Contents to be covered

- Introduction to R
- Installation of R and setting Environment Variable
- Basic Types

Introduction

- R is an open-source programming language mostly used by statisticians and data engineers who utilize it to build various algorithms and techniques for statistical modeling and data analysis.
- 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.
- R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.
- This programming language was named **R**, based on the first letter of first name of the two R authors (Robert Gentleman and Ross Ihaka).

Introduction

- It first came into the picture in August 1993.
- R includes a ton of inbuilt libraries that offer a wide variety of statistical and graphical techniques which include regression analysis, statistical tests, classification models, clustering and time-series analysis.
- R language runs on the R Studio platform which helps in initiating and executing codes and packages in R.
- It is heavily used in analyzing data that is both structured and unstructured.

Features of R Programming

- R is a programming language that provides support for procedural programming involving functions as well as object-oriented programming with generic functions.
- There are more than 10,000 packages in the repository of R programming. With these packages, one can make use of functions to facilitate easier programming.
- Being an interpreter based language, R produces a machine-independent code that is portable in nature.
- R facilitates complex operations with *vectors, arrays, data frames as well as other data objects* that have varying sizes.
- R can be easily integrated with many other technologies and frameworks like Hadoop and HDFS. It can also integrate with other programming languages like C, C++, Python, Java, FORTRAN, and JavaScript.

R Scripts

- R is the primary statistical programming language for performing modeling and graphical tasks.

- Editors and IDEs that facilitate GUI features for executing R scripts are:
  - **RGui (R Graphical User Interface)**
  - **Rstudio**

R Graphical User Interface (R GUI)

- R GUI is the standard GUI platform for working in R.
- The R Console Window forms an essential part of the R GUI. In this window, we input *various instructions, scripts and several other important operations.*
- In the main panel of R GUI,
  - go to the ‘**File**‘ menu and
  - select the ‘**New Script**‘ option. This will create a new script in R.
- In order to quit the active R session, you can type the following code after the R prompt ‘>’ as follows:
  - > q()

R Studio

- RStudio is an integrated and comprehensive **Integrated Development Environment** for R.

- It facilitates extensive code editing, development as well as various features that make R an easy language to implement.

Features

- RStudio provides various tools and features that allow you to boost your code productivity.

- It can also be accessed over the web and is cross-platform in nature.

R Studio

Components of RStudio

- **Source** – In the top left corner of the screen is the text editor that allows you to work within source scripting. Users can save the R scripts to files that are stored in local memory.
- **Console** – This is present on the bottom left corner of the main window of R Studio. It facilitates interactive scripting in R.
- **Workspace and History** – In the top right corner, you will find the R workspace and the history window. This will give you the list of all the variables that were created in the environment session. Furthermore, you can also view the list of past commands that were executed by R.

- Files, Plots, Package, and Help at the bottom right corner gives access to the following tools:
  - **Files –** A user can browse the various files and folders on a computer.
  - **Plots –** We obtain the user plots here.
  - **Packages –** we can view the list of all the installed packages.
  - **Help –** We can browse the built-in help system of R with this command.

Sourcing a Script in R

- While R console provides an interactive method to perform R programming.
- R Studio also provides various features to develop a script in the external editors and source the script into the console.
- You can source either selected lines or the entire code using R GUI and R Studio.
- An advantage of writing into the R editor is that multiple lines can be written at once without prompting R to evaluate them individually.

Source the script in the following ways:

- In order to execute a selected line of code:
- Select the line(s) of code, then press **Ctrl + R** in R GUI and **Ctrl + Enter** in RStudio.

Sourcing a Script in R

For example, we have two lines of code as follows:

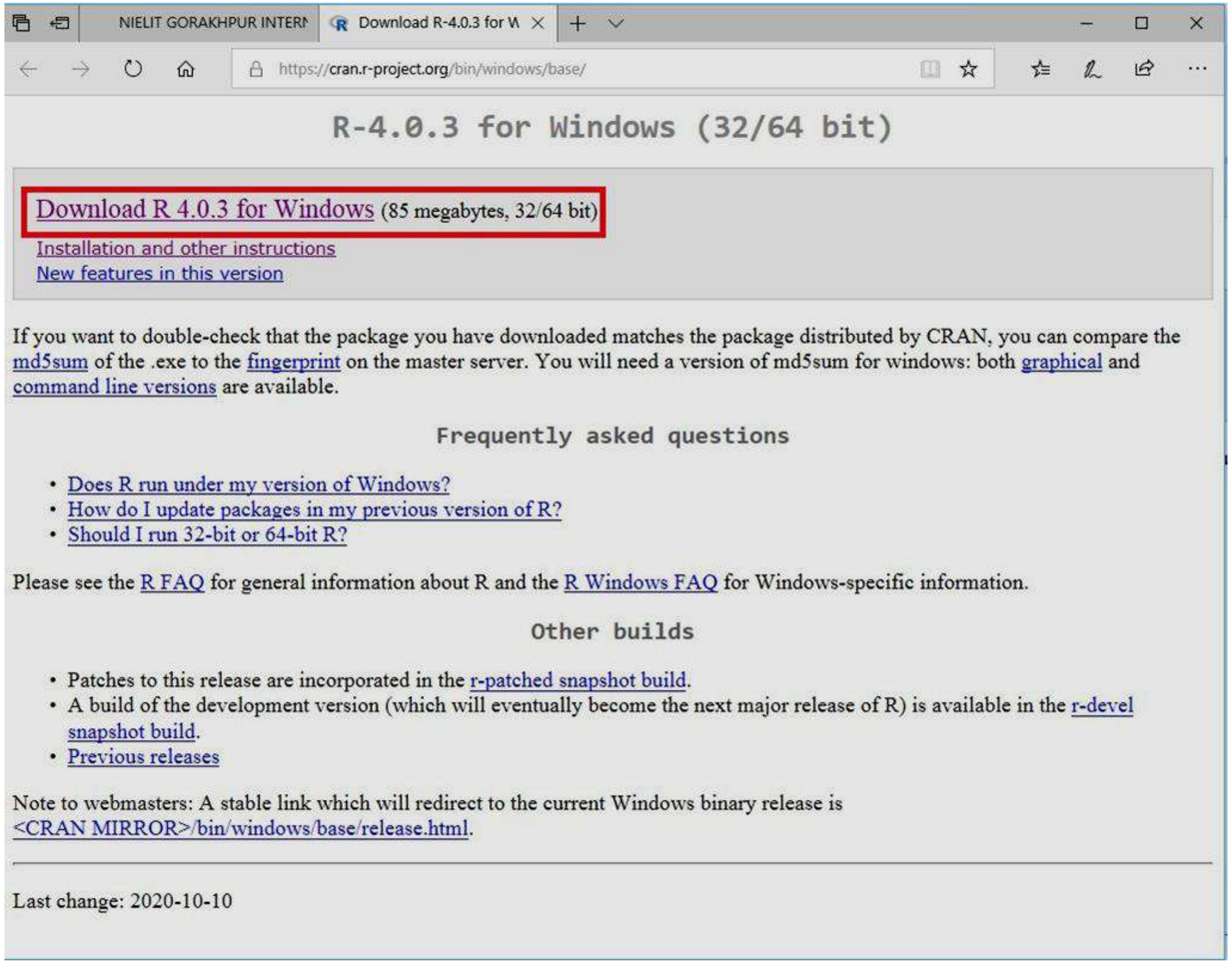
- print("Hello")
- print(“NIELIT Gorakhpur")

- In the above code, if you only want to print “Hello”,
  - then select only the first line and press **Ctrl + Enter** in RStudio.

- In order to execute the entire script:
  - **In R GUI**
    Go to Edit, and then click **Run All**.
  - **In the case of R Studio**
    Hold and press **Ctrl+Shift+ Enter**.

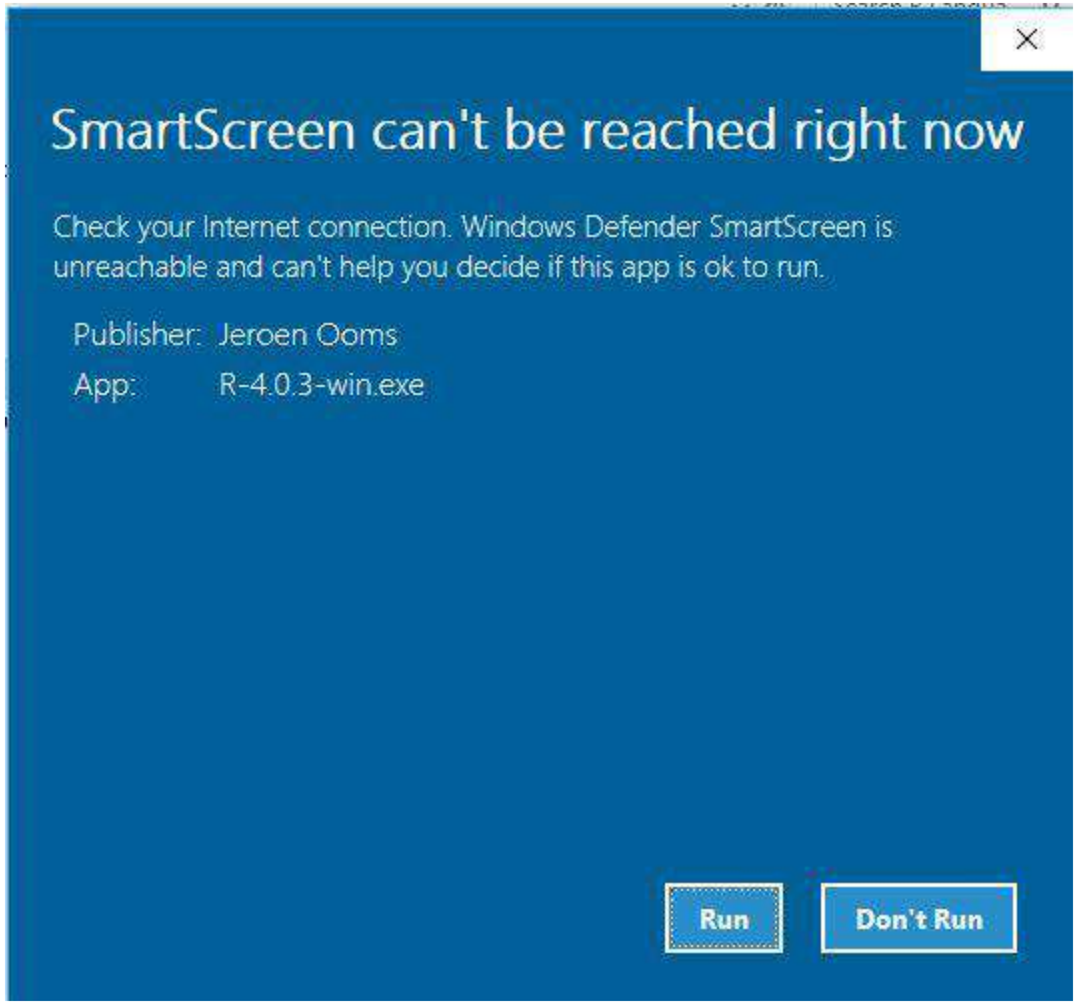
Installation of R Programming

- Go to: https://cran.r-project.org/bin/windows/base/
- Click the “**Download R 4.0.3 for Windows**” button, save it and run.
- Follow the step by step process of installation wizard.



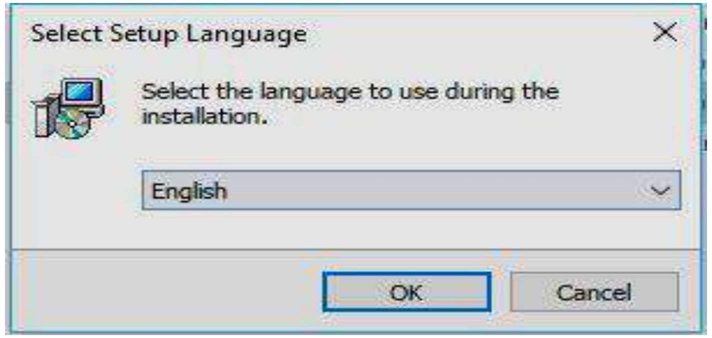
Installation of R Programming

- An **Open File - Security Warning** pop-up window will appear.
- Click **Run**. A **R-4.0.3 win.exe Setup** pop-up window will appear.



Installation of R Programming

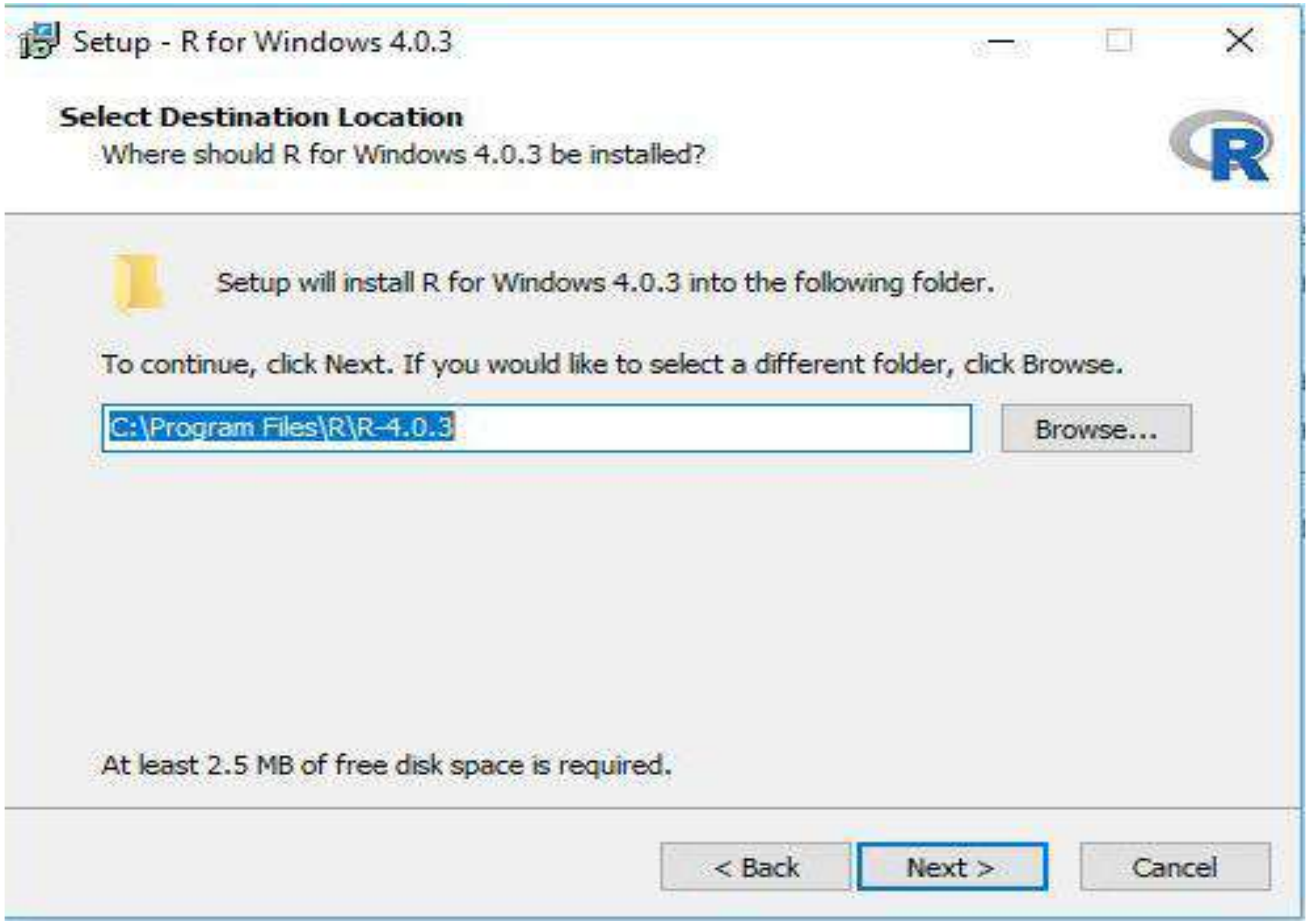
- Select the Language and Click on **OK** button



- Read the **License Agreement**, then Click on **Next** button

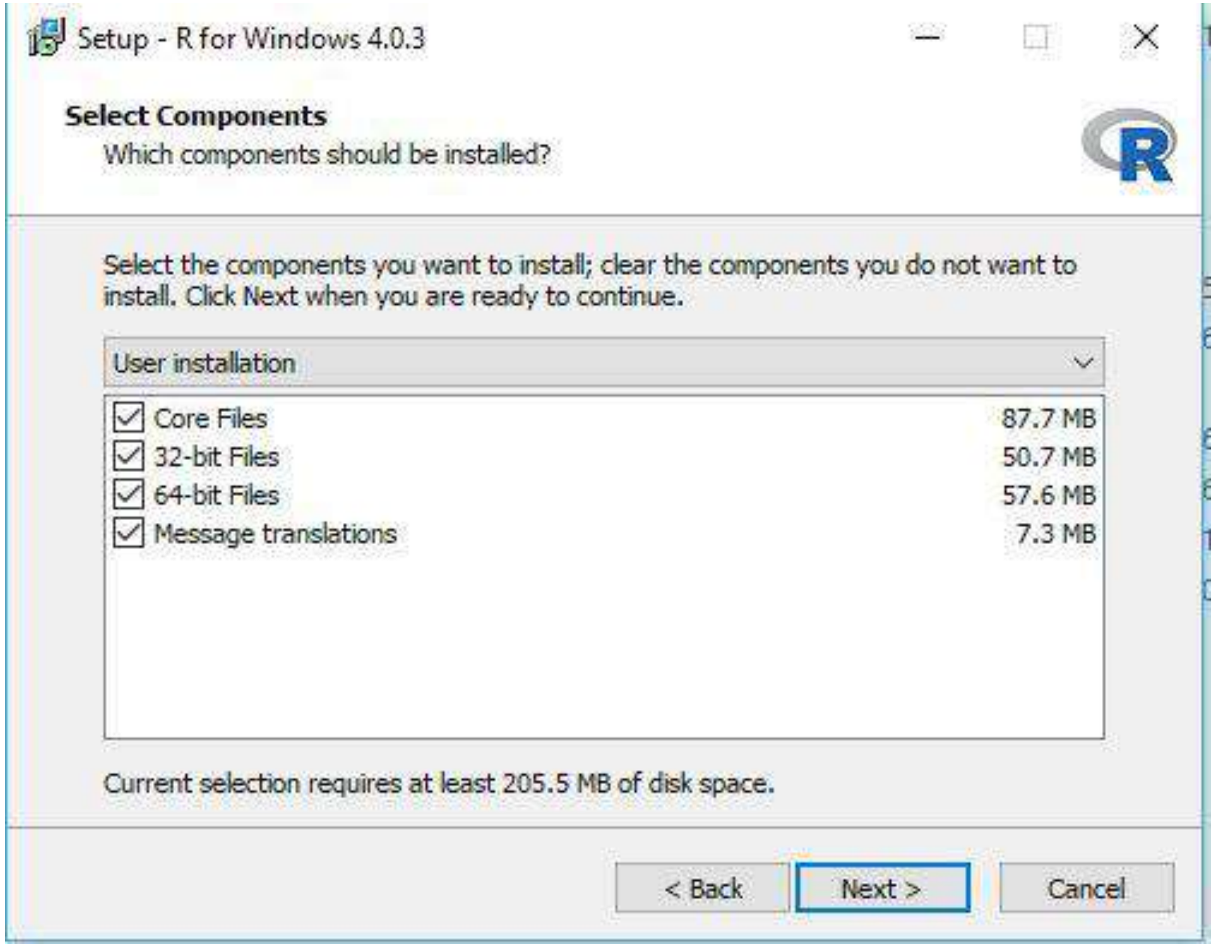
Installation of R Programming

- Browse the **Destination Location** (if necessary), and Click on **Next** button



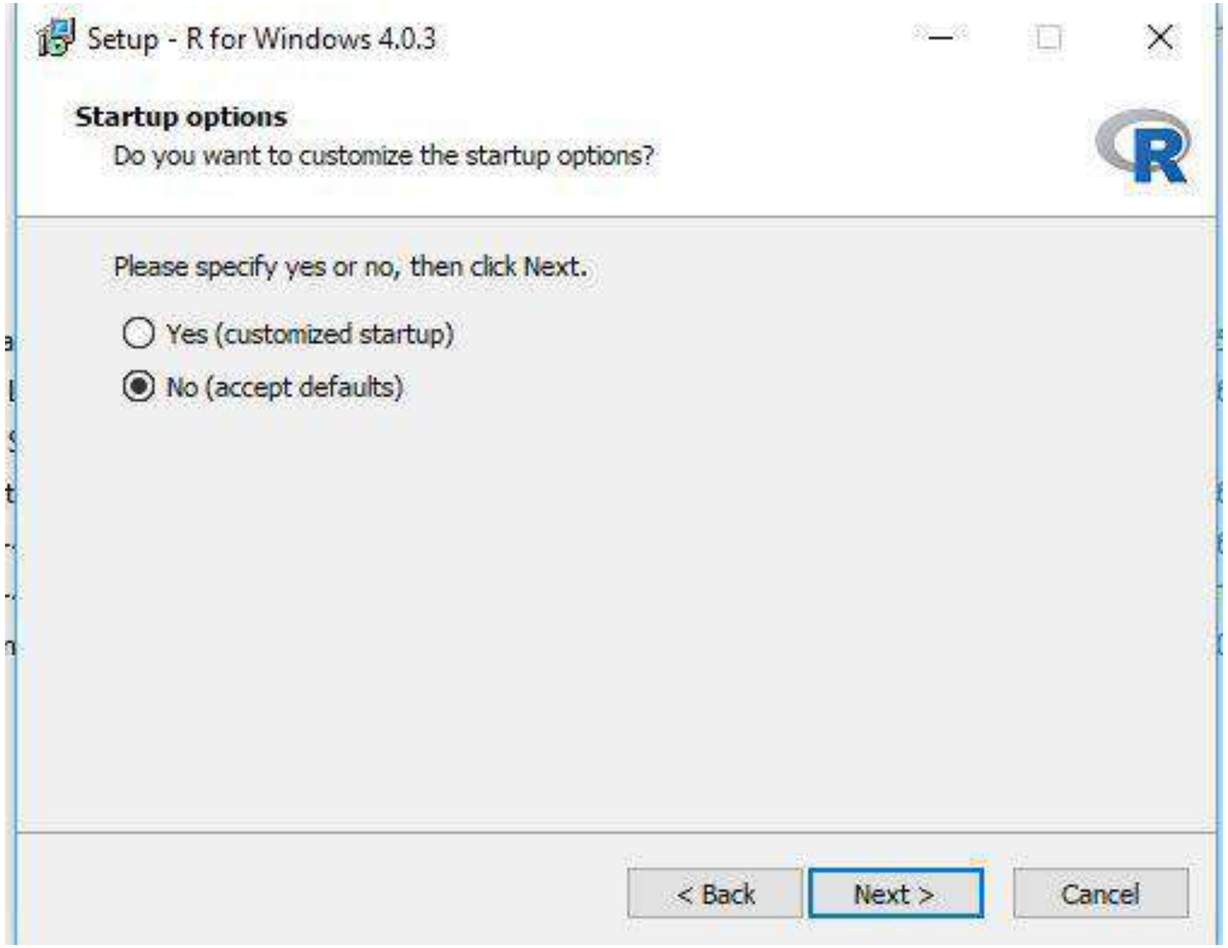
Installation of R Programming

- Select the **Components** (Already Ticked), Click on **Next** button



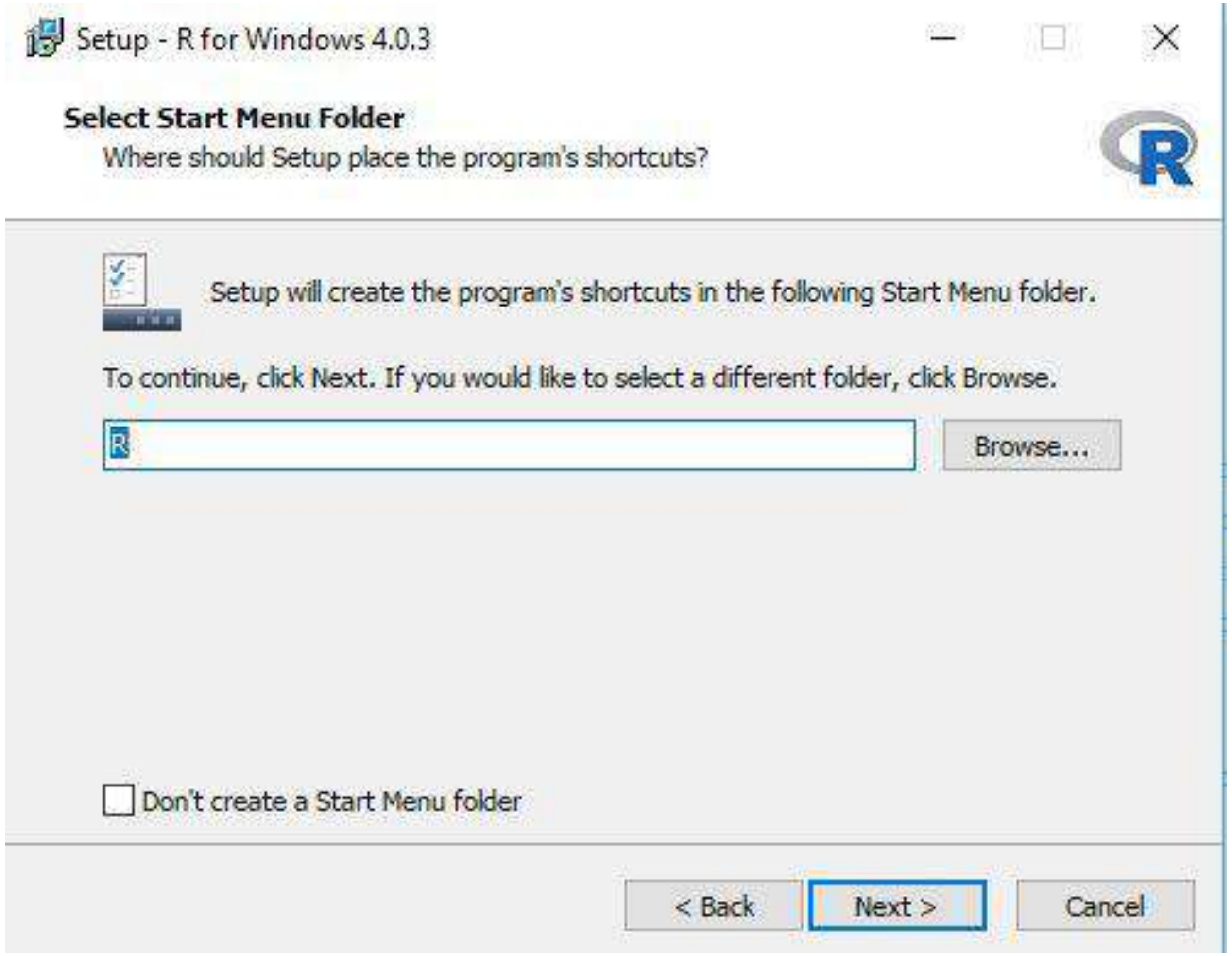
Installation of R Programming

- Select the **Startup options** (No (accept defaults) is selected), Click on **Next** button



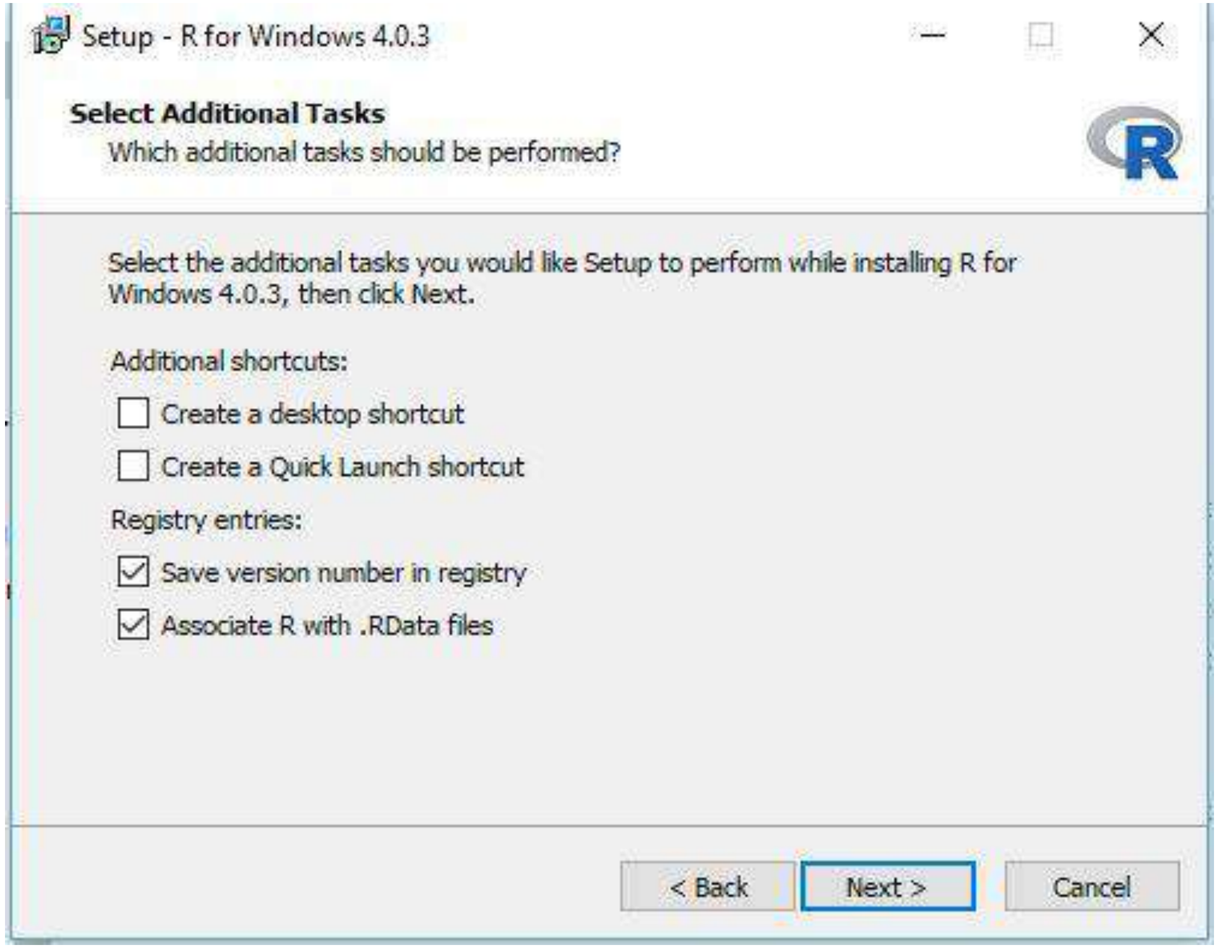
Installation of R Programming

- Select the **Start Menu Folder**, Click on **Next** button



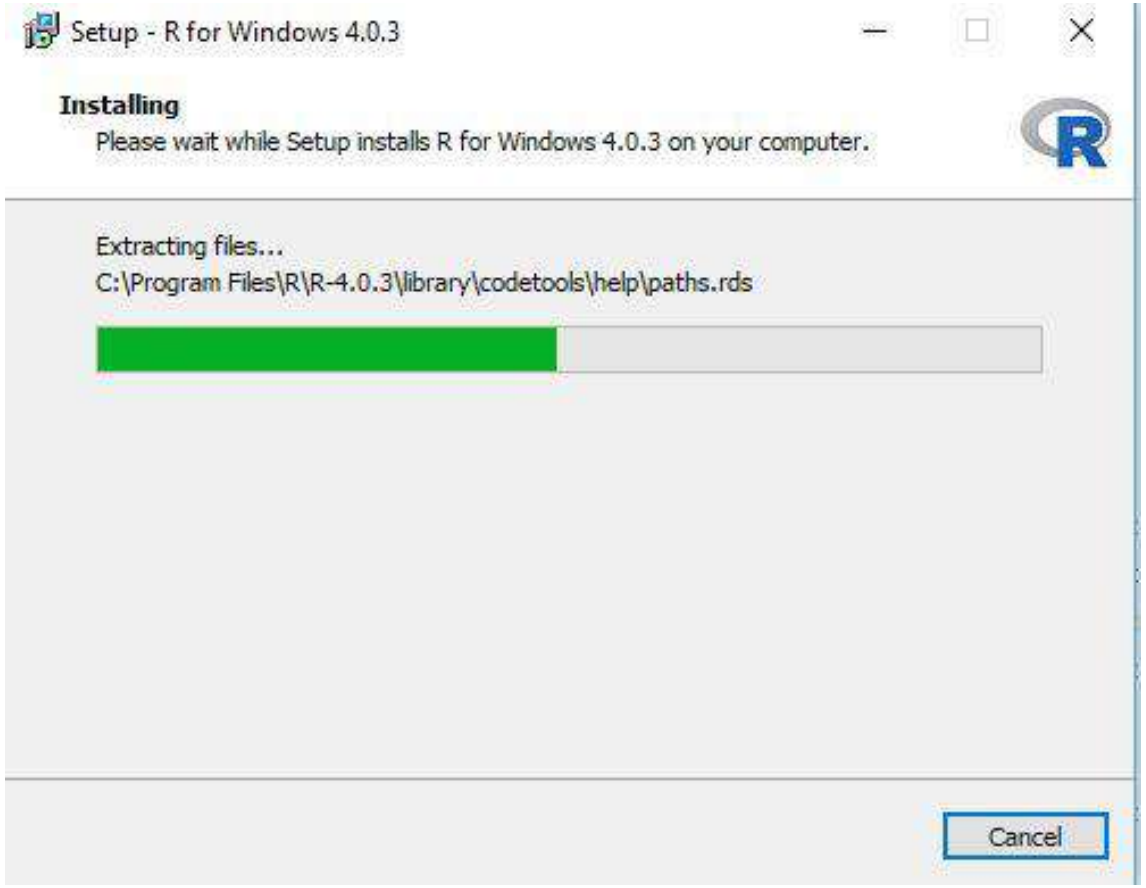
Installation of R Programming

- Select the **Additional shortcuts** (if needed), Click on **Next** button



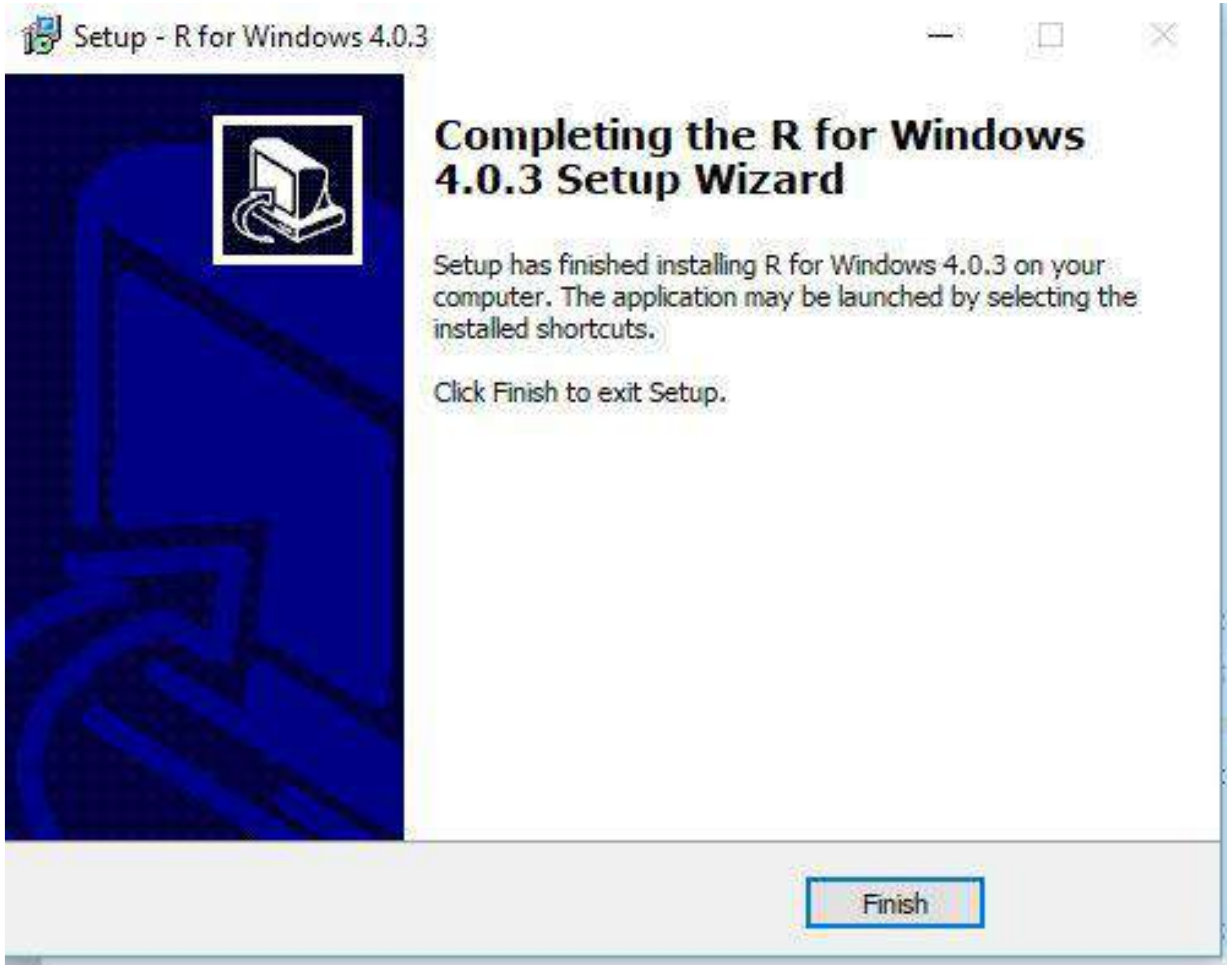
Installation of R Programming

- A new **R-4.0.3 win.exe Setup** pop-up window will appear with a **Setup Progress** message and a progress bar.



Installation of R Programming

- Pop-up window will appear with a **Setup was successfully** message.
- Click the **Finish** button.
- **R for Windows** should now be installed.



Basic Data Type

Numeric	10.23
Integer	10L
Complex	10+23i
Logical	TRUE FALSE
Character	“abc”, ‘TRUE’

Program to print the area of rectangle

Script File – T2.R

```
length=5
width=10
area=length*width
print('area of rectangle')
print(area)
```

```
> setwd("C:\\Users\\User\\Documents\\RClassPrg\\")
> source("t2.r")
[1] "area of rectangle"
[1] 50
```

Select the code lines
Click CTRL + R

Program to print the sum of 2 numbers

Script File – T3.R

```
n1=5
n2=10
s=n1+n2
cat(' sum=', s)
```

```
> setwd("C:\\Users\\User\\Documents\\RClassPrg\\")
> source("t3.r")
 sum= 15
```

Program to print the entered name and age

Script File – T4.R

```
name=readline('Enter Name : ')
age=readline('Enter Your Age : ')

cat('Name : ', name)
cat('\n')
cat('Age  : ', age)
```

```
> setwd("C:\\Users\\User\\Documents\\RClassPrg\\")
> source("t4.r")
 Enter Name : ajay
Enter Your Age : 45
Name :  ajay
Age  :  45
```

Program to print the entered name and age

Script File – T5.R

```
name=readline('Enter Name : ')
age=as.integer(readline('Enter Age : ' ))

cat('Name : ', name)
cat('\n')
age=age+10
cat('Age  : ', age)
```

```
> setwd("C:\\Users\\User\\Documents\\RClassPrg\\")
> source("t5.r")
 Enter Name : ajay
Enter Your Age : 45
Name :  ajay
Age  :  55
```

Contents to be covered

- Basic Data Types
- Introduction to Variables & Constants
- Types of Literals
- Input & Output Functions

Basic Data Types in R

- The list of all the basic data types provided by R:
  - Numeric
  - Integer
  - Complex
  - Logical
  - Character

Example of Data Types in R

- Numeric -- 10 , 10.24
- Integer -- 10L
- Complex -- 10i
- Logical -- TRUE / FALSE
- Character -- 'ABC' , "ABC"

Numeric Data Type

- **Decimal values** are referred to as numeric data types in R.
- This is the **default working** out data type.
- If you assign a decimal value for any variable **x** like given below, **x** will become a numeric type.

```
> g = 62.4          # assign a decimal value to g
> g                 # print the variable's value - g
```

Integer Data Type

- If you want to create an integer variable in R, you have to invoke the **as.integer()** function to define any integer type data.

- You can be certain that it is definitely an integer by applying the is.integer() function.

```
> s = as.integer(3)
>s                        # print the value of s
```

- Convert the decimal to integer

```
> as.integer(3.14)            # drives in a numeric value
```

  - But it will work like type casting where the value of 3.14 gets changed to 3.

Complex Data Type

A complex value for coding in R can be defined using the pure imaginary values **'i'**.

```
> k = 1 + 2i      # creating a complex number
> k               # printing the value of k
```

The below-mentioned example gives an error since −1 is not a complex value.

```
> sqrt(−1)        # square root of −1
```

And the error message will be something like this:

Warning message:

In sqrt(−1) : NaNs produced

Logical Data Type

A logical value is mostly created when a comparison between variables are done.

An example will be like:

```
>   a = 4; b = 6           # sample values
>   g = a > b              # is a larger than b?

>   g                      # print the logical value
```

Output:

[1] False

रा.इ.सू.प्रौ.सं NIELIT

R Programming

Character Data Type

- A character object can be used for representing string values in R.
- You have to convert objects into character values using the **as.character()** function within your code like this:
  - > g = as.character(3.14)
  - > g # prints the character string
    - Output:
    - [1] "3.14“
- To get the data class of a variable
  - > class(s) # print the class name of s
    - Output:
    - [1] "character"

What are variables in R?

- Variables are used for storing data where that can be altered based on need.
- Unique name has to be given to variable (also for functions and objects) is **identifier**.

Rules for writing Identifiers in R

- Identifier names are a combination of alphabets, digits, period (.) and also underscore (_).
- It is mandatory to start an identifier with a letter or a period.
- Another thing is if it starts with a period / dot operator, then you cannot write digit following it.

What are variables in R?

- Reserved words in R cannot be used as identifiers.

Valid identifiers in R are:

total, Sum, .work.with, this_is_accepted, Num6

Invalid identifiers in R:

t0t@l, 5um, _ray, TRUE, .0n3

Best Practices for Writing Identifiers

- **Former versions** of R used underscore to assign values.
- So, the period (.) operator was used broadly in variable names that have multiple words.

- **Present versions** of R support underscore (_) as valid identifier.
- But it is considered to be a good practice to use period for word separators.
- Here's an example, a.variable.name is preferred over a_variable_name.

What are Constants in R

- Constants are entities within a program whose value can't be changed.

There are 2 basic types of constant.

- Numeric constants
- Character constants.

Numeric Constants

- All numbers fall under this category. They can be of type **integer, double or complex**.
- It can be checked with the typeof() function.
- Numeric constants followed by L are regarded as integer and those followed by i are regarded as complex.

```
> typeof(5)       [1] "double"
> typeof(5L)      [1] "integer"
> typeof(5i)      [1] "complex“
```

- Numeric constants preceded by 0x or 0X are interpreted as hexadecimal numbers.

```
> 0xff            [1] 255
> 0XF + 1         [1] 16
```

Character Constant

- Character constants can be represented using either single quotes (') or double quotes (") as delimiters.

```
> 'example'
[1] "example"

> typeof("5")
[1] "character"
```

Built-in Constants

- Some of the built-in constants defined in R along with their values is shown below.

```
> LETTERS
  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
  [20] "T" "U" "V" "W" "X" "Y" "Z"
> letters
  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
  [20] "t" "u" "v" "w" "x" "y" "z"
> pi
  [1] 3.141593
```

Built-in Constants

```
> month.name
 [1] "January"   "February"  "March"     "April"     "May"       "June"
 [7] "July"      "August"    "September" "October"   "November"  "December"
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

But it is not good to rely on these, as they are implemented as variables whose values can be changed.

```
> pi
[1] 3.141593
> pi <- 56
> pi
[1] 56
```

Input & Output Function in R

Input and output functions are a programming language's ability to talk and interact with its users.

1. ***readline() function* -** The function returns a character vector containing the input values.

2. ***scan() function* -** This function can only read numeric values and returns a numeric vector.

How to read user input in R?

1. ***readline() function :*** Read the input given by the user in the terminal.

Syntax:

```
var_name=readline()
```

The function returns a character vector containing the input values.

If we need the input of other data types, then we need to make conversions.

For example:

```
var1=readline()
print(var1)
var1=as.integer(var1)
print(var1)
```

Functions that take vectors as input or give output in vectors are called vector functions.

Example of readline() function

```
name=readline(prompt="Enter name: ")
age=readline(prompt="Enter age: ")
age=as.integer(age)                                 #convert character into integer
print(paste("Hi,", name, "next year you will be", age+1, "years old."))
```

Output:

R Console

```
> source("C:\\Users\\Nielit-042\\Documents\\45.R")
Enter name: Fareed
Enter age: 33
[1] "Hi, Fareed next year you will be 34 years old."
>
```

In the above example

1. we convert the input age, which is a character vector into integer using the function as.integer().
2. paste() - Takes multiple elements from the multiple vectors and concatenates them into a single element.

How to read user input in R?

2. ***scan() function***

We can also use the **scan()** function to read user input. This function can only read numeric values and returns a numeric vector. If a non-numeric input is given, the function gives an error.

Syntax:

```
var_name <- scan()
```

Example:

```
var<-scan()
print(var)
```

NIELIT

R Programming

How to display output in R?

1. *print() functions*

1. We can use the **print()** function to display the output to the terminal.
2. It is a generic function.
3. This means that the function has a lot of different methods for different types of objects it may need to print.
4. The function takes an object as the argument.

Syntax:

```
print(var_name)
```

Example:

```
print("abc")
print(34)
```

How to display output in R?

2. *cat() function*

1. We can also use the **cat()** function to display a string.
2. The **cat()** function concatenates all of the arguments and forms a single string which it then prints.

Example:

```
cat("hello", "this","is","techvidvan",12345,TRUE)
```

Data Structures in R

- The list of data structure in provided by R:
  - Vector
  - Matrix
  - Arrays

Items of Same data type

  - List
  - Data.Frame
  - Factors

Items of multiple data type

Example of Data Structures in R

- Vector -- c(10, 20 ,30)
- Matrix -- matrix(c(1,2,3,4) , 2, 2)
- Arrays --
  - array(c(1,2,3,4,5,6,7,8) , dim=c(2,2,2))
- List --
  - m= list(1, c(1,2,3,4), c('ABC', 'XYZ'))
- Data.Frame --
  - empcode=c(101,102,103)
  - empname=c('Ajay', 'Vijay', 'Sanjay')
  - df=data.frame(empcode, empname)
- Factors
  - p=c('Ajay', 'Vijay', 'Sanjay', 'Ajay', 'Vijay')
  - t=factor(p)
    - **Ajay Vijay Sanjay Ajay Vijay**
    - **Levels: Ajay Sanjay Vijay**

Example of Data Structures in R

- **Vector**

  a= c(10, 20 ,30,40,50)

index	1	2	3	4	5
	10	20	30	40	50

a[1]   gives 10
a[4]   gives 40

a[1:3] gives 10 20 30

- **List**

  a=list(10, c(20,30), 40, c(50,60))

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

[[2]]
[1] 20 30

[[3]]
[1] 40

[[4]]
[1] 50 60
```

- a[[1]] gives   10
- a[[4]] gives   50 60

- b=unlist(a)   convert the list to vector
  [10,20,30,40,50,60]

Function in R

1. class ( ) - What kind of object (high level)

```
x=10                    y=10L
class(x)   --- numeric  class(y)   --- integer
```

2. typeof( ) - What is the object data type (low level)

```
x=10                    y=10L
class(x)   --- double   class(y)   --- integer

y=c(10,'ajay ',78,89)
class(y)    ---   character
typeof(y)   --- character
```

3. length() – length of string , vector etc

```
x=10
length(x)     --- 1

y=c(10,'ajay',78,89)
length(y)     --- 4

y='computer‘
length(y)     --- 1
```

4. nchar(x) - find the string length or number length

```
x='computer'
nchar(x)   --- 8

x=12345
nchar(x)    --- 5
```

5. str( ) – structure of a variable

```
x=10            str(x)   ---- num 10
x=10L           str(x)   ---- int 10
x='computer'    str(x)   ---- chr  "computer“

y=c(10,'ajay',78,89)

str(y)   --- chr [1:4] "10" "ajay" "78" "89"
```

Contents to be covered

- R Operators
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Miscellaneous Operators

R Operators

- Operators are the symbols directing the compiler to perform various kinds of operations between the operands.
- Operators simulate the various mathematical, logical and decision operations performed on a set of Complex Numbers, Integers, and Numericals as input operands.
- Operators can be categorized based upon their different functionality.
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Assignment Operators
  - Miscellaneous Operators

Arithmetic Operators

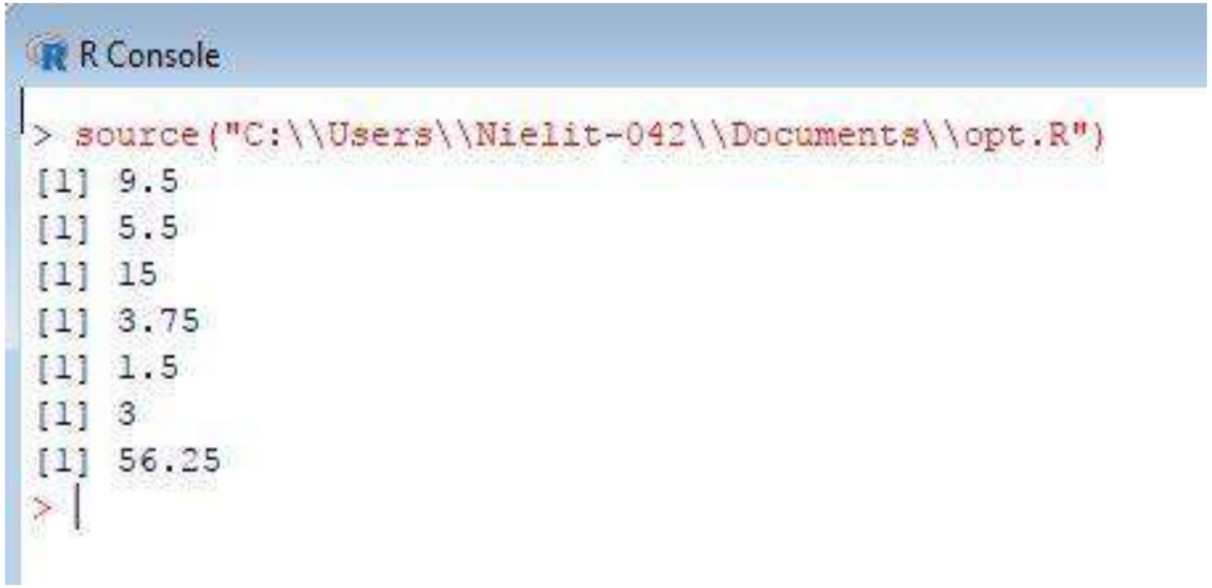
- Arithmetic Operators are used to accomplish arithmetic operations. They can be operated on the basic data types Numericals, Integers, Complex Numbers. Vectors with these basic data types can also participate in arithmetic operations, during which the operation is performed on one to one element basis.

Operator	Description	Usage
+	Addition of two operands	**a + b**
–	Subtraction of second operand from first	**a – b**
*	Multiplication of two operands	**a * b**
/	Division of first operand with second	**a / b**
%%	Remainder from division of first operand with second	**a %% b**
%/%	Quotient from division of first operand with second	**a %/% b**
^	First operand raised to the power of second operand	**a^b**

R Arithmetic Operators Example for integers

```
a=7.5
b=2
print(a+b)          #addition
print(a-b)          #subtraction
print(a*b)          #multiplication
print(a/b)          #Division
print(a%%b)         #Reminder
print(a%/%b)        #Quotient
print(a^b)          #Power of
```

Output:



R Arithmetic Operators Example for vectors

```
a=c(8,9,6)
b=c(2,4,5)

print(a+b)      #addition
print(a-b)      #subtraction
print(a*b)      #multiplication
print(a/b)      #Division
print(a%%b)     #Reminder
print(a%/%b)    #Quotient
print(a^b)      #Power of
```

Output:

R Console

```
> source("C:\\Users\\Nielit-042\\Documents\\opt1.R")
[1] 10 13 11
[1] 6 5 1
[1] 16 36 30
[1] 4.00 2.25 1.20
[1] 0 1 1
[1] 4 2 1
[1]   64 6561 7776
> |
```

R Relational Operators

- Relational Operators are those that find out relation between the two operands provided to them. Following are the six relational operations R programming language supports. The output is Boolean (TRUE or FALSE) for all of the Relational Operators in R programming language.

Operator	Description	Usage
<	Is first operand less than second operand	**a < b**
>	Is first operand greater than second operand	**a > b**
==	Is first operand equal to second operand	**a == b**
<=	Is first operand less than or equal to second operand	**a <= b**
>=	Is first operand greater than or equal to second operand	**a > = b**
!=	Is first operand not equal to second operand	**a!=b**

राइसूप्रौसं NIELIT R Programming

R relational Operators Example for integers

```
a<-7.5
b<-2

print(a<b)     # less than
print(a>b)     # greater than
print(a==b)    # equal to
print(a<=b)    # less than or equal to
print(a>=b)    # greater than or equal to
print(a!=b)    # not equal to
```

Output:

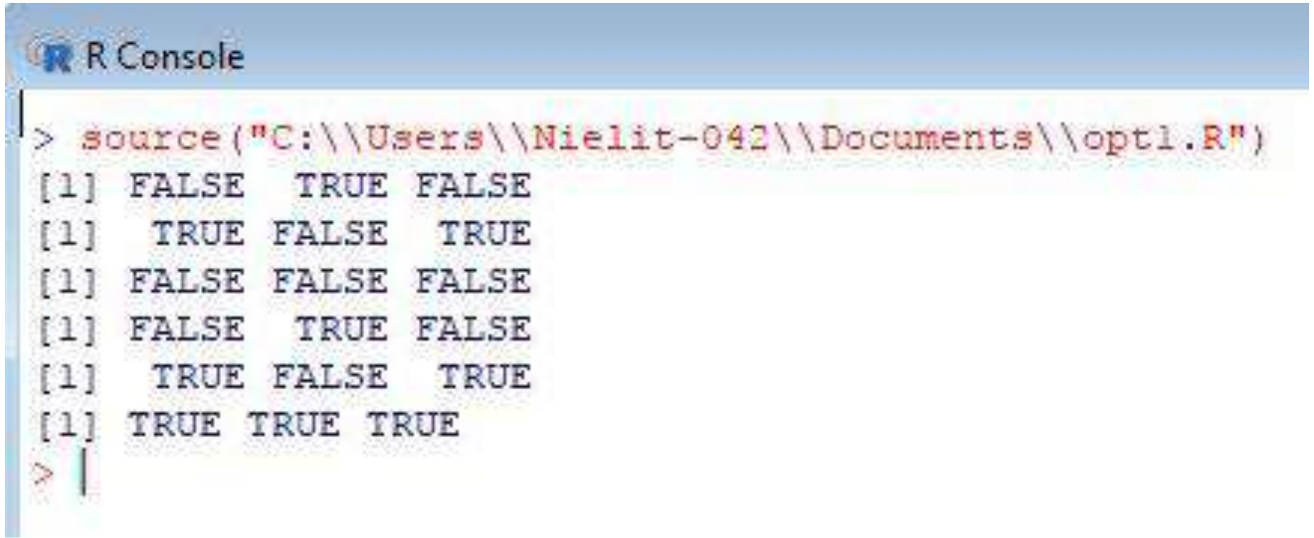
R Console

```
> source("C:\\Users\\Nielit-042\\Documents\\opt1.R")
[1] FALSE
[1] TRUE
[1] FALSE
[1] FALSE
[1] TRUE
[1] TRUE
>
```

R relational Operators Example for vectors

```
a<-c(7.5, 3, 5)
b<-c(2, 7, 0)

print(a<b)      # less than
print(a>b)      # greater than
print(a==b)     # equal to
print(a<=b)     # less than or equal to
print(a>=b)     # greater than or equal to
print(a!=b)     # not equal to
```

Output:



R Logical Operators

- Logical Operators in R programming language work only for the basic data types logical, numeric and complex and vectors of these basic data types.

<table>
<tr><th>Operator</th><th>Description</th><th>Usage</th></tr>
<tr><td>&</td><td>Element wise logical AND operation.</td><td>a & b</td></tr>
<tr><td>|</td><td>Element wise logical OR operation.</td><td>a | b</td></tr>
<tr><td>!</td><td>Element wise logical NOT operation.</td><td>!a</td></tr>
<tr><td>&&</td><td>Operand wise logical AND operation.</td><td>a && b</td></tr>
<tr><td>||</td><td>Operand wise logical OR operation.</td><td>a || b</td></tr>
</table>

R Logical Operators Example for basic logical elements

```
a=0           # logical FALSE
b=2           # logical TRUE

print(a&b)    # logical AND element wise
print(a|b)    # logical OR element wise
print(!a)     # logical NOT element wise
print(a&&b)   # logical AND consolidated for all elements
print(a||b)   # logical OR consolidated for all elements
```

Output:

R Logical Operators Example for boolean vector

```
a<-c(TRUE, TRUE, FALSE, FALSE)
b<-c(TRUE, FALSE, TRUE, FALSE)

print(a&b)      # logical AND element wise
print(a|b)      # logical OR element wise
print(!a)       # logical NOT element wise
print(a&&b)     # logical AND consolidated for all elements
print(a||b)     # logical OR consolidated for all elements
```

Output:

R Console

```
> source("C:\\Users\\Nielit-042\\Documents\\opt1.R")
[1]  TRUE FALSE FALSE FALSE
[1]  TRUE  TRUE  TRUE FALSE
[1] FALSE FALSE  TRUE  TRUE
[1] TRUE
[1] TRUE
>
```

R Assignment Operators

- Assignment Operators are those that help in assigning a value to the variable.

Operator	Description	Usage
=	Assigns right side value to left side operand	**a = 3**
<-	Assigns right side value to left side operand	**a <- 5**
->	Assigns left side value to right side operand	**4 -> a**
<<-	Assigns right side value to left side operand	**a <<- 3.4**
->>	Assigns left side value to right side operand	**c(1,2) ->> a**

R Assignment Operators Example

```
a=2
print(a)

a<-TRUE
print(a)

454->a
print(a)

a<<-2.9
print(a)

c(6,8,9)->a
print(a)
```

Output:

```
R Console
> source("C:\\Users\\Nielit-042\\Documents\\opt1.R")
[1] 2
[1] TRUE
[1] 454
[1] 2.9
[1] 6 8 9
>
```

R Miscellaneous Operators

- These operators does not fall into any of the categories mentioned above, but are significantly important during R programming for manipulating data.

Operator	Description	Usage
:	Creates series of numbers from left operand to right operand	**a:b**
%in%	Identifies if an element(a) belongs to a vector(b)	**a %in% b**
%*%	Performs multiplication of a vector with its transpose	**A %*% t(A)**

R Miscellaneous Operators Example

```
a=23:31
print(a)

a=c(25,27,76)
b=27
print(b%in%a)

M=matrix(c(1,2,3,4),2,2,TRUE)
print(M%*%t(M))
```

Output:

R Console

```
> source("C:\\Users\\Nielit-042\\Documents\\opt1.R")
[1] 23 24 25 26 27 28 29 30 31
[1] TRUE
     [,1] [,2]
[1,]    5   11
[2,]   11   25
> |
```

Precedence and Associativity of different operators in R from highest to lowest

Operator	Description	Associativity
^	Exponent	**Right to Left**
-x, +x	Unary minus, Unary plus	**Left to Right**
%%	Modulus	**Left to Right**
***, /**	Multiplication, Division	**Left to Right**
+, –	Addition, Subtraction	**Left to Right**
<, >, <=, >=, ==, !=	Comparisons	**Left to Right**
!	Logical NOT	**Left to Right**
&, &&	Logical AND	**Left to Right**
|, ||	Logical OR	**Left to Right**
->, ->>	Rightward assignment	**Left to Right**
<-, <<-	Leftward assignment	**Right to Left**
=	Leftward assignment	**Right to Left**

Example- Odd or Even number

```
n=122
r=n%%2

if ( r==0 )
{
print('Even  ')
} else
{
print('Odd  ')
}
```

Output

"sum of digits : "
6

Example- Number divisible by 3 and 5

```
n=122
r1=n%%3
r2=n%%5

if ( r1==0 && r2==0 )
{
print('Divisible by 3 and 5 ')
} else
{
print('Not Divisible by 3 and 5 ')
}
```

Output

"Not Divisible by 3 and 5"

Example- Sum of digits of number

```
n=123
d1=n%%10
n=n%/%10

d2=n%%10
n=n%/%10

d3=n%%10
n=n%/%10

print('sum of digits : ')
print( d1+d2+d3)
```

Output

```
"sum of digits : "
6
```

Example- Check for divisibilty

```
n=c(30, 60, 90 )
r1=n%%3
r2=n%%5

if ( all(r1==0 & r2==0) )
{
print('Divisible  ')
} else
{
print('Not divisible ')
}
```

Output

```
"Divisible "
```

Example- Check for divisibilty

```
n=c(30, 50, 90 )
r1=n%%3
r2=n%%5

if ( all(r1==0 & r2==0) )
{
print('Divisible  ')
} else
{
print('Not divisible ')
}
```

Output

```
"Not Divisible "
```

Contents to be covered

- R Flow Control Statement
- Branching - If , If else, ifelse()

R Flow Control Statement

- **Flow Control statements** decides the direction of flow of program execution.
- **Decision making:** Decision making is a prime feature of any programming language. It allows us to make a decision, based on the result of a condition. Decision making is involved in order to change the sequence of the execution of statements, depending upon certain conditions.
- A set of statements is provided to the program, along with a condition. Statements get executed only if the condition stands true, and, optionally, an alternate set of statements is executed if the condition becomes false.
  - If Statement
  - Else Statement
  - Else If Ladder

if statement

The syntax of if statement is:

if (test_expression) {

statement

}

If the test_expression is TRUE, the statement gets executed But if it's FALSE, nothing happens.

Here, test_expression can be a logical or numeric vector, but only the first element is taken into consideration.

In the case of numeric vector, zero is taken as FALSE, res as TRUE.

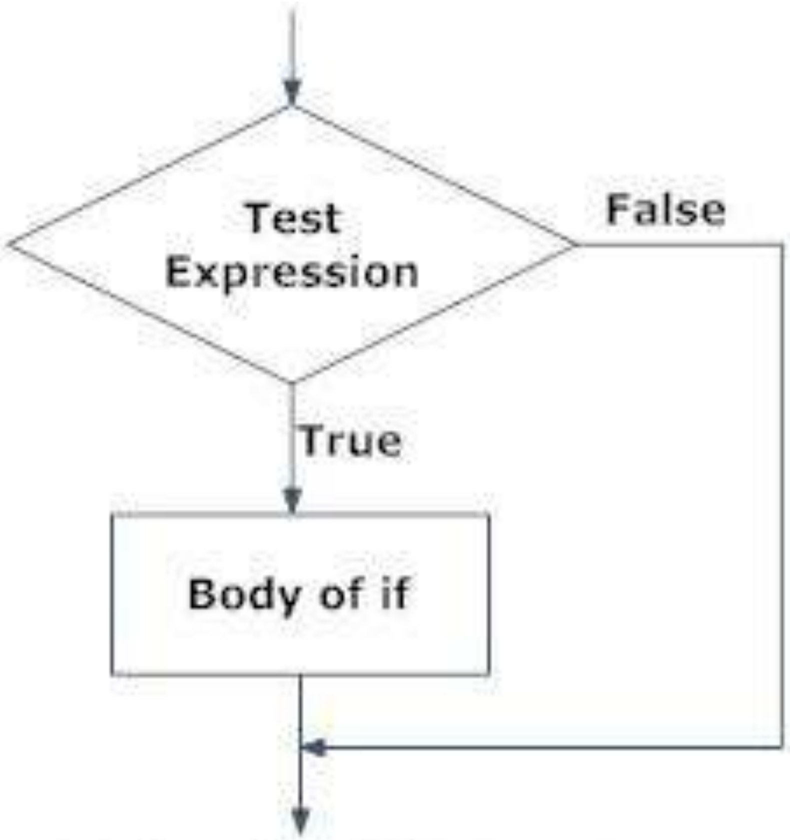


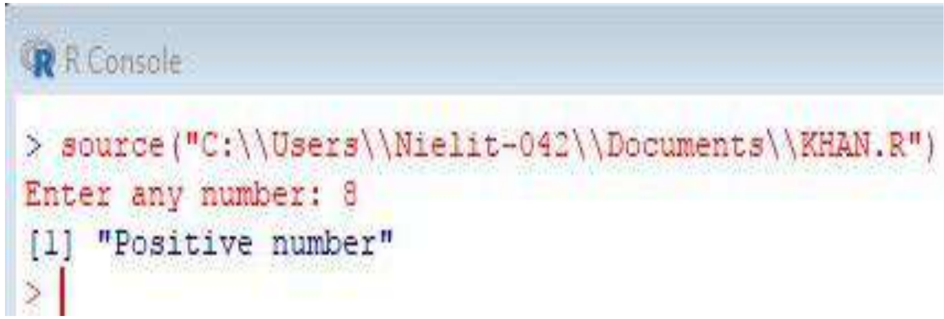
Fig: Operation of if statement

Example of if statement

Example -1

```
num<-readline(prompt="Enter any number: ")
num<-as.integer(num)
if(num>0)
{
print("Positive number")
}
```

Output:



Example-2

```
num=readline(prompt="Enter Item Quantity : ")
num=as.integer(num)
disc=0
if(num> 100)
{
   disc=num*10/100
}
cat('Discount= ', disc)
```

Output:

```
Enter Item Quantity : 110
Discount=  11
```

If....else statement

The syntax of if...else statement is:

```
if (test_expression) {
statement1
} else {
statement2
}
```

The else part is optional and is only evaluated if test_expression is FALSE.

It is important to note that else must be in the same line as the closing braces of the if statement.

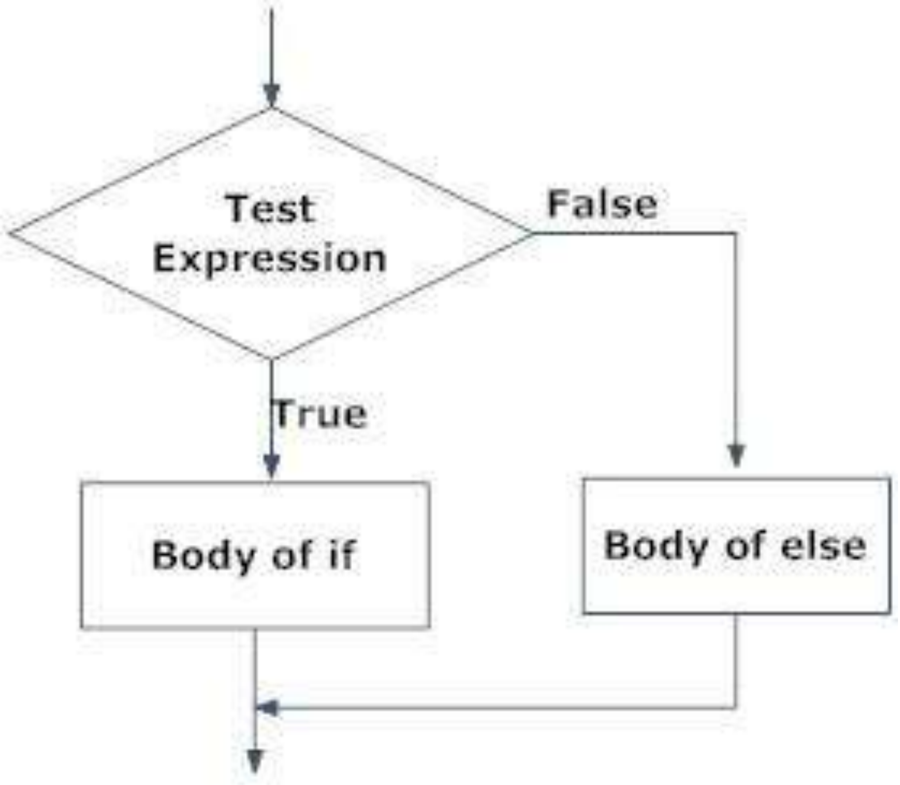


Fig: Operation of if...else statement

Example of if...else statement

Example- 1

```
num<-readline(prompt="Enter any number: ")
num<-as.integer(num)
if(num>0)
{
print("Non-negative number")
}  else
{
print("Negative number")
}
```

Output:

Example-2

```
num=readline(prompt="Enter Item Quantity : ")
num=as.integer(num)
if(num> 100)
{
   disc=num*10/100
} else
{
   disc=0
}
cat('Discount= ', disc)
```

Output:

```
Enter Item Quantity : 110
Discount=  11
```

If....else Ladder

The if…else ladder (if…else…if) statement allows you execute a block of code among more than 2 alternatives

The syntax of if…else statement is:

```
if ( test_expression1) {
statement1
} else if ( test_expression2) {
statement2
} else if ( test_expression3) {
statement3
} else {
statement4
}
```

Only one statement will get executed depending upon the test_expressions.

Example of if...else ladder

Example -1

```
num<-readline(prompt="Enter any number: ")
num<-as.integer(num)
if(num<0)
{
print("Negative number")
} else if (num>0){

print("Positive number")
}else
print("Zero")
```

Output:

```
R Console
> source("C:\\Users\\Nielit-042\\Documents\\KHAN.R")
Enter any number: -6
[1] "Negative number"
```

Example -2

```
marks=readline(prompt="Enter  Marks : ")
marks=as.integer(marks)
if(marks> 75)
{
   grade='A'
} else  if (marks > 60 )
{
  grade='B'
} else if (marks > 49)
{
  grade='C'
} else
{
 grade='F'
}
cat('Grade= ', grade)
```

Output:

Enter Marks : 61
Grade= B

Ifelse() function

The 'ifelse()' function is the alternative and shorthand form of the R if-else statement. Also, it uses the 'vectorized' technique, which makes the operation faster. All of the vector values are taken as an argument at once rather than taking individual values as an argument multiple times.

Syntax

ifelse(logical_expression, a , b)

The argument above in 'ifelse' states that:

logical_expression: Indicates an input vector, which in turn will return the vector of the same size as output.

a: Executes when the logical_expression is TRUE.

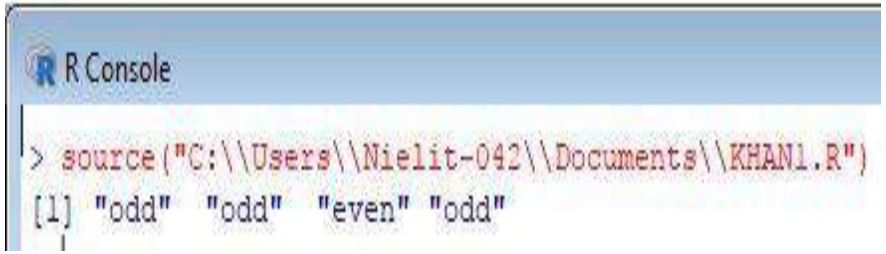
b: Executes when the logical_expression is FALSE.

Example of ifelse() function

Example 1

```
a = c(5,7,2,9)
b=ifelse(a %% 2 == 0,"even","odd")
print(b)
```

Output:



Example-2

```
marks= c(56,89,71,23)
grade=ifelse(marks>=50, 'Pass', 'Fail')
print(grade)

grade=ifelse(marks>75, 'A', ifelse( marks>60, 'B',
ifelse( marks > 49, 'C', 'F')))
print(grade)
```

Output

"Pass" "Pass" "Pass" "Fail"

"C" "A" "B" "F"

Vector using :

a=1:10

1 2 3 4 5 6 7 8 9 10

a=5:50

5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

a=-1:-10

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

a=-10:-1

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

Accessing and Modifying Vector

a=1:10

a[2] -- 2

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

a[-2] will delete the item at 2 index

a[3:5] -- 3 4 5

a[c(2,5,7)] -- 10 5 7

Seq ( ) Function

seq() function generates a sequence of numbers.

Syntax

seq(from = 1, to = 1, by = (to – from))

from, to : begin and end number of the sequence

by : step, increment (Default is 1)

Generate a sequence from -6 to 7:

```
> x = seq(-6,7)
> x
[1] -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7
> is.vector(x)
[1] TRUE
```

seq(-6,7,by=2)   -6 -4 -2 0 2 4 6

seq(-2,2,by=0.3)   -2.0 -1.7 -1.4 -1.1 -0.8 -0.5 -0.2 0.1 0.4 0.7 1.0 1.3 1.6 1.9

seq(-2,2,length.out=10)

-2.0000000 -1.5555556 -1.1111111 -0.6666667 -0.2222222 0.2222222 0.6666667 1.1111111 1.5555556 2.0000000

seq(10)   1 2 3 4 5 6 7 8 9 10

What is the output ?

1. seq(0, 1, length.out = 11)
2. seq(1, 9, by = 2)
3. seq(1, 9, by = pi)
4. seq(1, 6, by = 3)
5. seq(1.575, 5.125, by = 0.05)
6. seq(17)

Contents to be covered

- Looping Control Structure – while
- Using break & Continue with While Statement
- Looping Control Structure – repeat loop

Looping Control Structure - while

- **Loops are used in programming to repeat a specific block of code.**
- In R programming, while loops are used to loop until a specific condition is met.

Syntax of while loop

```
while (test_expression)
{
statement
}
```

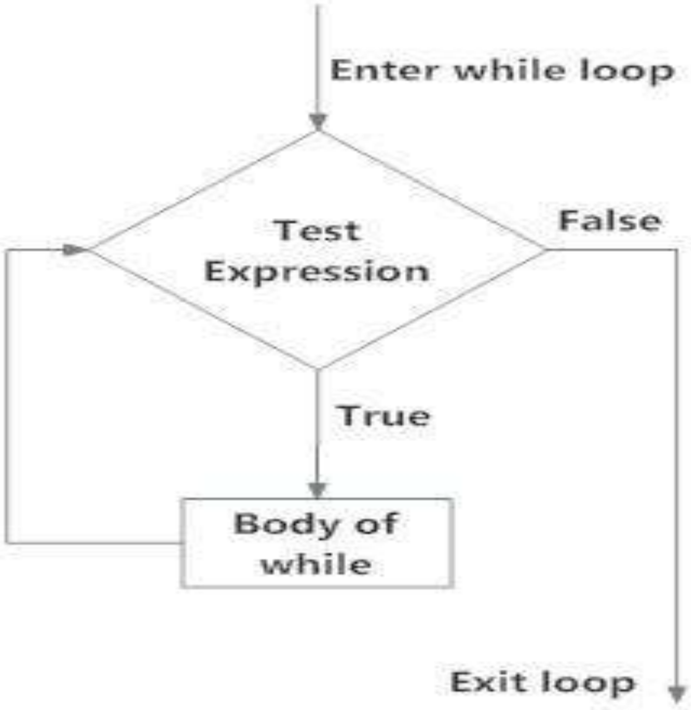


Fig: operation of while loop

- Here, test_expression is evaluated and the body of the loop is entered if the result is TRUE.
- The statements inside the loop are executed and the flow returns to evaluate the test_expression again.
- This is repeated each time until test_expression evaluates to FALSE, in which case, the loop exits.

Example of while loop

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

Output:

```
R Console
> source("C:\\Users\\Nielit-042\\Documents\\Loop.R")
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
>
```

- In the above example, i is initially initialized to 1.
- Here, the test_expression is i < 6 which evaluates to TRUE since 1 is less than 6. So, the body of the loop is entered and i is printed and incremented.
- Incrementing i is important as this will eventually meet the exit condition. Failing to do so will result into an infinite loop.
- In the next iteration, the value of i is 2 and the loop continues.
- This will continue until i takes the value 6. The condition 6 < 6 will give FALSE and the while loop finally exits.

Print Even Number

```
#Print Odd Number
num = as.integer(readline(prompt="Enter a number: "))
count=1
while(count<= num) {
  print( count )
  count=count+1
}
```

```
> source('t3.r')
Enter a number: 5
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

```
#Sum of Digits of number
num = as.integer(readline(prompt="Enter a number: "))
sum=0
n=num
while(n>0) {
  d=n%%10
  n=n%/%10
  sum=sum+d
}

cat('Sum of digits of',num , '=',sum)
```

```
> source('t3.r')
Enter a number: 123
Sum of digits of  123 = 6
```

#Print vector items

```
n=c(1,5,8,9)

i=1
while  ( i<= length(n) )
{
  print(n[i])
  i=i+1
}
```

```
> source('t3.r')
[1] 1
[1] 5
[1] 8
[1] 9
```

#Print List items

```
n=list(1,5,8,9)

i=1
while  ( i<= length(n) )
{
  print(n[i])
  i=i+1
}
```

```
> source('t3.r')
[[1]]
[1] 1

[[1]]
[1] 5

[[1]]
[1] 8

[[1]]
[1] 9
```

#Print arrays Items

```
n=array( c(1,5,8,9))

i=1
while  ( i<= length(n) )
{
  print(n[i])
  i=i+1
}
```

```
> source('t3.r')
[1] 1
[1] 5
[1] 8
[1] 9
```

```
#Print count of even items in vector
n= c( 1,5,8,9)

i=1
c=0
while  ( i<= length(n) )
{
  if (n[i]%%2==0) c=c+1
  i=i+1
}

print(c)
```

```
> source('t3.r')
[1] 1
```

```
#Print List items
n= list( 1,5,8,9)

i=1
c=0
while  ( i<= length(n) )
{
   if (as.integer( n[i]) %%2==0) c=c+1
   i=i+1
}

print(c)
```

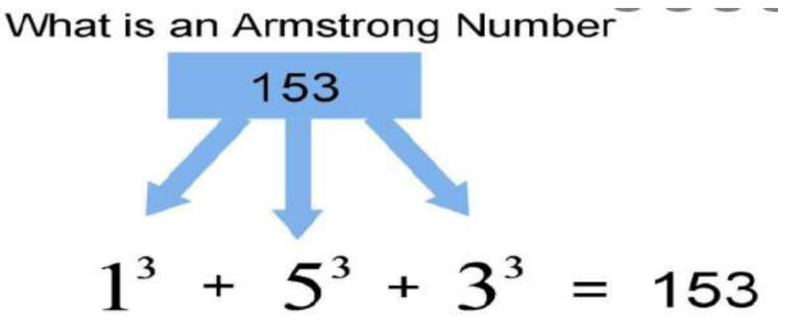
```
> source('t3.r')
[1] 1
```

```
#Print arrays Items
n=array( c(1,5,8,9))

i=1
while  ( i<= length(n) )
{
  if (n[i] %% 2==0 ) c=c+1
 i=i+1
}
```

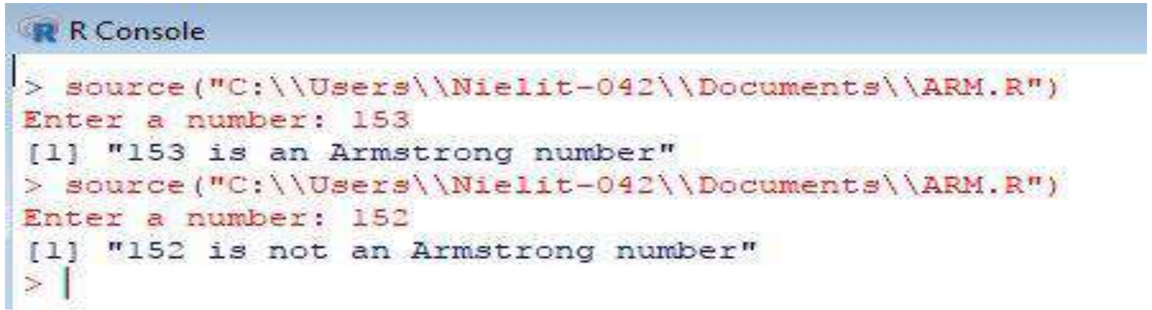
```
> source('t3.r')
[1] 1
```

Program to check Armstrong Number or Not

```
num = as.integer(readline(prompt="Enter a number: "))
sum = 0
temp = num
while(temp > 0) {
digit = temp %% 10
sum = sum + (digit ^ 3)
temp = floor(temp / 10)
}
if(num == sum) {
print(paste(num, "is an Armstrong number"))
} else {
print(paste(num, "is not an Armstrong number"))
}
```



$$153 = 1^3 + 5^3 + 3^3$$
$$370 = 3^3 + 7^3 + 0^3$$
$$371 = 3^3 + 7^3 + 1^3$$
$$407 = 4^3 + 0^3 + 7^3.$$



R Console

```
> source("C:\\Users\\Nielit-042\\Documents\\ARM.R")
Enter a number: 153
[1] "153 is an Armstrong number"
> source("C:\\Users\\Nielit-042\\Documents\\ARM.R")
Enter a number: 152
[1] "152 is not an Armstrong number"
>
```

Program to print Fibonacci Series

```
nterms = as.integer(readline(prompt="How many terms? "))
n1 = 0
n2 = 1
count = 2
if(nterms <= 0) {
     print("Plese enter a positive integer")
} else {
if(nterms == 1) {
    print("Fibonacci sequence:")
    print(n1)
} else {
    print("Fibonacci sequence:")
    print(n1)
    print(n2)
    while(count < nterms) {
         n3 = n1 + n2
         print(n3)
         n1 = n2
         n2 = n3
         count = count + 1
}}}
```

Output:

```
R Console
> source("C:\\Users\\Nielit-042\\Documents\\fibo.R")
How many terms? 7
[1] "Fibonacci sequence:"
[1] 0
[1] 1
[1] 1
[1] 2
[1] 3
[1] 5
[1] 8
>
```

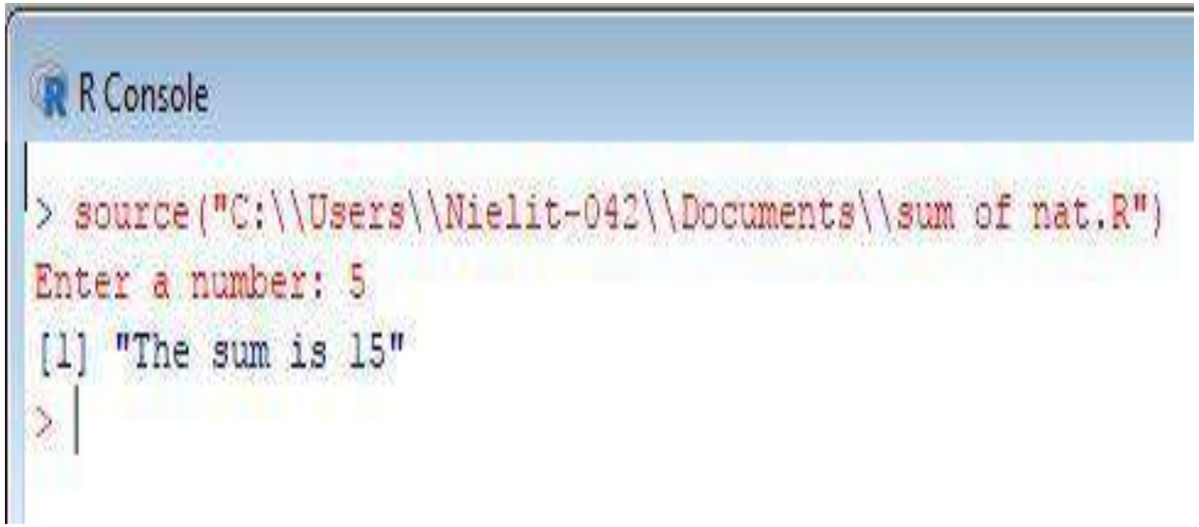
Program to print sum of natural number

```
num = as.integer(readline(prompt = "Enter a number: "))
if(num < 0) {
    print("Enter a positive number")
} else {
     sum = 0
     count=1
     while(count<= num) {
            sum = sum + count
            count = count + 1
         }
    print(paste("The sum is", sum))
   }
```

1 + 2+ 3+ 4+ 5 = 15

1+2+3+4+5+6+7+8+9+10 = 55

Output:



R break & next statement

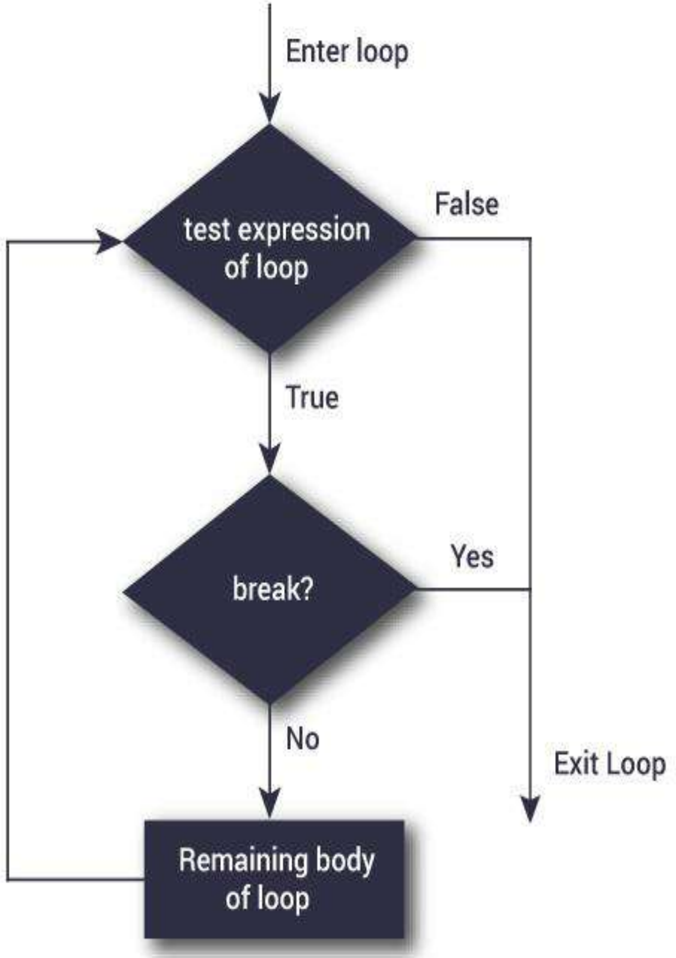
- In R programming, a normal looping sequence can be altered using the break or the next statement.

break statement

- A break statement is used inside a loop to stop the iterations and flow the control outside of the loop.
- In a nested looping situation, where there is a loop inside another loop, this statement exits from the innermost loop that is being evaluated.
- The syntax of break statement is:

  if (test_expression) { break }

Note: the break statement can also be used inside the else branch of if...else statement.



Example of Break Statement

```
num = 10
sum = 0
count = 0
     while(count <= num) {
        count = count + 1
        if (count == 5) break
        sum = sum + count
        }
print(paste("The sum is", sum))
```

This will give the output 10, instead of 55.

if (count == 5) break :
This statement will break the while loop prematurely, when the value of **count** is **5**.

R break & next statement

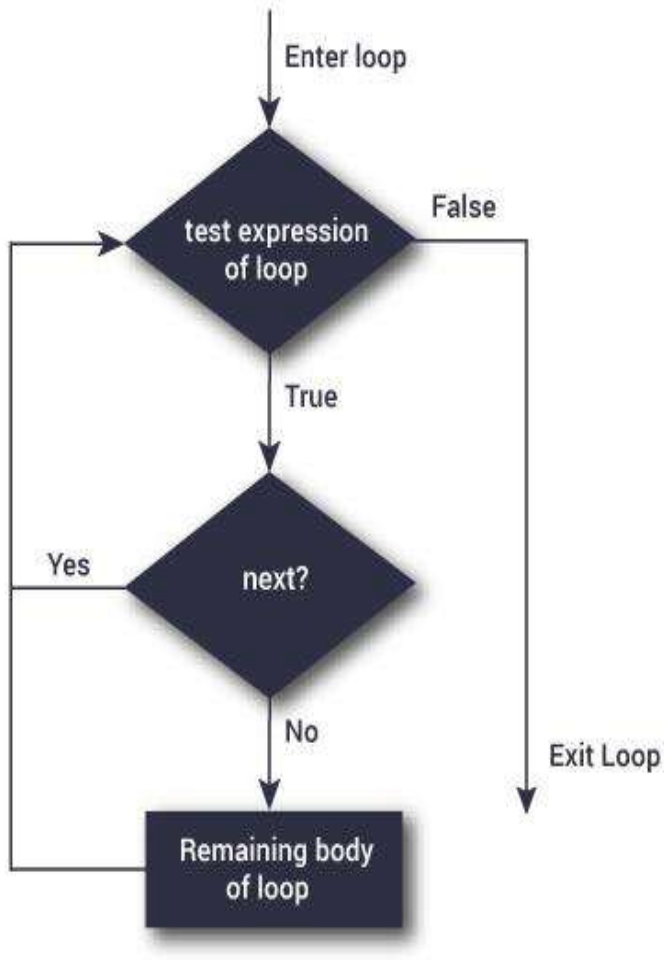
next statement

- A next statement is useful when we want to skip the current iteration of a loop without terminating it. On encountering next, the R parser skips further evaluation and starts next iteration of the loop.

The syntax of next statement is:

if (test_condition) {next}

Note: the next statement can also be used inside the else branch of if...else statement.



Example of Next Statement

```
num = 5
sum = 0
count = 0
    while(count <= num) {
        count = count + 1
        if (count == 5) next
        sum = sum + count
}
print(paste("The sum is", sum))
```

This will give the output 16, instead of 15.

if (count == 5) next :
This skip the remaining statement of the current iteration only, when the value of **count** is **5**.

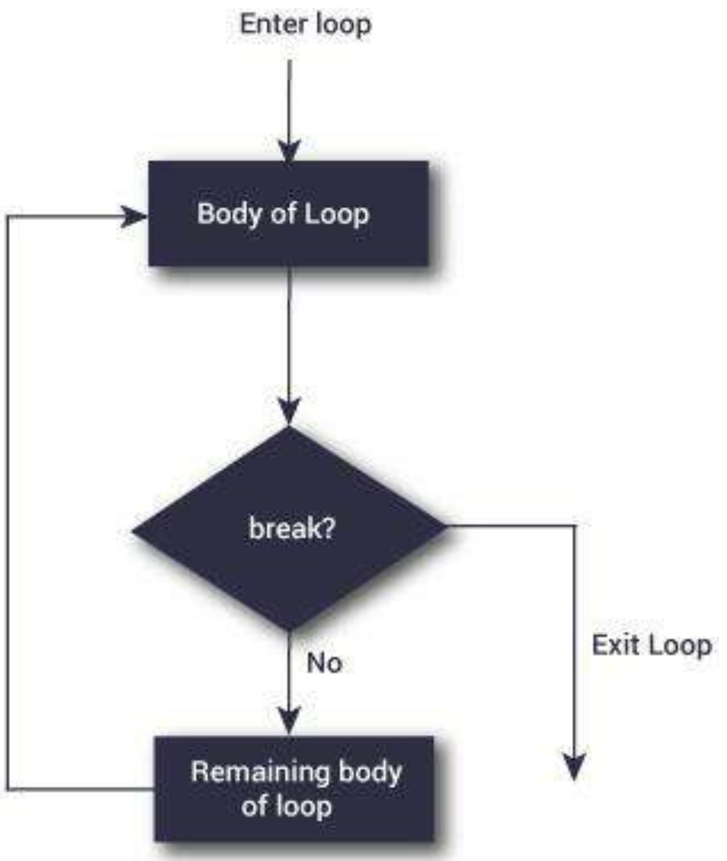
R repeat loop

- A repeat loop is used to iterate over a block of code multiple number of times.
- There is no condition check in repeat loop to exit the loop.
- We must ourselves put a condition explicitly inside the body of the loop and use the break statement to exit the loop. Failing to do so will result into an infinite loop.

Syntax of repeat loop

```
repeat {
statement
}
```

In the statement block, we must use the break statement to exit the loop.



Example of repeat loop

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

Output:

- In the above example, we have used a condition to check and exit the loop when x takes the value of 6.
- Hence, we see in our output that only values from 1 to 5 get printed.

Example of repeat loop

#Odd Numbers

```
count=1
repeat {
  print( count )
  count=count+ 2
  if (count > 10) break
}
```

Output

```
> source('t3.r')
[1] 1
[1] 3
[1] 5
[1] 7
[1] 9
```

#Factorial Numbers

```
num=as.integer(readline('Enter Number : '))
count=1
fact=1
repeat {
    fact=fact*count
    count=count+ 1
    if (count > num) break
  }
print( paste('Factorial : ', fact))
```

Output

```
> source('t3.r')
Enter Number : 5
[1] "Factorial :  120"
```

Contents to be covered

- For Loop
- Nested Loop
- SET Operations

Looping Control Structure - For

- A for loop is used to iterate over a vector in R programming.

Syntax of for loop

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

- Here, sequence is a vector and val takes on each of its value during the loop. In each iteration, statement is evaluated.

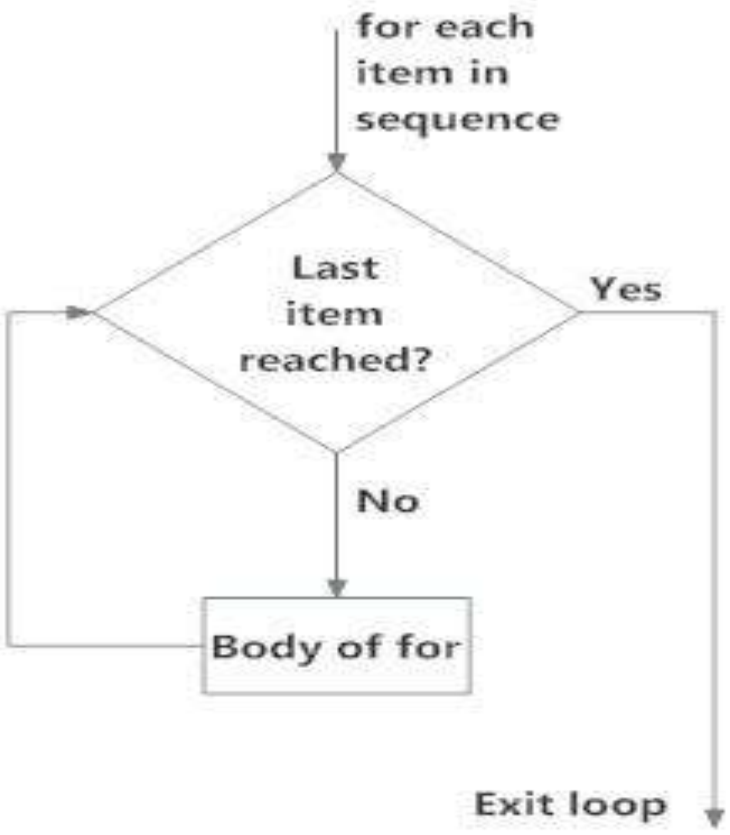
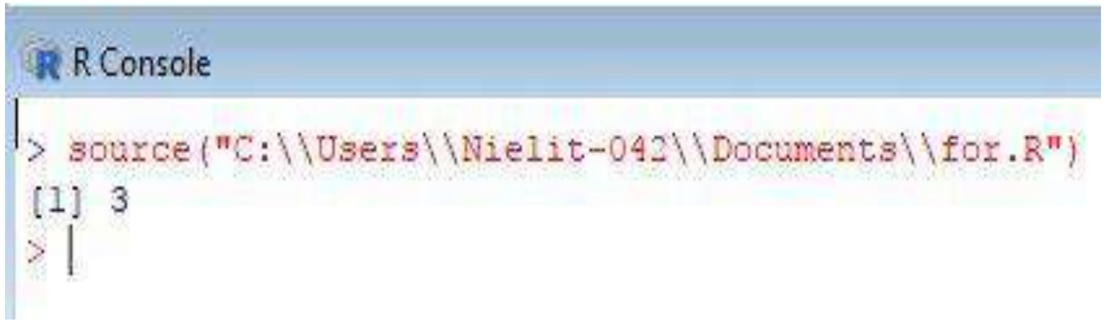


Fig: operation of for loop

Example of for loop

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

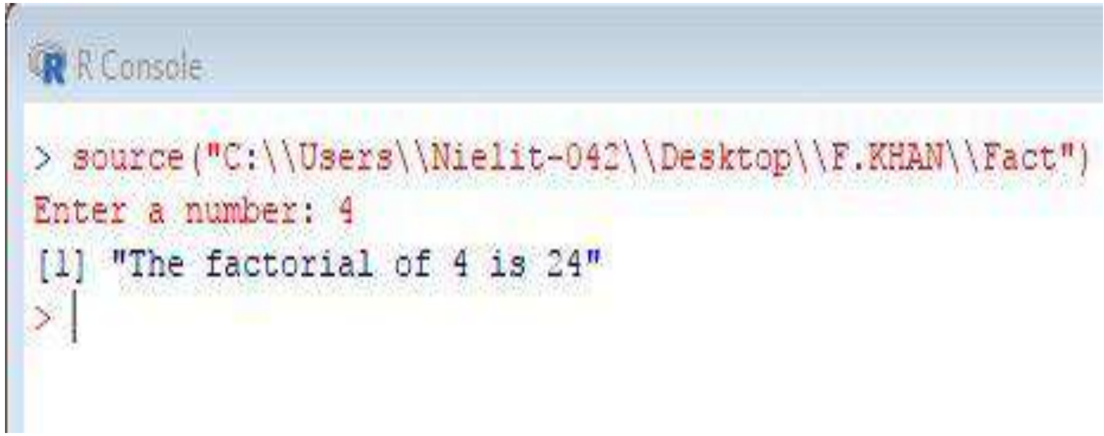


- In the above example, the loop iterates 7 times as the vector x has 7 elements.
- In each iteration, val takes on the value of corresponding element of x.
- We have used a counter to count the number of even numbers in x. We can see that x contains 3 even numbers.

Program to Find the Factorial of a Number

```
num = as.integer(readline(prompt="Enter a number: "))
factorial = 1
# check is the number is negative, positive or zero
if(num < 0) {
        print("Sorry, factorial does not exist for negative numbers")
} else if(num == 0) {
    print("The factorial of 0 is 1")
} else {
    for(i in 1:num) {
       factorial = factorial * i
    }
  print(paste("The factorial of", num ,"is",factorial))
}
```

Output:



Program to Find the Multiplication Table (From 1-10)

```
# take input from the user
num = as.integer(readline(prompt = "Enter a number: "))

# use for loop to iterate 10 times
for(i in 1:10) {
    print(paste(num, 'x', i, '=', num*i))
}
```

Output:

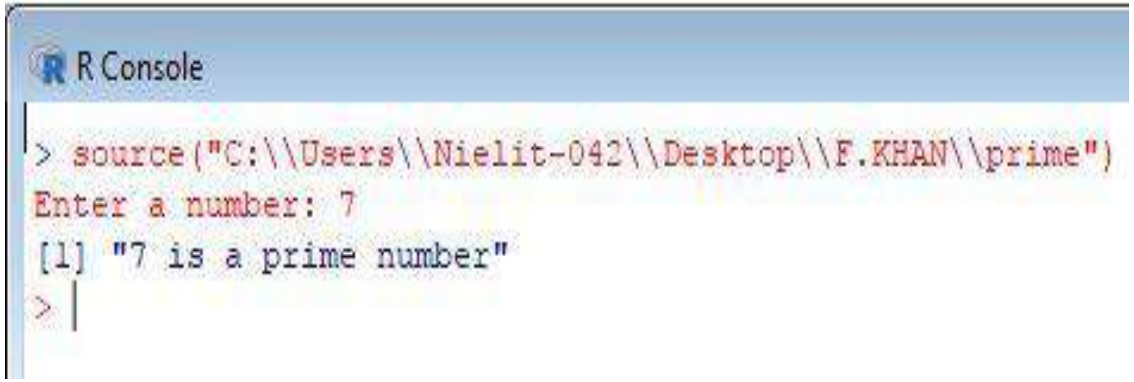
R Console

```
> source("C:\\Users\\Nielit-042\\Desktop\\F.KHAN\\table")
Enter a number: 5
[1] "5 x 1 = 5"
[1] "5 x 2 = 10"
[1] "5 x 3 = 15"
[1] "5 x 4 = 20"
[1] "5 x 5 = 25"
[1] "5 x 6 = 30"
[1] "5 x 7 = 35"
[1] "5 x 8 = 40"
[1] "5 x 9 = 45"
[1] "5 x 10 = 50"
> |
```

Program to check if the input number is prime or not

```
num = as.integer(readline(prompt="Enter a number: "))
flag = 0
if(num > 1) {
    # check for factors
    flag = 1

    for( i in 2:(num-1) ) {
        if ((num %% i) == 0) {
            flag = 0
            break
        }}}
if(num == 2)    flag = 1
if(flag == 1) {
       print(paste(num, "is a prime number"))
} else {
      print(paste(num, "is not a prime number"))
}
```

Output:



#Print the sum of items in vector

```
n=c( c(1,5,8,9) , c(2,6,9,10))
s=0
for (v in n) {
 s=s+v
}

cat('Sum = ',s)
```

```
> source('t3.r')
Sum =  50
```

#Print the sum of items of arrays

```
n=array(
c(1,5,8,9,2,6,9,10),dim=c(2,4))
s=0
for (v in n) {
 s=s+v
}

cat('Sum = ',s)
```

```
> source('t3.r')
Sum =  50
```

#Sum of items in List

```
n=list( c(1,5,8,9) , c(2,6,9,10))
s=0
for (v in n) {
 s=s+v
}

cat('Sum = ',s)
```

```
> source('t3.r')
Sum =  3 11 17 19
```

Nested Loops

- R programming language allows the usage of one loop inside another loop.
- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop”.

Syntax for Nested While loop

```
while(expression){
    while(expression){
          statement
     }
   statement
}
```

Example of Nested while loop

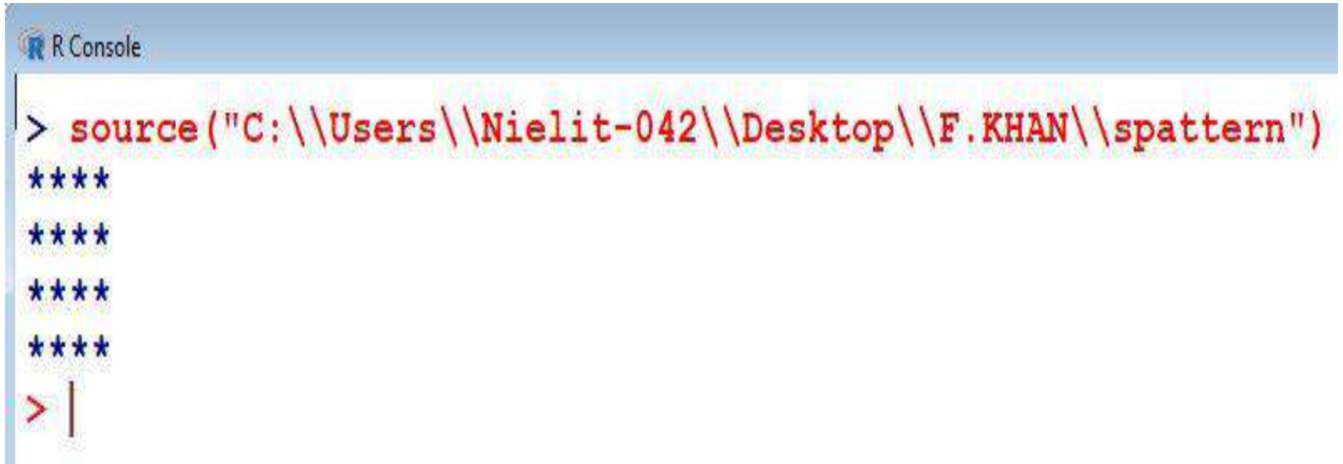
```
i<-1
while(i<3){
        j<-1
        while(j<3){
              cat(i, j,"\n")
              j=j+1
          }
    i=i+1
}
```

Output:

Example of Nested while loop (Pattern Printing)

```
i<-1
while(i<=4){
    j<-1
    while(j<=4){
        cat("*")
        j=j+1
    }
    cat("\n")
    i=i+1
}
```

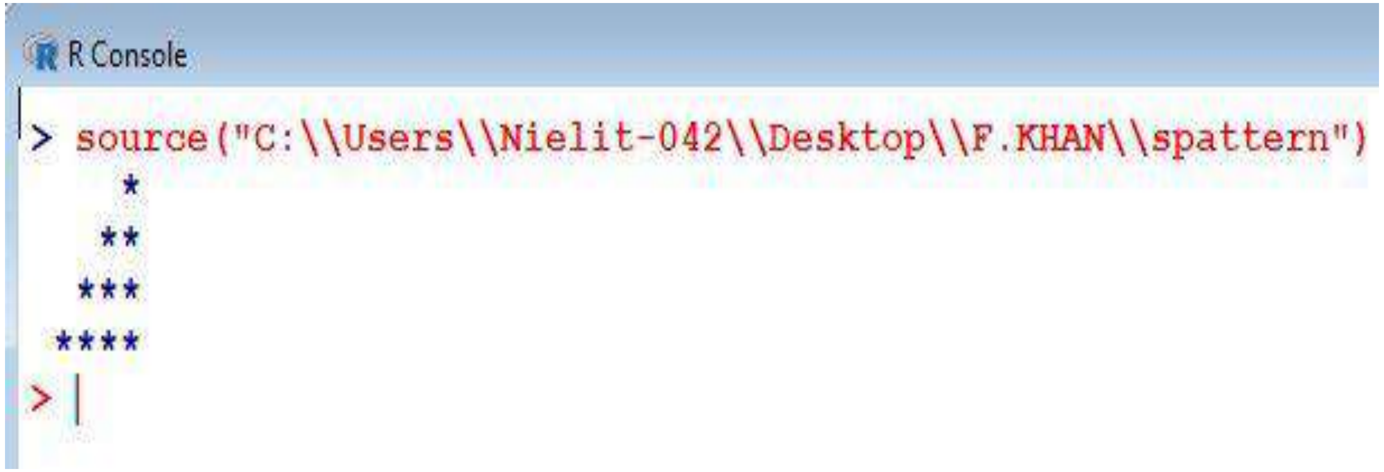
Output:



Example of Nested while loop (Pattern Printing)

```
i<-1
while(i<=4){
        s<-4
        while(s>=i){
            cat(" ")
            s=s-1
         }

        j<-1
        while(j<=i) {
             cat("*")
             j=j+1
         }

         cat("\n")
         i=i+1
}
```

Output:



Program to print all Palindrome Number b/w 1 to 100

```
i<-1
while(i<=100){
      sum=0
      temp=i

       while(temp > 0){
             digit=temp%%10
             sum=sum*10+digit
             temp=floor(temp/10)
          }

      if(i==sum){
          print(i)
      }

      i=i+1
}
```

Output:

```
R Console
> source("C:\\Users\\Nielit-042\\Desktop\\F.KHAN\\poli_nes")
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 11
[1] 22
[1] 33
[1] 44
[1] 55
[1] 66
[1] 77
[1] 88
[1] 99
>
```

Program to print all Prime Number b/w 1 to 100

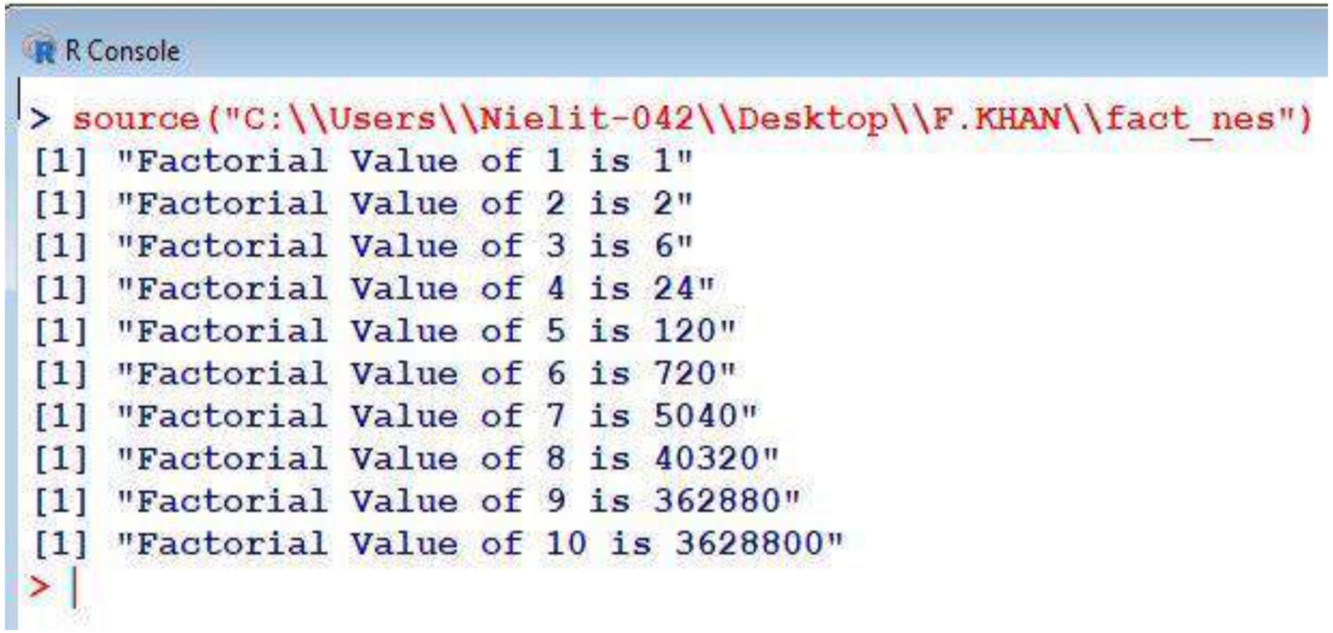
```
n=1
while(n<=100){
        f=1
        i=2

        while(i < n-1){
            if((n %% i)==0){
                f=0
                break
            }
          i=i+1
        }

if (n==1)    f=0
if( n==2)    f=1

if(f==1) print(n)

n=n+1
}
```

```
> source('t3.r')
[1] 2
[1] 3
[1] 5
[1] 7
[1] 11
[1] 13
[1] 17
[1] 19
[1] 23
[1] 29
[1] 31
[1] 37
[1] 41
[1] 43
[1] 47
[1] 53
[1] 59
[1] 61
[1] 67
[1] 71
[1] 73
[1] 79
[1] 83
[1] 89
[1] 97
>
```

Program to print all Factorial Value b/w 1 to 10

```
i<-1
while(i<=10){
   n=i
   f=1
   while(n>1){
     f=f*n
     n=n-1
    }

    print(paste("Factorial Value of",i,"is",f))
    i=i+1
}
```

Output:



#Sum of vectors in list

```
n=list( c(1,5,8,9) , c(2,6,9,10) )
for (v2 in n) {
  s=0
   for (v in v2) {
      s=s+v
   }

  cat('Sum = ',s)
}
```

Sum = 23   Sum = 27

#Sum of matrix

```
n=matrix( c(1,5,8,9,2,6,9,10), 2,4)
s=0
for (v in n) {
   s=s+v
   }

cat('Sum = ',s)
```

Sum = 50

#Sum of Array items

```
n=array( c(1,5,8,9,2,6,9,10), dim=c(2,4))
s=0

for (v in n) {
   s=s+v
   }

cat('Sum = ',s)
```

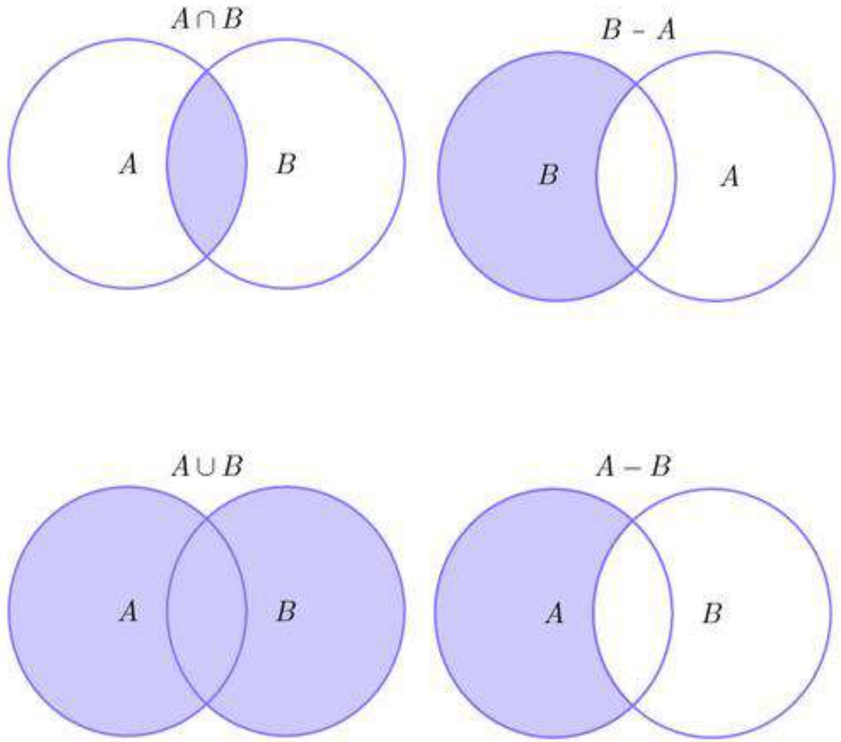
Sum = 50

SET Operations

```
a=c(10,20,30)
b=c(20,40,50)
c=c(30,20,10)

union(a,b)

intersect(a,b)

setdiff(a,b)

identical(a,c)

setequal(a,c)
```

OUTPUT

```
> union(a,b)
[1] 10 20 30 40 50

> intersect(a,b)
[1] 20

> setdiff(a,b)
[1] 10 30

> identical(a,c)
[1] FALSE

> setequal(a,c)
[1] TRUE
```



SET Operations

```
# two character vectors
set1 = c("some", "random", "words", "some")
set2 = c("some", "many", "none", "few")
```

 # union of set1 and set2

```
union(set1, set2)
[1] "some" "random" "words" "many" "none" "few"
```

```
# two character vectors
set3 = c("some", "random", "few", "words")
set4 = c("some", "many", "none", "few")
```

 # intersect of set3 and set4

```
intersect(set3, set4)
[1] "some" "few"
```

```
# two character vectors
set5 = c("some", "random", "few", "words")
set6 = c("some", "many", "none", "few")
```

 # difference between set5 and set6

```
setdiff(set5, set6)
[1] "random" "words”
```

setequal() -allows us to test the equality of two character vectors. If the vectors contain the same elements, setequal() returns TRUE (FALSE otherwise)

```
# three character vectors
set7 = c("some", "random", "strings")
set8 = c("some", "many", "none", "few")
set9 = c("strings", "random", "some")

# set7 == set8?
setequal(set7, set8)          ## [1] FALSE

# set7 == set9?
setequal(set7, set9)          ## [1] TRUE
```

identical()- to test whether two vectors are exactly equal (element by element)

```
# set7 identical to set7?
 identical(set7, set7)        ## [1] TRUE

# set7 identical to set9?
identical(set7, set9)         ## [1] FALSE
```

SET Operations

Is.element() - To test if an element is contained in a given set of character strings

```
# three vectors
set10 = c("some", "stuff", "to", "play", "with")
elem1 = "play" elem2 = "crazy“

# elem1 in set10?
is.element(elem1, set10)      ## [1] TRUE

# elem2 in set10?
is.element(elem2, set10)      ## [1] FALSE


# elem1 in set10?
elem1 %in% set10              ## [1] TRUE
# elem2 in set10?
elem2 %in% set10              ## [1] FALSE
```

sort() allows us to sort the elements of a vector, either in increasing order (by default) or in decreasing order

```
set11 = c("today", "produced", "example", "beautiful", "a",
"nicely")
```

 # sort (decreasing order)

```
sort(set11)
## [1] "a" "beautiful" "example" "nicely" "produced" "today"
```

 # sort (increasing order)

```
sort(set11, decreasing = TRUE)
## [1] "today" "produced" "nicely" "example" "beautiful" "a"
```

Repetition with rep()

```
# repeat 'x' 4 times
paste(rep("x", 4), collapse = "")
## [1] "xxxx”
```

Contents to be covered

- R Data Structure – Vector, Array, Matrix, List , DataFrame, Factor
- R Vector

What are Data Structures?

- A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks.
- Data structures in R programming are tools for holding multiple values.
- Data structures are made up of different data types. A **data type** defines what kind of data is held in a value.
- R's base data structures are organized by
  - their dimensionality (1D, 2D, or nD) and
  - they're homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types).

DIMENSION	HOMOGENOUS	HETEROGENEOUS
1D	Vector	**List**
2D	Matrix	**Data.frame**
nD	Array	

The most essential data structures used in R include:

1. **Vectors**
2. **Lists**
3. **Data.frames**
4. **Matrices**
5. **Arrays**
6. **Factors**

1. Vector

- Vectors are used to group together multiple values of the same data type, such as a range of numbers.
- Vectors can be created using the c() function, like this:

  V = c(1:60)

  Running the above code will create the vector named **'V'**, containing 60 numbers, from 1 through 60.
- Vectors can also hold character values, where each of values would have to be explicitly enclosed in double quotes.
- Technically it's possible to store data of different types in a vector, but it's not advisable, as all the values are likely to be converted to the character type.

2. List

- Lists are very similar to vectors, but with the added bonus that they can store values of any data type.
- For example, you can have a list that contains 45 numbers, 26 words or phrases, and 33 vectors.
- Create lists using the list() function, like this:

  myList = list(1, 2, 3, "one", "'two", "three", numericVector, characterVector, exampleDataFrame)

- The code above creates the list named **'myList'**, and stores into it 9 items: 3 numeric values, 3 character values, 2 vectors, and a data frame.
- You can even store a list inside another list.

3. Data Frames

- Data frames store data in two dimensions, rows and columns, much like a spreadsheet.
- Like lists, they can hold values of different data types, but each column in the data frame must hold values of the same type.
- For example, a single column can't hold both words and numbers. Also, each column must hold the same number of values. So, if column one has 10 values, column two must also have 10 values, and so on.
- The following code creates 3 different vectors named 'monthID', 'monthName' and 'temperature', and copies them into the data frame named 'myTable'.

```
monthID = c(1:12)
monthName = c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
temperature = c(3, 3, 7, 10, 14, 18, 21, 21, 18, 12, 7, 4)
myTable = data.frame(monthID, monthName, temperature)
```

- Running the above code will result in a data frame with 3 columns and 12 rows.
- Columns 'monthID' and 'temperature' hold numeric values, while 'monthName' holds character values.

4. Matrices

- A matrix is a data structure that is between a vector and a data frame.
- Like a vector, it can hold values of only one data type.
- But, like a data frame, it can store and display that data in tabular format, columns, and rows, like a spreadsheet.
- create a matrix, using the matrix() function:

```
myMatrix = matrix(1:60,  ncol=3)
```

- The above code will create a matrix named 'myMatrix' with three columns and 20 rows because 60/3 = 20.

5. Arrays

- R arrays are the data objects which can store data in more than two dimensions.
- In arrays, data is stored in the form of matrices, rows, and columns.
- We can use the matrix level, row index, and column index to access the matrix elements.
- create a arrays, using the **array**() function:
  - vector =c(1,2,3,4,5,6,7,8,9,10,11,12)
  - result=array(vector, dim = c(2,3,2))
  - print(result)
- The above code will create a array named ‘result' with dimension row=2,col=3,matrix=2.

```
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

6. Factors

- Factor is a data structure used for fields that takes only predefined, finite number of values (categorical data).
- For example: a data field such as marital status may contain only values from single, married, separated, divorced, or widowed.
- These predefined, distinct values are called levels.
- create a factor, using the factor() function:

```
x = factor(c("single", "married", "married", "single"))

[1] single  married married single
Levels: married single
```

Vector Basics

There are two types of vectors:

- **Atomic** vectors, of which there are six types: **logical**, **integer**, **double**, **character**, **complex**, and **raw**.
- Integer and double vectors are collectively known as **numeric** vectors.
- **Lists**, which are sometimes called recursive vectors because lists can contain other lists.
- The main difference between atomic vectors and lists is that atomic vectors are **homogeneous**, while lists can be **heterogeneous**.
- One other related object: **NULL**. NULL is used to represent the absence of a vector (as opposed to NA which is used to represent the absence of a value in a vector). NULL typically behaves like a vector of length 0.
- Every vector has two key properties:
    - Its **type**, which you can determine with **typeof**().
    - **length**(), which determine the count of items.

- **typeof**(letters)
  #> [1] "character"
- **typeof**(1:10)
  #> [1] "integer"

- Its **length**, which you can determine with length().
  x <- **list**("a", "b", 1:10)
  length(x)
  #> [1] 3

Types of atomic vector

- The four most important types of atomic vector are **logical, integer, double, and character**.
- Raw and complex are rarely used during a data analysis.

Logical

- Logical vectors are the simplest type of atomic vector , they can take only three possible values: **FALSE, TRUE, and NA.**
- Logical vectors are constructed with comparison operators, as described in comparisons.
- It can also be created with c():

Example

```
1:10 %% 3 == 0
#>  [1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE

c(TRUE, TRUE, FALSE, NA)
#> [1]  TRUE  TRUE FALSE    NA
```

Numeric

- Integer and double vectors are known collectively as numeric vectors.
- In R, numbers are doubles by default. To make an integer, place an L after the number.
- **Integers** have one special value: NA, while **doubles** have four: NA, NaN, Inf and -Inf.
- All three special values NaN, Inf and -Inf can arise during division.

```
typeof(1)
#> [1] "double“

typeof(1L)
#> [1] "integer"1.5L
#> [1] 1.5
```

Numeric

c(-1, 0, 1) / 0

#> [1] -Inf NaN Inf

- Avoid using == to check for these other special values.
- Instead use the helper functions is.finite(), is.infinite(), and is.nan():

	0	Inf	NA	NaN
is.finite()	x			
is.infinite()		x		
is.na()			x	x
is.nan()				x

Character

- Character vectors are the most complex type of atomic vector, because each element of a character vector is a string, and a string can contain an arbitrary amount of data.

Coercion

- There are two ways to **convert**, or **coerce**, one type of vector to another:
- **Explicit coercion** happens when you call a function like **as.logical(), as.integer(), as.double(), or as.character().**
- **Implicit coercion** happens when you use a vector in a specific context that expects a certain type of vector.
  - For example, when you use a logical vector with a numeric summary function, or when you use a double vector where an integer vector is expected.
  - In this case TRUE is converted to 1 and FALSE converted to 0.
  - That means the sum of a logical vector is the number of trues, and the mean of a logical vector is the proportion of trues
- Example

```
x <- sample(20, 100, replace = TRUE)
y <- x > 10
sum(y)                          # how many are greater than 10?
#> [1] 38
mean(y)                         # what proportion are greater than 10?
#> [1] 0.38
```

Coerce Vector (conversion)

- It is important to remember that a vector can only be composed of one data type. This means that you cannot have both a numeric and a character in the same vector. If you attempt to do this, the lower ranking type will be *coerced* into the higher ranking type.
- For example: c(1.5, "hello") results in c("1.5", "hello") where the numeric 1.5 has been coerced into the character data type.
- The hierarchy for coercion is:

  logical < integer < numeric < character

- Logicals are coerced a bit differently depending on what the highest data type is. c(TRUE, 1.5) will return c(1, 1.5) where TRUE is coerced to the numeric 1 (FALSE converted to a 0).
- On the other hand, c(TRUE, "this_char") is converted to c("TRUE", "this_char").
- The vectors a, b, and c have been defined for you from the following commands:

```
a <- c(1L , "I am a character")
b <- c(TRUE, "Hello")
c <- c(FALSE, 2)
```

- Normally, whatever is converted implicitly is referred to as **coercion**
- if converted explicitly then it is known as **casting.**
- **Conversion** signifies both types- coercion and casting.
- ***Explicit coercion to character***
- There are two functions to do so **as.character()** and **as.string()**.

FUNCTION	DESCRIPTION
as.logical	Converts the value to logical type. • If 0 is present then it is converted to FALSE • Any other value is converted to TRUE
as.integer	Converts the object to integer type
as.double	Converts the object to double precision type
as.complex	Converts the object to complex type
as.list	It accepts only dictionary type or vector as input arguments in the parameter

Test Functions

- Sometimes you want to do different things based on the type of vector. One option is to use typeof().
- Another is to use a test function which returns a TRUE or FALSE.
- Base R provides many functions like is.vector() and is.atomic(), but they often return surprising results.

	lgl	int	dbl	Chr	list
is_logical()	x				
is_integer()		x			
is_double()			x		
is_numeric()		x	x		
is_character()				X	
is_atomic()	x	x	x	X	
is_list()					x
is_vector()	x	x	x	X	x

Naming Vectors

- All types of vectors can be named. You can name them during creation with c():

```
c(x = 1, y = 2, z = 4)
#> x y z
#> 1 2 4
```

```
setNames(1:3, c("a", "b", "c"))
#> a b c
#> 1 2 3
```

Named vectors are most useful for subsetting.

Subsetting

- A numeric vector containing only integers. The integers must either be all positive, all negative, or zero.
- Subsetting with positive integers keeps the elements at those positions:

```
x <- c("one", "two", "three", "four", "five")
x[c(3, 2, 5)]        #> [1] "three" "two"   "five"
```

- By repeating a position, you can actually make a longer output than input:

```
x[c(1, 1, 5, 5, 5, 2)]       #> [1] "one"  "one"  "five" "five" "five" "two"
```

- Negative values drop the elements at the specified positions:

```
x[c(-1, -3, -5)]       #> [1] "two"  "four"
```

- It's an error to mix positive and negative values:

```
x[c(1, -1)]       #> Error in x[c(1, -1)]:
only 0's may be mixed with negative subscripts
```

- Subsetting with a logical vector keeps all values corresponding to a TRUE value. This is most often useful in conjunction with the comparison functions.

```
x <- c(10, 3, NA, 5, 8, 1, NA)
# All non-missing values of x

x[!is.na(x)]
#> [1] 10  3  5  8  1

# All even (or missing!) values of x
x[x %% 2 == 0]
#> [1] 10 NA  8 NA
```

- If you have a named vector, you can subset it with a character vector:

```
x <- c(abc = 1, def = 2, xyz = 5)
x[c("xyz", "def")]
#> xyz def
#>   5   2
```

Recursive Vectors (Lists)

- Lists are a step up in complexity from atomic vectors, because lists can contain other lists.
- This makes them suitable for representing hierarchical or tree-like structures. You create a list with list():

```
x <- list(1, 2, 3)
x
#> [[1]]
#> [1] 1
#> [[2]]
#> [1] 2
#> [[3]]
#> [1] 3
```

- A very useful tool for working with lists is **str()** because it focuses on the **str**ucture, not the contents.

```
str(x)
#> List of 3
#>  $ : num 1
#>  $ : num 2
#>  $ : num 3
```

```
x_named <- list(a = 1, b = 2, c = 3)
str(x_named)
#> List of 3
#>  $ a: num 1
#>  $ b: num 2
#>  $ c: num 3
```

- Unlike atomic vectors, list() can contain a mix of objects:

```
y <- list("a", 1L, 1.5, TRUE)
str(y)
#> List of 4
#>  $ : chr "a"
#>  $ : int 1
#>  $ : num 1.5
#>  $ : logi TRUE
```

- Lists can even contain other lists!

```
z <- list(list(1, 2), list(3, 4))
str(z)
#> List of 2
#>  $ :List of 2
#>   ..$ : num 1
#>   ..$ : num 2
#>  $ :List of 2
#>   ..$ : num 3
#>   ..$ : num 4
```

Conversion Example

#Example-1

Creating a list

```
x<-c(0, 1, 0, 3)

# Converting it to character type
as.character(x)
```

Output:

[1] "0" "1" "0" "3”

#Example-2

```
# Creating a list
x<-c("q", "w", "c")
as.numeric(x)
as.logical(x)
```

```
[1] NA NA NA
Warning message:
NAs introduced by coercion
[1] NA NA NA
```

Example-3

```
# Creating a list
x<-c(0, 1, 0, 3)
# Checking its class
class(x)
# Converting it to integer type
as.numeric(x)

# Converting it to double type
as.double(x)

# Converting it to logical type
as.logical(x)

# Converting it to a list
as.list(x)

# Converting it to complex numbers
as.complex(x)
```

Output:

```
[1] "numeric“
[1] 0 1 0 3
[1] 0 1 0 3
[1] FALSE  TRUE FALSE  TRUE
[[1]]
[1] 0
[[2]]
[1] 1
[[3]]
[1] 0
[[4]]
[1] 3
[1] 0+0i 1+0i 0+0i 3+0i
```

String Matching

Finding a String : Search for a particular pattern in a string.

grep() - Find the location of the required string/pattern.

To find all instances of ‘in’ in the string

```
x=c('rain', 'spain', ‘TRAIN', 'training', 'track')
grep('in', x)     #[1] 1 2 4

x=c('rain', 'spain', 'TRAIN', 'training', 'track')
grep('in', x, ignore.case = TRUE)      #[1] 1 2 3 4
```

To find all instances in the string

```
a=c(10,100,101,121,130,105)
r=grep("1[^0-2]",a, value=T)
print(r)

[1] "130“

b=grep("b[a-d]", c("abc", "bada", "cca", "abd"), value=TRUE)
print(b)

[1] "abc"  "bada" "abd"
```

grepl() - Gives the logical –TRUE , if required string/pattern found.

To find all instances of ‘in’ in the string

```
x=c('rain', 'spain', ‘TRAIN', 'training', 'track')
grepl('in', x)

[1]  TRUE  TRUE FALSE  TRUE FALSE

x=c('rain', 'spain', 'TRAIN', 'training', 'track')
grepl('in', x, ignore.case = TRUE)

[1]  TRUE  TRUE  TRUE  TRUE FALSE
```

String Matching

regexpr() - Find the occurrence of the required string/pattern. If found, gives the index , else -1

1. To find all instances of 'in' in the string

```
x=c('rain', 'spain', 'TRAIN', 'training', 'track')
regexpr('in', x)

[1]  3  4 -1  4 -1
attr(,"match.length")
[1]  2  2 -1  2 -1
attr(,"index.type")
```

2.

```
x=c('rain', 'spain', 'TRAIN', 'training', 'track')
regexpr('in', x, ignore.case = TRUE)

[1]  3  4  4  4 -1
attr(,"match.length")
[1]  2  2  2  2 -1
attr(,"index.type")
```

3. To find all instances of 'ra' or in the string

```
x=c('rain', 'spain', 'TRAIN', 'training', 'track')
regexpr('^r[ar]', x)

[1]  1 -1 -1 -1 -1
attr(,"match.length")
[1]  2 -1 -1 -1 -1
attr(,"index.type")
```

4.

```
x=c('rain', 'spain', 'TRAIN', 'training', 'track')
regexpr('[inIN]$', x)

[1]  4  5  5 -1 -1
attr(,"match.length")
[1]  1  1  1 -1 -1
attr(,"index.type")
```

5. To find all instances of 'in' in the string

```
x=c('rain', 'spain', 'TRAIN', 'training', 'spleen')
regexpr('sp[a-z]{3}', x)

[1] -1  1 -1 -1  1
attr(,"match.length")
[1] -1  5 -1 -1  5
attr(,"index.type")
```

6.

```
x=c('rain', 'spain', 'TRAIN', 'training', 'spleen')
regexpr('sp[a-z]{3}$', x)

[1] -1  1 -1 -1 -1
attr(,"match.length")
[1] -1  5 -1 -1 -1
attr(,"index.type")
```

String Substitution

sub (patternSTR, replaceSTR, STRING) –
It will replace the first occurrence of string.

1. **To replace all ‘ra’ with ‘RA’ in the string**

```
x=c('rain', 'spain', 'TRAIN', 'training', 'track')
sub('ra', 'RA', x)

[1] "RAin"    "spain"   "TRAIN"   "tRAining" "tRAck"
```

2. **To replace all ‘in’ with ‘XX’ in the string**

```
x=c('rain', 'spain', 'TRAIN', 'training', 'track')
sub('in', 'XX', x)

[1] "raXX"    "spaXX"   "TRAIN"   "traXXing" "track"
```

gsub (patternSTR, replaceSTR, STRING) –
It will replace the all occurrence of string.

1. **To replace all ‘in’ with ‘XX’ in the string**

```
x=c('rain', 'spain', 'TRAIN', 'training', 'track')
gsub('in', 'XX', x)

[1] "raXX"    "spaXX"   "TRAIN"   "traXXXXg" "track"
```

Contents to be covered

- R List
- R DataFrame

R List

- List is a data structure having components of mixed data types.
- A vector having all elements of the same type is called **atomic vector** but a vector having elements of different type is called **list**.
- We can check if it’s a list with **typeof**() function and find its length using **length**().
- Example of a list having three components each of different data type.

```
x = list ( a=2.5 , b=TRUE , c= c(1,2,3))
$a
[1] 2.5
$b
[1] TRUE
$c
[1] 1 2 3

> typeof(x)
[1] "list“

> length(x)
[1] 3
```

R List

- Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it.
- A list can also contain a matrix or a function as its elements.
- List is created using **list()** function.

Creating a List

- Following is an example to create a list containing strings, numbers, vectors and a logical values.

```
# Create a list containing strings, numbers, vectors and a logical
 values.list_data  <-  list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)

print(list_data)
```

```
[[1]]
[1] ″Red“
[[2]]
[1] ″Green″
[[3]]
[1] 21 32 11
[[4]]
[1] TRUE
[[5]]
[1] 51.23
[[6]]
[1] 119.1
```

Naming List Elements

- The list elements can be given names and they can be accessed using these names.
- # Create a list containing a vector, a matrix and a list.

```
list_data = list(  c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8),  nrow = 2),   list("green",12.3)
)
```

 # Give names to the elements in the list.

```
names(list_data) = c("1st Quarter", "A_Matrix", "A Inner list")
```

 # Show the list.

```
print(list_data)
```

Output:

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"
$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
$A_Inner_list
$A_Inner_list[[1]]
[1] "green"
$A_Inner_list[[2]]
[1] 12.3
```

Accessing List Elements

- Elements of the list can be accessed by the index of the element in the list.
- In case of named lists it can also be accessed using the names.

 # Create a list containing a vector, a matrix and a list.

list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))

 # Give names to the elements in the list.

names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

 # Access the first element of the list.

print(list_data[1])

 # Access the thrid element. As it is also a list, all its elements will be printed.

print(list_data[3])

 # Access the list element using the name of the element.

print(list_data$A_Matrix)

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"
$A_Inner_list
$A_Inner_list[[1]]
[1] "green"
$A_Inner_list[[2]]
[1] 12.3
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
```

रा.इ.सू.प्रौ.सं NIELIT

Manipulating List Elements

R Programming

- We can add, delete and update list elements.
- We can add elements only at the end of a list. But we can update any element.

 # Create a list containing a vector, a matrix and a list.

list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))

 # Give names to the elements in the list.

names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

 # Add element at the end of the list.

list_data[4] <- "New element“

print(list_data[4])

 # Remove the last element.

list_data[4] <- NULL

 # Print the 4th Element.

print(list_data[4])

 # Update the 3rd Element.

list_data[3] <- "updated element“

print(list_data[3])

```
[[1]]
[1] ″New element“
$<NA>NULL
$`A Inner list`
[1] ″updated element″
```

Merging List

- You can merge many lists into one list by placing all the lists inside one list() function.

```
# Create two lists.
list1 <- list(1,2,3)
list2 <- list("Sun","Mon","Tue")

# Merge the two lists.
list <- c(list1,list2)

# Print the merged list.
print(list)
```

```
[[1]]
[1] 1
[[2]]
[1] 2
[[3]]
[1] 3
[[4]]
[1] ″Sun“
[[5]]
[1] ″Mon“
[[6]]
[1] ″Tue″
```

Converting List to Vector

- A list can be converted to a vector so that the elements of the vector can be used for further manipulation.
- All the arithmetic operations on vectors can be applied after the list is converted into vectors.
- To do this conversion, we use the **unlist()** function. It takes the list as input and produces a vector.

```
# Create lists.
list1 <- list(1:5)
print(list1)

list2 <-list(10:14)
print(list2)

# Convert the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)
print(v1)
print(v2)

# Now add the vectors
result <- v1+v2
print(result)
```

```
[[1]]
[1] 1 2 3 4 5
[[1]]
[1] 10 11 12 13 14

[1] 1 2 3 4 5
[1] 10 11 12 13 14
[1] 11 13 15 17 19
```

DataFrame

- 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.
- Following are the characteristics of a data frame.
- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Create DataFrame

```
# 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)

Get the Structure of the DataFrame

- The structure of the data frame can be seen by using **str()** 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
)
# Get the structure of the data frame.
str(emp.data)
```

Conversion specification	Description	Example
%a	Abbreviated weekday	Sun, Thu
%A	Full weekday	Sunday, Thursday
%b or %h	Abbreviated month	May, Jul
%B	Full month	May, July
%d	Day of the month 01-31	27, 07
%m	Month 01-12	05, 07
%y	Year without century 00-99	84, 05
%Y	Year with century on input: 00 to 68 prefixed by 20 69 to 99 prefixed by 19	1984, 2005
%D	Date formatted %m/%d/%y	05/27/84, 07/07/05

Output:

```
'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 DataFrame

- 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))
```

```
     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
 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 DataFrame

- Extract specific column from a data frame using column name.

```
# 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
)

# Extract Specific columns.

result <- data.frame(emp.data$emp_name,  emp.data$salary)
print(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 Data from DataFrame

 # 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
)
```

 # Extract first two columns.

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

	emp_id	emp_name
1	1	Rick
2	2	Dan
3	3	Michelle
4	4	Ryan
5	5	Gary

 # Extract first two rows.

```
result <- emp.data[ 1:2, ]
print(result)
```

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

Expand DataFrame

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

Add Column
- Just add the column vector using a new column name.

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
)
 # Add the "dept" column.
dept= c("IT","Operations","IT","HR","Finance")

emp.data$dept <- dept
v <- emp.data
print(v)

emp=cbind(emp.data , dept)
print(emp )
```

	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

Expand DataFrame

Add Row

- To add more rows permanently to an existing data frame, we need to bring in the new rows in 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 first 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")),
  dept = c("IT","Operations","IT","HR","Finance"),
  stringsAsFactors = FALSE
)
```

	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

Expand DataFrame

```
# 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)
```

```
  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
6      6    Rasmi      578.00    2013-05-21     IT
7      7    Pranab     722.50    2013-07-30     Operations
8      8    Tusar      632.80    2014-06-17     Fianance
```

Delete from DataFrame

```
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")),
  dept = c("IT","Operations","IT","HR","Finance"),
  stringsAsFactors = FALSE
)
```

	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

#To delete the Column

```
emp.data$salary = NULL
print(emp.data)
```

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

#To delete the Row

```
emp.data[ -5 ,]=NULL
print(emp.data)
```

	emp_id	emp_name	start_date	dept
1	1	Rick	2012-01-01	IT
2	2	Dan	2013-09-23	Operations
3	3	Michelle	2014-11-15	IT
4	4	Ryan	2014-05-11	HR

Search Data from DataFrame

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","RGary"),
  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 = TRUE
)

#Query1
f=format(emp.data$start_date,'%Y')==2014
print(f)
print(emp.data[f,])

#Query2
f=grepl('R[iy]',emp.data$emp_name)
print(f)
print(emp.data[f,])

#Query3
f=grep('R[iy]',emp.data$emp_name)
print(f)
print(emp.data[f,])

print(emp.data[f, c("emp_name",'salary')]  )
```

```
  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     RGary     843.25  2015-03-27
```

```
[1] FALSE  FALSE  TRUE  TRUE  FALSE

  emp_id emp_name  salary  start_date
3   3    Michelle  611     2014-11-15
4   4    Ryan      729     2014-05-11
```

```
[1] TRUE  FALSE  FALSE  TRUE  FALSE

  emp_id  emp_name  salary  start_date
1   1     Rick      623.3   2012-01-01
4   4     Ryan      729.0   2014-05-11
```

```
[1] 1  4
  emp_id  emp_name  salary  start_date
1   1     Rick      623.3   2012-01-01
4   4     Ryan      729.0   2014-05-11

   emp_name  salary
1  Rick      623.3
4  Ryan      729.0
```

Applying Aggregate Function on DataFrame

apply() function- It takes Data frame or matrix as an input and gives output in vector, list or array.

apply(X, MARGIN, FUN)

-MARGIN: take a value or range between 1 and 2 to define where to apply t he function: -

MARGIN=1: the manipulation is performed on rows –

MARGIN=2: the manipulation is performed on columns –

FUN: tells which function to apply. Built functions like mean, median, sum, min, max and even user-defined functions can be applied>

```
df=read.csv('d:/datacsv/salesdata.csv')
print(df)
df['Total']= apply(df[,c(2:6)], 1, sum)
```

```
month= apply(df[,c(2:6)], 2, max)
print(month)
```

```
month= apply(df[,c(2:6)], 2, median)
print(month)
```

	Name	Jan	Feb	Mar	Apr	May	Jun
1	Ajay	10	21	23	31	7	22
2	Vijay	13	17	12	29	14	16
3	Sanjay	17	15	16	13	76	10
4	Ajit	45	21	7	34	22	34
5	Vikas	22	56	76	34	22	16
6	Vipul	12	17	22	36	31	23
7	Rakesh	31	23	27	41	32	22

	Name	Jan	Feb	Mar	Apr	May	Jun	Total
1	Ajay	10	21	23	31	7	22	92
2	Vijay	13	17	12	29	14	16	85
3	Sanjay	17	15	16	13	76	10	137
4	Ajit	45	21	7	34	22	34	129
5	Vikas	22	56	76	34	22	16	210
6	Vipul	12	17	22	36	31	23	118
7	Rakesh	31	23	27	41	32	22	154

Jan	Feb	Mar	Apr	May
45	56	76	41	76

Jan	Feb	Mar	Apr	May
17	21	22	34	22

Contents to be covered

- R Factor
- R Matrix

What is Factor in R?

- Factors are variables in R which take on a limited number of different values; such variables are often referred to as categorical variables.
- In a dataset, we can have two types of variables:
  - **categorical** and
  - **continuous**.
- In a **categorical variable**, the value is limited and usually based on a particular finite group.
  - Example - countries, year, gender, occupation.
- A **continuous variable**, can take any values, from integer to decimal.
  - Example, the revenue, price of a share, etc.

What is Factor in R?

Syntax : factor(x = character(), levels, labels = levels, ordered = is.ordered(x))

- **x**: A vector of data. Need to be a string or integer, not decimal.
- **Levels**: A vector of possible values taken by x. This argument is optional. The default value is the unique list of items of the vector x.
- **Labels**: Add a label to the x data. For example, 1 can take the label `male` while 0, the label `female`.
- **ordered**: Determine if the levels should be ordered.

```
# Create gender vector
  gender_vector = c("Male", "Female", "Female", "Male", "Male")
  class(gender_vector)
# Convert gender_vector to a factor
  factor_gender_vector = factor(gender_vector)
  class(factor_gender_vector)
```

Output:

```
## [1] "character“
## [1] "factor"
```

What is Factor in R?

- It is important to transform a **string** into factor when we perform Machine Learning task.
- A categorical variable can be divided into **nominal categorical variable** and **ordinal categorical variable**.

Nominal Categorical Variable

- A categorical variable has several values but the order does not matter. For instance, male or female categorical variable do not have ordering.

```
# Create a color vector
color_vector <- c('blue', 'red', 'green', 'white', 'black', 'yellow')

# Convert the vector to factor
factor_color <- factor(color_vector)
factor_color
```

Output:

```
## [1] blue   red    green  white  black  yellow
## Levels: black blue green red white yellow
```

From the factor_color, we can't tell any order.

What is Factor in R?

Ordinal Categorical Variable

- Ordinal categorical variables do have a natural ordering.
- We can specify the order, from the lowest to the highest with **order = TRUE** and highest to lowest with **order = FALSE**.

Example:

- We can use summary to count the values for each factor.

Create Ordinal categorical vector

```
day_vector <- c('evening', 'morning', 'afternoon', 'midday', 'midnight', 'evening')
```

Convert `day_vector` to a factor with ordered level

```
factor_day <- factor(day_vector, order = TRUE, levels =c('morning', 'midday', 'afternoon', 'evening', 'midnight'))
```

Print the new variable

```
factor_day                    [1] evening   morning   afternoon midday   midnight  evening
                              Levels: morning < midday < afternoon < evening < midnight


summary(factor_day)           ##   morning   midday   afternoon   evening   midnight
                              ##      1         1         1          2         1
```

- R ordered the level from 'morning' to 'midnight' as specified in the levels parenthesis.

What is Factor in R?

Continuous Variables

- Continuous class variables are the default value in R.
- They are stored as numeric or integer.

```
a=1:30
f1=cut( a , breaks=2)
print(f1)
print(summary(f1))

f2=cut(a , breaks=3)
print(f2)
print(summary(f2))

f3=cut(a , breaks=c(0,10,20,30))
print(f3)
print(summary(f3))
```

#Breaks=2

```
[1] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5]
[11] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5] (0.971,15.5] (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]
[21] (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]    (15.5,30]

Levels: (0.971,15.5] (15.5,30]
(0.971,15.5]    (15.5,30]
          15           15
```

#Breaks=3

```
[1] (0.971,10.7] (0.971,10.7] (0.971,10.7] (0.971,10.7] (0.971,10.7] (0.971,10.7] (0.971,10.7] (0.971,10.7] (0.971,10.7] (0.971,10.7]
[11] (10.7,20.3]  (10.7,20.3]  (10.7,20.3]  (10.7,20.3]  (10.7,20.3]  (10.7,20.3]  (10.7,20.3]  (10.7,20.3]  (10.7,20.3]  (10.7,20.3]
[21] (20.3,30]    (20.3,30]    (20.3,30]    (20.3,30]    (20.3,30]    (20.3,30]    (20.3,30]    (20.3,30]    (20.3,30]    (20.3,30]

Levels: (0.971,10.7] (10.7,20.3] (20.3,30]
(0.971,10.7]  (10.7,20.3]    (20.3,30]
          10           10           10
```

#Breaks=c(0,10,20,30)

```
[1] (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (0,10]  (10,20] (10,20] (10,20] (10,20] (10,20] (10,20] (10,20]
[18] (10,20] (10,20] (10,20] (20,30] (20,30] (20,30] (20,30] (20,30] (20,30] (20,30] (20,30] (20,30] (20,30]

Levels: (0,10] (10,20] (20,30]
 (0,10] (10,20] (20,30]
     10      10      10
```

Level in Factor

```
X= factor( c("single", "married", "married", "single") )
levels(x)          # [1] married single
nlevels(x)         # [1] 2
str(x)             # Factor w/ 2 levels "married","single": 2 1 1 2
x[3]               # access 3rd element
                     [1] married     Levels: married single
x[-1]              # access all but 1st element
                    [1] married married single
                    Levels: married single

x[c(TRUE, FALSE, FALSE, TRUE)]          # using logical vector
                                        [1] single single
                                         Levels: married single
x[2] = "divorced "          # doesn’t work
levels(x) = c(levels(x), “divorced")         # add new level
x[2] = "divorced“          # update the entry now
                             [1] single divorced married single
                             Levels: single married divorced
```

```
food <- factor(c("low", "high", "medium", "high", "low", "medium", "high"))
levels(food)             [1] "high" "low" "medium“

food <- factor(food, levels = c("low", "medium", "high"))
levels(food)             [1] "low" "medium" "high"
min(food)                # doesn't work

food <- factor( food,  levels = c("low", "medium", "high"), ordered = TRUE)
levels(food)         #[1] "low" "medium" "high"
min(food)          # [1] low Levels: low < medium < high
```

Matrix in R

- Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout.
- They contain elements of the **same atomic types.**
- we can create a matrix containing only characters or only logical values.
- We use matrices containing numeric elements to be used in mathematical calculations.
- A Matrix is created using the **matrix()** function.

Syntax

- The basic syntax for creating a matrix in R is −

  matrix(data, nrow, ncol, byrow, dimnames)

- Following is the description of the parameters used −
  - **data** is the input vector which becomes the data elements of the matrix.
  - **nrow** is the number of rows to be created.
  - **ncol** is the number of columns to be created.
  - **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row.
  - **dimname** is the names assigned to the rows and columns.

Matrix in R

- Matrix is a two dimensional data structure in R programming.
- Matrix is similar to vector but additionally contains the dimension attribute.
- All attributes of an object can be checked with the **attributes() function.**
- Dimension can be checked directly with the **dim() function**.
- We can check if a variable is a matrix or not with the **class() function**.

Example of different matrix dimension

2x2 matrix	column 1	column 2
row 1	1	2
row 2	3	4

3x3 matrix	column 1	column 2	Column 3
row 1	1	2	3
row 2	4	5	6
row 3	7	8	9

5x2 matrix	column 1	column 2
row 1	1	2
row 2	3	4
row 3	5	6
row 4	7	8
row 5	9	10

Example of Matrix

 # Elements are arranged sequentially by row.

```
M = matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)
```

 # Elements are arranged sequentially by column.

```
N = matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)
```

 # Define the column and row names.

```
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

P = matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```

```
class(P)          #[1] "matrix“

attributes(P)     $dim
                  [1] 3 3

dim(P)            #[1] 3 3
```

By Row

	[,1]	[,2]	[,3]
[1,]	3	4	5
[2,]	6	7	8
[3,]	9	10	11
[4,]	12	13	14

By Col

	[,1]	[,2]	[,3]
[1,]	3	7	11
[2,]	4	8	12
[3,]	5	9	13
[4,]	6	10	14

With DIMNAMES

	col1	col2	col3
row1	3	4	5
row2	6	7	8
row3	9	10	11
row4	12	13	14

Accessing Elements of a Matrix

- Elements of a matrix can be accessed by using the column and row index of the element.

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

# Create the matrix.
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))

# Access the element at 3rd column and 1st row.
print(P[1,3])                    # [1] 5

#Access the element at 2nd column and 4th row.
print(P[4,2])                    # [1] 13

# Access only the  2nd row.
print(P[ 2 , ])                  # col1   col2  col3
                                     6      7     8

# Access only the 3rd column.
print(P[  , 3 ])                 # row1  row2   row3   row4
                                     5     8     11     14
```

	col1	col2	col3
row1	3	4	5
row2	6	7	8
row3	9	10	11
row4	12	13	14

Matrix Computations

- Various mathematical operations are performed on the matrices using the R operators.
- The result of the operation is also a matrix.
- The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

```
# Create two 2x3 matrices.
matrix1 = matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

matrix2 = matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)

# Add the matrices.
result = matrix1 + matrix2
cat("Result of addition","\n")
print(result)

# Subtract the matrices
result = matrix1 - matrix2
cat("Result of subtraction","\n")
print(result)
```

```
Matrix-1
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6

Matrix-2
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4

Result of addition
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10

Result of subtraction
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2
```

Matrix Multiplication & Division

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)
matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)

# Multiply the matrices.
result <- matrix1 * matrix2
cat("Result of multiplication","\n")
print(result)

# Divide the matrices
result <- matrix1 / matrix2
cat("Result of division","\n")
print(result)
```

```
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6

     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4

Result of multiplication
     [,1] [,2] [,3]
[1,]   15    0    6
[2,]   18   36   24

Result of division
     [,1]      [,2]      [,3]
[1,]  0.6      -Inf 0.6666667
[2,]  4.5 0.4444444 1.5000000
```

Add a Row and Column to a Matrix

cbind() - 1. It is used for column binding.
2. It can concatenate as many matrix or columns as specified.

rbind() - 1. It is used for row binding.
2. It can concatenate as many matrix or row as specified.

```
# Create two 2x3 matrices.
A= matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(A)
B= matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(B)

C=cbind(A,B)
print(C)

D=rbind(A,B)
print(D)
```

Matrix -A

	[,1]	[,2]	[,3]
[1,]	3	-1	2
[2,]	9	4	6

Matrix-B

	[,1]	[,2]	[,3]
[1,]	5	0	3
[2,]	2	9	4

Matrix- C - CBIND(A , B)

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	3	-1	2	5	0	3
[2,]	9	4	6	2	9	4

Matrix-D - RBIND(A , B)

	[,1]	[,2]	[,3]
[1,]	3	-1	2
[2,]	9	4	6
[3,]	5	0	3
[4,]	2	9	4

Aggregate Function on Matrix

```
A= matrix(c(3, 9, -1, 4, 2, 6, 1,2,3), nrow=3, ncol = 3)
print(A)

r=sum(A)
print(r)        # 29

r=apply(A,1,sum)
print(r)        # 8   13   8

r=apply(A,2,sum)
print(r)        # 11  12   6

r=max(A)
print(r)        # 9

r=apply(A,2,max)
print(r)        # 9   6   3

r=mean(A)
print(r)        # 3.222222

r=apply(A,2,mean)
print(r)        # 3.666667    4.000000    2.000000

r=sd(A)
print(r)        #  2.905933

r=apply(A,2,sd)
print(r)        # 5.033223 2.000000 1.000000

r=min(A)
print(r)        #  -1

r=apply(A,1,min)
print(r)        #  1   2  -1

r=apply(A,2,min)
print(r)        #  -1  2   1

r=median(A)
print(r)        # 3

r=apply(A,1,median)
print(r)        # 3   2   3

r=apply(A,2,median)
print(r)        # 3   4   2
```

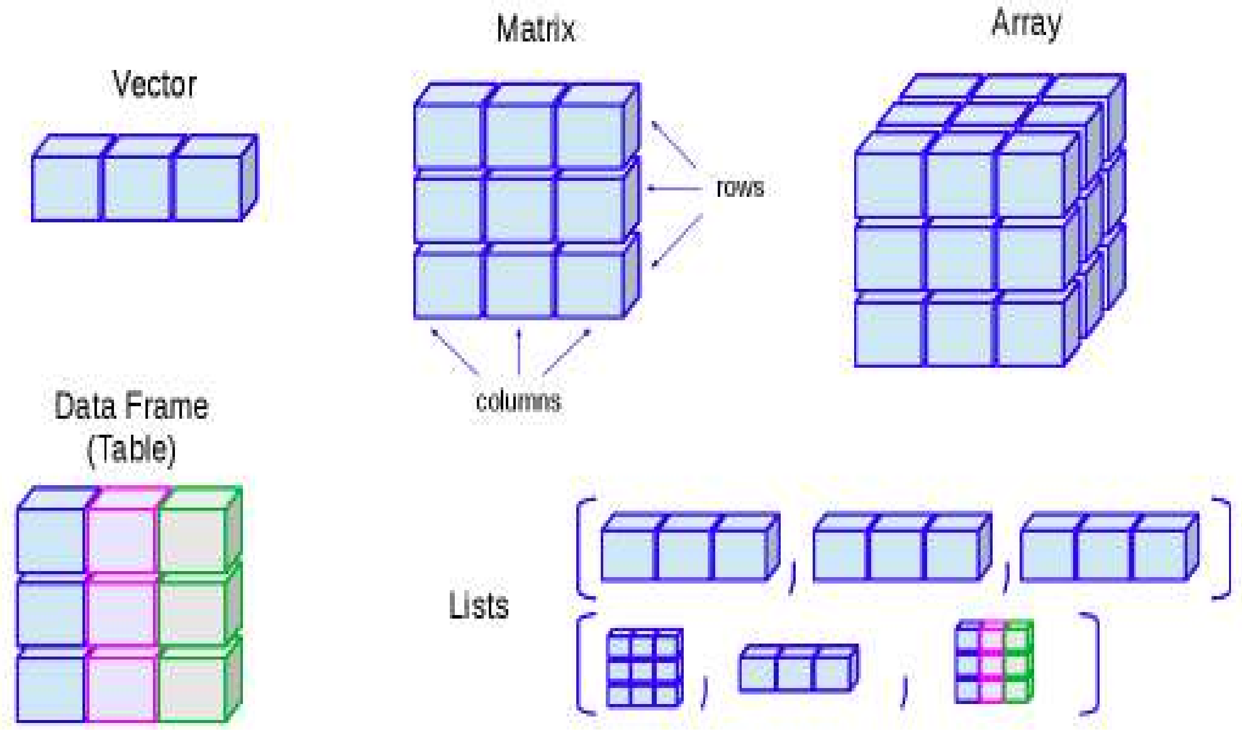
Matrix –A

Margin -1

		[,1]	**[,2]**	**[,3]**
	[1,]	**3**	**4**	**1**
2	**[2,]**	**9**	**2**	**2**
	[3,]	**-1**	**6**	**3**

Contents to be covered

- R Array
- R Table



R Array

- Arrays are essential data storage structures defined by a fixed number of dimensions.
- Arrays are used for the allocation of space at contiguous memory locations.
- **Uni-dimensional arrays** are called vectors with the length being their only dimension.
- **Two-dimensional arrays** are called matrices, consisting of fixed numbers of rows and columns.
- Arrays consist of all elements of the same data type.

Creating an Array

- An array in R can be created with the use of **array()** function.
- List of elements is passed to the array() functions along with the dimensions as required.

Syntax:

array(data, dim = (nrow, ncol, nmat), dimnames=names)

nrow *: Number of rows*

ncol *: Number of columns*

nmat *: Number of matrices of dimensions nrow * ncol*

dimnames *: Default value = NULL.*

Uni-Dimensional Array

- A vector is a uni-dimensional array, which is specified by a single dimension, length.
- A Vector can be created using ‘**c()**‘ function.
- A list of values is passed to the c() function to create a vector.

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
print (vec1)                # [1] 1 2 3 4 5 6 7 8 9
# cat is used to concatenate
# strings and print it.
cat ("Length of vector : ", length(vec1)     # Length of vector :  9
```

Multi-Dimensional Array

- A two-dimensional matrix is an array specified by a fixed number of rows and columns, each containing the same data type.
- A matrix is created by using **array()** function to which the values and the dimensions are passed.

```
# arranges data from 2 to 13  in two matrices of dimensions 2x3
arr = array(2:13, dim = c(2, 3, 2))
print(arr)
, , 1
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    3    5    7

, , 2
     [,1] [,2] [,3]
[1,]    8   10   12
[2,]    9   11   13
```

Naming of Arrays

- The row names, column names and matrices names are specified as a vector of the number of rows, number of columns and number of matrices respectively.
- By default, the rows, columns and matrices are named by their index values.

```
row_names <- c("row1", "row2")
col_names <- c("col1", "col2", "col3")
mat_names <- c("Mat1", "Mat2")

# the naming of the various elements is specified in a list and  fed to the function
arr = array(2:14, dim = c(2, 3, 2),
            dimnames = list(row_names,  col_names,  mat_names) )
print (arr)
```

```
,, Mat1
      col1 col2 col3
row1     2    4    6
row2     3    5    7

,, Mat2
      col1  col2  col3
row1     8    10    12
row2     9    11    13
```

Accessing Arrays

- The arrays can be accessed by using indices for different dimensions separated by commas.
- Different components can be specified by any combination of elements' names or positions.

Accessing Uni-Dimensional Array

- The elements can be accessed by using indexes of the corresponding elements.

```
# accessing elements
vec <- c(1:10)
# accessing entire vector
cat ("Vector is : ", vec)       # Vector is :  1 2 3 4 5 6 7 8 9 10
# accessing elements
cat ("Third element of vector is : ", vec[3])
                                # Third element of vector is : 3
```

Accessing entire matrices

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)
row_names <- c("row1", "row2")
col_names <- c("col1", "col2", "col3")
mat_names <- c("Mat1", "Mat2")
arr = array(c(vec1, vec2), dim = c(2, 3, 2),
  dimnames = list(row_names, col_names, mat_names))

# accessing matrix 1 by index value
print ("Matrix 1")
print (arr[  , , 1])

# accessing matrix 2 by its name

print ("Matrix 2")
print(arr[  ,   , "Mat2"])
```

```
[1] "Matrix 1"
      col1 col2 col3
row1    1    3    5
row2    2    4    6

[1] "Matrix 2"
      col1 col2 col3
row1    7    9   11
row2    8   10   12
```

Accessing Arrays

Accessing specific rows and columns of matrices

Rows and columns can also be accessed by both names as well as indices.

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)

row_names <- c("row1", "row2")
col_names <- c("col1", "col2", "col3")
mat_names <- c("Mat1", "Mat2")

arr = array(c(vec1, vec2), dim = c(2, 3, 2),  dimnames = list(row_names,  col_names, mat_names))
```

```
, , Mat1
     col1 col2 col3
row1   1    3    5
row2   2    4    6

, , Mat2
     col1 col2 col3
row1   7    9   11
row2   8   10   12
```

\# 1. **accessing matrix 1 by index value**

```
print ("1st column of matrix 1")
print (arr[ , 1, 1])
```

\# 2. **accessing matrix 2 by its name**

```
print ("2nd row of matrix 2")
print(arr["row2",,"Mat2"])
```

```
1. "1st column of matrix 1"
   row1  row2
     1     2
2. "2nd row of matrix 2"
   col1  col2  col3
     8    10    12
```

\# 3. **accessing matrix 1 by index value**

```
print ("2nd row 3rd column matrix 1 element")
print (arr[2, "col3", 1])
```

\# 4. **accessing matrix 2 by its name**

```
print ("2nd row 1st column element of matrix 2")
print(arr["row2", "col1", "Mat2"])
```

```
3. "2nd row 3rd column matrix 1 element"
   [1] 6

4. "2nd row 1st column element of matrix 2"
   [1] 8
```

\# 5. **print elements of both the rows and columns 2 and 3 of matrix 1**

```
print (arr[ , c(2, 3),  1])
```

```
5.
      col2  col3
row1    3     5
row2    4     6
```

Adding Elements to Array

- Elements can be appended at the different positions in the array. The sequence of elements is retained in order of their addition to the array. The time complexity required to add new elements is O(n) where n is the length of the array. The length of the array increases by the number of element additions. There are various in-built functions available in R to add new values:
    1. **c(vector, values):** c() function allows us to append values to the end of the array. Multiple values can also be added together.
    2. **append(vector, values):** This method allows the values to be appended at any position in the vector. By default, it adds the element at end.
    3. **append(vector, values, after=length(vector))** : adds new values after specified length of the array specified in the last argument of the function.

Using the length function of the array:
Elements can be added at length+x indices where x>0

```
# creating a uni-dimensional array
x <- c(1, 2, 3, 4, 5)

# addition of element using c() function
x <- c(x, 6)

print ("Array after 1st modification ")
print (x)

# addition of element using append function
x <- append(x, 7)

print ("Array after 2nd modification ")
print (x)
```

```
[1] "Array after 1st modification "
[1] 1 2 3 4 5 6

[1] "Array after 2nd modification "
[1] 1 2 3 4 5 6 7
```

Adding Elements to Array

```
# adding elements after computing the length
    len <- length(x)
    x[len + 1] <- 8
    print ("Array after 3rd modification ")
    print (x)

# adding on length + 3 index
    x[len + 3]<-9
    print ("Array after 4th modification ")
    print (x)

# append a vector of values to the array after
   length + 3 of array
   print ("Array after 5th modification")
   x <- append(x, c(10, 11, 12), after = length(x)+3)
   print (x)

# adds new elements after 3rd index
    print ("Array after 6th modification")
    x <- append(x, c(-1, -1), after = 3)
    print (x)
```

```
[1] "Array after 3rd modification "
[1] 1 2 3 4 5 6 7 8

[1] "Array after 4th modification "
[1]  1  2  3  4  5  6  7  8 NA  9

[1] "Array after 5th modification"
[1]  1  2  3  4  5  6  7  8 NA  9 10 11 12

[1] "Array after 6th modification"
[1]  1  2  3 -1 -1  4  5  6  7  8 NA  9 10 11 12
```

Removing Elements from Array

- Elements can be removed from arrays in R, either one at a time or multiple together.
- These elements are specified as indexes to the array, wherein the array values satisfying the conditions are retained and rest removed. The comparison for removal is based on array values. Multiple conditions can also be combined together to remove a range of elements.
- Another way to remove elements is by using **%in%** operator wherein the set of element values belonging to the TRUE values of the operator are displayed as result and the rest are removed.

```
# creating an array of length 9
  m <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
 print ("Original Array")
 print (m)

# remove a single value element:3 from array
m <- m[m != 3]
print ("After 1st modification")
print (m)

# removing elements based on condition where either element should be
# greater than 2 and less than equal to 8
m <- m[m>2 & m<= 8]

print ("After 2nd modification")
print (m)

])
```

```
# remove sequence of elements using another array
remove <- c(4, 6, 8)

# check which element satisfies the remove property
print (m % in % remove)
print ("After 3rd modification")
print (m [! m % in % remove
```

```
[1] "Original Array“
[1] 1 2 3 4 5 6 7 8 9
[1] "After 1st modification“
[1] 1 2 4 5 6 7 8 9
[1] "After 2nd modification“
[1] 4 5 6 7 8
[1]  TRUE FALSE  TRUE FALSE  TRUE
[1] "After 3rd modification“
[1] 5 7
```

Updating Existing Elements of Array

- The elements of the array can be updated with new values by assignment of the desired index of the array with the modified value.
- The changes are retained in the original array.
- If the index value to be updated is within the length of the array, then the value is changed, otherwise, the new element is added at the specified index.
- Multiple elements can also be updated at once, either with the same element value or multiple values in case the new values are specified as a vector.
- the FALSE value are printed, since the condition involves the NOT operator.

```
# creating an array of length 9
m <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
print ("Original Array")
print (m)

# updating single element
m[1] <- 0
print ("After 1st modification")
print (m)

# updating sequence of elements
m[7:9] = -1
print ("After 2nd modification")
print (m)
```

```
# updating two indices with two different values
m[c(2, 5)] <- c(-1, -2)
print ("After 3rd modification")
print (m)

# this add new element to the array
m[10] <- 10
print ("After 4th modification")
print (m)
```

```
[1] "Original Array"
[1] 1 2 3 4 5 6 7 8 9

[1] "After 1st modification"
[1] 0 2 3 4 5 6 7 8 9

[1] "After 2nd modification"
[1]  0  2  3  4  5  6 -1 -1 -1

[1] "After 3rd modification"
[1]  0 -1  3  4 -2  6 -1 -1 -1

[1] "After 4th modification"
[1]  0 -1  3  4 -2  6 -1 -1 -1 10
```

Calculation Across Array Elements

- We can do calculations across the elements in an array using the **apply()** function.

Syntax

apply(x, margin, fun)

description of the parameters used −

x is an array.
margin is the name of the data set used.
fun is the function to be applied across the elements of the array.

Example- sum of the elements in the rows of an array across all the matrices.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
new.array <- array( c(vector1,vector2), dim = c(3,3,2) )
print(new.array)

# Use apply to calculate the sum of the rows across all the matrices.
result <- apply(new.array,  c(1),  sum)
print(result)

# Use apply to calculate the sum of the columns across all the matrices.
result <- apply(new.array,  c(2),  sum)
print(result)
```

```
, , 1
     [,1]  [,2]  [,3]
[1,]   5    10    13
[2,]   9    11    14
[3,]   3    12    15

, , 2
     [,1]  [,2]  [,3]
[1,]   5    10    13
[2,]   9    11    14
[3,]   3    12    15

#Rows Sum
[1]   56  68  60

# Columns Sum
[1]   34 66 84
```

Table in R

- Another common way to store information is in a table- one way and two way tables.

One Way Tables

- One way to create a table is using the **table( )** command.
- The arguments it takes is a vector of factors, and it calculates the frequency that each factor occurs

```
a <- factor(c("A","A","B","A","B","B","C","A","C"))
results <- table(a)
```

```
results
  a
  A B C
  4 3 2
```

```
attributes(results)

$dim
[1] 3
 $dimnames
$dimnames$a
[1] "A" "B" "C"

$class
[1] "table"
```

```
summary(results)
Number of cases in table: 9
Number of factors: 1
```

Table in R

m<- **matrix**(**c**(4,3,2), ncol=3, byrow=**TRUE**)

```
> m
        [,1]  [,2 ]  [,3 ]
  [1,]    4     3      2
```

- At this point the variable “m” is a matrix with one row and three columns of numbers.

```
> colnames(m) <- c("A","B","C")
> m
      A   B   C
[1,]  4   3   2

> m <- as.table(m)
> m
   A  B  C
A  4  3  2
```

```
> attributes(m)
$dim
[1] 1 3

$dimnames
$dimnames[[1]]
[1] "A“
$dimnames[[2]]
[1] "A" "B" “C"

$class
[1] "table"
```

Table in R

Two Way Tables

- If you want to add rows to your table just add another vector to the argument of the table command.
- In the first , a - responses are labeled “Never,” “Sometimes,” or “Always.”
- In the second, b - responses are labeled “Yes,” “No,” or “Maybe.”
- The set of vectors “a” and “b” contain the response for each measurement.
- The third item in “a” is how the third person responded to the first question, and the third item in “b” is how the third person responded to the second question.

```
#Response of Question-1
>a=c("Sometimes","Sometimes","Never","Always","Always","Sometimes","Sometimes","Never")
#Response of Question-2
> b= c("Maybe","Maybe","Yes","Maybe","Maybe","No","Yes","No")
> results = table( a, b)
> results
           b
a          Maybe  No  Yes
Always       2     0    0
Never        0     1    1
Sometimes    2     1    1
```

Table in R

- The table command allows us to do a very quick calculation, and we can immediately see that two people who said “Maybe” to the first question also said “Sometimes” to the second question.
- Just as in the case with one-way tables it is possible to manually enter two way tables.
- The procedure is exactly the same as above except that we now have more than one row.
- We give a brief example below to demonstrate how to enter a two-way table that includes breakdown of a group of people by both their gender and whether or not they smoke.
- You enter all of the data as one long list but tell R to break it up into some number of columns:

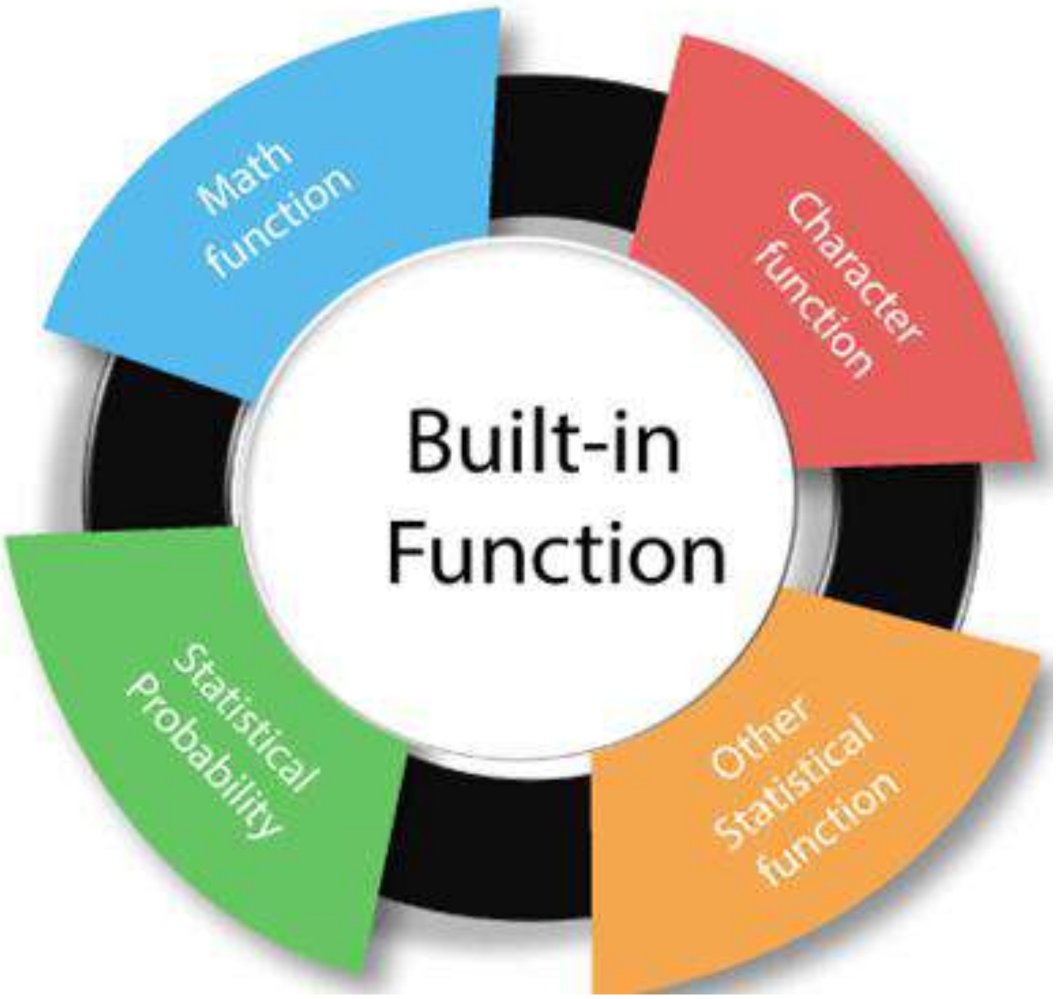
```
gendersmoke<-matrix(c(70,120,65,140),ncol=2,byrow=TRUE)
> rownames(gendersmoke)<-c("male","female")
> colnames(gendersmoke)<-c("smoke","nosmoke")
> sexsmoke <- as.table( gendersmoke )
> gendersmoke
        smoke   nosmoke
male     70      120
female   65      140
```

Contents to be covered

- R Built-in Function

R Built-in Functions

- The functions which are already created or defined in the programming framework are known as a built-in function.
- R has a rich set of functions that can be used to perform almost every task for the user.
- These built-in functions are divided into the following categories based on their functionality.
  - ➢ Math Function
  - ➢ String Function
  - ➢ Statistical Probability Function
  - ➢ Other Statistical Function



Math Functions

- R provides the various mathematical functions to perform the mathematical calculation.
- These mathematical functions are very helpful to find absolute value, square value and much more calculations.
- In R, there are the following functions which are used:

S. No	Function	Description	Example
1.	abs(x)	It returns the absolute value of input x.	x<- -4 print(abs(x)) **Output** [1] 4
2.	sqrt(x)	It returns the square root of input x.	x<- 4 print(sqrt(x)) **Output** [1] 2
3.	ceiling(x)	It returns the smallest integer which is larger than or equal to x.	x<- 4.5 print(ceiling(x)) **Output** [1] 5

Math Functions

S. No	Function	Description	Example
4.	floor(x)	It returns the largest integer, which is smaller than or equal to x.	x<- 2.5 print(floor(x)) 2
5.	trunc(x)	It returns the truncate value of input x.	x<- c(1.2,2.5,8.1) print(trunc(x)) **Output** [1] 1 2 8
6.	round(x, digits=n)	It returns round value of input x.	x<- -4 print(abs(x)) **Output** 4
7.	cos(x), sin(x), tan(x)	It returns cos(x), sin(x) value of input x.	x<- 4 print(cos(x)) print(sin(x)) print(tan(x)) **Output** [1] -06536436 [2] -0.7568025 [3] 1.157821

Math Functions

S. No	Function	Description	Example
8.	log(x)	It returns natural logarithm of input x.	x<- 4 print(log(x)) **Output** [1] 1.386294
9.	log10(x)	It returns common logarithm of input x.	x<- 4 print(log10(x)) **Output** [1] 0.60206
10.	exp(x)	It returns exponent.	x<- 4 print(exp(x)) **Output** [1] 54.59815

String Function

- R provides various string functions to perform tasks. These string functions allow us to extract sub string from string, search pattern etc.
- There are the following string functions in R:

S.No.	Function	Description	Example
1.	substr(x, start=n1,stop=n2)	It is used to extract substrings in a character vector.	a <- "987654321" substr(a, 3, 3) **Output** [1] “7"
2.	grep(pattern, x , ignore.case=FALSE, fixed=FALSE)	It searches for pattern in x.	st1 <- c('abcd','bdcd','abcdabcd') pattern<- '^abc' print(grep(pattern, st1)) **Output** [1] 1 3

String Function

S.No.	Function	Description	Example
3.	sub(pattern, replacement, x, ignore.case =FALSE, fixed=FALSE)	It finds pattern in x and replaces it with replacement (new) text.	st1<- "England is beautiful but England is not the part of EU" sub("England', "UK", st1) **Output** [1] "UK is beautiful but England is not a part of EU"
4.	gsub(pattern, replacement, x, ignore.case =FALSE, fixed=FALSE)	It finds pattern in x and replaces it with replacement (new) text.	st1<- "England is beautiful but England is not the part of EU" gsub("England', "UK", st1) **Output** [1] "UK is beautiful but UK is not a part of EU"
5.	paste(..., sep="")	It concatenates strings after using sep string to separate them.	paste('one', 2, 'three', 4, 'five') Output [1] one 2 three 4 five

String Function

S.No.	Function	Description	Example
6.	strsplit(x, split)	It splits the elements of character vector x at split point.	a<-"Split all the character" print(strsplit(a, "")) Output [[1]] [1] "split" "all" "the" "character"
7.	tolower(x)	It is used to convert the string into lower case.	st1<- “COMputer" print(tolower(st1)) Output [1] computer
8.	toupper(x)	It is used to convert the string into upper case.	st1<- “COMputer" print(toupper(st1)) Output [1] COMPUTER

Other Statistical Functions

- Apart from the functions mentioned above, there are some other useful functions which helps for statistical purpose. There are the following functions:

S. No	Function	Description	Example
1.	mean(x, trim=0, na.rm=FALSE)	It is used to find the mean for x object	**a<-c(0:10, 40) xm<-mean(a) print(xm) Output [1] 7.916667**
2.	sd(x)	It returns standard deviation of an object.	**a<-c(0:10, 40) xm<-sd(a) print(xm) Output [1] 10.58694**
3.	median(x)	It returns median.	**a<-c(0:10, 40) xm<-median(a) print(xm) Output [1] 5.5**

Other Statistical Functions

S. No	Function	Description	Example
4.	quantile(x, probs)	It returns quantile where x is the numeric vector whose quantiles are desired and probs is a numeric vector with probabilities in [0, 1]	**a<-c(0:10, 40)** **xm<-quantile(a)** **print(xm)** **Output** **0% 25% 50% 75% 100%** **0.00 2.75 5.50 8.25 40.00**
5.	range(x)	It returns range.	**a<-c(0:10, 40)** **xm<-range(a)** **print(xm)** **Output** **[1] 0 40**
6.	sum(x)	It returns sum.	**a<-c(0:10, 40)** **xm<-sum(a)** **print(xm)** **Output** **[1] 95**

Other Statistical Functions

S. No	Function	Description	Example
7.	diff(x, lag=1)	It returns differences with lag indicating which lag to use.	**a<-c(0:10, 40)** **xm<-diff(a)** **print(xm)** **Output** **[1] 1 1 1 1 1 1 1 1 1 1 30**
8.	min(x)	It returns minimum value.	**a<-c(0:10, 40)** **xm<-min(a)** **print(xm)** **Output** **[1] 0**
9.	max(x)	It returns maximum value	**a<-c(0:10, 40)** **xm<-max(a)** **print(xm)** **Output [1] 40**

Other Statistical Functions

S. No	Function	Description
10.	scale(x, center=TRUE, scale=TRUE)	Column center or standardize a matrix. **# Manually scaling** **(x - mean(x)) / sd(x)**

```
# Create matrix
mt <- matrix(1:10, ncol = 5)

# Print matrix
cat("Matrix:\n")
print(mt)

# Scale matrix with default arguments
cat("\nAfter scaling:\n")
scale(mt)
```

Matrix:

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

After scaling:

```
          [, 1]        [, 2]        [, 3]        [, 4]        [, 5]
[1, ]  -0.7071068  -0.7071068  -0.7071068  -0.7071068  -0.7071068
[2, ]   0.7071068   0.7071068   0.7071068   0.7071068   0.7071068
attr(, "scaled:center")
[1]  1.5   3.5   5.5   7.5   9.5
attr(, "scaled:scale")
[1]  0.7071068   0.7071068   0.7071068   0.7071068   0.7071068
```

```
# Create matrix
mt <- matrix(1:10, ncol = 2)

# Print matrix
cat("Matrix:\n")
print(mt)

# Scale center by vector of values
cat("\nScale center by vector of values:\n")
scale(mt, center = c(1, 2), scale = FALSE)

# Scale by vector of values
cat("\nScale by vector of values:\n")
scale(mt, center = FALSE, scale = c(1, 2))
```

Matrix:

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

Scale center by vector of values:

```
      [, 1]  [, 2]
[1, ]   0      4
[2, ]   1      5
[3, ]   2      6
[4, ]   3      7
[5, ]   4      8
attr(, "scaled:center")
[1]  1   2
```

Scale by vector of values:

```
      [, 1]  [, 2]
[1, ]    1    3.0
[2, ]    2    3.5
[3, ]    3    4.0
[4, ]    4    4.5
[5, ]    5    5.0
attr(, "scaled:scale")
[1]  1   2
```

Other Useful Functions

SNo	Function	Description
1	seq(from , to, by)	generate a sequence indices <- seq(1,10,2) #indices is c(1, 3, 5, 7, 9)
2	rep(x, ntimes)	repeat x to n times y <- rep(1:3, 2) # y is c(1, 2, 3, 1, 2, 3)
3	cut(x, n)	divide continuous variable in factor with n levels x=1:50 y <- cut(x, 5) **f3=cut(x , breaks=c(0,10,20,50))** **print(f3)** **print(summary(f3))**
4	which(x)	Index of TRUE in logical vector or matrix (index based on column major order) X=c(TRUE, FALSE, TRUE, FALSE, FALSE) print(which(x))
5	merge(df1, df2, by=key, [all=NULL]) all=TRUE all.x=TRUE all.y=TRUE	Merge the 2 dataframe or datatable on basis of key provided.

Contents to be covered

- R User Defined Function
- R Function Arguments Types
- R Variable Scope

R Function

Function Definition

- An R function is created by using the keyword **function**.
- The basic syntax of an R function definition is as follows –

```
function_name = function  (arg_1, arg_2, ...)  {

Function body

}
```

Components of Function

The different parts of a function are −

- **Function Name** − This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments** −
    1. An argument is a placeholder.
    2. When a function is invoked, you pass a value to the argument.
    3. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body** − The function body contains a collection of statements that defines what the function does.
- **Return Value** − The return value of a function is the last expression in the function body to be evaluated.

- R has many **in-built** functions which can be directly called in the program without defining them first.
- We can also create and use our own functions referred as **user defined** functions.

User Defined Function

- We can create user-defined functions in R.
- They are specific to what a user wants and once created they can be used like the built-in functions.
- **Example**

```
# Create a function to print squares of numbers in sequence.
new.function = function(a)  {
 for(i in 1:a) {
       b = i^2
       print(b)
   }
}
```

Calling a Function

```
new.function = function(a) {
 for(i in 1:a) {
       b = i^2
       print(b)
   }
}
```

 # Call the function new.function supplying 6 as an argument.

```
new.function(6)
```

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
>
```

Calling a Function without an Argument

 # Create a function without an argument.

```
new.function = function( )  {
  for(i in 1:5) {
    print(i^2)
  }
}
```

 # Call the function without supplying an argument.

```
new.function()
```

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
>
```

Calling a Function with Argument Value (by position and by name)

- The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

```
# Create a function with arguments.
new.function = function(a, b, c) {
  result = a * b + c
  print(result)
}
# Call the function by position of arguments.
new.function( 5, 3, 11)
# Call the function by names of the arguments.
new.function(a = 11, b = 5, c = 3)
```

```
[1] 26
[1] 58
>
```

Calling a Function with Default Argument

- We can define the value of the arguments in the function definition and call the function without supplying any argument to get the default result.
- But we can also call such functions by supplying new values of the argument and get non default result.

```
# Create a function with arguments.
new.function =function(a = 3, b = 6) {
  result <- a * b
  print(result)
}
# Call the function without giving any argument.
new.function()
# Call the function with giving new values of the argument.
new.function(9,5)
```

```
[1] 18
[1] 45
>
```

Lazy Evaluation of Function

- Arguments to functions are evaluated lazily, which means so they are evaluated only when needed by the function body.

 # Create a function with arguments.

```
new.function = function( a, b ) {
  print(a^2)
  print(a)
  print(b)
}
```

 # Evaluate the function without supplying one of the arguments.

```
new.function(6)
```

```
[1] 36
[1] 6
Error in print(b) : argument "b" is missing, with no default
>
```

Dots Argument

- Dots argument (…) is also known as **ellipsis** which allows the function to take an undefined number of arguments. It allows the function to take an arbitrary number of arguments.
- Example of a function with an arbitrary number of arguments.

```
# Function definition of dots operator
fun =function(n, ...){
  l = list(n, ...)
  paste(l, collapse = " ")
}

# Function call
fun(5, 1L,  6i, TRUE, “R Programming”, “Dots operator”)
```

Function as Argument of Function

- In R programming, functions can be passed to another functions as arguments.
- Example of implementation of function as an argument.

```
sum.odd=function(x)
{
  f=x%%2==1
  x=x[f]
  r=sum( x)
  return(r)
}

proc = function( x,FUN ) {
  r= FUN(x)
  print( r )
}
proc( c(5,6,7,8), sum.odd)                # 12

proc( c(2,3,4,5), mean)                   #   3.5
```

Scope of Variable in R

- In R, variables are the containers for storing data values. T
- hey are reference, or pointers, to an object in memory which means that whenever a variable is assigned to an instance, it gets mapped to that instance.
- A variable in R can store a vector, a group of vectors or a combination of many R objects.

```
# Assignment using equal operator
    var1 = c(0, 1, 2, 3)
    print(var1)

# Assignment using leftward operator
    var2 <- c("Python", "R")
    print(var2)

# A Vector Assignment
    a = c(1, 2, 3, 4)
    print(a)
    b = c("Debi", "Sandeep", "Subham", "Shiba")
    print(b)

# A group of vectors Assignment using list
    c = list(a, b)
    print(c)
```

```
[1] 0 1 2 3
[1] "Python" "R"
[1] 1 2 3 4
[1] "Debi"    "Sandeep" "Subham"  "Shiba"
[[1]]
[1] 1 2 3 4

[[2]]
[1] "Debi"    "Sandeep" "Subham"  "Shiba"
```

Scope of Variable in R

- The location where we can find a variable and also access it if required is called the scope of a variable.
- There are mainly two types of variable scopes:

Global Variables:

1. Global variables are those variables that exist throughout the execution of a program.
2. It can be changed and accessed from any part of the program.

Local Variables:

1. Local variables are those variables that exist only within a certain part of a program like a function and are released when the function call ends.

Global variable

```
# global variable
global = 5

# a function
f = function(){
```

Local variable

```
  # local variable with same
  # name as that of global variable
  global = 2

  print(global)
}
```

Global Variable

- Global Variables can be accessed from any part of the program.
- They are available throughout the lifetime of a program.
- They are declared anywhere in the program outside all of the functions or blocks.

Declaring global variables:

1. Global variables are usually declared outside of all of the functions and blocks.
2. They can be accessed from any portion of the program.

```
# usage of global variables
global = 5      # global variable

# global variable accessed from  within a function
display = function(){
  print(global)
}

display()            #  5
# changing value of global variable
global = 10
display()            #  10
```

In the above code, the variable 'global' is declared at the top of the program outside all of the functions so it is a global variable and can be accessed or updated from anywhere in the program.

Local Variable

- Variables defined within a function or block are said to be local to those functions.
- Local variables do not exist outside the block in which they are declared, i.e. they can not be accessed or used outside that block.

Declaring local variables: Local variables are declared inside a block.

```
# R program to illustrate usage of local variables
func = function(){
  # this variable is local to the function func() and cannot be accessed outside this function
  age = 18
}
print(age)        # Garbage Value
```

The above program displays a garbage value or an error saying "object 'age' not found".

The variable age was declared within the function "func()" so it is local to that function and not visible to the portion of the program outside this function.

Local Variable

- To correct the previous program error we have to display the value of variable age from the function “func()” only.

```
# R program to illustrate usage of local variables
func = function(){
  # this variable is local to the function func() and cannot be accessed outside this function
  age = 18
  print(age)
}
 cat("Age is:\n")
 func()          # 18
```

Accessing Global Variables

- Global Variables can be accessed from anywhere in the code unlike local variables that have a scope restricted to the block of code in which they are created.

```
f = function() {
  # a is a local variable here
  a <-1
}
f()
```

 # Can't access outside the function

```
print(a)        #Display garbage value
```

*In the above code, we see that we are unable to access variable "a" outside the function as it's assigned by an **assignment operator(<-)** that makes "a" as a local variable.*

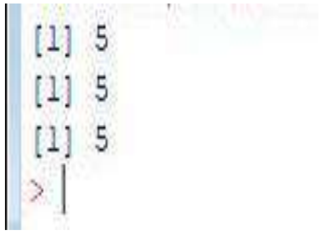
*To make assignments to global variables, a **super assignment operator(<<-)** is used.*

How super assignment Operator works?

- When using this operator within a function, it searches for the variable in the parent environment frame,
- if not found it keeps on searching the next level until it reaches the global environment.
- If the variable is still not found, it is created and assigned at the global level.

```
# R program to illustrate Scope of variables
outer_function = function(){

  inner_function = function(){

# Note that "<<-" operator here makes a as global variable
   a <<- 5
   print(a)
  }

  inner_function()

  print(a)
}

outer_function()
# Can access outside the function
```

- print(a)



*When the statement "a <<- 5" is encountered within **inner_function()**, it looks for the variable "a" in the **outer_function()** environment.*

*When the search fails, it searches in **R_GlobalEnv**.*

*Since "a" is not defined in this global environment as well, it is created and assigned there which is now referenced and printed from within **inner_function()** as well as **outer_function()**.*

Contents to be covered

- R Infix Operator
- R Switch Operator
- Within( ) function for dataframe
- apply( ), lapply ( ), sapply( ), tapply( )

R Infix Operator

- Infix operators in R are unique functions and methods that facilitate basic data expressions or transformations.
- Infix refers to the placement of the arithmetic operator between variables. For example
    - An infix operation is given by (a+b).
    - prefix are given by (+ab).
    - postfix operators are given by (ab+).
- The types of infix operators used in R include functions for data extraction, arithmetic, sequences, comparison, logical testing, variable assignments, and custom data functions.
- The rank of an operator indicates the order in which a function is processed by R.

List of Infix Operator in R

Operator	Rank	Type	Description	Comment
::	1	Extract	Function retrieval	**Extract function from a package namespace. my.package::mean extracts a new mean function from my.package**
:::	1	Extract	Function retrieval	**Extract a hidden function from a namespace**
$	2	Extract	List subset	**Extract list data by name. See name() function**
@	2	Extract	Slot selection	**Extract attributes by memory slot or location. See slotnames() function**
[and [[	3	Extract	Subscripting	**Extract data by index**
^	4	Arithmetic	Exponential	**2^3 = 8**
:	5	Sequence	Sequence	**1:3 = 1, 2, 3**
%/%	6	Arithmetic	Integer Divide	**5 %/% 2 = 2 See floor function**
%%	6	Arithmetic	Modulas	**5 %% 2 = 1**
%*%	6	Arithmetic	Matrix Multiplication	**Multiplies two matrices that are conformable**
%o%	6	Arithmetic	Outer Product	

List of Infix Operator in R

Operator	Rank	Type	Description	Comment
%x%	6	Arithmetic	Kronecker product	
*****	7	Arithmetic	Multiplication	**Also matrix dot product**
/	7	Arithmetic	Division	
+	8	Arithmetic	Addition	
-	8	Arithmetic	Subtraction	
!	8	Comparison	Not	
%in%	9	Comparison	Value Matching	**value1 %in% value2: true false false**
!=	9	Comparison	Not Equal To	**value1 != value2: true true true**
<	9	Comparison	Less Than	**value1< 5: false true true**
>	9	Comparison	Greater Than	**value2 > 4: false true true**
==	9	Comparison	Equal To	**value2 == 5: false true false**
<=	9	Comparison	Less Than Or Equal To	**value1 <= value2: false true true**
>=	9	Comparison	Greater Than Or Equal To	**value1 >= value2: true false**

List of Infix Operator in R

Operator	Rank	Type	Description	Comment
xor	10	Logical	Exclusive Or	**xor(value1, value2): false, false, false**
&	10	Logical	And (element)	**value1==4 & value2 >=6: false false true** **value1==4 & value2 >=6 : false false true**
&&	10	Logical	And (control)	**is.na(value1[2]) && value2[1]==5: false**
|	10	Logical	Or (element)	**value1==7 | value2>=7: true false true**
||	10	Logical	Or (control)	**is.na(value1[1]) || value2[1]==4: false**
~	11	Assignment	Equal	**Used in formulas and model building**
<<-	12	Assignment	Permanent Assignment	
<-	13	Assignment	Left Assignment	
->	13	Assignment	Right Assignment	
=	13	Assignment	Argument Assignment	

Infix Operator

```
>  5 + 3                    [1] 8
> `+`( 5 , 3)               [1] 8
> 5 - 3                     [1] 2
 > `-`( 5 , 3 )             [1] 2
> 5 * 3 - 1                 [1] 14
> `-`( `*` (5 , 3 ) , 1 )   [1] 14
```

User Defined Infix Operator

- It is possible to create user-defined infix operators in R.
- This is done by naming a function that at starts and ends with %.
- This function can be used as infix operator a %divisible% b or as a function call `%divisible%`(a, b). Both are the same.

```
>  10 %divisible% 3
[1] FALSE
> 10 %divisible% 2
[1] TRUE
> `%divisible%`(10,5)
[1] TRUE
```

Example of Infix Opeartor

```
'%divisible%' = function(a , b)
{
   r=a%%b
   if (r==0)
   { return (TRUE)
   } else
  { return(FALSE)
   }
}
```

Things to remember while defining your own infix operators are that they must start and end with %.

Surround it with back tick (`) in the function definition and escape any special symbols.

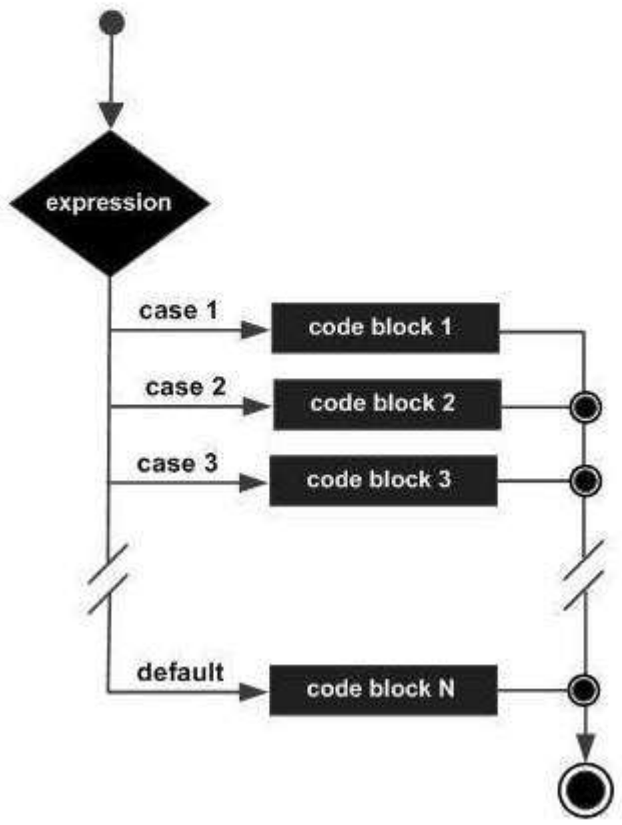
The switch( ) Function

- The **switch()** function in R tests an expression against elements of a list.
- If the value evaluated from the expression matches item from the list, the corresponding value is returned

Syntax of switch() function

switch (expression , list)

- The *expression* is evaluated and based on this value, the corresponding item in the list is returned.
- If the value evaluated from the *expression* matches with more than one item of the list, switch() function returns the first matched item.



Example: Switch() Function

- If the value evaluated is a number, that item of the list is returned.

```
> switch(2,"red", "green", "blue")
[1] "green"
> switch(1,"red","green","blue")
[1] "red"
```

- The switch() function returns the corresponding item to the numeric value evaluated.
- If the numeric value is out of range (greater than the number of items in the list or smaller than 1, then, NULL is returned.

```
> x <- switch(4,"red","green","blue")
> x
NULL

> x <- switch(0,"red","green","blue")
> x
NULL
```

Example: switch() function with as String Expression

The ***expression* used in the switch () function can be a string as well**. In this case, the matching named item’s value is returned.

```
switch( "color",  "color" = "red", "shape" = "square",  "length" = 5 )
[1] "red"
```

- Here, "color" is a string which matches with the first item of the list. Hence, we are getting “red” as an output.

```
> switch("length",  "color" = "red", "shape" = "square", "length" = 5)
[1] 5
```

Similarly, "length" expression matches with the last item of the list. Hence, we are getting 5

Example -1 of switch( )

```
val = switch(
  4,
  "Chandigarh",
  "Patna",
  "Kolkata",
  "Gorakhpur",
  "Lucknow",
  "Delhi"
)
print(val)        # Gorakhpur
```

Example -2 of switch( )

```
val1 = 6
val2 = 7
val3 = "s"
result = switch(
 val3,
 "a"= paste("Addition =", val1 + val2),
 "d"= paste("Subtraction =", val1 - val2),
 "r"= paste("Division = ", val1 / val2),
 "s"= paste("Multiplication =", val1 * val2),
 "m"= paste("Modulus =", val1 %% val2),
 "p"= paste("Power =", val1 ^ val2),
 paste('Invalid Operator' )
)
print(result)
```

Example -3 of switch( )

```
val1 = 6
val2 = 7
val3 = readline("Enter Operator : ")
result = switch(
  val3,
  "a"= paste("Addition =", val1 + val2),
  "d"= paste("Subtraction =", val1 - val2),
  "r"= paste("Division = ", val1 / val2),
  "s"= paste("Multiplication =", val1 * val2),
  "m"= paste("Modulus =", val1 %% val2),
  "p"= paste("Power =", val1 ^ val2),
  paste('Invalid Operator' )
)
print(result)
```

```
Enter Operator : s
[1] "Multiplication = 42"

Enter Operator : s1
[1] "Invalid Operator"
```

within () – To create a Column in DataFrame

```
df=read.csv('d:/datacsv/basket.csv')
print(df)

basket=within(df,
          { IS_DAIRY='NO'
           IS_DAIRY[Item_Name %in% c('Milk','Curd','Cheese','Paneer')]='YES'
           IS_DAIRY[!Item_Name %in% c('Milk','Curd','Cheese','Paneer')]='NO'
           }
          )
print(basket)
```

	Item_Group	Item_Name	Price	Tax
1	Fruit	Apple	100	2
2	Fruit	Banana	80	4
3	Fruit	Orange	80	5
4	Fruit	Mango	90	NA
5	Fruit	Papaya	65	2
6	Vegetable	Carrot	70	3
7	Vegetable	Potato	60	NA
8	Vegetable	Brinjal	70	1
9	Vegetable	Raddish	25	NA
10	Dairy	Milk	60	4
11	Dairy	Curd	40	5
12	Dairy	Cheese	35	NA
13	Dairy	Milk	50	4
14	Dairy	Paneer	60	NA

```
basket=within(df,
          { IS_DAIRY='NO'

          IS_DAIRY[Item_Group %in% c('Dairy')]='YES'
          IS_DAIRY[Item_Group %in% c('Fruit','Vegetable')]='NO'
          }
)

print(basket)
```

	Item_Group	Item_Name	Price	Tax	IS_DAIRY
1	Fruit	Apple	100	2	NO
2	Fruit	Banana	80	4	NO
3	Fruit	Orange	80	5	NO
4	Fruit	Mango	90	0	NO
5	Fruit	Papaya	65	2	NO
6	Vegetable	Carrot	70	3	NO
7	Vegetable	Potato	60	0	NO
8	Vegetable	Brinjal	70	1	NO
9	Vegetable	Raddish	25	0	NO
10	Dairy	Milk	60	4	YES
11	Dairy	Curd	40	5	YES
12	Dairy	Cheese	35	0	YES
13	Dairy	Milk	50	4	YES
14	Dairy	Paneer	60	0	YES

within () – To create a Column in DataFrame

```
df=read.csv('d:/datacsv/basket.csv')
print(df)

Newbasket=within(df,
        { NewRate=10
          NewRate[Tax %in% c(5)]=20
          NewRate[Tax %in% c(NA,1)]=0
          NewRate[Tax %in% c(2,3,4)]=15
        }
)
print(Newbasket)
```

	Item_Group	Item_Name	Price	Tax
1	Fruit	Apple	100	2
2	Fruit	Banana	80	4
3	Fruit	Orange	80	5
4	Fruit	Mango	90	NA
5	Fruit	Papaya	65	2
6	Vegetable	Carrot	70	3
7	Vegetable	Potato	60	NA
8	Vegetable	Brinjal	70	1
9	Vegetable	Raddish	25	NA
10	Dairy	Milk	60	4
11	Dairy	Curd	40	5
12	Dairy	Cheese	35	NA
13	Dairy	Milk	50	4
14	Dairy	Paneer	60	NA

	Item_Group	Item_Name	Price	Tax	NewRate
1	Fruit	Apple	100	2	15
2	Fruit	Banana	80	4	15
3	Fruit	Orange	80	5	20
4	Fruit	Mango	90	NA	0
5	Fruit	Papaya	65	2	15
6	Vegetable	Carrot	70	3	15
7	Vegetable	Potato	60	NA	0
8	Vegetable	Brinjal	70	1	0
9	Vegetable	Raddish	25	NA	0
10	Dairy	Milk	60	4	15
11	Dairy	Curd	40	5	20
12	Dairy	Cheese	35	NA	0
13	Dairy	Milk	50	4	15
14	Dairy	Paneer	60	NA	0

apply() function

- It takes Data frame or matrix as an input and gives output in vector, list or array.
- It is primarily used to avoid explicit uses of loop constructs.

apply(X, MARGIN, FUN)

-x: an array or matrix
-MARGIN: to define where to apply the function:
-MARGIN=1`: the manipulation is performed on rows
-MARGIN=2`: the manipulation is performed on columns
-FUN: tells which function to apply. Built functions like mean, median, sum, min, max and even user-defined functions can be applied>

```
df=read.csv('d:/datacsv/d13salesdata-1-6.csv')
print(df)
```

```
r=apply( df[,2:ncol(df)], 1,sum )
names(r)=df[1:nrow(df),1]
print(r)
```

```
r=apply( df[,2:ncol(df)], 2,sum )
print(r)
```

```
new.df=within(df,
          { Total=apply( df[,2:ncol(df)], 1,sum )
            TotalGrade=ifelse(Total>200,'A','B')
            }
          )
print(new.df)
```

```
    Name Jan Feb Mar Apr May Jun
1   Ajay  10  21  23  31   7  22  114
2  Vijay  13  17  12  29  14  16  101
3 Sanjay  17  15  16  13  18  10   89
4   Ajit  45  21   7  34  22  34  163
5  Vikas  22  56  76  34  22  16  226
6  Vipul  12  17  22  36  31  23  141
7 Rakesh  31  86  27  41  32  22  239

         150 233 183 218 146 143
```

```
  Ajay  Vijay Sanjay   Ajit  Vikas  Vipul Rakesh
   114    101     89    163    226    141    239
```

```
Jan Feb Mar Apr May Jun
150 233 183 218 146 143
```

```
    Name Jan Feb Mar Apr May Jun TotalGrade Total
1   Ajay  10  21  23  31   7  22          B   114
2  Vijay  13  17  12  29  14  16          B   101
3 Sanjay  17  15  16  13  18  10          B    89
4   Ajit  45  21   7  34  22  34          B   163
5  Vikas  22  56  76  34  22  16          A   226
6  Vipul  12  17  22  36  31  23          B   141
7 Rakesh  31  86  27  41  32  22          A   239
```

lapply() function

- It takes list, vector or data frame as input and gives output in list.
- It returns a list of the similar length as input list object.

lapply(X, FUN)
-X: A vector or an object
-FUN: Function applied to each element of x

	Name	Jan	Feb	Mar	Apr	May	Jun	
1	Ajay	**10**	**21**	**23**	**31**	**7**	**22**	**114**
2	Vijay	**13**	17	12	29	14	16	101
3	Sanjay	**17**	15	16	13	18	10	89
4	Ajit	**45**	21	7	34	22	34	163
5	Vikas	**22**	56	76	34	22	16	226
6	Vipul	**12**	17	22	36	31	23	141
7	Rakesh	**31**	86	27	41	32	22	239
		150	**233**	**183**	**218**	**146**	**143**	

```
df=read.csv('d:/datacsv/d13salesdata-1-6.csv')
print(df)

r=lapply( df[,2:ncol(df)], sum )
print(r)
```

```
$Jan
[1] 150

$Feb
[1] 233

$Mar
[1] 183

$Apr
[1] 218

$May
[1] 146

$Jun
[1] 143
```

```
print(unlist(r))
```

```
Jan  Feb  Mar  Apr  May  Jun
150  233  183  218  146  143
```

sapply() function

- It takes list, vector or data frame as input and gives output in vector.
- It returns a list of the similar length as input list object.

sapply(X, FUN)
-X: A vector or an object
-FUN: Function applied to each element of x

	Name	Jan	Feb	Mar	Apr	May	Jun	
1	Ajay	**10**	**21**	**23**	**31**	**7**	**22**	**114**
2	Vijay	**13**	17	12	29	14	16	101
3	Sanjay	**17**	15	16	13	18	10	89
4	Ajit	**45**	21	7	34	22	34	163
5	Vikas	**22**	56	76	34	22	16	226
6	Vipul	**12**	17	22	36	31	23	141
7	Rakesh	**31**	86	27	41	32	22	239
		150	**233**	**183**	**218**	**146**	**143**	

```
df=read.csv('d:/datacsv/d13salesdata-1-6.csv')
print(df)

r=sapply( df[,2:ncol(df)], sum )
print(r)
```

```
Jan  Feb  Mar  Apr  May  Jun
150  233  183  218  146  143
```

tapply() function

- It computes a measure (mean, median, min, max, etc..) or a function for each factor variable in a vector.
- It is a very useful function that lets you create a subset of a vector and then apply some functions to each of the subset.

tapply(X, INDEX, FUN = NULL)
Arguments:
-X: An object, usually a vector
-INDEX: A list containing factor
-FUN: Function applied to each element of x

```
df=read.csv('d:/datacsv/basket.csv')
print(df)

r=tapply(df$Price, df$Item_Group, sum)
print(r)

r=tapply(df$Price, df$Tax, sum)
print(r)
```

```
   Item_Group Item_Name Price Tax
1       Fruit     Apple   100   2
2       Fruit    Banana    80   4
3       Fruit    Orange    80   5
4       Fruit     Mango    90  NA
5       Fruit    Papaya    65   2
6   Vegetable    Carrot    70   3
7   Vegetable    Potato    60  NA
8   Vegetable   Brinjal    70   1
9   Vegetable   Raddish    25  NA
10      Dairy      Milk    60   4
11      Dairy      Curd    40   5
12      Dairy    Cheese    35  NA
13      Dairy      Milk    50   4
14      Dairy    Paneer    60  NA
```

#Sum Group Wise

Dairy	Fruit	Vegetable
245	415	225

#Sum Tax Wise

1	2	3	4	5
70	165	70	190	120

Contents to be covered

- R Object
- R Class

R Object Oriented Programming Concepts

- In R programming, OOPs provides classes and objects as its key tools to reduce and manage the complexity of the program.
- R is a functional language that uses concepts of OOPs.
- CLASS- Class like a sketch of a car , that contains all the details about the model_name, model_no, engine etc.
- OBJECT - Based on these descriptions we select a car. Car is the object. Each car object has its own characteristics and features.
- An object is also called an instance of a class and the process of creating this object is called instantiation.
- OOPs has following features:
  - Class
  - Object
  - Abstraction
  - Encapsulation
  - Polymorphism
  - Inheritance

S3 Class

- S3 class is somewhat primitive in nature. It lacks a formal definition and object of this class can be created simply by adding a class attribute to it.
- This simplicity accounts for the fact that it is widely used in R programming language. In fact most of the R built-in classes are of this type.

Example

```
> # create a list with required components
> s <- list(name = "John", age = 21, GPA = 3.5)
> # name the class appropriately
> class(s) <- "student“
```

Above example creates a S3 class with the given list.

S4 Class

- S4 class are an improvement over the S3 class. They have a formally defined structure which helps in making object of the same class look more or less similar.
- Class components are properly defined using the setClass() function and objects are created using the new() function.

Example

```
setClass("student", slots=list(name="character", age="numeric", GPA="numeric"))
```

 # Function setClass() command used to create S4 class containing list of slots.

```
setClass("Student", slots=list(name="character", Roll_No="numeric"))
```

 # 'new' keyword used to create object of class 'Student'

```
a <- new("Student", name="Adam", Roll_No=20)
```

 # Calling object

```
a
```

```
Slot "name":
[1] "Adam"

Slot "Roll_No":
[1] 20
```

Example Definition of S4 class

setClass("student", slots=list(name="character", age="numeric", GPA="numeric"))

- In the above example, we defined a new class called student along with three slots it's going to have name, age and GPA.

Example : Creation of S4 object

```
> # create an object using new()
> # provide the class name and value for slots
> s <- new("student", name="John", age=21, GPA=3.5)
> s
```

An object of class "student“

Slot "name":
[1] "John“

Slot "age":
[1] 21

Slot "GPA":
[1] 3.5

We can check if an object is an S4 object through the function isS4().

```
isS4(s)     # TRUE
```

How to create S4 objects?

- The function **setClass()** returns a generator function.
- This generator function (usually having same name as the class) can be used to create new objects. It acts as a constructor.

  student <-setClass("student", slots=list(name="character", age="numeric", GPA="numeric"))

- Now we can use this constructor function to create new objects.

```
student(name="John", age=21, GPA=3.5)

An object of class "student“
Slot "name":
[1] "John“
Slot "age":
[1] 21
Slot "GPA":
[1] 3.5
```

How to access and modify slot?

- Just as components of a list are accessed using $, slot of an object are accessed using @.

Accessing slot

> s@name
[1] "John“

> s@GPA
[1] 3.5

> s@age
[1] 21

Modifying slot directly

A slot can be modified through reassignment.

> # modify GPA
> s@GPA <- 3.7

> An object of class "student“ - s

Slot "name":
[1] "John“

Slot "age":
[1] 21

Slot "GPA":
[1] 3.7

Modifying slots using slot() function

Similarly, slots can be access or modified using the **slot()** function.

> slot(s,"name")
[1] "John“

> slot(s,"name") <- "Paul“
> s
An object of class "student“

Slot "name":
[1] "Paul“

Slot "age":
[1] 21

Slot "GPA":
[1] 3.7

How to write your own method?

- We can write our own method using **setMethod()** helper function.
- For example, we can implement our class method for the **show()** generic as follows.

```
setMethod("show", "student",
          function(object) {
            cat(object@name, "\n")
            cat(object@age, "years old\n")
            cat("GPA:", object@GPA, "\n")
          } )
```

- Now, if we write out the name of the object in interactive mode as before, the above code is executed.

```
s <- new("student", name="John", age=21, GPA=3.5)
> s
# this is same as show(s)
John
21 years old
GPA: 3.5
```

-

showMethods()- List all the S4 generic functions and methods available.

showMethods(show)

```
Function: show (package methods)
object="ANY“
object="classGeneratorFunction“
...
object="standardGeneric“
(inherited from: object="genericFunction")
object="traceable"
```

Check if a function is a generic function

```
isS4(print)
[1] FALSE
> isS4(show)
[1] TRUE
```

Example of Class Implementation

```
setClass("myNumber",
        slots=list(n="numeric")
)

setMethod("show",  "myNumber",
         function(object) {
           cat('Value : ',object@n, "\n")
         }
       )

x=new('myNumber', n=5)

show(x)

setGeneric('FACT', function(object)  standardGeneric('FACT'))

setGeneric('SumN', function(object)  standardGeneric('SumN'))

setGeneric('ODD', function(object)  standardGeneric('ODD'))

setGeneric('EVEN', function(object)  standardGeneric('EVEN'))


setMethod("FACT",  "myNumber",
         function(object) {
           f=1
           for (i in 1:object@n)
             f=f*i
           cat('Fact of ', object@n , ' = ', f, "\n")
         }
)
```

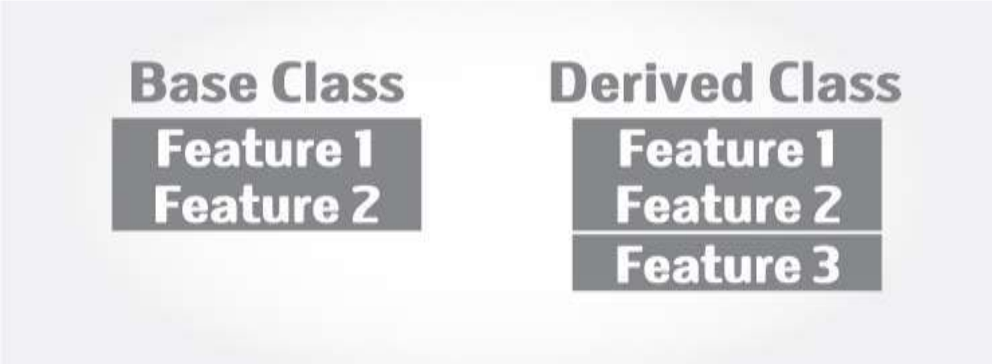
```
setMethod("SumN",  "myNumber",
         function(object) {
           f=0
           for (i in 1:object@n)
             f=f+i
           cat('Sum of ', object@n , ' = ', f, "\n")
         }
)

setMethod("ODD",  "myNumber",
         function(object) {
           f=1
           for (i in seq(1,object@n, by=2))
             cat(i, ' ' )
           cat('\n')
         }
)

setMethod("EVEN",  "myNumber",
                  function(object) {
                    f=1
                    for (i in seq(2,object@n, by=2))
                        cat(i, ' ' )
                        cat('\n')
                  }
                   )
```

Output

```
 show(x)
Value :  5

FACT(x)
Fact of  5  =  120

 ODD(x)
1  3  5

 EVEN(x)
2  4

 SumN(x)
Sum of  5  =  15
```

R Inheritance

- Inheritance is one of the key features of object-oriented programming which allows us to define a new class out of existing classes.
- This is to say, we can derive new classes from existing base classes and adding new features. We don't have to write from scratch.
- Inheritance provides reusability of code.
- Inheritance forms a hierarchy of class just like a family tree.
- The attributes define for a base class will automatically be present in the derived class.
- Moreover, the methods for the base class will work for the derived.



R Inheritance in S4 Class

- Derived classes will inherit both attributes and methods of the parent class.

define a class called student

```
setClass("student",
slots=list(name="character", age="numeric", GPA="numeric")
)
```

define class method for the show() generic function

```
setMethod("show",  "student",
function(object) {
	cat(object@name, "\n")
	cat(object@age, "years old\n")
	cat("GPA:", object@GPA, "\n")
}
)

s = new("student", name="John", age=21, GPA=3.5, country="France")
```

show(s)
John
21 years old
GPA: 3.5

- Inheritance is done during the derived class definition with the argument contains as shown below.

inherit from student

```
setClass( "InternationalStudent",
slots=list(country="character"),
contains="student"
 )
```

Here we have added an attribute country, rest will be inherited from the parent.

```
s = new("InternationalStudent", name="John", age=21, GPA=3.5,
  country="France")
```

show(s)
John
21 years old
GPA: 3.5

R Inheritance in S4 Class

 # define class method for the show() generic function for Inherited Class

```
setMethod("show", "InternationalStudent",
    function(object) {
        cat(object@name, "\n")
        cat(object@age, "years old\n")
        cat("GPA:", object@GPA, "\n")
        cat("Country:", object@country, "\n")
          }
)


show(s)

John
21 years old
GPA: 3.5
Country : France
```

getMethod( ) – Help to show the content of methods

```
getMethod(show," InternationalStudent ")
Method Definition:

function (object)
{
      cat(object@name, "\n")
       cat(object@age, "years old\n")
       cat("GPA:", object@GPA, "\n")
       cat("Country:", object@country, "\n")
}

Signatures:
      object
target  " InternationalStudent "
defined " InternationalStudent "
```

Creating new Method with Generic

```
# define class method for the GPAGrade() generic function

setGeneric('GPAGrade', function(object) standardGeneric('GPAGrade '))

setMethod(' GPAGrade'," InternationalStudent ",
    function(object) {
        cat(object@name, "\n")
        cat(object@age, "years old\n")
        cat("GPA:", object@GPA, "\n")
        cat("Country:", object@country, "\n")

        if (object@GPA>5)
       {
          cat("GPA is Good", "\n")
        } else
        {
            cat("GPA is not Good", "\n")
         }
                        }
)
```

```
GPAGrade(s)

John
21 years old
GPA: 3.5
Country : France
GPA is not Good
```

Example of Class and Inheritance Implementation

```
setClass("myNumber",
        slots=list(n="numeric")
)

setMethod("show",  "myNumber",
         function(object) {
           cat('Value : ',object@n, "\n")
         }
       )

x=new('myNumber', n=5)
show(x)

setGeneric('FACT', function(object)  standardGeneric('FACT'))

setGeneric('SumN', function(object) standardGeneric('SumN'))

setGeneric('ODD', function(object)  standardGeneric('ODD'))

setGeneric('EVEN', function(object)  standardGeneric('EVEN'))

setMethod("FACT",  "myNumber",
         function(object) {
           f=1
           for (i in 1:object@n)
             f=f*i
           cat('Fact of ', object@n , ' = ', f, "\n")
         }
)
```

```
setMethod("SumN",  "myNumber",
         function(object) {
           f=0
           for (i in 1:object@n)
             f=f+i
           cat('Sum of ', object@n , ' = ', f, "\n")
         }
)

setMethod("ODD",  "myNumber",
         function(object) {
           f=1
           for (i in seq(1,object@n, by=2))
             cat(i, ' ' )
           cat('\n')
         }
)

setMethod("EVEN",  "myNumber",
               function(object) {
                 f=1
                 for (i in seq(2,object@n, by=2))
                      cat(i, ' ' )
                      cat('\n')
               }
                 )
```

```
setClass( "NewmyNumber",
        slots=list(n2="numeric"),
        contains="myNumber" )

setMethod("show",  "NewmyNumber",
        function(object) {    cat('Value-1 : ',object@n, "\n")
                              cat('Value-2 : ',object@n2, "\n")  } )

y=new('NewmyNumber', n=5, n2=10)
show(y)

setGeneric('SUM', function(object) standardGeneric('SUM'))
setGeneric('PRODUCT', function(object)
standardGeneric('PRODUCT'))

setMethod("SUM",  "NewmyNumber",
        function(object) { r=object@n + object@n2
                           cat('Sum of 2 No : ', r, '\n' )   }  )

setMethod("PRODUCT",  "NewmyNumber",
        function(object) {   r=object@n * object@n2
                             cat('Product of 2 No : ', r, '\n' ) } )

SUM(y)
PRODUCT(y)
FACT(y)
ODD(y)
EVEN(y)
SumN(y)
```

Output

```
show(y)
Value-1 :  5
Value-2 :  10

 SUM(y)
Sum of 2 No :  15

 PRODUCT(y)
Product of 2 No :  50

 FACT(y)
Fact of  5  =  120

 ODD(y)
1  3  5


 EVEN(y)
2  4

SumN(y)
Sum of  5  =  15
```

Contents to be covered

- mtcars
- Iris
- Titanic

mtcars - DataSet

mtcars data comes from the 1974 Motor Trend magazine.

The data includes fuel consumption data, and ten aspects of car design for then-current car models.

#Loading Data

data(mtcars)

str(mtcars)

'data.frame': 32 obs. of 11 variables:

```
$ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
$ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
$ disp: num  160 160 108 258 360 ...
$ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
$ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
$ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
$ qsec: num  16.5 17 18.6 19.4 17 ...
$ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
$ am  : num  1 1 1 0 0 0 0 0 0 0 ...
$ gear: num  4 4 4 3 3 3 3 4 4 4 ...
$ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Description of variables:

mpg:	Miles/(US) gallon
cyl:	Number of cylinders **(4 , 6, or 8)**
disp:	Displacement (cu.in.)
hp:	Gross horsepower
drat:	Rear axle ratio
wt:	Weight (1000 lbs)
qsec:	1/4 mile time
vs:	Engine **(0 = V-shaped, 1 = straight**)
am:	Transmission (0 = automatic, 1 = manual)
gear:	Number of forward gears **(3, 4, or 5)**
carb:	Number of carburetors **(1, 2, 3, 4, 6, or 8)**

 # Number of rows (observations)

nrow(mtcars) #32

 # Number of columns (variables)

ncol(mtcars) #11

#dimension of dataset

dim(mtcars)

```
names(mtcars)
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am"
"gear"  "carb”
```

राइसूप्रौसं NIELIT

R Programming

mtcars - DataSet

head(mtcars)

```
                  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

#quantiles of dataset

```
quantile(mtcars$wt)
     0%     25%     50%     75%    100%
1.51300 2.58125 3.32500 3.61000 5.42400

quantile(mtcars$wt, c(.2, .4, .8))
  20%   40%   80%
2.349 3.158 3.770
```

#variance of weight

```
var(mtcars$wt)      # 0.957379
```

summary(mtcars)

```
      mpg             cyl             disp             hp             drat             wt             qsec
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760   Min.   :1.513   Min.   :14.50
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89
 Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695   Median :3.325   Median :17.71
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597   Mean   :3.217   Mean   :17.85
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930   Max.   :5.424   Max.   :22.90
       vs               am              gear            carb
 Min.   :0.0000   Min.   :0.0000   Min.   :3.000   Min.   :1.000
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
 Median :0.0000   Median :0.0000   Median :4.000   Median :2.000
 Mean   :0.4375   Mean   :0.4062   Mean   :3.688   Mean   :2.812
 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
 Max.   :1.0000   Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

#mode is not a built-in R function, we calculate it for each.

```
mode_mpg = names(sort(-table(mtcars$mpg)))[1]
mode_cyl = names(sort(-table(mtcars$cyl)))[1]

paste("The mode of the miles per gallon data is", mode_mpg)
paste("The mode of the number of cylinders data is", mode_cyl)
```

mtcars

#get table of group by gear

table(mtcars$gear)

3	4	5
15	12	5

 # get the count of missing values in the data set

sum(is.na(mtcars))

[1] 0

 # cross classification counts for cylinders by carburetors

table(mtcars$cyl, mtcars$carb)

##		**1**	**2**	**3**	**4**	**6**	**8**
##	**4**	5	6	0	0	0	0
##	**6**	2	0	0	4	1	0
##	**8**	0	4	3	6	0	1

tapply(mtcars$mpg, list(mtcars$cyl, mtcars$gear), mean)

	3	4	5
4	21.50	26.925	28.2
6	19.75	19.750	19.7
8	15.05	NA	15.4

tapply(mtcars$mpg, list(mtcars$cyl, mtcars$gear), mean, default=0)

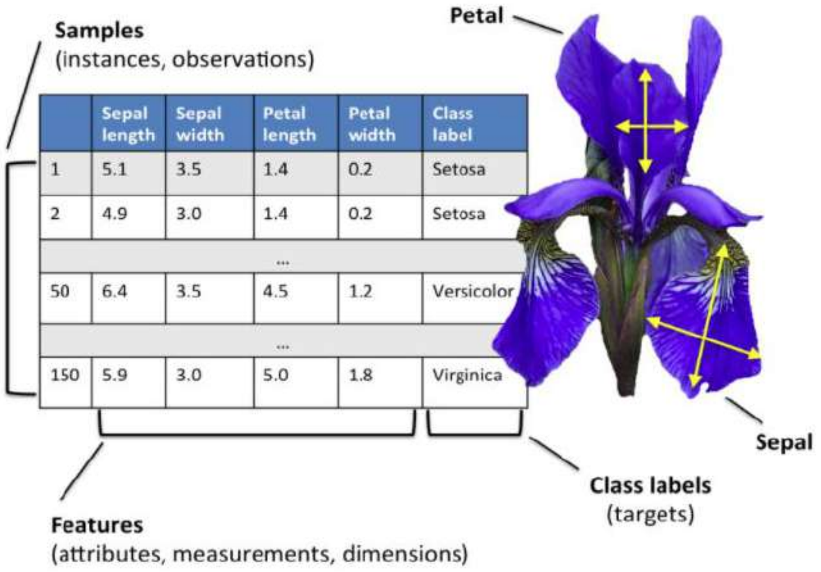
	3	4	5
4	21.50	26.925	28.2
6	19.75	19.750	19.7
8	15.05	0	15.4

R Programming

iris - DataSet

1. **iris** data set gives the measurements in centimeters of the variables sepal length, sepal width, petal length and petal width, respectively, for 50 flowers from each of 3 species of iris.
2. The species are Iris **setosa, versicolor, and virginica.**

Iris Versicolor | Iris Setosa | Iris Virginica



#Loading Data

data(iris)

str(iris)

'data.frame': 150 obs. of 5 variables:

$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...

$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...

$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...

$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...

$ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

 # Number of rows (observations)

nrow(iris) #150

 # Number of columns (variables)

ncol(iris) #5

#dimension of dataset

dim(iris) #150 5

names(mtcars)

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

iris - DataSet

head(iris)

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

 # Get first 5 rows of each subset
subset(iris, Species == "virginica")[1:5,]

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
101	6.3	3.3	6.0	2.5	virginica
102	5.8	2.7	5.1	1.9	virginica
103	7.1	3.0	5.9	2.1	virginica
104	6.3	2.9	5.6	1.8	virginica
105	6.5	3.0	5.8	2.2	virginica

 # Get column "Species" for all lines where Petal.Length < 2
subset(iris, Petal.Length < 2)[,"Species"]

```
 [1] setosa setosa setosa setosa setosa setosa
setosa setosa setosa setosa setosa setosa setosa
setosa setosa setosa setosa
[18] setosa setosa setosa setosa setosa setosa
setosa setosa setosa setosa setosa setosa setosa
setosa setosa setosa setosa
[35] setosa setosa setosa setosa setosa setosa
setosa setosa setosa setosa setosa setosa setosa
setosa setosa setosa
Levels: setosa versicolor virginica
```

iris - DataSet

summary(iris)

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

quantile(iris$Petal.Length)

```
  0%  25%  50%  75% 100%
1.00 1.60 4.35 5.10 6.90
```

tapply(iris$Petal.Length, iris$Species, length)

```
setosa versicolor  virginica
    50         50         50
```

tapply(iris$Petal.Length, iris$Species, mean)

```
setosa   versicolor  virginica
 1.462      4.260      5.552
```

NIELIT

R Programming

Titanic - DataSet

This data set provides information on the fate of passengers on the fatal maiden voyage of the ocean liner 'Titanic', summarized according to economic status (class), sex, age and survival.

A 4-dimensional array resulting from cross-tabulating 2201 observations on 4 variables.

The variables and their levels are as follows:

No	Name	Levels
1	Class	1st, 2nd, 3rd, Crew
2	Sex	Male, Female
3	Age	Child, Adult
4	Survived	No, Yes

#Loading Data

data(Titanic)

```
str(Titanic)
 'table' num [1:4, 1:2, 1:2, 1:2] 0 0 35 0 0 0 17 0 118 154 ...
 - attr(*, "dimnames")=List of 4
  ..$ Class   : chr [1:4] "1st" "2nd" "3rd" "Crew"
  ..$ Sex     : chr [1:2] "Male" "Female"
  ..$ Age     : chr [1:2] "Child" "Adult"
  ..$ Survived: chr [1:2] "No" "Yes“
```

dim(Titanic)

[1] 4 2 2 2

```
Titanic
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
  1st   118      4
  2nd   154     13
  3rd   387     89
  Crew  670      3

, , Age = Child, Survived = Yes

      Sex
Class  Male Female
  1st     5      1
  2nd    11     13
  3rd    13     14
  Crew    0      0

, , Age = Adult, Survived = Yes

      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```

Titanic - DataSet

Use an appropriate apply function to get the sum of males vs females aboard.
apply(Titanic, 2, sum)

#expected result

Male	Female
1731	470

b. Get a table with the sum of survivors vs sex.
apply(Titanic, c(2,4), sum)

#expected result

	Survived	
Sex	No	Yes
Male	1364	367
Female	126	344

c. Get a table with the sum of passengers by sex vs age.
apply(Titanic, c(3,2), sum)

#expected result

	Sex	
Age	Male	Female
Child	64	45
Adult	1667	425

```
Titanic
, , Age = Child, Survived = No

     Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

     Sex
Class  Male Female
  1st   118      4
  2nd   154     13
  3rd   387     89
  Crew  670      3

, , Age = Child, Survived = Yes

     Sex
Class  Male Female
  1st     5      1
  2nd    11     13
  3rd    13     14
  Crew    0      0

, , Age = Adult, Survived = Yes

     Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```

राइसूप्रौसं NIELIT

Contents to be covered

1. Line Graph
2. Bar Graph
3. Pie Graph
4. Histogram Graph

Types of Plots

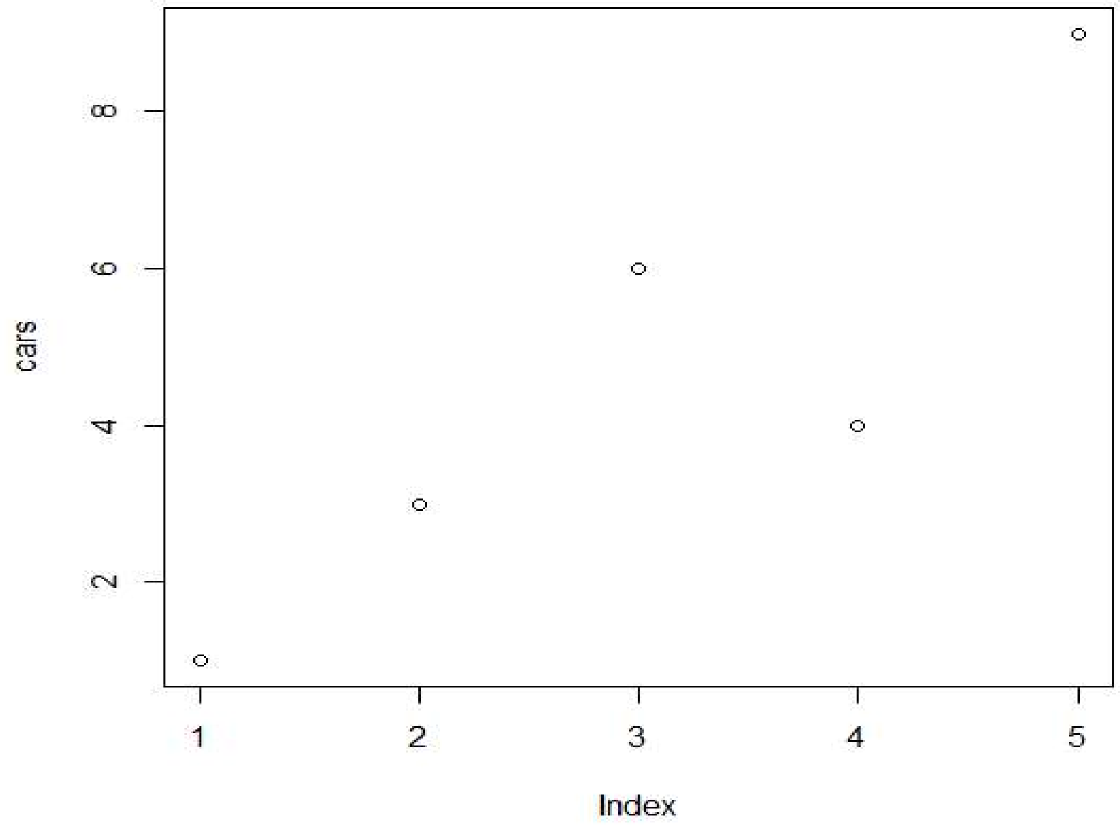
There are various plots which can be created using R. Some of them are listed below:

Line Graph

#Line

 # Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

 # Graph the cars vector with all defaults
plot(cars)

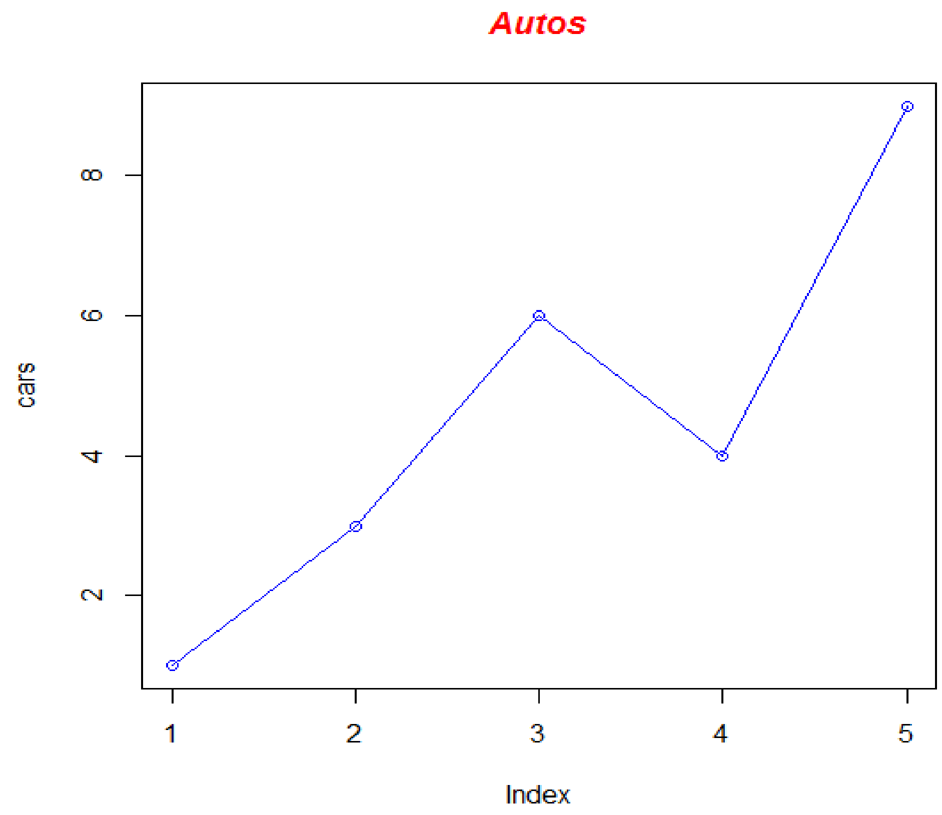


Line Graph

 # Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

 # Graph cars using blue points over layed by a line
plot(cars, type="o", col="blue")

 # Create a title with a red, bold/italic font
title(main="Autos", col.main="red", font.main=4)

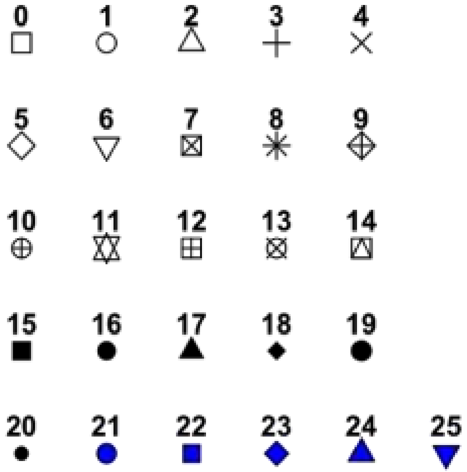


`type` what type of plot should be drawn. Possible types are

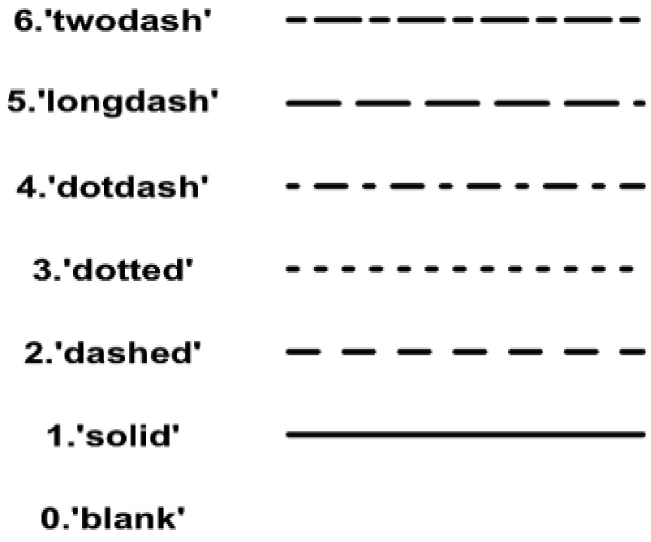
- `"p"` for **p**oints,
- `"l"` for **l**ines,
- `"b"` for **b**oth,
- `"c"` for the lines part alone of `"b"`,
- `"o"` for both '**o**verplotted',
- `"h"` for '**h**istogram' like (or 'high-density') vertical lines,
- `"s"` for stair **s**teps,
- `"S"` for other **s**teps, see 'Details' below,
- `"n"` for no plotting.

Line Graph

PCH- Plot Character



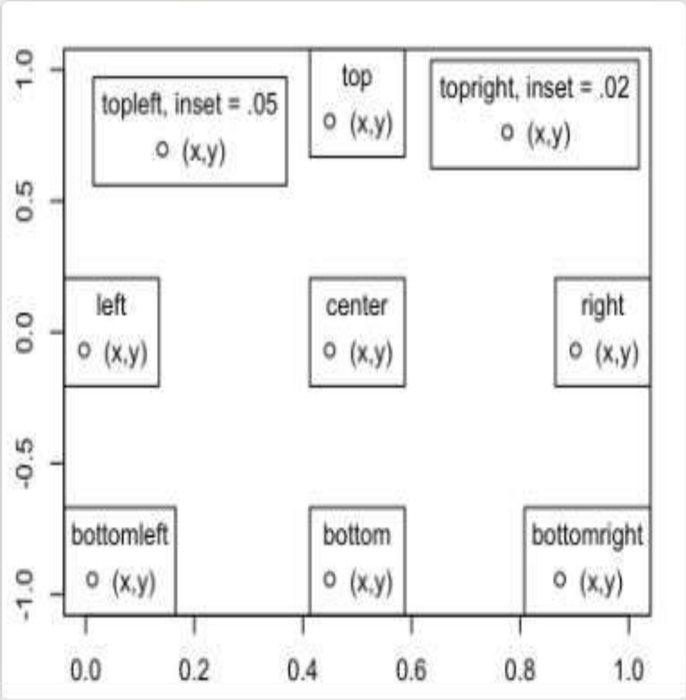
LTY – Line Style



Specify legend position by keywords

keywords : "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center“.

The effect of using each of these keywords are shown in the figure below :



Line Graph

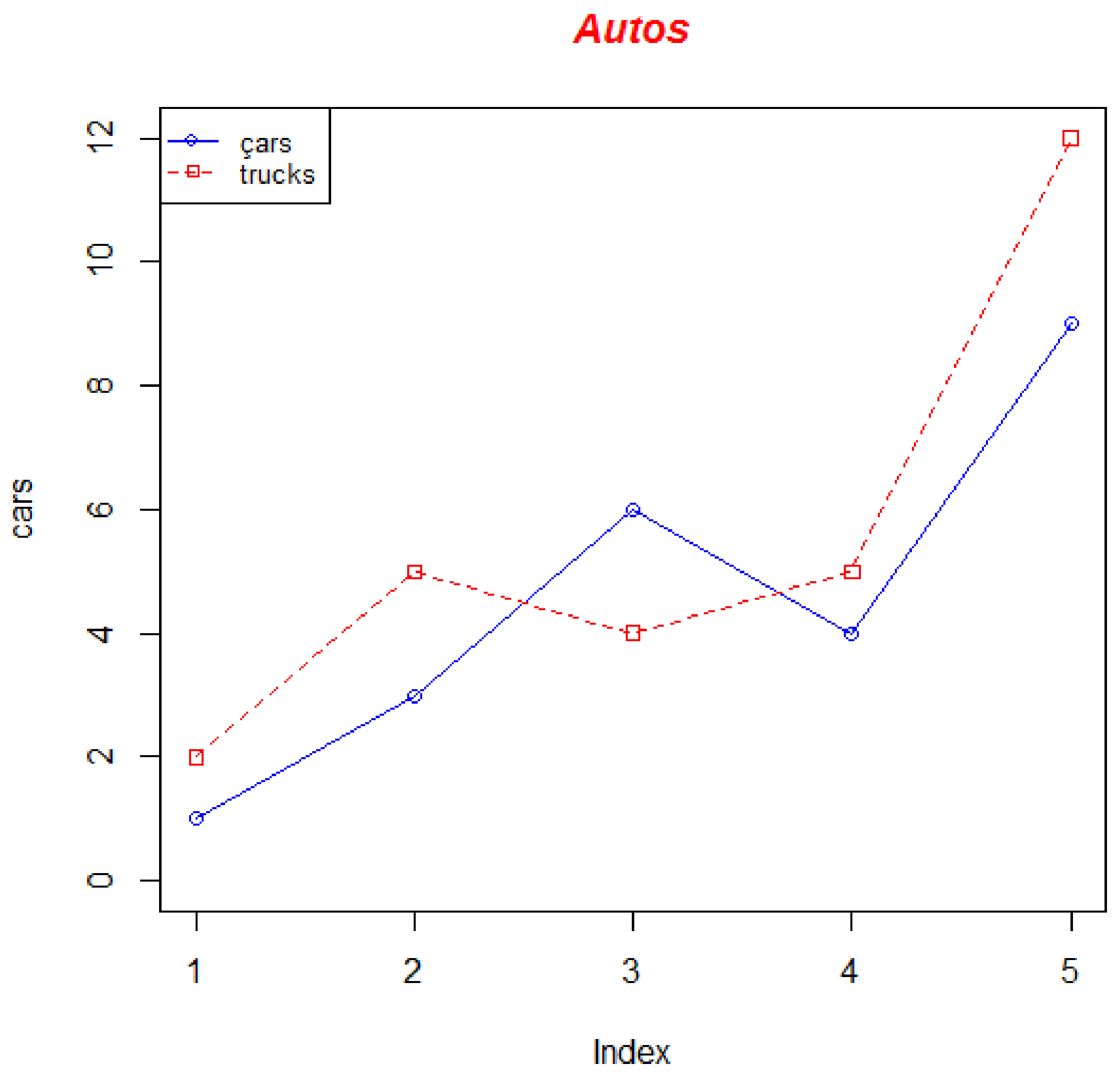
 # Define 2 vectors
cars <- c(1, 3, 6, 4, 9)
trucks <- c(2, 5, 4, 5, 12)

 # Graph cars using a y axis that ranges from 0 to 12
plot(cars, type="o", col="blue", ylim=c(0,12))

 # Graph trucks with red dashed line and square points
lines(trucks, type="o", pch=22, lty=2, col="red")

 # Create a title with a red, bold/italic font
title(main="Autos", col.main="red", font.main=4)

legend('topleft', legend=c('çars','trucks'), cex=0.8, col=c('blue','red'), pch=21:22, lty=1:2)



Bar Graph

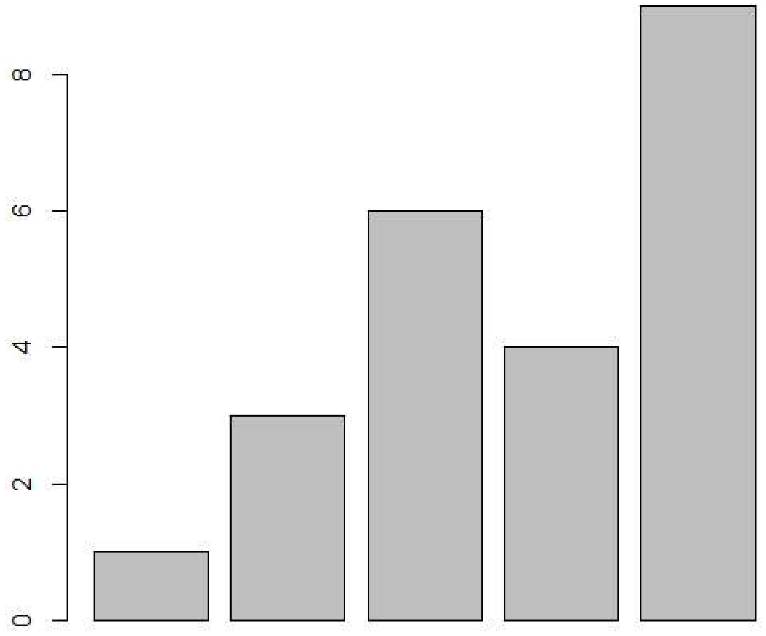
#Example-1

 # Define the cars vector with 5 values

```
cars <- c(1, 3, 6, 4, 9)
```

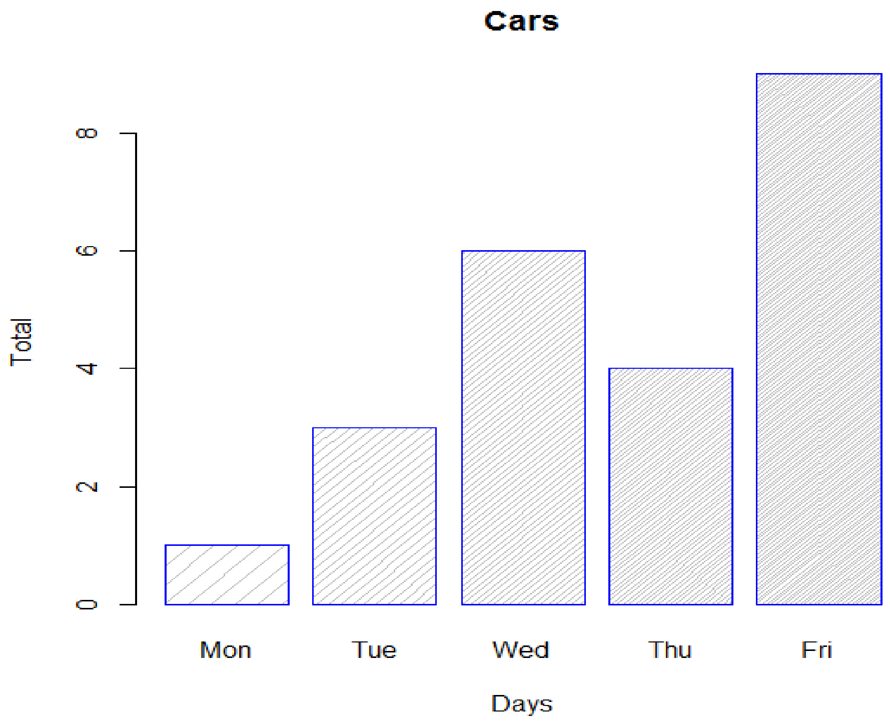
 # Graph cars

```
barplot(cars)
```



#Example-2

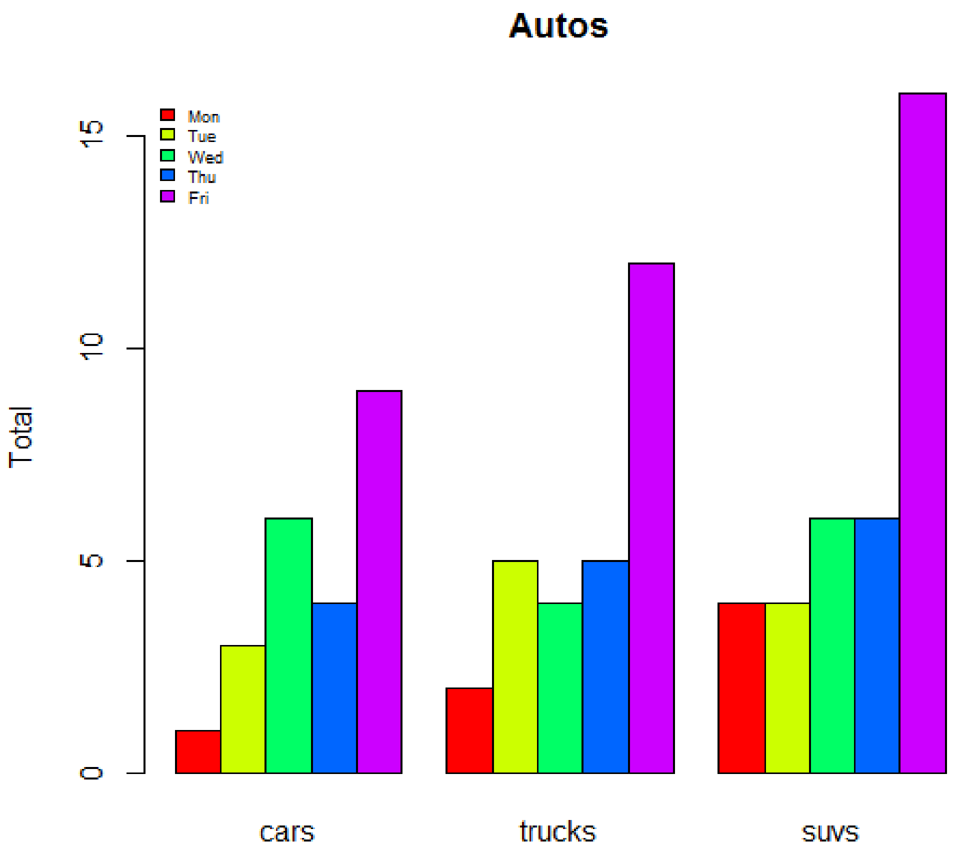
```
cars <- c(1, 3, 6, 4, 9)
month=c("Mon","Tue","Wed","Thu","Fri")

barplot(cars, names.arg=month ,  main='Cars' , xlab='Days', ylab='Total',
border='blue', density=c(10,20,30,40,50) )
```



Bar Graph

#Example-4

```
# Read values from tab-delimited autos.dat
#autos_data <- read.table("C:/Users/User/Documents/RPrg/autos.dat", header=T, sep="\t")

cars=c(1,3,6,4,9)
trucks=c(2,5,4,5,12)
suvs=c(4,4,6,6,16)
autos_data=data.frame(cars,trucks,suvs)

# Graph autos with adjacent bars using rainbow colors
barplot( as.matrix(autos_data), main="Autos", ylab= "Total",
      beside=TRUE, col=rainbow(5) )

# Place the legend at the top-left corner with no frame
# using rainbow colors
legend("topleft", c("Mon","Tue","Wed","Thu","Fri"), cex=0.6,
   bty="n", fill=rainbow(5))
```



Bar Graph

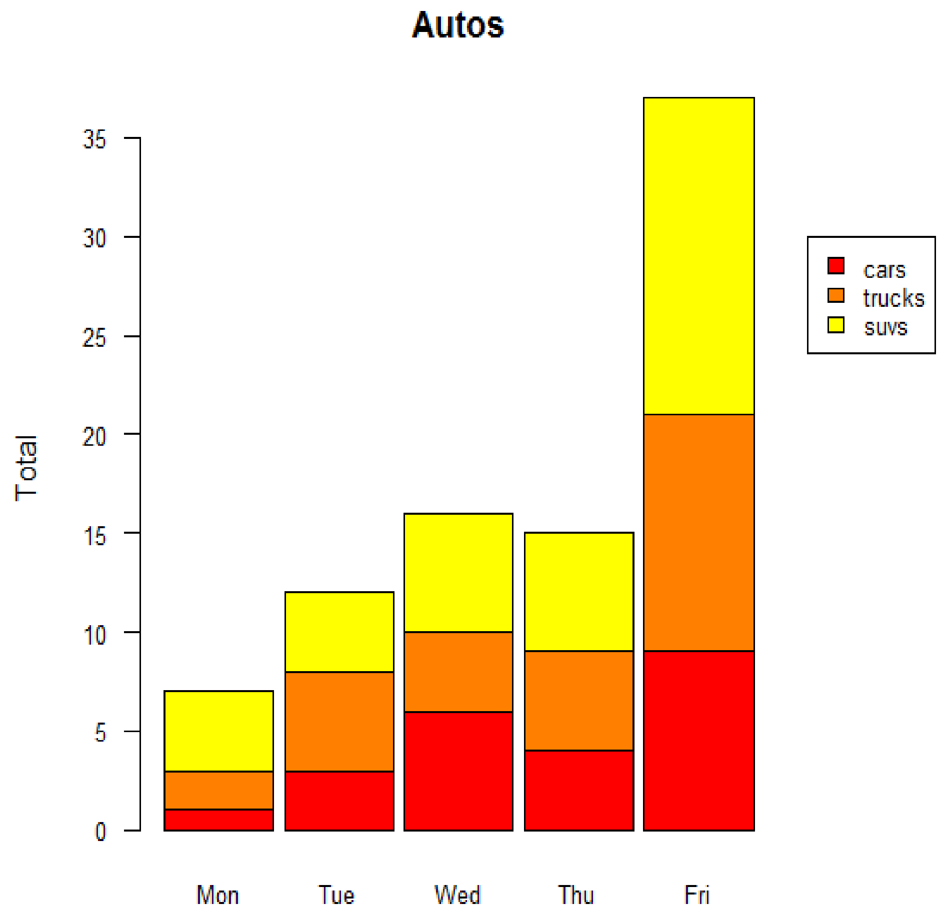
 # Example-5
cars=c(1,3,6,4,9)   trucks=c(2,5,4,5,12)   suvs=c(4,4,6,6,16)
autos_data = data.frame(cars,trucks,suvs)

 # Expand right side of clipping rect to make room for the legend
par(xpd=T, mar=par()$mar+c(0,0,0,4))

 # Graph autos (transposing the matrix) using heat colors,
 # put 10% of the space between each bar, and
 # cex.axis=0.8 make labels smaller with horizontal y-axis labels
 # cex=0.8 make labels smaller with horizontal x-axis labels
 # las = 1 horizontal , 2 vert Orientation of Tick Marks

barplot(t(autos_data), main="Autos", ylab="Total", col=heat.colors(3), space=0.1,
cex.axis=0.8, las=1, names.arg=c("Mon","Tue","Wed","Thu","Fri"), cex=0.8)

 # Place the legend at (6,30) using heat colors
legend(6, 30, names(autos_data), cex=0.8, fill=heat.colors(3));

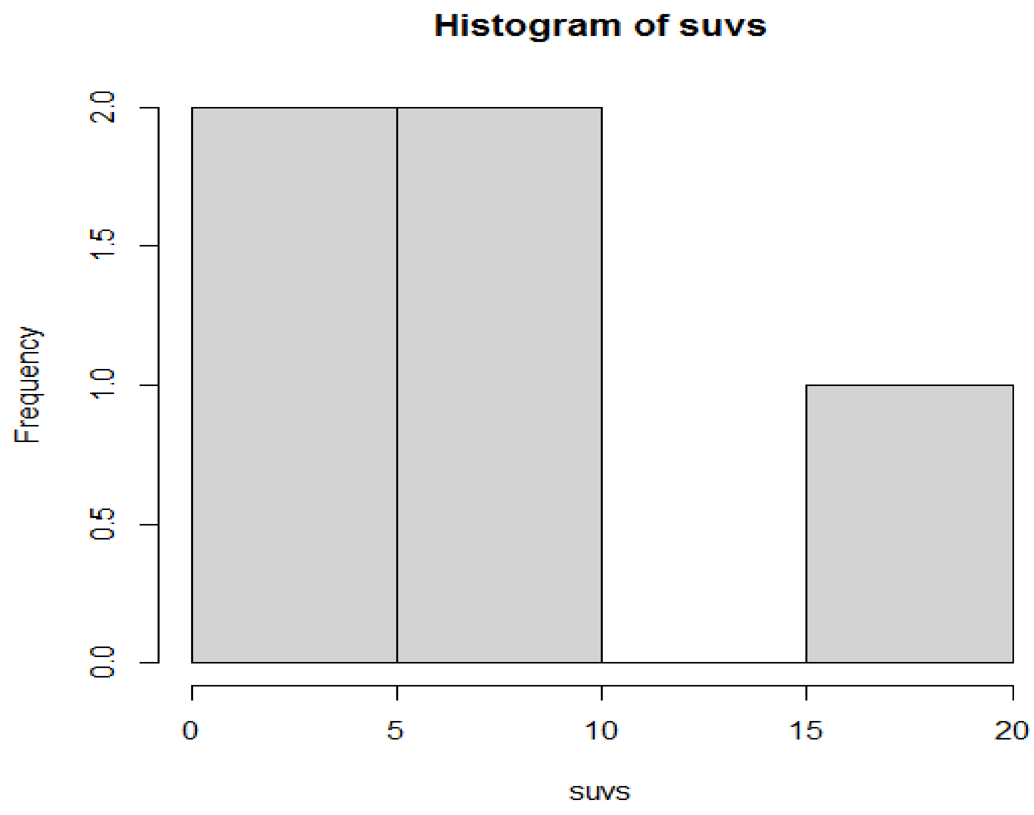


Histogram Graph

#Example-1

 # Define the suvs vector with 5 values
suvs <- c(4,4,6,6,16)

 # Create a histogram for suvs
hist(suvs)

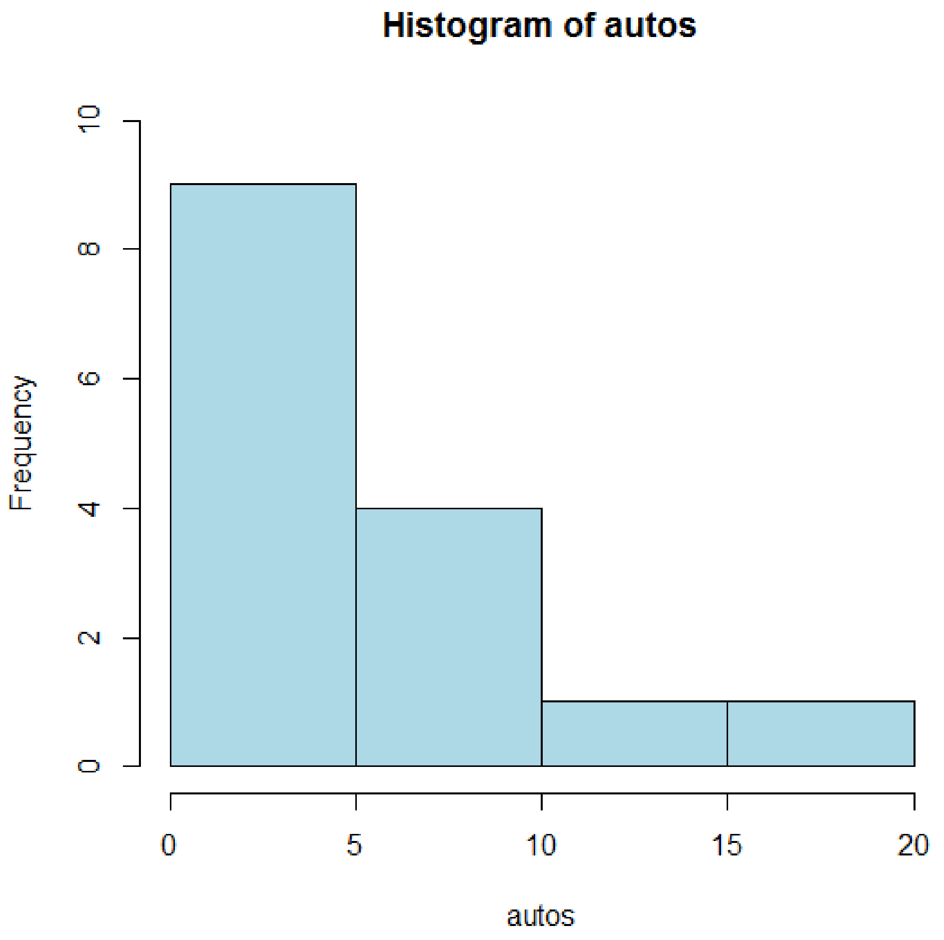


Histogram Graph

#Example-2
 # Read values from tab-delimited autos.dat
cars=c(1,3,6,4,9)
trucks=c(2,5,4,5,12)
suvs=c(4,4,6,6,16)
autos_data=data.frame(cars, trucks, suvs)

 # Concatenate the three vectors
autos <- c(autos_data$cars, autos_data$trucks,   autos_data$suvs)

 # Create a histogram for autos in light blue with the y axis
 # ranging from 0-10
hist(autos, col="lightblue", ylim=c(0,10))



Histogram Graph

#Example-3
 # Read values from tab-delimited autos.dat

```
cars=c(1,3,6,4,9)
trucks=c(2,5,4,5,12)
suvs=c(4,4,6,6,16)
autos_data=data.frame(cars,trucks,suvs)
```

 # Concatenate the three vectors

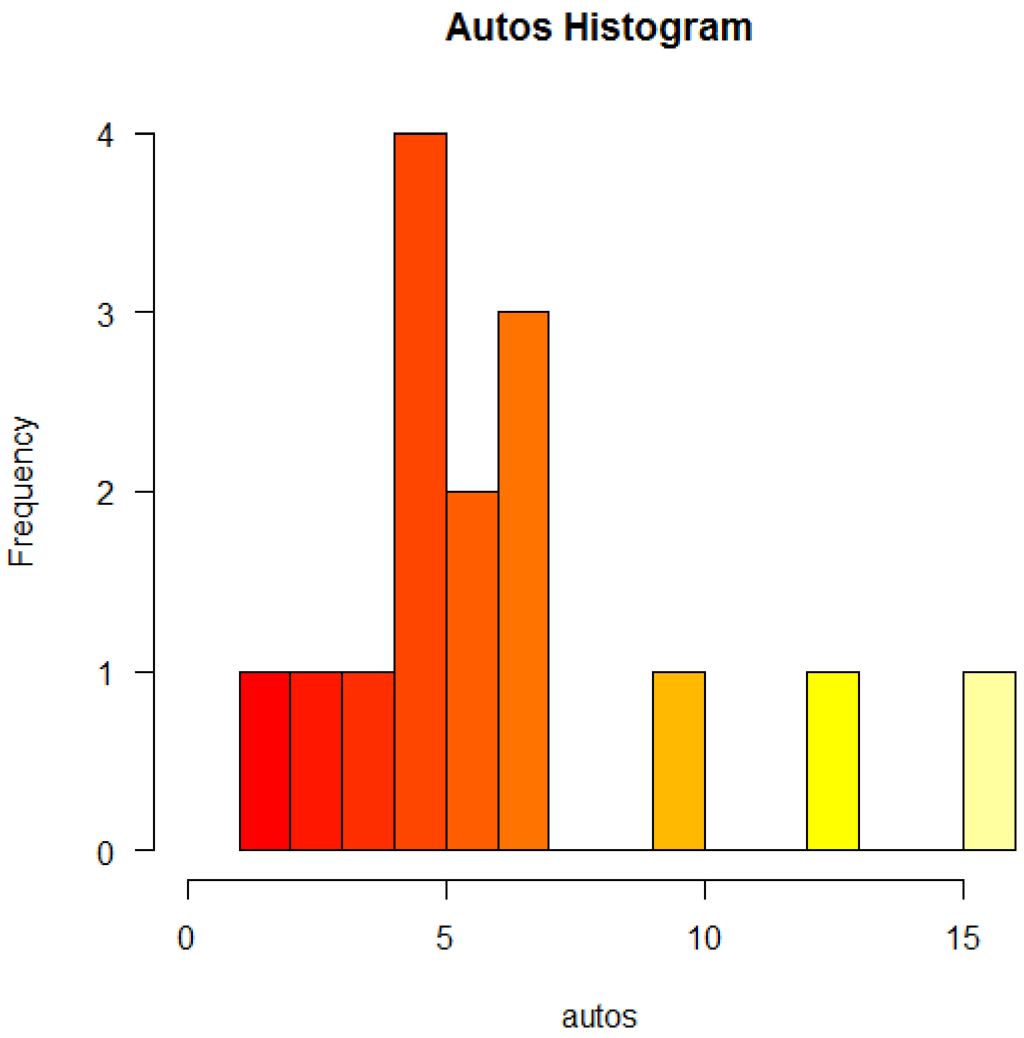
```
autos <- c(autos_data$cars, autos_data$trucks,     autos_data$suvs)
```

 # Compute the largest y value used in the autos

```
max_num <- max(autos)
```

 # Create a histogram for autos with fire colors, set breaks
 # so each number is in its own group, make x axis range from
 # 0-max_num, disable right-closing of cell intervals,
 # set heading, and make y-axis labels horizontal

```
hist(autos, col=heat.colors(max_num), breaks=max_num,
xlim=c(0,max_num), right=False, main="Autos Histogram", las=1)
```



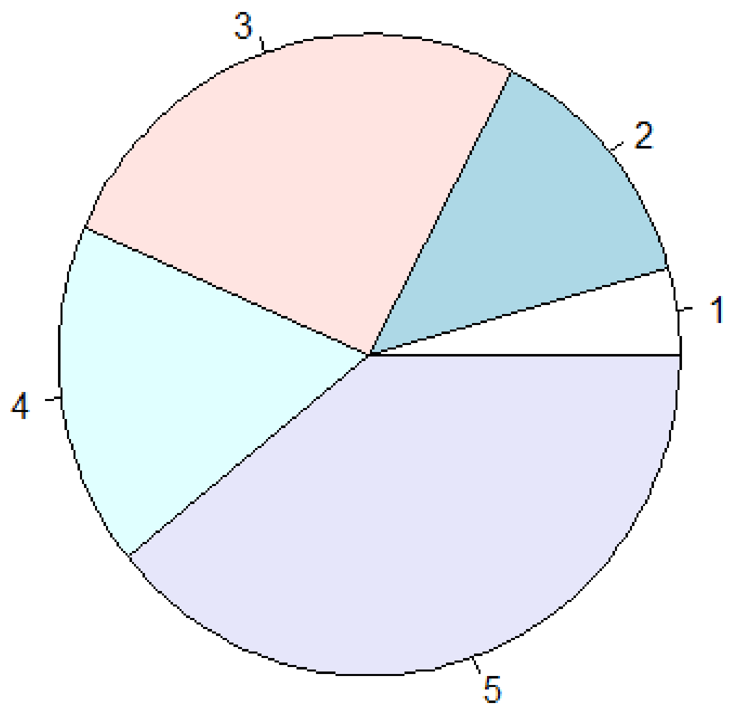
Pie Graph

#Example-1
 # Define cars vector with 5 values

```
cars <- c(1, 3, 6, 4, 9)
```

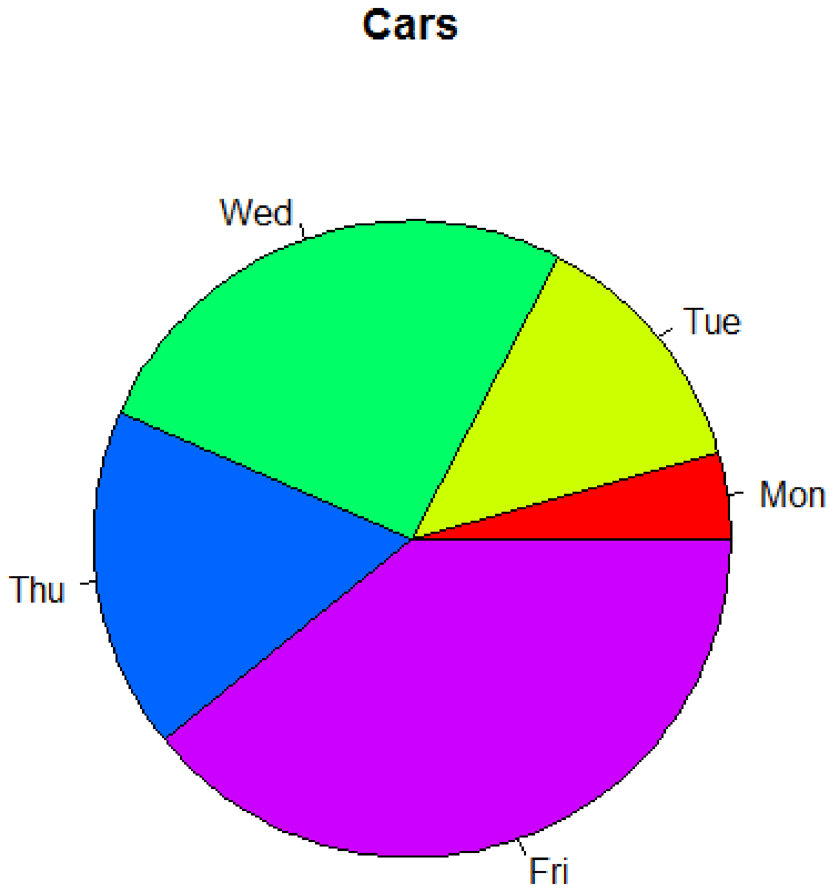
 # Create a pie chart for cars

```
pie(cars)
```



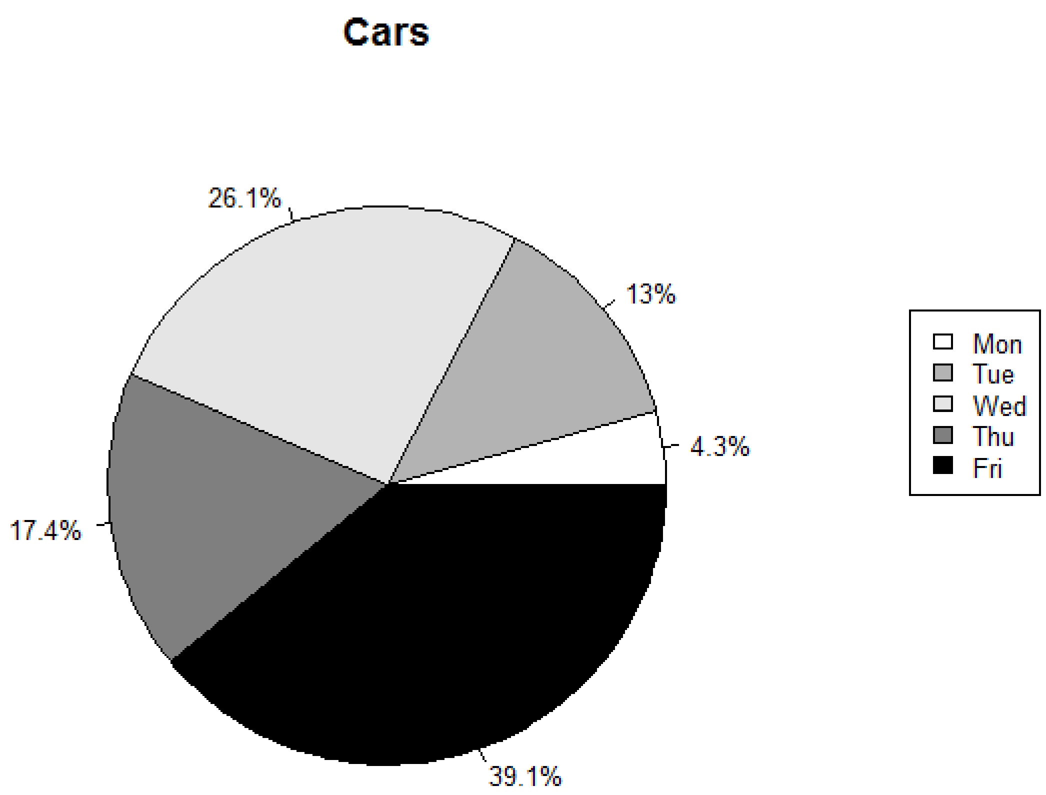
Pie Graph

```
#Example-2
# Define cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

# Create a pie chart with defined heading and
# custom colors and labels
pie(cars,  main="Cars",  col=rainbow(length(cars)) ,
labels=c("Mon", "Tue", "Wed", "Thu", "Fri"))
```



Pie Graph

```
#Example-3
# Define cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)

# Define some colors ideal for black & white print
colors <- c("white","grey70","grey90","grey50","black")

# Calculate the percentage for each day, rounded to one decimal place
car_labels <- round(cars/sum(cars) * 100, 1)

# Concatenate a '%' char after each value
car_labels <- paste(car_labels, "%", sep="")

# Create a pie chart with defined heading and custom colors
# and labels
pie(cars, main="Cars", col=colors, labels=car_labels,    cex=0.8)

# Create a legend at the right
legend(1.5, 0.5, c("Mon","Tue","Wed","Thu","Fri"), cex=0.8,
    fill=colors)
```



Contents to be covered

1. Box Chart
2. Strip Chart
3. Plot Function
4. Sub Plot
5. Saving Plot
6. Colors in Graph

Box Plot

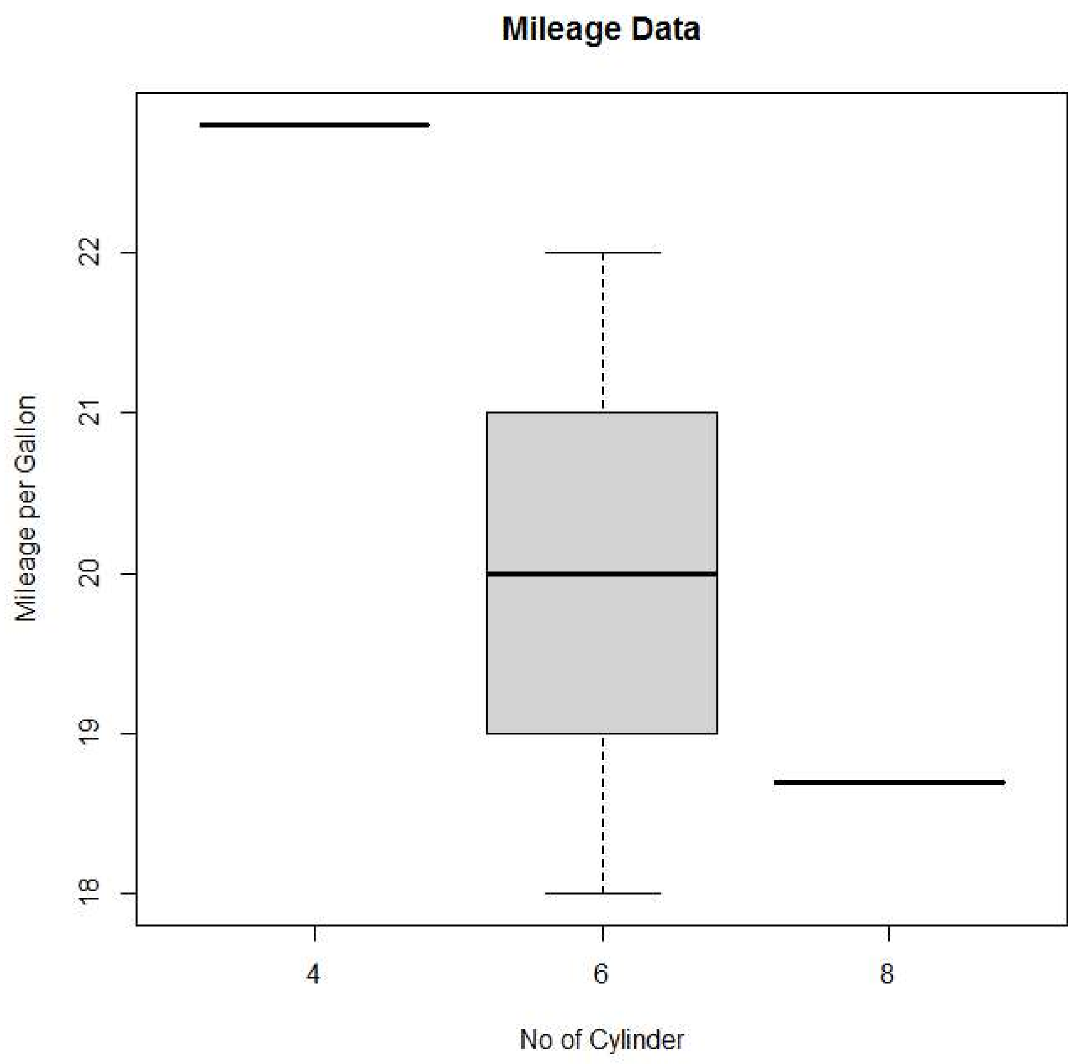
- Boxplots are a measure of how well distributed is the data in a data set.
- It divides the data set into three quartiles.
- This graph represents the minimum, maximum, median, first quartile and third quartile in the data set.
- It is also useful in comparing the distribution of data across data sets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

```
boxplot(x, data, notch, varwidth, names, main)
```

x is a vector or a formula.
data is the data frame.
notch is a logical value. Set as TRUE to draw a notch.
varwidth is a logical value. Set as true to draw width of the box proportionate to the sample size.
names are the group labels which will be printed under each boxplot.
main is used to give a title to the graph.

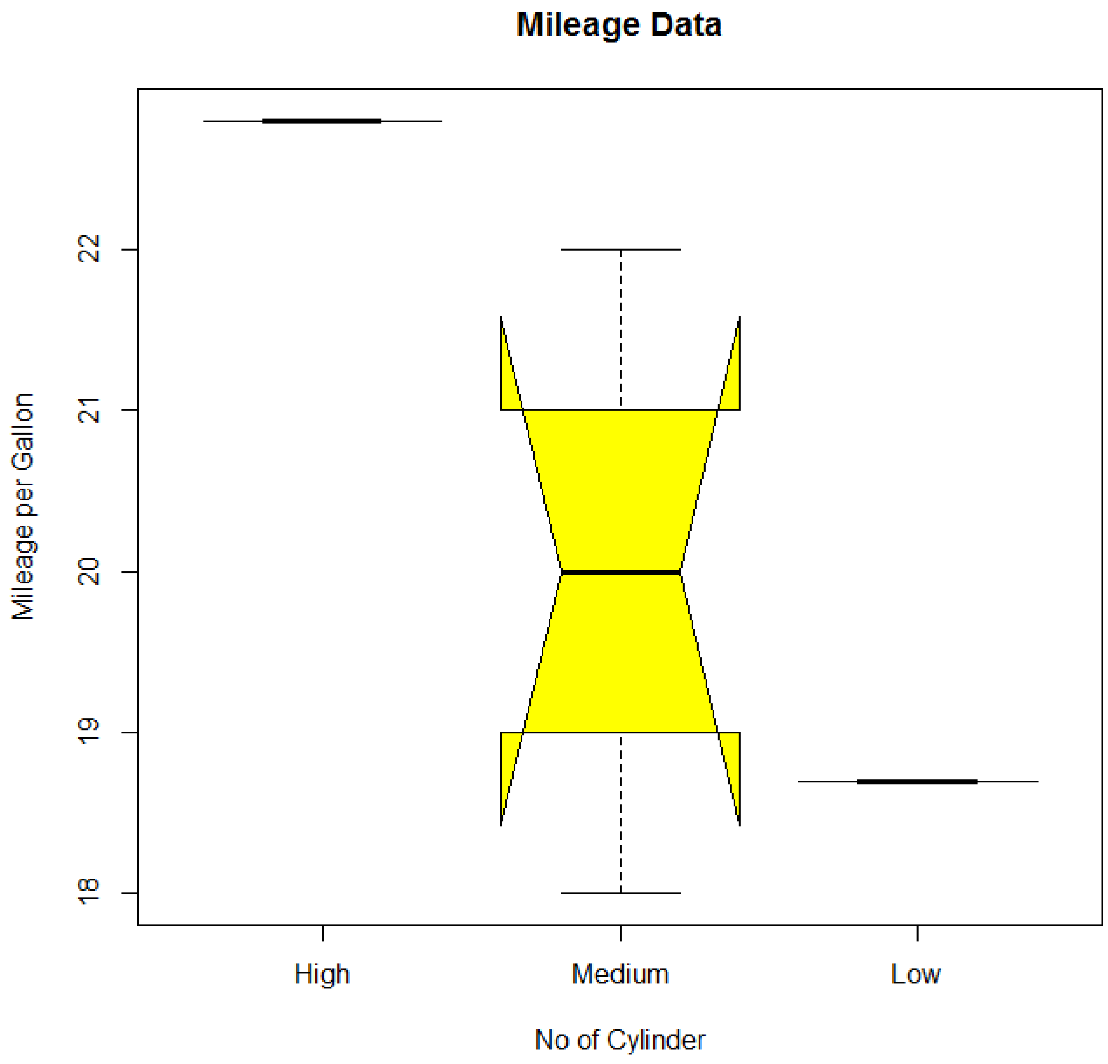
Box Plot

```
car_name=c('Mazda RX4', 'Mazda Rx4 Wag',
'Datsun 710', 'Hornet 4 Drive', 'Hornet Sportabout',
'Valiant')
mpg=c(20.0, 20.0, 22.8, 22.0 , 18.7 , 18.0 )
cyl=c(6,6,4,6,8,6)

car=data.frame( car_name , mpg, cyl)
print( car)


boxplot( mpg~cyl, data=car, xlab='No of Cylinder',
ylab='Mileage per Gallon',  main='Mileage Data' )
```



Box Plot

```
car_name=c('Mazda RX4', 'Mazda Rx4 Wag', 'Datsun
710', 'Hornet 4 Drive', 'Hornet Sportabout', 'Valiant')
mpg=c(20.0, 20.0, 22.8, 22.0 , 18.7 , 18.0 )
cyl=c(6,6,4,6,8,6)

car=data.frame( car_name , mpg, cyl)
print( car)


boxplot( mpg~cyl,data=car,notch=TRUE,
col = c("green","yellow","purple"),
  names = c("High","Medium","Low"),
xlab='No of Cylinder',
ylab='Mileage per Gallon',  main='Mileage Data' )
```



Stripchart Graph

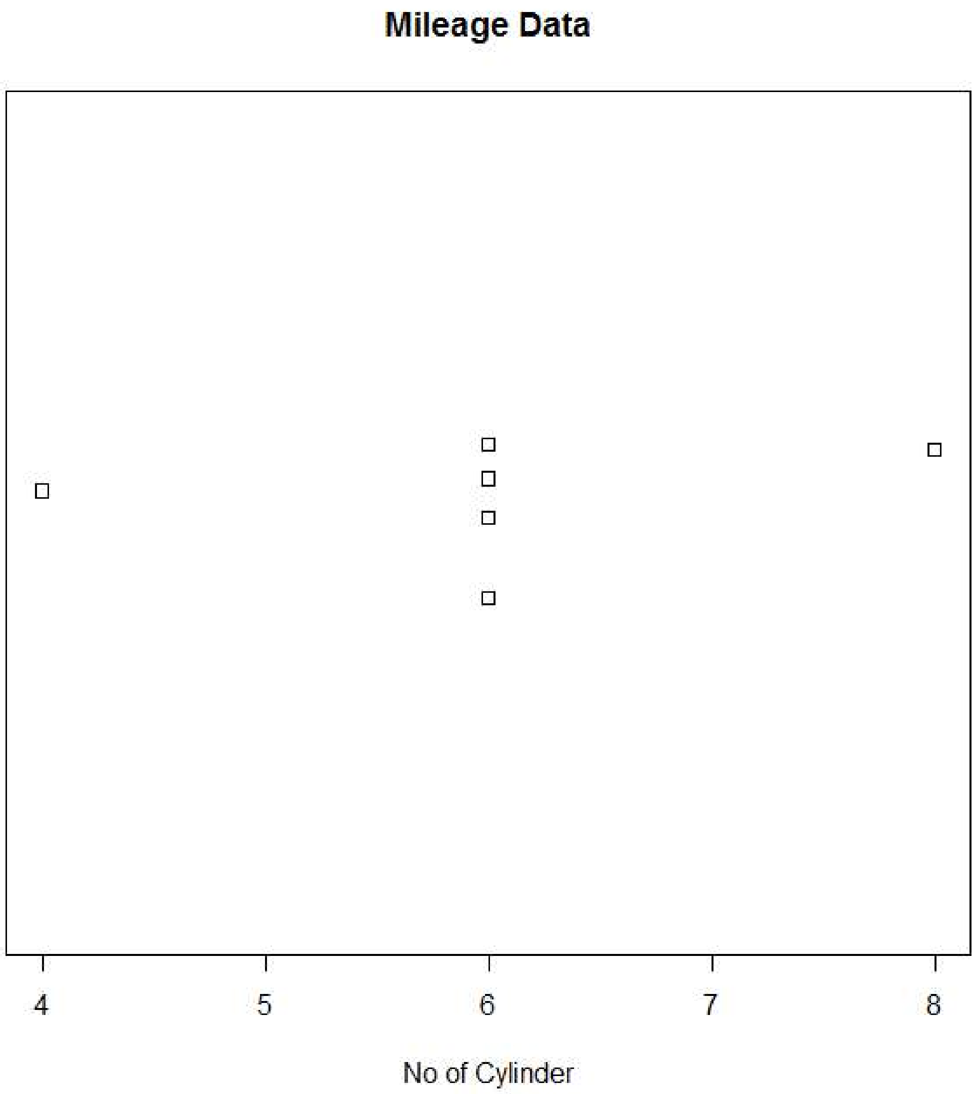
produces one dimensional scatter plots (or dot plots) of the given data. These plots are a good alternative to boxplots when sample sizes are small.

stripchart(x, method, jitter, main, xlab, ylab, col, pch, vertical, group.names)

Method- overplot, stack , jitter
pch: shape of the points in the plot
vertical: when vertical is "TRUE", the plot is drawn vertically rather than the default horizontal

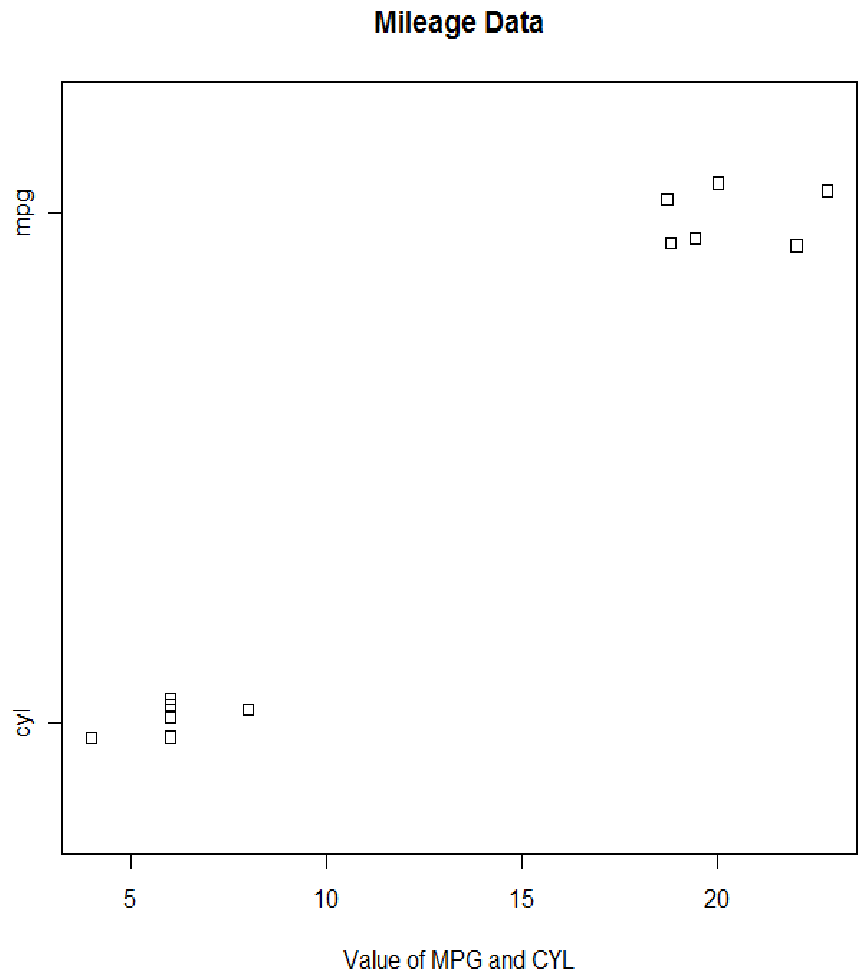
StripChart Graph

```
car_name=c('Mazda RX4', 'Mazda Rx4 Wag', 'Datsun 710', 'Hornet 4 Drive', 'Hornet Sportabout', 'Valiant')
mpg=c(19.4, 20.0, 22.8, 22.0 , 18.7 , 18.8 )
cyl=c(6,6,4,6,8,6)

car=data.frame( car_name , mpg, cyl)
stripchart( car$cyl, xlab='No of Cylinder', method='stack', main='Mileage Data' )

stripchart( car$cyl, xlab='No of Cylinder', method='jitter', main='Mileage Data' )
```



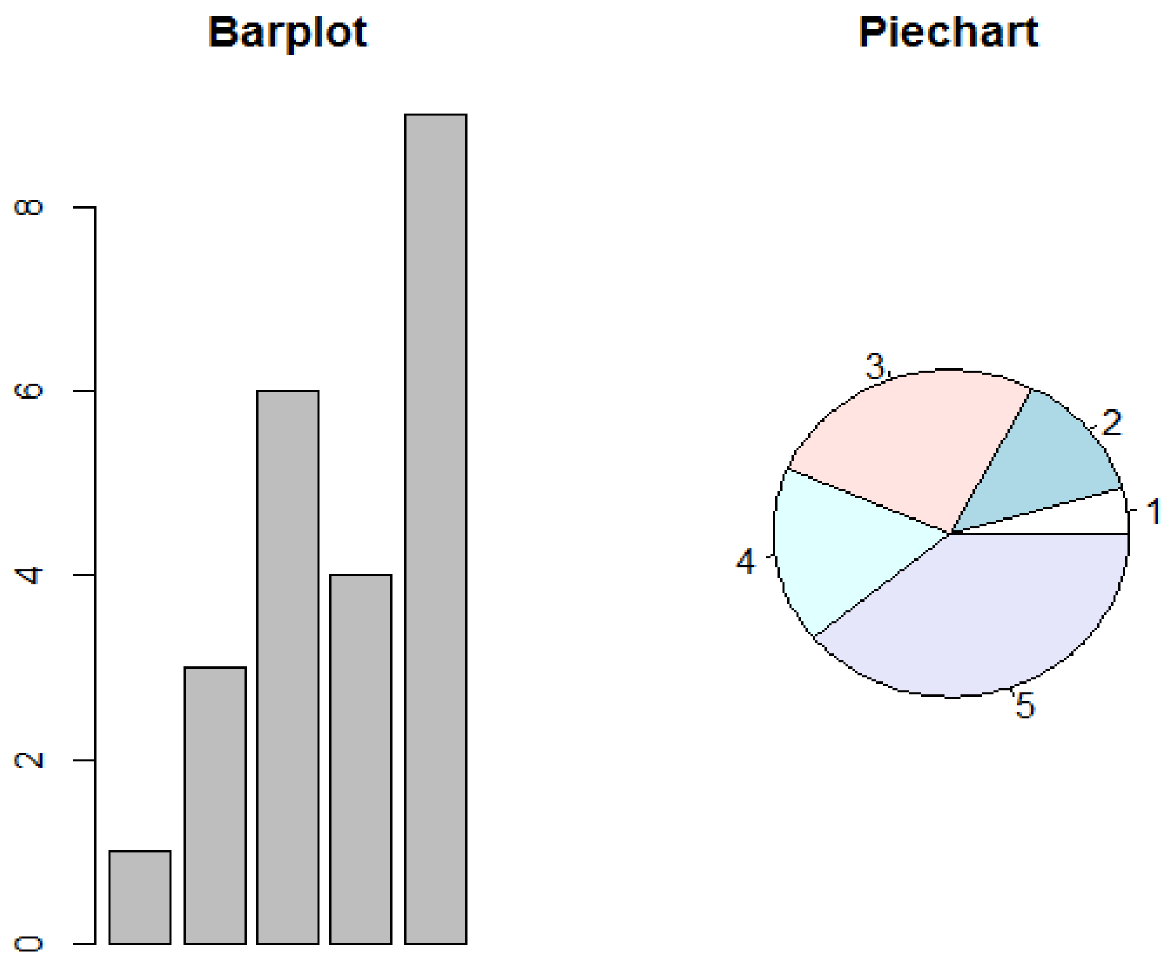
StripChart Graph

```
car_name=c('Mazda RX4', 'Mazda Rx4 Wag', 'Datsun
710', 'Hornet 4 Drive', 'Hornet Sportabout', 'Valiant')
mpg=c(19.4, 20.0, 22.8, 22.0 , 18.7 , 18.8 )
cyl=c(6,6,4,6,8,6)

car=data.frame( car_name , mpg, cyl)

stripchart( list('cyl'=car$cyl, 'mpg'=car$mpg),
xlab='Value of MPG and CYL',
 method='stack', main='Mileage Data' )

stripchart( list('cyl'=car$cyl, 'mpg'=car$mpg),
xlab='Value of MPG and CYL',
 method='jitter', main='Mileage Data' )
```



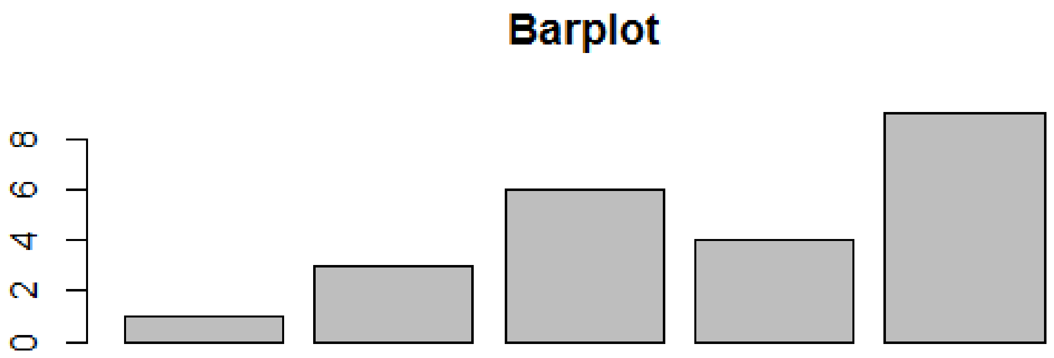
Multiple Graphs

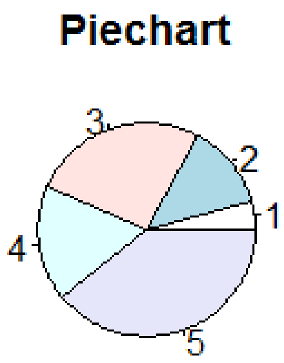
```
#Example-1
cars <- c(1, 3, 6, 4, 9)
par(mfrow=c(1,2))

# set the plotting area into a 1*2 array

barplot(cars, main="Barplot")
pie(cars, main="Piechart", radius=1)
dev.off()
```



Multiple Graphs

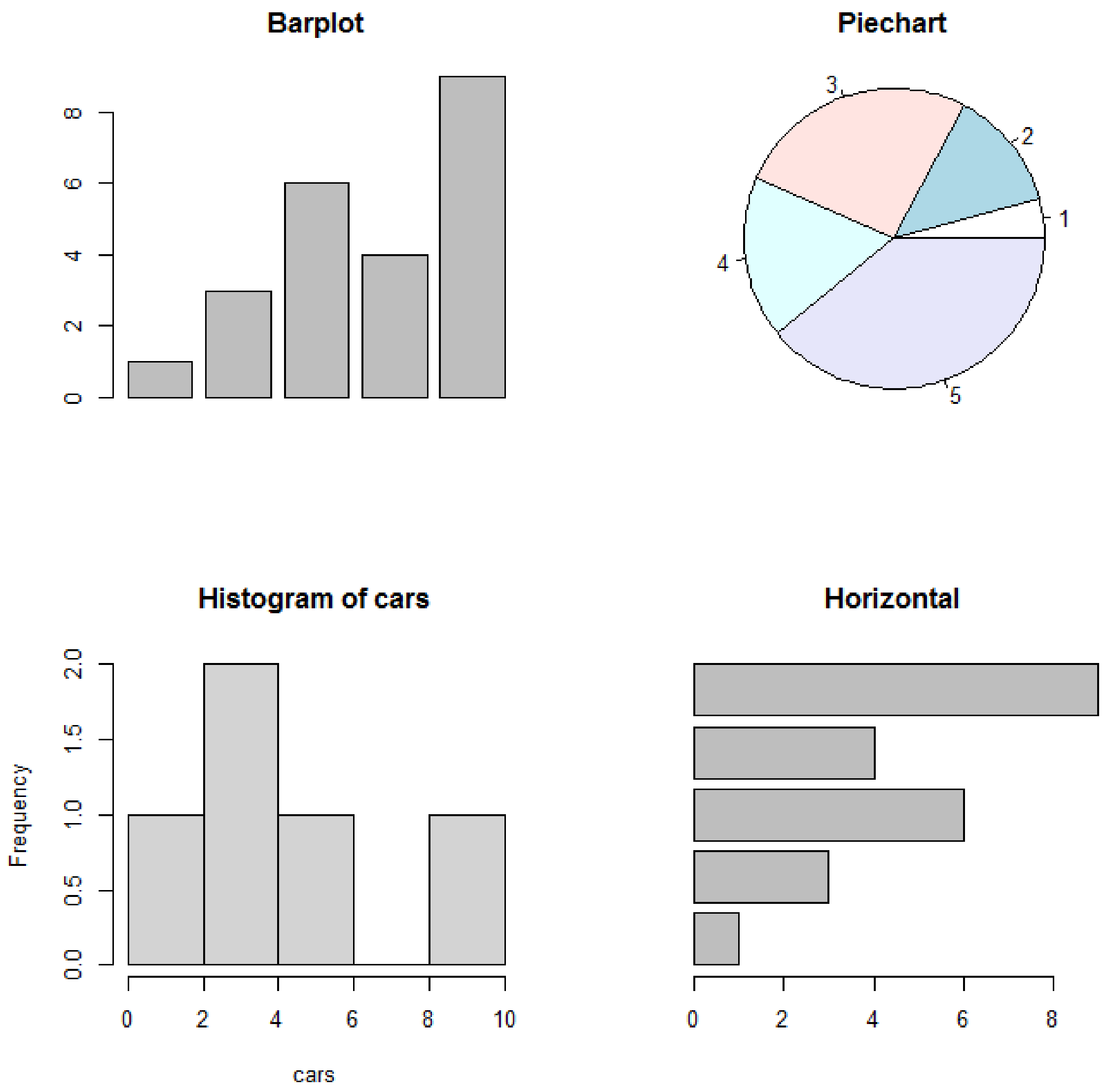
```
#Example-2
cars <- c(1, 3, 6, 4, 9)

par(mfcol=c(2,1))
# set the plotting area into a 1*2 array

barplot(cars, main="Barplot")
pie(cars, main="Piechart", radius=1)
dev.off()
```





Multiple Graphs

```
#Example-3
cars <- c(1, 3, 6, 4, 9)

par(mfrow=c(2,2))

# set the plotting area into a 1*2 array
barplot(cars, main="Barplot")

pie(cars, main="Piechart", radius=1)

hist(cars)

barplot(cars, main='Horizontal', horiz=TRUE)
dev.off()
```



Saving Graphs

```
cars <- c(1, 3, 6, 4, 9)

pdf("rplot.pdf")
par(mfrow=c(1,2))

# set the plotting area into a 1*2 array
barplot(cars, main="Barplot")
pie(cars, main="Piechart", radius=1)
dev.off()
```

```
cars <- c(1, 3, 6, 4, 9)
jpeg("rplot2.jpg", width = 350, height = 350)
par(mfrow=c(1,2))

# set the plotting area into a 1*2 array
barplot(cars, main="Barplot")
pie(cars, main="Piechart", radius=1)
dev.off()
```

Export File Format

pdf('rplot.pdf'): pdf file
png('rplot.png'): png file
jpeg('rplot.jpg'): jpeg file
postscript('rplot.ps'): postscript file
bmp('rplot.bmp'): bmp file
win.metafile('rplot.wmf'): windows metafile

Contents to be covered

1. Mean , Median , Mode
2. Standard Deviation
3. Variance

राइसूप्रौसं NIELIT

Data

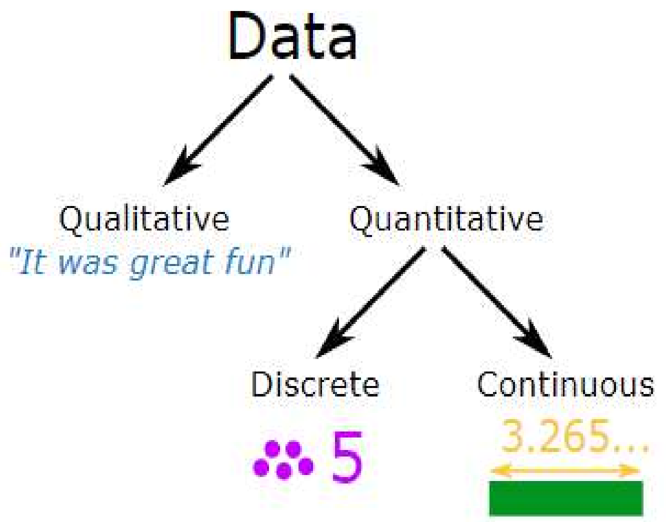
Data is a collection of facts, such as numbers, words, measurements, observations or just descriptions of things.

Qualitative vs Quantitative

Data can be qualitative or quantitative.

•**Qualitative data** is descriptive information (it *describes* something)

•**Quantitative data** is numerical information (numbers)



Data

Quantitative data can be Discrete or Continuous:

- **Discrete data** can only take certain values (like whole numbers)
- **Continuous data** can take any value (within a range)

Put simply: **Discrete data** is counted, **Continuous data** is measured

Example

Qualitative:
- Your friends' favorite holiday destination
- The most common given names in your town
- How people describe the smell of a new perfume

Quantitative:
- Height (Continuous)
- Weight (Continuous)
- Petals on a flower (Discrete)
- Customers in a shop (Discrete)

Qualitative:

- He is brown and black
- He has long hair
- He has lots of energy

Quantitative:

- Discrete:
  - He has 4 legs
  - He has 2 brothers
- Continuous:
  - He weighs 25.5 kg
  - He is 565 mm tall

Data Collection

Collecting
Data can be collected in many ways. The simplest way is direct observation.
Example: Counting Cars
You want to find how many cars pass by a certain point on a road in a 10-minute interval.

Census or Sample

A **Census** is when we collect data for **every** member of the group (the whole "population").
A **Sample** is when we collect data just for **selected members** of the group.

Example: 120 people in your local football club
You can ask everyone (all 120) what their age is. That is a census.
Or you could just choose the people that are there this afternoon. That is a sample.

A census is accurate, but hard to do. A sample is not as accurate, but may be good enough, and is a lot easier.

Measures of Central Value

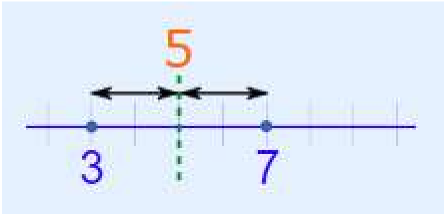
Finding a Central Value
Mean Value
Median Value
Mode or Modal Value

Central Value

Fining a Central Value

When you have two or more numbers it is nice to find a value for the "center".

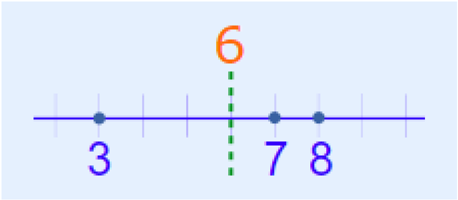
2 Numbers



With just 2 numbers the answer is easy: go half-way between.
Example: what is the central value for 3 and 7?
Answer: Half-way between, which is 5.

We can calculate it by adding 3 and 7 and then dividing the result by 2:
(3+7) / 2 = 10/2 = 5

3 or More Numbers



We can use that idea of "adding then dividing" when we have 3 or more numbers:

Example: what is the central value of 3, 7 and 8?
We calculate it by adding 3, 7 and 8 and then dividing the results by 3 :
(3+7+8) / 3 = 18/3 = 6

Mean

Mean: Add up the numbers and divide by how many numbers.

Example: Birthday Activities

Uncle Bob wants to know the average age at the party, to choose an activity.

There will be 6 kids aged 13, and also 5 babies aged 1. Add up all the ages, and divide by 11 (because there are 11 numbers):

(13+13+13+13+13+13+1+1+1+1+1) / 11 = **7.5...**

The mean age is about **7½**, so he gets a **Jumping Castle**!
The 13 year olds are embarrassed,
and the 1 year olds can't jump!

The Median

simply list all numbers in order and choose the middle one

Example: Birthday Activities (continued)

List the ages in order:
1, 1, 1, 1, 1, 13, 13, 13, 13, 13, 13

Choose the middle number:
1, 1, 1, 1, 1, **13**, 13, 13, 13, 13, 13

The Median age is **13** ... so let's have a **Disco**!

Sometimes there are **two** middle numbers. Just average those two:

Example: What is the Median of 3, 4, 7, 9, 12, 15
There are two numbers in the middle:
3, 4, 7, 9, 12, 15

So we average them:
(7+9) / 2 = 16/2 = 8

The Median is **8**

रा.इ.सू.प्रौ.सं NIELIT

Mode

The Mode is the value that occurs most often.

Example: Birthday Activities (continued)

Group the numbers so we can count them:
1, 1, 1, 1, 1, 13, 13, 13, 13, 13, 13

"13" occurs 6 times, "1" occurs only 5 times, so the mode is **13**.

But Mode can be tricky, there can sometimes be more than one Mode.

Example: What is the Mode of 3, 4, 4, 5, 6, 6, 7

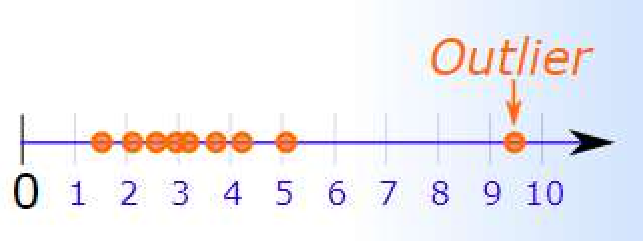
Well ... 4 occurs twice but 6 **also** occurs twice.
So **both 4 and 6** are modes.

When there are two modes it is called "bimodal", when there are three or more modes we call it "multimodal".

Outliers

Outliers are values that "**lie out**side" the other values.

They can change the mean a lot, so we can either not use them (and say so) or use the median or mode instead.



Example: 3, 4, 4, 5 and 104

Mean: Add them up, and divide by 5 (as there are 5 numbers):
(3+4+4+5+104) / 5 = **24**
24 does not represent those numbers well at all!

Without the 104 the mean is:
(3+4+4+5) / 4 = **4**
But please tell people you are not including the outlier.

Median: They are in order, so just choose the middle number, which is **4**:
3, 4, **4**, 5, 104

Mode: 4 occurs most often, so the Mode is **4**
3, **4, 4**, 5, 104

Grouping

In some cases (such as when all values appear the same number of times) the mode is not useful. But we can **group** the values to see if one group has more than the others.

Example: {4, 7, 11, 16, 20, 22, 25, 26, 33}
Each value occurs once, so let us try to group them.

We can try groups of 10:
0-9: **2 values** (4 and 7)
10-19: **2 values** (11 and 16)
20-29: **4 values** (20, 22, 25 and 26)
30-39: **1 value** (33)

In groups of 10, the "20s" appear most often, so we could choose **25** (the middle of the 20s group) as the mode. You could use different groupings and get a different answer.

Grouping also helps to find what the typical values are when the real world messes things .

Grouping

Example: How long to fill a pallet?

Philip recorded how long it takes to fill a pallet in minutes:
{35, 36, 32, 42, 58, 56, 35, 39, 46, 47, 34, 37}

It takes longer when there is break time or lunch so an average is not very useful.

But grouping by 5s gives:
30-34: **2**
35-39: **5**
40-44: **1**
45-49: **2**
50-54: **0**
54-59: **2**

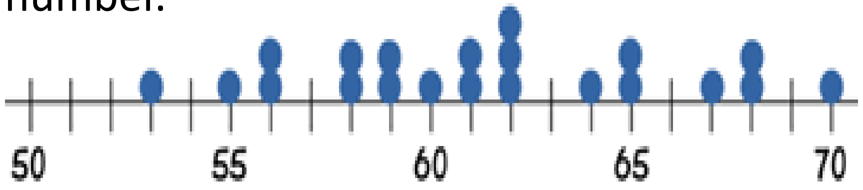
"35-39" appear most often, so we can say it normally takes **about 37 minutes** to fill a pallet.

Mean, Median and Mode

Alex timed 21 people in the sprint race, to the nearest second:
59, 65, 61, 62, 53, 55, 60, 70, 64, 56, 58, 58, 62, 62, 68, 65, 56, 59, 68, 61, 67

To find the **Mean** Alex adds up all the numbers, then divides by how many numbers:
Mean = (*59 + 65 + 61 + 62 + 53 + 55 + 60 + 70 + 64 + 56 + 58 + 58 + 62 + 62 + 68 + 65 + 56 + 59 + 68 + 61 + 67*) / 21
= 61.38095...

To find the **Median** Alex places the numbers in value order and finds the middle number.
In this case the median is the 11th number:
53, 55, 56, 56, 58, 58, 59, 59, 60, 61, 61, 62, 62, 62, 64, 65, 65, 67, 68, 68, 70
Median = 61



To find the **Mode**, or modal value, Alex places the numbers in value order then counts how many of each number. The Mode is the number which appears most often (there can be more than one mode):
53, 55, 56, 56, 58, 58, 59, 59, 60, 61, 61, 62, 62, 62, 64, 65, 65, 67, 68, 68, 70

62 appears three times, more often than the other values, so **Mode = 62**

Mean Deviation

How far, on average, all values are from the middle.

In three steps:

1. Find the mean of all values
2. Find the **distance** of each value from that mean (subtract the mean from each value, ignore minus signs)
3. Then find the **mean of those distances**

Mean Deviation

Example: The Mean Deviation of 3, 6, 6, 7, 8, 11, 15, 16

Step 1: Find the **mean**:

Mean = (*3 + 6 + 6 + 7 + 8 + 11 + 15 + 16)* / **8** = *72/* **8** = 9

Step 2: Find the **distance** of each value from that mean:

Which looks like this:

Value	Distance from 9
3	6
6	3
6	3
7	2
8	1
11	2
15	6
16	7

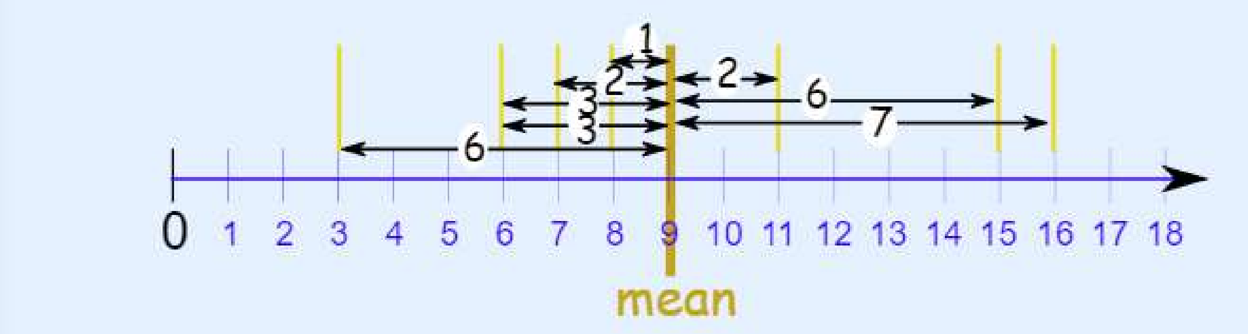
Step 3. Find the **mean of those distances**:

Mean Deviation = (*6 + 3 + 3 + 2 + 1 + 2 + 6 + 7)* **8** = *30 /* **8** = **3.75**

So, the **mean = 9**, and the **mean deviation = 3.75**

It tells us how far, on average, all values are from the middle.

In that example the values are, on average, 3.75 away from the middle.



Mean Deviation

Formula

The formula is:

Mean Deviation = $(\Sigma|x - \mu|) / \mathbf{N}$

Σ is Sigma, which means to sum up

|| (the vertical bars) mean Absolute Value, basically to ignore minus signs

x is each value (such as 3 or 16)

μ is the mean (in our example **μ = 9**)

N is the number of values (**N = 8**)

Example –

The heights (at the shoulders) are: 600mm, 470mm, 170mm, 430mm and 300mm.

Step 1: Find the **mean**:

μ = *600 + 470 + 170 + 430 + 300***5** = *1970***5** = **394**

Step 2: Find the **Absolute Deviations**:

x	|x - μ|
600	206
470	76
170	224
430	36
300	94
Σ|x - μ| = 636	

Step 3. Find the **Mean Deviation**:

Mean Deviation = $\Sigma|x - \mu| / \mathbf{N}$ = *636/5* = 127.2

So, on average, the dogs' heights are **127.2 mm from the mean**.

Range , Quartiles

The **Range** is the difference between the lowest and highest values.

Example: In **{4, 6, 9, 3, 7}**

the lowest value is 3, and
the highest is 9.
So the range is 9 − 3 = **6**.

Example: In **{8, 11, 5, 9, 7, 6, 3616}** :

the lowest value is 5, and
the highest is 3616,

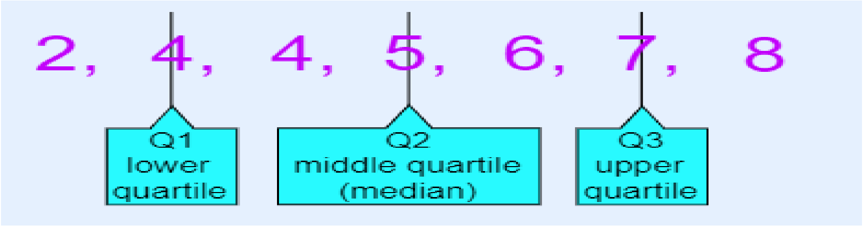
So the range is 3616 − 5 = **3611**.

Quartiles are the values that divide a list of numbers into quarters:

- Put the list of numbers **in order**
- Then cut the list into **four equal parts**
- The Quartiles are at the "cuts"

Example: 5, 7, 4, 4, 6, 2, 8
Put them in order: 2, 4, 4, 5, 6, 7, 8

Cut the list into quarters:



And the result is:

- Quartile 1 (Q1) = **4**
- Quartile 2 (Q2), which is also the Median, = **5**
- Quartile 3 (Q3) = **7**

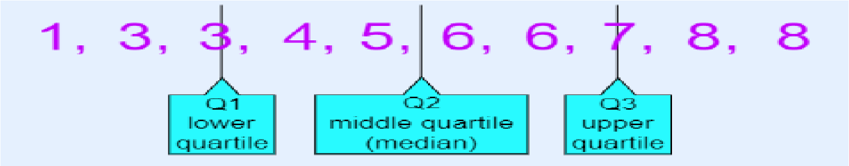
Range , Quartiles, Interquartile Range

Quartiles

Example: 1, 3, 3, 4, 5, 6, 6, 7, 8, 8

The numbers are already in order

Cut the list into quarters:



In this case Quartile 2 is half way between 5 and 6:

Q2 = (5+6)/2 = **5.5**

And the result is:

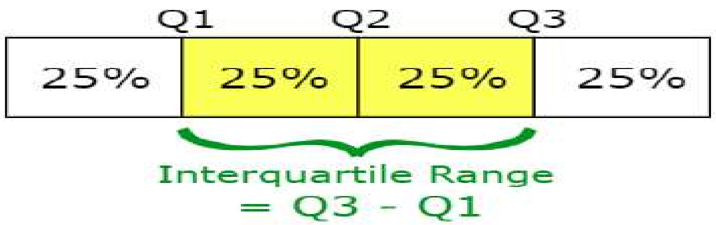
Quartile 1 (Q1) = **3**

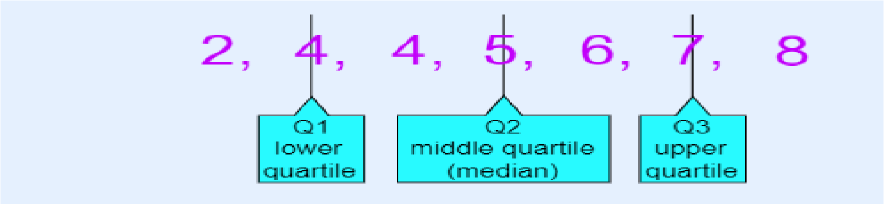
Quartile 2 (Q2) = **5.5**

Quartile 3 (Q3) = **7**

Interquartile Range

The "Interquartile Range" is from Q1 to Q3:



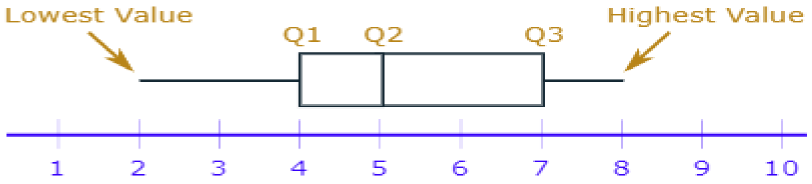


The **Interquartile Range** is:

Q3 − Q1 = 7 − 4 = **3**

Box and Whisker Plot

Important values in a "Box and Whisker Plot":



Example: **Box and Whisker Plot and Interquartile Range** for

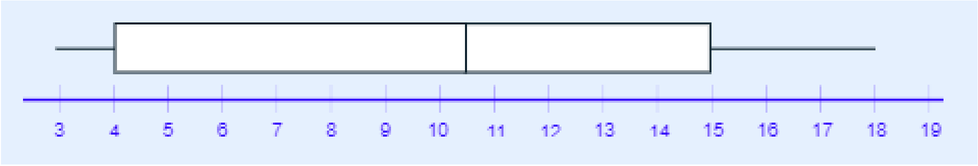
4, 17, 7, 14, 18, 12, 3, 16, 10, 4, 4, 11

Put them in order:
3, 4, 4, 4, 7, 10, 11, 12, 14, 16, 17, 18

Cut it into quarters:
3, 4, 4 | 4, 7, 10 | 11, 12, 14 | 16, 17, 18

In this case all the quartiles are between numbers:
Quartile 1 (Q1) = (4+4)/2 = **4**
Quartile 2 (Q2) = (10+11)/2 = **10.5**
Quartile 3 (Q3) = (14+16)/2 = **15**
Also:
The Lowest Value is **3**, The Highest Value is **18**

Box and Whisker Plot:



And the **Interquartile Range** is:
Q3 − Q1 = 15 − 4 = **11**

Standard Deviation

Standard Deviation

The Standard Deviation is a measure of how spread out numbers are.

Its symbol is **σ** (the greek letter sigma)

The formula : it is the **square root** of the **Variance.** So now you ask, "What is the Variance?“

Variance

The Variance is defined as:
The average of the **squared** differences from the Mean.

To calculate the variance follow these steps:

- Work out the Mean (the simple average of the numbers)
- Then for each number: subtract the Mean and square the result (the *squared difference*).
- Then work out the average of those squared differences.

Formulas

Here are the two formulas, explained at Standard Deviation Formulas if you want to know more:

The "**Population** Standard Deviation": $$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

The "**Sample** Standard Deviation": $$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

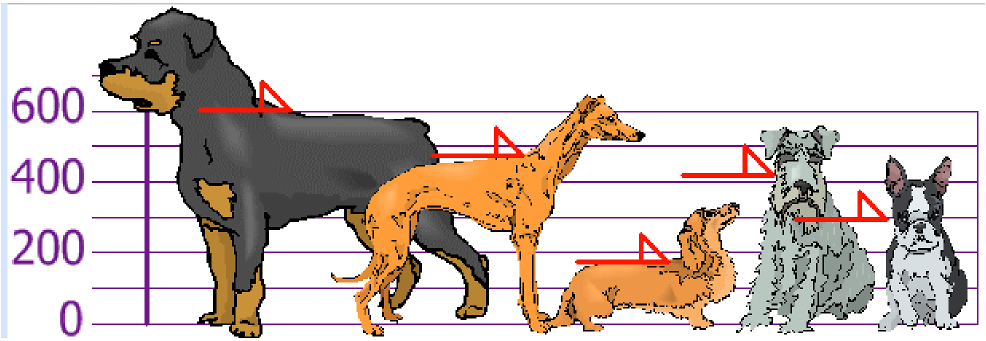
Looks complicated, but the important change is to
divide by **N-1** (instead of **N**) when calculating a Sample Variance.

Standard Deviation

Example

You and your friends have just measured the heights of your dogs (in millimeters):

The heights (at the shoulders) are: 600mm, 470mm, 170mm, 430mm and 300mm.



Find out the Mean, the Variance, and the Standard Deviation.

Your first step is to find the Mean:

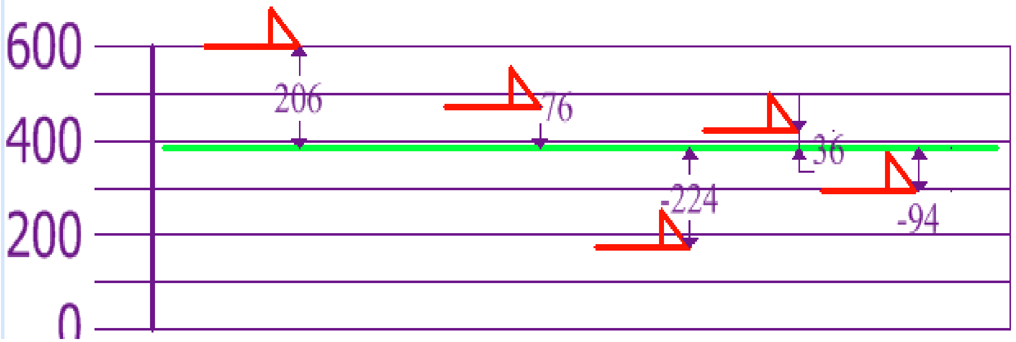
Answer:

Mean=*600 + 470 + 170 + 430 + 300/* **5**

= *1970/***5=**394

so the mean (average) height is 394 mm. Let's plot this on the chart:

Now we calculate each dog's difference from the Mean:



Standard Deviation

To calculate the Variance, take each difference, square it, and then average the result:

Variance

$$\sigma^2 = \frac{206^2 + 76^2 + (-224)^2 + 36^2 + (-94)^2}{5}$$

$$= \frac{42436 + 5776 + 50176 + 1296 + 8836}{5}$$

$$= \frac{108520}{5}$$

$$= 21704$$

So the Variance is **21,704**

And the Standard Deviation is just the square root of Variance, so:

Standard Deviation

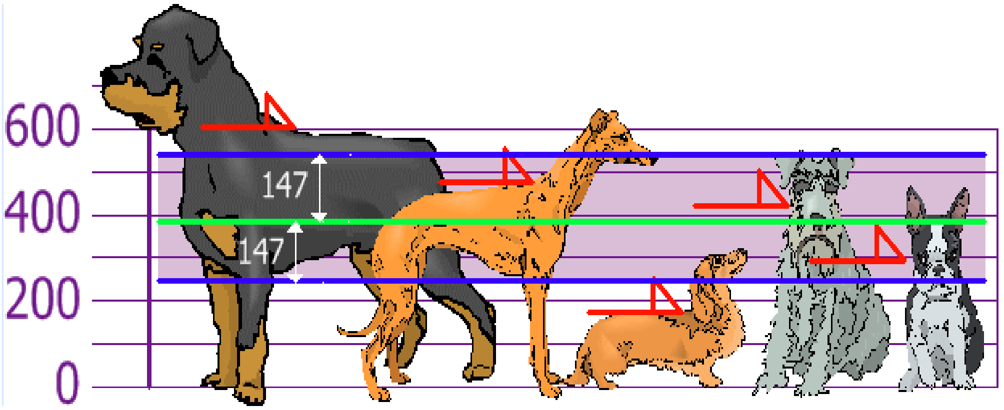
$$\sigma = \sqrt{21704}$$

$$= 147.32...$$

$$= \mathbf{147} \text{ (to the nearest mm)}$$

The Standard Deviation is useful.

Now we can show which heights are within one Standard Deviation (147mm) of the Mean.

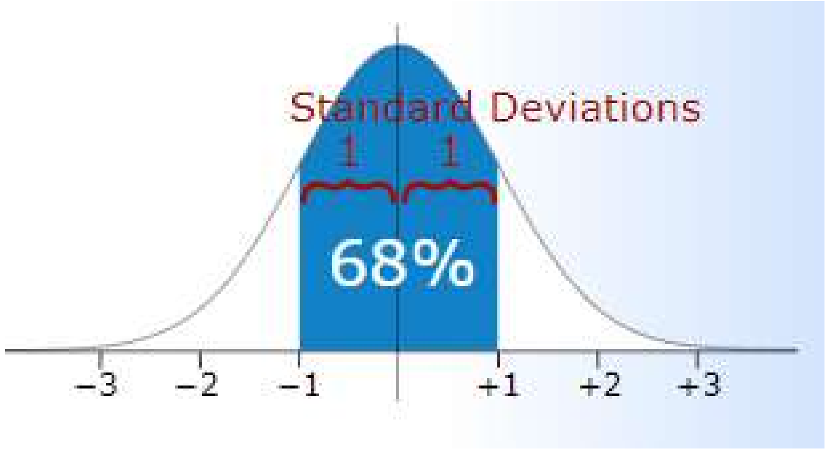


So, using the Standard Deviation we have a "standard" way of knowing what is normal, and what is extra large or extra small.

Rottweilers **are** tall dogs. And Dachshunds **are** a bit short, right?

Standard Deviation

We can expect about 68% of values to be within plus-or-minus 1 standard deviation.



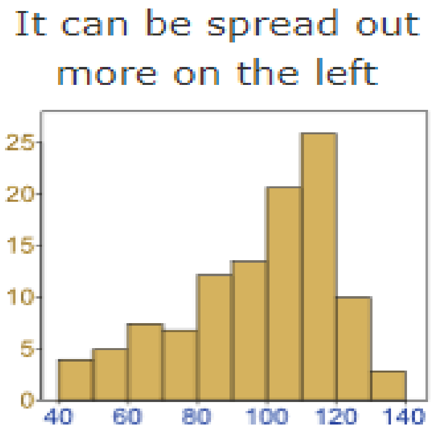
But if the data is a **Sample** (a selection taken from a bigger Population), then the calculation changes!

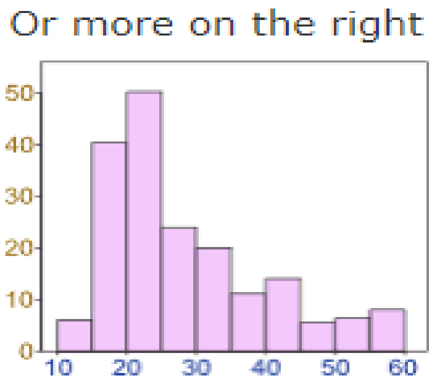
When you have "N" data values :

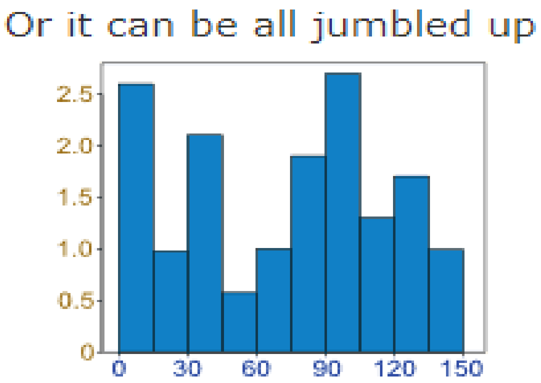
- **The Population**: divide by **N** when calculating Variance
- **A Sample**: divide by **N-1** when calculating Variance

Normal Distribution

Data can be "distributed" (spread out) in different ways.

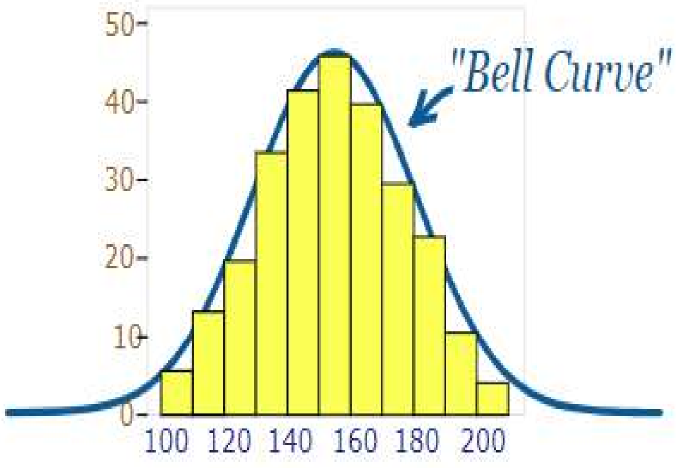






But there are many cases where the data tends to be around a central value with no bias left or right, and it gets close to a "Normal Distribution" like this:

The "Bell Curve" is a Normal Distribution.



A Normal Distribution

Normal Distribution

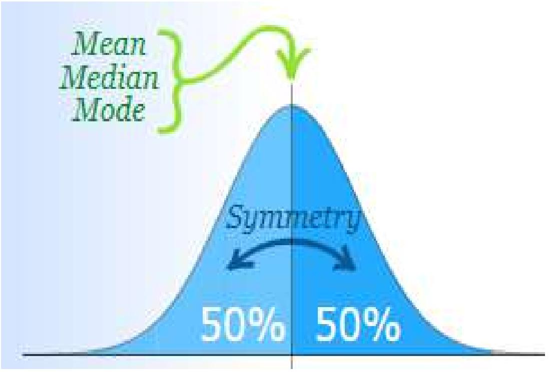
Many things closely follow a Normal Distribution:

• heights of people
• size of things produced by machines
• errors in measurements
• blood pressure
• marks on a test

We say the data is "normally distributed":

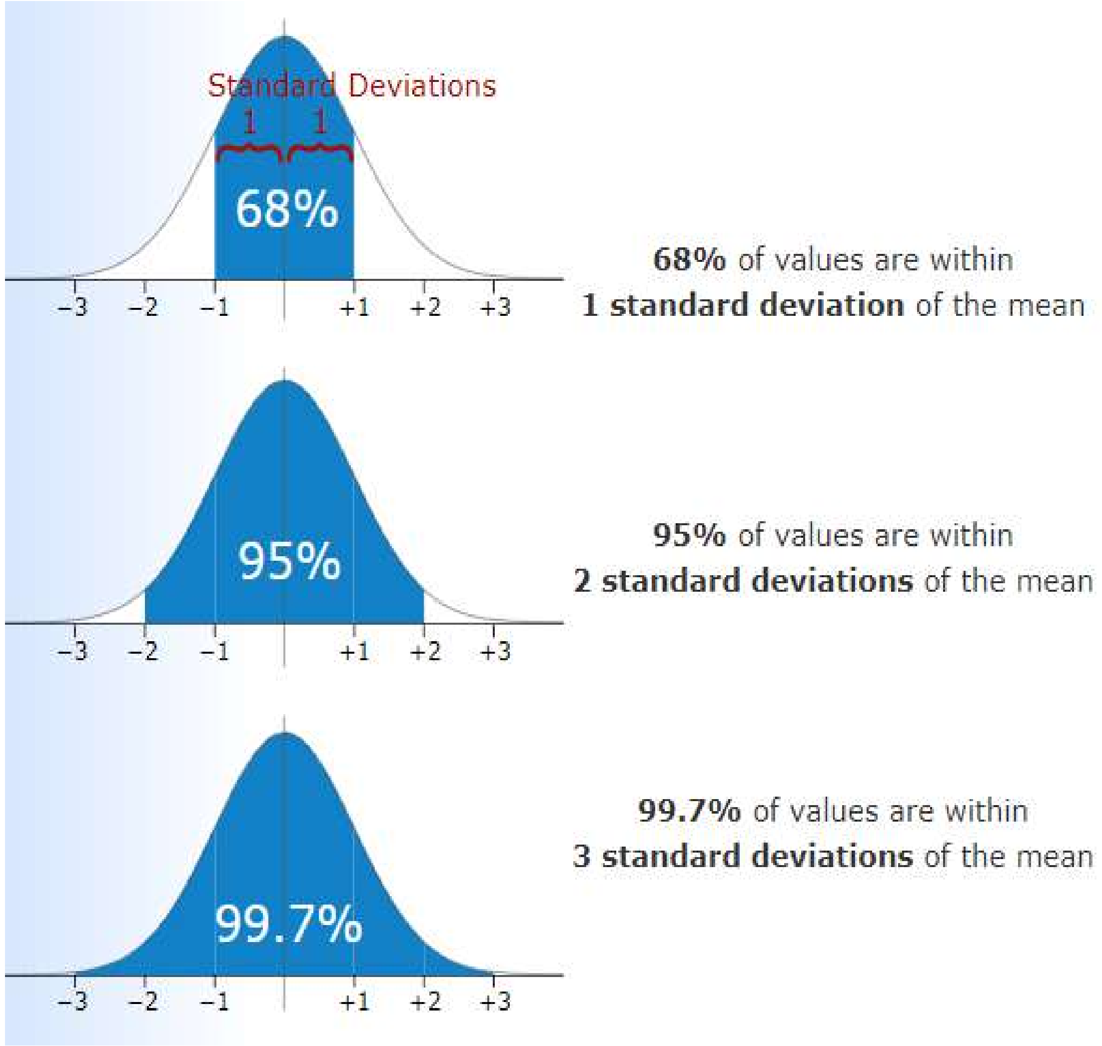
The **Normal Distribution** has:

• mean = median = mode
• symmetry about the center
• 50% of values less than the mean
and 50% greater than the mean



Standard Scores

The number of **standard deviations from the mean** is also called the **"Standard Score", "sigma" or "z-score".**



Contents to be covered

1. Data Science Introduction
2. Data Science Application
3. Data Science –R Libraries
4. Data Science Step Involved
5. Data Preprocessing

Data Science

- Data Science deals with the extraction of knowledge from large set of data, that may be structured or unstructured.
- Large volume of data source are
  - Scientific Experiments
  - Internet of Things
  - E-commerce
  - Financial Transactions,
  - Bank/credit transaction
  - Social Network

Big Data

- Big Data are large and complex data sets which are difficult for traditional methods to store , access and analyze.
- However, lot of potential values are hidden in this data. This makes is valuable for organizations.
- Big Data technologies and approaches are used to drive values out of data in a efficient ways.
- Types of Data
  - Structured – Fields / Tables / columns eg- Spreadsheet, RDBMS
  - Semi-Structured – Tags to separate items eg – XML, HTML
  - Unstructured – No field/attributes. Eg. email , articles

Data Science Fields

- **Artificial Intelligence -** Getting machines to do, what humans are good in.
- **Machine Learning** – Using algorithm to train and predict on data.
- **Deep Learning** – It is a type of **machine learning** that has networks capable of **learning** unsupervised data that is unstructured or unlabeled. Also known as **deep** neural **learning** or **deep neural network**.

Data Science Applications

- **Business** – From car design to pizza delivery, businesses are using data science to optimize their operations and better meet their customer expectations.
- **Health Care-** Health care units are managing the electronic health records, which helps in better point-of-care decisions.
- **Urban Leaving** – Urban informatics deals with challenges, world facing due to growing cities. e.g Traffic Management
- **Web Search Engine:** One of the reasons why search engines like google, bing etc work so well is because the system has learnt how to rank pages through a complex learning algorithm.
- **Photo tagging Applications-** Be it facebook or any other photo tagging application, the ability to tag friends makes it even more happening. It is all possible because of a face recognition algorithm that runs behind the application.
- **Spam Detector-**Our mail agent like Gmail or Hotmail does a lot of hard work for us in classifying the mails and moving the spam mails to spam folder. This is again achieved by a spam classifier running in the back end of mail application.

Data Science - Steps

1. **Data Gathering -** Put the necessary systems in place to gather data. Data collected may be fragmented and scattered.
2. **Data Preparation -** Clean and format the messy data sets, to make it usable for analysis. This includes managing missing values, error in data collection, data formatting, normalization.
3. **Exploration** - Understand the structure of data, by using the clustering algorithms and visualizing methods - scatter plot, bar graphs.
4. **Model Building** - Explore the variety of models (Random Forest, SVM, Neural Network, K-nearest Neighbors etc) on the data sets and identify and develop the model which fit for the problem.
5. **Model Validation** – Analyze the prediction accuracy of the model based on evaluation matrices.
6. **Model Deployment** – Deploy the model. Tweak and improve it based on feedback.

NIELIT

Data Preparation

Dataset - **Filename – property_data.csv**

	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000
1	100002000.0	197.0	LEXINGTON	N	3	1.5	--
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850
3	100004000.0	201.0	BERKELEY	12	1	NaN	700
4	NaN	203.0	BERKELEY	Y	3	2	1600
5	100006000.0	207.0	BERKELEY	Y	NaN	1	800
6	100007000.0	NaN	WASHINGTON	NaN	2	HURLEY	950
7	100008000.0	213.0	TREMONT	Y	--	1	NaN
8	100009000.0	215.0	TREMONT	Y	na	2	1800

Data Preparation - Steps

Find out the following questions:

- What are the features?
- What are the expected types (int, float, string, boolean)?
- Standard Missing data ?
- Non Standard Missing Data ?

Data Preparation – Loading Dataset

PropData=read.csv('d:/pyprg/19-property_data.csv')
print(PropData)

```
> PropData=read.csv('d:/pyprg/19-property_data.csv')
> print(PropData)
        PID ST_NUM    ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH SQ_FT
1 100001000    104     PUTNAM            Y            3        1  1000
2 100002000    197  LEXINGTON            N            3      1.5    --
3 100003000     NA  LEXINGTON            N          n/a        1   850
4 100004000    201   BERKELEY           12            1      NaN   700
5        NA    203   BERKELEY            Y            3        2  1600
6 100006000    207   BERKELEY            Y         <NA>        1   800
7 100007000     NA WASHINGTON                         2   HURLEY   950
8 100008000    213    TREMONT            Y            1        1
9 100009000    215    TREMONT            Y           na        2  1800
>
```

Data Preparation - Dataset Features

dim(PropData)

nrow(PropData)

ncol(PropData)

str(PropData)

```
> dim(PropData)
[1] 9 7
> nrow(PropData)
[1] 9
> ncol(PropData)
[1] 7
> str(PropData)
'data.frame':   9 obs. of  7 variables:
 $ PID         : int  100001000 100002000 100003000 100004000 NA 10000
 $ ST_NUM      : int  104 197 NA 201 203 207 NA 213 215
 $ ST_NAME     : chr  "PUTNAM" "LEXINGTON" "LEXINGTON" "BERKELEY" ...
 $ OWN_OCCUPIED: chr  "Y" "N" "N" "12" ...
 $ NUM_BEDROOMS: chr  "3" "3" "n/a" "1" ...
 $ NUM_BATH    : chr  "1" "1.5" "1" "NaN" ...
 $ SQ_FT       : chr  "1000" "--" "850" "700" ...
```

what are my features?

ST_NUM : Street number

ST_NAME : Street name

OWN_OCCUPIED : Is the residence owner occupied

NUM_BEDROOMS : Number of bedrooms

Data Preparation - Dataset Features

head(PropData,3)
tail(PropData,3)
colnames(PropData)
rownames(PropData)

```
> head(PropData,3)
        PID ST_NUM    ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH SQ_FT
1 100001000    104     PUTNAM            Y            3        1  1000
2 100002000    197 LEXINGTON            N            3      1.5    --
3 100003000     NA LEXINGTON            N          n/a        1   850
> tail(PropData,3)
        PID ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH SQ_FT
7 100007000     NA WASHINGTON                          2   HURLEY   950
8 100008000    213    TREMONT            Y            1        1
9 100009000    215    TREMONT            Y           na        2  1800
> colnames(PropData)
[1] "PID"          "ST_NUM"       "ST_NAME"      "OWN_OCCUPIED" "NUM_BEDROOMS" "NUM_BATH"     "SQ_FT"
> rownames(PropData)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9"
```

Data Preparation - Dataset Features

summary(PropData)

```
> summary(PropData)
      PID               ST_NUM          ST_NAME            OWN_OCCUPIED       NUM_BEDROOMS        NUM_BATH            SQ_FT          
 Min.   :1e+08   Min.   :104.0   Length:9           Length:9           Length:9           Length:9           Length:9          
 1st Qu.:1e+08   1st Qu.:199.0   Class :character   Class :character   Class :character   Class :character   Class :character  
 Median :1e+08   Median :203.0   Mode  :character   Mode  :character   Mode  :character   Mode  :character   Mode  :character  
 Mean   :1e+08   Mean   :191.4                                                                                                 
 3rd Qu.:1e+08   3rd Qu.:210.0                                                                                                 
 Max.   :1e+08   Max.   :215.0                                                                                                 
 NA's   :1       NA's   :2                                                                                                     
```

Data Preparation - Expected Data Types

what are the expected types?

ST_NUM : float or int… some sort of numeric type

ST_NAME : string

OWN_OCCUPIED : string… Y (“Yes”) or N (“No”)

NUM_BEDROOMS : float or int, a numeric type

	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000
1	100002000.0	197.0	LEXINGTON	N	3	1.5	--
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850
3	100004000.0	201.0	BERKELEY	12	1	NaN	700
4	NaN	203.0	BERKELEY	Y	3	2	1600

Data Preparation - Standard Missing Data

Standard Missing data ?

 # Finding Missing data in ST_NUM Column

print(PropData$ST_NUM)

is.na(PropData$ST_NUM)

```
> print(PropData$ST_NUM)
[1] 104 197  NA 201 203 207  NA 213 215
> is.na(PropData$ST_NUM)
[1] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
>
```

Standard Missing Data

NA

NaN

रा.इ.सू.प्रौ.सं NIELIT

Data Preparation - Standard Missing Data

Standard Missing data ?

is.na(PropData)

which(is.na(PropData$ST_NUM))

```
> is.na(PropData)
       PID ST_NUM ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH SQ_FT
 [1,] FALSE  FALSE   FALSE        FALSE        FALSE    FALSE FALSE
 [2,] FALSE  FALSE   FALSE        FALSE        FALSE    FALSE FALSE
 [3,] FALSE   TRUE   FALSE        FALSE        FALSE    FALSE FALSE
 [4,] FALSE  FALSE   FALSE        FALSE        FALSE    FALSE FALSE
 [5,]  TRUE  FALSE   FALSE        FALSE        FALSE    FALSE FALSE
 [6,] FALSE  FALSE   FALSE        FALSE         TRUE    FALSE FALSE
 [7,] FALSE   TRUE   FALSE        FALSE        FALSE    FALSE FALSE
 [8,] FALSE  FALSE   FALSE        FALSE        FALSE    FALSE FALSE
 [9,] FALSE  FALSE   FALSE        FALSE        FALSE    FALSE FALSE
> which(is.na(PropData$ST_NUM))
[1] 3 7
> 
```

Standard Missing Data

NA

NaN

Data Preparation - Standard Missing Data

Standard Missing data ?

apply(is.na(PropData), 2 , which)

colnames(PropData) [apply(PropData,2,anyNA)]

```
> apply(is.na(PropData),2, which)
$PID
[1] 5

$ST_NUM
[1] 3 7

$ST_NAME
integer(0)

$OWN_OCCUPIED
integer(0)

$NUM_BEDROOMS
[1] 6

$NUM_BATH
integer(0)

$SQ_FT
integer(0)

> colnames( PropData) [ apply(PropData,2,anyNA)]
[1] "PID"           "ST_NUM"        "NUM_BEDROOMS"
>
```

Standard Missing Data

NA

NaN

Data Preparation - Non Standard Missing Data

Non Standard Missing data ?

 # Finding Missing data in ST_NUM Column

print(PropData$NUM_BEDROOMS)

is.na(PropData$NUM_BEDROOMS)

Non standard Missing Data

n/a
NAN
—
na

```
> print(PropData$NUM_BEDROOMS)
[1] "3"   "3"   "n/a" "1"   "3"   NA    "2"   "1"   "na"
> is.na(PropData$NUM_BEDROOMS)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
>
```

Data Preparation - Summarizing Missing Value

Summarizing Missing Values

 # Total missing values for each Column

which(is.na(PropData))

apply(is.na(PropData) , 2, which)

#Total number of complete Rows

sum(complete.cases(PropData))

```
> which(is.na(PropData))
[1]  5 12 16 42
> apply(is.na(PropData),2, which)
$PID
[1] 5

$ST_NUM
[1] 3 7

$ST_NAME
integer(0)

$OWN_OCCUPIED
integer(0)

$NUM_BEDROOMS
[1] 6

$NUM_BATH
integer(0)

$SQ_FT
integer(0)

> sum(complete.cases(PropData))
[1] 5
>
```

Data Preparation - Summarizing Missing Value

Summarizing Missing Values

 # Total missing values for each Column
sum(is.na(PropData$ST_NUM))

#Total number of Missing Value
sum(is.na(PropData))

colSums(is.na(PropData))

```
> sum(is.na(PropData$ST_NUM))
[1] 2
> sum(is.na(PropData))
[1] 4
> colSums(is.na(PropData))
         PID       ST_NUM      ST_NAME OWN_OCCUPIED NUM_BEDROOMS     NUM_BATH        SQ_FT
           1            2            0            0            1            0            0
> |
```

Data Preparation - Replace Missing Value

Imputation – To replace the missing value with mean/median/mode of all the value for that column.

Replacing Missing Values

```
# Replace missing values with a number
PropData$ST_NUM[ is.na(PropData$ST_NUM)]=0
PropData[ is.na(PropData) ]=0

replace( PropData$ST_NUM ,  is.na(PropData$ST_NUM) , 100 )
```

Data Preparation - Replacing Values

Replacing Missing Values

replace(PropData$OWN_OCCUPIED , PropData$OWN_OCCUPIED=='12‘ , ' Y‘)

Remove the Missing Values

na.omit(PropData$NUM_BEDROOMS)
na.omit(PropData)

Unique Values of Column

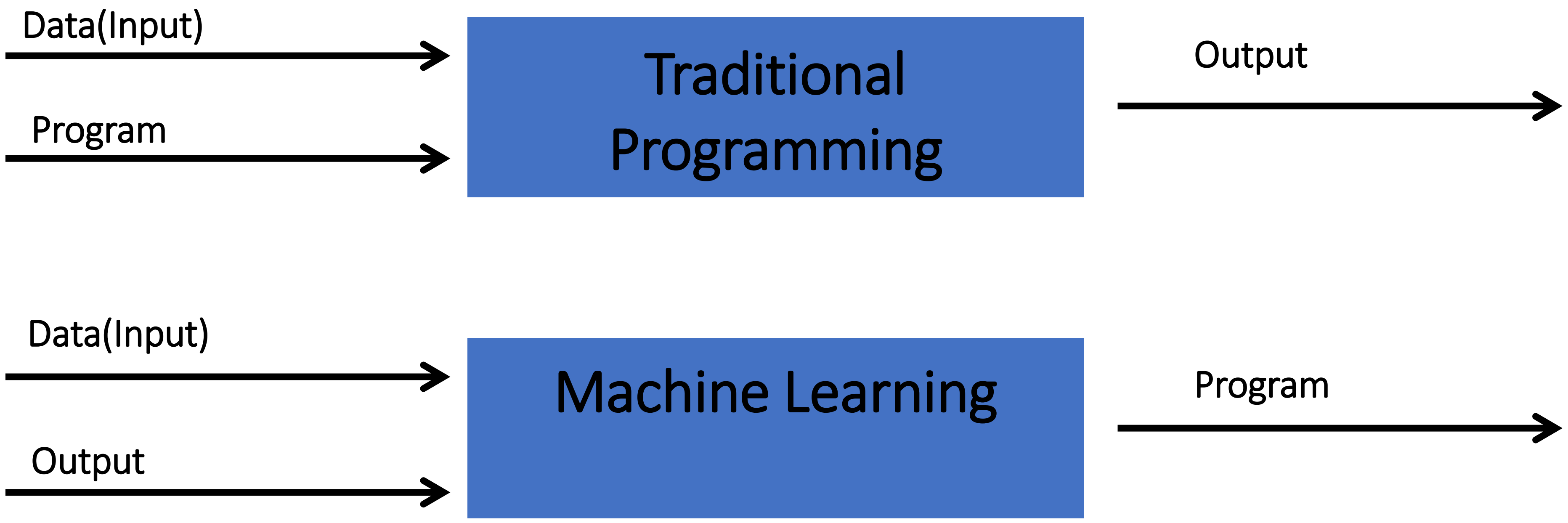
unique(PropData$NUM_BEDROOMS)

Contents to be covered

1. Machine Learning
2. Machine Learning Classification
3. Machine Learning Working
4. Machine Learning Application

Machine Learning

- **Traditional Programming** : We feed in DATA (Input) + PROGRAM (logic), run it on machine and get output.
- **Machine Learning** : We feed in DATA(Input) + Output, run it on machine during training and the machine creates its own program(logic), which can be evaluated while testing.



राइसूप्रौसं NIELIT

Machine Learning - Classification

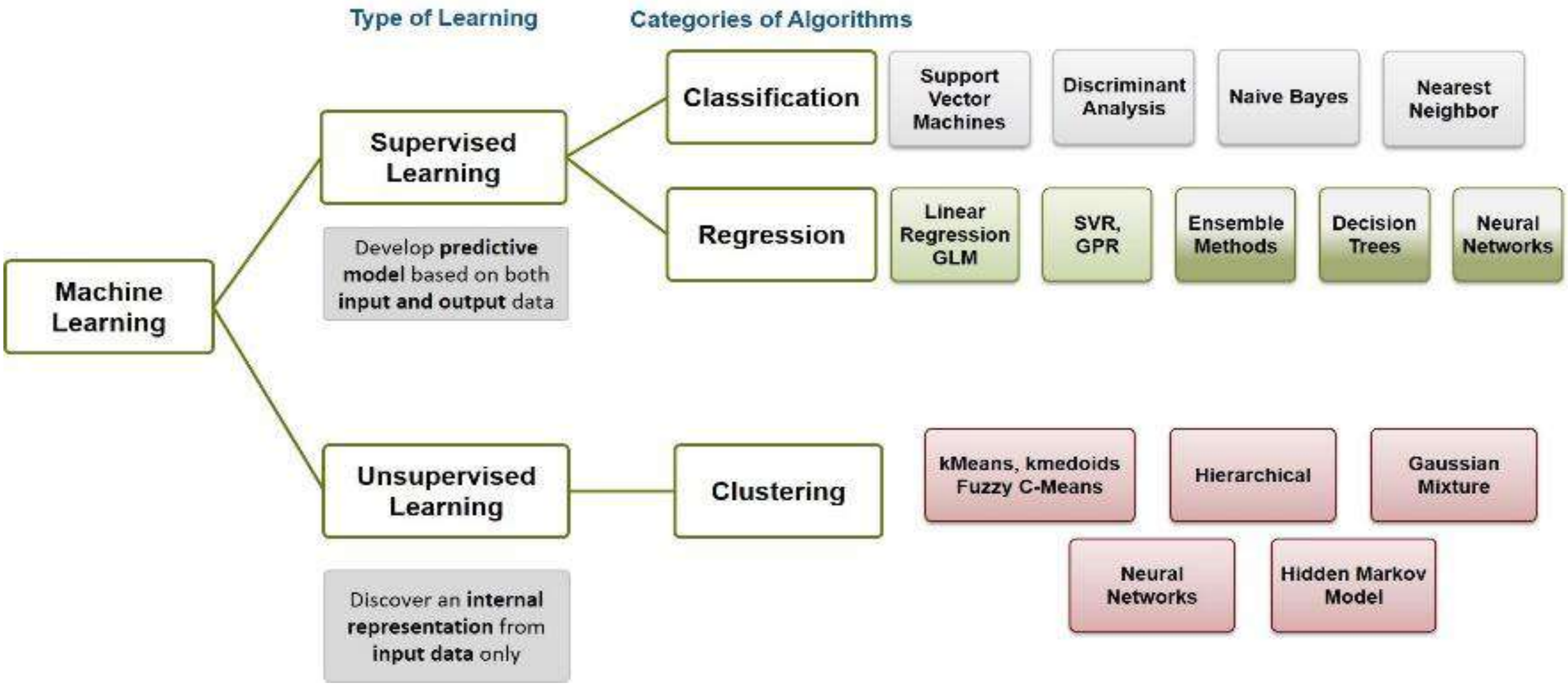
- Machine learning implementations are classified into three major categories, depending on the nature of the learning “signal” or “response” available to a learning system which are as follows:-

  1. Supervised learning
  2. Unsupervised learning
  3. Reinforcement learning

Machine Learning - Classification

1. **Supervised learning –**
   - When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples.
   - The teacher provides good examples for the student to memorize, and the student then derives general rules from these specific examples.
2. **Unsupervised learning**
   - When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own.
   - It resembles the methods humans use to figure out that certain objects or events are from the same class, by observing the degree of similarity between objects.
3. **Reinforcement learning –**
   - When you present the algorithm with examples that lack labels, as in unsupervised learning. However, you can provide an example with positive or negative feedback according to the solution . In the human world, it is just like learning by trial and error.
   - Errors help you learn because they have a penalty added (cost, loss of time, regret, pain, and so on), teaching you that a certain course of action is less likely to succeed than others.

Category of Algorithms



राइसूप्रौसं
NIELIT

Machine Learning - Working

- Divide the input data into
  - training,
  - cross-validation and
  - test sets.
- The ratio between the respective sets must be 6:2:2
- Building models with suitable algorithms and techniques on the training set.
- Testing our conceptualized model with data which was not fed to the model at the time of training and evaluating its performance using metrics such as F1 score, precision and recall.

Machine Learning - Implementation

#Loading the CSV File into DataFrame

```
require(caTools)

PersonData=read.csv('d:/pyprg/20-htwgtMale.csv' , header=TRUE)

print(PersonData)
str(PersonData)
```

```
> str(PersonData)
'data.frame':   38 obs. of  2 variables:
 $ Height: int  105 110 119 120 132 138 140 152 156 157 ...
 $ Weight: int  12 10 21 28 33 29 42 46 60 95 ...
```

	Height	Weight
0	105	12
1	110	10
2	119	21
3	120	28
4	132	33

Machine Learning - Implementation

Extracting the required Columns from the DataFrame

```
#Returns a Height of the DataFrame.
Height=PersonData$Height

# Returns a Weight of the DataFrame.
Weight=PersonData$Weight

cat('type(Height):',  class(Height))
print('Height :', length(Height), 'Weight :', length(Weight))
```

Type integer

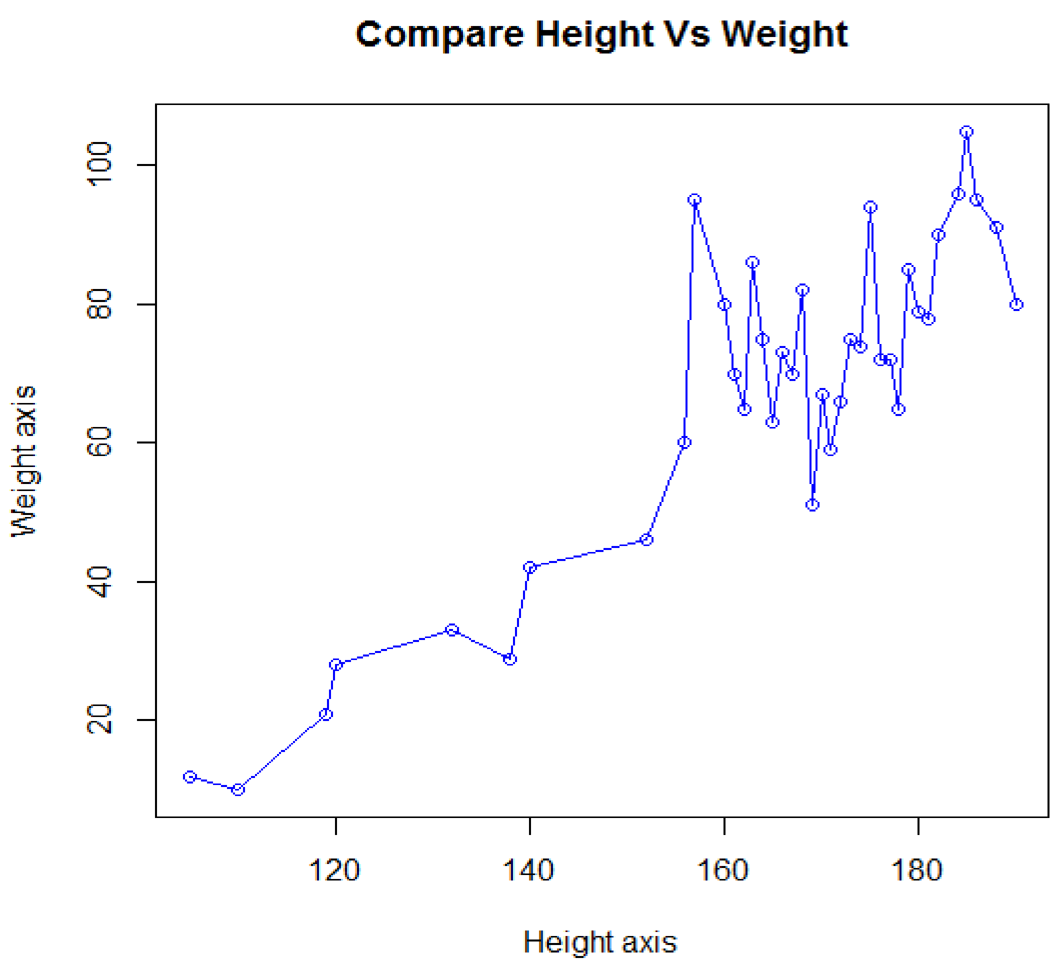
Height : 38   Weight : 38

Machine Learning - Implementation

Plotting the Trend in the Data

#Plot the Trend of Original Data

plot(Height , Weight , type='o', xlab="Height axis", ylab="Weight axis", main="Compare Height Vs Weight", col="blue")



Machine Learning - Implementation

Splitting the Data for Training & Testing

#Split the Data into Training & Validation Data

```
require(caTools)
sample=sample.split(Weight, SplitRatio=0.75)
trainY=subset(Weight, sample==TRUE)
testY=subset(Weight, sample==FALSE)

trainX=subset(Height, sample==TRUE)
testX=subset(Height, sample==FALSE)

cat ( 'Weight :',  length(trainY),  length(testY) )
cat ( 'Height :',  length(trainX),  length(testX) )
```

Data Original Shape

Height : 38   Weight : 38

Spitted Data

Weight : 28   10

Height : 28   10

Machine Learning - Implementation

Training the Model - LinearRegression

```
MyModel.lm=lm(formula=trainX~trainY)
#                                H  ~  W
#Print the Coefficients
print(coefficients( MyModel.lm))

#Prediction
newdata=data.frame( trainY=c(72,85))
predict(MyModel.lm , newdata)

newdataY=data.frame( trainY=testY)
p=predict(MyModel.lm , newdataY)
comp=data.frame( testY , testX, p)
print(comp)
```

```
coefficients( MyModel.lm))
(Intercept)     trainY
109.3278075   0.7934524

72                85
166.4564      176.7713
```

	testY	testX	p
1	10	110	117.2623
2	65	162	160.9022
3	63	165	159.3153
4	73	166	167.2498
5	67	170	162.4891
6	75	173	168.8367
7	94	175	183.9123
8	72	177	166.4564
9	65	178	160.9022
10	90	182	180.7385

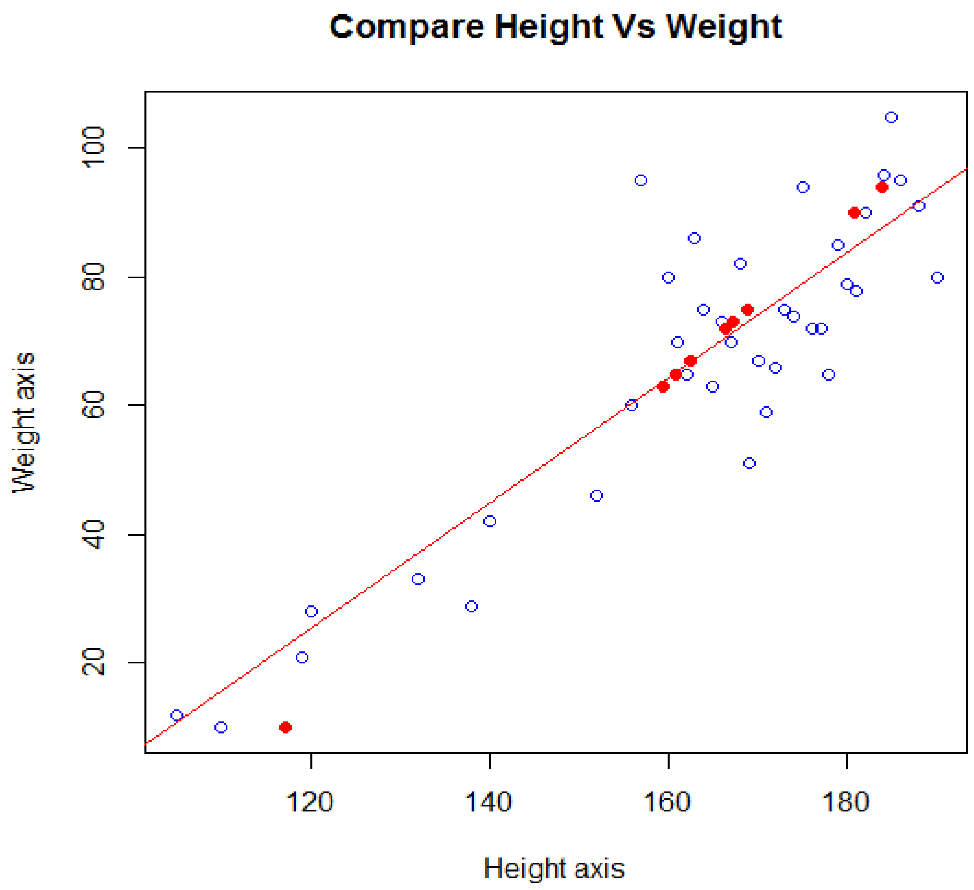
रा.इ.सू.प्रौ.सं NIELIT

Machine Learning - Implementation

Plotting the Model - LinearRegression

#We can plot X vs y using the equation of the form y=m*x+c

```
#Print the regression Line
plot( Height,Weight, xlab="Height axis", ylab="Weight axis",
main="Compare Height Vs Weight", col="blue")

my.fit= lm(trainY~trainX)
summary(my.fit)
abline(my.fit, col='red')

#point( Y, X) to add point s in graph
points(p, testY, pch=19, col="red")
```



Accuracy of the Model - LinearRegression

#R-squared is a statistical measure of how close the data are to the fitted regression line.

```
summary( my.fit  )

Call:
lm(formula = trainY ~ trainX)

Residuals:
    Min      1Q  Median      3Q     Max
-22.226  -7.218  -1.022   5.550  33.486

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -91.7199    16.8942  -5.429 1.09e-05 ***
trainX        0.9760     0.1033   9.448 6.83e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12.07 on 26 degrees of freedom
Multiple R-squared:  0.7744,    Adjusted R-squared:  0.7657
F-statistic: 89.26 on 1 and 26 DF,  p-value: 6.829e-10
```

R Programming

Day-1

Q.1 MCQs

1. In 1991, R was created by Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of _________.
    a. Auckland
    b. Harvard
    c. California
    d. John Hopkins

2. How many types of basic data types are present in R?
    a. 4
    b. 5
    c. 6
    d. 7

3. What is the output of typeof(10)?
    a. Numeric
    b. Double
    c. Integer
    d. Logical

4. Descriptive analysis tells about________?
    a. Past
    b. Present
    c. Future
    d. Wrong!

5. ___________ is used to make predictions about unknown future events?
    a. Descriptive analysis
    b. Predicative analysis
    c. Both the above
    d. Correct!

Q.2 Write a program to calculate and print the sum of 3 numbers. [Take the input from User]

Q.3 Write a program to calculate and print the Simple Interest , if the principal , ROI and Time is given.

SI = Principal * ROI * Time / 100

Q.4 Write a program to calculate and print the area of a circle.

A = 3.14 * radius * radius [Input Radius from User]

Q.5 Write a program to read Length and Width of the rectangle and print the area and perimeter of the rectangle.

Area = L * W Perimeter = 2 * (L + W)

Q.6 Read Name and Age of the person, print the following output

Your Name is ……………………….

Your Age is ……………………………

Q.7 What is the output :

a. 2 * 2 *2 /2 + 5

b. 2 * 2 * 2 / (2 +5)

c. (2 * 2 * 2) / 2 + 5

d. (2 * 2 * 2 /2) + 5

R Programming
Day-2

Q.1 MCQ

1. Which is an invalid Literal?
   a. Test
   b. Numeric
   c. Integer
   d. Character

2. Which is an invalid Character Literal?
   a. "This is Python"
   b. "P"
   c. 'Sum'
   d. #Sum#

3. Output of print(0x1a) is ___________.
   a. 0x1a
   b. error
   c. 26
   d. 28

4. Which is an invalid data type?
   a. Numeric
   b. Double
   c. Integer
   d. Character

5. Which is invalid Constant Name?
   a. MYVAR
   b. MY.VAR
   c. MY_VAR
   d. _MY_VAR

Q.2 Write a program to calculate and print the sum of 3 non-integer numbers.

[Take the input from User]

Q.3 Write a program to calculate and print average of 3 numbers.

[Take the input from User]

Q.4. What is the output of following?

a)	y = 2.8 b = as.integer(y) print(b)	b)	x = "10" y=as.integer(x) + 10 print(y)
c)	x=10.73 y=as.integer(x) print(x+y)	d)	x=10E2 y=10+as.integer(x) print(y)

Q.5. What is the output of following?

a) m1=text1='hello user' print(m1)	b) msg='welcome to NIELIT' print(msg)
c) m1=text1='Apple Mango' cat('You like the fruit - : ', m1)	d) fruit='Mango' print(fruit + 100)

Q.6 Write program to exchange the values of two numbers without using a temporary variable.

Q.7 Write program to calculate gross salary where gross salary=Basic+HRA+DA

In this HRA is 16% of Basic, DA is 12% of Basic

Q.8 What is the output ?

a). a=c(10,20,30) b=a+2 print(b)	b). a=c(10,TRUE,30, FALSE) b=a+2 print(b)
c). a=c(10, 'Ajay', FALSE) b=a+2 print(b)	d). a=c(10, 'Ajay', FALSE) b=length(a) print(b)

Day-03:

Q.1 Multiple Choice Question

1. Which is an invalid Arithmetic Operator?
   a. +
   b. -
   c. ^
   d. %

2. Which is an output 10 %/% 3?
   a. error
   b. 3
   c. 3.3
   d. 1

3. Which is an invalid Relational Operator ?
   a. <
   b. =
   c. !=
   d. ==

4. Which is invalid Logical Operator ?
   a. &&
   b. ||
   c. &
   d. !!

5. Which is an invalid Assignment Operator?
   a. ->
   b. ->>
   c. ==
   d. =

Q.2. What is the output of following?

a) x = 6/3/2 print(x)	b) x=2 a = x * -5 b = x ^ -5 print(a) print(b)
c) x = (7 + 3) * 10 / 5 print(x)	**d)** a = 27 // 5 b = 27/ 5 print(a) print(b)

Q.3. What is the output of following?

<table>
<tr><td>a) x = (10 < 5) && ((5 / 0) < 10)
print(x)</td><td>b) x = (10 > 5) && ((5 / 0) < 10)
print(x)</td></tr>
<tr><td>c) x = ! (10 < 5) && ((5 / 0) < 10)
print(x)</td><td>d) x = (10 > 5) || ((5 / 0) < 10)
print(x)</td></tr>
</table>

Q.4. Evaluate the following Expression.

a) 5 %% 10 + 10 < 50 && 29 >= 29
b) 7 ^ 2 <=5 %/% 9 %% 3
c) 5 %% 10 < 10 && –25 > 1 * 8 %/% 5
d) 7 ^ 2 // 4 + 5 > 8 || 5 != 6
e) 10 + 6 * 2 ^ 2 ! = 9 %/% 44 -3 && 29 >= 29/9

Q.5. Construct logical expression for representing the following conditions:

a) **marks** scored should be greater than 300 and less than 400.

b) Whether the value of **grade** is an upper case letter.

c) The **post** is engineer and **experience** is more than four years.

Q.6 Write a program to print the sum of digits of 3 digit number.

Q.7 Write a program to print the reverse of a 3 digit number.

Q.8 Write a program to extract the odd numbers from the vector

A=c(10,11,12,13,14,15,16)

Result – 11,13, 15

Q.9 Write a program to check how many Ajay exist in vector
A=c('Ajay', 'Vijay', 'Ajay', 'Sanjay', 'Vikas', 'Ajay')

Result= 3

Day-04:

Q.1 Multiple Choice Question

1. R is an__________programming language?
   a. Closed source
   b. GPL
   c. Open source
   d. Definite source

2. Which is not true for IF statement ?
   a. Else is required part of if statement.
   b. Ifelse( ) is also known as compact if.
   c. Nested IF statements are allowed
   d. Ladder IF…ELSE IF… can be used.

3. What is the output of - paste("a", "b", sep = ":")
   a. "a+b"
   b. "a=b"
   c. "a:b"
   d. a*b

4. If you explicitly want an integer, you need to specify the _____ suffix.
   a) D
   b) R
   c) L
   d) K

5. Numbers in R are generally treated as _______ precision real numbers.
   a) single
   b) double
   c) real
   d) imaginary

Q.2 Write a program to display the square and cube of a positive number.

Q.3 Write a program to display the greater of 2 numbers.

Q.4 Write a program to check an entered number is Odd or Even. [**hint** – use % modulus operator to determine the remainder]

Q.5 Write a program to check an entered number is divisible by 7 or not.

Q.6 Write a program to check greatest among three numbers.

Q.7 Write a program to input the number and check it is divisible 3 and 5.

Q.8 Write a program to check a number is positive or negative.

Q.9 Write a program to check a year leap year or not.

Q.10 Write a program to check a three digit number is palindrome or no.

[Hint – use the % operator fragment the digit

123%10 = 3

12 % 10 = 2

1 % 10 = 1

Q.11 Write a program to input the cost price and selling price of an item and check for profit or loss. Also calculate it.

Q.12 In an examination, the grades are awarded to the students in 'SCIENCE' according to the average marks obtained in the examination.

Marks	Grades
80% and above	Distinction
60% or more but less than 80%	First Division
45% or more but less than 60%	Second Division
40% or more but less than 45%	Pass
Less than 40%	Promotion not granted

Write a program to input marks in Physics , Chemistry and Biology. Calculate the average marks. Display the average marks and grade obtained.

Day-5

Q.1 Multiple Choice Question

1. Which one is not allowed with While Loop?
    a. Test expression
    b. break
    c. continue
    d. next

2. Which one is not allowed with Repeat Loop?
    a. Test expression
    b. If statement in body
    c. break
    d. next

3. Which is true about the BREAK statement with While Loop?
    a. It skips the remaining body of the loop.
    b. It repeats the body of the loop.
    c. It is performs nothing.
    d. It transfers the flow outside the body of loop.

4. Which is true about the NEXT statement with While Loop?
    a. It skips the remaining body of the loop, jumps to the beginning, for current iteration.
    b. It repeats the body of the loop.
    c. It is performs nothing.
    d. It transfers the flow outside the body of loop.

5. **readline**( ) function return data type is
    1. integer
    2. decimal
    3. bool
    4. character

6. Which is not true about the **print**( ) function?
    a. It is an in-built function.
    b. It prints the output on screen.
    c. It can print multiple variable on screen.
    d. It applies the new line character at the end.

Q. 2 Write the program to display the first 10 terms of the following series :

a. 1 , 3, 5,…………
b. 2 , 4 , 6 …………
c. 1 , 4 , 9 , 16…….
d. 1.5 , 3.0 , 4.5 , 6.0 ……..
e. -5 , -10 , -15, -20 ……….

Q.3 Write a program to calculate and display the sum of all odd numbers and even numbers between a range of numbers from m to n where m < n. Input m and n.

Q.4 Write a program to print the 10 multiples of any entered number.

Q.5 Write a program to display the sum of 10 natural numbers.

Q.6 Write a program to calculate and display the factorial of an entered number.

Q.7 Write a program to count the vowels in entered string.

```
n='computer'
x=unlist(c(strsplit(n , NULL)))

c=0
i=1
while ( i <= length(x) )
{
  if ( x[i] %in%  c('a','e','i','o','u') )
   { c=c+1 }

  i=i+1
}

cat('Vowels : ',c)
```

Q.8 What is the output

1. n='computer' strsplit(n,NULL)	2. n='c,o,m,p,u,t,e,r' strsplit(n, ',')	3. n='Vijay.Ajay.Vikas' strsplit(n, '\\.')	4. n='computer' strsplit(n,NULL)[[1]]
5. n='Vijay.Ajay.Vikas' strsplit(n, '.')	6. n=c('a','e','i','o','u') paste(n,sep='')	7. n=c('a','e','i','o','u') paste(n,collapse='')	

Day-6

Q.1 Multiple Choice Question

1. Which one is not allowed with for Loop?
   a. If statement
   b. break
   c. continue
   d. next

2. Which is true about the **Break** statement with for Loop?
   a. It skips the remaining body of the loop.
   b. It repeats the body of the loop.
   c. It is performs nothing.
   d. It transfers the flow outside the body of loop.

3. Which is true about the **Next** statement with **for Loop**?
   a. It skips the remaining body of the loop, for current iteration only.
   b. It repeats the body of the loop.
   c. It is performs nothing.
   d. It transfers the flow outside the body of loop.

4. **for loop not** allows to iterate on is
   a. seq
   b. list
   c. vector
   d. string

5. Which is not valid for **seq**( ) function.?
   a. It generates the sequence of numbers upto STOP value.
   b. START value is optional.
   c. STEP value is optional.
   d. Data type of sequence is 'NUMERIC.

Q.2 Write the program using **SEQ**( ) function to display the first 10 terms of the following series :

a. 1 , 3, 5,…………19
b. 4 ,8 , 12 …………40
c. 1 , 4 , 9 , 16…….100
d. 2.5 , 5.0 , 7.5 , 9.0 ……..25.0
e. -5 , -10 , -15, -20 ……….-50
f. -20, -18,-16, -14 , -12,…….-2

Q.3 Write a program to calculate and display the sum of all odd numbers and even numbers between a range of numbers from m to n where m < n. Input m and n. Use the **SEQ**( ) Function for generating the sequence from '**m'** to '**n'.**

Q.4 Write a program to print the 10 multiples of any entered number, using **SEQ**( ) Function.

Q.5 Write a program to display the sum of 10 natural numbers, using **SEQ**( ) Function.

Q.7 Write a program print the items at the odd position the list mentioned below ;

Month = ['Jan', 'Feb', 'Mar', 'Apr' , 'May', 'Jun']

1 2 3 4 5 6

Output **in this case , items at odd position 1 , 3 & 5 . i.e. Jan Mar May**

Q.8 Write a program to print the sum of all the numbers at the even position in the sequence generated through seq(1,20,2).

Eg seq(1,20,3) - 1 3 5 7 9 11 13 15 17 19

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

Output - sum of items at index 2 , 4 , 6, 8,10 = 3 + 7 + 11 +15 +19 = 55

Q.9 Write a program to print

[A]	[B]
1	1
1 2	2 1
1 2 3	3 2 1
1 2 3 4	4 3 2 1
1 2 3 4 5	5 4 3 2 1

Day- 7

Q.1 Multiple Choice Question

1. Which one is not atomic datatype in R?
   a. decimal
   b. integer
   c. logical
   d. list

2. Which one is not valid data structure in R?
   a. dataframe
   b. list
   c. vector
   d. set

3. Which is not true about the atomic data types in R?
   a. Logical allows only TRUE and FALSE
   b. 10 is considered as decimal
   c. 10L is considered as integer.
   d. Character data enclosed in single or double quotes.

4. Vectors come in two parts____ and ____.
   a. Atomic vectors and matrix
   b. Atomic vectors and array
   c. Atomic vectors and list
   d. Atomic vector and integer

5. ________ initiates an infinite loop right from the start.
   a. Never
   b. Repeat
   c. Break
   d. Set

6. Which function is used to create the vector with more than one element?
   a. Library()
   b. plot()
   c. c()
   d. par()

Q.2 What is the output ?

1. What is the mode of b in the following R code? b <- c(TRUE, TRUE, 1) mod(b)	2. What will be the output of the following R code? x <- c(3, 7, NA, 4, 7) y <- c(5, NA, 1, 2, 2) x + y
3. What is the length of b? b <- 2:7	4. What is the mode of 'a' in the following R code? a <- c(1,'a', FALSE)
5. What are the typeof(x) and mode(x) in the following R syntax? x<-1:3	6. What will be the output of the following R code? x <- c ('a' , 'b') as.logical(x)

Q.3 What is the output ?

1. `x=c( 12, 23, 49, 45, 35, 23, 12)` `x %% 3`	4. `x=c( 12 ,23, 49, 45, 35, 23, 12)` `x = x[x== 23]`
2. `x= c( 12, 23, 49, 45, 35, 23, 12)` `x = x[x%%3]`	5. `x=c(12, 23, 34, 45, 35, 23, 12)` `x[length(x)-4] = 49`
3. `x= c( 12, 23, 49, 45, 35, 23, 12)` `x = x[x!= 23]`	6. `a <- c(1, 2, 3, 4, 5)` `b <- c(3, 4)` `setdiff(a, b)`

Q.4. Write a program for

x=c(2, NA, 4, NaN, Inf, NaN, NA)

(a) Find the sum of number

(b) Find the count of NaN

(c) Find the count of Infinite

(d) Find the count of NA

Use – is.finite( ) , is.nan( ), is.infinite( ) , is.na( )

e.g. Sum = 2+ 4 = 6 , count of NaN = 2 , count of Infinite =1 , count of NA = 2

Q.5 Write a program to do the operation on vector

1) Create a vector-A, having number from 1 to 10
2) Create a vector-B, having number from 11 to 20
3) Merge the vector A & B to form vector C.
4) Extract and print the even numbers from vector C
5) Extract and print the numbers divisible by 3 from vector C
6) Remove the numbers divisible by 5 from vector C
7) Replace the all the numbers divisible by 3 with 100 in vector C.

Q.6 Write a script to do the operation on vector

X=c(2,NA,4,NaN, Inf,NaN,NA)

1. Remove the Infinite from X.
2. Remove the NA from X.
3. Remove the NaN from X
4. Remove the NaN, NA and Infinite from X.
5. Remove the 2 & 4 from X.

Q.7 What is the output ?

1. X=c(2,4,6,9,10) Id=which(X %%3==0)	2. X=c(2,4,6,9,10) Y=c(4,9,10) Id=which(X %in% Y)
3. X=c(2,4,6,9,10) X[-2]	4. X=c(2,4,6,9,10) X[c(-2,-4)]

Q.8 Write a script to perform the following

x=c('Ajay Singh', 'Ajay Kumar', 'Vijay Singh', 'Vikas Singh', 'Sanjay Jain', 'Sanjay Gupta')

1) Find the name having 'Singh' in name , using grepl( ) function

```
x=c('Ajay Singh', 'Ajay Kumar', 'Vijay Singh', 'Vikas Singh', 'Sanjay Jain', 'Sanjay Gupta')
f=grepl('Singh',x)
r=x[f]
print(r)
```

2) Find the name not having 'Singh' in name , using grepl( ) function

```
x=c('Ajay Singh', 'Ajay Kumar', 'Vijay Singh', 'Vikas Singh', 'Sanjay Jain', 'Sanjay Gupta')
f=grepl('Singh',x)
r=x[! f]
```

```
print(r)
```

3) Find the name having 'Singh' in name , using grep( ) function

```
x=c('Ajay Singh', 'Ajay Kumar', 'Vijay Singh', 'Vikas Singh', 'Sanjay Jain', 'Sanjay Gupta')
f=grep('Singh',x)
r=x[f]
print(r)
```

4) Find the name not having 'Singh' in name , using grepl( ) function

```
x=c('Ajay Singh', 'Ajay Kumar', 'Vijay Singh', 'Vikas Singh', 'Sanjay Jain', 'Sanjay Gupta')
f=grep('Singh',x)
a=1:length(x)
f1=setdiff(a,f)
r=x[f1]
print(r)
```

5) Find the name having 'Singh' in name , using grep( ) function

```
x=c('Ajay Singh', 'Ajay Kumar', 'Vijay Singh', 'Vikas Singh', 'Sanjay Jain', 'Sanjay Gupta')
f=regexpr('Singh', x)
f1=( f!=-1)
r=x[f1]
print(r)
```

6) Find the name not having 'Singh' in name , using grepl( ) function

```
x=c('Ajay Singh', 'Ajay Kumar', 'Vijay Singh', 'Vikas Singh', 'Sanjay Jain', 'Sanjay Gupta')
f=regexpr('Singh', x)
f1=! ( f!=-1)
r=x[f1]
print(r)
```

Day- 8

Q.1 Multiple Choice Question

1. A ________ is a set of elements appearing in rows and columns where the elements are of the same mode whether they are logical, numeric (integer or double), complex or character.
   a) Vector
   b) Matrix
   c) Lists
   d) Data frames

2. Which are indexed by either row or column using a specific name or number?
   a) Datasets
   b) Data frames
   c) Data
   d) Functions

3. By what function we can create data frames?
   a) Data.frame()
   b) Data.sets ()
   c) Function ()
   d) C ()

4. A data frame is a special type of list where every element of the list has _____ length.
   a) Same
   b) Different
   c) May be different
   d) May be same

5. Data frames can have additional attributes such as ________
   a) Rowname()
   b) Rownames()
   c) R.names()
   d) D.names()

6. Lists can be coerced with which function?
   a) As.lists
   b) Has.lists
   c) In.lists
   d) Co.lists

7. A ________ is an R-object which can contain many different types of elements inside it.
   a) Vector
   b) Lists
   c) Matrix
   d) Functions

Q.2 What is the output ?

1. What is the mode of b in the following R code?
`b = list( 10, c(TRUE, TRUE, 1) )`
	2. What will be the output of the following R code?
	`x = list(3, 7, 2, 4, 7)`
	`y = list(5, 2, 1, 2, 2)`
	`x + y`

3. What is the length of b?
`b =list( 2:7)`
	4. What is the mode of ‘a’ in the following R code?
	`a = list(1,2,3)`
	`as.integer(a) +1`

5. What are the typeof(x) and mode(x) in the following R syntax?
`x<-list(1:3)`
	6. What will be the output of the following R code?
	`x = list ('a' , 'b' )`
	`as.logical(x)`

Q.3 What is the output ?

1.
`x=list( 12, 23, 49, 45, 35, 23, 12)`
`unlist(x) %% 3`
	4.
	`x=list ( c(12 ,23, 49) , c(45, 35, 23), 12)`
	`x[1]`
	`x[2]`

2.
`x=list ( c(12 ,23, 49) , c(45, 35, 23), 12)`
`x = unlist(x[2]) +10`
	5.
	`x=c(12, 23, 34, 45, 35, 23, 12)`
	`x[length(x)-4] = 49`

3.
`x=list ( c(12 ,23, 49) , c(45, 35, 23), 12)`
`x[[2]][2] +10`
	6.
	`a = list( c(1, 2, 3), 4, 5 )`
	`b = list( c(3, 4))`
	`a[5]=b`

Q.4. Write a script for

```
x=list ( c(12 ,23, 49) , c(45, 35, 23), 12)
```

(a) To add 100 , 200 , 300 at the end of list

(b) To add 200 at x[10]

(c) remove the item at index 3

Q.5 What is the output

```
df = data.frame( 'X'= c(78,85,96,80,86) , 'Y'= c(84,94,89,83,86) ,'Z'= c(86,97,96,72,83) )
print(df)

df['S']= df['X'] + df['Y'] + df['X']
print(df)
```

Q.6 Perform the Following Operation on Data Frame

1. Create a dataframe with following data of month 1-6

month_number	facecream	facewash	toothpaste	bathingsoap	shampoo	moisturizer
1	2500	1500	5200	9200	1200	1500
2	2630	1200	5100	6100	2100	1200
3	2140	1340	4550	9550	3550	1340
4	3400	1130	5870	8870	1870	1130
5	3600	1740	4560	7760	1560	1740
6	2760	1555	4890	7490	1890	1555

2. Add the column Total_Sales.

3. Create a new data frame with data of month 7-11

7	2980	1120	4780	8980	1780	1120
8	3700	1400	5860	9960	2860	1400
9	3540	1780	6100	8100	2100	1780
10	1990	1890	8300	10300	2300	1890
11	2340	2100	7300	13300	2400	2100

4. Concatenate this data Frame with the DataFrame created in Step-1

5. Print the sales detail of month -7 and month-8.

6. Print the sales detail of shampoo.

7. Print the Total_Sales of all the months.

Q.7 Find the output based on this data

Sales data of Parle Biscuit(in KG)

Name	Jan	Feb	Mar	Apr	May	Jun
Ajay	10	21	23	31	7	22
Vijay	13	17	12	29	14	16
Sanjay	17	15	16	13	18	10
Ajit	45	21	7	34	22	34
Vikas	22	56	76	34	22	16
Vipul	12	17	22	36	31	23
Rakesh	31	23	27	41	32	22

1. Find the Total Sales of each sales person. [df['Total']= apply(df[,c(1:6)], 1, sum)]

2. Find the Maximum Sales of each sales person

3. Show the sales data of “Vikas” for all the months [df[df['Name']==’Vikas’ ,]

4. Show the Sales data of all the sales person for the month of APR.

5. Show the sales data of FEB and JUN for the entire sales person.

6. Show the sales data of “Ajit” for the month of Apr, May, Jun.

7. Find average sales of each sales person.

8. Find the Minimum sales of each sales person. [apply(df[,c(1:3)], 2, min)]

9. Find the Minimum sales for each month

10. Find the median sales of each sales person.

11. Find the median sales for each month.

Day- 9

Q.1 Multiple Choice Question

1. To bind a row onto an already existing matrix, the ______ function can be used.
   a) Rbind
   b) Sbnd
   c) Gbind
   d) Sbind

2. Data frames can be converted to a matrix by calling _______
   a) matr()
   b) matrix()
   c) data.matrix()
   d) None of the above

3. Which of the following statement changes column name to h and f?
   a) colnames(m) = c("h", " f")
   b) columnnames(m) = c("h", " f")
   c) rownames(m) = c("h", " f")
   d) rownames(m) = c("h", " f")

4. Which one invalid function for **factor**( ) ?
   a) levels( )
   b) is.factor( )
   c) is.ordered( )
   d) rev.factor( )

5. Which is invalid parameter for matrix ( ) function?
   a) nrows
   b) ncols
   c) dimnam
   d) byrow=TRUE

6. Which is invalid for matrix( ) ?
   a. Matrix has homogeneous data items.
   b. Data items can be accessed by Row and Column index.
   c. Matrix can have labels as index for Row and Column.
   d. Matrix can have decimals as index for Row and Columns

Q.2 What is the output ?

1. x = 1: 3 y = 10:12 rbind(x, y)	4. x =factor(c("yes", "yes", "no", "yes", "no")) print(x)
2. x = vector("list", length = 5) print(x)	5. x = data.frame(foo = 1:4, bar = c(T, T, F, F)) print(x)
3. x =1: 3 names(x)	6. m = matrix(1:4, nrow = 2, ncol = 2) dimnames(m) = list(c("a", "b"), c("c", "d")) print(m)

Q.3 What is the output ?

A. m= matrix(1: 4, ncol=4) print(m) Y= 11:14 m= rbind(m , Y) print(m)	**B.** m= matrix(1: 4, nrow=4) print(m) Y= 11:14 m= cbind(m , Y) print(m)

Q.4. Write a script for

A =	11	12	13	B =	1	4	7
	14	15	16		2	5	8
	17	18	19		3	6	9

(a) Create the matrix, A and B.

(b) Add the two matrixes, C=A+B.

(c) Join the matrix A and B, horizontally.

(d) Join the matrix A and B, vertically.

(d) Print the sum , max and min of matrix C.

Q.5 Write a R program to extract the submatrix whose rows have column value > 7 from a given matrix.

Q.6 Write a R program to convert a matrix to a 1 dimensional array.

Q.7 Find the output based on this data

Sales data of Parle Biscuit(in KG)

Sales Data for the month of Jan-Jun.

Name	Jan	Feb	Mar	Apr	May	Jun
Ajay	10	21	23	31	7	22
Vijay	13	17	12	29	14	16
Sanjay	17	15	16	13	18	10
Ajit	45	21	7	34	22	34
Vikas	22	56	76	34	22	16
Vipul	12	17	22	36	31	23
Rakesh	31	86	27	41	32	22

Sales Data for the month of Jul-Dec.

Name	Jul	Aug	Sep	Oct	Nov	Dec
Ajay	20	31	33	41	17	32
Vijay	23	27	86	39	24	26
Sanjay	27	25	26	23	28	20
Ajit	55	31	17	44	32	44
Vikas	32	66	86	44	32	26
Vipul	22	27	32	46	41	33
Rakesh	41	33	37	51	42	32

Perform the following operation on the data available in 2 CSV files.

1. Find the Total Sales of each sales person.
2. Find the Maximum Sales of each sales person
3. Show the sales data of “Vikas” for all the months
4. Show the sales data of “Ajit” for the month of Apr, May, Jun.
5. Show the Sales data of all the sales person for the month of APR, AUG and DEC.
6. Find average sales of each sales person.
7. Find the Minimum sales of each sales person.
8. Find the Minimum sales for each month.
9. Find the Sales which are more than the average of sales data.
10. Find the Names of sales person who have achieved level of maximum sales in a year.

1 read the data in two dataframes
2. merge this data together using cbind() function
3. remove the duplicate column - Name
4. use the apply function to solve the query.

```
Q.7
M1=read.csv('d:/datacsv/d13salesdata-1-6.csv', stringsAsFactors = TRUE)
M2=read.csv('d:/datacsv/d13salesdata-7-12.csv',, stringsAsFactors = TRUE)
print(M1)
print(M2)

M=merge(M1,M2, by='Name')
print(M)

#Q.10
mx=max(M[2:length(M)])
f=M[2:length(M)]==mx
r=unlist(apply(f, 2, which))
M$Name[r]

print(multi.which(f))
print(M[2:length(M)][f])

#Q.9
avg=mean(as.matrix(M[2:length(M)]))
print(avg)

f=M[2:length(M)]>avg
print(M[2:length(M)][f])

#Q.3
f=M$Name=='Vikas'
t=M[ f , ]
print(t)

#Q.1
t=apply( M[ ,2:length(M)] , 1 , sum)
print(t)




#M=cbind(M1,M2)
#print(M)
#M=M[ ,-8]
#print(M)

#M=cbind(M1,M2[,2:length(M2)])
#print(M)
```

Day-10

Q.1 Multiple Choice

1. Uni-dimensional arrays are called
   a) vector
   b) matrix
   c) factor
   d) table

2. Default value of dimnames
   a) String
   b) Null
   c) NULL
   d) 2

3. Adding of elements in array using ……….
   a) cat()
   b) join()
   c) merge
   d) append()

4. We can do calculations across the elements in an array using………. With margin.
   a) apply()
   b) remove()
   c) calc()
   d) append()

5. Which is not valid for arrays ?
   a) Access of items is allowed.
   b) Updation of items is allowed.
   c) Numeric and Labeled index for access are allowed.
   d) More than 3 matrix cannot be stored in arrays.

Q.2. Create an uni dimensional array from 1 to 10 and print the length of array?

Q.3 Arranges data from 2 to 18 in two matrices of dimensions 3x3

Q.4

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)
```

Elements are combined into a single vector , then create array of given dimension=2,3,2 and print it.

Q.5. Write a r program to create any 2 matrices of given input as below.

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)
row_names <- c("row1", "row2")
col_names <- c("col1", "col2", "col3")
mat_names <- c("Mat1", "Mat2")
arr = array(c(vec1, vec2), dim = c(2, 3, 2), dimnames = list(row_names, col_names,
mat_names))
```

a) Print both matrices by index and its name
b) Print 1st column of matrix 2
c) Print 1st row of matrix 2
d) Print ("2nd row 3rd column matrix 1 element")
e) Print ("2nd row 1st column element of matrix 2")
f) Print elements of both the rows and columns 2 nd 3 of matrix 1

Q.6.

(a) Add new element 100 on length + 3 index in

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

(b) adds new elements after 3rd index in x<-c(1,2,3,4,5,6,7,8)

Q.7
(a) Creating an array of length 9 as given below and remove all those values greater than 6 and less than 9.

```
m <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

(b) remove sequence of elements using another array using %in %   remove <- c(4, 6, 8)

(c) Print all remaining elements using %in%

Q.8. Create a table using factor values and print occurrence of each element.

```
a <-factor(c("rama","uma","rama","uma","ganesh","rama","ganesh","uma","rama"))
```

Q.9 What will be the output:

```
marks<-matrix(c(70,120,65,140),ncol=2,byrow=TRUE)
rownames(marks)<-c("male","female")
colnames(marks)<-c("math","science")
marks <- as.table(marks)
print(marks)
```

Day-11

Q.1 Multiple Choice

1. Which function is used to counts the total number of characters in a string.
a) length()
b) nchar()
c) count()
d) ccount()

2. Default separator of paste function is
a) ” ”
b) ’ ‘
c) :
d) ;

3. Function returns the smallest integer which is larger than or equal to x.
a) floor()
b) ceiling()
c) round()
d) min()

4. Characters can be translated using
a) grep()
b) chartr()
c) translate()
d) rep()

5. grepl () returns a
a) integer vector
b) logical vctor
c) character vector
d) Numeric vector

Q.2 What will be the output of the following?

1) print(grep("rect", "draw a rectangle", value=T))
2) print(sqrt(-16))
3) print(round(3.45678,3))
4) str = "Learn Code of r programming“
   len = nchar(str)
   print(substr(str, 1, 7))
   print(substr(str, len-4, len))
5) str = c("suresh”, “Chandra”,” Narayan”, “singh”, yadav")
   substr(str, 2, 2) = "%"
   print(str)

Q.3 What is the output ?

1. a=c(11.15, 12.55, 12.95, 13.01, -13.90) r=floor(a) print(r)	2. a=c(11.15, 12.55, 12.95, 13.01, -13.90) r=trunc(a) print(r)	3. a=c(11.15, 212.55, 12.95, 113.01, -213.90) r=round(a, digits=0) print(r)
4. a=c(11.15, 212.55, 12.95, 113.01, -213.90) r=substr(a, 2,3) print(r)	5. a=c('Ajay', 'Vijay', 'SANJAY', 'DIGVIJAY') r=grep('jay',a, ignore.case = TRUE) b=a[r] print(b)	6. a=c('Ajay', 'Vijay', 'SANJAY', 'DIGVIJAY') r=sub('^','Hello! ', a, ignore.case = TRUE) print(r)
7. a='The cat jumps over the Lazy Dog' r=sub('a', 'XX', a) print(r)	8. a='The cat jumps over the Lazy Dog' r=gsub('a', 'XX', a) print(r)	9. a='The cat jumps over the Lazy Dog' r=strsplit(a,split=' ') print(r)

Q.4 What is the output ?

1. a=1:50 y=cut(a,breaks=c(0,20,30,50)) print(y) print(summary(y))	2. a=c(11.15, 12.55, 12.95, 13.01, -13.90) r=scale(a, center=c(10), scale=FALSE) print(r)	3. a=c(11.15, 12.55, 12.95, 13.01, -13.90) r=scale(a, scale=c(10), center=FALSE) print(r)
4. a=c(2, 8, 14, 21, 27) r=diff(a) print(r) print(table(r))	5. a=c(101, 102, 103, 104) f1=ifelse(a%%2==0, 'ODD', 'EVEN') print(f1)	

Q.5 Update the name with proper prefix based on gender, using script .

Name=c('Ajay', 'Ajita', 'SANJAY', 'Sanjana')
Gender=c('M', 'F', 'M', 'F')

Output - ‘Mr. Ajay’, ‘Ms. Ajita’ , ‘Mr. SANJAY’, ‘Ms. Sanjana’

Q.6 For the data available in the data frame,

a) Calculate the Tax at the following rates,
   Tax @10% if BasicPay <= Rs. 15000 , else @15%

b) Add the proper prefix with the name

EmpCode	**Name**	**Gender**	**BasicPay**
101	Ajay	M	10000
102	Ajita	F	15000
103	Vijay	M	20000
104	Sanjana	F	18000

After, adding proper prefix and calculating tax,

EmpCode	**Name**	**Gender**	**BasicPay**	**Tax**
101	Mr.Ajay	M	10000	1000
102	Ms.Ajita	F	15000	1500
103	Mr.Vijay	M	20000	3000
104	Ms.Sanjana	F	18000	2700

Day-12

Q.1 Multiple Choice Question

1. An R function is created by using the keyword?

   a) fun
   b) function
   c) declare
   d) extends

2. Which is invalid Function Argument Types?

   a) Position Argument.
   b) Named Argument.
   c) Partial Argument.
   d) Default Argument.

3. Point out the wrong statement?

   a) Functions in R are "second class objects"
   b) The writing of a function allows a developer to create an interface to the code, that is explicitly specified with a set of parameters
   c) Functions provides an abstraction of the code to potential users
   d) Writing functions is a core activity of an R programmer

4. Which one is not true for User Defined Function?

   a) (…) ellipse can be used to pass any count of items.
   b) Named argument allows changing the position of argument.
   c) Default argument allows argument with default value.
   d) Default argument can be used for integer value only.

5. Which one is not true for Scope of variable?

   a) Global variable can be created using <<- operator inside function.
   b) Global variables can be accessed in any function.
   c) Variables created using <<- operator cannot be accessed outside function.
   d) Local variable has a global lifetime.

Q.2 What will be the output of the following R code snippet?

1)
```
f <- function(a, b) {
        print(a)
        }
    f(45)
```
a) Garbage value
b) Error
c) 45
d) 60

2)
```
f <- function(a, b) {
    a^2
    }
    f(2)
```
a)4
b)6
c)9
d)8

3) Which is not function component
    a) function anme
    b) return statement
    c) arguments
    d) function calling

4) What will be the output of the following R code snippet?
```
new.function <- function(a = 3, b = 6) {
  result <- a * b
  print(result)
}
new.function()
```
    a) 20
    b) 18
    c) 30
    d) 40

Q.3 Write a user function to check given number is positive or negative.
Q.4 Write a user function to input a number and print sum of digit.

Q.5 Write a to find the sum of 4 numbers .

Q.6 Write a function to calculate the area of circle.

Q.7 Write a function to calculate the factorial of a number.

Q.8 Write a function to calculate the sum of first 10 natural number.

Q.9 Write a function to find the person eligible to vote. Details of the person (name , age , address) passed as argument.

Q.10 Write a function to find the sum of all the numbers only passed as argument. The function can accept the arguments of any type. [Use ...]

Example SUM(10, 20, 'Ajay', 30, '#rr' , 40 , '50')

In this case, only 10, 20, 30, 40 is numeric, so the function will return the 100.

Q.11 Write a function to find the count of numbers in the range specified below for the 10 numbers passed as argument and return result.

Count for < 0
Count for 0 - 50
Count for 51 – 100
Count for > 100

Example

-78	89	76	12	45	67	78	91	2	-5

Count for < 0 : 2
Count for 0 - 50 : 3
Count for 51 – 100 : 5
Count for > 100 : 0

Day-13

Q.1 Multiple Choice Question

1. x = 3 ; switch(2, 2+2, mean(1:10), rnorm(5))
a) 5
b) 5.5
c) NULL
d) 58

2. x = 3; switch(6, 2+2, mean(1:10), rnorm(5))
a) 10
b) 1
c) NULL
d) 5

3. y = "fruit"; switch(y, fruit = "banana", vegetable = "broccoli", "Neither")
a) “banana”
b) “Neither”
c) “broccoli”
d) Error

4. `-` (`*` (5,3) , 1)
a) 10
b) 14
c) 12
d) 11

5. '%sum%'= function(x, y, z = 0) x + y + z ; 4 %sum% 5
a) 9
b) Error
c) 4
d) Garbage value

Q. 2. Write an infix operator function to take 10 input numbers and display the power of each number.

Q.3. Write an infix operator function to take one input numbers and display the factorial value of given number.

Q.4 Records of T20 Matches conducted a different cities are available in a CSV file.The File have multiple Columns

Id, Season, city , date , team1 , team2 , tos_winner, toss_decision, result, dl_applied, winner, win_by_run, win_by_wickets, Player_of_match, venue, umpire1, umpire2, umpire3.

Find the following

1. List of cities(unique values only) where matches conducted.
2. List of team(unique values only) participated in the match.
3. Count of matches conducted in year -2010, 2015, 2017
4. Count of matches conducted at 'Bangalore' city.
5. Count of matches conducted in April-2010.
6. Count of matched in which result is 'TIE'
7. Count of matches in which 'SK Raina' was the Player of the match.
8. Count of tos won by each team.
9. Name the Player got maximum Player of Match
10. Name of team won the maximum match in Season-2014 .

Day-14

Q.1 Multiple Choice Questions:

1) Hiding an object's properties from other objects that is
a) Encapsulation
b) Class
c) Object
d) Polymorphism

2) Which allows one class to derive the features and functionalities of another class
a) Encapsulation
b) Inheritance
c) Object
d) Polymorphism

3) An object is an instance of a ____________.
a) Program
b) Class
c) Method
d) Data

4) Which one is allows you to register certain names to be treated as methods in R
a) Generics
b) Module
c) Function
d) Object

5) The Attributes are accessed in S4 class using
a) @
b) $
c) *
d) &

Day-16

Q.1. Multiple Choice Question

1. Which one is not a valid graph type?
   a. plot
   b. Pie
   c. Barplot
   d. LineBar

2. Which one is not true for PLOT function?
   a. main.col is used to set the line color.
   b. main is used to add title for graph.
   c. pch is used specify the plot character for graph.
   d. lty is used to specify the line color for graph.

3. Which one is not related for BARPLOT Graph?
   a. names.arg is used to specify the label for X ticks.
   b. ylab – set the label for Y axis.
   c. ylim – set the range for X axis.
   d. border – to set the box color.

4. Which one is not true statement in HIST graph?
   a. col – color for the Title
   b. ylim – set the range for Y axis.
   c. This graph shows the frequency count of items.
   d. main is used add title for graph.

5. Which is not valid for graph in PIE graph?
   a. col- color for sectors in graph.
   b. labels- used to set the label for each sector in graph.
   c. main – used add title for graph.
   d. setlinep( ) – used to set properties of lines.

Q.2 Create a Histogram graph based on this data available in CSV File

CSV file - Name , Gender , Marks

1) Histogram for Genders

2) Histogram for Marks scored by students

3) Histogram for Marks scored by students in the range 1-50 , 51-75, 75-100

Q. 3 Data of COVID cases in various states in the month of May-2020 and, June-2020.

State	May-2020	June-2020
Maharashtra	28078	39678
Rajasthan	17067	13456
Uttar Pradesh	12670	19654
Kerala	19765	10879
Panjab	18566	12009
Jharkhand	5700	9100
Haryana	3450	8700
Orisha	2300	7800

1. Draw bar chart to compare the COVID cases in various states in the month of May-2020 and, June-2020.

2. Draw a Pie Chart to represent the share of states in positive cases in the Month of May-2020. .

3. Draw a Line Chart to compare the cases of all the states in May-2020 and June-2020.

4. Draw a Pie Chart showing the share of cases of top 4 states and cases of all remaining states as other.