SSBT's College of Engineering & Technology, Bambhori, Jalgaon -425001.

Included under section 2(f) & 12(B) of the UGC Act, 1956
With NAAC Accredited courses & ISO 9001: 2015
Post Box No. 94, Phone: 0257-2258393, 94 (Fax: 0257-2258392)
Grade B ++ (2.91) NAAC Accredited
E-mail:sscoetjal@gmail.com
Website: www.sscoetjalgaon.ac.in

ISO 9000:2015

Department of Master *in Computer Application*

Laboratory Journal

MCA(Second Year)

Lab on Data Analytics

Academic Year- 20 - 20

Name of student: ……..
Section:……..
Roll no:……..
University Exam Seat No.: ……..

SSBT's COLLEGE OF ENGINEERING & TECHNOLOGY, BAMBHORI, JALGAON -425001

Year: 20 -20

Department of Master in Computer Application

Vision of the Institute

Today we carry the flame of quality education, knowledge and progressive technology for global societal development; tomorrow the flame will glow even brighter.

Mission of the Institute

To provide conducive environment for preparing competent, value added and patriotic engineers of integrity of par excellence to meet global standards for societal development.

Vision of the Department

To emerge as the leading Computer Engineering department for inclusive development of students.

Mission of the Department

To provide student-centered conducive environment for preparing knowledgeable, competent and value-added computer engineers.

Objectives:

- To impart innovative teaching and learning
- To provide quality education with futuristic trends in engineering and technology
- To develop the institute as a research center for academic excellence
- To ensure continual improvement in quality management system
- To inculcate social values, patriotism and professional ethics among the students.

SSBT's COLLEGE OF ENGINEERING & TECHNOLOGY, BAMBHORI, JALGAON -425001
Year: 20 -20

Department of Master in Computer Application

Program Educational Objectives of Computer Engineering Dept. (PEOs)

PEO 1. Core Knowledge -Computer engineering graduates will have the knowledge of basic science and Engineering skills, Humanities, social science, management and conceptual and practical understanding of core computer engineering area with project development.

PEO 2. Employment/ Continuing Education - Computer engineering graduates will have the knowledge of Industry-based technical skills to succeed in entry level engineering position at various industries as well as in academics.

PEO 3. Professional Competency - Computer engineering graduates will have the ability to communicate effectively in English, to accumulate and disseminate the knowledge and to work effectively in a team with a sense of social awareness.

Program outcomes of Computer Engineering Dept. (POs)

Engineering Graduates will be able to:

- **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

- **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific outcomes (PSOs)

Computer Engineering Graduates will be able to::
Software Systems Development: Apply the theoretical concepts of computer engineering and practical knowledge in analysis, design and development of software systems.
Open Source Software: Demonstrate familiarity and practical competence with a broad range of programming languages and open source platforms
Computer Proficiency: Exhibit proficiency through latest technologies in demonstrating the ability for work efficacy to the industry & society

Lab on Data Analytics

Course Title: Lab on Data Analytics

Short Title: CA LAB-XI(C)

Course Objectives:

1) Learn Data Science concepts of R and functioning of R
2) Understand Exploratory Data Analytics
3) Learn to program various analysis techniques

Course Outcomes (Lab on Data Analytics)

After completion of this course students shall be able to

1) Develop code using R programming constructs.
2) Manipulate data using R.
3) Write code for various data analysis techniques.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Experiment No: 1

Aim: To perform the basic mathematical operations in r programming

Theory:

What is R

R is a popular programming language used for statistical computing and graphical presentation. Its most common use is to analyze and visualize data.

"R is an interpreted computer programming language which was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand."

The *R Development Core Team* currently develops R. It is also a software environment used to analyze **statistical information**, **graphical representation**, **reporting**, and **data modeling**.

R is the implementation of the **S programming** language, which is combined with **lexical scoping semantics**.

Features of R programming

R is a domain-specific programming language which aims to do data analysis. It has some unique features which make it very powerful. The most important arguably being the notation of vectors. These vectors allow us to perform a complex operation on a set of values in a single command. There are the following features of R programming:

1. It is a simple and effective programming language which has been well developed.
2. It is data analysis software.
3. It is a well-designed, easy, and effective language which has the concepts of user-defined, looping, conditional, and various I/O facilities.
4. It has a consistent and incorporated set of tools which are used for data analysis.
5. For different types of calculation on arrays, lists and vectors, R contains a suite of operators.
6. It provides effective data handling and storage facility.
7. It is an open-source, powerful, and highly extensible software.
8. It provides highly extensible graphical techniques.
9. It allows us to perform multiple calculations using vectors.
10. R is an interpreted language.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Why Use R?

- It is a great resource for data analysis, data visualization, data science and machine learning
- It provides many statistical techniques (such as statistical tests, classification, clustering and data reduction)
- It is easy to draw graphs in R, like pie charts, histograms, box plot, scatter plot, etc++
- It works on different platforms (Windows, Mac, Linux)
- It is open-source and free
- It has a large community support
- It has many packages (libraries of functions) that can be used to solve different problems

Learn basic Command of R Programming

Assignment

The first operator you'll run into is the assignment operator. The assignment operator is used to assign a value. For instance we can assign the value 3 to the variable x using the <- assignment operator.

```
# assignment
x <- 3
```

Interestingly, R actually allows for five assignment operators:

```
# leftward assignment
x <- value


x = value
x <<- value
# rightward assignment
value -> x
value ->> x
```

The original assignment operator in R was <- and has continued to be the preferred among R users. The = assignment operator was added in 2001 primarily because it is the accepted assignment operator in many other languages and beginners to R coming from other languages were so prone to use it.

The operators <<- is normally only used in functions which we will not get into the details.

Evaluation

We can then evaluate the variable by simply typing x at the command line which will return the value of x. Note that prior to the value returned you'll see ## [1] in the command line. This simply implies that the output returned is the first output. Note that you can type any comments in your code by preceding the comment with the hash tag (#) symbol. Any values, symbols, and texts following # will not be evaluated.

```
# evaluation
x
## [1] 3
```

Case Sensitivity

Lastly, note that R is a case sensitive programming language. Meaning all variables, functions, and objects must be called by their exact spelling:

```
x <- 1
y <- 3
z <- 4
x * y * z
## [1] 12
x * Y * z
## Error in eval(expr, envir, enclos): object 'Y' not found
```

Basic Arithmetic

At its most basic function R can be used as a calculator. When applying basic arithmetic, the PEMDAS order of operations applies: parentheses first followed by exponentiation, multiplication and division, and final addition and subtraction.

```
8 + 9 / 5 ^ 2
## [1] 8.36

8 + 9 / (5 ^ 2)
## [1] 8.36
8 + (9 / 5) ^ 2


## [1] 11.24
(8 + 9) / 5 ^ 2
## [1] 0.68
```

By default R will display seven digits but this can be changed using options() as previously outlined.

```
1 / 7
## [1] 0.1428571
options(digits = 3)
1 / 7
## [1] 0.143
pi
## [1] 3.141592654
options(digits = 22)
pi
## [1] 3.141592653589793115998
```

We can also perform integer divide (%/%) and modulo (%%) functions. The integer divide function will give the integer part of a fraction while the modulo will provide the remainder.

```
42 / 4          # regular division
```

```
## [1] 10.5
42 %/% 4       # integer division
## [1] 10
42 %% 4        # modulo (remainder)
## [1] 2
```

Miscellaneous Mathematical Functions

There are many built-in functions to be aware of. These include but are not limited to the following. Go ahead and run this code in your console.

```
x <- 10
abs(x)     # absolute value
sqrt(x)    # square root
exp(x)     # exponential transformation
log(x)     # logarithmic transformation
cos(x)     # cosine and other trigonometric functions
```

Infinite, and NaN Numbers:

When performing undefined calculations, R will produce Inf (infinity) and NaN (not a number) outputs.

```
1 / 0          # infinity
## [1] Inf
Inf - Inf      # infinity minus infinity
## [1] NaN
```

The workspace environment will also list your user defined objects such as vectors, matrices, data frames, lists, and functions. For example, if you type the following in your console:

```
x <- 2
y <- 3
```

You will now see x and y listed in your workspace environment. To identify or remove the objects (i.e. vectors, data frames, user defined functions, etc.) in your current R environment:

```
# list all objects
ls()

# identify if an R object with a given name is present
exists("x")

# remove defined object from the environment
rm(x)

# you can remove multiple objects
rm(x, y)
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
# basically removes everything in the working environment -- use with caution!
rm(list = ls())
```

Conclusion: In this way we had understand the basics of r programming.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Experiment No: 2

Aim: Write program for Creating and Manipulating R Objects in R – Vectors, Matrices, Arrays, Data Frames and Lists.

Theory:

With R, it's Important that one understand that there is a difference between the actual R object and the manner in which that R object is printed to the console. Often, the printed output may have additional bells and whistles to make the output more friendly to the users. However, these bells and whistles are not inherently part of the object

R has five basic or "atomic" classes of objects:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

The most basic type of R object is a vector. Empty vectors can be created with the vector() function. There is really only one rule about vectors in R, which is that A vector can only contain objects of the same class. But of course, like any good rule, there is an exception, which is a list, which we will get to a bit later. A list is represented as a vector but can contain objects of different classes. Indeed, that's usually why we use them.

There is also a class for "raw" objects, but they are not commonly used directly in data analysis

Creating Vectors

The c() function can be used to create vectors of objects by concatenating things together.

```
> x <- c(0.5, 0.6) ## numeric

> x <- c(TRUE, FALSE) ## logical

> x <- c(T, F) ## logical

> x <- c("a", "b", "c") ## character

> x <- 9:29 ## integer
```

```
> x <- c(1+0i, 2+4i) ## complex
```

Note that in the above example, T and F are short-hand ways to specify TRUE and FALSE. However, in general one should try to use the explicit TRUE and FALSE values when indicating logical values. The T and F values are primarily there for when you're feeling lazy.

You can also use the vector() function to initialize vectors.

```
> x <- vector("numeric", length = 10)

> x

[1] 0 0 0 0 0 0 0 0 0 0
```

A vector is an object that contains a set of values called its elements.

Numeric vector

```
x <- c(1,2,3,4,5,6)
```

The operator <– is equivalent to "=" sign.

Character vector

```
State <- c("DL", "MU", "NY", "DL", "NY", "MU")
```

To calculate frequency for State vector, you can use table function.

```
> State <- c("DL", "MU", "NY", "DL", "NY", "MU")
> table(State)
State
DL MU NY
 2  2  2
```

To calculate mean for a vector, you can use mean function.

```
> x <- c(1,2,3,NA,5,6)
> mean(x)
[1] NA
```

Since the above vector contains a NA (not available) value, the mean function returns NA.

To calculate mean for a vector excluding NA values, you can include na.rm = TRUE parameter in mean function.

```
> x <- c(1,2,3,NA,5,6)
> mean(x)
[1] NA
> mean(x,na.rm = TRUE)
[1] 3.4
```

You can use subscripts to refer elements of a vector.

```
> x <- c(1,2,3,4,5,6)
> sum(x[c(3,5)]) #refers 3rd and 5th elements of x vector
[1] 8
```

Convert a column "x" to numeric

data$x = as.numeric(data$x)

Some useful vectors can be created quickly with R. The colon operator is

used to generate integer sequences

> 1:10

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

7

> -3:4

[1] -3 -2 -1 0 1 2 3 4

> 9:5

[1] 9 8 7 6 5

More generally, the function seq() can generate any arithmetic progression.

> seq(from=2, to=6, by=0.4)

[1] 2.0 2.4 2.8 3.2 3.6 4.0 4.4 4.8 5.2 5.6 6.0

> seq(from=-1, to=1, length=6)

[1] -1.0 -0.6 -0.2 0.2 0.6 1.0

Sometimes it's necessary to have repeated values, for which we use rep()

> rep(5,3)

[1] 5 5 5

> rep(2:5,each=3)

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

> rep(-1:3, length.out=10)

[1] -1 0 1 2 3 -1 0 1 2 3

We can also use R's vectorization to create more interesting sequences:

> 2^(0:10)

[1] 1 2 4 8 16 32 64 128 256 512 1024

8

> 1:3 + rep(seq(from=0,by=10,to=30), each=3)

[1] 1 2 3 11 12 13 21 22 23 31 32 33

Lists:

A list allows you to store a variety of objects.

```
> mylist <- list(x, y, z, gender, mydata)
> mylist
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] 1 3 5 7 9

[[3]]
[1] 1 2 5 4 7

[[4]]
[1] "m" "f" "m" "m" "f"

[[5]]
  x y z gender
1 1 1 1      m
2 2 3 2      f
3 3 5 5      m
4 4 7 4      m
5 5 9 7      f
```

You can use subscripts to select the specific component of the list.

```
> mylist[[3]]
[1] 1 2 5 4 7
```

> x <- list(1:3, TRUE, "Hello", list(1:2, 5))

Here x has 4 elements: a numeric vector, a logical, a string and another list.

We can select an entry of x with double square brackets:

```
> x[[3]]

[1] "Hello"
```

To get a sub-list, use single brackets:

```
> x[c(1,3)]

[[1]]

[1] 1 2 3

[[2]]

[1] "Hello"
```

Notice the difference between x[[3]] and x[3].

We can also name some or all of the entries in our list, by supplying argument names to list():

```
> x <- list(y=1:3, TRUE, z="Hello")

> x

$y

[1] 1 2 3

[[2]]

[1] TRUE

$z

[1] "Hello"
```

Notice that the [[1]] has been replaced by $y, which gives us a clue as to how we can recover the entries by their name. We can still use the numeric position if we prefer:

```
> x$y

[1] 1 2 3

> x[[1]]
```

[1] 1 2 3

The function names() can be used to obtain a character vector of all

thenames of objects in a list.

> names(x)

[1] "y" ""

"z"

Implementation of various operations on matrix, array and factors in R

Theory:

Matrices are much used in statistics, and so play an important role in R. To create a matrix use the function matrix(), specifying elements by column first:

> matrix(1:12, nrow=3,

ncol=4)[,1] [,2] [,3] [,4]

[1,] 1 4 7 10

[2,] 2 5 8 11

[3,] 3 6 9 12

This is called column-major order. Of course, we need only give one of the dimensions:

> matrix(1:12, nrow=3)

unless we want vector recycling to help us:

> matrix(1:3, nrow=3,

ncol=4)[,1] [,2] [,3] [,4]

[1,] 1 1 1 1

[2,] 2 2 2 2

[3,] 3 3 3 3

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Sometimes it's useful to specify the elements by row first

```
> matrix(1:12, nrow=3, byrow=TRUE)
```

There are special functions for constructing certain matrices:

```
> diag(3)

[,1] [,2]

[,3]

[1,] 1 0 0

[2,] 0 1 0

[3,] 0 0 1


> diag(1:3)


[,1] [,2] [,3]

[1,] 1 0 0

[2,] 0 2 0

[3,] 0 0 3

> 1:5 %o% 1:5

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

[1,] 1 2 3 4 5

[2,] 2 4 6 8 10

[3,] 3 6 9 12 15

[4,] 4 8 12 16 20

[5,] 5 10 15 20 25
```

The last operator performs an outer product, so it creates a matrix with (i, j)-th entry xiyj . The function outer() generalizes this to any function f on two arguments, to create a matrix

with entries f(xi , yj). (More on functions later.)

> outer(1:3, 1:4,

"+") [,1] [,2] [,3]

[,4]

[1,] 2 3 4 5

[2,] 3 4 5 6

[3,] 4 5 6 7

Matrix multiplication is performed using the operator %*%, which is

quitedistinct from scalar multiplication *.

> A <- matrix(c(1:8,10), 3, 3)

> x <- c(1,2,3)

> A %*% x # matrix

multiplication[,1]

[1,] 30

[2,] 36

[3,] 45

> A*x # NOT matrix

multiplication[,1] [,2] [,3]

[1,] 1 4 7

[2,] 4 10 16

[3,] 9 18 30

Standard functions exist for common mathematical operations on matrices

> t(A) #

transpose[,1]

[,2] [,3]

```
[1,] 1 2 3

[2,] 4 5 6

[3,] 7 8 10

> det(A) #

determinant[1] -3

> diag(A) #

diagonal[1] 1 5 10

> solve(A) #

inverse[,1] [,2]

[,3]

[1,] -0.6667 -0.6667 1

[2,] -1.3333 3.6667 -2

[3,] 1.0000 -2.0000 1
```

Array:

Of course, if we have a data set consisting of more than two pieces of categorical information about each subject, then a matrix is not sufficient. The generalization of matrices to higher dimensions is the array. Arrays are defined much like matrices, with a call to the array() command. Here is a $2 \times 3 \times 3$ array:

```
> arr = array(1:18, dim=c(2,3,3))

> arr

, , 1

[,1] [,2] [,3]

[1,] 1 3 5

[2,] 2 4 6

, , 2
```

[,1] [,2] [,3]

[1,] 7 9 11

[2,] 8 10 12

, , 3

[,1] [,2] [,3]

[1,] 13 15 17

[2,] 14 16 18

Each 2-dimensional slice defined by the last co-ordinate of the array is shown as a 2 × 3 matrix. Note that we no longer specify the number of rows and columns separately, but use a single vector dim whose length is the number of dimensions. You can recover this vector with the dim() function.

> dim(ar

r)[1] 2 3

3

Note that a 2-dimensional array is identical to a matrix. Arrays can be

subsetted and modified in exactly the same way as a matrix, only using

theappropriate number of co-ordinates:

> arr[1,2,

3][1] 15

> arr[,2,]
[,1] [,2] [,3]

[1,] 3 9 15

[2,] 4 10 16

> arr[1,1,] = c(0,-1,-2) # change some values

> arr[,,1,drop=FALSE]

, , 1

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
     [,1] [,2] [,3]

[1,]    0    3    5

[2,]    2    4    6
```

Factors

R has a special data structure to store categorical variables. It tells R that a variable isnominal or ordinal by making it a factor.

Simplest form of the factor function :

```
> gender <- c(1,2,1,2,1,2,1,2)
> gender <- factor(gender)
```

Ideal form of the factor function :

```
> gender <- c(1,2,1,2,1,2,1,2)
>
> gender <- factor(gender,
+ levels = c(1,2),
+ labels = c("male","female"))

> table(gender)
gender
  male female
     4      4
```

The factor function has three parameters:
1. Vector Name
2. Values (Optional)
3. Value labels (Optional)

Convert a column "x" to
factor

data$x = as.factor(data$x)

Implementation and perform the various operations on data frames in R

Theory:

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

- Data frames are tabular data objects.
- A Data frame is a list of vector of equal length.
- **Data frame** in R is used for storing data tables.

Characteristics of a data frame:

1. The column names should be non-empty.
2. The row names should be unique.
3. The data stored in a data frame can be of numeric, factor or character type.

Create Data Frame

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),
      start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-
11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)# Print the data frame.
print(emp.data)
```

When we execute the above code, it produces the following result –

```
emp_id    emp_name     salary     start_date
1     1     Rick        623.30     2012-01-01
2     2     Dan         515.20     2013-09-23
3     3     Michelle    611.00     2014-11-15
4     4     Ryan        729.00     2014-05-11
5     5     Gary        843.25     2015-03-27
```

Get the Structure of the Data Frame

The structure of the data frame can be seen by using str() function.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)
# Get the structure of the data frame.
str(emp.data)
```

When we execute the above code, it produces the following result –

```
'data.frame':   5 obs. of  4 variables:
 $ emp_id    : int  1 2 3 4 5
 $ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...
 $ salary    : num  623 515 611 729 843
 $ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying summary()function.

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)
# Print the summary.
print(summary(emp.data))
```

When we execute the above code, it produces the following result –

```
     emp_id    emp_name            salary        start_date
 Min.   :1   Length:5           Min.   :515.2   Min.   :2012-01-01
 1st Qu.:2   Class :character   1st Qu.:611.0   1st Qu.:2013-09-23
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
 Median :3   Mode  :character   Median :623.3   Median :2014-05-11
 Mean   :3                     Mean   :664.4   Mean   :2014-01-14
 3rd Qu.:4                     3rd Qu.:729.0   3rd Qu.:2014-11-15
 Max.   :5                     Max.   :843.2   Max.   :2015-03-27
```

Extract Data from Data Frame:

```
# Extract Specific columns.

result <- data.frame(emp.data$emp_name,emp.data$salary)

print(result)
```

When we execute the above code, it produces the following result –

```
  emp.data.emp_name emp.data.salary
1              Rick          623.30
2               Dan          515.20
3          Michelle          611.00
4              Ryan          729.00
5              Gary          843.25

# Extract first two rows.

result <- emp.data[1:2,]

print(result)
```

When we execute the above code, it produces the following result –

```
  emp_id    emp_name   salary    start_date
1      1     Rick      623.3     2012-01-01
2      2     Dan       515.2     2013-09-23

# Extract 3rd and 5th row with 2nd and 4th column.

result <- emp.data[c(3,5),c(2,4)]

print(result)
```

When we execute the above code, it produces the following result –

```
emp_id      name         start_date
3           Michelle    2014-11-15
5           Gary        2015-03-27
```

Expand Data Frame

A data frame can be expanded by adding columns and rows.0

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

1. Add Column

Just add the column vector using a new column name.

```
# Add the "dept" coulmn.

emp.data$dept <- c("IT","Operations","IT","HR","Finance")

v <- emp.data

print(v)
```

When we execute the above code, it produces the following result –

```
  emp_id   emp_name    salary    start_date       dept
1      1    Rick        623.30    2012-01-01       IT
2      2    Dan         515.20    2013-09-23       Operations
3      3    Michelle    611.00    2014-11-15       IT
4      4    Ryan        729.00    2014-05-11       HR
5      5    Gary        843.25    2015-03-27       Finance
```

2. Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rowsin the same structure as the existing data frame and use the **rbind()** function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

```
# Create the second data frame
emp.newdata <-    data.frame(
  emp_id = c (6:8),
  emp_name = c("Rasmi","Pranab","Tusar"),
  salary = c(578.0,722.5,632.8),
  start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
  dept = c("IT","Operations","Fianance"),
  stringsAsFactors = FALSE
)

# Bind the two data frames.
emp.finaldata <- rbind(emp.data,emp.newdata)
print(emp.finaldata)
```

Conclusion:

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Experiment No: 3

Aim: Write program to demonstrate Loops & Vectorization Missing Values.

Theory:

In R programming, we require a control structure to run a block of code multiple times. Loops come in the class of the most fundamental and strong programming concepts. A loop is a control statement that allows multiple executions of a statement or a set of statements. The word 'looping' means cycling or iterating.

A loop asks a query, in the loop structure. If the answer to that query requires an action, it will be executed. The same query is asked again and again until further action is taken. Any time the query is asked in the loop, it is known as an iteration of the loop. There are two components of a loop, the control statement, and the loop body. The control statement controls the execution of statements depending on the condition and the loop body consists of the set of statements to be executed.

In order to execute the identical lines of code numerous times in a program, a programmer can simply use a loop.

There are three types of loop in R programming:

- For Loop
- While Loop
- Repeat Loop

For Loop in R

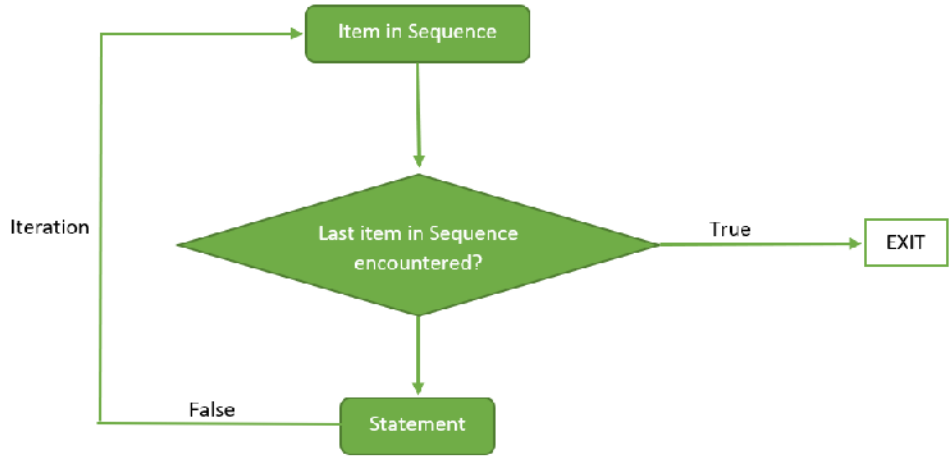
It is a type of control statement that enables one to easily construct a loop that has to run statements or a set of statements multiple times. For loop is commonly used to iterate over items of a sequence. It is an entry controlled loop, in this loop the test condition is tested first, then the body of the loop is executed, the loop body would not be executed if the test condition is false.

R – For loop Syntax:

```
for (value in sequence)
{
  statement
}
```

For Loop Flow Diagram:

Name of Student: **Experiment No.**
DOP: **DOC:** **Roll No:**



Below are some programs to illustrate the use of **for** loop in R programming.
Example 1: Program to display numbers from 1 to 5 using for loop in R.

