**Twitter Sentiment Analysis**

**Data Exploration**: To overview the dataset download the dataset from Kaggle and load it using libraries like pandas. Code looks like

🡪import pandas as pd

df = pd.read\_csv(‘sentiment140.csv’, encoding = ‘latin-1’)

Review the dataset’s structure using .info() and .head() to understand features and their types. Identify key variables using three types. Tweet content, sentiment labels, timestamp. Timestamp is the dataset might not contain a timestamp if it’s not in the provided dataset.

**Data Cleaning:** Check for and handle any missing values.

🡪print(df.isnull().sum())

Remove duplicate entries if any. 🡪df = df.drop\_duplicates()

Review and clean irrelevant information based on your objectives.

**Exploratory Data Analysis (EDA ) :** Use descriptive statistics to understand the distribution of sentiment labels and tweet lengths.

🡪print(df[‘sentiment’].value\_counts())

For visualizations we use histograms and word clouds. Histograms is to show sentiment distributions and word clouds are for visualizing frequent terms.

🡪from wordcloud import WordCloud

Import matplotlib.pyplot as plt

text = ‘ ‘.jion(df[‘text’])

wordcloud = WordCloud().generate(text)

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

plt.axis(‘off’)

plt.show()

**Sentiment Distribution :** Plot the distribution of sentiment labels to visualize balance.

🡪import seaborn as sns

sns.countplot(data=df, x='sentiment')

plt.xlabel('Sentiment')

plt.ylabel('Count')

plt.title('Distribution of Sentiments')

plt.show()

**Word Frequency Analysis**

Below is the code for frequency analysis:

🡪from collections import Counter

import nltk

nltk.download('punkt')

from nltk.tokenize import word\_tokenize

all\_words = ' '.join(df['text'])

words = word\_tokenize(all\_words)

word\_freq = Counter(words)

# Most common words

common\_words = word\_freq.most\_common(20)

print(common\_words)

For creating word clouds:

🡪text = ' '.join(df['text'])

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

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

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

plt.axis('off')

plt.show()

**Temporal Analysis**

Analyze how sentiment varies over time. Steps like convert timestamp and plot sentiment over time can be used.

For convert timestamp:

🡪df['timestamp'] = pd.to\_datetime(df['timestamp'])

df.set\_index('timestamp', inplace=True)

for plot sentiment over time:

🡪df.resample('M').size().plot()

plt.title('Number of Tweets Over Time')

plt.xlabel('Date')

plt.ylabel('Tweet Count')

plt.show()

**Text preprocessing**

Clean the text using below code:

🡪import re

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

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

lemmatizer = WordNetLemmatizer()

def preprocess\_text(text):

text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

text = re.sub(r'[^\w\s]', '', text)

tokens = word\_tokenize(text.lower())

tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop\_words]

return ' '.join(tokens)

df['clean\_text'] = df['text'].apply(preprocess\_text)

**Sentiment prediction model**

Implement a sentiment prediction model using machine learning or natural language processing techniques.

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, f1\_score

vectorizer = TfidfVectorizer(max\_features=5000)

X = vectorizer.fit\_transform(df['clean\_text']).toarray()

y = df['target']

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = LogisticRegression()

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

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

accuracy = accuracy\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred, average='weighted')

print(f"Accuracy: {accuracy}")

print(f"F1 Score: {f1}")

**Feature Importance**

Identify the most important features (words or phrases) contributing to sentiment predictions.Visualize feature importance using techniques such as bar charts or word clouds.

🡪import numpy as np

importance = np.abs(model.coef\_[0])

indices = np.argsort(importance)[-10:] # Get indices of the 10 most important features

important\_features = [vectorizer.get\_feature\_names\_out()[i] for i in indices]

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

plt.barh(range(len(indices)), importance[indices], align='center')

plt.yticks(range(len(indices)), important\_features)

plt.title('Top 10 Important Features for Sentiment Prediction')

plt.show()

**User Interface**

Develop a simple user interface allowing users to input custom text for sentiment analysis.

🡪import streamlit as st

st.title("Twitter Sentiment Analysis")

user\_input = st.text\_area("Enter a tweet for sentiment analysis:")

if st.button("Analyze"):

input\_data = vectorizer.transform([user\_input])

prediction = model.predict(input\_data)

sentiment = "Positive" if prediction == 4 else "Negative"

st.write(f"The sentiment of the tweet is: {sentiment}")

**Documentation**

Use Jupyter Notebook to combine code, visualizations, and explanations. Ensure that each section of the project is well-documented with markdown cells explaining