AIWIR Assignment - 2 (UE19CS322)

Team Details - Team 4

S. No.	Name	Section	SRN
1	Nidhi Gupta	D	PES2UG19CS256
2	Nikhil M Adyapak	D	PES2UG19CS257
3	Manoj Kumar	D	PES2UG19CS222

Done by - NIDHI GUPTA

Link to Colab Notebook with results:

https://colab.research.google.com/drive/1nB-FAbfrdc02Wt4--TZh2S554uDNYqsh#scrollTo=4Iwt1Y9rYBku

Dataset used : Amazon Fine Food Reviews

Link to dataset -> https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

#dataset https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

Code for importing the dataset after mounting my google drive

```
#import pandas to read the csv file
import pandas as pd
#reading the csv file
df = pd.read_csv('/content/drive/MyDrive/Reviews.csv')
```

Checking content of the dataset

#printing the info of the dataset
df.info()

```
[ ] #printing the info of the dataset
    df.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 568454 entries, 0 to 568453
    Data columns (total 10 columns):
     #   Column                  Non-Null Count   Dtype
    ---  ------                  --------------   -----
     0   Id                      568454 non-null  int64
     1   ProductId               568454 non-null  object
     2   UserId                  568454 non-null  object
     3   ProfileName             568438 non-null  object
     4   HelpfulnessNumerator    568454 non-null  int64
     5   HelpfulnessDenominator  568454 non-null  int64
     6   Score                   568454 non-null  int64
     7   Time                    568454 non-null  int64
     8   Summary                 568427 non-null  object
     9   Text                    568454 non-null  object
    dtypes: int64(5), object(5)
    memory usage: 43.4+ MB
```

Getting a sample of the dataset

#getting a sample of 3 rows
df.sample(3)

```
[ ] #getting a sample of 3 rows
    df.sample(3)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
94802	94803	B000SATIBO	A2LXTY704ZWBK0	Hepius	0	1	2	1312329600	Bad lemon tea	This did not taste like Earl Grey. It tasted l...
455823	455824	B007HERNSO	A3OWU1R4LAL8GF	GirlX ""Addicted to Amazon""	7	7	5	1301875200	Excellent choice for my finicky feline!	Since my elderly cat has started eating a wet ...
547982	547983	B00522VKWI	A185FYY1Z4JZQZ	Elena Marie	0	0	5	1350432000	great flavor	My cousin uses Maxwell House Instant coffee an...

Importing Modules

#Import modules

```
import pandas as pd
import nltk
from nltk import word_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
import re
import timeit
import sys
import time

lemmatizer = WordNetLemmatizer()
tokenizer = RegexpTokenizer("[\w']+")
stop_words = set(stopwords.words('english'))
stop_words = stop_words.union(",","(",")","[","]","{","}","#","@","!",":",";",".","?")
```

Casefolding, Tokenization, Stop-word removal, Stemming, Lemmatization

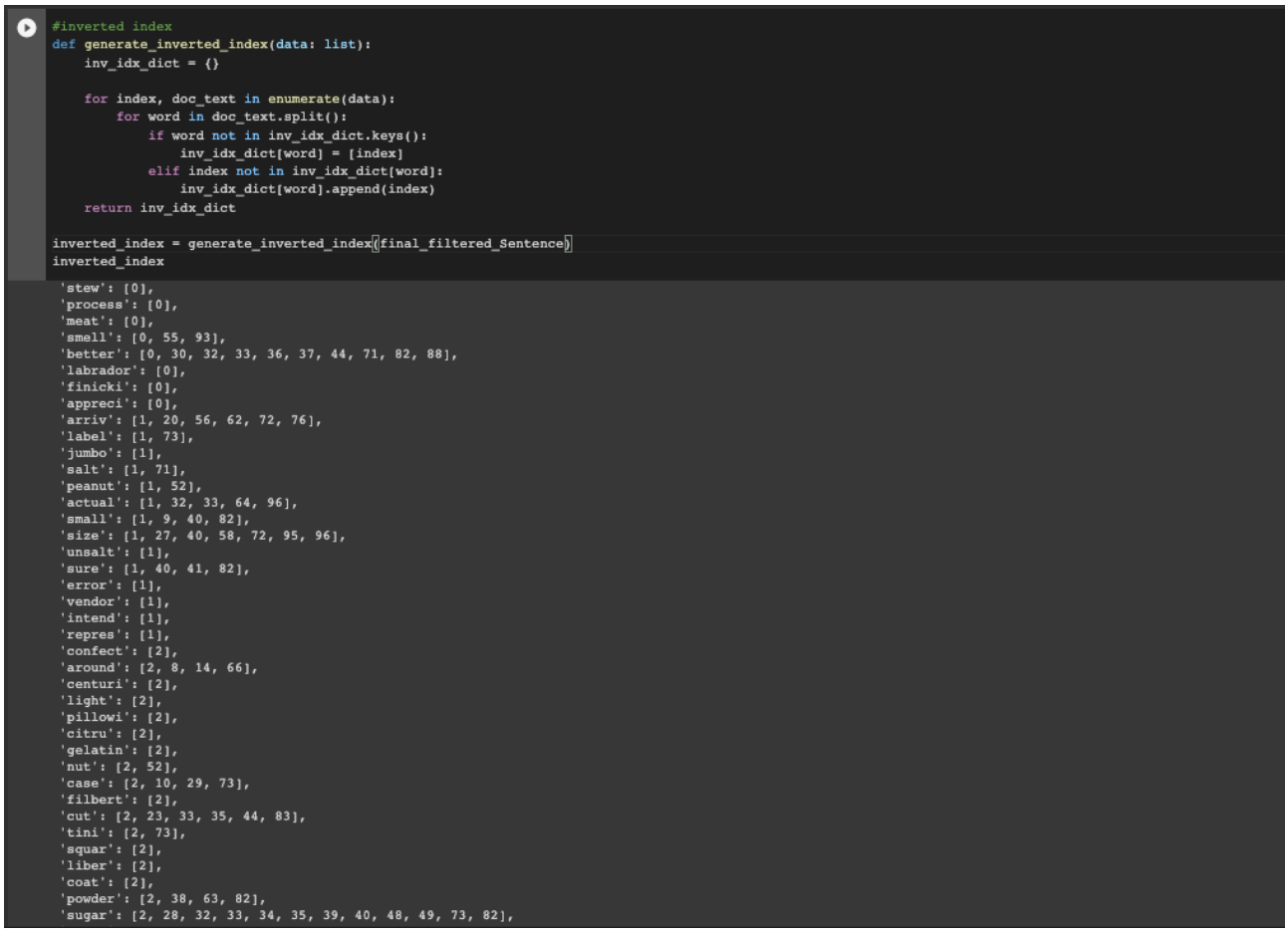
```
#Converting all words to lower case
text = text.str.lower() #CASEFOLDING
#text

filtered_Sentence=[]
ps = PorterStemmer()
#Using RegEx
Cond1 = r'@[A-Za-z0-9]+'
Cond2 = r'https?://[A-Za-z0-9./]+'
combined_Cond = r'|'.join((Cond1, Cond2))
Cond3 = r'[^a-zA-Z0-9]'
finalCombinedCond = r'|'.join((combined_Cond,Cond3))
for i in range(0, len(text)):
    txt = re.sub(finalCombinedCond,' ',text[i])
    txt = word_tokenize(txt) #TOKENIZATION
    txt = [word for word in txt if not word in set(stopwords.words('english'))] #STOPWORD
REMOVAL
    #print(txt)
    txt = ' '.join(txt)
    filtered_Sentence.append(txt)
```

```
ps = PorterStemmer()
def stemSentence(sentence):
    token_words=word_tokenize(sentence)
    #token_words
    stem_sentence=[]
    for word in token_words:
        stem_sentence.append(ps.stem(word)) #STEMMING
        stem_sentence.append(" ")
    return "".join(stem_sentence)

lemmatizer = WordNetLemmatizer()
def lemmatizeSentence(sentence):
    token_words=word_tokenize(sentence)
    #token_words
    lem_sentence=[]
    for word in token_words:
        word1 = lemmatizer.lemmatize(word, pos = "n")
        word2 = lemmatizer.lemmatize(word1, pos = "v")
        word3 = lemmatizer.lemmatize(word2, pos = ("a"))
        lem_sentence.append(word1)
        #stem_sentence.append(lemmatizer.lemmatize(word))
        lem_sentence.append(" ")
    return "".join(lem_sentence)

new_filtered_Sentence=[]

for i in filtered_Sentence:
    j = stemSentence(i)
    new_filtered_Sentence.append(j)

#new_filtered_Sentence

final_filtered_Sentence=[]
for i in new_filtered_Sentence:
    j = lemmatizeSentence(i) #LEMMATISATION
    final_filtered_Sentence.append(j)

final_filtered_Sentence
```

```
```

Inverted Index

```
#inverted index
def generate_inverted_index(data: list):
    inv_idx_dict = {}

    for index, doc_text in enumerate(data):
        for word in doc_text.split():
            if word not in inv_idx_dict.keys():
                inv_idx_dict[word] = [index]
            elif index not in inv_idx_dict[word]:
                inv_idx_dict[word].append(index)
    return inv_idx_dict

inverted_index = generate_inverted_index(final_filtered_Sentence)
inverted_index
```



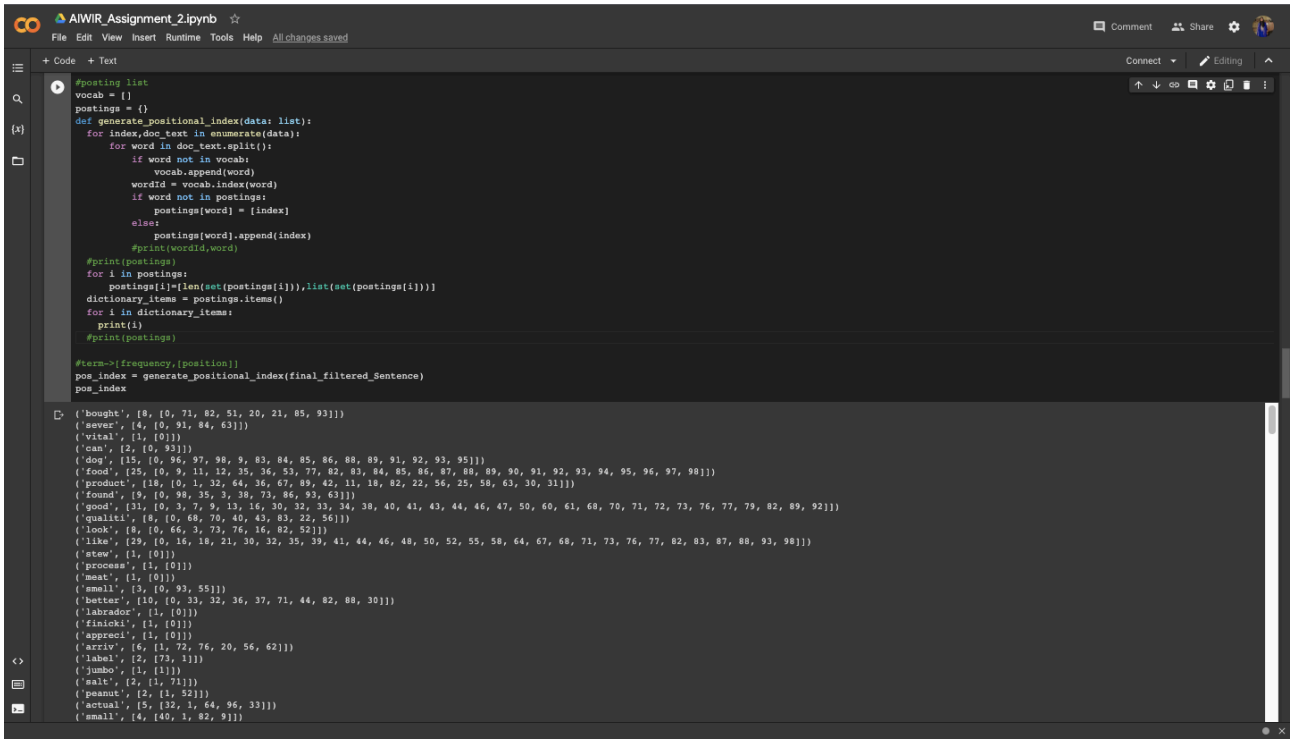
```
#inverted index
def generate_inverted_index(data: list):
    inv_idx_dict = {}

    for index, doc_text in enumerate(data):
        for word in doc_text.split():
            if word not in inv_idx_dict.keys():
                inv_idx_dict[word] = [index]
            elif index not in inv_idx_dict[word]:
                inv_idx_dict[word].append(index)
    return inv_idx_dict

inverted_index = generate_inverted_index(final_filtered_Sentence)
inverted_index
```

```
 'stew': [0],
 'process': [0],
 'meat': [0],
 'smell': [0, 55, 93],
 'better': [0, 30, 32, 33, 36, 37, 44, 71, 82, 88],
 'labrador': [0],
 'finicki': [0],
 'appreci': [0],
 'arriv': [1, 20, 56, 62, 72, 76],
 'label': [1, 73],
 'jumbo': [1],
 'salt': [1, 71],
 'peanut': [1, 52],
 'actual': [1, 32, 33, 64, 96],
 'small': [1, 9, 40, 82],
 'size': [1, 27, 40, 58, 72, 95, 96],
 'unsalt': [1],
 'sure': [1, 40, 41, 82],
 'error': [1],
 'vendor': [1],
 'intend': [1],
 'repres': [1],
 'confect': [2],
 'around': [2, 8, 14, 66],
 'centuri': [2],
 'light': [2],
 'pillowi': [2],
 'citru': [2],
 'gelatin': [2],
 'nut': [2, 52],
 'case': [2, 10, 29, 73],
 'filbert': [2],
 'cut': [2, 23, 33, 35, 44, 83],
 'tini': [2, 73],
 'squar': [2],
 'liber': [2],
 'coat': [2],
 'powder': [2, 38, 63, 82],
 'sugar': [2, 28, 32, 33, 34, 35, 39, 40, 48, 49, 73, 82],
```

Posting List Creation

```
#posting list
vocab = []
postings = {}
def generate_positional_index(data: list):
  for index,doc_text in enumerate(data):
      for word in doc_text.split():
          if word not in vocab:
              vocab.append(word)
          wordId = vocab.index(word)
          if word not in postings:
```

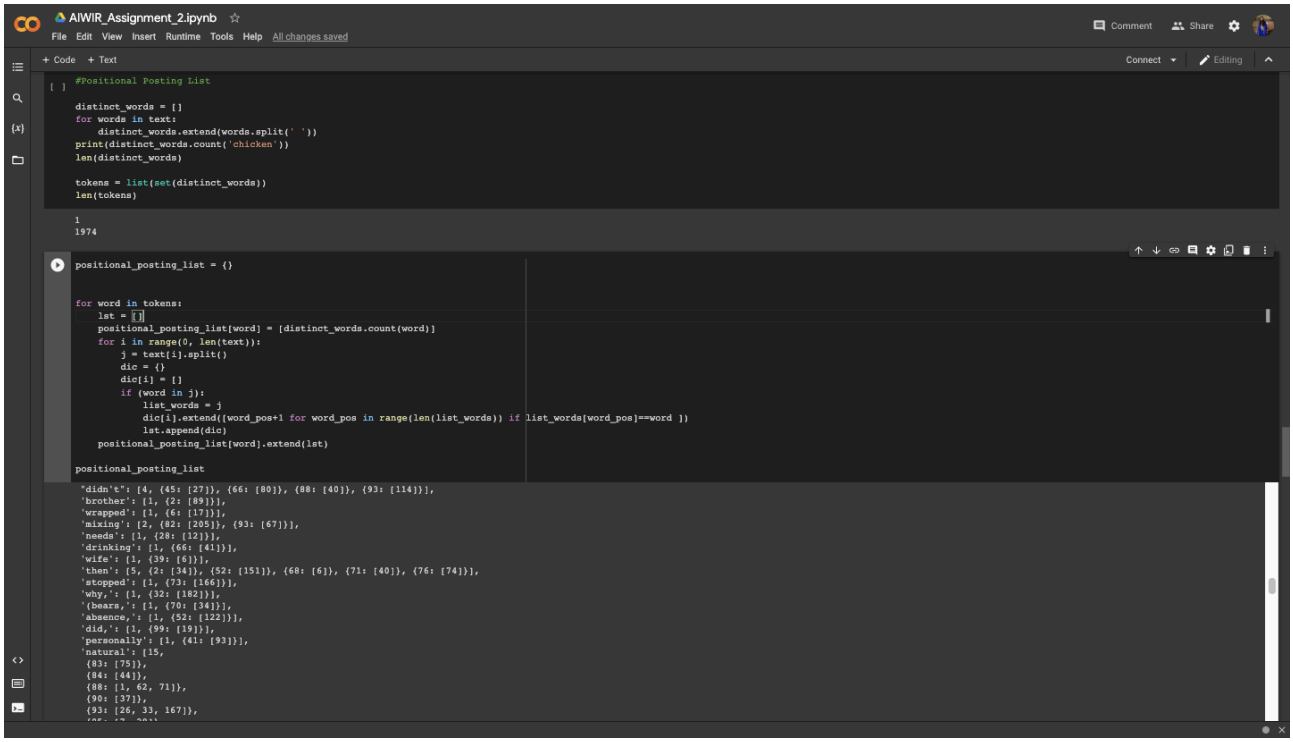
```
            postings[word] = [index]
        else:
            postings[word].append(index)
        #print(wordId,word)
  #print(postings)
  for i in postings:
     postings[i]=[len(set(postings[i])),list(set(postings[i]))]
  dictionary_items = postings.items()
  for i in dictionary_items:
    print(i)
  #print(postings)

#term->[frequency,[position]]
pos_index = generate_positional_index(final_filtered_Sentence)
pos_index
```



Positional Posting List

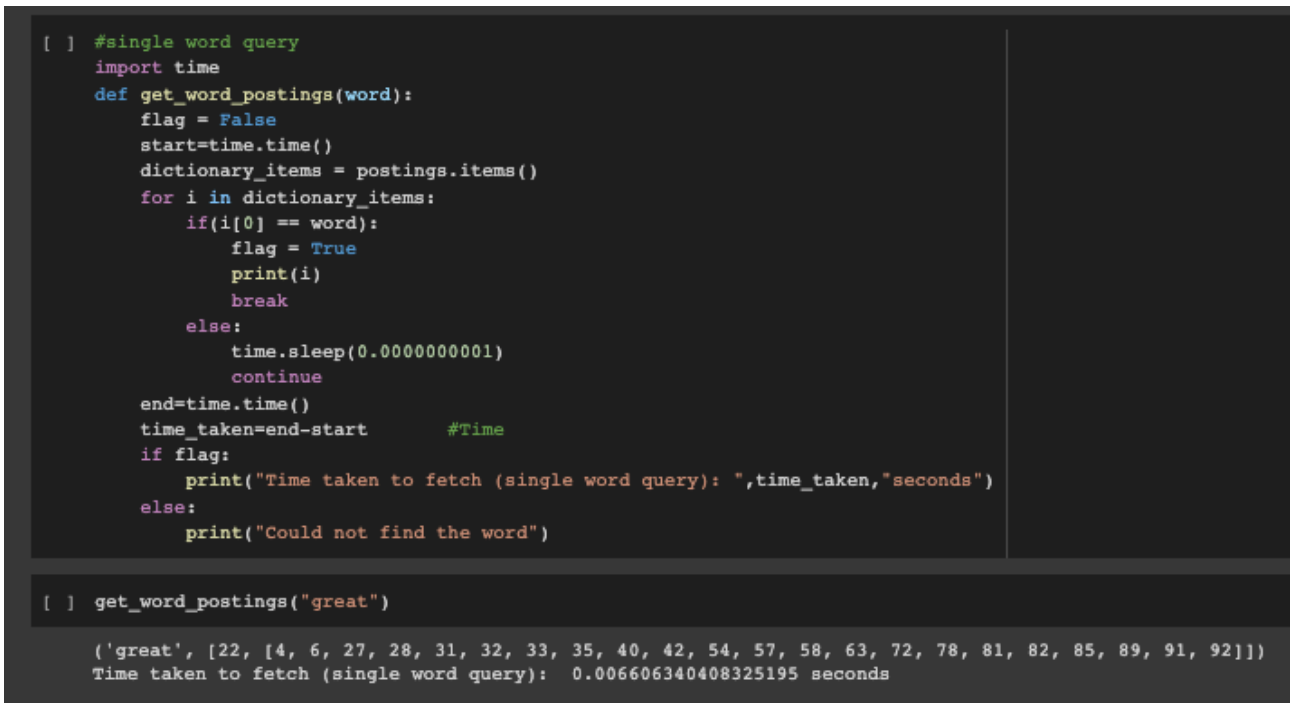
```
#Positional Posting List

distinct_words = []
for words in text:
    distinct_words.extend(words.split(' '))
print(distinct_words.count('chicken'))
len(distinct_words)

tokens = list(set(distinct_words))
len(tokens)

positional_posting_list = {}


for word in tokens:
    lst = []
    positional_posting_list[word] = [distinct_words.count(word)]
```

```
    for i in range(0, len(text)):
        j = text[i].split()
        dic = {}
        dic[i] = []
        if (word in j):
            list_words = j
            dic[i].extend([word_pos+1 for word_pos in range(len(list_words)) if
list_words[word_pos]==word ])
            lst.append(dic)
    positional_posting_list[word].extend(lst)

positional_posting_list
```



Single Word Query

```
#single word query
import time
def get_word_postings(word):
    flag = False
    start=time.time()
    dictionary_items = postings.items()
    for i in dictionary_items:
        if(i[0] == word):
            flag = True
            print(i)
            break
        else:
            time.sleep(0.0000000001)
            continue
    end=time.time()
    time_taken=end-start       #Time
    if flag:
        print("Time taken to fetch (single word query): ",time_taken,"seconds")
    else:
        print("Could not find the word")
```



```
[ ]  #single word query
     import time
     def get_word_postings(word):
         flag = False
         start=time.time()
         dictionary_items = postings.items()
         for i in dictionary_items:
             if(i[0] == word):
                 flag = True
                 print(i)
                 break
             else:
                 time.sleep(0.0000000001)
                 continue
         end=time.time()
         time_taken=end-start       #Time
         if flag:
             print("Time taken to fetch (single word query): ",time_taken,"seconds")
         else:
             print("Could not find the word")

[ ]  get_word_postings("great")

     ('great', [22, [4, 6, 27, 28, 31, 32, 33, 35, 40, 42, 54, 57, 58, 63, 72, 78, 81, 82, 85, 89, 91, 92]])
     Time taken to fetch (single word query):  0.006606340408325195 seconds
```

Boolean Query (Intersection)

```
#boolean query (Intersection)
def get_intersection_postings(word1, word2):
    flag = False
    start=time.time()
    #locating words in postings dictionary
    required = []
    answer = {}
    dictionary_items = postings.items()
    for i in dictionary_items:
        if(i[0] == word1):
            required.append(i)
        if(i[0] == word2):
            required.append(i)
        else:
            continue

    #print(required)

    indexes = []
    list1 = []
    list2 = []

    #Finding the intersection
    for i in required:
        #print(i)
        word, posting2 = i
        #print(posting2)
        frequency, index = posting2[0], posting2[1]
        #print(index)
        indexes.append(index)
        #print(indexes)

    list1, list2 = indexes[0], indexes[1]

    #print(list1)
```

```
    #print(list2)
    list3 = [value for value in list1 if value in list2]
    #print(list3)
    #answer[word1+ " AND " + word2]=[len(set(list3)),list(set(list3))]
    answer[word1+ " AND " + word2]= list(set(list3))

    end=time.time()
    time_taken=end-start       #Time

    if len(list3):
        print(answer)
        print("Time taken to fetch (boolean query): ",time_taken,"seconds")
    else:
        print("No intersection possible")
```

```
[ ]  get_intersection_postings("dog","food")

     {'dog AND food': [0, 97, 98, 96, 9, 83, 84, 85, 86, 88, 89, 91, 92, 93, 95]}
     Time taken to fetch (boolean query):  0.0003948211669921875 seconds
```

Boolean Query (Union)

```
#boolean query (Union)
def get_union_postings(word1, word2):
    flag = False
    start=time.time()
    #locating words in postings dictionary
    required = []
    answer = {}
    dictionary_items = postings.items()
    for i in dictionary_items:
      if(i[0] == word1):
          required.append(i)
      if(i[0] == word2):
          required.append(i)
      else:
        continue
    #print(required)

    indexes = []
    list1 = []
    list2 = []
    #Finding the union
    for i in required:
      #print(i)
      word, posting2 = i
      #print(posting2)
      frequency, index = posting2[0], posting2[1]
      #print(index)
      indexes.append(index)
    #print(indexes)

    list1, list2 = indexes[0], indexes[1]

    #print(list1)
```

```
    #print(list2)
    list3 = list1 + list2
    #print(list3)
    #answer[word1+ " OR " + word2]=[len(set(list3)),list(set(list3))]
    answer[word1+ " OR " + word2]= list(set(list3))
    end=time.time()
    time_taken=end-start       #Time
    if len(list3):
      print(answer)
      print("Time taken to fetch (boolean query): ",time_taken,"seconds")
    else:
      print("No Union possible")
```

```
[ ] get_union_postings("dog","food")

    {'dog OR food': [0, 9, 11, 12, 35, 36, 53, 77, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98]}
    Time taken to fetch (boolean query):  0.0004322528839111328 seconds
```

Phrase Query

```
#For finding if two words occur together and in which document.
def get_phrase_query(phrase):

    start=time.time()
    str_to_process = phrase.split()

    i=0
    j=0

    lim1=0
    lim2=0

    ans=[]
    if (str_to_process[0] in postings) and (str_to_process[1] in postings):
        while (lim1<len(postings[str_to_process[0]][1]) and lim2<len(postings[str_to_process[1]][1]) ):
            if(postings[str_to_process[0]][1][i] == postings[str_to_process[1]][1][j]):
                ans.append(postings[str_to_process[1]][1][j])
                i+=1
                j+=1

            elif (postings[str_to_process[0]][1][i] < postings[str_to_process[1]][1][j]):
                i+=1

            else:
                j+=1

            lim1+=1
            lim2+=1

    else:
        print("Not found in any tweet")

    final_tweets_id=[]
    pos_idx = []
    for p in ans:
        held_for_now=filtered_Sentence[p].split()
```

```
    if( held_for_now.index(str_to_process[0]) == (held_for_now.index(str_to_process[1])-1) ):
        final_tweets_id.append(p)
        pos_idx.append(len(final_tweets_id))
        pos_idx.append(final_tweets_id)

end=time.time()
time_taken=end-start

print("The phrase is present in tweet ids:",pos_idx)
print("Time taken to fetch the phrase query: ",time_taken,"seconds")
```