```
from nltk.tokenize import sent_tokenize, word_tokenize

text = "Natural language processing is an exciting area."

print(sent_tokenize(text))
```

```
['Natural language processing is an exciting area.']
```

```
import nltk
from nltk.corpus import stopwords
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
```

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

example_sent = """This is a sample sentence,
                  showing off the stop words filtration."""

stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example_sent)

filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]

filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print(word_tokens)
print(filtered_sentence)
```

```
['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words',
['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

```
# import these modules
from nltk.stem import WordNetLemmatizer
```

```
lemmatizer = WordNetLemmatizer()

print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))

# a denotes adjective in "pos"
print("better :", lemmatizer.lemmatize("better", pos ="a"))
```

```
     rocks : rock
     corpora : corpus
     better : good
```

```
# import these modules
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))

# a denotes adjective in "pos"
print("better :", lemmatizer.lemmatize("better", pos ="a"))
```

```
     rocks : rock
     corpora : corpus
     better : good
```

```
import nltk
from nltk.corpus import stopwords
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk.tokenize import word_tokenize, sent_tokenize
stop_words = set(stopwords.words('english'))
txt = "Natural language processing is an exciting area. Huge budget have been allocated for t
# sent_tokenize is one of instances of
# PunktSentenceTokenizer from the nltk.tokenize.punkt module
tokenized = sent_tokenize(txt)
for i in tokenized:
  # Word tokenizers is used to find the words
  # and punctuation in a string
  wordsList = nltk.word_tokenize(i)
  # removing stop words from wordList
  wordsList = [w for w in wordsList if not w in stop_words]
  # Using a Tagger. Which is part-of-speech
  # tagger or POS-tagger.
  tagged = nltk.pos_tag(wordsList)
  print(tagged)
```

```
     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Package punkt 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!
[('Natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ('exciting', 'JJ'), ('area
[('Huge', 'NNP'), ('budget', 'NN'), ('allocated', 'VBD'), ('.', '.')]
```

```
import pandas as pd
import sklearn as sk
import math


first_sentence = "Data Science is the sexiest job of the 21st century"
second_sentence = "machine learning is the key for data science"
#split so each word have their own string
first_sentence = first_sentence.split(" ")
second_sentence = second_sentence.split(" ")#join them to remove common duplicate words
total= set(first_sentence).union(set(second_sentence))
print(total)
```

```
{'the', 'job', 'for', 'century', 'learning', 'key', 'is', '21st', 'science', 'of', 'mach
```

```
wordDictA = dict.fromkeys(total, 0)
wordDictB = dict.fromkeys(total, 0)
for word in first_sentence:
    wordDictA[word]+=1

for word in second_sentence:
    wordDictB[word]+=1


pd.DataFrame([wordDictA, wordDictB])
```

	the	job	for	century	learning	key	is	21st	science	of	machine	data	Sc
0	2	1	0	1	0	0	1	1	0	1	0	0	
1	1	0	1	0	1	1	1	0	1	0	1	1	

```
#No let’s writing the TF Function :

def computeTF(wordDict, doc):
    tfDict = {}
    corpusCount = len(doc)
    for word, count in wordDict.items():
        tfDict[word] = count/float(corpusCount)
    return(tfDict)
#running our sentences through the tf function:
tfFirst = computeTF(wordDictA, first_sentence)
```

```
tfSecond = computeTF(wordDictB, second_sentence)
#Converting to dataframe for visualization
tf = pd.DataFrame([tfFirst, tfSecond])
print(tf)
```

```
     the  job    for  century  learning    key     is  21st  science   of  \
0  0.200  0.1  0.000      0.1     0.000  0.000  0.100   0.1    0.000  0.1
1  0.125  0.0  0.125      0.0     0.125  0.125  0.125   0.0    0.125  0.0

   machine   data  Science  Data  sexiest
0    0.000  0.000      0.1   0.1      0.1
1    0.125  0.125      0.0   0.0      0.0
```

```
#And now that we finished the TF section, we move onto the IDF part:

def computeIDF(docList):
    idfDict = {}
    N = len(docList)

    idfDict = dict.fromkeys(docList[0].keys(), 0)
    for word, val in idfDict.items():
        idfDict[word] = math.log10(N / (float(val) + 1))

    return(idfDict)
#inputing our sentences in the log file
idfs = computeIDF([wordDictA, wordDictB])


def computeTFIDF(tfBow, idfs):
    tfidf = {}
    for word, val in tfBow.items():
        tfidf[word] = val*idfs[word]
    return(tfidf)
#running our two sentences through the IDF:
idfFirst = computeTFIDF(tfFirst, idfs)
idfSecond = computeTFIDF(tfSecond, idfs)
#putting it in a dataframe
idf= pd.DataFrame([idfFirst, idfSecond])
print(idf)
```

```
         the       job       for   century  learning       key        is  \
0  0.060206  0.030103  0.000000  0.030103  0.000000  0.000000  0.030103
1  0.037629  0.000000  0.037629  0.000000  0.037629  0.037629  0.037629

        21st   science        of   machine      data   Science      Data  \
0  0.030103  0.000000  0.030103  0.000000  0.000000  0.030103  0.030103
1  0.000000  0.037629  0.000000  0.037629  0.037629  0.000000  0.000000

     sexiest
0  0.030103
1  0.000000
```

✓ 0s completed at 8:16 PM