In [43]:
```
# Name:Atharv Santosh Danave
# Roll No.:11
# Practical No.:7
# Academic Year 2025-26
```

In [1]:
```
import ssl
import nltk
nltk.download('punkt_tab')

# Disable SSL certificate verification
ssl._create_default_https_context = ssl._create_unverified_context
# Now try downloading NLTK datasets again
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger_eng')
```

```
[nltk_data] Downloading package punkt_tab to /home/atharv/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /home/atharv/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/atharv/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /home/atharv/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
```

Out[1]: True

In [24]:
```
text= """Tokenization is the first step in text analytics. The process of breaking d
such as words or sentences is called Tokenization."""
```

In [25]:
```
#Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print(tokenized_text)
```

```
['Tokenization is the first step in text analytics.', 'The process of breaking down a
text paragraph into smaller chunks\nsuch as words or sentences is called Tokenizatio
n.']
```

In [26]:
```
#Word Tokenization
from nltk.tokenize import word_tokenize
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', 'ch
unks', 'such', 'as', 'words', 'or', 'sentences', 'is', 'called', 'Tokenization', '.']
```

In [27]:
```
# print stop words of English
import re
from nltk.corpus import stopwords
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("Filterd Sentence:",filtered_text)
```

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

In [28]:
```
#Perform stemming
from nltk.stem import PorterStemmer
e_words= ["wait", "waiting", "waited", "waits"]
ps =PorterStemmer()
for w in e_words:
  rootWord=ps.stem(w)
print(rootWord)
```

```
wait
```

In [29]:
```
#Perform lemmitization
from nltk.stem import WordNetLemmatizer
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)))
```

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

In [30]:
```
#Perform POS Tagging
import nltk
from nltk.tokenize import word_tokenize
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 [31]:
```
# Create representation of document by calculating TFIDF
```

In [32]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [33]:
```
documentA = 'Jupiter is the largest Planet'
```

```
documentB = 'Mars is the fourth planet from the Sun'
```

In [34]:
```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

In [35]:
```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

In [36]:
```
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 [37]:
```
def computeTF(wordDict, bagOfWords):
  tfDict = {}
  bagOfWordsCount = len(bagOfWords)
  for word, count in wordDict.items():
    tfDict[word] = count / float(bagOfWordsCount)
  return tfDict
```

In [38]:
```
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

In [39]:
```
def computeIDF(documents):
  import math
  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
```

In [40]:
```
def computeTFIDF(tfBagOfWords, idfs):
  tfidf = {}
  for word, val in tfBagOfWords.items():
    tfidf[word] = val * idfs[word]
  return tfidf
```

In [41]:
```
idfs = computeIDF([numOfWordsA, numOfWordsB])
idfs
```

Out[41]:
```
{'is': 0.0,
 'Mars': 0.6931471805599453,
 'from': 0.6931471805599453,
 'fourth': 0.6931471805599453,
 'largest': 0.6931471805599453,
 'Sun': 0.6931471805599453,
 'the': 0.0,
 'planet': 0.6931471805599453,
 'Planet': 0.6931471805599453,
 'Jupiter': 0.6931471805599453}
```

In [42]:
```
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
```

In [19]:
```
df = pd.DataFrame([tfidfA, tfidfB])
df
```

Out[19]:

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

In [ ]: