**Department of**

**Computer Science and Engineering**

***Information Retrieval***

***CSE645***

***Assignment-1***

**20-marks lab component**

**Team member 1: Arun J Kennedy**

**USN: 1MS18CS029**

**Team member 2: Ashik MP**

**USN: 1MS18CS030**

**M S Ramaiah Institute of Technology, Bangalore**

**Department of Computer Science & Engineering**

**Bangalore - 560054, Karnataka, India**

**Exercise 1**: Implementation of Pre-processing of a Text Document.

**Aim:** Given a set of documents, implement the preprocessing procedures and store the resulting file.

**Steps to execute:**

*Step 1:* read the document and store it in a string

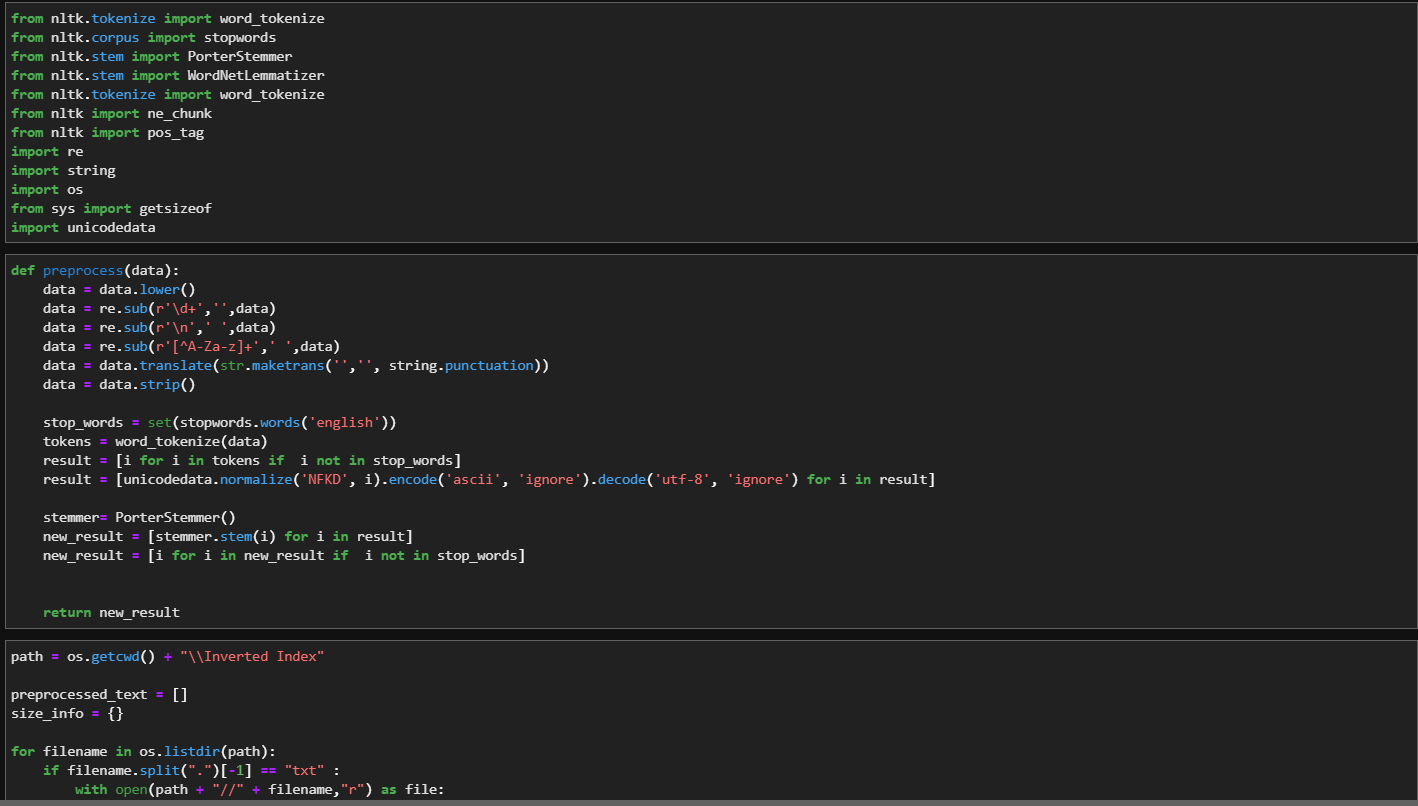
*Step 2:* convert to lower case and remove all the digits , non-ascii characters , whitespaces

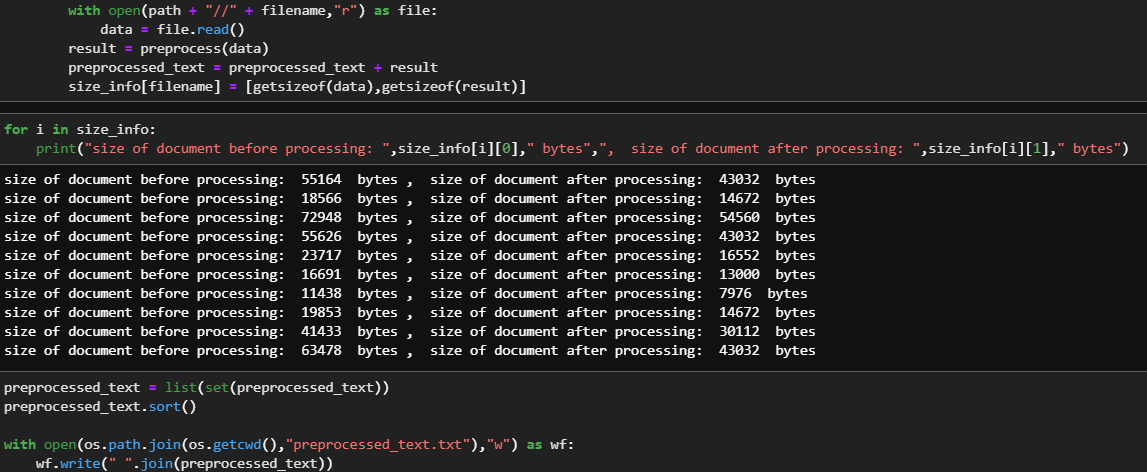
*Step 3:* tokenize the string and remove all the stop words present in that language

*Step 4:* use stemming on the resultant tokenized words and stem the words to its root form

*Step 5:* store the processed words in a text file.

**Code (python3 / jupyter notebook) / input / output:**

****



**Inference / Conclusion:**

Every document has certain format / styles / content that is irrelevant when someone is searching for the document, preprocessing the document by removing stopwords, numbers, and stemming each word makes it easier to compress the content of the document for easier searching and ranking.

**Exercise 2**: Implementation of Inverted Index: Construction and Searching

**Aim:** To construct an inverted index to access word count of all the words in a particular document from the processed text

**Steps to execute:**

*Step 1:* pre-process the set of documents and store the processed words in a text .

*Step 2:* store each processed document separately using a data structure (dictionary) .

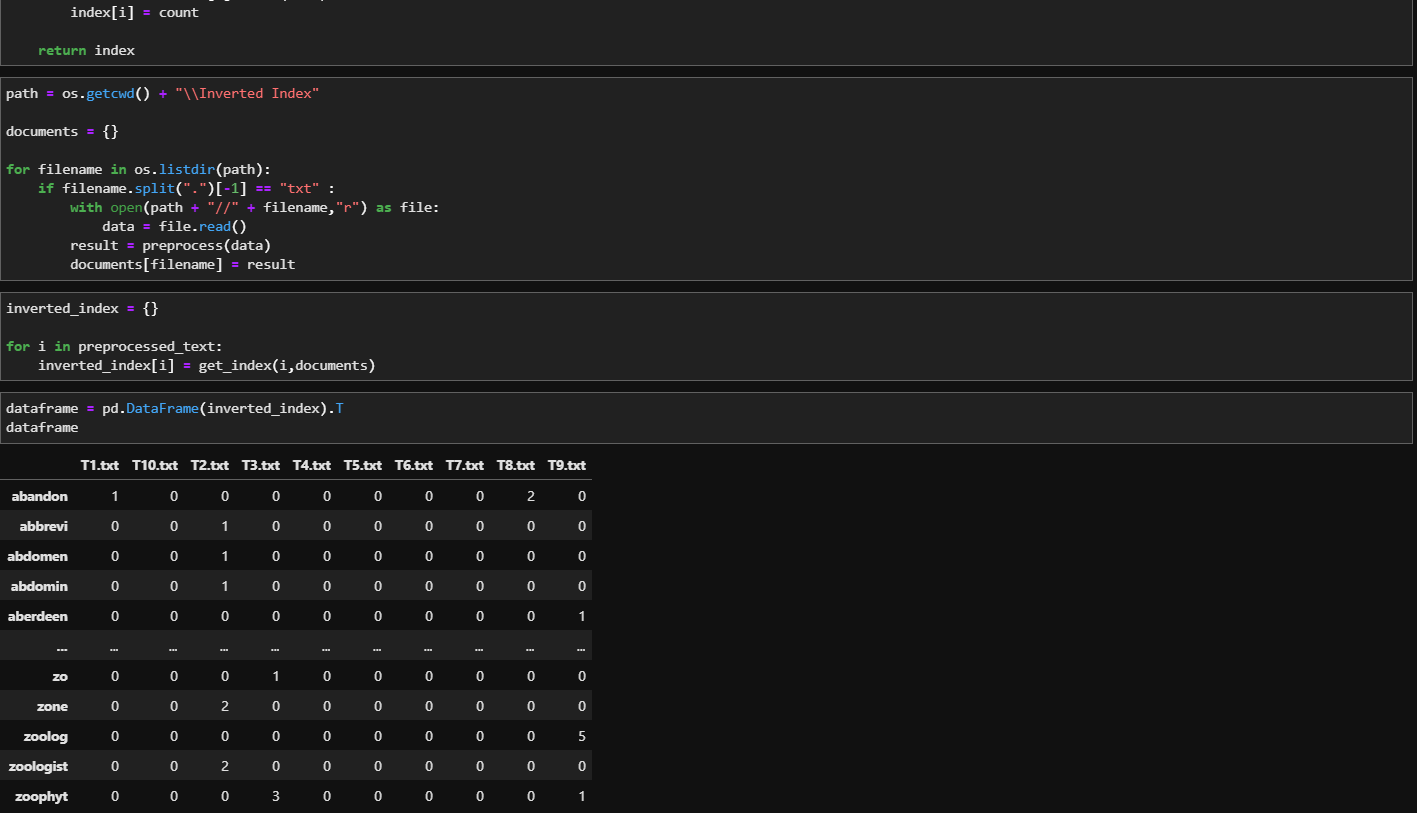
*Step 3:* for each word in the processed text file get the count of that word in every document and store it in an easily accessible data structure (hash table / DataFrame in python) .

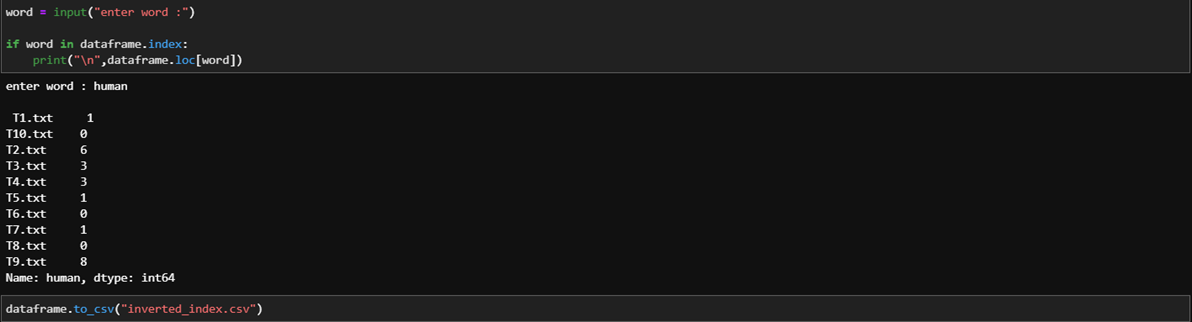
*Step 4:* take some word as a query input and get its count details by using the word as the hashed value to access it .

*Step 5:* Store the resultant inverted index in csv format.

**Code (python 3 / jupyter notebook) / input / output:**

****





**Inference/ Conclusion:**

Inverted indexing is the procedure in which the frequency of occurrence of each word in the document is stored. Preprocessing of documents is required before storing the inverted index to store relevant content.

**Exercise 3:** Implementation of vector Space model

A. Rank 10 documents for a given query.

B. Computing Similarity between any two documents.

**Aim:** To implement a vector Space model and for a given query rank 10 documents and also compute similarity between any two documents using and similarity method.

**Steps to execute:**

*Step 1:* pre-process the set of documents and store the processed words in a text

*Step 2:* construct an inverted index for all the words in the pre-processed text file as mentioned in 2nd exercise.

*Step 3:* for the inverted index find the tf-idf scores as follows :

*Step 3.1:* normalise the data w.r.t document (columns)

*Step 3.2:* get count of number of documents containing a word w and also get count of number of documents as N.

*Step 3.3:* get tf-idf score of a word by using formula log10(N/count[w])

*Step 4:* prompt the user to enter a query word/words and pre-process those words.

*Step 5:* get the tf-idf scores for the pre-processed words and store it in a dictionary

*Step 6: t*o get the rank of the query words follow the steps:

*Step 6.1:* calculate the distance of the query vector using square root of sum of squares and store in it variable ‘dist1’

*Step 6.2:* calculate the vector product by multiplying the query and each document vector and store it in ‘vec\_product’

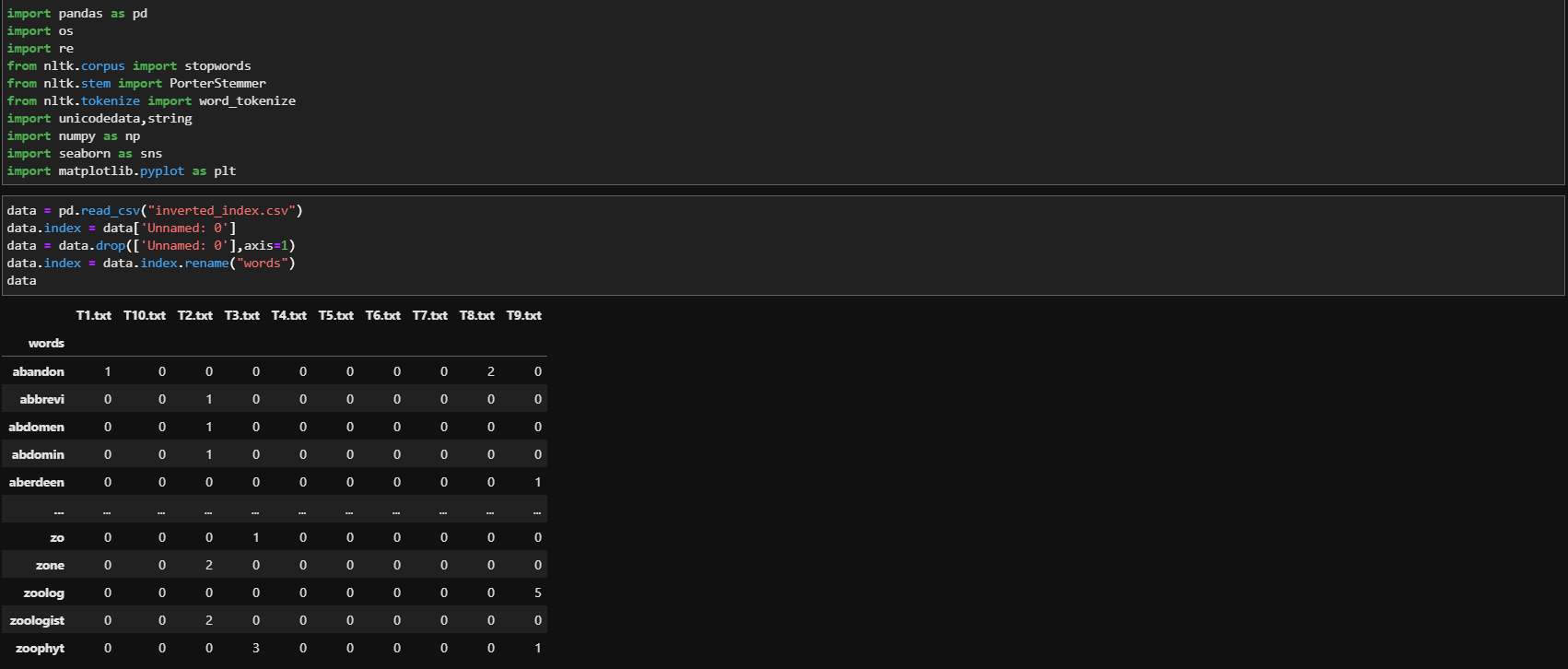
*Step 6.3:* for each document find the distance ‘dist2’ using same method and get the similarity between query vector and document vector as (vec\_product / dist1 \* dist2)

*Step 7*: sort and display the rank of the query w.r.t the documents

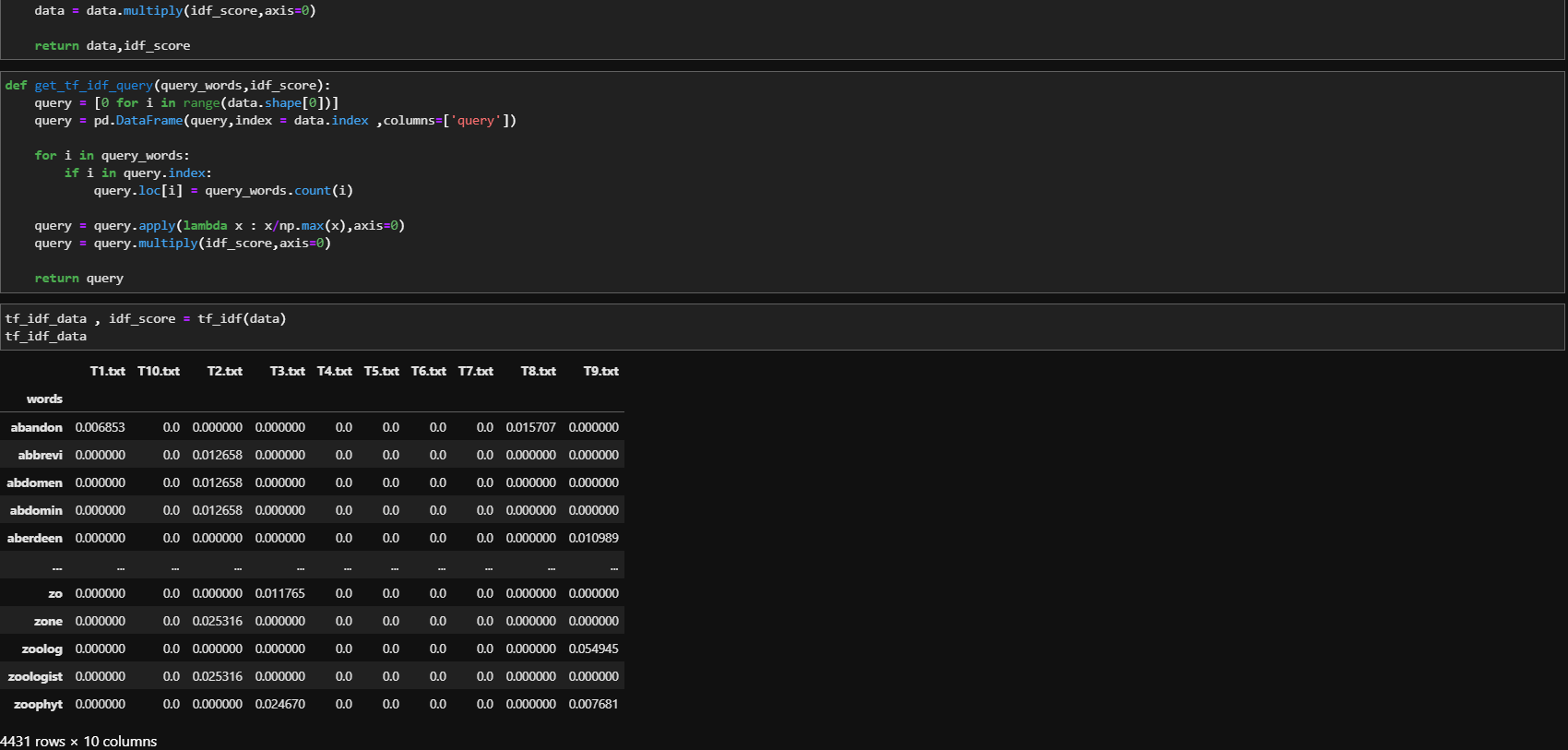
*Step 8:* repeat same steps for document similarity but instead of query vector compare each document vector with rest and store the result

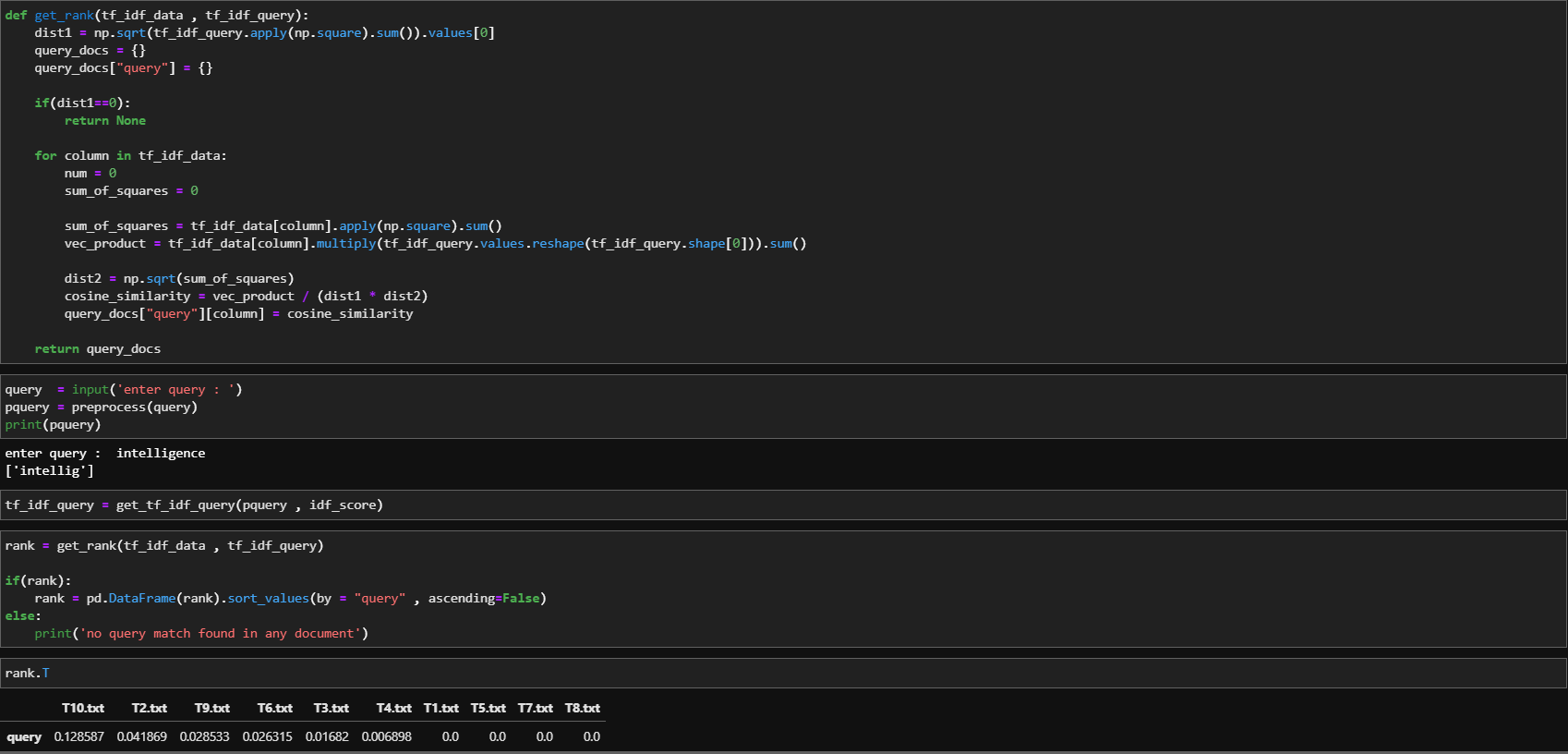
Step 9: display the confusion matrix obtained from comparison.

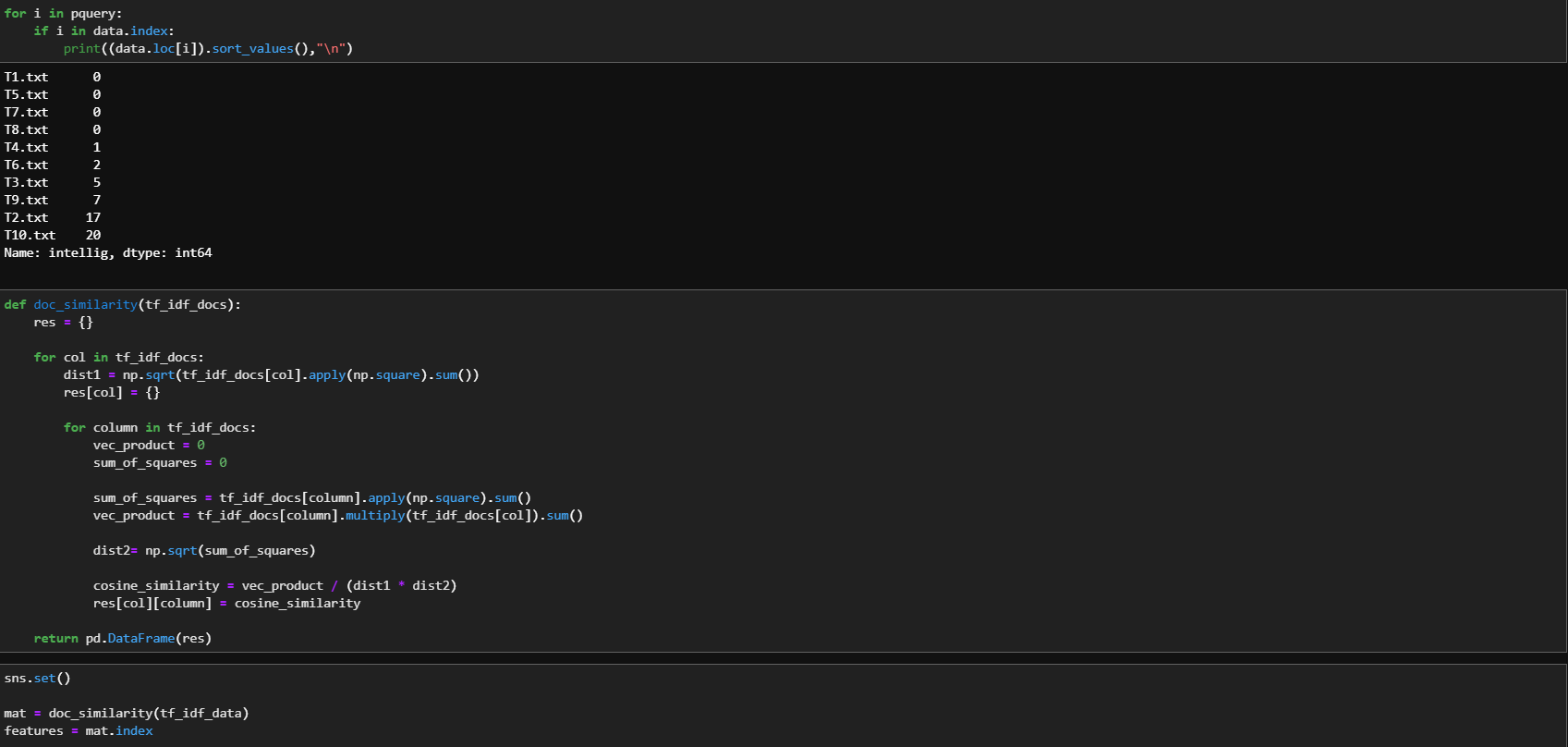
**Code (python 3 / jupyter notebook) / input / output:**

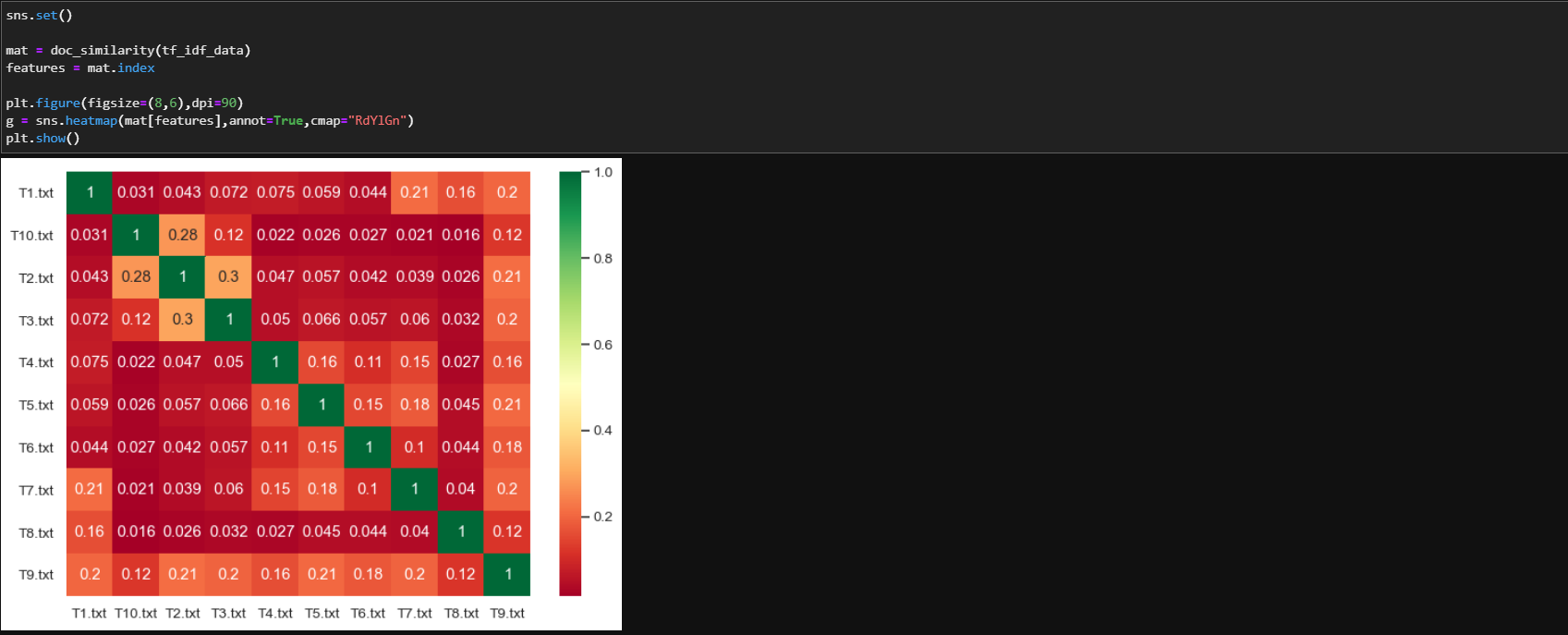
****











**Inference/Conclusion:**

Vector Space model is a clever method to find and rank the documents based on a query and compare the similarity between two documents. It treats each document as a vector of tf-idf weights and when a cosine product is applied it gives a similarity in the range of 0 to 1.

**Exercise 4:** Implementation of probabilistic Model.

Rank 10 documents for a given query.

**Aim:** Implement probabilistic model on a given set of 10 documents and Rank the documents for a given query.

**Steps to execute:**

*Step 1:* Preprocess the documents and obtain the inverted index for the files.

*Step 2:* Input the query from the user.

*Step 3:* Preprocess the query to obtain appropriate tokens to implement probabilistic model.

*Step 4:* Send the preprocessed query and the inverted index file into get\_conditional\_probability function to obtain a probability matrix.

*Step5:* Arrange the probabilities from the matrix obtained in descending order to get the Documents ranked from most relevant to least relevant.

**Code (python 3):**

import nltk  
import re  
from nltk.tokenize import RegexpTokenizer  
from nltk.corpus import stopwords  
from nltk.stem import PorterStemmer  
import pandas as pd  
  
xfiles = ['T1.txt', 'T2.txt', 'T3.txt', 'T4.txt', 'T5.txt', 'T6.txt', 'T7.txt', 'T8.txt', 'T9.txt', 'T10.txt']  
  
  
# Tokenizing the text, return token list  
  
  
def preprocess(sentence):  
 sentence = sentence.lower()  
 tokenizer = RegexpTokenizer(r'\w+')  
 tokens = tokenizer.tokenize(sentence)  
 return nltk.word\_tokenize(" ".join(tokens))  
  
  
# Stopwords removal, return list  
def stop\_words\_remove(tokens):  
 stop = stopwords.words('english')  
 new\_tokens = [i for i in tokens if i not in stop]  
 return new\_tokens  
  
def remove\_numbers(words):  
 *"""Replace all interger occurrences in list of tokenized words with textual representation"""* new\_words = []  
 for word in words:  
 new\_word = re.sub(r'\d+','',word)  
 if new\_word != '':  
 new\_words.append(new\_word)  
 return new\_words  
  
# Stemming of tokens, return list  
def stem\_tokens(new\_tokens):  
 ps = PorterStemmer()  
 stemmed = []  
 for i in new\_tokens:  
 stemmed.append(ps.stem(i))  
 return stemmed  
  
  
inv\_file = r'Inverted.csv'  
  
  
def get\_relevance(n, nw):  
 return (n - nw + 0.5) / (nw + 0.5)  
  
  
def get\_probability\_matrix(n, df, toks):  
 prob\_matrix = {}  
  
 for i in toks:  
 nw = df.loc[i, 'Occurences'].count(')')  
  
 prob\_matrix[i] = [nw, get\_relevance(n, nw)]  
 return prob\_matrix  
  
  
def get\_query\_tokens(query):  
 tokens = preprocess(query.lower())  
 tokens = remove\_numbers(tokens)  
 tokens = stop\_words\_remove(tokens)  
 tokens = stem\_tokens(tokens)  
 return tokens  
  
  
def get\_conditional\_probability(qtok, inv\_file):  
 prob\_matrix = {}  
  
 df = pd.read\_csv(inv\_file)  
 toks = list(df['Tokens'])  
  
 df.set\_index('Tokens', inplace=True)  
  
 word\_matrix = get\_probability\_matrix(len(xfiles), df, toks)  
  
 # print(word\_matrix)  
 for i in xfiles:  
 flag = False  
 val = 1  
 prob\_matrix[i] = 0  
  
 for j in qtok:  
 if j in toks:  
 if i in df.loc[j, 'Occurences']:  
 flag = True  
 val \*= word\_matrix[j][1]  
 prob\_matrix[i] = val if flag else 0  
  
 return prob\_matrix  
  
  
print("Program 4: \n\tProbablistic Model Implementation")  
  
# print("Query : ")  
  
# kldsjflksajdk = get\_query\_tokens(input())  
print("Enter your query here :", end=" ")  
out1=get\_query\_tokens(input())  
print(out1)  
rel\_docs = get\_conditional\_probability(out1, inv\_file)  
  
rel\_docs = {k: "{0:.5f}".format(v) for k, v in sorted(rel\_docs.items(), key=lambda item: item[1], reverse=True)}  
  
# print(vect)  
print("The documents in the order of relevance to the query are as follows: ")  
print(pd.DataFrame(rel\_docs.items(), columns=['File', 'Relevance']))

**input/output:**

Program 4:

Probabilistic Model Implementation

Enter your query here: Sunshine ten

The documents in the order of relevance to the query are as follows:

File Relevance

0 T7.txt 3.40000

1 T6.txt 1.00000

2 T1.txt 0.29412

3 T2.txt 0.29412

4 T3.txt 0.29412

5 T4.txt 0.29412

6 T5.txt 0.29412

7 T9.txt 0.29412

8 T10.txt 0.29412

9 T8.txt 0.00000

Process finished with exit code 0

**Inference/Conclusion:**

Probabilistic model is used in a corpus where the set of relevant documents for a query is predetermined and appropriate formula is applied to find the most relevant document for a given query.

**Exercise 5:** Implementation of various evaluation measures.

1. Calculate recall and precision values for all relevant documents and draw precision vs recall curve. Also calculate R-precision.
2. Compare performance of two IR algorithms for the same query q.
3. Calculate harmonic mean and E-measure (All three cases b=1, b>1, b<1)

**Aim:** To build a program that executes above tasks sequentially.

**Steps to execute (a):**

*Step 1:* Assume a set of relevant documents and a set of retrieved documents for a query q.

*Step 2:* Calculate the recall and precision value for each document in the retrieved set.

*Step 3:* Plot a curve with recall in x-axis and precision in y-axis using calculated values.

*Step 4:* Calculate the R-precision value which is equal to total documents in the retrieved set that belong to the relevant set divided by the total number of documents in the relevant set.

**Steps to execute (b):**

*Step 1:* Assume 5 queries were made with each query having it’s own set of relevant documents and the list of documents retrieved by the algorithm A and algorithm B.

*Step 2:* Calculate the R-precision value for each algorithm for each query.

*Step 3:* Subtract the R-precision value of algorithm B from algorithm A and store the value in an array X.

*Step 4:* Plot the array X into a histogram with x-axis being the query number and y-axis being R-precision A/B.

*Step 5:* If the sum of elements in array X is positive print algorithm A is better else print algorithm B is better.

**Steps to execute (c):**

*Step 1:* Assume a query for which relevant documents set and a list of Retrieved documents for an algorithm is available.

*Step 2:* Calculate the recall and precision for each document in the relevant set found in the retrieved list.

*Step 3:* Calculate the harmonic mean, E-precision for b value equal to 1 , 0.2 and 2 respectively and save the output in the form of a table.

**Code (python 3):**

import matplotlib.pyplot as plt  
# matplotlib.use("gtk")  
import pandas as pd

"""  
5.a : Recall Precision graph for the following relevant documents and documents retrieved  
  
Rq= {d3,d5, d9,d25,d39,d44,d56,d71,d89,d94,d105,d119,d124,d136, d144}  
Aq ={d123,d84,d56,d6,d8,d9,d511,d129,d187,d25,d38,d48,d250,d113 , d44,d99,d95,d214,d136,d39,d128,d71,d14,d5}  
  
"""  
# Relevant documents set  
# rq = [3,5,9,25,39,44,56,71,89,123]  
rq = [3, 5, 9, 25, 39, 44, 56, 71, 89, 94, 105, 119, 124, 136, 144]  
  
# Answer set  
  
aq = [123, 84, 56, 6, 8, 9, 511, 129, 187, 25, 38, 48, 250, 113, 44, 99, 95, 214, 136, 39, 128, 71, 14, 5]  
# aq = [123,84,56,6,8,9,511,129,187,25,38,48,250,113,3]  
  
# Recall list initialization  
recall = []  
  
# Precision list initialization  
precision = []  
  
rlen = len(rq)  
alen = len(aq)  
  
recallCount = 0  
  
# to keep track of the retrieved documents  
retrievedDocumentCount = 0  
# pc = 0  
  
rr = 0  
pr = 0  
  
for i in aq:  
 retrievedDocumentCount += 1  
 if i in rq:  
 recallCount += 1  
 rr = recallCount / rlen  
  
 pr = recallCount / retrievedDocumentCount  
  
 # print(rr,pp,recall\_count,precision\_count)  
 recall.append(rr \* 100)  
 precision.append(pr \* 100)  
  
print("\n\nThe R-precision value is :", rr)  
dashline = "\n\n---------------------------------------------------"  
print(dashline)  
  
plt.plot(recall, precision, color='orange')  
plt.title('Recall Precision curve')  
plt.xlabel('Recall')  
plt.ylabel('Precision')  
plt.xlim(0, 115)  
plt.ylim(0, 115)  
plt.show()

# 5.b: r-precision comparison for to different algorithms for 5 different queries  
  
algo\_a = []  
algo\_b = []  
  
  
def r\_precision(a, b, r):  
 c1 = 0  
 c2 = 0  
 for i in a:  
 if i in r:  
 c1 += 1  
 for i in b:  
 if i in r:  
 c2 += 1  
  
 algo\_a.append(c1 / len(a))  
 algo\_b.append(c2 / len(b))  
  
  
# list of the 5 queries, relevant documents, documents retrieved by algorithm a and b respectively  
rel1 = [3, 5, 9, 25, 39, 44, 56, 71, 89, 123]  
algoA1 = [123, 84, 56, 6, 8, 9, 511, 129, 187, 25, 38, 48, 250, 113, 3]  
algoB1 = [12, 39, 13, 123, 8, 9, 19, 89, 87, 25, 70, 71, 29, 44, 3]  
  
rel2 = [3, 20, 5, 68, 51, 21, 27, 64, 6, 93]  
algoA2 = [3, 13, 5, 68, 51, 67, 32, 64, 45, 6, 94, 95, 93]  
algoB2 = [20, 30, 7, 78, 21, 27, 14, 15, 16, 6, 54, 4, 6]  
  
rel3 = [38, 65, 73, 88, 93, 74, 36, 4, 28, 30]  
algoA3 = [66, 88, 45, 43, 23, 12, 188, 200, 34, 4]  
algoB3 = [56, 73, 65, 3, 2, 99, 146, 93, 76, 74, 4]  
  
rel4 = [85, 95, 25, 64, 52, 12, 43, 18, 6, 66]  
algoA4 = [52, 62, 64, 77, 12, 45, 18, 43, 6]  
algoB4 = [95, 85, 25, 77, 123, 3213, 78, 18, 6]  
  
rel5 = [9, 76, 78, 31, 7, 47, 30, 8, 43, 51]  
algoA5 = [76, 75, 31, 7, 30, 44, 56, 50, 94, 223]  
algoB5 = [78, 9, 48, 47, 4, 31, 43, 56, 55, 99, 123, 222]  
  
  
r\_precision(algoA1, algoB1, rel1)  
r\_precision(algoA2, algoB2, rel2)  
r\_precision(algoA3, algoB3, rel3)  
r\_precision(algoA4, algoB4, rel4)  
r\_precision(algoA5, algoB5, rel5)  
  
print('\n\n')  
print('algorithm a is better' if sum(algo\_a) > sum(algo\_b) else 'algorithms b is better')  
print(dashline)  
  
# algo\_a = [0.3,0.6,0.3,0.5,1,0.78,0.24]  
# algo\_b = [0.1,0.3,0.6,0.4,0,0.7,0.01]  
  
x = map(lambda a, b: a - b, algo\_a, algo\_b)  
x = list(x)  
  
fig = plt.figure(figsize=(10, 10))  
langs = [i for i in range(1, len(x) + 1)]  
  
plt.xlabel("Query Number")  
plt.ylabel("R Precision A/B")  
plt.title("Precision Histogram")  
  
plt.bar(langs, list(x), color='purple', width=0.5)  
  
plt.show()  
  
# 5.c : Harmonic Mean and E-Measure  
  
Rq = ['d3', 'd5', 'd9', 'd25', 'd39', 'd44', 'd56', 'd71', 'd89', 'd123']  
A1 = ['d123', 'd84', 'd56', 'd6', 'd8', 'd9', 'd511', 'd129', 'd187', 'd25', 'd38', 'd48', 'd250', 'd113', 'd3']  
  
  
def calhme(Rq, Aq):  
 rel\_doc\_count = 0  
 rn = len(Rq)  
 recall, precision, harmonic\_mean, em1, em2, em0 = {}, {}, {}, {}, {}, {}  
  
 for i in range(len(Aq)):  
 if Aq[i] in Rq:  
 rel\_doc\_count += 1  
 recall[Aq[i]] = (round(rel\_doc\_count / rn, 2))  
 precision[Aq[i]] = (round(rel\_doc\_count / (i + 1), 2))  
 harmonic\_mean[Aq[i]] = round(2 / ((1 / recall[Aq[i]]) + (1 / precision[Aq[i]])), 2)  
 em0[Aq[i]] = round(1 - harmonic\_mean[Aq[i]], 2)  
 # Set b=2 for E-Measure  
 b = 2  
 em1[Aq[i]] = round(1 - ((1 + (b \*\* 2)) / (((b \*\* 2) / recall[Aq[i]]) + (1 / precision[Aq[i]]))), 2)  
  
 b = 0.2  
 em2[Aq[i]] = round(1 - ((1 + (b \*\* 2)) / (((b \*\* 2) / recall[Aq[i]]) + (1 / precision[Aq[i]]))), 2)  
  
 else:  
 pass  
  
  
 return pd.DataFrame({'Recall': pd.Series(recall), 'Precision': pd.Series(precision), 'Harmonic mean': pd.Series(harmonic\_mean),  
 'E-Measure (b=1)': pd.Series(em0), 'E-Measure (b>1)': pd.Series(em1),  
 'E-Measure (b<1)': pd.Series(em2)})  
  
  
# Harmonic Mean and E-Measure  
resultDataframe = calhme(Rq, A1)  
print()  
print()  
print(resultDataframe)  
# resultDataframe.to\_csv('5(c).csv')

**Input/output:**

1. ***Console output:***

The R-precision value is : 0.5333333333333333

---------------------------------------------------

algorithms b is better

---------------------------------------------------

Recall Precision ... E-Measure (b>1) E-Measure (b<1)

d123 0.1 1.00 ... 0.88 0.26

d56 0.2 0.67 ... 0.77 0.39

d9 0.3 0.50 ... 0.67 0.51

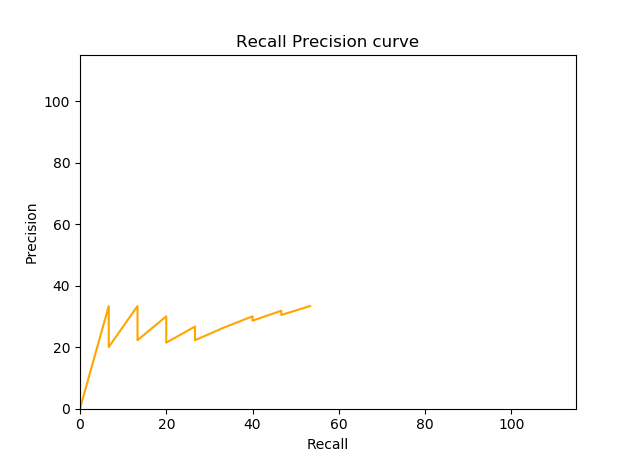
d25 0.4 0.40 ... 0.60 0.60

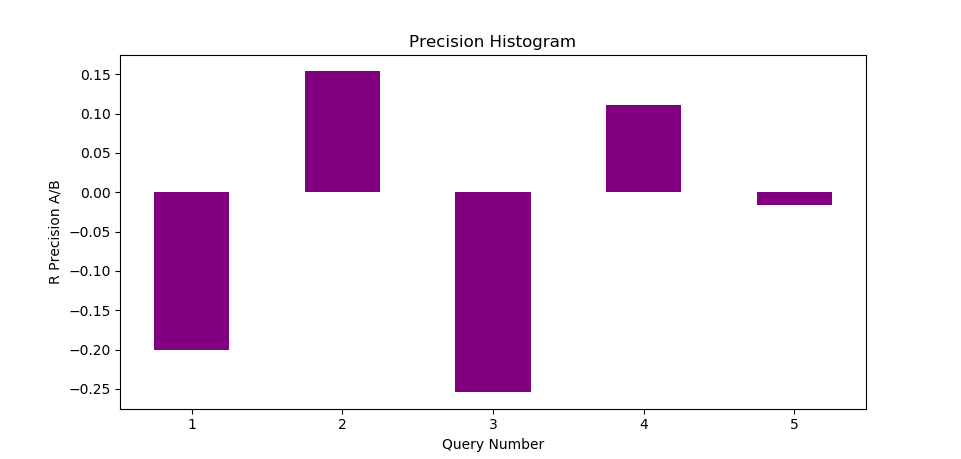
d3 0.5 0.33 ... 0.55 0.67

[5 rows x 6 columns]

Process finished with exit code 0

***Graph output:***





***Saved table:***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Recall | Precision | Harmonic mean | E-Measure (b=1) | E-Measure (b>1) | E-Measure (b<1) |
| d123 | 0.1 | 1 | 0.18 | 0.82 | 0.88 | 0.26 |
| d56 | 0.2 | 0.67 | 0.31 | 0.69 | 0.77 | 0.39 |
| d9 | 0.3 | 0.5 | 0.37 | 0.63 | 0.67 | 0.51 |
| d25 | 0.4 | 0.4 | 0.4 | 0.6 | 0.6 | 0.6 |
| d3 | 0.5 | 0.33 | 0.4 | 0.6 | 0.55 | 0.67 |

**Inference/Conclusion:**

Precision and Recall are one of the measures to determine if the algorithm used to obtain relevant documents is good or not. R-precision can be used to compare the algorithms, Harmonic mean and E-Measures are more refined measures to determine how good an algorithm is.