0.001094   2.070 0.038465 *
gpa               0.804038   0.331819   2.423 0.015388 *
as.factor(rank)2 -0.675443   0.316490  -2.134 0.032829 *
as.factor(rank)3 -1.340204   0.345306  -3.881 0.000104 ***
as.factor(rank)4 -1.551464   0.417832  -3.713 0.000205 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 499.98  on 399  degrees of freedom
Residual deviance: 458.52  on 394  degrees of freedom
AIC: 470.52

Number of Fisher Scoring iterations: 4
```

Here is the interpretation are done log odds. Odds ratio is the probability of winning over the probability of losing. 4 to 1 odds equal to to an odds ratio of .25/.80 = .33. then th log odd ratio become log(0.33)=-1.10.
In our example we can think odd ratio as probability of getting admission over not getting it in log scale. if gre score increases by one then odds of admission (versus non-admission) increases by 0.002. if gpa increases by one, then the log odds of being admitted to graduate school increases by 0.804. If student attended an institution with rank of 2, versus an institution with a rank of 1, the odds are decreased by -0.675.

```
> confint(res)# will calculate the confidence interval for the coefficients.

                         2.5 %       97.5 %
(Intercept)      -6.2716202334 -1.792547080
gre               0.0001375921  0.004435874
gpa               0.1602959439  1.464142727
as.factor(rank)2 -1.3008888002 -0.056745722
as.factor(rank)3 -2.0276713127 -0.670372346
as.factor(rank)4 -2.4000265384 -0.753542605
```

Pseudo R^2 for glm

In normal $lm()$ function the R^2 values are calculated as the ratio of the Explained Sum of Squares (ESS) and the Total Sum of Squares (TSM).
Let us consider the following example

```
> y=c(8.05,3.78,7.77,8.78,4.00)
> x=c(0.71,0.01,0.87,0.89,0.23)
```

Total sum of squares can be calculated using the formula $\sum_{i=1}^{n}(y_i - \bar{y})^2$. Thus TSS can be calculated as

```
> TSS=sum((y-mean(y))^2)
> TSS

[1] 22.85932
```

Now let us fit the model

```
> res=lm(y~x)
> summary(res)

Call:
lm(formula = y ~ x)

Residuals:
      1       2       3       4       5
 0.5993  0.3906 -0.6090  0.2849 -0.6658

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.3314     0.5568   5.983  0.00935 **
x             5.8019     0.8579   6.763  0.00661 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6849 on 3 degrees of freedom
Multiple R-squared:  0.9384,        Adjusted R-squared:  0.9179
F-statistic: 45.74 on 1 and 3 DF,  p-value: 0.006605
```

Now we can calculate the Explained sum of squares from the model as follows

```
> PR=3.331+5.802*x
> PR

[1] 7.45042 3.38902 8.37874 8.49478 4.66546

> ESS=sum((mean(y)-PR)^2)
> ESS

[1] 21.45289
```

Here PR is the predicted values using the model. Now we can calculate the R^2 as

```
> ESS/TSS

[1] 0.9384744
```

For the Linear models the response(observed variables) are continuous variables, whereas for the glm the response variables are categorical variables. Thus the R^2 values cannot be calculated and various pseudo R^2 values has been proposed for the model fit. Let us consider such a method in the following section. Null deviance is the sum of residuals when the model has only intercept means($Y \sim 1$). For our score data Null deviance can be calculated as follows.

```
> score=read.table("score.txt",header=T)
> mod0 = glm(admit~ 1,family=binomial, data=score)

> out

[1] "    Null deviance: 499.98  on 399  degrees of freedom"
[2] "Residual deviance: 499.98  on 399  degrees of freedom"
[3] "AIC: 501.98"
[4] ""
[5] "Number of Fisher Scoring iterations: 4"
[6] ""
```

Here you can see the the Null deviance is 499.98 which same as the sum of the residuals $sum(resid(mod0)^2)$. Now add another term gpe to the model ($y \sim 1 + gre$)

```
> mod1 = glm(admit~1+gre,family=binomial, data=score)
> summary(mod1)

Call:
glm(formula = admit ~ 1 + gre, family = binomial, data = score)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1623  -0.9052  -0.7547   1.3486   1.9879

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.901344   0.606038  -4.787 1.69e-06 ***
gre          0.003582   0.000986   3.633  0.00028 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 499.98  on 399  degrees of freedom
Residual deviance: 486.06  on 398  degrees of freedom
AIC: 490.06

Number of Fisher Scoring iterations: 4
```

Now we can calculate the pseudo R^2 values as 1-(Residual deviance/Null deviance).
Or we can check these two models are significantly different using an F test with *anova*() function.

```
> anova(mod0,mod1,test = "Chisq")

Analysis of Deviance Table

Model 1: admit ~ 1
Model 2: admit ~ 1 + gre
```

```
  Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
1       399     499.98
2       398     486.06  1    13.92 0.0001907 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Mixed linear models

Up to now we only considered the categorical explanatory variable as a fixed effect in the model. However sometimes we have to consider the categorical variable as a random effect in the model. So the categorical variable can have fixed effect or random effect.
The fixed effects influences only the mean of the response variable y
The random effects influences the variance of the response variable y
Suppose if you want to study the average height of people in different cities all over Germany. Suppose if we consider 100 cities all over Germany and we are not interested in the pairwise mean of different cities. We would more like to see who much variation is explained by cities. Such cases we need to model cities as a random factor in the model for that we need to use mixed model(mixed means both random and fixed factor). Fixed effects will have informative factor levels(eg sex). Whereas the random effects have uninformative factor levels(eg. genotype A and B or cities) R provide $lme()$ function as a part of the $lme4$ package for the mixed model analysis. So please install $nlme$ before started to use $lme()$ function. In lme function you have to specify which factor is fixed and which factor is random using $fixed$ and $random$ options. Example:
$> lme(fixed = y \sim 1, random =\sim 1|a)$
There is a faster version $lmer$ available with the package $lme4$. The syntax is $lmer(y \sim 1 + (1|a))$
Here there is no separate formula for fixed and random effect. It is easier to use $lmer$ when you have multiple random factors. The syntax is:
$lmer(y \sim 1 + (1|first\_random\_factor) + (1|second\_random\_factor)$ Example:

```
> dat<-read.table("real_data_lmer.txt",header=T)
> head(dat,3)

  Genotype Region Loc      DH     TSW
1       19   Bale   1 82.8465 38.9677
2       19   Bale   2 73.6415 34.8074
3       41 Sidamo   1 88.1556 32.8589
```

Now define the factors

```
> dat$Loc=factor(dat$Loc)
> dat$Genotype=factor(dat$Genotype)

> library(lme4)
> mod<-lmer(DH~ Region+(1|Genotype)+(1|Loc),data=dat)
```

Here we have region as fixed effect and location and genotype as random effect.
If you want to nest Location inside the region then

```
> mod<-lmer(DH~ Region+(1|Genotype)+(1|Region/Loc),data=dat)
> va=c(39.873,69.697,42.952,5.499)
> 100*va/sum(va)

[1] 25.232722 44.106163 27.181197  3.479917
```

Here you can see that location region interaction explains the most variation 44%.
Suppose if you want to include Region location interaction as random factor in the model then

```
> mod<-lmer(DH~ Region+(1|Genotype)+(1|Region:Loc),data=dat)
```

You can also make model comparison using *anova*() function.
Also you can you either maximum likelihood(ML) or restricted maximum likelihood(REML) for the estimation by setting $REML = T/F$. In ML method the effect of fixed effect is first removed. Normally the REML method is preferred for the estimation of variance and covariance parameters. When the response variable is categorical variable we can use function *glmer*() in order to fit the data.
Let us read the cancer.txt file.

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

In this data file the cure of the cancer patient is mentioned as 0 and 1. Also the experience of the Doctor, cancer stage, body mass index and C reactive protein measured. Additionally the ID of the doctor who treated the patient is recorded. We want to measure the variance due to the doctor ID.

Least Squares Mean

These are the means estimated from a linear model. Least squares means also known as adjusted mean are less sensitive to missing data. Load the library *lsmeans*. Then create a model object

```
> library(lsmeans)
> mod<-lmer(DH~ Region+(1|Genotype),data=dat)
> ls=lsmeans(mod,list("Region"))
> summary(ls)

$`lsmeans of Region`
 Region    lsmean       SE  df lower.CL upper.CL
 Arsi    80.23922 1.685666 189 76.91408 83.56435
 Bale    78.58706 1.280356 189 76.06144 81.11268
 Gamo    79.29286 1.305712 189 76.71722 81.86850
 Gojjam  79.49568 1.968434 189 75.61275 83.37860
 Gonder  80.65979 1.538796 189 77.62437 83.69521
 Harerge 79.40276 1.391893 189 76.65712 82.14841
 Shewa   80.61228 1.497754 189 77.65782 83.56674
 Sidamo  79.17049 1.256422 189 76.69208 81.64890
 Tigray  79.16382 1.632139 189 75.94427 82.38337
 Wollo   79.72523 1.459830 189 76.84558 82.60489

Degrees-of-freedom method: satterthwaite
Confidence level used: 0.95
```

Non-linear regression

Some time we cannot linearize relationship between the response variable ,y and the explanatory variable x. In such ceases we have to use non-linear regression. R provided nls() function for the non-linear regression analysis. A non-linear model allows us more complicated relationships. In our example data the jaw bone length of the dear is measured at different ages and the relationship between bone length and age is a exponential with three parameters.

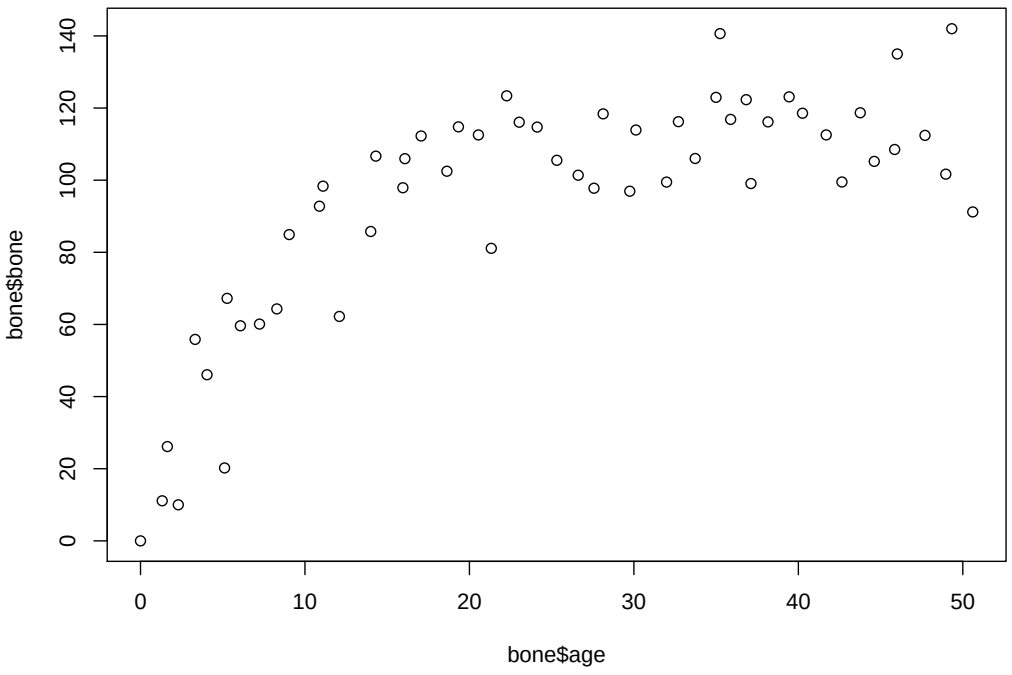
$$y = a - b * e^{-c*x} \tag{1}$$

So our model look like $y \sim a - b * e^{-c*x}$.The most difficult thing here is we need to specify some starting values for the parameters a, b and c. For an initial guess let us look at the limits of the equation at $x = 0$ and $x = \infty$. At $x = 0$ $exp(0) = 1$ so y=a-b is the intercept. The asymptotic value (is a line and distance between the curve and the line approaches zero as they move to infinity). Thus $exp(-\infty)$ is 0, hence y=a is the asymptotic.

