INDEX

EXP NO	TITLE	PAGE NO	DATE	SIGN
1	Implement Sentiment Analysis by a movie			
2	Implement Named Entity Recognition (NER) in Python with Spacy			
3	Implement Stemming & Lemmatization			
4	Implement Bag of Words			
5	Implement Term Frequency–Inverse Document Frequency (TF-IDF)			
6	Implement Stopwords			
7	Implement POS Tagging			
8	Implement Chunking			
9	Implement WordNet			
10	Implement Word Cloud			

Exp No:1

Implement Sentiment Analysis by a movie

Aim: To implement Sentiment Analysis by a movie.

Code:

```
import pandas as pd

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer


# reading and wragling data

df_avatar = pd.read_csv('avatar.csv', engine='python')

df_avatar_lines = df_avatar.groupby('character').count()

df_avatar_lines = df_avatar_lines.sort_values(by=['character_words'], ascending=False)[:10]

top_character_names = df_avatar_lines.index.values


# filtering out non-top characters

df_character_sentiment = df_avatar[df_avatar['character'].isin(top_character_names)]

df_character_sentiment = df_character_sentiment[['character', 'character_words']]


# calculating sentiment score

sid = SentimentIntensityAnalyzer()

df_character_sentiment.reset_index(inplace=True, drop=True)

df_character_sentiment[['neg', 'neu', 'pos', 'compound']] =
df_character_sentiment['character_words'].apply(sid.polarity_scores).apply(pd.Series)

df_character_sentiment
```

Output:

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

	character	character_words	neg	neu	pos	compound
0	Katara	Water. Earth. Fire. Air. My grandmother used t...	0.196	0.735	0.069	-0.9718
1	Sokka	It's not getting away from me this time. Watc...	0.000	1.000	0.000	0.0000
2	Katara	Sokka, look!	0.000	1.000	0.000	0.0000
3	Sokka	Sshh! Katara, you're going to scare it away. ...	0.200	0.800	0.000	-0.5411
4	Katara	But, Sokka! I caught one!	0.000	1.000	0.000	0.0000
...	...	...	...	...	...	...
7053	Zuko	At least you don't look like a boar-q-pine! My...	0.183	0.817	0.000	-0.4007
7054	Suki	And why did you paint me firebending?	0.000	1.000	0.000	0.0000
7055	Sokka	I thought it looked more exciting that way. O...	0.000	0.687	0.313	0.7501
7056	Iroh	Hey, my belly's not that big anymore. I've rea...	0.000	1.000	0.000	0.0000
7057	Toph	Well I think you all look perfect!	0.000	0.396	0.604	0.7263

7058 rows × 6 columns

Result:

Sentiment Analysis by a movie is successfully implemented.

Exp No:2

Implement Named Entity Recognition (NER) in Python with Spacy

Aim: To Implement Named Entity Recognition (NER) in Python with Spacy.

Code:

```
! pip install spacy

! pip install nltk

! python -m spacy download en_core_web_sm

import spacy

from spacy import displacy


NER = spacy.load("en_core_web_sm")

raw_text="The Indian Space Research Organisation or is the national space agency of India, headquartered in Bengaluru. It operates under Department of Space which is directly overseen by the Prime Minister of India while Chairman of ISRO acts as executive of DOS as well."

text1= NER(raw_text)

for word in text1.ents:

  print(word.text,word.label_)
```

Output:

```
import spacy
from spacy import displacy

NER = spacy.load("en_core_web_sm")
raw_text="The Indian Space Research Organisation or is the national space agency of India, headquartered in Bengaluru. It operate
text1= NER(raw_text)
for word in text1.ents:
    print(word.text,word.label_)
```

```
The Indian Space Research Organisation ORG
India GPE
Bengaluru ORG
Department of Space ORG
India GPE
```

Result:

NER in python with spacy is implemented successfully.

Exp No:3

Implement Stemming & Lemmatization

Aim: To Implement Stemming & Lemmatization

Code:

Stemming

```
import nltk
from nltk.stem.porter import PorterStemmer
porter_stemmer  = PorterStemmer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
 print("Stemming for {} is {}".format(w,porter_stemmer.stem(w)))
```

Output:

```
import nltk
from nltk.stem.porter import PorterStemmer
porter_stemmer  = PorterStemmer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
 print("Stemming for {} is {}".format(w,porter_stemmer.stem(w)))
```

```
Stemming for studies is studi
Stemming for studying is studi
Stemming for cries is cri
Stemming for cry is cri
```

Lemmatization

```
import nltk
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)))
```

Output:

```
import nltk
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
```

Result:

Stemming & Lemmatization is successfully implemented.

Exp No:4

Implement Bag of Words

Aim: To implement Bag of Words.

Code:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]
df = pd.DataFrame({'review': ['review1', 'review2'], 'text':text})
cv = CountVectorizer(stop_words='english')
cv_matrix = cv.fit_transform(df['text'])
df_dtm = pd.DataFrame(cv_matrix.toarray(),
                      index=df['review'].values,
                      columns=cv.get_feature_names())
df_dtm
```

Output:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]
df = pd.DataFrame({'review': ['review1', 'review2'], 'text':text})
cv = CountVectorizer(stop_words='english')
cv_matrix = cv.fit_transform(df['text'])
df_dtm = pd.DataFrame(cv_matrix.toarray(),
                      index=df['review'].values,
                      columns=cv.get_feature_names())
df_dtm
```

```
C:\Users\k.chaitya\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is de
precated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	code	hate	java	love	python	writing
review1	2	0	0	2	2	1
review2	2	2	2	0	0	1

Result:

Bag of Words implemented successfully.

Exp No:5

Implement Term Frequency–Inverse Document Frequency(TF-IDF)

Aim: To implement Term Frequency -Inverse Document Frequency(TF-IDF).

Code:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]
df = pd.DataFrame({'review': ['review1', 'review2'], 'text':text})
tfidf = TfidfVectorizer(stop_words='english', norm=None)
tfidf_matrix = tfidf.fit_transform(df['text'])
df_dtm = pd.DataFrame(tfidf_matrix.toarray(),
                      index=df['review'].values,
                      columns=tfidf.get_feature_names())
df_dtm
```

Output:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
text = ["I love writing code in Python. I love Python code",
        "I hate writing code in Java. I hate Java code"]
df = pd.DataFrame({'review': ['review1', 'review2'], 'text':text})
tfidf = TfidfVectorizer(stop_words='english', norm=None)
tfidf_matrix = tfidf.fit_transform(df['text'])
df_dtm = pd.DataFrame(tfidf_matrix.toarray(),
                      index=df['review'].values,
                      columns=tfidf.get_feature_names())
df_dtm
```

```
C:\Users\k.chaitya\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names is de
precated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

	code	hate	java	love	python	writing
review1	2.0	0.00000	0.00000	2.81093	2.81093	1.0
review2	2.0	2.81093	2.81093	0.00000	0.00000	1.0

Result: Term Frequency -Inverse Document Frequency(TF-IDF) is implemented successfully.

Exp No:6

Implement Stopwords

Aim: To implement stopwords.

Code:

```
import nltk

from nltk.corpus import stopwords

sw_nltk = stopwords.words('english')

print(sw_nltk)

print(len(sw_nltk))

text = "When I first met her she was very quiet. She remained quiet during the entire two hour long journey from Stony Brook to New York."

words = [word for word in text.split() if word.lower() not in sw_nltk]

new_text = " ".join(words)

print(new_text)

print("Old length: ", len(text))

print("New length: ", len(new_text))
```

Output:

```
import nltk
from nltk.corpus import stopwords
sw_nltk = stopwords.words('english')
print(sw_nltk)
print(len(sw_nltk))
text = "When I first met her she was very quiet. She remained quiet during the entire two hour long journey from Stony Brook to N
words = [word for word in text.split() if word.lower() not in sw_nltk]
new_text = " ".join(words)
print(new_text)
print("Old length: ", len(text))
print("New length: ", len(new_text))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'y
ourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those',
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'a
n', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'b
etween', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
f', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'ar
en', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "have
n't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
179
first met quiet. remained quiet entire two hour long journey Stony Brook New York.
Old length:  129
New length:  82
```

Result: Stopwords successfully implemented.

