In [1]:
```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /home/student/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/student/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/student/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/student/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[1]:
```
True
```

In [2]:
```
text = "Tokenization is the first step in text analytics. Theprocess of breaking down a text par
```

In [3]:
```
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print(tokenized_text)
```

```
['Tokenization is the first step in text analytics.', 'Theprocess of breaking down a text paragr
aph into smaller chunkssuch as words or sentences is called Tokenization.']
```

In [4]:
```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics', '.', 'Theprocess', 'o
f', 'breaking', 'down', 'a', 'text', 'paragraph', 'into', 'smaller', 'chunkssuch', 'as', 'word
s', 'or', 'sentences', 'is', 'called', 'Tokenization', '.']
```

In [5]:
```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

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

In [6]:
```
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())
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', 'p
ython']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

In [7]:
```
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 [8]:
```
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 [9]:
```
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 [10]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [11]:
```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

In [12]:
```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

In [13]:
```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

In [14]:
```
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 [16]:
```
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 [17]:
```
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[17]:
```
{'the': 0.0,
 'Mars': 0.6931471805599453,
 'planet': 0.6931471805599453,
 'Jupiter': 0.6931471805599453,
 'largest': 0.6931471805599453,
 'is': 0.0,
 'fourth': 0.6931471805599453,
 'from': 0.6931471805599453,
 'Sun': 0.6931471805599453,
 'Planet': 0.6931471805599453}
```

In [18]:
```
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])
df
```

Out[18]:

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

In [ ]: