Get started with R PROGRAMMING

Import, explore, manipulate, summarize and plot the data

Rajendra Choure

2022-03-18

Contents

1 Introduction **2**

2 Installtion of software **3**
2.1 R programming, . . . 3
2.2 R studio . . . 3
2.3 Getting started . . . 3
2.4 Required packages . . . 3
2.5 R coding rules. . . . 6

3 Start learning R **7**
3.1 Operators in R . . . 7
3.2 R functions . . . 8
3.3 Introdcution to packages . . . 9

4 Data types **10**
4.1 numeric data : double and integer . . . 10
4.2 character data . . . 10
4.3 Logical data . . . 11

5 vectors **11**
5.1 data coercion . . . 12
5.2 Creating vectors . . . 12
5.3 naming vlaues in the vectors . . . 17
5.4 addressing values in the vectors . . . 17
5.5 Vectorised operatoions . . . 18

6 metrices **19**
6.1 creating matrix . . . 19
6.2 Naming matrix columns and rows . . . 20
6.3 Addresing matrix . . . 21
6.4 functions on matrix . . . 24
6.5 mathemtical operations on matrix . . . 25

7 data frames **26**
7.1 adding columns to a dataframe . . . 27
7.2 two data frames with same columns can be joined together using rbind . . . 27
7.3 adding columnd using '$' operator . . . 27
7.4 rownames and colnames of data frame . . . 28
7.5 Fucntions to explore data frame . . . 28

7.6 subsetting dataframe . . . 30

8 lists **32**

9 Exporting Data to a file in R **33**
9.1 Working directory in R . . . 33
9.2 Saving as .txt file . . . 33
9.3 Saving as .csv file . . . 34
9.4 Saving data as xls file . . . 34
9.5 Save data as .dta file . . . 35

10 Importing data **35**
10.1 Import Text data . . . 35
10.2 Import data from .csv file . . . 35
10.3 Import data form .xls or xlsx file . . . 35
10.4 import data form .dta file . . . 36

11 Manipulating data: **36**
11.1 taking care of missing data . . . 36
11.2 Piping of code . . . 38
11.3 Intordcution to tidyverse . . . 39
11.4 fucntions from `dplyr` package . . . 39
11.5 Function of "tidyr' package . . . 42
11.6 pivot . . . 42

12 Plot data **43**
12.1 the first plot . . . 43
12.2 Lets plot . . . 45
12.3 Density plot . . . 49
12.4 box plot . . . 51
12.5 violin plot . . . 52
12.6 Scatter plot . . . 53
12.7 bar plot . . . 55
12.8 Build the barplot . . . 55

13 Linear regression and prediction **58**
13.1 predict . . . 59
13.2 Diagnostic plots for validating model . . . 59

14 Conclusion **62**

1 Introduction

R is a programming language and software suit developed by statisticians. Its a *de-facto* programming language for data analysis,visualization and data modelling in today's data driven era.

R is a functional language and easy to learn language for non programmers. its intuitive code and line by line code execution makes comes very handy for early learners who don't have sufficient expertise to debugging.

R libraries provide almost all functions necessary for any data analysis task, its online community provide solution to almost any problem in data analysis.

R is the best programming environment for data visualization.

In this workshop we will experience power of R as a language and software suit for data analysis tasks.

2 Installtion of software

All the software required to use R, are free to download and use. It can be installed on the computer available when you need data analysis.

2.1 R programming,

R is free to download and use. YOu can download R for windows 64 bit OS by clicking on following link.

https://cran.r-project.org/bin/windows/base/R-4.1.3-win.exe

After download double click the installation file(usually saved in your download folder) and give all permissions by clicking accept or OK on appearing dialogue boxes. R programming will be installed within few minutes.

2.2 R studio

R studio is **I**ntegrated **D**evelopment **E**environment which facilitate coding in R. The most useful feature is **Comman completion**, which completes the code when you type first three letters of the code. It provides scripting window, file explorer, package library and plot visualization pane.

Click on the following link to download R studio for 64 bit Windows OS. You can Google to get link to install R studio on other OS(Mail me if you don't get the link form Google).

https://www.rstudio.com/products/rstudio/download/#download

After download double click the installation file(usually saved in your download folder) and give all permissions by clicking accept or ok on appearing dialogue boxes. R programming will be installed within few minutes.

2.3 Getting started

After completion of installation of R programming and Rstudio,

1. just type in **Rstudio** in the search box of windows(Bottom-left of your windows desktop, see fig.1),

1. a list with "Rstudio" at top will pop up. Click the R stduio on the list and Rstudio will get opened.

1. Rstudio IDE will open

In this GUI, the left pane is "console", here R shows the code and its output. Top right pane has many tabs, where all R objects you create will be displayed. The lower pane has also many tabs, The plot tab , package tab, explorer tabs are important for us.

1. Starting the new script file

In the top left you have file menu. Click on file , then click on new file and then R script.

A script pane will open by pushing the console down. You write the r code in this script pane. You can right code in console also, but it will not be saved. Script can be saved as file, using "save" or "save as" from the file menu. R saves script with ".R" extension.

2.4 Required packages

You need to install following package and for this you must be connected to internet or shall have packages downloaded to a storage device.

Packages and their required packages("Dependencies") will get installed.(single or multiple packages can be installed using 'install.packages' command, with one command. enclose each package name between inverted quotes and separate package names by comma.)

For package installation online, type in following code without any spelling mistakes or uppercase-lowercase mix up and click run button.

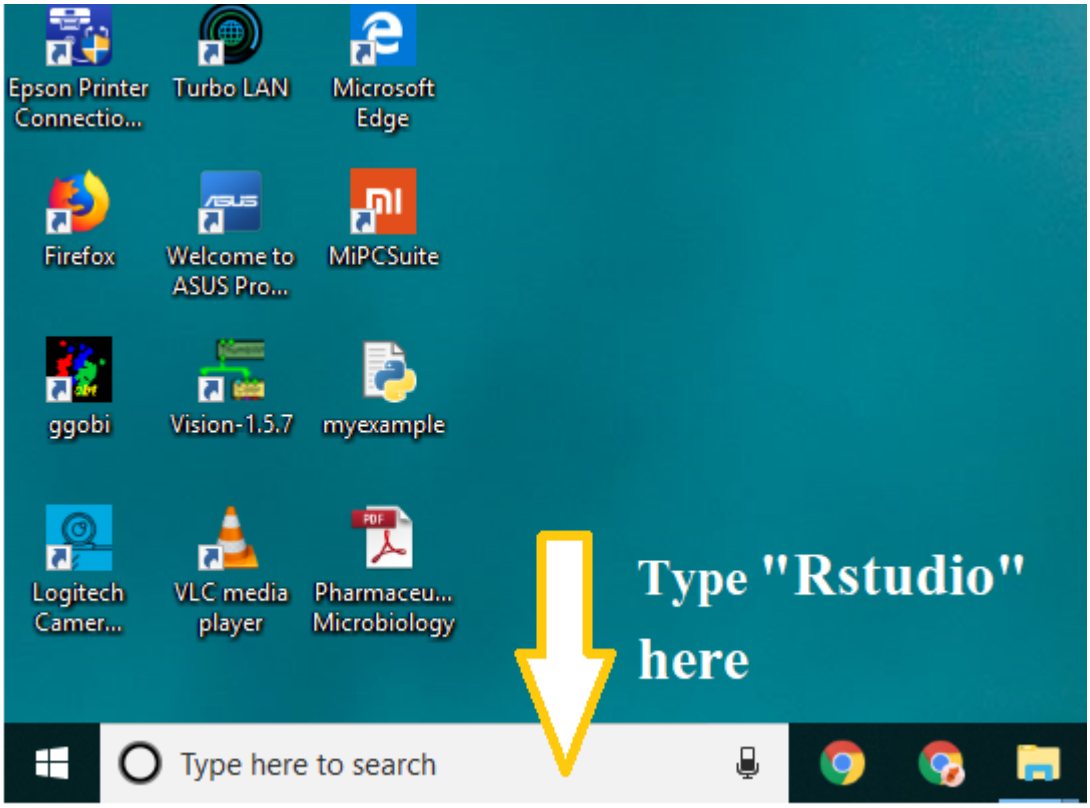


Figure 1: Windows start menu

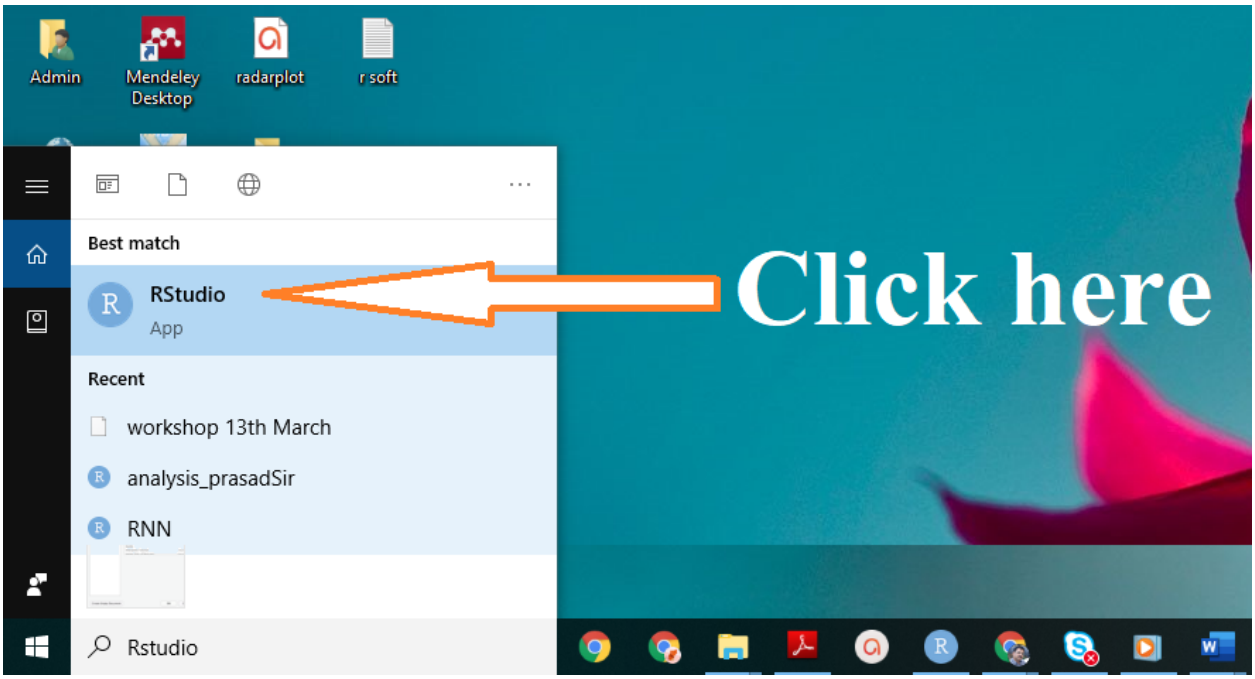


Figure 2: Pop up list

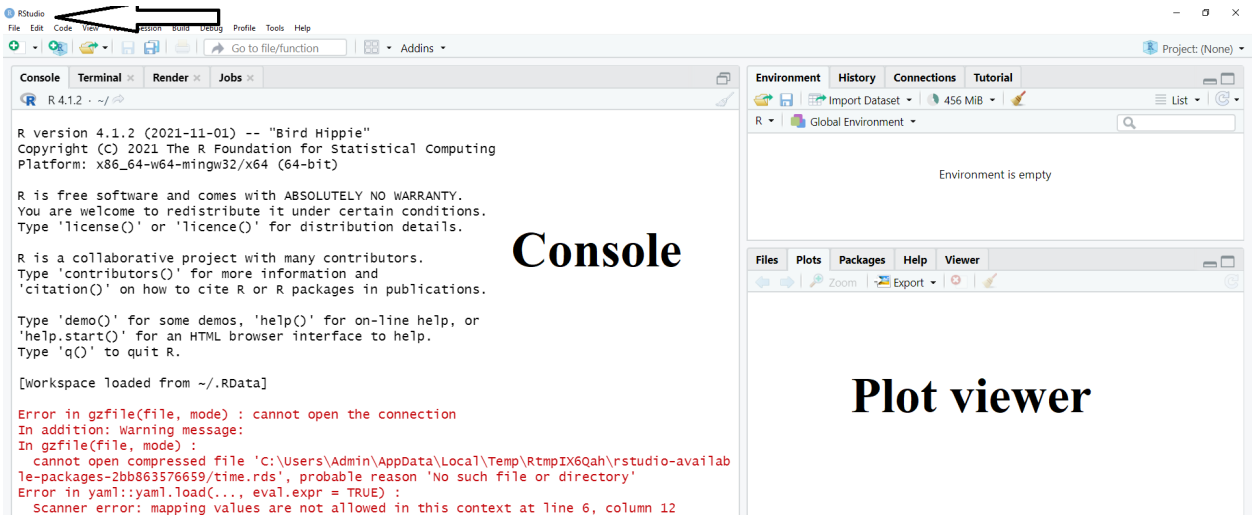


Figure 3: R studio IDE

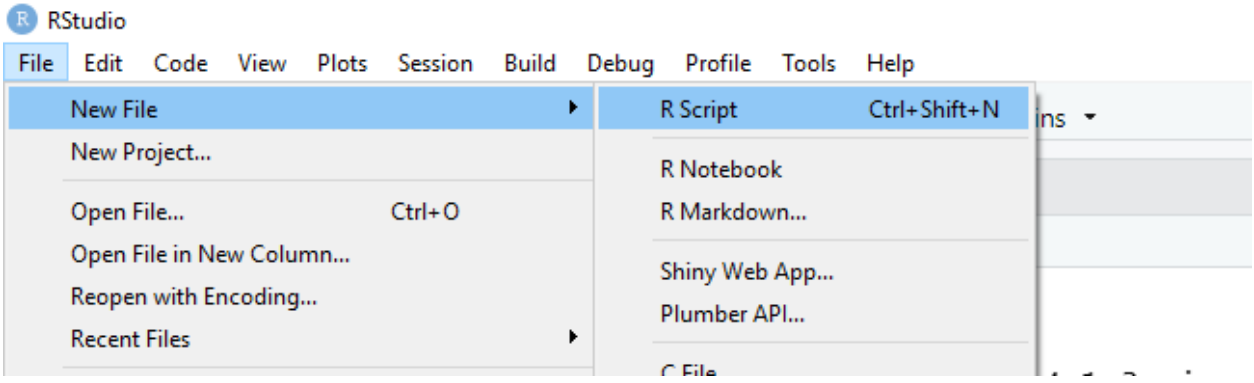


Figure 4: Opening new script

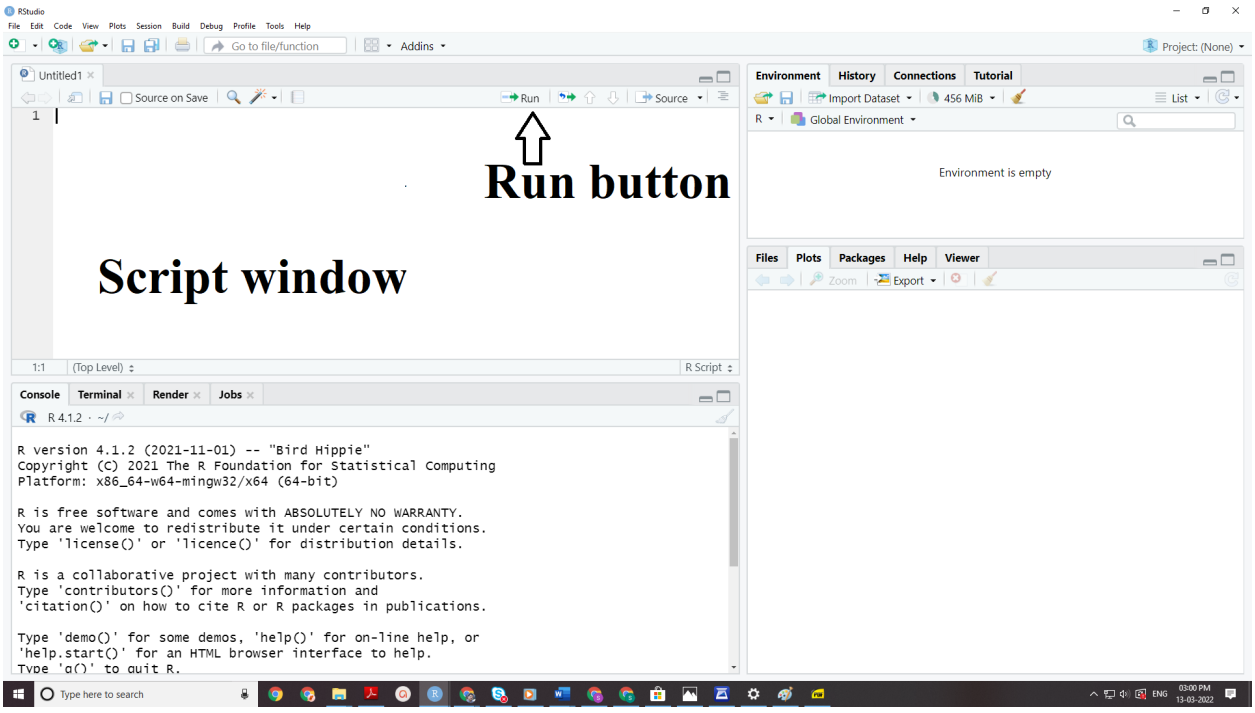


Figure 5: R studio IDE

```
install.packages("tidyverse","xlsx","readstata13","readxl","mice","GGally","ggsci")
```

Alternatively , you can install packages by clicking *Install Packages* from **Tools** menu. When you click that "Install Packages", a dialogue box will open , and type in package name or names separated by commas in the search box. Click install and packages and dependencies will get installed.

```
library(tidyverse)
```

2.5 R coding rules.

1. R is a case sensitive language. If you type " Rammesh" in your code and then ask R abot "ramesh" or "raMesh" , r will say that "object"ramesh" not found." and your code will not run.
2. To run a code, click on the line and then click run or press `ctrl+enter` on keyboard.
3. All the data types, functions , plots etc in R are treated as objects. When you save an object with a name, then it can be called when required by using its name. If you don't name an object it cannot be reused in the code for obvious reason- you cant call it as it doesn't have name.
4. Name of objects cannot start with numbers or special character(except '.').
5. Lines starting with '#' are ignored by R, these are comments used to explain the code. Use comments as much as possible to make the code readable for you or other readers.
6. To get help form R type the question mark followed by function or package name for which you want help and press enter. In help pane of rstudio you will see help. Or alternatively you can Google the question, R community will have answer for it.
7. If you get any error during your work in R studio, just copy the error and post it on google , you will get satisfactory solution most of the times.

3 Start learning R

3.1 Operators in R

Just type in the code given in grey boxes below line by line in your script and run that. The lines starting with '#' are output of the code. You will see output in console.

```
3 + 4
```

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

```
3 - 4
```

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

```
3 * 4
```

```
## [1] 12
```

```
12/7 # returns division
```

```
## [1] 1.714286
```

```
12%%7 # returns remainder of division
```

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

```
12%/% 7 # returns quotient
```

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

3.1.1 assignment operator

'<-' is assignment operator. we are assigning value or data or data structure or any object type to object named towards point of arrow of "<-" operator.

Once saved as object with name, the name can be typed and run to print the value it stores.

```
a <- 3 # 3 is assigned to a

a
```

```
## [1] 3
```

```
b <- 4 # 4 is assigned to a

b
```

```
## [1] 4
```

```
Ganesh <-  3+4

Ganesh
```

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

```
is.male <- TRUE

is.male
```

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

3.2 R functions

R function is an names R object which acts as a verb to do a specific task. function structure is function name followed by arguments in parenthesis. (see fig.)

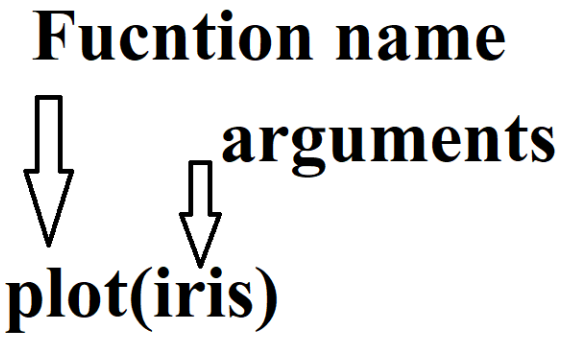
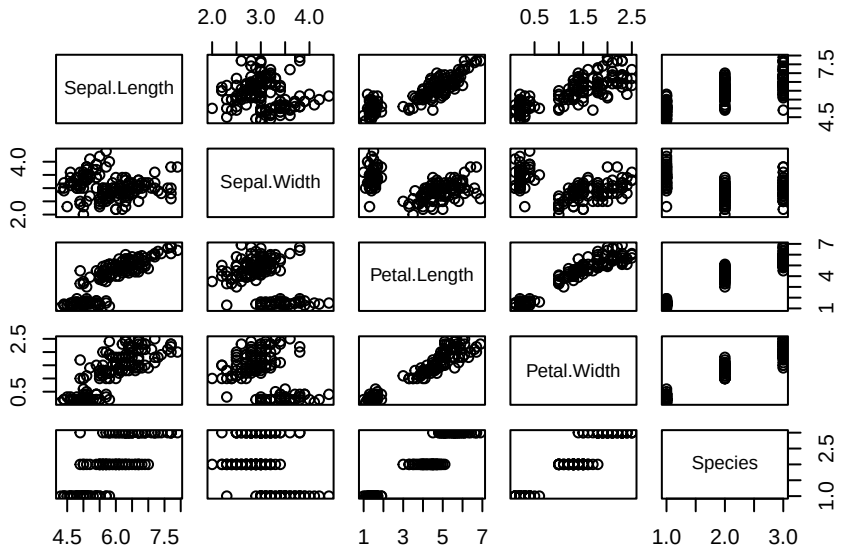


Figure 6: structure of R fucntion

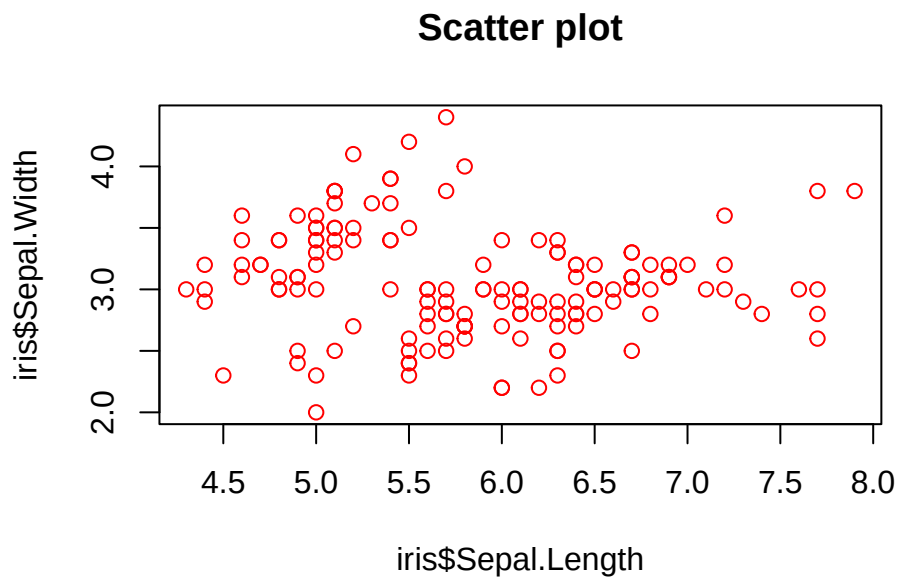
for example plot() function plots according to the argument enclosed in the parenthesis. In the example below we have asked R , to plot data set iris (iris data set is bundled with R installation)

```
plot(iris)
```



Multiple arguments of a function are separated by commas.

```
plot(x=iris$Sepal.Length,y=iris$Sepal.Width, col="red", main="Scatter plot")
```



In the example above, arguments are named and separated by commas. If you write arguments in the sequence as defined by the function code, you don't have to name those.

Some functions have default arguments. You will learn that when you achive some expertise in R.

3.2.1 User defined functions

This is advanced aspect of R. You can define your own functions to automate the tasks which are required frequently. here is an example

```
# define the function :
SimpleInterest <- function(p,r,t){
  int= p*r*t/100
  return(int)
}

SimpleInterest(p=12000,r=3,t=4)
```

```
## [1] 1440
```

to learn more write down as many functions as possible. This ability of writing function will improve trhoughput of your work.

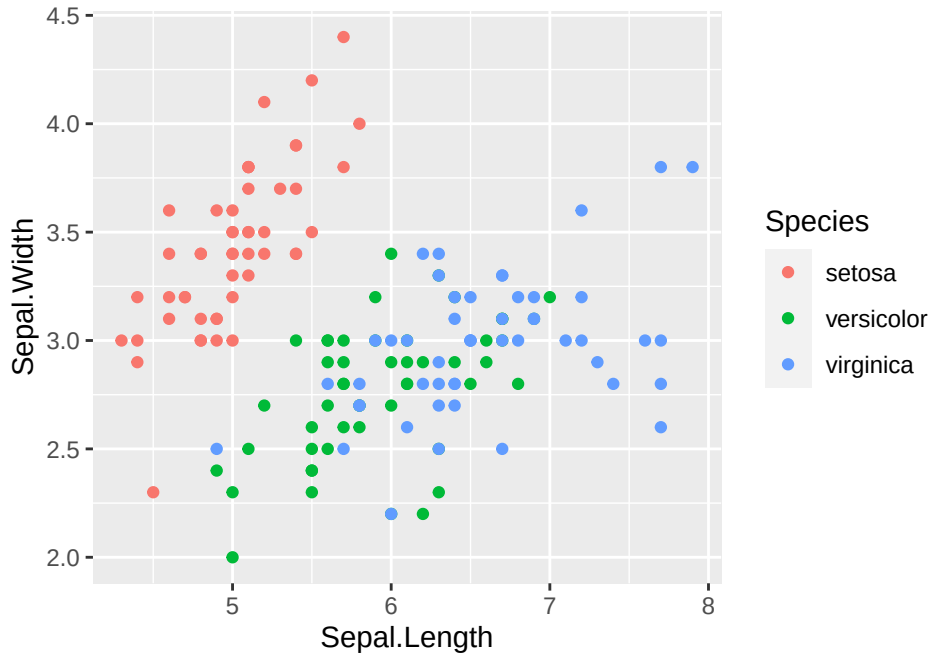
3.3 Introdcution to packages

Packages are libraries fo functions required for specific tasks. for example `ggplot` package contains function to plot nice plots based on "philosophy"grammer of Graphics". R community has developed and made available thousands opf packages to do almost any task required for analysis of any data. In order to use a package, you need to install it and then call it as library.

for example, we will call library `ggplot2` and plot a nice plot. (We have installed `ggplot2` as it is member of `tidyverse` universe which we installed above.)

```
library(ggplot2)
```

```
ggplot(iris,mapping=aes(x=Sepal.Length,y=Sepal.Width,color=Species))+
  geom_point()
```



Don't bother here about the above code. Just copy paste it in your script window and run that.

4 Data types

R has 5 data types. of these first three are routinely used and the data type `complex` and `raw` are rarely required.

4.1 numeric data : double and integer

```
1. double : numeric data with decimal values

2. integer: numeric data with no decimal fractions. To input integers, the number must be followed by `
```

`typeof()` function is used to know data type of the object.

```
a <-  25

typeof(a)
```

```
## [1] "double"
```

```
b <-  23L

typeof(b)
```

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

4.2 character data

these are character values and are enclosed in inverted quotes to separate those from R object names.

```
a <-  "Ramesh"
typeof(a)
```

```
## [1] "character"
```

```
b <-  " R programming language"

typeof(b)
```

```
## [1] "character"
```

4.3 Logical data

This data can have any of the two values `TRUE` or `FALSE`.

```
is.male <-  c(TRUE,FALSE,FALSE, TRUE)
is.male
```

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

```
typeof(is.male)
```

```
## [1] "logical"
```

```
 d <-  3>4

 d
```

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

here the c() function is used to combine many values together. In the arguments those values are separated by commas.

```
e <-  20:30 > 25
e
```

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

```
typeof(e)
```

```
## [1] "logical"
```

in above example : operator is used to specify series of values separated by 1. This is shortcut to generate large numeric series separated by 1.

5 vectors

Data structures

R has many data structures, but most common are vectors, metrices, data frames and lists.

This data types hold n number of data values, but all the values must of of same data type.

the type of vector is the data type of values it stores.

vector is one dimensional data structure, it has length only.

```
r  <-  1:100
typeof(r)
```

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

```
length(r)
```

```
## [1] 100
```

```
grade <- c( "I","II","III","IV")
typeof(grade)
```

```
## [1] "character"
```

```
length(grade)
```

```
## [1] 4
```

```
gramPositive <- c(TRUE,TRUE,TRUE,FALSE,FALSE)
typeof(gramPositive)
```

```
## [1] "logical"
```

```
length(gramPositive)
```

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

5.1 data coercion

if you mix data types in a vector all the data will be forcible converted to simplest of the data types in that vector. This is called as coercion. R just give warning in this regard, but will not stop the code from running.

Data coercion can be reason for serious errors during data analysis.

The pattern of coercion is

logical -> numeric -> character

```
male <- c(TRUE, "TRUE", 1, 0)

male
```

```
## [1] "TRUE" "TRUE" "1"    "0"
```

```
typeof(male)
```

```
## [1] "character"
```

here logical and numeric data , all got converted to character, as its the simplest of the data types.

5.2 Creating vectors

1. using c() function

c() function combine many values which are separated by commas to create a vector.

```
age <- c( 25, 27,31,41)

age
```

```
## [1] 25 27 31 41
```

```
friends <- c( "Ganesh", "Ramesh", "Anita","Rose")
friends
```

```
## [1] "Ganesh" "Ramesh" "Anita"  "Rose"
```

```
is.male <- c(TRUE, TRUE, FALSE, FALSE)
is.male
```

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

2. using : operator

':' operator , as seen above gives a series of numbers bound by the numbers specified, and separated by 1.

```
mileage <-  25:37
mileage
```

```
##  [1] 25 26 27 28 29 30 31 32 33 34 35 36 37
```

```
height <-  57:65
height
```

```
## [1] 57 58 59 60 61 62 63 64 65
```

```
cost <-  54:59
cost
```

```
## [1] 54 55 56 57 58 59
```

3. using seq() function

seq function is very elegant function to get a sequence , specified by rules in the arguments. Study following examples

Using 'by' argument

```
seq1 <- seq(25, 50, by =5)
seq1
```

```
## [1] 25 30 35 40 45 50
```

Using length.out argument

```
seq2 <- seq(25, 50, length.out =5)
seq2
```

```
## [1] 25.00 31.25 37.50 43.75 50.00
```

4. using sequence() function Generates a series from 1 to the number in the argument.

```
seq3 <- sequence(20)
seq3
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
seq5 <- sequence(9)
seq5
```

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

5. Using `rep()` function

rep function repeats the vlaues given in argument to specifed number of times.

```
rep1 <-  rep(5, times=3)
rep1
```

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

rep() function cna repeat vlues of a vector alos

```
treatment <-  rep(c("t1","t2"), by =5)
treatment
```

```
## [1] "t1" "t2"
```

Each argument makes the values of vector repeated each one by one to the given number.

```
treatment2 <-  rep(c("t1","t2"), each =5)
treatment2
```

```
##  [1] "t1" "t1" "t1" "t1" "t1" "t2" "t2" "t2" "t2" "t2"
```

4. vectors of certain statistical distributions

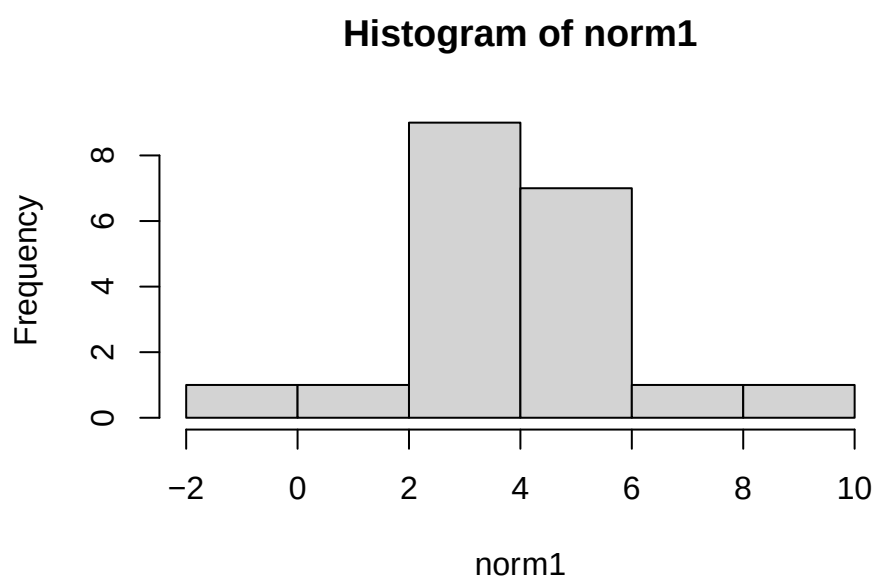
vectors having randomly selected values belonging to specified statistical distributions can also be created in r.

a. rnorm(): normally distributed randomly selected values

```
norm1 <-  rnorm(n=20,mean=3, sd=2)
norm1
```

```
##  [1]  2.2020045  4.2899879  6.1685938  5.4743781  5.7890009  2.0842460
##  [7]  3.1856158  2.6751717  8.7999009  5.0603311  4.8972189  2.8816799
## [13]  4.2839720  3.8237565  2.7787615 -0.0732293  2.6904897  0.5100668
## [19]  2.8448136  5.4971746
```

```
hist(norm1) # plots histogram
```

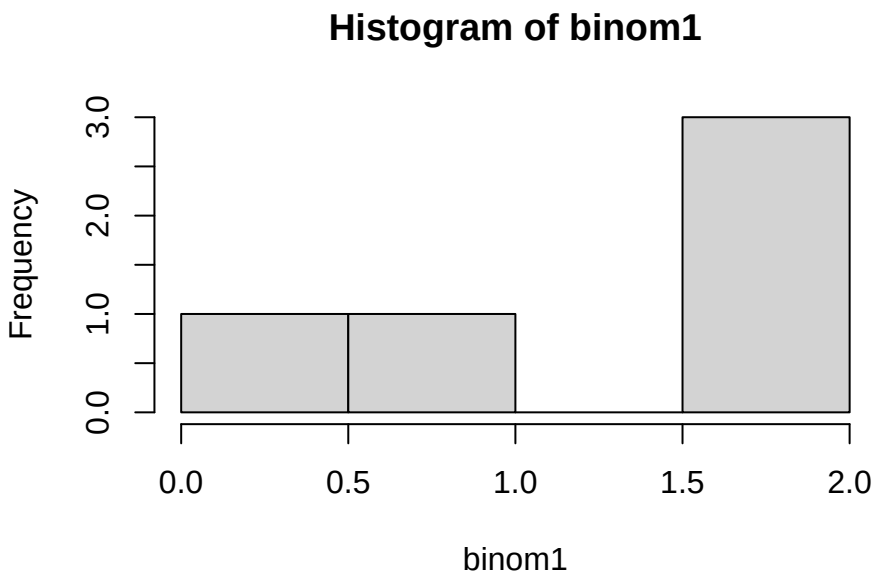


b. rbinom(): binomial distributed randomly selected values

```
binom1 <- rbinom(n=5,size=3,prob=0.5)
binom1
```

```
## [1] 0 2 1 2 2
```

```
hist(binom1)
```

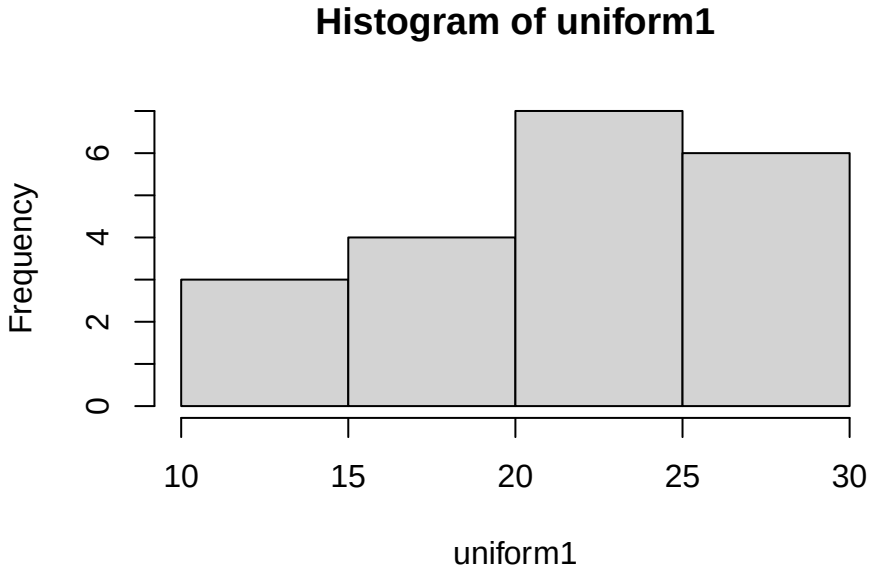


c. runif() : Uniformly distributed randomly selected values

```
uniform1 <- runif(20,min=10,max=30)
uniform1
```

```
##  [1] 15.30734 27.52731 19.38936 25.14575 21.26984 21.44775 11.56250 24.08841
##  [9] 23.84930 20.09501 20.47103 14.09951 22.99437 28.53405 29.44888 26.98038
## [17] 19.22337 18.74877 12.24275 27.26601
```

```
hist(uniform1)
```

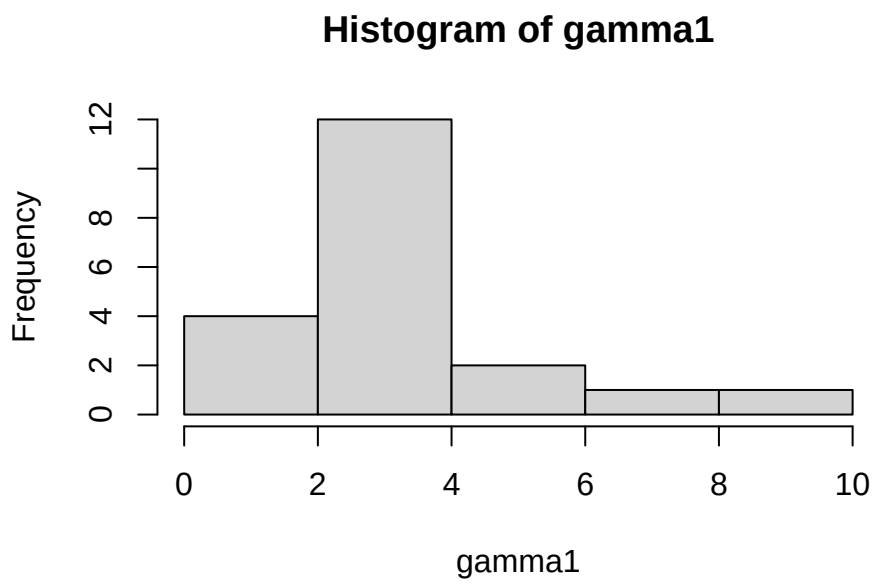


d. rgamma(): gamma distributed randomly selected values

```
gamma1 <- rgamma(20,shape=3)
gamma1
```

```
##  [1] 2.507303 1.763174 3.386967 2.144730 3.168549 2.635703 5.335902 3.943452
##  [9] 2.252227 5.363994 3.339420 0.544242 2.431380 9.653093 3.294173 6.822966
## [17] 2.364090 1.665067 0.865213 2.917335
```

```
hist(gamma1)
```

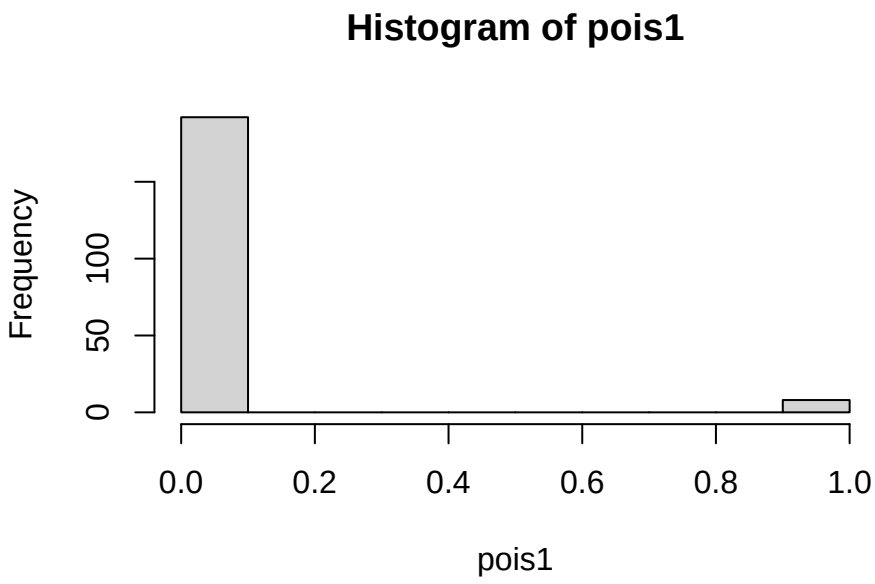


e. rpois : Poisson destributed randomly selected values

```
pois1 <- rpois(200,0.03)
pois1
```

```
##   [1] 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
## [149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
hist(pois1)
```



There is much to these distributions, which you will learn as you progress in R. Note that these functions are very useful addition to your skills.

5.3 naming vlaues in the vectors

Values in the vectors can be named. for exampole if we have age values of 5 individuals, we can name each age with the names of the person. names() function s used for naming values in a vector.

```
age <- c(25,35,34,32,21)
persons <- c("Ganesh","Ramesh","Suvarna","Ummed","Rose")

names(age) <- persons

age
```

```
##  Ganesh  Ramesh Suvarna   Ummed    Rose
##      25      35      34      32      21
```

Names can be used to select values of the vector. Please note the square bracket used in the code below to specifiy name of the values in the vector.

```
age["Ramesh"]
```

```
## Ramesh
##     35
```

```
age[c("Ramesh","Rose")]
```

```
## Ramesh   Rose
##     35     21
```

We will learn more about adressing values in the vector in following section.

5.4 addressing values in the vectors

1. using indices

values in the vector can be selected using their names if available, as we saw in previous section. Index is positional rank of the value in the vector. index or vector of indices can be used to select specific value or values of a vector. please note that indices are enclosed in square brackets after name of the object.

```
age[2]
```

```
## Ramesh
##     35
```

```
age[c(2,5)]
```

```
## Ramesh   Rose
##     35     21
```

Using logical condition

```
age[age>33]
```

```
##  Ramesh Suvarna
##      35      34
```

Using logical vector

```
is.male <- c(TRUE, TRUE,FALSE,TRUE, FALSE)

ageofMales <- age[is.male]
ageofMales
```

```
## Ganesh Ramesh  Ummed
##     25     35     32
```

5.5 Vectorised operatoions

In R , you can apply a function to entire vector. This is very handy while dealing with variables in data.

Here we will calculate body mass index of members we introduced above.

we will apply following function

$$BMI = \frac{weight}{hight^2}$$

```
height <- c(1.35,1.56,1.76,1.83,1.52)
weight <- c(65,67,54,92,67)
persons <- c("Ganesh","Ramesh","Suvarna","Ummed","Rose")

BMI <- weight/height^2
BMI
```

```
## [1] 35.66529 27.53123 17.43285 27.47171 28.99931
```

```
obese <- BMI >=25
obese
```

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

```
is.obese <- persons[obese]

is.obese
```

```
## [1] "Ganesh" "Ramesh" "Ummed"  "Rose"
```

```
not.obese <- persons[!obese]

not.obese
```

```
## [1] "Suvarna"
```

As you have noted, the operations took place on each member of the vector.

lets use this to make a multiplication table and square of that table.

```
seq1 <- seq(0,100,by=10)
seq1
```

```
##  [1]   0  10  20  30  40  50  60  70  80  90 100
```

```
seq1.10 <- 10*seq1
seq1.10
```

```
##  [1]    0  100  200  300  400  500  600  700  800  900 1000
```

```
sqr.seq1 <- seq1.10^2
sqr.seq1
```

```
##  [1]       0   10000   40000   90000  160000  250000  360000  490000  640000
## [10]  810000 1000000
```

6 metrices

Just like vectors matrix can hold a single data type but its a two dimensional data structure- has rows and columns. Like vectors, if we mix data types in matrix, those will be coerced to simplest data type.

6.1 creating matrix

Matrix can be created by arranging vector in rows and columns using matrix function.

```
mat1 <- matrix( 1:20, nrow=5, byrow=TRUE)
mat1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
```

Matrix can be filled columnwise by specifying `byrow=FALSE`.

```
mat2 <- matrix(1:20,nrow=5,byrow=FALSE)
mat2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

6.1.1 Matrix using rowbind or colbind

Vectors of equal length and holding same data types can be combined together in a matrix using rowbind or colbind function.

```
Jan <- c(25,27,28,23)
Feb <- c(31,32,31,33)
Mar <- c(32,32,34,34)
Apr <- c(33,33,34,35)
May <- c(36,37,37,36)

temp <- rbind(Jan,Feb,Mar,Apr,May)
temp
```

```
##     [,1] [,2] [,3] [,4]
## Jan   25   27   28   23
## Feb   31   32   31   33
## Mar   32   32   34   34
## Apr   33   33   34   35
## May   36   37   37   36
```

```
temp_t <- cbind(Jan,Feb,Mar,Apr,May)
temp_t
```

```
##      Jan Feb Mar Apr May
## [1,]  25  31  32  33  36
## [2,]  27  32  32  33  37
## [3,]  28  31  34  34  37
## [4,]  23  33  34  35  36
```

6.2 Naming matrix columns and rows

Matrix columns and rows can be named like vectors.

```
colnames(mat1) <- c("a","b","c","d")
mat1
```

```
##       a  b  c  d
## [1,]  1  2  3  4
## [2,]  5  6  7  8
## [3,]  9 10 11 12
## [4,] 13 14 15 16
## [5,] 17 18 19 20
```

```
rownames(mat1) <- c("E","F","G","H","I")
mat1
```

```
##    a  b  c  d
## E  1  2  3  4
## F  5  6  7  8
## G  9 10 11 12
## H 13 14 15 16
## I 17 18 19 20
```

row names and colnames can be retrieved using the same rownames and colnames functions.

```
rownames(mat1)
```

```
## [1] "E" "F" "G" "H" "I"
```

```
colnames(mat1)
```

```
## [1] "a" "b" "c" "d"
```

To remove names , those can be set to NULL.

```
rownames(mat1) <- NULL
mat1
```

```
##       a  b  c  d
## [1,]  1  2  3  4
## [2,]  5  6  7  8
## [3,]  9 10 11 12
## [4,] 13 14 15 16
## [5,] 17 18 19 20
```

```
colnames(mat1) <- NULL
```

6.3 Addresing matrix

1. using indices

Matrix indices are specified as matrix[row, columns].

a. Selecting specific row or rows

In index only row is specified, column index is left blank to select all columns.

```
mat1[1,] # first row will be given, column index is empty to select all columns
```

```
## [1] 1 2 3 4
```

```
mat1[c(3,2),] # third and second row will be selected
```

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

b. selecting specific column or columns

Row index is left empty and column index is specified.

```
mat1[,2]
```

```
## [1]  2  6 10 14 18
```

```
mat1[,3:4]
```

```
##      [,1] [,2]
## [1,]    3    4
## [2,]    7    8
## [3,]   11   12
## [4,]   15   16
## [5,]   19   20
```

c. selecting specific row and column

In this use case, both indices are specified.

```
mat1[3,2] # value form 3rd row and 2nd column
```

```
## [1] 10
```

```
mat1[2:3, c(2,4)]
```

```
##      [,1] [,2]
## [1,]    6    8
## [2,]   10   12
```

2. Selecting using rownmes and colnames

First name the rows and columns

```
colnames(mat1) <- c("a","b","c","d")
mat1
```

```
##       a  b  c  d
## [1,]  1  2  3  4
## [2,]  5  6  7  8
## [3,]  9 10 11 12
## [4,] 13 14 15 16
## [5,] 17 18 19 20
```

```
rownames(mat1) <- c("E","F","G","H","I")
mat1
```

```
##    a  b  c  d
## E  1  2  3  4
## F  5  6  7  8
## G  9 10 11 12
## H 13 14 15 16
## I 17 18 19 20
```

a. selecting specific row by name

```
mat1["E",]
```

```
## a b c d
## 1 2 3 4
```

```
mat1[c("E","H"),]
```

```
##    a  b  c  d
## E  1  2  3  4
## H 13 14 15 16
```

b. selecting specific column by name

```
mat1[,"a"]
```

```
##  E  F  G  H  I
##  1  5  9 13 17
```

```
mat1[, c("a","c","d")]
```

```
##    a  c  d
## E  1  3  4
## F  5  7  8
## G  9 11 12
## H 13 15 16
## I 17 19 20
```

c. selecting specific rows and columns by names

```
mat1["E","a"]
```

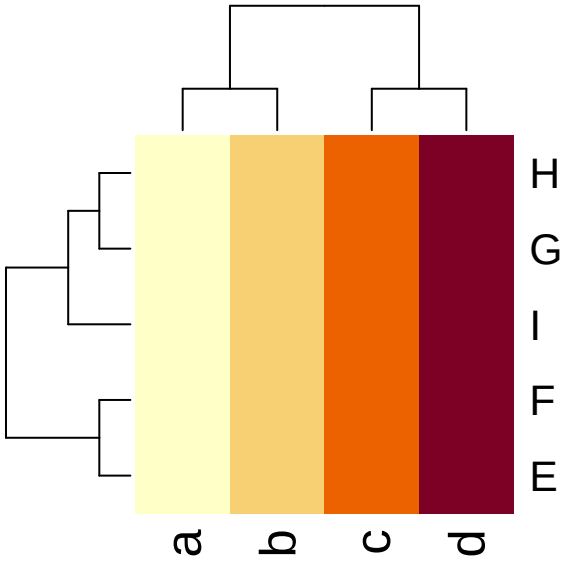
```
## [1] 1
```

```
mat1[c("F","G"), c("a","c","d")]
```

```
##   a  c  d
## F 5  7  8
## G 9 11 12
```

Matrix is deafult input for heatmap fucntion.

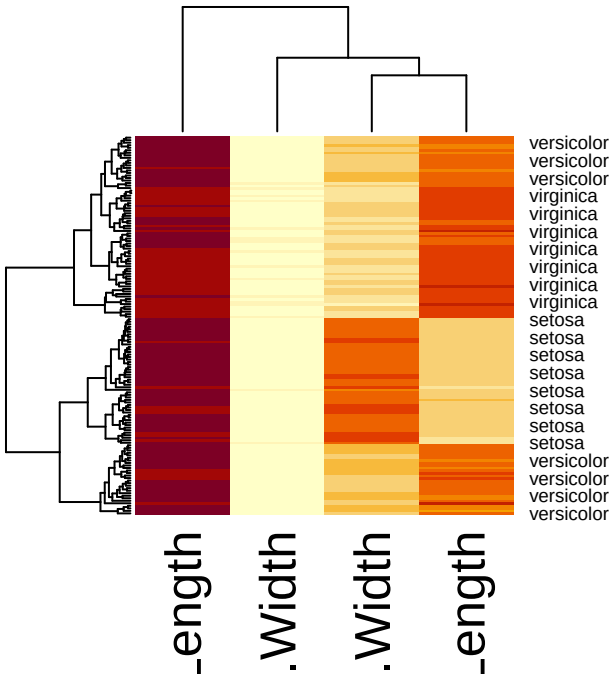
```
heatmap(mat1)
```



A data frame can also be convted to matrix using `as.matrix()` function and used for heatmap plotting.

In following example we have removed the character vector form iris data frame and converted it to matrix. Then we assigned species names to rows of the matrix using `rownames()` function. resulting heatmap is not final, can be customized further.

```
mat <- as.matrix(iris[,1:4])
rownames(mat) <- iris[,5]
heatmap(mat)
```



6.4 functions on matrix

following fucntions give more information about data structures

```
dim(mat)
```

```
## [1] 150   4
```

```
nrow(mat)
```

```
## [1] 150
```

```
ncol(mat)
```

```
## [1] 4
```

```
is.matrix(mat)
```

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

```
class(mat)
```

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

```
rownames(mat)
```

```
##   [1] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##   [6] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [11] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [16] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [21] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [26] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [31] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [36] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [41] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [46] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
##  [51] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
```

```
##  [56] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [61] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [66] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [71] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [76] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [81] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [86] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [91] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
##  [96] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
## [101] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [106] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [111] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [116] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [121] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [126] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [131] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [136] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [141] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
## [146] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
```

```
colnames(mat)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
```

6.5 mathemtical operations on matrix

```
mat1 +5
```

```
##    a  b  c  d
## E  6  7  8  9
## F 10 11 12 13
## G 14 15 16 17
## H 18 19 20 21
## I 22 23 24 25
```

```
mat1-5
```

```
##    a  b  c  d
## E -4 -3 -2 -1
## F  0  1  2  3
## G  4  5  6  7
## H  8  9 10 11
## I 12 13 14 15
```

```
mat1*5
```

```
##    a  b  c   d
## E  5 10 15  20
## F 25 30 35  40
## G 45 50 55  60
## H 65 70 75  80
## I 85 90 95 100
```

```
mat1 /5
```

```
##     a   b   c   d
## E 0.2 0.4 0.6 0.8
## F 1.0 1.2 1.4 1.6
```

```
## G 1.8 2.0 2.2 2.4
## H 2.6 2.8 3.0 3.2
## I 3.4 3.6 3.8 4.0
```

```
mat1 %% 5
```

```
##   a b c d
## E 1 2 3 4
## F 0 1 2 3
## G 4 0 1 2
## H 3 4 0 1
## I 2 3 4 0
```

```
mat1 %/% 5
```

```
##   a b c d
## E 0 0 0 0
## F 1 1 1 1
## G 1 2 2 2
## H 2 2 3 3
## I 3 3 3 4
```

```
mat1+mat2
```

```
##    a  b  c  d
## E  2  8 14 20
## F  7 13 19 25
## G 12 18 24 30
## H 17 23 29 35
## I 22 28 34 40
```

```
mat1-mat2
```

```
##    a  b  c   d
## E  0 -4 -8 -12
## F  3 -1 -5  -9
## G  6  2 -2  -6
## H  9  5  1  -3
## I 12  8  4   0
```

```
mat1*mat2
```

```
##    a   b   c   d
## E  1  12  33  64
## F 10  42  84 136
## G 27  80 143 216
## H 52 126 210 304
## I 85 180 285 400
```

7 data frames

Like matrix data frames are rectangular data structures with rows and columns. In data frame columns are variables or indivudual vectors and rows are observations on those variables.columns of data frame can hold different data types, but data type cannot be mixed up in a single column. If this happens data coercion will take place as columns follow all the rules for a vector.

Like matrix data frames have dimensions specified as row X columns.

Data frame is the most commonly used data type in R.

Lets create a data frame: data.frame() function is used to combine columns in a data frame.

```
flower <- c("hibiscus","hibiscus","rose","periwinkle")
no <- c(25,18,23,28)
color <- c("red","white","orange","pink")

df <- data.frame(flower,no,color)

df
```

7.1 adding columns to a dataframe

1. using colbind()

```
avgFlowers <- c(12,35,8,47)

df <- cbind(df,avgFlowers)

df
```

7.2 two data frames with same columns can be joined together using rbind

```
flower <- "rose"
no <- 18
color <- "yellow"
avgFlowers <- 27
df2 <- data.frame(flower,no,color,avgFlowers)

df3 <- rbind(df,df2)
df3
```

7.3 adding columnd using '$' operator

If you type data frame name and then '$' operator , Rstudio will give you dropdwon list of columns of that dataframe. when you click on the name of column you want to select, that column name will be added after '$'. this is standard way to address a column of dataframe.

Here in this example we wlill add column named " investigator" using '$' operator.

