Time series Analysis Project

Yashika

A **time series** is a sequence of data points recorded or measured at successive points in time, usually at uniform intervals. Time series analysis involves methods for analyzing these data points to extract meaningful statistics and identify characteristics of the data.

Components of Time Series

1. **Trend Component (T)**: Represents the general direction or pattern of the data over a long period.
2. **Seasonal Component (S)**: Captures the periodic fluctuations within a specific period (e.g., quarterly sales).
3. **Cyclic Component (C)**: Refers to long-term oscillations that are not necessarily periodic.
4. **Irregular Component (I)**: Random or residual variations after accounting for trend, seasonality, and cyclic components.

Data has increasing / decreasing width and height of the seasonal patterns or peaks. Trend of the data is non-linear.

Information about dataset

[Link to dataset (https://www.rbi.org.in/Scripts/BS_PressReleaseDisplay.aspx?prid=49901#)](https://www.rbi.org.in/Scripts/BS_PressReleaseDisplay.aspx?prid=49901#)

This dataset contain 3 columns :

1. Date of transaction
2. Volumne in lakhs
3. Value in INR Crores

Exploratory Data Analysis

In [ ]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from datetime import datetime
```

In [ ]:
```
# Loading dataset
df = pd.read_csv("/content/UPI transaction data.csv")
```

In [ ]:
```
# Let's see first 5 rows
df.head(5)
```

Out[4]:

	Date	Vol	Val
0	June 1, 2020	476.97	10413.11
1	June 2, 2020	476.78	9951.30
2	June 3, 2020	456.26	9622.38
3	June 4, 2020	463.05	9639.50
4	June 5, 2020	464.79	9539.52

In [ ]:
```
# Let's see last 5 rows
df.tail(5)
```

Out[5]:

	Date	Vol	Val
1486	June 26, 2024	4481.40	62349.06
1487	June 27, 2024	4504.87	62009.24
1488	June 28, 2024	4527.52	66808.91
1489	June 29, 2024	4755.43	70499.43
1490	June 30, 2024	4619.75	59293.86

In [ ]:
```
# Converting date to pandas datetime
df['Date'] = pd.to_datetime(df['Date'])
# Now index will be Date
df.set_index('Date', inplace=True)
```

In [ ]:
```
# Checking shape of dataset
df.shape
```

Out[7]:
```
(1491, 2)
```

In [ ]:
```
# Checking null values
df.isnull().sum()
```

Out[8]:

	0
Vol	0
Val	0

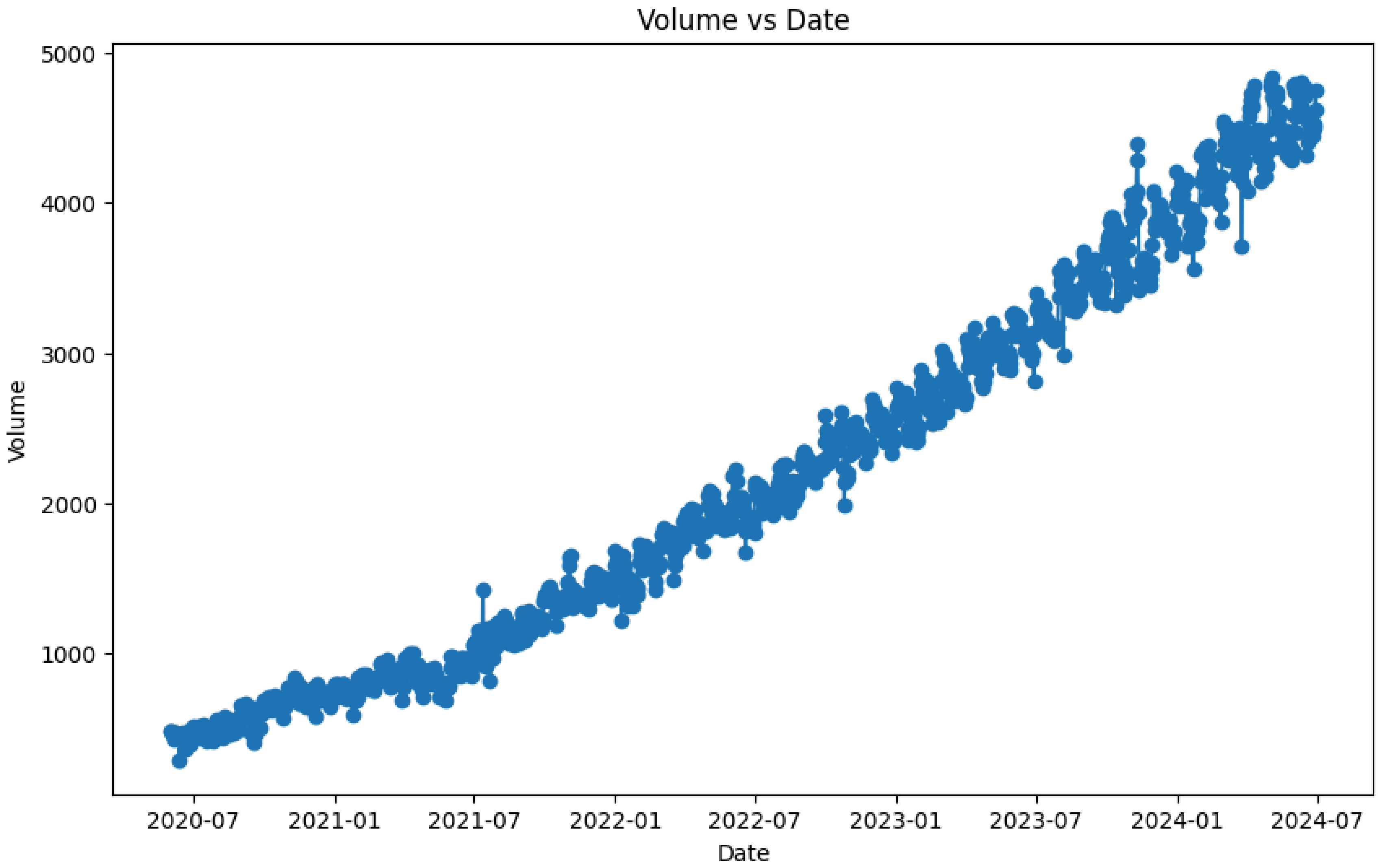
dtype: int64

In [ ]:
```
df.describe()
```

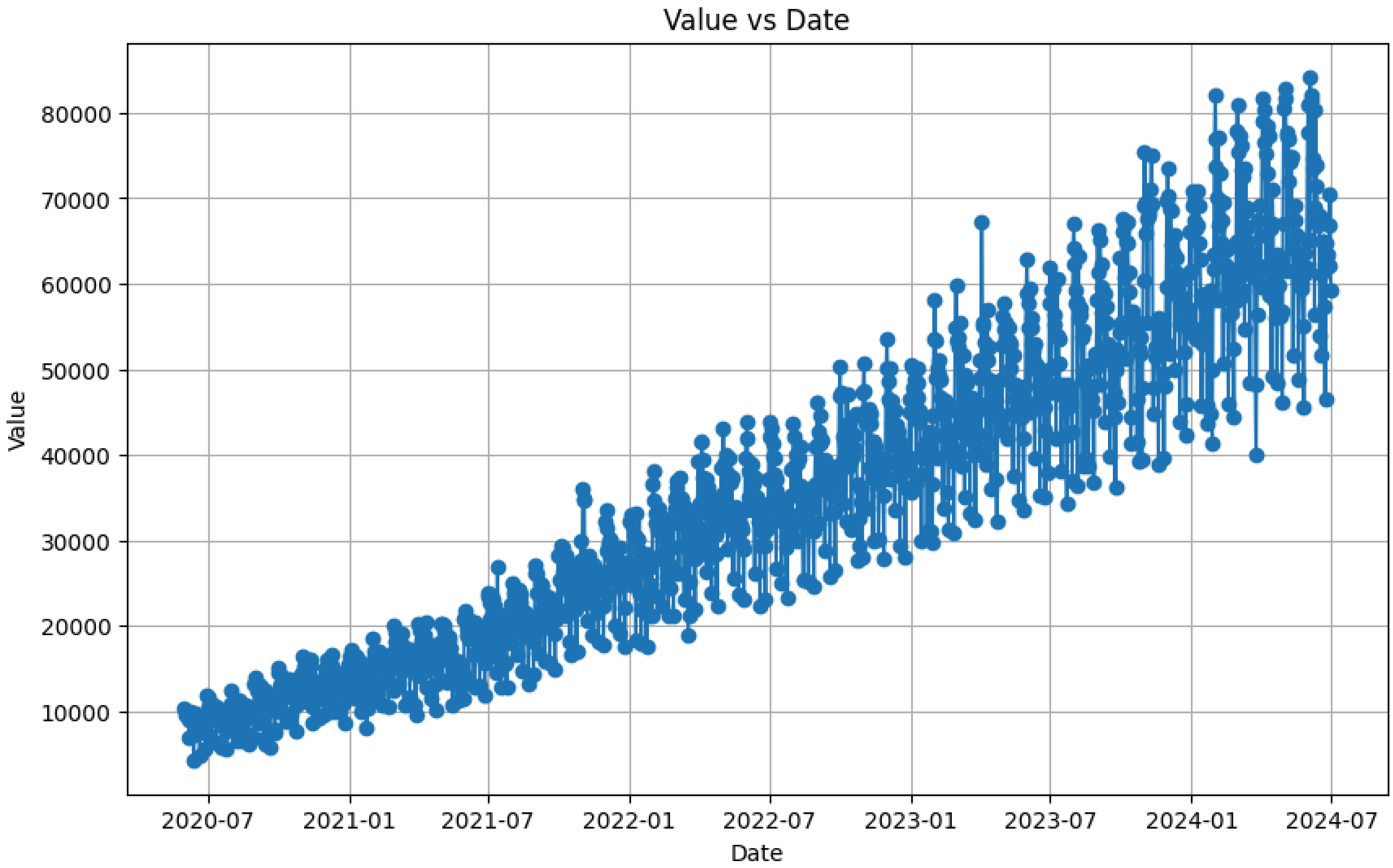
Out[9]:

	Vol	Val
count	1491.000000	1491.000000
mean	2160.439611	34927.247458
std	1266.795148	18632.921051
min	289.000000	4333.740000
25%	929.760000	18035.945000
50%	1972.090000	33487.990000
75%	3136.965000	48262.555000
max	4840.870000	84201.750000

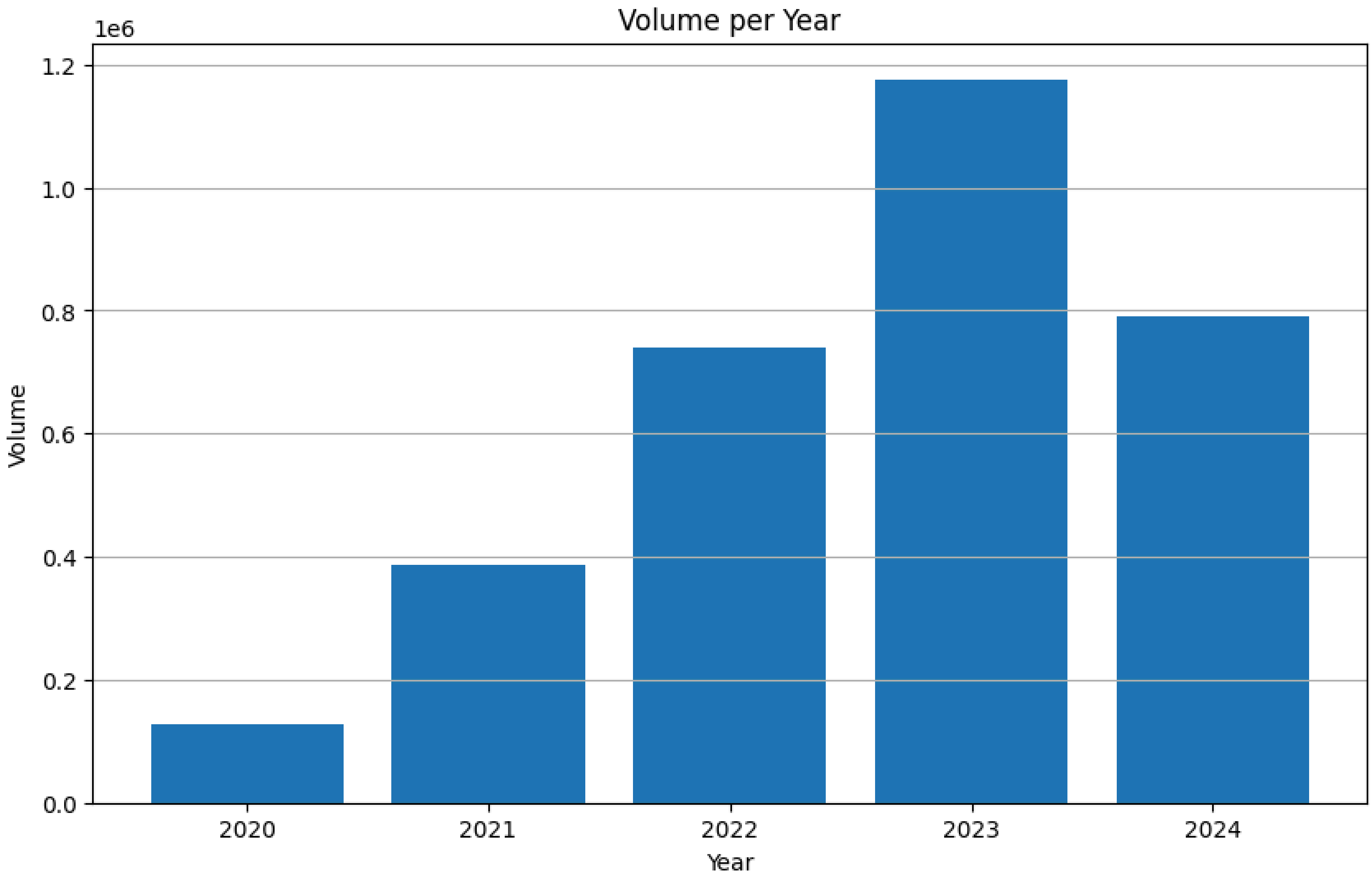
In [ ]:
```
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame with a datetime index and a 'Vol' column
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Vol'], marker='o', linestyle='-')

# Adding title and labels
plt.title('Volume vs Date')
plt.xlabel('Date')
plt.ylabel('Volume')

# Display the plot
plt.show()
```


Volume vs Date
Volume
5000
4000
3000
2000
1000
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
Date

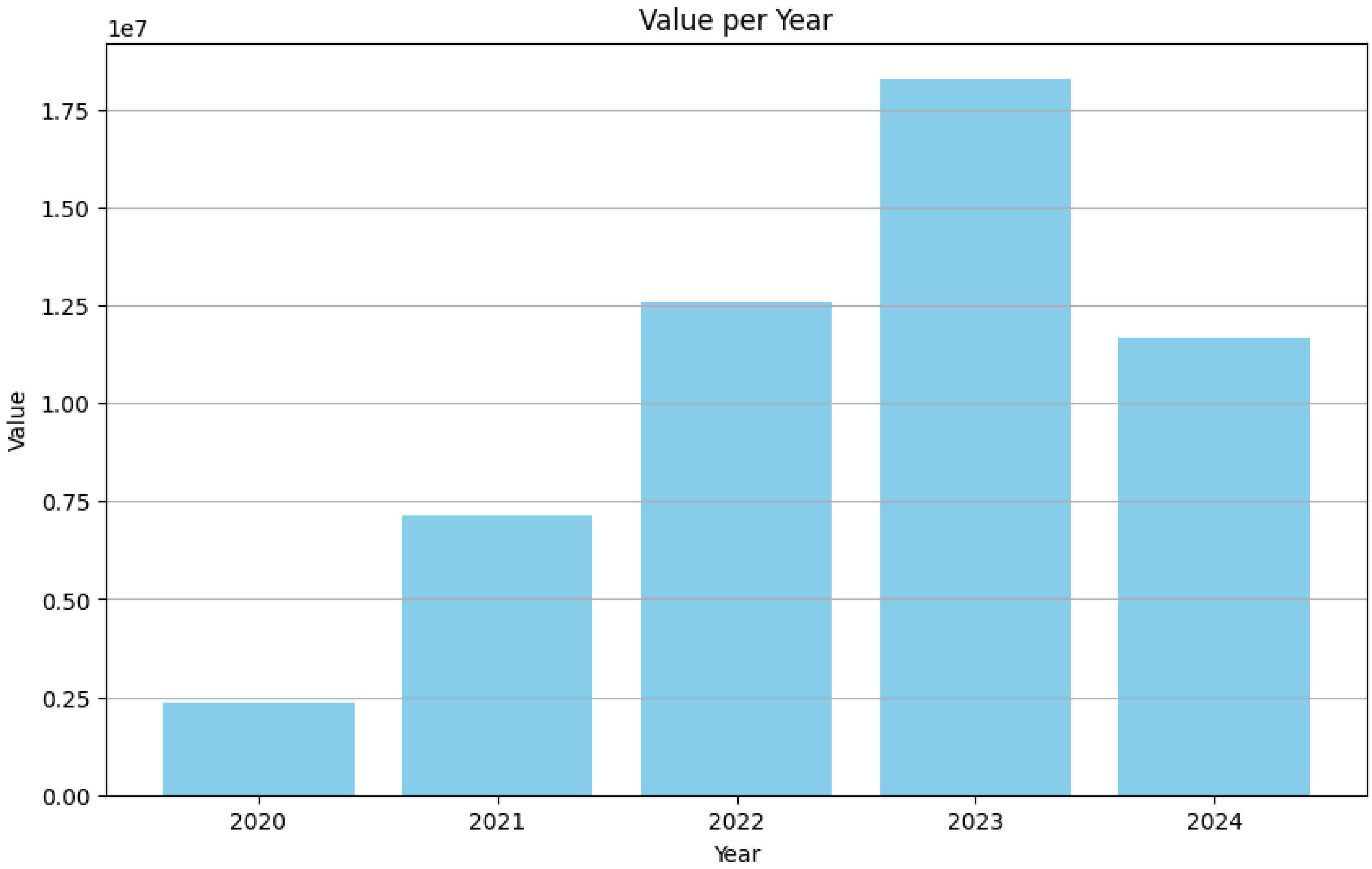
In [ ]:
```
import matplotlib.pyplot as plt

# Plotting the line graph of 'Val' vs 'date'
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Val'], marker='o')
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Value vs Date')
plt.grid(True)
plt.show()
```


Value vs Date
80000
70000
60000
50000
40000
30000
20000
10000
Value
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
Date

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

# Grouping the data by year and summing the 'Vol' column
df_year = df.groupby(df.index.year)['Vol'].sum()

# Plotting the bar graph of 'Volume' vs 'Year'
plt.figure(figsize=(10, 6))
plt.bar(df_year.index, df_year.values)
plt.xlabel('Year')
plt.ylabel('Volume')
plt.title('Volume per Year')
plt.grid(axis='y')

plt.show()
```


Volume per Year
1e6
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Volume
2020
2021
2022
2023
2024
Year

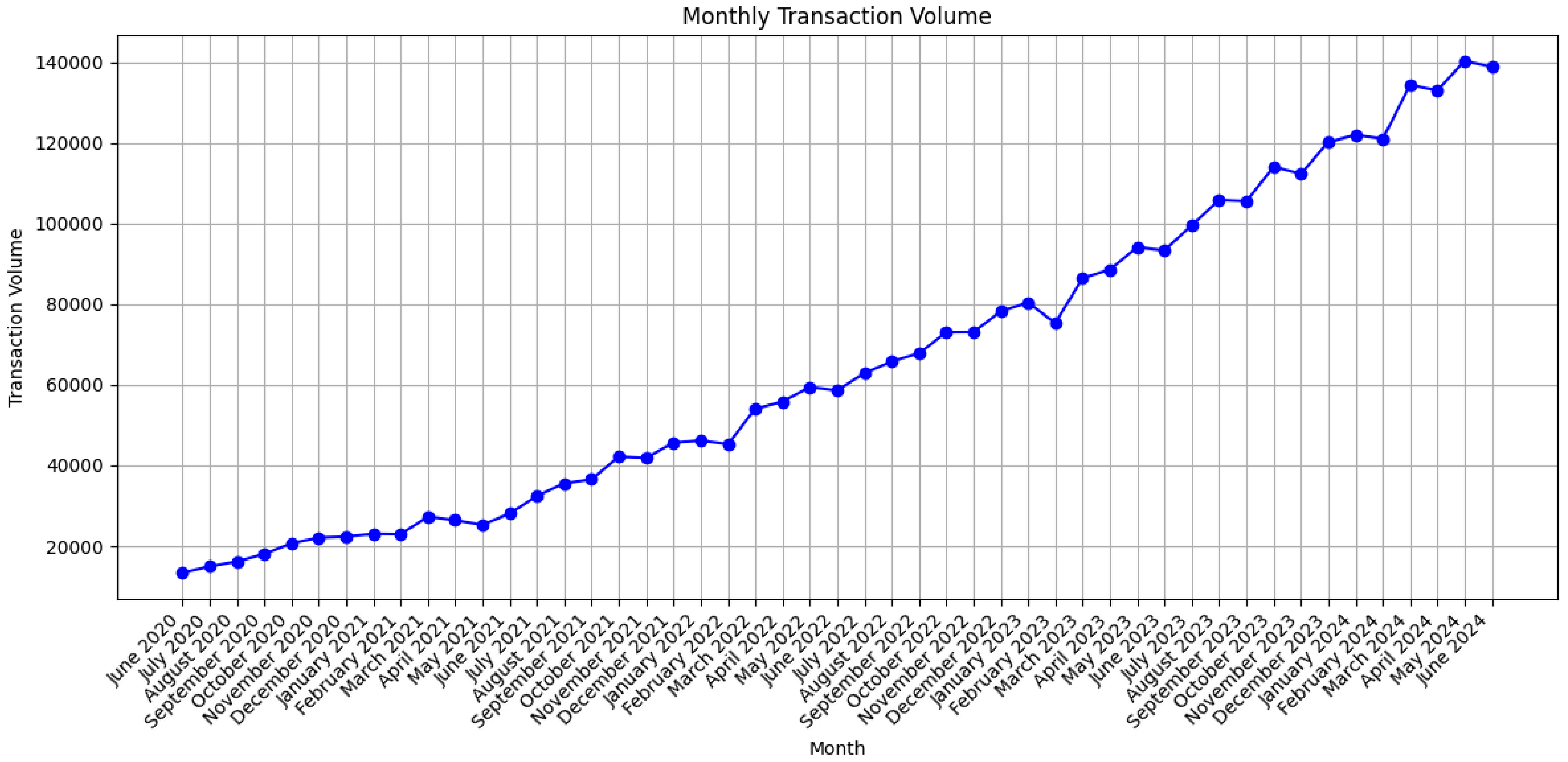
In [ ]:
```
import matplotlib.pyplot as plt

# Grouping the data by year and summing the 'Val' column
df_year = df.groupby(df.index.year)['Val'].sum()

# Plotting the bar graph of 'Value' vs 'Year'
plt.figure(figsize=(10, 6))
plt.bar(df_year.index, df_year.values, color='skyblue')
plt.xlabel('Year')
plt.ylabel('Value')
plt.title('Value per Year')
plt.grid(axis='y')

plt.show()
```



Monthly data

In [ ]:
```
# Grouping by month and summing the 'Vol' column
df_monthly = df.resample('M').sum()
```

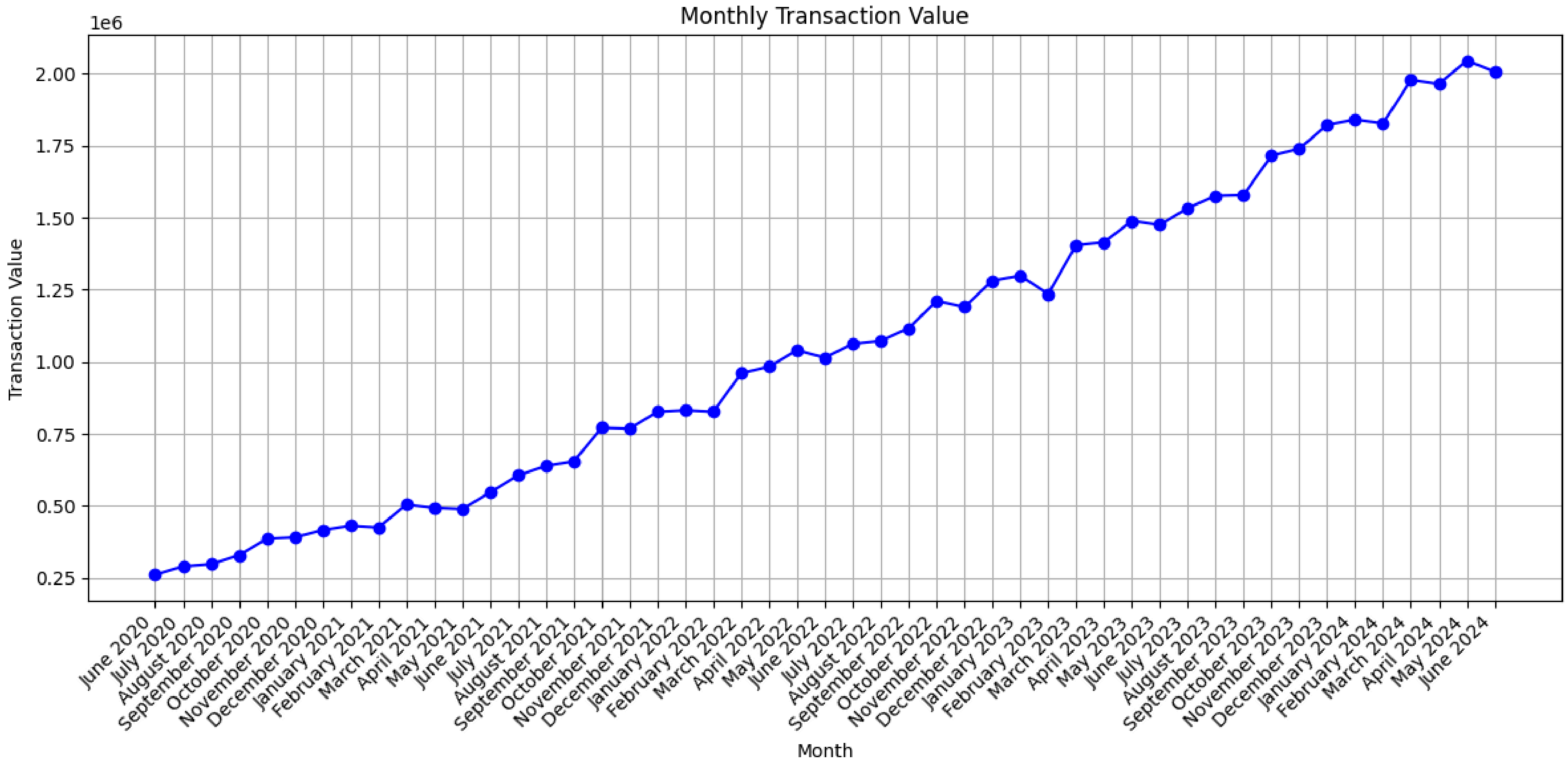
In [ ]:
```
# Plotting the line graph of 'Val' vs 'Month'
plt.figure(figsize=(12, 6))
plt.plot(df_monthly.index, df_monthly['Vol'], marker='o', color='blue')
plt.xticks(rotation=45, ha='right')  # Rotate month labels for better readability
plt.xlabel('Month')
plt.ylabel('Transaction Volume')
plt.title('Monthly Transaction Volume')
plt.grid(True)
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```



In [ ]:

```

```

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

# Plotting the line graph of 'Val' vs 'Month'
plt.figure(figsize=(12, 6))
plt.plot(df_monthly.index, df_monthly['Val'], marker='o', color='blue')
plt.xticks(rotation=45, ha='right')  # Rotate month labels for better readability
plt.xlabel('Month')
plt.ylabel('Transaction Value')
plt.title('Monthly Transaction Value')
plt.grid(True)
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```



Weekly data

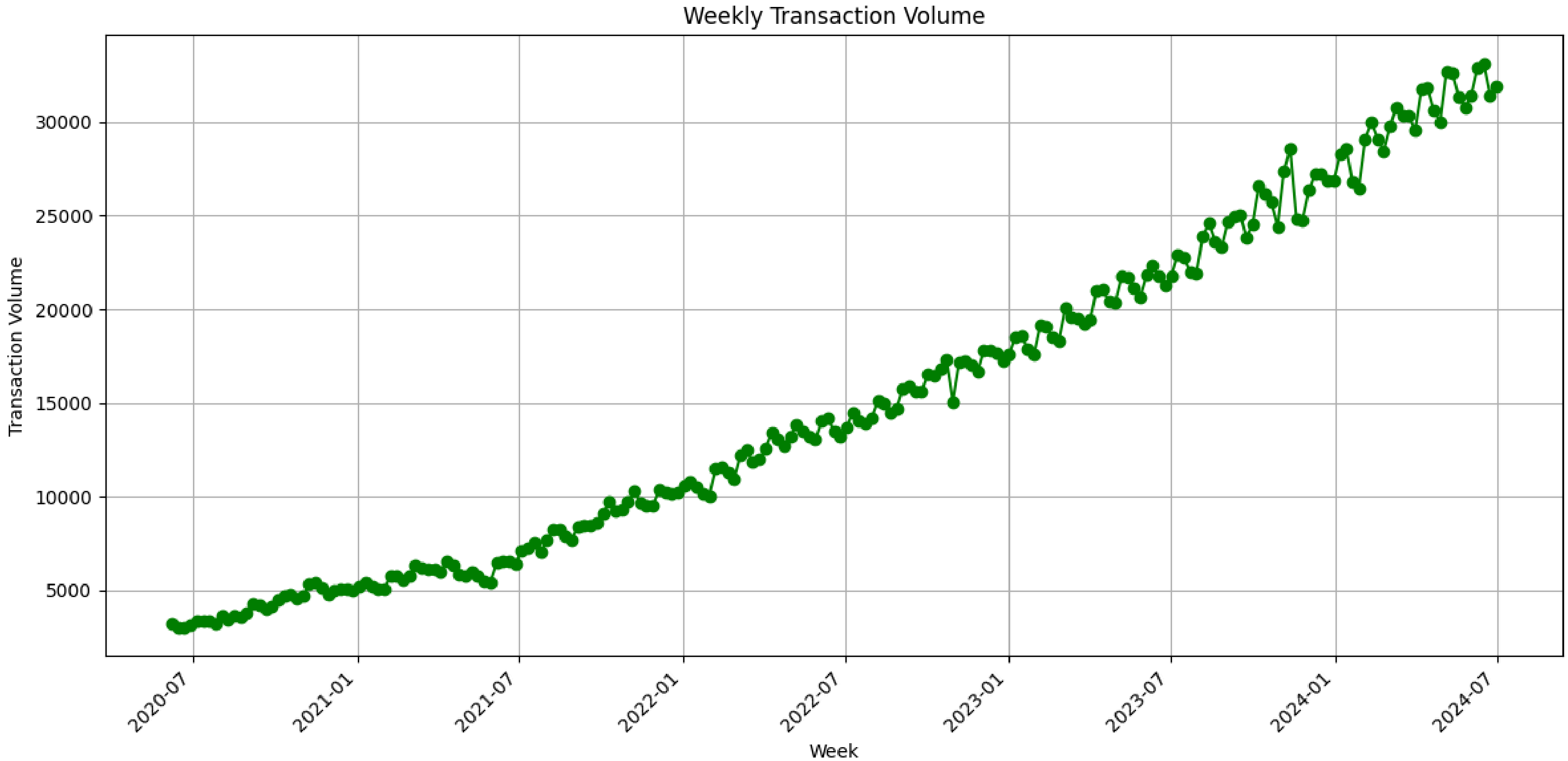
In [ ]:

```

```

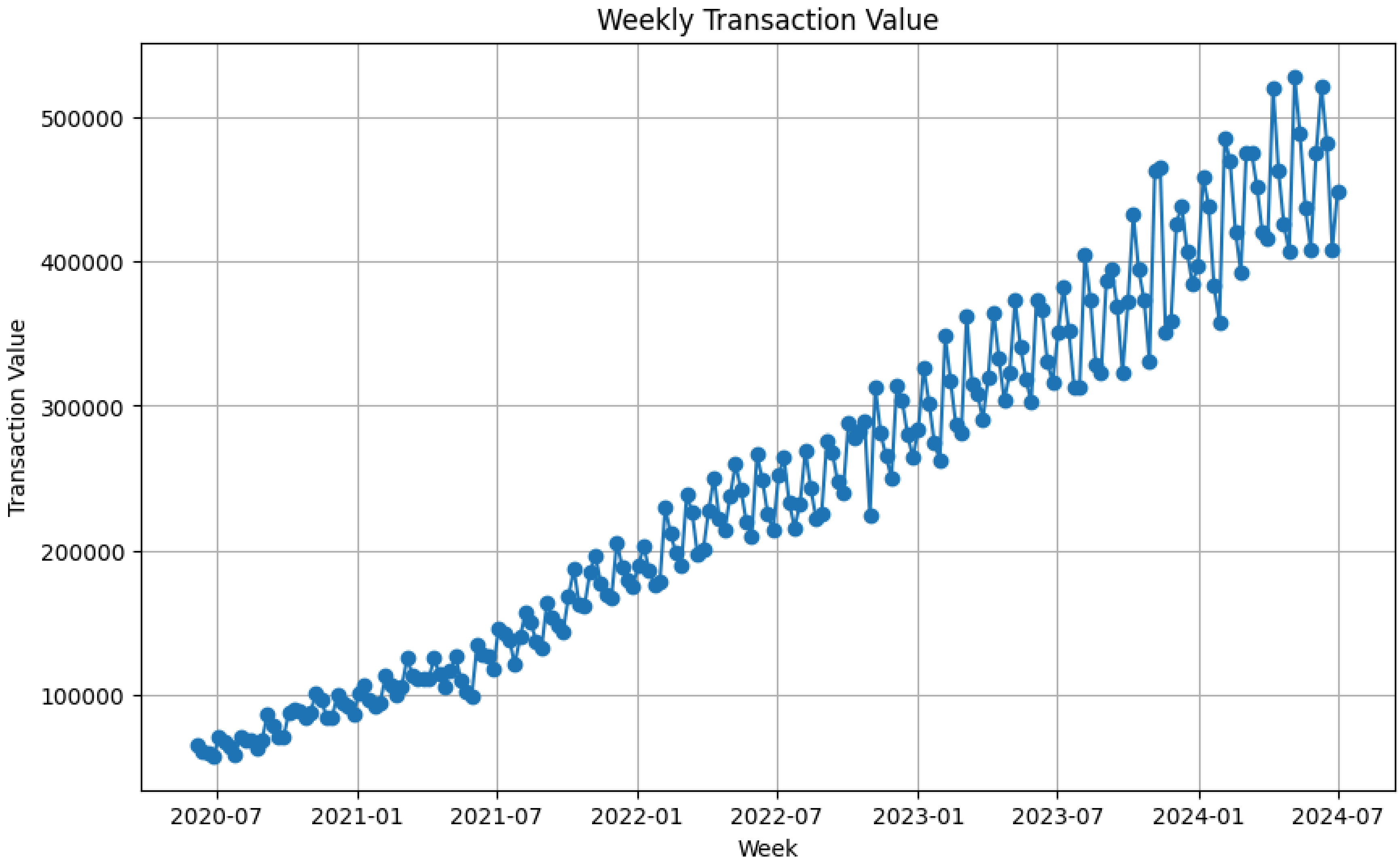
In [ ]:

```
# Grouping by week and summing the 'Vol' column
df_weekly = df.resample('W').sum()
```

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

# Plotting the line graph of 'Vol' vs 'Week'
plt.figure(figsize=(12, 6))
plt.plot(df_weekly.index, df_weekly['Vol'], marker='o', color='green')
plt.xticks(rotation=45, ha='right')  # Rotate week labels for better readability
plt.xlabel('Week')
plt.ylabel('Transaction Volume')
plt.title('Weekly Transaction Volume')
plt.grid(True)
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```

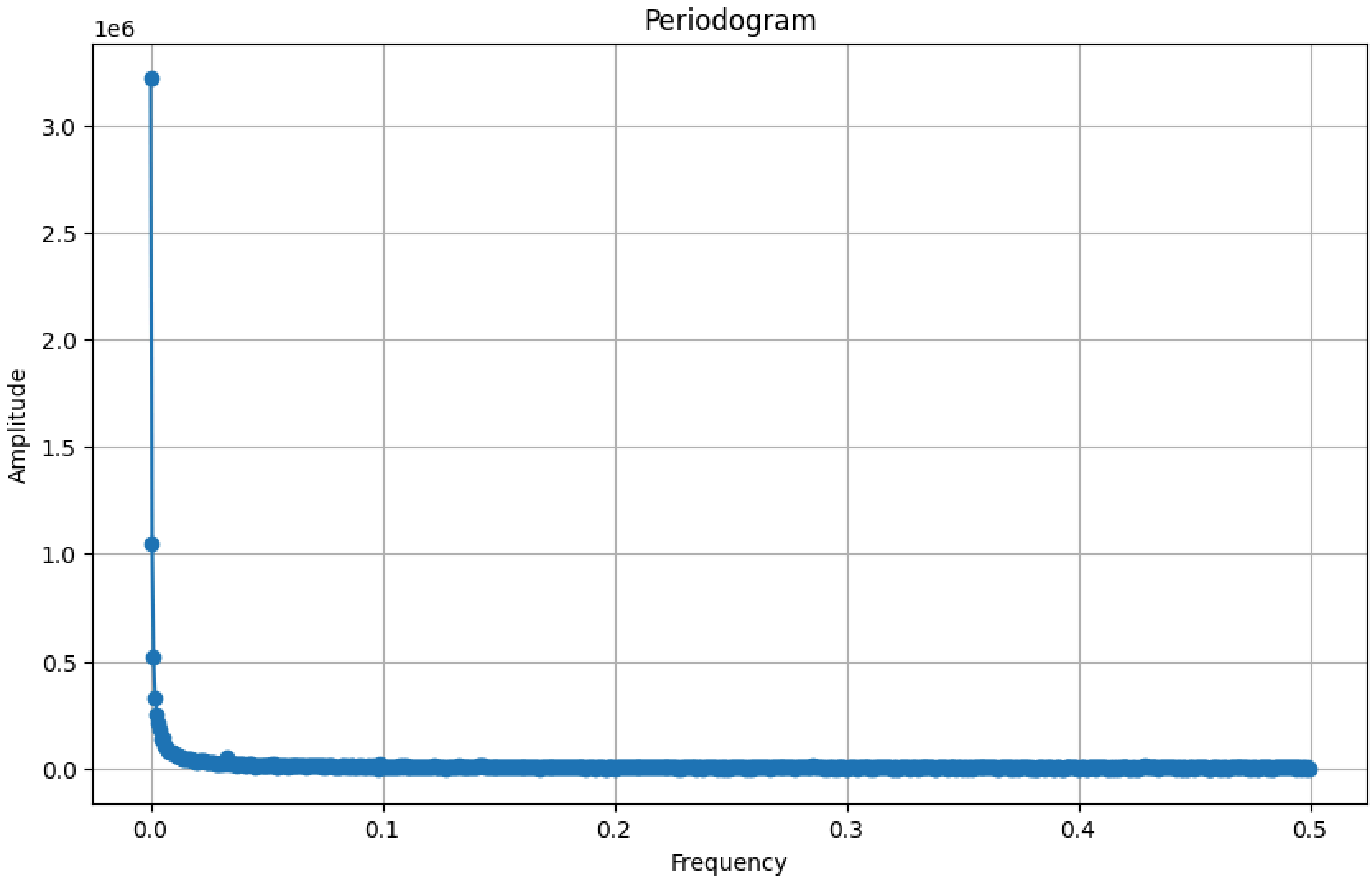


In [ ]:
```
import matplotlib.pyplot as plt

# Assuming df_weekly has a DateTime index
plt.figure(figsize=(10, 6))
plt.plot(df_weekly.index, df_weekly['Val'], marker='o', linestyle='-')

# Adding titles and labels
plt.title('Weekly Transaction Value')
plt.xlabel('Week')
plt.ylabel('Transaction Value')

# Display the plot
plt.grid(True)
plt.show()
```



Let's find the period of Seasonality

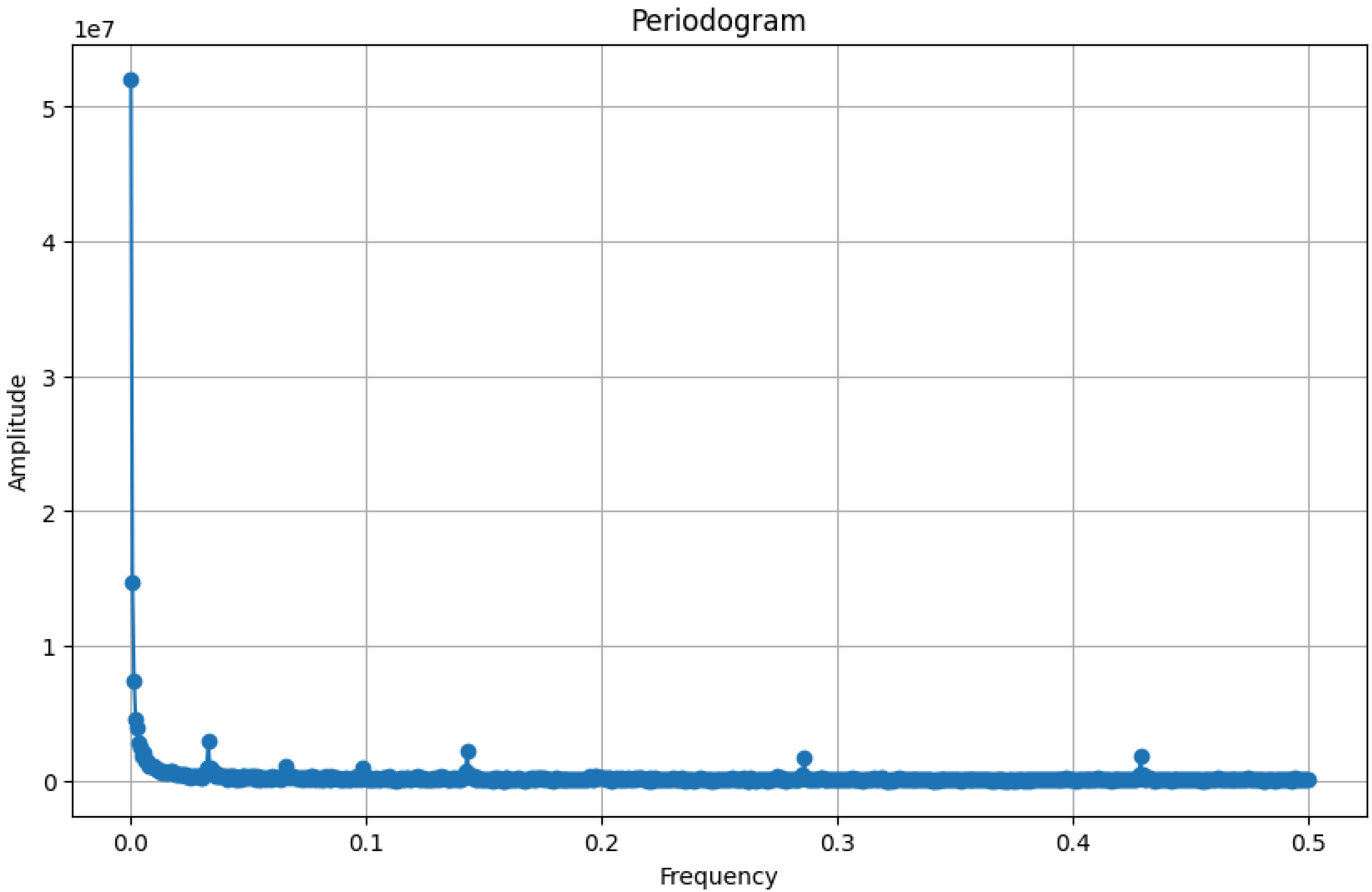
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# Perform Fourier Transform
fft_vals = np.fft.fft(df['Vol'])
fft_freqs = np.fft.fftfreq(len(df))

# Only keep the positive frequencies
positive_freqs = fft_freqs[fft_freqs >= 0]
positive_vals = np.abs(fft_vals[fft_freqs >= 0])

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(positive_freqs, positive_vals, marker='o', linestyle='-')

# Adding titles and labels
plt.title('Periodogram')
plt.xlabel('Frequency')
plt.ylabel('Amplitude')

# Display the plot
plt.grid(True)
plt.show()
```



Let's decompose our data to see Trend, Seasonality and Irregular components

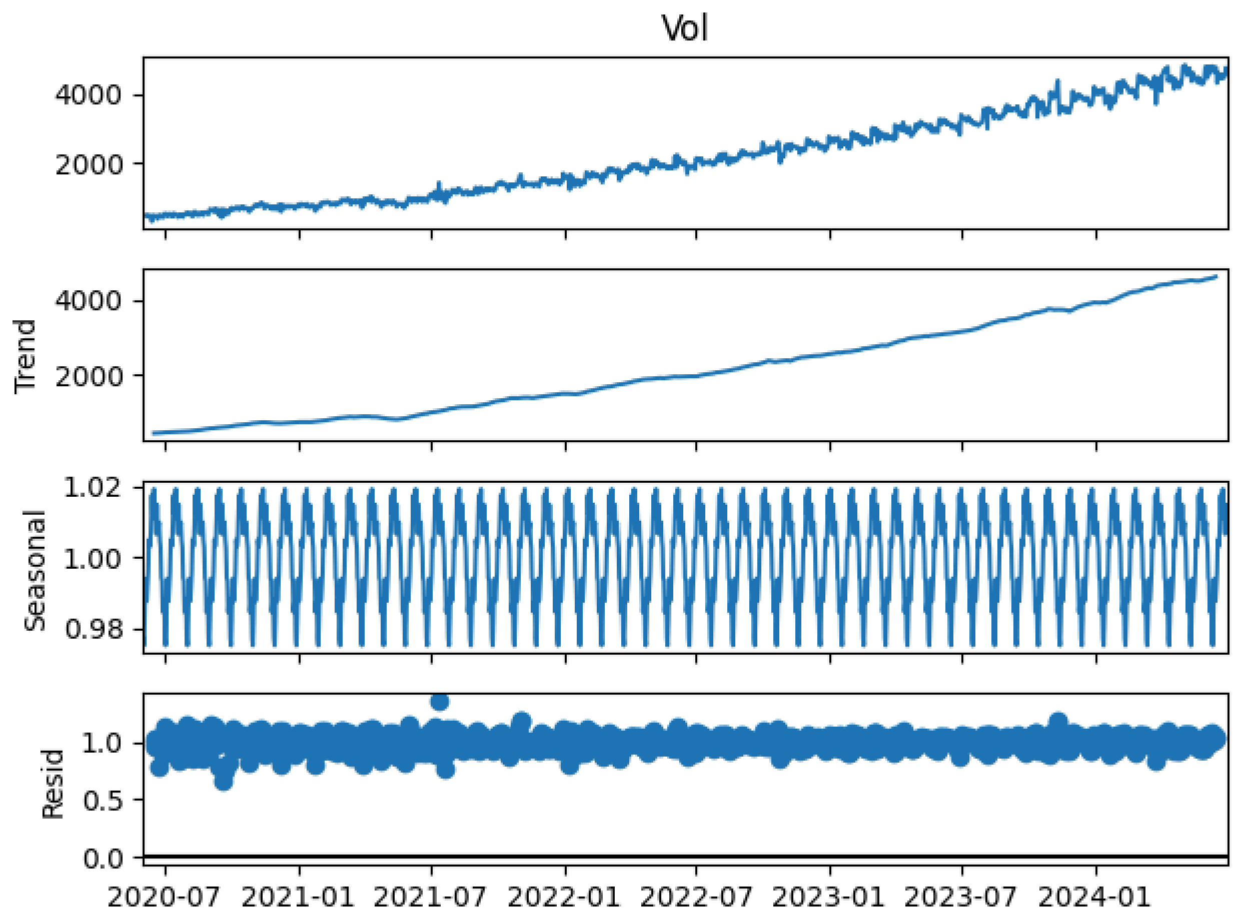
In [ ]:

```
# Perform Fourier Transform
fft_vals = np.fft.fft(df['Val'])
fft_freqs = np.fft.fftfreq(len(df))

# Only keep the positive frequencies
positive_freqs = fft_freqs[fft_freqs >= 0]
positive_vals = np.abs(fft_vals[fft_freqs >= 0])

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(positive_freqs, positive_vals, marker='o', linestyle='-')

# Adding titles and labels
plt.title('Periodogram')
plt.xlabel('Frequency')
plt.ylabel('Amplitude')

# Display the plot
plt.grid(True)
plt.show()
```

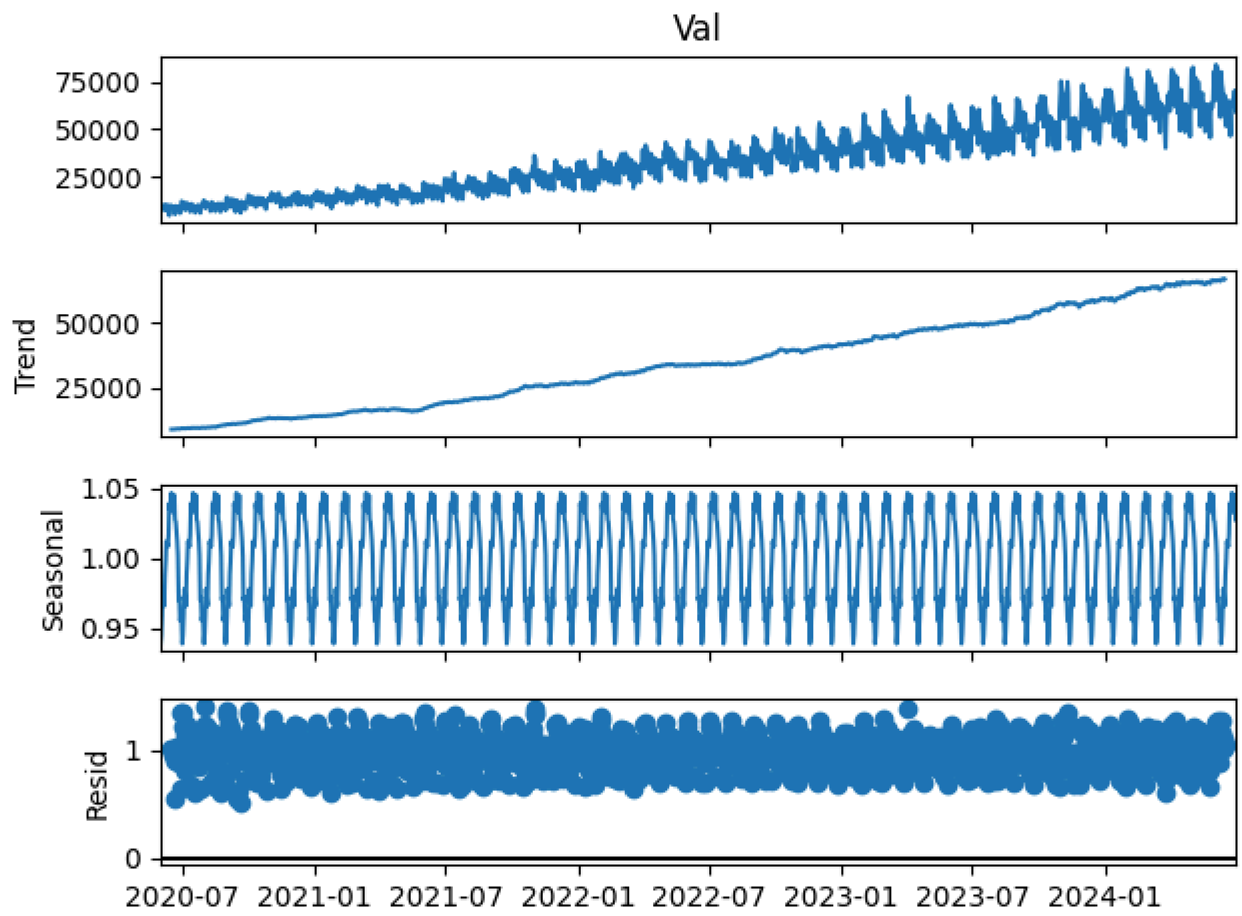


In [ ]:
```
import statsmodels.api as sm
import statsmodels.tsa.api as smt
```

In [ ]:
```
dec = sm.tsa.seasonal_decompose(df['Vol'],period = 30, model = 'multiplicative').plot()
plt.show()
```



In [ ]:
```
dec = sm.tsa.seasonal_decompose(df['Val'],period = 30, model = 'multiplicative').plot()
plt.show()
```



Defining important functions

Defining Dickey-Fuller test for Stationarity

In [ ]:
```
#Ho: It is non stationary
#H1: It is stationary
from statsmodels.tsa.stattools import adfuller

def adfuller_test(col):
    result=adfuller(col)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stati
```

In [ ]:
```
adfuller_test(df['Vol'])
```

```
ADF Test Statistic : 1.9914203313883863
p-value : 0.9986616986026696
#Lags Used : 24
Number of Observations Used : 1466
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

In [ ]:
```
adfuller_test(df['Val'])
```

```
ADF Test Statistic : -0.3564569648359322
p-value : 0.9171190012280932
#Lags Used : 24
Number of Observations Used : 1466
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

From test also it is now evident that data is not stationary. So we need to do differencing.

Differencing

In [ ]:
```
df['Vol_30st_diff'] = df['Vol'].diff(periods = 30)

adfuller_test(df['Vol_30st_diff'][30:])
```

```
ADF Test Statistic : -5.3345307157577775
p-value : 4.642310952725903e-06
#Lags Used : 23
Number of Observations Used : 1437
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is s
tationary
```

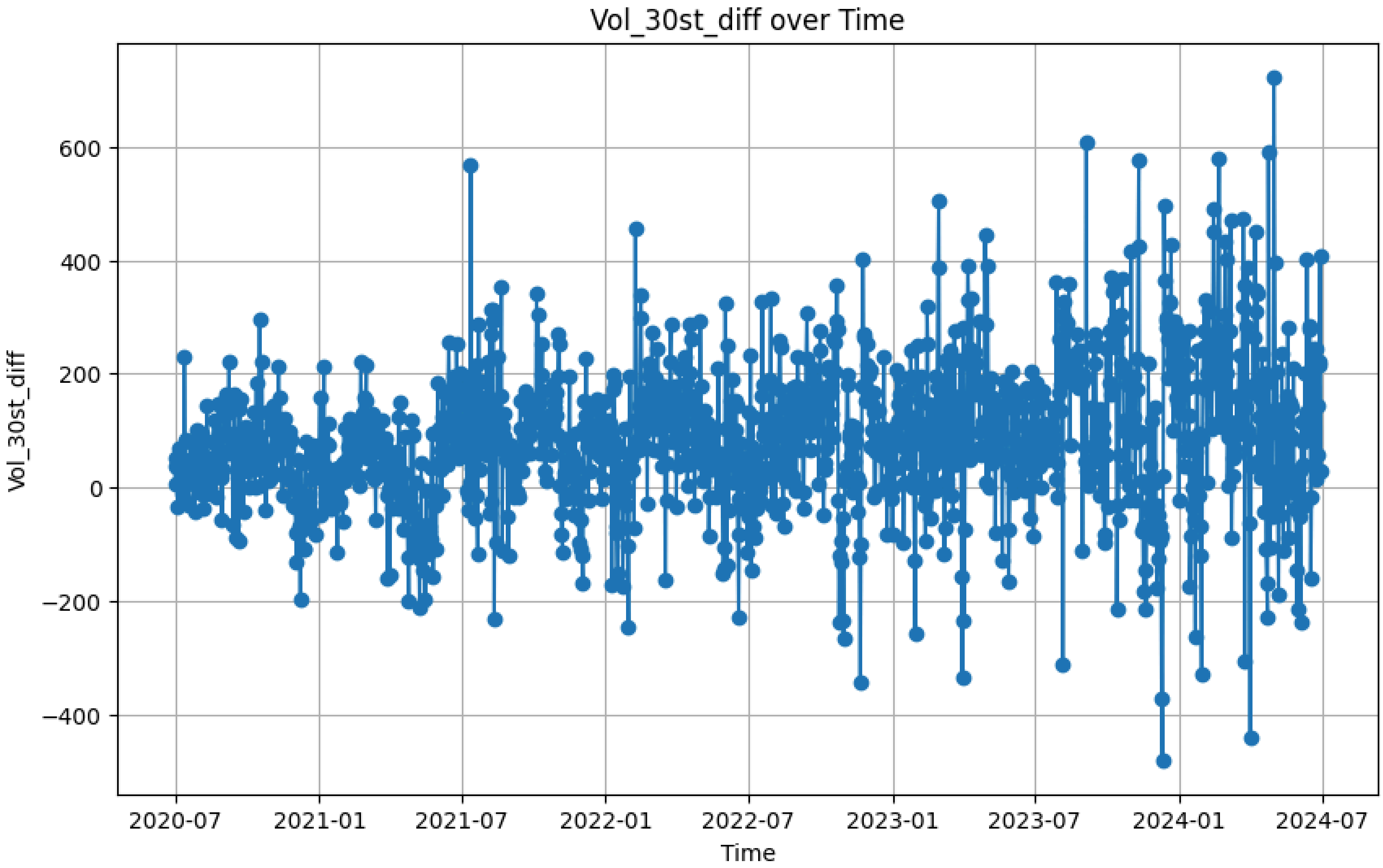
In [ ]:
```
df['Vol_30st_diff'][30:].head()
```

Out[39]:

	Vol_30st_diff
Date	
2020-07-01	38.21
2020-07-02	5.18
2020-07-03	51.15
2020-07-04	35.72
2020-07-05	-32.27

dtype: float64

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

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Vol_30st_diff'], marker='o', linestyle='-')

# Adding titles and labels
plt.title('Vol_30st_diff over Time')
plt.xlabel('Time')
plt.ylabel('Vol_30st_diff')

# Display the plot
plt.grid(True)
plt.show()
```


Vol_30st_diff over Time
Vol_30st_diff
600
400
200
0
−200
−400
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07
Time

In [ ]:
```
df['Val_30st_diff'] = df['Val'].diff(periods = 30)

adfuller_test(df['Val_30st_diff'][30:])
```

```
ADF Test Statistic : -6.3194666358845435
p-value : 3.09165280616763e-08
#Lags Used : 22
Number of Observations Used : 1438
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is s
tationary
```

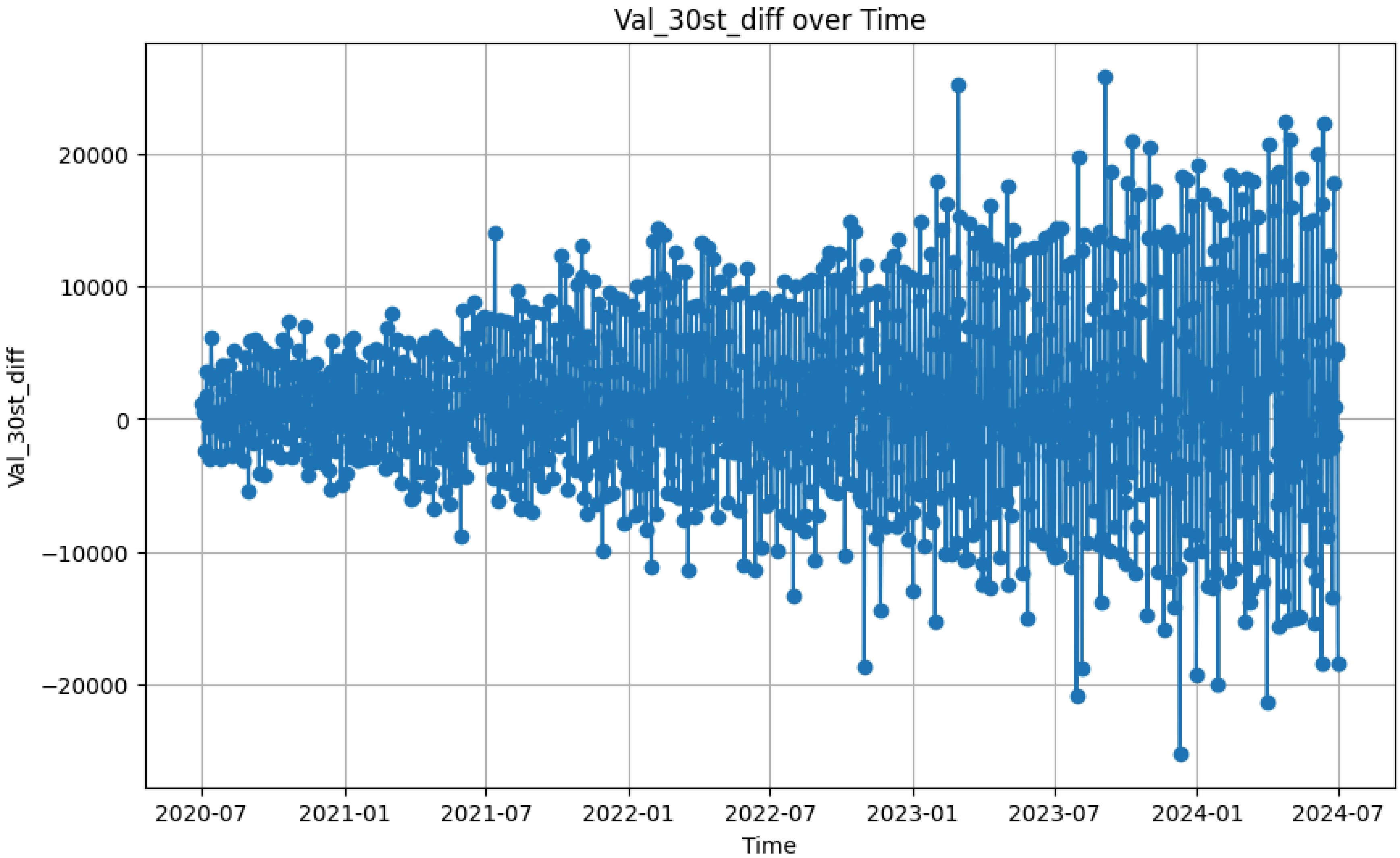
In [ ]:
```
df['Val_30st_diff'][30:].head()
```

Out[43]:

Date	Val_30st_diff
2020-07-01	1198.60
2020-07-02	543.14
2020-07-03	1113.07
2020-07-04	475.08
2020-07-05	-2424.47

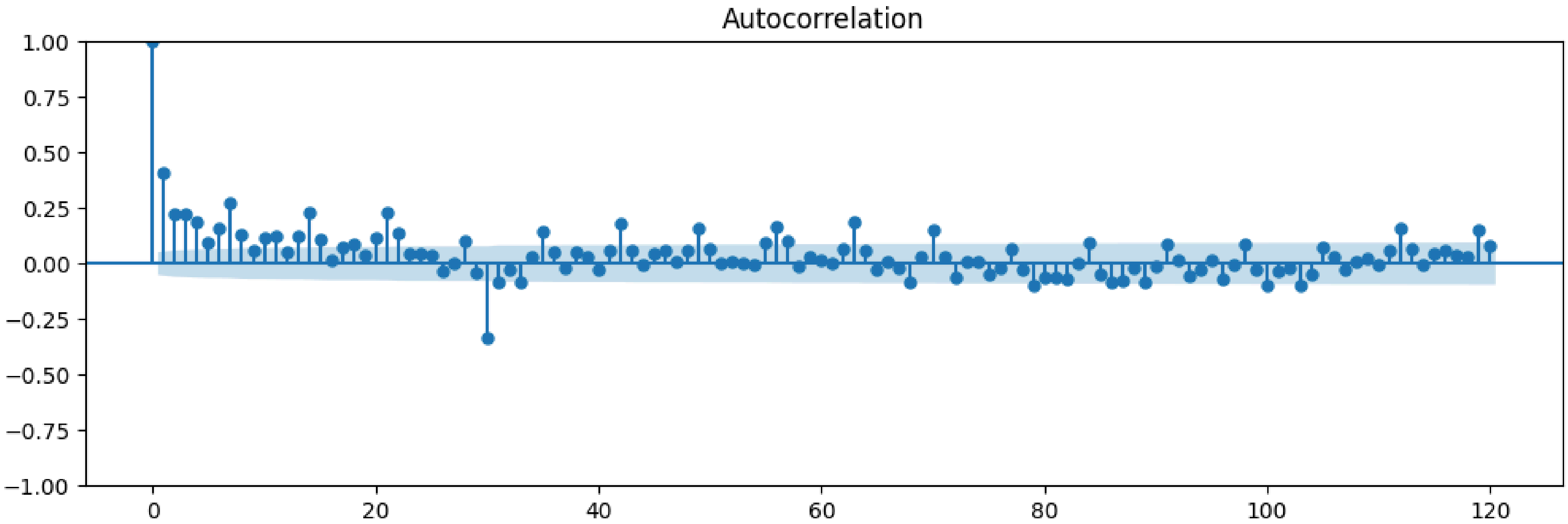
dtype: float64

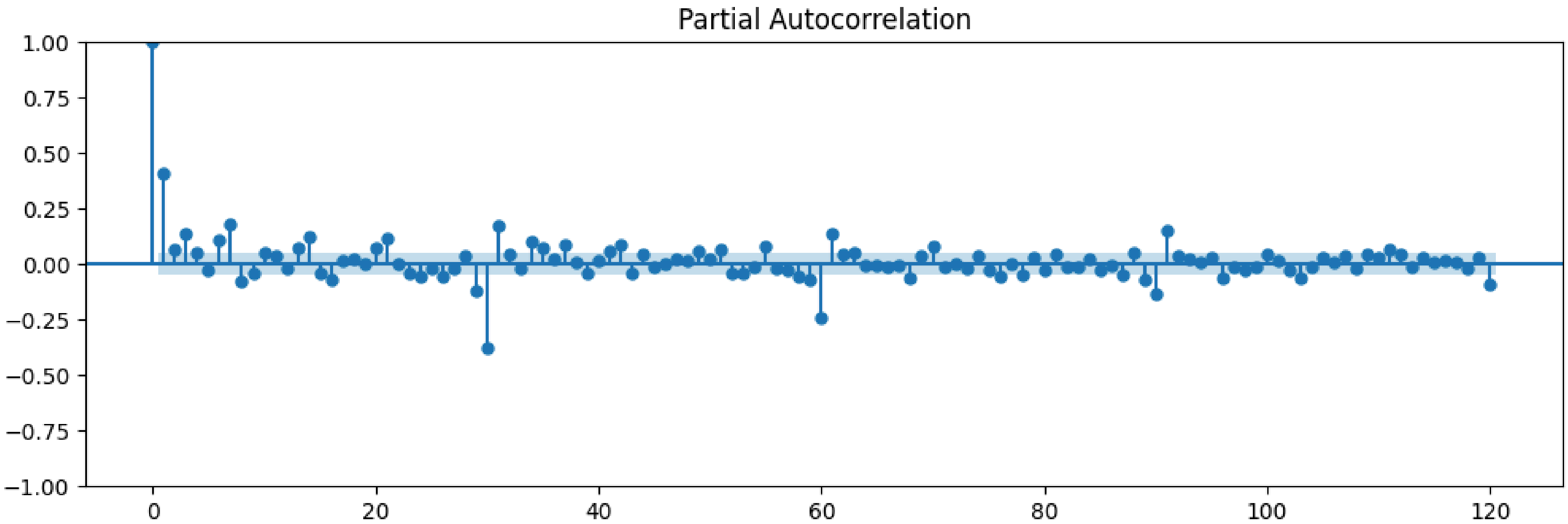
In [ ]:
```
import matplotlib.pyplot as plt

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Val_30st_diff'], marker='o', linestyle='-')

# Adding titles and labels
plt.title('Val_30st_diff over Time')
plt.xlabel('Time')
plt.ylabel('Val_30st_diff')

# Display the plot
plt.grid(True)
plt.show()
```



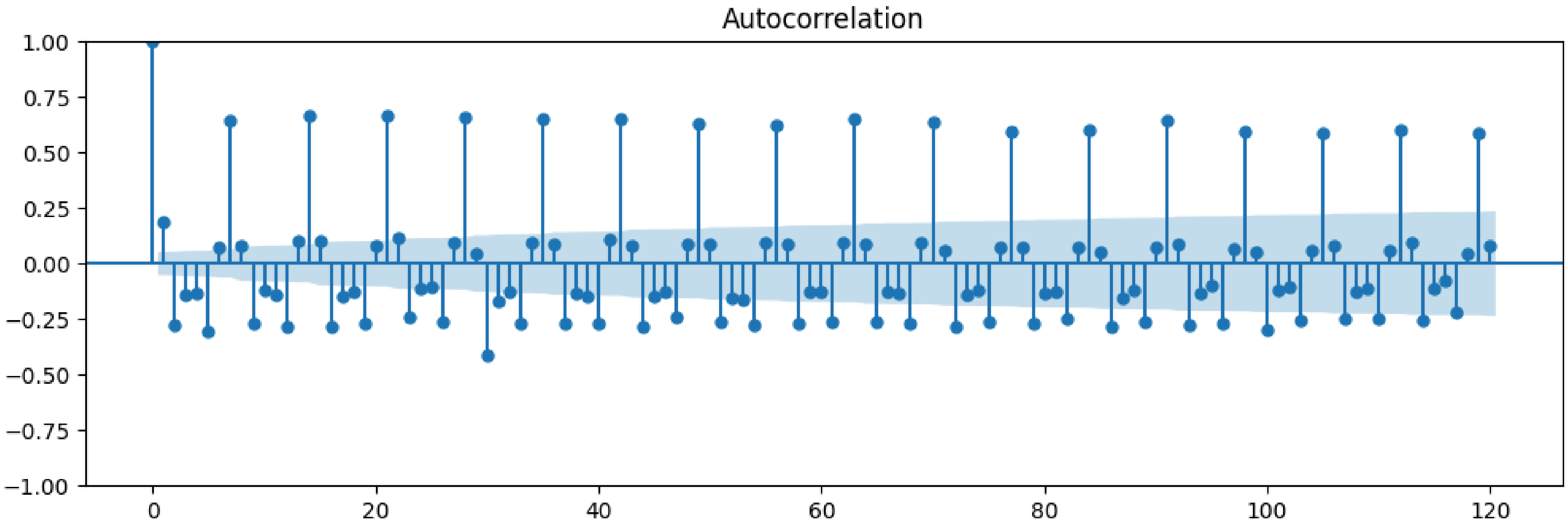
In [ ]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

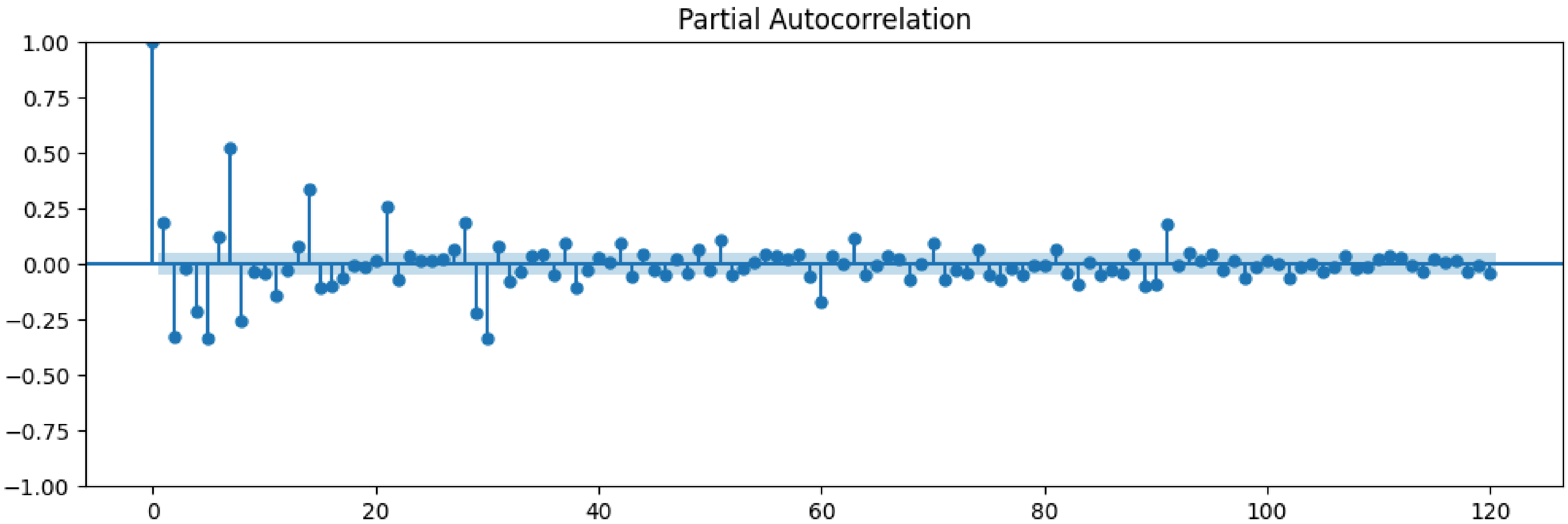
In [ ]:
```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['Vol_30st_diff'].iloc[30:],lags=120,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['Vol_30st_diff'].iloc[30:],lags=120,ax=ax2)
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
120


Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
120

In [ ]:
```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df['Val_30st_diff'].iloc[30:],lags=120,ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df['Val_30st_diff'].iloc[30:],lags=120,ax=ax2)
```





Model fitting

Splitting Dataset

In [ ]:
```
# Split the data into train and test sets
train_size = int(len(df) * 0.8)
train_Vol, test_Vol = df['Vol'].iloc[:train_size], df['Vol'].iloc[train_size:]
train_Val, test_Val = df['Val'].iloc[:train_size], df['Val'].iloc[train_size:]
```

ARIMA(p,d,q)

In [ ]:
```
pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (0.14.2)
Requirement already satisfied: numpy>=1.22.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels)
(1.26.4)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.10/dist-packages (from statsmod
els) (1.13.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.10/dist-packages (from statsmo
dels) (2.1.4)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels)
(0.5.6)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodel
s) (24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pand
as!=2.1.0,>=1.4->statsmodels) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,
>=1.4->statsmodels) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=2.1.
0,>=1.4->statsmodels) (2024.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.6->statsmode
ls) (1.16.0)
```

In [ ]:
```
from statsmodels.tsa.arima.model import ARIMA
import warnings
```

In [ ]:
```
# Suppress warnings for cleaner output
warnings.filterwarnings("ignore")
```

For Volume

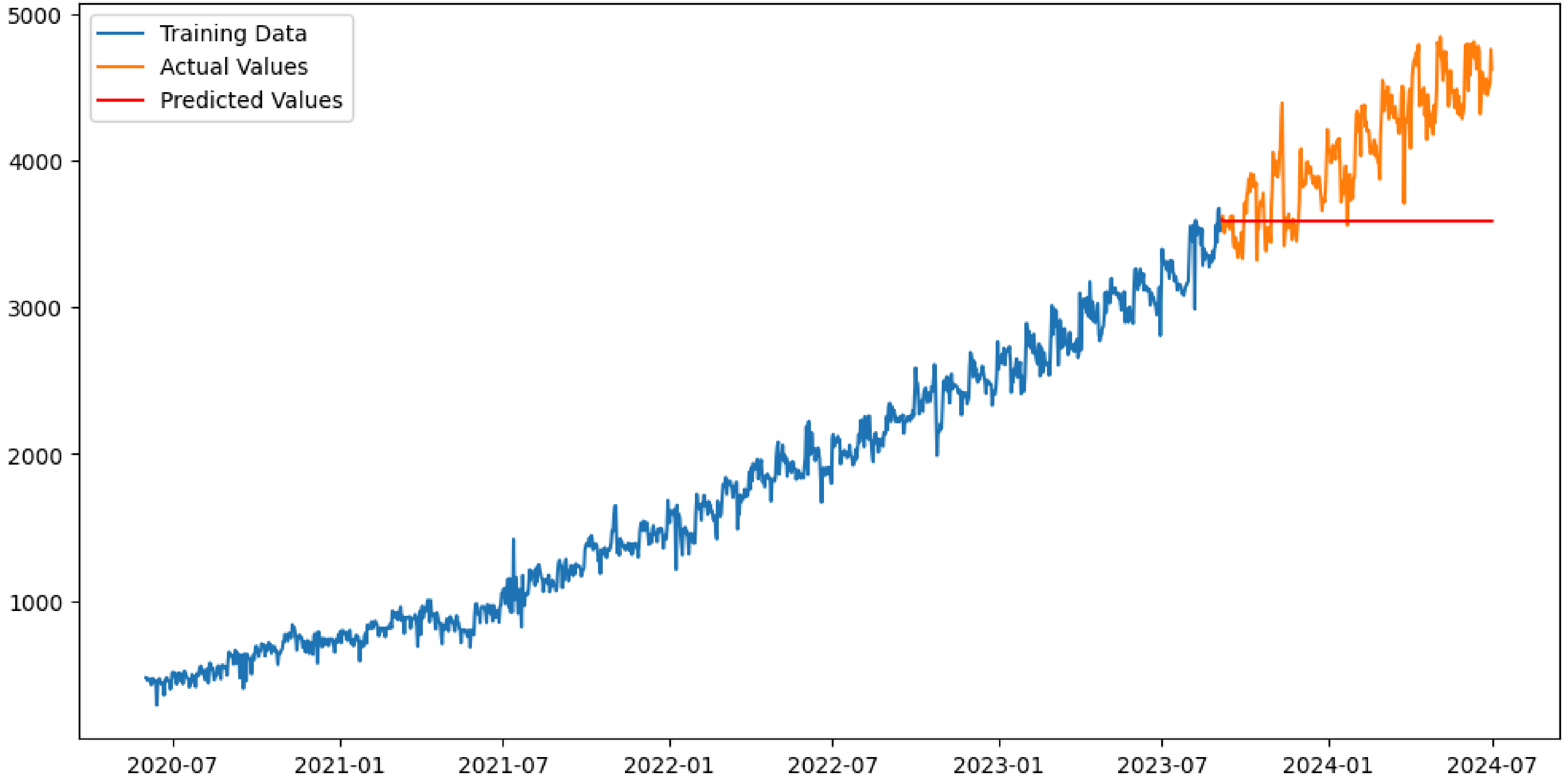
In [ ]:
```
# Fit ARIMA model
# (p, d, q) are the parameters of the ARIMA model. You can adjust these parameters or use auto_arima to deter
model = ARIMA(train_Vol, order=(1, 1, 4))
model_fit = model.fit()
```

