KNOWLEDGE INSTITUTE OF TECHNOLOGY

AN AUTONOMOUS INSTITUTION

Accredited by NAAC & NBA, Approved by AICTE, New Delhi and Affiliated To Anna University, Chennai

Kakapalayam (PO), Salem - 637 504.

RECORD NOTE BOOK

REG NO. _____

Certified that this is the bonafide record of work done by Selvan/Selvi……of the……..
Semester **ARTIFICIAL INTELLIGENCE & DATA SCIENCE** *Branch during the year* 2023-2024 *in* **CCS369 TEXT AND SPEECH ANALYSIS.**

Staff-Incharge *Head of the Department*

Submitted for the university Practical Examination on……

Internal Examiner *External Examiner*

CCS369 TEXT AND SPEECH ANALYSIS

CONTENTS

CCS369 TEXT AND SPEECH ANALYSIS

OBJECTIVES:

- Understand natural language processing basics
- Apply classification algorithms to text documents
- Build question-answering and dialogue systems
- Develop a speech recognition system
- Develop a speech synthesizer

LIST OF EXPERIMENTS:

1. Create Regular expressions in Python for detecting word patterns and tokenizing text
2. Getting started with Python and NLTK - Searching Text, Counting Vocabulary, Frequency Distribution, Collocations, Bigrams
3. Accessing Text Corpora using NLTK in Python
4. Write a function that finds the 50 most frequently occurring words of a text that are not stop words
5. Implement the Word2Vec model
6. Use a transformer for implementing classification
7. Design a chatbot with a simple dialog system
8. Convert text to speech and find accuracy
9. Design a speech recognition system and find the error rate

OUTCOMES:

At the end of this course, the students will be able to:

CO1: Explain existing and emerging deep learning architectures for text and speech processing
CO2: Apply deep learning techniques for NLP tasks, language modelling and machine translation
CO3: Explain coreference and coherence for text processing
CO4: Build question-answering systems, chatbots and dialogue systems
CO5: Apply deep learning models for building speech recognition and text-to-speech systems

1. Regular Expressions and Tokenizing Text

AIM:

To create Regular expressions in Python for detecting word patterns and tokenizing text

SOURCE CODE:

```
import re

# Example text
text = "Regular expressions are a powerful tool for pattern matching
and text processing."

# Define a regular expression pattern to match words
word_pattern = re.compile(r'\b\w+\b')

# Tokenize the text using the word pattern
tokens = word_pattern.findall(text)

# Print the tokens
print("Tokens:", tokens)

# Another example with more complex pattern
email_text = "Contact us at support@example.com or info@company.org
for assistance."

# Define a regular expression pattern to match email addresses
email_pattern = re.compile(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-
Za-z]{2,}\b')

# Find email addresses in the text
emails = email_pattern.findall(email_text)

# Print the found email addresses
print("Email addresses:", emails)
```

OUTPUT:

Tokens: ['Regular', 'expressions', 'are', 'a', 'powerful', 'tool', 'for', 'pattern', 'matching', 'and', 'text', 'processing']
Email addresses: ['support@example.com', 'info@company.org']

RESULT:

Thus the program for creating Regular expression was executed and verified successfully.

2. Searching Text, Counting Vocabulary, Frequency Distribution, Collocations, Bigrams

AIM:

To demonstrate and Getting started with Python and NLTK - Searching Text, Counting vocabulary, Frequency distribution, Collocations, Bigrams.

SOURCE CODE:

```
# Import necessary modules from NLTK import nltk
import nltk
nltk.download('punkt')
from nltk import FreqDist, bigrams
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures
from nltk.corpus import reuters


# Download NLTK resources (if not already downloaded) nltk.download('reuters')
nltk.download('stopwords')
nltk.download('reuters')
documents = reuters.fileids()

# Load the Reuters corpus for demonstration documents = reuters.fileids()
corpus = [reuters.raw(doc_id) for doc_id in documents]


# Search for a specific word in the corpus
search_word = 'oil'
oil_docs = [doc_id for doc_id in documents if search_word in reuters.words(doc_id)]
print(f"Documents containing the word '{search_word}': {oil_docs}")

# Count vocabulary and create a frequency distribution
words = nltk.word_tokenize(" ".join(corpus))
lowercase_words = [word.lower() for word in words if word.isalpha()]
# Remove non- alphabetic tokens
fdist = FreqDist(lowercase_words)

# Print the most common words and their frequencies
print("Most common words and their frequencies:")
```

```
for word, frequency in fdist.most_common(10):
  print(f"{word}: {frequency}")


# Find collocations (bigram phrases that occur frequently together)
bigram_measures = BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(lowercase_words)
collocations = finder.nbest(bigram_measures.raw_freq, 10)

# Print the most common collocations
print("\nMost common collocations:")
for collocation in collocations:
  print(" ".join(collocation))


# Create bigrams and calculate their frequencies
bi_grams = list(bigrams(lowercase_words))
bi_gram_fdist = FreqDist(bi_grams)

# Print the most common bigrams and their frequencies
print("\nMost common bigrams and their frequencies:")
for bigram, frequency in bi_gram_fdist.most_common(10):
  print(f"{bigram}: {frequency}")
```

OUTPUT:

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package reuters to /root/nltk_data...
[nltk_data]   Package reuters is already up-to-date!
Documents containing the word 'oil': ['test/14829', 'test/14832', 'test/14833', 'test/14840', 'test/14863',
'test/14873', 'test/14891', 'test/14892', 'test/14931', 'test/15038', 'test/15063', 'test/15198', 'test/15200',
'test/15212', 'test/15227', 'test/15230', 'test/15238', 'test/15244', 'test/15250', 'test/15322', 'test/15325',
'test/15341', 'test/15344', 'test/15351', 'test/15366', 'test/15386', 'test/15389', 'test/15396', 'test/15416',
'test/15500', 'test/15551', 'test/15573', 'test/15607', 'test/15639', 'test/15733', 'test/15829', 'test/15875',
'test/15923', 'test/15939', 'test/15975', 'test/16005', 'test/16007', 'test/16077', 'test/16080', 'test/16093',
'test/16125', 'test/16130', 'test/16155', 'test/16166', 'test/16176', 'test/16195', 'test/16214', 'test/16215',
'test/16226', 'test/16252', 'test/16268', 'test/16270', 'test/16366', 'test/16429', 'test/16438', 'test/16463',
'test/16483', 'test/16589', 'test/16593', 'test/16597', 'test/16604', 'test/16607', 'test/16636', 'test/16649',
'test/16651', 'test/16658', 'test/16680', 'test/16710', 'test/16723', 'test/16739', 'test/16762', 'test/16833',
'test/17054', 'test/17441', 'test/17446', 'test/17452', 'test/17473', 'test/17477', 'test/17478', 'test/17516',
'test/17519', 'test/17618', 'test/17750', 'test/17759', 'test/17771', 'test/17780', 'test/17813', 'test/17816',
'test/17875', 'test/17878', 'test/17883', 'test/17886', 'test/17888', 'test/17892', 'test/17896', 'test/17913',
'test/17924', 'test/17925', 'test/17926', 'test/17929', 'test/17958', 'test/17963', 'test/18066', 'test/18085',
'test/18108', 'test/18146', 'test/18150', 'test/18186', 'test/18213', 'test/18228', 'test/18231', 'test/18234',
'test/18271', 'test/18325', 'test/18328', 'test/18329', 'test/18332', 'test/18340', 'test/18367', 'test/18480',
'test/18493', 'test/18521', 'test/18523', 'test/18567', 'test/18621', 'test/18651', 'test/18689', 'test/18698',
'test/18728', 'test/18736', 'test/18738', 'test/18744', 'test/18746', 'test/18754', 'test/18765', 'test/18773',
'test/18774', 'test/18781', 'test/18795', 'test/18798', 'test/18810', 'test/18835', 'test/18840', 'test/18857',
```

'test/18896', 'test/18990', 'test/19017', 'test/19059', 'test/19069', 'test/19082', 'test/19083', 'test/19088',
'test/19110', 'test/19128', 'test/19193', 'test/19223', 'test/19285', 'test/19291', 'test/19367', 'test/19397',
'test/19403', 'test/19478', 'test/19492', 'test/19497', 'test/19499', 'test/19505', 'test/19506', 'test/19509',
'test/19556', 'test/19559', 'test/19672', 'test/19684', 'test/19756', 'test/19832', 'test/19844', 'test/19869',
'test/19874', 'test/19882', 'test/19903', 'test/19927', 'test/19947', 'test/19984', 'test/19996', 'test/19998',
'test/20008', 'test/20030', 'test/20045', 'test/20090', 'test/20092', 'test/20093', 'test/20095', 'test/20101',
'test/20103', 'test/20232', 'test/20270', 'test/20333', 'test/20352', 'test/20406', 'test/20420', 'test/20459',
'test/20464', 'test/20474', 'test/20517', 'test/20632', 'test/20653', 'test/20682', 'test/20709', 'test/20719',
'test/20721', 'test/20756', 'test/20774', 'test/20778', 'test/20828', 'test/20869', 'test/20878', 'test/20881',
'test/20882', 'test/20890', 'test/20902', 'test/20909', 'test/20919', 'test/20936', 'test/20959', 'test/20981',
'test/20991', 'test/21002', 'test/21006', 'test/21013', 'test/21018', 'test/21067', 'test/21076', 'test/21131',
'test/21149', 'test/21197', 'test/21216', 'test/21267', 'test/21363', 'test/21369', 'test/21417', 'test/21443',
'test/21459', 'test/21475', 'test/21482', 'test/21485', 'test/21486', 'test/21492', 'test/21501', 'test/21502',
'test/21506', 'test/21510', 'test/21519', 'test/21525', 'test/21541', 'test/21561', 'test/21568', 'training/10011',
'training/10078', 'training/10080', 'training/10091', 'training/10106', 'training/10135', 'training/10175',
'training/10190', 'training/10192', 'training/10200', 'training/10228', 'training/1024', 'training/1026',
'training/10261', 'training/10268', 'training/10272', 'training/10275', 'training/10291', 'training/10292',
'training/10300', 'training/10306', 'training/10330', 'training/10341', 'training/10348', 'training/10371',
'training/10373', 'training/10375', 'training/10385', 'training/10395', 'training/10406', 'training/10452',
'training/10539', 'training/10567', 'training/10588', 'training/10620', 'training/10621', 'training/10627',
'training/10632', 'training/10641', 'training/10693', 'training/10703', 'training/10718', 'training/10720',
'training/10750', 'training/10758', 'training/10797', 'training/10816', 'training/1084', 'training/10845',
'training/10847', 'training/10873', 'training/10944', 'training/10947', 'training/11000', 'training/11007',
'training/11025', 'training/1110', 'training/11118', 'training/11145', 'training/11149', 'training/11171',
'training/11177', 'training/11213', 'training/11224', 'training/11227', 'training/11231', 'training/11232',
'training/11236', 'training/11241', 'training/11249', 'training/11273', 'training/11291', 'training/11350',
'training/11388', 'training/11403', 'training/11406', 'training/11444', 'training/11455', 'training/11491',
'training/11522', 'training/11559', 'training/11588', 'training/11632', 'training/11639', 'training/11671',
'training/11711', 'training/11723', 'training/11724', 'training/11752', 'training/11753', 'training/11768',
'training/11778', 'training/11839', 'training/11852', 'training/11880', 'training/11882', 'training/11886',
'training/11898', 'training/11949', 'training/12050', 'training/1211', 'training/12122', 'training/12149',
'training/12166', 'training/12209', 'training/12344', 'training/12345', 'training/12361', 'training/12454',
'training/12503', 'training/12533', 'training/127', 'training/12746', 'training/12775', 'training/12786',
'training/12791', 'training/12799', 'training/12803', 'training/12818', 'training/12822', 'training/12851',
'training/12986', 'training/1301', 'training/1306', 'training/13080', 'training/13096', 'training/13102',
'training/13115', 'training/13142', 'training/1316', 'training/13179', 'training/13184', 'training/13200',
'training/13245', 'training/13247', 'training/13256', 'training/13265', 'training/13266', 'training/13320',
'training/1343', 'training/13539', 'training/13542', 'training/13611', 'training/13738', 'training/1379',
'training/1387', 'training/13915', 'training/13963', 'training/140', 'training/14211', 'training/14395', 'training/144',
'training/145', 'training/14509', 'training/14558', 'training/1456', 'training/14679', 'training/14698',
'training/14732', 'training/14770', 'training/1550', 'training/1556', 'training/1616', 'training/1650', 'training/1658',
'training/1660', 'training/1692', 'training/1696', 'training/1709', 'training/1711', 'training/1723', 'training/1751',
'training/1856', 'training/1875', 'training/1878', 'training/1909', 'training/191', 'training/1920', 'training/194',
'training/1948', 'training/1959', 'training/1964', 'training/1980', 'training/1990', 'training/200', 'training/2007',
'training/2046', 'training/2061', 'training/2074', 'training/2121', 'training/213', 'training/2173', 'training/2175',
'training/2187', 'training/2228', 'training/2251', 'training/235', 'training/236', 'training/237', 'training/2383',
'training/242', 'training/2420', 'training/2435', 'training/2449', 'training/2456', 'training/246', 'training/247',
'training/2475', 'training/2479', 'training/248', 'training/2483', 'training/2511', 'training/2515', 'training/2517',
'training/2522', 'training/2530', 'training/2542', 'training/2585', 'training/2688', 'training/2696', 'training/273',
'training/274', 'training/2767', 'training/2775', 'training/2789', 'training/2828', 'training/2833', 'training/2838',
'training/2925', 'training/2957', 'training/2970', 'training/2973', 'training/2975', 'training/2998', 'training/3003',
'training/3015', 'training/3017', 'training/3019', 'training/3024', 'training/3048', 'training/3065', 'training/3115',
'training/313', 'training/3145', 'training/3169', 'training/3174', 'training/3181', 'training/3189', 'training/320',
'training/3204', 'training/3206', 'training/3249', 'training/3269', 'training/3303', 'training/3310', 'training/332',
'training/3332', 'training/3338', 'training/3342', 'training/3364', 'training/3372', 'training/3389', 'training/3430',
'training/3445', 'training/3452', 'training/3455', 'training/3488', 'training/349', 'training/3490', 'training/3505',
'training/3507', 'training/3509', 'training/352', 'training/353', 'training/3535', 'training/3540', 'training/3556',
'training/3563', 'training/3571', 'training/3592', 'training/3593', 'training/3594', 'training/3597', 'training/3609',
'training/368', 'training/3798', 'training/3800', 'training/3840', 'training/3843', 'training/3846', 'training/3853',
'training/3855', 'training/3864', 'training/3869', 'training/3872', 'training/3906', 'training/3950', 'training/3980',
'training/3985', 'training/3995', 'training/4005', 'training/4016', 'training/4028', 'training/4039', 'training/4041',
'training/4049', 'training/4061', 'training/4067', 'training/4080', 'training/4125', 'training/4129', 'training/4138',
'training/4162', 'training/4171', 'training/4174', 'training/4199', 'training/4232', 'training/4246', 'training/4249',
'training/4290', 'training/4305', 'training/4315', 'training/4328', 'training/4338', 'training/4340', 'training/4353',

'training/4365', 'training/4367', 'training/4386', 'training/4425', 'training/4429', 'training/4453', 'training/4467', 'training/4474', 'training/4481', 'training/4525', 'training/4564', 'training/4569', 'training/4573', 'training/4576', 'training/4584', 'training/4590', 'training/4593', 'training/4600', 'training/4604', 'training/4609', 'training/4634', 'training/4650', 'training/4662', 'training/4679', 'training/4681', 'training/4689', 'training/4713', 'training/4742', 'training/4744', 'training/4755', 'training/4785', 'training/4835', 'training/4848', 'training/4867', 'training/489', 'training/4908', 'training/4930', 'training/4951', 'training/4953', 'training/4962', 'training/4963', 'training/4981', 'training/4983', 'training/502', 'training/5037', 'training/5044', 'training/5061', 'training/5116', 'training/5118', 'training/5119', 'training/5123', 'training/5125', 'training/5142', 'training/5145', 'training/5150', 'training/5152', 'training/5156', 'training/5167', 'training/5169', 'training/5171', 'training/5178', 'training/5193', 'training/5218', 'training/5236', 'training/5238', 'training/5244', 'training/5250', 'training/5255', 'training/5268', 'training/5270', 'training/5273', 'training/5274', 'training/5281', 'training/5292', 'training/5295', 'training/5315', 'training/5318', 'training/5323', 'training/5330', 'training/5342', 'training/5376', 'training/5389', 'training/543', 'training/5538', 'training/5542', 'training/5544', 'training/5553', 'training/5559', 'training/5561', 'training/5655', 'training/5675', 'training/5683', 'training/5684', 'training/5692', 'training/5706', 'training/5712', 'training/5739', 'training/5761', 'training/5769', 'training/5787', 'training/5791', 'training/5793', 'training/5796', 'training/5830', 'training/5848', 'training/5852', 'training/5866', 'training/5879', 'training/5887', 'training/5949', 'training/5985', 'training/6', 'training/6023', 'training/6037', 'training/6054', 'training/6060', 'training/6086', 'training/6111', 'training/6119', 'training/6121', 'training/6125', 'training/6163', 'training/6169', 'training/6177', 'training/6184', 'training/6201', 'training/6208', 'training/6225', 'training/6264', 'training/6271', 'training/6301', 'training/6342', 'training/6344', 'training/6348', 'training/6371', 'training/6404', 'training/6412', 'training/6413', 'training/6421', 'training/6432', 'training/6436', 'training/6535', 'training/6562', 'training/6573', 'training/6578', 'training/6598', 'training/6606', 'training/6638', 'training/6652', 'training/6656', 'training/6708', 'training/6722', 'training/6740', 'training/6742', 'training/6746', 'training/6760', 'training/68', 'training/6876', 'training/6893', 'training/6905', 'training/6922', 'training/6954', 'training/697', 'training/6994', 'training/6996', 'training/704', 'training/7067', 'training/708', 'training/7097', 'training/7119', 'training/7135', 'training/7150', 'training/7174', 'training/7287', 'training/7355', 'training/7366', 'training/739', 'training/7397', 'training/7408', 'training/7423', 'training/7462', 'training/7496', 'training/7500', 'training/7529', 'training/7534', 'training/7548', 'training/7606', 'training/7611', 'training/7618', 'training/7639', 'training/7642', 'training/7643', 'training/7742', 'training/7790', 'training/7854', 'training/791', 'training/7937', 'training/8003', 'training/8015', 'training/8032', 'training/8039', 'training/8041', 'training/8050', 'training/8051', 'training/8069', 'training/8086', 'training/8089', 'training/8100', 'training/8109', 'training/8117', 'training/8119', 'training/8131', 'training/8134', 'training/8149', 'training/8156', 'training/8159', 'training/8160', 'training/8167', 'training/8173', 'training/8188', 'training/8209', 'training/8210', 'training/829', 'training/834', 'training/835', 'training/837', 'training/8421', 'training/843', 'training/8440', 'training/8478', 'training/8493', 'training/8516', 'training/8530', 'training/855', 'training/8596', 'training/8598', 'training/8600', 'training/8606', 'training/8610', 'training/8615', 'training/8623', 'training/8630', 'training/8672', 'training/8675', 'training/8688', 'training/873', 'training/8747', 'training/8755', 'training/8765', 'training/8780', 'training/8815', 'training/8820', 'training/8835', 'training/885', 'training/8856', 'training/8882', 'training/8884', 'training/8905', 'training/8914', 'training/8959', 'training/896', 'training/8964', 'training/8971', 'training/9031', 'training/9065', 'training/9077', 'training/9149', 'training/915', 'training/9155', 'training/9156', 'training/918', 'training/9193', 'training/9206', 'training/9208', 'training/9213', 'training/9253', 'training/9279', 'training/9293', 'training/930', 'training/9352', 'training/9381', 'training/9392', 'training/9436', 'training/944', 'training/9445', 'training/945', 'training/9479', 'training/952', 'training/9550', 'training/9583', 'training/9634', 'training/9639', 'training/9650', 'training/9674', 'training/9706', 'training/9718', 'training/9733', 'training/9734', 'training/9736', 'training/9756', 'training/9761', 'training/9763', 'training/9769', 'training/9770', 'training/9784', 'training/9799', 'training/9801', 'training/9821', 'training/9848', 'training/9849', 'training/9853', 'training/9913', 'training/9947', 'training/9952']
Most common words and their frequencies:
the: 69245
of: 36749
to: 36275
in: 29217
and: 25616
said: 25381
a: 24724
mln: 18598
vs: 14332
for: 13420

Most common collocations:
in the
of the
said the
mln dlrs
said it
vs mln

```
mln vs
cts vs
the company
for the

Most common bigrams and their frequencies:
('in', 'the'): 7103
('of', 'the'): 6915
('said', 'the'): 5355
('mln', 'dlrs'): 4471
('said', 'it'): 4366
('vs', 'mln'): 3946
('mln', 'vs'): 3919
('cts', 'vs'): 3311
('the', 'company'): 3090
('for', 'the'): 2811
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
 # Ensure stopwords are downloaded
import nltk
nltk.download('stopwords', quiet=True)
 # Load English stopwords
stop_words = set(stopwords.words('english'))
 # Example text
text = "This is an example sentence with some common stopwords."
 # Tokenize the text
words = word_tokenize(text)
# Remove stopwords
filtered_words = [word for word in words if word.lower() not in
stop_words]
print("Original words:", words)
print("Words after removing stopwords:", filtered_words)
```

OUTPUT:

```
Original words: ['This', 'is', 'an', 'example', 'sentence', 'with', 'some', 'common', 'stopwords', '.']
Words after removing stopwords: ['example', 'sentence', 'common', 'stopwords', '.']
```

RESULT:

Thus the program for Searching Text, Counting Vocabulary, Frequency distribution, Collocations, Bigrams was executed and verified successfully.

3. Text Corpora using NLTK in Python

AIM:

To implement and accessing Text Corpora using NLTK in Python.

SOURCE CODE:

```
from nltk.corpus import gutenberg
nltk.download('gutenberg')
hamlet_sents = gutenberg.sents('shakespeare-hamlet.txt')
print(hamlet_sents) # prints out the introduction

print() # prints out a blank line
print(hamlet_sents[19]) # finds one particular sentence by its number
in a text
print() # prints out a blank line
longest_sent = max(len(s) for s in hamlet_sents) # prints out a number
of the longest sentence
shortest_sent = min(len(s) for s in hamlet_sents) # prints out a
number of the shortest sentence
print(longest_sent)
print(shortest_sent)
print([s for s in hamlet_sents if len(s) == longest_sent])
print([s for s in hamlet_sents if len(s) == shortest_sent]) # may
print out some "shortest sentences"
```

OUTPUT:

```
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Unzipping corpora/gutenberg.zip.
[['[', 'The', 'Tragedie', 'of', 'Hamlet', 'by', 'William', 'Shakespeare', '1599', ']'], ['Actus', 'Primus', '.'], ...]

['For', 'this', 'releefe', 'much', 'thankes', ':', "'", 'Tis', 'bitter', 'cold', ',', 'And', 'I', 'am', 'sicke', 'at', 'heart']

174
1
[['To', 'thine', 'owne', 'peace', ':', 'if', 'he', 'be', 'now', 'return', "'", 'd', ',', 'As', 'checking', 'at', 'his', 'Voyage', ',', 'and', 'that', 'he', 'meanes', 'No', 'more', 'to', 'vndertake', 'it', ';', 'I', 'will', 'worke', 'him', 'To', 'an', 'exployt', 'now', 'ripe', 'in', 'my', 'Deuice', ',', 'Vnder', 'the', 'which', 'he', 'shall', 'not', 'choose', 'but', 'fall', ';', 'And', 'for', 'his', 'death', 'no', 'winde', 'of', 'blame', 'shall', 'breath', ',', 'But', 'euen', 'his', 'Mother', 'shall', 'vncharge', 'the', 'practice', ',', 'And', 'call', 'it', 'accident', ':', 'Some', 'two', 'Monthes', 'hence', 'Here', 'was', 'a', 'Gentleman', 'of', 'Normandy', ',', 'I', "'", 'ue', 'seene', 'my', 'selfe', ',', 'and', 'seru', "'", 'd', 'against', 'the', 'French', ',', 'And', 'they', 'ran', 'well', 'on', 'Horsebacke', ';', 'but', 'this', 'Gallant', 'Had', 'witchcraft', 'in', "'", 't', ';', 'he', 'grew', 'into', 'his', 'Seat', ',', 'And', 'to', 'such', 'wondrous', 'doing', 'brought', 'his', 'Horse', ',', 'As', 'had', 'he', 'beene', 'encorps', "'", 't', 'and', 'demy', '-', 'Natur', "'", 'd', 'With', 'the', 'braue', 'Beast', ',', 'so', 'farre', 'he', 'past', 'my', 'thought', ',', 'That', 'I', 'in', 'forgery', 'of', 'shapes', 'and', 'trickes', ',', 'Come', 'short', 'of', 'what', 'he', 'did']]
[['He'], ['Marcellus'], ['no'], ['Exeunt'], ['Hamlet'], ['Sweare'], ['Sweare'], ['Sweare'], ['Exit'], ['Farewell'], ['Farwell'], ['Nothing'], ['Sleepes'], ['Exit'], ['Away'], ['Good'], ['Exit'], ['Exit'], ['Dead'], ['Ophelia'], ['Puh'], ['One'], ['No'], ['Iudgement'], ['Dyes']]
```

1. Gutenberg Corpus

NLTK includes a small selection of texts from the Project Gutenberg electronic text archive, which contains 25,000 free electronic books, hosted at http://www.gutenberg.org/. We begin by getting the Python interpreter to load the NLTK package, then ask to see nltk.corpus.gutenberg.fileids(), the file identifiers in this corpus:

CODE:

```
import nltk
file_ids = nltk.corpus.gutenberg.fileids()
print(file_ids)

print()# prints out a blank line
hamlet = nltk.corpus.gutenberg.words('shakespeare-hamlet.txt')
print("the total amount of words in 'shakespeare-hamlet.txt': " +
str(len(hamlet)))
# prints out how many words it contains
```

OUTPUT:

```
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt', 'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt', 'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt', 'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.txt', 'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']

the total amount of words in 'shakespeare-hamlet.txt': 37360
```

CODE:

```
from nltk.corpus import gutenberg
for w in gutenberg.fileids():
  print(w)

for files in gutenberg.fileids():
  num_char = len(gutenberg.raw(files))
  num_words = len(gutenberg.words(files))
  num_sents = len(gutenberg.sents(files))
  num_vocab = len(set(w.lower() for w in gutenberg.words(files)))
  print(round(num_char/num_words),          round(num_words/num_sents),
round(num_words/num_vocab), files)
# ctrl + s = stop
# ctrl + q = continue # ctrl + c = cansel
```

OUTPUT:

```
austen-emma.txt
austen-persuasion.txt
austen-sense.txt
bible-kjv.txt
blake-poems.txt
bryant-stories.txt
burgess-busterbrown.txt
carroll-alice.txt
chesterton-ball.txt
chesterton-brown.txt
chesterton-thursday.txt
edgeworth-parents.txt
melville-moby_dick.txt
milton-paradise.txt
shakespeare-caesar.txt
shakespeare-hamlet.txt
shakespeare-macbeth.txt
whitman-leaves.txt
5 25 26 austen-emma.txt
5 26 17 austen-persuasion.txt
5 28 22 austen-sense.txt
4 34 79 bible-kjv.txt
5 19 5 blake-poems.txt
4 19 14 bryant-stories.txt
4 18 12 burgess-busterbrown.txt
4 20 13 carroll-alice.txt
5 20 12 chesterton-ball.txt
5 23 11 chesterton-brown.txt
5 18 11 chesterton-thursday.txt
4 21 25 edgeworth-parents.txt
5 26 15 melville-moby_dick.txt
5 52 11 milton-paradise.txt
4 12 9 shakespeare-caesar.txt
4 12 8 shakespeare-hamlet.txt
4 12 7 shakespeare-macbeth.txt
5 36 12 whitman-leaves.txt
```

2.Web-Text

Project Gutenberg represents established literature. It is important to consider less formal la nguage as well. NLTK's small collection of web text includes content from a Firefox discussi on forum, conversations overheard in New York, the movie script of Pirates of the Carribean, personal advertisements, and wine reviews

CODE:

```
import nltk
nltk.download('webtext')
from nltk.corpus import webtext
for files in webtext.fileids():
  print(files, webtext.raw(files)[:11], "...")
wine = nltk.corpus.webtext.words("wine.txt")

print("the number of words in 'wine.txt' is: " + str(len(wine)))
# you should convert "len" into a "str" so python can combine two strings.
```

OUTPUT:

```
firefox.txt Cookie Mana ...
grail.txt SCENE 1: [w ...
overheard.txt White guy:  ...
pirates.txt PIRATES OF  ...
singles.txt 25 SEXY MAL ...
wine.txt Lovely deli ...
the number of words in 'wine.txt' is: 31350
```

There is also a corpus of instant messaging chat sessions, originally collected by the Naval Postgraduate School. The corpus contains over 10,000 posts, anonymized by replacing usernames with generic names of the form "UserNNN". The corpus is organized into 15 files, where each file contains several hundred posts collected on a given date, for an age-specific chatroom (teens, 20s, 30s, 40s, plus a generic adults chatroom).
The filename contains the date, chatroom, and number of posts; e.g., 10-19-20s_706posts.xml contains 706 posts gathered from the 20s chat room on 10/19/2006.

CODE:

```
import nltk
nltk.download('nps_chat')
from nltk.corpus import nps_chat
chatroom = nps_chat.posts("10-19-20s_706posts.xml")
print(chatroom[123])
```

OUTPUT:

```
['i', 'do', "n't", 'want', 'hot', 'pics', 'of', 'a', 'female', ',', 'I', 'can', 'look', 'in', 'a', 'mirror', '.']
```

The Brown Corpus was the first million-word electronic corpus in English, created in 1961 at Brown University. This corpus contains text from 500 sources, and the sources have been categorized by genre, such as news, editorial, and so on(for a complete genre-list, see http://icame.uib.no/brown/bcm-los.html).

```
import nltk
from nltk.corpus import brown

print(brown.categories())
print()

print(brown.words(categoric
```

```
s="humor")) print()

print(brown.words(fileid
s=["ch15"])) print()

print(brown.sents(categories=["mystery","science_fiction",
"adventure"]))
```

OUTPUT:

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
['adventure', 'belles_lettres', 'editorial', 'fiction', 'government', 'hobbies', 'humor', 'learned', 'lore', 'mystery', 'news',
'religion', 'reviews', 'romance', 'science_fiction']

['It', 'was', 'among', 'these', 'that', 'Hinkle', ...]

['At', 'the', 'entrance', 'side', 'of', 'the', ...]

[['There', 'were', 'thirty-eight', 'patients', 'on', 'the', 'bus', 'the', 'morning', 'I', 'left', 'for', 'Hanover', ',', 'most', 'of', 'them',
'disturbed', 'and', 'hallucinating', '.'], ['An', 'interne', ',', 'a', 'nurse', 'and', 'two', 'attendants', 'were', 'in', 'charge', 'of', 'us',
'.'], ...]
```

CODE:

```
for q in questions:
  print(q + ":", frequency_distribution[q], ", ", end=" ")

# We need to include end=' ' in order
for the print function to put its
output on a single line.
```

OUTPUT:

why: 8 , where: 15 , when: 28 , who: 13 , what: 41 ,

CODE:

```
import nltk
from nltk.corpus import brown
from nltk.probability import ConditionalFreqDist


cond_freq_dist           =
  ConditionalFreqDist(
  (g,w)
  for g in brown.categories()

  for w in brown.words(categories=g))

genres                                          =
["government","fiction","mystery","science_fiction
","adventure"]            modal_verbs           =
["may","can","could","should","must","might","will
"]
```

```
print(cond_freq_dist.tabulate(conditions=genres,samples=modal_verbs)
)

# Conditional frequency distributions are used to record the
number of times each sample occ urred.
#   Conditional   frequency   distributions   are   typically
constructed by repeatedly running an expe riment under a
variety of condition
```

OUTPUT:

```
#              may  can  could should  must   might   will
#   government   153   117    38    112  102      13    244
#      fiction    8    37   166     35    55     44     52
#      mystery    13    42   141     29  30     57     20
#science_fiction     4    16    493      8     12     16
#    adventure     5    46   151     15  27     58     50
## None
```

RESULT:

Thus the program to implement and accessing Text Corpora was executed and verified successfully.

4. Text that are not Stop words.

AIM:

Write a function that finds the 50 most frequently occurring words of a text that are not stop words

SOURCE CODE:

```
import nltk
from nltk.corpus import stopwords
from nltk import FreqDist
from nltk.tokenize import word_tokenize
def most_frequent_words(text, num_words=50): # Download NLTK stopwords
data nltk.download('stopwords')
# Download NLTK punkt tokenizer nltk.download('punkt')
# Tokenize the text
  words = word_tokenize(text.lower()) # Convert to lowercase for case-
insensitive comparison
# Remove stop words
  stop_words = set(stopwords.words('english'))
  filtered_words = [word for word in words if word.isalpha() and word
not in stop_words]
# Calculate word frequencies
  fdist = FreqDist(filtered_words)
# Get the most common words
  most_common_words = fdist.most_common(num_words)
  return most_common_words
# Example usage
sample_text = """Natural language processing (NLP) is a field of
artificial intelligence that focuses on the inter action between
computers and humans through natural language. It enables computers to
und erstand, interpret, and generate human-like text. NLP is used in
various applications, includin g machine translation, sentiment
analysis, and chatbot development."""
result = most_frequent_words(sample_text, num_words=50)
print("50 Most Frequent Words (excluding stopwords):")
for word, frequency in result:
  print(f"{word}: {frequency}")
```

OUTPUT:

```
50 Most Frequent Words (excluding stopwords):
natural: 2
language: 2
nlp: 2
computers: 2
processing: 1
field: 1
artificial: 1
intelligence: 1
focuses: 1
inter: 1
action: 1
humans: 1
enables: 1
und: 1
```

```
erstand: 1
interpret: 1
generate: 1
text: 1
used: 1
various: 1
applications: 1
includin: 1
g: 1
machine: 1
translation: 1
sentiment: 1
analysis: 1
chatbot: 1
development: 1
```

RESULT:

Thus the program to write a function that finds the 50 most frequently occurring words of a text that are not stop words was executed and verified successfully.

5.Word2Vec Model

AIM:

To Implement the Word2Vec model

SOURCE CODE:

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize # You can use any tokenizer you prefer
import nltk
nltk.download("punkt") # Download the punkt tokenizer
# Sample data (replace this with your own text data)
corpus = "Word embeddings are a type of word representation that allows words to be represented as vectors in a continuous vector space."
# Tokenize the text
tokenized_text = word_tokenize(corpus.lower()) # Convert to lowercase for consistency
# Define the Word2Vec model
model = Word2Vec(sentences=[tokenized_text], vector_size=100, window=5, sg=1, min_count=1)
# Training the Word2Vec model
model.train([tokenized_text], total_examples=1, epochs=10)
# Save the model
model.save("word2vec.model")
# Load the model
model = Word2Vec.load("word2vec.model")
# Get the vector representation of a word
vector = model.wv["word"]
# Find similar words
similar_words = model.wv.most_similar("word", topn=5)
# Print results
print("Vector for word", vector)
print("Similar words to word" ,similar_words)
```

OUTPUT:

```
Vector for word [-5.5466092e-04  2.5012434e-04  5.0892807e-03  9.0372479e-03
 -9.2822211e-03 -7.1585788e-03  6.4871050e-03  9.0257684e-03
 -5.0597130e-03 -3.8020581e-03  7.3782010e-03 -1.5786611e-03
 -4.5337854e-03  6.5808198e-03 -4.8375176e-03 -1.8134720e-03
  2.9152322e-03  1.0071403e-03 -8.2673877e-03 -9.4795758e-03
  7.3322230e-03  5.0640488e-03  6.7715142e-03  7.3392037e-04
  6.3549490e-03 -3.4045286e-03 -9.8645489e-04  5.7906061e-03
 -7.5067324e-03 -3.9466550e-03 -7.4604200e-03 -9.4752293e-04
  9.5319245e-03 -7.3126010e-03 -2.3547220e-03 -1.8911593e-03
  8.0995485e-03 -5.9078019e-03  5.3631786e-05 -4.7513954e-03
 -9.5484406e-03  4.9795178e-03 -8.7561142e-03 -4.3622586e-03
 -2.5252801e-05 -2.6343751e-04 -7.6987674e-03  9.5928898e-03
  5.0103413e-03  9.2233568e-03 -8.1536593e-03  4.4730781e-03
 -4.1411328e-03  8.3059812e-04  8.4971115e-03 -4.4478630e-03
  4.5438502e-03 -6.7874910e-03 -3.5072158e-03  9.4045494e-03
 -1.5942811e-03  3.2952611e-04 -4.0926719e-03 -7.6800385e-03
 -1.5111530e-03  2.4983983e-03 -8.8613341e-04  5.5593839e-03
 -2.7505371e-03  2.2510567e-03  5.4544555e-03  8.3404295e-03
 -1.4181926e-03 -9.1825062e-03  4.4081411e-03  5.6442770e-04
  7.4427784e-03 -7.8674732e-04 -2.6436201e-03 -8.7705292e-03
 -9.0044236e-04  2.8165644e-03  5.4136524e-03  7.0741391e-03
 -5.6783408e-03  1.8377738e-03  6.1044549e-03 -4.8125032e-03
 -3.0868063e-03  6.7811562e-03  1.6387857e-03  2.1426704e-04
  3.4865059e-03  2.2381512e-04  9.6579231e-03  5.0731204e-03
 -8.8937553e-03 -7.0351120e-03  9.3331042e-04  6.4113727e-03]
Similar    words    to    word    [('in',    0.21906763315200806),    ('allows',
0.21704821288585663),    ('representation',    0.09334444254636765),    ('be',
0.09298395365476608), ('to', 0.08011001348495483)]
```

```
import numpy as np
from collections import defaultdict
from sklearn.utils import shuffle
from sklearn.metrics.pairwise import cosine_similarity

class Word2Vec:
    def      __init__(self,      vector_size=100,      window=5,
negative_samples=5, learning_rate=0.025):
        self.vector_size = vector_size
        self.window = window
        self.negative_samples = negative_samples
        self.learning_rate = learning_rate
        self.words = set()
        self.word_index = {}
        self.word_count = defaultdict(int)
        self.word_vectors = None

    def tokenize(self, corpus):
        tokens = corpus.split()
        return [token.lower() for token in tokens]

    def generate_training_data(self, corpus):
```

```
        tokens = self.tokenize(corpus)
        for i, target_word in enumerate(tokens):
            for j in range(max(0, i - self.window), min(i +
self.window, len(tokens))):
                if i != j:
                    context_word = tokens[j]
                    yield target_word, context_word

    def initialize_vectors(self):
        self.word_vectors = np.random.rand(len(self.words),
self.vector_size)

    def train(self, corpus, epochs=10):
        for epoch in range(epochs):
            loss = 0
            training_data                                    =
list(self.generate_training_data(corpus))
            training_data = shuffle(training_data)
            for target_word, context_word in training_data:
                loss      +=      self.skip_gram(target_word,
context_word)
            print(f"Epoch {epoch + 1}, Loss: {loss}")

    def skip_gram(self, target_word, context_word):
        target_index = self.word_index[target_word]
        context_index = self.word_index[context_word]
        # Positive example
        score     =     np.dot(self.word_vectors[target_index],
self.word_vectors[context_index])
        prob = self.sigmoid(score)
        loss = -np.log(prob)
        # Negative examples
        negative_indices  =  np.random.choice(len(self.words),
size=self.negative_samples, replace=False)
        for neg_index in negative_indices:
            score   =   np.dot(self.word_vectors[target_index],
self.word_vectors[neg_index])
            prob = self.sigmoid(score)
            loss -= np.log(1 - prob)
            # Update negative sample vector
            self.word_vectors[neg_index] -= self.learning_rate
* (1 - prob) * self.word_vectors[target_index]
        # Update target and context vectors
        self.word_vectors[target_index]  -=  self.learning_rate
* (1 - prob) * self.word_vectors[context_index]
        self.word_vectors[context_index] -= self.learning_rate
* (1 - prob) * self.word_vectors[target_index]
        return loss
```

```
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def build_vocab(self, corpus):
        tokens = self.tokenize(corpus)
        for token in tokens:
            self.word_count[token] += 1
            self.words.add(token)
        self.word_index = {word: i for i, word in
enumerate(self.words)}

    def get_word_vector(self, word):
        return self.word_vectors[self.word_index[word]]

# Example usage
corpus = "Word embeddings are a type of word representation
that allows words to be represented as vectors in a continuous
vector space."
word2vec_model = Word2Vec(vector_size=50, window=2,
negative_samples=5, learning_rate=0.01)
word2vec_model.build_vocab(corpus)
word2vec_model.initialize_vectors()
word2vec_model.train(corpus, epochs=10)
# Get word vectors
vector_word = word2vec_model.get_word_vector('word')
print("Vector for 'word':", vector_word)
# Find similar words
word_similarity =
cosine_similarity(word2vec_model.word_vectors, [vector_word])
similar_words_indices = np.argsort(word_similarity[:, 0])[::-
1][1:6]
similar_words = [list(word2vec_model.words)[i] for i in
similar_words_indices]
print("Similar words to 'word':", similar_words)
```

OUTPUT:

```
Epoch 1, Loss: 3700.9866486345077
Epoch 2, Loss: 3713.6013179857114
Epoch 3, Loss: 3692.159427971019
Epoch 4, Loss: 3691.4201951495256
Epoch 5, Loss: 3667.456705268403
Epoch 6, Loss: 3707.7173999772717
Epoch 7, Loss: 3682.866294382986
Epoch 8, Loss: 3678.544178302956
Epoch 9, Loss: 3681.499736357797
Epoch 10, Loss: 3730.6180509786996
Vector for 'word': [0.58263744 0.35911002 0.34304845 0.83513312 0.27539995
0.20161799
 0.28650031 0.51133136 0.86728329 0.98584827 0.34853612 0.52269866
 0.23435707 0.50669535 0.64574134 0.89786769 0.44298268 0.26017309
 0.71280356 0.63997053 0.77700461 0.3917771  0.68813592 0.85579034
```

```
 0.4828248  0.67140497 0.53407789 0.87273619 0.47202019 0.37922163
 0.11173634 0.36148538 0.38163774 0.18540995 0.79988111 0.88903758
 0.0617453  0.66687891 0.02478728 0.25145586 0.14695297 0.42986099
 0.36846583 0.97134113 0.98747539 0.76553032 0.48468291 0.49752828
 0.66218462 0.00101609]
Similar words to 'word': ['vectors', 'of', 'to', 'in', 'vector']
```

RESULT:

Thus the program to implement the Word2Vec model was executed and verified successfully.

6.Text Classification by using Transformer

AIM:

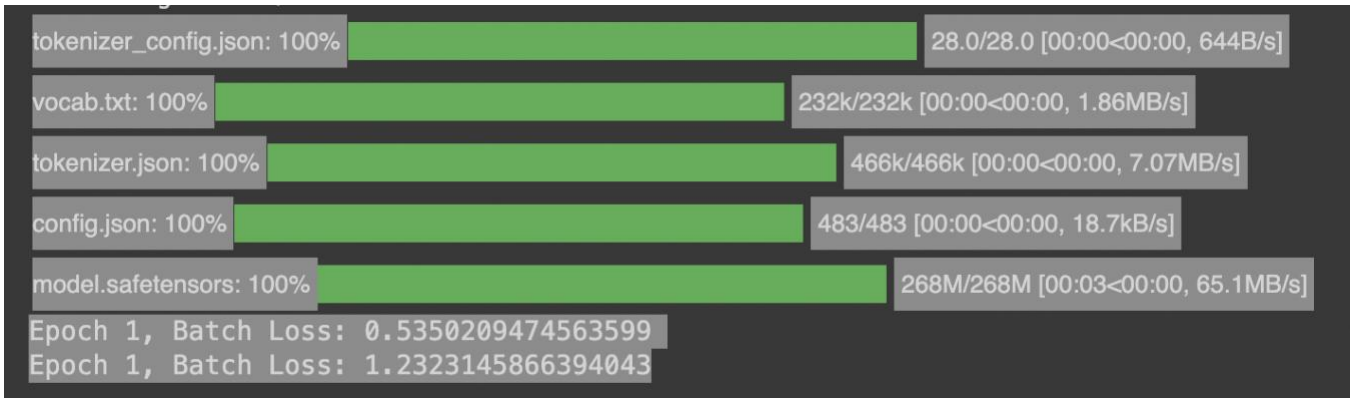
To implantation of Text Classification by using transformer.

SOURCE CODE:

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from transformers import DistilBertTokenizer, DistilBertModel

# Define the custom dataset class
class TextDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_length=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = torch.tensor(self.labels[idx])

        encoding = self.tokenizer(text, truncation=True,
padding='max_length', max_length=self.max_length, return_tensors='pt')

        return {
            'input_ids': encoding['input_ids'].squeeze(),
            'attention_mask': encoding['attention_mask'].squeeze(),
            'label': label
        }

# Define the transformer-based text classification model
class TextClassifier(nn.Module):
    def __init__(self, transformer_model, num_classes):
        super(TextClassifier, self).__init__()
        self.transformer = transformer_model
        self.classifier =
nn.Linear(self.transformer.config.hidden_size, num_classes)

    def forward(self, input_ids, attention_mask):
        outputs = self.transformer(input_ids=input_ids,
attention_mask=attention_mask)
        last_hidden_state = outputs.last_hidden_state[:, 0, :]  # Use
the [CLS] token representation
        logits = self.classifier(last_hidden_state)
        return logits

# Example usage
if __name__ == '__main__':
    # Sample data
```

```
    texts = ["This is a positive example.", "This is a negative

example.", "Another positive sentence."]
    labels = [1, 0, 1]  # 1 for positive, 0 for negative

    # Load pre-trained transformer model and tokenizer
    model_name = 'distilbert-base-uncased'
    tokenizer = DistilBertTokenizer.from_pretrained(model_name)
    transformer_model = DistilBertModel.from_pretrained(model_name)

    # Create the dataset and dataloader
    dataset = TextDataset(texts, labels, tokenizer)
    dataloader = DataLoader(dataset, batch_size=2, shuffle=True)

    # Instantiate the text classification model
    num_classes = 2  # Positive and negative classes
    text_classifier = TextClassifier(transformer_model, num_classes)

    # Training loop (for simplicity, just one epoch)
    criterion = nn.CrossEntropyLoss()
    optimizer  =  torch.optim.AdamW(text_classifier.parameters(),  lr=2e-
5)

    for epoch in range(1):
        for batch in dataloader:
            input_ids = batch['input_ids']
            attention_mask = batch['attention_mask']
            labels = batch['label']

            optimizer.zero_grad()
            outputs = text_classifier(input_ids, attention_mask)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()


            print(f"Epoch {epoch + 1}, Batch Loss: {loss.item()}")

 # Save or use the trained model for predictions
```

OUTPUT:



RESULT:

Thus the program to implantation of Text Classification by using transformer was executed and verified successfully.

7.Design a chatbot

AIM:

Design a chatbot with a simple dialog system

SOURCE CODE:

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

# Define a simple rule-based chatbot
def simple_chatbot(user_input):
    # Tokenize the user input
    tokens = word_tokenize(user_input.lower())

    # Simple rule-based responses
    if any(word in tokens for word in ['hello', 'hi',
'hey']):
        return "Hello! How can I help you today?"
    elif any(word in tokens for word in ['how', 'are',
'you']):
        return "I'm just a computer program, but thanks for
asking!"
    elif any(word in tokens for word in ['bye', 'goodbye']):
        return "Goodbye! Have a great day."
    else:
        return "I'm sorry, I don't understand. Can you
please rephrase?"

# Simple interactive chat with the user
print("Simple Chatbot: Hi there! Type 'bye' to exit.")
while True:
    user_input = input("You: ")
```

OUTPUT:

```
Simple Chatbot: Hi there! Type 'bye' to
exit.You: hi
Simple Chatbot: Hello! How can I help you
today?You: hello
Simple Chatbot: Hello! How can I help you
today?You: how r u
Simple Chatbot: I'm just a computer program, but thanks
for asking!You: bye
Simple Chatbot: Goodbye!
```

RESULT:

Thus the program to design a chatbot with a simple dialog system was executed and verified successfully.

8.Text to Speech Analysis

AIM:

To convert text to speech and find accuracy

SOURCE CODE:

```
import os
from gtts import gTTS
import speech_recognition as sr

def text_to_speech(text, language='en'):
    # Convert text to speech
    tts = gTTS(text=text, lang=language, slow=False)
    tts.save("output.mp3")

def speech_to_text():
    # Use the SpeechRecognition library to recognize speech from a
recorded audio file
    recognizer = sr.Recognizer()

    with sr.AudioFile("output.mp3") as source:
        audio_data = recognizer.record(source)
        try:
            # Use the Google Web Speech API for speech recognition
            text = recognizer.recognize_google(audio_data)
            return text
        except sr.UnknownValueError:
            return "Speech recognition could not understand audio."
        except sr.RequestError as e:
            return f"Could not request results from Google Speech
Recognition service; {e}"

def calculate_accuracy(original_text, recognized_text):
    # Simple accuracy calculation
    original_words = set(original_text.lower().split())
    recognized_words = set(recognized_text.lower().split())

    correct_words = original_words.intersection(recognized_words)
    accuracy = len(correct_words) / len(original_words) * 100

    return accuracy

if __name__ == "__main__":
    # Example text
    input_text = "Hello, how are you? This is a test of text-to-speech
and speech-to-text conversion."

    # Convert text to speech
    text_to_speech(input_text)
```

```
# Convert speech to text
recognized_text = speech_to_text()

# Print the recognized text
print("Recognized Text:", recognized_text)

# Calculate and print accuracy
accuracy = calculate_accuracy(input_text, recognized_text)
print("Accuracy:", accuracy, "%")
```

OUTPUT:

text = "Hello, this is a sample text to be converted to speech."

Convert text to speech and play the generated audio

text_to_speech(text)

RESULT:

Thus the program to Convert text to speech and find accuracy was executed and verified successfully.

9.Speech Recognition System

AIM:

To design a speech recognition system and find the error rate

SOURCE CODE:

```
import speech_recognition as sr

def recognize_speech(audio_file):
    recognizer = sr.Recognizer()

    with sr.AudioFile(audio_file) as source:
        audio_data = recognizer.record(source)

    try:
        # Using Google Web Speech API for speech recognition
        text = recognizer.recognize_google(audio_data)
        return text
    except sr.UnknownValueError:
        print("Speech Recognition could not understand the
audio")
        return None
    except sr.RequestError as e:
        print(f"Could not request results from Google Web Speech
API; {e}")
        return None

def calculate_error_rate(reference_text, recognized_text):
    reference_words = reference_text.split()
    recognized_words = recognized_text.split()

    total_words = len(reference_words)
    errors = sum(1 for ref, rec in zip(reference_words,
recognized_words) if ref != rec)

    error_rate = errors / total_words
    return error_rate

# Example usage
reference_text = "hello how are you"
audio_file_path = "C:/Users/malar/Desktop/audio file/voice.vlc"
# Replace with the path to your audio file

recognized_text = recognize_speech(audio_file_path)

if recognized_text is not None:
```

```
    print("Reference Text:", reference_text)
    print("Recognized Text:", recognized_text)

    error_rate = calculate_error_rate(reference_text,
recognized_text)
    print(f"Error Rate: {error_rate:.2%}")
```

OUTPUT:

Audio File: File Name.Wav

Error Rate:10.07

RESULT:

Thus the program to Design a speech recognition system and find the error rate was executed and verified successfully.