COTB29 ANURAG ABHAY PARGAONKAR ASSIGNMENT NO.07

In [1]:

```
import pandas as pd
import nltk
import re
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
```

In [2]:

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Error loading punkt: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
[nltk_data] Error loading stopwords: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
[nltk_data] Error loading wordnet: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
[nltk_data] Error loading averaged_perceptron_tagger: <urlopen error
[nltk_data]     [Errno 11001] getaddrinfo failed>
```

Out[2]:

```
False
```

In [3]:

```
text="Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or
```

In [4]:

```
tokenized_text=sent_tokenize(text)
print(tokenized_text)
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Tokenization is the first step in text analytics.', 'The process of breaking down a text paragraph into smaller chunks su
ch as words or sentences is called Tokenization.']
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics', '.', 'The', 'process', 'of', 'breaking', 'down',
'a', 'text', 'paragraph', 'into', 'smaller', 'chunks', 'such', 'as', 'words', 'or', 'sentences', 'is', 'called', 'Tokenizat
ion', '.']
```

In [5]:

```
stop_words=set(stopwords.words("english"))
print(stop_words)
text="How to remove stop words with NLTK library in Python?"
text=re.sub('[^a-zA-Z]',' ',text)
tokens=word_tokenize(text.lower())
filtered_text=[]
for w in tokens:
    if w not in stop_words:
        filtered_text.append(w)
print("Tokenized Sentence:",tokens)
print("Filtered Sentence:",filtered_text)
```

```
{'couldn', 'these', 'and', 'didn', 'both', 'me', 'myself', 'you', 're', 'in', 'any', 'own', 'shan', 'then', "mightn't", 'we
re', 'we', 'because', 'against', "hadn't", "hasn't", "haven't", 'over', 'such', "that'll", 'between', "you've", "aren't",
'while', 'all', 'into', "weren't", 'theirs', "wasn't", "should've", 'isn', 'been', 'had', 'they', 'i', 'for', 'yours', 'its
elf', 'from', "you'll", 'wasn', 'again', 'be', 'them', 'haven', 'are', 'about', 'ours', 'of', "you'd", "you're", 'this', 'h
im', 'he', 'did', 'yourselves', 'a', 'only', 'herself', 'during', 'd', 'having', 'when', 'her', 'hasn', 'further', 'which',
'ma', 'by', 'the', 'his', 't', 'above', 'your', 'down', 'once', 'will', 'our', 'now', 'who', 'wouldn', 'can', "wouldn't",
'more', "needn't", 'has', 've', 'doesn', 'do', 'than', 'shouldn', "mustn't", 'is', 'or', 'whom', 'nor', 'being', "didn't",
'hadn', 'if', 'other', 'an', 'their', 'on', 'm', 'to', "couldn't", 'should', 'after', 'how', 'few', 'mustn', "shan't", 'bu
t', 'what', 'not', 'doing', 'won', 'it', 'themselves', 'very', 'at', 'under', 'yourself', 'most', 'where', 'just', 'until',
'weren', "doesn't", 'ain', 'here', "won't", 'as', "she's", 'off', 'll', 'through', 'there', 'with', 'does', 'some', "is
n't", "don't", 'below', 'himself', 'hers', 'each', 'so', "it's", 'mightn', 'no', 'its', "shouldn't", 'same', 's', 'don', 'a
ren', 'before', 'that', 'was', 'up', 'too', 'those', 'needn', 'ourselves', 'she', 'out', 'y', 'have', 'why', 'am', 'my',
'o'}
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filtered Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

In [6]:

```
e_words=["wait","waiting","waited","waits"]
ps=PorterStemmer()
for w in e_words:
    rootWord=ps.stem(w)
print(rootWord)
```

```
wait
```

In [7]:

```
import nltk
nltk.download('omw-1.4')
wordnet_lemmatizer=WordNetLemmatizer()
text="studies studying cries cry"
tokenization=nltk.word_tokenize(text)
for w in tokenization:
    print("Lemma for {} is {}".format(w,wordnet_lemmatizer.lemmatize(w)))
```

```
[nltk_data] Error loading omw-1.4: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>

Lemma for studies is study
Lemma for studying is studying
Lemma for cries is cry
Lemma for cry is cry
```

In [8]:

```
data="The pink sweater fit her perfectly"
words=word_tokenize(data)
for word in words:
    print(nltk.pos_tag([word]))
```

```
[('The', 'DT')]
[('pink', 'NN')]
[('sweater', 'NN')]
[('fit', 'NN')]
[('her', 'PRP$')]
[('perfectly', 'RB')]
```

In [9]:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import math
```

In [10]:

```
documentA='Jupiter is the largest Planet'
documentB='Mars is the fourth planet from the Sun'
bagOfWordsA=documentA.split(' ')
bagOfWordsB=documentB.split(' ')
uniqueWords=set(bagOfWordsA).union(set(bagOfWordsB))
numOfWordsA=dict.fromkeys(uniqueWords,0)
for word in bagOfWordsA:
    numOfWordsA[word]+=1
    numOfWordsB=dict.fromkeys(uniqueWords,0)
for word in bagOfWordsB:
    numOfWordsB[word]+=1
```

In [11]:

```
def computeTF(wordDict,bagOfWords):
    tfDict={}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

In [12]:

```
def computeIDF(documents):
    N = len(documents)
    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1
    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict
idfs = computeIDF([numOfWordsA, numOfWordsB])
idfs
```

Out[12]:

```
{'the': 0.0,
 'fourth': 0.6931471805599453,
 'Sun': 0.6931471805599453,
 'planet': 0.6931471805599453,
 'from': 0.6931471805599453,
 'Mars': 0.6931471805599453,
 'Planet': 0.6931471805599453,
 'largest': 0.6931471805599453,
 'is': 0.0,
 'Jupiter': 0.6931471805599453}
```

In [13]:

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
```

In [14]:

```
df
```

Out[14]:

	the	fourth	Sun	planet	from	Mars	Planet	largest	is	Jupiter
0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.138629	0.138629	0.0	0.138629
1	0.0	0.086643	0.086643	0.086643	0.086643	0.086643	0.000000	0.000000	0.0	0.000000