```
# R program to demonstrate the use of for loop

# using for loop
for (val in 1: 5)
{
        # statement
        print(val)
}
```

Output:
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
Here, for loop is iterated over a sequence having numbers from 1 to 5. In each iteration, each item of the sequence is displayed.

Example 2: Program to display days of a week.

```
# R program to illustrate
# application of for loop

# assigning strings to the vector
week < - c('Sunday',
                'Monday',
                'Tuesday',
                'Wednesday',
                'Thursday',
```

```
            'Friday',
            'Saturday')

# using for loop to iterate
# over each string in the vector
for (day in week)
{

        # displaying each string in the vector
        print(day)
}
```

Output:

```
[1] "Sunday"
[1] "Monday"
[1] "Tuesday"
[1] "Wednesday"
[1] "Thusrday"
[1] "Friday"
[1] "Saturday"
```

In the above program, initially, all the days(strings) of the week are assigned to the vector week. Then for loop is used to iterate over each string in a week. In each iteration, each day of the week is displayed.

While Loop in R

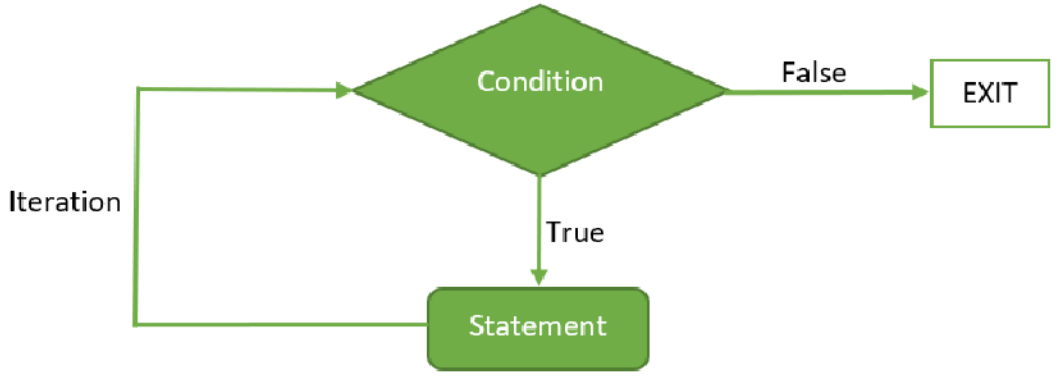
It is a type of control statement which will run a statement or a set of statements repeatedly unless the given condition becomes false. It is also an entry controlled loop, in this loop the test condition is tested first, then the body of the loop is executed, the loop body would not be executed if the test condition is false.

R – While loop Syntax:

```
while ( condition )
{
  statement
}
```

While loop Flow Diagram:

Name of Student: **Experiment No.**
DOP: **DOC:** **Roll No:**



Below are some programs to illustrate the use of the **while** loop in R programming.
Example 1: Program to display numbers from 1 to 5 using while loop in R.

```
# R program to demonstrate the use of while loop


val = 1

# using while loop

while (val <= 5)

{

        # statements

        print(val)

        val = val + 1

}
```

Output:

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

Initially, the variable value is initialized to 1. In each iteration of the while loop the condition is checked and the value of val is displayed and then it is incremented until it becomes 5 and the condition becomes false, the loop is terminated.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Example 2: Program to calculate factorial of a number.

```
# R program to illustrate
# application of while loop
# assigning value to the variable
# whose factorial will be calculated
n < - 5

# assigning the factorial variable
# and iteration variable to 1
factorial < - 1
i < - 1
# using while loop
while (i <= n)
{

        # multiplying the factorial variable
        # with the iteration variable
        factorial = factorial * i
        # incrementing the iteration variable
        i = i + 1
}


# displaying the factorial

print(factorial)
```

Output:

[1] 120

Here, at first, the variable n is assigned to 5 whose factorial is going to be calculated, then variable i and factorial are assigned to 1. i will be used for iterating over the loop, and factorial will be used for calculating the factorial. In each iteration of the loop, the condition is checked i.e. i should be less than or equal to 5, and after that factorial is multiplied with the value of i, then i is incremented. When i becomes 5, the loop is terminated and the factorial of 5 i.e. 120 is displayed beyond the scope of the loop.

Repeat Loop in R

It is a simple loop that will run the same statement or a group of statements repeatedly until the stop condition has been encountered. Repeat loop does not have any condition to terminate the loop, a programmer must specifically place a condition within the loop's body and use the declaration of a break statement to terminate this loop. If no condition is present in the body of the repeat loop then it will iterate infinitely.

R – Repeat loop Syntax:

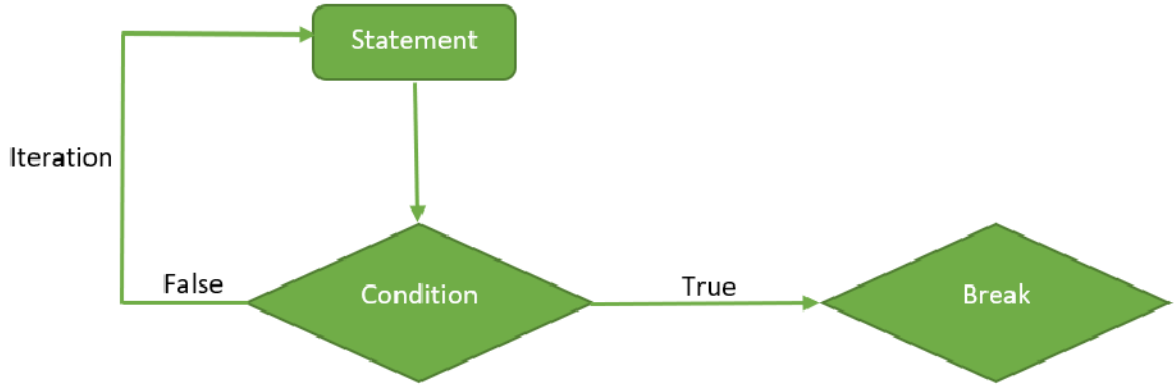
Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
repeat
{
  statement

  if( condition )
  {
    break
  }
}
```

Repeat loop Flow Diagram:



To terminate the **repeat** loop, we use a jump statement that is the **break** keyword. Below are some programs to illustrate the use of repeat loops in R programming.

Example 1: Program to display numbers from 1 to 5 using repeat loop in R.

```
# R program to demonstrate the use of repeat loop

val = 1

# using repeat loop
repeat
{
        # statements
        print(val)
        val = val + 1

        # checking stop condition
        if(val > 5)
        {
                # using break statement
                # to terminate the loop
                break
        }
}
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Output:

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

In the above program, the variable val is initialized to 1, then in each iteration of the repeat loop the value of val is displayed and then it is incremented until it becomes greater than 5. If the value of val becomes greater than 5 then break statement is used to terminate the loop.

Example 2: Program to display a statement five times.

```
# R program to illustrate
# the application of repeat loop

# initializing the iteration variable with 0
i < - 0

# using repeat loop
repeat
{
        # statement to be executed multiple times
        print("Geeks 4 geeks!")

        # incrementing the iteration variable
        i = i + 1

        # checking the stop condition
        if (i == 5)
        {
                # using break statement
                # to terminate the loop
                break
        }
}
```

Output:

```
[1] "Geeks 4 geeks!"
[1] "Geeks 4 geeks!"
[1] "Geeks 4 geeks!"
[1] "Geeks 4 geeks!"
[1] "Geeks 4 geeks!"
```

Here, initially the variable i is initialized with 0 then in each iteration of the repeat loop after printing Geeks 4 geeks! the value of i is incremented till it becomes 5 and the condition in the if statement becomes true then, the break statement is executed to terminate the repeat loop.

Jump Statements in Loop

We use a jump statement in loops to terminate the loop at a particular iteration or to skip a particular iteration in the loop. The two most commonly used jump statements in loops are:

- **Break Statement:** The break keyword is a jump statement that is used to terminate the loop at a particular iteration.

Example:

```
# R program to illustrate
# the use of break statement

# using for loop
# to iterate over a sequence
for (val in 1: 5)
{
        # checking condition
        if (val == 3)
        {
                # using break keyword
                break
        }

        # displaying items in the sequence
        print(val)
}
```

Output:

[1] 1

[1] 2

In the above program, if the value of val becomes 3 then the break statement will be executed and the loop will terminate.

- **Next Statement:** The next keyword is a jump statement which is used to skip a particular iteration in the loop.

Example:

```
# R program to illustrate
# the use of next statement

# using for loop
# to iterate over the sequence
for (val in 1: 5)
```

```
{
        # checking condition
        if (val == 3)
        {
                # using next keyword
                next
        }

        # displaying items in the sequence
        print(val)
}
```

Output:

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

In the above program, if the value of Val becomes 3 then the next statement will be executed hence the current iteration of the loop will be skipped. So 3 is not displayed in the output.

What is Missing Values

1. A missing value is one whose value is unknown.
2. Missing values are represented in R by the `NA` symbol.
3. `NA` is a special value whose properties are different from other values.
4. `NA` is one of the very few reserved words in R: you cannot give anything this name. (Because R is case-sensitive, `na` and `Na` are okay to use)
5. Missing values are often legitimate: values really are missing in real life.
6. `NA`s can arise when you read in a Excel spreadsheet with empty cells, for example. You will also see `NA` when you try certain operations that are illegal or don't make sense.

Here are some examples of operations that produce `NA`'s.

```
> var (8)                          # Variance of one number

[1] NA

> as.numeric (c("1", "2", "three", "4"))   # Illegal conversion

[1]  1  2 NA  4
Warning message:
NAs introduced by coercion

> c(1, 2, 3)[4]                    # Vector subscript out of range
```

```
[1] NA

> NA - 1                    # Most operations on NAs produce NAs

[1] NA

> a <- data.frame (a = 1:3, b = 2:4)
> a[4,]                     # Data frame row subscript out of range
   a  b
NA NA NA                        # The first NA there is the row number

> a[,4]                     # Specifying a non-existent column just produces an error
Error in `[.data.frame`(a, , 4) : undefined columns selected

#
# Here's one that's particularly irksome
#
> a[1,2] <- NA                # Suppose you have an NA in your dataframe...
> a[a$b < 4,]                 # ...and you try to index on that column
   a  b
NA NA NA                        # You get one of these NA rows for each NA in that column.
 2  2  3
```

Note that if you specify a row number out of range for a data frame, that's not an error. You just get a row full of NAs. Interestingly, if you specify a row index of a matrix that's too big, you get a different respose altogether. That is an error. It's also an error to specify too big an index for a column of either a matrix or a data frame.

Example
we construct a 4×3 data frame including two NA values to learn how to deal with missing values.

```
data <- as.data.frame(matrix(c(1:7, NA, 8:10, NA), ncol = 3, byrow = TRUE))
data
```

Output

```
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
## 3  7 NA  8
## 4  9 10 NA
```

How to Discover NA in R

First, let's find which cells of the data are missing by using is.na() function. Then, we discover which rows include NA values by using complete.cases() function.

```
is.na(data)
##        V1    V2    V3
## [1,] FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE
## [3,] FALSE  TRUE FALSE
## [4,] FALSE FALSE  TRUE

complete.cases(data)
## [1]  TRUE  TRUE FALSE FALSE
```

How to Count NA in R

In this part, we learn three ways of counting NA values in R. Firstly, we count all missing observations in R with sum() function. Then, we count the number of missing values in each column by utilizing colSums() function. Last, we learn how to determine the number of NA values in each row by using rowSums() function.

```
sum(is.na(data))

## [1] 2

colSums(is.na(data))

## V1 V2 V3
##  0  1  1

rowSums(is.na(data))

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

How to Remove NA in R

In this section, we work on six ways of removing NA values in R. Firstly, we use brackets with complete.cases() function to exclude missing values in R. Secondly, we omit missing values with na.omit() function. Thirdly, we learn how to get rid of NA values by using na.exclude() function. Then, we exclude NA values using drop_na() function available in tidyr package (Wickham, 2020). Moreover, we get rid of missing values of a column specified in drop_na() function. Last, we learn how to remove missing values of specified columns in drop_na() function.

```
data[complete.cases(data),]
##   V1 V2 V3
## 1  1  2  3
```

```
## 2  4  5  6
```

```
na.omit(data)
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
```

```
na.exclude(data)
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
```

```
library(tidyr)
data %>% drop_na()
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
```

```
data %>% drop_na(V2)
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
## 3  9 10 NA
```

```
data %>% drop_na(V2,V3)
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
```

How to Leave Data with No Action

We leave data without removing any missing values by using na.pass() function.

```
na.pass(data)
##   V1 V2 V3
## 1  1  2  3
## 2  4  5  6
## 3  7 NA  8
## 4  9 10 NA
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Operations on Missing Values

Almost every operation performed on an NA produces an NA. For example:

```
> x <- c(1, 2, NA, 4)              # Set up a numeric vector
> x                                # There's an NA in there
[1]  1  2 NA  4
> x + 1                            # NA + 1 = NA
[1]  2  3 NA  5
> sum(x)                           # This produces NA because we can't add NAs
[1] NA
> length(x)                        # This is okay
[1] 4
```

By the way, the default mode of NA is logical. That generally won't affect us.

Detecting NAs

You can't find missing values by looking at x == NA. Like most other functions, the == operator returns NA when either argument is NA. The is.na() function will find missing values for you: this function returns a logical vector the same length as its argument, with T for missing values and F for non-missings. It's fairly common to want to know the index of the missing values, and the which() function will help do this for you. For example:

```
> x              # Here's my vector
[1]  1  2 NA  4
> is.na(x)       # Is it NA?
[1] F F T F      # Answer: no, no, yes, no.
> which (is.na(x)) # Which one is NA?
[1] 3            # Answer: the third one
```

To find all the rows in a data frame with at least one NA, try this:

```
> unique (unlist (lapply (your.data.frame, function (x) which (is.na (x)))))
```

lapply() applies the function to each column and returns a list whose i-th element is a vector containing the indices of the elements which have missing values in column i. unlist() turns that list into a vector and unique() gets rid of the duplicates. To learn more about lapply(), see the apply family of functions.

Ways to Exclude Missing Values

Math functions generally have a way to exclude missing values in their calculations.
mean(), median(), colSums(), var(), sd(), min() and max() all take the na.rm argument.
When this is TRUE, missing values are omitted.
The default is FALSE, meaning that each of these functions returns NA if any input number is NA.
Note that cor() and its relatives don't work that way: with those you need to supply the use= argument.
This is to permit more complicated handling of missing values than simply omitting them.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

R's modeling functions accept an na.action argument that tells the function what to do when it encounters an NA.
This causes the modeling function to call one of the missing value filter functions.
These functions replace the original data set by a new data set in which the NAs have been altered.
The default setting is na.omit, which excludes all rows with any missing values.
An alternative is na.action=na.fail, which just stops when it encounters any missing values.
This is useful if you didn't know you had any. The filter functions are:

na.fail: Stop if any missing values are encountered
na.omit: Drop out any rows with missing values anywhere in them and forgets them forever.
na.exclude: Drop out rows with missing values, but keeps track of where they were (so that when you make predictions, for example, you end up with a vector whose length is that of the original response.)
na.pass: Take no action.

A couple of other packages supply more alternatives:

na.tree.replace (library (tree): For discrete variables, adds a new category called "NA" to replace the missing values.
na.gam.replace (library gam): Operates on discrete variables like na.tree.replace(); for numerics, NAs are replaced by the mean of the non-missing entries.
Here are examples of these functions at work.
You can call them directly, as I will do here, but they are also commonly used as values for the na.action= argument to the modeling functions.

```
#
# Set up a data frame, make a couple of elements NA.
#
> a <- data.frame (c1 = 1:8, c2 = factor (c("a", "b", "a", "c", "b", "c", "a", "b")))
> a[4,1] <- a[6,2] <- NA    # This repeated assignment is legal and does what you expect.
> a
  c1  c2
1  1   a
2  2   b
3  3   a
4 NA   c
5  5   b
6  6                # Note the slightly different display of missings inside factors
7  7   a
8  8   b
> levels(a$c2)             # Note the levels of c2 are "a," "b" and "c." NA is not a level.
[1] "a" "b" "c"

> na.fail (a)              # Fails if NAs are present
Error in na.fail.default(a) : missing values in object
```

```
> na.exclude (a)          # Omits rows with NAs in them
  c1 c2
1  1  a
2  2  b
3  3  a
5  5  b
7  7  a
8  8  b

> na.gam.replace (a)       # Replace missing in c1 with the mean of the non-missings;
       c1 c2             # Add a new level to c2
1 1.000000  a
2 2.000000  b
3 3.000000  a
4 4.571429  c
5 5.000000  b
6 6.000000 NA
7 7.000000  a
8 8.000000  b
> levels (na.gam.replace(a)$c2)  # There's now a fourth level in that column
[1] "a"  "b"  "c"  "NA"
```

Special Case 1a: Missing Values in Factor Vectors

We noted above that a missing value in a factor variable is displayed as <NA> rather than just NA.
Again, missing values do not have a level, but you can change a missing value to one of the existing levels. (You can't create a new level on the fly, though -- see the discussion of factors.

Special Case 2: Missing Values in Character Vectors

Character vectors can have missing values.
They display as NA in the usual way.
This really isn't a special case at all.

Special Case 3: NaNs

In addition to NA, R has a special value NaN for "not a number." 0/0 is an example of a calculation that will produce a NaN. NaNs print as NaN, but generally act like NAs. (For example, a computation done on an NaN produces an NaN; if you try to extract the NaNth element of a vector, you get NA.)
One more special value is Inf. If you need them, there are is.nan() and functions for finding things that are NaN or infinite and not NA.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Master in Computer Application

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Conclusion:-

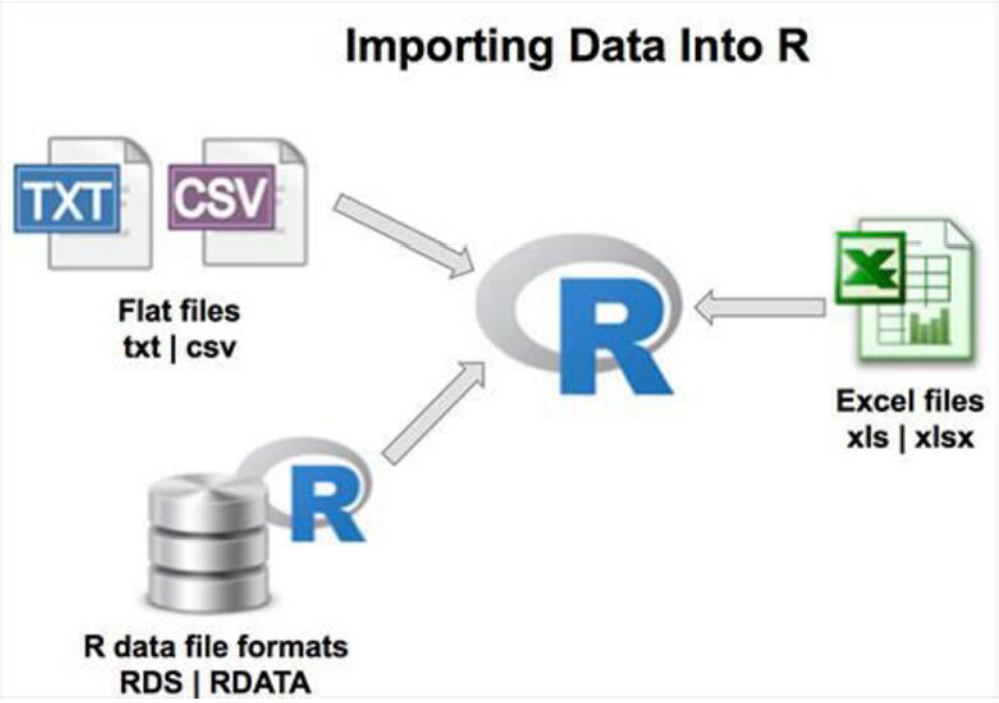
Name of Student: **Experiment No.**
DOP: **DOC:** **Roll No:**

Experiment No: 4

Aim: Write program to Demonstrate Importing and exporting data.

Theory:-

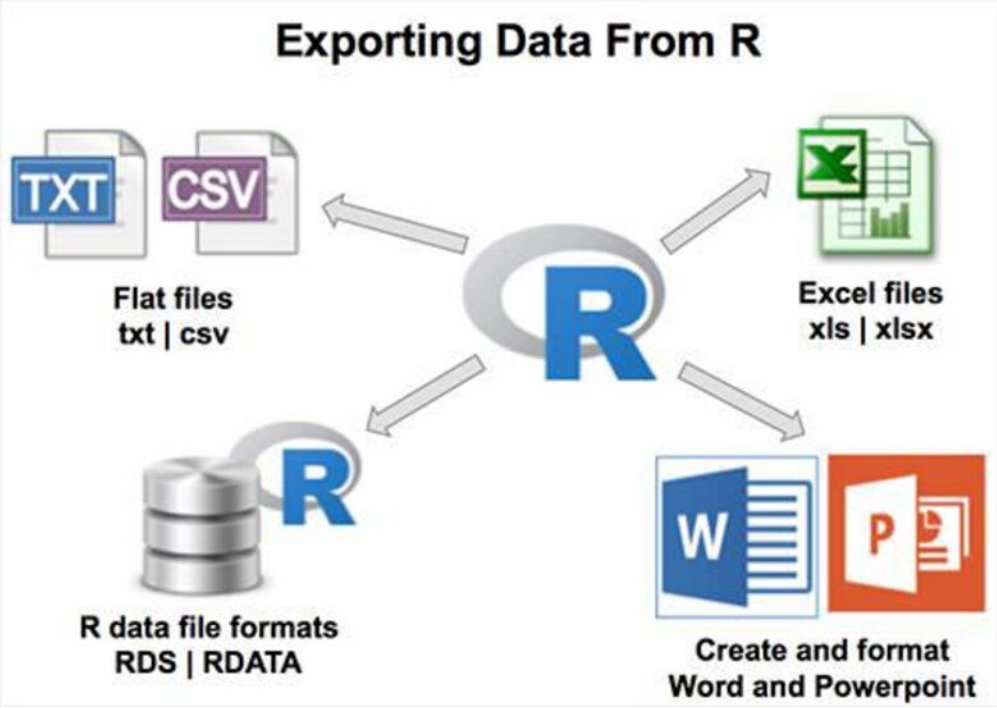
What is the meaning of importing data?
Data Import **lets you upload data from external sources and combine it with data you collect via Analytics**. You can then use Analytics to organize and analyze all of your data in ways that better reflect your business.



What is exporting of data?
Data export is **the extraction and conversion of raw data from their existing format into a format required by another application**. Exporting data is also a way of backing up data or moving it between two different versions of programs.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**



Different ways of Importing Data into R

How to handle different types of data in R

Data can come from many sources. Some of the most common ones are

- Flat Files — CSV, txt, tsv, etc
- Data from Excel
- DataBases — Postgresql, Mysql, etc
- Web
- Statistical Softwares — SAS, SPSS, STATA

How do I get the current working directory in R?

Get and Set working directory in R, Your environment is always pointed to some working location when you use R. To get the working directory and set the working directory in R, use the following functions: Get the current working directory with getwd (). Set the current working directory with setwd ('Path/To/Your/Folder').

```
getwd()
```

The collection of facts is known as data. Data can be in different forms. To analyze data using R programming Language, data should be first imported in R which can be in different formats like txt, CSV, or any other delimiter separated files. After importing data then manipulate, analyze, and report it.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Import Data from a File in R Programming Language

In this experiment, we are going to see how to import different files in R programming Language.

Import CSV file into R

Method 1: Using read.csv() methods.

Here we will import csv file using read.csv() method in R.

Syntax: *read.csv(path, header = TRUE, sep = ",")*

Arguments :

- *path : The path of the file to be imported*
- *header : By default : TRUE . Indicator of whether to import column headings.*
- *sep = "," : The separator for the values in each row.*

**********Program **********

```
# specifying the path
path <- "C:/Users/HCL/Documents/myfile.csv"

# reading contents of csv file
content <- read.csv(path)

# contents of the csv file
print (content)
```

**********Program **********

**********output **********

```
> path <- "C:/Users/HCL/Documents/myfile.csv"
> content <- read.csv(path)
> print (content)
  v1 v2 v3
1  1  5  8
2  2  6  9
3  3  7  0
4  4  8  7
5  5  9  6
```

**********output **********

Method 2: Using read.table() methods.

Here we will use read.table() methods to import CSV file into R Programming Language.

**********Program **********

```
# simple R program to read csv file using read.table()
x <- read.csv2("C:/Users/HCL/Documents/myfile.csv", header = TRUE, sep=",")

# print x
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
print(x)
******************************Program ****************************

******************************output ****************************
> x <- read.csv2("C:/Users/HCL/Documents/myfile.csv", header = TRUE, sep=",")
> print(x)
  v1 v2 v3
1  1  5  8
2  2  6  9
3  3  7  0
4  4  8  7
5  5  9  6
******************************output ****************************
```

Importing Data from a Text File

We can easily import or read **.txt** file using basic R function **read.table()**. read.table() is used to read a file in table format. This function is easy to use and flexible.

Syntax:

 # read data stored in .txt file

x<-read.table("file_name.txt", header=TRUE/FALSE)

```
******************************Program ****************************
# Simple R program to read txt file
x<-read.table("C://Users//HCL//Documents//myfile.txt", header=FALSE)

# print x
print(x)
******************************Program ****************************

******************************output ****************************
> x<-read.table("C:/Users/HCL/Documents/myfile.txt", header=FALSE)
> print(x)
  V1 V2 V3
1  1  a xy
2  2  b yx
3  3  c we
4  4  d qw
5  5  e qr
6  6  u ui
******************************output ****************************
```

If the header argument is set at TRUE, which reads the column names if they exist in the file.

Importing Data from a delimited file

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

R has a function **read.delim()** to read the delimited files into the list. The file is by default separated by a tab which is represented by sep="", that separated can be a comma(,), dollar symbol($), etc.

Syntax: *read.delim("file_name.txt", sep="", header=TRUE)*

```
**********Program **********
x <- read.delim("C:/Users/HCL/Documents/myfile.csv", sep="|", header=TRUE)

# print x
print(x)

# print type of x
typeof(x)
**********Program **********
```

```
**********output **********
> x <- read.delim("C:/Users/HCL/Documents/myfile.csv", sep="|", header=TRUE)
> print(x)
  v1.v2.v3
1    1,5,8
2    2,6,9
3    3,7,0
4    4,8,7
5    5,9,6
> typeof(x)
[1] "list"
**********output **********
```

Exporting Data from R Scripts

When a program is terminated, the entire data is lost. Storing in a file will preserve one's data even if the program terminates. If one has to enter a large number of data, it will take a lot of time to enter them all. However, if one has a file containing all the data, he/she can easily access the contents of the file using a few commands in R. One can easily move his data from one computer to another without any changes. So those files can be stored in various formats. It may be stored in .txt(tab-separated value) file, or in a tabular format i.e .csv(comma-separated value) file or it may be on internet or cloud. R provides very easier methods to export data to those files.

Exporting data to a text file

One of the important formats to store a file is in a text file. R provides various methods that one can export data to a text file.

- `write.table()`: The R base function `write.table()` can be used to export a data frame or a matrix to a text file.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Syntax:

write.table(x, file, append = FALSE, sep = " ", dec = ".", row.names = TRUE, col.names = TRUE)

Parameters:

x: *a matrix or a data frame to be written.*

file: *a character specifying the name of the result file.*

sep: *the field separator string, e.g., sep = "\t" (for tab-separated value).*

dec: *the string to be used as decimal separator. Default is "."*

row.names: *either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written.*

col.names: *either a logical value indicating whether the column names of x are to be written along with x, or a character vector of column names to be written.*

Example:

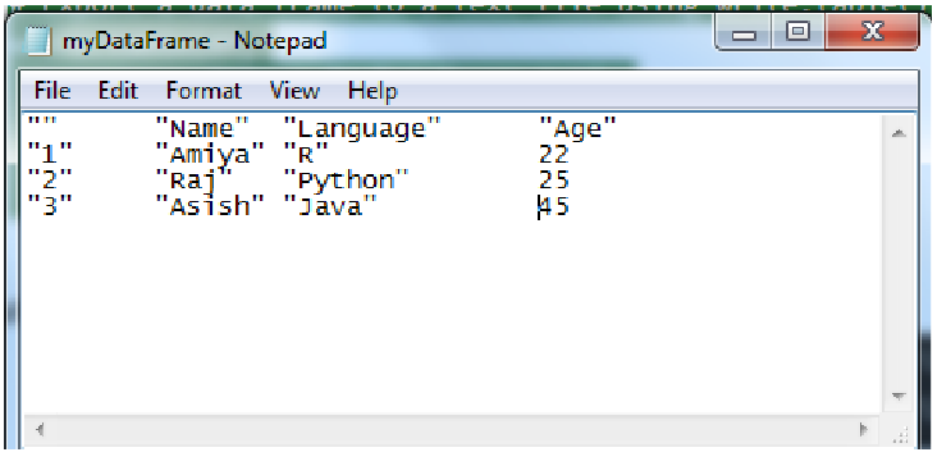
```
# R program to illustrate
# Exporting data from R
# Creating a dataframe
df = data.frame(
  "Name" = c("Amiya", "Raj", "Asish"),
  "Language" = c("R", "Python", "Java"),
  "Age" = c(22, 25, 45)
)

# Export a data frame to a text file using
write.table()
write.table(df,
            file = "myDataFrame.txt",
            sep = "\t",
            row.names = TRUE,
            col.names = NA)
```

**********output **********

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**



**********output **********

write_tsv(): This method is also used for to export data to a tab separated ("\t") values by using the help of **readr** package.

Syntax: *write_tsv(file, path)*
Parameters:
file: *a data frame to be written*
path: *the path to the result file*

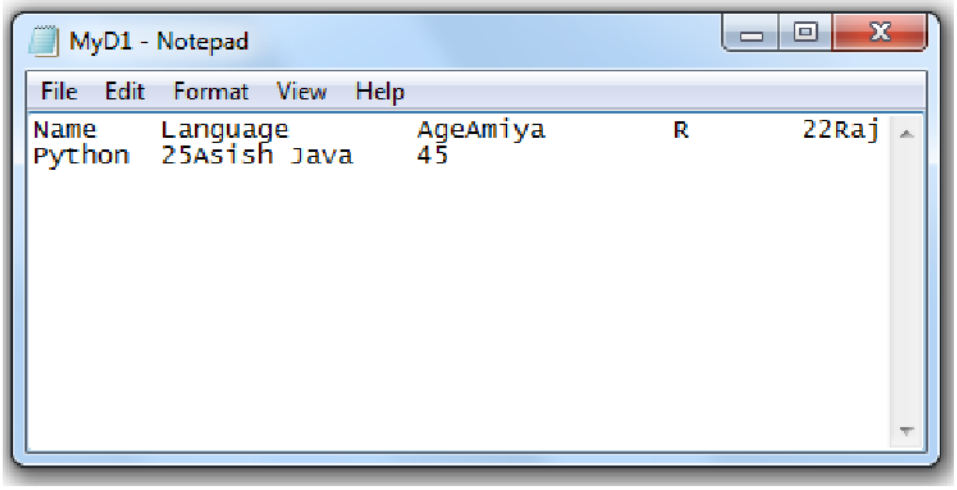
```
# R program to illustrate
# Exporting data from R
getwd()
install.packages("tidyverse")
install.packages("readr")
# Importing readr library
library(readr)
# Creating a dataframe
df = data.frame(
"Name" = c("Amiya", "Raj", "Asish"),
"Language" = c("R", "Python", "Java"),
"Age" = c(22, 25, 45)
)

# Export a data frame using write_tsv()
write_tsv(df, path = "MyD1.txt")
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

**********output **********



**********output **********

Exporting data to a csv file

Another popular format to store a file is in a csv(comma-separated value) format. R provides various methods that one can export data to a csv file.

- `write.table()`: The R base function `write.table()` can also be used to export a data frame or a matrix to a csv file.

  Syntax:
  write.table(x, file, append = FALSE, sep = " ", dec = ".", row.names = TRUE, col.names = TRUE)
  Parameters:
  x: *a matrix or a data frame to be written.*
  file: *a character specifying the name of the result file.*
  sep: *the field separator string, e.g., sep = "\t" (for tab-separated value).*
  dec: *the string to be used as decimal separator. Default is "."*
  row.names: *either a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written.*
  col.names: *either a logical value indicating whether the column names of x are to be written along with x, or a character vector of column names to be written.*

```
# R program to illustrate
# Exporting data from R

# Creating a dataframe
df = data.frame(
"Name" = c("Amiya", "Raj", "Asish"),
"Language" = c("R", "Python", "Java"),
"Age" = c(22, 25, 45)
```

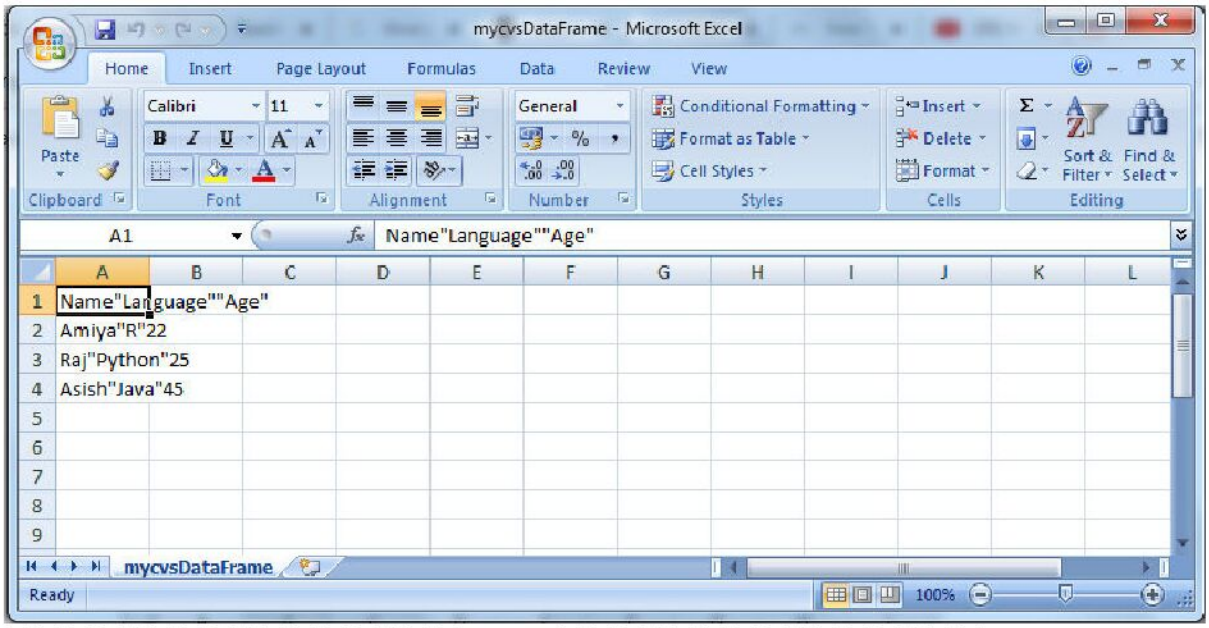
Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
)

# Export a data frame to a text file using write.table()
write.table(df,
                        file = "mycvsDataFrame.csv",
                        sep = "\t",
                        row.names = FALSE,
                        )
```

**********output **********



**********output **********

write_csv(): This method is also used for to export data to a comma separated (“, ”) values by using the help of readr package.

Syntax: *write_csv(file, path)*
Parameters:
file: *a data frame to be written*
path: *the path to the result file*

```
# R program to illustrate
# Exporting data from R

# Importing readr library
library(readr)

# Creating a dataframe
```

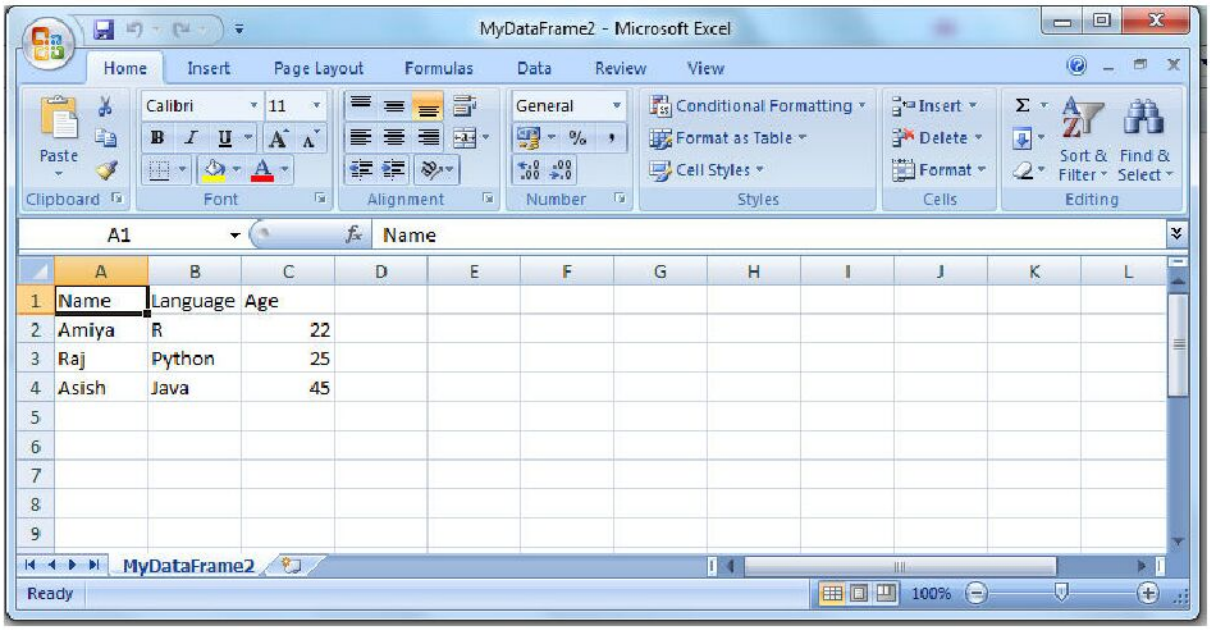
Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
df = data.frame(
"Name" = c("Amiya", "Raj", "Asish"),
"Language" = c("R", "Python", "Java"),
"Age" = c(22, 25, 45)
)

# Export a data frame using write_csv()
write_csv(df, path = "MyDataFrame2.csv")
```

**********output **********



**********output **********

https://www.geeksforgeeks.org/exporting-data-from-scripts-in-r-programming/

Conclusion:-

SSBT's College of Engineering & Technology, Bambhori, Jalgaon
Department of Master in Computer Application

Name of Student: **Experiment No.**
DOP: **DOC:** **Roll No:**

Experiment No: 5

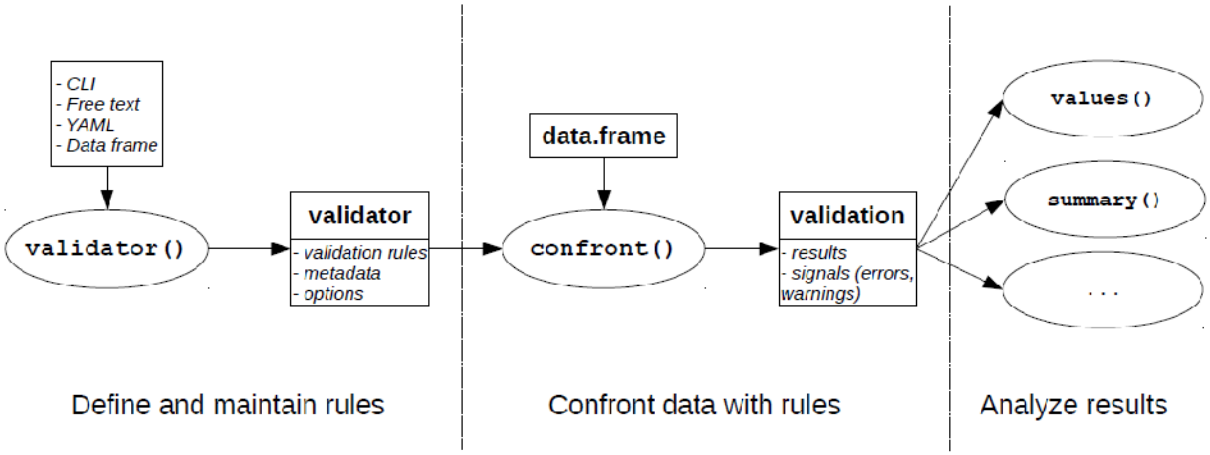
Aim: Write program for Validating & Exploring Data Manipulations (Summarizing, Sorting, Subsetting, Merging, joining)

Theory:-

What is data validation in R?

The R package validate facilitates this task by capturing and applying expert knowledge in the form of validation rules: logical restrictions on variables, records, or data sets that should be satisfied before they are considered valid input for further analysis.

A data validation task can be split up in three consecutive subtasks: loading the data and the validation rules, confronting the data with the rules, and transforming and analyzing the results.



What is data manipulation used for?

It is used in order to make data more understandable or more structured.
Data is best used when it can be manipulated for marketing, sales, accounting, and customer support.
Proper data analysis involves rearranging, sorting, modifying, and shifting data.

Which library would you use for data manipulation in R?

Dplyr is mainly used for data manipulation in R.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

summarize() method

Using the summarize method we can summarize the data in the data frame by using aggregate functions like sum(), mean(), etc. The syntax of summarize() method is specified below-

summarize(dataframeName, aggregate_function(columnName))

Example:

In the below code we presented the summarized data present in the runs column using summarize() method.

```
# import dplyr package
library(dplyr)

# create a data frame
stats <- data.frame(player=c('A', 'B', 'C', 'D'),
                    runs=c(100, 200, 408, 19),
                    wickets=c(17, 20, 7, 5))

# summarize method
summarize(stats, sum(runs), mean(runs))
```

```
> # import dplyr package
> library(dplyr)
>
> # create a data frame
> stats <- data.frame(player=c('A', 'B', 'C', 'D'),
+ runs=c(100, 200, 408, 19),
+ wickets=c(17, 20, 7, 5))
>
> # summarize method
> summarize(stats, sum(runs), mean(runs))
  sum(runs) mean(runs)
1       727     181.75
>
```

Sorting DataFrame in R using Dplyr

We will discuss about how to sort a dataframe in R programming language using Dplyr package. The package Dplyr in R programming language provides a function called arrange() function which is useful for sorting the dataframe.

Syntax :

arrange(.data, ...)

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

The methods given below show how this function can be used in various ways to sort a dataframe.

Sorting in Ascending order

Sorting in ascending order is the default sorting order in arrange() function. The attribute to sort by should be given as an argument to this function.

Example: Sorting the dataframe in ascending order

```
# Installing the loading the package
install.packages("dplyr")

library(dplyr)

# Creating dataframe
gfg = data.frame(Customers = c("Roohi", "James", "Satish", "Heera",
                               "Sehnaaz", "Joe","Raj", "Simran",
                               "Priya","Tejaswi"),

                 Product = c("Product A", "Product B", "Product C",
                             "Product A", "Product D", "Product B",
                             "Product D", "Product C", "Product D",
                             "Product A"),

                 Salary = c(514.65, 354.99, 345.44, 989.56, 767.50,
                            576.90, 878.67, 904.56,123.45, 765.78)
                 )

gfg

# Sorting the dataframe in ascending order
arrange(gfg, Salary)
```

**********output **********

```
> gfg = data.frame(Customers = c("Roohi", "James", "Satish", "Heera",
+ "Sehnaaz", "Joe","Raj", "Simran",
+ "Priya","Tejaswi"),
+
+ Product = c("Product A", "Product B", "Product C",
+ "Product A", "Product D", "Product B",
+ "Product D", "Product C", "Product D",
+ "Product A"),
+
+ Salary = c(514.65, 354.99, 345.44, 989.56, 767.50,
+ 576.90, 878.67, 904.56,123.45, 765.78)
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
+ )
>
> gfg
  Customers   Product Salary
1     Roohi Product A 514.65
2     James Product B 354.99
3    Satish Product C 345.44
4     Heera Product A 989.56
5   Sehnaaz Product D 767.50
6       Joe Product B 576.90
7       Raj Product D 878.67
8    Simran Product C 904.56
9     Priya Product D 123.45
10  Tejaswi Product A 765.78
>
> # Sorting the dataframe in ascending order
> arrange(gfg, Salary)
  Customers   Product Salary
1     Priya Product D 123.45
2    Satish Product C 345.44
3     James Product B 354.99
4     Roohi Product A 514.65
5       Joe Product B 576.90
6   Tejaswi Product A 765.78
7   Sehnaaz Product D 767.50
8       Raj Product D 878.67
9    Simran Product C 904.56
10    Heera Product A 989.56
>
```

**********output **********

Sorting in Descending order

For sorting our dataframe in descending order, we will use desc() function along with the arrange() function. We will also use % operator for comparison of the dataframe column which we are taking for sorting purpose.

Example: Sorting the dataframe in descending order

```
library(dplyr)

# Creating dataframe
gfg = data.frame(Customers = c("Roohi", "James", "Satish", "Heera",
                               "Sehnaaz", "Joe", "Raj", "Simran",
                               "Priya", "Tejaswi"),

                 Product = c("Product A", "Product B", "Product C",
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
                                        "Product A", "Product D", "Product B",
                                        "Product D", "Product C", "Product D",
                                        "Product A"),

                          Salary = c(514.65, 354.99, 345.44, 989.56, 767.50,
                                        576.90, 878.67, 904.56,123.45, 765.78))


# Sorting the dataframe in descending
# order
gfg %>% arrange(desc(Customers))
```

**********output **********

```
> gfg %>% arrange(desc(Salary))
   Customers   Product Salary
1      Heera Product A 989.56
2     Simran Product C 904.56
3        Raj Product D 878.67
4    Sehnaaz Product D 767.50
5    Tejaswi Product A 765.78
6        Joe Product B 576.90
7      Roohi Product A 514.65
8      James Product B 354.99
9     Satish Product C 345.44
10     Priya Product D 123.45
> gfg %>% arrange(desc(Customers))
   Customers   Product Salary
1    Tejaswi Product A 765.78
2     Simran Product C 904.56
3    Sehnaaz Product D 767.50
4     Satish Product C 345.44
5      Roohi Product A 514.65
6        Raj Product D 878.67
7      Priya Product D 123.45
8        Joe Product B 576.90
9      James Product B 354.99
10     Heera Product A 989.56
>
```

**********output **********

Sorting the dataframe using multiple variables

We will now sort our dataframe using multiple variables using the arrange() function. The attributes should be given to the function separated by a column. For example, in the given example the dataframe is sorted by salary column in descending order and product column in

ascending order. We will use the % operator for comparing data to be sorted in descending order.

Example: Sorting the dataframe using multiple variables

```
library(dplyr)

# Creating dataframe
gfg = data.frame(Customers = c("Roohi", "James", "Satish", "Heera",
                               "Sehnaaz", "Joe", "Raj", "Simran",
                               "Priya", "Tejaswi"),

                 Product = c("Product A", "Product B", "Product C",
                             "Product A", "Product D", "Product B",
                             "Product D", "Product C", "Product D",
                             "Product A"),

                 Salary = c(514.65, 354.99, 345.44, 989.56, 767.50,
                            576.90, 878.67, 904.56,123.45, 765.78))


# Sorting the dataframe in descending
# order
gfg %>% arrange(Product, desc(Salary))
```

**********output **********

```
> gfg %>% arrange(Product, desc(Salary))
   Customers   Product Salary
1      Heera Product A 989.56
2    Tejaswi Product A 765.78
3      Roohi Product A 514.65
4        Joe Product B 576.90
5      James Product B 354.99
6     Simran Product C 904.56
7     Satish Product C 345.44
8        Raj Product D 878.67
9    Sehnaaz Product D 767.50
10     Priya Product D 123.45
>
```

**********output **********

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Subsetting in R Programming

In R Programming Language, subsetting allows the user to access elements from an object. It takes out a portion from the object based on the condition provided. There are 4 ways of subsetting in R programming. Each of the methods depends on the usability of the user and the type of object. For example, if there is a dataframe with many columns such as states, country, and population and suppose the user wants to extract states from it, then subsetting is used to do this operation. In this article, let us discuss the implementation of different types of subsetting in R programming.

R – subsetting

Method 1: Subsetting in R Using [ ] Operator

Using the '[ ]' operator, elements of vectors and observations from data frames can be accessed. To neglect some indexes, '-' is used to access all other indexes of vector or data frame.

Example 1:
In this example, let us create a vector and perform subsetting using the [ ] operator.

```
# Create vector
x <- 1:15

# Print vector
cat("Original vector: ", x, "\n")

# Subsetting vector
cat("First 5 values of vector: ", x[1:5], "\n")

cat("Without values present at index 1, 2 and 3: ",
                                        x[-c(1, 2, 3)], "\n")
```

**********output **********

```
Original vector:  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
First 5 values of vector:  1 2 3 4 5
Without values present at index 1, 2 and 3:  4 5 6 7 8 9 10 11 12 13 14 15
```

**********output **********

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Example 2:

In this example, let us use mtcars data frame present in R base package for subsetting.

```
# Dataset
cat("Original dataset: \n")
print(mtcars)

# Subsetting data frame
cat("HP values of all cars:\n")
print(mtcars['hp'])

# First 10 cars
cat("Without mpg and cyl column:\n")
print(mtcars[1:10, -c(1, 2)])
```

**********output **********

```
Original 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
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

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

HP values of all cars:

	hp
Mazda RX4	110
Mazda RX4 Wag	110
Datsun 710	93
Hornet 4 Drive	110
Hornet Sportabout	175
Valiant	105
Duster 360	245
Merc 240D	62
Merc 230	95
Merc 280	123
Merc 280C	123
Merc 450SE	180
Merc 450SL	180
Merc 450SLC	180
Cadillac Fleetwood	205

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
Lincoln Continental 215
Chrysler Imperial   230
Fiat 128             66
Honda Civic          52
Toyota Corolla       65
Toyota Corona        97
Dodge Challenger    150
AMC Javelin         150
Camaro Z28          245
Pontiac Firebird    175
Fiat X1-9            66
Porsche 914-2        91
Lotus Europa        113
Ford Pantera L      264
Ferrari Dino        175
Maserati Bora       335
Volvo 142E          109


Without mpg and cyl column:
                   disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          167.6 123 3.92 3.440 18.30  1  0    4    4
```

**********output **********

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Method 2: Subsetting in R Using [[ ]] Operator

[[ ]] operator is used for subsetting of list-objects. This operator is the same as [ ] operator but the only difference is that [[ ]] selects only one element whereas [ ] operator can select more than 1 element in a single command.

Example 1: In this example, let us create a list and select the elements using [[]] operator.

```
# Create list
ls <- list(a = 1, b = 2, c = 10, d = 20)

# Print list
cat("Original List: \n")
print(ls)

# Select first element of list
cat("First element of list: ", ls[[1]], "\n")
```

**********output **********

```
Original List:
$a
[1] 1
$b
[1] 2
$c
[1] 10
$d
[1] 20
First element of list:  1
```

**********output **********

Example 2: In this example, let us create a list and recursively select elements using **c()** function.

```
# Create list
z <- list(a = list(x = 1, y = "GFG"), b = 1:10)

# Print list
cat("Original list:\n")
print(z)

# Print GFG using c() function
cat("Using c() function:\n")
print(z[[c(1, 2)]])

# Print GFG using only [[]] operator
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
cat("Using [[]] operator:\n")
print(z[[1]][[2]])
```

**********output **********

```
Original list:
$a
$a$x
[1] 1
$a$y
[1] "GFG"
$b
 [1]  1  2  3  4  5  6  7  8  9 10
Using c() function:
[1] "GFG"
Using [[]] operator:
[1] "GFG"
```

**********output **********

Method 3: Subsetting in R Using $ Operator

$ operator can be used for lists and data frames in R. Unlike [ ] operator, it selects only a single observation at a time. It can be used to access an element in named list or a column in data frame. $ operator is only applicable for recursive objects or list-like objects.

Example 1: In this example, let us create a named list and access the elements using $ operator

```
# Create list
ls <- list(a = 1, b = 2, c = "Hello", d = "GFG")

# Print list
cat("Original list:\n")
print(ls)

# Print "GFG" using $ operator
cat("Using $ operator:\n")
print(ls$d)
```

**********output **********

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
Original list:
$a
[1] 1
$b
[1] 2
$c
[1] "Hello"
$d
[1] "GFG"
Using $ operator:
[1] "GFG"
```

**********output **********

Example 2: In this example, let us use the mtcars dataframe and select a particular column using $ operator.

```
# Dataset
cat("Original data frame:\n")
print(mtcars)

# Access hp column
cat("Using $ operator:\n")
print(mtcars$hp)
```

**********output **********

```
Original data frame:
                   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
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
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


Using $ operator:
[1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230  66  52
[20]  65  97 150 150 245 175  66  91 113 264 175 335 109
```

**********output **********

Method 4: Subsetting in R Using subset() Function

subset() function in R programming is used to create a subset of vectors, matrices, or data frames based on the conditions provided in the parameters.

Syntax: *subset(x, subset, select)*

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Parameters:

- ***x:*** *indicates the object*
- ***subset:*** *indicates the logical expression on the basis of which subsetting has to be done*
- ***select:*** *indicates columns to select*

Example 1: In this example, let us use airquality data frame present in R base package and select Month where Temp < 65.

```
# Subsetting
airq <- subset(airquality, Temp < 65,
                           select = c(Month))

# Print subset
print(airq)
```

**********output **********

```
  Month
4       5
5       5
8       5
9       5
15      5
16      5
18      5
20      5
21      5
23      5
24      5
25      5
26      5
27      5
144     9
148     9
```

**********output **********

Example 2: In this example, let us use mtcars data frame present in R base package and selects the car with 5 gears and hp > 200.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
# Subsetting
mtc <- subset(mtcars, gear == 5 & hp > 200,
                                    select = c(gear, hp))

# Print subset
print(mtc)
```

**********output **********

	gear	hp
Ford Pantera L	5	264
Maserati Bora	5	335

**********output **********

Merge DataFrames by Row Names in R

In this article, we are going to see how to merge Dataframe by Row Name using merge in R Programming Language.

The merge() function in base R can be used to merge input dataframes by common columns or row names. The merge() function retains all the row names of the dataframes, behaving similarly to the inner join. The dataframes are combined in order of the appearance in the input function call.

Syntax: *merge(x, y, by, all)*

Arguments :

- ***x, y –*** *The input dataframes*
- ***by –*** *specifications of the columns used for merging. In case of merging using row names, the by attribute uses 'row.names' value.*
- ***all –*** *logical true or false.*

Example 1: Merge Dataframe

The output displayed is in the order of row numbers of x dataframe followed by y dataframe row numbers.

```
# creating a dataframe
data_frame1 <- data.frame(col1 = c(6:8),
                                    col2 = letters[1:3],
                                    col3 = c(1,4,NA))

print ("Original DataFrame1")
print (data_frame1)
data_frame2 <- data.frame(col1 = c(5:7),
                                    col2 = letters[7:9])
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
print ("Original DataFrame2")
print (data_frame2)
data_frame_merge <- merge(data_frame1, data_frame2,
                          by = 'row.names', all = TRUE)

print ("Merged DataFrame")
print (data_frame_merge)
```

**********output **********

```
[1] "Original DataFrame1"
  col1 col2 col3
1    6    a    1
2    7    b    4
3    8    c   NA
[1] "Original DataFrame2"
  col1 col2
1    5    g
2    6    h
3    7    i
[1] "Merged DataFrame"
  Row.names col1.x col2.x col3 col1.y col2.y
1         1      6      a    1      5      g
2         2      7      b    4      6      h
3         3      8      c   NA      7      i
```

**********output **********

Example 2: Merge unequal Dataframe

In case of the unequal number of row numbers in the dataframes, the dataframe with the lesser number of rows is supplied with NA values, which appear in the merged dataframe.

```
#creating a dataframe
data_frame1 <- data.frame(col1 = c(6:8),
                          col2 = letters[1:3],
                          col3 = c(1,4,NA))

print ("Original DataFrame1")
print (data_frame1)
data_frame2 <- data.frame(col4 = c(5:6),
                          col5 = letters[7:8])
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
print ("Original DataFrame2")
print (data_frame2)
data_frame_merge <- merge(data_frame1, data_frame2,
                          by = 'row.names', all = TRUE)

print ("Merged DataFrame")
print (data_frame_merge)
```

**********output **********

```
[1] "Original DataFrame1"
  col1 col2 col3
1    6    a    1
2    7    b    4
3    8    c   NA
[1] "Original DataFrame2"
  col4 col5
1    5    g
2    6    h
[1] "Merged DataFrame"
  Row.names col1 col2 col3 col4 col5
1         1    6    a    1    5    g
2         2    7    b    4    6    h
3         3    8    c   NA   NA <NA>
```

**********output **********

Joining Data in R with Dplyr Package

In this article, we will be looking at the different methods of joining data with the dplyr in the R programming language.

We need to load the dplyr package. Type the below commands –

Install - install.packages("dplyr")
Load - library("dplyr")
Method 1: Using inner join
In this method of joining data, the user call the inner_join function, which will result to jointed data with the records that have matching values in both tables in the R programming language.

inner_join() function:
This function includes all rows in `x` and `y`.

Name of Student: **Experiment No.**

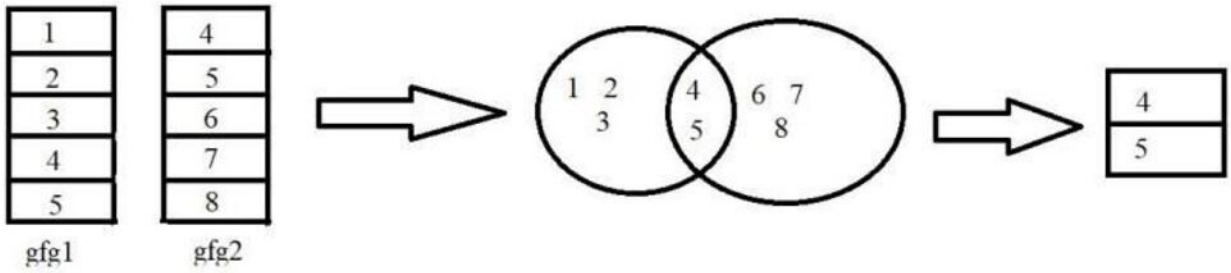
DOP: **DOC:** **Roll No:**

Syntax:

inner_join(x, y, by = NULL, on = NULL)

Parameters:

- *x: A data.table*
- *y: A data.table*
- *by: A character vector of variables to join by.*
- *on: Indicate which columns in x should be joined with which columns in y.*



Example:

In this example, we will be using the inner_join() function from the dplyr package to join two different data as shown in the image above in the R programming language.

```
# load the library
library("dplyr")

# create dataframe with 1 to 5 integers
gfg1 < -data.frame(ID=c(1: 5))

# create dataframe with 4 to 8 integers
gfg2 < -data.frame(ID=c(4: 8))

# perform inner join
inner_join(gfg1, gfg2, by="ID")
```

**********output **********

```
ID

1 4

2 5
```

**********output **********

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Method 2: Using left join

In this method of joining data, the user call the left_join function and this will result to jointed data consisting of matching all the rows in the first data frame with the corresponding values on the second.s in the R programming language.

left_join() function:

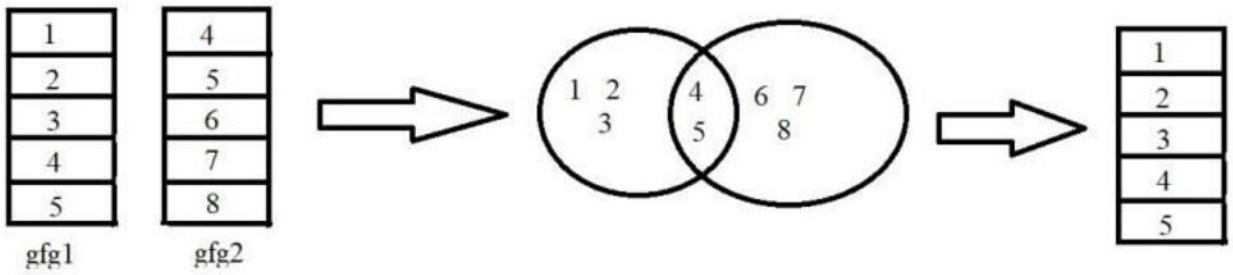
This function includes all rows in `x`.

Syntax:

left_join(x, y, by = NULL, on = NULL)

Parameters:

- *x: A data.table*
- *y: A data.table*
- *by: A character vector of variables to join by.*
- *on: Indicate which columns in x should be joined with which columns in y.*




```
# load the library
library("dplyr")

# create the dataframes
gfg1<-data.frame(ID=c(1:5))

gfg2<-data.frame(ID=c(4:8))

# perform left join
left_join(gfg1,gfg2, by = "ID")
```

**********output **********

```
  ID
1  1
2  2
3  3
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
4  4
5  5
```

**********output **********

Conclusion:-

Name of Student: **Experiment No.**
DOP: **DOC:** **Roll No:**

Experiment No: 6

Aim: Write program to implement the following analysis techniques using R • Statistical hypothesis generation and testing • Chi-Square test • t-Test • Analysis of variance • Correlation analysis • Maximum likelihood test • Regression analysis • Classification techniques • Clustering • Association rules analysis

Theory:

What is meant by statistical hypothesis?

A statistical hypothesis is a hypothesis concerning the parameters or from of the probability distribution for a designated population or populations, or, more generally, of a probabilistic mechanism which is supposed to generate the observations.

What is the objective of hypothesis testing?
The purpose of hypothesis testing is **to make an inference about the population of interest on the basis of a random sample taken from that population**.

What are the 7 steps of hypothesis testing?

The 7 Step Process of Statistical Hypothesis Testing
Step 1: State the Null Hypothesis. ...
Step 2: State the Alternative Hypothesis. ...
Step 3: Set. ...
Step 4: Collect Data. ...
Step 5: Calculate a test statistic. ...
Step 6: Construct Acceptance / Rejection regions. ...
Step 7: Based on steps 5 and 6, draw a conclusion about.

What is the importance of statistical hypothesis?

Hypothesis testing **allows the researcher to determine whether the data from the sample is statistically significant**. Hypothesis testing is one of the most important processes for measuring the validity and reliability of outcomes in any systematic investigation.

What are the five elements of a hypothesis test?
Five Steps in Hypothesis Testing:

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

Specify the Null Hypothesis.

Specify the Alternative Hypothesis.

Set the Significance Level (a)

Calculate the Test Statistic and Corresponding P-Value.

Drawing a Conclusion.

A hypothesis is made by the researchers about the data collected for any experiment or data set. A hypothesis is an assumption made by the researchers that are not mandatory true. In simple words, a hypothesis is a decision taken by the researchers based on the data of the population collected. **Hypothesis Testing** in R Programming is a process of testing the hypothesis made by the researcher or to validate the hypothesis. To perform hypothesis testing, a random sample of data from the population is taken and testing is performed. Based on the results of testing, the hypothesis is either selected or rejected. This concept is known as **Statistical Inference**. In this article, we'll discuss the four-step process of hypothesis testing, One sample T-Testing, Two-sample T-Testing, Directional Hypothesis, one sample -test, two sample -test and correlation test in R programming.

Four Step Process of Hypothesis Testing

There are 4 major steps in hypothesis testing:

- **State the hypothesis-** This step is started by stating null and alternative hypothesis which is presumed as true.
- **Formulate an analysis plan and set the criteria for decision-** In this step, significance level of test is set. The significance level is the probability of a false rejection in a hypothesis test.
- **Analyze sample data-** In this, a test statistic is used to formulate the statistical comparison between the sample mean and the mean of the population or standard deviation of the sample and standard deviation of the population.
- **Interpret decision-** The value of the test statistic is used to make the decision based on the significance level. For example, if the significance level is set to 0.1 probability, then the sample mean less than 10% will be rejected. Otherwise, the hypothesis is retained to be true.

One Sample T-Testing

One sample T-Testing approach collects a huge amount of data and tests it on random samples. To perform T-Test in R, normally distributed data is required. This test is used to test the mean of the sample with the population. For example, the height of persons living in an area is different or identical to other persons living in other areas.

Syntax: *t.test(x, mu)*
Parameters:
x: *represents numeric vector of data*
mu: *represents true value of the mean*

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

To know about more optional parameters of **t.test()**, try below command:
help("t.test")

```
# Defining sample vector
x <- rnorm(100)

# One Sample T-Test
t.test(x, mu = 5)
```

**********output **********

```
One Sample t-test


data:  x
t = -49.504, df = 99, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 5
95 percent confidence interval:
 -0.1910645  0.2090349
sample estimates:
  mean of x
0.008985172
```

Two Sample T-Testing

In two sample T-Testing, the sample vectors are compared. If var.equal = TRUE, the test assumes that the variances of both the samples are equal.

Syntax: *t.test(x, y)*
Parameters:
x *and* ***y:*** *Numeric vectors*

Example:

```
# Defining sample vector
x <- rnorm(100)
y <- rnorm(100)

# Two Sample T-Test
t.test(x, y)
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

**********output **********

```
Welch Two Sample t-test

data:  x and y

t = -1.0601, df = 197.86, p-value = 0.2904

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

 -0.4362140  0.1311918

sample estimates:

  mean of x   mean of y

-0.05075633  0.10175478
```

Directional Hypothesis

Using the directional hypothesis, the direction of the hypothesis can be specified like, if the user wants to know the sample mean is lower or greater than another mean sample of the data.

Syntax: *t.test(x, mu, alternative)*

Parameters:

x: *represents numeric vector data*

mu: *represents mean against which sample data has to be tested*

alternative: *sets the alternative hypothesis*

```
# Defining sample vector
x <- rnorm(100)

# Directional hypothesis testing
t.test(x, mu = 2, alternative = 'greater')
```

**********output **********

```
One Sample t-test


data:  x

t = -20.708, df = 99, p-value = 1

alternative hypothesis: true mean is greater than 2
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
95 percent confidence interval:
 -0.2307534        Inf
sample estimates:
 mean of x
-0.0651628
```

One Sample μ-Test

This type of test is used when comparison has to computed on one sample and the data is non-parametric. It is performed using **wilcox.test()** function in R programming.

Syntax: *wilcox.test(x, y, exact = NULL)*

Parameters:

x *and* ***y:*** *represents numeric vector*

exact: *represents logical value which indicates whether p-value be computed*

To know about more optional parameters of **wilcox.test()**, use below command:
help("wilcox.test")

```
# Define vector
x <- rnorm(100)

# one sample test
wilcox.test(x, exact = FALSE)
```

**********output **********

```
    Wilcoxon signed rank test with continuity correction


data:  x
V = 2555, p-value = 0.9192
alternative hypothesis: true location is not equal to 0
```

Two Sample μ -Test

This test is performed to compare two samples of data.

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
# Define vectors

x <- rnorm(100)

y <- rnorm(100)



# Two sample test
wilcox.test(x, y)
```

**********output **********

```
Wilcoxon rank sum test with continuity correction


data:  x and y

W = 5300, p-value = 0.4643

alternative hypothesis: true location shift is not equal to 0
```

Correlation Test

This test is used to compare the correlation of the two vectors provided in the function call or to test for the association between the paired samples.

Syntax: *cor.test(x, y)*

Parameters:

x *and* ***y:*** *represents numeric data vectors*

To know about more optional parameters in **cor.test()** function, use below command:
help("cor.test")

Example:

Example:

```
# Using mtcars dataset in R
cor.test(mtcars$mpg, mtcars$hp)
```

**********output **********

```
Pearson's product-moment correlation


data:  mtcars$mpg and mtcars$hp

t = -6.7424, df = 30, p-value = 1.788e-07

alternative hypothesis: true correlation is not equal to 0
```

Name of Student: **Experiment No.**

DOP: **DOC:** **Roll No:**

```
95 percent confidence interval:
 -0.8852686 -0.5860994
sample estimates:
       cor
-0.7761684
```

Conclusion:-