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 [32]:
```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

Out[32]:
```
True
```

In [6]:
```
text="Tokenization is the first step in text analytics."
```

In [10]:
```
tokenized_text=sent_tokenize(text)
print(tokenized_text)
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
['Tokenization is the first step in text analytics.']
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics', '.']
```

In [13]:
```
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)
```

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

In [15]:
```
e_words=["wait","waiting","waited","waits"]
ps=PorterStemmer()
for w in e_words:
    rootWord=ps.stem(w)
print(rootWord)
```

```
wait
```

In [27]:
```
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 [19]:
```
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 [20]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
import math
```

In [23]:
```
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 [25]:
```
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 [28]:
```
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[28]:
```
{'planet': 0.6931471805599453,
 'largest': 0.6931471805599453,
 'is': 0.0,
 'Mars': 0.6931471805599453,
 'Planet': 0.6931471805599453,
 'Jupiter': 0.6931471805599453,
 'Sun': 0.6931471805599453,
 'the': 0.0,
 'from': 0.6931471805599453,
 'fourth': 0.6931471805599453}
```

In [29]:
```
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 [30]:
```
df
```

Out[30]:

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

In [ ]:
```

```