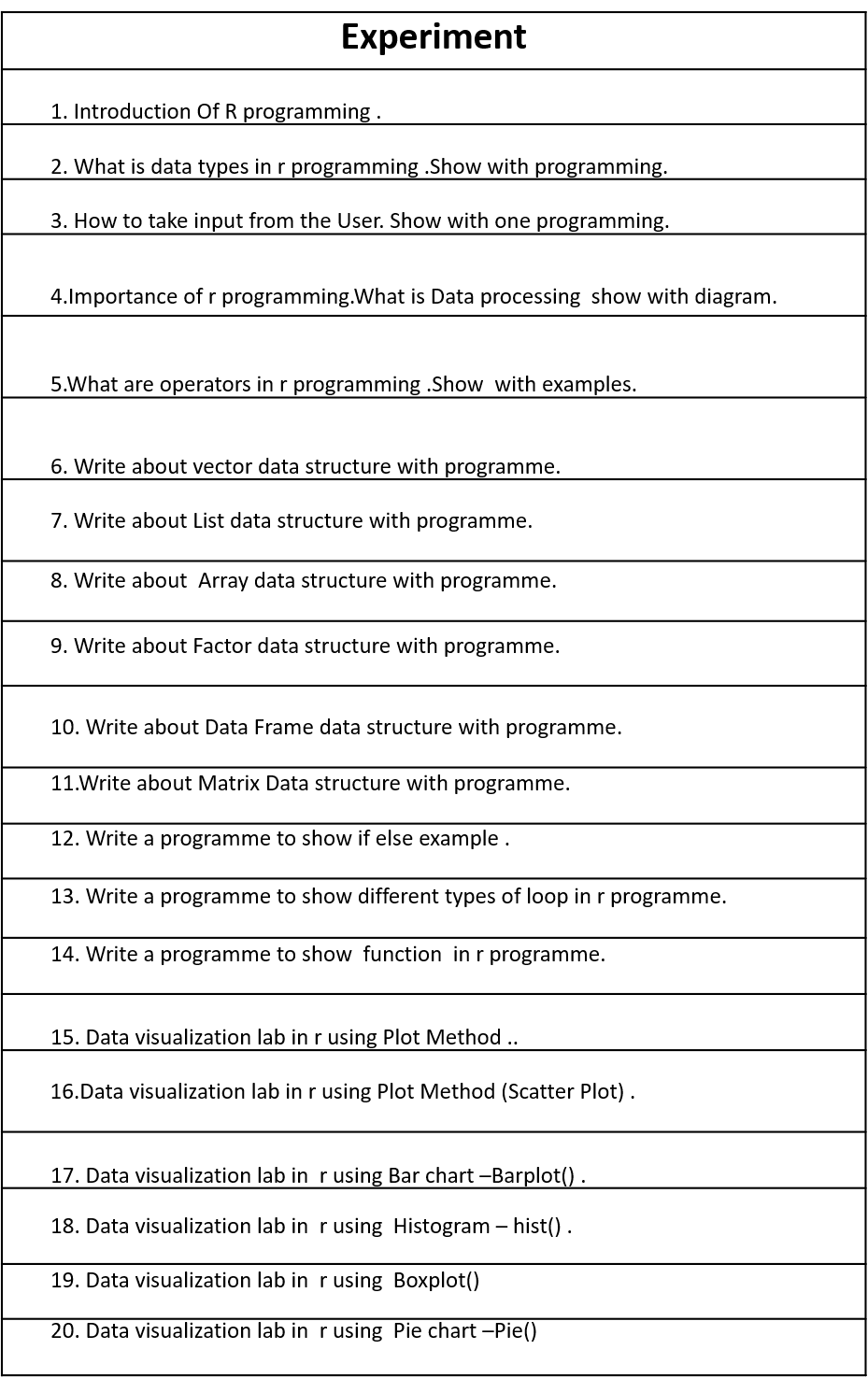
**STAT LAB**



1. 1. Introduction of R programming.

