1a)

import re

text = "Follow us at https://twitter.com/elonmusk and https://twitter.com/nasa."

handles = re.findall(r"https://twitter\.com/([A-Za-z0-9\_]+)", text)

print("Twitter Handles:", handles)

1b)

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer

text = "This is an example text for preprocessing."

# Tokenization

tokens = word\_tokenize(text)

# Stop Word Removal

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

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

# Stemming

stemmer = PorterStemmer()

stemmed\_tokens = [stemmer.stem(word) for word in filtered\_tokens]

print("Processed Text:", stemmed\_tokens)

2a)

import nltk

from nltk import word\_tokenize

import matplotlib.pyplot as plt

text=""" There is a man on the hill, and I watched him with my telescope. There is a man on the hill, and

he has a telescope. I’m on a hill, and I saw a man using my telescope."""

words=word\_tokenize(text)

#Stop Words

from nltk.corpus import stopwords

Stopwords=stopwords.words("english")

#removal of punctuation

word\_no\_pun=[]

for w in words:

if w.isalpha():

word\_no\_pun.append(w.lower())

print(word\_no\_pun)

print(len(word\_no\_pun))

#removal of stop words

clean\_words=[]

for w in word\_no\_pun:

if w not in Stopwords:

clean\_words.append(w)

print(clean\_words)

print(len(clean\_words))

fdist=nltk.FreqDist(clean\_words)

fdist.most\_common(10)

fdist.plot(10)

2b)

import re

from nltk.tokenize import word\_tokenize

text = "This is a simple example text."

# Using word\_tokenize

nltk\_split = word\_tokenize(text)

# Using split()

python\_split = text.split()

# Using regex

regex\_split = re.split(r'\s+', text)

print("NLTK:", nltk\_split)

print("Python Split:", python\_split)

print("Regex Split:", regex\_split)

3a)

import nltk

from nltk import pos\_tag, word\_tokenize

from nltk.tree import Tree

# Download necessary NLTK data files (only needs to be done once)

nltk.download('punkt')

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

# Input text

text = "The quick brown fox jumps over the lazy dog."

# Tokenize the text

tokens = word\_tokenize(text)

# Generate part-of-speech tags

pos\_tags = pos\_tag(tokens)

print(pos\_tags)

# Print the POS tags

# Convert POS tags into a tree structure for visualization

tree = Tree('S',pos\_tags)

# Draw the tree

tree.draw()

3b)

import re

text='''

Born Elon Reeve Musk

June 28, 1971 (age 50)

Pretoria, Transvaal, South Africa

Citizenship

South Africa (1971–present)

Canada (1971–present)

United States (2002–present)

Education University of Pennsylvania (BS, BA)

Title

Founder, CEO and Chief Engineer of SpaceX

CEO and product architect of Tesla, Inc.

Founder of The Boring Company and X.com (now part of PayPal)

Co-founder of Neuralink, OpenAI, and Zip2

Spouse(s)

Justine Wilson

(m. 2000; div. 2008)

Talulah Riley

(m. 2010; div. 2012)

(m. 2013; div. 2016)

'''

def get\_pattern\_match(pattern, text):

matches = re.findall(pattern, text)

if matches:

return matches[0]

def extract\_personal\_information(text):

age = get\_pattern\_match('age (\d+)', text)

full\_name = get\_pattern\_match('Born(.\*)\n', text)

birth\_date = get\_pattern\_match('Born.\*\n(.\*)\(age', text)

birth\_place = get\_pattern\_match('\(age.\*\n(.\*)', text)

return {

'age': int(age),

'name': full\_name.strip(),

'birth\_date': birth\_date.strip(),

'birth\_place': birth\_place.strip()

}

extract\_personal\_information(text)

4a)

from nltk.util import ngrams

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

text = "Artificial intelligence has made significant advancements..."

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

tokens = [word for word in word\_tokenize(text) if word.lower() not in stop\_words]

bigrams = list(ngrams(tokens, 2))

print("Bigrams:", bigrams)

4b)

import nltk

from nltk import word\_tokenize

import matplotlib.pyplot as plt

text=""" There is a man on the hill, and I watched him with my telescope. There is a man on the hill, and

he has a telescope. I’m on a hill, and I saw a man using my telescope."""

words=word\_tokenize(text)

#Stop Words

from nltk.corpus import stopwords

Stopwords=stopwords.words("english")

#removal of punctuation

word\_no\_pun=[]

for w in words:

if w.isalpha():

word\_no\_pun.append(w.lower())

print(word\_no\_pun)

print(len(word\_no\_pun))

#removal of stop words

clean\_words=[]

for w in word\_no\_pun:

if w not in Stopwords:

clean\_words.append(w)

print(clean\_words)

print(len(clean\_words))

fdist=nltk.FreqDist(clean\_words)

fdist.most\_common(10)

fdist.plot(10)

5a)

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import wordnet

from nltk.stem import WordNetLemmatizer

def morphological\_analysis(text):

words = word\_tokenize(text)

pos\_tags = nltk.pos\_tag(words)

lemmatizer = WordNetLemmatizer()

lemmatized\_words = []

for word, pos in pos\_tags:

pos\_category = pos[0].lower()

pos\_category = pos\_category if pos\_category in ['a', 'n', 'v', 'r'] else None

if pos\_category:

lemma = lemmatizer.lemmatize(word, pos=pos\_category)

else:

lemma = lemmatizer.lemmatize(word)

lemmatized\_words.append(lemma)

return lemmatized\_words

text = '''I saw birds and geese playing with boxes in that room. In Front of that room there is

an garden with full of flowers on the plant made artificially'''

result = morphological\_analysis(text)

print(result)

5b)

import nltk

from nltk import pos\_tag, word\_tokenize

from nltk.tree import Tree

# Download necessary NLTK data files (only needs to be done once)

nltk.download('punkt')

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

# Input text

text = "The quick brown fox jumps over the lazy dog."

# Tokenize the text

tokens = word\_tokenize(text)

# Generate part-of-speech tags

pos\_tags = pos\_tag(tokens)

print(pos\_tags)

# Print the POS tags

# Convert POS tags into a tree structure for visualization

tree = Tree('S',pos\_tags)

# Draw the tree

tree.draw()

6a)

import nltk

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer, WordNetLemmatizer

# define an method

def morphological\_analysis(sentence):

words = word\_tokenize(sentence)

# Porter Stemmer

porter\_stemmer = PorterStemmer()

porter\_stems = [porter\_stemmer.stem(word) for word in words]

# Lancaster Stemmer

lancaster\_stemmer = LancasterStemmer()

lancaster\_stems = [lancaster\_stemmer.stem(word) for word in words]

# Snowball Stemmer

snowball\_stemmer = SnowballStemmer('english')

snowball\_stems = [snowball\_stemmer.stem(word) for word in words]

# WordNet Lemmatizer

lemmatizer = WordNetLemmatizer()

lemmas = [lemmatizer.lemmatize(word) for word in words]

print("Original Sentence:", sentence)

print("\nPorter Stems:", porter\_stems)

print("Lancaster Stems:", lancaster\_stems)

print("Snowball Stems:", snowball\_stems)

print("WordNet Lemmas:", lemmas)

# Example usage

sentence = "Morphological analysis is important for natural language processing."

morphological\_analysis(sentence)

6b)

import spacy

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

text = "The quick brown fox jumps over the lazy dog."

doc = nlp(text)

for token in doc:

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

7)

from collections import Counter

from nltk import word\_tokenize

from nltk.util import ngrams

from nltk.stem import WordNetLemmatizer

from nltk.corpus import stopwords

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

doc\_1 = '''Convolutional Neural Networks are very similar to ordinary Neural Networks from

the previous chapter'''

doc\_2 = '''Convolutional Neural Networks take advantage of the fact that the input consists of

images and they constrain the architecture in a more sensible way.'''

doc\_3 = '''In particular, unlike a regular Neural Network, the layers of a ConvNet have

neurons arranged in 3 dimensions: width, height, depth.'''

docs = [doc\_1, doc\_2, doc\_3]

docs = (' '.join(docs)).lower()

tokens = word\_tokenize(docs)

tokens = [t for t in tokens if t not in stop\_words]

word\_l = WordNetLemmatizer()

tokens = [word\_l.lemmatize(t) for t in tokens if t.isalpha()]

bi\_grams = list(ngrams(tokens, 2))

counter = Counter(bi\_grams)

counter.most\_common(5)

8)

from collections import defaultdict

from nltk.util import ngrams

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

from collections import Counter

corpus = ["I love NLP", "I love Python"]

tokens = [word for sentence in corpus for word in word\_tokenize(sentence)]

bigrams = list(ngrams(tokens, 2))

bigram\_freq = Counter(bigrams)

vocab\_size = len(set(tokens))

smoothed\_prob = {bigram: (freq + 1) / (len(tokens) + vocab\_size) for bigram, freq in bigram\_freq.items()}

print("Bigram Probabilities:", smoothed\_prob)

9)

non\_terminals = ["NP", "Nom", "Det", "AP",

"Adv", "A"]

terminals = ["book", "orange", "man",

"tall", "heavy",

"very", "muscular"]

R = {

"NP": [["Det", "Nom"]],

"Nom": [["AP", "Nom"], ["book"],

["orange"], ["man"]],

"AP": [["Adv", "A"], ["heavy"],

["orange"], ["tall"]],

"Det": [["a"]],

"Adv": [["very"], ["extremely"]],

"A": [["heavy"], ["orange"], ["tall"],

["muscular"]]

}

def cykParse(w):

n = len(w)

T = [[set([]) for j in range(n)] for i in range(n)]

for j in range(0, n):

for lhs, rule in R.items():

for rhs in rule:

if len(rhs) == 1 and \

rhs[0] == w[j]:

T[j][j].add(lhs)

for i in range(j, -1, -1):

for k in range(i, j + 1):

for lhs, rule in R.items():

for rhs in rule:

if len(rhs) == 2 and rhs[0] in T[i][k] and rhs[1] in T[k + 1][j]:

T[i][j].add(lhs)

if len(T[0][n-1]) != 0:

print("True")

else:

print("False")

w = "a very heavy orange book".split()

cykParse(w)

10)

def min\_edit\_dist(source, target):

    m, n = len(source), len(target)

    dp = [[0] \* (n + 1) for \_ in range(m + 1)]

    for i in range(m + 1):

        for j in range(n + 1):

            if i == 0:

                dp[i][j] = j

            elif j == 0:

                dp[i][j] = i

            elif source[i - 1] == target[j - 1]:

                dp[i][j] = dp[i - 1][j - 1]

            else:

                dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1])

    return dp[m][n]

print("Edit Distance:", min\_edit\_dist("intention", "execution"))

11)

import nltk

from textblob import TextBlob

#sampleSentence = '''This is a very good book, felt really nice reading it.'''

#sampleSentence = '''Ravi is a bad leader, he misguides the team too often'''

sampleSentence=''' Learning NLP equips individuals good with skills

to analyze vast amounts of textual data, build intelligent chatbots,

automate language-related tasks, and contribute to groundbreaking

advancements in fields like artificial intelligence and linguistics'''

sampleSentence = nltk.word\_tokenize(sampleSentence)

sampleSentencePOS = nltk.pos\_tag(sampleSentence)

#Find Nouns or proper Nouns

OnlyNouns = (" ").join([POStags[0] for POStags in

sampleSentencePOS if POStags[1] in ['NN','NNP']])

# Find only Adjectives

OnlyAdjectives= (" ").join([POStags[0] for POStags in sampleSentencePOS if POStags[1] in ['JJ','JJR','JJS']])

print ('Nouns: ', OnlyNouns)

print ('Adjectives: ', OnlyAdjectives)

print('Overall sentiment score of ajdectives:',TextBlob(OnlyAdjectives).sentiment)

12)

import pandas as pd

import re

import nltk

import matplotlib.pyplot as plt

from textblob import TextBlob

from sklearn.feature\_extraction.text import CountVectorizer

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay

from wordcloud import WordCloud

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# Preprocess text

nltk.download('stopwords')

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

def preprocess(text):

text = re.sub(r"http\S+|@\w+|#", '', text.lower()) # Remove URLs, mentions, hashtags

return " ".join([word for word in word\_tokenize(text) if word not in stop\_words])

# Load and preprocess data

df = pd.read\_csv('vaccination\_tweets.csv')

df['text'] = df['text'].dropna().apply(preprocess)

df['sentiment'] = df['text'].apply(lambda x: 'Positive' if TextBlob(x).sentiment.polarity > 0

else 'Negative' if TextBlob(x).sentiment.polarity < 0 else 'Neutral')

# Visualize sentiment distribution

df['sentiment'].value\_counts().plot.pie(autopct='%1.1f%%', startangle=90, colors=['yellowgreen', 'gold', 'red'], title='Sentiments')

plt.show()

# Generate word clouds for each sentiment

for sentiment in ['Positive', 'Negative', 'Neutral']:

text = ' '.join(df[df['sentiment'] == sentiment]['text'])

WordCloud(background\_color='white').generate(text).to\_image().show()

# Train sentiment classification model

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df['text'], df['sentiment'], test\_size=0.2, random\_state=42)

vectorizer=CountVectorizer(ngram\_range=(1,2))

x\_train=vectorizer.fit\_transform(x\_train)

x\_test=vectorizer.transform(x\_test)

model = LogisticRegression().fit(x\_train, y\_train)

# Evaluate the model

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

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred) \* 100:.2f}%")

print(classification\_report(y\_test, y\_pred))

ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels=model.classes\_).plot()

plt.show()