Time Series Analysis of Ethereum (ETH/USDT) Market Projections using ARIMA

Project Documentation

Table of Contents

Introduction

This project performs a time series analysis on Ethereum (ETH/USDT) price data using the Autoregressive Integrated Moving Average (ARIMA) methodology. The analysis aims to:

- Understand historical price patterns and trends in Ethereum
- Test for stationarity in the price data
- Develop an ARIMA model to capture the underlying patterns
- Evaluate the model's performance
- Generate forecasts for future price movements

ARIMA models are powerful tools for analyzing time series data and making forecasts. They combine three components:

- **Autoregressive (AR)**: Using past values to predict future values
- **Integrated (I)**: Differencing the data to make it stationary
- **Moving Average (MA)**: Using past forecast errors in the model

Project Setup

Required Libraries

The following Python libraries are required for this project:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
```

Data Source

The project uses historical Ethereum price data from a CSV file named "ETH-USD.csv", which contains daily price information including Open, High, Low, Close prices, and trading Volume.

Data Loading and Preparation

Loading the Data

The first step is to load the data from the CSV file, set the Date column as the index, and sort the data by date:

```python
# Load the data
df = pd.read_csv('ETH-USD.csv', parse_dates=['Date'])

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

Data Preparation

Next, we select the relevant columns for analysis and check for missing values:

python

```python
# 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.")
```

Exploratory Data Analysis

Summary Statistics

We calculate and display summary statistics for the selected columns:

python

```python
# 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
print("\nData Types:\n", df_selected.dtypes)
```

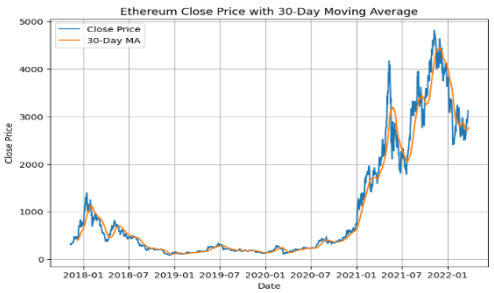
Visualizations

Close Price with 30-Day Moving Average

We visualize the Close price with a 30-day moving average to identify trends:

python

```python
# Plot the 'Close' price with a 30-day moving average
plt.figure(figsize=(8,6))
plt.plot(df_selected.index, df_selected['Close'], label='Close Price')
plt.plot(df_selected.index, df_selected['Close'].rolling(window=30).mean(), label='30-Day MA')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.title('Ethereum Close Price with 30-Day Moving Average')
plt.legend()
plt.grid(True)
plt.show()
```

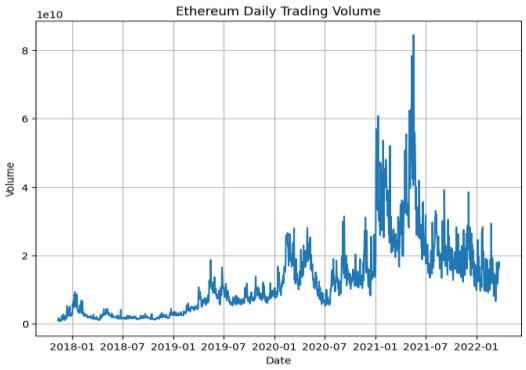


Daily Trading Volume

We also visualize the daily trading volume:

python

```
# 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()
```



Stationarity Testing

Augmented Dickey-Fuller (ADF) Test

We perform the ADF test to check if the time series is stationary:

python

```
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 prices and the first difference:

python

```python
# 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

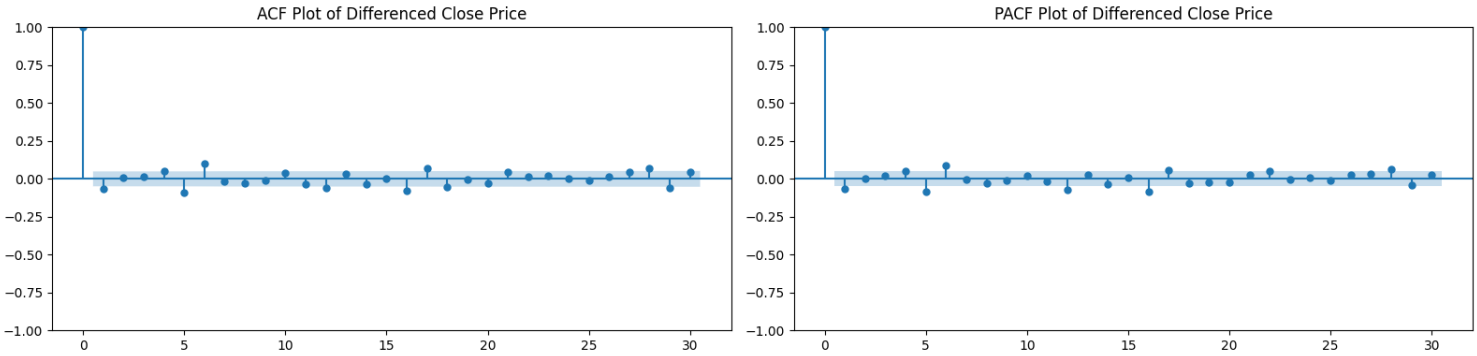
We visualize both the raw and differenced time series:

python

```python
plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
plt.plot(df_selected['Close'], color='crimson')
plt.title("Raw Close Prices", fontsize=14, fontweight='bold')
plt.grid(True)

plt.subplot(2,1,2)
plt.plot(diff_series, color='darkgreen')
plt.title("1st Difference of Close Prices", fontsize=14, fontweight='bold')
plt.grid(True)
plt.tight_layout()
plt.show()
```

