SECOND PROJECT

GOLD PRICE PREDICTION

ANIKET KUMAR

MENTOR NESS

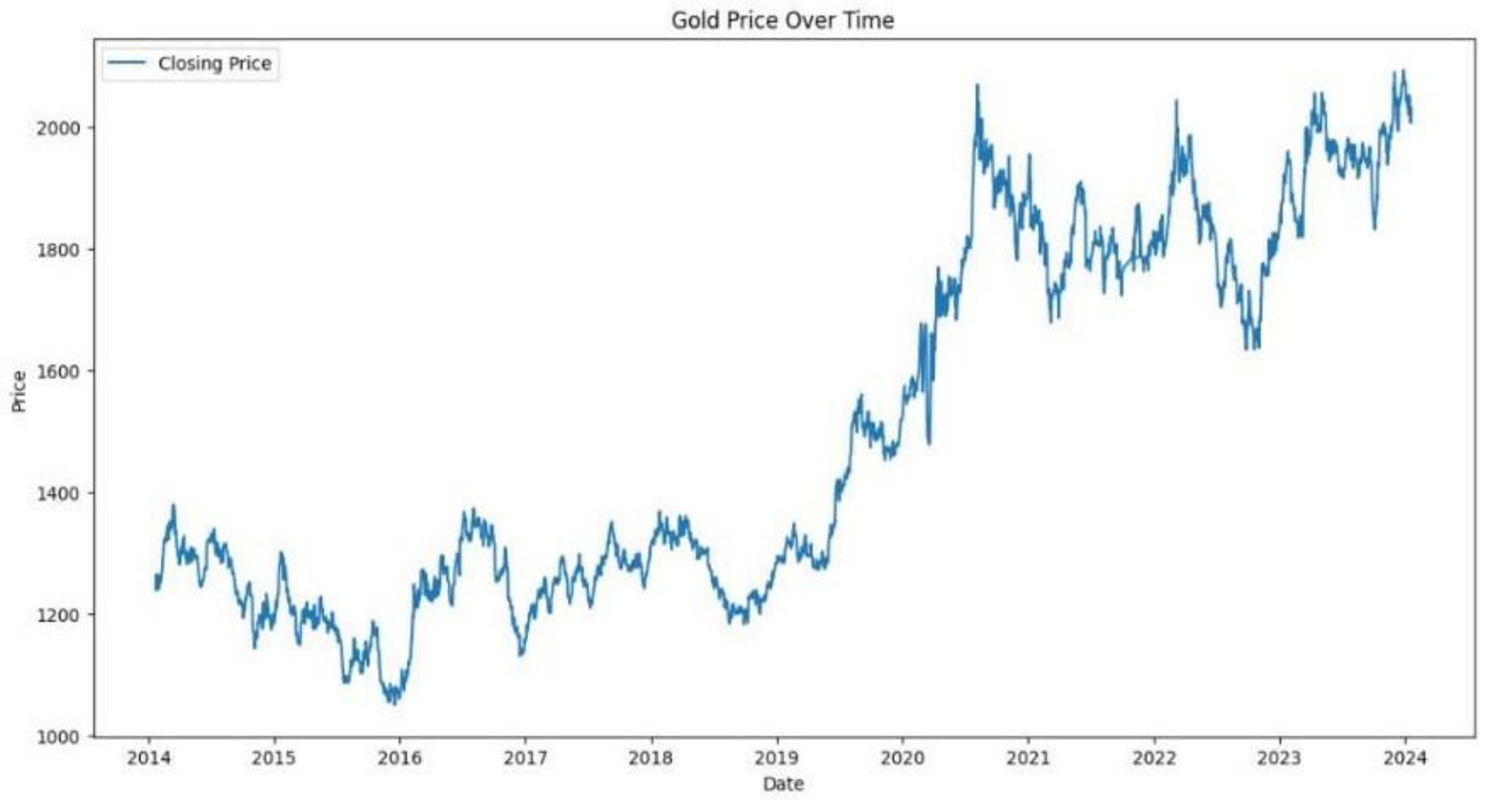
Problem Statement

This project aims to leverage a comprehensive dataset of daily gold prices spanning from January 19, 2014, to January 22, 2024, obtained from Nasdaq. The dataset encompasses key financial metrics for each trading day, including the opening and closing prices, trading volume, as well as the highest and lowest prices recorded during the day.

Dataset Description

1. Date: A unique identifier for each trading day.
2. Close: Closing price of gold on the respective date.
3. Volume: Gold trading volume on the corresponding date.
4. Open: Opening price of gold on the respective date.
5. High: The highest recorded price of gold during the trading day.
6. Low: The lowest price recorded for gold in the trading day.

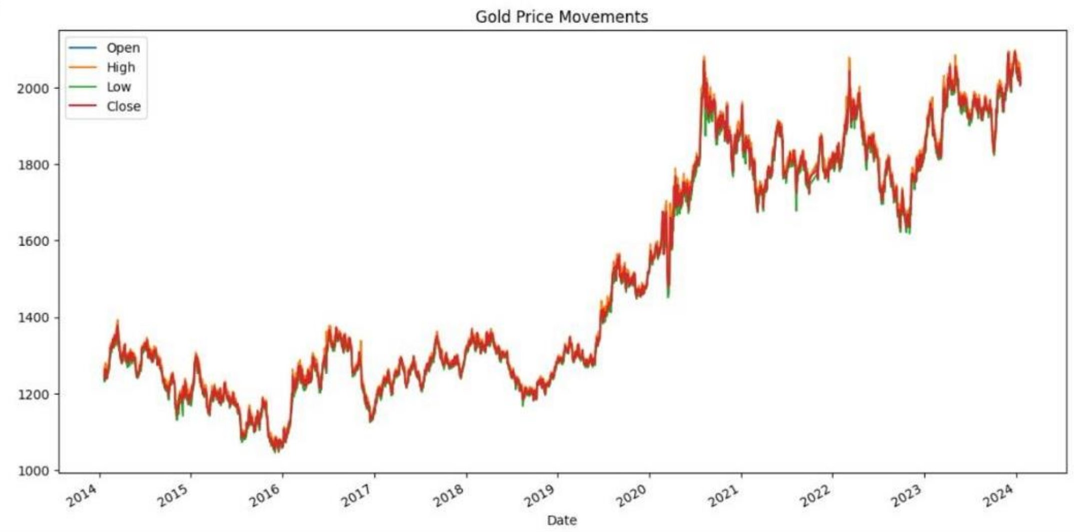
```
# Time series visualization
plt.figure(figsize=(14, 7))
plt.plot(df['Close'], label='Closing Price')
plt.title('Gold Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

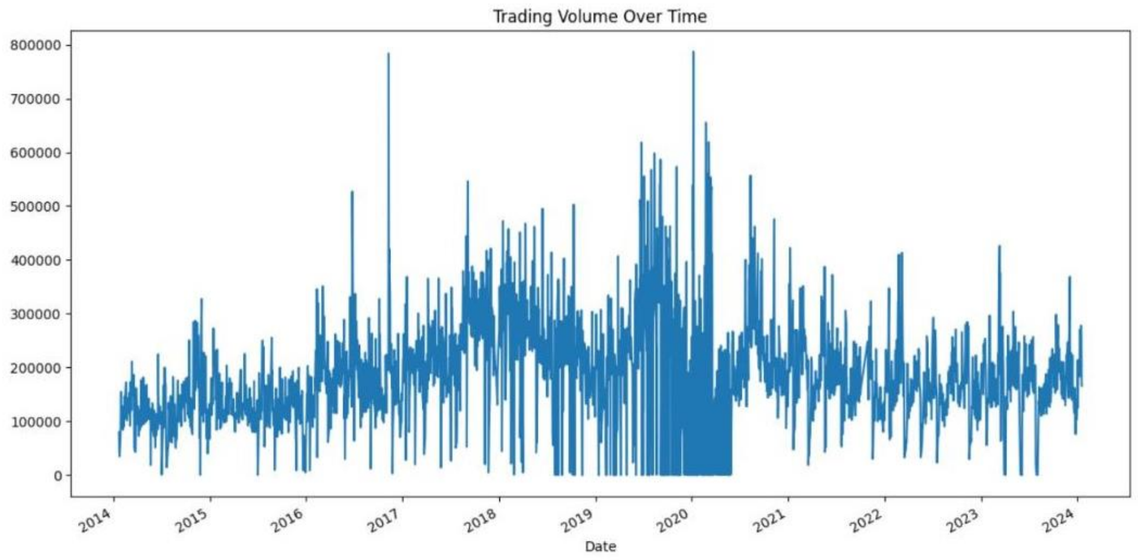


MENTOR NESS

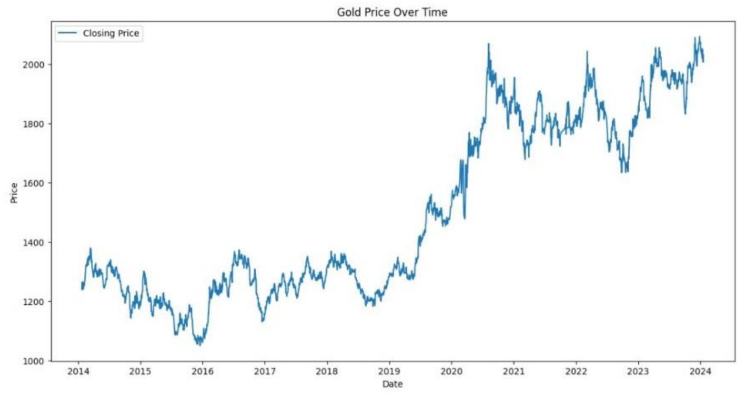
Time Series Analysis

```
# Visualize opening, high, low, and closing prices
df[['Open', 'High', 'Low', 'Close']].plot(figsize=(14, 7))
plt.title('Gold Price Movements')
plt.show()
```



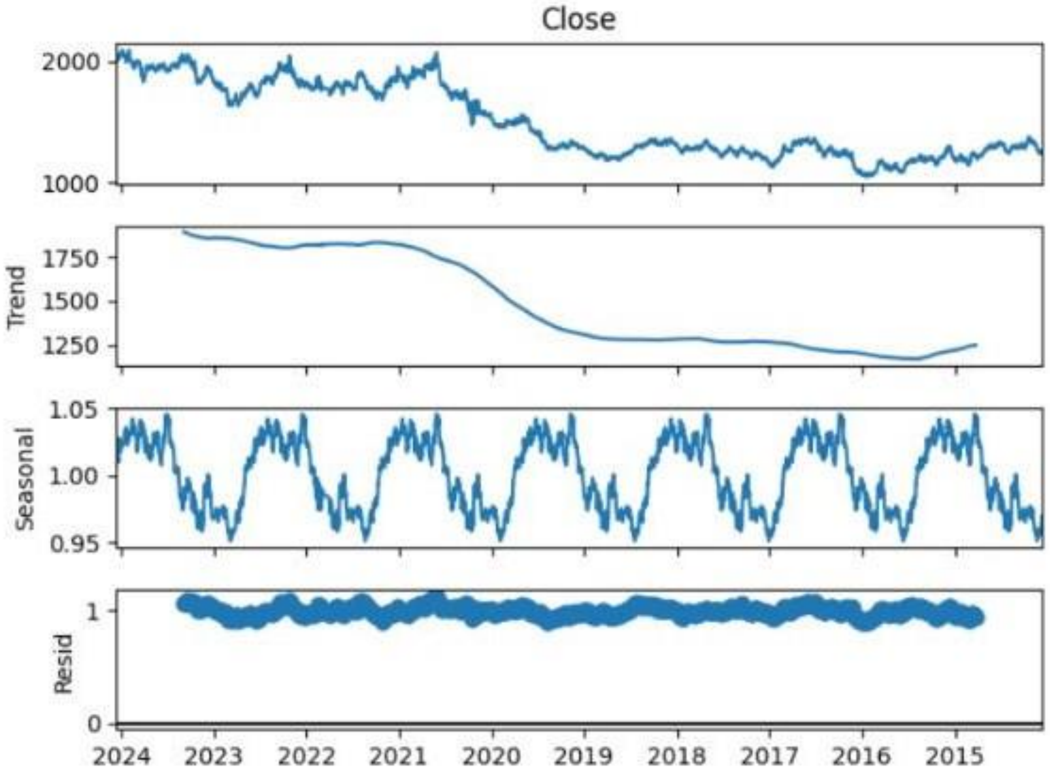


```python
# Time series visualization
plt.figure(figsize=(14, 7))
plt.plot(df['Close'], label='Closing Price')
plt.title('Gold Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```



Identify Seasonality, Cyclicality, and Long-Term Trends

```python
from statsmodels.tsa.seasonal import seasonal_decompose
# Decompose the time series
result = seasonal_decompose(df['Close'], model='multiplicative', period=365)
result.plot()
plt.show()
```



ARIMA Model Development

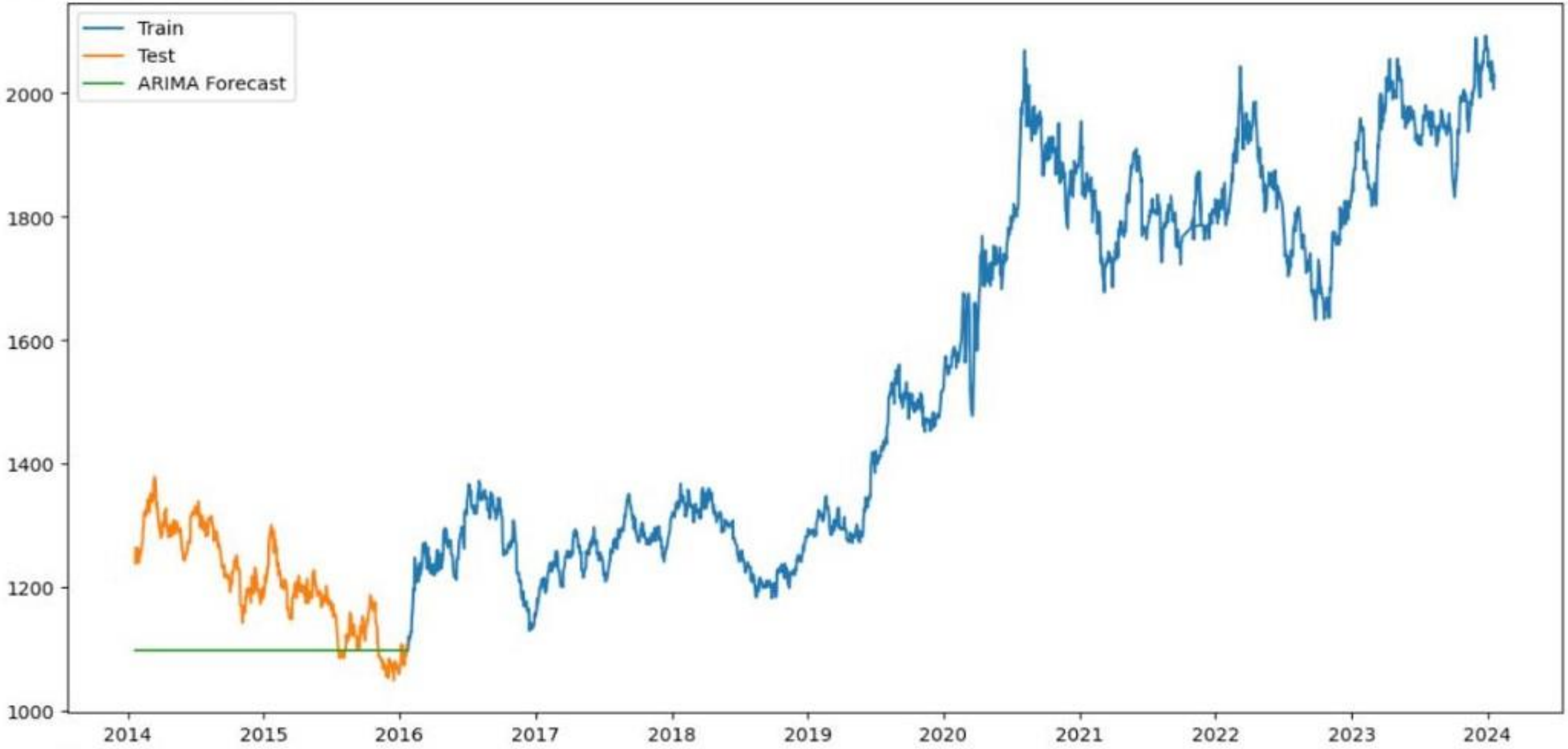
```
[ ] # Train-test split for ARIMA
    train_size = int(len(df) * 0.8)
    train, test = df['Close'][:train_size], df['Close'][train_size:]
```

Fit and Forecast Using ARIMA

```
[ ] from statsmodels.tsa.arima.model import ARIMA
    from sklearn.metrics import mean_squared_error

    # Fit ARIMA model
    arima_model = ARIMA(train, order=(5, 1, 0))
    arima_fit = arima_model.fit()

    # Forecast
    arima_forecast = arima_fit.forecast(steps=len(test))

    # Plot the results
    plt.figure(figsize=(14, 7))
    plt.plot(train, label='Train')
    plt.plot(test, label='Test')
    plt.plot(test.index, arima_forecast, label='ARIMA Forecast')
    plt.legend()
    plt.show()

    # Evaluate model
    arima_rmse = mean_squared_error(test, arima_forecast, squared=False)
    print(f'ARIMA RMSE: {arima_rmse}')
```

MENTOR NESS

Advanced Modeling




```
ARIMA RMSE: 135.8767871226694
```

LSTM Model for Prediction

```
# Prepare data for LSTM
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(df['Close'].values.reshape(-1, 1))

train_scaled = scaled_data[:train_size]
test_scaled = scaled_data[train_size:]

# Prepare the data for LSTM model
time_step = 100
X_train, Y_train = [], []
for i in range(time_step, len(train_scaled)):
    X_train.append(train_scaled[i - time_step:i, 0])
    Y_train.append(train_scaled[i, 0])
X_train, Y_train = np.array(X_train), np.array(Y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

X_test, Y_test = [], []
for i in range(time_step, len(test_scaled)):
    X_test.append(test_scaled[i - time_step:i, 0])
    Y_test.append(test_scaled[i, 0])
X_test, Y_test = np.array(X_test), np.array(Y_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Build LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(50, return_sequences=True, input_shape=(time_step, 1)))
lstm_model.add(LSTM(50, return_sequences=False))
lstm_model.add(Dense(1))
lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Train LSTM model
lstm_model.fit(X_train, Y_train, epochs=50, batch_size=64, verbose=1)

# Predict with LSTM model
train_predict = lstm_model.predict(X_train)
test_predict = lstm_model.predict(X_test)

# Invert predictions
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
Y_train = scaler.inverse_transform([Y_train])
Y_test = scaler.inverse_transform([Y_test])
```

LSTM RMSE: 15.725709067534996

MENTOR NESS

Moving Average Strategy

```
# Moving Average Strategy
df['SMA50'] = df['Close'].rolling(window=50).mean()
df['SMA200'] = df['Close'].rolling(window=200).mean()

# Trading signals
df['Signal'] = 0
df['Signal'][50:] = np.where(df['SMA50'][50:] > df['SMA200'][50:], 1, 0)
df['Position'] = df['Signal'].diff()

# Plot signals
plt.figure(figsize=(14, 7))
plt.plot(df['Close'], label='Close Price')
plt.plot(df['SMA50'], label='50-Day SMA')
plt.plot(df['SMA200'], label='200-Day SMA')
plt.plot(df[df['Position'] == 1].index, df['SMA50'][df['Position'] == 1], '^', markersize=10, color='g', lw=0, label='Buy Signal')
plt.plot(df[df['Position'] == -1].index, df['SMA50'][df['Position'] == -1], 'v', markersize=10, color='r', lw=0, label='Sell Signal')
plt.legend()
plt.show()
```


Close Price
50-Day SMA
200-Day SMA
Buy Signal
Sell Signal
2000
1800
1600
1400
1200
1000
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
MENTOR NESS

Market Events and Sentiment Analysis

```
# Simulate market events data
events_data = {
    'Date': pd.to_datetime(['2020-01-15', '2020-03-11', '2020-06-30', '2020-09-21', '2020-12-15']),
    'Event': [
        'Trade Deal Signed',
        'COVID-19 Declared Pandemic',
        'Economic Stimulus Announced',
        'Stock Market Crash',
        'Vaccine Approval'
    ]
}

events_df = pd.DataFrame(events_data)

# Simulate sentiment scores (random for illustration purposes)
np.random.seed(42)
events_df['Sentiment_Score'] = np.random.uniform(-1, 1, events_df.shape[0])

print(events_df)
```

```
        Date                        Event  Sentiment_Score
0 2020-01-15            Trade Deal Signed        -0.250920
1 2020-03-11   COVID-19 Declared Pandemic         0.901429
2 2020-06-30  Economic Stimulus Announced         0.463988
3 2020-09-21           Stock Market Crash         0.197317
4 2020-12-15             Vaccine Approval        -0.687963
```

Statistical Tests

```
from statsmodels.tsa.stattools import adfuller
from scipy.stats import jarque_bera

# Augmented Dickey-Fuller test for stationarity
adf_test = adfuller(df['Close'])
print("ADF Statistic:", adf_test[0])
print("p-value:", adf_test[1])

# Jarque-Bera test for normality
jb_test = jarque_bera(df['Close'])
print("Jarque-Bera Statistic:", jb_test[0])
print("p-value:", jb_test[1])
```

```
ADF Statistic: -1.7172755789704768
p-value: 0.4222342775667287
Jarque-Bera Statistic: 282.1807582831075
p-value: 5.311618675316005e-62
```

Correlation Analysis

```
[ ]  # Simulate macroeconomic indicators data
     np.random.seed(42)
     df['Interest_Rate'] = np.random.uniform(0, 5, df.shape[0])
     df['Inflation_Rate'] = np.random.uniform(-1, 10, df.shape[0])
     df['Stock_Index'] = np.random.uniform(1000, 5000, df.shape[0])

     # Correlation matrix
     correlation_matrix = df[['Close', 'Interest_Rate', 'Inflation_Rate', 'Stock_Index']].corr()
     print(correlation_matrix)
```

```
                   Close  Interest_Rate  Inflation_Rate  Stock_Index
Close           1.000000      -0.014164        0.006452     0.008731
Interest_Rate  -0.014164       1.000000        0.008186    -0.001345
Inflation_Rate  0.006452       0.008186        1.000000     0.006351
Stock_Index     0.008731      -0.001345        0.006351     1.000000
```

Summary

- This project supports informed decision-making and strategy development for researchers and analysts

MENTOR NESS

Thank you

- **ANIKET KUMAR**
- **aniketsk668@gmail.com**