Lab Manual

Part A

1. Write a program for different types of data structures in R

```
# Creating a Numeric Vector
n <- c(1, 2, 3, 4, 5)
print("Numeric Vector:")
print(n)
# Creating a Character Vector
cv <- c("apple", "banana", "cherry")
print("Character Vector:")
print(cv)

# Creating a List
lst <- list(1, "apple", TRUE)
print("List:")
print(lst)
# Creating a Matrix
m <- matrix(1:6, nrow = 2, ncol = 3)
print("Matrix:")
print(m)
# Creating a Data Frame
df <- data.frame(Name = c("Alice", "Bob", "Charlie"),
                 Age = c(25, 30, 22))
print(" Data Frame:")
print(df)
```

Output

```
[1] "Numeric Vector:"
[1] 1 2 3 4 5
[1] "Character Vector:"
[1] "apple"  "banana" "cherry"

[1] "List:"
[[1]]
[1] 1
[[2]]
[1] "apple"
[[3]]
[1] TRUE

[1] "Matrix:"
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

[1] " Data Frame:"
  Name Age
1 Alice  25
2 Bob  30
3 Charlie  22
```

2. Write a program that include variables, constants, data types

```
# Constants
PI <- 3.14159
GRAVITY <- 9.81includes
# Variables
name <- "John Doe"
age <- 30
is_student <- TRUE
grades <- c(85, 92, 78, 95, 89)
# Displaying information
cat("Name:", name, "\n")
cat("Age:", age, "\n")
cat("Is Student:", is_student, "\n")
```

```
cat("Grades:", grades, "\n"
# Data Types
ntype <- typeof(name)
atype <- typeof(age)
itype <- typeof(is_student)
gtype <- typeof(grades)
cat("Data Type of 'name':", ntype, "\n")
cat("Data Type of 'age':", atype, "\n")
cat("Data Type of 'is_student':", itype, "\n")
cat("Data Type of 'grades':", gtype, "\n")
```

Output

```
Name: John Doe
Age: 30
is Student: TRUE
Grades: 85 92 78 95 89
Data Type of 'name': character
Data Type of ‘age’: double
Data Type of 'is_student': logical
Data Type of 'grades': double
```

3. Write a R program that includes different operators, control structures, default values for arguments, returning complex objects

```
# Function with default argument values
calculate_area <- function(shape = "rectangle", length = 5, width = 3) {
 if (shape == "rectangle") {
  area <- length * width
 } else if (shape == "circle") {
  radius <- length /2
  area <- pi * radius^2
 } else {
  cat("Unsupported shape:", shape, "\n")
  return(NULL)
 }

 # Conditional operator
 message <- ifelse(area > 10, "Large area", "Small area")
  result <- list(
  Shape = shape,
```

```
    Length = length,
    Width = width,
    Area = area,
    Message = message
  )
   return(result)
}
# Using the function with default arguments
result1 <- calculate_area()
result2 <- calculate_area("rectangle", 8, 4)
result3 <- calculate_area("circle", 6)
# Displaying results
cat("Result 1:\n")
print(result1)
cat("\nResult 2:\n")
print(result2)
cat("\nResult 3:\n")
print(result3)
```

Output

```
Result 1:
$Shape
[1] "rectangle"
$Length
[1] 5
$Width
[1] 3
$Area
[1] 15
$Message
[1] "Large area"
Result 2:
$Shape
[1] "rectangle"
$Length
[1] 8
$Width
[1] 4
```

```
$Area
[1] 32
$Message
[1] "Large area"
Result 3:
$Shape
[1] "circle"
$Length
[1] 6
$Width
[1] 3
$Area
[1] 28.27433
$Message
[1] "Large area"
```

4.Write a R program for quick sort implementation, binary search tree

```
# Quick Sort Implementation
quick_sort <- function(arr) {
 if (length(arr) <= 1) {
  return(arr)
 }
  pivot <- arr[1]
 smaller <- arr[arr < pivot]
 equal <- arr[arr == pivot]
 larger <- arr[arr > pivot]
  sorted_smaller <- quick_sort(smaller)
 sorted_larger <- quick_sort(larger)
 return(c(sorted_smaller, equal, sorted_larger))
}
# Binary Search Tree Implementation
insert_node <- function(root, key) {
 if (is.null(root)) {
  return(list(key = key, left = NULL, right = NULL))
 }
  if (key < root$key) {
  root$left <- insert_node(root$left, key)
 } else if (key > root$key) {
```

```
$Area
[1] 32
$Message
[1] "Large area"
Result 3:
$Shape
[1] "circle"
$Length
[1] 6
$Width
[1] 3
$Area
[1] 28.27433
$Message
[1] "Large area"
```

4.Write a R program for quick sort implementation, binary search tree

```
# Quick Sort Implementation
quick_sort <- function(arr) {
 if (length(arr) <= 1) {
  return(arr)
 }
  pivot <- arr[1]
 smaller <- arr[arr < pivot]
 equal <- arr[arr == pivot]
 larger <- arr[arr > pivot]
  sorted_smaller <- quick_sort(smaller)
 sorted_larger <- quick_sort(larger)
 return(c(sorted_smaller, equal, sorted_larger))
}
# Binary Search Tree Implementation
insert_node <- function(root, key) {
 if (is.null(root)) {
  return(list(key = key, left = NULL, right = NULL))
 }
  if (key < root$key) {
  root$left <- insert_node(root$left, key)
 } else if (key > root$key) {
```

```
    root$right <- insert_node(root$right, key)
  }
   return(root)
}
in_order_traversal <- function(root) {
 if (!is.null(root)) {
   in_order_traversal(root$left)
   cat(root$key, " ")
   in_order_traversal(root$right)
 }
}
# Example usage of Quick Sort
arr <- c(6, 2, 8, 3, 1, 7, 5)
sorted_arr <- quick_sort(arr)
cat("Quick Sort Result:", sorted_arr, "\n")
# Example usage of Binary Search Tree
bst_root <- NULL
keys <- c(6, 2, 8, 3, 1, 7, 5)
for (key in keys) {
 bst_root <- insert_node(bst_root, key)
}
cat("In-order Traversal of BST:")
in_order_traversal(bst_root)
```

Output

```
Quick Sort Result: 1 2 3 5 6 7 8
In-order Traversal of BST:1  2  3  5  6  7  8
```

5. Write a R program for calculating cumulative sums, and products minima maxima calculus.

```
cum <- cumsum(numbers)
cat("Cumulative Sum:", cum, "\n")

# Sample dataset
data <- c(2, 4, 1, 8, 5, 7)

# Cumulative sums
csum <- cumsum(data)
cat("Cumulative Sums:", csum, "\n")
```

```
# Cumulative products
cproduct <- cumprod(data)
cat("Cumulative Products:", cproduct, "\n")

# Minimum and Maximum
minvalue <- min(data)
maxvalue <- max(data)
cat("Minimum Value:", minvalue, "\n")
cat("Maximum Value:", maxvalue, "\n")

# Calculus: Differentiation
differentiate <- diff(data)
cat("Differentiation (First Difference):", differentiate, "\n")

# Calculus: Integration
integrate <- cumsum(data)
cat("Integration (Cumulative Sum):", integrate, "\n")
```

Output

```
Cumulative Sum: 2 6 12 20 30
Cumulative Sums: 2 6 7 15 20 27
Cumulative Products: 2 8 8 64 320 2240
Minimum Value: 1
Maximum Value: 8
Differentiation (First Difference): 2 -3 7 -3 2
Integration (Cumulative Sum): 2 6 7 15 20 27
```

6. Write a R program for finding stationary distribution of markanov chains.

```
# Install and load the markovchain package
 # install.packages("markovchain")
library(markovchain)
# Define the transition matrix for your Markov chain
tm <- matrix(c(
  0.7, 0.3,
  0.2, 0.8
), byrow = TRUE, nrow = 2, dimnames = list(c("State1", "State2"), c("State1", "State2")))
# Create a Markov chain object
markov_chain <- new("markovchain", states = c("State1", "State2"), transitionMatrix = tm)
# Find the stationary distribution
```

```
stationary_distribution <- steadyStates(markov_chain)
print("Stationary Distribution:")
print(stationary_distribution)
```

Output

```
"Stationary Distribution:"
     State1 State2
[1,]    0.4    0.6
```

7. Write a R program that include linear algebra operations on vectors and matrices

```
# Create vectors
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)

# Create matrices
matrix1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
matrix2 <- matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)

# Display vectors and matrices
cat("Vector 1:\n")
print(vector1)

cat("\nVector 2:\n")
print(vector2)

cat("\nMatrix 1:\n")
print(matrix1)

cat("\nMatrix 2:\n")
print(matrix2)

# Linear algebra operations
cat("\nLinear Algebra Operations:\n")

# Vector addition
vector_sum <- vector1 + vector2
cat("Vector Addition (vector1 + vector2):\n")
print(vector_sum)
```

```
# Vector subtraction
vector_diff <- vector1 - vector2
cat("Vector Subtraction (vector1 - vector2):\n")
print(vector_diff)

# Scalar multiplication
scalar <- 2
vector_scalar_mult <- scalar * vector1
cat("Scalar Multiplication (scalar * vector1):\n")
print(vector_scalar_mult)

# Matrix addition
matrix_sum <- matrix1 + matrix2
cat("Matrix Addition (matrix1 + matrix2):\n")
print(matrix_sum)

# Matrix subtraction
matrix_diff <- matrix1 - matrix2
cat("Matrix Subtraction (matrix1 - matrix2):\n")
print(matrix_diff)

# Matrix multiplication
matrix_product <- matrix1 %*% t(matrix2)  # Matrix multiplication with transpose of matrix2
cat("Matrix Multiplication (matrix1 * matrix2):\n")
print(matrix_product)

# Matrix transpose
matrix1_transpose <- t(matrix1)
cat("Matrix Transpose (transpose of matrix1):\n")
print(matrix1_transpose)
```

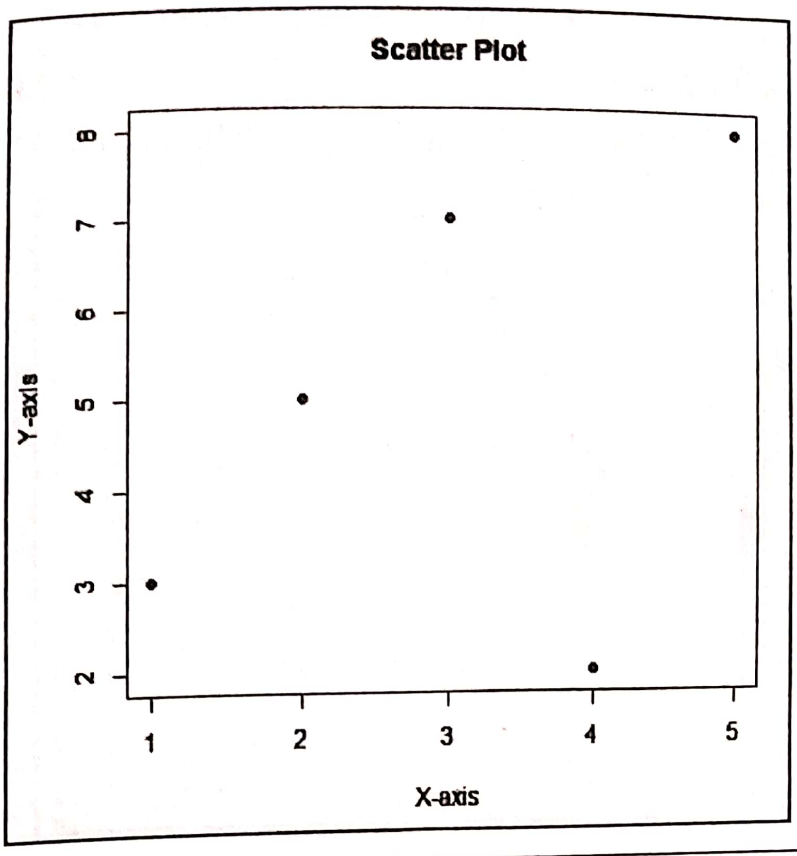
Output

```
Vector 1:
[1] 1 2 3
Vector 2:
[1] 4 5 6
Matrix 1:
     [,1] [,2] [,3]
```

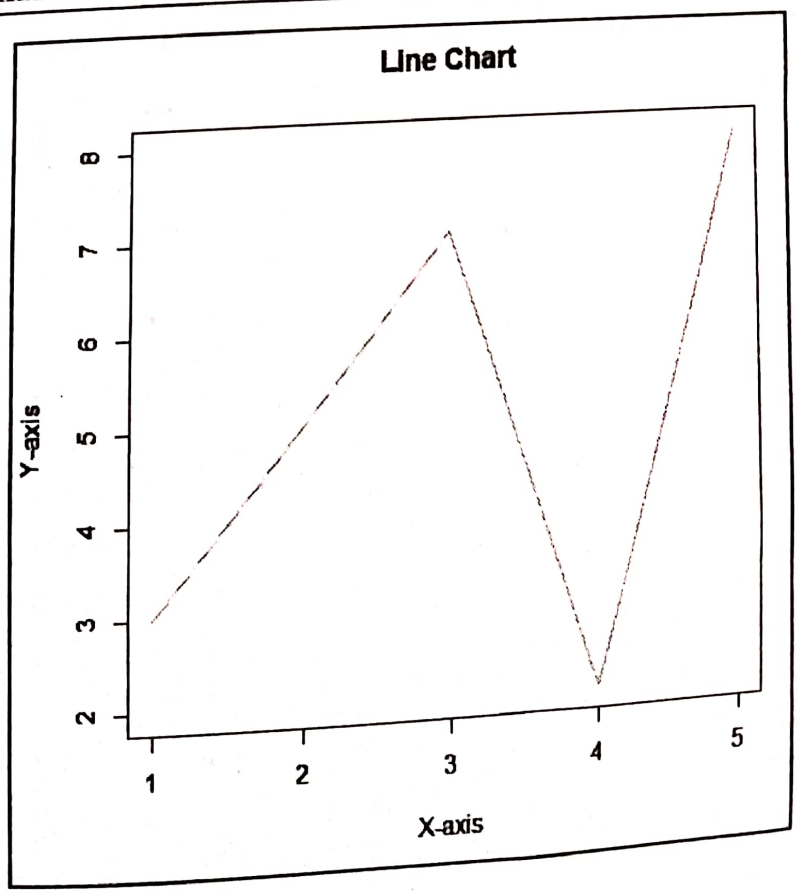
```
[1,]   1   3   5
[2,]   2   4   6
Matrix 2:
     [,1] [,2] [,3]
[1,]   7   9   11
[2,]   8   10   12
Linear Algebra Operations:
Vector Addition (vector1 + vector2):
[1] 5 7 9
Vector Subtraction (vector1 - vector2):
[1] -3 -3 -3
Scalar Multiplication (scalar * vector1):
[1] 2 4 6
Matrix Addition (matrix1 + matrix2):
     [,1] [,2] [,3]
[1,]   8   12   16
[2,]   10   14   18
Matrix Subtraction (matrix1 - matrix2):
     [,1] [,2] [,3]
[1,]  -6   -6   -6
[2,]  -6   -6   -6
Matrix Multiplication (matrix1 * matrix2):
     [,1] [,2]
[1,]  89   98
[2,] 116  128
Matrix Transpose (transpose of matrix1):
     [,1] [,2]
[1,]   1   2
[2,]   3   4
[3,]   5   6
```