```
df3$investigator <- c("Ganesh","Ramesh","Suvarna","Ummed","Rose")
df3
```

As we said above The '$' operator can be used to retrive columns of a dataframe. Study following examples

```
df3$avgFlowers
```

```
## [1] 12 35  8 47 27
```

```
df3$color
```

```
## [1] "red"    "white"  "orange" "pink"   "yellow"
```

```
df3$investigator
```

```
## [1] "Ganesh"  "Ramesh"  "Suvarna" "Ummed"   "Rose"
```

7.4 rownames and colnames of data frame

Like matrix, we can name rows of the data frame also using `row.names()` function. By default R assign sr. no. of row as its row name. When we use `row.names()` function to see the row names , it gives a charcater vector of serial number of rows.

```
row.names(df3)
```

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

We can change these rows by assigning a vector to 'row.names()' fucntion.

```
row.names(df3) <- c("a","b","c","d","e")
df3
```

```
row.names(df3)
```

```
## [1] "a" "b" "c" "d" "e"
```

Assigning 'NULL' to `row.names` , removes these names. R will again sssign character vector of serial numbers as names.

```
row.names(df3) <- NULL
df3
```

```
row.names(df3)
```

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

7.5 Fucntions to explore data frame

Many functions are common to matrix and data frames. fucntions like `ncol`, 'nrow', 'dim', 'rownames' and 'colnames' we can use to know more about data frames and matrices.

```
dim(df3)
```

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

```
nrow(df3)
```

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

```
ncol(df3)
```

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

```
rownames(df3)
```

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

```
colnames(df3)
```

```
## [1] "flower"       "no"           "color"        "avgFlowers"   "investigator"
```

```
names(df3) # synonym of colnames
```

```
## [1] "flower"       "no"           "color"        "avgFlowers"   "investigator"
```

Function like `head()`,`tail()`, `summary()` and `str()` give more information about the data frame.

1. `head() or tail()` function

If data frames contains more observation by loading it fully in console, will flood the console. And console will become difficult to handle. Till now we have seen small data frames. The `iris` data frame has 150 observations and when loaded fully it floods the console. To avoid flooding of console, `head()` function or

`tail()` fcutnion is used. by default it shows first 6 or last 6 rows respectively to give a glimpse of the data frame. This glimpse tails us about the vector data types, and we can have gross idea about contents of the data frame.

```
head(iris)
```

```
tail(iris)
```

```
head(iris,10) # to show 10 rows
```

```
tail(iris,10) # to show 10 rows
```

2. summary of data frame

Summary of data frame can be obtained using summary() function. This is very generic function. You can use it for any object , you will be returned with suitable summary for that respective object.

5 point summary on all numeric variables, counts of the categorical and logical variables will be provided. The five point summary include mean, median,minimum and maximum and 1st and third quantiles. (Mean is not part of the 5 point summary concept).

This information very aptly tells range and destribution of the numeric data and counts of the categorical and logical data.

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

```
summary(df3)
```

```
##     flower                no          color             avgFlowers
##  Length:5           Min.   :18.0   Length:5           Min.   : 8.0
##  Class :character   1st Qu.:18.0   Class :character   1st Qu.:12.0
##  Mode  :character   Median :23.0   Mode  :character   Median :27.0
##                     Mean   :22.4                      Mean   :25.8
##                     3rd Qu.:25.0                      3rd Qu.:35.0
##                     Max.   :28.0                      Max.   :47.0
##  investigator
##  Length:5
##  Class :character
##  Mode  :character
##
##
##
```

3. Str() fucntion

This is also very useful function used to explore the data in initial phases of data analysis. It tells dimensions of data frame followed by list of vectors with their data types and initial few values. This can be a good alternative to head()and tail() fucntions.

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 ...
```

7.6 subsetting dataframe

Just like matrix data frame can be sub-setted using indices, using names of rows and columns and using logical statements. As we have seen above '$' operator is one more way of adressing data frame elements. The df$colname can be treated as vector, where all the rules for sub-setting the vectors can be used.

1. selecting a column or row using index

When a data frame is substetted to get a row only or a column only we get vector as output. This can be done by using using single index n place of row or column.

```
iris[1,] # first row
```

```
iris[,1] # firts column
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

2. selecting a column using "$" operator

Row operator is very commonly used to specify columns of a data-frame or objects of list which we will see later. Structure of the command is df$columnname.

```
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

3. to select multiple columns and rows

For this the indices are the best.

This subsetting returns dataframes which are subsets of the original. Vectors or conitional statements can be used in indices.

```
iris[3:4,1:2]
```

```
iris[c(1,5,10),c(1,5)]
```

7.6.1 to select based on logical conidtion

Logical condition like '>','<' etc. can be used to select specific segments of a data frame.

In example below we will select the rows where sepal length of iris flowers in greater than the mean of sepal Length.

```
iris[iris$Sepal.Length > mean(iris$Sepal.Length),]
```

In the example below we will select Setosa flowers only which have sepal length greater than mean sepal length. Please note the "==' (double equal sign) for equality. If you forget that '=' , R will throw error."& is used to combine the conditions.

```
iris[iris$Sepal.Length > mean(iris$Sepal.Length) & iris$Species == "setosa",]
```

This tells us that all the setosa flowers have length below the mean of Sepal Length.

Here we will use "!=" (not equal to) operator

```
iris[iris$Species != "virginica" & iris$Sepal.Length > mean(iris$Sepal.Length),]
```

7.6.2 subsetting dataframes using sample fucntion

Sample function sub sets a vector by selecting random values form it.

To subset a data set using sample function we create a vector of randomly selected indices to randomly select rows.

a. create vector of indices

```
samp <- sample ( 1: nrow(iris), 10)
samp
```

```
##  [1] 128  96 136  31  82  59   7  87  28 107
```

b. select the rows using these indices

```
iris[samp,]
```

To exclude these rows from the selection use '-' operator in front of the index vector names.

Here i have used to dim to show dimensions of the subsetted data frame, to save space

```
dim(iris[-samp,])
```

```
## [1] 140   5
```

This random sampling is very useful to sample rows form a large data for study. In Monte Carlo simulation, this is used to generate thousands of subsamples form the sample to generate population estimates.

7.6.3 Subsetting data frame using subset() function

This fucntion subsets data frame using condition. This is easier and simplified than using conditions in indices. Code also remains readable.

Please note this function takes dataset name as first argument and then subsetting condition to select rows.

```
subset(iris,Sepal.Length< mean(Sepal.Length))
```

Select argument is used to select from the specific columns.

```
subset(iris,Sepal.Length> mean(Sepal.Length),select=Species)
```

Please note here : subset function returns a dataframe, even if we select a single row or single column even a single value.

Play with these to understand these more. Use different data sets to learn these functions.

8 lists

Lists are very interesting data structure in R. lists can accommodate anything. It can have individual data values , vectors, matrices, data frames, plots or any other object R can create.

Lets add what we have in our environment in a list and see it using indices and '$' operator. `list` function is used to create list.

```
lst_1 <- list("Ganesh",temp,flower, df3,mat2)
lst_1
```

```
## [[1]]
## [1] "Ganesh"
##
## [[2]]
##     [,1] [,2] [,3] [,4]
## Jan   25   27   28   23
## Feb   31   32   31   33
## Mar   32   32   34   34
## Apr   33   33   34   35
## May   36   37   37   36
##
## [[3]]
## [1] "rose"
##
## [[4]]
##       flower no  color avgFlowers investigator
## 1   hibiscus 25    red         12       Ganesh
## 2   hibiscus 18  white         35       Ramesh
## 3       rose 23 orange          8      Suvarna
## 4 periwinkle 28   pink         47        Ummed
## 5       rose 18 yellow         27         Rose
##
## [[5]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

9 Exporting Data to a file in R

9.1 Working directory in R

For this we first need to understand the concept of working directory. Working directory is the directory on our computer drive, from which we want to import the data or to which we want to save our work in the current R session.

Working directory can be set using `setwd()` fucntion. For this just go to the directory which you want to use as working directory and click in the address bar of the directory, path of the directory will be selected. Cpy the path of the directroy and pasdte it between the inverted quotes of the setwd() fucntion and just add one more backslash in fornt of each backslash in the path. your path shll look like as given below

setwd("H:\Rworks\Workshop 20th March") and run that code. You can see the directory by typing in 'getwd()" fucntioj in console or just looking at top of the console, right to the R logo.

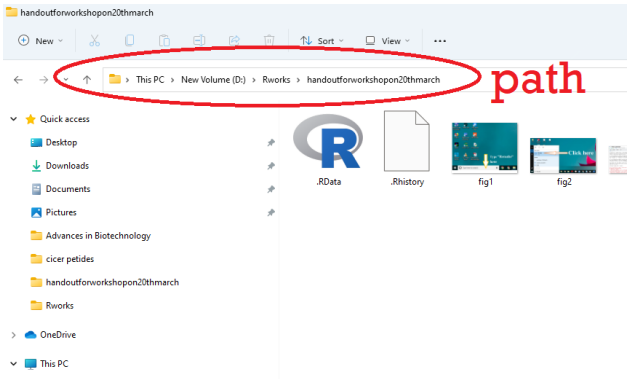


Figure 7: Path of the directory

Data frames and matrices can be exported to different data file formats as per requirements.

Lets export the data frame df3 we created above in different data formats. Write functions are used for this purpose. Don't forget to put the extension in file name and enclose the file name in inverted quotes. The file will be saved in working directory.

9.2 Saving as .txt file

here we will use function write.table function.

```
write.table(df3,"flowerinfo.txt")
```

Check the file in your working directory. It will have space between the values and linebreak after each row. If we wish to specify separator between the values , we can use `sep` argument. include separator n inverted quote. Here we will use comma. it can be ';' also. these two are most common.

The data in .txt file can be opened in spreadsheet application like MS-Excel. But during opening, it will ask to specify appropriate separator.

```
write.table(df3,"flowerinfo_comma.txt",sep=",")
```

Some people use tab as separator.

```
write.table(df3,"flowerinfo_tab.txt",sep="\t")
```

9.3 Saving as .csv file

CSV files are most common data files. like the .txt files they use minimum memory. the function write.csv(), writes the .csv file. Most spreadsheet applications like MS-Excel, LibreOffice, OpenOffice open .CSV file as spreadsheet. In some instances these applications may ask to specify separators.

```
write.csv(df3,"flowerinfo_tab.csv")
```

By default .csv file, use "comma" as separator, but you can specify separator we have discussed above.

9.4 Saving data as xls file

For this you require to install package `xlsx`. As we discussed above, when you install the package you have to call that as library to bring its function in memory.

1. install the package `xlsx`

You can install the package by clicking Tools -> install package.

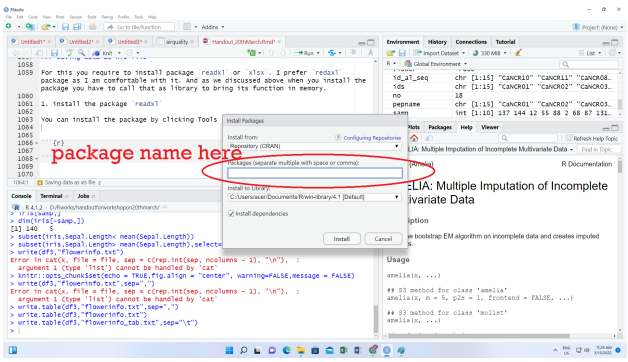


Figure 8: Install packages dilogue box

To install using r code, remove the '#' at the beginning of `install.packages()` command and run that line. Please remember not to mixup uppercase/ lowercase in package name. Please note package name is between inverted quotes.

```
#install.packages("xlsx")
```

After installation of package, call the library.

```
library(xlsx)
```

(You may face error " error: JAVA_HOME cannot be determined from the Registry" while laoding library "xlsx". read instruction form the following link to fix that.

https://statisticsglobe.com/r-error-java-home-cannot-be-determined-from-the-registry#:~:text=So%20why%20does%20the%20error,64%2Dbit%20version%20of%20Java.)

Now we are ready to write data as file. Here we will save file as .xlsx file.

```
write.xlsx(df3,"flowerinfo_tab.xls")
```

If you see the written file, The first column is row number column,. Mny a times that column creates issues , if you read the file in R. Therfore we will use argument `rownames= false` to avoid that

```
write.xlsx(df3,"flowerinfo_tab.xls",row.names = FALSE)
```

We can name the sheet to which data is being written.

```
write.xlsx(df3,"flowerinfo_tab.xls",row.names = FALSE,sheetName = "FlowerInfo")
```

To know more about `write.xlsx()` function just type in and run `?write.xlsx()` in console.

We can add data to a new sheet in existing Excel workbook. Just use argumwent `append=TRUE`

```
write.xlsx(iris,"flowerinfo_tab.xls",row.names = FALSE,sheetName = "Iris Data set", append=TRUE)
```

There is much to `xlsx` package, but we rarely need all those functions. In this session , this much is enough.

9.5 Save data as .dta file

`.dta` is a file formats in which STATA save its data. R can export data as `.dta` file using `save.dta13()` function.

```
#install.packages("readstata13")
 df =read.csv("polyphenolassay.csv")
library(readstata13)
save.dta13(df,"PPassay.dta")
```

10 Importing data

R can import almost all data formats. Here we will learn how to import data in text format, csv format and xls and xlsx format.

10.1 Import Text data

The in `.txt` or other plain text formats can be imported using `read.table()` function, but here don't forget to specify `sep=` argument . it specifies separators separating the values. separators can be space(' '), tab ('^t), comma(',') or semi-colon(';').

```
pg <- read.table("PlantGrowth.txt",sep=",")
head(pg)
```

Header of our file has got read as data and not as header. We will sepcify argument to get header properly.

```
pg1 <- read.table("PlantGrowth.txt",sep=",",header = TRUE)
head(pg1)
```

10.2 Import data from .csv file

Base R provide `read.csv` function to read csv files. csv is very common data file format in which separator is comma and hence the name comma separated value file.

```
df_csv <- read.csv("flowerinfo_tab.csv")
head(df_csv)
```

10.3 Import data form .xls or xlsx file

MOst of the students and researchers are well versed with MS-excel and use it as the only spreadsheet application for their data entry and data analysis operations. R provide very powerful visualization and analytics functions. The data saved as `.xls` or `.xlsx` format can be imported to R using functions from many packages. Here we will use `read_xls` or `read_xlsx` fucntion form `readxl` package.

```
library(readxl)
df_xls <- read_xls("flowerinfo_tab.xls")
```

we can specify rows and columns to be imported as well as specific sheet to be imported.

10.4 import data form .dta file

STATA program use .dta as file extension to save data. `readstata13` package provides function to read .dta file.

```
library(readstata13)
df_dta <- read.dta13("PPassay.dta")
head(df_dta)
```

11 Manipulating data:

11.1 taking care of missing data

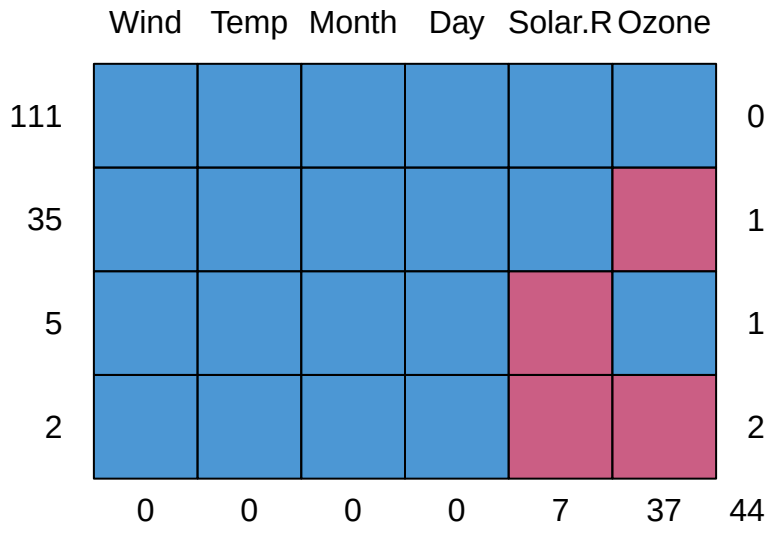
Missing data can pose a big issue in data anlysis. If very few missing data values are there sometimes the rows containig missing data can be safely omitted out. But in many cases omitting the rows result in loss of significant amount of data. Data anl;ysis on such data can give spurious results.

For that the missing values are replaced by most probable values. these values can be mean of the column, or median of that column or can be obtained by fitting statistical model to the whole data set to predict the specific missing value. This process is called as dat imputation

Many packages like mice, AMELIA, metan provide functions to impute missing data.

We will first visualize the missing data using `md.pattern()` function of `mice` package. and then will impute the data. the dataset we will use is `airquality`.

```
library(mice)

md.pattern(airquality)
```




```
##     Wind Temp Month Day Solar.R Ozone
## 111    1    1     1   1       1     1  0
## 35     1    1     1   1       1     0  1
```

```
## 5      1    1     1   1       0     1  1
## 2      1    1     1   1       0     0  2
##        0    0     0   0       7    37 44
```

here we can see that solar.R and Ozone columns have missing data. We will now impute those uysing mice fucntion.

```
imp_data <- mice(airquality)
```

```
##
##  iter imp variable
##   1   1  Ozone  Solar.R
##   1   2  Ozone  Solar.R
##   1   3  Ozone  Solar.R
##   1   4  Ozone  Solar.R
##   1   5  Ozone  Solar.R
##   2   1  Ozone  Solar.R
##   2   2  Ozone  Solar.R
##   2   3  Ozone  Solar.R
##   2   4  Ozone  Solar.R
##   2   5  Ozone  Solar.R
##   3   1  Ozone  Solar.R
##   3   2  Ozone  Solar.R
##   3   3  Ozone  Solar.R
##   3   4  Ozone  Solar.R
##   3   5  Ozone  Solar.R
##   4   1  Ozone  Solar.R
##   4   2  Ozone  Solar.R
##   4   3  Ozone  Solar.R
##   4   4  Ozone  Solar.R
##   4   5  Ozone  Solar.R
##   5   1  Ozone  Solar.R
##   5   2  Ozone  Solar.R
##   5   3  Ozone  Solar.R
##   5   4  Ozone  Solar.R
##   5   5  Ozone  Solar.R
```

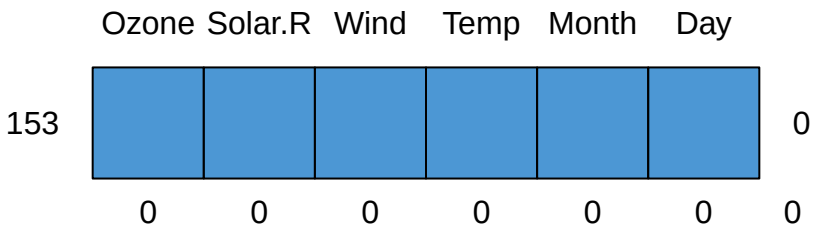
Imputation has been done and the results are saved in imp_data object. We will now construct the complete data(where missing values have been imputed, or replaced with the most probable value at that position.)

```
Na_rmd_airquality <- complete(imp_data)
```

We will confirm whether the missing have got imputed or not.

```
md.pattern(Na_rmd_airquality)
```

```
##  /\     /\
## {  `---'  }
## {  O   O  }
## ==>  V <==  No need for mice. This data set is completely observed.
##  \  \|/  /
##   `-----'
```




```
##     Ozone Solar.R Wind Temp Month Day
## 153     1       1    1    1     1   1 0
##         0       0    0    0     0   0 0
```

11.2 Piping of code

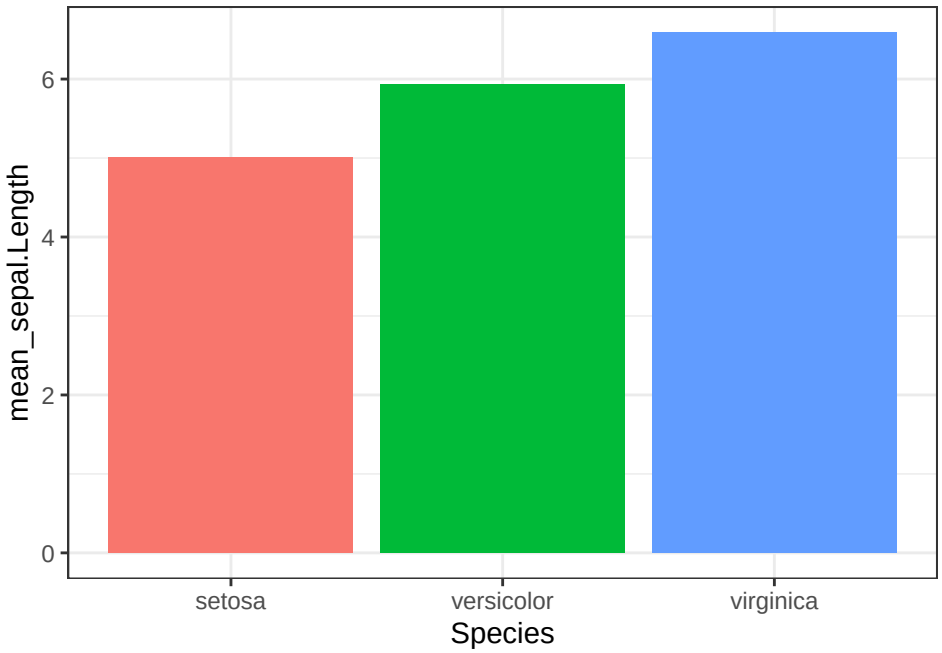
pipe operator %>% is very useful addition to R programming. This is adopted form `unix`. Here we provide input via pipe to a function , then output of theta function can be piped to next one and such pip line can be of any length. At the end of the pipe we will get output. This output can be assigned to an object.

Note here that you don't have to put data in the functions used in pipes. data is added at the beginning of the pipes. usual syntax for pipe is

```
output object <- data %>% function1() %>% fucntion2 %>% .......
```

Here is an example of pipe concept to get a graph by ggplot2. Don't bother about understanding the code here, all that code we are going to learn in due time.

```
barplot <- iris %>%
  select(Sepal.Length,Species) %>%
  group_by(Species) %>% summarise(mean_sepal.Length =mean(Sepal.Length)) %>% ggplot(aes(Species,mean_se
  geom_bar(stat="identity")+
  theme_bw()+
  theme(legend.position = "none")

barplot
```



Piped code is easy to read code and at the same time it saves labor and time required to get an output.

And note that pipe operator has been placed at the end of line and not in the beginning of line. The '+' operator we used in ggplot is also placed at the end of line.

11.3 Intordcution to tidyverse

`Tidyverse` includes many packages which have become gold standard in data analytics. for us `dplyr`,`tidyr`, `lubridate` and `ggplot2` are very useful. Of these In this session we are going to learn `dplyr`, `tidyr` and `ggplot2`. We are going to touch `lubridate` just to get date data in correct format.

11.4 fucntions from `dplyr` package

11.4.1 select

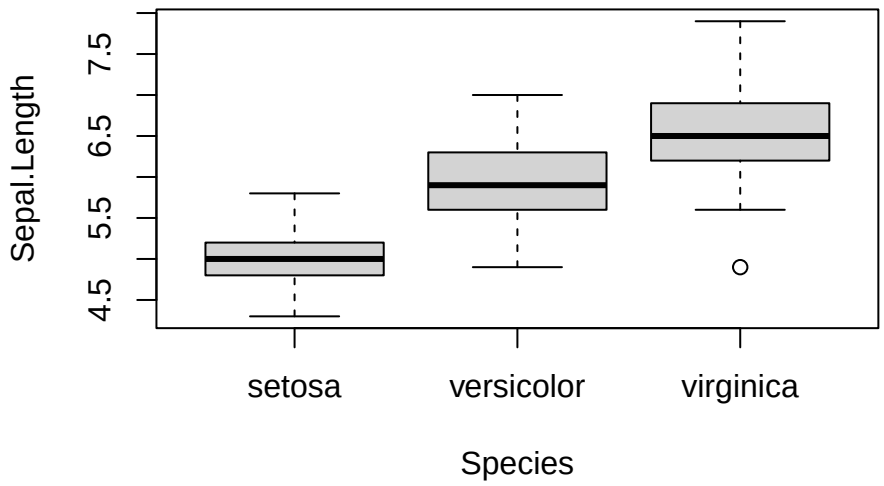
This function is useful to subset the data set by selecting bout some columns. During data analysis situations arise when we don't need certain columns of the data during analysis. Under these situations we require the select function.

here we will use iris dataset as an example dataset. This dataset contain 4 numeric data columns and one character data column.

In following example, we will use select function to select a numeric column and a character column and use that new dataset for plotting boxplots, to compare data distribution.

Try this with all three remaining numeric variables.

```
library(dplyr)

df_sl <- iris %>% select(Species,Sepal.Length)

boxplot(Sepal.Length~Species,iris)
```




```
dim(df_sl)
```

```
## [1] 150   2
```

```
head(df_sl)
```

11.4.2 filter

Filter is a function which filters rows of a data frame using one or more conditions.

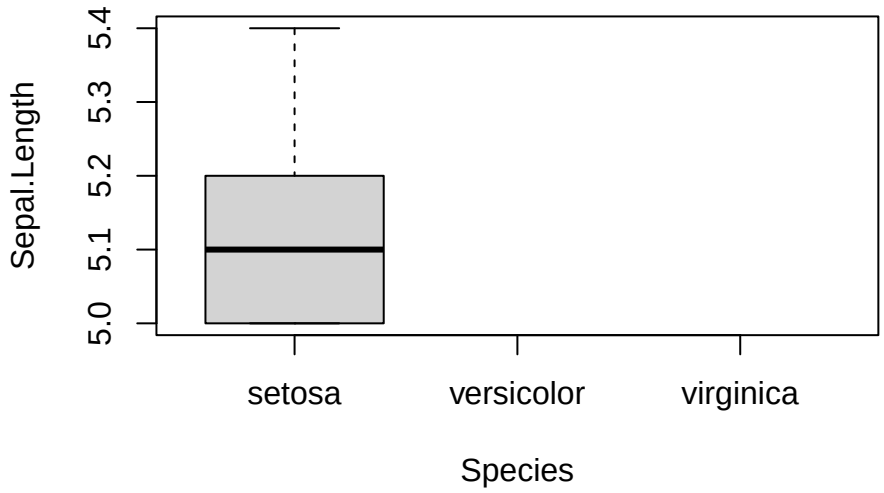
here in this example we will filter rows which have `Species== "setosa"`. Note here the double equal to sign. In R equality is specified by '==' sign.

```
iris_setosa <- iris %>% filter(Species=="setosa" & Sepal.Length >=5 & Sepal.Length <=5.4)

dim(iris_setosa)
```

```
## [1] 25  5
```

```
head(iris_setosa)
```

```
boxplot(Sepal.Length~Species,iris_setosa)
```



11.4.3 Mutate

This function adds new columns to a dataframe. these new columns are fucntions of the existing column.

```
iris_ratios <- iris %>% mutate(Sepal.ratio = Sepal.Length/Sepal.Width, Petal.ratio = Petal.Length/Petal

head(iris_ratios)
```

We will study practical application to other data set.

```
head(women)
```

```
women_metric_bmi <- women %>% mutate(height_mtrs = 0.0254*height, weight_kg=0.453592*weight) %>% mutate

head(women_metric_bmi)
```

11.4.4 Transmute

This function is similar to mutate, but it returns a dataframe with newly computed columns.

In case of the above example of `women` dataset , transmue fucntion will be more approrpiate.

```
women_metric_bmi_2 <- women %>% transmute(height_mtrs = 0.0254*height, weight_kg=0.453592*weight) %>% mu

head(women_metric_bmi_2)
```

11.4.5 arrange

This function arrange the rows of dataset as per values of specified column, we can arrange in ascending or descending order.

```
women_arranged_asc <- women %>% arrange(height) # ASCENDING
head(women_arranged_asc)
```

Negative sign in front of the vector to be used for arrange, orders the rows in descending order.

```
women_arranged_dsc <- women %>% arrange(-height) # ASCENDING
head(women_arranged_dsc)
```

More than ne column can also be used in the arrange function, data will be arranged as per first columns and then as per second column and so on. Here also we can use '-' sign to arrange in descending order.

This is helpful to arrange data where ties are there in the column used for arranging.

Women data set is very small data set , therefore such ties are not there. But for vey large data sets, this arrange using multiple columns is very useful. We use this function to generate merit lists during admission process.

In Excel `custom sort` function does this work.

11.4.6 group_by Summarise fucntion

group by and summarise function are very useful functions for data analysis work. group_by functions group the data according to the given criterion and summarise function apply summary functions to those grouped data and calculate summary fr each group.

```
iris_summary <- iris %>% group_by(Species) %>% summarise(mean_sepal.Length=mean(Sepal.Length),mean_Sepa

iris_summary
```

11.5 Function of "tidyr' package

11.6 pivot

`pivoe_longer()` function of `tidyr` package of tidyverse gathers values form different columns in single column, and also makes another column for the names of the gathered columns. This is done to make a tidy data.

Tidy data is a data where each column is a variable and each row is an observation. This means that if observations regarding a single measurement are spread in many columns, those need to be gathred in single column to make it tidy.

For example in `iris` data set length measurements of flower parts are spread in four numeric columns. These need to gathered in a single column , which can be named as "flower_part_length" and the column headings can be gathered in other column named as "flower part ' so that the value belonging to specific organ will be in the row of the specific column. You will understand this better with example.

The iris data which we are using erstwhile is wide data and we ill make it long data or tidy data using following R code.

```
library(tidyr)
iris_long <- iris %>% pivot_longer(cols=1:4, names_to="Flower_part", values_to="Flower_part_length")

head(iris_long)
```

```
summary(iris_long)
```

```
##        Species    Flower_part        Flower_part_length
##  setosa    :200   Length:600         Min.   :0.100
##  versicolor:200   Class :character   1st Qu.:1.700
##  virginica :200   Mode  :character   Median :3.200
##                                      Mean   :3.465
##                                      3rd Qu.:5.100
##                                      Max.   :7.900
```

```
str(iris_long)
```

```
## tibble [600 x 3] (S3: tbl_df/tbl/data.frame)
##  $ Species           : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Flower_part       : chr [1:600] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
##  $ Flower_part_length: num [1:600] 5.1 3.5 1.4 0.2 4.9 3 1.4 0.2 4.7 3.2 ...
```

See here carefully now. The column `Flower_part` has headings of the four numeric column and in the column `Flower_part_length` corresponding values are gathered.

These long tables make analysis easier. Here we can apply data analysis functions very easily.

following analysis we will do to know effect of species and flower parts on the length using analysis of variance(anova) test.

```
mod_anova= aov(Flower_part_length~.,iris_long)
summary(mod_anova)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## Species       2  309.6   154.8   247.3 <2e-16 ***
## Flower_part   3 1656.3   552.1   882.1 <2e-16 ***
## Residuals   594  371.8     0.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In anova we are interested in which vector has significant effect on variation in response variable. Here our response variable is Flower_part_length. By gathering the values form four numeric variables, we are able to use them as response variable and successfully tested effect of other variables on it.

12 Plot data

There are many graph plotting systems in R. Base R is a very elegant plotting system , but it is little difficult to learn. Lattice, plotrix are other popular packages for getting data visualized. In this session we will see how to visualize data using using ggplot2 package.

GGplot plots data using grammar of graphics. It builds plot layer by layer. following are the steps in getting a nice plot using ggplot2 package.

1. specify data
2. specify how variables are to be mapped to visual elements of plot(axis, color, fill , shape)
3. specify geometry to be used to show data
4. specify statistics to show the geometry
5. customize to make plot more readable, more informative and more appealing.

12.1 the first plot

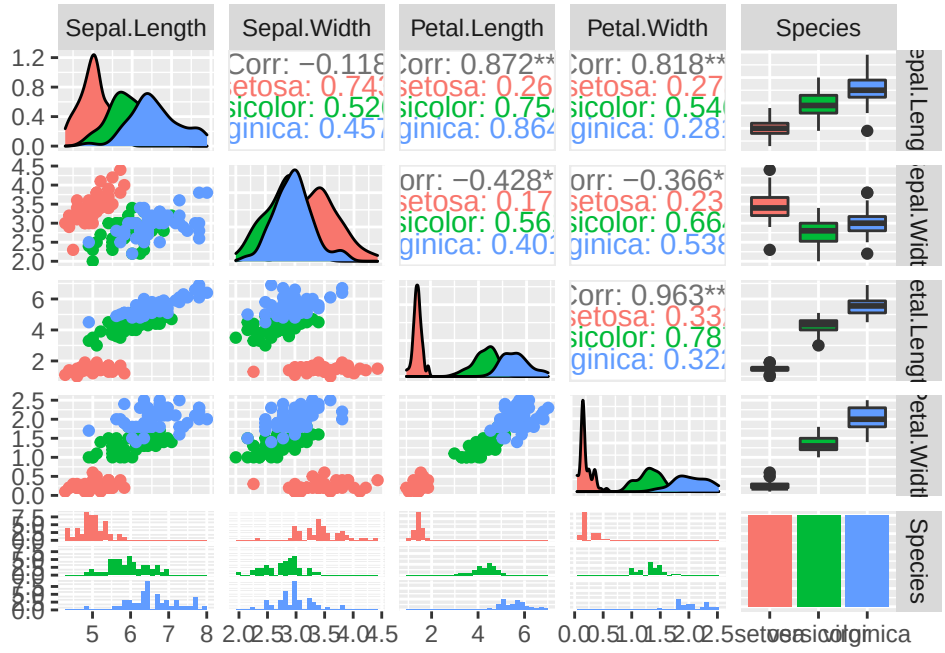
12.1.1 Preparation:

If you have not loaded tidyverse earlier, you need to load library ggplot2, before starting to plot using ggplot functions. Also you need to understand the data and decide what is you purpose of plotting? what information in data you want to reveal, for example if you want to show correlation between two numeric variables, you need scatter plot, to rank variables according to magnitude of their values , you need to use barplot or its alternatives, if you want to show distribution of data you need boxplot, density plot or histogram, if you need to show evolution over time or other continuous variable, you need to use line graph..

Still many more plot types are available. But in this session we will learn basic plot types only. And when we get sufficient level of expertise in R, we will progress to other plot types.

To decide on the plot first know what types of variable you have? for that you can use the function `head()`, `summary()`,`str()`etc. you can install package `GGally` and use function `ggpairs` to see different plot types possible with the data.

```
library(GGally)

ggpairs(iris,mapping=aes(color=Species))
```



Here you can see that you can plot density plot to see distribution, you can plot box plot also for this purpose, and can plot scatter plot to show correlation between numeric variable. Study this multipanel graph carefully. Its very useful display to understand patterns and trends in the data.

Lets explore the data using `head(), str() and summary()` function to know the data more.

```
head(iris)
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
```

```
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Now we have understood that our data contain 4 numeric variables and 1 categorical variable with three levels(Species- setosa, versicolor and verginica). We can plot most of the plots using this dataset.

12.1.2 Lets load the library

```
library(ggplot2) # tidyverse has already loaded this libarary. But many a times you mey not require all
```

Now we are ready to plot.

12.2 Lets plot

12.2.1 Histogram

1 Specify the data

ggplot is the base function of ggplot2, it's major arguments are - data and variable mapping to axis. HEre I just have jsut specified the data.

```
ggplot(iris)
```

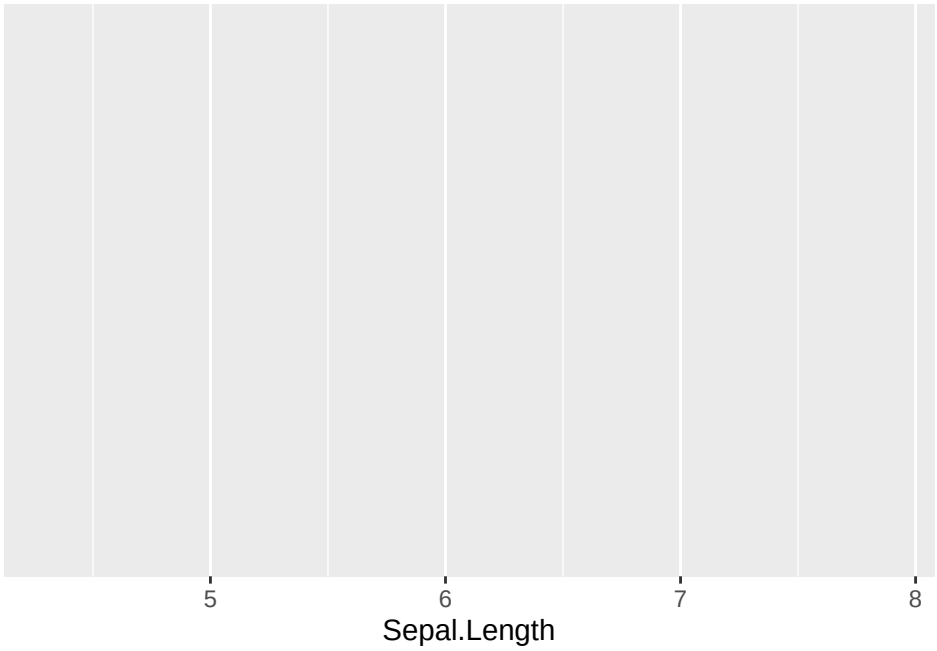

We have not got the plot. We have got just the gray canvas, on which se are going to build plot layer by layer.

2 Specify mapping

we map variables to various asthetics or visual elements of plot using aes() function. (aes = asthetics.) first two arguments of aes are x and y axis, here only you can specify colorm fill, shape size etc. mapping.

we will plot most used monovariate plot first, frequency of observations we will plot as histogram.

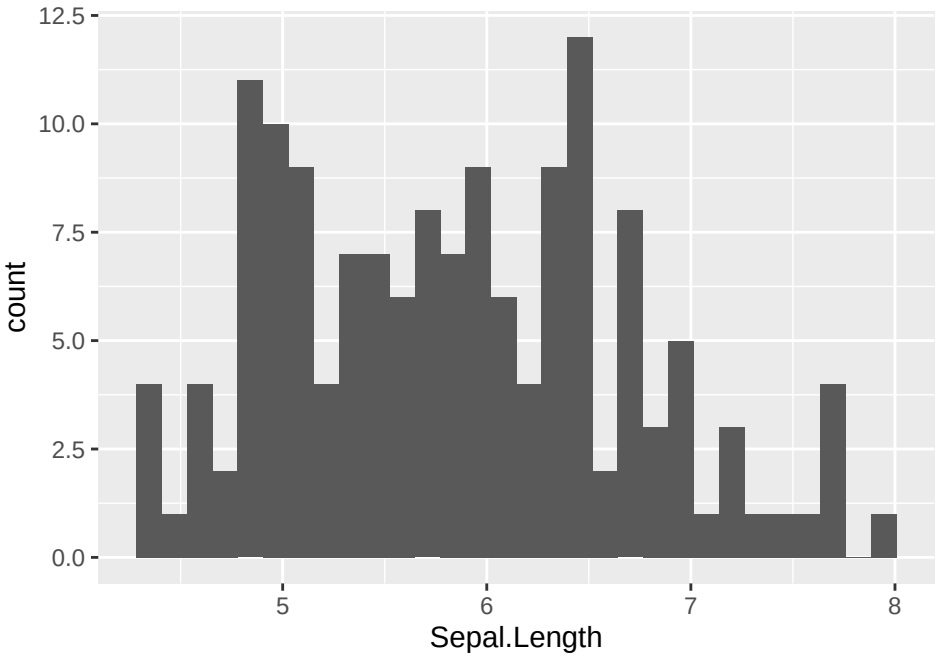
```
ggplot(iris,mapping=aes(x=Sepal.Length))
```



Note here- We have got scale on x xis. This scale is calculated based on range of observation values in x. the range is divided into equidistant breaks. But data is not shown as we have not yet specified geometry.

Lets specify geometry using geom layer. We add layer using '+' at the end of first line of ggplot code. (If you put that '+' at beginning of line, R will throw error)

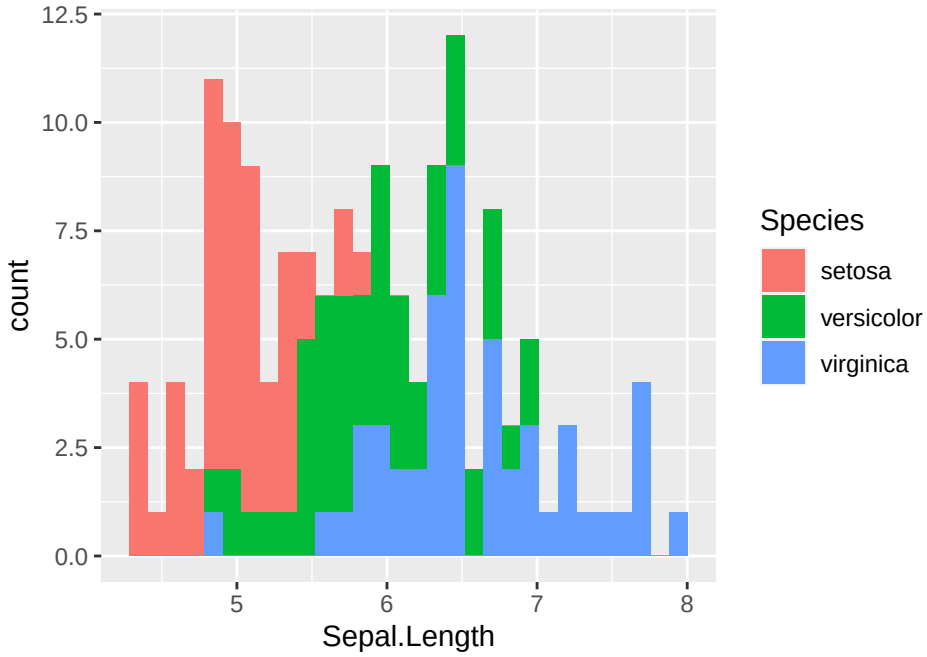
```
ggplot(iris,mapping=aes(x=Sepal.Length))+
  geom_histogram()
```



Hello, we have got the plot. Now lets make it beautiful.

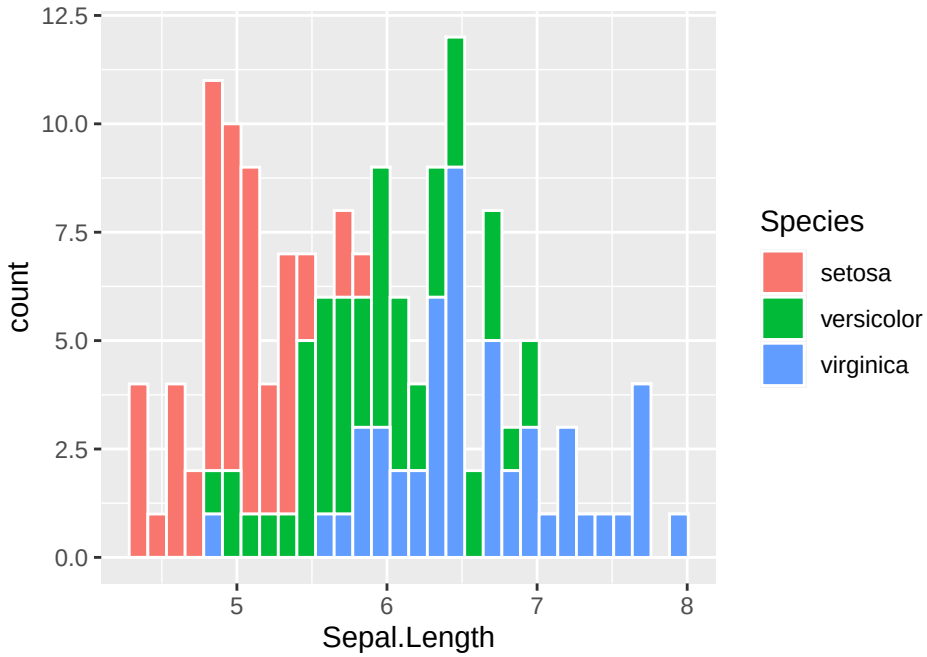
First we will fill color as per species. For this we will map fill attribute to species variable.

```
ggplot(iris,mapping=aes(x=Sepal.Length,fill=Species))+
  geom_histogram()
```



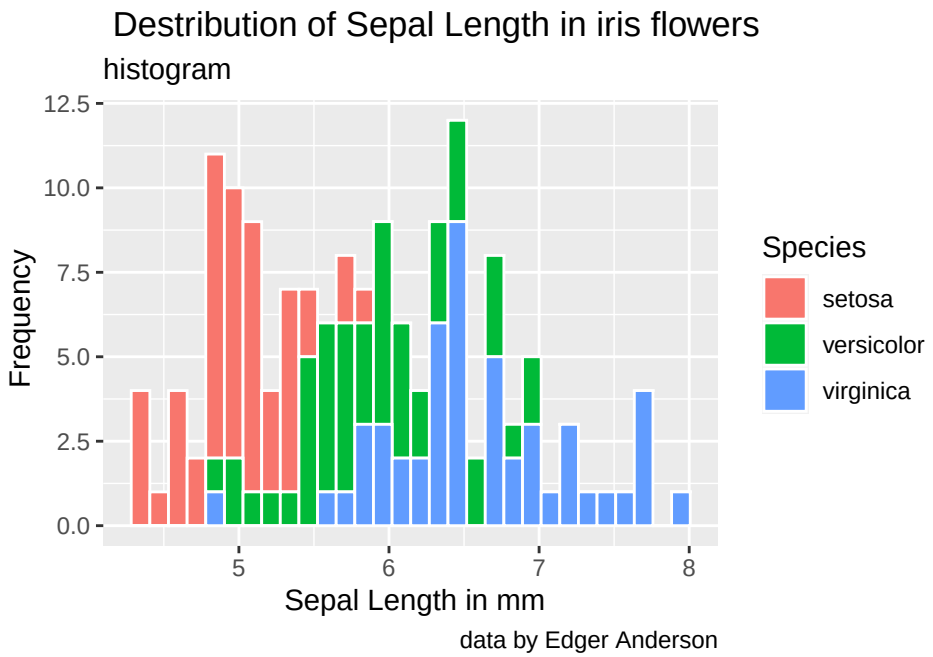
Still more beautiful by specifying color to mark boundary for bars. here we are not mapping color to any variable, therefore we will not specify it in aes. We will specify that in geom_histogram()

```
ggplot(iris,mapping=aes(x=Sepal.Length,fill=Species))+
  geom_histogram(color="white")
```



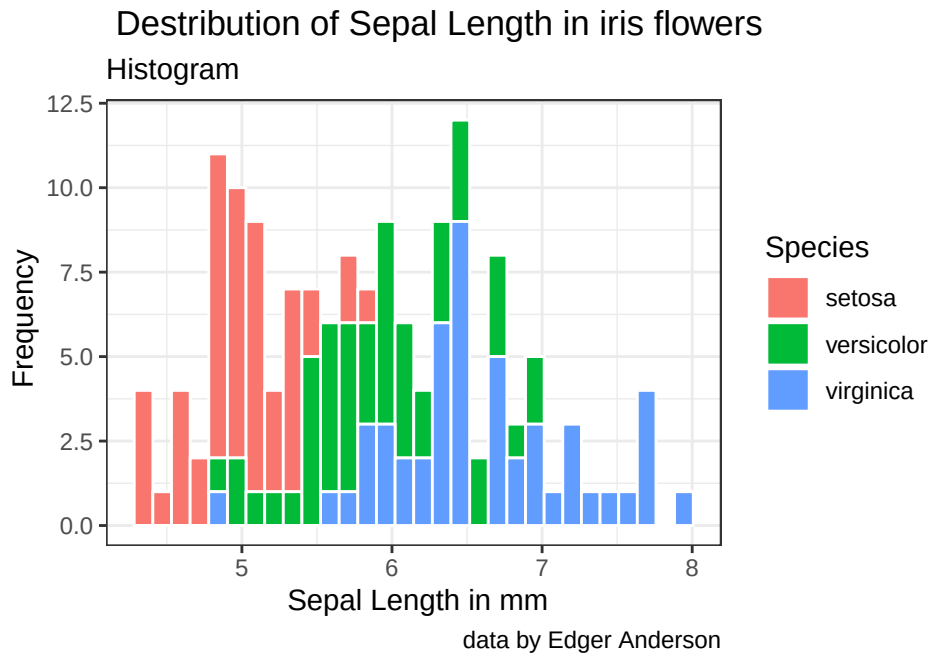
Now we will give labels to plot.

```
ggplot(iris,mapping=aes(x=Sepal.Length,fill=Species))+
  geom_histogram(color="white")+
  labs(title=" Destribution of Sepal Length in iris flowers", subtitle="histogram", caption="data by Edg
```



Now we will change overall appearance of plot by specifying theme. GGplot comes with many predefined themes.

```
ggplot(iris,mapping=aes(x=Sepal.Length,fill=Species))+
  geom_histogram(color="white")+
  labs(title=" Destribution of Sepal Length in iris flowers", subtitle="Histogram", caption="data by Edg
  theme_bw()
```



Still much can be done with this plot to make it publication ready. But this is also satsfactory.

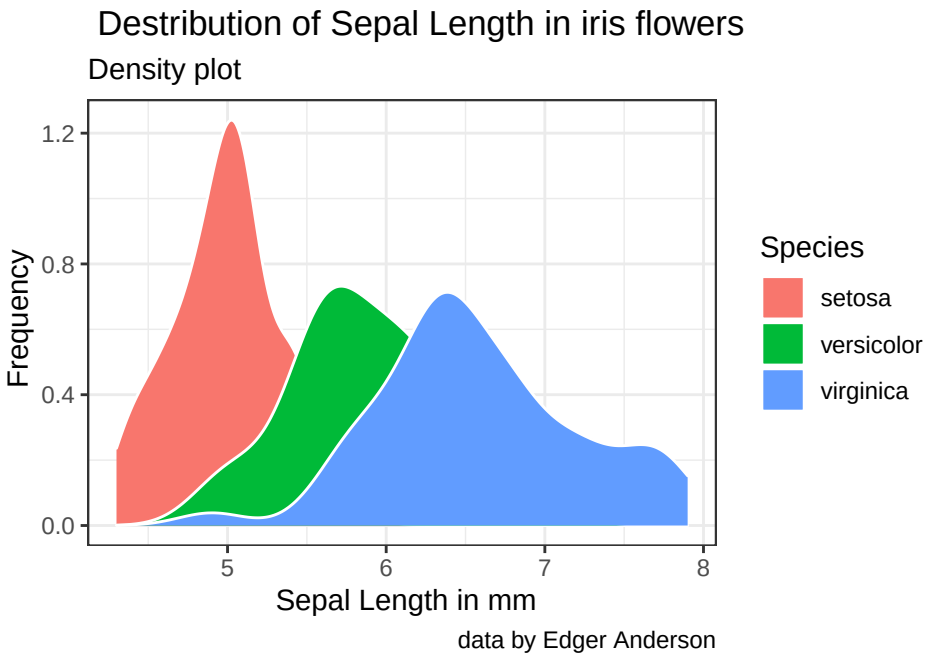
Graph plotting is a great skill to learn.

Now we will see how to use the same code which we devloepd above to get different plot types.

12.3 Density plot

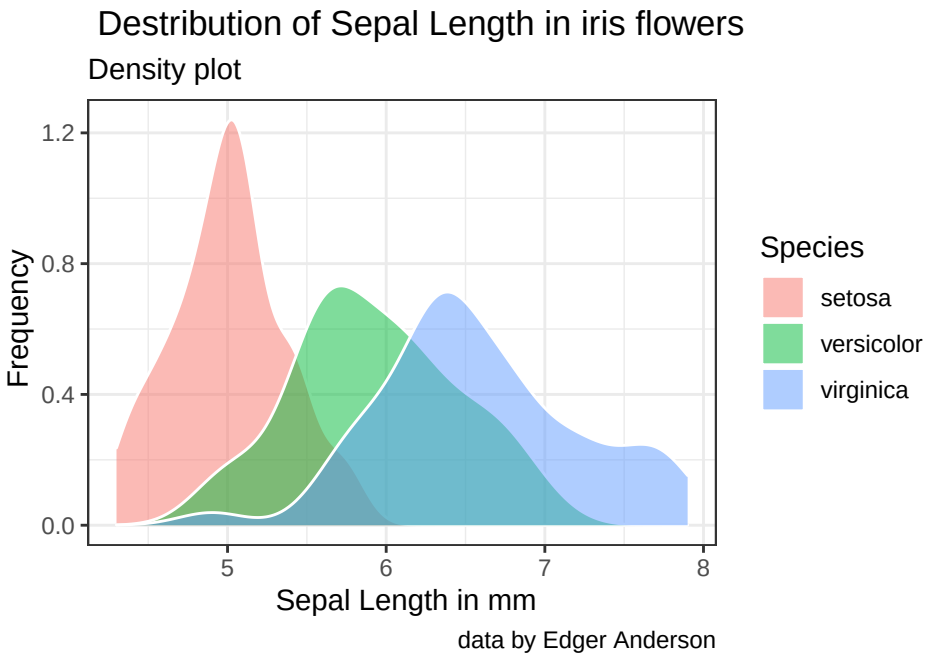
Density plot is also a monovariate plot , and is alternative of histogram, but it shows density on y axis instead of frequency. Just change the geom to density and density plot is ready.

