TS1

**re.search(r'at',text)**

**re.findall("[A-Z]\w+",text)**

**re.findall("[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}",text)**

**re.findall(r'\+?\d{1,2}[-\s]+?\(?\d{1,3}\)?[-.\s]?\d{1,4}[-\s]?\d{1,4}[-.\s]?\d{1,4}', text)**

**re.findall(r'\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b',text)**

**re.findall(r'\b(?:\d{2}/\d{2}/\d{4}|\d{4}/\d{2}/\d{2})\b',text)**

**re.findall(r'https?://(?:www\.)?[a-zA-Z0-9./-]+',text)**

**re.findall(r'#[\w-]+',text)**

**re.findall(r'@[\w-]+',text)**

**re.findall(r'\b[A-Z]{2,}\b',text)**

**re.findall(r'\b-?\d+(\.\d+)?\b',text)**

**re.findall(r'\$+?\d+(?:,\d{3})\*(?:\.\d{2})?',text)**

**re.findall(r'\b[Aa]pt #\d+\b',text)**

**re.findall(r'([A-Z][^.!?]\*[.!?])',text)**

**re.findall(r'\b\d+\.\d+e[-+]?\d+\b',text)**

TS2

import nltk

import spacy

from nltk.tokenize import word\_tokenize

from nltk.tokenize import sent\_tokenize

from nltk.stem import WordNetLemmatizer

from nltk import pos\_tag, ne\_chunk

from nltk.collocations import BigramCollocationFinder

from nltk.metrics import BigramAssocMeasures

nltk.download('punkt')

nltk.download('wordnet')

nltk.download('omw-1.4')

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

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

nltk.download('words')

nltk.download('punkt\_tab')

tokens = word\_tokenize(corpus)

for i in tokens:

print(i)

def word\_search(tokens, search\_term):

indices = [i for i, token in enumerate(tokens) if token.lower() == search\_term.lower()]

return indices

search\_word = "formula"

result\_indices = word\_search(tokens, search\_word)

print(result\_indices)

unique\_tokens=set(tokens)

print(len(unique\_tokens))

from nltk.probability import FreqDist

import matplotlib.pyplot as plt

fdist = FreqDist(tokens)

fdist.plot(30,cumulative=False)

plt.show()

nltk.download('stopwords')

from nltk.corpus import stopwords

stop\_words = set(stopwords.words('english'))

filtered\_tokens = [w for w in tokens if not w.lower() in stop\_words and w.isalnum()]

fdist = FreqDist(filtered\_tokens)

fdist.plot(30, cumulative=False)

plt.show()

lemmatizer = WordNetLemmatizer()

lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in filtered\_tokens]

print(lemmatized\_tokens)

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

first\_five\_tokens = tokens[:5]

print(first\_five\_tokens)

pos\_tags = pos\_tag(first\_five\_tokens)

print("Part-of-speech tags:", pos\_tags)

finder = BigramCollocationFinder.from\_words(lemmatized\_tokens)

finder.apply\_freq\_filter(2)

bigram\_measures = BigramAssocMeasures()

collocations = finder.nbest(bigram\_measures.pmi, 10)

print("Top 10 Bigram Collocations (PMI):", collocations)

TS3

import nltk

from nltk.corpus import \*

from nltk.tokenize import word\_tokenize, sent\_tokenize

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk.probability import FreqDist

from nltk import pos\_tag, ne\_chunk

from nltk.classify import NaiveBayesClassifier

from nltk.classify.util import accuracy

nltk.download('reuters')

TS4

import re

import pandas as pd

from collections import Counter

from nltk.corpus import stopwords

import matplotlib.pyplot as plt

import seaborn as sns

import zipfile

import os

def unzip\_file(zip\_path, extract\_to):

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

zip\_path = "/content/drive/MyDrive/twitter\_sentiment.zip"

extract\_to = "/content/twitter"

unzip\_file(zip\_path, extract\_to)

import nltk

nltk.download('stopwords')

def plot\_top\_words(word\_counts):

words, counts = zip(\*word\_counts)

plt.figure(figsize=(12,6))

sns.barplot(y=list(counts), x=list(words), palette='viridis')

plt.xlabel("Words")

plt.ylabel("Frequency")

plt.xticks(rotation=90)

plt.title("Top 50 Most Frequent Words")

plt.show()

def clean\_text(text):

text = re.sub(r'[^a-zA-Z\s]', '', text)

text = text.lower()

words = text.split()

return words

def get\_top\_words(df, column, n=50):

stop\_words = set(stopwords.words('english'))

word\_counts = Counter()

for tweet in df[column].dropna():

cleaned\_words = clean\_text(tweet)

filtered\_words = [word for word in cleaned\_words if word not in stop\_words]

word\_counts.update(filtered\_words)

return word\_counts.most\_common(n)

top\_words = get\_top\_words(df1, 'Text')

print(top\_words)

plot\_top\_words(top\_words)

TS5

!pip install --upgrade --force-reinstall numpy gensim

import pandas as pd

import zipfile

from gensim.models import Word2Vec

from nltk.tokenize import word\_tokenize

import nltk

nltk.download('punkt')

nltk.download('punkt\_tab')

def unzip\_file(zip\_path, extract\_to):

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

zip\_path = "/content/drive/MyDrive/archive (5).zip"

extract\_to = "/content/IMDB"

unzip\_file(zip\_path, extract\_to)

df=pd.read\_csv('/content/IMDB/IMDB Dataset.csv')

df.head()

df['tokenized\_reviews'] = df['review'].astype(str).apply(word\_tokenize)

df['tokenized\_reviews'].head()

model = Word2Vec(sentences=df['tokenized\_reviews'], vector\_size=100, window=5, min\_count=2, workers=4)

word = "good"

similar\_words = model.wv.most\_similar(word, topn=10)

print("Similar words for good are:")

for word, similarity in similar\_words:

print(f"{word}: {similarity}")

TS 6

!pip install -U transformers

import zipfile

import os

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

from sklearn.preprocessing import LabelEncoder

from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, Seq2SeqTrainer, Seq2SeqTrainingArguments, DataCollatorForSeq2Seq

from datasets import Dataset

import pandas as pd

zip\_path = "/content/drive/MyDrive/archive (5).zip"

extract\_to = "/content/ecommerce"

os.makedirs(extract\_to, exist\_ok=True)

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

zip\_ref.extractall(extract\_to)

print("Files unzipped to:", extract\_to)

!pip install -q transformers datasets accelerate

df=pd.read\_csv('/content/ecommerce/ecommerceDataset.csv')

df.info()

df = df.rename(columns={

df.columns[0]: "label",

df.columns[1]: "text"

})

df = df[['text', 'label']]

df.head()

le = LabelEncoder()

df['label\_encoded'] = le.fit\_transform(df['label'])

train\_texts, val\_texts, train\_labels, val\_labels = train\_test\_split(

df['text'].tolist(),

df['label\_encoded'].tolist(),

test\_size=0.2,

random\_state=42

)

label\_names = le.classes\_.tolist()

train\_data = [{"input": f"Classify this product: {t}", "output": label\_names[l]}

for t, l in zip(train\_texts, train\_labels)]

val\_data = [{"input": f"Classify this product: {t}", "output": label\_names[l]}

for t, l in zip(val\_texts, val\_labels)]

model\_name = "google/flan-t5-small"

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

model = AutoModelForSeq2SeqLM.from\_pretrained(model\_name)

def tokenize\_fn(examples):

model\_inputs = tokenizer(examples["input"], truncation=True, padding="max\_length", max\_length=256)

with tokenizer.as\_target\_tokenizer():

labels = tokenizer(examples["output"], truncation=True, padding="max\_length", max\_length=10)

model\_inputs["labels"] = labels["input\_ids"]

return model\_inputs

train\_ds = Dataset.from\_list(train\_data).map(tokenize\_fn, batched=True)

val\_ds = Dataset.from\_list(val\_data).map(tokenize\_fn, batched=True)

training\_args = Seq2SeqTrainingArguments(

output\_dir="./results",

eval\_strategy="epoch",

learning\_rate=2e-4,

per\_device\_train\_batch\_size=8,

per\_device\_eval\_batch\_size=8,

num\_train\_epochs=1,

weight\_decay=0.01,

logging\_dir='./logs',

save\_total\_limit=1

)

trainer = Seq2SeqTrainer(

model=model,

args=training\_args,

train\_dataset=train\_ds,

eval\_dataset=val\_ds,

tokenizer=tokenizer,

data\_collator=DataCollatorForSeq2Seq(tokenizer, model=model)

)

trainer.train()

def predict(text):

input\_text = f"Classify this product: {text}"

inputs = tokenizer(input\_text, return\_tensors="pt", truncation=True).to(model.device)

outputs = model.generate(inputs["input\_ids"], max\_length=10)

return tokenizer.decode(outputs[0], skip\_special\_tokens=True)

print(predict("Beautiful wall hanging with floral patterns and wooden frame"))

TS 8

import asyncio

from googletrans import Translator

from gtts import gTTS

import speech\_recognition as sr

from jiwer import wer

import os

# Step 1: Translate text from English to Tamil

translator = Translator()

text\_in\_english = "Hello, I am Aswath"

translated\_text = translator.translate(text\_in\_english, src='en', dest='ta')

# Step 2: Convert translated Tamil text to speech

tamil\_text = translated\_text.text

print("Translated Text (Tamil):", tamil\_text)

tts = gTTS(text=tamil\_text, lang='ta')

tts.save("tamil\_output.mp3")

os.system("start tamil\_output.mp3") # Use "xdg-open" on Linux, "open" on macOS

# Step 3: Convert the speech back to text for accuracy evaluation

recognizer = sr.Recognizer()

# Convert MP3 to WAV

os.system("ffmpeg -i tamil\_output.mp3 tamil\_output.wav -y") # Requires ffmpeg

try:

with sr.AudioFile("tamil\_output.wav") as source:

audio = recognizer.record(source)

recognized\_text = recognizer.recognize\_google(audio, language='ta')

print("Recognized Text from Speech (Tamil):", recognized\_text)

# Step 4: Accuracy Calculation using Word Error Rate (WER)

error\_rate = wer(tamil\_text, recognized\_text)

accuracy = (1 - error\_rate) \* 100

print(f"Accuracy: {accuracy:.2f}%")

except sr.RequestError:

print("Could not request results from the speech recognition service.")

except sr.UnknownValueError:

print("Speech was not recognized.")

TS 9

# exp no 9 Design a speech recognition system and find the error rate

!pip install gtts

!pip install SpeechRecognition

!pip install googletrans==4.0.0-rc1

!pip install jiwer

import speech\_recognition as sr

from jiwer import wer

recognizer = sr.Recognizer()

reference\_text = "the stale smell of old beer lingers it takes heat to bring a cold dip restores health and zest a salt pickle taste fine with ham tacos al pastor are my favorite a zestful food is the hot cross bun"

audio\_file = "/content/harvard.wav" # Ensure this file exists in the same directory or give full path

try:

with sr.AudioFile(audio\_file) as source:

audio = recognizer.record(source)

recognized\_text = recognizer.recognize\_google(audio)

print("Recognized Text:", recognized\_text)

error = wer(reference\_text, recognized\_text)

accuracy = (1 - error) \* 100

print(f"Word Error Rate: {error:.2f}")

print(f"Accuracy: {accuracy:.2f}%")

except sr.UnknownValueError:

print("Speech could not be understood.")

except sr.RequestError as e:

print(f"Could not request results; {e}")