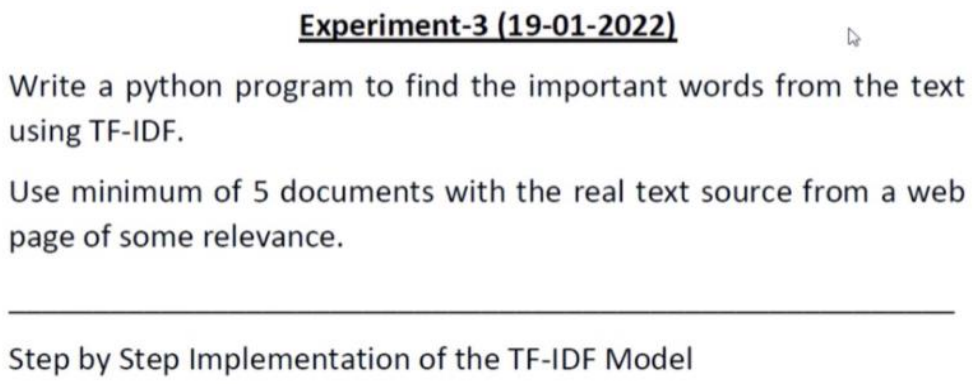
CSE-3024 Web Mining

Lab Assignment 2

Alokam Nikhitha

19BCE2555

Question



Experiment-3 (19-01-2022)

Write a python program to find the important words from the text using TF-IDF.

Use minimum of 5 documents with the real text source from a web page of some relevance.

Step by Step Implementation of the TF-IDF Model

Problem statement:

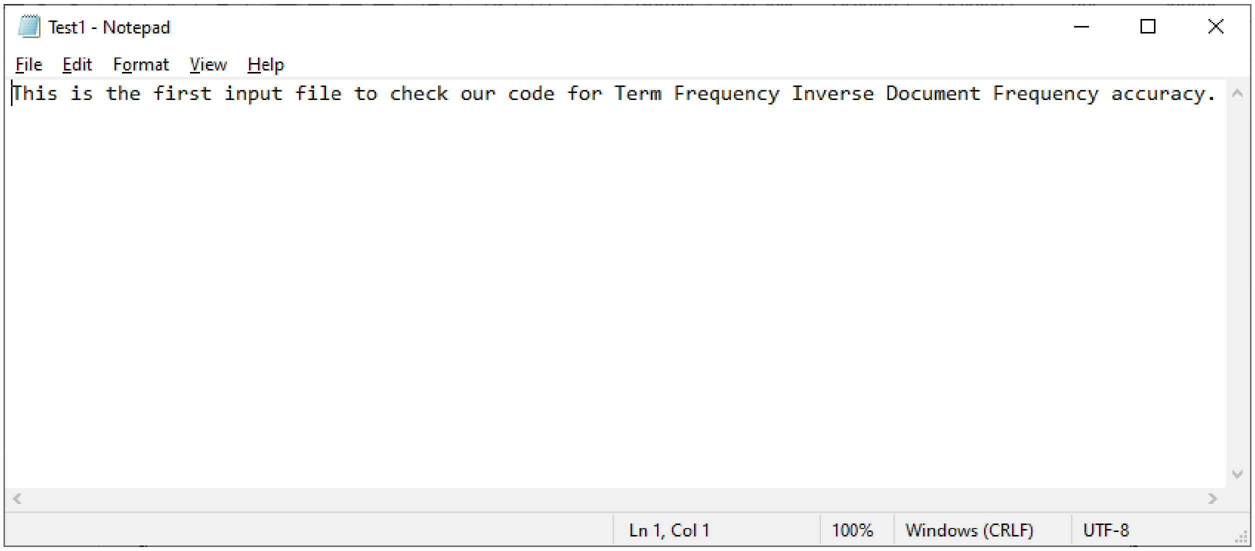
Python program to find the important words from the textfile using TF-IDF using atleast minimum of 5 documents

Procedure:

- **We will Firstly import our libraries Which are required in doing the term frequency count.**
- **Later, we will declare and define tf, idf, n_containing and tf_idf functions that will help assist the return values and make code more readable.**
- **We will create 5 Text File inputs and read them in our workspace.**
- **Later, We will make the bloblist that contains all the Text File Inputs in list format. And then we will print the counts of top 3 words in every document.**
- **We will then calculate the cosine similarity using inbuilt cosine_similarity matrix.**
- **For the above we need to create a pandas data frame of count vectors.**

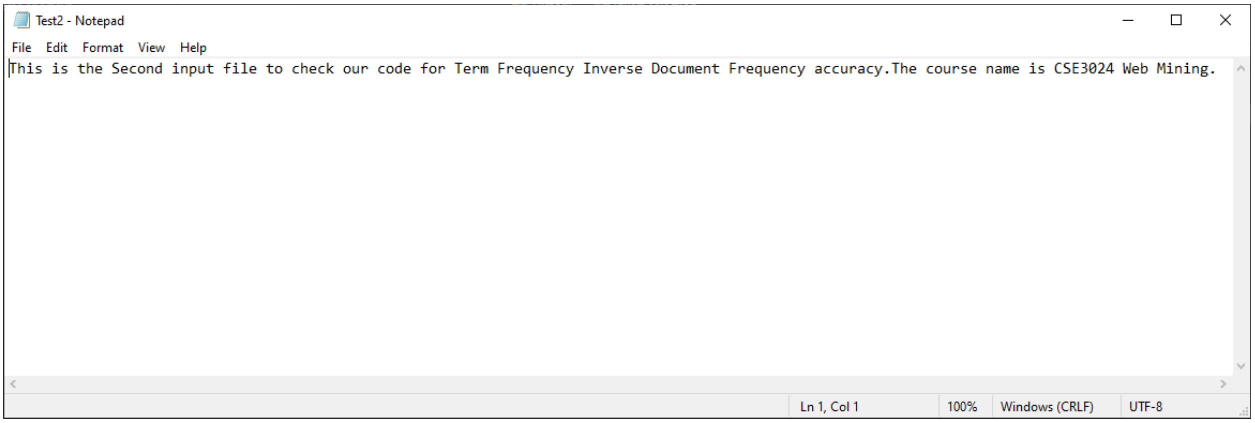
Text File Taken as Input:

