```
import nltk
#nltk.download()

nltk.download('punkt')
nltk.download('stopwords')
from nltk.stem.lancaster import LancasterStemmer

stemmer = LancasterStemmer()
print(stemmer.stem('eating'))
print(stemmer.stem('eats'))
print(stemmer.stem('eaten'))
print(stemmer.stem('ate'))
```

```
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()
print(stemmer.stem('eating'))
print(stemmer.stem('eats'))
print(stemmer.stem('eaten'))
print(stemmer.stem('ate'))
```

```
from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer('english')
print(stemmer.stem('eating'))
print(stemmer.stem('eats'))
print(stemmer.stem('eaten'))
print(stemmer.stem('ate'))
```

```
#import nltk
#nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("better", pos="a"))
print(lemmatizer.lemmatize("driving", pos="v"))
print(lemmatizer.lemmatize("ate", pos="v"))
```

```
import nltk
nltk.download('averaged_perceptron_tagger')
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.tag import pos_tag

my_text = "James Smith lives in the United States."
tokens = pos_tag(word_tokenize(my_text))

print(tokens)
```

```
import nltk
nltk.download('tagsets')
nltk.help.upenn_tagset()
```

```
import nltk
nltk.download('maxent_ne_chunker')
nltk.download('words')
from nltk.chunk import ne_chunk

my_text = "James Smith lives in the United States."
tokens = pos_tag(word_tokenize(my_text))

entities = ne_chunk(tokens)
print(entities)
```

```
from nltk.tokenize import MWETokenizer

mwe_tokenizer = MWETokenizer([('James', 'Smith'), ('United', 'States')], separator=' ')
print(mwe_tokenizer.tokenize(word_tokenize(my_text)))
```

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

```
sentence_1="This movie is excellent"
sentence_2="This movie is quite impressive and lengthy"
sentence_3="This movie is very bad and boring"

CountVec = CountVectorizer(ngram_range=(1,1), # to use bigrams ngram_range=(2,2)
                   stop_words='english')
#transform
Count_data = CountVec.fit_transform([sentence_1,sentence_2, sentence_3])

#create dataframe
cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_names_out()
)
print(cv_dataframe)
```

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()

X=["This movie is excellent","This movie is quite impressive and lengthy","This movie is very
bad and boring"]

X = vectorizer.fit_transform(X)

feature_names = vectorizer.get_feature_names_out()

X_array = X.toarray()

print("Unique Word List: \n", feature_names)
print("Bag of Words Matrix: \n", X_array)
```

```
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)
```

Now, lets add a way to count the words using a dictionary key-value pairing for both sentences:

```
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])
```

Now define TF-IDF functions:

```
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)
```

IDF

```
import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

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

filtered_sentence = [w for w in wordDictA if not w in stop_words]

print(filtered_sentence)



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])
```

Calculate TF-IDF

```
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)
```