Task 01: Time Series Forecasting for Ethereum (ETH/USDT) Using ARIMA

Table of Contents:

Introduction

Ethereum, a leading and volatile cryptocurrency, poses challenges for price forecasting. This project utilizes the ARIMA model to predict ETH/USDT short-term price movements. By examining historical data, the goal is to assist

in smarter trading decisions through pattern recognition and model-based predictions.

Objectives:

- Analyze historical Ethereum trends
- Test for stationarity in the data
- Build an ARIMA model to identify patterns
- Assess model accuracy
- Forecast upcoming price trends

ARIMA Model Overview

The ARIMA (Autoregressive Integrated Moving Average) model is a widely used method for single-variable time series forecasting. It includes:

- AR (Autoregression): Relates current values to past ones
- I (Integration): Differencing used to stabilize the series
- MA (Moving Average): Captures patterns from past prediction errors

---

Project Setup

Libraries Required:

Python libraries such as pandas, matplotlib, statsmodels, etc.

```
[1]: import pandas as pd
     import numpy as npas
     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 pmdarima import auto_arima
     from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
     import matplotlib.dates as mdates
     import yfinance as yf
```

Data Source

Historical ETH/USDT daily data was collected from platforms like Binance, Coin Gecko, and Yahoo Finance. The dataset includes closing prices along with optional features like open, high, low, volume, and market cap.

Data Loading and Preparation

Data is imported from a CSV file, with the date set as the index and sorted chronologically. Necessary columns are selected, and missing values are checked before analysis.

```
[2]: # Load the data
     df = pd.read_csv('ETH-USD.csv')

     # Set 'Date' as index and sort the DataFrame
     df = df.set_index('Date')
     df = df.sort_index(ascending=True)

     # Display the first 5 rows
     display(df.head())
```

Data Preparation:

```
[3]: # Select relevant columns
     df_selected = df[['Open', 'High', 'Low', 'Close', 'Volume']]

     # Check for missing values
     print("There are", df_selected.isnull().sum().sum(), "null values.")

     display(df_selected.head())
```

```
There are 0 null values.
```

Exploratory Data Analysis

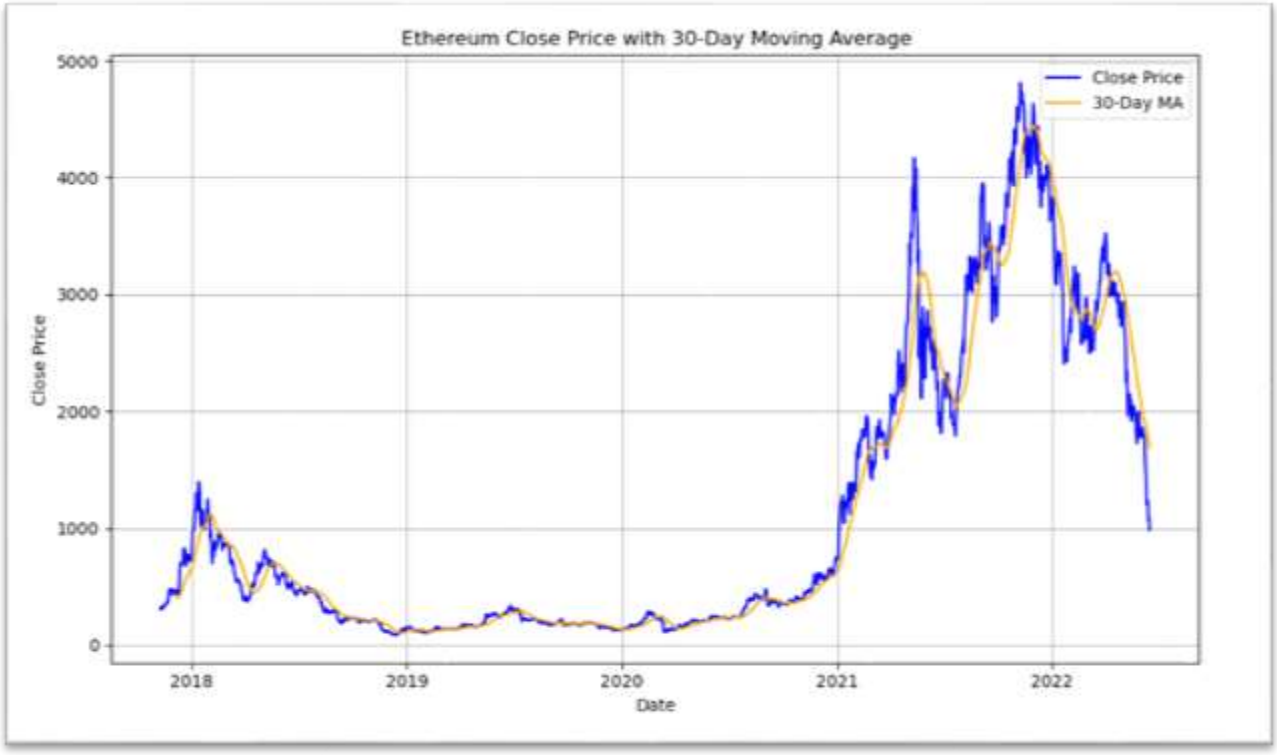
- Summary Statistics: Descriptive stats of key variables are computed.

```
[4]: # Calculate and display summary statistics
     print(df_selected.describe())

     # Print the shape of the DataFrame
     print("\nShape of dataset is:", df_selected.shape)

     # Explore data types (optional)
     print("\nData Types:\n", df_selected.dtypes)
```

- Visualizations:
  - ETH Close price with a 30-day moving average
  - Daily trading volume trends

```
[5]: import matplotlib.pyplot as plt

     # Ensure datetime index
     df_selected.index = pd.to_datetime(df_selected.index)

     # Create the figure
     plt.figure(figsize=(10, 6))

     # Plot Close Price
     plt.plot(df_selected.index, df_selected['Close'], label='Close Price', color='blue')

     # Plot 30-Day Moving Average
     plt.plot(df_selected.index, df_selected['Close'].rolling(window=30).mean(), label='30-Day MA', color='orange')

     # Set white background for figure and axes
     fig = plt.gcf()
     fig.patch.set_facecolor('white')  # White background for figure

     ax = plt.gca()
     ax.set_facecolor('white')         # White background for axes

     # Add labels, legend, title, and grid
     plt.xlabel('Date')
     plt.ylabel('Close Price')
     plt.title('Ethereum Close Price with 30-Day Moving Average')
     plt.legend()
     plt.grid(True)

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

Graph:

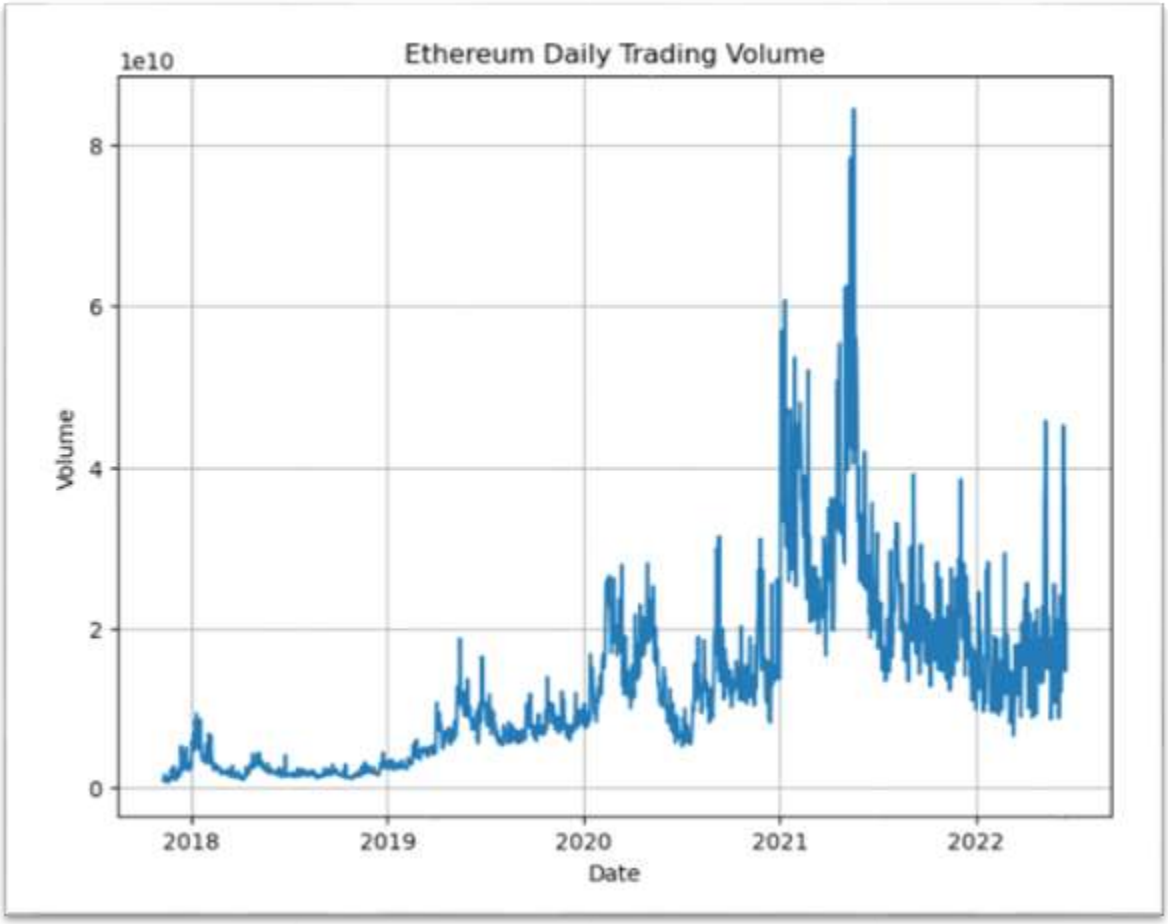


Daily Trading Volume

We also visualize the trading volume

```
[6]: # Plot the 'Volume' over time
     plt.figure(figsize=(8,6))
     plt.plot(df_selected.index, df_selected['Volume'])
     plt.xlabel('Date')
     plt.ylabel('Volume')
     plt.title('Ethereum Daily Trading Volume')
     plt.grid(True)
     plt.show()
```

Graph:



Stationarity Testing

- ADF Test: Conducted to determine if the series is stationary.
- Applied to both raw data and differenced series.
- Visual comparisons of both versions are also provided.

Code Implementation

```
[7]: from statsmodels.tsa.stattools import adfuller

def print_adf_results(series, label):
    result = adfuller(series)

    print(f"\n ADF Test Results for {label}")
    print("="*40)
    print(f" ADF Statistic      : {result[0]:.6f}")
    print(f" p-value            : {result[1]:.6f}")
    print(" Critical Values    :")
    for key, value in result[4].items():
        print(f"   - {key}: {value:.3f}")

    if result[1] <= 0.05:
        print(" The series is likely **stationary** (p <= 0.05).")
    else:
        print(" The series is likely **non-stationary** (p > 0.05).")
```

We apply the ADF test to both the raw Close Price and the first difference

-

```
# Apply ADF test on original Close prices
print_adf_results(df_selected['Close'], label="Raw Close Prices")

# First difference of Close prices
diff_series = df_selected['Close'].diff().dropna()
print_adf_results(diff_series, label="1st Difference of Close Prices")
```

- Visualizing Raw and Differenced Data
- We visualize both the raw and differenced time series.

```
[8]: import matplotlib.pyplot as plt

     # Create subplots
     fig, ax = plt.subplots(2, 1, figsize=(14, 6), sharex=True)

     # Plot raw close prices
     ax[0].plot(df_selected['Close'], color='crimson')
     ax[0].set_title('Raw Close Prices')
     ax[0].set_facecolor('white')  # Reset subplot background

     # Plot 1st difference of close prices
     ax[1].plot(df_selected['Close'].diff(), color='blue')
     ax[1].set_title('1st Difference of Close Prices')
     ax[1].set_facecolor('white')  # Reset subplot background

     # Optional: reset the entire figure background (if you had set it previously)
     fig.patch.set_facecolor('white')

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

-
- Graph

-
-
-

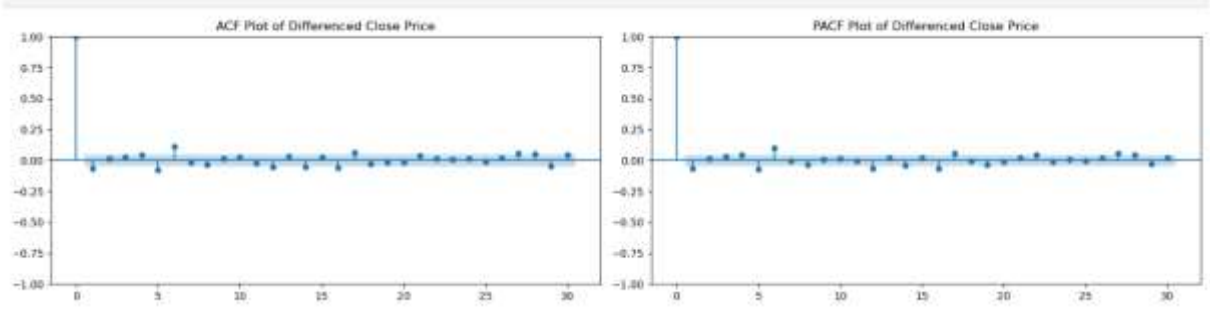
ACF & PACF Plots

Autocorrelation and partial autocorrelation plots guide the selection of ARIMA parameters.

Code Implementation:

```
[9]: from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Calculate the first difference of the 'Close' price
diff_series = df_selected['Close'].diff().dropna()

# Generate ACF plot
fig, axes = plt.subplots(1, 2, figsize=(16, 4))
plot_acf(diff_series, lags=30, ax=axes[0], title="ACF Plot of Differenced Close Price")

# Generate PACF plot
plot_pacf(diff_series, lags=30, ax=axes[1], title="PACF Plot of Differenced Close Price")

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

Graph



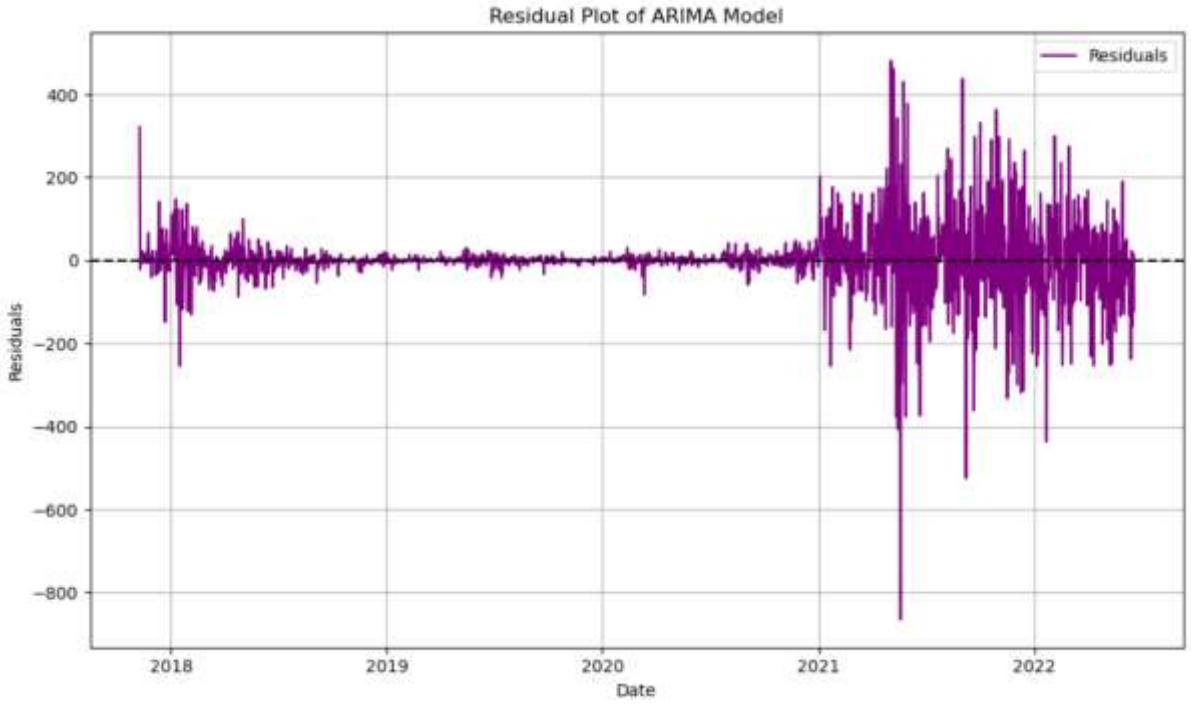
ARIMA Model Development

An ARIMA model is built using selected parameters.
Forecasting relies on patterns found in historical data.

Code Implementation:

```python
# Plot residuals
residuals = valid_df['Close'] - valid_df['Predicted']
plt.figure(figsize=(10, 6))
plt.plot(valid_df.index, residuals, label='Residuals', color='purple')
plt.axhline(y=0, color='black', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.title('Residual Plot of ARIMA Model')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

Graph

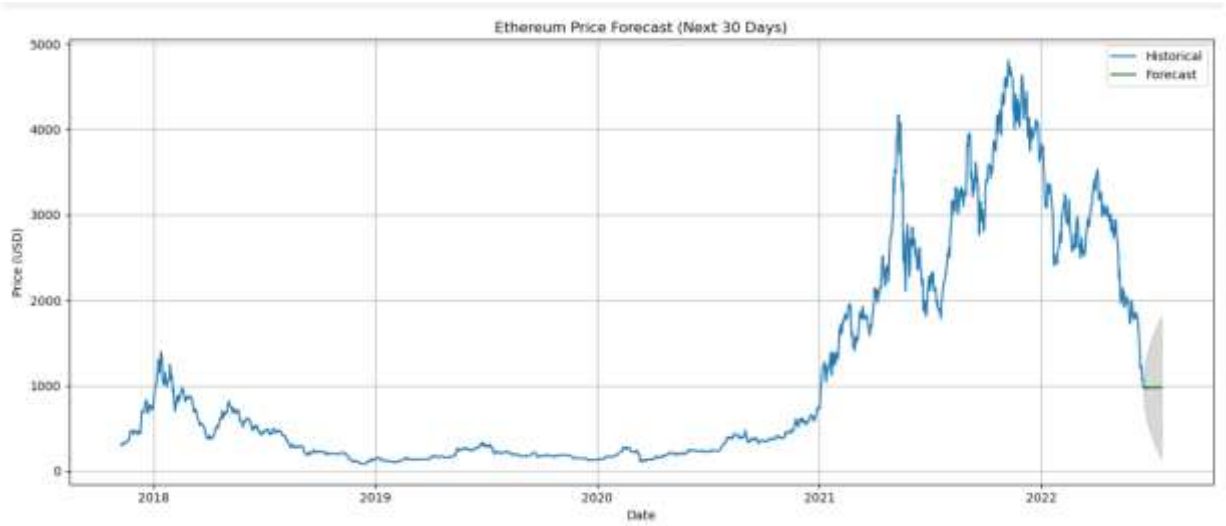


Forecasting

Future ETH prices are projected using the trained ARIMA model. These forecasts consider time-based structures in the data.

```python
# Plot
plt.figure(figsize=(14, 6))
plt.plot(df_selected['Close'], label='Historical')
plt.plot(forecast_df['Forecast'], label='Forecast', color='green')
plt.fill_between(forecast_df.index, forecast_df['Lower CI'], forecast_df['Upper CI'], color='gray', alpha=0.3)
plt.title('Ethereum Price Forecast (Next 30 Days)')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True)

# White background
fig = plt.gcf()
fig.patch.set_facecolor('white')
ax = plt.gca()
ax.set_facecolor('white')

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

Graph



Results

- The model performed well for short-term forecasts, with residuals showing no major anomalies.
- Forecasts had smooth trends with meaningful confidence intervals.

- Evaluation metrics like MAE and RMSE confirmed good performance.
- However, the model had limitations in capturing sudden market spikes.

```
[15]: # Calculate RMSE
rmse = np.sqrt(mean_squared_error(valid_df['Close'], valid_df['Predicted']))
print(f'RMSE: {rmse:.2f}')

# Calculate MAPE
mape = np.mean(np.abs((valid_df['Close'] - valid_df['Predicted']) / valid_df['Close'])) * 100
print(f'MAPE: {mape:.2f}%')
```

```
RMSE: 80.22
MAPE: 3.67%
```

Conclusion

ARIMA proves useful for short-term ETH forecasting, offering dependable predictions based on past behavior. However, due to high market volatility, forecasts should be used cautiously and ideally combined with more complex models like LSTM or SARIMA.

For analysts and algorithmic traders, ARIMA serves as a strong foundational model.

Future Work

To improve forecasting, future efforts could explore:

- Incorporating seasonality and external factors
- Combining ARIMA with deep learning or hybrid models
- Predicting volatility
- Building real-time forecasting systems

- Enhancing model optimization
- Developing multi-step and probabilistic forecasts.