```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/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 chunks such as words or s
```

```
#Perform Tokenization , 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 c
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics', '.', 'The', 'process', 'of', 'breaking', 'down', 'a', 'text',
```

```
#Removing Punctuations and Stop Word
# print stop words of English

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=[]
for w in tokens:
  if w not in stop_words:
    filtered_text.append(w)
print("Tokenized Sentence:",tokens)
print("Filterd Sentence:",filtered_text)
```

```
{"won't", "you'd", 'same', "wasn't", 'couldn', 'been', 'they', 'had', 'doesn', 'shouldn', 'and', "you're", 'with', 'having', 'here', 'ou
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

```
#Perform Stemming
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
```

```
#Perform Lemmatization
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
```

```
#Apply POS Tagging to text
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')]
```

```
# Algorithm for Create 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'
```

```
#Create BagofWords (BoW) for Document A and B.
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

Start coding or generate with AI.

```
bagOfWordsA
```

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

```
bagOfWordsB
```

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

```
# Create Collection of Unique words from Document A and B.
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

```
uniqueWords
```

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

```
# Create a dictionary of words and their occurrence for each document in the corpus
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
```

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

```
numOfWordsB
```

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

```
# Compute the term frequency for each of our documents.


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
```

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

```
tfB
```

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

```
# Compute the term Inverse Document Frequency.
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
```

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

```
# Compute the term TF/IDF for all words.
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
```

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

```
tfidfA
```

```
{'largest': 0.13862943611198905,
 'Jupiter': 0.13862943611198905,
 'planet': 0.0,
 'is': 0.0,
 'Planet': 0.13862943611198905,
 'Mars': 0.0,
 'the': 0.0,
 'Sun': 0.0,
 'from': 0.0,
 'fourth': 0.0}
```

```
tfidfB
```

```
{'largest': 0.0,
 'Jupiter': 0.0,
 'planet': 0.08664339756999316,
 'is': 0.0,
 'Planet': 0.0,
 'Mars': 0.08664339756999316,
 'the': 0.0,
 'Sun': 0.08664339756999316,
 'from': 0.08664339756999316,
 'fourth': 0.08664339756999316}
```

```
df
```

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