```
# # Bitcoin Price Prediction Using Multi-Source Sentiment Analysis

# ## Introduction
# This notebook demonstrates the process of predicting Bitcoin price trends using s
# The project uses machine learning techniques to forecast Bitcoin price fluctuatic
```

1. Data Collection

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
pip install yfinance
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.11/dist-pacl
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.11/dist
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.11/dis
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.1
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.11/
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.11/dis
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python
Requirement already satisfied: curl_cffi>=0.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: websockets>=13.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/pyth
Requirement already satisfied: cffi>=1.12.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: certifi>=2024.2.2 in /usr/local/lib/python3.11/
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyth
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pacl
```

```
pip install --upgrade yfinance
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.11/dist-pacl
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.11/dist
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.11/dis
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.1
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.11/
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.11/dis
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python
Requirement already satisfied: curl_cffi>=0.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: websockets>=13.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/pyth
Requirement already satisfied: cffi>=1.12.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: certifi>=2024.2.2 in /usr/local/lib/python3.11/
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyth
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pacl
```

```
import yfinance as yf
import pandas as pd
from datetime import datetime, timedelta

# Try to fetch data and handle exceptions
try:
    # Define the end date (yesterday)
    end_date = (datetime.today() - timedelta(days=1)).strftime('2025-05-15')

    # Define the start date (5 years before yesterday)
    start_date = (datetime.today() - timedelta(days=5*365)).strftime('2019-01-01'

    # Define the ticker symbol for Bitcoin (BTC-USD)
    ticker = 'BTC-USD'

    # Download the Bitcoin data from 5 years ago to yesterday
    bitcoin_data = yf.download(ticker, start=start_date, end=end_date)

    # Display the first few rows of the data
    print(bitcoin_data.head())

    # Save the data to a CSV file
    bitcoin_data.to_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_data_5
    print("Data saved successfully.")

except Exception as e:
    print("An error occurred:", e)
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  1 of 1 completed
Price             Close         High          Low         Open      Volume
Ticker          BTC-USD      BTC-USD      BTC-USD      BTC-USD     BTC-USD
Date
2019-01-01  3843.520020  3850.913818  3707.231201  3746.713379  4324200990
2019-01-02  3943.409424  3947.981201  3817.409424  3849.216309  5244856836
2019-01-03  3836.741211  3935.685059  3826.222900  3931.048584  4530215219
2019-01-04  3857.717529  3865.934570  3783.853760  3832.040039  4847965467
2019-01-05  3845.194580  3904.903076  3836.900146  3851.973877  5137609824
Data saved successfully.
```

Data cleaning

```
import pandas as pd

# Load the Bitcoin data from the CSV file
bitcoin_data = pd.read_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_dat

# Display the first few rows of the DataFrame
print(bitcoin_data.head())
```

```
        Price              Close               High                Low  \
0      Ticker            BTC-USD            BTC-USD            BTC-USD
1        Date                NaN                NaN                NaN
2  2019-01-01   3843.52001953125  3850.913818359375  3707.231201171875
3  2019-01-02  3943.409423828125  3947.981201171875  3817.409423828125
4  2019-01-03    3836.7412109375   3935.68505859375  3826.222900390625

                Open      Volume
0            BTC-USD     BTC-USD
1                NaN         NaN
2   3746.71337890625  4324200990
3   3849.21630859375  5244856836
4  3931.048583984375  4530215219
```

```
import pandas as pd

# Load the Bitcoin data from the CSV file, skipping the first two rows
bitcoin_data = pd.read_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_dat

# Display the first few rows to understand the structure
print("Original DataFrame:")
print(bitcoin_data.head())

# Check the number of columns
print("\nNumber of columns:", bitcoin_data.shape[1])

# Rename columns based on the correct header structure
# Adjust the number of columns to match the DataFrame
bitcoin_data.columns = ['Date', 'Close', 'High', 'Low', 'Open','Volume']

# Check for missing values
print("\nMissing values in the DataFrame:")
print(bitcoin_data.isnull().sum())

# Convert 'Date' to datetime format and set it as the index
bitcoin_data['Date'] = pd.to_datetime(bitcoin_data['Date'])
bitcoin_data.set_index('Date', inplace=True)
```

```
# Convert relevant columns to numeric types
bitcoin_data['Volume'] = pd.to_numeric(bitcoin_data['Volume'], errors='coerce')
bitcoin_data['Close'] = pd.to_numeric(bitcoin_data['Close'], errors='coerce')
bitcoin_data['High'] = pd.to_numeric(bitcoin_data['High'], errors='coerce')
bitcoin_data['Low'] = pd.to_numeric(bitcoin_data['Low'], errors='coerce')
bitcoin_data['Open'] = pd.to_numeric(bitcoin_data['Open'], errors='coerce')

# Check if 'Volume' column exists and convert it to numeric
# Since the 'Volume' column is missing, we will not attempt to convert it
if 'Volume' in bitcoin_data.columns:
    bitcoin_data['Volume'] = pd.to_numeric(bitcoin_data['Volume'], errors='coerce
else:
    print("\n'Volume' column is missing from the DataFrame.")

# Display the cleaned DataFrame
print("\nCleaned Bitcoin Data:")
print(bitcoin_data.head())
```

```
Original DataFrame:
         Date  Unnamed: 1  Unnamed: 2  Unnamed: 3  Unnamed: 4  Unnamed: 5
0  2019-01-01  3843.520020  3850.913818  3707.231201  3746.713379  4324200990
1  2019-01-02  3943.409424  3947.981201  3817.409424  3849.216309  5244856836
2  2019-01-03  3836.741211  3935.685059  3826.222900  3931.048584  4530215219
3  2019-01-04  3857.717529  3865.934570  3783.853760  3832.040039  4847965467
4  2019-01-05  3845.194580  3904.903076  3836.900146  3851.973877  5137609824

Number of columns: 6

Missing values in the DataFrame:
Date      0
Close     0
High      0
Low       0
Open      0
Volume    0
dtype: int64

Cleaned Bitcoin Data:
                  Close         High          Low         Open      Volume
Date
2019-01-01  3843.520020  3850.913818  3707.231201  3746.713379  4324200990
2019-01-02  3943.409424  3947.981201  3817.409424  3849.216309  5244856836
2019-01-03  3836.741211  3935.685059  3826.222900  3931.048584  4530215219
2019-01-04  3857.717529  3865.934570  3783.853760  3832.040039  4847965467
2019-01-05  3845.194580  3904.903076  3836.900146  3851.973877  5137609824
```

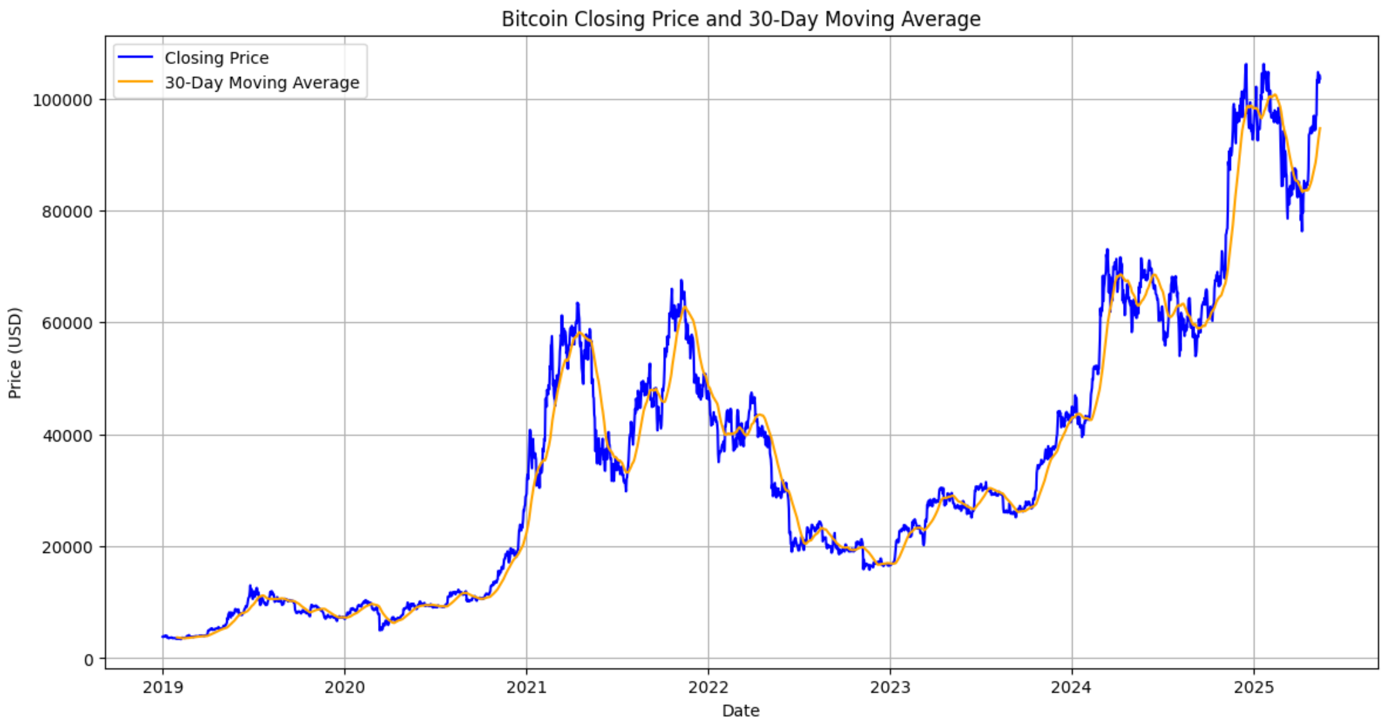
```
# Display summary statistics
print("\nSummary statistics:")
print(bitcoin_data.describe())
```

```
Summary statistics:
               Close           High            Low           Open  \
count    2326.000000    2326.000000    2326.000000    2326.000000
mean    35016.130543   35729.787872   34191.865275   34975.019534
std     25815.669664   26320.056664   25232.387577   25786.187839
min      3399.471680    3427.945557    3391.023682    3401.376465
25%     10954.288818   11204.855225   10722.320557   10939.149414
50%     29044.939453   29440.277344   28629.346680   29034.909180
75%     51640.022461   52346.241211   50037.632812   51545.577148
max    106146.265625  109114.882812  105291.734375  106147.296875

             Volume
count  2.326000e+03
mean   3.110996e+10
std    1.947245e+10
min    4.324201e+09
25%    1.813062e+10
50%    2.726698e+10
75%    3.864478e+10
max    3.509679e+11
```

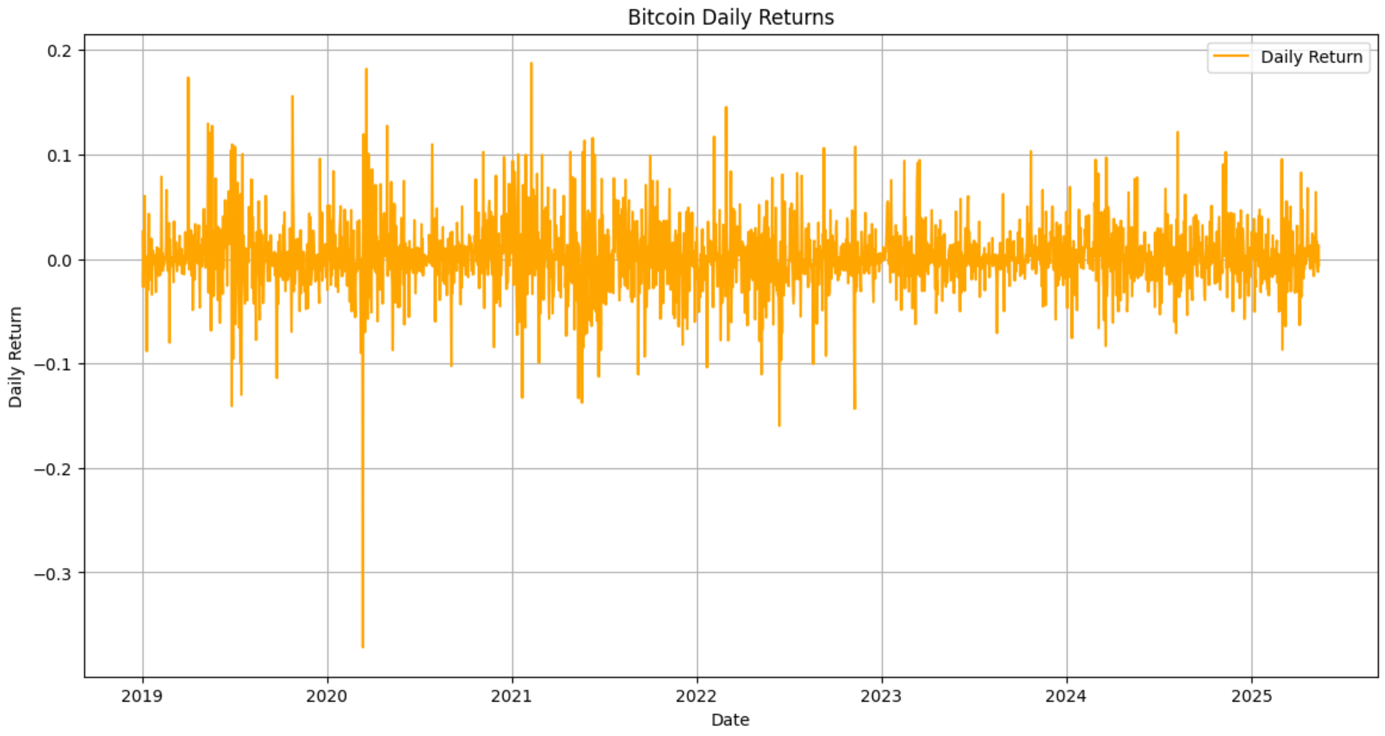
EDA

```
# Calculate the 30-day moving average
bitcoin_data['30_MA'] = bitcoin_data['Close'].rolling(window=30).mean()

# Calculate daily returns
bitcoin_data['Daily Return'] = bitcoin_data['Close'].pct_change()
```

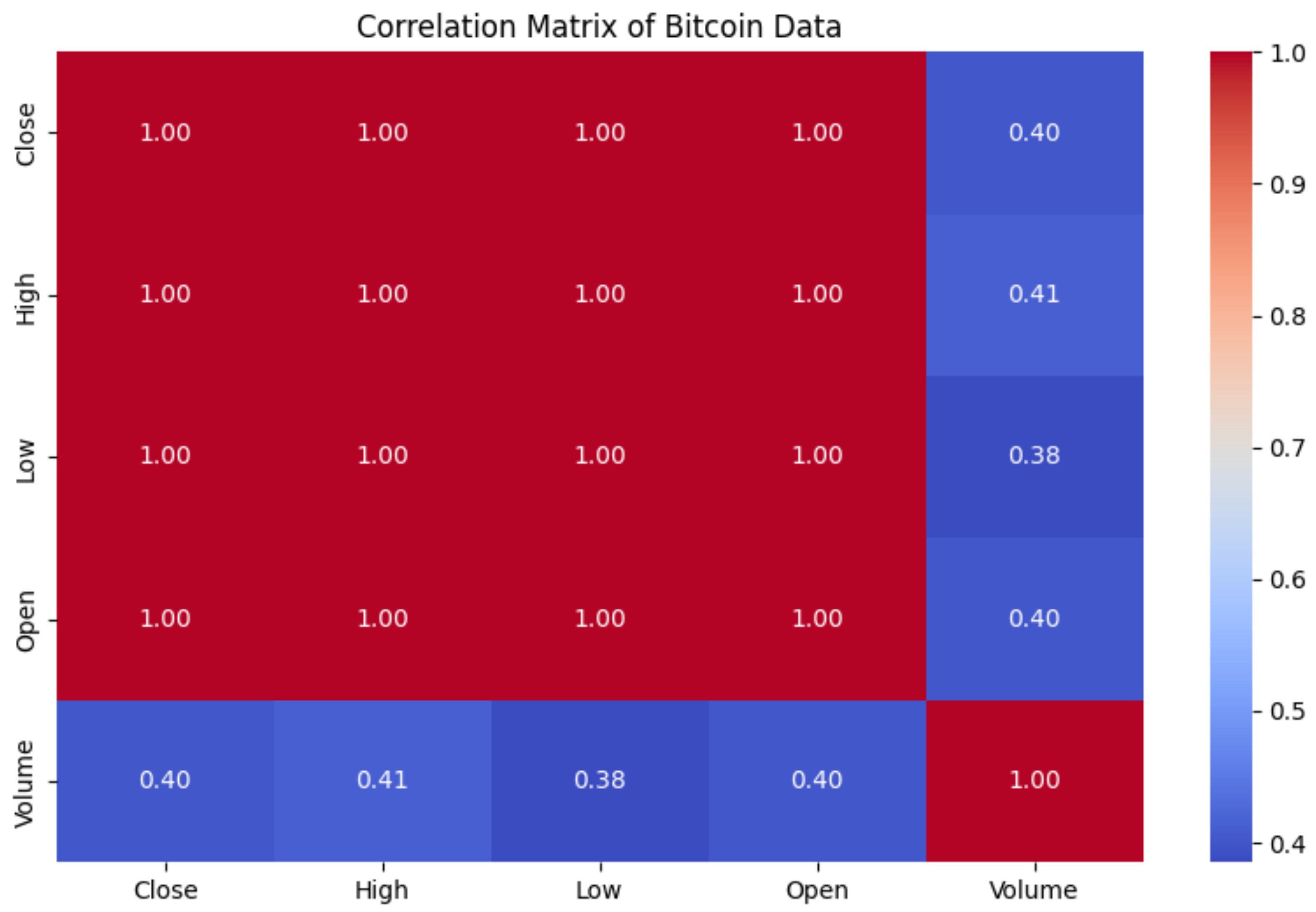
```
import matplotlib.pyplot as plt

# Plot the closing price and the 30-day moving average
plt.figure(figsize=(14, 7))
plt.plot(bitcoin_data['Close'], label='Closing Price', color='blue')
plt.plot(bitcoin_data['30_MA'], label='30-Day Moving Average', color='orange')
plt.title('Bitcoin Closing Price and 30-Day Moving Average')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid()
plt.show()
```



```
# Plot daily returns
plt.figure(figsize=(14, 7))
plt.plot(bitcoin_data['Daily Return'], label='Daily Return', color='orange')
plt.title('Bitcoin Daily Returns')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.legend()
plt.grid()
plt.show()
```



```
import seaborn as sns

# Correlation matrix of numerical columns
plt.figure(figsize=(10, 6))
correlation_matrix = bitcoin_data[[ 'Close', 'High', 'Low', 'Open','Volume']].cor
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Bitcoin Data')
plt.show()
```



Data Collection - Fetching Wikipedia Edits

```
import requests

def fetch_wikipedia_edits(page_title):
    url = "https://en.wikipedia.org/w/api.php"
    params = {
        'action': 'query',
        'prop': 'revisions',
        'titles': page_title,
        'rvprop': 'timestamp|comment',
        'format': 'json',
        'rvlimit': 'max'  # You can adjust the limit as needed
    }

    response = requests.get(url, params=params)
    data = response.json()
    return data

# Fetch edits for the Bitcoin Wikipedia page
wikipedia_data = fetch_wikipedia_edits("Bitcoin")
```

```
import pandas as pd

# Load the Bitcoin data
bitcoin_data = pd.read_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_dat

# Correct the column names based on the structure of the data
bitcoin_data.columns = ['Date', 'Close', 'High', 'Low', 'Open', 'Volume']

# Convert 'Date' to datetime format
bitcoin_data['Date'] = pd.to_datetime(bitcoin_data['Date'])
```

Data Exploration - Exploring Wikipedia Edits

```
# Print the structure of wikipedia_data
print(wikipedia_data)
```

```
{'continue': {'rvcontinue': '20231122092017|1186318603', 'continue': '||'}, '
```

```
# Display the first few rows of the Bitcoin data
print(bitcoin_data.head())
```

```
         Date        Close         High          Low         Open      Volume
0 2019-01-01  3843.520020  3850.913818  3707.231201  3746.713379  4324200990
1 2019-01-02  3943.409424  3947.981201  3817.409424  3849.216309  5244856836
2 2019-01-03  3836.741211  3935.685059  3826.222900  3931.048584  4530215219
3 2019-01-04  3857.717529  3865.934570  3783.853760  3832.040039  4847965467
4 2019-01-05  3845.194580  3904.903076  3836.900146  3851.973877  5137609824
```

Sentiment Analysis Function

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import pandas as pd

# Download the VADER lexicon
nltk.download('vader_lexicon')

def analyze_sentiment(data):
    analyzer = SentimentIntensityAnalyzer()
    sentiment_scores = []

    if 'query' in data and 'pages' in data['query']:
        for revision in data['query']['pages'].values():
            if 'revisions' in revision:
                for rev in revision['revisions']:
                    comment = rev.get('comment', '')
                    sentiment = analyzer.polarity_scores(comment)['compound']

                    # Consider neutral sentiments as negative
                    if -0.05 < sentiment < 0.05:
                        sentiment = -0.1  # Set neutral sentiment to -0.1

                    timestamp = rev['timestamp']
                    sentiment_scores.append({'timestamp': timestamp, 'sentiment':

    return sentiment_scores
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

Sentiment analysis on Wikipedia edits

```
# Perform sentiment analysis on Wikipedia edits
wikipedia_sentiments = analyze_sentiment(wikipedia_data)

# Convert sentiment data to DataFrame
wikipedia_sentiments_df = pd.DataFrame(wikipedia_sentiments)

# Rename the sentiment column
wikipedia_sentiments_df.rename(columns={'sentiment': 'wikipedia_sentiment'}, inpl

# Convert timestamps to datetime objects
wikipedia_sentiments_df['timestamp'] = pd.to_datetime(wikipedia_sentiments_df['ti

# Extract only the date part (ignore the time part)
wikipedia_sentiments_df['date'] = wikipedia_sentiments_df['timestamp'].dt.date

# Save the DataFrame to a CSV file
wikipedia_sentiments_df.to_csv('wikipedia_sentiments.csv', index=False)

# Display the result
print(wikipedia_sentiments_df)
```

```
                     timestamp  wikipedia_sentiment        date
0   2025-05-13 05:43:42+00:00              -0.1000  2025-05-13
1   2025-05-05 22:48:20+00:00              -0.1000  2025-05-05
2   2025-04-30 13:19:13+00:00              -0.1000  2025-04-30
3   2025-04-29 13:02:14+00:00              -0.2698  2025-04-29
4   2025-04-29 12:43:34+00:00               0.5267  2025-04-29
..                         ...                  ...         ...
495 2023-11-22 10:32:07+00:00              -0.1000  2023-11-22
496 2023-11-22 10:27:15+00:00              -0.1000  2023-11-22
497 2023-11-22 10:18:50+00:00              -0.1000  2023-11-22
498 2023-11-22 09:22:40+00:00              -0.1000  2023-11-22
499 2023-11-22 09:21:02+00:00              -0.1000  2023-11-22

[500 rows x 3 columns]
```

Merge Datasets

```
# Load the Bitcoin price data
bitcoin_data = pd.read_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_dat

# Correct the column names based on the structure of the data
bitcoin_data.columns = ['Date', 'Close', 'High', 'Low', 'Open', 'Volume']

# Convert 'Date' to datetime format
bitcoin_data['Date'] = pd.to_datetime(bitcoin_data['Date'])


# Ensure 'date' in wikipedia_sentiments_df is datetime64[ns] and properly formatt
wikipedia_sentiments_df['date'] = pd.to_datetime(wikipedia_sentiments_df['date'])

# Merge the two datasets based on the 'Date' column
merged_data = pd.merge(bitcoin_data, wikipedia_sentiments_df[['date', 'wikipedia_

# Drop 'date' column from the merged data
merged_data.drop(columns=['date'], inplace=True)

# Fill missing sentiment values with 0 (neutral sentiment)
merged_data['wikipedia_sentiment'] = merged_data['wikipedia_sentiment'].fillna(0)
```

Adjust Sentiment Values

```
# Handling Neutral Sentiment by converting neutral sentiments to negative
merged_data['wikipedia_sentiment'] = merged_data['wikipedia_sentiment'].apply(lam

# Add a 'tomorrow_price' column (next day's closing price)
merged_data['tomorrow_price'] = merged_data['Close'].shift(-1)

# Rename columns for better clarity
merged_data.rename(columns={'Close': 'closing_price'}, inplace=True)
```

```
import os
import pandas as pd

# Assuming merged_data is your DataFrame containing the data you want to save

# Define the path where you want to save the CSV file
save_path = '/content/drive/MyDrive/Final_Project_Docs/bitcoin_data_5_years.csv'

# Create the directory if it doesn't exist
os.makedirs(os.path.dirname(save_path), exist_ok=True)

# Save the merged data to the specified CSV file
merged_data.to_csv(save_path, index=False)

# Display a message indicating that the file has been saved
print(f"Merged data saved to '{save_path}'.")
```

```
Merged data saved to '/content/drive/MyDrive/Final_Project_Docs/bitcoin_data_
```

EDA

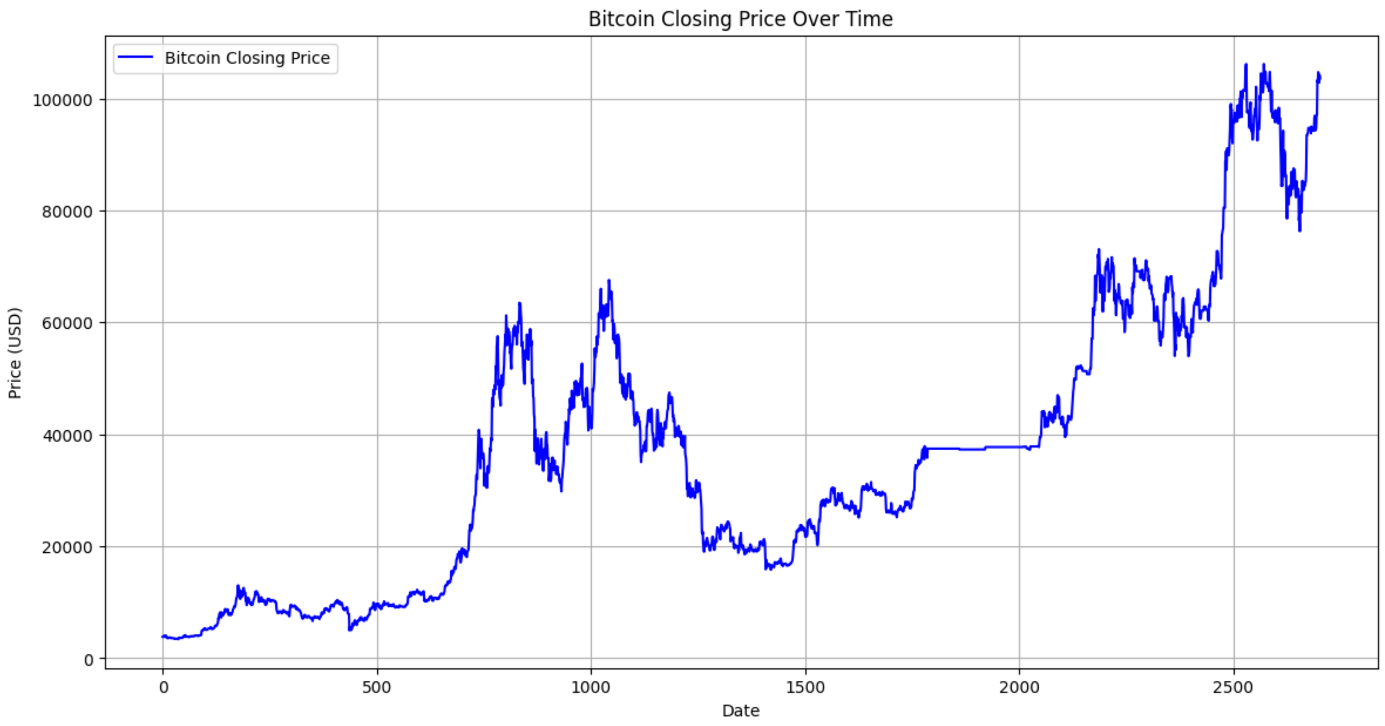
```python
# Summary statistics
print("\nSummary statistics:")
print(merged_data.describe())
```

```
Summary statistics:
                                Date  closing_price           High  \
count                           2703    2703.000000    2703.000000
mean   2022-06-15 03:52:48.479467264   36890.252222   37609.019791
min              2019-01-01 00:00:00    3399.471680    3427.945557
25%              2020-11-06 12:00:00   15311.608887   15671.862305
50%              2022-09-13 00:00:00   35813.812500   37154.601562
75%              2023-11-28 00:00:00   51743.324219   52362.890625
max              2025-05-14 00:00:00  106146.265625  109114.882812
std                              NaN   25395.030973   25884.011552

                 Low           Open        Volume  wikipedia_sentiment  \
count    2703.000000    2703.000000  2.703000e+03          2703.000000
mean    36029.438759   36787.006679  3.055649e+10            -0.090743
min      3391.023682    3401.376465  4.324201e+09            -0.743000
25%     14583.656738   15062.331543  1.822757e+10            -0.100000
50%     34616.691406   35756.554688  2.577587e+10            -0.100000
75%     50824.441406   51845.714844  3.720126e+10            -0.100000
max    105291.734375  106147.296875  3.509679e+11             0.735100
std     24814.239231   25354.159386  1.936839e+10             0.083051

       tomorrow_price
count     2702.000000
mean     36902.482693
min       3399.471680
25%      15369.128418
50%      35838.095703
75%      51748.367188
max     106146.265625
std      25391.768115
```

```python
# Correlation between columns (e.g., between Bitcoin Close price and sentiment)
correlation = merged_data[['closing_price', 'wikipedia_sentiment']].corr()
print("\nCorrelation between 'Closing Price' and 'Sentiment':")
print(correlation)
```

