Practical No : 01

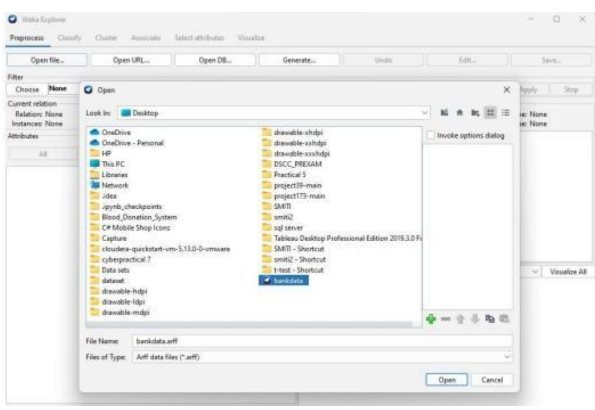
Aim: Pre-process the given data set to perform clustering using various techniques **(WEKA)**

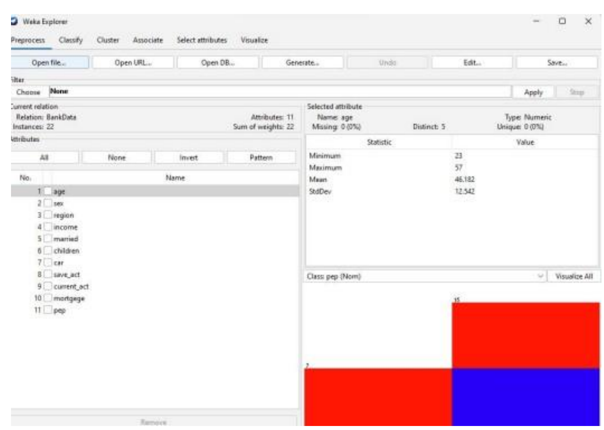
Step1: First create a file with the name bankdata.arff using notepad as shown in the below:

```
relation Banklata
@attribute age numeric
@attribute sex (FEMALE, MALE}
@attribute region (INNER_CITY, TOWN, RURAL, SUBURBAN)
@attribute income numeric
@attribute married (NO, YES)
@attribute children numeric
@attribute car [NO, YES]
@attribute save act (NO, YES}
@attribute current_act (NO, YES)
@attribute mortgege (NO, YES)
@attribute pep (YES, NO)
@data
48,FEMALE, INNER_CITY, 17546, NO, 1, NO, NO, NO, NO, YES
40, MALE, TOWN, 38885.1, YES, 3, YES, NO, YES, YES, NO
51, FEMALE, INNER CITY, 16525.4, YES, 0, YES, YES, YES, NO, NO
23, FEMALE, TOWN, 20375.4, YES, 3, NO, NO, YES, NO, NO
57, FEMALE, RURAL, 50576.3, YES, 0, NO, YES, NO, NO, NO
57, FEMALE, TOWN, 37869.6, YES, 2, NO, YES, YES, NO, YES
48, FEMALE, INNER_CITY, 17546, NO, 1, NO, NO, NO, NO, YES
40, MALE, TOWN, 38085.1, YES, 3, YES, NO, YES, YES, NO
51, FEMALE, INNER_CITY, 16525.4, YES, 0, YES, YES, YES, NO, NO
23, FEMALE, TOWN, 20375.4, YES, 3, NO, NO, YES, NO, NO
57, FEMALE, RURAL, 50576.3, YES, 0, NO, YES, NO, NO, NO
57, FEMALE, TOWN, 37869.6, YES, 2, NO, YES, YES, NO, YES
48 FEMALE, INNER_CITY, 17546, NO, 1, NO, NO, NO, NO, YES
40, MALE, TOWN, 38085.1, YES, 3, YES, NO, YES, YES, NO
51, FEMALE, INNER_CITY, 16525.4, YES, 0, YES, YES, YES, NO, NO
23, FEMALE, TOWN, 20375.4, YES, 3, NO, NO, YES, NO, NO
57, FEMALE, RURAL, 50576.3, YES, O, NO, YES, NO, NO, NO
57, FEMALE, TOWN, 37869.6, YES, 2, NO, YES, YES, NO, YES
51, FEMALE, INNER_CITY, 16525.4, YES, 8, YES, YES, YES, NO, NO
23, FEMALE, TOWN, 20375.4, YES, 3, NO, NO, YES, NO, NO
57, FEMALE, RURAL, 50576.3, YES, 0, NO, YES, NO, NO, NO
57, FEMALE, TOWN, 37869.6, YES, 2, NO, YES, YES, NO, YES
```

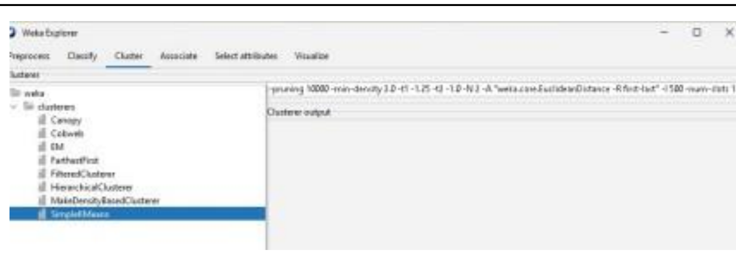
Step2: Open the software Weka and click on Explorer

Step3: Open file bankdata.arff in Weka Explorer.



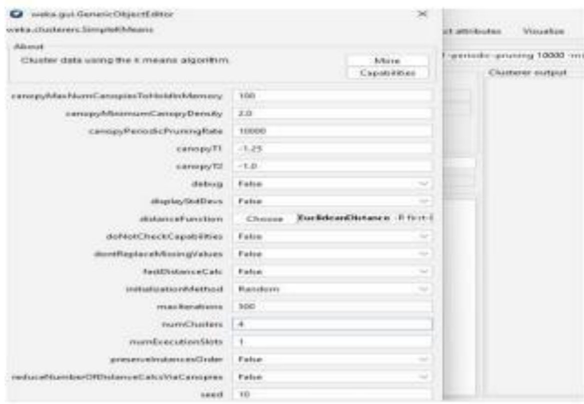


Step4: Go to cluster and choose SimpleKMeans

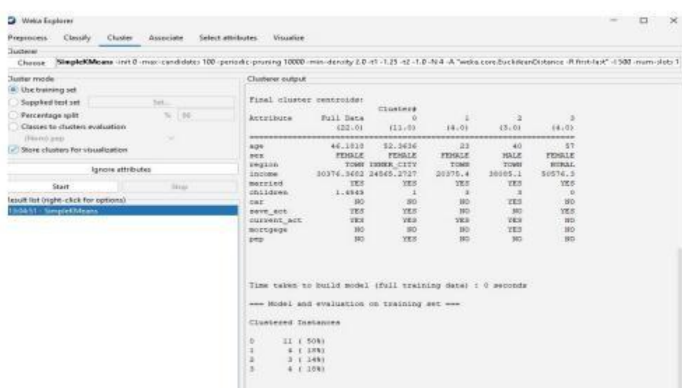


Step5: Right click on cluster and click show properties

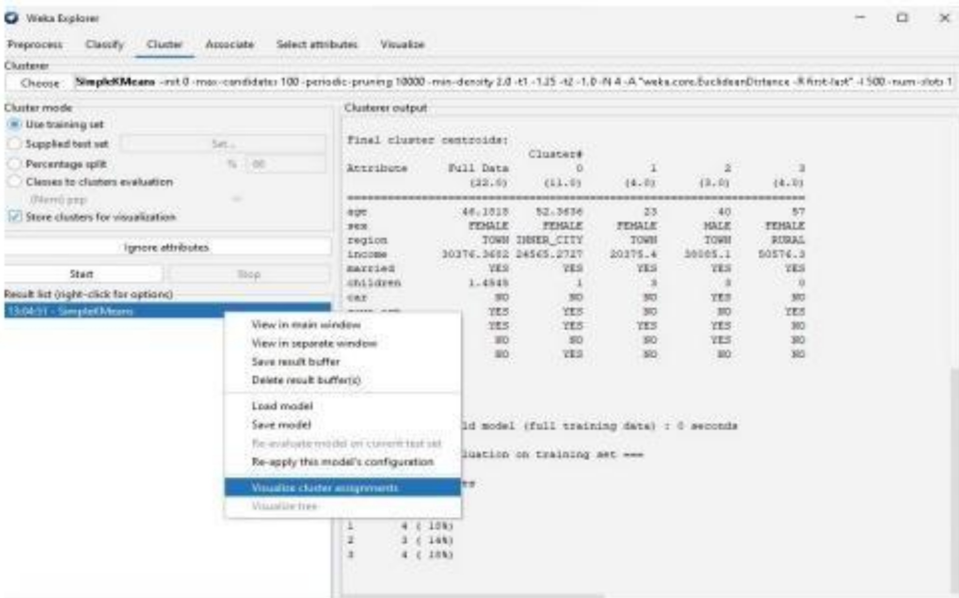
Step 6: Change property numcluster to 4



Step 7: Start then You can see SimpleKMeans output



Step 8: After clustering is done, right click on result list>click on visualize assignments



OUTPUT:

Practical No : 02

Aim: Write a program to implement step-by-step a Collaborative Filtering Recommender System.

Code:- (R studio)

```
user_item_matrix <- matrix(c(5, 3, 0, 1,
                              4, 0, 0, 1,
                              1, 1, 0, 5,
                              0, 1, 5, 4,
                              0, 1, 4, 0),
                            nrow = 5,
                            byrow = TRUE)

cosine_similarity <- function(vec1, vec2) {
  dot_prod <- sum(vec1 * vec2)
  magnitude <- sqrt(sum(vec1^2) * sum(vec2^2))
  if (magnitude == 0) return(0) # to handle cases where denominator is zero
  return(dot_prod / magnitude)
}

find_similar_users <- function(user_id, user_item_matrix) {
  similarities <- numeric()
  for (i in 1:nrow(user_item_matrix)) {
    if (i != user_id) {
      similarities <- c(similarities, cosine_similarity(user_item_matrix[user_id, ],
user_item_matrix[i, ]))
    }
  }
  similar_users <- which(similarities == max(similarities))
  return(similar_users)
}

recommend_items <- function(user_id, user_item_matrix, similar_users,
                             num_recommendations) {
  recommendations <- numeric()
  for (item_id in 1:ncol(user_item_matrix)) {
    if (user_item_matrix[user_id, item_id] == 0) {
      item_score <- 0
```

```
    for (similar_user in similar_users) {
      item_score <- item_score + user_item_matrix[similar_user, item_id]
    }
    recommendations <- c(recommendations, item_score)
   } else {
    recommendations <- c(recommendations, -1) # to indicate already rated items
   }
  }
  recommended_items <- order(recommendations, decreasing =
                              TRUE)[1:num_recommendations]
  return(recommended_items)
}

user_id <- 1
num_recommendations <- 2
similar_users <- find_similar_users(user_id, user_item_matrix)
recommended_items <- recommend_items(user_id, user_item_matrix, similar_users,
                                     num_recommendations)
print(paste("Recommended items for user", user_id, ": ", recommended_items))
```

Output:-

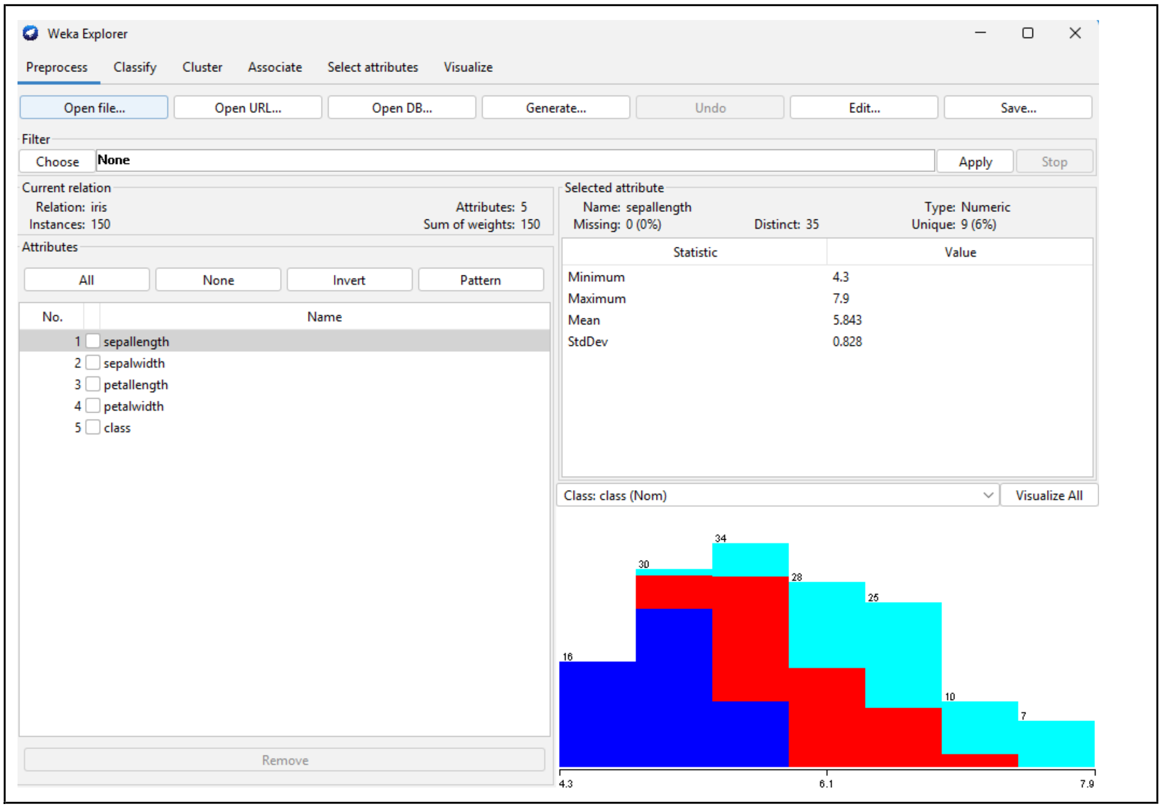
```
[1] "Recommended items for user 1 :  3" "Recommended items for user 1 :  1"
```

Practical No : 03

Aim: Demonstrate an application of Near Neighbor search. (WEKA)

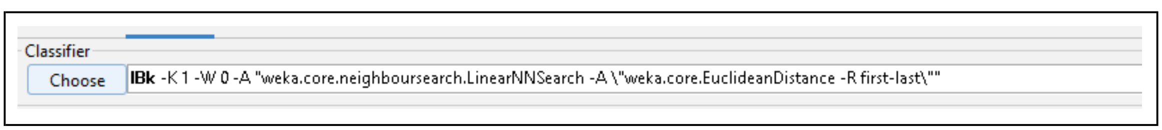
1. **Load Data**

- Open Weka
- Load a dataset (eg, iris atff dataset provided with Weka)



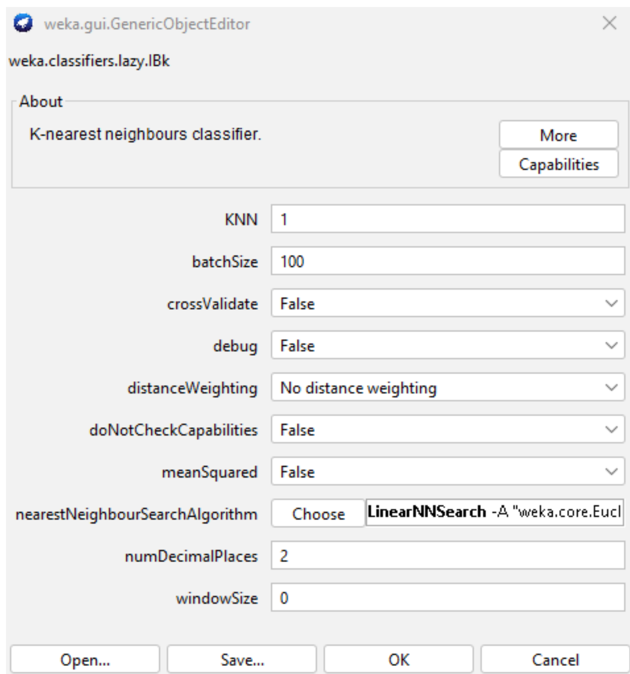
2. **Choose k-NN Algorithm**

- Go to the'Classify" tab
- Select the IBk classifier (this is Weka's implementation of k-Nearest Neighbor)



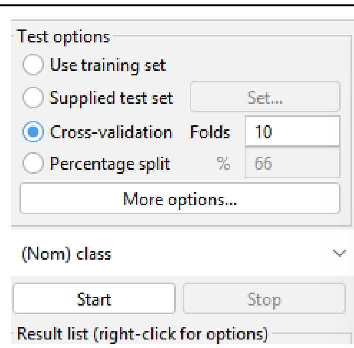
3. Set Parameters:

- Click on IBK
- Set the k parameter (number of neighbors) and other settings, such as distance function (Euclidean, Manhattan, ete.)



4. Run Classiication

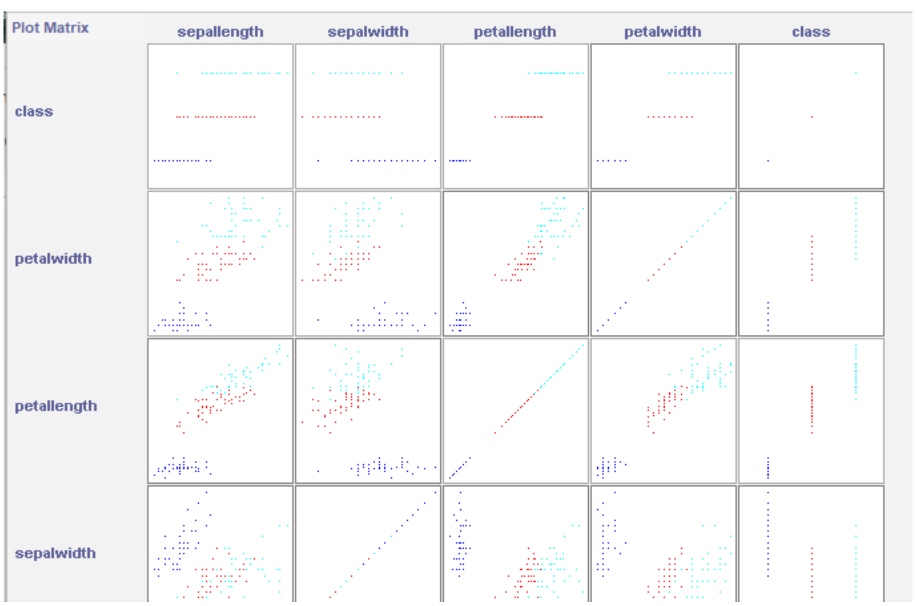
- Choose a test option (e. g.. cross-validation or pereentage splt) Cick "Stat" to run the k-NN classification
- Observe the results, including accuracy and confusion matrix



5. Visualize Results:

- Use the "Visualize" tab to explore how the k-NN algorithm classified instances based on their neighbors

```
Classifier output
=== Run information ===

Scheme:       weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.Euclidear
Relation:     iris
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         143               95.3333 %
Incorrectly Classified Instances         7                4.6667 %
Kappa statistic                          0.93
Mean absolute error                      0.0399
Root mean squared error                  0.1747
Relative absolute error                  8.9763 %
Root relative squared error             37.0695 %
Total Number of Instances              150
```

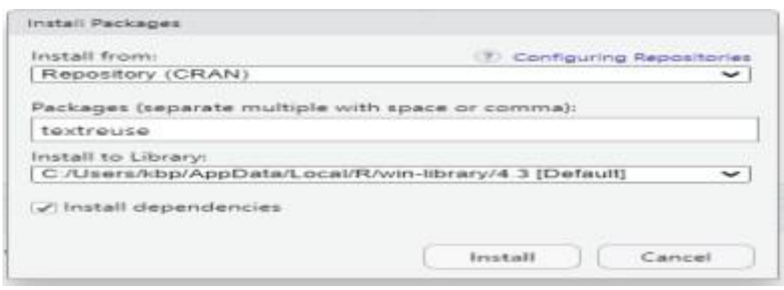


Practical No : 04

Aim: Write a program for measuring similarity among documents and detecting passages which have been reused.(BI 2)

Text Files:

Note: Install package textreuse before writing code

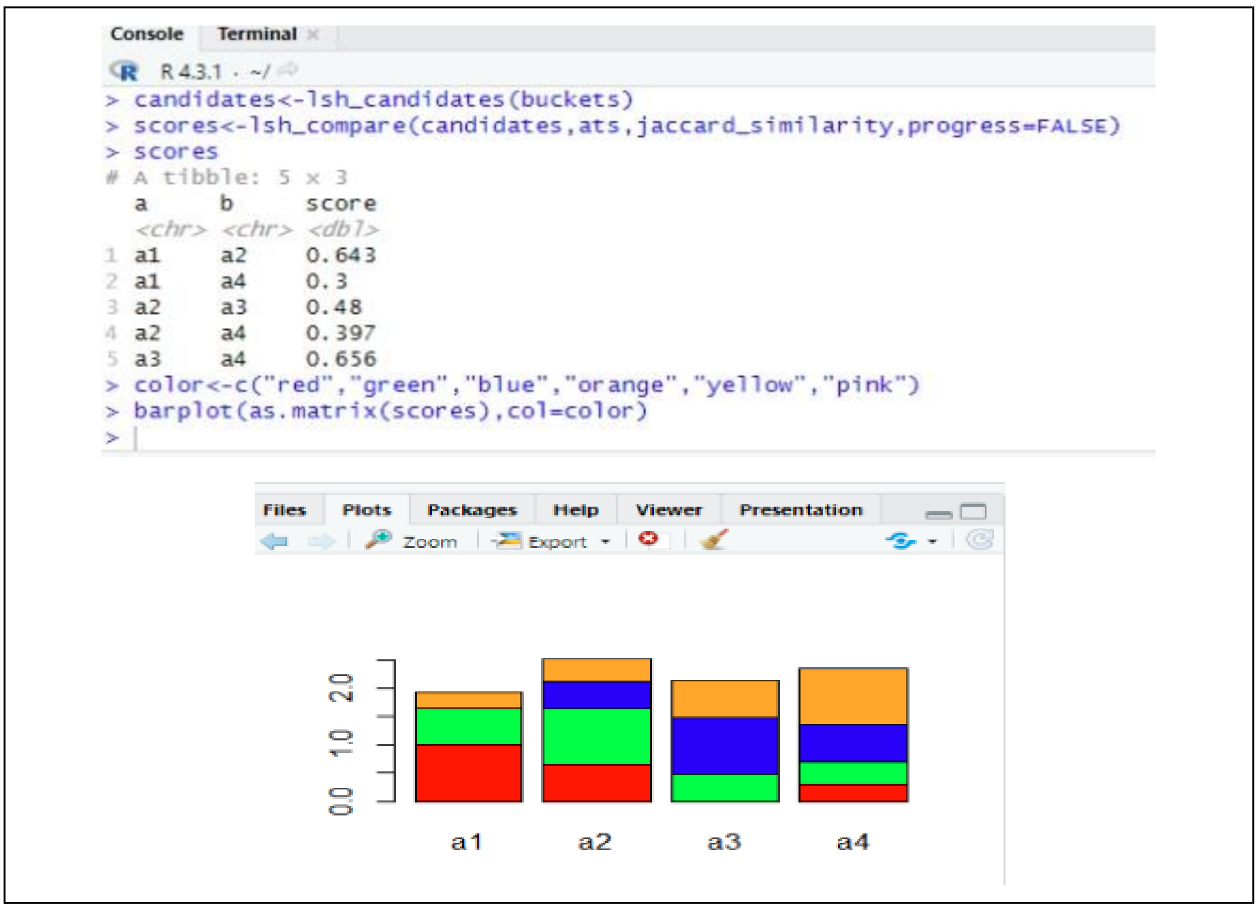



```
package 'pillar' successfully unpacked and MD5 sums checked
package 'tidyselect' successfully unpacked and MD5 sums checked
package 'stringi' successfully unpacked and MD5 sums checked
package 'fansi' successfully unpacked and MD5 sums checked
package 'purrr' successfully unpacked and MD5 sums checked
package 'assertthat' successfully unpacked and MD5 sums checked
package 'digest' successfully unpacked and MD5 sums checked
package 'dplyr' successfully unpacked and MD5 sums checked
package 'NLP' successfully unpacked and MD5 sums checked
package 'RcppProgress' successfully unpacked and MD5 sums checked
package 'stringr' successfully unpacked and MD5 sums checked
package 'tibble' successfully unpacked and MD5 sums checked
package 'tidyr' successfully unpacked and MD5 sums checked
package 'BH' successfully unpacked and MD5 sums checked
package 'textreuse' successfully unpacked and MD5 sums checked
```

CODE : (R Studio)

```
library(textreuse)
minhash<-minhash_generator(200,seed=235)
ats<-TextReuseCorpus(dir="C:/textfiles",tokenizer =
tokenize_ngrams,n=5,minhash_func = minhash)
buckets<-lsh(ats,bands=50,progress=interactive())
candidates<-lsh_candidates(buckets)
scores<-lsh_compare(candidates,ats,jaccard_similarity,progress=FALSE)
scores
color<-c("red","green","blue","orange","yellow","pink")
barplot(as.matrix(scores),col=color)
```

OUTPUT:



Practical No : 05

Aim: Demonstrate an application of Locality sensitive hashing technique for large datasets.

CODE : (Google Coolab)

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.random_projection import SparseRandomProjection
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt

# Sample dataset: Text documents
documents = [
    "Data science is an inter-disciplinary field.",
    "Machine learning is a subset of data science.",
    "Deep learning is a part of machine learning.",
    "Artificial intelligence includes machine learning and deep
learning.",
    "Data mining is a technique used in data science.",
    "Machine learning and data mining are related fields.",
    "Artificial intelligence is transforming industries."
]

# 1. Convert text data to TF-IDF feature vectors
vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer.fit_transform(documents)

# 2. Apply LSH using Random Projection
def apply_lsh(features, n_components=2):
    lsh = SparseRandomProjection(n_components=n_components)
    transformed_features = lsh.fit_transform(features)
    return transformed_features

# 3. Apply LSH to reduce dimensionality of TF-IDF features
lsh_features = apply_lsh(tfidf_matrix.toarray())

# 4. Find similar documents using Nearest Neighbors (LSH based)
```

```
def find_similar_documents(lsh_features, query_index, n_neighbors=3):
    nn = NearestNeighbors(n_neighbors=n_neighbors, metric='cosine')
    nn.fit(lsh_features)
    distances, indices = nn.kneighbors([lsh_features[query_index]])
    return indices

# 5. Find similar documents (e.g., for the first document)
query_index = 0
similar_documents_indices = find_similar_documents(lsh_features,
query_index)

# 6. Display the results
print(f"Query Document: {documents[query_index]}")
print("\nSimilar Documents:")
for idx in similar_documents_indices[0]:
    print(f"- {documents[idx]}")
```

OUTPUT:

```
Query Document: Data science is an inter-disciplinary field.

Similar Documents:
- Data science is an inter-disciplinary field.
- Data mining is a technique used in data science.
- Artificial intelligence includes machine learning and deep learning.
```

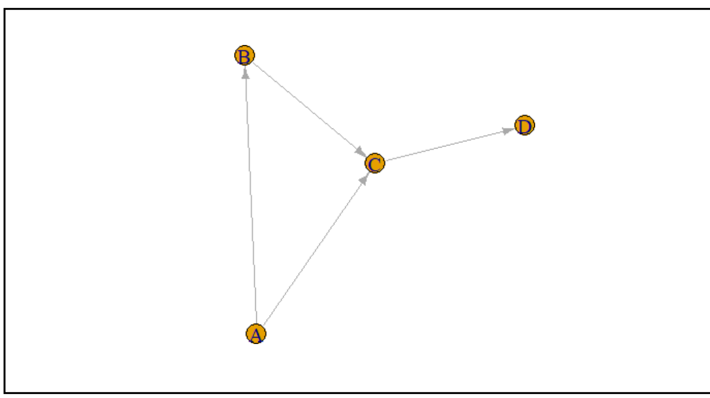
Practical No : 06

Aim: Write a program to explain links to establish higher-order relationships among entities in Link Analysis.(BI 3)

CODE : (R Studio)

```
add_edge <- function(graph, node1, node2) {
graph[[node1]] <- c(graph[[node1]], node2)
graph[[node2]] <- c(graph[[node2]], node1)
graph
}
graph <- list()
nodes <- c("A", "B", "C", "D")
graph <- add_edge(graph, "A", "B")
graph <- add_edge(graph, "A", "C")
graph <- add_edge(graph, "B", "C")
graph <- add_edge(graph, "C", "D")
for (node in names(graph)) {
cat(node, " --> ", paste(graph[[node]], collapse = ", "), "\n")
}
library(igraph)
nodes <- c("A", "B", "C", "D")
edges <- c("A", "B", "A", "C", "B", "C", "C", "D")
graph <- graph(edges, directed = TRUE)
V(graph)$label <- nodes
plot(graph, layout = layout_nicely(graph), edge.arrow.size = 0.5)
```

OUTPUT :




```
+ }
A  -->  B, C
B  -->  A, C
C  -->  A, B, D
D  -->  C
```

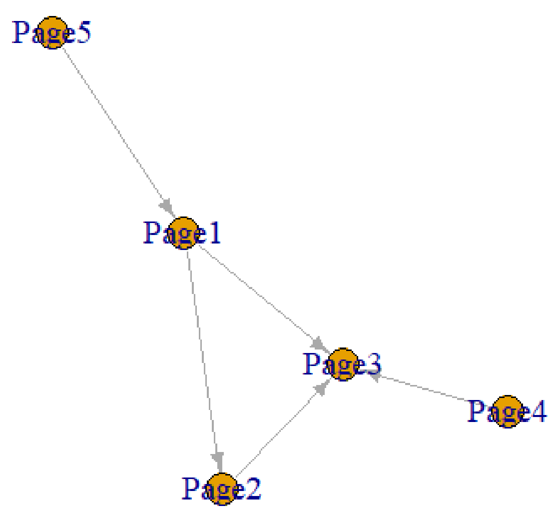
Practical No : 07

Aim: Demonstrate page ranking with an appropriate application.

CODE : (R Studio)

```
library(igraph)
pages<- c("Page1", "Page2", "Page3", "Page4", "Page5")
links<- c("Page1", "Page2", "Page1", "Page3", "Page2", "Page3", "Page4",
         "Page3", "Page5")
graph <-graph(edges = links, directed=TRUE)
page_rank<-(graph)$vector
plot(graph, layout = layout_nicely(graph), edge.arrow.size = 0.5)
pagerank_df <- data.frame(Page = names(pagerank), PageRank = pagerank)
pagerank_df
pagerank_df <- pagerank_df[order(pagerank_df$PageRank, decreasing =
                                    TRUE), ] print(pagerank_df)
```

OUTPUT:



Practical No : 08

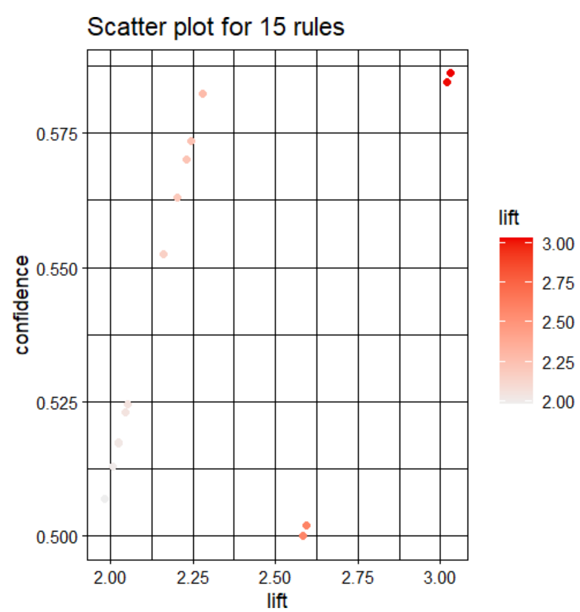
Aim: Develop an application to implement the apriori algorithm

CODE: (R Studio)

```
install.packages("arules")
install.packages("arulesViz")
library(arules)
library(arulesViz)
data("Groceries")
rules <- apriori(Groceries,
            parameter = list(supp = 0.01, conf = 0.5, minlen = 2))
inspect(rules[1:10])
plot(rules, method = "scatter", measure = "lift")
```

OUTPUT:

```
> inspect(rules[1:10])
    lhs                    rhs                support confidence
coverage    lift count
[1]  {curd,
      yogurt}            => {whole milk}      0.01006609
0.5823529 0.01728521 2.279125    99
[2]  {other vegetables,
      butter}            => {whole milk}      0.01148958
0.5736041 0.02003050 2.244885   113
[3]  {other vegetables,
      domestic eggs}     => {whole milk}
0.01230300  0.5525114 0.02226741 2.162336   121
[4]  {yogurt,
      whipped/sour cream} => {whole milk}
0.01087951  0.5245098 0.02074225 2.052747   107
[5]  {other vegetables,
      whipped/sour cream} => {whole milk}
0.01464159  0.5070423 0.02887646 1.984385   144
[6]  {pip fruit,
      other vegetables}  => {whole milk}
0.01352313  0.5175097 0.02613116 2.025351   133
[7]  {citrus fruit,
      root vegetables}   => {other vegetables}
0.01037112  0.5862069 0.01769192 3.029608   102
[8]  {tropical fruit,
      root vegetables}   => {other vegetables}
0.01230300  0.5845411 0.02104728 3.020999   121
```



Practical No : 09

Aim: Write a map-reduce program to count the number of occurrences of each alphabetic character in the given dataset. The count for each letter should be case-insensitive (i.e., include both upper-case and lower-case versions of the letter; Ignore non-alphabetic characters).

CODE: (Google Colab)

```
from collections import Counter
import re

# Function to map: Convert text to lowercase and return list of characters
def map_function(text):
    # Remove non-alphabetic characters and convert text to lowercase
    cleaned_text = re.sub(r'[^a-zA-Z]', '', text.lower())
    # Return list of characters
    return list(cleaned_text)
# Function to reduce: Count occurrences of each character
def reduce_function(mapped_data):
    # Use Counter to count occurrences of each character
    return dict(Counter(mapped_data))
# Path to your text file
file_path = r"/map.txt"
# Read the content of the file
with open(file_path, 'r') as file:
    dataset = file.read()
# Simulating the MapReduce process:
# Step 1: Apply the map function
mapped_data = map_function(dataset)
# Step 2: Apply the reduce function to count occurrences of each character
reduced_data = reduce_function(mapped_data)
# Display the result
print("Character Occurrences:")
for char, count in reduced_data.items():
    print(f"{char}: {count}")
```

OUTPUT:

```
Character Occurrences:
h: 6
e: 13
l: 6
o: 9
w: 1
r: 8
d: 4
t: 12
i: 6
s: 11
a: 15
m: 4
p: 3
c: 12
n: 8
g: 1
b: 2
u: 4
k: 1
f: 1
```

Practical No : 10

Aim: Write a map-reduce program to count the number of occurrences of each word in the given dataset. (A word is defined as any string of alphabetic characters appearing between non-alphabetic characters like nature's is two words. The count should be case-insensitive. If a word occurs multiple times in a line, all should be counted)

CODE: (Google Colab)

```
import re
from functools import reduce
from collections import defaultdict
# Sample dataset
dataset = [
    "Nature's beauty is unmatched in Nature's own way.",
    "The quick brown fox jumps over the lazy dog.",
    "Hello world! This is a test, hello world."
]
# Mapper function
def mapper(line):
    # Use regular expression to split words, considering alphabetic characters only.
    words = re.findall(r'[a-zA-Z]+', line.lower())
    return words
# Reducer function
def reducer(accumulated_counts, word_list):
    for word in word_list:
        accumulated_counts[word] += 1
    return accumulated_counts
# Map function
mapped_data = map(mapper, dataset)
# Reduce function: Accumulate word counts
word_counts = reduce(reducer, mapped_data, defaultdict(int))
# Print result
for word, count in word_counts.items():
    print(f'{word}: {count}')
```

OUTPUT:

```
nature: 2
s: 2
beauty: 1
is: 2
unmatched: 1
in: 1
own: 1
way: 1
the: 2
quick: 1
brown: 1
fox: 1
jumps: 1
over: 1
lazy: 1
dog: 1
hello: 2
world: 2
this: 1
a: 1
test: 1
```