EX.NO : 01	**Fundamentals of NLP I Tokenization & Lemmatization**
DATE : 04/07/2023	

AIM :

To create tokenization and lemmatization programs

THEORY:

- **Tokenization :**
  Tokenization is the process of replacing sensitive data with unique identification symbols that retain all the essential information about the data without compromising its security. Tokenization, which seeks to minimize the amount of sensitive data a business needs to keep on hand, has become a popular way for small and midsize businesses to bolster the security of credit card and e-commerce transactions while minimizing the cost and complexity of compliance with industry standards and government regulations
- **Lemmatization:**

  Lemmatization is another technique used to reduce inflected words to their root word. It describes the algorithmic process of identifying an inflected word's "**lemma**" (dictionary form) based on its intended meaning.

  As opposed to stemming, lemmatization relies on accurately determining the intended **part-of-speech** and the meaning of a word based on its context. This means it takes into consideration where the inflected word falls within a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

PROCEDURE :

- Import Libraries
- Tokenization
- Lemmatization

IMPORT LIBRARIES :

```
import nltk
from nltk import sent_tokenize, word_tokenize
```

TEXT DATA :

```
text_file = open("C:\\Users\\mayur\\Downloads\\NLP.txt")
text = text_file.read()
print(text)
print("lenth",len(text))
```

```
In this article, we explore the basics of natural language processing (NLP) with code examples. We dive into the natural langua
ge toolkit (NLTK) library to present how it can be useful for natural language processing related-tasks. Afterward, we will dis
cuss the basics of other Natural Language Processing libraries and other essential methods for NLP, along with their respective
coding sample implementations in Python.
lenth 422
```

SENTENCE TOKENIZATION :

```
import nltk
from nltk import sent_tokenize
sent = sent_tokenize(text)
print(sent)
print()
print("Number of sentences =",len(sent))
```

```
['In this article, we explore the basics of natural language processing (NLP) with code examples.', 'We dive into the natural l
anguage toolkit (NLTK) library to present how it can be useful for natural language processing related-tasks.', 'Afterward, we
will discuss the basics of other Natural Language Processing libraries and other essential methods for NLP, along with their re
spective coding sample implementations in Python.']

Number of sentences = 3
```

WORD TOKENIZATION :

```
from nltk import word_tokenize
word = word_tokenize(text)
print("Lenthe of WORD = ",len(word))
print()
print(word)
```

```
Lenthe of WORD =  73

['In', 'this', 'article', ',', 'we', 'explore', 'the', 'basics', 'of', 'natural', 'language', 'processing', '(', 'NLP', ')', 'w
ith', 'code', 'examples', '.', 'We', 'dive', 'into', 'the', 'natural', 'language', 'toolkit', '(', 'NLTK', ')', 'library', 't
o', 'present', 'how', 'it', 'can', 'be', 'useful', 'for', 'natural', 'language', 'processing', 'related-tasks', '.', 'Afterwar
d', ',', 'we', 'will', 'discuss', 'the', 'basics', 'of', 'other', 'Natural', 'Language', 'Processing', 'libraries', 'and', 'oth
er', 'essential', 'methods', 'for', 'NLP', ',', 'along', 'with', 'their', 'respective', 'coding', 'sample', 'implementations',
'in', 'Python', '.']
```

TREE BANK TOKENIZER :

```
from nltk.tokenize import TreebankWordTokenizer
token = TreebankWordTokenizer()
print(token.tokenize(text))

```

```
['In', 'this', 'article', ',', 'we', 'explore', 'the', 'basics', 'of', 'natural', 'language', 'processing', '(', 'NLP', ')', 'w
ith', 'code', 'examples.', 'We', 'dive', 'into', 'the', 'natural', 'language', 'toolkit', '(', 'NLTK', ')', 'library', 'to', 'p
resent', 'how', 'it', 'can', 'be', 'useful', 'for', 'natural', 'language', 'processing', 'related-tasks.', 'Afterward', ',', 'w
e', 'will', 'discuss', 'the', 'basics', 'of', 'other', 'Natural', 'Language', 'Processing', 'libraries', 'and', 'other', 'essen
tial', 'methods', 'for', 'NLP', ',', 'along', 'with', 'their', 'respective', 'coding', 'sample', 'implementations', 'in', 'Pyth
on', '.']
```

PunktSentence Tokenizer :

```
from nltk.tokenize import PunktSentenceTokenizer
pu_token = PunktSentenceTokenizer()
punkt = pu_token.tokenize(text)
print(punkt)
print("Lenth",len(punkt))
```

```
['In this article, we explore the basics of natural language processing (NLP) with code examples.', 'We dive into the natural l
anguage toolkit (NLTK) library to present how it can be useful for natural language processing related-tasks.', 'Afterward, we
will discuss the basics of other Natural Language Processing libraries and other essential methods for NLP, along with their re
spective coding sample implementations in Python.']
Lenth 3
```

MWE TOKENIZER :

```
from nltk.tokenize import MWETokenizer
from nltk.tokenize import word_tokenize
token = MWETokenizer()
token.add_mwe(("In","this","article"))
print(token.tokenize(word_tokenize(text)))



```

```
['In_this_article', ',', 'we', 'explore', 'the', 'basics', 'of', 'natural', 'language', 'processing', '(', 'NLP', ')', 'with',
'code', 'examples', '.', 'We', 'dive', 'into', 'the', 'natural', 'language', 'toolkit', '(', 'NLTK', ')', 'library', 'to', 'pre
sent', 'how', 'it', 'can', 'be', 'useful', 'for', 'natural', 'language', 'processing', 'related-tasks', '.', 'Afterward', ',',
'we', 'will', 'discuss', 'the', 'basics', 'of', 'other', 'Natural', 'Language', 'Processing', 'libraries', 'and', 'other', 'ess
ential', 'methods', 'for', 'NLP', ',', 'along', 'with', 'their', 'respective', 'coding', 'sample', 'implementations', 'in', 'Py
thon', '.']
```

LEMMATIZATION :

```
from nltk.stem import WordNetLemmatizer
lemma = WordNetLemmatizer()
for i in list:
    print(lemma.lemmatize(i))
```

```
listing
listed
studying
foot
Studied
```

```
lemma.lemmatize("i am walking toward home")
```

```
'i am walking toward home'
```

```
text = "A mountain is an elevated portion of the Earth's crust, generally with steep sides that show significant exposed bed
```

```
words = word_tokenize(text)
print(words)
```

```
['A', 'mountain', 'is', 'an', 'elevated', 'portion', 'of', 'the', 'Earth', "'s", 'crust', ',', 'generally', 'with', 'steep', 's
ides', 'that', 'show', 'significant', 'exposed', 'bedrock', '.', 'Although', 'definitions', 'vary', ',', 'a', 'mountain', 'ma
y', 'differ', 'from', 'a', 'plateau', 'in', 'having', 'a', 'limited', 'summit', 'area', ',', 'and', 'is', 'usually', 'higher',
'than', 'a', 'hill', ',', 'typically', 'rising', 'at', 'least', '300', 'metres', '(', '980', 'ft', ')', 'above', 'the', 'surrou
nding', 'land', '.']
```

```
lemm = [lemma.lemmatize(i) for i in words]
print(lemm)
```

```
['A', 'mountain', 'is', 'an', 'elevated', 'portion', 'of', 'the', 'Earth', "'s", 'crust', ',', 'generally', 'with', 'steep', 's
ide', 'that', 'show', 'significant', 'exposed', 'bedrock', '.', 'Although', 'definition', 'vary', ',', 'a', 'mountain', 'may',
'differ', 'from', 'a', 'plateau', 'in', 'having', 'a', 'limited', 'summit', 'area', ',', 'and', 'is', 'usually', 'higher', 'tha
n', 'a', 'hill', ',', 'typically', 'rising', 'at', 'least', '300', 'metre', '(', '980', 'ft', ')', 'above', 'the', 'surroundin
g', 'land', '.']
```

RESULT :

Thus program is completed successfully

EX.NO : 02	**Fundamentals of NLP II Stemming & Sentence Segmentation**
DATE : 18/07/2023	

AIM :

To implement process of stemming and sentence segmentation

THEORY:

- **Stemming:**

  Stemming is a technique used to reduce an inflected word down to its word stem. For example, the words “programming,” “programmer,” and “programs” can all be reduced down to the common word stem “program.” In other words, “program” can be used as a synonym for the prior three inflection words.

  Performing this **text-processing** technique is often useful for dealing with sparsity and/or standardizing vocabulary. Not only does it help with reducing redundancy, as most of the time the word stem and their inflected words have the same meaning, it also allows NLP models to learn links between inflected words and their word stem, which helps the model understand their usage in similar contexts.

- **Sentence Segmentation:**

  The process of deciding from where the sentences actually start or end in NLP or we can simply say that here we are dividing a paragraph based on sentences. This process is known as Sentence Segmentation. In Python, we implement this part of NLP using the spacy library.

PROCEDURE :

1. Stemming.
2. Sentence Segmentation.

DIFFERENT METHODS OF STEMMING :

- **Porter Stemming:**

```
# Stemming (Convert words to their base form)
from nltk.stem import PorterStemmer
porter = PorterStemmer()
for i in clean_words:
    print(porter.stem(i), end = ", ")


```

```
natur, languag, process, basic, nlp, articl, explor, code, exampl, dive, toolkit, nltk, librari, present, use, afterward, discu
ss, librari, essenti, method, along, respect, code, sampl, implement, python,
```

- ## SnowBall Stemmer :

```
# Snowball Stemmer is same as porter Stemmer
from nltk.stem import SnowballStemmer
stem = SnowballStemmer('english')
```

```
for w in list:
    print(stem.stem(w))
```

```
list
list
studi
feet
```

- ## Regexp Stemmer :

```
# RegexpStemmer we need to provid Suffix at the time inialization
list = ["listing","listed","studying","feet","Studied"]
from nltk.stem import RegexpStemmer
regex = RegexpStemmer("ing$|ed$|")
for i in list:
    print(regex.stem(i))
```

```
list
list
study
feet
Studi
```

- ## Lancaster Stemmer :

```
# Lancaster Stemmer
from nltk.stem import LancasterStemmer
lan = LancasterStemmer()
for i in list:
    print(lan.stem(i))
```

```
list
list
study
feet
study
```

IMAGE SEGMENTATION :

```
import spacy

# Load core englisg library
nlp = spacy.load("en_core_web_sm")

# Take unicode string
doc = nlp("In this article, we explore the basics of natural language processing (NLP) with code examples. We dive into the 

# Original string
text_file = open("C:\\Users\\mayur\\Downloads\\NLP.txt")
text = text_file.read()
print("Original String")
print(text)
print()

# To print sentences
print("Image Segmentation")
for sentence in doc.sents:
    print(sentence)
```

```
Original String
In this article, we explore the basics of natural language processing (NLP) with code examples. We dive into the natural langua
ge toolkit (NLTK) library to present how it can be useful for natural language processing related-tasks. Afterward, we will dis
cuss the basics of other Natural Language Processing libraries and other essential methods for NLP, along with their respective
coding sample implementations in Python.

Image Segmentation
In this article, we explore the basics of natural language processing (NLP) with code examples.
We dive into the natural language toolkit (NLTK) library to present how it can be useful for natural language processing relate
d-tasks.
Afterward, we will discuss the basics of other Natural Language Processing libraries and other essential methods for NLP, along
with their respective coding sample implementations in Python.
```

RESULT:

Thus Program is completed successfully

EX.NO : 03	NLP Using Scikit Library
DATE : 25/07/2023	

AIM :

To create a program for NLP using Scikit library

THEORY :

Natural language processing (NLP) refers to the branch of computer science— and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

PROCEDURE :

1. Importing Libraries.
2. Loading Training and Testing Texts.
3. Preprocessing.
4. Model Building.
5. Classifying Report.

PROGRAM:

1. **Import libraries :**

```
# Importing Necessary libraries
import nltk
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

2. Loading Dataset :

```
# Load Data

train = pd.read_csv("/kaggle/input/nlp-ex03/NLP EX 03.csv")
train.head()
```

	id	keyword	location	text	target
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1

```
train.tail()
```

	id	keyword	location	text	target
7608	10869	NaN	NaN	Two giant cranes holding a bridge collapse int...	1
7609	10870	NaN	NaN	@aria_ahrary @TheTawniest The out of control w...	1
7610	10871	NaN	NaN	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	1
7611	10872	NaN	NaN	Police investigating after an e-bike collided ...	1
7612	10873	NaN	NaN	The Latest: More Homes Razed by Northern Calif...	1

3. Preprocessing :

```
# Check null values
train.isnull().sum()
```

```
id             0
keyword       61
location    2533
text           0
target         0
dtype: int64
```

```
# Information
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        7613 non-null   int64 
 1   keyword   7552 non-null   object
 2   location  5080 non-null   object
 3   text      7613 non-null   object
 4   target    7613 non-null   int64 
dtypes: int64(2), object(3)
memory usage: 297.5+ KB
```

Dropping columns

```
# Drop columns
train = train.drop(['id','keyword','location'], axis =1 )
```

```
train
```

	text	target
0	Our Deeds are the Reason of this #earthquake M...	1
1	Forest fire near La Ronge Sask. Canada	1
2	All residents asked to 'shelter in place' are ...	1
3	13,000 people receive #wildfires evacuation or...	1
4	Just got sent this photo from Ruby #Alaska as ...	1
...	...	...
7608	Two giant cranes holding a bridge collapse int...	1
7609	@aria_ahrary @TheTawniest The out of control w...	1
7610	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	1
7611	Police investigating after an e-bike collided ...	1

Creating function to preprocess data

```
#create a function for cleaning the text data to remove special characters,
#lowering the case for text and performing tokenizer

def preprocessing(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]','',text)
    words = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]
    stemmer = PorterStemmer()
    words = [stemmer.stem(word) for word in words]
    return ' '.join(words)
```

```
train['cleaned_text'] = train['text'].apply(preprocessing)
train.head()
```

	text	target	cleaned_text
0	Our Deeds are the Reason of this #earthquake M...	1	deed reason earthquak may allah forgiv us
1	Forest fire near La Ronge Sask. Canada	1	forest fire near la rong sask canada
2	All residents asked to 'shelter in place' are ...	1	resid ask shelter place notifi offic evacu she...
3	13,000 people receive #wildfires evacuation or...	1	peopl receiv wildfir evacu order california
4	Just got sent this photo from Ruby #Alaska as ...	1	got sent photo rubi alaska smoke wildfir pour ...

4. Model Building :

CountVectorizer :

```
# Vectorizing
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(train['cleaned_text'])
X_test = vectorizer.fit_transform(test_data['cleaned_data'])
Y = train['target']
```

Splitting Data :

```
x_train, x_valid, y_train, y_valid = train_test_split(X,Y, test_size = 0.25,random_state = 42)
```

Loading Random Forest algorithm :

```
model = RandomForestClassifier()
model.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

Prediction :

```
y_pred = model.predict(x_valid)
```

5. Accuracy Score :

```
accuracy = accuracy_score(y_valid, y_pred)
print(f"Validation Accuracy: {accuracy*100:.2f}%")
```

```
Validation Accuracy: 79.41%
```

accuracy score for SVM :

```
model_svm = SVC()
model_svm.fit(x_train, y_train)
```

```
▾ SVC
SVC()
```

```
y_pred = model_svm.predict(x_valid)
```

```
accuracy_svm = accuracy_score(y_valid, y_pred)
print(f"Validation Accuracy: {accuracy_svm*100:.2f}%")
```

```
Validation Accuracy: 81.14%
```

RESULT :

Program is completed Sucessfully

EX.NO : 04	NLP using Spacy library
DATE : 01/08/2023	

AIM :

To create program for spacy library

THEORY :

Spacy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython. The library is published under the MIT license and its main developers are Matthew Honnibal and Ines Montani, the founders of the software company Explosion.

PROCEDURE :

1. Sentence Detection.
2. Stop Words.
3. Word Frequency.
4. Part-of-speech Tagging.
5. Dependency Parsing

PROGRAM :

1. **Importing Libraries and Data :**

```
path = "C:/Users/mayur/Downloads/NLP ex04.txt"
with open(path,'r') as f:
    text = f.read()
print(text)
```

```
There is no universally accepted definition of a mountain. Elevation, volume, relief, steepness, spacing and continuity have be
en used as criteria for defining a mountain.[4] In the Oxford English Dictionary a mountain is defined as "a natural elevation
of the earth surface rising more or less abruptly from the surrounding level and attaining an altitude which, relatively to the
adjacent elevation, is impressive or notable."[4]

Whether a landform is called a mountain may depend on local usage. John Whittow's Dictionary of Physical Geography[5] states "S
ome authorities regard eminences above 600 metres (1,969 ft) as mountains, those below being referred to as hills."

In the United Kingdom and the Republic of Ireland, a mountain is usually defined as any summit at least 2,000 feet (610 m) hig
h,[6] which accords with the official UK government's definition that a mountain, for the purposes of access, is a summit of 2,
000 feet (610 m) or higher.[7] In addition, some definitions also include a topographical prominence requirement, such as that
the mountain rises 300 metres (984 ft) above the surrounding terrain.[1] At one time the US Board on Geographic Names defined a
mountain as being 1,000 feet (305 m) or taller,[8] but has abandoned the definition since the 1970s. Any similar landform lower
than this height was considered a hill. However, today, the US Geological Survey concludes that these terms do not have technic
al definitions in the US.
```

2. Sentence Detection :

```
doc = nlp(text)
sent = list(doc.sents)
print(f"Length of document is {len(sent)}")
print()
print("***********Sentences***********")
sent = [sentence for sentence in sent]
print(sent)
```

```
Length of document is 10

***********Sentences***********
[There is no universally accepted definition of a mountain., Elevation, volume, relief, steepness, spacing and continuity have been used as criteria for defining a mountain.[4], In the Oxford English Dictionary a mountain is defined as "a natural elevation of the earth surface rising more or less abruptly from the surrounding level and attaining an altitude which, relatively to the adjacent elevation, is impressive or notable., "[4]

Whether a landform is called a mountain may depend on local usage., John Whittow's Dictionary of Physical Geography[5] states "Some authorities regard eminences above 600 metres (1,969 ft) as mountains, those below being referred to as hills., "

In the United Kingdom and the Republic of Ireland, a mountain is usually defined as any summit at least 2,000 feet (610 m) high,[6] which accords with the official UK government's definition that a mountain, for the purposes of access, is a summit of 2,000 feet (610 m) or higher.[7], In addition, some definitions also include a topographical prominence requirement, such as that the mountain rises 300 metres (984 ft) above the surrounding terrain.[1], At one time the US Board on Geographic Names defined a mountain as being 1,000 feet (305 m) or taller,[8] but has abandoned the definition since the 1970s., Any similar landform lower than this height was considered a hill., However, today, the US Geological Survey concludes that these terms do not have technical definitions in the US.]
```

3. Stop Words :

```
# Remove stopwords
from nltk.corpus import stopwords
from nltk import word_tokenize

words = word_tokenize(text)
print(f"Length of words : {len(words)}")

stopwords = stopwords.words("english")
words = [word for word in words if word not in stopwords]
print(f"Length of words without stopwords :{len(words)}")
print()
print("*****WORDS WITHOUT STOPWORDS")
print(words)

```

```
Length of words : 296
Length of words without stopwords :202

*****WORDS WITHOUT STOPWORDS
['There', 'universally', 'accepted', 'definition', 'mountain', '.', 'Elevation', ',', 'volume', ',', 'relief', ',', 'steepness', ',', 'spacing', 'continuity', 'used', 'criteria', 'defining', 'mountain', '.', '[', '4', ']', 'In', 'Oxford', 'English', 'Dictionary', 'mountain', 'defined', '``', 'natural', 'elevation', 'earth', 'surface', 'rising', 'less', 'abruptly', 'surrounding', 'level', 'attaining', 'altitude', ',', 'relatively', 'adjacent', 'elevation', ',', 'impressive', 'notable', '.', '``', '[', '4', ']', 'Whether', 'landform', 'called', 'mountain', 'may', 'depend', 'local', 'usage', '.', 'John', 'Whittow', "'s", 'Dictionary', 'Physical', 'Geography', '[', '5', ']', 'states', '``', 'Some', 'authorities', 'regard', 'eminences', '600', 'metres', '(', '1,969', 'ft', ')', 'mountains', ',', 'referred', 'hills', '.', "''", 'In', 'United', 'Kingdom', 'Republic', 'Ireland', ',', 'mountain', 'usually', 'defined', 'summit', 'least', '2,000', 'feet', '(', '610', ')', 'high', ',', '[', '6', ']', 'accords', 'official', 'UK', 'government', "'s", 'definition', 'mountain', ',', 'purposes', 'access', ',', 'summit', '2,000', 'feet', '(', '610', ')', 'higher', '.', '[', '7', ']', 'In', 'addition', ',', 'definitions', 'also', 'include', 'topographical', 'prominence', 'requirement', ',', 'mountain', 'rises', '300', 'metres', '(', '984', 'ft', ')', 'surrounding', 'terrain', '.', '[', '1', ']', 'At', 'one', 'time', 'US', 'Board', 'Geographic', 'Names', 'defined', 'mountain', '1,000', 'feet', '(', '305', ')', 'taller', ',', '[', '8', ']', 'abandoned', 'definition', 'since', '1970s', '.', 'Any', 'similar', 'landform', 'lower', 'height', 'considered', 'hill', '.', 'However', ',', 'today', ',', 'US', 'Geological', 'Survey', 'concludes', 'terms', 'technical', 'definitions', 'US', '.']
```

4. Removing Punctuations :

```python
# Remove punctuations
word_without_punc = []
for word in words:
    if word.isalpha():
        word_without_punc.append(word)

print(f"Number of words without punctuations : {len(word_without_punc)}")
print()
print("******WORDS WITHOUT PUNCTUATIONS********")
print(word_without_punc)
```

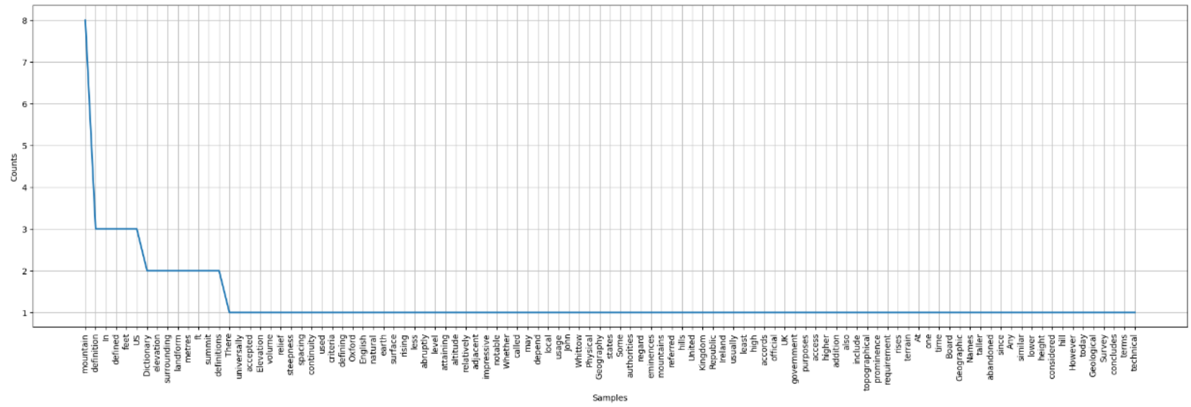
```
Number of words without punctuations : 128

******WORDS WITHOUT PUNCTUATIONS********
['There', 'universally', 'accepted', 'definition', 'mountain', 'Elevation', 'volume', 'relief', 'steepness', 'spacing', 'contin
uity', 'used', 'criteria', 'defining', 'mountain', 'In', 'Oxford', 'English', 'Dictionary', 'mountain', 'defined', 'natural',
'elevation', 'earth', 'surface', 'rising', 'less', 'abruptly', 'surrounding', 'level', 'attaining', 'altitude', 'relatively',
'adjacent', 'elevation', 'impressive', 'notable', 'Whether', 'landform', 'called', 'mountain', 'may', 'depend', 'local', 'usag
e', 'John', 'Whittow', 'Dictionary', 'Physical', 'Geography', 'states', 'Some', 'authorities', 'regard', 'eminences', 'metres',
'ft', 'mountains', 'referred', 'hills', 'In', 'United', 'Kingdom', 'Republic', 'Ireland', 'mountain', 'usually', 'defined', 'su
mmit', 'least', 'feet', 'high', 'accords', 'official', 'UK', 'government', 'definition', 'mountain', 'purposes', 'access', 'sum
mit', 'feet', 'higher', 'In', 'addition', 'definitions', 'also', 'include', 'topographical', 'prominence', 'requirement', 'moun
tain', 'rises', 'metres', 'ft', 'surrounding', 'terrain', 'At', 'one', 'time', 'US', 'Board', 'Geographic', 'Names', 'defined',
'mountain', 'feet', 'taller', 'abandoned', 'definition', 'since', 'Any', 'similar', 'landform', 'lower', 'height', 'considere
d', 'hill', 'However', 'today', 'US', 'Geological', 'Survey', 'concludes', 'terms', 'technical', 'definitions', 'US']
```