Text File 1:



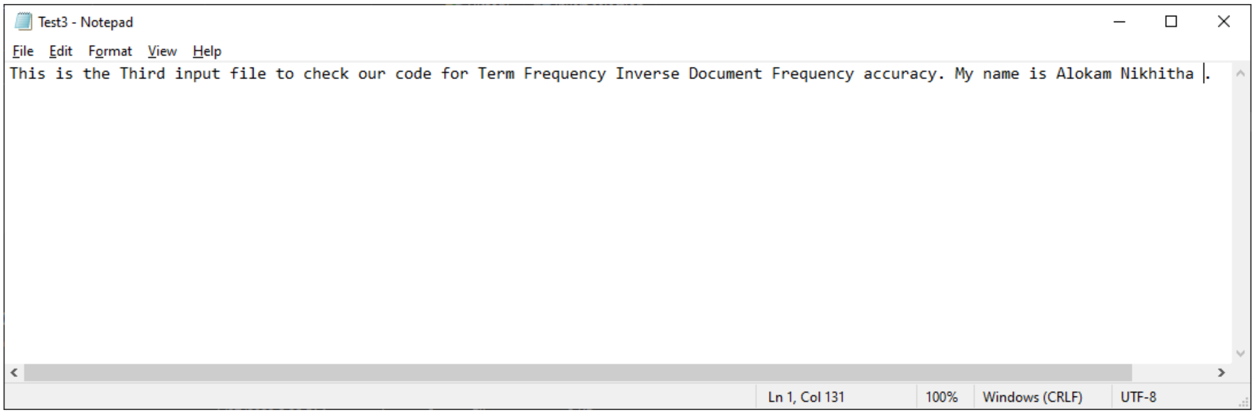
```
This is the first input file to check our code for Term Frequency Inverse Document Frequency accuracy.
```

Text File 2:



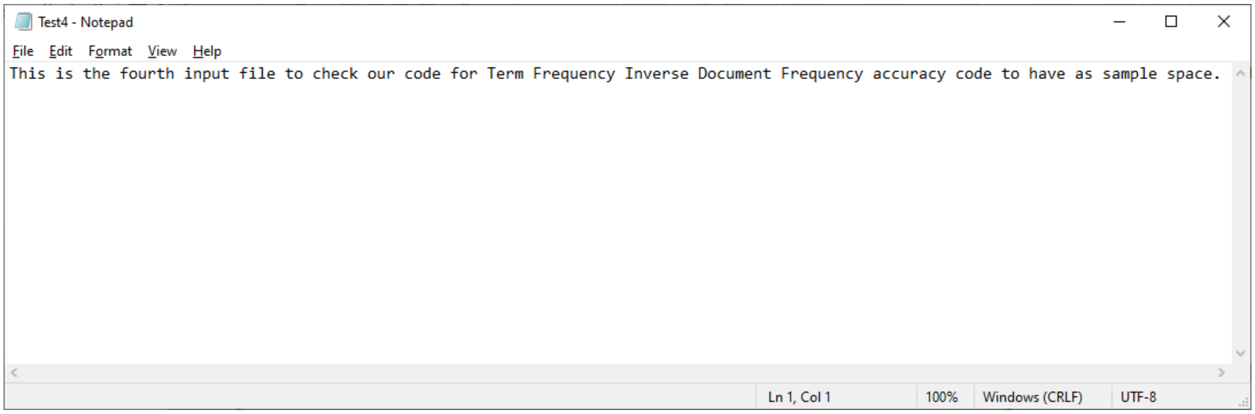
```
This is the Second input file to check our code for Term Frequency Inverse Document Frequency accuracy.The course name is CSE3024 Web Mining.
```

Text File 3:



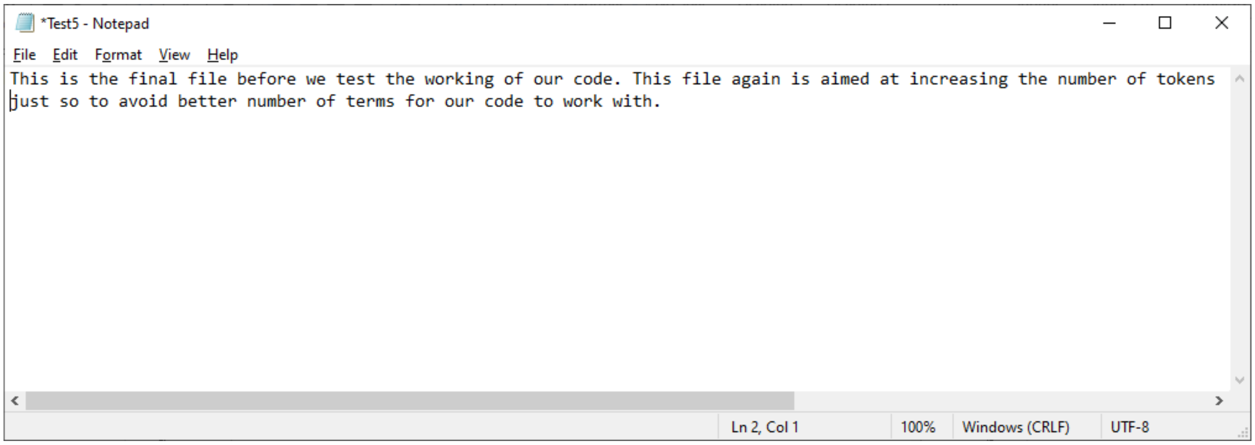
```
This is the Third input file to check our code for Term Frequency Inverse Document Frequency accuracy. My name is Alokam Nikhitha .
```

Text File 4:



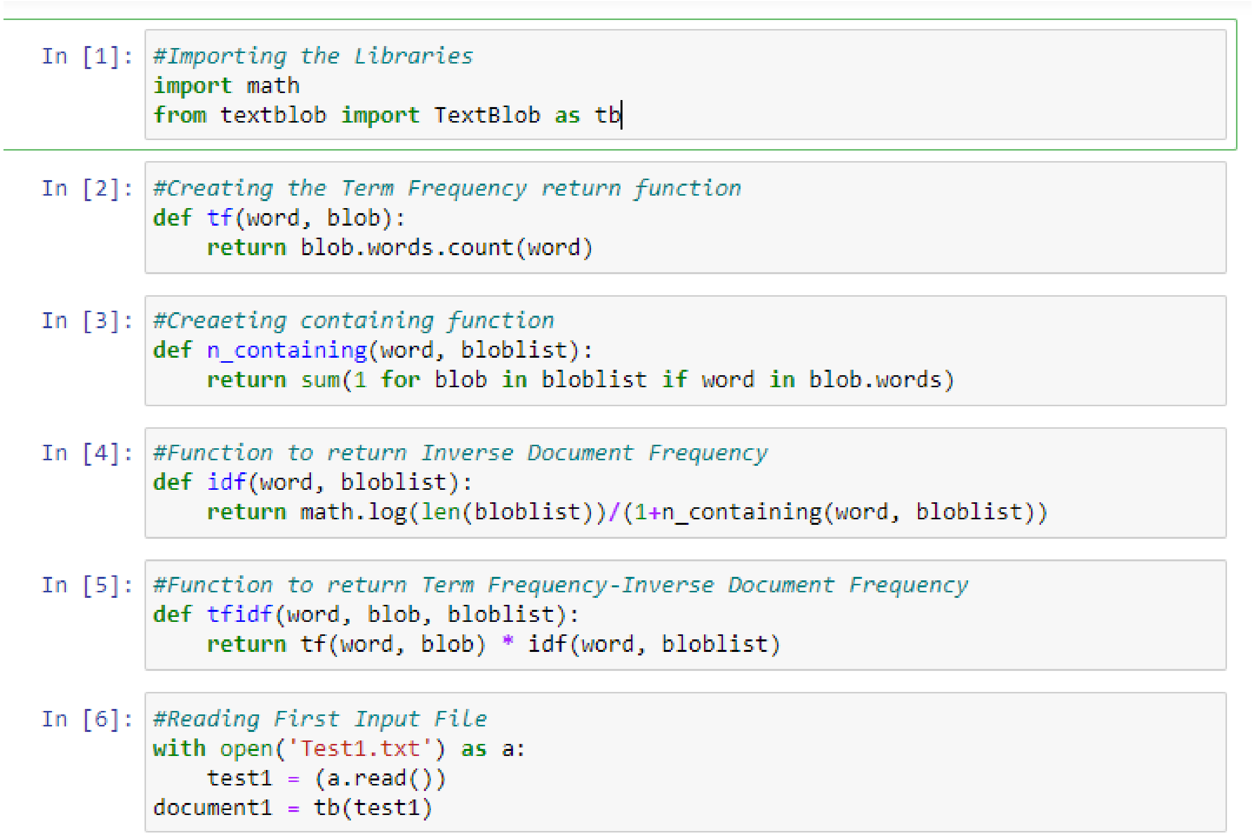
```
This is the fourth input file to check our code for Term Frequency Inverse Document Frequency accuracy code to have as sample space.
```

Text File 5:

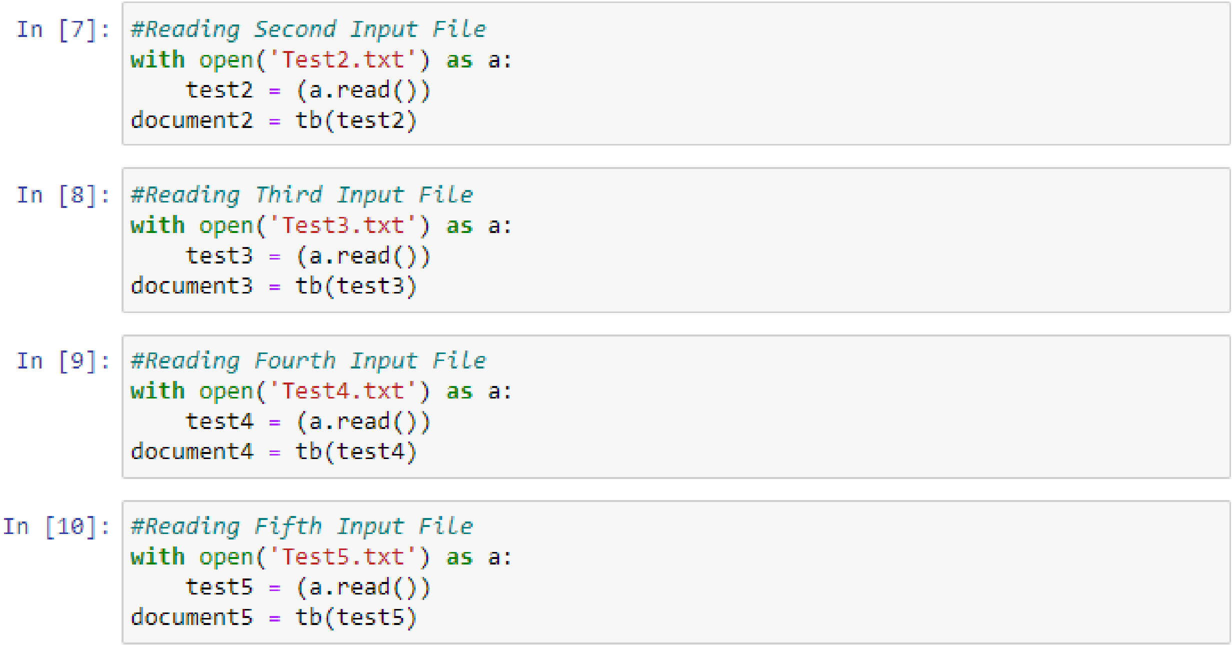


```
This is the final file before we test the working of our code. This file again is aimed at increasing the number of tokens
just so to avoid better number of terms for our code to work with.
```

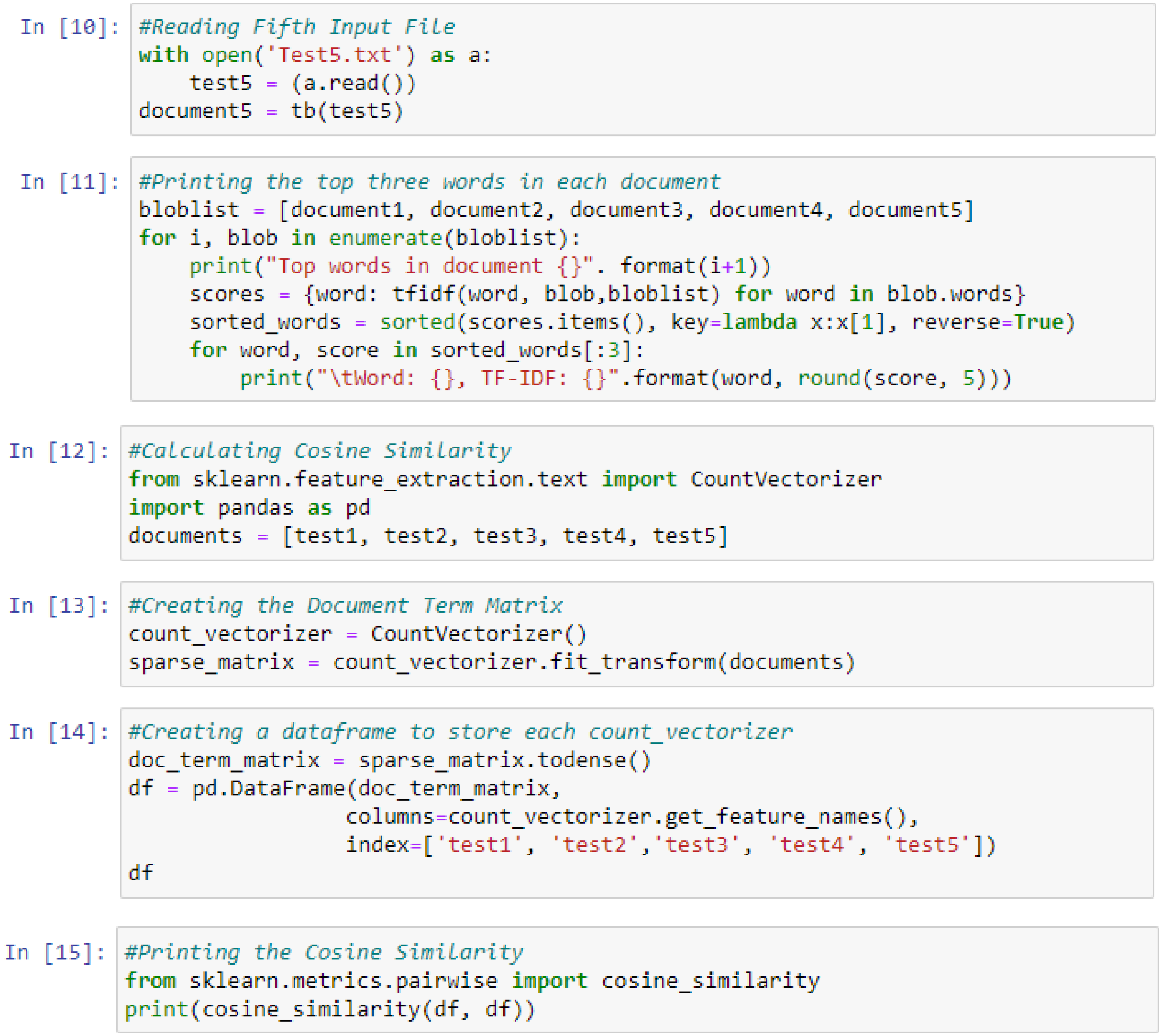
Code:



```
In [1]: #Importing the 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)

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)
```



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

```
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)))
```

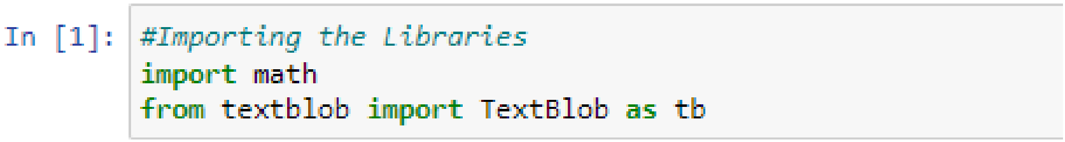
```
In [12]: #Calculating Cosine Similarity
         from sklearn.feature_extraction.text import CountVectorizer
         import pandas as pd
         documents = [test1, test2, test3, test4, test5]
```

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

```
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
```

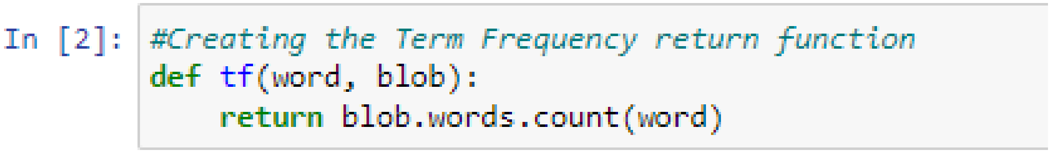
```
In [15]: #Printing the Cosine Similarity
         from sklearn.metrics.pairwise import cosine_similarity
         print(cosine_similarity(df, df))
```

Code Snippets and Outputs:



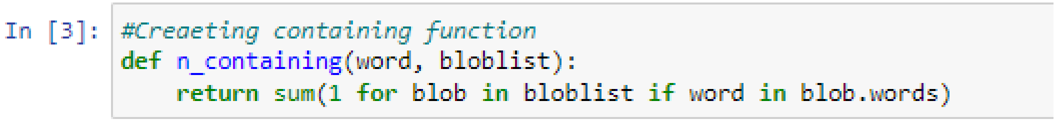
```
In [1]: #Importing the Libraries
        import math
        from textblob import TextBlob as tb
```

Here we are importi8ng the necessary Libraries



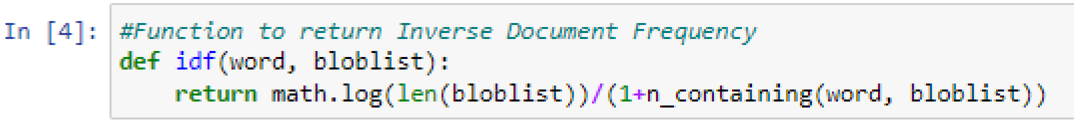
```
In [2]: #Creating the Term Frequency return function
        def tf(word, blob):
            return blob.words.count(word)
```

Here we are creating the Term Frequency return Function which takes word and blob as attributes.



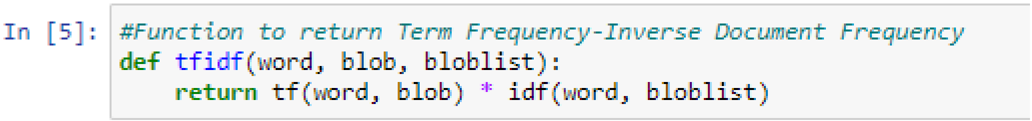
```
In [3]: #Creaeting containing function
        def n_containing(word, bloblist):
            return sum(1 for blob in bloblist if word in blob.words)
```

Here we are now creating the n_containing Function wwhich takes words and bloblist as attributes.



