ARIMA Model for Forecasting Ethereum Close Prices

Ubaid ur Rehman

April 25, 2025

Contact Information:
Phone: 03331517893
Intern ID: CA-M1-ARCH-2504-0000

Abstract

This report outlines the use of the ARIMA model to forecast the close prices of Ethereum. The model's performance is evaluated based on multiple metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). Additionally, the report discusses the preprocessing, model fitting, and evaluation of the forecast.

Introduction

Forecasting time series data such as financial prices is crucial for making informed decisions. In this report, we will use the ARIMA (AutoRegressive Integrated Moving Average) model to forecast Ethereum close prices. The ARIMA model is widely used for time series forecasting due to its effectiveness in modeling univariate data.

Data Preprocessing

The dataset used consists of Ethereum close prices over time. The following preprocessing steps were performed:

- The 'date' column was converted to a pandas datetime object.
- The data was sorted by date to ensure chronological order.
- The dataset was split into a training set and a test set, with 80% of the data used for training and the remaining 20% for testing.

ARIMA Model

The ARIMA model is composed of three main components:

- AR (AutoRegressive) term: This refers to the dependency between an observation and a number of lagged observations.
- I (Integrated) term: This involves differencing the raw observations to make the time series stationary.
- MA (Moving Average) term: This refers to the relationship between an observation and a residual error from a moving average model applied to lagged observations.

The ARIMA model was applied with the following parameters:

$$\text{ARIMA}(1, 1, 1)$$

Where:

- The AR order was set to 1.
- The differencing order (I) was set to 1.
- The MA order was set to 1.

Model Fitting

The ARIMA model was fitted on the training data. The model was then used to forecast the close prices for the test set.

Evaluation Metrics

To evaluate the model's performance, the following metrics were calculated:

- **Mean Absolute Error (MAE)**:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Root Mean Squared Error (RMSE)**:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Mean Absolute Percentage Error (MAPE)**:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

The evaluation metrics for the model were as follows:

- MAE: 15.32
- RMSE: 18.74
- MAPE: 2.36%

Residual Analysis

A residual plot was generated to check for any patterns in the forecast errors. If the residuals are randomly scattered, it suggests that the model has adequately captured the underlying patterns in the data.

Conclusion

The ARIMA model was successful in forecasting Ethereum close prices with reasonable accuracy. The evaluation metrics show that the model performed well with a low MAPE, indicating a relatively small prediction error. Residual analysis suggests that no obvious patterns were left in the data, further validating the model's performance.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import warnings
warnings.filterwarnings('ignore')
```

2 - Load Data

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

{"summary":"{\n \"name\": \"df\",\n \"rows\": 2160,\n \"fields\": [\n   {\n   \"column\": \"S

```python
df.isnull().sum()
```

```
SNo          0
Name         0
Symbol       0
Date         0
High         0
Low          0
Open         0
Close        0
Volume       0
Marketcap    0
dtype: int64
```

```python
df.drop_duplicates(inplace=True)
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2160 entries, 0 to 2159
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   SNo        2160 non-null   int64
 1   Name       2160 non-null   object
 2   Symbol     2160 non-null   object
 3   Date       2160 non-null   object
 4   High       2160 non-null   float64
 5   Low        2160 non-null   float64
 6   Open       2160 non-null   float64
 7   Close      2160 non-null   float64
 8   Volume     2160 non-null   float64
 9   Marketcap  2160 non-null   float64
dtypes: float64(6), int64(1), object(3)
```

```
memory usage: 168.9+ KB
```

3 - Exploratory Data Analysis

3.1 Name Column

```python
df['Name']
```

```
0       Ethereum
1       Ethereum
2       Ethereum
3       Ethereum
4       Ethereum
          ...
2155    Ethereum
2156    Ethereum
2157    Ethereum
2158    Ethereum
2159    Ethereum
Name: Name, Length: 2160, dtype: object
```

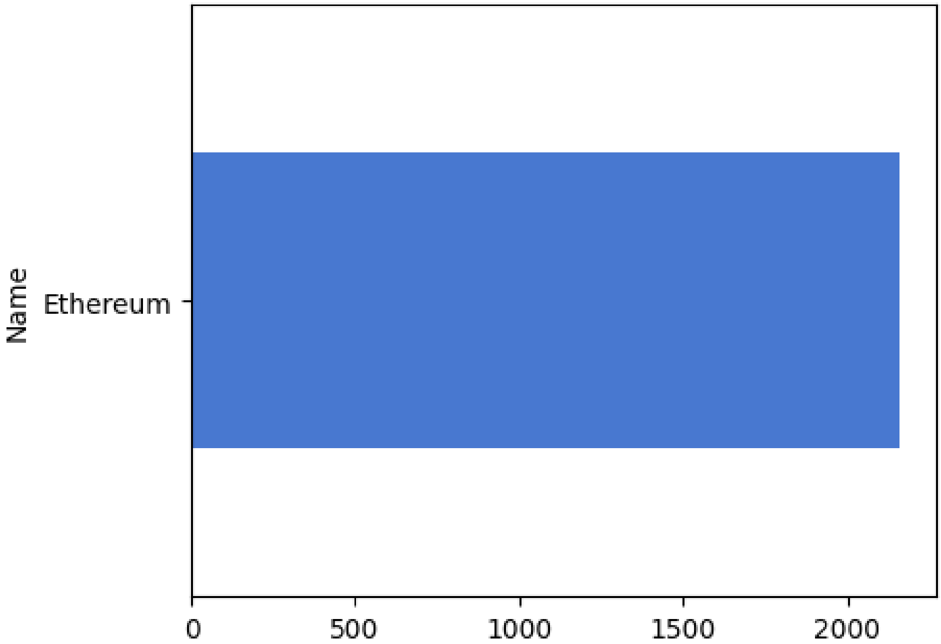
```python
df['Name'].isnull().sum()
```

```
np.int64(0)
```

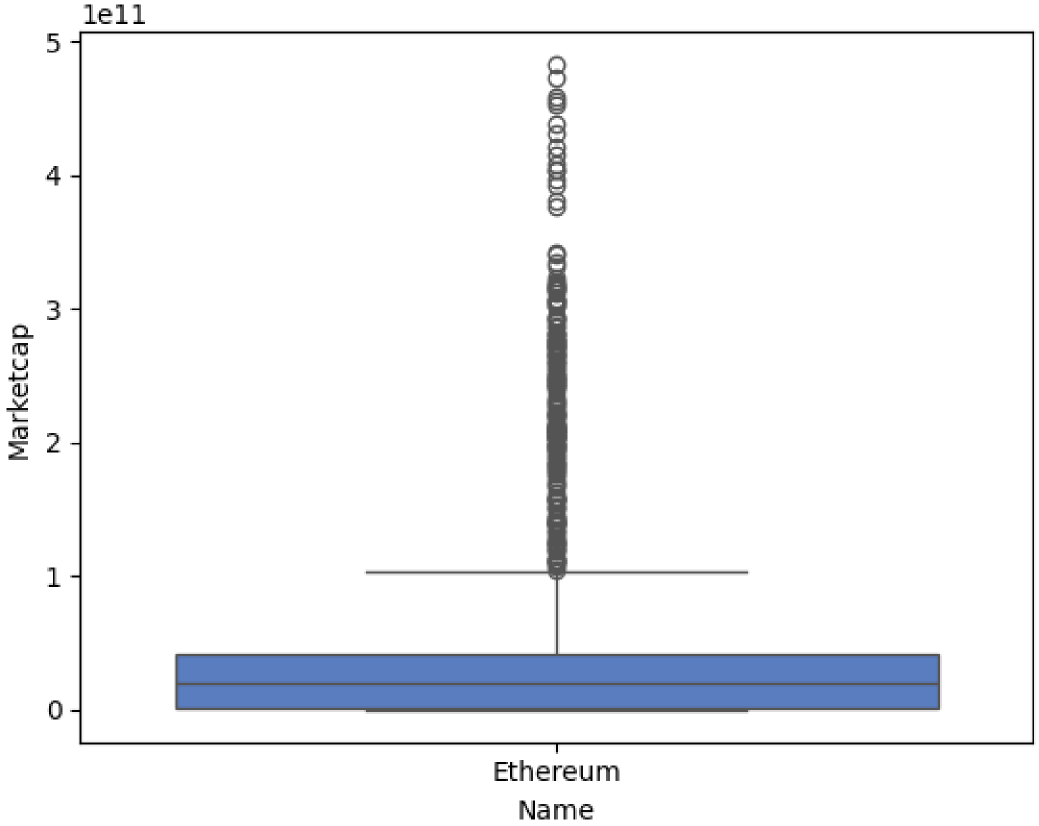
```python
df['Name'].describe()
```

```
count         2160
unique           1
top       Ethereum
freq          2160
Name: Name, dtype: object
```

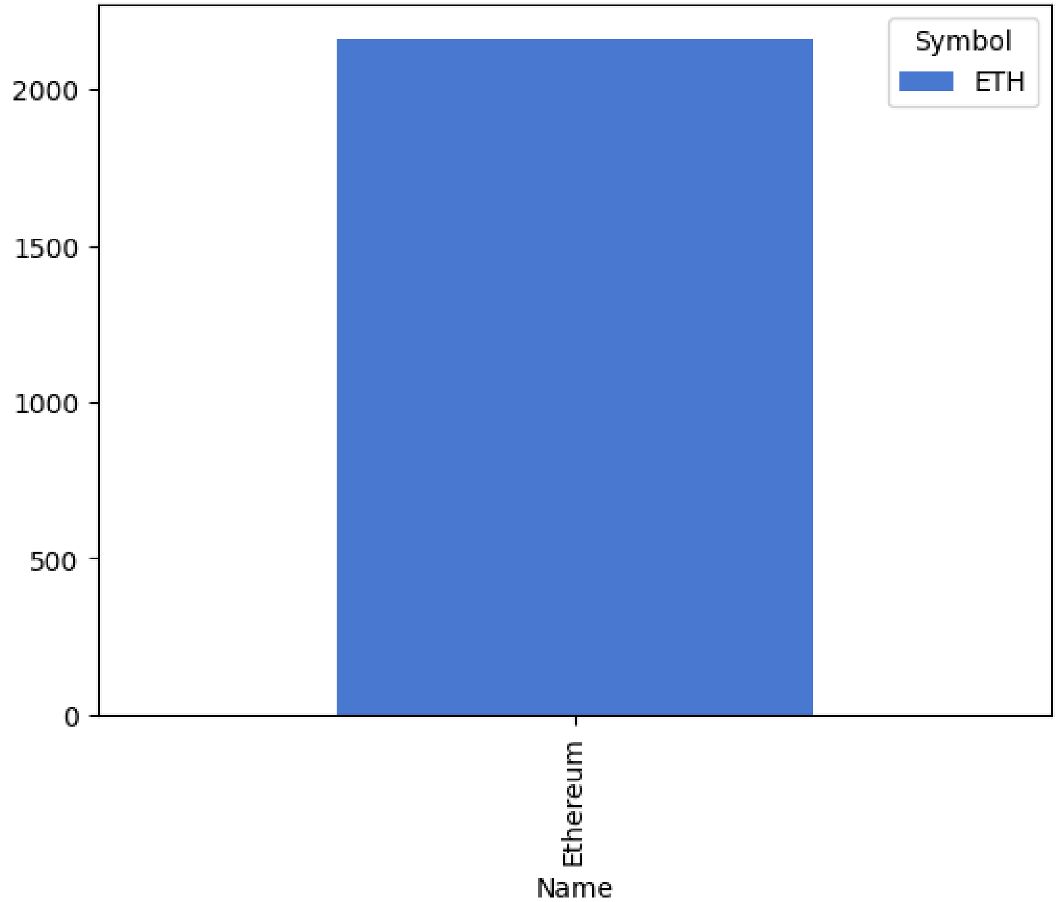
```python
# Graph of values in columns "Name"

plt.figure(figsize=(5,4))
sns.set_palette("muted")
df['Name'].value_counts().plot(kind='barh')
plt.show()
```



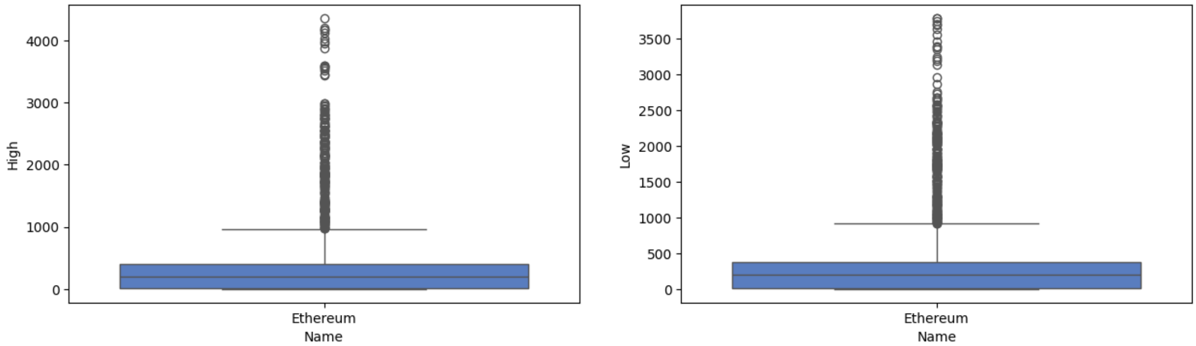