5. Frequency Distribution :

```python
from nltk.probability import FreqDist
frequency = FreqDist(word_without_punc)
print("*********10 MOST COMMON WORDS IN TEXT**********")
print(frequency.most_common(10))
```

```
*********10 MOST COMMON WORDS IN TEXT**********
[('mountain', 8), ('definition', 3), ('In', 3), ('defined', 3), ('feet', 3), ('US', 3), ('Dictionary', 2), ('elevation', 2),
('surrounding', 2), ('landform', 2)]
```

```python
import matplotlib.pyplot as plt
plt.figure(figsize=(25,7))
frequency.plot()
plt.show()
```



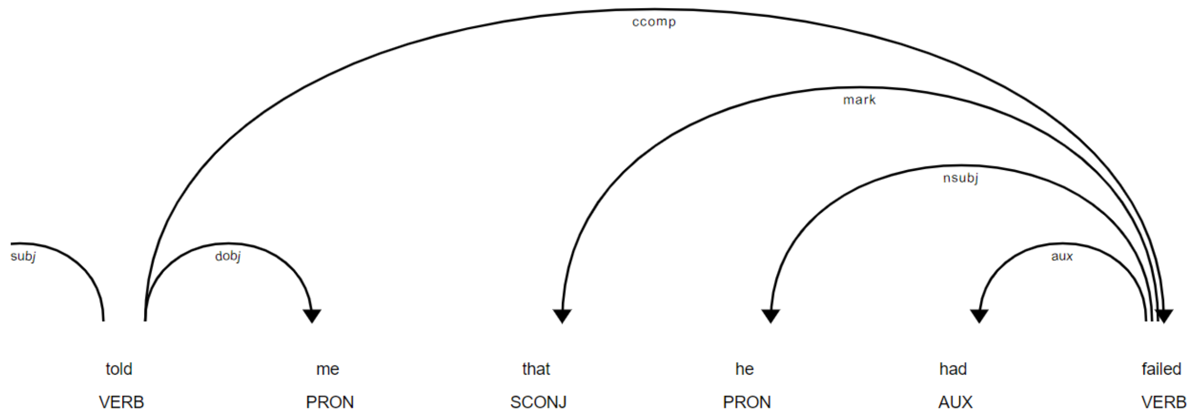
6. Part Of Speech Tagging :

```python
for word in doc:
    print(
        f"""
        TOKEN: {str(word)}
        =====
        TAG: {str(word.tag_):10} POS: {word.pos_}
        EXPLANATION: {spacy.explain(word.tag_)}"""
    )
```

```
TOKEN: There
=====
TAG: EX         POS: PRON
EXPLANATION: existential there

TOKEN: is
=====
TAG: VBZ        POS: VERB
EXPLANATION: verb, 3rd person singular present

TOKEN: no
=====
TAG: DT         POS: DET
EXPLANATION: determiner

TOKEN: universally
=====
TAG: RB         POS: ADV
EXPLANATION: adverb

TOKEN: accepted
=====
TAG: VBN        POS: VERB
EXPLANATION: verb, past participle

TOKEN: definition
=====
TAG: NN         POS: NOUN
EXPLANATION: noun, singular or mass
```

7. Dependency Parsing :

```python
from spacy import displacy
doc = nlp(u"No one told me that he had failed")
displacy.render(doc, style='dep',jupyter = True)
```



Result :

Thus programs are completed sucessfully

EX.NO : 05	Working With TF-IDF
DATE : 08/08/2023	

AIM :

To program for working with TF-IDF

THEORY :

Term Frequency - Inverse Document Frequency (TF-IDF) is a widely used statistical method in natural language processing and information retrieval. It measures how important a term is within a document relative to a collection of documents (i.e., relative to a corpus). Words within a text document are transformed into importance numbers by a text vectorization process. There are many different text vectorization scoring schemes, with TF-IDF being one of the most common.

PROCEDURE :

1. Importing Libraries.
2. Loading Training and Testing Texts.
3. Preprocessing.
4. Visualization
5. TF-IDF.
6. Model building

PROGRAM:

1. **Importing libraries :**

```
import seaborn as sns
import matplotlib.pyplot as plt
import os
import re
import spacy
from nltk.corpus import stopwords
from spacy import displacy
import nltk
from wordcloud import WordCloud
from nltk import word_tokenize
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
```

2. Loading Data :

```
data = pd.read_csv("/kaggle/input/tweet-analysis/sentiment_tweets3.csv")
data.drop('Index',axis = 1,inplace = True)
data.head()
```

	message to examine	label (depression result)
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat Need to send 'em to my accountant tomo...	0
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0

3. Preprocessing Data :

a) Changing columns name :

```
data.rename(columns={"message to examine": "review", "label (depression result)":"label"}, inplace=True)
data.head()
```

	review	label
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat Need to send 'em to my accountant tomo...	0
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0

b) Checking null values and dtypes :

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10314 entries, 0 to 10313
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   review  10314 non-null  object
 1   label   10314 non-null  int64
dtypes: int64(1), object(1)
memory usage: 161.3+ KB
```

```
data.isnull().sum()
```

```
review    0
label     0
dtype: int64
```

c) Removing non-characters from text :

```
data['review'] = data['review'].astype(str)
```

```
def preprocessing(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]','',text)
    text = re.sub(r'http\S+','',text)
    text = re.sub(r'\s+.com','',text)
    #words = word_tokenize(text)
    #stop_words = set(stopwords.words('english'))
    #words = [word for word in words if word not in stop_words]
    #stemmer = PorterStemmer()
    #words = [stemmer.stem(word) for word in words]
    return text
```

```
data['clean_review'] = data['review'].apply(preprocessing)
data.head()
```

	review	label	clean_review
0	just had a real good moment. i missssssssss hi...	0	just had a real good moment i missssssssss him...
1	is reading manga http://plurk.com/p/mzp1e	0	is reading manga
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0	comeagainjen
3	@lapcat Need to send 'em to my accountant tomo...	0	lapcat need to send em to my accountant tomorr...
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0	add me on myspace myspacecomlookthunder

d) Result after preprocessing :

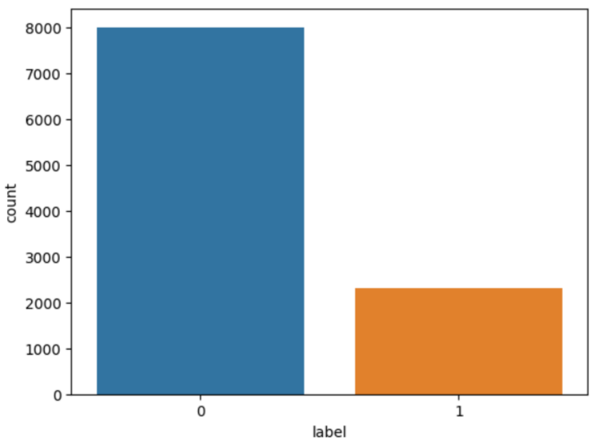
```
print("**Original Text**")
print(data['review'][1])
print("**Text after preprocessing**")
print(data['clean_review'][1])
```

```
**Original Text**
is reading manga  http://plurk.com/p/mzp1e
**Text after preprocessing**
is reading manga
```

4. Visualizing Data :

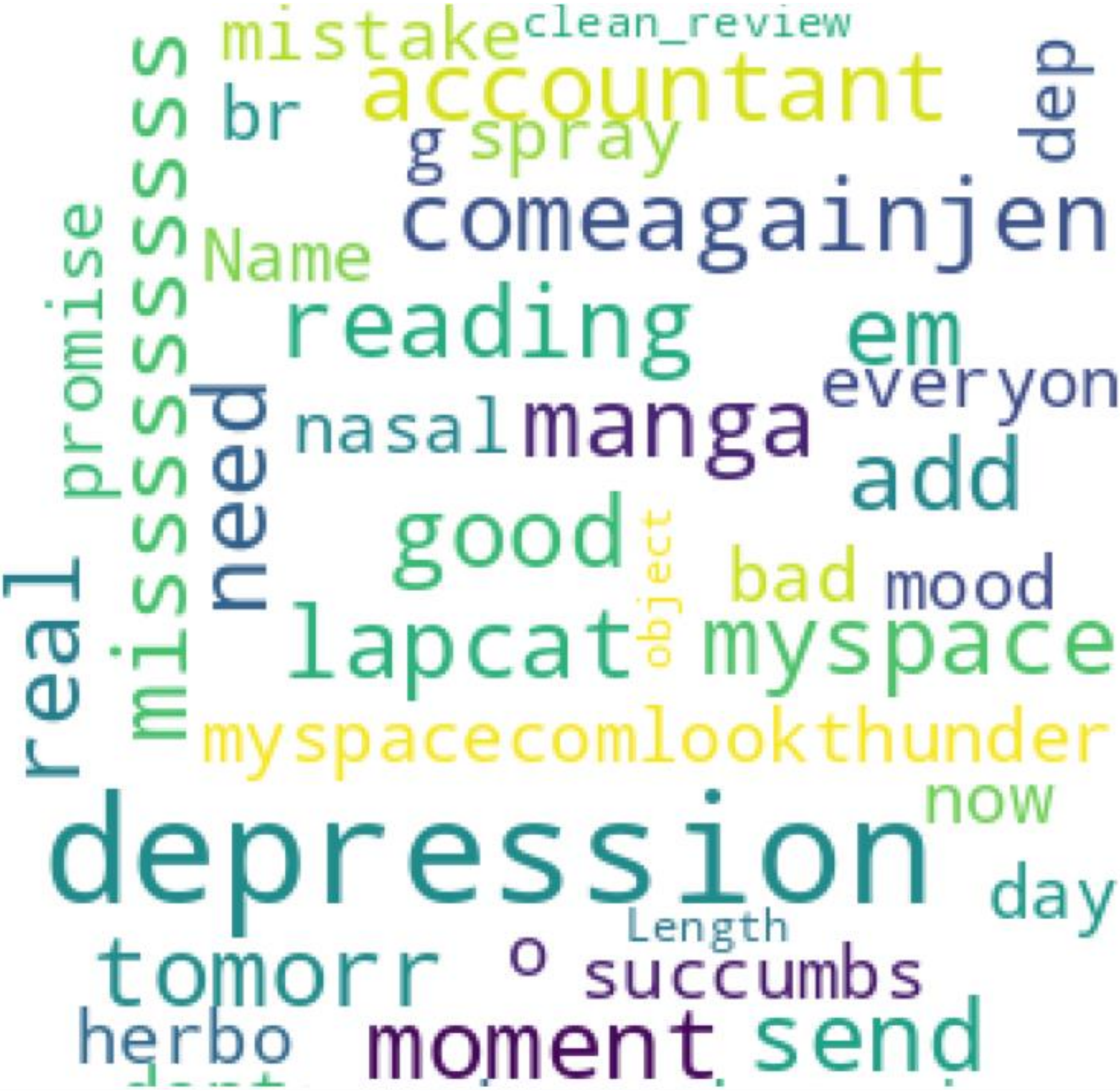
```
sns.countplot(data = data, x = 'label')
```

