##ML-SDL Assiggnment

##Name --AMIT KUMAR

#Roll no.-137

```
[i] NLP-intro
```

In [2]:

```
sentence = 'Hello, how are you?'
```

In [3]:

```
words = sentence.split()
```

In [4]:

```
print(words)
```

```
['Hello,', 'how', 'are', 'you?']
```

In [8]:

```
import nltk
nltk.download('punkt')
words = nltk.word_tokenize(sentence)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

In [9]:

```
print(words)
```

```
['Hello', ',', 'how', 'are', 'you', '?']
```

In [10]:

```
text = '''Hydnum repandum, the hedgehog mushroom,
is a fungus of the family Hydnaceae.
First described by Carl Linnaeus in 1753, it is the type species of its genus.
The cap is dry, colored yellow to light orange to brown,
and often develops an irregular shape, especially when crowded.
The mushrooms are characterized by spore-bearing structures—in the form of spines rather
than gills—which hang down from the underside of the cap.
The mushroom tissue is white with a pleasant odor and a spicy or bitter taste.
All parts of the mushroom stain orange with age or when bruised.'''
```

In [11]:

```
sent = nltk.sent_tokenize(text)
```

In [12]:

```
sent
```

Out[12]:

```
['Hydnum repandum, the hedgehog mushroom, \nis a fungus of the family Hydnac
eae.',
 'First described by Carl Linnaeus in 1753, it is the type species of its ge
nus.',
 'The cap is dry, colored yellow to light orange to brown, \nand often devel
ops an irregular shape, especially when crowded.',
 'The mushrooms are characterized by spore-bearing structures—in the form of
spines rather \nthan gills—which hang down from the underside of the cap.',
 'The mushroom tissue is white with a pleasant odor and a spicy or bitter ta
ste.',
 'All parts of the mushroom stain orange with age or when bruised.']
```

In [13]:

```
for x in sent:
    print(nltk.word_tokenize(x))
```

```
['Hydnum', 'repandum', ',', 'the', 'hedgehog', 'mushroom', ',', 'is', 'a',
'fungus', 'of', 'the', 'family', 'Hydnaceae', '.']
['First', 'described', 'by', 'Carl', 'Linnaeus', 'in', '1753', ',', 'it', 'i
s', 'the', 'type', 'species', 'of', 'its', 'genus', '.']
['The', 'cap', 'is', 'dry', ',', 'colored', 'yellow', 'to', 'light', 'orang
e', 'to', 'brown', ',', 'and', 'often', 'develops', 'an', 'irregular', 'shap
e', ',', 'especially', 'when', 'crowded', '.']
['The', 'mushrooms', 'are', 'characterized', 'by', 'spore-bearing', 'structu
res—in', 'the', 'form', 'of', 'spines', 'rather', 'than', 'gills—which', 'ha
ng', 'down', 'from', 'the', 'underside', 'of', 'the', 'cap', '.']
['The', 'mushroom', 'tissue', 'is', 'white', 'with', 'a', 'pleasant', 'odo
r', 'and', 'a', 'spicy', 'or', 'bitter', 'taste', '.']
['All', 'parts', 'of', 'the', 'mushroom', 'stain', 'orange', 'with', 'age',
'or', 'when', 'bruised', '.']
```

In [14]:

```
from nltk.corpus import stopwords
```

In [17]:

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[17]:

```
True
```

In [19]:

```
print(stopwords.words('german'))
```

```
['aber', 'alle', 'allem', 'allen', 'aller', 'alles', 'als', 'also', 'am', 'a
n', 'ander', 'andere', 'anderem', 'anderen', 'anderer', 'anderes', 'anderm',
'andern', 'anderr', 'anders', 'auch', 'auf', 'aus', 'bei', 'bin', 'bis', 'bi
st', 'da', 'damit', 'dann', 'der', 'den', 'des', 'dem', 'die', 'das', 'das
s', 'daß', 'derselbe', 'derselben', 'denselben', 'desselben', 'demselben',
'dieselbe', 'dieselben', 'dasselbe', 'dazu', 'dein', 'deine', 'deinem', 'dei
nen', 'deiner', 'deines', 'denn', 'derer', 'dessen', 'dich', 'dir', 'du', 'd
ies', 'diese', 'diesem', 'diesen', 'dieser', 'dieses', 'doch', 'dort', 'durc
h', 'ein', 'eine', 'einem', 'einen', 'einer', 'eines', 'einig', 'einige', 'e
inigem', 'einigen', 'einiger', 'einiges', 'einmal', 'er', 'ihn', 'ihm', 'e
s', 'etwas', 'euer', 'eure', 'eurem', 'euren', 'eurer', 'eures', 'für', 'geg
en', 'gewesen', 'hab', 'habe', 'haben', 'hat', 'hatte', 'hatten', 'hier', 'h
in', 'hinter', 'ich', 'mich', 'mir', 'ihr', 'ihre', 'ihrem', 'ihren', 'ihre
r', 'ihres', 'euch', 'im', 'in', 'indem', 'ins', 'ist', 'jede', 'jedem', 'je
den', 'jeder', 'jedes', 'jene', 'jenem', 'jenen', 'jener', 'jenes', 'jetzt',
'kann', 'kein', 'keine', 'keinem', 'keinen', 'keiner', 'keines', 'können',
'könnte', 'machen', 'man', 'manche', 'manchem', 'manchen', 'mancher', 'manch
es', 'mein', 'meine', 'meinem', 'meinen', 'meiner', 'meines', 'mit', 'muss',
'musste', 'nach', 'nicht', 'nichts', 'noch', 'nun', 'nur', 'ob', 'oder', 'oh
ne', 'sehr', 'sein', 'seine', 'seinem', 'seinen', 'seiner', 'seines', 'selbs
t', 'sich', 'sie', 'ihnen', 'sind', 'so', 'solche', 'solchem', 'solchen', 's
olcher', 'solches', 'soll', 'sollte', 'sondern', 'sonst', 'über', 'um', 'un
d', 'uns', 'unsere', 'unserem', 'unseren', 'unser', 'unseres', 'unter', 'vie
l', 'vom', 'von', 'vor', 'während', 'war', 'waren', 'warst', 'was', 'weg',
'weil', 'weiter', 'welche', 'welchem', 'welchen', 'welcher', 'welches', 'wen
n', 'werde', 'werden', 'wie', 'wieder', 'will', 'wir', 'wird', 'wirst', 'w
o', 'wollen', 'wollte', 'würde', 'würden', 'zu', 'zum', 'zur', 'zwar', 'zwis
chen']
```

In [20]:

```
import urllib
import nltk
from bs4 import BeautifulSoup
response = urllib.request.urlopen('https://en.wikipedia.org/wiki/Rajgad_Fort')
```

In [21]:

```
html = response.read()
html
```

Out[21]:

```
b'<!DOCTYPE html>\n<html class="client-nojs" lang="en" dir="ltr">\n<head>
\n<meta charset="UTF-8"/>\n<title>Rajgad Fort - Wikipedia</title>\n<script
>document.documentElement.className="client-js";RLCONF={"wgBreakFrames":!
1,"wgSeparatorTransformTable":["",""],"wgDigitTransformTable":["",""],"wgD
efaultDateFormat":"dmy","wgMonthNames":["","January","February","March","A
pril","May","June","July","August","September","October","November","Decem
ber"],"wgRequestId":"cd9d2cf8-a1fb-484d-8b97-258cf66f8bf6","wgCSPNonce":!
1,"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":!1,"wgNamespaceNu
mber":0,"wgPageName":"Rajgad_Fort","wgTitle":"Rajgad Fort","wgCurRevisionI
d":983221827,"wgRevisionId":983221827,"wgArticleId":12475798,"wgIsArticl
e":!0,"wgIsRedirect":!1,"wgAction":"view","wgUserName":null,"wgUserGroup
s":["*"],"wgCategories":["Articles with short description","Short descript
ion matches Wikidata","Use dmy dates from July 2017","Use Indian English f
rom July 2017","All Wikipedia articles written in Indian English","All art
icles with unsourced statements","Articles with unsourced statements from
July 2017",\n"Commons category link is on Wikidata","Forts in Pune distric
t","Former capital cities in India"],"wgPageContentLanguage":"en","wgPageC
ontentModel":"wikitext","wgRelevantPageName":"Rajgad_Fort","wgRelevantArti
```

In [22]:

```
soup = BeautifulSoup(html,"html.parser")
text = soup.get_text(strip=True)
text
```

Out[22]:

```
'Rajgad Fort - Wikipediadocument.documentElement.className="client-js";RLC
ONF={"wgBreakFrames":!1,"wgSeparatorTransformTable":["",""],"wgDigitTransf
ormTable":["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":["","Januar
y","February","March","April","May","June","July","August","September","Oc
tober","November","December"],"wgRequestId":"cd9d2cf8-a1fb-484d-8b97-258cf
66f8bf6","wgCSPNonce":!1,"wgCanonicalNamespace":"","wgCanonicalSpecialPage
Name":!1,"wgNamespaceNumber":0,"wgPageName":"Rajgad_Fort","wgTitle":"Rajga
d Fort","wgCurRevisionId":983221827,"wgRevisionId":983221827,"wgArticleI
d":12475798,"wgIsArticle":!0,"wgIsRedirect":!1,"wgAction":"view","wgUserNa
me":null,"wgUserGroups":["*"],"wgCategories":["Articles with short descrip
tion","Short description matches Wikidata","Use dmy dates from July 201
7","Use Indian English from July 2017","All Wikipedia articles written in
Indian English","All articles with unsourced statements","Articles with un
sourced statements from July 2017",\n"Commons category link is on Wikidat
a","Forts in Pune district","Former capital cities in India"],"wgPageConte
ntLanguage":"en","wgPageContentModel":"wikitext","wgRelevantPageName":"Raj
gad_Fort","wgRelevantArticleId":12475798,"wgIsProbablyEditable":!0,"wgRele
vantPageIsProbablyEditable":!0,"wgRestrictionEdit":[],"wgRestrictionMove":
```

In [23]:

```
tokens = [t for t in text.split()]
```

In [24]:

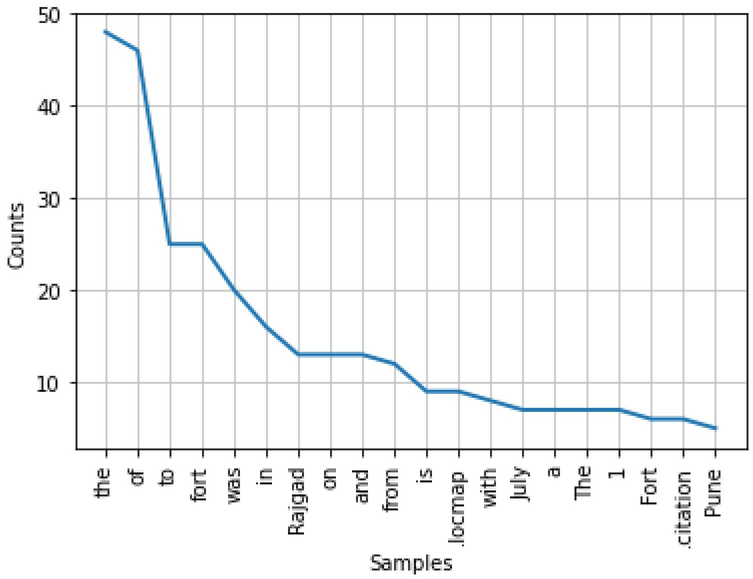
```
tokens
```

Out[24]:

```
['Rajgad',
 'Fort',
 '-',
 'Wikipediadocument.documentElement.className="client-js";RLCONF={"wgBreak
Frames":!1,"wgSeparatorTransformTable":["",""],"wgDigitTransformTable":
["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":["","January","Februar
y","March","April","May","June","July","August","September","October","Nov
ember","December"],"wgRequestId":"cd9d2cf8-a1fb-484d-8b97-258cf66f8bf6","w
gCSPNonce":!1,"wgCanonicalNamespace":"","wgCanonicalSpecialPageName":!1,"w
gNamespaceNumber":0,"wgPageName":"Rajgad_Fort","wgTitle":"Rajgad',
 'Fort","wgCurRevisionId":983221827,"wgRevisionId":983221827,"wgArticleI
d":12475798,"wgIsArticle":!0,"wgIsRedirect":!1,"wgAction":"view","wgUserNa
me":null,"wgUserGroups":["*"],"wgCategories":["Articles',
 'with',
 'short',
 'description","Short',
 'description',
 'matches',
```

In [25]:

```
freq = nltk.FreqDist(tokens)
freq.plot(20, cumulative=False)
```



In [26]:

```
from nltk.corpus import stopwords
```

In [27]:

```
sword = stopwords.words('english')
```

In [28]:

```
sword
```

Out[28]:

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
```

In [31]:

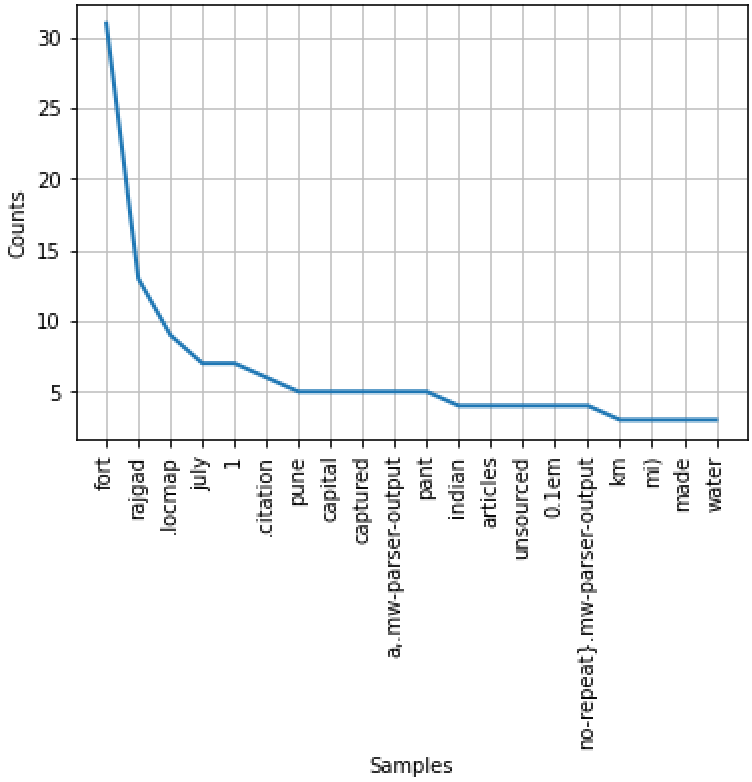
```
clean_tokens = []
for token in tokens:
    if token.lower() not in sword:
        clean_tokens.append(token.lower())
clean_tokens
```

Out[31]:

```
['rajgad',
 'fort',
 '-',
 'wikipediadocument.documentelement.classname="client-js";rlconf={"wgbreak
frames":!1,"wgseparatortransformtable":["",""],"wgdigittransformtable":
["",""],"wgdefaultdateformat":"dmy","wgmonthnames":["","january","februar
y","march","april","may","june","july","august","september","october","nov
ember","december"],"wgrequestid":"cd9d2cf8-a1fb-484d-8b97-258cf66f8bf6","w
gcspnonce":!1,"wgcanonicalnamespace":"","wgcanonicalspecialpagename":!1,"w
gnamespacenumber":0,"wgpagename":"rajgad_fort","wgtitle":"rajgad',
 'fort","wgcurrevisionid":983221827,"wgrevisionid":983221827,"wgarticlei
d":12475798,"wgisarticle":!0,"wgisredirect":!1,"wgaction":"view","wguserna
me":null,"wgusergroups":["*"],"wgcategories":["articles',
 'short',
 'description","short',
 'description',
 'matches',
 'wikidata","use',
```

In [32]:

```
freq = nltk.FreqDist(clean_tokens)
freq.plot(20, cumulative=False)
```



In [33]:

```
import string
string.punctuation
```

Out[33]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [35]:

```
from nltk.corpus import wordnet
nltk.download('wordnet')

syno = wordnet.synsets("earth")

print(syno[0].definition())
print(syno[0].examples())
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
the 3rd planet from the sun; the planet we live on
['the Earth moves around the sun', 'he sailed around the world']
```

In [36]:

```
synonyms = []

for syn in wordnet.synsets('page'):
    for lemma in syn.lemmas():
        synonyms.append(lemma.name())

print(set(synonyms))
```

```
{'paginate', 'Page', 'varlet', 'pageboy', 'Thomas_Nelson_Page', 'foliate',
'Sir_Frederick_Handley_Page', 'page'}
```

In [37]:

```
antonyms = []

for syn in wordnet.synsets("up"):
    for l in syn.lemmas():
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())

print(set(antonyms))
```

```
{'downwards', 'downward', 'downwardly', 'down'}
```

In [38]:

```
from nltk.stem import PorterStemmer
from nltk.stem import LancasterStemmer

stemmer = PorterStemmer()
print(stemmer.stem('gone'))
stemmer = LancasterStemmer()
print(stemmer.stem('gone'))
```

```
gone
gon
```

In [39]:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize('ate'))
```

```
ate
```

In [40]:

```
# Verb
print(lemmatizer.lemmatize('associations', pos="v"))
# Noun
print(lemmatizer.lemmatize('associations', pos="n"))
# Ajective
print(lemmatizer.lemmatize('associations', pos="a"))
# Adverb
print(lemmatizer.lemmatize('associations', pos="r"))

print(lemmatizer.lemmatize('players', pos="n"))
print(lemmatizer.lemmatize('playing', pos="n"))
```

```
associations
association
associations
associations
player
playing
```

In [41]:

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

txt = '''Shivneri is a hill fort having a triangular shape and has its entrance from the So
```

In [43]:

```
nltk.download('averaged_perceptron_tagger')
tokenized = sent_tokenize(txt)
for i in tokenized:
    wordsList = nltk.word_tokenize(i)
    # removing stop words from wordList
    wordsList = [w for w in wordsList if not w in stop_words]
    #  Using a Tagger. Which is part-of-speech
    # tagger or POS-tagger.
    tagged = nltk.pos_tag(wordsList)
    print(tagged)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[('Shivneri', 'NNP'), ('hill', 'NN'), ('fort', 'NN'), ('triangular', 'JJ'),
('shape', 'NN'), ('entrance', 'NN'), ('South-west', 'NNP'), ('side', 'NN'),
('hill', 'NN'), ('.', '.')]
[('Apart', 'RB'), ('main', 'JJ'), ('gate', 'NN'), ('entrance', 'NN'), ('for
t', 'NN'), ('side', 'NN'), ('called', 'VBN'), ('locally', 'RB'), ('chain',
'NN'), ('gate', 'NN'), (',', ','), ('one', 'CD'), ('hold', 'NN'), ('chains',
'NNS'), ('climb', 'VBP'), ('fort', 'JJ'), ('gate', 'NN'), ('.', '.')]
[('The', 'DT'), ('fort', 'NN'), ('extends', 'VBZ'), ('1', 'CD'), ('mi', 'N
N'), ('(', '('), ('1.6', 'CD'), ('km', 'NN'), (')', ')'), ('seven', 'CD'),
('spiral', 'JJ'), ('well-defended', 'JJ'), ('gates', 'NNS'), ('.', '.')]
[('There', 'EX'), ('mud', 'NN'), ('walls', 'NNS'), ('around', 'IN'), ('for
t', 'NN'), ('.', '.')]
[('Inside', 'IN'), ('fort', 'NN'), (',', ','), ('major', 'JJ'), ('building
s', 'NNS'), ('prayer', 'NN'), ('hall', 'NN'), (',', ','), ('tomb', 'NN'),
('mosque', 'NN'), ('.', '.')]
```

In [44]:

```
tokenized = sent_tokenize(txt)
for i in tokenized:
    wordsList = nltk.word_tokenize(i)
    # removing stop words from wordList
    wordsList = [w for w in wordsList if not w in stop_words]
    #  Using a Tagger. Which is part-of-speech
    # tagger or POS-tagger.
    tagged = nltk.pos_tag(wordsList)
    print(tagged)
```

```
[('Shivneri', 'NNP'), ('hill', 'NN'), ('fort', 'NN'), ('triangular', 'JJ'),
('shape', 'NN'), ('entrance', 'NN'), ('South-west', 'NNP'), ('side', 'NN'),
('hill', 'NN'), ('.', '.')]
[('Apart', 'RB'), ('main', 'JJ'), ('gate', 'NN'), ('entrance', 'NN'), ('for
t', 'NN'), ('side', 'NN'), ('called', 'VBN'), ('locally', 'RB'), ('chain',
'NN'), ('gate', 'NN'), (',', ','), ('one', 'CD'), ('hold', 'NN'), ('chains',
'NNS'), ('climb', 'VBP'), ('fort', 'JJ'), ('gate', 'NN'), ('.', '.')]
[('The', 'DT'), ('fort', 'NN'), ('extends', 'VBZ'), ('1', 'CD'), ('mi', 'N
N'), ('(', '('), ('1.6', 'CD'), ('km', 'NN'), (')', ')'), ('seven', 'CD'),
('spiral', 'JJ'), ('well-defended', 'JJ'), ('gates', 'NNS'), ('.', '.')]
[('There', 'EX'), ('mud', 'NN'), ('walls', 'NNS'), ('around', 'IN'), ('for
t', 'NN'), ('.', '.')]
[('Inside', 'IN'), ('fort', 'NN'), (',', ','), ('major', 'JJ'), ('building
s', 'NNS'), ('prayer', 'NN'), ('hall', 'NN'), (',', ','), ('tomb', 'NN'),
('mosque', 'NN'), ('.', '.')]
```

In [ ]:

##[ii] Synonyms

In [46]:

```

#importing wordnet:
from nltk.corpus import wordnet

# Then, we're going to use the term "program" to find synsets like so:
syns = wordnet.synsets("world")

# An example of a synset:
print(syns[0].name())

# Just the word:
print(syns[0].lemmas()[0].name())

# Definition of that first synset:
print(syns[0].definition())

# Examples of the word in use in sentences:
print(syns[0].examples())

```

```
universe.n.01
universe
everything that exists anywhere
['they study the evolution of the universe', 'the biggest tree in existenc
e']
```

In [47]:

```
synonyms = []
for syn in wordnet.synsets("catch"):
    for l in syn.lemmas():
        synonyms.append(l.name())
```

In [48]:

```
set(synonyms)
```

Out[48]:

```
{'apprehension',
 'arrest',
 'becharm',
 'beguile',
 'bewitch',
 'captivate',
 'capture',
 'catch',
 'catch_up_with',
 'charm',
 'collar',
 'enamor',
 'enamour',
 'enchant',
 'entrance',
 'fascinate',
 'get',
 'gimmick',
 'grab',
 'haul',
 'hitch',
 'match',
 'overhear',
 'overtake',
 'pick_up',
 'pinch',
 'see',
 'snap',
 'snatch',
 'stop',
 'take_hold_of',
 'take_in',
 'taking_into_custody',
 'trance',
 'trip_up',
 'view',
 'watch'}
```

In [49]:

```
import nltk
from nltk.corpus import wordnet     #Import wordnet from the NLTK
first_word = wordnet.synset("Travel.v.01")
second_word = wordnet.synset("Walk.v.01")
print('Similarity: ' + str(first_word.wup_similarity(second_word)))
first_word = wordnet.synset("Ship.n.01")
second_word = wordnet.synset("boat.n.01")
print('Similarity: ' + str(first_word.wup_similarity(second_word)))
```

```
Similarity: 0.6666666666666666
Similarity: 0.9090909090909091
```

In [50]:

```
antonyms = []
for syn in wordnet.synsets("slow"):
    for l in syn.lemmas():
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())
```

In [51]:

```
set(antonyms)
```

Out[51]:

```
{'accelerate', 'fast', 'quickly'}
```

In [ ]:

```

```

##Lemmatization

In [52]:

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

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

Out[52]:

```
True
```

In [53]:

```
from nltk.stem import WordNetLemmatizer
```

In [54]:

```
word='largest'
```

In [55]:

```
lt = WordNetLemmatizer()
```

In [56]:

```
print(lt.lemmatize(word, pos='v'))
print(lt.lemmatize(word, pos='n'))
print(lt.lemmatize(word, pos='a'))
print(lt.lemmatize(word, pos='r'))
```

```
largest
largest
large
largest
```

In [57]:

```
sent = '''He has played a finest innings in world.
The opportunities will never come again'''
```

In [58]:

```
for word in sent.split():
    print(lt.lemmatize(word, pos='a'))
```

```
He
has
played
a
fine
innings
in
world.
The
opportunities
will
never
come
again
```

In [ ]:

```

```

##[iii] Tokenizer

In [59]:

```
# import the existing word and the sentence tokenizing
# libraries
from nltk.tokenize import sent_tokenize, word_tokenize

text = '''Natural language processing (NLP) is a field
       of computer science, artificial intelligence
       and computational linguistics concerned with
       the interactions between computers and human
       (natural) languages, and, in particular,
       concerned with programming computers to
       fruitfully process large natural language
       corpora. Challenges in natural language
       processing frequently involve natural
       language understanding, natural language
       generation frequently from formal, machine
       -readable logical forms), connecting language
       and machine perception, managing human-
       computer dialog systems, or some combination
       thereof.'''

print(sent_tokenize(text))
```

```
['Natural language processing (NLP) is a field\n       of computer science,
artificial intelligence \n       and computational linguistics concerned wit
h \n       the interactions between computers and human \n       (natural) l
anguages, and, in particular,  \n       concerned with programming computers
to \n       fruitfully process large natural language \n       corpora.', 'C
hallenges in natural language \n       processing frequently involve natural
\n       language understanding, natural language \n       generation freque
ntly from formal, machine \n       -readable logical forms), connecting lang
uage \n       and machine perception, managing human-  \n       computer dia
log systems, or some combination  \n       thereof.']
```

In [60]:

```
print(word_tokenize(text))
```

```
['Natural', 'language', 'processing', '(', 'NLP', ')', 'is', 'a', 'field',
'of', 'computer', 'science', ',', 'artificial', 'intelligence', 'and', 'comp
utational', 'linguistics', 'concerned', 'with', 'the', 'interactions', 'betw
een', 'computers', 'and', 'human', '(', 'natural', ')', 'languages', ',', 'a
nd', ',', 'in', 'particular', ',', 'concerned', 'with', 'programming', 'comp
uters', 'to', 'fruitfully', 'process', 'large', 'natural', 'language', 'corp
ora', '.', 'Challenges', 'in', 'natural', 'language', 'processing', 'frequen
tly', 'involve', 'natural', 'language', 'understanding', ',', 'natural', 'la
nguage', 'generation', 'frequently', 'from', 'formal', ',', 'machine', '-rea
dable', 'logical', 'forms', ')', ',', 'connecting', 'language', 'and', 'mach
ine', 'perception', ',', 'managing', 'human-', 'computer', 'dialog', 'system
s', ',', 'or', 'some', 'combination', 'thereof', '.']
```

In [61]:

```
# import TabTokenizer() method from nltk
from nltk.tokenize import TabTokenizer

# Create a reference variable for Class TabTokenizer
tk = TabTokenizer()

# Create a string input
gfg = "विज्ञानाचा उगम मानवी जिज्ञानेतून झाला आहे. \tज्ञानासंबंधीचे विशुद्ध प्रेम ही विज्ञानाची प्रेरणा आहे. वस्तुनिष्ठ सत

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['विज्ञानाचा उगम मानवी जिज्ञानेतून झाला आहे. ', 'ज्ञानासंबंधीचे विशुद्ध प्रेम ही विज्ञानाची प्रेरणा आ
हे. वस्तुनिष्ठ सत्याचा शोध घेणे हे विज्ञानाचे एक महत्त्वाचे वैशिष्ट्य मानले जाते. विज्ञान हे सत्यसंशोध
नासाठी प्रयत्नशील असते; परंतु वैज्ञानिक सत्य हे विशेष स्वरूपाचे असते.']
```

In [62]:

```
# import SpaceTokenizer() method from nltk
from nltk.tokenize import SpaceTokenizer

# Create a reference variable for Class SpaceTokenizer
tk = SpaceTokenizer()

# Create a string input
gfg = "Geeksfor Geeks.. .$$&* \nis\t for geeks"

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['Geeksfor', 'Geeks..', '.$$&*', '\nis\t', 'for', 'geeks']
```

In [63]:

```
# import MWETokenizer() method from nltk
from nltk.tokenize import MWETokenizer

# Create a reference variable for Class MWETokenizer
tk = MWETokenizer([('g', 'f', 'g'), ('geeks', 'for', 'geek')])

# Create a string input
gfg = "geeks for geeks g f g"

# Use tokenize method
geek = tk.tokenize(gfg.split())

print(geek)
```

```
['geeks', 'for', 'geeks', 'g_f_g']
```

In [64]:

```
# import LineTokenizer() method from nltk
from nltk.tokenize import LineTokenizer

# Create a reference variable for Class LineTokenizer
tk = LineTokenizer()

# Create a string input
gfg = "GeeksforGeeks...$$&* \nis\n for geeks"

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['GeeksforGeeks...$$&* ', 'is', ' for geeks']
```

In [65]:

```
# import WhitespaceTokenizer() method from nltk
from nltk.tokenize import WhitespaceTokenizer

# Create a reference variable for Class WhitespaceTokenizer
tk = WhitespaceTokenizer()

# Create a string input
gfg = "GeeksforGeeks \nis\t for geeks"

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['GeeksforGeeks', 'is', 'for', 'geeks']
```

In [66]:

```
# import SExprTokenizer() method from nltk
from nltk.tokenize import SExprTokenizer

# Create a reference variable for Class SExprTokenizer
tk = SExprTokenizer()

# Create a string input
gfg = "( a * ( b + c ))ab( a-c )"

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['( a * ( b + c ))', 'ab', '( a-c )']
```

In [67]:

```
# import TweetTokenizer() method from nltk
from nltk.tokenize import TweetTokenizer

# Create a reference variable for Class TweetTokenizer
tk = TweetTokenizer()

# Create a string input
gfg = "Geeks for Geeks"

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['Geeks', 'for', 'Geeks']
```

In [68]:

```
# import TweetTokenizer() method from nltk
from nltk.tokenize import TweetTokenizer

# Create a reference variable for Class TweetTokenizer
tk = TweetTokenizer()

# Create a string input
gfg = ":-) <> () {} [] :-p"

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
[':-)', '<', '>', '(', ')', '{', '}', '[', ']', ':-p']
```

In [69]:

```
# import WordPunctTokenizer() method from nltk
from nltk.tokenize import WordPunctTokenizer

# Create a reference variable for Class WordPunctTokenizer
tk = WordPunctTokenizer()

# Create a string input
gfg = "The price\t of burger \nin BurgerKing is Rs.36.\n"

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['The', 'price', 'of', 'burger', 'in', 'BurgerKing', 'is', 'Rs', '.', '36',
'.']
```

In [70]:

```
from nltk.tokenize import BlanklineTokenizer
# Create a reference variable for Class WordPunctTokenizer
tk = BlanklineTokenizer()

# Create a string input
gfg = '''Hello friends

How are you?
Good bye!!!'''

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['Hello friends', 'How are you?\nGood bye!!!']
```

In [71]:

```
from nltk.tokenize import ToktokTokenizer

tk = ToktokTokenizer()

# Create a string input
gfg = '''Hello friends.

How are you?
Good bye.'''

# Use tokenize method
geek = tk.tokenize(gfg)

print(geek)
```

```
['Hello', 'friends.', 'How', 'are', 'you', '?', 'Good', 'bye', '.']
```

In [71]:

##[iv] Stopwords

In [72]:

```
import nltk
nltk.download('stopwords')
```

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

Out[72]:

```
True
```

In [73]:

```
from nltk.corpus import stopwords
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves',
'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'i
t', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselve
s', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'tho
se', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'bu
t', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for',
'with', 'about', 'against', 'between', 'into', 'through', 'during', 'befor
e', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'o
n', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'the
re', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mo
re', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'sa
me', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "d
on't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y',
'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
oesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 's
han', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "were
n't", 'won', "won't", 'wouldn', "wouldn't"]
```

In [74]:

```
from nltk.tokenize import word_tokenize

example_sent = "The project is completed by a developer"

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

word_tokens = word_tokenize(example_sent)
```

In [75]:

```
filtered_sentence = [w for w in word_tokens if not w in stop_words]
```

In [76]:

```
filtered_sentence
```

Out[76]:

```
['The', 'project', 'completed', 'developer']
```

In [77]:

```
word_tokens
```

Out[77]:

```
['The', 'project', 'is', 'completed', 'by', 'a', 'developer']
```

In [ ]:

```

```

##--

##[v] Stemming

In [78]:

```
s1 = 'cats', 'catlike', 'catty', 'cat'
s2 = 'stemmer', 'stemming', 'stemmed', 'stem'
s3 = 'fishing', 'fished', 'fisher', 'fish'
s4 = 'argue', 'argued', 'argues', 'arguing', 'argus', 'argu'
s5 = 'argument', 'arguments', 'argument'
s6 = 'play','player','players','played'
```

In [79]:

```
import nltk
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem import SnowballStemmer
```

In [80]:

```
ps=PorterStemmer()
for word in s3:
    print(ps.stem(word))
```

```
fish
fish
fisher
fish
```

In [81]:

```
ls=LancasterStemmer()
for word in s6:
    print(ls.stem(word))
```

```
play
play
play
play
```

In [82]:

```
ss=SnowballStemmer('english')
for word in s6:
    print(ss.stem(word))
```

```
play
player
player
play
```

In [ ]:

```

```

##[vi] Spam-ham

Importing libraries & Data

In [84]:

```
import nltk
import pandas as pd
import re
from sklearn.feature_extraction.text import TfidfVectorizer
import string
data = pd.read_csv("SMSSpamCollection",
                   names=['label', 'body_text'], sep='\t')
data.head()
```

Out[84]:

	label	body_text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

In [85]:

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

Out[85]:

```
ham     4825
spam     747
Name: label, dtype: int64
```

Preprocessing of data

In [86]:

```
stopwords = nltk.corpus.stopwords.words('english')
ps = nltk.PorterStemmer() #

def count_punct(text):
    count = sum([1 for char in text if char in string.punctuation])
    return round(count/(len(text) - text.count(" ")), 3)*100

data['body_len'] = data['body_text'].apply(lambda x: len(x) - x.count(" "))
data['punct%'] = data['body_text'].apply(lambda x: count_punct(x))

def clean_text(text):
    text = "".join([word.lower() for word in text if word not in string.punctuation])
    tokens = re.split('\W+', text)
    text = [ps.stem(word) for word in tokens if word not in stopwords]
    return text
```

split into train or test

In [87]:

```
from sklearn.model_selection import train_test_split

X=data[['body_text', 'body_len', 'punct%']]
y=data['label']

X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.2, random_state=0)
```

Vectorization of text

In [88]:

```
tfidf_vect = TfidfVectorizer(analyzer=clean_text)
tfidf_vect_fit = tfidf_vect.fit(X_train['body_text'])

tfidf_train = tfidf_vect_fit.transform(X_train['body_text'])
tfidf_test = tfidf_vect_fit.transform(X_test['body_text'])

X_train_vect = pd.concat([X_train[['body_len', 'punct%']].reset_index(drop=True),
           pd.DataFrame(tfidf_train.toarray())], axis=1)
X_test_vect = pd.concat([X_test[['body_len', 'punct%']].reset_index(drop=True),
           pd.DataFrame(tfidf_test.toarray())], axis=1)

X_train_vect.head()
```

Out[88]:

	body_len	punct%	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	47	6.4	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	60	3.3	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	56	8.9	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	34	5.9	0.283926	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	26	11.5	0.295509	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 7183 columns

Final Evolution of given model

In [89]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [90]:

```
rf = RandomForestClassifier(n_estimators=150, random_state=0)
rf_model = rf.fit(X_train_vect, y_train)
y_pred = rf_model.predict(X_test_vect)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, classification_report
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[955   0]
 [ 24 136]]
```

In [91]:

```
accuracy_score(y_test, y_pred) * 100
```

Out[91]:

```
97.847533632287
```

In [92]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99       955
        spam       1.00      0.85      0.92       160

    accuracy                           0.98      1115
   macro avg       0.99      0.93      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

In [94]:

```
new = data.head(3)
new
```

Out[94]:

	label	body_text	body_len	punct%
0	ham	Go until jurong point, crazy.. Available only ...	92	9.8
1	ham	Ok lar... Joking wif u oni...	24	25.0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	128	4.7

In [96]:

```
new_vect = tfidf_vect_fit.transform(new['body_text'])
```

In [97]:

```
new_vect
```

Out[97]:

```
<3x7181 sparse matrix of type '<class 'numpy.float64'>'
	with 43 stored elements in Compressed Sparse Row format>
```

In [98]:

```
new
```

Out[98]:

	label	body_text	body_len	punct%
0	ham	Go until jurong point, crazy.. Available only ...	92	9.8
1	ham	Ok lar... Joking wif u oni...	24	25.0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	128	4.7

In [99]:

```
sample_vect = pd.concat([new[['body_len', 'punct%']].reset_index(drop=True),
           pd.DataFrame(new_vect.toarray())], axis=1)
```

In [100]:

```
sample_vect
```

Out[100]:

	body_len	punct%	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	92	9.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	24	25.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	128	4.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

3 rows × 7183 columns

In [101]:

```
rf_model.predict(sample_vect)
```

Out[101]:

```
array(['ham', 'ham', 'spam'], dtype=object)
```

In [ ]:

```
```

##---

In [ ]:

```
```