**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE: DJ19ITL504**  **DATE: 4/11/22**

**COURSE NAME: Artificial Intelligence Laboratory**  **CLASS: TYBTech-IT**

**EXPERIMENT NO. 6**

**CO/LO:**Apply NLP algorithms and methods to solve domain-specific problems

**AIM:**To perform Text Processing on a particular dataset using NLTK.

**DESCRIPTIONOF EXPERIMENT:**

1. Steps in Text Processing
2. Sentence Tokenization
3. Word Tokenization
4. Text Lemmatization and Stemming
5. Removing Stop Words
6. Regular Expressions
7. Bag of Words
8. Term Frequency – Inverse Document Frequency (TF-IDF)
9. Bag of Words

* Bag of words is a Natural Language Processing technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents.
* A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a “bag” of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

1. TF-IDF

* One approach of scoring words is to rescale the frequency of words by how often they appear in all documents so that the scores for frequent words like “the” that are also frequent across all documents are penalized. This approach is called term frequency-inverse document frequency or shortly known as TF-IDF approach of scoring.

**TECHNOLOGY STACK USED: python**

**SOURCE CODE and OUTPUT:**

1. **Tokenization by word, sentence, paragraph :**

doc=open("this.txt" , "r")

def tokenizationWords(para):

    print("tokenization by words : ")

    para.replace(".","")

    print(para.split())

def tokenizationSentence(para):

    print("tokenization by sentence: ")

    l=para.split(".")

    l.remove("")

    print(l)

def tokenizationPara(para):

    print("tokenization by para : ")

    l=para.split("\n")

    l.remove("")

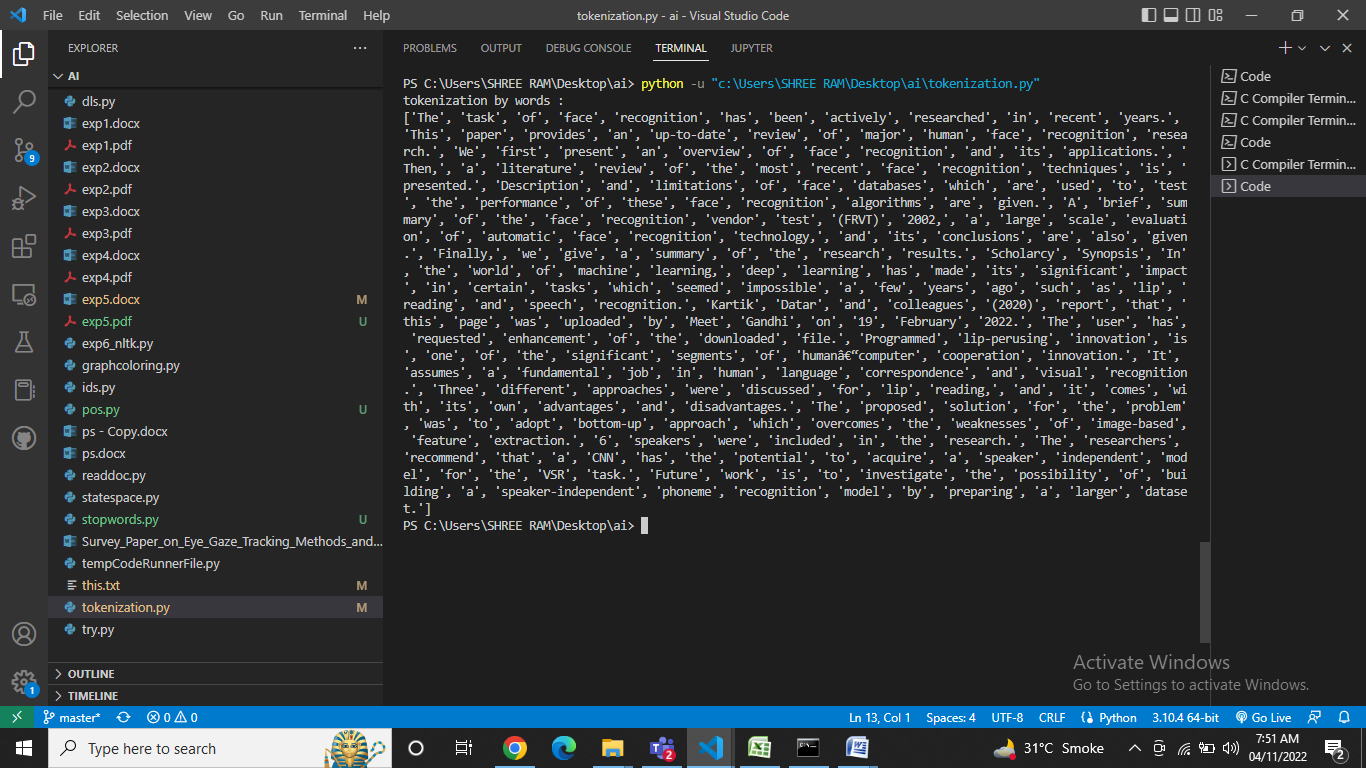
    print(l)

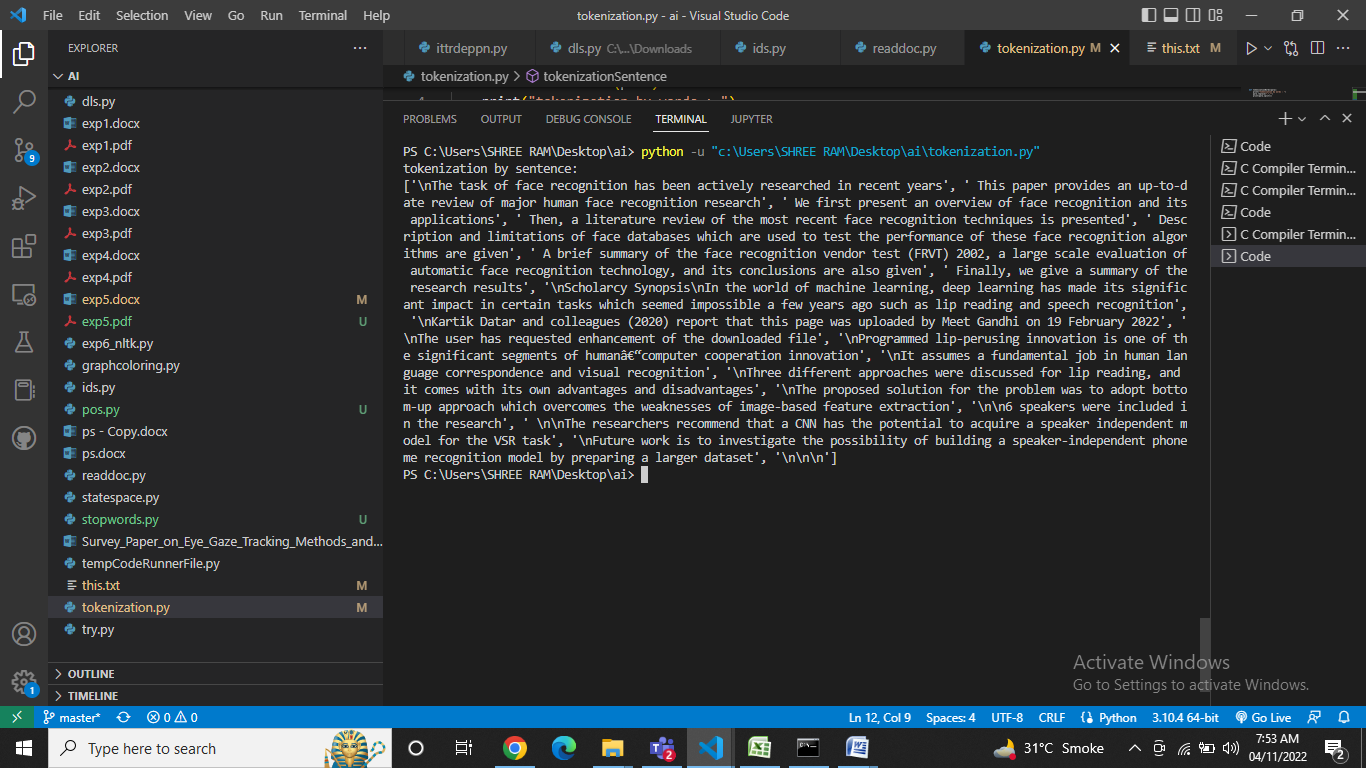
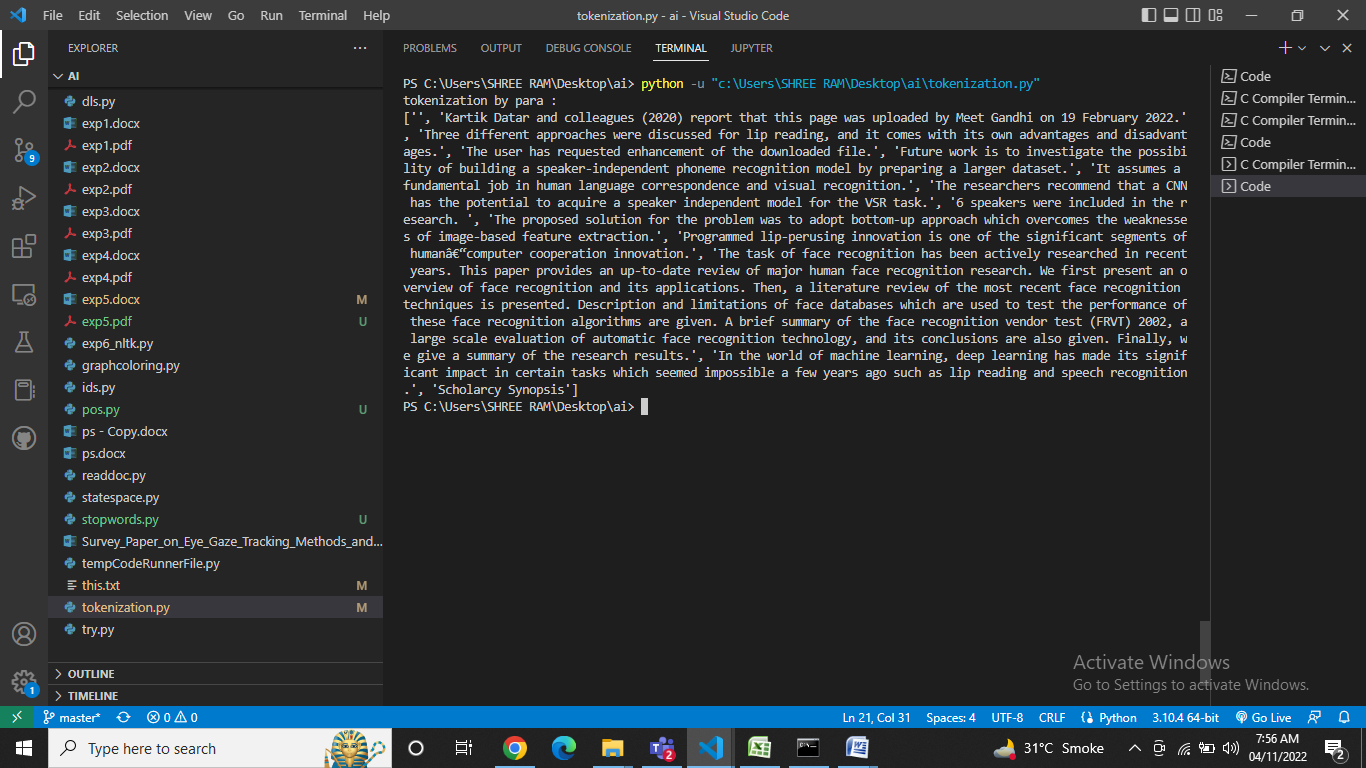
#tokenizationWords(doc.read())