```
<Axes: xlabel='label', ylabel='count'>
```



WordCloud :

```
wordcloud = WordCloud(width = 300, height = 300,
                      background_color = 'white',
                      min_font_size = 10).generate(str(data['clean_review']))

plt.figure(figsize = (6,6), facecolor = None)
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



5. TF-IDF :

a) Text Corpus :

```
corpus = data['clean_review'][:5]
corpus
```

```
0    just had a real good moment i missssssssss him...
1                                   is reading manga
2                                     comeagainjen
3    lapcat need to send em to my accountant tomorr...
4             add me on myspace  myspacecomlookthunder
Name: clean_review, dtype: object
```

b) Apply TF-IDF on Text Corpus :

```
tfid = TfidfVectorizer()
corp = tfid.fit_transform(corpus)
```

c) Terms repeated :

```
print(tfid.vocabulary_)
```

```
{'just': 11, 'had': 8, 'real': 25, 'good': 7, 'moment': 16, 'missssssssss': 15, 'him': 9, 'so': 28, 'much': 17, 'is': 10, 'readi
ng': 24, 'manga': 13, 'comeagainjen': 3, 'lapcat': 12, 'need': 21, 'to': 33, 'send': 27, 'em': 4, 'my': 18, 'accountant': 0, 'to
morrow': 34, 'oddly': 22, 'wasnt': 35, 'even': 5, 'referring': 26, 'taxes': 30, 'those': 31, 'are': 2, 'supporting': 29, 'eviden
ce': 6, 'though': 32, 'add': 1, 'me': 14, 'on': 23, 'myspace': 19, 'myspacecomlookthunder': 20}
```

d) Score for each term :

```
print(corp.toarray())
```

```
[[0.         0.         0.         0.         0.         0.
  0.         0.33333333 0.33333333 0.33333333 0.         0.33333333
  0.         0.         0.         0.33333333 0.33333333 0.33333333
  0.         0.         0.         0.         0.         0.
  0.         0.33333333 0.         0.         0.33333333 0.
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.57735027 0.
  0.         0.57735027 0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.57735027 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         1.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.        ]
 [0.18569534 0.         0.18569534 0.         0.18569534 0.18569534
  0.18569534 0.         0.         0.         0.         0.
  0.18569534 0.         0.         0.         0.         0.
  0.37139068 0.         0.         0.18569534 0.18569534 0.
  0.         0.         0.18569534 0.18569534 0.         0.18569534
  0.18569534 0.18569534 0.18569534 0.55708601 0.18569534 0.18569534]
 [0.         0.4472136  0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.4472136  0.         0.         0.
  0.         0.4472136  0.4472136  0.         0.         0.4472136
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.        ]]
```

6. Model Building :

a) Splitting Data:

```
input_data = data["clean_review"]
output_data = data['label']

train_data, test_data, train_output, test_output = train_test_split(input_data, output_data, test_size = 0.
```

```
print(train_data.shape)
print(train_output.shape)
print(test_data.shape)
print(test_output.shape)
```

```
(8251,)
(8251,)
(2063,)
(2063,)
```

b) Loading Model :

```
from sklearn.ensemble import RandomForestClassifier
clf = Pipeline([('TfidfVectorizer',TfidfVectorizer()),('SVM',RandomForestClassifier())])
clf.fit(train_data, train_output)
prediction = clf.predict(test_data)
```

+ Code + Markdown

```
print(classification_report(test_output, prediction))
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      1614
           1       0.89      0.96      0.92       449

    accuracy                           0.97      2063
   macro avg       0.94      0.96      0.95      2063
weighted avg       0.97      0.97      0.97      2063
```

Result :

Thus programs are completed successfully with 98% accuracy

EX.NO : 06	Naïve Bayes Classifier
DATE : 29/08/2023	

AIM : To build Naïve Bayes classifier

THEORY :

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

where A and B are events and $P(B) \neq 0$.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

Now, with regards to our dataset, we can apply Bayes' theorem in following way:

PROCEDURE :

1) Import libraries
2) Load Text Data
3) Exploratory Analysis
4) Preprocess Data
5) Word Cloud
6) Build Model

PROGRAM :

1) **Import Libraries :**

```
import re
import seaborn as sns
import matplotlib.pyplot as plt
from nltk import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from wordcloud import WordCloud
from sklearn.metrics import classification_report , confusion_matrix, accuracy_score
```

2) Load Text Data :

```
data = pd.read_csv("/kaggle/input/corona-text-sentiment/Corona_NLP_train.csv",encoding='latin-1')
print(data.shape)
data.head()
```

```
(41157, 6)
```

	UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
0	3799	48751	London	16-03-2020	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	3800	48752	UK	16-03-2020	advice Talk to your neighbours family to excha...	Positive
2	3801	48753	Vagabonds	16-03-2020	Coronavirus Australia: Woolworths to give elde...	Positive
3	3802	48754	NaN	16-03-2020	My food stock is not the only one which is emp...	Positive
4	3803	48755	NaN	16-03-2020	Me, ready to go at supermarket during the #COV...	Extremely Negative

3) Explorartory Analysis :

a) Number of Categories :

```
data['Sentiment'].value_counts()
```

```
Sentiment
Positive              11422
Negative               9917
Neutral                7713
Extremely Positive     6624
Extremely Negative     5481
Name: count, dtype: int64
```

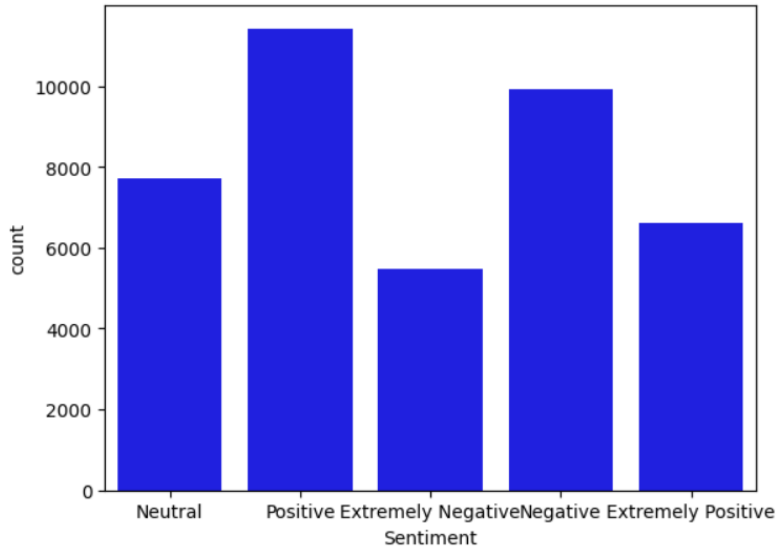
b) Data types :

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41157 entries, 0 to 41156
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   UserName       41157 non-null  int64
 1   ScreenName     41157 non-null  int64
 2   Location       32567 non-null  object
 3   TweetAt        41157 non-null  object
 4   OriginalTweet  41157 non-null  object
 5   Sentiment      41157 non-null  object
dtypes: int64(2), object(4)
memory usage: 1.9+ MB
```

b) Visualizing Categories :

```
sns.countplot(x = 'Sentiment', data = data,color = 'Blue')
plt.show()
```



4) Preprocessing :

a) Dropping Columns :

```
data = data.drop(['UserName','ScreenName','Location','TweetAt'],axis = 1)
data.head()
```

	OriginalTweet	Sentiment
0	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral
1	advice Talk to your neighbours family to excha...	Positive
2	Coronavirus Australia: Woolworths to give elde...	Positive
3	My food stock is not the only one which is emp...	Positive
4	Me, ready to go at supermarket during the #COV...	Extremely Negative

b) Replacing Category Names :

```
for label in data['Sentiment']:
    if label == 'Extremely Negative':
        data['Sentiment'] = data['Sentiment'].replace(label,'Negative')
    elif label == 'Extremely Positive':
        data['Sentiment'] = data['Sentiment'].replace(label, 'Positive')
```

c) Removing Unnecessary Words like Stop Words and Punctuations :

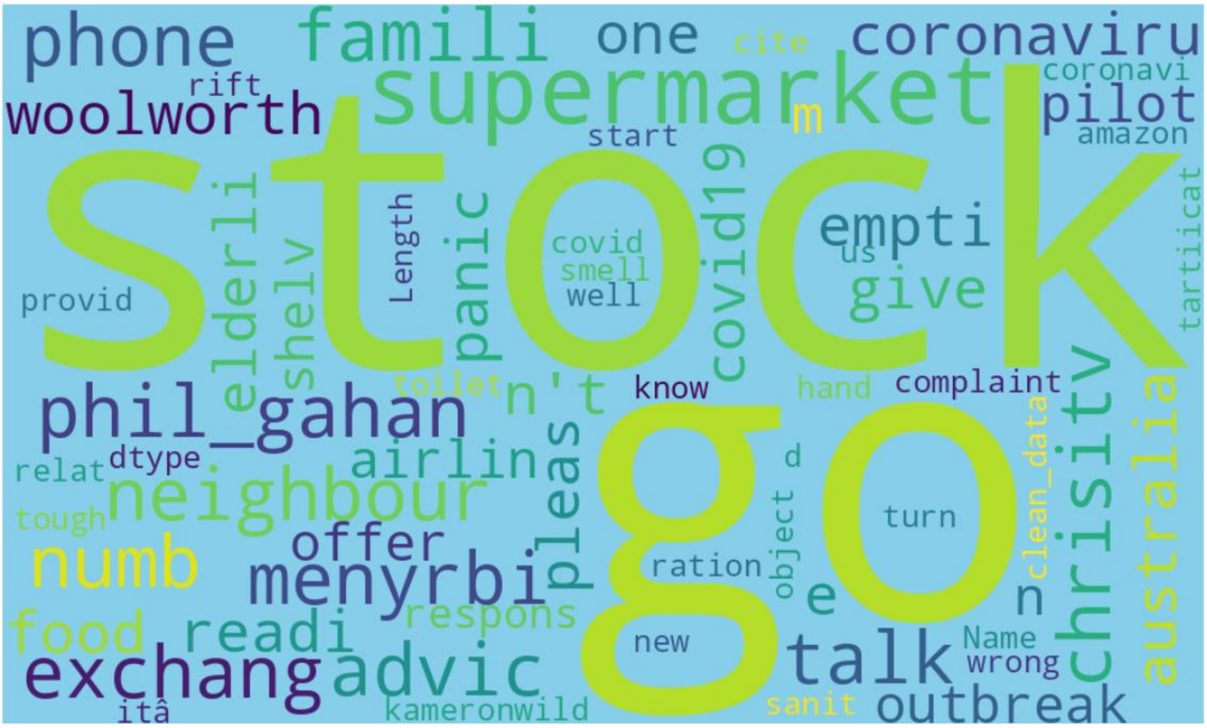
```
stop_word = stopwords.words('english')
stemmer = PorterStemmer()
def preprocess(text):
    text = text.lower()
    text = re.sub(r'http\S+',' ',text)
    text = re.sub(r'@','',text)
    #text = re.sub(r'[a-zA-Z]','',text)
    words = word_tokenize(text)
    words = [word for word in words if word not in stop_word]
    words = [stemmer.stem(word) for word in words]
    return " ".join(words)
```

```
data['clean_data'] = data['OriginalTweet'].apply(preprocess)
data.head()
```

	OriginalTweet	Sentiment	clean_data
0	@MeNyrbie @Phil_Gahan @Chrisitv https://t.co/i...	Neutral	menyrbi phil_gahan chrisitv
1	advice Talk to your neighbours family to excha...	Positive	advic talk neighbour famili exchang phone numb...
2	Coronavirus Australia: Woolworths to give elde...	Positive	coronaviru australia : woolworth give elderli ...
3	My food stock is not the only one which is emp...	Positive	food stock one empti ... pleas , n't panic , e...
4	Me, ready to go at supermarket during the #COV...	Negative	, readi go supermarket # covid19 outbreak . 'm...

6. Word Cloud :

```
wordcloud = WordCloud(
    width = 1000,
    height = 600,
    background_color = 'skyblue',
    min_font_size = 10).generate(str(data['clean_data']))
plt.figure(figsize = (10, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad = 0)
plt.show()
```



7. Model Building :

a) Label Encoding :

```
from sklearn.preprocessing import LabelEncoder
lb = LabelEncoder()
y = lb.fit_transform(data['Sentiment'])
print(y[:10])
```

```
[1 2 2 2 0 2 2 1 2 0]
```

b) Feature Extraction using TfidfVectorizer :

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfid = TfidfVectorizer()
x = tfid.fit_transform(data['clean_data']).toarray()
print(x[:5])
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

c) Splitting Data into train and test :

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, stratify = data['Sentiment'])
```

+ Code   + Markdown

```
print(len(x_train))
print(len(y_train))
print(len(x_test))
print(len(y_test))
```

```
32925
32925
8232
8232
```

d) Training Model :

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
```

```
alphas = [0.001,0.01,0.1,1,10,100,1000]
#accuracy = []
for alpha in alphas:
    model = MultinomialNB(alpha = alpha)
    model.fit(x_train, y_train)
    prediction = model.predict(x_test)
    accuracy = accuracy_score(y_test, prediction)

    print(f"Accuracy of MultiNomial model at alpha {alpha} : {accuracy:.2f}%")
```

```
Accuracy of MultiNomial model at alpha 0.001 : 0.61%
Accuracy of MultiNomial model at alpha 0.01 : 0.62%
Accuracy of MultiNomial model at alpha 0.1 : 0.65%
Accuracy of MultiNomial model at alpha 1 : 0.63%
Accuracy of MultiNomial model at alpha 10 : 0.57%
Accuracy of MultiNomial model at alpha 100 : 0.48%
Accuracy of MultiNomial model at alpha 1000 : 0.44%
```

RESULT :

Thus program is completed successfully with 64% accuracy

EX.NO : 07	Word Cloud Using Python
DATE : 05/09/2023	

AIM :

To program for working with TF-IDF

THEORY :

Word clouds (also known as text clouds or tag clouds) work in a simple way: the more a specific word appears in a source of textual data (such as a speech, blog post, or database), the bigger and bolder it appears in the word cloud.

A word cloud is a collection, or cluster, of words depicted in different sizes. The bigger and bolder the word appears, the more often it's mentioned within a given text and the more important it is.

Also known as tag clouds or text clouds, these are ideal ways to pull out the most pertinent parts of textual data, from blog posts to databases. They can also help business users compare and contrast two different pieces of text to find the wording similarities between the two.

PROCEDURE :

1. Importing Libraries.
2. Loading Training and Testing Texts.
3. Preprocessing.
4. Word Cloud

PROGRAM :

1. **Importing Libraries :**

```
import seaborn as sns
import matplotlib.pyplot as plt
import os
import re
import spacy
from nltk.corpus import stopwords
from spacy import displacy
import nltk
from wordcloud import WordCloud
from nltk import word_tokenize
from nltk.stem import PorterStemmer
```

2. Loading Data :

```
data = pd.read_csv("/kaggle/input/tweet-analysis/sentiment_tweets3.csv")
data.drop('Index',axis = 1,inplace = True)
data.head()
```

	message to examine	label (depression result)
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat Need to send 'em to my accountant tomo...	0
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0

3. Preprocessing Data :

a) Changing columns name :

```
data.rename(columns={"message to examine": "review", "label (depression result)":"label"}, inplace=True)
data.head()
```

	review	label
0	just had a real good moment. i missssssssss hi...	0
1	is reading manga http://plurk.com/p/mzp1e	0
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0
3	@lapcat Need to send 'em to my accountant tomo...	0
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0

b) Checking null values and dtypes :

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10314 entries, 0 to 10313
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   review  10314 non-null  object
 1   label   10314 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 161.3+ KB
```

```
data.isnull().sum()
```

```
review    0
label     0
dtype: int64
```

c) Removing non-characters from text :

```
data['review'] = data['review'].astype(str)
```

```
def preprocessing(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]','',text)
    text = re.sub(r'http\S+','',text)
    text = re.sub(r'\s+.com','',text)
    #words = word_tokenize(text)
    #stop_words = set(stopwords.words('english'))
    #words = [word for word in words if word not in stop_words]
    #stemmer = PorterStemmer()
    #words = [stemmer.stem(word) for word in words]
    return text
```

```
data['clean_review'] = data['review'].apply(preprocessing)
data.head()
```

	review	label	clean_review
0	just had a real good moment. i missssssssss hi...	0	just had a real good moment i missssssssss him...
1	is reading manga http://plurk.com/p/mzp1e	0	is reading manga
2	@comeagainjen http://twitpic.com/2y2lx - http:...	0	comeagainjen
3	@lapcat Need to send 'em to my accountant tomo...	0	lapcat need to send em to my accountant tomorr...
4	ADD ME ON MYSPACE!!! myspace.com/LookThunder	0	add me on myspace myspacecomlookthunder

d) Result after preprocessing :

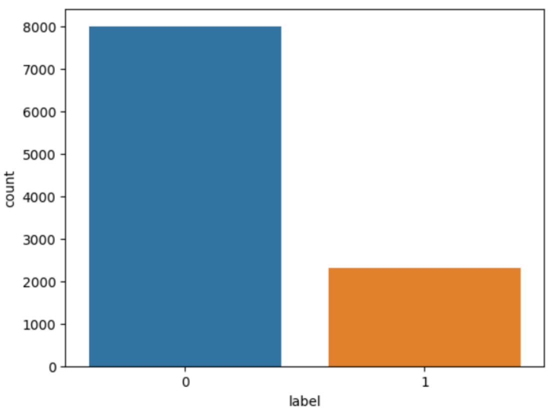
```
print("**Original Text**")
print(data['review'][1])
print("**Text after preprocessing**")
print(data['clean_review'][1])
```

```
**Original Text**
is reading manga  http://plurk.com/p/mzp1e
**Text after preprocessing**
is reading manga
```

4. Visualizing Data :

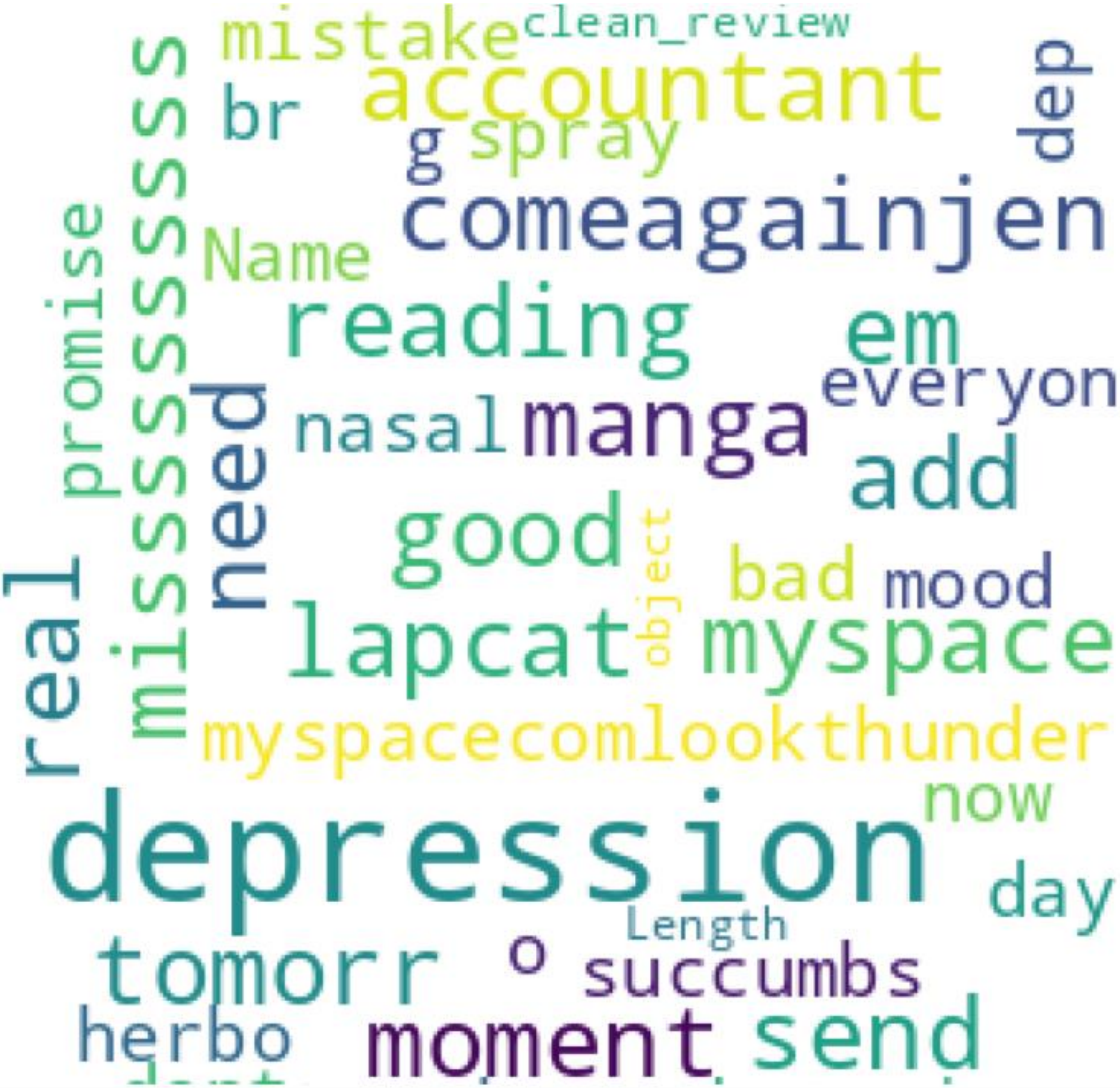
```
sns.countplot(data = data, x = 'label')
```

```
<Axes: xlabel='label', ylabel='count'>
```



Word Cloud :

```
wordcloud = WordCloud(width = 300, height = 300,
                      background_color = 'white',
                      min_font_size = 10).generate(str(data['clean_review']))

plt.figure(figsize = (6,6), facecolor = None)
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



RESULT :

Thus program is completed sucessfully

EX.NO : 08	Python Keyword Extraction
DATE : 11/09/2023	

AIM :

To implement the process of python keyword extraction

THEORY :

Keyphrase or keyword extraction in NLP is a text analysis technique that extracts important words and phrases from the input text. These key phrases can be used in a variety of tasks, including information retrieval, document summarization, and content categorization. This task is performed in two stages:

1. **Candidate Generation:** This process involves the identification of all possible keywords from the input text.
2. **Keyphrase Ranking:** After the candidate keywords are generated, they are ranked in order of importance for the identification of the best keywords.

Some of the popular key phrase generating tools and algorithms are **RAKE**, **YAKE, spaCy, Textacy**.

PROCEDURE :

1. Importing Libraries.
2. Loading Data
3. Rake
4. Yake
5. Spacy
6. Textacy

PROGRAM :

1. **Import libraries :**

```
from rake_nltk import Rake
import yake
import spacy
from textacy import *
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

2. Loading Text Corpus :

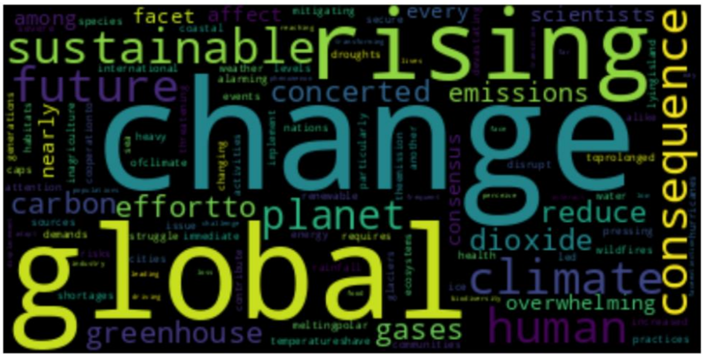
```
data = "Climate change, a pressing global issue, is transforming the way we perceive and interact with \
our planet. The overwhelming consensus among scientists is that human activities, particularly the\
emission of greenhouse gases like carbon dioxide, are driving this phenomenon. The consequences of\
climate change are far-reaching and affect nearly every facet of our lives. Rising global temperatures\
have led to more frequent and severe weather events, from devastating hurricanes and wildfires to\
prolonged droughts and heavy rainfall. These changes disrupt ecosystems and human communities alike,\
leading to food and water shortages, displacement of populations, and increased health risks. The melting\
polar ice caps and glaciers contribute to rising sea levels, threatening coastal cities and low-lying\
island nations. Biodiversity loss is another alarming consequence, as species struggle to adapt or face\
extinction in the face of changing habitats. Mitigating climate change requires a concerted global effort\
to reduce emissions, transition to renewable energy sources, and implement sustainable practices in\
agriculture and industry. It's a challenge that demands immediate attention and international cooperation\
to secure a sustainable future for our planet and future generations."
```

3. Rake :

```
rake = Rake()
rake.extract_keywords_from_text(data)
keywords = rake.get_ranked_phrases()
print(keywords)
```

```
['greenhouse gases like carbon dioxide', 'concerted global effortto reduce emissions', 'overwhelming consensus among scientists', 'affect nearly every facet', 'rising global temperatures
have led', 'implement sustainable practices inagriculture', 'mitigating climate change requires', 'pressing global issue', 'rising sea levels', 'wildfires toprolonged droughts', 'threate
ning coastal cities', 'severe weather events', 'renewable energy sources', 'meltingpolar ice caps', 'international cooperationto secure', 'increased health risks', 'demands immediate att
ention', 'consequences ofclimate change', 'changes disrupt ecosystems', 'another alarming consequence', 'human communities alike', 'climate change', 'sustainable future', 'human activiti
es', 'water shortages', 'species struggle', 'particularly theemission', 'lyingisland nations', 'heavy rainfall', 'glaciers contribute', 'future generations', 'devastating hurricanes', 'c
hanging habitats', 'biodiversity loss', 'way', 'transition', 'transforming', 'reaching', 'populations', 'planet', 'planet', 'phenomenon', 'perceive', 'low', 'lives', 'leading', 'interac
t', 'industry', 'frequent', 'food', 'far', 'faceextinction', 'face', 'driving', 'displacement', 'challenge', 'adapt']
```

```
wordcloud = WordCloud().generate(' '.join(keywords))
plt.figure(figsize=(10,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



4. Yake :

```
yake = yake.KeywordExtractor()
keywords = yake.extract_keywords(data)
print(keywords)
```

```
[('pressing global issue', 0.006715833971354186), ('perceive and interact', 0.020053544814553306), ('global issue', 0.04520339888385852), ('pressing global', 0.0508303667988021), ('Clima
te change', 0.06520464039553504), ('consequences ofclimate change', 0.0877644559738012), ('change', 0.10112787946193774), ('Mitigating climate change', 0.10230100029077827), ('climate ch
ange requires', 0.11056399690372985), ('global', 0.11499667481325007), ('Rising global temperatureshave', 0.11904044953140082), ('issue', 0.1253613730740891), ('carbon dioxide', 0.136586
66423055238), ('driving this phenomenon', 0.13658666423055238), ('human activities', 0.13986963327702334), ('pressing', 0.1402116473045658), ('transforming', 0.1402116473045658), ('perce
ive', 0.1402116473045658), ('interact', 0.1402116473045658), ('overwhelming consensus', 0.1504077774631919)]
```

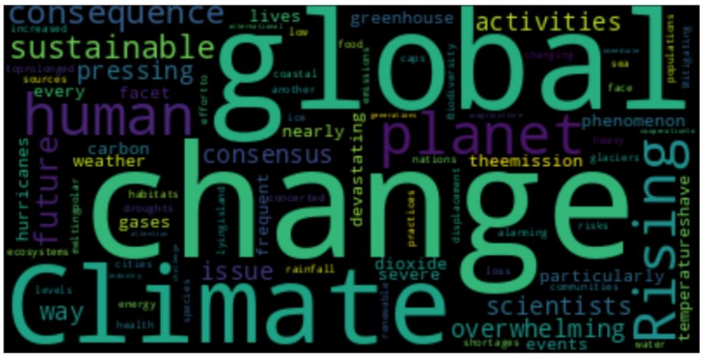
```
keywords = [kw for kw, _ in keywords]
wordcloud = WordCloud().generate(' '.join(keywords))
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

5. Spacy :

```
nlp = spacy.load('en_core_web_sm')
spacy_doc = nlp(data)
keywords = []

for chunk in spacy_doc.noun_chunks:
    if chunk.text.lower not in nlp.Defaults.stop_words:
        keywords.append(chunk.text)

print(keywords)
```

```
['Climate change', 'a pressing global issue', 'the way', 'we', 'our planet', 'The overwhelming consensus', 'scientists', 'human activities', 'particularly theemission', 'greenhouse gase
s', 'carbon dioxide', 'this phenomenon', 'The consequences', 'change', 'nearly every facet', 'our lives', 'Rising global temperatureshave', 'more frequent and severe weather events', 'de
vastating hurricanes', 'toprolonged droughts', 'heavy rainfall', 'These changes', 'ecosystems', 'human communities', 'food and water shortages', 'displacement', 'populations', 'increased
health risks', 'The meltingpolar ice caps', 'glaciers', 'rising sea levels', 'coastal cities', 'low-lyingisland nations', 'Biodiversity loss', 'another alarming consequence', 'species',
'the face', 'changing habitats', 'Mitigating climate change', 'a concerted global effortto', 'emissions', 'renewable energy sources', 'sustainable practices inagriculture', 'industry',
'It', 'a challenge', 'that', 'immediate attention', 'international cooperationto', 'a sustainable future', 'our planet', 'future generations']
```

```
wordcloud = WordCloud().generate(' '.join(keywords))
plt.figure(figsize=(10,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



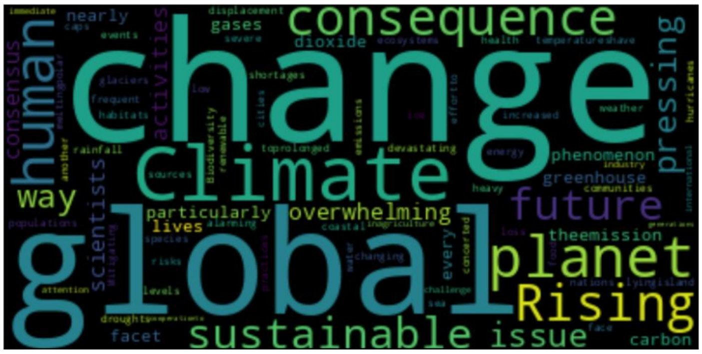
6. Textacy :

```
text = textacy.load_spacy_lang('en_core_web_sm')
doc = textacy.make_spacy_doc(data, lang=text)

print("Textrank output: \n", textacy.extract.keyterms.textrank(doc,
                                                              normalize="lemma",
                                                              topn=5))
```

```
Textrank output:
 [('climate change', 0.0256179868518401), ('concerted global effortto', 0.025430693022693154), ('sustainable practice inagriculture', 0.020916994707502835), ('global temperatureshave',
0.019821609033177177), ('global issue', 0.01965728316728757)]
```

```python
wordcloud = WordCloud().generate(' '.join(keywords))
plt.figure(figsize=(10,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Textrank')
plt.show()
```

Textrank

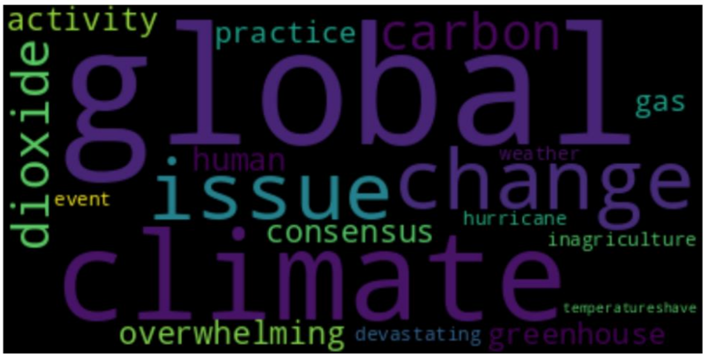



```python
print("SGRank output: \n", [kps for kps, weights in
                            textacy.extract.keyterms.sgrank(doc, topn=5)])
```

```
SGRank output:
 ['climate change', 'global issue', 'carbon dioxide', 'overwhelming consensus', 'human activity']
```

```python
keywords = [kps for kps, weights in
                            textacy.extract.keyterms.sgrank(doc, normalize="lemma")]

# Generate WordCloud
wordcloud = WordCloud().generate(' '.join(keywords))

# Display the WordCloud
plt.figure(figsize=(10,10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('SGRank')
plt.show()
```

SGRank



Result :

The program is completed with using four different keyword extraction techniques

EX.NO : 09	**Named Entity Recognition**
DATE : 29/09/2023	

AIM :

Named entity recognition in NLP

THEORY :

Named entity recognition (NER) is a natural language processing (NLP) method that extracts information from text. NER involves detecting and categorizing important information in text known as *named entities*. Named entities refer to the key subjects of a piece of text, such as names, locations, companies, events and products, as well as themes, topics, times, monetary values and percentages.

PROGRAM :

1. Import Libraries
2. Loading Text corpus
3. Name entity recognition techniques
   a) NLTK
   b) Spacy

Program :

1. **Import Libraries :**

```
import nltk
import spacy
from rake_nltk import Rake
import yake
from textacy import *
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

2. **Loading Text Corpus :**

\

```
parag = "The European Union, headquartered in Brussels, Belgium, is a prime example of successful international collaboration. Established in 1957,\
this political and economic union has evolved over the years, now comprising 27 member countries. Leaders from across the continent regularly convene in\
the European Parliament, where policies and legislation are discussed and enacted. Angela Merkel, the former Chancellor of Germany, played a significant\
role in shaping the EU's policies during her tenure. The European Central Bank, based in Frankfurt, is responsible for overseeing the monetary policies\
of the Eurozone, where the Euro is the common currency. The EU's open-border policy, known as the Schengen Agreement, allows for passport-free travel across\
many of its member states. Additionally, the European Space Agency (ESA), an intergovernmental organization, leads collaborative space exploration efforts.\
They've been involved in missions to Mars, such as the ExoMars program. This exemplifies the EU's commitment to scientific cooperation and international\
space exploration."
```

3. Named Entity Techniques :

a) NLTK :

```
for sent in nltk.sent_tokenize(parag):
    for chunk in nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(sent))):
        if hasattr(chunk, 'label'):
            print(chunk.label(), ' '.join(c[0] for c in chunk))
```

```
ORGANIZATION European Union
GPE Brussels
GPE Belgium
ORGANIZATION European Parliament
PERSON Angela
ORGANIZATION Merkel
GPE Germany
GPE EU
ORGANIZATION European Central Bank
GPE Frankfurt
GPE Eurozone
GPE Euro
GPE EU
ORGANIZATION Schengen Agreement
ORGANIZATION European Space Agency
ORGANIZATION ESA
PERSON Mars
ORGANIZATION ExoMars
GPE EU
```

b) Spacy :

```
nlp = spacy.load('en_core_web_sm')
ner = nlp(parag)

for ent in ner.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
The European Union 0 18 ORG
Brussels 37 45 GPE
Belgium 47 54 GPE
the years 189 198 DATE
27 215 217 CARDINAL
European Parliament 294 313 ORG
Angela Merkel 373 386 PERSON
Germany 413 420 GPE
EU 462 464 ORG
The European Central Bank 495 520 ORG
Frankfurt 531 540 GPE
Euro 620 624 WORK_OF_ART
EU 653 655 ORG
the Schengen Agreement 687 709 ORG
the European Space Agency 790 815 ORG
ESA 817 820 ORG
Mars 941 945 LOC
ExoMars 959 966 PRODUCT
EU 997 999 ORG
```

Result :

Program is successfully completed using two named entity techniques

EX.NO : 10	**Latent Semantic Analysis**
DATE : 03/10/2023	

AIM :

Perform latent semnatic analysis

THEORY :

Latent Semantic Analysis (LSA) involves creating structured data from a collection of unstructured texts. Before getting into the concept of LSA, let us have a quick intuitive understanding of the concept. When we write anything like text, the words are not chosen randomly from a vocabulary.Rather, we think about a theme (or topic) and then chose words such that we can express our thoughts to others in a more meaningful way. This theme or topic is usually considered as a latent dimension.

