# 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 - Manoj Kumar K**

**Link to Colab Notebook with results:**

https://colab.research.google.com/drive/1MTMkjv7Mpo4EMGKq_lcGgq5eWqrJ2iKZ

**Dataset used** : Hotel Reviews

Link to dataset -> https://www.kaggle.com/datasets/andrewmvd/trip-advisor-hotel-reviews

#dataset https://www.kaggle.com/datasets/andrewmvd/trip-advisor-hotel-reviews

**Code for importing the dataset after mounting my google drive**

```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/
tripadvisor_hotel_reviews.csv')
```

**Checking content of the dataset**

```
df.info()
```

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20491 entries, 0 to 20490
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Review  20491 non-null  object
 1   Rating  20491 non-null  int64
dtypes: int64(1), object(1)
memory usage: 320.3+ KB
```

**Getting a sample of the dataset**

```
df.sample(10)
```

```
df.sample(10)
```

| | Review | Rating |
|---|---|---|
| 15166 | good value money, wife stayed hotel 5 nights a... | 4 |
| 2020 | great hotel price booked hotel stay hotwire, I... | 4 |
| 8508 | montebello splendid staff efficient courteous ... | 5 |
| 1429 | overpriced poor value money just returned long... | 2 |
| 15198 | taste elegance post nite azamara southeast asi... | 5 |
| 6683 | soooo relaxing, went punta cana sept 17-sept 2... | 4 |
| 6054 | got married palace loved little background wif... | 5 |
| 19275 | nice hotel n't expect great design hotel exagg... | 4 |
| 16244 | probably good value spent 5 nights crown twin ... | 4 |
| 12832 | good bu, second stay clean friendly convenient... | 3 |

**Importing Modules**

```
import nltk
from nltk import word_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
import re
```

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

```
#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))
```

```
filtered_Sentence=[]
for i in range(0, len(text)):
    reviews = re.sub(finalCombinedCond,' ',text[i])
    reviews = word_tokenize(reviews) #TOKENIZATION
    reviews = [word for word in reviews if not word in
set(stopwords.words('english'))] #STOPWORD REMOVAL
    reviews = ' '.join(reviews)
    filtered_Sentence.append(reviews)
```

```python
#Text after stemming
Stem_words = []
ps = PorterStemmer()

listToStr = ' '.join(map(str, filtered_Sentence))
words = word_tokenize(listToStr)

for i in words:
    rootWord = ps.stem(i)
    Stem_words.append(rootWord)

#text after applying lemmatization
lemma_word = []
wordnet_lemmatizer = WordNetLemmatizer()
for w in filtered_Sentence:
    word1 = wordnet_lemmatizer.lemmatize(w, pos = "n")
    word2 = wordnet_lemmatizer.lemmatize(word1, pos = "v")
    word3 = wordnet_lemmatizer.lemmatize(word2, pos =
("a"))
    lemma_word.append(word1)
print("The text after lemmatization")
lemma_word
```

```python
#text after applying lemmatization
lemma_word = []


wordnet_lemmatizer = WordNetLemmatizer()
for w in filtered_Sentence:
    word1 = wordnet_lemmatizer.lemmatize(w, pos = "n")
    word2 = wordnet_lemmatizer.lemmatize(word1, pos = "v")
    word3 = wordnet_lemmatizer.lemmatize(word2, pos = ("a"))
    lemma_word.append(word1)
print("The text after lemmatization")

lemma_word
```

```
The text after lemmatization
['nice hotel expensive parking got good deal stay hotel anniversary arrived late evening took advice previous reviews valet parking check quick easy little d
 'ok nothing special charge diamond member hilton decided chain shot 20th anniversary seattle start booked suite paid extra website description suite bedroom
 'nice rooms 4 experience hotel monaco seattle good hotel n 4 level positives large bathroom mediterranean suite comfortable bed pillowsattentive housekeepin
 'unique great stay wonderful time hotel monaco location excellent short stroll main downtown shopping area pet friendly room showed signs animal hair smells
 'great stay great stay went seahawk game awesome downfall view building n complain room huge staff helpful booked hotels website seahawk package charge park
 'love monaco staff husband stayed hotel crazy weekend attending memorial service best friend husband celebrating 12th wedding anniversary talk mixed emotion
 'cozy stay rainy city husband spent 7 nights monaco early january 2008 business trip chance come ride booked monte carlo suite proved comfortable longish st
 'excellent staff housekeeping quality hotel chocked staff make feel home experienced exceptional service desk staff concierge door men maid service needs wo
 'hotel stayed hotel monaco cruise rooms generous decorated uniquely hotel remodeled pacific bell building charm sturdiness everytime walked bell men felt li
 'excellent stayed hotel monaco past w e delight reception staff friendly professional room smart comfortable bed particularly liked reception small dog rece
 'poor value stayed monaco seattle july nice hotel priced 100 150 night hotel takes beating quotient experience simply average nothing exceptional paying 300
 'nice value seattle stayed 4 nights late 2007 looked comparable hilton marriott westin area points miles n gave monaco shot pleasantly surprised nice room s
 'nice hotel good location hotel kimpton design whimsical vibe fun staff young casual problem hotel busy stay friendly helpful group reserved rooms gave conn
 'nice hotel nice staff hotel lovely staff quite rude bellhop desk clerk going way make things difficult waited forever check heavy bags help getting through
 'great hotel night quick business trip loved little touches like goldfish leopard print robe complaint wifi complimentary internet access business center gr
 'horrible customer service hotel stay february 3rd 4th 2007my friend picked hotel monaco appealing website online package included champagne late checkout 3
 'disappointed say anticipating stay hotel monaco based reviews seen tripadvisor definitely disppointment decor room hotel envisioned nice housekeeping staff
```

**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(filtered_Sentence)
inverted_index
```

```
{'nice': [0,
  1,
  2,
  3,
  7,
  10,
  11,
  12,
  13,
  16,
  18,
  19,
  22,
  28,
  34,
  35,
  36,
  39,
  41,
  45,
  47,
  55,
  56,
  57,
  60,
  64,
  66,
  67,
  74,
  78,
  81,
  84,
  89,
  92,
  94,
  95,
  96,
  99],
 'hotel': [0,
  1,
  2,
  3,
  4,
  5
```

**Positional Posting List**

```
#positional index
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)
  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(filtered_Sentence)
pos_index

('nice', [38, [0, 1, 2, 3, 7, 10, 11, 12, 13, 16, 18, 19, 22, 28, 34, 35, 36, 39, 41, 45, 47, 55, 56, 57, 60, 64, 66, 67, 74, 78, 81, 84, 89, 92, 94, 95, 96,
('hotel', [83, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 22, 23, 25, 26, 27, 28, 29, 32, 33, 35, 36, 37, 38, 39, 40, 41, 43, 44,
('expensive', [8, [0, 38, 84, 85, 54, 89, 94, 31]])
('parking', [19, [0, 4, 12, 27, 31, 32, 35, 36, 39, 40, 41, 66, 67, 71, 84, 89, 90, 93, 99]])
('got', [19, [0, 1, 2, 4, 13, 15, 16, 25, 41, 45, 46, 56, 61, 67, 79, 85, 89, 92, 99]])
('good', [46, [0, 1, 2, 4, 5, 6, 7, 12, 16, 17, 18, 22, 24, 28, 29, 30, 32, 33, 34, 35, 38, 41, 42, 49, 53, 56, 57, 66, 67, 68, 72, 79, 80, 81, 82, 83, 84, 8
('deal', [3, [0, 67, 79]])
('stay', [57, [0, 1, 2, 3, 4, 5, 6, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 24, 26, 27, 32, 35, 36, 39, 40, 45, 52, 53, 55, 56, 59, 60, 61, 62, 67, 71, 7
('anniversary', [4, [0, 1, 83, 5]])
('arrived', [5, [0, 1, 36, 82, 30]])
('late', [8, [0, 1, 11, 76, 15, 79, 92, 30]])
('evening', [8, [0, 64, 4, 12, 80, 85, 30, 63]])
('took', [10, [0, 1, 99, 4, 7, 13, 15, 19, 20, 92]])
('advice', [4, [0, 10, 92, 7]])
('previous', [3, [0, 82, 45]])
('reviews', [11, [0, 96, 42, 45, 46, 16, 48, 55, 56, 25, 61]])
('valet', [6, [0, 66, 15, 52, 89, 90]])
('check', [19, [0, 2, 4, 13, 15, 16, 28, 33, 40, 45, 46, 54, 72, 73, 80, 87, 92, 96, 99]])
('quick', [8, [0, 71, 11, 45, 14, 20, 22, 92]])
('easy', [12, [0, 2, 36, 6, 17, 51, 20, 57, 59, 28, 61, 31]])
('little', [21, [0, 8, 12, 14, 16, 19, 27, 30, 45, 47, 51, 56, 61, 67, 72, 75, 85, 86, 89, 96, 97]])
('disappointed', [4, [0, 16, 10, 71]])
('non', [5, [0, 2, 75, 87, 28]])
('existent', [1, [0]])
('view', [33, [0, 1, 4, 6, 17, 20, 23, 28, 30, 33, 34, 35, 37, 39, 41, 50, 51, 52, 54, 58, 59, 60, 61, 62, 63, 75, 78, 80, 84, 85, 87, 89, 90]])
('room', [80, [0, 1, 2, 3, 4, 5, 6, 8, 9, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 23, 25, 26, 27, 28, 30, 31, 32, 34, 35, 36, 37, 39, 40, 41, 43, 45, 46, 47,
('clean', [30, [0, 1, 6, 12, 20, 25, 26, 28, 29, 30, 35, 37, 44, 51, 52, 53, 62, 66, 68, 72, 74, 78, 81, 84, 86, 90, 92, 94, 98, 99]])
('size', [6, [0, 67, 12, 17, 49, 29]])
```

**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
    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
    if flag:
        print("Time taken to fetch (single word query): ",time_taken,"seconds")
    else:
        print("Could not find the word")
```

```
get_word_postings("advice")
```

```
('advice', [4, [0, 10, 92, 7]])
Time taken to fetch (single word query):  0.010023355484008789 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

    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("centrally","excellent")
```

```
{'centrally AND excellent': [23]}
Time taken to fetch (boolean query):  0.0008854866027832031 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("centrally","excellent")

    {'centrally OR excellent': [3, 7, 9, 17, 20, 21, 23, 27, 29, 30, 35, 43, 48, 50, 52, 55, 59, 61, 62, 64, 86, 95, 98]}
    Time taken to fetch (boolean query):  0.00081634521484375 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("hotel monaco")

    The phrase is present in tweet ids: [1, [2, 3, 9], 2, [2, 3, 9], 3, [2, 3, 9]]
    Time taken to fetch the phrase query:  0.00013756752014160156 seconds
```