P.L. :- V

EXPERIMENT NO.:-1

NAME :- AMIT KUMAR

CLASS :- COMP - 2

ROLL NO. : - 150

PRN : - 1714110422

AIM :-

INTRODUCTION TO R PROGRAMMING

THEORY:-

1. Evolution of R programming

R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.

The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.

R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

R is free software distributed under a GNU-style copy left, and an official part of the GNU project called GNU S.

R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.

• A large group of individuals has contributed to R by sending code and bug reports.

• Since mid-1997 there has been a core group (the "R Core Team") who can modify the R source code archive.

2. . R as a powerhouse of Big Data Analysis

R, the open source scripting language was released in 1995 and since then it has grown efficiently and has become a go-to language for the data scientists around the globe. R includes a large number of data packages, shelf graph functions, etc. which proves as a proficient language for big data analytics as it has effective data handling capability. Tech giants like Microsoft, Google are using R for large data analysis. In this article, we list down 6 ways R, the statistical language can be utilised for big data analytics.

1| Data Analysis Exploratory data analysis is a term minted in data analysis using R. This is an approach for data analysis which includes a variety of techniques such as extraction of important variables, test underlying assumptions, maximising insights into the dataset, etc.

2| Data Visualisation R has certain inbuilt plotting commands which makes it easier to create simple graphs. While ggplot2 can be said as one of the most versatile data visualisation package. ggplot2 implements the grammar of graphics which is a coherent system for describing and building graphs. This package allows the user to add, remove or alter components in a plot at a high level of abstraction.

3| Data Wrangling Data Wrangling is the art of getting your data into R in a useful form for visualisation and modelling. It encompasses data transformation and plays a crucial part during a project. It includes basically three main parts, import, tidy and transform.

4| RHIPE RHIPE stands for R and Hadoop Integrated Programming Environment. It is a software package which allows the R user to create MapReduce jobs that work entirely within the R environment using R expressions. The package uses the Divide and Recombine technique to perform data analytics over Big Data. This integration with R is a transformative change to MapReduce as it allows an analyst to quickly specify Maps and Reduces using the full power, flexibility, and expressiveness of the R interpreted language.

5| ORCH ORCH stands for Oracle R Connector for Hadoop is a collection of R packages which provides predictive analytic techniques, written in R or Java as Hadoop MapReduce jobs, that can be applied to data in HDFS files. It also provides interfaces to work with Hive tables, the Apache Hadoop compute infrastructure, the local R environment, and Oracle database tables. There are several analytic algorithms in ORCH such as linear regression, neural networks for prediction, clustering, matrix completion using low-rank matrix factorization, and non-negative matrix factorization.

6| RHadoop RHadoop is an open source collection of five R packages which allows users to manage as well as analyse data with Hadoop from an R environment. It allows data scientists familiar with R to quickly utilize the enterprise-grade capabilities of the MapR Hadoop distribution directly with the analytic capabilities of R. The three packages of RHadoop are as follows

- rhdfs – This package provides basic connectivity to the Hadoop Distributed File System. rmr2 – This package allows R developer to perform statistical analysis in R via Hadoop MapReduce functionality on a Hadoop cluster. rhbase – This package provides basic connectivity to the HBASE distributed database, using the Thrift server.
- plyrmr – This package enables the R user to perform common data manipulation operations, as found in popular packages such as plyr and reshape2, on very large data sets stored on Hadoop.
- ravro – This package adds the ability to read and write avro files from local and HDFS file system and adds an avro input format for rmr2.
- rmr2 – This package allows R developer to perform statistical analysis in R via Hadoop MapReduce functionality on a Hadoop cluster.
- rhbase – This package provides basic connectivity to the HBASE distributed database, using the Thrift server.
- plyrmr – This package enables the R user to perform common data manipulation operations, as found in popular packages such as plyr and reshape2, on very large data sets stored on Hadoop.
- ravro – This package adds the ability to read and write avro files from local and HDFS file system and adds an avro input format for rmr2.

3. CRAN

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.

CRAN task views aim to provide some guidance which packages on CRAN are relevant for tasks related to a certain topic. They give a brief overview of the included packages and can be automatically installed using the ctv package. The views are intended to have a sharp focus so that it is sufficiently clear which packages should be included (or excluded) - and they are not meant to endorse the "best" packages for a given task.

•To automatically install the views, the ctv package needs to be installed, e.g., via install.packages("ctv") and then the views can be installed via install.views or update.views (where the latter only installs those packages are not installed and up-to-date), e.g., ctv::install.views("Econometrics") ctv::update.views("Econometrics")

•The task views are maintained by volunteers. You can help them by suggesting packages that should be included in their task views. The contact e-mail addresses are listed on the individual task view pages.

•For general concerns regarding task views contact the ctv package maintainer.

4. Features of R programming

As stated earlier, R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R −

• R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.

• R has an effective data handling and storage facility,

• R provides a suite of operators for calculations on arrays, lists, vectors and matrices.

• R provides a large, coherent and integrated collection of tools for data analysis.

• R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

As a conclusion, R is world's most widely used statistics programming language. It's the # 1 choice of data scientists and supported by a vibrant and talented community of contributors. R is taught in universities and deployed in mission critical business applications. This tutorial will teach you R programming along with suitable examples in simple and easy steps.

5. Alternatives to R programming

MATLAB

•Classes and Objects: Object-Oriented programming brings great simplicity to the program. In Matlab, a class can be defined as a folder or directory of functions used to work with objects of that class whereas in R classes are available in the form of Packages.

•Installation: R can run on any operating system but Matlab cannot run the same licensed version on the different operating systems unless different operating systems installed on the same physical machine.

•Usage: R is basically used to solve problems related to statistics whereas Matlab is used for other aspects of mathematics such as calculus, plotting graphs, etc. R has many prepackaged programs that help to solve analytical problems, so in the analytics field, R is preferred over Matlab.

•Learning Curve: R can be difficult for those who are new into the programming language as R uses natural programming syntax. Since it is open-source R has a massive amount of open source codes available that might help users to get started. On the contrary, Matlab is easier to learn as it has a lot of toolboxes for most of the functionalities. Since Matlab requires the license amount of code available online is scarce.

•Preferred Language: R is the preferred language for performing data analysis by most of the industries whereas Matlab is mostly used by many Universities.

•Add-on Products: Matlab has a lot of add-on products available to perform various tasks. One of them is Simulink. It is a graphical programming environment for modeling, simulating and analyzing multi-domain systems. R doesn't have any add-on products as such it does have a lot of packages that need to be included in the program.

GNU OCTAVE

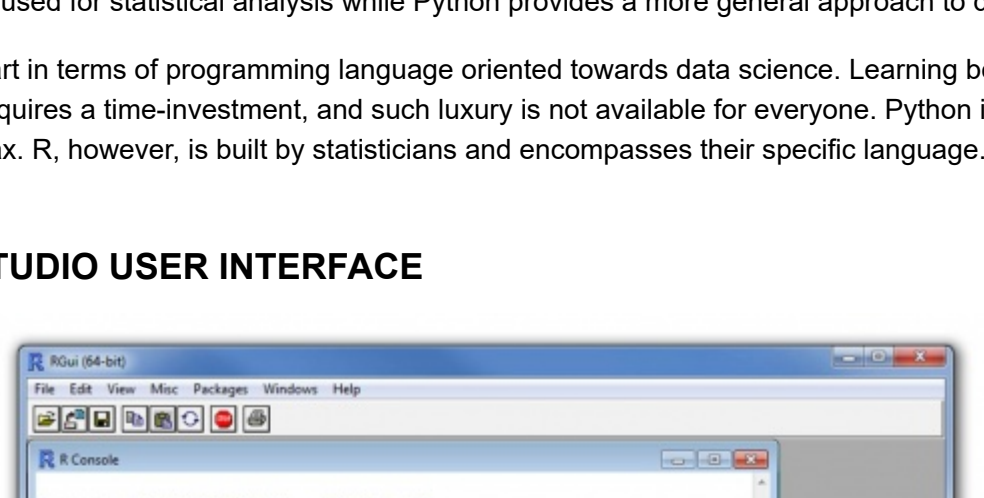
Octave is easier to learn for the reasons other have stated, plus google almost anything in octave and the Matlab help will get you started. Octave is used for machine learning. As if octave best use is machine learning, is overstated (assuming that is what anyone meant). I cannot question that Octave has an advantage for teaching machine learning over R. But octave is good for a lot of things, and based on the octave forge libraries, it is used for sorts of engineering and science, as an open source substitute for Matlab. Which is being replaced in many labs by python. While octave is not open source Matlab. It is very useful to think of it as open source matlab. It is similar enough to make getting started easy. You might think about something else, the popularity of Matlab in my areas seems to be dropping rapidly, it is replace by Python with all those open source libraries. I see a lot of people using Jupiter notebook with python, and I forced to learn it whether or not a like it.

PYTHON

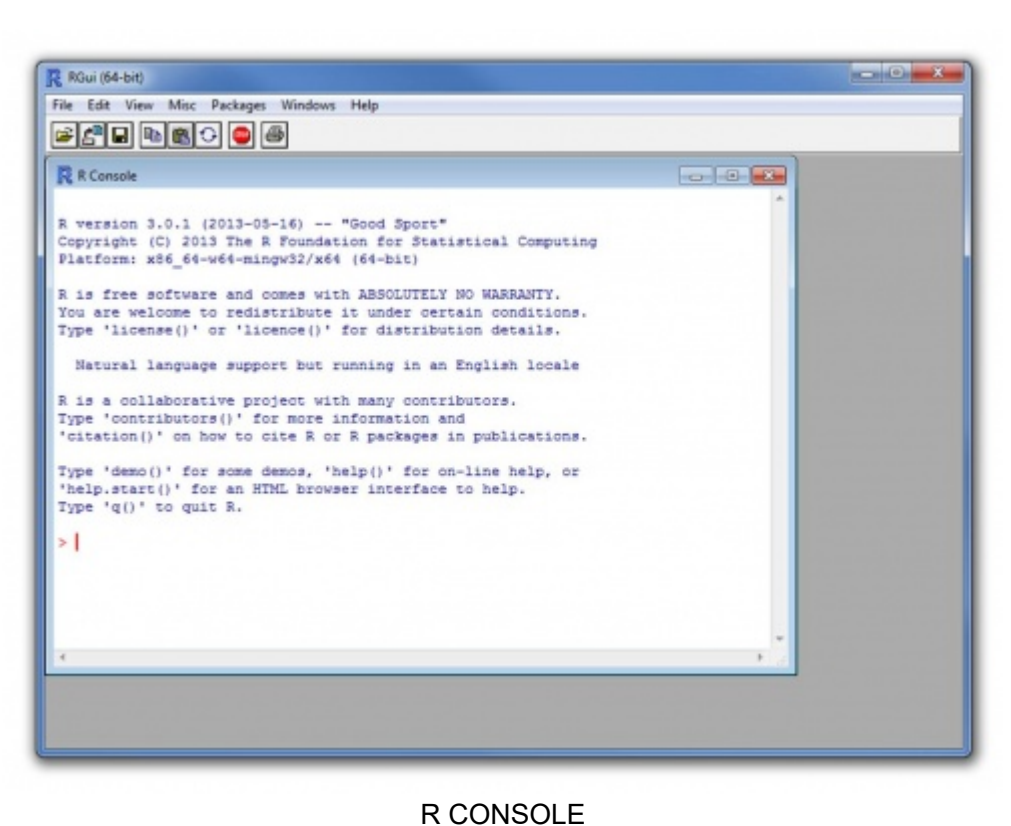
R and Python are both open-source programming languages with a large community. New libraries or tools are added continuously to their respective catalog. R is mainly used for statistical analysis while Python provides a more general approach to data science.

R and Python are state of the art in terms of programming language oriented towards data science. Learning both of them is, of course, the ideal solution. R and Python requires a time-investment, and such luxury is not available for everyone. Python is a general-purpose language with a readable syntax. R, however, is built by statisticians and encompasses their specific language.

R CONSOLE AND R STUDIO USER INTERFACE



R CONSOLE



R STUDIO

Program

Basic Types

```
In [1]: #declaring variable of different classes
#numeric
x <- 20
class(x)
```

'numeric'

```
In [2]: #string
y <- "R is for data science"
class(y)
```

'character'

```
In [3]: #boolean
x <- TRUE
class(x)
```

'logical'

Variables

Variables store values and are an important component in programming, especially for a data scientist. A variable can store a number, an object, a statistical result, vector, dataset, a model prediction basically anything R outputs. We can use that variable later simply by calling the name of the variable.

First way to declare a variable: use the `<-`

name_of_variable <- value

Second way to declare a variable: use the `=`

name_of_variable = value

```
In [4]: x <- 21
x
```

21

```
In [5]: y <- 10
y
```

10

```
In [6]: x-y
```

11

```
In [7]: z = 100
z
```

100

```
In [8]: my.name <- readline(prompt="Enter name: ")
my.roll_no <- readline(prompt="Enter roll no: ")
# convert character into integer
my.roll_no <- as.integer(my.roll_no)
print(paste(my.name,my.roll_no))
```

```
Enter name: Amit Kumar
Enter roll no: 150
[1] "Amit Kumar 150"
```

Arithmatic Operators

We will first see the basic arithmetic operations in R. The following operators stand for:

Operator

Description

Addition ---------------------- +

Subtraction --------------------- -

Multiplication ------------------ *

Division -------------------------/

Exponentiation-----^ or **

```
In [9]: # Addition
3 + 4
```

7

```
In [10]: #multiplication
3 * 5
```

15

```
In [11]: #divison
(5+15)/2
```

10

```
In [12]: #exponentiation
2 ^ 9
```

512

```
In [13]: #Modulo
28%%5
```

3

```
In [14]: # addition with variables
my.no1 <- readline(prompt="Enter num 1: ")
my.no2 <- readline(prompt="Enter num 2 ")
# convert character into integer
my.no1 <- as.integer(my.no1)
my.no2 <- as.integer(my.no2)
add = my.no1 + my.no2
print(paste("Addition ", add))
```

```
Enter num 1: 99999
Enter num 2 999
[1] "Addition  100998"
```

```
In [15]: # subtraction with with variables
my.no1 <- readline(prompt="Enter num 1: ")
my.no2 <- readline(prompt="Enter num 2 ")
# convert character into integer
my.no1 <- as.integer(my.no1)
my.no2 <- as.integer(my.no2)
sub = my.no1 - my.no2
print(paste("Subtraction ", sub))
```

```
Enter num 1: 110000
Enter num 2 11
[1] "Subtraction  109989"
```

```
In [16]: # Multiplication with with variables
my.no1 <- readline(prompt="Enter num 1: ")
my.no2 <- readline(prompt="Enter num 2: ")
# convert character into integer
my.no1 <- as.integer(my.no1)
my.no2 <- as.integer(my.no2)
mul = my.no1 * my.no2
print(paste("Multiplication: ", mul))
```

```
Enter num 1: 999
Enter num 2: 11
[1] "Multiplication:  10989"
```

```
In [17]: # Division with with variables
my.no1 <- readline(prompt="Enter num 1: ")
my.no2 <- readline(prompt="Enter num 2 ")
# convert character into integer
my.no1 <- as.integer(my.no1)
my.no2 <- as.integer(my.no2)
div = my.no1 / my.no2
print(paste("Division ", div))
```

```
Enter num 1: 999
Enter num 2 3
[1] "Division  333"
```

Logical Operators

With logical operators, we want to return values inside the vector based on logical conditions. Following is a detailed list of logical operators available in R

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x	y
x & y	x AND y
isTRUE(x)	Test if X is TRUE

The logical statements in R are wrapped inside the []. We can add many conditional statements as we like but we need to include them in a parenthesis. We can follow this structure to create a conditional statement:

variable_name[(conditional_statement)]

With variable_name referring to the variable, we want to use for the statement. We create the logical statement i.e. variable_name > 0. Finally, we use the square bracket to finalize the logical statement. Below, an example of a logical statement.

```
In [18]: logical_vector <- c(1:10)
logical_vector>5
```

FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE

```
In [19]: logical_vector[logical_vector>5]
```

6 7 8 9 10

```
In [20]: logical_vector <- c(1:10)
logical_vector[(logical_vector>2) & (logical_vector<8)]
```

3 4 5 6 7

```
In [21]: #loading a pre built daataset
dataset <- mtcars
class(dataset$mpg)
```

