INFORMATION RETRIEVAL ASSIGNMENT 2

SUBMITTED BY

ANURAG GAUTAM **MT22015**

MUSKAAN GUPTA **MT22113**

SALONI GARG **MT22063**

SCREENSHOTS FOR THE OUTPUT OF QUES 1

Assumptions: Removed blank spaces as tokens. Replaced punctuation with a space. Didnot remove numbers as token.

Step1: connected with the drive to fetch the dataset and read first 5 files.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[2] import os
    os.getcwd()
```

```
'/content'
```

```
[3] def read_text_file(file_path):
        f = open(file_path, 'r+')
        file_content = f.read()
        print (file_content)
        f.close()


    for i in range (1,6):
          print( "Document "+"cranfield"+str('{0:04}'.format(i)))
          read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
```

Step2: Using beautiful soap library extract the text from each document which is in between text and title tags.

```
[4]  from bs4 import BeautifulSoup
     import requests
     def read_text_file(file_path):
         f = open(file_path, 'r+')
         file_content = f.read()
         soup = BeautifulSoup(file_content,'html.parser')
         txt1=soup.find('title')
         txt2=soup.find('text')
         text=txt1.get_text(' ', strip=True)+" "+txt2.get_text(' ', strip=True)
         f = open(file_path, "w")
         f.write(text)
         f.close()
     for i in range (1,1401):
            read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
```

```
[5]  def read_text_file(file_path):
         f = open(file_path, 'r+')
         file_content = f.read()
         print(file_content)
         print('\n')
         f.close()

     for i in range (1,6):
           print( "Document "+"cranfield"+str('{0:04}'.format(i)))
           read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
```

Step3 : did some preprocessing like lowercasing, tokennization,remove punctuations, stopwords and extra blank spaces.

```
[6]  def read_text_file(file_path):
         f = open(file_path, 'r+')
         file_content = f.read()
         print (file_content)
         f.close()
     def to_lower_case(file_path):
         f = open(file_path, 'r')
         g= f.read().lower()
         f1 = open(file_path, 'w')
         f1.write(g)
         f.close()
         f1.close()
     print('\n')
     print("before converting to lower case")
     for i in range (1,6):
            print( "Document "+"cranfield"+str('{0:04}'.format(i)))
            read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
     for i in range (1,1401):
            to_lower_case('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
     print('\n')
     print("After converting to lower case")
     for i in range (1,6):
            print( "Document "+"cranfield"+str('{0:04}'.format(i)))
            read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
```

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

def read_text_file(file_path):
    f = open(file_path, 'r+')
    file_content = f.read()
    print (file_content)
    f.close()
def perform_tokenization(file_path):
    f = open(file_path, 'r')
    word=f.read()
    tokens=word_tokenize(word)
    f1 = open(file_path, 'w')
    for i in tokens:
      f1.write(str(i)+str(" "))
    f.close()
    f1.close()

print("Before tokenization")
for i in range (1,6):
       print( "Document "+"cranfield"+str('{0:04}'.format(i)))
       read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
for i in range (1,1401):
       perform_tokenization('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
print("After tokenization")
for i in range (1,6):
       print( "Document "+"cranfield"+str('{0:04}'.format(i)))
       read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
```

```
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

def read_text_file(file_path):
    f = open(file_path, 'r+')
    file_content = f.read()
    print (file_content)
    f.close()
def remove_punctuation(file_path):
    f = open(file_path, 'r')
    word=f.read()
    test_str =word.translate(str.maketrans(string.punctuation, ' ' * len(string.punctuation)))
    f1 = open(file_path, 'w')
    for i in test_str:
      f1.write(str(i))
    f.close()
    f1.close()
print('\n')
print("before removeing punctuation")
for i in range (1,6):
       print( "Document "+"cranfield"+str('{0:04}'.format(i)))
       read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
for i in range (1,1401):
       remove_punctuation('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
print('\n')
print("After  removeing punctuation")
```

```
def read_text_file(file_path):
    f = open(file_path, 'r+')
    file_content = f.read()
    print (file_content)
    f.close()
def remove_stopwords(file_path):
    f = open(file_path, 'r')
    word=f.read()
    word_tokens=word_tokenize(word)
    stop_words = set(stopwords.words('english'))
    filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
    filtered_sentence = []
    for w in word_tokens:
      if w not in stop_words:
        filtered_sentence.append(w)
    f1 = open(file_path, 'w')
    for i in filtered_sentence:
      f1.write(str(i)+str(" "))
    f.close()
    f1.close()
print('\n')
print("before removing stopwords")
for i in range (1,6):
       print( "Document "+"cranfield"+str('{0:04}'.format(i)))
       read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
for i in range (1,1401):
       remove_stopwords('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
print('\n')
print("After removing stopwords")
for i in range (1,6):
       print( "Document "+"cranfield"+str('{0:04}'.format(i)))
       read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
```

```
[11] import string
     import re

     #need to correct it
     def read_text_file(file_path):
         f = open(file_path, 'r+')
         file_content = f.read()
         print (file_content)
         f.close()
     def remove_blankspace(file_path):
         f = open(file_path, 'r')
         text=f.read()
         res = re.sub(' +', ' ', text)
         f1 = open(file_path, 'w')
         f1.write(res)
         f.close()
         f1.close()
     print('\n')
     print("before removing blankspace")
     for i in range (1,6):
            print( "Document "+"cranfield"+str('{0:04}'.format(i)))
            read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
     for i in range (1,1401):
            remove_blankspace('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
     print('\n')
     print("After removing blankspace")
     for i in range (1,6):
            print( "Document "+"cranfield"+str('{0:04}'.format(i)))
            read_text_file('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
```

Step4: made a global dictionary which consist of unique word present in all the documents. And remove token which consist only space. Afterwords made a row count dictionary which the frequency of each term present in particular document.

```
[12] global_list={}
     def spliting_sentance(file_path):
       f = open(file_path, 'r+')
       file_content = f.read()
       file_content=file_content.split(' ')
       file_content=set(file_content)
       for item in file_content:
         if item in global_list:
           continue
         else:
           global_list[item]=0
       f.close()

     for i in range(1,1401):
         spliting_sentance('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/cranfield'+str('{0:04}'.format(i)))
```

```
[45] print(len(global_list))
```

```
[13] doc_name=[]
     for i in range(1,1401):
         doc_name.append("cranfield"+str('{0:04}'.format(i)))

     print(doc_name)
```

```
['cranfield0001', 'cranfield0002', 'cranfield0003', 'cranfield0004', 'cranfield0005', 'cranfield0006', 'cranfield0007', 'cranfield0008', 'cranfield0009', '
```

```
[14] print(global_list)
     del global_list['']
     print(global_list)
```

```
{'theory': 0, '': 0, 'boundary': 0, 'together': 0, 'effects': 0, 'substantial': 0, 'control': 0, 'increase': 0, 'due': 0, 'produced': 0, 'ratios': 0, 'found
{'theory': 0, 'boundary': 0, 'together': 0, 'effects': 0, 'substantial': 0, 'control': 0, 'increase': 0, 'due': 0, 'produced': 0, 'ratios': 0, 'found': 0,
```

```
[15] import copy
     Tf_dict={}
     for doc in doc_name:
       Tf_dict[doc]=copy.deepcopy(global_list)
     for doc in doc_name:
       f = open('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/'+doc, 'r+')
       file_content = f.read()
       file=file_content.split(' ')
       for item in file:
           if item in Tf_dict[doc]:
             Tf_dict[doc][item]+=1
     print(Tf_dict)
```

Step6:same as row dictionary created binary dictionary which only tell term is present in particular document or not. Created term dictionary which follows the formular like divide each tokens frequency with total frequency.

```
[16]
     Tf_bindict={}
     for doc in doc_name:
       Tf_bindict[doc]=copy.deepcopy(global_list)
     for doc in doc_name:
       f = open('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/'+doc, 'r+')
       file_content = f.read()
       file=file_content.split(' ')
       for item in file:
           if item in Tf_bindict[doc]:
             Tf_bindict[doc][item]=1
     print(Tf_bindict)
```

```
     Tf_termdict={}
     for doc in doc_name:
       Tf_termdict[doc]=copy.deepcopy(global_list)
     for doc in doc_name:
       f = open('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/'+doc, 'r+')
       file_content = f.read()
       file=file_content.split(' ')
       for item in file:
           if item in Tf_termdict[doc]:
             Tf_termdict[doc][item]+=1
       for key,val in Tf_termdict[doc].items():
         Tf_termdict[doc][key]/=len(file)
     print(Tf_termdict)
     for i in Tf_dict:
       print(Tf_dict[i])
```

step7 : created normalizesed dictionary using formula.

```
[18] import copy
     import math
     Tf_normdict=copy.deepcopy(Tf_dict)

     for doc in doc_name:
       for key,val in Tf_normdict[doc].items():
         Tf_normdict[doc][key]=(math.log(1+Tf_normdict[doc][key]))
     print(Tf_normdict)
```

step8 : created double normalizesed dictionary using formula.

```
[19] Tf_dnormdict={}
     for doc in doc_name:
       Tf_dnormdict[doc]=copy.deepcopy(global_list)
     for doc in doc_name:
       f = open('/content/drive/MyDrive/CSE508_Winter2023_Dataset3/'+doc, 'r+')
       file_content = f.read()
       file=file_content.split(' ')
       max_count=max(Tf_dnormdict[doc].values())
       for item in file:
           if item in Tf_dnormdict[doc]:
             Tf_dnormdict[doc][item]+=1
             if max_count< max(Tf_dnormdict[doc].values()):
                   max_count=max(Tf_dnormdict[doc].values())
       for key,val in Tf_dnormdict[doc].items():
         Tf_dnormdict[doc][key]=0.5+(0.5*(Tf_dnormdict[doc][key]/max_count))
     print(Tf_dnormdict)
```

Idf metrix

```
[20] #idf dictionary
     import math
     tf_idf={}
     tf_idf=copy.deepcopy(global_list)
     for key,value in tf_idf.items():
       c=0
       for doc in doc_name:
         if Tf_bindict[doc][key]==1:
           c+=1
       tf_idf[key]=math.log2(1400/(c+1))
     print(tf_idf)
```

Tf-idf metrix for double normalization

```
[21] #tf_idf dictionary for double normalization
     tf_idf_new1=copy.deepcopy(Tf_dnormdict)
     for i in Tf_dnormdict:
       for j in tf_idf:
         tf_idf_new1[i][j]=Tf_dnormdict[i][j]*tf_idf[j]
     print(tf_idf_new1)
```

Tf-idf matrix for normalization

```
[22] #tf_idf dictionary for normalization
     tf_idf_new2=copy.deepcopy(Tf_normdict)
     for i in Tf_normdict:
       for j in tf_idf:
         tf_idf_new2[i][j]=Tf_normdict[i][j]*tf_idf[j]
     print(tf_idf_new2)
```

Tf-idf matrix for term frequency

```
[23] #tf_idf dictionary for term frequency
     tf_idf_new3=copy.deepcopy(Tf_termdict)
     for i in Tf_termdict:
       for j in tf_idf:
         tf_idf_new3[i][j]=Tf_termdict[i][j]*tf_idf[j]
     print(tf_idf_new3)
```

Tf-idf matrix for binary weight

```
[24] #tf_idf dictionary for binary
     tf_idf_new4=copy.deepcopy(Tf_bindict)
     for i in Tf_bindict:
       for j in tf_idf:
         tf_idf_new4[i][j]=Tf_bindict[i][j]*tf_idf[j]
     print(tf_idf_new4)
```

Tf-idf matrix for raw count

```
[25] #tf_idf dictionary for raw count
     tf_idf_new5=copy.deepcopy(Tf_dict)
     for i in Tf_dict:
       for j in tf_idf:
         tf_idf_new5[i][j]=Tf_dict[i][j]*tf_idf[j]
     print(tf_idf_new5)
```

Query input by user

```
[47]
query=input("input the query")
query=query.lower()
```

```
input the querytogether':, 'results':'loading': 'effect'
```

```
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
word_tokens=word_tokenize(query)
stop_words = set(stopwords.words('english'))
filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]
filtered_sentence
```

Doing preprocessing of query like remove punctuation,stopwords etc.

```
[28] j=0
     for i in filtered_sentence:
         filtered_sentence[j]=i.translate(str.maketrans('', '', string.punctuation))
         j+=1
```

```
[29] filtered_sentence
```

```
['together', '', '', '', 'results', '', '', 'loading', '', '', 'effect', '']
```

```
[30] j=0
     while(j<len(filtered_sentence)):
       if(filtered_sentence[j]=='' or filtered_sentence[j]==' ' ):
         del filtered_sentence[j]
         j-=1
       j+=1
     print(filtered_sentence)
```

```
['together', 'results', 'loading', 'effect']
```

```
[31] j=0
     for i in filtered_sentence:
         if filtered_sentence[j]=='' or filtered_sentence[j]==' ':
           del filtered_sentence[j]
           j-=1
         j+=1
     print(filtered_sentence)
```

Tf-idf score for query using double normalization matrix

```
32] #TF-idf with double normalization for query
    list_first={}
    for i in doc_name:
      s=0
      for j in Tf_dnormdict[i]:
            if (Tf_dnormdict[i][j])!=0:
               if j in filtered_sentence :
                  s+=tf_idf_new1[i][j]
      list_first[i]=s
    list_first
```

```
[34]
      j=0
      for i in converted_dict1:
          if j<5:
            print(i)
          else:
            break
          j+=1
```

```
[33] sortedDict1 = sorted(list_first.items(), key=lambda x:x[1], reverse=True)
     converted_dict1 = dict(sortedDict1 )
     converted_dict1
```

Tf-idf score for query using term frequency matrix

```
[36] #TF-idf with term frequency for query
     list_first3={}
     for i in doc_name:
       s=0
       for j in Tf_termdict[i]:
             if (Tf_termdict[i][j])!=0:
                if j in filtered_sentence :
                   s+=tf_idf_new3[i][j]
       list_first3[i]=s
     print(list_first3)
     sortedDict3 = sorted(list_first3.items(), key=lambda x:x[1], reverse=True)
     converted_dict3 = dict(sortedDict3)
     print(converted_dict3)
     j=0
     for i in converted_dict3:
         if j<5:
           print(i)
         else:
           break
         j+=1
```

Tf-idf score for query using raw count matrix

```
[38] #TF-idf with raw count for query
     list_first5={}
     for i in doc_name:
       s=0
       for j in Tf_dict[i]:
             if (Tf_dict[i][j])!=0:
                if j in filtered_sentence :
                   s+=tf_idf_new5[i][j]
       list_first5[i]=s
     print(list_first5)
     sortedDict5 = sorted(list_first5.items(), key=lambda x:x[1], reverse=True)
     converted_dict5 = dict(sortedDict5 )
     print(converted_dict5)
     j=0
     for i in converted_dict5:
         if j<5:
           print(i)
         else:
           break
         j+=1
```

Jaccard Coefficient

```
[39] #tokens for query
     filtered_sentence
```

```
['together', 'results', 'loading', 'effect']
```

```
[40] #intersection of query and document term
     list_second={}
     for i in doc_name:
       s=0
       for j in filtered_sentence:
             if j in Tf_dict[i]:
               if (Tf_dict[i][j])!=0:
                   s+=1
             else:
               continue
       list_second[i]=s
     list_second
```

```
41] #total no of terms present in each documents
    list_third={}
    for i in doc_name:
      s=0
      for j in Tf_dict[i]:
            if (Tf_dict[i][j])!=0:
                s+=Tf_dict[i][j]
      list_third[i]=s
    list_third
```

```
[42] #union of query tokens and document tokens
     list_four={}
     k=len(filtered_sentence)
     j=0
     for i in list_third:
       s=(list_third[i]+k)-list_second[i]
       list_four[i]=s
     list_four
```

```
[43] #calculating the jacard coefficient
     list_five={}
     for i in list_four:
       list_five[i]=list_second[i]/list_four[i]
     sortedDict1 = sorted(list_five.items(), key=lambda x:x[1], reverse=True)
     converted_dict1 = dict(sortedDict1 )
     converted_dict1
```

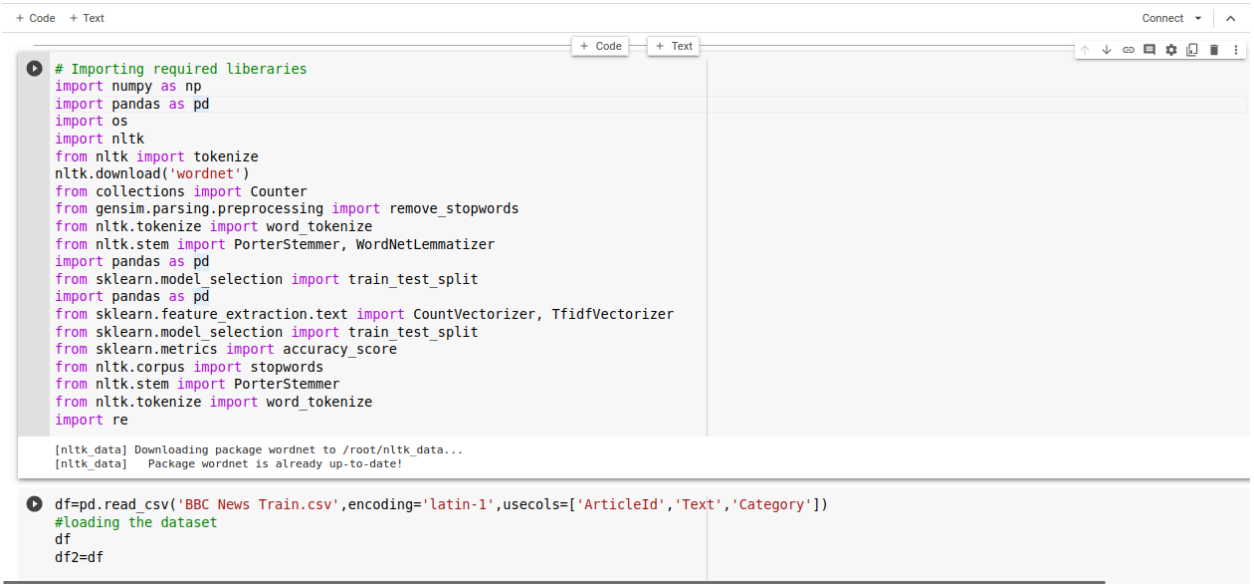
```
[44] j=0
     for i in converted_dict1:
         if j<10:
           print(i)
         else:
           break
         j+=1
```

SCREENSHOTS FOR THE OUTPUT OF QUES 2

Assumption: Assumption that each term appears at least twice in the DataFrame is made in find_cf() function in order to avoid the possibility of having terms with zero frequency in the DataFrame.

1. **Preprocessing the dataset:**

Importing Required Libraries & loading the dataset



```
# Importing required liberaries
import numpy as np
import pandas as pd
import os
import nltk
from nltk import tokenize
nltk.download('wordnet')
from collections import Counter
from gensim.parsing.preprocessing import remove_stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
import pandas as pd
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import re
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
df=pd.read_csv('BBC News Train.csv',encoding='latin-1',usecols=['ArticleId','Text','Category'])
#loading the dataset
df
df2=df
```

Preprocessing steps:

1. Removing Punctuations
2. Removing Stopwords
3. Converting all text to lowercase
4. Perform tokenization
5. Stemming and lemmatization

```
#count number of words
def num_of_words(df):
    df['Text'].apply(lambda x : len(str(x).split(" ")))
    print(df[['Text']].head())
num_of_words(df)
```

```
                                                Text
0  worldcom ex-boss launches defence lawyers defe...
1  german business confidence slides german busin...
2  bbc poll indicates economic gloom citizens in ...
3  lifestyle  governs mobile choice  faster  bett...
4  enron bosses in $168m payout eighteen former e...
```

```
#count number of characters
def num_of_chars(df):
    df['Text'].str.len() ## this also includes spaces
    print(df[['Text']].head())

num_of_chars(df)
```

```
                                                Text
0  worldcom ex-boss launches defence lawyers defe...
1  german business confidence slides german busin...
2  bbc poll indicates economic gloom citizens in ...
3  lifestyle  governs mobile choice  faster  bett...
4  enron bosses in $168m payout eighteen former e...
```

```
#number of stop words
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
#removing punctuation
def punctuation_removal(text):
    text=re.sub(r'[^\w\s]', '', text).replace("_"," ")
    return text
df['Text_new']=df['Text'].apply(punctuation_removal)
df
```

	ArticleId	Text	Category	Text_new
0	1833	worldcom ex-boss launches defence lawyers defe...	business	worldcom exboss launches defence lawyers defen...
1	154	german business confidence slides german busin...	business	german business confidence slides german busin...
2	1101	bbc poll indicates economic gloom citizens in ...	business	bbc poll indicates economic gloom citizens in ...
3	1976	lifestyle governs mobile choice faster bett...	tech	lifestyle governs mobile choice faster bett...
4	917	enron bosses in $168m payout eighteen former e...	business	enron bosses in 168m payout eighteen former en...
...	...	...	...	...
1485	857	double eviction from big brother model caprice...	entertainment	double eviction from big brother model caprice...
1486	325	dj double act revamp chart show dj duo jk and ...	entertainment	dj double act revamp chart show dj duo jk and ...
1487	1590	weak dollar hits reuters revenues at media gro...	business	weak dollar hits reuters revenues at media gro...
1488	1587	apple ipod family expands market apple has exp...	tech	apple ipod family expands market apple has exp...
1489	538	sanity worm makes unwelcome visit thousands of...	tech	sanity worm makes unwelcome visit thousands of...

+ Code + Text Connect

```
# removing stop words
from gensim.parsing.preprocessing import remove_stopwords
def stop_words_removal(text):
    text=remove_stopwords(text)
    return text

df['Text_new']=df['Text_new'].apply(stop_words_removal)
```

```
df
```

	ArticleId	Text	Category	Text_new
0	1833	worldcom ex-boss launches defence lawyers defe...	business	worldcom exboss launches defence lawyers defen...
1	154	german business confidence slides german busin...	business	german business confidence slides german busin...
2	1101	bbc poll indicates economic gloom citizens in ...	business	bbc poll indicates economic gloom citizens maj...
3	1976	lifestyle governs mobile choice faster bett...	tech	lifestyle governs mobile choice faster better ...
4	917	enron bosses in $168m payout eighteen former e...	business	enron bosses 168m payout eighteen enron direct...
...	...	...	...	...
1485	857	double eviction from big brother model caprice...	entertainment	double eviction big brother model caprice holb...
1486	325	dj double act revamp chart show dj duo jk and ...	entertainment	dj double act revamp chart dj duo jk joel taki...
1487	1590	weak dollar hits reuters revenues at media gro...	business	weak dollar hits reuters revenues media group ...
1488	1587	apple ipod family expands market apple has exp...	tech	apple ipod family expands market apple expande...
1489	538	santy worm makes unwelcome visit thousands of ...	tech	santy worm makes unwelcome visit thousands web...

1490 rows × 4 columns

+ Code + Text Connect

1489	538	santy worm makes unwelcome visit thousands of ...	tech	santy worm makes unwelcome visit thousands web...

1490 rows × 4 columns

```
#converting all text to lowercase
def lower_case(text):
    text=text.lower()
    return text

df['Text_new'] = df['Text_new'].apply(lower_case)
```

```
df
```

	ArticleId	Text	Category	Text_new
0	1833	worldcom ex-boss launches defence lawyers defe...	business	worldcom exboss launches defence lawyers defen...
1	154	german business confidence slides german busin...	business	german business confidence slides german busin...
2	1101	bbc poll indicates economic gloom citizens in ...	business	bbc poll indicates economic gloom citizens maj...
3	1976	lifestyle governs mobile choice faster bett...	tech	lifestyle governs mobile choice faster better ...
4	917	enron bosses in $168m payout eighteen former e...	business	enron bosses 168m payout eighteen enron direct...
...	...	...	...	...
1485	857	double eviction from big brother model caprice...	entertainment	double eviction big brother model caprice holb...
1486	325	dj double act revamp chart show dj duo jk and ...	entertainment	dj double act revamp chart dj duo jk joel taki...
1487	1590	weak dollar hits reuters revenues at media gro...	business	weak dollar hits reuters revenues media group ...
1488	1587	apple ipod family expands market apple has exp...	tech	apple ipod family expands market apple expande...
1489	538	santy worm makes unwelcome visit thousands of ...	tech	santy worm makes unwelcome visit thousands web...

1490 rows × 4 columns

```
# perform tokenization
# stemming
# lemmatization
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
def perform_tok_stemm_lemm(text):
     wordTokens=tokenize.RegexpTokenizer(r'\w+').tokenize(text)
     stemmer = PorterStemmer()
     tokens = [stemmer.stem(token) for token in wordTokens]
     finalTokens=[WordNetLemmatizer().lemmatize(w) for w in tokens]
     return finalTokens

df['Text_new'] = df['Text_new'].apply(perform_tok_stemm_lemm)
df
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

	ArticleId	Text	Category	Text_new
0	1833	worldcom ex-boss launches defence lawyers defe...	business	[worldcom, exboss, launch, defenc, lawyer, def...
1	154	german business confidence slides german busin...	business	[german, busi, confid, slide, german, busi, co...
2	1101	bbc poll indicates economic gloom citizens in ...	business	[bbc, poll, indic, econom, gloom, citizen, maj...
3	1976	lifestyle governs mobile choice faster bett...	tech	[lifestyl, govern, mobil, choic, faster, bette...
4	917	enron bosses in $168m payout eighteen former e...	business	[enron, bos, 168m, payout, eighteen, enron, di...
...	...	...	...	...
1485	857	double eviction from big brother model caprice...	entertainment	[doubl, evict, big, brother, model, capric, ho...
1486	325	dj double act revamp chart show dj duo jk and ...	entertainment	[dj, doubl, act, revamp, chart, dj, duo, jk, j...
1487	1590	weak dollar hits reuters revenues at media gro...	business	[weak, dollar, hit, reuter, revenu, medium, gr...

Finding TF-ICF values

find_cf() function initializes an empty dictionary ‘cd_dict’ to store the collection frequency of each term.

If the term is not already in ‘cf_dict’ it is added with the frequency of 2. If the term is already in the dictionary, its frequency is incremented by 1.

1489	538	santy worm makes unwelcome visit thousands of ...	tech	[santi, worm, make, unwelcom, visit, thousand,...

1490 rows × 4 columns

```
# printing all countof each classes
classes=Counter(df['Category'])
classes
```

```
Counter({'business': 336,
         'tech': 261,
         'politics': 274,
         'sport': 346,
         'entertainment': 273})
```

```
# finding CF
def find_cf(df):
  # Create an empty dictionary to store the collection frequency of each term.
  cf_dict={}
  for index in range(len(df['Text_new'])):
    for term in df['Text_new'][index]:
      # If the current term is not already in the dictionary, add it with a frequency of 2.
      if cf_dict.get(term) is None:
        cf_dict[term]=2
        # If the term is already in the dictionary, increment its frequency by 1.
      else :
        cf_dict[term]=cf_dict[term]+1
  return cf_dict
```

```
CF= find_cf(df)
```

```
print(CF)
```

```
# function to create a dictionary for each class
def dictionary_of_words(df):
  Dict= {}
  for index in range(0,len(df)):
      try:
          Dict[df['Category'][index]] = Dict[df['Category'][index]] + df['Text_new'][index]
      except:
          Dict[df['Category'][index]] = df['Text_new'][index]
  return Dict
```

```
dict_of_words= dictionary_of_words(df)
dict_of_words
```

```
{'business': ['worldcom',
  'exboss',
  'launch',
  'defenc',
  'lawyer',
  'defend',
  'worldcom',
  'chief',
  'berni',
  'ebber',
  'batteri',
  'fraud',
  'charg',
  'call',
  'compani',
  'whistleblow',
  'wit',
  'cynthia',
  'cooper',
  'worldcom',
```

tf_icf() function calculates the tf-icf value of the term and stores it in a dictionary called tf_Icf. Finally, the function returns a dictionary tfIcf, where the keys are the class names and the values are dictionaries containing the tf-icf values of each term for that class.

```
print(len(dict_of_words))
```

```
5
```

```
# function to find unique terms
def find_Unique(dict_of_words):
    unique_Words = set(word for words in dict_of_words.values() for word in words)
    unique_Words_Count = len(unique_Words)
    return unique_Words, unique_Words_Count
```

```
# distinct_word shows the all the distict words present
# distinct_words_count is the count of distinct words
distinct_words,distinct_words_count = find_Unique(dict_of_words)
```

```
# finding tf-Icf values referring to each class
import math

def tf_Icf(word_dict, cf_dict, classes_Freq):
    tfIcf = {}
    # Loop through each class in the word_dict.
    for className, doc in word_dict.items():
        termFreqDict = {}
        # Loop through each term in the document and calculate its frequency.
        for term in doc:
            if term not in termFreqDict:
                termFreqDict[term] = 0
            termFreqDict[term] += 1

        tf_Icf = {}
        for term, freq in termFreqDict.items():
            # Calculate the term frequency (tf)
```

```
        tf_Icf = {}
        for term, freq in termFreqDict.items():
            # Calculate the term frequency (tf).
            tf = freq / len(doc)
            # Get the collection frequency (cf) of the term from the cf_dict.
            cf = cf_dict[term]
            # Calculate the inverse class frequency (icf) of the term.
            icf = math.log(len(classes_Freq) / cf)
            # Calculate the tf-icf value of the term.
            tf_Icf[term] = tf * icf

        tfIcf[className] = tf_Icf

    return tfIcf
```

```
test_TFICF = tf_Icf(dict_of_words, CF, classes)
```

```
test_TFICF
```

```
{'business': {'worldcom': -0.00206165466796873,
  'exboss': 1.626651882006753e-05,
  'launch': -0.0025133512454043718,
  'defenc': -0.0007504578833263215,
  'lawyer': -0.0011265254051962042,
  'defend': -0.0002575102014301675,
  'chief': -0.010006240028198351,
  'berni': -7.486893688947015e-05,
  'ebber': -0.0016411012420063703,
  'batteri': -2.7988248404672422e-05,
  'fraud': -0.0030627746378911608,
  'charg': -0.0041868852037485 44,
  'call': -0.003971997700084044,
```

2. Splitting the dataset

(2) Part

```
# Splitting the dataset into train test in ratio of 70-30
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.30, random_state=43)
```

```
train_data=train_data.reset_index(drop=True)
test_data= test_data.reset_index(drop=True)
train_data
```

	ArticleId	Text	Category	Text_new
0	248	steady job growth continues in us the us creat...	business	[steadi, job, growth, continu, creat, fewer, j...
1	1034	labour s eu propaganda a taxpayer subsidise...	politics	[labour, s, eu, propaganda, taxpay, subsidis, ...
2	2091	oscar nominee dan o herlihy dies irish actor d...	entertainment	[oscar, nomine, dan, o, herlihi, die, irish, a...
3	991	us in eu tariff chaos trade row the us has ask...	business	[eu, tariff, chao, trade, row, ask, world, tra...
4	654	lennon brands rangers favourites celtic s neil...	sport	[lennon, brand, ranger, favourit, celtic, s, n...
...	...	...	...	...
1038	342	u2 s desire to be number one u2 who have won ...	entertainment	[u2, s, desir, number, u2, won, prestigi, gram...
1039	1518	tory backing for id cards the tories are to ba...	politics	[torus, back, id, card, torus, controversi, go...
1040	1439	microsoft debuts security tools microsoft is r...	tech	[microsoft, debut, secur, tool, microsoft, rel...
1041	283	consumers snub portable video consumers want...	tech	[consum, snub, portabl, video, consum, want, m...
1042	2114	gadget show heralds mp3 christmas partners of ...	tech	[gadget, herald, mp3, christma, partner, love,...

1043 rows × 4 columns

3. Training the Naive Bayes classifier with TF-ICF

4. Testing the Naive Bayes classifier with TF-ICF

Created function to implement Naive-Bayes algorithm.
Evaluated the accuracy of our dataset using predicted value and truth value that we get from the Naive-Bayes algorithm after training the dataset.
We have implemented Naive-Bayes from scratch, in this function we are calculating the probability of each feature given each category and the probability of each category based on the frequency of documents as well.

+ Code + Text Connect

1042	2114	gadget show heralds mp3 christmas partners of ...	tech	[gadget, herald, mp3, christma, partner, love,...

1043 rows × 4 columns

(3) and (4) Part

```
# def compute_word_freq(w, l,classOne,classTwo):
#     try:
#         return classOne[l, w],classTwo[l]
#     except:
#         return 0, classTwo[l]
```

```
def compute_word_freq(word, label, class_one, class_two):
    freq_one = class_one.get((label, word), 0)
    freq_two = class_two.get(label, 0)
    return freq_one, freq_two
```

```
# Implementing Naive Bayes classifier with the TF-ICF weighting scheme
def naive_bayes_classifier(vocab_size, class_prior_prob, train_data, test_data, classes, class_f, class_c):
    true_labels = []
    predicted_labels = []

    # Iterate over each row in the test data
    for i, row in test_data.iterrows():
        true_labels.append(row['Category'])  #Actual values

        # Compute the log probabilities for each class
        class_word_probs = []
        for c in classes:
            log_prob = np.log(class_prior_prob[c])
```

+ Code + Text Connect

```
            class_word_probs = []
            for c in classes:
                log_prob = np.log(class_prior_prob[c])
                for word in row['Text_new']:
                    freq, count = compute_word_freq(word, c, class_f, class_c)
                    log_prob += np.log((freq + 1) / (count + vocab_size))
                class_word_probs.append(log_prob)

            # print(class_word_probs)
            # Predict the class with the highest log probability
            predicted_labels.append(classes[np.argmax(class_word_probs)]) # Pedicted values

    return true_labels, predicted_labels
```

```
# Evaluating Accuracy
def accuracy_evaluation(predValues, truthValues):
  # Sum up the number of correct predictions
    correct_predictions = sum(1 for pred, truth in zip(predValues, truthValues) if pred == truth)
    # Count the total number of predictions made
    total_predictions = len(predValues)
    # Calculate the accuracy as the proportion of correct predictions out of the total predictions
    accuracy = correct_predictions / total_predictions
    return accuracy
```

```
# feature_list: a list of all features in the training data
# feature_dict: a dictionary that stores the top 200 features for each class
# class_feature_freq: a dictionary that stores the frequency of each feature for each class
# class_count: a dictionary that stores the total count of features for each class
feature_list = []
```

```
    return accuracy
```

```
# feature_list: a list of all features in the training data
# feature_dict: a dictionary that stores the top 200 features for each class
# class_feature_freq: a dictionary that stores the frequency of each feature for each class
# class_count: a dictionary that stores the total count of features for each class
feature_list = []
feature_dict = {}
class_feature_freq = {}
class_count = {}

# These lines initialize empty dictionaries and lists that will be used to store
# the features of the training data, as well as the frequency of those features in each class.
for class_name in test_TFICF.keys():
    features = []
    sorted_features = sorted(test_TFICF[class_name], key=test_TFICF[class_name].get, reverse=True)
    for feature_name in sorted_features[0:int(200)]:
        feature_list.append(feature_name)
        features.append(feature_name)
        feature_dict[class_name] = features

for class_name in classes:
  # create a Counter object to count the frequency of each feature in feature_dict
    count_top_features = Counter(feature_dict[class_name])
    for feature_name in feature_list:
        class_feature_freq[class_name, feature_name] = count_top_features[feature_name]
         # store the frequency of the feature for the current class in class_feature_freq
        try:
            class_count[class_name] += count_top_features[feature_name]
        except:
```

```
            class_count[class_name] += count_top_features[feature_name]
        except:
            class_count[class_name] = count_top_features[feature_name]

truth_values, predicted_values = naive_bayes_classifier(distinct_words_count, classes, train_data, test_data, list(classes.keys()), class_feature_
accuracy = accuracy_evaluation(predicted_values, truth_values)
print(str("{:.2f}".format(accuracy * 100)))
```

```
59.06
```

```
# function to calculate confusion matrix
def calculate_confusion_matrix(predictions, ground_truth, classes):
    num_classes = len(classes)
    confusion_matrix = [[0] * num_classes for _ in range(num_classes)]

    for predicted, actual in zip(predictions, ground_truth):
        predicted_index = classes.index(predicted)
        actual_index = classes.index(actual)
        confusion_matrix[predicted_index][actual_index] += 1

    return confusion_matrix
```

```
print("Confusion Matrix: ")
calculate_confusion_matrix(predicted_values, truth_values,list(classes.keys()))
```

```
Confusion Matrix:
[[54, 0, 0, 0, 0],
 [0, 29, 0, 0, 0],
 [0, 0, 44, 0, 0],
 [56, 52, 35, 100, 40],
 [0, 0, 0, 0, 37]]
```

Printed the confusion matrix
Accuracy obtained: 59.06%

5. Improving the classifier

Improving the classification result by more preprocessing like Removing extra spaces, Removing digits, etc.,

```
[0, 29, 0, 0, 0],
[0, 0, 44, 0, 0],
[56, 52, 35, 100, 40],
[0, 0, 0, 0, 37]]
```

(5) Improving the Classifier

```
# Improving the preprocessing
def pre_data(Text):
  # Remove punctuation and replace '_' with space
  text=re.sub(r'[^\w\s]', '', Text).replace("_"," ")
  # remove stopwords
  text=remove_stopwords(text)
  # Remove extra spaces
  text = re.sub(r'\s+', ' ', text)
  # Remove digits
  text = re.sub(r'\d+', '', text)
  # Convert to lowercase
  text=text.lower()
  return text
df2['Text'] = df2['Text'].apply(pre_data)

def tok_lemm(Text):
  wordTokens=tokenize.RegexpTokenizer(r'\w+').tokenize(Text)
  finalTokens=[WordNetLemmatizer().lemmatize(w) for w in wordTokens]
  return finalTokens
df2=df2.drop(['Text_new'], axis=1)
df2['Text'] = df2['Text'].apply(tok_lemm)
df2
```

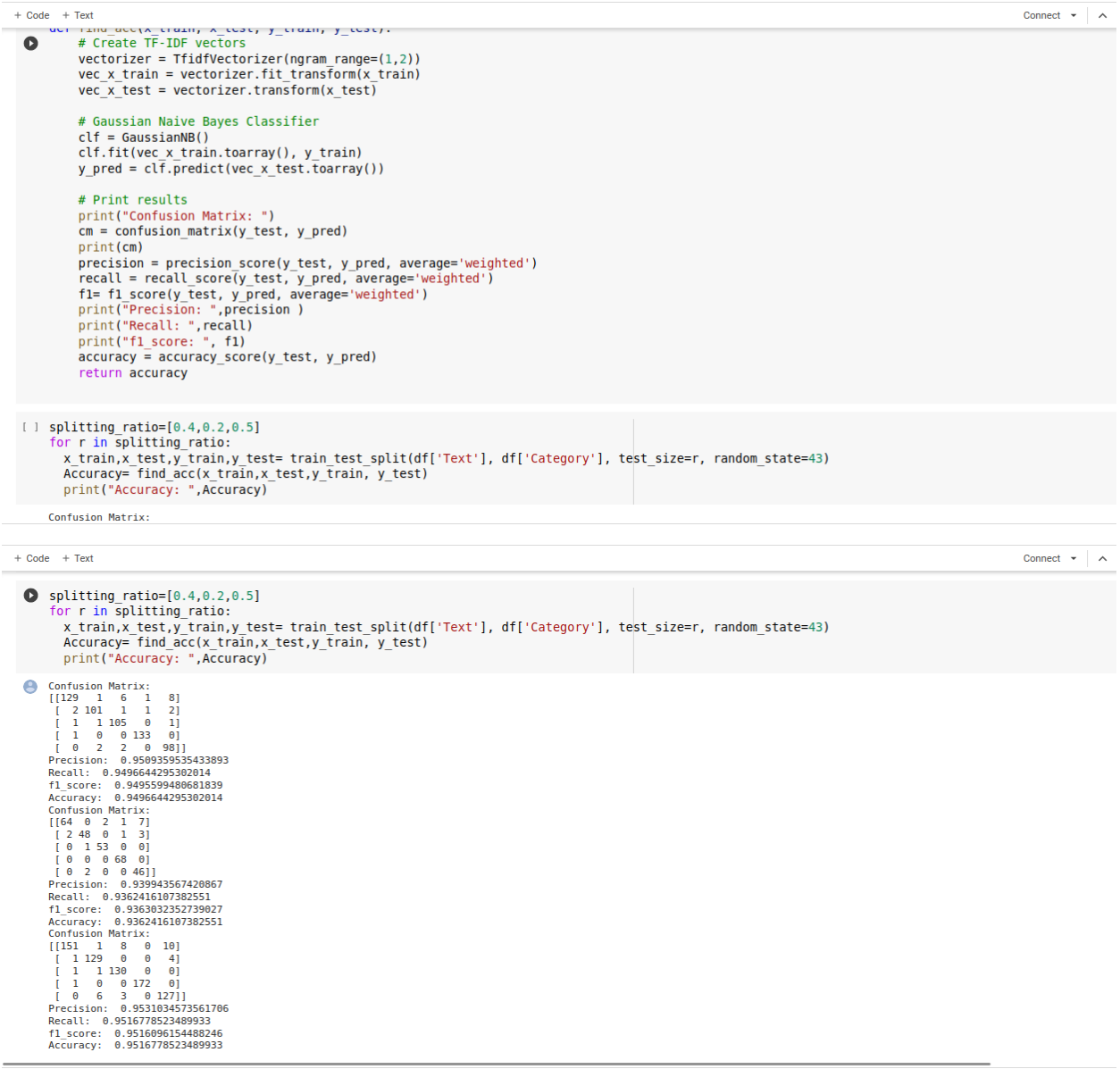
	ArticleId	Text	Category
0	1833	[worldcom, exboss, launch, defence, lawyer, de...	business
1	154	[german, business, confidence, slide, german, ...	business
2	1101	[bbc, poll, indicates, economic, gloom, citize...	business
3	1976	[lifestyle, governs, mobile, choice, faster, b...	tech
4	917	[enron, boss, m, payout, eighteen, enron, dire...	business
...	...	...	...
1485	857	[double, eviction, big, brother, model, capric...	entertainment
1486	325	[dj, double, act, revamp, chart, dj, duo, jk, ...	entertainment
1487	1590	[weak, dollar, hit, reuters, revenue, medium, ...	business
1488	1587	[apple, ipod, family, expands, market, apple, ...	tech
1489	538	[santy, worm, make, unwelcome, visit, thousand...	tech

1490 rows × 3 columns

Find_acc() function is used to find accuracy, precision, recall, and f1_score. For different splitting ratios of train and test data[60-40, 80-20, 50-50] we are doing the following steps:

- Create TF-IDF vectors using ngrams.
- Using the Gaussian Naive Bayes classifier to fitting the dataset and find the predicted values.
- Printing the result.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.naive_bayes import GaussianNB

def find_acc(x_train, x_test, y_train, y_test):
    # Create TF-IDF vectors
    vectorizer = TfidfVectorizer(ngram_range=(1,2))
```



```
    # Create TF-IDF vectors
    vectorizer = TfidfVectorizer(ngram_range=(1,2))
    vec_x_train = vectorizer.fit_transform(x_train)
    vec_x_test = vectorizer.transform(x_test)

    # Gaussian Naive Bayes Classifier
    clf = GaussianNB()
    clf.fit(vec_x_train.toarray(), y_train)
    y_pred = clf.predict(vec_x_test.toarray())

    # Print results
    print("Confusion Matrix: ")
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1= f1_score(y_test, y_pred, average='weighted')
    print("Precision: ",precision )
    print("Recall: ",recall)
    print("f1_score: ", f1)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy
```

```
splitting_ratio=[0.4,0.2,0.5]
for r in splitting_ratio:
  x_train,x_test,y_train,y_test= train_test_split(df['Text'], df['Category'], test_size=r, random_state=43)
  Accuracy= find_acc(x_train,x_test,y_train, y_test)
  print("Accuracy: ",Accuracy)
```

```
Confusion Matrix:
```



```
splitting_ratio=[0.4,0.2,0.5]
for r in splitting_ratio:
  x_train,x_test,y_train,y_test= train_test_split(df['Text'], df['Category'], test_size=r, random_state=43)
  Accuracy= find_acc(x_train,x_test,y_train, y_test)
  print("Accuracy: ",Accuracy)
```

```
Confusion Matrix:
[[129   1   6   1   8]
 [  2 101   1   1   2]
 [  1   1 105   0   1]
 [  1   0   0 133   0]
 [  0   2   2   0  98]]
Precision:  0.9509359535433893
Recall:  0.9496644295302014
f1_score:  0.9495599480681839
Accuracy:  0.9496644295302014
Confusion Matrix:
[[64  0  2  1  7]
 [ 2 48  0  1  3]
 [ 0  1 53  0  0]
 [ 0  0  0 68  0]
 [ 0  2  0  0 46]]
Precision:  0.939943567420867
Recall:  0.9362416107382551
f1_score:  0.9363032352739027
Accuracy:  0.9362416107382551
Confusion Matrix:
[[151   1   8   0  10]
 [  1 129   0   0   4]
 [  1   1 130   0   0]
 [  1   0   0 172   0]
 [  0   6   3   0 127]]
Precision:  0.9531034573561706
Recall:  0.9516778523489933
f1_score:  0.9516096154488246
Accuracy:  0.9516778523489933
```

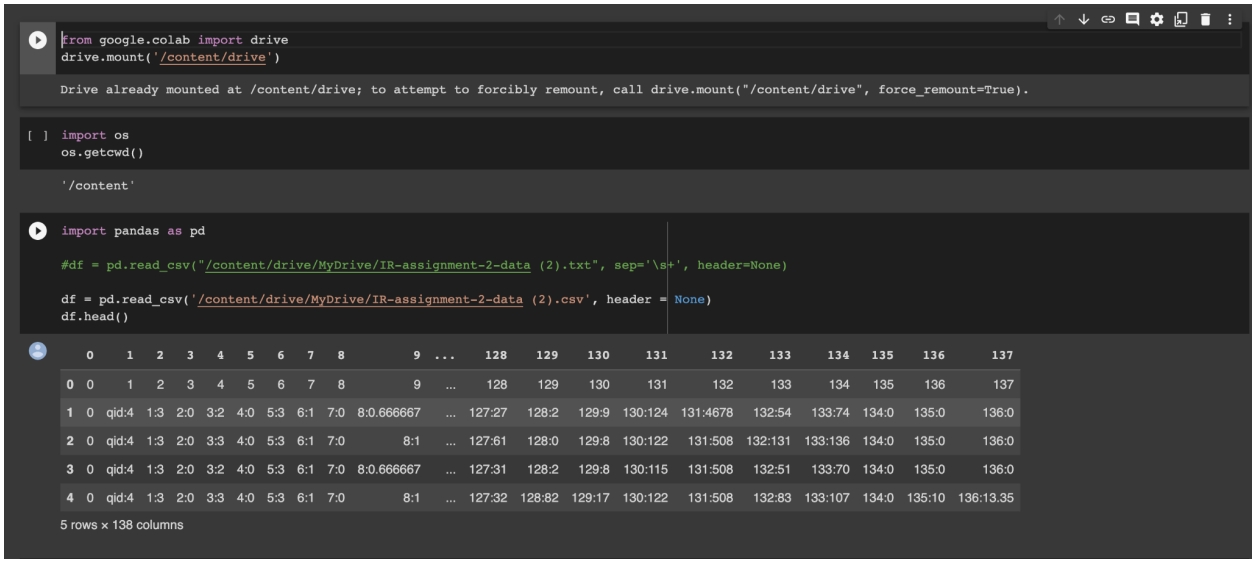
Conclusion: In summary, the utilization of the TF-IDF approach with N-gram and the Gaussian Naive Bayes Classifier, while adjusting the parameters to ngram_range=(1,2) and alpha=0.1, resulted in a significantly higher accuracy rate of 95.16% as compared to the usage of the TF-ICF classifier which only managed to achieve an accuracy rate of 59.06%. These findings suggest that the former approach may be more appropriate for this particular classification task.

SCREENSHOTS FOR THE OUTPUT OF QUES 3

Assumptions:- No assumptions were taken into consideration while performing the below tasks.

Methods and results

The below screenshots shows the loading of the data given in csv format as a pandas dataframe and printing its first five rows.



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
os.getcwd()
```

```
'/content'
```

```
import pandas as pd

#df = pd.read_csv("/content/drive/MyDrive/IR-assignment-2-data (2).txt", sep='\s+', header=None)

df = pd.read_csv('/content/drive/MyDrive/IR-assignment-2-data (2).csv', header = None)
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	128	129	130	131	132	133	134	135	136	137
0	0	1	2	3	4	5	6	7	8	9	...	128	129	130	131	132	133	134	135	136	137
1	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	127:27	128:2	129:9	130:124	131:4678	132:54	133:74	134:0	135:0	136:0
2	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	127:61	128:0	129:8	130:122	131:508	132:131	133:136	134:0	135:0	136:0
3	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	127:31	128:2	129:8	130:115	131:508	132:51	133:70	134:0	135:0	136:0
4	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	127:32	128:82	129:17	130:122	131:508	132:83	133:107	134:0	135:10	136:13.35

5 rows × 138 columns

The below screenshot shows the last 60 rows of the loaded dataset.

```
df.isnull().sum()
df.tail(60)
```

239037	1	qid:29989	1:2	2:0	3:2	4:0	5:2	6:1	7:0	8:1	...	127:40	128:1	129:0	130:53491	131:24053	132:14	133:101	134:0	135:0	13
239038	1	qid:29989	1:2	2:0	3:2	4:2	5:2	6:1	7:0	8:1	...	127:16	128:0	129:0	130:1140	131:100	132:126	133:28	134:0	135:0	13
239039	0	qid:29989	1:2	2:0	3:1	4:1	5:2	6:1	7:0	8:0.50000	...	127:47	128:0	129:0	130:153	131:6214	132:5	133:3	134:0	135:0	13
239040	0	qid:29989	1:2	2:0	3:2	4:0	5:2	6:1	7:0	8:1	...	127:58	128:0	129:0	130:7348	131:40770	132:47	133:23	134:0	135:0	13
239041	1	qid:29989	1:2	2:0	3:2	4:0	5:2	6:1	7:0	8:1	...	127:57	128:0	129:0	130:7549	131:40770	132:10	133:15	134:0	135:0	13
239042	1	qid:29989	1:1	2:0	3:1	4:0	5:1	6:0.50000	7:0	8:0.50000	...	127:16	128:0	129:0	130:40429	131:8509	132:2	133:40	134:0	135:0	13
239043	1	qid:29989	1:2	2:0	3:2	4:1	5:2	6:1	7:0	8:1	...	127:63	128:0	129:1	130:283	131:19587	132:7	133:11	134:0	135:0	13
239044	0	qid:29989	1:2	2:0	3:0	4:0	5:2	6:1	7:0	8:0	...	127:20	128:0	129:0	130:595	131:27	132:9	133:19	134:0	135:4	136:
239045	1	qid:29989	1:0	2:0	3:0	4:0	5:0	6:0	7:0	8:0	...	127:40	128:0	129:2	130:1071	131:49404	132:27	133:98	134:0	135:0	13
239046	0	qid:29989	1:1	2:0	3:0	4:2	5:2	6:0.50000	7:0	8:0	...	127:38	128:1	129:0	130:6057	131:46366	132:238	133:65	134:0	135:0	13
239047	1	qid:29989	1:2	2:0	3:2	4:0	5:2	6:1	7:0	8:1	...	127:57	128:0	129:0	130:560	131:65535	132:1	133:1	134:0	135:0	13
239048	1	qid:29989	1:2	2:0	3:2	4:2	5:2	6:1	7:0	8:1	...	127:70	128:0	129:6	130:1342	131:60198	132:9	133:4	134:0	135:0	13
239049	0	qid:29989	1:1	2:0	3:1	4:1	5:1	6:0.50000	7:0	8:0.50000	...	127:21	128:0	129:0	130:24171	131:1782	132:109	133:23	134:0	135:0	13
239050	0	qid:29989	1:2	2:0	3:2	4:2	5:2	6:1	7:0	8:1	...	127:58	128:20	129:1	130:37218	131:55940	132:33	133:53	134:0	135:0	13
239051	1	qid:29989	1:2	2:0	3:2	4:0	5:2	6:1	7:0	8:1	...	127:26	128:0	129:1	130:32695	131:27219	132:14	133:132	134:0	135:0	13
239052	1	qid:29989	1:2	2:0	3:2	4:0	5:2	6:1	7:0	8:1	...	127:59	128:0	129:0	130:1391	131:65535	132:1	133:1	134:0	135:0	13
239053	0	qid:29989	1:1	2:0	3:1	4:2	5:2	6:0.50000	7:0	8:0.50000	...	127:75	128:7	129:9	130:37444	131:54796	132:3	133:3	134:0	135:0	13
239054	1	qid:29989	1:1	2:0	3:1	4:2	5:2	6:0.50000	7:0	8:0.50000	...	127:75	128:0	129:9	130:277	131:54796	132:6	133:3	134:0	135:0	13
239055	2	qid:29989	1:2	2:0	3:1	4:0	5:2	6:1	7:0	8:0.50000	...	127:34	128:0	129:0	130:4781	131:27150	132:29	133:1	134:0	135:0	13

The below screenshot shows the code for filtering out the query that has “qid:4” and storing them in a new dataframe.

```
#The queries with qid:4 are extracted out from the dataset
df3=pd.DataFrame(columns=df.columns)
j=0;
for i in range(0,239093):
  if (df.iloc[i][1] == 'qid:4'):
      df3.loc[j]=df.iloc[i]
      j+=1
```

```
[ ] df3.shape
```

```
(103, 138)
```

```
df3.head(20)
```

	0	1	2	3	4	5	6	7	8	9	...	128	129	130	131	132	133	134	135	136	137
0	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	127:27	128:2	129:9	130:124	131:4678	132:54	133:74	134:0	135:0	136:0
1	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	127:61	128:0	129:8	130:122	131:508	132:131	133:136	134:0	135:0	136:0
2	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	127:31	128:2	129:8	130:115	131:508	132:51	133:70	134:0	135:0	136:0
3	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	127:32	128:82	129:17	130:122	131:508	132:83	133:107	134:0	135:10	136:13.35
4	1	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	127:29	128:11	129:8	130:121	131:508	132:103	133:120	134:0	135:0	136:0
5	0	qid:4	1:3	2:1	3:2	4:1	5:3	6:1	7:0.333333	8:0.666667	...	127:45	128:82	129:7	130:123	131:4262	132:47	133:79	134:0	135:4	136:6.68
6	1	qid:4	1:3	2:0	3:1	4:1	5:3	6:1	7:0	8:0.333333	...	127:46	128:22	129:2	130:256	131:9102	132:28	133:23	134:0	135:0	136:0
7	3	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:32	128:349	129:8	130:123	131:281	132:22	133:6	134:0	135:0	136:0
8	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	127:51	128:766	129:0	130:119	131:32560	132:45	133:24	134:0	135:0	136:0
9	0	qid:4	1:3	2:0	3:3	4:0	5:3	6:1	7:0	8:1	...	127:33	128:367	129:8	130:266	131:398	132:89	133:114	134:0	135:0	136:0

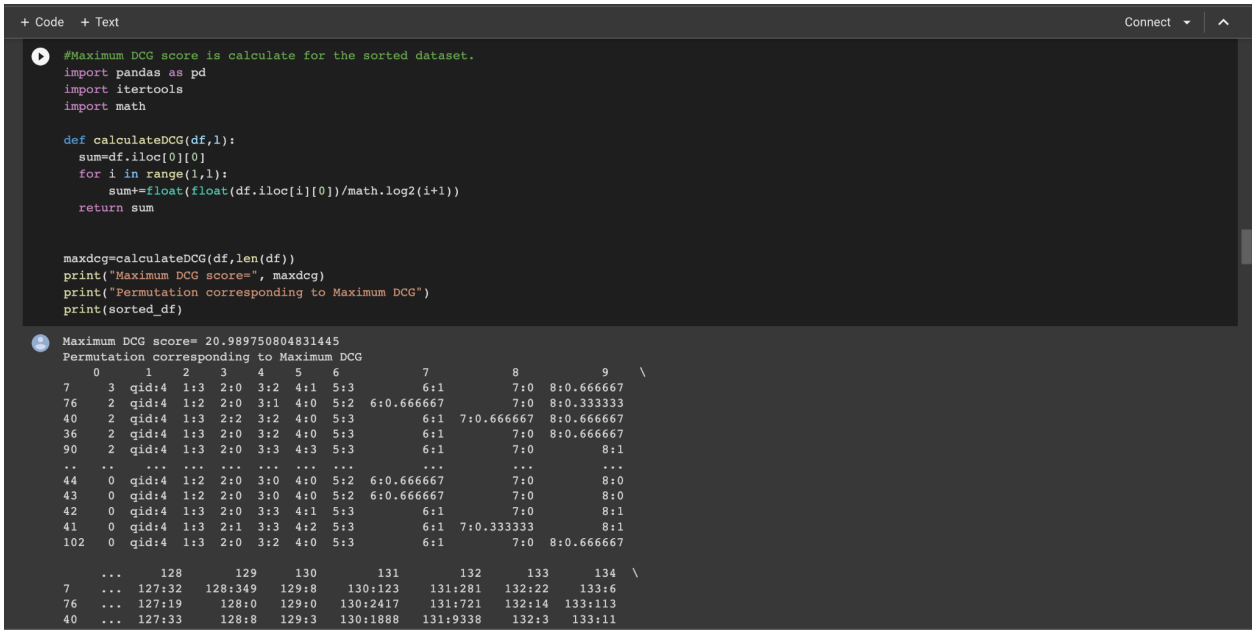
This screenshot shows the code that the above data frame consisting only “qid:4” are sorted on the basis of their relevance score and printing its first 20 rows.

```
+ Code  + Text                                                    Connect
```

```
sorted_df = df3.sort_values(by=df3.columns[0], ascending=False)
df=sorted_df
df.head(20)
```

	0	1	2	3	4	5	6	7	8	9	...	128	129	130	131	132	133	134	135	136	137
7	3	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:32	128:349	129:8	130:123	131:281	132:22	133:6	134:0	135:0	136:0
76	2	qid:4	1:2	2:0	3:1	4:0	5:2	6:0.666667	7:0	8:0.333333	...	127:19	128:0	129:0	130:2417	131:721	132:14	133:113	134:0	135:13	136:47.9
40	2	qid:4	1:3	2:2	3:2	4:0	5:3	6:1	7:0.666667	8:0.666667	...	127:33	128:8	129:3	130:1888	131:9338	132:3	133:11	134:0	135:0	136:0
36	2	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	127:17	128:0	129:2	130:12028	131:11379	132:26	133:24	134:0	135:77	136:23.9595223404047
90	2	qid:4	1:3	2:0	3:3	4:3	5:3	6:1	7:0	8:1	...	127:67	128:27	129:0	130:814	131:13555	132:108	133:113	134:0	135:0	136:0
25	2	qid:4	1:3	2:0	3:3	4:1	5:3	6:1	7:0	8:1	...	127:52	128:2664	129:0	130:5753	131:11746	132:8	133:68	134:0	135:0	136:0
37	2	qid:4	1:2	2:0	3:2	4:0	5:2	6:0.666667	7:0	8:0.666667	...	127:23	128:0	129:0	130:16417	131:9338	132:29	133:29	134:6	135:68	136:28.1902038750723
22	2	qid:4	1:3	2:1	3:3	4:0	5:3	6:1	7:0.333333	8:1	...	127:59	128:189	129:8	130:144	131:4307	132:82	133:108	134:0	135:0	136:0
21	2	qid:4	1:3	2:1	3:3	4:2	5:3	6:1	7:0.333333	8:1	...	127:67	128:8	129:5	130:144	131:395	132:13	133:56	134:0	135:0	136:0
19	2	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:49	128:553	129:2	130:876	131:10008	132:42	133:45	134:0	135:0	136:0
18	2	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:60	128:0	129:5	130:790	131:4744	132:18	133:60	134:0	135:0	136:0
34	2	qid:4	1:3	2:1	3:3	4:2	5:3	6:1	7:0.333333	8:1	...	127:79	128:1901	129:0	130:5555	131:6578	132:6	133:94	134:0	135:0	136:0
58	2	qid:4	1:2	2:1	3:2	4:2	5:3	6:0.666667	7:0.333333	8:0.666667	...	127:67	128:367	129:6	130:153	131:2542	132:20	133:32	134:0	135:0	136:0
68	2	qid:4	1:3	2:1	3:2	4:1	5:3	6:1	7:0.333333	8:0.666667	...	127:54	128:872	129:4	130:8346	131:4861	132:10	133:55	134:0	135:0	136:0
62	2	qid:4	1:3	2:0	3:3	4:1	5:3	6:1	7:0	8:1	...	127:87	128:89	129:7	130:153	131:2650	132:105	133:118	134:0	135:6	136:15.3
100	2	qid:4	1:2	2:0	3:2	4:0	5:2	6:0.666667	7:0	8:0.666667	...	127:28	128:0	129:0	130:49182	131:26966	132:15	133:69	134:0	135:193	136:21.9355595468361
52	2	qid:4	1:2	2:0	3:2	4:0	5:2	6:0.666667	7:0	8:0.666667	...	127:21	128:1067	129:1	130:6930	131:8855	132:4	133:95	134:0	135:0	136:0

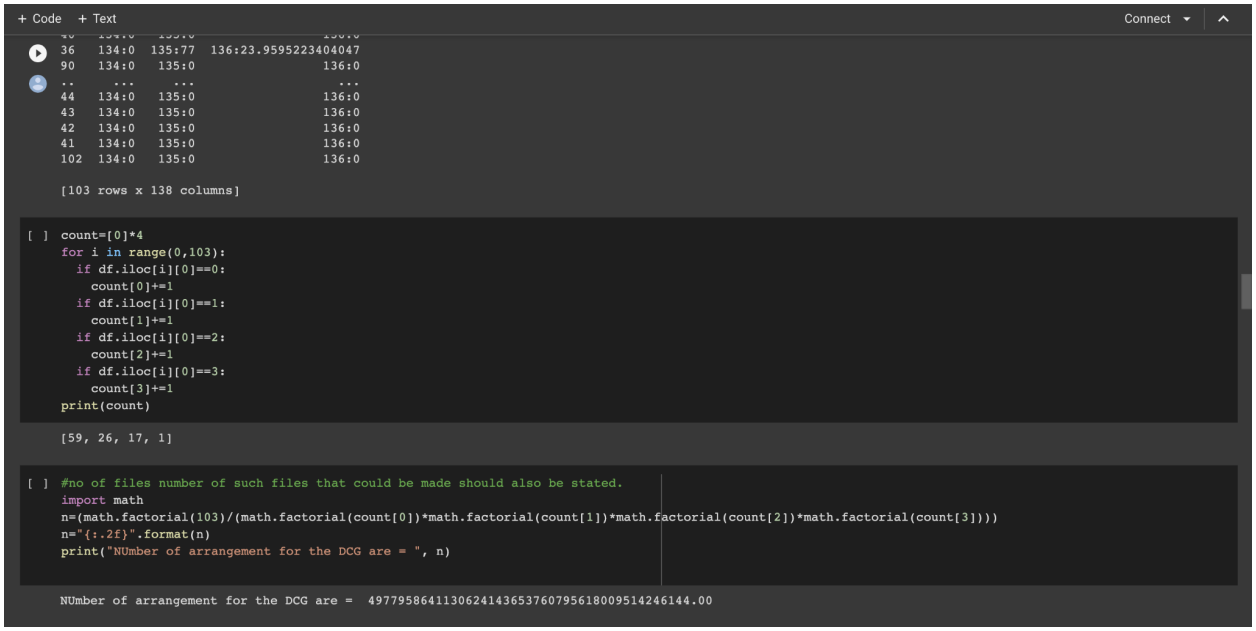
The below screenshot shows the code for the code to calculate DCG for n the data row. Here this function is called on the relevance score sorted dataset so that we get maximum DCG score.



```
#Maximum DCG score is calculate for the sorted dataset.
import pandas as pd
import itertools
import math

def calculateDCG(df,l):
  sum=df.iloc[0][0]
  for i in range(1,l):
      sum+=float(float(df.iloc[i][0])/math.log2(i+1))
  return sum


maxdcg=calculateDCG(df,len(df))
print("Maximum DCG score=", maxdcg)
print("Permutation corresponding to Maximum DCG")
print(sorted_df)
```

```
Maximum DCG score= 20.989750804831445
Permutation corresponding to Maximum DCG
     0      1    2    3    4    5    6           7           8           9    \
7     3  qid:4  1:3  2:0  3:2  4:1  5:3         6:1         7:0  8:0.666667
76    2  qid:4  1:2  2:0  3:1  4:0  5:2  6:0.666667         7:0  8:0.333333
40    2  qid:4  1:3  2:2  3:2  4:0  5:3         6:1  7:0.666667  8:0.666667
36    2  qid:4  1:3  2:0  3:2  4:0  5:3         6:1         7:0  8:0.666667
90    2  qid:4  1:3  2:0  3:3  4:3  5:3         6:1         7:0         8:1
..   ..     ...  ...  ...  ...  ...  ...         ...         ...         ...
44    0  qid:4  1:2  2:0  3:0  4:0  5:2  6:0.666667         7:0         8:0
43    0  qid:4  1:2  2:0  3:0  4:0  5:2  6:0.666667         7:0         8:0
42    0  qid:4  1:3  2:0  3:3  4:1  5:3         6:1         7:0         8:1
41    0  qid:4  1:3  2:1  3:3  4:2  5:3         6:1  7:0.333333         8:1
102   0  qid:4  1:3  2:0  3:2  4:0  5:3         6:1         7:0  8:0.666667

      ...     128       129     130        131        132      133      134  \
7     ...  127:32   128:349   129:8    130:123    131:281   132:22    133:6
76    ...  127:19     128:0   129:0   130:2417    131:721   132:14  133:113
40    ...  127:33     128:8   129:3   130:1888   131:9338    132:3   133:11
```

The below screenshot shows the code that shows how many entries have relevance scores equal to 0, 1,2 and 3. After that using these counts we have calculated the number of permutations of this dataset for the different DCG score score value.



```
36   134:0  135:77  136:23.9595223404047
90   134:0   135:0                136:0
..     ...     ...                  ...
44   134:0   135:0                136:0
43   134:0   135:0                136:0
42   134:0   135:0                136:0
41   134:0   135:0                136:0
102  134:0   135:0                136:0

[103 rows x 138 columns]
```

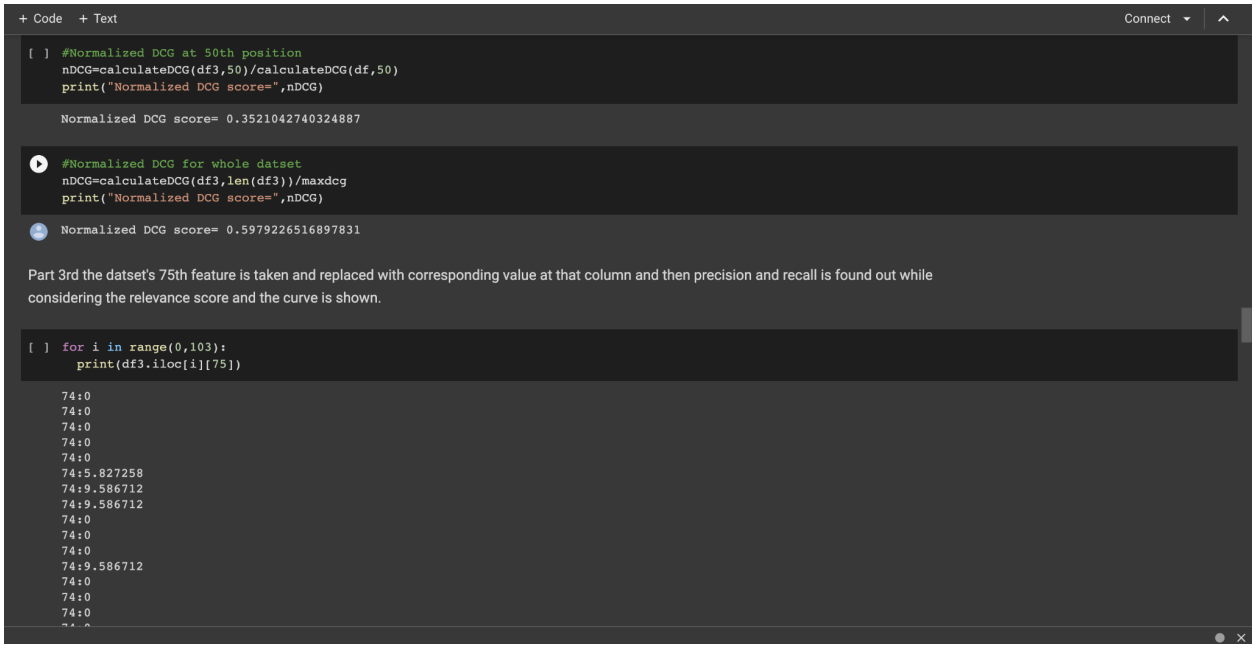
```
count=[0]*4
for i in range(0,103):
  if df.iloc[i][0]==0:
    count[0]+=1
  if df.iloc[i][0]==1:
    count[1]+=1
  if df.iloc[i][0]==2:
    count[2]+=1
  if df.iloc[i][0]==3:
    count[3]+=1
print(count)
```

```
[59, 26, 17, 1]
```

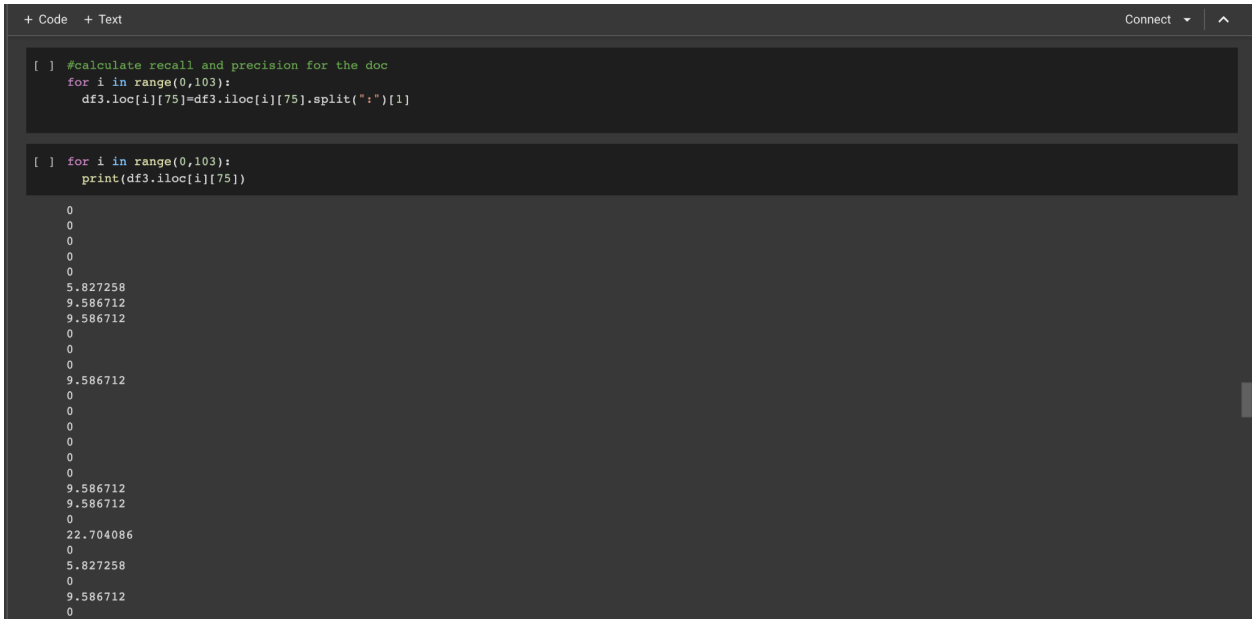
```
#no of files number of such files that could be made should also be stated.
import math
n=(math.factorial(103)/(math.factorial(count[0])*math.factorial(count[1])*math.factorial(count[2])*math.factorial(count[3])))
n="{:.2f}".format(n)
print("NUmber of arrangement for the DCG are = ", n)
```

```
NUmber of arrangement for the DCG are =  49779586411306241436537607956180095142461 44.00
```

The below screenshot shows the code to calculate the DCG score value at the position 50 and for the entire dataset and the dividing that value by respective Ideal DCG score by calling the function implemented above to calculate DCG score of n values. After that printing the values of the dataframe at column 75.



Here, we are splitting the value of the dataframe at column 75 and storing the part at position 1 after splitting and printing those values.



In the below screenshot the dataframe rows are sorted again but now on the values of the dataframe at column 75.

```
[ ] sorted_df3 = df3.sort_values(by=[75], ascending=False)
    sorted_df3.head(20)
```

	0	1	2	3	4	5	6	7	8	9	...	128	129	130	131	132	133	134	135	136	137
42	0	qid:4	1:3	2:0	3:3	4:1	5:3	6:1	7:0	8:1	...	127:65	128:83	129:5	130:144	131:262	132:157	133:179	134:0	135:0	136:0
11	0	qid:4	1:2	2:1	3:1	4:1	5:2	6:0.666667	7:0.333333	8:0.333333	...	127:85	128:4972	129:1	130:25395	131:21052	132:2	133:7	134:0	135:0	136:0
25	2	qid:4	1:3	2:0	3:3	4:1	5:3	6:1	7:0	8:1	...	127:52	128:2664	129:0	130:5753	131:11746	132:8	133:68	134:0	135:0	136:0
50	0	qid:4	1:3	2:0	3:1	4:1	5:3	6:1	7:0	8:0.333333	...	127:39	128:10	129:2	130:802	131:11439	132:17	133:18	134:0	135:0	136:0
19	2	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:49	128:553	129:2	130:876	131:10008	132:42	133:45	134:0	135:0	136:0
18	2	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:60	128:0	129:5	130:790	131:4744	132:18	133:60	134:0	135:0	136:0
68	2	qid:4	1:3	2:1	3:2	4:1	5:3	6:1	7:0.333333	8:0.666667	...	127:54	128:872	129:4	130:8346	131:4861	132:10	133:55	134:0	135:0	136:0
48	0	qid:4	1:3	2:0	3:3	4:1	5:3	6:1	7:0	8:1	...	127:67	128:23	129:7	130:277	131:2635	132:134	133:136	134:0	135:1	136:1.6
82	1	qid:4	1:3	2:0	3:1	4:1	5:3	6:1	7:0	8:0.333333	...	127:44	128:34	129:2	130:1093	131:11103	132:24	133:30	134:0	135:0	136:0
62	2	qid:4	1:3	2:0	3:3	4:1	5:3	6:1	7:0	8:1	...	127:87	128:89	129:7	130:153	131:2650	132:105	133:118	134:0	135:6	136:15.3
7	3	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:32	128:349	129:8	130:123	131:281	132:22	133:6	134:0	135:0	136:0
6	1	qid:4	1:3	2:0	3:1	4:1	5:3	6:1	7:0	8:0.333333	...	127:46	128:22	129:2	130:256	131:9102	132:28	133:23	134:0	135:0	136:0
46	0	qid:4	1:3	2:0	3:3	4:1	5:3	6:1	7:0	8:1	...	127:53	128:141	129:8	130:277	131:2635	132:92	133:114	134:0	135:0	136:0
71	0	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:42	128:392	129:2	130:11488	131:12941	132:32	133:174	134:0	135:0	136:0
45	0	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:42	128:390	129:3	130:11156	131:12901	132:23	133:148	134:0	135:9	136:12.7
23	0	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:61	128:9	129:5	130:1856	131:12003	132:24	133:97	134:0	135:0	136:0
5	0	qid:4	1:3	2:1	3:2	4:1	5:3	6:1	7:0.333333	8:0.666667	...	127:45	128:82	129:7	130:123	131:4262	132:47	133:79	134:0	135:4	136:6.68
69	0	qid:4	1:3	2:0	3:2	4:1	5:3	6:1	7:0	8:0.666667	...	127:30	128:0	129:3	130:33146	131:12941	132:15	133:162	134:0	135:88	136:25.8698773448774

Here, we have calculated the values for recall and precision for each of the positions based on the fact that the respective document is relevant or not which depends on its relevance score.

```
+ Code + Text                                                    Connect

sert code cell below
⌘/Ctrl+M B
    l_relevant=0
    for i in range(0,103):
      if float(sorted_df3.loc[i][75])==0:
        continue
      else:
        total_relevant+=1
    print(total_relevant)
```

```
36
```

```
[ ] retrieved_doc=1
    relevant_retrived=0
    precision=[]
    recall=[]
    for i in range(0,103):
      if int(sorted_df3.iloc[i][0])>0:
        relevant_retrived+=1
      recall.append(relevant_retrived/total_relevant)
      precision.append(relevant_retrived/retrieved_doc)
      retrieved_doc+=1
```

```
[ ] print(recall)
```

```
[0.0, 0.0, 0.027777777777777776, 0.027777777777777776, 0.05555555555555555, 0.08333333333333333, 0.1111111111111111, 0.1111111111111111, 0.1388888888888889,
```

```
[ ] print(precision)
```

```
[0.0, 0.0, 0.3333333333333333, 0.25, 0.4, 0.5, 0.5714285714285714, 0.5, 0.5555555555555556, 0.6, 0.6363636363636364, 0.6666666666666666, 0.6153846153846154,
```

In the below screenshot, a graph is shown between the recall and precision values calculated above.

From this graph, we made the following observation that the value of precision increases with increase in recall till a certain value after that the precision remains constant even when there is increase in the value of recall.