```
# The news article dataset consists of various news articles with the following key columns:

# title: The title of the news article.
# Lead Paragraph: The introductory paragraph of the article.
# category_level_1 and category_level_2: The broader and more specific categories of the articles.
# sentiment: A column that seems to be intended for sentiment classification, though it currently contains NaN values.
# To create a VADER-based sentiment analysis model, the follwoing will be completed:

# Apply the VADER sentiment analyzer to the text based on Lead Paragraph
# Populate the sentiment column based on VADER's analysis.
# Visualize the distribution of sentiment scores.
```

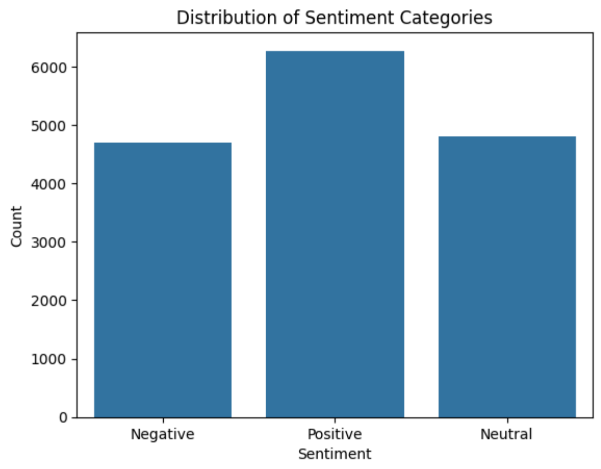
```
pip install vaderSentiment
```

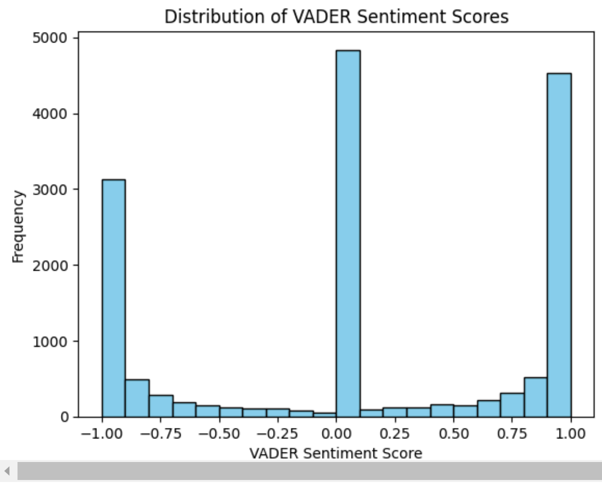
```
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl.metadata (572 bytes)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from vaderSentiment) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (2024.7.4)
Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 2.7 MB/s eta 0:00:00
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import pandas as pd

# Load your CSV file
df = pd.read_csv('/content/MN-DS-news-classification_combined.csv', encoding='ISO-8859-1')

# Initialize the VADER sentiment analyzer
analyzer = SentimentIntensityAnalyzer()

# Apply VADER sentiment analysis to the 'Lead Paragraph' column
df['vader_sentiment'] = df['Lead Paragraph'].apply(lambda x: analyzer.polarity_scores(str(x))['compound'])

# Display the updated dataframe
print(df[['title', 'Lead Paragraph', 'vader_sentiment']].head())
```

```
                                               title  \
0  Virginia mom charged with murder in 2-year-old...
1  2 escaped murder suspects arrested at US-Mexic...
2  Family turns in escaped boy, 13, suspected in ...
3  Mother charged with murder in deaths of 2 youn...
4  Physician, Father and Caretaker of 29 Year Old...

                                      Lead Paragraph  vader_sentiment
0  The Virginia woman whose 2-year-old son was fo...          -0.9848
1  Authorities are trying to determine if anyone ...          -0.9948
2  A 13-year-old suspect in a double homicide who...          -0.9661
3  The mother of two young children found hanging...          -0.9664
4  "One family member said Derek â€can be violen...          -0.9589
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have already applied the VADER sentiment analysis and have the 'vader_sentiment' column in your DataFrame

# Categorize the sentiment scores into Positive, Neutral, and Negative
df['sentiment_category'] = df['vader_sentiment'].apply(lambda score: 'Positive' if score > 0.05
                                                        else ('Negative' if score < -0.05 else 'Neutral'))

# Plot the distribution of sentiment categories
sns.countplot(x='sentiment_category', data=df)
plt.title('Distribution of Sentiment Categories')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()

# Alternatively, you can plot a histogram of the VADER sentiment scores
plt.hist(df['vader_sentiment'], bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of VADER Sentiment Scores')
plt.xlabel('VADER Sentiment Score')
plt.ylabel('Frequency')
plt.show()
```

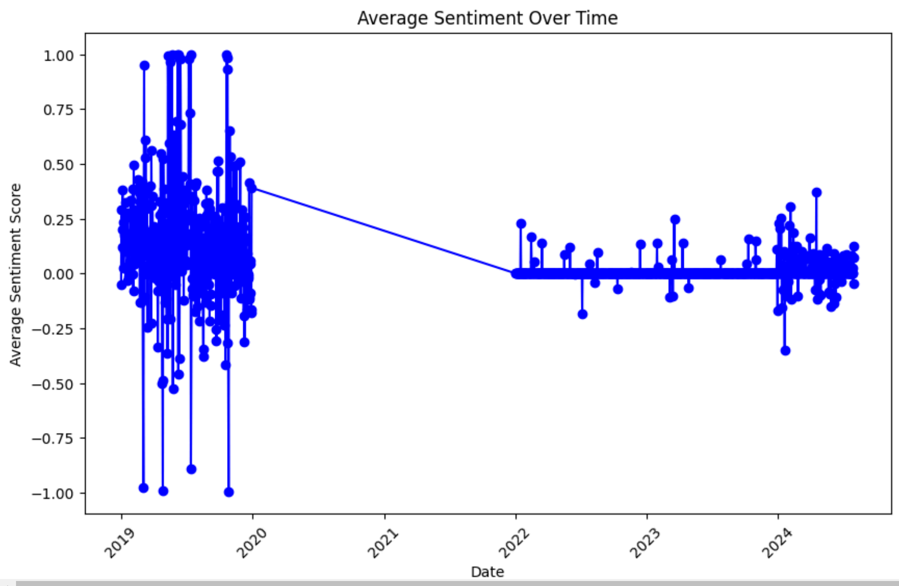




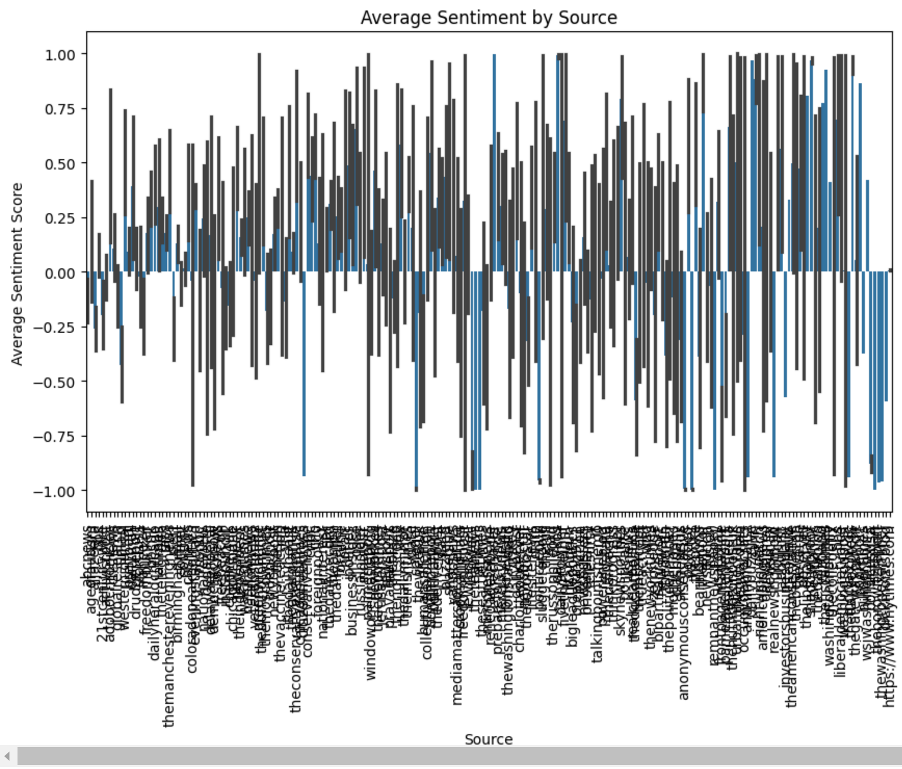
```
# Export the DataFrame to a CSV file
output_file_path = 'vader_sentiment_analysis_results.csv'
df.to_csv(output_file_path, index=False)

print(f"DataFrame successfully exported to {output_file_path}")
```

```
DataFrame successfully exported to vader_sentiment_analysis_results.csv
```

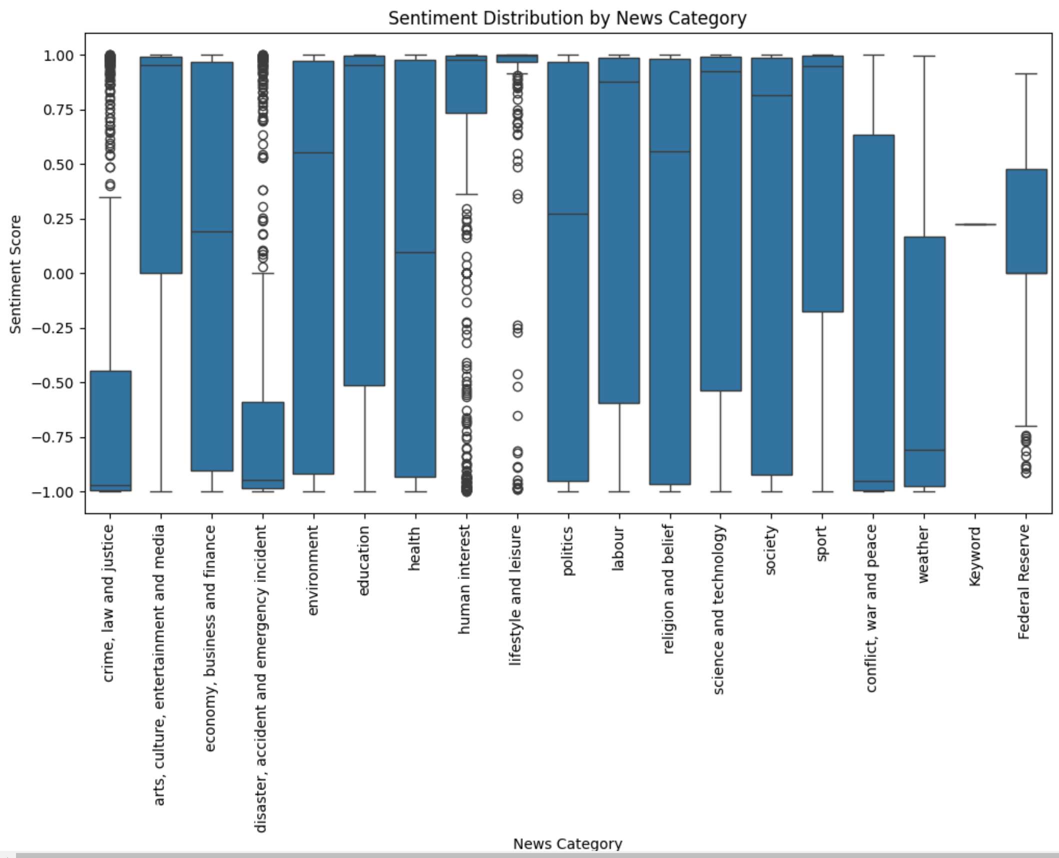
```
# Sentiment Over Time
# Purpose: Visualize how sentiment evolves over time (e.g., by publication date).
# Visualization: Line chart or scatter plot.


# Convert the publication date to a datetime format
df['Publication Date'] = pd.to_datetime(df['Publication Date'])

# Group by date and compute the average sentiment per day
sentiment_over_time = df.groupby(df['Publication Date'].dt.date)['vader_sentiment'].mean()

# Plot the sentiment over time
plt.figure(figsize=(10,6))
plt.plot(sentiment_over_time, marker='o', linestyle='-', color='b')
plt.title('Average Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Average Sentiment Score')
plt.xticks(rotation=45)
plt.show()
```



```
# Sentiment by Source
# Purpose: Compare sentiment across different sources of news or categories.
# Visualization: Bar chart or boxplot.

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np  # Import NumPy

# Plot average sentiment by source
plt.figure(figsize=(10,6))
sns.barplot(x='source', y='vader_sentiment', data=df, estimator=np.mean)
plt.title('Average Sentiment by Source')
plt.xlabel('Source')
plt.ylabel('Average Sentiment Score')
plt.xticks(rotation=90)
plt.show()
```

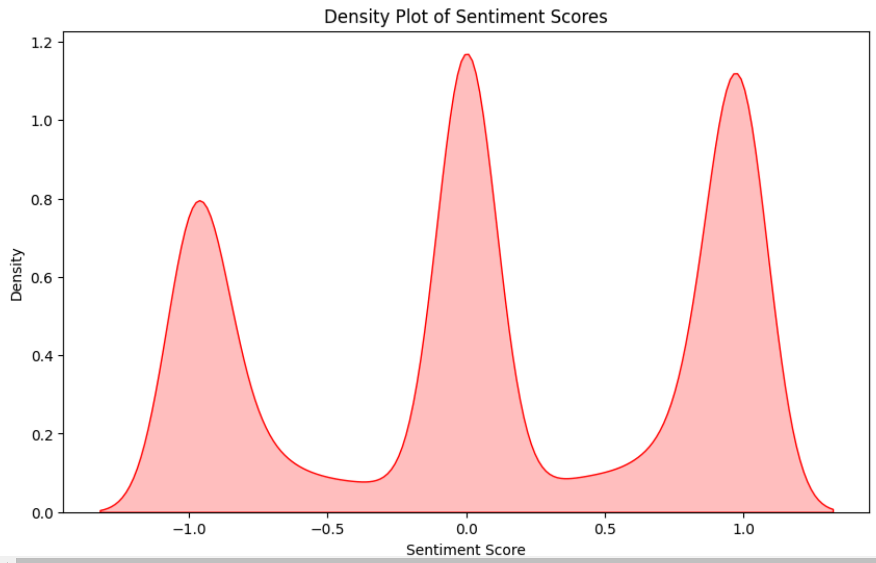


```
#  Sentiment Distribution per Category
# Purpose: Analyze how sentiment varies across different categories of news.
# Visualization: Boxplot or violin plot.

# Plot sentiment distribution by category level 1
plt.figure(figsize=(12,6))
sns.boxplot(x='category_level_1', y='vader_sentiment', data=df)
plt.title('Sentiment Distribution by News Category')
plt.xlabel('News Category')
plt.ylabel('Sentiment Score')
plt.xticks(rotation=90)
plt.show()
```

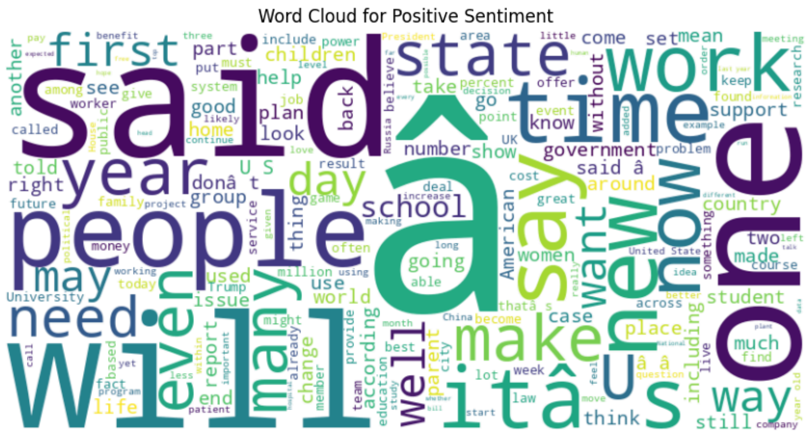


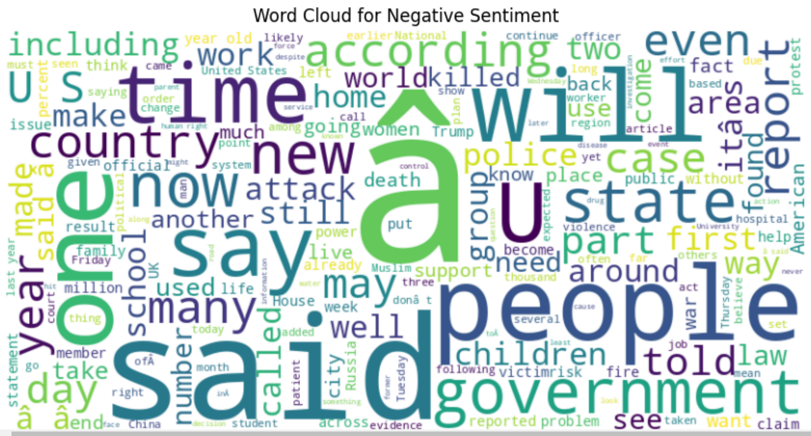
```
# Sentiment Polarity Distribution
# Purpose: Get a deeper look at the polarity distribution across the dataset.
# Visualization: KDE (Kernel Density Estimation) plot.

plt.figure(figsize=(10,6))
sns.kdeplot(df['vader_sentiment'], shade=True, color='r')
plt.title('Density Plot of Sentiment Scores')
plt.xlabel('Sentiment Score')
plt.ylabel('Density')
plt.show()
```

```
<ipython-input-10-d1fc7d61ffa3>:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(df['vader_sentiment'], shade=True, color='r')
```

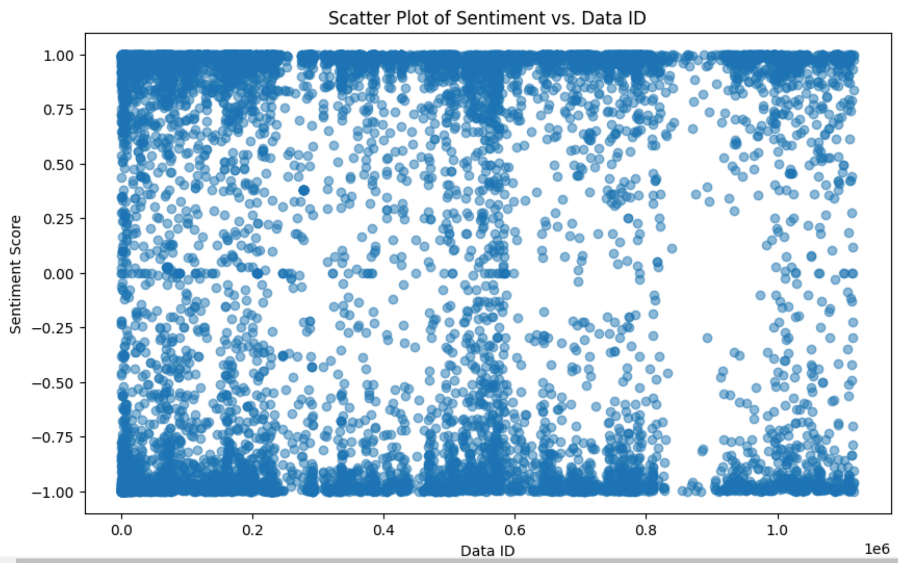


```python
# Word Clouds for Positive and Negative Sentiment
# Purpose: Visualize the most common words in positive and negative articles.
# Visualization: Word cloud.

from wordcloud import WordCloud

# Generate word clouds for positive and negative sentiment
positive_text = ' '.join(df[df['vader_sentiment'] > 0.05]['Lead Paragraph'].dropna())
negative_text = ' '.join(df[df['vader_sentiment'] < -0.05]['Lead Paragraph'].dropna())

# Word cloud for positive sentiment
plt.figure(figsize=(10,5))
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(positive_text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud for Positive Sentiment')
plt.axis('off')
plt.show()

# Word cloud for negative sentiment
plt.figure(figsize=(10,5))
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(negative_text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Word Cloud for Negative Sentiment')
plt.axis('off')
plt.show()
```

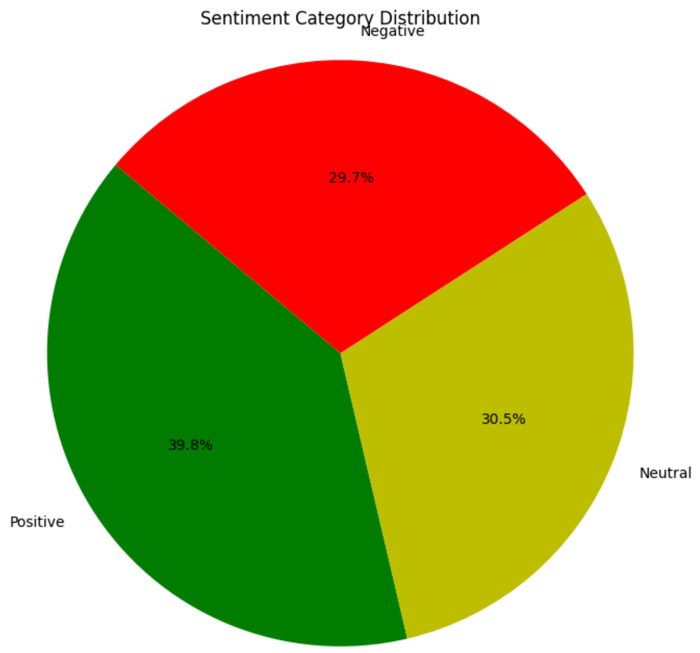




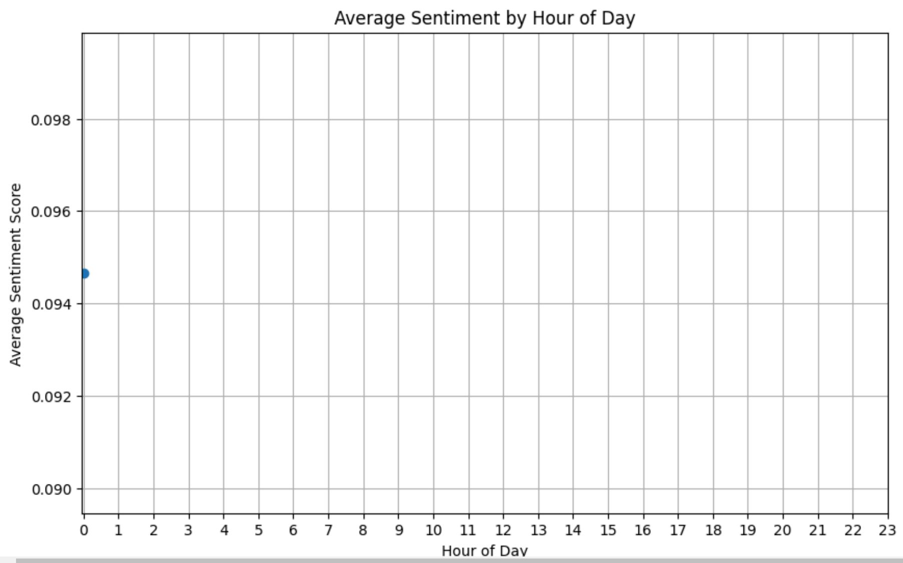
```
# Correlation Between Sentiment and Other Numeric Variables
# Purpose: Identify potential relationships between sentiment and other numerical features.
# Visualization: Scatter plots or correlation heatmaps.

# Example: Scatter plot between sentiment and data_id (or any other numeric column)
plt.figure(figsize=(10,6))
plt.scatter(df['data_id'], df['vader_sentiment'], alpha=0.5)
plt.title('Scatter Plot of Sentiment vs. Data ID')
plt.xlabel('Data ID')
plt.ylabel('Sentiment Score')
plt.show()
```



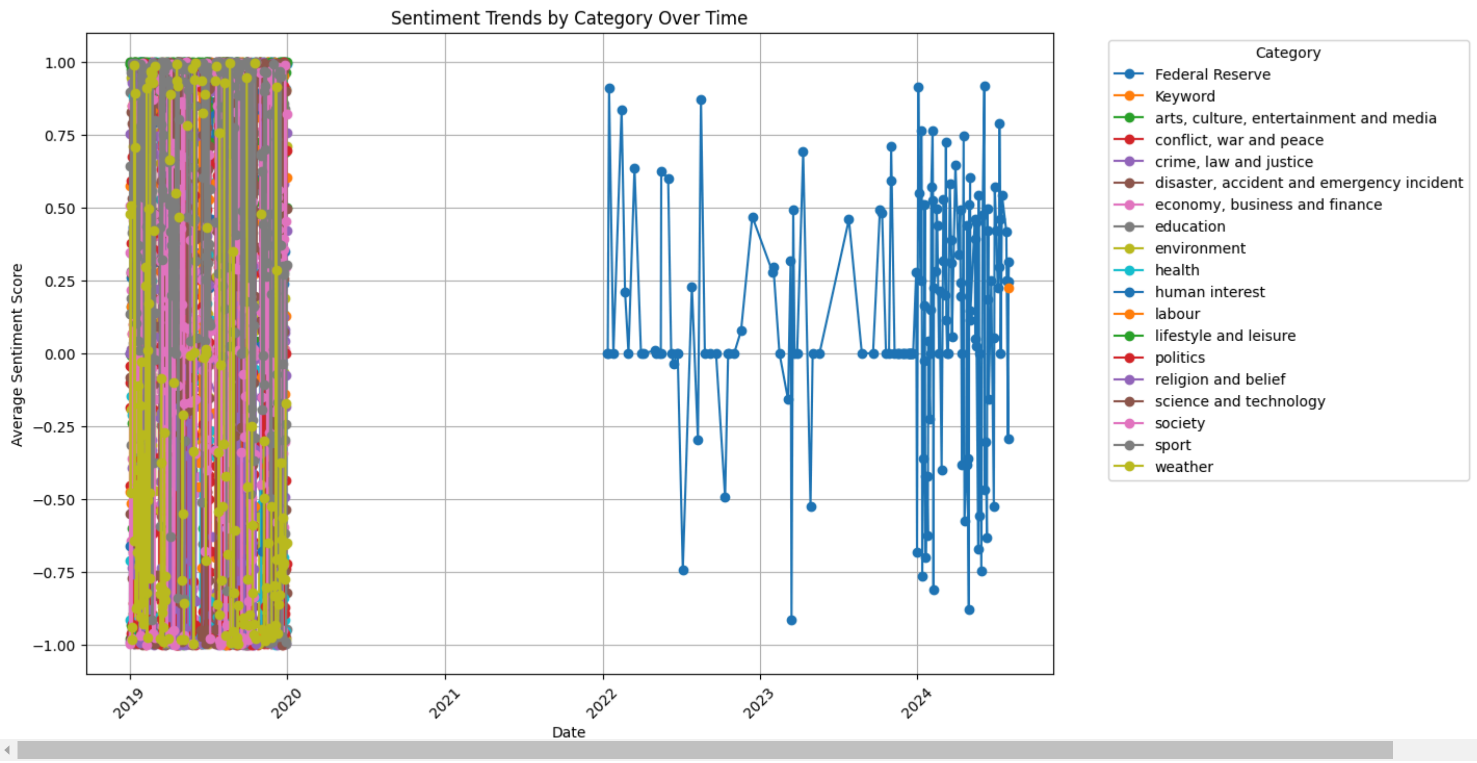
```
# Sentiment Category Comparison
# Purpose: Compare the distributions of positive, neutral, and negative sentiment categories.
# Visualization: Pie chart or stacked bar chart.

# Pie chart of sentiment categories
sentiment_counts = df['sentiment_category'].value_counts()
plt.figure(figsize=(8,8))
plt.pie(sentiment_counts, labels=sentiment_counts.index, autopct='%1.1f%%', startangle=140, colors=['g', 'y', 'r'])
plt.title('Sentiment Category Distribution')
plt.axis('equal')
plt.show()
```



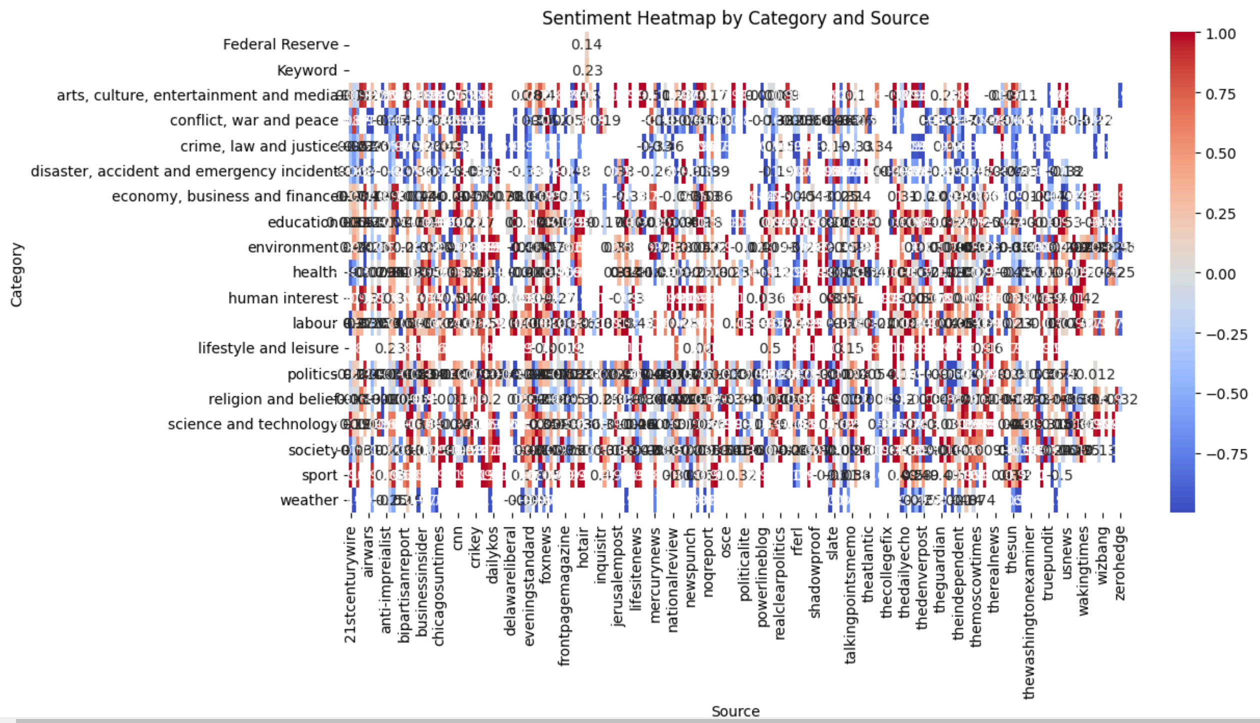
```
# Time of Day or Day of Week Sentiment
# Purpose: Analyze how sentiment varies by time of day or day of the week.
# Visualization: Bar chart or heatmap

# Extract hour from the publication time
df['hour'] = df['Publication Date'].dt.hour

# Group by hour and calculate average sentiment
sentiment_by_hour = df.groupby('hour')['vader_sentiment'].mean()

# Plot sentiment by hour of the day
plt.figure(figsize=(10,6))
plt.plot(sentiment_by_hour.index, sentiment_by_hour.values, marker='o')
plt.title('Average Sentiment by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Average Sentiment Score')
plt.xticks(range(0, 24))
plt.grid(True)
plt.show()
```



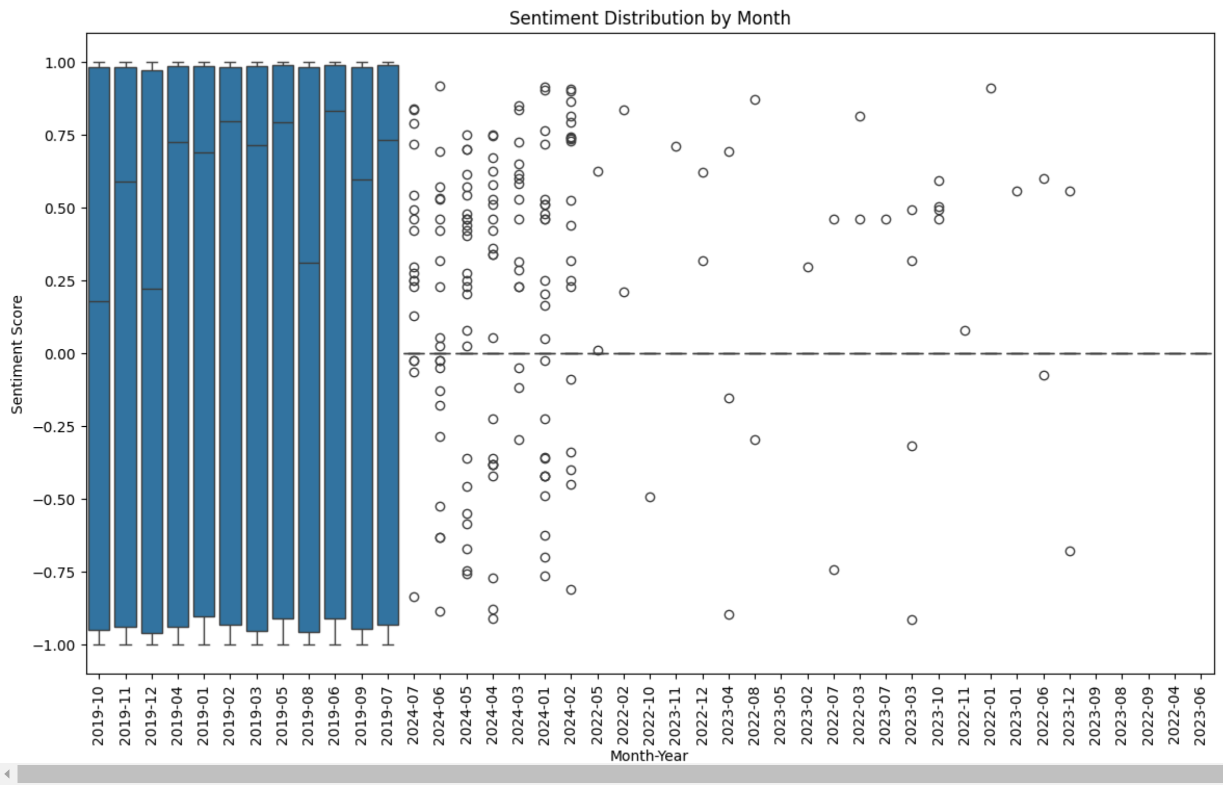
```
# Sentiment Trend per Category Over Time
# Purpose: Analyze how sentiment trends change over time within specific categories (e.g., crime, politics).
# Visualization: Line plot with multiple categories or a heatmap.

# Group by date and category to compute the average sentiment
sentiment_category_over_time = df.groupby([df['Publication Date'].dt.date, 'category_level_1'])['vader_sentiment'].mean().unstack()

# Plot sentiment trends per category over time
sentiment_category_over_time.plot(figsize=(12,8), marker='o')
plt.title('Sentiment Trends by Category Over Time')
plt.xlabel('Date')
plt.ylabel('Average Sentiment Score')
plt.legend(title='Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



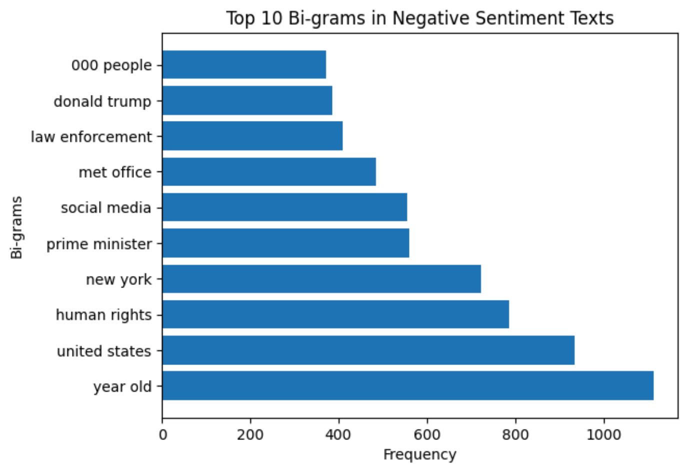
```
# Heatmap of Sentiment Across Categories and Sources
# Purpose: Compare sentiment intensity across different categories and sources.
# Visualization: Heatmap.

# Pivot the data to create a heatmap of sentiment by category and source
sentiment_heatmap_data = df.pivot_table(values='vader_sentiment', index='category_level_1', columns='source', aggfunc='mean')

# Plot the heatmap
plt.figure(figsize=(12,6))
sns.heatmap(sentiment_heatmap_data, annot=True, cmap='coolwarm', center=0)
plt.title('Sentiment Heatmap by Category and Source')
plt.xlabel('Source')
plt.ylabel('Category')
plt.show()
```



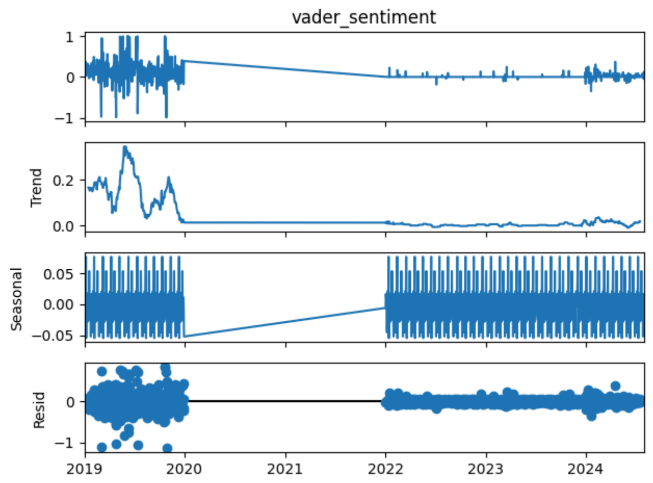
```
# Sentiment Boxplots Grouped by Time Intervals
# Purpose: Compare sentiment variability within different time periods (e.g., monthly, quarterly).
# Visualization: Boxplot grouped by time intervals.

# Extract month and year
df['month_year'] = df['Publication Date'].dt.to_period('M')

# Plot sentiment boxplots grouped by month
plt.figure(figsize=(14,8))
sns.boxplot(x='month_year', y='vader_sentiment', data=df)
plt.title('Sentiment Distribution by Month')
plt.xlabel('Month-Year')
plt.ylabel('Sentiment Score')
plt.xticks(rotation=90)
plt.show()
```



```
# Sentiment-Specific N-Gram Analysis
# Purpose: Identify common phrases (bi-grams, tri-grams) within positive, neutral, or negative sentiment.
# Visualization: Bar chart or word cloud for top n-grams

from sklearn.feature_extraction.text import CountVectorizer

# Function to get n-grams for a specific sentiment
def get_top_n_grams(corpus, n=2, ngram_range=(2,2), top_n=10):
    vec = CountVectorizer(ngram_range=ngram_range, stop_words='english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:top_n]

# Get top 10 bi-grams for negative sentiment
negative_texts = df[df['vader_sentiment'] < -0.05]['Lead Paragraph'].dropna().tolist()
top_bigrams_negative = get_top_n_grams(negative_texts, n=2)

# Plot the top bi-grams for negative sentiment
bigrams, counts = zip(*top_bigrams_negative)
plt.barh(bigrams, counts)
plt.title('Top 10 Bi-grams in Negative Sentiment Texts')
plt.xlabel('Frequency')
plt.ylabel('Bi-grams')
plt.show()
```



```
# Time Series Decomposition of Sentiment
# Purpose: Decompose sentiment trends into seasonal, trend, and residual components.
# Visualization: Time series decomposition plot.

from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the sentiment time series (this example assumes daily sentiment)
sentiment_time_series = df.groupby(df['Publication Date'].dt.date)['vader_sentiment'].mean()
decomposition = seasonal_decompose(sentiment_time_series, model='additive', period=30)

# Plot the decomposition
decomposition.plot()
plt.show()
```



```
# Sentiment Comparison by Length of Article
# Purpose: Compare sentiment across different article lengths (e.g., short vs. long articles).
# Visualization: Boxplot or scatter plot.

# Calculate the length of the lead paragraph
df['paragraph_length'] = df['Lead Paragraph'].apply(lambda x: len(str(x).split()))

# Scatter plot of sentiment by paragraph length
plt.figure(figsize=(10,6))
plt.scatter(df['paragraph_length'], df['vader_sentiment'], alpha=0.5)
plt.title('Sentiment by Paragraph Length')
plt.xlabel('Paragraph Length (words)')
plt.ylabel('Sentiment Score')
plt.grid(True)
plt.show()
```