ACF and PACF Analysis

We generate Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots to help determine the ARIMA parameters:



python

```python
# 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()
```

The ACF and PACF plots help determine the order of the ARIMA model:

- The significant lags in the PACF plot suggest the AR order (p)
- The significant lags in the ACF plot suggest the MA order (q)
- The differencing needed to achieve stationarity determines thel order (d)

ARIMA Model Development

Based on the stationarity tests and ACF/PACF plots, we develop an ARIMA model:

python

```python
# Use p=5 and q=1 as an example, d=1 from the ADF test
model = ARIMA(df_selected['Close'], order=(5, 1, 1))
model_fit = model.fit()
print(model_fit.summary())
```

The ARIMA(5,1,1) model was selected where:

- p=5: 5 autoregressive terms
- d=1: 1st differencing to achieve stationarity
- q=1: 1 moving average term

Model Evaluation

Calculating Performance Metrics

We evaluate the model by calculating RMSE (Root Mean Square Error) and MAPE (Mean Absolute Percentage Error):

python

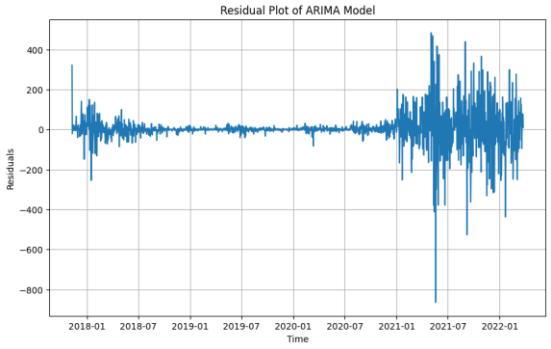
```python
# Predict on the training data
predictions = model_fit.predict(start=0, end=len(df_selected['Close']) - 1)

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(df_selected['Close'], predictions))
print(f'RMSE: {rmse}')

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

Residual Analysis

We analyze the residuals to check for any remaining patterns:

python

```python
# Plot residuals
residuals = df_selected['Close'] - predictions
plt.figure(figsize=(10, 6))
plt.plot(residuals)
plt.xlabel('Time')
plt.ylabel('Residuals')
plt.title('Residual Plot of ARIMA Model')
plt.grid(True)
plt.show()
```



Forecasting

We forecast Ethereum prices for the next 30 days:

python

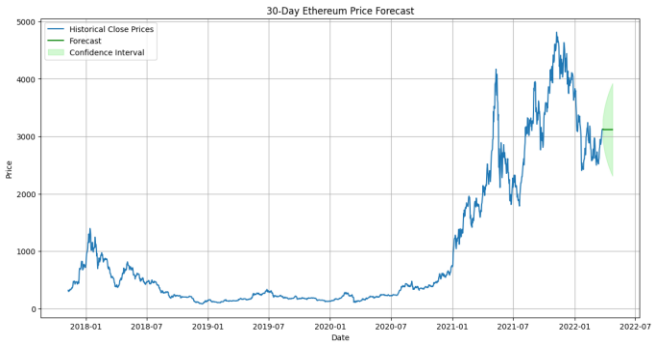
```python
# Forecast Ethereum prices for the next 30 days
forecast = model_fit.forecast(steps=30)

# Get confidence intervals
conf_int = model_fit.get_forecast(steps=30).conf_int()

# Print the forecast and confidence intervals
print("Forecast:\n", forecast)
print("\nConfidence Intervals:\n", conf_int)
```

Visualizing the Forecast

We visualize the forecast with confidence intervals:

python

```python
# Plot the forecast
plt.figure(figsize=(14, 7))
plt.plot(df_selected['Close'], label='Historical Close Prices')
plt.plot(forecast.index, forecast, label='Forecast', color='green')
plt.fill_between(forecast.index, conf_int.iloc[:, 0], conf_int.iloc[:, 1],
                 color='lightgreen', alpha=0.4, label='Confidence Interval')
plt.title('30-Day Ethereum Price Forecast')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



Results and Conclusions

Key Findings

1. **Stationarity**:
   - The raw Ethereum closing price data is non-stationary (p-value = 0.878)
   - First differencing makes the series stationary (p-value= 0.000)
2. **Model Performance**:
   - RMSE: 78.78
   - MAPE: 3.67%
   - These metrics indicate the model has reasonable predictive accuracy

3. **Forecast**:
    - The 30-day forecast suggests prices around 3116-3120
    - The confidence intervals are wide, ranging from approximately 2300 to 3800
    - This reflects the inherent volatility and uncertainty in cryptocurrency markets
4. **Model Diagnostics**:
    - The model summary indicates statistically significant coefficients for most AR and MA terms
    - The Jarque-Bera test indicates that the errors do not follow a normal distribution
    - Heteroskedasticity is present in the residuals

Limitations

1. **Non-normal residuals**: The model's errors do not follow a normal distribution, which may affect the validity of the confidence intervals
2. **Heteroskedasticity**: The variance of errors changes over time, which could affect the model's predictive accuracy
3. **Wide confidence intervals**: The forecast has wide confidence intervals, indicating high uncertainty

Future Work

1. **Parameter optimization**: Further tune the ARIMA parameters to improve model fit
2. **Alternative models**: Explore other time series models like SARIMA (to capture seasonality), GARCH (to model volatility), or Prophet
3. **Feature engineering**: Include external factors like market sentiment, Bitcoin prices, or macroeconomic indicators
4. **Validation strategy**: Implement walk-forward validation to test the model's performance over different time periods
5. **Address heteroskedasticity**: Implement GARCH models to handle changing volatility

---