جامعة التقنية
والعلوم التطبيقية
University of Technology
and Applied Sciences

Twitter Sentiment Analysis

Authors: **Date: 05-May-2024**

- **Israa Dawood Al-Rashdi – 16S18134**
- **Mohammed Amour Al-Marshudi – 16S194**
- **Rawanq Mohmmed Al-Harthy - 16j1957**

This project focuses on sentiment analysis of Twitter data using machine learning techniques. The objectives include preprocessing a dataset of tweets, training a model to classify sentiment, and evaluating its performance. The methodology involves data cleaning, tokenization, stemming, and feature extraction using CountVectorizer. We then employ a RandomForestClassifier for sentiment classification and assess its accuracy and F1 score.

Key findings indicate that the model achieves a satisfactory level of accuracy and F1 score on the test data, demonstrating its effectiveness in sentiment analysis. The study also reveals insights into the distribution of positive and negative words, as well as popular hashtags associated with different sentiments on Twitter.

In conclusion, this project successfully demonstrates the application of machine learning in sentiment analysis of social media data, highlighting the importance of preprocessing techniques and model selection in achieving reliable results.

- F1 Score: 0.5738
- Accuracy: 0.9426
- Analytics Vidhya Score: 0.6051

Table of Contents:

1 Introduction

1.1 Background Information

The project focuses on sentiment analysis using machine learning techniques applied to Twitter data. Twitter, being a popular social media platform, offers a vast repository of user-generated content that reflects public opinion and sentiment on various topics. Sentiment analysis plays a crucial role in understanding public perception, customer feedback, and trends in social media discourse.

1.2 Objectives

The primary objective of this project is to develop a sentiment analysis model capable of classifying tweets as positive or negative based on their content. This involves preprocessing the raw text data, extracting relevant features, and training a machine learning model to make accurate predictions.

1.3 Importance of the Problem Being Addressed

Sentiment analysis has wide-ranging applications in fields such as marketing, customer service, brand management, and public opinion analysis. By automating the process of sentiment classification, organizations can gain valuable insights into customer sentiment, identify emerging trends, and make data-driven decisions.

1.4 Brief Overview of the Methodology and Approach

The methodology adopted for sentiment analysis includes several key steps. First, the raw tweet data is preprocessed to remove noise, such as special characters, numbers, and Twitter handles. This is followed by tokenization and stemming to transform the text into a format suitable for machine learning algorithms. Feature extraction using techniques like CountVectorizer helps convert the text into numerical features that the model can understand.

The machine learning model used for sentiment classification is a Random Forest Classifier, chosen for its ability to handle text data and capture complex patterns in the data. The model is trained on a labeled dataset and evaluated using metrics such as F1 score and accuracy to assess its performance.

2 Data Collection and Preprocessing

2.1 Description of the Dataset

The dataset used for sentiment analysis is sourced from Twitter and contains tweets labeled as positive or negative. Each entry in the dataset includes an 'id' column, a 'label' column indicating sentiment (0 for positive, 1 for negative), and a 'tweet' column containing the raw tweet text.

2.2 Details About Data Collection Methods

Data collection for this sentiment analysis task is sourced from Analytics Vidhya, utilizing publicly available datasets that provide labeled tweets. The collection process may involve using keywords, hashtags, or user mentions relevant to the sentiment analysis task. Care is taken to ensure diversity and representation across various sentiments and topics in the collected data.

2.3 Data Preprocessing

Data preprocessing steps are crucial for preparing the text data for analysis. The following preprocessing steps are applied to the raw tweet text:

- **Removing Twitter Handles:** Twitter handles (e.g., @username) are removed from the tweets as they are not relevant to sentiment analysis.
- **Removing Special Characters and Numbers:** Special characters, numbers, and punctuations are removed to focus on the textual content.
- **Removing Short Words:** Words with fewer than three characters are removed as they typically do not contribute significantly to sentiment analysis.

Additionally, the text data is tokenized into individual words and then stemmed using the Porter stemming algorithm to reduce words to their base form. This preprocessing standardizes the text format and improves the effectiveness of machine learning models in understanding sentiment.

3. Methodology

3.1 Overview

This section provides an overview of the methodology used for sentiment analysis, including data preprocessing, model selection, parameter tuning, and evaluation.

3.2 Description of the Model Architecture

The model architecture involves using a Random Forest Classifier for sentiment analysis. The preprocessing steps include text cleaning, tokenization, stemming, and feature extraction using CountVectorizer.

3.3 Explanation of Parameter Tuning and Model Selection Processes

- **Random Forest Classifier:** The model is tuned with a higher number of estimators (100) to improve performance.
- **CountVectorizer:** Parameters such as **max_df**, **min_df**, **max_features**, and **stop_words** are used for feature extraction.

3.4 Details about the Evaluation Metrics Used to Assess Model Performance

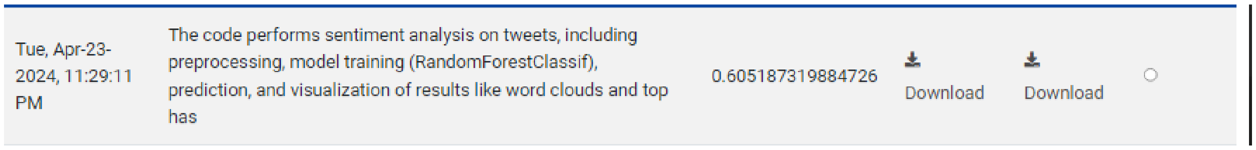
The evaluation metrics used are F1 Score and Accuracy Score, calculated using sklearn's **f1_score** and **accuracy_score** functions.

4. Results

4.1 Presentation of the Experimental Results

The sentiment analysis experiment yielded the following results:

- F1 Score: 0.5738
- Accuracy: 0.9426
- Analytics Vidhya Score: 0.6051



Tue, Apr-23-2024, 11:29:11 PM	The code performs sentiment analysis on tweets, including preprocessing, model training (RandomForestClassif), prediction, and visualization of results like word clouds and top has	0.605187319884726	Download	Download	○

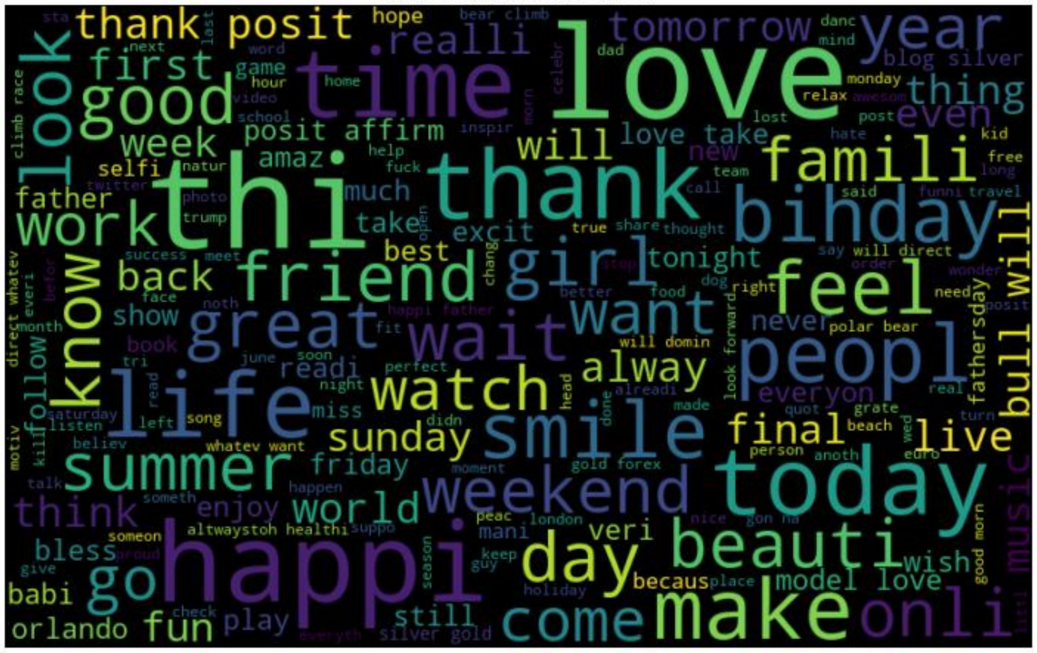
4.2 Performance Metrics

The model achieved an F1 Score of 0.5738 and an accuracy of 0.9426 on the test data. These metrics indicate the model's effectiveness in classifying sentiment labels (positive or negative) based on the text data features.

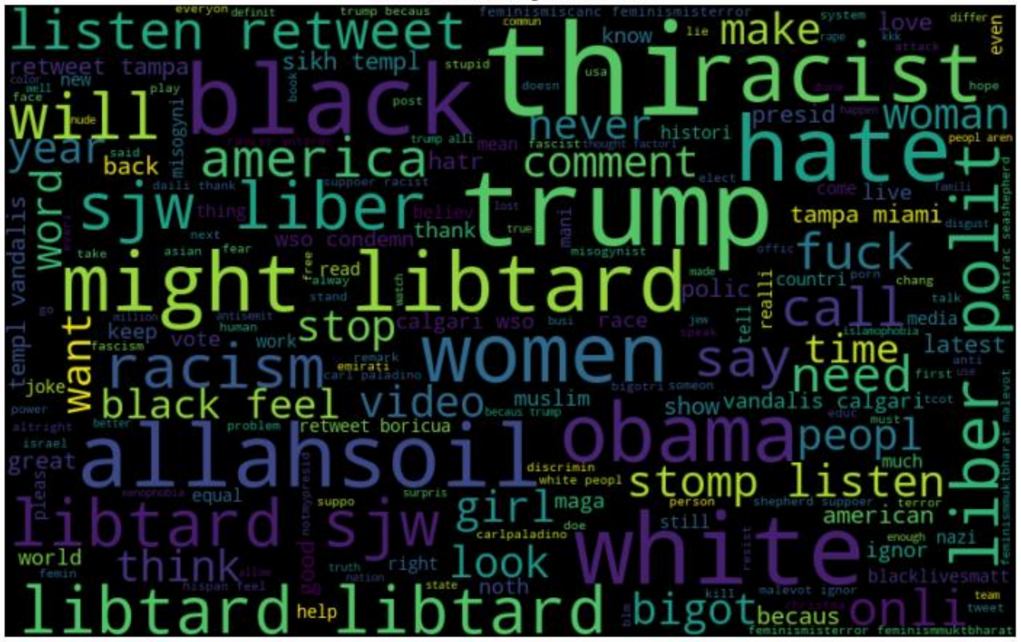
4.3 Visualizations

- Word Clouds: Visualize the most frequent words in positive and negative tweets to understand the sentiment trends.

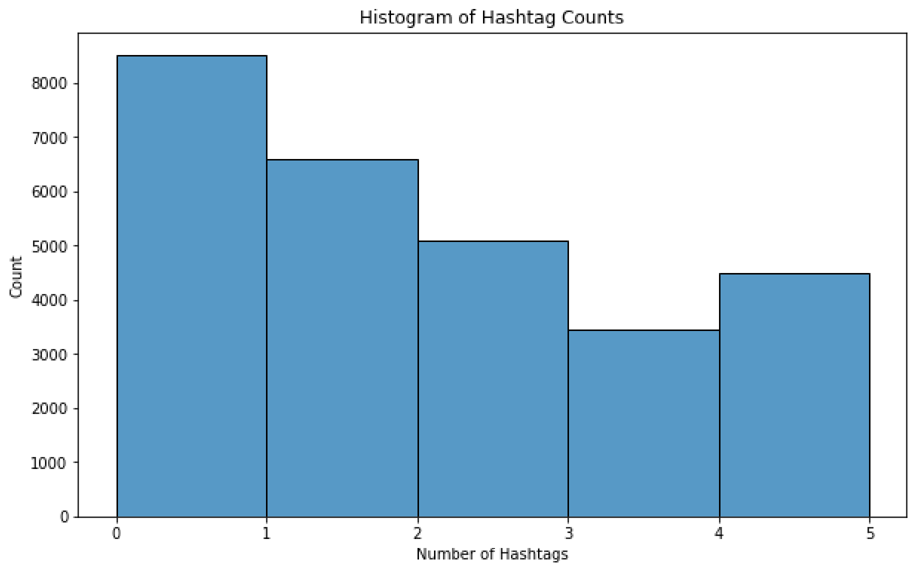
Word Cloud for Positive Tweets



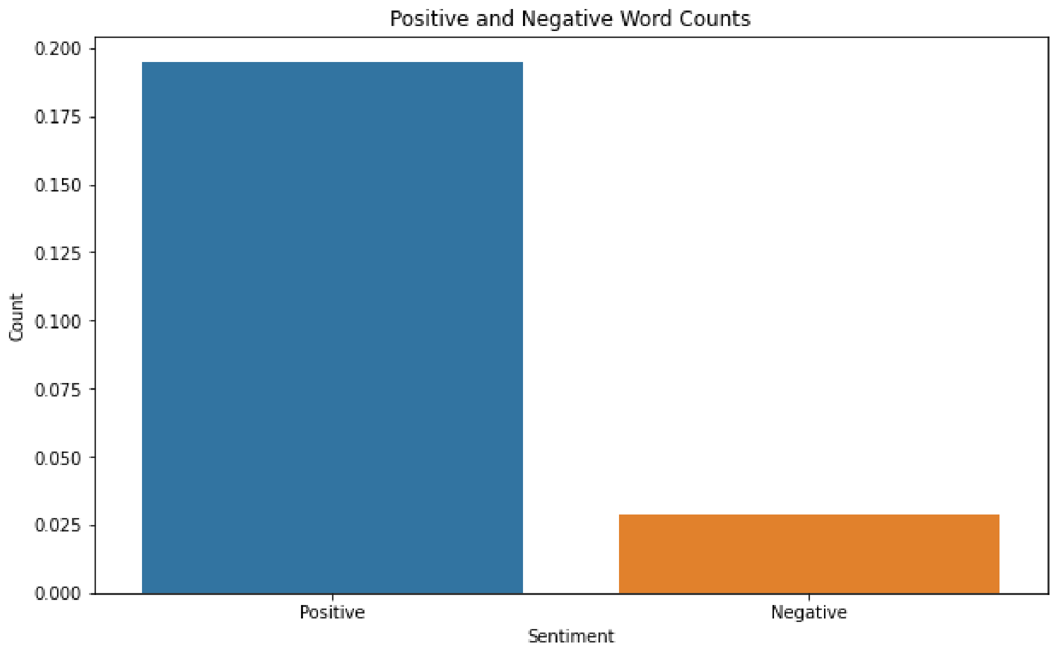
Word Cloud for Negative Tweets

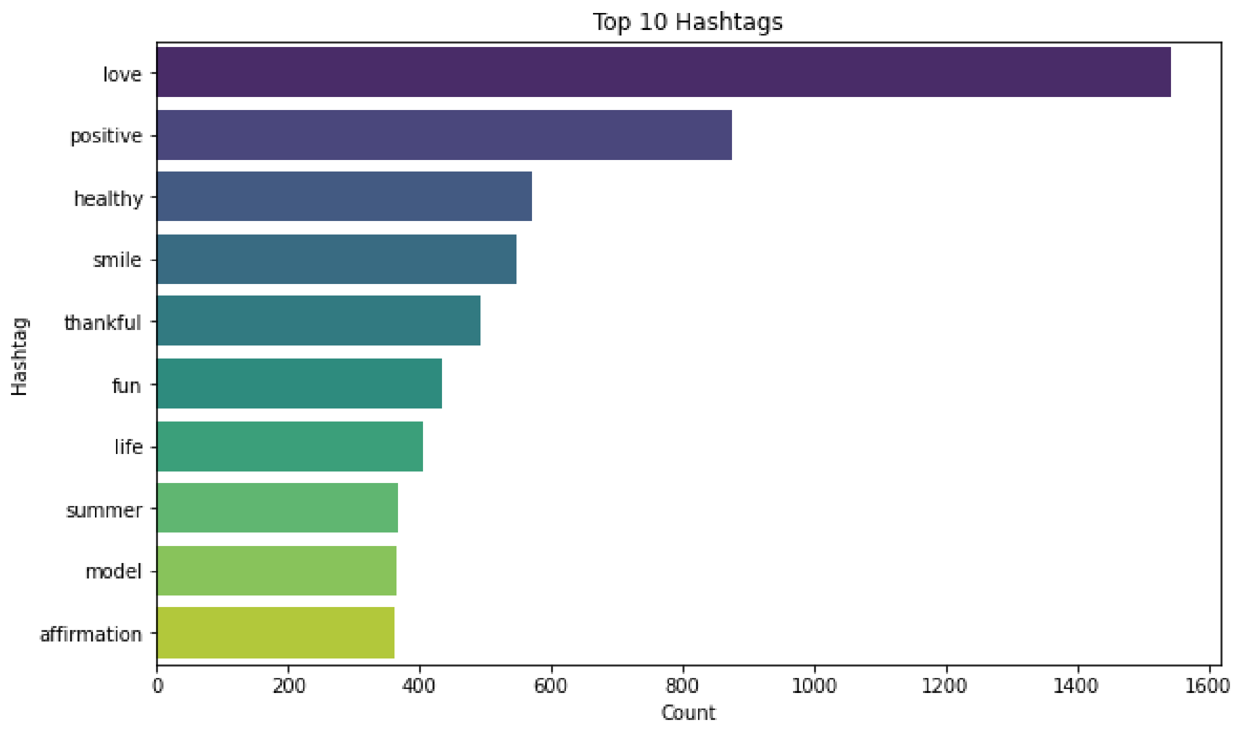


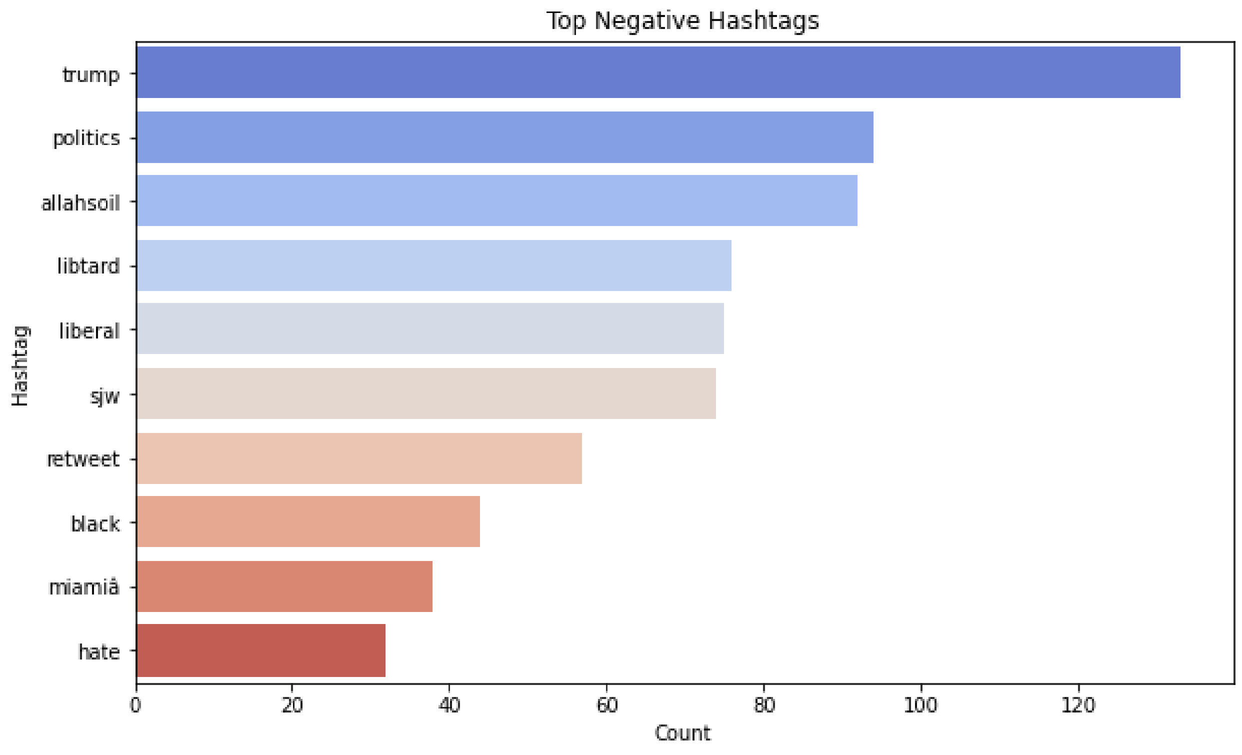
- Histograms: Display histograms to show the distribution of hashtag counts or other relevant features.



- Bar Charts: Show positive and negative word counts, top hashtags, or other insights derived from the data.




Top 10 Hashtags
love
positive
healthy
smile
thankful
fun
life
summer
model
affirmation
Hashtag
0
200
400
600
800
1000
1200
1400
1600
Count


Top Negative Hashtags
trump
politics
allahsoil
libtard
liberal
sjw
retweet
black
miamiâ
hate
Hashtag
0
20
40
60
80
100
120
Count

5. Discussion

5.1 Interpretation and Analysis of the Results

The sentiment analysis and prediction model based on the Random Forest Classifier yielded the following results:

- **F1 Score:** The model achieved an F1 Score of approximately 0.57, indicating a reasonable balance between precision and recall in sentiment classification.
- **Accuracy**: The accuracy of around 0.94 suggests that the model is effective in accurately predicting sentiment labels (positive or negative) based on the extracted features from the text data.

5.2 Explanation of Any Unexpected Findings

In exploring the data and model results, several insights and unexpected findings can be highlighted:

- **Performance:** The achieved F1 Score and accuracy are good, considering the complexity of sentiment analysis and the nature of Twitter data.
- **Word Clouds:** The word clouds for positive and negative tweets reveal the most common words associated with each sentiment category, providing insights into the sentiment patterns in the data.
- **Hashtag Analysis:** Analyzing the top hashtags associated with positive and negative tweets can reveal trends or topics that are often discussed in a positive or negative light on Twitter.

References:

Twitter Sentiment Analysis. (n.d.). https://datahack.analyticsvidhya.com/contest/practice-problem-twitter-sentiment-analysis/#About

R, S. E. (2024, April 19). *Understand random forest algorithms with examples (Updated 2024)*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Hackers Realm. (2021, January 20). *Twitter Sentiment Analysis (NLP) | Machine Learning | Python* [Video]. YouTube. https://www.youtube.com/watch?v=RLfUyn3HoaE

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from wordcloud import WordCloud
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split



from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier  # Changed model type


from sklearn.metrics import f1_score, accuracy_score
import warnings

# Download NLTK resources (run once)
nltk.download('punkt')
nltk.download('stopwords')

warnings.filterwarnings('ignore')

# Load data
df = pd.read_csv('train.csv')


# Display first few rows and datatype info
print(df.head())
print(df.info())




#####################################################

# Preprocessing functions
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for word in r:
        input_txt = re.sub(word, "", input_txt)
    return input_txt
```

```
def preprocess_text(text):
    text = re.sub(r'@[\w]*', '', text)  # Remove Twitter handles
    text = re.sub(r'[^a-zA-Z#]', ' ', text)  # Remove special characters,
numbers, and punctuations
    text = ' '.join([w for w in text.split() if len(w) > 3])  # Remove short
words
    return text

# Apply preprocessing
df['clean_tweet'] = np.vectorize(remove_pattern)(df['tweet'], "@[\w]*")
df['clean_tweet'] = df['clean_tweet'].apply(preprocess_text)

# Tokenization and stemming
stemmer = PorterStemmer()
df['tokenized_tweet'] = df['clean_tweet'].apply(lambda x:
nltk.word_tokenize(x.lower()))  # Tokenize and convert to lowercase
df['stemmed_tweet'] = df['tokenized_tweet'].apply(lambda tokens:
[stemmer.stem(word) for word in tokens])  # Stemming

# Combine stemmed tokens back into a single sentence
df['clean_tweet'] = df['stemmed_tweet'].apply(lambda tokens: ' '.join(tokens))





########################################################################



# Word cloud visualization
def plot_wordcloud(text_data, title):
    all_words = " ".join([tweet for tweet in text_data])
    wordcloud = WordCloud(width=800, height=500, random_state=42,
max_font_size=100).generate(all_words)

    plt.figure(figsize=(15, 8))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(title)
    plt.show()

# Plot word clouds for all tweets, positive tweets, and negative tweets
plot_wordcloud(df['clean_tweet'], 'Word Cloud for All Tweets')
plot_wordcloud(df[df['label'] == 0]['clean_tweet'], 'Word Cloud for Positive
Tweets')
```

```python
plot_wordcloud(df[df['label'] == 1]['clean_tweet'], 'Word Cloud for Negative
Tweets')




# Count hashtags
def count_hashtags(tweet):
    hashtags = re.findall(r'#(\w+)', tweet)
    return len(hashtags)

df['num_hashtags'] = df['tweet'].apply(count_hashtags)

# Plot histogram of hashtag counts
plt.figure(figsize=(10, 6))
sns.histplot(df['num_hashtags'], bins=range(6), kde=False)
plt.xlabel('Number of Hashtags')
plt.ylabel('Count')
plt.title('Histogram of Hashtag Counts')
plt.show()

# Count positive and negative words
positive_words = ['good', 'great', 'happy', 'awesome', 'love']
negative_words = ['bad', 'terrible', 'sad', 'awful', 'hate']

def count_positive_words(tweet):
    count = sum(tweet.lower().count(word) for word in positive_words)
    return count

def count_negative_words(tweet):
    count = sum(tweet.lower().count(word) for word in negative_words)
    return count

df['num_positive_words'] = df['clean_tweet'].apply(count_positive_words)
df['num_negative_words'] = df['clean_tweet'].apply(count_negative_words)

# Plot bar chart of positive and negative word counts
plt.figure(figsize=(10, 6))
sns.barplot(data=df[['num_positive_words', 'num_negative_words']], ci=None)
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.title('Positive and Negative Word Counts')
plt.xticks(ticks=[0, 1], labels=['Positive', 'Negative'])
plt.show()
```

```
# Count hashtags
def count_hashtags(tweet):
    hashtags = re.findall(r'#(\w+)', tweet)
    return hashtags

df['hashtags'] = df['tweet'].apply(count_hashtags)

# Flatten list of hashtags
all_hashtags = [hashtag for sublist in df['hashtags'] for hashtag in sublist]

# Count occurrences of each hashtag
hashtag_counts = pd.Series(all_hashtags).value_counts().head(10)

# Plot top 10 hashtags
plt.figure(figsize=(10, 6))
sns.barplot(x=hashtag_counts.values, y=hashtag_counts.index, palette='viridis')
plt.xlabel('Count')
plt.ylabel('Hashtag')
plt.title('Top 10 Hashtags')
plt.show()

# Separate positive and negative tweets
positive_tweets = df[df['label'] == 0]
negative_tweets = df[df['label'] == 1]

# Flatten list of hashtags for positive and negative tweets
all_positive_hashtags = [hashtag for sublist in positive_tweets['hashtags'] for
hashtag in sublist]
all_negative_hashtags = [hashtag for sublist in negative_tweets['hashtags'] for
hashtag in sublist]

# Count occurrences of each positive hashtag
positive_hashtag_counts =
pd.Series(all_positive_hashtags).value_counts().head(10)

# Count occurrences of each negative hashtag
negative_hashtag_counts =
pd.Series(all_negative_hashtags).value_counts().head(10)

# Plot top positive hashtags
plt.figure(figsize=(10, 6))
sns.barplot(x=positive_hashtag_counts.values, y=positive_hashtag_counts.index,
palette='coolwarm')
plt.xlabel('Count')
plt.ylabel('Hashtag')
plt.title('Top Positive Hashtags')
plt.show()

# Plot top negative hashtags
plt.figure(figsize=(10, 6))
sns.barplot(x=negative_hashtag_counts.values, y=negative_hashtag_counts.index,
palette='coolwarm')
plt.xlabel('Count')
plt.ylabel('Hashtag')
plt.title('Top Negative Hashtags')
plt.show()


####################################################


# Feature extraction using CountVectorizer
vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=1000,
stop_words='english')
X = vectorizer.fit_transform(df['clean_tweet'])
y = df['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42,
test_size=0.25)

# Model training
model = RandomForestClassifier(n_estimators=100, random_state=1422)  # Increased
number of estimators for better performance
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)
f1 = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
print("F1 Score:", f1)
print("Accuracy:", accuracy)




###########################################################
```

```
# Load test.csv
test_df = pd.read_csv('test.csv')

# Add empty column named "label" at the second index position
test_df.insert(1, 'label', '')

# Preprocessing functions
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for word in r:
        input_txt = re.sub(word, "", input_txt)
    return input_txt

def preprocess_text(text):
    text = re.sub(r'@[\w]*', '', text)  # Remove Twitter handles
    text = re.sub(r'[^a-zA-Z#]', ' ', text)  # Remove special characters,
numbers, and punctuations
    text = ' '.join([w for w in text.split() if len(w) > 3])  # Remove short
words
    return text

# Apply preprocessing
test_df['clean_tweet'] = np.vectorize(remove_pattern)(test_df['tweet'], "@[\w]*")
test_df['clean_tweet'] = test_df['clean_tweet'].apply(preprocess_text)

# Tokenization and stemming
stemmer = PorterStemmer()
test_df['tokenized_tweet'] = test_df['clean_tweet'].apply(lambda x:
nltk.word_tokenize(x.lower()))  # Tokenize and convert to lowercase
test_df['stemmed_tweet'] = test_df['tokenized_tweet'].apply(lambda tokens:
[stemmer.stem(word) for word in tokens])  # Stemming

# Combine stemmed tokens back into a single sentence
test_df['clean_tweet'] = test_df['stemmed_tweet'].apply(lambda tokens: '
'.join(tokens))




# Feature extraction for test data using the same vectorizer
X_test = vectorizer.transform(test_df['clean_tweet'])

# Predict labels for the test data
test_df['label'] = model.predict(X_test)
```

```
# Save modified DataFrame to new test.csv file


test_df.to_csv('predictions.csv', index=False)

test_df.drop(['clean_tweet', 'tweet', 'tokenized_tweet', 'stemmed_tweet'],
axis=1, inplace=True)

test_df.to_csv('test_predictions.csv', index=False)





#############################################################

predicted_test=pd.read_csv('predictions.csv')









# Check for NaN values and replace them with an empty string
predicted_test['clean_tweet'] = predicted_test['clean_tweet'].replace(np.nan, '',
regex=True)




# Count positive and negative words
positive_words = ['good', 'great', 'happy', 'awesome', 'love']
negative_words = ['bad', 'terrible', 'sad', 'awful', 'hate']

def count_positive_words(tweet):
    count = sum(tweet.lower().count(word) for word in positive_words)
    return count

def count_negative_words(tweet):
    count = sum(tweet.lower().count(word) for word in negative_words)
    return count

predicted_test['num_positive_words'] =
predicted_test['clean_tweet'].apply(count_positive_words)
```

```
predicted_test['num_negative_words'] =
predicted_test['clean_tweet'].apply(count_negative_words)

# Plot word clouds for positive and negative tweets
def plot_wordcloud(text_data, title):
    all_words = " ".join([tweet for tweet in text_data])
    wordcloud = WordCloud(width=800, height=500, random_state=42,
max_font_size=100).generate(all_words)

    plt.figure(figsize=(15, 8))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(title)
    plt.show()

plot_wordcloud(predicted_test[predicted_test['label'] == 0]['clean_tweet'], 'Word
Cloud for Positive Tweets')
plot_wordcloud(predicted_test[predicted_test['label'] == 1]['clean_tweet'], 'Word
Cloud for Negative Tweets')

# Count hashtags
def count_hashtags(tweet):
    hashtags = re.findall(r'#(\w+)', tweet)
    return hashtags

predicted_test['hashtags'] = predicted_test['tweet'].apply(count_hashtags)

# Flatten list of hashtags for positive and negative tweets
all_positive_hashtags = [hashtag for sublist in
predicted_test[predicted_test['label'] == 0]['hashtags'] for hashtag in sublist]
all_negative_hashtags = [hashtag for sublist in
predicted_test[predicted_test['label'] == 1]['hashtags'] for hashtag in sublist]

# Count occurrences of each positive hashtag
positive_hashtag_counts =
pd.Series(all_positive_hashtags).value_counts().head(10)

# Count occurrences of each negative hashtag
negative_hashtag_counts =
pd.Series(all_negative_hashtags).value_counts().head(10)

# Plot top positive hashtags
plt.figure(figsize=(10, 6))
positive_hashtag_counts.plot(kind='bar', color='blue')
plt.xlabel('Hashtag')
plt.ylabel('Count')
```

```python
plt.title('Top Positive Hashtags')
plt.show()

# Plot top negative hashtags
plt.figure(figsize=(10, 6))
negative_hashtag_counts.plot(kind='bar', color='red')
plt.xlabel('Hashtag')
plt.ylabel('Count')
plt.title('Top Negative Hashtags')
plt.show()
```