EXPERIMENT-1

AIM:- Exercises to implement the basic matrix operations in R.

THEORY:-

Matrix Multiplication:

Matrix multiplication is not performed element-wise. Instead, it's a bit more complex.

For two matrices A and B to be multiplied, the number of columns in A must be equal to the number of rows in B.

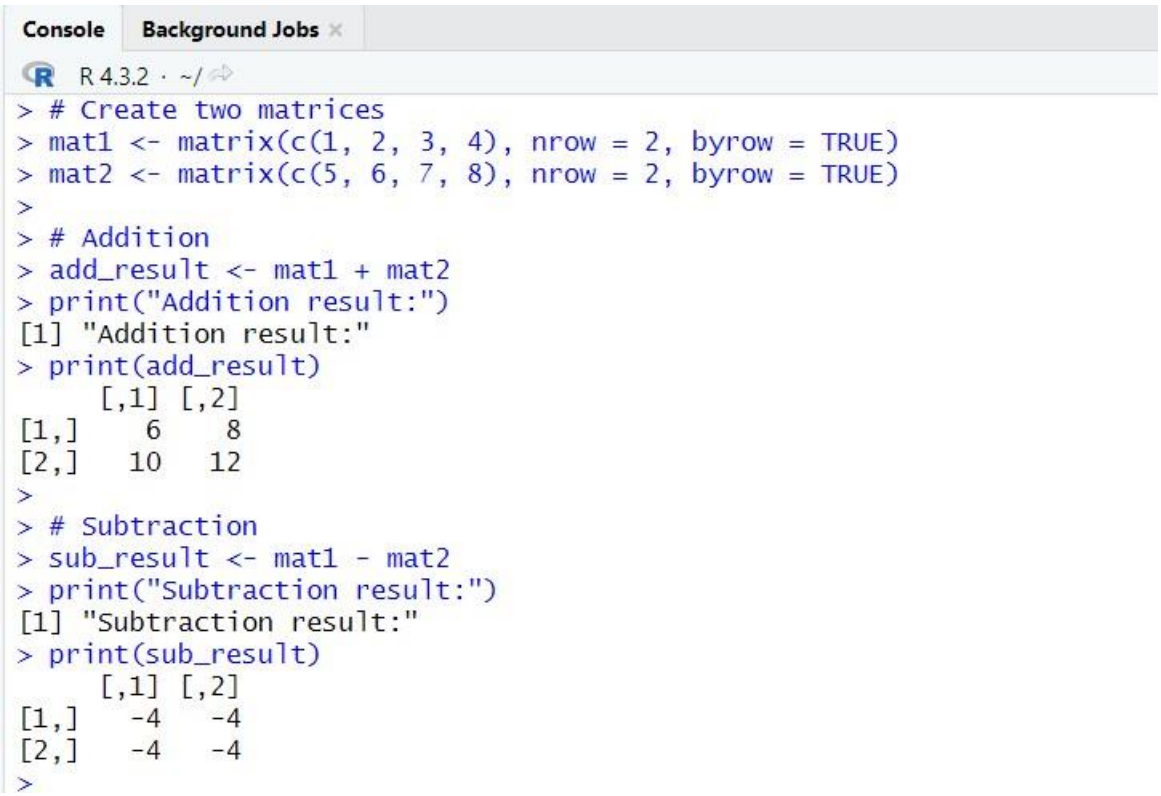
The resulting matrix will have dimensions equal to the number of rows in A and the number of columns in B.

In R, you use the %*% operator for matrix multiplication.

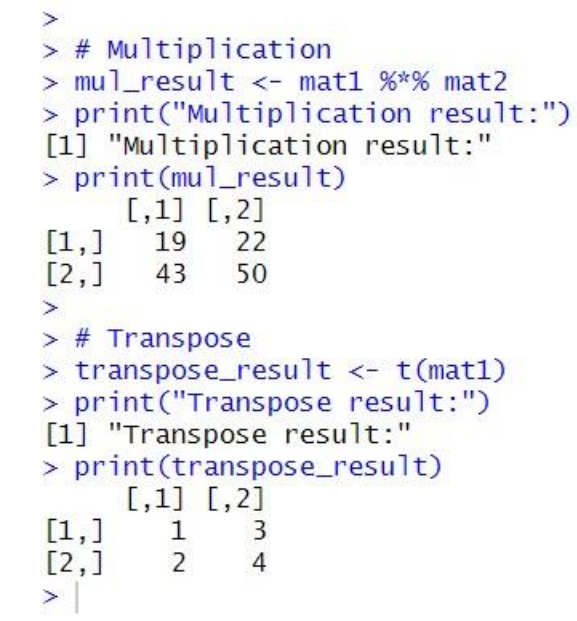
CODE:-

```
# Create two matrices mat1 <- matrix(c(1, 2, 3, 4), nrow =
2, byrow = TRUE) mat2 <- matrix(c(5, 6, 7, 8), nrow = 2,
byrow = TRUE)
# Addition add_result <-
mat1 + mat2 print("Addition
result:") print(add_result)
# Subtraction sub_result <-
mat1 - mat2
print("Subtraction result:")
print(sub_result)

# Multiplication mul_result <-
mat1 %*% mat2
print("Multiplication result:")
print(mul_result)
#Transpose transpose_result
<- t(mat1) print("Transpose
result:")
print(transpose_result)
```

OUTPUT:-



```
Console  Background Jobs
R 4.3.2 · ~/
> # Create two matrices
> mat1 <- matrix(c(1, 2, 3, 4), nrow = 2, byrow = TRUE)
> mat2 <- matrix(c(5, 6, 7, 8), nrow = 2, byrow = TRUE)
>
> # Addition
> add_result <- mat1 + mat2
> print("Addition result:")
[1] "Addition result:"
> print(add_result)
     [,1] [,2]
[1,]    6    8
[2,]   10   12
>
> # Subtraction
> sub_result <- mat1 - mat2
> print("Subtraction result:")
[1] "Subtraction result:"
> print(sub_result)
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4
>
```



```
>
> # Multiplication
> mul_result <- mat1 %*% mat2
> print("Multiplication result:")
[1] "Multiplication result:"
> print(mul_result)
     [,1] [,2]
[1,]   19   22
[2,]   43   50
>
> # Transpose
> transpose_result <- t(mat1)
> print("Transpose result:")
[1] "Transpose result:"
> print(transpose_result)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
>
```

EXPERIMENT - 2

AIM:- . Exercises to find the Eigenvalues and eigenvectors in R.

THEORY:-

Eigenvalues:

Eigenvalues are scalars associated with a square matrix.

For a given square matrix A, an eigenvalue λ is a scalar such that there exists a non-zero vector $\mathbf{v}$ (the eigenvector) satisfying the equation $A\mathbf{v}=\lambda\mathbf{v}$.

Geometrically, multiplying a vector by a matrix transforms the vector, and an eigenvector remains in the same direction after the transformation, only scaled by its corresponding eigenvalue.

Eigenvectors:

Eigenvectors are non-zero vectors corresponding to eigenvalues.

An eigenvector $\mathbf{v}$ of a matrix A is a vector that remains in the same direction after multiplication by A.

Eigenvectors are not unique; any scalar multiple of an eigenvector is also an eigenvector.

The set of all eigenvectors corresponding to a particular eigenvalue forms an eigenspace.

CODE:-

```
# Power Iteration Method to compute eigenvalues and eigenvectors

power_iteration <- function(A, num_iterations = 1000, tol = 1e-6) { n <-

nrow(A)

x <- rep(1, n)  # Initial guess for eigenvector lambda <- 0

# Initial guess for eigenvalue


for (i in 1:num_iterations) {

x_old <- x x <- A %*% x

lambda_old <- lambda

lambda <- max(x) x <- x /

lambda

# Check for convergence if (abs(lambda

- lambda_old) < tol) { break }
```

```
}
return(list(eigenvalue = lambda, eigenvector = x))
}
# Example usage
A <- matrix(c(2, -1, -1, 2), nrow = 2, byrow = TRUE)
result <- power_iteration(A) print("Eigenvalue:")
print(result$eigenvalue) print("Eigenvector:")
print(result$eigenvector)
```

OUTPUT:-

```
> A <- matrix(c(2, -1, -1, 2), nrow = 2, byrow = TRUE)
> result <- power_iteration(A)
> print("Eigenvalue:")
[1] "Eigenvalue:"
> print(result$eigenvalue)
[1] 1
> print("Eigenvector:")
[1] "Eigenvector:"
> print(result$eigenvector)
     [,1]
[1,]    1
[2,]    1
>
```

EXPERIMENT - 3

AIM: Exercises to solve equations by Gauss elimination, Gauss Jordan Method and Gauss Seidel in Scilab.

THEORY:

Gauss Elimination

The method we use to perform the three types of matrix row operations on an augmented matrix obtained from a linear system of equations to find the solutions for such a system is known as the Gaussian elimination method.

Algorithm:

- Partial pivoting: Find the kth pivot by swapping rows, to move the entry with the largest absolute value to the pivot position. This imparts computational stability to the algorithm.
- For each row below the pivot, calculate the factor f which makes the kth entry zero, and for every element in the row subtract the fth multiple of the corresponding element in the kth row.
- Repeat above steps for each unknown. We will be left with a partial r.e.f. matrix.

CODE:

```
clc; // Clear command window

// Converting equations to augmented matrix in SciLab printf("Matrix
of Coefficients is\n");
A = [1 1 1; 1 2 3; 1 3 2]; // Example coefficients matrix disp(A);

printf("\nMatrix of Constants is\n"); B = [3; 0;
3]; // Example constants matrix disp(B);

C = [A, B]; // Augmented matrix
printf("\nAugmented Matrix is:\n");
disp(C);

n = 3; // Number of variables

for i = 1:n
   C(i, :) = C(i, :) / C(i, i);
   disp(C);

   for j = 1:n-1
if i + j < n+1
         C(i+j, :) = C(i+j, :) - C(i+j, i) * C(i, :);
```

```
    end
end
disp(C); end

z = C(3, 4); y = C(2, 4) -
C(2, 3) * z;
x = C(1, 4) - C(1, 3) * z - C(1, 2) * y;

printf("\nx = "); disp(x);
```

OUTPUT:

```
 Matrix of Coefficients is
    1.    1.    1.
    1.    2.    3.
    1.    3.    2.
Matrix of Constants is
    3.
    0.
    3.
Augmented Matrix is:
    1.    1.    1.    3.
    1.    2.    3.    0.
    1.    3.    2.    3.

    1.    1.    1.    3.
    1.    2.    3.    0.
    1.    3.    2.    3.

    1.    1.    1.    3.
    0.    1.    2.   -3.
    0.    2.    1.    0.

    1.    1.    1.    3.
    0.    1.    2.   -3.
    0.    2.    1.    0.

    1.    1.    1.    3.
    0.    1.    2.   -3.
    0.    0.   -3.    6.

    1.    1.    1.    3.
    0.    1.    2.   -3.
    0.    0.    1.   -2.

    1.    1.    1.    3.
    0.    1.    2.   -3.
    0.    0.    1.   -2.

x =
    4.
y =
    1.
z =
   -2.
```

Gauss Jordan

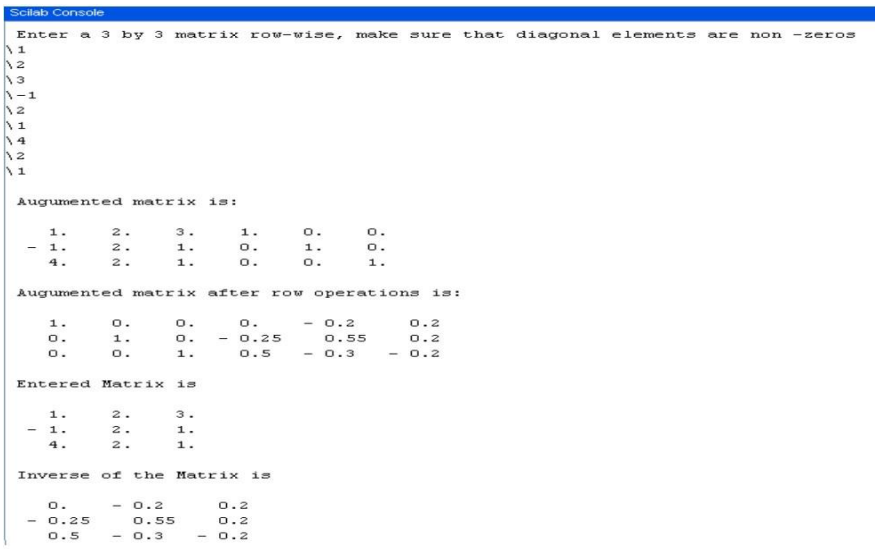
Gauss-Jordan Elimination is an algorithm used to solve systems of linear equations and to find the inverse of any invertible matrix. It relies upon three elementary row operations:

1. Swap the positions of two rows.
2. Multiply one row by a nonzero scalar.
3. Add or subtract a scalar multiple of one row to another row.

CODE:

```
clc; // Clear command window

disp('Enter a 3 by 3 matrix row-wise, make sure that diagonal elements are non-zeros')
A = zeros(3, 3); // Initialize matrix A
for i = 1:3     for j = 1:3
       A(i, j) = input(''); // Input each element of the matrix A      end
end

disp('Entered Matrix is:') disp(A);

if det(A) == 0
   disp('Matrix is singular, Inverse does not exist');
   break; end

// Taking the augmented matrix [A|I] B
= [A eye(3, 3)]; disp('Augmented
matrix is:'); disp(B);

// Making B(1,1)=1
B(1, :) = B(1, :) / B(1, 1);

// Making B(2,1) and B(3,1)=0
B(2, :) = B(2, :) - B(2, 1) * B(1, :);
B(3, :) = B(3, :) - B(3, 1) * B(1, :);

// Making B(2,2)=1 and B(1,2), B(3,2)=0
B(2, :) = B(2, :) / B(2, 2);
B(1, :) = B(1, :) - B(1, 2) * B(2, :);
B(3, :) = B(3, :) - B(3, 2) * B(2, :);

// Making B(3,3)=1 and B(1,3), B(2,3)=0
B(3, :) = B(3, :) / B(3, 3);
B(1, :) = B(1, :) - B(1, 3) * B(3, :);
B(2, :) = B(2, :) - B(2, 3) * B(3, :);

disp('Augmented matrix after row operations is:'); disp(B);

B(:, 1:3) = []; disp('Inverse of the
Matrix is:'); disp(B);
```

OUTPUT:



Gauss Seidel

Gauss-Seidel Method Theory:

The Gauss-Seidel method is an iterative technique used to solve a system of linear equations. It is particularly useful when dealing with large systems of equations, where direct methods like Gaussian elimination can be computationally expensive and prone to large round-off errors.

The basic idea behind the Gauss-Seidel method is to repeatedly update the estimates for the unknown variables until convergence is reached. Unlike direct methods that solve the entire system of equations at once, the Gauss-Seidel method updates the solution one variable at a time, using the most recent estimates for the other variables.

CODE:

```
clear, clc

a = [3 -2 0; 4 -13 3; 0 -3 18];
b = [4; 0; 1];

n = length(b);
for i = 1:n
 j = 1:n;
  j(i) = [];
   B = abs(a(i,j));
//disp(sum(B))     if
abs(a(i,i)) <= sum(B)
      disp("Matrix is not diagonally dominant");     end
end
```

```
tol = 1e-4;  iter =
1; itmax = 8; x =
zeros(n, 1); k =
1;
xold = x;

while k <= itmax
   x(1) = (b(1) - a(1,2:n)*xold(2:n)) / a(1,1);     for i =
2:n-1
      x(i) = (b(i) - a(i,1:i-1)*x(1:i-1) - a(i,i+1:n)*xold(i+1:n)) / a(i,i);     end
   x(n) = (b(n) - a(n,1:n-1)*x(1:n-1)) / a(n,n);

   if abs((x - xold) ./ x) < tol
disp(x - xold);         break;
   end

   xold = x;
k = k + 1;
   iter = iter + 1; end

disp(x(1), "Current I1 (in Amperes) is = "); disp(x(2),
"Current I2 (in Amperes) is = "); disp(x(3), "Current I3 (in
Amperes) is = "); disp("Number of iterations are", iter);
```

OUTPUT:

Scilab 6.1.1 Console

```
  0.0000645
  0.0000236
  0.0000039

  1.7061939

 "Current I1 (in Amperes) is = "

  0.5593144

 "Current I2 (in Amperes) is = "

  0.1487746

 "Current I3 (in Amperes) is = "

 "Number of iterations are"

  8.
```

EXPERIMENT - 4

AIM: Exercises to implement the associative, commutative and distributive property in a matrix in Scilab.

Theory: Matrices in R follow specific rules for addition and scalar multiplication. These rules are essential for performing accurate matrix calculations. Here, we'll explore three fundamental properties:

Associative Property (Addition): (A + B) + C = A + (B + C). This means the order of adding three matrices doesn't affect the final result.

Commutative Property (Addition): A + B = B + A. The order of adding two matrices doesn't change the sum. (Note: This applies to addition only, not multiplication).

Distributive Property (Scalar Multiplication): k(A + B) = kA + kB. Multiplying a scalar (number) by the sum of two matrices is the same as multiplying the scalar by each matrix individually and then adding the products.

CODE:
Associative Property:

```
// Define matrices A, B, and C
A = [1 2; 3 4];
B = [5 6; 7 8];
C = [9 10; 11 12];
// Left-hand side: (A * B) * C
left_hand_side = (A * B) * C;
// Right-hand side: A * (B * C)
right_hand_side = A * (B * C);
// Check if both sides are equal disp("Left-
hand side:"); disp(left_hand_side);
disp("Right-hand side:");
disp(right_hand_side);
```

```
"Left-hand side:"

 413.   454.
 937.   1030.

"Right-hand side:"

 413.   454.
 937.   1030.

-->
```

Commutative Property:

```
// Define matrices A and B
A = [1 2; 3 4];
B = [5 6; 7 8];
// Left-hand side: A * B
left_hand_side = A * B;
// Right-hand side: B * A
```

```
right_hand_side = B * A;
// Check if both sides are equal disp("Left-
hand side:"); disp(left_hand_side);
disp("Right-hand side:");
disp(right_hand_side);
```

```
"Left-hand side:"

 19.   22.
 43.   50.

"Right-hand side:"

 23.   34.
 31.   46.
```

Distributive Property:

```
// Define matrices A, B, and C
A = [1 2; 3 4];
B = [5 6; 7 8];
C = [9 10; 11 12];
// Left-hand side: A * (B + C)
left_hand_side = A * (B + C);
// Right-hand side: A * B + A * C
right_hand_side = A * B + A * C;
// Check if both sides are equal disp("Left-
hand side:"); disp(left_hand_side);
disp("Right-hand side:");
disp(right_hand_side);
```

```
"Left-hand side:"

 50.    56.
 114.   128.

"Right-hand side:"

 50.    56.
 114.   128.

-->
```

EXPERIMENT-5

AIM:- Exercises to find the reduced row echelon form of a matrix in R

THEORY:-

Reduced row echelon form (RREF) is a form of a matrix where:

- The leading entry (the first nonzero entry) in each row is 1.
- The leading 1 in each row occurs to the right of the leading 1 in the previous row.
  - All entries in a column containing a leading 1 are zero, except for the leading 1.

Here are the general steps to find the RREF of a matrix:

- Start with the leftmost nonzero column.
- This column will be the pivot column for the first row.
- Make the first nonzero entry in this column 1 by scaling the row appropriately.
- Use row operations to make all other entries in the pivot column zero.
- Move to the next column and repeat the process, treating the row you're working on as the first row.
- Continue until all nonzero rows are in RREF, and each pivot column has only zeros above and below the leading 1.

CODE:-

```
# Install and load necessary packages
if (!requireNamespace("pracma", quietly = TRUE)) {
  install.packages("pracma")
}
if (!requireNamespace("MASS", quietly = TRUE)) {
 install.packages("MASS")
}
library(pracma)  library(MASS)

# Define a function to compute RREF using pracma package
compute_RREF_pracma <- function(A) {    return(rref(A))
}

# Define a function to compute RREF using MASS package
compute_RREF_MASS <- function(A) {    return(ginv(A))
}
```

```
# Example matrix
A <- matrix(c(1, 2, 1, 3, 4, 5, 2, 2, 2), nrow = 3, byrow = TRUE)

# Compute RREF using pracma package RREF_A_pracma <-
compute_RREF_pracma(A) print("Reduced Row Echelon Form
(pracma):")
print(RREF_A_pracma)

# Compute RREF using MASS package  RREF_A_MASS <-
compute_RREF_MASS(A)  print("Reduced Row Echelon
Form (MASS):")  print(RREF_A_MASS)
```

OUTPUT:-

```
> print("Reduced Row Echelon Form (pracma):")
[1] "Reduced Row Echelon Form (pracma):"
> print(RREF_A_pracma)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
>
> # Compute RREF using MASS package
> RREF_A_MASS <- compute_RREF_MASS(A)
> print("Reduced Row Echelon Form (MASS):")
[1] "Reduced Row Echelon Form (MASS):"
> print(RREF_A_MASS)
     [,1]          [,2] [,3]
[1,] -0.5 -5.000000e-01  1.5
[2,]  1.0 -5.551115e-17 -0.5
[3,] -0.5  5.000000e-01 -0.5
>
```

EXPERIMENT-6

AIM:- Exercises to plot the functions and to find its first and second derivatives in R

THEORY:-

Plotting Functions:

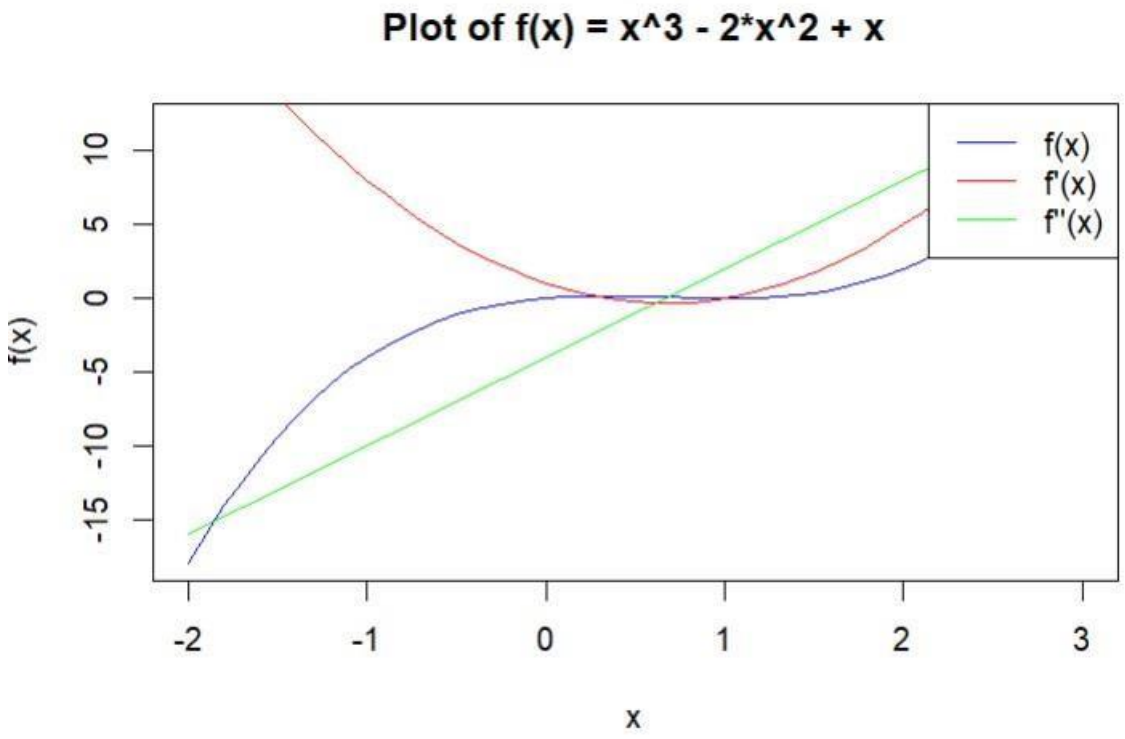
Plotting functions involves representing mathematical relationships graphically. In R, you can use various plotting libraries like ggplot2, base, or lattice. Here's a brief overview:

- Define the Function: Start by defining the mathematical function you want to plot.
- Generate Data: Create a sequence of x-values over the range you want to plot. This will be the domain of your function.
- Evaluate the Function: Calculate the corresponding y-values for each x-value using the defined function. Plot the Function: Use a plotting function (e.g., plot() or ggplot()) to create the plot, specifying x-values and their corresponding yvalues.
- Customize the Plot: Add labels, titles, axis scales, colors, and other aesthetic elements to enhance readability.

CODE:-

```
# Define the function
 f <- function(x) {
return(x^3 - 2*x^2 + x)

}

# Generate x values
 x <- seq(-2, 3, by = 0.1)

# Compute y values for the function
y <- f(x)


# Plot the function

plot(x, y, type = "l", col = "blue", xlab = "x", ylab = "f(x)", main = "Plot of f(x) = x^3 - 2*x^2 + x")


# Find the first derivative of the function
 f_prime <- function(x) {    return(3*x^2 -
4*x + 1)

}

# Compute y values for the first derivative
y_prime <- f_prime(x)

# Plot the first derivative

lines(x, y_prime, col = "red")
```

```
 # Find the second derivative of the
function

f_double_prime <- function(x) {
return(6*x - 4)

}


# Compute y values for the second derivative

y_double_prime <- f_double_prime(x)


# Plot the second derivative lines(x, y_double_prime, col
= "green")


# Add legend  legend("topright", legend = c("f(x)", "f'(x)", "f''(x)"), col = c("blue", "red", "green"), lty
      = 1)
```



EXPERIMENT-7

AIM:- Exercises to present the data as a frequency table in R.

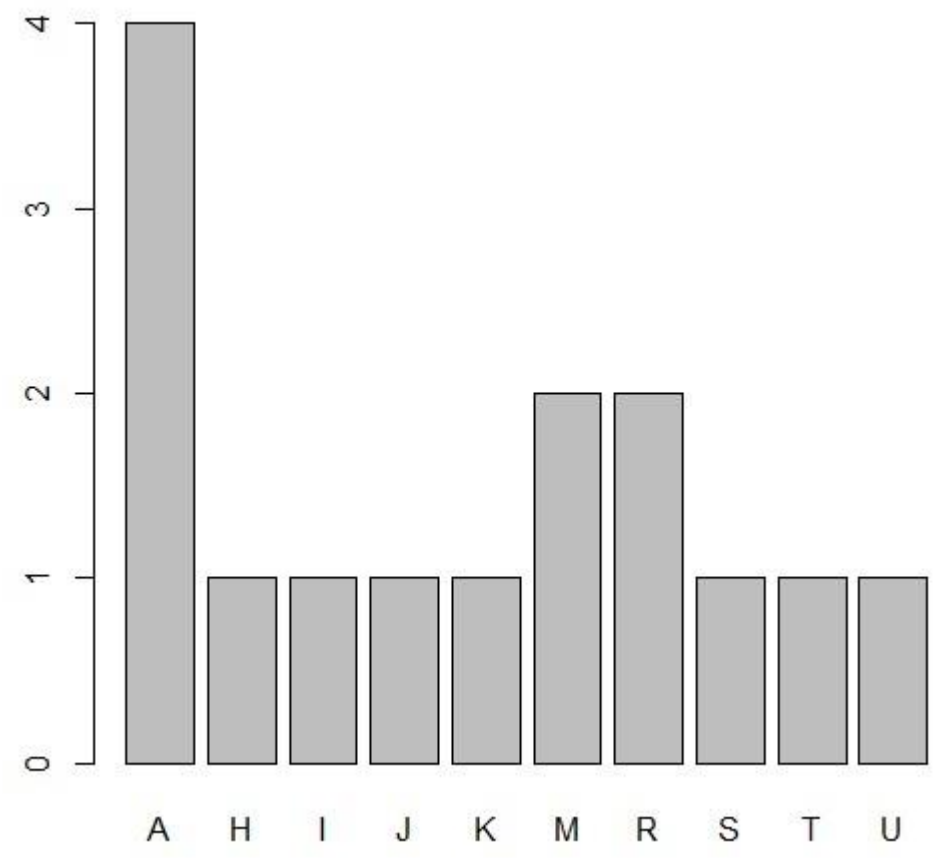
Theory: Data points far from the dataset's other points are considered outliers. This refers to the data values dispersed among other data values and upsetting the dataset's general distribution. Outlier detection is a statistical approach used to find outliers in datasets. Measurement errors, incorrect data entry, or really anomalous data values are just a few of the causes of outliers.

Code:-

```
#One-Way Frequency Tables

data<-c('A','J','I','T','K','U','M','A','R','S','H','A','R','M','A') freq_table <-

table(data)

#Method 1:Create Frequency Table in base R

print("simple frequency table : ") print(freq_table)

#Method 2:Create Frequency Table with Proportions prob_table <-

freq_table / sum(freq_table)      print("frequency table with

Proportions : ") print(prob_table)

#Method 3:Create Cumulative Frequency Table

cumsum_table <- cumsum(freq_table)

print("cumulative Frequency Table :")

print(cumsum_table) barplot(freq_table)
```

Output:-

```
[1] "simple frequency table : "
data
A H I J K M R S T U
4 1 1 1 1 2 2 1 1 1
[1] "frequency table with Proportions : "
data
         A          H          I          J          K          M          R
0.26666667 0.06666667 0.06666667 0.06666667 0.06666667 0.13333333 0.13333333
         S          T          U
0.06666667 0.06666667 0.06666667
[1] "cumulative Frequency Table :"
 A  H  I  J  K  M  R  S  T  U
 4  5  6  7  8 10 12 13 14 15
```



EXPERIMENT- 8

AIM:- Exercises to find the outliers in a dataset in R.

THEORY :- Data points far from the dataset's other points are considered outliers. This refers to the data values dispersed among other data values and upsetting the dataset's general distribution. Outlier detection is a statistical approach used to find outliers in datasets. Measurement errors, incorrect data entry, or really anomalous data values are just a few of the causes of outliers.

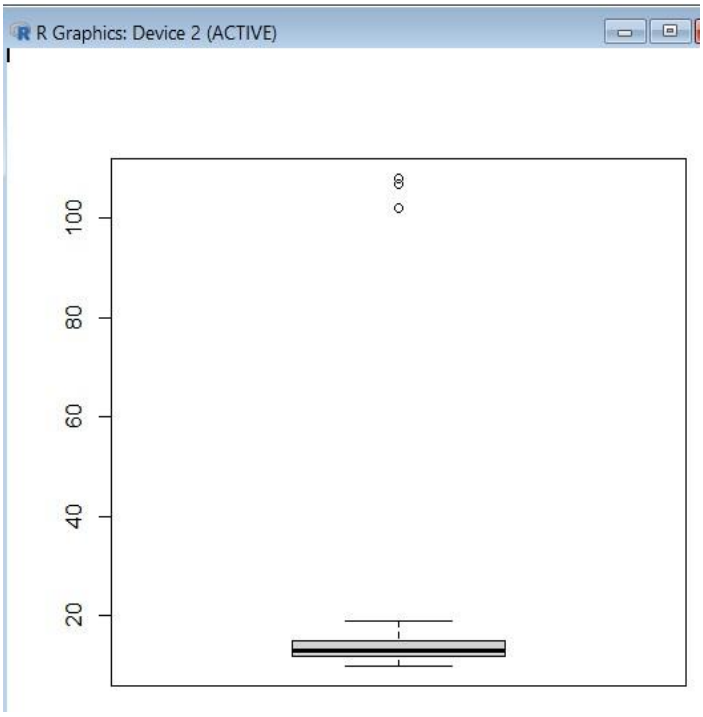
CODE :-

```
# Define your dataset dataset
<-
c(11,10,12,14,12,15,14,13,15,102,12,14,17,19,107,10,13,12,14,12,108,12,11,14,13,15,10,15,12,
10,14,13,15,10)

# Detecting outlier using Z score
detect_outliers <- function(data) {    outliers
<- vector()
  threshold <- 3    mean_value
<- mean(data)    std_dev <-
sd(data)
  for (i in data) {

    z_score <- abs((i - mean_value) / std_dev)
if (z_score > threshold) {        outliers <-
c(outliers, i)
    }

  }

  return(outliers)

}

outliers_z <- detect_outliers(dataset) print(outliers_z)


# Detecting outliers using Interquartile Range (IQR) detect_outliers_iqr <-
function(data) {

  outliers <- vector()

  q1 <- quantile(data, 0.25)
q3 <- quantile(data, 0.75)
iqr <- q3 - q1
  lower_bound <- q1 - 1.5 * iqr    upper_bound
<- q3 + 1.5 * iqr
  for (i in data) {      if (i < lower_bound | i >
upper_bound) {        outliers <- c(outliers, i)
    }
```

```
  }
  return(outliers)
}
outliers_iqr <- detect_outliers_iqr(dataset)
print(outliers_iqr) boxplot(dataset)
```

OUTPUT:-

```
[1] 102 107 108
[1] 102 107 108
```



EXPERIMENT – 9

Aim: The aim of this experiment is to determine the most risky project out of two mutually exclusive projects using statistical analysis in SPSS.

Theory:

When considering mutually exclusive projects, decision-makers often need to assess which project poses the highest risk. Risk can be measured in various ways, such as variance, standard deviation, or other statistical measures of uncertainty. In this experiment, we will compare the risk associated with two projects based on their historical or simulated data using SPSS.

CODE :

```
Begin data entry for Project A and Project B. data list
free / Project_A Project_B.
begin data 500
600
550 580
480 620
520 590
510 610
490 630
530 570
470 640
540 560 510
600 end
data.

* Compute the mean for Project A and Project B. compute Mean_A =
  mean(Project_A). compute Mean_B = mean(Project_B).

* Compute the standard deviation for Project A and Project B. compute
  SD_A = stddev(Project_A). compute SD_B = stddev(Project_B).
```

* Compute the coefficient of variation for Project A and Project B.

  compute CV_A = SD_A / Mean_A. compute CV_B = SD_B /

  Mean_B.

* Compare the coefficient of variation to determine the riskier project. if

  (CV_A > CV_B) Riskier_Project = 'Project A'. if (CV_B > CV_A)

  Riskier_Project = 'Project B'. if (CV_A = CV_B) Riskier_Project =

  'Both projects have equal risk'.

* Print the results.

Output:

The output of the experiment will be a statement indicating which project is deemed riskier based on the coefficient of variation analysis. For example, the output might be: "Project A is the riskier project" or "Both projects have equal risk" depending on the results of the analysis.

Conclusion:

This experiment provides a systematic approach to determine the most risky project out of two mutually exclusive projects using statistical analysis in SPSS. By comparing the coefficient of variation, decision-makers can better assess and manage the risks associated with different project

EXPERIMENT – 10

Aim: The aim of this experiment is to visualize data relationships using scatter diagrams and identify outliers, leverage points, and influential data points through residual plots in R.

Procedure:

Data Import:

Import the dataset containing variables of interest into R. Ensure that the dataset is properly formatted and contains the necessary variables for analysis.

Scatter Diagram:

Create a scatter plot to visualize the relationship between two variables of interest. Use the plot() function in R to generate the scatter plot. # Example scatter plot between variables X and Y

plot(X, Y, main = "Scatter Plot of X vs. Y", xlab = "X", ylab = "Y") **Linear**

Regression Model:

Fit a linear regression model to the data using the lm() function in R. This will be used to generate the predicted values and residuals.

Fit linear regression model

model <- lm(Y ~ X, data = dataset)

Residual Plots:

Create residual plots to assess the goodness-of-fit of the linear regression model and identify patterns in the residuals. Use the plot() function with the argument which = c(1, 2, 3) to generate multiple plots. **# Residual plots** par(mfrow = c(2, 2)) plot(model, which = c(1, 2, 3)) **Identifying Outliers:**

Identify outliers by examining the residual plot for extreme values that do not follow the general pattern. Outliers are points with large residuals compared to the majority of the data points.

Leverage Points:

Calculate leverage statistics to identify points with high leverage. Leverage points have extreme predictor values that can significantly influence the regression model.

 # Leverage statistics

leverage <- hatvalues(model)

Influential Data Points:

Identify influential data points using measures such as Cook's distance or DFFITS. Influential points have a large impact on the regression coefficients or predictions.

 # Cook's distance

cooks_distance <- cooks.distance(model)

Visualizing Outliers, Leverage, and Influential Points:

Overlay the identified outliers, leverage points, and influential points on the scatter plot for visualization.

 # Overlay outliers, leverage points, and influential points on the scatter plot points(X[outliers],

Y[outliers], col = "red", pch = 19)

points(X[leverage > threshold], Y[leverage > threshold], col = "blue", pch = 17)

points(X[cooks_distance > threshold], Y[cooks_distance > threshold], col = "green", pch = 15)

Conclusion:

This experiment demonstrates how to visualize data relationships using scatter diagrams, assess the goodness-of-fit of linear regression models using residual plots, and identify outliers, leverage points, and influential data points in R. By understanding these concepts, researchers can gain insights into their data and make informed decisions about their analysis.

Experiment-11

Aim: To calculate the correlation between two variables using R programming language.

Theory:

Correlation is a statistical technique that measures the strength and direction of the relationship between two variables. The correlation coefficient, denoted by r, ranges from -1 to 1. A value of 1 indicates a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 indicates no linear relationship between the variables.

Code:

```
# Generating sample data

x <- c(2, 4, 6, 8, 10)

y <- c(3, 6, 9, 12, 15)


# Calculating correlation

correlation <- cor(x, y)


# Printing correlation coefficient

print(paste("Correlation coefficient:", correlation))
```

Output:

Experiement-12

Aim: To perform time series analysis on a given dataset using R. This involves loading the data, visualizing it, fitting an ARIMA model, and making forecasts.

Theory:

Time series analysis involves analyzing data points collected over time to uncover patterns, trends, and seasonal variations. ARIMA (AutoRegressive Integrated Moving Average) models are widely used for time series forecasting. These models capture the autocorrelation in the data by incorporating past values, differences (to achieve stationarity), and moving average terms.

Code:

```
# Load time series data
 data <- c(23, 25, 30, 28, 32, 35, 36, 38, 40, 42, 45, 48, 50, 53, 55)


# Convert data to time series object
ts_data <- ts(data, start = c(2020, 1), frequency = 12)


# Visualize time series data
plot(ts_data, main = "Monthly Sales Data", xlab = "Year-Month", ylab = "Sales")


# Decompose time series data
 decomposed_data <-decompose(ts_data)


# Plot decomposed components
 plot(decomposed_data)


# Fit ARIMA model
arima_model <- auto.arima(ts_data)


# Summary of ARIMA model
summary(arima_model)
```

```
# Forecast future values using ARIMA model
forecast_values <- forecast(arima_model, h = 12)

# Plot forecasted values
plot(forecast_values, main = "Forecasted Sales", xlab = "Year-Month", ylab = "Sales")
```

Output:

```
Series: ts_data
ARIMA(1,0,0) with non-zero mean

Coefficients:
         ar1  intercept
      0.5117    38.9727
s.e.  0.1656     1.8489

sigma^2 estimated as 9.529:  log likelihood=-34.01
AIC=74.02   AICc=75.11   BIC=76.69

Training set error measures:
                    ME     RMSE      MAE       MPE     MAPE      MASE
Training set 0.294845 3.035134 2.460679 -1.616661 5.207079 0.2116857
                    ACF1
Training set -0.1652564
```

EXPERIMENT-13

AIM:- Exercises to implement linear regression using R.

THEORY:-

Linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. The main idea is to find the best-fitting straight line (or hyperplane in higher dimensions) that describes the relationship between the variables. Here's a brief overview of linear regression:

- **Dependent Variable (Response Variable)**: This is the variable we want to predict or explain.
- **Independent Variables (Predictors)**: These are the variables used to predict or explain the dependent variable.
- **Linear Relationship**: Linear regression assumes a linear relationship between the independent variables and the dependent variable. It means that the change in the dependent variable is proportional to the change in the independent variable(s).
- **Simple Linear Regression**: In simple linear regression, there's only one independent variable. The relationship between the independent and dependent variables is described by a straight line.
- **Multiple Linear Regression**: In multiple linear regression, there are multiple independent variables. The relationship is described by a hyperplane in higher dimensions.

CODE:-

```
# Step 1: Generate synthetic data set.seed(123)  # For

reproducibility n <- 100  # Number of observations x <-

rnorm(n)  # Independent variable y <- 2*x + 3 + rnorm(n)  #

Dependent variable (with noise)


# Combine data into a data frame data <-

data.frame(x, y)


# Step 2: Fit the linear regression model lm_model <-

lm(y ~ x, data = data)


# Step 3: View summary of the model

summary(lm_model)
```

Step 4: Make predictions (Optional)

For demonstration purpose, let's make predictions on the same data predictions <-

predict(lm_model)

Step 5: Evaluate the model (Optional)

For simplicity, let's just print the mean squared error mse <-

mean((predictions - data$y)^2) print(paste("Mean Squared

Error:", mse))

OUTPUT:-

```
Call:
lm(formula = y ~ x, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-1.9073 -0.6835 -0.0875  0.5806  3.2904

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.89720    0.09755   29.70   <2e-16 ***
x            1.94753    0.10688   18.22   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.9707 on 98 degrees of freedom
Multiple R-squared:  0.7721,    Adjusted R-squared:  0.7698
F-statistic:   332 on 1 and 98 DF,  p-value: < 2.2e-16

>
> # Step 4: Make predictions (Optional)
> # For demonstration purpose, let's make predictions on the same data
> predictions <- predict(lm_model)
>
> # Step 5: Evaluate the model (Optional)
> # For simplicity, let's just print the mean squared error
> mse <- mean((predictions - data$y)^2)
> print(paste("Mean Squared Error:", mse))
[1] "Mean Squared Error: 0.923441292618146"
>
```

EXPERIMENT-14

AIM:- Exercises to implement concepts of probability and distributions in R.

THEORY:-

Probability theory deals with the study of uncertainty or randomness in events. It provides a mathematical framework to quantify uncertainty and make predictions about the likelihood of different outcomes. Probability theory is widely used in various fields such as statistics, machine learning, finance, and physics. Here's a brief overview of probability theory and distributions in the context of R:

Probability Theory:

- **Probability**: Probability measures the likelihood of an event occurring. It ranges from 0 (impossible event) to 1 (certain event).
- **Random Variable**: A random variable is a variable whose possible values are outcomes of a random phenomenon. It can be discrete or continuous.
- **Probability Distribution**: A probability distribution describes the probabilities of possible outcomes of a random variable. It can be discrete (e.g., binomial, Poisson) or continuous (e.g., normal, exponential).
- **Cumulative Distribution Function (CDF)**: The CDF gives the probability that a random variable is less than or equal to a certain value.
- **Probability Mass Function (PMF)**: For discrete random variables, the PMF gives the probability that the random variable is equal to a specific value.
- **Probability Density Function (PDF)**: For continuous random variables, the PDF gives the probability density at a particular value.

Distributions in R:

- **Uniform Distribution**: A distribution where each value within a certain range is equally likely.
- **Normal Distribution**: A bell-shaped distribution characterized by its mean and standard deviation.
- **Binomial Distribution**: A distribution describing the number of successes in a fixed number of independent Bernoulli trials.
- **Poisson Distribution**: A distribution describing the number of events occurring in a fixed interval of time or space.
- **Exponential Distribution**: A distribution describing the time between events in a Poisson process.

Implementation in R:

- **Generating Random Numbers**: R provides functions like **runif()** (uniform), **rnorm()** (normal), **rbinom()** (binomial), **rpois()** (Poisson), and **rexp()** (exponential) to generate random numbers from different distributions.
- **Calculating Probabilities**: R provides functions like **pnorm()** (normal CDF), **dbinom()** (binomial PMF), **ppois()** (Poisson CDF), and **pexp()** (exponential CDF) to calculate probabilities.
- **Fitting Distributions**: R provides functions like **fitdistr()** in the **MASS** package to fit probability distributions to data.

CODE:-

```
# Load the MASS package for fitdistr function library(MASS)


# Generate 100 random numbers from a uniform distribution between 0 and 1
uniform_numbers <- runif(100) print("Uniform Numbers:")
print(uniform_numbers)


# Calculate the probability of a standard normal random variable being less than 1 prob_less_than_1
<- pnorm(1)
print("Probability of a standard normal random variable being less than 1:") print(prob_less_than_1)


# Calculate the probability of a standard normal random variable being between -1 and 1
prob_between_minus1_and_1 <- diff(pnorm(c(-1, 1))) print("Probability of a standard normal
random variable being between -1 and 1:") print(prob_between_minus1_and_1)


# Generate 100 random numbers from a normal distribution with mean 0 and standard deviation 1

normal_numbers <- rnorm(100, mean = 0, sd = 1) print("Normal Numbers:")

print(normal_numbers)


# Simulate 1000 coin flips with a fair coin (probability of heads = 0.5) coin_flips <-
sample(c("Heads", "Tails"), size = 1000, replace = TRUE, prob = c(0.5, 0.5)) print("Coin Flips:")
print(coin_flips)


# Define a vector of outcomes and their corresponding probabilities
outcomes <- c(1, 2, 3, 4, 5, 6) probabilities <- rep(1/6, 6)  # Fair six-sided
die


# Calculate the PMF of each outcome pmf <- dpois(outcomes, lambda = 3)  # PMF of
Poisson distribution with lambda = 3 print("Probability Mass Function:") print(pmf)
```

```
# Generate sample data from a normal distribution
sample_data <- rnorm(1000, mean = 5, sd = 2)
print("Sample Data:") print(sample_data)
# Fit a normal distribution to the sample data fit <-
fitdistr(sample_data, densfun = "normal")
# Print the estimated mean and standard deviation print("Estimated
Mean and Standard Deviation:") print(fit$estimate)
```

OUTPUT:-

```
> # Load the MASS package for fitdistr function
> library(MASS)
>
> # Generate 100 random numbers from a uniform distribution between 0 and 1
> uniform_numbers <- runif(100)
> print("Uniform Numbers:")
[1] "Uniform Numbers:"
> print(uniform_numbers)
  [1] 0.64732548 0.57095688 0.35474325 0.97108114 0.48185979 0.35337224 0.44527589 0.24297427 0.02039458
 [10] 0.30583225 0.46488476 0.55398142 0.96432615 0.92777850 0.96780160 0.72508712 0.06654035 0.01902744
 [19] 0.51217252 0.64989885 0.01879524 0.96096581 0.27699507 0.70223709 0.95886445 0.70510291 0.79735264
 [28] 0.47406278 0.36571280 0.07243124 0.34631470 0.42355374 0.46737690 0.60521153 0.11838041 0.87725803
 [37] 0.53992891 0.72766231 0.20011359 0.88772453 0.08777110 0.06890286 0.20489802 0.68874427 0.85210150
 [46] 0.45796848 0.14615527 0.66540255 0.16916574 0.42060409 0.01004938 0.59222975 0.70609206 0.70256870
 [55] 0.53477313 0.07322317 0.34941445 0.89432586 0.81785423 0.80769549 0.50816467 0.86227243 0.71739101
 [64] 0.86618765 0.33704934 0.22858519 0.26280117 0.75971126 0.86493926 0.39180439 0.79770942 0.99473961
 [73] 0.81958743 0.83893398 0.66145671 0.61381202 0.45135587 0.08075046 0.19250362 0.15853239 0.97606767
 [82] 0.25301259 0.56132800 0.98075741 0.36543555 0.53063507 0.96167056 0.99054400 0.95414101 0.70114185
 [91] 0.25266915 0.86062492 0.49896515 0.46587591 0.38743896 0.14755627 0.83062677 0.01391085 0.37712037
[100] 0.98912460
>
> # Calculate the probability of a standard normal random variable being less than 1
> prob_less_than_1 <- pnorm(1)
> print("Probability of a standard normal random variable being less than 1:")
[1] "Probability of a standard normal random variable being less than 1:"
> print(prob_less_than_1)
[1] 0.8413447
>
> # Calculate the probability of a standard normal random variable being between -1 and 1
> prob_between_minus1_and_1 <- diff(pnorm(c(-1, 1)))
> print("Probability of a standard normal random variable being between -1 and 1:")
[1] "Probability of a standard normal random variable being between -1 and 1:"
> print(prob_between_minus1_and_1)
[1] 0.6826895
>

>
> # Generate 100 random numbers from a normal distribution with mean 0 and standard deviation 1
> normal_numbers <- rnorm(100, mean = 0, sd = 1)
> print("Normal Numbers:")
[1] "Normal Numbers:"
> print(normal_numbers)
  [1]  0.272647072  0.041452123 -0.004881852 -0.976219229  0.144050084  0.456619219  0.037749934
  [8] -0.370973645  0.565148530  0.121622643  0.832894636 -1.023809939  1.947171223 -1.044154082
 [15] -0.491334929  0.750693303  0.389697614 -1.486406176 -0.631021883  0.727124612  2.149449998
 [22]  1.712917458  1.954157995  1.141655266  0.485864527 -0.278109908  0.673627712 -0.613720157
 [29] -2.003155864  1.212281013 -0.516169941  0.726821603 -1.725303028 -3.467490141 -0.113366223
 [36] -1.447171945 -0.369244736  0.339561339 -0.536169019  0.030225685 -0.724943076 -1.097239519
 [43] -0.236897259  1.012892986 -1.101558830  0.875086861 -0.543361286 -1.325192575  0.634937003
 [50]  0.730203038  1.166521543  0.592739363 -0.042944646  1.521016712 -0.437363049 -0.085210563
 [57] -0.330301354  0.745570490 -0.375118979  0.651792945 -1.434744213 -0.779157634  0.293401899
 [64]  0.193904036 -0.699101703  1.391548865  0.819058634  0.086194420  0.295126009 -0.267514353
 [71]  1.217317078  0.370593705 -2.000720942  0.031763939 -0.049580863 -1.017214275 -0.369167110
 [78]  1.715967126  2.535347076  1.089658414 -0.444511549  0.992464847  0.700825360  1.757615058
 [85] -0.649686187  0.886951944  2.619141869 -2.101744072  0.877830613  1.322032814  0.488794900
 [92]  0.574214543 -0.754463269 -0.735498838 -1.025130966 -1.719262364  0.432840582  0.110807156
 [99] -0.029487083  0.157066618
>
```

```
>
> # Fit a normal distribution to the sample data
> fit <- fitdistr(sample_data, densfun = "normal")
>
> # Print the estimated mean and standard deviation
> print("Estimated Mean and Standard Deviation:")
[1] "Estimated Mean and Standard Deviation:"
> print(fit$estimate)
    mean       sd
4.904987 1.961980
> |
```