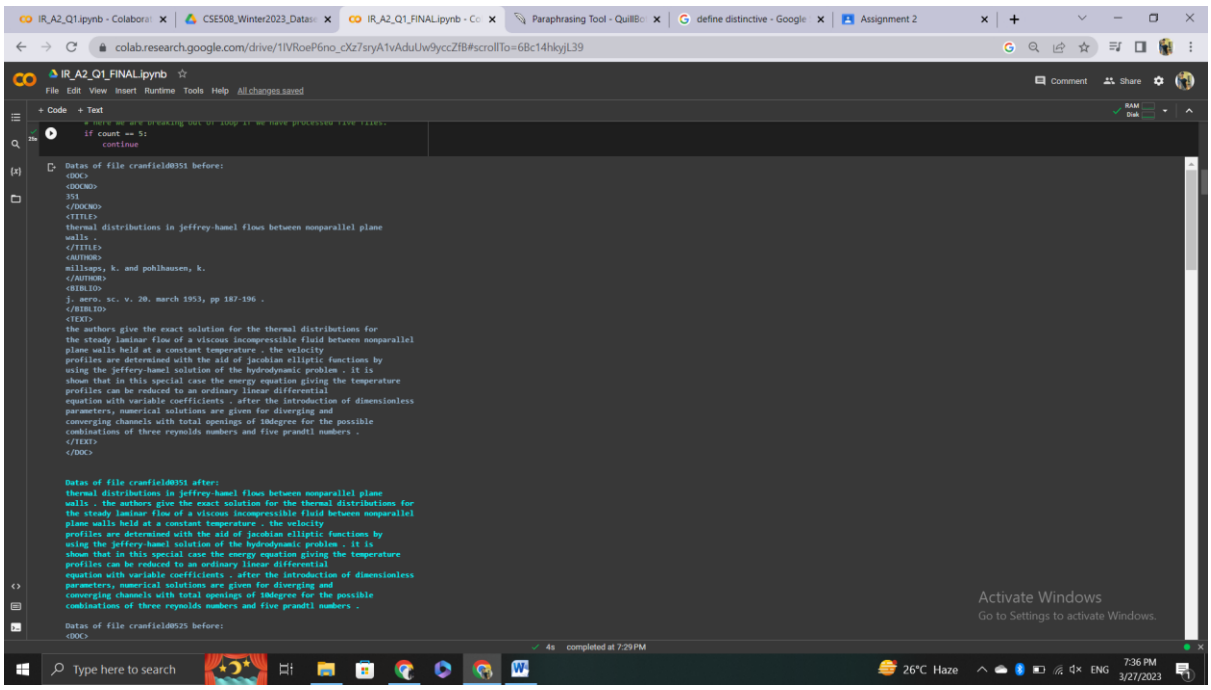
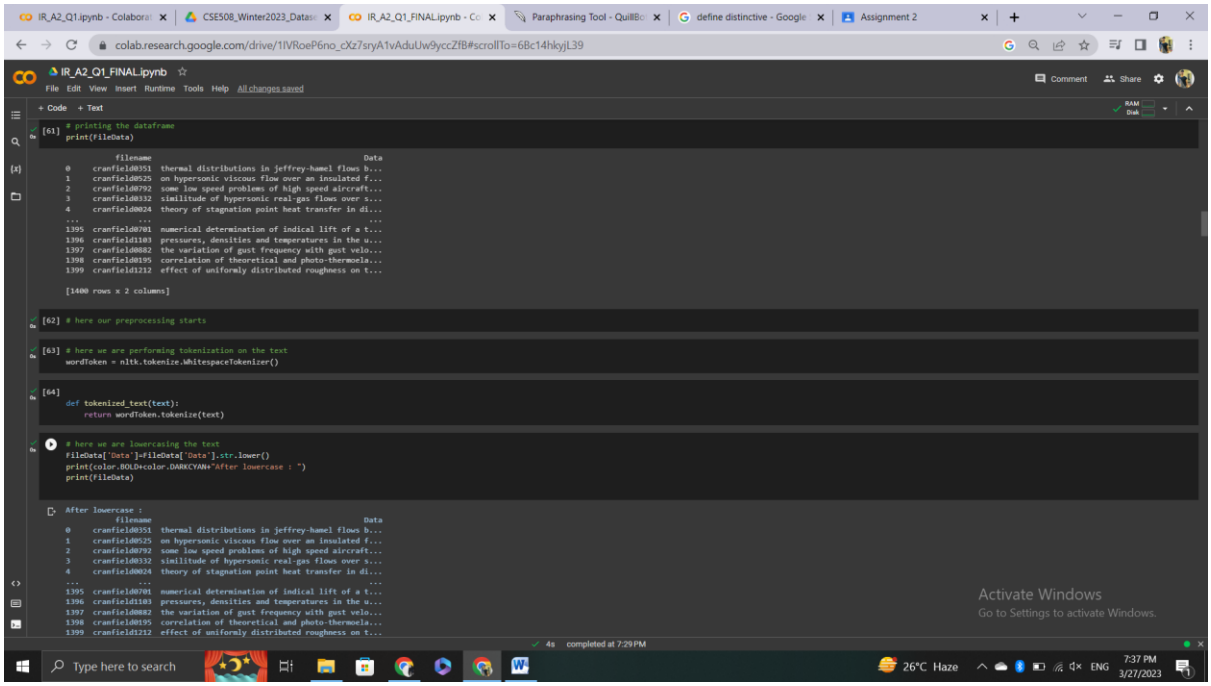
Information Retrieval
Assignment-2
Atishay Jain (MT22021)
Rishika Jain (MT22055)

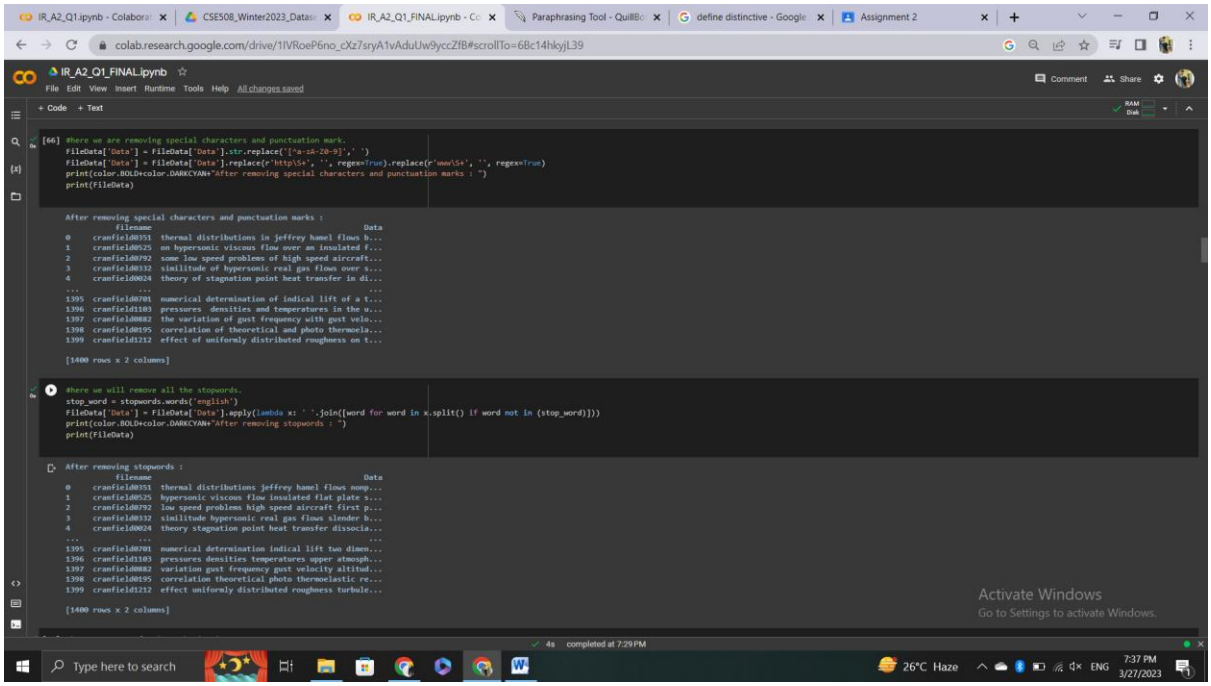
Question 1

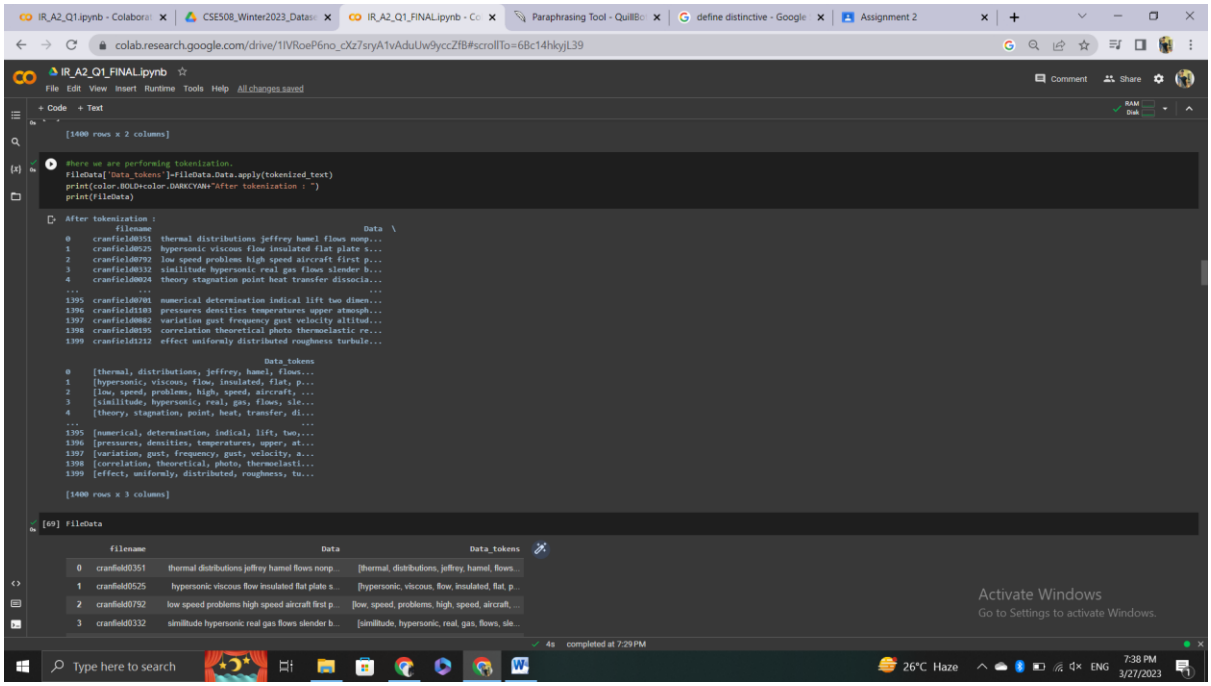
For this question, we have used the following python libraries:
- Pandas, Numpy.
- NLTK library for pre-processing the data.

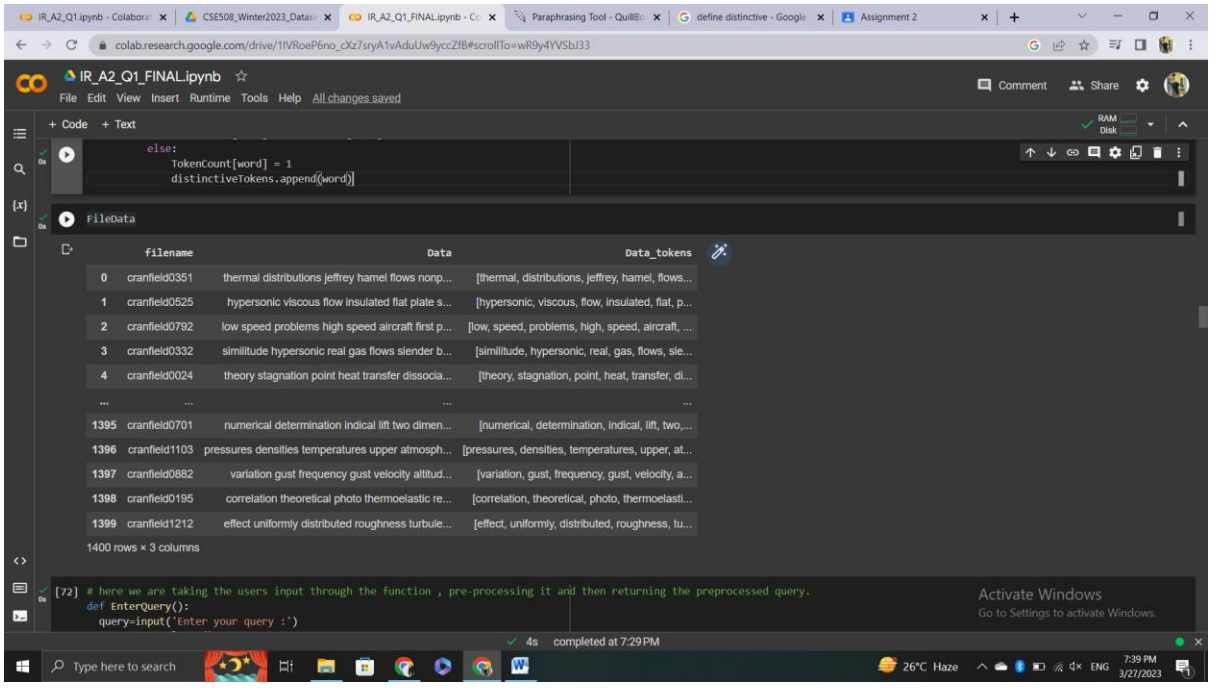
Pre-Processing:
1. We have used the os.listdir() function to read data files from the directory.
2. Read the contents of every file when it was opened, then put it in a dataframe.
3. Changed every word to lower case.
4. Eliminated all punctuation and special letters.
5. Using NLTK stopwords, all stopwords were eliminated from the corpus.
6. Tokenization of the document corpus was done.




IR_A2_Q1.ipynb - Colabora
CSE508_Winter2023_Datas
IR_A2_Q1_FINAL.ipynb - Co
Paraphrasing Tool - QuillBo
define distinctive - Google
Assignment 2
colab.research.google.com/drive/1IVRoeP6no_cXz7sryA1vAduUw9yccZfB#scrollTo=6Bc14hkyjL39
IR_A2_Q1_FINAL.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
[61] # printing the dataframe
print(FileData)
filename Data
0 cranfield0351 thermal distributions in jeffrey-hamel flows b...
1 cranfield0525 on hypersonic viscous flow over an insulated f...
2 cranfield0792 some low speed problems of high speed aircraft...
3 cranfield0332 similitude of hypersonic real-gas flows over s...
4 cranfield0024 theory of stagnation point heat transfer in di...
... ...
1395 cranfield0701 numerical determination of indical lift of a t...
1396 cranfield1103 pressures, densities and temperatures in the u...
1397 cranfield0882 the variation of gust frequency with gust velo...
1398 cranfield0195 correlation of theoretical and photo-thermoela...
1399 cranfield1212 effect of uniformly distributed roughness on t...
[1400 rows x 2 columns]
[62] # here our preprocessing starts
[63] # here we are performing tokenization on the text
wordToken = nltk.tokenize.WhitespaceTokenizer()
[64]
def tokenized_text(text):
return wordToken.tokenize(text)
here we are lowercasing the text
FileData['Data']=FileData['Data'].str.lower()
print(color.BOLD+color.DARKCYAN+"After lowercase : ")
print(FileData)
After lowercase :
filename Data
0 cranfield0351 thermal distributions in jeffrey-hamel flows b...
1 cranfield0525 on hypersonic viscous flow over an insulated f...
2 cranfield0792 some low speed problems of high speed aircraft...
3 cranfield0332 similitude of hypersonic real-gas flows over s...
4 cranfield0024 theory of stagnation point heat transfer in di...
... ...
1395 cranfield0701 numerical determination of indical lift of a t...
1396 cranfield1103 pressures, densities and temperatures in the u...
1397 cranfield0882 the variation of gust frequency with gust velo...
1398 cranfield0195 correlation of theoretical and photo-thermoela...
1399 cranfield1212 effect of uniformly distributed roughness on t...
Activate Windows
Go to Settings to activate Windows.
4s completed at 7:29 PM
Type here to search
26°C Haze
ENG
7:37 PM
3/27/2023


IR_A2_Q1.ipynb - Colabora
CSE508_Winter2023_Datas
IR_A2_Q1_FINAL.ipynb - Co
Paraphrasing Tool - QuillBo
define distinctive - Google
Assignment 2
colab.research.google.com/drive/1IVRoeP6no_cXz7sryA1vAduUw9yccZfB#scrollTo=6Bc14hkyjL39
IR_A2_Q1_FINAL.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
[66] #here we are removing special characters and punctuation mark:
FileData['Data'] = FileData['Data'].str.replace('[^a-zA-Z0-9]',' ')
FileData['Data'] = FileData['Data'].replace(r'http\S+', '', regex=True).replace(r'www\S+', '', regex=True)
print(color.BOLD+color.DARKCYAN+"After removing special characters and punctuation marks : ")
print(FileData)
After removing special characters and punctuation marks :
filename Data
0 cranfield0351 thermal distributions in jeffrey hamel flows b...
1 cranfield0525 on hypersonic viscous flow over an insulated f...
2 cranfield0792 some low speed problems of high speed aircraft...
3 cranfield0332 similitude of hypersonic real gas flows over s...
4 cranfield0024 theory of stagnation point heat transfer in di...
... ...
1395 cranfield0701 numerical determination of indical lift of a t...
1396 cranfield1103 pressures densities and temperatures in the u...
1397 cranfield0882 the variation of gust frequency with gust velo...
1398 cranfield0195 correlation of theoretical and photo thermoela...
1399 cranfield1212 effect of uniformly distributed roughness on t...
[1400 rows x 2 columns]
#here we will remove all the stopwords.
stop_word = stopwords.words('english')
FileData['Data'] = FileData['Data'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop_word)]))
print(color.BOLD+color.DARKCYAN+"After removing stopwords : ")
print(FileData)
After removing stopwords :
filename Data
0 cranfield0351 thermal distributions jeffrey hamel flows nonp...
1 cranfield0525 hypersonic viscous flow insulated flat plate s...
2 cranfield0792 low speed problems high speed aircraft first p...
3 cranfield0332 similitude hypersonic real gas flows slender b...
4 cranfield0024 theory stagnation point heat transfer dissocia...
... ...
1395 cranfield0701 numerical determination indical lift two dimen...
1396 cranfield1103 pressures densities temperatures upper atmosph...
1397 cranfield0882 variation gust frequency gust velocity altitud...
1398 cranfield0195 correlation theoretical photo thermoelastic re...
1399 cranfield1212 effect uniformly distributed roughness turbule...
[1400 rows x 2 columns]
Activate Windows
Go to Settings to activate Windows.
4s completed at 7:29 PM
Type here to search
26°C Haze
ENG
7:37 PM
3/27/2023





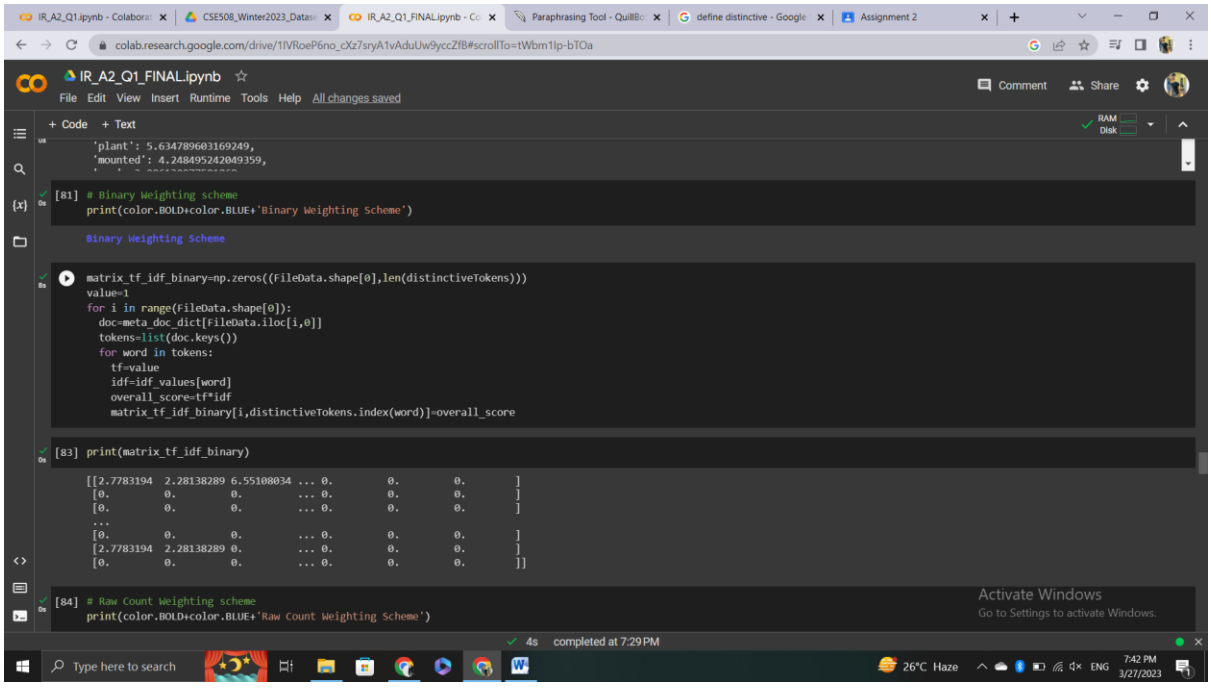
METHODOLOGY:

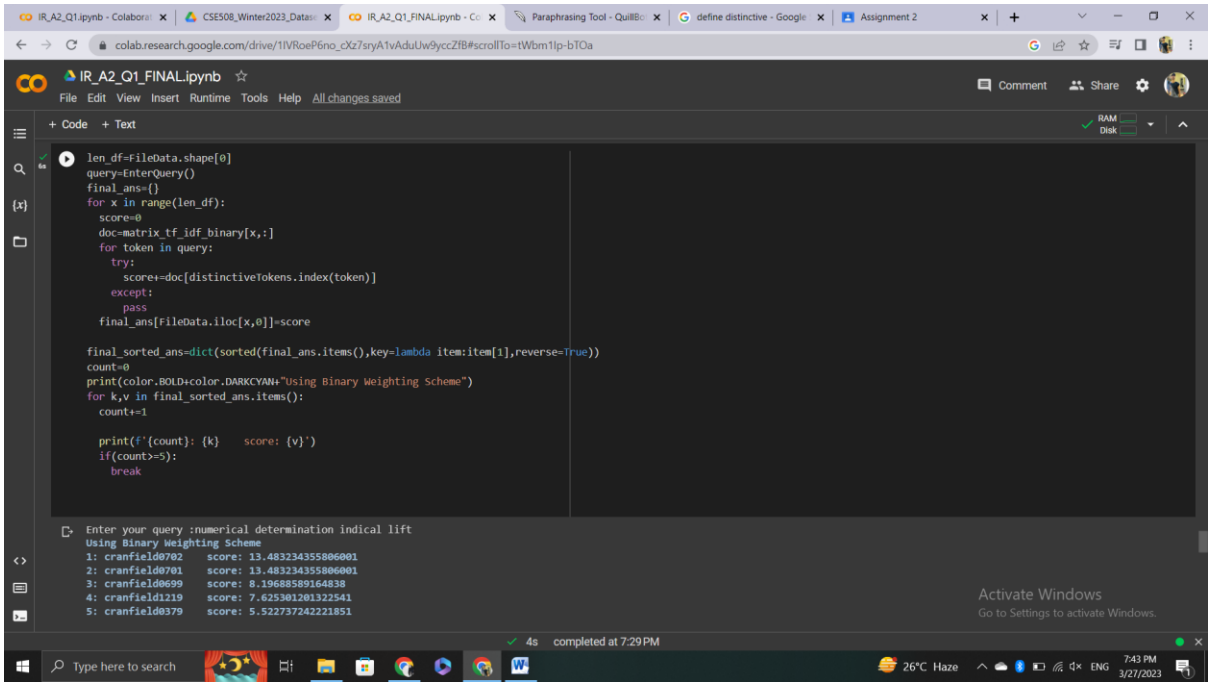
A. TF-IDF Matrix:

For each document, we generated a dictionary that lists the terms and the number of words that are used in that document.
After that, we determined the IDF score for each distinct word in the corpus.
IDF score is equal to the log of the total number of files divided by the length of the posting list for that word.

TF-IDF matrix for every Five scoring schemes:-

❖ **BINARY:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."
When a token is included in the document, the TF value is 1, otherwise it is 0.
Score for TF-IDF: tf * idf

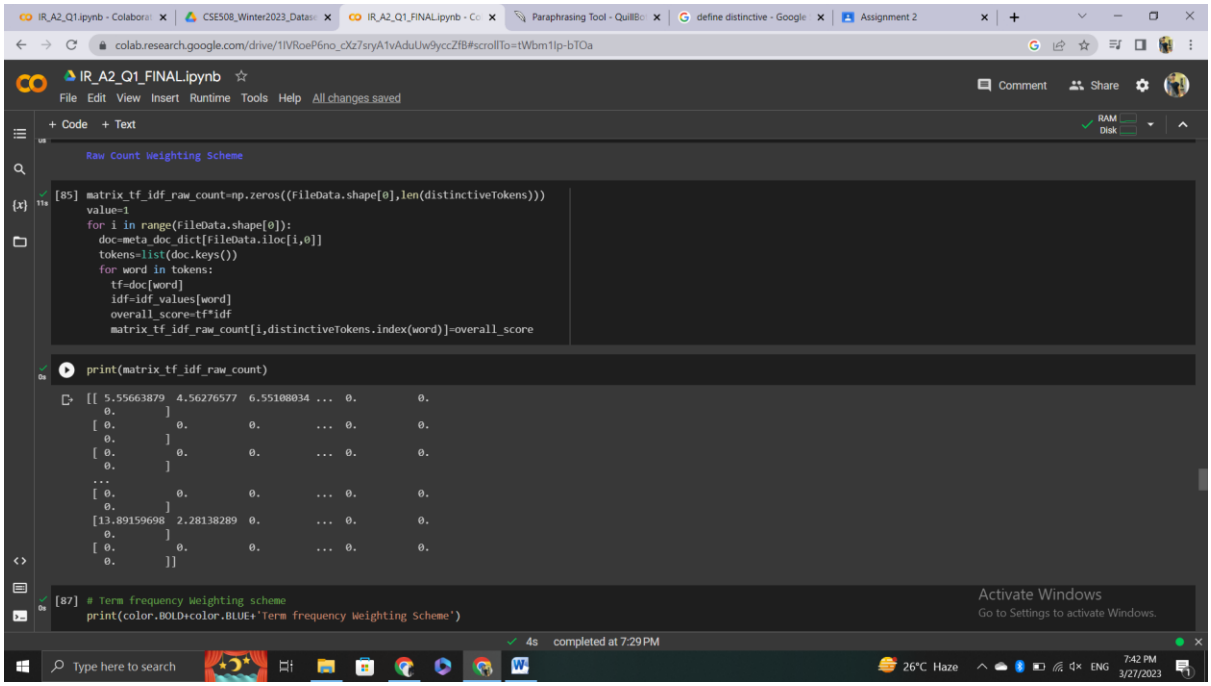




```
len_df=FileData.shape[0]
query=EnterQuery()
final_ans={}
for x in range(len_df):
  score=0
  doc=matrix_tf_idf_binary[x,:]
  for token in query:
    try:
      score+=doc[distinctiveTokens.index(token)]
    except:
      pass
  final_ans[FileData.iloc[x,0]]=score

final_sorted_ans=dict(sorted(final_ans.items(),key=lambda item:item[1],reverse=True))
count=0
print(color.BOLD+color.DARKCYAN+"Using Binary Weighting Scheme")
for k,v in final_sorted_ans.items():
  count+=1

  print(f'{count}: {k}    score: {v}')
  if(count>=5):
    break
```

```
Enter your query :numerical determination indical lift
Using Binary Weighting Scheme
1: cranfield0702    score: 13.483234355806001
2: cranfield0701    score: 13.483234355806001
3: cranfield0699    score: 8.19688589164838
4: cranfield1219    score: 7.625301201322541
5: cranfield0379    score: 5.522737242221851
```

❖ **RAW COUNT:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."
TF value is stored in the dictionary .
Score for TF-IDF: tf * idf

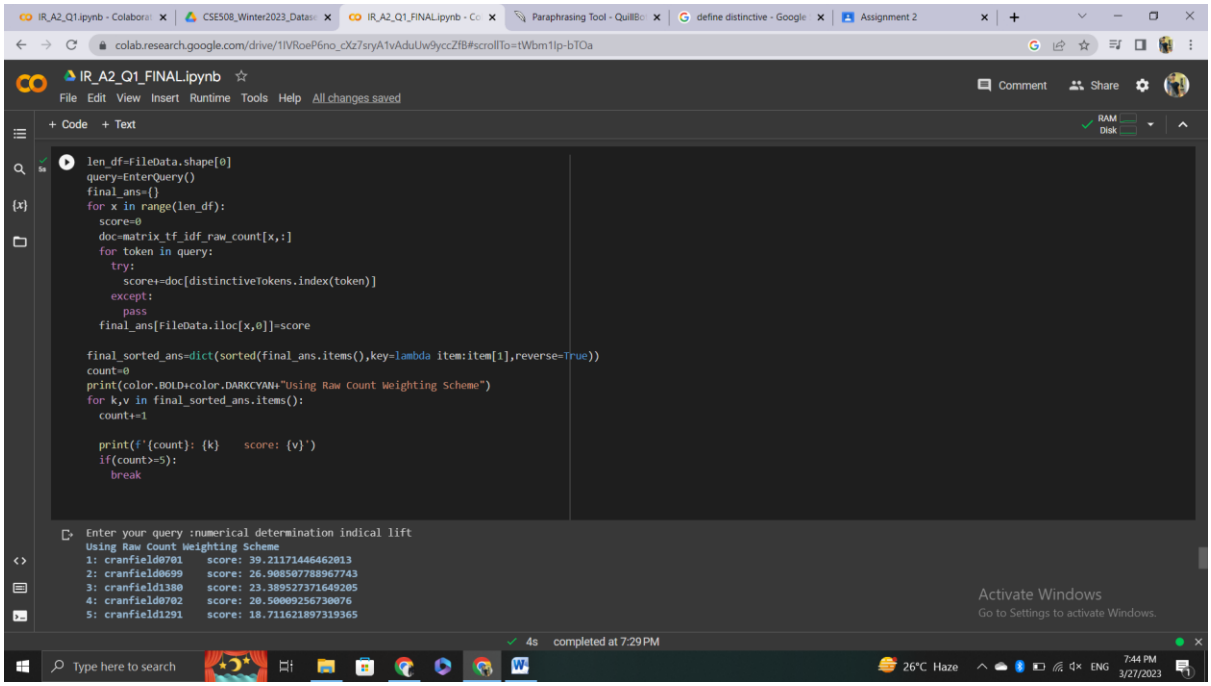


```
Raw Count Weighting Scheme

matrix_tf_idf_raw_count=np.zeros((FileData.shape[0],len(distinctiveTokens)))
value=1
for i in range(FileData.shape[0]):
  doc=meta_doc_dict[FileData.iloc[i,0]]
  tokens=list(doc.keys())
  for word in tokens:
    tf=doc[word]
    idf=idf_values[word]
    overall_score=tf*idf
    matrix_tf_idf_raw_count[i,distinctiveTokens.index(word)]=overall_score
```

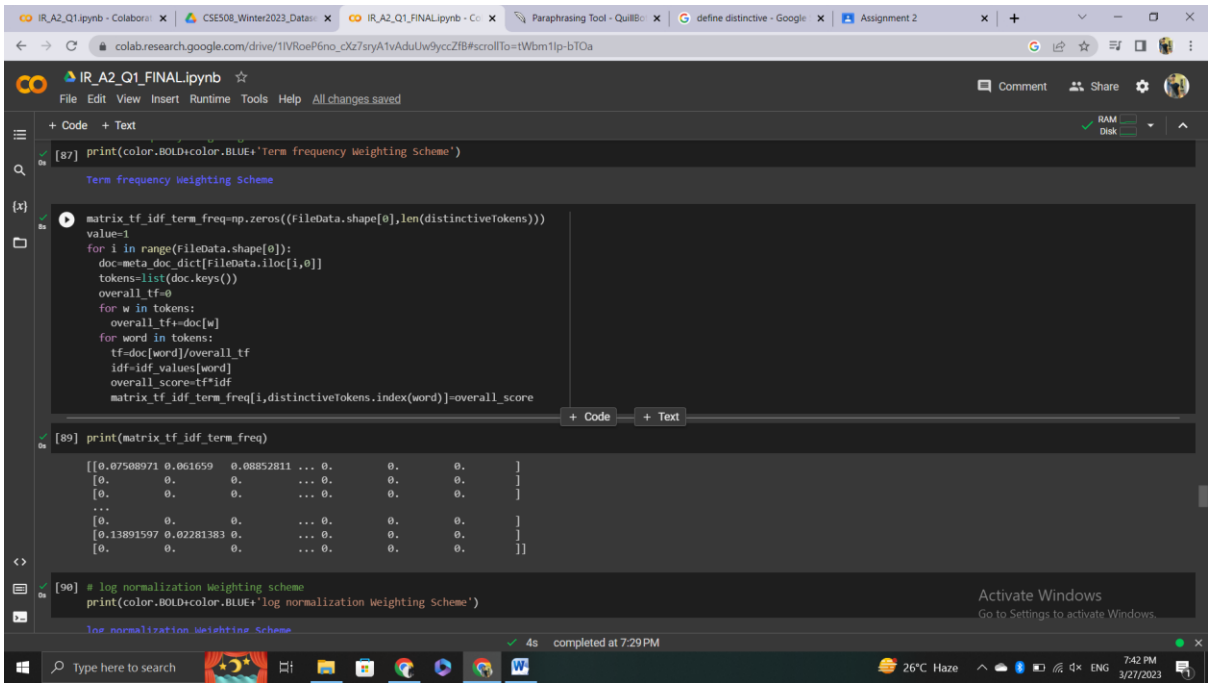
```
print(matrix_tf_idf_raw_count)
```

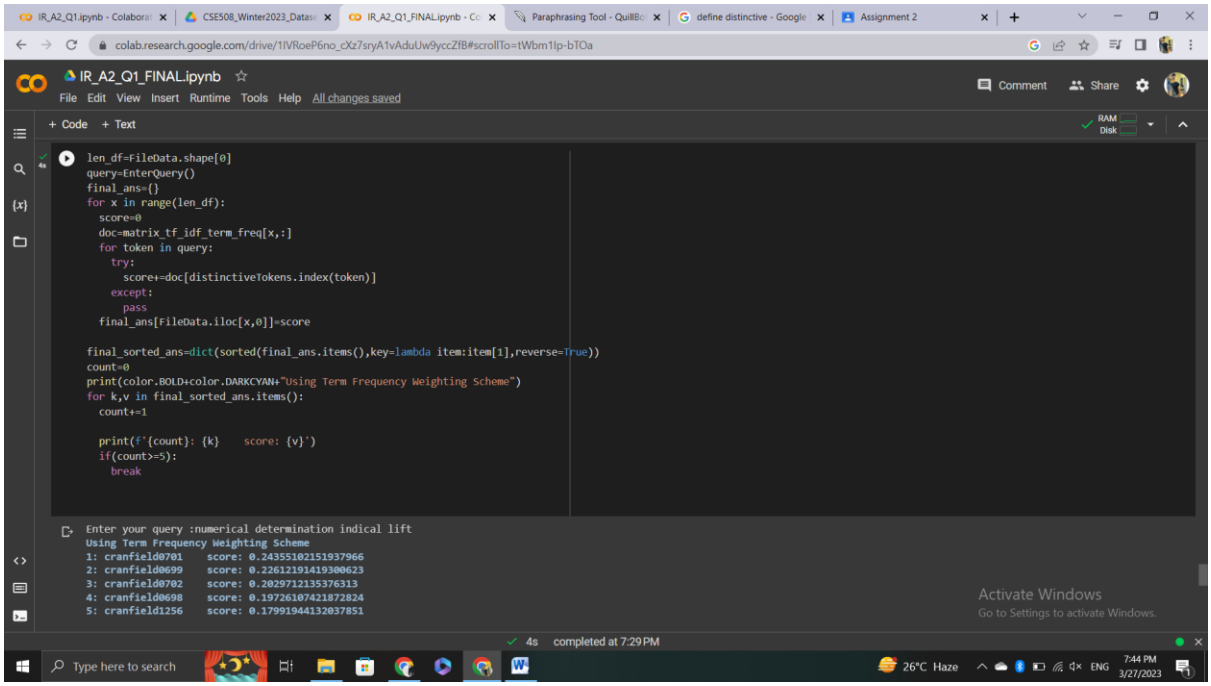
```
[[ 5.55663879  4.56276577  6.55108034 ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 ...
 [ 0.          0.          0.         ...  0.          0.
   0.        ]
 [13.89159698  2.28138289  0.         ...  0.          0.
   0.        ]
 [ 0.          0.          0.         ...  0.          0.
   0.        ]]
```

```
# Term frequency Weighting scheme
print(color.BOLD+color.BLUE+'Term frequency Weighting Scheme')
```



❖ **TERM FREQUENCY:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."
TF value is equals to the Word count stored in the dictionary created divided by the total number of words in that document.
TF-IDF score = tf * idf

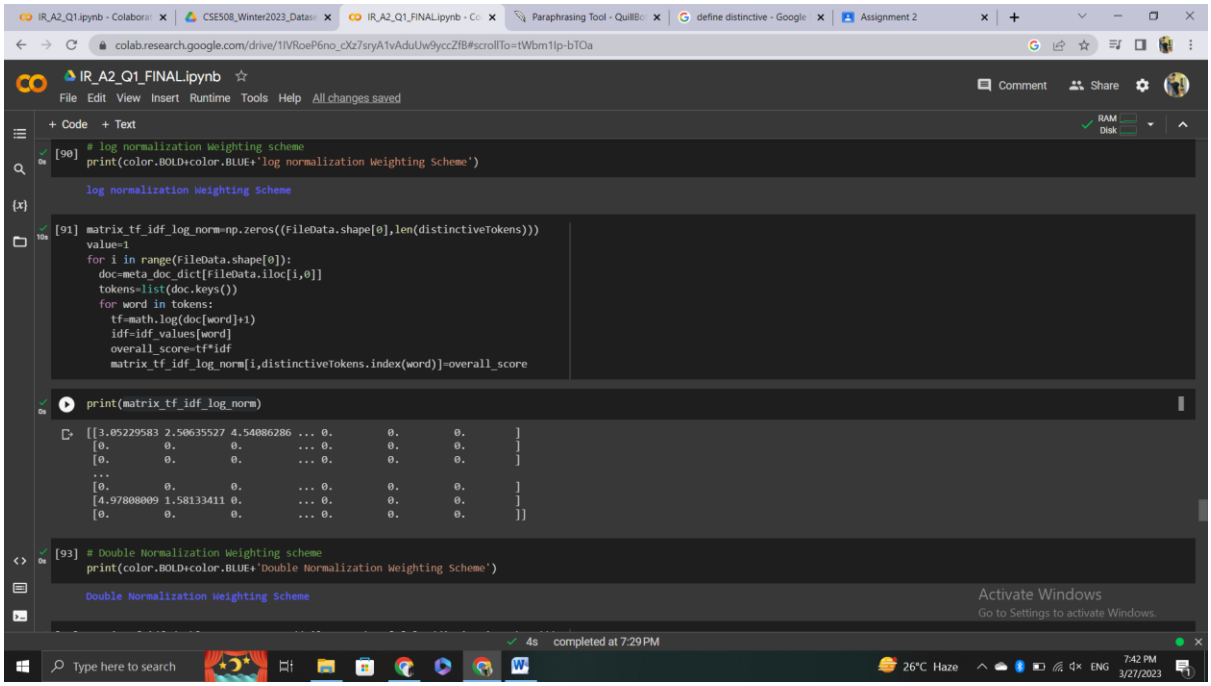




```
len_df=FileData.shape[0]
query=EnterQuery()
final_ans={}
for x in range(len_df):
  score=0
  doc=matrix_tf_idf_term_freq[x,:]
  for token in query:
    try:
      score+=doc[distinctiveTokens.index(token)]
    except:
      pass
  final_ans[FileData.iloc[x,0]]=score

final_sorted_ans=dict(sorted(final_ans.items(),key=lambda item:item[1],reverse=True))
count=0
print(color.BOLD+color.DARKCYAN+"Using Term Frequency Weighting Scheme")
for k,v in final_sorted_ans.items():
  count+=1

  print(f'{count}: {k}    score: {v}')
  if(count>=5):
    break
```

```
Enter your query :numerical determination indical lift
Using Term Frequency Weighting Scheme
1: cranfield0701    score: 0.24355102151937966
2: cranfield0699    score: 0.22612191419300623
3: cranfield0702    score: 0.2029712135376313
4: cranfield0698    score: 0.19726107421872824
5: cranfield1256    score: 0.17991944132037851
```

❖ **LOG NORMALIZATION:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."
TF value = LOG(Word count stored in the dictionary created
divided by the total number of words in that document)
TF-IDF score = tf * idf



```
# log normalization Weighting scheme
print(color.BOLD+color.BLUE+'log normalization Weighting Scheme')
```

```
log normalization Weighting Scheme
```

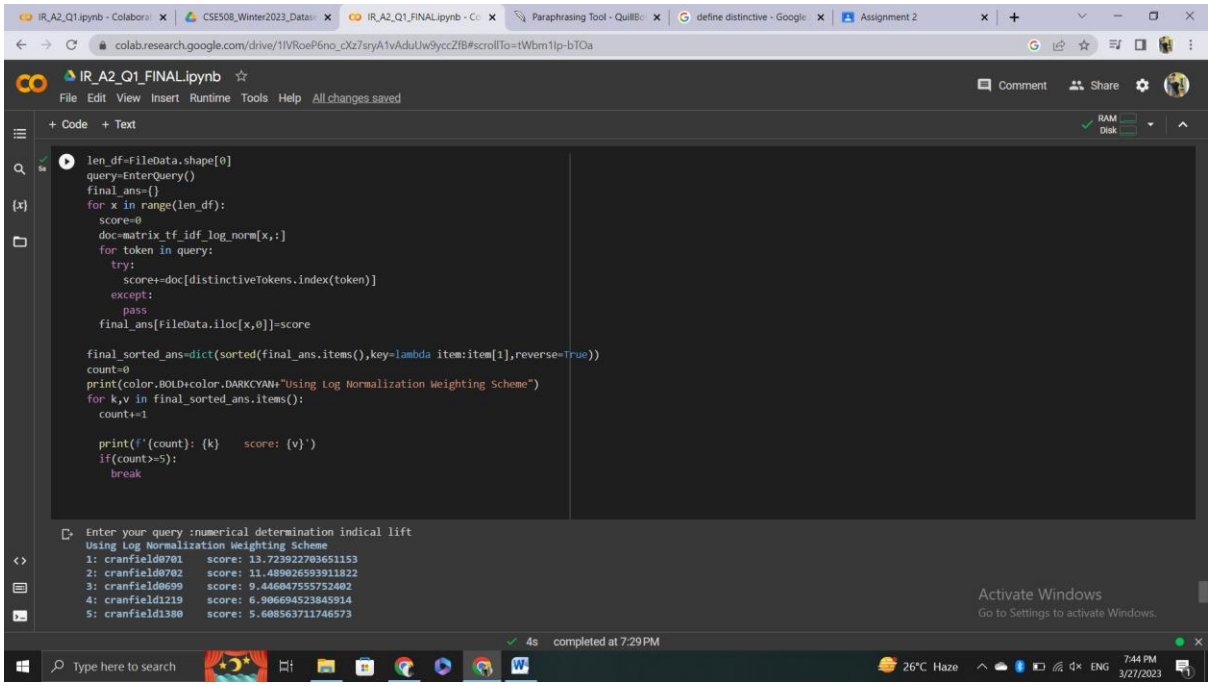
```
matrix_tf_idf_log_norm=np.zeros((FileData.shape[0],len(distinctiveTokens)))
value=1
for i in range(FileData.shape[0]):
  doc=meta_doc_dict[FileData.iloc[i,0]]
  tokens=list(doc.keys())
  for word in tokens:
    tf=math.log(doc[word]+1)
    idf=idf_values[word]
    overall_score=tf*idf
    matrix_tf_idf_log_norm[i,distinctiveTokens.index(word)]=overall_score
```

```
print(matrix_tf_idf_log_norm)
```

```
[[3.05229583 2.50635527 4.54086286 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [4.97808009 1.58133411 0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

```
# Double Normalization Weighting scheme
print(color.BOLD+color.BLUE+'Double Normalization Weighting Scheme')
```

```
Double Normalization Weighting Scheme
```

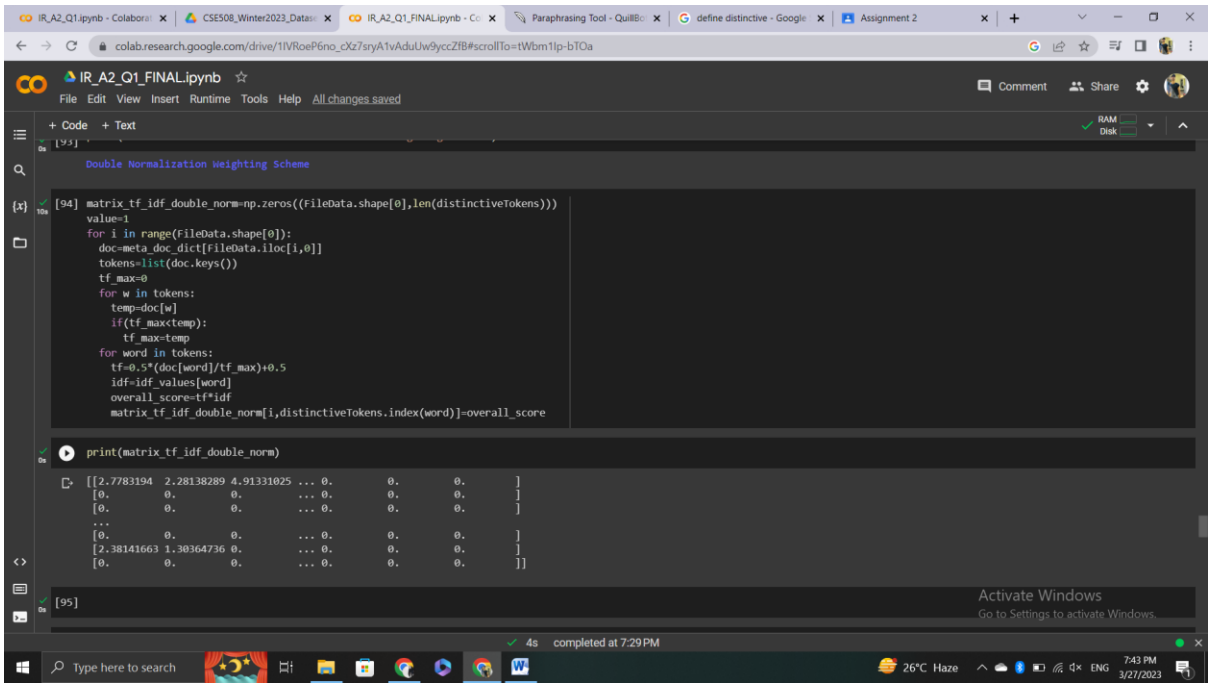


```
len_df=FileData.shape[0]
query=EnterQuery()
final_ans={}
for x in range(len_df):
  score=0
  doc=matrix_tf_idf_log_norm[x,:]
  for token in query:
    try:
      score+=doc[distinctiveTokens.index(token)]
    except:
      pass
  final_ans[FileData.iloc[x,0]]=score

final_sorted_ans=dict(sorted(final_ans.items(),key=lambda item:item[1],reverse=True))
count=0
print(color.BOLD+color.DARKCYAN+"Using Log Normalization Weighting Scheme")
for k,v in final_sorted_ans.items():
  count+=1

  print(f'{count}: {k}    score: {v}')
  if(count>=5):
    break
```

```
Enter your query :numerical determination indical lift
Using Log Normalization Weighting Scheme
1: cranfield0701    score: 13.723922703651153
2: cranfield0702    score: 11.489026593911822
3: cranfield0699    score: 9.446047555752402
4: cranfield1219    score: 6.906694523845914
5: cranfield1380    score: 5.608563711746573
```

❖ **DOUBLE NORMALIZATION:** In the TF-IDF matrix, ROW represents "Number of Documents" and COLUMN represents "Number of Unique Words in the Word Corpus."
TF value = 0.5+0.5*(f(t,d)/ max(f(t',d))
TF-IDF score = tf * idf

Result:

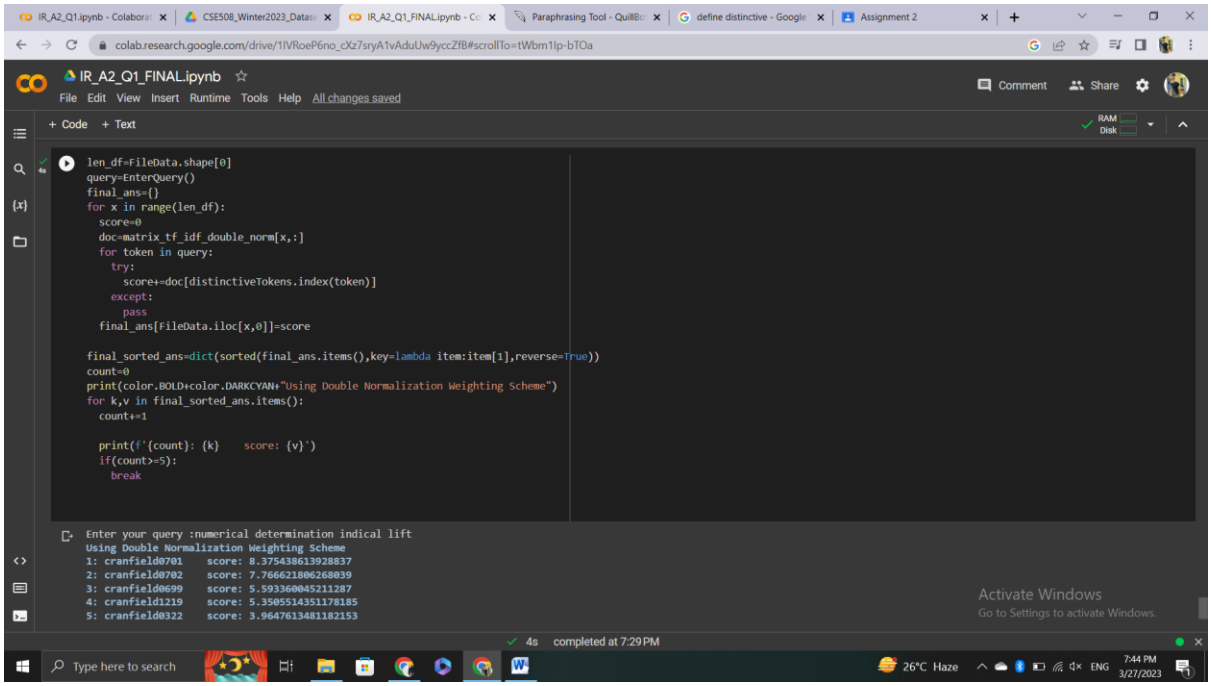


```
Double Normalization Weighting Scheme

matrix_tf_idf_double_norm=np.zeros((FileData.shape[0],len(distinctiveTokens)))
value=1
for i in range(FileData.shape[0]):
  doc=meta_doc_dict[FileData.iloc[i,0]]
  tokens=list(doc.keys())
  tf_max=0
  for w in tokens:
    temp=doc[w]
    if(tf_max<temp):
      tf_max=temp
  for word in tokens:
    tf=0.5*(doc[word]/tf_max)+0.5
    idf=idf_values[word]
    overall_score=tf*idf
    matrix_tf_idf_double_norm[i,distinctiveTokens.index(word)]=overall_score
```

```
print(matrix_tf_idf_double_norm)
```

```
[[2.7783194  2.28138289 4.91331025 ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [2.38141663 1.30364736 0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

```
[95]
```



Pros and cons of using each scoring scheme:

1. JACCARD COEFFICIENT

Benefits:

The sets don't have to be the same size.

The Jaccard similarity rating usually ranges from 0 to 1.

Cons:

The term frequency is not taken into account when determining similarity.

We are unable to determine which terms are more and less frequent .

The length normalisation is not taken into account.

2. TF-IDF

Binary Weighting Scheme:

Pros:

Simple and easy to implement.

Efficient in terms of computation time.

Cons:

Ignores the frequency of the term in the document, which can lead to inaccurate results.

All terms are given equal weight, which can result in less relevant documents being ranked Higher

.

Raw Count Weighting Scheme:

Pros:

Takes into account the frequency of the term in the document, which can improve the accuracy of the results.

Allows for different weighting of terms based on their frequency.

Cons:

Does not take into account the frequency of the term in the collection, which can lead to skewed results.

Can be biased towards longer documents, as they may have more occurrences of a given term.

Term Frequency (TF) Weighting Scheme:
Pros:
Takes into account the frequency of the term in the document, which can improve the accuracy of the results.
Allows for different weighting of terms based on their frequency.
Cons:
Does not take into account the frequency of the term in the collection, which can lead to skewed results.
Can be biased towards longer documents, as they may have more occurrences of a given term.

Log Normalization Weighting Scheme:
Pros:
Reduces the impact of very high term frequencies.
Helps to account for differences in document length.
Cons:
Can be sensitive to very low term frequencies.
May require tuning of the parameter to achieve optimal results.

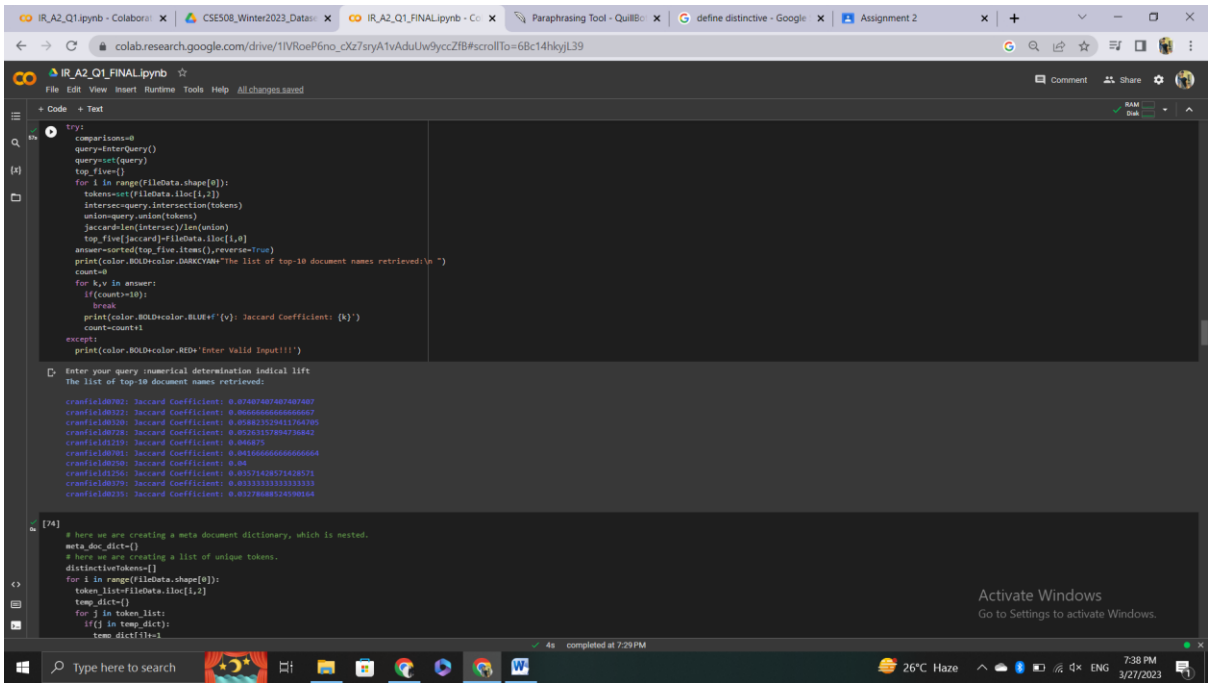
Double Normalization Weighting Scheme:
Pros:
Helps to account for differences in document length and term frequency.
Can improve the accuracy of the results compared to other weighting schemes.
Cons:
Can be sensitive to very low term frequencies.
May require tuning of the parameters to achieve optimal results.

B. Jaccard Coefficient
● With the help of the pre-processing techniques mentioned above, we were able to obtain distinct tokens for each of the queries and the word corpus. Following that, we created sets for the word corpus and query tokens.
● Then, using the intersection and union operations of Python sets, we collected the union and intersection between the set of query tokens and the set of document tokens for each file.

The Jaccard score for a given document was determined after obtaining the union and intersection by dividing the length of the intersection between the query and document corpus by the length of the union between the two.

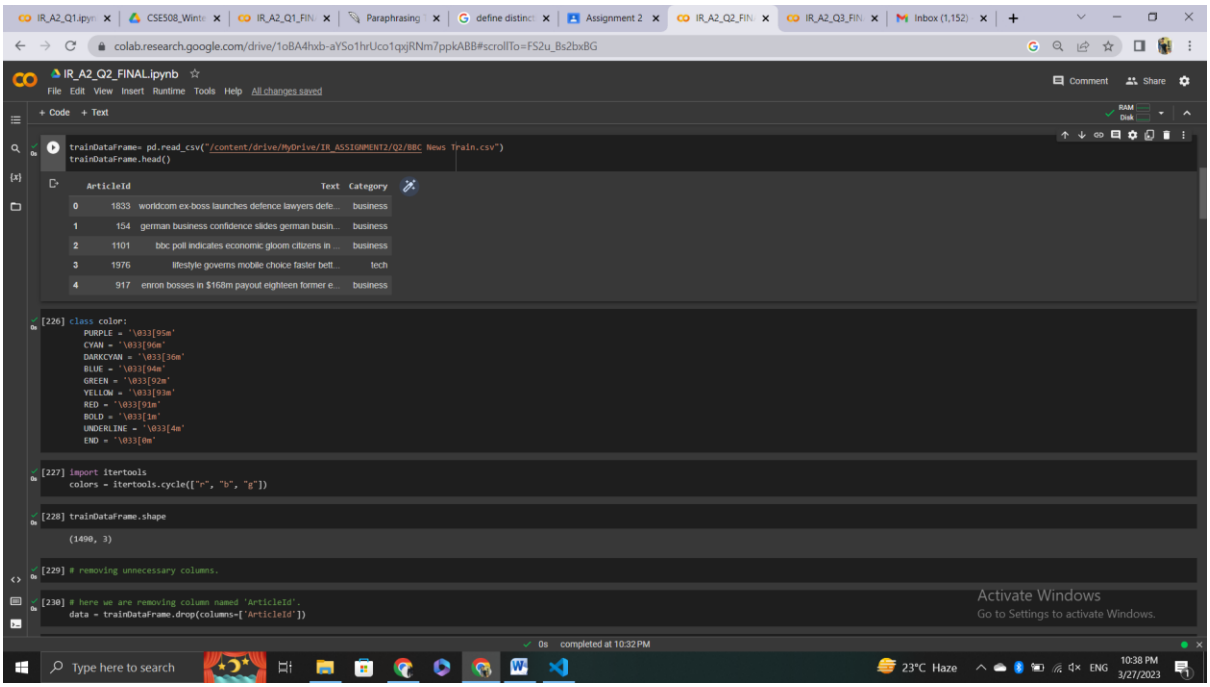
● The top ten documents with the highest Jaccard coefficient were then determined by ranking the results from the previous steps.

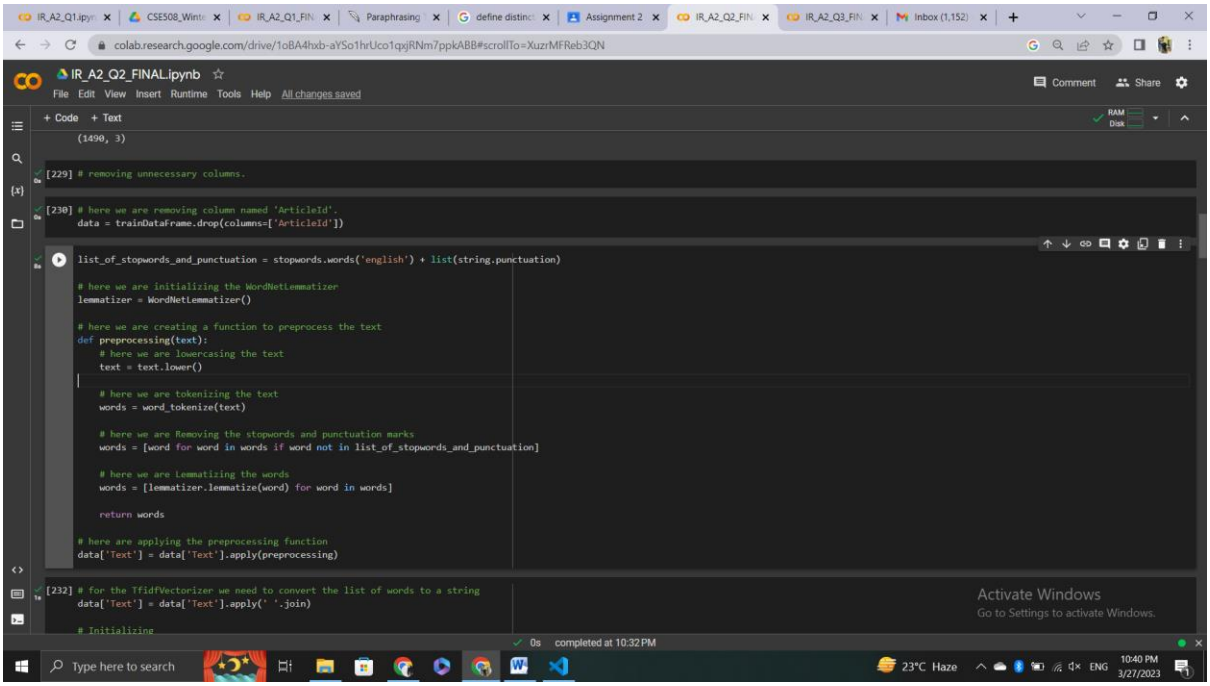
Result:



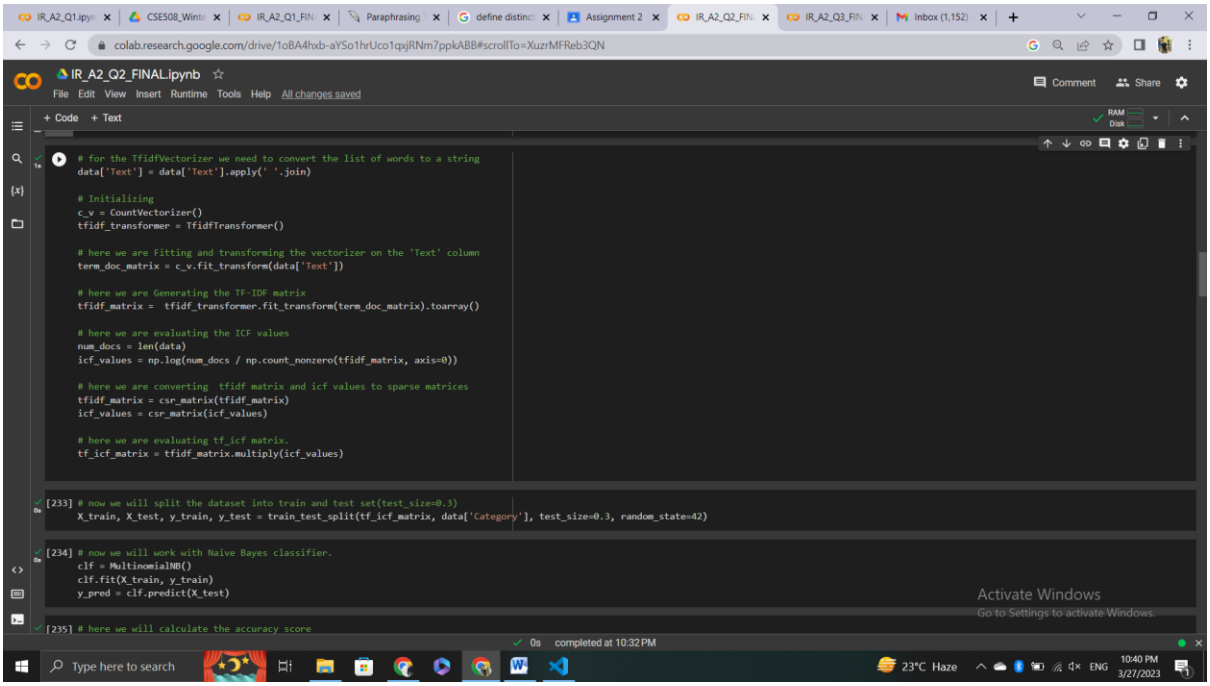
Question 2

1. Preprocessing the dataset:

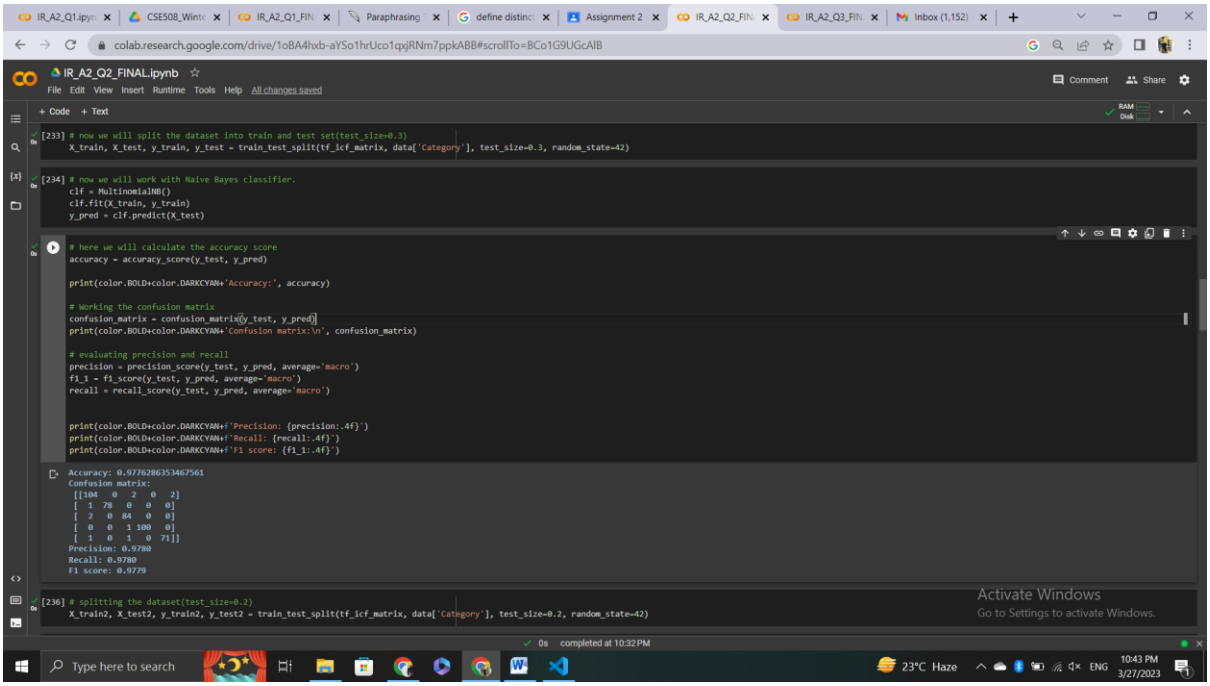


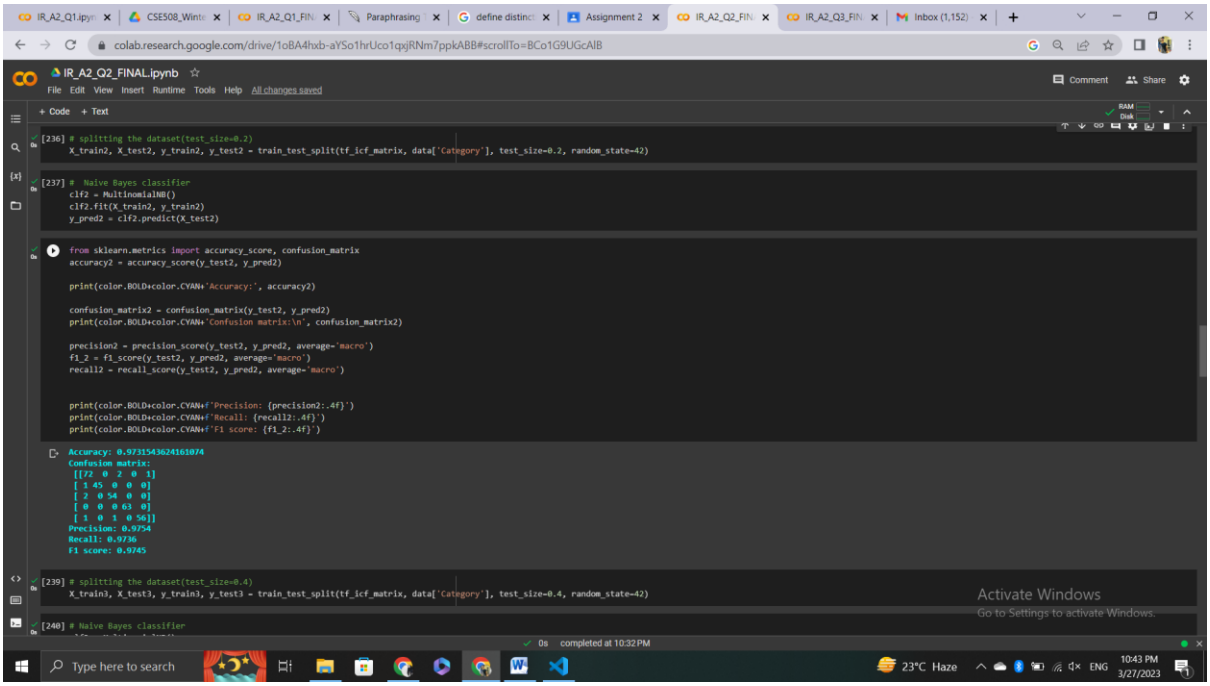


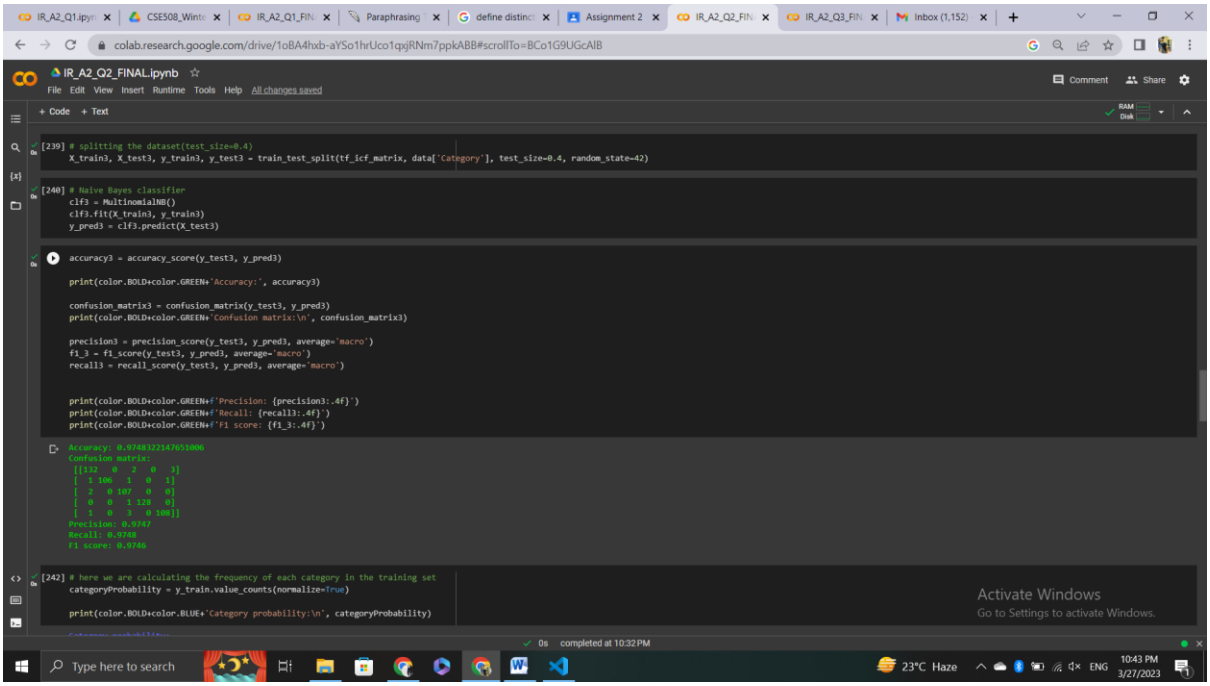
2. the dataset:

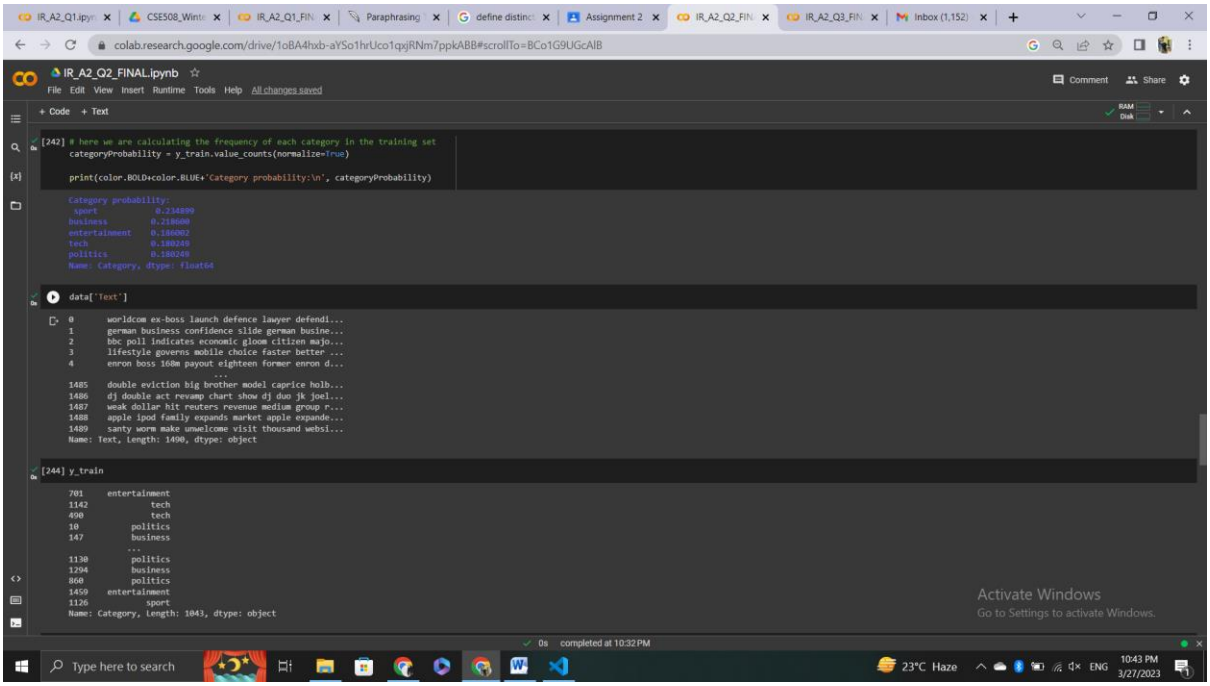


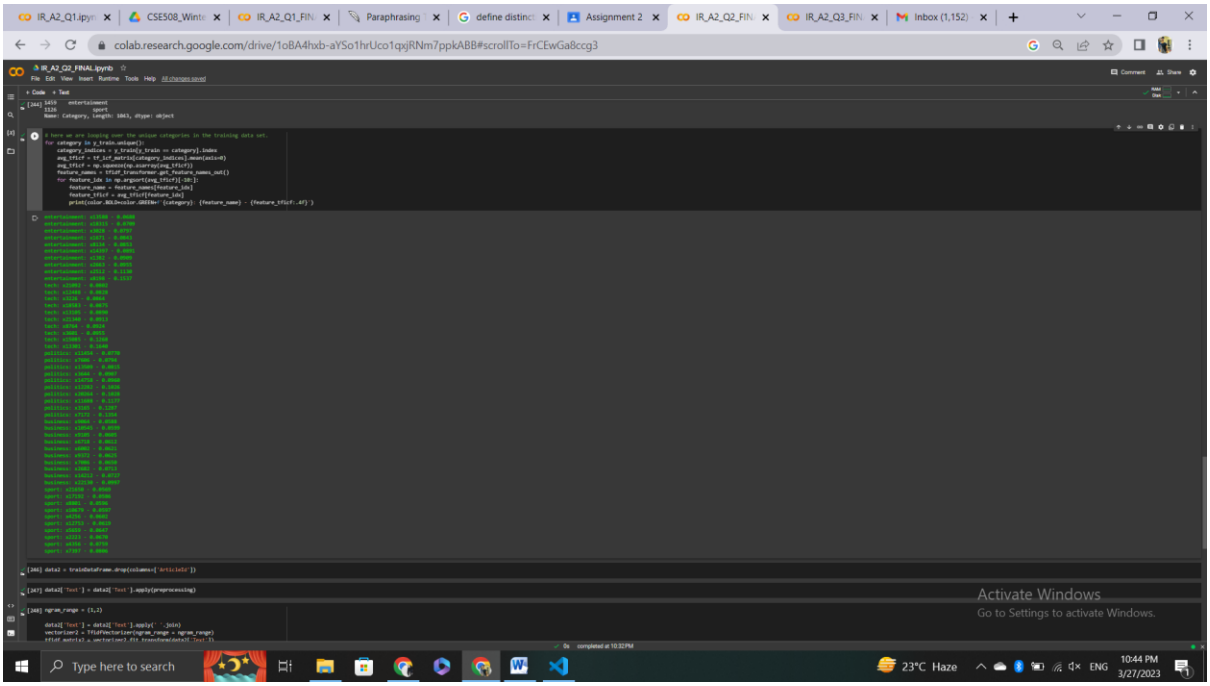
3. Training, Testing and improving the classifier:

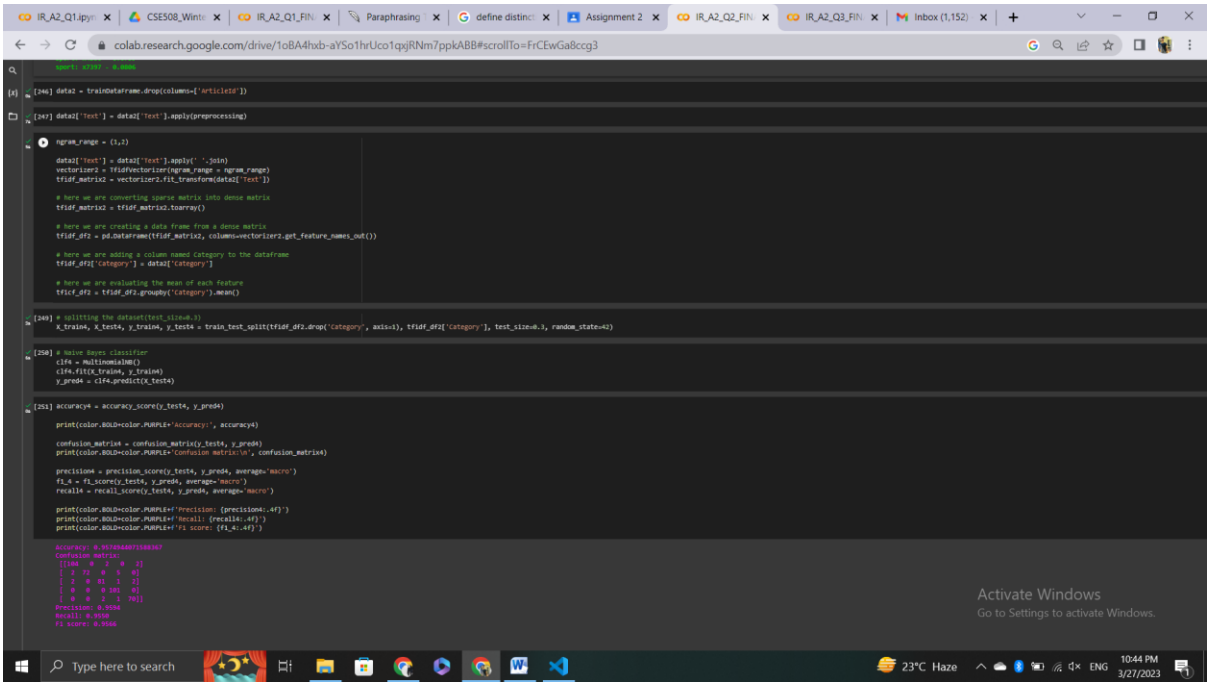

IR_A2_Q2_FINAL.ipynb
[233] # now we will split the dataset into train and test set(test_size=0.3)
X_train, X_test, y_train, y_test = train_test_split(tf_icf_matrix, data['Category'], test_size=0.3, random_state=42)
[234] # now we will work with Naive Bayes classifier.
clf = MultinomialNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
# here we will calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(color.BOLD+color.DARKCYAN+'Accuracy:', accuracy)
# Working the confusion matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(color.BOLD+color.DARKCYAN+'Confusion matrix:\n', confusion_matrix)
# evaluating precision and recall
precision = precision_score(y_test, y_pred, average='macro')
f1_1 = f1_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
print(color.BOLD+color.DARKCYAN+f'Precision: {precision:.4f}')
print(color.BOLD+color.DARKCYAN+f'Recall: {recall:.4f}')
print(color.BOLD+color.DARKCYAN+f'F1 score: {f1_1:.4f}')
Accuracy: 0.9776286353467561
Confusion matrix:
[[104 0 2 0 2]
[1 78 0 0 0]
[2 0 84 0 0]
[0 0 1 100 0]
[1 0 1 0 71]]
Precision: 0.9780
Recall: 0.9780
F1 score: 0.9779
[236] # splitting the dataset(test_size=0.2)
X_train2, X_test2, y_train2, y_test2 = train_test_split(tf_icf_matrix, data['Category'], test_size=0.2, random_state=42)
Activate Windows
Go to Settings to activate Windows.
completed at 10:32 PM
Type here to search
23°C Haze
ENG
10:43 PM
3/27/2023


IR_A2_Q2_FINAL.ipynb
[236] # splitting the dataset(test_size=0.2)
X_train2, X_test2, y_train2, y_test2 = train_test_split(tf_icf_matrix, data['Category'], test_size=0.2, random_state=42)
[237] # Naive Bayes classifier
clf2 = MultinomialNB()
clf2.fit(X_train2, y_train2)
y_pred2 = clf2.predict(X_test2)
from sklearn.metrics import accuracy_score, confusion_matrix
accuracy2 = accuracy_score(y_test2, y_pred2)
print(color.BOLD+color.CYAN+'Accuracy:', accuracy2)
confusion_matrix2 = confusion_matrix(y_test2, y_pred2)
print(color.BOLD+color.CYAN+'Confusion matrix:\n', confusion_matrix2)
precision2 = precision_score(y_test2, y_pred2, average='macro')
f1_2 = f1_score(y_test2, y_pred2, average='macro')
recall2 = recall_score(y_test2, y_pred2, average='macro')
print(color.BOLD+color.CYAN+f'Precision: {precision2:.4f}')
print(color.BOLD+color.CYAN+f'Recall: {recall2:.4f}')
print(color.BOLD+color.CYAN+f'F1 score: {f1_2:.4f}')
Accuracy: 0.9731543624161074
Confusion matrix:
[[72 0 2 0 1]
[1 45 0 0 0]
[2 0 54 0 0]
[0 0 0 63 0]
[1 0 1 0 56]]
Precision: 0.9754
Recall: 0.9730
F1 score: 0.9745
[239] # splitting the dataset(test_size=0.4)
X_train3, X_test3, y_train3, y_test3 = train_test_split(tf_icf_matrix, data['Category'], test_size=0.4, random_state=42)
[240] # Naive Bayes classifier
Activate Windows
Go to Settings to activate Windows.
completed at 10:32 PM
Type here to search
23°C Haze
ENG
10:43 PM
3/27/2023


IR_A2_Q2_FINAL.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[239] # splitting the dataset(test_size=0.4)
X_train3, X_test3, y_train3, y_test3 = train_test_split(tf_icf_matrix, data['Category'], test_size=0.4, random_state=42)
[240] # Naive Bayes classifier
clf3 = MultinomialNB()
clf3.fit(X_train3, y_train3)
y_pred3 = clf3.predict(X_test3)
accuracy3 = accuracy_score(y_test3, y_pred3)
print(color.BOLD+color.GREEN+'Accuracy:', accuracy3)
confusion_matrix3 = confusion_matrix(y_test3, y_pred3)
print(color.BOLD+color.GREEN+'Confusion matrix:\n', confusion_matrix3)
precision3 = precision_score(y_test3, y_pred3, average='macro')
f1_3 = f1_score(y_test3, y_pred3, average='macro')
recall3 = recall_score(y_test3, y_pred3, average='macro')
print(color.BOLD+color.GREEN+f'Precision: {precision3:.4f}')
print(color.BOLD+color.GREEN+f'Recall: {recall3:.4f}')
print(color.BOLD+color.GREEN+f'F1 score: {f1_3:.4f}')
Accuracy: 0.9748322147651006
Confusion matrix:
[[132 0 2 0 1]
[1 106 1 0 1]
[2 0 107 0 0]
[0 0 1 128 0]
[1 0 1 0 100]]
Precision: 0.9747
Recall: 0.9748
F1 score: 0.9746
[242] # here we are calculating the frequency of each category in the training set
categoryProbability = y_train.value_counts(normalize=True)
print(color.BOLD+color.BLUE+'Category probability:\n', categoryProbability)
Activate Windows
Go to Settings to activate Windows.
0s completed at 10:32 PM
Type here to search
23°C Haze
ENG
10:43 PM
3/27/2023


IR_A2_Q2_FINAL.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[242] # here we are calculating the frequency of each category in the training set
categoryProbability = y_train.value_counts(normalize=True)
print(color.BOLD+color.BLUE+'Category probability:\n', categoryProbability)
Category probability:
sport 0.234899
business 0.218600
entertainment 0.186002
tech 0.180249
politics 0.180249
Name: Category, dtype: float64
data['Text']
0 worldcom ex-boss launch defence lawyer defendi...
1 german business confidence slide german busine...
2 bbc poll indicates economic gloom citizen majo...
3 lifestyle governs mobile choice faster better ...
4 enron boss 168m payout eighteen former enron d...
1485 double eviction big brother model caprice holb...
1486 dj double act revamp chart show dj duo jk joel...
1487 weak dollar hit reuters revenue medium group r...
1488 apple ipod family expands market apple expande...
1489 santy worm make unwelcome visit thousand websi...
Name: Text, Length: 1490, dtype: object
[244] y_train
701 entertainment
1142 tech
490 tech
10 politics
147 business
1130 politics
1294 business
860 politics
1459 entertainment
1126 sport
Name: Category, Length: 1043, dtype: object
Activate Windows
Go to Settings to activate Windows.
0s completed at 10:32 PM
Type here to search
23°C Haze
ENG
10:43 PM
3/27/2023





4. **Conclusion:**

   ● **Write a brief report summarizing your findings.**

   the code is performing text classification on the BBC News dataset using Naive Bayes classifier.

   First, the dataset is read using Pandas and the 'ArticleId' column is removed. Then, the NLTK library is used to preprocess the text data by converting it to lowercase,

tokenizing it, removing stop words and punctuation marks, and lemmatizing the words.

After preprocessing the data, the TF-IDF vectorizer is used to generate a term-document matrix and the TF-IDF matrix is computed from it. ICF (Inverse Category Frequency) values are then computed from the TF-IDF matrix. The TF-ICF matrix is computed from the element-wise multiplication of the TF-IDF matrix and the ICF matrix.

The dataset is split into training and testing sets using the train_test_split function from scikit-learn. The Naive Bayes classifier is then initialized and fit on the training data. The model is evaluated on the testing data by predicting the categories of the articles and calculating various evaluation metrics such as accuracy, precision, recall, and F1 score.

The code is also performing the same text classification task on the dataset for two other test sizes (20% and 40%) and calculating the evaluation metrics for each of them separately.

- **Discuss the performance of the classifier and the impact of different preprocessing techniques, features, and weighting schemes on the results.**

The performance of a classifier can be evaluated using various metrics such as accuracy, precision, recall, F1-score, and area under the curve (AUC) of the receiver operating characteristic (ROC) curve. The impact of different preprocessing techniques, features, and weighting schemes can also have a significant impact on the performance of a classifier.

Preprocessing techniques such as data cleaning, normalization, and feature selection can help to improve the performance of a classifier. For example, removing irrelevant features and handling missing values can reduce noise and improve the accuracy of the model. Similarly, normalization can help to standardize the range of values for each feature, which can prevent some features from dominating others and improve the stability of the model.

Feature selection is another important preprocessing technique that can help to reduce the dimensionality of the data and improve the accuracy of the model. It involves selecting a subset of the most relevant features for the classification task. Different feature selection algorithms can be used, such as the chi-squared test, mutual information, and correlation-based feature selection.

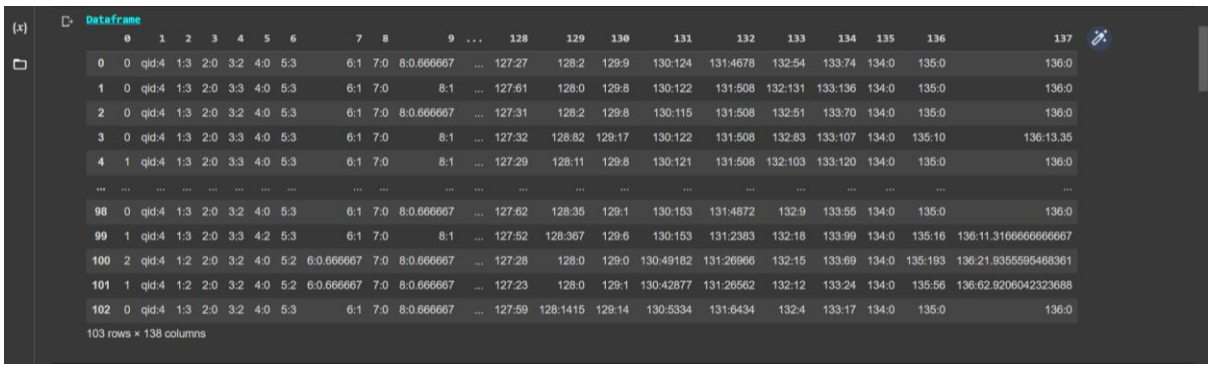
Weighting schemes are also important in improving the performance of a classifier. For example, using inverse document frequency (IDF) weighting can help to reduce the impact of frequent terms in text classification tasks. Similarly, using term frequency-inverse document frequency (TF-IDF) weighting can improve the

performance of text classification models by giving more weight to terms that are specific to a particular document or class.

Overall, the performance of a classifier depends on various factors such as the quality of the data, the choice of preprocessing techniques, the selection of features, and the weighting scheme used. Therefore, it is important to experiment with different combinations of these factors to determine the best approach for a particular classification task.

Question 3

Part 1:
Using pandas, load the dataset, and extract all of the rows having the qid:4 value.

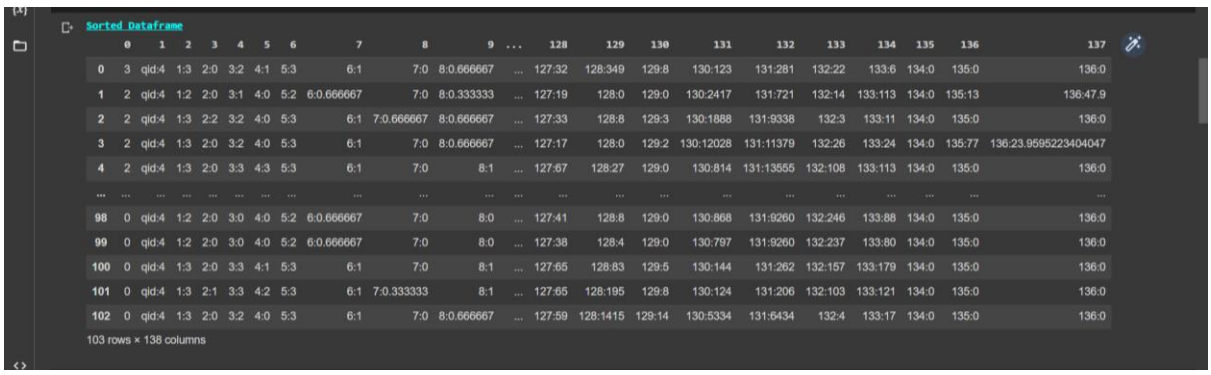


Dataframe

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
98	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	127:62	128:35	129:1	130:153	131:4872	132:9	133:55	134:0	135:0	136:0
99	1	qid:4	1:3	2:0	3:3	4:2	5:3	6:1	7:0	8:1	...	127:52	128:367	129:6	130:153	131:2383	132:18	133:99	134:0	135:16	136:11.3166666666667
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
101	1	qid:4	1:2	2:0	3:2	4:0	5:2	6:0.666667	7:0	8:0.666667	...	127:23	128:0	129:1	130:42877	131:26562	132:12	133:24	134:0	135:56	136:62.9206042323688
102	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	127:59	128:1415	129:14	130:5334	131:6434	132:4	133:17	134:0	135:0	136:0

103 rows × 138 columns

Part 2:
The relevance of all the terms is sorted before the DCG value is calculated to determine the maximum DCG. DCG value is calculated using the formula:



Sorted_Dataframe

	0	1	2	3	4	5	6	7	8	9	...	128	129	130	131	132	133	134	135	136	137
0	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
1	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
2	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
3	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
4	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
98	0	qid:4	1:2	2:0	3:0	4:0	5:2	6:0.666667	7:0	8:0	...	127:41	128:8	129:0	130:868	131:9260	132:246	133:88	134:0	135:0	136:0
99	0	qid:4	1:2	2:0	3:0	4:0	5:2	6:0.666667	7:0	8:0	...	127:38	128:4	129:0	130:797	131:9260	132:237	133:80	134:0	135:0	136:0
100	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
101	0	qid:4	1:3	2:1	3:3	4:2	5:3	6:1	7:0.333333	8:1	...	127:65	128:195	129:8	130:124	131:206	132:103	133:121	134:0	135:0	136:0
102	0	qid:4	1:3	2:0	3:2	4:0	5:3	6:1	7:0	8:0.666667	...	127:59	128:1415	129:14	130:5334	131:6434	132:4	133:17	134:0	135:0	136:0

103 rows × 138 columns

```
Maximum Discounted Cumulative Gain = 20.989750804831445

Discounted Cumulative Gain = 12.550247459532576

Count of possible files that could get MAXDCG are : 19893497375938370599826047614905329896936840170566570588205180312704857992695193482412686565431050240000000000000000000000
```

To calculate the number of files that could be made, we calculate the factorial of each relevant class and then multiply them to get the total number of ways.

Part 3:

nDCG is calculated using the formula : $nDCG = DCG \ / \ Max \ DCG$

For first 50 dataset, we calculate nDCG for first 50 rows of dataset.

```
Maximum Discounted Cumulative Gain (MDCG) for whole dataset is: 0.5979226516897831

Discounted Cumulative Gain (DCG) for first 50 rows is: 0.5253808413557646
```

Part 4:

Based on the 75th feature, we rank the URL. The URL is more pertinent the higher the value of this feature. After setting all nonnegative relevance to 1, we compute precision@k and recall@k while iterating through the dataset.