2.

R programming is a powerful language used for statistical computing, data analysis, and visualization. Developed in the early 1990s, it has become popular among data scientists, statisticians, and researchers for its flexibility and strong data manipulation capabilities. R offers a wide range of packages and libraries (like ggplot2 for visualization and dplyr for data manipulation) that make it easy to analyze large datasets and produce high-quality graphics.

Key features include:

1. **Statistical Analysis**: R can handle complex statistical calculations, making it suitable for research and data science.
2. **Data Visualization**: With built-in tools for creating graphs and charts, R makes it easy to visualize insights.
3. **Open-source**: R is free and has a large community contributing packages and libraries, which expand its functionality.
4. **Integrated Development Environment (IDE)**: RStudio is a popular IDE that provides an organized workspace to write and run R code efficiently.

2.What is data type in r programming. Show with programming.

# 1. Numeric

Numeric data includes all numbers, both integers and decimals.

# example: num <- 10.5

print(class(num)) # Output will be "numeric"

# 2. Integer

Integers are whole numbers. In R, integers are specified by adding an "L" suffix. # example: int <- 10L

print(class(int)) # Output will be "integer"

# 3. Character

Character data type is used for text or strings.

# example:

char <- "Hello, R!"

print(class(char)) # Output will be "character"

# 4. Logical

Logical data type has two possible values: TRUE or FALSE.

# example: log <- TRUE

print(class(log)) # Output will be "logical"

# 5. Complex

Complex numbers have a real and imaginary part.

# example: comp <- 4 + 2i

print(class(comp)) # Output will be "complex" **6. Factor**

Factors are used to represent categorical data.

# example:

colors <- factor(c("Red", "Blue", "Green")) print(class(colors)) # Output will be "factor"

3.How to take input from the user.

# Taking name as character input name <- readline(prompt = "Enter your name: ")

# Taking age as numeric input

age <- as.numeric(readline(prompt = "Enter your age: "))

# Taking scores as numeric inputs

scores <- as.numeric(unlist(strsplit(readline(prompt = "Enter your scores for three subjects separated by spaces: "), " ")))

# Calculating the average score average\_score <- mean(scores)

# Displaying the results cat("Hello,", name, "\n") cat("Your age is:", age, "\n") cat("Your scores are:", scores, "\n") cat("Your average score is:", average\_score, "\n")

4.Importance of r programming. What is Data processing show with diagram.

# Importance of R Programming

R programming is significant in data science, statistics, and research for several reasons:

1. **Statistical Analysis**: R is specifically designed for statistical analysis, making it a go-to tool for statisticians.
2. **Data Visualization**: With libraries like ggplot2, R allows users to create high-quality data visualizations, which are essential for interpreting complex datasets.
3. **Extensive Package Ecosystem**: R has thousands of packages for specialized data analysis, which can be easily integrated.
4. **Community and Open-source**: R is free and open-source, with a large, active community that continuously adds to its functionality.
5. **Data Manipulation**: Packages like dplyr and tidyr make it easy to manipulate large datasets efficiently.
6. **Compatibility with Other Languages**: R can work alongside Python, C++, and SQL, making it versatile for different applications.

**What is Data Processing?**

Data processing is the method of collecting raw data and transforming it into meaningful information. This process is crucial in data analysis, as it cleans, organizes, and structures the data, making it suitable for analysis.

**Steps in Data Processing:**

1. **Data Collection**: Gathering raw data from various sources.
2. **Data Cleaning**: Removing inconsistencies, missing values, and errors from the data.
3. **Data Transformation**: Converting data into a suitable format (e.g., numerical, categorical).
4. **Data Integration**: Combining data from different sources for a unified view.
5. **Data Analysis**: Applying statistical and machine learning methods to find patterns.
6. **Data Visualization**: Presenting the results in charts, graphs, and tables.

# Diagram of Data Processing Steps

Here's a diagram representing the data processing flow:

+---------------------+

| Data Collection |

+---------------------+

| v

+---------------------+

| Data Cleaning |

+---------------------+

|

v

+---------------------+

| Data Transformation|

+---------------------+

| v

+---------------------+

| Data Integration |

+---------------------+

| v

+---------------------+

| Data Analysis |

+---------------------+

| v

+---------------------+

| Data Visualization |

+---------------------+

5.What are operators in r programming . Show with examples.

**1. Arithmetic Operators**

These operators are used for basic mathematical operations.

# Operator Description Example

+ Addition 5 + 3 -> 8

- Subtraction 5 - 3 -> 2

\* Multiplication 5 \* 3 -> 15

/ Division 5 / 3 -> 1.6667

^ or \*\* Exponent 5 ^ 3 -> 125

%% Modulus 5 %% 3 -> 2

%/% Integer Division 5 %/% 3 -> 1

# Examples of Arithmetic Operators a <- 10 b <- 3

cat("Addition: ", a + b, "\n") cat("Subtraction: ", a - b, "\n") cat("Multiplication: ", a \* b, "\n") cat("Division: ", a / b, "\n") cat("Exponent: ", a ^ b, "\n") cat("Modulus: ", a %% b, "\n") cat("Integer Division: ", a %/% b, "\n")

# 2. Relational Operators

These operators are used to compare values and return a logical value (TRUE or FALSE).

|  |  |
| --- | --- |
| **Operator Description Example** | |
| == | Equal to 5 == 3 -> FALSE |
| != | Not equal to 5 != 3 -> TRUE |
| > | Greater than 5 > 3 -> TRUE |
| < | Less than 5 < 3 -> FALSE |
| >= | Greater than or equal 5 >= 3 -> TRUE |
| <= | Less than or equal 5 <= 3 -> FALSE |

# Examples of Relational Operators x <- 5 y <- 10

cat("Equal to: ", x == y, "\n") cat("Not equal to: ", x != y, "\n") cat("Greater than: ", x > y, "\n") cat("Less than: ", x < y, "\n")

cat("Greater than or equal to: ", x >= y, "\n") cat("Less than or equal to: ", x <= y, "\n")

**3. Logical Operators**

Logical operators are used for combining multiple conditions.

# Operator Description Example

& Logical AND (element-wise) (x > 3) & (y < 15) -> TRUE

|  |  |
| --- | --- |
| **Operator Description** | **Example** |
| ` ` | Logical OR (elementwise) |
| ! Logical NOT | !(x == y) -> TRUE |
| Logical AND (for single  &&  evaluation) | (x > 3) && (y < 15) |

# Examples of Logical Operators p <- TRUE q <- FALSE

cat("Logical AND: ", p & q, "\n") cat("Logical OR: ", p | q, "\n") cat("Logical NOT: ", !p, "\n")

cat("AND for single evaluation: ", p && q, "\n") cat("OR for single evaluation: ", p || q, "\n")

**4. Assignment Operators**

These operators are used to assign values to variables.

# Operator Description Example

<- Left assignment x <- 5

-> Right assignment 5 -> x

<<- Global assignment (left) x <<- 5

->> Global assignment (right) 5 ->> x

r

Copy code

# Examples of Assignment Operators a <- 5 5 -> b cat("a =", a, "\n") cat("b =", b, "\n")

6.Write about vector data structure with programme.

A vector is the most basic data structure and is used to store a collection of elements of the same data type (numeric, character, logical, etc.). Vectors are one-dimensional arrays that can hold multiple elements but must be of the same type.

# Types of Vectors in R

1. **Numeric Vectors**: Store numeric values.
2. **Character Vectors**: Store strings.
3. **Logical Vectors**: Store TRUE or FALSE values.

4.

**Creating Vectors:** Vectors can be created using the c() function, which combines elements into a vector.

# Numeric Vector num\_vector <- c(1, 2, 3, 4, 5) print(num\_vector)

# Character Vector

char\_vector <- c("apple", "banana", "cherry") print(char\_vector)

# Logical Vector

log\_vector <- c(TRUE, FALSE, TRUE) print(log\_vector)

7. Write about List data structure with programme.

A **list** is a flexible data structure that can hold elements of different data types, such as numeric, character, vector, matrix, and even other lists. This makes lists more versatile than vectors, as vectors require all elements to be of the same type, whereas lists allow a mix of types.

# Key Features of Lists in R

* A list can contain elements of various types, including vectors, matrices, data frames, and other lists.
* Lists are created using the list() function.
* List elements are accessed using double square brackets [[ ]] or by $ when referring to named elements.

# Creating a List

You can create a list using the list() function and combine different data types in it.

# Creating a list with different data types my\_list <- list(

name = "John Doe", # Character element age = 28, # Numeric element scores = c(90, 85, 88), # Numeric vector

details = list(department = "IT", skills = c("R", "Python")) # Nested list

)

print(my\_list)

8. Write about Array data structure with programme.

An **array** is a data structure that can hold elements in multiple dimensions. Arrays are similar to matrices but are not limited to two dimensions. Arrays in R are used to store data in dimensions such as 2D, 3D, or even higher. Each element in an array must be of the same data type.

# Key Features of Arrays in R

* An array can have one or more dimensions.
* All elements in an array must be of the same data type (numeric, character, etc.).
* You specify the dimensions when creating an array, using the dim parameter in the array() function.

# Creating an Array

Arrays are created using the array() function, where you pass the elements and specify the dimensions. **Syntax**

array(data, dim = c(dim1, dim2, ...), dimnames = list("dim1\_name", "dim2\_name", ...))

* data: The data to fill the array with.
* dim: The dimensions of the array, specified as a vector.
* dimnames: Optional names for each dimension.

## Example: Creating a 2D Array (Matrix-like)

* # Creating a 2x3 array
* arr1 <- array(c(1, 2, 3, 4, 5, 6), dim = c(2, 3))
* print("2x3 Array:")
* print(arr1)

•

## Example: Creating a 3D Array

# Creating a 3x3x2 array

* arr2 <- array(c(1:18), dim = c(3, 3, 2))
* print("3x3x2 Array:")
* print(arr2)
* This example creates a 3x3x2 array, which means 3 rows, 3 columns, and 2 layers.

## Accessing Array Elements

Array elements can be accessed by specifying indices for each dimension.

* # Accessing specific elements in a 3D array
* cat("Element at [2, 3, 1]:", arr2[2, 3, 1], "\n")
* cat("Element at [1, 2, 2]:", arr2[1, 2, 2], "\n")

## Array Operations

* You can perform operations across dimensions in an array, such as sum, mean, etc., using the apply() function.
* r
* Copy code
* # Sum of elements along rows (1st dimension)
* row\_sums <- apply(arr2, 1, sum)
* cat("Sum of elements along rows:", row\_sums, "\n")

•

* # Sum of elements along columns (2nd dimension)
* col\_sums <- apply(arr2, 2, sum)
* cat("Sum of elements along columns:", col\_sums, "\n")

9. Write about Factor data structure with programme.

A **factor** is a data structure used to represent categorical data. Factors are particularly useful for statistical modeling and data analysis, as they enable R to treat categorical variables appropriately, allowing for meaningful comparisons and statistical summaries. **Key Features of Factors in R**

1. **Categorical Data**: Factors are used to store categorical data, which can take on a limited number of values (levels), such as

"male" and "female," "low," "medium," and "high."

1. **Levels**: Each unique value in a factor is called a level. R stores factors as integers internally, with levels corresponding to the unique values.
2. **Ordering**: Factors can be ordered or unordered. An ordered factor represents categories that have a natural order (e.g., "low," "medium," "high").

# Creating Factors

Factors are created using the factor() function. You can specify the levels explicitly and set the order if necessary. **Syntax**

factor(x, levels = NULL, labels = NULL, ordered = FALSE)

* x: A vector of categorical data.
* levels: A vector of unique values to define the levels of the factor.
* labels: A vector of labels for the levels.
* ordered: A logical value indicating whether the factor is ordered.

**Example: Creating a Factor** # Creating a factor for gender

gender <- factor(c("male", "female", "female", "male", "female"))

# Printing the factor print("Gender Factor:") print(gender)

**Example: Creating an Ordered Factor** # Creating an ordered factor for education level

education <- factor(c("high school", "bachelor", "master", "phd"), levels = c("high school", "bachelor", "master", "phd"), ordered = TRUE)

# Printing the ordered factor print("Ordered Education Factor:") print(education)

# Accessing Factor Levels

You can access the levels of a factor using the levels() function and retrieve the underlying integer representation with as.integer().

# Accessing factor levels

cat("Levels of Gender Factor:", levels(gender), "\n") cat("Underlying integer representation of Gender Factor:", as.integer(gender), "\n")

# Example Program: Working with Factors in R

Here’s a program that creates factors, accesses levels, and demonstrates their use in data analysis.

# Creating a factor for fruits

fruit <- factor(c("apple", "banana", "apple", "orange", "banana", "banana"))

# Printing the fruit factor cat("Fruit Factor:\n") print(fruit)

# Accessing and printing levels

cat("\nLevels of Fruit Factor:", levels(fruit), "\n")

# Counting occurrences of each level fruit\_count <- table(fruit) cat("\nCount of each fruit:\n") print(fruit\_count)

# Creating an ordered factor for satisfaction ratings

satisfaction <- factor(c("satisfied", "very satisfied", "dissatisfied",

"satisfied", "very dissatisfied"),

levels = c("very dissatisfied", "dissatisfied", "satisfied",

"very satisfied"),

ordered = TRUE)

# Printing the ordered satisfaction factor cat("\nOrdered Satisfaction Factor:\n") print(satisfaction)

# Finding the frequency of each satisfaction level satisfaction\_count <- table(satisfaction) cat("\nCount of each satisfaction level:\n") print(satisfaction\_count)

10. Write about Data Frame data structure with programme.

a **data frame** is a two-dimensional, table-like data structure that can hold data of different types (numeric, character, factor, etc.). Data frames are essential for data manipulation and analysis, making them one of the most commonly used data structures in R, especially when dealing with datasets.

# Key Features of Data Frames in R

1. **Tabular Structure**: Data frames consist of rows and columns, similar to a spreadsheet or a SQL table.
2. **Heterogeneous Data Types**: Each column in a data frame can contain different data types, allowing for a flexible representation of data.
3. **Row and Column Names**: Data frames can have named rows and columns, which makes data manipulation easier and more intuitive.

# Creating a Data Frame

You can create a data frame using the data.frame() function, where you define each column as a separate vector. **Syntax**

data.frame(..., row.names = NULL, check.rows = FALSE, check.names = TRUE, stringsAsFactors = default.stringAsFactors()) • ...: Named columns to be included in the data frame.

* row.names: Optional names for the rows.
* check.rows: If TRUE, checks for duplicate rows.
* check.names: If TRUE, ensures valid column names.
* stringsAsFactors: If TRUE, converts string vectors to factors (deprecated in R 4.0.0). **Example: Creating a Data Frame** r

Copy code

# Creating a data frame for students students <- data.frame(

Name = c("Alice", "Bob", "Charlie"),

Age = c(22, 23, 21),

Gender = c("Female", "Male", "Male"),

Scores = c(85, 90, 88)

)

# Printing the data frame print("Students Data Frame:") print(students)

# Accessing Data Frame Elements

You can access elements in a data frame using:

1. **Column Names**: Using the $ operator or square brackets.
2. **Row Indices**: Using square brackets. # Accessing a column using $ cat("Scores of Students:\n") print(students$Scores)

# Accessing a specific row and column cat("Details of the first student:\n") print(students[1, ])

# Modifying Data Frames

You can add new columns, modify existing ones, or delete columns in a data frame.

# Adding a new column for Pass/Fail status

students$PassFail <- ifelse(students$Scores >= 85, "Pass", "Fail") cat("Data Frame after adding Pass/Fail status:\n") print(students)

# Modifying the age of the first student students$Age[1] <- 23

cat("Data Frame after modifying Age of Alice:\n") print(students)

# Removing the Gender column students$Gender <- NULL

cat("Data Frame after removing the Gender column:\n") print(students)

# Example Program: Working with Data Frames in R

Here’s a complete program that demonstrates the creation, manipulation, and analysis of a data frame.

# Creating a data frame for employee information

employees <- data.frame(

EmployeeID = c(101, 102, 103),

Name = c("John Doe", "Jane Smith", "Emily Johnson"),

Age = c(30, 25, 28),

Department = c("HR", "Finance", "IT"),

Salary = c(70000, 80000, 75000)

)

# Printing the original data frame cat("Original Employees Data Frame:\n") print(employees)

# Accessing a specific column cat("\nSalaries of Employees:\n") print(employees$Salary)

# Accessing specific rows

cat("\nDetails of the second employee:\n") print(employees[2, ])

# Adding a new column for Experience employees$Experience <- c(5, 2, 4) cat("\nData Frame after adding Experience:\n") print(employees)

# Modifying the salary of the first employee employees$Salary[1] <- 72000

cat("\nData Frame after modifying Salary of John Doe:\n") print(employees)

# Removing the Department column employees$Department <- NULL

cat("\nData Frame after removing the Department column:\n") print(employees)

# Summarizing the data cat("\nSummary of Employee Data:\n") summary(employees)

11. Write about Matrix Data structure with programme.

A matrix is a two-dimensional array that stores data in rows and columns. R is especially suited for handling matrix operations efficiently and provides many built-in functions for matrix manipulation.

**Characteristics of Matrices in R:**

1. **Dimensions**: Defined by the number of rows and columns (e.g., a 3×33 \times 33×3 matrix).
2. **Element Access**: Elements can be accessed using row and column indices (e.g., matrix[row, column]).
3. **Types of Matrices**:
   * **Square Matrix**: Number of rows equals the number of columns.
   * **Identity Matrix**: Diagonal elements are 1, and others are

0. o **Transpose**: Flipping rows and columns.

# Basic Operations in R

* **Matrix Addition**: Adding corresponding elements.
* **Matrix Multiplication**: Using matrix algebra to multiply matrices.
* **Transpose**: Interchanging rows and columns.

# Example Program in R

Here’s an example in R to create matrices and perform basic operations such as addition, multiplication, and transpose.

# Creating two matrices

1. <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
2. <- matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)

# Matrix Addition matrix\_addition <- A + B

# Matrix Multiplication (A %\*% B transpose)

matrix\_multiplication <- A %\*% t(B) # 't()' is used to transpose B

# Transpose of matrix A transpose\_A <- t(A)

# Print Results print("Matrix A:") print(A)

print("Matrix B:") print(B)

print("A + B:") print(matrix\_addition)

print("A \* B (dot product):") print(matrix\_multiplication) print("Transpose of A:") print(transpose\_A)

OUTPUT:

Matrix A:

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

[1,] 1 2 3

[2,] 4 5 6

Matrix B:

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

[1,] 7 8 9

[2,] 10 11 12

A + B:

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

[1,] 8 10 12

[2,] 14 16 18

A \* B (dot product):

[,1] [,2]

[1,] 50 68

[2,] 122 167

Transpose of A:

[,1] [,2]

[1,] 1 4

[2,] 2 5

[3,] 3 6

12. Write a programme to show if else example .

# Input: A number

num <- as.numeric(readline(prompt = "Enter a number: "))

# Check if the number is positive, negative, or zero if (num > 0) {

print("The number is positive.")

} else if (num < 0) { print("The number is negative.")

} else {

print("The number is zero.")

}

OUTPUT:

Enter a number: 5

[1] "The number is positive."

Enter a number: -3

[1] "The number is negative."

Enter a number: 0

[1] "The number is zero."

13. Write a programme to show different types of loop in r programme.

1. For loop example: Print numbers from 1 to 5 for (i in 1:5) { print(i)

}

Output:- [1] 1

[1] 2

[1] 3

[1] 4

[1] 5

2. While loop example: Print numbers from 1 to 5 i <- 1 while (i <= 5) { print(i) i <- i + 1

}

Output:- [1] 1

[1] 2

[1] 3

[1] 4

[1] 5

3.Repeat loop example: Print numbers from 1 to 5 i <- 1 repeat { print(i) i <- i + 1 if (i > 5) { break

}

}

Output:- [1] 1

[1] 2

[1] 3

[1] 4

[1] 5

14. . Write a programme to show function in r programme.

# Define the function

factorial\_function <- function(num) {

# Initialize result to 1 result <- 1

# Calculate factorial using a for loop for (i in 1:num) { result <- result \* i

}

# Return the result return(result)

}

# Test the function with an example

num <- as.numeric(readline(prompt = "Enter a number to calculate its factorial: "))

factorial\_result <- factorial\_function(num)

print(paste("The factorial of", num, "is", factorial\_result))

output:- Enter a number to calculate its factorial: 5

[1] "The factorial of 5 is 120"

Data visualization lab in r using Plot Method .

1.Basic Line Plot:- # Creating the data x <- 1:10 y <- x^2

# Creating a line plot

plot(x, y, type = "l", col = "blue", main = "Basic Line Plot", xlab = "Xaxis", ylab = "Y-axis")

2.Plot with point:-

plot(x, y, type = "p", col = "red", main = "Point Plot", xlab = "X-axis", ylab = "Y-axis")

3.Bar chart:-

# Data for the bar chart values <- c(3, 7, 5, 9)

barplot(values, main = "Bar Chart", xlab = "Categories", ylab = "Values", col = "purple")

Data visualization lab in r using Plot Method (Scatter Plot) .

# Scatter Plot Example in R

Suppose we have two variables, x and y, with some data.

# Sample data

x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) y <- c(2, 4, 5, 7, 8, 7, 10, 12, 9, 15)

# Creating the scatter plot plot(x, y,

main = "Scatter Plot Example", # Title of the plot xlab = "X-axis Label", # X-axis label ylab = "Y-axis Label", # Y-axis label col = "blue", # Color of the points pch = 19) # Shape of the points

Data visualization lab in r using Bar chart – Barplot().

# 1.Basic Bar Chart Example

Suppose we have a set of categories and values:

# Sample data values <- c(4, 7, 9, 6) categories <- c("A", "B", "C", "D")

# Creating the bar chart barplot(values,

names.arg = categories, # Names of the categories main = "Basic Bar Chart", # Title of the chart xlab = "Categories", # X-axis label ylab = "Values", # Y-axis label col = "skyblue", # Color of bars border = "black") # Border color for bars

**2.Horizontal Bar Chart** For a horizontal bar chart: set horiz = TRUE: barplot(values, names.arg = categories,

horiz = TRUE, # Makes the chart horizontal

main = "Horizontal Bar Chart", xlab = "Values", col = "lightgreen", border = "black")

# 3.Grouped Bar Chart Example

If you have multiple sets of values, you can create grouped bars:

# Sample data for grouped bars:

grouped\_values <- matrix(c(4, 7, 9, 6, 5, 6, 8, 7), nrow = 2, byrow = TRUE)

barplot(grouped\_values,

beside = TRUE, # Makes bars grouped, not stacked names.arg = categories,

col = c("skyblue", "orange"), # Color for each group main = "Grouped Bar Chart", legend = c("Group 1", "Group 2"))

18. Data visualization lab in r using Histogram – hist() .

# Generating sample data

data <- rnorm(100, mean = 50, sd = 10) # 100 random values with mean 50 and sd 10

# Creating the histogram hist(data,

main = "Basic Histogram", # Title of the histogram xlab = "Values", # X-axis label ylab = "Frequency", # Y-axis label col = "lightblue", # Color of bars border = "black") # Border color of bars

Customizing the Number of Bins:

hist(data,

main = "Histogram with Custom Bins", xlab = "Values", col = "lightgreen", border = "black",

breaks = 20) # Number of bins

Density Plot on Histogram:

hist(data,

main = "Histogram with Density Curve", xlab = "Values", col = "lightcoral", border = "black",

probability = TRUE) # Converts y-axis to density

# Adding a density line

lines(density(data), col = "blue", lwd = 2) # Line width = 2

19.Data visualization lab in r using Boxplot().

# Generating sample data set.seed(123) # For reproducibility

data1 <- rnorm(100, mean = 50, sd = 10) # First dataset data2 <- rnorm(100, mean = 60, sd = 15) # Second dataset

# Creating a boxplot boxplot(data1, data2,

names = c("Dataset 1", "Dataset 2"), # Names for the datasets main = "Basic Boxplot", # Title of the boxplot ylab = "Values", # Y-axis label

col = c("lightblue", "lightgreen"), # Colors for the boxes border = "black") # Border color for the boxes

Customizing the Boxplot:

boxplot(data1, data2,

names = c("Dataset 1", "Dataset 2"), main = "Customized Boxplot", ylab = "Values",

col = c("lightblue", "lightgreen"), border = "black",

notch = TRUE, # Adds notches to the boxes horizontal = TRUE) # Makes the boxplot horizontal

Adding Multiple Groups:

# Generating a larger dataset with a grouping factor group <- rep(c("A", "B", "C"), each = 100) values <- c(rnorm(100, mean = 50, sd = 10), rnorm(100, mean = 60, sd = 15), rnorm(100, mean = 55, sd = 5))

# Creating a boxplot for multiple groups boxplot(values ~ group,

main = "Boxplot for Multiple Groups", ylab = "Values", xlab = "Group",

col = c("lightblue", "lightgreen", "lightcoral"), border = "black")

20.Data visualization lab in r using Pie chart –Pie().

# Sample data

values <- c(20, 30, 25, 25) # Values for each category categories <- c("Category A", "Category B", "Category C", "Category D") # Category names

# Creating the pie chart pie(values,

labels = categories, # Labels for each slice main = "Basic Pie Chart", # Title of the pie chart col = rainbow(length(values))) # Colors for each slice

# Customizing the Pie Chart: # Calculating percentages

percentages <- round(100 \* values / sum(values))

# Creating a pie chart with percentages pie(values,

labels = paste(categories, percentages, "%"), # Labels with percentages

main = "Pie Chart with Percentages", col = rainbow(length(values)),

border = "black") # Border color for slices **Exploding a Slice:**

# Exploding the first slice (Category A)

explode <- c(0.1, 0, 0, 0) # Values indicate how far to pull out each slice

# Creating the exploded pie chart pie(values,

labels = paste(categories, percentages, "%"), main = "Exploded Pie Chart", col = rainbow(length(values)), border = "black",

init.angle = 90, # Start the first slice at 90 degrees radius = 0.8, # Radius of the pie density = NULL, # No shading angle = 45, # Angle for shading explode = explode) # Explode effect