**#Python script for Part-of-Speech tagging and aspect-based sentiment analysis**

import matplotlib.pyplot as plt

import seaborn as sns

import nltk

from nltk.tokenize import word\_tokenize

from textblob import TextBlob

from wordcloud import WordCloud

nltk.download('punkt')

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

df['Text'] = df['clean\_tokens2'].astype(str)

# Tokenize and perform POS tagging for each row in the 'Text' column

df['POS\_tags'] = df['clean\_tokens2'].apply(lambda x: nltk.pos\_tag(word\_tokenize(x)))

# Create a DataFrame from the POS tags

df\_pos = pd.DataFrame(df['POS\_tags'].explode().tolist(), columns=['Word', 'POS'])

# Count the occurrences of each POS tag

pos\_counts = df\_pos['POS'].value\_counts()

# Create a horizontal bar plot

font = {'family': 'serif', 'weight': 'normal', 'size': 16}

plt.rc('font', \*\*font)

plt.figure(figsize= (10, 8))

ax = sns.barplot(x=pos\_counts.values, y=pos\_counts.index, color ='teal')

plt.xlabel('Count')

plt.ylabel('POS Tag')

plt.tight\_layout() # Adjust layout to prevent clipping

plt.show()

from scipy.stats import chi2\_contingency

# Calculate observed counts

observed\_counts = [pos\_counts.get(tag, 0) for tag in ['NN', 'JJ', 'RB']]

# Define expected counts assuming equal proportions under the null hypothesis

total\_count = sum(observed\_counts)

expected\_count = total\_count / 3

expected\_counts = [expected\_count] \* 3

# Perform chi-square test

chi2\_stat, p\_val, dof, expected = chi2\_contingency([observed\_counts, expected\_counts])

# Print chi-square statistic and p-value

print(f"Chi-square statistic: {chi2\_stat}")

print(f"P-value: {p\_val}")

# Initialize lists to store words based on sentiment

positive\_words = []

negative\_words = []

neutral\_words = []

# Extract aspects (nouns) from POS tags column

aspects = [word for word, pos in df['POS\_tags'].explode().tolist() if pos.startswith('JJ')]

# Perform sentiment analysis for each aspect

for aspect in aspects:

aspect\_textblob = TextBlob(aspect)

aspect\_sentiment = aspect\_textblob.sentiment.polarity

# Append word to corresponding list based on sentiment score

if aspect\_sentiment > 0:

positive\_words.append(aspect)

elif aspect\_sentiment < 0:

negative\_words.append(aspect)

else:

neutral\_words.append(aspect)

# Convert lists to strings

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

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

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

# Generate word clouds

wordcloud\_positive = WordCloud(background\_color='white').generate(positive\_text)

wordcloud\_negative = WordCloud(background\_color='white').generate(negative\_text)

wordcloud\_neutral = WordCloud(background\_color='white').generate(neutral\_text)

# Plot word cloud for positive words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_positive, interpolation='bilinear')

plt.title('Positive Words')

plt.axis('off')

plt.show()

# Plot word cloud for negative words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_negative, interpolation='bilinear')

plt.title('Negative Words')

plt.axis('off')

plt.show()

# Plot word cloud for neutral words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_neutral, interpolation='bilinear')

plt.title('Neutral Words')

plt.axis('off')

plt.show()

# Initialize lists to store words based on sentiment

positive\_words = []

negative\_words = []

neutral\_words = []

# Extract aspects (nouns) from POS tags column

aspects = [word for word, pos in df['POS\_tags'].explode().tolist() if pos.startswith('NN')]

# Perform sentiment analysis for each aspect

for aspect in aspects:

aspect\_textblob = TextBlob(aspect)

aspect\_sentiment = aspect\_textblob.sentiment.polarity

# Append word to corresponding list based on sentiment score

if aspect\_sentiment > 0:

positive\_words.append(aspect)

elif aspect\_sentiment < 0:

negative\_words.append(aspect)

else:

neutral\_words.append(aspect)

# Convert lists to strings

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

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

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

# Generate word clouds

wordcloud\_positive = WordCloud(background\_color='white').generate(positive\_text)

wordcloud\_negative = WordCloud(background\_color='white').generate(negative\_text)

wordcloud\_neutral = WordCloud(background\_color='white').generate(neutral\_text)

# Plot word cloud for positive words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_positive, interpolation='bilinear')

plt.title('Positive Words')

plt.axis('off')

plt.show()

# Plot word cloud for negative words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_negative, interpolation='bilinear')

plt.title('Negative Words')

plt.axis('off')

plt.show()

# Plot word cloud for neutral words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_neutral, interpolation='bilinear')

plt.title('Neutral Words')

plt.axis('off')

plt.show()

# Initialize lists to store words based on sentiment

positive\_words = []

negative\_words = []

neutral\_words = []

# Extract aspects (nouns) from POS tags column

aspects = [word for word, pos in df['POS\_tags'].explode().tolist() if pos.startswith('RB')]

# Perform sentiment analysis for each aspect

for aspect in aspects:

aspect\_textblob = TextBlob(aspect)

aspect\_sentiment = aspect\_textblob.sentiment.polarity

# Append word to corresponding list based on sentiment score

if aspect\_sentiment > 0:

positive\_words.append(aspect)

elif aspect\_sentiment < 0:

negative\_words.append(aspect)

else:

neutral\_words.append(aspect)

# Convert lists to strings

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

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

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

# Generate word clouds

wordcloud\_positive = WordCloud(background\_color='white').generate(positive\_text)

wordcloud\_negative = WordCloud(background\_color='white').generate(negative\_text)

wordcloud\_neutral = WordCloud(background\_color='white').generate(neutral\_text)

# Plot word cloud for positive words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_positive, interpolation='bilinear')

plt.title('Positive Words')

plt.axis('off')

plt.show()

# Plot word cloud for negative words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_negative, interpolation='bilinear')

plt.title('Negative Words')

plt.axis('off')

plt.show()

# Plot word cloud for neutral words

plt.figure(figsize=(6, 8))

plt.imshow(wordcloud\_neutral, interpolation='bilinear')

plt.title('Neutral Words')

plt.axis('off')

plt.show()

**Python script for performing a frequency analysis**

Import nltk

from nltk import FreqDist

from wordcloud import WordCloud

import seaborn as sns

import matplotlib.pyplot as plt

text = ' '.join(df['clean\_tokens2'].values)

tokens = nltk.word\_tokenize(text)

fdist = FreqDist(tokens)

freq\_df = pd.DataFrame(fdist.most\_common(), columns=['Term', 'Frequency'])

print(freq\_df)

text\_data = ''.join(df["clean\_tokens2"].values)

wordcloud = WordCloud(width=800, height=400, background\_color="white").generate(text\_data)

plt.figure(figsize=(10, 8))

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

plt.axis("off")

plt.show()

font = {'family': 'serif', 'weight': 'normal', 'size': 14}

plt.rc('font', \*\*font)

common\_words = fdist.most\_common(30)

terms, frequencies = zip(\*common\_words)

# Create a horizontal bar plot

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

bars = plt.barh(terms, frequencies, color="blue")

plt.xlabel('Frequency count')

plt.ylabel('Terms')

plt.gca().spines['top'].set\_visible(False)

plt.gca().spines['right'].set\_visible(False)

plt.show()

# Exclude the top 3 most frequent words

top\_words = dict(fdist.most\_common()[1:31]) # Excluding the top 3

# Create a WordCloud

wordcloud = WordCloud(width=800, height=400, background\_color="white").generate\_from\_frequencies(top\_words)

# Display the WordCloud

plt.figure(figsize=(10, 5))

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

plt.axis("off")

# Convert the word frequencies to a matrix for clustering

common\_words = dict(fdist.most\_common()[1:31])

terms, frequencies = list(common\_words.keys()), list(common\_words.values())

word\_frequencies = [[freq] for freq in frequencies]

# Perform hierarchical clustering

from sklearn.feature\_extraction.text import TfidfVectorizer

from scipy.cluster.hierarchy import linkage, dendrogram

linkage\_matrix = linkage(word\_frequencies, method='ward')

# Plot the dendrogram

plt.figure(figsize=(18, 10))

dendrogram(linkage\_matrix, labels=terms, orientation='top')

plt.xlabel('Top 30 Words')

plt.ylabel('Linkage Distance')

plt.gca().spines['top'].set\_visible(False)

plt.gca().spines['right'].set\_visible(False)

plt.xticks(rotation=35)

plt.yticks(range(0, int(linkage\_matrix[:, 2].max()) + 1, 250)) # Adjust y-axis ticks to have intervals of 250

plt.show()

#COOCCURENCE PLOT

import networkx as nx

from collections import Counter

from itertools import combinations

from nltk.tokenize import word\_tokenize

from matplotlib.cm import ScalarMappable

corpus = df["clean\_tokens2"].apply(word\_tokenize) # Tokenize each sentence in the corpus

# Calculate word frequency

fdist = Counter()

for sentence in corpus:

fdist.update(sentence)

# Get the most common terms

words\_wn = fdist.most\_common(30)

common\_terms = [word for word, \_ in words\_wn]

# Create the term network

term\_network = nx.Graph()

term\_network.add\_nodes\_from(common\_terms)

# Calculate co-occurrences

co\_occurrences = Counter()

for sentence in corpus:

combinations\_list = list(combinations(set(sentence), 2))

co\_occurrences.update(combinations\_list)

for (term1, term2), weight in co\_occurrences.items():

if term1 in common\_terms and term2 in common\_terms:

term\_network.add\_edge(term1, term2, weight=weight)

# Create the plot

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

pos = nx.spring\_layout(term\_network) # Position nodes using a spring layout algorithm

labels = nx.get\_edge\_attributes(term\_network, 'weight')

edge\_widths = [0.01 \* term\_network[u][v]['weight'] for u, v in term\_network.edges()]

# Calculate edge colors based on edge weights

edge\_colors = edge\_widths

nx.draw(term\_network, pos, with\_labels=True, node\_color='green', font\_size=14, node\_size=8, width=edge\_widths, edge\_color=edge\_colors)

sm = plt.cm.ScalarMappable(cmap=plt.cm.Blues, norm=plt.Normalize(vmin=min(edge\_widths),

vmax=max(edge\_widths)))

sm.set\_array([])

plt.colorbar(sm)

plt.show()