Gyan Ganga Institute of Technology and Sciences, Jabalpur

Computer Science & Engineering

Natural Language Processing

(AL – 504 (B))

PRACTICAL FILE

NAME:

_____

ENROLLMENT NUMBER:

_____

BRANCH: CSE-AIML

SESSION: 2023-24

Submitted to:

Dr Saurabh Tewari

Assistant Professor

INDEX

S.No	TITLE	DATE	GRADE	Teachers Sign.
1.	Tutorial for introduction to NLTK toolkit	07-08-2023		
2.	Example using NLTK for preprocessing text	21-08-2023		
3.	Preprocessing of text (Tokenization, Filtration, Script Validation, Stop Word Removal , Stemming	28-08-2023		
4.	Morphological Analysis	04-09-2023		
5.	N-gram Model	11-09-2023		
6.	POS Tagging	18-09-2023		
7.	Chunking	09-10-2023		
8.	Named Entity Recognition	16-10-2023		
9.	Finite State Automata	30-10-2023		
10	Study of PorterStemmer, LancasterStemmer, RegexpStemmer, snowball etc.	06-11-2023		

Experiment 1

Aim:-Tutorial for introduction to NLTK toolkit

Double-click (or enter) to edit

```
pip install --user -U nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.1)
```

```
import nltk
nltk.download()
```

```
NLTK Downloader
---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> l
Packages:
  [ ] abc..................... Australian Broadcasting Commission 2006
  [ ] alpino.................. Alpino Dutch Treebank
  [ ] averaged_perceptron_tagger Averaged Perceptron Tagger
  [ ] averaged_perceptron_tagger_ru Averaged Perceptron Tagger (Russian)
  [ ] basque_grammars...... Grammars for Basque
  [ ] bcp47................ BCP-47 Language Tags
  [ ] biocreative_ppi...... BioCreAtIvE (Critical Assessment of Information
                            Extraction Systems in Biology)
  [ ] bllip_wsj_no_aux.... BLLIP Parser: WSJ Model
  [ ] book_grammars........ Grammars from NLTK Book
  [ ] brown................ Brown Corpus
  [ ] brown_tei............ Brown Corpus (TEI XML Version)
  [ ] cess_cat............. CESS-CAT Treebank
  [ ] cess_esp............. CESS-ESP Treebank
  [ ] chat80............... Chat-80 Data Files
  [ ] city_database........ City Database
  [ ] cmudict.............. The Carnegie Mellon Pronouncing Dictionary (0.6)
  [ ] comparative_sentences Comparative Sentence Dataset
  [ ] comtrans............. ComTrans Corpus Sample
  [ ] conll2000............ CONLL 2000 Chunking Corpus
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-7-3b02390c7677> in <cell line: 2>()
      1 import nltk
----> 2 nltk.download()

                              6 frames
/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in _input_request(self, prompt, ident, parent,
password)
    893             except KeyboardInterrupt:
    894                 # re-raise KeyboardInterrupt, to truncate traceback
--> 895                 raise KeyboardInterrupt("Interrupted by user") from None
    896             except Exception as e:
    897                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

SEARCH STACK OVERFLOW

```
import nltk
pip install --user -U nltk
```

```
# import all the resources for Natural Language Processing with Python
nltk.download("book")
```

```
[nltk_data] Downloading collection 'book'
[nltk_data]    |
[nltk_data]    | Downloading package abc to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/abc.zip.
[nltk_data]    | Downloading package brown to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/brown.zip.
[nltk_data]    | Downloading package chat80 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/chat80.zip.
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
```

```
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package conll2000 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2000.zip.
[nltk_data]    | Downloading package conll2002 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/conll2002.zip.
[nltk_data]    | Downloading package dependency_treebank to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/dependency_treebank.zip.
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/genesis.zip.
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gutenberg.zip.
[nltk_data]    | Downloading package ieer to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/ieer.zip.
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/inaugural.zip.
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/movie_reviews.zip.
[nltk_data]    | Downloading package nps_chat to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/nps_chat.zip.
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/names.zip.
[nltk_data]    | Downloading package ppattach to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/ppattach.zip.
[nltk_data]    | Downloading package reuters to /root/nltk_data...
[nltk_data]    | Downloading package senseval to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/senseval.zip.
[nltk_data]    | Downloading package state_union to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/state_union.zip.
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/stopwords.zip.
[nltk_data]    | Downloading package swadesh to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/swadesh.zip.
[nltk_data]    | Downloading package timit to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/timit.zip.
[nltk_data]    | Downloading package treebank to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/treebank.zip.
[nltk_data]    | Downloading package toolbox to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/toolbox.zip.
[nltk_data]    | Downloading package udhr to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/udhr.zip.
[nltk_data]    | Downloading package udhr2 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/udhr2.zip.
[nltk_data]    | Downloading package unicode_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/unicode_samples.zip.
[nltk_data]    | Downloading package webtext to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/webtext.zip.
```

Take a sentence and tokenize into words. Then apply a part-of-speech tagger

```
sentence = """At eight o'clock on Thursday morning
Arthur didn't feel very good."""
tokens = nltk.word_tokenize(sentence)
print(tokens)
tagged = nltk.pos_tag(tokens)
print(tagged)
```

```
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'NN'), ('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN'), ('Arthur', 'NNP
```

From the tagged words, identify the proper names.

