

**School Of Computer Science**

**Computational Linguistics and Natural Language Processing Lab File**

**Name:** Charu Gupta

**Sap ID:** 500090912

**Roll No.:** R2142210244

**Batch:** B1 Hons. AIML

**Semester:** VI

**Subject Code:** CSEG3124

**Session:** Jan 2024- June 24

**Submitted to: Dr. Achala Shakya**

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO. | EXPERIMENT NAME | DATE | PAGES |
| 1. | Write code to load CSV file containing information about employee of a company in python and draw graph showing average salary department wise. | 10-01-24 | 3 - 8 |
| 2 | Data Preprocessing | 17-01-24 | 9 - 13 |
| 3 | Calculate word count of a given specific document and show top 10 frequent words with their frequency and Create world cloud and show graphically. | 24-01-24 | 14-17 |
| 4 | TF-IDF and N-gram (unigram, bigram & trigram) probability distribution | 31-01-24 | 18-25 |
| 5 | Implement chunking and parsing to extract Noun Phrases. | 07-02-24 | 26-28 |
| 6 | Sentiment Analysis | 21-02-24 | 29-34 |
| 7 | Identify semantic relationship between the words from the text (use WorNet library) | 13-03-24 | 35-37 |
| 8 | Text summarization | 20-03-24 | 38-39 |
| 9 | Information Retrieval | 27-03-24 | 40-41 |
| 10 | Huffmann Coding | 10-04-24 | 42-43 |

**Experiment – 01**

**Title:** Write code to load CSV file containing information about employee of a company in python and draw graph showing average salary department wise.

**Input:**

import pandas as pd

import matplotlib.pyplot as plt

# Load your CSV file (replace 'your\_dataset.csv' with the actual path or URL)

df = pd.read\_csv('Employee.csv')

# Calculate average salary department-wise

average\_salary\_per\_department = df.groupby('Department')['Salary'].mean().reset\_index()

# Plotting Average Salary as a Bar Chart

plt.figure(figsize=(12, 8))

plt.bar(average\_salary\_per\_department['Department'], average\_salary\_per\_department['Salary'], color='skyblue')

plt.xlabel('Department')

plt.ylabel('Average Salary')

plt.title('Average Salary Department Wise (Bar Chart)')

plt.xticks(rotation=45, ha='right')

plt.show()

# Plotting Average Salary as a Pie Chart

plt.figure(figsize=(8, 8))

plt.pie(average\_salary\_per\_department['Salary'], labels=average\_salary\_per\_department['Department'], autopct='%1.1f%%', startangle=90)

plt.title('Average Salary Department Wise (Pie Chart)')

plt.show()

# Calculate median salary department-wise

median\_salary\_per\_department = df.groupby('Department')['Salary'].median().reset\_index()

# Plotting Median Salary as a Bar Chart

plt.figure(figsize=(12, 8))

plt.bar(median\_salary\_per\_department['Department'], median\_salary\_per\_department['Salary'], color='salmon')

plt.xlabel('Department')

plt.ylabel('Median Salary')

plt.title('Median Salary Department Wise (Bar Chart)')

plt.xticks(rotation=45, ha='right')

plt.show()

# Plotting Median Salary as a Pie Chart

plt.figure(figsize=(8, 8))

plt.pie(median\_salary\_per\_department['Salary'], labels=median\_salary\_per\_department['Department'], autopct='%1.1f%%', startangle=90)

plt.title('Median Salary Department Wise (Pie Chart)')

plt.show()

# Calculate minimum salary department-wise

min\_salary\_per\_department = df.groupby('Department')['Salary'].min().reset\_index()

# Plotting Minimum Salary as a Bar Chart

plt.figure(figsize=(12, 8))

plt.bar(min\_salary\_per\_department['Department'], min\_salary\_per\_department['Salary'], color='lightgreen')

plt.xlabel('Department')

plt.ylabel('Minimum Salary')

plt.title('Minimum Salary Department Wise (Bar Chart)')

plt.xticks(rotation=45, ha='right')

plt.show()

# Plotting Minimum Salary as a Pie Chart

plt.figure(figsize=(8, 8))

plt.pie(min\_salary\_per\_department['Salary'], labels=min\_salary\_per\_department['Department'], autopct='%1.1f%%', startangle=90)

plt.title('Minimum Salary Department Wise (Pie Chart)')

plt.show()

# Calculate maximum salary department-wise

max\_salary\_per\_department = df.groupby('Department')['Salary'].max().reset\_index()

# Plotting Maximum Salary as a Bar Chart

plt.figure(figsize=(12, 8))

plt.bar(max\_salary\_per\_department['Department'], max\_salary\_per\_department['Salary'], color='lightcoral')

plt.xlabel('Department')

plt.ylabel('Maximum Salary')

plt.title('Maximum Salary Department Wise (Bar Chart)')

plt.xticks(rotation=45, ha='right')

plt.show()

# Plotting Maximum Salary as a Pie Chart

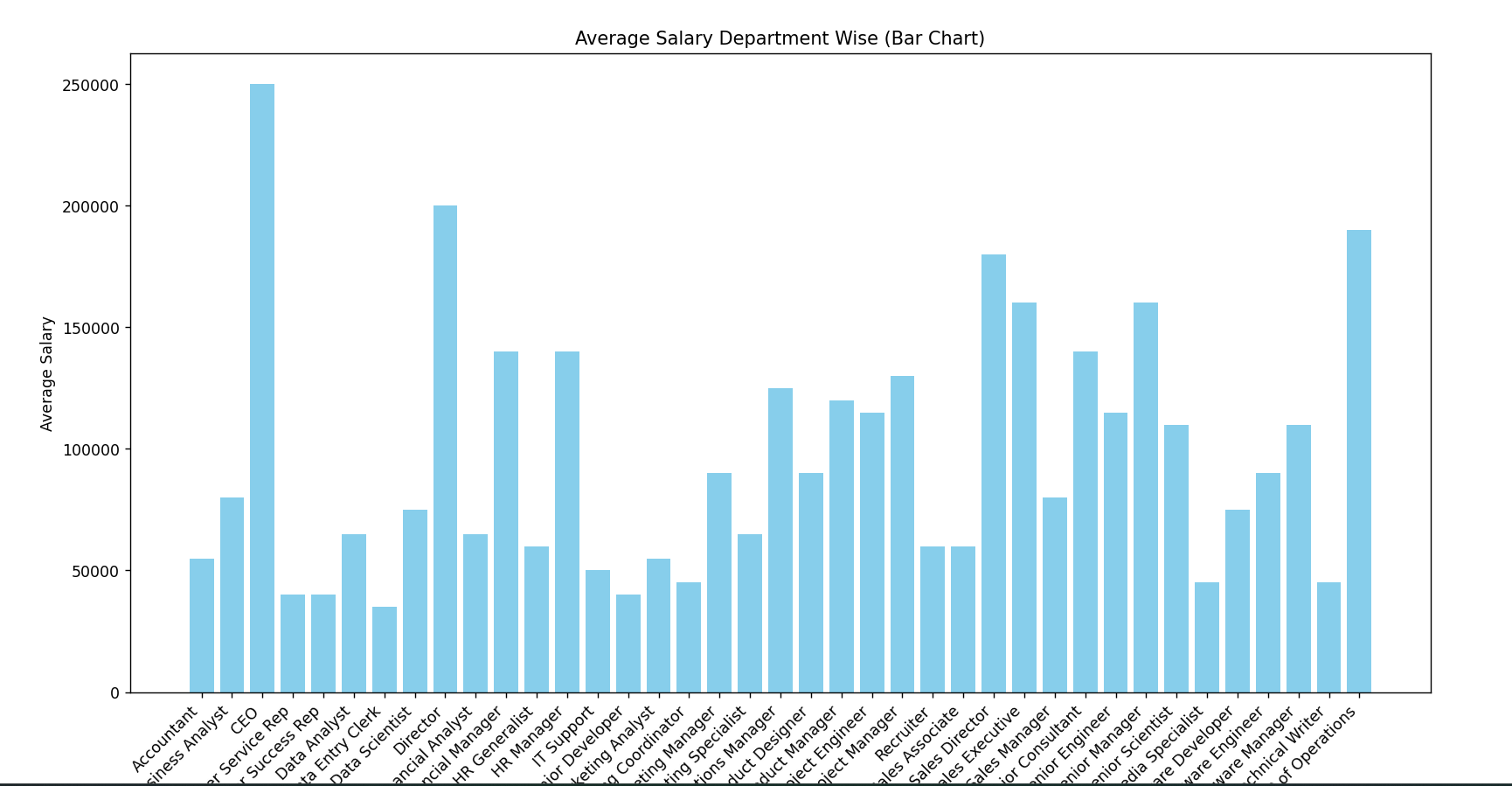
plt.figure(figsize=(8, 8))

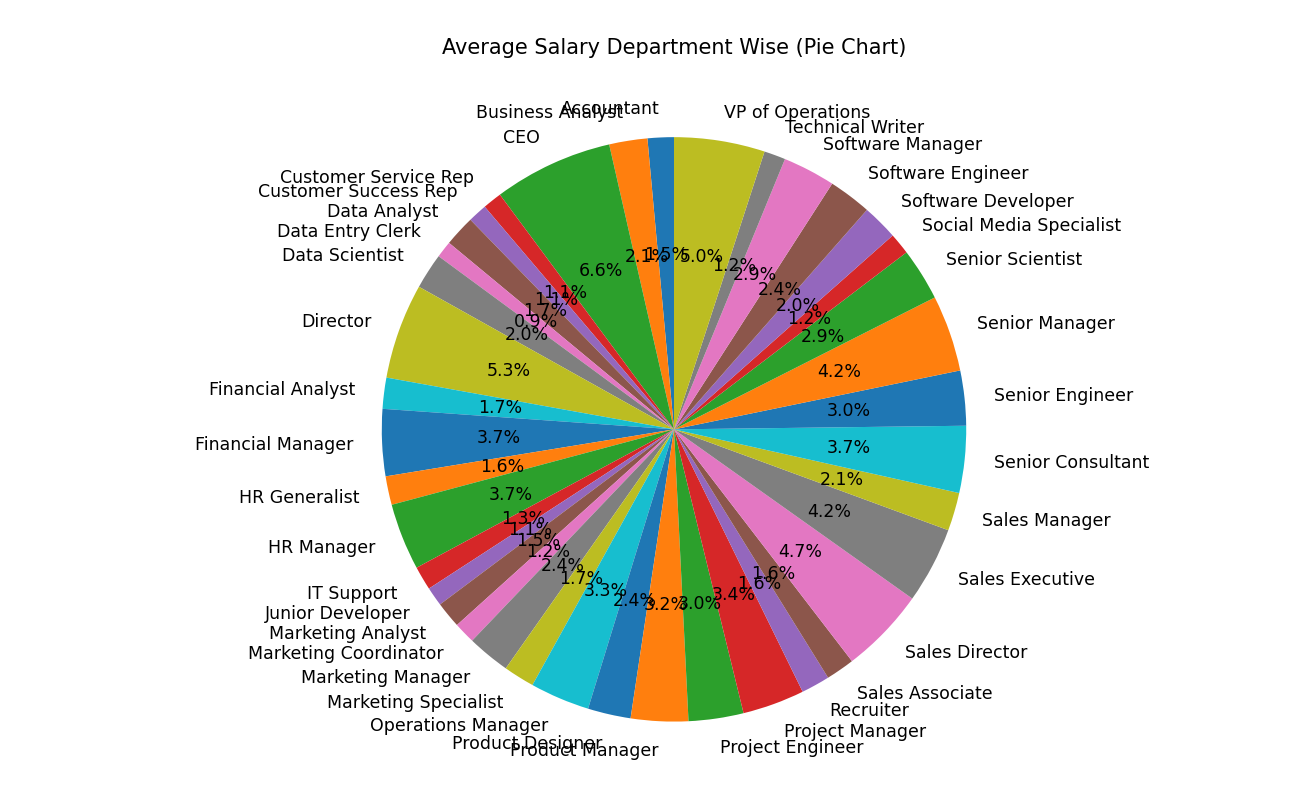
plt.pie(max\_salary\_per\_department['Salary'], labels=max\_salary\_per\_department['Department'], autopct='%1.1f%%', startangle=90)

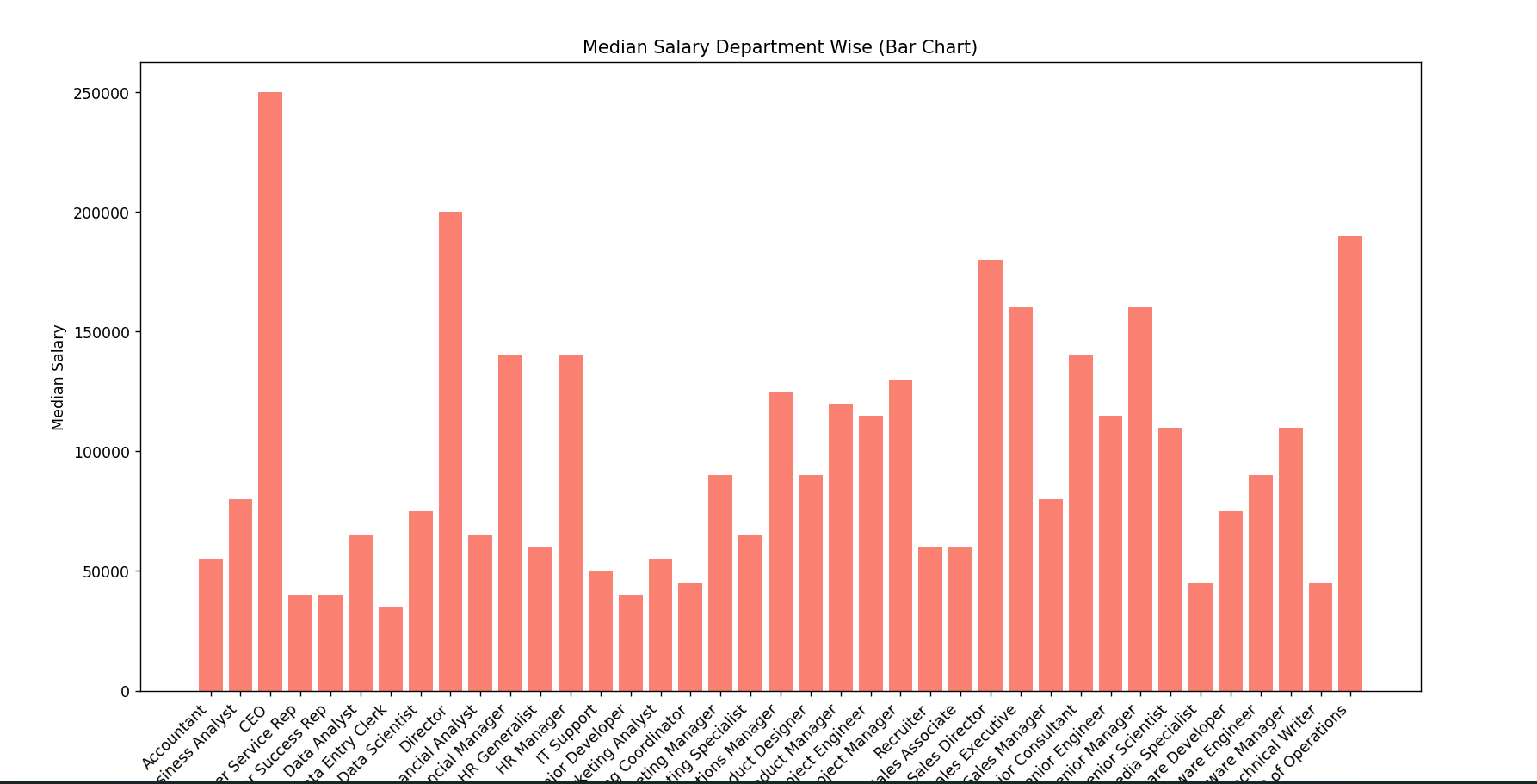
plt.title('Maximum Salary Department Wise (Pie Chart)')

plt.show()

**Output:**

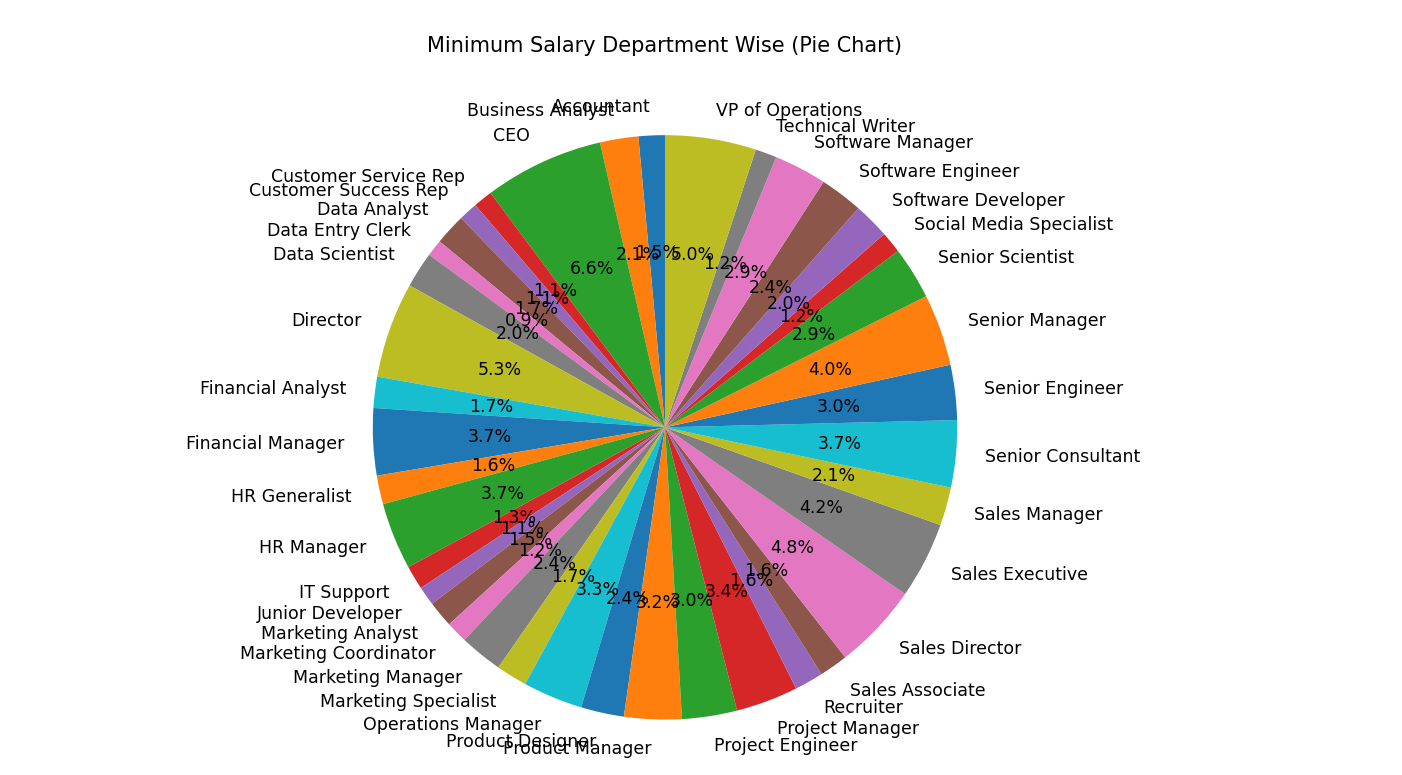
****

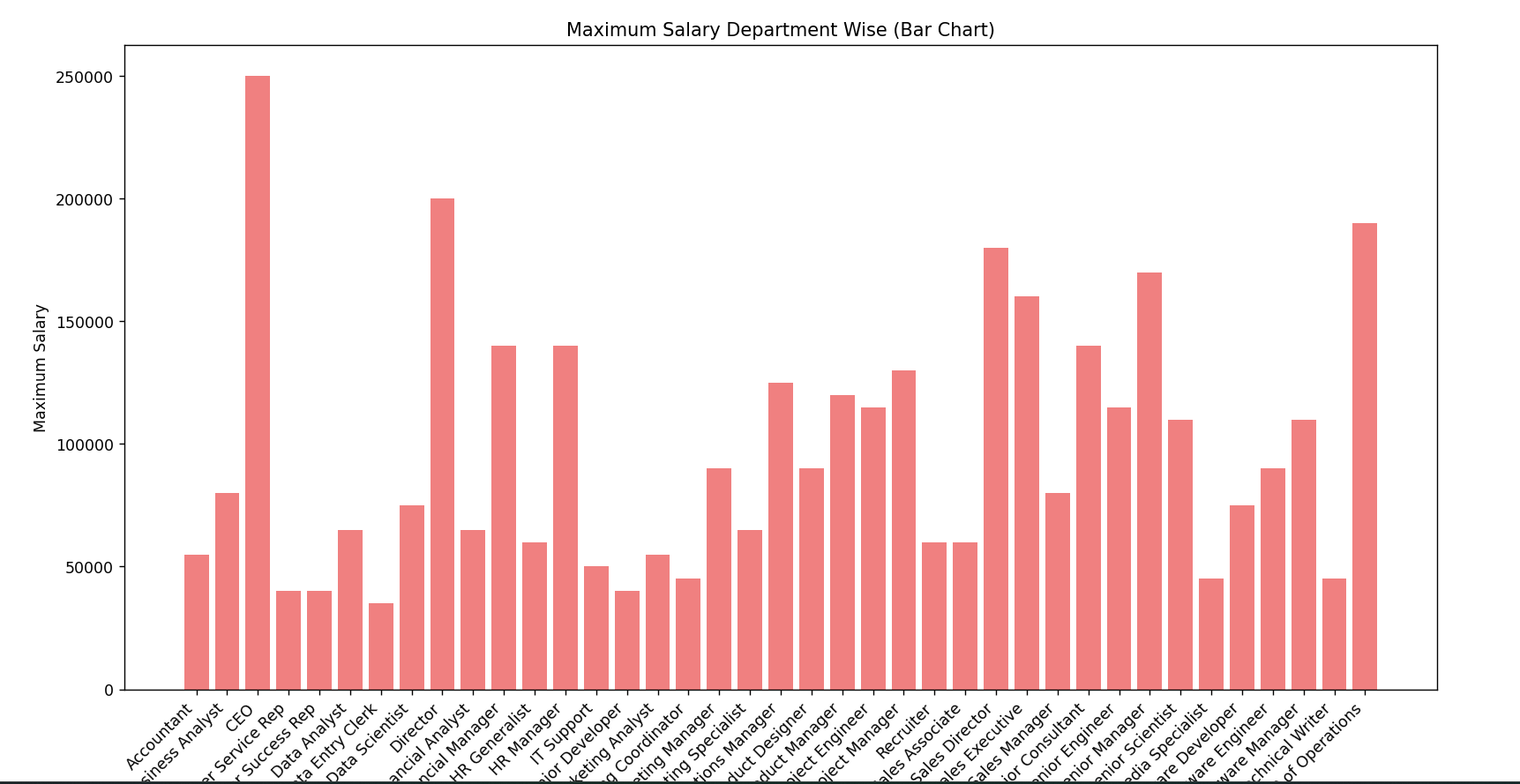
****

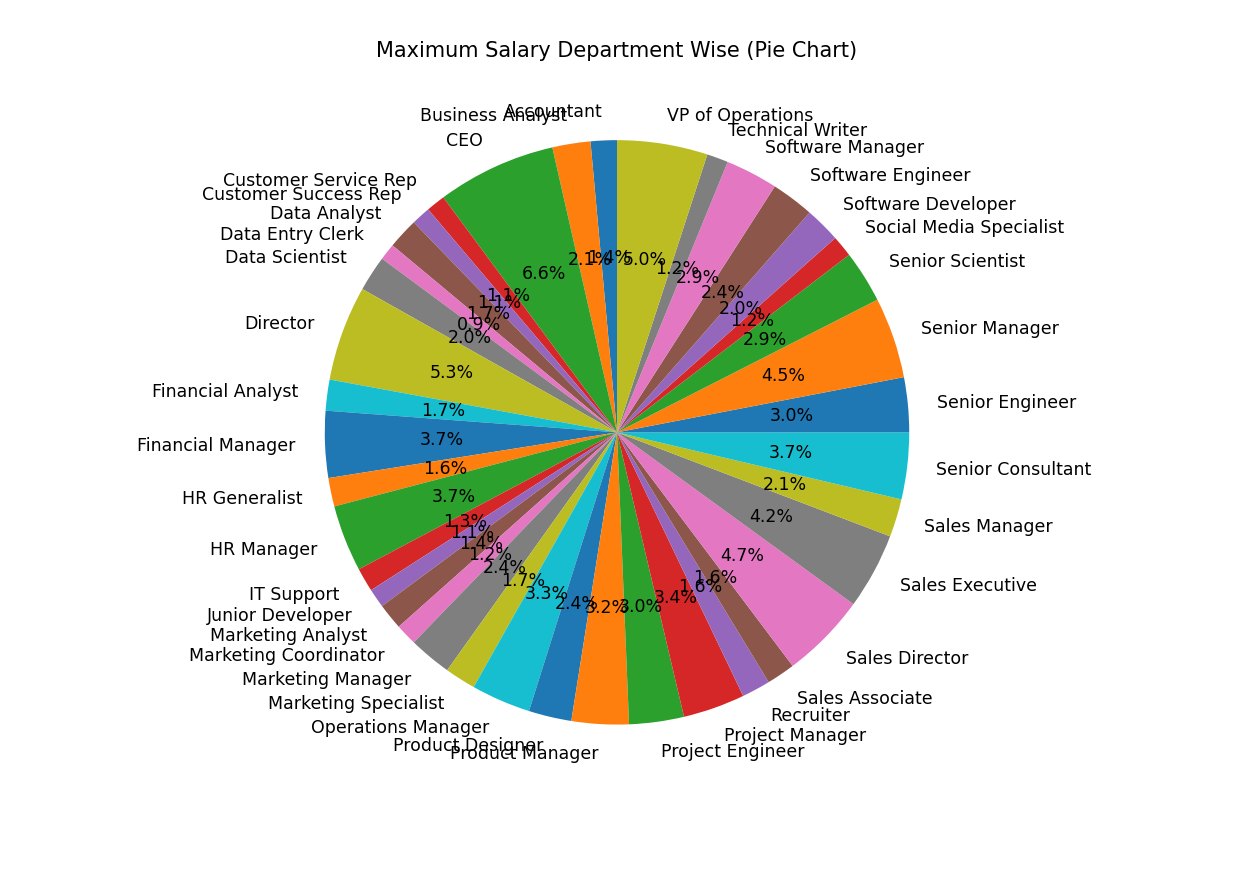
****

****

****

****

****

****

**EXPERIMENT- 02**

**Title:** Processing Data and Language Processor

**Input:**

import nltk

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

from nltk.stem import WordNetLemmatizer, PorterStemmer

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk import pos\_tag, download

# Download the required NLTK packages

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

download('averaged\_perceptron\_tagger')

file\_path = 'business\_2.txt'

with open(file\_path, 'r', encoding='utf-8') as file:

    text\_content = file.read()

# Sentence Tokenization

def tokenize\_sentences(text):

    sentences = sent\_tokenize(text)

    return sentences

# Sentence Tokenization

sentences = tokenize\_sentences(text\_content)

print("Tokenized Sentences:")

print(sentences)

Tokenized\_text = ' '.join(sentences)

with open('Tokenized\_text.txt', 'w') as f:

    f.write(Tokenized\_text)

# Tokenization

def tokenize\_text(text):

    words = word\_tokenize(text)

    return words

# Tokenization

tokenized\_text = tokenize\_text(text\_content)

print("Tokenized Text:")

print(tokenized\_text)

Tokenized\_Sen = ' '.join(Tokenized\_text)

with open('Tokenized\_Sen .txt', 'w') as f:

    f.write(Tokenized\_text)

# Remove stopwords

def remove\_stopwords(words):

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

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

    return filtered\_words

# Remove stopwords

filtered\_text = remove\_stopwords(tokenized\_text)

print("\nText after Stopword Removal:")

print(filtered\_text)

filtered\_text = ' '.join(filtered\_text)

with open('filtered\_text .txt', 'w') as f:

    f.write(filtered\_text)

# Stemming

def stem\_words(words):

    stemmer = PorterStemmer()

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

    return stemmed\_words

# Stemming

stemmed\_words = stem\_words(lemmatized\_words)

print(stemmed\_words)

stemmed\_text = ' '.join(stemmed\_words)

with open('stemmed.txt', 'w') as f:

    f.write(stemmed\_text)

# Lemmatization

def lemmatize\_words(words):

    lemmatizer = WordNetLemmatizer()

    lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]

    return lemmatized\_words

# Lemmatization

lemmatized\_text = lemmatize\_words(filtered\_text)

print("\nLemmatized Text:")

print(lemmatized\_text)

lemmatized\_text= ' '.join(filtered\_text)

with open('lemmatized\_text.txt', 'w') as f:

    f.write(lemmatized\_text)

# POS Tagging

pos\_tags = pos\_tag(stemmed\_words)

print(pos\_tags)

preprocessed\_text = ' '.join(lemmatized\_text)

with open('preprocessed\_text.txt', 'w') as f:

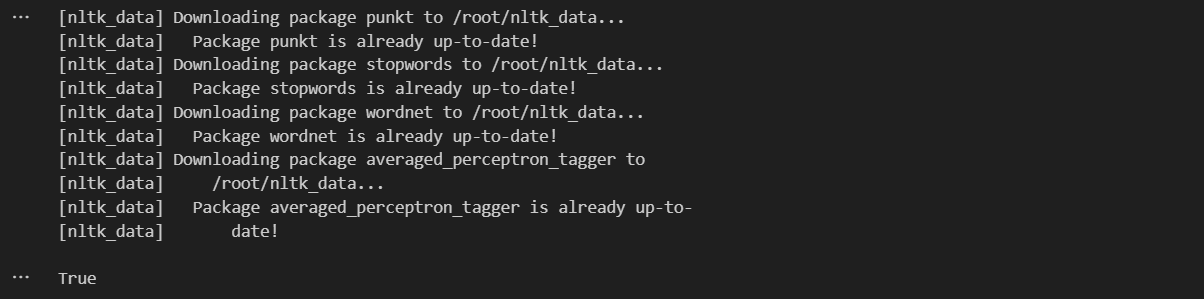
    f.write(preprocessed\_text)

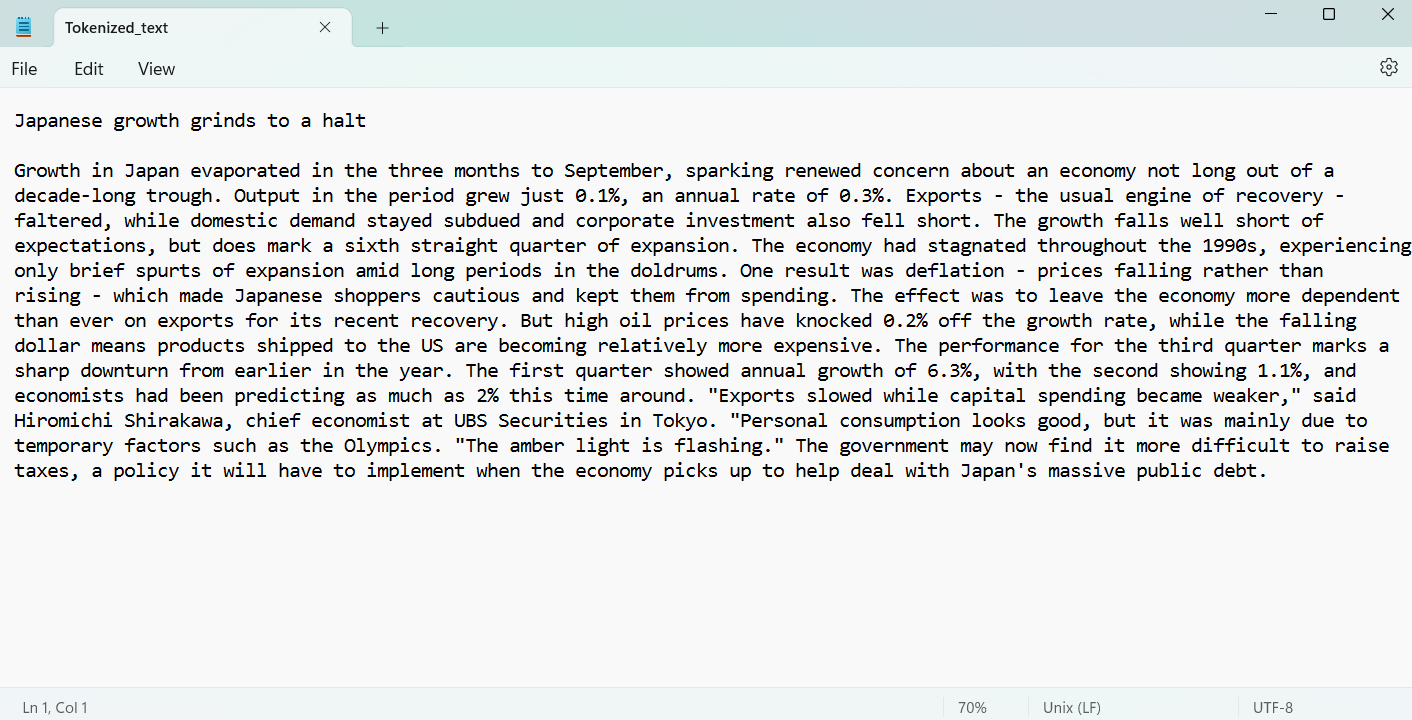
pos\_tags\_text = ' '.join(stemmed\_words)

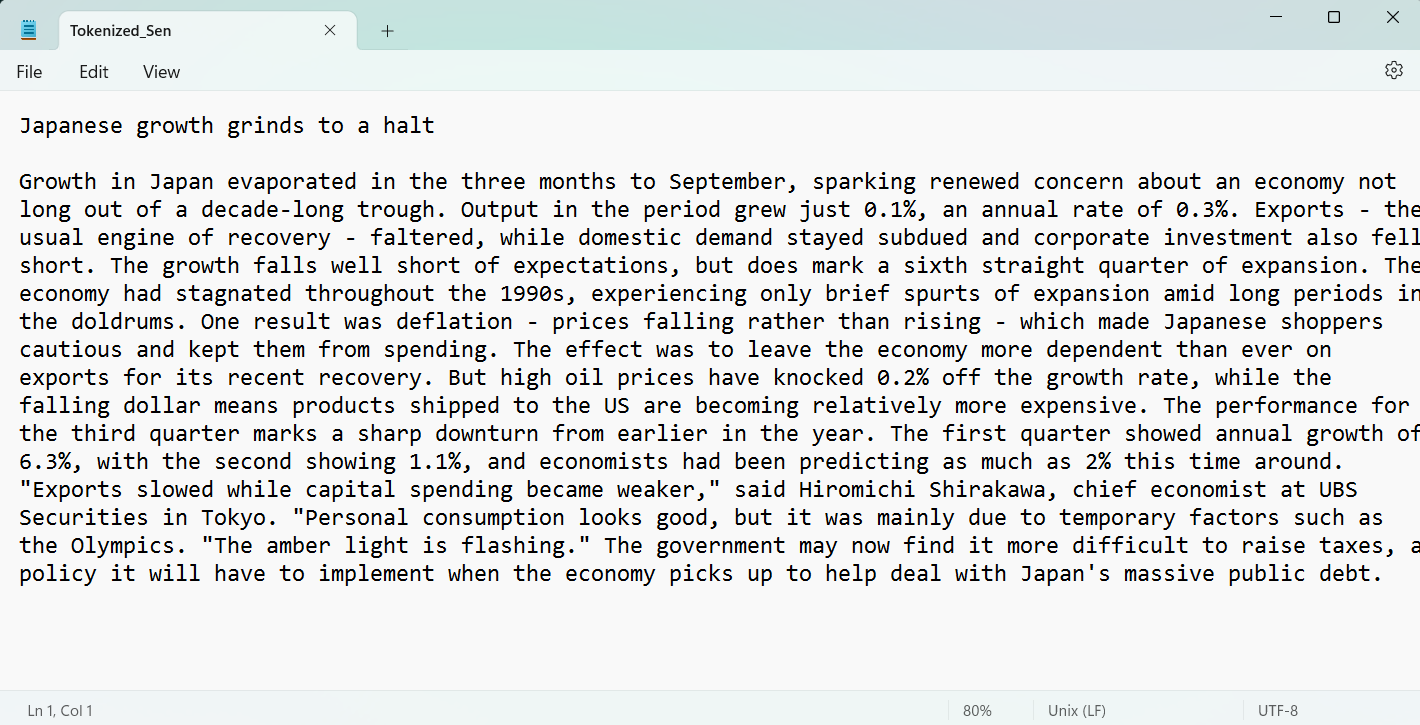
with open('pos\_tags\_text.txt', 'w') as f:

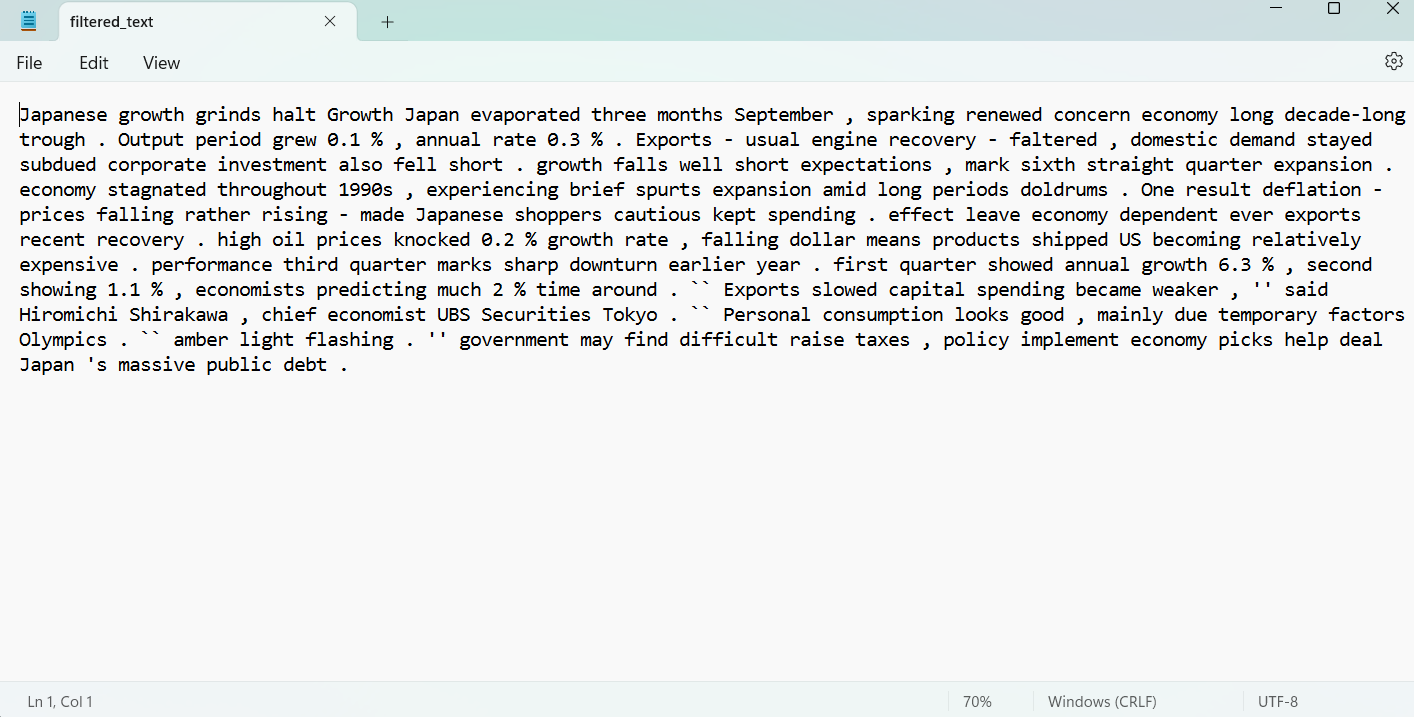
    f.write(pos\_tags\_text)

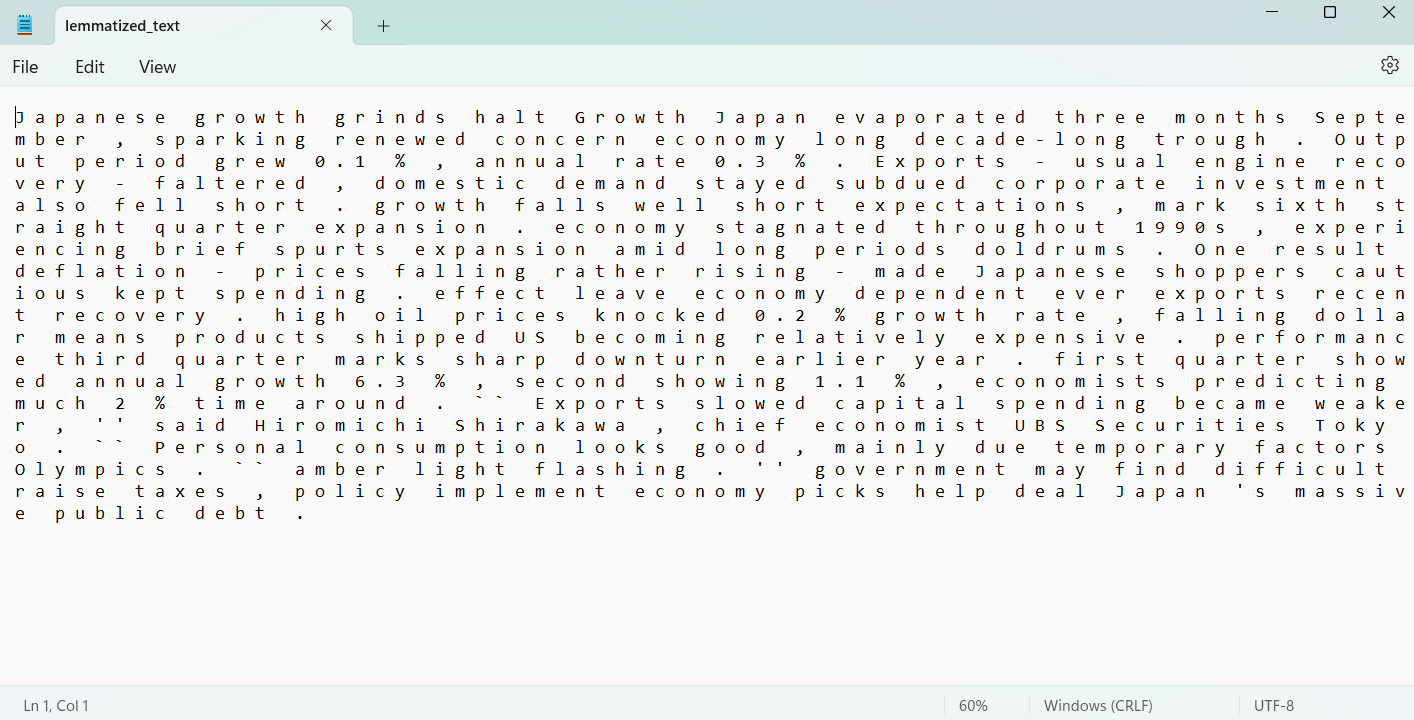
**Output:**

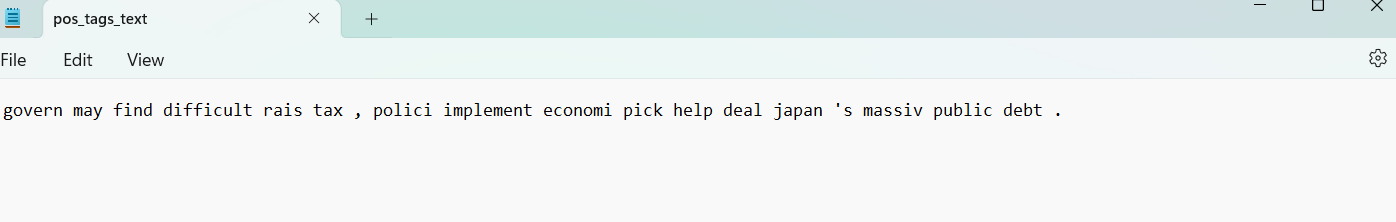
****

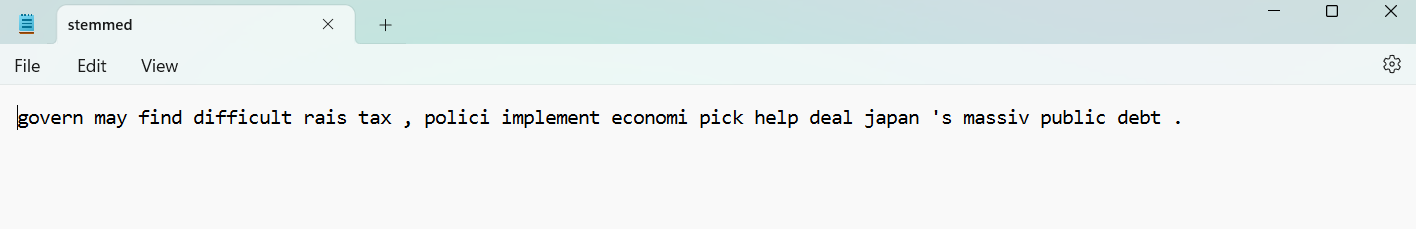
****

****

****

****

****

****

****

**EXPERIMENT- 03**

**Title:** Spacy Language Processor

**Input:**

import nltk

from nltk import FreqDist

from wordcloud import WordCloud, STOPWORDS

import matplotlib.pyplot as plt

import string

# Download NLTK data (run this line only once)

nltk.download('punkt')

nltk.download('stopwords')

# Function to calculate word count and display top 10 frequent words with separate Word Cloud and Frequency Distribution graphs

def analyze\_text(text):

    # Tokenize the text into words

    words = nltk.word\_tokenize(text)

    # Remove punctuation, including full stops

    words = [word for word in words if word not in string.punctuation]

    # Remove stopwords

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

    words = [word for word in words if word.lower() not in stop\_words]

    # Calculate word count

    word\_count = len(words)

    # Display word count

    print(f"Total words in the document: {word\_count}")

    # Calculate word frequencies

    word\_freq = FreqDist(words)

    # Display top 10 frequent words

    print("\nTop 10 frequent words:")

    top\_words = word\_freq.most\_common(10)

    for word, freq in top\_words:

        print(f"{word}: {freq} times")

    # Plot Frequency Distribution graph for top 10 words

    plt.figure(figsize=(10, 5))

    plt.bar([word[0] for word in top\_words], [word[1] for word in top\_words])

    plt.title('Frequency Distribution of Top 10 Words (excluding stopwords)')

    plt.xlabel('Words')

    plt.ylabel('Frequency')

    plt.xticks(rotation=45, ha='right')

    plt.tight\_layout()

    plt.show()

    # Create Word Cloud for top 10 words

    wordcloud = WordCloud(width=800, height=400, random\_state=21, max\_font\_size=110, stopwords=STOPWORDS).generate\_from\_frequencies(dict(top\_words))

    # Display Word Cloud

    plt.figure(figsize=(10, 5))

    plt.imshow(wordcloud, interpolation="bilinear")

    plt.axis('off')

    plt.title('Word Cloud of Top 10 Words (excluding stopwords)')

    plt.show()

    # Create Word Cloud

    wordcloud = WordCloud(width=800, height=400, random\_state=21, max\_font\_size=110).generate(text)

    # Display Word Cloud

    plt.figure(figsize=(10, 5))

    plt.imshow(wordcloud, interpolation="bilinear")

    plt.axis('off')

    plt.title('Word Cloud')

    plt.show()

# Sample text (replace this with the content of your specific document)

sample\_text = """

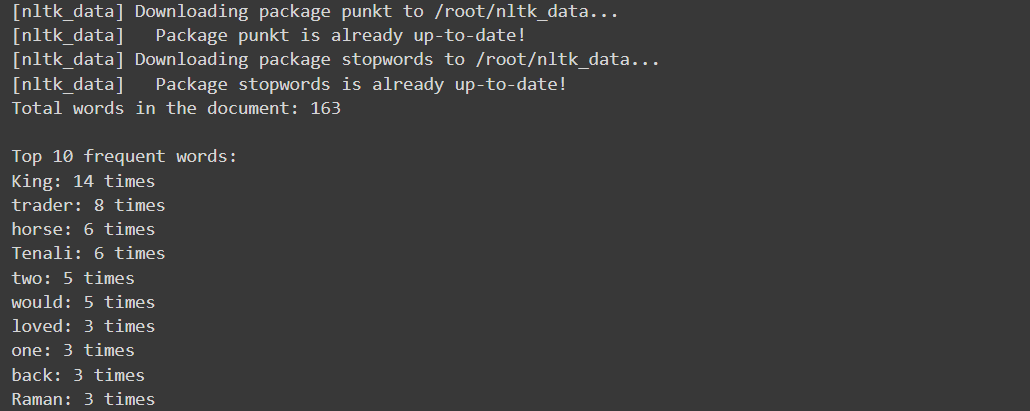
    King Krishnadevaraya loved horses and had the best collection of horse breeds in the Kingdom. Well, one day, a trader came to the King and told him that he had brought with him a horse of the best breed in Arabia.He invited the King to inspect the horse. King Krishnadevaraya loved the horse; so the trader said that the King could buy this one and that he had two more like this one, back in Arabia that he would go back to get. The King loved the horse so much that he had to have the other two as well. He paid the trader 5000 gold coins in advance. The trader promised that he would return within two days with the other horses.Two days turned into two weeks, and still, there was no sign of the trader and the two horses. One evening, to ease his mind, the King went on a stroll in his garden. There he spotted Tenali Raman writing down something on a piece of paper. Curious, the King asked Tenali what he was jotting down.Tenali Raman was hesitant, but after further questioning, he showed the King the paper. On the paper was a list of names, the King’s being at the top of the list. Tenali said these were the names of the biggest fools in the Vijayanagara Kingdom!As expected, the King was furious that his name was at the top and asked Tenali Raman for an explanation. Tenali referred to the horse story, saying the King was a fool to believe that the trader, a stranger, would return after receiving 5000 gold coins.Countering his argument, the King then asked, what happens if/when the trader does come back? In true Tenali humour, he replied saying, in that case, the trader would be a bigger fool, and his name would replace the King’s on the list!

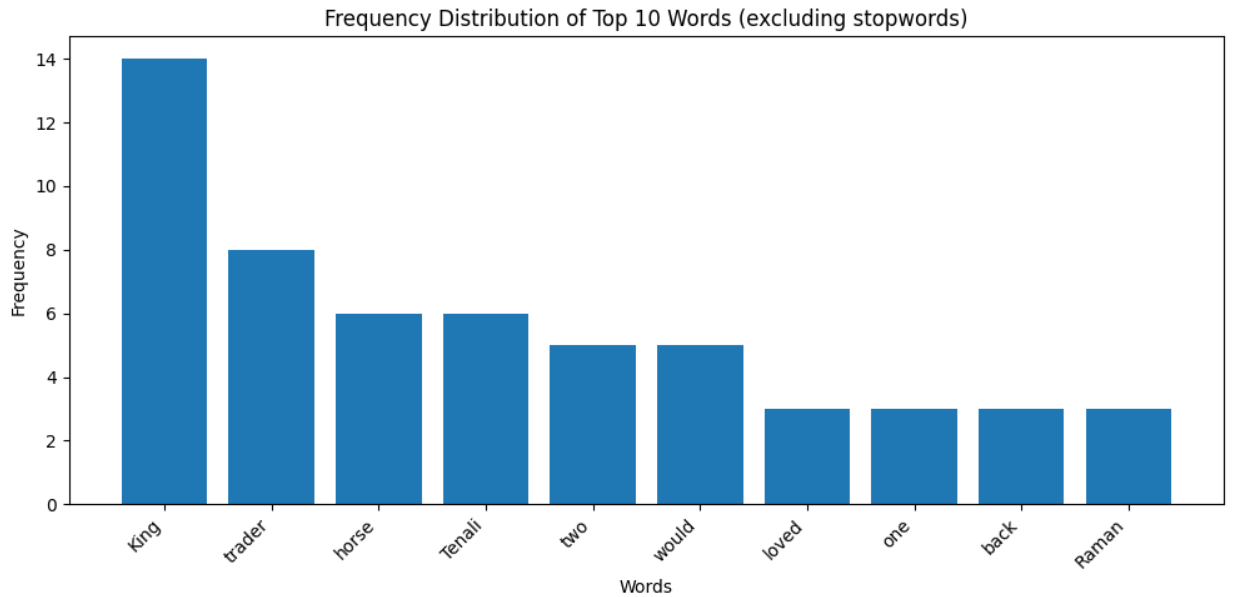
"""

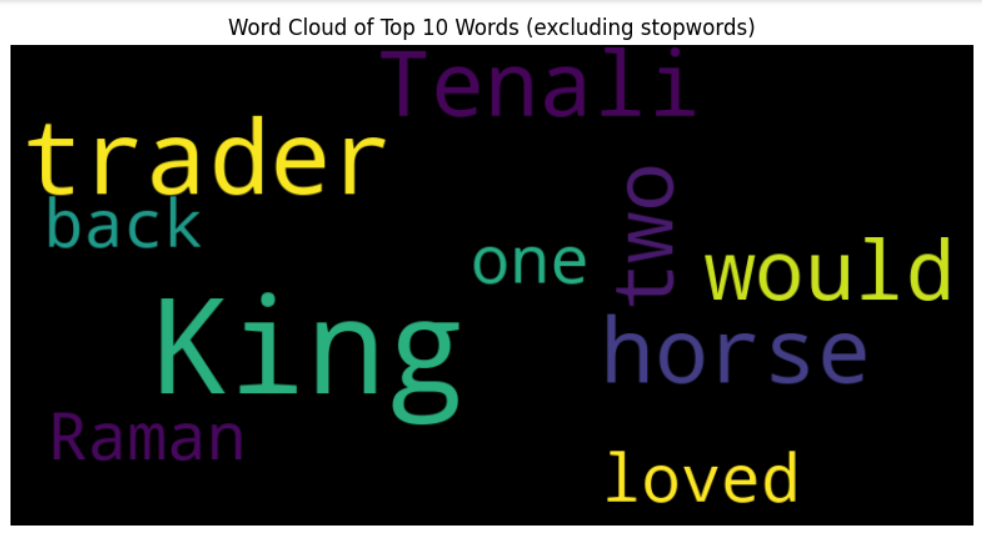
# Analyze the text

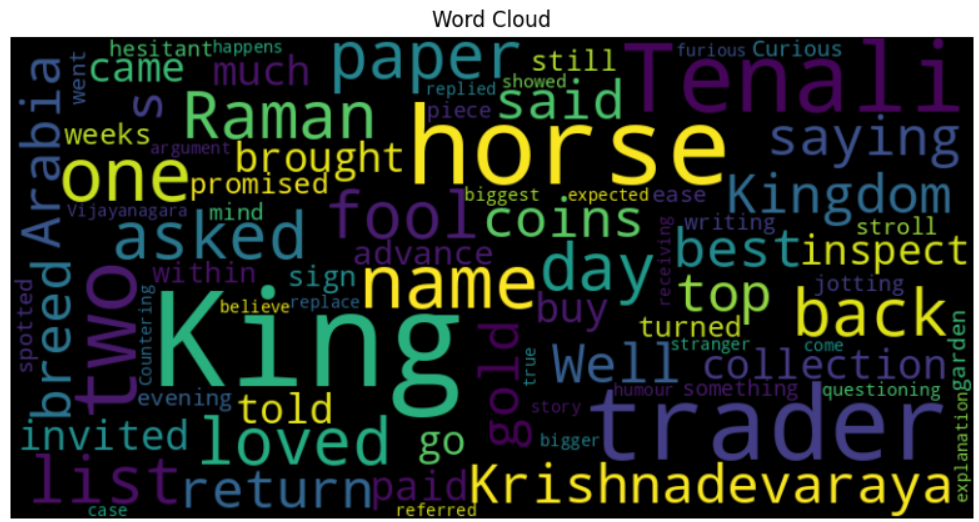
analyze\_text(sample\_text)

**Output:**

****

****

****



**EXPERIMENT- 04**

**Title:** TF-IDF and N-gram (unigram, bigram & trigram) probability distribution

**Input:**

import nltk

import string

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk import ngrams

from nltk.corpus import stopwords

from collections import Counter

# Download the 'punkt' resource

nltk.download('punkt')

nltk.download('stopwords')

# Sample corpus

corpus = [

    "King Krishnadevaraya loved horses and had the best collection of horse breeds in the Kingdom."

    " Well, one day, a trader came to the King and told him that he had brought with him a horse of the best breed in Arabia."

    "He invited the King to inspect the horse."

    " King Krishnadevaraya loved the horse; so the trader said that the King could buy this one and that he had two more like this one, back in Arabia that he would go back to get."

    " The King loved the horse so much that he had to have the other two as well. "

    "He paid the trader 5000 gold coins in advance. The trader promised that he would return within two days with the other horses."

    "Two days turned into two weeks, and still, there was no sign of the trader and the two horses. "

    "One evening, to ease his mind, the King went on a stroll in his garden. There he spotted Tenali Raman writing down something on a piece of paper."

    " Curious, the King asked Tenali what he was jotting down.Tenali Raman was hesitant, but after further questioning, he showed the King the paper. "

    "On the paper was a list of names, the King’s being at the top of the list. Tenali said these were the names of the biggest fools in the Vijayanagara Kingdom!As expected, the King was furious that his name was at the top and asked Tenali Raman for an explanation. "

    "Tenali referred to the horse story, saying the King was a fool to believe that the trader, a stranger, would return after receiving 5000 gold coins."

    "Countering his argument, the King then asked, what happens if/when the trader does come back? In true Tenali humour, he replied saying, in that case, the trader would be a bigger fool, and his name would replace the King’s on the list!"

]

# Remove stopwords and punctuation

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

translator = str.maketrans('', '', string.punctuation)

filtered\_corpus = [' '.join([word.translate(translator) for word in nltk.word\_tokenize(sentence) if word.lower() not in stop\_words]) for sentence in corpus]

# TF-IDF

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform(filtered\_corpus)

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

# Convert TF-IDF matrix to a dense NumPy array for better readability

dense\_tfidf\_matrix = tfidf\_matrix.todense()

# Display TF-IDF results

print("TF-IDF Matrix:")

print(dense\_tfidf\_matrix)

print("\nFeature Names:")

print(feature\_names)

for n in [1, 2, 3]:

    # Tokenize the corpus into words

    tokenized\_corpus = [nltk.word\_tokenize(sentence) for sentence in filtered\_corpus]

    # Create N-grams

    n\_grams = [ngrams(sentence, n) for sentence in tokenized\_corpus]

    # Flatten the list of N-grams

    flattened\_n\_grams = [gram for grams in n\_grams for gram in grams]

    # Calculate N-gram frequencies

    n\_gram\_freq = Counter(flattened\_n\_grams)

    # Calculate N-gram probabilities

    total\_n\_grams = sum(n\_gram\_freq.values())

    n\_gram\_probabilities = {gram: freq / total\_n\_grams for gram, freq in n\_gram\_freq.items()}

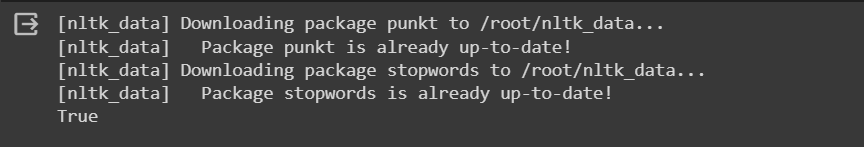
    # Display N-gram probabilities

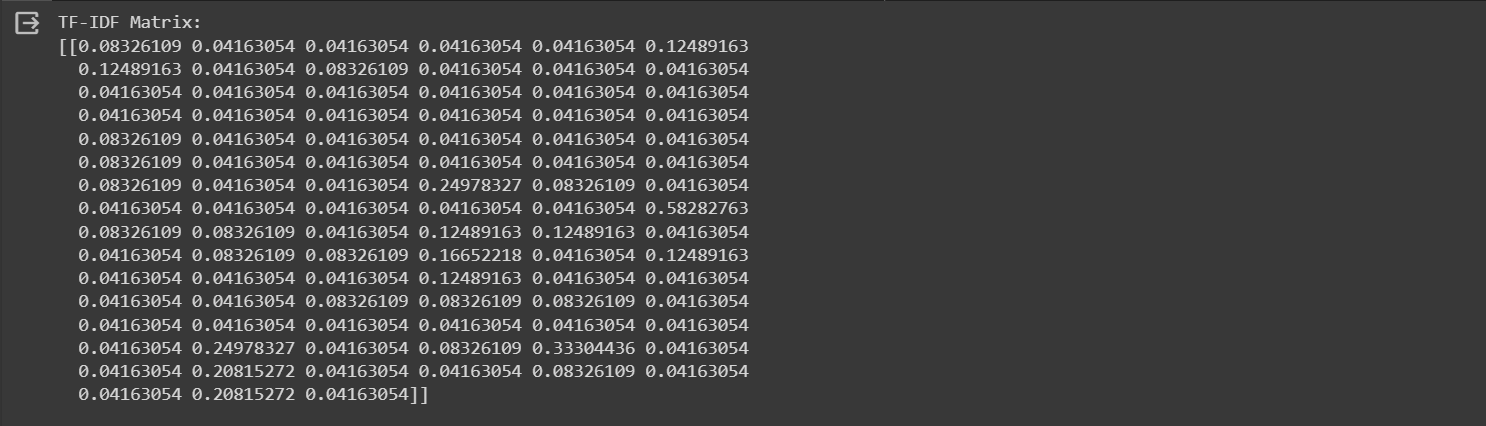
    print(f"\n{n}-gram Probabilities:")

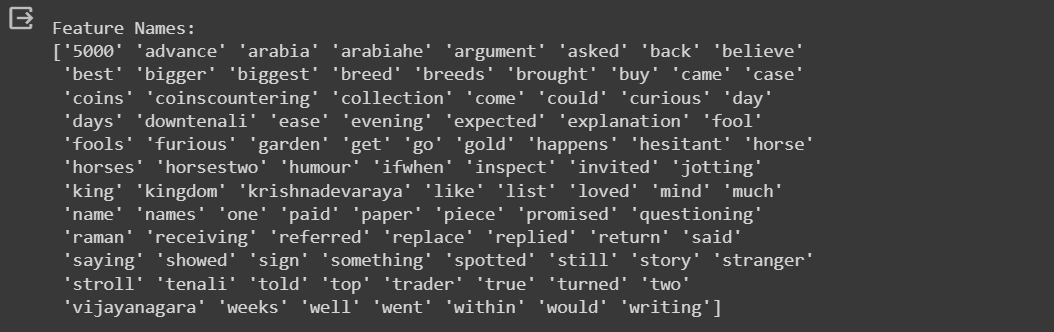
    for gram, prob in n\_gram\_probabilities.items():

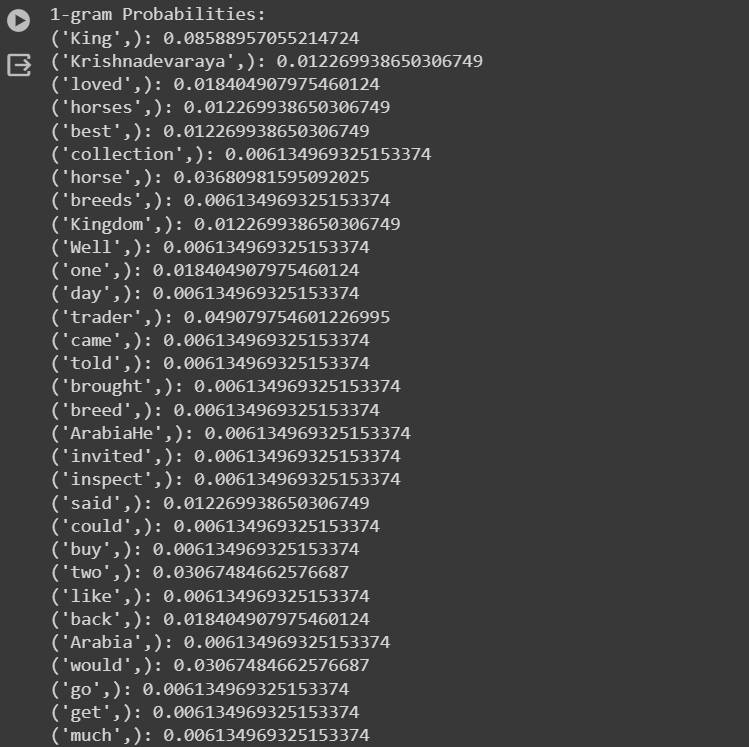
        print(f"{gram}: {prob}")

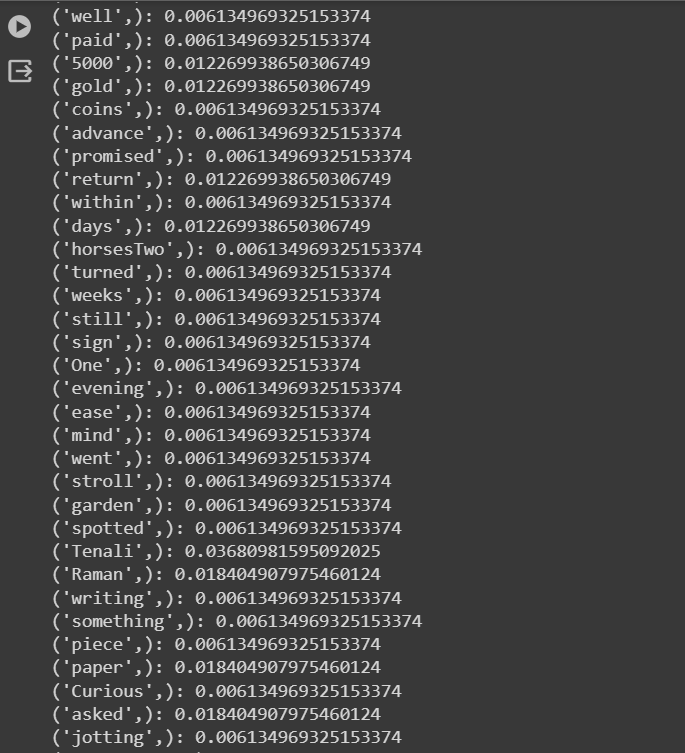
**Output:**

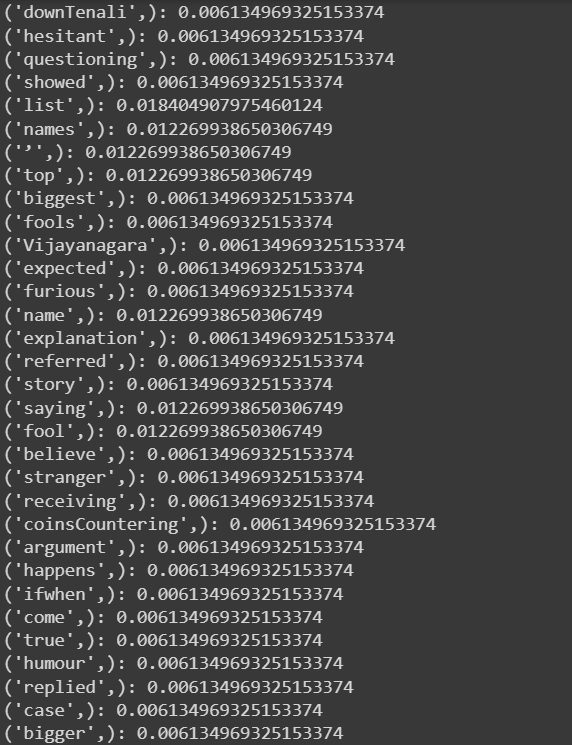
****

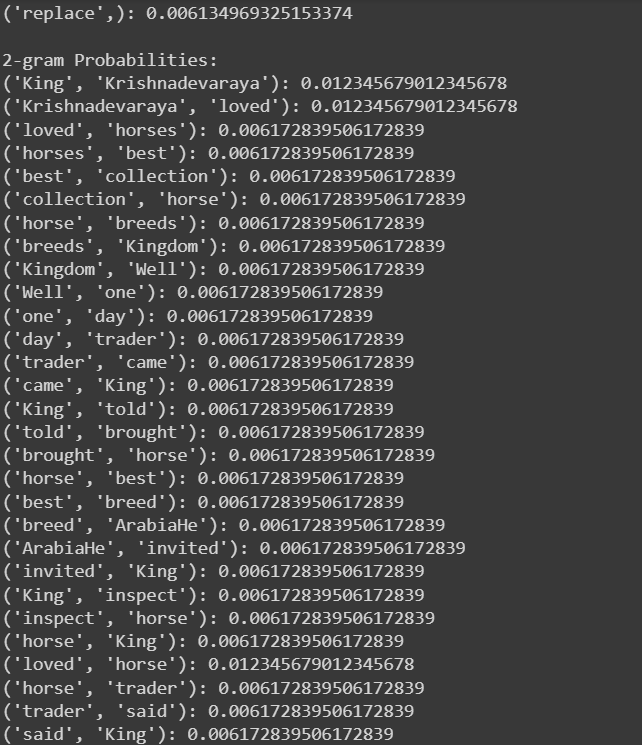
****

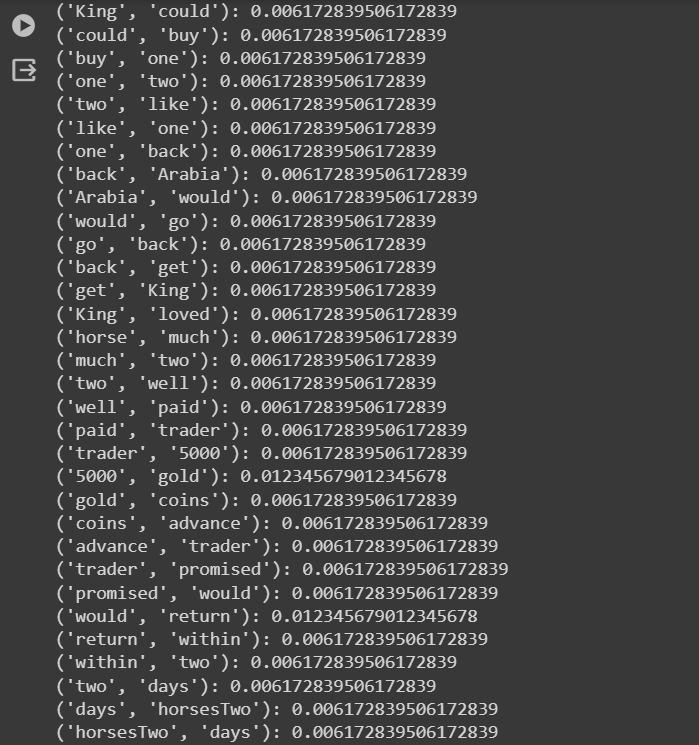
****

****

****

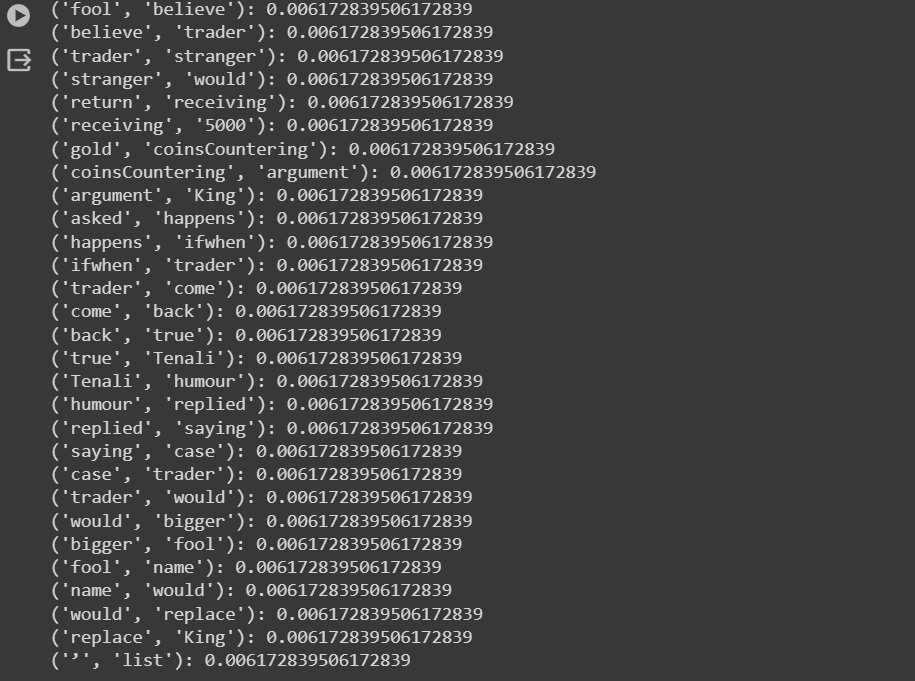
****

****

****

****

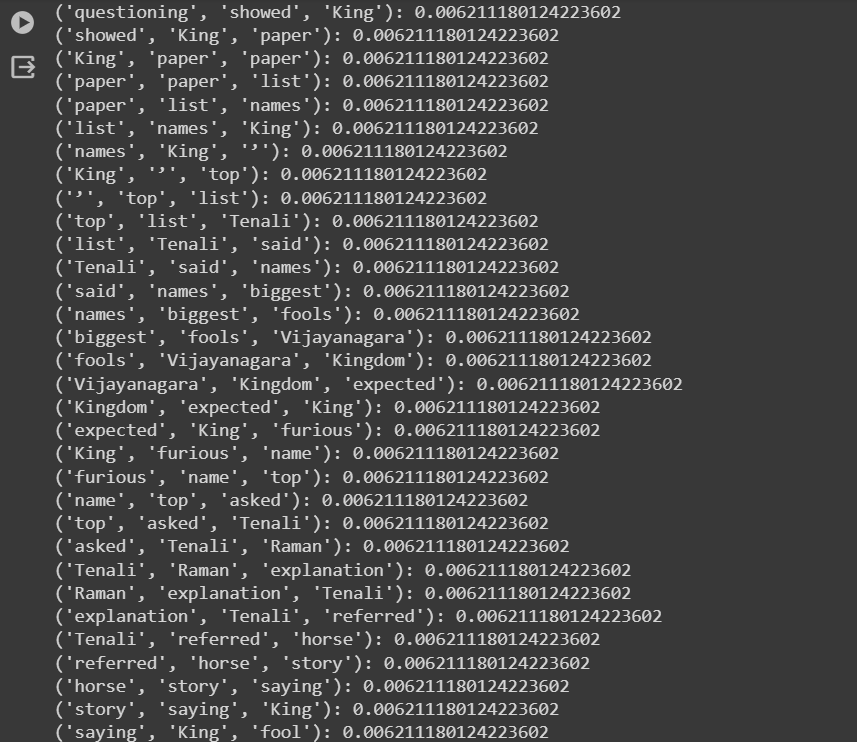
****

****

****

****

****

****

****

****

**EXPERIMENT- 05**

**Title:** Implement chunking to extract Noun Phrases.

**Input:**

import nltk

from nltk import pos\_tag, RegexpParser

from nltk.tokenize import word\_tokenize

# Download NLTK resources (you only need to do this once)

nltk.download('punkt')

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

# Function to extract noun phrases using chunking

def extract\_noun\_phrases(sentence):

    # Tokenize the sentence into words

    words = word\_tokenize(sentence)

    # Part-of-speech tagging

    tagged\_words = pos\_tag(words)

    # Define a chunking grammar to capture noun phrases

    grammar = r"""

        NP: {<DT>?<JJ>\*<NN>}   # NP: Noun Phrase

    """

    # Create a chunk parser with the defined grammar

    chunk\_parser = RegexpParser(grammar)

    # Apply chunking to the part-of-speech tagged words

    chunks = chunk\_parser.parse(tagged\_words)

    # Extract noun phrases from the chunked result

    noun\_phrases = []

    for subtree in chunks.subtrees():

        if subtree.label() == 'NP':

            noun\_phrases.append(" ".join([word for word, tag in subtree.leaves()]))

    return noun\_phrases

# Example usage

sentence = """King Krishnadevaraya loved horses and had the best collection of horse breeds in the Kingdom."

    "Well, one day, a trader came to the King and told him that he had brought with him a horse of the best breed in Arabia."

    "He invited the King to inspect the horse."

    " King Krishnadevaraya loved the horse; so the trader said that the King could buy this one and that he had two more like this one, back in Arabia that he would go back to get."

    " The King loved the horse so much that he had to have the other two as well. "

    "He paid the trader 5000 gold coins in advance. The trader promised that he would return within two days with the other horses."

    "Two days turned into two weeks, and still, there was no sign of the trader and the two horses. "

    "One evening, to ease his mind, the King went on a stroll in his garden. There he spotted Tenali Raman writing down something on a piece of paper."

    " Curious, the King asked Tenali what he was jotting down.Tenali Raman was hesitant, but after further questioning, he showed the King the paper. "

    "On the paper was a list of names, the King’s being at the top of the list. Tenali said these were the names of the biggest fools in the Vijayanagara Kingdom!As expected, the King was furious that his name was at the top and asked Tenali Raman for an explanation. "

    "Tenali referred to the horse story, saying the King was a fool to believe that the trader, a stranger, would return after receiving 5000 gold coins."

    "Countering his argument, the King then asked, what happens if/when the trader does come back? In true Tenali humour, he replied saying, in that case, the trader would be a bigger fool, and his name would replace the King’s on the list!"""

result = extract\_noun\_phrases(sentence)

# Save the results to a text file

output\_file\_path = "noun\_phrases\_output.txt"

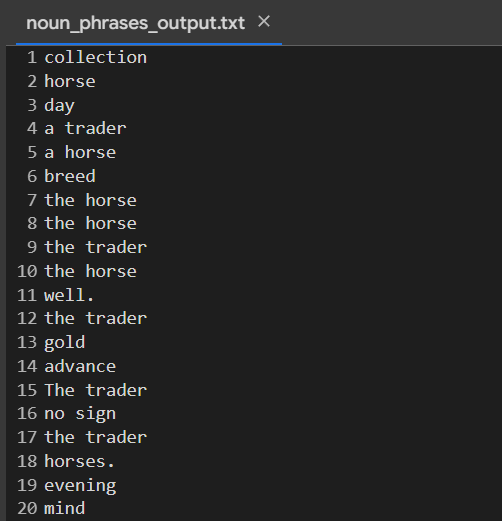
with open(output\_file\_path, 'w') as output\_file:

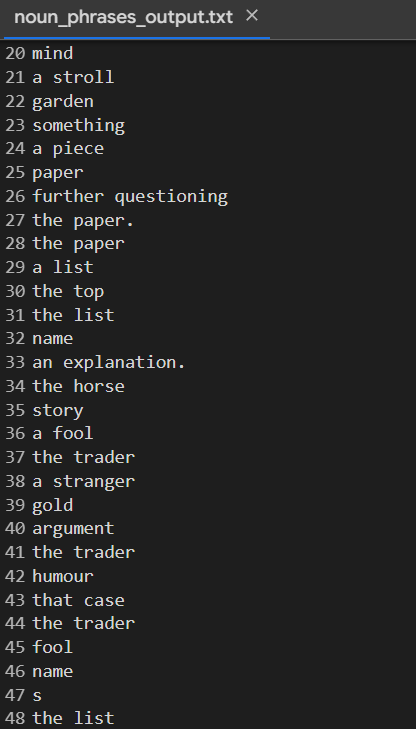
    for phrase in result:

        output\_file.write(phrase + '\n')

print(f"Noun Phrases saved to {output\_file\_path}")

**Output:**

****

****

**EXPERIMENT- 06**

**Title:** Sentiment Analysis

**Input:**

import numpy as np

import pandas as pd

df = pd.read\_csv("train.txt",sep=";",

    names=["Description","Emotion"])

df.head(5)

df['Emotion'].value\_counts()

df['label\_num'] = df['Emotion'].map({

    'joy': 0,

    'sadness': 1,

    'anger': 2,

    'fear': 3,

    'love': 4,

    'surprise':5

})

df.head(5)

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

X\_train,X\_test,y\_train,y\_test = train\_test\_split(df.Description,df.label\_num,test\_size=0.2)

print("Shape of X\_train: ", X\_train.shape)

print("Shape of X\_test: ", X\_test.shape)

y\_train.value\_counts()

from sklearn.neighbors import KNeighborsClassifier

from sklearn.pipeline import Pipeline

from sklearn.metrics import classification\_report

from sklearn.feature\_extraction.text import TfidfVectorizer

#1. create a pipeline object

clf = Pipeline([

     ('vectorizer\_tfidf',TfidfVectorizer()),

     ('KNN', KNeighborsClassifier())

])

#2. fit with X\_train and y\_train

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

#3. get the predictions for X\_test and store it in y\_pred

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

#4. print the classfication report

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

import spacy

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

def preprocess(text):

    doc = nlp(text)

    filtered\_tokens = []

    for token in doc:

        if token.is\_stop or token.is\_punct:

            continue

        else:

            filtered\_tokens.append(token.lemma\_)

    return " ".join(filtered\_tokens)

# convert it into one sentence without stop words and punctuations(.,-) and also with the base words.

df['processed\_text'] = df["Description"].apply(preprocess)

df

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    df.processed\_text,

    df.label\_num,

    test\_size=0.2, # 20% samples will go to test dataset

    random\_state=2022,

    stratify=df.label\_num

)

from sklearn.neighbors import KNeighborsClassifier

clf = Pipeline([

     ('vectorizer\_tfidf',TfidfVectorizer()),        #using the ngram\_range parameter

     ('KNN', KNeighborsClassifier())

])

#2. fit with X\_train and y\_train

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

#3. get the predictions for X\_test and store it in y\_pred

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

#4. print the classfication report

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

from sklearn.naive\_bayes import MultinomialNB

clf = Pipeline([

     ('vectorizer\_tfidf',TfidfVectorizer()),        #using the ngram\_range parameter

     ('Multi NB', MultinomialNB())

])

#2. fit with X\_train and y\_train

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

#3. get the predictions for X\_test and store it in y\_pred

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

#4. print the classfication report

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

from sklearn.ensemble import RandomForestClassifier

clf = Pipeline([

     ('vectorizer\_tfidf',TfidfVectorizer()),        #using the ngram\_range parameter

     ('Random Forest', RandomForestClassifier())

])

#2. fit with X\_train and y\_train

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

#3. get the predictions for X\_test and store it in y\_pred

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

#4. print the classfication report

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

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

cm

from matplotlib import pyplot as plt

import seaborn as sn

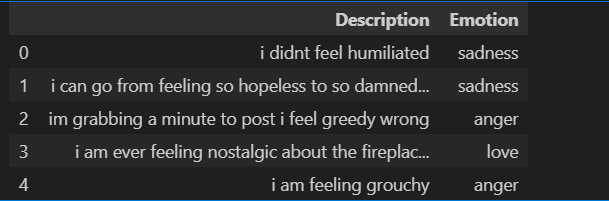
plt.figure(figsize = (10,7))

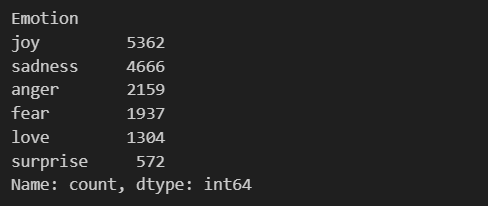
sn.heatmap(cm, annot=True, fmt='d')

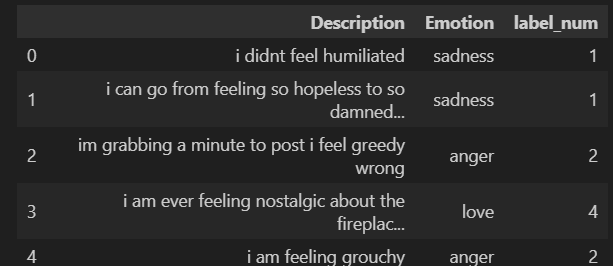
plt.xlabel('Prediction')

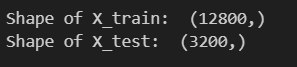
plt.ylabel('Truth')

**Output:**

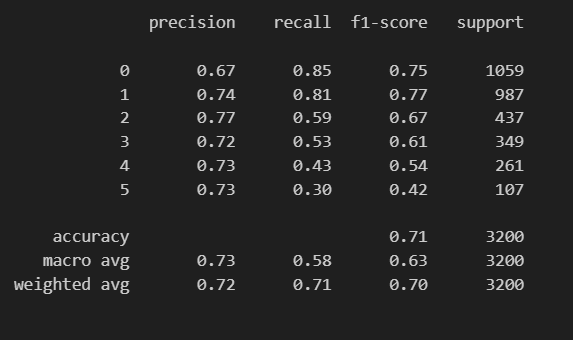
****

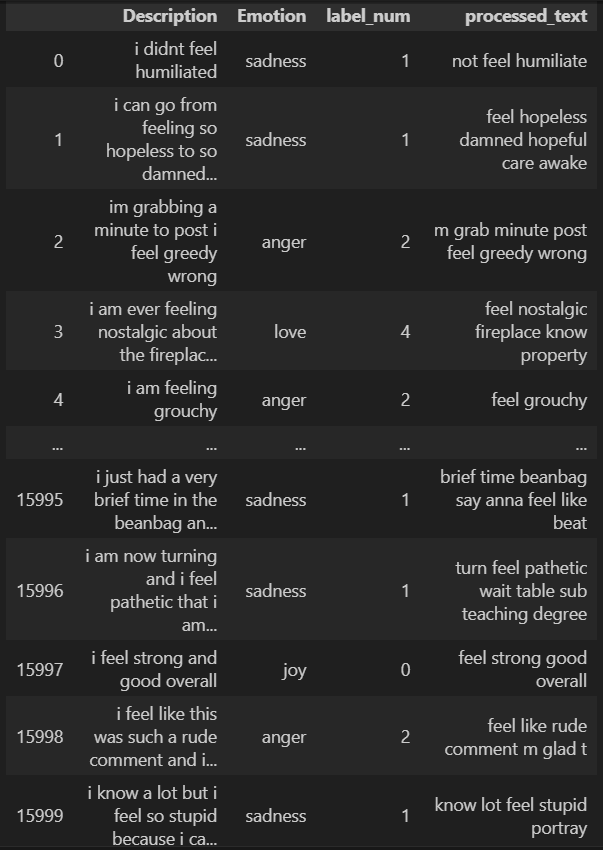
****

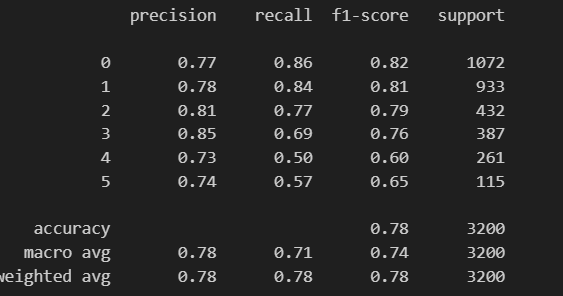
****

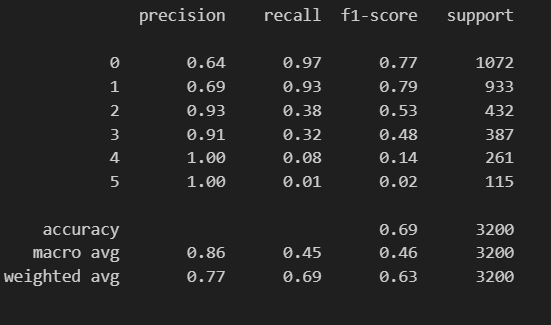
****

****

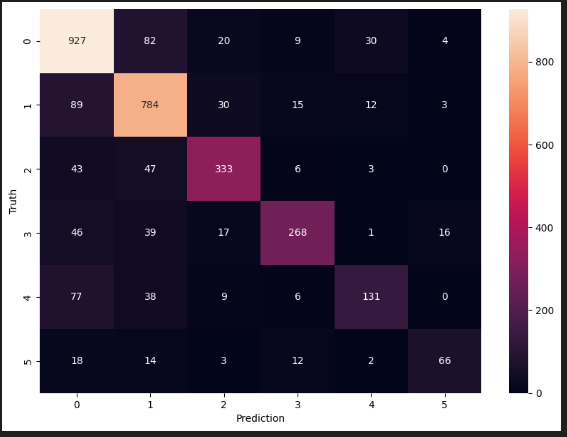
****

****

****

****

****

****

**EXPERIMENT- 07**

**Title:** Identify semantic relationship between the words from the text

**Input:**

**For words:**

import nltk

from nltk.corpus import wordnet as wn

from nltk.corpus import brown

# Initialize variables

nltk.download('wordnet')

nltk.download('brown')

brown\_freqs = dict()

N = 0

for word in brown.words():

    brown\_freqs[word] = brown\_freqs.get(word, 0) + 1

    N += 1

def get\_best\_synset\_pair(word\_1, word\_2):

    """

    Choose the pair with the highest path similarity among all pairs.

    Mimics pattern-seeking behavior of humans.

    """

    max\_sim = -1.0

    synsets\_1 = wn.synsets(word\_1)

    synsets\_2 = wn.synsets(word\_2)

    for synset\_1 in synsets\_1:

        for synset\_2 in synsets\_2:

            sim = synset\_1.path\_similarity(synset\_2)

            if sim is not None and sim > max\_sim:

                max\_sim = sim

    return max\_sim

def get\_word\_similarity(word\_1, word\_2):

    """

    Calculate the semantic similarity between two words using WordNet.

    """

    if word\_1 == word\_2:

        return 1.0

    best\_sim = get\_best\_synset\_pair(word\_1, word\_2)

    if best\_sim == -1.0:

        return 0.0

    return best\_sim

# Test the code

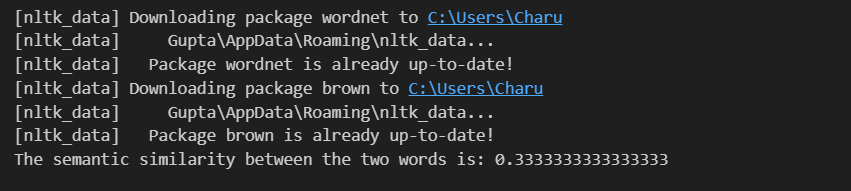
word\_1 = "happy"

word\_2 = "excited"

similarity = get\_word\_similarity(word\_1, word\_2)

print("The semantic similarity between the two words is:", similarity)

**Output:**

****

**For sentences:**

from nltk.corpus import wordnet as wn

from nltk.corpus import brown

# Initialize variables

brown\_freqs = dict()

N = 0

for word in brown.words():

    if word not in brown\_freqs:

        brown\_freqs[word] = brown\_freqs.get(word, 0) + 1

    N += 1

def get\_best\_synset\_pair(word\_1, word\_2):

    max\_sim = -1.0

    synsets\_1 = wn.synsets(word\_1)

    synsets\_2 = wn.synsets(word\_2)

    for synset\_1 in synsets\_1:

        for synset\_2 in synsets\_2:

            sim = synset\_1.path\_similarity(synset\_2)

            if sim is not None and sim > max\_sim:

                max\_sim = sim

    return max\_sim

def get\_word\_similarity(word\_1, word\_2):

    """

    Calculate the semantic similarity between two words using WordNet.

    """

    if word\_1 == word\_2:

        return 1.0

    best\_sim = get\_best\_synset\_pair(word\_1, word\_2)

    if best\_sim == -1.0:

        return 0.0

    return best\_sim

def get\_sentence\_similarity(sentence\_1, sentence\_2):

    """

    Calculate the semantic similarity between two sentences using WordNet.

    """

    words\_1 = sentence\_1.split()

    words\_2 = sentence\_2.split()

    similarity = 0.0

    total\_pairs = 0

    for word\_1 in words\_1:

        max\_sim = 0.0

        for word\_2 in words\_2:

            sim = get\_word\_similarity(word\_1, word\_2)

            if sim > max\_sim:

                max\_sim = sim

        similarity += max\_sim

        total\_pairs += 1

    if total\_pairs == 0:

        return 0.0

    return similarity / total\_pairs

# Test the code

sentence\_1 = "The dog is happy"

sentence\_2 = "The dog is excited"

similarity = get\_sentence\_similarity(sentence\_1, sentence\_2)

print("The semantic similarity between the two sentences is:", similarity)

**Output:**

****

**EXPERIMENT- 08**

**Title:** Text summarization

**Input:**

import spacy

# Load the English language model

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

# Function to generate text summary

def generate\_summary(text, num\_sentences=2):

    # Process the input text

    doc = nlp(text)

    # Get sentences with highest scores based on token ranks

    sentences = list(doc.sents)

    sentence\_scores = {sentence: sum(token.rank for token in sentence) for sentence in sentences}

    top\_sentences = sorted(sentences, key=lambda x: sentence\_scores[x], reverse=True)[:num\_sentences]

    # Join the top sentences to generate the summary

    summary = " ".join(str(sentence) for sentence in top\_sentences)

    return summary

# Input text

text = """

King Krishnadevaraya loved horses and had the best collection of horse breeds in the Kingdom. Well, one day, a trader came to the King and told him that he had brought with him a horse of the best breed in Arabia.He invited the King to inspect the horse. King Krishnadevaraya loved the horse; so the trader said that the King could buy this one and that he had two more like this one, back in Arabia that he would go back to get. The King loved the horse so much that he had to have the other two as well. He paid the trader 5000 gold coins in advance. The trader promised that he would return within two days with the other horses.Two days turned into two weeks, and still, there was no sign of the trader and the two horses. One evening, to ease his mind, the King went on a stroll in his garden. There he spotted Tenali Raman writing down something on a piece of paper. Curious, the King asked Tenali what he was jotting down.Tenali Raman was hesitant, but after further questioning, he showed the King the paper. On the paper was a list of names, the King’s being at the top of the list. Tenali said these were the names of the biggest fools in the Vijayanagara Kingdom!As expected, the King was furious that his name was at the top and asked Tenali Raman for an explanation. Tenali referred to the horse story, saying the King was a fool to believe that the trader, a stranger, would return after receiving 5000 gold coins.Countering his argument, the King then asked, what happens if/when the trader does come back? In true Tenali humour, he replied saying, in that case, the trader would be a bigger fool, and his name would replace the King’s on the list!"""

