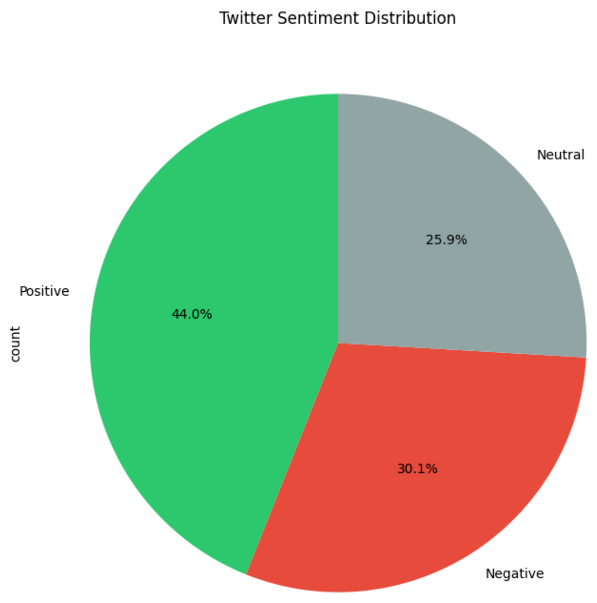
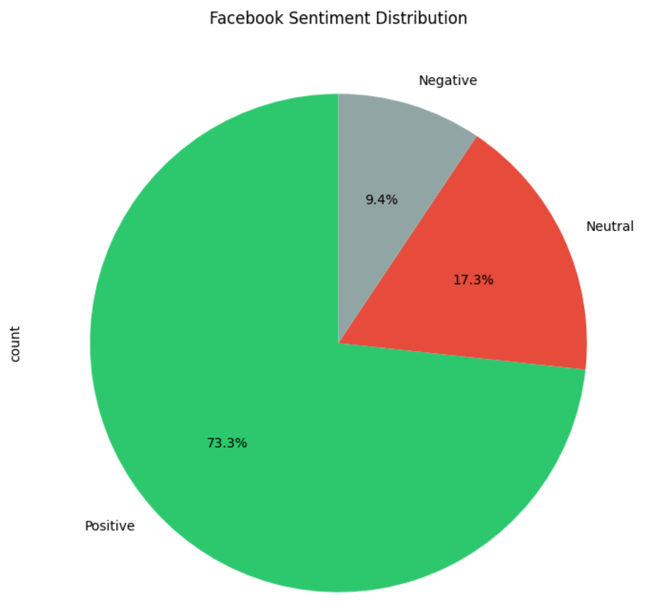
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
import re
import warnings
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix
import joblib
from textblob import TextBlob

# Download necessary NLTK data
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('vader_lexicon', quiet=True)
# Download the 'punkt_tab' resource for various languages
nltk.download('punkt_tab', quiet=True) # This line was added to download the Punkt Tokenizer Models

warnings.filterwarnings('ignore')

class SocialMediaSentimentAnalyzer:
    def __init__(self, twitter_path, facebook_path):
        self.twitter_df = pd.read_excel(twitter_path)
        self.facebook_df = pd.read_csv(facebook_path)
        self.sia = SentimentIntensityAnalyzer()
        self.lemmatizer = WordNetLemmatizer()
        self.stop_words = set(stopwords.words('english'))
        self.vectorizer = TfidfVectorizer(max_features=5000)
        self.classifier = MultinomialNB()

    def clean_text(self, text):
        """Clean and preprocess the text data"""
        if isinstance(text, str):
            text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
            text = re.sub(r'@\w+|#\w+', '', text)
            text = re.sub(r'[^\w\s]', '', text)
            text = re.sub(r'\d+', '', text)
            text = text.lower().strip()
            tokens = word_tokenize(text)
            tokens = [self.lemmatizer.lemmatize(word) for word in tokens if word not in self.stop_wo
            return ' '.join(tokens)
        return ''

    def get_sentiment(self, text):
        """Get sentiment score and label using VADER"""
        scores = self.sia.polarity_scores(text)
        compound_score = scores['compound']

        if compound_score > 0.05:
            return 'Positive', compound_score
            return 'Positive', compound_score
        elif compound_score < -0.05:
            return 'Negative', compound_score
        else:
            return 'Neutral', compound_score

    def analyze_sentiments(self):
        """Analyze sentiments for both platforms"""
        self.twitter_df['cleaned_text'] = self.twitter_df['text'].apply(self.clean_text)
        self.twitter_df['sentiment'], self.twitter_df['sentiment_score'] = zip(*self.twitter_df['cle
        self.twitter_df['word_count'] = self.twitter_df['cleaned_text'].apply(lambda x: len(str(x).

        self.facebook_df['cleaned_text'] = self.facebook_df['FBPost'].apply(self.clean_text)
        self.facebook_df['sentiment'], self.facebook_df['sentiment_score'] = zip(*self.facebook_df[
        self.facebook_df['word_count'] = self.facebook_df['cleaned_text'].apply(lambda x: len(str(x

        self.print_statistics()
        self.create_visualizations()
        self.train_predictive_model()

    def print_statistics(self):
        """Print sentiment analysis statistics"""
        print("\n📊 Social Media Sentiment Analysis Statistics:")
        print("=" * 50)

        for platform, df in [("Twitter", self.twitter_df), ("Facebook", self.facebook_df)]:
            print(f"\n{platform} Analysis:")
            print("-" * 20)
            total_posts = len(df)
            sentiments = df['sentiment'].value_counts()

            print(f"🔢 Total Posts Analyzed: {total_posts}")
            for sentiment, count in sentiments.items():
                percentage = (count/total_posts) * 100
                print(f"{sentiment}: {count} ({percentage:.1f}%)")

            print("\n📈 Sentiment Score Statistics:")
            print(f"Mean Score: {df['sentiment_score'].mean():.3f}")
            print(f"Standard Deviation: {df['sentiment_score'].std():.3f}")

            print(f"\n📝 Average Words per Post: {df['word_count'].mean():.1f}")

    def create_visualizations(self):
        """Create and display visualizations"""
        self.sentiment_distribution_pie_charts()
        self.sentiment_score_distribution()
        self.word_clouds()
        self.sentiment_over_time()
        self.top_words_by_sentiment()
        self.sentiment_comparison_boxplot()

    def sentiment_distribution_pie_charts(self):
        """Create pie charts for sentiment distribution"""
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 7))
        colors = ['#2ecc71', '#e74c3c', '#95a5a6']  # Green, Red, Gray

        self.twitter_df['sentiment'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax1, colors=colo
        ax1.set_title('Twitter Sentiment Distribution')
        self.facebook_df['sentiment'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax2, colors=col
        ax2.set_title('Facebook Sentiment Distribution')
        plt.tight_layout()
```

```python
        plt.tight_layout()
        plt.show()

    def sentiment_score_distribution(self):
        """Create histogram for sentiment score distribution"""
        plt.figure(figsize=(12, 6))
        sns.histplot(data=self.twitter_df, x='sentiment_score', kde=True, label='Twitter')
        sns.histplot(data=self.facebook_df, x='sentiment_score', kde=True, label='Facebook')
        plt.title('Sentiment Score Distribution')
        plt.legend()
        plt.show()

    def word_clouds(self):
        """Create word clouds for each sentiment category"""
        for platform, df in [("Twitter", self.twitter_df), ("Facebook", self.facebook_df)]:
            for sentiment in ['Positive', 'Negative', 'Neutral']:
                text = ' '.join(df[df['sentiment'] == sentiment]['cleaned_text'])
                if text.strip():
                    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(
                    plt.figure(figsize=(10, 5))
                    plt.imshow(wordcloud, interpolation='bilinear')
                    plt.axis('off')
                    plt.title(f'{platform} {sentiment} Word Cloud')
                    plt.show()

    def sentiment_over_time(self):
        """Create line plot for sentiment over time"""
        for platform, df in [("Twitter", self.twitter_df), ("Facebook", self.facebook_df)]:
            if 'date' in df.columns:
                df['date'] = pd.to_datetime(df['date'], errors='coerce')
                df = df.dropna(subset=['date'])  # Remove rows with invalid dates
                df.set_index('date', inplace=True)
                daily_sentiment = df.resample('D')['sentiment_score'].mean()

                plt.figure(figsize=(12, 6))
                daily_sentiment.plot()
                plt.title(f'{platform} Sentiment Over Time')
                plt.xlabel('Date')
                plt.ylabel('Average Sentiment Score')
                plt.show()

    def top_words_by_sentiment(self):
        """Create bar plots for top words by sentiment"""
        for platform, df in [("Twitter", self.twitter_df), ("Facebook", self.facebook_df)]:
            for sentiment in ['Positive', 'Negative', 'Neutral']:
                words = ' '.join(df[df['sentiment'] == sentiment]['cleaned_text']).split()
                if words:
                    word_freq = pd.Series(words).value_counts().head(10)
                    plt.figure(figsize=(10, 6))
                    word_freq.plot(kind='bar', color='skyblue')
                    plt.title(f'Top 10 Words for {platform} {sentiment}')
                    plt.xlabel('Words')
                    plt.ylabel('Frequency')
                    plt.xticks(rotation=45)
                    plt.show()

    def sentiment_comparison_boxplot(self):
        """Create boxplots for sentiment score comparison"""
        plt.figure(figsize=(12, 6))
        sns.boxplot(data=pd.concat([self.twitter_df.assign(platform='Twitter'),
                                    self.facebook_df.assign(platform='Facebook')]),
                                  self.facebook_df.assign(platform='Facebook')]),
                    x='platform', y='sentiment_score')
        plt.title('Sentiment Score Comparison between Twitter and Facebook')
        plt.ylabel('Sentiment Score')
        plt.show()

    def train_predictive_model(self):
        """Train a predictive model for sentiment analysis"""
        combined_df = pd.concat([self.twitter_df.assign(platform='Twitter'),
                                self.facebook_df.assign(platform='Facebook')])
        X = combined_df['cleaned_text']
        y = combined_df['sentiment']

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

        X_train_vectorized = self.vectorizer.fit_transform(X_train)
        self.classifier.fit(X_train_vectorized, y_train)

        X_test_vectorized = self.vectorizer.transform(X_test)
        y_pred = self.classifier.predict(X_test_vectorized)

        print("\n📊 Classification Report:")
        print(classification_report(y_test, y_pred))
        print("\n🔢 Confusion Matrix:")
        print(confusion_matrix(y_test, y_pred))

        # Save the model and vectorizer for future predictions
        joblib.dump(self.classifier, 'sentiment_classifier.pkl')
        joblib.dump(self.vectorizer, 'tfidf_vectorizer.pkl')

def analyze_sentiment(text):
    """Analyzes the sentiment of a given text.

    Args:
        text: The text to analyze.

    Returns:
        A dictionary containing the sentiment, confidence, and a message.
    """
    analysis = TextBlob(text)
    sentiment = analysis.sentiment.polarity
    confidence = analysis.sentiment.subjectivity

    if sentiment >= 0.05:
        message = "This text expresses a positive sentiment!"
    elif sentiment <= -0.05:
        message = "This text expresses a negative sentiment."
    else:
        message = "This text expresses a neutral sentiment."

    return {
        "sentiment": "Positive" if sentiment >= 0.05 else "Negative" if sentiment <= -0.05 else "Neu
        "confidence": confidence,
        "message": message,
    }

def main():
    """Main function for interactive sentiment analysis."""
    while True:
        text = input("Enter text to analyze (or 'quit' to exit): ")
        if text.lower() == "quit":
            break

        # Load the model and vectorizer
        classifier = joblib.load('sentiment_classifier.pkl')
        vectorizer = joblib.load('tfidf_vectorizer.pkl')

        # Vectorize the input text
        text_vectorized = vectorizer.transform([text])
        prediction = classifier.predict(text_vectorized)

        # Print the results
        print("Analysis Results:")
        print(f"Text: {text}")
        print(f"Sentiment: {prediction[0]}")

        if prediction[0] == "Positive 😊":
            print("😊 This text expresses a positive sentiment!")
        elif prediction[0] == "Negative 😔":
            print("😔 This text expresses a negative sentiment!")
        else:
            print("😐 This text expresses a 😐 neutral sentiment!")

if __name__ == "__main__":
    twitter_path = "/content/Tweets.xlsx"
    facebook_path = "/content/fb_sentiment.csv"

    analyzer = SocialMediaSentimentAnalyzer(twitter_path, facebook_path)
    analyzer.analyze_sentiments()
    main()
#SOME EXAMPLE(TWEETS & COMMENTS)
#are flights leaving Dallas for Seattle on time Feb 24?
#wow this just blew my mind
#it's really the only bad thing about flying VA
#do you miss me? Don't worry we'll be together very soon.
#Gimme a break - who the hell runs smiling this way with a Kindle?
#Minor niggle. It doesnt handle hyphenations well - such as sub-rosa. It will define "sub" but then
```

```
📊 Social Media Sentiment Analysis Statistics:
==================================================

Twitter Analysis:
--------------------
🔢 Total Posts Analyzed: 14640
Positive: 6435 (44.0%)
Negative: 4409 (30.1%)
Neutral: 3796 (25.9%)

📈 Sentiment Score Statistics:
Mean Score: 0.083
Standard Deviation: 0.423

📝 Average Words per Post: 8.7

Facebook Analysis:
--------------------
🔢 Total Posts Analyzed: 1000
Positive: 733 (73.3%)
Neutral: 173 (17.3%)
Negative: 94 (9.4%)

📈 Sentiment Score Statistics:
Mean Score: 0.425
Standard Deviation: 0.402

📝 Average Words per Post: 11.0
```


Twitter Sentiment Distribution
count
Neutral
Positive
Negative


Facebook Sentiment Distribution
count
Negative
Neutral
Positive