Exp No:7

Implement POS Tagging

Aim: To implement POS Tagging.

Code:

```
 import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
stop_words = set(stopwords.words('english'))

txt = "Sukanya, Rajib and Naba are my good friends. " \
   "Sukanya is getting married next year. " \
   "Marriage is a big step in one’s life." \
   "It is both exciting and frightening. " \
   "But friendship is a sacred bond between people." \
   "It is a special kind of love between us. " \
   "Many of you must have tried searching for a friend "\
   "but never found the right one."

# sent_tokenize is one of instances of
# PunktSentenceTokenizer from the nltk.tokenize.punkt module

tokenized = sent_tokenize(txt)
for i in tokenized:

   # Word tokenizers is used to find the words
   # and punctuation in a string
   wordsList = nltk.word_tokenize(i)

   # removing stop words from wordList
   wordsList = [w for w in wordsList if not w in stop_words]
```

```
#  Using a Tagger. Which is part-of-speech

# tagger or POS-tagger.

tagged = nltk.pos_tag(wordsList)


print(tagged)
```

Output:

```
tokenized = sent_tokenize(txt)
for i in tokenized:

    # Word tokenizers is used to find the words
    # and punctuation in a string
    wordsList = nltk.word_tokenize(i)

    # removing stop words from wordList
    wordsList = [w for w in wordsList if not w in stop_words]

    #  Using a Tagger. Which is part-of-speech
    # tagger or POS-tagger.
    tagged = nltk.pos_tag(wordsList)

    print(tagged)
```

```
[('Sukanya', 'NNP'), (',', ','), ('Rajib', 'NNP'), ('Naba', 'NNP'), ('good', 'JJ'), ('friends', 'NNS'), ('.', '.')]
[('Sukanya', 'NNP'), ('getting', 'VBG'), ('married', 'VBN'), ('next', 'JJ'), ('year', 'NN'), ('.', '.')]
[('Marriage', 'NN'), ('big', 'JJ'), ('step', 'NN'), ('one', 'CD'), ('’', 'NN'), ('life.It', 'NN'), ('exciting', 'VBG'), ('frigh
tening', 'NN'), ('.', '.')]
[('But', 'CC'), ('friendship', 'NN'), ('sacred', 'VBD'), ('bond', 'NN'), ('people.It', 'NN'), ('special', 'JJ'), ('kind', 'N
N'), ('love', 'VB'), ('us', 'PRP'), ('.', '.')]
[('Many', 'JJ'), ('must', 'MD'), ('tried', 'VB'), ('searching', 'VBG'), ('friend', 'NN'), ('never', 'RB'), ('found', 'VBD'),
('right', 'JJ'), ('one', 'CD'), ('.', '.')]
```

Result:

POS Tagging is successfully implemented.

Exp No:8

Implement Chunking

Aim: To implement Chunking.

Code:

```
import nltk

sentence = [

  ("the", "DT"),

  ("book", "NN"),

  ("has","VBZ"),

  ("many","JJ"),

  ("chapters","NNS")

]

chunker = nltk.RegexpParser(

  r'''

  NP:{<DT><NN.*><.*>*<NN.*>}

  }<VB.*>{

  '''

)

chunker.parse(sentence)

Output = chunker.parse(sentence)

print(Output)
```

Output:

```
import nltk
sentence = [
   ("the", "DT"),
   ("book", "NN"),
   ("has","VBZ"),
   ("many","JJ"),
   ("chapters","NNS")
]
chunker = nltk.RegexpParser(
   r'''
   NP:{<DT><NN.*><.*>*<NN.*>}
   }<VB.*>{
   '''
)
chunker.parse(sentence)
Output = chunker.parse(sentence)
print(Output)
```

```
(S (NP the/DT book/NN) has/VBZ (NP many/JJ chapters/NNS))
```

Result: Chunking implemented successfully.

Exp No:9

Implement WordNet

Aim: To implement WordNet.

Code:

```
import nltk

from nltk.corpus import wordnet

synonyms = []

antonyms = []


for synset in wordnet.synsets("evil"):

   for l in synset.lemmas():

      synonyms.append(l.name())

      if l.antonyms():

         antonyms.append(l.antonyms()[0].name())

print(set(synonyms))

print(set(antonyms))
```

Output:

```
import nltk
from nltk.corpus import wordnet
synonyms = []
antonyms = []

for synset in wordnet.synsets("evil"):
    for l in synset.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())
print(set(synonyms))
print(set(antonyms))
```

```
{'malefic', 'immorality', 'wickedness', 'iniquity', 'malevolent', 'malign', 'evilness', 'vicious', 'evil'}
{'good', 'goodness'}
```

Result:

WordNet successfully implemented.

Exp No:10

Implement Word Cloud

Aim: To implement Word Cloud.

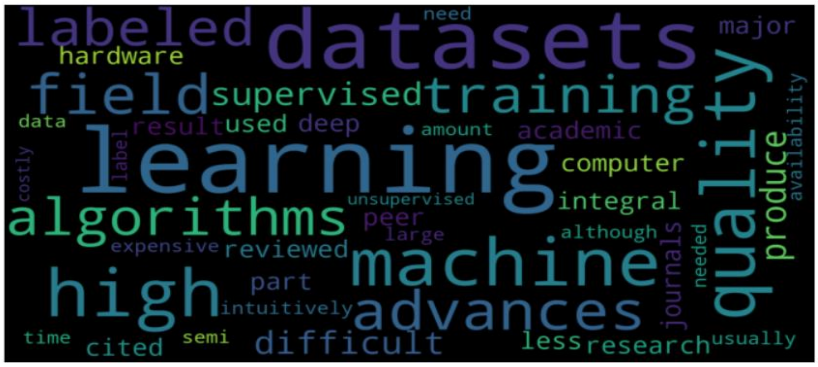
Code:

```
import matplotlib.pyplot as plt

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from wordcloud import WordCloud

class WordCloudGeneration:

    def preprocessing(self, data):

        # convert all words to lowercase

        data = [item.lower() for item in data]

        # load the stop_words of english

        stop_words = set(stopwords.words('english'))

        # concatenate all the data with spaces.

        paragraph = ' '.join(data)

        # tokenize the paragraph using the inbuilt tokenizer

        word_tokens = word_tokenize(paragraph)

        # filter words present in stopwords list

        preprocessed_data = ' '.join([word for word in word_tokens if not word in stop_words])

        print("\n Preprocessed Data: " ,preprocessed_data)

        return preprocessed_data


    def create_word_cloud(self, final_data):

        # initiate WordCloud object with parameters width, height, maximum font size and background
color
```

```
        # call the generate method of WordCloud class to generate an image

        wordcloud = WordCloud(width=1600, height=800, max_font_size=200,
background_color="black").generate(final_data)

        # plt the image generated by WordCloud class

        plt.figure(figsize=(12,10))

        plt.imshow(wordcloud)

        plt.axis("off")

        plt.show()

wordcloud_generator = WordCloudGeneration()

# you may uncomment the following line to use custom input

# input_text = input("Enter the text here: ")

input_text = 'These datasets are used for machine-learning research and have been cited in peer-reviewed academic journals. Datasets are an integral part of the field of machine learning. Major advances in this field can result from advances in learning algorithms (such as deep learning), computer hardware, and, less-intuitively, the availability of high-quality training datasets.[1] High-quality labeled training datasets for supervised and semi-supervised machine learning algorithms are usually difficult and expensive to produce because of the large amount of time needed to label the data. Although they do not need to be labeled, high-quality datasets for unsupervised learning can also be difficult and costly to produce.'

input_text = input_text.split('.')

clean_data = wordcloud_generator.preprocessing(input_text)

wordcloud_generator.create_word_cloud(clean_data)
```

Output:

Preprocessed Data: datasets used machine-learning research cited peer-reviewed academic journals datasets integral part field machine learning major advances field result advances learning algorithms (deep learning) , computer hardware , , less-intuitively , availability high-quality training datasets [1] high-quality labeled training datasets supervised semi-supervised machine learning algorithms usually difficult expensive produce large amount time needed label data although need labeled , high-quality datasets unsupervised learning also difficult costly produce



Result:

Word Cloud successfully implemented.