#tokenizationSentence(doc.read())

tokenizationPara(doc.read())

**output:**

****

****

1. **Stemming and Lemmatization:**

from nltk.stem.snowball import SnowballStemmer

from nltk.stem.wordnet import WordNetLemmatizer

def stemming(words):

    print("#Stemming is a process of normalization, in which words are reduced to their root word (or) stem : ")

    print("stemming words : ",words)

    snow\_stem = SnowballStemmer(language='english')

    for word in words:

        print(word + '--->' + snow\_stem.stem(word))

def lemminization(tokenized\_word):

    print("lemmatization is also used to reduce the word to their root word. Lemmatizing gives the complete meaning of the word which makes sense.It uses vocabulary and morphological analysis to transform a word into a root word. ")

    print("lemminization words : ",tokenized\_word)

    lemmatizer = WordNetLemmatizer()

    lemmatized\_words\_list = []

    for each\_word in tokenized\_word:

        lem\_word = lemmatizer.lemmatize(each\_word)

        lemmatized\_words\_list.append(lem\_word)

    print('Text with Stop Words: {}'.format(tokenized\_word))

    print('Lemmatized Words list {}'.format(lemmatized\_words\_list))

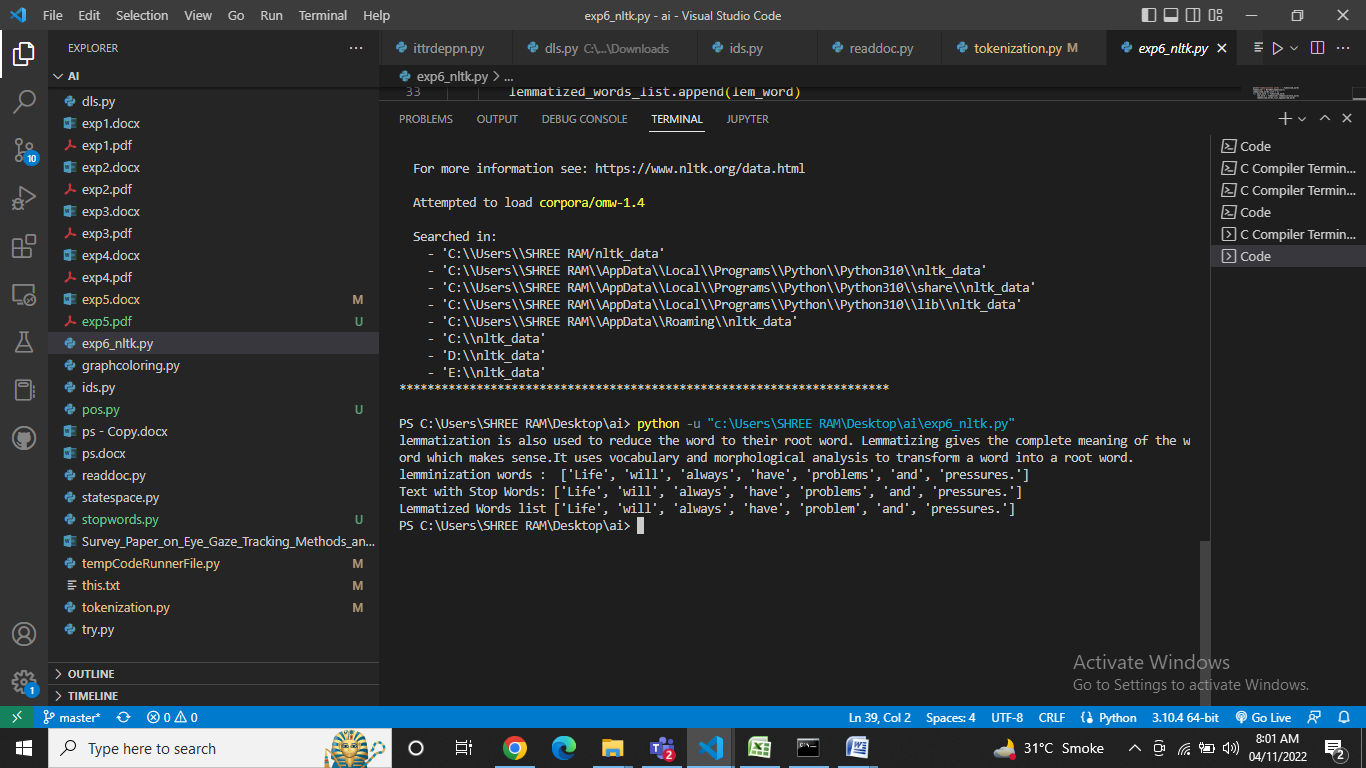
words = ['happy', 'happier', 'happiest', 'happiness', 'breathing','fairly','eating']

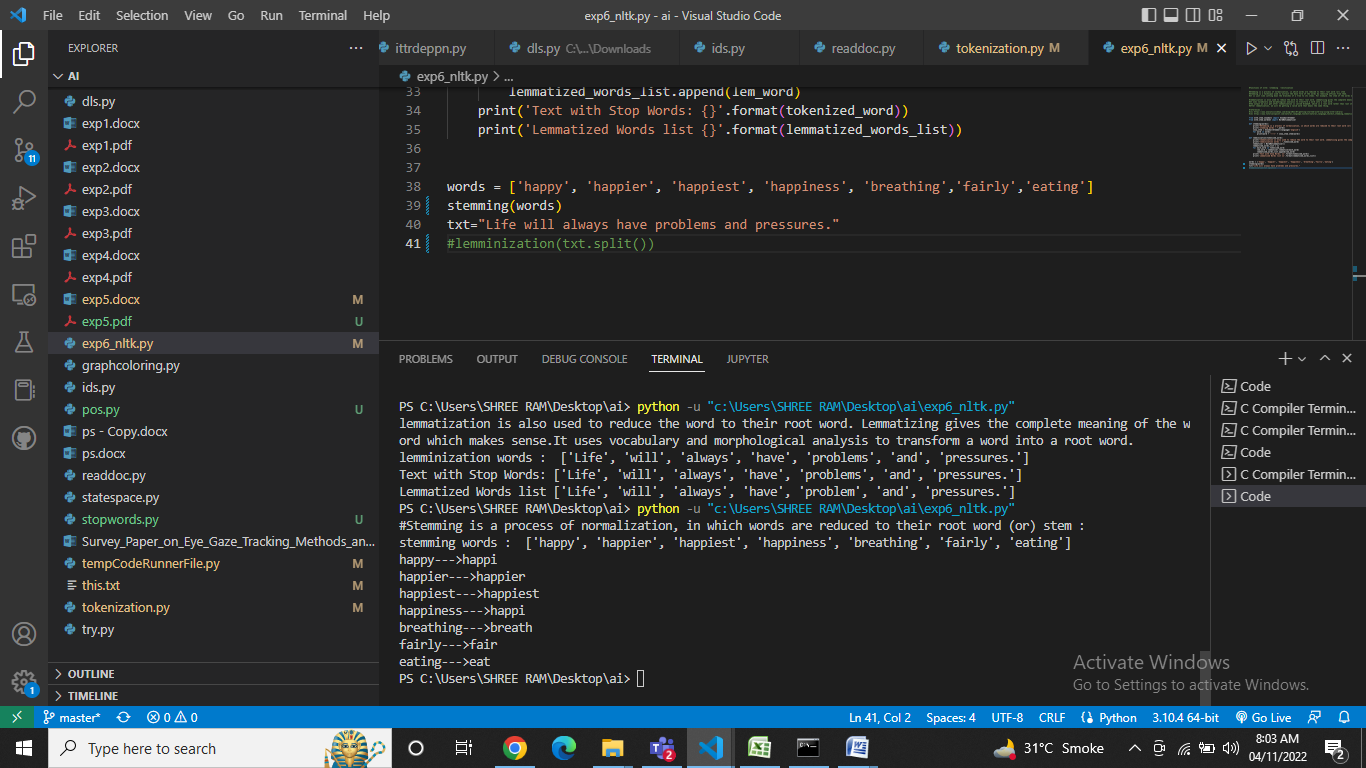
stemming(words)

txt="Life will always have problems and pressures."

