```
In [1]: # Assignment - A7 | Name :                         | Roll No :
```

```
In [15]: # Importing the libraries
import nltk
import pandas as pd
import sklearn as sk
import math
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\StepInfotech\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\StepInfotech\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\StepInfotech\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\StepInfotech\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\StepInfotech\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
Out[15]: True
```

Sample Sentences

```
In [16]: sentence1 = "I will walk 500 miles and I would walk 500 more. Just to be the man who w
"a thousand miles to fall down at your door!"
sentence2 = "I played the play playfully as the players were playing in the play with
```

Tokenization

```
In [17]: from nltk import word_tokenize, sent_tokenize
print('Tokenized words:', word_tokenize(sentence1))
print('\nTokenized sentences:', sent_tokenize(sentence1))
```

```
Tokenized words: ['I', 'will', 'walk', '500', 'miles', 'and', 'I', 'would', 'walk',
'500', 'more', '.', 'Just', 'to', 'be', 'the', 'man', 'who', 'walks', 'a', 'thousan
d', 'miles', 'to', 'fall', 'down', 'at', 'your', 'door', '!']

Tokenized sentences: ['I will walk 500 miles and I would walk 500 more.', 'Just to be
the man who walks a thousand miles to fall down at your door!']
```

POS Tagging

In [18]:

```
from nltk import pos_tag
token = word_tokenize(sentence1) + word_tokenize(sentence2)
tagged = pos_tag(token)
print("Tagging Parts of Speech:", tagged)
```

```
Tagging Parts of Speech: [('I', 'PRP'), ('will', 'MD'), ('walk', 'VB'), ('500', 'C
D'), ('miles', 'NNS'), ('and', 'CC'), ('I', 'PRP'), ('would', 'MD'), ('walk', 'VB'),
('500', 'CD'), ('more', 'JJR'), ('.', '.'), ('Just', 'NNP'), ('to', 'TO'), ('be', 'V
B'), ('the', 'DT'), ('man', 'NN'), ('who', 'WP'), ('walks', 'VBZ'), ('a', 'DT'), ('th
ousand', 'NN'), ('miles', 'NNS'), ('to', 'TO'), ('fall', 'VB'), ('down', 'RP'), ('a
t', 'IN'), ('your', 'PRP$'), ('door', 'NN'), ('!', '.'), ('I', 'PRP'), ('played', 'VB
D'), ('the', 'DT'), ('play', 'NN'), ('playfully', 'RB'), ('as', 'IN'), ('the', 'DT'),
('players', 'NNS'), ('were', 'VBD'), ('playing', 'VBG'), ('in', 'IN'), ('the', 'DT'),
('play', 'NN'), ('with', 'IN'), ('playfullness', 'NN')]
```

Stop-Words Removal

In [19]:

```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
token = word_tokenize(sentence1)
cleaned_token = []
for word in token:
    if word not in stop_words:
        cleaned_token.append(word)
print('Unclean version:', token)
print('\nCleaned version:', cleaned_token)
```

```
Unclean version: ['I', 'will', 'walk', '500', 'miles', 'and', 'I', 'would', 'walk',
'500', 'more', '.', 'Just', 'to', 'be', 'the', 'man', 'who', 'walks', 'a', 'thousan
d', 'miles', 'to', 'fall', 'down', 'at', 'your', 'door', '!']

Cleaned version: ['I', 'walk', '500', 'miles', 'I', 'would', 'walk', '500', '.', 'Jus
t', 'man', 'walks', 'thousand', 'miles', 'fall', 'door', '!']
```

Stemming

In [20]:

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
token = word_tokenize(sentence2)
stemmed = [stemmer.stem(word) for word in token]
print(" ".join(stemmed))
```

```
i play the play play as the player were play in the play with playful
```

Lemmatization

In [21]:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
token = word_tokenize(sentence2)
lemmatized_output = [lemmatizer.lemmatize(word) for word in token]
print(" ".join(lemmatized_output))
```

```
I played the play playfully a the player were playing in the play with playfullness
```

Term Frequency - Inverse Document Frequency

In [22]:

```python
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)
```

```
{'key', 'Data', 'job', 'Science', 'the', 'science', 'century', 'for', 'is', '21st',
'machine', 'of', 'data', 'sexiest', 'learning'}
```

In [23]:

```python
# 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
```

In [24]:

```python
# Now we put them in a dataframe and then view the result
pd.DataFrame([wordDictA, wordDictB])
```

Out[24]:

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

In [25]:

```python
# 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)
```

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

   machine   of   data  sexiest  learning  
0    0.000  0.1  0.000      0.1     0.000  
1    0.125  0.0  0.125      0.0     0.125  
```

In [26]:

```
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])
```

In [27]:

```
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)
```

```
        key      Data       job   Science       the   science   century  \
0  0.000000  0.030103  0.030103  0.030103  0.060206  0.000000  0.030103   
1  0.037629  0.000000  0.000000  0.000000  0.037629  0.037629  0.000000   

        for        is      21st   machine        of      data   sexiest  \
0  0.000000  0.030103  0.030103  0.000000  0.030103  0.000000  0.030103   
1  0.037629  0.037629  0.000000  0.037629  0.000000  0.037629  0.000000   

   learning  
0  0.000000  
1  0.037629  
```

In [ ]:

In [ ]: