Hindi Vidya Prachar Samiti's
RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE & COMMERCE
(EMPOWERED AUTONOMUS)

Natural Language Processing

Name: Ranu Singh

Roll No.: 10444

Class: MSc Data Science and Artificial Intelligence Part-II

Ramniranjan Jhunjhunwala College of Arts, Science and Commerce

Department of Data Science and Artificial Intelligence

CERTIFICATE

This is to certify Mast. Ranu Singh of MSc. Data Science and Artificial Intelligence, roll no. 10444 has successfully completed the practical of NATURAL LANGUAGE PROCESSING during the Academic Year 2024-2025.

Date:

Prof. Dr. Neha Ansari
(Prof-in-charge)

External Examiner

INDEX

Prac. No	Practical Name	Date	Signature
1.	**Web Scrapping: Scrape data from a webpage**	**11/07/2024**	
2.	**Implementation of Sentiment Analysis**	**18/07/2024**	
3.	**TEXT PREPROCESSING** **a] Performing word tokenization using different python library such as; Keras and tensorflow, spacy, gensim etc.** **b] Text Tokenization using py split function** **c] Performing sentence and word tokenization using nltk's tokenize api**	**25/07/2024**	
4.	**Demonstrate Praser in NLP**	**01/08/2024**	
5.	**Feature Extraction technique: Word2Vec**	**08/08/2024**	
6.	**Demonstrate word representation technique Bag-Of-Word (BOW)**	**08/08/2024**	
7.	**Demonstrate the word representation N-Gram technique**	**08/08/2024**	
8.	**Performing Tf - idf**	**22/08/2024**	
9.	**Study Porter Stemmer, Lancaster Stemmer, Regexp Stemmer, WordNet Stemmer**	**29/08/2024**	
10.	**Tagging in NLP**	**29/09/2024**	

PRACTICAL NO.: 09

AIM: Study Porter Stemmer, Lancaster Stemmer, Regexp Stemmer, WordNet Stemmer **Date: 29/08/2024**

```
from nltk.stem import PorterStemmer
wordstemmer = PorterStemmer()
print(wordstemmer.stem("writing"))
```

```
write
```

```
from nltk.stem import LancasterStemmer
wordstemmer2 = LancasterStemmer()
print(wordstemmer2.stem("writing"))
```

```
writ
```

```
from nltk.stem import RegexpStemmer
wordstemmer3 = RegexpStemmer("ing$|s$|ed$|able$")
print(wordstemmer3.stem("writing"))
```

```
writ
```

```
from nltk.stem import SnowballStemmer
wordstemmer4 = SnowballStemmer("english")
print(wordstemmer4.stem("writing"))
```

```
write
```

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
wordstemmer5 = WordNetLemmatizer()
```

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

```
print("rocks :", wordstemmer5.lemmatize("rocks"))
print("corpora :", wordstemmer5.lemmatize("corpora"))
```

```
rocks : rock
corpora : corpus
```

```
print("better :", wordstemmer5.lemmatize("better", pos ="a"))
print("best :", wordstemmer5.lemmatize("best", pos ="v"))
```

```
better : good
best : best
```

print("better :", wordstemmer5.lemmatize("better"))

```
better : better
```

PRACTICAL NO.: 03

AIM: Implementation of Text Processing

(a) Text Tokenization using py split function Date: 25/07/2024

Tokenization using py split function

!pip install nltk

texts = """A gravitational singularity, spacetime singularity or simply singularity is a condition in which gravity is predicted to be so intense that spacetime itself would break down catastrophically. As such, a singularity is by definition no longer part of the regular spacetime and cannot be determined by "where" or "when". Gravitational singularities exist at a junction between general relativity and quantum mechanics; therefore, the properties of the singularity cannot be described without an established theory of quantum gravity. Trying to find a complete and precise definition of singularities in the theory of general relativity, the current best theory of gravity, remains a difficult problem. A singularity in general relativity can be defined by the scalar invariant curvature becoming infinite[3] or, better, by a geodesic being incomplete."""

print(texts)

```
data = texts.split(".")
for i in data:
  print("Sentence segmented:",i)
print(data)
```

