Problem Statement

Analyze social media data (e.g., Twitter) to understand public sentiment towards specific topics, products or events. Use natural language processing (NLP) techniques to preprocess text data, extract sentiment scores and visualize sentiment trends over time.

[1]:
```
import pandas as pd
import numpy as np

# Importing dataset
df=pd.read_csv("C:/Users/sukan/Downloads/Tweeter_Sentiment_Data.csv")
df.head(3)
```

[1]:

	textID	text	selected_text	sentiment	Date
0	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative	2024-01-01
1	088c60f138	my boss is bullying me...	bullying me	negative	2024-01-01
2	9642c003ef	what interview! leave me alone	leave me alone	negative	2024-01-01

[2]:
```
# Checking the number of rows & columns present in the dataset
df.shape
```

[2]:
```
(16363, 5)
```

[4]:
```
# Checking if there is any null value present
df.isnull().sum()
```

[4]:
```
textID          0
text            0
selected_text   0
sentiment       0
Date            0
dtype: int64
```

[5]:
```
# Checking how many positive & negative sentiments are there in the dataset
df.sentiment.value_counts()
```

[5]:
```
sentiment
positive    8582
negative    7781
Name: count, dtype: int64
```

[6]:
```
# Segregating the input and output column
input_data=df[["selected_text"]]
output_data=df["sentiment"]
```

[7]:
```
# Balancing data using over sampling
from imblearn.over_sampling import RandomOverSampler
ro=RandomOverSampler()
ro_input_data,ro_output_data=ro.fit_resample(input_data,output_data)
ro_output_data.value_counts()
```

```
C:\Users\sukan\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn
ed in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation._check_n_fe
  warnings.warn(
C:\Users\sukan\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn
cated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation._check_f
  warnings.warn(
```

[7]:
```
sentiment
negative    8582
positive    8582
Name: count, dtype: int64
```

[8]:
```
# labeling sentiments
df["sentiment_number"]=df.sentiment.map({"positive":1,"negative":0})
df.head(3)
```

[8]:

	textID	text	selected_text	sentiment	Date	sentiment_number
0	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative	2024-01-01	0
1	088c60f138	my boss is bullying me...	bullying me	negative	2024-01-01	0
2	9642c003ef	what interview! leave me alone	leave me alone	negative	2024-01-01	0

[9]:
```
# Preprocess each selected text by removing punctuation and stopwords,
# applying lemmatization, and then converting the remaining tokens into vectors
# using Gensim's pretrained word embeddings (glove-twitter-25).
# Finally, compute the mean vector for each text as its overall representation.

import spacy
import gensim.downloader as api

nlp = spacy.load("en_core_web_lg")
word_vector = api.load("glove-twitter-25")

def preprocess_and_vectorize(selected_text):
    doc = nlp(selected_text)
    filtered_tokens = []
```

```
    for token in doc:
        if token.is_punct or token.is_stop:
            continue
        filtered_tokens.append(token.lemma_)

    if not filtered_tokens:
        return None

    return word_vector.get_mean_vector(filtered_tokens)

df["vector"] = df["selected_text"].apply(lambda text: preprocess_and_vectorize(text))

print(df.head(3))
```

```
       textID                                          text   selected_text  \
0  549e992a42   Sooo SAD I will miss you here in San Diego!!!        Sooo SAD
1  088c60f138                         my boss is bullying me...     bullying me
2  9642c003ef                   what interview! leave me alone  leave me alone

  sentiment        Date  sentiment_number  \
0  negative  2024-01-01                 0
1  negative  2024-01-01                 0
2  negative  2024-01-01                 0

                                              vector
0  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
1  [0.004638199, 0.24648528, -0.082712136, 0.0624...
2  [-0.10465123, 0.17052855, 0.009479763, 0.03759...
```

[10]:
```
# Checking how many rows in the 'vector' column have null values (i.e., texts that couldn't be vectorize

print(df["vector"].isnull().sum())
```

```
221
```

[11]:
```
# Removeing rows with null vectors and reset the index to keep the DataFrame clean and consistent

df = df[df["vector"].notnull()].reset_index(drop=True)
```

[12]:
```
# Counting how many vectors have each shape to ensure all vectorized texts have consistent dimensions

df["vector"].apply(lambda x: x.shape).value_counts()
```

[12]:
```
vector
(25,)    16142
Name: count, dtype: int64
```

[13]:
```
# Counting the number of occurrences of each sentiment label (e.g., positive, negative) in the dataset

df.sentiment.value_counts()
```

[13]:
```
sentiment
positive    8522
negative    7620
Name: count, dtype: int64
```

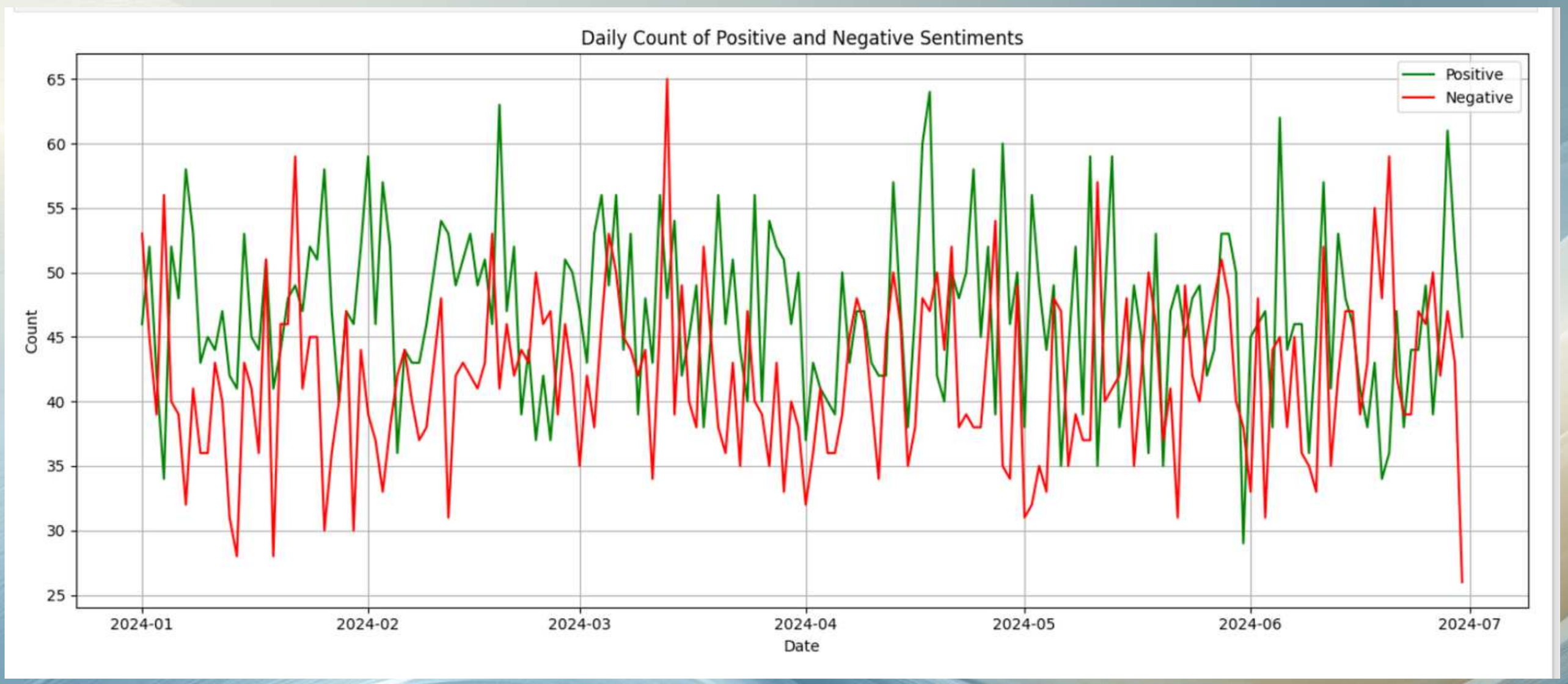
[43]:
```
# Spliting the dataset into training and testing sets (80% train, 20% test),

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df.vector.values,df.sentiment_number.values,test_size=0.2,random_state=2025,stratify=df.sentiment_number.va
X_train_2d=np.stack(X_train)
X_test_2d=np.stack(X_test)
print(X_train.shape)
print(X_train_2d.shape)
```

```
(12913,)
(12913, 25)
```

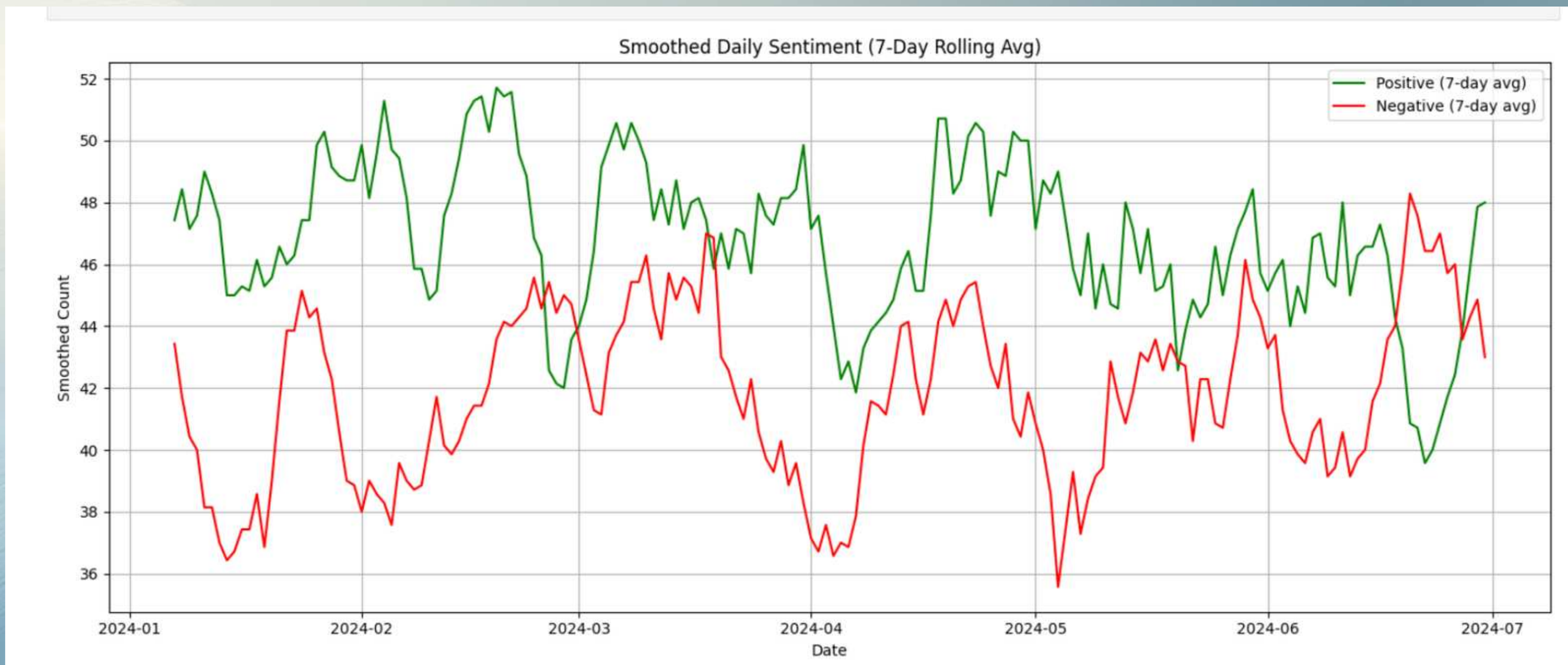
[16]:
```
# Train a Random Forest Classifier with balanced class weights to handle class imbalance.
# Fit the model on the training data, make predictions on the test set,
# and evaluate the model using classification report and accuracy score.

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,accuracy_score,f1_score

model = RandomForestClassifier(class_weight='balanced')
model.fit(X_train_2d, y_train)
y_pred = model.predict(X_test_2d)
accuracy = accuracy_score(y_test, y_pred) * 100
f1 = f1_score(y_test, y_pred) * 100

print(classification_report(y_test, y_pred))
print(f" Accuracy: {accuracy:.2f}%")
print(f" F1 Score: {f1:.2f}%")
```

```
              precision    recall  f1-score   support

           0       0.88      0.85      0.86      1524
           1       0.87      0.89      0.88      1705

    accuracy                           0.87      3229
   macro avg       0.87      0.87      0.87      3229
weighted avg       0.87      0.87      0.87      3229

 Accuracy: 87.43%
 F1 Score: 88.25%
```

[19]:
```python
# Visualizing sentiment trends over time
import matplotlib.pyplot as plt

# Conveting Date into date-time format
df['Date'] = pd.to_datetime(df['Date'])

# Date only (without time)
df['day'] = df['Date'].dt.date

# Grouping by day and sentiment, then counting
df_grouped = df.groupby(['day', 'sentiment']).size().unstack().fillna(0)

# Plotting the two lines
plt.figure(figsize=(14, 6))
plt.plot(df_grouped.index, df_grouped['positive'], label='Positive', color='green')
plt.plot(df_grouped.index, df_grouped['negative'], label='Negative', color='red')

plt.xlabel("Date")
plt.ylabel("Count")
plt.title("Daily Count of Positive and Negative Sentiments")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```


Daily Count of Positive and Negative Sentiments
Positive
Negative
Count
65
60
55
50
45
40
35
30
25
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
Date

[20]:
```
# Another way
df_grouped['positive_rolling'] = df_grouped['positive'].rolling(window=7).mean()
df_grouped['negative_rolling'] = df_grouped['negative'].rolling(window=7).mean()

plt.figure(figsize=(14, 6))
plt.plot(df_grouped.index, df_grouped['positive_rolling'], label='Positive (7-day avg)', color='green')
plt.plot(df_grouped.index, df_grouped['negative_rolling'], label='Negative (7-day avg)', color='red')
plt.xlabel("Date")
plt.ylabel("Smoothed Count")
plt.title("Smoothed Daily Sentiment (7-Day Rolling Avg)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```


Smoothed Daily Sentiment (7-Day Rolling Avg)
Positive (7-day avg)
Negative (7-day avg)
Smoothed Count
52
50
48
46
44
42
40
38
36
2024-01
2024-02
2024-03
2024-04
2024-05
2024-06
2024-07
Date