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

```
        age      bone
1  0.000000   0.00000
2  5.112000  20.22000
3  1.320000  11.11130
4 35.240000 140.65000
5  1.632931  26.15218
6  2.297635  10.00100
```

```
> plot(bone$age,bone$bone)
```



By looking at the graph you can see the asymptote is around 120 and the intercept is around 10. So at $x = 0$ our intercept is a-b so b=120-10.

The curve is rising steeply around the age 4 and bone length 40. Using these values in equation (1) gives us a c value around 0.063.

```
> res=nls(bone~a-b*exp(-c*age),start=list(a=120,b=110,c=0.06),data=bone)
> summary(res)

Formula: bone ~ a - b * exp(-c * age)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a 115.2528     2.9139   39.55  < 2e-16 ***
b 118.6875     7.8925   15.04  < 2e-16 ***
c   0.1235     0.0171    7.22 2.44e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 13.21 on 51 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 3.532e-06
```

Here the value of a is 115.25 with standard error 2.91 and b is 118.6875 with standard error 7.89. So they are significantly not different. So we can consider a and b as similar and fit a simpler model.

Multivariate statistics

Multivariate statistical methods do not have any response variables, and in multivariate analysis we are trying to find the structure/pattern in the data. R implement different multivariate techniques and some of the most commonly used ones are: Principal component analysis (*prcomp*())
Factor analysis ($factanal()$)
Cluster analysis (*hclust*(), *kmeans*())
Discriminant analysis (*lda*, *qda*)

Principal Component analysis(PCA)

Principal component analysis(PCA) is one of the most commonly used multivariate statistical method. The main idea behind PCA is to reduce the the dimensionality of the data without loosing much information. PCA is a way to identify patterns in data especially in high dimensional data. One of th main advantage of PCA is that you can reduce the number of dimensions without loosing much information. PCA is done by converting a set of correlated observations into a set of values of linearly uncorrelated variables called principal components. After the transformation the number of principal components is less than or equal to the number of original variables. PCA can be done by two methods if the matrix is covariance matrix then eigenvalue decomposition is used whereas if the matrix is a data matrix then singular value decomposition is used. There are two functions in R to carry out PCA. *prcomp* and *princomp*. The preferred method is *prcomp* because of numerical accuracy.
First read the table.

```
> dat=read.table("car.txt",header=T,row.names=1)
> head(dat,5)
```

	Cpower	Horsepower	Speed	Weight	Width	Length
Citroen_C2	1024	61	158	922	1659	3666
Smart_Fortwo	698	52	135	730	1515	2500
Mini	1498	160	218	1115	1690	3625
Nissan_Micra	1240	65	154	965	1660	3715
Renault_Clio	2246	205	245	1400	1210	3812

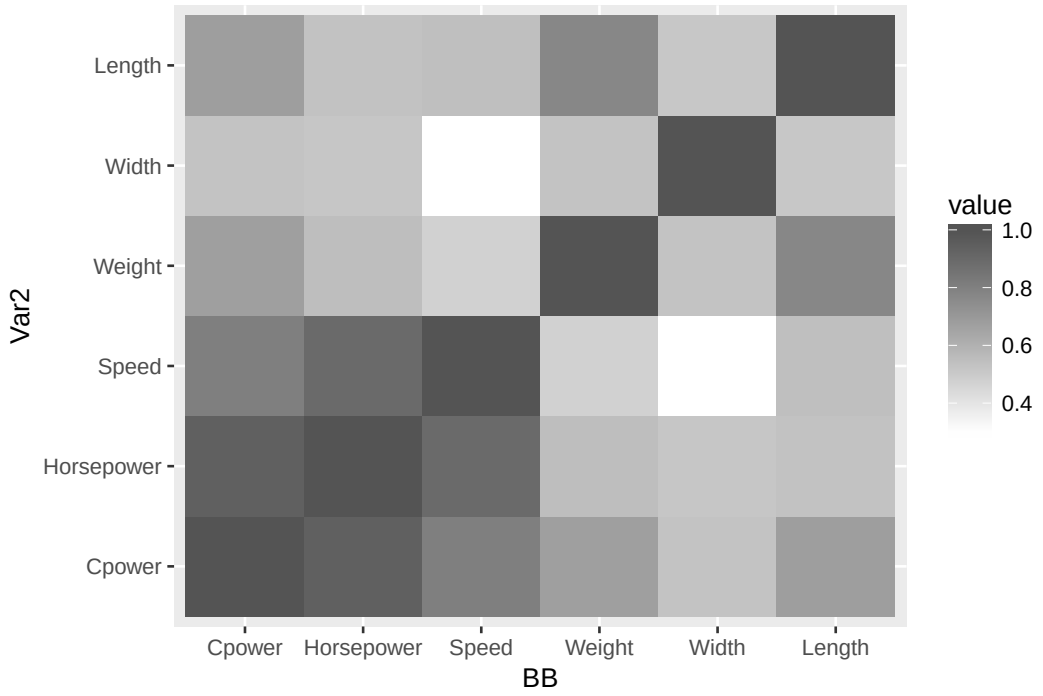
Then look at the correlation table.

```
> co=cor(dat)
> as.dist(co)
```

	Cpower	Horsepower	Speed	Weight	Width
Horsepower	0.9432701				
Speed	0.8099404	0.8990369			
Weight	0.6755096	0.5501838	0.4739109		
Width	0.5304039	0.5179371	0.2965048	0.5280138	
Length	0.6788427	0.5344063	0.5429795	0.7765072	0.5143819

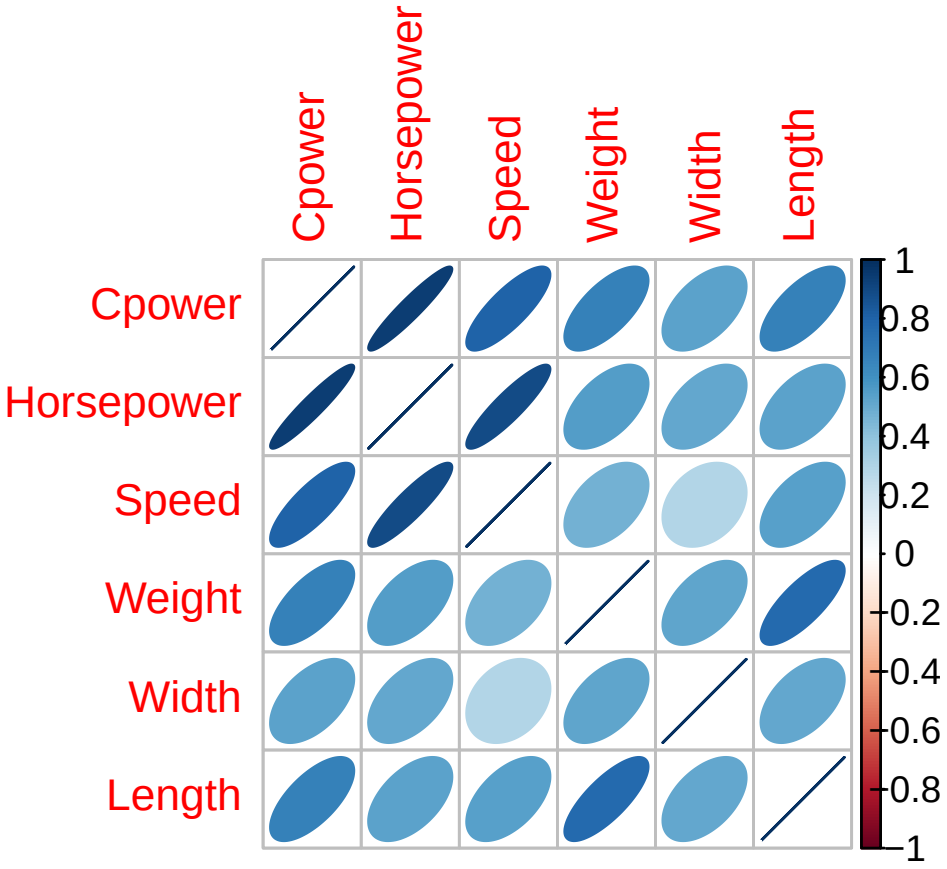
From the tables you can see that some variables like "CCpower" and "Horsepower are highly correlated. Here we can represent the correlation matrix using a heat map. Heat map is a graphical representation of the values in a matrix using different colors. If the values are the same those points are represented using the same color.

```
> library(ggplot2)
> library(reshape2)
> qplot(x=Var1, y=Var2, data=melt(cor(dat)), xlab="BB",fill=value, geom="tile")+
+ scale_fill_gradient(low = "white", high = "grey33")
```



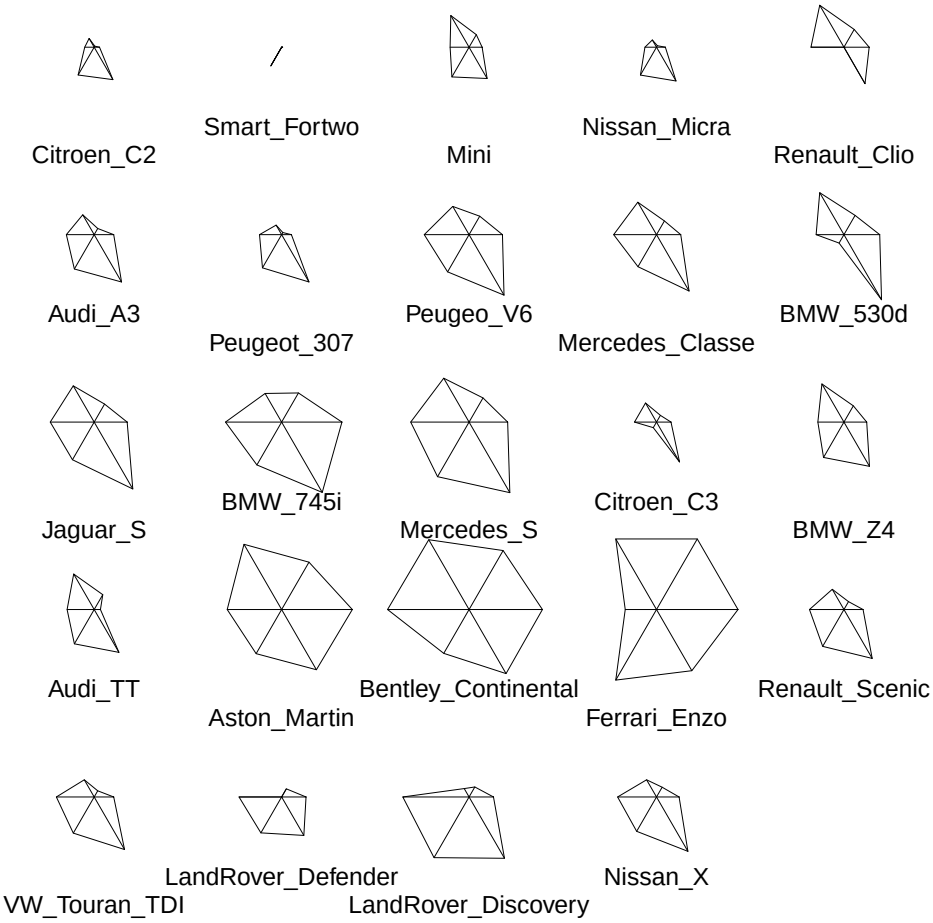
Library "corrplot" gives even nicer functions to plot the correlation graph.

```
> library(corrplot)
> corrplot(cor(dat), method="ellipse")
```



Or even you can look at a star plot

```
> stars(dat,nrow=5)
```



In the plot length of each spoke is proportional to the magnitude of the variable under consideration.

```
> pc=prcomp(dat,scale=T)
```

The output from the *prcomp*() give the following information.

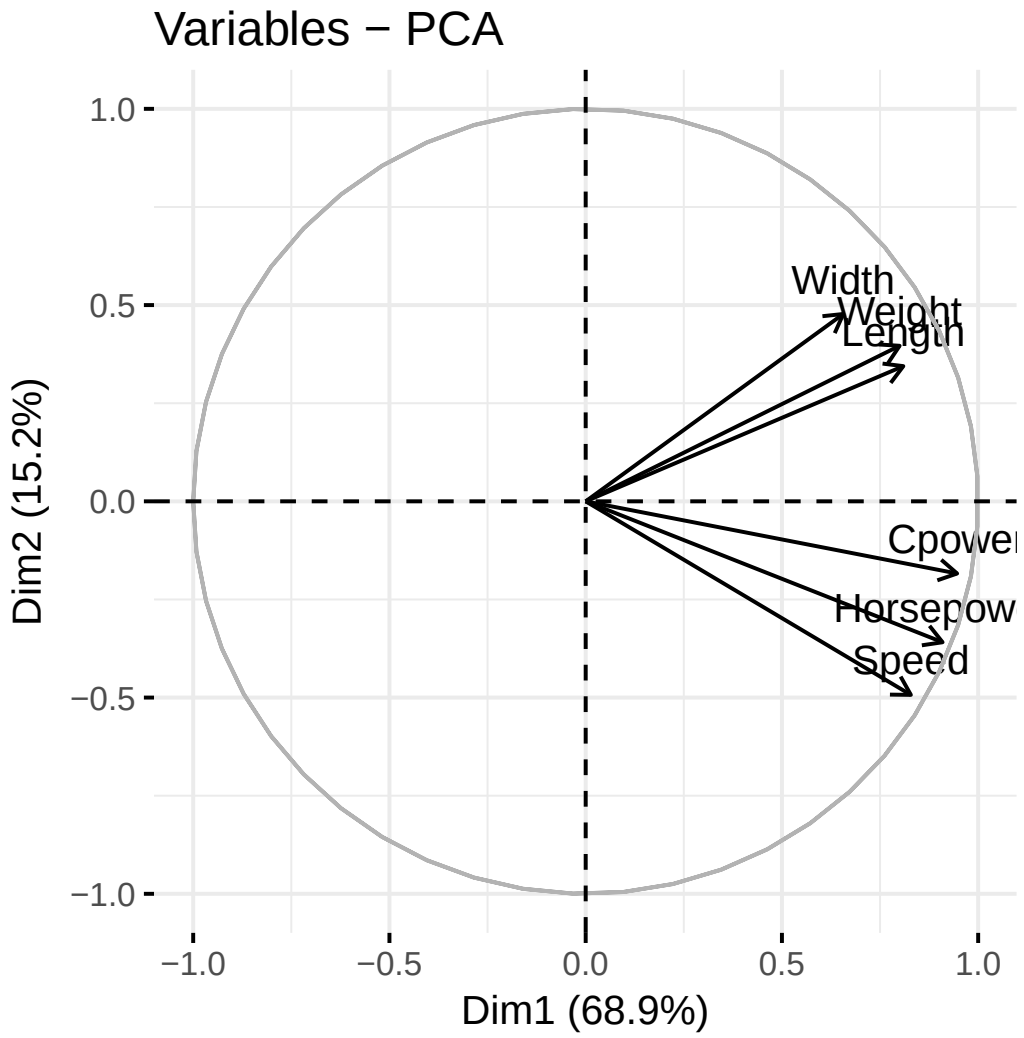
```
> names(pc)
```

```
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

Here *sdev* contains the square root of the eigenvalues. $pc\$sdev^2$ will give the variance explained by each principal component. *center* is the mean and *scale* is the standard deviation of the columns of the data set. *rotation* is contains the eigen vectors corresponds to the eigen values.Finally x is the scaled data multiplied by the *rotation*.

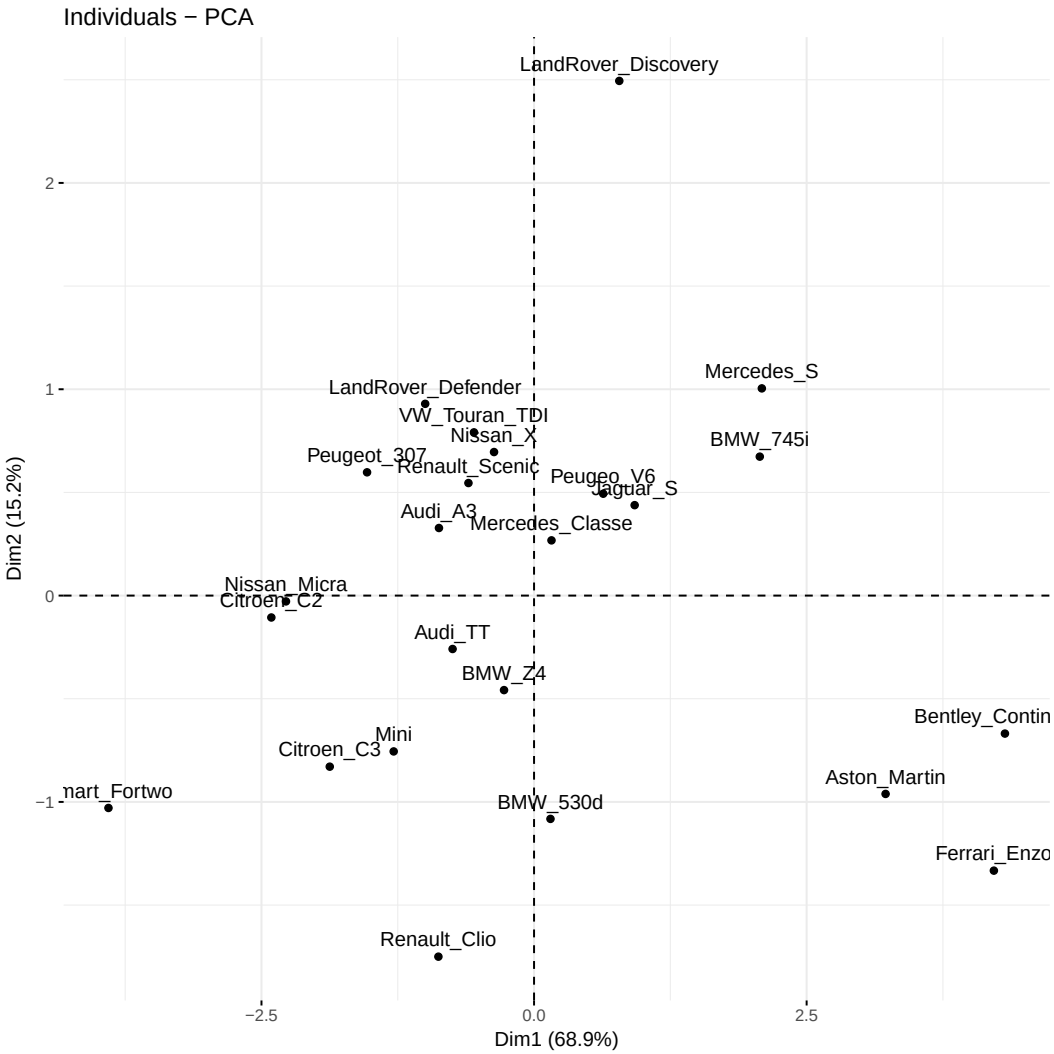
We can use the package "factoextra" which provide nice functions to plot the PCA components based on ggplot2. In order to plot the graph of variables we can use $fviz_pca_var()$ function.

```
> library(factoextra)
> fviz_pca_var(pc)
```



For plotting the individuals $fviz_pca_ind()$ can be used.

```
> library(factoextra)
> fviz_pca_ind(pc)
```



FactoMineR is another package, which can be used for PCA analysis.

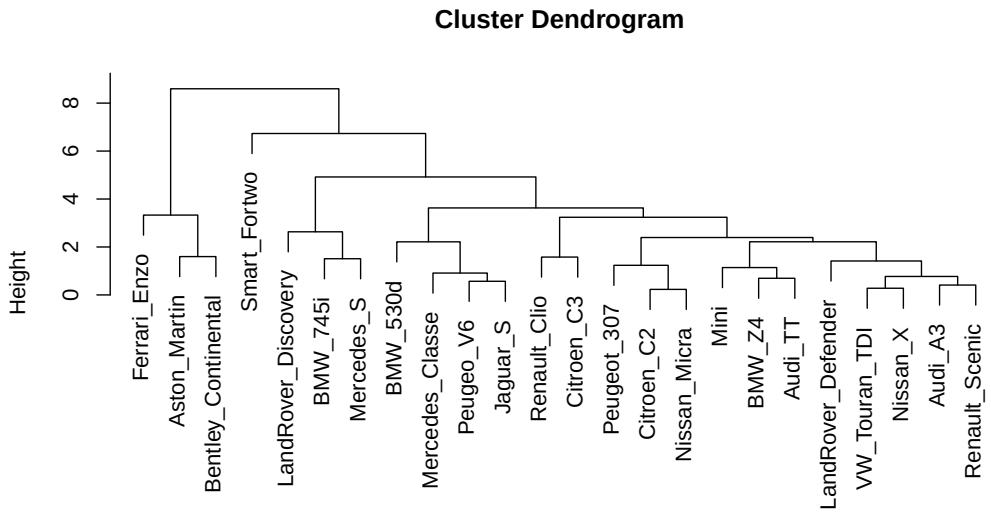
```
> library("FactoMineR")
> pca=PCA(dat,graph=F)
```

Cluster analysis is used to divide objects into meaningful groups. Hierarchical clustering is one of the commonly used method in clustering. We can do a cluster analysis for the transformed data based on the PCA analysis. First we need to find the dissimilarly between the data points using function *dist*(). This function calculate the euclidean distance between the rows in a matrix or data frame.

```
> dst=dist(pca$ind$coord)
```

Here *pca*$*ind*$*coord* is a matrix of the rotated data. Now we can perform a clustering using the *hclust*() function in R.

```
> clust=hclust(dst)
> plot(clust,xlab="",sub="")
```



Let us plot the clusters now. First you have to assign each car into a cluster and that can be done using the function *cutree*(). Now we assign the cars into 2 groups using $k = 2$ option.

```
> three=cutree(clust,k=2)
```

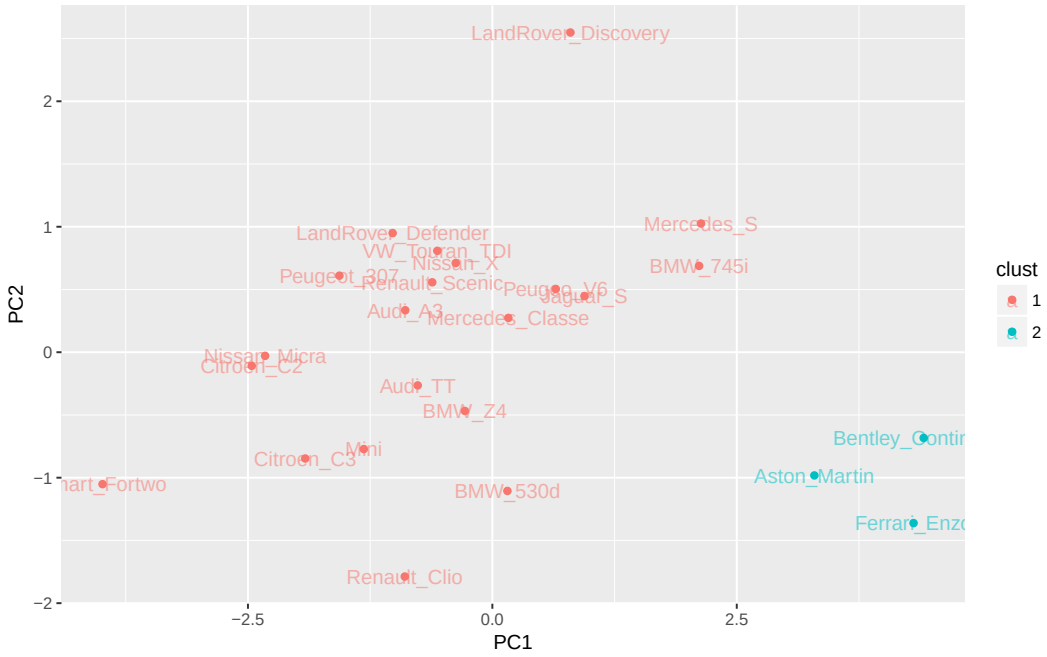
Then we need to create a new data frame with the coordinates to plot from the rotated data and add a column with the group information.

```
> obs=data.frame(pca$ind$coord[,1:2])
> obs$clust=as.factor(three)
> head(obs,4)
```

```
                Dim.1       Dim.2 clust
Citroen_C2   -2.461299 -0.10831459     1
Smart_Fortwo -3.987270 -1.05134100     1
Mini         -1.315167 -0.77151470     1
Nissan_Micra -2.324196 -0.02872455     1
```

Now you can plot the ggplot with different colors for different group.

```
> ggplot(obs,aes(x=Dim.1,y=Dim.2,label=rownames(dat)))+geom_point(aes(color=clust))+
+ geom_text(aes(color=clust),alpha=0.55,size=4)+xlab("PC1")+ylab("PC2")
```



Let us consider another example here the file "*rice_all_pheno.txt*" is a rice field data having 413 lines and 16 phenotypes. We want to see which traits are correlated based on a correlation plot. Additionally we can also visualize them using the PCA. Read the data and we have lots of missing values

```
> dat_rice=read.table("rice_all_pheno.txt",header=T,row.names=1,sep="\t",na.strings="NA")
```

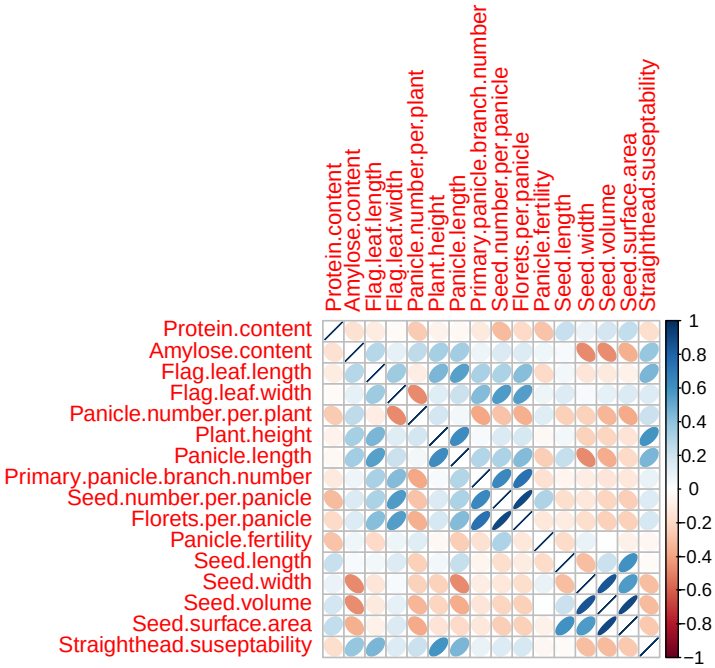
We cannot include missing values in PCA. So we need to select only lines having phenotypic information for all the 16 traits. This can be done using the *complete.cases*() function.

```
> dat_sub= dat_rice[complete.cases( dat_rice),]
> dim(dat_sub)
```

```
[1] 314  16
```

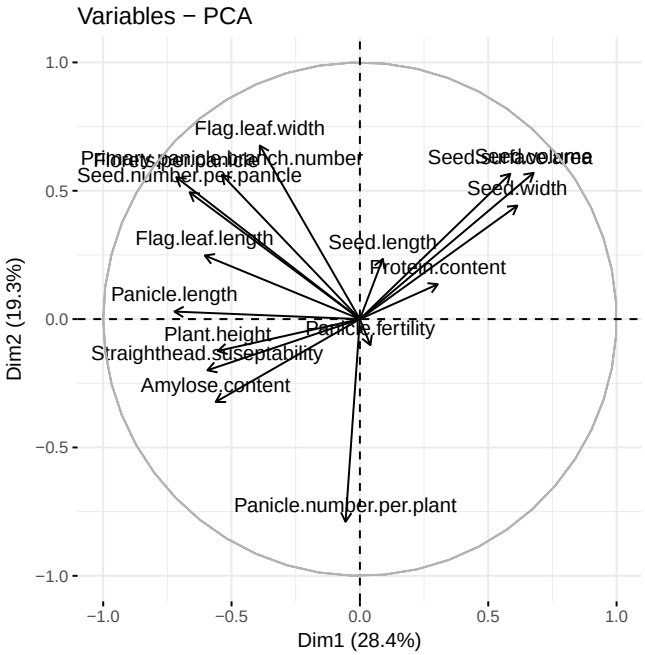
Now we can see that only 314 lines having data for all the 16 traits. First check the correlation plot.

```
> library(corrplot)
> corrplot(cor(dat_sub), method="ellipse")
```



Let us check the PCA and plot the coordinates

```
> pc=prcomp(dat_sub,scale=T)
> library(factoextra)
> fviz_pca_var(pc)
```



From the plot you can see that some traits are highly correlated. Let us consider two traits *Plant.height* and *Straighthead.suseptability* .They are close to each other and let us check the correlation.

```
> cor(dat_sub$Plant.height,dat_sub$Straighthead.suseptability)
```

```
[1] 0.593168
```

Consider two traits *Amylose.content* and *Seed.width* they are on opposite direction and calculate the correlation

```
> cor(dat_sub$Amylose.content,dat_sub$Seed.width)

[1] -0.472095
```

You can see both these traits are negatively correlated and hence they are on the opposite directions in the PCA plot.

Quantitative Trait Loci

QTL is a DNA sequence that influence a quantitative trait. QTL can be a gene or a group of genes. We need phenotypic observations and genetic marker data to perform a QTL analysis. Single marker mapping, interval mapping and composite interval mapping are traditional QTL mapping approaches. However there are some variants eg. family based mapping approach, are available.
The single marker analysis(sometime called marker regression) is the easiest one. The single marker analysis is simple, easily extend to more complicated models. The drawbacks are, suffers when the marker density is low, gives imperfect information about QTL location and only consider one QTL at a time. In interval mapping can take proper account of missing data along with improved estimate for QTL effects.The disadvantages are increased computation time and require specialized softwares for the analysis. Composite interval mapping is a modified interval mapping approach using a subset of marker loci as covariates.This approach can increase the resolution of interval mapping, by accounting the linked QTLs.However, recently many studies showed that when the number of markers are high(covering all the genome) both interval mapping and single marker approach provide the same results. The QTL mapping problem can be split into two parts missing data problem and model selection problem(Broman 2009). Model selection means you have to find a proper model which can include different covariates and the different marker interactions. The genetic marker is associated with QTL and we generally only observe the genotype of the marker(not the genotype of the QTL), this refer to the missing value problem
Now a days people prefer to use Single Nucleotide Polymorphism (SNP) as marker for the QTL analysis. The reasons are SNP is so common and evenly distributed along the genome and the the methods for SNP detection can be easily automated.
What is the difference between association mapping and QTL mapping?. QTL analysis are more general than association analysis. QTL analysis will tell us about the location of the genes which are influencing a trait(the location in the chromosome can vary from one million to 20 million) whereas the association mapping can tell us more precise location of the gene(may be within thousands or even less number of base pairs). Association analysis is more focused on specific traits of interest so the markers in association analysis should come from candidate genes which affect the trait. Whereas in QTL analysis the marker come from anywhere in the genome all matters is that the markers should cover all genome. In association analysis it is really important to consider factor like the population structure in the analysis, otherwise such factors can cause false positive results.
Some of the questions before you start the QTL mapping: start with the design of experiment, means choice of strains, which phenotypes to measure, perform back cross or inter cross?, which individuals should be genotype, what markers should be genotyped.

R/qtl

R/qtl is R package for mapping quantitative trait(QTL). Using R/qtl you can estimate genetic map,single-QTL analysis,two-QTL mapping, interval mapping and multiple imputation. Also you can detect genotying errors. Moreover R/qtl provides functions to simulate various dataset(F2-population, BC(back cross) population).
Install the package *'qtl'*.

In the note $'>'$ symbol is used to represent R commands.
$> install.packages('qtl')$

```
> library(qtl)
```

Simulation in R/qtl

R/qtl provides various functions to simulate simple and more complex(with epistatic interactions) datasets. Let us start with *sim.map*. *sim.map*() function is used to create maps. If you want to create a map with single chromosome of length 200 cM and having markers equally spaced:

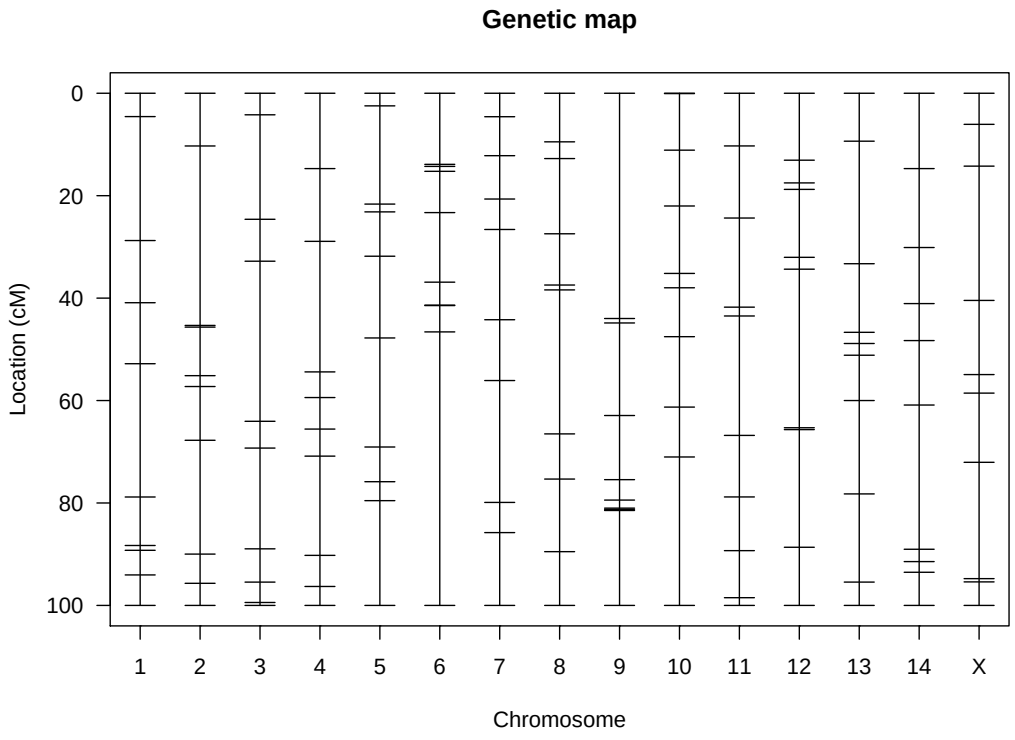
```
> map_sim=sim.map(200,11,include.x=F,eq.spacing=T)
```

The option *include.x* is used to include a X chromosome in the map. Suppose if you want to develop a map with 10 chromosomes all having length 100 cM and each chromosome containing 10 randomly positioned markers:

```
> map_sim=sim.map(rep(100,15),10)
```

To draw the map:

```
> plot(map_sim)
```



To display the summary of the map:
$> sumary(map\_sim)$

Once you develop a map then you can simulate the actual data using function *sim.cross*():

```
> dat_sim=sim.cross(map_sim,n.ind=10,type="bc")
```

If you want to have an inter cross then use $type = "f2")$
If you want to see how your simulated data look like:
$>$ *names*(*dat_bar*) shows that *geno* and *pheno*
If you want to display values of phenotype and genotypes then use: *dat_bar*$*geno* and *dat_bar*$*pheno* to display the genotypes and phenotypes respectively. If you want to define a QTL on the chromosome use *model* option:
Example for a back cross:
$>$ $model = c(2, 30, 0.5)$ # here you define a QTL on the 2 chromosome at 30 th position having an additive QTL effect of 0.5 For a inter cross(F2) you have to define dominance effect also in the *model* option.
$>$ $model = c(2, 30, 0.5, 0.2)$ here 0.2 is the dominance effect. If you want to define more QTLs then:

```
> mymod=rbind(c(2,20,3,0.0),c(3,30,4,0.0),c(4,50,5,0.0))
```

Then simulate the data using defined QTLs.

```
> dat_sim=sim.cross(map_sim,n.ind=10,type="f2",model=mymod)
```

To simulate missing data you can use the option $missing.prob = 0.05$(will generate data with 5% missing genotypes)

Checking map order

It is important to check the markers are placed on the chromosome in a correct order. For that, the first step is to estimate the recombination fraction between the marker. You can use $est.rf()$ function to calculate the recombination fraction:

```
> rf=est.rf(dat_sim)
```

Some more useful functions
geno.crosstab() Will create a cross tabulation of the genotypes at a pair of markers.
Example:

```
> geno.crosstab(dat_sim,"D1M1","D1M2")
```

```
    D1M2
D1M1 - AA AB BB
  -  0  0  0  0
  AA 0  5  0  0
  AB 0  1  2  1
  BB 0  0  0  1
```

Plotting the recombination fraction

$>$ $plot.rf()$ # will display the estimated recombination fraction in the upper left and LOD scores on the lower part.

Red (low r and high LOD) indicates the markers are linked and blue indicates they are not linked(low LOD and high r) In order to reestimate the inter maker distance and draw the genetic map use function *est.map*()

Example:
$> new\_map = est.map(dat\_bar)$
$> plot(new\_map)$
Once you draw the new reestimate map you can see whether there is some map extension. Based on the map and recombination fraction you can decide whether to re arrange the markers in the map. For the rearrangement you can use the function $movemarker()$.
Example:
$> dat\_bar = movemarker(dat\_bar, find.marker(dat\_bar, 4, index = 7), 1)$
Then you can replace the old map with the new rearranged map.

$> new\_map = est.map(dat\_bar)$
$> dat\_bar = replace.map(dat\_bar, new\_map)$
$> plot.map(dat\_bar)$
In order to find if there exist some genotyping errors you can use $cal.errorlod()$ function. Also you can count the number of crossover for each individuals using $countXO()$ function.
The $bychr = T$ option will return a matrix containing the number of crossovers on the individual chromosome.
$> cross = countXO(dat\_bar, bychr = T)$

Field data

Often you will face problems to import data into r/qtl. The developers suggest the comma-delimited format might be the best way to import data into r/qtl package. You can create the comma-delimited(csv) in OpenOffice or Microsoft Excel.
The initial columns are the phenotypes(you have to specify at least one phenotype) followed by the markers. The first row contains the phenotype and marker names and the second row should have empty field for the phenotype columns. Whereas for the genotype columns the second row should contain chromosome assignments(numbers are the best). For the genotype columns and optional third row contains the centiMorgan(cM) positions of the genetic markers(the phenotypic columns should be blank). Missing values should be indicated "NA" or "-".
Once you have formatted the data you can use $read.cross()$ function to read the data.
Example:

```
> dat=read.cross("csv","","qtl_data.csv",na.string="NA",genotype=c("0","1"))

 --Read the following data:
        301  individuals
        364  markers
        1  phenotypes
 --Cross type: bc
```

Here the first argument is the type of the file(it can be csv,csvr,csvs) the second argument is the directory name, third argument is the file name, fourth argument tells how the missing values are coded in the file and the final argument is how the genotypes are coded. Also you can use the data formats used by MapMaker and QTL Cartographer.
If the genetic map positions are not provided then you ca estimate the estimate the inter marker distance using the option $estimate.map = T$. You can save the cross object into the disk using the function $write.cross()$.
Syntax is:

$> write.cross(cross\_object, "format", "file\_name")$

Example:

```
> write.cross(dat,"csv","test.csv")
```

Exploring the data

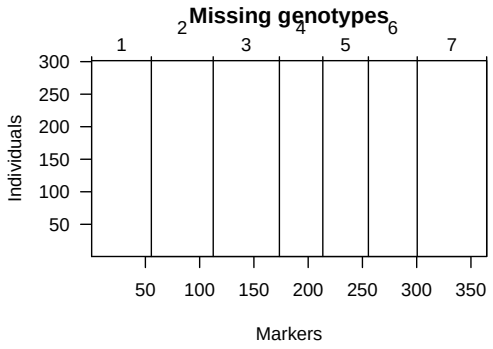
Once you successfully uploaded the data you can check the data using various functions.
For a short summary

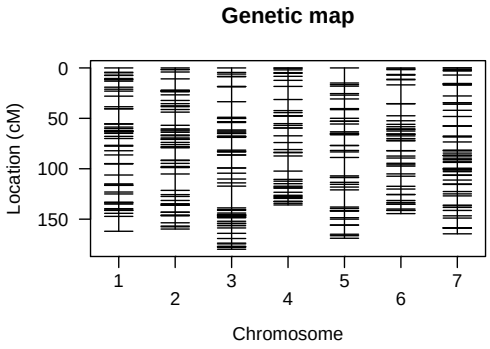
```
> summary(dat)
```

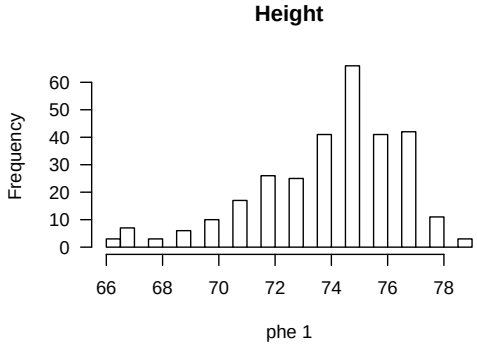
```
    Backcross

    No. individuals:    301

    No. phenotypes:     1
    Percent phenotyped: 100

    No. chromosomes:    7
        Autosomes:      1 2 3 4 5 6 7

    Total markers:      364
    No. markers:        55 57 61 40 42 45 64
    Percent genotyped:  100
    Genotypes (%):      AA:15.5  AB:84.5
```

If you want to plot phenotype and the map for the uploaded data

```
> plot(dat)
```







If you want to separate the map then

```
> map=pull.map(dat)
```

$>$ *plot*(*map*) Will display the map

Check the map order

Next thing is to do estimate the recombination fraction between the markers.

You can use $est.rf()$ function to estimate the recombination fraction. This will calculate the recombination fraction and will add to the table *dat*.

```
> dat=est.rf(dat)
```

calc.genoprob() function will calculate the genotype probabilities.

checkAlleles() function will identify markers whose alleles might have been switched by comparing the LOD score(after calculating the recombination frequency).

```
> checkAlleles(dat)
```

```
No apparent problems.
```

Then you can estimate the map using function *est.map*(). Example:

```
> map=est.map(dat)
```

Then you can plot it
$>$ *plot*(*map*)

Single-QTL analysis

The most commonly used QTL mapping method is interval mapping. The simplest method for QTL analysis is to consider each marker individually, split the lines into groups according to their genotypes at the marker and compare the groups phenotypes. But this approach is seldom used in practice.
The disadvantages of single-QTL analysis is that we only consider one QTL at a time and we cannot separated liked QTLs, moreover we cannot estimate possible QTL interactions. In 'r/qtl' the single-QTL analysis can be done using function *scanone*() Example:

```
> meth_s=scanone(dat,method="mr")
```

Let us print the summary

```
> summary(meth_s)
```

```
          chr   pos    lod
GBM1042     1  40.5  2.873
GBM1052     2  42.0 22.462
bPb_9110    3 118.7 19.200
HDAMYB      4 146.9  3.678
bPb_0071    5 126.8  0.997
EBmac624    6  68.2  3.064
bPb_9914    7 103.4  4.252
```

If you want to find QTLs in separate chromosomes:

```
> meth_s=scanone(dat,chr=1,method="mr")
```

If you have many phenotypes in the file you can specify the phenotype using *pheno.col* option

```
> meth_s=scanone(dat,chr=1,pheno.col=1,method="mr")
```

In order to plot the LOD score:
$> plot(meth\_sing)$

Interval mapping

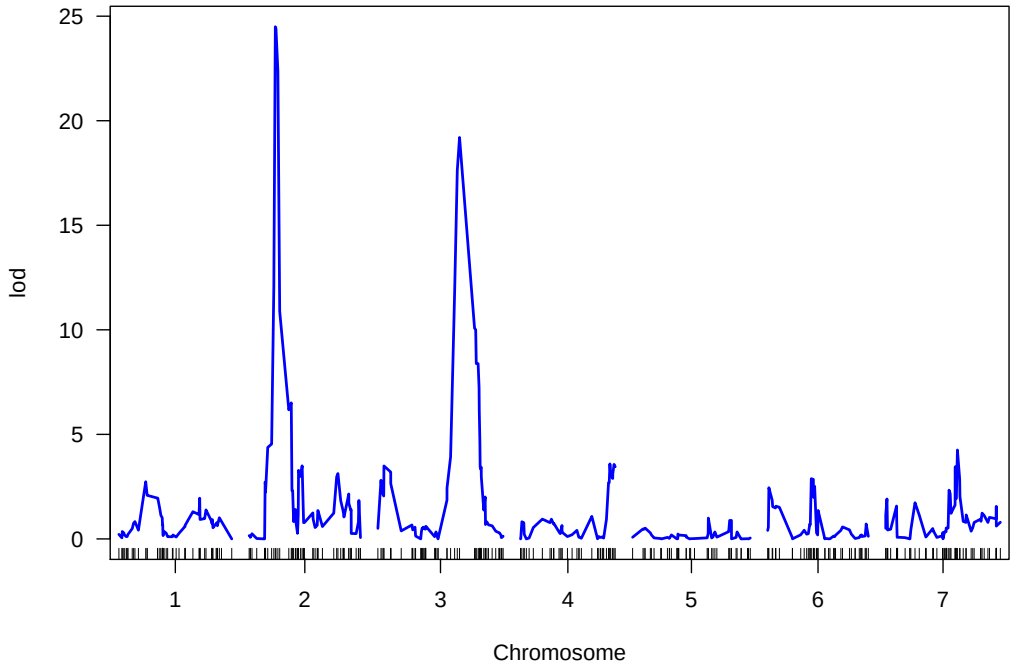
There are various interval mapping methods. But the commonly used method is the maximum likelihood estimation method. In standard interval mapping we assume a grid of positions a long the genome as possible location for a QTL. Interval mapping can provide a clear understanding of the QTL location, however interval mapping approaches are computationally intensive.
In interval mapping approach first we have to calculate the conditional genotype probabilities. You can use *calc.genoprob*() function to calculate the conditional genotype probabilities.

```
> dat=calc.genoprob(dat)
```

Then scan for QTL using maximum likelihood estimation method:

```
> meth_int=scanone(dat,method="em")
```

```
> plot(meth_int,col="blue")
```




```
> summary(meth_int,threshold=3)
```

```
          chr   pos   lod
PpdH1       2  41.1 24.49
bPb_9110    3 118.7 19.20
VrnH2       4 141.0  3.59
bPb_9914    7 103.4  4.25
```

You can plot the LOD scores from the interval mapping and marker regression approach as one figure.

Imputation

Missing data is one of the major problems faced in QTL analysis. R/qtl provides functions for multiple imputation for the missing genotypes. But such imputations should be done multiple times and the final result is a combination of all the results from multiple imputations. Genotypes are imputed randomly, but conditional on the observed marker type. Example:

```
> meth_imp=scanone(dat,method="imp")
```

```
> summary(meth_imp,threshold=3)
```

```
         chr   pos   lod
PpdH1      2  41.1 25.67
bPb_9110   3 118.7 19.20
VrnH2      4 141.0  3.58
bPb_9914   7 103.4  4.25
```

Multiple QTL models

Single marker analysis and interval mapping consider single QTL. Composite interval mapping approach can fit multiple QTLs. In composite interval mapping approach you can separate linked QTLs, moreover you can find the interactions among QTLs. In interval mapping approach one putative QTL will be added as covariate in the search for further QTLs. First you have to calculate conditional genotype probabilities.

```
> dat=calc.genoprob(dat)
```

Then do a interval mapping and find marker with highest LOD score.

```
> meth_com=scanone(dat)
```

Then get the genotype information for the marker having peak LOD score.

```
> g=pull.geno(dat)[,"GBM1052"]
```

Then include the genotype information as a covariate and search for further QTLs.

```
> meth_cov=scanone(dat,addcovar=g)
> summary(meth_cov)
```

```
         chr   pos   lod
Mla12      1  38.5  3.26
PpdH1      2  41.1  2.04
bPb_9110   3 118.7 31.03
VrnH2      4 141.0  3.33
bPb_0071   5 126.8  2.79
EBmac624   6  68.2  2.26
BMS64      7 100.3  2.90
```

QTL effects

The function *scanone* will not provide the estimate for QTL effect. You can use $effectplot$ function to plot the phenotype averages for the genotypes at an estimated QTL.

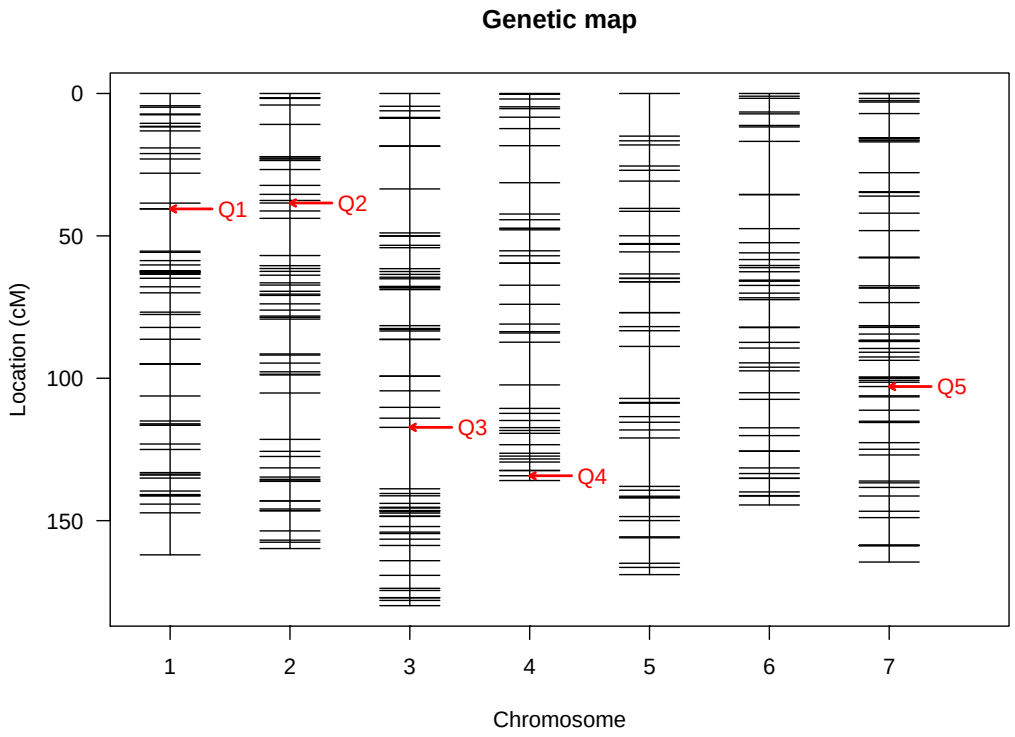
```
> effectplot(dat,mname1="GBM1052")
```

Here $mname1$ argument is used to provide the marker name. If you want to plot more than one marker then $mname2$ $mname3$ and so on. In order to see the location of QTL position in the map you can use *makeqtl* function.

```
> dat=sim.geno(dat)
```

sim.geno() function will impute the missing genotypes.

```
> qt=makeqtl(dat,chr=c("1","2","3","4","7"),pos=c(40.5,42.0,118.7,146.5,103.4))
> plot(qt)
```



Permutation test

Some times you have to estimate the genome wide LOD score and that can be done by a permutation test. In permutation test the phenotypes are shuffled randomly and calculate the height LOD score. The process is repeated n times and the $95th$ percentile will be used as the significant threshold for the population. Example let us do a permutation test.

```
> per=scanone(dat,n.perm=10)

Permutation 1
Permutation 2
Permutation 3
Permutation 4
Permutation 5
Permutation 6
Permutation 7
Permutation 8
Permutation 9
Permutation 10

> summary(per)

LOD thresholds (10 permutations)
     lod
```

```
5%  2.55
10% 2.42
```

Here option *n.perm* defines the no of permutations to perform.

Duplicated markers

Some times you will have duplicated markers(exact duplicates) and may need to find them. Duplication mainly occur due to LD between marekrs which are close to each other.
$findDupMarkers()$ function can be used to find the duplicated marker. Let us consider an example.

```
> dup <- findDupMarkers(dat, exact.only=FALSE)
> length(dup)

[1] 30

> head(dup)

$bPb_9608
[1] "bPb_7137"

$bPb_0482
[1] "bPb_1604"

$HVM20
[1] "Bmag211" "MGB325"

$bPb_6911
[1] "bPb_1213"

$bPb_5014
[1] "bPb_5198"

$HvGOGAT
[1] "HvFT4"
```

Here we have around 30 duplicated markers and let us drop them from the original data set and create a new data set called dat.nodup with out any duplicates.

```
> dat.nodup <- drop.markers(dat, unlist(dup))
> summary(dat.nodup)

    Backcross

    No. individuals:    301

    No. phenotypes:     1
    Percent phenotyped: 100
```

```
No. chromosomes:    7
    Autosomes:      1 2 3 4 5 6 7

Total markers:      327
No. markers:        49 52 53 38 37 44 54
Percent genotyped:  100
Genotypes (%):      AA:15.7  AB:84.3
```

Now you can see that our number of markers is 327 (original no was 364)

Genomic selection

Recent advances in low cost high throughput DNA sequencing technologies have helped genome-wide association mapping (GWAS) to emerge as an alternative to linkage mapping and which offers high mapping resolution and is more time-efficient.Despite the availability of large number of single-nucleotide polymorphisms (SNPs), standard GWAS analysis methods consider one SNP at a time and identify the marker-trait association using a single-locus model. Single-locus model is the simplest and most commonly used model to identify associations between SNPs and continuous trait of interest. Let us consider a single locus model for GWAS using rice field data with 413 lines and 3697 SNPs. $rrBLUP$ is the one of the commonly used R package for GWAS.
Install the package 'rrBLUP'.

$> install.packages('rrBLUP')$

Upload the data into R

The one of the most challenging thing is to format the data, which is readable to the package. The phenotypic data should contain the lines and the phenotypic information for at least one trait. Whereas for the genotypic information file the first column contains the marker name followed by the chromosome number, marker position and the line name for each plant. Suppose our SNPs are coded as "T/T","A/A" "A/T". Here "A/T" is the heterozygous. First we need to recode the SNPs to numerical coding (0 and 1) in order to perform GWAS using rrBLUP. For that we can use the package $'synbreed'$. First read the phenotype and genotype and the map information.

```
> library(rrBLUP)
> geno=read.table("rice_geno.txt",header=T,row.names=1,na.strings="NA")
> dim(geno)

[1] 3697  413

> pheno=read.table("rice_pheno_multi_env.txt",header=T,na.strings="NA")
> dim(pheno)

[1] 413   4

> map=read.table("rice_map.txt",header=T)
> dim(map)

[1] 3697    3
```

Here table geno contains the markers information and pheno with three traits. The markers are arranged in rows and lines are in the column. We have 3697 markers and 413 lines.

```
> library(synbreed)
> gp =create.gpData(geno=t(geno))
```

Here *create.gpData*() function will combine genotype phenotype and map information. However at the moment we only provide the genotype and note that we need to transpose the marker information in order to be used in the package synbreed. For that we can use the function $t()$.
Then we can obtain the corresponding numerical coding (0 and 1) using the following command

```
> gp.coded =codeGeno(gp,impute=T,label.heter="alleleCoding")

     Summary of imputation
    total number of missing values                : 65848
    number of random imputations                  : 65848
```

Here $impute = T$ will impute the missing marker information based on the other markers.The option *label.heter* will tell how the heterozygous are coded. If you want to filter based on the minor allele frequency (MAF) use the maf option.
The numerical coding will be stored in a table called geno.

```
> mat=gp.coded$geno
> mat[1:5,1:5]
```

	dd1000015	dd1000036	dd1000055	dd1000252	dd1000336
NSFTV_1	2	2	0	0	0
NSFTV_10	2	0	0	0	0
NSFTV_100	0	0	0	0	0
NSFTV_101	0	0	0	2	0
NSFTV_102	0	0	0	0	0

Here the dominant allele is coded as 0 recessive as 2 and the heterozygous as 1.
Please note that here the package will reorder the markers. So we need to rearrange the markers according to our map.
We can reorder the markers according to the map information and create a new matrix called *newmat*.

```
> newmat=mat[,match(map$marker,colnames(mat))]
> t(newmat[1:5,1:5])
```

	NSFTV_1	NSFTV_10	NSFTV_100	NSFTV_101	NSFTV_102
id1000001	2	1	0	0	0
id1000024	1	0	0	0	0
id1000075	2	0	0	0	0
id1000109	0	0	0	0	0
id1000196	0	0	0	0	0

```
> geno[1:5,1:5]
```

	NSFTV_1	NSFTV_10	NSFTV_100	NSFTV_101	NSFTV_102
id1000001	C/C	T/C	T/T	T/T	T/T
id1000024	G/A	G/G	G/G	G/G	G/G
id1000075	C/C	T/T	T/T	T/T	T/T
id1000109	C/C	C/C	C/C	C/C	C/C
id1000196	C/C	C/C	C/C	C/C	C/C

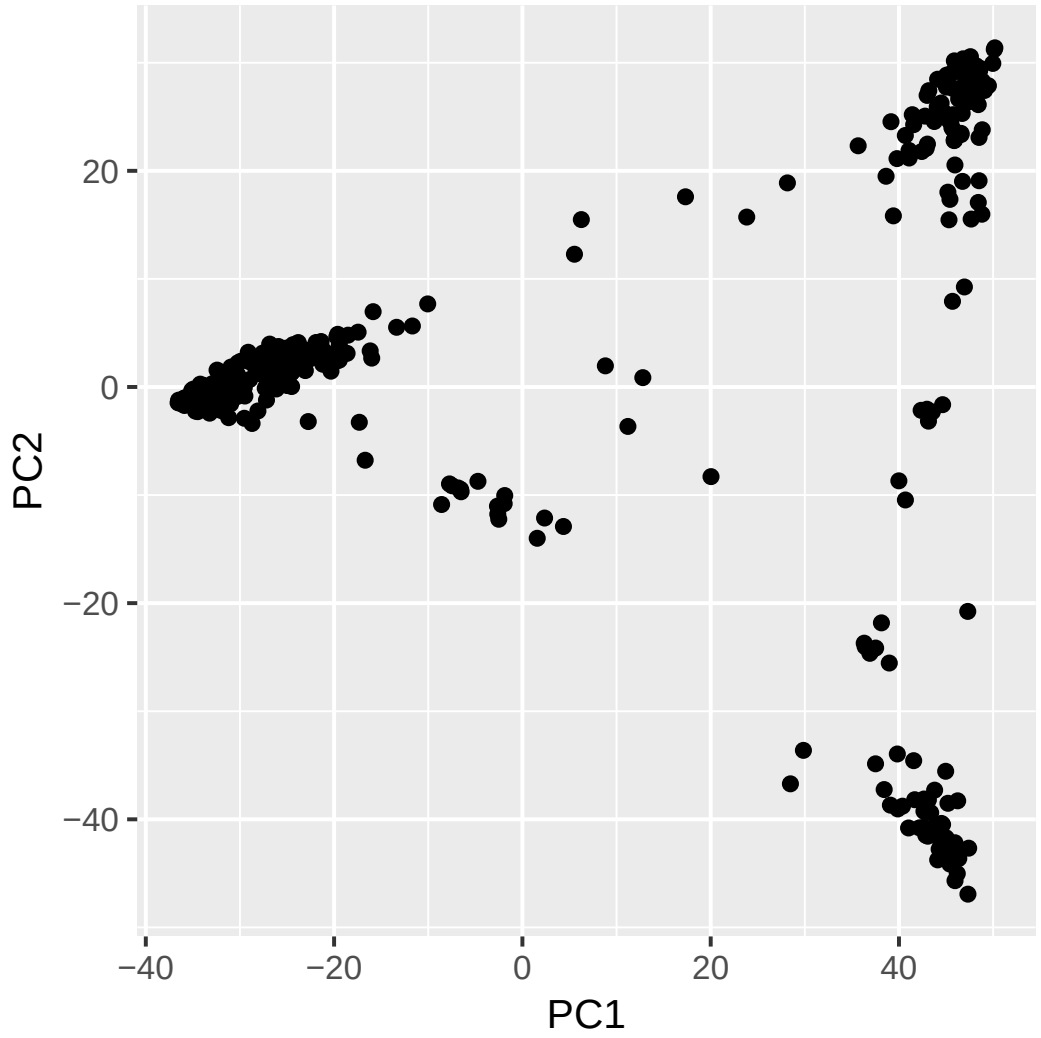
population structure

Before we go to GWAS we can check whether there exist some population structure in the data using PCA analysis.

```
> pca_rice=prcomp(newmat,scale.=T)
```

Then you look at the summary using *summary*(*pca_rice*) and if the first two principal comportments (PC) explains more that 20% variation (as a rule of thumb) we can say there exist strong population structure. In this example PC1 explains 0.3314 and PC2 around 0.09638. Thus indication of a strong population structure.
Let us visualize the population structure using *ggplot*2. For that we store the first two principal components into a data frame.

```
> library(ggplot2)
> dat_rice=data.frame(PC1=pca_rice$x[,1],PC2=pca_rice$x[,2])
> ggplot(dat_rice, aes(PC1, PC2),main ="Rice")+ geom_point()
```



From the plot it can seen that there exist strong population structure.

Kinship matrix

Kinship/relatedness coefficient which is a measure of genetic similarity between a pair of individuals, is commonly used to estimate heritability and breeding values. The kinship coefficients can be derived from pedigree or marker

information. In our case we are going to estimate the kinship matrix based on the marker information. There are many ways to calculate it but we are going to use the one implemented in $rrBLUP$ .This one is commonly used in GWAS. The package $rrBLUP$ provide function named $A.mat()$ for the estimation of kinship matrix and the markers should be coded as $-1, 0, 1$. Here 0 corresponds to the heterozygous. Let us calculate the kinship

```
> A=A.mat(newmat-1)
> dim(A)
```

```
[1] 413 413
```

When using rrBLUP, one of the most challenging thing is to format the data, which is readable to the package. The phenotypic data should contain the lines and the phenotypic information for at least one trait. Whereas for the genotypic information file the first column contains the marker name followed by the chromosome number, marker position and the line name for each plant.
Let us create the data file for rrbLUP.

```
> rrblup_geno=data.frame(Marker=map$marker,Chr=map$chr,Pos=map$pos,t(newmat-1))
> head(rrblup_geno)[,1:5]
```

```
             Marker Chr    Pos NSFTV_1 NSFTV_10
id1000001 id1000001   1  13147       1        0
id1000024 id1000024   1 152899       0       -1
id1000075 id1000075   1 212714       1       -1
id1000109 id1000109   1 327764      -1       -1
id1000196 id1000196   1 382993      -1       -1
id1000256 id1000256   1 481261      -1       -1
```

Note that we already uploaded the map file and the newmat is the marker coding based on the synbreed package. The markers should be coded as -1, 0 1. So we use $newmat - 1$ here.

GWAS

Let us perform the GWAS now. Kinship matrix is mainly used to correct the false positives and the population structure. Before the availability of genome wide SNPs researchers normally used PCA components to correct for the populations structure. However now the kinship matrix can correct for the population structure too with genome wide SNPs.
For more clarity first we do GWAS with out kinship and for that we create an identity matrix(ones in the diagonals). We have three traits in *pheno* table and let us select only the first trait.

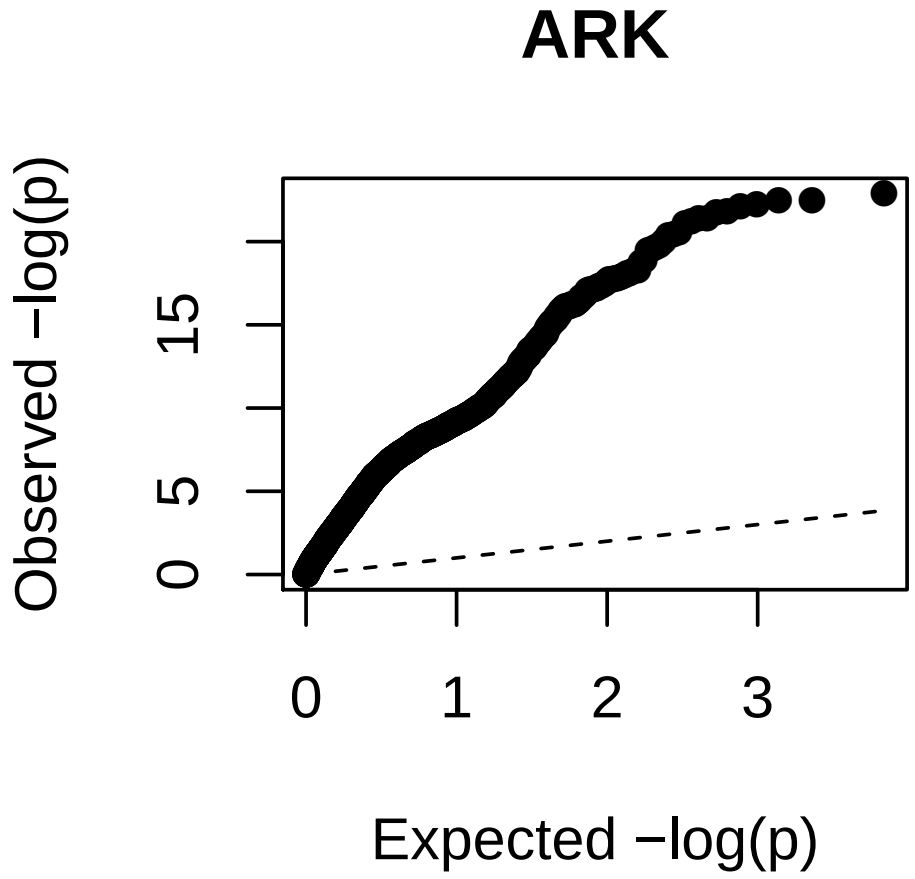
```
> I=diag(dim(pheno)[1])
> pheno_one=pheno[,1:2]
> head(pheno_one)
```

```
        Gid      ARK
1   NSFTV_1 75.08333
2  NSFTV_10 89.00000
3 NSFTV_100 84.11111
4 NSFTV_101 86.16667
5 NSFTV_102 92.22696
6 NSFTV_103 84.00000
```

Function $GWAS()$ function perform a genome-wide association analysis based on the mixed model.

```
> score=GWAS(pheno_one,rrblup_geno,K=I,plot=T)
```

```
[1] "GWAS for trait: ARK"
[1] "Variance components estimated. Testing markers."
```



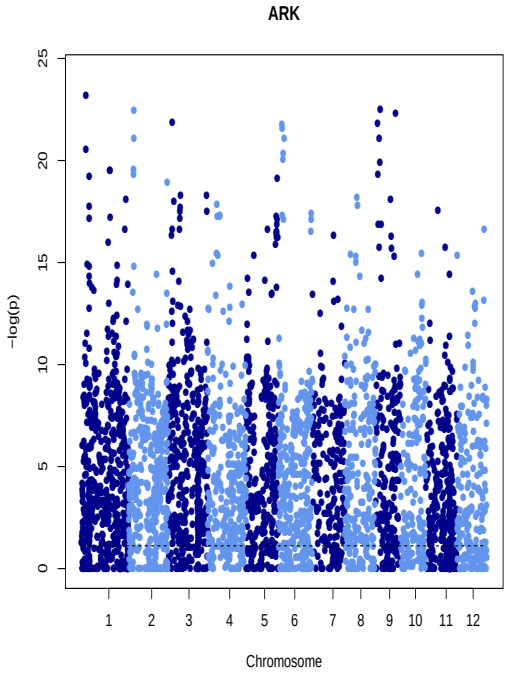
Here I is the identity matrix.
Here you can see two plots the first plot is called the Q-Q plot(quantile-quantile plot). Here X -axis corresponds to the expected p values (logarithm of values from a uniform distribution between 0 and 1) and Y -axis is the observed p-value. Here the straight line represent X=Y and any deviation from the X=Y line is a significant SNP. Form the Q-Q plot you can see almost all SNPs are significant.

Now let us correct for the population structure and false positives based on the kinship matrix.

```
> score=GWAS(pheno_one,rrblup_geno,K=A,plot=T)
```

```
[1] "GWAS for trait: ARK"
[1] "Variance components estimated. Testing markers."
```

Figure 1: Manhattan plot without kinship information



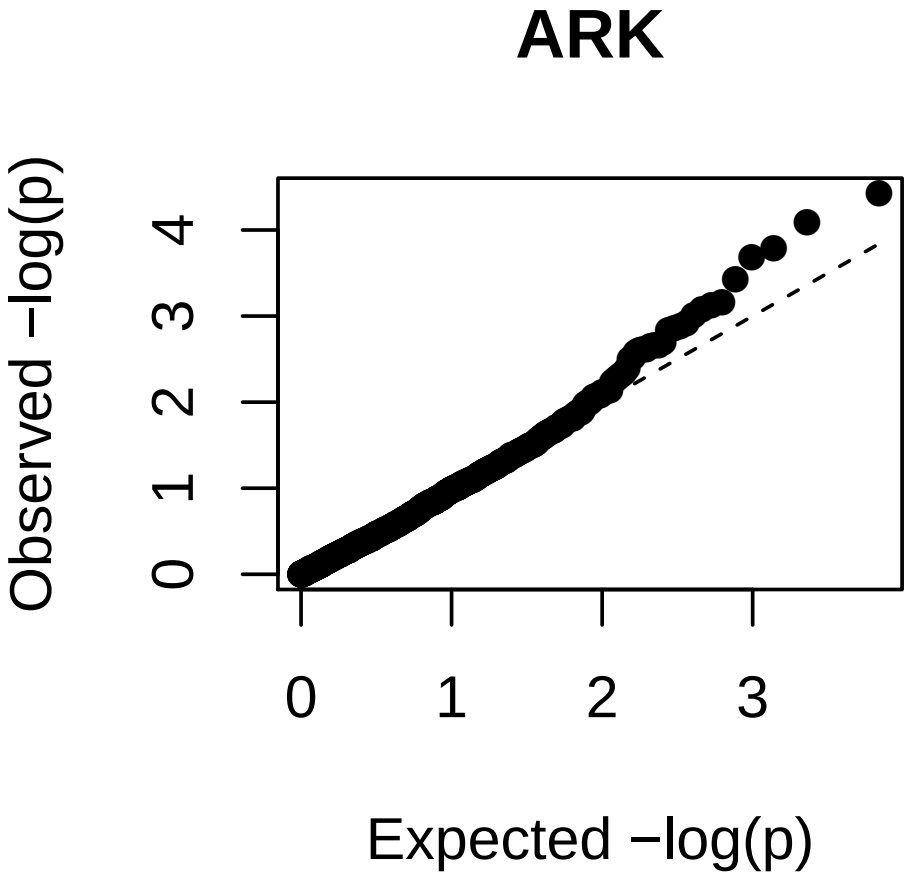
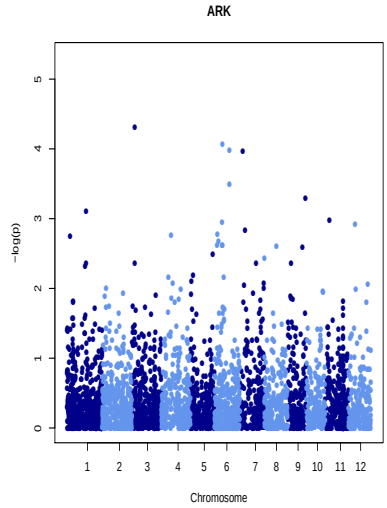


Figure 2: Manhattan plot with kinship information



If you want principal component in the analysis you can use the option $n.PC = n$, here n is the number of components to be included in the analysis.

The $GWAS$ function will return the marker name, chromosome, position and the marker scores for the traits. Then we need to order them according to the highest p-values.

```
> new_score=score[order(-score$ARK),]
> head(new_score,10)
```

```
             Marker Chr      Pos      ARK
id3001967 id3001967   3  3639159 4.425586
id6006146 id6006146   6  9683473 4.086642
id6010063 id6010063   6 17993684 3.784313
id6010122 id6010122   6 18119356 3.683478
id9007431 id9007431   9 21900226 3.426578
id1013422 id1013422   1 23366407 3.161702
id6005881 id6005881   6  9316441 3.130020
id6003300 id6003300   6  4612943 3.077372
id6002767 id6002767   6  3364148 3.013001
wd4001623 wd4001623   4 11650104 2.921831
```

Genomic breeding values (GEBVs)

In $rrBLUP$ you can calculate the genomic breeding values of each line using the function *kin.blup*. The estimated genomic breeding values can be used a as index for the selection.
Genomic breeding values can be calumniated either based on mixed models or whole genome regression model (WGR). Function *kin.blup*() use $GBLUP$ method which is based on mixed model for the estimation of genomic breeding values.

For the estimation of GEBV we need kinship matrix and we already calculated it (matrix A)

```
> ans=kin.blup(data=pheno,geno="Gid",pheno="ARK",K=A)
> ls(ans)

[1] "g"     "pred"  "resid" "Ve"    "Vg"
```

It is really important that you provide the *geno* and *pheno* arguments properly. Here the *geno* argument is column name of line names from the data file were the phenotypic information is stored whereas *pheno* argument is the name of the column containing the phenotypic information in the same file.
Here using option K we include realized additive relationship matrix in the prediction. You can also include user defined covariance matrices using the option K. If you have the pedigree information you can calculate the additive relationship matrix from the pedigree and can include that covariance matrix in the prediction. Here vector g is the GEBV and Vg is the genetic variance based on marker information and Ve is the residual variance.
Then you can calculate the SNP-heritability as follows:

```
> ans$Vg/(ans$Vg+ans$Ve)

[1] 0.6549697
```

The SNP-heritability of the trait is 0.66.
In order to calculate the correlation between GEBVs and phenotypic values.

```
> cor(ans$g,pheno_one$ARK)

[1] 0.9378751
```

Genomic selection

Marker assisted selection(MAS) has been revolutionized the breeding programs over two decades. First step in MAS is to find the statistically significant QTL(s) in a genome wide scan. In order to identify a QTL, we have to perform very stringent statistical tests and will fail to detect QTLs with small effect. When you select a QTL with big effect we may fail to select polygenes(because they have genes having small effect). MAS is effective for alleles with large effects. Current research studies showed MAS fails to improve polygenic trait and many important traits in plant breeding are polygenic. Genomic selection (GS) account th problem by including all the markers in the prediction model.

In GS the breeding values(BV) are calculated based on genome-wide dense DNA know as GEBV(genomic estimated breeding value). GS is a selection criteria and GEBV's say nothing about the underlying function of the gene. 'rrBLUP' is a R package, which has been developed for genomic prediction with mixed models. In rrBLUP genomic prediction can be done by estimating marker effect(RR-BLUP) or by estimating the line effect (G-BLUP)

With availability of cheap and abundant molecular markers genomic selection (GS) is becoming an efficient method for selection in both animal and plant breeding programs. Noways the phenotyping is very expensive compared to the genotyping and researchers are trying to predict the phenotypes based on the marker information. Genomic selection procedure is based on two datasets, the first one is called the training set and the second one is know as the validation set. In the training set we have the marker information as well as the phenotypic information. But in validation we have only the marker information and we try to predict the phenotypes. Bot validation set and the training set should be genotyped with the same set of markers.
Marker effects are estimated based on the training population,so both marker information and the phenotypic information should be available for the training population. However for the validation set the phenotypic values are

predicted based on the marker information and marker effects which are estimated from the training population. For GS first we need to get the GEBVs. Both G-BLP model and WGR model can be used for obtaining the GEBVs. We consider G-BLUP here.
In our rice data set we have 413 lines. As an example we are gong to consider 100 lines as the validation set out of 413 lines. Meaning that we do not have the phenotypic information for those 100 lines (but we have the marker information).
First we randomly select 100 lines as follows:

```
> set.seed(100)
> val=sample(1:413,100)
> val
```

```
  [1] 128 107 228  24 192 198 331 151 222  69 252 355 113 160 305 267  82 142
 [19] 396 272 211 279 393 293 164  67 299 341 212 412 188 402 133 363 264 337
 [37] 404 237 372  49 124 322 289 307 223 181 287 324  76 112 120  72  86  99
 [55] 213  91  45 397 359  75 389 411 338 376 156 125 159 155  85 239 395 401
 [73] 196 329 225 385 371 261 280  31 154 200 399 325  13 190 240 356  98 238
 [91] 390  68 115 144 290 373 165  40  10 243
```

Then we keep those lines phenotypic informations as missing "NA"

```
> pheno_CV=pheno_one
> pheno_CV[val,2]=NA
> head(pheno_CV)
```

```
        Gid      ARK
1   NSFTV_1 75.08333
2  NSFTV_10 89.00000
3 NSFTV_100 84.11111
4 NSFTV_101 86.16667
5 NSFTV_102 92.22696
6 NSFTV_103 84.00000
```

Then we estimate the GEBVs based on the *kin.blup*() function as follows:

```
> ans=kin.blup(data=pheno_CV,geno="Gid",pheno="ARK",K=A)
```

Let us calculate the correlation between the predicted phenotypes and the original phenotypes

```
> cor(ans$g[val],pheno_one$ARK[val])
```

```
[1] 0.6295832
```

Here the correlation is depended on many factors such as the size of the training set, heritability of the trait, the trait itself (effects of genes), LD in the data, number of markers. High correlation is an indication that we can perform efficient genomic selection on the selected trait with the given data.

GEBVs based on WGR model

First create the training set which is the 313 lines not in the validation set (100 lines) and we can select those lines based on the $setdiff()$ function. And define M markers as newmat-1. Then select the training set from the phenotype file and marker information as follows:

```
> M=newmat-1
> train=setdiff(1:413,val) # those lines no in the validation sel
> pheno_train=pheno_one[train,] # define the training phenotypes
> M_train=newmat[train,] # define the training marker set
```

Then create the validation population with the marker information and phenotype as follows

```
> M_val=M[val,]
> pheno_val=pheno_one[val,]
```

Calculate the marker effect based on the training population using the function *mixed.solve*() in rrBLUP

```
> res_train=mixed.solve(pheno_train$ARK,Z=M_train)
```

Predict the phenotypes of the validation set based on the estimated marker effect and the genotypes from the validation set

```
> fi=M_val%*%as.matrix(res_train$u)
> head(fi,5)
```

```
              [,1]
NSFTV_216 -5.972802
NSFTV_196 -1.590334
NSFTV_309 -0.625844
NSFTV_12   1.850624
NSFTV_276 -6.814177
```

Calculate the correlation with the original phenotypes.

```
> cor(pheno_val$ARK,fi)
```

```
          [,1]
[1,] 0.6295828
```

Here you can see the correlation based on both models are same.

GS in multi environment trials (MET)

In plant breeding, Multi-environment trials (MET) are commonly used to assess the intensity of genotype-by-environment (G×E) interactions in order to select high-performing lines across environments. In plant breeding, multi-environment trials (MET) are used to assess the genotype-by-environment (G×E) interactions, which are crucial for selecting stable and high-performing lines across environments. It is a common practice to perform genomic prediction estimation on a trait from a single environment at a time. But you can use multi-variate linear mixed models (MLMM) and analyze them together. Let us consider as an example our rice data. In rice phenotype data we have phenotypic informations from three locations. In the flowing section I will show how we can analyze the MET data using a MLMM. Until now we used rrBLUP. However, rrbLUP does not support MLMM analysis and so we need to consider the R-package "*sommer*".

```
> library(sommer)
> head(pheno)
```

```
        Gid      ARK      Fad Abr
1   NSFTV_1 75.08333 64.00000  81
2  NSFTV_10 89.00000 55.00000  74
3 NSFTV_100 84.11111 78.00000 122
4 NSFTV_101 86.16667 79.00000  88
5 NSFTV_102 92.22696 73.58621 165
6 NSFTV_103 84.00000 78.00000 108
```

Here ARK, Fad and Abr are three locations and the trait is the flowering time in days. You can use the function $mmer2$ to fit the MLMM as follows

```
> mix2$var.comp

$`g(Gid)`
          ARK      Fad       Abr
ARK  80.69879 17.52273 138.94836
Fad  17.52273 11.51187  47.27955
Abr 138.94836 47.27955 765.86654


$units
          ARK       Fad       Abr
ARK 42.180659  9.650347  81.50758
Fad  9.650347 36.462604  22.75554
Abr 81.507580 22.755537 573.67304
```

Here function $cbind()$ will combine the three traits to a vector. Random term is used to specify the random factors and the genotypes are random in our case. $rcov$ option is used to specify the error term and G option is used to provide the covariance matrix. In our case we need to include the genomic relationship matrix and included as $G = list(Gid = A)$. And the table "var.comp" returns the genetic and residual/error variance covariance components.
Genomic selection in MET.
You can also perform GS with MLMM and in this case you need to set the phenotypic informations as missing in all three environments.

```
> pheno_CV=pheno
> pheno_CV[val,2:4]=NA
> mix2<- mmer2(cbind(ARK,Fad,Abr) ~1, random = ~ us(trait):g(Gid), rcov= ~ us(trait):units,
+ G=list(Gid=A),data =pheno_CV,draw=FALSE, silent=T)
```

Then calculate the correlations for each location/environment , between EBVs and missing phenotypes.

```
> cor(mix2$u.hat[[1]][,1][val],pheno$ARK[val])

[1] 0.6168709

> cor(mix2$u.hat[[1]][,2][val],pheno$Fad[val])

[1] 0.2319355

> cor(mix2$u.hat[[1]][,3][val],pheno$Abr[val])

[1] 0.504856
```