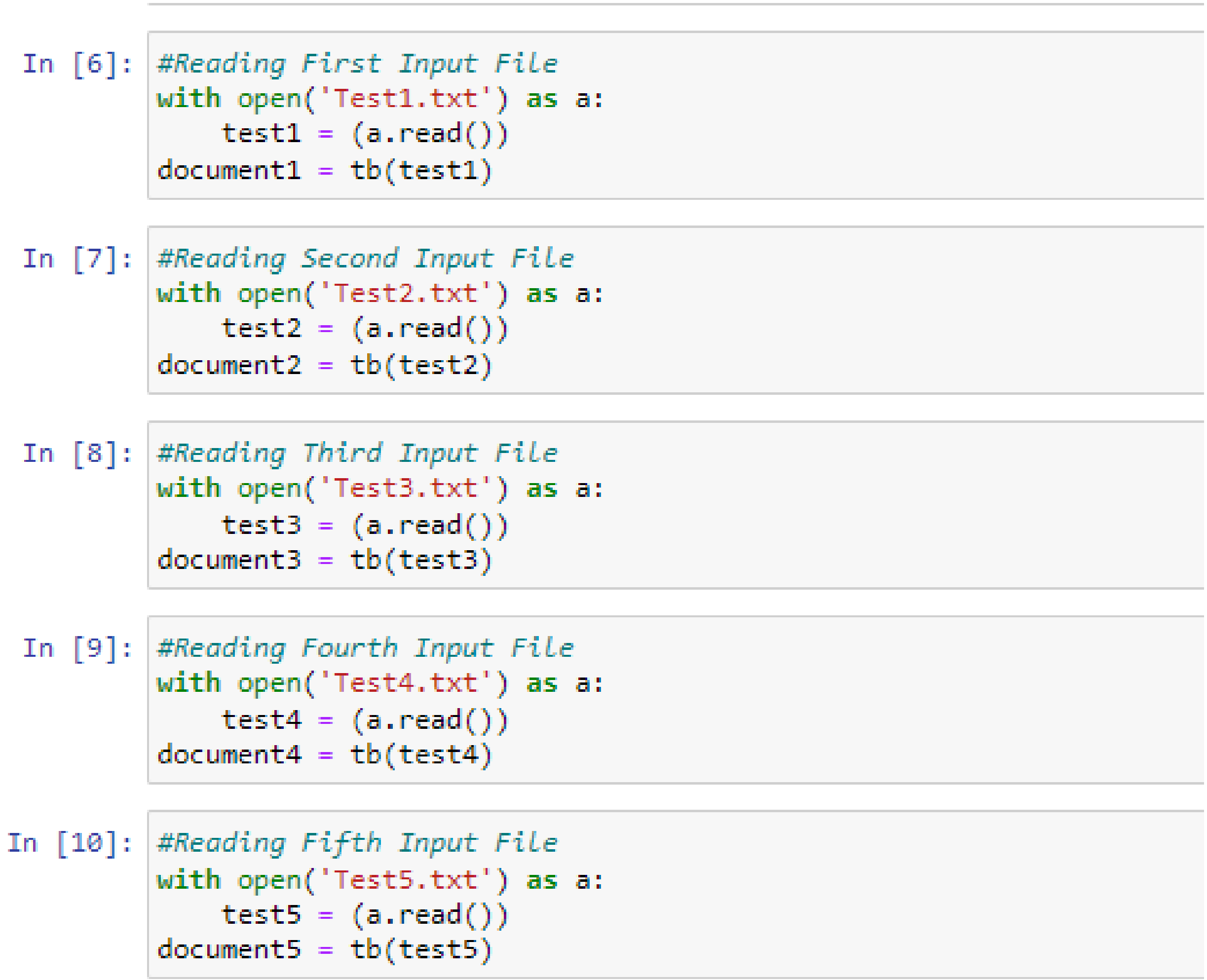
```
In [4]: #Function to return Inverse Document Frequency
        def idf(word, bloblist):
            return math.log(len(bloblist))/(1+n_containing(word, bloblist))
```

A function named idf is created inorder to Inverse the Document Frequency



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

Here we create a Function named ifidtf to return Term Frequency-Inverse Document Frequency



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

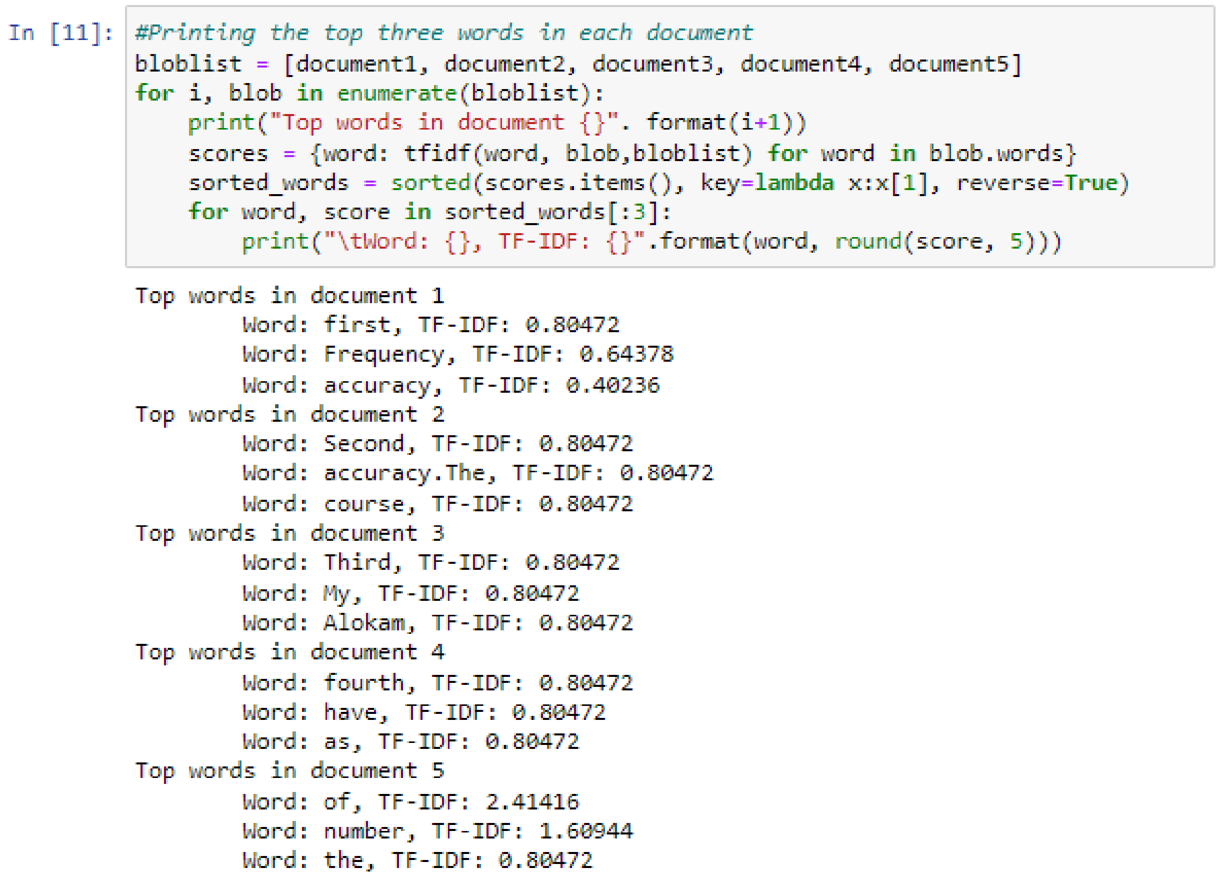
```
In [7]: #Reading Second Input File
        with open('Test2.txt') as a:
            test2 = (a.read())
        document2 = tb(test2)
```

```
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)
```

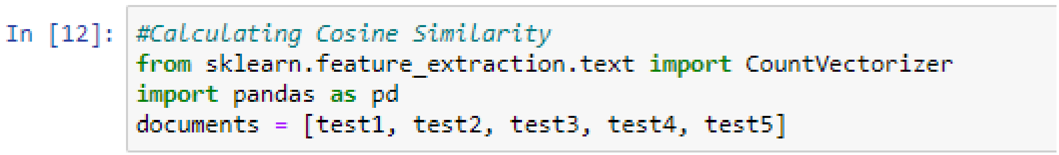
Here we are reading all the 5 Input Text Files(i.e, Test1.txt, Test2.txt, Test3.txt, Test4.txt, Test5.txt)



```
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)))
```

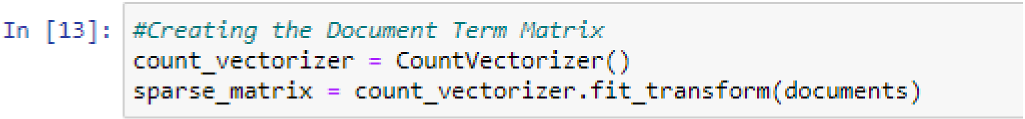
```
Top words in document 1
        Word: first, TF-IDF: 0.80472
        Word: Frequency, TF-IDF: 0.64378
        Word: accuracy, TF-IDF: 0.40236
Top words in document 2
        Word: Second, TF-IDF: 0.80472
        Word: accuracy.The, TF-IDF: 0.80472
        Word: course, TF-IDF: 0.80472
Top words in document 3
        Word: Third, TF-IDF: 0.80472
        Word: My, TF-IDF: 0.80472
        Word: Alokam, TF-IDF: 0.80472
Top words in document 4
        Word: fourth, TF-IDF: 0.80472
        Word: have, TF-IDF: 0.80472
        Word: as, TF-IDF: 0.80472
Top words in document 5
        Word: of, TF-IDF: 2.41416
        Word: number, TF-IDF: 1.60944
        Word: the, TF-IDF: 0.80472
```

Here we've printed the top words in every document. We've printed only top 3 words and the TF-IDF values of them in the same line with the word/term.



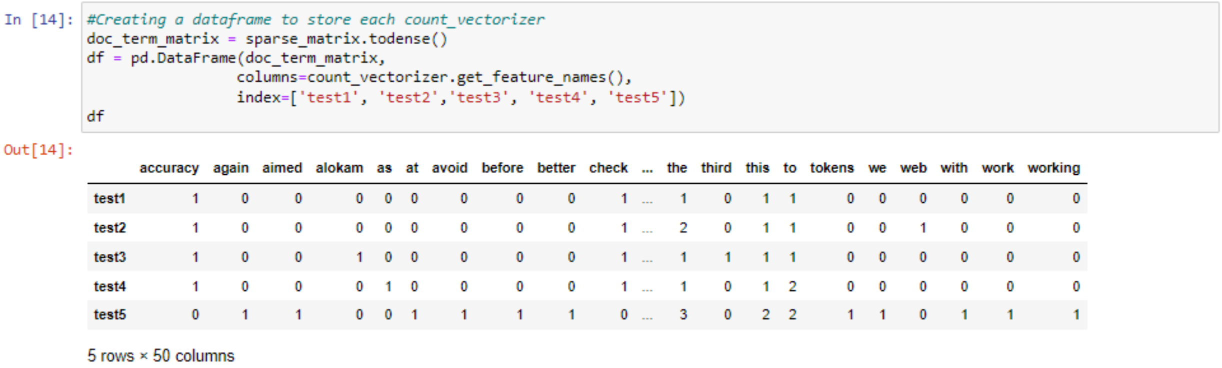
```
In [12]: #Calculating Cosine Similarity
         from sklearn.feature_extraction.text import CountVectorizer
         import pandas as pd
         documents = [test1, test2, test3, test4, test5]
```

Here we are Calculating the Cosine Similarity of all the Input Text files.



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

Here we’ve created count vector which contains the frequency of each word of each document. This is for finding the Cosine Similarity.

