In [1]:
```
#Extract Sample document and apply following document preprocessing
#methods: Tokenization, POS Tagging, stop words removal, Stemming and
#Lemmatization
```

In [2]:
```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\KSV\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\KSV\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\KSV\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\wordnet.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\KSV\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

Out[2]: True

In [4]:
```
text= "Tokenization is the first step in text analytics."
```

In [5]:
```
#Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
tokenized_text
```

Out[5]: ['Tokenization is the first step in text analytics.']

In [6]:
```
#Word Tokenization
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
tokenized_word
```

Out[6]: ['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics', '.']

In [7]:
```
# print stop words of English
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
stop_words
```

Out[7]:
```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
 'be',
 'because',
 'been',
 'before',
 'being',
 'below',
```

```
'between',
'both',
'but',
'by',
'can',
'couldn',
"couldn't",
'd',
'did',
'didn',
"didn't",
'do',
'does',
'doesn',
"doesn't",
'doing',
'don',
"don't",
'down',
'during',
'each',
'few',
'for',
'from',
'further',
'had',
'hadn',
"hadn't",
'has',
'hasn',
"hasn't",
'have',
'haven',
"haven't",
'having',
'he',
'her',
'here',
'hers',
'herself',
'him',
'himself',
'his',
'how',
'i',
'if',
'in',
'into',
'is',
'isn',
"isn't",
'it',
"it's",
'its',
'itself',
'just',
'll',
'm',
'ma',
'me',
'mightn',
"mightn't",
'more',
'most',
'mustn',
"mustn't",
'my',
'myself',
'needn',
"needn't",
'no',
'nor',
'not',
'now',
'o',
'of',
'off',
'on',
'once',
'only',
'or',
'other',
'our',
```

```
'ours',
'ourselves',
'out',
'over',
'own',
're',
's',
'same',
'shan',
"shan't",
'she',
"she's",
'should',
"should've",
'shouldn',
"shouldn't",
'so',
'some',
'such',
't',
'than',
'that',
"that'll",
'the',
'their',
'theirs',
'them',
'themselves',
'then',
'there',
'these',
'they',
'this',
'those',
'through',
'to',
'too',
'under',
'until',
'up',
've',
'very',
'was',
'wasn',
"wasn't",
'we',
'were',
'weren',
"weren't",
'what',
'when',
'where',
'which',
'while',
'who',
'whom',
'why',
'will',
'with',
'won',
"won't",
'wouldn',
"wouldn't",
'y',
'you',
"you'd",
"you'll",
"you're",
"you've",
'your',
'yours',
'yourself',
'yourselves'}
```

In [8]:

```
import re
text= "How to remove stop words with NLTK library in Python?"
text= re.sub('[^a-zA-Z]', ' ',text)
tokens = word_tokenize(text.lower())
tokens
```

Out[8]:

```
['how',
```

```
'to',
'remove',
'stop',
'words',
'with',
'nltk',
'library',
'in',
'python']
```

In [9]:

```
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)
```

```
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

In [11]:

```
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
wait
wait
wait
```

In [12]:

```
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 [14]:

```
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 [15]:

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

In [16]:
```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

In [17]:
```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

In [18]:
```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

In [19]:
```
uniqueWords
```

Out[19]:
```
{'Jupiter',
 'Mars',
 'Planet',
 'Sun',
 'fourth',
 'from',
 'is',
 'largest',
 'planet',
 'the'}
```

In [20]:
```
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 [21]:
```
numOfWordsA
```

Out[21]:
```
{'is': 1,
 'the': 1,
 'planet': 0,
 'from': 0,
 'Sun': 0,
 'Planet': 1,
 'fourth': 0,
 'largest': 1,
 'Jupiter': 1,
 'Mars': 0}
```

In [22]:
```
numOfWordsB
```

Out[22]:
```
{'is': 1,
 'the': 2,
 'planet': 1,
 'from': 1,
 'Sun': 1,
 'Planet': 0,
 'fourth': 1,
 'largest': 0,
 'Jupiter': 0,
 'Mars': 1}
```

In [23]:
```
def computeTF(wordDict, bagOfWords):
    TfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        TfDict[word] = count / float(bagOfWordsCount)
    return TfDict
```

In [24]:
```
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

In [25]:
```
tfA
```

Out[25]:
```
{'is': 0.2,
 'the': 0.2,
 'planet': 0.0,
 'from': 0.0,
 'Sun': 0.0,
 'Planet': 0.2,
 'fourth': 0.0,
 'largest': 0.2,
 'Jupiter': 0.2,
 'Mars': 0.0}
```

In [26]:
```
tfB
```

Out[26]:
```
{'is': 0.125,
 'the': 0.25,
 'planet': 0.125,
 'from': 0.125,
 'Sun': 0.125,
 'Planet': 0.0,
 'fourth': 0.125,
 'largest': 0.0,
 'Jupiter': 0.0,
 'Mars': 0.125}
```

In [27]:
```
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
idfs = computeIDF([numOfWordsA, numOfWordsB])
idfs
```

Out[27]:
```
{'is': 0.0,
 'the': 0.0,
 'planet': 0.6931471805599453,
 'from': 0.6931471805599453,
 'Sun': 0.6931471805599453,
 'Planet': 0.6931471805599453,
 'fourth': 0.6931471805599453,
 'largest': 0.6931471805599453,
 'Jupiter': 0.6931471805599453,
 'Mars': 0.6931471805599453}
```

In [28]:
```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

In [29]:
```
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
df
```

Out[29]:

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

In [ ]:
```

```