Assignment 7

1. Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.
2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.

1. Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.

Download the required packages

In [1]:

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

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

Out[1]:

```
True
```

Initialize the text

In [2]:

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

Perform Tokenization

In [3]:

```
#Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
tokenized_text
```

Out[3]:

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

In [4]:

```
#Word Tokenization
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
tokenized_word
```

Out[4]:

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

Removing Punctuations and Stop Word

In [5]:

```
# print stop words of English
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
stop_words
```

Out[5]:

```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
 'be',
```

In [6]:

```
import re
text= "How to remove stop words with NLTK library in Python?"
text= re.sub('[^a-zA-Z]', ' ',text)
tokens = word_tokenize(text.lower())
tokens
```

Out[6]:

```
['how',
 'to',
 'remove',
 'stop',
 'words',
 'with',
 'nltk',
 'library',
 'in',
 'python']
```

In [7]:

```
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)
```

```
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk',
'library', 'in', 'python']
Filterd Sentence: ['remove']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk',
'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk',
'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk',
'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk',
'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library']
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk',
'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

Perform Stemming

In [8]:

```
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
```

Perform Lemmatization

In [9]:

```
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

In [10]:

```
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')]
```

2.Create representation of document by calculating Term Frequency and Inverse Document Frequency.

Import the necessary libraries.

In [11]:

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

Initialize the Documents.

In [12]:

```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

Create BagofWords (BoW) for Document A and B.

In [13]:

```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

Create Collection of Unique words from Document A and B.

In [14]:

```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

In [15]:

```
uniqueWords
```

Out[15]:

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

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

In [16]:

```
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1
numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1
```

In [17]:

```
numOfWordsA
```

Out[17]:

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

In [18]:

```
numOfWordsB
```

Out[18]:

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

Compute the term frequency for each of our documents.

In [19]:

```
def computeTF(wordDict, bagOfWords):
    TfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        TfDict[word] = count / float(bagOfWordsCount)
    return TfDict
```

In [20]:

```
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

In [21]:

```
tfA
```

Out[21]:

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

In [22]:

```
tfB
```

Out[22]:

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

Compute the term Inverse Document Frequency.

In [23]:

```
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
```

Out[23]:

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

Compute the term TF/IDF for all words.

In [24]:

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

In [25]:

```
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
df
```

Out[25]:

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