8. Write a R program for any visual representation of an object with creating graphs using graphic functions: Plot(),Hist(),Linechart(),Pie(),Boxplot(),Scatterplots().

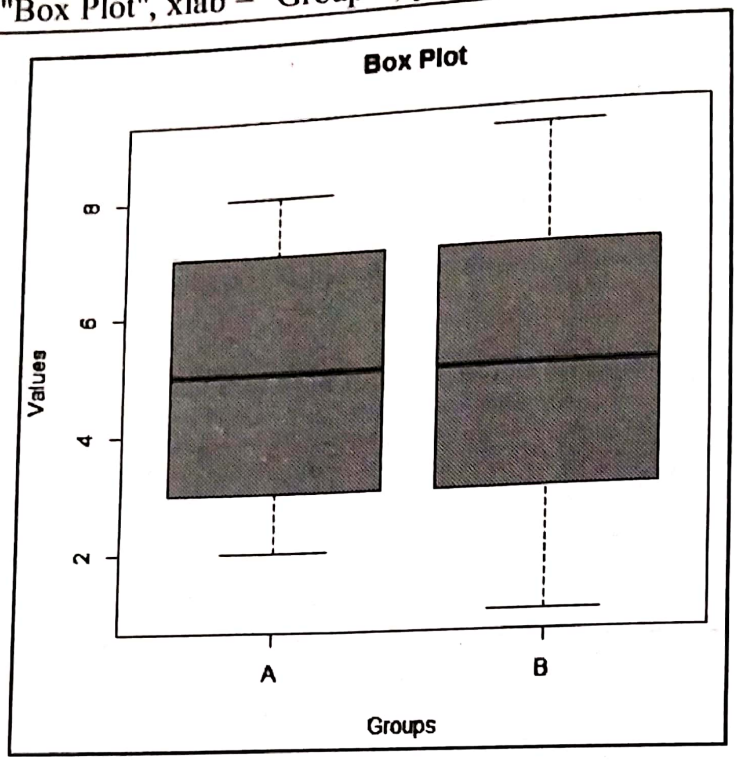
```
Scatterplots()
x <- c(1, 2, 3, 4, 5)
y <- c(3, 5, 7, 2, 8)
plot(x, y, main = "Scatter Plot", xlab = "X-axis", ylab = "Y-axis", pch = 19, col = "blue")
```



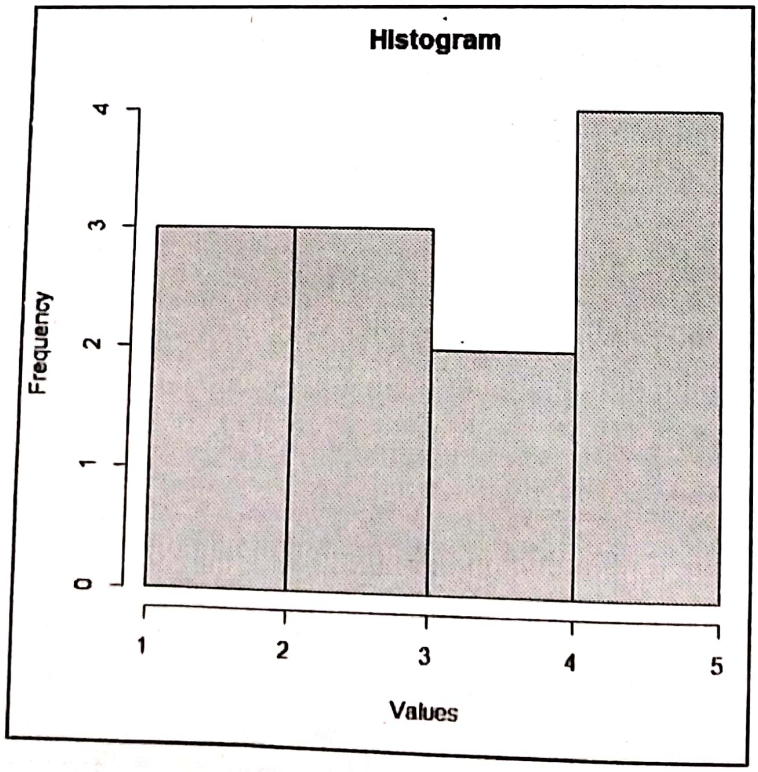

```
Linechart()
x <- c(1, 2, 3, 4, 5)
y <- c(3, 5, 7, 2, 8)
plot(x, y, type = "l", main = "Line Chart", xlab = "X-axis", ylab = "Y-axis", col = "green")
```



```
Boxplot()
data <- list(A = c(2, 4, 6, 8), B = c(1, 3, 5, 7, 9))
boxplot(data, main = "Box Plot", xlab = "Groups", ylab = "Values", col = "orange")
```

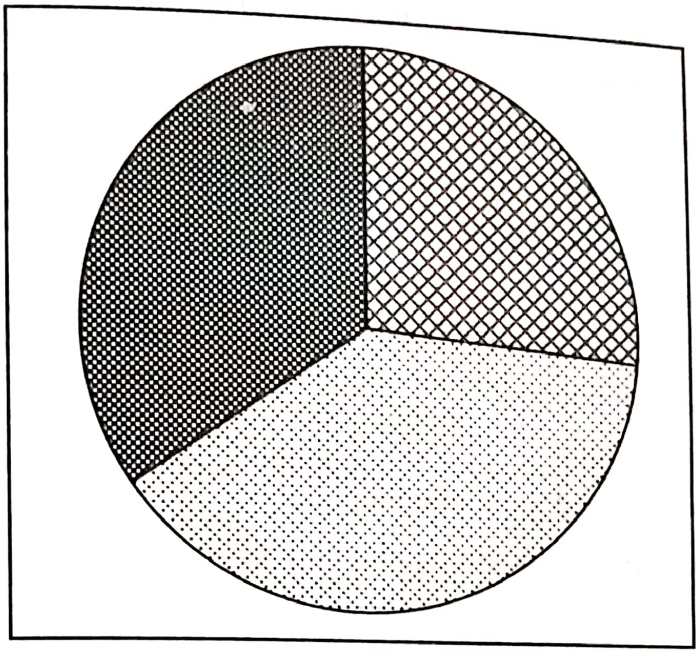



```
Hist(),
data <- c(1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5)
hist(data, main = "Histogram", xlab = "Values", ylab = "Frequency", col = "lightblue")
```



```
pie()
categories <- c("A", "B", "C")
values <- c(30, 40, 20)
pie(values, labels = categories, main = "Pie Chart", col = c("red", "green", "blue"))
```

Pie Chart



9. Write a R program for with any dataset containing dataframe objects, indexing and subsetting data frames, and employ manipulating and analyzing data.

```
# Create a sample employee dataset as a data frame
employee_data <- data.frame(
 EmployeeID = c(1, 2, 3, 4, 5),
 FirstName = c("John", "Alice", "Bob", "Carol", "David"),
 LastName = c("Smith", "Johnson", "Johnson", "Smith", "Davis"),
 Age = c(30, 25, 28, 35, 32),
 Department = c("HR", "Marketing", "Finance", "HR", "IT"),
 Salary = c(50000, 55000, 60000, 52000, 70000)
)
# Print the entire employee dataset
cat("Employee Data:\n")
print(employee_data)
# Subset and index the data frame
cat("\nSubset and Indexing:\n")
# Select employees in the HR department
hr_employees <- employee_data[employee_data$Department == "HR", ]
cat("HR Employees:\n")
print(hr_employees)
```

```
# Select employees aged 30 or older
older_employees <- employee_data[employee_data$Age >= 30, ]
cat("Employees Aged 30 or Older:\n")
print(older_employees)
# Select employees with a salary greater than $55,000
high_salary_employees <- employee_data[employee_data$Salary > 55000, ]
cat("Employees with High Salary:\n")
print(high_salary_employees)
# Manipulate and analyze the data
cat("\nData Manipulation and Analysis:\n")
# Calculate the average salary
average_salary <- mean(employee_data$Salary)
cat("Average Salary:", average_salary, "\n")
# Calculate the maximum age
max_age <- max(employee_data$Age)
cat("Maximum Age:", max_age, "\n")
# Calculate the number of employees in each department
department_counts <- table(employee_data$Department)
cat("Number of Employees in Each Department:\n")
print(department_counts)
# Calculate the total payroll for each department
department_payroll <- tapply(employee_data$Salary, employee_data$Department, sum)
cat("Total Payroll in Each Department:\n")
print(department_payroll)
```

Output:

Employee Data:

	EmployeeID	FirstName	LastName	Age	Department	Salary
1	1	John	Smith	30	HR	50000
2	2	Alice	Johnson	25	Marketing	55000
3	3	Bob	Johnson	28	Finance	60000
4	4	Carol	Smith	35	HR	52000
5	5	David	Davis	32	IT	70000

Subset and Indexing:

HR Employees:

	EmployeeID	FirstName	LastName	Age	Department	Salary
1	1	John	Smith	30	HR	50000
4	4	Carol	Smith	35	HR	52000

Employees Aged 30 or Older:

	EmployeeID	FirstName	LastName	Age	Department	Salary
1	1	John	Smith	30	HR	50000
4	4	Carol	Smith	35	HR	52000
5	5	David	Davis	32	IT	70000

Employees with High Salary:

	EmployeeID	FirstName	LastName	Age	Department	Salary
3	3	Bob	Johnson	28	Finance	60000
5	5	David	Davis	32	IT	70000

Data Manipulation and Analysis:

Average Salary: 57400

Maximum Age: 35

Number of Employees in Each Department:

Finance	HR	IT	Marketing
1	2	1	1

Total Payroll in Each Department:

Finance	HR	IT	Marketing
60000	102000	70000	55000

10. Write a program to create an any application of Linear Regression in a multivariate context for predictive purpose

```
# Sample dataset: Salary, Years of Experience, Education Level
data <- data.frame(
 Salary = c(50000, 60000, 75000, 80000, 95000, 110000, 120000, 130000),
 Experience = c(1, 2, 3, 4, 5, 6, 7, 8),
 Education = c(12, 14, 16, 16, 18, 20, 20, 22)
)

# Perform multivariate linear regression
model <- lm(Salary ~ Experience + Education, data = data)
model
# Predict salaries for new data
```

```
new_data <- data.frame(
  Experience = c(9, 10),
  Education = c(22, 24)
)
predicted_salaries <- predict(model, newdata = new_data)
# Print the predicted salaries
cat("Predicted Salaries:\n")
print(predicted_salaries)
```

Output:

```
Call:
lm(formula = Salary ~ Experience + Education, data = data)

Coefficients:
(Intercept)  Experience   Education
    11500        8500        2333

Predicted Salaries:
      1       2
139333.3 152500.0
```

Part B

1. Write a Program to calculate the factorial of a number:

```
# Function to calculate the factorial of a number
factorial <- function(n) {
  if (n == 0) {
    return(1)  # The factorial of 0 is defined as 1
  } else {
    return(n * factorial(n - 1))
  }
}

# Test the factorial function
result <- factorial(5)  # Calculate 5!
cat("Factorial of 5 is:", result, "\n")
```

Output

```
Factorial of 5 is: 120
```

2. Program to find the sum of 2 number using function.

```
# Defining a simple function in R
sum_function <- function(a, b) {
 result <- a + b  # Compute the sum of the two arguments
 return(result)   # Return the result
}

# Using the function with sample arguments
x <- 3
y <- 4
z <- sum_function(x, y)
print(z)
```

Output

```
[1] 7
```

3. Write a program that demonstrates the use of lists:

```
# Creating a list
my_list <- list(name = "John Doe",
          age = 30,
          is_student = TRUE,
          grades = c(85, 90, 95, 87, 92))
# Printing the list
print(my_list)
# Accessing elements in the list
print(paste("Name:", my_list$name))
print(paste("Age:", my_list$age))
print(paste("Is Student:", my_list$is_student))
print(paste("Grades:", my_list$grades))
# Modifying elements in the list
my_list$name <- "Jane Doe"
my_list$age <- 25
my_list$is_student <- FALSE
my_list$grades <- c(75, 80, 85, 77, 82)
# Printing the modified list
print(my_list)
```

Output

```
$name
[1] "Jane Doe"
$age
[1] 25
$is_student
[1] FALSE
$grades
[1] 75 80 85 77 82
```

4. Program to calculate the area of a triangle

```
triangle_area <- function(base, height) {
  area <- 0.5 * base * height
  return(area)
}
# Example usage of the function
base_length <- 6
height_length <- 8
area <- triangle_area(base_length, height_length)
cat("The area of the triangle is:", area, "\n")
```

Output

```
The area of the triangle is: 24
```

5. program to swap the values of two variable.

```
# Function to swap two numbers
swap_numbers <- function(a, b) {
  temp <- a
  a <- b
  b <- temp
  return(list(a = a, b = b))
}
# Example usage of the function
num1 <- 5
num2 <- 10
Cat ("Before swapping: num1 =", num1, "and num2 =", num2, "\n")
# Call the function to swap the numbers
result <- swap_numbers(num1, num2)
cat("After swapping: num1 =", result$a, "and num2 =", result$b, "\n")
```

Output

```
Before swapping: num1 = 5 and num2 = 10
After swapping: num1 = 10 and num2 = 5
```

6. Write a Program to perform basic matrix operations such as addition, subtraction, multiplication, and transpose.

```
# Create matrices
matrix1 <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE)
matrix2 <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2, byrow = TRUE)
# Print the matrices
cat("Matrix 1:\n")
print(matrix1)
cat("\nMatrix 2:\n")
print(matrix2)
# Addition of matrices
addition_result <- matrix1 + matrix2
cat("\nAddition Result:\n")
print(addition_result)
# Subtraction of matrices
subtraction_result <- matrix1 - matrix2
cat("\nSubtraction Result:\n")
print(subtraction_result)
# Multiplication of matrices
multiplication_result <- matrix1 %*% matrix2
cat("\nMultiplication Result:\n")
print(multiplication_result)
# Transpose of matrices
transpose_matrix1 <- t(matrix1)
transpose_matrix2 <- t(matrix2)
cat("\nTranspose of Matrix 1:\n")
print(transpose_matrix1)
cat("\nTranspose of Matrix 2:\n")
print(transpose_matrix2)
```

Output

```
Addition Result:
     [,1] [,2]
[1,]    6    8
```

```
[2,]  10  12
Subtraction Result:
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4
Multiplication Result:
     [,1] [,2]
[1,]   19   22
[2,]   43   50
Transpose of Matrix 1:
     [,1] [,2]
[1,]    1    3
[2,]    2    4
Transpose of Matrix 2:
     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

7. Program to demonstrate creating, modifying, and accessing data in data frames

```
# Creating a dataframe
data <- data.frame(
  Name = c("John", "Doe", "Jane", "Smith"),
  Age = c(25, 30, 28, 35),
  Salary = c(50000, 60000, 55000, 70000)
)
# Printing the dataframe
cat("Initial Dataframe:\n")
print(data)

# Adding a new column
data$Department <- c("HR", "Finance", "IT", "Marketing")
# Printing the updated dataframe
cat("\nDataframe with New Column:\n")
print(data)
# Modifying a value in the dataframe
data[3, "Salary"] <- 60000
# Printing the dataframe after modification
cat("\nDataframe after Modification:\n")
```

```
print(data)
# Accessing specific elements
cat("\nAccessing Specific Elements:\n")
cat("Second row, second column: ", data[2, 2], "\n")
cat("Name column: ", data$Name, "\n")
# Filtering data
cat("\nFiltered Dataframe (Age > 25):\n")
filtered_data <- data[data$Age > 25, ]
print(filtered_data)
```

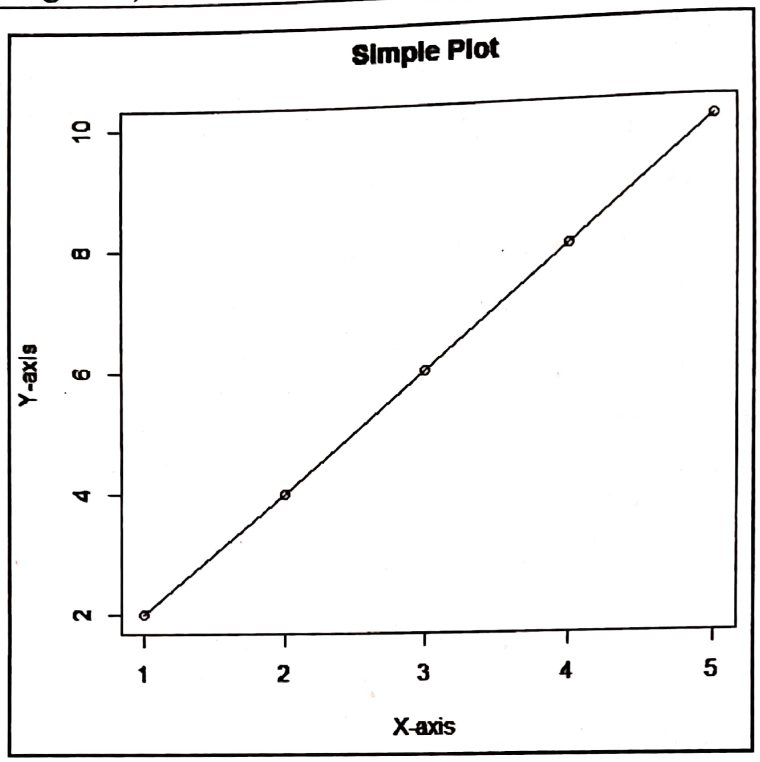
Output

```
Dataframe with New Column:
  Name Age Salary Department
1 John  25  50000         HR
2  Doe  30  60000    Finance
3 Jane  28  55000         IT
4 Smith  35  70000  Marketing
Dataframe after Modification:
  Name Age Salary Department
1 John  25  50000         HR
2  Doe  30  60000    Finance
3 Jane  28  60000         IT
4 Smith  35  70000  Marketing
Accessing Specific Elements:
Second row, second column:  30
Name column:  John Doe Jane Smith
Filtered Dataframe (Age > 25)
  Name Age Salary Department
2  Doe  30  60000    Finance
3 Jane  28  60000         IT
Smith  35  70000  Marketing
```

8. Program to generate a basic plot in R

```
# Creating data for the plot
x <- c(1, 2, 3, 4, 5)
y <- c(2, 4, 6, 8, 10)
# Plotting the data
plot(x, y, type = "o", col = "blue", xlab = "X-axis", ylab = "Y-axis", main = "Simple Plot")
```

```
# Adding points to the plot
points(x, y, col = "red")
# Adding a line to the plot
abline(lm(y ~ x), col = "green")
```



9. program to calculate the sum of the first 10 natural numbers

```
# Initializing sum variable
sum_result <- 0
# Using a for loop to calculate the sum of the first 10 natural numbers
for (i in 1:10) {
  sum_result <- sum_result + i
}
# Printing the result
cat("Sum of the first 10 natural numbers is:", sum_result)
```

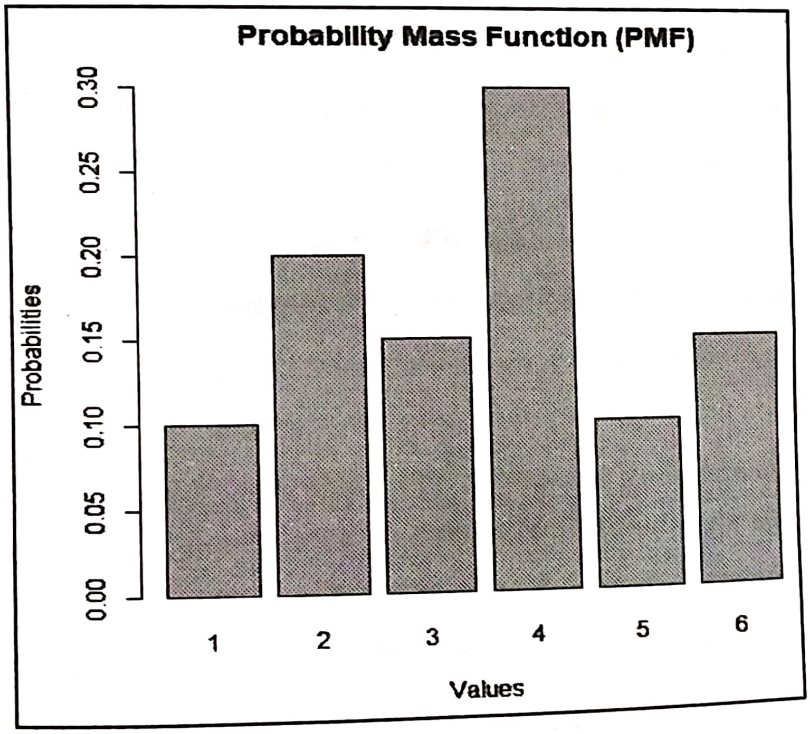
Output

```
Sum of the first 10 natural numbers is: 55
10. Program to find the mean, variance, and plot bar chart
# Define the probabilities for each value
pmf <- c(0.1, 0.2, 0.15, 0.3, 0.1, 0.15)  # Example probabilities for values 1 to 6
# Set the corresponding values
values <- 1:6
# Print the PMF
print(pmf)
```

```
# Print the values
print(values)
# Calculate the mean
mean_value <- sum(pmf * values)
print(paste("Mean:", mean_value))
# Calculate the variance
var_value <- sum(pmf * (values - mean_value)^2)
print(paste("Variance:", var_ value))
# Plot the PMF
barplot(pmf, names.arg = values, xlab = "Values", ylab = "Probabilities", main = "Probability Mass Function (PMF)")
```

Output

```
[1] "Mean: 3.55"
[1] "Variance: 2.3475"
```



Ω♦Ω♦Ω♦Ω