```
ggplot(iris,mapping=aes(x=Sepal.Length,fill=Species))+
  geom_density(color="white")+
  labs(title=" Destribution of Sepal Length in iris flowers", subtitle="Density plot", caption="data by
  theme_bw()
```



Density plot can be made more informative by making each increasing transoperency of the color using attribute alpha(range - 0 to 1).

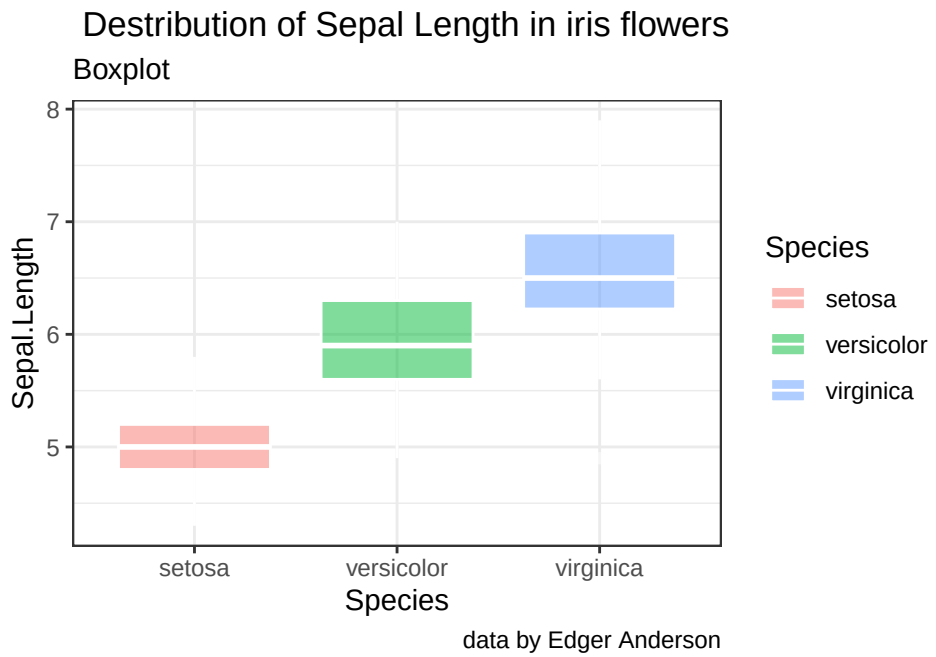
```
ggplot(iris,mapping=aes(x=Sepal.Length,fill=Species))+
  geom_density(color="white", alpha=0.5)+
  labs(title=" Destribution of Sepal Length in iris flowers", subtitle="Density plot", caption="data by
  theme_bw()
```



12.4 box plot

Box plot is also called as box and whisker plot. it shows five point summary to display distribution of data in categories of categorical variable. We have seen boxplot earlier in this session, but we used base R plotting system then. Here we will appropriate above code to gate boxplot just by adding one more variable in ggplot `aes()` as boxplot is bivariate plot and changing the geom to boxplot.

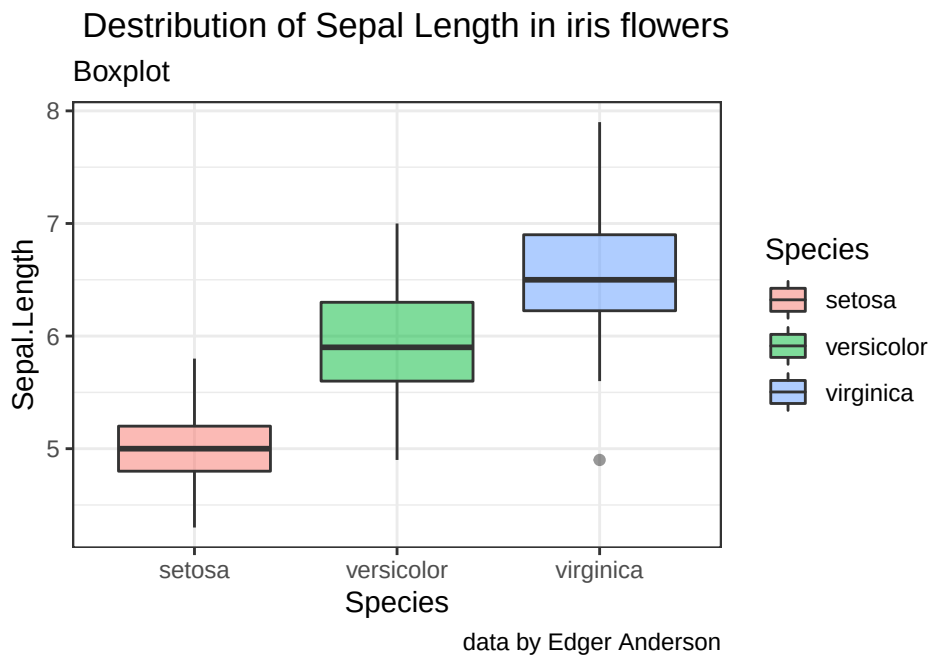
```
ggplot(iris,mapping=aes(x=Species, y=Sepal.Length,fill=Species))+
  geom_boxplot(color="white", alpha=0.5)+
  labs(title=" Destribution of Sepal Length in iris flowers", subtitle="Boxplot", caption="data by Edge
  theme_bw()
```



Note that we have removed the x and y axis titles form the labs() as those are not suitable here in boxplot.

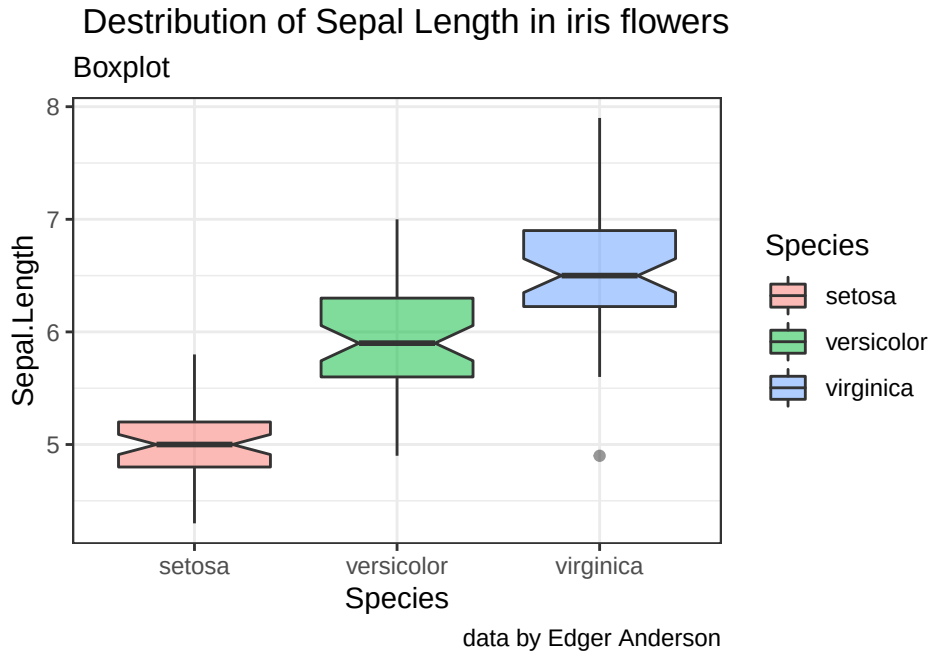
Color white is not good here. We will remove that rom argument of geom_boxplot().

```
ggplot(iris,mapping=aes(x=Species, y=Sepal.Length,fill=Species))+
  geom_boxplot( alpha=0.5)+
  labs(title=" Destribution of Sepal Length in iris flowers", subtitle="Boxplot", caption="data by Edge
  theme_bw()
```



We will add notch to the boxes to make it still ore attractive(in my opinion)

```
ggplot(iris,mapping=aes(x=Species, y=Sepal.Length,fill=Species))+
  geom_boxplot( alpha=0.5,notch = TRUE)+
  labs(title=" Destribution of Sepal Length in iris flowers", subtitle="Boxplot", caption="data by Edge
  theme_bw()
```

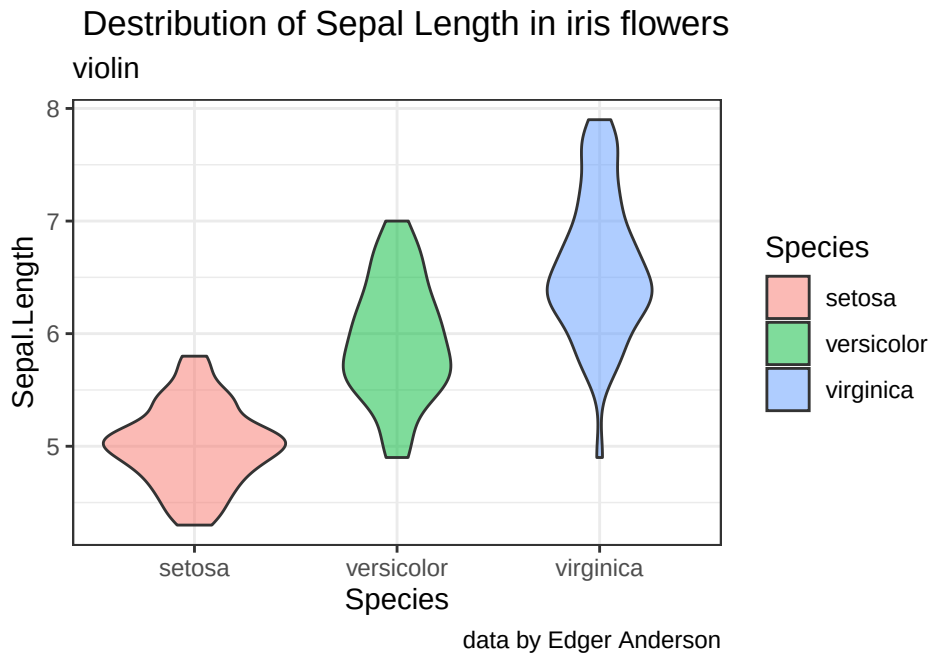


12.5 violin plot

The boxplot code can be appropriated to make violin plot. The notch attribute has no place in violin plot. Violin plot shows distribution of values across the range of those values in that variable. In some cases violin

plot is more appropriate than boxplot.

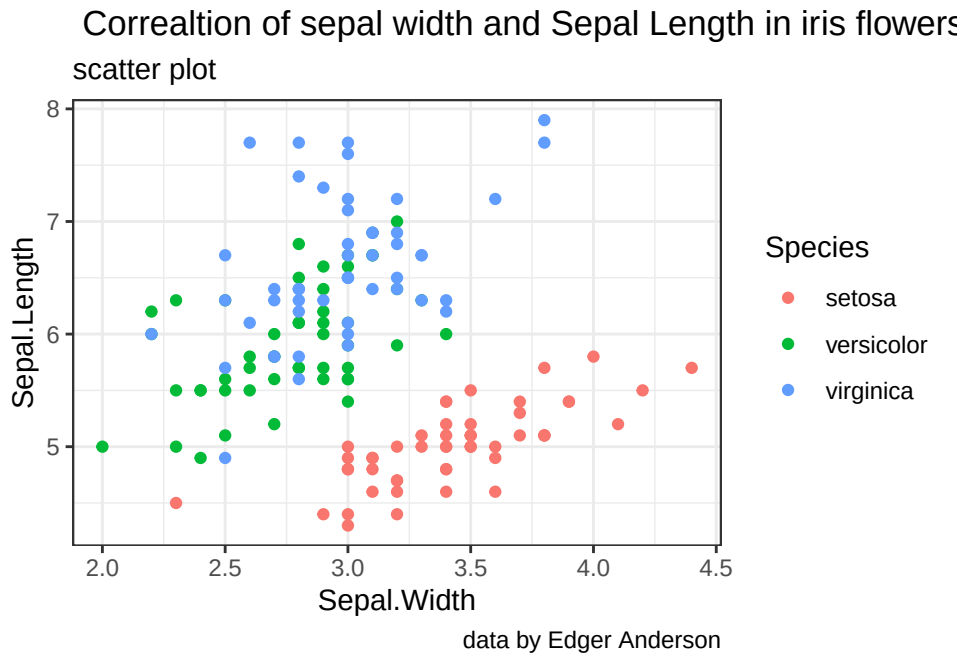
```
ggplot(iris,mapping=aes(x=Species, y=Sepal.Length,fill=Species))+
  geom_violin( alpha=0.5)+
  labs(title=" Destribution of Sepal Length in iris flowers", subtitle="violin", caption="data by Edger
  theme_bw()
```



12.6 Scatter plot

Scatter plot shows correlation between two numeric variables. We will use above code but change the x and y variables in `aes()` of ggplot2 and the change the fill attribute to color as points cant have fill attribute. We will remove alpha also from the geom_point() . We have chnaged the plot tiel also to make it suitanle for correlation plot.

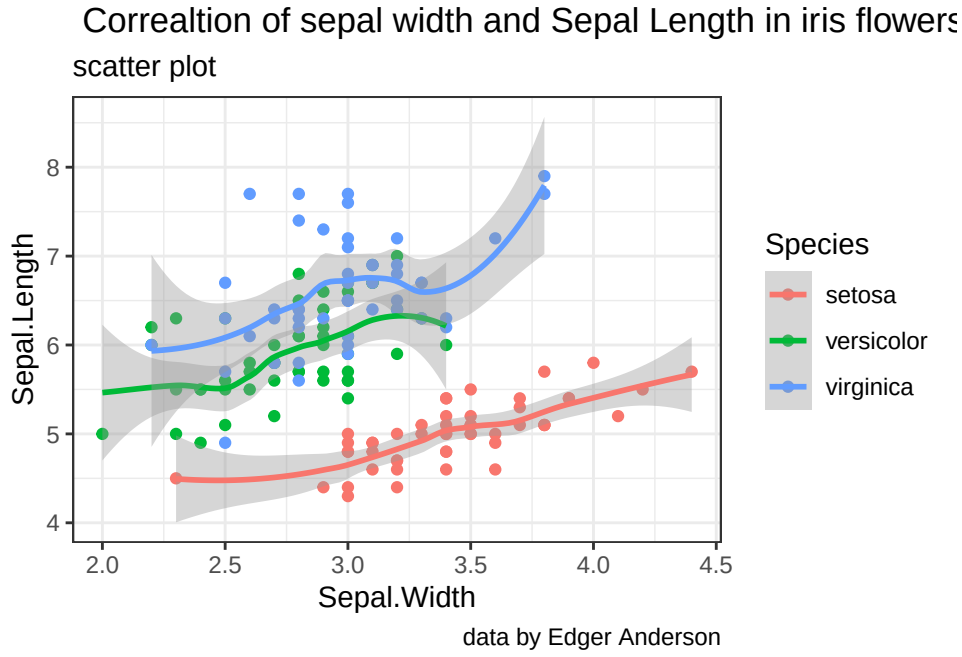
```
ggplot(iris,mapping=aes(x=Sepal.Width, y=Sepal.Length,color=Species))+
  geom_point()+
  labs(title=" Correaltion of sepal width and Sepal Length in iris flowers", subtitle="scatter plot", ca
  theme_bw()
```



Scatter plot can be made more informative if we show the regression line here. As we have shown the three categories in three colors by mapping those to color attribute, ggplot will give us the linear regression lines for each of these categories.

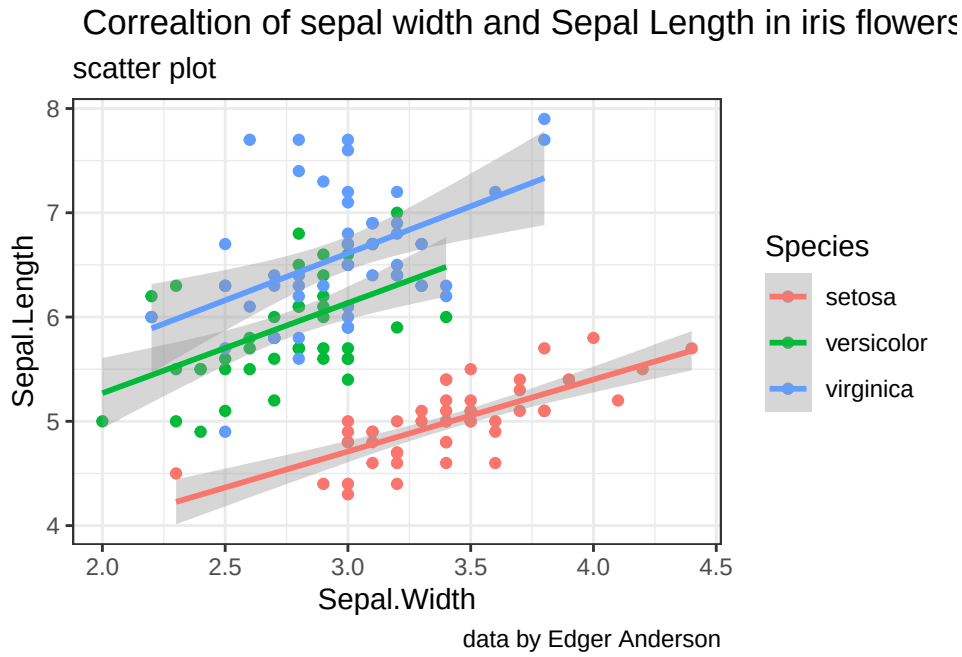
`Geom_smooth()` is used to add the regression line(trend line in Excel terminology)

```
ggplot(iris,mapping=aes(x=Sepal.Width, y=Sepal.Length,color=Species))+
  geom_point()+
  labs(title=" Correaltion of sepal width and Sepal Length in iris flowers", subtitle="scatter plot", ca
  theme_bw()+
  geom_smooth()
```



These lines are plotted using "loess regression". We will make those lines straight by using "lm" method of regression to sjho linear regression lines. For this we will specify `method="lm"`in `geom_smooth()`

```
ggplot(iris,mapping=aes(x=Sepal.Width, y=Sepal.Length,color=Species))+
  geom_point()+
  labs(title=" Correaltion of sepal width and Sepal Length in iris flowers", subtitle="scatter plot", ca
  theme_bw()+
  geom_smooth(method ="lm")
```



12.7 bar plot

Bar plot ranks the categories of a categorical variable according values of numeric variables. The hight of bar signify the value. Bar charts are mostly used to show summary of data. the summary can be mean or median, in most cases it is mean. Different categories can be compared using barplot. There are two types of barplots - stacked barplot where categories are stacked above each other ion a bar and grouped barplot where categories grouped together are shown as groups of the bars.

12.7.1 Get summary for the barplot

We will sue here pivot, group_by and summarise fucntions in pipe to get summary.

```
library(dplyr)
library(tidyr)
iris_summary <- iris %>%
  pivot_longer(cols=1:4,names_to = "flowerPart",values_to = "value") %>%
  group_by(Species,flowerPart) %>%
  summarise(meanLength=mean(value),sdLength=sd(value))
```

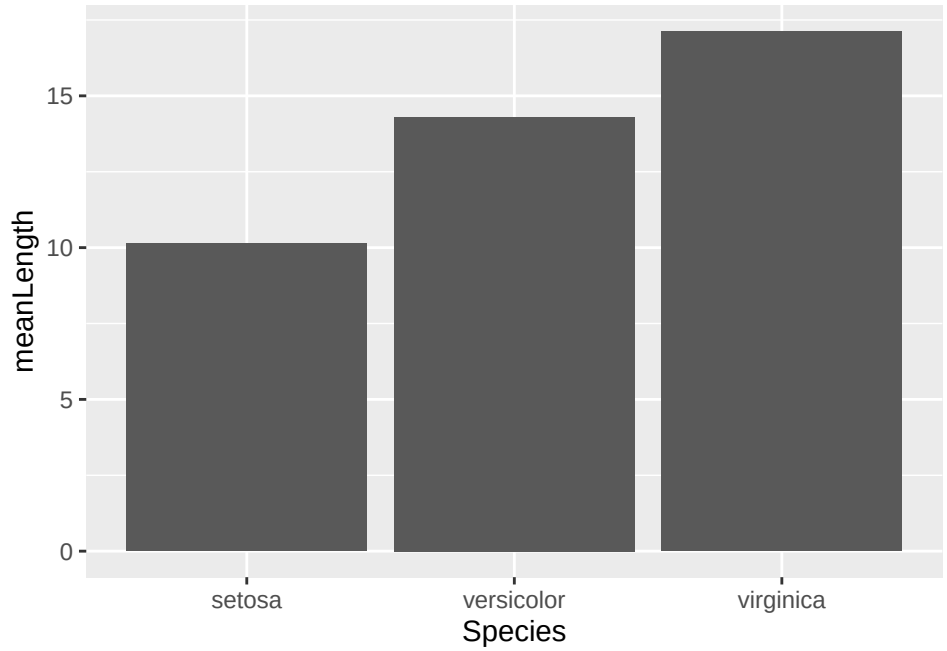
12.8 Build the barplot

And now we will build a barplot- step by step .

In barplot - in the geom_bar(), stat= : identity" has to be specified so that the values of numeric variable mapped to y axis will be used to specify height of bars. If you dont specify the stat=identity, R will show

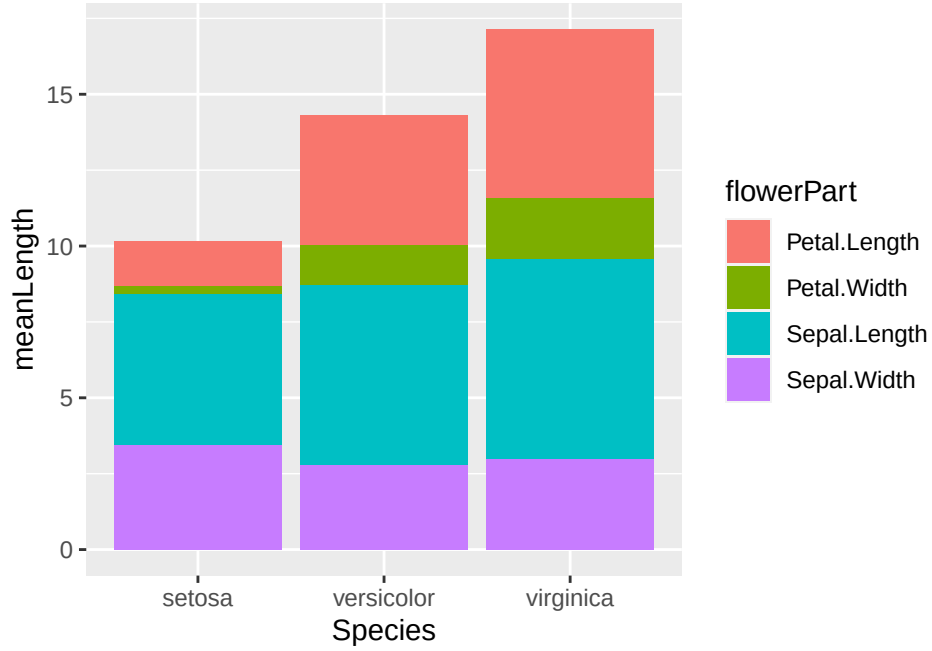
error .