PROVEDURE :

1. Import libraries
2. Load Data
3. Preprocess Data
4. Transform Data
5. Latent Semantic Analysis (LSA)

PROGRAM :

1. Import libraries :

```
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer, LancasterStemmer
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from nltk import ne_chunk
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
```

2. Load Data :

```
data = pd.read_csv('/kaggle/input/abc-news/abcnews-date-text.csv')
data.head()
```

	publish_date	headline_text
0	20030219	aba decides against community broadcasting lic...
1	20030219	act fire witnesses must be aware of defamation
2	20030219	a g calls for infrastructure protection summit
3	20030219	air nz staff in aust strike for pay rise
4	20030219	air nz strike to affect australian travellers

```
data = data.drop(['publish_date'], axis = 1)
data.head()
```

	headline_text
0	aba decides against community broadcasting lic...
1	act fire witnesses must be aware of defamation
2	a g calls for infrastructure protection summit
3	air nz staff in aust strike for pay rise
4	air nz strike to affect australian travellers

3. Preprocess Data :

```
stop_words = stopwords.words('english')
stemmer = PorterStemmer()
def clean_text(headline):
    text = headline.lower()
    words = word_tokenize(text)
    words = [w for w in words if w not in stop_words]
    words = [stemmer.stem(w) for w in words]
    return " ".join(words)
```

```
data['headline_cleaned_text']=data['headline_text'].apply(clean_text)
```

```
data.head()
```

	headline_cleaned_text
0	aba decid commun broadcast licenc
1	act fire wit must awar defam
2	g call infrastructur protect summit
3	air nz staff aust strike pay rise
4	air nz strike affect australian travel

4. Transform Data :

```
vect =TfidfVectorizer(stop_words=stop_words,max_features=1000)
vect_text=vect.fit_transform(data['headline_cleaned_text'])
```

+ Code + Markdown

```
print(vect_text.shape)
print(vect_text)
```

```
(1244184, 1000)
  (0, 190)	1.0
  (1, 577)	0.5648845718512744
  (1, 983)	0.5613262186342711
  (1, 335)	0.388645233196061
  (1, 15)	0.4634362733668268
  (2, 675)	0.8031095011783936
  (2, 134)	0.5958314603283311
  (3, 739)	0.3358350220275444
  (3, 625)	0.35987383218253743
  (3, 843)	0.36981008504745183
  (3, 65)	0.39767024929073325
  (3, 827)	0.4033666675171786
  (3, 597)	0.3991782963072273
  (3, 32)	0.3751753670080188
  (4, 911)	0.44240540659438593
  (4, 67)	0.32533847523715337
  (4, 21)	0.45624646678594905
  (4, 843)	0.39176555206870584
  (4, 597)	0.42287734150511347
  (4, 32)	0.3974493685309207
  (5, 979)	1.0
  (6, 117)	0.7411081175072274
```

```
idf=vect.idf_
dd=dict(zip(vect.get_feature_names_out(), idf))
l=sorted(dd, key=(dd).get)
# print(l)
print(l[0],l[-1])
print(dd['polic'])
```

```
polic semi
4.437306523204708
```

5. Semantic Analysis :

```
from sklearn.decomposition import TruncatedSVD
lsa_model = TruncatedSVD(n_components=10, algorithm='randomized', n_iter=10, random_state=42)

lsa_top=lsa_model.fit_transform(vect_text)
```

```
print(lsa_top)
print(lsa_top.shape)
```

```
[[ 0.00041836  0.02038933  0.02714149 ... -0.00319518  0.00091593
   0.00071417]
 [ 0.0011945   0.0600878   0.04072359 ...  0.01923555 -0.16908529
  -0.19874375]
 [ 0.00109207  0.061968    0.08973196 ...  0.3746724   0.32827177
  -0.11221815]
 ...
 [ 0.00171047  0.11330495  0.24387278 ...  0.00573426  0.01656982
  -0.03600617]
 [ 0.00054127  0.04083816 -0.00405718 ...  0.00610421 -0.02532323
   0.01622376]
 [ 0.00153814  0.09662501  0.2112592  ...  0.01744683  0.00956658
  -0.01188148]]
(1244184, 10)
```

```
l=lsa_top[0]
print("Document 0 :")
for i,topic in enumerate(l):
    print("Topic ",i," : ",topic*100)
```

```
Document 0 :
Topic  0  :  0.04183634389399514
Topic  1  :  2.038933150631539
Topic  2  :  2.7141490471415266
Topic  3  :  -0.19243766923279004
Topic  4  :  -1.189662223979564
Topic  5  :  0.4251041812831996
Topic  6  :  -2.0980195238112525
Topic  7  :  -0.31951809521461105
Topic  8  :  0.09159347173828995
Topic  9  :  0.07141675294146262
```

```
vocab = vect.get_feature_names_out()

for i, comp in enumerate(lsa_model.components_):
    vocab_comp = zip(vocab, comp)
    sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
    print("Topic "+str(i)+": ")
    for t in sorted_words:
        print(t[0],end=" ")
    print("\n")
```

```
Topic 0:
interview extend michael nrl john david smith jame polic andrew

Topic 1:
polic man charg new say court murder death face crash

Topic 2:
new say plan council australia call win govt back us

Topic 3:
polic investig probe search hunt offic say warn miss seek

Topic 4:
new polic zealand year charg name case search law investig

Topic 5:
win australia call open back cup world fire australian us

Topic 6:
win say new australia open polic cup world australian man

Topic 7:
call australia us media day kill death australian fire warn

Topic 8:
call win charg say plan council court murder face new

Topic 9:
australia plan charg day court face polic back murder get
```

Result :

Latent semantic analysis completed sucessfully

EX.NO : 11	Determine optimum number of topics in a document
DATE : 10/10/2023	

AIM :

Program to perform latent semantic analysis

THEORY :

Latent Semantic Analysis (LSA) involves creating structured data from a collection of unstructured texts. Before getting into the concept of LSA, let us have a quick intuitive understanding of the concept. When we write anything like text, the words are not chosen randomly from a vocabulary.Rather, we think about a theme (or topic) and then chose words such that we can express our thoughts to others in a more meaningful way. This theme or topic is usually considered as a latent dimension.

PROCEDURE :

1. Import libraries
2. Load Data
3. Preprocess Data
4. Create Gensim Model
5. Plot Graph

PROGRAM :

1. **Import Libraries :**

```
import numpy as np
import pandas as pd
import os.path
from gensim import corpora
from gensim.models import LsiModel
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from gensim.models.coherencemodel import CoherenceModel
import matplotlib.pyplot as plt
```

2. Load Data :

```
def load_data(path, file_name):
    documents_list = []
    titles = []

    with open(os.path.join(path, file_name), 'r') as fin:
        for line in fin.readlines():
            text = line.strip()
            documents_list.append(text)
            titles.append(text)

    print("Total Number of Documents:", len(documents_list))

    return documents_list, titles
```

3. Preprocess Data :

```
def preprocess_data(doc_set):
    """
    Input  : docuemnt list
    Purpose: preprocess text (tokenize, removing stopwords, and stemming)
    Output : preprocessed text
    """
    # initialize regex tokenizer
    tokenizer = RegexpTokenizer(r'\w+')
    # create English stop words list
    en_stop = set(stopwords.words('english'))
    # Create p_stemmer of class PorterStemmer
    p_stemmer = PorterStemmer()
    # list for tokenized documents in loop
    texts = []
    # loop through document list
    for i in doc_set:
        # clean and tokenize document string
        raw = i.lower()
        tokens = tokenizer.tokenize(raw)
        # remove stop words from tokens
        stopped_tokens = [i for i in tokens if not i in en_stop]
        # stem tokens
        stemmed_tokens = [p_stemmer.stem(i) for i in stopped_tokens]
        # add tokens to list
        texts.append(stemmed_tokens)
    return texts
```

```
def prepare_corpus(doc_clean):
    """
    Input  : clean document
    Purpose: create term dictionary of our courpus and Converting list of documents (corpus) into Document
    Output : term dictionary and Document Term Matrix
    """
    # Creating the term dictionary of our courpus, where every unique term is assigned an index. dictionary
    dictionary = corpora.Dictionary(doc_clean)
    # Converting list of documents (corpus) into Document Term Matrix using dictionary prepared above.
    doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
    # generate LDA model
    return dictionary,doc_term_matrix
```

4. Create Model :

```
def create_gensim_lsa_model(doc_clean,number_of_topics,words):
    """
    Input  : clean document, number of topics and number of words associated with each topic
    Purpose: create LSA model using gensim
    Output : return LSA model
    """
    dictionary,doc_term_matrix=prepare_corpus(doc_clean)
    # generate LSA model
    lsamodel = LsiModel(doc_term_matrix, num_topics=number_of_topics, id2word = dictionary)  # train model
    print(lsamodel.print_topics(num_topics=number_of_topics, num_words=words))
    return lsamodel
```

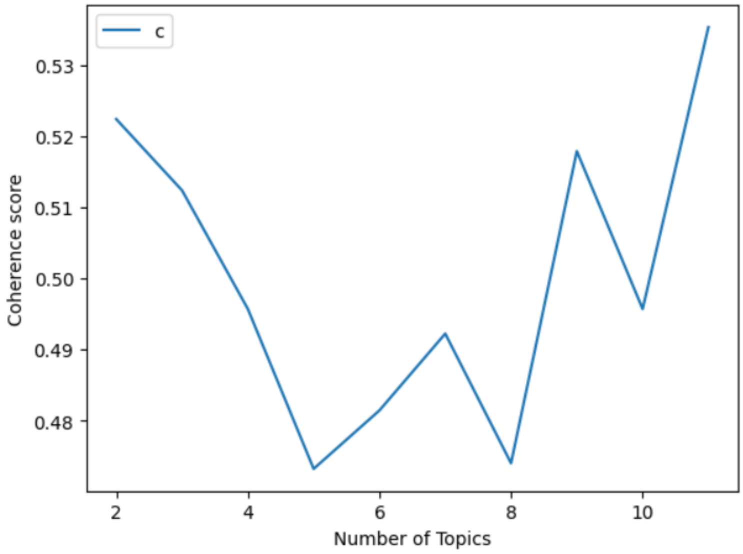
```
def compute_coherence_values(dictionary, doc_term_matrix, doc_clean, stop, start=2, step=3):
    """
    Input   : dictionary : Gensim dictionary
              corpus : Gensim corpus
              texts : List of input texts
              stop : Max num of topics
    purpose : Compute c_v coherence for various number of topics
    Output  : model_list : List of LSA topic models
              coherence_values : Coherence values corresponding to the LDA model with respective number of topics
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, stop, step):
        # generate LSA model
        model = LsiModel(doc_term_matrix, num_topics=number_of_topics, id2word = dictionary)  # train model
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=doc_clean, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())
    return model_list, coherence_values
```

```
number_of_topics=7
words=10
document_list,titles=load_data("/kaggle/input/raw-text-data/","Raw text.txt")
clean_text=preprocess_data(document_list)
model=create_gensim_lsa_model(clean_text,number_of_topics,words)
```

```
Total Number of Documents: 4551
[(0, '0.361*"trump" + 0.272*"say" + 0.233*"said" + 0.166*"would" + 0.160*"clinton" + 0.140*"peopl" + 0.136*"one" + 0.126*"campai
gn" + 0.123*"year" + 0.110*"time"'), (1, '0.389*"citi" + 0.370*"v" + 0.356*"h" + 0.355*"2016" + 0.354*"2017" + 0.164*"unit" + 0.
159*"west" + 0.157*"manchest" + 0.116*"apr" + 0.112*"dec"'), (2, '-0.612*"trump" + -0.264*"clinton" + 0.261*"eu" + 0.148*"say" +
0.137*"would" + -0.135*"donald" + 0.134*"leav" + 0.134*"uk" + -0.119*"republican" + 0.110*"cameron"'), (3, '-0.400*"min" + 0.261
*"eu" + -0.183*"goal" + -0.152*"ball" + -0.132*"play" + 0.128*"said" + 0.128*"say" + -0.126*"leagu" + 0.122*"leav" + -0.122*"gam
e"'), (4, '-0.404*"bank" + 0.305*"eu" + 0.290*"min" + -0.189*"year" + 0.164*"leav" + 0.153*"cameron" + -0.143*"market" + -0.140
*"rate" + 0.139*"vote" + 0.133*"say"'), (5, '0.310*"bank" + -0.307*"say" + -0.221*"peopl" + 0.203*"trump" + 0.166*"1" + 0.164*"m
in" + 0.163*"0" + 0.152*"market" + 0.152*"eu" + -0.138*"like"'), (6, '-0.570*"say" + -0.237*"min" + 0.170*"vote" + -0.158*"gover
n" + 0.154*"poll" + -0.122*"tax" + -0.115*"bank" + -0.115*"statement" + -0.112*"budget" + 0.108*"one"')]
```

