Text Analytics

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.

```
# Setup
!pip install -q wordcloud
import wordcloud
```

```
import nltk
# Step 1: Download the required packages
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
True
```

```
# Step 2: Initialize the text
text= "Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or s
```

Saving...

```
#Sentence Tokenization
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 breaking down a text paragraph into smaller chunks such as words o
```

```
#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', 'text',
```

```
# Step 4: Removing Punctuations and Stop Word
# print stop words of English
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)
```

```
{'isn', 'those', 'him', "don't", 'themselves', 'weren', 'she', 'own', 'was', 'above', 'himself', 'from', 'all', 'aren', 'too', 'ours',
```

```
# Step 4: Removing Punctuations and Stop Word
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())
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', 'stop', 'words', 'nltk', 'library', 'python']
```

```
# Step 5 : 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
```

```
# Step 6: Perform Lemmatization
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
```

```
# Step 7: 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:
```

Saving...

```
[('pink', 'NN')]
[('sweater', 'NN')]
[('fit', 'NN')]
[('her', 'PRP$')]
[('perfectly', 'RB')]
```

Algorithm for Create representation of document by calculating TFIDF

```
# Step 1: Import the necessary libraries.
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
# Step 2: Initialize the Documents.
documentA = 'Jupiter is the largest Planet'
documentB = 'Mars is the fourth planet from the Sun'
```

```
# Step 3: Create BagofWords (BoW) for Document A and B.
bagOfWordsA = documentA.split(' ')
print(bagOfWordsA)
bagOfWordsB = documentB.split(' ')
print(bagOfWordsB)
```

```
['Jupiter', 'is', 'the', 'largest', 'Planet']
['Mars', 'is', 'the', 'fourth', 'planet', 'from', 'the', 'Sun']
```

```
# Step 4: Create Collection of Unique words from Document A and B.
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
uniqueWords
```

```
{'Jupiter',
 'Mars',
 'Planet',
 'Sun',
 'fourth',
```

```
       'from',
       'is',
       'largest',
       'planet',
       'the'}


# Step 5: 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


numOfWordsA
```

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

```
numOfWordsB
```

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



```
# Step 6: 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)



tfA = computeTF(numOfWordsA, bagOfWordsA)


tfB = computeTF(numOfWordsB, bagOfWordsB)


# Step 7: Compute the term Inverse Document Frequency.
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



idfs = computeIDF([numOfWordsA, numOfWordsB])
idfs
```

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


# Step 8: 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])



tfidf = computeTFIDF(numOfWordsA, idfs)
tfidf
```

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

Saving...

```
import math
from collections import Counter

# Sample corpus
corpus = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?",
]

# Function to calculate term frequency (TF)
def calculate_tf(document):
    word_counts = Counter(document.split())
    total_words = len(document.split())
    tf = {word: count / total_words for word, count in word_counts.items()}
    return tf

# Function to calculate inverse document frequency (IDF)
def calculate_idf(corpus):
    total_documents = len(corpus)
    idf = {}
    word_document_counts = Counter()
    for document in corpus:
        unique_words = set(document.split())
        word_document_counts.update(unique_words)
    for word, count in word_document_counts.items():
        idf[word] = math.log(total_documents / count)
    return idf

# Example document
document = "This is the first document."

# Calculate TF for the document
tf = calculate_tf(document)

# Calculate IDF for the corpus
```

```
idf = calculate_idf(corpus)

# Print TF for the document
print("Term Frequency (TF) for the document:")
for word, tf_value in tf.items():
    print(f"    Word: {word}, TF: {tf_value:.4f}")

# Print IDF for the corpus
print("Inverse Document Frequency (IDF) for the corpus:")
for word, idf_value in idf.items():
    print(f"    Word: {word}, IDF: {idf_value:.4f}")
```

```
 Term Frequency (TF) for the document:
    Word: This, TF: 0.2000
    Word: is, TF: 0.2000
    Word: the, TF: 0.2000
    Word: first, TF: 0.2000
    Word: document., TF: 0.2000
 Inverse Document Frequency (IDF) for the corpus:
    Word: This, IDF: 0.6931
    Word: is, IDF: 0.2877
    Word: document., IDF: 0.6931
    Word: first, IDF: 0.6931
    Word: the, IDF: 0.0000
    Word: document, IDF: 1.3863
    Word: second, IDF: 1.3863
    Word: third, IDF: 1.3863
    Word: And, IDF: 1.3863
    Word: one., IDF: 1.3863
    Word: this, IDF: 0.6931
    Word: document?, IDF: 1.3863
    Word: Is, IDF: 1.3863
```

```
import math

    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

def computeTFIDF(tfBagOfWords, idfs):
    tfidfDict = {}

    for word, val in tfBagOfWords.items():
        tfidfDict[word] = val * idfs[word]

    return tfidfDict

# Example documents and TF bag of words
documents = [
    {"apple": 2, "banana": 1, "orange": 0},
    {"apple": 1, "banana": 0, "orange": 2}
]

# Compute IDF
idfs = computeIDF(documents)

# Compute TF-IDF for each document
tfidfDocuments = []
for document in documents:
    tfidf = computeTFIDF(document, idfs)
    tfidfDocuments.append(tfidf)

# Print IDF values
print("IDF values:")
for word, value in idfs.items():
    print(f"{word}: {value}")
```

```
# Print TF-IDF values for each document
print("\nTF-IDF values for each document:")
for i, tfidf in enumerate(tfidfDocuments):
    print(f"Document {i+1}:")
    for word, value in tfidf.items():
        print(f"{word}: {value}")
```

```
 IDF values:
 apple: 0.0
 banana: 0.6931471805599453
 orange: 0.6931471805599453

 TF-IDF values for each document:
 Document 1:
 apple: 0.0
 banana: 0.6931471805599453
 orange: 0.0
 Document 2:
 apple: 0.0
 banana: 0.0
 orange: 1.3862943611198906
```