'numeric'

```
In [22]: dataset
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
In [23]: #loading a csv from network
PATH <-'https://raw.githubusercontent.com/guru99-edu/R-Programming/master/prison.csv'
df <- read.csv(PATH)[1:5]
head(df, 5)
```

x	state	year	govelec	black
1	1	80	0	0.2560
2	1	81	0	0.2557
3	1	82	1	0.2554
4	1	83	0	0.2551
5	1	84	0	0.2548

```
In [24]: # Structure of the data
str(df)
```

```
'data.frame':   714 obs. of  5 variables:
 $ x      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ state  : int  1 1 1 1 1 1 1 1 1 1 ...
 $ year   : int  80 81 82 83 84 85 86 87 88 89 ...
 $ govelec: int  0 0 1 0 0 0 1 0 0 0 ...
 $ black  : num  0.256 0.256 0.255 0.255 0.255 ...
```

IF, ELSE, ELSE IF Statement in R

The if else statement

An if-else statement is a great tool for the developer trying to return an output based on a condition. In R, the syntax is:

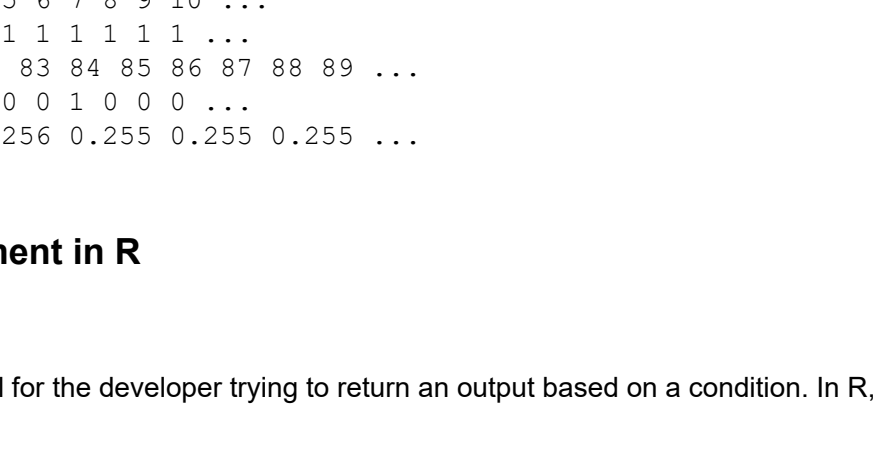
if (condition) {

```
Expr1
```

} else {

```
Expr2
```

}




```
In [25]: # Create vector quantity
quantity <-  25
# Set the is-else statement
if (quantity > 20) {
    print('You sold a lot!')
} else {
    print('Not enough for today')
}
```

```
[1] "You sold a lot!"
```

The else if statement

We can further customize the control level with the else if statement. With elif, you can add as many conditions as we want. The syntax is:

if (condition1) {

```
expr1

} else if (condition2) {

expr2

} else if  (condition3) {

expr3

} else {

expr4
```

}

```
In [26]: # Create vector quantiy
quantity <-  10
# Create multiple condition statement
if (quantity <20) {
      print('Not enough for today')
} else if (quantity > 20  &quantity <= 30) {
     print('Average day')
} else {
      print('What a great day!')
}
```

```
[1] "Not enough for today"
```

Functions in R Programming

A function, in a programming environment, is a set of instructions. A programmer builds a function to avoid repeating the same task, or reduce complexity.

A function should be

- written to carry out a specified a tasks
- may or may not include arguments
- contain a body
- may or may not return one or more values.

A general approach to a function is to use the argument part as inputs, feed the body part and finally return an output. The Syntax of a function is the following:

function (arglist) {

Function body

}

R important built-in functions

There are a lot of built-in function in R. R matches your input parameters with its function arguments, either by value or by position, then executes the function body. Function arguments can have default values: if you do not specify these arguments, R will take the default value.

Note: It is possible to see the source code of a function by running the name of the function itself in the console.

We will see three groups of function in action

- General function
- Maths function
- Statistical function

General functions

We are already familiar with general functions like cbind(), rbind(),range(),sort(),order() functions. Each of these functions has a specific task, takes arguments to return an output. Following are important functions one must know-

```
In [27]: dt <- cars
## number columns
length(dt)
```

2

```
In [28]: length(dt[,2])
```

50

Maths functions

R has an array of mathematical functions.

Operator Description

abs(x)---------Takes the absolute value of x

log(x,base=y)--Takes the logarithm of x with base y; if base is not specified, returns the natural logarithm

exp(x)---------Returns the exponential of x

sqrt(x)--------Returns the square root of x

factorial(x)---Returns the factorial of x (x!)

```
In [29]: # sequence of number from 44 to 55 both including incremented by 1
x_vector <- seq(45,55, by = 1)
#logarithm
log(x_vector)
```

3.80666248977032 3.8286413964891 3.85014760171006 3.87120101090789 3.89182029811063 3.91202300542815 3.93182563272433 3.95124371858143 3.97029191355212 3.98898404656427 4.00733318523247

```
In [30]: #exponential
exp(x_vector)
```

34934271057485094912 94961194206024482826 2.58131288619007e+20 7.01673591209763e+20 1.9073465724951e+21 5.18470552858707e+21 1.40934908242694e+22 3.83100800071658e+22 1.04137594330291e+23 2.83075330327469e+23 7.69478526514202e+23

```
In [31]: #squared root
sqrt(x_vector)
```

6.70820393249937 6.78232998312527 6.85565460040104 6.92820323027551 7 7.07106781186548 7.14142842854285 7.21110255092798 7.28010988928052 7.34846922834953 7.41619848709566

```
In [32]: #factorial
factorial(x_vector)
```

1.1962222086548e+56 5.50262215981209e+57 2.58623241511168e+59 1.24139155925361e+61 6.08281864034268e+62 3.0414093201713e+64 1.55111875328741e+66 8.06581751709449e+67 4.27488328406012e+69 2.30843697339245e+71 1.26964033536586e+73

Statistical functions

R standard installation contains wide range of statistical functions. In this tutorial, we will briefly look at the most important function..

Operator-------------------------------------Description

mean(x)--------------------------------------Mean of x

median(x)------------------------------------Median of x

var(x)--Variance of x

sd(x)---Standard deviation of x

scale(x)-------------------------------------Standard scores (z-scores) of x

quantile(x)----------------------------------The quartiles of x

summary(x)-----------------------------------Summary of x: mean, min, max etc..

```
In [33]: speed <- dt$speed
speed
# Mean speed of cars dataset
mean(speed)
```

4 4 7 7 8 9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25

15.4

```
In [34]: # Median speed of cars dataset
median(speed)
```

15

```
In [35]: # Variance speed of cars dataset
var(speed)
```

27.9591836734694

```
In [36]: # Standard deviation speed of cars dataset
sd(speed)
```

5.28764443523478

```
In [37]: # Standardize vector speed of cars dataset
head(scale(speed), 5)
```

-2.155969
-2.155969
-1.588609
-1.588609
-1.399489

```
In [38]: # Quantile speed of cars dataset
quantile(speed)
```

0%	4
25%	12
50%	15
75%	19
100%	25

```
In [39]: summary(speed)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
    4.0    12.0    15.0    15.4    19.0    25.0 
```

Write function in R

In some occasion, we need to write our own function because we have to accomplish a particular task and no ready made function exists. A user-defined function involves a name, arguments and a body.

function.name <- function(arguments)

{

```
computations on the arguments

some other code
```

}

In [40]:
```
square_function<- function(n)
{
  # compute the square of integer `n`
  n^2
}
# calling the function and passing value 4
square_function(4)
```

16

Multi arguments function

We can write a function with more than one argument. Consider the function called "times". It is a straightforward function multiplying two variables.

In [41]:
```
times <- function(x,y) {
  x*y
        }
times(2,4)
```

8

Functions with condition

Sometimes, we need to include conditions into a function to allow the code to return different outputs.

split_data <- function(df, train = TRUE)

Arguments:

-df: Define the dataset

-train: Specify if the function returns the train set or test set. By default, set to TRUE

In [42]:
```
#IMP :- Train Test Split Function
split_data <- function(df, train = TRUE){
  length<- nrow(df)
  total_row <- length *0.8
  split <- 1:total_row
  if (train ==TRUE){
    train_df <- df[split, ]
      return(train_df)
  } else {
    test_df <- df[-split, ]
      return(test_df)
  }
}
```

In [43]:
```
train <- split_data(airquality, train = TRUE)
dim(train)
```

122 6

In [44]:
```
test <- split_data(airquality, train = FALSE)
dim(test)
```

31 6

For Loop in R

A for loop is very valuable when we need to iterate over a list of elements or a range of numbers.Loop can be used to iterate over a list, data frame, vector, matrix or any other object. The braces and square bracket are compulsory.

