INDEX

9	Implement Naive Bayes classifier	
10Ai	Speech tagging using spacy	
10Aii	Speech tagging using nltk	
10Bi	Usage of Give and Gave in the Penn Treebank sample	
10Bii	probabilistic parser	
10C	Malt parsing: Parse a sentence and draw a tree using malt parsing.	
11A	Multiword Expressions in NLP.	
11B	Normalized Web Distance and Word Similarity.	
11C	Word Sense Disambiguation.	

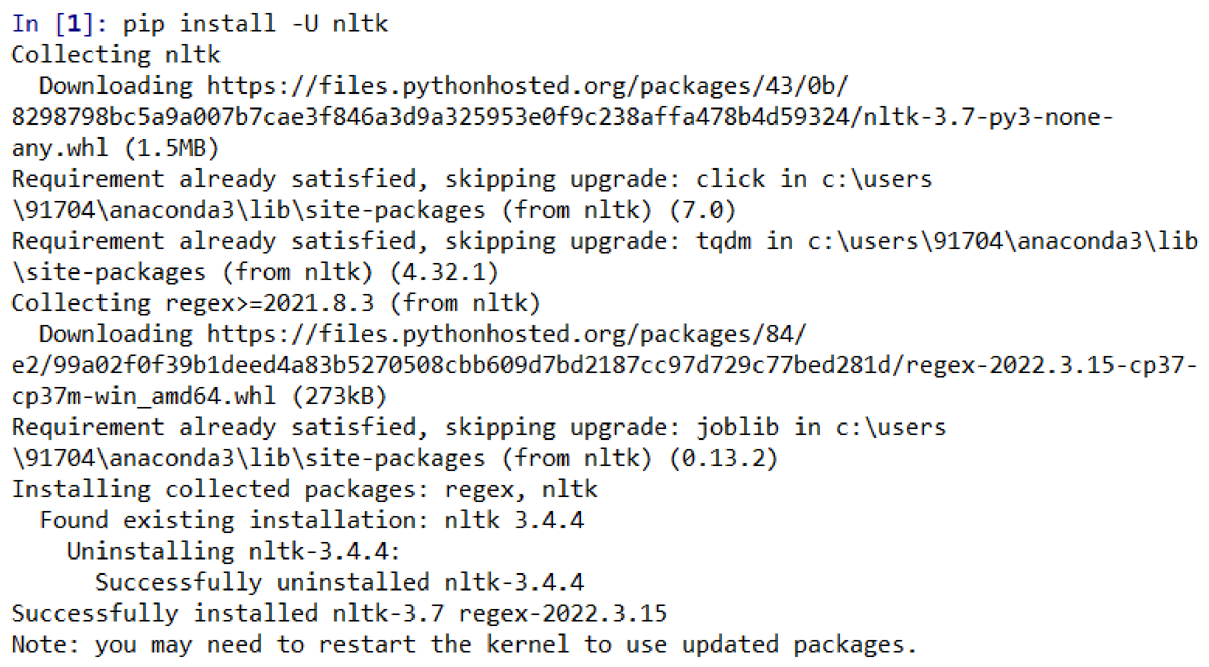
Practical 1A

Aim: To install NLTK in Python

Code:

pip install -U nltk

Output:



```
In [1]: pip install -U nltk
Collecting nltk
  Downloading https://files.pythonhosted.org/packages/43/0b/
8298798bc5a9a007b7cae3f846a3d9a325953e0f9c238affa478b4d59324/nltk-3.7-py3-none-
any.whl (1.5MB)
Requirement already satisfied, skipping upgrade: click in c:\users
\91704\anaconda3\lib\site-packages (from nltk) (7.0)
Requirement already satisfied, skipping upgrade: tqdm in c:\users\91704\anaconda3\lib
\site-packages (from nltk) (4.32.1)
Collecting regex>=2021.8.3 (from nltk)
  Downloading https://files.pythonhosted.org/packages/84/
e2/99a02f0f39b1deed4a83b5270508cbb609d7bd2187cc97d729c77bed281d/regex-2022.3.15-cp37-
cp37m-win_amd64.whl (273kB)
Requirement already satisfied, skipping upgrade: joblib in c:\users
\91704\anaconda3\lib\site-packages (from nltk) (0.13.2)
Installing collected packages: regex, nltk
  Found existing installation: nltk 3.4.4
    Uninstalling nltk-3.4.4:
      Successfully uninstalled nltk-3.4.4
Successfully installed nltk-3.7 regex-2022.3.15
Note: you may need to restart the kernel to use updated packages.
```

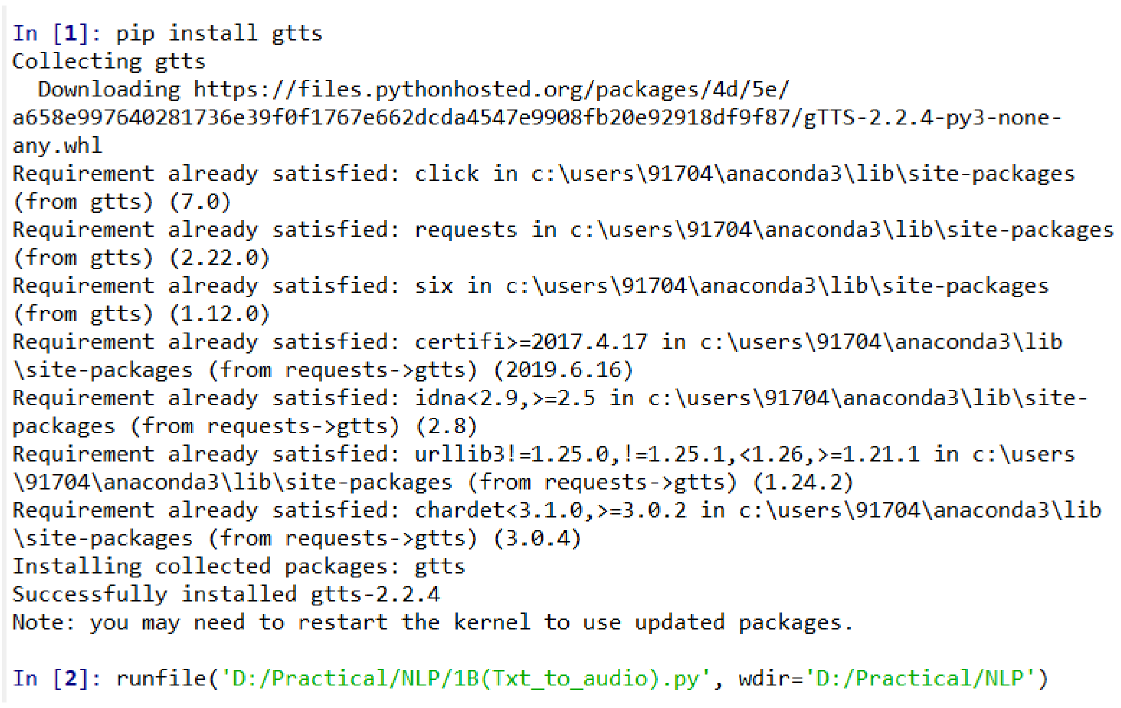
Practical 1B

Aim: To convert a text (sentence or file) to an audio outputand play the same.

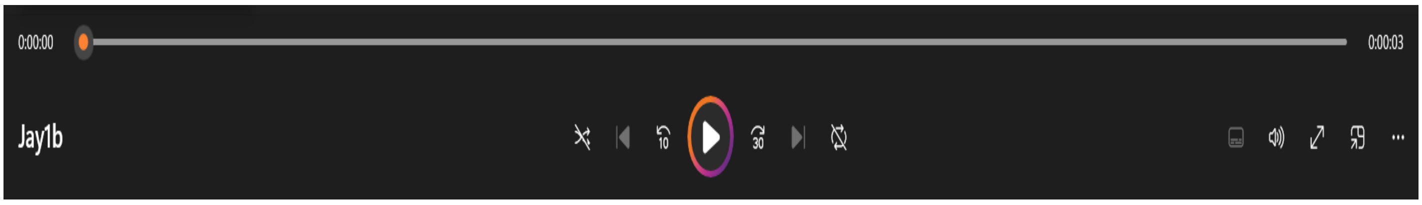
Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
Sapid: 53004200018
"""
from gtts import gTTS
# This module is imported so that we can
# play the converted audio
import os
# The text that you want to convert to audio
mytext = 'Practical of text to Speech performed by Ankit Patel'
# Language in which you want to convert
language = 'en'
# Passing the text and language to the engine,
# here we have marked slow=False. Which tells
# the module that the converted audio should
# have a high speed
myobj = gTTS(text=mytext, lang=language, slow=False)
# Saving the converted audio in a mp3 file named
# welcome
myobj.save("Ankit1b.wav")
# Playing the converted file
os.system("Ankit1b.wav")
```

Output:



```
In [1]: pip install gtts
Collecting gtts
  Downloading https://files.pythonhosted.org/packages/4d/5e/
a658e997640281736e39f0f1767e662dcda4547e9908fb20e92918df9f87/gTTS-2.2.4-py3-none-
any.whl
Requirement already satisfied: click in c:\users\91704\anaconda3\lib\site-packages
(from gtts) (7.0)
Requirement already satisfied: requests in c:\users\91704\anaconda3\lib\site-packages
(from gtts) (2.22.0)
Requirement already satisfied: six in c:\users\91704\anaconda3\lib\site-packages
(from gtts) (1.12.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\91704\anaconda3\lib
\site-packages (from requests->gtts) (2019.6.16)
Requirement already satisfied: idna<2.9,>=2.5 in c:\users\91704\anaconda3\lib\site-
packages (from requests->gtts) (2.8)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users
\91704\anaconda3\lib\site-packages (from requests->gtts) (1.24.2)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\users\91704\anaconda3\lib
\site-packages (from requests->gtts) (3.0.4)
Installing collected packages: gtts
Successfully installed gtts-2.2.4
Note: you may need to restart the kernel to use updated packages.

In [2]: runfile('D:/Practical/NLP/1B(Txt_to_audio).py', wdir='D:/Practical/NLP')
```



Practical 1C

Aim: To convert a given audio file to a readable text and print the text.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
Sapid: 53004200018
"""
import speech_recognition as sr
filename="Audio.wav"
r = sr.Recognizer()
with sr.AudioFile(filename) as source:
   audio = r.record(source)
try:
   Text = r.recognize_google(audio)
   print("Text: "+Text)
except Exception as e:
   print("Exception: "+str(e))
```

Output:

```
In [16]: pip install SpeechRecognition pydub
Collecting SpeechRecognition
  Downloading https://files.pythonhosted.org/packages/26/
e1/7f5678cd94ec1234269d23756dbdaa4c8cfaed973412f88ae8adf7893a50/
SpeechRecognition-3.8.1-py2.py3-none-any.whl (32.8MB)
Collecting pydub
  Downloading https://files.pythonhosted.org/packages/a6/53/
d78dc063216e62fc55f6b2eebb447f6a4b0a59f55c8406376f76bf959b08/pydub-0.25.1-py2.py3-
none-any.whl
Installing collected packages: SpeechRecognition, pydub
Successfully installed SpeechRecognition-3.8.1 pydub-0.25.1
Note: you may need to restart the kernel to use updated packages.

In [17]: runfile('D:/Practical/NLP/1C(audio_to_Txt).py', wdir='D:/Practical/NLP')
```

```
Text: summary the sides to break a teacher for the you keep adequate coverage the
works of places to save money baby is taking longer to getting squared away then the
bank was expected during the life events company in AVN heartattack se retirement
income the British were inadequate news of the saving lives are heard it has done
that you naked Bond what a discussion can insert when the title of this type of song
is in question or waxing or gasing needed I prevent my be personalized now back work
lace leather and lace work on a flat surface and smooths out this post and a separate
system uses a single sirf contained Unity op shop at store holds a good mechanical
isliye bad bus figures good Gauhar in late summer curable chairs cabinets chest down
house is a set
```

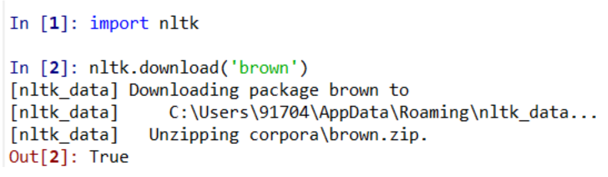
Practical 2A

Aim: Study of various Corpus – Brown, Inaugural, Reuters, udhr with various methods like fields, raw, words, sents, categories.

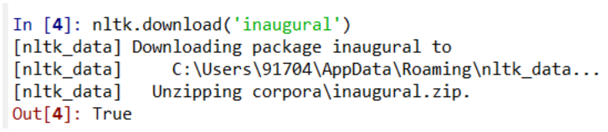
Code:

```
from nltk.corpus import brown
print(brown.categories())
print(brown.words(categories='news'))
print(brown.words(fileids=['cg22']))
print(brown.sents(categories=['news', 'editorial', 'reviews']))
from nltk.corpus import inaugural
print(inaugural.fileids())
print(inaugural.words()) #print(inaugural.categories())
from nltk.corpus import reuters
print(reuters.fileids())
print(reuters.words())
from nltk.corpus import udhr
print(udhr.fileids())
print(udhr.words())
```

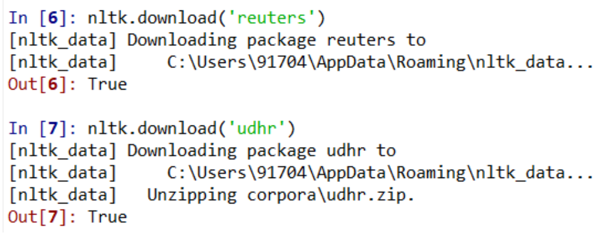
Output:



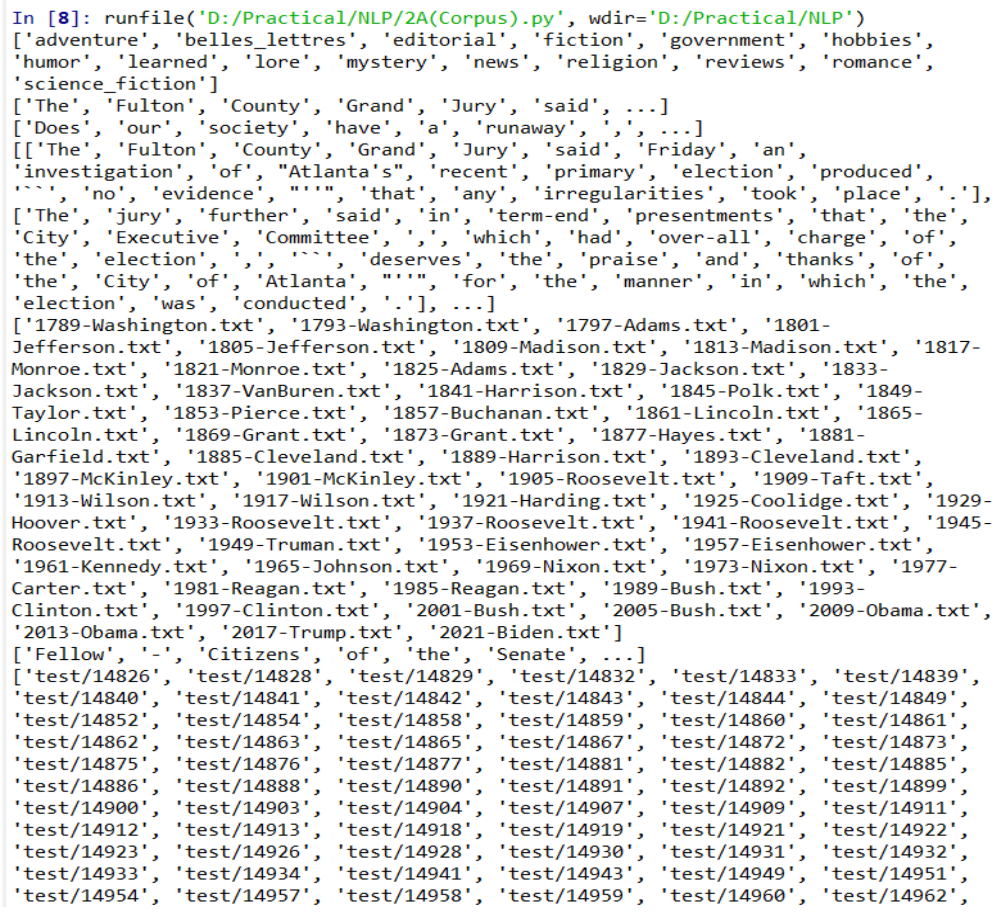
```
In [1]: import nltk

In [2]: nltk.download('brown')
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\brown.zip.
Out[2]: True
```



```
In [4]: nltk.download('inaugural')
[nltk_data] Downloading package inaugural to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\inaugural.zip.
Out[4]: True
```



```
In [6]: nltk.download('reuters')
[nltk_data] Downloading package reuters to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
Out[6]: True

In [7]: nltk.download('udhr')
[nltk_data] Downloading package udhr to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\udhr.zip.
Out[7]: True
```



```
In [8]: runfile('D:/Practical/NLP/2A(Corpus).py', wdir='D:/Practical/NLP')
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies',
'humor', 'learned', 'lore', 'mystery', 'news', 'religion', 'reviews', 'romance',
'science_fiction']
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
['Does', 'our', 'society', 'have', 'a', 'runaway', ',', ...]
[['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', '.'], ...]
['1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt', '1801-
Jefferson.txt', '1805-Jefferson.txt', '1809-Madison.txt', '1813-Madison.txt', '1817-
Monroe.txt', '1821-Monroe.txt', '1825-Adams.txt', '1829-Jackson.txt', '1833-
Jackson.txt', '1837-VanBuren.txt', '1841-Harrison.txt', '1845-Polk.txt', '1849-
Taylor.txt', '1853-Pierce.txt', '1857-Buchanan.txt', '1861-Lincoln.txt', '1865-
Lincoln.txt', '1869-Grant.txt', '1873-Grant.txt', '1877-Hayes.txt', '1881-
Garfield.txt', '1885-Cleveland.txt', '1889-Harrison.txt', '1893-Cleveland.txt',
'1897-McKinley.txt', '1901-McKinley.txt', '1905-Roosevelt.txt', '1909-Taft.txt',
'1913-Wilson.txt', '1917-Wilson.txt', '1921-Harding.txt', '1925-Coolidge.txt', '1929-
Hoover.txt', '1933-Roosevelt.txt', '1937-Roosevelt.txt', '1941-Roosevelt.txt', '1945-
Roosevelt.txt', '1949-Truman.txt', '1953-Eisenhower.txt', '1957-Eisenhower.txt',
'1961-Kennedy.txt', '1965-Johnson.txt', '1969-Nixon.txt', '1973-Nixon.txt', '1977-
Carter.txt', '1981-Reagan.txt', '1985-Reagan.txt', '1989-Bush.txt', '1993-
Clinton.txt', '1997-Clinton.txt', '2001-Bush.txt', '2005-Bush.txt', '2009-Obama.txt',
'2013-Obama.txt', '2017-Trump.txt', '2021-Biden.txt']
['Fellow', '-', 'Citizens', 'of', 'the', 'Senate', ...]
['test/14826', 'test/14828', 'test/14829', 'test/14832', 'test/14833', 'test/14839',
'test/14840', 'test/14841', 'test/14842', 'test/14843', 'test/14844', 'test/14849',
'test/14852', 'test/14854', 'test/14858', 'test/14859', 'test/14860', 'test/14861',
'test/14862', 'test/14863', 'test/14865', 'test/14867', 'test/14872', 'test/14873',
'test/14875', 'test/14876', 'test/14877', 'test/14881', 'test/14882', 'test/14885',
'test/14886', 'test/14888', 'test/14890', 'test/14891', 'test/14892', 'test/14899',
'test/14900', 'test/14903', 'test/14904', 'test/14907', 'test/14909', 'test/14911',
'test/14912', 'test/14913', 'test/14918', 'test/14919', 'test/14921', 'test/14922',
'test/14923', 'test/14926', 'test/14928', 'test/14930', 'test/14931', 'test/14932',
'test/14933', 'test/14934', 'test/14941', 'test/14943', 'test/14949', 'test/14951',
'test/14954', 'test/14957', 'test/14958', 'test/14959', 'test/14960', 'test/14962',
```

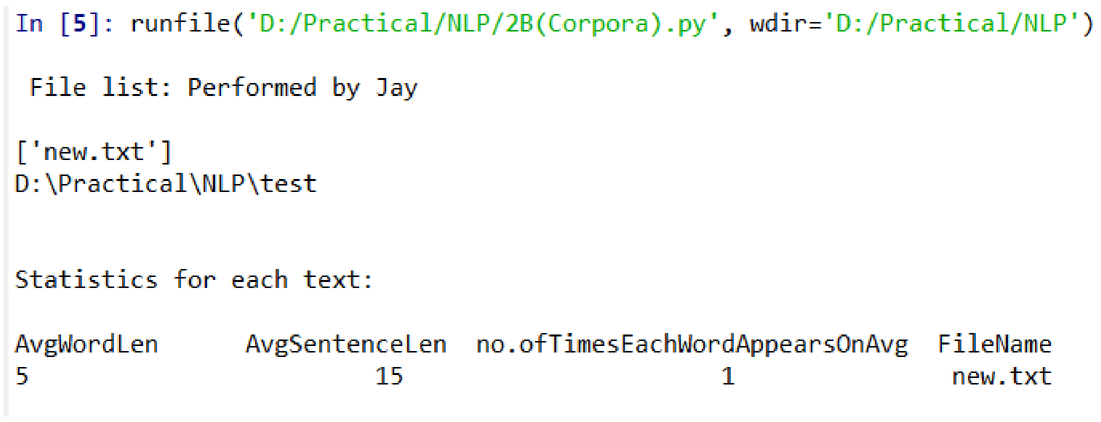
Practical 2B

Aim: Create and use your own corpora (plaintext, categorical)

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
Sapid: 53004200018
"""
from nltk.corpus import PlaintextCorpusReader
corpus_root = 'D:/Practical/NLP/test'
filelist = PlaintextCorpusReader(corpus_root, '.*')
print ('\n File list: Performed by Ankit \n')
print (filelist.fileids())
print (filelist.root)
print ('\n\nStatistics for each text:\n')
print
('AvgWordLen\tAvgSentenceLen\tno.ofTimesEachWordAppearsOnAvg\tFileN
ame')
for fileid in filelist.fileids():
   num_chars = len(filelist.raw(fileid))
   num_words = len(filelist.words(fileid))
   num_sents = len(filelist.sents(fileid))
   num_vocab = len(set([w.lower() for w in filelist.words(fileid)]))
   print (int(num_chars/num_words),'\t\t\t', int(num_words/num_sents),'\t\t\t',
int(num_words/num_vocab),'\t\t', fileid)
```

Output:



```
In [5]: runfile('D:/Practical/NLP/2B(Corpora).py', wdir='D:/Practical/NLP')

 File list: Performed by Jay

['new.txt']
D:\Practical\NLP\test


Statistics for each text:

AvgWordLen      AvgSentenceLen  no.ofTimesEachWordAppearsOnAvg  FileName
5                          15                              1          new.txt
```

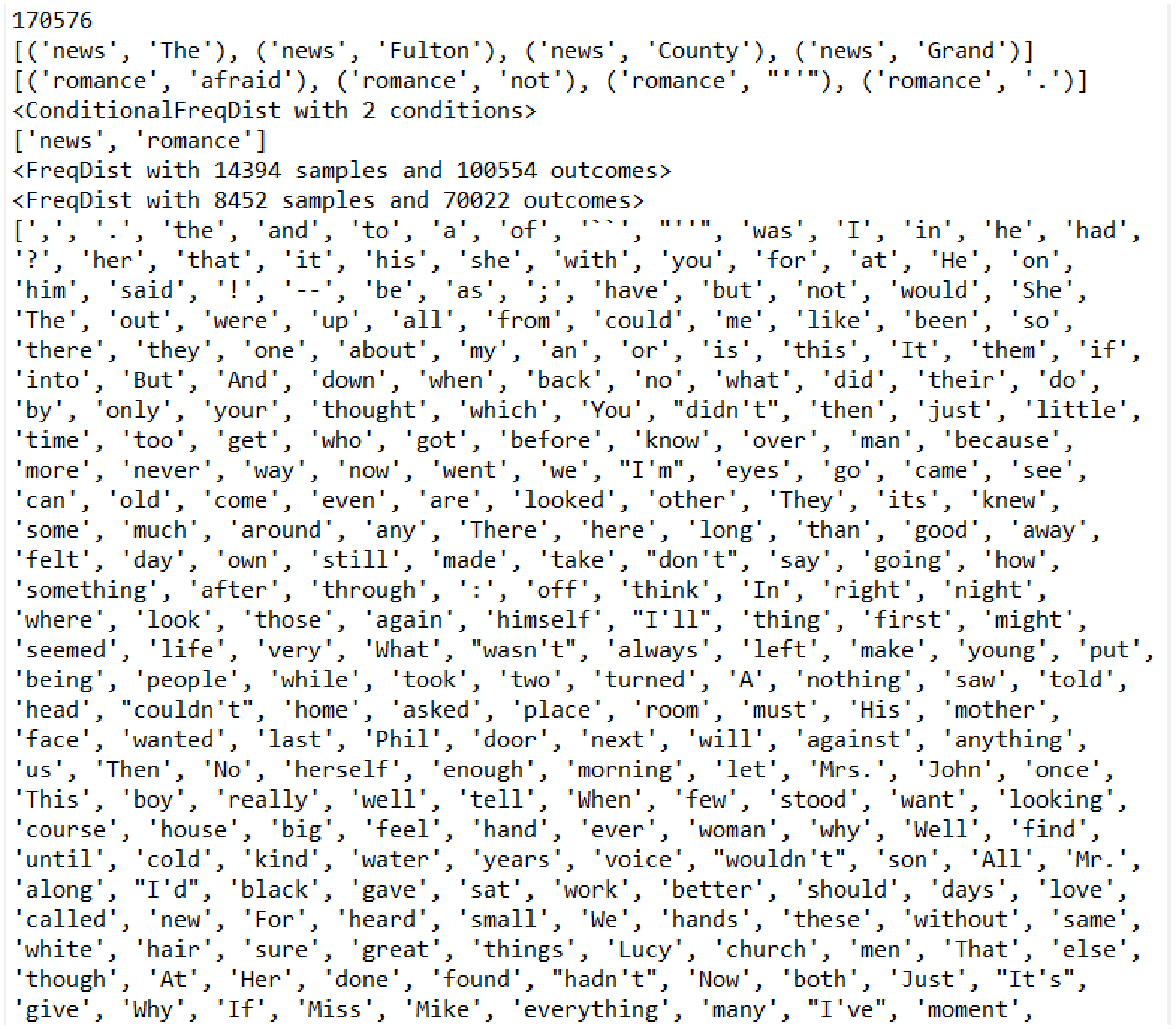
Practical 2C

Aim: Study Conditional frequency distributions.

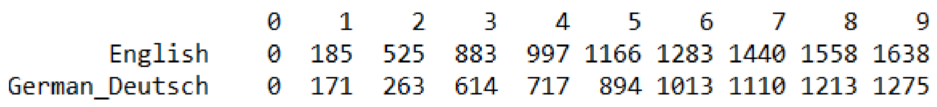
Code:

```
"""
@author: Ankit Patel
Sapid: 53004200018
"""
#process a sequence of pairs
text = ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
pairs = [('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ...]
import nltk
from nltk.corpus import brown
fd = nltk.ConditionalFreqDist((genre, word)
for genre in brown.categories()
for word in brown.words(categories=genre))
genre_word = [(genre, word)
for genre in ['news', 'romance']
for word in brown.words(categories=genre)]
print(len(genre_word))
print(genre_word[:4])
print(genre_word[-4:])
cfd = nltk.ConditionalFreqDist(genre_word)
print(cfd)
print(cfd.conditions())
print(cfd['news'])
print(cfd['romance'])
print(list(cfd['romance']))
from nltk.corpus import inaugural
cfd = nltk.ConditionalFreqDist(
(target, fileid[:4])
for fileid in inaugural.fileids()
for w in inaugural.words(fileid)
for target in ['america', 'citizen']
if w.lower().startswith(target))
from nltk.corpus import udhr
languages = ['Chickasaw', 'English', 'German_Deutsch',
'Greenlandic_Inuktikut', 'Hungarian_Magyar', 'Ibibio_Efik']
cfd = nltk.ConditionalFreqDist(
(lang, len(word))
for lang in languages
for word in udhr.words(lang + '-Latin1'))
cfd.tabulate(conditions=['English', 'German_Deutsch'],
samples=range(10), cumulative=True)
```

Output:



```
170576
[('news', 'The'), ('news', 'Fulton'), ('news', 'County'), ('news', 'Grand')]
[('romance', 'afraid'), ('romance', 'not'), ('romance', "''"), ('romance', '.')]
<ConditionalFreqDist with 2 conditions>
['news', 'romance']
<FreqDist with 14394 samples and 100554 outcomes>
<FreqDist with 8452 samples and 70022 outcomes>
[',', '.', 'the', 'and', 'to', 'a', 'of', '``', "''", 'was', 'I', 'in', 'he', 'had',
'?', 'her', 'that', 'it', 'his', 'she', 'with', 'you', 'for', 'at', 'He', 'on',
'him', 'said', '!', '--', 'be', 'as', ';', 'have', 'but', 'not', 'would', 'She',
'The', 'out', 'were', 'up', 'all', 'from', 'could', 'me', 'like', 'been', 'so',
'there', 'they', 'one', 'about', 'my', 'an', 'or', 'is', 'this', 'It', 'them', 'if',
'into', 'But', 'And', 'down', 'when', 'back', 'no', 'what', 'did', 'their', 'do',
'by', 'only', 'your', 'thought', 'which', 'You', "didn't", 'then', 'just', 'little',
'time', 'too', 'get', 'who', 'got', 'before', 'know', 'over', 'man', 'because',
'more', 'never', 'way', 'now', 'went', 'we', "I'm", 'eyes', 'go', 'came', 'see',
'can', 'old', 'come', 'even', 'are', 'looked', 'other', 'They', 'its', 'knew',
'some', 'much', 'around', 'any', 'There', 'here', 'long', 'than', 'good', 'away',
'felt', 'day', 'own', 'still', 'made', 'take', "don't", 'say', 'going', 'how',
'something', 'after', 'through', ':', 'off', 'think', 'In', 'right', 'night',
'where', 'look', 'those', 'again', 'himself', "I'll", 'thing', 'first', 'might',
'seemed', 'life', 'very', 'What', "wasn't", 'always', 'left', 'make', 'young', 'put',
'being', 'people', 'while', 'took', 'two', 'turned', 'A', 'nothing', 'saw', 'told',
'head', "couldn't", 'home', 'asked', 'place', 'room', 'must', 'His', 'mother',
'face', 'wanted', 'last', 'Phil', 'door', 'next', 'will', 'against', 'anything',
'us', 'Then', 'No', 'herself', 'enough', 'morning', 'let', 'Mrs.', 'John', 'once',
'This', 'boy', 'really', 'well', 'tell', 'When', 'few', 'stood', 'want', 'looking',
'course', 'house', 'big', 'feel', 'hand', 'ever', 'woman', 'why', 'Well', 'find',
'until', 'cold', 'kind', 'water', 'years', 'voice', "wouldn't", 'son', 'All', 'Mr.',
'along', "I'd", 'black', 'gave', 'sat', 'work', 'better', 'should', 'days', 'love',
'called', 'new', 'For', 'heard', 'small', 'We', 'hands', 'these', 'without', 'same',
'white', 'hair', 'sure', 'great', 'things', 'Lucy', 'church', 'men', 'That', 'else',
'though', 'At', 'Her', 'done', 'found', "hadn't", 'Now', 'both', 'Just', "It's",
'give', 'Why', 'If', 'Miss', 'Mike', 'everything', 'many', "I've", 'moment',
```



```
                  0    1    2    3    4    5    6    7    8    9
       English    0  185  525  883  997 1166 1283 1440 1558 1638
German_Deutsch    0  171  263  614  717  894 1013 1110 1213 1275
```

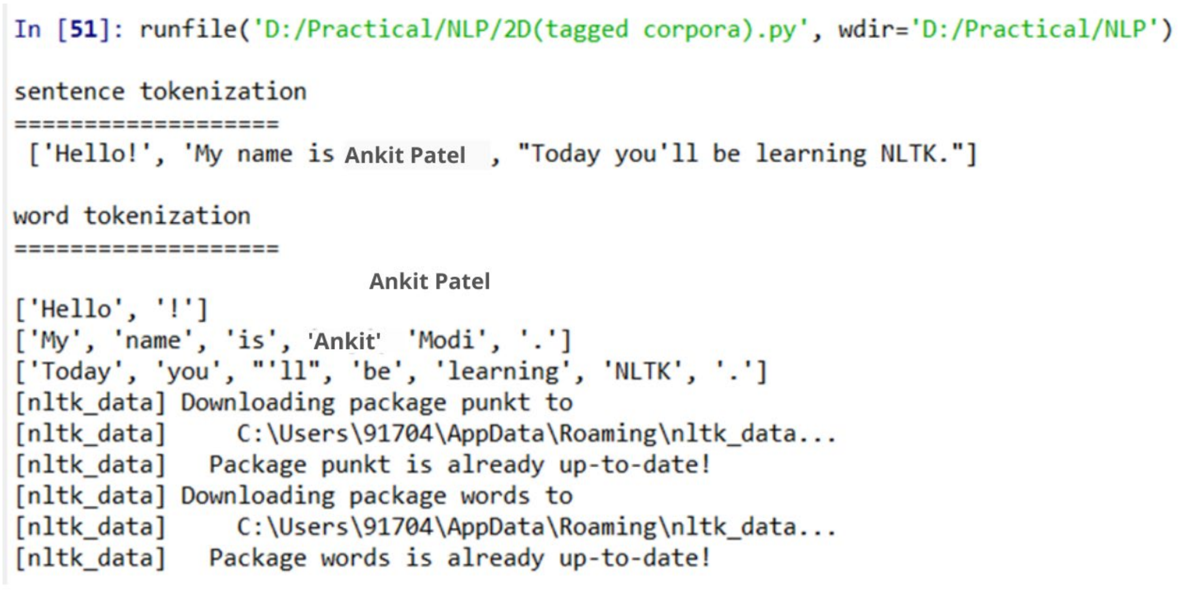
Practical 2D

Aim: Study of tagged corpora with methods like tagged_sents, tagged_words.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
Sapid : 53004200018
"""
import nltk
from nltk import tokenize
nltk.download('punkt')
nltk.download('words')
para = "Hello! My name is Ankit Patel. Today you'll be learning NLTK."
sents = tokenize.sent_tokenize(para)
print("\nsentence tokenization\n===================\n",sents)
# word tokenization
print("\nword tokenization\n===================\n")
for index in range(len(sents)):
   words = tokenize.word_tokenize(sents[index])
   print(words)
```

Output:



```
In [51]: runfile('D:/Practical/NLP/2D(tagged corpora).py', wdir='D:/Practical/NLP')

sentence tokenization
===================
 ['Hello!', 'My name is Ankit Patel , "Today you'll be learning NLTK."]

word tokenization
===================
                          Ankit Patel
['Hello', '!']
['My', 'name', 'is', 'Ankit'  'Modi', '.']
['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
```

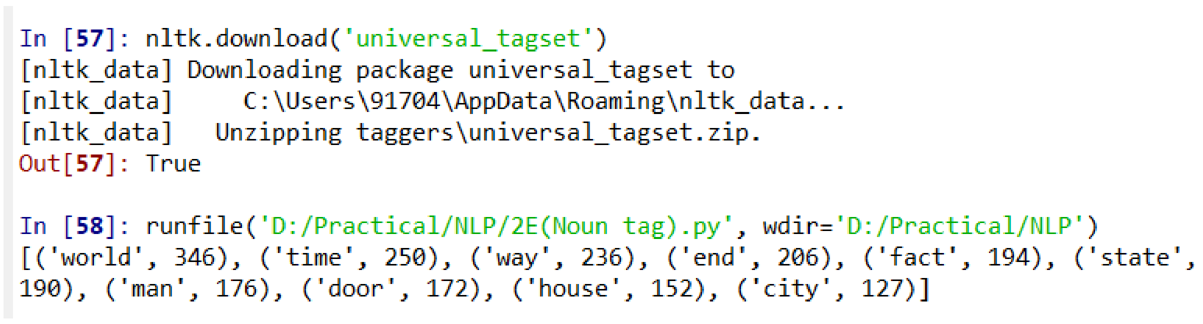
Practical 2E

Aim: Write a program to find the most frequent noun tags

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
from nltk.corpus import brown
tagged = brown.tagged_words(tagset='universal')
noundist = nltk.FreqDist(w2 for ((w1, t1), (w2, t2)) in
nltk.bigrams(brown.tagged_words(tagset="universal"))
if w1.lower() == "the" and t2 == "NOUN")
print(noundist.most_common(10))
```

Output:



```
In [57]: nltk.download('universal_tagset')
[nltk_data] Downloading package universal_tagset to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping taggers\universal_tagset.zip.
Out[57]: True

In [58]: runfile('D:/Practical/NLP/2E(Noun tag).py', wdir='D:/Practical/NLP')
[('world', 346), ('time', 250), ('way', 236), ('end', 206), ('fact', 194), ('state',
190), ('man', 176), ('door', 172), ('house', 152), ('city', 127)]
```

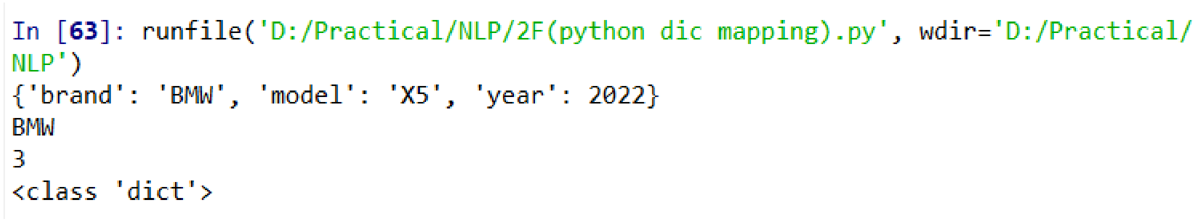
Practical 2F

Aim: Map Words to Properties Using Python Dictionaries

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
#creating and printing a dictionay by mapping word with its properties
thisdict = {
"brand": "BMW",
"model": "X5",
"year": 2022
}
print(thisdict)
print(thisdict["brand"])
print(len(thisdict))
print(type(thisdict))
```

Output:



```
In [63]: runfile('D:/Practical/NLP/2F(python dic mapping).py', wdir='D:/Practical/
NLP')
{'brand': 'BMW', 'model': 'X5', 'year': 2022}
BMW
3
<class 'dict'>
```

Practical 2G

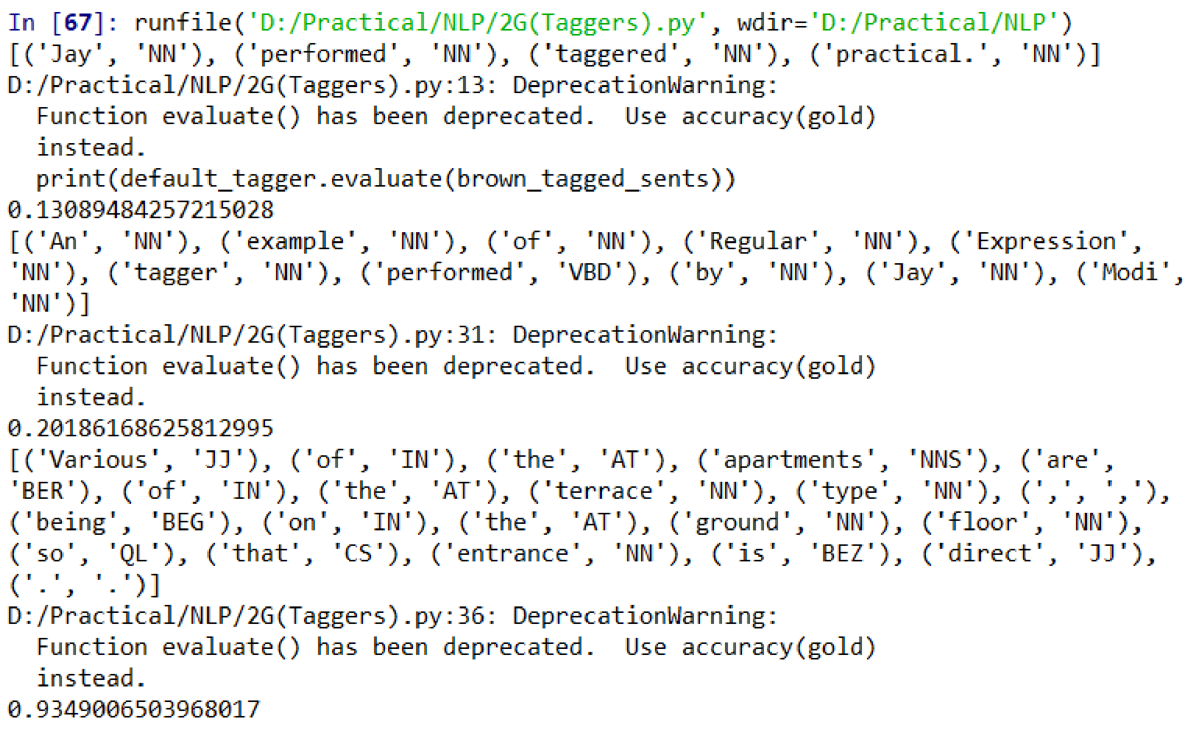
Aim: Study DefaultTagger, Regular expression tagger, UnigramTagger

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
from nltk.corpus import brown
tokens = 'Ankit performed taggered practical.'.split()
default_tagger = nltk.DefaultTagger('NN')
print(default_tagger.tag(tokens))
brown_tagged_sents = brown.tagged_sents(categories='news')
print(default_tagger.evaluate(brown_tagged_sents))
patterns = [
(r'.*ing$', 'VBG'), # gerunds
(r'.*ed$', 'VBD'), # simple past
(r'.*es$', 'VBZ'), # 3rd singular present
(r'.*ould$', 'MD'), # modals
(r'.*\'s$', 'NN$'), # possessive nouns
(r'.*s$', 'NNS'), # plural nouns
(r'^-?[0-9]+(\.[0-9]+)?$', 'CD'), # cardinal numbers
(r'.*', 'NN') # nouns (default)
]
regexp_tagger = nltk.RegexpTagger(patterns)
```

```
exp=(regexp_tagger.tag("An example of Regular Expression tagger performed
by Ankit Patel".split()))

print(exp)

brown_tagged_sents = brown.tagged_sents(categories='news')

print(regexp_tagger.evaluate(brown_tagged_sents))

brown_tagged_sents = brown.tagged_sents(categories='news')

brown_sents = brown.sents(categories='news')

unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)

print(unigram_tagger.tag(brown_sents[2007]))

print(unigram_tagger.evaluate(brown_tagged_sents)
```

Output:



```
In [67]: runfile('D:/Practical/NLP/2G(Taggers).py', wdir='D:/Practical/NLP')
[('Jay', 'NN'), ('performed', 'NN'), ('taggered', 'NN'), ('practical.', 'NN')]
D:/Practical/NLP/2G(Taggers).py:13: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  print(default_tagger.evaluate(brown_tagged_sents))
0.13089484257215028
[('An', 'NN'), ('example', 'NN'), ('of', 'NN'), ('Regular', 'NN'), ('Expression',
'NN'), ('tagger', 'NN'), ('performed', 'VBD'), ('by', 'NN'), ('Jay', 'NN'), ('Modi',
'NN')]
D:/Practical/NLP/2G(Taggers).py:31: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
0.20186168625812995
[('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'), ('apartments', 'NNS'), ('are',
'BER'), ('of', 'IN'), ('the', 'AT'), ('terrace', 'NN'), ('type', 'NN'), (',', ','),
('being', 'BEG'), ('on', 'IN'), ('the', 'AT'), ('ground', 'NN'), ('floor', 'NN'),
('so', 'QL'), ('that', 'CS'), ('entrance', 'NN'), ('is', 'BEZ'), ('direct', 'JJ'),
('.', '.')]
D:/Practical/NLP/2G(Taggers).py:36: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
0.9349006503968017
```

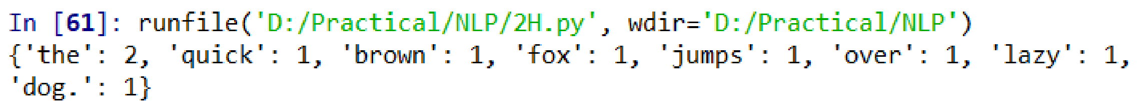
Practical 2H

Aim: Find different words from a given plain text without any space by comparing this text with a given corpus of words. Also find the score of words.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
Sapid: 53004200018"""
def word_count(str):
    counts = dict()
    str=str.lower()
    words = str.split()
    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1
    return counts
print( word_count('The Quick Brown fox jumps over the lazy dog.'))
```

Output:



```
In [61]: runfile('D:/Practical/NLP/2H.py', wdir='D:/Practical/NLP')
{'the': 2, 'quick': 1, 'brown': 1, 'fox': 1, 'jumps': 1, 'over': 1, 'lazy': 1,
'dog.': 1}
```

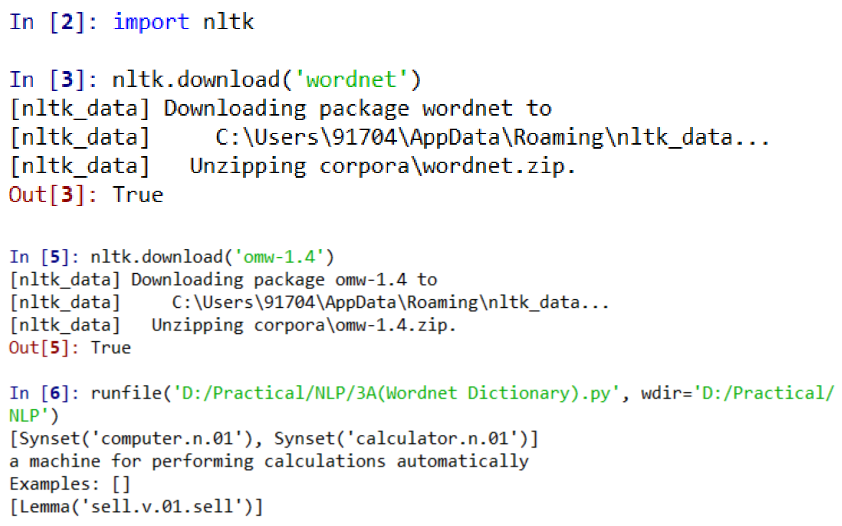
Practical 3A

Aim: Study of Wordnet Dictionary with methods as synsets, definitions, examples, antonyms.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
from nltk.corpus import wordnet
print(wordnet.synsets("computer"))
# definition and example of the word ‘computer’
print(wordnet.synset("computer.n.01").definition())
#examples
print("Examples:", wordnet.synset("computer.n.01").examples())
#get Antonyms
print(wordnet.lemma('buy.v.01.buy').antonyms())
```

Output:



```
In [2]: import nltk

In [3]: nltk.download('wordnet')
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\wordnet.zip.
Out[3]: True

In [5]: nltk.download('omw-1.4')
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\91704\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\omw-1.4.zip.
Out[5]: True

In [6]: runfile('D:/Practical/NLP/3A(Wordnet Dictionary).py', wdir='D:/Practical/
NLP')
[Synset('computer.n.01'), Synset('calculator.n.01')]
a machine for performing calculations automatically
Examples: []
[Lemma('sell.v.01.sell')]
```

Practical 3B

Aim: Study lemmas, hyponyms, hypernyms, entailments

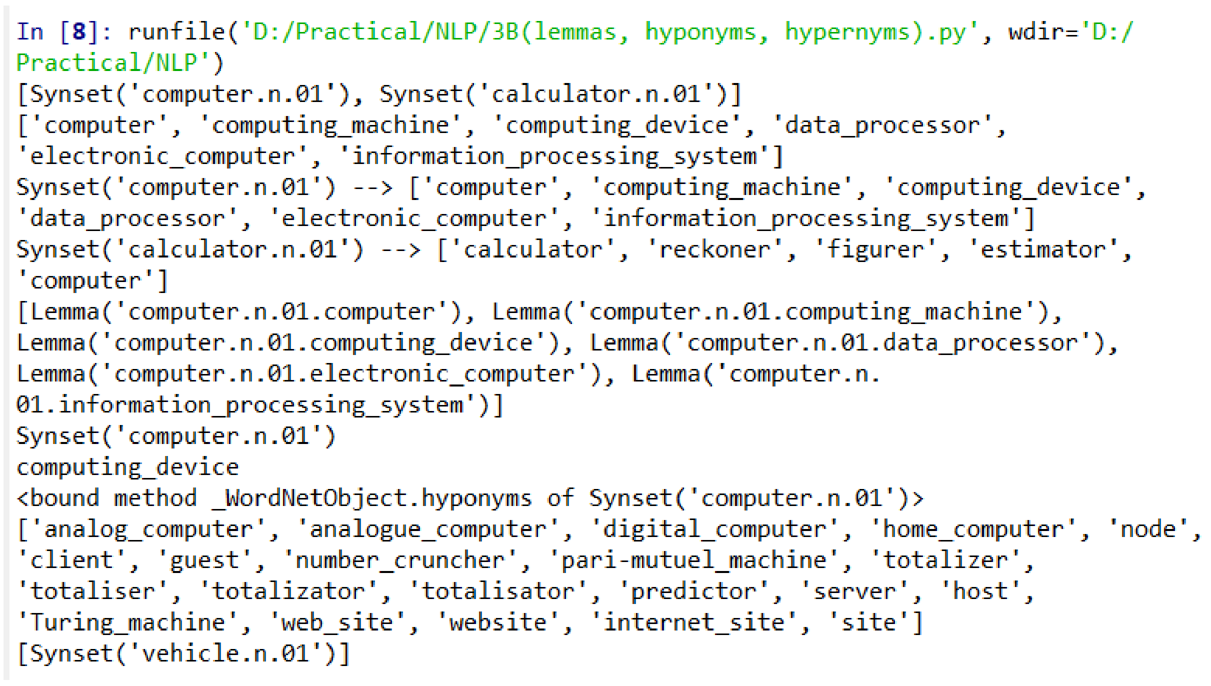
Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
from nltk.corpus import wordnet
print(wordnet.synsets("computer"))
print(wordnet.synset("computer.n.01").lemma_names())
#all lemmas for each synset.
for e in wordnet.synsets("computer"):
   print(f'{e} --> {e.lemma_names()}')
#print all lemmas for a given synset
print(wordnet.synset('computer.n.01').lemmas())
#get the synset corresponding to lemma
print(wordnet.lemma('computer.n.01.computing_device').synset())
#Get the name of the lemma
print(wordnet.lemma('computer.n.01.computing_device').name())
#Hyponyms give abstract concepts of the word that are much more specific
#the list of hyponyms words of the computer
syn = wordnet.synset('computer.n.01')
print(syn.hyponyms)
print([lemma.name() for synset in syn.hyponyms() for lemma in
synset.lemmas()])
#the semantic similarity in WordNet
vehicle = wordnet.synset('vehicle.n.01')
```

car = wordnet.synset('car.n.01')

print(car.lowest_common_hypernyms(vehicle))

Output:



```
In [8]: runfile('D:/Practical/NLP/3B(lemmas, hyponyms, hypernyms).py', wdir='D:/
Practical/NLP')
[Synset('computer.n.01'), Synset('calculator.n.01')]
['computer', 'computing_machine', 'computing_device', 'data_processor',
'electronic_computer', 'information_processing_system']
Synset('computer.n.01') --> ['computer', 'computing_machine', 'computing_device',
'data_processor', 'electronic_computer', 'information_processing_system']
Synset('calculator.n.01') --> ['calculator', 'reckoner', 'figurer', 'estimator',
'computer']
[Lemma('computer.n.01.computer'), Lemma('computer.n.01.computing_machine'),
Lemma('computer.n.01.computing_device'), Lemma('computer.n.01.data_processor'),
Lemma('computer.n.01.electronic_computer'), Lemma('computer.n.
01.information_processing_system')]
Synset('computer.n.01')
computing_device
<bound method _WordNetObject.hyponyms of Synset('computer.n.01')>
['analog_computer', 'analogue_computer', 'digital_computer', 'home_computer', 'node',
'client', 'guest', 'number_cruncher', 'pari-mutuel_machine', 'totalizer',
'totaliser', 'totalizator', 'totalisator', 'predictor', 'server', 'host',
'Turing_machine', 'web_site', 'website', 'internet_site', 'site']
[Synset('vehicle.n.01')]
```

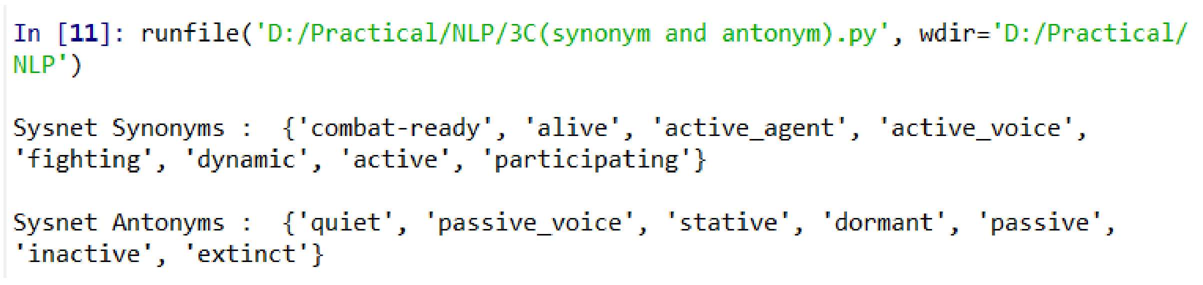
Practical 3C

Aim: Write a program using python to find synonym and antonym of word "active" using Wordnet

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
from nltk.corpus import wordnet
synonyms = []
antonyms = []
for syn in wordnet.synsets("active"):
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())
print("\nSysnet Synonyms : ",set(synonyms))
print("\nSysnet Antonyms : ",set(antonyms))
```

Output:



```
In [11]: runfile('D:/Practical/NLP/3C(synonym and antonym).py', wdir='D:/Practical/
NLP')

Sysnet Synonyms :  {'combat-ready', 'alive', 'active_agent', 'active_voice',
'fighting', 'dynamic', 'active', 'participating'}

Sysnet Antonyms :  {'quiet', 'passive_voice', 'stative', 'dormant', 'passive',
'inactive', 'extinct'}
```

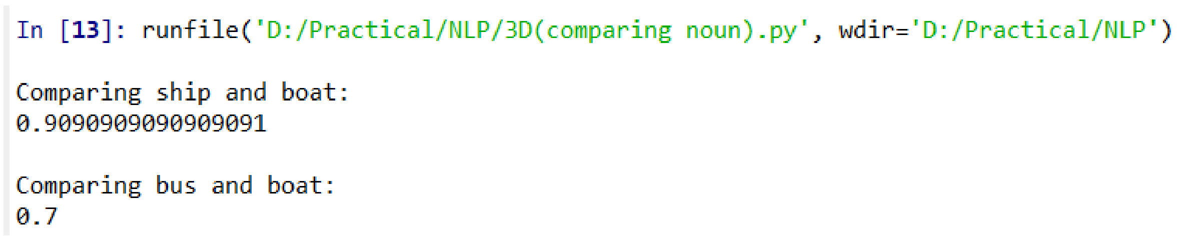
Practical 3D

Aim: Compare two nouns

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
from nltk.corpus import wordnet
print("\nComparing ship and boat:")
n1 = wordnet.synset('ship.n.01')
n2 = wordnet.synset('boat.n.01')
print(n1.wup_similarity(n2))
print("\nComparing bus and boat:")
n1 = wordnet.synset('bus.n.01')
n2 = wordnet.synset('boat.n.01')
print(n1.wup_similarity(n2))
```

Output:



```
In [13]: runfile('D:/Practical/NLP/3D(comparing noun).py', wdir='D:/Practical/NLP')

Comparing ship and boat:
0.9090909090909091

Comparing bus and boat:
0.7
```

Practical 3E

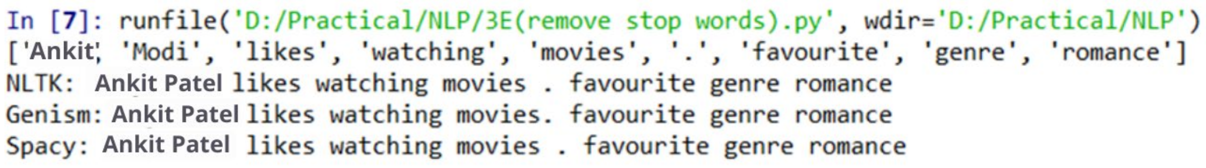
Aim: Handling stopword. Using nltk Adding or Removing Stop Words in NLTK's Default Stop Word List. Using Gensim Adding and Removing Stop Words in Default Gensim Stop Words List. Using Spacy Adding and Removing Stop Words in Default Spacy Stop Words List

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
text = "Ankit Patel likes watching movies. one of his most favourite genre is
romance"
text_tokens = word_tokenize(text)
tokens_without_sw = [word for word in text_tokens if not word in
stopwords.words()]
print(tokens_without_sw)
filtered_sentence = (" ").join(tokens_without_sw)
print("NLTK: ",filtered_sentence)
from gensim.parsing.preprocessing import remove_stopwords
filtered_sentence = remove_stopwords(text)
print("Genism:",filtered_sentence)
import spacy
sp = spacy.load('en_core_web_sm')
all_stopwords = sp.Defaults.stop_words
text_tokens = word_tokenize(text)
```

```
tokens_without_sw= [word for word in text_tokens if not word in all_stopwords]
filtered_sentence = (" ").join(tokens_without_sw)
print("Spacy: ",filtered_sentence)
```

Output:



```
In [7]: runfile('D:/Practical/NLP/3E(remove stop words).py', wdir='D:/Practical/NLP')
['Ankit', 'Modi', 'likes', 'watching', 'movies', '.', 'favourite', 'genre', 'romance']
NLTK:  Ankit Patel likes watching movies . favourite genre romance
Genism: Ankit Patel likes watching movies. favourite genre romance
Spacy: Ankit Patel likes watching movies . favourite genre romance
```

Practical 4

Aim: Text Tokenization

a. Tokenization using Python’s split() function

b. Tokenization using Regular Expressions (RegEx)

c. Tokenization using NLTK

d. Tokenization using the spaCy library

e. Tokenization using Keras

f. Tokenization using Gensim

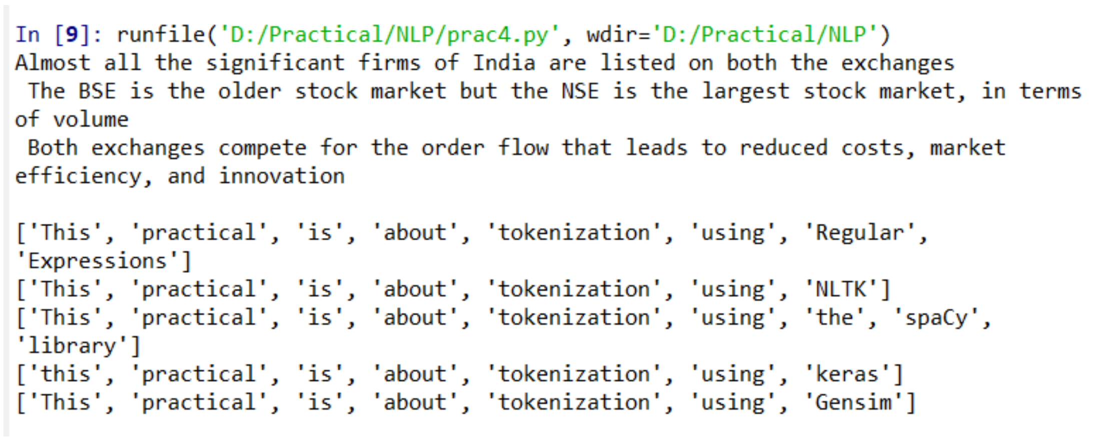
Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import spacy
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import word_tokenize
from keras.preprocessing.text import text_to_word_sequence
from gensim.utils import tokenize
#A. Tokenization using Python’s split() function
text = "Almost all the significant firms of India are listed on both the exchanges.
The BSE is the older stock market but the NSE is the largest stock market, in
terms of volume. Both exchanges compete for the order flow that leads to
reduced costs, market efficiency, and innovation. "
data = text.split('.')
for i in data:
    print (i)
```

```
#B. Tokenization using Regular Expressions (RegEx)
# Create a reference variable for Class RegexpTokenizer
tk = RegexpTokenizer('\s+', gaps = True)
# Create a string input
str = "This practical is about tokenization using Regular Expressions"
# Use tokenize method
tokens = tk.tokenize(str)
print(tokens)
#C. Tokenization using NLTK
# Create a string input
str = "This practical is about tokenization using NLTK "
# Use tokenize method
print(word_tokenize(str))
#D. Tokenization using the spaCy library
nlp = spacy.blank("en")
# Create a string input
str = "This practical is about tokenization using the spaCy library"
# Create an instance of document;
# doc object is a container for a sequence of Token objects.
doc = nlp(str)
# Read the words; Print the words
words = [word.text for word in doc]
print(words)

#E. Tokenization using Keras
# Create a string input
str = "This practical is about tokenization using Keras"
```

```
# tokenizing the text
tokens = text_to_word_sequence(str)
print(tokens)
#F. Tokenization using Gensim
# Create a string input
str = "This practical is about tokenization using Gensim"
# tokenizing the text
print(list(tokenize(str)))
```

Output:



```
In [9]: runfile('D:/Practical/NLP/prac4.py', wdir='D:/Practical/NLP')
Almost all the significant firms of India are listed on both the exchanges
 The BSE is the older stock market but the NSE is the largest stock market, in terms
of volume
 Both exchanges compete for the order flow that leads to reduced costs, market
efficiency, and innovation

['This', 'practical', 'is', 'about', 'tokenization', 'using', 'Regular',
'Expressions']
['This', 'practical', 'is', 'about', 'tokenization', 'using', 'NLTK']
['This', 'practical', 'is', 'about', 'tokenization', 'using', 'the', 'spaCy',
'library']
['this', 'practical', 'is', 'about', 'tokenization', 'using', 'keras']
['This', 'practical', 'is', 'about', 'tokenization', 'using', 'Gensim']
```

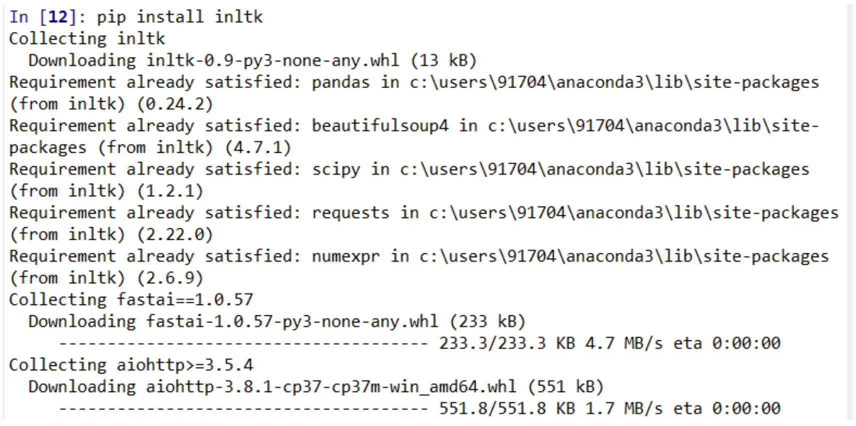
Practical 5A

Aim: Import NLP Libraries for Indian Languages and perform: Word tokenization in Hindi

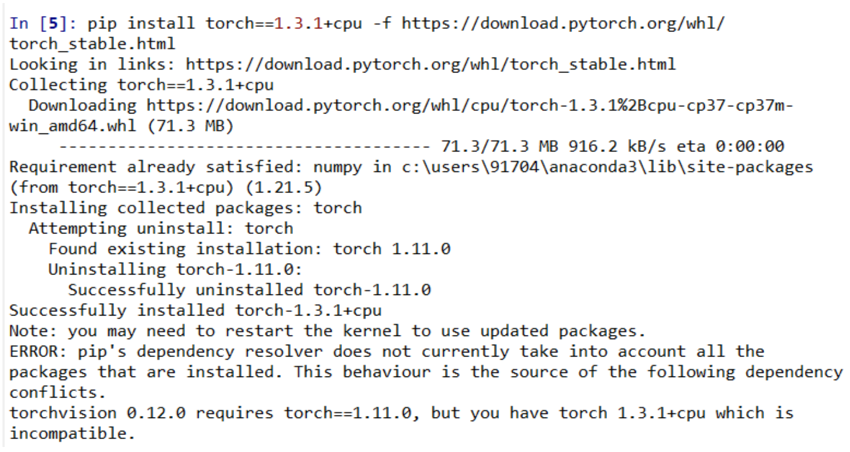
Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
from inltk.inltk import tokenize
from inltk.inltk import setup
setup('hi')
hindi_text = "यह प्रयोग जय द्वारा किया जाता है"
# tokenize(input text, language code)
print(tokenize(hindi_text, "hi"))
```

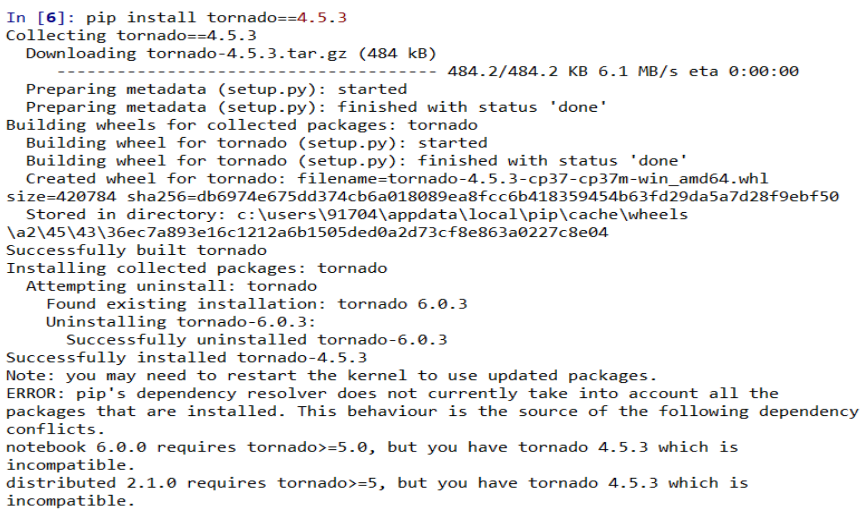
Output:



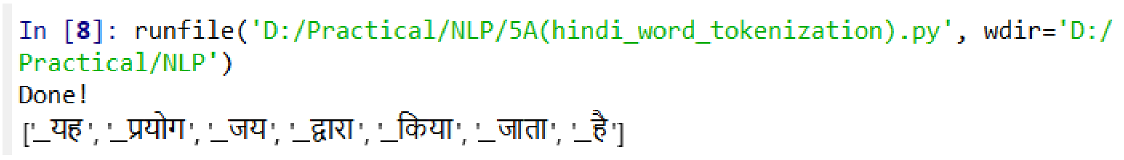
```
In [12]: pip install inltk
Collecting inltk
  Downloading inltk-0.9-py3-none-any.whl (13 kB)
Requirement already satisfied: pandas in c:\users\91704\anaconda3\lib\site-packages
(from inltk) (0.24.2)
Requirement already satisfied: beautifulsoup4 in c:\users\91704\anaconda3\lib\site-
packages (from inltk) (4.7.1)
Requirement already satisfied: scipy in c:\users\91704\anaconda3\lib\site-packages
(from inltk) (1.2.1)
Requirement already satisfied: requests in c:\users\91704\anaconda3\lib\site-packages
(from inltk) (2.22.0)
Requirement already satisfied: numexpr in c:\users\91704\anaconda3\lib\site-packages
(from inltk) (2.6.9)
Collecting fastai==1.0.57
  Downloading fastai-1.0.57-py3-none-any.whl (233 kB)
     -------------------------------------- 233.3/233.3 KB 4.7 MB/s eta 0:00:00
Collecting aiohttp>=3.5.4
  Downloading aiohttp-3.8.1-cp37-cp37m-win_amd64.whl (551 kB)
     -------------------------------------- 551.8/551.8 KB 1.7 MB/s eta 0:00:00
```



```
In [5]: pip install torch==1.3.1+cpu -f https://download.pytorch.org/whl/
torch_stable.html
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.3.1+cpu
  Downloading https://download.pytorch.org/whl/cpu/torch-1.3.1%2Bcpu-cp37-cp37m-
win_amd64.whl (71.3 MB)
     -------------------------------------- 71.3/71.3 MB 916.2 kB/s eta 0:00:00
Requirement already satisfied: numpy in c:\users\91704\anaconda3\lib\site-packages
(from torch==1.3.1+cpu) (1.21.5)
Installing collected packages: torch
  Attempting uninstall: torch
    Found existing installation: torch 1.11.0
    Uninstalling torch-1.11.0:
      Successfully uninstalled torch-1.11.0
Successfully installed torch-1.3.1+cpu
Note: you may need to restart the kernel to use updated packages.
ERROR: pip's dependency resolver does not currently take into account all the
packages that are installed. This behaviour is the source of the following dependency
conflicts.
torchvision 0.12.0 requires torch==1.11.0, but you have torch 1.3.1+cpu which is
incompatible.
```



```
In [6]: pip install tornado==4.5.3
Collecting tornado==4.5.3
  Downloading tornado-4.5.3.tar.gz (484 kB)
     -------------------------------------- 484.2/484.2 KB 6.1 MB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Building wheels for collected packages: tornado
  Building wheel for tornado (setup.py): started
  Building wheel for tornado (setup.py): finished with status 'done'
  Created wheel for tornado: filename=tornado-4.5.3-cp37-cp37m-win_amd64.whl
size=420784 sha256=db6974e675dd374cb6a018089ea8fcc6b418359454b63fd29da5a7d28f9ebf50
  Stored in directory: c:\users\91704\appdata\local\pip\cache\wheels
\a2\45\43\36ec7a893e16c1212a6b1505ded0a2d73cf8e863a0227c8e04
Successfully built tornado
Installing collected packages: tornado
  Attempting uninstall: tornado
    Found existing installation: tornado 6.0.3
    Uninstalling tornado-6.0.3:
      Successfully uninstalled tornado-6.0.3
Successfully installed tornado-4.5.3
Note: you may need to restart the kernel to use updated packages.
ERROR: pip's dependency resolver does not currently take into account all the
packages that are installed. This behaviour is the source of the following dependency
conflicts.
notebook 6.0.0 requires tornado>=5.0, but you have tornado 4.5.3 which is
incompatible.
distributed 2.1.0 requires tornado>=5, but you have tornado 4.5.3 which is
incompatible.
```



```
In [8]: runfile('D:/Practical/NLP/5A(hindi_word_tokenization).py', wdir='D:/
Practical/NLP')
Done!
['_यह', '_प्रयोग', '_जय', '_द्वारा', '_किया', '_जाता', '_है']
```

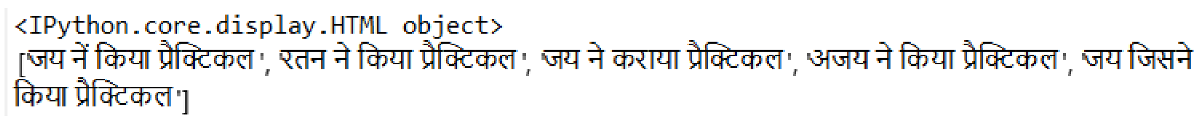
Practical 5B

Aim: Generate similar sentences from a given Hindi text input

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
from inltk.inltk import setup
setup('hi')
from inltk.inltk import get_similar_sentences
# get similar sentences to the one given in hindi
output = get_similar_sentences('जय ने किया प्रैक्टिकल', 5, 'hi')
print(output)
```

Output:



```
<IPython.core.display.HTML object>
['जय नें किया प्रैक्टिकल', 'रतन ने किया प्रैक्टिकल', 'जय ने कराया प्रैक्टिकल', 'अजय ने किया प्रैक्टिकल', 'जय जिसने
किया प्रैक्टिकल']
```

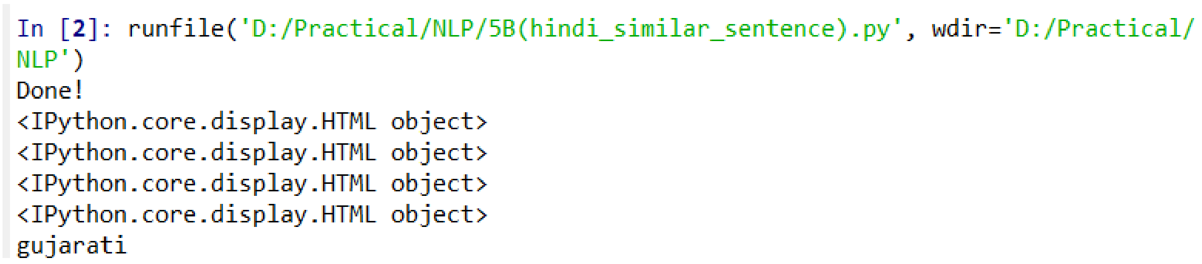
Practical 5C

Aim: Identify the Indian language of a text

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
from inltk.inltk import setup
setup('gu')
from inltk.inltk import identify_language
#Identify the Lnaguage of given text
print(identify_language('જય દ્વારા કરવામાં આવેલ પ્રાયોગિક'))
```

Output:



```
In [2]: runfile('D:/Practical/NLP/5B(hindi_similar_sentence).py', wdir='D:/Practical/
NLP')
Done!
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
gujarati
```

Practical 6A

Aim: Illustrate part of speech tagging.

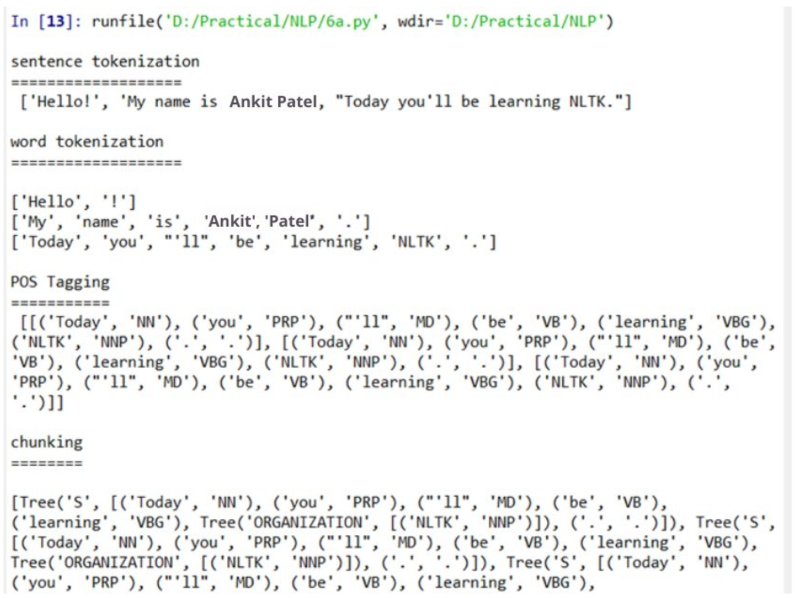
A. Part of speech Tagging and chunking of user defined text.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
from nltk import tokenize
nltk.download('punkt')
from nltk import tag
from nltk import chunk
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
para = "Hello! My name is Ankit Patel. Today you'll be learning NLTK."
sents = tokenize.sent_tokenize(para)
print("\nsentence tokenization\n===================\n",sents)
# word tokenization
print("\nword tokenization\n===================\n")
for index in range(len(sents)):
 words = tokenize.word_tokenize(sents[index])
 print(words)
# POS Tagging
tagged_words = []
```

```
for index in range(len(sents)):
 tagged_words.append(tag.pos_tag(words))
print("\nPOS Tagging\n===========\n",tagged_words)
# chunking
tree = []
for index in range(len(sents)):
 tree.append(chunk.ne_chunk(tagged_words[index]))
print("\nchunking\n========\n")
print(tree)
```

Output:



```
In [13]: runfile('D:/Practical/NLP/6a.py', wdir='D:/Practical/NLP')

sentence tokenization
===================
 ['Hello!', 'My name is Ankit Patel, "Today you'll be learning NLTK."]

word tokenization
===================

['Hello', '!']
['My', 'name', 'is', 'Ankit', 'Patel', '.']
['Today', 'you', "'ll", 'be', 'learning', 'NLTK', '.']

POS Tagging
===========
 [[('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'),
('NLTK', 'NNP'), ('.', '.')], [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be',
'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.', '.')], [('Today', 'NN'), ('you',
'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'), ('NLTK', 'NNP'), ('.',
'.')]]

chunking
========

[Tree('S', [('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'),
('learning', 'VBG'), Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')]), Tree('S',
[('Today', 'NN'), ('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'),
Tree('ORGANIZATION', [('NLTK', 'NNP')]), ('.', '.')]), Tree('S', [('Today', 'NN'),
('you', 'PRP'), ("'ll", 'MD'), ('be', 'VB'), ('learning', 'VBG'),
```

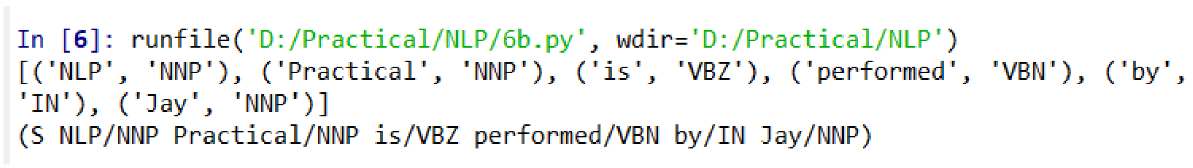
Practical 6B

Aim: Named Entity recognition of user defined text.

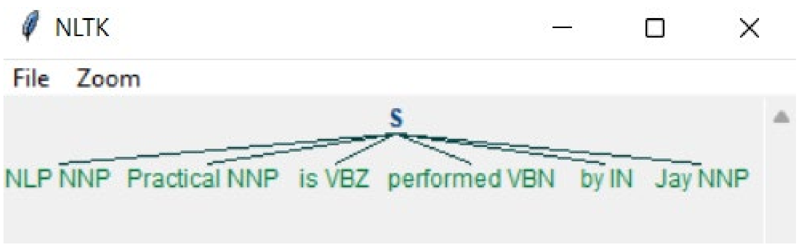
Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
ex = 'NLP Practical is performed by Ankit'
def preprocess(sent):
    sent = nltk.word_tokenize(sent)
    sent = nltk.pos_tag(sent)
    return sent
sent = preprocess(ex)
print(sent)
pattern = 'NP: {<DT>?<JJ>*<NN>}'
cp = nltk.RegexpParser(pattern)
cs = cp.parse(sent)
print(cs)
cs.draw()
```

Output:



```
In [6]: runfile('D:/Practical/NLP/6b.py', wdir='D:/Practical/NLP')
[('NLP', 'NNP'), ('Practical', 'NNP'), ('is', 'VBZ'), ('performed', 'VBN'), ('by',
'IN'), ('Jay', 'NNP')]
(S NLP/NNP Practical/NNP is/VBZ performed/VBN by/IN Jay/NNP)
```



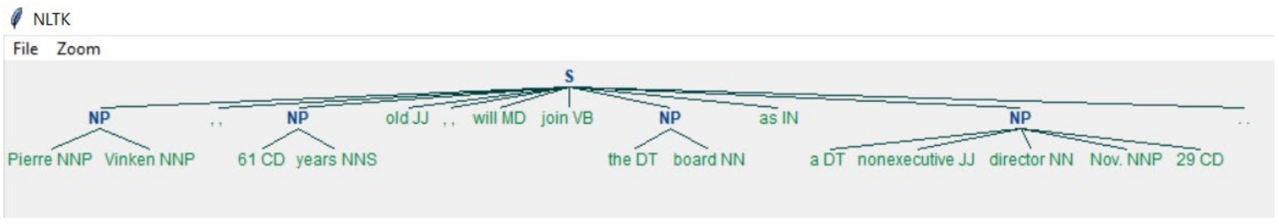
Practical 6C

Aim: Named Entity recognition with diagram using NLTK corpus – treebank.

Code:

```
# -*- coding: utf-8 -*-
"""
@author:Ankit Patel
"""
import nltk
nltk.download('treebank')
from nltk.corpus import treebank_chunk
treebank_chunk.tagged_sents()[0]
treebank_chunk.chunked_sents()[0]
treebank_chunk.chunked_sents()[0].draw()
```

Output:



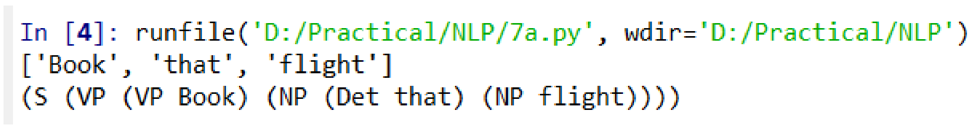
Practical 7A

Aim: Define grammar using nltk. Analyze a sentence using the same.

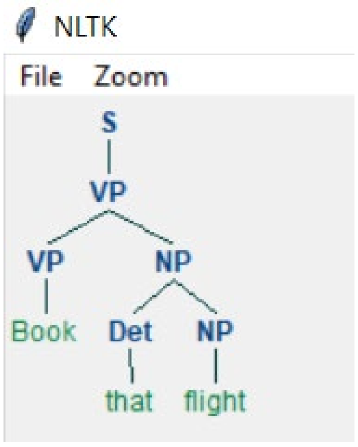
Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
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()
```

Output:



```
In [4]: runfile('D:/Practical/NLP/7a.py', wdir='D:/Practical/NLP')
['Book', 'that', 'flight']
(S (VP (VP Book) (NP (Det that) (NP flight))))
```



Practical 7B

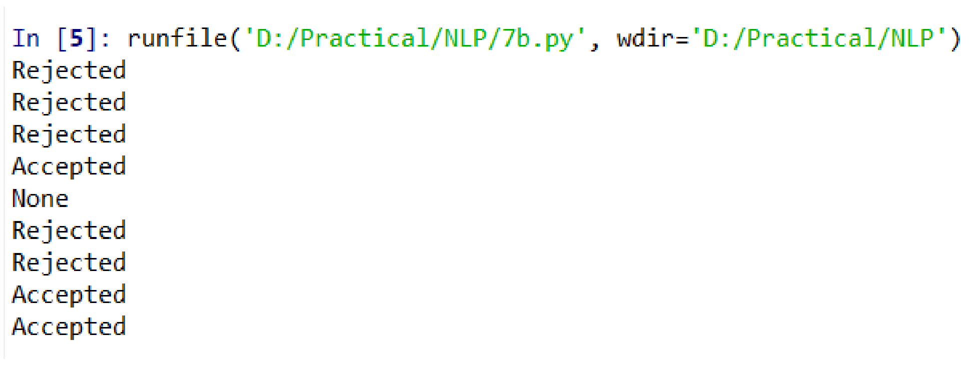
Aim: Accept the input string with Regular expression of Finite Automaton: 101+.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
def FA(s):
#if the length is less than 3 then it can't be accepted, Therefore end the process.
 if len(s)<3:
    return "Rejected"
#first three characters are fixed. Therefore, checking them using index
 if s[0]=='1':
    if s[1]=='0':
        if s[2]=='1':
 # After index 2 only "1" can appear. Therefore break the process if any other
character is detected
            for i in range(3,len(s)):
                if s[i]!='1':
                    return "Rejected"
                return "Accepted" # if all 4 nested if true
            return "Rejected" # else of 3rd if
        return "Rejected" # else of 2nd if
    return "Rejected" # else of 1st if
inputs=['1','10101','101','10111','01010','100','','10111101','1011111']
for i in inputs:
```

print(FA(i))

Output:



```
In [5]: runfile('D:/Practical/NLP/7b.py', wdir='D:/Practical/NLP')
Rejected
Rejected
Rejected
Accepted
None
Rejected
Rejected
Accepted
Accepted
```

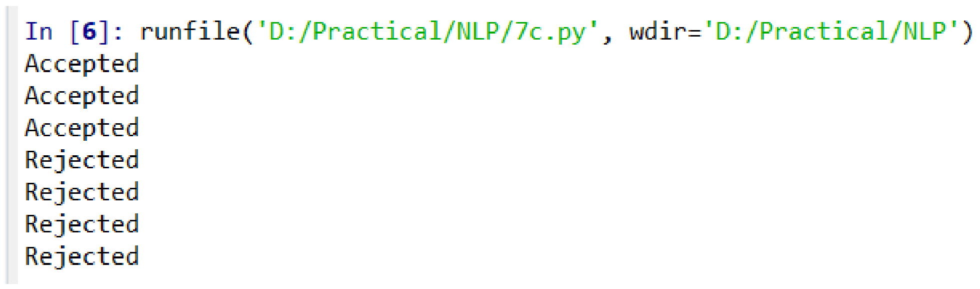
Practical 7C

Aim: Accept the input string with Regular expression of FA: (a+b)*bba.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
def FA(s):
 size=0
#scan complete string and make sure that it contains only 'a' & 'b'
 for i in s:
    if i=='a' or i=='b':
        size+=1
    else:
        return "Rejected"
#After checking that it contains only 'a' & 'b'
#check it's length it should be 3 atleast
 if size>=3:
#check the last 3 elements
    if s[size-3]=='b':
        if s[size-2]=='b':
            if s[size-1]=='a':
                return "Accepted" # if all 4 if true
            return "Rejected" # else of 4th if
        return "Rejected" # else of 3rd if
    return "Rejected" # else of 2nd if
 return "Rejected" # else of 1st if
inputs=['bba', 'ababbba', 'abba','abb', 'baba','bbb','']
for i in inputs:
 print(FA(i))
```

Output:



```
In [6]: runfile('D:/Practical/NLP/7c.py', wdir='D:/Practical/NLP')
Accepted
Accepted
Accepted
Rejected
Rejected
Rejected
Rejected
```

Practical 7D

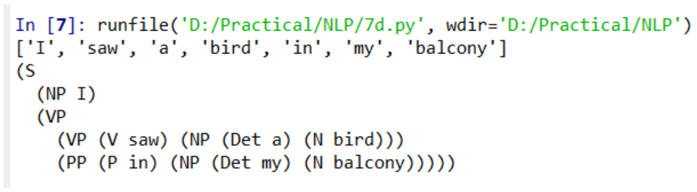
Aim: Implementation of Deductive Chart Parsing using context free grammar and a given sentence.

Code:

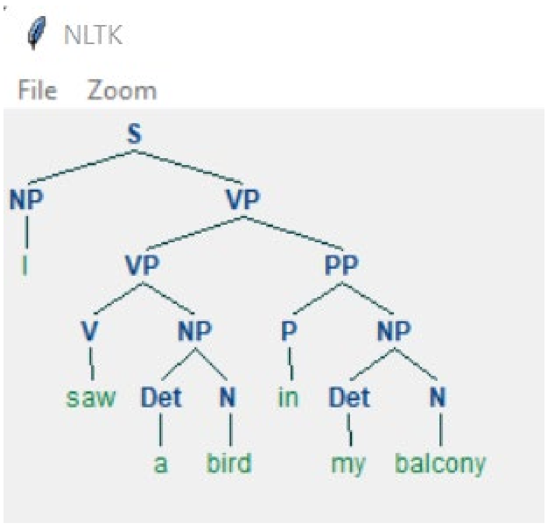
```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
from nltk import tokenize
grammar1 = nltk.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | 'I'
VP -> V NP | VP PP
Det -> 'a' | 'my'
N -> 'bird' | 'balcony'
V -> 'saw'
P -> 'in'
""")
sentence = "I saw a bird in my balcony"
for index in range(len(sentence)):
 all_tokens = tokenize.word_tokenize(sentence)
print(all_tokens)
# all_tokens = ['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
parser = nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
```

```
print(tree)

tree.draw()
```

Output:



```
In [7]: runfile('D:/Practical/NLP/7d.py', wdir='D:/Practical/NLP')
['I', 'saw', 'a', 'bird', 'in', 'my', 'balcony']
(S
  (NP I)
  (VP
    (VP (V saw) (NP (Det a) (N bird)))
    (PP (P in) (NP (Det my) (N balcony)))))
```



Practical 8

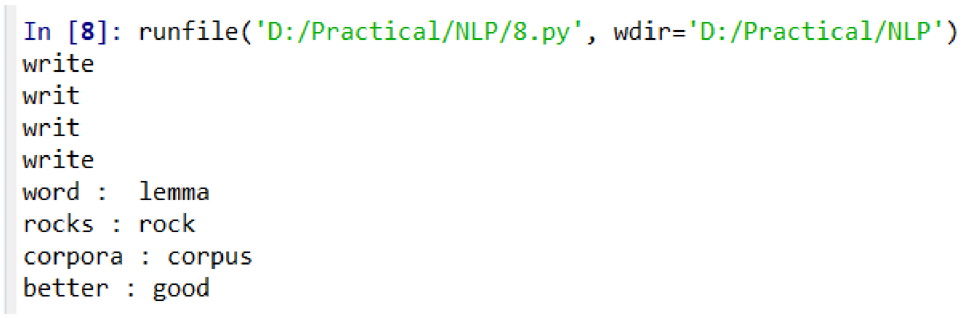
Aim: Study PorterStemmer, LancasterStemmer, RegexpStemmer, SnowballStemmer Study WordNetLemmatizer

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
# PorterStemmer
import nltk
from nltk.stem import PorterStemmer
word_stemmer = PorterStemmer()
print(word_stemmer.stem('writing'))
#LancasterStemmer
import nltk
from nltk.stem import LancasterStemmer
Lanc_stemmer = LancasterStemmer()
print(Lanc_stemmer.stem('writing'))
#RegexpStemmer
import nltk
from nltk.stem import RegexpStemmer
Reg_stemmer = RegexpStemmer('ing$|s$|e$|able$', min=4)
print(Reg_stemmer.stem('writing'))
#SnowballStemmer
import nltk
from nltk.stem import SnowballStemmer
english_stemmer = SnowballStemmer('english')
```

```
print(english_stemmer.stem ('writing'))
#WordNetLemmatizer
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print("word :\tlemma")
print("rocks :", lemmatizer.lemmatize("rocks"))
print("corpora :", lemmatizer.lemmatize("corpora"))
# a denotes adjective in "pos"
print("better :", lemmatizer.lemmatize("better", pos ="a"))
```

Output:



```
In [8]: runfile('D:/Practical/NLP/8.py', wdir='D:/Practical/NLP')
write
writ
writ
write
word :  lemma
rocks : rock
corpora : corpus
better : good
```

Practical 9

Aim: Implement Naive Bayes classifier.

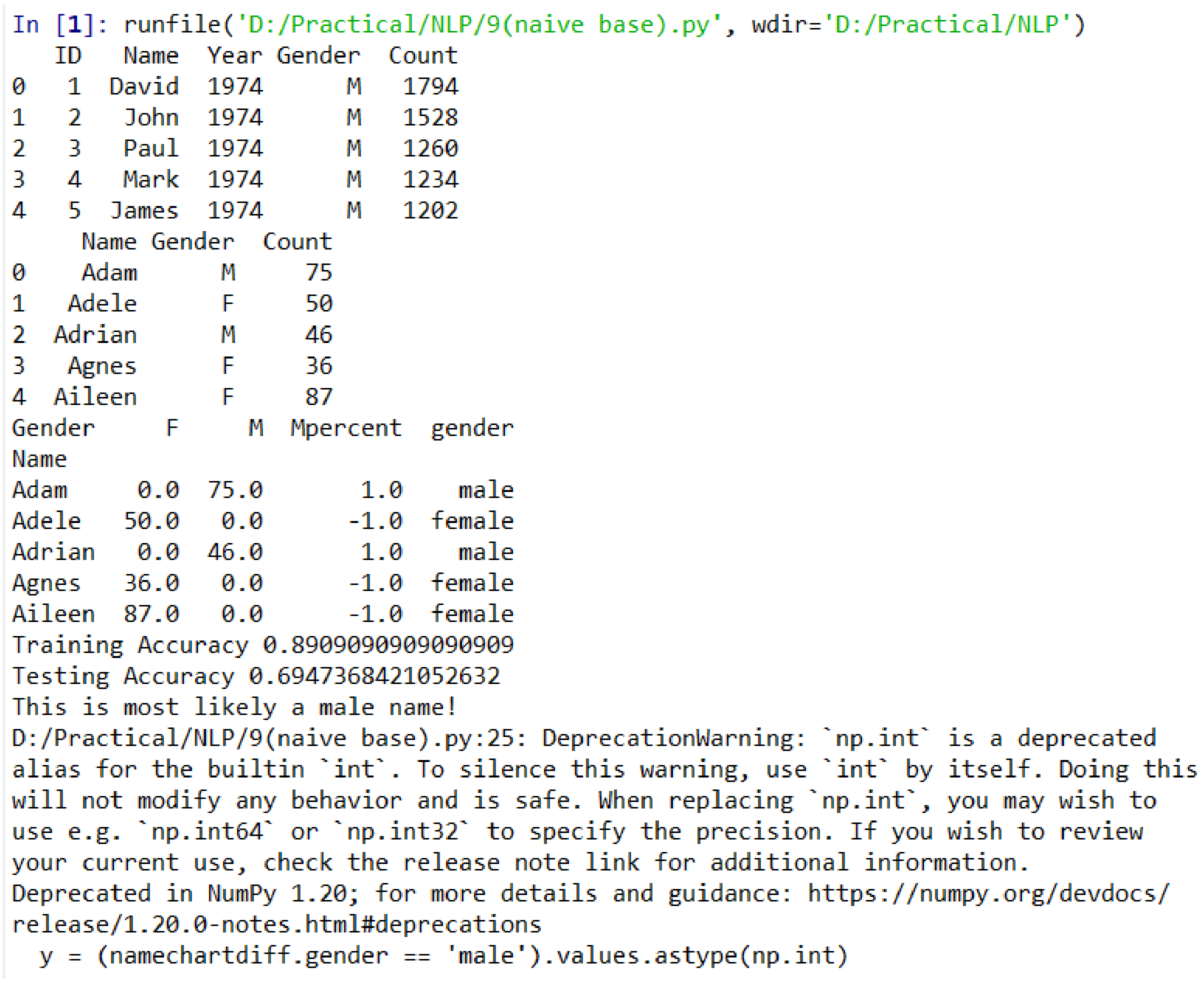
Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import pandas as pd
import numpy as np
from sklearn.naive_bayes import MultinomialNB
import matplotlib as mlp
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
mlp.rcParams.update({'font.family': "Open Sans", 'font.size' : 16})
names = pd.read_csv("C:/Users/91704/Downloads/Names.csv", dtype =
{'Count': np.int32})
names = names.fillna(0)
print(names.head())
namechart = names.groupby(['Name', 'Gender'], as_index =
False)['Count'].sum()
print(namechart.head(5))
namechartdiff = namechart.reset_index().pivot('Name', 'Gender', 'Count')
namechartdiff = namechartdiff.fillna(0)
namechartdiff["Mpercent"] = ((namechartdiff["M"] -
namechartdiff["F"])/(namechartdiff["M"] + namechartdiff["F"]))
namechartdiff['gender'] = np.where(namechartdiff['Mpercent'] > 0.001, 'male',
'female')
print(namechartdiff.head())
```

```
char_vectorizer = CountVectorizer(analyzer='char', ngram_range=(2, 2))
X = char_vectorizer.fit_transform(namechartdiff.index)
X = X.tocsc()
y = (namechartdiff.gender == 'male').values.astype(np.int)
itrain, itest = train_test_split(range(namechartdiff.shape[0]), train_size=0.7)
mask=np.ones(namechartdiff.shape[0], dtype='int')
mask[itrain]=1
mask[itest]=0
mask = (mask==1)
Xtrainthis=X[mask]
Ytrainthis=y[mask]
Xtestthis=X[~mask]
Ytestthis=y[~mask]
clf = MultinomialNB(alpha = 1)
clf.fit(Xtrainthis, Ytrainthis)
training_accuracy = clf.score(Xtrainthis,Ytrainthis)
test_accuracy = clf.score(Xtestthis,Ytestthis)
print("Training Accuracy",training_accuracy)
print("Testing Accuracy",test_accuracy)
def lookup(x):
    str(x)
    new = char_vectorizer.transform([x])
    y_pred = clf.predict(new)
    if (y_pred == 1):
        print("This is most likely a male name!")
    else:
        print("This is most likely a female name!")
```

```
lookup("David")
```

Output:



```
In [1]: runfile('D:/Practical/NLP/9(naive base).py', wdir='D:/Practical/NLP')
   ID   Name  Year Gender  Count
0   1  David  1974      M   1794
1   2   John  1974      M   1528
2   3   Paul  1974      M   1260
3   4   Mark  1974      M   1234
4   5  James  1974      M   1202
     Name Gender  Count
0    Adam      M     75
1   Adele      F     50
2  Adrian      M     46
3   Agnes      F     36
4  Aileen      F     87
Gender     F     M  Mpercent  gender
Name
Adam     0.0  75.0       1.0    male
Adele   50.0   0.0      -1.0  female
Adrian   0.0  46.0       1.0    male
Agnes   36.0   0.0      -1.0  female
Aileen  87.0   0.0      -1.0  female
Training Accuracy 0.8909090909090909
Testing Accuracy 0.6947368421052632
This is most likely a male name!
D:/Practical/NLP/9(naive base).py:25: DeprecationWarning: `np.int` is a deprecated
alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this
will not modify any behavior and is safe. When replacing `np.int`, you may wish to
use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review
your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
release/1.20.0-notes.html#deprecations
  y = (namechartdiff.gender == 'male').values.astype(np.int)
```

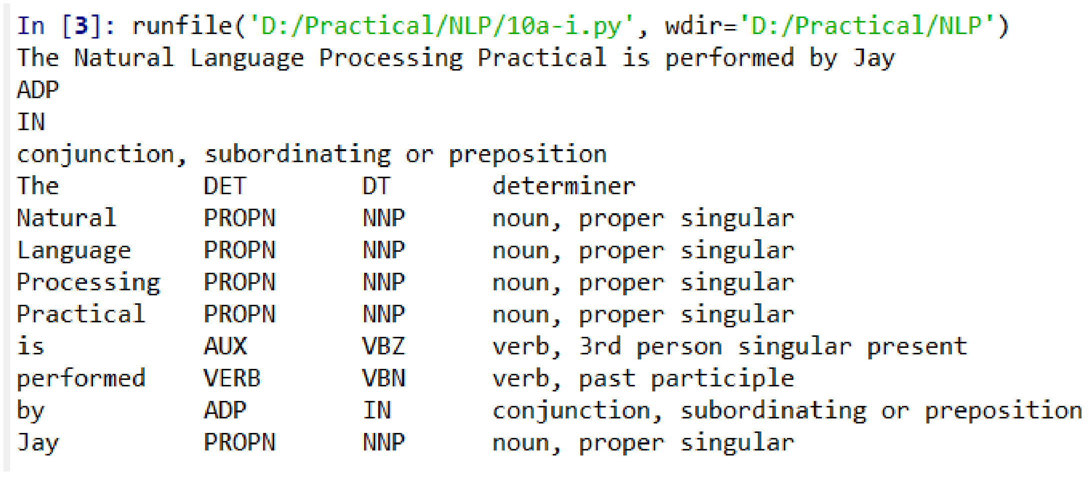
Practical 10A-i

Aim: Speech tagging using spacy.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import spacy
sp = spacy.load('en_core_web_sm')
sen = sp(u"The Natural Language Processing Practical is performed by Ankit")
print(sen.text)
print(sen[7].pos_)
print(sen[7].tag_)
print(spacy.explain(sen[7].tag_))
for word in sen:
    print(f'{word.text:{12}} {word.pos_:{10}} {word.tag_:{8}}
{spacy.explain(word.tag_)}')
```

Output:



```
In [3]: runfile('D:/Practical/NLP/10a-i.py', wdir='D:/Practical/NLP')
The Natural Language Processing Practical is performed by Jay
ADP
IN
conjunction, subordinating or preposition
The          DET        DT       determiner
Natural      PROPN      NNP      noun, proper singular
Language     PROPN      NNP      noun, proper singular
Processing   PROPN      NNP      noun, proper singular
Practical    PROPN      NNP      noun, proper singular
is           AUX        VBZ      verb, 3rd person singular present
performed    VERB       VBN      verb, past participle
by           ADP        IN       conjunction, subordinating or preposition
Jay          PROPN      NNP      noun, proper singular
```

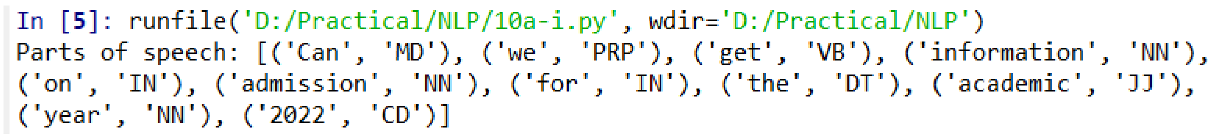
Practical 10A-ii

Aim: Speech tagging using nktl.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
tokens=nltk.word_tokenize("Can we get information on admission for the academic year 2022")
print("Parts of speech:",nltk.pos_tag(tokens))
```

Output:



```
In [5]: runfile('D:/Practical/NLP/10a-i.py', wdir='D:/Practical/NLP')
Parts of speech: [('Can', 'MD'), ('we', 'PRP'), ('get', 'VB'), ('information', 'NN'),
('on', 'IN'), ('admission', 'NN'), ('for', 'IN'), ('the', 'DT'), ('academic', 'JJ'),
('year', 'NN'), ('2022', 'CD')]
```

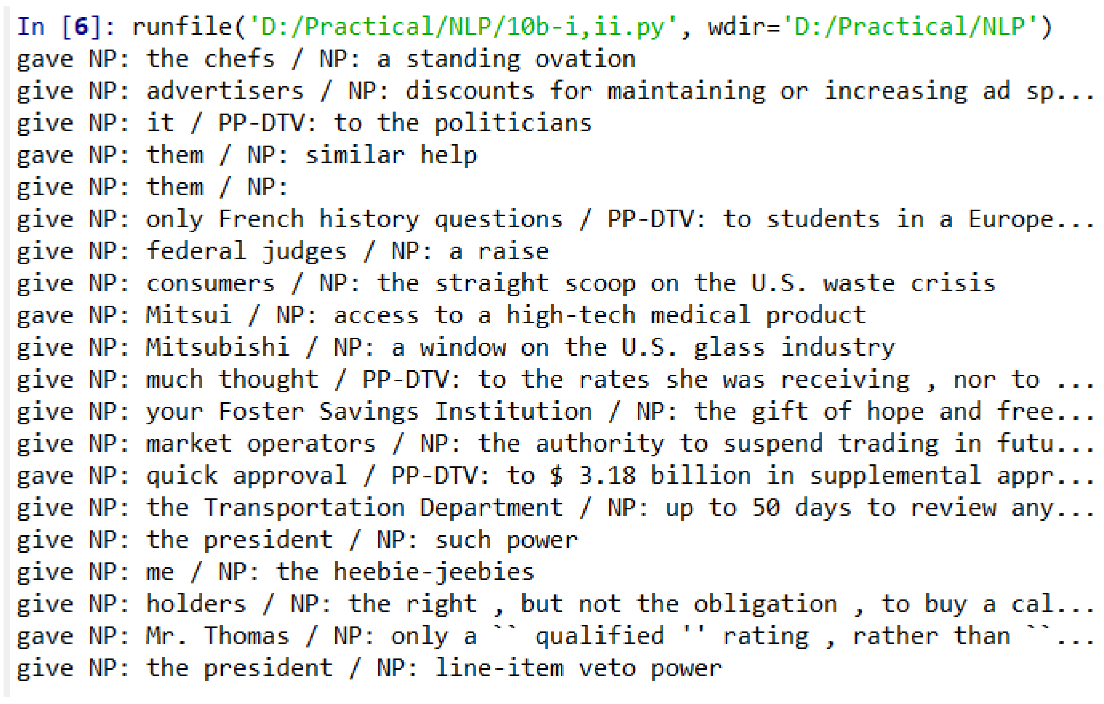
Practical 10B-i

Aim: Statistical parsing: Usage of Give and Gave in the Penn Treebank sample.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
def give(t):
   return t.label() == 'VP' and len(t) > 2 and t[1].label() == 'NP'\
   and (t[2].label() == 'PP-DTV' or t[2].label() == 'NP')\
   and ('give' in t[0].leaves() or 'gave' in t[0].leaves())
def sent(t):
   return ' '.join(token for token in t.leaves() if token[0] not in '*-0')
def print_node(t, width):
   output = "%s %s: %s / %s: %s" %\
   (sent(t[0]), t[1].label() , sent(t[1]), t[2].label() , sent(t[2]))
   if len(output) > width:
      output = output[:width] + "..."
   print(output)
for tree in nltk.corpus.treebank.parsed_sents():
   for t in tree.subtrees(give):
      print_node(t, 72)
```

Output:



```
In [6]: runfile('D:/Practical/NLP/10b-i,ii.py', wdir='D:/Practical/NLP')
gave NP: the chefs / NP: a standing ovation
give NP: advertisers / NP: discounts for maintaining or increasing ad sp...
give NP: it / PP-DTV: to the politicians
gave NP: them / NP: similar help
give NP: them / NP:
give NP: only French history questions / PP-DTV: to students in a Europe...
give NP: federal judges / NP: a raise
give NP: consumers / NP: the straight scoop on the U.S. waste crisis
gave NP: Mitsui / NP: access to a high-tech medical product
give NP: Mitsubishi / NP: a window on the U.S. glass industry
give NP: much thought / PP-DTV: to the rates she was receiving , nor to ...
give NP: your Foster Savings Institution / NP: the gift of hope and free...
give NP: market operators / NP: the authority to suspend trading in futu...
gave NP: quick approval / PP-DTV: to $ 3.18 billion in supplemental appr...
give NP: the Transportation Department / NP: up to 50 days to review any...
give NP: the president / NP: such power
give NP: me / NP: the heebie-jeebies
give NP: holders / NP: the right , but not the obligation , to buy a cal...
gave NP: Mr. Thomas / NP: only a `` qualified '' rating , rather than ``...
give NP: the president / NP: line-item veto power
```

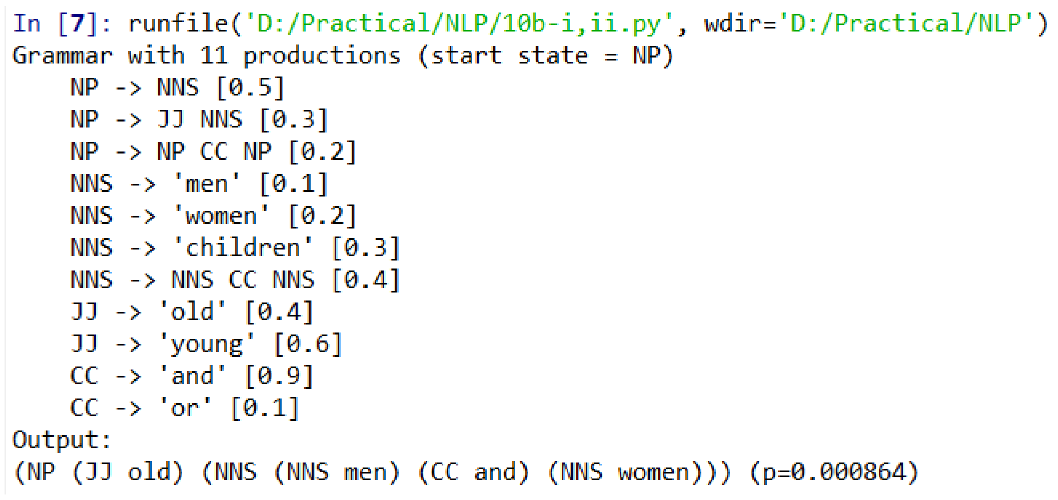
Practical 10B-ii

Aim: Statistical parsing: probabilistic parser.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import nltk
from nltk import PCFG
grammar = PCFG.fromstring('''
NP -> NNS [0.5] | JJ NNS [0.3] | NP CC NP [0.2]
NNS -> "men" [0.1] | "women" [0.2] | "children" [0.3] | NNS CC NNS [0.4]
JJ -> "old" [0.4] | "young" [0.6]
CC -> "and" [0.9] | "or" [0.1]
''')
print(grammar)
viterbi_parser = nltk.ViterbiParser(grammar)
token = "old men and women".split()
obj = viterbi_parser.parse(token)
print("Output: ")
for x in obj:
    print(x)
```

Output:


```
In [7]: runfile('D:/Practical/NLP/10b-i,ii.py', wdir='D:/Practical/NLP')
Grammar with 11 productions (start state = NP)
    NP -> NNS [0.5]
    NP -> JJ NNS [0.3]
    NP -> NP CC NP [0.2]
    NNS -> 'men' [0.1]
    NNS -> 'women' [0.2]
    NNS -> 'children' [0.3]
    NNS -> NNS CC NNS [0.4]
    JJ -> 'old' [0.4]
    JJ -> 'young' [0.6]
    CC -> 'and' [0.9]
    CC -> 'or' [0.1]
Output:
(NP (JJ old) (NNS (NNS men) (CC and) (NNS women))) (p=0.000864)
```

Practical 10C

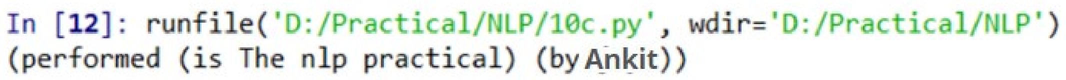
Aim: Malt parsing: Parse a sentence and draw a tree using malt parsing.

Concept:

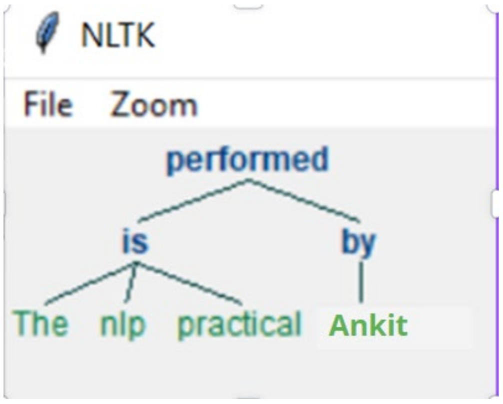
Code:

```
# -*- coding: utf-8 -*-
"""
@author:Ankit Patel
"""
from nltk.parse import malt
mp = malt.MaltParser('C:/Users/91704/Anaconda3/maltparser-1.9.2',
'C:/Users/91704/Anaconda3/engmalt.linear-1.7.mco')#file
t = mp.parse_one('The nlp practical is performed by Ankit.'.split()).tree()
print(t)
t.draw()
```

Output:



```
In [12]: runfile('D:/Practical/NLP/10c.py', wdir='D:/Practical/NLP')
(performed (is The nlp practical) (by Ankit))
```



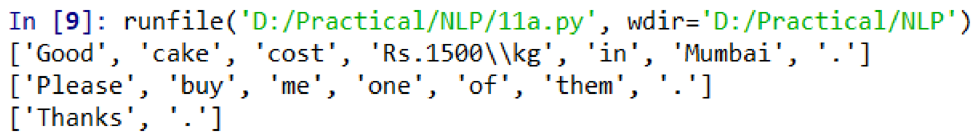
Practical 11A

Aim: Multiword Expressions in NLP.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
# Multiword Expressions in NLP
from nltk.tokenize import MWETokenizer
from nltk import sent_tokenize, word_tokenize
s = '''Good cake cost Rs.1500\kg in Mumbai. Please buy me one of them.\n\nThanks.'''
mwe = MWETokenizer([('New', 'York'), ('Hong', 'Kong')], separator='_')
for sent in sent_tokenize(s):
 print(mwe.tokenize(word_tokenize(sent)))
```

Output:



```
In [9]: runfile('D:/Practical/NLP/11a.py', wdir='D:/Practical/NLP')
['Good', 'cake', 'cost', 'Rs.1500\\kg', 'in', 'Mumbai', '.']
['Please', 'buy', 'me', 'one', 'of', 'them', '.']
['Thanks', '.']
```

Practical 11B

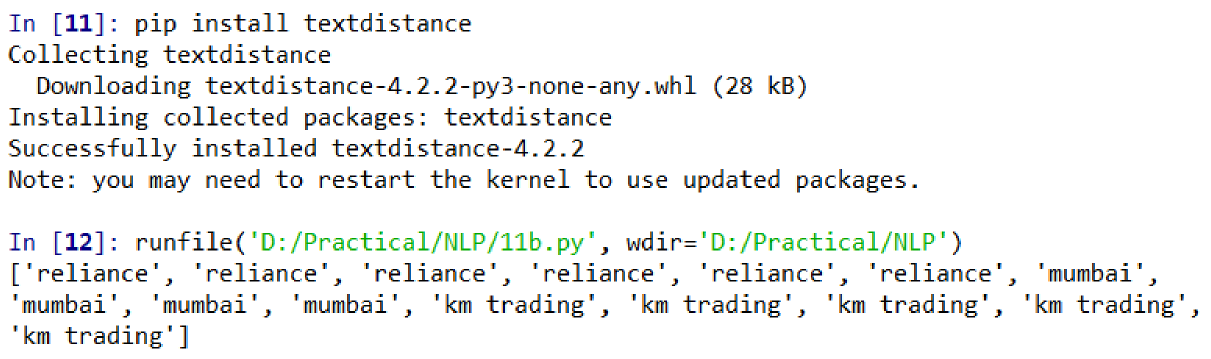
Aim: Normalized Web Distance and Word Similarity.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
import numpy as np
import re
import textdistance # pip install textdistance
# we will need scikit-learn>=0.21
import sklearn #pip install sklearn
from sklearn.cluster import AgglomerativeClustering
texts = ['Reliance supermarket', 'Reliance hypermarket', 'Reliance', 'Reliance',
'Reliance downtown', 'Relianc market','Mumbai', 'Mumbai Hyper', 'Mumbai
dxb', 'mumbai airport', 'k.m trading', 'KM Trading', 'KM trade', 'K.M. Trading',
'KM.Trading']
def normalize(text):
 """ Keep only lower-cased text and numbers"""
 return re.sub('[^a-z0-9]+', ' ', text.lower())
def group_texts(texts, threshold=0.4):
 """ Replace each text with the representative of its cluster"""
 normalized_texts = np.array([normalize(text) for text in texts])
 distances = 1 - np.array([
 [textdistance.jaro_winkler(one, another) for one in normalized_texts]
 for another in normalized_texts
 ])
```

```
clustering = AgglomerativeClustering(
distance_threshold=threshold, # this parameter needs to be tuned carefully
affinity="precomputed", linkage="complete", n_clusters=None
).fit(distances)
centers = dict()
for cluster_id in set(clustering.labels_):
    index = clustering.labels_ == cluster_id
    centrality = distances[:, index][index].sum(axis=1)
    centers[cluster_id] = normalized_texts[index][centrality.argmin()]
return [centers[i] for i in clustering.labels_]
print(group_texts(texts))
```

Output:



```
In [11]: pip install textdistance
Collecting textdistance
  Downloading textdistance-4.2.2-py3-none-any.whl (28 kB)
Installing collected packages: textdistance
Successfully installed textdistance-4.2.2
Note: you may need to restart the kernel to use updated packages.

In [12]: runfile('D:/Practical/NLP/11b.py', wdir='D:/Practical/NLP')
['reliance', 'reliance', 'reliance', 'reliance', 'reliance', 'reliance', 'mumbai',
'mumbai', 'mumbai', 'mumbai', 'km trading', 'km trading', 'km trading', 'km trading',
'km trading']
```

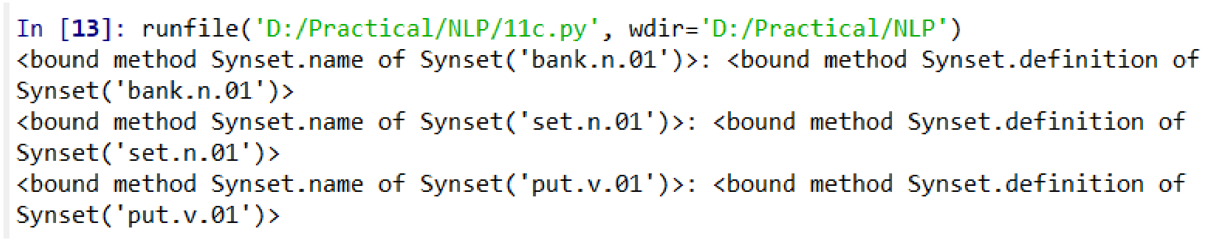
Practical 11C

Aim: Word Sense Disambiguation.

Code:

```
# -*- coding: utf-8 -*-
"""
@author: Ankit Patel
"""
#Word Sense Disambiguation
from nltk.corpus import wordnet as wn
def get_first_sense(word, pos=None):
 if pos:
     synsets = wn.synsets(word,pos)
 else:
     synsets = wn.synsets(word)
 return synsets[0]
best_synset = get_first_sense('bank')
print ('%s: %s' % (best_synset.name, best_synset.definition))
best_synset = get_first_sense('set','n')
print ('%s: %s' % (best_synset.name, best_synset.definition))
best_synset = get_first_sense('set','v')
print ('%s: %s' % (best_synset.name, best_synset.definition))
```

Output:



```
In [13]: runfile('D:/Practical/NLP/11c.py', wdir='D:/Practical/NLP')
<bound method Synset.name of Synset('bank.n.01')>: <bound method Synset.definition of
Synset('bank.n.01')>
<bound method Synset.name of Synset('set.n.01')>: <bound method Synset.definition of
Synset('set.n.01')>
<bound method Synset.name of Synset('put.v.01')>: <bound method Synset.definition of
Synset('put.v.01')>
```