# Generate summary

summary = generate\_summary(text)

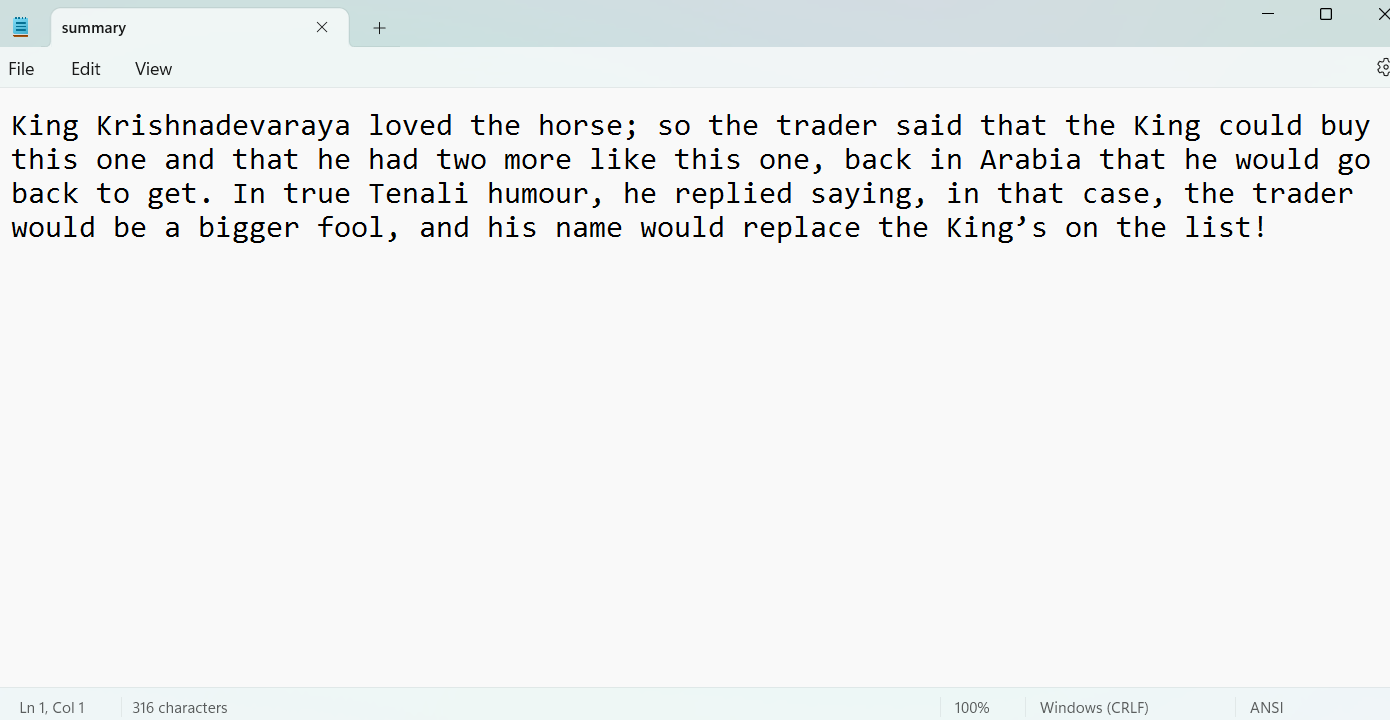
with open("summary.txt", "w") as file:

    file.write(summary)

# Print confirmation message

print("Summary saved to summary.txt")

**Output:**

****

**Experiment – 09**

**Title:**  Information Retrieval

**Input:**

import requests

# Define the username whose activity you want to fetch

username = 'Charu-2718'

# Define the endpoint to fetch the user's events

endpoint = f"https://api.github.com/users/Charu-2718/events"

# Send GET request to the API endpoint

response = requests.get(endpoint)

# Check if the request was successful (status code 200)

if response.status\_code == 200:

    # Parse the JSON response

    events = response.json()

    # Create a file to save the events

    with open('github\_events.txt', 'w') as f:

        # Write events to the file

        for event in events:

            f.write(str(event) + '\n')

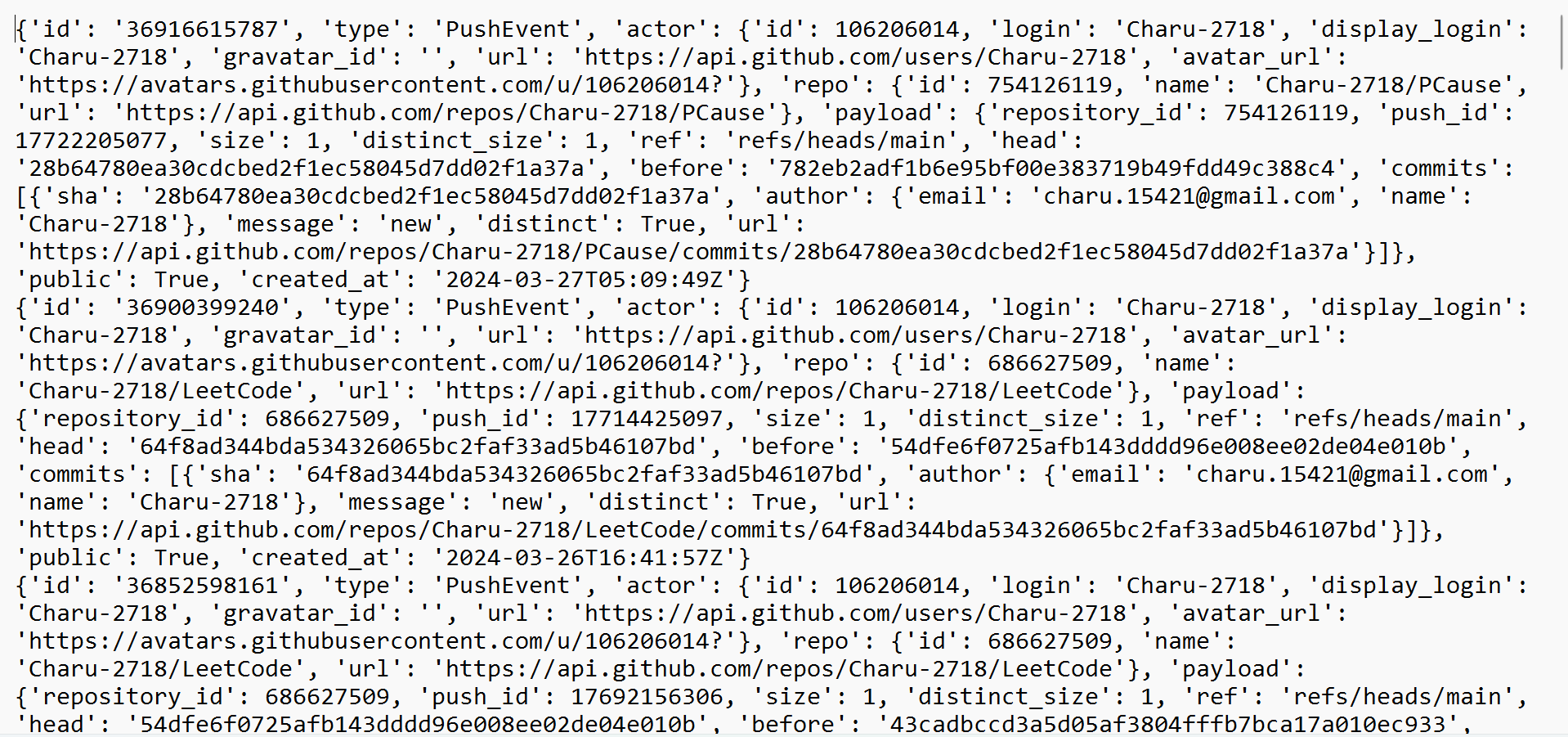
    print("Events saved successfully.")

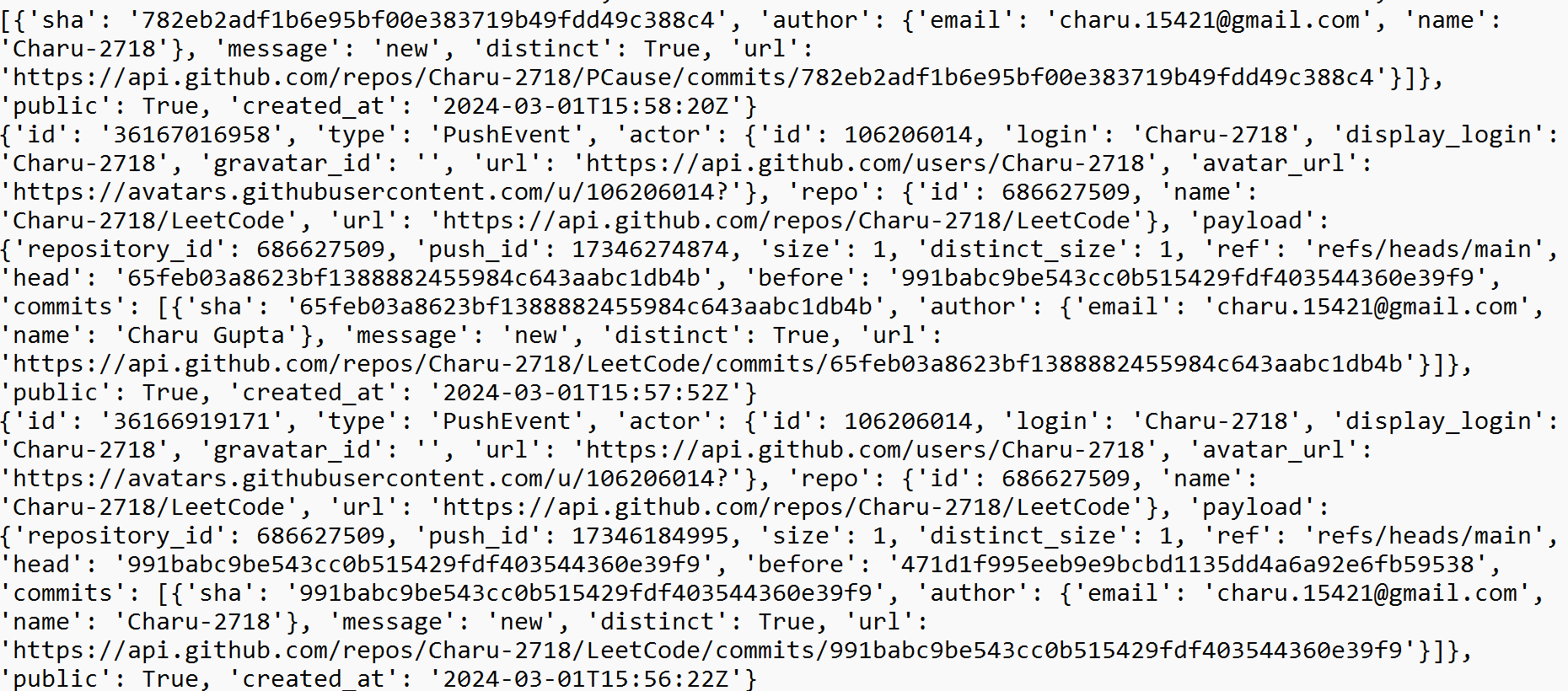
else:

    print("Failed to fetch events. Status code:", response.status\_code)

**Output:**

****

****

****

****

**EXPERIMENT- 10**

**Title:** Implement Huffmann Coding.

**Input:**

# A Huffman Tree Node

import heapq

class node:

    def \_\_init\_\_(self, freq, symbol, left=None, right=None):

        # frequency of symbol

        self.freq = freq

        # symbol name (character)

        self.symbol = symbol

        # node left of current node

        self.left = left

        # node right of current node

        self.right = right

        # tree direction (0/1)

        self.huff = ''

    def \_\_lt\_\_(self, nxt):

        return self.freq < nxt.freq

# utility function to print huffman

# codes for all symbols in the newly

# created Huffman tree

def printNodes(node, val=''):

    # huffman code for current node

    newVal = val + str(node.huff)

    # if node is not an edge node

    # then traverse inside it

    if(node.left):

        printNodes(node.left, newVal)

    if(node.right):

        printNodes(node.right, newVal)

    # if node is edge node then

    # display its huffman code

    if(not node.left and not node.right):

        print(f"{node.symbol} -> {newVal}")

# characters for huffman tree

chars = ['a', 'b', 'c', 'd', 'e', 'f']

# frequency of characters

freq = [5, 9, 12, 13, 16, 45]

# list containing unused nodes

nodes = []

# converting characters and frequencies

# into huffman tree nodes

for x in range(len(chars)):

    heapq.heappush(nodes, node(freq[x], chars[x]))

while len(nodes) > 1:

    # sort all the nodes in ascending order

    # based on their frequency

    left = heapq.heappop(nodes)

    right = heapq.heappop(nodes)

    # assign directional value to these nodes

    left.huff = 0

    right.huff = 1

    # combine the 2 smallest nodes to create

    # new node as their parent

    newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

    heapq.heappush(nodes, newNode)

# Huffman Tree

printNodes(nodes[0])

# Add this to print the total message size

total\_bits = 0

for i in range(len(chars)):

    total\_bits += freq[i] \* len(nodes[0].symbol)

print("Total message size (in bits):", total\_bits)

**Output:**

****