```
entities = nltk.chunk.ne_chunk(tagged)
print(entities)
```

```
(S
  At/IN
  eight/CD
  o'clock/NN
  on/IN
  Thursday/NNP
  morning/NN
  (PERSON Arthur/NNP)
  did/VBD
  n't/RB
  feel/VB
  very/RB
  good/JJ
  ./.)
```

Get texts for corpus analysis

```
%matplotlib inline
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

Generate a key-word in context concordance

```
text1.concordance("monstrous")
#There are many ways to examine the context of a text apart from simply reading it.
#A concordance view shows us every occurrence of a given word, together with some context.
#Here we look up the word monstrous in Moby Dick by entering text1 followed by a period, then the term concordance, and t
```

```
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
```

Find words with similar concordance to a given word

```
print(text1)
text1.similar("monstrous")
print(text2)
text2.similar("monstrous")
#A concordance permits us to see words in context. For example, we saw that monstrous occurred in contexts such as the __
#What other words appear in a similar range of contexts?
#We can find out by appending the term similar to the name of the text in question, then inserting the relevant word in p
```

```
<Text: Moby Dick by Herman Melville 1851>
true contemptible christian abundant few part mean careful puzzled
mystifying passing curious loving wise doleful gamesome singular
delightfully perilous fearless
<Text: Sense and Sensibility by Jane Austen 1811>
very so exceedingly heartily a as good great extremely remarkably
sweet vast amazingly
```

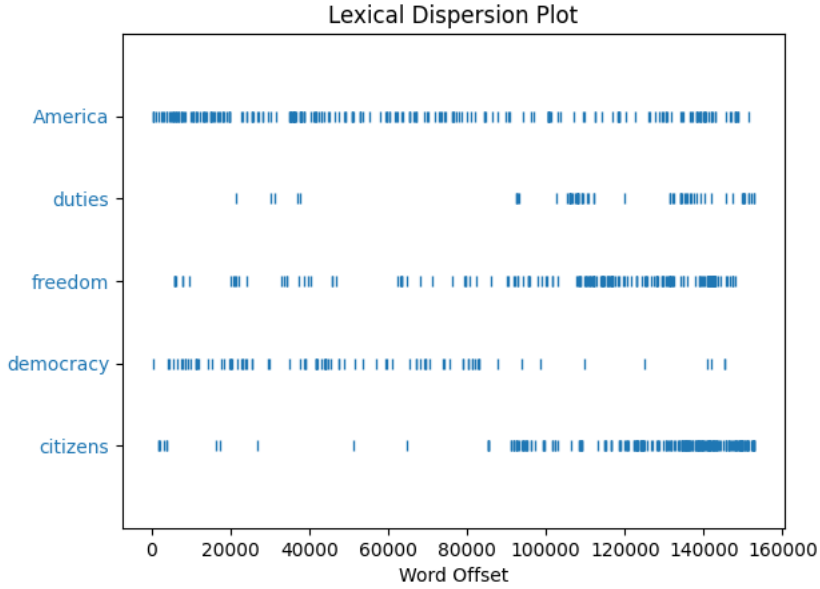
Find contexts which are similar for the given words

```
text2.common_contexts(["monstrous", "very"])
```

```
am_glad a_pretty a_lucky is_pretty be_glad
```

Plot where in the text certain words appear

```
text4.dispersion_plot(["citizens", "democracy", "freedom", "duties", "America"])
#Lexical dispersion is a measure of how frequently a word appears across the parts of a corpus.
#This plot notes the occurrences of a word and how many words from the beginning of the corpus it appears (word offsets).
#This is particularly useful for a corpus that covers a longer time period and for which you want to analyse how specific
```



Print the identity of a text, the length of the text and its vocabulary

```
print(text3)
print(len(text3))
print(sorted(set(text3)))
```

```
<Text: The Book of Genesis>
44764
['!', "'", '(', ')', ',', ',)', '.', '.)', ':', ';', ';)', '?', '?)', 'A', 'Abel', 'Abelmizraim', 'Abidah', 'Abide', 'Ab
```

```
text3.generate()
#let's try generating some random text in the various styles we have just seen. To do this, we type the name of the text
```

```
Building ngram index...
laid by her , and said unto Cain , Where art thou , and said , Go to ,
I will not do it for ten ' s sons ; we dreamed each man according to
their generatio the firstborn said unto Laban , Because I said , Nay ,
but Sarah shall her name be . , duke Elah , duke Shobal , and Akan .
and looked upon my affliction . Bashemath Ishmael ' s blood , but Isra
for as a prince hast thou found of all the cattle in the valley , and
the wo The
'laid by her , and said unto Cain , Where art thou , and said , Go to ,\nI wi
ll not do it for ten ' s sons ; we dreamed each man according to\ntheir gener
atio the firstborn said unto Laban , Because I said , Nay ,\nbut Sarah shall
her name be . , duke Elah , duke Shobal , and Akan .\nand looked upon my affl
```

```
len(text3)
#So Genesis has 44,764 words and punctuation symbols,
#or "tokens." A token is the technical name for a sequence of characters — such as hairy, his, or :) — that we want to tr
```

```
44764
```

```
print(sorted(set(text3)))
#By wrapping sorted() around the Python expression set(text3) [1],
# we obtain a sorted list of vocabulary items, beginning with various punctuation symbols and continuing with words start
#Although it has 44,764 tokens, this book has only 2,789 distinct words, or "word types." A word type is the form or spel
# that is, the word considered as a unique item of vocabulary. Our count of 2,789 items will include punctuation symbols,

```

```
['!', "'", '(', ')', ',', ',)', '.', '.)', ':', ';', ';)', '?', '?)', 'A', 'Abel', 'Abelmizraim', 'Abidah', 'Abide', 'Ab
```

Print some statistics of word occurrence in the text

```
def lexical_diversity(text):
    return len(set(text)) / len(text)
# Lexical richness refers to the range and variety of vocabulary deployed in a text by a speaker/write
def percentage(count, total):
    return 100 * count / total
print(lexical_diversity(text3))
print(lexical_diversity(text5))
print(percentage(text4.count('a'), len(text4)))
#The most common approach to measuring lexical richness or diversity is based on the ratio of different words (types) to
```

```
0.06230453042623537
0.13477005109975562
1.457806031353621
```

Experiment 2

AIM:-Example using NLTK for preprocessing text

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

```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

```
import pandas as pd
import matplotlib.pyplot as plt
import io
import unicodedata
import numpy as np
import re
import string
```

```
#Constants
# POS (Parts Of Speech) for: nouns, adjectives, verbs and adverbs
DI_POS_TYPES = {'NN':'n', 'JJ':'a', 'VB':'v', 'RB':'r'}
POS_TYPES = list(DI_POS_TYPES.keys())
```

```
# Constraints on tokens
MIN_STR_LEN = 3
RE_VALID = '[a-zA-Z]'
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Upload from google drive
from google.colab import files
uploaded = files.upload()
print("len(uploaded.keys():", len(uploaded.keys()))
```

Choose files quotes.txt

- **quotes.txt**(text/plain) - 6935 bytes, last modified: 23/11/2023 - 100% done

```
Saving quotes.txt to quotes.txt
len(uploaded.keys(): 1
```

```
for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(name=fn, length=len(uploaded[fn])))
```

```
User uploaded file "quotes.txt" with length 6935 bytes
```

```
# Get list of quotes
df_quotes = pd.read_csv(io.StringIO(uploaded['quotes.txt'].decode('utf-8')), sep='\t')
```

```
# Display
print("df_quotes:")
print(df_quotes.head().to_string())
print(df_quotes.describe())
```

```
df_quotes:
            Author
0  Agatha Christie  I like living. I have sometimes been wildly, despairingly, acutely miserable, racked with sorrow; bu
```

```
1  Agatha Christie                                                                The impossible could n
2  Agatha Christie
3  Agatha Christie
4  Agatha Christie
                    Author                                          Quote
count                   46                                             46
unique                   5                                             46
top     Arthur Conan Doyle  I like living. I have sometimes been wildly, d...
freq                    14                                              1
```

```
# Convert quotes to list
li_quotes = df_quotes['Quote'].tolist()
print()
print("len(li_quotes):", len(li_quotes))
```

```
len(li_quotes): 46
```

Tokenization

```
# Get stopwords, stemmer and lemmatizer
stopwords = nltk.corpus.stopwords.words('english')
stemmer = nltk.stem.PorterStemmer()
lemmatizer = nltk.stem.WordNetLemmatizer()
```

```
# Remove accents function
def remove_accents(data):
    return ''.join(x for x in unicodedata.normalize('NFKD', data) if x in string.ascii_letters or x == " ")
# Process all quotes
li_tokens = []
li_token_lists = []
li_lem_strings = []
```

```
1
```

```
for i,text in enumerate(li_quotes):
# Tokenize by sentence, then by lowercase word
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
# Process all tokens per quote
li_tokens_quote = []
li_tokens_quote_lem = []
for token in tokens:
# Remove accents
    t = remove_accents(token)
    # Remove punctuation
    t = str(t).translate(string.punctuation)
    li_tokens_quote.append(t)
    # Add token that represents "no lemmatization match"
    li_tokens_quote_lem.append("-") # this token will be removed if a lemmatization match is found below
    # Process each token
    if t not in stopwords:
        if re.search(RE_VALID, t):
            if len(t) >= MIN_STR_LEN:
            # Note that the POS (Part Of Speech) is necessary as input to the lemmatizer # (otherwise it assumes the worc
                pos = nltk.pos_tag([t])[0][1][:2]
                pos2 = 'n' # set default to noun
                if pos in DI_POS_TYPES:
                    pos2 = DI_POS_TYPES[pos]

                stem = stemmer.stem(t)
                lem = lemmatizer.lemmatize(t, pos=pos2)
                # lemmatize with the correct POS
                if pos in POS_TYPES:
                    li_tokens.append((t, stem, lem, pos))
                    # Remove the "-" token and append the lemmatization match
                    li_tokens_quote_lem = li_tokens_quote_lem[:-1]
                    li_tokens_quote_lem.append(lem)
```

```
# Build list of token lists from lemmatized tokens
li_token_lists.append(li_tokens_quote)
# Build list of strings from lemmatized tokens
str_li_tokens_quote_lem = ' '.join(li_tokens_quote_lem)
li_lem_strings.append(str_li_tokens_quote_lem)
```

```
# Build resulting dataframes from lists
df_token_lists = pd.DataFrame(li_token_lists)
print("df_token_lists.head(5):")
print(df_token_lists.head(5).to_string())
# Replace None with empty string
for c in df_token_lists:
    if str(df_token_lists[c].dtype) in ('object', 'string_', 'unicode_'):
        df_token_lists[c].fillna(value='', inplace=True)
df_lem_strings = pd.DataFrame(li_lem_strings, columns=['lem quote'])
print()
print("")
print("df_lem_strings.head():")
print(df_lem_strings.head().to_string())
```

```
df_token_lists.head(5):
  0         1       2    3   4       5    6      7   8   9     10      11     12   13       14   15     16   17     18
0  to  sherlock  holmes  she  is  always  the  woman       i  have  seldom  heard  him  mention  her  under  any  other


df_lem_strings.head():

0  - sherlock holmes - - always - woman - - - seldom heard - mention - - - - name - - - eye - eclipse - predominates - w
```

Process results, and the most popular lemmatized words and group results by Part

```
# Add counts
print("Group by lemmatized words, add count and sort:")
df_all_words = pd.DataFrame(li_tokens, columns=['token', 'stem', 'lem', 'pos'])
df_all_words['counts'] = df_all_words.groupby(['lem'])['lem'].transform('count')
df_all_words = df_all_words.sort_values(by=['counts', 'lem'], ascending=[False, True]).reset_index()
print("Get just the first row in each lemmatized group")
df_words = df_all_words.groupby('lem').first().sort_values(by='counts', ascending=False).reset_index()
print("df_words.head(10):")
print(df_words.head(10))
```

```
Group by lemmatized words, add count and sort:
Get just the first row in each lemmatized group
df_words.head(10):
            lem  index         token          stem pos  counts
0         woman      3         woman         woman  NN       6
1         adler     18         adler         adler  NN       4
2         irene     17         irene          iren  NN       4
3       emotion     14       emotion          emot  NN       4
4        reason     29     reasoning        reason  VB       2
5       observe     30     observing        observ  VB       2
6  particularly     20  particularly  particularli  RB       2
7       perfect     28       perfect       perfect  NN       2
8       precise     23       precise        precis  NN       2
9  predominates     10  predominates      predomin  NN       2
```

Top 10 words per Part Of Speech (POS)

```
df_words = df_words[['lem', 'pos', 'counts']].head(200)
for v in POS_TYPES:
    df_pos = df_words[df_words['pos'] == v]
    print()
    print("POS_TYPE:", v)
    print(df_pos.head(10).to_string())
```

```
POS_TYPE: NN
             lem pos  counts
0          woman  NN       6
1          adler  NN       4
2          irene  NN       4
3        emotion  NN       4
7        perfect  NN       2
8        precise  NN       2
9   predominates  NN       2
11     abhorrent  NN       2
12          name  NN       2
13        seldom  NN       2

POS_TYPE: JJ
             lem pos  counts
10  questionable  JJ       2
17         whole  JJ       2
28       dubious  JJ       2

POS_TYPE: VB
        lem pos  counts
```

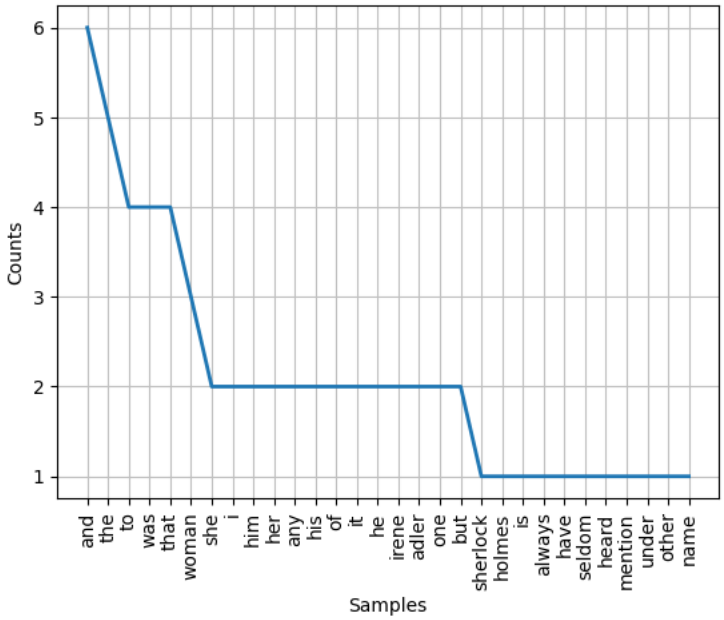
```
4    reason  VB       2
5   observe  VB       2
16     take  VB       2
19      see  VB       2
26  balance  VB       2

POS_TYPE: RB
              lem pos  counts
6   particularly  RB       2
23     admirably  RB       2
25        always  RB       2
34          late  RB       2
37           yet  RB       2
```

Sorted frequency plot for all words

```
li_token_lists_flat = [y for x in li_token_lists for y in x]
# flatten the list of token lists to a single list
print("li_token_lists_flat[:10]:", li_token_lists_flat[:10])
```

```
li_token_lists_flat[:10]: ['to', 'sherlock', 'holmes', 'she', 'is', 'always', 'the', 'woman', '', 'i']
```

```
di_freq = nltk.FreqDist(li_token_lists_flat)
del di_freq['']
li_freq_sorted = sorted(di_freq.items(), key=lambda x: x[1], reverse=True)
# sorted list print(li_freq_sorted)
di_freq.plot(30, cumulative=False)
```

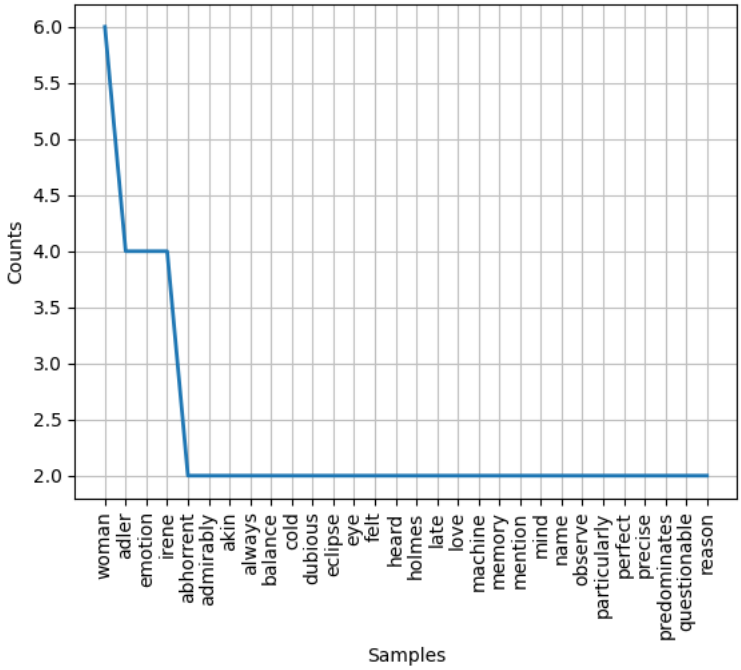



```
<Axes: xlabel='Samples', ylabel='Counts'>
```

Sorted frequency plot for Lemmatized words after removing stopwords

```
li_lem_words = df_all_words['lem'].tolist()
di_freq2 = nltk.FreqDist(li_lem_words)
li_freq_sorted2 = sorted(di_freq2.items(), key=lambda x: x[1], reverse=True)
```

```
# sorted list print(li_freq_sorted2 SK)
di_freq2.plot(30, cumulative=False)
```




```
<Axes: xlabel='Samples', ylabel='Counts'>
```

Experiment 3

Aim:-Preprocessing of text (tokenization,Filtration, Script Validation, Stop Word Removal, Stemming)

Code:

String Handling

```
print(len("what it is what isnt"))
s= ["what", "it", "is", "what", "it","isnt"]
print(len(s))
x=sorted(s)
print(s)
print(x)
d=x+s
print(d)
```

```
20
6
['what', 'it', 'is', 'what', 'it', 'isnt']
['is', 'isnt', 'it', 'it', 'what', 'what']
['is', 'isnt', 'it', 'it', 'what', 'what', 'what', 'it', 'is', 'what', 'it', 'isnt']
```

File Handling (tokenization and Filtering)

```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving sk.txt to sk.txt
```

```
print("Output:-->")
for line in open("sk.txt"):
    for word in line.split():
        if word.endswith('ing' or 'sk'):

            print(word)
            print(len(word))
```

```
Output:-->
eating
6
cycling
7
```

Experiment 4

Aim:-To study Morphological Ananlysis

```
#Regular Expression code.

import re
input= "The 5 greatest batsmen in odi cricket 1 Sachin Tendulkar 2 Sir Vivian Richards 3 Brian Lara 4 Virat Kohli 5 Ricky
input= input.lower()
print(input,'\n')


result=(re.sub(r'\d+','',input))
print(result)
```

```
the 5 greatest batsmen in odi cricket 1 sachin tendulkar 2 sir vivian richards 3 brian lara 4 virat kohli 5 ricky pontin

the  greatest batsmen in odi cricket  sachin tendulkar  sir vivian richards  brian lara  virat kohli  ricky ponting. sk
```

```
#StopWord Removal
def punctuations(raw_review):
    text=raw_review
    text=text.replace("n't",'not')
    text=text.replace("'s",'is')
    text=text.replace("'re",'are')
    text=text.replace("'ve",'have')
    text=text.replace("'m",'am')
    text=text.replace("'d",'would')
    text=text.replace("'ll",'will')
    text=text.replace("in",'ing')
     #import re
    letters_only=re.sub("[^a-z,A-Z]"," ",text)
    return("".join(letters_only))
t="how's  my  team  SK  doin , you're  supposed  to  be  not  loosin"
p=punctuations(t)
print(p)
```

```
howis  my  team  SK  doing , youare  supposed  to  be  not  loosing
```

```

#synonym
import nltk
nltk.download('wordnet')


```

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

```
from nltk.corpus import wordnet
```

```
synonyms=[]
for syn in wordnet.synsets("machine"):
    for lemma in syn.lemmas():
        synonyms.append(lemma.name())
        print(synonyms)
```

```
['machine']
['machine', 'machine']
['machine', 'machine', 'machine']
['machine', 'machine', 'machine', 'machine']
['machine', 'machine', 'machine', 'machine', 'simple_machine']
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine']
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine']
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine', 'car']
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine', 'car', 'auto']
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine', 'car', 'auto', 'automobil
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine', 'car', 'auto', 'automobil
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine', 'car', 'auto', 'automobil
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine', 'car', 'auto', 'automobil
['machine', 'machine', 'machine', 'machine', 'simple_machine', 'machine', 'political_machine', 'car', 'auto', 'automobil
```

```
#stemming
from nltk.stem import PorterStemmer

```

```
stemmer=PorterStemmer()
print(stemmer.stem("coding"))
print(stemmer.stem("coded"))
```

```
code
code
```

Experiment 5

Aim : To study N-gram

CODE:

```
import re
from nltk.util import ngrams
S= "machine learning is an important part of AI""and AI is going to become important for daily functioning"
tokens = [token for token in S.split(" ")]
output=list(ngrams(tokens,2))
print("Generated Output.\n",output,"\n")
```

```
Generated Output.
 [('machine', 'learning'), ('learning', 'is'), ('is', 'an'), ('an', 'important'), ('important', 'part'), ('part', 'of'),
```

Experiment 6

Aim:To study chuncking

CODE:

```
import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
```

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

```
text=nltk.word_tokenize("And now for Everything completely Same")
print("Generated Output:\n")
nltk.pos_tag(text)
```

```
Generated Output:

[('And', 'CC'),
 ('now', 'RB'),
 ('for', 'IN'),
 ('Everything', 'VBG'),
 ('completely', 'RB'),
 ('Same', 'JJ')]
```

Experiment 7

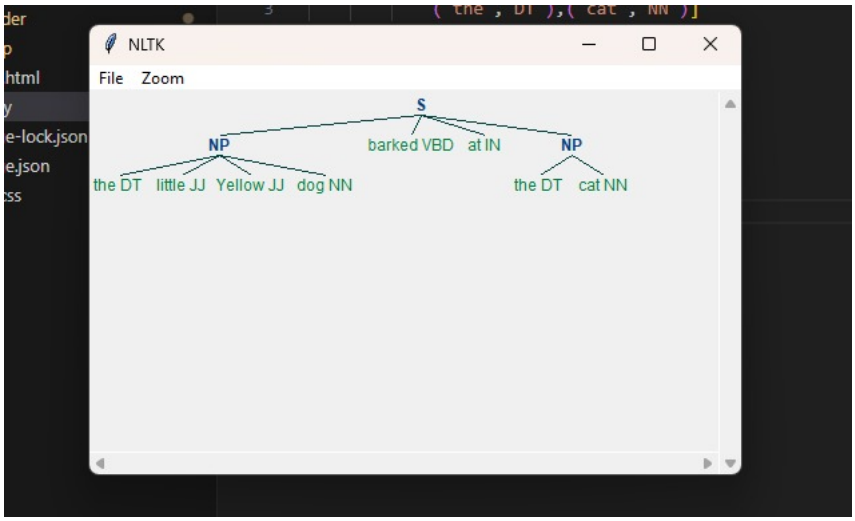
Aim:- To study Chunking

CODE:

```
import nltk
sentence = [("the","DT"),("little","JJ"),("Yellow","JJ"),("dog","NN"),("barked","VBD"),("at","IN"),
            ("the","DT"),("cat","NN")]
grammar="NP:{<DT>?<JJ>*<NN>}"
cp=nltk.RegexpParser(grammar)
result= cp.parse(sentence)
print("Output :=>")
print(result)
```

```
Output :=>
(S
  (NP the/DT little/JJ Yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```

```
result.draw()
```



Experiment 8

Aim:To study Named Entity Recognition

CODE:

```
locs=[('Omnicom','IN','New York'),
      ('DDB Needham','IN','New York'),
      ('Kalpan Thaler Group','IN','New York'),
      ('BBDO','IN','Atlanta'),
      ('Georgia-pacific','IN','Atlanta')]
query=[e1 for(e1,rel,e2)in locs if e2=='Atlanta']
print("OUTPUT\n",query)
```

```
OUTPUT
 ['BBDO', 'Georgia-pacific']
```

Experiment 9

Aim:Finite State Automata

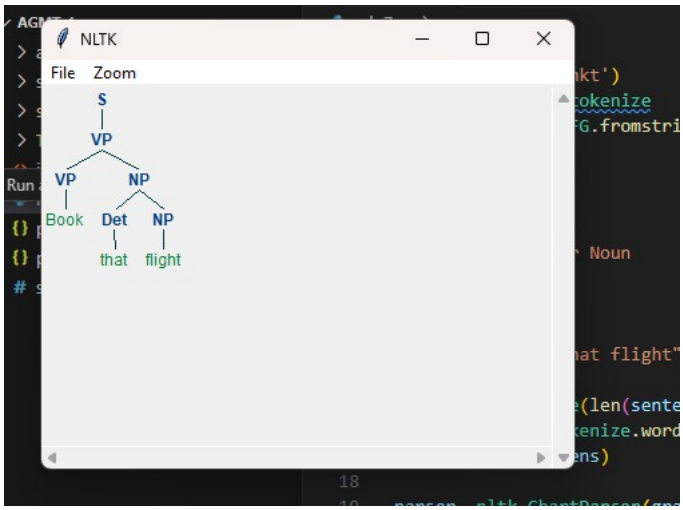
a) Define Grammer Using NLTK . Analyze a sentence using the same.

CODE:

```
import nltk
nltk.download('punkt')
from nltk import tokenize
grammar1 = nltk.CFG.fromstring("""
    S -> VP
    VP -> VP NP
    NP -> Det NP
    Det ->'that'
    NP -> singular Noun
    NP ->'flight'
    VP ->'Book'
    """)
sentence= "Book that flight"

for index in range(len(sentence)):
    all_tokens=tokenize.word_tokenize(sentence)
    print(all_tokens)

parser =nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
    print(tree)
    tree.draw()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
['Book', 'that', 'flight']
(S (VP (VP Book) (NP (Det that) (NP flight))))
```

OUTPUT



b) Accept the input string with regular expression of Finite Automata:

CODE:

```
def FA(s):
#if the lenght is less than 3 it can't be accepted.Therefore program stops.
        if len(s)<3:
            return "Rejected"
#first 3 characters are fixed then check them using index
        if s[0]=='1':
            if s[1]=='0':
                if s[2]=='1':
                    for i in range(3,len(s)):
                        if s[i]!='1':
                            return "Rejected"
                    return "Accepted" #if all 4 nested are true
                return "Rejected" #else of 3rd if
            return "Rejected"
        return "Rejected"
    inputs=['1','10101','101','10111',
            '01010','100','',
            '10111101','1011111']
    print("output\n")
    for i in inputs:
        print(FA(i))

```

```
output

Rejected
Rejected
Accepted
Accepted
Rejected
Rejected
Rejected
Rejected
Accepted
```

c) Accept the input string with regular expression pf FA:(a+b)* bba.

CODE:

```
def FA(s):
    size=0
    for i in s:
        if i=='a' or i=='b':
            size+=1
        else:
            return "Rejected"
    if size>=3: #after checking it only contains a and b only.& checking it lenght should be atleast 3
        if s[size-3]=='b':
            if s[size-2]=='b':
                if s[size-1]=='a':
                    return "Accepted"
                return "Rejected"
                return "Rejected"
        return "Rejected"
    return "Rejected"
inputs=['bba','ababbba','abb','baba','bbb',]
print("OUTPUT\n")
for i in inputs:
    print(FA(i))
```

```
OUTPUT

Accepted
Accepted
Rejected
Rejected
Rejected
```

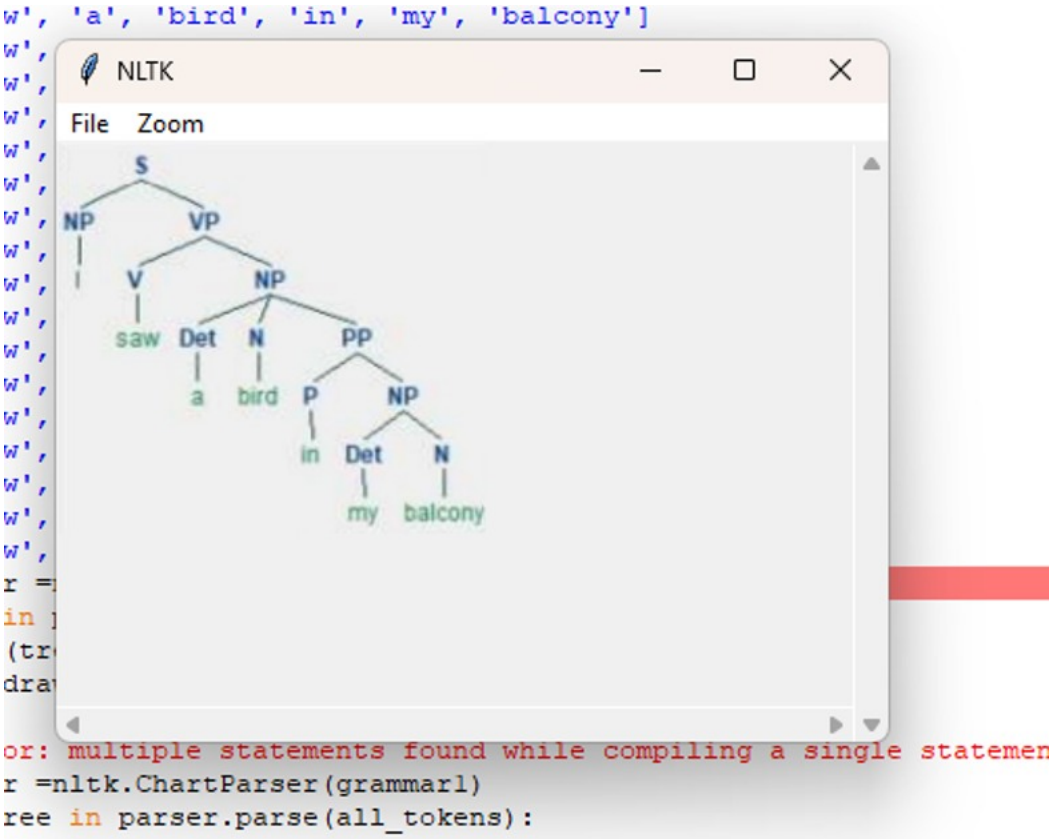
d) Implemntation of Deductive chart parsing using context free grammar and a given sentence.

