Name : Suryal D . Khirade
Roll NO: T190424399
Assignment No :07

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.

Part 1:

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

In [1]:
```
#Installation of punkt from nltk
import nltk
nltk.download('punkt')
## The NLTK data package includes a pre-trained Punkt tokenizer for English.
### PUNKT is unsupervised trainable model, which means it can be trained on unl
#(Data that has not been tagged with information identifying its characteristic
#properties, or categories is referred to as unlabeled data.)###
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\alisu\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
```

Out[1]:
```
True
```

1.TOKENIZATION

In [4]:
```
from nltk import word_tokenize, sent_tokenize
sent="Sachin is considered to be one of the greatest cricket players. Rohit Sharma is
print(word_tokenize(sent))
print(sent_tokenize(sent))
```

```
['Sachin', 'is', 'considered', 'to', 'be', 'one', 'of', 'the', 'greatest', 'cricke
t', 'players', '.', 'Rohit', 'Sharma', 'is', 'the', 'captain', 'of', 'the', 'India
n', 'cricket', 'team']
['Sachin is considered to be one of the greatest cricket players.', 'Rohit Sharma is
the captain of the Indian cricket team']
```

2.Stop Words Removal

In [5]:
```
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
stop_words = stopwords.words('english')
print(stop_words)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\alisu\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.

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

In [6]:
```
len(stop_words)
```

Out[6]: 179

In [8]:
```
token = word_tokenize(sent)
cleaned_token = []
for word in token:
 if word not in stop_words:
     cleaned_token.append(word)
print("This is the unclean version : ",token)
print("This is the cleaned version : ",cleaned_token)
```

```
This is the unclean version :  ['Sachin', 'is', 'considered', 'to', 'be', 'one', 'o
f', 'the', 'greatest', 'cricket', 'players', '.', 'Rohit', 'Sharma', 'is', 'the', 'c
aptain', 'of', 'the', 'Indian', 'cricket', 'team']
This is the cleaned version :  ['Sachin', 'considered', 'one', 'greatest', 'cricke
t', 'players', '.', 'Rohit', 'Sharma', 'captain', 'Indian', 'cricket', 'team']
```

In [9]:
```
1 len(token)
```

Out[9]: 22

In [10]:
```
len(cleaned_token)
```

Out[10]: 13

In [12]:
```
words = [cleaned_token.lower() for cleaned_token in cleaned_token
         if cleaned_token.isalpha()]
```

In [13]:
```
print(words)
```

```
['sachin', 'considered', 'one', 'greatest', 'cricket', 'players', 'rohit', 'sharma',
'captain', 'indian', 'cricket', 'team']
```

3.Stemming

Stemming just removes or stems the last few characters of a word, often leading to incorrect meanings and spelling.

In [14]:
```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
port_stemmer_output = [stemmer.stem(words) for words in words]
print(port_stemmer_output)
```

```
['sachin', 'consid', 'one', 'greatest', 'cricket', 'player', 'rohit', 'sharma', 'cap
tain', 'indian', 'cricket', 'team']
```

4.Lemmatization

Lemmaization considers the context and converts the word to its meaningul base form , which is called Lemma

In [16]:
```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
lemmatizer_output = [lemmatizer.lemmatize(words) for words in words]
print(lemmatizer_output)
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\alisu\AppData\Roaming\nltk_data...
```

```
['sachin', 'considered', 'one', 'greatest', 'cricket', 'player', 'rohit', 'sharma',
'captain', 'indian', 'cricket', 'team']
```

5.POS Tagging

In [18]:
```
from nltk import pos_tag
import nltk
nltk.download('averaged_perceptron_tagger')
token = word_tokenize(sent)
cleaned_token = []
for word in token:
    if word not in stop_words:
         cleaned_token.append(word)
tagged = pos_tag(cleaned_token)
print(tagged)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\alisu\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\averaged_perceptron_tagger.zip.
```

```
[('Sachin', 'NNP'), ('considered', 'VBD'), ('one', 'CD'), ('greatest', 'JJS'), ('cri
cket', 'NN'), ('players', 'NNS'), ('.', '.'), ('Rohit', 'NNP'), ('Sharma', 'NNP'),
('captain', 'NN'), ('Indian', 'JJ'), ('cricket', 'NN'), ('team', 'NN')]
```

Part 2:

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

In [19]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
```

In [20]:
```
docs = [ "Sachin is considered to be one of the greatest cricket players",
 "Federer is considered one of the greatest tennis players",
 "Nadal is considered one of the greatest tennis players",
 "Rohit is the captain of the Indian cricket team"]
```

In [22]:
```
vectorizer = TfidfVectorizer(analyzer = "word", norm = None , use_idf = True , smooth
Mat = vectorizer.fit(docs)
print(Mat.vocabulary_)
```

```
{'sachin': 13, 'is': 7, 'considered': 2, 'to': 17, 'be': 0, 'one': 10, 'of': 9, 'th
e': 16, 'greatest': 5, 'cricket': 3, 'players': 11, 'federer': 4, 'tennis': 15, 'nad
al': 8, 'rohit': 12, 'captain': 1, 'indian': 6, 'team': 14}
```

In [23]:
```
tfidfMat = vectorizer.fit_transform(docs)
```

In [24]:
```
print(tfidfMat)
```

```
  (0, 11)       1.2231435513142097
  (0, 3)        1.5108256237659907
  (0, 5)        1.2231435513142097
  (0, 16)       1.0
  (0, 9)        1.0
  (0, 10)       1.2231435513142097
  (0, 0)        1.916290731874155
  (0, 17)       1.916290731874155
  (0, 2)        1.2231435513142097
  (0, 7)        1.0
  (0, 13)       1.916290731874155
  (1, 15)       1.5108256237659907
  (1, 4)        1.916290731874155
  (1, 11)       1.2231435513142097
  (1, 5)        1.2231435513142097
  (1, 16)       1.0
  (1, 9)        1.0
  (1, 10)       1.2231435513142097
  (1, 2)        1.2231435513142097
  (1, 7)        1.0
  (2, 8)        1.916290731874155
  (2, 15)       1.5108256237659907
  (2, 11)       1.2231435513142097
  (2, 5)        1.2231435513142097
  (2, 16)       1.0
  (2, 9)        1.0
  (2, 10)       1.2231435513142097
  (2, 2)        1.2231435513142097
  (2, 7)        1.0
  (3, 14)       1.916290731874155
  (3, 6)        1.916290731874155
  (3, 1)        1.916290731874155
  (3, 12)       1.916290731874155
  (3, 3)        1.5108256237659907
  (3, 16)       2.0
  (3, 9)        1.0
  (3, 7)        1.0
```

In [25]:
```
features_names = vectorizer.get_feature_names_out()
print(features_names)
```

```
['be' 'captain' 'considered' 'cricket' 'federer' 'greatest' 'indian' 'is'
 'nadal' 'of' 'one' 'players' 'rohit' 'sachin' 'team' 'tennis' 'the' 'to']
```

In [26]:
```
dense = tfidfMat.todense()
denselist = dense.tolist()
df = pd.DataFrame(denselist , columns = features_names)
```

In [27]:
```
df
```

Out[27]:

	be	captain	considered	cricket	federer	greatest	indian	is	nadal	of	one
0	1.916291	0.000000	1.223144	1.510826	0.000000	1.223144	0.000000	1.0	0.000000	1.0	1.223144
1	0.000000	0.000000	1.223144	0.000000	1.916291	1.223144	0.000000	1.0	0.000000	1.0	1.223144
2	0.000000	0.000000	1.223144	0.000000	0.000000	1.223144	0.000000	1.0	1.916291	1.0	1.223144
3	0.000000	1.916291	0.000000	1.510826	0.000000	0.000000	1.916291	1.0	0.000000	1.0	0.000000

In [30]:
```
docList = ['Doc 1','Doc 2','Doc 3','Doc 4']
skDocsIfIdfdf = pd.DataFrame(tfidfMat.todense(),index = sorted(docList), columns=feat
print(skDocsIfIdfdf)
```

```
             be   captain  considered   cricket   federer  greatest    indian  \
Doc 1  1.916291  0.000000    1.223144  1.510826  0.000000  1.223144  0.000000
Doc 2  0.000000  0.000000    1.223144  0.000000  1.916291  1.223144  0.000000
Doc 3  0.000000  0.000000    1.223144  0.000000  0.000000  1.223144  0.000000
Doc 4  0.000000  1.916291    0.000000  1.510826  0.000000  0.000000  1.916291

            is     nadal   of       one   players     rohit    sachin      team  \
Doc 1  1.0  0.000000  1.0  1.223144  1.223144  0.000000  1.916291  0.000000
Doc 2  1.0  0.000000  1.0  1.223144  1.223144  0.000000  0.000000  0.000000
Doc 3  1.0  1.916291  1.0  1.223144  1.223144  0.000000  0.000000  0.000000
Doc 4  1.0  0.000000  1.0  0.000000  0.000000  1.916291  0.000000  1.916291

          tennis  the        to
Doc 1  0.000000  1.0  1.916291
Doc 2  1.510826  1.0  0.000000
Doc 3  1.510826  1.0  0.000000
Doc 4  0.000000  2.0  0.000000
```

In [31]:
```
#Compute Cosine Similarity
csim = cosine_similarity(tfidfMat,tfidfMat)
```

In [32]:
```
csimDf = pd.DataFrame(csim,index=sorted(docList),columns=sorted(docList))
```

In [33]:
```
print(csimDf)
```

```
           Doc 1     Doc 2     Doc 3     Doc 4
Doc 1  1.000000  0.492416  0.492416  0.277687
Doc 2  0.492416  1.000000  0.754190  0.215926
Doc 3  0.492416  0.754190  1.000000  0.215926
Doc 4  0.277687  0.215926  0.215926  1.000000
```

In [ ]:
```

```