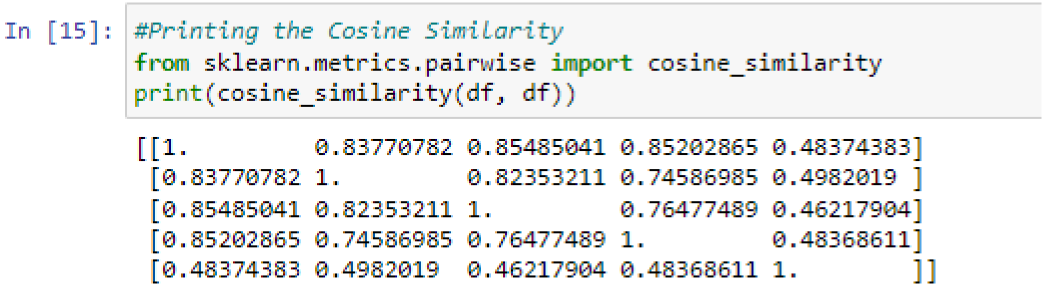


```
In [14]: #Creating a dataframe to store each count_vectorizer
         doc_term_matrix = sparse_matrix.todense()
         df = pd.DataFrame(doc_term_matrix,
                           columns=count_vectorizer.get_feature_names(),
                           index=['test1', 'test2','test3', 'test4', 'test5'])
         df
Out[14]:
```

	accuracy	again	aimed	alokam	as	at	avoid	before	better	check	...	the	third	this	to	tokens	we	web	with	work	working
test1	1	0	0	0	0	0	0	0	0	1	...	1	0	1	1	0	0	0	0	0	0
test2	1	0	0	0	0	0	0	0	0	1	...	2	0	1	1	0	0	1	0	0	0
test3	1	0	0	1	0	0	0	0	0	1	...	1	1	1	1	0	0	0	0	0	0
test4	1	0	0	0	1	0	0	0	0	1	...	1	0	1	2	0	0	0	0	0	0
test5	0	1	1	0	0	1	1	1	1	0	...	3	0	2	2	1	1	0	1	1	1

5 rows × 50 columns

Here we’ve combined the count vectors of every document into Pandas Data Frame.

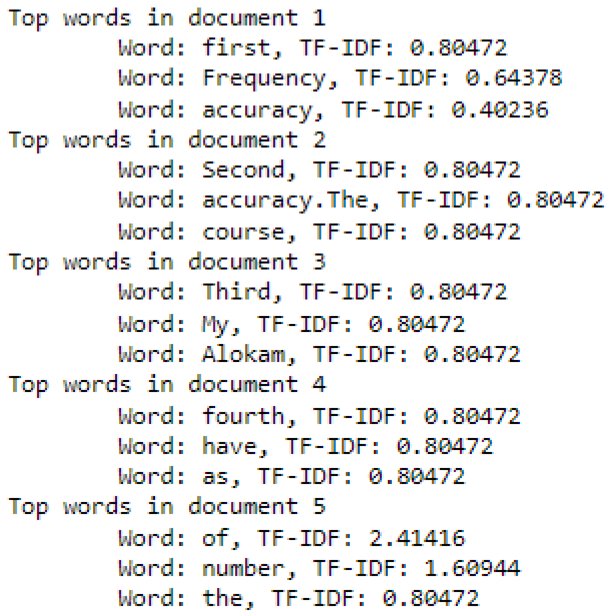


```
In [15]: #Printing the Cosine Similarity
         from sklearn.metrics.pairwise import cosine_similarity
         print(cosine_similarity(df, df))

         [[1.         0.83770782 0.85485041 0.85202865 0.48374383]
          [0.83770782 1.         0.82353211 0.74586985 0.4982019 ]
          [0.85485041 0.82353211 1.         0.76477489 0.46217904]
          [0.85202865 0.74586985 0.76477489 1.         0.48368611]
          [0.48374383 0.4982019  0.46217904 0.48368611 1.        ]]
```

Here we printed Csonne Similarity of Every Document

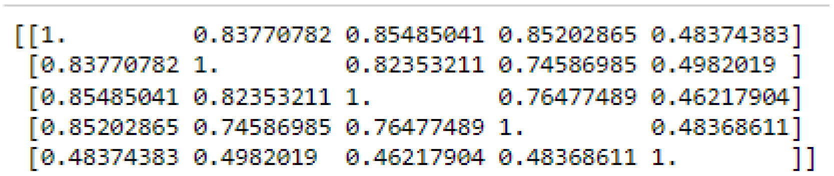
Results and Output

Top words in Each Input Text file:



```
Top words in document 1
        Word: first, TF-IDF: 0.80472
        Word: Frequency, TF-IDF: 0.64378
        Word: accuracy, TF-IDF: 0.40236
Top words in document 2
        Word: Second, TF-IDF: 0.80472
        Word: accuracy.The, TF-IDF: 0.80472
        Word: course, TF-IDF: 0.80472
Top words in document 3
        Word: Third, TF-IDF: 0.80472
        Word: My, TF-IDF: 0.80472
        Word: Alokam, TF-IDF: 0.80472
Top words in document 4
        Word: fourth, TF-IDF: 0.80472
        Word: have, TF-IDF: 0.80472
        Word: as, TF-IDF: 0.80472
Top words in document 5
        Word: of, TF-IDF: 2.41416
        Word: number, TF-IDF: 1.60944
        Word: the, TF-IDF: 0.80472
```

Cosine similarity



```
[[1.         0.83770782 0.85485041 0.85202865 0.48374383]
 [0.83770782 1.         0.82353211 0.74586985 0.4982019 ]
 [0.85485041 0.82353211 1.         0.76477489 0.46217904]
 [0.85202865 0.74586985 0.76477489 1.         0.48368611]
 [0.48374383 0.4982019  0.46217904 0.48368611 1.        ]]
```