In [1]:

```
import nltk
```

In [18]:

```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\hp\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\hp\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\hp\AppData\Roaming\nltk_data...
```

Out[18]:

```
True
```

In [19]:

```
text = "Tokenization is the first step in text analytics. The process of breaking down a
```

In [20]:

```
from nltk.tokenize import sent_tokenize
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

```
['Tokenization is the first step in text analytics.', 'The process of brea
king down a text paragraph into smaller chunks such as words or sentences
is called Tokenization.']
```

In [21]:

```
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', 'or', 'sentences', 'i
s', 'called', 'Tokenization', '.']
```

In [22]:

```
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
```

In [23]:

```
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
```

In [30]:

```
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')]
```

Part B

In [31]:

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

In [32]:

```
first_sentence = "Jupiter is the largest Planet"
second_sentence = "Mars is the fourth planet from the Sun"
#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)
```

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

In [34]:

```
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
```

In [35]:

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

Out[35]:

	Mars	Planet	largest	from	Sun	the	planet	is	Jupiter	fourth
0	0	1	1	0	0	1	0	1	1	0
1	1	0	0	1	1	2	1	1	0	1

In [38]:

```
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])
tf
```

Out[38]:

	Mars	Planet	largest	from	Sun	the	planet	is	Jupiter	fourth
0	0.000	0.2	0.2	0.000	0.000	0.20	0.000	0.200	0.2	0.000
1	0.125	0.0	0.0	0.125	0.125	0.25	0.125	0.125	0.0	0.125

In [41]:

```
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])
idfs
```

Out[41]:

```
{'Mars': 0.3010299956639812,
 'Planet': 0.3010299956639812,
 'largest': 0.3010299956639812,
 'from': 0.3010299956639812,
 'Sun': 0.3010299956639812,
 'the': 0.3010299956639812,
 'planet': 0.3010299956639812,
 'is': 0.3010299956639812,
 'Jupiter': 0.3010299956639812,
 'fourth': 0.3010299956639812}
```

In [42]:

```
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)
```

```
       Mars    Planet   largest      from       Sun       the    planet  \
0  0.000000  0.060206  0.060206  0.000000  0.000000  0.060206  0.000000
1  0.037629  0.000000  0.000000  0.037629  0.037629  0.075257  0.037629

         is   Jupiter    fourth
0  0.060206  0.060206  0.000000
1  0.037629  0.000000  0.037629
```

In [ ]: