```
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\91997\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\91997\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\91997\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\91997\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
True
```

```
text= "Tokenization is the first step in text analytics. The process
of breaking down a text paragraph into smaller chunkssuch as words or
sentences is called 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 chunkssuch 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', 'chunkssuch', 'as', 'words',
'or', 'sentences', 'is', 'called', 'Tokenization', '.']
```

```
from nltk.corpus import stopwords
import re
```

```
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("Filterd Sentence:",filtered_text)
```

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

```
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
```

```
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 {}: {}".format(w,
wordnet_lemmatizer.lemmatize(w)))
```

```
Lemma for studies: study
Lemma for studying: studying
Lemma for cries: cry
Lemma for cry: cry
```

```
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')]
```

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

documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'

bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')

bagOfWordsA
```

```
['Jupiter', 'is', 'the', 'largest', 'Planet']
```

```
bagOfWordsB
```

```
['Mars', 'is', 'the', 'fourth', 'planet', 'from', 'the', 'Sun']
```

```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
uniqueWords
```

```
{'Jupiter',
 'Mars',
 'Planet',
 'Sun',
 'fourth',
```

```
 'from',
 'is',
 'largest',
 'planet',
 'the'}

numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1

numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1

numOfWordsA
```

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

```
numOfWordsB
```

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

```
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)

tfA
```

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

tfB

{'fourth': 0.125,
 'largest': 0.0,
 'from': 0.125,
 'is': 0.125,
 'Sun': 0.125,
 'the': 0.25,
 'Mars': 0.125,
 'planet': 0.125,
 'Planet': 0.0,
 'Jupiter': 0.0}

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

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

```
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)

# Create a DataFrame for visualization
df = pd.DataFrame([tfidfA, tfidfB])
print(df)
```

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

     Planet   Jupiter
0  0.138629  0.138629
1  0.000000  0.000000
```