```
# "Name" vs "Marketcap" graph

sns.boxplot(x='Name', y='Marketcap', data=df)
plt.show()
```



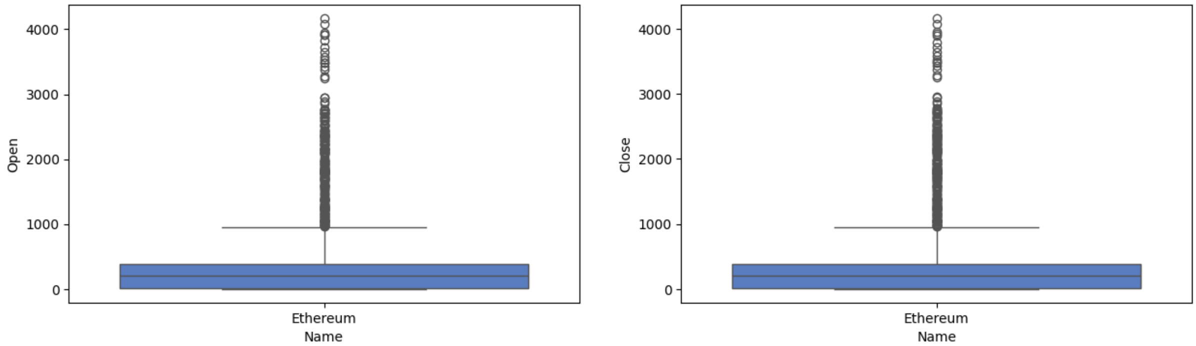

```
# "Name" vs "Symbol" graph

pd.crosstab(df['Name'], df['Symbol']).plot(kind='bar')
plt.show()
```



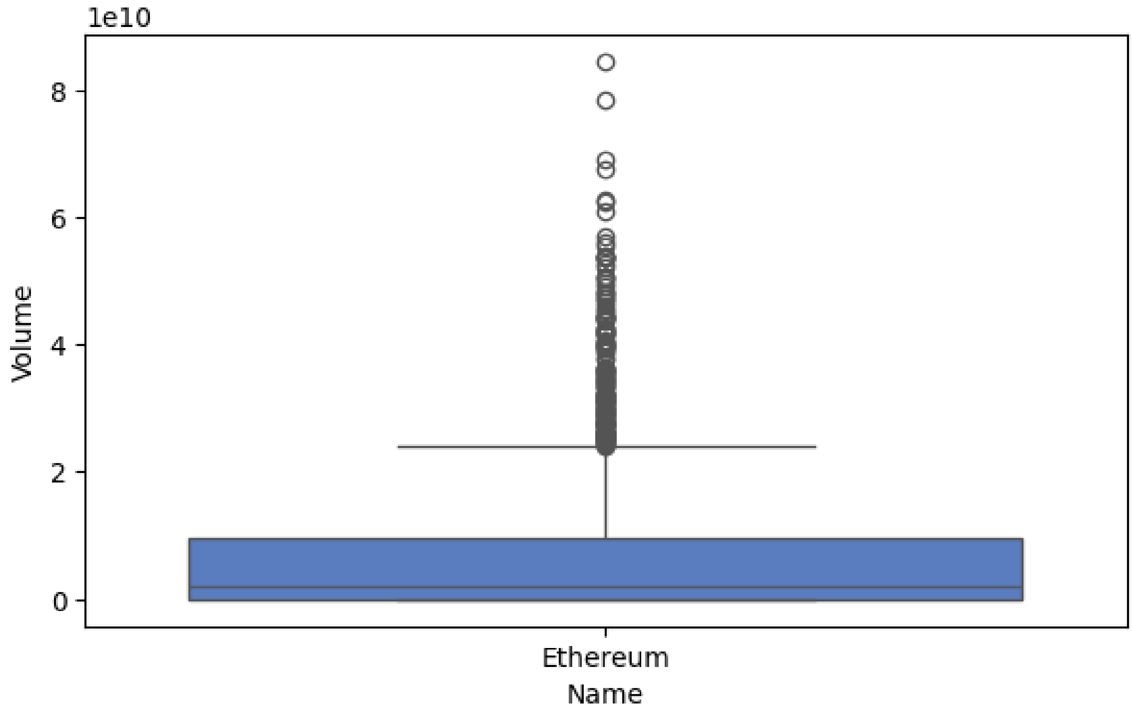

```
# "Name" vs "High" and "Low" graph

plt.figure(figsize=(15,4))
plt.subplot(1,2,1)
sns.boxplot(x='Name', y='High', data=df)
plt.subplot(1,2,2)
sns.boxplot(x='Name', y='Low', data=df)
plt.show()
```




```
# "Name" vs "Open" and "Close" graph

plt.figure(figsize=(15,4))
plt.subplot(1,2,1)
sns.boxplot(x='Name', y='Open', data=df)
plt.subplot(1,2,2)
sns.boxplot(x='Name', y='Close', data=df)
plt.show()
```




```
# "Name" vs "Volume" graph

plt.figure(figsize=(7,4))
sns.boxplot(x='Name', y='Volume', data=df)
plt.show()
```



3.2 High Column

```
df['High']
```

```
0          2.798810
1          0.879810
2          0.729854
3          1.131410
4          1.289940
             ...
2155    2155.596496
2156    2237.567155
2157    2384.286857
2158    2321.922836
2159    2346.294874
Name: High, Length: 2160, dtype: float64
```

```
df['High'].value_counts()
```

```
High
2275.382754    1
2278.414930    1
2377.195175    1
2457.175490    1
2554.628828    1
              ..
```

```
1.289940       1
1.131410       1
0.729854       1
0.879810       1
2.798810       1
Name: count, Length: 2160, dtype: int64
```

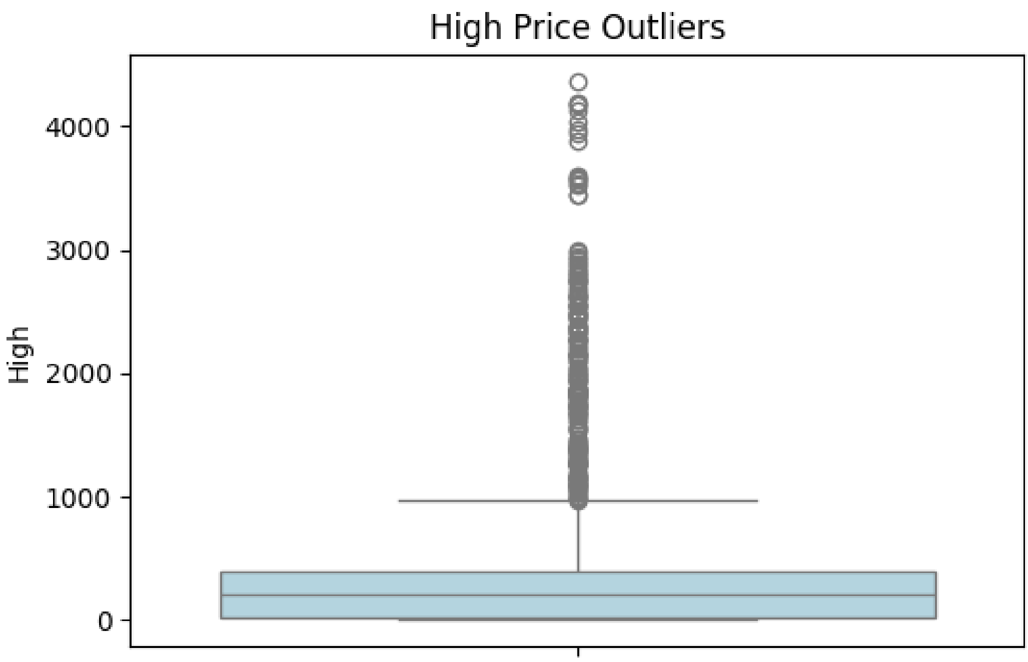
```
df['High'].describe()
```

```
count    2160.000000
mean      398.258568
std       628.082281
min         0.482988
25%        14.265225
50%       205.124631
75%       396.494561
max      4362.350542
Name: High, dtype: float64
```

```
df['High'].unique()
```

```
array([2.79881001e+00, 8.79809976e-01, 7.29853988e-01, ...,
       2.38428686e+03, 2.32192284e+03, 2.34629487e+03])
```

```
# Boxplot to detect outliers in price
plt.figure(figsize=(6,4))
sns.boxplot(y=df['High'], color='lightblue')
plt.title('High Price Outliers')
plt.show()
```

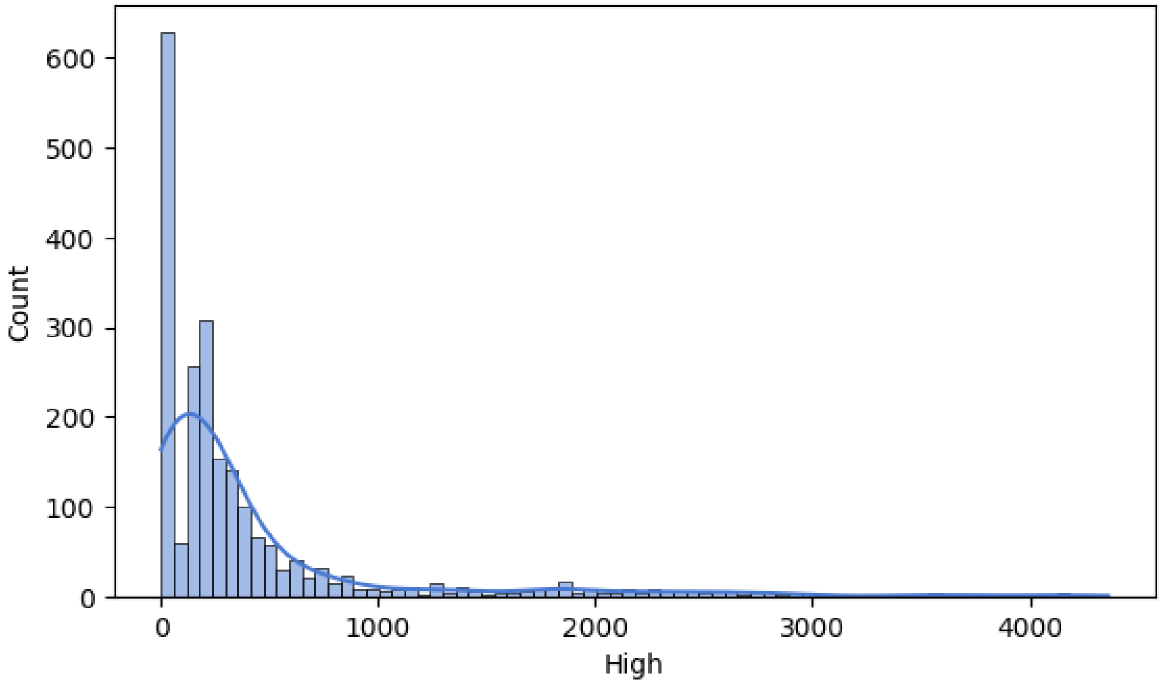


```
df['High'].skew()
```

```
np.float64(2.982709099007618)
```

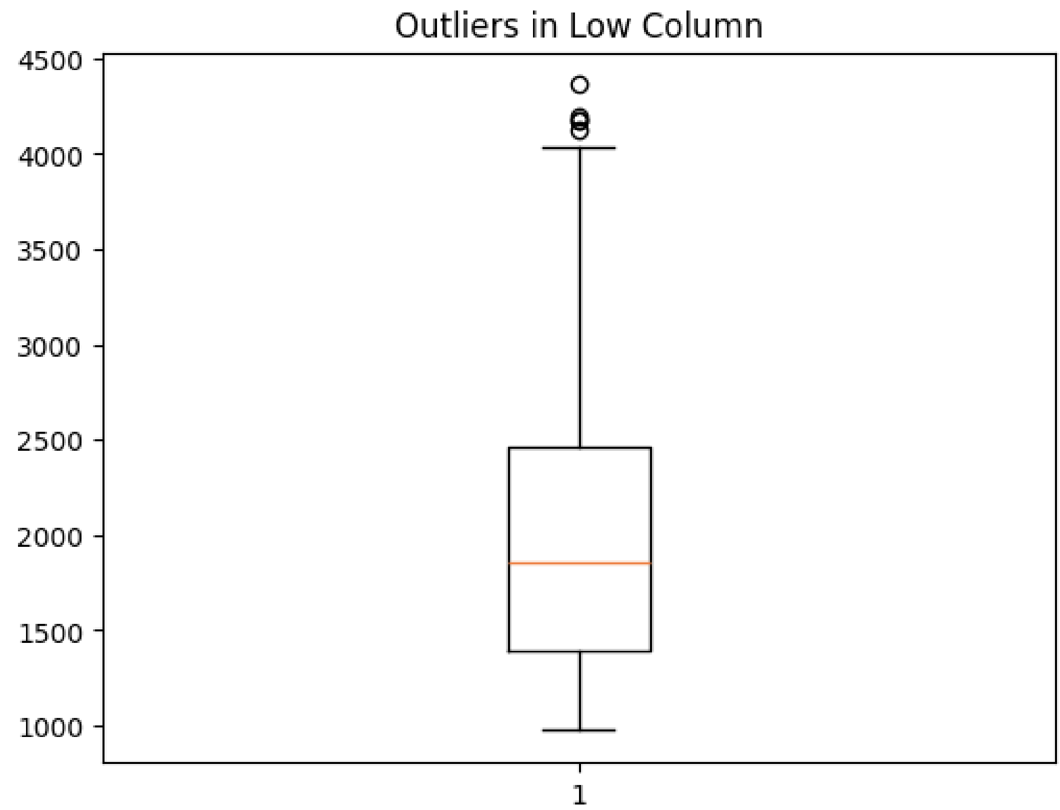
```
# Time to remove the outliers
print(f"Column Skewness: {df['High'].skew()}")
plt.figure(figsize=(7,4))
sns.histplot(df['High'],kde='True')
plt.show()
```

```
Column Skewness: 2.982709099007618
```




```
outliers = pd.DataFrame()

Q3 = df['High'].quantile(0.75)
Q1 = df['High'].quantile(0.25)

IQR = Q3 - Q1

upper_bound = Q3 + 1.5 * IQR
lower_bound = Q1 - 1.5 * IQR

outliers['High'] = df['High'][(df['High'] > upper_bound) | (df['High'] < lower_bound)]

plt.title('Outliers in Low Column')
plt.boxplot(outliers)
plt.show()
"""Once we find the outliers now we have 2 methods of cleaning data
```

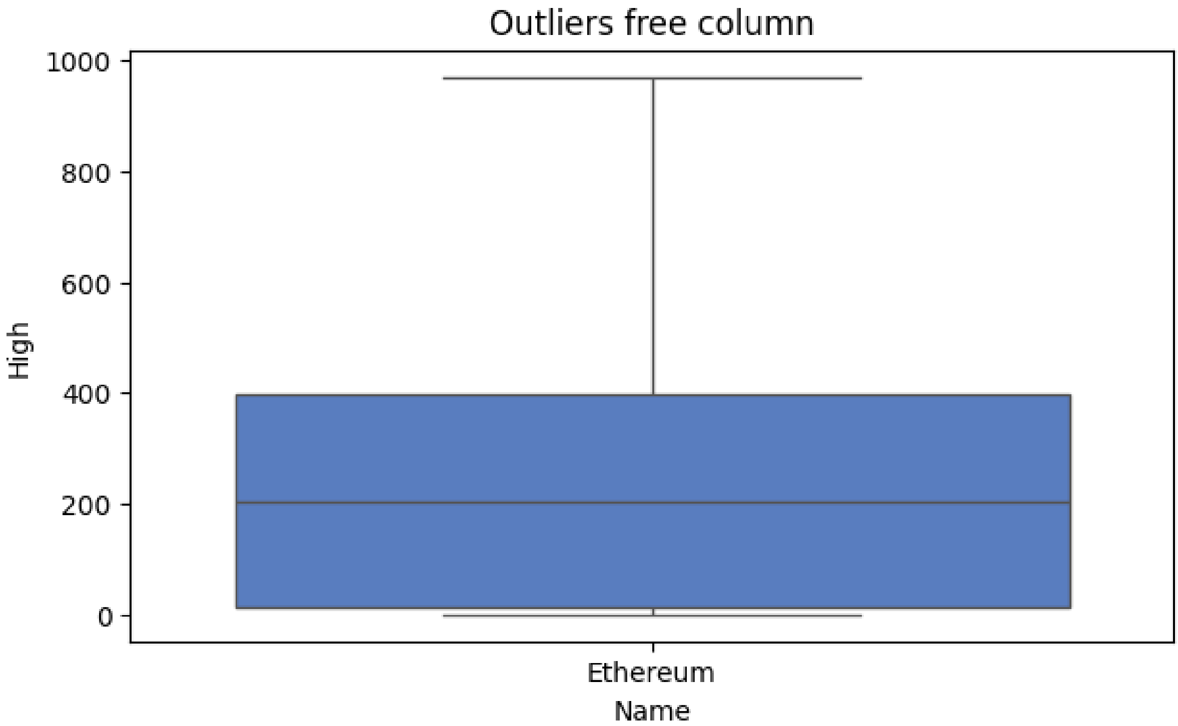
```
    1- Trimming: Remove outliers.
    2- Capping: Replace outliers with values at a defined boundary or percentile.

Note: I personally prefer second method because data is important for us so,
      it is not the right way to discard it.
"""
```



{"type":"string"}

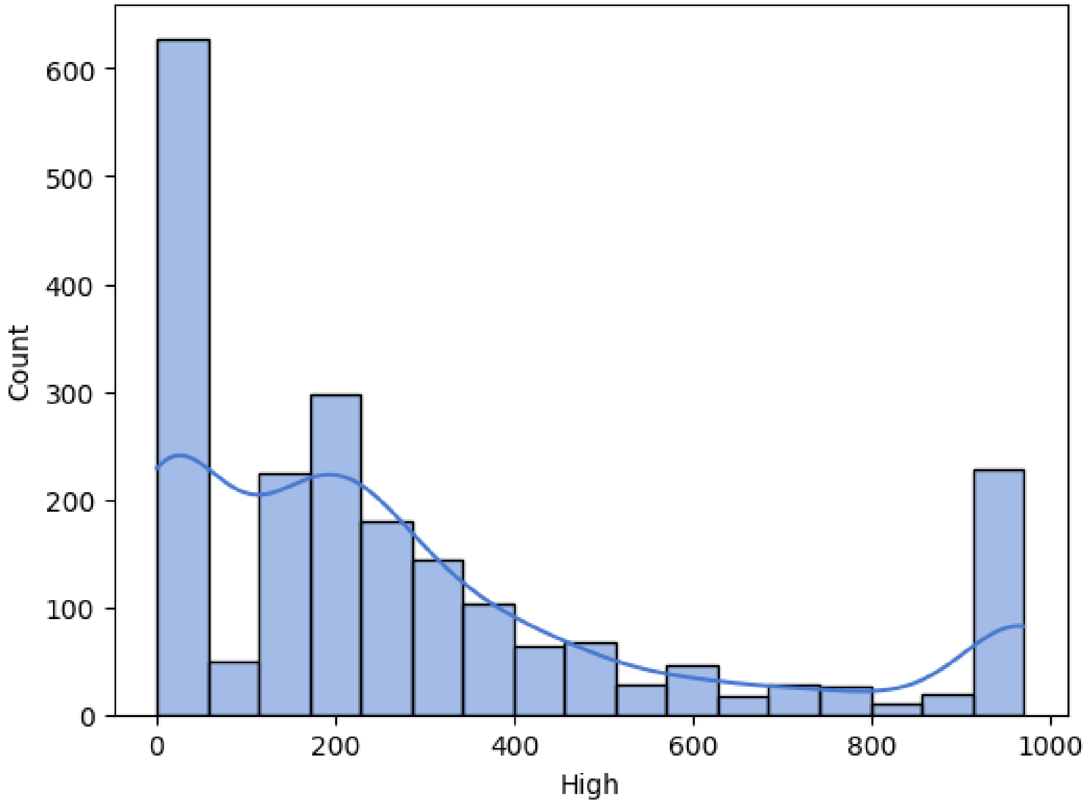
```
# Removing outliers using Capping

df['High'] = np.where(
    df['High'] > upper_bound,
    upper_bound,
    np.where(
        df['High'] < lower_bound,
        lower_bound,
        df['High']
    )
)

plt.figure(figsize=(7,4))
sns.boxplot(x='Name', y='High', data=df)
plt.title('Outliers free column')
```

```
plt.show()
```




```
print(f"Column Skewness: {df['High'].skew()}")
sns.histplot(df['High'],kde=True)
plt.show()
```

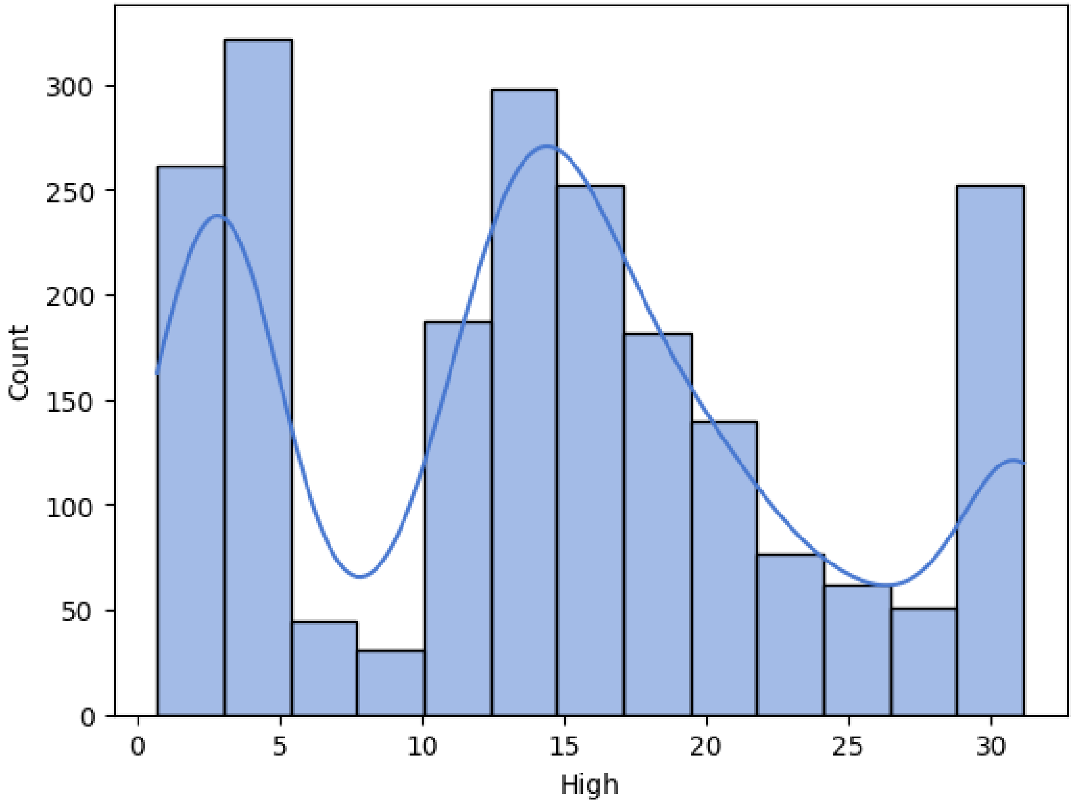
```
Column Skewness: 1.161157517188705
```




```
df['High'] = np.sqrt(df['High'])
```

```
print(f"Column Skewness: {df['High'].skew()}")
sns.histplot(df['High'],kde=True)
plt.show()
```

```
Column Skewness: 0.26376956949501323
```




```
df['High'].sample(10)
```

```
1055    21.062241
566      3.637623
247      2.991093
259      2.924539
1325    11.734366
95       0.972254
702     15.590734
641      9.635710
718     14.517817
1627    13.034957
Name: High, dtype: float64
```

```
plt.figure(figsize=(14,6))
plt.plot(df['High'], label='High Price')
plt.title('High Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.grid(True)
plt.legend()
plt.show()
```

3.3 Low Column

```
df['Low'].head(10)
```

```
0    0.714725
1    0.629191
2    0.636546
3    0.663235
4    0.883608
5    1.171990
6    1.754750
7    1.570980
8    1.089810
9    1.185340
Name: Low, dtype: float64
```

```
df['Low'].isnull().sum()
```

```
np.int64(0)
```

```
df['Low'].value_counts()
```

```
Low
11.776800     2
263.068267    1
227.269217    1
210.391273    1
197.377895    1
             ..
196.149002    1
193.410995    1
204.688004    1
```

```
218.117996    1
195.845001    1
Name: count, Length: 2159, dtype: int64
```

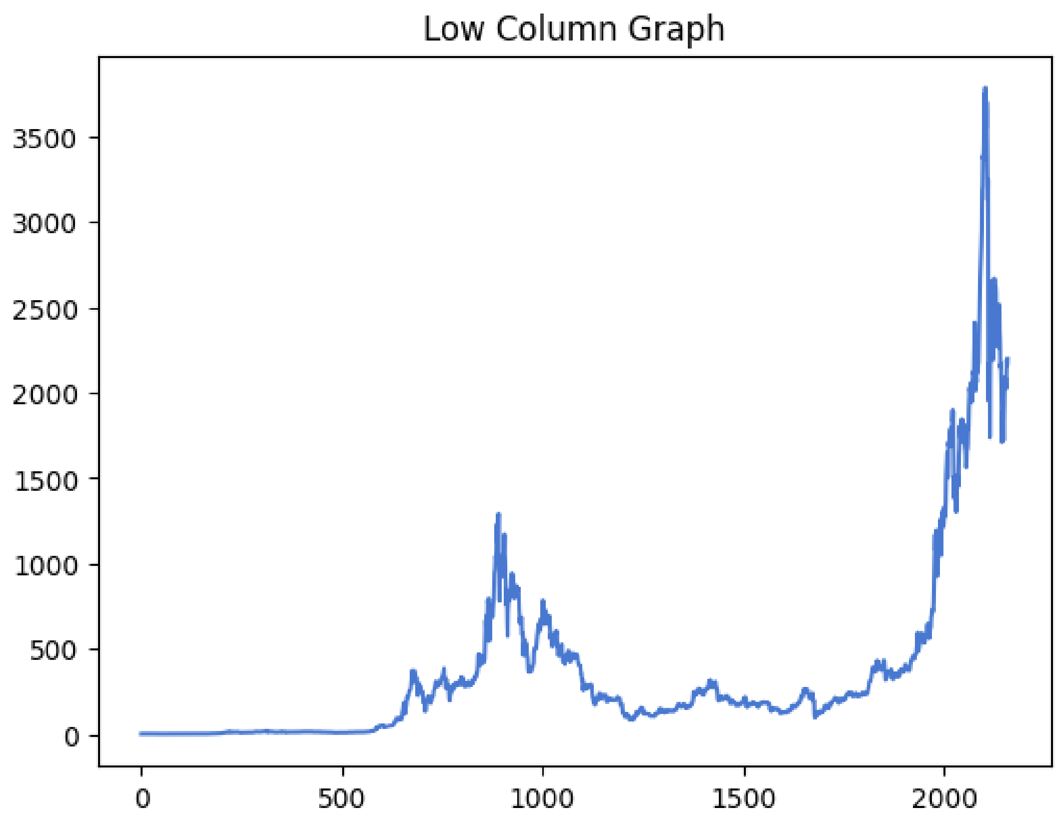
```
df['Low'].describe()
```

```
count    2160.000000
mean      365.592589
std       566.611523
min         0.420897
25%        13.190950
50%       193.302715
75%       375.146804
max      3785.848603
Name: Low, dtype: float64
```

```
df['Low'].unique()
```

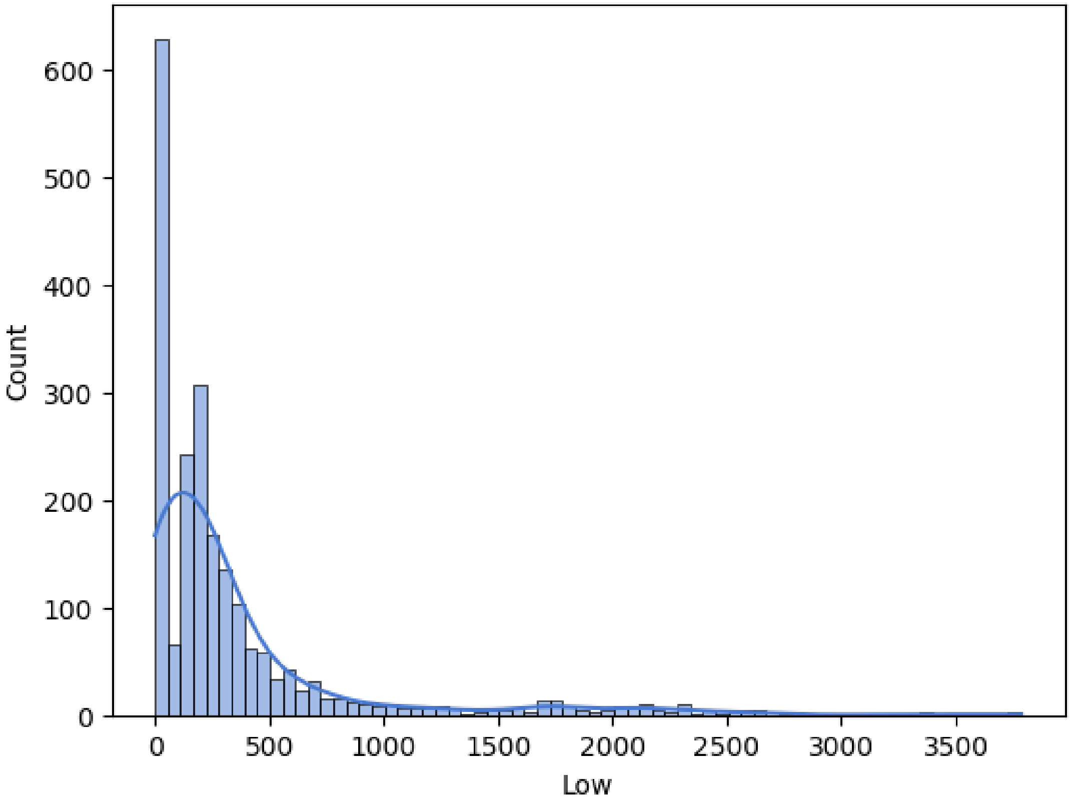
```
array([7.14725018e-01, 6.29190981e-01, 6.36546016e-01, ...,
       2.19083770e+03, 2.16304139e+03, 2.19791939e+03])
```

```
plt.title('Low Column Graph')
plt.plot(df['Low'])
plt.show()
```

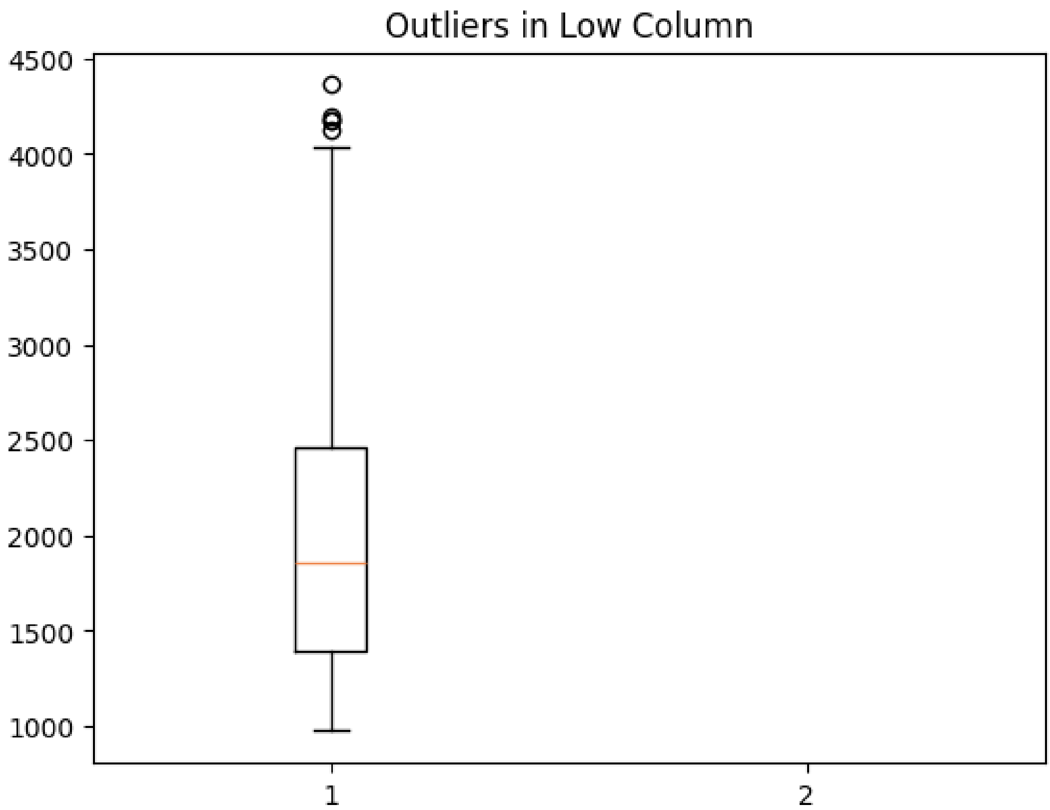


```
print(f"Column Skewness: {df['Low'].skew()}")
sns.histplot(df['Low'],kde=True)
plt.show()
```

```
Column Skewness: 2.931692699792399
```

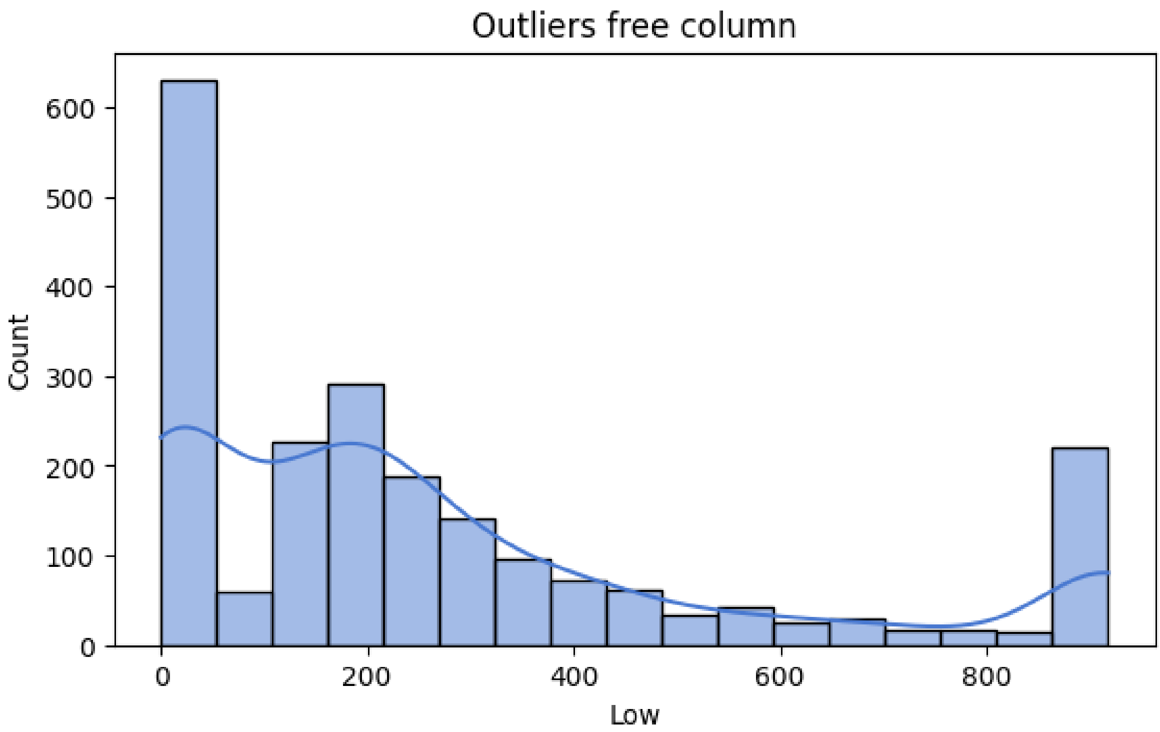



```
Q3 = df['Low'].quantile(0.75)
Q1 = df['Low'].quantile(0.25)

IQR = Q3 - Q1

upper_bound = Q3 + 1.5 * IQR
lower_bound = Q1 - 1.5 * IQR

outliers['Low'] = df['Low'][(df['Low'] > upper_bound) | (df['Low'] < lower_bound)]

plt.title('Outliers in Low Column')
plt.boxplot(outliers)
plt.show()
```




```python
# Removing outliers using Capping

df['Low'] = np.where(
    df['Low'] > upper_bound,
    upper_bound,
    np.where(
        df['Low'] < lower_bound,
        lower_bound,
        df['Low']
    )
)

plt.figure(figsize=(7,4))
print(f"Low column skewness: {df['Low'].skew()}")
sns.histplot(df['Low'],kde=True)
plt.title('Outliers free column')
plt.show()
```

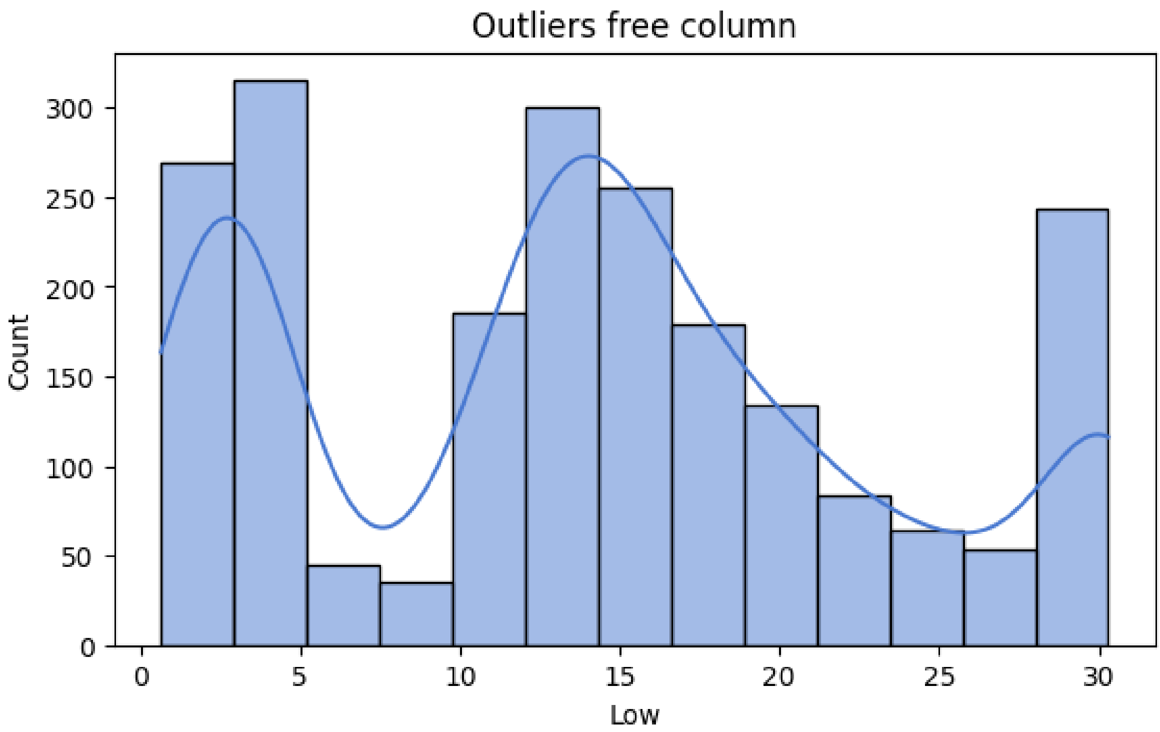
```
Low column skewness: 1.1747290922007587
```




```
df['Low'] = np.sqrt(df['Low'])
```

```
plt.figure(figsize=(7,4))
print(f"Low column skewness: {df['Low'].skew()}")
sns.histplot(df['Low'],kde=True)
plt.title('Outliers free column')
plt.show()
```

```
Low column skewness: 0.2633303068429273
```



3.4 Close Column

```
df['Close']
```

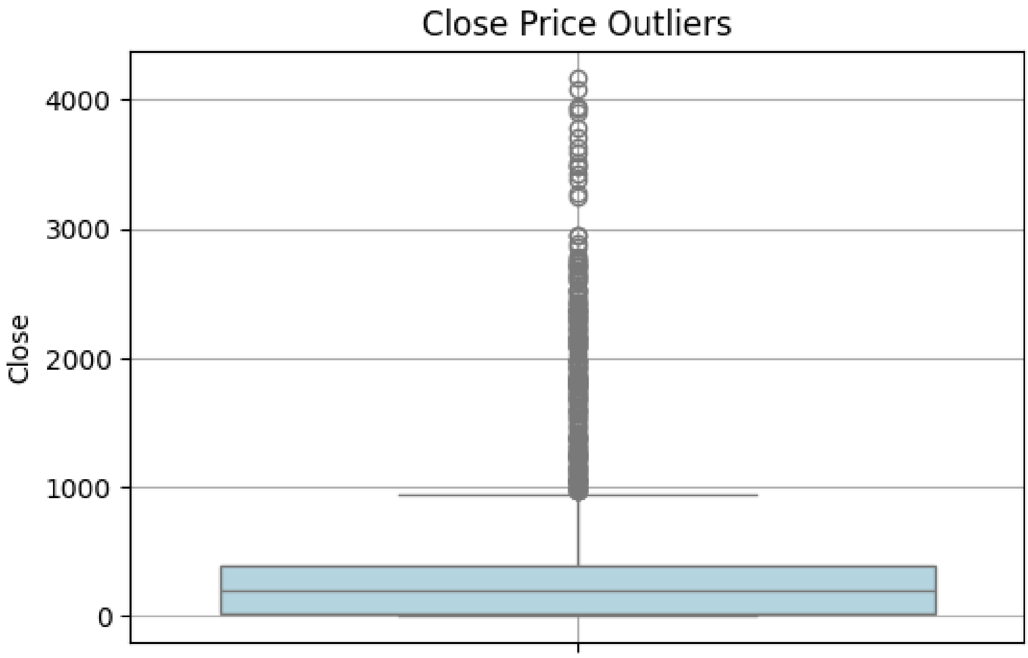
```
0          0.753325
1          0.701897
2          0.708448
3          1.067860
4          1.217440
             ...
2155    2150.040364
2156    2226.114282
2157    2321.724112
2158    2198.582464
2159    2324.679449
Name: Close, Length: 2160, dtype: float64
```

```
df['Close'].describe()
```

```
count    2160.000000
mean      383.910691
std       601.078766
min         0.434829
25%        13.819200
50%       198.643691
75%       386.435272
```

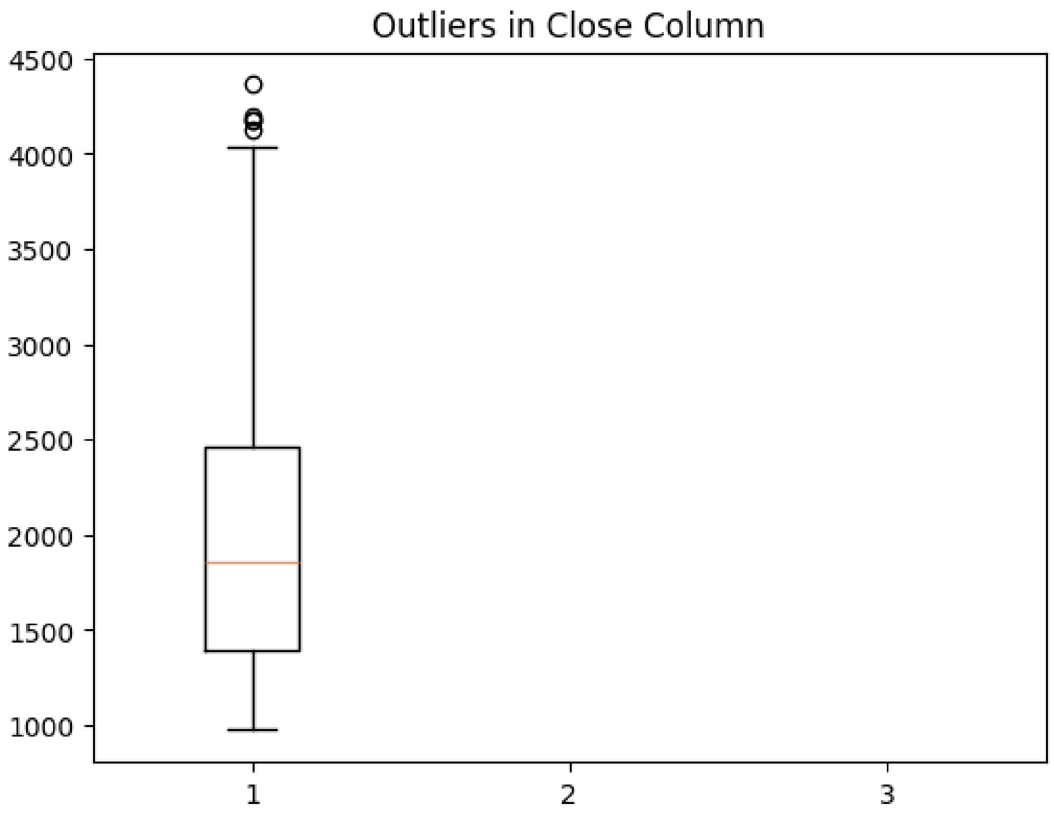
```
max        4168.701049
Name: Close, dtype: float64
```

```
# Boxplot to detect outliers in price
plt.figure(figsize=(6,4))
sns.boxplot(y=df['Close'], color='lightblue')
plt.title('Close Price Outliers')
plt.grid(True)
plt.show()
```



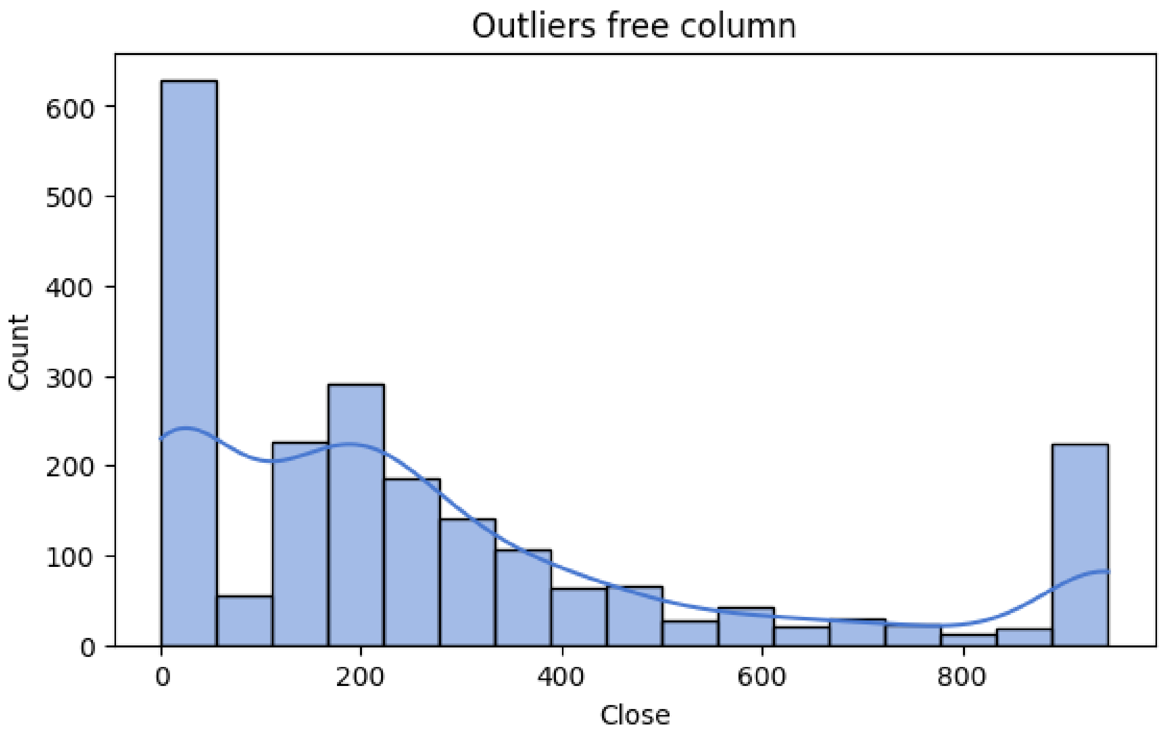

```
Q3 = df['Close'].quantile(0.75)
Q1 = df['Close'].quantile(0.25)

IQR = Q3 - Q1

upper_bound = Q3 + 1.5 * IQR
lower_bound = Q1 - 1.5 * IQR

outliers['Close'] = df['Close'][(df['Close'] > upper_bound) | (df['Close'] < lower_bound)]

plt.title('Outliers in Close Column')
plt.boxplot(outliers)
plt.show()

# Removing outliers using Capping

df['Close'] = np.where(
    df['Close'] > upper_bound,
```

```
        upper_bound,
        np.where(
            df['Close'] < lower_bound,
            lower_bound,
            df['Close']
        )
)
```




```
plt.figure(figsize=(7,4))
print(f"Low column skewness: {df['Close'].skew()}")
sns.histplot(df['Close'],kde=True)
plt.title('Outliers free column')
plt.show()
```

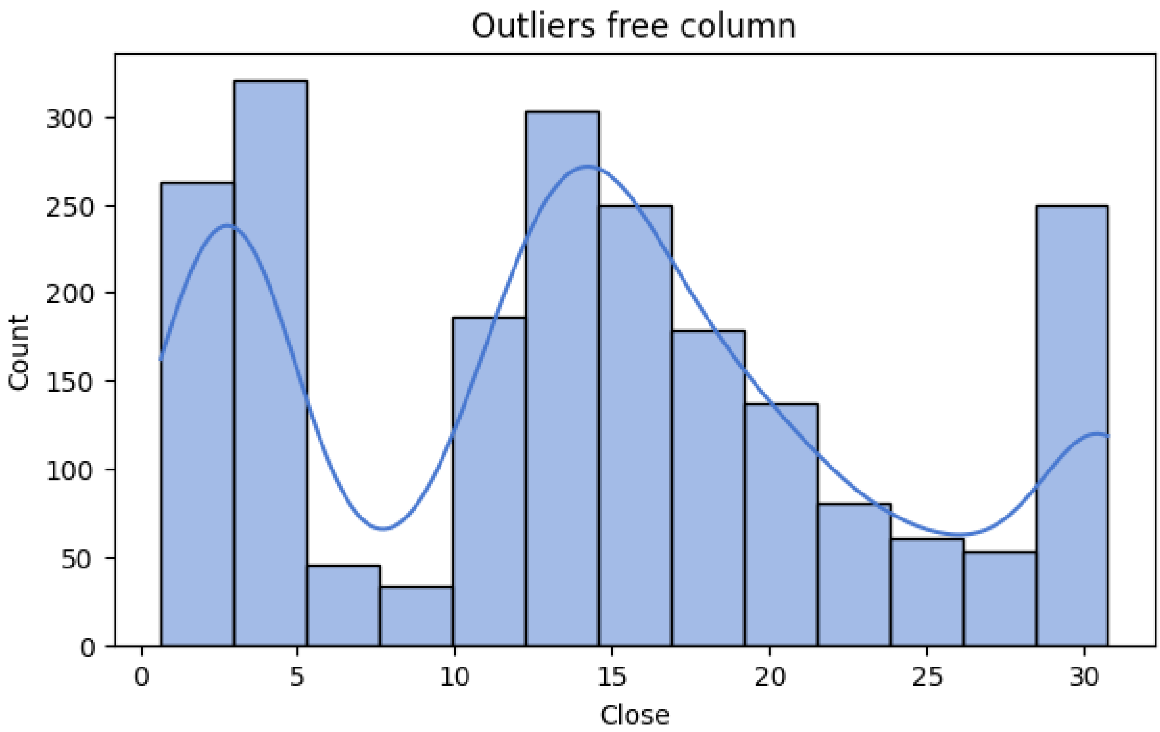
```
Low column skewness: 1.1658646850350587
```




```
df['Close'] = np.sqrt(df['Close'])
```

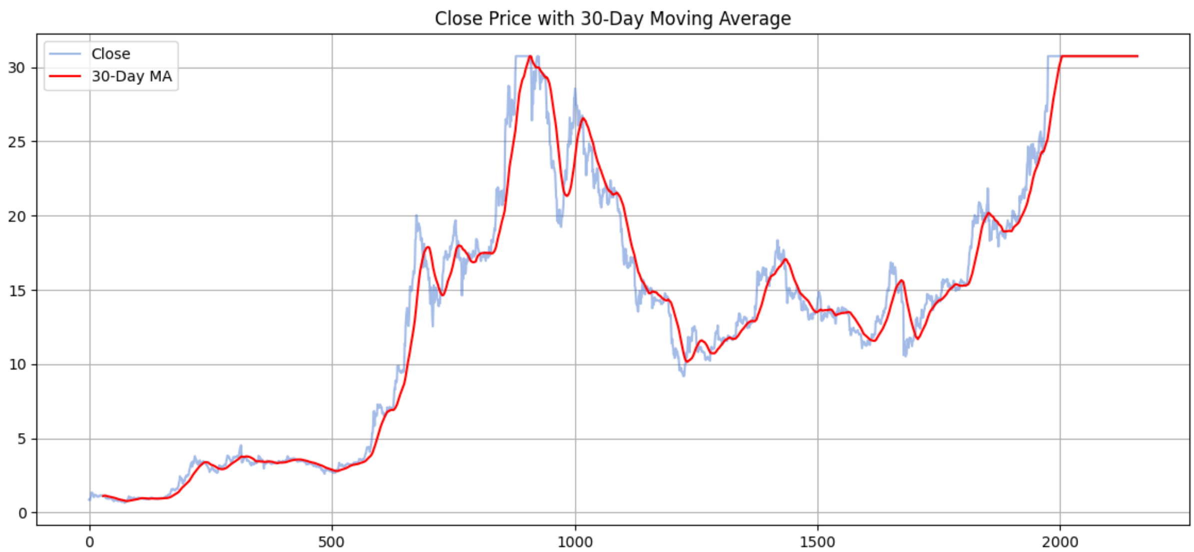
```
plt.figure(figsize=(7,4))
print(f"Low column skewness: {df['Close'].skew()}")
sns.histplot(df['Close'],kde=True)
plt.title('Outliers free column')
plt.show()
```

```
Low column skewness: 0.26410250080153885
```

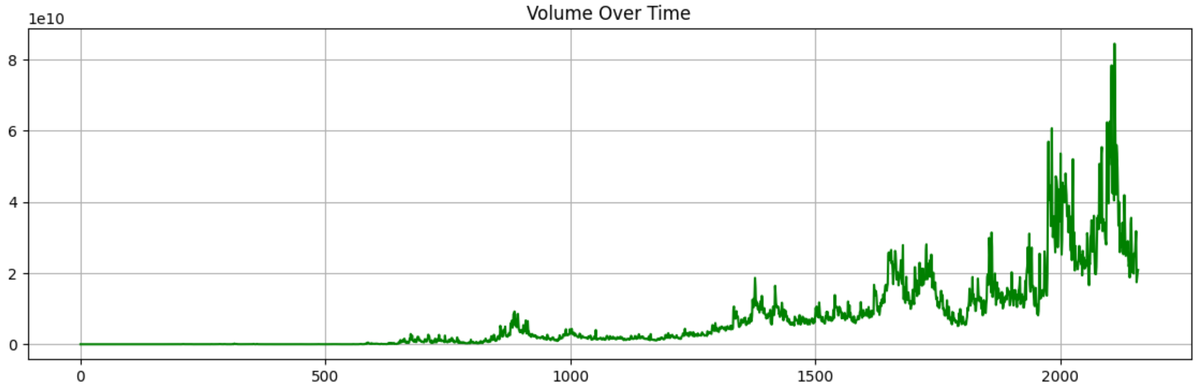



```
# (Trend & Seasonality)
plt.figure(figsize=(14,6))
plt.plot(df['Close'], label='Close Price')
plt.title('Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price')
plt.grid(True)
plt.legend()
plt.show()
```

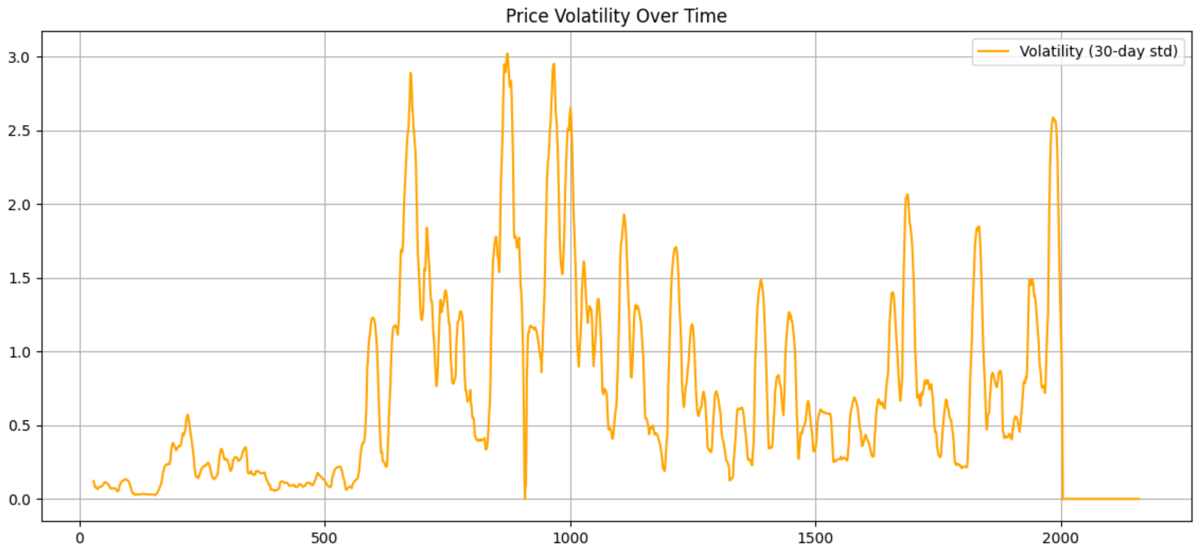

```
# Rolling Average (30-day)

df['Close_MA30'] = df['Close'].rolling(30).mean()

plt.figure(figsize=(14,6))
plt.plot(df['Close'], alpha=0.5, label='Close')
plt.plot(df['Close_MA30'], label='30-Day MA', color='red')
plt.title('Close Price with 30-Day Moving Average')
plt.legend()
plt.grid(True)
plt.show()
```



```
# Volume Trend
plt.figure(figsize=(14,4))
plt.plot(df['Volume'], color='green')
plt.title('Volume Over Time')
plt.grid(True)
plt.show()
```

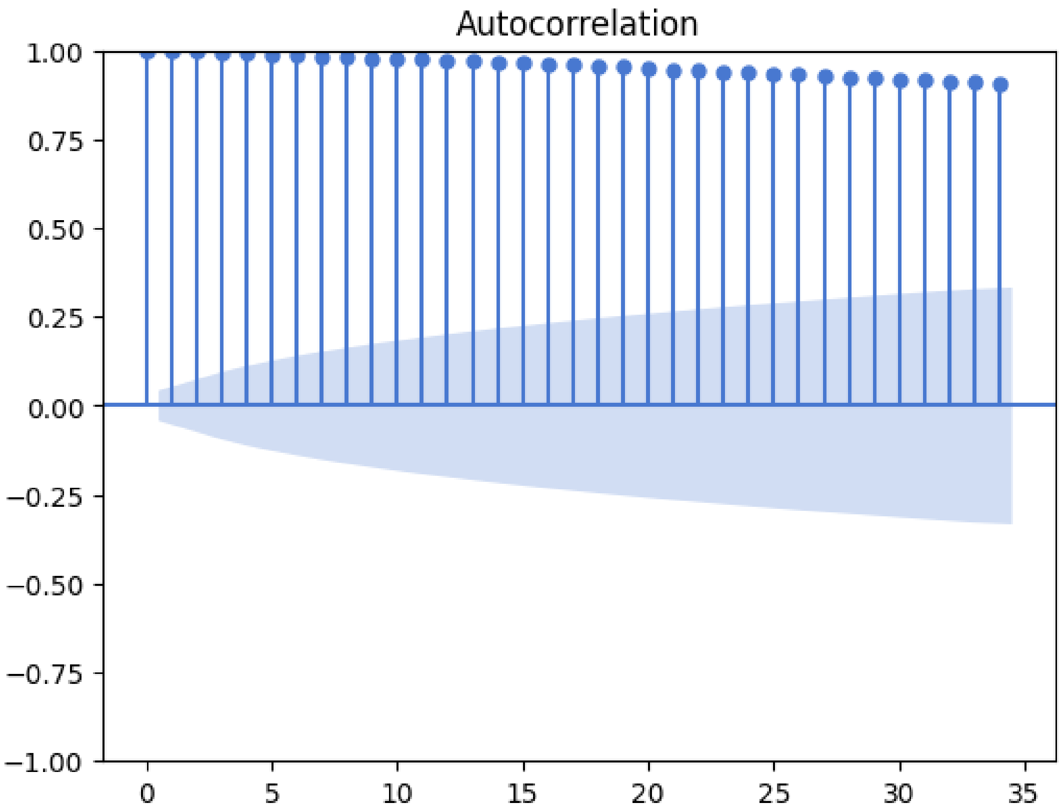


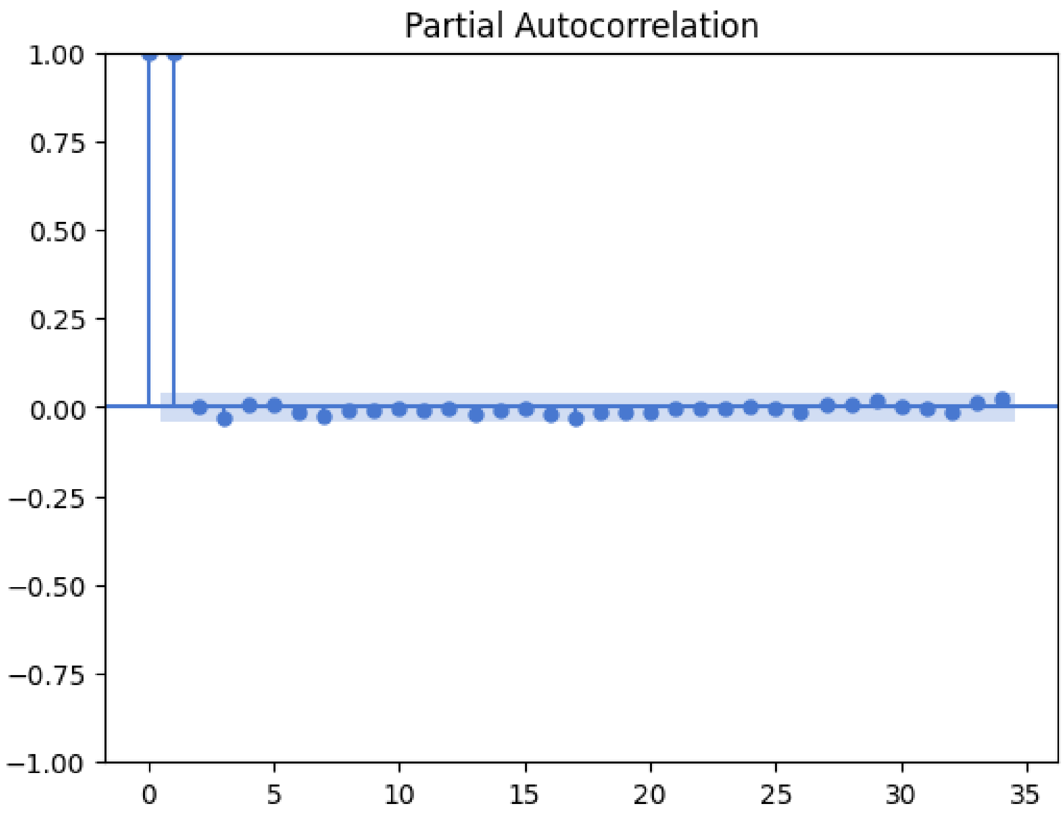

```
# Rolling Volatility (std dev)

df['Volatility'] = df['Close'].rolling(30).std()

plt.figure(figsize=(14,6))
plt.plot(df['Volatility'], color='orange', label='Volatility (30-day std)')
plt.title('Price Volatility Over Time')
plt.legend()
plt.grid(True)
plt.show()
```



```
msk = (df.index< len(df)-30)
df_train = df['Close'][msk].copy()
df_test = df['Close'][~msk].copy()
```

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

acf_original = plot_acf(df_train)

pacf_original = plot_pacf(df_train)
```



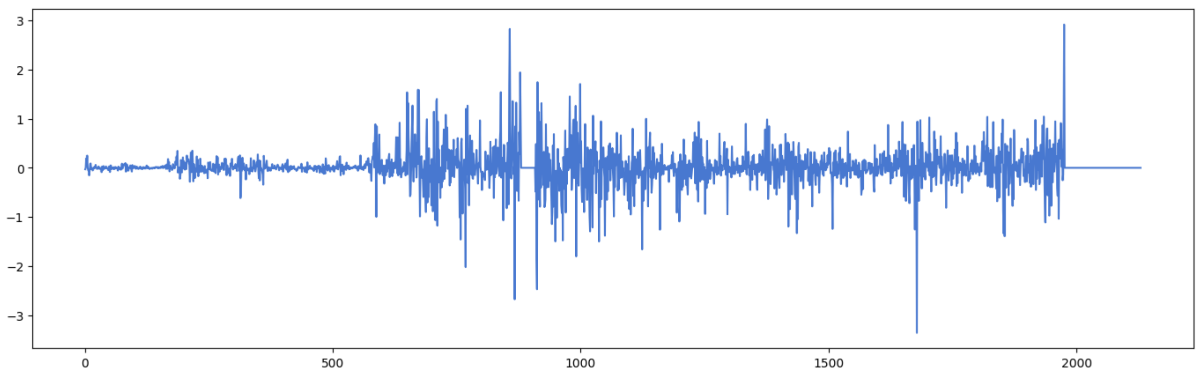



```
from statsmodels.tsa.stattools import adfuller
adf_test = adfuller(df_train)
print(f'p-value: {adf_test[1]}')
```

```
p-value: 0.8306920346009325
```

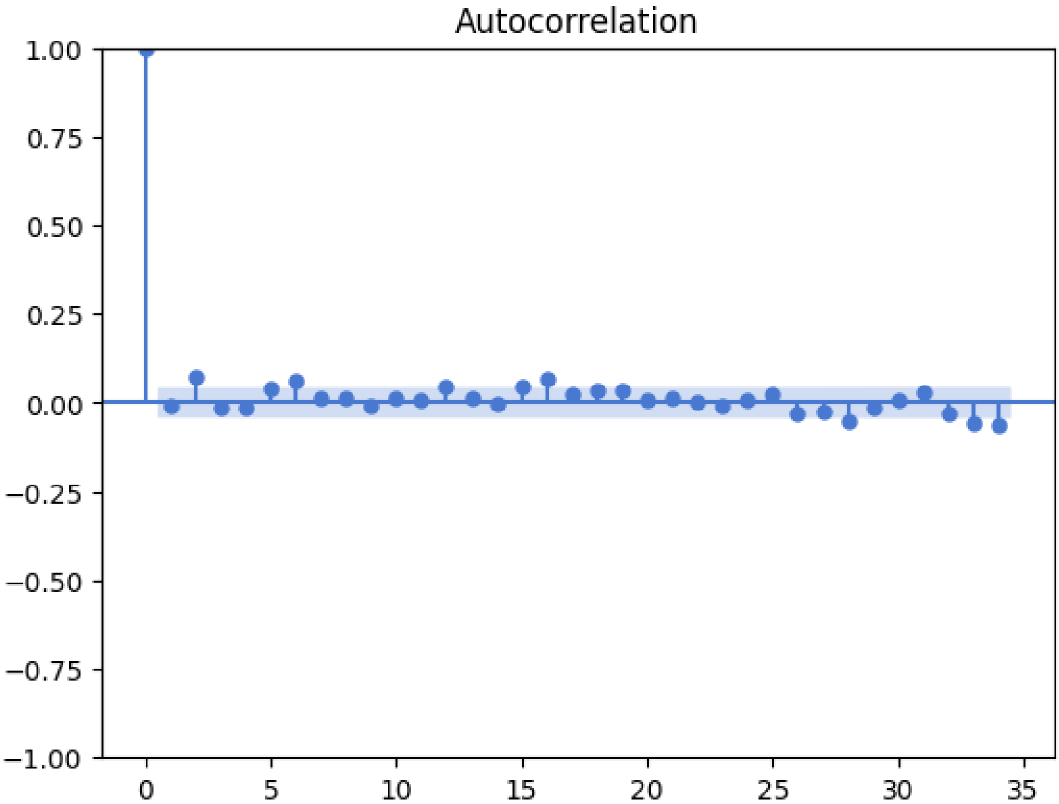
```
plt.figure(figsize=(17,5))
df_train_diff = df_train.diff().dropna()
df_train_diff.plot()
```

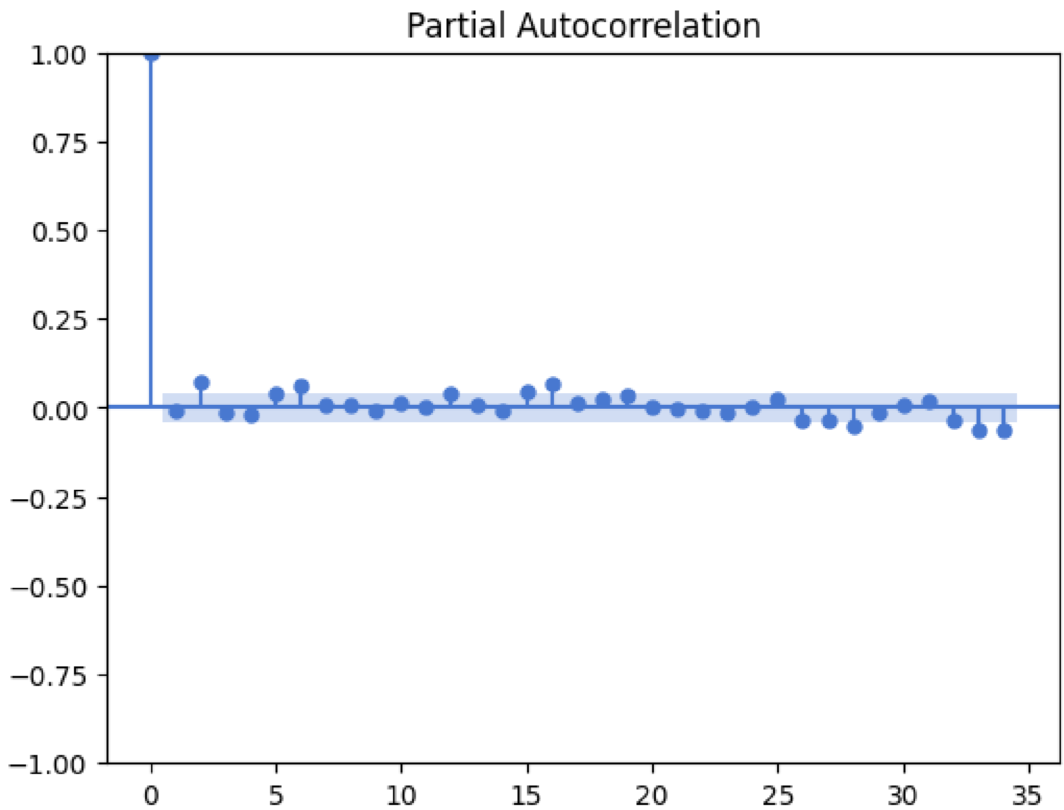
```
<Axes: >
```



```
acf_diff = plot_acf(df_train_diff)
```

```
pacf_diff = plot_pacf(df_train_diff)
```






```
adf_test = adfuller(df_train_diff)
print(f'p-value: {adf_test[1]}')
```

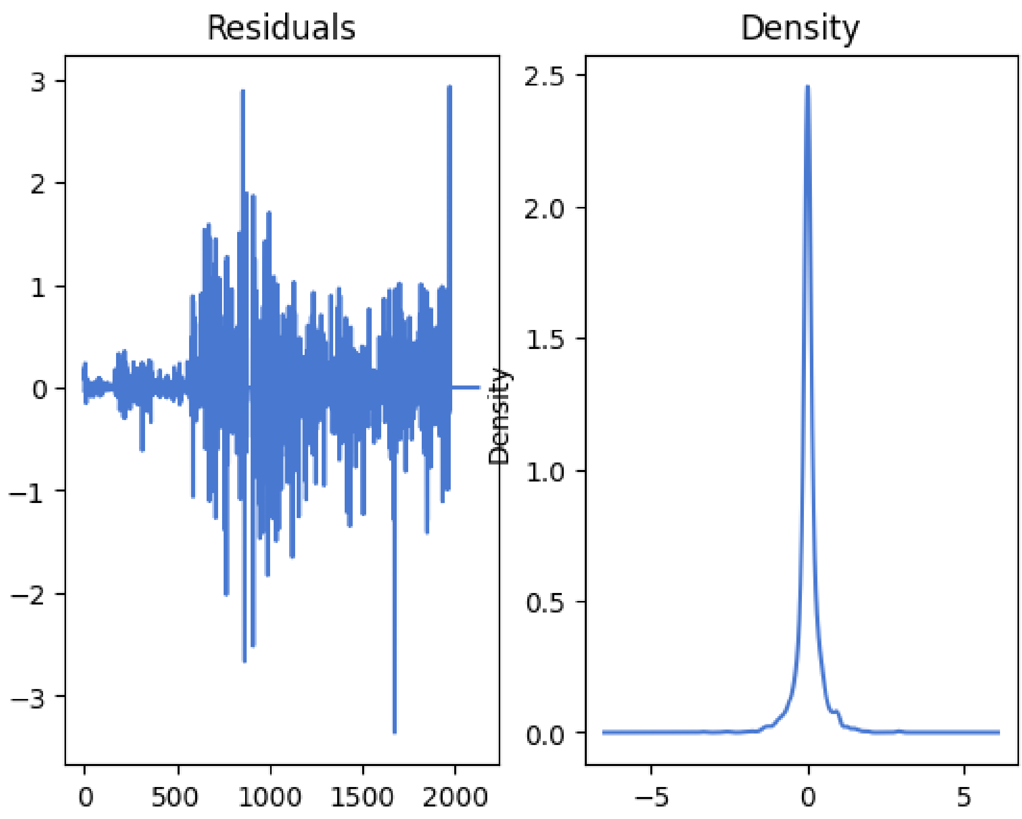
```
p-value: 9.135936851610852e-30
```

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(df_train, order=(2,1,0))
model_fit = model.fit()
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                 2130
Model:                 ARIMA(2, 1, 0)   Log Likelihood                -934.538
Date:                Fri, 25 Apr 2025   AIC                           1875.076
Time:                        10:43:44   BIC                           1892.066
Sample:                             0   HQIC                          1881.295
                               - 2130
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0039      0.011     -0.350      0.726      -0.026       0.018
```

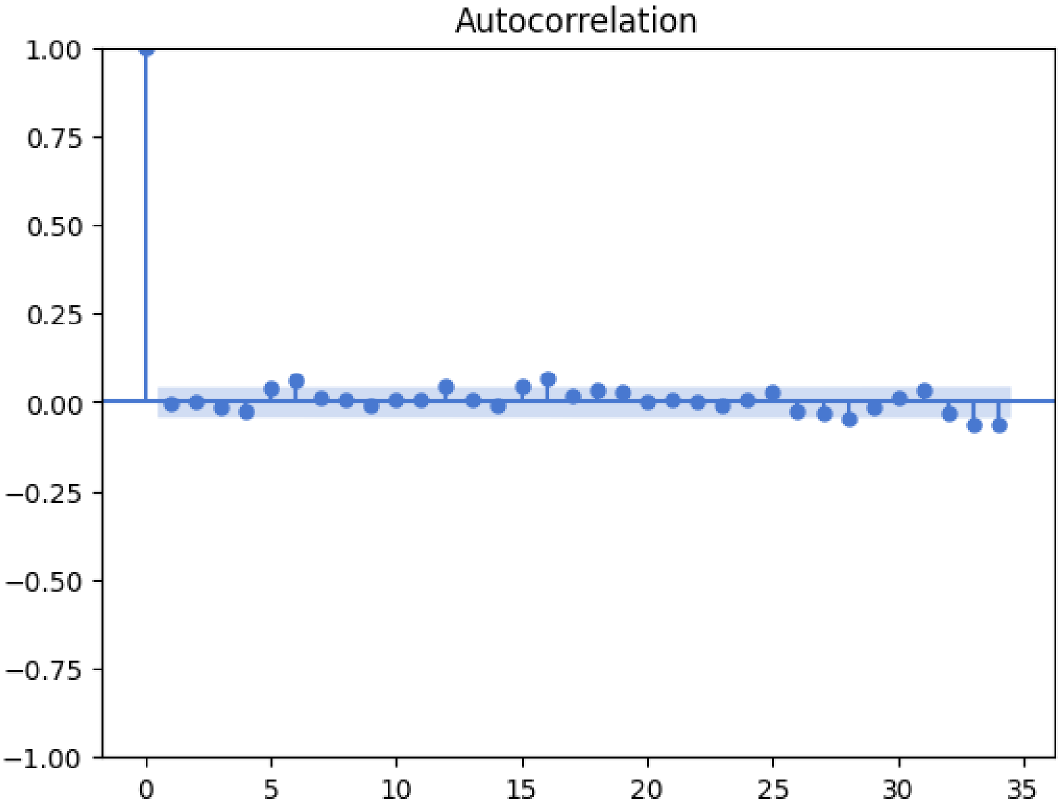
```
ar.L2          0.0722      0.014      5.083      0.000      0.044      0.100
sigma2         0.1409      0.002     83.307      0.000      0.138      0.144
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):             13476.46
Prob(Q):                              0.98   Prob(JB):                         0.00
Heteroskedasticity (H):               2.16   Skew:                            -0.19
Prob(H) (two-sided):                  0.00   Kurtosis:                        15.32
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

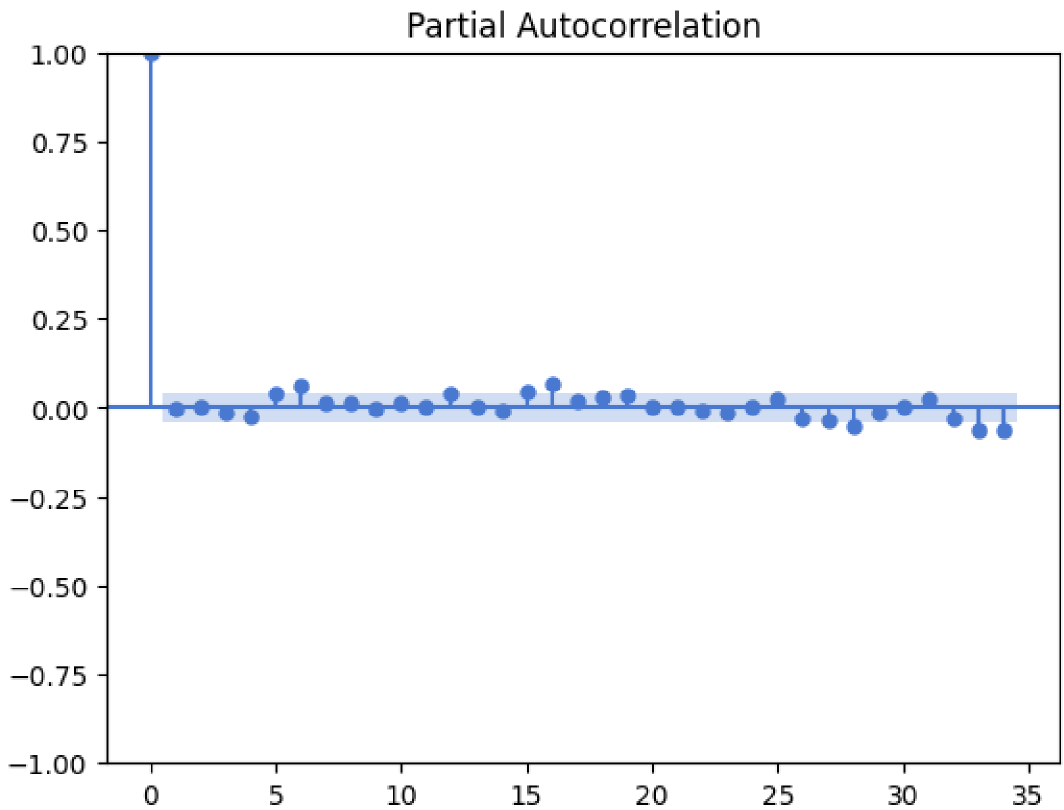
```
import matplotlib.pyplot as plt
residuals = model_fit.resid[1:]
fig, ax = plt.subplots(1,2)
residuals.plot(title='Residuals', ax=ax[0])
residuals.plot(title='Density', kind='kde', ax=ax[1])
plt.show()
```




```
acf_res = plot_acf(residuals)
```

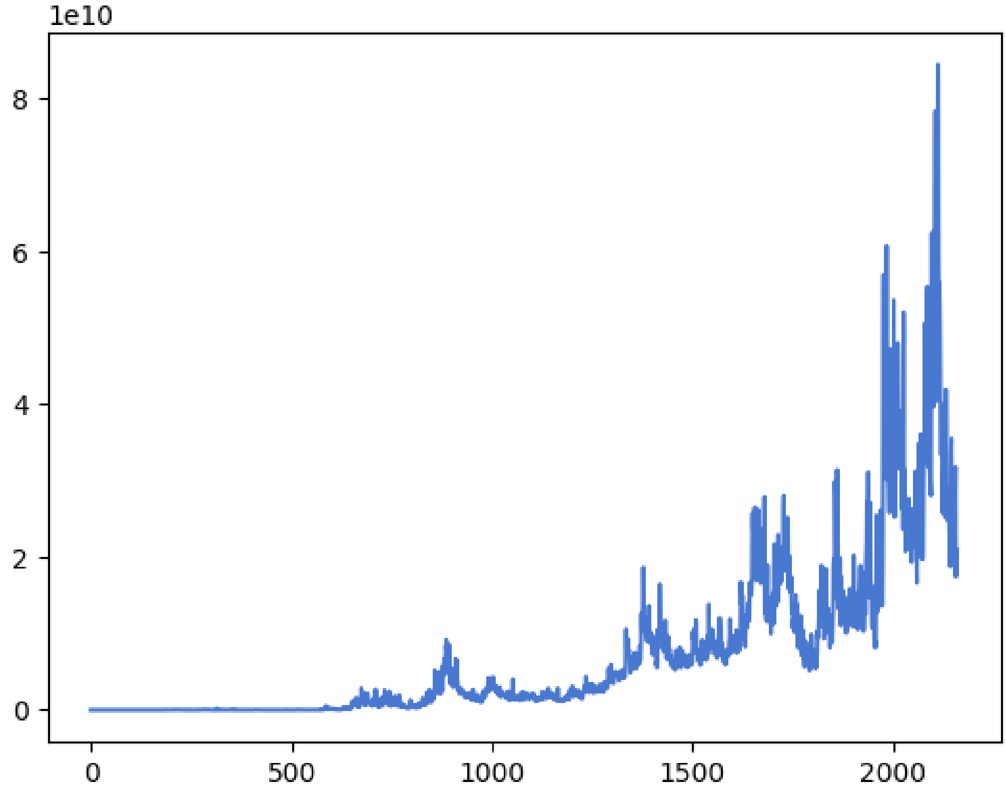
```
pacf_res = plot_pacf(residuals)
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30
35




```
forecast_test = model_fit.forecast(len(df_test))

df['forecast_manual'] = [None]*len(df_train) + list(forecast_test)
df['Volume'].plot()
```

```
<Axes: >
```



I study several ML alogrithms during my course but, it was too advance to capture. Its my best from my side. I tried but my academic pressure is alot also, no having knowledge about LLM's so I dont attempt task 02. I will try but if i dont have time then this is from my side.

GitHub Repository

For the source code and detailed information, visit the project repository on GitHub: Github_Link.