```
ggplot(iris_summary, aes(Species,meanLength))+
  geom_bar(stat="identity")
```



These bars are not showing categories in the individual bars. Fro this we will map variable Flowerpot to fill attribute in aes of `ggplot()` function.

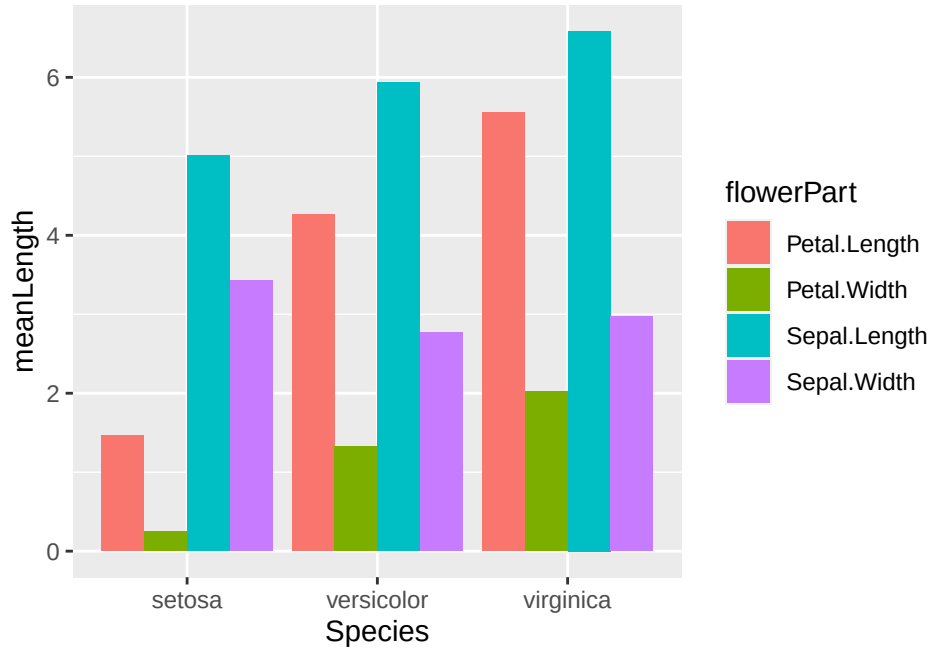
```
ggplot(iris_summary, aes(Species,meanLength,fill=flowerPart))+
  geom_bar(stat="identity")
```



This is stacked barplot. We can make it grouped bar plot by specifying position argument to dodge , in

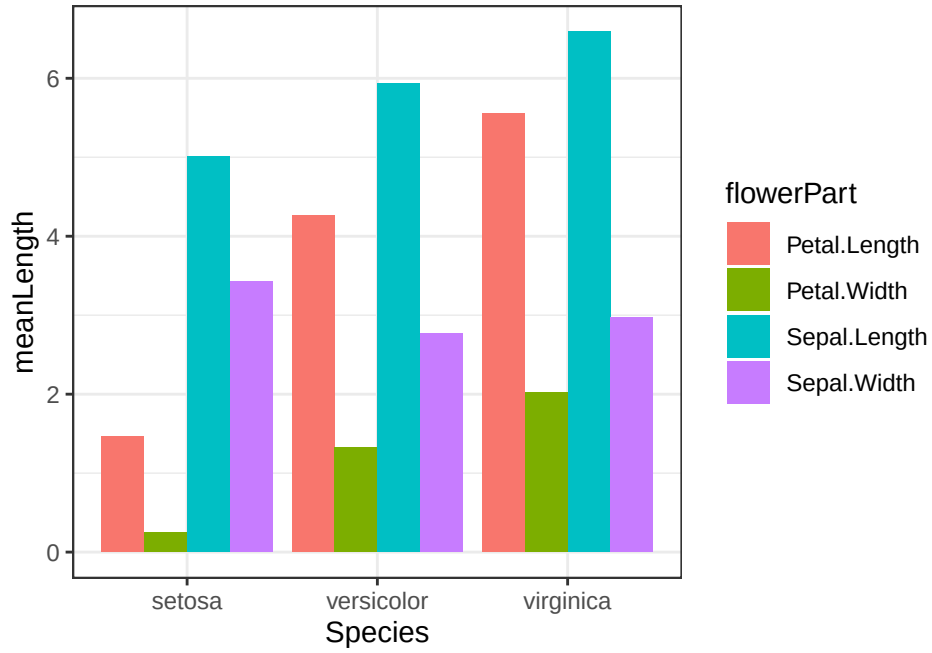
geom_bar()

```
ggplot(iris_summary, aes(Species,meanLength,fill=flowerPart))+
  geom_bar(stat="identity", position="dodge")
```



Making it more appealing by changing the theme.

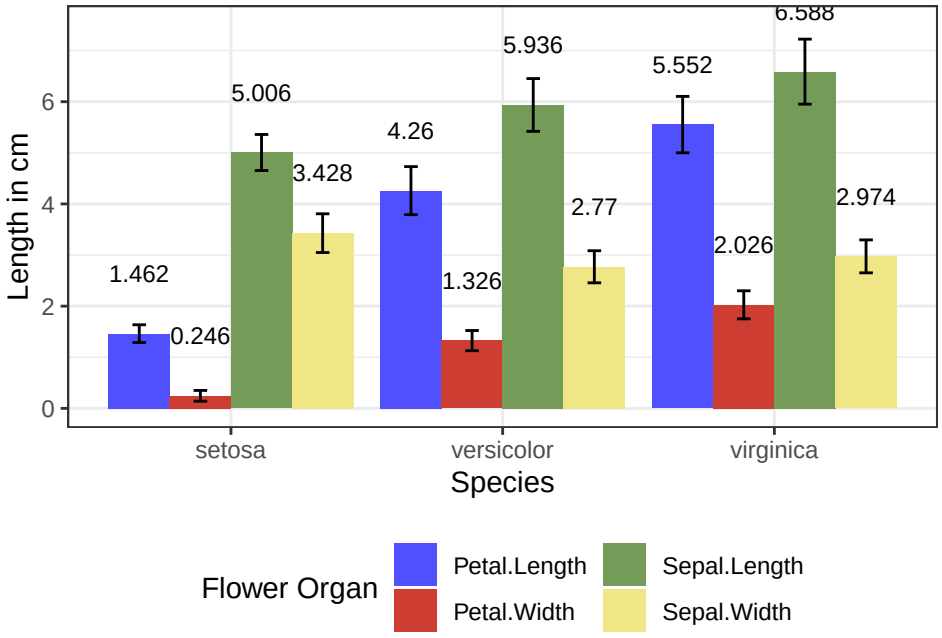
```
ggplot(iris_summary, aes(Species,meanLength,fill=flowerPart))+
  geom_bar(stat="identity", position="dodge")+
  theme_bw()
```



This barplot is still far from being publication ready. The following code gives us publication ready barplot.

But tio understand the code and implement it , you need to learn more of ggplot2.

```
library(ggsci)
iris_bars <- ggplot(iris_summary,aes(reorder(Species,meanLength),meanLength,fill=flowerPart,label=meanL
  geom_bar(stat="identity",position = "dodge")+
  geom_errorbar(aes(ymin=meanLength-sdLength,ymax=meanLength+sdLength),width=0.2,position=position_dodg
  geom_text(vjust=-3,position=position_dodge(0.9),size=3)+
  labs(x="Species",y="Length in cm", fill= "Flower Organ")+
  theme_bw()+
  theme(legend.position = "bottom")+
  guides(fill=guide_legend(ncol=2))+
  scale_fill_igv()+
  ylim(c(0,7.5))



iris_bars
```



R programming is very powerful statistical programming language. We have learnt how to code in R and how to plot graphs in R using ggplot2. Still more is to learn. But this session is just to initiate you in R programming. Now you can learn form all the available sources. You can learn form YouTube videos, from web sites and blog posts and form documentation of R packages.

Now we will just touch to linear regression model.

13 Linear regression and prediction

We will fit linear regression model to women dataset and predict weight of women form given height.

Model requires lm fucntion with formula interface as firts argument. Here we have seen formula above in boxplot and anova test. the tilde sign '~' is used to separate dependent and independent variables.

```
model= lm(weight~height,women)
```

We have fitted linear regression model and now we will see the results of the mdoel fitting.

```
summary(model)
```

```
##
## Call:
## lm(formula = weight ~ height, data = women)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7333 -1.1333 -0.3833  0.7417  3.1167
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -87.51667    5.93694  -14.74 1.71e-09 ***
## height        3.45000    0.09114   37.85 1.09e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.525 on 13 degrees of freedom
## Multiple R-squared:  0.991,  Adjusted R-squared:  0.9903
## F-statistic:  1433 on 1 and 13 DF,  p-value: 1.091e-14
```

In this summary coefficient, adjusted R square and p-value are important for us.

1- coefficient : coefficient of height is the factor by whch weight changes per unit change in height.

2- intercept - it is weight when height is zero. This is little unrealistic here.

3. adjusted R square value: this figure measure how much variation in weight is explained by height. This shall be as close to 1 as posiible.
4. p value : probability of accepting the model even when it is not useful. This value shall be less than 0.05 in most applications. (it can be 0.01 or 0,. as er requiremnts of stringency)

13.1 predict

Prediction formhe fitted model is very easy. Just provide new data to predict fucntion . new data is data frame which contin height as vector. Theis height contain the height values for which weight is to be predicted.

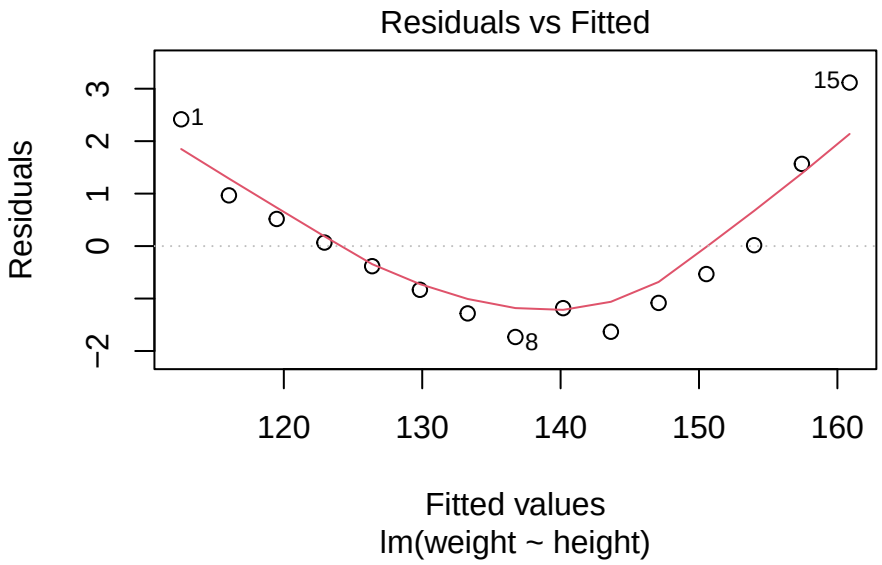
```
newdata= data.frame(height=c(57, 63, 69))

newdata$weight= predict(model, newdata)
newdata
```

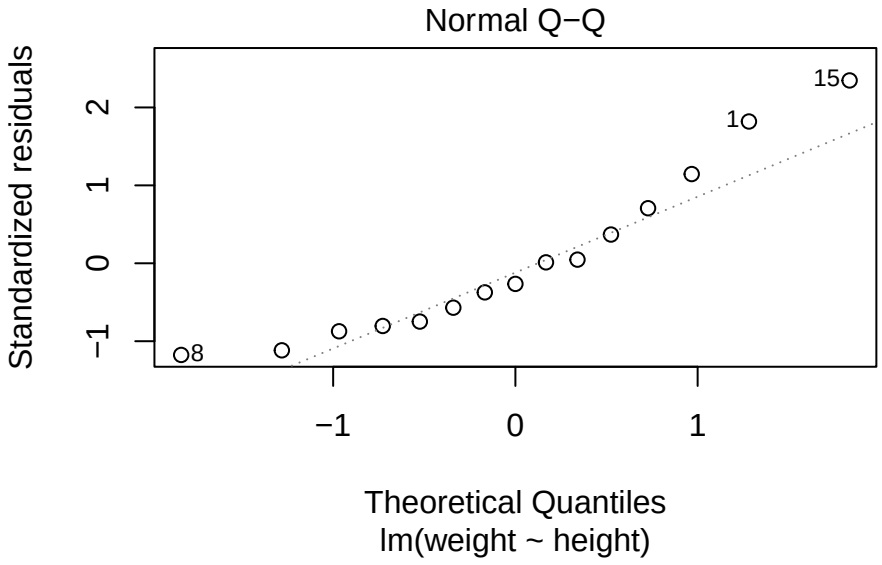
Fitting statistical model, validating those and using those for predcitons is also very easy with R.

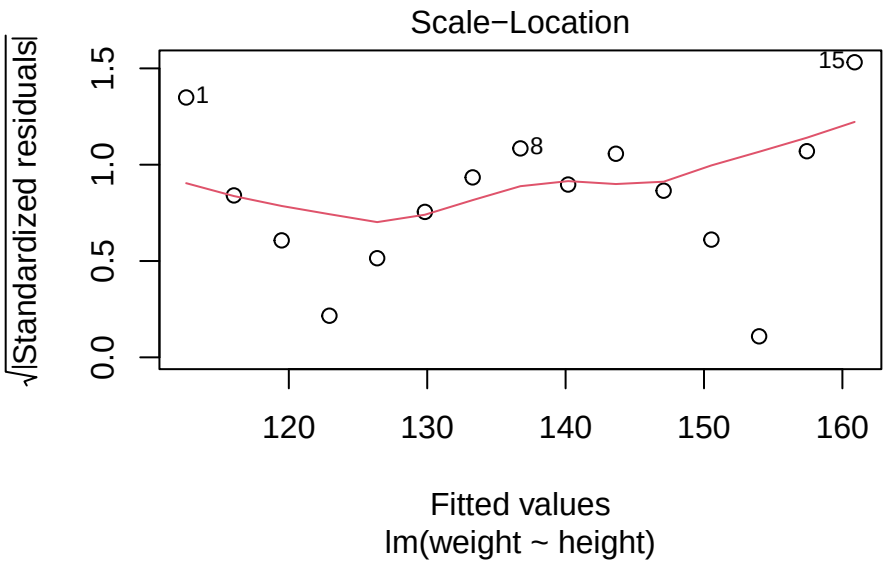
13.2 Diagnostic plots for validating model

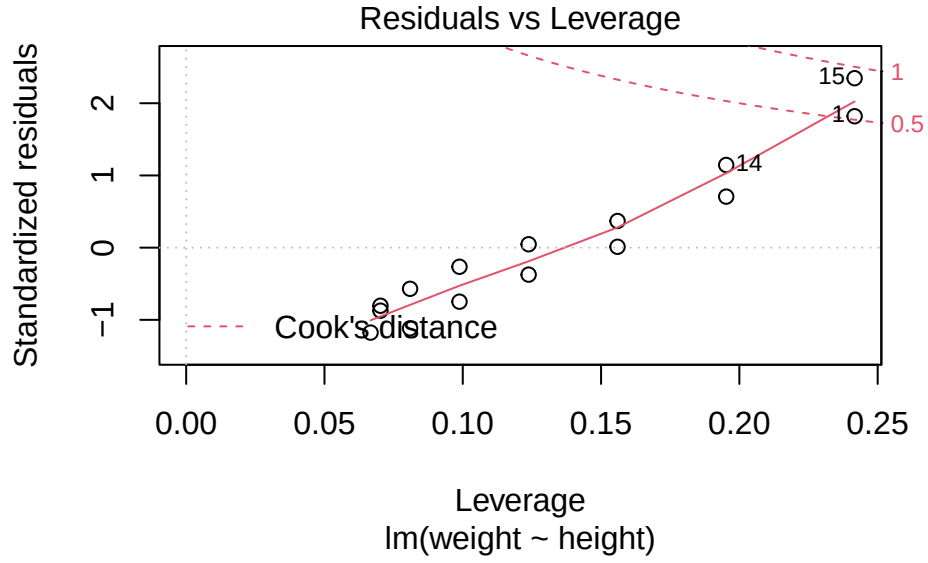
IF you provide the model object to `plot()` function you will get four diagnostic plots to test whether the assumptions while fitting the model hold tru or not.

```
plot(model)
```


Residuals vs Fitted
Residuals
Fitted values
lm(weight ~ height)
1
8
15
−2
0
1
2
3
120
130
140
150
160


Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(weight ~ height)
8
1
15
−1
0
1
2





1. The first plot is residuals vs fitted plot. Residual is the difference between observed and predicted value. This difference shall be randomly distributed along the '0' line. The red ;ine shall be horizontal. This model is not fitted perfectly.
2. The second plot is normal QQ plot. here the points shall be on the diagonal line. This also shows the loose fit of the model. The outliers labeled with values are also there.
3. Scale location plot shall also shall show random distribution across horizontal line. This is not the case here.
4. In the Cooks distance plot , we can see cooks distance lines and one outlier in those lines. This shows that the outliers have significant effect on model, These shall be removed to get better fit.

14 Conclusion

Hope you have got good introduction of R programming and how to code R using Rstudio. We have got introduced to most of the concepts required for analysis of research data. There is still more to learn. And practice is the best teacher.

there are many resources to learn R. Keep up the learning at steady pace.

Thank you.