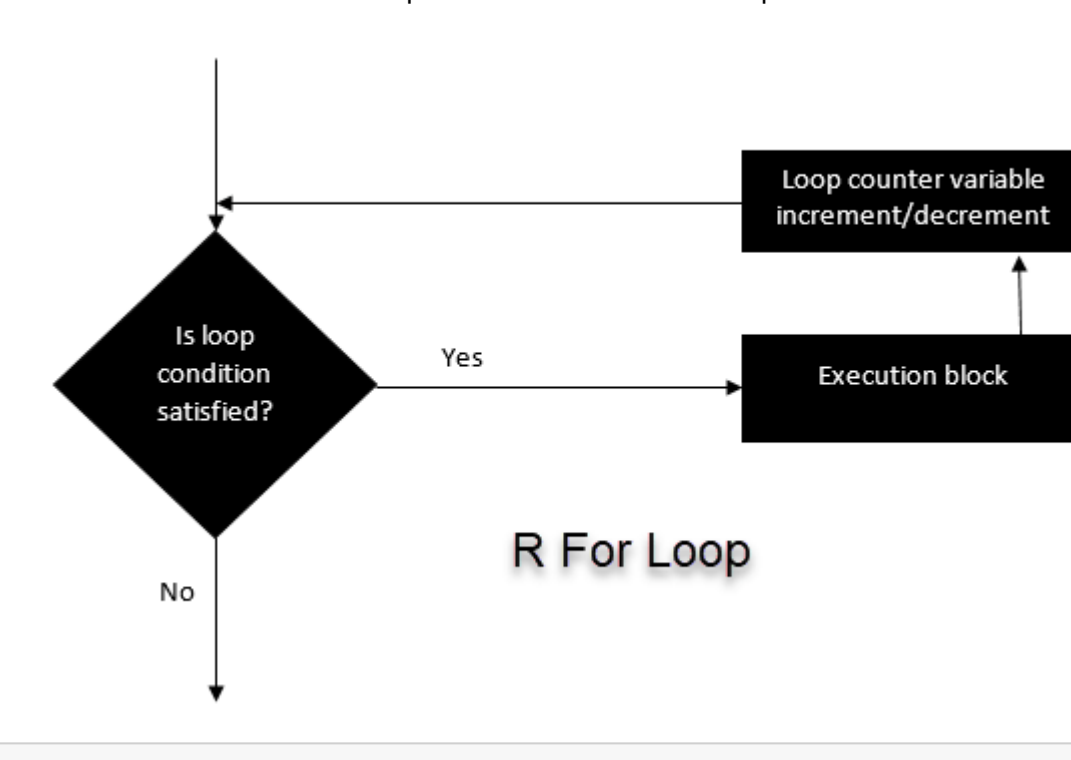
For Loop Syntax and Examples

For (i in vector) {

```
Exp
```

}

R will loop over all the variables in vector and do the computation written inside the exp.



In [45]:
```
# Create fruit vector
fruit <- c('Apple', 'Orange', 'Passion fruit', 'Banana')
# Create the for statement
for ( i in fruit){
 print(i)
}
```

```
[1] "Apple"
[1] "Orange"
[1] "Passion fruit"
[1] "Banana"
```

In [46]:
```
# Create an empty list
list <- c()
# Create a for statement to populate the list
for (i in seq(1, 4, by=1)) {
  list[[i]] <- i*i
}
print(list)
```

```
[1]  1  4  9 16
```

For Loop over a list

Looping over a list is just as easy and convenient as looping over a vector. Let's see an example

In [47]:
```
# Create a list with three vectors
fruit <- list(Basket = c('Apple','Orange','Passion fruit', 'Banana'),
Money = c(10, 12, 15), purchase = FALSE)
for (p  in fruit)
{
        print(p)
}
```

```
[1] "Apple"         "Orange"        "Passion fruit" "Banana"
[1] 10 12 15
[1] FALSE
```

For Loop over a matrix

A matrix has 2-dimension, rows and columns. To iterate over a matrix, we have to define two for loop, namely one for the rows and another for the column.

In [48]:
```
# Create a matrix
mat <- matrix(data = seq(10, 20, by=1), nrow = 6, ncol =2)
# Create the loop with r and c to iterate over the matrix
for (r in 1:nrow(mat))
    for (c in 1:ncol(mat))
         print(paste("Row", r, "and column",c, "have values of", mat[r,c]))
```

```
Warning message in matrix(data = seq(10, 20, by = 1), nrow = 6, ncol = 2):
"data length [11] is not a sub-multiple or multiple of the number of rows [6]"
```

```
[1] "Row 1 and column 1 have values of 10"
[1] "Row 1 and column 2 have values of 16"
[1] "Row 2 and column 1 have values of 11"
[1] "Row 2 and column 2 have values of 17"
[1] "Row 3 and column 1 have values of 12"
[1] "Row 3 and column 2 have values of 18"
[1] "Row 4 and column 1 have values of 13"
[1] "Row 4 and column 2 have values of 19"
[1] "Row 5 and column 1 have values of 14"
[1] "Row 5 and column 2 have values of 20"
[1] "Row 6 and column 1 have values of 15"
[1] "Row 6 and column 2 have values of 10"
```

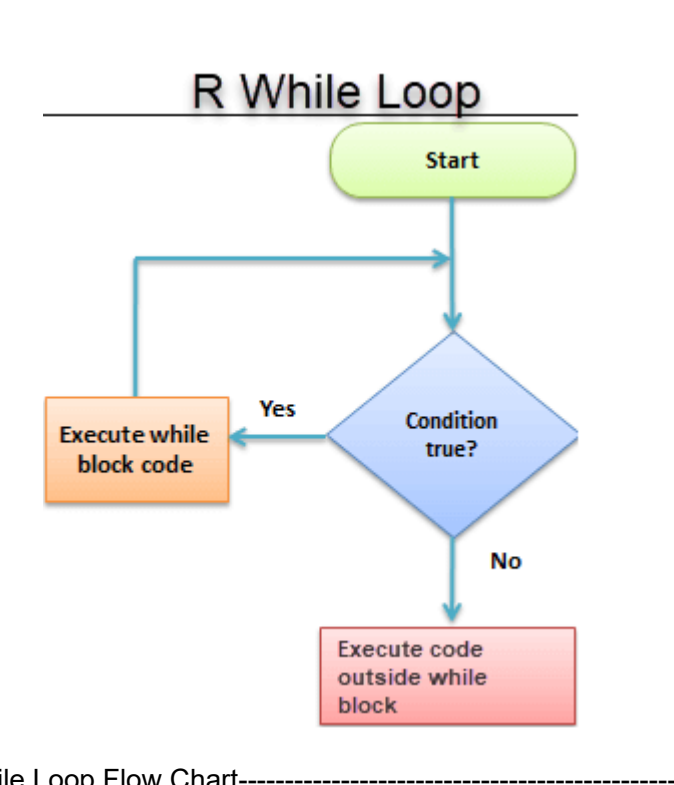
While Loop in R

A loop is a statement that keeps running until a condition is satisfied. The syntax for a while loop is the following:

while (condition) {

```
Exp
```

}



---While Loop Flow Chart---

Note: Remember to write a closing condition at some point otherwise the loop will go on indefinitely.

In [49]:
```
#Create a variable with value 1
begin <- 1

#Create the loop
while (begin <= 10){

#See which we are
cat('This is loop number',begin)

#add 1 to the variable begin after each loop
begin <- begin+1
print(begin)
}
```

```
This is loop number 1[1] 2
This is loop number 2[1] 3
This is loop number 3[1] 4
This is loop number 4[1] 5
This is loop number 5[1] 6
This is loop number 6[1] 7
This is loop number 7[1] 8
This is loop number 8[1] 9
This is loop number 9[1] 10
This is loop number 10[1] 11
```

Example:- You bought a stock at price of 50 dollars. If the price goes below 45, we want to short it. Otherwise, we keep it in our portfolio. The price can fluctuate between -10 to +10 around 50 after each loop. You can write the code as follow:

In [50]:
```
set.seed(123)
# Set variable stock and price
stock <- 50
price <- 50

# Loop variable counts the number of loops
loop <- 1

# Set the while statement
while (price > 45){

# Create a random price between 40 and 60
price <- stock + sample(-10:10, 1)

# Count the number of loop
loop = loop +1

# Print the number of loop
print(loop)
}
```

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

In [ ]: