Statistical Computing and R Programming

Course Code: **CAC14 [FAE412]**	Course Title: **Statistical Computing and R Programming**
Course Credits: **04**	Hour of Teaching / Week: **04 Hours**
Formative Assessment Marks: **40**	Total Contact Hours: **54Hours**
Exam Marks: **60**	Exam Duration: **2½Hours**

Course Outcomes (COs):

After the successful completion of the course, the student will be able to:

CO1. Explore fundamentals of statistical analysis in R environment.

CO2. Describe key terminologies, concepts and techniques employed in Statistical Analysis.

CO3. Define Calculate, Implement Probability and Probability Distributions to solve a wide variety of problems.

CO4. Conduct and interpret a variety of Hypothesis Tests to aid Decision Making.

CO5. Understand, Analyse, and Interpret Correlation Probability and Regression to analyse the underlying relationships between different variables.

Course Content	Hours
Unit - 1	
Introduction of the language, numeric, arithmetic, assignment, and vectors, Matrices and Arrays, Non-numeric Values, Lists and Data Frames, Special Values, Classes, and Coercion, Basic Plotting. Reading and writing files, Programming, Calling Functions, Conditions and Loops: stand- alone Statement with illustrations in exercise 10.1,stacking statements, coding loops, Writing Functions, Exceptions, Timings, and Visibility.	18
Unit - 2	
Statistics And Probability, basic data visualization, probability, common probability distributions: common probability mass functions, Bernoulli, binomial, Poisson distributions, common probability density functions, uniform, normal, student's t- distribution. Statistical testing and modeling, sampling distributions, hypothesis testing, components of hypothesis test, testing means, testing proportions, testing categorical variables, errors and power, Analysis of variance.	18
Unit - 3	
Simple linear regression, multiple linear regression, linear model selection and diagnostics. Advanced graphics: plot customization, plotting regions and margins, point and click coordinate interaction, customizing traditional R plots, specialized text and label notation. Defining colors and plotting in higher dimensions, representing and using color, 3D scatter plots.	16

References

1. Tilman M. Davies, "The book of R: A first course in programming and ststistics", San Francisco, 2016.
2. Vishwas R. Pawgi, "Statistical computing using R software", Nirali prakashan publisher, e1 edition, 2022.
3. https://www.youtube.com/watch?v=KlsYCECWEWEhttps://www.geeksforgeeks.org/r-tutorial/
4. https://www.tutorialspoint.com/r/index.htm

Formative Assessment for Theory	
Assessment Occasion/ type	**Marks**
Internal Assessment Test 1	10%
Internal Assessment Test 2	10%
Quiz/ Assignment/ Small Project	10%
Seminar	10%
Total	**40 Marks**
Formative Assessment as per guidelines.	

Course Code: **CAC14P [FAE413]**	Course Title:**R Programming Lab**
Course Credits: **02**	Hour of Teaching / Week: **04 Hours**
Formative Assessment Marks: **40**	Total Contact Hours: 5**4Hours**
Exam Marks: **60**	Exam Duration: **3Hours**

Unit - 1

Introduction to R programming

R is a programming language and software environment for statistical analysis, graphics representation and reporting.

R is an open-source programming language that is widely used as a statistical software and data analysis tool. R is an open-source language and it is available for free for everyone to use for statistical and graphical purposes.R is an interpreted language that supports both procedural programming and object-oriented programming.

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), and partly a play on the name of the Bell Labs Language S.It also combines with lexical scoping semantics inspired by Scheme. Moreover, the project conceives in 1992, with an initial version released in 1995 and a stable beta version in 2000.

Why do we need analytics?

Before an answer to above question, let us see some of the problems and their solutions in R inmultiple domains.

Banking:

Large amount of customer data is generated every day in Banks. While dealing with millions of customers on regular basis, it becomes hard to track their mortgages.

Solution:

R builds a custom model that maintains the loans provided to every individual customer which helps us to decide the amount to be paid by the customer over time.

Insurance:

Insurance extensively depends on forecasting. It is difficult to decide which policy to accept or reject.

Solution:

By using the continuous credit report as input, we can create a model in R that will not only

assess risk appetite but also make a predictive forecast as well.

Healthcare:

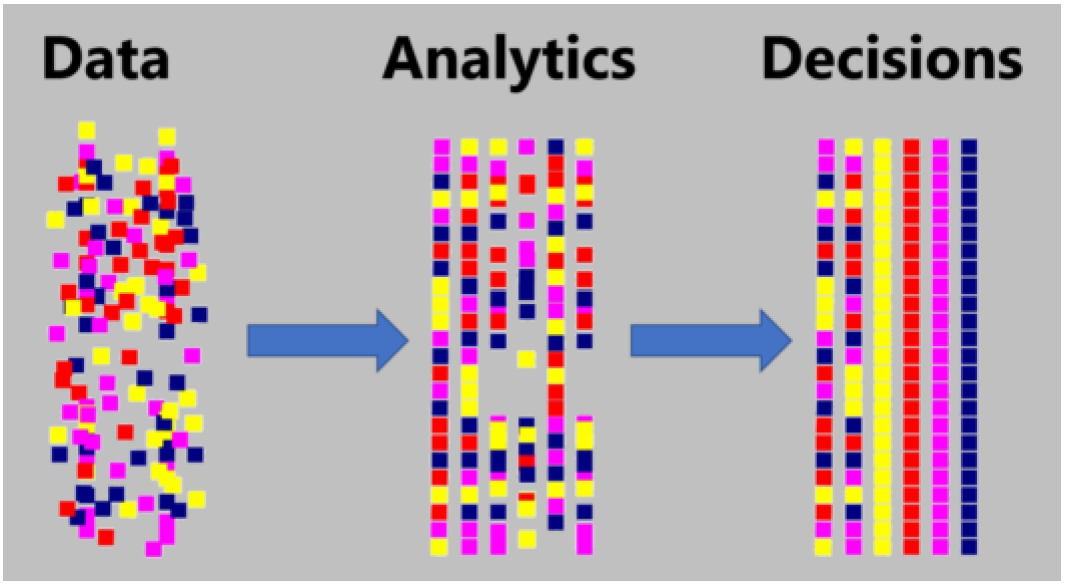
Every year millions of people are admitted in hospital and billions are spent annually just in the admission process.

Solution:

Given the patient history and medical history, a predictive model can be built to identify who isat risk for hospitalization and to what extent the medical equipment should be scaled.

What is Business Analytics?

Business analytics is a process of examining large sets of data and achieving hidden patterns, correlations and other insights. It basically helps you understand all the data that you have gathered, be it organizational data, market or product research data or any other kind of data. It becomes easy for you to make better decisions, better products, better marketing strategies etc. Refer to the below image for better understanding:



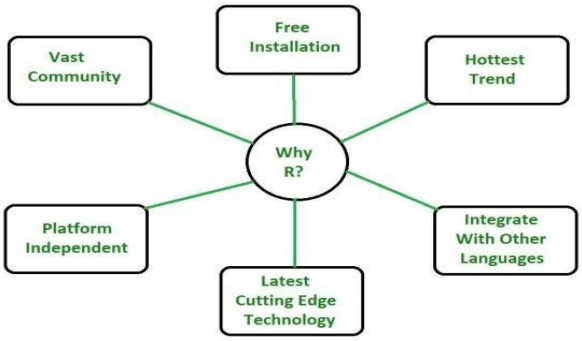
If you look at the above figure, your data in the first image is scattered. Now, if you want something specific such as a particular record in a database, it becomes cumbersome. Tosimplify this, you need analysis. With analysis, it becomes easy to strike a correlation between the data. Once you have established what to do, it becomes quite easy for you to make decisions such as, which path you want to follow or in terms of business analytics, which path will lead to the betterment of your organization.
But you can't expect people in the chain above to always understand the raw data that you are providing them after analytics. So to overcome this gap, we have a concept of *data visualization*.

Data visualization: Data visualization is a visual access to huge amounts of data that you havegenerated after analytics. The human mind processes visual images and visual graphics are better than compare to

raw data. It's always easy for us to understand a pie chart or a bar graphcompare to raw numbers. Now you may be wondering how you can achieve this data visualization from the data you have already analyzed.

There are various tools available in the market for Data Visualization: R, Power BI, Spark, Qlikview etc.

Why R Programming Language?



- R programming is used as a leading tool for machine learning, statistics, and data analysis. Objects, functions, and packages can easily be created by R.
- It's a platform-independent language. This means it can be applied to all operating system.
- It's an open-source free language. That means anyone can install it in any organization without purchasing a license.
- R programming language is not only a statistic package but also allows us to integrate with other languages (C, C++). Thus, you can easily interact with many data sources and statistical packages.
- The R programming language has a vast community of users and it"s growing day by day.
- R is currently one of the most requested programming languages in the Data Science job market that makes it the hottest trend nowadays.

Features of R Programming Language

There are 2 types of features:

a) Statistical Features of R
b) Programming Features of R

a) Statistical Features of R:

- **Basic Statistics:** The most common basic statistics terms are the mean, mode, and median. These are all known as "Measures of Central Tendency." So using the R language we can measure central tendency very easily.
- **Static graphics:** R is rich with facilities for creating and developing interesting static graphics. R contains functionality for many plot types including graphic maps, mosaic plots, biplots, and the list goes on.
- **Probability distributions:** Probability distributions play a vital role in statistics and by using R we can easily handle various types of probability distribution such as Binomial Distribution,

Normal Distribution, Chi-squared Distribution and many more.

- **Data analysis:** It provides a large, coherent and integrated collection of tools for data analysis.

b) Programming Features of R:

- **R Packages:** One of the major features of R is it has a wide availability of libraries. R has CRAN(Comprehensive R Archive Network), which is a repository holding more than 10, 0000 packages.
- **Distributed Computing/ Cross-Platform Support:** Distributed computing is a model in which components of a software system are shared among multiple computers to improve efficiency and performance. Two new packages **ddR and multidplyr** used for distributed programming in R were released in November 2015. R works on multiple operating systems, making it versatile for different environments.
- **Interactive Development:** R allows users to interactively experiment with data and see the results immediately.
- **Data Wrangling:** Tools like **dplyr** and **tidyr** help simplify data cleaning and transformation.
- **Statistical Modeling:** R has built-in support for various statistical models like regression, time-series analysis and clustering.
- **Reproducible Research:** With R Markdown, users can combine code, output and narrative in one document, ensuring their analysis is reproducible.

Programming in R:

Since R is much similar to other widely used languages syntactically, it is easier to code and learn in R. Programs can be written in R in any of the widely used IDE like **R Studio, Rattle, Tinn-R**, etc. After writing the program save the file with the extension **.r**. To run the program use the following command on the command line:

Sample Program

R file_name.r

```
# Below line will print "Welcome to JSSCACS!"
print("Welcome to JSSCACS!")
```

Output

Welcome to JSSCACS!

Advantages of R:

- R is the most comprehensive statistical analysis package. As new technology and concepts often appear first in R.
- As R programming language is an open source. Thus, you can run R anywhere and at any time.
- R programming language is suitable for GNU/Linux and Windows operating system.
- R programming is cross-platform which runs on any operating system.
- In R, everyone is welcome to provide new packages, bug fixes, and code enhancements.

Disadvantages of R:

- In the R programming language, the standard of some packages is less than perfect.
- Although, R commands give little pressure to memory management. So R programming

language may consume all available memory.
- In R basically, nobody to complain if something doesn‟t work.
- R programming is much slower than other programming languages as Python and MATLAB.

Applications of R:
- We use R for Data Science. It gives us a broad variety of libraries related to statistics. It also provides the environment for statistical computing and design.
- R is used by many quantitative analysts as its programming tool. Thus, it helps in data importing and cleaning.
- R is the most prevalent language. So many data analysts and research programmers use it. Hence, it is used as a fundamental tool for finance.
- Tech giants like Google, Facebook, bing, Twitter, Accenture, Wipro and many more using R nowadays.

Who uses R?
- The Consumer Financial Protection Bureau uses R for data analysis
- Statisticians at John Deere use R for time series modeling and geospatial analysis in a reliable and reproducible way.
- Bank of America uses R for reporting.
- R is part of technology stack behind Foursquare's famed recommendation engine.
- ANZ, the fourth largest bank in Australia, using R for credit risk analysis.
- Google uses R to predict Economic Activity.
- Mozilla, the foundation responsible for the Firefox web browser, uses R to visualize Web activity.

Evolution of R
- R is an implementation of S programming language which was created by JohnChambers at Bell Labs.
- R was initially written by Ross Ihaka and Robert Gentleman at the Department ofStatistics of the University of Auckland in Auckland, New Zealand.
- R made its first public appearance in 1993.
- A large group of individuals has contributed to R by sending code and bug reports.Since mid-1997 there has been a core group (the "R Core Team") who can modify the R sourcecode archive.
- In the year 2000 R 1.0.0 released.
- R 3.0.0 was released in 2013.

Installation

R can be downloaded from related mirror sites because it is an open-source software. You should pick your nearest and freely available softwares.

The prominent editors available for R programming language are:

- **RGUI**(R graphical user interface) - Rstudio – Studio R offers a richer editing environment than RGUI and makes some common tasks easier and more fun.
- **RStudio -** RStudio is an integrated development environment (IDE) for R language. RStudio is a code editor and development environment, with some nice features that make code development in R easy and fun.

1. Step-by-step instructions to install R for Windows:

✅ **Step 1: Visit CRAN Website**

- Open your web browser.
- Go to: https://cran.r-project.org

✅ **Step 2: Choose Windows**

- On the CRAN homepage, click on **"Download R for Windows"**.

✅ **Step 3: Base Installation**

- Click on **"base"** (This is the standard version of R for users).
- Then click **"Download R x.x.x for Windows"** (e.g., R 4.4.0).

✅ **Step 4: Run the Installer**

- After the .exe file is downloaded, double-click it to start the installer.

✅ **Step 5: Installation Wizard**

1. **Select Language** – Choose your preferred language and click **OK**.
2. **Welcome Screen** – Click **Next**.
3. **License Agreement** – Click **Next** after reading.
4. **Installation Path** – Default is usually fine (e.g., C:\Program Files\R\R-4.x.x) → Click **Next**.
5. **Components** – Leave default options checked → Click **Next**.
6. **Startup Options** – Choose default unless you need customization → Click **Next**.
7. **Select Start Menu Folder** – Leave default or rename → Click **Next**.
8. **Create Shortcuts** – Choose as per your preference → Click **Next**.
9. **Installation Progress** – Wait while files are copied.
10. **Finish** – Click **Finish** when installation is complete.

✅ **Step 6: Launch R**

- Open **R GUI** from the desktop shortcut or Start Menu.

The R console window will open. You can now start using R.

2. Install RStudio (Recommended IDE)

1. Go to: https://posit.co/download/rstudio-desktop
2. Download the **free RStudio Desktop for Windows**.
3. Install RStudio after R is installed.
4. Open RStudio and start coding in a user-friendly environment.

➢ **R Command Prompt**

Once you have R environment setup, then it's easy to start your R command prompt by just clicking on R Software icon. This will launch R interpreter and you will get a prompt > where you can start typing your programs or commands.

Example:
>x=6
> print(x)

Basic operations:

1. **Using External Data**

R offers plenty of options for loading external data, including Excel, Minitab and SPSS files. We have included a tutorial titled *Data Import* on the subject for the purpose.

2. **Variable Assignment**

We assign values to variables with the assignment operator "=". Just typing the variable by itself at the prompt will print out the value. We should note that another form of assignment operator "<-" is also in use.

```
> x = 1
> x
[1] 1
```

3. **Functions**

R functions are invoked by its name, then followed by the parenthesis, and zero or more arguments. The following apply the function c to combine three numeric values into a vector.

```
> c(1, 2, 3)
[1] 1 2 3
```

4. **Comments**

All text after the pound sign "#" within the same line is considered a comment.

```
> 1 + 1      # this is a comment
[1] 2
```

5. **Extension Package**

Sometimes we need additional functionality beyond those offered by the core R library. In order to install an extension package, you should invoke the install.packages function at the prompt and follow the instruction.

```
> install.packages()
```

6. **Getting Help**

R provides extensive documentation. For example, entering ?c or help(c) at the prompt gives documentation of the function c in R. Please give it a try.

```
> help(c)
```

R Data types

R Data types are used in computer programming to specify the kind of data that can be stored in a variable. For effective memory consumption and precise computation, the right data type must be selected. Each R data type has its own set of regulations and restrictions.

Each variable in R has an associated data type. Each R-Data Type requires different amounts of memory and has some specific operations which can be performed over it.

Features of R Data Types

Feature	Details
Dynamic Typing	R automatically assigns data types based on the value assigned.
Implicit Coercion	R can automatically convert (coerce) data to a common type in operations.
Type Checking	Use typeof(), class(), is.numeric(), etc. to check or test types.
Type Conversion	Use as.numeric(), as.character(), etc. to convert data types.

R Programming language has the following basic R-data types and the following table shows the data type and the values that each data type can take

Data Type	Definition	Example	Size / Capacity
Numeric	Numbers (default type for real numbers)	x <- 5.67	64-bit floating point; up to 1.8e308
Integer	Whole numbers (with L suffix)	x <- 10L	32-bit; up to ~2 billion
Character	Text or string values	x <- "hello"	Depends on memory; each character = 1 byte
Logical	Boolean values (TRUE, FALSE)	x <- TRUE	1 byte per logical value
Complex	Numbers with imaginary part	x <- 3 + 4i	Two 64-bit doubles (real + imaginary)
Raw	Raw bytes (for binary data or strings)	charToRaw("A")	1 byte per value

Hierarchy of Data Types in R (Type Coercion Order)
When combining different types, R converts to the most **flexible type**. The hierarchy is:

raw < logical < integer < numeric < complex < character

Check Data Type in R:

You can check the data type of any object using:

Function	Description	Example	Output
class(x)	Returns the high-level class of the object. For understanding what kind of object (used in functions, object-oriented code)	class(10)	"numeric"
typeof(x)	Returns the internal storage type of the object.	typeof(10)	"double"

is.*() Functions

- **Purpose**: Check whether an object is of a specific data type.
- **Returns**: TRUE or FALSE

Function	Description	Example	Output
is.numeric(x)	Checks if x is numeric (double)	is.numeric(5.6)	TRUE
is.integer(x)	Checks if x is	is.integer(5L)	TRUE

Function	Description	Example	Output
	integer		
is.character(x)	Checks if x is character string	is.character("R")	TRUE
is.logical(x)	Checks if x is logical (TRUE/FALSE)	is.logical(TRUE)	TRUE
is.complex(x)	Checks if x is complex	is.complex(1+2i)	TRUE
is.raw(x)	Checks if x is of raw type	is.raw(as.raw(10))	TRUE

R Data Type Conversion Functions:

Function	Description	Example	Output
as.numeric(x)	Converts x to numeric (double) type	as.numeric("5.6")	5.6
as.integer(x)	Converts x to integer type	as.integer(5.9)	5
as.character(x)	Converts x to character (string) type	as.character(100)	"100"
as.logical(x)	Converts x to logical (TRUE/FALSE)	as.logical(0)	FALSE
as.complex(x)	Converts x to complex number	as.complex(4)	4+0i
as.raw(x)	Converts integer x to raw byte		

Numerics in R

Decimal values are called numerics in R. It is the default R data type for numbers in R. If you assign a decimal value to a variable x as follows, x will be of numeric type. Real numbers with a decimal point are represented using this data type in R. it uses a format for double-precision floating-point numbers to represent numerical values.

It is the default computational data type. If we assign a decimal value to a variable x as follows, x will be of numeric type.

Numerics data type Stores floating-point numbers (64-bit double precision). It Supports arithmetic operations like +, -, *, /.

Sample Program

```
x = 10.5      # assign a decimal value
x             # print the value of x
[1] 10.5
class(x)      # print the class name of x
[1] "numeric"
```

Furthermore, even if we assign an integer to a variable k, it is still being saved as a numeric value.

```
k = 1
k             # print the value of k
[1] 1

class(k)      # print the class name of k
[1] "numeric"
```

The fact that k is *not* an integer can be confirmed with the is.integer function. We will discuss how to create an integer in our next tutorial on the integer type.

```
is.integer(k)  # is k an integer?
[1] FALSE
```

as.integer(k) # as function is used to convert the value of k into integer type

Integer

In order to create an **integer** variable in R, we invoke the integer function. We can be assured that y is indeed an integer by applying the is.integer function.

Sample Program

```
y = as.integer(3)
y             # print the value of y
[1] 3

class(y)      # print the class name of y
[1] "integer"

is.integer(y)  # is y an integer?
[1] TRUE
```

Complex

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

Sample Program

```
z = 1 + 2i    # create a complex number
z             # print the value of z

[1] 1+2i
class(z)      # print the class name of z
[1] "complex"
```

Logical

A **logical** value is often created via comparison between variables.
Sample Program

```
 x = 1; y = 2   # sample values
> z = x > y      # is x larger than y?
> z             # print the logical value
[1] FALSE
> class(z)       # print the class name of z
[1] "logical"
```

Character

A **character** object is used to represent string values in R. We convert objects into character values with the as.character() function:
Sample Program

```
 x = as.character(3.14)
 x             # print the character string
[1] "3.14"
 class(x)       # print the class name of x
[1] "character"
```

Operators

Operators are used in R to perform various operations on variables and values. Among the most commonly used ones are arithmetic and assignment 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.

Types of the operator in R language

- Arithmetic Operators
- Assignment Operators
- Logical Operators
- Relational Operators
- Miscellaneous Operator

Arithmetic Operators
Arithmetic operations in R simulate various math operations, like addition, subtraction, multiplication, division, and modulo using the specified operator between operands, which may be either scalar values, complex numbers, or vectors. The R operators are performed element-wise at the corresponding positions of the vectors.

R supports the following arithmetic operators:

- Addition, +, which returns the sum of two numbers.
- Subtraction, -, which returns the difference between two numbers.
- Multiplication, *, which returns the product of two numbers.
- Division, /, which returns the quotient of two numbers.
- Exponents, ^, which returns the value of one number raised to the power of another.
- Modulus, %%, which returns the remainder of one number divided by another.
- Integer Division, %/%, which returns the integer quotient of two numbers.

Sample Program

```
# R program to illustrate
# the use of Arithmetic operators
a <- 2
b <- 3

# Performing operations on Operands
print("Addition of operators :", a + b, "\n")
print("Subtraction of operators :", a - b, "\n")
print("Multiplication of operators :", a * b, "\n")
print("Division of operators :", a / b, "\n")
print("Modulo of operators :",a %% b, "\n")
print("Power operators :",a ^ b)
```

Output

```
Addition of operators :  5
Subtraction of operators :  -1
Multiplication of operators :  6
Division of operators :  0.6666667
Modulo of operators :  2
Power operators :  8
```

Assignment Operators

Assignment operators in R are used to assigning values to various data objects in R. The objects may be integers, vectors, or functions. These values are then stored by the assigned variable names.

R uses the following assignment operators:

- <- assigns a value to a variable from right to left.
- -> assigns a value to a variable left to right.
- <<- is a global version of <-.
- ->> is a global version of ->.
- = works the same way as <-, but its use is discouraged.

```
# R program to illustrate
# the use of Assignment operators
vec1 <- c(2:5)
c(2:5) ->> vec2
vec3 <<- c(2:5)
```

```
vec4 = c(2:5)
c(2:5) -> vec5

# Performing operations on Operands
print("Assign 1 :", vec1, "\n")
print("Assign 2 :", vec2, "\n")
print("Assign 3 :", vec3, "\n")
print("Assign 4 :", vec4, "\n")
print("Assign 5 :", vec5)
```

Output

```
Assign 1 : 2 3 4 5
Assign 2 : 2 3 4 5
Assign 3 : 2 3 4 5
Assign 4 : 2 3 4 5
Assign 5 : 2 3 4 5
```

Logical Operators

Logical operations in R simulate element-wise decision operations, based on the specified operator between the operands, which are then evaluated to either a True or False boolean value. Any non-zero integer value is considered as a TRUE value, be it a complex or real number.

Element-wise Logical AND operator (&)

Returns True if both the operands are True.

```
list1 <- c(TRUE, 0.1)
list2 <- c(0,4+3i)
print(list1 & list2)
```

Output : FALSE TRUE

Any non zero integer value is considered as a TRUE value, be it complex or real number.

Element-wise Logical OR operator (|)

Returns True if either of the operands is True.

```
list1 <- c(TRUE, 0.1)
list2 <- c(0,4+3i)
print(list1|list2)
```

Output : TRUE TRUE

NOT operator (!)

A unary operator that negates the status of the elements of the operand.

```
list1 <- c(0,FALSE)
print(!list1)
```

Output : TRUE TRUE

Logical AND operator (&&)

Returns True if both the first elements of the operands are True.

```
list1 <- c(TRUE, 0.1)
list2 <- c(0,4+3i)
print(list1 && list2)
```

Output : FALSE
Compares just the first elements of both the lists.

Logical OR operator (||)
Returns True if either of the first elements of the operands is True.

```
list1 <- c(TRUE, 0.1)
list2 <- c(0,4+3i)
print(list1||list2)
```

Output : TRUE

Relational Operators
The relational operators in R carry out comparison operations between the corresponding elements of the operands. Returns a boolean TRUE value if the first operand satisfies the relation compared to the second. A TRUE value is always considered to be greater than the FALSE.

Less than (<)
Returns TRUE if the corresponding element of the first operand is less than that of the second operand. Else returns FALSE.

Less than equal to (<=)
Returns TRUE if the corresponding element of the first operand is less than or equal to that of the second operand. Else returns FALSE.

Greater than (>)
Returns TRUE if the corresponding element of the first operand is greater than that of the second operand. Else returns FALSE.

Greater than equal to (>=)
Returns TRUE if the corresponding element of the first operand is greater or equal to that of the second operand. Else returns FALSE.

Not equal to (!=)
Returns TRUE if the corresponding element of the first operand is not equal to the second operand. Else returns FALSE.

The following R code illustrates the usage of all Relational Operators in R:

```
# R program to illustrate
# the use of Relational operators
vec1 <- c(0, 2)
vec2 <- c(2, 3)
# Performing operations on Operands
cat ("Vector1 less than Vector2 :", vec1 < vec2, "\n")
cat ("Vector1 less than equal to Vector2 :", vec1 <= vec2, "\n")
cat ("Vector1 greater than Vector2 :", vec1 > vec2, "\n")
cat ("Vector1 greater than equal to Vector2 :", vec1 >= vec2, "\n")
cat ("Vector1 not equal to Vector2 :", vec1 != vec2, "\n")
```

Output

Vector1 less than Vector2 : TRUE TRUE
Vector1 less than equal to Vector2 : TRUE TRUE
Vector1 greater than Vector2 : FALSE FALSE
Vector1 greater than equal to Vector2 : FALSE FALSE
Vector1 not equal to Vector2 : TRUE TRUE

Miscellaneous Operators

These operators are used to for specific purpose and not general mathematical or logical computation.

Operator	Description	Example
:	Colon operator. It creates the series of numbers in sequence for a vector.	v <- 2:8 print(v) It produces the following result − [1] 2 3 4 5 6 7 8
%in%	This operator is used to identify if an element belongs to a vector.	v1 <- 8 v2 <- 12 t <- 1:10 print(v1 %in% t) print(v2 %in% t) It produces the following result − [1] TRUE [1] FALSE
%*%	This operator is used to multiply a matrix with its transpose.	Live Demo M = matrix(c(2,6,5,1,10,4), nrow = 2,ncol = 3, byrow = TRUE) t = M %*% t(M) print(t) It produces the following result − [,1] [,2] [1,] 65 82 [2,] 82 117

Data Structures /Data Objects in R Programming

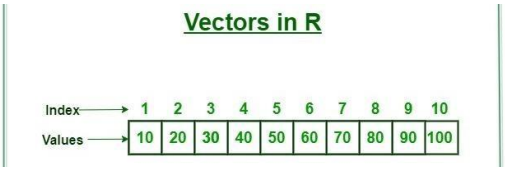
A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures in R programming are tools for holding multiple values.
R‟s base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they‟re homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types). This gives rise to the six data types which are most frequently utilized in data analysis.

The most essential data structures used in R include:

- Vectors
- Matrices
- Arrays
- Lists
- Dataframes
- Factors

Vectors

R vectors are the same as the arrays in C language which are used to hold multiple data values of the same type. One major key point is that in R the indexing of the vector will start from „1‟ and not from „0‟. We can create numeric vectors and character vectors as well.



Types of R vectors
Vectors are of different types which are used in R. Following are some of the types of vectors: Numeric vectors: Numeric vectors are those which contain numeric values such as integer, float, etc.

```
# R program to create numeric Vectors

# creation of vectors using c() function.
v1<- c(4, 5, 6, 7)

# display type of vector
typeof(v1)

# by using 'L' we can specify that we want integer values.
v2<- c(1L, 4L, 2L, 5L)

# display type of vector
typeof(v2)
```

Output

```
[1] "double"

[1] "integer"
```

Character vectors: Character vectors in R contain alphanumeric values and special characters. # R program to create Character Vectors

```
# by default numeric values
# are converted into characters
v1<- c('geeks', '2', 'hello', 57)
# Displaying type of vector
typeof(v1)
```

Output:
[1] "character"

Logical vectors: Logical vectors in R contain Boolean values such as TRUE, FALSE and NA for Null values.
R program to create Logical Vectors

```
# Creating logical vector
# using c() function
v1<- c(TRUE, FALSE, TRUE, NA)
 # Displaying type of vector
typeof(v1)
```

Output:
[1] "logical"

Creating a vector

There are different ways of creating R vectors. Generally, we use „c" to combine different elements together.

```
# R program to create Vectors

# we can use the c function
# to combine the values as a vector.
# By default the type will be double
X<- c(61, 4, 21, 67, 89, 2)
cat('using c function', X, '\n')

# seq() function for creating
# a sequence of continuous values.
# length.out defines the length of vector.
Y<- seq(1, 10, length.out = 5)
cat('using seq() function', Y, '\n')

# use':' to create a vector # of continuous values.
Z<- 2:7
```

```
cat('using colon', Z)
```
Output:

```
using c function 61 4 21 67 89 2
using seq() function 1 3.25 5.5 7.75 10
using colon 2 3 4 5 6 7
```

Length of R vector

```
# Create a numeric vector
x <- c(1, 2, 3, 4, 5)

# Find the length of the vector
length(x)
# Create a character vector
y <- c("apple", "banana", "cherry")

# Find the length of the vector
length(y)
# Create a logical vector
z <- c(TRUE, FALSE, TRUE, TRUE)

# Find the length of the vector
length(z)
```
Output:

```
> length(x)
[1] 5

> length(y)
[1] 3

> length(z)
[1] 4
```

Accessing R vector elements

Accessing elements in a vector is the process of performing operation on an individual element of a vector. There are many ways through which we can access the elements of the vector. The most common is using the „[]“, symbol.

Note: Vectors in R are 1 based indexing unlike the normal C, python, etc format.

```
# R program to access elements of a Vector

# accessing elements with an index number.
X<- c(2, 5, 18, 1, 12)
cat('Using Subscript operator', X[2], '\n')

# by passing a range of values
# inside the vector index.
```

```
Y<- c(4, 8, 2, 1, 17)
cat('Using combine() function', Y[c(4, 1)], '\n')
```
Output:

```
Using Subscript operator 5
Using combine() function 1 4
```

Modifying a R vector

Modification of a Vector is the process of applying some operation on an individual element of a vector to change its value in the vector. There are different ways through which we can modify a vector:

```
# R program to modify elements of a Vector

# Creating a vector
X<- c(2, 7, 9, 7, 8, 2)

# modify a specific element
X[3] <- 1
X[2] <-9
cat('subscript operator', X, '\n')

# Modify using different logics.
X[1:5]<- 0
cat('Logical indexing', X, '\n')

# Modify by specifying
# the position or elements.
X<- X[c(3, 2, 1)]
cat('combine() function', X)
```
Output:

```
subscript operator 2 9 1 7 8 2
Logical indexing 0 0 0 0 0 2
combine() function 0 0 0
```

Deleting a R vector

Deletion of a Vector is the process of deleting all of the elements of the vector. This can be done by assigning it to a NULL value.

```
# R program to delete a Vector

# Creating a Vector
M<- c(8, 10, 2, 5)

# set NULL to the vector
M<- NULL
cat('Output vector', M)
```
Output:
```
Output vector NULL
```

Sorting elements of a R Vector
sort() function is used with the help of which we can sort the values in ascending or descending order.

```
# R program to sort elements of a Vector

# Creation of Vector
X<- c(8, 2, 7, 1, 11, 2)

# Sort in ascending order
A<- sort(X)
cat('ascending order', A, '\n')

# sort in descending order
# by setting decreasing as TRUE
B<- sort(X, decreasing = TRUE)
cat('descending order', B)
```

Output:

```
ascending order 1  2  2  7  8 11
descending order 11  8  7  2  2  1
```

R – Matrices

Matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. In R programming, matrices are two-dimensional, homogeneous data structures. These are some examples of matrices.

Creating a Matrix
To create a matrix in R you need to use the function called matrix(). The arguments to this matrix() are the set of elements in the vector. You have to pass how many numbers of rows and how many numbers of columns you want to have in your matrix.

Note: By default, matrices are in column-wise order.

```
# R program to create a matrix
# Taking sequence of elements
A = matrix(  c(1, 2, 3, 4, 5, 6, 7, 8, 9),
  nrow = 3,                                  # No of rows
    ncol = 3,                                # No of columns
  byrow = TRUE                               # By default matrices are in column-wise order
                                             # So this parameter decides how to arrange the matrix
 )
rownames(A) = c("a", "b", "c")               # Naming rows
colnames(A) = c("c", "d", "e")               # Naming columns
cat("The 3x3 matrix:\n")
print(A)
```

Output
The 3x3 matrix:

```
  c d e
a 1 2 3
b 4 5 6
c 7 8 9
```

Creating special matrices
R allows the creation of various different types of matrices with the use of arguments passed to the matrix() function.
Matrix where all rows and columns are filled by a single constant „k": To create such a R matrix the syntax is given below:
Syntax: matrix(k, m, n)

parameters:
k: the constant
m: no of rows
n: no of columns

```
# R program to illustrate
# special matrices

# Matrix having 3 rows and 3 columns
# filled by a single constant 5

print(matrix(5, 3, 3))
```

Output:

```
     [,1] [,2] [,3]
[1,]    5    5    5
[2,]    5    5    5
[3,]    5    5    5
```

Diagonal matrix:
A diagonal matrix is a matrix in which the entries outside the main diagonal are all zero. To create such a R matrix the syntax is given below:
Syntax: diag(k, m, n)
Parameters:
k: the constants/array
m: no of rows
n: no of columns

Example:

```
# R program to illustrate
# special matrices

# Diagonal matrix having 3 rows and 3 columns
# filled by array of elements (5, 3, 3)

print(diag(c(5, 3, 3), 3, 3))
```

Output:

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

```
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

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. Vectors are supplied as input to the function and then create an array based on the number of dimensions.

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)
```

where,
nrow : Number of rows
ncol : Number of columns
nmat : Number of matrices of dimensions nrow * ncol
dimnames : Default value = NULL.

Otherwise, a list has to be specified which has a name for each component of the dimension. Each component is either a null or a vector of length equal to the dim value of that corresponding dimension.

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.

Example:

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
print (vec1)

# cat is used to concatenate
# strings and print it.

cat ("Length of vector : ", length(vec1))
```

Output:

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

Example:

```
# arranges data from 2 to 13
# in two matrices of dimensions 2x3

arr = array(2:13, dim = c(2, 3, 2))

print(arr)
```

Output:

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

Output:

```
 Mat1
      col1 col2 col3
row1    2    4    6
row2    3    5    7
 Mat2
      col1 col2 col3
row1    8   10   12
row2    9   11   13
```

R – Lists

A list in R is a generic object consisting of an ordered collection of objects. Lists are one-dimensional, heterogeneous data structures. The list can be a list of vectors, a list of matrices, a list of characters and a list of functions, and so on.

A list is a vector but with heterogeneous data elements. A list in R is created with the use of list() function. R allows accessing elements of an R list with the use of the index value. In R, the indexing of a list starts with 1 instead of 0 like in other programming languages.
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)
```

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

```
[1] "Red"
[1] "Green"
[1] 21 32 11
[1] TRUE
[1] 51.23
[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)
```

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

```
$`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.

We continue to use the list in the above example −

```
# 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)
```

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

```
$`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
```

Manipulating List Elements

We can add, delete and update list elements as shown below. We can add and delete 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])
```

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

```
[[1]]
[1] "New element"

$<NA>
NULL
```

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)
```

When we execute the above code, it produces the following 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
```

Data frame

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 in R Programming Language

To create an R data frame use data.frame() command and then pass each of the vectors you have created as arguments to the function.

Example:

```
# R program to create dataframe

# creating a data frame

friend.data <- data.frame(
   friend_id = c(1:5),
   friend_name = c("Sachin", "Sourav",
"Dravid", "Sehwag",
"Dhoni"),
   stringsAsFactors = FALSE
)
# print the data frame

print(friend.data)
```

```
  friend_id friend_name
1         1      Sachin
2         2      Sourav
3         3      Dravid
4         4      Sehwag
5         5       Dhoni
```

Get the Structure of the R – Data Frame

One can get the structure of the R data frame using str() function in R. It can display even the internal

structure of large lists which are nested. It provides one-liner output for the basic R objects letting the user know about the object and its constituents.

Example:

```
# R program to get the
# structure of the data frame

# creating a data frame

friend.data <- data.frame(
   friend_id = c(1:5),
   friend_name = c("Sachin", "Sourav",
"Dravid", "Sehwag",
"Dhoni"),
   stringsAsFactors = FALSE
)
# using str()

print(str(friend.data))
```

```
'data.frame':    5 obs. of  2 variables:
 $ friend_id  : int  1 2 3 4 5
 $ friend_name: chr  "Sachin""Sourav""Dravid""Sehwag" ...
NULL
```

Summary of data in the R data frame

In the R data frame, the statistical summary and nature of the data can be obtained by applying summary() function. It is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

```
# R program to get the
# summary of the data frame

# creating a data frame

friend.data <- data.frame(
   friend_id = c(1:5),
   friend_name = c("Sachin", "Sourav",
"Dravid", "Sehwag",
"Dhoni"),
   stringsAsFactors = FALSE
)
# using summary()

print(summary(friend.data))
```

Output:

```
   friend_id friend_name
 Min.   :1   Length:5
 1st Qu.:2   Class :character
 Median :3   Mode  :character
 Mean   :3
 3rd Qu.:4
```

Max. :5

Extract Data from Data Frame

Extracting data from an R data frame means that to access its rows or columns. One can extract a specific column from an R data frame using its column name.

Example:

```
# R program to extract
# data from the data frame

# creating a data frame

friend.data <- data.frame(
   friend_id = c(1:5),
   friend_name = c("Sachin", "Sourav",
"Dravid", "Sehwag", "Dhoni"),
   stringsAsFactors = FALSE
)

# Extracting friend_name column

result <- data.frame(friend.data$friend_name)

Print(result)
library(dplyr)
# Create a data frame
data <- data.frame(
  friend_id = c(1, 2, 3, 4, 5),
  friend_name = c("Sachin", "Sourav", "Dravid", "Sehwag", "Dhoni"),
  location = c("Kolkata", "Delhi", "Bangalore", "Hyderabad", "Chennai")
)

# Remove a row with friend_id = 3
data <- subset(data, friend_id != 3)
# Remove the 'location' column
data <- select(data, -location)
```

Output:

```
  friend_id friend_name
1         1      Sachin
2         2      Sourav
4         4      Sehwag
5         5       Dhoni
```

In the above code, we first created a data frame called data with three columns: friend_id, friend_name, and location. To remove a row with friend_id equal to 3, we used the subset() function and specified the condition friend_id != 3. This removed the row with friend_id equal to 3.

To remove the location column, we used the select() function and specified -location. The – sign indicates that we want to remove the location column. The resulting data frame data will have only two columns: friend_id and friend_name

Non – numeric values

Non – numeric value is any form of data that is measured in non-number (or word) form. It makes use of symbols and letters. Such data can only be identified in a word format. For example, employee address, name, etc.

Example

```
# Create a data frame

dataframe <- data.frame(students=c('Bhuwanesh', 'Anil', 'Suraj', 'Piyush', 'Dheeraj'),

        section=c('A', 'A', 'C', 'C', 'B'),
        minor=c(87, 98, 71, 89, 82),
        major=c(80, 88, 84, 74, 70))


# Print the dataframe
print(dataframe)
# Create a data frame

dataframe <- data.frame(students=c('Bhuwanesh',
'Anil', 'Suraj',
'Piyush', 'Dheeraj'),

        section=c('A', 'A', 'C', 'C', 'B'),
        minor=c(87, 98, 71, 89, 82),
        major=c(80, 88, 84, 74, 70))

# Try to calculate mean of values in
# section column

mean(dataframe$team)
```

Output

NA

How to avoid this warning ?

The only way to avoid this warning is to use the mean() function with vectors having numeric values only. For example, In the above example we can calculate the mean of the minor column as it contains numerical values only:

```
# Create a data frame
dataframe <- data.frame(students=c('Bhuwanesh', 'Anil', 'Suraj', 'Piyush','Dheeraj'),
        section=c('A', 'A', 'C', 'C', 'B'),
        minor=c(87, 98, 71, 89, 82),
        major=c(80, 88, 84, 74, 70))

# Try to calculate mean of values stored
# at the column minor in the dataframe
mean(dataframe$minor)
Print ( mean)
```

Output

85.4

How to Fix: non-numeric argument to binary operator in R

We will see How to Fix: non-numeric argument to the binary operator in R Programming Language.
The "non-numeric argument to binary operator" error occurs when we perform arithmetic operations on non-numeric elements.

How to produce this error
Here we can see, we have to take the string element and try to add it with a numeric element, so it will occur.

```
num <- "2"
res <- num + 4
print(res)
```

Output:

Error in num + 4: non-numeric argument to binary operator

How to solve it?
To solve this error we will convert non-numeric data into numeric data using as.numeric() methods.

Example 1: Perform into vector
We will convert non-numeric data from vector into numeric data using as.numeric() methods.

```
num <- "2"
res <- as.numeric(num) + 3
print(res)
```

Output is 5

Check if an Object is of Type Numeric in R Programming – is.numeric() Function
is.numeric() function in R Language is used to check if the object passed to it as argument is of numeric type.

Syntax: is.numeric(x)

Parameters:
x: Object to be checked

Example 1:

```
# R program to check if
# object is of numeric type
# Calling is.numeric() function

is.numeric(1)
is.numeric("Program")
is.numeric(-1.5)
```

Output:

[1] TRUE
[1] FALSE
[1] TRUE

Special values in R

R comes with some special values. Some of the special values in R are NA, Inf, -Inf, and NaN. As the name indicates, Missing values are those elements that are not known. NA or NaN are reserved words that indicate a missing value in R Programming language for q arithmetical operations that are undefined.

R – Handling Missing Values
Missing values are practical in life. For example, some cells in spreadsheets are empty. If an insensible or impossible arithmetic operation is tried then NAs occur.

Dealing Missing Values in R
Missing Values in R, are handled with the use of some pre-defined functions:
is.na() Function for Finding Missing values:

A logical vector is returned by this function that indicates all the NA values present. It returns a Boolean value. If NA is present in a vector it returns TRUE else FALSE.

```
x<- c(NA, 3, 4, NA, NA, NA)

is.na(x)
```

Output:

[1] TRUE FALSE FALSE TRUE TRUE TRUE

Properties of Missing Values:

1. For testing objects that are NA use is.na() For testing objects that are NaN use is.nan()
2. There are classes under which NA comes. Hence integer class has integer type NA, the character class has character type NA, etc.
3. A NaN value is counted in NA but the reverse is not valid.
4. The creation of a vector with one or multiple NAs is also possible. Removing NA or NaN values

There are two ways to remove missing values:

Extracting values except for NA or NaN values:
Example 1:

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

d <- is.na(x)
x[! d]
```

Output:

[1] 1 2 3 4

Classes in R Programming

A class is just a blueprint or a sketch of these objects. It represents the set of properties or methods that are common to all objects of one type.

Unlike most other programming languages, R has a three-class system. These are S3, S4, and Reference Classes.

Classes and Objects in R
A class is a user-defined data type from which objects are created. They are also referred to as blueprints sometimes. This is because they define the structure of objects. A class is a passive entity whereas an object is an active entity.

R considers all things as objects. An object is a single instance of a class. R provides us a class() function which can be either used to define the class of a function or get the information of the class for an already defined object.

A class in R is present in a vector form. Due to this property, the objects can inherit from many classes, and also we can specify the order of inheritance even for complex classes.
There are 3 types of Class:
S3 Class
S4 Class
Reference Class

S3 Class
S3 is the simplest yet the most popular OOP system and it lacks formal definition and structure. An object of this type can be created by just adding an attribute to it. Following is an example to make things more clear:
Example:

```
# create a list with required components

movieList <- list(name = "Iron man", leadActor = "Robert Downey Jr")

# give a name to your class
class(movieList) <- "movie"
movieList
```

Output

```
$name
[1] "Iron man"
$leadActor
[1] "Robert Downey Jr
```

In S3 systems, methods don‟t belong to the class. They belong to generic functions. It means that we can‟t create our own methods here, as we do in other programming languages like C++ or Java. But we can define what a generic method (for example print) does when applied to our objects.

```
print(movieList)
```

Output:

```
$name
[1] "Iron man"
$leadActor
[1] "Robert Downey Jr"
```

Example: Creating a user-defined print function

```
# now let us write our method

print.movie <- function(obj)
{
   cat("The name of the movie is", obj$name,".\n")
   cat(obj$leadActor, "is the lead actor.\n")
}
```

Output:

```
The name of the movie is Iron man .
Robert Downey Jr is the lead actor.
```

S4 Class

Programmers of other languages like C++, Java might find S3 to be very much different than their normal idea of classes as it lacks the structure that classes are supposed to provide. S4 is a slight improvement over S3 as its objects have a proper definition and it gives a proper structure to its objects.
Example:

```
library(methods)
# definition of S4 class

setClass("movies", slots=list(name="character", leadActor = "character"))

# creating an object using new() by passing class name and slot values
movieList <- new("movies", name="Iron man", leadActor = "Robert Downey Jr")
movieList
```

Output:

```
An object of class "movies"
Slot "name":
[1] "Iron man"
Slot "leadActor":
[1] "Robert Downey Jr"
```

As shown in the above example, setClass() is used to define a class and new() is used to create the objects.

Reference Class

Reference Class is an improvement over S4 Class. Here the methods belong to the classes. These are much similar to object-oriented classes of other languages.
Defining a Reference class is similar to defining S4 classes. We use setRefClass() instead of setClass() and “fields” instead of “slots”.

Example:

```
library(methods)
# setRefClass returns a generator
movies <- setRefClass("movies", fields = list(name = "character",
```

```
                leadActor = "character", rating = "numeric"))

#now we can use the generator to create objects
movieList <- movies(name = "Iron Man",
                    leadActor = "Robert downey Jr", rating = 7)
movieList
```

Output:

```
Reference class object of class "movies"
Field "name":
[1] "Iron Man"
Field "leadActor":
[1] "Robert downey Jr"
Field "rating":
[1] 7
```

Coercion

When you call a function with an argument of the wrong type, R will try to coerce values to a different type so that the function will work. There are two types of coercion that occur automatically in R: coercion with formal objects and coercion with built-in types.

Coercion includes type conversion . Type conversion means change of one type of data into another type of data. We have to type of coercion occurs :

1. Implicit Coercion
2. Explicit Coercion

Implicit Coercion : When type conversion occurs by itself in R. We input numeric and character data in an object . R converts numeric data to character data by itself

Implicit coercion occurs when we operate on a vector in a way that is not intended for its type. For example, if we add 1 to a logical vector, then the logical values are converted to 0s and 1s implicitly, and 1 is added to each element

Explicit Coercion :

In explicit coercion , we can change one data type to another data type by applying function. We create an object “x” which stores integer values from 1 to 6.

```
x<-0:6
```

We can check data type of “x” object.

```
class(x)
```

We used as.numeric() to change integer data type to numeric data type.

```
z<-as.numeric(x)
```

We check data type of z. It shows “numeric” data type.

class(z)

Basic plots

Basic plots in R. R has a number of built-in tools for basic graph types such as histograms, scatter plots, bar charts, boxplots and much more. Rather than going through all of different types, we will focus on plot() , a generic function for plotting x-y data.

Plot

The plot() function is used to draw points (markers) in a diagram.
The function takes parameters for specifying points in the diagram.

Parameter 1 specifies points on the x-axis.
Parameter 2 specifies points on the y-axis.
At its simplest, you can use the plot() function to plot two numbers against each other:

Graph plotting in R is of two types:

One-dimensional Plotting: In one-dimensional plotting, we plot one variable at a time. For example, we may plot a variable with the number of times each of its values occurred in the entire dataset (frequency). So, it is not compared to any other variable of the dataset.

Two-dimensional Plotting

In two-dimensional plotting, we visualize and compare one variable with respect to the other. For example, in a dataset of Air Quality measures, we would like to compare how the AQI varies with the temperature at a particular place. So, temperature and AQI are two different variables and we wish to see how one changes with respect to the other. These are the 3 major kinds of graphs used for such kinds of analysis

R - Bar Charts

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function barplot() to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax
The basic syntax to create a bar-chart in R is −

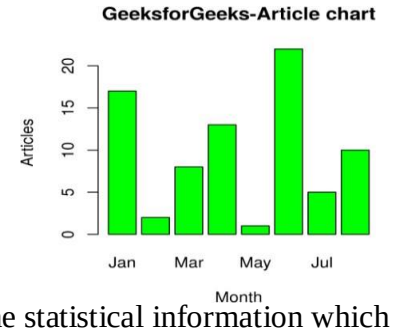
barplot(H,xlab,ylab,main, names.arg,col)
Following is the description of the parameters used −

H is a vector or matrix containing numeric values used in bar chart.
xlab is the label for x axis.
ylab is the label for y axis.
main is the title of the bar chart.
names.arg is a vector of names appearing under each bar.
col is used to give colors to the bars in the graph.

Example
A simple bar chart is created using just the input vector and the name of each bar.

```
# Create the data for the chart
A <- c(17, 2, 8, 13, 1, 22)
B <- c("Jan", "feb", "Mar", "Apr", "May", "Jun")

# Plot the bar chart
barplot(A, names.arg = B, xlab ="Month",
        ylab ="Articles", col ="green",
        main ="GeeksforGeeks-Article chart")
```



Histograms in R language

A histogram contains a rectangular area to display the statistical information which is proportional to the frequency of a variable and its width in successive numerical intervals. A graphical representation that manages a group of data points into different specified ranges. It has a special feature that shows no gaps between the bars and is similar to a vertical bar graph.

R – Histograms
We can create histograms in R Programming Language using the hist() function.

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

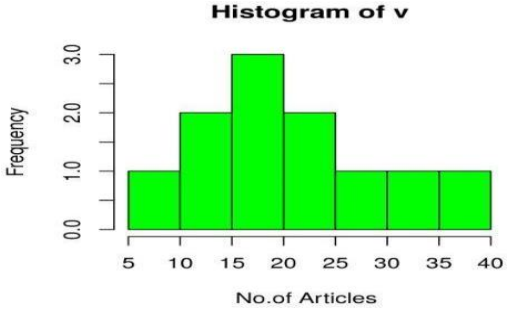
Parameters:

v: This parameter contains numerical values used in histogram.
main: This parameter main is the title of the chart.
col: This parameter is used to set color of the bars.
xlab: This parameter is the label for horizontal axis.
border: This parameter is used to set border color of each bar.
xlim: This parameter is used for plotting values of x-axis.
ylim: This parameter is used for plotting values of y-axis.
breaks: This parameter is used as width of each bar.
Creating a simple Histogram in R
Creating a simple histogram chart by using the above parameter. This vector v is plot using hist().

Example:

```
# Create data for the graph.
v <- c(19, 23, 11, 5, 16, 21, 32, 14, 19, 27, 39)
# Create the histogram.
hist(v, xlab = "No.of Articles ", col = "green", border = "black")
```

Output:



Boxplots in R Language

A box graph is a chart that is used to display information in the form of distribution by drawing boxplots for each of them. This distribution of data is based on five sets (minimum, first quartile, median, third quartile, and maximum).
Boxplots in R Programming Language
Boxplots are created in R by using the boxplot() function.

Syntax: boxplot(x, data, notch, varwidth, names, main)

Parameters:

x: This parameter sets as a vector or a formula.
data: This parameter sets the data frame.
notch: This parameter is the label for horizontal axis.
varwidth: This parameter is a logical value. Set as true to draw width of the box proportionate to the sample size.
main: This parameter is the title of the chart.
names: This parameter are the group labels that will be showed under each boxplot.

Creating a Dataset
We use the data set "mtcars".
Let"s look at the columns "mpg" and "cyl" in mtcars. input
<- mtcars[, c('mpg', 'cyl')]

print(head(input))
Output:

```
                  mpg cyl
Mazda RX4         21.0   6
Mazda RX4 Wag     21.0   6
Datsun 710        22.8   4
Hornet 4 Drive    21.4   6
Hornet Sportabout 18.7   8
Valiant           18.1   6
```

Creating the Boxplot
Creating the Boxplot graph.

Take the parameters which are required to make a boxplot.
Now we draw a graph for the relation between "mpg" and "cyl".

Load the dataset

data(mtcars)

Create the box plot
boxplot(disp ~ gear, data = mtcars,
main = "Displacement by Gear",
xlab = "Gear",
ylab = "Displacement")