5. Plot Graph :

```
def plot_graph(doc_clean,start, stop, step):
    dictionary,doc_term_matrix=prepare_corpus(doc_clean)
    model_list, coherence_values = compute_coherence_values(dictionary, doc_term_matrix,doc_clean,
                                                            stop, start, step)
    # Show graph
    x = range(start, stop, step)
    plt.plot(x, coherence_values)
    plt.xlabel("Number of Topics")
    plt.ylabel("Coherence score")
    plt.legend(("coherence_values"), loc='best')
    plt.show()

start,stop,step=2,12,1
plot_graph(clean_text,start,stop,step)
```



Result :

The optimal number of topics are generated sucessfully

EX.NO : 12	**Fundamental of topic modelling**
DATE : 17/10/2023	

AIM :

Perform latent semnatic analysis

THEORY :

Latent Semantic Analysis (LSA) involves creating structured data from a collection of unstructured texts. Before getting into the concept of LSA, let us have a quick intuitive understanding of the concept. When we write anything like text, the words are not chosen randomly from a vocabulary.Rather, we think about a theme (or topic) and then chose words such that we can express our thoughts to others in a more meaningful way. This theme or topic is usually considered as a latent dimension.

PROVEDURE :

1. Import libraries
2. Load Data
3. Preprocess Data
4. Transform Data
5. Latent Semantic Analysis (LSA)
6. Latent Dirichlet Allocation (LDA)

PROGRAM :

1. **Import libraries :**

```
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer, LancasterStemmer
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from nltk import ne_chunk
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
```

2. Load Data :

```
data = pd.read_csv('/kaggle/input/abc-news/abcnews-date-text.csv')
data.head()
```

	publish_date	headline_text
0	20030219	aba decides against community broadcasting lic...
1	20030219	act fire witnesses must be aware of defamation
2	20030219	a g calls for infrastructure protection summit
3	20030219	air nz staff in aust strike for pay rise
4	20030219	air nz strike to affect australian travellers

```
data = data.drop(['publish_date'], axis = 1)
data.head()
```

	headline_text
0	aba decides against community broadcasting lic...
1	act fire witnesses must be aware of defamation
2	a g calls for infrastructure protection summit
3	air nz staff in aust strike for pay rise
4	air nz strike to affect australian travellers

3. Preprocess Data :

```
stop_words = stopwords.words('english')
stemmer = PorterStemmer()
def clean_text(headline):
    text = headline.lower()
    words = word_tokenize(text)
    words = [w for w in words if w not in stop_words]
    words = [stemmer.stem(w) for w in words]
    return " ".join(words)
```

```
data['headline_cleaned_text']=data['headline_text'].apply(clean_text)
```

```
data.head()
```

	headline_cleaned_text
0	aba decid commun broadcast licenc
1	act fire wit must awar defam
2	g call infrastructur protect summit
3	air nz staff aust strike pay rise
4	air nz strike affect australian travel

4. Transform Data :

```
vect =TfidfVectorizer(stop_words=stop_words,max_features=1000)
vect_text=vect.fit_transform(data['headline_cleaned_text'])
```

+ Code + Markdown

```
print(vect_text.shape)
print(vect_text)
```

```
(1244184, 1000)
  (0, 190)	1.0
  (1, 577)	0.5648845718512744
  (1, 983)	0.5613262186342711
  (1, 335)	0.388645233196061
  (1, 15)	0.4634362733668268
  (2, 675)	0.8031095011783936
  (2, 134)	0.5958314603283311
  (3, 739)	0.3358350220275444
  (3, 625)	0.35987383218253743
  (3, 843)	0.36981008504745183
  (3, 65)	0.39767024929073325
  (3, 827)	0.4033666675171786
  (3, 597)	0.3991782963072273
  (3, 32)	0.3751753670080188
  (4, 911)	0.44240540659438593
  (4, 67)	0.32533847523715337
  (4, 21)	0.45624646678594905
  (4, 843)	0.39176555206870584
  (4, 597)	0.42287734150511347
  (4, 32)	0.3974493685309207
  (5, 979)	1.0
  (6, 117)	0.7411081175072274
```

```
idf=vect.idf_
dd=dict(zip(vect.get_feature_names_out(), idf))
l=sorted(dd, key=(dd).get)
# print(l)
print(l[0],l[-1])
print(dd['polic'])
```

```
polic semi
4.437306523204708
```

5. Semantic Analysis :

```
from sklearn.decomposition import TruncatedSVD
lsa_model = TruncatedSVD(n_components=10, algorithm='randomized', n_iter=10, random_state=42)

lsa_top=lsa_model.fit_transform(vect_text)
```

```
print(lsa_top)
print(lsa_top.shape)
```

```
[[ 0.00041836  0.02038933  0.02714149 ... -0.00319518  0.00091593
   0.00071417]
 [ 0.0011945   0.0600878   0.04072359 ...  0.01923555 -0.16908529
  -0.19874375]
 [ 0.00109207  0.061968    0.08973196 ...  0.3746724   0.32827177
  -0.11221815]
 ...
 [ 0.00171047  0.11330495  0.24387278 ...  0.00573426  0.01656982
  -0.03600617]
 [ 0.00054127  0.04083816 -0.00405718 ...  0.00610421 -0.02532323
   0.01622376]
 [ 0.00153814  0.09662501  0.2112592  ...  0.01744683  0.00956658
  -0.01188148]]
(1244184, 10)
```

```python
l=lsa_top[0]
print("Document 0 :")
for i,topic in enumerate(l):
    print("Topic ",i," : ",topic*100)
```

```
Document 0 :
Topic  0  :  0.04183634389399514
Topic  1  :  2.038933150631539
Topic  2  :  2.7141490471415266
Topic  3  :  -0.19243766923279004
Topic  4  :  -1.189662223979564
Topic  5  :  0.4251041812831996
Topic  6  :  -2.0980195238112525
Topic  7  :  -0.31951809521461105
Topic  8  :  0.09159347173828995
Topic  9  :  0.07141675294146262
```

```python
vocab = vect.get_feature_names_out()

for i, comp in enumerate(lsa_model.components_):
    vocab_comp = zip(vocab, comp)
    sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
    print("Topic "+str(i)+": ")
    for t in sorted_words:
        print(t[0],end=" ")
    print("\n")
```

```
Topic 0:
interview extend michael nrl john david smith jame polic andrew

Topic 1:
polic man charg new say court murder death face crash

Topic 2:
new say plan council australia call win govt back us

Topic 3:
polic investig probe search hunt offic say warn miss seek

Topic 4:
new polic zealand year charg name case search law investig

Topic 5:
win australia call open back cup world fire australian us

Topic 6:
win say new australia open polic cup world australian man

Topic 7:
call australia us media day kill death australian fire warn

Topic 8:
call win charg say plan council court murder face new

Topic 9:
australia plan charg day court face polic back murder get
```

6. Latent Dirichlet Allocation :

```
from sklearn.decomposition import LatentDirichletAllocation
lda_model=LatentDirichletAllocation(n_components=10,learning_method='online',random_state=42,max_iter=1)
lda_top=lda_model.fit_transform(vect_text)
```

```
print(lda_top.shape)
print(lda_top)
```

```
(1244184, 10)
[[0.05       0.05       0.55       ... 0.05       0.05       0.05      ]
 [0.03357868 0.03357629 0.03357893 ... 0.03357629 0.03357629 0.03358359]
 [0.62483445 0.04168506 0.04168506 ... 0.04168506 0.04168506 0.04168506]
 ...
 [0.03350683 0.03350683 0.03350683 ... 0.1945794  0.16795598 0.03350683]
 [0.04151832 0.04151832 0.04151832 ... 0.36011604 0.30773739 0.04151832]
 [0.03104763 0.16964913 0.03104764 ... 0.03104763 0.29098809 0.03104763]]
```

```
print(lda_model.components_)
print(lda_model.components_.shape)
```

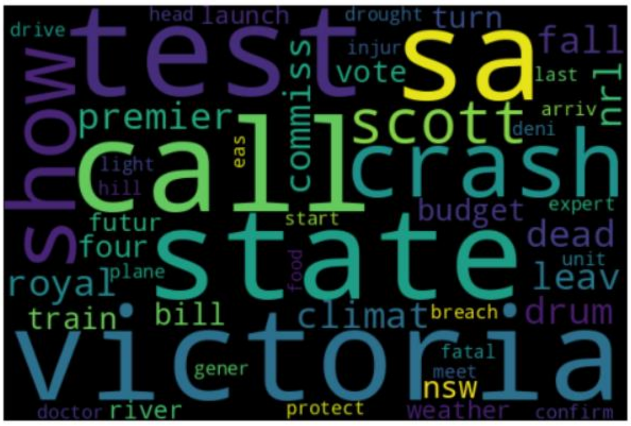
```
[[1.00004813e-01 1.00008927e-01 1.00012479e-01 ... 1.00006630e-01
  1.00004293e-01 1.00006611e-01]
 [1.00004604e-01 1.00006433e-01 1.00011208e-01 ... 1.00008387e-01
  1.00002651e-01 1.00007285e-01]
 [1.00005328e-01 1.46911463e+03 1.00008942e-01 ... 1.00007974e-01
  1.00005767e-01 1.00007248e-01]
 ...
 [1.00005003e-01 1.00011541e-01 1.00009927e-01 ... 1.00006492e-01
  1.00003876e-01 1.00011341e-01]
 [1.00007155e-01 1.00015500e-01 1.47515648e+04 ... 1.00009927e-01
  2.60466583e+03 1.00008409e-01]
 [1.00005641e-01 1.00009526e-01 1.00014645e-01 ... 1.00014224e-01
  1.00006289e-01 1.00012193e-01]]
(10, 1000)
```

```
vocab = vect.get_feature_names_out()

for i, comp in enumerate(lda_model.components_):
    vocab_comp = zip(vocab, comp)
    sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
    print("Topic "+str(i)+": ")
    for t in sorted_words:
        print(t[0],end=" ")
    print("\n")
```

```
Topic 0:
victoria call test sa state crash show scott premier royal

Topic 1:
melbourn donald south kill adelaid victorian make qld trial perth

Topic 2:
australian polic day protest open peopl could coronaviru warn commun

Topic 3:
queensland news live health restrict nsw coast hit help morrison

Topic 4:
govern home elect nt women school speak worker announc work

Topic 5:
case vaccin chang die face get alleg court afl return

Topic 6:
coronaviru trump sydney win nation tasmania canberra death investig hospit
```

```python
from wordcloud import WordCloud
# Generate a word cloud image for given topic
def draw_word_cloud(index):
    imp_words_topic=""
    comp=lda_model.components_[index]
    vocab_comp = zip(vocab, comp)
    sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:50]
    for word in sorted_words:
        imp_words_topic=imp_words_topic+" "+word[0]

        wordcloud = WordCloud(width=600, height=400).generate(imp_words_topic)
        plt.figure( figsize=(5,5))
        plt.imshow(wordcloud)
        plt.axis("off")
        plt.tight_layout()
        plt.show()
```

```python
# topic 0
draw_word_cloud(0)
```



Result :

The program is completed sucessfully