

## 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=4lwt1Y9rYBku>

**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	B000SATIBD	A2LXTY704ZWBK0	Hephus	0	1	2	1312329800	Bad lemon tea	This did not taste like Earl Grey. It tasted L...
455823	455824	B007HERNSD	A3OWU1R4LAL8GF	GirlX "&#34;Addicted to Amazon&#34;"	7	7	5	1301875200	Excellent choice for my finicky feline!	Since my elderly cat has started eating a wet ...
547982	547983	B00522VKWI	A185FYY1Z4JQZ	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
```

```

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

final_filtered_Sentence

['bought sever vital can dog food product found good qualitti product look like stew process meat smell better labrador finicki appreci product better ',
'product arriv label jumbo salt peanut peanut actual small size unsalt sure error vendor intend repres product jumbo ',
'connect around century light pillow citi galatin nut case filbert cut tini sugar liker coat powder sugar tini mouth heaven chervi flavor highli recommend yummi treat familiar stori c lewi lion witch wardrob t
'look secret ingredi robittusish believ found got addit root beer extract order good made cherri soda flavor medicin ',
'great taffi great price wide assort yummi taffi deliveri quick taffi lover deal ',
'got wild hair taffi order five pound bag taffi enjoy mani flavor watermelon root beer melon peppermint grape etc complaint bit much red black licoric flavor piec particular favorit kid husband last two week wo
'saltwat taffi great flavor soft chedi candi individua wrap well none candi stuck together happen expens version fraling would highli recommend candi serv beach theme parti everyon love ',
'taffi good soft chervi flavor amaz would definit recommend buy satisfi ',
'right mostli sprout cat eat grass love rotat around wheatgrass rye ',
'healthi dog food good digest also good small puppi dog eat requir amount everi feed ',
'know cactu tegula unigu combin ingredi flavour hot sauc make one kind pick bottli trip brought back home u total blown away realis simpli find anywher citi bum br hr magic internet case sauc ecstat br hr love
'one boy need lose weight put food floor chubby guy protein rich product food higher skinni boy jump higher food sit go stale realli go food chubby boy lose ounce week ',
'cat happili eat felidia platinum two year got new bag shape food differ tri new food first put bowl bowl sit full kitti touch food notic similar review relat formula chang past unfortun need find new food cat e
'good flavor came secue pack fresh delici love twizzler ',
'strawberri twizzler quitti pleassur yummi six pound around son ',
'daughter love twizzler shipment six pound realli hit spot exactli would expect six packag strawberri twizzler ',
'love eat good watch tv look movi sweet like transfer zip lock baggi stay fresh take time eat ',
'satisfi twizzler purchas share other enjoy definit order ',
'twizzler strawberri childhood favorit candi made lancast pennsylvania candi inc one oldest confectioneri firm unit state subsidiari hershey compani compani establish 1945 young smyli also make appli licoric twi
'candi deliv fast purchas reason price home bound unabl get store perfect ',
'husband twizzler addict bought mani time amazon govern employe live overseas get countri assign alway fresh tasti pack well arriv time manner ',
'bought husband current overseas love appar staff like also hr gener amount twizzler 16 ounce bag well worth price href twizzler strawberri 16 ounce bag pack 6 ',
'rememb buy candi kid qualitti drop year stilli superb product disappoint ',
'love candi weight watcher cut back stilli crave ',
'live u 'yr miss twizzler go back visit noston visit alway stock say yum hr sell mexico faith buyer often abl buy right ',
'product recealy advertis br hr href twizzler strawberri 16 ounce bag pack 6 ',
'candi red flavor plan chervi would never buy ',
'glad amazon carri batteri hard time find elsewher unigu size need garag door open hr great deal price ',
'got mum diabet need watch sugar intak father simpli choos limit unnecessary sugar intak one sweet tooth love toffe would never guess sugar free great eat pretti much guilt free impress order w dark chocol take
'know cactu tegula unigu combin ingredi flavour hot sauc make one kind pick bottli trip brought back home u total blown away realis simpli find anywher citi bum br hr magic internet case sauc ecstat br hr love
'never huge coffe fan howev mother purchas littli machin talk tri latt macolato coffe shop better one like product usual non coffe drinker hr littli dolch questo machin super easi use prepar realli good coffe lat
'offer great price great tast thank amazon sell product br hr staral ',
'mocann instant oatmeal great must oatmeal scrape togeth two three minut prepar escop fact howev even best instant oatmeal nowher near good even store brand oatmeal requir stovetop prepar stilli mocann good get
'good instant oatmeal best oatmeal brand use cane sugar instead high fructuos corn syrup better sweet doctor say form sugar better great cold morn time make mocann steel cut oat appli cinnamon best mapl brown ou
'instant oatmeal becom soggi minut water hit bowl mocann instant oatmeal hold textur excel flavor good time mocann regular oat meal excel may take bit longer prepar time morn best instant brand ever eaten clos
'mocann instant irish oatmeal variet pack regular appli cinnamon mapl brown sugar 10 count box pack 6 br hr fan mocann steel cut oat thought give instant variet tri found hardi meal sweet great folk like post
'u celiac disease product lifesav could better get almost half price groceri health food store lowe mocann instant oatmeal flavor br hr thank br abbi ',
'al need know oatmeal instant nake half cup low fat milk add raisin nake 90 second expens kroger store brand oatmeal mayb littli tastier better textur smeth stilli oatmeal mm conveni ',
'visit friend nate morn coffe came storag room packet mocann instant irish oatmeal suggest tri use stash sometim nate dose give chanc say end tri appli cinn found tasteful made water powder milk goe good j coffe
'order wife recommend daughter almost everi morn like flavor happi happi br href mocann instant irish oatmeal variet pack regular appli cinnamon mapl brown sugar 10 count box pack 6 ',
'variet pack tast great br hr everi morn 0 30 cent per meal understand everyon earth buy stuff hr br mapl brown sugar terrif follow appli cinnamon follow regular get tire ole thing tast great hr hr boll water s
'mocann nake oatmeal everi oatmeal commisseur whether one like raw pellet state cook half hour sloth addi instant done microwav three minut good sure beauti instant variet avail differ flavor well regular br
'mocann oatmeal everi morn order amazon abl save almost 3 00 per box br great product tast great healthi ',
'mocann oatmeal good qualitti choic favorit appli cinnamon find none overli sugari good hot breakfast 2 minut excel ',
'realli like mocann steel cut oat find cook often br tast much better groceri store brand conveni br anyth keep eat oatmeal regularli good thing ',
'seem littli wholemeal supermarket brand somewhat mushi quit much flavo either pas muster kid probabi buy ',
'good oatmeal like appli cinnamon best though follow direct packag sinc alway come soupi tast could sinc like oatmeal realli thick add milk top ',
'flavor good howev see differo oaker oat brand mushi ',

```

## 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],
'pillow': [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

```

The screenshot shows a Jupyter Notebook titled "AIWIR\_Assignment\_2.ipynb". The code in the cell defines a function `generate_positional_index` that takes a list of documents and returns a dictionary of word positions. The function iterates over each document, splits it into words, and updates a vocabulary and a postings dictionary. The output of the function is a dictionary where each word is mapped to a list of [frequency, [positions]].

```

#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[word] = {}
            wordId = vocab[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

```

```

('bought', [8, [0, 71, 82, 51, 20, 21, 85, 93]])
('sever', [4, [0, 91, 84, 63]])
('vital', [1, [0]])
('can', [2, [0, 93]])
('dog', [15, [0, 96, 97, 98, 9, 83, 84, 85, 86, 88, 89, 91, 92, 93, 95]])
('food', [25, [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]])
('product', [18, [0, 1, 32, 64, 36, 67, 89, 42, 11, 18, 82, 22, 56, 25, 50, 63, 30, 31]])
('found', [5, [0, 94, 35, 3, 30, 73, 86, 93, 63]])
('good', [13, [0, 3, 7, 9, 13, 16, 30, 32, 33, 34, 38, 40, 41, 43, 44, 46, 47, 50, 60, 41, 68, 70, 71, 72, 73, 76, 77, 79, 82, 89, 92]])
('quality', [8, [0, 68, 70, 40, 43, 83, 22, 56]])
('look', [8, [0, 66, 3, 73, 76, 16, 82, 52]])
('like', [29, [0, 15, 18, 21, 30, 32, 35, 39, 41, 44, 46, 48, 50, 52, 55, 58, 64, 67, 68, 71, 73, 76, 77, 82, 83, 87, 88, 93, 98]])
('stew', [1, [0]])
('process', [1, [0]])
('meat', [1, [0]])
('small', [13, [0, 93, 95]])
('better', [10, [0, 33, 32, 36, 37, 71, 44, 82, 88, 30]])
('labrador', [1, [0]])
('finicky', [1, [0]])
('appreci', [1, [0]])
('arriv', [6, [1, 72, 76, 20, 56, 62]])
('label', [2, [73, 1]])
('jumbo', [1, [1]])
('salt', [2, [1, 71]])
('peanut', [2, [1, 52]])
('actual', [5, [12, 1, 64, 96, 33]])
('small', [5, [40, 1, 82, 91]])

```

## 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

```

```

#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)

1
1974

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

{'didn\'t': [4, (45: [27]), (66: [80]), (88: [40]), (93: [114])],
 'brother': [1, (2: [89])],
 'stopped': [1, (6: [173])],
 'mining': [2, (82: [205]), (93: [67])],
 'needs': [1, (28: [12])],
 'drinking': [1, (66: [41])],
 'wife': [1, (39: [61])],
 'then': [5, (2: [34]), (52: [151]), (68: [6]), (71: [40]), (76: [74])],
 'stopped': [1, (73: [166])],
 'why': [1, (32: [182])],
 'beats': [1, (70: [241])],
 'absence': [1, (52: [122])],
 'did': [1, (98: [19])],
 'personally': [1, (41: [93])],
 'natural': [15,
 (83: [75]),
 (84: [44]),
 (88: [1, 42, 71]),
 (90: [37]),
 (93: [26, 33, 167])],
 ...}

```

## 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")
```

```
[ ] get_phrase_query("fed golden")
```

```
The phrase is present in tweet ids: [1, [99]]
```

```
Time taken to fetch the phrase query: 1.71661376953125e-05 seconds
```