```
Sentence segmented: A gravitational singularity, spacetime singularity or simply singularity is a condition in which gravity is predicted to be so
Sentence segmented:  As such, a singularity is by definition no longer part of the regular spacetime and cannot be determined by "where" or "when"
Sentence segmented:  Gravitational singularities exist at a junction between general relativity and quantum mechanics; therefore, the properties of
Sentence segmented:  Trying to find a complete and precise definition of singularities in the theory of general relativity, the current best theory
Sentence segmented:  A singularity in general relativity can be defined by the scalar invariant curvature becoming infinite[3] or, better, by a geo
Sentence segmented:
['A gravitational singularity, spacetime singularity or simply singularity is a condition in which gravity is predicted to be so intense that space
```

AIM [b]: Performing sentence and word tokenization using nltk's tokenize api

```
import nltk
from nltk import tokenize
nltk.download('punkt')
nltk.download('words')

para = "Hello! Myself Abc is xyz. Today we study about NLTK"
sents = tokenize.sent_tokenize(para)
print("sentence
tokenization\n==========================================\n",sents)
```

```
sentence tokenization
==========================================
 ['Hello!', 'Myself Abc is xyz.', 'Today we study about NLTK']
```

```
print("word
tokenization\n==========================================\n")
for i in range(len(sents)):
  words = tokenize.word_tokenize(sents[i])
  print(words)
```

```
word tokenization
==================================================

['Hello', '!']
['Myself', 'Abc', 'is', 'xyz', '.']
['Today', 'we', 'study', 'about', 'NLTK']
```

AIM [c]: Performing word tokenization using different python library

```
import nltk
from nltk import tokenize
from nltk.tokenize import word_tokenize
str = "I like to perform NLPexperiments in Python"
nltk.download('punkt')
print(word_tokenize(str))
```

```
['I', 'like', 'to', 'perform', 'NLPexperiments', 'in', 'Python']
```

using keras and tensorflow libraries

```
!pip install tensorflow
!pip install keras
import keras
import tensorflow as tf
#from keras.processing text import text_to_word_sequence
str  = 'I like to perform NLP experiments in python'
tokens = tf.keras.preprocessing.text.text_to_word_sequence(str)
print(tokens)
```

```
['i', 'like', 'to', 'perform', 'nlp', 'experiments', 'in', 'python']
```

using regular experssion tokenizer

```
import nltk
from nltk.tokenize import RegexpTokenizer
tk = RegexpTokenizer('\s+',gaps=True)
str = 'I like to code in python'
tokens = tk.tokenize(str)
print(tokens)
```

```
['I', 'like', 'to', 'code', 'in', 'python']
```

using gensim library

```
!pip install gensim
from gensim.utils import tokenize
str = 'I like to code in python'
#tokens = tokenize(str)
```

```
#print(tokens)
list(tokenize(str))
```

```
['I', 'like', 'to', 'code', 'in', 'python']
```

```
tokens=tokenize(str)
tokens
```

```
<generator object simple_tokenize at 0x7c057cfe8f90>
```

```
list(tokens)
```

```
['I', 'like', 'to', 'code', 'in', 'python']
```

```
import spacy
nlp = spacy.blank('en')
str = 'I like to code in python'
doc = nlp(str)
print(doc)
```

```
I like to code in python
```

```
words = [words.text for words in doc]
print(words)
```

```
['I', 'like', 'to', 'code', 'in', 'python']
```

PRACTICAL NO.: 08

AIM: Perform Tf-idf **Date: 22/08/2024**

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidfmodel = TfidfVectorizer()
corpus = ['data science is one of the most important feilds of computer science',
        'this is one of the best data science platform',
        'data scientists analyze the data']
tf_idf_vector = tfidfmodel.fit_transform(corpus)
tf_idf_vector.shape
```

```
(3, 15)
```

```
tf_idf_vector = tf_idf_vector.toarray()
tf_idf_vector
```

```
array([[0.        , 0.        , 0.30887673, 0.18242757, 0.30887673,
        0.30887673, 0.23490871, 0.30887673, 0.46981743, 0.23490871,
        0.        , 0.46981743, 0.        , 0.18242757, 0.        ],
       [0.        , 0.40786601, 0.        , 0.24089223, 0.        ,
        0.        , 0.31019261, 0.        , 0.31019261, 0.31019261,
        0.40786601, 0.31019261, 0.        , 0.24089223, 0.40786601],
       [0.51680194, 0.        , 0.        , 0.61046311, 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.51680194, 0.30523155, 0.        ]])
```

```
word_sets= tfidfmodel.get_feature_names_out()
word_sets
```

```
array(['analyze', 'best', 'computer', 'data', 'feilds', 'important', 'is',
       'most', 'of', 'one', 'platform', 'science', 'scientists', 'the',
       'this'], dtype=object)
```

```
import pandas as pd
df = pd.DataFrame(tf_idf_vector,columns=word_sets)
df
```

	analyze	best	computer	data	feilds	important	is	most	of	one	platform	science	scientists	the	this
0	0.000000	0.000000	0.308877	0.182428	0.308877	0.308877	0.234909	0.308877	0.469817	0.234909	0.000000	0.469817	0.000000	0.182428	0.000000
1	0.000000	0.407866	0.000000	0.240892	0.000000	0.000000	0.310193	0.000000	0.310193	0.310193	0.407866	0.310193	0.000000	0.240892	0.407866
2	0.516802	0.000000	0.000000	0.610463	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.516802	0.305232	0.000000

PRACTICAL NO.: 02

AIM: Implementation Sentiment analysis **Date: 18/07/2024**

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB,BernoulliNB
newsgroups = fetch_20newsgroups(subset='all')
newsgroups.target_names
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

```
vectorizer1 = CountVectorizer(binary=True)
vectorizer2 = CountVectorizer(binary=False)
vectorizer3 = CountVectorizer(binary=True,stop_words='english')
vectorizer4 = CountVectorizer(binary=False,stop_words='english')
X1 = vectorizer1.fit_transform(newsgroups.data)
X2 = vectorizer2.fit_transform(newsgroups.data)
X3 = vectorizer3.fit_transform(newsgroups.data)
X4 = vectorizer4.fit_transform(newsgroups.data)
y = newsgroups.target

from sklearn.model_selection import train_test_split
xtrain1,xtest1,ytrain1,ytest1 = train_test_split(X1,y,test_size=0.25,random_state=0)
xtrain2,xtest2,ytrain2,ytest2 = train_test_split(X2,y,test_size=0.25,random_state=0)
xtrain3,xtest3,ytrain3,ytest3 = train_test_split(X3,y,test_size=0.25,random_state=0)
xtrain4,xtest4,ytrain4,ytest4 = train_test_split(X4,y,test_size=0.25,random_state=0)

bnb=BernoulliNB()
mnb=MultinomialNB()
bnb1=BernoulliNB()
mnb1=MultinomialNB()
bnb.fit(xtrain1,ytrain1)
mnb.fit(xtrain2,ytrain2)
```

```
▾ MultinomialNB
MultinomialNB()
```

```
bnb1.fit(xtrain3,ytrain3)
```

```
mnb1.fit(xtrain4,ytrain4)
```

```
MultinomialNB()
```

```
y_pred1  =  bnb.predict(xtest1)
y_pred2  =  mnb.predict(xtest2)
y_pred3  =  bnb1.predict(xtest3)
y_pred4 = mnb1.predict(xtest4)
from sklearn.metrics import accuracy_score
accuracy_score(ytest1,y_pred1)
```

```
0.6899405772495756
```

```
accuracy_score(ytest2,y_pred2)
```

```
0.8548387096774194
```

```
accuracy_score(ytest3,y_pred3)
```

```
0.719439728353141
```

```
accuracy_score(ytest4,y_pred4)
```

```
0.8779711375212224
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model1 = make_pipeline(TfidfVectorizer(stop_words='english'), MultinomialNB())
train_data = fetch_20newsgroups(subset='train')
test_data = fetch_20newsgroups(subset='test')
model.fit(train_data.data, train_data.target)
```

```
Pipeline
TfidfVectorizer
MultinomialNB
```

```
model1.fit(train_data.data, train_data.target)
```

```
Pipeline
TfidfVectorizer
MultinomialNB
```

```
predictions_tf = model.predict(test_data.data)
```

```
predictions_tf1 = model1.predict(test_data.data)
accuracy_score(test_data.target, predictions_tf)
```

```
0.7738980350504514
```

```
accuracy_score(test_data.target,  predictions_tf1)
```

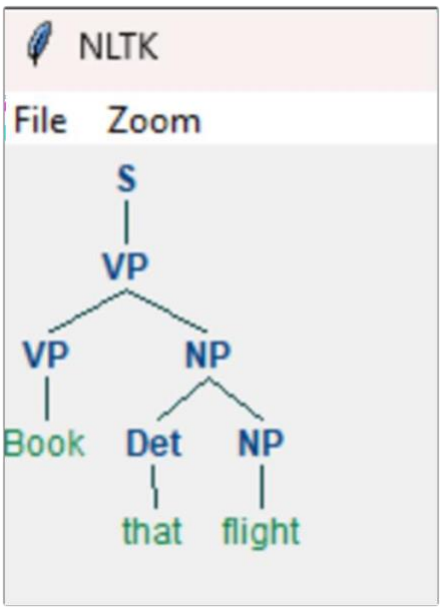
```
0.8169144981412639
```

PRACTICAL NO.: 04

AIM: Demonstrate Parser (CFG) in NLP **Date: 01/08/2024**

```
import nltk
nltk.download('punkt')
from nltk import tokenize
grammar1 = nltk.CFG.fromstring("""
    S -> VP
    VP -> VP NP
    NP -> Det NP
    Det -> 'that'
    NP -> 'singular noun'
    NP -> 'flight'
    VP -> 'Book'
""")
sentence="Book that flight"
for index in range(len(sentence)):
    all_tokens = tokenize.word_tokenize(sentence)
print(all_tokens)
parser=nltk.ChartParser(grammar1)
for tree in parser.parse(all_tokens):
    print(tree)
    tree.draw()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\User20\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
['Book', 'that', 'flight']
(S (VP (VP Book) (NP (Det that) (NP flight))))
```



PRACTICAL NO.: 07

AIM: Demonstrate the word representation Bag-Of-Words (BOW) technique
Date: 08/08/2024

```
from sklearn.feature_extraction.text import CountVectorizer
documents = ["I love programming in Python or Python programming ",
        "Python is a great programming language",
        "I love coding in Python"]
documents
```

```
['I love programming in Python or Python programming ',
 'Python is a great programming language',
 'I love coding in Python']
```

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)
print(X.toarray())
```

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

```
print(vectorizer.get_feature_names_out())
```

```
['coding' 'great' 'in' 'is' 'language' 'love' 'or' 'programming' 'python']
```

PRACTICAL NO.: 05

AIM: Perform Feature Extraction technique in NLP task

Date: 08/08/2024

a) Tf-idf

```
documents = ["I love programming in Python or Python programming ",
        "Python is a great programming language",
        "I love coding in Python"]

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer2 = TfidfVectorizer()
X2 = vectorizer2.fit_transform(documents)
print(X2.toarray())
```

```
[[0.         0.         0.31401745 0.         0.         0.31401745
  0.41289521 0.6280349  0.48772512]
 [0.         0.50461134 0.         0.50461134 0.50461134 0.
  0.         0.38376993 0.29803159]
 [0.63174505 0.         0.4804584  0.         0.         0.4804584
  0.         0.         0.37311881]]
```

b) WORD2VEC

```
from gensim.models import Word2Vec
documents = [["I", "love","programming" ,"in" ,"Python", "or", "Python" , "programming
"],
        ["Python", "is" ,"a" ,"great" ,"programming" ,"language"],
        ["I", "love", "coding", "in", "Python"]]
model =  Word2Vec(documents, vector_size=100, window=5, min_count=1, workers=4)
vector = model.wv['Python']
print(vector)
```

```
[-5.3622725e-04  2.3643136e-04  5.1033497e-03  9.0092728e-03
 -9.3029495e-03 -7.1168090e-03  6.4588725e-03  8.9729885e-03
 -5.0154282e-03 -3.7633716e-03  7.3805046e-03 -1.5334714e-03
 -4.5366134e-03  6.5540518e-03 -4.8601604e-03 -1.8160177e-03
  2.8765798e-03  9.9187379e-04 -8.2852151e-03 -9.4488179e-03
  7.3117660e-03  5.0702621e-03  6.7576934e-03  7.6286553e-04
  6.3508903e-03 -3.4053659e-03 -9.4640139e-04  5.7685734e-03
 -7.5216377e-03 -3.9361035e-03 -7.5115822e-03 -9.3004224e-04
  9.5381187e-03 -7.3191668e-03 -2.3337686e-03 -1.9377411e-03
  8.0774371e-03 -5.9308959e-03  4.5162440e-05 -4.7537340e-03
 -9.6035507e-03  5.0072931e-03 -8.7595852e-03 -4.3918253e-03
 -3.5099984e-05 -2.9618145e-04 -7.6612402e-03  9.6147433e-03
  4.9820580e-03  9.2331432e-03 -8.1579173e-03  4.4957981e-03
 -4.1370760e-03  8.2453608e-04  8.4986202e-03 -4.4621765e-03
  4.5175003e-03 -6.7869602e-03 -3.5484887e-03  9.3985079e-03
 -1.5776526e-03  3.2137157e-04 -4.1406299e-03 -7.6826881e-03
 -1.5080082e-03  2.4697948e-03 -8.8802696e-04  5.5336617e-03
 -2.7429771e-03  2.2600652e-03  5.4557943e-03  8.3459532e-03
 -1.4537406e-03 -9.2081428e-03  4.3705525e-03  5.7178497e-04
  7.4419081e-03 -8.1328274e-04 -2.6384138e-03 -8.7530091e-03
 -8.5655687e-04  2.8265631e-03  5.4014288e-03  7.0526563e-03
 -5.7031214e-03  1.8588197e-03  6.0888636e-03 -4.7980510e-03
 -3.1072604e-03  6.7976294e-03  1.6314756e-03  1.8991709e-04
  3.4736372e-03  2.1777749e-04  9.6188262e-03  5.0606038e-03
 -8.9173904e-03 -7.0415605e-03  9.0145587e-04  6.3925339e-03]
```

```
similar_words = model.wv.most_similar('Python',topn=2)
similar_words
```

```
[('is', 0.216171428561210 63), ('great', 0.09291706234216 69)]
```

PRACTICAL NO.: 07

AIM: Demonstrate the word representation N-gram technique
Date: 08/08/2024

```
count_vectorizer = CountVectorizer(ngram_range=(2,2))
documents = ["I love programming in Python or Python programming ",
             "Python is a great programming language",
             "I love coding in Python"]
xcount=count_vectorizer.fit_transform(documents)
print("Count Vectorizer with bigram")
print(xcount.toarray())
print(count_vectorizer.get_feature_names_out())
```

```
Count Vectorizer with bigram
[[0 0 1 0 0 1 1 1 0 0 1 1]
 [0 1 0 1 0 0 0 0 1 1 0 0]
 [1 0 1 0 1 0 0 0 0 0 0 0]]
['coding in' 'great programming' 'in python' 'is great' 'love coding'
 'love programming' 'or python' 'programming in' 'programming language'
 'python is' 'python or' 'python programming']
```

```
tfidf_vectorizer = TfidfVectorizer(ngram_range=(2,2))
xcount2=tfidf_vectorizer.fit_transform(documents)
print("TFIDF Vectorizer with bigram")
print(xcount2.toarray())
print(tfidf_vectorizer.get_feature_names_out())
```

```
TFIDF Vectorizer with bigram
[[0.         0.         0.32200242 0.         0.         0.42339448
  0.42339448 0.42339448 0.         0.         0.42339448 0.42339448]
 [0.         0.5        0.         0.5        0.         0.
  0.         0.         0.5        0.5        0.         0.        ]
 [0.62276601 0.         0.4736296  0.         0.62276601 0.
  0.         0.         0.         0.         0.         0.        ]]
['coding in' 'great programming' 'in python' 'is great' 'love coding'
 'love programming' 'or python' 'programming in' 'programming language'
 'python is' 'python or' 'python programming']
```

PRACTICAL NO.: 10

AIM: Tagging in NLP **Date: 29/08/2024**

a) Default Tagger

```
import nltk
nltk.download('treebank')
from nltk.tag import DefaultTagger
dt=DefaultTagger('NN')
from nltk.corpus import treebank
sents=treebank.tagged_sents()[1000:]
print(dt.evaluate(sents))
```

```
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Unzipping corpora/treebank.zip.
<ipython-input-2-3684e3892b6a>:9: DeprecationWarning:
  Function evaluate() has been deprecated.  Use accuracy(gold)
  instead.
  print(dt.evaluate(sents))
0.13198749536374715
```

```
#Tagging a list of sentence
import nltk
from nltk.tag import DefaultTagger
dt=DefaultTagger('NN')
print(dt.tag_sents([['Hi',','],['How','are','you','?']]))
```

```
[[('Hi', 'NN'), (',', 'NN')], [('How', 'NN'), ('are', 'NN'), ('you', 'NN'), ('?', 'NN')]]
```

b) Regular Expression Tagging

```
from nltk.corpus import brown
nltk.download('brown')
from nltk.tag import RegexpTagger
test_sent = brown.sents(categories='news')[0]
regexp_tagger = RegexpTagger(
[(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),   # cardinal
  (r'(The|the|A|a|An|an)$', 'AT'),   # articles
  (r'.*able$', 'JJ'),                # adjectives
  (r'.*ness$', 'NN'),                # nouns formed from adjectives
  (r'.*ly$', 'RB'),                  # adverbs
  (r'.*s$', 'NNS'),                  # plural nouns
  (r'.*ing$', 'VBG'),                # gerunds
  (r'.*ed$', 'VBD'),                 # past tense verbs
  (r'.*', 'NN')                     # nouns (default)
])
print(regexp_tagger)
```

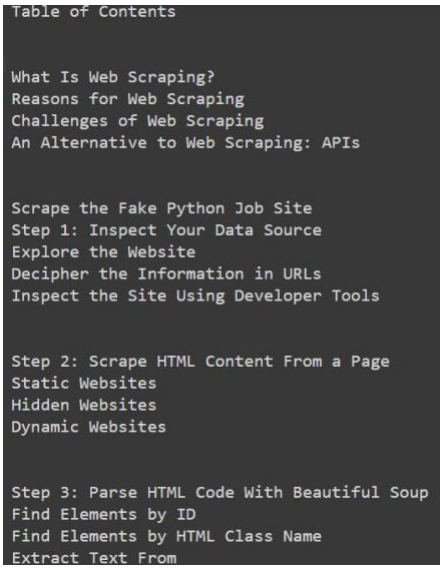
print(regexp_tagger.tag(test_sent))

```
<Regexp Tagger: size=9>
[('The', 'AT'), ('Fulton', 'NN'), ('County', 'NN'), ('Grand', 'NN'), ('Jury', 'NN'), ('said', 'NN'), ('Friday', 'NN'), ('an', 'AT'),
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

PRACTICAL NO.: 01

AIM: Web Scrapping: Scrape data from a webpage Date: 11/07/2024

```
import requests
from bs4 import BeautifulSoup

#Specify the url
url = "https://realpython.com/beautiful-soup-web-scraper-python/"

#Query the website and return the html to the variable 'page'
response = requests.get(url)

if response.status_code == 200:
  #Parse the page content with BeasutifulSoup
  soup = BeautifulSoup(page.content, 'html.parser')

  #Extract the main text content from the article
  #The content is usually in a <div> tag with a class like 'article-body'
  post_content = soup.find('div', class_='article-body').get_text()

  #Print the main text content
print(post_content[:500])
else:
  print(f"Failed to retrieve the page. Status code: {response.status_code}")
```




```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from collections import Counter

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

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

```
tokens = word_tokenize(post_content)

#Convert to lowercase and filter out non-alphabetic tokens
words = [word.lower() for word in tokens if word.isalpha()]

#Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in words if word not in stop_words]

#Calculate word frequencies
word_freq = Counter(filtered_words)

#print the most common words
print(word_freq.most_common(10))
```

```
[('html', 91), ('python', 81), ('job', 61), ('web', 49), ('elements', 45), ('code', 42), ('scraping', 41), ('information', 41), ('page', 39)
```

```
from nltk.sentiment import SentimentIntensityAnalyzer

#Download the vader lexicon
nltk.download('vader_lexicon') #words and rules (word ka sentiment probability kitna hai)

#Initialize the sentiment analyzer
sia = SentimentIntensityAnalyzer()

#Analyze the sentiment of the text
sentiment = sia.polarity_scores(post_content) #it gives score of positive negative neutral

#Print the sentiment scores
print(f"Sentiment Analysis: {sentiment}")
```

```
Sentiment Analysis: {'neg': 0.023, 'neu': 0.88, 'pos': 0.098, 'compound': 0.9999}
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
filtered_text = ' '.join(filtered_words)

#Analyze thr seniment of filtered content
sentiment = sia.polarity_scores(filtered_text)

#Print the sentiment scores
print(f"Sentiment Analysis: {sentiment}")
```

```
Sentiment Analysis: {'neg': 0.031, 'neu': 0.822, 'pos': 0.147, 'compound': 0.9999}
```

```
from operator import ne
import nltk
nltk.download('averaged_perceptron_tagger') #POS karne ke liye
nltk.download('maxent_ne_chunker') #words to phrases
#maxentoropy named entropy
```

```
#POS
pos_tags = nltk.pos_tag(filtered_words)

proper_nouns = [word for word, pos in pos_tags if pos in ('NNP', 'NNPS')]

#NER
ner_tags = nltk.ne_chunk(pos_tags)

#print named enitities
print("Named Entities:")
print(ner_tags)

#print proper nouns
print("Named Entities based on POS tags NNP and NNPs:")
print(proper_nouns)
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
Named Entities:
(S
  table/JJ
  contents/NNS
  web/VBP
  scraping/VBG
```

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

#Create a word cloud from the filtered words
wordcloud = WordCloud(width=800, height=400, background_color='white').generate('
'.join(filtered_words))

#Plot the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```