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


nltk.download('punkt')  # Download the Punkt tokenizer
nltk.download('stopwords')  # Download the stopwords corpus
nltk.download('wordnet')  # Download the WordNet corpus
nltk.download('averaged_perceptron_tagger')  # Download the averaged perceptron tagger
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
from nltk.tokenize import sent_tokenize, word_tokenize


text= "Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or s


#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)
```

```
 paragraph into smaller chunks such as words or sentences is called Tokenization.']
cess', 'of', 'breaking', 'down', 'a', 'text', 'paragraph', 'into', 'smaller', 'chunks', 'such', 'as', 'words', 'or', 'sentences', 'is',
```

```
# Print stop words of English
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)
```

```
{'itself', 'shan', "you've", 'will', 'these', 'hasn', 'there', 'with', 'only', "it's", 'whom', 'isn', 'own', "doesn't", 'a', 'not', "had
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filtered 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
```

```
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
```

```
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')]
```

```
#representation of document by calculating TFIDF


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


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


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)
```

```
import math

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])
print(idfs)
```

```
{'Sun': 0.6931471805599453, 'from': 0.6931471805599453, 'largest': 0.6931471805599453, 'planet': 0.6931471805599453, 'Mars': 0.693147180
```

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf

# Call the function with appropriate arguments
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
df
```

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

Next steps: Generate code with df | View recommended plots

Start coding or generate with AI.