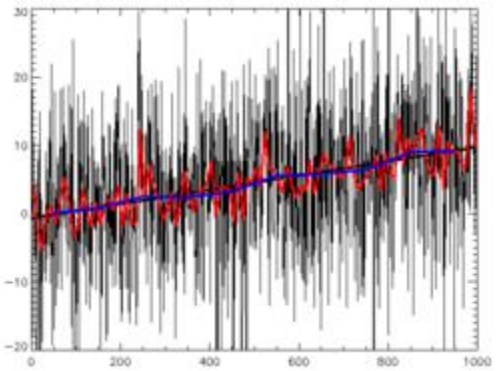
OVERVIEW- TIME SERIES ANALYSIS PROJECT- *Sentiment Analysis*

Rough Draft

Time series analysis is a statistical method for analysing data points collected over time to identify patterns, trends, and relationships. It's used in a variety of fields, including investing, physics, and signal processing.



***Sentiment Analysis**

Track and Analyse time-series sentiment trends in tweets or posts about a topic.

NLP and Sentiment Analysis

Elaboration: Sentiment Analysis of Social Media Data

****Objective:****

Build a model that analyzes the sentiment of social media posts (e.g., tweets) over time and predicts trends. This could help businesses, researchers, or analysts understand public opinions about products, events, or topics.

**Steps to Get Started**

1. **Understand the Basics**

- **Sentiment Analysis**: The process of categorizing text as positive, negative, or neutral.

- **Time-Series Component**: Sentiments are tracked over time to observe trends or predict future spikes.

- Skills Required:

- **Natural Language Processing (NLP)**: For extracting sentiment from text.

- **Time-Series Analysis**: For forecasting trends.

- **Programming**: Python is widely used for both tasks.

- **Visualization**: Libraries like Matplotlib or Plotly.

---

2. **Data Collection**

- **Source Social Media Data**:

- **Twitter API** (using `tweepy`): Fetch tweets in real-time or historical tweets based on hashtags or topics.

- Other Platforms: Use platforms like Reddit or YouTube for comment analysis (via APIs or scraping).

- **Prerequisites**:

- Create an API key and secret from the Twitter Developer portal.

- Install Python packages: `tweepy` for fetching data.

---

3. **Preprocessing the Data**

- Clean the text:

- Remove URLs, mentions (`@username`), hashtags, and emojis.

- Convert to lowercase.

- Remove stopwords (e.g., "the", "is") using NLP libraries like NLTK or spaCy.

- Tokenize the text (split it into words or phrases).

- Example Code:

```python

import re

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize
```

```
    # Clean text function
    def clean_text(text):
        text = re.sub(r"http\S+|@\S+|#\S+", "", text)  # Remove URLs, mentions, hashtags
        text = re.sub(r"[^\w\s]", "", text)  # Remove punctuation
        text = text.lower()  # Convert to lowercase
        tokens = word_tokenize(text)
        tokens = [word for word in tokens if word not in stopwords.words("english")]
        return " ".join(tokens)
```

---

4. **Sentiment Analysis**
  - Use pre-built sentiment analysis libraries:
    - **TextBlob**:
      ```python
      from textblob import TextBlob

      def get_sentiment(text):
          analysis = TextBlob(text)
          if analysis.sentiment.polarity > 0:
              return "Positive"
          elif analysis.sentiment.polarity == 0:
              return "Neutral"
          else:
              return "Negative"
      ```
    - **VADER** (Valence Aware Dictionary for Sentiment Reasoning):
      ```python
      from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
      ```

```
analyzer = SentimentIntensityAnalyzer()
sentiment = analyzer.polarity_scores("Your tweet text")
print(sentiment)
```

- **Advanced Models**:
  - Use fine-tuned **BERT** or **RoBERTa** models for higher accuracy. Libraries like Hugging Face simplify this.

---

5. **Visualizing Sentiment Trends**

- Aggregate sentiments over time:
  - Group by days, weeks, or hours.
  - Calculate the percentage of positive, negative, and neutral sentiments.
- Plot the results:

```python
import matplotlib.pyplot as plt

# Example sentiment trend
sentiments = {"Positive": 40, "Negative": 20, "Neutral": 40}
plt.pie(sentiments.values(), labels=sentiments.keys(), autopct="%1.1f%%")
plt.title("Sentiment Distribution")
plt.show()
```

---

6. **Time-Series Modeling**

- Extract daily/weekly sentiment scores (e.g., average polarity for each day).
- Apply **Time-Series Models**:
  - **Statistical Models**: ARIMA, SARIMA for trend analysis.

- **Machine Learning Models**: LSTM or GRU networks for sequential prediction.
- Libraries:
  - ARIMA: `statsmodels`
  - LSTM/GRU: `TensorFlow` or `PyTorch`

---

7. **Deployment and Dashboard**
- Create a live dashboard for sentiment trends:
  - Use `Dash` (Python) or `Streamlit` for easy visualization.
  - Embed interactive charts using Plotly.
- Include sentiment forecasts or alerts for sentiment spikes.

Requirements

Technical Tools
- **Programming**: Python is the primary language.
- **APIs**: Twitter API or Reddit API for fetching data.
- **Libraries**:
  - NLP: `TextBlob`, `NLTK`, `spaCy`, `vaderSentiment`
  - Data Preprocessing: `pandas`, `numpy`
  - Time-Series Analysis: `statsmodels`, `TensorFlow`
  - Visualization: `matplotlib`, `seaborn`, `plotly`

Hardware
- A computer with a decent processor (e.g., Core i5) and at least 8GB RAM for deep learning models.

Skills to Develop

- **NLP Basics**: Tokenization, stopword removal, and basic sentiment analysis.
- **Data Preprocessing**: Handling missing data, cleaning raw text.
- **Time-Series Analysis**: Understanding trends, seasonality, and forecasting techniques.
- **Visualization**: Using libraries like Matplotlib, Plotly, or Tableau.

Expected Outcomes

- A tool that can:
  1. Fetch live or historical social media data.
  2. Analyze the sentiment of each post.
  3. Display trends in sentiment over time.
  4. Forecast sentiment spikes.
- An interactive dashboard for real-time monitoring.

Setting up the Twitter API for Sentiment Analysis

The Twitter API is widely used for fetching tweets based on keywords, hashtags, user mentions, or specific topics. Here's a step-by-step guide to set it up and integrate it into your project:

---

1. Create a Twitter Developer Account

1. **Sign Up**:

   Visit the [Twitter Developer Portal](https://developer.twitter.com/) and log in with your Twitter account.

2. **Apply for a Developer Account**:
   - Provide details about your project and why you need the API.
   - Accept Twitter's Developer Agreement.
3. **Create a Project and App**:
   - Once approved, create a project and an app under it.
   - Note the **API Key**, **API Key Secret**, **Bearer Token**, and **Access Tokens**.

---

2. Install Required Libraries

You need `tweepy` to interact with the Twitter API. Install it using pip:

```bash
pip install tweepy
```

Other recommended libraries:

- **pandas**: For data handling.
- **re**: For text preprocessing.
- **nltk** or **spaCy**: For text processing.

---

3. Authenticate Using Tweepy

Set up your project to connect to the Twitter API. Here's the code snippet:

```python
import tweepy

# Replace with your credentials
api_key = "your_api_key"
api_key_secret = "your_api_key_secret"
access_token = "your_access_token"
access_token_secret = "your_access_token_secret"

# Authenticate
auth = tweepy.OAuth1UserHandler(api_key, api_key_secret, access_token, access_token_secret)
```

```
api = tweepy.API(auth)

# Verify the connection
try:
    api.verify_credentials()
    print("Authentication Successful")
except Exception as e:
    print("Authentication Error:", e)
```

---

4. Fetch Tweets

Example: Fetch Tweets by Hashtag

```python
# Fetch tweets containing a specific hashtag
query = "#AI"  # Replace with your desired hashtag or keyword
tweets = api.search_tweets(q=query, lang="en", count=100)  # Fetch 100 tweets

# Print fetched tweets
for tweet in tweets:
    print(tweet.text)
```

Example: Fetch Tweets by User

```python
# Fetch tweets from a specific user
username = "elonmusk"
tweets = api.user_timeline(screen_name=username, count=10)

# Print tweets
```

```
for tweet in tweets:
    print(tweet.text)
```

---

5. Save Tweets to a File

You can store tweets in a CSV file for analysis:

```python
import pandas as pd

data = []
for tweet in tweets:
    data.append({"Date": tweet.created_at, "Tweet": tweet.text, "Likes": tweet.favorite_count})

# Convert to DataFrame and save as CSV
df = pd.DataFrame(data)
df.to_csv("tweets.csv", index=False)
print("Saved to tweets.csv")
```

---

6. Handle Rate Limits

Twitter limits API calls (e.g., 450 requests per 15 minutes for standard accounts). Use Tweepy's rate limit handler to avoid hitting these limits:

```python
from tweepy import Cursor
```

```
# Fetch tweets with rate limit handling
for tweet in Cursor(api.search_tweets, q=query, lang="en").items(1000):
    print(tweet.text)
```

---

7. Upgrade API Access (Optional)

If the free tier doesn't meet your requirements:

- Upgrade to a paid tier for higher API limits.

- Use Twitter's Academic Research product for large-scale data collection (requires proof of academic affiliation).

**8. Preprocess the Data**

Before performing sentiment analysis, clean the tweets:

- Remove URLs, mentions, hashtags, and special characters:

```python
import re

def clean_tweet(tweet):
    tweet = re.sub(r"http\S+", "", tweet)  # Remove URLs
    tweet = re.sub(r"@\S+", "", tweet)    # Remove mentions
    tweet = re.sub(r"#", "", tweet)       # Remove hashtags
    tweet = re.sub(r"[^\w\s]", "", tweet) # Remove special characters
    return tweet.lower()

# Apply to all tweets
df['Cleaned_Tweet'] = df['Tweet'].apply(clean_tweet)
```

---

9. Automate Data Collection (Optional)

For continuous data collection, use Tweepy's **streaming API** to fetch real-time tweets:

```python
from tweepy import Stream
from tweepy.streaming import StreamListener

class MyStreamListener(StreamListener):
    def on_status(self, status):
        print(status.text)

    def on_error(self, status_code):
        print("Error:", status_code)
        return False  # Stop the stream in case of an error

# Stream tweets containing specific keywords
myStream = Stream(auth=api.auth, listener=MyStreamListener())
myStream.filter(track=["#AI", "#MachineLearning"])
```

---

10. Next Steps

Once you've set up the API:

1. **Sentiment Analysis**: Apply NLP techniques (e.g., VADER, TextBlob, or Hugging Face models) to analyze the fetched tweets.

2. **Visualization**: Use tools like Matplotlib, Seaborn, or Dash to display sentiment trends.

3. **Time-Series Forecasting**: Aggregate sentiment scores over time and predict future trends using ARIMA or LSTMs.

Processing tweets is crucial for effective sentiment analysis. Tweets are often noisy, containing unnecessary characters like URLs, hashtags, mentions, and emojis. Here's a step-by-step guide:

---

1. Import Necessary Libraries

Install and import the required libraries for text preprocessing:

```bash
pip install nltk pandas re
```

```python
import re
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download NLTK data (first-time setup)
nltk.download('stopwords')
nltk.download('punkt')
```

---

2. Load the Tweet Data

If you've saved tweets in a CSV file:

```python
df = pd.read_csv("tweets.csv")
print(df.head())
```

```

Assume the column containing tweets is named `Tweet`.

---

### **3. Define a Cleaning Function**

A function to clean tweets is essential for removing unwanted text, converting text to lowercase, and handling emojis.

#### Example Code:

```python
def clean_tweet(tweet):
    # Remove URLs
    tweet = re.sub(r"http\S+", "", tweet)
    # Remove mentions and hashtags
    tweet = re.sub(r"@\S+|#\S+", "", tweet)
    # Remove special characters, numbers, and punctuations
    tweet = re.sub(r"[^A-Za-z\s]", "", tweet)
    # Convert to lowercase
    tweet = tweet.lower()
    return tweet
```

Apply Cleaning:

```python
df['Cleaned_Tweet'] = df['Tweet'].apply(clean_tweet)
print(df[['Tweet', 'Cleaned_Tweet']].head())
```

```

4. Remove Stopwords

Stopwords (e.g., "the", "is", "and") don’t add value to sentiment analysis. Use NLTK to remove them.

Example Code:

```python
stop_words = set(stopwords.words("english"))

def remove_stopwords(tweet):
    words = word_tokenize(tweet)
    filtered_words = [word for word in words if word not in stop_words]
    return " ".join(filtered_words)
```

Apply Stopword Removal:

```python
df['Processed_Tweet'] = df['Cleaned_Tweet'].apply(remove_stopwords)
print(df[['Cleaned_Tweet', 'Processed_Tweet']].head())
```

---

5. Handle Emojis (Optional)

Use libraries like `emoji` to convert emojis into text:

```bash
pip install emoji
```

```python
import emoji
```

```
def convert_emojis(tweet):
    return emoji.demojize(tweet)

df['Emoji_Handled_Tweet'] = df['Processed_Tweet'].apply(convert_emojis)
print(df['Emoji_Handled_Tweet'].head())
```

---

6. Tokenization
Tokenization splits a sentence into individual words for further analysis.

Example Code:
```python
def tokenize_tweet(tweet):
    return word_tokenize(tweet)

df['Tokens'] = df['Emoji_Handled_Tweet'].apply(tokenize_tweet)
print(df[['Processed_Tweet', 'Tokens']].head())
```

---

7. Lemmatization (Optional but Recommended)
Lemmatization reduces words to their base or dictionary form (e.g., "running" → "run").

Example Code:
```python
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
```

```
lemmatizer = WordNetLemmatizer()

def lemmatize_tweet(tokens):
    lemmatized = [lemmatizer.lemmatize(word) for word in tokens]
    return " ".join(lemmatized)

df['Lemmatized_Tweet'] = df['Tokens'].apply(lambda x: lemmatize_tweet(x))
print(df[['Processed_Tweet', 'Lemmatized_Tweet']].head())
```

---

8. Final Preprocessed Data

After processing:

- **Raw Tweet**: Original tweet.
- **Cleaned_Tweet**: URLs, mentions, and hashtags removed.
- **Processed_Tweet**: Stopwords removed.
- **Lemmatized_Tweet**: Final text ready for sentiment analysis.

Save the processed tweets to a new CSV:

```python
df.to_csv("processed_tweets.csv", index=False)
```

---

9. Next Steps

Now that the tweets are preprocessed:

1. **Sentiment Analysis**:
   - Use tools like **TextBlob**, **VADER**, or **Hugging Face Transformers** for sentiment extraction.

- Example using TextBlob:

```python
from textblob import TextBlob


def get_sentiment(tweet):
    analysis = TextBlob(tweet)
    if analysis.sentiment.polarity > 0:
        return "Positive"
    elif analysis.sentiment.polarity == 0:
        return "Neutral"
    else:
        return "Negative"


df['Sentiment'] = df['Lemmatized_Tweet'].apply(get_sentiment)
print(df[['Lemmatized_Tweet', 'Sentiment']].head())
```

2. **Time-Series Analysis**:

- Group tweets by date and calculate the percentage of positive, negative, and neutral tweets over time.

```python
df['Date'] = pd.to_datetime(df['Date'])
sentiment_trend = df.groupby(df['Date'].dt.date)['Sentiment'].value_counts(normalize=True).unstack()
sentiment_trend.plot(kind='line', figsize=(10, 6))
```

Step 1: Perform Sentiment Analysis

Once your tweets are preprocessed, you can use libraries like **TextBlob**, **VADER**, or **Hugging Face Transformers** to extract sentiment. Here's how you can proceed with each:

Option A: Using TextBlob

TextBlob is simple and effective for basic sentiment analysis.

Install TextBlob:

```bash
pip install textblob
```

Code for Sentiment Analysis:

```python
from textblob import TextBlob

def get_sentiment(tweet):
    analysis = TextBlob(tweet)
    if analysis.sentiment.polarity > 0:
        return "Positive"
    elif analysis.sentiment.polarity == 0:
        return "Neutral"
    else:
        return "Negative"

# Apply sentiment analysis
df['Sentiment'] = df['Lemmatized_Tweet'].apply(get_sentiment)
print(df[['Lemmatized_Tweet', 'Sentiment']].head())
```

Option B: Using VADER

VADER (Valence Aware Dictionary and sEntiment Reasoner) is designed for analyzing social media content.

Install VADER:

```bash
pip install vaderSentiment
```

Code for Sentiment Analysis:

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

analyzer = SentimentIntensityAnalyzer()

def get_vader_sentiment(tweet):
    score = analyzer.polarity_scores(tweet)
    if score['compound'] > 0.05:
        return "Positive"
    elif score['compound'] < -0.05:
        return "Negative"
    else:
        return "Neutral"

# Apply sentiment analysis
df['Sentiment'] = df['Lemmatized_Tweet'].apply(get_vader_sentiment)
print(df[['Lemmatized_Tweet', 'Sentiment']].head())
```

Option C: Using Hugging Face Transformers

For more advanced analysis, use pre-trained transformer models like **BERT**.

Install Hugging Face Libraries:

```bash
pip install transformers torch
```

Code for Sentiment Analysis:

```python
from transformers import pipeline

# Load pre-trained sentiment analysis model
sentiment_model = pipeline("sentiment-analysis")

def get_transformer_sentiment(tweet):
    result = sentiment_model(tweet)[0]
    return result['label']

# Apply sentiment analysis
df['Sentiment'] = df['Lemmatized_Tweet'].apply(get_transformer_sentiment)
print(df[['Lemmatized_Tweet', 'Sentiment']].head())
```

Step 2: Visualize Sentiment Trends Over Time

After performing sentiment analysis, you can group tweets by date and visualize sentiment distribution.

---

1. Prepare Time-Series Data

Ensure the date column is in a `datetime` format:

```python
df['Date'] = pd.to_datetime(df['Date'])
df['Date'] = df['Date'].dt.date  # Convert to date format (optional)
```

---

2. Calculate Sentiment Counts

Group tweets by date and calculate the count of each sentiment type:

```python
sentiment_counts = df.groupby(['Date', 'Sentiment']).size().unstack(fill_value=0)
print(sentiment_counts.head())
```

---

3. Calculate Percentages

To plot sentiment trends, normalize the data (convert counts to percentages):

```python
sentiment_percentages = sentiment_counts.div(sentiment_counts.sum(axis=1), axis=0) * 100
print(sentiment_percentages.head())
```

---

4. Plot Sentiment Trends

Use Matplotlib or Seaborn to visualize:

```python
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
for sentiment in sentiment_percentages.columns:
    plt.plot(sentiment_percentages.index, sentiment_percentages[sentiment], label=sentiment)

plt.title("Sentiment Trends Over Time")
plt.xlabel("Date")
plt.ylabel("Percentage")
plt.legend(title="Sentiment")
plt.grid(True)
plt.show()
```

Step 3: Advanced Visualization

Option A: Stacked Bar Chart
A bar chart shows daily sentiment composition:
```python
sentiment_counts.plot(kind='bar', stacked=True, figsize=(12, 6), colormap="coolwarm")
plt.title("Daily Sentiment Composition")
plt.xlabel("Date")
plt.ylabel("Tweet Count")
plt.legend(title="Sentiment")
plt.show()
```

Option B: Interactive Dashboard with Plotly
Install Plotly for interactive plots:

```bash
pip install plotly
```

Code:

```python
import plotly.express as px

fig = px.line(sentiment_percentages.reset_index(),
              x='Date',
              y=['Positive', 'Neutral', 'Negative'],
              labels={"value": "Percentage", "Date": "Date"},
              title="Sentiment Trends Over Time")

fig.update_layout(legend_title_text='Sentiment')
fig.show()
```

Next Steps

1. **Optimize Sentiment Models**:
   - Experiment with different libraries and evaluate accuracy.
   - Add custom preprocessing for domain-specific tweets (e.g., slang, abbreviations).
2. **Predict Future Sentiment Trends**:
   - Use **ARIMA** or **LSTMs** on the sentiment time series to forecast future trends.
3. **Create a Dashboard**:
   - Use tools like **Dash**, **Flask**, or **Streamlit** to present real-time analysis.

1. **Forecasting Future Sentiment Trends**

To forecast future sentiment trends, use models like **ARIMA** (AutoRegressive Integrated Moving Average) for univariate time series or **LSTMs** (Long Short-Term Memory) for multivariate and more complex patterns.

A. Using ARIMA

Step 1: Install Required Libraries

```bash
pip install statsmodels matplotlib
```

Step 2: Prepare Data

Aggregate sentiment counts by date:

```python
daily_sentiment = df.groupby(['Date', 'Sentiment']).size().unstack(fill_value=0)
daily_positive = daily_sentiment['Positive']

# Ensure the index is a datetime object
daily_positive.index = pd.to_datetime(daily_positive.index)
```

Step 3: Fit an ARIMA Model

```python
from statsmodels.tsa.arima.model import ARIMA

# Train ARIMA model
model = ARIMA(daily_positive, order=(5, 1, 0))  # (p, d, q)
model_fit = model.fit()

# Forecast
forecast = model_fit.forecast(steps=30)  # Predict next 30 days
```

print(forecast)
```

**Step 4: Visualize the Forecast**

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(daily_positive, label="Historical Positive Sentiment")
plt.plot(forecast, label="Forecast", linestyle='--', color='red')
plt.title("Positive Sentiment Forecast")
plt.xlabel("Date")
plt.ylabel("Tweet Count")
plt.legend()
plt.show()
```

B. Using LSTMs

LSTMs are better for capturing complex patterns.

Step 1: Install Required Libraries

```bash
pip install tensorflow numpy matplotlib
```

Step 2: Prepare Data

Normalize and reshape the data:

```python
import numpy as np
```
```

```
from sklearn.preprocessing import MinMaxScaler

# Normalize data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(daily_positive.values.reshape(-1, 1))

# Create sequences
def create_sequences(data, time_steps=5):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i + time_steps])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)

X, y = create_sequences(scaled_data, time_steps=5)
```

Step 3: Build LSTM Model

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(X.shape[1], 1)),
    LSTM(50, return_sequences=False),
    Dense(1)
])

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X, y, epochs=20, batch_size=32)
```

```

**Step 4: Forecast and Visualize**

```python
# Predict
future_steps = 30
forecast = []
current_input = X[-1]

for _ in range(future_steps):
    prediction = model.predict(current_input.reshape(1, -1, 1))
    forecast.append(prediction[0][0])
    current_input = np.append(current_input[1:], prediction)

# Transform back to original scale
forecast = scaler.inverse_transform(np.array(forecast).reshape(-1, 1))

# Visualize
plt.figure(figsize=(12, 6))
plt.plot(daily_positive.values, label="Historical Positive Sentiment")
plt.plot(range(len(daily_positive), len(daily_positive) + future_steps), forecast, label="Forecast", linestyle='--', color='red')
plt.title("Positive Sentiment Forecast with LSTM")
plt.xlabel("Date")
plt.ylabel("Tweet Count")
plt.legend()
plt.show()
```

```

2. **Create a Dashboard**

Use **Streamlit** for an interactive dashboard.

Step 1: Install Streamlit

```bash
pip install streamlit
```

Step 2: Create a Streamlit App

Save the following code in a file, e.g., `app.py`:

```python
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt

# Load Data
df = pd.read_csv("processed_tweets.csv")

# Sentiment Count Visualization
st.title("Tweet Sentiment Analysis")
st.subheader("Sentiment Over Time")
daily_sentiment = df.groupby(['Date', 'Sentiment']).size().unstack(fill_value=0)

st.line_chart(daily_sentiment)

# Forecast Visualization
st.subheader("Positive Sentiment Forecast")
forecast_fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(daily_sentiment.index, daily_sentiment['Positive'], label="Historical Positive Sentiment")
ax.plot(forecast, label="Forecast", linestyle='--', color='red')
```

ax.set_title("Positive Sentiment Forecast")

ax.set_xlabel("Date")

ax.set_ylabel("Tweet Count")

st.pyplot(forecast_fig)

```

**Step 3: Run the App**

Run the app in your terminal:

```bash

streamlit run app.py

```

---

3. **Advanced Visualizations**

Option A: Heatmap

Show sentiment distribution over time:

```python

import seaborn as sns

heatmap_data = daily_sentiment.T

sns.heatmap(heatmap_data, cmap="coolwarm", annot=True, fmt="d")

plt.title("Sentiment Distribution Heatmap")

plt.show()

```

Option B: Sentiment Pie Chart

Visualize sentiment proportions:

```python

sentiment_counts = df['Sentiment'].value_counts()

```
plt.pie(sentiment_counts, labels=sentiment_counts.index, autopct='%1.1f%%', colors=["green", "grey", "red"])
plt.title("Overall Sentiment Distribution")
plt.show()
```

Option C: Interactive Word Cloud

Install `wordcloud` and `plotly`:

```bash
pip install wordcloud plotly
```

Generate a word cloud for positive tweets:

```python
from wordcloud import WordCloud

positive_tweets = " ".join(df[df['Sentiment'] == 'Positive']['Processed_Tweet'])
wordcloud = WordCloud(background_color="white", colormap="viridis").generate(positive_tweets)

plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Positive Tweet Word Cloud")
plt.show()
```

Next Steps

2. Enhancing the Dashboard with More Interactivity

Streamlit allows for rich, interactive features, enabling you to create user-friendly dashboards for sentiment analysis. Here are enhancements you can add:

Feature 1: Filters for Sentiment and Time Range

Allow users to filter tweets by sentiment and time range.

Add Filters:

```python
# Sidebar filters
st.sidebar.header("Filters")
sentiment_filter = st.sidebar.multiselect(
    "Select Sentiments", options=df['Sentiment'].unique(), default=df['Sentiment'].unique()
)
date_filter = st.sidebar.date_input(
    "Select Date Range", [df['Date'].min(), df['Date'].max()]
)

# Apply filters
filtered_df = df[
    (df['Sentiment'].isin(sentiment_filter)) &
    (df['Date'] >= pd.to_datetime(date_filter[0])) &
    (df['Date'] <= pd.to_datetime(date_filter[1]))
]

st.write(f"Displaying data for {len(filtered_df)} tweets.")
st.dataframe(filtered_df)
```

Feature 2: Sentiment Count Bar Chart

Visualize sentiment counts dynamically based on filters.

Add a Bar Chart:

```python
import plotly.express as px

sentiment_counts = filtered_df['Sentiment'].value_counts().reset_index()
sentiment_counts.columns = ['Sentiment', 'Count']

fig = px.bar(sentiment_counts, x='Sentiment', y='Count', color='Sentiment', title="Filtered Sentiment Counts")
st.plotly_chart(fig)
```

Feature 3: Word Cloud Generator

Generate word clouds for each sentiment dynamically.

Add Word Clouds:

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

st.subheader("Word Cloud by Sentiment")
selected_sentiment = st.selectbox("Choose Sentiment", options=filtered_df['Sentiment'].unique())

**# Generate word cloud**
```

```
sentiment_text = " ".join(filtered_df[filtered_df['Sentiment'] ==
selected_sentiment]['Processed_Tweet'])

wordcloud = WordCloud(background_color="white",
colormap="coolwarm").generate(sentiment_text)


fig, ax = plt.subplots()

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

ax.axis("off")

st.pyplot(fig)
```

Feature 4: Interactive Sentiment Trends

Plot trends over time with dynamic updates based on filters.

Add Interactive Trend Line:

```python
trend_data = filtered_df.groupby(['Date', 'Sentiment']).size().unstack(fill_value=0).reset_index()

trend_fig = px.line(
    trend_data, x='Date', y=['Positive', 'Neutral', 'Negative'],
    title="Sentiment Trends Over Time", labels={'value': "Tweet Count", 'Date': "Date"}
)
st.plotly_chart(trend_fig)
```

Feature 5: Display Sample Tweets

Show example tweets for each sentiment.

Add Sample Tweet Display:

```python

st.subheader("Sample Tweets")

selected_sentiment_sample = st.selectbox("Select Sentiment for Sample Tweets", options=filtered_df['Sentiment'].unique())

sample_tweets = filtered_df[filtered_df['Sentiment'] == selected_sentiment_sample].head(5)['Processed_Tweet']


for i, tweet in enumerate(sample_tweets, 1):

   st.write(f"{i}. {tweet}")

```


---

3. Deployment of the App

Deploying your Streamlit app ensures it is accessible online. You can use **Heroku**, **Streamlit Community Cloud**, or other platforms. Here's how to deploy:

Option A: Deploying on Streamlit Community Cloud

1. **Sign Up and Create a Repository**:

   - Sign up at [Streamlit Community Cloud](https://streamlit.io/cloud).

   - Push your project files (`app.py`, `requirements.txt`, and datasets) to a GitHub repository.

2. **Add a Requirements File**:

   Create a `requirements.txt` file listing all dependencies:

   ```text

   streamlit

   pandas

   matplotlib

   numpy
   ```

wordcloud

plotly

vaderSentiment

textblob

transformers

```

3. **Deploy**:

- Go to Streamlit Cloud, click on "New App," and link your GitHub repository.

- Specify `app.py` as the main script.

---

## #### **Option B: Deploying on Heroku**

1. **Install Heroku CLI**:

Download and install the [Heroku CLI](https://devcenter.heroku.com/articles/heroku-cli).

2. **Create a `Procfile`**:

Add a `Procfile` to specify the command to run your app:

```text

web: streamlit run app.py --server.port=$PORT

```

3. **Prepare the Project**:

Ensure your project has:

- `app.py`: The main script.

- `requirements.txt`: Dependency list.

- `Procfile`: Run command.

4. **Deploy**:

Run the following commands in your terminal:
```

```bash
heroku login
heroku create your-app-name
git init
git add .
git commit -m "Initial commit"
git push heroku main
```

Your app will be live at `https://your-app-name.herokuapp.com`.

Option C: Using AWS or GCP

For more control, deploy your app on **Amazon Web Services (AWS)** or **Google Cloud Platform (GCP)**:

1. Set up a virtual machine (e.g., EC2 on AWS or Compute Engine on GCP).
2. Install Python, Streamlit, and necessary dependencies.
3. Open the Streamlit app with port forwarding to make it accessible online.
