**Set**1

1.import re

def extract\_emails(text):

email\_pattern = r'[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'

emails = re.findall(email\_pattern, text)

return emails

text = "Please contact us at info@example.com, support@website.org, and admin123@service.net"

emails = extract\_emails(text)

print("Extracted Emails:", emails)

2. from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

texts = [

"Win a free iPhone now by clicking this link!",

"Your account has been updated successfully.",

"Congratulations! You've won a lottery.",

"Important update regarding your recent transaction.",

"Get rich fast with this easy method!"

]

labels = ['spam', 'ham', 'spam', 'ham', 'spam']

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(texts)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels, test\_size=0.3, random\_state=42)

model = MultinomialNB()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("Predictions:", y\_pred)

3. from sklearn.feature\_extraction.text import TfidfVectorizer

sentences = [

"Artificial intelligence (AI) is a field of computer science.",

"Machine learning is a subset of AI that focuses on training models to make predictions.",

"Deep learning is a type of machine learning that uses neural networks with multiple layers.",

"Neural networks are composed of interconnected nodes called neurons.",

"Recurrent neural networks (RNNs) are commonly used in natural language processing tasks."

]

vectorizer = TfidfVectorizer(stop\_words='english')

X = vectorizer.fit\_transform(sentences)

features = vectorizer.get\_feature\_names\_out()

for i, scores in enumerate(X.toarray()):

print(f"\nSentence {i+1}:")

for term, score in zip(features, scores):

if score > 0:

print(f"{term}: {score:.4f}")

4. import nltk

from nltk import word\_tokenize, pos\_tag, ne\_chunk

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('maxent\_ne\_chunker')

nltk.download('words')

def extract\_named\_entities(text):

tokens = word\_tokenize(text)

pos\_tags = pos\_tag(tokens)

return ne\_chunk(pos\_tags)

def reference\_resolution(text):

return text.replace("institution", "Harvard University")

text = "Harvard University, located in Cambridge, Massachusetts, is a prestigious institution."

print(extract\_named\_entities(text))

print("Resolved Text:", reference\_resolution(text))

set 2

1.def has\_equal\_O\_and\_I(s):

count\_O = 0

count\_I = 0

for char in s:

if char == 'O':

count\_O += 1

elif char == 'I':

count\_I += 1

return count\_O == count\_I

input\_string = "OIOI"

print("Equal number of 'O's and 'I's:", has\_equal\_O\_and\_I(input\_string))

2. import spacy

nlp = spacy.load("en\_core\_web\_sm")

def perform\_ner(text):

doc = nlp(text)

for ent in doc.ents:

print(ent.text, ent.label\_)

sentence = "Barack Obama was the 44th President of the United States, and he was born in Honolulu, Hawaii"

perform\_ner(sentence)

3. import wikipedia

user\_agent = "YourAppName/1.0 (your.email@example.com)"

wikipedia.set\_user\_agent(user\_agent)

def resolve\_entities(text)

words = text.split()

for word in words:

try:

page = wikipedia.page(word)

print(f"Entity: {word} - Wikipedia Page: {page.title}")

except wikipedia.exceptions.DisambiguationError as e:

print(f"Entity: {word} - Multiple meanings: {e.options}")

except wikipedia.exceptions.PageError:

print(f"Entity: {word} - No Wikipedia page found")

sentences = [

"Apple is a leading tech company.",

"I love apples as a fruit.",

"Python is a popular programming language.",

"The python is a non-venomous snake."

]

for sentence in sentences:

resolve\_entities(sentence)

4. def evaluate\_coherence(text):

sentences = text.split('. ')

coherence = True

if len(sentences) < 2:

return False

for i in range(len(sentences) - 1):

if sentences[i].strip() == "" or sentences[i+1].strip() == "":

coherence = False

return coherence

sample\_text = """Once upon a time, there was a young boy named Peter.

He lived in a small village.

One day, he decided to explore the nearby forest."""

print("Text is coherent:", evaluate\_coherence(sample\_text))

set 3

1. import nltk

nltk.download('punkt')

nltk.download('average\_perceptron\_tagger')

text="The cats are playing in the garden."

tokens=nltk.word\_tokenize(text)

tagged=nltk.pos\_tag(tokens)

print(tagged)

2. from textblob import TextBlob

sentences = ["I love this product! It's amazing.", "The weather is terrible today."]

for sentence in sentences:

blob = TextBlob(sentence)

sentiment = blob.sentiment.polarity

if sentiment > 0:

print(f"'{sentence}' - Positive")

elif sentiment < 0:

print(f"'{sentence}' - Negative")

else:

print(f"'{sentence}' - Neutral")

3. from nltk.corpus import wordnet

word = 'bank'

synsets = wordnet.synsets(word)

for syn in synsets:

print(syn.name(), syn.definition())

4. import nltk

nltk.download('punkt')

conversation = [

"Good morning! How's the weather today?",

"I heard it's going to be sunny and warm.",

"Could you please send me the report by 3 PM?",

"Of course, I'll send it over before the deadline.",

"Do you know where the nearest post office is?",

"The post office is two blocks down the street."

]

for sentence in conversation:

if sentence.endswith('?'):

print(f"'{sentence}' - Question")

elif 'please' in sentence.lower():

print(f"'{sentence}' - Request")

else:

print(f"'{sentence}' - Statement")

set 4

1. import re

def to\_past\_tense(verb):

if verb.endswith('e'):

return verb + 'd'

elif verb.endswith('y'):

return verb[:-1] + 'ied'

elif re.search(r'[aiou]$', verb):

return verb + 'ed'

else:

return verb + 'ed'

def parse\_sentences(sentences):

for sentence in sentences:

words = sentence.split()

past\_tense\_verbs = [to\_past\_tense(word) for word in words if word.lower() in ['walk', 'jump', 'run', 'play']]

print(f"Original Sentence: {sentence}")

print(f"Past Tense Conversion: {' '.join(past\_tense\_verbs)}\n")

sentences = [

"She walked to the park yesterday",

"He jumped over the fence"

]

parse\_sentences(sentences)

2. from sumy.parsers.plaintext import PlaintextParser

from sumy.nlp.tokenizers import Tokenizer

from sumy.summarizers.lsa import LsaSummarizer

document = """

Natural language processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and humans through natural language.

NLP technologies are used to process, analyze, and understand large amounts of natural language data.

One of the primary applications of NLP is sentiment analysis, which determines the sentiment or emotional tone of a piece of text. Sentiment analysis is widely used in social media monitoring, customer feedback analysis, and brand reputation management.

Text summarization is another important NLP task. Extractive summarization involves selecting a subset of sentences from a text to create a shorter version that retains the most critical information. Abstractive summarization, on the other hand, generates a summary by paraphrasing and rephrasing the original text.

"""

parser = PlaintextParser.from\_string(document, Tokenizer("english"))

summarizer = LsaSummarizer()

summary = summarizer(parser.document, 3)

print("Extractive Summary:")

for sentence in summary:

print(sentence)

3. from sympy import symbols, And, Or, Not

x, y = symbols('x y')

expressions = [

And(x, y),

Or(x, Not(y)),

And(x, Or(y, Not(x)))

]

def evaluate\_expression(expr, values):

return expr.subs(values)

values = {x: True, y: False}

for expr in expressions:

print(f"Expression: {expr}")

print(f"Evaluation with {values}: {evaluate\_expression(expr, values)}\n")

4. from transformers import pipeline

generator = pipeline("text-generation", model="gpt2")

def generate\_text(prompt):

result = generator(prompt, max\_length=10000, num\_return\_sequences=1)

return result[0]["generated\_text"]

prompt = input("Enter the question: ")

response=generate\_text(prompt)

print(response)

set 6

1. import nltk

nltk.download('punkt');

nltk.download('averaged\_perceptron\_tagger')

tokens = nltk.word\_tokenize("This is an example.")

tags = nltk.pos\_tag(tokens)

print(tags)

2. from textblob import TextBlob

text = "Apple is looking at buying U.K. startup for $1 billion."

blob = TextBlob(text)

entities = blob.noun\_phrases

print("Named Entities:")

for entity in entities:

print(entity)

3. from sklearn.feature\_extraction.text import TfidfVectorizer

docs = ["The cat sat on the mat.", "The dog sat on the log.", "Cats and dogs are pets."]

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform(docs)

print(tfidf\_matrix.toarray())

4. import re

pattern=r'\b[a-zA-Z]+\b'

text="Hello, World!"

matches=re.findall(pattern,text)

print(matches)

set 7

1. import nltk

nltk.download('punkt')

nltk.download('average\_perceptron\_tagger')

text="The cats are playing in the garden."

tokens=nltk.word\_tokenize(text)

tagged=nltk.pos\_tag(tokens)

print(tagged)

2. from transformers import T5ForConditionalGeneration, T5Tokenizer

model = T5ForConditionalGeneration.from\_pretrained('t5-small')

tokenizer = T5Tokenizer.from\_pretrained('t5-small')

text = """

The World Health Organization (WHO) plays a vital role in global health.

WHO is headquartered in Geneva, Switzerland, and it is responsible for coordinating international efforts

to control and prevent the spread of diseases. Its mission is to promote and protect the health of people worldwide.

"""

input\_text = "summarize: " + text

input\_ids = tokenizer.encode(input\_text, return\_tensors="pt", max\_length=512, truncation=True)

summary\_ids = model.generate(input\_ids, num\_beams=4, min\_length=30, max\_length=100, length\_penalty=2.0, early\_stopping=True)

summary = tokenizer.decode(summary\_ids[0], skip\_special\_tokens=True)

print("Summary:")

print(summary)

3. import spacy

nlp = spacy.load('en\_core\_web\_sm')

sentences = [

"The quick brown fox jumps over the lazy dog.",

"She is an excellent chef and loves to cook delicious meals.",

"The Eiffel Tower in Paris is a famous landmark."

]

for sentence in sentences:

doc = nlp(sentence)

print(f"\nSentence: {sentence}")

print("Noun Phrases:")

for np in doc.noun\_chunks:

print(f" - {np.text} (head: {np.root.text})")

print("Recognized Entities (with meaning):")

for ent in doc.ents:

print(f" - {ent.text}: {ent.label\_} ({spacy.explain(ent.label\_)})")

4. import nltk

from nltk.tokenize import word\_tokenize

from nltk.stem import WordNetLemmatizer

from nltk.corpus import wordnet

nltk.download('punkt')

nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

def get\_wordnet\_pos(treebank\_tag):

if treebank\_tag.startswith('J'):

return wordnet.ADJ

elif treebank\_tag.startswith('V'):

return wordnet.VERB

elif treebank\_tag.startswith('N'):

return wordnet.NOUN

elif treebank\_tag.startswith('R'):

return wordnet.ADV

else:

return wordnet.NOUN

sentences = [

"The quick brown foxes jumped over the lazy dogs."

]

for sentence in sentences:

print(f"\nSentence: {sentence}")

tokens = word\_tokenize(sentence)

tagged\_tokens = nltk.pos\_tag(tokens)

print("Lemmatized Words:")

for word, tag in tagged\_tokens:

wn\_tag = get\_wordnet\_pos(tag)

lemmatized\_word = lemmatizer.lemmatize(word, pos=wn\_tag)

print(f" - {word} (POS: {tag}) -> {lemmatized\_word}")

set 8

1. import nltk

patterns = [(r'.\*ing$', 'VBG'), (r'.\*ed$', 'VBD')]

tagger = nltk.RegexpTagger(patterns)

tokens = nltk.word\_tokenize("He is swimming.")

print(tagger.tag(tokens))

2. from textblob import TextBlob

text = "Apple is looking at buying U.K. startup for $1 billion."

blob = TextBlob(text)

entities = blob.noun\_phrases

print("Named Entities:")

for entity in entities:

print(entity)

3. import nltk

grammar = nltk.PCFG.fromstring("""

S -> NP VP [1.0]

NP -> Det N [0.5] | NP PP [0.4] | 'the' [0.1]

VP -> V NP [0.7] | VP PP [0.3]

PP -> P NP [1.0]

Det -> 'the' [0.7] | 'a' [0.3]

N -> 'fox' [0.4] | 'dog' [0.3] | 'cat' [0.2] | 'bird' [0.1]

V -> 'jumps' [0.5] | 'runs' [0.3] | 'sits' [0.2]

P -> 'over' [0.6] | 'on' [0.4]

""")

parser = nltk.ViterbiParser(grammar)

sentence = ['the', 'fox', 'jumps', 'over', 'the', 'dog']

for tree in parser.parse(sentence):

print(tree)

4. from nltk.stem import PorterStemmer

from nltk.tokenize import word\_tokenize

sentences = ["Coding with Python is very enjoyable.", "I had a delicious meal at the restaurant."]

ps = PorterStemmer()

for sentence in sentences:

words = word\_tokenize(sentence)

stemmed\_words = [ps.stem(word) for word in words]

print(stemmed\_words)

set 9

1. import nltk

from nltk.parse import ViterbiParser

pcfg\_grammar = nltk.PCFG.fromstring("""

S -> NP VP [1.0]

NP -> Det N [0.5]

NP -> NP PP [0.5]

VP -> V NP [0.6]

VP -> VP PP [0.4]

PP -> P NP [1.0]

Det -> 'The' [0.5] | 'the' [0.5]

N -> 'cat' [0.5] | 'mouse' [0.5]

V -> 'chased' [1.0]

P -> 'on' [0.4] | 'with' [0.6]

""")

parser = ViterbiParser(pcfg\_grammar)

sentence = "The cat chased the mouse".split()

for tree in parser.parse(sentence):

print(tree)

tree.pretty\_print()

2. import nltk

from nltk.sentiment import SentimentIntensityAnalyzer

nltk.download('vader\_lexicon')

sia = SentimentIntensityAnalyzer()

text = "I love this product! It's amazing."

sentiment\_scores = sia.polarity\_scores(text)

compound\_score = sentiment\_scores['compound']

if compound\_score >= 0.05:

sentiment = "Positive"

elif compound\_score <= -0.05:

sentiment = "Negative"

else:

sentiment = "Neutral"

print(f"Sentiment scores: {sentiment\_scores}")

print(f"Overall sentiment: {sentiment}")

3. def classify\_dialog\_act(sentence):

sentence = sentence.lower()

if any(greeting in sentence for greeting in ["hello", "hi", "hey"]):

return "Greeting"

elif sentence.endswith("?"):

return "Question"

elif "please" in sentence:

return "Request"

elif any(ack in sentence for ack in ["thank you", "thanks"]):

return "Acknowledgment"

elif any(affirm in sentence for affirm in ["sure", "okay", "here you go"]):

return "Affirmative"

else:

return "Statement"

dialog = [

"Hello! How are you today?",

"I'm doing well, thank you. How about you?",

"Can you please pass the salt?",

"Sure, here you go.",

"What time is the meeting tomorrow?",

"The meeting is at 2:00 PM."

]

for sentence in dialog:

dialog\_act = classify\_dialog\_act(sentence)

print(f"'{sentence}' -> {dialog\_act}")

4. import re

def is\_valid\_date(date\_str):

pattern = r'^\d{2}/\d{2}/\d{4}$'

return bool(re.match(pattern, date\_str))

test\_dates = [

"12/08/2023",

"12-08-2023"]

for date in test\_dates:

print(f"'{date}' -> {'Valid' if is\_valid\_date(date) else 'Invalid'}")

set 10

1. from sklearn.feature\_extraction.text import TfidfVectorizer

documents = [

"Natural language processing (NLP) is a field of study in artificial intelligence.",

"NLP techniques are used in various applications like machine translation and sentiment analysis.",

"The development of NLP tools and libraries has made text analysis easier."

]

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform(documents)

feature\_names = vectorizer.get\_feature\_names\_out()

for i, doc in enumerate(tfidf\_matrix):

print(f"Document {i+1} TF-IDF Scores:")

for word, score in zip(feature\_names, doc.toarray()[0]):

print(f" {word}: {score:.4f}")

print()

2. import spacy

nlp = spacy.load("en\_core\_web\_sm")

sentences = [

"John and Mary went to the store.",

"The big brown dog chased the small black cat."

]

for sentence in sentences:

doc = nlp(sentence)

print(f"Dependency Parsing for: {sentence}")

for token in doc:

print(f"{token.text}: {token.dep\_} -> {token.head.text}")

print()

3. from transformers import MarianMTModel, MarianTokenizer

model\_name = 'Helsinki-NLP/opus-mt-en-de'

tokenizer = MarianTokenizer.from\_pretrained(model\_name)

model = MarianMTModel.from\_pretrained(model\_name)

sentence = "Natural language processing is an important field of artificial intelligence."

input\_ids = tokenizer.encode(sentence, return\_tensors="pt")

output = model.generate(input\_ids)

translated\_sentence = tokenizer.decode(output[0], skip\_special\_tokens=True)

print(f"Translated Sentence: {translated\_sentence}")

4. variables = {'p': True, 'q': True, 'r': False}

expressions = ["p and r", "p or r", "not p", "q and (r or p)"]

for expr in expressions:

try:

result = eval(expr, {}, variables)

print(f"Expression: {expr} => {result}")

except Exception as e:

print(f"Error evaluating expression {expr}: {e}")