**EX.NO 5 DATA PREDICTION ALGORITHM**

**AIM:**

To build and evaluate a Linear Regression model for prediction tasks using R programming language.

**ALGORITHM:**

1. ****Install and Load Required Packages:****

**install.packages("caret")**

**library(caret)**

1. ****Load Dataset:****

**# Load dataset**

**data(mtcars)**

1. ****Data Preprocessing:****

Perform necessary preprocessing steps such as removing missing values, scaling, encoding categorical variables, etc. For example:

# Check for missing values

sum(is.na(mtcars))

# Split data into predictors and target variable

predictors <- mtcars[, -1] # All columns except the first one (mpg)

target <- mtcars[, 1] # First column (mpg)

1. ****Split Data into Training and Testing Sets:****

Split the data into training and testing sets for model evaluation.

# Split data into training and testing sets

set.seed(123)

trainIndex <- createDataPartition(target, p = 0.8, list = FALSE)

trainData <- predictors[trainIndex, ]

testData <- predictors[-trainIndex, ]

trainTarget <- target[trainIndex]

testTarget <- target[-trainIndex]

1. ****Build Linear Regression Model:****

Train a Linear Regression model using the training data.

**# Train Linear Regression model**

**lm\_model <- train(trainData, trainTarget, method = "lm")**

1. ****Model Evaluation:****

Evaluate the performance of the model using the testing data.

# Predict on test data

predictions <- predict(lm\_model, testData)

# Calculate Root Mean Squared Error (RMSE)

rmse <- sqrt(mean((predictions - testTarget)^2))

print(paste("RMSE:", rmse))

1. ****Visualization****

**plot(testTarget, predictions, main = "Actual vs. Predicted", xlab = "Actual", ylab = "Predicted")**

**abline(0, 1, col = "red")**

RESULT:

Thus the Linear Regression model for prediction tasks using R programming language has been executed successfully.

**EX.NO 6A Data Clustering using Partitioning Algorithms**

**(i)K-means algorithm**

#### **AIM:**

To perform data clustering using the K-means algorithm in R.

**Steps:**

**Load Data:**

Load a sample dataset into R. You can use built-in datasets like iris or mtcars, or import your own dataset.

**Preprocess Data:**

If needed, preprocess the data (e.g., scaling/normalization) to ensure all variables have the same scale.

**Choose the Number of Clusters (K):**

Decide the number of clusters you want to create. You can use techniques like the elbow method or silhouette analysis to determine an optimal K value.

**Perform Clustering:**

Use the kmeans() function in R to perform clustering on the dataset. Pass the data and the chosen number of clusters (K) as parameters.

**Visualize Results:**

Plot the clustered data using scatter plots or other suitable visualization techniques. You can use the ggplot2 package for visualization.

**Interpret Results:**

Analyze the clusters formed and interpret the results. Evaluate the cluster centroids, cluster sizes, and the characteristics of data points within each cluster.

**PROGRAM**

# Step 1: Load Data

**data(iris) # Load Iris dataset**

**df <- iris # Create a dataframe**

# Step 2: Preprocess Data (if needed)

# Skip this step if preprocessing is not required

# Step 3: Choose the Number of Clusters (K)

**k <- 3 # Number of clusters**

# Step 4: Perform Clustering

**set.seed(123) # For reproducibility**

**kmeans\_result <- kmeans(df[, -5], centers = k**) # Perform clustering

# Step 5: Visualize Results

**library(ggplot2)**

**ggplot(df, aes(Petal.Length, Petal.Width, color = factor(kmeans\_result$cluster))) + geom\_point() +**

**geom\_point(data = as.data.frame(kmeans\_result$centers[, c("Petal.Length", "Petal.Width")]), color = "black", size = 3, shape = 17) +**

**labs(title = "K-means Clustering of Iris Dataset") +**

**theme\_minimal()**

# Step 6: Interpret Results

# Analyze cluster centroids, cluster sizes, etc.

**RESULT:**

**Thus the** data clustering using the K-means algorithm in R has been executed successfully.

**(6B)K-medoids (PAM - Partitioning Around Medoids)**

**AIM:**

To implement the K-medoids clustering algorithm using the Partitioning Around Medoids (PAM) approach in R

**STEPS:**

#### **1. Data Loading and Exploration:**

* Load the dataset (e.g., iris dataset).
* Explore the structure and summary statistics of the dataset.

#### **2. Data Preprocessing:**

* Select relevant features for clustering.
* Standardize the data to ensure equal contribution of variables.

#### **3. Implement K-medoids (PAM) Algorithm:**

* Choose the number of clusters (K).
* Use the **pam()** function from the **cluster** package to perform K-medoids clustering.
* Extract and analyze clustering results.

#### **4. Visualization:**

* Visualize the clusters using scatter plots or other suitable visualization techniques.
* Interpret the clustering results based on the visualizations

**PROGRAM:**

# Step 1: Load Data

data(iris) # Load Iris dataset

df <- iris # Create a dataframe

# Step 2: Preprocess Data (if needed)

# Skip this step if preprocessing is not required

# Step 3: Choose the Number of Clusters (K)

k <- 3 # Number of clusters

# Step 4: Perform Clustering

library(cluster)

set.seed(123) # For reproducibility

pam\_result <- pam(df[, -5], k) # Perform clustering

# Step 5: Visualize Results

library(ggplot2)

ggplot(df, aes(Petal.Length, Petal.Width, color = factor(pam\_result$clustering))) +

geom\_point() +

geom\_point(data = as.data.frame(pam\_result$medoids[, c("Petal.Length", "Petal.Width")]), color = "black", size = 3, shape = 17) +

labs(title = "K-medoids (PAM) Clustering of Iris Dataset") +

theme\_minimal()

# Step 6: Interpret Results

# Analyze cluster medoids, cluster sizes, etc.

**RESULT:**

Thus the data clustering using K-medoids (PAM - Partitioning Around Medoids) in R has been executed successfully.

**EX.NO 7A Clustering using hierarchical algorithms**

**(i)Dendrogram**

#### **AIM:**

To perform hierarchical clustering using agglomerative (bottom-up) algorithm in R.

#### **Steps:**

****Load Data****:

* + Load a sample dataset into R. You can use built-in datasets like **iris** or **mtcars**, or import your own dataset.

****Preprocess Data****:

* + If needed, preprocess the data (e.g., scaling/normalization) to ensure all variables have the same scale.

****Calculate Distance Matrix****:

* + Compute the pairwise distance matrix between data points using a suitable distance metric (e.g., Euclidean distance, Manhattan distance).

****Perform Clustering****:

* + Use the **hclust()** function in R to perform hierarchical clustering on the distance matrix. Specify the linkage method (e.g., complete linkage, single linkage) as needed.

****Visualize Dendrogram****:

* + Plot the dendrogram to visualize the hierarchical clustering structure. You can use the **plot()** function to plot the dendrogram.

****Cut Dendrogram****:

* + Optionally, cut the dendrogram at a certain height to obtain a specific number of clusters. You can use the **cutree()** function for this purpose.

****Visualize Clustered Data****:

* + Plot the clustered data using scatter plots or other suitable visualization techniques. You can use the **ggplot2** package for visualization.

**PROGRAM:**

# Step 1: Load Data

data(iris) # Load Iris dataset

df <- iris # Create a dataframe

# Step 2: Preprocess Data (if needed)

# Skip this step if preprocessing is not required

# Step 3: Calculate Distance Matrix

dist\_matrix <- dist(df[, -5]) # Compute pairwise distance matrix

# Step 4: Perform Clustering

hc <- hclust(dist\_matrix, method = "complete") # Perform hierarchical clustering

# Step 5: Visualize Dendrogram

plot(hc, main = "Dendrogram of Iris Dataset")

# Step 6: Cut Dendrogram (Optional)

cut\_height <- 2 # Specify the height to cut the dendrogram

clusters <- cutree(hc, h = cut\_height)

# Step 7: Visualize Clustered Data (Optional)

library(ggplot2)

df$cluster <- as.factor(clusters)

ggplot(df, aes(Petal.Length, Petal.Width, color = cluster)) +

geom\_point() +

labs(title = "Hierarchical Clustering of Iris Dataset") +

theme\_minimal()

RESULT:

Thus the hierarchical clustering using agglomerative (bottom-up) algorithm in R has been executed successfully.

### **(7B)Divisive (Top-Down) Hierarchical Clustering**

#### **AIM:**

To perform hierarchical clustering using **Divisive (**top-down manner) algorithm in R

**STEPS:**

# Step 1: Load Data

data(iris)

# Step 2: Preprocess Data (if needed)

# No preprocessing required for this example

# Step 3: Implement Divisive Hierarchical Clustering

divisive\_hc <- function(data, max\_clusters) {

# Define recursive function to divide clusters

divide\_clusters <- function(cluster, max\_clusters) {

if (length(cluster) <= max\_clusters) {

return(list(cluster))

} else {

# Split cluster into two subclusters

# Here, we split the cluster in half for simplicity

midpoint <- length(cluster) %/% 2

subcluster1 <- cluster[1:midpoint]

subcluster2 <- cluster[(midpoint + 1):length(cluster)]

return(c(divide\_clusters(subcluster1, max\_clusters), divide\_clusters(subcluster2, max\_clusters)))

}

}

# Initialize with all data points in a single cluster

initial\_cluster <- 1:nrow(data)

# Perform divisive hierarchical clustering

clusters <- divide\_clusters(initial\_cluster, max\_clusters)

return(clusters)

}

# Call divisive hierarchical clustering function

max\_clusters <- 3 # Specify the maximum number of clusters

divisive\_clusters <- divisive\_hc(iris[, -5], max\_clusters)

# Step 4: Visualize Dendrogram (Optional)

# Since divisive clustering does not directly produce a dendrogram, visualization is not applicable here.

# Step 5: Visualize Clustered Data

# Plot the clustered data using different colors for each cluster

plot(iris[, c("Petal.Length", "Petal.Width")], col = divisive\_clusters$cluster, main = "Divisive Hierarchical Clustering of Iris Dataset")

RESULT:

Thus the hierarchical clustering using **Divisive (**top-down manner) algorithm in R

has been executed successfully.

**EX.NO 8 WEBPAGE RANKING ALGORITHMS**

**AIM:**

To implement the webpage ranking algorithm using R language

**procedure :**

## **1. Open RStudio:**

* Launch the RStudio application on your computer.

## **2. Start a New Script:**

* Click on the File menu in the top-left corner.
* Hover over New File in the dropdown menu.
* Click R Script from the submenu. This will open a new script tab in the script editor pane.

## **3. Write or Paste Your R Program:**

* In the script editor pane, type out your R code

## **4. Save the Script (optional, but recommended):**

* Click on the File menu in the top-left corner.
* Select Save or Save As.
* Choose a location on your computer to save the file.
* Give the file a name, ensuring it has the .R extension, and click Save.

## **5. Run the Program:**

**To Run the Entire Script:** Click on the Source button located at the top-right of the script editor pane (or press Ctrl + Shift + S on the keyboard).

**Program:**

**pageRank <- function(M, num\_iter = 1, d = 0.85) {**

**N <- ncol(M)**

**v <- rep(1/N, N)**

**M\_hat <- d \* M + (1 - d) / N**

**for (i in 1:num\_iter) {**

**v <- M\_hat %\*% v**

**}**

**return(v)**

**}**

**M <- matrix(c(0, 0, 0, 0, 1,**

**0.5, 0, 0, 0, 0,**

**0.5, 1, 0, 0, 0,**

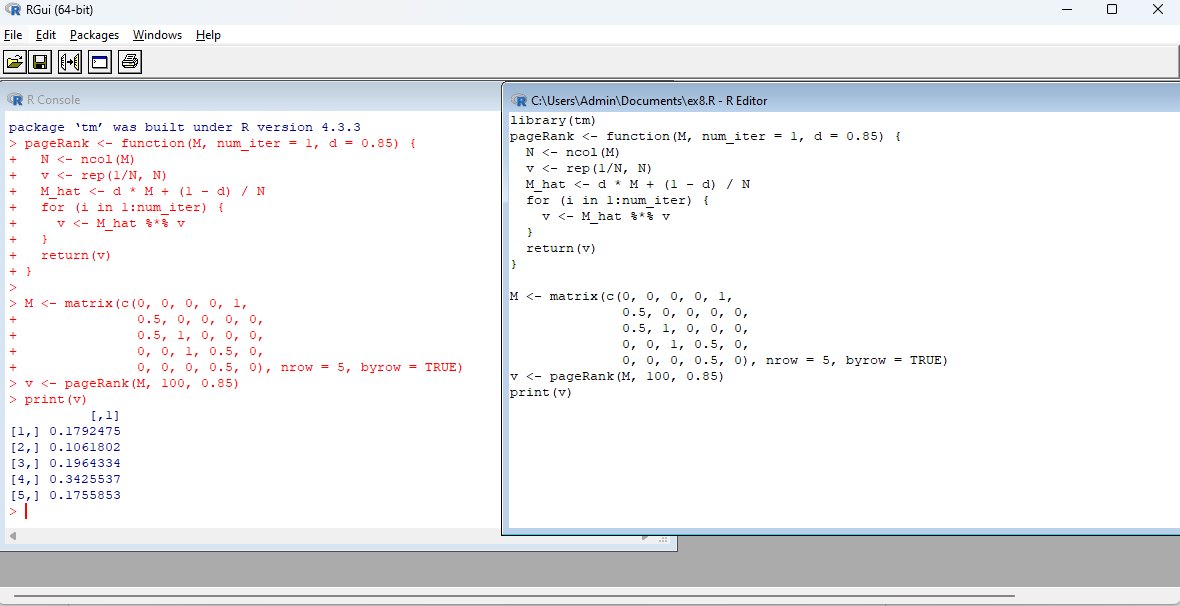
**0, 0, 1, 0.5, 0,**

**0, 0, 0, 0.5, 0), nrow = 5, byrow = TRUE)**

**v <- pageRank(M, 100, 0.85)**

**print(v)**

Output:



RESULT:

Thus the webpage ranking algorithm using R language has been executed successfully.

**EX.NO 9 : TEXT MINING ALGORITHM**

**AIM:** To implement the text mining algorithm using R language

**STEPS:**

1. Open RStudio:

Launch the RStudio application on your computer.

1. Start a New Script:
   * Click on the "File" menu in the top-left corner.
   * Hover over "New File" in the dropdown menu.
   * Click "R Script" from the submenu. This will open a new script tab in the script editor pane.
2. Write or Paste Your R Program:
   * In the script editor pane, type out your R code or paste it if you already have it.
   * install and load the tm and ggplot2 packages ,
3. Save the Script (optional, but recommended):
   * Click on the "File" menu in the top-left corner.
   * Select "Save" or "Save As".
   * Choose a location on your computer to save the file.
   * Give the file a name, ensuring it has the .R extension, and click "Save".
4. Run the Program:
   * To Run the Entire Script: Click on the "Source" button located at the top-right of the script editor pane (or press Ctrl + Shift + S on the keyboard).

**Program:**

# Install

install.packages("tm") # for text mining

install.packages("SnowballC") # for text stemming

install.packages("wordcloud") # word-cloud generator

install.packages("RColorBrewer") # color palettes

# Load

library("tm")

library("SnowballC")

library("wordcloud")

library("RColorBrewer")

filePath <- "H:\KRCE.txt"

text <- readLines(filePath)

docs <- Corpus(VectorSource(text))

inspect(docs)

toSpace <- content\_transformer(function (x , pattern ) gsub(pattern, " ", x))

docs <- tm\_map(docs, toSpace, "/")

docs <- tm\_map(docs, toSpace, "@")

docs <- tm\_map(docs, toSpace, "\\|")

# Convert the text to lower case

docs <- tm\_map(docs, content\_transformer(tolower))

# Remove numbers

docs <- tm\_map(docs, removeNumbers)

# Remove english common stopwords

docs <- tm\_map(docs, removeWords, stopwords("english"))

# Remove your own stop word

# specify your stopwords as a character vector

docs <- tm\_map(docs, removeWords, c("blabla1", "blabla2"))

# Remove punctuations

docs <- tm\_map(docs, removePunctuation)

# Eliminate extra white spaces

docs <- tm\_map(docs, stripWhitespace)

# Text stemming

docs <- tm\_map(docs, stemDocument)

dtm <- TermDocumentMatrix(docs)

m <- as.matrix(dtm)

v <- sort(rowSums(m),decreasing=TRUE)

d <- data.frame(word = names(v),freq=v)

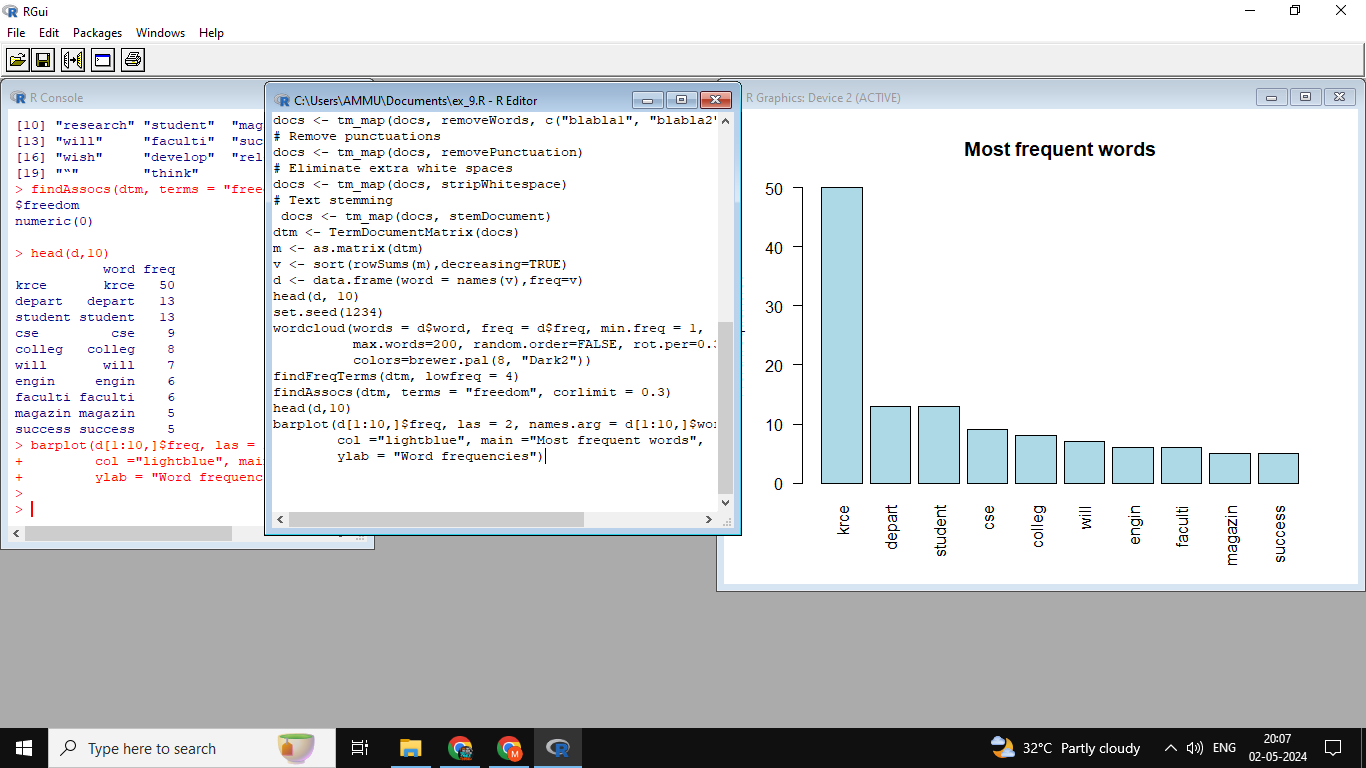
head(d, 10)

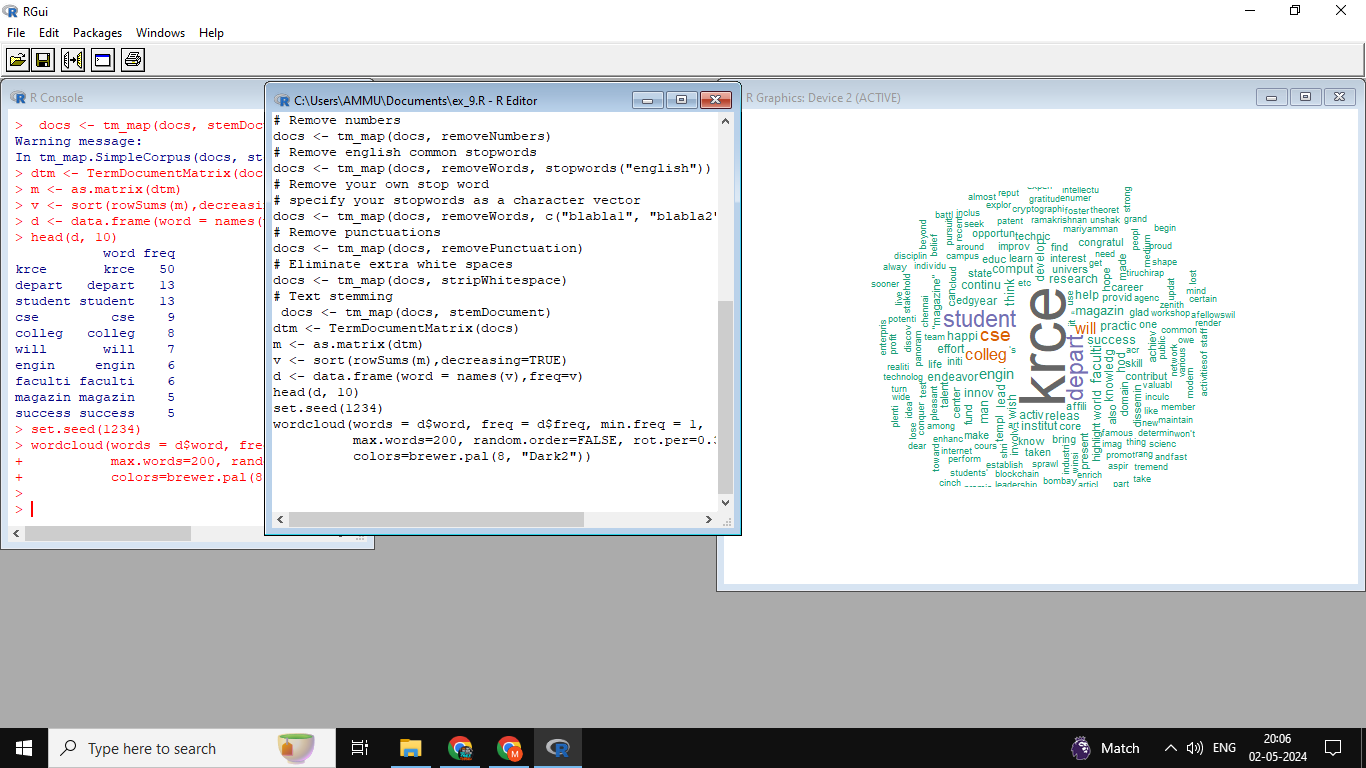
set.seed(1234)

wordcloud(words = d$word, freq = d$freq, min.freq = 1,

max.words=200, random.order=FALSE, rot.per=0.35,

colors=brewer.pal(8, "Dark2"))





RESULT:

Thus the text mining algorithms using R has been executed successfully.