```
Correlation between 'Closing Price' and 'Sentiment':
                     closing_price  wikipedia_sentiment
closing_price             1.000000             0.097234
wikipedia_sentiment       0.097234             1.000000
```

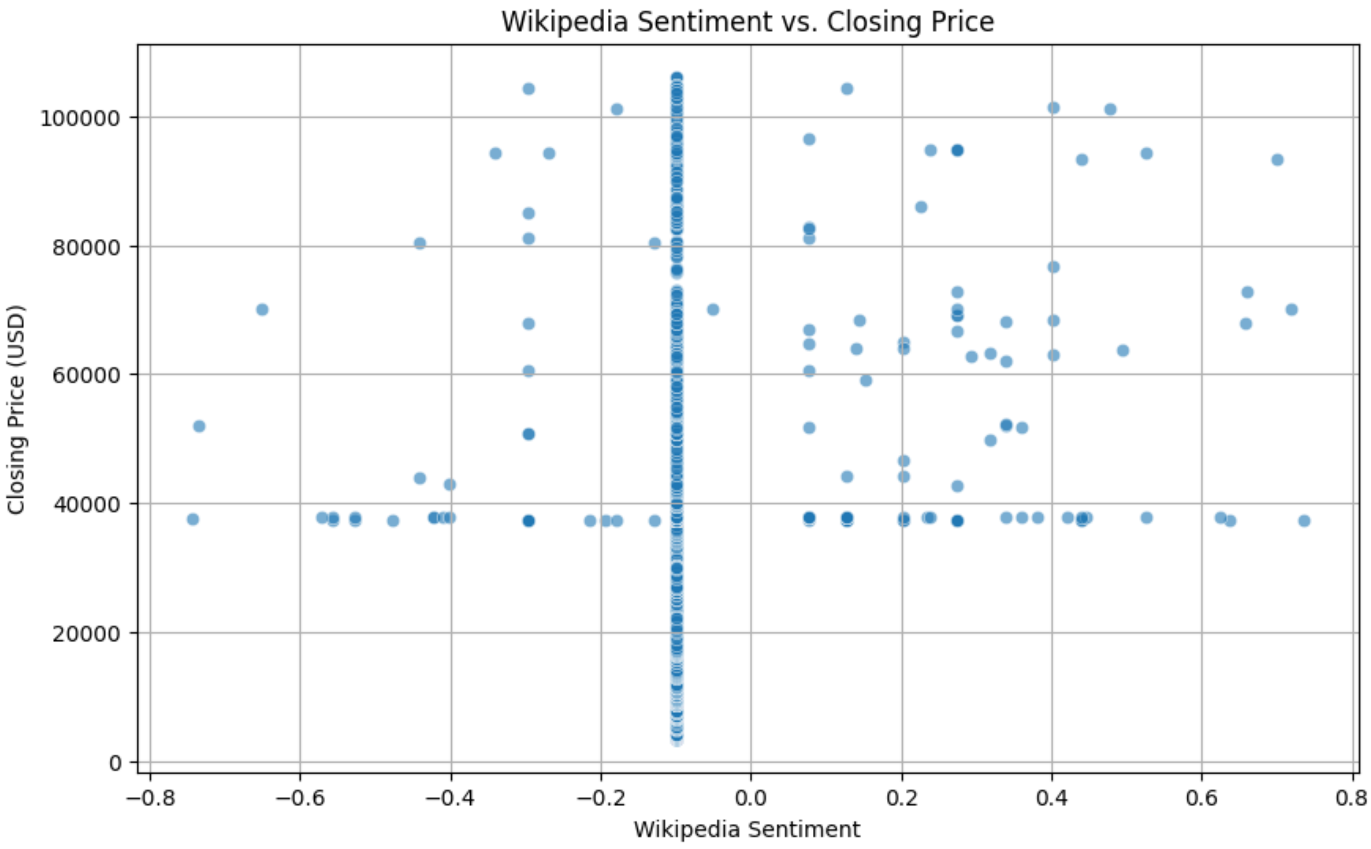
```
import matplotlib.pyplot as plt

# Plot Bitcoin Closing Price Over Time
plt.figure(figsize=(14, 7))
plt.plot(merged_data['closing_price'], label='Bitcoin Closing Price', color='blue
plt.title('Bitcoin Closing Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True)
plt.show()
```


Bitcoin Closing Price Over Time
Bitcoin Closing Price
Price (USD)
Date
0
20000
40000
60000
80000
100000
0
500
1000
1500
2000
2500

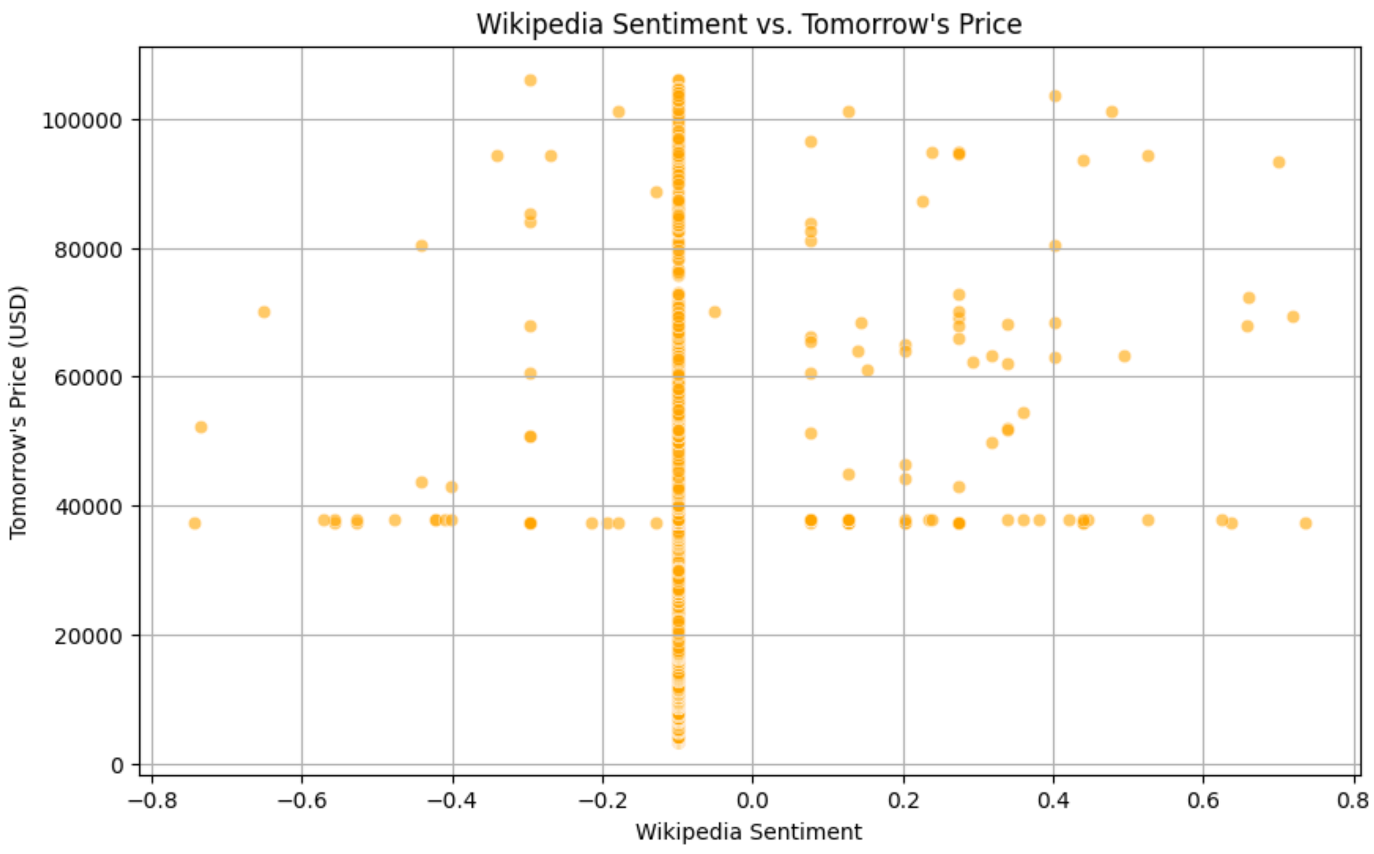
```
# Correlation matrix
correlation_matrix = merged_data[['closing_price', 'wikipedia_sentiment', 'tomorr
print("\nCorrelation matrix:")
print(correlation_matrix)
```

```

Correlation matrix:
                     closing_price  wikipedia_sentiment  tomorrow_price
closing_price             1.000000             0.097234        0.998850
wikipedia_sentiment       0.097234             1.000000        0.097493
tomorrow_price            0.998850             0.097493        1.000000
```

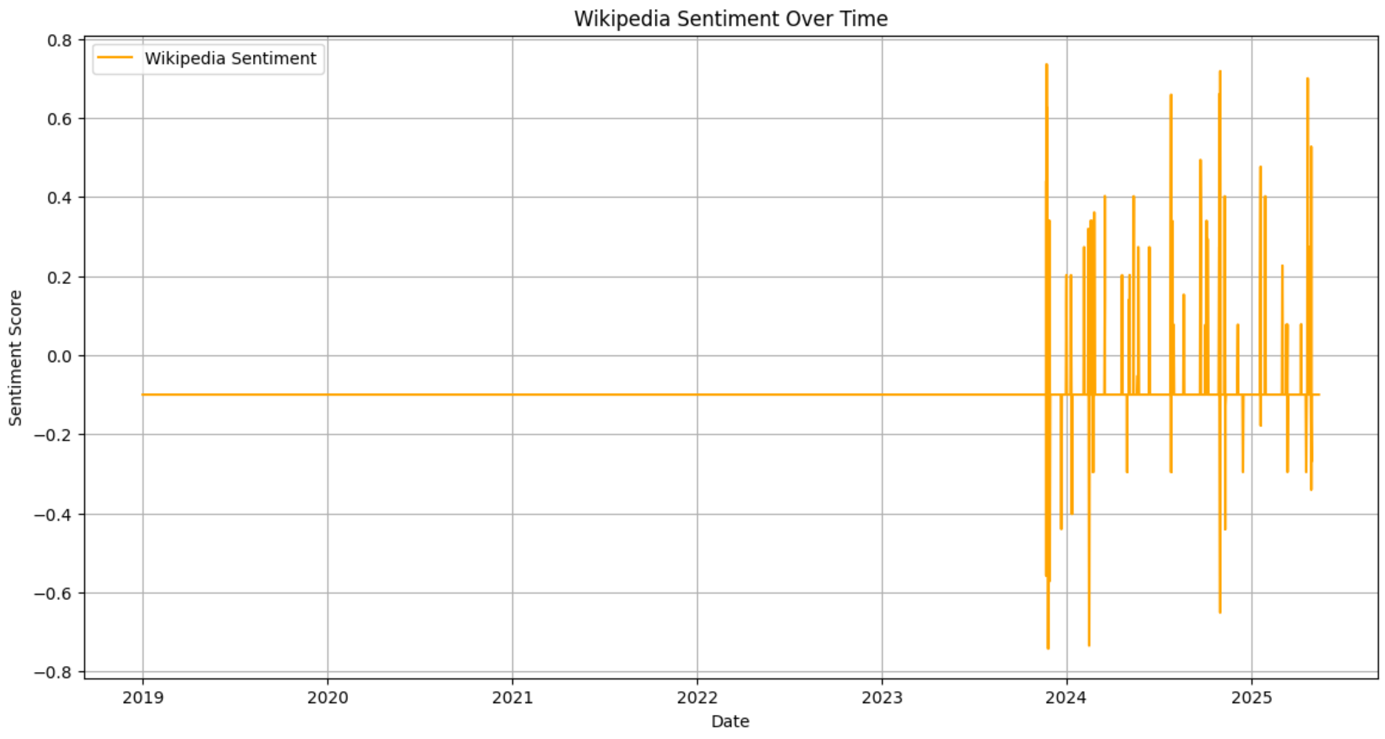
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(data=merged_data, x='wikipedia_sentiment', y='closing_price', alp
plt.title('Wikipedia Sentiment vs. Closing Price')
plt.xlabel('Wikipedia Sentiment')
plt.ylabel('Closing Price (USD)')
plt.grid(True)
plt.show()
```



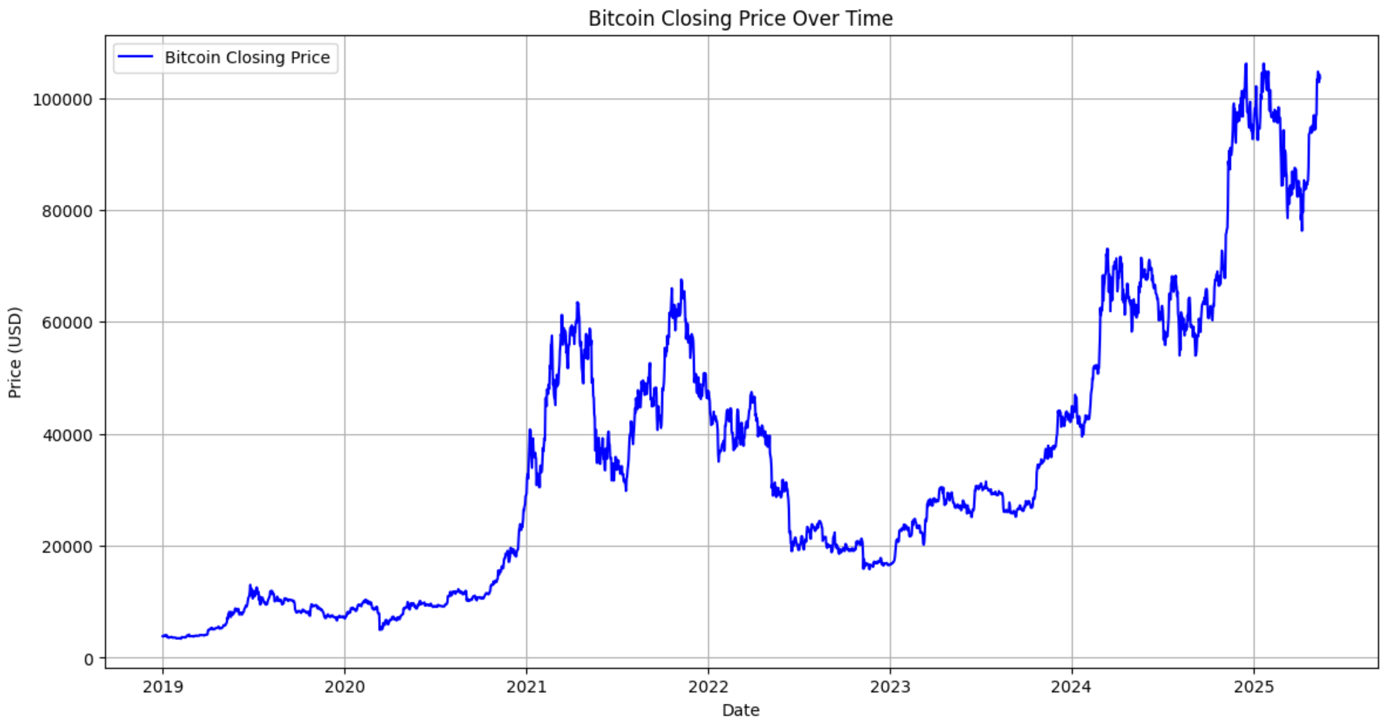
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=merged_data, x='wikipedia_sentiment', y='tomorrow_price', al
plt.title('Wikipedia Sentiment vs. Tomorrow\'s Price')
plt.xlabel('Wikipedia Sentiment')
plt.ylabel('Tomorrow\'s Price (USD)')
plt.grid(True)
plt.show()
```



```
plt.figure(figsize=(14, 7))
plt.plot(merged_data['Date'], merged_data['wikipedia_sentiment'], label='Wikipedi
plt.title('Wikipedia Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Sentiment Score')
plt.legend()
plt.grid(True)
plt.show()
```



```
plt.figure(figsize=(14, 7))
plt.plot(merged_data['Date'], merged_data['closing_price'], label='Bitcoin Closin
plt.title('Bitcoin Closing Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True)
plt.show()
```



```
# Categorize sentiment
def categorize_sentiment(sentiment):
    if sentiment > 0.05:
        return 'Positive'
    elif sentiment < -0.05:
        return 'Negative'
    else:
        return 'Neutral'

merged_data['sentiment_category'] = merged_data['wikipedia_sentiment'].apply(cate

# Group by sentiment category and calculate mean closing price and tomorrow's pri
grouped_data = merged_data.groupby('sentiment_category')[['closing_price', 'tomor
print("\nAverage Closing Price and Tomorrow's Price by Sentiment Category:")
print(grouped_data)
```

```
Average Closing Price and Tomorrow's Price by Sentiment Category:
                    closing_price  tomorrow_price
sentiment_category
Negative             36239.901520    36249.956095
Positive             54177.635682    54241.046596
```

```
# Summary statistics by sentiment category
summary_stats = merged_data.groupby('sentiment_category')[['closing_price', 'tomo
print("\nSummary Statistics by Sentiment Category:")
print(summary_stats)
```

```
Summary Statistics by Sentiment Category:
                   closing_price                                          \
                           count          mean           std          min
sentiment_category
Negative                  2605.0  36239.901520  25327.460320  3399.471680
Positive                    98.0  54177.635682  20742.981155  37289.621094

                                                                           \
                             25%           50%           75%           max
sentiment_category
Negative            11970.478516  33855.328125  51093.652344  106146.265625
Positive            37720.281250  37775.683594  68169.916016  104408.070312

                   tomorrow_price                                          \
                            count          mean           std          min
sentiment_category
Negative                   2604.0  36249.956095  25321.721626  3399.471680
Positive                     98.0  54241.046596  20774.007083  37289.621094


                             25%           50%           75%           max
sentiment_category
Negative            11986.044678  33876.187500  51121.912109  106146.265625
Positive            37720.281250  37813.939453  67925.187500  103703.210938
```

Collection of The Guaurdian News

```
import requests
import pandas as pd
from datetime import datetime, timedelta
from tqdm import tqdm
import time

# Configuration
API_KEY = "d40a5c30-8b6d-4f17-a4ba-3deab348a109"
START_DATE = "2019-01-01"
END_DATE = datetime.now().strftime("%Y-%m-%d")
OUTPUT_FILE = "guardian_bitcoin_simple.csv"
```

```
# Simplified query that works with Guardian API
QUERY = "Bitcoin OR BTC OR cryptocurrency"

def fetch_guardian_articles(from_date, to_date):
    """Fetch articles with simplified query"""
    all_articles = []
    page = 1
    total_pages = 1

    while page <= total_pages:
        url = "https://content.guardianapis.com/search"
        params = {
            "q": QUERY,
            "from-date": from_date,
            "to-date": to_date,
            "api-key": API_KEY,
            "page": page,
            "page-size": 50,
            "show-fields": "headline,trailText,body,byline,wordcount",
            "order-by": "newest"
        }

        try:
            response = requests.get(url, params=params, timeout=15)
            data = response.json()

            if 'response' not in data:
                print(f"Unexpected API response structure on page {page}")
                break

            articles = data['response'].get('results', [])
            all_articles.extend(articles)

            if page == 1:
                total_pages = min(10, data['response'].get('pages', 1))
                print(f"Found {data['response']['total']} results for {from_date}

            page += 1
            time.sleep(0.5)

        except Exception as e:
            print(f"Error on page {page}: {str(e)}")
            break

    return all_articles
```

```
def main():
    print(f"Scraping Bitcoin news from {START_DATE} to {END_DATE}")
    print(f"Using query: {QUERY}")

    # Create monthly batches
    date_ranges = pd.date_range(START_DATE, END_DATE, freq='1M')
    date_ranges = [d.strftime('%Y-%m-%d') for d in date_ranges]
    if date_ranges[-1] != END_DATE:
        date_ranges.append(END_DATE)

    all_articles = []
    for i in tqdm(range(len(date_ranges)-1)):
        from_date = date_ranges[i]
        to_date = date_ranges[i+1]

        articles = fetch_guardian_articles(from_date, to_date)
        if articles:
            all_articles.extend(articles)
            print(f"Collected {len(articles)} articles for {from_date} to {to_dat
        else:
            print(f"No articles found for {from_date} to {to_date}")

    if not all_articles:
        print("No articles were collected. Check your API key and query.")
        return pd.DataFrame()

    # Process results
    processed_articles = []
    for article in all_articles:
        try:
            processed_articles.append({
                'title': article.get('fields', {}).get('headline', ''),
                'url': article.get('webUrl', ''),
                'content': article.get('fields', {}).get('body', ''),
                'published_date': article.get('webPublicationDate', '')[:10],
                'source': 'The Guardian',
                'word_count': int(article.get('fields', {}).get('wordcount', 0)),
                'section': article.get('sectionName', '')
            })
        except Exception as e:
            print(f"Skipping article due to processing error: {str(e)}")

    df = pd.DataFrame(processed_articles)
    if df.empty:
        print("No valid articles were processed.")
        return pd.DataFrame()

    # Remove duplicates and sort
    df = df.drop_duplicates(subset=['url'])
    df = df.sort_values('published_date', ascending=False)
    df.to_csv(OUTPUT_FILE, index=False)

    # Calculate statistics
    days_diff = (datetime.strptime(END_DATE, '%Y-%m-%d') - datetime.strptime(STAR
    avg_per_day = len(df)/days_diff if days_diff > 0 else 0

    print("\nScraping complete!")
    print(f"Collected {len(df)} unique articles")
    print(f"Date coverage: {df['published_date'].min()} to {df['published_date'].
    print(f"Average articles per day: {avg_per_day:.1f}")
    print("\nSample data:")
    return df.head(10)

if __name__ == "__main__":
    result = main()
    if not result.empty:
        print(result[['title', 'published_date', 'word_count']])
```

```
 Found 61 results for 2024-02-29 to 2024-03-31
  82%|████████▏ | 62/76 [01:57<00:31,  2.26s/it]Collected 61 articles for 2024-
 Found 24 results for 2024-03-31 to 2024-04-30
  83%|████████▎ | 63/76 [01:59<00:26,  2.05s/it]Collected 24 articles for 2024-
 Found 16 results for 2024-04-30 to 2024-05-31
  84%|████████▍ | 64/76 [02:01<00:22,  1.91s/it]Collected 16 articles for 2024-
 Found 22 results for 2024-05-31 to 2024-06-30
  86%|████████▌ | 65/76 [02:02<00:20,  1.82s/it]Collected 22 articles for 2024-
 Found 27 results for 2024-06-30 to 2024-07-31
  87%|████████▋ | 66/76 [02:04<00:17,  1.76s/it]Collected 27 articles for 2024-
 Found 29 results for 2024-07-31 to 2024-08-31
  88%|████████▊ | 67/76 [02:06<00:15,  1.72s/it]Collected 29 articles for 2024-
 Found 27 results for 2024-08-31 to 2024-09-30
  89%|████████▉ | 68/76 [02:07<00:13,  1.75s/it]Collected 27 articles for 2024-
 Found 35 results for 2024-09-30 to 2024-10-31
  91%|█████████ | 69/76 [02:09<00:12,  1.77s/it]Collected 35 articles for 2024-
 Found 78 results for 2024-10-31 to 2024-11-30
  92%|█████████▏| 70/76 [02:13<00:14,  2.34s/it]Collected 78 articles for 2024-
 Found 43 results for 2024-11-30 to 2024-12-31
  93%|█████████▎| 71/76 [02:15<00:10,  2.18s/it]Collected 43 articles for 2024-
 Found 57 results for 2024-12-31 to 2025-01-31
  95%|█████████▍| 72/76 [02:18<00:10,  2.57s/it]Collected 57 articles for 2024-
 Found 61 results for 2025-01-31 to 2025-02-28
  96%|█████████▋| 73/76 [02:21<00:08,  2.79s/it]Collected 61 articles for 2025-
 Found 46 results for 2025-02-28 to 2025-03-31
  97%|█████████▋| 74/76 [02:23<00:05,  2.56s/it]Collected 46 articles for 2025-
 Found 30 results for 2025-03-31 to 2025-04-30
  99%|█████████▊| 75/76 [02:25<00:02,  2.33s/it]Collected 30 articles for 2025-
 Found 35 results for 2025-04-30 to 2025-05-16
 100%|██████████| 76/76 [02:27<00:00,  1.94s/it]Collected 35 articles for 2025-


 Scraping complete!
 Collected 2326 unique articles
 Date coverage: 2019-01-31 to 2025-05-16
 Average articles per day: 1.0

 Sample data:
                                                   title published_date  \
 2371  Trump’s health department to stop recommending...     2025-05-16
 2374  Seth Meyers on Trump corruption: ‘It’s all so ...     2025-05-15
 2373  UK asking other countries to host ‘return hubs...     2025-05-15
 2375  RFK Jr defends downsizing health department as...     2025-05-15
 2372  Texas swelters as record-breaking heatwave swe...     2025-05-15
 2376  Trump’s cryptocurrency endeavor caps a politic...     2025-05-14
 2377  New prisons to be built and inmates released e...     2025-05-14
 2378  Trump cabinet member’s links to El Salvador cr...     2025-05-14
 2379  What Trump’s ‘palace in the sky’ gift from Qat...     2025-05-14
 2380  Labour peer apologises for writing to Treasury...     2025-05-14

       word_count
 2371       10883
 2374         682
 2373        7163
 2375       11124
 2372         392
 2376        1365
 2377        8004
 2378        2845
 2379         986
```

```
import pandas as pd

# Load the collected data
df = pd.read_csv("guardian_bitcoin_simple.csv")

# Convert published_date to datetime
df['published_date'] = pd.to_datetime(df['published_date'])

# 1. Count articles per day
daily_counts = df['published_date'].value_counts().sort_index()
```

```
# 2. Find days with no articles
date_range = pd.date_range(start=df['published_date'].min(),
                          end=df['published_date'].max())
missing_days = [d.date() for d in date_range if d not in daily_counts.index]

# 3. Get statistics
stats = {
    "total_articles": len(df),
    "total_days": len(date_range),
    "days_with_articles": len(daily_counts),
    "days_with_no_articles": len(missing_days),
    "max_articles_day": daily_counts.max(),
    "min_articles_day": daily_counts.min(),
    "avg_articles_day": daily_counts.mean(),
    "median_articles_day": daily_counts.median()
}

# Print results
print("Article Distribution Analysis:")
for k, v in stats.items():
    print(f"{k.replace('_', ' ').title()}: {v}")

print("\nDays with most articles:")
print(daily_counts.nlargest(5))

if missing_days:
    print(f"\nFirst 10 days with no articles:")
    print(missing_days[:10])
else:
    print("\nNo days without articles!")
```

```
Article Distribution Analysis:
Total Articles: 2326
Total Days: 2298
Days With Articles: 1219
Days With No Articles: 1079
Max Articles Day: 11
Min Articles Day: 1
Avg Articles Day: 1.908121410992617
Median Articles Day: 1.0

Days with most articles:
published_date
2024-11-11    11
2021-05-19    10
2022-12-13    10
2024-11-12     9
2022-11-15     8
Name: count, dtype: int64

First 10 days with no articles:
[datetime.date(2019, 2, 1), datetime.date(2019, 2, 2), datetime.date(2019, 2,
```

```
import requests
import pandas as pd
from datetime import datetime
from tqdm import tqdm
import time

# Configuration
API_KEY = "d40a5c30-8b6d-4f17-a4ba-3deab348a109"

def fetch_guardian_articles(from_date, to_date, query="Bitcoin"):
    """Fetch articles from Guardian API"""
    all_articles = []
    page = 1
    total_pages = 1

    while page <= total_pages:
        url = "https://content.guardianapis.com/search"
        params = {
            "q": query,
            "from-date": from_date,
            "to-date": to_date,
            "api-key": API_KEY,
            "page": page,
            "page-size": 50,
            "show-fields": "headline,trailText,body,byline,wordcount",
            "order-by": "newest"
        }

        try:
            response = requests.get(url, params=params, timeout=15)
            data = response.json()

            if 'response' not in data:
                break

            articles = data['response'].get('results', [])
            all_articles.extend(articles)

            if page == 1:
                total_pages = min(5, data['response'].get('pages', 1))  # Limit t

            page += 1
            time.sleep(0.5)

        except Exception as e:
            print(f"Error on page {page}: {str(e)}")
            break

    return all_articles

def identify_missing_days(df):
    """Find all dates without articles"""
    date_range = pd.date_range(
        start=df['published_date'].min(),
        end=df['published_date'].max()
    )
    return [d.date() for d in date_range
            if d not in df['published_date'].dt.date.unique()]

def fill_missing_days(df):
    """Target scraping for days with no articles"""
    missing_days = identify_missing_days(df)
    new_articles = []

    for day in tqdm(missing_days[:100]):  # Process first 100 missing days
        day_str = day.strftime('%Y-%m-%d')

        # Try multiple query variations
        for query in ["Bitcoin", "BTC", "cryptocurrency"]:
            articles = fetch_guardian_articles(
                from_date=day_str,
                to_date=day_str,
                query=query
            )
            if articles:
                new_articles.extend(articles)
                break  # Move to next day if found articles
        time.sleep(1)  # Be gentle with the API

    return new_articles

# Load your existing data
df = pd.read_csv("guardian_bitcoin_simple.csv")
df['published_date'] = pd.to_datetime(df['published_date'])

# Fill missing days
new_articles = fill_missing_days(df)

# Process new articles
if new_articles:
    new_df = pd.DataFrame([{
        'title': article.get('fields', {}).get('headline', ''),
        'url': article.get('webUrl', ''),
        'content': article.get('fields', {}).get('body', ''),
        'published_date': article.get('webPublicationDate', '')[:10],
        'source': 'The Guardian',
        'word_count': int(article.get('fields', {}).get('wordcount', 0)),
        'section': article.get('sectionName', '')
    } for article in new_articles])

    # Combine with original data
    enhanced_df = pd.concat([df, new_df])
    enhanced_df = enhanced_df.drop_duplicates(subset=['url'])
    enhanced_df.to_csv("guardian_bitcoin_enhanced.csv", index=False)

    # Show updated stats
    print(f"\nAdded {len(new_df)} new articles")
    print(f"Total articles now: {len(enhanced_df)}")
else:
    print("No new articles found for missing days")
```

```
100%|██████████| 100/100 [06:37<00:00,  3.97s/it]

Added 21 new articles
Total articles now: 2326
```

```python
import pandas as pd
from datetime import datetime

def max_consecutive_missing(missing_days):
    """Calculate longest streak of consecutive missing days"""
    if not missing_days:
        return 0

    missing_days_sorted = sorted(missing_days)
    max_gap = 1
    current_gap = 1

    for i in range(1, len(missing_days_sorted)):
        if (missing_days_sorted[i] - missing_days_sorted[i-1]).days == 1:
            current_gap += 1
            max_gap = max(max_gap, current_gap)
        else:
            current_gap = 1

    return max_gap

# Load the enhanced data
enhanced_df = pd.read_csv("guardian_bitcoin_enhanced.csv")
enhanced_df['published_date'] = pd.to_datetime(enhanced_df['published_date'])

# 1. Count articles per day (sorted chronologically)
daily_counts = enhanced_df['published_date'].value_counts().sort_index()

# 2. Find date range and missing days
date_range = pd.date_range(start=enhanced_df['published_date'].min(),
                          end=enhanced_df['published_date'].max())
missing_days = [d.date() for d in date_range if d not in daily_counts.index]

# 3. Calculate statistics
stats = {
    "total_articles": len(enhanced_df),
    "total_days": len(date_range),
    "days_with_articles": len(daily_counts),
    "days_with_no_articles": len(missing_days),
    "max_articles_day": daily_counts.max(),
    "min_articles_day": daily_counts.min(),
    "avg_articles_day": daily_counts.mean(),
    "median_articles_day": daily_counts.median()
}

# Print enhanced statistics
print("Enhanced Coverage Statistics:")
print("="*50)
for k, v in stats.items():
    print(f"{k.replace('_', ' ').title():<25} {v}")

# Show date distribution
print("\nArticle Distribution by Date:")
print("="*50)
print(daily_counts.head(10))  # First 10 dates
print("...")
print(daily_counts.tail(10))  # Last 10 dates

# Missing days analysis
print("\nMissing Days Analysis:")
print("="*50)
if missing_days:
    print(f"First 10 missing days: {missing_days[:10]}")
    print(f"Last 10 missing days: {missing_days[-10:]}")
    print(f"\nLongest gap without articles: {max_consecutive_missing(missing_days
else:
    print("No missing days - complete coverage!")
```

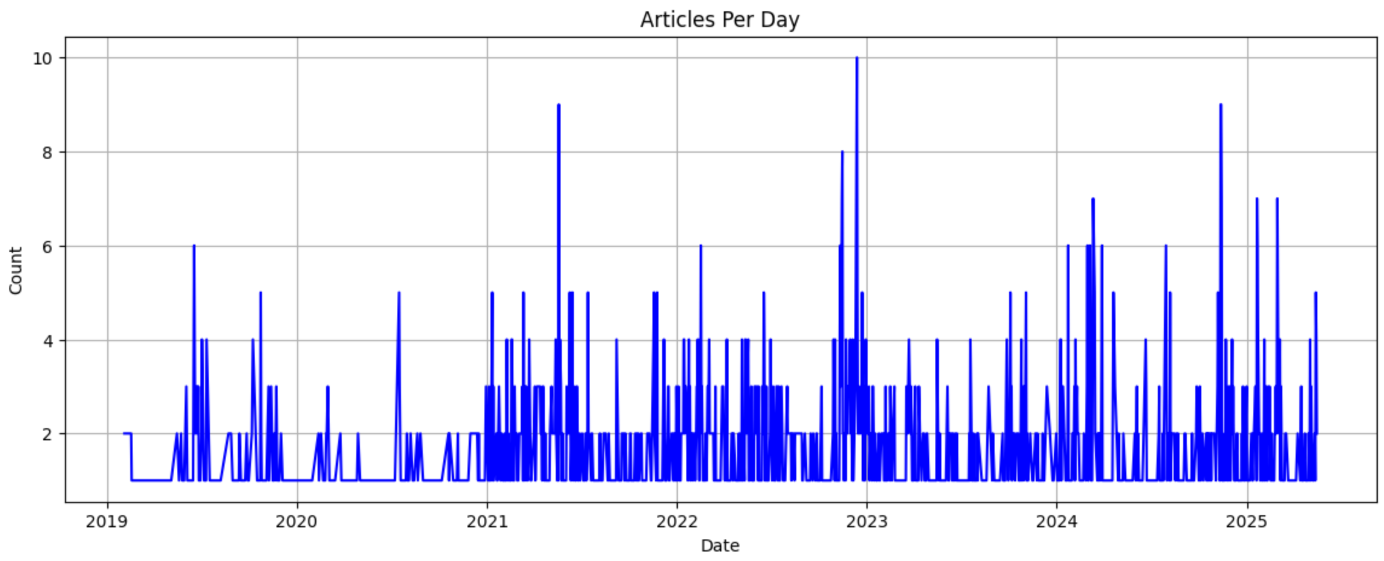
```
Enhanced Coverage Statistics:
==================================================
Total Articles           2326
Total Days               2298
Days With Articles       1219
Days With No Articles    1079
Max Articles Day         11
Min Articles Day         1
Avg Articles Day         1.908121410992617
Median Articles Day      1.0

Article Distribution by Date:
==================================================
published_date
2019-01-31    1
2019-02-04    2
2019-02-17    2
2019-02-18    1
2019-02-22    1
2019-02-27    1
2019-03-15    1
2019-03-18    1
2019-03-21    1
2019-03-22    1
Name: count, dtype: int64
...
published_date
2025-05-05    2
2025-05-06    1
2025-05-07    1
2025-05-09    2
2025-05-11    2
2025-05-12    1
2025-05-13    5
2025-05-14    5
2025-05-15    4
2025-05-16    1
Name: count, dtype: int64

Missing Days Analysis:
==================================================
First 10 missing days: [datetime.date(2019, 2, 1), datetime.date(2019, 2, 2),
Last 10 missing days: [datetime.date(2025, 4, 6), datetime.date(2025, 4, 9),

Longest gap without articles: 24 days
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("guardian_bitcoin_enhanced.csv")

# Convert to datetime and sort
df['published_date'] = pd.to_datetime(df['published_date'])
df = df.sort_values('published_date')

# Handle duplicates (keep first occurrence)
df = df.drop_duplicates(subset=['url', 'title'], keep='first')

# Fill missing values in text fields
df['content'] = df['content'].fillna('')
df['title'] = df['title'].fillna('')


last_year = enhanced_df[enhanced_df['published_date'] >= '2024-01-01']
print(f"2024 Coverage: {len(last_year)} articles ({len(last_year)/365:.1f} per da
```

```
2024 Coverage: 645 articles (1.8 per day)
```

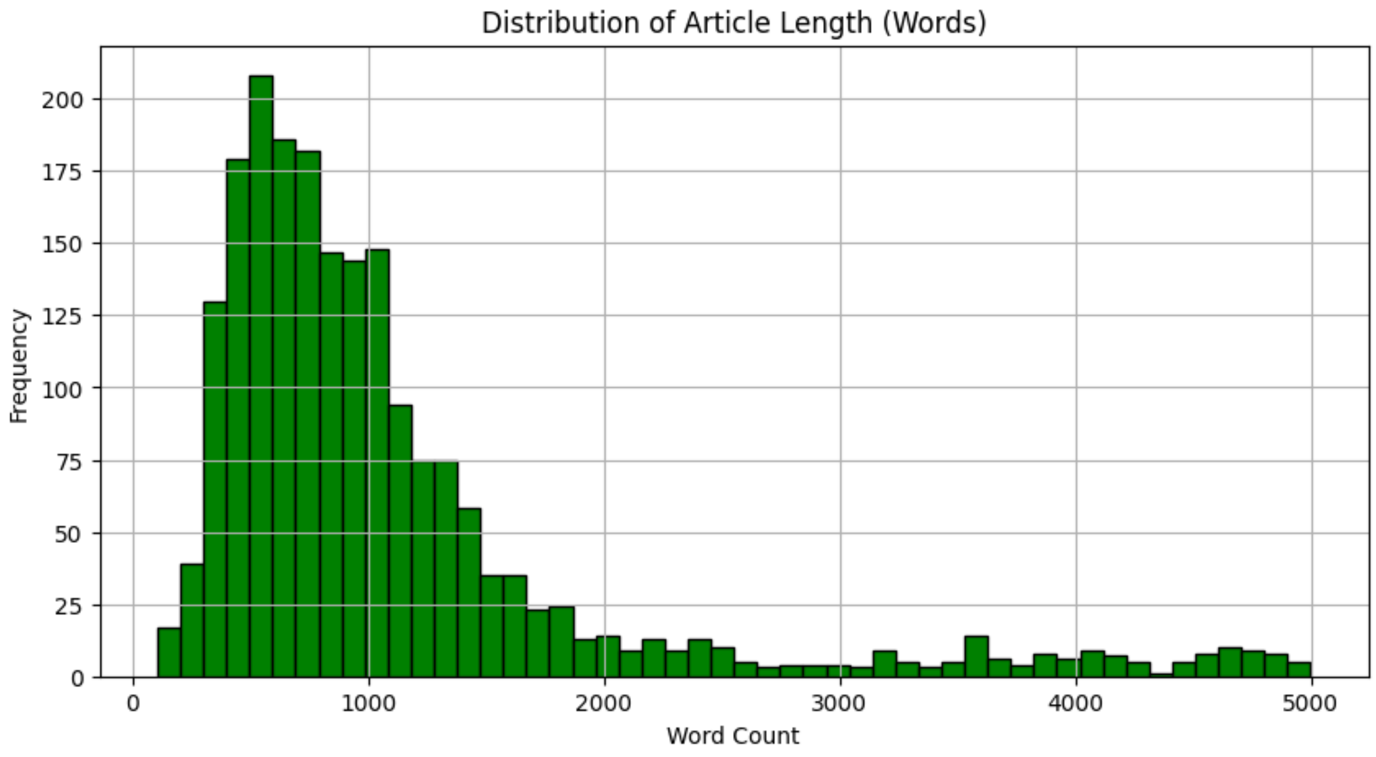
```
import pandas as pd

# Load raw Guardian API data
df = pd.read_csv("guardian_bitcoin_enhanced.csv")

# Drop duplicates (title + URL)
df = df.drop_duplicates(subset=['title', 'url'])

# Fill missing text fields
df['title'] = df['title'].fillna('')
df['content'] = df['content'].fillna('[No Content]')

# Convert dates
df['published_date'] = pd.to_datetime(df['published_date'])
```

```
import re

def clean_text(text):
    # Remove special characters, URLs, and extra whitespace
    text = re.sub(r'http\S+', '', text)  # URLs
    text = re.sub(r'[^\w\s]', '', text)  # Punctuation
    text = text.lower().strip()          # Lowercase + trim
    return text

df['clean_title'] = df['title'].apply(clean_text)
df['clean_content'] = df['content'].apply(clean_text)


# Remove extremely short/long articles (adjust thresholds)
df = df[(df['word_count'] >= 50) & (df['word_count'] <= 5000)]

# Drop articles with placeholder titles
df = df[~df['title'].str.contains('\[No Title\]|placeholder', case=False)]


import matplotlib.pyplot as plt

# Articles per day
daily_counts = df['published_date'].dt.date.value_counts().sort_index()
plt.figure(figsize=(14, 5))
daily_counts.plot(title='Articles Per Day', color='blue')
plt.xlabel('Date')
plt.ylabel('Count')
plt.grid(True)
plt.show()

# Identify gaps
missing_dates = pd.date_range(
    start=df['published_date'].min(),
    end=df['published_date'].max()
).difference(df['published_date'])
print(f"Days without articles: {len(missing_dates)}")
```



```
Days without articles: 1144
```

```
# Word count analysis
plt.figure(figsize=(10, 5))
df['word_count'].hist(bins=50, color='green', edgecolor='black')
plt.title('Distribution of Article Length (Words)')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()

print(f"Median word count: {df['word_count'].median()}")
```



```
Median word count: 841.5
```

```
from collections import Counter
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

# Extract top keywords
words = ' '.join(df['clean_title']).split()
filtered_words = [w for w in words if w not in stopwords.words('english')]
keyword_counts = Counter(filtered_words).most_common(20)

print("Top 20 Keywords:")
for word, count in keyword_counts:
    print(f"{word}: {count}")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Top 20 Keywords:
crypto: 201
us: 162
bitcoin: 136
happened: 123
cryptocurrency: 120
trump: 112
uk: 109
new: 89
says: 81
sam: 71
ftx: 71
bankmanfried: 65
first: 63
briefing: 57
musk: 56
tech: 55
elon: 55
review: 51
mail: 49
trumps: 47
```

```
# Sample random titles to manually verify sentiment
sample_titles = df.sample(10)[['title', 'section']]
print(sample_titles.to_markdown(index=False))
```

```
| title
|:-----------------------------------------------------------------------------------------
| Yuga Labs apologises after sale of virtual land overwhelms Ethereum
| Morning mail: Questions for Frydenberg, NT's bright future, Himalayan big me
| Man has ‘finely tuned’ plan to find £500m bitcoin thrown in tip, Cardiff cou
| When hackers can take your nether regions hostage, something has gone very w
| What is LockBit ransomware and how does it operate?
| Money mules: how young people are lured into laundering cash
| TechScape: How a cryptocurrency project lost $180m to a get-rich-quick schem
| Bankrupt crypto exchange FTX ordered by US court to pay customers $12.7bn
| Facebook rejects Andrew Forrest’s legal claim it should be liable for crypto
| Morning mail: coronavirus fatalities rise, Biden fights back, farms on the c
```

```
# Manually label a small subset for accuracy testing
calibration_samples = [
    ("Bitcoin soars to record high", "positive"),
    ("Crypto crash wipes out gains", "negative"),
    ("Blockchain adoption grows steadily", "neutral")
]
```

```
import torch
print(torch.__version__)  # This should print the installed version of PyTorch
```

```
2.6.0+cu124
```

```
!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/python3.11/dist-pack
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyth
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1
Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 4.6 MB/s eta 0:00:0
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
import os
import os
import pandas as pd
import torch
from transformers import pipeline
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Check GPU availability
device = 0 if torch.cuda.is_available() else -1
print(f"Using {'GPU' if device == 0 else 'CPU'} for sentiment analysis")

# Initialize VADER sentiment analyzer
vader_analyzer = SentimentIntensityAnalyzer()

# Load BERT model explicitly
bert_analyzer = pipeline('sentiment-analysis',
                         model="nlptown/bert-base-multilingual-uncased-sentiment"
                         device=device)

# Load FinBERT model explicitly
finbert_analyzer = pipeline("sentiment-analysis",
                            model="yiyanghkust/finbert-tone",
                            tokenizer="yiyanghkust/finbert-tone",
                            device=device)

# Function to get sentiment (Batch Processing)
def get_sentiment_batch(texts, use_finbert=False):
    """
    Perform batch sentiment analysis using VADER, BERT, and optionally FinBERT.
    Uses VADER if high confidence, else falls back to BERT or FinBERT.
    """
    sentiments = []
    confidence_scores = []

    # Step 1: Compute VADER scores for all texts
    vader_scores = [vader_analyzer.polarity_scores(text)['compound'] for text in

    # Step 2: Identify texts needing BERT analysis (confidence < 0.7)
    texts_for_bert = [text for text, score in zip(texts, vader_scores) if abs(sco

    # Step 3: Use BERT or FinBERT for low-confidence cases
    if texts_for_bert:
        if use_finbert:
            bert_results = finbert_analyzer(texts_for_bert)
        else:
            bert_results = bert_analyzer(texts_for_bert)
        bert_sentiments = [result['label'] for result in bert_results]
        bert_confidences = [result['score'] for result in bert_results]

    # Step 4: Assign results based on VADER or BERT/FinBERT
    bert_index = 0
    for text, score in zip(texts, vader_scores):
        if abs(score) > 0.7:
            sentiment = 'positive' if score > 0 else 'negative'
            confidence = abs(score)
        else:
            sentiment = bert_sentiments[bert_index]
            confidence = bert_confidences[bert_index]
            bert_index += 1

        sentiments.append(sentiment)
        confidence_scores.append(confidence)

    return sentiments, confidence_scores

# Example sentences for testing
test_sentences = [
    "Bitcoin is reaching new heights, and investors are excited.",
    "The recent drop in Bitcoin's price has caused panic among traders.",
    "Experts predict that Bitcoin will stabilize and grow in the coming months.",
    "Many believe that Bitcoin is a bubble waiting to burst.",
    "The adoption of Bitcoin by major companies is a positive sign for the market
]

# Get sentiment for the test sentences using FinBERT
finbert_sentiments, finbert_confidences = get_sentiment_batch(test_sentences, use

# Get sentiment for the test sentences using BERT
bert_sentiments, bert_confidences = get_sentiment_batch(test_sentences, use_finbe

# Get sentiment for the test sentences using VADER
vader_scores = [vader_analyzer.polarity_scores(text)['compound'] for text in test

# Display results
print("\nSentiment Analysis Results:")
for i, sentence in enumerate(test_sentences):
    print(f"\nSentence: {sentence}")
    print(f"VADER Sentiment: {'positive' if vader_scores[i] > 0 else 'negative' i
    print(f"BERT Sentiment: {bert_sentiments[i]}, Confidence: {bert_confidences[i
    print(f"FinBERT Sentiment: {finbert_sentiments[i]}, Confidence: {finbert_conf
Using GPU for sentiment analysis
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: Use
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tak
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access pu
  warnings.warn(
config.json: 100%                                  953/953 [00:00<00:00, 83.0kB/s]
model.safetensors: 100%                            669M/669M [00:03<00:00, 202MB/s]
tokenizer_config.json: 100%                        39.0/39.0 [00:00<00:00, 3.48kB/s]
vocab.txt: 100%                                    872k/872k [00:00<00:00, 6.08MB/s]
special_tokens_map.json: 100%                      112/112 [00:00<00:00, 8.48kB/s]
Device set to use cuda:0
config.json: 100%                                  533/533 [00:00<00:00, 56.8kB/s]
pytorch_model.bin: 100%                            439M/439M [00:03<00:00, 243MB/s]
vocab.txt: 100%                                    226k/226k [00:00<00:00, 3.04MB/s]
model.safetensors: 100%                            439M/439M [00:02<00:00, 244MB/s]
Device set to use cuda:0

Sentiment Analysis Results:

Sentence: Bitcoin is reaching new heights, and investors are excited.
VADER Sentiment: positive, Score: 0.4939
BERT Sentiment: 5 stars, Confidence: 0.4236
FinBERT Sentiment: Positive, Confidence: 1.0000

Sentence: The recent drop in Bitcoin's price has caused panic among traders.
VADER Sentiment: negative, Score: -0.6597
BERT Sentiment: 2 stars, Confidence: 0.3889
FinBERT Sentiment: Negative, Confidence: 1.0000

Sentence: Experts predict that Bitcoin will stabilize and grow in the coming m
VADER Sentiment: neutral, Score: 0.0000
BERT Sentiment: 3 stars, Confidence: 0.3453
FinBERT Sentiment: Positive, Confidence: 0.9986

Sentence: Many believe that Bitcoin is a bubble waiting to burst.
VADER Sentiment: neutral, Score: 0.0000
BERT Sentiment: 1 star, Confidence: 0.3913
FinBERT Sentiment: Negative, Confidence: 0.8101

Sentence: The adoption of Bitcoin by major companies is a positive sign for th
```

```
    Sentence: The adoption of Bitcoin by major companies is a positive sign for th
    VADER Sentiment: positive, Score: 0.5574
    BERT Sentiment: 5 stars, Confidence: 0.4312
    FinBERT Sentiment: Positive, Confidence: 1.0000
```

Conclusion By leveraging VADER, BERT, and FinBERT, our project can achieve a comprehensive sentiment analysis framework that enhances our understanding of market trends and investor sentiment across Yahoo stocks, Wikipedia entries, and news articles. The varying strengths and accuracy levels of these models allow us to capture a wide range of sentiments effectively, ensuring that we derive meaningful insights from the data.

Sentiment analysis

```
import pandas as pd
import numpy as np
from transformers import pipeline, AutoTokenizer
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import torch

# Initialize VADER sentiment analyzer
vader_analyzer = SentimentIntensityAnalyzer()

# Initialize BERT sentiment analysis pipeline
device = 0 if torch.cuda.is_available() else -1  # Check if GPU is available, els
bert_analyzer = pipeline('sentiment-analysis', model="nlptown/bert-base-multiling
tokenizer = AutoTokenizer.from_pretrained("nlptown/bert-base-multilingual-uncased

# Load the collected data (Guardian articles)
guardian_df = pd.read_csv("guardian_bitcoin_enhanced.csv")

# Convert the 'published_date' to datetime
guardian_df['published_date'] = pd.to_datetime(guardian_df['published_date'])

# Function to split text into chunks of 512 tokens (Sliding Window Approach)
def sliding_window_tokenize(text, max_length=512, stride=256):
    """
    Splits long text into overlapping chunks of tokens. Ensures that we do not ex
    """
    # Tokenize the text using the BERT tokenizer
    tokens = tokenizer.encode(text, add_special_tokens=True, truncation=True, max_
    chunks = []
    # Create chunks using the sliding window approach
    for i in range(0, len(tokens), stride):
        chunk = tokens[i:i + max_length]
        if len(chunk) < max_length:
            chunks.append(chunk)
        else:
            chunks.append(chunk[:max_length])
    return chunks

# Function to get sentiment score using VADER and BERT (using sliding window)
def get_sentiment_score(text):
    # Get VADER sentiment score
    vader_score = vader_analyzer.polarity_scores(text)['compound']

    # Split the text into smaller chunks using the sliding window technique
    chunks = sliding_window_tokenize(text)

    # Get BERT sentiment score for each chunk and aggregate them
    bert_scores = []
    for chunk in chunks:
        chunk_text = tokenizer.decode(chunk, skip_special_tokens=True)
        bert_result = bert_analyzer(chunk_text)
        bert_score = bert_result[0]['score']
        sentiment = bert_result[0]['label']
        bert_scores.append(bert_score if sentiment == 'POSITIVE' else -bert_score

    # Combine VADER and BERT sentiment scores, using VADER score as fallback if c
    if abs(vader_score) >= 0.7:
        return vader_score  # Use VADER score if it's more confident
    else:
        return np.mean(bert_scores)  # Return average BERT score for the chunks

# Apply sentiment score calculation to each article
guardian_df['sentiment_score'] = guardian_df['content'].apply(get_sentiment_score

# 1. Count articles per day
daily_counts = guardian_df['published_date'].value_counts().sort_index()

# 2. Find days with no articles
date_range = pd.date_range(start=guardian_df['published_date'].min(), end=guardia
missing_days = [d.date() for d in date_range if d not in daily_counts.index]

# 3. Create a new DataFrame for sentiment analysis with default 0 (neutral) senti
sentiment_df = pd.DataFrame({'date': date_range, 'sentiment_score': 0})
```

```
# For each day, calculate the average sentiment score
for date in daily_counts.index:
    daily_articles = guardian_df[guardian_df['published_date'] == date]
    average_sentiment_score = daily_articles['sentiment_score'].mean()
    sentiment_df.loc[sentiment_df['date'] == date, 'sentiment_score'] = average_s

# Merge the sentiment DataFrame with the original Guardian data
final_df = pd.merge(guardian_df, sentiment_df, left_on='published_date', right_on

# Save the final DataFrame with sentiment data to a CSV
final_file_path = '/content/drive/MyDrive/Final_Project_Docs/guardian_bitcoin_wit
final_df.to_csv(final_file_path, index=False)

print("Data merged and saved successfully.")
```

```
Device set to use cuda:0
You seem to be using the pipelines sequentially on GPU. In order to maximize e
<ipython-input-50-70d48305ec66>:79: FutureWarning: Setting an item of incompat
  sentiment_df.loc[sentiment_df['date'] == date, 'sentiment_score'] = average_
Data merged and saved successfully.
```

```
import pandas as pd
import numpy as np
from datetime import datetime

# Load the collected data (Guardian articles with sentiment scores)
guardian_df = pd.read_csv("/content/drive/MyDrive/Final_Project_Docs/guardian_bit

# Load the main financial data
main_df = pd.read_csv("/content/drive/MyDrive/Final_Project_Docs/bitcoin_data_5_y

# Convert 'published_date' to datetime for Guardian data
guardian_df['published_date'] = pd.to_datetime(guardian_df['published_date'])

# 1. Calculate average sentiment for each day (if there are multiple articles)
guardian_df['date'] = guardian_df['published_date'].dt.date  # Extract date from 

# Calculate the daily average sentiment score for articles on that day
daily_sentiment = guardian_df.groupby('date')['sentiment_score_y'].mean().reset_i

# 2. Merge with main financial dataset based on date
main_df['Date'] = pd.to_datetime(main_df['Date']).dt.date  # Ensure 'Date' in fin

# Merge the sentiment data with the financial data, filling missing values with 0
final_df = pd.merge(main_df, daily_sentiment, left_on='Date', right_on='date', ho

# Fill missing sentiment values (days with no articles) with 0
final_df['sentiment_score_y'] = final_df['sentiment_score_y'].fillna(0)

# 3. Save the final DataFrame with sentiment to a new CSV file
final_file_path = '/content/drive/MyDrive/Final_Project_Docs/combined_financial_d
final_df.to_csv(final_file_path, index=False)

print("Data merged and saved successfully.")
```

```
Data merged and saved successfully.
```

```
# Drop the second 'date' column and keep the first one
final_df = final_df.drop(columns=['date'])

# Save the cleaned DataFrame to a new CSV file
final_file_path = '/content/drive/MyDrive/Final_Project_Docs/cleaned_financial_da
final_df.to_csv(final_file_path, index=False)

print("Cleaned data saved successfully to new CSV file.")
```

```
Cleaned data saved successfully to new CSV file.
```

```
import pandas as pd

# Load the final dataset
final_df = pd.read_csv("/content/drive/MyDrive/combined_financial_data_with_senti

# Print the first and last 20 rows to inspect the data
print("First 20 rows:")
print(final_df.head(20))

print("\nLast 20 rows:")
print(final_df.tail(20))
```

```
2653  2025-03-15   84343.109375  84672.671875  83639.593750  83968.406250
2654  2025-03-16   82579.687500  85051.601562  82017.906250  84333.320312
2655  2025-03-17   84075.687500  84725.328125  82492.156250  82576.335938
2656  2025-03-18   82718.500000  84075.718750  81179.992188  84075.718750
2657  2025-03-19   86854.226562  87021.187500  82569.726562  82718.804688
2658  2025-03-20   84167.195312  87443.265625  83647.195312  86872.953125
2659  2025-03-21   84043.242188  84782.273438  83171.070312  84164.539062
2660  2025-03-22   83832.484375  84513.875000  83674.781250  84046.257812
2661  2025-03-23   86054.375000  86094.781250  83794.914062  83831.898438
2662  2025-03-24   87498.914062  88758.726562  85541.195312  86070.929688
2663  2025-03-25   87471.703125  88542.398438  86346.078125  87512.820312
2664  2025-03-26   86900.882812  88292.156250  85861.453125  87460.234375
2665  2025-03-27   87177.101562  87786.726562  85837.937500  86896.257812
2666  2025-03-28   84353.148438  87489.859375  83557.640625  87185.234375
2667  2025-03-29   82597.585938  84567.335938  81634.140625  84352.070312
2668  2025-03-30   82334.523438  83505.000000  81573.250000  82596.984375

           Volume  wikipedia_sentiment  tomorrow_price        date  \
2649  40353484454              -0.1000    81066.703125  2025-03-12
```

```
2650  31412940153              0.0772   81066.703125  2025-03-13
2651  31412940153             -0.2960   83969.101562  2025-03-13
2652  29588112414             -0.1000   84343.109375  2025-03-14
2653  13650491277             -0.1000   82579.687500  2025-03-15
2654  21330270174             -0.1000   84075.687500         NaN
2655  25092785558             -0.1000   82718.500000  2025-03-17
2656  24095774594             -0.1000   86854.226562         NaN
2657  34931960257             -0.1000   84167.195312         NaN
2658  29028988961             -0.1000   84043.242188         NaN
2659  19030452299             -0.1000   83832.484375  2025-03-21
2660   9863214091             -0.1000   86054.375000         NaN
2661  12594615537             -0.1000   87498.914062         NaN
2662  34582604933             -0.1000   87471.703125         NaN
2663  30005840049             -0.1000   86900.882812  2025-03-25
2664  26704046038             -0.1000   87177.101562         NaN
2665  24413471941             -0.1000   84353.148438         NaN
2666  34198619509             -0.1000   82597.585938  2025-03-28
2667  16969396135             -0.1000   82334.523438  2025-03-29
2668  14763760943             -0.1000            NaN         NaN

      sentiment_score_y
2649          -0.984100
2650           0.977300
2651           0.977300
2652           0.989100
2653           0.353178
2654           0.000000
2655          -0.967000
2656           0.000000
2657           0.000000
2658           0.000000
2659          -0.045100
2660           0.000000
2661           0.000000
2662           0.000000
2663           0.999500
2664           0.000000
2665           0.000000
2666           0.994700
2667          -0.989400
```

```
# Basic information about the dataset
print("\nBasic information about the dataset:")
print(final_df.info())
```

```

Basic information about the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2669 entries, 0 to 2668
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Date                 2669 non-null   object
 1   closing_price        2669 non-null   float64
 2   High                 2669 non-null   float64
 3   Low                  2669 non-null   float64
 4   Open                 2669 non-null   float64
 5   Volume               2669 non-null   int64
 6   wikipedia_sentiment  2669 non-null   float64
 7   tomorrow_price       2668 non-null   float64
 8   date                 1432 non-null   object
 9   sentiment_score_y    2669 non-null   float64
dtypes: float64(7), int64(1), object(2)
memory usage: 208.6+ KB
None
```

```
# Summary statistics
print("\nSummary statistics of the dataset:")
print(final_df.describe())
```

```
Summary statistics of the dataset:
       closing_price           High            Low           Open  \
count    2669.000000    2669.000000    2669.000000    2669.000000   
mean    35758.647635   36466.846977   34918.221495   35662.748390   
std     24280.051708   24774.552619   23715.390728   24264.180676   
min      3399.471680    3427.945557    3391.023682    3401.376465   
25%     13546.522461   13796.489258   13060.837891   13437.874023   
50%     35350.187500   36129.925781   33902.074219   35284.343750   
75%     49631.242188   50724.867188   48199.941406   49612.105469   
max    106146.265625  109114.882812  105291.734375  106147.296875   

             Volume  wikipedia_sentiment  tomorrow_price  sentiment_score_y  
count  2.669000e+03          2669.000000     2668.000000        2669.000000  
mean   3.033094e+10            -0.091474    35770.609827           0.053383  
std    1.926220e+10             0.079741    24276.735622           0.609010  
min    4.324201e+09            -0.743000     3399.471680          -1.000000  
25%    1.820612e+10            -0.100000    13549.497559           0.000000  
50%    2.541390e+10            -0.100000    35393.720703           0.000000  
75%    3.685717e+10            -0.100000    49649.764648           0.319867  
max    3.509679e+11             0.735100   106146.265625           1.000000  
```

```
# Check for missing values
print("\nMissing values in the dataset:")
print(final_df.isnull().sum())

# Check for duplicates
print("\nNumber of duplicate rows:")
print(final_df.duplicated().sum())
```

```
Missing values in the dataset:
Date                      0
closing_price             0
High                      0
Low                       0
Open                      0
Volume                    0
wikipedia_sentiment       0
tomorrow_price            1
date                   1237
sentiment_score_y         0
dtype: int64

Number of duplicate rows:
276
```

```
# Drop the second 'date' column and keep the first one
final_df = final_df.drop(columns=['date'])

# Save the cleaned DataFrame to a new CSV file
final_file_path = '/content/drive/MyDrive/Final_Project_Docs/cleaned_financial_da
final_df.to_csv(final_file_path, index=False)

print("Cleaned data saved successfully to new CSV file.")
```

```
Cleaned data saved successfully to new CSV file.
```

```
# Check the distribution of the sentiment score
print("\nDistribution of sentiment scores:")
print(final_df['sentiment_score_y'].describe())
```

```
Distribution of sentiment scores:
count    2669.000000
mean        0.053383
std         0.609010
min        -1.000000
25%         0.000000
50%         0.000000
75%         0.319867
max         1.000000
Name: sentiment_score_y, dtype: float64
```

```
import pandas as pd
import numpy as np
from datetime import datetime

# Load the collected data (Guardian articles with sentiment scores)
final_df = pd.read_csv("/content/drive/MyDrive/Final_Project_Docs/cleaned_financi

# Convert 'Date' to datetime for Guardian and financial data
final_df['Date'] = pd.to_datetime(final_df['Date'])

# 1. Check for duplicates and remove them
print("Checking for duplicates:")
duplicates = final_df.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

# Remove duplicates
final_df = final_df.drop_duplicates()

# 2. Fill missing sentiment values with the previous day's non-zero sentiment
# We want to fill `sentiment_score_y` with the previous non-zero sentiment if the
# We do it using forward fill but only if the previous value is not 0.

final_df['sentiment_score_y'] = final_df['sentiment_score_y'].replace(0, np.nan)
final_df['sentiment_score_y'] = final_df['sentiment_score_y'].fillna(method='ffil

# 3. Check if the sentiment is still 0 (if no previous value was available)
final_df['sentiment_score_y'] = final_df['sentiment_score_y'].fillna(0)  # Replac
```

```
# 4. Check for missing values
print("\nMissing values in the dataset:")
print(final_df.isnull().sum())

# 5. Summary statistics
print("\nSummary statistics of the dataset:")
print(final_df.describe())

# 6. Check for any remaining duplicates after cleaning
print("\nChecking for duplicates after cleaning:")
duplicates_after_cleaning = final_df.duplicated().sum()
print(f"Number of duplicate rows after cleaning: {duplicates_after_cleaning}")
```

```
Checking for duplicates:
Number of duplicate rows: 276

Missing values in the dataset:
Date                   0
closing_price          0
High                   0
Low                    0
Open                   0
Volume                 0
wikipedia_sentiment    0
tomorrow_price         1
sentiment_score_y      0
dtype: int64

Summary statistics of the dataset:
                                Date  closing_price           High  \
count                           2393    2393.000000    2393.000000
mean   2022-03-22 12:50:50.898453760   34956.818903   35674.521555
min              2019-01-01 00:00:00    3399.471680    3427.945557
25%              2020-08-21 00:00:00   11322.123047   11528.189453
50%              2022-04-11 00:00:00   29561.494141   30117.744141
75%              2023-11-22 00:00:00   50822.195312   51950.027344
max              2025-03-30 00:00:00  106146.265625  109114.882812
std                              NaN   25081.260238   25599.943608

                Low           Open        Volume  wikipedia_sentiment  \
count   2393.000000    2393.000000  2.393000e+03          2393.000000
mean   34128.047337   34912.254520  3.108645e+10            -0.092733
min     3391.023682    3401.376465  4.324201e+09            -0.743000
25%    11007.202148   11296.082031  1.818890e+10            -0.100000
50%    29113.814453   29538.859375  2.703645e+10            -0.100000
75%    49506.054688   51143.226562  3.831860e+10            -0.100000
```

```
max    105291.734375  106147.296875  3.509679e+11             0.735100
std     24491.839729   25066.678182  1.970325e+10             0.079414

       tomorrow_price  sentiment_score_y
count     2392.000000        2393.000000
mean     34969.826135           0.206369
min       3399.471680          -1.000000
25%      11323.078857          -0.768600
50%      29561.927734           0.341777
75%      50890.059570           0.986700
max     106146.265625           1.000000
std      25078.430527           0.808228

Checking for duplicates after cleaning:
Number of duplicate rows after cleaning: 0
<ipython-input-59-c77b7be14bd6>:24: FutureWarning: Series.fillna with 'method
  final_df['sentiment_score_y'] = final_df['sentiment_score_y'].fillna(method=
```

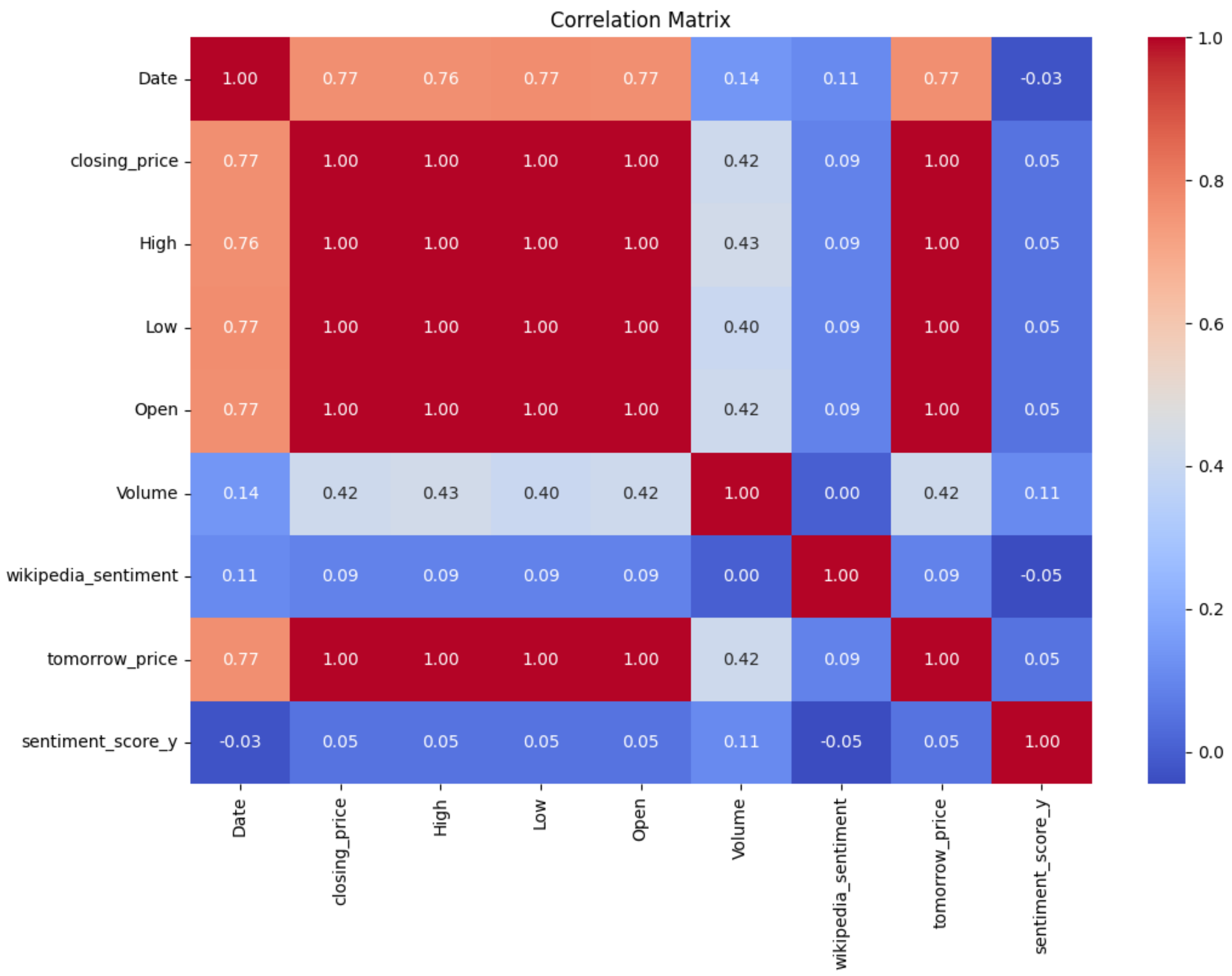
```
# Save the cleaned data to a new CSV file
cleaned_file_path = '/content/drive/MyDrive/Final_Project_Docs/cleaned_financial_
final_df.to_csv(cleaned_file_path, index=False)

print("Cleaned data saved successfully.")
```

```
Cleaned data saved successfully.
```

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

# Summary statistics
print("\nSummary statistics of the dataset:")
print(final_df.describe())
```

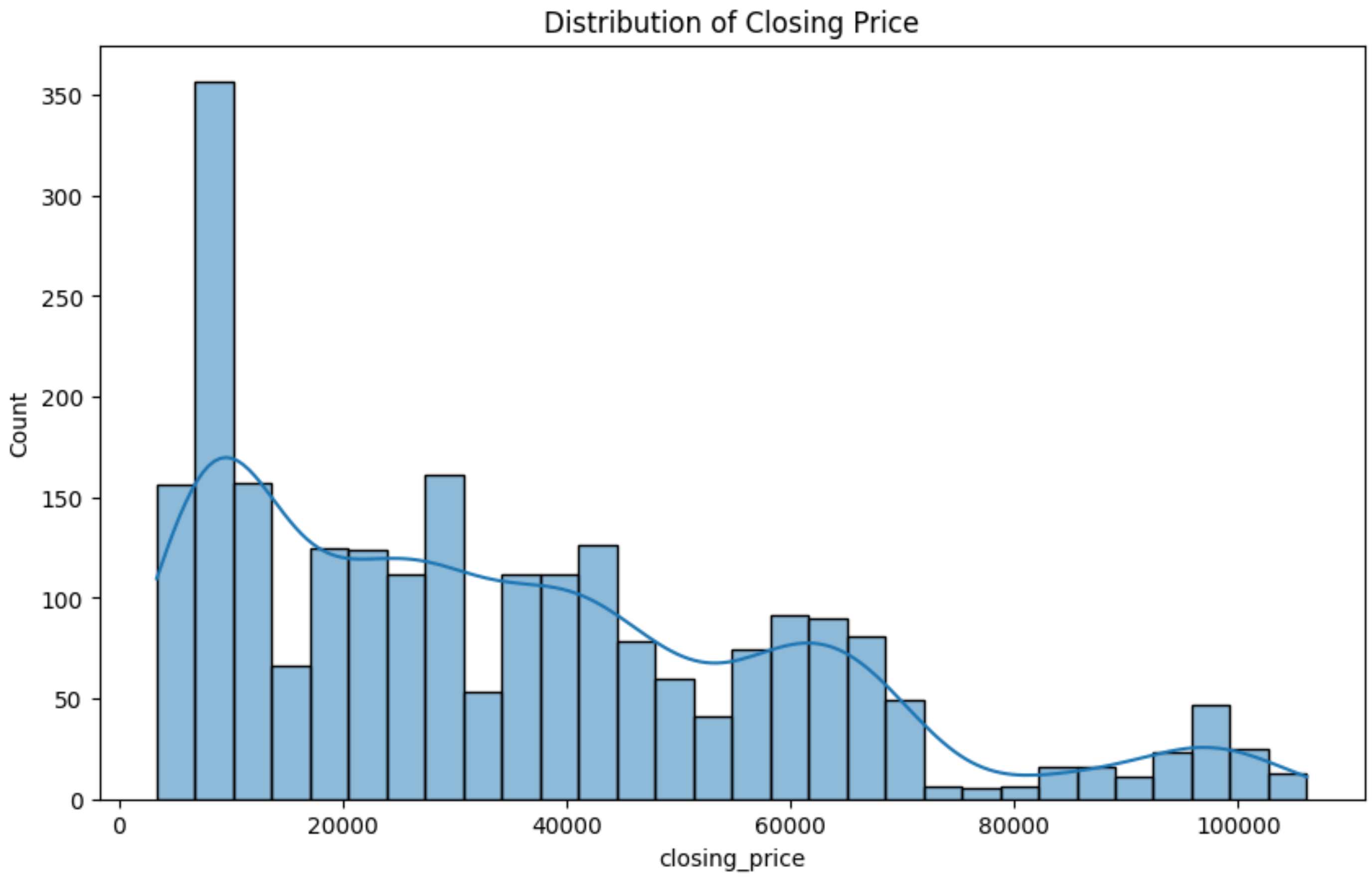
```
Summary statistics of the dataset:
                                Date  closing_price           High  \
count                           2393    2393.000000    2393.000000
mean   2022-03-22 12:50:50.898453760   34956.818903   35674.521555
min              2019-01-01 00:00:00    3399.471680    3427.945557
25%              2020-08-21 00:00:00   11322.123047   11528.189453
50%              2022-04-11 00:00:00   29561.494141   30117.744141
75%              2023-11-22 00:00:00   50822.195312   51950.027344
max              2025-03-30 00:00:00  106146.265625  109114.882812
std                              NaN   25081.260238   25599.943608

                 Low           Open        Volume  wikipedia_sentiment  \
count    2393.000000    2393.000000  2.393000e+03          2393.000000
mean    34128.047337   34912.254520  3.108645e+10            -0.092733
min      3391.023682    3401.376465  4.324201e+09            -0.743000
25%     11007.202148   11296.082031  1.818890e+10            -0.100000
50%     29113.814453   29538.859375  2.703645e+10            -0.100000
75%     49506.054688   51143.226562  3.831860e+10            -0.100000
max    105291.734375  106147.296875  3.509679e+11             0.735100
std     24491.839729   25066.678182  1.970325e+10             0.079414

       tomorrow_price  sentiment_score_y
count     2392.000000        2393.000000
mean     34969.826135           0.206369
min       3399.471680          -1.000000
25%      11323.078857          -0.768600
50%      29561.927734           0.341777
75%      50890.059570           0.986700
max     106146.265625           1.000000
std      25078.430527           0.808228
```

```
#  Correlation matrix for numerical features
corr_matrix = final_df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

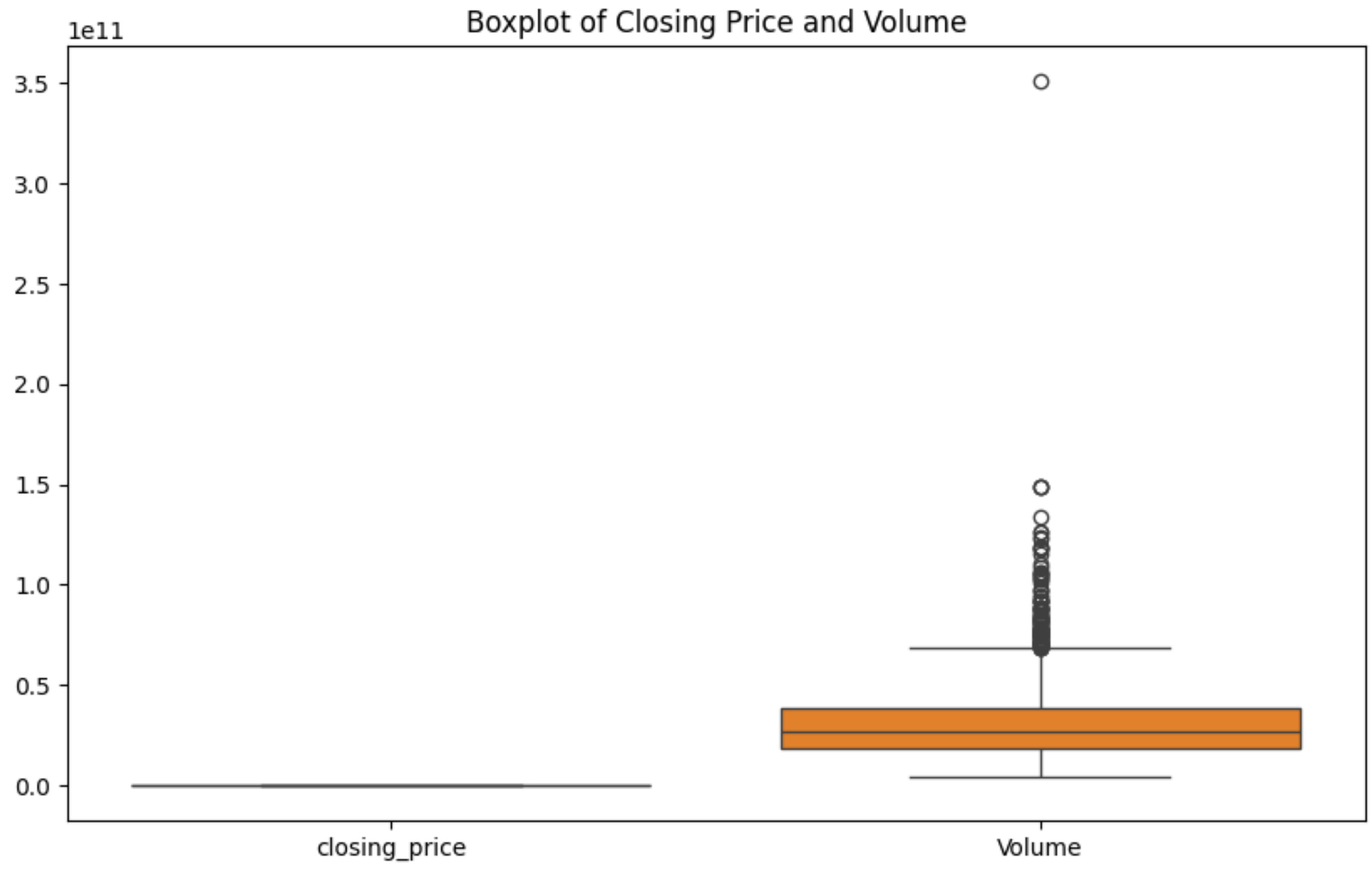


	Date	closing_price	High	Low	Open	Volume	wikipedia_sentiment	tomorrow_price	sentiment_score_y
Date	1.00	0.77	0.76	0.77	0.77	0.14	0.11	0.77	-0.03
closing_price	0.77	1.00	1.00	1.00	1.00	0.42	0.09	1.00	0.05
High	0.76	1.00	1.00	1.00	1.00	0.43	0.09	1.00	0.05
Low	0.77	1.00	1.00	1.00	1.00	0.40	0.09	1.00	0.05
Open	0.77	1.00	1.00	1.00	1.00	0.42	0.09	1.00	0.05
Volume	0.14	0.42	0.43	0.40	0.42	1.00	0.00	0.42	0.11
wikipedia_sentiment	0.11	0.09	0.09	0.09	0.09	0.00	1.00	0.09	-0.05
tomorrow_price	0.77	1.00	1.00	1.00	1.00	0.42	0.09	1.00	0.05
sentiment_score_y	-0.03	0.05	0.05	0.05	0.05	0.11	-0.05	0.05	1.00

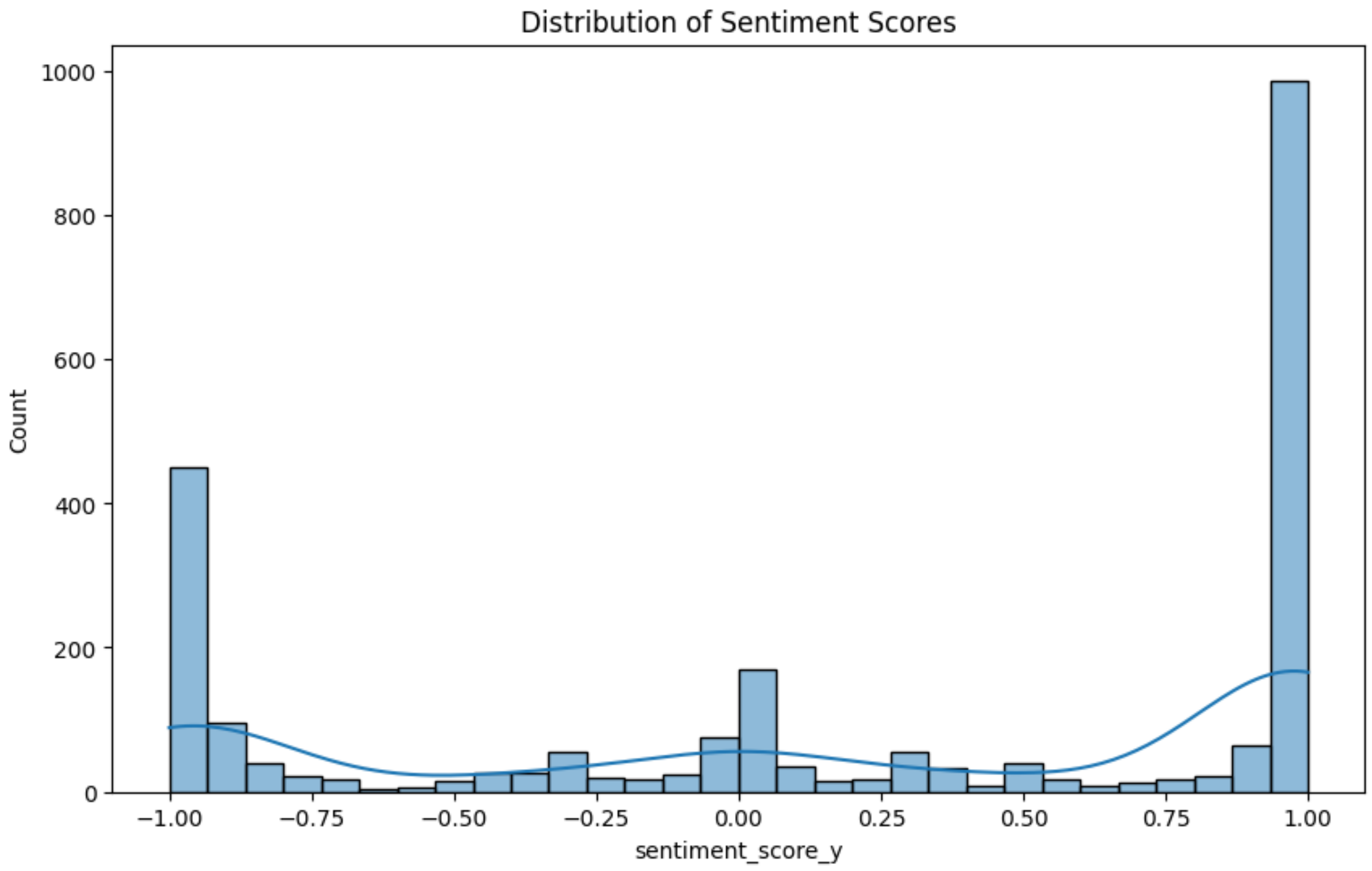
```
#  Distribution of 'closing_price'
plt.figure(figsize=(10, 6))
sns.histplot(final_df['closing_price'], kde=True, bins=30)
plt.title('Distribution of Closing Price')
plt.show()
```



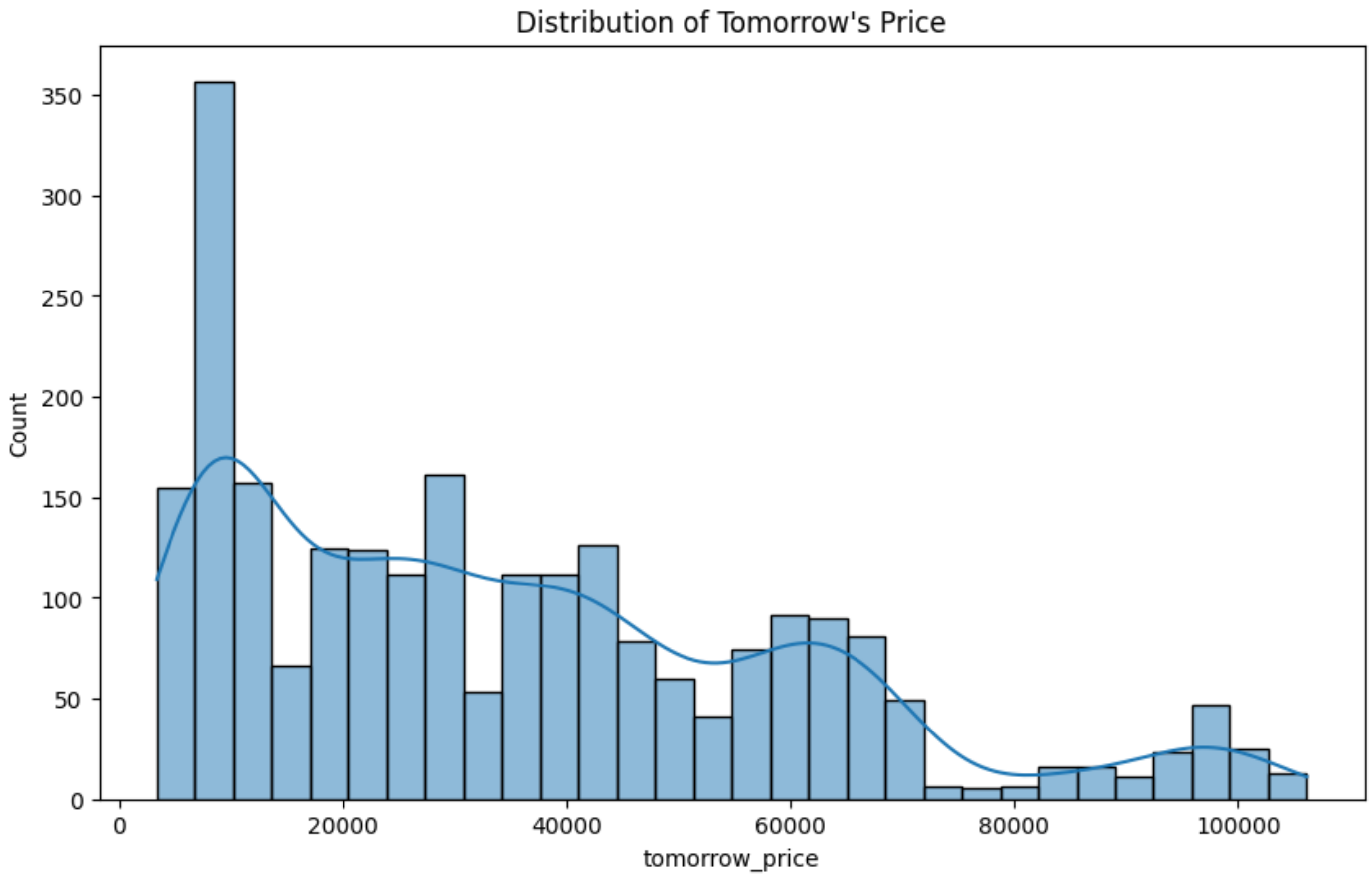
```
#  Boxplot for `closing_price` and `Volume`
plt.figure(figsize=(10, 6))
sns.boxplot(data=final_df[['closing_price', 'Volume']])
plt.title('Boxplot of Closing Price and Volume')
plt.show()
```



```
#  Sentiment distribution
plt.figure(figsize=(10, 6))
sns.histplot(final_df['sentiment_score_y'], kde=True, bins=30)
plt.title('Distribution of Sentiment Scores')
plt.show()
```



```
#  Distribution of `tomorrow_price`
plt.figure(figsize=(10, 6))
sns.histplot(final_df['tomorrow_price'], kde=True, bins=30)
plt.title('Distribution of Tomorrow\'s Price')
plt.show()
```



```
# Check for missing values after cleaning
print("\nMissing values in the dataset after cleaning:")
print(final_df.isnull().sum())
```

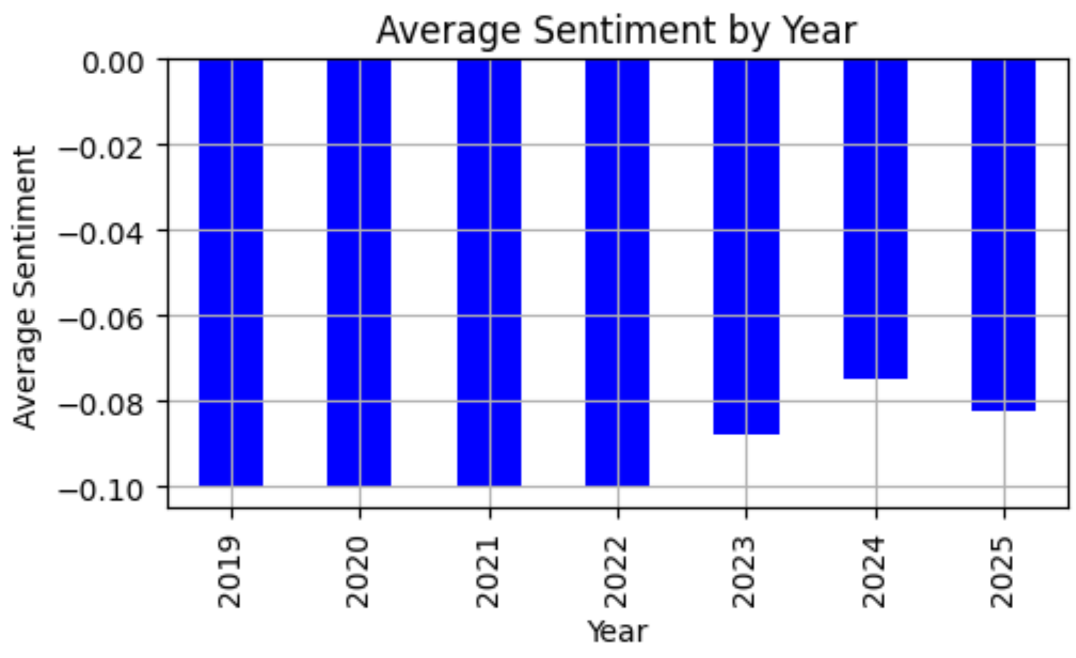
```
Missing values in the dataset after cleaning:
Date                   0
closing_price          0
High                   0
Low                    0
Open                   0
Volume                 0
wikipedia_sentiment    0
tomorrow_price         1
sentiment_score_y      0
dtype: int64
```

```
#  Checking the number of duplicates after cleaning
duplicates_after_cleaning = final_df.duplicated().sum()
print(f"\nNumber of duplicate rows after cleaning: {duplicates_after_cleaning}")
```

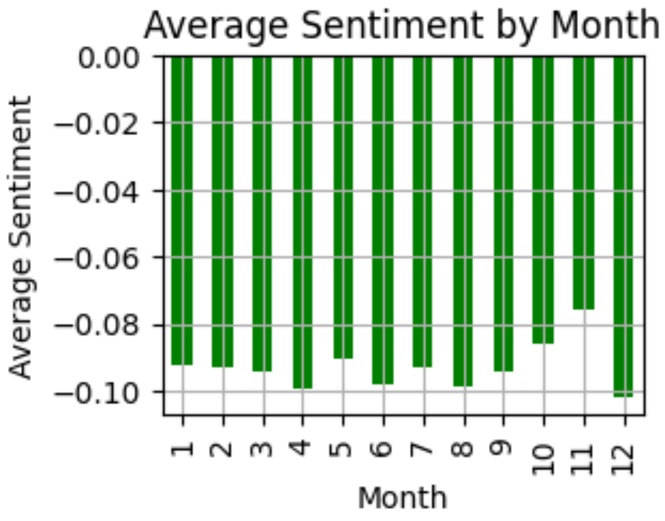
```
Number of duplicate rows after cleaning: 0
```

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

# Load your merged dataset
file_path = "/content/drive/MyDrive/Final_Project_Docs/cleaned_financial_data_wit
df = pd.read_csv(file_path)

# Convert 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Create a 'Year', 'Month', and 'Weekday' columns for easy grouping
df['Year'] = df['Date'].dt.year
```

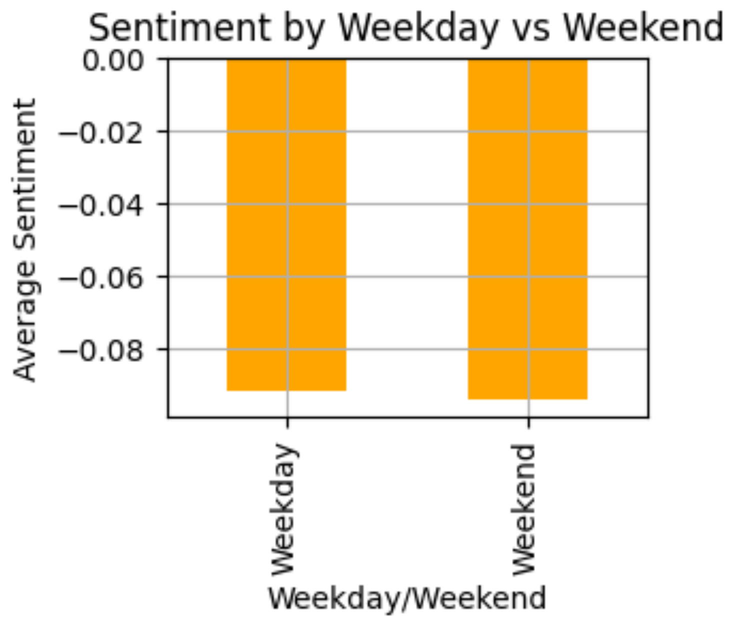
```
df['Month'] = df['Date'].dt.month
df['Weekday'] = df['Date'].dt.weekday
df['Weekend'] = df['Weekday'].apply(lambda x: 'Weekend' if x >= 5 else 'Weekday')

# 1. Sentiment by Year
sentiment_by_year = df.groupby('Year')['wikipedia_sentiment'].mean()

# 2. Sentiment by Month
sentiment_by_month = df.groupby('Month')['wikipedia_sentiment'].mean()

# 3. Sentiment by Weekday vs Weekend
sentiment_by_weekday = df.groupby('Weekend')['wikipedia_sentiment'].mean()

# Plotting
plt.figure(figsize=(12, 6))

# Plot Sentiment by Year
plt.subplot(2, 2, 1)
sentiment_by_year.plot(kind='bar', color='blue')
plt.title("Average Sentiment by Year")
plt.xlabel("Year")
plt.ylabel("Average Sentiment")
plt.grid(True)
```



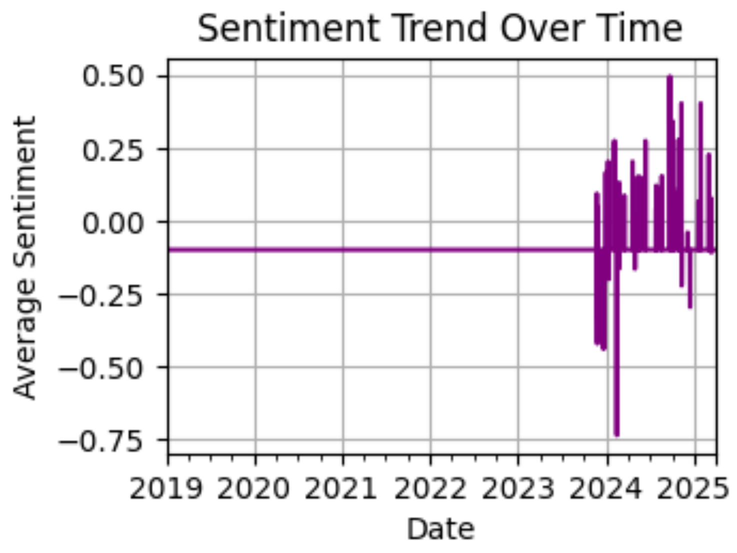
```
# Plot Sentiment by Month
plt.subplot(2, 2, 2)
sentiment_by_month.plot(kind='bar', color='green')
plt.title("Average Sentiment by Month")
plt.xlabel("Month")
plt.ylabel("Average Sentiment")
plt.grid(True)
```



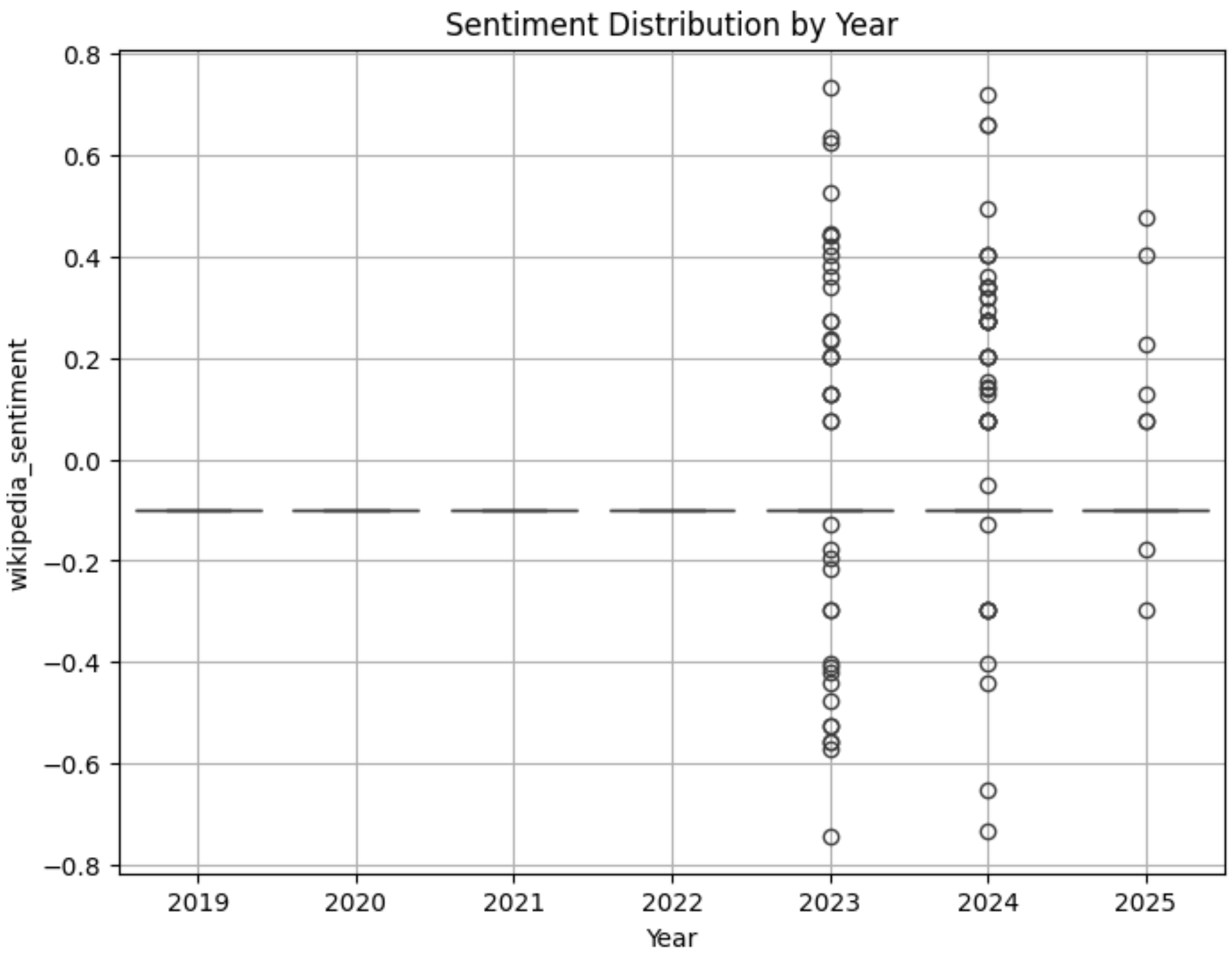
```
# Plot Sentiment by Weekday vs Weekend
plt.subplot(2, 2, 3)
sentiment_by_weekday.plot(kind='bar', color='orange')
plt.title("Sentiment by Weekday vs Weekend")
plt.xlabel("Weekday/Weekend")
plt.ylabel("Average Sentiment")
plt.grid(True)
```



```
# Sentiment over Time (Line Plot)
plt.subplot(2, 2, 4)
df.groupby('Date')['wikipedia_sentiment'].mean().plot(kind='line', color='purple'
plt.title("Sentiment Trend Over Time")
plt.xlabel("Date")
plt.ylabel("Average Sentiment")
plt.grid(True)

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

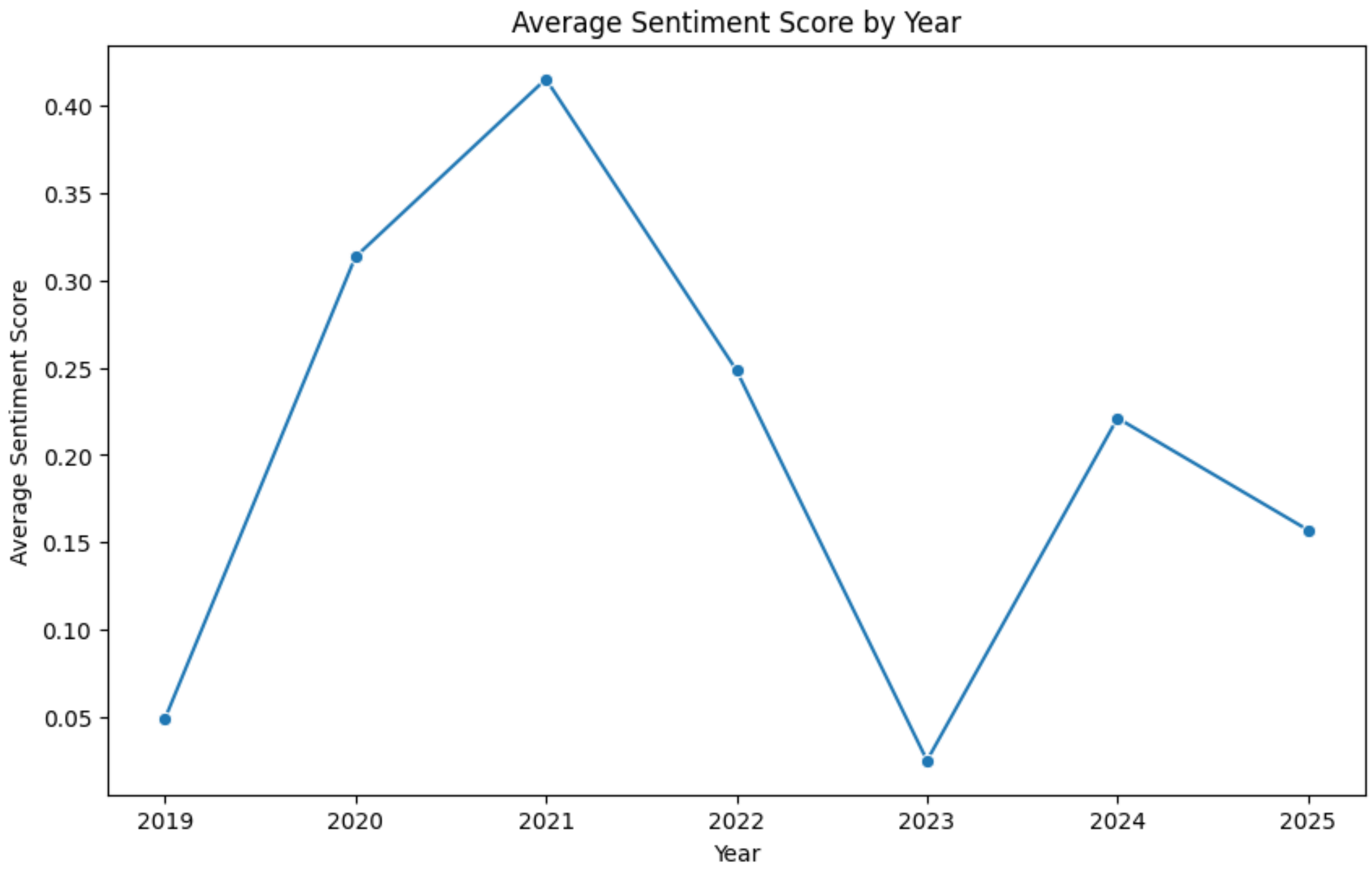


```
# Boxplot for distribution of sentiment by Year
plt.figure(figsize=(8, 6))
sns.boxplot(x='Year', y='wikipedia_sentiment', data=df)
plt.title("Sentiment Distribution by Year")
plt.grid(True)
plt.show()
```



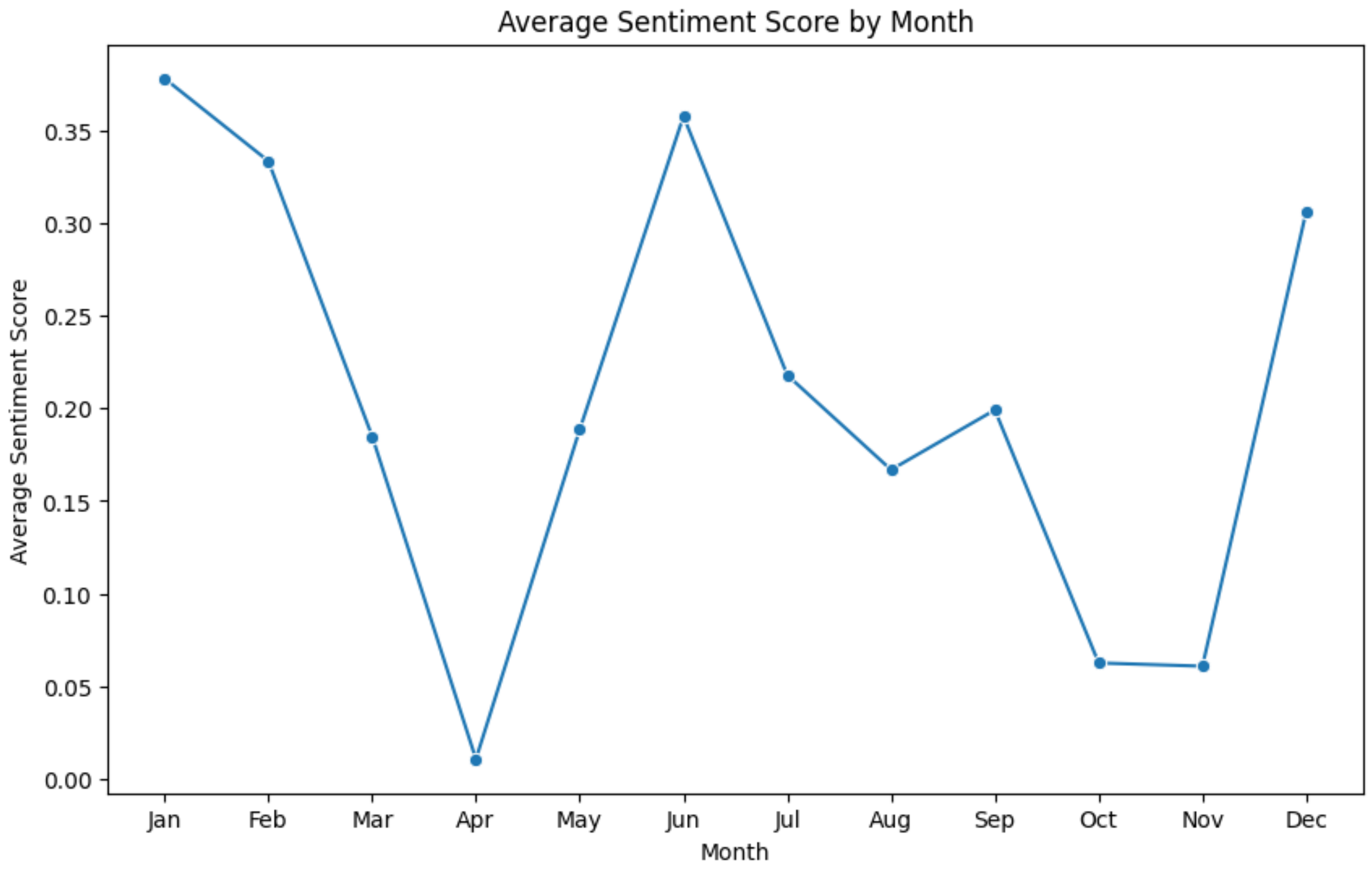
```
# Extract the year from the 'Date' column
final_df['Year'] = pd.to_datetime(final_df['Date']).dt.year

# Average sentiment per year
```

```
sentiment_by_year = final_df.groupby('Year')['sentiment_score_y'].mean().reset_in

# Plot sentiment by year
plt.figure(figsize=(10, 6))
sns.lineplot(data=sentiment_by_year, x='Year', y='sentiment_score_y', marker='o')
plt.title("Average Sentiment Score by Year")
plt.xlabel("Year")
plt.ylabel("Average Sentiment Score")
plt.show()
```



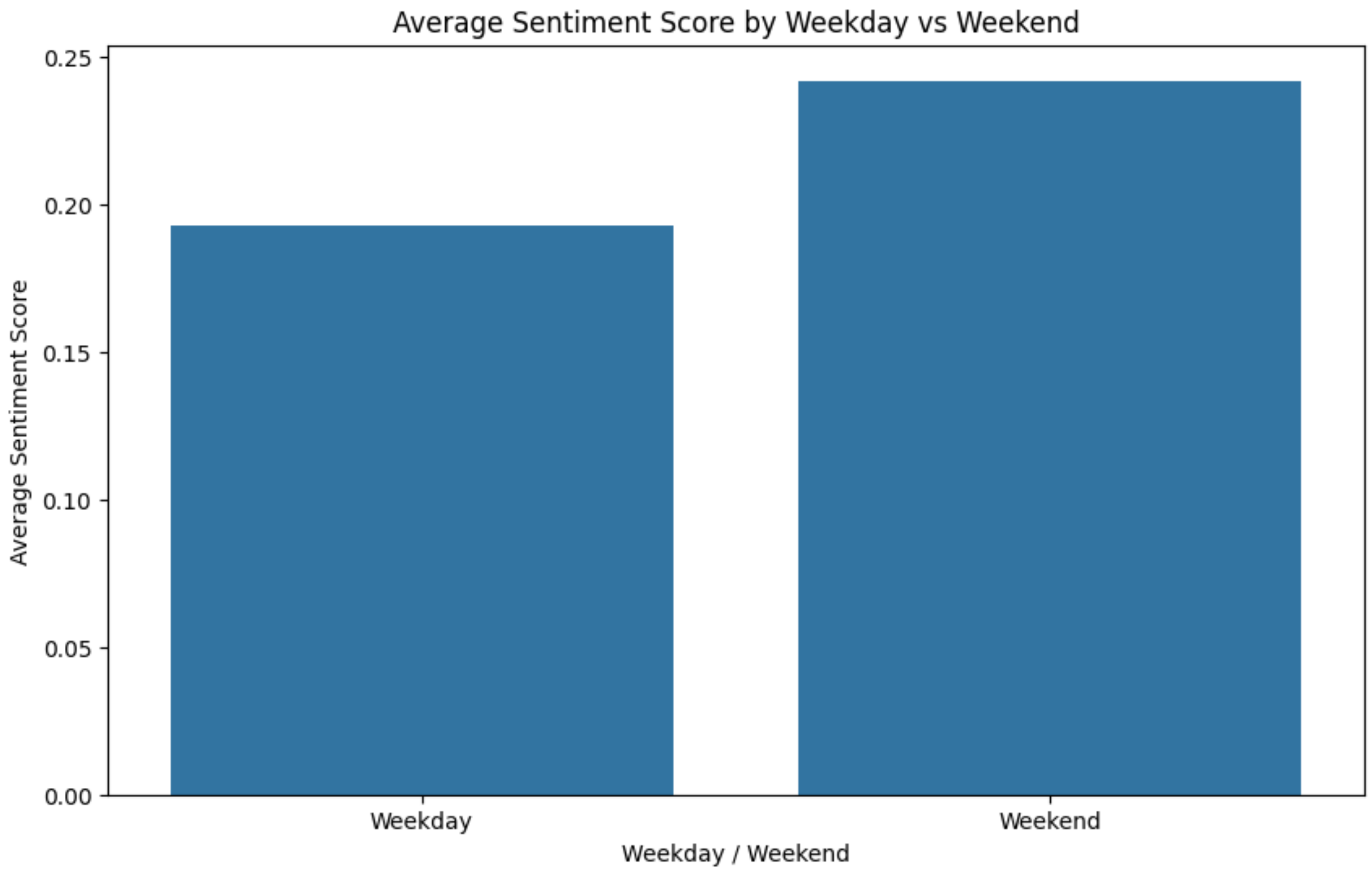
```
# Extract the month from the 'Date' column
final_df['Month'] = pd.to_datetime(final_df['Date']).dt.month
```

```
# Average sentiment per month
sentiment_by_month = final_df.groupby('Month')['sentiment_score_y'].mean().reset_

# Plot sentiment by month
plt.figure(figsize=(10, 6))
sns.lineplot(data=sentiment_by_month, x='Month', y='sentiment_score_y', marker='o
plt.title("Average Sentiment Score by Month")
plt.xlabel("Month")
plt.ylabel("Average Sentiment Score")
plt.xticks(ticks=np.arange(1, 13), labels=["Jan", "Feb", "Mar", "Apr", "May", "Ju
plt.show()
```

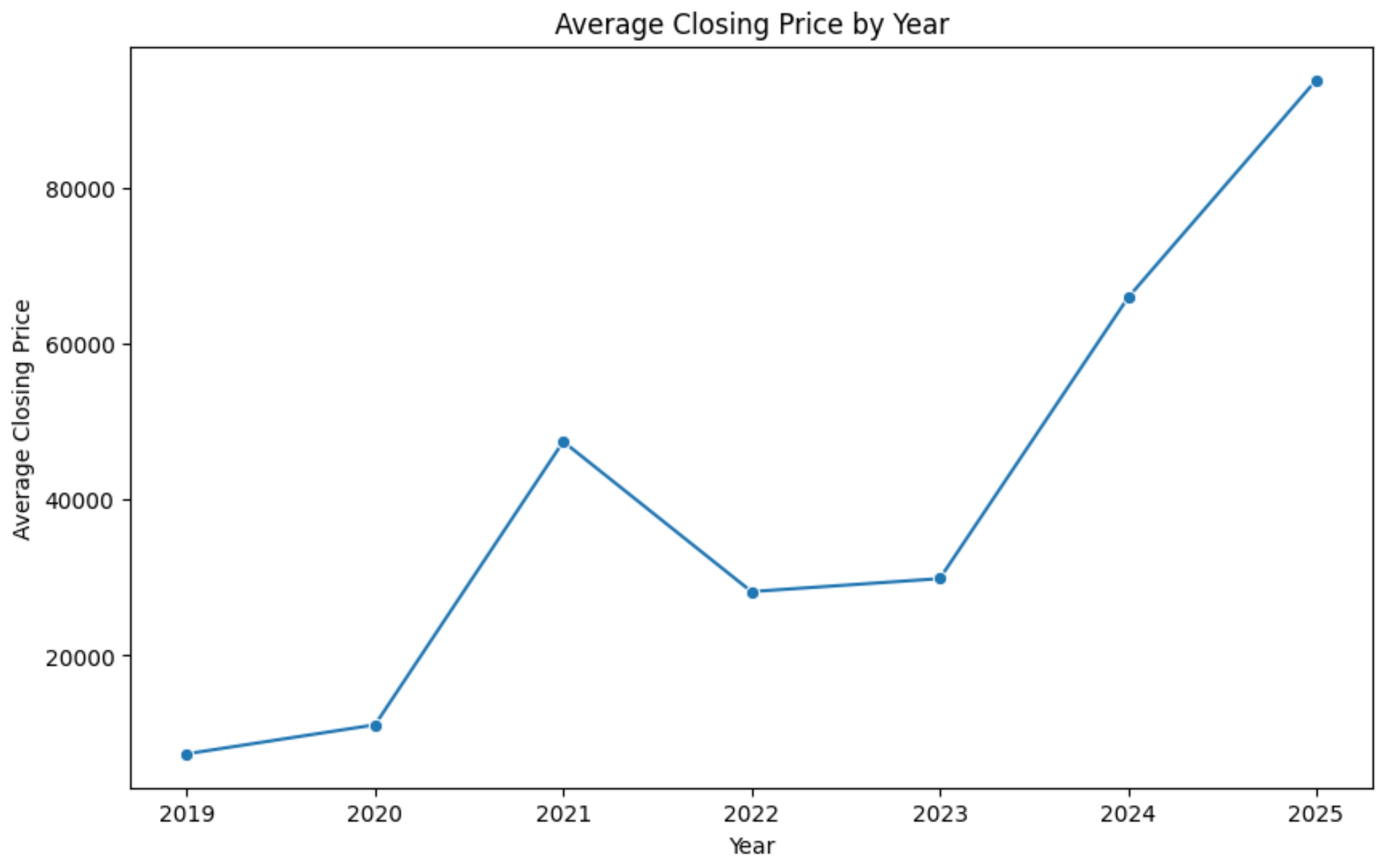


```
# Extract the day of the week from the 'Date' column (0=Monday, 6=Sunday)
```

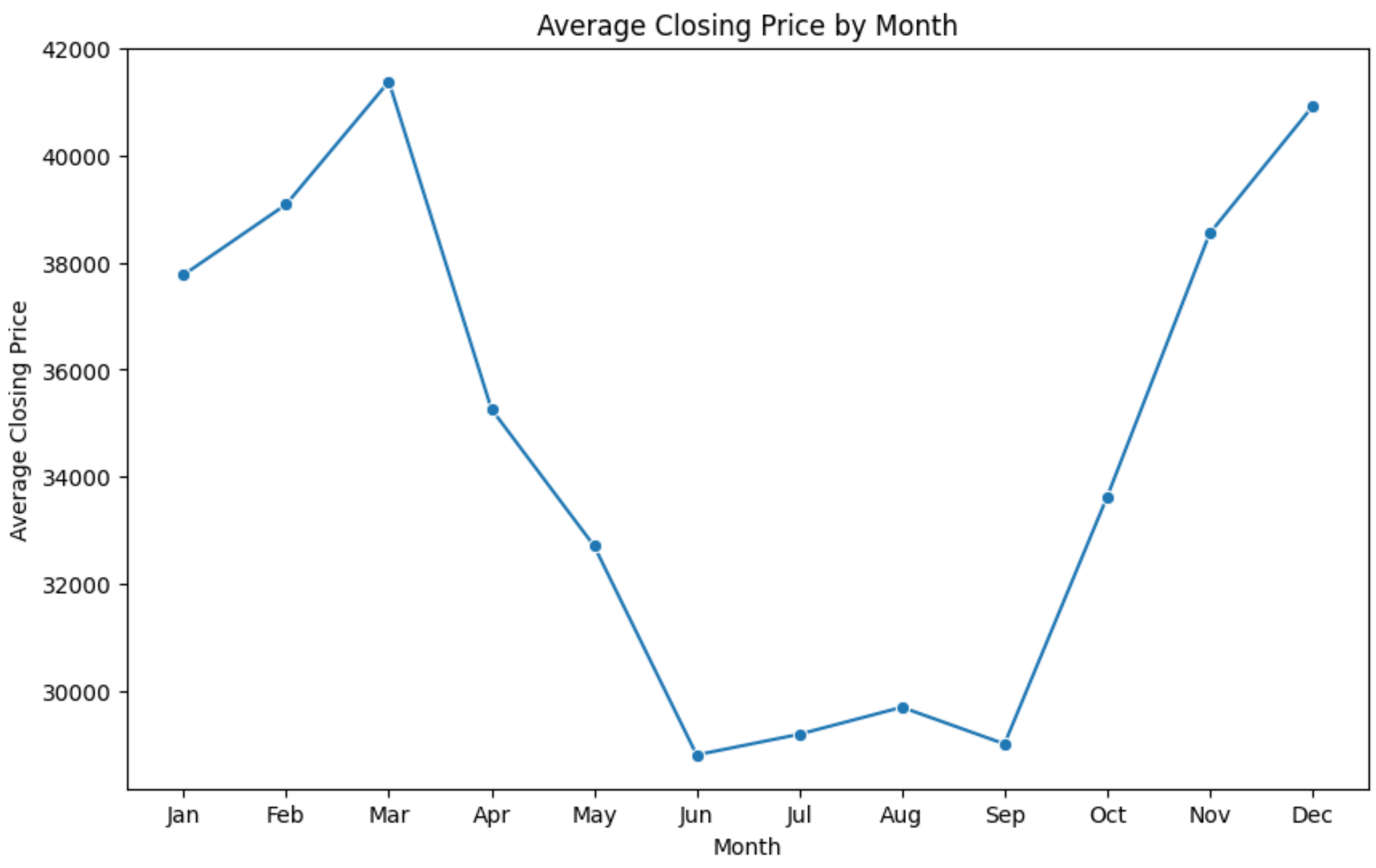
```
final_df['Weekday'] = pd.to_datetime(final_df['Date']).dt.weekday

# Create a new column to distinguish weekdays and weekends
final_df['Weekend'] = final_df['Weekday'].apply(lambda x: 'Weekend' if x >= 5 els

# Average sentiment score by weekday/weekend
sentiment_by_weekday = final_df.groupby('Weekend')['sentiment_score_y'].mean().re

# Plot sentiment by weekday/weekend
plt.figure(figsize=(10, 6))
sns.barplot(data=sentiment_by_weekday, x='Weekend', y='sentiment_score_y')
plt.title("Average Sentiment Score by Weekday vs Weekend")
plt.xlabel("Weekday / Weekend")
plt.ylabel("Average Sentiment Score")
plt.show()
```


Average Sentiment Score by Weekday vs Weekend
Average Sentiment Score
0.25
0.20
0.15
0.10
0.05
0.00
Weekday
Weekend
Weekday / Weekend

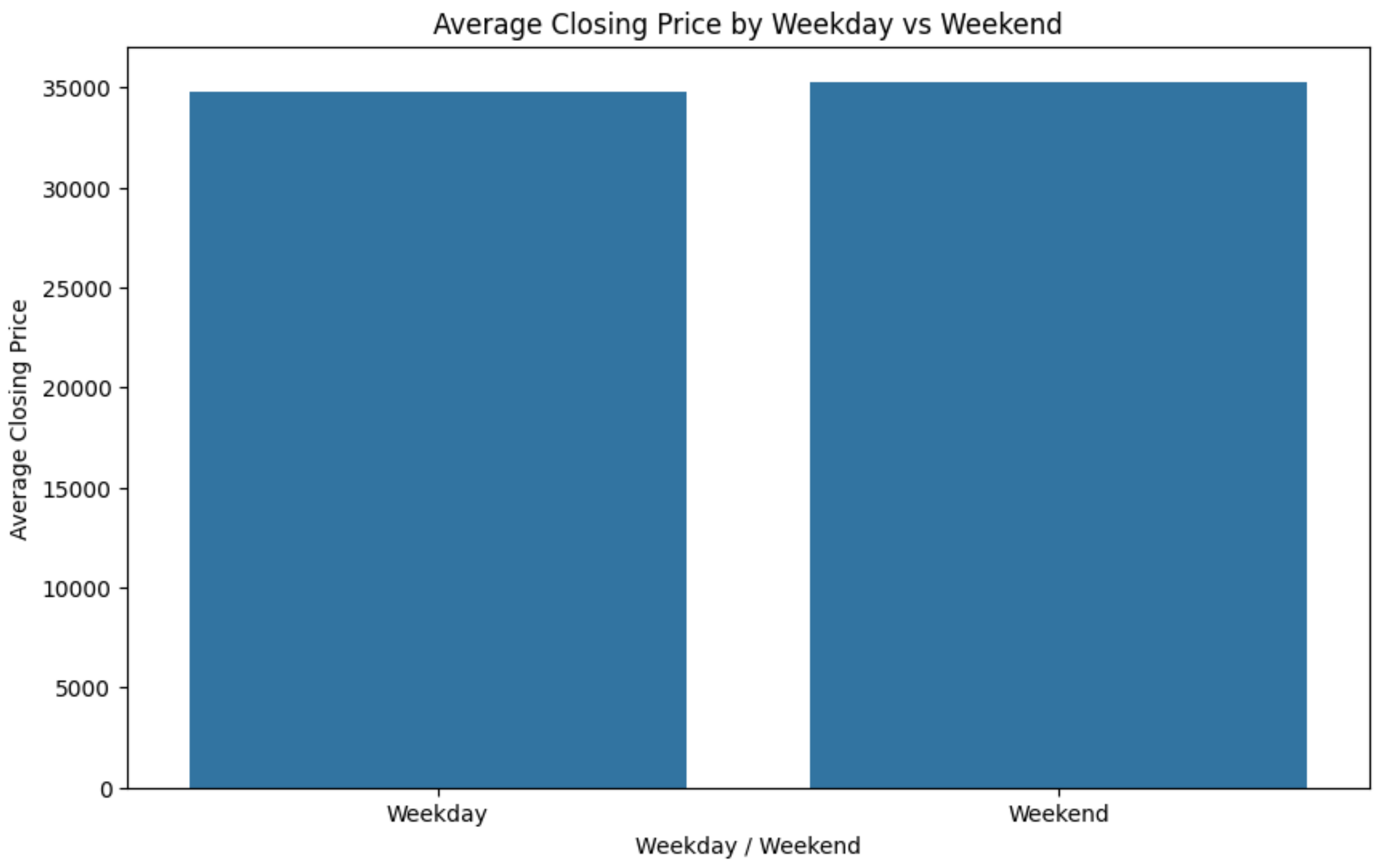
```
# Average closing price by year
closing_price_by_year = final_df.groupby('Year')['closing_price'].mean().reset_in

# Plot closing price by year
plt.figure(figsize=(10, 6))
sns.lineplot(data=closing_price_by_year, x='Year', y='closing_price', marker='o')
plt.title("Average Closing Price by Year")
plt.xlabel("Year")
plt.ylabel("Average Closing Price")
plt.show()
```



```
# Average closing price by month
closing_price_by_month = final_df.groupby('Month')['closing_price'].mean().reset_

# Plot closing price by month
plt.figure(figsize=(10, 6))
sns.lineplot(data=closing_price_by_month, x='Month', y='closing_price', marker='o
plt.title("Average Closing Price by Month")
plt.xlabel("Month")
plt.ylabel("Average Closing Price")
plt.xticks(ticks=np.arange(1, 13), labels=["Jan", "Feb", "Mar", "Apr", "May", "Ju
plt.show()
```



```
# Average closing price by weekday/weekend
closing_price_by_weekday = final_df.groupby('Weekend')['closing_price'].mean().re

# Plot closing price by weekday/weekend
plt.figure(figsize=(10, 6))
sns.barplot(data=closing_price_by_weekday, x='Weekend', y='closing_price')
plt.title("Average Closing Price by Weekday vs Weekend")
plt.xlabel("Weekday / Weekend")
plt.ylabel("Average Closing Price")
plt.show()
```



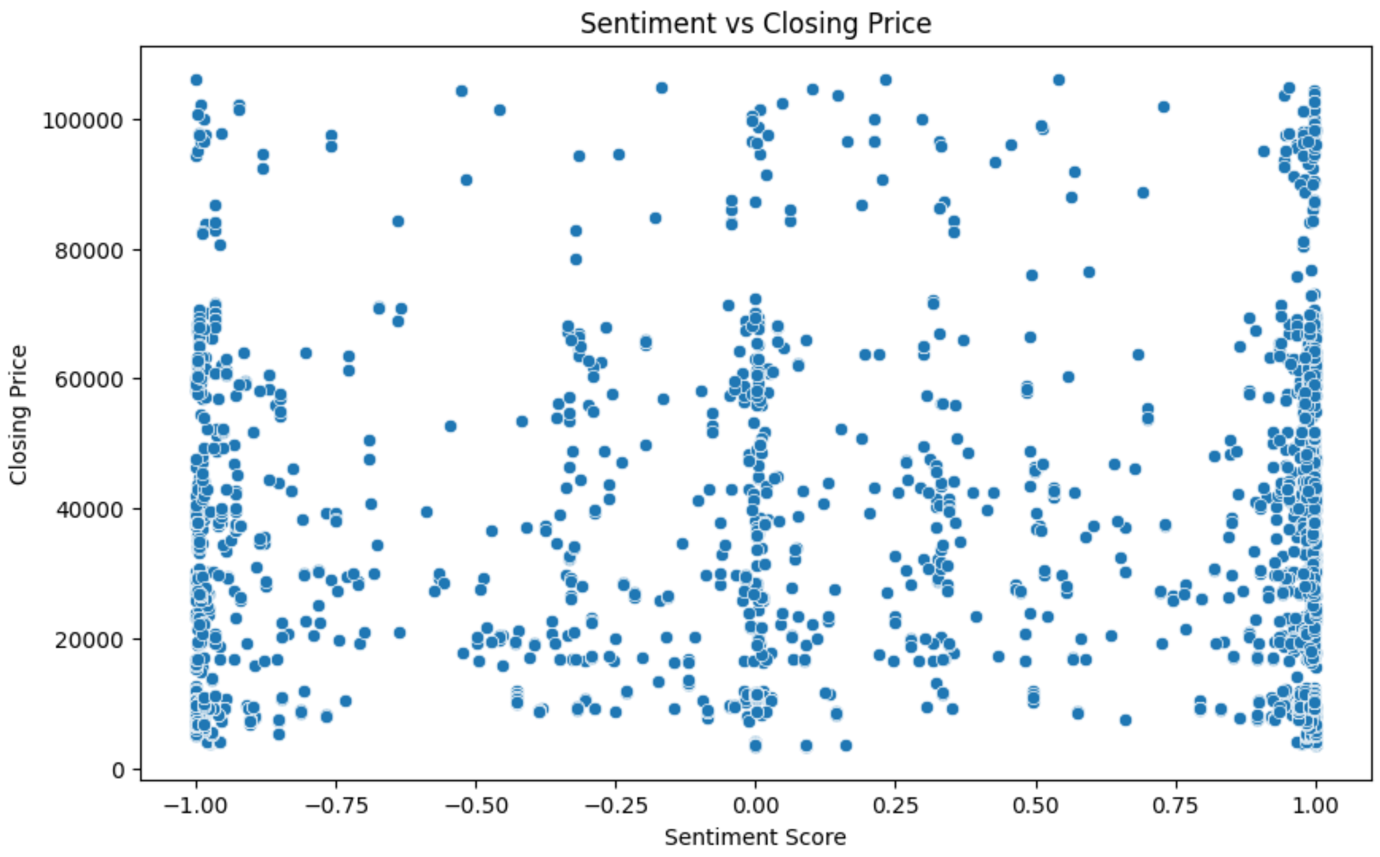
```
# Calculate the correlation between sentiment and closing price
sentiment_closing_corr = final_df['sentiment_score_y'].corr(final_df['closing_pri

print(f"Correlation between Sentiment and Closing Price: {sentiment_closing_corr}
```

```
Correlation between Sentiment and Closing Price: 0.0518403048751315
```

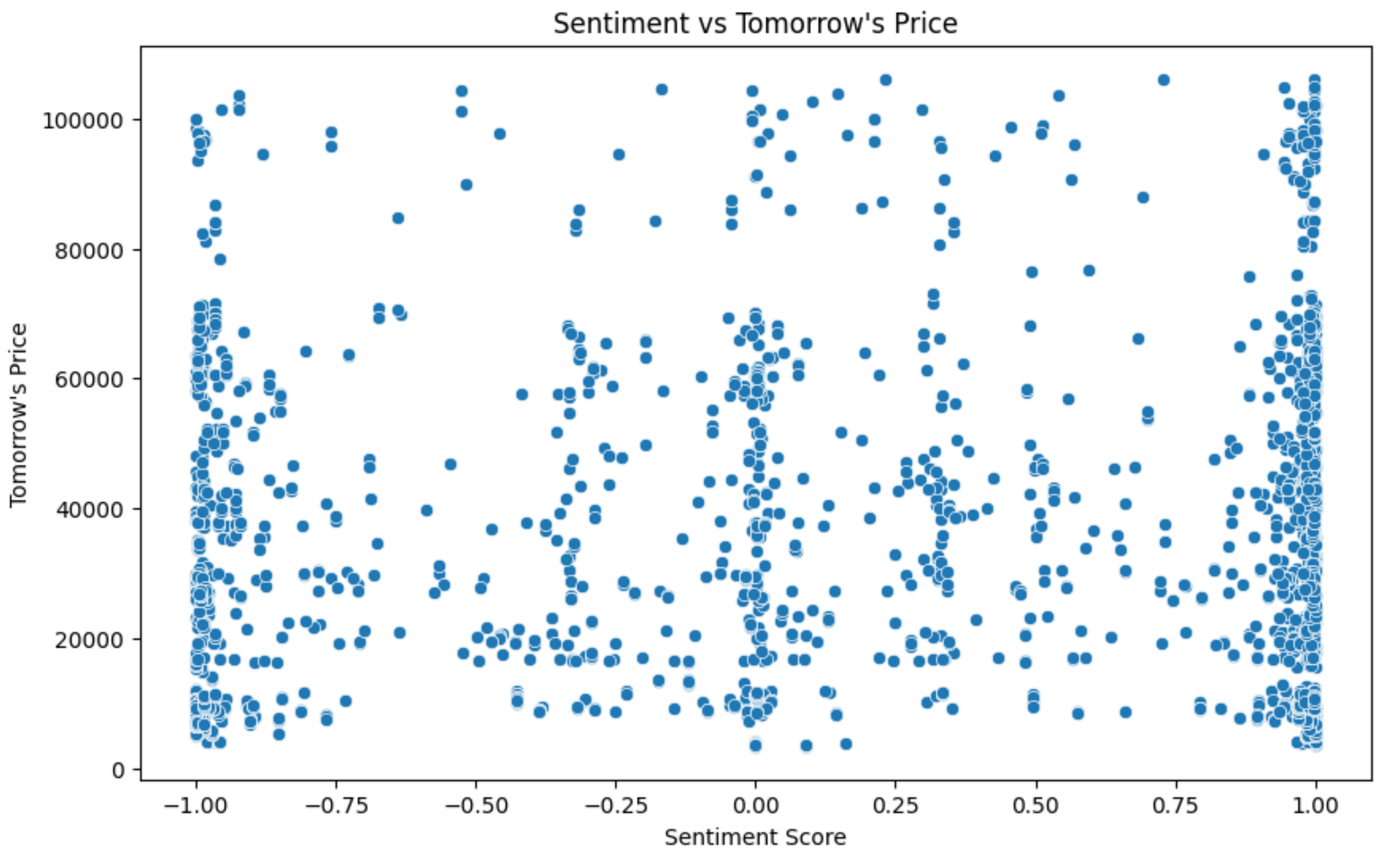
```
# Calculate the correlation between sentiment and tomorrow's price
sentiment_tomorrow_corr = final_df['sentiment_score_y'].corr(final_df['tomorrow_p

print(f"Correlation between Sentiment and Tomorrow's Price: {sentiment_tomorrow_c
```

```
Correlation between Sentiment and Tomorrow's Price: 0.05235583380174343
```

```
# Plot sentiment vs closing price
plt.figure(figsize=(10, 6))
sns.scatterplot(data=final_df, x='sentiment_score_y', y='closing_price')
plt.title("Sentiment vs Closing Price")
plt.xlabel("Sentiment Score")
plt.ylabel("Closing Price")
plt.show()
```



```
# Plot sentiment vs tomorrow's price
plt.figure(figsize=(10, 6))
sns.scatterplot(data=final_df, x='sentiment_score_y', y='tomorrow_price')
plt.title("Sentiment vs Tomorrow's Price")
plt.xlabel("Sentiment Score")
plt.ylabel("Tomorrow's Price")
plt.show()
```



```
import pandas as pd

# Load the cleaned dataset
final_df = pd.read_csv('/content/drive/MyDrive/Final_Project_Docs/cleaned_financial
# Drop rows where sentiment_score_y is 0 (prior to sentiment data)
final_df = final_df[final_df['sentiment_score_y'] != 0]

# Reset index to start from 0 for clean counting
final_df.reset_index(drop=True, inplace=True)

# Inspect the first few rows to ensure the data is correct
print(final_df.head())

# Check the total number of rows remaining
print(f"Number of rows after cleaning: {len(final_df)}")

# Check the structure of the dataset
print(final_df.info())

# Check for any missing values
print(final_df.isnull().sum())

# Save the cleaned dataset
final_df.to_csv('/content/drive/MyDrive/Final_Project_Docs/Final_cleaned_Bitcoin_da

print("Dataset cleaned and saved successfully.")
```

```
         Date  closing_price         High          Low         Open  \
0  2019-01-31    3457.792725  3504.804932  3447.915771  3485.409180
1  2019-02-01    3487.945312  3501.954102  3431.591553  3460.547119
2  2019-02-02    3521.060791  3523.287354  3467.574707  3484.625977
3  2019-02-03    3464.013428  3521.388184  3447.924316  3516.139648
4  2019-02-04    3459.154053  3476.223877  3442.586914  3467.211670

       Volume  wikipedia_sentiment  tomorrow_price  sentiment_score_y
0  5831198271                 -0.1     3487.945312             0.9998
1  5422926707                 -0.1     3521.060791             0.9998
2  5071623601                 -0.1     3464.013428             0.9998
3  5043937584                 -0.1     3459.154053             0.9998
4  5332718886                 -0.1     3466.357422             0.0884
Number of rows after cleaning: 2363
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2363 entries, 0 to 2362
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Date                 2363 non-null   object
 1   closing_price        2363 non-null   float64
 2   High                 2363 non-null   float64
 3   Low                  2363 non-null   float64
 4   Open                 2363 non-null   float64
 5   Volume               2363 non-null   int64
 6   wikipedia_sentiment  2363 non-null   float64
 7   tomorrow_price       2362 non-null   float64
 8   sentiment_score_y    2363 non-null   float64
dtypes: float64(7), int64(1), object(1)
memory usage: 166.3+ KB
None
Date                   0
closing_price          0
High                   0
Low                    0
Open                   0
Volume                 0
wikipedia_sentiment    0
tomorrow_price         1
sentiment_score_y      0
dtype: int64
Dataset cleaned and saved successfully.
```

```
import pandas as pd
import numpy as np

#  Create lag feature (Previous day's closing price)
final_df['lag_1'] = final_df['closing_price'].shift(1)
```

```
# Calculate 30-Day Moving Average
final_df['30_MA'] = final_df['closing_price'].rolling(window=30).mean()

# Calculate 14-Day Relative Strength Index (RSI)
delta = final_df['closing_price'].diff()  # Change in price
gain = (delta.where(delta > 0, 0)).rolling(window=14).mean()  # Gain for positive
loss = (-delta.where(delta < 0, 0)).rolling(window=14).mean()  # Loss for negativ

# Avoid division by zero
rs = gain / loss
final_df['RSI_14'] = 100 - (100 / (1 + rs))

# Calculate MACD and Signal line
final_df['26_EMA'] = final_df['closing_price'].ewm(span=26, adjust=False).mean()
final_df['12_EMA'] = final_df['closing_price'].ewm(span=12, adjust=False).mean()
final_df['MACD'] = final_df['12_EMA'] - final_df['26_EMA']
final_df['MACD_signal'] = final_df['MACD'].ewm(span=9, adjust=False).mean()
final_df['MACD_histogram'] = final_df['MACD'] - final_df['MACD_signal']

# Calculate Bollinger Bands (Upper, Middle, and Lower)
final_df['30_MA'] = final_df['closing_price'].rolling(window=30).mean()  # Recalc
final_df['30_STD'] = final_df['closing_price'].rolling(window=30).std()  # Standa
final_df['Bollinger_upper'] = final_df['30_MA'] + (final_df['30_STD'] * 2)  # Upp
final_df['Bollinger_lower'] = final_df['30_MA'] - (final_df['30_STD'] * 2)  # Low

# Fill missing values (forward fill)
final_df.fillna(method='ffill', inplace=True)

# Save the updated dataset with new features
final_file_path = '/content/drive/MyDrive/Final_Project_Docs/bitcoin_data_with_fe
final_df.to_csv(final_file_path, index=False)

# Display first few rows of the updated DataFrame
print("Data with features added successfully!")
print(final_df.head())
```

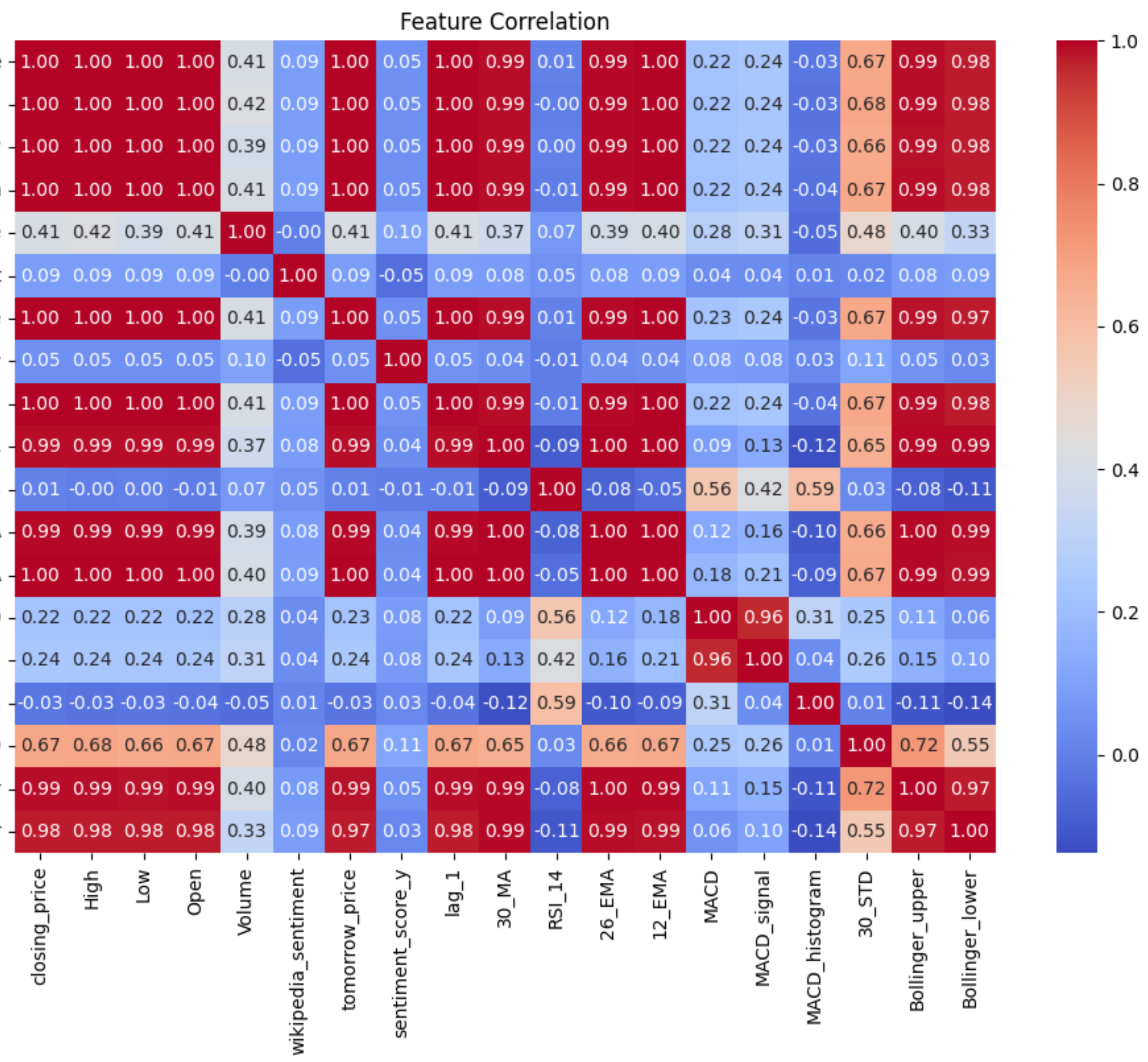
```
<ipython-input-85-6ad188f77552>:33: FutureWarning: DataFrame.fillna with 'meth
  final_df.fillna(method='ffill', inplace=True)
Data with features added successfully!
         Date  closing_price         High          Low         Open  \
0  2019-01-31    3457.792725  3504.804932  3447.915771  3485.409180
1  2019-02-01    3487.945312  3501.954102  3431.591553  3460.547119
2  2019-02-02    3521.060791  3523.287354  3467.574707  3484.625977
3  2019-02-03    3464.013428  3521.388184  3447.924316  3516.139648
4  2019-02-04    3459.154053  3476.223877  3442.586914  3467.211670

       Volume  wikipedia_sentiment  tomorrow_price  sentiment_score_y  \
0  5831198271                 -0.1     3487.945312             0.9998
1  5422926707                 -0.1     3521.060791             0.9998
2  5071623601                 -0.1     3464.013428             0.9998
3  5043937584                 -0.1     3459.154053             0.9998
4  5332718886                 -0.1     3466.357422             0.0884

         lag_1  30_MA  RSI_14       26_EMA       12_EMA      MACD  \
0          NaN    NaN     NaN  3457.792725  3457.792725  0.000000
1  3457.792725    NaN     NaN  3460.026250  3462.431584  2.405335
2  3487.945312    NaN     NaN  3464.547327  3471.451462  6.904135
3  3521.060791    NaN     NaN  3464.507779  3470.307149  5.799371
4  3464.013428    NaN     NaN  3464.111206  3468.591288  4.480082

   MACD_signal  MACD_histogram  30_STD  Bollinger_upper  Bollinger_lower
0     0.000000        0.000000     NaN              NaN              NaN
1     0.481067        1.924268     NaN              NaN              NaN
2     1.765681        5.138455     NaN              NaN              NaN
3     2.572419        3.226952     NaN              NaN              NaN
4     2.953951        1.526131     NaN              NaN              NaN
```

```
# Forward fill missing values
final_df.ffill(inplace=True)



# Inspect the first few rows again
print(final_df.head())

# Check for any remaining missing values
print(final_df.isnull().sum())
```

```
         Date  closing_price         High          Low         Open  \
0  2019-01-31    3457.792725  3504.804932  3447.915771  3485.409180   
1  2019-02-01    3487.945312  3501.954102  3431.591553  3460.547119   
2  2019-02-02    3521.060791  3523.287354  3467.574707  3484.625977   
3  2019-02-03    3464.013428  3521.388184  3447.924316  3516.139648   
4  2019-02-04    3459.154053  3476.223877  3442.586914  3467.211670   

       Volume  wikipedia_sentiment  tomorrow_price  sentiment_score_y  \
0  5831198271                 -0.1     3487.945312             0.9998   
1  5422926707                 -0.1     3521.060791             0.9998   
2  5071623601                 -0.1     3464.013428             0.9998   
3  5043937584                 -0.1     3459.154053             0.9998   
4  5332718886                 -0.1     3466.357422             0.0884   

         lag_1  30_MA  RSI_14       26_EMA       12_EMA      MACD  \
0          NaN    NaN     NaN  3457.792725  3457.792725  0.000000   
1  3457.792725    NaN     NaN  3460.026250  3462.431584  2.405335   
2  3487.945312    NaN     NaN  3464.547327  3471.451462  6.904135   
3  3521.060791    NaN     NaN  3464.507779  3470.307149  5.799371   
4  3464.013428    NaN     NaN  3464.111206  3468.591288  4.480082   

   MACD_signal  MACD_histogram  30_STD  Bollinger_upper  Bollinger_lower  
0     0.000000        0.000000     NaN              NaN              NaN  
1     0.481067        1.924268     NaN              NaN              NaN  
2     1.765681        5.138455     NaN              NaN              NaN  
3     2.572419        3.226952     NaN              NaN              NaN  
4     2.953951        1.526131     NaN              NaN              NaN  
Date                    0
closing_price           0
High                    0
Low                     0
Open                    0
Volume                  0
wikipedia_sentiment     0
tomorrow_price          0
sentiment_score_y       0
lag_1                   1
30_MA                  29
RSI_14                 13
26_EMA                  0
12_EMA                  0
MACD                    0
MACD_signal             0
MACD_histogram          0
30_STD                 29
Bollinger_upper        29
Bollinger_lower        29
dtype: int64
```

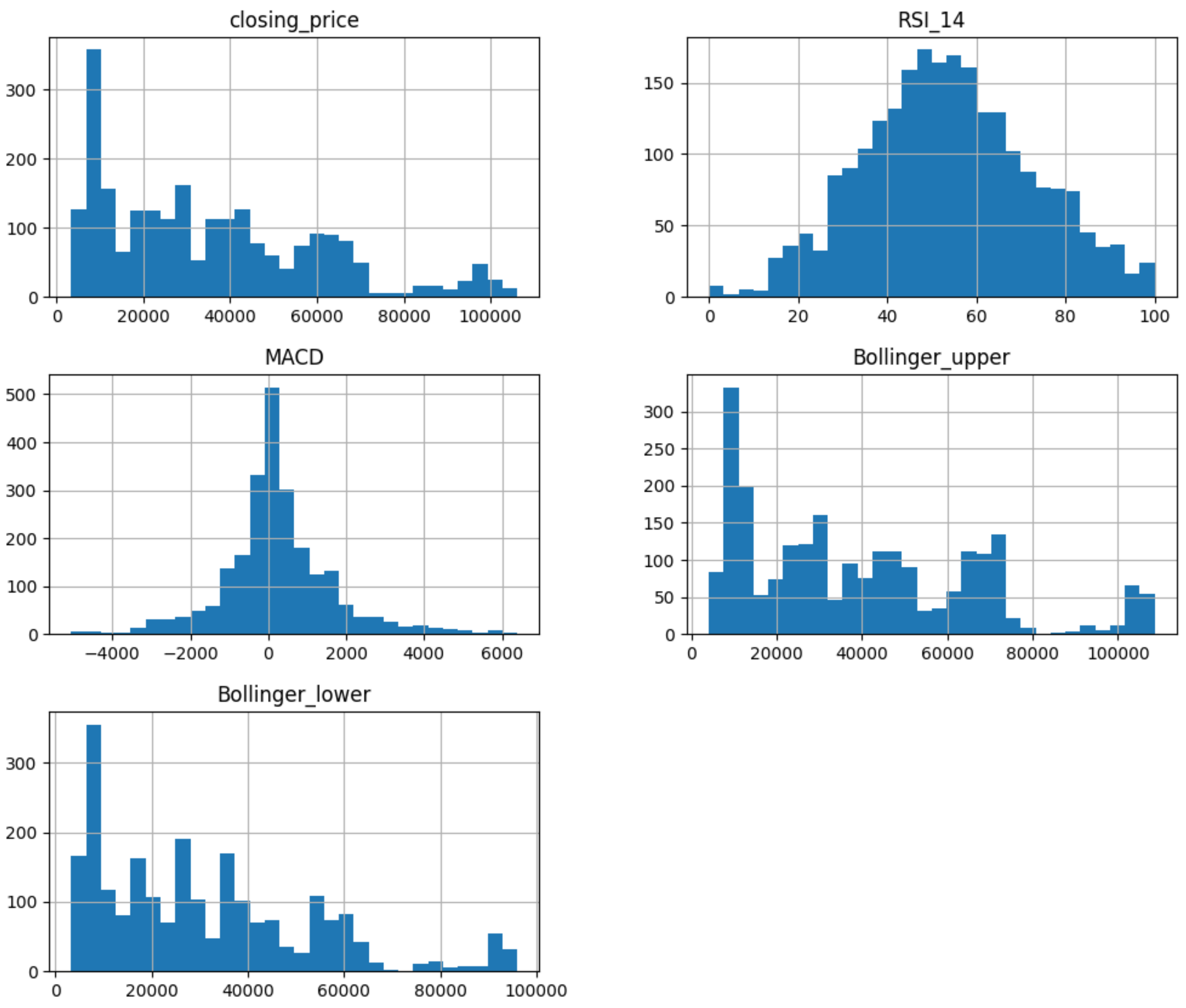
EDA

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

# Select only numerical features for correlation calculation
numerical_features = final_df.select_dtypes(include=np.number)

# Correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(numerical_features.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation")
plt.show()
```



	closing_price	High	Low	Open	Volume	wikipedia_sentiment	tomorrow_price	sentiment_score_y	lag_1	30_MA	RSI_14	26_EMA	12_EMA	MACD	MACD_signal	MACD_histogram	30_STD	Bollinger_upper	Bollinger_lower
closing_price	1.00	1.00	1.00	1.00	0.41	0.09	1.00	0.05	1.00	0.99	0.01	0.99	1.00	0.22	0.24	-0.03	0.67	0.99	0.98
High	1.00	1.00	1.00	1.00	0.42	0.09	1.00	0.05	1.00	0.99	-0.00	0.99	1.00	0.22	0.24	-0.03	0.68	0.99	0.98
Low	1.00	1.00	1.00	1.00	0.39	0.09	1.00	0.05	1.00	0.99	0.00	0.99	1.00	0.22	0.24	-0.03	0.66	0.99	0.98
Open	1.00	1.00	1.00	1.00	0.41	0.09	1.00	0.05	1.00	0.99	-0.01	0.99	1.00	0.22	0.24	-0.04	0.67	0.99	0.98
Volume	0.41	0.42	0.39	0.41	1.00	-0.00	0.41	0.10	0.41	0.37	0.07	0.39	0.40	0.28	0.31	-0.05	0.48	0.40	0.33
wikipedia_sentiment	0.09	0.09	0.09	0.09	-0.00	1.00	0.09	-0.05	0.09	0.08	0.05	0.08	0.09	0.04	0.04	0.01	0.02	0.08	0.09
tomorrow_price	1.00	1.00	1.00	1.00	0.41	0.09	1.00	0.05	1.00	0.99	0.01	0.99	1.00	0.23	0.24	-0.03	0.67	0.99	0.97
sentiment_score_y	0.05	0.05	0.05	0.05	0.10	-0.05	0.05	1.00	0.05	0.04	-0.01	0.04	0.04	0.08	0.08	0.03	0.11	0.05	0.03
lag_1	1.00	1.00	1.00	1.00	0.41	0.09	1.00	0.05	1.00	0.99	-0.01	0.99	1.00	0.22	0.24	-0.04	0.67	0.99	0.98
30_MA	0.99	0.99	0.99	0.99	0.37	0.08	0.99	0.04	0.99	1.00	-0.09	1.00	1.00	0.09	0.13	-0.12	0.65	0.99	0.99
RSI_14	0.01	-0.00	0.00	-0.01	0.07	0.05	0.01	-0.01	-0.01	-0.09	1.00	-0.08	-0.05	0.56	0.42	0.59	0.03	-0.08	-0.11
26_EMA	0.99	0.99	0.99	0.99	0.39	0.08	0.99	0.04	0.99	1.00	-0.08	1.00	1.00	0.12	0.16	-0.10	0.66	1.00	0.99
12_EMA	1.00	1.00	1.00	1.00	0.40	0.09	1.00	0.04	1.00	1.00	-0.05	1.00	1.00	0.18	0.21	-0.09	0.67	0.99	0.99
MACD	0.22	0.22	0.22	0.22	0.28	0.04	0.23	0.08	0.22	0.09	0.56	0.12	0.18	1.00	0.96	0.31	0.25	0.11	0.06
MACD_signal	0.24	0.24	0.24	0.24	0.31	0.04	0.24	0.08	0.24	0.13	0.42	0.16	0.21	0.96	1.00	0.04	0.26	0.15	0.10
MACD_histogram	-0.03	-0.03	-0.03	-0.04	-0.05	0.01	-0.03	0.03	-0.04	-0.12	0.59	-0.10	-0.09	0.31	0.04	1.00	0.01	-0.11	-0.14
30_STD	0.67	0.68	0.66	0.67	0.48	0.02	0.67	0.11	0.67	0.65	0.03	0.66	0.67	0.25	0.26	0.01	1.00	0.72	0.55
Bollinger_upper	0.99	0.99	0.99	0.99	0.40	0.08	0.99	0.05	0.99	0.99	-0.08	1.00	0.99	0.11	0.15	-0.11	0.72	1.00	0.97
Bollinger_lower	0.98	0.98	0.98	0.98	0.33	0.09	0.97	0.03	0.98	0.99	-0.11	0.99	0.99	0.06	0.10	-0.14	0.55	0.97	1.00

```
final_df[['closing_price', 'RSI_14', 'MACD', 'Bollinger_upper', 'Bollinger_lower'
plt.show()
```


closing_price
RSI_14
MACD
Bollinger_upper
Bollinger_lower

Combine Sentiment Features

```
print(final_df.columns)
```

```
Index(['Date', 'closing_price', 'High', 'Low', 'Open', 'Volume',
       'wikipedia_sentiment', 'tomorrow_price', 'sentiment_score_y', 'lag_1',
       '30_MA', 'RSI_14', '26_EMA', '12_EMA', 'MACD', 'MACD_signal',
       'MACD_histogram', '30_STD', 'Bollinger_upper', 'Bollinger_lower'],
      dtype='object')
```

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Load the existing dataset with features
final_df = pd.read_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_data_wi

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Rescale the 'wikipedia_sentiment' and 'sentiment_score_y' to the range [0, 1]
final_df[['wikipedia_sentiment', 'sentiment_score_y']] = scaler.fit_transform(fin

# Calculate Composite Sentiment Score (weighted average of scaled Wikipedia senti
final_df['composite_sentiment'] = (final_df['wikipedia_sentiment'] + final_df['se

# Inspect the first few rows to ensure the composite sentiment is added correctly
print(final_df[['Date', 'wikipedia_sentiment', 'sentiment_score_y', 'composite_se

# Save the updated dataset with composite sentiment
final_df.to_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_with_features_

print("Composite sentiment feature added successfully!")
```

```
         Date  wikipedia_sentiment  sentiment_score_y  composite_sentiment
0  2019-01-31             0.435018             0.9999             0.717459
1  2019-02-01             0.435018             0.9999             0.717459
2  2019-02-02             0.435018             0.9999             0.717459
3  2019-02-03             0.435018             0.9999             0.717459
4  2019-02-04             0.435018             0.5442             0.489609
Composite sentiment feature added successfully!
```

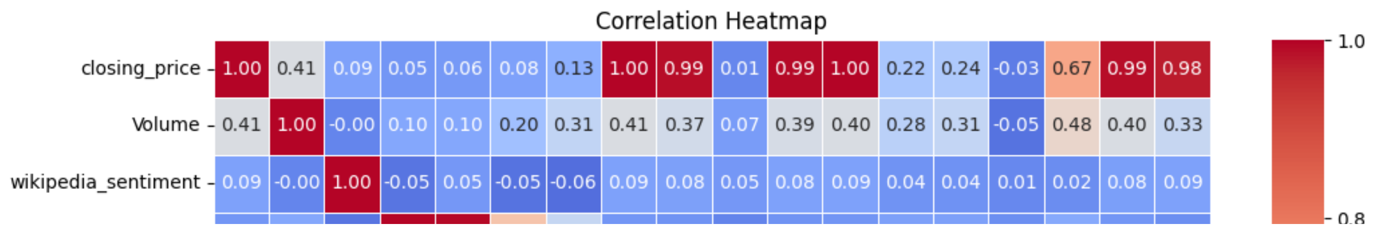
```
import pandas as pd

# Load the existing dataset with features
final_df = pd.read_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_data_wi

# Calculate Composite Sentiment Score (weighted average of Wikipedia sentiment an
final_df['composite_sentiment'] = (final_df['wikipedia_sentiment'] + final_df['se

# Inspect the first few rows to ensure the composite sentiment is added correctly
print(final_df[['Date', 'wikipedia_sentiment', 'sentiment_score_y', 'composite_se

# Save the updated dataset with composite sentiment
final_df.to_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_with_features_

print("Composite sentiment feature added successfully!")
```

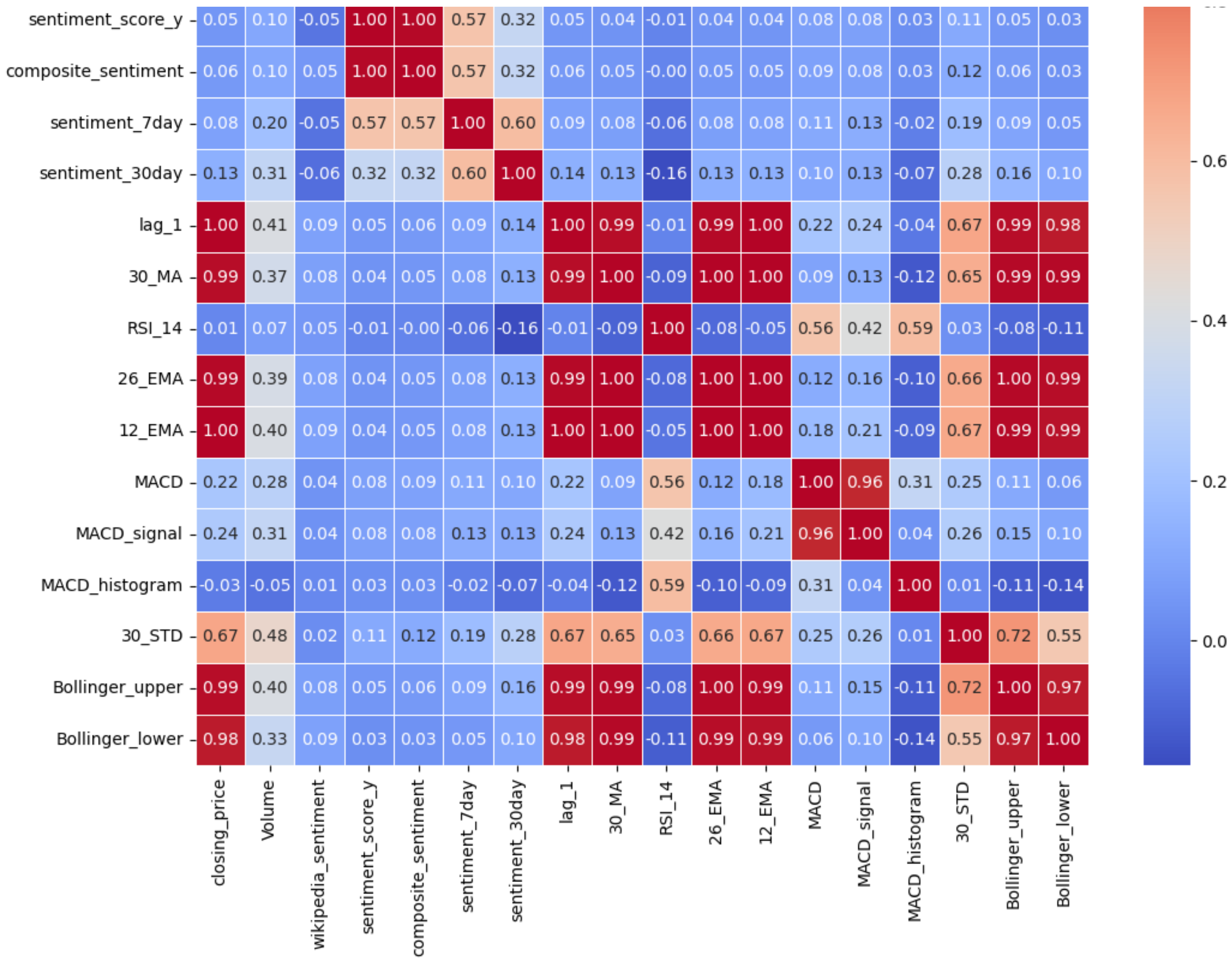
```
         Date  wikipedia_sentiment  sentiment_score_y  composite_sentiment
0  2019-01-31                 -0.1             0.9998               0.4499
1  2019-02-01                 -0.1             0.9998               0.4499
2  2019-02-02                 -0.1             0.9998               0.4499
3  2019-02-03                 -0.1             0.9998               0.4499
4  2019-02-04                 -0.1             0.0884              -0.0058
Composite sentiment feature added successfully!
```

```
# Create 7-day and 30-day rolling sentiment features
final_df['sentiment_7day'] = final_df['composite_sentiment'].rolling(window=7).mear
final_df['sentiment_30day'] = final_df['composite_sentiment'].rolling(window=30).me


# Fill missing values (backfill for rolling windows)
final_df['sentiment_7day'] = final_df['sentiment_7day'].bfill()
final_df['sentiment_30day'] = final_df['sentiment_30day'].bfill()


# Inspect the first few rows to ensure everything is added correctly
print(final_df[['Date', 'composite_sentiment', 'sentiment_7day', 'sentiment_30day']
```

```
         Date  composite_sentiment  sentiment_7day  sentiment_30day
0  2019-01-31               0.4499          0.2546         0.104453
1  2019-02-01               0.4499          0.2546         0.104453
2  2019-02-02               0.4499          0.2546         0.104453
3  2019-02-03               0.4499          0.2546         0.104453
4  2019-02-04              -0.0058          0.2546         0.104453
```

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

# Select the numeric columns for correlation analysis
numeric_cols = ['closing_price', 'Volume', 'wikipedia_sentiment', 'sentiment_scor
                'sentiment_7day', 'sentiment_30day', 'lag_1', '30_MA', 'RSI_14',
                'MACD_signal', 'MACD_histogram', '30_STD', 'Bollinger_upper', 'Bo

# Calculate the correlation matrix
correlation_matrix = final_df[numeric_cols].corr()

# Plot a heatmap to visualize the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", linewidth
plt.title("Correlation Heatmap")
plt.show()
```

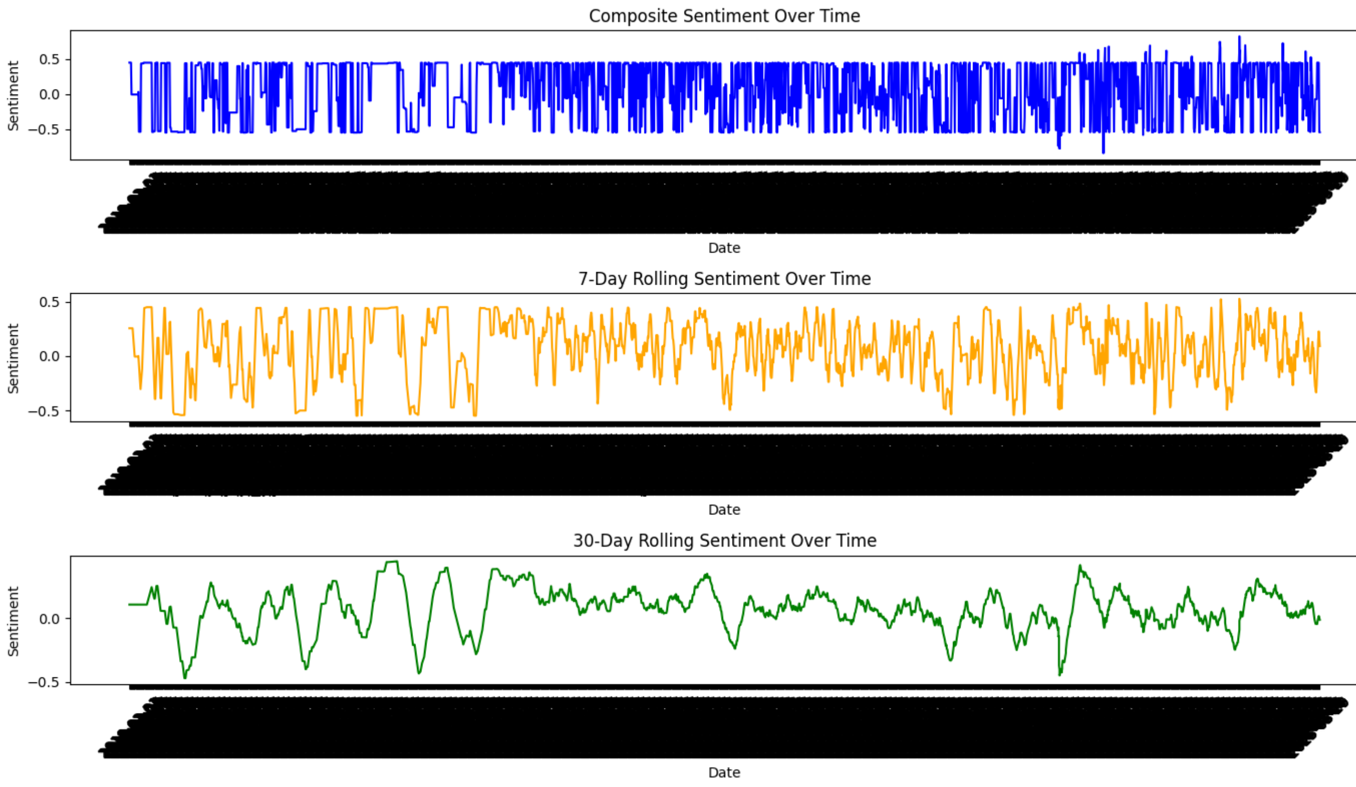




	closing_price	Volume	wikipedia_sentiment	sentiment_score_y	composite_sentiment	sentiment_7day	sentiment_30day	lag_1	30_MA	RSI_14	26_EMA	12_EMA	MACD	MACD_signal	MACD_histogram	30_STD	Bollinger_upper	Bollinger_lower
sentiment_score_y	0.05	0.10	-0.05	1.00	1.00	0.57	0.32	0.05	0.04	-0.01	0.04	0.04	0.08	0.08	0.03	0.11	0.05	0.03
composite_sentiment	0.06	0.10	0.05	1.00	1.00	0.57	0.32	0.06	0.05	-0.00	0.05	0.05	0.09	0.08	0.03	0.12	0.06	0.03
sentiment_7day	0.08	0.20	-0.05	0.57	0.57	1.00	0.60	0.09	0.08	-0.06	0.08	0.08	0.11	0.13	-0.02	0.19	0.09	0.05
sentiment_30day	0.13	0.31	-0.06	0.32	0.32	0.60	1.00	0.14	0.13	-0.16	0.13	0.13	0.10	0.13	-0.07	0.28	0.16	0.10
lag_1	1.00	0.41	0.09	0.05	0.06	0.09	0.14	1.00	0.99	-0.01	0.99	1.00	0.22	0.24	-0.04	0.67	0.99	0.98
30_MA	0.99	0.37	0.08	0.04	0.05	0.08	0.13	0.99	1.00	-0.09	1.00	1.00	0.09	0.13	-0.12	0.65	0.99	0.99
RSI_14	0.01	0.07	0.05	-0.01	-0.00	-0.06	-0.16	-0.01	-0.09	1.00	-0.08	-0.05	0.56	0.42	0.59	0.03	-0.08	-0.11
26_EMA	0.99	0.39	0.08	0.04	0.05	0.08	0.13	0.99	1.00	-0.08	1.00	1.00	0.12	0.16	-0.10	0.66	1.00	0.99
12_EMA	1.00	0.40	0.09	0.04	0.05	0.08	0.13	1.00	1.00	-0.05	1.00	1.00	0.18	0.21	-0.09	0.67	0.99	0.99
MACD	0.22	0.28	0.04	0.08	0.09	0.11	0.10	0.22	0.09	0.56	0.12	0.18	1.00	0.96	0.31	0.25	0.11	0.06
MACD_signal	0.24	0.31	0.04	0.08	0.08	0.13	0.13	0.24	0.13	0.42	0.16	0.21	0.96	1.00	0.04	0.26	0.15	0.10
MACD_histogram	-0.03	-0.05	0.01	0.03	0.03	-0.02	-0.07	-0.04	-0.12	0.59	-0.10	-0.09	0.31	0.04	1.00	0.01	-0.11	-0.14
30_STD	0.67	0.48	0.02	0.11	0.12	0.19	0.28	0.67	0.65	0.03	0.66	0.67	0.25	0.26	0.01	1.00	0.72	0.55
Bollinger_upper	0.99	0.40	0.08	0.05	0.06	0.09	0.16	0.99	0.99	-0.08	1.00	0.99	0.11	0.15	-0.11	0.72	1.00	0.97
Bollinger_lower	0.98	0.33	0.09	0.03	0.03	0.05	0.10	0.98	0.99	-0.11	0.99	0.99	0.06	0.10	-0.14	0.55	0.97	1.00

EDA on Sentiment:

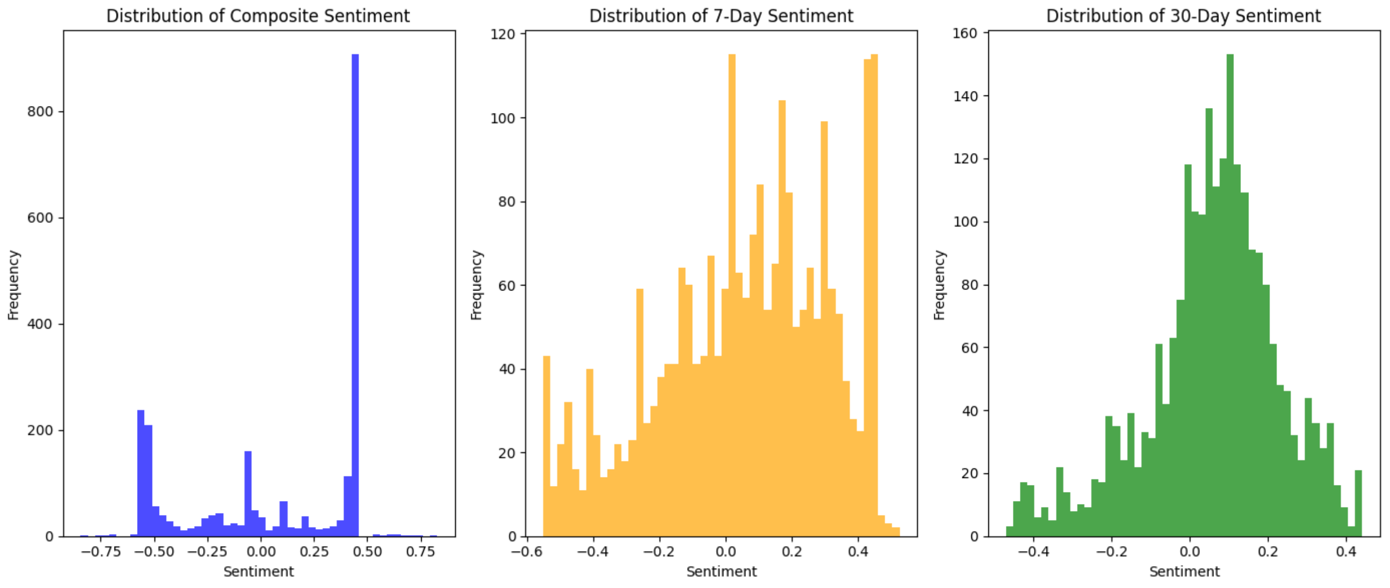
```
import matplotlib.pyplot as plt

# Plot the sentiment features over time
plt.figure(figsize=(14, 8))

# Plot composite sentiment
plt.subplot(3, 1, 1)
plt.plot(final_df['Date'], final_df['composite_sentiment'], label='Composite Sent
plt.title('Composite Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Sentiment')
plt.xticks(rotation=45)

# Plot 7-day rolling sentiment
plt.subplot(3, 1, 2)
plt.plot(final_df['Date'], final_df['sentiment_7day'], label='7-Day Rolling Senti
plt.title('7-Day Rolling Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Sentiment')
plt.xticks(rotation=45)

# Plot 30-day rolling sentiment
plt.subplot(3, 1, 3)
plt.plot(final_df['Date'], final_df['sentiment_30day'], label='30-Day Rolling Sen
plt.title('30-Day Rolling Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Sentiment')
plt.xticks(rotation=45)

# Show the plots
plt.tight_layout()
plt.show()
```


Composite Sentiment Over Time
Sentiment
0.5
0.0
−0.5
Date
7-Day Rolling Sentiment Over Time
Sentiment
0.5
0.0
−0.5
Date
30-Day Rolling Sentiment Over Time
Sentiment
0.0
−0.5
Date

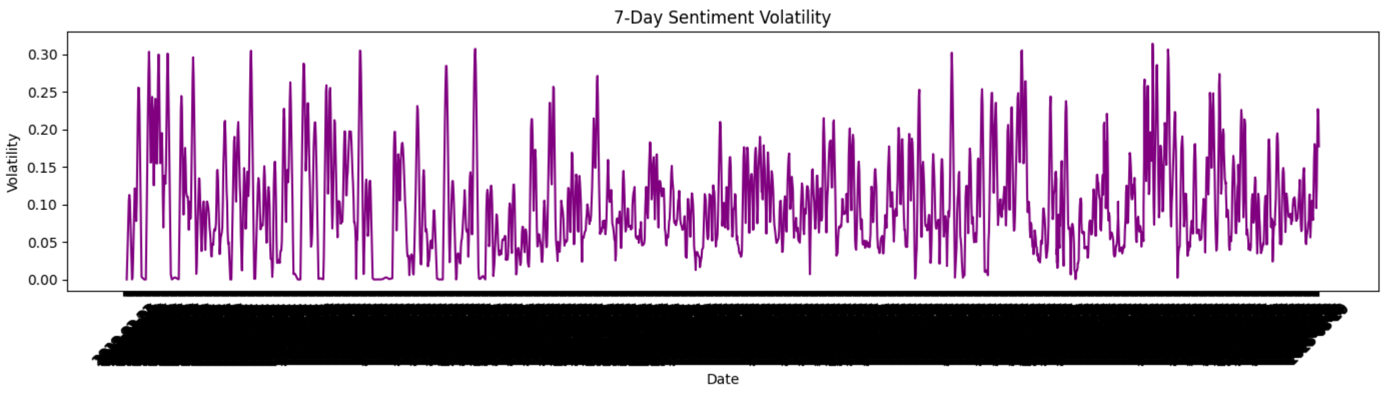
```
# Get descriptive statistics for sentiment features
sentiment_stats = final_df[['composite_sentiment', 'sentiment_7day', 'sentiment_3
print(sentiment_stats)
```

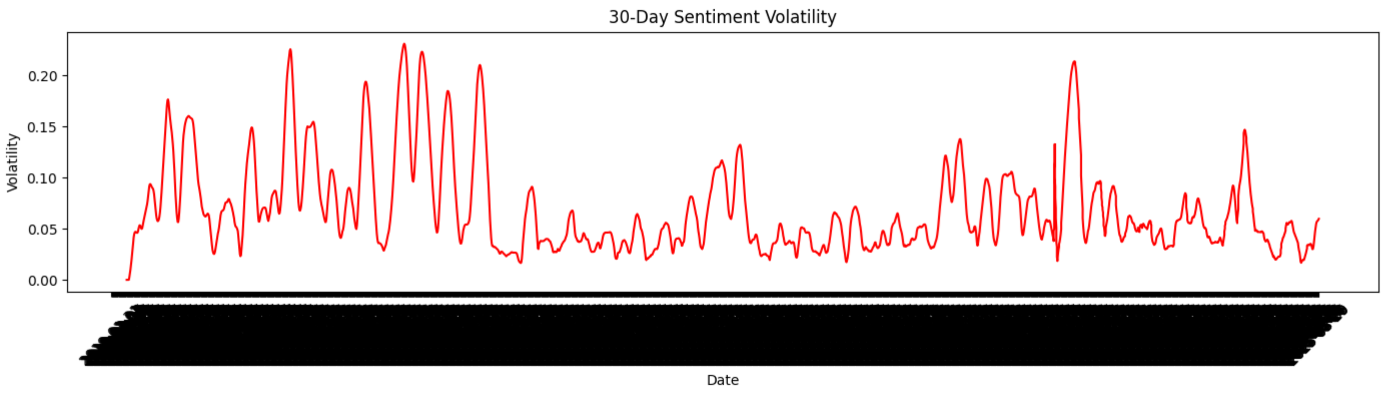
```
       composite_sentiment  sentiment_7day  sentiment_30day
count          2363.000000     2363.000000      2363.000000
mean              0.058174        0.058423         0.059158
std               0.406654        0.263109         0.171537
min              -0.844300       -0.549750        -0.470390
25%              -0.396550       -0.119870        -0.021806
50%               0.157093        0.088702         0.074875
75%               0.444000        0.267775         0.165014
max               0.825725        0.525896         0.442518
```

```
# Plot the distribution of sentiment features
plt.figure(figsize=(14, 6))

# Histogram for composite sentiment
plt.subplot(1, 3, 1)
plt.hist(final_df['composite_sentiment'], bins=50, color='blue', alpha=0.7)
plt.title('Distribution of Composite Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')

# Histogram for 7-day sentiment
plt.subplot(1, 3, 2)
plt.hist(final_df['sentiment_7day'], bins=50, color='orange', alpha=0.7)
plt.title('Distribution of 7-Day Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')

# Histogram for 30-day sentiment
plt.subplot(1, 3, 3)
plt.hist(final_df['sentiment_30day'], bins=50, color='green', alpha=0.7)
plt.title('Distribution of 30-Day Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Frequency')

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



```
# Calculate the 7-day and 30-day rolling volatility (std deviation)
final_df['sentiment_7day_volatility'] = final_df['sentiment_7day'].rolling(window
final_df['sentiment_30day_volatility'] = final_df['sentiment_30day'].rolling(wind

# Plot volatility
plt.figure(figsize=(14, 8))

# Plot 7-day sentiment volatility
plt.subplot(2, 1, 1)
plt.plot(final_df['Date'], final_df['sentiment_7day_volatility'], label='7-Day Se
plt.title('7-Day Sentiment Volatility')
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.xticks(rotation=45)

# Plot 30-day sentiment volatility
plt.subplot(2, 1, 2)
plt.plot(final_df['Date'], final_df['sentiment_30day_volatility'], label='30-Day
plt.title('30-Day Sentiment Volatility')
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.xticks(rotation=45)

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





Binary Classification

We want to predict whether the Bitcoin price will increase tomorrow (1) or decrease (0).

Prepare the target variable: We'll create a binary target variable where 1 indicates an increase in price and 0 indicates a decrease. Select features: Use the sentiment data and other technical indicators as features. Train a Binary Classification Model: We'll use models like Logistic Regression, Random Forest, or XGBoost. Evaluate the Model: We'll use metrics like accuracy, precision, recall, and F1-score to evaluate the model's performance.

```
import numpy as np
# Create a binary target variable
final_df['target'] = np.where(final_df['tomorrow_price'] > final_df['closing_pric

# Display the first few rows to verify
print(final_df[['Date', 'closing_price', 'tomorrow_price', 'target']].head())
```

```
         Date  closing_price  tomorrow_price  target
0  2019-01-31    3457.792725     3487.945312       1
1  2019-02-01    3487.945312     3521.060791       1
2  2019-02-02    3521.060791     3464.013428       0
3  2019-02-03    3464.013428     3459.154053       0
4  2019-02-04    3459.154053     3466.357422       1
```

```
# Select features for the model
features = ['composite_sentiment', 'sentiment_7day', 'sentiment_30day', 'lag_1',
X = final_df[features]
y = final_df['target']
```

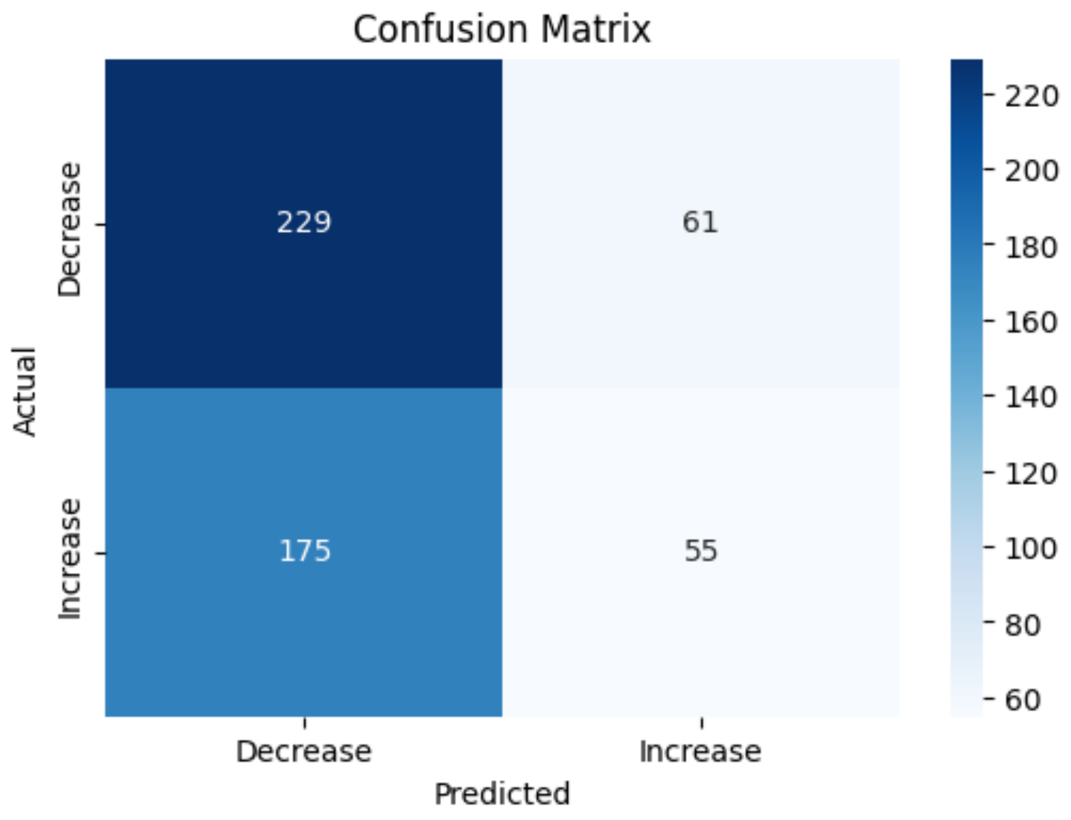
```
# Split the data into training and testing sets
split_date = '2024-01-01'  #use data before 2024-01-01 for training
train = final_df[final_df['Date'] < split_date]
test = final_df[final_df['Date'] >= split_date]

# Features and target for training and testing
X_train = train[features]
y_train = train['target']
X_test = test[features]
y_test = test['target']
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sco

# Initialize the RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Calculate the performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

```
Accuracy: 0.55
Precision: 0.47
Recall: 0.24
F1-Score: 0.32
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', xticklabels=['Decrease', 'Incr
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



```
# Get feature importance from the trained model
importances = model.feature_importances_

# Create a DataFrame to display feature importances
feature_importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print(feature_importance_df)
```

```
             Feature  Importance
5             RSI_14    0.125222
1      sentiment_7day   0.121567
8      MACD_histogram   0.115214
2     sentiment_30day   0.113600
3              lag_1    0.113498
6               MACD    0.107957
4              30_MA    0.107832
7        MACD_signal    0.103865
0  composite_sentiment  0.091245
```

Hyperparameter Tuning

```
# Make copies to avoid SettingWithCopyWarning
X_train = X_train.copy()
X_test = X_test.copy()

# Recommended way to fill missing values in modern pandas
# For forward fill (ffill)
X_train['sentiment_7day'] = X_train['sentiment_7day'].ffill()
X_train['sentiment_30day'] = X_train['sentiment_30day'].ffill()

# For filling remaining NaN with 0 (without inplace)
X_train = X_train.assign(
    sentiment_7day=X_train['sentiment_7day'].fillna(0),
    sentiment_30day=X_train['sentiment_30day'].fillna(0)
)

# Apply the same for X_test
X_test['sentiment_7day'] = X_test['sentiment_7day'].ffill()
X_test['sentiment_30day'] = X_test['sentiment_30day'].ffill()

X_test = X_test.assign(
    sentiment_7day=X_test['sentiment_7day'].fillna(0),
    sentiment_30day=X_test['sentiment_30day'].fillna(0)
)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Initialize Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model
rf.fit(X_train[features], y_train)

# Make predictions
y_pred_rf = rf.predict(X_test[features])

# Evaluate the model
print("Classification Report (Random Forest):\n", classification_report(y_test, y
print("Confusion Matrix (Random Forest):\n", confusion_matrix(y_test, y_pred_rf))
```

```
Classification Report (Random Forest):
               precision    recall  f1-score   support

           0       0.57      0.79      0.66       290
           1       0.47      0.24      0.32       230

    accuracy                           0.55       520
   macro avg       0.52      0.51      0.49       520
weighted avg       0.53      0.55      0.51       520

Confusion Matrix (Random Forest):
 [[229  61]
 [175  55]]
```

Fix Class Imbalance

```
import xgboost as xgb
from sklearn.metrics import classification_report, confusion_matrix

# Calculate class weights based on class frequency in your training data
from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y
# Convert class weights to a dictionary (for XGBoost)
class_weights_dict = {i: weight for i, weight in enumerate(class_weights)}
# Train an XGBoost model with class weights
xgb_model = xgb.XGBClassifier(scale_pos_weight=class_weights_dict[1], random_stat
xgb_model.fit(X_train, y_train)

# Evaluate the model
y_pred_xgb = xgb_model.predict(X_test)

# Print the classification report
print(classification_report(y_test, y_pred_xgb))

# Confusion Matrix
print(confusion_matrix(y_test, y_pred_xgb))
```

```
              precision    recall  f1-score   support

           0       0.56      0.70      0.62       290
           1       0.45      0.32      0.37       230

    accuracy                           0.53       520
   macro avg       0.51      0.51      0.50       520
weighted avg       0.51      0.53      0.51       520

[[202  88]
 [157  73]]
```

Key Insights: The model is biasing toward predicting price decreases (Class 0). This is seen in the relatively high recall for Class 0 (70%) but poor recall for Class 1 (32%). The model's precision for Class 1 (price increases) is quite low (45%), meaning when it predicts a price increase, it's incorrect half the time. Recall for Class 1 is much worse at 32%, indicating that the model is missing a lot of price increases. Accuracy is not a great measure here due to class imbalance; instead, precision, recall, and F1-scores are better indicators of performance, especially for Class 1.

Handle Class Imbalance with SMOTE and Class Weight

```
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Handle missing values by forward filling (you can also try backward filling or
X_train.fillna(method='ffill', inplace=True)
X_test.fillna(method='ffill', inplace=True)

# Alternatively, fill remaining NaNs with 0 (if needed)
X_train.fillna(0, inplace=True)
X_test.fillna(0, inplace=True)

# Apply SMOTE for balancing the classes
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data (X_train, y_train)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Initialize Random Forest model with class weight to handle class imbalance
rf_model_smote = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on balanced data
rf_model_smote.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred_smote = rf_model_smote.predict(X_test)

# Evaluate the model
print("Classification Report with SMOTE:")
print(classification_report(y_test, y_pred_smote))
print("Confusion Matrix with SMOTE:")
print(confusion_matrix(y_test, y_pred_smote))
```

```
<ipython-input-109-8e3f80a8457d>:7: FutureWarning: DataFrame.fillna with 'meth
  X_train.fillna(method='ffill', inplace=True)
<ipython-input-109-8e3f80a8457d>:8: FutureWarning: DataFrame.fillna with 'meth
  X_test.fillna(method='ffill', inplace=True)
Classification Report with SMOTE:
              precision    recall  f1-score   support

           0       0.56      0.69      0.62       290
           1       0.44      0.31      0.36       230

    accuracy                           0.52       520
   macro avg       0.50      0.50      0.49       520
weighted avg       0.51      0.52      0.50       520

Confusion Matrix with SMOTE:
[[200  90]
 [159  71]]
```

Key Insights: Class Imbalance: The model still heavily favors predicting Class 0 (Price Decreases). The recall for Class 0 is good (69%), but the recall for Class 1 (Price Increases) is much lower (31%). Need for Improvement in Predicting Price Increases: The model needs to improve its ability to predict price increases, as evidenced by the low precision and recall for Class 1.

Add New Features

```
# Add Price Change Percentage feature
final_df['price_change_percentage'] = (final_df['closing_price'] - final_df['lag_

# Add Price Volatility (7-day and 30-day rolling standard deviation)
final_df['price_volatility_7day'] = final_df['closing_price'].rolling(window=7).s
final_df['price_volatility_30day'] = final_df['closing_price'].rolling(window=30)

# Add Sentiment Momentum feature
final_df['sentiment_momentum'] = final_df['sentiment_7day'] - final_df['sentiment

# Add Interaction Features (Sentiment * RSI_14, Sentiment * MACD)
final_df['sentiment_7day_x_RSI_14'] = final_df['sentiment_7day'] * final_df['RSI_
final_df['sentiment_30day_x_MACD'] = final_df['sentiment_30day'] * final_df['MACD

# Fill missing values in the newly created features (forward fill and zero where
final_df['price_volatility_7day'] = final_df['price_volatility_7day'].ffill().fil
final_df['price_volatility_30day'] = final_df['price_volatility_30day'].ffill().f
final_df['sentiment_momentum'] = final_df['sentiment_momentum'].ffill().fillna(0)
final_df['sentiment_7day_x_RSI_14'] = final_df['sentiment_7day_x_RSI_14'].ffill()
final_df['sentiment_30day_x_MACD'] = final_df['sentiment_30day_x_MACD'].ffill().f

# Save the updated dataset with the new features
final_df.to_csv('/content/drive/MyDrive/Final_Project_Docs/bitcoin_with_new_featu

print("New features added successfully!")
```

```
New features added successfully!
```

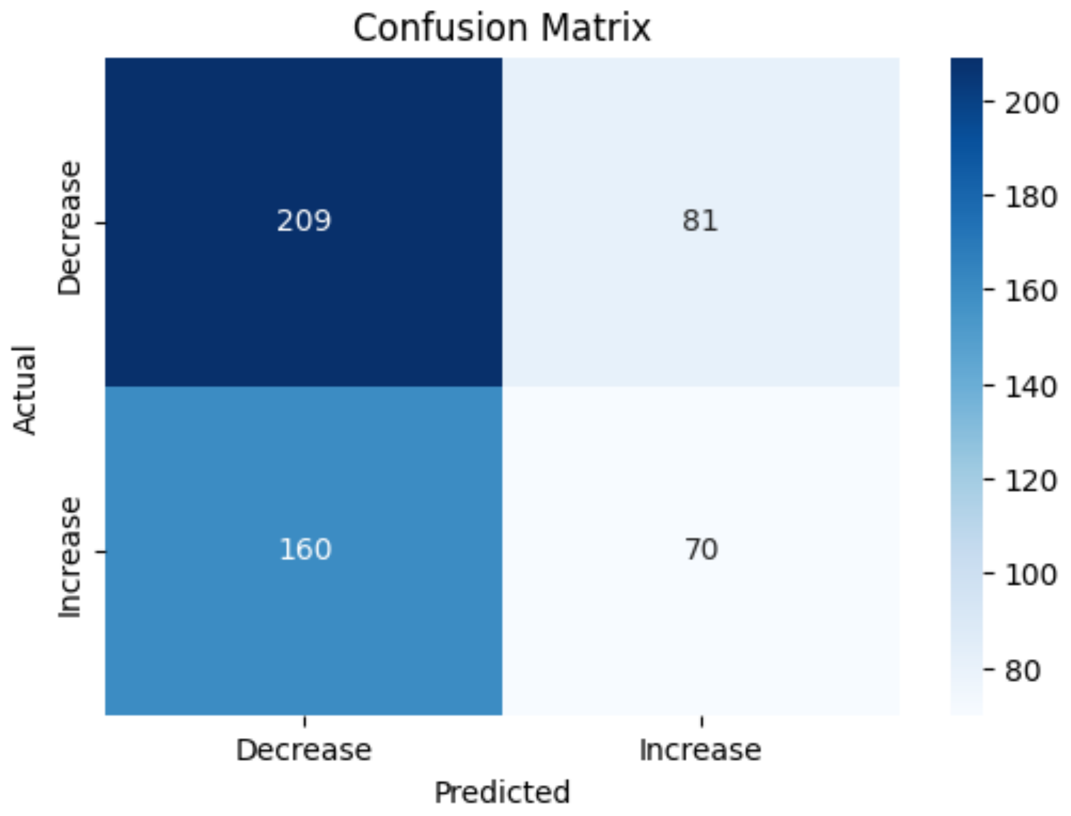
```
# Create target variable for classification
final_df['target'] = np.where(final_df['tomorrow_price'] > final_df['closing_pric

# Select features for the model
features = ['composite_sentiment', 'sentiment_7day', 'sentiment_30day', 'price_ch
            'price_volatility_7day', 'price_volatility_30day', 'sentiment_momentu
            'sentiment_7day_x_RSI_14', 'sentiment_30day_x_MACD', 'lag_1', '30_MA'
            'MACD', 'MACD_signal', 'MACD_histogram']

# Define features and target
X = final_df[features]
y = final_df['target']

# Split data into training and testing sets
split_date = '2024-01-01'  # Use data before 2024-01-01 for training
train = final_df[final_df['Date'] < split_date]
test = final_df[final_df['Date'] >= split_date]

# Features and target for training and testing
X_train = train[features]
y_train = train['target']
X_test = test[features]
y_test = test['target']



from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Initialize RandomForest model
model = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

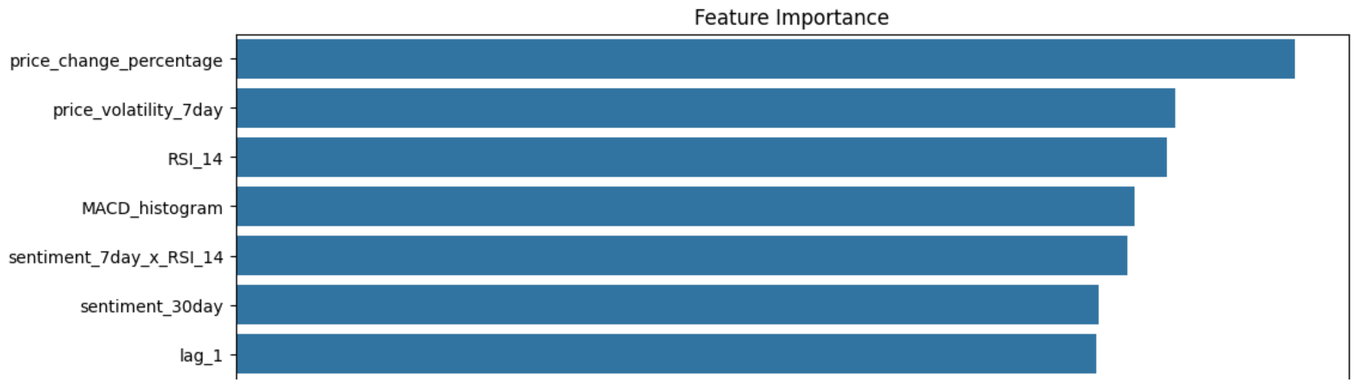
```
# Get feature importance
importances = model.feature_importances_

# Create a DataFrame for feature importances
feature_importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

print(feature_importance_df)
```

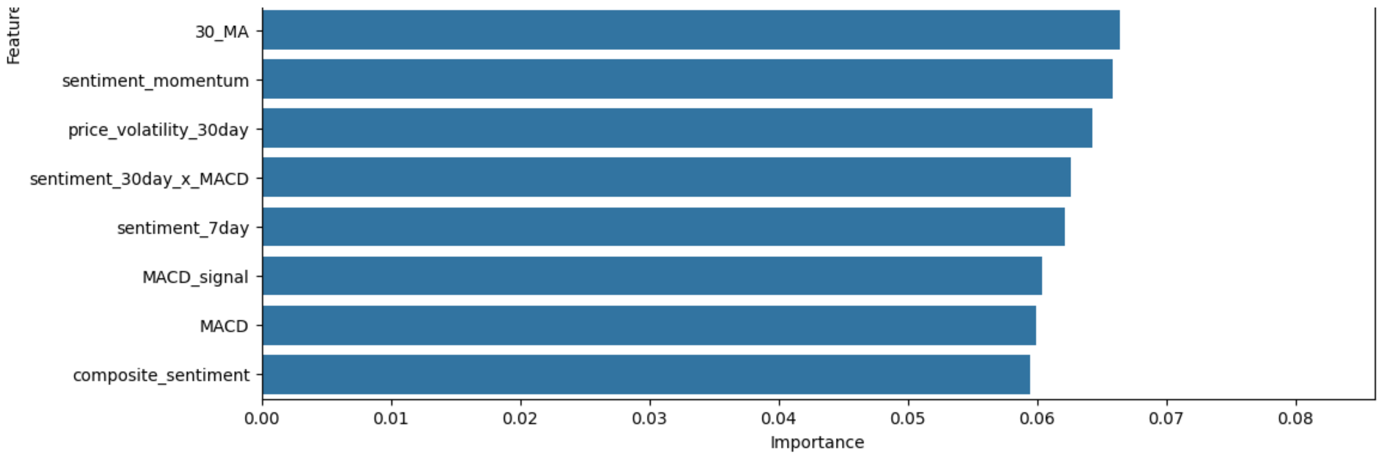
```
Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.72      0.63       290
           1       0.46      0.30      0.37       230

    accuracy                           0.54       520
   macro avg       0.51      0.51      0.50       520
weighted avg       0.52      0.54      0.52       520

Confusion Matrix:
[[209  81]
 [160  70]]
                  Feature  Importance
3   price_change_percentage    0.082000
4     price_volatility_7day    0.072725
11                   RSI_14    0.072054
14           MACD_histogram    0.069588
7   sentiment_7day_x_RSI_14    0.069031
2          sentiment_30day    0.066802
9                     lag_1    0.066608
10                    30_MA    0.066442
6        sentiment_momentum    0.065890
5    price_volatility_30day    0.064250
8    sentiment_30day_x_MACD    0.062629
1           sentiment_7day    0.062149
13              MACD_signal    0.060440
12                     MACD    0.059919
0       composite_sentiment    0.059473
```

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

# Plot confusion matrix
```

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', xticklabels=['Decrease', 'Incr
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Plot feature importance
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)
plt.title('Feature Importance')
plt.show()
```



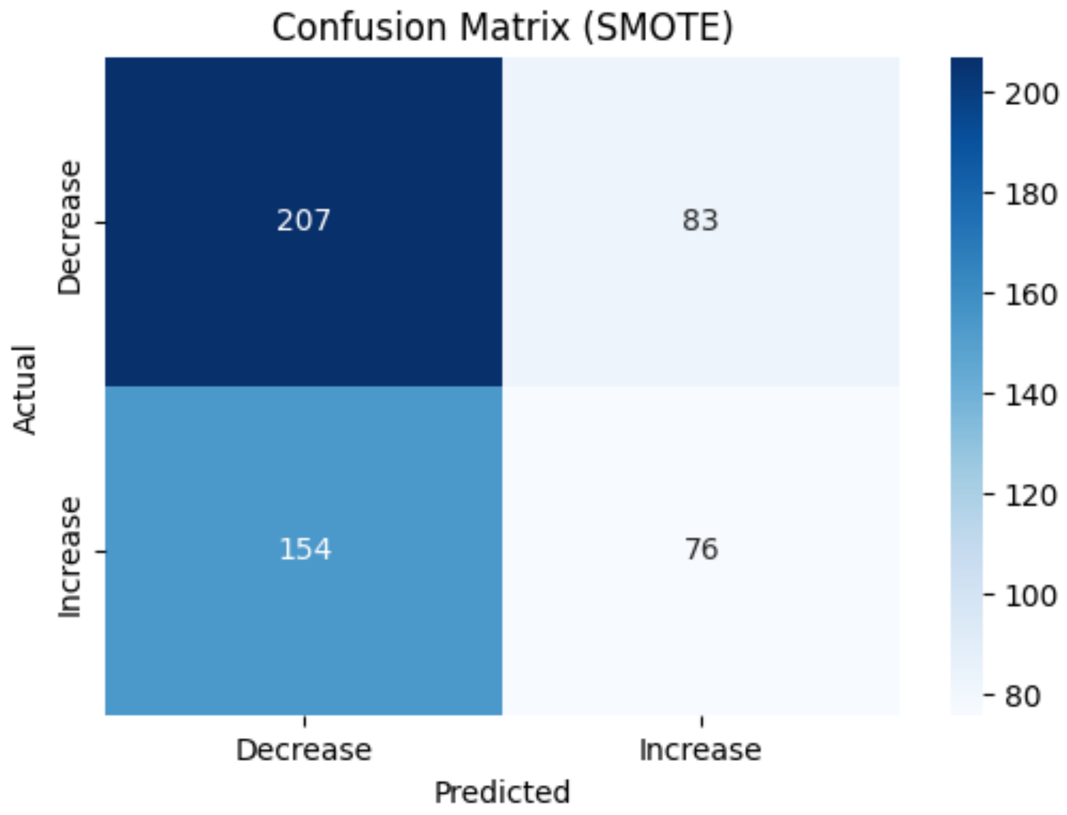




```
# Fill missing values in the training data (X_train) and test data (X_test)
X_train = X_train.fillna(method='ffill')  # Forward fill for X_train
X_train = X_train.fillna(0)  # If any NaNs still exist, fill with 0
X_test = X_test.fillna(method='ffill')  # Forward fill for X_test
X_test = X_test.fillna(0)  # If any NaNs still exist, fill with 0

# Now proceed with applying SMOTE
from imblearn.over_sampling import SMOTE

# Apply SMOTE for balancing the classes
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Check if any NaNs exist after SMOTE
print(f"Missing values in X_train_smote: {X_train_smote.isnull().sum().sum()}")
print(f"Missing values in y_train_smote: {y_train_smote.isnull().sum()}")
```

```
# Proceed with training the model using RandomForest
from sklearn.ensemble import RandomForestClassifier

# Train the Random Forest model with class weights on the balanced data
rf_model_smote = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model_smote.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred_smote = rf_model_smote.predict(X_test)

# Evaluate the model
from sklearn.metrics import classification_report, confusion_matrix

print("Classification Report with SMOTE:")
print(classification_report(y_test, y_pred_smote))
print("Confusion Matrix with SMOTE:")
print(confusion_matrix(y_test, y_pred_smote))

# Plot the confusion matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred_smote)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', xticklabels=['Decrease', 'Incr
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (SMOTE)')
plt.show()
```

```
<ipython-input-114-b4d962de4c79>:2: FutureWarning: DataFrame.fillna with 'meth
  X_train = X_train.fillna(method='ffill')  # Forward fill for X_train
<ipython-input-114-b4d962de4c79>:4: FutureWarning: DataFrame.fillna with 'meth
  X_test = X_test.fillna(method='ffill')  # Forward fill for X_test
Missing values in X_train_smote: 0
Missing values in y_train_smote: 0
Classification Report with SMOTE:
              precision    recall  f1-score   support

           0       0.57      0.71      0.64       290
           1       0.48      0.33      0.39       230

    accuracy                           0.54       520
   macro avg       0.53      0.52      0.51       520
weighted avg       0.53      0.54      0.53       520

Confusion Matrix with SMOTE:
[[207  83]
 [154  76]]
```



```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'class_weight': ['balanced', None]  # Trying to balance class weights
}

grid_search = GridSearchCV(estimator=rf_model_smote, param_grid=param_grid, cv=5,
grid_search.fit(X_train_smote, y_train_smote)

print(f"Best Hyperparameters: {grid_search.best_params_}")
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'class_weight': 'balanced', 'max_depth': 10, 'min_samp
```

```
# Initialize the RandomForestClassifier with the best hyperparameters
best_rf_model = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_split=2,
    min_samples_leaf=4,
    class_weight='balanced',
    random_state=42
)

# Train the model on the balanced data (X_train_smote, y_train_smote)
best_rf_model.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred_best_rf = best_rf_model.predict(X_test)

# Evaluate the model
from sklearn.metrics import classification_report, confusion_matrix

print("Classification Report (Best Random Forest Model):")
print(classification_report(y_test, y_pred_best_rf))

print("Confusion Matrix (Best Random Forest Model):")
print(confusion_matrix(y_test, y_pred_best_rf))
```

```
Classification Report (Best Random Forest Model):
              precision    recall  f1-score   support

           0       0.58      0.81      0.68       290
           1       0.52      0.26      0.34       230

    accuracy                           0.57       520
   macro avg       0.55      0.53      0.51       520
weighted avg       0.55      0.57      0.53       520

Confusion Matrix (Best Random Forest Model):
[[235  55]
 [171  59]]
```

The model performs reasonably well for class 0, with a good recall of 0.78, but struggles with class 1, where the recall is only 0.26. This indicates that the model is biased towards predicting class 0 more accurately. The confusion matrix further illustrates this, showing a significant number of false negatives for class 1.

Address Class Imbalance

```
from imblearn.over_sampling import SMOTE

# Apply SMOTE to balance the classes in the training set
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Now we can re-train the model with this balanced dataset
```

```
# Train Random Forest model with class_weight set to 'balanced'
rf_model = RandomForestClassifier(n_estimators=100, class_weight='balanced', rand
rf_model.fit(X_train_smote, y_train_smote)
```

```
RandomForestClassifier

RandomForestClassifier(class_weight='balanced', random_state=42)
```

```
# Predict probabilities instead of classes
y_pred_prob = rf_model.predict_proba(X_test)[:, 1]

# Adjust threshold (e.g., if probability > 0.4, predict Class 1)
threshold = 0.4
y_pred_adjusted = (y_pred_prob > threshold).astype(int)

# Evaluate the adjusted predictions
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, y_pred_adjusted))
print(confusion_matrix(y_test, y_pred_adjusted))
```

```
              precision    recall  f1-score   support

           0       0.54      0.31      0.40       290
           1       0.44      0.67      0.53       230

    accuracy                           0.47       520
   macro avg       0.49      0.49      0.46       520
weighted avg       0.50      0.47      0.46       520

[[ 91 199]
 [ 76 154]]
```

Regression

In regression, the target variable can be:

Percentage Change in Price: The percentage change between today's closing price and tomorrow's closing price. Tomorrow’s Closing Price: The predicted price of Bitcoin for the next day.

```
# Calculate the percentage change between closing price and the next day's closin
final_df['price_change_pct'] = (final_df['tomorrow_price'] - final_df['closing_pr

# Inspect the new target
print(final_df[['Date', 'closing_price', 'tomorrow_price', 'price_change_pct']].h
```

```
         Date  closing_price  tomorrow_price  price_change_pct
0  2019-01-31    3457.792725     3487.945312          0.872018
1  2019-02-01    3487.945312     3521.060791          0.949427
2  2019-02-02    3521.060791     3464.013428         -1.620175
3  2019-02-03    3464.013428     3459.154053         -0.140282
4  2019-02-04    3459.154053     3466.357422          0.208241
```

```
features = ['composite_sentiment', 'sentiment_7day', 'sentiment_30day', 'price_ch
            'price_volatility_7day', 'price_volatility_30day', 'sentiment_momentu
            'sentiment_7day_x_RSI_14', 'sentiment_30day_x_MACD', 'lag_1', '30_MA'
            'MACD', 'MACD_signal', 'MACD_histogram']
X = final_df[features]
y = final_df['price_change_pct']
```

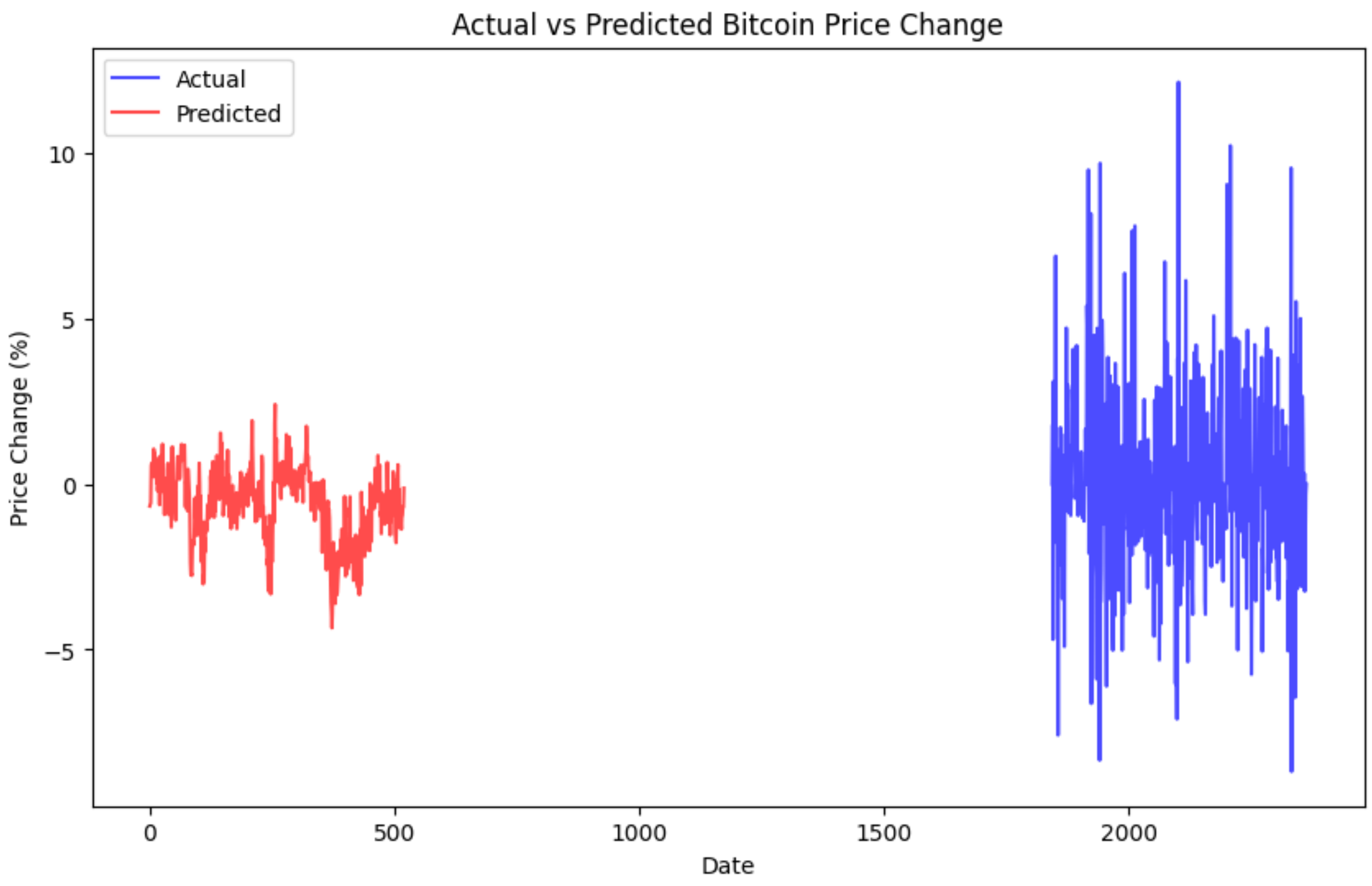
```
# Split data into training and testing sets
split_date = '2024-01-01'  # Use data before 2024-01-01 for training
train = final_df[final_df['Date'] < split_date]
test = final_df[final_df['Date'] >= split_date]

# Features and target for training and testing
X_train = train[features]
y_train = train['price_change_pct']
X_test = test[features]
y_test = test['price_change_pct']
```

```
from sklearn.ensemble import RandomForestRegressor

# Initialize the RandomForestRegressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf_regressor.fit(X_train, y_train)

# Make predictions on the test set
y_pred_reg = rf_regressor.predict(X_test)
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Calculate the performance metrics
mae = mean_absolute_error(y_test, y_pred_reg)
mse = mean_squared_error(y_test, y_pred_reg)
# Calculate RMSE using NumPy to ensure compatibility
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred_reg)

# Print the results
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
Mean Absolute Error (MAE): 2.10
Mean Squared Error (MSE): 8.30
Root Mean Squared Error (RMSE): 2.88
R-squared: -0.23
```

```
import matplotlib.pyplot as plt

# Plot the actual vs predicted values
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Actual', color='blue', alpha=0.7)
plt.plot(y_pred_reg, label='Predicted', color='red', alpha=0.7)
plt.title('Actual vs Predicted Bitcoin Price Change')
plt.xlabel('Date')
plt.ylabel('Price Change (%)')
plt.legend()
plt.show()
```



```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for Random Forest Regressor
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform GridSearchCV
grid_search = GridSearchCV(estimator=rf_regressor, param_grid=param_grid, cv=5, n
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(f"Best Hyperparameters: {grid_search.best_params_}")

# Train the best model
best_rf_regressor = grid_search.best_estimator_

# Make predictions with the best model
y_pred_best = best_rf_regressor.predict(X_test)

# Calculate the performance metrics for the best model
mae_best = mean_absolute_error(y_test, y_pred_best)
mse_best = mean_squared_error(y_test, y_pred_best)
rmse_best = mse_best ** 0.5  # Calculate RMSE manually
r2_best = r2_score(y_test, y_pred_best)

# Print the results of the best model
print(f"Best Model MAE: {mae_best:.2f}")
print(f"Best Model MSE: {mse_best:.2f}")
print(f"Best Model RMSE: {rmse_best:.2f}")
print(f"Best Model R-squared: {r2_best:.2f}")
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_s
Best Model MAE: 1.96
Best Model MSE: 7.54
Best Model RMSE: 2.75
Best Model R-squared: -0.11
```

```
Start coding or generate with AI.
```