Harshit Mishra (19BCE0799)

CSE-3024 WEB MINING LAB ASSIGNMENT 3

Aim: To write a python program to find important words from the text using TF-IDF. Use a minimum of 5 documents with the real text source from a web page of some relevance.

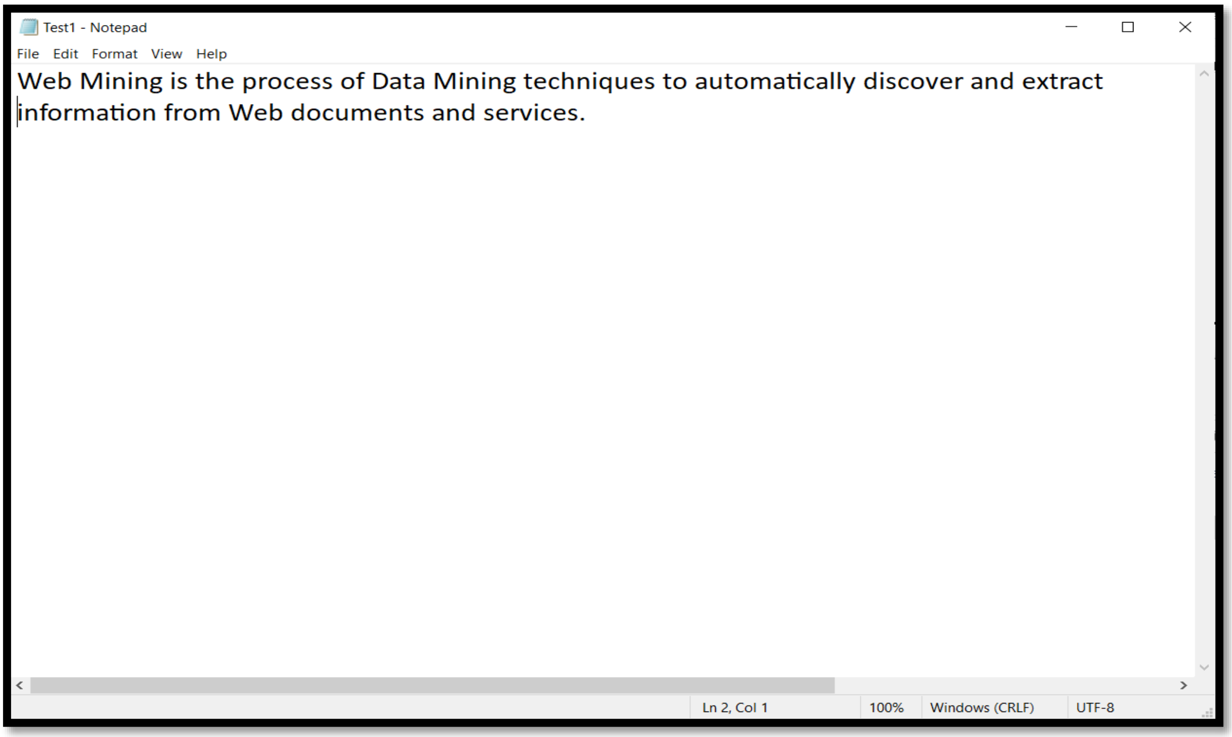
Procedure:

- Firstly, we import our libraries that would help us in doing this term frequency count.
- Next, we declare and write tf, idf, n_containing and tf_idf functions that will help assist the return values and make code more readable.
- Then we create 5 documents and read them in our workspace.
- We then make the bloblist that contains all the documents in list format. And then we print the counts of top 3 words in each document.
- Then we calculate the cosine similarity using inbuilt cosine_similarity matrix.
- For the above we need to create a pandas data frame of count vectors.

Harshit Mishra (19BCE0799)

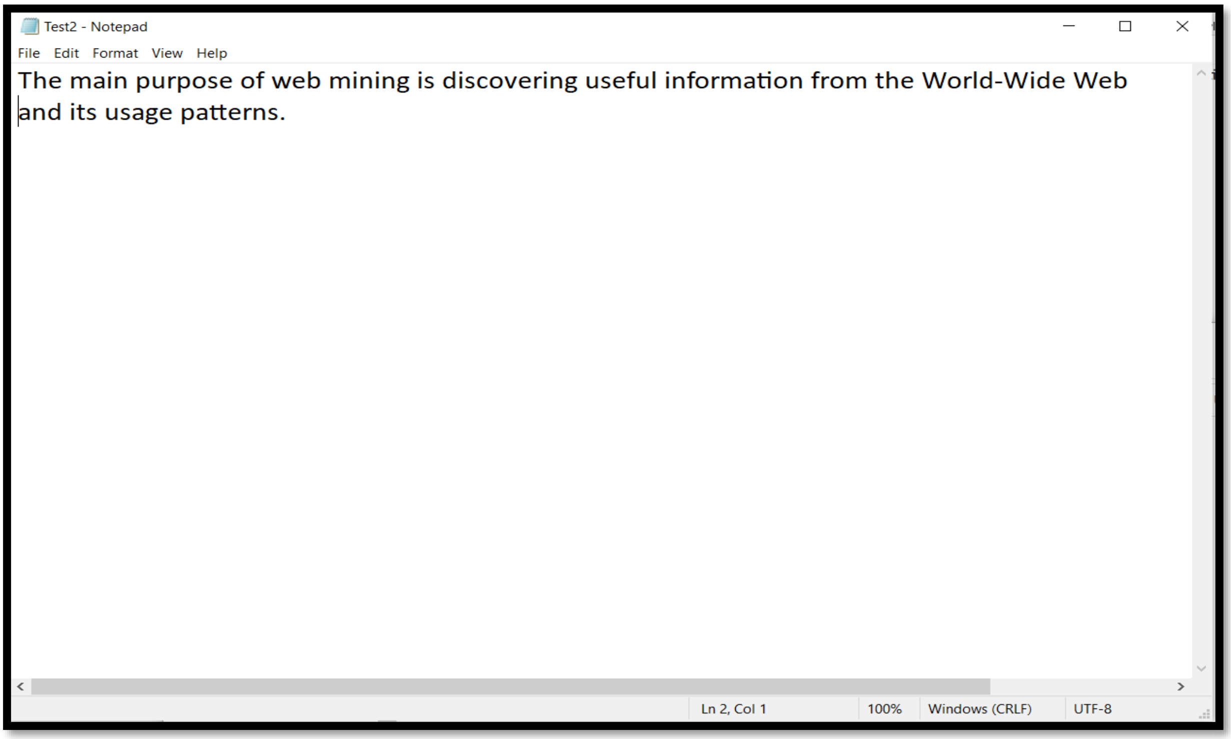
Input Documents:

Test1.txt:



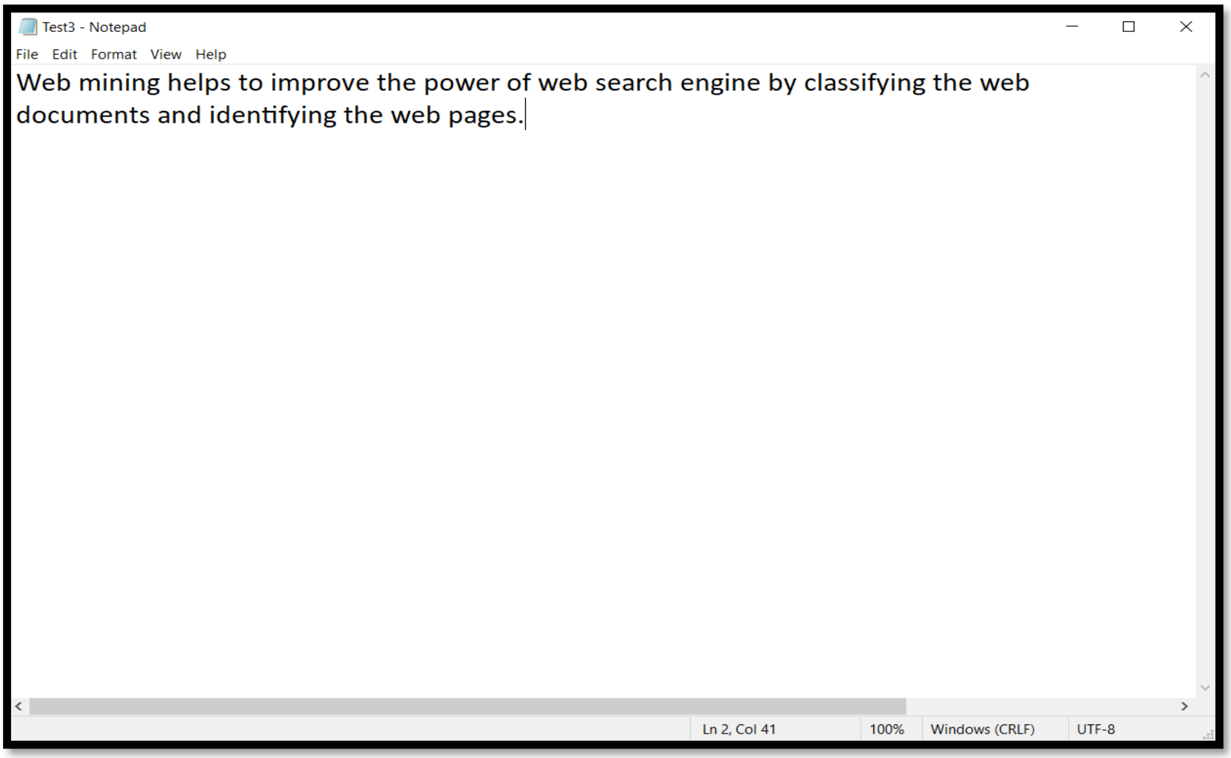
Web Mining is the process of Data Mining techniques to automatically discover and extract information from Web documents and services.

Test2.txt:



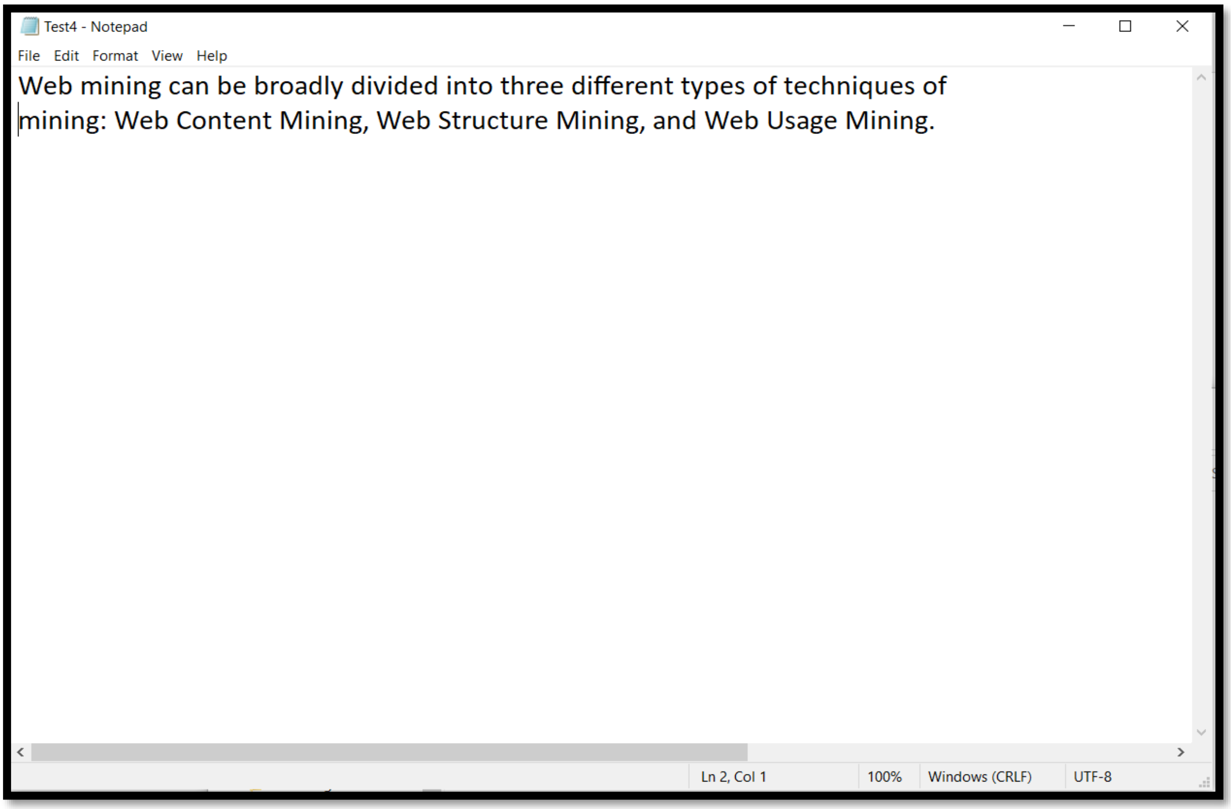
The main purpose of web mining is discovering useful information from the World-Wide Web and its usage patterns.

Test3.txt:



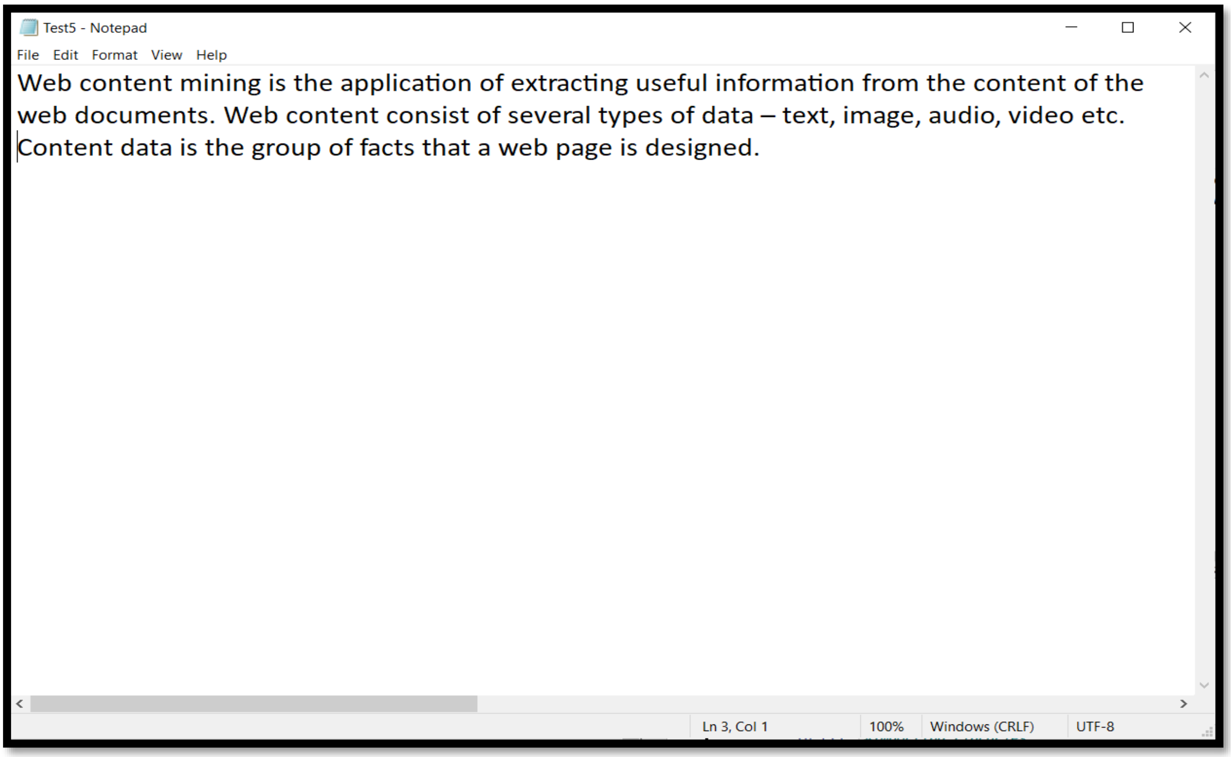
Web mining helps to improve the power of web search engine by classifying the web documents and identifying the web pages.

Test4.txt:



Web mining can be broadly divided into three different types of techniques of mining: Web Content Mining, Web Structure Mining, and Web Usage Mining.

Test5.txt:



Web content mining is the application of extracting useful information from the content of the web documents. Web content consist of several types of data – text, image, audio, video etc.
Content data is the group of facts that a web page is designed.

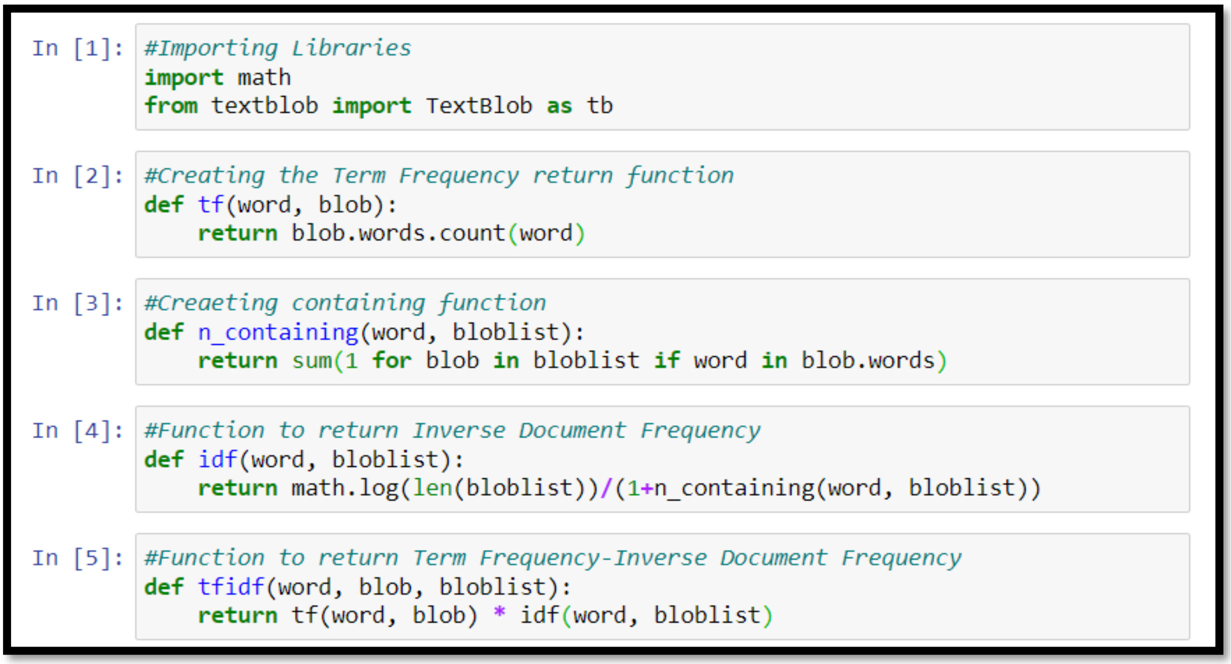
Harshit Mishra (19BCE0799)

Code:

```
#Importing Libraries
import math
from textblob import TextBlob as tb


#Creating the Term Frequency return function
def tf(word, blob):
    return blob.words.count(word)


#Creaeting containing function
def n_containing(word, bloblist):
    return sum(1 for blob in bloblist if word in blob.words)


#Function to return Inverse Document Frequency
def idf(word, bloblist):
    return math.log(len(bloblist))/(1+n_containing(word, bloblist))


#Function to return Term Frequency-Inverse Document Frequency
def tfidf(word, blob, bloblist):
    return tf(word, blob) * idf(word, bloblist)


#Reading First Input File
with open('Test1.txt') as a:
    test1 = (a.read())
document1 = tb(test1)


#Reading Second Input File
with open('Test2.txt') as a:
    test2 = (a.read())
document2 = tb(test2)


#Reading Third Input File
with open('Test3.txt') as a:
    test3 = (a.read())
document3 = tb(test3)

#Reading Fourth Input File
with open('Test4.txt') as a:
    test4 = (a.read())
document4 = tb(test4)
```

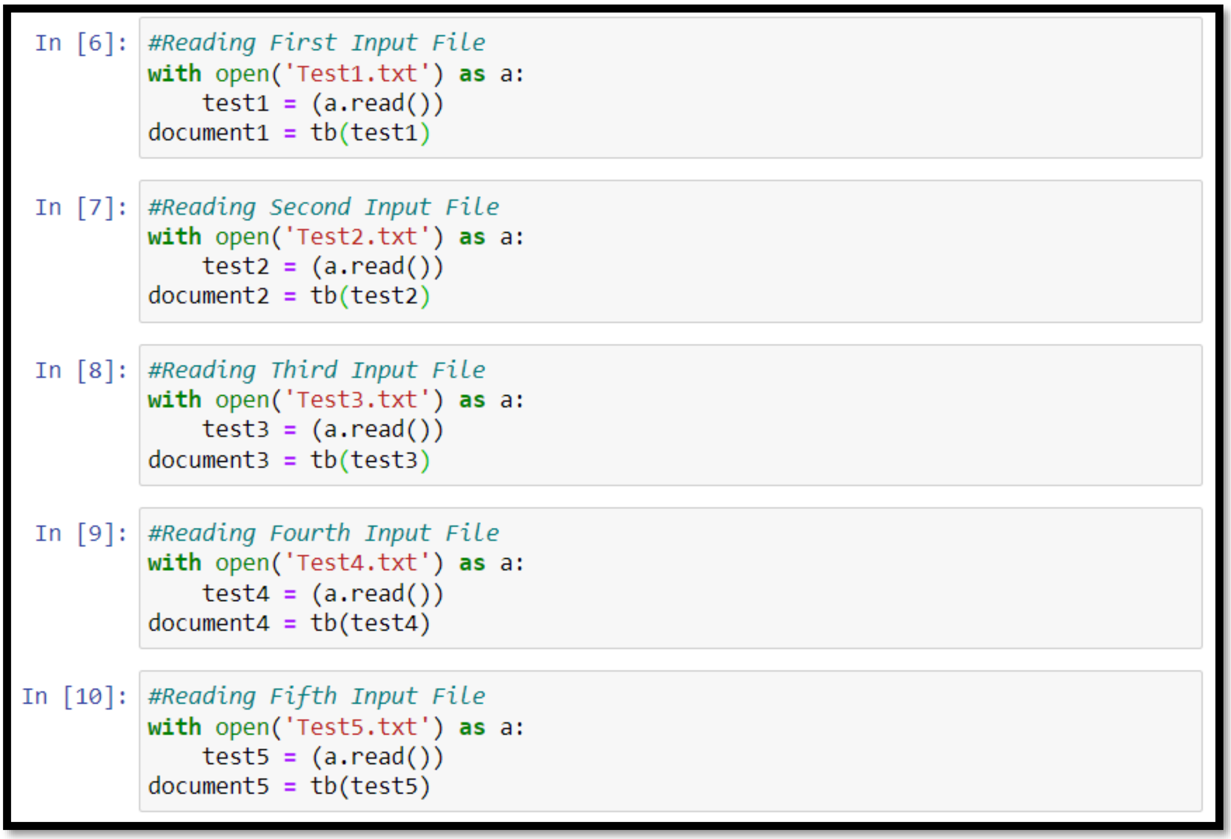
```
#Reading Fifth Input File
with open('Test5.txt') as a:
    test5 = (a.read())
document5 = tb(test5)


#Printing the top three words in each document
bloblist = [document1, document2, document3, document4, document5]
for i, blob in enumerate(bloblist):
    print("Top words in document {}". format(i+1))
    scores = {word: tfidf(word, blob,bloblist) for word in blob.words}
    sorted_words = sorted(scores.items(), key=lambda x:x[1], reverse=True)
    for word, score in sorted_words[:3]:
        print("\tWord: {}, TF-IDF: {}".format(word, round(score, 5)))


#Calculating Cosine Similarity
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
documents = [test1, test2, test3, test4, test5]


#Creating the Document Term Matrix
count_vectorizer = CountVectorizer()
sparse_matrix = count_vectorizer.fit_transform(documents)

#Creating a dataframe to store each count_vectorizer
doc_term_matrix = sparse_matrix.todense()
df = pd.DataFrame(doc_term_matrix,
                columns=count_vectorizer.get_feature_names(),
                index=['test1', 'test2','test3', 'test4', 'test5'])
df


#Printing the Cosine Similarity
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(df, df))
```

Code Snippet and Outputs:



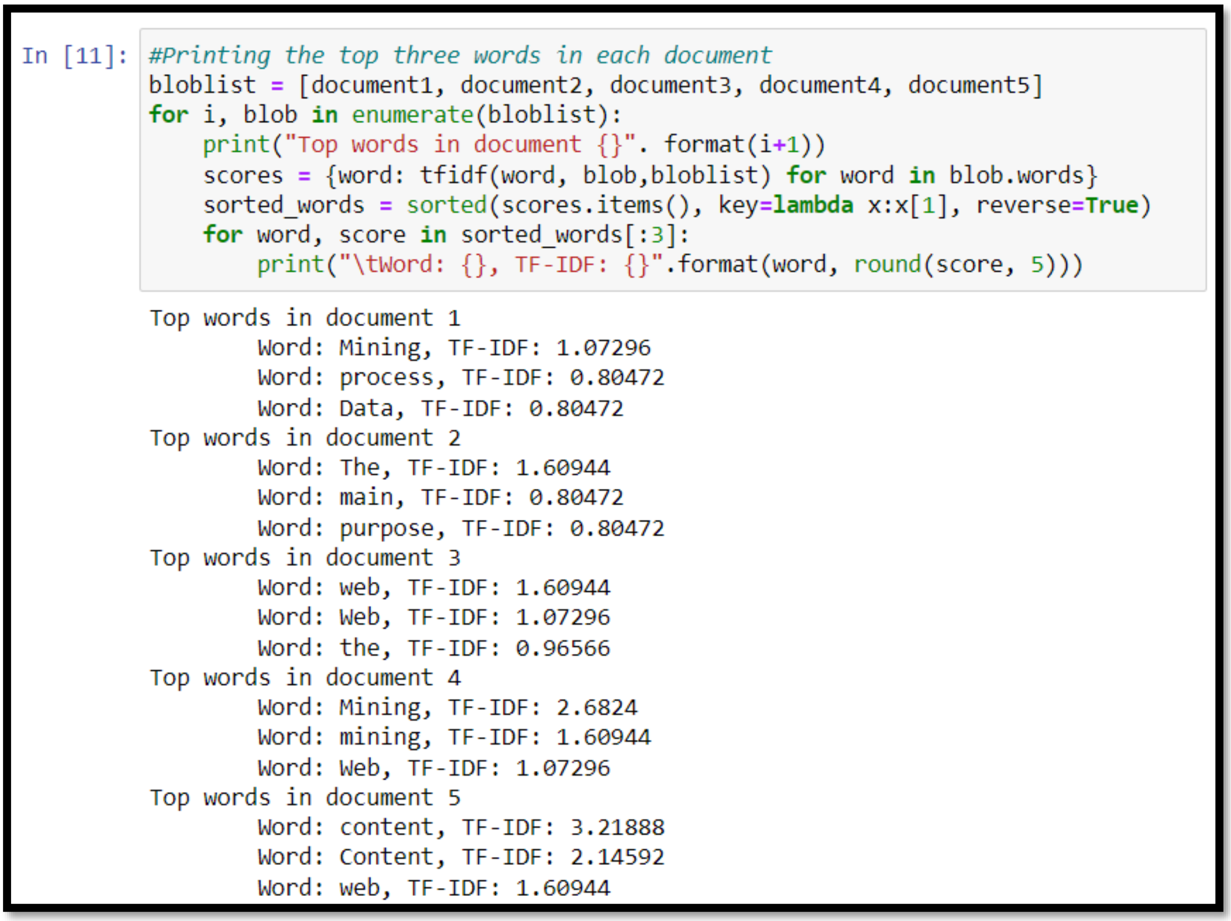
```
In [1]: #Importing Libraries
        import math
        from textblob import TextBlob as tb

In [2]: #Creating the Term Frequency return function
        def tf(word, blob):
            return blob.words.count(word)

In [3]: #Creaeting containing function
        def n_containing(word, bloblist):
            return sum(1 for blob in bloblist if word in blob.words)

In [4]: #Function to return Inverse Document Frequency
        def idf(word, bloblist):
            return math.log(len(bloblist))/(1+n_containing(word, bloblist))

In [5]: #Function to return Term Frequency-Inverse Document Frequency
        def tfidf(word, blob, bloblist):
            return tf(word, blob) * idf(word, bloblist)
```

Here we are declaring and returning the values of our reusable functions.



```
In [6]:  #Reading First Input File
         with open('Test1.txt') as a:
             test1 = (a.read())
         document1 = tb(test1)

In [7]:  #Reading Second Input File
         with open('Test2.txt') as a:
             test2 = (a.read())
         document2 = tb(test2)

In [8]:  #Reading Third Input File
         with open('Test3.txt') as a:
             test3 = (a.read())
         document3 = tb(test3)

In [9]:  #Reading Fourth Input File
         with open('Test4.txt') as a:
             test4 = (a.read())
         document4 = tb(test4)

In [10]: #Reading Fifth Input File
         with open('Test5.txt') as a:
             test5 = (a.read())
         document5 = tb(test5)
```

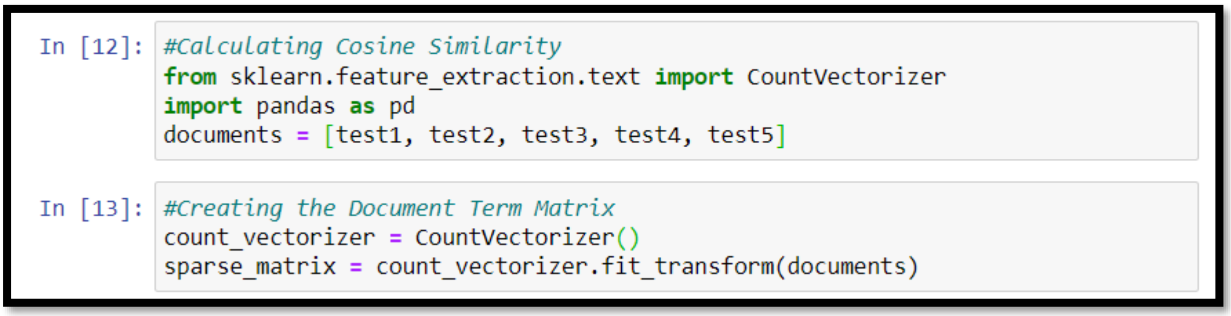
Here we have imported the 5 text files into our workspace. These text files are named as Text1, Text2, Text3, Text4 and Text5.



```
In [11]: #Printing the top three words in each document
         bloblist = [document1, document2, document3, document4, document5]
         for i, blob in enumerate(bloblist):
             print("Top words in document {}". format(i+1))
             scores = {word: tfidf(word, blob,bloblist) for word in blob.words}
             sorted_words = sorted(scores.items(), key=lambda x:x[1], reverse=True)
             for word, score in sorted_words[:3]:
                 print("\tWord: {}, TF-IDF: {}".format(word, round(score, 5)))
```

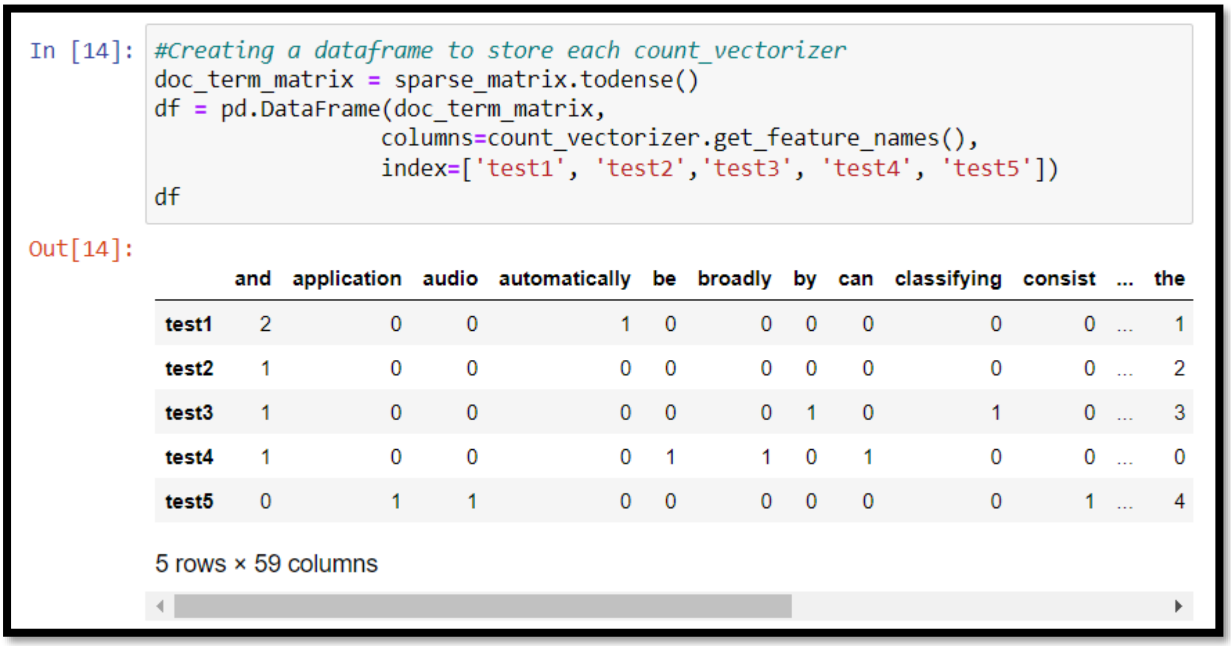
```
Top words in document 1
        Word: Mining, TF-IDF: 1.07296
        Word: process, TF-IDF: 0.80472
        Word: Data, TF-IDF: 0.80472
Top words in document 2
        Word: The, TF-IDF: 1.60944
        Word: main, TF-IDF: 0.80472
        Word: purpose, TF-IDF: 0.80472
Top words in document 3
        Word: web, TF-IDF: 1.60944
        Word: Web, TF-IDF: 1.07296
        Word: the, TF-IDF: 0.96566
Top words in document 4
        Word: Mining, TF-IDF: 2.6824
        Word: mining, TF-IDF: 1.60944
        Word: Web, TF-IDF: 1.07296
Top words in document 5
        Word: content, TF-IDF: 3.21888
        Word: Content, TF-IDF: 2.14592
        Word: web, TF-IDF: 1.60944
```

Here we have printed the top words in each document. We have printed only top 3 words and their TF-IDF values in the same line along with the word/term.



```
In [12]: #Calculating Cosine Similarity
         from sklearn.feature_extraction.text import CountVectorizer
         import pandas as pd
         documents = [test1, test2, test3, test4, test5]

In [13]: #Creating the Document Term Matrix
         count_vectorizer = CountVectorizer()
         sparse_matrix = count_vectorizer.fit_transform(documents)
```

Here we have created the count vector that contains the frequency of each word of each document. This is done in order for Cosine Similarity.



In [14]:

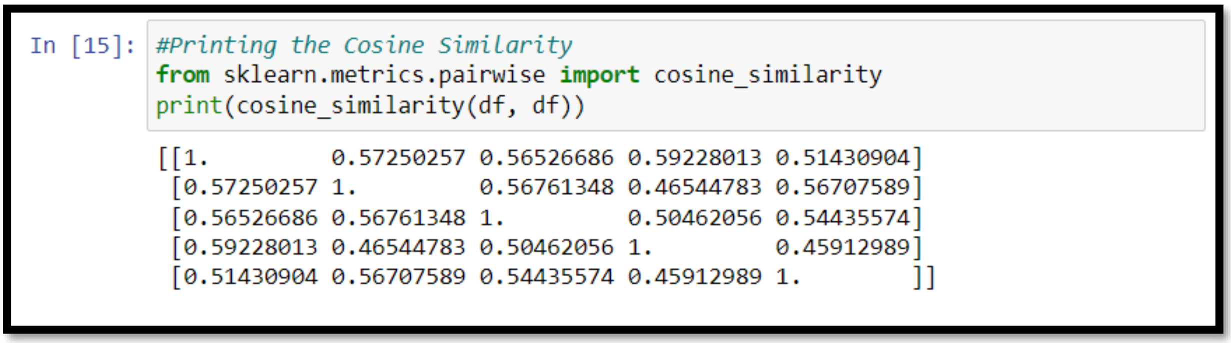
```
#Creating a dataframe to store each count_vectorizer
doc_term_matrix = sparse_matrix.todense()
df = pd.DataFrame(doc_term_matrix,
                  columns=count_vectorizer.get_feature_names(),
                  index=['test1', 'test2','test3', 'test4', 'test5'])
df
```

Out[14]:

	and	application	audio	automatically	be	broadly	by	can	classifying	consist	...	the
test1	2	0	0	1	0	0	0	0	0	0	...	1
test2	1	0	0	0	0	0	0	0	0	0	...	2
test3	1	0	0	0	0	0	1	0	1	0	...	3
test4	1	0	0	0	1	1	0	1	0	0	...	0
test5	0	1	1	0	0	0	0	0	0	1	...	4

5 rows × 59 columns

Here we have combined the count vectors of each document into Pandas Data Frame.



In [15]:

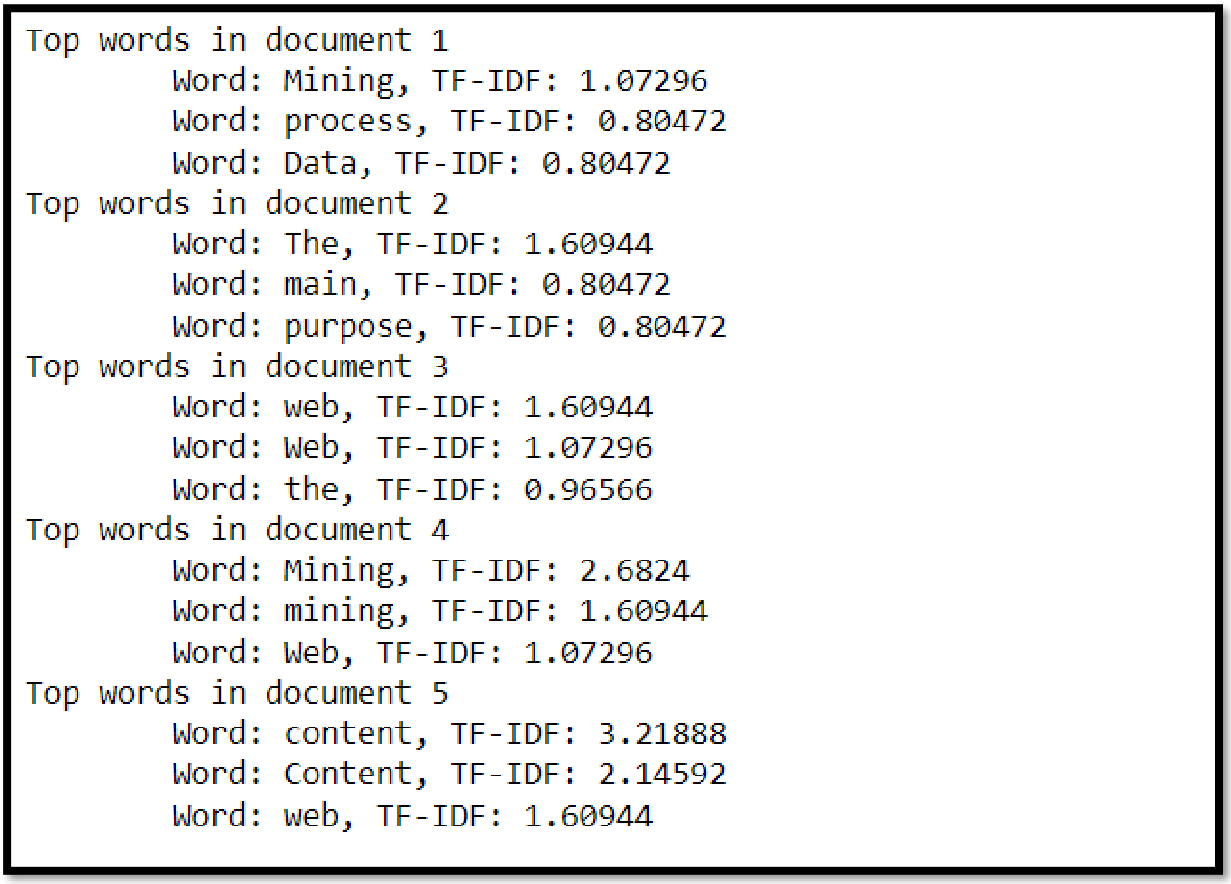
```
#Printing the Cosine Similarity
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(df, df))
```

```
[[1.         0.57250257 0.56526686 0.59228013 0.51430904]
 [0.57250257 1.         0.56761348 0.46544783 0.56707589]
 [0.56526686 0.56761348 1.         0.50462056 0.54435574]
 [0.59228013 0.46544783 0.50462056 1.         0.45912989]
 [0.51430904 0.56707589 0.54435574 0.45912989 1.        ]]
```

Here we have printed the cosine similarity of each document.

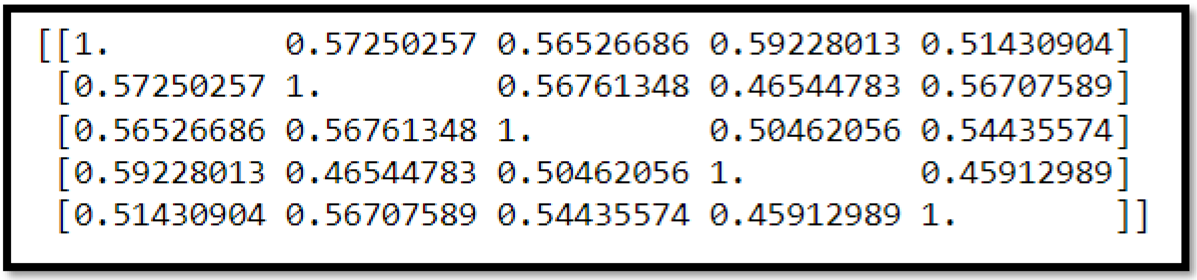
Results:

Top Words in each document:



```
Top words in document 1
        Word: Mining, TF-IDF: 1.07296
        Word: process, TF-IDF: 0.80472
        Word: Data, TF-IDF: 0.80472
Top words in document 2
        Word: The, TF-IDF: 1.60944
        Word: main, TF-IDF: 0.80472
        Word: purpose, TF-IDF: 0.80472
Top words in document 3
        Word: web, TF-IDF: 1.60944
        Word: Web, TF-IDF: 1.07296
        Word: the, TF-IDF: 0.96566
Top words in document 4
        Word: Mining, TF-IDF: 2.6824
        Word: mining, TF-IDF: 1.60944
        Word: Web, TF-IDF: 1.07296
Top words in document 5
        Word: content, TF-IDF: 3.21888
        Word: Content, TF-IDF: 2.14592
        Word: web, TF-IDF: 1.60944
```

Cosine Similarity:



```
[[1.         0.57250257 0.56526686 0.59228013 0.51430904]
 [0.57250257 1.         0.56761348 0.46544783 0.56707589]
 [0.56526686 0.56761348 1.         0.50462056 0.54435574]
 [0.59228013 0.46544783 0.50462056 1.         0.45912989]
 [0.51430904 0.56707589 0.54435574 0.45912989 1.        ]]
```