#lemminization(txt.split())

**Output:**

****

****

1. **Stop words:**

from nltk import word\_tokenize

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

stop\_words=set(stopwords.words("english"))

text = 'Learn to lose your destiny to find where it leads you'

filtered\_text = []

tokenized\_word = word\_tokenize(text)

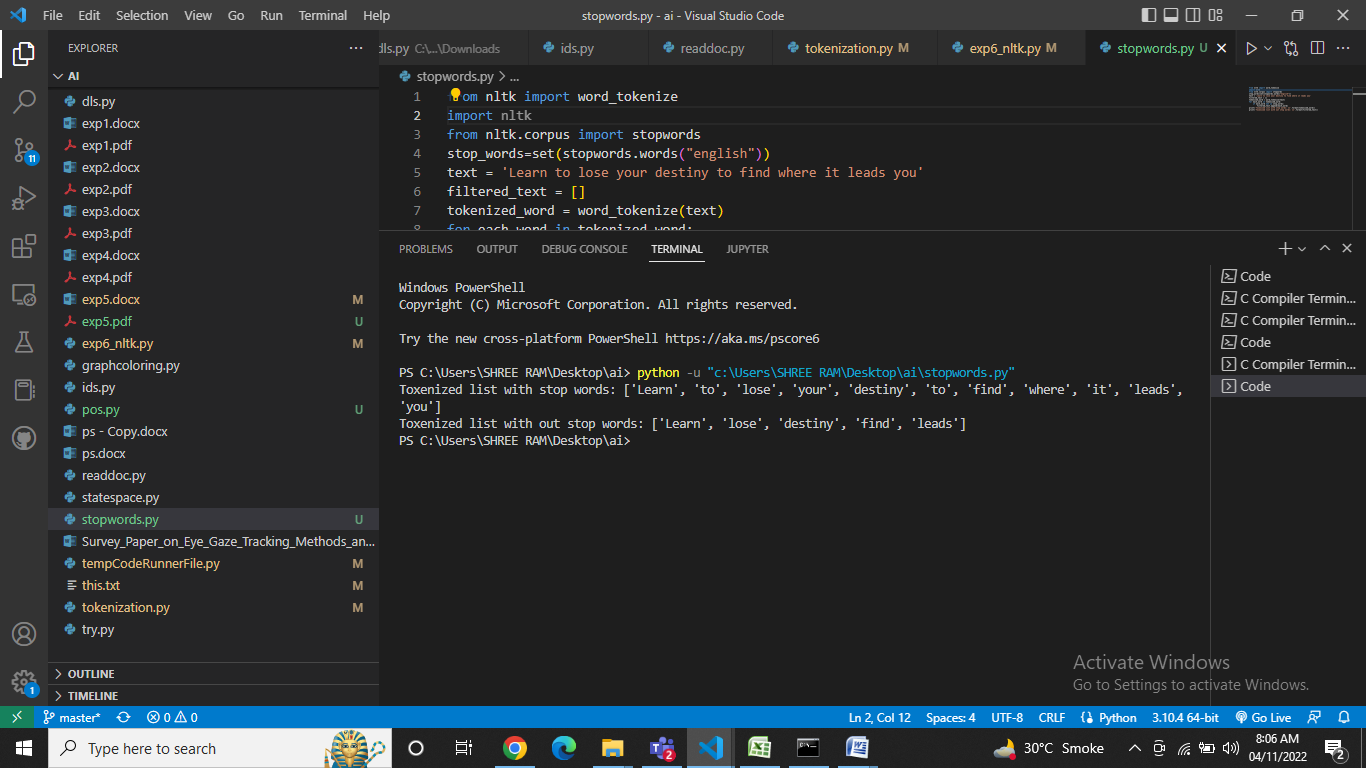
for each\_word in tokenized\_word:

    if each\_word not in stop\_words:

        filtered\_text.append(each\_word)

print('Toxenized list with stop words: {}'.format(tokenized\_word))

print('Toxenized list with out stop words: {}'.format(filtered\_text))

****

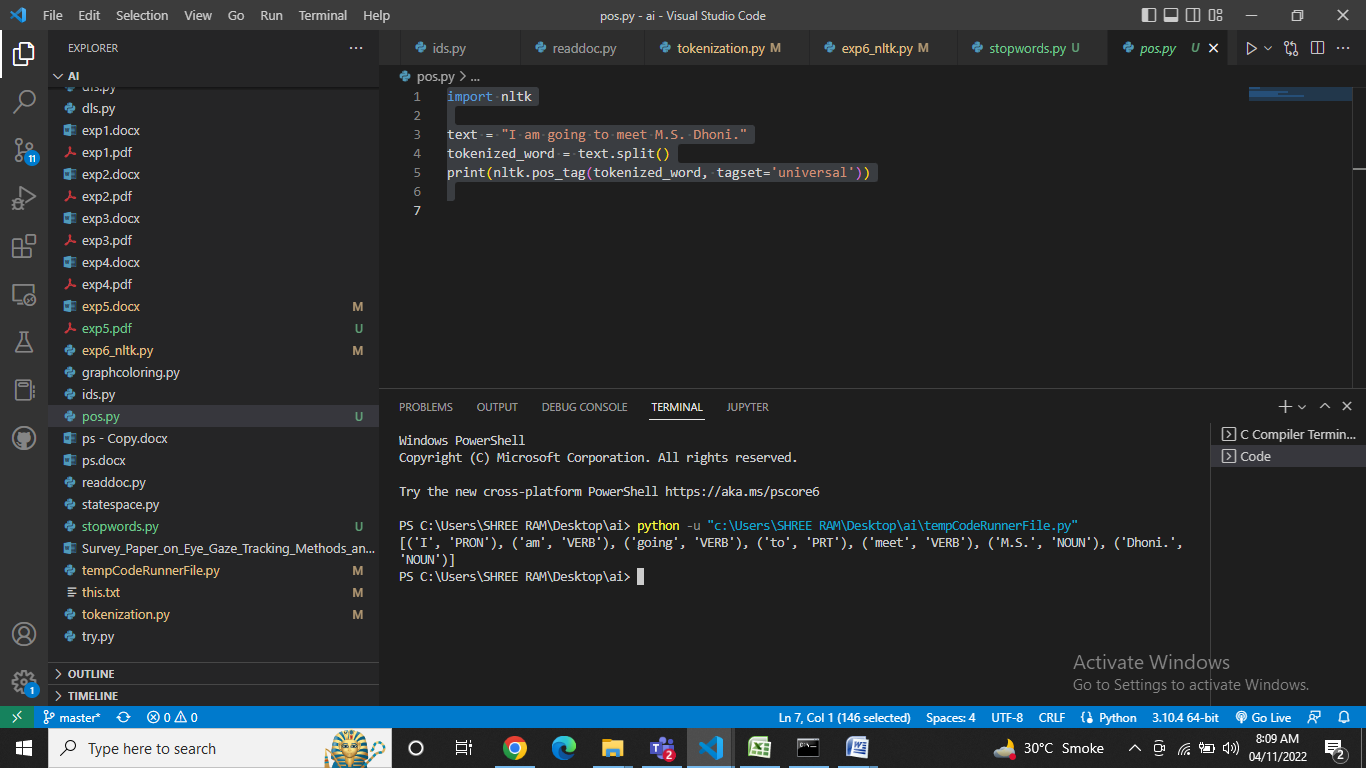
1. **Pos:**

import nltk

text = "I am going to meet M.S. Dhoni."

tokenized\_word = text.split()

print(nltk.pos\_tag(tokenized\_word, tagset='universal'))

**output:**

1. **Bag of words:**

n=int(input("Enter number of sentence to build vocabulary : "))

sentences = []

for i in range(0,n):

    l=input()

    sentences.append(l)

vocab = set()

for i in sentences:

    for j in i.split():

        vocab.add(j)

print("vocab: ",vocab)

print("sentence: ",sentences)

voclist = list(vocab)

bow=[]

for i in sentences:

    vi=[]

    for k in range(0,len(vocab)):

        vi.append(0)

    l=i.split()

    for j in range(0,len(vi)):

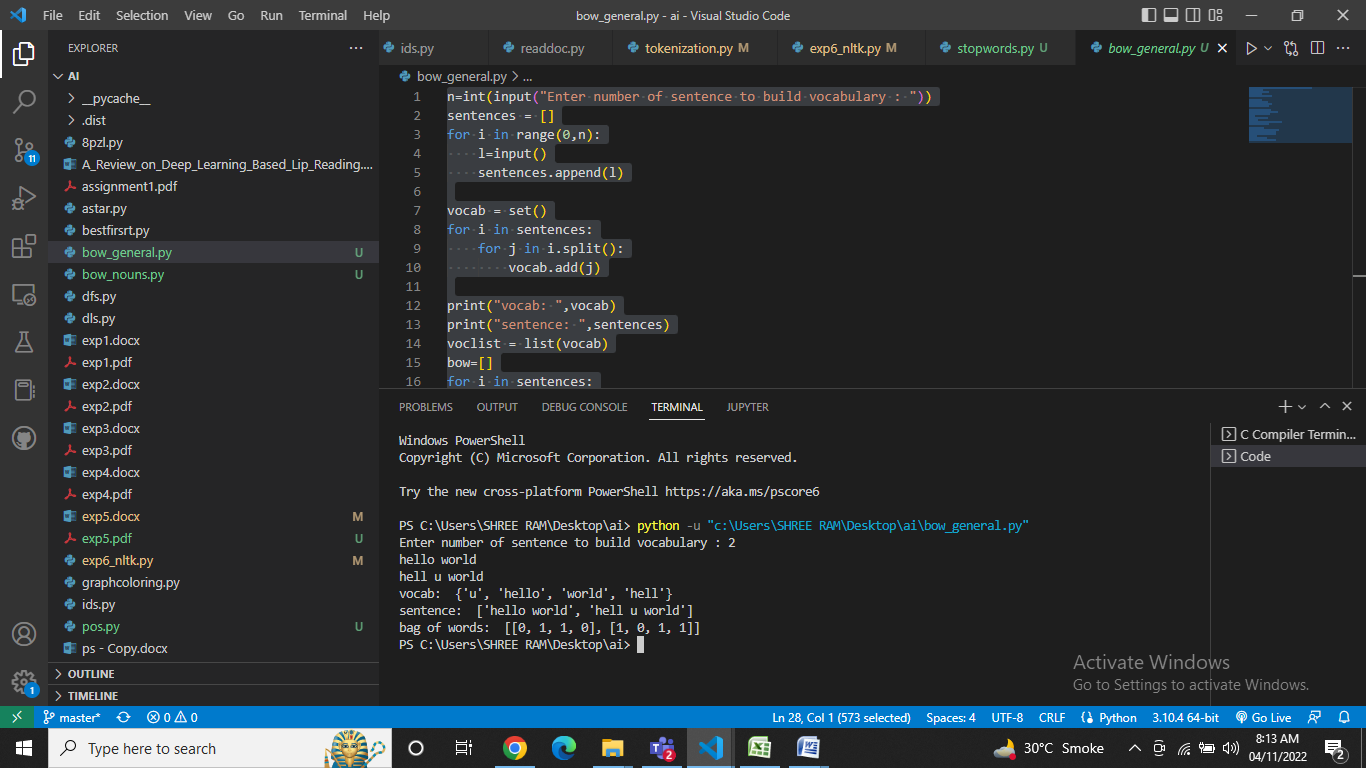
        if voclist[j] in l:

            vi[j] = vi[j]+1

    bow.append(vi)

print("bag of words: ",bow)

**output:**

****

1. **Bag of words for nouns only:**

import nltk

def get(sentence):

    t=[]

    tokenized\_word = sentence.split()

    need=nltk.pos\_tag(tokenized\_word, tagset='universal')

    for i in range(0,len(need)):

        if(need[i][1]=="NOUN"):

            t.append(need[i][0])

    return t

n=int(input("enter number of sentences to build vocabulary : "))

sentences=[]

for i in range(0,n):

    l=input()

    sentences.append(l)

vocab=set()

for i in sentences:

    t=get(i)

    for k in t:

        vocab.add(k)

print("vocab : ",vocab)

print("sentences : ",sentences)

voclist=list(vocab)

bow=[]

for i in sentences:

    vi=[]

    for m in range(0,len(vocab)):

        vi.append(0)

    l=i.split()

    for i in range(0,len(vi)):

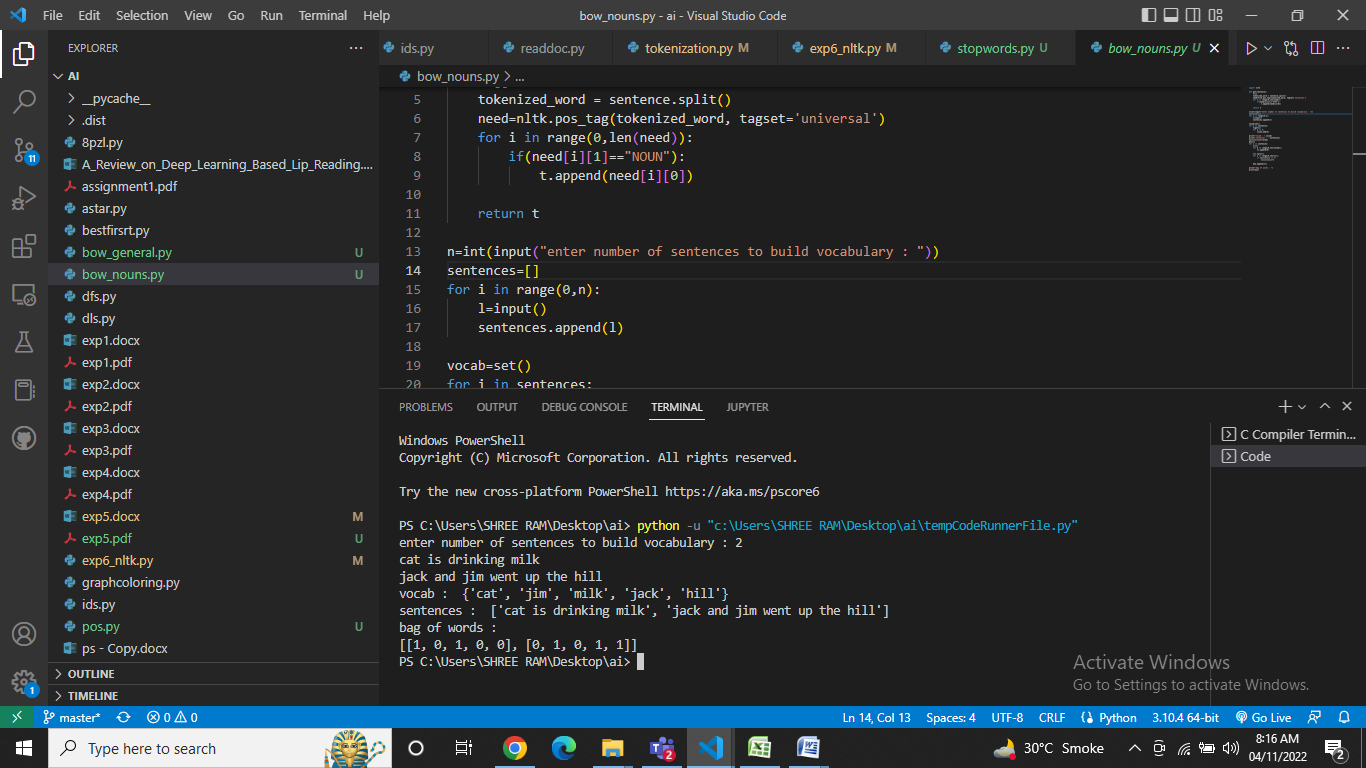
        if voclist[i] in l:

            vi[i]=vi[i]+1

    bow.append(vi)

print("bag of words : ")

print(bow)

**output:**

1. **TF-IDF:**

from sklearn.feature\_extraction.text import TfidfVectorizer

d0 = 'hello world'

d1 = 'hell you world'

d2 = 'hello from the otherside of world'

corpus = [d0, d1, d2]

tfidf = TfidfVectorizer()

result = tfidf.fit\_transform(corpus)

print('\nWord indexes:')

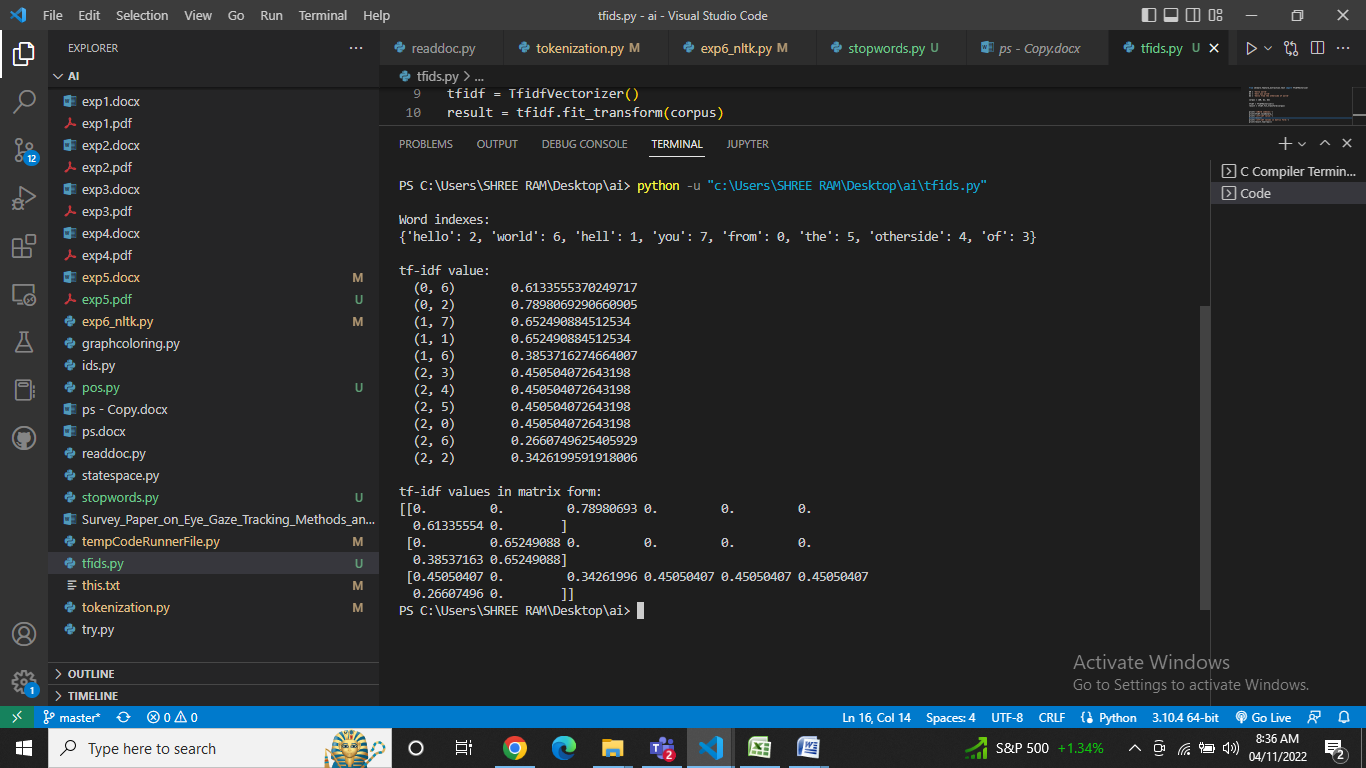
print(tfidf.vocabulary\_)

print('\ntf-idf value:')

print(result)

print('\ntf-idf values in matrix form:')

print(result.toarray())

**output:**

**CONCLUSION:**

bag-of-words major downside is that because it emphasizes words only on the basis of counts. To overcome this, a simple twist to bag-of-words introduces tf-idf approach.

Unlike, bag-of-words, tf-idf creates a normalized count where each word count is divided by the number of documents this word appears in.

Thus, Tf-Idf makes rare words more prominent and effectively ignores common words. It is closely related to frequency based filter, but much more mathematically elegant than placing hard cutoff thresholds.

While modelling an algorithm on tf-idf feature space, it is important to make use of other feature reduction technique as discussed earlier like stemming, part-of-speech etc, to get an effective results.

Hence, implemented Natural Language Processing using NLTK library, Bag of words and TF-IDF.

**REFERENCES:**

<https://www.analyticsvidhya.com/blog/2021/07/getting-started-with-nlp-using-nltk-library/>

<https://www.tutorialspoint.com/natural_language_toolkit/natural_language_toolkit_stemming_lemmatization.html>

<https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/>