UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER SCIENCE

M.SC (Computer Science)

CERTIFICATE

Certified that the work entered in this journal was done in the computer laboratory by the student

Mr. Karankumar Yadav

Seat No.112002

Of the Class - MSc computer Science

First Year Sem 2

Semester during the year **2021-2022** in a Satisfactory manner.

Practical 1

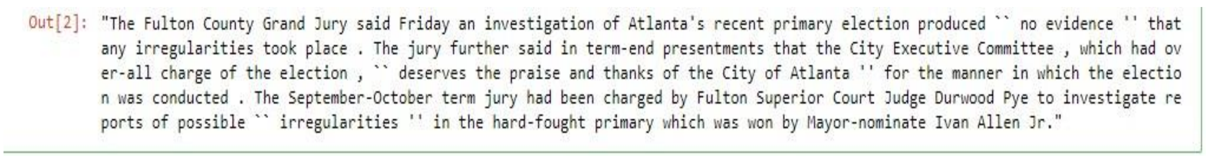
AIM :- Practical to implement tokenization

THEORY :-

Tokenization is breaking the raw text into small chunks. Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words.

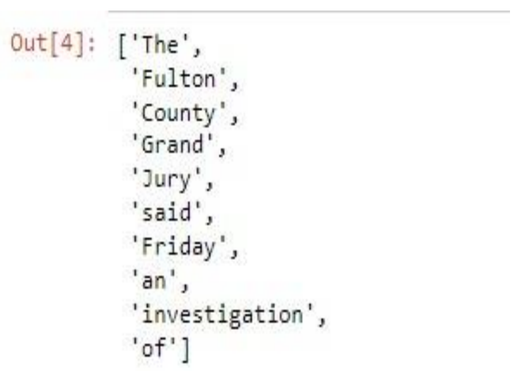
CODE / OUTPUT :-

```
import nltk

nltk.download('punkt')

import nltk.corpus

from nltk.tokenize import word_tokenize

from nltk import sent_tokenize


nltk.download('brown')

brown_words = " ".join(list(nltk.corpus.brown.words()[:102]))

brown_words
```



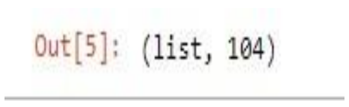
```
Out[2]: "The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced `` no evidence '' that
        any irregularities took place . The jury further said in term-end presentments that the City Executive Committee , which had ov
        er-all charge of the election , `` deserves the praise and thanks of the City of Atlanta '' for the manner in which the electio
        n was conducted . The September-October term jury had been charged by Fulton Superior Court Judge Durwood Pye to investigate re
        ports of possible `` irregularities '' in the hard-fought primary which was won by Mayor-nominate Ivan Allen Jr."
```

```
brown_tokens = word_tokenize(brown_words)

brown_tokens[:10]
```



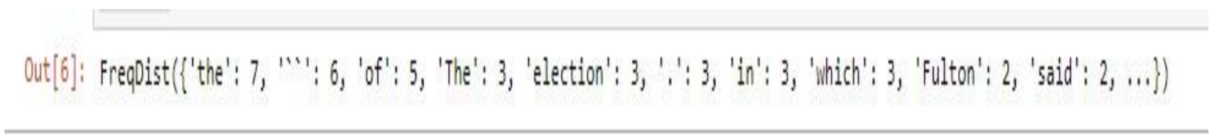
```
Out[4]: ['The',
         'Fulton',
         'County',
         'Grand',
         'Jury',
         'said',
         'Friday',
         'an',
         'investigation',
         'of']
```

```
type(brown_tokens), len(brown_tokens)
```



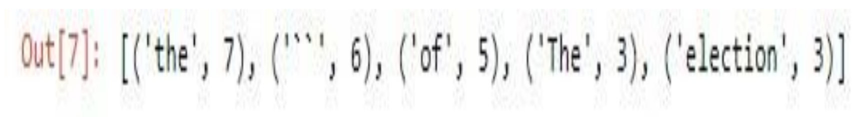
```
Out[5]: (list, 104)
```

```
from nltk.probability import FreqDist

freqList = FreqDist(brown_tokens)

freqList
```



```
Out[6]: FreqDist({'the': 7, '``': 6, 'of': 5, 'The': 3, 'election': 3, '.': 3, 'in': 3, 'which': 3, 'Fulton': 2, 'said': 2, ...})
```

```
freqList.most_common(5)
```



```
Out[7]: [('the', 7), ('``', 6), ('of', 5), ('The', 3), ('election', 3)]
```

Practical 2

AIM :- Practical to implement POS Tagging

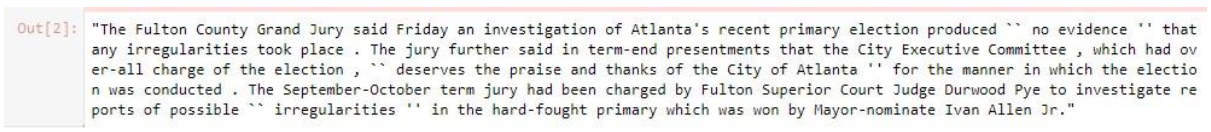
THEORY : -

What is Part-of-speech (POS) tagging ?

It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

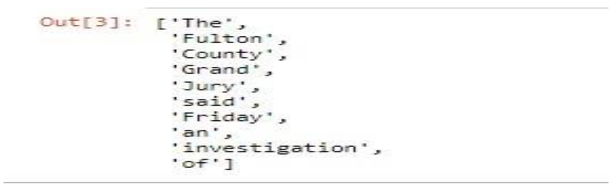
CODE / OUTPUT : -

```
import nltk
nltk.download('punkt')
import nltk.corpus
from nltk.tokenize import word_tokenize

nltk.download('brown')
brown_words = " ".join(list(nltk.corpus.brown.words()[:102]))
brown_words
```



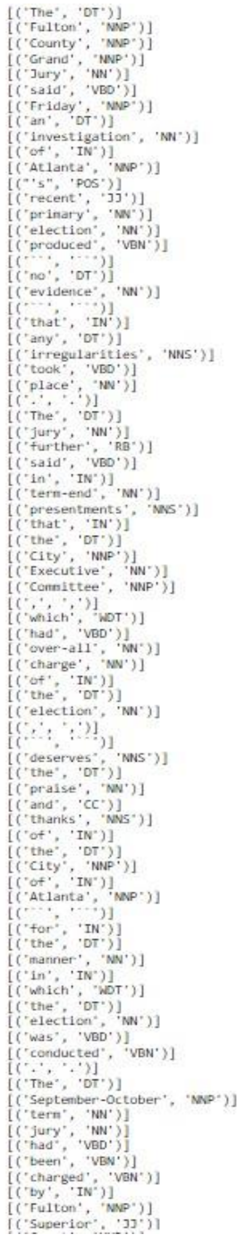
```
Out[2]: "The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced `` no evidence '' that
        any irregularities took place . The jury further said in term-end presentments that the City Executive Committee , which had ov
        er-all charge of the election , `` deserves the praise and thanks of the City of Atlanta '' for the manner in which the electio
        n was conducted . The September-October term jury had been charged by Fulton Superior Court Judge Durwood Pye to investigate re
        ports of possible `` irregularities '' in the hard-fought primary which was won by Mayor-nominate Ivan Allen Jr."
```

```
brown_tokens = word_tokenize(brown_words)
brown_tokens[:10]
```

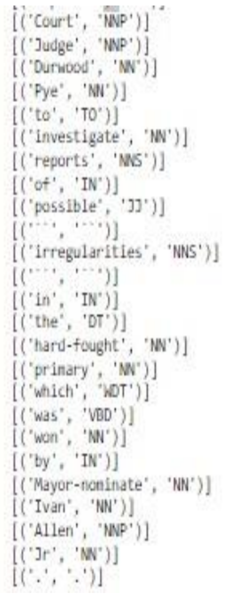


```
Out[3]: ['The',
         'Fulton',
         'County',
         'Grand',
         'Jury',
         'said',
         'Friday',
         'an',
         'investigation',
         'of']
```

```
for token in brown_tokens:
  print(nltk.pos_tag([token]))
```

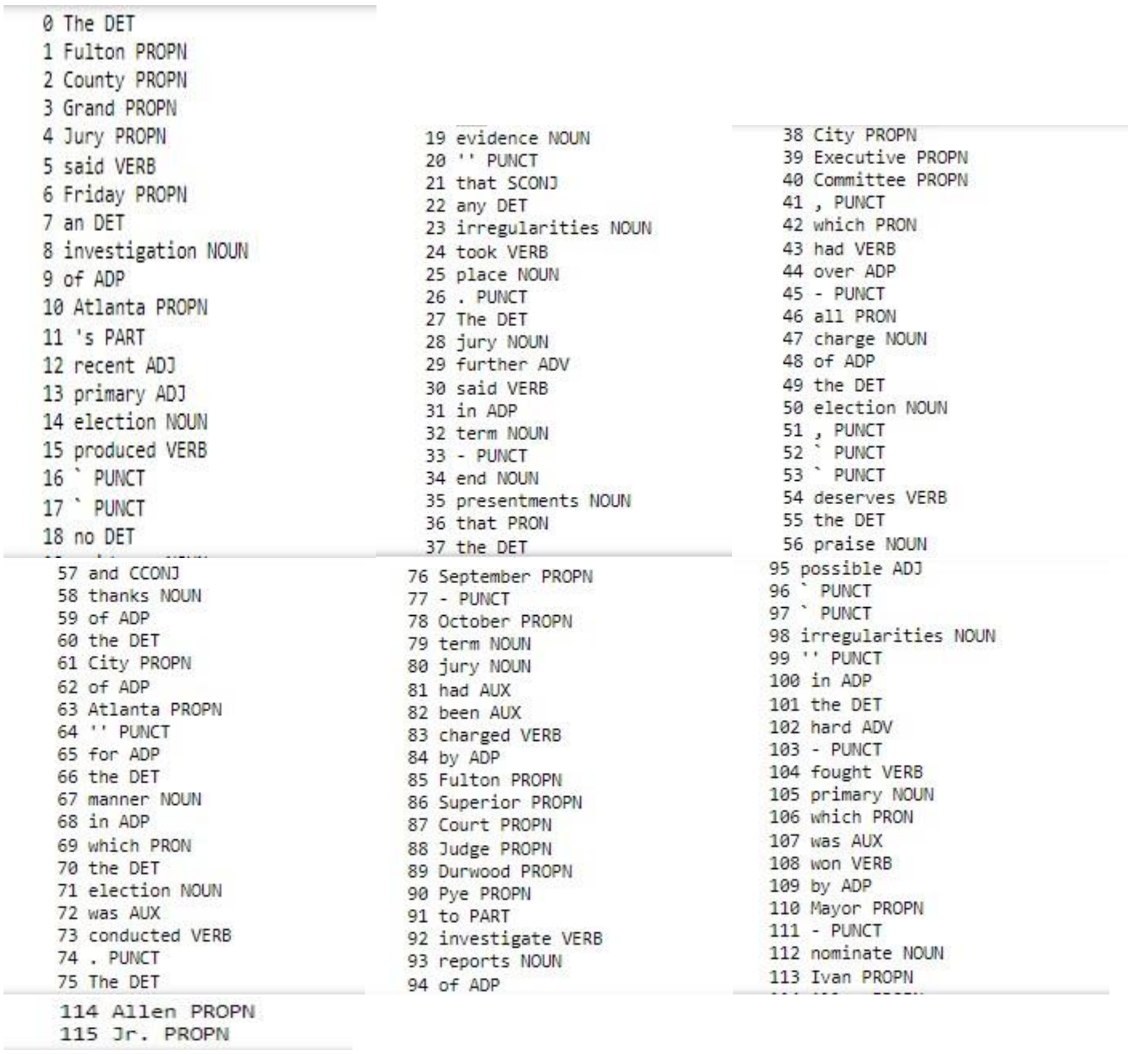


```
[('The', 'DT')]
[('Fulton', 'NNP')]
[('County', 'NNP')]
[('Grand', 'NNP')]
[('Jury', 'NN')]
[('said', 'VBD')]
[('Friday', 'NNP')]
[('an', 'DT')]
[('investigation', 'NN')]
[('of', 'IN')]
[('Atlanta', 'NNP')]
[("'s", 'POS')]
[('recent', 'JJ')]
[('primary', 'NN')]
[('election', 'NN')]
[('produced', 'VBN')]
[('``', '``')]
[('no', 'DT')]
[('evidence', 'NN')]
[('``', '``')]
[('that', 'IN')]
[('any', 'DT')]
[('irregularities', 'NNS')]
[('took', 'VBD')]
[('place', 'NN')]
[('.', '.')]
[('The', 'DT')]
[('jury', 'NN')]
[('further', 'RB')]
[('said', 'VBD')]
[('in', 'IN')]
[('term-end', 'NN')]
[('presentments', 'NNS')]
[('that', 'IN')]
[('the', 'DT')]
[('City', 'NNP')]
[('Executive', 'NN')]
[('Committee', 'NNP')]
[(',', ',')]
[('which', 'WDT')]
[('had', 'VBD')]
[('over-all', 'NN')]
[('charge', 'NN')]
[('of', 'IN')]
[('the', 'DT')]
[('election', 'NN')]
[(',', ',')]
[('``', '``')]
[('deserves', 'NNS')]
[('the', 'DT')]
[('praise', 'NN')]
[('and', 'CC')]
[('thanks', 'NNS')]
[('of', 'IN')]
[('the', 'DT')]
[('City', 'NNP')]
[('of', 'IN')]
[('Atlanta', 'NNP')]
[('``', '``')]
[('for', 'IN')]
[('the', 'DT')]
[('manner', 'NN')]
[('in', 'IN')]
[('which', 'WDT')]
[('the', 'DT')]
[('election', 'NN')]
[('was', 'VBD')]
[('conducted', 'VBN')]
[('.', '.')]
[('The', 'DT')]
[('September-October', 'NNP')]
[('term', 'NN')]
[('jury', 'NN')]
[('had', 'VBD')]
[('been', 'VBN')]
[('charged', 'VBN')]
[('by', 'IN')]
[('Fulton', 'NNP')]
[('Superior', 'JJ')]
```



```
[('Court', 'NNP')]
[('Judge', 'NNP')]
[('Durwood', 'NN')]
[('Pye', 'NN')]
[('to', 'TO')]
[('investigate', 'NN')]
[('reports', 'NNS')]
[('of', 'IN')]
[('possible', 'JJ')]
[('``', '``')]
[('irregularities', 'NNS')]
[('``', '``')]
[('in', 'IN')]
[('the', 'DT')]
[('hard-fought', 'NN')]
[('primary', 'NN')]
[('which', 'WDT')]
[('was', 'VBD')]
[('won', 'NN')]
[('by', 'IN')]
[('Mayor-nominate', 'NN')]
[('Ivan', 'NN')]
[('Allen', 'NNP')]
[('Jr', 'NN')]
[('.', '.')]
```

```
import spacy

nlp = spacy.load('en_core_web_sm')



brown_doc = nlp(brown_words)

for token in brown_doc:

  print(token.i, token.text, token.pos_)
```



```
0 The DET
1 Fulton PROPN
2 County PROPN
3 Grand PROPN
4 Jury PROPN
5 said VERB
6 Friday PROPN
7 an DET
8 investigation NOUN
9 of ADP
10 Atlanta PROPN
11 's PART
12 recent ADJ
13 primary ADJ
14 election NOUN
15 produced VERB
16 ` PUNCT
17 ` PUNCT
18 no DET
19 evidence NOUN
20 '' PUNCT
21 that SCONJ
22 any DET
23 irregularities NOUN
24 took VERB
25 place NOUN
26 . PUNCT
27 The DET
28 jury NOUN
29 further ADV
30 said VERB
31 in ADP
32 term NOUN
33 - PUNCT
34 end NOUN
35 presentments NOUN
36 that PRON
37 the DET
38 City PROPN
39 Executive PROPN
40 Committee PROPN
41 , PUNCT
42 which PRON
43 had VERB
44 over ADP
45 - PUNCT
46 all PRON
47 charge NOUN
48 of ADP
49 the DET
50 election NOUN
51 , PUNCT
52 ` PUNCT
53 ` PUNCT
54 deserves VERB
55 the DET
56 praise NOUN
57 and CCONJ
58 thanks NOUN
59 of ADP
60 the DET
61 City PROPN
62 of ADP
63 Atlanta PROPN
64 '' PUNCT
65 for ADP
66 the DET
67 manner NOUN
68 in ADP
69 which PRON
70 the DET
71 election NOUN
72 was AUX
73 conducted VERB
74 . PUNCT
75 The DET
76 September PROPN
77 - PUNCT
78 October PROPN
79 term NOUN
80 jury NOUN
81 had AUX
82 been AUX
83 charged VERB
84 by ADP
85 Fulton PROPN
86 Superior PROPN
87 Court PROPN
88 Judge PROPN
89 Durwood PROPN
90 Pye PROPN
91 to PART
92 investigate VERB
93 reports NOUN
94 of ADP
95 possible ADJ
96 ` PUNCT
97 ` PUNCT
98 irregularities NOUN
99 '' PUNCT
100 in ADP
101 the DET
102 hard ADV
103 - PUNCT
104 fought VERB
105 primary NOUN
106 which PRON
107 was AUX
108 won VERB
109 by ADP
110 Mayor PROPN
111 - PUNCT
112 nominate NOUN
113 Ivan PROPN
114 Allen PROPN
115 Jr. PROPN
```

Practical 3

AIM :- Practical to implement NER (Named Entity Recognition)

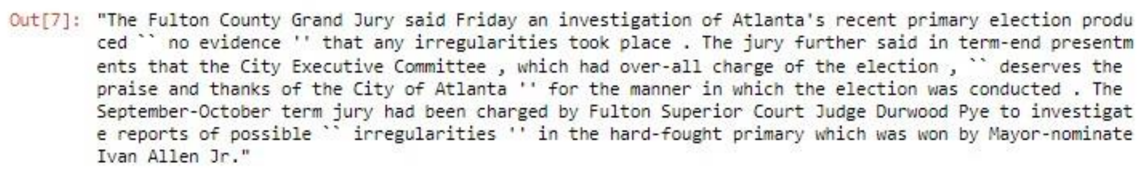
THEORY : -

Named entity recognition (NER) — sometimes referred to as entity chunking, extraction, or identification — is the task of identifying and categorizing key information (entities) in text. An entity can be any word or series of words that consistently refers to the same thing. Every detected entity is classified into a predetermined category. For example, an NER machine learning (ML) model might detect the word “super.AI” in a text and classify it as a “Company”.

NER is a form of natural language processing (NLP), a subfield of artificial intelligence. NLP is concerned with computers processing and analyzing natural language, i.e., any language that has developed naturally, rather than artificially, such as with computer coding languages.

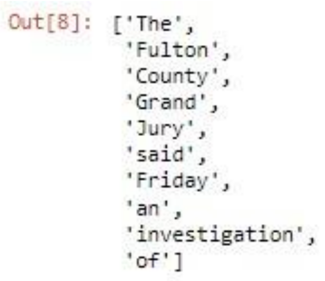
CODE / OUTPUT : -

```
import nltk

nltk.download('wordnet')

nltk.download('punkt')

nltk.download('averaged_perceptron_tagger')

from nltk.tokenize import word_tokenize


nltk.download('brown')

brown_words = " ".join(list(nltk.corpus.brown.words()[:102]))

brown_words
```



```
Out[7]: "The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produ
        ced `` no evidence '' that any irregularities took place . The jury further said in term-end presentm
        ents that the City Executive Committee , which had over-all charge of the election , `` deserves the
        praise and thanks of the City of Atlanta '' for the manner in which the election was conducted . The
        September-October term jury had been charged by Fulton Superior Court Judge Durwood Pye to investigat
        e reports of possible `` irregularities '' in the hard-fought primary which was won by Mayor-nominate
        Ivan Allen Jr."
```

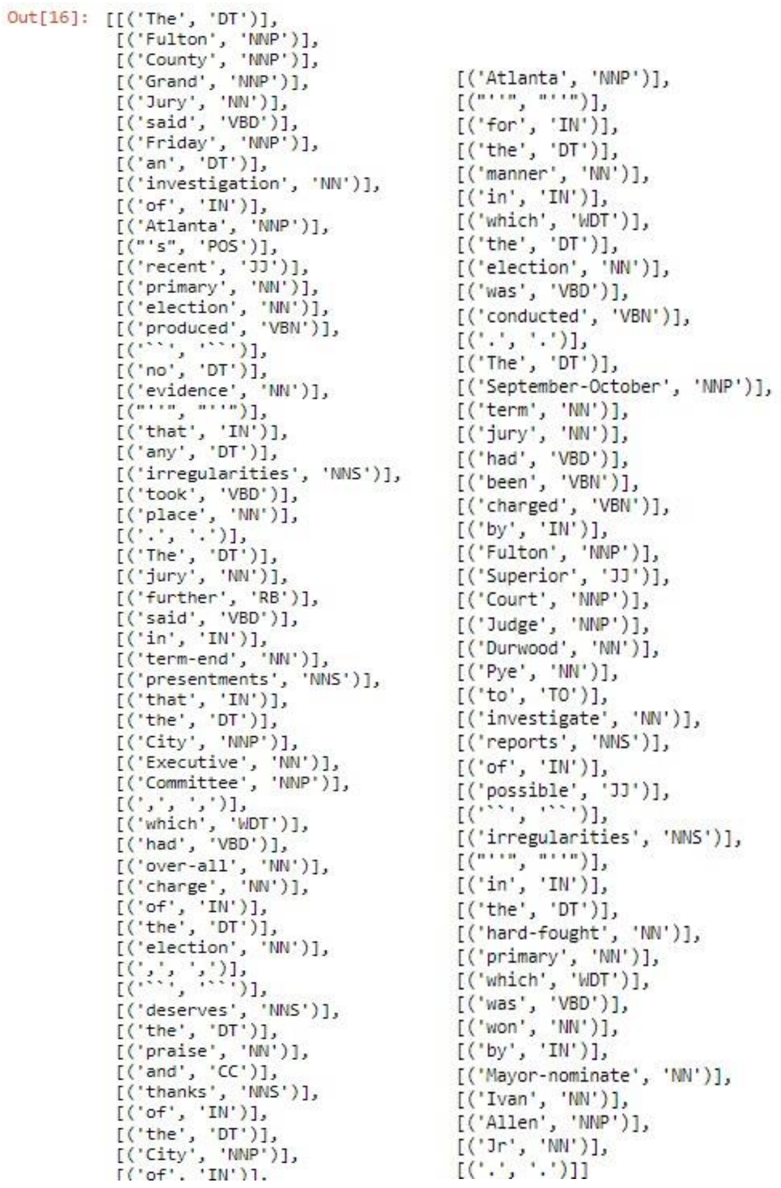
```
brown_tokens = word_tokenize(brown_words)

brown_tokens[:10]
```



```
Out[8]: ['The',
         'Fulton',
         'County',
         'Grand',
         'Jury',
         'said',
         'Friday',
         'an',
         'investigation',
         'of']
```

brown_pos = [nltk.pos_tag([token]) for token in brown_tokens]

brown_pos



```
Out[16]: [[('The', 'DT')],
          [('Fulton', 'NNP')],
          [('County', 'NNP')],
          [('Grand', 'NNP')],
          [('Jury', 'NN')],
          [('said', 'VBD')],
          [('Friday', 'NNP')],
          [('an', 'DT')],
          [('investigation', 'NN')],
          [('of', 'IN')],
          [('Atlanta', 'NNP')],
          [("'s", 'POS')],
          [('recent', 'JJ')],
          [('primary', 'NN')],
          [('election', 'NN')],
          [('produced', 'VBN')],
          [('``', '``')],
          [('no', 'DT')],
          [('evidence', 'NN')],
          [("''", "''")],
          [('that', 'IN')],
          [('any', 'DT')],
          [('irregularities', 'NNS')],
          [('took', 'VBD')],
          [('place', 'NN')],
          [('.', '.')],
          [('The', 'DT')],
          [('jury', 'NN')],
          [('further', 'RB')],
          [('said', 'VBD')],
          [('in', 'IN')],
          [('term-end', 'NN')],
          [('presentments', 'NNS')],
          [('that', 'IN')],
          [('the', 'DT')],
          [('City', 'NNP')],
          [('Executive', 'NN')],
          [('Committee', 'NNP')],
          [(',', ',')],
          [('which', 'WDT')],
          [('had', 'VBD')],
          [('over-all', 'NN')],
          [('charge', 'NN')],
          [('of', 'IN')],
          [('the', 'DT')],
          [('election', 'NN')],
          [(',', ',')],
          [('``', '``')],
          [('deserves', 'NNS')],
          [('the', 'DT')],
          [('praise', 'NN')],
          [('and', 'CC')],
          [('thanks', 'NNS')],
          [('of', 'IN')],
          [('the', 'DT')],
          [('City', 'NNP')],
          [('of', 'IN')],
```

```
 [('Atlanta', 'NNP')],
 [("''", "''")],
 [('for', 'IN')],
 [('the', 'DT')],
 [('manner', 'NN')],
 [('in', 'IN')],
 [('which', 'WDT')],
 [('the', 'DT')],
 [('election', 'NN')],
 [('was', 'VBD')],
 [('conducted', 'VBN')],
 [('.', '.')],
 [('The', 'DT')],
 [('September-October', 'NNP')],
 [('term', 'NN')],
 [('jury', 'NN')],
 [('had', 'VBD')],
 [('been', 'VBN')],
 [('charged', 'VBN')],
 [('by', 'IN')],
 [('Fulton', 'NNP')],
 [('Superior', 'JJ')],
 [('Court', 'NNP')],
 [('Judge', 'NNP')],
 [('Durwood', 'NN')],
 [('Pye', 'NN')],
 [('to', 'TO')],
 [('investigate', 'NN')],
 [('reports', 'NNS')],
 [('of', 'IN')],
 [('possible', 'JJ')],
 [('``', '``')],
 [('irregularities', 'NNS')],
 [("''", "''")],
 [('in', 'IN')],
 [('the', 'DT')],
 [('hard-fought', 'NN')],
 [('primary', 'NN')],
 [('which', 'WDT')],
 [('was', 'VBD')],
 [('won', 'NN')],
 [('by', 'IN')],
 [('Mayor-nominate', 'NN')],
 [('Ivan', 'NN')],
 [('Allen', 'NNP')],
 [('Jr', 'NN')],
 [('.', '.')]]
```

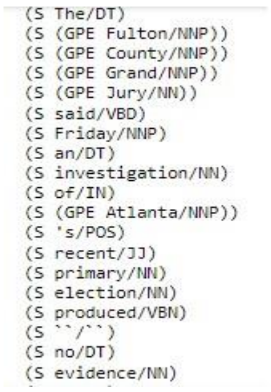
nltk.download('maxent_ne_chunker')

nltk.download('words')

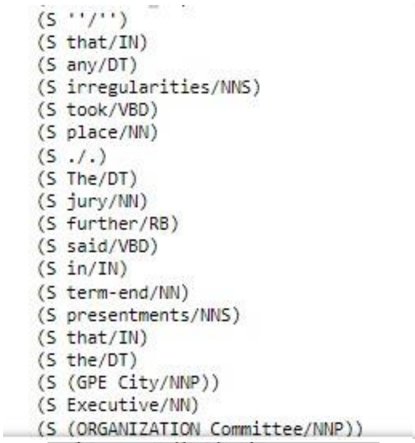
from nltk.chunk import ne_chunk

for item in brown_pos:

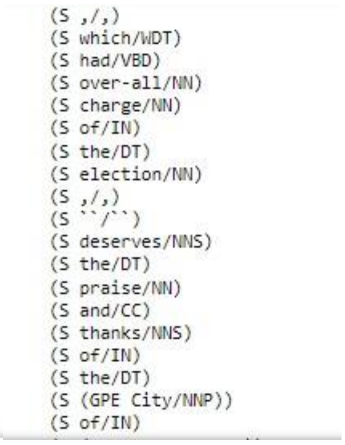
  print(ne_chunk(item))



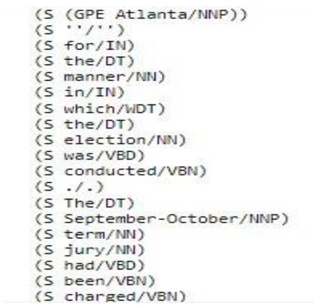
```
(S The/DT)
(S (GPE Fulton/NNP))
(S (GPE County/NNP))
(S (GPE Grand/NNP))
(S (GPE Jury/NN))
(S said/VBD)
(S Friday/NNP)
(S an/DT)
(S investigation/NN)
(S of/IN)
(S (GPE Atlanta/NNP))
(S 's/POS)
(S recent/JJ)
(S primary/NN)
(S election/NN)
(S produced/VBN)
(S ``/``)
(S no/DT)
(S evidence/NN)
```



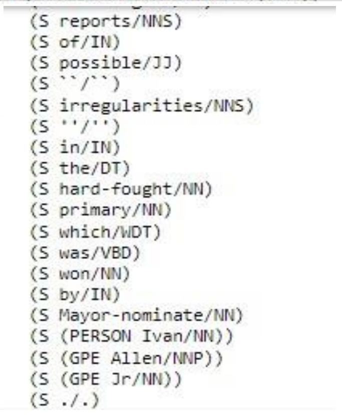
```
(S ''/'')
(S that/IN)
(S any/DT)
(S irregularities/NNS)
(S took/VBD)
(S place/NN)
(S ./.)
(S The/DT)
(S jury/NN)
(S further/RB)
(S said/VBD)
(S in/IN)
(S term-end/NN)
(S presentments/NNS)
(S that/IN)
(S the/DT)
(S (GPE City/NNP))
(S Executive/NN)
(S (ORGANIZATION Committee/NNP))
```



```
(S ,/,)
(S which/WDT)
(S had/VBD)
(S over-all/NN)
(S charge/NN)
(S of/IN)
(S the/DT)
(S election/NN)
(S ,/,)
(S ``/``)
(S deserves/NNS)
(S the/DT)
(S praise/NN)
(S and/CC)
(S thanks/NNS)
(S of/IN)
(S the/DT)
(S (GPE City/NNP))
(S of/IN)
```



```
(S (GPE Atlanta/NNP))
(S ''/'')
(S for/IN)
(S the/DT)
(S manner/NN)
(S in/IN)
(S which/WDT)
(S the/DT)
(S election/NN)
(S was/VBD)
(S conducted/VBN)
(S ./.)
(S The/DT)
(S September-October/NNP)
(S term/NN)
(S jury/NN)
(S had/VBD)
(S been/VBN)
(S charged/VBN)
```



```
(S reports/NNS)
(S of/IN)
(S possible/JJ)
(S ``/``)
(S irregularities/NNS)
(S ''/'')
(S in/IN)
(S the/DT)
(S hard-fought/NN)
(S primary/NN)
(S which/WDT)
(S was/VBD)
(S won/NN)
(S by/IN)
(S Mayor-nominate/NN)
(S (PERSON Ivan/NN))
(S (GPE Allen/NNP))
(S (GPE Jr/NN))
(S ./.)
```

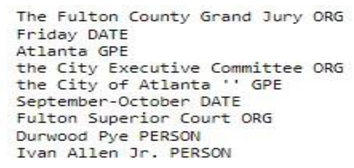
import spacy

nlp = spacy.load('en_core_web_sm')

brown_doc = nlp("".join(brown_words))

for ent in brown_doc.ents:

  print(ent.text, ent.label_)



```
The Fulton County Grand Jury ORG
Friday DATE
Atlanta GPE
the City Executive Committee ORG
the City of Atlanta '' GPE
September-October DATE
Fulton Superior Court ORG
Durwood Pye PERSON
Ivan Allen Jr. PERSON
```

Practical 4

AIM :- Practical to implement Stemming and Lemmatization

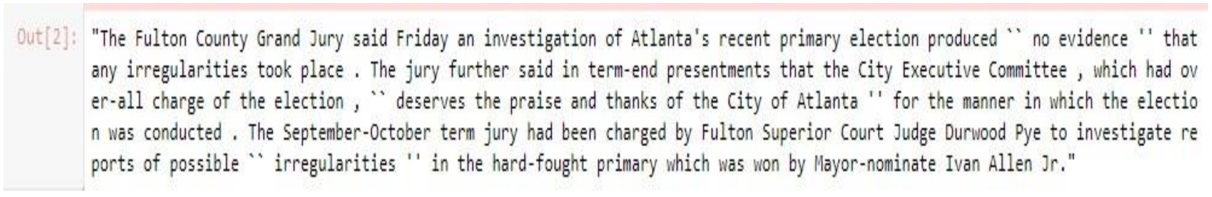
THEORY : -

To put simply, stemming is the process of removing a part of a word, or reducing a word to its stem or root. This might not necessarily mean we're reducing a word to its dictionary root.

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma .

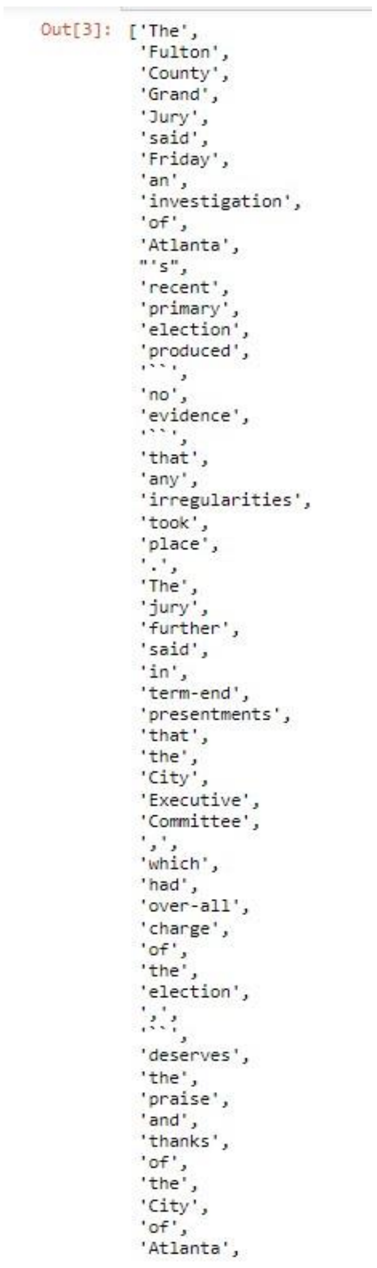
CODE / OUTPUT : -

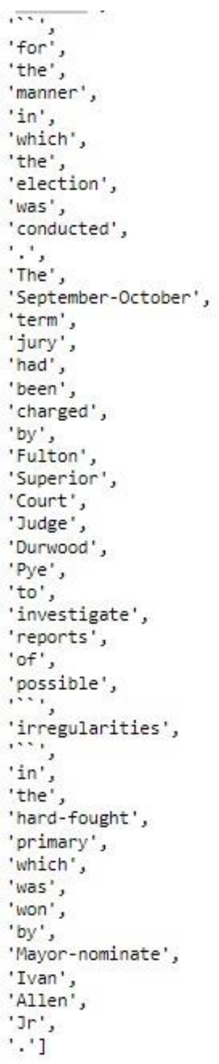
```
import nltk

nltk.download('punkt')

import nltk.corpus

from nltk.tokenize import word_tokenize


nltk.download('brown')

brown_words = " ".join(list(nltk.corpus.brown.words()[:102]))

brown_words
```



```
Out[2]: "The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced `` no evidence '' that
        any irregularities took place . The jury further said in term-end presentments that the City Executive Committee , which had ov
        er-all charge of the election , `` deserves the praise and thanks of the City of Atlanta '' for the manner in which the electio
        n was conducted . The September-October term jury had been charged by Fulton Superior Court Judge Durwood Pye to investigate re
        ports of possible `` irregularities '' in the hard-fought primary which was won by Mayor-nominate Ivan Allen Jr."
```

```
brown_tokens = word_tokenize(brown_words)
brown_tokens
```



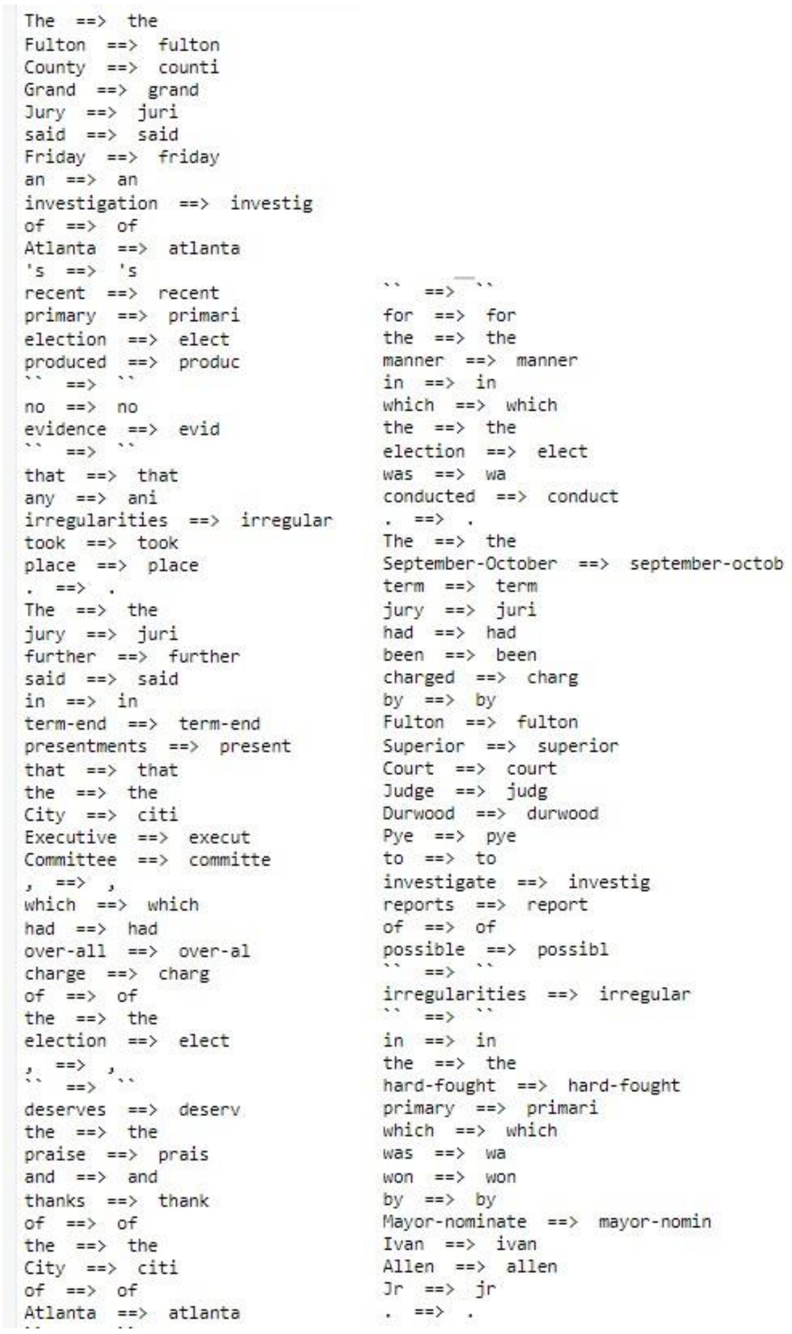


```
Out[3]: ['The',
         'Fulton',
         'County',
         'Grand',
         'Jury',
         'said',
         'Friday',
         'an',
         'investigation',
         'of',
         'Atlanta',
         "'s",
         'recent',
         'primary',
         'election',
         'produced',
         '``',
         'no',
         'evidence',
         '``',
         'that',
         'any',
         'irregularities',
         'took',
         'place',
         '.',
         'The',
         'jury',
         'further',
         'said',
         'in',
         'term-end',
         'presentments',
         'that',
         'the',
         'City',
         'Executive',
         'Committee',
         ',',
         'which',
         'had',
         'over-all',
         'charge',
         'of',
         'the',
         'election',
         ',',
         '``',
         'deserves',
         'the',
         'praise',
         'and',
         'thanks',
         'of',
         'the',
         'City',
         'of',
         'Atlanta',
```

```
'``',
'for',
'the',
'manner',
'in',
'which',
'the',
'election',
'was',
'conducted',
'.',
'The',
'September-October',
'term',
'jury',
'had',
'been',
'charged',
'by',
'Fulton',
'Superior',
'Court',
'Judge',
'Durwood',
'Pye',
'to',
'investigate',
'reports',
'of',
'possible',
'``',
'irregularities',
'``',
'in',
'the',
'hard-fought',
'primary',
'which',
'was',
'won',
'by',
'Mayor-nominate',
'Ivan',
'Allen',
'Jr',
'.']
```

```
from nltk.stem import PorterStemmer
pst = PorterStemmer()
```

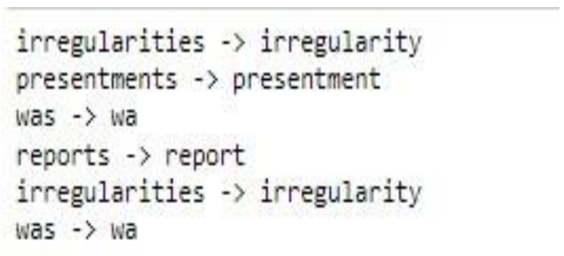
```
for token in brown_tokens:
  print(token, " ==> ", pst.stem(token))
```



```
The  ==>  the
Fulton  ==>  fulton
County  ==>  counti
Grand  ==>  grand
Jury  ==>  juri
said  ==>  said
Friday  ==>  friday
an  ==>  an
investigation  ==>  investig
of  ==>  of
Atlanta  ==>  atlanta
's  ==>  's
recent  ==>  recent
primary  ==>  primari
election  ==>  elect
produced  ==>  produc
``  ==>  ``
no  ==>  no
evidence  ==>  evid
``  ==>  ``
that  ==>  that
any  ==>  ani
irregularities  ==>  irregular
took  ==>  took
place  ==>  place
.  ==>  .
The  ==>  the
jury  ==>  juri
further  ==>  further
said  ==>  said
in  ==>  in
term-end  ==>  term-end
presentments  ==>  present
that  ==>  that
the  ==>  the
City  ==>  citi
Executive  ==>  execut
Committee  ==>  committe
,  ==>  ,
which  ==>  which
had  ==>  had
over-all  ==>  over-al
charge  ==>  charg
of  ==>  of
the  ==>  the
election  ==>  elect
,  ==>  ,
``  ==>  ``
deserves  ==>  deserv
the  ==>  the
praise  ==>  prais
and  ==>  and
thanks  ==>  thank
of  ==>  of
the  ==>  the
City  ==>  citi
of  ==>  of
Atlanta  ==>  atlanta
```

```
``  ==>  ``
for  ==>  for
the  ==>  the
manner  ==>  manner
in  ==>  in
which  ==>  which
the  ==>  the
election  ==>  elect
was  ==>  wa
conducted  ==>  conduct
.  ==>  .
The  ==>  the
September-October  ==>  september-octob
term  ==>  term
jury  ==>  juri
had  ==>  had
been  ==>  been
charged  ==>  charg
by  ==>  by
Fulton  ==>  fulton
Superior  ==>  superior
Court  ==>  court
Judge  ==>  judg
Durwood  ==>  durwood
Pye  ==>  pye
to  ==>  to
investigate  ==>  investig
reports  ==>  report
of  ==>  of
possible  ==>  possibl
``  ==>  ``
irregularities  ==>  irregular
``  ==>  ``
in  ==>  in
the  ==>  the
hard-fought  ==>  hard-fought
primary  ==>  primari
which  ==>  which
was  ==>  wa
won  ==>  won
by  ==>  by
Mayor-nominate  ==>  mayor-nomin
Ivan  ==>  ivan
Allen  ==>  allen
Jr  ==>  jr
.  ==>  .
```

```
nltk.download('wordnet')
from nltk.stem import wordnet
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()



for token in brown_tokens:
  if token == lemmatizer.lemmatize(token):
    continue
  print(token + " -> " + lemmatizer.lemmatize(token))
```



```
irregularities -> irregularity
presentments -> presentment
was -> wa
reports -> report
irregularities -> irregularity
was -> wa
```

Practical 5

AIM :- Practical to implement Bigrams, Trigrams, and N-grams

THEORY : -

Consider the following sentence - “I love reading blogs about data science"

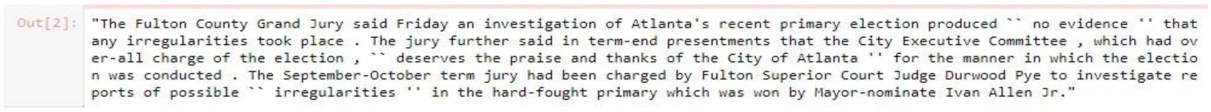
A 1-gram (or unigram) is a one-word sequence. For the above sentence, the unigrams would simply be: “I”, “love”, “reading”, “blogs”, “about”, “data”, “science”

A 2-gram (or bigram) is a two-word sequence of words, like “I love”, “love reading”

3-gram (or trigram) is a three-word sequence of words like “I love reading”, “about data science”

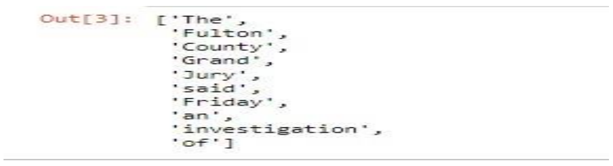
CODE / OUTPUT : -

```
import nltk

nltk.download('punkt')

import nltk.corpus

from nltk.tokenize import word_tokenize


nltk.download('brown')

brown_words = " ".join(list(nltk.corpus.brown.words()[:102]))

brown_words
```



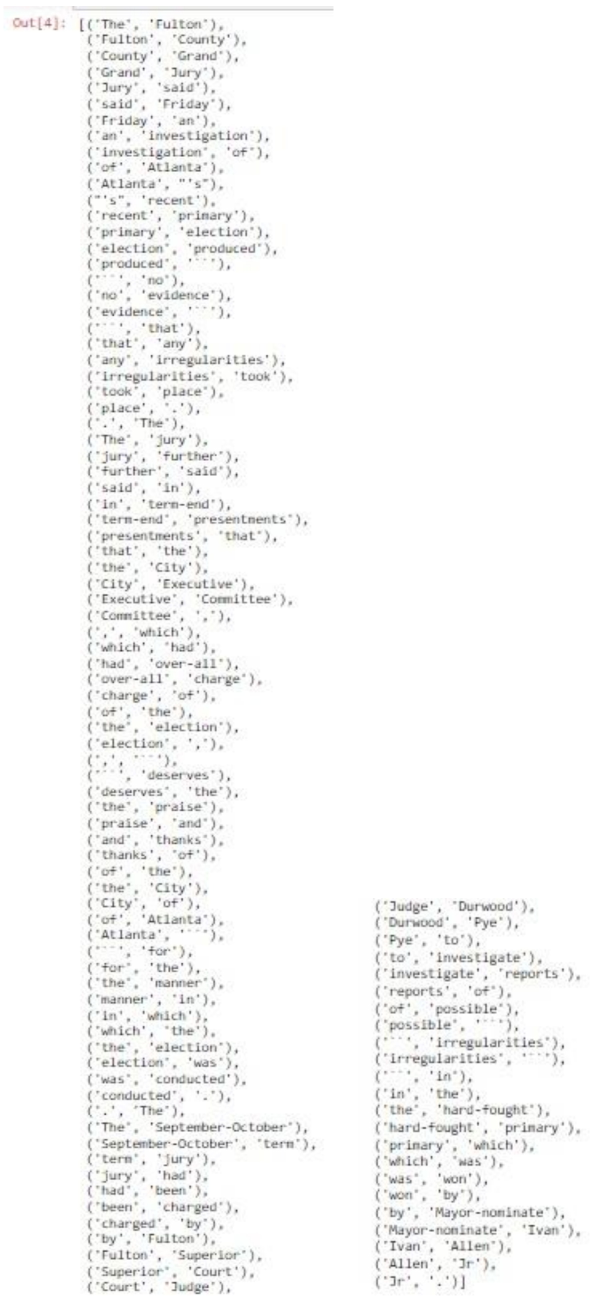
```
Out[2]: "The Fulton County Grand Jury said Friday an investigation of Atlanta's recent primary election produced `` no evidence '' that
        any irregularities took place . The jury further said in term-end presentments that the City Executive Committee , which had ov
        er-all charge of the election , `` deserves the praise and thanks of the City of Atlanta '' for the manner in which the electio
        n was conducted . The September-October term jury had been charged by Fulton Superior Court Judge Durwood Pye to investigate re
        ports of possible `` irregularities '' in the hard-fought primary which was won by Mayor-nominate Ivan Allen Jr."
```

```
brown_tokens = word_tokenize(brown_words)

brown_tokens[:10]
```



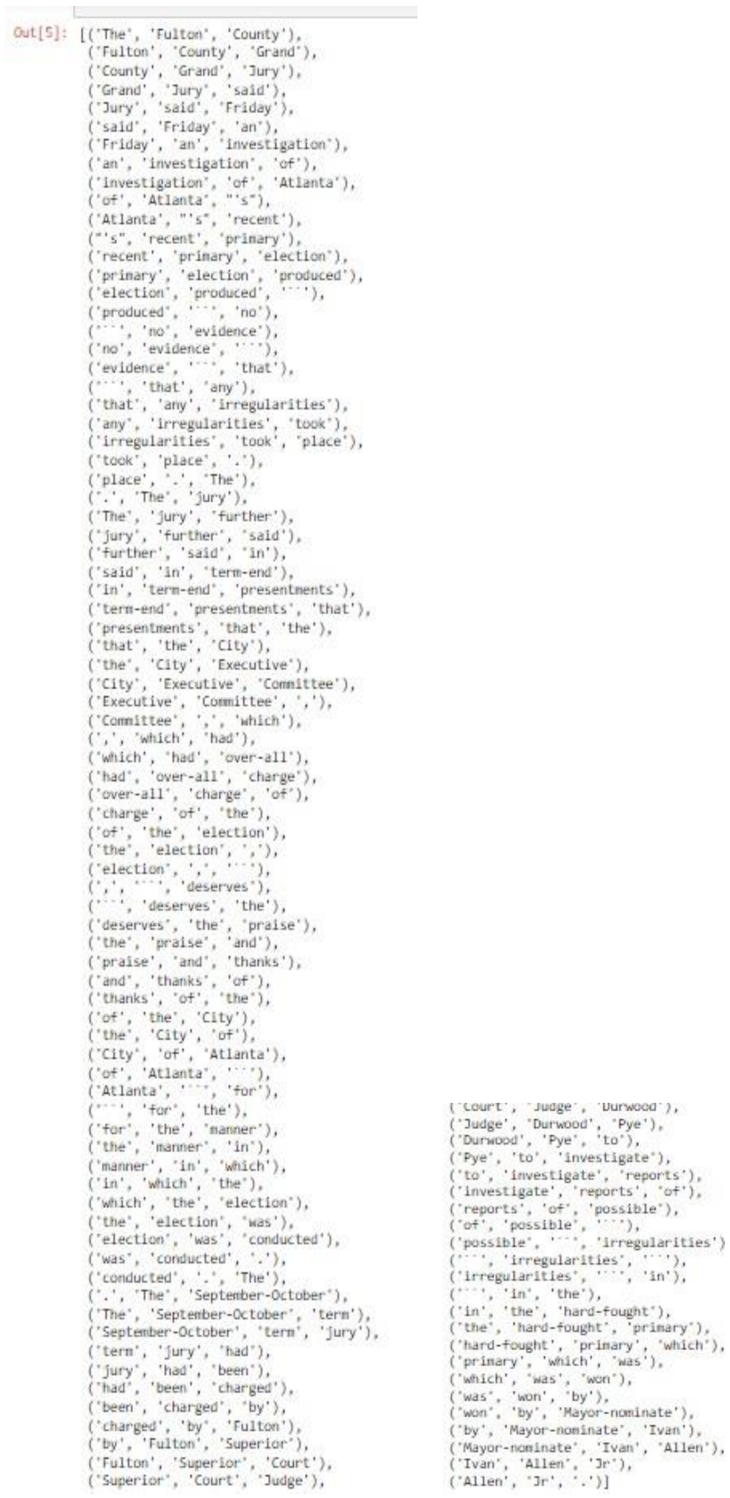
```
Out[3]: ['The',
         'Fulton',
         'County',
         'Grand',
         'Jury',
         'said',
         'Friday',
         'an',
         'investigation',
         'of']
```

```
list(nltk.bigrams(brown_tokens))
```



```
Out[4]: [('The', 'Fulton'),
         ('Fulton', 'County'),
         ('County', 'Grand'),
         ('Grand', 'Jury'),
         ('Jury', 'said'),
         ('said', 'Friday'),
         ('Friday', 'an'),
         ('an', 'investigation'),
         ('investigation', 'of'),
         ('of', 'Atlanta'),
         ('Atlanta', "'s"),
         ("'s", 'recent'),
         ('recent', 'primary'),
         ('primary', 'election'),
         ('election', 'produced'),
         ('produced', '``'),
         ('``', 'no'),
         ('no', 'evidence'),
         ('evidence', '``'),
         ('``', 'that'),
         ('that', 'any'),
         ('any', 'irregularities'),
         ('irregularities', 'took'),
         ('took', 'place'),
         ('place', '.'),
         ('.', 'The'),
         ('The', 'jury'),
         ('jury', 'further'),
         ('further', 'said'),
         ('said', 'in'),
         ('in', 'term-end'),
         ('term-end', 'presentments'),
         ('presentments', 'that'),
         ('that', 'the'),
         ('the', 'City'),
         ('City', 'Executive'),
         ('Executive', 'Committee'),
         ('Committee', ','),
         (',', 'which'),
         ('which', 'had'),
         ('had', 'over-all'),
         ('over-all', 'charge'),
         ('charge', 'of'),
         ('of', 'the'),
         ('the', 'election'),
         ('election', ','),
         (',', '``'),
         ('``', 'deserves'),
         ('deserves', 'the'),
         ('the', 'praise'),
         ('praise', 'and'),
         ('and', 'thanks'),
         ('thanks', 'of'),
         ('of', 'the'),
         ('the', 'City'),
         ('City', 'of'),
         ('of', 'Atlanta'),
         ('Atlanta', '``'),
         ('``', 'for'),
         ('for', 'the'),
         ('the', 'manner'),
         ('manner', 'in'),
         ('in', 'which'),
         ('which', 'the'),
         ('the', 'election'),
         ('election', 'was'),
         ('was', 'conducted'),
         ('conducted', '.'),
         ('.', 'The'),
         ('The', 'September-October'),
         ('September-October', 'term'),
         ('term', 'jury'),
         ('jury', 'had'),
         ('had', 'been'),
         ('been', 'charged'),
         ('charged', 'by'),
         ('by', 'Fulton'),
         ('Fulton', 'Superior'),
         ('Superior', 'Court'),
         ('Court', 'Judge'),
         ('Judge', 'Durwood'),
         ('Durwood', 'Pye'),
         ('Pye', 'to'),
         ('to', 'investigate'),
         ('investigate', 'reports'),
         ('reports', 'of'),
         ('of', 'possible'),
         ('possible', '``'),
         ('``', 'irregularities'),
         ('irregularities', '``'),
         ('``', 'in'),
         ('in', 'the'),
         ('the', 'hard-fought'),
         ('hard-fought', 'primary'),
         ('primary', 'which'),
         ('which', 'was'),
         ('was', 'won'),
         ('won', 'by'),
         ('by', 'Mayor-nominate'),
         ('Mayor-nominate', 'Ivan'),
         ('Ivan', 'Allen'),
         ('Allen', 'Jr'),
         ('Jr', '.')]
```

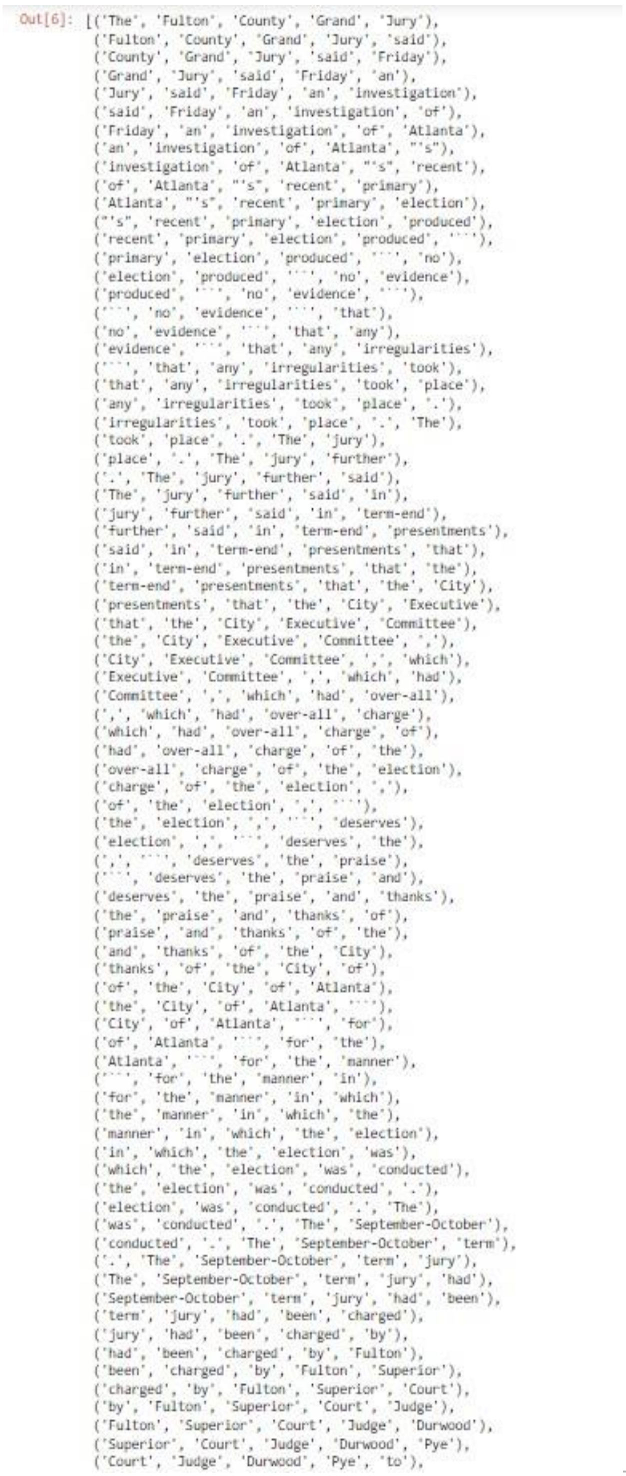
```
list(nltk.trigrams(brown_tokens))
```



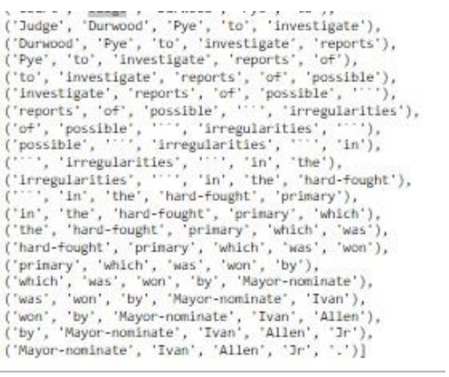
```
Out[5]: [('The', 'Fulton', 'County'),
         ('Fulton', 'County', 'Grand'),
         ('County', 'Grand', 'Jury'),
         ('Grand', 'Jury', 'said'),
         ('Jury', 'said', 'Friday'),
         ('said', 'Friday', 'an'),
         ('Friday', 'an', 'investigation'),
         ('an', 'investigation', 'of'),
         ('investigation', 'of', 'Atlanta'),
         ('of', 'Atlanta', "'s"),
         ('Atlanta', "'s", 'recent'),
         ("'s", 'recent', 'primary'),
         ('recent', 'primary', 'election'),
         ('primary', 'election', 'produced'),
         ('election', 'produced', '``'),
         ('produced', '``', 'no'),
         ('``', 'no', 'evidence'),
         ('no', 'evidence', "''"),
         ('evidence', "''", 'that'),
         ("''", 'that', 'any'),
         ('that', 'any', 'irregularities'),
         ('any', 'irregularities', 'took'),
         ('irregularities', 'took', 'place'),
         ('took', 'place', '.'),
         ('place', '.', 'The'),
         ('.', 'The', 'jury'),
         ('The', 'jury', 'further'),
         ('jury', 'further', 'said'),
         ('further', 'said', 'in'),
         ('said', 'in', 'term-end'),
         ('in', 'term-end', 'presentments'),
         ('term-end', 'presentments', 'that'),
         ('presentments', 'that', 'the'),
         ('that', 'the', 'City'),
         ('the', 'City', 'Executive'),
         ('City', 'Executive', 'Committee'),
         ('Executive', 'Committee', ','),
         ('Committee', ',', 'which'),
         (',', 'which', 'had'),
         ('which', 'had', 'over-all'),
         ('had', 'over-all', 'charge'),
         ('over-all', 'charge', 'of'),
         ('charge', 'of', 'the'),
         ('of', 'the', 'election'),
         ('the', 'election', ','),
         ('election', ',', '``'),
         (',', '``', 'deserves'),
         ('``', 'deserves', 'the'),
         ('deserves', 'the', 'praise'),
         ('the', 'praise', 'and'),
         ('praise', 'and', 'thanks'),
         ('and', 'thanks', 'of'),
         ('thanks', 'of', 'the'),
         ('of', 'the', 'City'),
         ('the', 'City', 'of'),
         ('City', 'of', 'Atlanta'),
         ('of', 'Atlanta', "''"),
         ('Atlanta', "''", 'for'),
         ("''", 'for', 'the'),
         ('for', 'the', 'manner'),
         ('the', 'manner', 'in'),
         ('manner', 'in', 'which'),
         ('in', 'which', 'the'),
         ('which', 'the', 'election'),
         ('the', 'election', 'was'),
         ('election', 'was', 'conducted'),
         ('was', 'conducted', '.'),
         ('conducted', '.', 'The'),
         ('.', 'The', 'September-October'),
         ('The', 'September-October', 'term'),
         ('September-October', 'term', 'jury'),
         ('term', 'jury', 'had'),
         ('jury', 'had', 'been'),
         ('had', 'been', 'charged'),
         ('been', 'charged', 'by'),
         ('charged', 'by', 'Fulton'),
         ('by', 'Fulton', 'Superior'),
         ('Fulton', 'Superior', 'Court'),
         ('Superior', 'Court', 'Judge'),
```

```
('Court', 'Judge', 'Durwood'),
('Judge', 'Durwood', 'Pye'),
('Durwood', 'Pye', 'to'),
('Pye', 'to', 'investigate'),
('to', 'investigate', 'reports'),
('investigate', 'reports', 'of'),
('reports', 'of', 'possible'),
('of', 'possible', '``'),
('possible', '``', 'irregularities')
('``', 'irregularities', "''"),
('irregularities', "''", 'in'),
("''", 'in', 'the'),
('in', 'the', 'hard-fought'),
('the', 'hard-fought', 'primary'),
('hard-fought', 'primary', 'which'),
('primary', 'which', 'was'),
('which', 'was', 'won'),
('was', 'won', 'by'),
('won', 'by', 'Mayor-nominate'),
('by', 'Mayor-nominate', 'Ivan'),
('Mayor-nominate', 'Ivan', 'Allen'),
('Ivan', 'Allen', 'Jr'),
('Allen', 'Jr', '.')]
```

```
list(nltk.ngrams(brown_tokens, 5))
```



```
Out[6]: [('The', 'Fulton', 'County', 'Grand', 'Jury'),
         ('Fulton', 'County', 'Grand', 'Jury', 'said'),
         ('County', 'Grand', 'Jury', 'said', 'Friday'),
         ('Grand', 'Jury', 'said', 'Friday', 'an'),
         ('Jury', 'said', 'Friday', 'an', 'investigation'),
         ('said', 'Friday', 'an', 'investigation', 'of'),
         ('Friday', 'an', 'investigation', 'of', 'Atlanta'),
         ('an', 'investigation', 'of', 'Atlanta', "'s"),
         ('investigation', 'of', 'Atlanta', "'s", 'recent'),
         ('of', 'Atlanta', "'s", 'recent', 'primary'),
         ('Atlanta', "'s", 'recent', 'primary', 'election'),
         ("'s", 'recent', 'primary', 'election', 'produced'),
         ('recent', 'primary', 'election', 'produced', '``'),
         ('primary', 'election', 'produced', '``', 'no'),
         ('election', 'produced', '``', 'no', 'evidence'),
         ('produced', '``', 'no', 'evidence', "''"),
         ('``', 'no', 'evidence', "''", 'that'),
         ('no', 'evidence', "''", 'that', 'any'),
         ('evidence', "''", 'that', 'any', 'irregularities'),
         ("''", 'that', 'any', 'irregularities', 'took'),
         ('that', 'any', 'irregularities', 'took', 'place'),
         ('any', 'irregularities', 'took', 'place', '.'),
         ('irregularities', 'took', 'place', '.', 'The'),
         ('took', 'place', '.', 'The', 'jury'),
         ('place', '.', 'The', 'jury', 'further'),
         ('.', 'The', 'jury', 'further', 'said'),
         ('The', 'jury', 'further', 'said', 'in'),
         ('jury', 'further', 'said', 'in', 'term-end'),
         ('further', 'said', 'in', 'term-end', 'presentments'),
         ('said', 'in', 'term-end', 'presentments', 'that'),
         ('in', 'term-end', 'presentments', 'that', 'the'),
         ('term-end', 'presentments', 'that', 'the', 'City'),
         ('presentments', 'that', 'the', 'City', 'Executive'),
         ('that', 'the', 'City', 'Executive', 'Committee'),
         ('the', 'City', 'Executive', 'Committee', ','),
         ('City', 'Executive', 'Committee', ',', 'which'),
         ('Executive', 'Committee', ',', 'which', 'had'),
         ('Committee', ',', 'which', 'had', 'over-all'),
         (',', 'which', 'had', 'over-all', 'charge'),
         ('which', 'had', 'over-all', 'charge', 'of'),
         ('had', 'over-all', 'charge', 'of', 'the'),
         ('over-all', 'charge', 'of', 'the', 'election'),
         ('charge', 'of', 'the', 'election', ','),
         ('of', 'the', 'election', ',', '``'),
         ('the', 'election', ',', '``', 'deserves'),
         ('election', ',', '``', 'deserves', 'the'),
         (',', '``', 'deserves', 'the', 'praise'),
         ('``', 'deserves', 'the', 'praise', 'and'),
         ('deserves', 'the', 'praise', 'and', 'thanks'),
         ('the', 'praise', 'and', 'thanks', 'of'),
         ('praise', 'and', 'thanks', 'of', 'the'),
         ('and', 'thanks', 'of', 'the', 'City'),
         ('thanks', 'of', 'the', 'City', 'of'),
         ('of', 'the', 'City', 'of', 'Atlanta'),
         ('the', 'City', 'of', 'Atlanta', "''"),
         ('City', 'of', 'Atlanta', "''", 'for'),
         ('of', 'Atlanta', "''", 'for', 'the'),
         ('Atlanta', "''", 'for', 'the', 'manner'),
         ("''", 'for', 'the', 'manner', 'in'),
         ('for', 'the', 'manner', 'in', 'which'),
         ('the', 'manner', 'in', 'which', 'the'),
         ('manner', 'in', 'which', 'the', 'election'),
         ('in', 'which', 'the', 'election', 'was'),
         ('which', 'the', 'election', 'was', 'conducted'),
         ('the', 'election', 'was', 'conducted', '.'),
         ('election', 'was', 'conducted', '.', 'The'),
         ('was', 'conducted', '.', 'The', 'September-October'),
         ('conducted', '.', 'The', 'September-October', 'term'),
         ('.', 'The', 'September-October', 'term', 'jury'),
         ('The', 'September-October', 'term', 'jury', 'had'),
         ('September-October', 'term', 'jury', 'had', 'been'),
         ('term', 'jury', 'had', 'been', 'charged'),
         ('jury', 'had', 'been', 'charged', 'by'),
         ('had', 'been', 'charged', 'by', 'Fulton'),
         ('been', 'charged', 'by', 'Fulton', 'Superior'),
         ('charged', 'by', 'Fulton', 'Superior', 'Court'),
         ('by', 'Fulton', 'Superior', 'Court', 'Judge'),
         ('Fulton', 'Superior', 'Court', 'Judge', 'Durwood'),
         ('Superior', 'Court', 'Judge', 'Durwood', 'Pye'),
         ('Court', 'Judge', 'Durwood', 'Pye', 'to'),
```



```
('Judge', 'Durwood', 'Pye', 'to', 'investigate'),
('Durwood', 'Pye', 'to', 'investigate', 'reports'),
('Pye', 'to', 'investigate', 'reports', 'of'),
('to', 'investigate', 'reports', 'of', 'possible'),
('investigate', 'reports', 'of', 'possible', '``'),
('reports', 'of', 'possible', '``', 'irregularities'),
('of', 'possible', '``', 'irregularities', "''"),
('possible', '``', 'irregularities', "''", 'in'),
('``', 'irregularities', "''", 'in', 'the'),
('irregularities', "''", 'in', 'the', 'hard-fought'),
("''", 'in', 'the', 'hard-fought', 'primary'),
('in', 'the', 'hard-fought', 'primary', 'which'),
('the', 'hard-fought', 'primary', 'which', 'was'),
('hard-fought', 'primary', 'which', 'was', 'won'),
('primary', 'which', 'was', 'won', 'by'),
('which', 'was', 'won', 'by', 'Mayor-nominate'),
('was', 'won', 'by', 'Mayor-nominate', 'Ivan'),
('won', 'by', 'Mayor-nominate', 'Ivan', 'Allen'),
('by', 'Mayor-nominate', 'Ivan', 'Allen', 'Jr'),
('Mayor-nominate', 'Ivan', 'Allen', 'Jr', '.')]
```

Practical 6

AIM :- Practical to implement Sentiment Analysis using Naive Bayes

THEORY :-

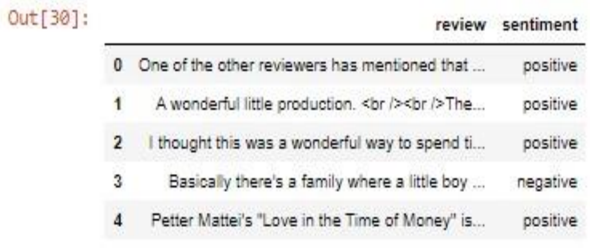
Sentiment analysis, also referred to as opinion mining, is an approach to natural language processing (NLP) that identifies the emotional tone behind a body of text. This is a popular way for organizations to determine and categorize opinions about a product, service, or idea.

Naive Bayes is the simplest and fastest classification algorithm for a large chunk of data. In various applications such as spam filtering, text classification, sentiment analysis, and recommendation systems, Naive Bayes classifier is used successfully. It uses the Bayes probability theorem for unknown class prediction.

The Naive Bayes classification technique is a simple and powerful classification task in machine learning. The use of Bayes' theorem with a strong independence assumption between the features is the basis for naive Bayes classification.

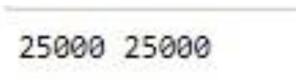
CODE / OUTPUT :-

```
import nltk
import random
from nltk.tokenize import word_tokenize
import re
import pandas as pd

df=pd.read_csv("./IMDB Dataset.csv")

df.head()
```



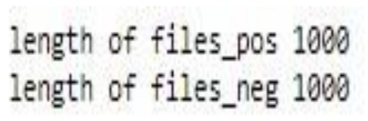
```
Out[30]:
                                           review  sentiment
0  One of the other reviewers has mentioned that ...   positive
1       A wonderful little production. <br /><br />The...   positive
2  I thought this was a wonderful way to spend ti...   positive
3      Basically there's a family where a little boy ...   negative
4    Petter Mattei's "Love in the Time of Money" is...   positive
```

```
pos=df[df['sentiment']== 'positive']

neg=df[df['sentiment']== 'negative']

print(len(neg), len(pos))
```



```
25000 25000
```

```
files_pos=pos[0:1000]
files_neg=neg[0:1000]
print("length of files_pos",len(files_pos))
print("length of files_neg",len(files_neg))
```

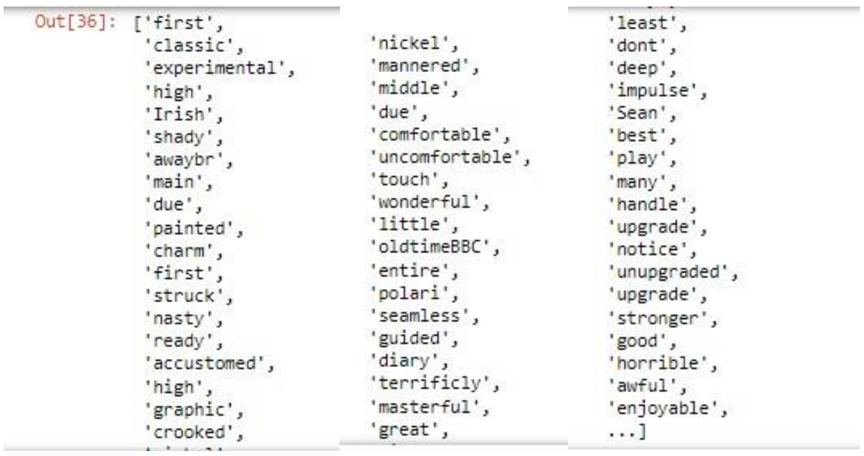


```
length of files_pos 1000
length of files_neg 1000
```

```
all_words = []
documents=[]
cleaned = []
from nltk.corpus import stopwords
stop_words = list(set(stopwords.words('english')))
allowed_word_types =["J"]
count = 0
```

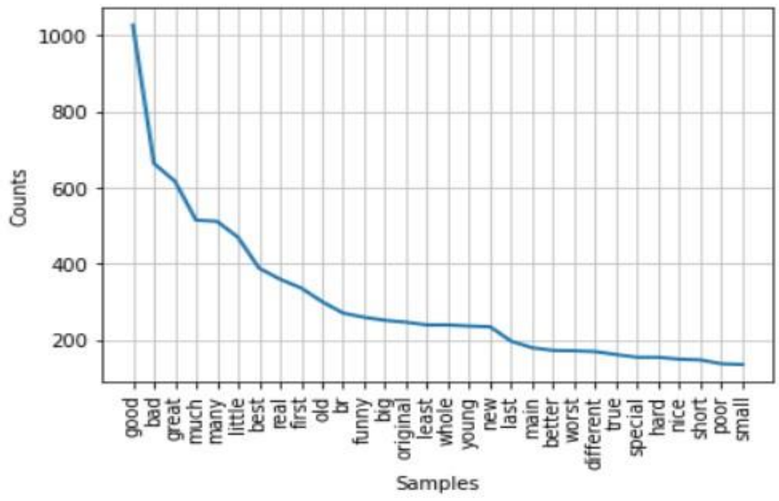
```
for p in files_pos['review']:
    documents.append((p, "pos"))
    #remove punctuation
    cleaned = re.sub(r'[^(a-zA-Z)\s]','', p)
    #tokenize
    tokenized = word_tokenize(cleaned)
    #remove stopwords
    stopped=[w for w in tokenized if not w in stop_words]
    #parts of speech tagging for each word
    pos=nltk.pos_tag(stopped)
    for w in pos:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0])


for p in files_neg['review']:
    documents.append((p, "neg"))
    #remove punctuation
    cleaned = re.sub(r'[^(a-zA-Z)\s]','', p)
    #tokenize
    tokenized = word_tokenize(cleaned)
    #remove stopwords
    stopped=[w for w in tokenized if not w in stop_words]
    #parts of speech tagging for each word
    pos=nltk.pos_tag(stopped)
    for w in pos:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0])
```

```
all_words
```



```
Out[36]: ['first',
          'classic',
          'experimental',
          'high',
          'Irish',
          'shady',
          'awaybr',
          'main',
          'due',
          'painted',
          'charm',
          'first',
          'struck',
          'nasty',
          'ready',
          'accustomed',
          'high',
          'graphic',
          'crooked',

          'nickel',
          'mannered',
          'middle',
          'due',
          'comfortable',
          'uncomfortable',
          'touch',
          'wonderful',
          'little',
          'oldtimeBBC',
          'entire',
          'polari',
          'seamless',
          'guided',
          'diary',
          'terrificly',
          'masterful',
          'great',

          'least',
          'dont',
          'deep',
          'impulse',
          'Sean',
          'best',
          'play',
          'many',
          'handle',
          'upgrade',
          'notice',
          'unupgraded',
          'upgrade',
          'stronger',
          'good',
          'horrible',
          'awful',
          'enjoyable',
          ...]
```

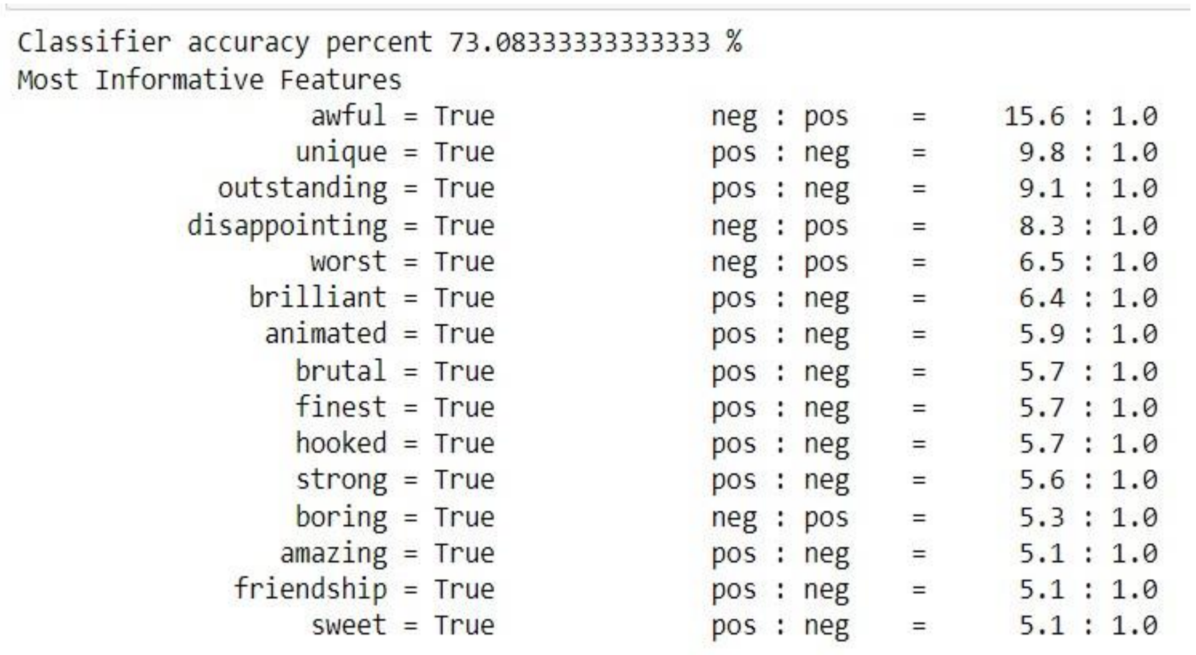
```
#freq of words
freq = nltk.FreqDist(all_words)
import matplotlib.pyplot as plt
freq.plot(30, cumulative = False)
plt.show()
```



```
#listing the 1000 most freqeunt words
word_features = list(freq.keys())[:1000]
word_features[10]
```

```
Out[38]: 'charm'
```

```
def find_features(document):
    words = word_tokenize(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    return features


featuresets = [(find_features(rev), category) for (rev, category) in
documents]


random.shuffle(featuresets)
training_set = featuresets[:800]
testing_set = featuresets[800:]



classifier = nltk.NaiveBayesClassifier.train(training_set)
print('Classifier accuracy percent', (nltk.classify.accuracy(classifier,
testing_set))* 100, '%')
classifier.show_most_informative_features(15)
```



```
Classifier accuracy percent 73.08333333333333 %
Most Informative Features
                   awful = True              neg : pos    =     15.6 : 1.0
                  unique = True              pos : neg    =      9.8 : 1.0
             outstanding = True              pos : neg    =      9.1 : 1.0
           disappointing = True              neg : pos    =      8.3 : 1.0
                   worst = True              neg : pos    =      6.5 : 1.0
               brilliant = True              pos : neg    =      6.4 : 1.0
                animated = True              pos : neg    =      5.9 : 1.0
                  brutal = True              pos : neg    =      5.7 : 1.0
                  finest = True              pos : neg    =      5.7 : 1.0
                  hooked = True              pos : neg    =      5.7 : 1.0
                  strong = True              pos : neg    =      5.6 : 1.0
                  boring = True              neg : pos    =      5.3 : 1.0
                 amazing = True              pos : neg    =      5.1 : 1.0
              friendship = True              pos : neg    =      5.1 : 1.0
                   sweet = True              pos : neg    =      5.1 : 1.0
```

Practical 7

AIM :- Practical to implement Sentiment Analysis using MultinomialNB, BernoulliNB

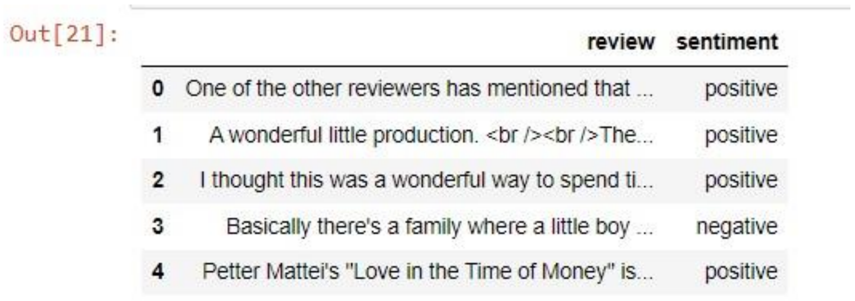
THEORY :-

MultiNomial NB: It should be used for the features with discrete values like word count 1,2,3... 3.

Bernoulli NB: It should be used for features with binary or boolean values like True/False or 0/1

CODE / OUTPUT :-

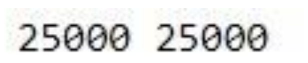
```
import re

import nltk

import random

from nltk.tokenize import word_tokenize

import pandas as pd

from sklearn.naive_bayes import MultinomialNB, BernoulliNB

from nltk.classify.scikitlearn import SklearnClassifier


df=pd.read_csv("./IMDB Dataset.csv")


df.head()
```



Out[21]:

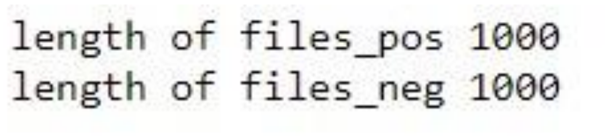
	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
pos=df[df['sentiment']== 'positive']

neg=df[df['sentiment']== 'negative']

print(len(neg), len(pos))
```



```
25000 25000
```

```
files_pos=pos[0:1000]
files_neg=neg[0:1000]
print("length of files_pos",len(files_pos))
print("length of files_neg",len(files_neg))
```

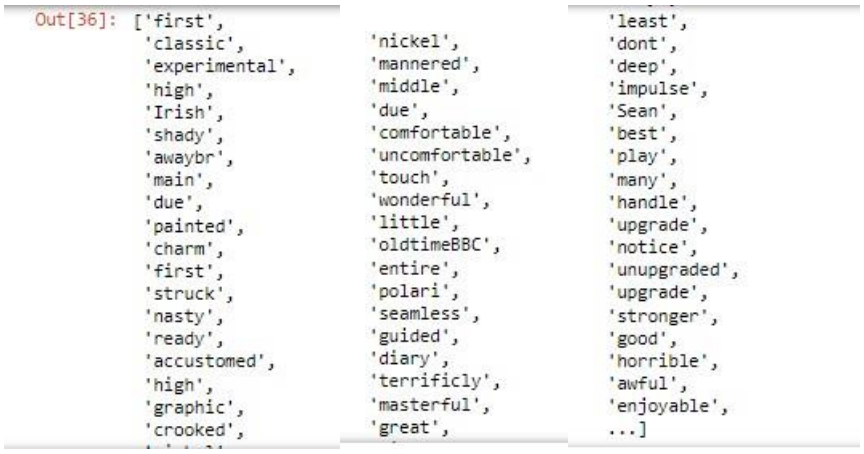


```
length of files_pos 1000
length of files_neg 1000
```

```
all_words = []
documents=[]
cleaned = []
from nltk.corpus import stopwords

stop_words = list(set(stopwords.words('english')))
allowed_word_types =["J"]
count = 0
```

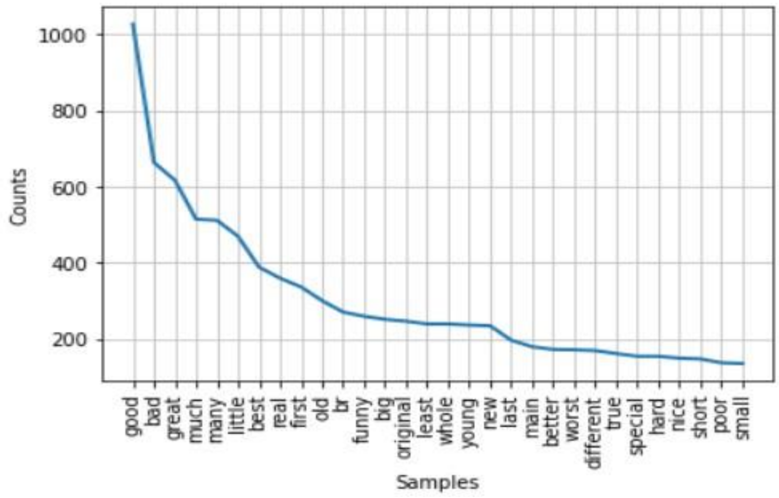
```
for p in files_pos['review']:
    documents.append((p, "pos"))
    #remove punctuation
    cleaned = re.sub(r'[^(a-zA-Z)\s]','', p)
    #tokenize
    tokenized = word_tokenize(cleaned)
    #remove stopwords
    stopped=[w for w in tokenized if not w in stop_words]
    #parts of speech tagging for each word
    pos=nltk.pos_tag(stopped)
    for w in pos:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0])


for p in files_neg['review']:
    documents.append((p, "neg"))
    #remove punctuation
    cleaned = re.sub(r'[^(a-zA-Z)\s]','', p)
    #tokenize
    tokenized = word_tokenize(cleaned)
    #remove stopwords
    stopped=[w for w in tokenized if not w in stop_words]
    #parts of speech tagging for each word
    pos=nltk.pos_tag(stopped)
    for w in pos:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0])
```

```
all_words
```



```
Out[36]: ['first',
          'classic',
          'experimental',
          'high',
          'Irish',
          'shady',
          'awaybr',
          'main',
          'due',
          'painted',
          'charm',
          'first',
          'struck',
          'nasty',
          'ready',
          'accustomed',
          'high',
          'graphic',
          'crooked',

          'nickel',
          'mannered',
          'middle',
          'due',
          'comfortable',
          'uncomfortable',
          'touch',
          'wonderful',
          'little',
          'oldtimeBBC',
          'entire',
          'polari',
          'seamless',
          'guided',
          'diary',
          'terrificly',
          'masterful',
          'great',

          'least',
          'dont',
          'deep',
          'impulse',
          'Sean',
          'best',
          'play',
          'many',
          'handle',
          'upgrade',
          'notice',
          'unupgraded',
          'upgrade',
          'stronger',
          'good',
          'horrible',
          'awful',
          'enjoyable',
          ...]
```

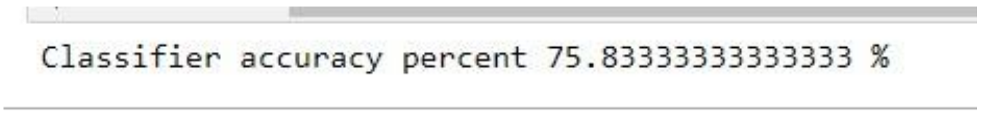
```
#freq of words

freq = nltk.FreqDist(all_words)

import matplotlib.pyplot as plt

freq.plot(30, cumulative = False)

plt.show()
```



```
#listing the 1000 most freqeunt words

word_features = list(freq.keys())[:1000]

word_features[10]
```

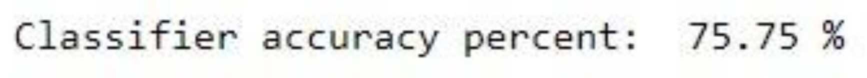
```
Out[38]: 'charm'
```

```
def find_features(document):
    words = word_tokenize(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    return features


featuresets = [(find_features(rev), category) for (rev, category) in
documents]


random.shuffle(featuresets)
training_set = featuresets[:800]
testing_set = featuresets[800:]



MNB_clf = SklearnClassifier(MultinomialNB())
mnb_cls = MNB_clf.train(training_set)
print('Classifier accuracy percent', (nltk.classify.accuracy(mnb_cls,
testing_set)) * 100, '%')
```



```
Classifier accuracy percent 75.83333333333333 %
```

```
MNB_clf = SklearnClassifier(BernoulliNB())
bnb_cls = MNB_clf.train(training_set)
print('Classifier accuracy percent: ', (nltk.classify.accuracy(bnb_cls,
testing_set))*100, '%')
```



```
Classifier accuracy percent:  75.75 %
```

Practical 8

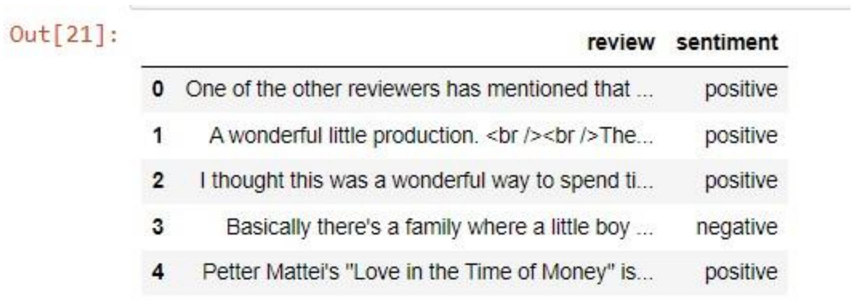
AIM :- Practical to implement Sentiment Analysis using SGDClassifier

THEORY : -

SGDClassifier supports multi-class classification by combining multiple binary classifiers in a “one versus all” (OVA) scheme. For each of the classes, a binary classifier is learned that discriminates between that and all other classes.

CODE / OUTPUT : -

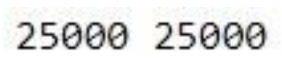
```
import re

import nltk

import random

from nltk.tokenize import word_tokenize

import pandas as pd

from sklearn.linear_model import SGDClassifier

from nltk.classify.scikitlearn import SklearnClassifier


df=pd.read_csv("./IMDB Dataset.csv")


df.head()
```



Out[21]:

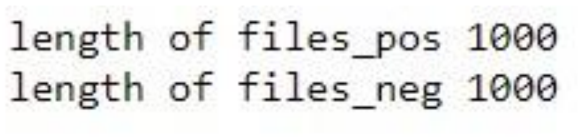
	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. <br /><br />The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
pos=df[df['sentiment']== 'positive']

neg=df[df['sentiment']== 'negative']

print(len(neg), len(pos))
```



```
25000 25000
```

```
files_pos=pos[0:1000]
files_neg=neg[0:1000]
print("length of files_pos",len(files_pos))
print("length of files_neg",len(files_neg))
```

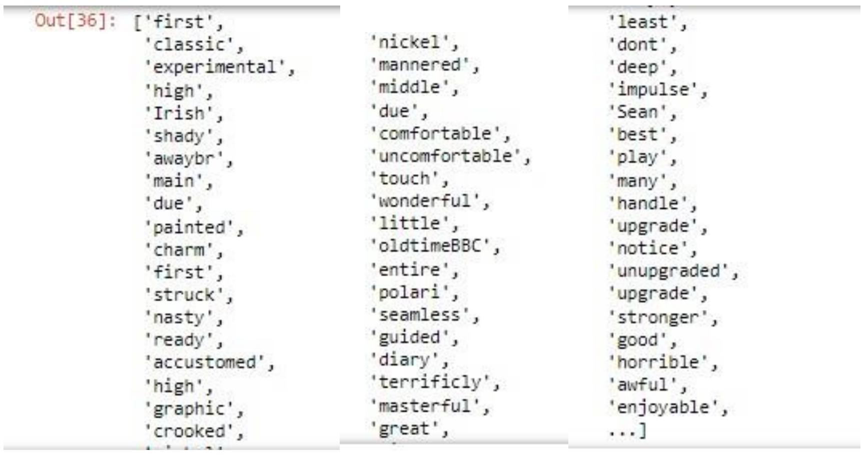


```
length of files_pos 1000
length of files_neg 1000
```

```
all_words = []
documents=[]
cleaned = []
from nltk.corpus import stopwords

stop_words = list(set(stopwords.words('english')))
allowed_word_types =["J"]
count = 0
```

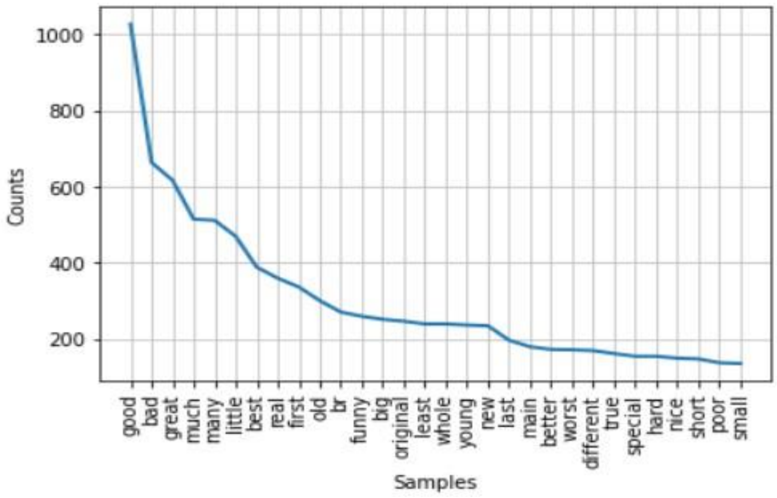
```
for p in files_pos['review']:
    documents.append((p, "pos"))
    #remove punctuation
    cleaned = re.sub(r'[^(a-zA-Z)\s]','', p)
    #tokenize
    tokenized = word_tokenize(cleaned)
    #remove stopwords
    stopped=[w for w in tokenized if not w in stop_words]
    #parts of speech tagging for each word
    pos=nltk.pos_tag(stopped)
    for w in pos:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0])


for p in files_neg['review']:
    documents.append((p, "neg"))
    #remove punctuation
    cleaned = re.sub(r'[^(a-zA-Z)\s]','', p)
    #tokenize
    tokenized = word_tokenize(cleaned)
    #remove stopwords
    stopped=[w for w in tokenized if not w in stop_words]
    #parts of speech tagging for each word
    pos=nltk.pos_tag(stopped)
    for w in pos:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0])
```

```
all_words
```



```
Out[36]: ['first',
          'classic',
          'experimental',
          'high',
          'Irish',
          'shady',
          'awaybr',
          'main',
          'due',
          'painted',
          'charm',
          'first',
          'struck',
          'nasty',
          'ready',
          'accustomed',
          'high',
          'graphic',
          'crooked',

          'nickel',
          'mannered',
          'middle',
          'due',
          'comfortable',
          'uncomfortable',
          'touch',
          'wonderful',
          'little',
          'oldtimeBBC',
          'entire',
          'polari',
          'seamless',
          'guided',
          'diary',
          'terrificly',
          'masterful',
          'great',

          'least',
          'dont',
          'deep',
          'impulse',
          'Sean',
          'best',
          'play',
          'many',
          'handle',
          'upgrade',
          'notice',
          'unupgraded',
          'upgrade',
          'stronger',
          'good',
          'horrible',
          'awful',
          'enjoyable',
          ...]
```

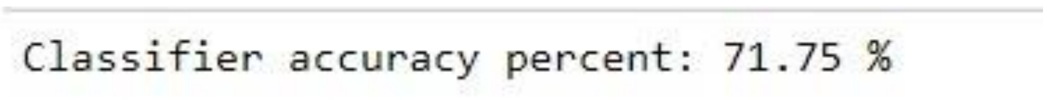
```
#freq of words
freq = nltk.FreqDist(all_words)
import matplotlib.pyplot as plt
freq.plot(30, cumulative = False)
plt.show()
```



```
#listing the 1000 most freqeunt words
word_features = list(freq.keys())[:1000]
word_features[10]
```

```
Out[38]: 'charm'
```

```
def find_features(document):
    words = word_tokenize(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    return features


featuresets = [(find_features(rev), category) for (rev, category) in
documents]


random.shuffle(featuresets)
training_set = featuresets[:800]
testing_set = featuresets[800:]




SGD_clf = SklearnClassifier(SGDClassifier())
sgd_cls = SGD_clf.train(training_set)
print("Classifier accuracy percent:", nltk.classify.accuracy(sgd_cls,
testing_set) * 100, "%")
```



```
Classifier accuracy percent: 71.75 %
```

Practical 9

AIM :- Practical to implement Sentiment Analysis using LogisticRegression, SVC

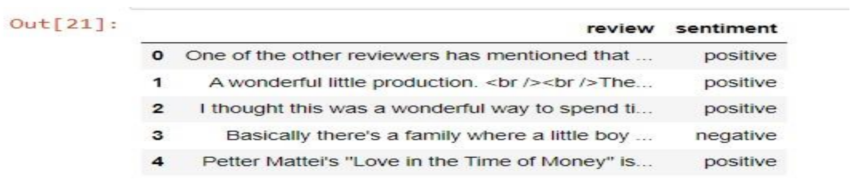
THEORY :-

Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on.

The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.

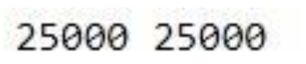
CODE / OUTPUT :-

```
import re

import nltk

import random

from nltk.tokenize import word_tokenize

import pandas as pd

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from nltk.classify.scikitlearn import SklearnClassifier


df=pd.read_csv("./IMDB Dataset.csv")


df.head()
```



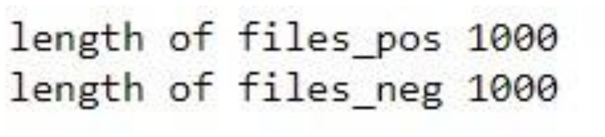
```
Out[21]:
                                              review  sentiment
0  One of the other reviewers has mentioned that ...   positive
1     A wonderful little production. <br /><br />The...   positive
2  I thought this was a wonderful way to spend ti...   positive
3      Basically there's a family where a little boy ...   negative
4  Petter Mattei's "Love in the Time of Money" is...   positive
```

```
pos=df[df['sentiment']== 'positive']

neg=df[df['sentiment']== 'negative']

print(len(neg), len(pos))
```



```
25000 25000
```

```
files_pos=pos[0:1000]
files_neg=neg[0:1000]
print("length of files_pos",len(files_pos))
print("length of files_neg",len(files_neg))
```

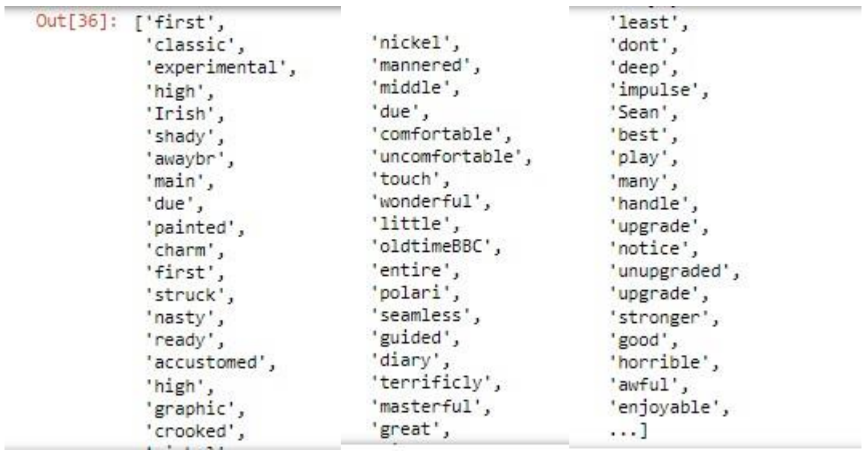


```
length of files_pos 1000
length of files_neg 1000
```

```
all_words = []
documents=[]
cleaned = []
from nltk.corpus import stopwords

stop_words = list(set(stopwords.words('english')))
allowed_word_types =["J"]
count = 0
```

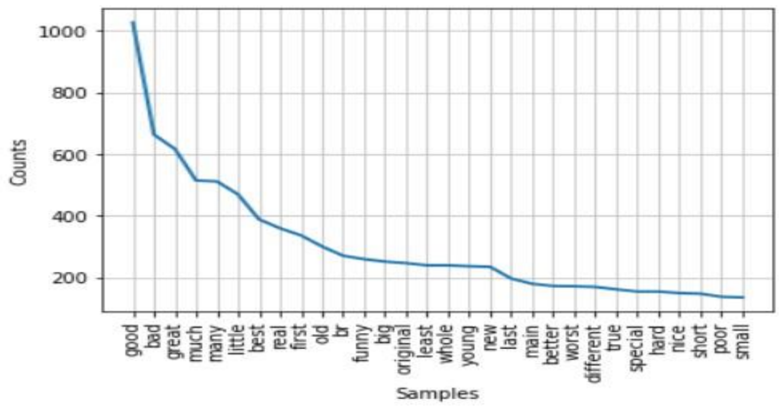
```
for p in files_pos['review']:

    documents.append((p, "pos"))

    #remove punctuation

    cleaned = re.sub(r'[^(a-zA-Z)\s]','', p)

    #tokenize

    tokenized = word_tokenize(cleaned)

    #remove stopwords

    stopped=[w for w in tokenized if not w in stop_words]

    #parts of speech tagging for each word

    pos=nltk.pos_tag(stopped)

    for w in pos:

        if w[1][0] in allowed_word_types:

            all_words.append(w[0])



for p in files_neg['review']:

    documents.append((p, "neg"))

    #remove punctuation

    cleaned = re.sub(r'[^(a-zA-Z)\s]','', p)

    #tokenize

    tokenized = word_tokenize(cleaned)

    #remove stopwords

    stopped=[w for w in tokenized if not w in stop_words]

    #parts of speech tagging for each word

    pos=nltk.pos_tag(stopped)

    for w in pos:

        if w[1][0] in allowed_word_types:

            all_words.append(w[0])
```

```
all_words
```



```
Out[36]: ['first',
          'classic',
          'experimental',
          'high',
          'Irish',
          'shady',
          'awaybr',
          'main',
          'due',
          'painted',
          'charm',
          'first',
          'struck',
          'nasty',
          'ready',
          'accustomed',
          'high',
          'graphic',
          'crooked',

          'nickel',
          'mannered',
          'middle',
          'due',
          'comfortable',
          'uncomfortable',
          'touch',
          'wonderful',
          'little',
          'oldtimeBBC',
          'entire',
          'polari',
          'seamless',
          'guided',
          'diary',
          'terrificly',
          'masterful',
          'great',

          'least',
          'dont',
          'deep',
          'impulse',
          'Sean',
          'best',
          'play',
          'many',
          'handle',
          'upgrade',
          'notice',
          'unupgraded',
          'upgrade',
          'stronger',
          'good',
          'horrible',
          'awful',
          'enjoyable',
          ...]
```

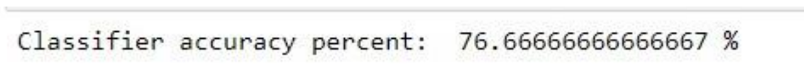
```
#freq of words
freq = nltk.FreqDist(all_words)
import matplotlib.pyplot as plt
freq.plot(30, cumulative = False)
plt.show()
```



```
#listing the 1000 most freqeunt words
word_features = list(freq.keys())[:1000]
word_features[10]
```

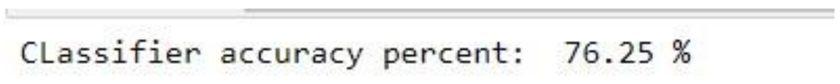
```
Out[38]: 'charm'
```

```
def find_features(document):
    words = word_tokenize(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    return features


featuresets = [(find_features(rev), category) for (rev, category) in
documents]


random.shuffle(featuresets)
training_set = featuresets[:800]
testing_set = featuresets[800:]


LogReg_clf = SklearnClassifier(LogisticRegression())
log_cls = LogReg_clf.train(training_set)
print("Classifier accuracy percent: ", nltk.classify.accuracy(log_cls,
testing_set)*100, '%')
```



```
Classifier accuracy percent:  76.66666666666667 %
```

```
SVC_clf = SklearnClassifier(SVC())
svc_cls = SVC_clf.train(training_set)
print("CLassifier accuracy percent: ", nltk.classify.accuracy(svc_cls,
testing_set)*100, '%')
```



```
CLassifier accuracy percent:  76.25 %
```