Practical No:7

In [16]:

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

```
[nltk_data] Downloading package punkt to /home/student/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/student/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /home/student/nltk_dat
a...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/student/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[16]:

```
True
```

In [15]:

```
text= "Tokenization is the first step in text analytics.The process o
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print(tokenized_text)

print ('-'*80)

from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)

print ('-'*80)

from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)

print ('-'*80)

word_tokens= word_tokenize(text.lower())
filtered_sentence = []
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
print("Tokenized Sentence:",word_tokens)
print("Filterd Sentence:",filtered_sentence)

print ('-'*80)

from nltk.stem import PorterStemmer
e_words= ["wait", "waiting", "waited", "waits"]
ps =PorterStemmer()
for w in e_words:
    rootWord=ps.stem(w)
    print(rootWord)

print ('-'*80)

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

print ('-'*80)

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]))
```

```
['Tokenization is the first step in text analytics.The process of b
reaking down a text paragraph into smaller chunks such as words or
sentences is called Tokenization.']
-------------------------------------------------------------------
-------------
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analy
tics.The', 'process', 'of', 'breaking', 'down', 'a', 'text', 'parag
raph', 'into', 'smaller', 'chunks', 'such', 'as', 'words', 'or', 's
entences', 'is', 'called', 'Tokenization', '.']
--------------------------------------------------------------------
-------------
{'then', 'yourselves', 'him', 'that', 'until', 'as', 'here', 'not',
'where', 'my', 'both', 'about', 'so', 'each', 'aren', 'am', 'she',
'does', 'have', 'should', 'your', "you've", 'during', 'out', 'do',
'just', 'through', "isn't", 'these', 'won', 'its', 'myself', 'under
', "needn't", 'weren', 'a', 're', 'same', "hadn't", "you'll", 'how
', 't', 'on', 'some', 'can', 'ma', 'them', 'shouldn', 'further', 't
hemselves', "should've", 'such', "it's", 'which', 'now', "shouldn'
t", 'between', 'too', 'other', 'll', 'than', "didn't", 'there', 'no
', "you'd", 'by', 'those', 'above', 'all', "hasn't", "won't", 'your
self', "doesn't", 'doesn', "you're", 'don', "she's", 'yours', 'own
', 'an', 'most', 'at', 'with', 'are', 've', 'was', 'this', "weren'
t", 'needn', 'ourselves', 'ain', 'if', 'only', "couldn't", 'they',
'his', 'again', 'before', 'into', 'having', 's', 'had', 'her', 'it
', 'what', 'below', 'isn', 'wasn', 'we', 'who', 'the', 'mustn', 'di
d', 'itself', 'to', 'haven', 'while', 'been', 'o', 'and', 'nor', 't
heir', "mustn't", 'more', "wouldn't", 'shan', "mightn't", 'couldn',
'once', 'y', 'hasn', 'has', 'he', 'didn', "wasn't", 'be', 'against
', 'is', 'because', 'doing', 'ours', 'but', 'hers', "don't", 'will
', 'hadn', 'you', 'for', 'of', 'when', 'any', 'why', 'himself', 'me
', "aren't", "haven't", 'herself', 'from', 'over', 'our', 'off', 'm
', 'wouldn', "that'll", 'in', 'being', 'after', 'were', 'or', 'migh
tn', 'down', "shan't", 'up', 'very', 'theirs', 'i', 'd', 'few', 'wh
om'}
--------------------------------------------------------------------
-------------
Tokenized Sentence: ['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', '.']
Filterd Sentence: ['tokenization', 'first', 'step', 'text', 'analyt
ics.the', 'process', 'breaking', 'text', 'paragraph', 'smaller', 'c
hunks', 'words', 'sentences', 'called', 'tokenization', '.']
--------------------------------------------------------------------
-------------
wait
wait
wait
wait
--------------------------------------------------------------------
-------------
Lemma for studies is study
Lemma for studying is studying
Lemma for cries is cry
Lemma for cry is cry
--------------------------------------------------------------------
-------------
[('The', 'DT')]
[('pink', 'NN')]
[('sweater', 'NN')]
[('fit', 'NN')]
[('her', 'PRP$')]
[('perfectly', 'RB')]
```

```
In [ ]:
```

In [1]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [2]:
```
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
bagOfWordsA = documentA.split(' ')
bagOfWordsA
```

Out[2]:
```
['Jupiter', 'is', 'the', 'largest', 'Planet']
```

In [3]:
```
bagOfWordsB = documentB.split(' ')
bagOfWordsB
```

Out[3]:
```
['Mars', 'is', 'the', 'fourth', 'planet', 'from', 'the', 'Sun']
```

In [4]:
```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
uniqueWords
```

Out[4]:
```
{'Jupiter',
 'Mars',
 'Planet',
 'Sun',
 'fourth',
 'from',
 'is',
 'largest',
 'planet',
 'the'}
```

In [5]:
```
numOfWordsA = dict.fromkeys(uniqueWords, 0)
```

In [6]:
```
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 [7]:
```
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

In [10]:

```
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[10]:

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

In [11]:

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
df
```

Out[11]:

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

In [ ]: