In [2]:
```
import nltk
```

In [3]:
```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\System21\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\System21\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\System21\AppData\Roaming\nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\System21\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

Out[3]:
```
True
```

In [4]:
```
text= "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
```

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

In [12]:
```
# 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)
```

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

In [10]:
```
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 [14]:
```
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 [15]:
```
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 [16]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [17]:
```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

In [18]:
```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

In [19]:
```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

In [21]:
```
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 [36]:
```
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 [34]:
```
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), 10)
    return idfDict
```

In [35]:
```
idfs = computeIDF([numOfWordsA, numOfWordsB])
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[35]:

	Mars
0	0.000000
1	0.037629

In [ ]:
```
```