EXP.NO: 1 DATE:	**GETTING USED TO R: DESCRIBING DATA**

AIM:

To write a R program that calculates and summary statistics such as the mean, median and standard deviation based on the student grades.

ALGORITHM:

- Step 1: Create a vector grades that contains a set of grades.
- Step 2: Use the mean() function to calculate the mean (average) grade.
- Step 3: Use the median() function to find the median (middle value) of the grades.
- Step 4: Use sd() to calculate the standard deviation, which shows how spread out the grades are.
- Step 5: Use the summary() function to get a quick summary of the dataset, including min, 1st quartile, median, mean, 3rd quartile, and max values.

PROGRAM:

```
grades <- c(85, 90, 78, 92, 88, 76, 95, 89, 84, 91)
mean_grade <- mean(grades)
cat(“Mean of the grades:”, mean_grade, “\n”)
median_grade <- median(grades)
cat(“Median of the grades:”, median_grade,”\n”)
sd_grade <- sd(grades)
cat(“Standard Deviation of the grades:”, sd_grade,”\n”)
summary_stats <- summary(grades)
cat(“\nSummary of the grades:”)
print(summary_stats)
```

OUTPUT:

```
Mean of the grades: 86.8
Median of the grades: 88
Standard Deviation of the grades: 6.511528
```

RESULT:

Thus, R program was executed successfully.

EXP.NO: 2 DATE:	**CREATING AND DISPLAYING DATA**

AIM:

To create a R Program to Create a dataset to store information abouth the employees, including their names, ages, and salaries.

ALGORITHM:

- Step 1: We create three separate vectors:
    i. employee_names: Stores the names of employees.
    ii. employee_ages: Stores the ages of employees.
    iii. employee_salaries: Stores the salaries of employees.
- Step 2: We combine the vectors into a data frame using the data.frame() function, where each column corresponds to a specific attribute (name, age, or salary).
- Step 3: The print() function is used to display the dataset in the console.

PROGRAM:

```
employee_names <- c("John Doe", "Jane Smith", "Peter Johnson", "Emily Davis", "Michael Brown")

employee_ages <- c(30, 25, 45, 40, 35)

employee_salaries <- c(55000, 62000, 75000, 68000, 72000)

employee_data <- data.frame(Name = employee_names, Age = employee_ages, Salary = employee_salaries)

print("Employee Data:")

print(employee_data)
```

OUTPUT:

```
"Employee Data:"

Name  Age  Salary

1.  John Doe  30  55000

2. Jane Smith  25  62000

3. Peter Johnson  45  75000

4. Emily Davis  40  68000
```

```
5. Michael Brown  35  72000
```

RESULT:

Thus, R program was executed successfully.

EXP.NO: 3 DATE:	**CREATING AND MANIPULATING A LIST AND AN ARRAY**

AIM:

To create and manipulate a list and an array in R, demonstrating operations such as adding, removing, and updating elements for a set of students and their corresponding grades/scores.

ALGORITHM:

- Initialize the student list with their names and corresponding grades.
- Display the original list.
- Add a new student with a grade to the list.
- Remove a student from the list.
- Update an existing student&#39;s grade in the list.
- Check if a specific student is present in the list.
- Display the final list after all modifications.

For the array part:

- Initialize the student score array for three subjects.
- Display the original array.
- Update the score of an existing student.
- Add a new student with scores to the array.
- Remove an existing student from the array.

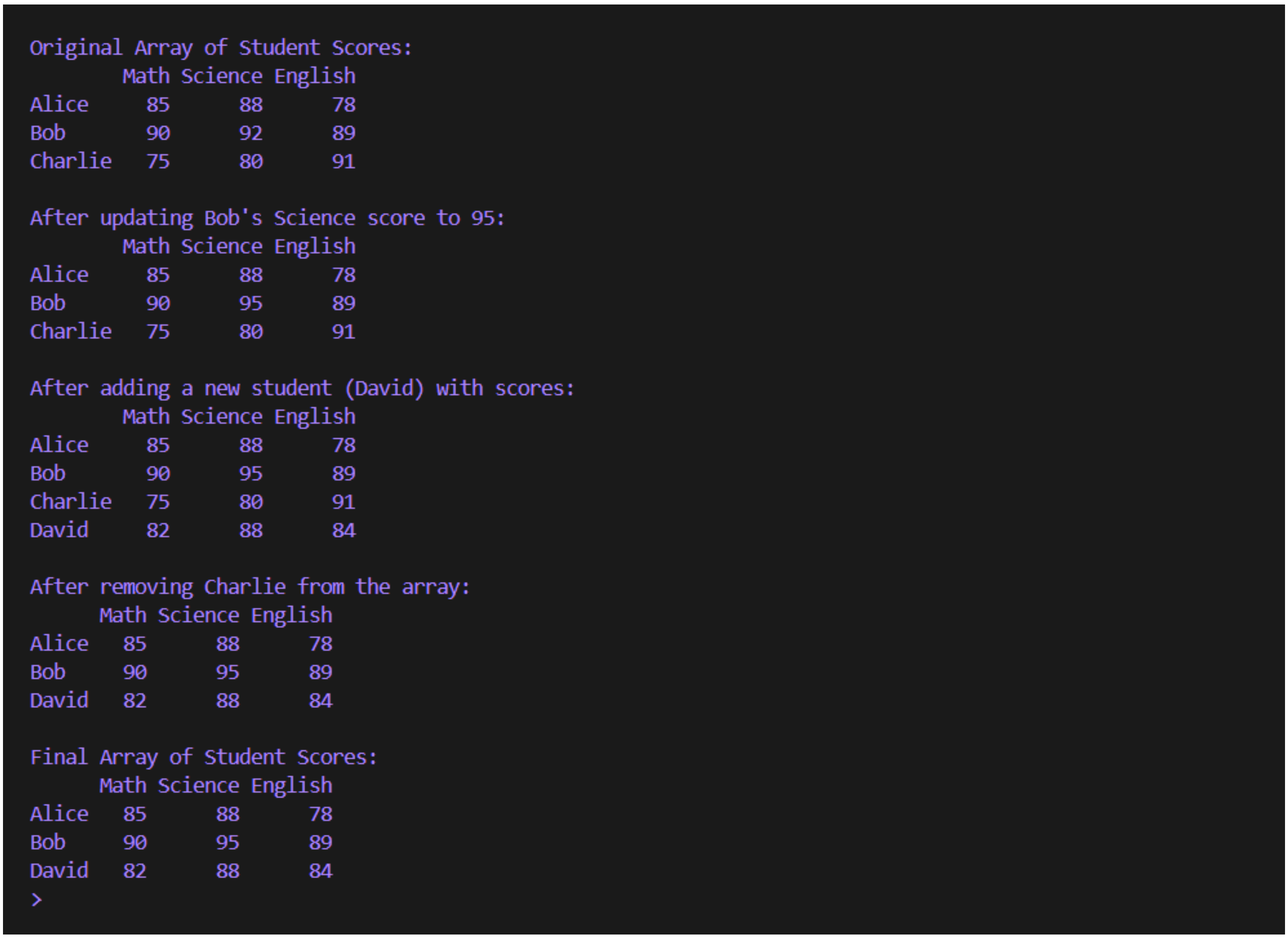
Display the final array after all changes.

PROGRAM:

```
students_scores <- array(c(85, 90, 75, 88, 92, 80, 78, 89, 91),
                    dim = c(3, 3),
                    dimnames = list(c("Alice", "Bob", "Charlie"),
                                c("Math", "Science", "English")))
cat("\nOriginal Array of Student Scores:\n")
print(students_scores)

students_scores["Bob", "Science"] <- 95
```

```r
cat("\nAfter updating Bob's Science score to 95:\n")
print(students_scores)

new_student_scores <- c(82, 88, 84) # Scores for Math, Science, and English
students_scores <- rbind(students_scores, David = new_student_scores)
cat("\nAfter adding a new student (David) with scores:\n")
print(students_scores)

students_scores <- students_scores[-which(rownames(students_scores) == "Charlie"),]
cat("\nAfter removing Charlie from the array:\n")
print(students_scores)

cat("\nFinal Array of Student Scores:\n")
print(students_scores)
```

OUTPUT:



```
Original Array of Student Scores:
        Math Science English
Alice     85      88      78
Bob       90      92      89
Charlie   75      80      91

After updating Bob's Science score to 95:
        Math Science English
Alice     85      88      78
Bob       90      95      89
Charlie   75      80      91

After adding a new student (David) with scores:
        Math Science English
Alice     85      88      78
Bob       90      95      89
Charlie   75      80      91
David     82      88      84

After removing Charlie from the array:
      Math Science English
Alice   85      88      78
Bob     90      95      89
David   82      88      84

Final Array of Student Scores:
      Math Science English
Alice   85      88      78
Bob     90      95      89
David   82      88      84
>
```

RESULT:

The program successfully creates and manipulates a list of students with their grades, as well as an array of student scores. It demonstrates operations such as adding, removing, and updating both the list and array elements in R.

EXP.NO: 4 DATE:	**CREATING A DATA FRAME AND MATRIX-LIKE OPERATIONS ON A DATA FRAME.**

AIM:

To create a Data Frame in R and perform matrix-like operations such as accessing, modifying, and performing calculations on the Data Frame.

ALGORITHM:

1. Create a Data Frame with student names, their marks in different subjects, and their total

marks.

2. Display the original Data Frame.
3. Access specific rows or columns of the Data Frame (matrix-like operations).
4. Perform matrix-like operations:
    - Compute the sum of marks in each subject.
    - Calculate the average marks for each student.
    - Add a new column with the grade for each student based on their total marks.
    - Modify specific values in the Data Frame.

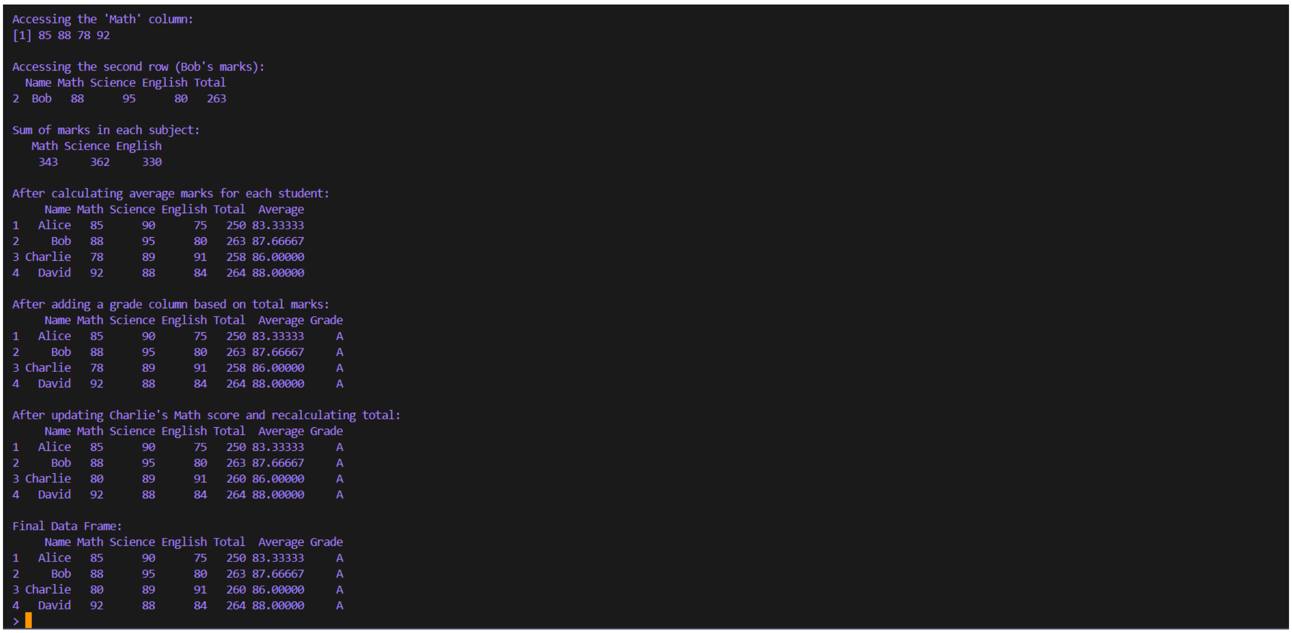
Display the final Data Frame after performing the operations.

PROGRAM:

```
students_df <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David"),
  Math = c(85, 88, 78, 92),
  Science = c(90, 95, 89, 88),
  English = c(75, 80, 91, 84),
  Total = c(85+90+75, 88+95+80, 78+89+91, 92+88+84)
)
cat("Original Data Frame:\n")
print(students_df)
```

```
cat("\nAccessing the 'Math' column:\n")
print(students_df$Math)

cat("\nAccessing the second row (Bob's marks):\n")
print(students_df[2, ])

subject_sums <- colSums(students_df[, 2:4])
cat("\nSum of marks in each subject:\n")
print(subject_sums)

students_df$Average <- rowMeans(students_df[, 2:4])
cat("\nAfter calculating average marks for each student:\n")
print(students_df)

students_df$Grade <- ifelse(students_df$Total >= 250, "A",
                     ifelse(students_df$Total >= 230, "B", "C"))
cat("\nAfter adding a grade column based on total marks:\n")
print(students_df)

students_df[students_df$Name == "Charlie", "Math"] <- 80
students_df$Total <- rowSums(students_df[, 2:4]) # Recalculate Total after modification
cat("\nAfter updating Charlie's Math score and recalculating total:\n")
print(students_df)

cat("\nFinal Data Frame:\n")
print(students_df)
```

OUTPUT:



```
Accessing the 'Math' column:
[1] 85 88 78 92

Accessing the second row (Bob's marks):
  Name Math Science English Total
2  Bob   88      95      80   263

Sum of marks in each subject:
   Math Science English
    343     362     330

After calculating average marks for each student:
     Name Math Science English Total  Average
1   Alice   85      90      75   250 83.33333
2     Bob   88      95      80   263 87.66667
3 Charlie   78      89      91   258 86.00000
4   David   92      88      84   264 88.00000

After adding a grade column based on total marks:
     Name Math Science English Total  Average Grade
1   Alice   85      90      75   250 83.33333     A
2     Bob   88      95      80   263 87.66667     A
3 Charlie   78      89      91   258 86.00000     A
4   David   92      88      84   264 88.00000     A

After updating Charlie's Math score and recalculating total:
     Name Math Science English Total  Average Grade
1   Alice   85      90      75   250 83.33333     A
2     Bob   88      95      80   263 87.66667     A
3 Charlie   80      89      91   260 86.00000     A
4   David   92      88      84   264 88.00000     A

Final Data Frame:
     Name Math Science English Total  Average Grade
1   Alice   85      90      75   250 83.33333     A
2     Bob   88      95      80   263 87.66667     A
3 Charlie   80      89      91   260 86.00000     A
4   David   92      88      84   264 88.00000     A
>
```

RESULT:

The program successfully creates a Data Frame and performs matrix-like operations, such as accessing specific rows and columns, calculating sums and averages, modifying values, and adding new columns. The final Data Frame is displayed with the updated values and additional calculated fields.

EXP.NO: 5 a DATE:	**STRING MANIPULATIONS**

AIM:

To extract email addresses from a custom dataset containing names and email information, and convert the emails into upper case using string manipulation techniques in R.

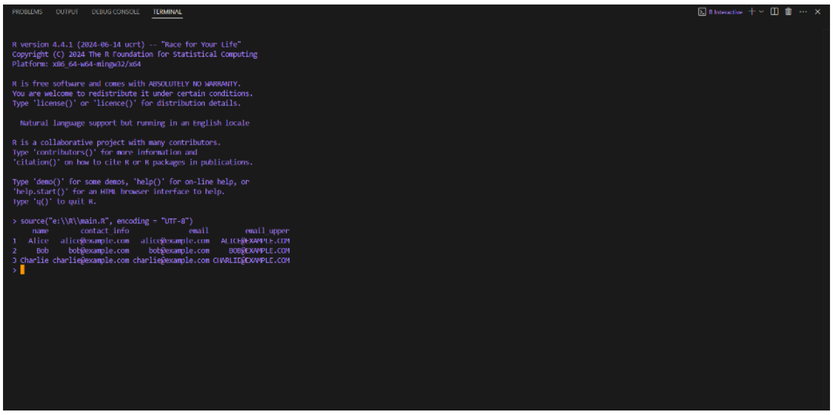
ALGORITHM:

- Load necessary libraries like stringr.
- Create or load a dataset with names and contact information.
- Use a regular expression with the str_extract() function to extract the email addresses from the contact information.
- Convert the extracted emails to upper case using the toupper() function.
- Store the resulting emails in a new column and print or visualize the dataset.

PROGRAM:

```
library(stringr)
data <- data.frame(
  name = c("Alice", "Bob", "Charlie"),
  contact_info = c("alice@example.com", "bob@example.com", "charlie@example.com")
)
data$email <- str_extract(data$contact_info, "\\S+@\\S+")
data$email_upper <- toupper(data$email)
print(data)
```

OUTPUT:



```
R version 4.4.1 (2024-06-14 ucrt) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> source("e:\\R\\main.R", encoding = "UTF-8")
     name        contact_info               email         email_upper
1   Alice   alice@example.com   alice@example.com   ALICE@EXAMPLE.COM
2     Bob     bob@example.com     bob@example.com     BOB@EXAMPLE.COM
3 Charlie charlie@example.com charlie@example.com CHARLIE@EXAMPLE.COM
> 
```

RESULT:

The output will be a dataset with a new column containing the extracted emails in upper case.

EXP.NO: 5 b DATE:	**STRING MANIPULATIONS**

AIM:

To write an R program that splits customer full names into first and last names and verifies if email addresses belong to a specific domain.

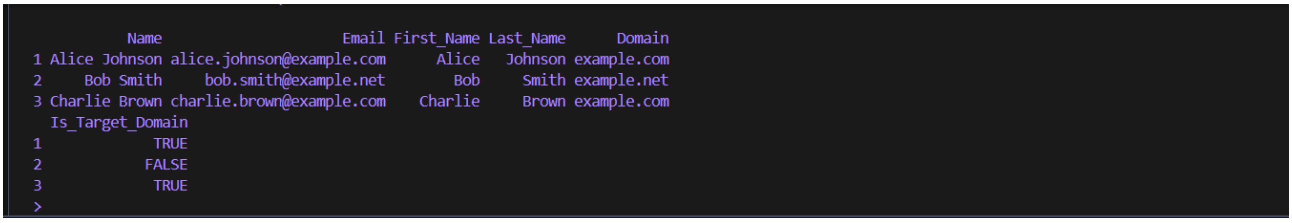
ALGORITHM:

- Start the program.
- Create a dataset with customer names and email addresses.
- Split the customer names into first and last names.
- Extract the domain names from the email addresses.
- Check if the domain matches a specific domain (e.g.,”example.com”).
- Display the manipulated data and whether the email domain matches.
- End the program.

PROGRAM:

```
library(dplyr)
customer_data <- data.frame(
  Name = c("Alice Johnson", "Bob Smith", "Charlie Brown"),
  Email = c("alice.johnson@example.com", "bob.smith@example.net",
"charlie.brown@example.com")
)
customer_data <- customer_data %>%
  mutate(
    First_Name = sapply(strsplit(as.character(Name), " "), `[`, 1),
    Last_Name = sapply(strsplit(as.character(Name), " "), `[`, 2)
  )
customer_data$Domain <- sub(".*@", "", customer_data$Email)
target_domain <- "example.com"
customer_data$Is_Target_Domain <- customer_data$Domain == target_domain
print(customer_data)
```

OUTPUT:



```
           Name                     Email First_Name Last_Name      Domain
1 Alice Johnson alice.johnson@example.com      Alice   Johnson example.com
2     Bob Smith     bob.smith@example.net        Bob     Smith example.net
3 Charlie Brown charlie.brown@example.com    Charlie     Brown example.com
  Is_Target_Domain
1             TRUE
2            FALSE
3             TRUE
>
```

RESULT:

The R program successfully splits customer names into first and last names and checks if the email domains match a specified domain (“example.com”). The output shows the manipulated data, including the first and last names and whether each email domain matches the target domain.

EXP.NO: 6 DATE:	**DATA TRANSPOSE OPERATIONS**

AIM:

To perform data transpose operations in R, converting rows into columns and vice versa.

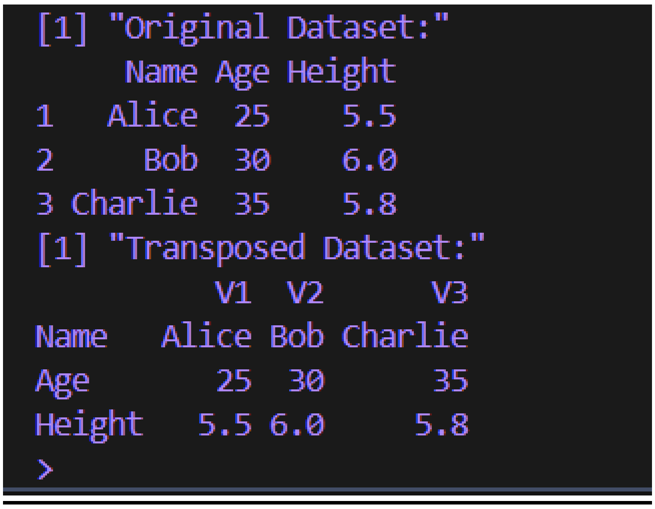
ALGORITHM:

- Load the Dataset: Load the dataset into R from an external source or define it manually within R.
- Transpose the Data: Use the t() function to transpose the data. The t() function in R transposes a matrix or dataframe, converting rows into columns and columns into rows.
- Convert Transposed Data (if necessary): If the data is not in a desired format(matrix to dataframe), convert the transposed result into a dataframe.
- Verify the Transposed Data: Print or display the transposed data to verify the operation.

PROGRAM:

```
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie"),
  Age = c(25, 30, 35),
  Height = c(5.5, 6.0, 5.8)
)
print("Original Dataset:")
print(data)
transposed_data <- t(data)
transposed_df <- as.data.frame(transposed_data)
print("Transposed Dataset:")
print(transposed_df)
```

OUTPUT:



```
[1] "Original Dataset:"
     Name Age Height
1   Alice  25    5.5
2     Bob  30    6.0
3 Charlie  35    5.8
[1] "Transposed Dataset:"
          V1  V2      V3
Name   Alice Bob Charlie
Age       25  30      35
Height   5.5 6.0     5.8
>
```

RESULT:

Thus, data transpose program was executed successfully.

EXP.NO: 7 DATE:	**NORMAL DISTRIBUTION SIMULATION**

AIM:

To model a random variable using a specific probability distribution, such as the normal distribution, and simulate data based on that distribution in R.

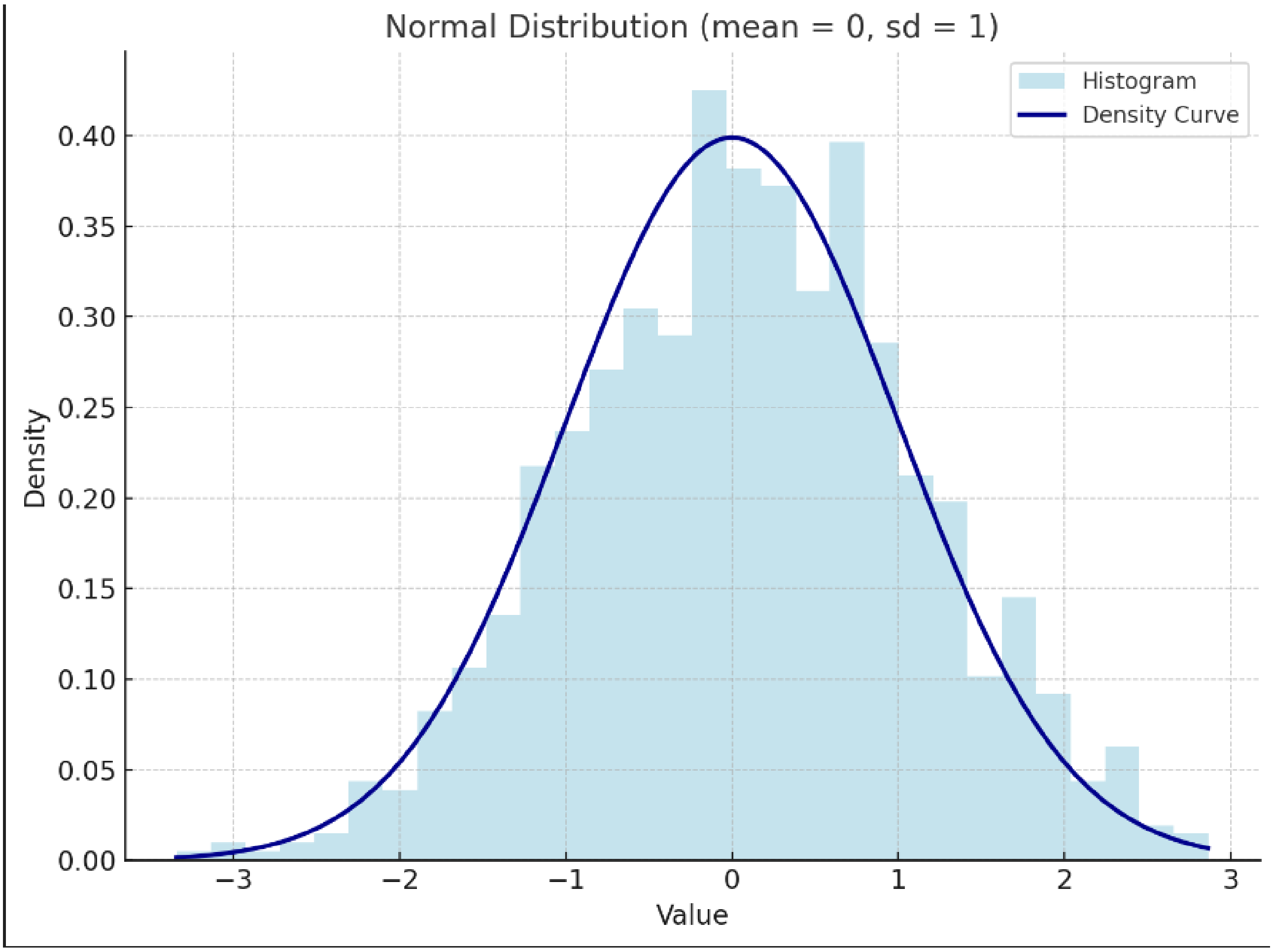
ALGORITHM:

- Identify the Probability Distribution: Choose the probability distribution to model the random variable, such as the normal distribution, binomial, Poisson, etc.
- Set Parameters for the Distribution: Define the parameters required for the distribution. For the normal distribution, these are the mean (μ) and standard deviation (σ).
- Generate Random Data: Use the appropriate function to generate random Data based on the selected distribution. For normal distribution: rnorm()
- Visualize the Data: Plot the data using a histogram or density plot to visually inspect the distribution.
- Validate the Distribution: Use summary statistics or other validation techniques to ensure the generated data follows the chosen distribution.

PROGRAM:

```
mean_value <- 0
sd_value <- 1
random_data <- rnorm(n = 1000, mean = mean_value, sd = sd_value)
print("Summary Statistics:")
summary(random_data)
hist(random_data, breaks = 30, probability = TRUE,
    main = "Normal Distribution (mean = 0, sd = 1)",
    xlab = "Value", col = "lightblue")
curve(dnorm(x, mean = mean_value, sd = sd_value),
     col = "darkblue", lwd = 2, add = TRUE)
```

OUTPUT:



RESULT:

Thus, the normal distribution program was executed successfully in R programming language.