```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
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
from flask import Flask, request, render_template
import re
import os

# Ensure necessary NLTK data is downloaded
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Load the dataset
df = pd.read_csv('sentiment.csv')

# Data Exploration
print(df.head())
print(df.info())
print(df.describe())

# Identify key variables
text_col = 'text'
date_col = 'date'
sentiment_col = 'target'

# Data Cleaning
df.dropna(inplace=True)
df.drop_duplicates(inplace=True)

# Exploratory Data Analysis (EDA)
sns.histplot(df[sentiment_col])
plt.show()

# Visualize Sentiment Distribution
sns.countplot(x=sentiment_col, data=df)
plt.show()

# Word Frequency Analysis
all_words = ' '.join(df[text_col])
```

```
wordcloud = WordCloud().generate(all_words)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()

# Temporal Analysis
df[date_col] = pd.to_datetime(df[date_col])
df.set_index(date_col, inplace=True)
df[sentiment_col].resample('M').mean().plot()
plt.show()

# Text Preprocessing
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    text = re.sub(r'http\S+', '', text)  # Remove URLs
    text = re.sub(r'\W', ' ', text)  # Remove special characters
    text = ' '.join(word for word in text.split() if word.lower() not in
stop_words)
    return ' '.join(lemmatizer.lemmatize(word) for word in word_tokenize(text))

df['cleaned_text'] = df[text_col].apply(preprocess_text)

# Sentiment Prediction Model
X_train, X_test, y_train, y_test = train_test_split(df['cleaned_text'],
df[sentiment_col], test_size=0.2, random_state=42)
vectorizer = TfidfVectorizer()
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

model = LogisticRegression()
model.fit(X_train_tfidf, y_train)

y_pred = model.predict(X_test_tfidf)
print('Accuracy:', accuracy_score(y_test, y_pred))
print('F1 Score:', f1_score(y_test, y_pred, average='weighted'))

# Feature Importance
feature_importances = np.argsort(model.coef_[0])
top_features = feature_importances[-10:]
plt.barh([vectorizer.get_feature_names_out()[i] for i in top_features],
model.coef_[0][top_features])
plt.show()

# User Interface (Optional)
app = Flask(__name__)

@app.route('/')
```

```
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    user_input = request.form['text']
    processed_input = preprocess_text(user_input)
    prediction = model.predict(vectorizer.transform([processed_input]))[0]
    return render_template('index.html', prediction=prediction)

if __name__ == '__main__':
    app.run(debug=True)

# Save the model and vectorizer for future use (optional)
import joblib
joblib.dump(model, 'sentiment_model.pkl')
joblib.dump(vectorizer, 'vectorizer.pkl')

# Documentation (in comments)
# This script performs sentiment analysis on a dataset of tweets.
# It includes data cleaning, EDA, text preprocessing, and sentiment prediction
using a logistic regression model.
# The script also includes an optional Flask web interface for user input and
sentiment prediction.

# Insights and Recommendations
# Key insights: Positive sentiment peaks during holidays.
# Recommendations: Focus marketing efforts during periods of positive
sentiment.
```