Algorithm for Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization

In [48]:
```
# Download Required Packages
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\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!
```

Out[48]:
```
True
```

In [49]:
```
# Initialize the text
text= "Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunksuch as words or sentences is called Tokenization."
```

In [50]:
```
# Perform Tokenization
# Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print(tokenized_text)
# Word Tokenization
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 te
xt paragraph into smaller chunksuch as words or sentences is called Tokenization.']
['Tokenization', 'is', 'the', 'first', 'step', 'in', 'text', 'analytics', '.', 'The', 'p
rocess', 'of', 'breaking', 'down', 'a', 'text', 'paragraph', 'into', 'smaller', 'chunksu
ch', 'as', 'words', 'or', 'sentences', 'is', 'called', 'Tokenization', '.']
```

In [51]:
```
# Removing Punctuations and Stop Word
# print stop words of English
```

Loading [MathJax]/extensions/Safe.js

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

```
{"won't", 'myself', "she's", 'under', "shouldn't", 'until', "hadn't", 'because', "that'll", 'shan', 'do', 'a', 'same', 'theirs', 'him', 'mustn', 'haven', 'after', 'isn', 'before', 'mightn', 'their', 'of', 's', "should've", 'aren', 'i', 'doesn', 'has', "aren't", 'than', 'does', 'we', 'those', 'she', 'now', 'an', 'these', 'and', 'himself', 'if', 'ain', "hasn't", 'needn', 'ourselves', 'from', 'where', 'was', 'its', 'by', 'your', "you'd", 'just', 'further', 'yourself', 'any', 'can', 'which', 'being', 'more', 'they', 'were', 'above', 'out', 'yours', 've', 'over', 'about', 'down', 'had', 'so', "needn't", 'below', "mustn't", 'are', 'how', 'here', 'up', 'you', 'at', 'ours', 'not', 'but', 'have', "you're", 'am', 'again', 'itself', 'with', 'll', 'm', 'that', "haven't", 'o', 'other', 'should', 'all', 'some', "you'll", 'don', 'as', "you've", 'in', 'will', 'on', 'who', 'did', 're', 'is', 't', 'whom', 'during', 'for', 'having', 'only', 'yourselves', 'been', 'no', 'it', 'ma', "shan't", 'be', 'off', "couldn't", 'while', "didn't", 'hasn', 'against', 'her', 'doing', 'why', 'he', 'between', 'won', 'own', 'shouldn', 'through', 'then', 'weren', "weren't", 'my', 'couldn', "it's", 'most', 'once', 'into', "wasn't", 'when', 'each', 'hadn', 'wasn', 'hers', 'both', 'this', 'his', 'there', 'themselves', "don't", 'few', "wouldn't", 'wouldn', "isn't", 'didn', "mightn't", 'them', 'the', 'nor', 'y', 'me', 'very', "doesn't", 'too', 'what', 'such', 'd', 'herself', 'our', 'or', 'to'}
Tokenized Sentence: ['how', 'to', 'remove', 'stop', 'words', 'with', 'nltk', 'library', 'in', 'python']
Filterd Sentence: ['remove', 'stop', 'words', 'nltk', 'library', 'python']
```

In [52]:

```
# Perform Stemming
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 [57]:

```
# Perform Lemmatization
```

Loading [MathJax]/extensions/Safe.js

```
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
    print(f"Lemma for '{w}' is '{wordnet_lemmatizer.lemmatize(w)}'")
```

```
Lemma for 'studies' is 'study'
Lemma for 'studying' is 'studying'
Lemma for 'cries' is 'cry'
Lemma for 'cry' is 'cry'
```

```
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\Hp\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

In [58]:

```
# Apply POS Tagging to text
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')]
```

Algorithm for Create representation of document by calculating TFIDF

In [59]:

```
# Import the necessary libraries.
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [60]:

```
# Initialize the Documents.
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

In [61]:

```
# Create BagofWords (BoW) for Document A and B.
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

Loading [MathJax]/extensions/Safe.js

In [62]:
```
# Create Collection of Unique words from Document A and B.
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

In [64]:
```
# Create a dictionary of words and their occurrence for each document in the corpus
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 [66]:
```
# Compute the term frequency for each of our documents.
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 [67]:
```
# Compute the term Inverse Document Frequency.
import math
def computeIDF(documents):
    N = len(documents)
    idfDict = dict.fromkeys(documents[0].keys(), 0.0)
    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
```

Loading [MathJax]/extensions/Safe.js

Out[67]:

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

In [70]:

```
# Compute the term TF/IDF for all words.
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[70]:

	the
0	0.0
1	0.0

Loading [MathJax]/extensions/Safe.js