GOKHALE EDUCATION SOCIETY'S

N. B. MEHTA (V) SCIENCE COLLEGE

ACHARYA BHISE VIDYANAGAR, BORDI

TAL - DAHANU, DIST. PALGHAR, 401701, Tel. 02528 - 254357

DEPARTMENT OF INFORMATION TECHNOLOGY AND COMPUTER SCIENCE

Certificate

Class _____ Year _____

This is to certify that the work entered in this journal is the work of Shri/Kumari _____

Of _____ division_____ Roll No. _____

Uni. Exam No. _____ has satisfactorily completed the required University of Mumbai Practicals on Subject PSCSP203: Web Mining and worked for the Master of Computer Science 1st term / 2nd term/ both the terms of the Year _____ in the college laboratory as laid down by the university.

_____	_____	_____
Head of the Department	External Examiner	Internal Examiner Subject teacher

Date: / / 2023 Department of IT-CS

INDEX

SR.No	Practicals	Page no	Date	Sign
01	Page Rank for link analysis using python Create a small set of pages namely page1, page2, page3 and page4 apply random walk on the same	2		
02	Perform spam classifier	6		
03	Demonstrate Text Mining and Webpage Pre-processing using meta information from the web pages (Local/Online)	11		
04	Apriori Algorithm implementation in case study	13		
05	Develop a basic crawler for the web search for user defined keywords	16		
06	Develop a focused crawler for local search	17		
07	Develop a program for deep search implementation to detect plagiarism in documents online	18		
08	Sentiment analysis for reviews by customers and visualize the same	20		

Practical No. 1

Aim: Page Rank for link analysis using python Create a small set of pages namely page1, page2, page3 and page4 apply random walk on the same.

the PageRank algorithm calculates the importance of web pages based on their incoming links. Without the random walk component, the PageRank scores will be determined solely based on the structure of the web pages and the links between them. The scores will reflect the importance of each page as determined by the incoming links from other pages.

Random Walk: The random walk sequence represents the path taken during the random walk. Each page is visited multiple times, and the order in which they are visited is determined randomly. In the given sequence, the random walk starts at page3, then goes to page2, page4, page4 again, page3, and so on.

PageRank Scores: The PageRank scores represent the importance or ranking of each page based on the random walk. The scores indicate the probability of landing on a specific page during a random walk.

Code :

```
import random

# Create a dictionary representing the link structure

links = {

   >‘page1>‘: [>‘page2>‘, >‘page3>‘],

   >‘page2>‘: [>‘page1>‘, >‘page4>‘],

   >‘page3>‘: [>‘page2>‘],

   >‘page4>‘: []

}

# Define the number of iterations for the random walk

num_iterations = 100

# Perform the random walk

current_page = random.choice(list(links.keys()))
```

```
random_walk = [current_page]
for _ in range(num_iterations):
    # Get the outgoing links from the current page
    outgoing_links = links[current_page]

    if outgoing_links:
        # Randomly choose the next page to visit from the outgoing links
        next_page = random.choice(outgoing_links)
    else:
        # If the current page has no outgoing links, restart the random walk from a random page

        next_page = random.choice(list(links.keys()))

    # Append the next page to the random walk sequence
    random_walk.append(next_page)

    # Update the current page
    current_page = next_page
# Print the random walk sequence
print(&amp;quot;Random Walk:&amp;quot;)
print(>‘ -> >‘.join(random_walk))
# Calculate the PageRank scores
pagerank_scores = {}
total_pages = len(links)
for page in links.keys():
    page_visits = random_walk.count(page)
    pagerank_scores[page] = page_visits / num_iterations
```

```
# Print the PageRank scores
print(&amp;quot;\nPageRank Scores:&amp;quot;)
for page, score in pagerank_scores.items():
    print(f&amp;quot;{page}: {score}&amp;quot;)
```

Output:

Random Walk:

page3 -> page2 -> page4 -> page4 -> page3 -> page2 -> page1 -> page3 ->page2 -> page4 -> page2 -> page4 -> page2 -> page1 -> page2 -> page4 ->page4 -> page1 -> page3 -> page2 -> page4 -> page1 -> page3 -> page2 -> page4 -> page2 -> page1 -> page3 -> page2 -> page4 -> page2 -> page1 ->page3 -> page2 -> page4 -> page3 -> page2 -> page1 -> page3 -> page2 -> page4 -> page2 -> page4 -> page4 -> page1 -> page2 -> page1 -> page2 -> page4 -> page4 -> page4 -> page2 -> page1 -> page2 -> page4 -> page3 ->page2 -> page1 -> page2 -> page1 -> page3 -> page2 -> page4 -> page2 ->page4 -> page2 -> page4 -> page3 -> page2 -> page1 -> page3 -> page2 -> page1 -> page3 -> page2 -> page1 -> page3 -> page2 -> page1 -> page3 -> page2 -> page4 -> page4 -> page2 -> page4 -> page3 -> page2 -> page1 ->page3 -> page2 -> page4 -> page2 -> page4 -> page3 -> page2 -> page1 -> page2 -> page4 -> page3 -> page2 -> page4

PageRank Scores:

page1: 0.18
page2: 0.36
page3: 0.2
page4: 0.27

page1 has a PageRank score of 0.18.
page2 has a PageRank score of 0.36, making it the page with the highest score.
page3 has a PageRank score of 0.2.
page4 has a PageRank score of 0.27.

In the PageRank scores without random walk:
page1 has a score of 0.1006
page2 has a score of 0.1485
page3 has a score of 0.0803

page4 has a score of 0.1006
These scores are calculated using the built-in PageRank algorithm, which considers the link structure of the pages. The scores represent the probability of a random surfer reaching each page after a large number of iterations.
In the PageRank scores with random walk:
page1 has a score of 0.24
page2 has a score of 0.35
page3 has a score of 0.17
page4 has a score of 0.25

These scores are calculated based on the number of visits to each page in the random walk sequence. The scores represent the proportion of times each page was visited during the random walk. The difference between the two sets of scores is due to the random nature of the walk. The random walk introduces a stochastic element where the next page to visit is chosen randomly from the outgoing links of the current page. This randomness can lead to variations in the PageRank scores. However, the overall ranking order of the pages remains similar, with page2 having the highest score and page3 having the lowest score in both cases.

Practical No. 2

Aim: Perform spam classifier

Code:

```
#Perform Spam Classifier.
import matplotlib.pyplot as plt
import csv
import sklearn
import pickle
from wordcloud import WordCloud
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import
GridSearchCV,train_test_split,StratifiedKFold,cross_val_score,learning_curve
data=pd.read_csv("/content/sample_data/spam.csv",encoding="ISO-8859-1")
data.head()
data= data.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
data= data.rename(columns={"v2" : "text", "v1":"label"})
data[1990:2000]
data['label'].value_counts()

# Import nltk packages and Punkt Tokenizer Models
import nltk
nltk.download("punkt")
import warnings
warnings.filterwarnings('ignore')
ham_words = ''
spam_words = ''
# Creating a corpus of spam messages
for val in data[data['label'] == 'spam'].text:
  text = val.lower()
  tokens = nltk.word_tokenize(text)
  for words in tokens:
```

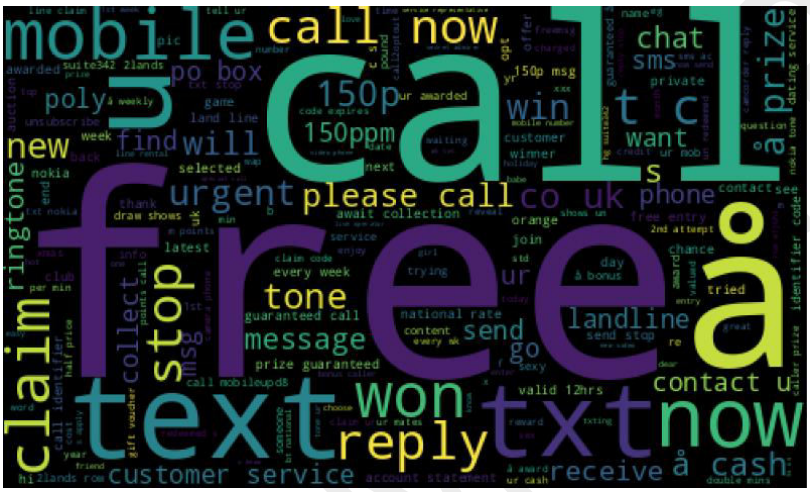
```
    spam_words = spam_words + words + ' '

# Creating a corpus of ham messages
for val in data[data['label'] == 'ham'].text:
  text = text.lower()
  tokens = nltk.word_tokenize(text)
  for words in tokens:
    ham_words = ham_words + words + ' '
spam_wordcloud = WordCloud(width=500, height=300).generate(spam_words)
ham_wordcloud = WordCloud(width=500, height=300).generate(ham_words)
#Spam Word cloud
plt.figure( figsize=(10,8), facecolor='w')
plt.imshow(spam_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
#Creating Ham wordcloud
plt.figure( figsize=(10,8), facecolor='g')
plt.imshow(ham_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
data = data.replace(['ham','spam'],[0, 1])
data.head(10)
import nltk
nltk.download('stopwords')

#remove the punctuations and stopwords
import string
def text_process(text):

  text = text.translate(str.maketrans('', '', string.punctuation))
  text = [word for word in text.split() if word.lower() not in stopwords.words('english')]

  return " ".join(text)

data['text'] = data['text'].apply(text_process)
data.head()
text = pd.DataFrame(data['text'])
label = pd.DataFrame(data['label'])

## Counting how many times a word appears in the dataset
```

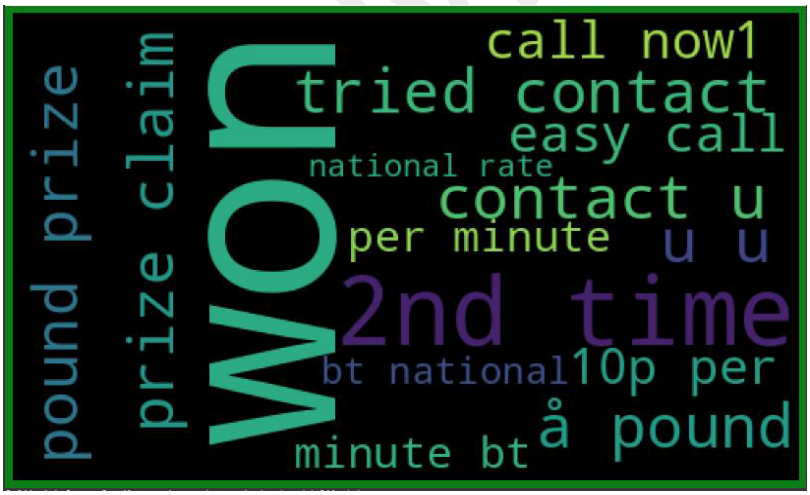
```
from collections import Counter

total_counts = Counter()
for i in range(len(text)):
   for word in text.values[i][0].split(" "):
      total_counts[word] += 1

print("Total words in data set: ", len(total_counts))

# Sorting in decreasing order (Word with highest frequency appears first)
vocab = sorted(total_counts, key=total_counts.get, reverse=True)
print(vocab[:60])

# Mapping from words to index

vocab_size = len(vocab)
word2idx = {}
#print vocab_size
for i, word in enumerate(vocab):
   word2idx[word] = i

# Text to Vector
def text_to_vector(text):
   word_vector = np.zeros(vocab_size)
   for word in text.split(" "):
      if word2idx.get(word) is None:
         continue
      else:
         word_vector[word2idx.get(word)] += 1
   return np.array(word_vector)
# Convert all titles to vectors
word_vectors = np.zeros((len(text), len(vocab)), dtype=np.int_)
for i, (_, text_) in enumerate(text.iterrows()):
   word_vectors[i] = text_to_vector(text_[0])

word_vectors.shape


# Convert all titles to vectors
word_vectors = np.zeros((len(text), len(vocab)), dtype=np.int_)
for i, (_, text_) in enumerate(text.iterrows()):
```

```
    word_vectors[i] = text_to_vector(text_[0])

word_vectors.shape

#convert the text data into vectors
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(data['text'])
vectors.shape

#features = word_vectors
features = vectors
#split the dataset into train and test set
X_train, X_test, y_train, y_test = train_test_split(features, data['label'], test_size=0.15,
random_state=111)

#import sklearn packages for building classifiers
from sklearn.linear_model import LogisticRegression

lrc = LogisticRegression(solver='liblinear', penalty='l1')
#create a dictionary of variables and models
clfs = {'LR': lrc}
#fit the data onto the models
def train(clf, features, targets):
    clf.fit(features, targets)

def predict(clf, features):
    return (clf.predict(features))

from sklearn.metrics import accuracy_score
pred_scores_word_vectors = []
for k,v in clfs.items():
    train(v, X_train, y_train)
    pred = predict(v, X_test)
    pred_scores_word_vectors.append((k, [accuracy_score(y_test , pred)]))

pred_scores_word_vectors

#write functions to detect if the message is spam or not
def find(x):
    if x == 1:
```

```
    print ("Message is SPAM")
  else:
    print ("Message is NOT Spam")


newtext = ["Free entry"]
integers = vectorizer.transform(newtext)

x = lrc.predict(integers)
find(x)
```

Output:





Total words in data set: 11305
['u', '2', 'call', 'U', 'get', 'Im', 'ur', '4', 'ltgt', 'know', 'go', 'like', 'dont', 'come', 'got', 'time', 'day', 'want', 'Ill', 'lor', 'Call', 'home', 'send', 'going', 'one', 'need', 'Ok', 'good', 'love', 'back', 'n', 'still', 'text', 'im', 'later', 'see', 'da', 'ok', 'think', 'ì', 'free', 'FREE', 'r', 'today', 'Sorry', 'week', 'phone', 'mobile', 'cant', 'tell', 'take', 'much', 'night', 'way', 'Hey', 'reply', 'work', 'make', 'give', 'new']
Message is NOT Spam

Practical No. 3

Aim: Demonstrate Text Mining and Webpage Pre-processing using meta information from the web pages (Local/Online)

Code:

```
from bs4 import BeautifulSoup
import requests
# Step 1: Fetch webpage content
url = "https://www.google.com"
response = requests.get(url)
html_content = response.content
# Step 2: Parse HTML and extract meta information
soup = BeautifulSoup(html_content, "html.parser")
meta_tags = soup.find_all("meta")

# Step 3: Pre-process meta information
meta_text = ""
for tag in meta_tags:
    if tag.get("content"):
        meta_text += tag.get("content") + " "

# Perform cleaning and pre-processing on meta_text

# Step 4: Perform text mining and analysis
# Example: Word frequency analysis
word_counts = {}
words = meta_text.split()
for word in words:
    if word not in word_counts:
        word_counts[word] = 1
    else:
        word_counts[word] += 1
# Step 5: Visualization
# Example: Print word frequency counts
for word, count in word_counts.items():
    print(f"{word}: {count}")
```

Output:

Search: 1
the: 1
world's: 1
information,: 1
including: 1
webpages,: 1
images,: 1
videos: 1
and: 1
more.: 1
Google: 1
has: 1
many: 1
special: 1
features: 1
to: 1
help: 1
you: 1
find: 1
exactly: 1
what: 1
you're: 1
looking: 1
for.: 1
noodp: 1
text/html;: 1
charset=UTF-8: 1
/logos/doodles/2023/juneteenth-2023-6753651837109890.2-l.png: 1
Juneteenth: 1
2023: 1
Celebrating: 2
Juneteenth!: 2
#GoogleDoodle: 2
summary_large_image: 1
@GoogleDoodles: 1
https://www.google.com/logos/doodles/2023/juneteenth-2023-6753651837109890-2x.png: 2
1150: 1
460: 1

Practical No. 4

Aim: Apriori Algorithm implementation in case study.

Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another. The apriori algorithm is an association rule learning that analyzes that people who bought product A also bought product B.

Code:

```
# Step 1: Import the libraries

import pandas as pd

from apyori import apriori

# Step 2: Load the dataset

df = pd.read_csv('transaction.csv', header=None)

# Step 3: Display statistics of records

print("Display statistics")

print("====================")

print(df.describe())

# Step 4: Display shape of the dataset

print("\nShape:",df.shape)

# Step 5: Convert dataframe into a nested list

database = []

for i in range(0,30):

database.append([str(df.values[i,j]) for j in range(0,6)])

# Step 6: Develop the Apriori model

arm_rules = apriori(database, min_support=0.5, min_confidence=0.7, min_lift=1.2)

arm_results = list((arm_rules))
```

```
# Step 7: Display the number of rule(s)

print("\nNo. of rule(s):",len(arm_results))

# Step 8: Display the rule(s)

print("\nResults: ")

print("========")

print(arm_results)
```

Output:

```
Display statistics:
==================
0 1 2 3 4 5
count 19 18 23 23 20 22
unique 1 1 1 1 1 1
top Juice Chips Bread Butter Milk Banana
freq 19 18 23 23 20 22
Shape: (30, 6)
No. of rule(s): 1
Results:
========
[RelationRecord(items=frozenset({>'Butter>', >'Bread>', >'Milk>'}), support=0.5,
ordered_statistics=[OrderedStatistic(items_base=frozenset({>'Bread>', >'Milk>'}),
items_add=frozenset({>'Butter>'}), confidence=0.9375, lift=1.2228260869565217)])]
```

The single rule displayed states that if a customer buys >'Bread>' and >'Milk>', there is a 93.75%
confidence that they will also buy >Butter>, with a lift value of 1.2228.

Support: The support of an itemset is the proportion of transactions in the dataset that contain that itemset. It indicates the frequency or popularity of the itemset.
- Example: If the support of {Milk, Bread} is 0.5, it means that 50% of the transactions in the dataset contain both Milk and Bread.

- Confidence: The confidence of an association rule measures the conditional probability of the consequent given the antecedent. It represents the strength of the rule.
- Example: If the confidence of {Milk} -> {Bread} is 0.8, it means that in 80% of the transactions where Milk is present, Bread is also present.

 Lift: Lift is a measure of the strength of association between the antecedent and the consequent in a rule, taking into account the underlying support of the items. It compares the observed support of the rule to what would be expected if the items were independent.
 Example: If the lift of {Milk} -> {Bread} is 1.2, it means that the likelihood of buying Milk and Bread together is 1.2 times higher than if the items were purchased independently.
 support indicates the frequency of an itemset, confidence measures the strength of an association rule, and lift quantifies the deviation from independence in the rule

Practical No. 5

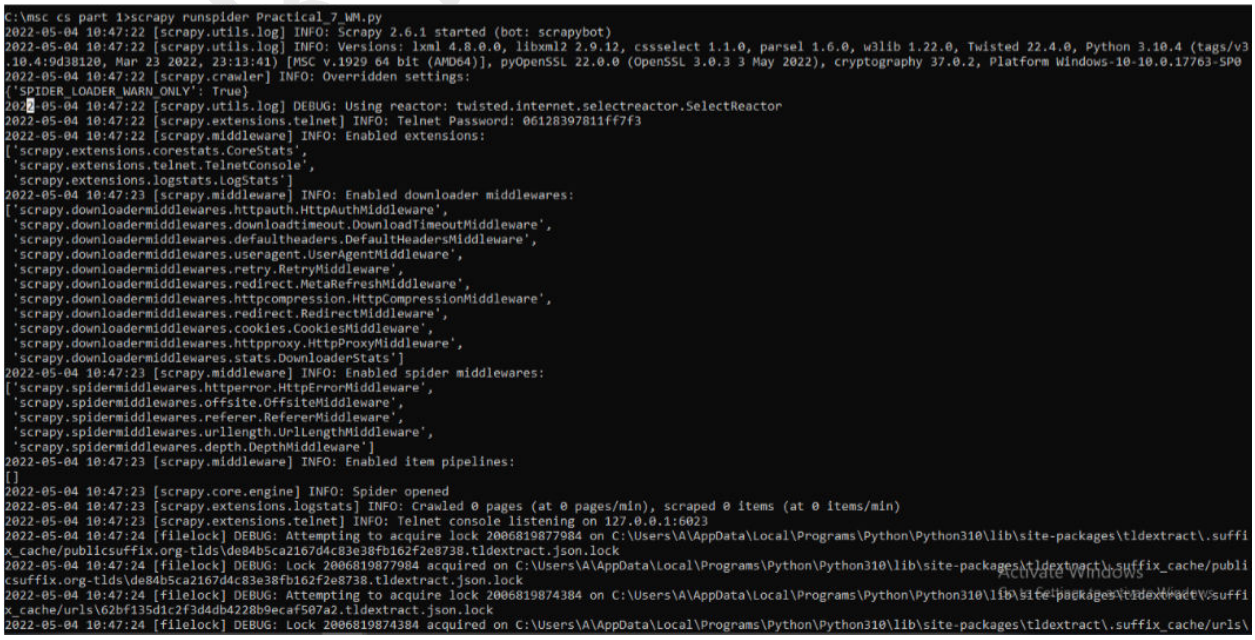
Aim: Develop a basic crawler for the web search for user defined keywords.

A crawler, also known as a web crawler or spider, is an automated program or script that systematically browses the internet to discover and retrieve web pages. Crawlers are commonly used by search engines to index web pages for search results.
A basic crawler is a general-purpose web crawler that systematically explores the web, starting from a seed URL and following links to discover new pages. For example, search engines like Google use basic crawlers to index and retrieve web pages for search results.
A focused crawler, on the other hand, is designed to crawl specific areas or topics of interest on the web. It prioritizes relevant domains, directories, or pages related to a particular theme. For example, a news aggregator website may use a focused crawler to crawl news websites and collect specific articles or news updates for its platform.

Code:

```
import scrapy
class spider1(scrapy.Spider):
name = 'Wikipedia';
start_urls = ['https://en.wikipedia.org/wiki/Battery_(electricity)']
def parse(self, response):
    pass
```

Output:



Practical No. 6

Aim: Develop a focused crawler for local search.

Code:

```
Link:-https://www.topcoder.com/thrive/articles/web-crawler-in-python
import requests
import lxml
from bs4 import BeautifulSoup
url = &amp;quot;https://www.rottentomatoes.com/top/bestofrt/&amp;quot;
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36 QIHU
360SE'}
f = requests.get(url, headers = headers)
movies_lst = []
soup = BeautifulSoup(f.content, 'lxml')
movies = soup.find('table', {'class': 'table'}).find_all('a')
num = 0
for anchor in movies:
urls = 'https://www.rottentomatoes.com' + anchor['href']
movies_lst.append(urls)
num += 1
movie_url = urls
movie_f = requests.get(movie_url, headers = headers)
movie_soup = BeautifulSoup(movie_f.content, 'lxml')
movie_content = movie_soup.find('div', {'class': 'movie_synopsis clamp clamp-6 js-
clamp'})
print(num, urls, '\n', 'Movie:' + anchor.string.strip())
print('Movie info:' + movie_content.string.strip())
```

Output:

```
========== RESTART: C:/msc cs part 1/Practical_8_WM.py ==========
1 https://www.rottentomatoes.com/m/the_battle_of_algiers
 Movie:The Battle of Algiers (La Battaglia di Algeri) (1967)
Movie info:Paratrooper commander Colonel Mathieu (Jean Martin), a former French Resistance fighter during World War II, is sent to 1950s Algeria to re
inforce efforts to squelch the uprisings of the Algerian War. There he faces Ali la Pointe (Brahim Haggiag), a former petty criminal who, as the leader of t
he Algerian Front de Liberation Nationale, directs terror strategies against the colonial French government occupation. As each side resorts to ever-incre
asing brutality, no violent act is too unthinkable.
```

Practical No. 7

Aim: Develop a program for deep search implementation to detect plagiarism in documents online

Code:

```
# from difflib module
from difflib import SequenceMatcher

# Declaring string variables
string1 = 'I am geek'
string2 = 'I am geeks'

# Using the SequenceMatcher()
match = SequenceMatcher(None,
                                                    string1, string2)

# convert above output into ratio
# and multiplying it with 100
result = match.ratio() * 100

# Display the final result
print(int(result), "%")

import pandas as pd

# Upload the text files
uploaded1 = files.upload()
uploaded2 = files.upload()

# Read the uploaded files
file1 = uploaded1['1.txt'].decode('utf-8')
file2 = uploaded2['2.txt'].decode('utf-8')

# Comparing the contents of the files
similarity_ratio = SequenceMatcher(None, file1, file2).ratio()

# Converting the decimal ratio to percentage
plagiarism_percentage = int(similarity_ratio * 100)
```

```
# Display the result
print(f"{plagiarism_percentage}% Plagiarized Content")
```

Output:

```
94 %
Choose Files 1.txt
• 1.txt(text/plain) - 0 bytes, last modified: 6/20/2023 - 100% done
Saving 1.txt to 1 (4).txt
Choose Files 2.txt
• 2.txt(text/plain) - 0 bytes, last modified: 6/20/2023 - 100% done
Saving 2.txt to 2 (4).txt
100% Plagiarized Content
```

Practical No. 8

Aim: Sentiment analysis for reviews by customers and visualize the same

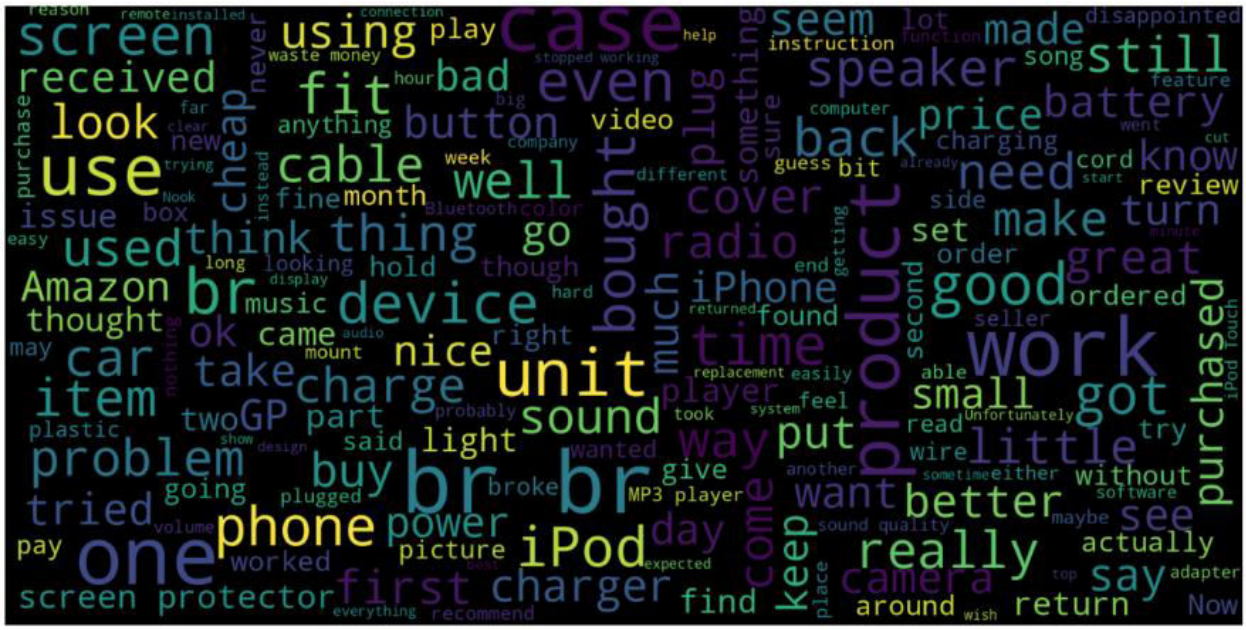
Code:

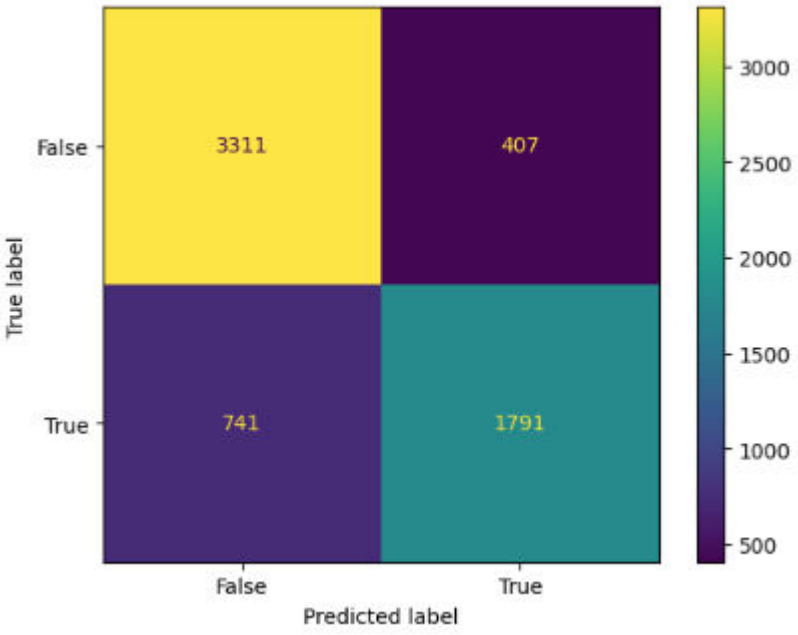
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
from wordcloud import WordCloud
data=pd.read_csv("/content/sample_data/AmazonReview.csv")
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords

data.head()
data.info()
data.dropna(inplace=True)
#1,2,3->negative(i.e 0)
data.loc[data['Sentiment']<=3,'Sentiment'] = 0

#4,5->positive(i.e 1)
data.loc[data['Sentiment']>3,'Sentiment'] = 1
stp_words=stopwords.words('english')
def clean_review(review):
  cleanreview=" ".join(word for word in review.
                                        split() if word not in stp_words)
  return cleanreview

data['Review']=data['Review'].apply(clean_review)
data.head()
data['Sentiment'].value_counts()
consolidated=' '.join(word for word in data['Review'][data['Sentiment']==0].astype(str))
wordCloud=WordCloud(width=1600,height=800,random_state=21,max_font_size=110)
plt.figure(figsize=(15,10))
plt.imshow(wordCloud.generate(consolidated),interpolation='bilinear')
plt.axis('off')
plt.show()
cv = TfidfVectorizer(max_features=2500)
```

```
X = cv.fit_transform(data['Review'] ).toarray()
from sklearn.model_selection import train_test_split
x_train ,x_test,y_train,y_test=train_test_split(X,data['Sentiment'],

        test_size=0.25 ,

        random_state=42)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix

model=LogisticRegression()

#Model fitting
model.fit(x_train,y_train)

#testing the model
pred=model.predict(x_test)

#model accuracy
print(accuracy_score(y_test,pred))

from sklearn import metrics
cm = confusion_matrix(y_test,pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm,
                                                                    display_labels
= [False, True])

cm_display.plot()
plt.show()
```

Output:

```
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 2 columns):
 #   Column     Non-Null   Count     Dtype
---  ------         -----------     -----        -----
 0   Review     24999      non-null     object
 1   Sentiment  25000      non-null     int64
```


screen
using
play
case
seem
made
disappointed
still
received
fit
bad
even
speaker
battery
look
button
video
back
price
use
cable
Well
cover
need
know
review
issue
think
thing
go
radio
make
turn
used
bought
product
good
great
Amazon
br
device
iPhone
work
thought
ok
came
nice
unit
time
purchased
car
take
charge
item
sound
player
small
got
problem
buy
br
br
way
put
little
tried
phone
power
iPod
day
want
better
see
one
really
actually
screen protector
first
charger
find
keep
camera
say
around
return
Now


3311
407
741
1791
False
True
True label
False
True
Predicted label
3000
2500
2000
1500
1000
500