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\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\admin\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[2]:
```
True
```

In [4]:
```
text="Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks
```

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', 'dow
n', 'a', 'text', 'paragraph', 'into', 'smaller', 'chunks', 'such', 'as', 'words', 'or', 'sentences', 'is', 'called', 'To
kenization', '.']
```

In [25]:
```
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

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 = [w for w in tokens if w not in stop_words]

print("Tokenized Sentence:", tokens)
print("Filtered Sentence:", filtered_text)
```

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

In [26]:
```
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 [27]:
```
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 [28]:
```
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 [29]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [30]:
```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

In [31]:
```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

In [32]:
```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

In [33]:
```
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 [34]:
```
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 [35]:
```
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_same = idfs.copy()
```

In [36]:
```
import pandas as pd
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])
print(df)
```

```
       Mars    Planet  the   largest    fourth      from    planet       Sun  \
0  0.000000  0.138629  0.0  0.138629  0.000000  0.000000  0.000000  0.000000
1  0.086643  0.000000  0.0  0.000000  0.086643  0.086643  0.086643  0.086643

    Jupiter   is
0  0.138629  0.0
1  0.000000  0.0
```

In [ ]: