**Experiment No : 01**

**Code :**

#Numeric Variables:

print("Numeric Variables:")

# Creating numeric variables

x <- 5

y <- 3.14

# Performing operations with numeric variables

sum <- x + y

product <- x \* y

# Printing the results

print(sum)

print(product)

print("-------------------------------------------------------------------------")

#Strings(Character Variables):

print("Strings-Character Variables:")

# Creating character variables

name <- "John"

city <- 'New York' # Single or double quotes can be used

# Concatenating strings

full\_name <- paste(name, "Doe")

print(full\_name)

# Accessing individual characters

first\_letter <- substr(name, 1, 1)

print(first\_letter)

print("-------------------------------------------------------------------------")

#Factors:

print("Factors:")

# Creating a factor variable

gender <- factor(c("Male", "Female", "Male", "Female"))

# Viewing the levels/categories of the factor

print(levels(gender))

# Assigning labels to factor levels

levels(gender) <- c("M", "F")

print(gender)

print("-------------------------------------------------------------------------")

#Data Types:

print("Data Types :")

# Checking the data type of variables

print(class(x))

print(class(name))

print(class(gender))

print("-------------------------------------------------------------------------")

#Vectors:

print("Vectors:")

# Creating vectors of different types

numeric\_vector <- c(1, 2, 3, 4, 5)

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

factor\_vector <- factor(c("low", "medium", "high"))

# Accessing elements of a vector

print(numeric\_vector[3])

print(char\_vector[2])

print(factor\_vector[1])

print("-------------------------------------------------------------------------")

#Dataframes:

print("Dataframes:")

# Creating a dataframe

data <- data.frame(

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

Age = c(25, 30, 22),

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

)

# Accessing dataframe elements

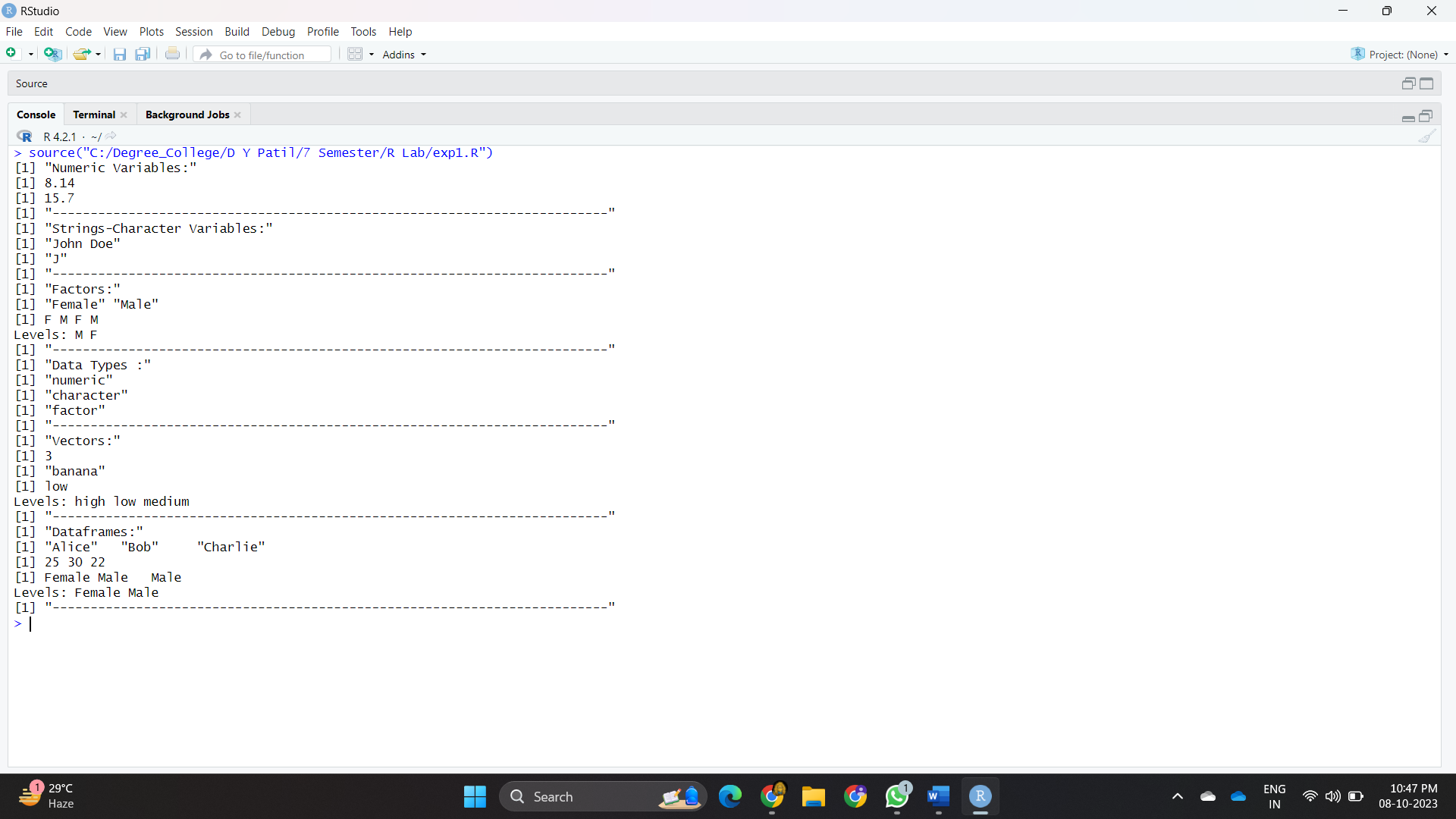
print(data$Name)

print(data$Age)

print(data$Gender)

print("-------------------------------------------------------------------------")

**Output :**



**Experiment No : 02**

**Code :**

custom\_stats <- function(data\_vector) {

# Calculate mean

mean\_val <- mean(data\_vector)

# Calculate median

median\_val <- median(data\_vector)

# Calculate mode (most frequent value)

mode\_val <- as.numeric(names(sort(table(data\_vector), decreasing = TRUE)[1]))

# Calculate standard deviation

sd\_val <- sd(data\_vector)

# Store results in a vector

result\_vector <- c(mean\_val, median\_val, mode\_val, sd\_val)

return(result\_vector)

}

# Example usage:

data\_vector <- c(1,2,2,3,4,5,8,8,7,6,6,1,1,9,9,5,5,2,2,4,3,2,1,5,2,6,8,4,10,10,11,12,4,6,13,11,12,15,15,10,11)

result <- custom\_stats(data\_vector)

# Print the results

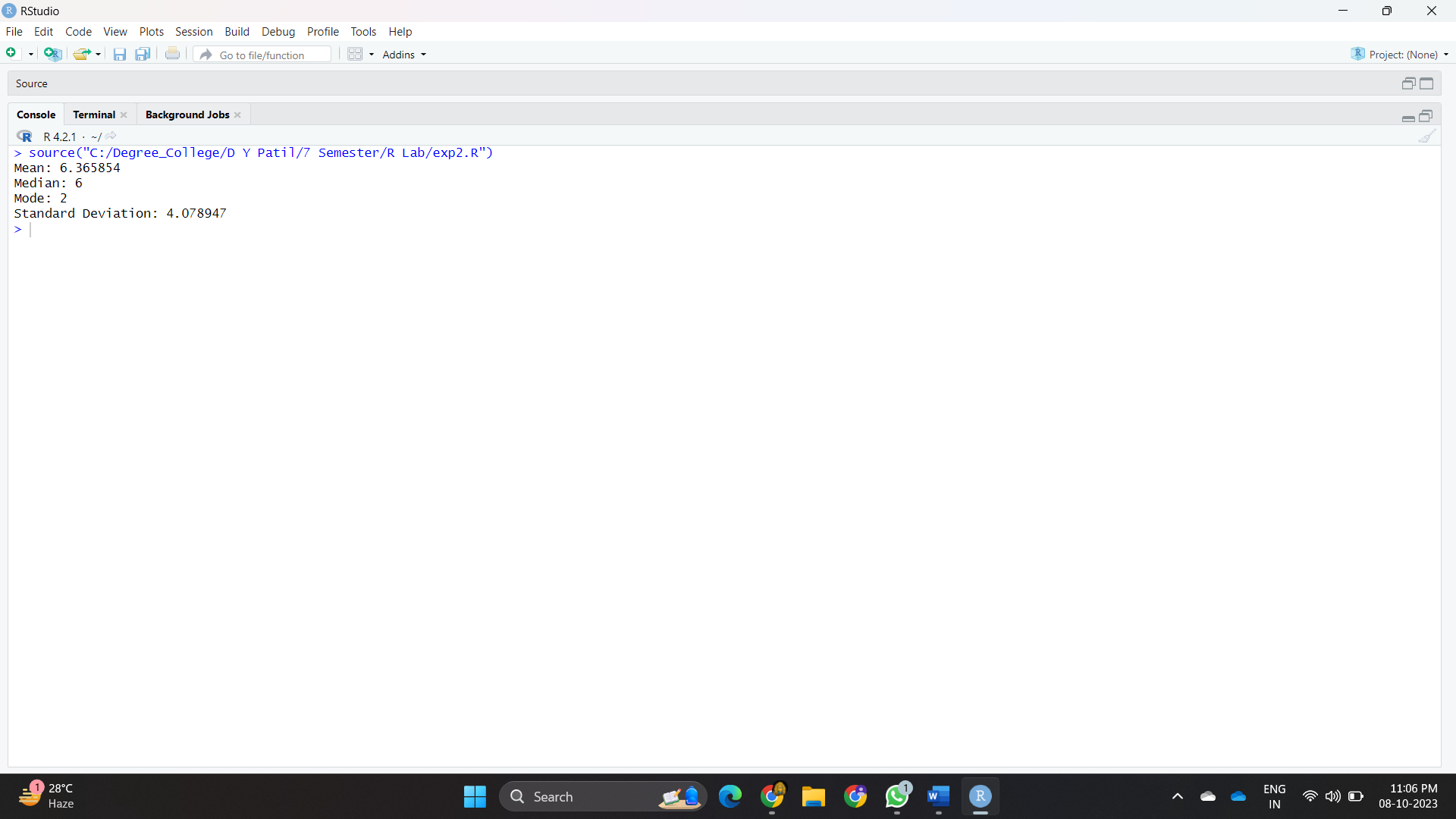
cat("Mean:", result[1], "\n")

cat("Median:", result[2], "\n")

cat("Mode:", result[3], "\n")

cat("Standard Deviation:", result[4], "\n")

**Output :**



**Experiment No : 03**

**Code :**

sales\_data <- read.csv("C:/Degree\_College/datasets/sales\_data.csv")

#Explore the Data:

head(sales\_data) # View the first few rows of the data

tail(sales\_data) # View the last few rows of the data

summary(sales\_data) # Get summary statistics for numeric columns

str(sales\_data) # Examine the structure of the data frame

sales\_data <- na.omit(sales\_data)

print(sales\_data)

sales\_data <- unique(sales\_data)

print(sales\_data)

total\_sales <- sum(sales\_data$SalesAmount)

print(total\_sales)

mean\_sales <- mean(sales\_data$SalesAmount)

print(mean\_sales)

median\_sales <- median(sales\_data$SalesAmount)

print(median\_sales)

max\_sales <- max(sales\_data$SalesAmount)

print(max\_sales)

min\_sales <- min(sales\_data$SalesAmount)

print(min\_sales)

library(ggplot2)

#Box Plot for Sales Amount by Product Category:

ggplot(sales\_data, aes(x = ProductCategory, y = SalesAmount)) +

geom\_boxplot() +

labs(title = "Sales Amount Distribution by Product Category", x = "Product Category", y = "Sales Amount")

#Histogram for Quantity Sold:

ggplot(sales\_data, aes(x = QuantitySold)) +

geom\_histogram(binwidth = 1, fill = "blue", color = "black") +

labs(title = "Distribution of Quantity Sold", x = "Quantity Sold", y = "Frequency")

#Histogram for Sales Amount:

ggplot(sales\_data, aes(x = SalesAmount)) +

geom\_histogram(binwidth = 100, fill = "green", color = "red") +

labs(title = "Distribution of Sales Amounts", x = "Sales Amount", y = "Frequency")

#Time Series Plot for Sales Over Time:

ggplot(sales\_data, aes(x = Date, y = SalesAmount)) +

geom\_line(color = "red") +

labs(title = "Sales Over Time", x = "Date", y = "Sales Amount")

#Bar Plot for Top Selling Products:

library(dplyr)

library(ggplot2)

top\_products <- sales\_data %>%

group\_by(ProductName) %>%

summarize(SalesCount = n()) %>%

arrange(desc(SalesCount)) %>%

head(10) # Change '10' to the desired number of top products

ggplot(data = top\_products, aes(x = reorder(ProductName, SalesCount), y = SalesCount)) +

geom\_bar(stat = "identity", fill = "brown") +

labs(title = "Top Selling Products", x = "Product Name", y = "Sales Count") +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))

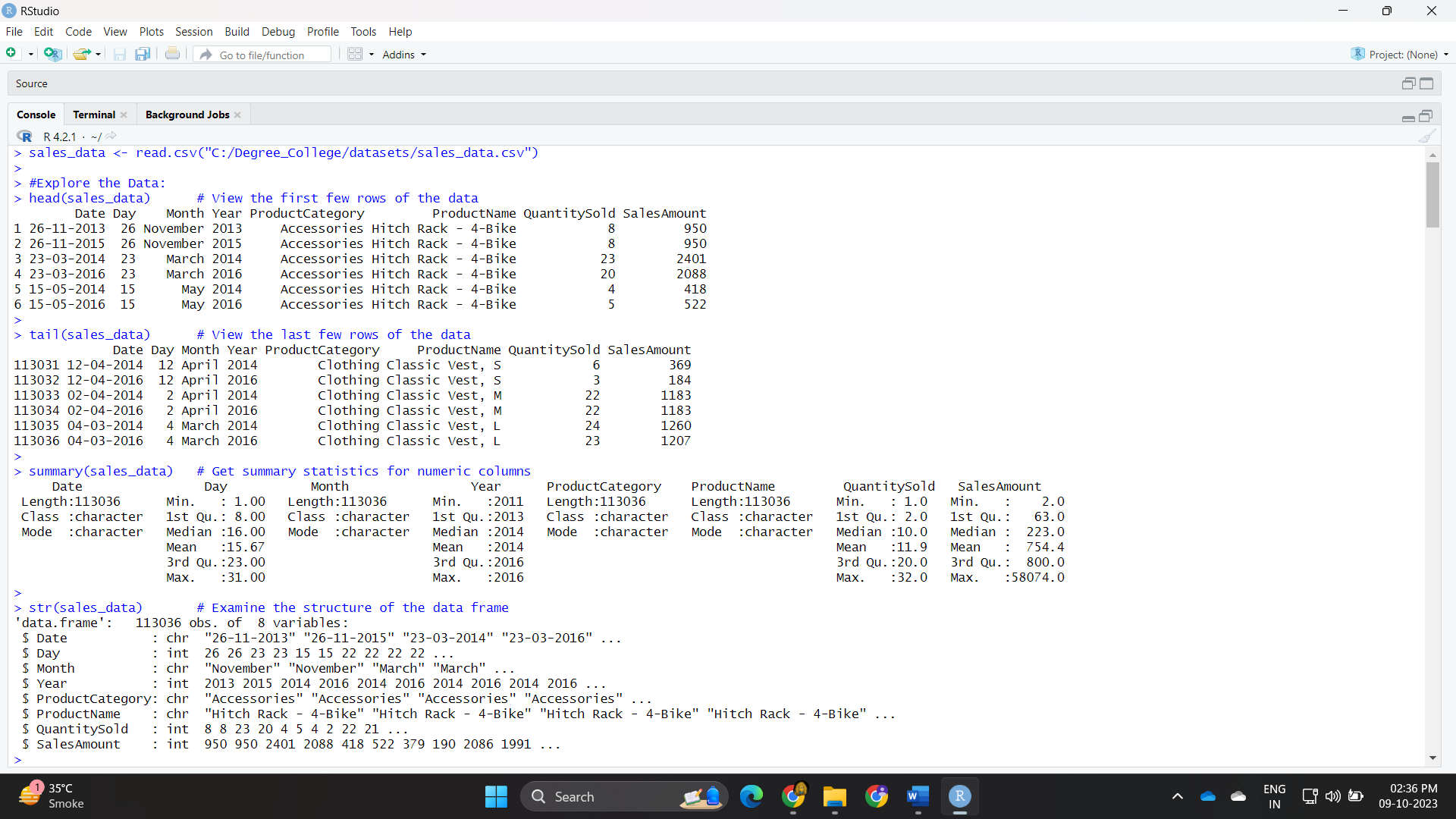
#Scatter Plot for Sales Amount vs. Quantity Sold:

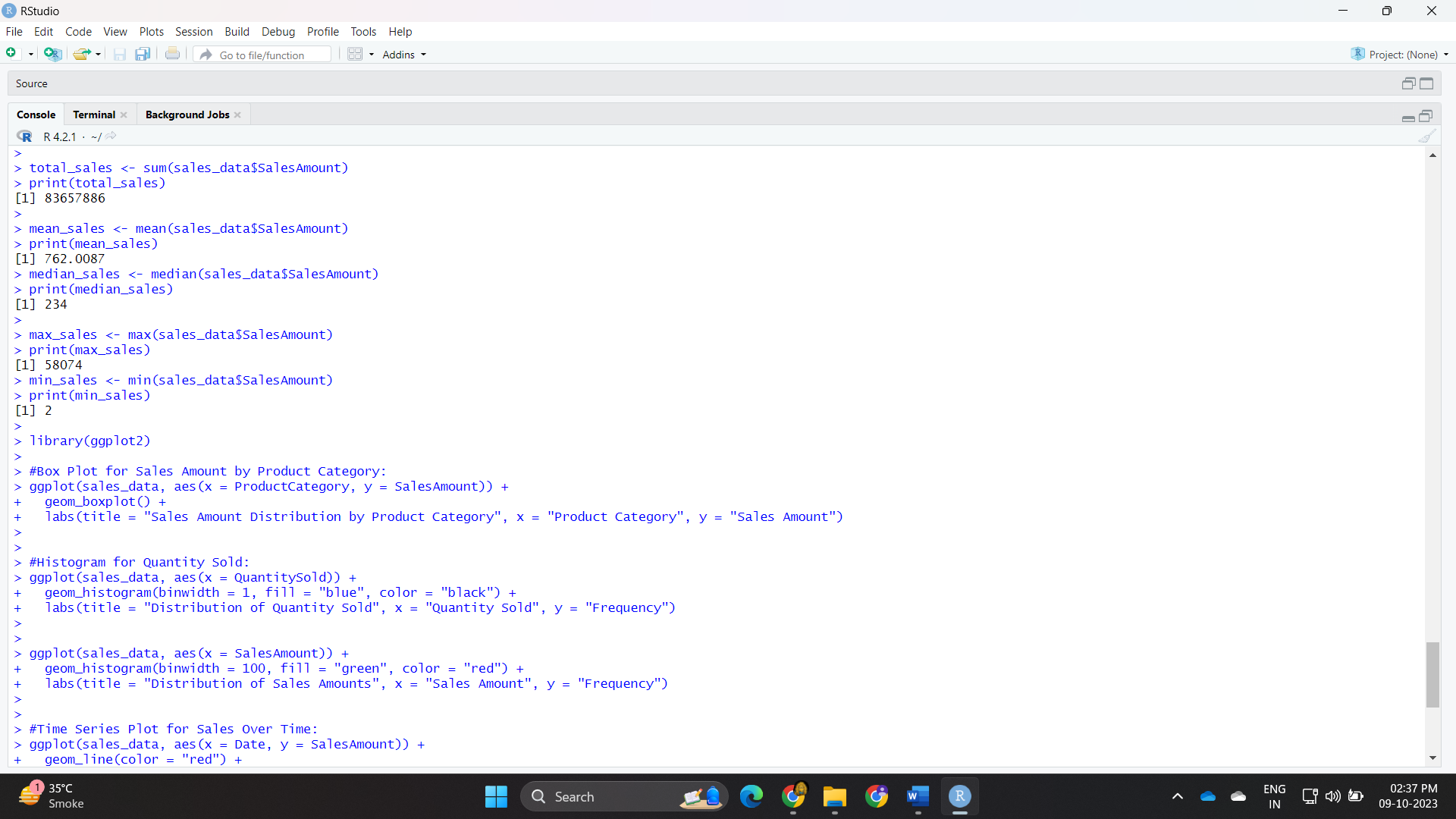
ggplot(sales\_data, aes(x = QuantitySold, y = SalesAmount)) +

geom\_point(color = "purple") +

labs(title = "Scatter Plot of Sales Amount vs. Quantity Sold", x = "Quantity Sold", y = "Sales Amount")

**Output :**





**Experiment No : 04**

**Code :**

# Load the necessary libraries

library(dplyr)

# Read the "sales\_data.csv" dataset

sales\_data <- read.csv("C:/Degree\_College/datasets/sales\_data.csv")

head(sales\_data) # View the first few rows of the data

tail(sales\_data) # View the last few rows of the data

summary(sales\_data) # Get summary statistics for numeric columns

str(sales\_data) # Examine the structure of the data frame

sales\_data <- distinct(sales\_data)

print(sales\_data)

sales\_data <- na.omit(sales\_data)

print(sales\_data)

summary\_stats <- summarise(sales\_data,

Mean\_Sales = mean(SalesAmount),

Median\_Sales = median(SalesAmount),

SD\_Sales = sd(SalesAmount),

Min\_Sales = min(SalesAmount),

Max\_Sales = max(SalesAmount),

Range\_Sales = max(SalesAmount)- min(SalesAmount),

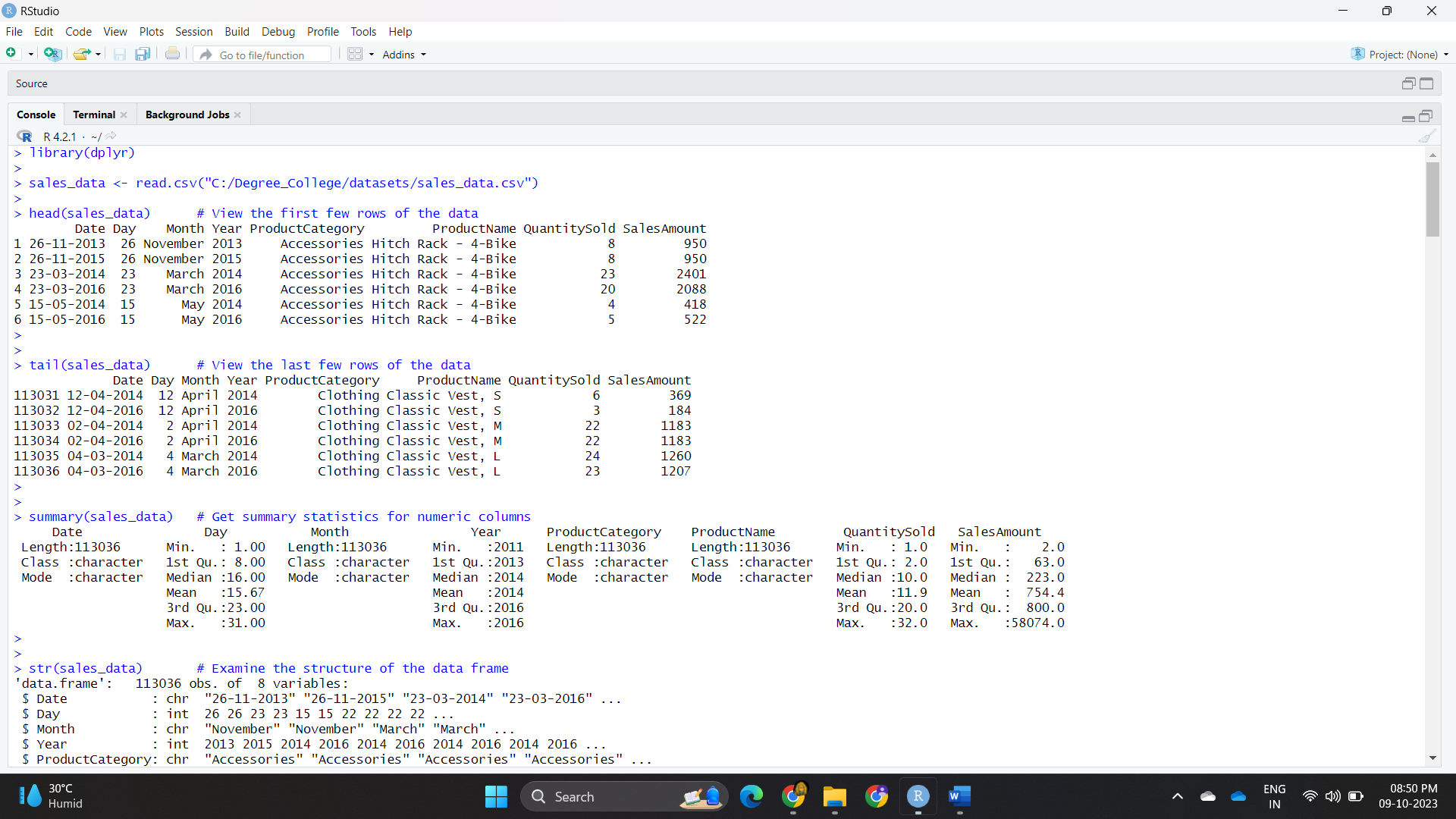
Q1\_Sales = quantile(SalesAmount, 0.25),

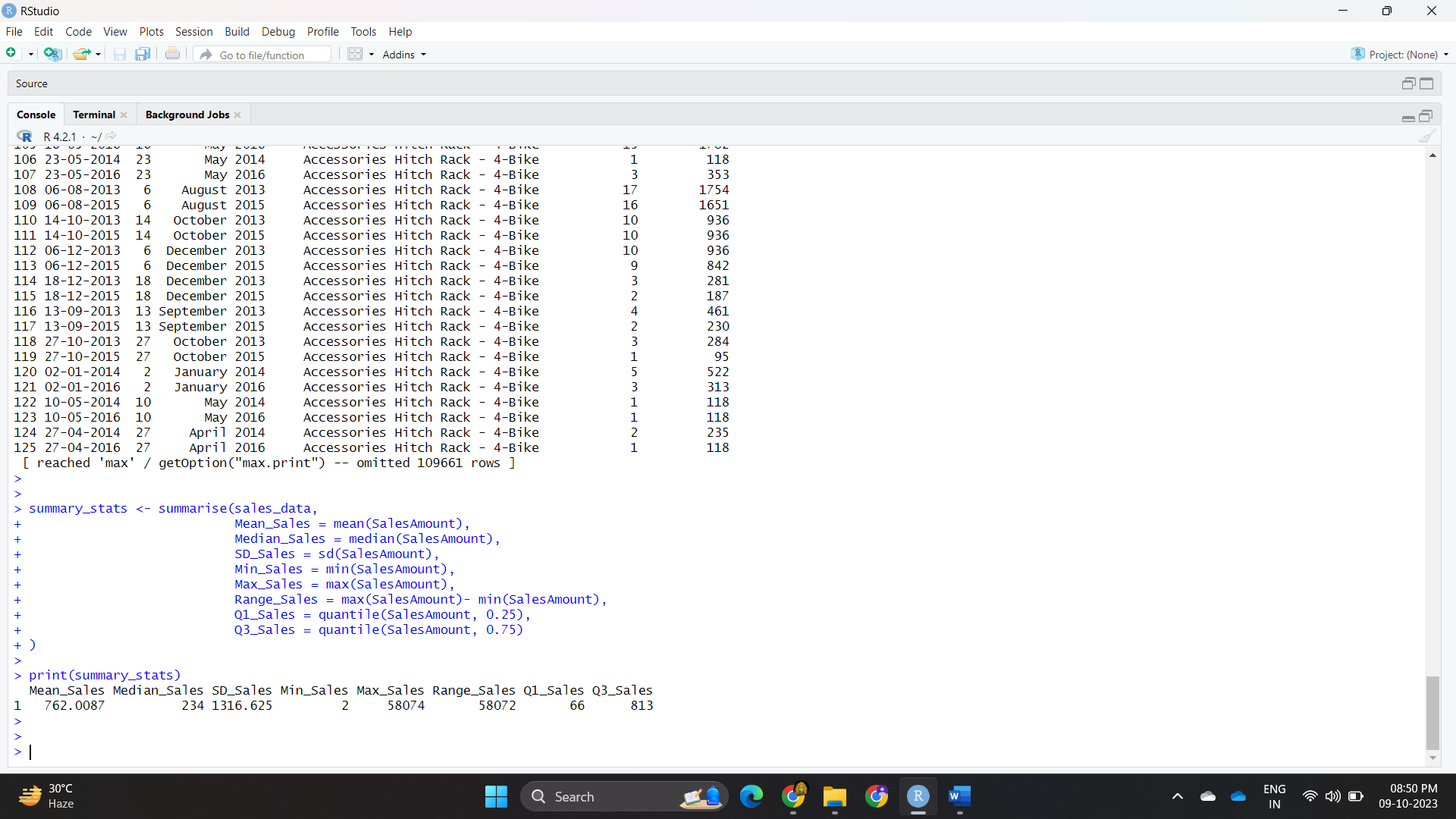
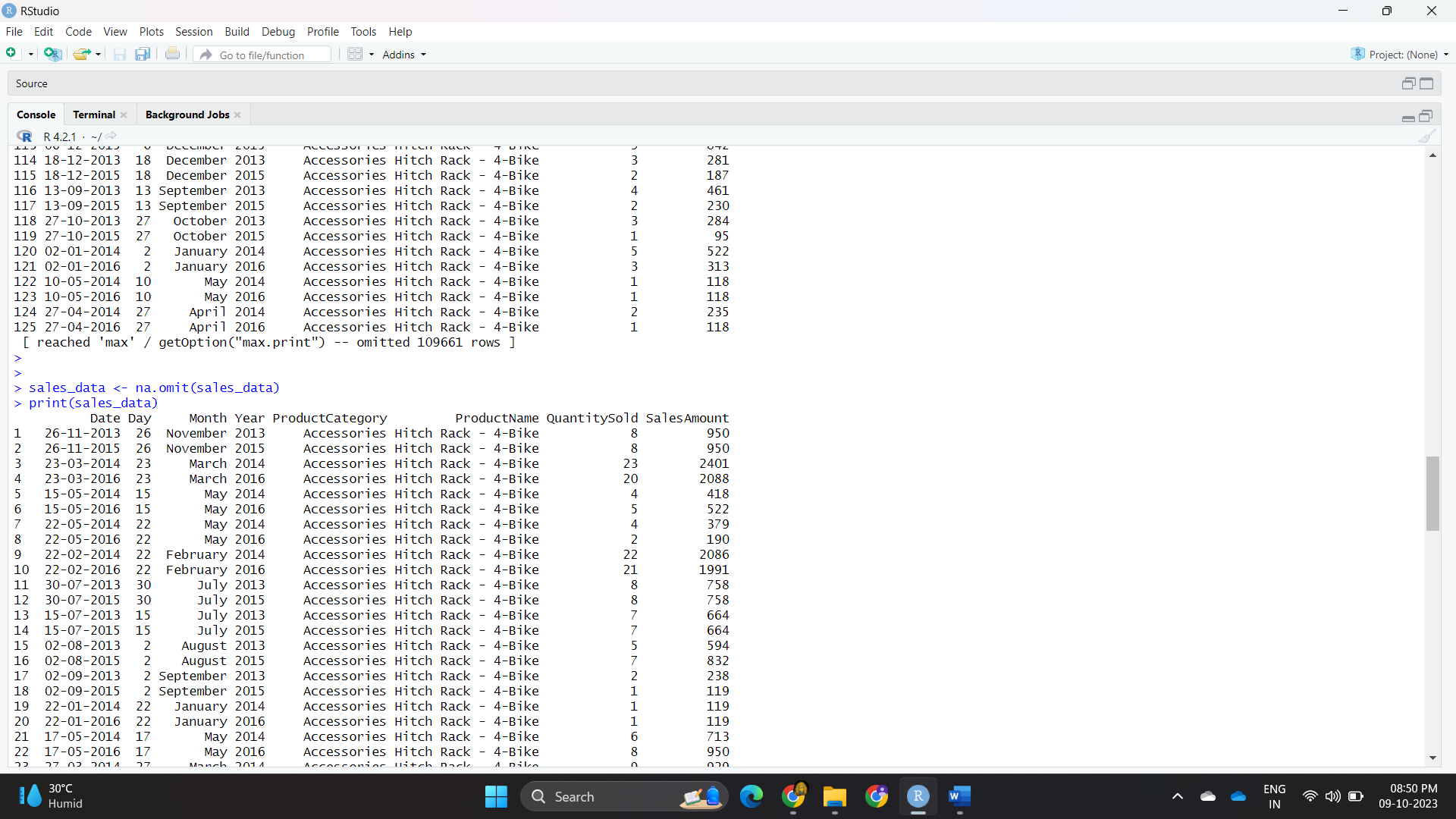
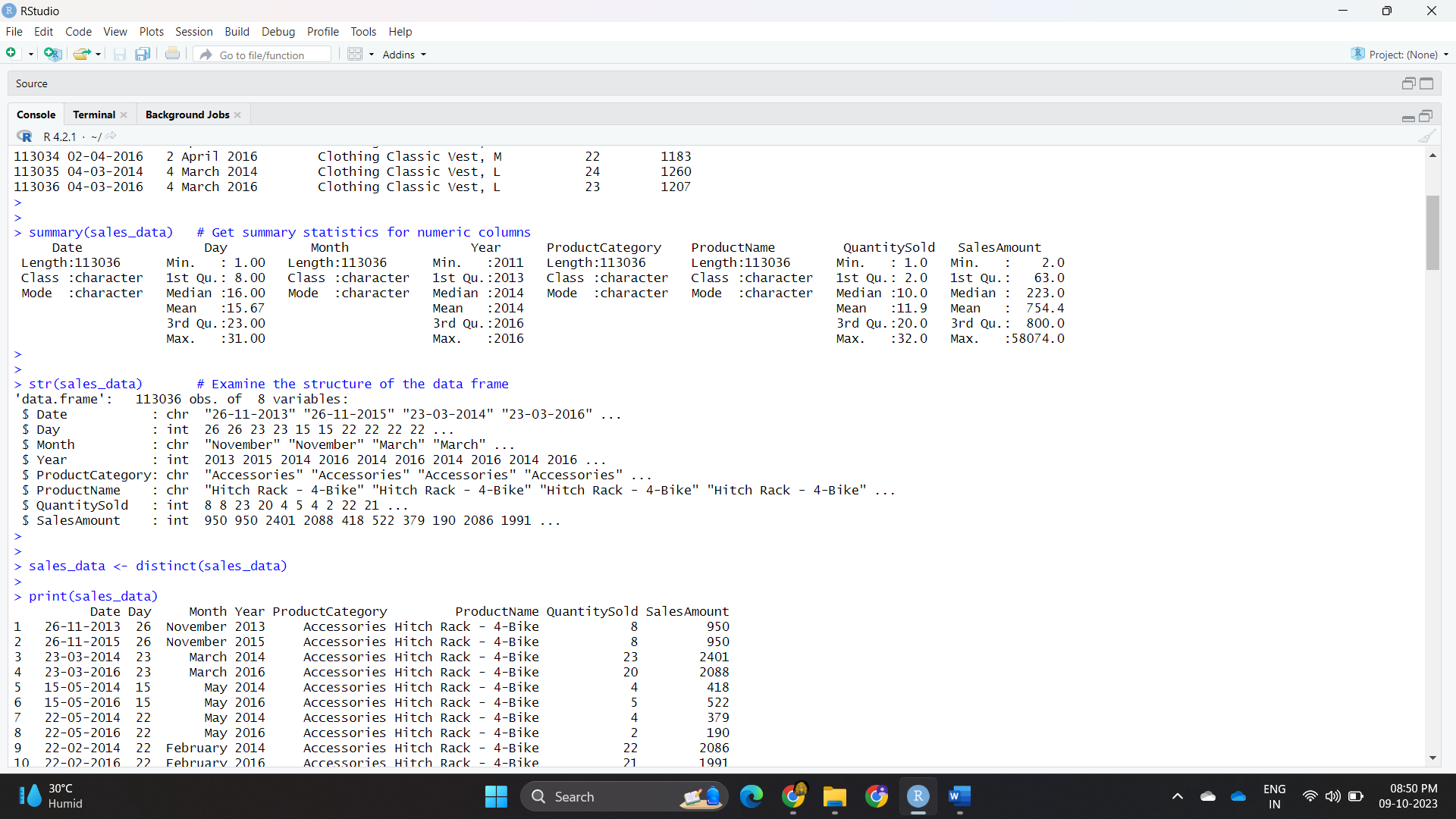
Q3\_Sales = quantile(SalesAmount, 0.75)

)

print(summary\_stats)

**Output :**





**Experiment No : 05**

**Code :**

# Load the required libraries

library(ggplot2)

library(dplyr)

# Import the dataset

sales\_data <- read.csv("C:/Degree\_College/datasets/sales\_data.csv")

# Basic Bar Chart: Visualizing sales by category

bar\_chart <- ggplot(sales\_data, aes(x = ProductCategory, y = SalesAmount)) +

geom\_bar(stat = "summary", fun = "sum", fill = "blue") +

labs(title = "Sales by Category", x = "Category", y = "Total Sales")

histogram <- ggplot(sales\_data, aes(x = SalesAmount)) +

geom\_histogram(fill = "green", bins = 20) +

labs(title = "Distribution of Sales Amounts", x = "Sales Amount", y = "Frequency")

scatter\_plot <- ggplot(sales\_data, aes(x = QuantitySold, y = SalesAmount)) +

geom\_point(color = "red") +

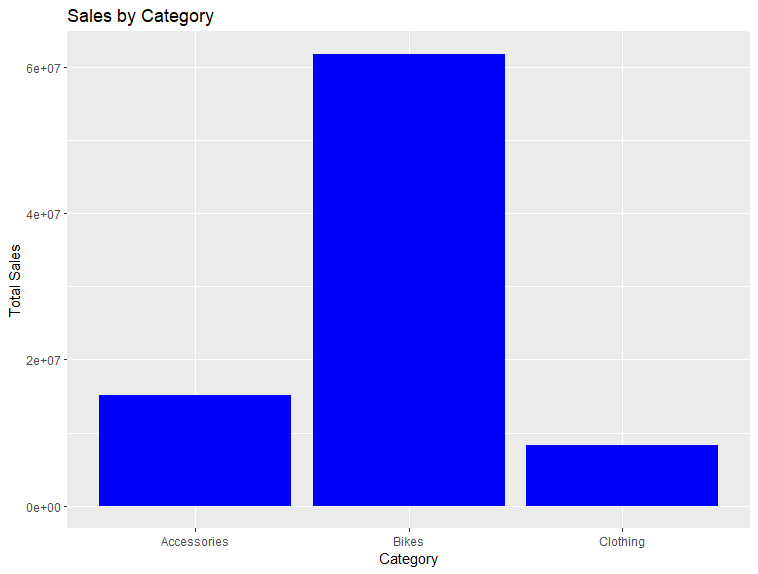
labs(title = "Relationship between Sales and Quantities Sold", x = "Quantity Sold", y = "Sales Amount")

print(bar\_chart)

print(histogram)

print(scatter\_plot)

**Output :**



**Experiment No : 06 A**

**Aim : To perform time series analysis on the "sales\_data.csv" dataset,** create a time series object, and apply smoothing techniques like moving averages or exponential smoothing using the smooth() function from the tsibble package to identify underlying patterns. Use the decompose() function from the "stats" package to decompose the time series into its components (trend, seasonal, and remainder).

**Code :**

# Install and load the necessary packages

library(tsibble)

library(fable)

sales\_data <- read.csv("C:/Degree\_College/datasets/sales\_data.csv")

# Assuming your dataset is named "sales\_data"

duplicates <- duplicated(sales\_data) # Find duplicate rows

sum(duplicates) # Check the total number of duplicates

# Remove duplicate rows if needed

sales\_data <- sales\_data[!duplicates, ]

# Assuming your "Date" column is in Date format

sales\_data$Date <- as.Date(sales\_data$Date)

sales\_data <- distinct(sales\_data, Date, .keep\_all = TRUE)

# Create the tsibble with the "Date" column as the index

sales\_tsibble <- as\_tsibble(sales\_data, index = Date)

summary(sales\_tsibble)

# Load the necessary packages

library(ggplot2)

# Create a time series plot with ggplot2

ggplot(data = sales\_tsibble, aes(x = Date, y = SalesAmount)) +

geom\_line(color = "black") +

labs(title = "Sales Data")

library(dplyr)

library(lubridate)

# Filter data from 2010 onwards

sales\_tsibble\_filtered <- sales\_tsibble %>%

filter(year(Date) >= 07-03-2011)

library(slider)

#For Moving Averages:

smoothed\_sales\_ma <- sales\_tsibble\_filtered %>%

mutate(Smoothed\_Sales = slide\_dbl(SalesAmount, .before = 3, .after = 0, .f = mean))

ggplot(data = smoothed\_sales\_ma, aes(x = Date, y = Smoothed\_Sales)) +

geom\_line(color = "blue") +

labs(title = "Moving Average Smoothing", x = "Date", y = "Smoothed Sales")

#For Exponential Smoothing (ETS):

smoothed\_sales\_ets <- forecast::ets(filtered\_sales$SalesAmount)

# Create a ggplot visualization for the smoothed sales

library(ggplot2)

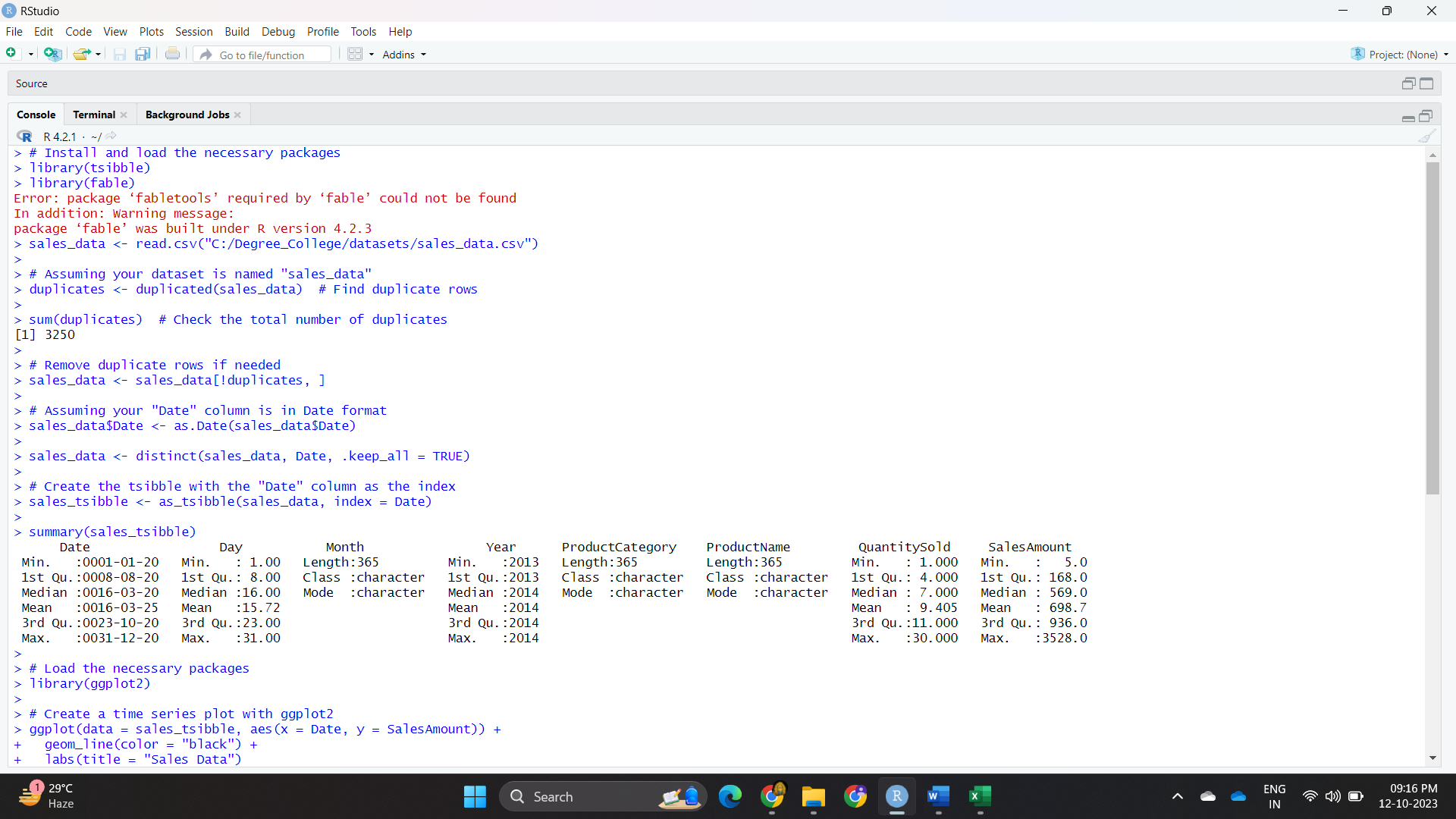
ggplot(data = data.frame(Date = filtered\_sales$Date, Smoothed\_Sales = smoothed\_sales\_ets$fitted),

aes(x = Date, y = Smoothed\_Sales)) +

geom\_line(color = "red") +

labs(title = "Exponential Smoothing (ETS) - Smoothed Sales", x = "Date", y = "Smoothed Sales")

**Output :**





**Experiment No : 06 B**

**Code :**

# Load the required libraries

library(survival)

library(survminer)

# Load your sales data from the "sales\_data.csv" file

sales\_data <- read.csv("C:/Degree\_College/datasets/sales\_data.csv")

# Fit a Kaplan-Meier survival model

surv\_fit <- survfit(Surv(SalesAmount, Event) ~ 1, data = sales\_data)

# Plot the Kaplan-Meier survival curve

ggsurvplot(

surv\_fit,

conf.int = TRUE, # Add confidence intervals

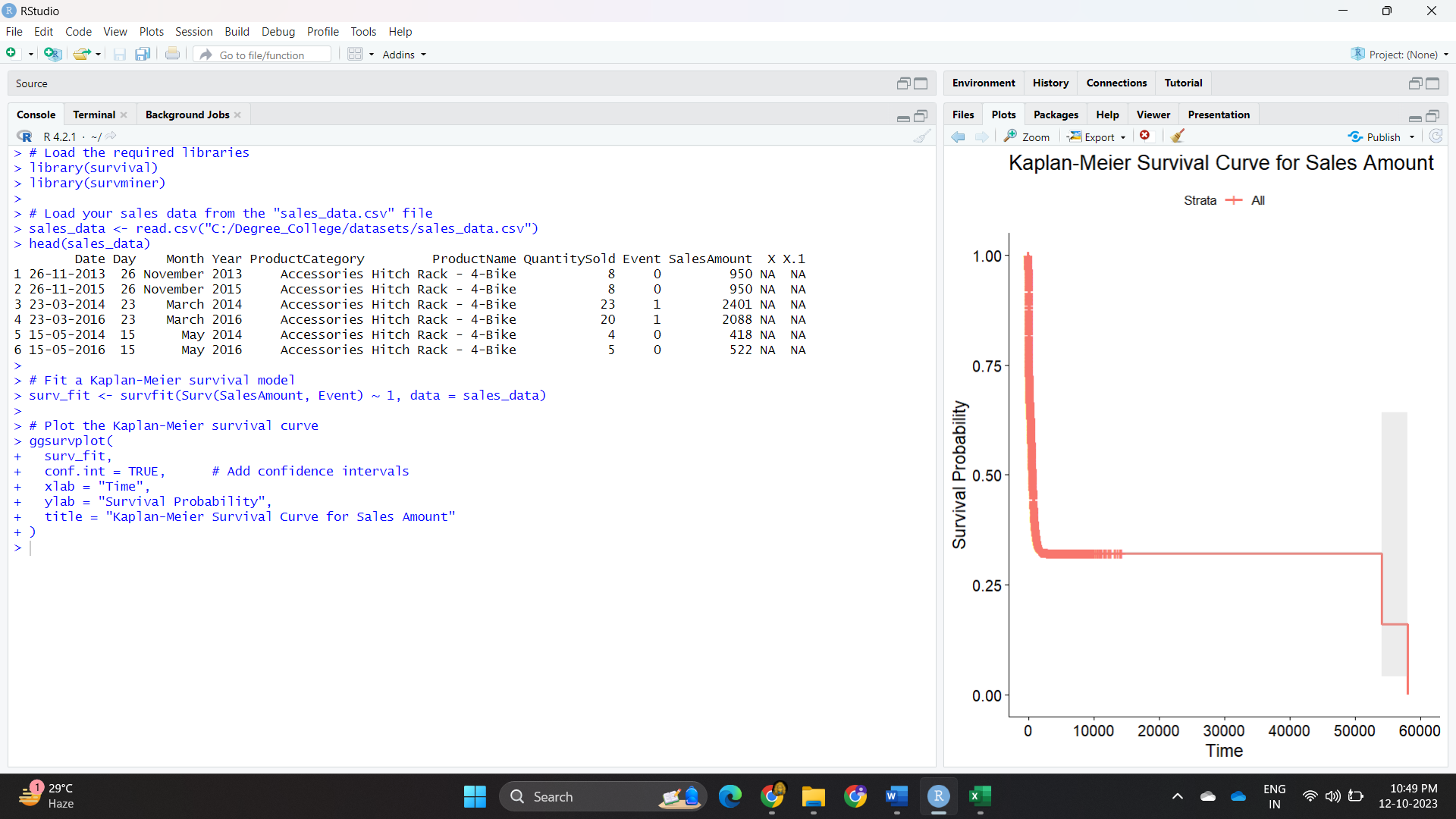
xlab = "Time",

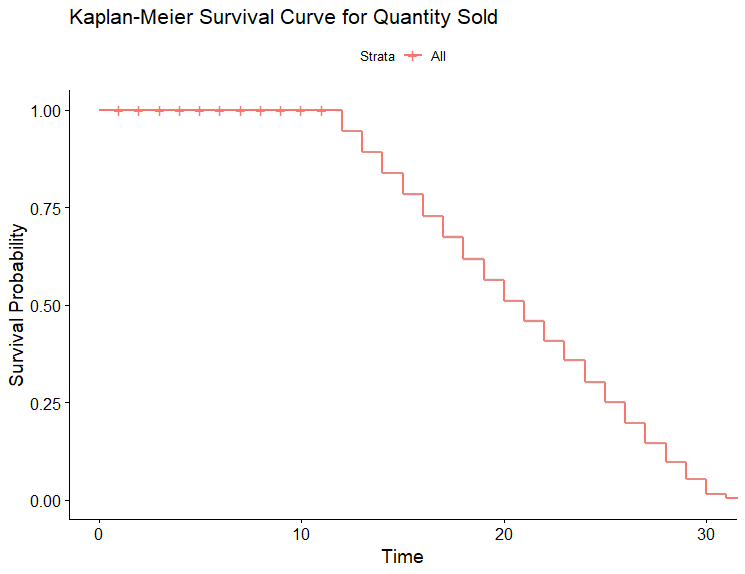
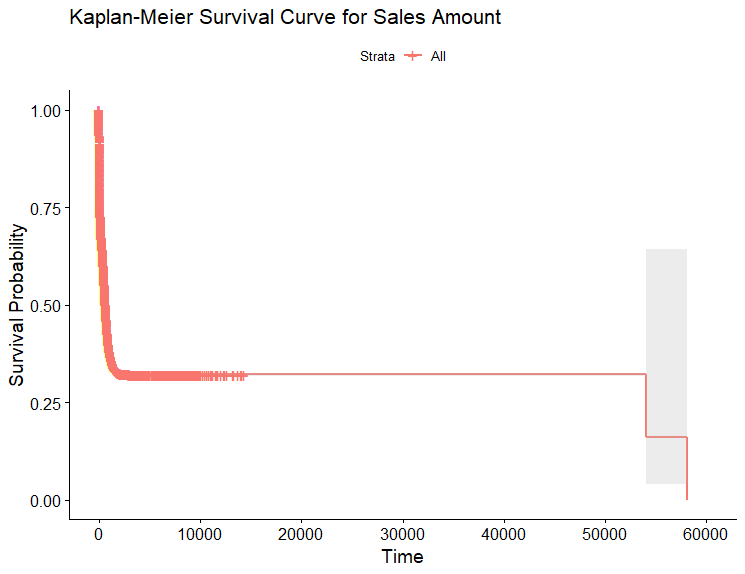
ylab = "Survival Probability",

title = "Kaplan-Meier Survival Curve for Sales Amount"

)

**Output :**





**Experiment No : 07**

**Code :**

# Load necessary libraries

library(dplyr)

library(caret)

library(ggplot2)

# Read the dataset

sales\_data <- read.csv("C:/Degree\_College/datasets/sales\_data.csv")

head(sales\_data)

# Identify and remove constant/zero variance columns

missing\_threshold <- 0.5 # Adjust this threshold as needed

sales\_data <- sales\_data %>%

select\_if(function(x) mean(is.na(x)) <= missing\_threshold)

# Separate the target variable (if present)

if ("SalesAmount" %in% colnames(sales\_data)) {

SalesAmount <- sales\_data$SalesAmount

sales\_data <- sales\_data[, -which(names(sales\_data) == "SalesAmount")]

}

# Filter only the numeric columns

data\_filtered <- sales\_data %>%

select\_if(is.numeric)

# Scale or normalize the numeric features

data\_scaled <- scale(data\_filtered)

# Apply PCA to the scaled data

num\_components <- 2 # Change this to the desired number of components

pca\_result <- prcomp(data\_scaled, center = TRUE, scale. = TRUE)

# Extract the reduced data

reduced\_data <- as.data.frame(pca\_result$x[, 1:num\_components])

# Print the PCA results, including the proportion of variance explained by each component

summary(pca\_result)

# Plot the variance explained by each component

plot(cumsum(pca\_result$sdev^2) / sum(pca\_result$sdev^2), xlab = "Number of Components", ylab = "Proportion of Variance Explained")

# Optionally, choose the number of components to retain based on the explained variance

# For example, if you want to retain 95% of the variance:

desired\_variance\_explained <- 0.95

# Assuming you've already computed the PCA

num\_components\_to\_retain <- which(cumsum(pca\_result$sdev^2) / sum(pca\_result$sdev^2) >= desired\_variance\_explained)[1]

# Extract the desired number of components

reduced\_data <- as.data.frame(pca\_result$x[, 1:num\_components\_to\_retain])

# Visualize the reduced data (assuming 2 components)

ggplot(data = reduced\_data, aes(x = PC1, y = PC2, color = "blue")) +

geom\_point() +

labs(x = "Principal Component 1", y = "Principal Component 2") +

ggtitle("Reduced Data Visualization")

# Visualize the reduced data

ggplot(data = reduced\_data, aes(x = PC1, y = PC2)) +

geom\_point(aes(color = SalesAmount)) +

labs(x = "Principal Component 1", y = "Principal Component 2") +

ggtitle("Reduced Data Visualization")

# Step 2: Apply K-means clustering using the stats package

# Identify the optimal number of clusters (K)

sales\_data <- sales\_data[complete.cases(sales\_data), ]

# Assuming "target" is the non-numeric column to remove

# Remove rows with missing values

sales\_data <- sales\_data[complete.cases(sales\_data), ]

# Remove non-numeric columns with zero variance

sales\_data <- sales\_data[, apply(sales\_data, 2, var) > 0]

# Check the data types

if (!all(sapply(sales\_data, is.numeric))) {

cat("Not all columns are numeric. Please check your data.")

} else {

cat("All columns are numeric. Proceed with clustering.")

}

# 1. Elbow Method

wss <- numeric(10) # Store within-cluster sum of squares (inertia)

for (i in 1:10) {

kmeans\_model <- kmeans(sales\_data, centers = i) # Ensure 'sales\_data' is the correct dataset

wss[i] <- kmeans\_model$tot.withinss

}

# Check the results of the elbow method

plot(1:10, wss, type = "b", xlab = "Number of Clusters (K)", ylab = "Within-Cluster Sum of Squares")

# 2. Silhouette Analysis

sil\_scores <- numeric(10) # Store silhouette scores

for (i in 2:10) {

kmeans\_model <- kmeans(sales\_data, centers = i) # Use 'sales\_data' instead of 'data'

sil\_scores[i] <- silhouette(kmeans\_model$cluster, dist(sales\_data))$avg.width

}

# Plot silhouette scores

plot(2:10, sil\_scores[-1], type = "b", xlab = "Number of Clusters (K)", ylab = "Silhouette Score")

# Load the necessary libraries

library(stats)

library(fpc) # Load the fpc package for Davies-Bouldin index

# Read the dataset

data <- read.csv("C:/Degree\_College/datasets/sales\_data.csv")

data <- na.omit(data)

data <- as.data.frame(sapply(data, as.numeric))

k\_values <- 2:10 # You can adjust this range

db\_indices <- numeric(length(k\_values))

# Calculate Davies-Bouldin index for each K value

for (i in 1:length(k\_values)) {

k <- k\_values[i]

kmeans\_model <- kmeans(data, centers = k)

db\_indices[i] <- cluster.stats(data, kmeans\_model$cluster)$db

}

# Plot the Davies-Bouldin indices for different K values

plot(k\_values, db\_indices, type = "b", xlab = "Number of Clusters (K)", ylab = "Davies-Bouldin Index")

# Find the optimal K with the lowest Davies-Bouldin index

optimal\_k <- k\_values[which.min(db\_indices)]

# Print the Davies-Bouldin indices

cat("Davies-Bouldin Indices for each K:\n")

print(db\_indices)

cat("Optimal K (lowest Davies-Bouldin index):", optimal\_k, "\n")

# Specify the optimal number of clusters (you may need to determine this based on your analysis)

optimal\_k <- 2

set.seed(123)

km.res <- kmeans(data\_filtered, 2, nstart = 25)

print(km.res)

# Load the necessary libraries

library(ggplot2)

# Assuming you've already applied K-means clustering and have the results in `km.res`

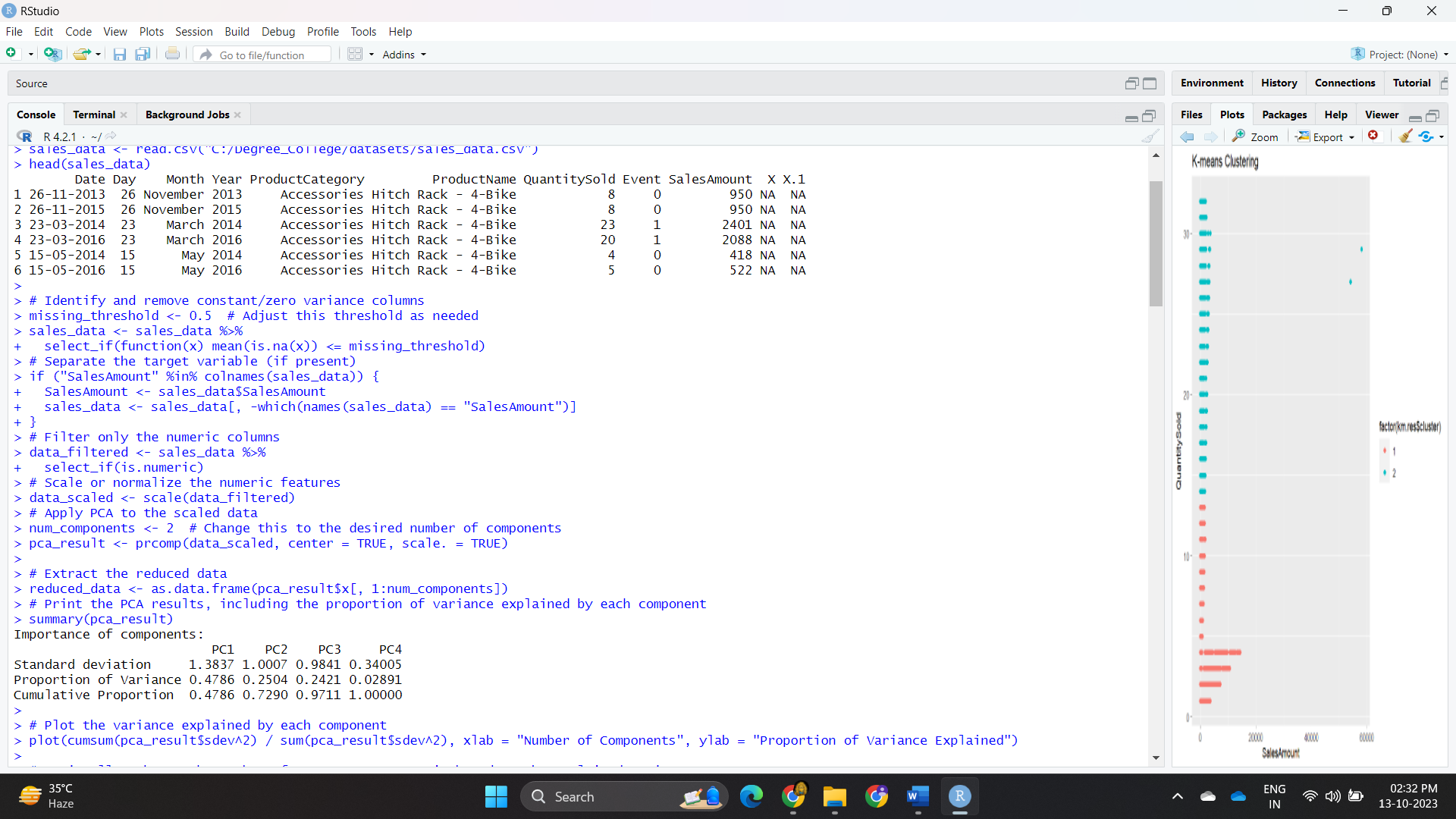
# Plot K-means clusters using ggplot2

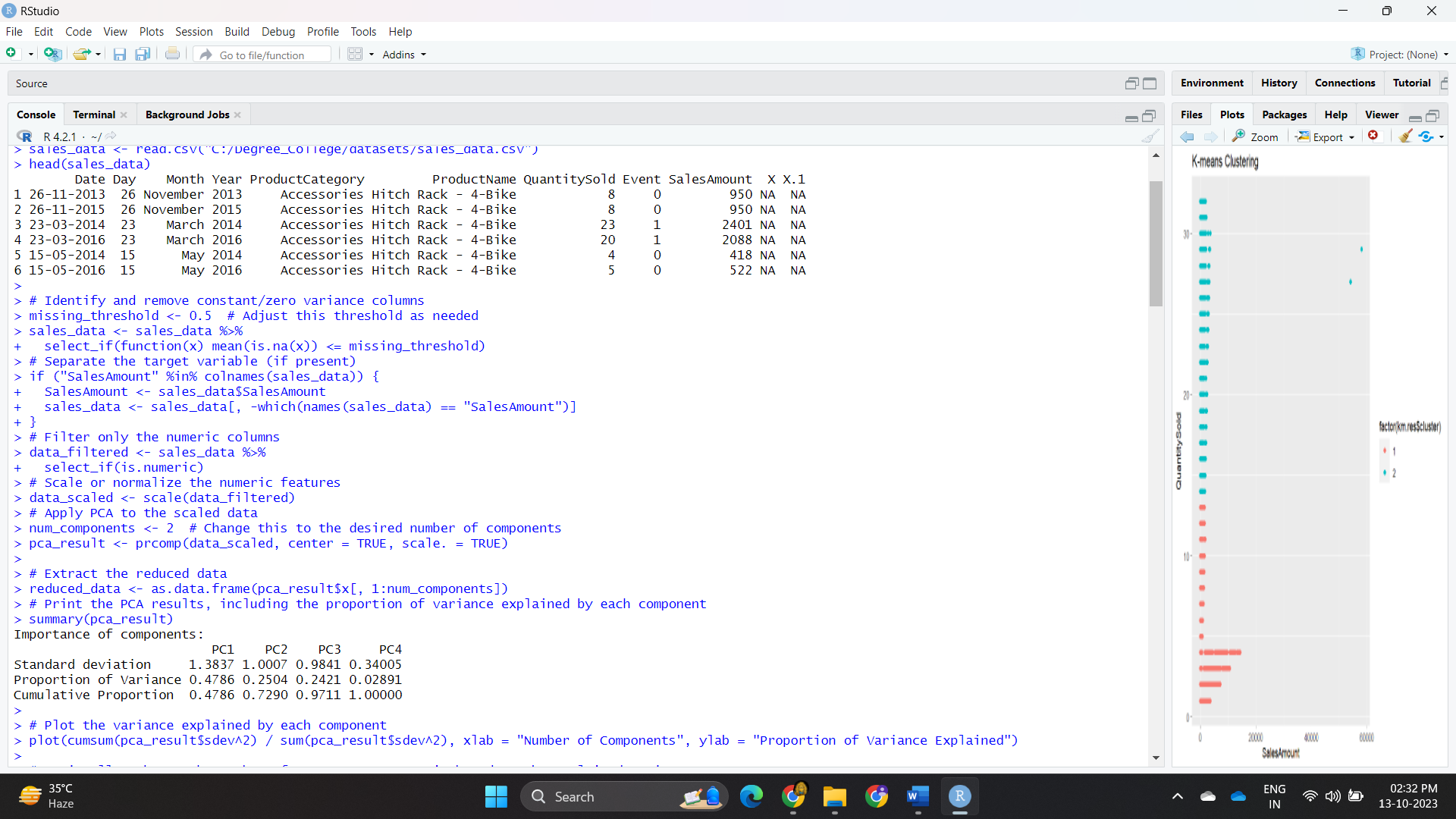
ggplot(data = data\_filtered, aes(x = SalesAmount, y = QuantitySold, color = factor(km.res$cluster))) +

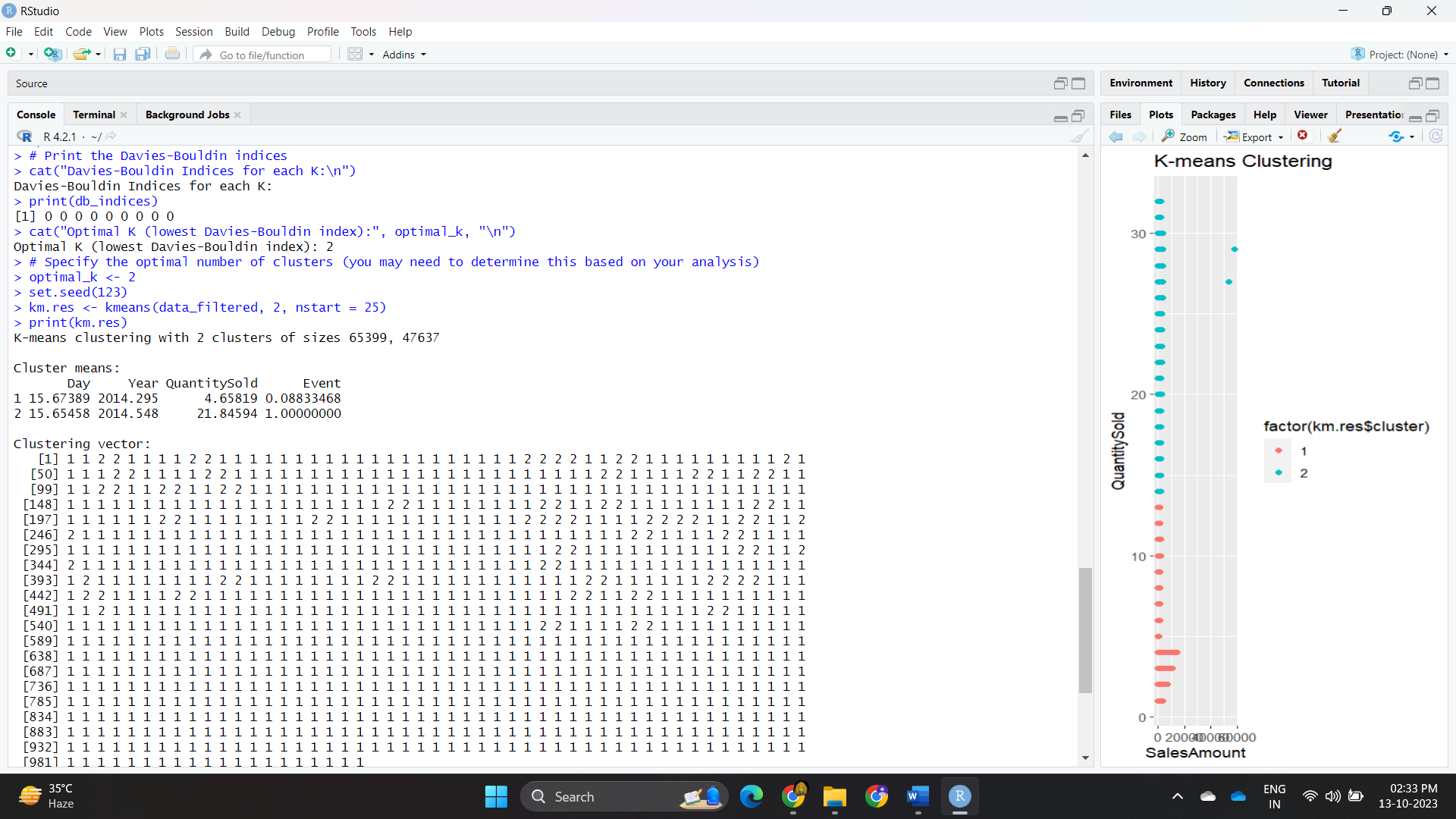
geom\_point() +

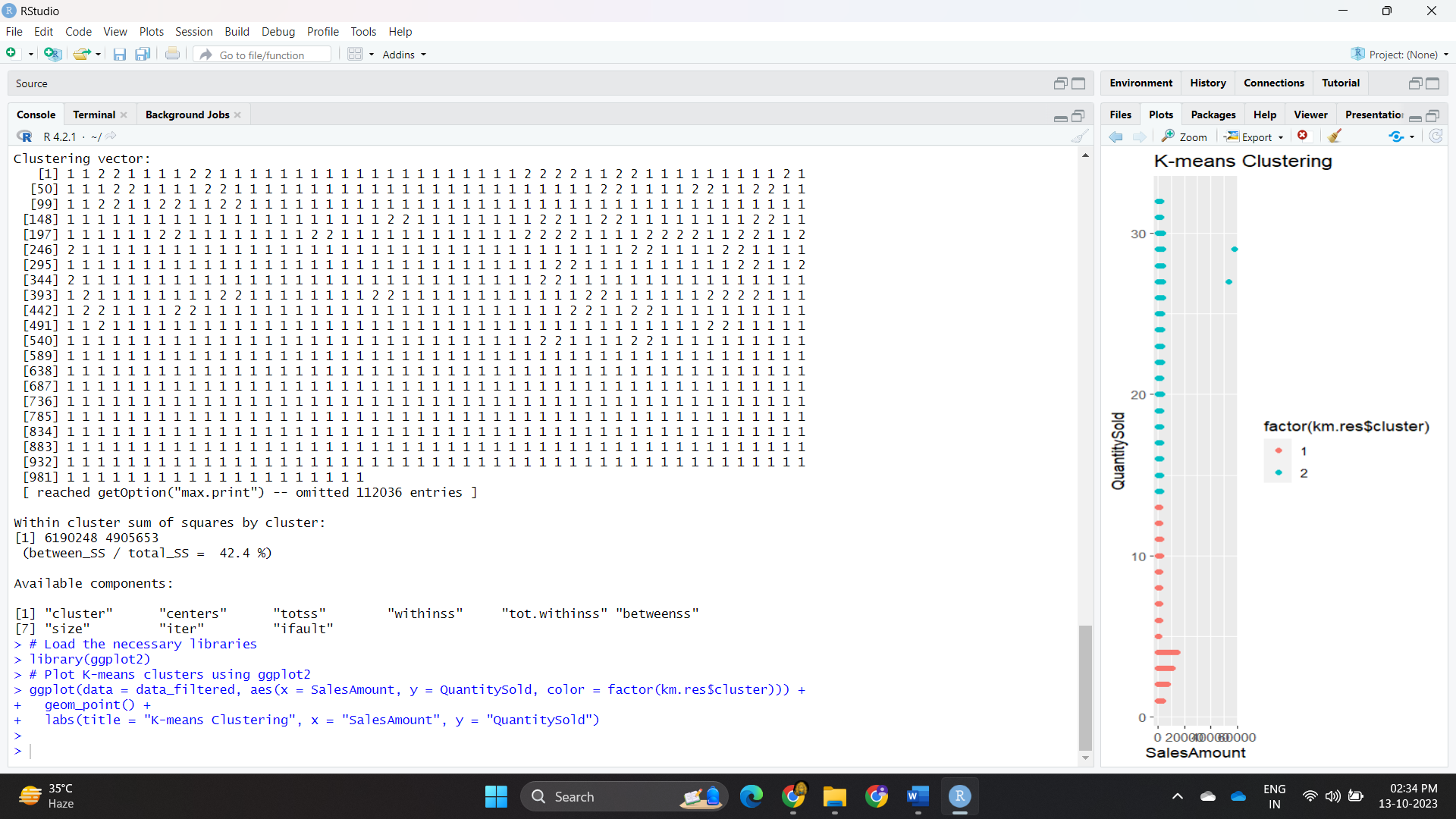
labs(title = "K-means Clustering", x = "SalesAmount", y = "QuantitySold")

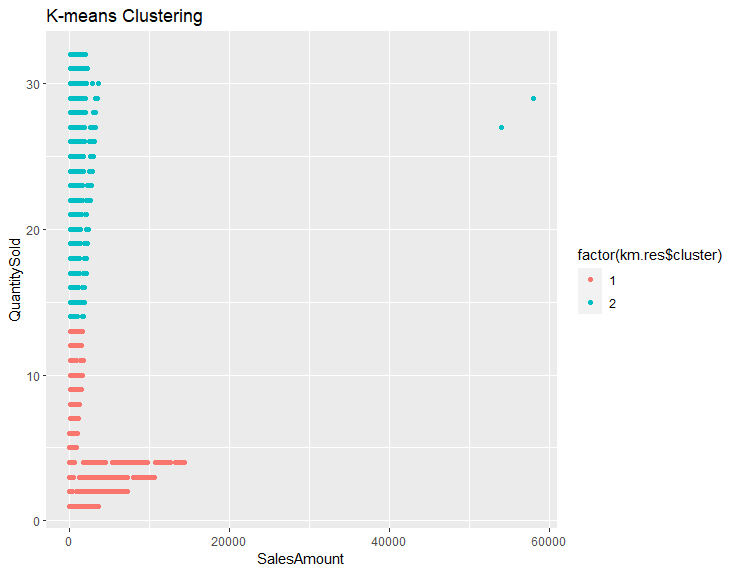
**Output :**











**Experiment No : 08**

**Code :**

library(caret)

library(rpart)

library(e1071)

sales\_data <- read.csv("C:/Degree\_College/datasets/sales\_data.csv")

head(sales\_data)

sales\_data <- data.frame(sales\_data, model.matrix(~ ProductName - 1, data = sales\_data))

# Split the data into training and testing sets

set.seed(123) # for reproducibility

unique\_categories <- unique(sales\_data$ProductName)

print(unique\_categories)

# Before splitting, factorize the target variable

sales\_data$ProductName <- factor(sales\_data$ProductName)

trainIndex <- createDataPartition(sales\_data$ProductName, p = 0.7,list = FALSE, times = 1)

data\_train <- sales\_data[trainIndex, ]

data\_test <- sales\_data[-trainIndex, ]

# Train the decision tree model

library(rpart.plot)

tree\_model <- rpart(Year ~ Day + Month + ., data = data\_train)

# You can visualize the tree with plot(tree\_model)

plot(tree\_model)

# If you want to tune hyperparameters, you can use cross-validation

ctrl <- rpart.control(minsplit = 5, cp = 0.001)

cv\_tree\_model <- rpart(Year ~ Day + Month + ., data = data\_train, control = ctrl)

plot(cv\_tree\_model)

# Check the levels of the target variable in the training and test datasets

levels(data\_train$ProductName)

levels(data\_test$ProductName)

# Ensure that the levels are the same

data\_test$ProductName <- factor(data\_test$ProductName, levels = levels(data\_train$ProductName))

# Make predictions on the test set

predictions <- as.character(predict(tree\_model, data\_test))

# Create a confusion matrix

confusion\_matrix <- table(Actual = data\_test$ProductName, Predicted = predictions)

# Print the confusion matrix

print(confusion\_matrix)

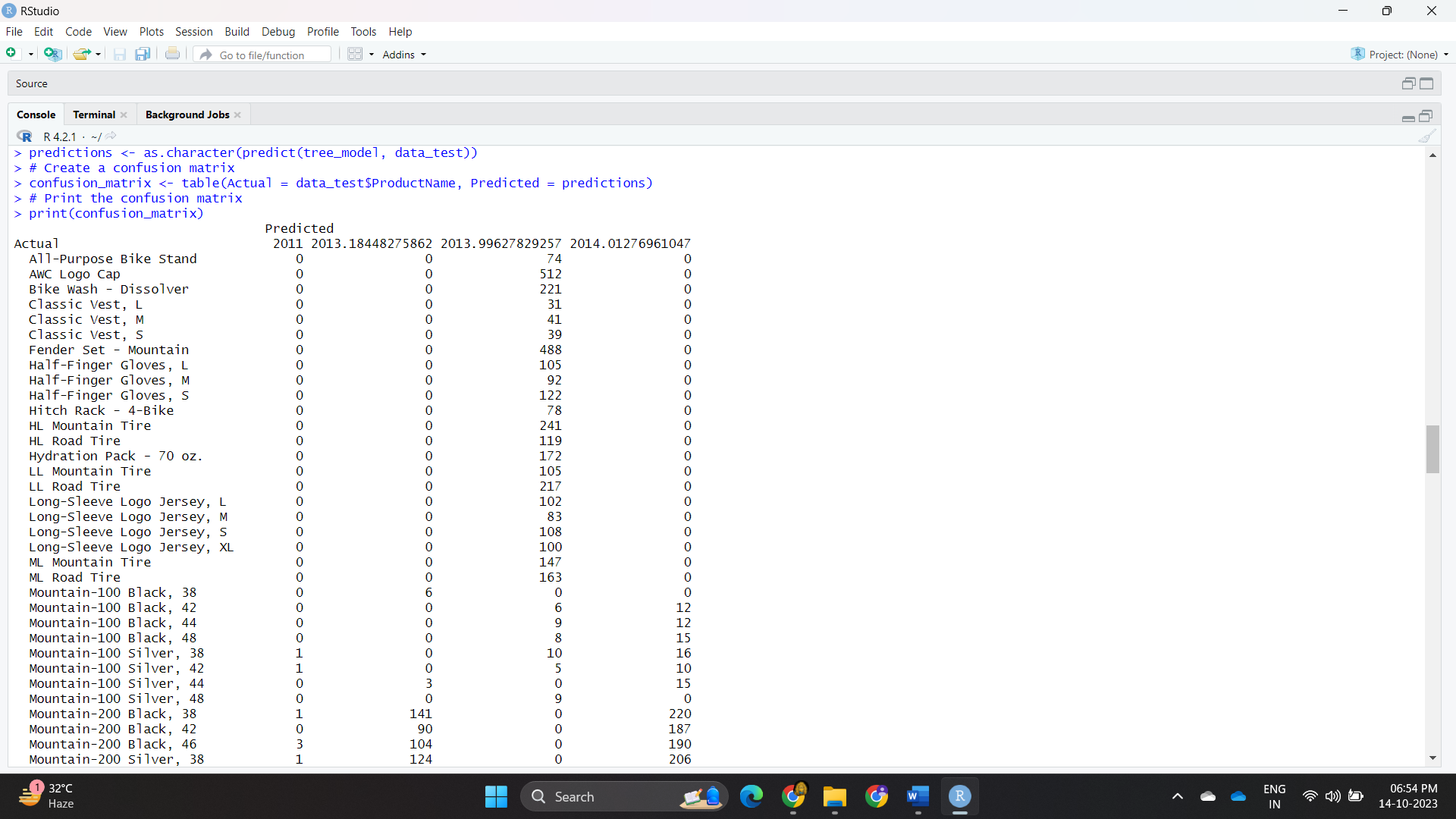
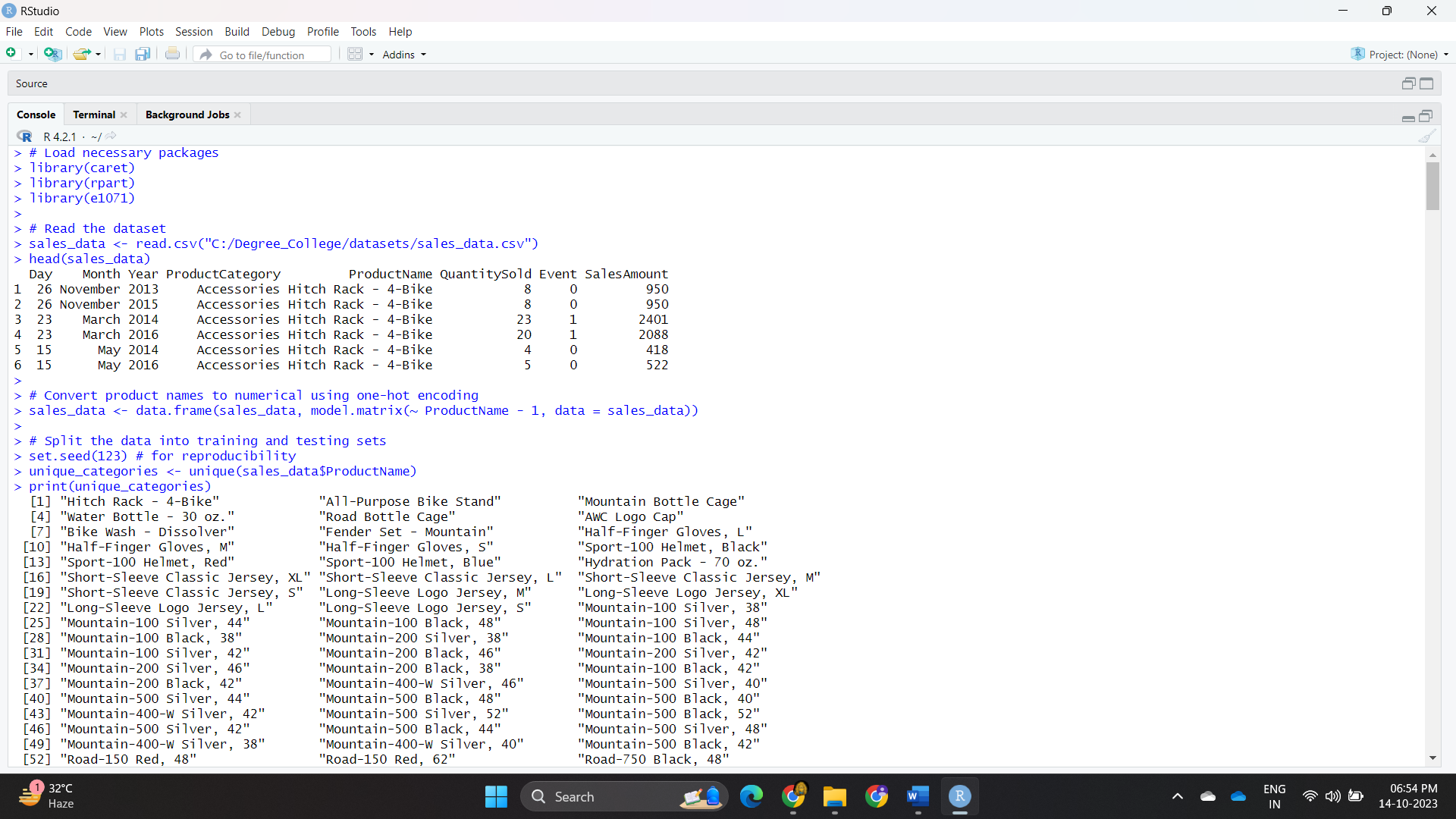
# Ensure that the levels of the target variable in test and predictions match

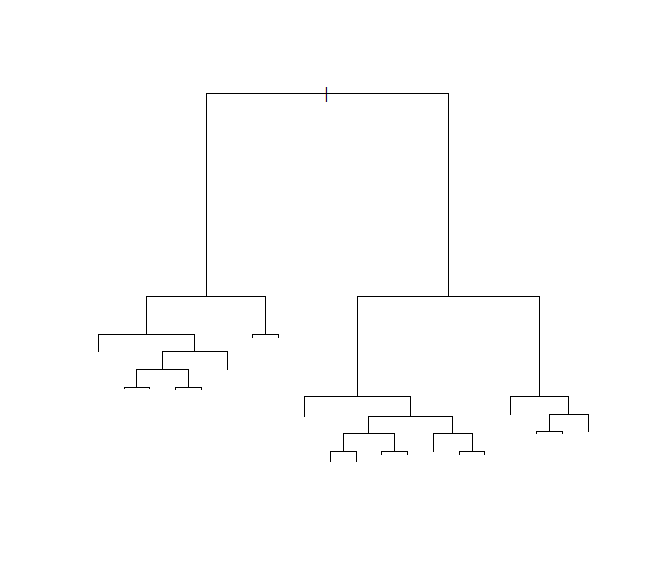
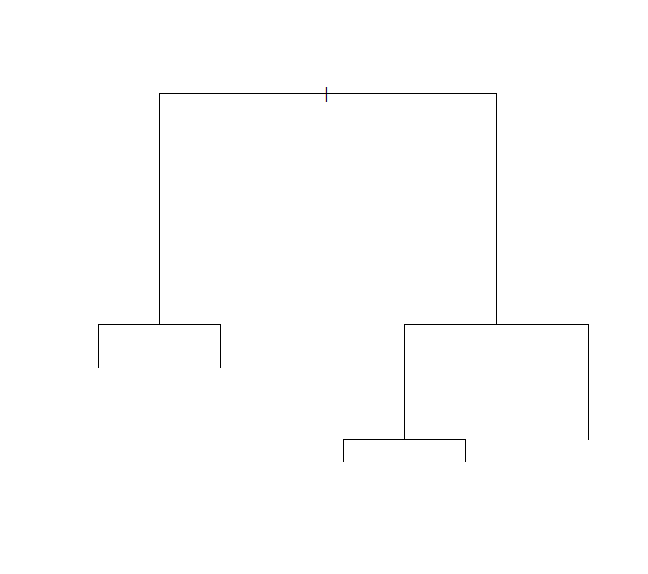
data\_test$ProductName <- factor(data\_test$ProductName, levels = levels(predictions))

# Plot the ROC curve

plot(roc\_values, print.auc = TRUE)

**Output :**







**Experiment No : 09**

**Aim : Case study-Impact of social networking websites on quality of recruitment**

**Code :**

# Load necessary libraries

library(ggplot2)

library(dplyr)

# Load your recruitment data

recruitment\_data <- read.csv("C:/Degree\_College/datasets/Placement\_data\_full\_class.csv")

# Explore the data

head(recruitment\_data) # View the first few rows of the data

summary(recruitment\_data) # Get summary statistics

# Analyze the impact of social networking websites on recruitment quality

recruitment\_data %>%

group\_by(status) %>%

summarize(

MeanQuality = mean(etest\_p),

MedianQuality = median(etest\_p),

SampleSize = n()

) %>%

mutate(status = ifelse(status == 1, "Yes", "No")) %>%

ggplot(aes(x = status, y = MeanQuality, label = SampleSize)) +

geom\_bar(stat = "identity", position = "dodge", fill = "skyblue") +

geom\_text(position = position\_dodge(width = 0.9), vjust = -0.5) +

labs(

title = "Impact of Social Networking on Recruitment Quality",

x = "Using specialisation for Recruitment",

y = "Mean Quality Score"

)

# Example: T-test comparing recruitment quality between social networking and non-social networking recruitment.

t\_test\_result <- t.test(etest\_p ~ status, data = recruitment\_data)

# Print the t-test results

print(t\_test\_result)

# 3. Data Visualization

# Example: A boxplot to compare recruitment quality by source

ggplot(recruitment\_data, aes(x=etest\_p, y=status)) +

geom\_boxplot() +

labs(title="Impact of Social Networking on Recruitment Quality", x="etest\_p", y="Recruitment Quality")

# Box plot to compare recruitment quality between social networking and non-social networking recruitment

ggplot(recruitment\_data, aes(x = factor(specialisation), y = status)) +

geom\_boxplot(fill = "skyblue") +

labs(

title = "Recruitment Quality by Social Networking Usage",

x = "Using Social Networking for Recruitment",

y = "status"

)

# Histograms of recruitment quality for social networking and non-social networking recruitment

ggplot(recruitment\_data, aes(x = etest\_p, fill = factor(specialisation))) +

geom\_histogram(binwidth = 1, position = "dodge") +

labs(

title = "Distribution of Recruitment Quality",

x = "Quality Score",

y = "Count"

)

# Scatter plot to explore the relationship between years of experience and recruitment quality

ggplot(recruitment\_data, aes(x = mba\_p, y = etest\_p, color = factor(specialisation))) +

geom\_point() +

labs(

title = "Recruitment Quality vs. Years of Experience",

x = "Years of Experience",

y = "Quality Score"

)

# Density plot to visualize the distribution of recruitment quality scores

ggplot(recruitment\_data, aes(x = etest\_p, fill = factor(specialisation))) +

geom\_density(alpha = 0.5) +

labs(

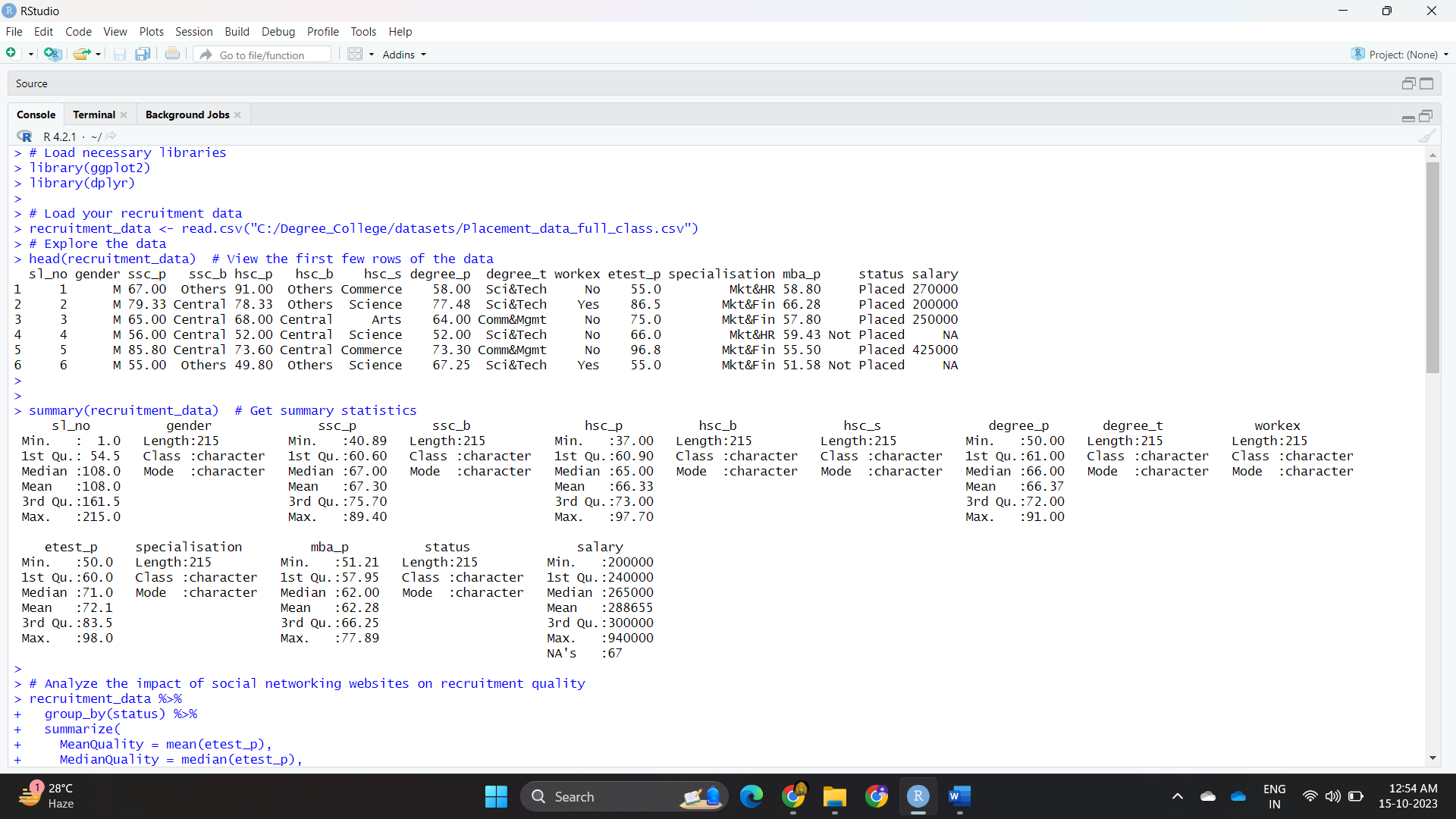
title = "Density of Recruitment Quality Scores",

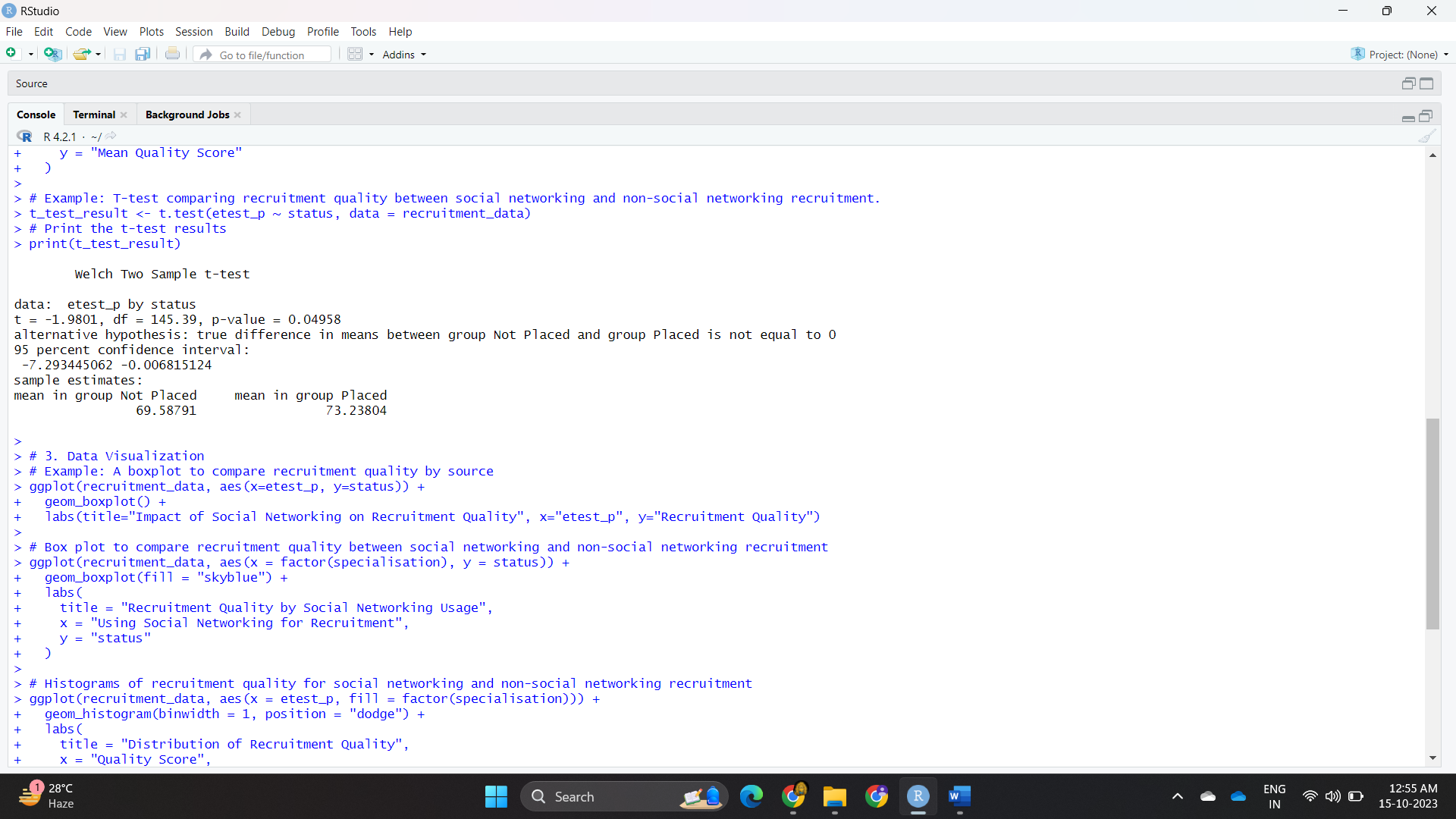
x = "Quality Score",

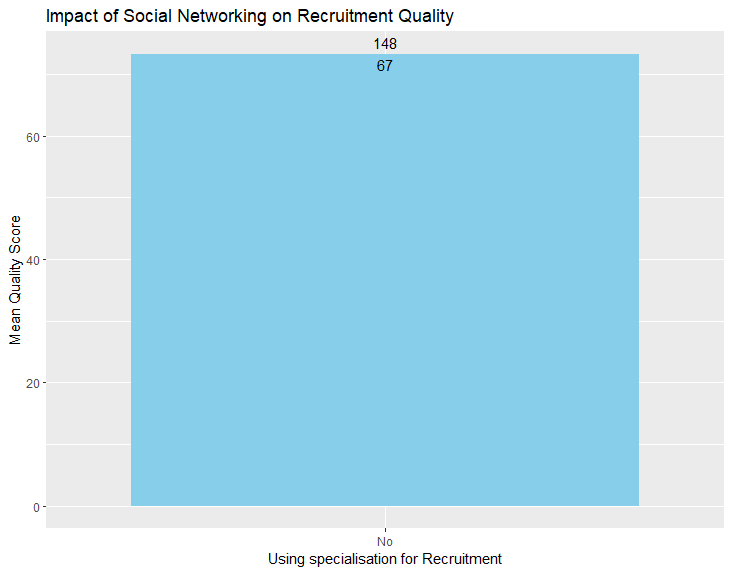
y = "Density"

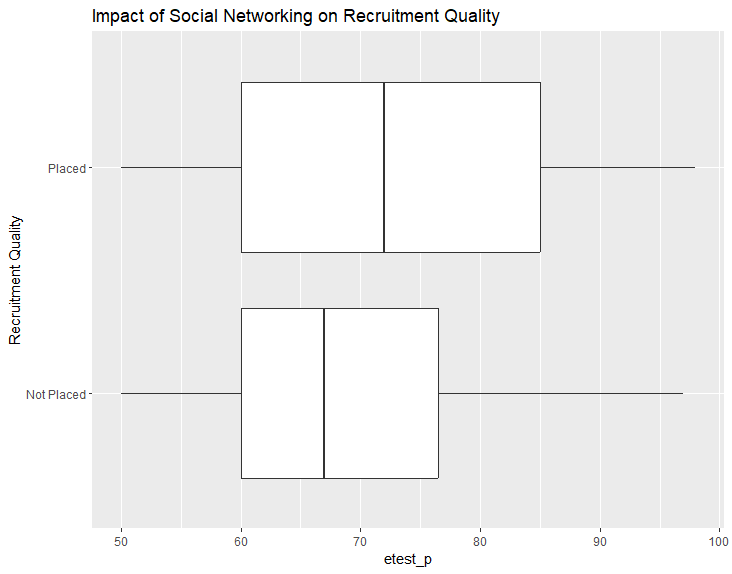
)

**Output :**









**Experiment No : 10**

**Code :**

library(tm)

library(e1071)

library(caret)

data <- read.csv("C:/Degree\_College/datasets/SPAM\_text\_message.csv")

corpus <- Corpus(VectorSource(data$Category))

dtm <- DocumentTermMatrix(corpus)

set.seed(123)

n <- nrow(data)

sample\_indices <- sample(1:n, 0.8 \* n) # 80% for training, 20% for testing

trainData <- dtm[sample\_indices, ]

testData <- dtm[-sample\_indices, ]

trainLabels <- as.factor(data$Category[sample\_indices])

testLabels <- as.factor(data$Category[-sample\_indices])

trainMatrix <- as.matrix(trainData)

model <- naiveBayes(trainMatrix, trainLabels)

predicted <- predict(model, newdata = as.matrix(testData))

confusion\_matrix <- table(predicted, testLabels)

accuracy <- sum(diag(confusion\_matrix)) / sum(confusion\_matrix)

precision <- confusion\_matrix[2, 2] / sum(confusion\_matrix[, 2])

recall <- confusion\_matrix[2, 2] / sum(confusion\_matrix[2, ])

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

# Print evaluation metrics

print(confusion\_matrix)

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

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

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

cat("F1 Score:", f1\_score, "\n")

# Create a data frame for plotting

metrics <- data.frame(Metric = c("Accuracy", "Precision", "Recall", "F1 Score"),

Value = c(accuracy, precision, recall, f1\_score))

# Create a bar plot

ggplot(metrics, aes(x = Metric, y = Value, fill = Metric)) +

geom\_bar(stat = "identity") +

labs(title = "Model Performance Metrics", y = "Value") +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))

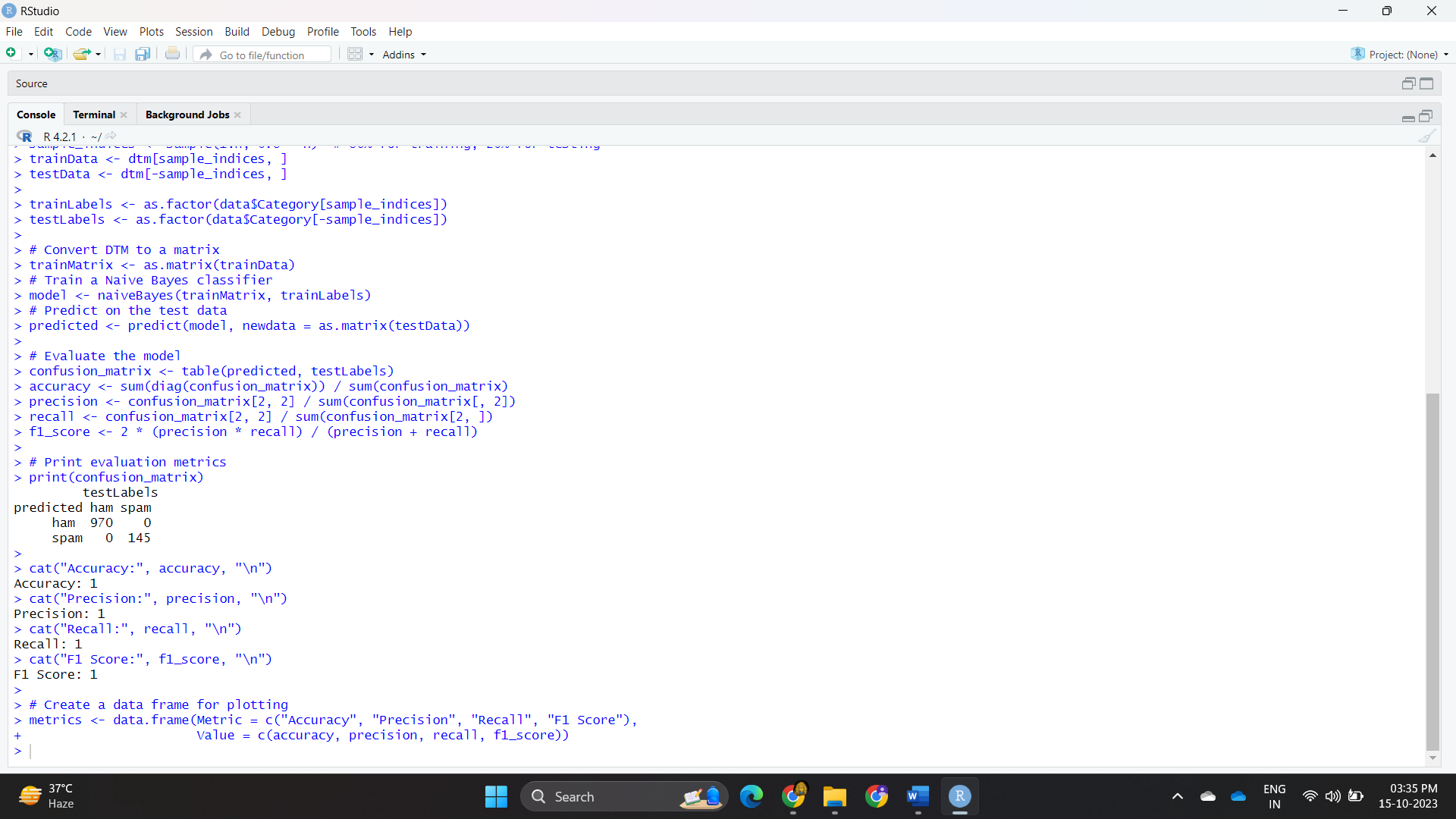
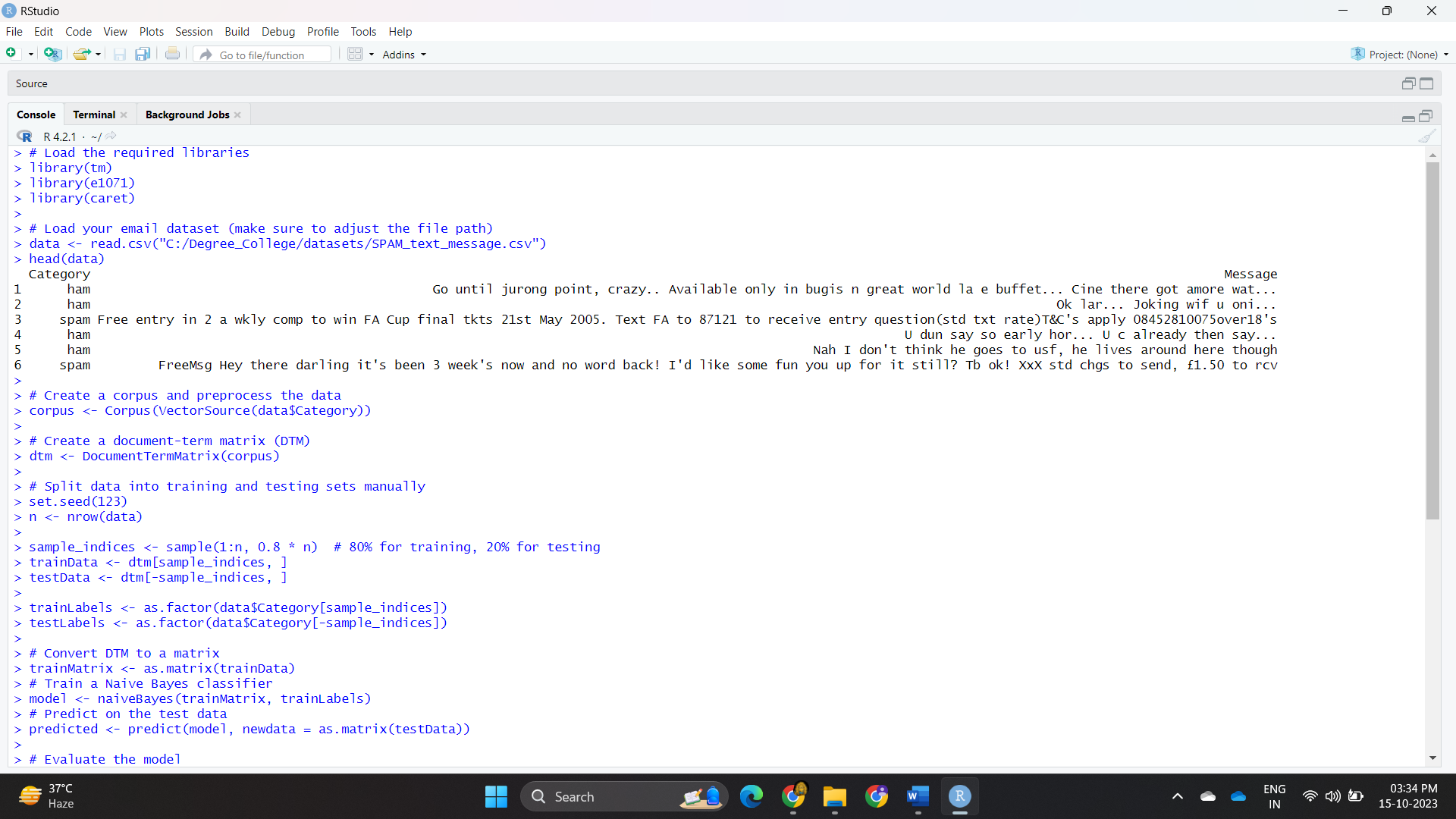
library(gplots)

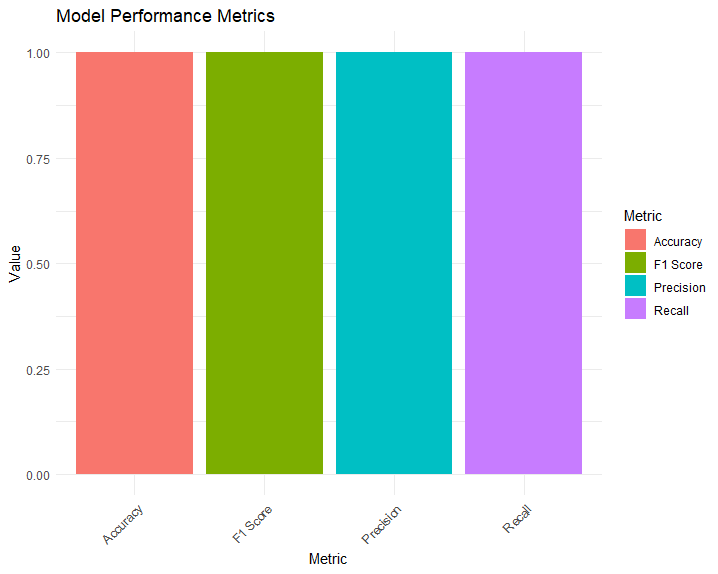
heatmap(confusion\_matrix, col = colorRampPalette(c("green", "yellow"))(5))

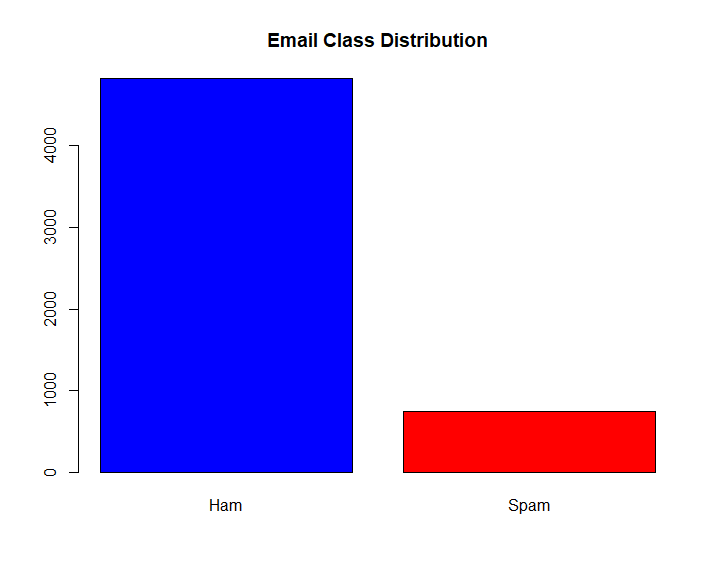
barplot(table(data$Category), col = c("blue", "red"),

names.arg = c("Ham", "Spam"), main = "Email Class Distribution")

**Output :**







**Experiment No : 11**

**Code :**

library(tm)

library(SentimentAnalysis)

library(SnowballC)

news\_data <- read.csv("C:/Degree\_College/datasets/amazon\_review.csv")

head(news\_data)

summary(news\_data)

corpus <- Corpus(VectorSource(news\_data))

length(sentiments)

length(historical\_sales)

length(historical\_popularity)

# Perform sentiment analysis

sentiment\_scores <- lapply(corpus, analyzeSentiment)

sentiments <- sapply(sentiment\_scores, function(x) ifelse(is.null(x$rating), "unknown", x$rating))

historical\_sales <- c(100, 50, 120, 40) # Replace with your data

historical\_popularity <- c(8, 4, 10, 3) # Replace with your data

newdata <- data.frame(rating = as.factor(sentiments), Sales = historical\_sales, Popularity = historical\_popularity)

data$SentimentValue <- ifelse(data$Sentiment == "positive", 1,

ifelse(data$Sentiment == "negative", -1, 0))

# Build a predictive model (linear regression)

model\_sales <- lm(Sales ~ SentimentValue, data = data)

model\_popularity <- lm(Popularity ~ SentimentValue, data = data)

new\_data <- data.frame(Sentiment = "positive") # Replace with your data

new\_data$SentimentValue <- ifelse(new\_data$Sentiment == "positive", 1,

ifelse(new\_data$Sentiment == "negative", -1, 0))

predicted\_sales <- predict(model\_sales, newdata = new\_data)

predicted\_popularity <- predict(model\_popularity, newdata = new\_data)

cat("Predicted Sales:", predicted\_sales, "\n")

cat("Predicted Popularity:", predicted\_popularity, "\n")

# Load required visualization packages

library(ggplot2)

library(plotly)

# Create a data frame for visualization

visualization\_data <- data.frame(

Metric = c("Sales", "Popularity"),

Predicted = c(predicted\_sales, predicted\_popularity),

Actual = c(historical\_sales, historical\_popularity) # Replace with actual data

)

print(visualization\_data)

# Create a bar plot

bar\_plot <- ggplot(visualization\_data, aes(x = Metric, y = Predicted, fill = Metric)) +

geom\_bar(stat = "identity", position = position\_dodge()) +

labs(title = "Predicted Sales and Popularity vs. Actual",

y = "Values",

fill = "Metric") +

theme\_minimal()

print(bar\_plot)

# Create an interactive plotly bar chart

plotly\_bar\_plot <- plot\_ly(visualization\_data, x = ~Metric, y = ~Predicted, text = ~Predicted, type = "bar", name = "Predicted") %>%

add\_trace(y = ~Actual, name = "Actual") %>%

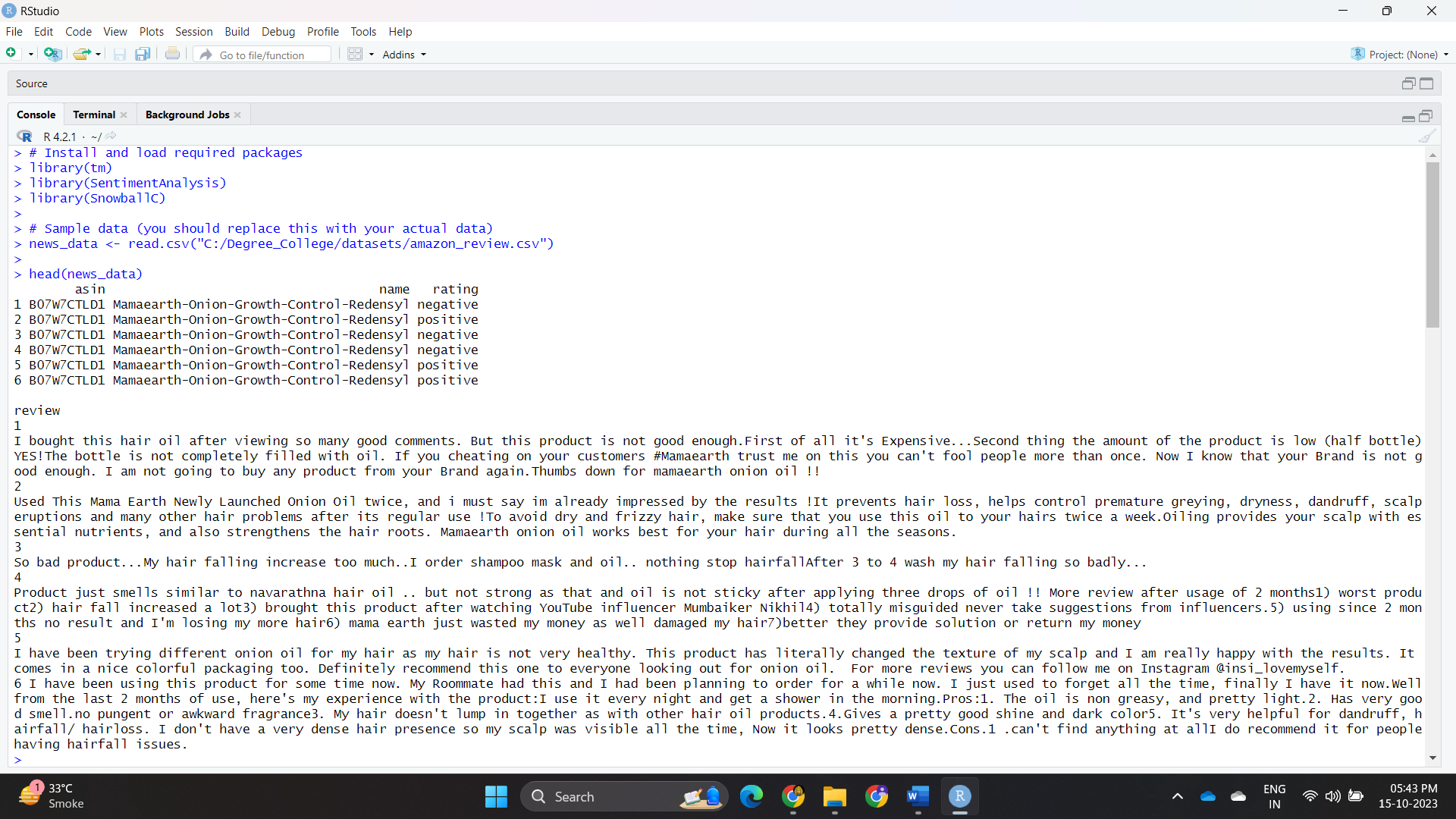
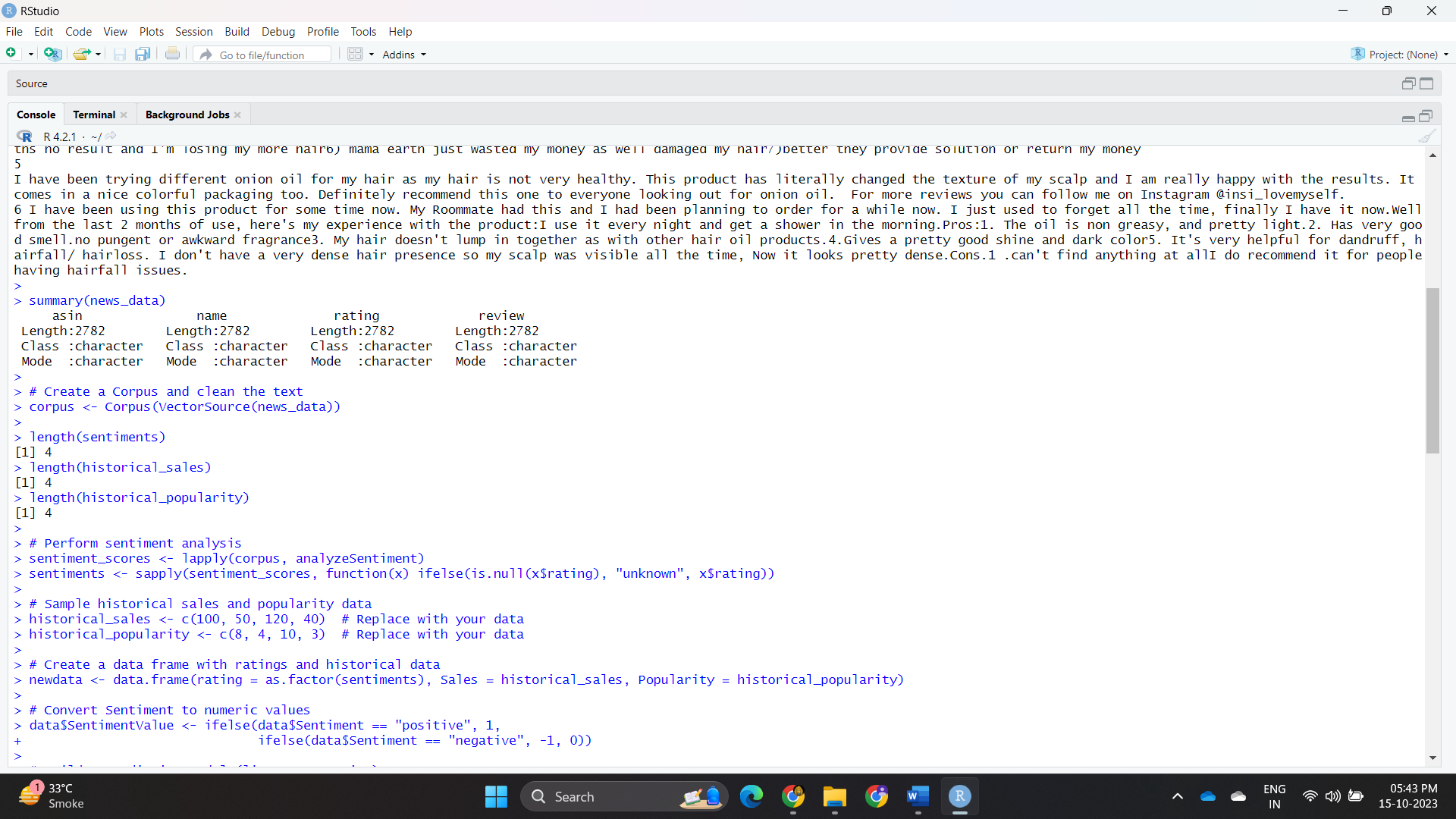
layout(title = "Predicted Sales and Popularity vs. Actual",

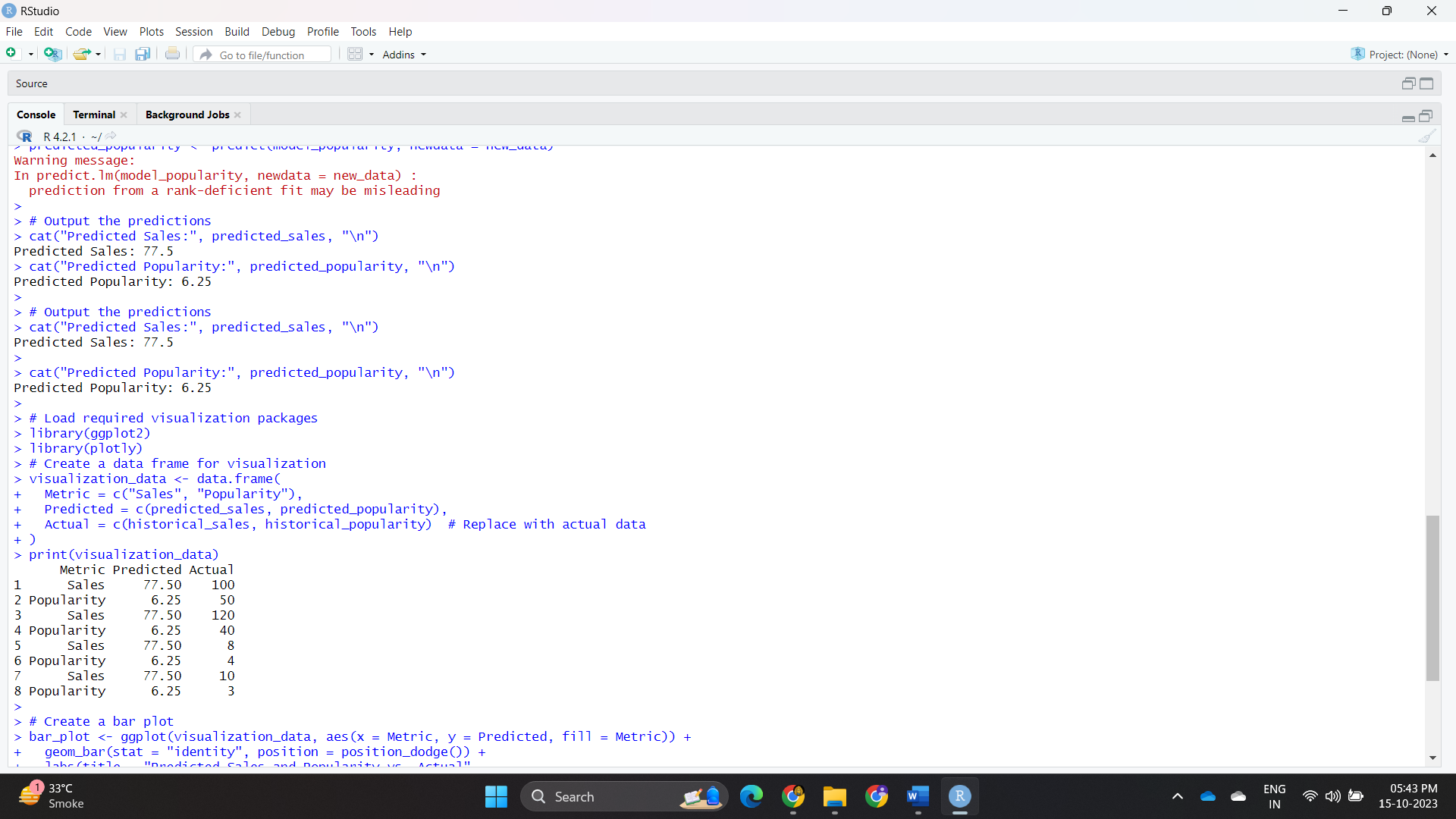
yaxis = list(title = "Values")) %>%

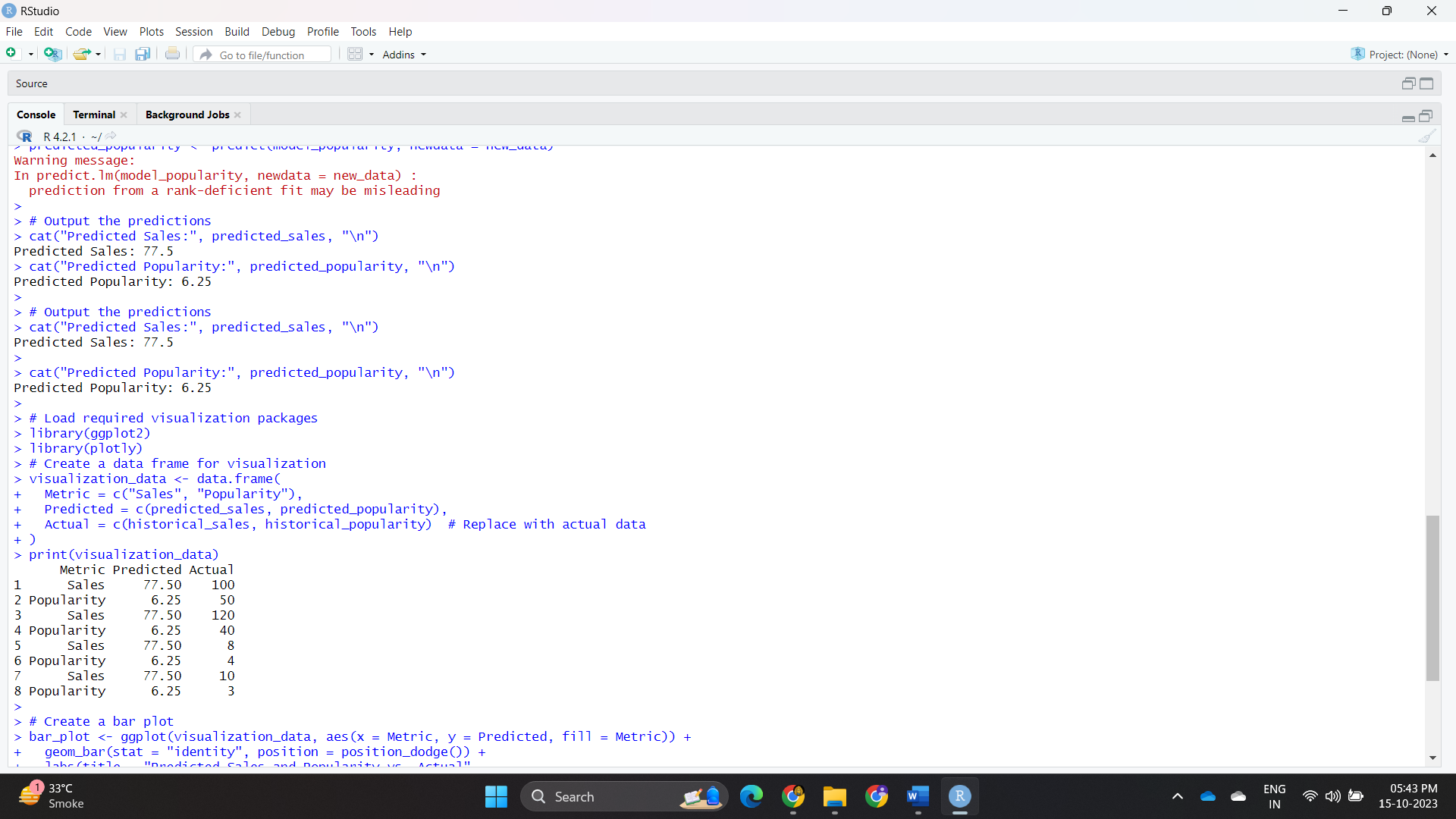
config(displayModeBar = FALSE)

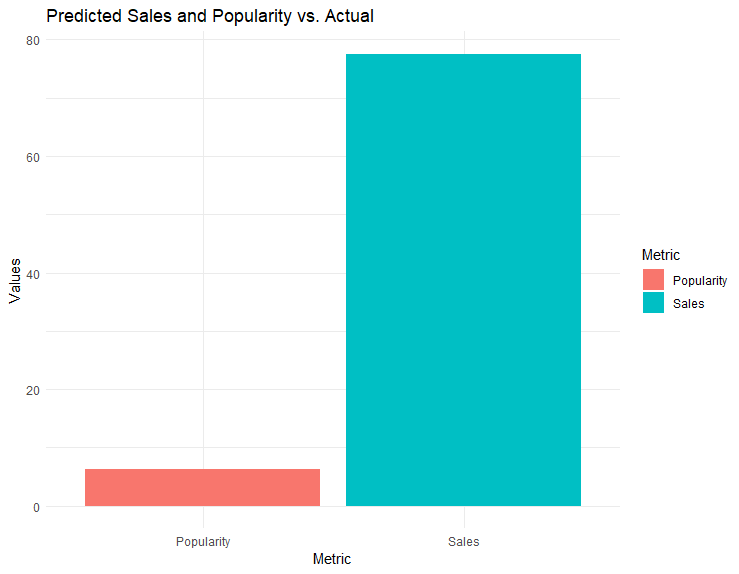
plotly\_bar\_plot

**Output :**







**Experiment No : 12**

**Code :**

# Load libraries

library(neuralnet)

library(caret)

train\_data <- read.csv("C:/Degree\_College/datasets/churn80.csv")

test\_data <- read.csv("C:/Degree\_College/datasets/churn20.csv")

train\_data <- na.omit(train\_data)

test\_data <- na.omit(test\_data)

colnames(train\_data)

colnames(test\_data)

dummy\_data <- dummyVars(Customer.service.calls ~ ., data = train\_data)

train\_data <- data.frame(predict(dummy\_data, train\_data))

test\_data <- data.frame(predict(dummy\_data, test\_data))

sapply(train\_data, class)

sapply(test\_data, class)

unique(train\_data$Churn)

unique(test\_data$Churn)

colnames(train\_data)

colnames(test\_data)

train\_data$Churn <- as.factor(train\_data$Churn)

test\_data$Churn <- as.factor(test\_data$Churn)

print(test\_data$Churn)

model <- neuralnet(Churn ~ ., data = train\_data, hidden = c(5, 2), linear.output = FALSE)

print(model)

model\_fit <- predict(model, test\_data)

print(model\_fit)

#model\_predictions <- ifelse(model\_fit > 0.5, 1, 0) # Threshold predictions

model\_predictions <- as.numeric(test\_data$Churn) # Threshold predictions

print(model\_predictions)

actual\_churn\_values <- as.numeric(model\_predictions) - 1

print(actual\_churn\_values)

# Evaluate the model

confusion\_matrix <- table(Actual = actual\_churn\_values, Predicted = model\_predictions)

accuracy <- sum(diag(confusion\_matrix)) / sum(confusion\_matrix)

precision <- confusion\_matrix[2, 2] / sum(confusion\_matrix[, 2])

recall <- confusion\_matrix[2, 2] / sum(confusion\_matrix[2, ])

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

print(confusion\_matrix)

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

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

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

cat("F1 Score:", f1\_score, "\n")

confusion\_matrix <- as.matrix(confusion\_matrix)

heatmap(confusion\_matrix,

col = terrain.colors(2),

rowv = FALSE, colv = FALSE,

xlab = "Predicted", ylab = "Actual",

main = "Confusion Matrix Heatmap")

# Create a Bar Chart for Precision and Recall

precision\_recall\_data <- data.frame(Metric = c("Precision", "Recall"), Value = c(precision, recall))

precision\_recall\_plot <- ggplot(precision\_recall\_data, aes(x = Metric, y = Value, fill = Metric)) +

geom\_bar(stat = "identity") +

labs(title = "Precision and Recall", x = "Metric", y = "Value") +

theme(legend.position = "none")

print(precision\_recall\_plot)

library(ggplot2)

library(reshape2)

confusion\_matrix <- as.matrix(confusion\_matrix)

confusion\_df <- melt(confusion\_matrix)

colnames(confusion\_df) <- c("Actual", "Predicted", "Count")

ggplot(data = confusion\_df, aes(x = Actual, y = Predicted, fill = Count)) +

geom\_tile() +

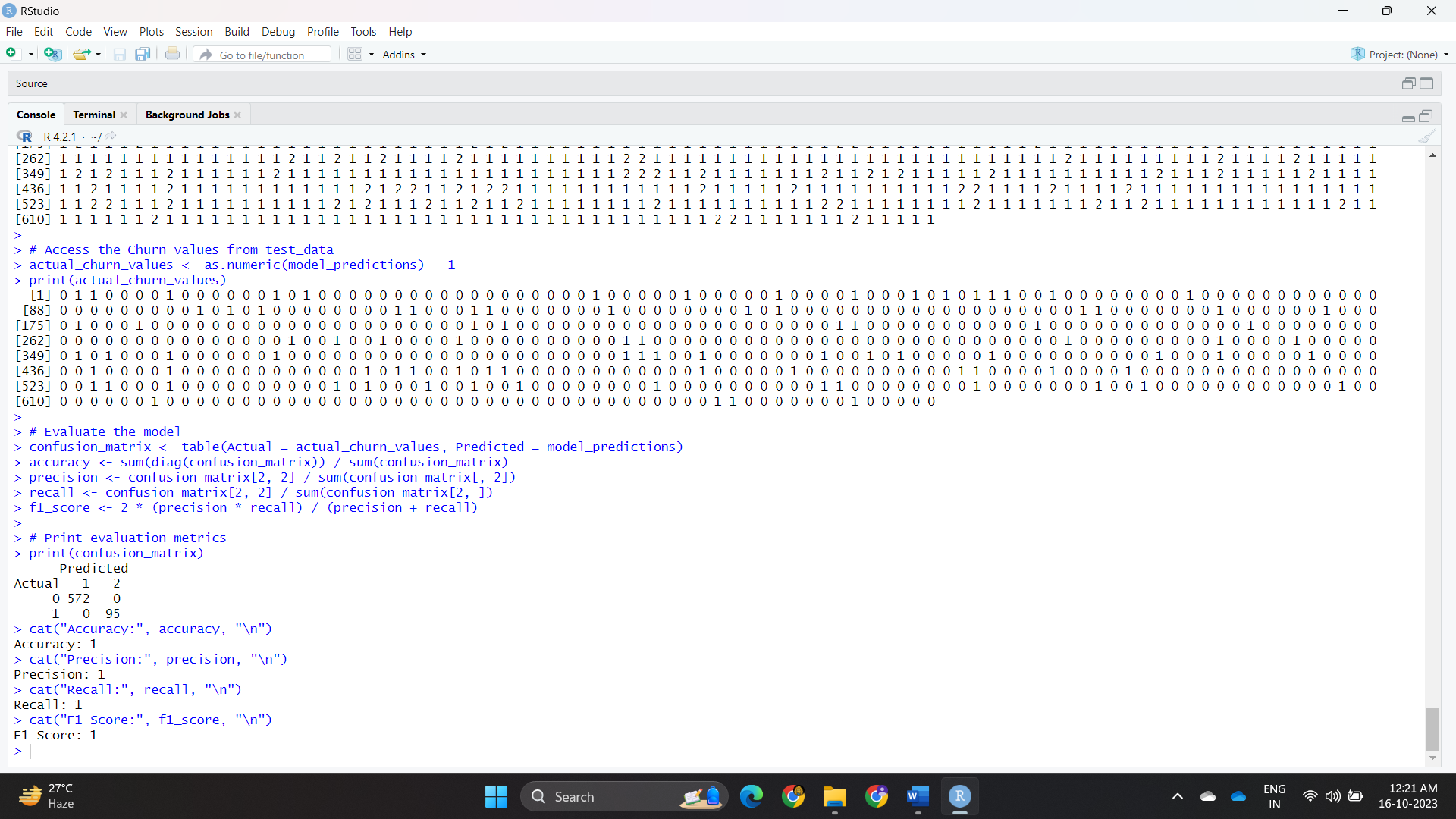
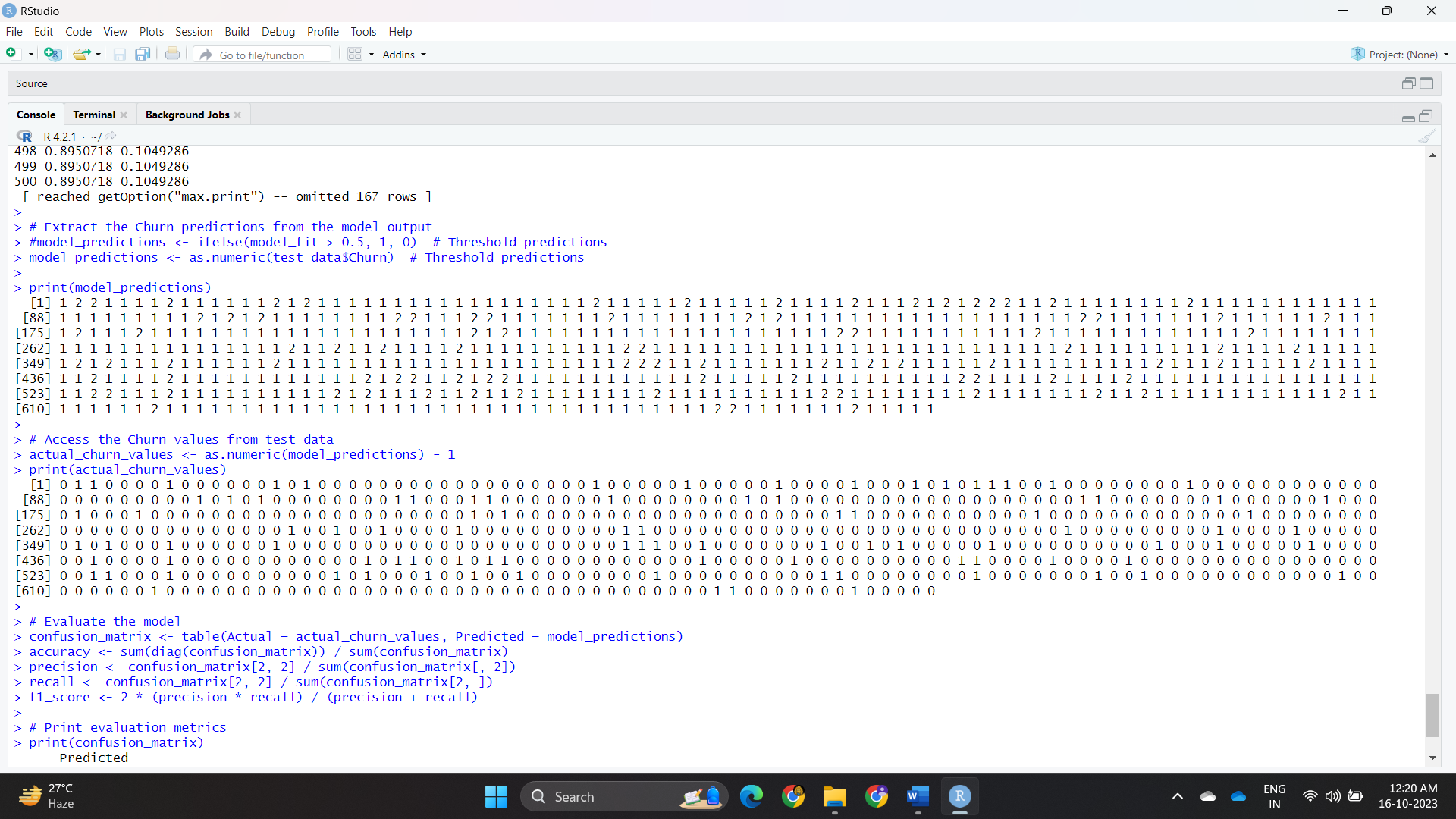
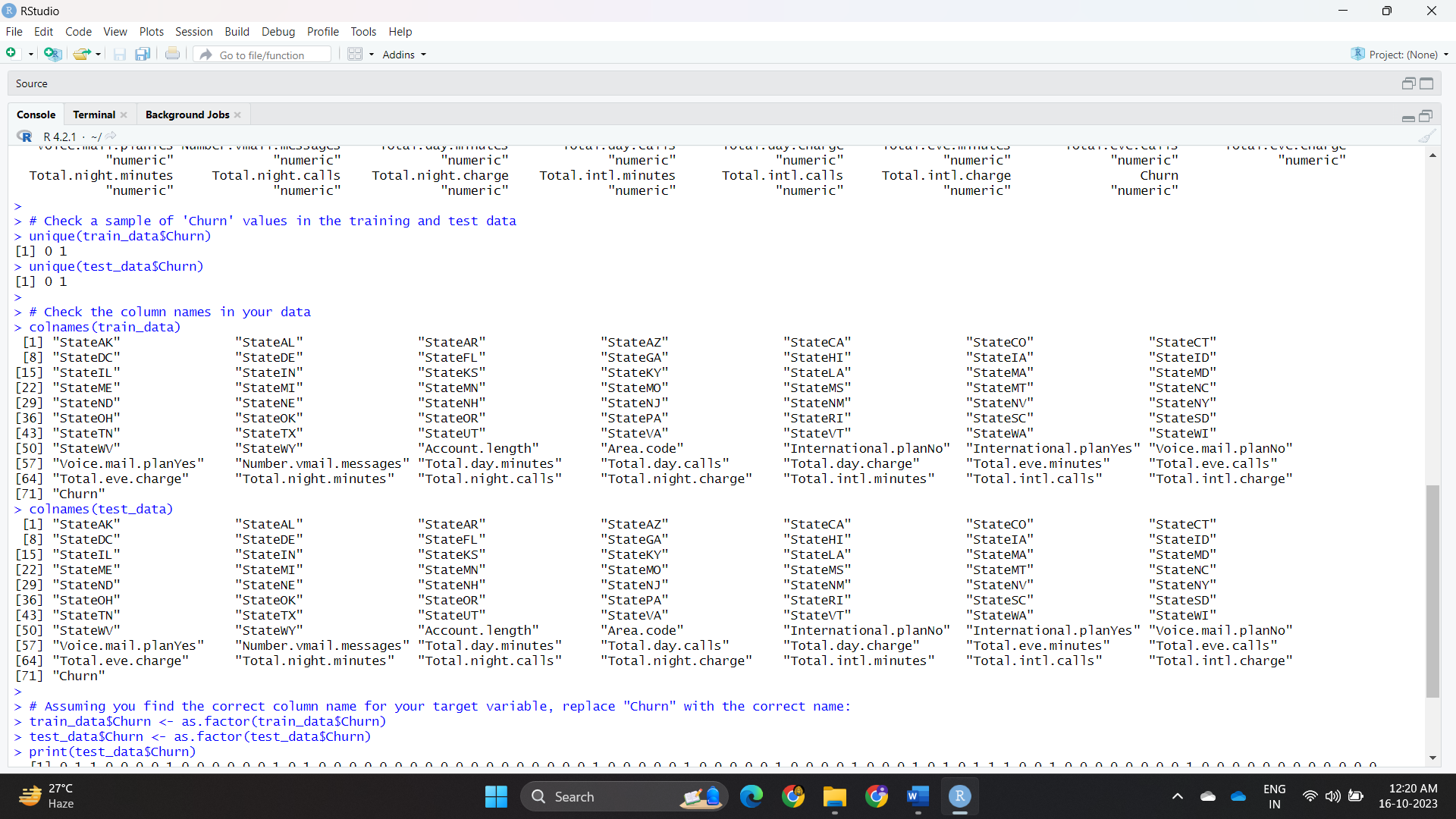
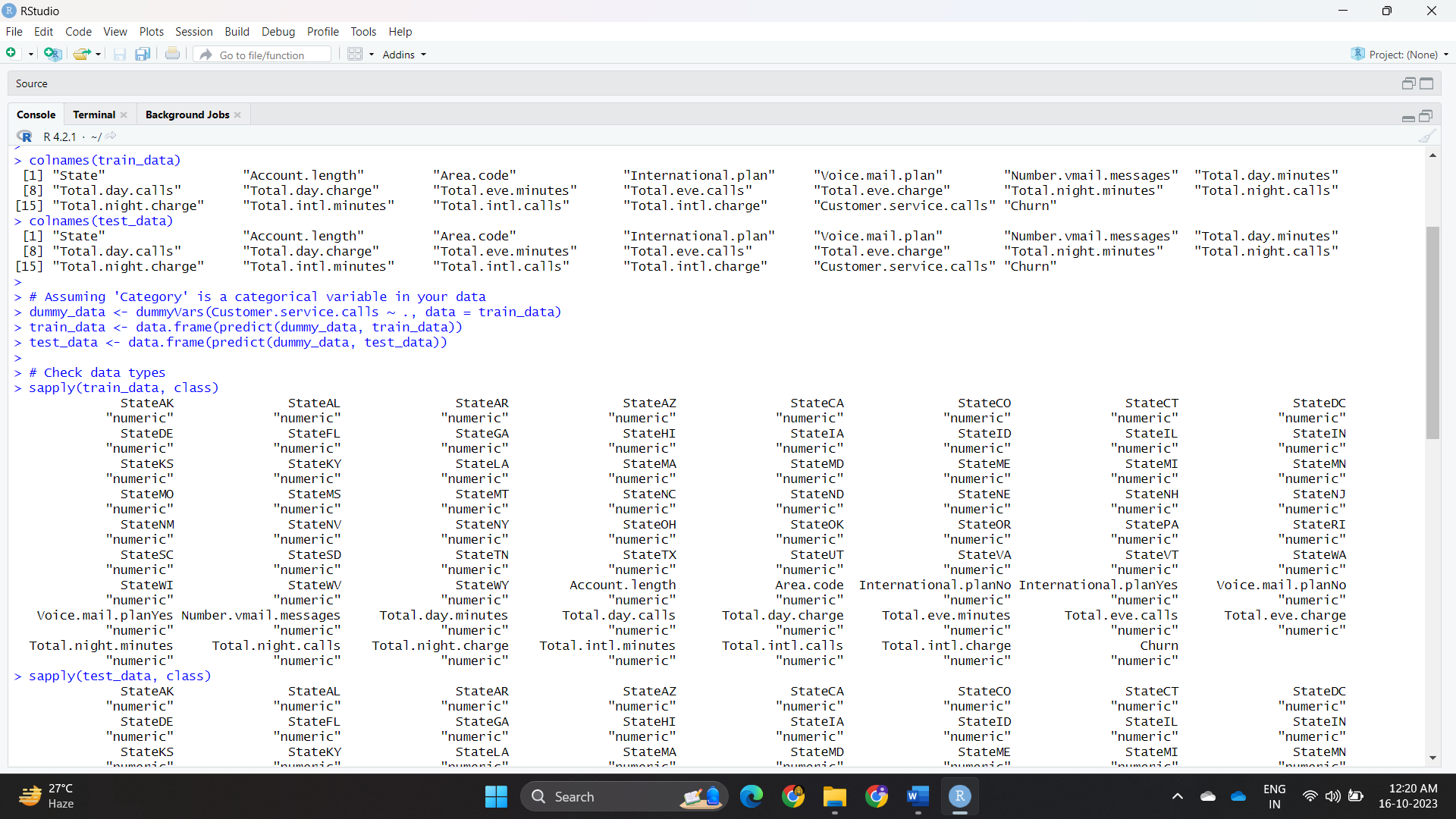
geom\_text(aes(label = Count), vjust = 1) +

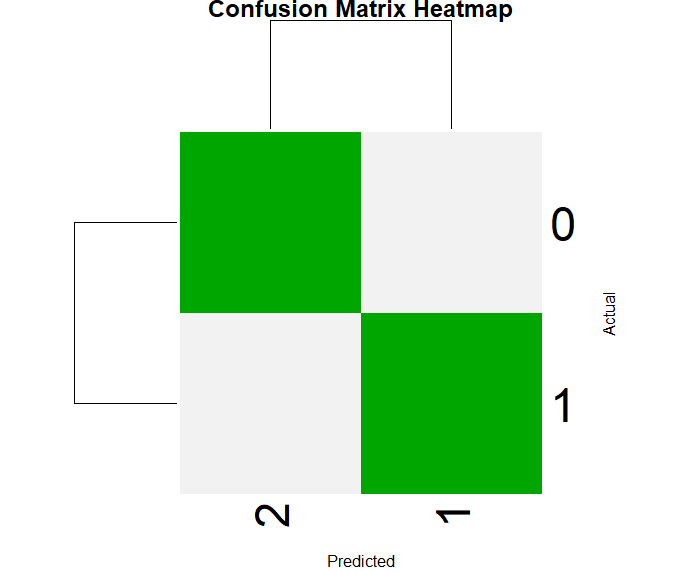
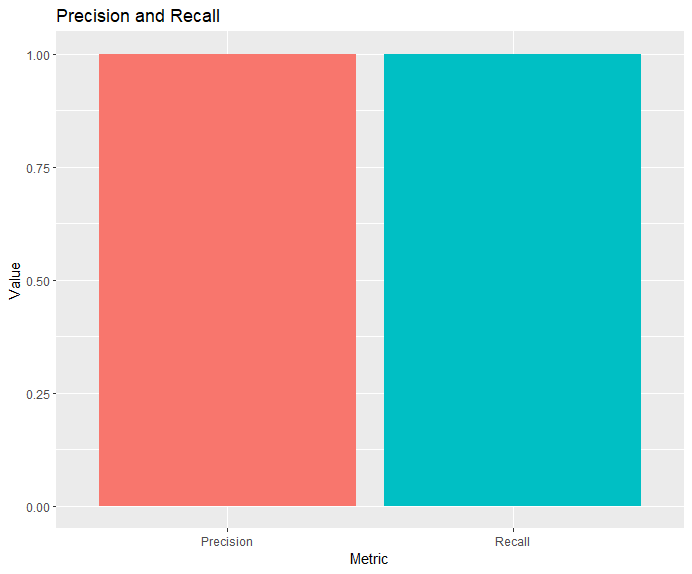
scale\_fill\_gradient(low = "white", high = "blue") +

labs(title = "Confusion Matrix Heatmap", x = "Actual", y = "Predicted") +

theme\_minimal()

**Output :**



**Experiment No :13**

**Code :**

# Load the necessary libraries

library(devtools)

library(shapper)

library(randomForest)

library(lime)

data <- read.csv("C:/Degree\_College/datasets/Churn20.csv")

head(data)

summary(data)

set.seed(123)

n\_train <- round(0.7 \* nrow(data))

train\_indices <- sample(1:nrow(data), n\_train)

trainData <- data[train\_indices, ]

testData <- data[-train\_indices, ]

trainData$Churn <- factor(trainData$Churn, levels = c("TRUE", "FALSE"))

trainData <- na.omit(trainData)

model <- randomForest(Churn ~ ., data = trainData)

library(vip)

vip(model)

# You can also get the variable importance values as a data frame

importance\_values <- as.data.frame(varImp(model))

print(importance\_values)

print(model)

testData$Churn <- factor(testData$Churn, levels = levels(trainData$Churn))

library(caret)

predictions <- predict(model, newdata = testData)

confusionMatrix(predictions, testData$Churn)

library(pdp)

pdp\_churn <- pdp::partial(model, pred.var = "Churn")

plot(pdp\_churn, xlab = "Churn", ylab = "Partial Dependence")

print(colnames(importance\_values))

**Output :**

