Section-IV

Big Data

Assignment 1: Basic R Programming

Introduction:

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

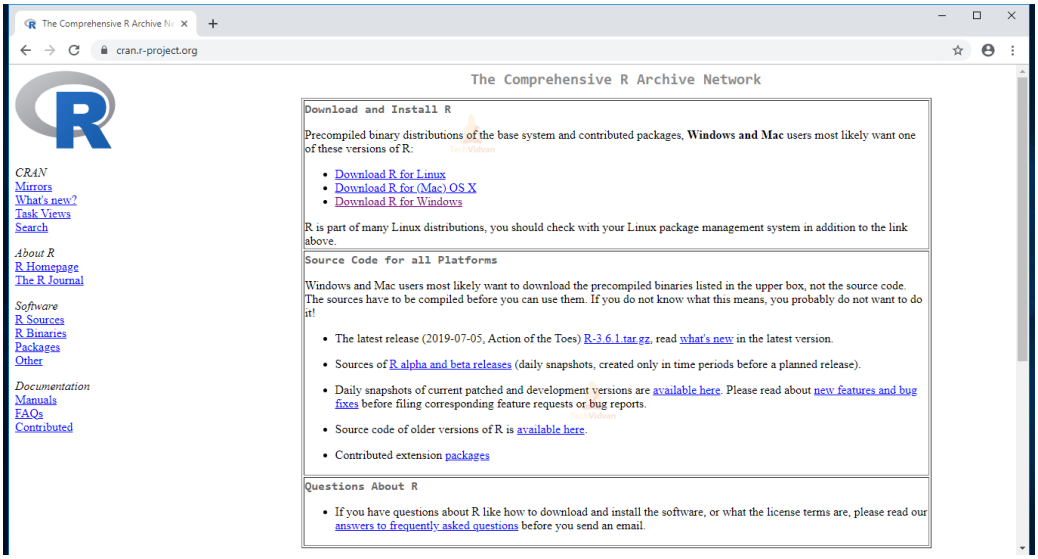
The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.

Installing R and RStudio:

To install R and RStudio on windows, go through the following steps:

Install R on windows

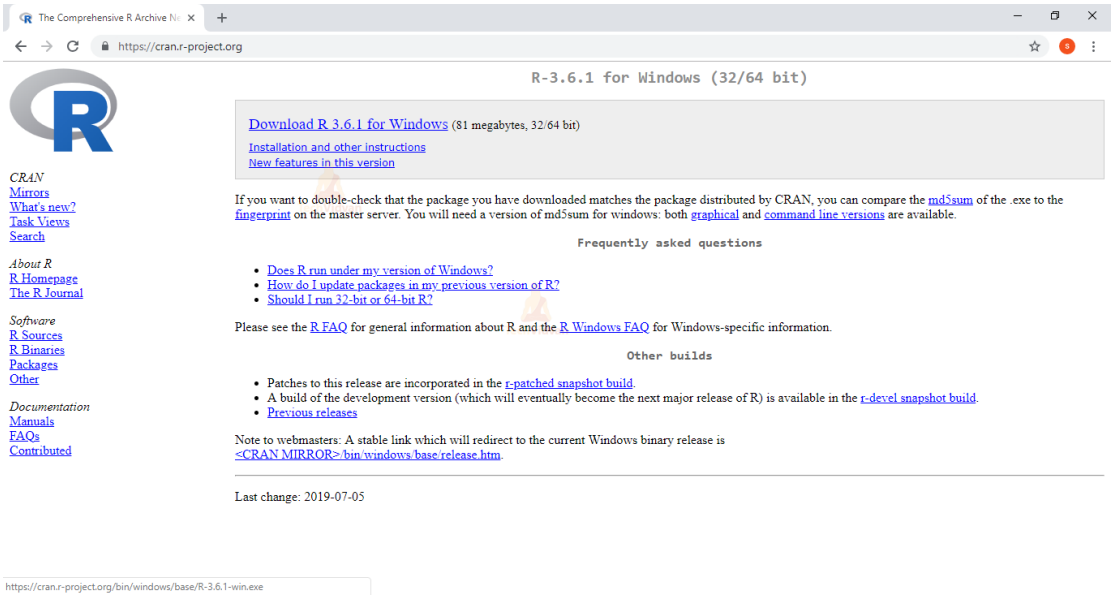
Step – 1: Go to CRAN R project website.



Step – 2: Click on the **Download R for Windows** link.

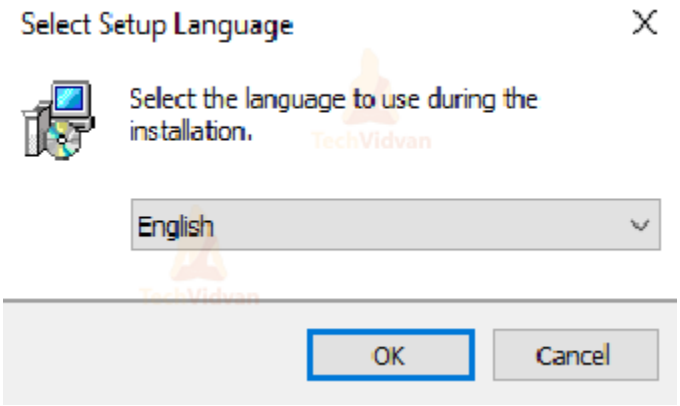
Step – 3: Click on the **base** subdirectory link or **install R for the first time** link.

Step – 4: Click **Download R X.X.X for Windows** (X.X.X stand for the latest version of R. eg: 3.6.1) and save the executable .exe file.

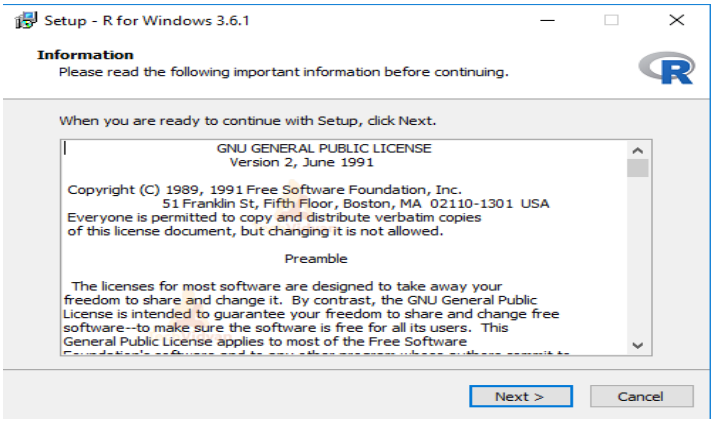


Step – 5: Run the **.exe** file and follow the installation instructions.

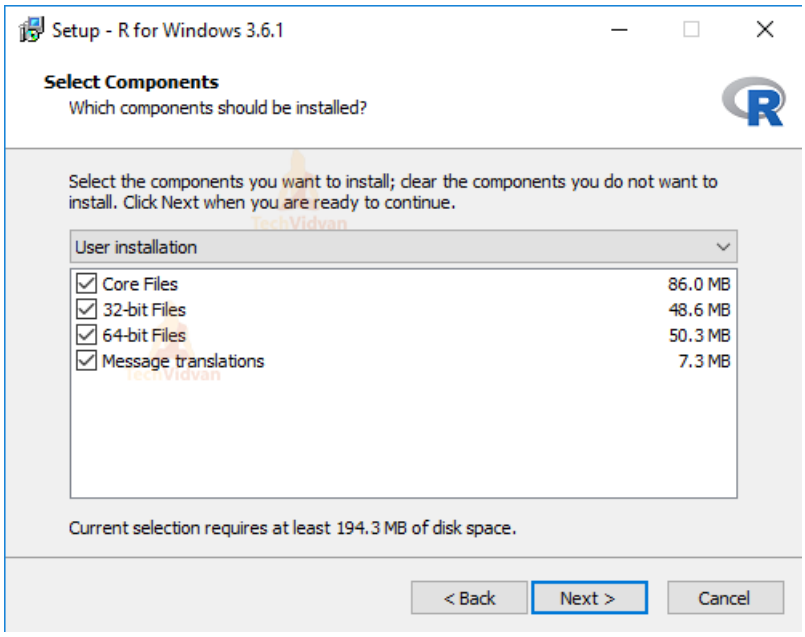
5.a. Select the desired language and then click **Next**.



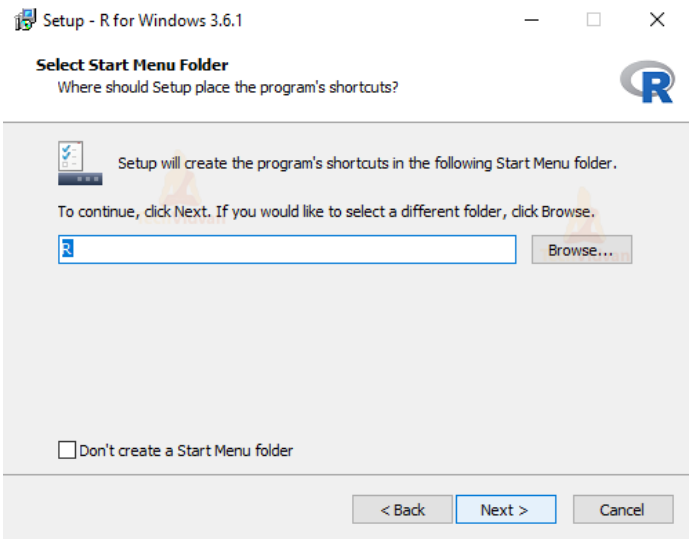
5.b. Read the license agreement and click **Next**.



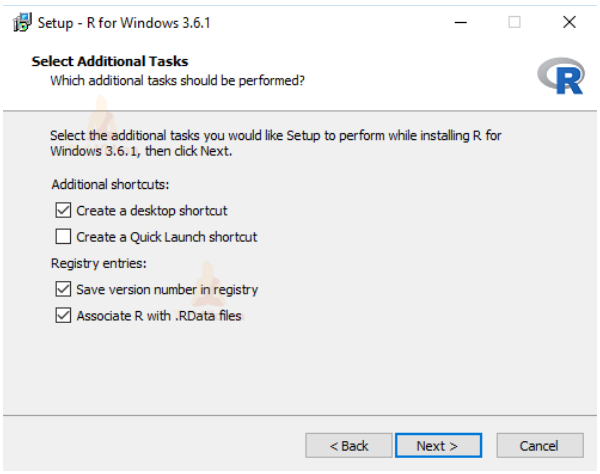
5.c. Select the components you wish to install (it is recommended to install all the components). Click **Next**.



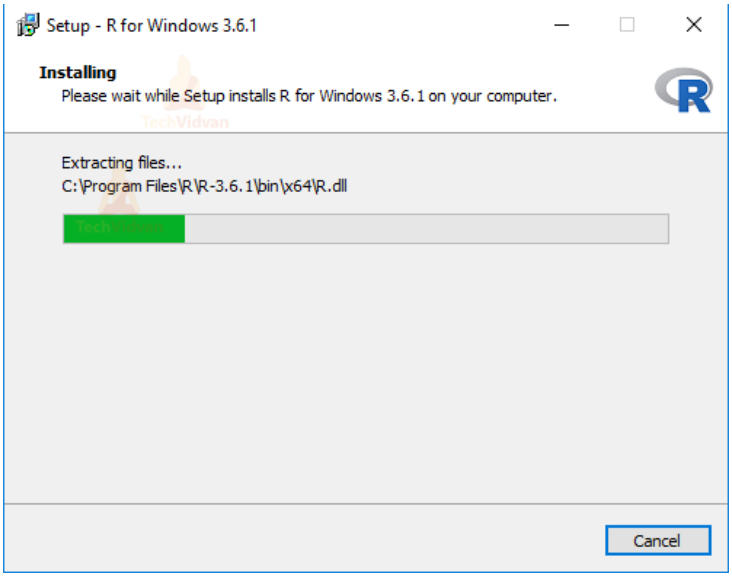
5.d. Enter/browse the folder/path you wish to install R into and then confirm by clicking **Next**.



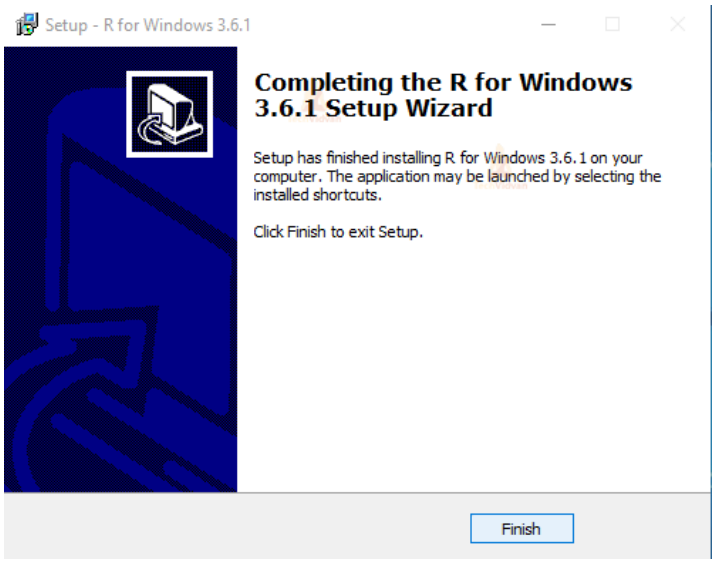
5.e. Select additional tasks like creating desktop shortcuts etc. then click **Next**.



5.f. Wait for the installation process to complete.



5.g. Click on **Finish** to complete the installation.

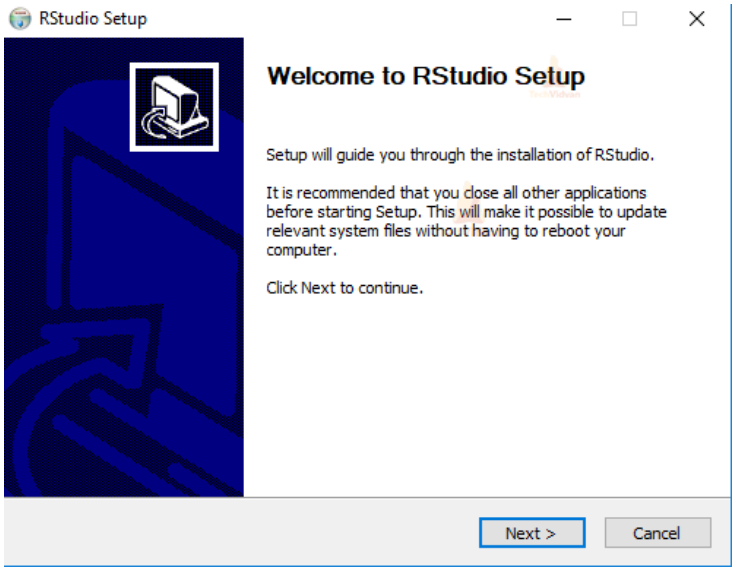


Install RStudio on Windows:

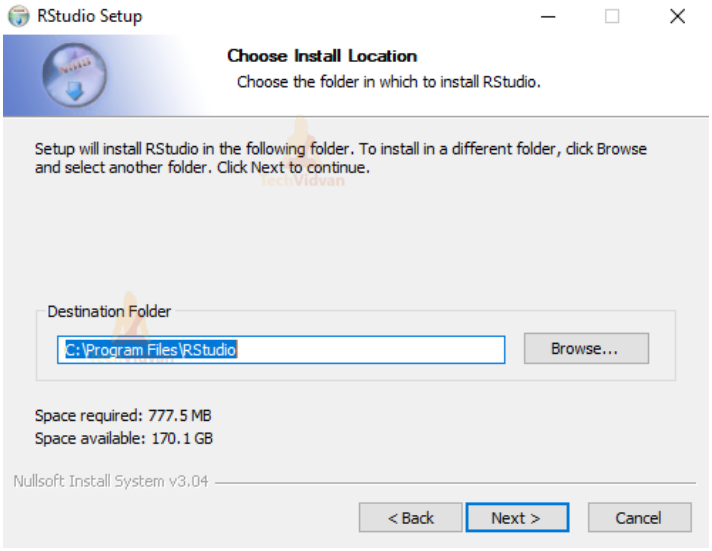
Step – 1: With R-base installed, let's move on to installing RStudio. To begin, go to download RStudio and click on the download button for **RStudio desktop**.

Step – 2: Click on the link for the windows version of RStudio and save the .exe file.
Step – 3: Run the .exe and follow the installation instructions.

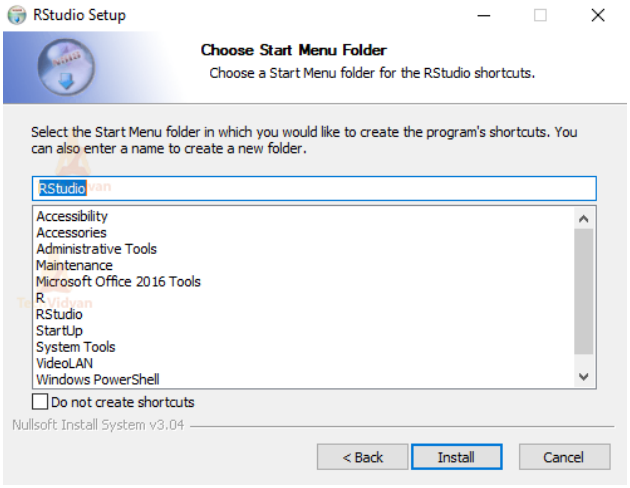
3.a. Click **Next** on the welcome window.



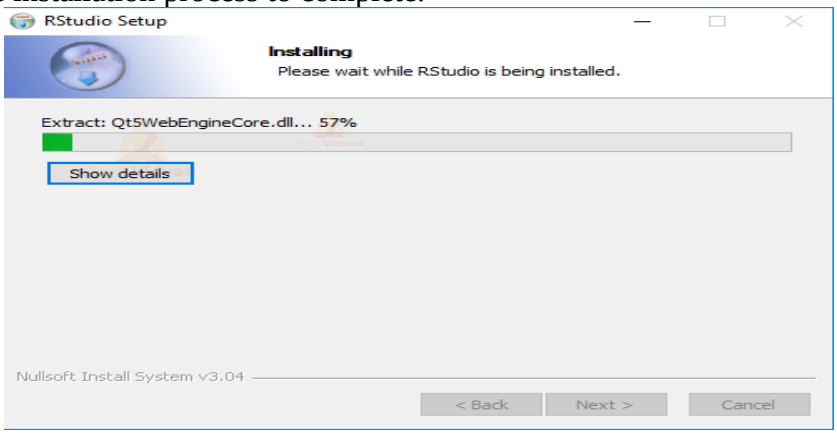
3.b. Enter/browse the path to the installation folder and click **Next** to proceed.



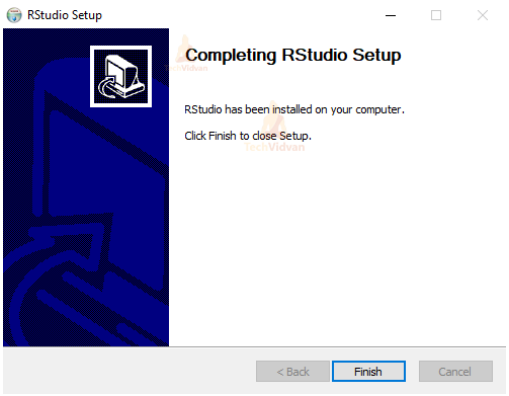
3.c. Select the folder for the start menu shortcut or click on do not create shortcuts and then click **Next**.



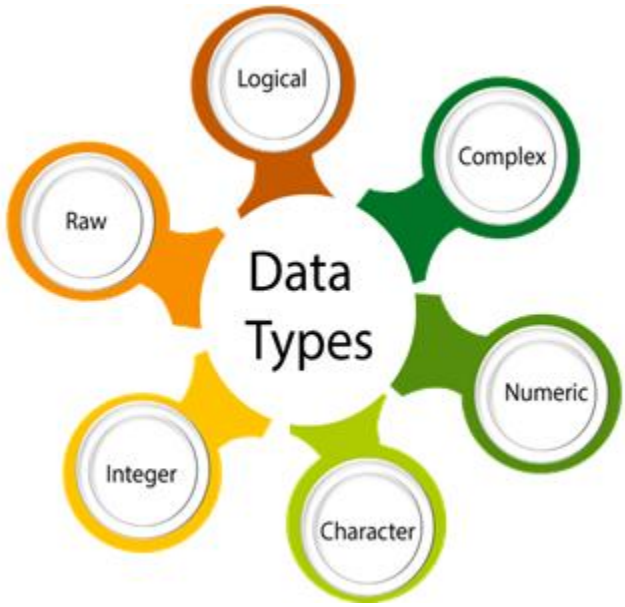
3.d. Wait for the installation process to complete.



3.e. Click **Finish** to end the installation.



Data Types:



1. **Logical:**

It is a special data type for data with only two possible values which can be construed as true/false.

Example:

True, False

2. **Numeric:**

Decimal value is called numeric in R, and it is the default computational data types.

Example:

12,32,112,5432

3. **Integer:**

Here, L tells R to store the value as an integer

Example:

3L, 66L, 2346L

4. **Complex:**

A complex value in R is defined as the pure imaginary value i.

Example:

Z=1+2i, t=7+3i

5. **Character**

In R programming, a character is used to represent string values. We convert objects into character values with the help ofas.character() function.

Example:

'a', '"good"', "TRUE", '35.4'

Variables in R:

Variables are used to store the information to be manipulated and referenced in the R program. The R variable can store an atomic vector, a group of atomic vectors, or a combination of many R objects.

A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.

Example:

Var.1 = c(0,1,2,3)

Operators in R programming:

Operators		**Description**
Arithmetic Operators()	+	Adds two vectors
	-	Subtracts second vector from the first
	*	Multiplies both vectors
	/	Divide the first vector with the second
	%%	Give the remainder of the first vector with the second
	%/%	The result of division of first vector with second (quotient)
	^	The first vector raised to the exponent of second

		vector
Relational Operators	>	Checks if each element of the first vector is greater than the corresponding element of the second vector
	<	Checks if each element of the first vector is less than the corresponding element of the second vector.
	==	Checks if each element of the first vector is equal to the corresponding element of the second vector.
	<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.
	>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.
	!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.
Logical Operators	&&	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.
	||	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.
Assignment Operators	<− or = or <<−	Called Left Assignment
	-> or ->>	Called Right Assignment
Miscellaneous Operators	:	Colon operator. It creates the series of numbers in sequence for a vector.
	%in%	This operator is used to identify if an element belongs to a vector.
	%*%	This operator is used to multiply a matrix with its transpose.

Practice Programs:

1. Write a R program to take input from the user (name and age) and display the values. Also print the version of R installation.
2. Write a R program to get the details of the objects in memory.

SET A:

1. Write a R program to accept dimensions of a cylinder and print the surface area and volume.
2. Write a R program to accept temperatures in Fahrenheit (F) and print it in Celsius(C) and Kelvin (K).
3. Write a R program to accept two numbers and print arithmetic and harmonic mean of the two number.
4. Accept three dimensions length (l), breadth(b) and height(h) of a cuboid and print surface area and volume

SET B:

1. Accept the x and y coordinates of two points and computes the distance between the two points.
2. A cashier has currency notes of denomination 1, 5 and 10. Accept the amount to be withdrawn from the user and print the total number of currency notes of each denomination the cashier will have to give.

SET C:

1. Write a R program to create a sequence of numbers from 20 to 50 and find the mean of numbers from 20 to 60 and sum of numbers from 51 to 91.

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment 2: Decision making and loop control structures

if Statement:

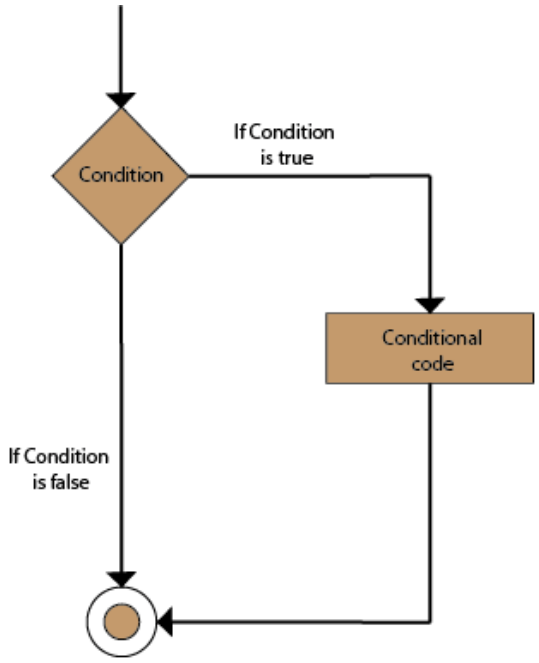
The if statement consists of the Boolean expressions followed by one or more statements. The if statement is the simplest decision-making statement which helps us to take a decision on the basis of the condition.

The if statement is a conditional programming statement which performs the function and displays the information if it is proved true.

The syntax of if statement in R is as follows:

```
if(boolean_expression) {
    // If the boolean expression is true, then statement(s) will be executed.
}
```

Flow Chart:



Example:

```
x <- 5
if(x > 0){
print("Positive number")
```

}

Output
[1] "Positive number"

If-else statement:

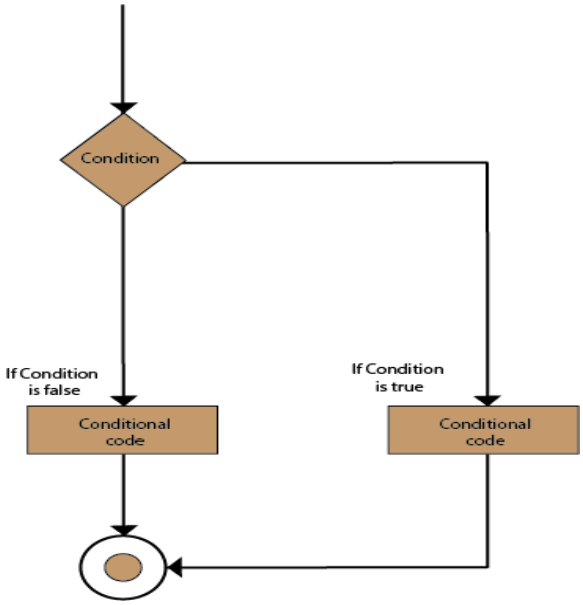
In the if statement, the inner code is executed when the condition is true. The code which is outside the if block will be executed when the if condition is false.

There is another type of decision-making statement known as the if-else statement. An if-else statement is the if statement followed by an else statement. An if-else statement, else statement will be executed when the boolean expression will false. In simple words, If a Boolean expression will have true value, then the if block gets executed otherwise, the else block will get executed.

The basic syntax of If-else statement is as follows:

```
if(boolean_expression) {
        // statement(s) will be executed if the boolean expression is true.
} else {
        // statement(s) will be executed if the boolean expression is false.
}
```

Flow Chart:



Example:

```
x <- -5
if(x > 0){
print("Non-negative number")
} else {
print("Negative number")
}
```

Output:

```
[1] "Negative number"
```

Switch Statement:

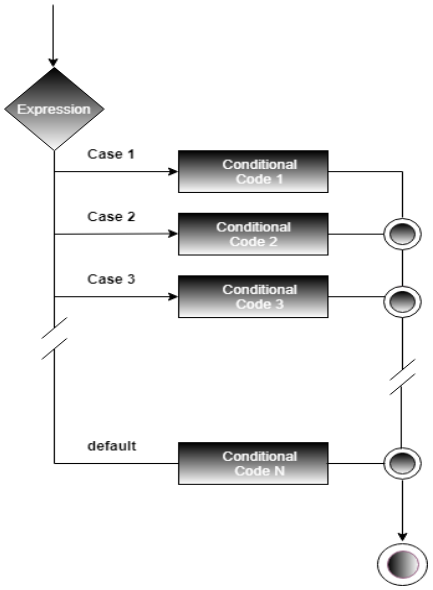
A switch statement is a selection control mechanism that allows the value of an expression to change the control flow of program execution via map and search.

The switch statement is used in place of long if statements which compare a variable with several integral values. It is a multi-way branch statement which provides an easy way to dispatch execution for different parts of code. This code is based on the value of the expression.

The basic syntax of If-else statement is as follows:

switch(expression, case1, case2, case3....)

Flow Chart:



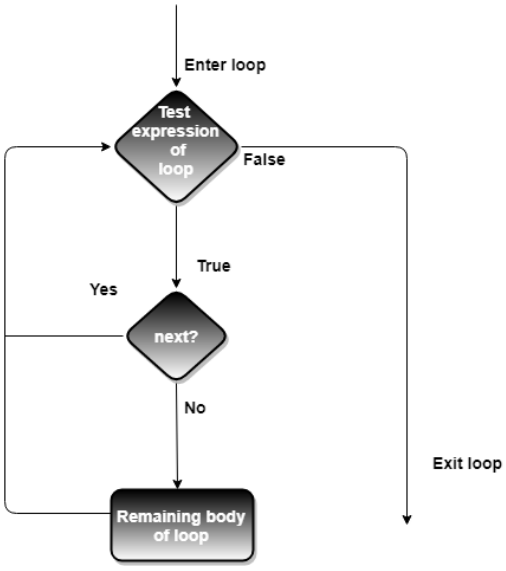
next Statement:

The next statement is used to skip any remaining statements in the loop and continue executing. In simple words, a next statement is a statement which skips the current iteration of a loop without terminating it. When the next statement is encountered, the R parser skips further evaluation and starts the next iteration of the loop.

Syntax

next

Flowchart



Example:

```
x <- 1:5
for (val in x) {
if (val == 3){
        next
}
print(val)
}
```

Output:

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

Break Statement:

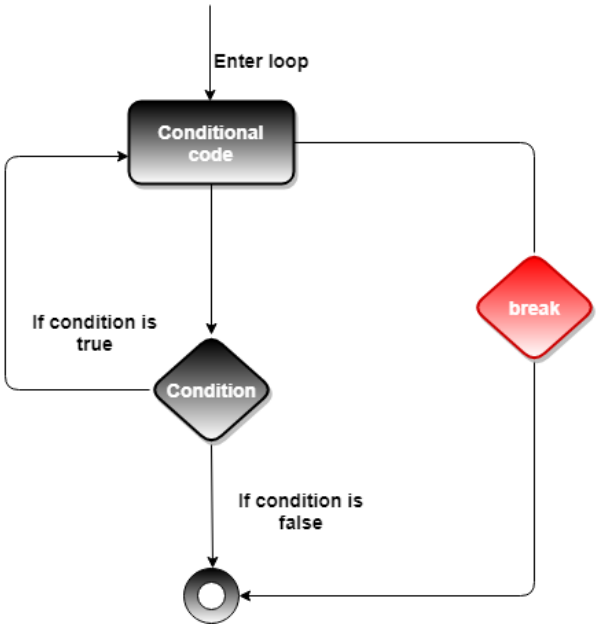
The break statement is used to break the execution and for an immediate exit from the loop. In nested loops, break exits from the innermost loop only and control transfer to the outer loop.

It is useful to manage and control the program execution flow. We can use it to various loops like: for, repeat, etc.

Syntax:

Break

Flowchart:



Example:

```
x <- 1:5
for (val in x) {
if (val == 3){
        break
}
print(val)
}
```

Output:

[1] 1

[1] 2

Loops:

The function of a looping statement is to execute a block of code, several times and to provide various control structures that allow for more complicated execution paths than a usual sequential execution.

Repeat Loop:

A repeat loop is one of the control statements in R programming that executes a set of statements in a loop until the exit condition specified in the loop, evaluates to TRUE.

Syntax

```
repeat{
        Statements
        if(exit_condition){
        break
}
}
```

Example:

```
x <- 1
repeat {
        print(x)
        x = x+1
        if (x == 6){
                break
        }
}
```

Output:

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

While Loop:

A while loop is one of the control statements in R programming which executes a set of statements in a loop until the condition (the Boolean expression) evaluates to TRUE.

```
while(Boolean expression)
{
        Statement
}
```

Example:

```
i<- 1
while (i< 6) {
        print(i)
        i = i+1
}
```

Output:

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

For Loop:

A for loop is the most popular control flow statement. A for loop is used to iterate a vector. It is similar to the while loop. There is only one difference between for and while, i.e., in while loop, the condition is checked before the execution of the body, but in for loop condition is checked after the execution of the body.

Syntax

```
for (value in vector) {
        statements
}
```

Example:

```
x <- c(2,5,3,9,8,11,6)
count<- 0
for (val in x) {
if(val %% 2 == 0)  count = count+1
}
print(count)
```

Output:

```
[1] 3
```

Practice Programs:

1. Write a R program to accept an integer and check if it is even or odd.
2. Write a R program, which accepts annual basic salary of an employee and calculates and displays the Income tax as per the following rules.
   Basic: < 1,50,000 Tax = 0
   Basic: 1,50,000 to 3,00,000 Tax = 20%
   Basic: > 3,00,000 Tax = 30%
3. Writet a R program to accept character from the user and check whether the character is a vowel or consonant.
4. Write a R program accept any year as input and check whether the year is a leap year or not.

SET A:

1. Write a R program to display the first 10 Fibonacci numbers.
2. Write a program to calculate sum of digits of a given input number.
3. Write program to check whether a input number is Armstrong number or not.
4. Write a program to check whether a input number is perfect number of not.
5. Write a R script to display multiplication table of a given input number.

SET B:

1. Write a R program to get all prime numbers up to a given number.
2. Accept the cost price and selling price from the keyboard. Find out if the seller has made a profit or loss and display how much profit or loss has been made.
3. Accept the x and y coordinate of a point and find the quadrant in which the point lies.
4. Write a program to check whether a input number is palindrome or not.
5. Write a program to accept a number and count number of even, odd, zero digits within that number.

SET C:

1. Use a while loop to simulate one stock price path starting at 100 and random normally distributed percentage jumps with mean 0 and standard deviation of 0.01 each period. How long does it take to reach above 150 or below 50?
2. Implement a multiplication game. A while loop that gives the user two random numbers from 2 to 12 and asks the user to multiply them without using * operator.

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment 3: String and Function in R Programming

String:

Any value written within a pair of single quote or double quotes in R is treated as a string. Internally R stores every string within double quotes, even when you create them with single quote.

Example:

```
string1 <- “This is a string”
```

R String Manipulation Functions

1. **grep()**

It is used for pattern matching and replacement. grep, grepl, regexpr, gregexpr and regexec search for matches with argument pattern within each element of a character vector. Here we subsitute the first and other matches with sub and gsub. sub and gsub perform replacement of the first and all matches.

Example:

```
g <- grep("iconHomicideShooting", homicides)
length(g)
```

2. **nchar()**

With the help of this function, we can count the characters. This function consists of a character vector as its argument which then returns a vector comprising of different sizes of the elements of x. nchar is the fastest way to find out if elements of a character vector are non-empty strings or not.

Example

```
# Using nchar() function
nchar("hel'lo")
```

3.substr():

It is the substrings of a character vector. The extractor replaces substrings in a character vector.

4. str_length()

The length of strings indicate the number of characters present in the string.The function **str_length()** belonging to the **‘stringr’** package or **nchar()** inbuilt function of R can be used to determine the length of strings in R.

Example

```
# Importing package
```

```
library(stringr)

# Calculating length of string
str_length("hello")
```

5. cat() function

Different types of strings can be concatenated together using the **cat()**) function in R, where sep specifies the separator to give between the strings and file name, in case we wish to write the contents onto a file.

Syntax:

```
cat(..., sep=" ", file)
```

Example:

```
# Concatenation using cat() function
str<- cat("learn", "code", "tech", sep = ":")
print (str)
```

6. Conversion to upper case

All the characters of the strings specified are converted to upper case.

Example:

```
print(toupper(c("Learn Code", "hI")))
```

7. Conversion to lower case

All the characters of the strings specified are converted to lower case.

Example:

```
print(tolower(c("Learn Code", "hI")))
```

8. Character replacement

Characters can be translated using the chartr(oldchar, newchar, …) function in R, where every instance of old character is replaced by the new character in the specified set of strings.

Example:

chartr("a", "A", "An honest man gave that")

Output:

"An honest mAngAvethAt"

9. Splitting the string

A string can be split into corresponding individual strings using ” ” the default separator.

Example:

strsplit("Learn Code Teach !", " ")

Output:

[1] "Learn" "Code" "Teach" "!"

10. Working with substrings

substr(…, start, end) or substring(…, start, end) function in R extracts substrings out of a string beginning with the start index and ending with the end index. It also replaces the specified substring with a new set of characters.

Example:

substr("Learn Code Tech", 1, 4)

Output:

"Lear"

Function in R:

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

A function should be

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

Syntax

```
func_name<- function (argument) {

        statement

}
```

Example:

```
Pow <- function(x,y){
# function to print x raised to the power y
Result <- x^y
Print(paste(x,"raised to the power", y,"is",result))
}
```

Output:

```
>pow(8,2)
[1] "8 raised to the power 2 is 64"
```

1. Default Values for Arguments

We can assign default values to arguments in a function in R.

Example

```
pow<- function(x, y = 2) {
# function to print x raised to the power y
result<- x^y
print(paste(x,"raised to the power", y, "is", result))
}
```

Output:

```
>pow(3)
[1] "3 raised to the power 2 is 9"
```

3. **Return Value from Function**

Many a times, we will require our functions to do some processing and return back the result. This is accomplished with the return() function in R.

Syntax

```
return(expression)
```

Example:

```
check<- function(x) {
        if (x > 0) {
        result<- "Positive"
        }
        else if (x < 0) {
```

```
        result<- "Negative"
    }
    else {
        result<- "Zero"
    }
        return(result)
    }
```

Output:

```
>check(1)
[1] "Positive"
>check(-10)
[1] "Negative"
```

3. Recursive Function:

A function that calls itself is called a recursive function and this technique is known as recursion.This special programming technique can be used to solve problems by breaking them into smaller and simpler sub-problems.

Example:

```
# Recursive function to find factorial
recursive.factorial<- function(x) {
    if (x == 0)    return (1)
    else          return (x * recursive.factorial(x-1))
}
```

Output:

```
>recursive.factorial(0)
[1] 1
>recursive.factorial(5)
[1] 120
```

In-built Functions:

These functions in R programming are provided by R environment for direct execution, to make our work easier.

Function Name	Description	Example
seq()	To create a sequence of numbers	print(seq(1,9)) O/P [1] 1 2 3 4 5 6 7 8 9
sum()	To find the sum of numbers	print(sum(25,50)) O/P [1] 75
mean()	To find the mean of numbers	print(mean(41:68)) O/P [1] 54.5
paste()	To combine vectors after converting them to characters	paste(1,"sam",2,"rob",3,"max") O/P [1] "1 sam 2 rob 3 max"
min()	To return the minimum value from a vector.	x<-c(3,45,6,7,89,9)print(min(x)) O/P [1] 3
max()	To return the maximum value from a vector	x <- c(3,45,6,7,89,9) print(max(x)) O/P [1] 89

Practice Programs:

1. Write a R program to accept a string from user and display the length of the string.
2. Write a R program to accept a string in lowercase and display it uppercase and vice versa
3. Write a program to check whether a input number is prime number or not using user defined function.

SET A:

1. Write a program to calculate factorial of a input number using user defined function.
2. Write R program to find the factors of a given number using user defined function
3. Write a R Program to connect two different strings.

SET B:

1. Write a program to calculate x^y using user defined function (Use default parameters)
2. Write R program to accept a string and character from user and replace all occurrences of that character from string with other character.
3. Write a recursive function in R to calculate multiplication of all digits of a given input number.

4. Write a function which accepts one number. Function should return 1 if the number is Perfect No, otherwise function should return 0.
5. Write a function isPrime, which accepts an integer as parameter and returns 1 if the number is prime and 0 otherwise.

SET C:

1. Write a R program to print the numbers from 1 to 100 and print "Fizz" for multiples of 3, print "Buzz" for multiples of 5, and print "FizzBuzz" for multiples of both.
2. Write a program to calculate sum of following series up to n terms using user defined function

$$Sum=X+X^2/2!+X^3/3!+\cdots\cdots$$

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment 4: Vector and List in R programming

Introduction:

The Vector is the most basic Data structure in R programming. R Vector can hold a collection of elements of similar types. A vector supports logical, integer, numeric, character, complex, or raw data type. The elements which are contained in vector known as components of the vector. We can check the type of vector with the help of the typeof() function. The length is an important property of a vector. A vector length is basically the number of elements in the vector, and it is calculated with the help of the length () function. Vector is classified into two parts, i.e., Atomic vectors and Lists.

There is only one difference between atomic vectors and lists. In an atomic vector, all the elements are of the same type, but in the list, the elements are of different data types.

Creating vector in R:

In R, we use c () function to create a vector. This function returns a one-dimensional array or simply vector. The c() function is a generic function which combines its argument. All arguments are restricted with a common data type which is the type of the returned value.

For Example:

1. **# Numeric Vector:** a=c(1, 2, 3, 4)
2. **# Character vector:** b=c("India", "China", "Japan", "Russia“)
3. **#Boolean vector:** d=c(TRUE, FALSE, FALSE, TRUE, TRUE)
4. **# Mixed Vector and its Type will be Character** : e= c("India", 2, "China", 1, TRUE)
5. **# Placing or Nesting One Vector inside the another** : f = c("UK", "USA", TRUE, FALSE, b) where b is another Vector

There are various other ways to create a vector in R, which are as follows:

1. **Create R Vector using Range:** In R programming, there is a special operator called Range or Colon, and this will help to create a vector. For example
    1. **# Vector with Range :** i =1 : 10
    2. **Vector with Decimal Range :** j =1.5 : 5.5
    3. **# Vector with Decimal Range :** n = -10 : -20
    4. **Letter Vector with Range:** l =letters[1:6]
2. **Using the seq() function:** In R, we can create a vector with the help of the seq() function. A sequence function creates a sequence of elements as a vector. For example
    1. v1=seq(1,5) #v1 will be 1,2,3,4,5

2. v2=seq(1,10,by=2) #v2 will be 1,3,5,7,9
3. v3=seq(1,4,length.out=6) #v3 will be 1.0,1.6,2.2,2.8,3.4,4.0

Atomic vectors in R:

Atomic vectors are created with the help of **c()** function. In R, there are four types of atomic vectors. These atomic vectors are numeric vector, integer vector, character vector and logical vector.

1. **Numeric vector:** The decimal values are known as numeric data types in R. If we assign a decimal value to any variable d, then this d variable will become a numeric type. A vector which contains numeric elements is known as a numeric vector. For example

```
> d=c(10.5, 24.5,7)
> d                    #prints 10.5 24.5 7.0 at console
>class(d)              #prints  "numeric"
```

2. **Integer Vector:** A non-fraction numeric value is known as integer data. There is two way to assign an integer value to a variable, i.e., by using as.integer() function and appending of L to the value. A vector which contains integer elements is known as an integer vector. For example

```
>int_vec=as.interger(c(1,2,3,4,5) )
>int_vec1=c(1L,2L,3L,4L,5L)
>d=as.integer(5)
>e=5L
```

3. **Character Vector:** In R, there are two different ways to create a character data type value, i.e., using as.character() function and by typing string between double quotes("") or single quotes('). A vector which contains character elements is known as an character vector. For example

```
>f=c("shubham","arpita","nishka","vaishali")
>g=as.character(c(123, 234))
>d='shubham'
>e="Arpita"
```

4. **Logical vector:** The logical data types have only two values i.e., True or False. These values are based on which condition is satisfied. A vector which contains Boolean values is known as the logical vector. For example

```
> a=10
> b=4
> c=8
> log_vec=c(a>b, b<c, c>a, c<a)
> log_vec                    # it prints TRUE TRUE FALSE TRUE
```

Accessing elements of vectors:

We can access the elements of a vector with the help of vector indexing. Indexing denotes the position where the value in a vector is stored. Indexing will be performed with the help of integer, character, or logic.

1. **Indexing with integer vector:** On integer vector, indexing is performed in the same way as we have applied in C. There is only one difference, i.e., in C the indexing starts from 0, but in R, the indexing starts from 1. we perform indexing by specifying an integer value in square braces [] next to our vector. For example

```
> d=c(10,20,30,40,50)
> d                    #Prints 10 20 30 40 50
> d[2]                 # Prints 20
> d[3]                 # Prints 30
> d[2:4]               #Prints 20 30 40
```

2. **Indexing with a character vector:** In character vector indexing, we assign a unique key to each element of the vector. These keys are uniquely defined as each element and can be accessed very easily. For example

```
>  Stud=c("Rollno"=101, "Marks"=80.74)
 > Stud["Rollno"]                  #prints  101
>Stud["Marks"]                          #prints 80.74
>Stud[c("Rollno","Marks")]        #prints 101 80.74
```

3. **Indexing with a logical vector**: In logical indexing, it returns the values of those positions whose corresponding position has a logical vector TRUE. For example

```
> vec=c(1,2,3,4,5,6)
>vec[c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE)]                    #It
```

prints 1 2 5

4. **Access using Vector**: In this example, we will show how to access the Vector elements using another Vector in R. for example

```
 >a =c("India", "China", "Japan", "UK", "USA", "Russia", "Sri Lanka")
> b =c(2, 4, 6)
>print(a[b])                 # It  prints      China  UK      Russia
>print(a[c(5, 7)])           # It  prints USA         Sri Lanka
>print(a[c(7, 4, 1)])        # It  prints Sri Lanka   UK      India
```

5. **Using Negative Values in R:** We can access the Vector elements using Negative values and the Boolean values. In R Vectors, Negative index position is used to omit those values. For example

```
>a=c("India", "China", "Japan", "UK", "USA", "Russia")
>print(a[-3])                #it prints India China  UK      USA    Russia
>b =c(-3, -6)
>print(a[b])                 #it prints India China  UK      USA
```

```
>print(a[c(-4, -6)])          #it prints "India" "China" "Japan" "USA"
```

Manipulate R Vector Elements:

In R Programming, we can manipulate the Vector elements in following ways:

```
>a <- c(10, 20, 30, -15, 40, -25, 60, -5)
>a[7] =77                    #it update the 7th element of vector to 77
>a[a < 0]=99   #it modifies all the elements of vector to 99 if the element is less than 99
>a= a[1:5]     # it truncates all the elements of vector except elements 1 to 5
> a= NULL     #it deletes the vector a
```

Vector Operation:

1. **Combining Vectors:** The c() function is not only used to create a vector, but also it is also used to combine two vectors. By combining one or more vectors, it forms a new vector which contains all the elements of each vector.

```
>a=c(1,2,3)
>b=c("p","q","r")
>c=c(a,b)
>c                          #prints "1" "2" "3" "p" "q" "r“
>d=(b,a)
>d                          #prints "p" "q" "r" "1" "2" "3"
```

2. **Arithmetic operations:** We can perform all the arithmetic operation on vectors. The arithmetic operations are performed member-by-member on vectors. We can add, subtract, multiply, or divide two vectors. For example

```
> a=c(8,4,10)
> b=c(2,6,5)
> a+b                       #10 10 15
> a-b                       #6 -2 5
> a*b                       #16 24 50
> a/b                       #4.0000000 0.6666667 2.0000000
> a%%b                      #0 4 0
> a%/%b                     # 4 0 2
```

Important Functions of Vector:

1. **typeof(Vector):** This method tells you the data type of the vector.
2. **Sort(Vector):** This method helps us to sort the items in the Ascending order.
3. **length(Vector):** This method counts the number of elements in a vector.
4. **head(Vector, limit):** This method return the top six elements (if you Omit the limit). If you specify the limit as 4 then, it returns the first 4 elements.
5. **tail(Vector, limit):** It returns the last six elements (if you Omit the limit). If you specify the limit as 2, then it returns the last two elements.

R List:

Lists are the objects of R which contain elements of different types such as number, vectors, string and another list inside it. It can also contain a function or a matrix as its elements.

A list is a data structure which has components of mixed data types. We can say, a list is a generic vector which contains other objects.

In R, the list is created with the help of list() function.

```
> a_vec=c(1,2,3)
> b_vec=c("Pune","Mumbai")
>list_var=list(2.3,45L,"R Programming", TRUE, a_vec,b_vec)
> print(list_var)
```

Here print(list_var) will display the content of list on console as ouput

Giving a name to list elements:

R provides a very easy way for accessing elements, i.e., by giving the name to each element of a list. After creating list we can assign a name to the list elements with the help of names() function. For example

```
> list_var=list(101L,"Nilesh",80.45)
> names(list_var)=c("Roll No","Name","Marks")
> print(list_var)
        $`Roll No`
                [1] 101
        $Name
                [1] "Nilesh“
        $Marks
                [1] 80.45
```

Accessing List Elements:

R provides two ways through which we can access the elements of a list. First one is the indexing method performed in the same way as a vector. In the second one, we can access the elements of a list with the help of names. It will be possible only with the named list.

```
> list1=list (10, 20, 30)
> list1[1]                    # display  10
> list1[2]                    #display 20
> for(i in list1)
        + print(i)            # display 10 20 30
> list2=list(101,"Nilesh",80.57)
>names(list2)=c("Rollno", "Name", "Marks")
> list2["Rollno"]                    #display $Rllono 101
```

Manipulation of list elements:

R allows us to add, delete or update elements in the list. We can update an element of a list from anywhere, but elements can add only at the end of the list. To remove an element from a specified index, we will assign it a NULL value.

We can update the element of a list by overriding it from the new value.

```
> list1=list (10, 20, 30)
>list1                  #display 10 20 30
>list1 [4] =40
>list1                  #display 10 20 30 40
>list1[2]=200
>list1                  #display 10 200 30 40
>list1[4]=NULL
>list1                  #display 10 200 30
```

Converting list to vector:

There is a drawback with the list, i.e., we cannot perform all the arithmetic operations on list elements. To remove this, drawback R provides unlist() function. This function converts the list into vectors. In some cases, it is required to convert a list into a vector so that we can use the elements of the vector for further manipulation.The unlist() function takes the list as a parameter and change into a vector.

```
> list1=list(2:5)              #list1=(2,3,4,5)
> list2=list(11:14)            #list2=(11,12,13,14)
> list1+list2                  #Error
> v1=unlist(list1)             #converting list1 to vector v1
> v2=unlist(list2)             # converting list1 to vector v2
> v1+v2                        #display 13 15 17 19
```

Merging List:

R allows us to merge one or more lists into one list. Merging is done with the help of the list () function also. To merge the lists, we have to pass all the lists into list function as a parameter, and it returns a list which contains all the elements which are present in the lists.

```
>even=list (2, 4, 6)
>odd=list (1, 3, 5)
> mix=list(even, odd)
> print (mix)                  #display 2 4 6 1 3 5
```

Sample R Scripts using Vector and List

Q.) Write an R program to find the maximum and the minimum value of a given vector.

Solution:

```
n=as.integer(readline(prompt="How many nos do u want to strore in vector="))
```

```
vec=c()
a=1
while(a<=n)
{
        num=as.integer(readline(prompt="Enter Elemnt="))
        vec[a]=num
        a=a+1
}
cat("\n Original Vector=")
print(vec)
cat("\n Maximum elemennt of vector=",max(vec))
cat("\n Minimum elemennt of vector=",min(vec))
```

Q.) Write an R program to sort a list of 10 strings in ascending and descending order.

```
n=as.integer(readline(prompt="How many strings u want to store in list="))
lst=list()

for(a in seq(1,n))
{
  lst[a]=readline(prompt="Enter any String=")
}

b=unlist(lst)
b=sort(b)
lst=list(b)
cat("\n List in Ascending Order=")
print(lst)

b=sort(b,decreasing=TRUE)
lst=list(b)
cat("\n List in Descending Order=")
print(lst)
```

Practice Programs:

1. Write a R program to create a vector of a specified type and length. Create vector of numeric, complex, logical and character types of length 6.
2. Write a R program to add, multiply and divide two vectors of integers type and length 3.
3. Write a R program to create a list containing strings, numbers, vectors and a logical values.

4. Write a R program to list containing a vector, a matrix and a list and give names to the elements in the list. Access the first and second element of the list.

SET A:

1. Write an R program to sort a Vector in ascending and descending order.
2. Write an R program to find Sum, Mean and Product of a Vector.
3. Write a R program to sort a Vector in ascending and descending order.
4. Create a list containing a four vectors and give names to the elements in the list
5. Write a R program to merge two given lists into one list.
6. Write a R program to convert a given list to vector.
7. Write a R program to create a list named s containing sequence of 15 capital letters, starting from 'E'.

SET B:

1. Write a R program to find all elements of a given list that are not in another given list.
2. Write a R program to extract all elements except the third element of the first vector of a given list.
3. Write a script in R to create a list of cities and perform the following
    a. Give names to the elements in the list.
    b. Add an element at the end of the list.
    c. Remove the last element.
    d. Update the 3rd Element
4. Write a script in R to create a list of students and perform the following
    a. Give names to the students in the list.
    b. Add a student at the end of the list.
    c. Remove the first Student.
    d. Update the second last student.
5. Write a script in R to create a vector of numbers and perform the following
    a. Search for specific element
    b. Count the occurrences of specific element
    c. Access the last element of given vector

SET C:

1. Write a R program to extract every n^{th} element of a given vector.
2. Write a R program to select second element of a given nested list.

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment 5: Arrays and Matrices in R programming

R Arrays:

In R, arrays are the data objects which allow us to store data in more than two dimensions. In R, an array is created with the help of the array() function.

Syntax: array_name <- array (data, dim = (row_size, column_size, matrices), dim_names))

Where,

1. **data:** The data is the first argument in the array() function. It is an input vector which is given to the array.
2. **row_size:** This parameter defines the number of row elements which an array can store.
3. **column_size:** This parameter defines the number of columns elements which an array can store.
4. **Matrices:** This parameter defines number of arrays to create.
5. **dim_names:** This parameter is used to change the default names of rows and columns. It is list of 3 vectors, where first vector correspond to row names, second vector represent column names and third vector represent matrix name.

Ex:

```
>d=array(c(1,2,3,4,5,6,7,8,9),dim=c(3,3,1),dimnames=list(c("r1","r2","r3"),c("c1","c2","c3"),c("m1")))
>print(d)
, , m1

   c1 c2 c3
r1  1  4  7
r2  2  5  8
r3  3  6  9
```

Accessing R Array Elements:

In R programming, we can use the index position to access the array elements. Using the index, we can access or alter/change each and every individual element present in an array. Index value starts at 1 and an end at n where n is the size of a matrix, row, or column.

The syntax behind this R Array accessing is: *Array_Name[row_position, Column_Position, Matrix_Level]*.

For example, we declared an array of two matrices of size 6 rows * 4 columns.

To access or alter 1st value use Array_name[1, 1, 1], to access or alter 2nd-row 3rd column value at 1st Matrix level then use Array_name[2, 3, 1]

Ex. >A[2,3,1]=10

Accessing Subset of a Array Elements:

In our previous example, we show you how to access the single element from tan Array. In this example, we will show how to access the subset of multiple items from the Array. To achieve the same, we use the R array A as follows

```
>A =array(1: 24, dim = c(3, 4, 2))
# Access the elements of 1st, 3rd row and 2nd, 4th column in Matrix 1.
        >print(A[c(1, 3), c(2, 4), 1])
# Access all the element of 2nd and 3rd row in Matrix 2.
        >print(A[c(2, 3), , 2])
# Access all the element of 1st and 4th Column in Matrix 1.
        >print(A[ , c(1, 4), 1])
TIP: Negative index position is used to omit those values from an Array.
# Access all the element except 2nd row and 3rd Columm in Matrix 2.
        >print(A[-2, -3, 2])
```

R Array Addition and Subtraction:

In this example, we show how to use Arithmetic Operators on Matrices to perform arithmetic Operations on Array.

```
# Adding and Subtracting Elements of Array in R
    >vect1= c(10, 20, 40 )
    >vect2=c(55, 67, 89, 96, 100)
    >A=array(c(vect1, vect2), dim = c(3, 4, 2))
    >print(A)
    >mat.A=A[, , 1]
    >mat.B=A[, , 2]
    >print(mat.A + mat.B)
    >print(mat.B - mat.A)
```

R Matrix:

The Matrix in R is the most two-dimensional Data structure. In R Matrix, data is stored in row and columns, and we can access the matrix element using both the row index and column index. A matrix is created with the help of the vector input to the matrix function. On R matrices, we can perform addition, subtraction, multiplication, and division operation. In the R matrix, elements are arranged in a fixed number of rows and columns. In R, we use matrix function, which can easily reproduce the memory representation of the matrix. In the R matrix, all the elements must share a common basic type.

R provides the matrix() function to create a matrix.

Syntax of creating Matrix: *matrix(data, nrow, ncol, byrow, dim_name)*

Where,

1. **Data:** The first argument in matrix function is data. It is the input vector which is the data elements of the matrix.
2. **Nrow:** It is the number of rows in the matrix.
3. **Ncol:** It is the number of columns in the matrix.
4. **Byrow:** If its value is true, then the input vector elements are arranged by row.
5. **dim_name:** The dim_name parameter is the name assigned to the rows and columns.

For Example M1= matrix(c (11, 13, 15, 12, 14, 16), nrow =2, ncol =3, byrow = TRUE)

R = matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))

Different ways of creating Matrix in R:

1. # R Create Matrix

   >A=matrix(c(1:12), nrow = 3, ncol = 4) >print(A)
2. # Elements are arranged sequentially by column.

   >B=matrix(c(1:12), nrow = 3, ncol = 4, byrow = FALSE) >print(B)
3. # Elements are arranged sequentially by row.

   >D=matrix(c(1:12), nrow = 3, ncol = 4, byrow = TRUE)
4. # It will create a Matrix of 3 Rows and the remaining elements will be arranged Accordingly

   >A=matrix(c(1:12), nrow = 3)
5. # It will create a Matrix of 4 Columns and the remaining (row) elements will be arranged Accordingly

   >B=matrix(c(1:12), ncol = 4)
6. # It will create a Matrix of 3 rows and 4 Columns

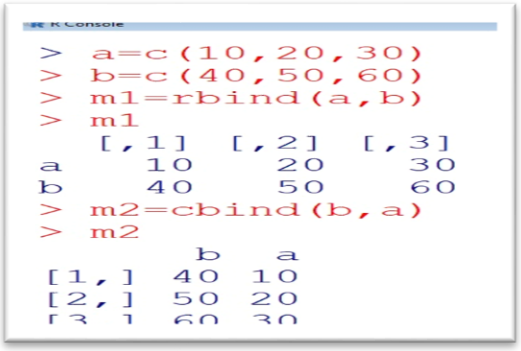
   > D=matrix(c(1:12), 3, 4)
7. # It will create a Matrix of 3 rows

   >E=matrix(c(1:12), 3)
8. # It will create a Matrix of 4 Rows. To create 4 Columns you have to specify *ncol = 4* explicitly

   >G=matrix(c(1:12), ncol=4)

Create R Matrix using cbind and rbind:

In this example, we will show you another way of creating a Matrix in R programming. ***cbind*** is used for binding vectors in Columns wise, and the ***rbind*** is used for binding vectors in Row wise.



Define Row names and Column names for matrix in R:

```
> M1=matrix(c(1,2,3,4,5,6),2,3,byrow=TRUE,dimnames=list(c("A","B"),c("P","Q","R")))
> M1
  P Q R
A 1 2 3
B 4 5 6
>
```

Accessing matrix elements in R:

There are three ways to access the elements from the matrix.

1. We can access the element which presents on nth row and mth column.
2. We can access all the elements of the matrix which are present on the nth row.
3. We can also access all the elements of the matrix which are present on the mth column.

E.g. A=matrix(c(1:12), nrow = 3, ncol = 4, byrow = TRUE)

I. # Access the element at 1st row and 2nd column.
>print(A[1, 2])

II. # Access the element at 3rd row and 4th column.
>print(A[3, 4])

III. # Access only the 2nd row.
>print(A[2,])

IV. # Access only the 4th column.
>print(A[, 4])

V. # Access Complete Matrix.
>print(A[,])

Accessing Subset of a Matrix in R: Following are the examples of accessing subset of a matrix

1. A =matrix(c(1:12), nrow = 3, ncol = 4, byrow = TRUE)
>print(A)
2. # Access the elements at 1st, 2rd row and 3nd, 4th column.

```
>print(A[c(1, 2), c(3, 4)])
```

3. # Access All the element at 2nd and 3rd row.
```
>print(A[c(2, 3), ])
```
4. # Access All the element at 1st and 4th Column.
```
>print(A[ , c(1, 4)])
```
5. # Access All the element except 2nd row.
```
>print(A[-2, ])
```
6. # Access All the element except 2nd row and 3rd Columm.
```
>print(A[-2, -3])
```
7. # Access All the element except 3rd and 4th Columm.
```
>print(A[, c(-3, -4)])
```

Accessing R Matrix Elements using Character Index:

By assigning the Row names and Columns Names can help us to extract the Matrix elements using the Row names or column names as the Index values.

```
> row.names=c("Row1", "Row2", "Row3")            #
>column.names=c("Col1", "Col2", "Col3", "Col4")
>B=matrix(c(1:12), nrow = 3, dimnames = list(row.names, column.names))
```

1. # Access the elements at 1st row and 3rd Column.
```
>print(B["Row1", "Col3"])
```
2. # Access only the 2nd row.
```
>print(B["Row2",])
```
3. # Access only the 4th column.
```
print(B[, "Col4"])
```
4. # Access the elements at 2nd row and 2, 3, 4th Column.
```
>print(B["Row2", 2:4])
```
5. # Access the elements at 1st, 3rd row and 1, 2, 3rd Column.
```
>print(B[c("Row1", "Row3"), 1:3])
```

Modify R Matrix Elements:

In R programming, We can use the index position to modify the elements in a Matrix. Using this index value, we can access or alter/change each and every individual element present in the vector. For example, if we declare a 3 * 4 matrix that stores 12 elements (3 rows and 4 columns). To access or alter 1st value use Matrix.name[1, 1], to access or alter 2nd row 3rd column value use Matrix.name[2, 3].

Modifying Matrix in R Programming

```
>A= matrix(c(1:9), nrow = 3, ncol = 3)
>A[2, 2] =100          #modify second row, second column element to 100
>A[A < 5] =222 # modifies all element to 222 if the element is less than 5
```

Matrix Arithmetic in R:

R Arithmetic Operators are used on Matrices to perform arithmetic Operations.

```
# Create 2x3 matrices.
>a=matrix( c(15, 34, 38, 44, 75, 93), nrow = 2)
>b=matrix( c(10, 20, 30, 40, 50, 60), nrow = 2)
```

1. # Addiing two Matrices
   ```
   >print(a + b)
   ```
2. # Subtraction One Matrix from another
   ```
   >print(a - b)
   ```
3. # R Matrix Multiplication
   ```
   >print(a * b)
   ```
4. # Matrix Division
   ```
   >print(a / b)
   ```

Practice Programs:

1. Write a R program to create an array of two 3x3 matrices each with 3 rows and 3 columns from two given two vectors.
2. Write a R program to create a blank matrix.
3. Write a R program to create a matrix taking a given vector of numbers as input. Display the matrix.
4. Write a R program to create a two-dimensional 5x3 array of sequence of even integers greater than 50.
5. Write a R program to convert a given matrix to a 1 dimensional array.

SET A:

1. Write a R program to create a matrix taking a given vector of numbers as input. Display the matrix.
2. Write a R program to create a matrix taking a given vector of numbers as input and define the column and row names. Display the matrix.
3. Write a R program to access the element at 3rd column and 2nd row, only the 3rd row and only the 4th column of a given matrix.
4. Write an R program to create three vectors a,b,c with 3 integers. Combine the three vectors to become a 3×3 matrix where each column represents a vector. Print the content of the matrix.
5. Write an R program to create a list of elements using vectors, matrices and a functions. Print the content of the list.

SET B:

1. Write a R program to create an array of three 3x2 matrices each with 3 rows and 2 columns from two given two vectors of different length.

2. Write a R program to create an array of two 3x3 matrices each with 3 rows and 3 columns from two given two vectors. Print the second row of the second matrix of the array and the element in the 3rd row and 3rd column of the 1st matrix.
3. Write a R program to access the element at 3rd column and 2nd row, only the 3rd row and only the 4th column of a given matrix.
4. Write a R program to create two 2x3 matrices and add, subtract, multiply and divide the matrix elements.
5. Write an R program to convert a given matrix to a list and print list in ascending order.

SET C:

1. Write a R program to combine three arrays so that the first row of the first array is followed by the first row of the second array and then first row of the third array.
2. Write a R program to find row and column index of maximum and minimum value in a given matrix.

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment 6: Factor and Data Frame in R language

R factors:

Factors are the data objects which are used to categorize the data and store it as levels. In order to categorize the data and store it on multiple levels, we use the data object called R factor. They are useful in the columns which have a limited number of unique values. Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling.

By default, R always sorts levels in alphabetical order.

The command used to create or modify a factor in R language is – factor() with a vector as input.

The two steps to creating a factor are:

1. Creating a vector
2. Converting the vector created into a factor using function factor()

```
# Creating a vector:
        >x=c("female", "male", "male", "female")
        >print(x)               #it prints "female" "male"   "male"   "female"

# Converting the vector x into a factor named gender
        >gender=factor(x)
        >print(gender)          #it prints      [1] female male   male   female
                                                Levels: female male
```

Levels can also be predefined by the programmer. For eample

```
# Creating a factor with levels defined by programmer:
>gender=factor(c("female", "male", "male", "female"), levels = c("female", "transgender", "male"));
```

Further one can check the levels of a factor by using function levels(). Function is.factor() is used to check whether the variable is a factor and returns “TRUE” if it is a factor.

```
        >gender=factor(c("female", "male", "male", "female"));
        >print(is.factor(gender))
```

Function class() is also used to check whether the variable is a factor and if true returns “factor”.

```
        >gender=factor(c("female", "male", "male", "female"));
        >class(gender)
```

Accessing elements of a Factor:

Like we acess elements of a vector, same way we acess the elements of a factor. If gender is a factor then gender[i] would mean acessing i th element in the factor.

Example:

```
        >gender =factor(c("female", "male", "male", "female"))
        >gender[4]                      #It prints   [1] female
                                                       Levels: female male
        >gender[c(2, 4)]                #It prints   [1] male   female
                                                     Levels: female male
```

For selecting all the elements of the factor gender except ith element, gender[-i] should be used.

>gender[-3]

Modification of a Factor

After a factor is formed, its components can be modified but the new values which needs to be assigned must be in the predefined level. For example

>gender[2]<-“female”

Data Frame in R:

The Data Frame in R is a table or two-dimensional data structure. In R Data Frames, data is stored in row and columns, and we can access the data frame elements using the row index and column index. A data frame is a list of variables, and it must contain the same number of rows with unique row names. The Column Names should not be Empty

The data frame’s data can be only of three types- factor, numeric and character type.

Data frame in R is created as follows

```
>Id=c(1:5)
>Name=c(“Nilesh”, “Suresh”, “Ramesh”, “Kamlesh”, “Rajesh”)
>Salary=c(80000, 70000, 90000, 50000, 60000)
>employee=data.frame(Id, Name,Salary)
> print(employee)       will print
        Id    Name Salary
1       1  Nilesh  80000
2       2  Suresh  70000
3       3  Ramesh  90000
4       4 Kamlesh  50000
5       5  Rajesh  60000
```

Create Named Data Frame in R:

```
# We are assigning new names to the Columns
>employee=data.frame("Empid" = Id, "Full_Name" = Name, "income" = Salary)
>print(employee)  will print
        Empid Full_Name income
1       1       Nilesh          80000
2       2       Suresh      70000
3       3       Ramesh     90000
4       4       Kamlesh    50000
5       5       Rajesh      60000

# Names function will display the Index Names of each Item >print(names(employee))
> print(names(employee))
```

[1] "Empid" "Full_Name" "income"

Access R Data Frame Elements:

In R programming, We can access the Data Frame item in multiple ways. In this example, we will show you how to access the data frame items using the index position. Using this index value, we can access each and every item present in the Data Frame. Index value starts at 1 and ends at n where n is the number of items in a data frame. For example

```
>Id=c(1:5)
>Name=c(“Nilesh”, “Suresh”, “Ramesh”, “Kamlesh”, “Rajesh”)
>Salary=c(80000, 70000, 90000, 50000, 60000)>
>employee=data.frame("Empid" = Id, "Full_Name" = Name, “Income" = Salary)
```

1. # Accessing all the Elements (Rows) Present in the Name Items (Column)
   ```
   >employee["Name"]
   ```
2. # Accessing all the Elements (Rows) Present in the 3rd Column (i.e., Occupation)
   ```
   >employee[3]
   ```
3. #Accessing Name column as vector
   ```
   > employee[[“Full_Name"]] or
   >employee[[2]]
   ```
4. # Accessing Element at 1st Row and 2nd Column
   ```
   >employee[1, 2]
   ```
5. # Accessing Element at 4th Row and 3rd Column
   ```
   >employee[4, 3]
   ```
6. # Accessing All Elements at 5th Row
   ```
   >employee[5, ]
   ```
7. # Accessing All Item of the 4th Column
   ```
   >employee[, 4]
   ```

Accessing Multiple Values from R Data:

1. # Accessing Item at 1st, 2nd Rows and 3rd, 4th Columns
   ```
   >employee[c(1, 2), c(3, 4)]
   ```
2. # Accessing Item at 2nd, 3rd, 4th Rows and 2nd, 4th Columns
   ```
   >employee[2:4, c(2, 4)]
   ```
3. # Accessing All Item at 2nd, 3rd, 4th, 5th Rows
   ```
   >employee[2:5, ]
   ```
4. # Accessing All Item of 2nd and 4th Column
   ```
   >employee[ ,c(2, 4)]
   ```
5. # Extract first three rows.
   ```
   >employee[(1:3), ]
   ```
6. #Adding column in data frame

> employee$dept=c(“Comp”, “Math”, “Ele”, “Stat”, “Eng.”)

Access R Data Frame Elements using $:

In R Programming, We can also access the Data frame elements using the **$** dollar symbol.

Ex. >employee$Empid

>employee$Full_Name

1. # Accessing Item at 2nd, 4th Rows of Full_Name Columns
   >employee$Full_Name[c(2, 4)]
2. # Accessing Item at 2nd, 3rd, 4th, 5th Rows of income Column
   >employee$income[2:5]
3. # Accessing Specific columns.
   >data.frame(employee$Full_Name, employee$income)

Modifying R Data Frame Elements:

In R programming, We can access the data frame elements using the index position. Using this index value, we can alter or change each and every individual element present in the data frame. For example

>Id=c(1:5)

>Name=c(“Nilesh”, “Suresh”, “Ramesh”, “Kamlesh”, “Rajesh”)

>Salary=c(80000, 70000, 90000, 50000, 60000)>

>employee=data.frame(Id, Name, Salary)

1. # Modifying Item at 2nd Row and 3rd Column
   >employee[2, 3] =100000
2. # Modifying All Item of 1st Column
   >employee[, 1]=c(10:15)

Adding Elements to Data Frame:

1. **cbind(Data Frame, Values):** This method is used to add extra Columns with values. In general, we prefer Vector as values parameter
   Ex. # Adding Extra Column
   >address=c(“Pimpri”, “Pune”, “Sangvi”, “Kalewadi”, “Nigdi”)
   >cbind(employee, address) where employee is a dataframe

2. **rbind(Data Frame, Values):** This method is used to add extra Row with values.
   Ex. # Adding Extra Row
   >rbind(employee, list(7, “Kamlesh", 8000))

Important Functions of Data Frame in R:

1. **typeof(Data Frame):** This method will tell you the type of Data Frame. Since the data frame is a kind of list, this function will return a list

2. **class(Data Frame):** This method will tell you the class of the Data Frame
3. **length(Data Frame):** This method will count the number of items (columns) in a Data Frame
4. **nrow(Data Frame):** This method will return the total number of Rows present in the Data Frame.
5. **ncol(Data Frame):** This method will return the total number of Columns available in the Data Frame.
6. **dim(Data Frame):** This method will return the total number of Rows and Columns present in the Data Frame.
7. **str(Data Frame):** This method returns the structure of the data present in the Data Frame.
8. **summary(Data Frame):** This R Programming method returns the nature of the data and the statistical summary such as Minimum, Median, Mean, Median, etc.

Use of Head and Tail Functions in R Data Frame:

1. **head(Data Frame, limit):** This method will return the top six elements (if you Omit the limit). If you specify the limit as 2 then, it will return the first 2 records. It is something like selecting the top 10 records.
2. **tail(Data Frame, limit):** This method will return the last six elements (if you Omit the limit). If you specify the limit as 4, it will return the last four records.

1. # No limit - It means Displaying First Six Records
   >head(emp)
2. # Limit is 4 - It means Displaying First Four Records
   >head(emp, 4)
3. # Limit is 10 - It means Displaying First Four Records
   >head(emp, 10)
4. # No limit - It means Displaying Last Six Records
   >tail(emp)
5. # Limit is 4 - It means Displaying Last four Records
   >tail(emp, 4)

Practice Programs:

1. Write a R program to find the levels of factor of a given vector.
2. Write a R program to create a data frame from four given vectors.
3. Write a R program to count the number of NA values in a data frame column.

SET A:

1. Write a R program to change the first level of a factor with another level of a given factor.

2. Write a R program to create a data frame from four given vectors and display the structure and statistical summary of a data frame.
3. Write a R program to display second row using row index and third column using column name of a data frame.
4. Write a R program to create a data frame using two given vectors and display the duplicated elements and unique rows of the said data frame.
5. Write a R program to call the (built-in) dataset airquality. Remove the variables 'Solar.R' and 'Wind' and display the data frame.

SET B:

1. Write an R program to concatenate two given factor in a single factor and display in descending order.
2. Write a R program to extract the five of the levels of factor created from a random sample from the LETTERS.
3. Write a R program to compare two data frames to find the row(s) in first data frame that are not present in second data frame.
4. Write a R program to create a data frame from four given vectors and perform the following
    a. add a new column in a given data frame
    b. add new row to data frame.
    c. drop specific column by name from a given data frame.
    d. drop row by number from a given data frame.
5. Write a R program to create a data frame from four given vectors and perform the following
    a. Extract 3rd and 5th rows with 1st and 3rd columns from a given data frame.
    b. Sort and display given data frame by specific column.

SET C:

1. Write a R program to create inner, outer, left, right join(merge) from given two data frames.
2. Write a R program to save the information of a data frame in a file and display the information of the file.

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment 7: Data Analysis

R CSV Files:

A Comma-Separated Values (CSV) file is a plain text file which contains a list of data. These files are often used for the exchange of data between different applications. These files can sometimes be called character-separated values or comma-delimited files. They often use the comma character to separate data. The idea is that we can export the complex data from one application to a CSV file, and then importing the data in that CSV file to another application. R allows us to read data from files which are stored outside the R environment. The file should be present in the current working directory so that R can read it. We can also set our directory and read file from there.

Getting and setting the working directory:

In R, getwd() and setwd() are the two useful functions.

- The getwd() function is used to check on which directory the R workspace is pointing.
- And the setwd() function is used to set a new working directory to read and write files from that directory.

1. # Getting and printing current working directory.
   ```
   >print(getwd())
   ```
2. # Setting the current working directory.
   ```
   >setwd("C:/Users/ajeet")
   ```

Creating a CSV File:

A text file in which a comma separates the value in a column is known as a CSV file.

Let's start by creating a CSV file with the help of the data from ms excel, which is mentioned below and by saving the file (save as) with .csv extension

```
id,name,marks
1,Nilesh,80.67                    #student.csv file
2,Shubham,90
3,Rajesh,65
```

Reading a CSV file:

R provides read.csv() function, which allows us to read a CSV file available in our current working directory.

Syntax: *data=read.csv(file, header = , sep = , quote =)*

some of the most useful arguments in read csv function:

1. **file:** You have to specify the file name, or Full path along with file name.
2. **header:** If the csv contains Columns names as the First Row then please specify TRUE otherwise, FALSE
3. **sep:** It is a short form of separator. You have to specify the character that is separating the fields. **" , "** means data is separated by comma

4. **quote:** If your character values (FirstName, Education column tc) are enclosed in quotes then you have to specify the quote type. For double quotes we use: quote = **"\""** in r read.csv function
5. **nrows:** It is an integer value. You can use this argument to restrict the number of rows to read. For example, if you want top 5 records, use nrows = 5
6. **skip:** Please specify the number of rows you want to skip from file before beginning the csv read. For example, if you want to skip top 2 records, use skip = 2

R Read csv File from Current Working Directory:

In this example, we will show you, How to read data from the csv (comma separated values) file that is present in the current working directory in R Programming.

```
>setwd("E:\R")
>data <- read.csv("student.csv")
>print(data)
```

Accessing csv file Data:

- In R programming, read.csv function will automatically convert the data into Data Frame.
- So, all the functions that are supported by the Data Frame can be used on csv data.
- While we are working with csv files or read from csv files in R programming, the following functions are the common functions used for data analysis.

1. **max:** This method will return the maximum value within the column
2. **min:** This method will return the minimum value within the column
3. **subset(data, condition):** This method will return the subset of data, and the data depends on the condition.

Examples:

1. # Creating a data frame by reading csv file
   ```
   >csv_data=read.csv("student.csv")
   ```
2. #Accessing all the Elements (Rows) Present in the marks Column
   ```
   >print(csv_data$marks)
   ```
3. # Getting the maximum marks from data frame.
   ```
   >Max_Marks=max(csv_data$marks)
   >print(Max_Marks)
   ```
4. # Accessing Element at 4th Row and 3rd Column
   ```
   >print(csv_data[4, 3] )
   ```
5. # Accessing Item at 1st, 2nd 4th Rows and 4th, 5th, 6th, 7th Columns
   ```
   >csv_data [c(1, 2, 4), c(4:7)]
   ```
6. #Getting the details of the Student who score minimum marks
   ```
   >Details=subset(csv_data, marks==min(marks))
   >print(Details)
   ```

7. #Getting the details of all the students whose name is Nilesh
   ```
   >details=subset(csv_data,name==“Nilesh")
   >print(details)
   ```
8. #Getting the details of all the student whose name is Nilesh and rollno is 5
   ```
   >details=subset(csv_data,rollno==5 & name==“Nilesh”)
   ```
9. #Getting the details of all the students who score more than 60 marks
   ```
   > details=subset(csv_data,marks>60)
   >print(details)
   ```
10. #using inbuilt dataset mtcars find the number of cars of each gear type
    ```
    >data=mtcars
    >f=factor(data$gear)
    >print(table(f))
    ```

11. #using inbuilt dataset mtcars find the number of cars having 3 gear and 2 carburetor
    ```
    >data=mtcars
    >d=subset(data,gear==3 & carb==2)
    >print(nrow(d))
    ```

Data Frame and dplyr package:

The data frame is a key data structure in statistics and in R. dplyr package is very very helpful for managing data frames. The dplyr package was developed by Hadley Wickham of RStudio and is an optimized and distilled version of his plyr package. The dplyr package does not provide any “new” functionality to R, in the sense that everything dplyr does could already be done with base R, but it greatly simplifies existing functionality in R. Filtering, re-ordering, and collapsing, can often be tedious operations in R whose syntax is not very intuitive. The dplyr package is designed to mitigate a lot of these problems and to provide a highly optimized set of routines specifically for dealing with data frames.

Installing the dplyr package:

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

After installing the package it is important that you load it into your R session with the library() function.

```
>library(dplyr)
```

Some of the key functions provided by the dplyr package are:

1. **select:** Select columns with select(). It returns a subset of the columns of a data frame.

   Ex.
   ```
   df=iris
   x<-select(df,c(Species,Sepal.Length))
   head(x)
   ```

2. **filter:** Filter rows with filter().It extracts a subset of rows from a data frame based on logical conditions.

   Ex x<-filter(iris, Sepal.Length > 5.843)
   head(x)

3. **arrange:** Arrange rows with arrange(). It helps to reorder rows of a data frame
   Ex x<-arrange(mtcars, cyl)
   Print(x)
   y<-arrange(mtcars, desc(cyl))
   print(y)
4. **group_by:**
   The group_by() function first sets up how you want to group your data.
   The general operation here is a combination of splitting a data frame into separate pieces defined by a variable or group of variables (group_by()), and then applying a summary function across those subsets (summarize()).
   Ex cyl <- group_by(mtcars, cyl)
   summarise(cyl, mean(disp), mean(hp))

Practice Programs:

1. Using inbuilt dataset women perform the following
   a. display all rows of dataset having height greater than 120
   b. display all rows of dataset in ascending order of weight
2. Using the inbuilt mtcar dataset perform the following
   a. Display all the cars having 4 gears
   b. Display all the cars having 3 gears and 2 carburetor.
3. Using inbuilt PlantGrowth dataset perform the following
   a. Find the flowers of each type of group
   b. Display all rows of type “ctrl” having weight greater than 5.0

SET A:

1. Using the inbuilt mtcar dataset perform the following
   a. Display all the cars having mpg more than 20
   b. Subset the dataset by mpg column for values greater than 15.0.
2. Using the inbuilt airquality dataset perform the following
   a. Find the temperature of day 30 of month 8
   b. Display the details of all the days if the temperature is greater than 90
3. Using the inbuilt airquality dataset perform the following
   a. Subset the dataset for the month July having Wind value greater than 10
   b. Find the number of days having temperature less than 60

SET B:

1. Using iris inbuilt dataset perform the following
   a. Find the flowers of each type of species

b. Find the Sepal length and width of the flower of type setosa having maximum petal length

2. Using iris inbuilt dataset perform the following
    a. Display details of all flowers of type virginica in ascending order of petal length. (use order function)
    b. Display details of first five flowers of type setosa having maximum petal length.
3. Using inbuilt PlantGrowth dataset perform the following
    a. Display details of all plant having weigth greater than 5.80
    b. Display details of all Plants of group trt1 in ascending order of their weight.

SET C:

1. Using inbuilt ToothGrowth dataset perform the following
    a. Find supplement (supp) wise maximum and minimum length of tooth
    b. Display details of first 3 tooth having minimum length for supplement OJ for dose 1.0

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor

Assignment 8: Data Visualization

Introduction:

Data visualization is an efficient technique for gaining insight about data through a visual medium. With the help of visualization techniques, a human can easily obtain information about hidden patterns in data that might be neglected.

By using the data visualization technique, we can work with large datasets to efficiently obtain key insights about it.

In R, we can create visually appealing data visualizations by writing few lines of code.

Advantages of Data Visualization in R:

1. **Understanding:** It is easier to understand through graphics and charts than a written document with text and numbers. Thus, it can attract a wider range of audiences. Also, it promotes the widespread use of business insights that come to make better decisions.
2. **Efficiency:** Its applications allow us to display a lot of information in a small space. Although, the decision-making process in business is inherently complex and multifunctional, displaying evaluation findings in a graph can allow companies to organize a lot of interrelated information in useful ways.

R Bar Charts:

A bar chart is a pictorial representation in which numerical values of variables are represented by length or height of lines or rectangles of equal width. A bar chart is used for summarizing a set of categorical data. In bar chart, the data is shown through rectangular bars having the length of the bar proportional to the value of the variable.In R, we can create a bar chart to visualize the data in an efficient manner.

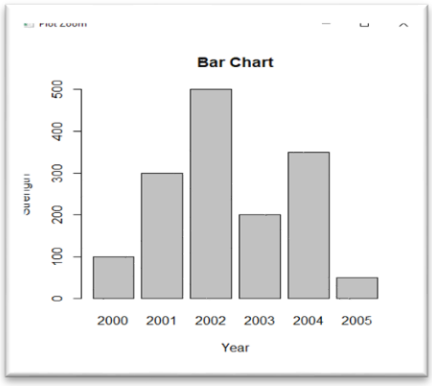
For this purpose, R provides the barplot() function, which has the following syntax:

Syntax: barplot(h, xlab, ylab, main, names.arg, col) where

1. **h:** A vector or matrix which contains numeric values used in the bar chart.
2. **xlab:** A label for the x-axis.
3. **ylab:** A label for the y-axis.
4. **main:** A title of the bar chart.
5. **names.arg:** A vector of names that appear under each bar.
6. **Col:** It is used to give colors to the bars in the graph.

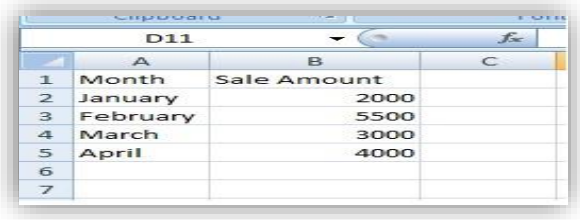
Example: # Creating the data for Bar chart

```
> h=c(100,300,500,200,350,50)
> barplot(h, xlab="Year", ylab="Strength", main="Bar Chart", names.arg = c (2000, 2001, 2002,2003,2004,2005))
```



Creating a barplot in R by reading data from CSV file:

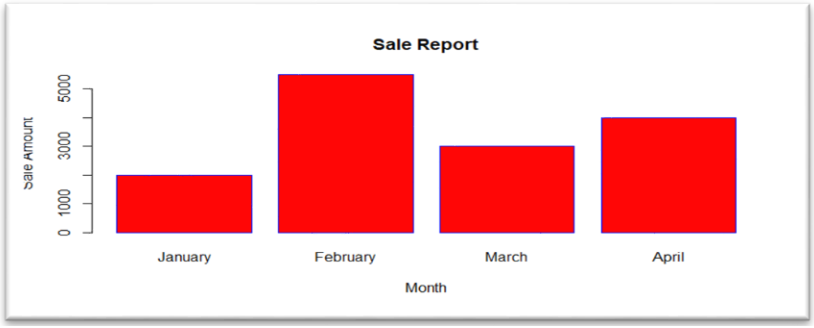
```
>setwd("F:/SYBBA/R/")                          #setting the path where csv file is stored
>d=read.csv("SaleReport.csv", header=TRUE)      #reading csv file data in data frame d
```



	A	B	C
1	Month	Sale Amount	
2	January	2000	
3	February	5500	
4	March	3000	
5	April	4000	
6			
7			

SaleReport.csv File

```
>print(d)                                      #printing data frame
>barplot(d$Sale.Amount, xlab="Month", ylab="Sale Amount", main="Sale Report", names.arg=
d$Month, col="red“, border="blue")
```



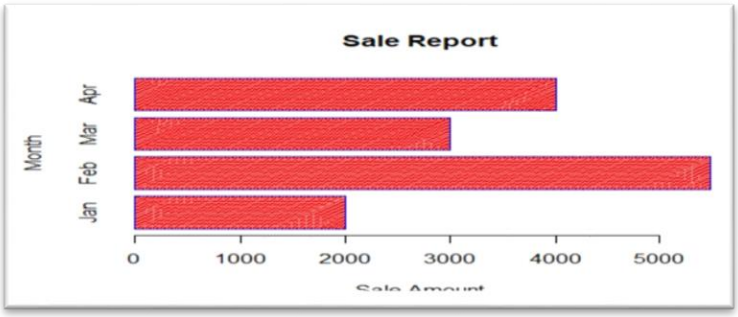
Horizontal Bar Chart in R Programming:

In this example, we change the default vertical bar chart into a horizontal bar chart using ***horiz*** argument in R.

We also change the bar density using ***density*** argument in R barplot

```
>setwd("F:/Yogesh/R/")
>getwd()
>d=read.csv("SaleReport.csv",header=TRUE)
>print(d)
```

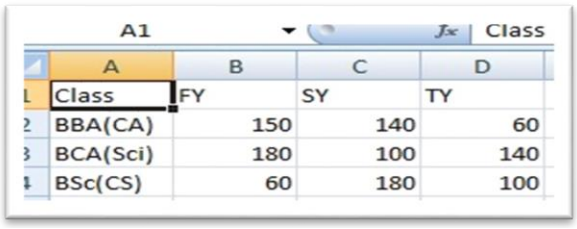
>barplot(d$Sale.Amount, xlab="Month", ylab="Sale Amount", main="Sale Report", names.arg=d$Month, col="red", border="blue", horiz = TRUE, density=100)



Creating Stacked Bar Plot using Matrix:

We can create bar charts with groups of bars and stacks using matrices as input values in each bar. One or more variables are represented as a matrix that is used to construct group bar charts and stacked bar charts.

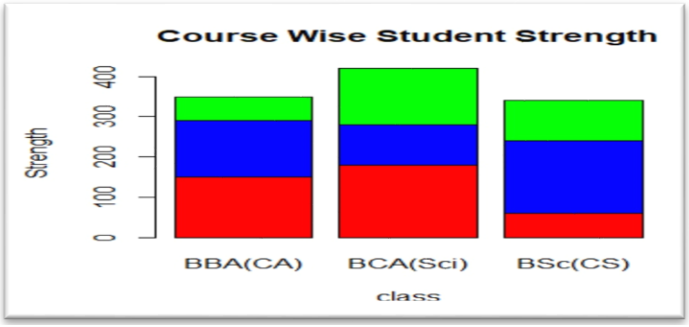
>setwd("F:/Yogesh/R/")
>d=read.csv("class.csv",header=TRUE)
>print(d)



	A	B	C	D
1	Class	FY	SY	TY
2	BBA(CA)	150	140	60
3	BCA(Sci)	180	100	140
4	BSc(CS)	60	180	100

Class.csv File

>fy=d$FY
>sy=d$SY
>ty=d$TY
>data=matrix(c(fy,sy,ty),ncol=3,byrow=TRUE)
>barplot(data, xlab="class", ylab="Strength", main="Course Wise Student Strength", names.arg=d$Class, col= c ("red", "blue", "green"))



R Scatterplots:

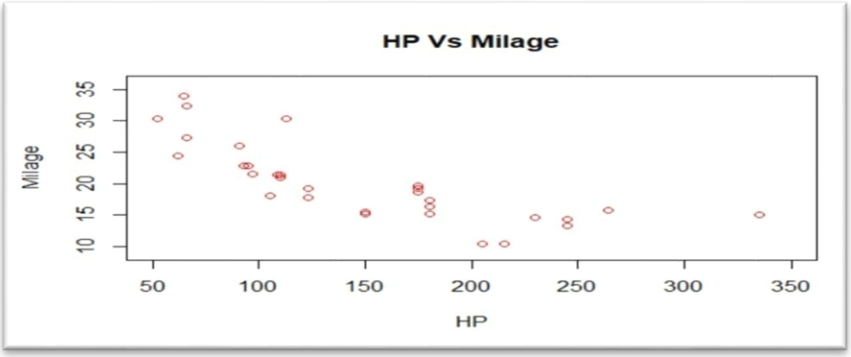
In a scatterplot, the data is represented as a collection of points. Each point on the scatterplot defines the values of the two variables. One variable is selected for the vertical axis and other for the horizontal axis.

The scatter plots are used to compare variables. A comparison between variables is required when we need to define how much one variable is affected by another variable.
Scatterplot is created using plot() function. The syntax is as follows

Syntax: plot(x, y, main, xlab, ylab, xlim, ylim, axes) where,

1. **X**- It is the dataset whose values are the horizontal coordinates.
2. **Y**- It is the dataset whose values are the vertical coordinates.
3. **Main**- It is the title of the graph.
4. **Xlab**- It is the label on the horizontal axis.
5. **Ylab**- It is the label on the vertical axis.
6. **Xlim**- It is the limits of the x values which is used for plotting.
7. **Ylim**- It is the limits of the values of y, which is used for plotting.
8. **axes**- It indicates whether both axes should be drawn on the plot.

Example:
Following scatterplot show the relationship between HP and MPG attribute of mtcars dataset
>plot(mtcars$hp, mtcars$mpg, xlab="HP", ylab="Milage", xlim=c(50,350), ylim=c(9,36), main= "HP Vs Milage", col="red")



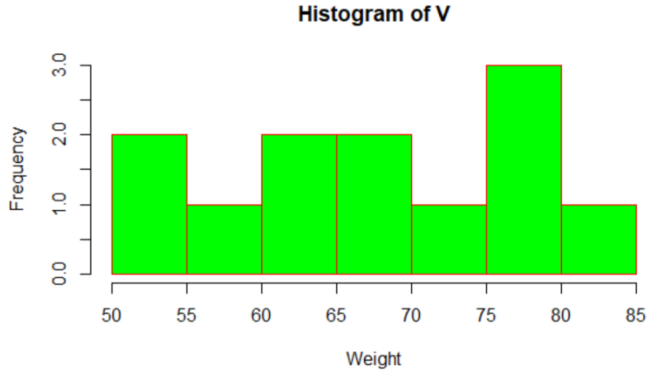
R Histogram:
A histogram is a type of bar chart which shows the frequency of the number of values which are compared with a set of values ranges. The histogram is used for the distribution, whereas a bar chart is used for comparing different entities.
In the histogram, each bar represents the height of the number of values present in the given range.
For creating a histogram, R provides hist() function, which takes a vector as an input and uses more parameters to add more functionality.
There is the following syntax of hist() function:

Syntax: hist(v, main, xlab, ylab, xlim, ylim, breaks, col, border) where

1. **V:** It is a vector that contains numeric values.
2. **Main:** It indicates the title of the chart.
3. **Col:** It is used to set the color of the bars.
4. **Border:** It is used to set the border color of each bar.
5. **Xlab:** It is used to describe the x-axis.
6. **Ylab:** It is used to describe the y-axis.
7. **Xlim:** It is used to specify the range of values on the x-axis.
8. **Ylim:** It is used to specify the range of values on the y-axis.
9. **Breaks:** It is used to mention the width of each bar.

Example: Consider Vector V which consists of weight of different students

```
> V=c(55,67,78,82,57,62,74,80,52,64,76,66)
>hist(v, xlab = "Weight", ylab="Frequency", col = "green", border = "red")
```



R Boxplot:

Boxplots are a measure of how well data is distributed across a data set. This divides the data set into three quartiles. This graph represents the minimum, maximum, average, first quartile, and the third quartile in the data set. Boxplot is also useful in comparing the distribution of data in a data set by drawing a boxplot for each of them.

R provides a boxplot() function to create a boxplot. There is the following syntax of boxplot() function

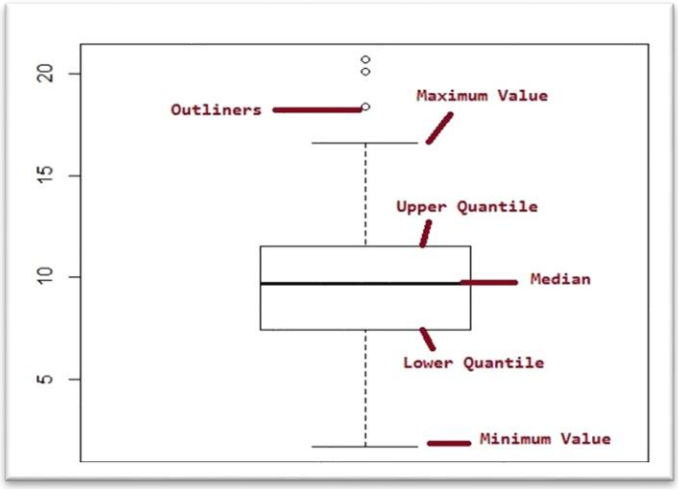
Syntax: boxplot(data or formula, xlab, ylab, main, names, col) where,

1. **data:** DataFrame, or List that contains the data to draw boxplot.
2. **Xlab:** It is used to describe the x-axis.
3. **Ylab:** It is used to describe the y-axis.
4. **Main:** It is used to give a title to the graph.
5. **Names:** It is the group of labels that will be printed under each boxplot.

Creating a Boxplot in R Programming:

In this example, we create a Boxplot using the ***airquality*** data set

```
>a=airquality
> boxplot(a$Wind)
```



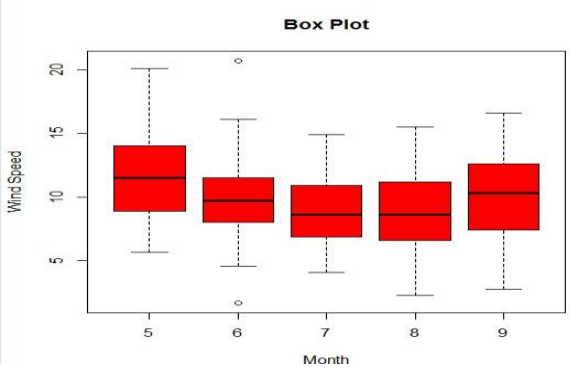
Use Formula to create a Boxplot in R:

In this example, we create a Boxplot using the formula argument

formula: It should be something like ***value~group,*** where value is the vector of numeric values, and the group is the column you want to use as a group by.

e.g., if you want to draw a boxplot for Monthwise wind speed, then ***value*** = Wind and ***group*** = Month

```
>a=airquality
>boxplot(a$Wind~a$Month, xlab="Month", ylab="Wind Speed", main="Box Plot",
col="red")
```



Practice Programs:

1. Write an R program to draw an empty plot and an empty plot specifies the axes limits of the graphic.
2. Using inbuilt airquality dataset make a scatter plot to compare Wind speed and temperature.
3. Using inbuilt iris dataset create Histogram for Petal.length values
4. Using iris dataset draw horizontal bar plot for Petal length values for species setosa

SET A:

1. Using inbuilt mtcars dataset
    a) Create a bar plot for attribute mpg for all cars having 3 gears
    b) Create a Histogram to show number of cars per carburetor type whose mpg is greater than 20
2. Using airquality dataset
    a) Create a scatter plot to show the relationship between ozone and wind values by giving appropriate value to color argument
    b) Create a bar plot to show the ozone level for all the days having temperature greater than 70

SET B:

1. Using inbuilt mtcars dataset
    a. Create a bar plot that shows the number of cars of each gear type.

   b. Draw a scatter plot showing the relationship between wt and mpg for all the cars having 4 gears
2. Using airquality dataset
   a. Show the statistical summary using box plot for Temprature value of month June
   b. Using histogram show the frequency of number of days for Temp values of month August

SET C:

1. Using inbuilt mtcars dataset show a stacked bar graph of the number of each gear type and how they are further divided out by cyl
2. Draw boxplot to show the distribution of mpg values per number of gears

Assignment Evaluation

0: Not Done [ ]	**1: Incomplete [ ]**	**2: Late Complete [ ]**
3: Need Improvement [ ]	**4: Complete [ ]**	**5: Well Done [ ]**

Signature of Instructor