CODE:

```
# import nltk
nltk.download('punkt')
# from nltk import tokenize
grammar2 = nltk.CFG.fromstring("""
    S -> NPVP
    PP -> PNP
```

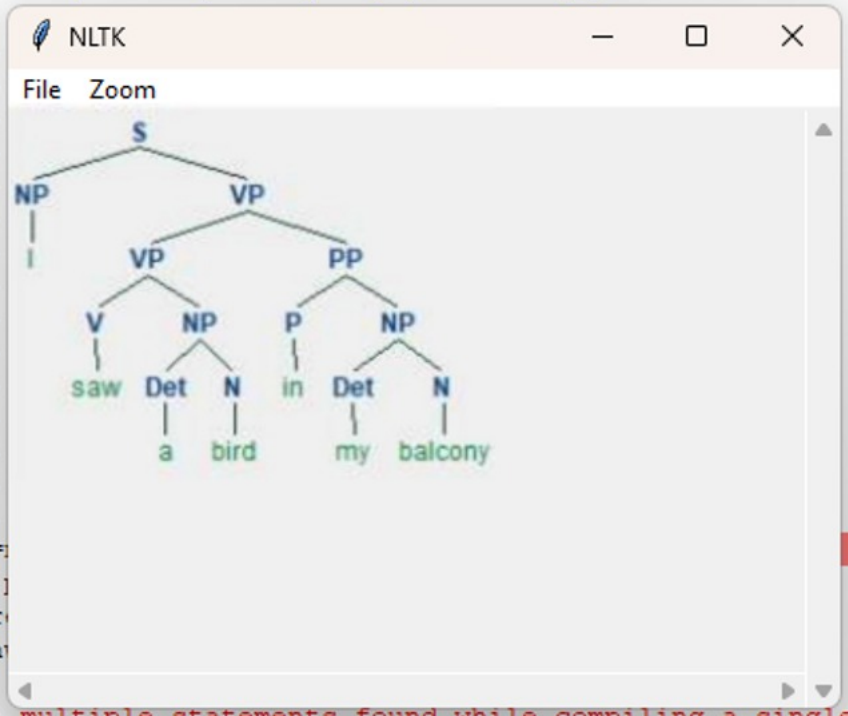
```
    NP -> DetN|DetNPP|'I'
    VP -> VNP|VPPP
    Det ->'a'|'my'
    N -> 'bird'|'balcony'
    v ->'saw'
    P ->'in'
    """)
sentences= "I saw a bird in my balcony"

for index in range(len(sentences)):
    all_tokens1=tokenize.word_tokenize(sentences)
    print(all_tokens1)

parser1 =nltk.ChartParser(grammar2)
for tree in parser1.parse(all_tokens1):
    print(tree)
     tree.draw()
```

```
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```



```
w', 'a', 'bird', 'in', 'my', 'balcony']
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
r =
in
(tr
dra
or: multiple statements found while compiling a single statemen
r =nltk.ChartParser(grammar1)
ree in parser.parse(all_tokens):
```

```
w', 'a', 'bird', 'in', 'my', 'balcony']
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
w',
r =
in
(tr
dra
```




```
or: multiple statements found while compiling a single statemen
r =nltk.ChartParser(grammar1)
ree in parser.parse(all_tokens):
```

Experiment 10

AIM:Study PorterStemmer, LancasterStemmer, RegexpStemmer,SnowballStammer, WordNetLemmatizer

CODE:

```
import nltk
```

```
#PorterStemer
from nltk.stem import PorterStemmer
word_stemmer=PorterStemmer()
print("OUTPUT=>\n")
print(word_stemmer.stem('Coding'))
```

```
OUTPUT=>

code
```

```
#LancasterStemmer
from nltk.stem import LancasterStemmer
Lanc_stemmer=LancasterStemmer()
print("OUTPUT=>\n")
print(Lanc_stemmer.stem('Coding'))
```

```
OUTPUT=>

cod
```

```
#RegexpStemmer
from nltk.stem import RegexpStemmer
st = RegexpStemmer('ing$|s$|e$|able$', min=4)
print("OUTPUT=>")
print(st.stem('coding'))
```

```
OUTPUT=>
cod
```

```
#SnowballStemmer
from nltk.stem import SnowballStemmer
sk_stemmer=SnowballStemmer('english')
print("output=>")
print(sk_stemmer.stem('coding'))
```

```
output=>
code
```

```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
#WordNetLemmatizer
from nltk.stem import WordNetLemmatizer
lemmatizer=WordNetLemmatizer()
print("word :\t Lemma")
print("rocks:", lemmatizer.lemmatize("rocks"))
print("Corpora:", lemmatizer.lemmatize("Corpora"))
print("better:", lemmatizer.lemmatize("better",pos="a"))
```

```
word :   Lemma
rocks: rock
Corpora: Corpora
better: good
```