In [ ]:
```
# Print summary of the model
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    Vol   No. Observations:                 1192
Model:                 ARIMA(1, 1, 4)   Log Likelihood               -6870.496
Date:                Sat, 10 Aug 2024   AIC                          13752.992
Time:                        08:23:28   BIC                          13783.487
Sample:                    06-01-2020   HQIC                         13764.483
                         - 09-05-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.7759      0.202     -3.833      0.000      -1.173      -0.379
ma.L1          0.3051      0.205      1.488      0.137      -0.097       0.707
ma.L2         -0.4093      0.101     -4.055      0.000      -0.607      -0.211
ma.L3         -0.0423      0.030     -1.428      0.153      -0.100       0.016
ma.L4         -0.0517      0.028     -1.824      0.068      -0.107       0.004
sigma2      5998.2343    131.396     45.650      0.000    5740.703    6255.766
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):              1966.26
Prob(Q):                              0.89   Prob(JB):                         0.00
Heteroskedasticity (H):               3.82   Skew:                             0.15
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.29
===================================================================================

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

In [ ]:
```
# Make predictions
predictions = model_fit.forecast(steps=len(test_Vol))
predictions = pd.Series(predictions, index=test_Vol.index)
```

In [ ]:
```
# Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(train_Vol, label='Training Data')
plt.plot(test_Vol, label='Actual Values')
plt.plot(predictions, label='Predicted Values', color='red')
plt.legend()
plt.show()
```



In [ ]:
```
pip install pmdarima
```

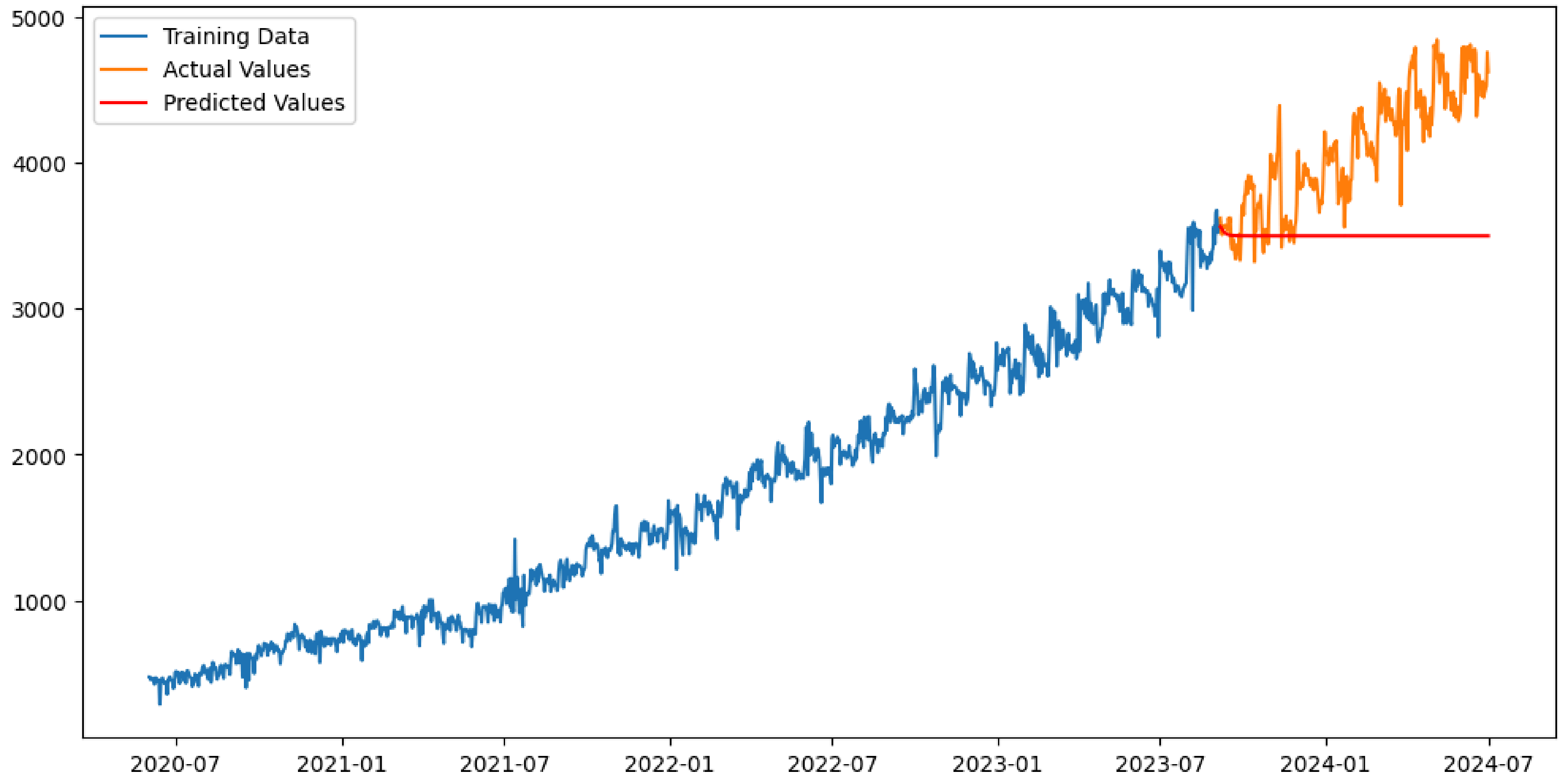
```
Collecting pmdarima
  Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.w
hl.metadata (7.8 kB)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.
4.2)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.10/dist-packages
(from pmdarima) (3.0.11)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.
26.4)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (2.
1.4)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages (from pmdarim
a) (1.3.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (1.1
3.1)
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/dist-packages (from pmdarim
a) (0.14.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from pmdarima) (2.0.7)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.10/dist-packages (from
pmdarima) (71.0.4)
Requirement already satisfied: packaging>=17.1 in /usr/local/lib/python3.10/dist-packages (from pmdarima)
(24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pand
as>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19->
pmdarima) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.19
->pmdarima) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit
-learn>=0.22->pmdarima) (3.5.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=
0.13.2->pmdarima) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.6->statsmode
ls>=0.13.2->pmdarima) (1.16.0)
Downloading pmdarima-2.0.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.manylinux_2_28_x86_64.whl
(2.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 12.0 MB/s eta 0:00:00
Installing collected packages: pmdarima
Successfully installed pmdarima-2.0.4
```

Using Predefined library which is able to find parameters od ARIMA model.

In [ ]:
```
from pmdarima import auto_arima

# Find the best ARIMA parameters using auto_arima
auto_model = auto_arima(train_Vol, seasonal=True, stepwise=True, suppress_warnings=True, trace=True)

# Print the summary of the best model found
print(auto_model.summary())

# Fit the model with the determined parameters
best_order = auto_model.order
model = ARIMA(train_Vol, order=best_order)
model_fit = model.fit()

# Make predictions
predictions = model_fit.forecast(steps=len(test_Vol))
predictions = pd.Series(predictions, index=test_Vol.index)

# Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(train_Vol, label='Training Data')
plt.plot(test_Vol, label='Actual Values')
plt.plot(predictions, label='Predicted Values', color='red')
plt.legend()
plt.show()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=13749.146, Time=6.03 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=13986.207, Time=0.11 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=13809.455, Time=0.25 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=13748.998, Time=3.81 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=13985.319, Time=0.08 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=13746.818, Time=4.10 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=13702.387, Time=6.59 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=13767.786, Time=0.76 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=13696.446, Time=10.75 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=13759.823, Time=1.07 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=13698.440, Time=12.32 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=14.31 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=13760.413, Time=1.05 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=14.12 sec
 ARIMA(3,1,1)(0,0,0)[0]             : AIC=13731.999, Time=3.08 sec

Best model:  ARIMA(3,1,1)(0,0,0)[0] intercept
Total fit time: 78.480 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1192
Model:               SARIMAX(3, 1, 1)   Log Likelihood               -6842.223
Date:                Sat, 10 Aug 2024   AIC                          13696.446
Time:                        08:25:08   BIC                          13726.941
Sample:                    06-01-2020   HQIC                         13707.938
                         - 09-05-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.7657      0.138      5.550      0.000       0.495       1.036
ar.L1          0.4450      0.023     19.602      0.000       0.400       0.489
ar.L2          0.1728      0.026      6.559      0.000       0.121       0.224
ar.L3          0.0833      0.027      3.131      0.002       0.031       0.135
ma.L1         -0.9657      0.014    -68.105      0.000      -0.994      -0.938
sigma2      5716.7659    133.454     42.837      0.000    5455.201    5978.331
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              1616.35
Prob(Q):                              0.98   Prob(JB):                         0.00
Heteroskedasticity (H):               3.55   Skew:                             0.22
```

```
Prob(H) (two-sided):                  0.00   Kurtosis:                         8.69
===================================================================================

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



For Value

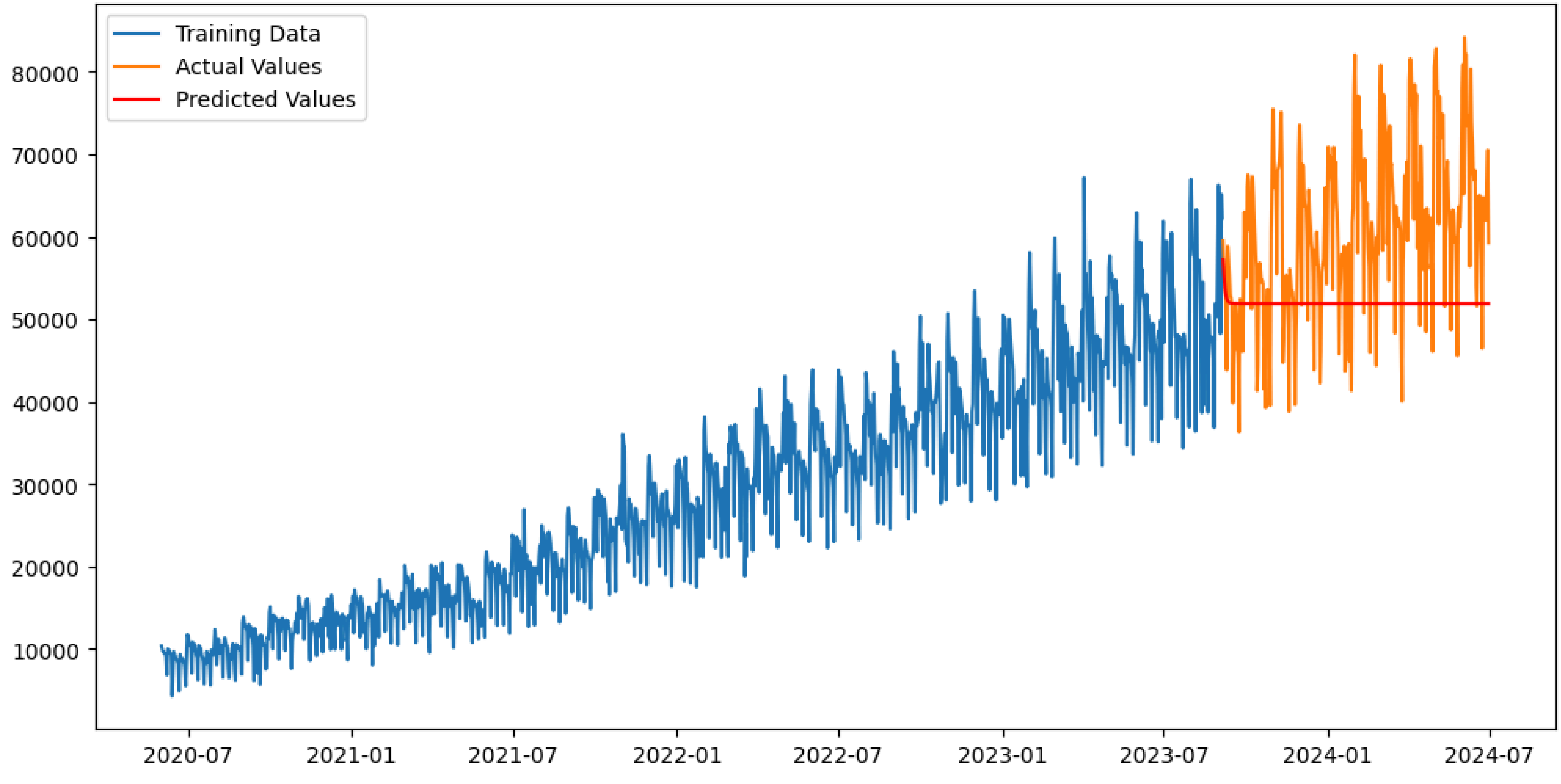
In [ ]:
```
# Fit ARIMA model
# (p, d, q) are the parameters of the ARIMA model. You can adjust these parameters or use auto_arima to deter
model = ARIMA(train_Val, order=(2, 1, 1))
model_fit = model.fit()
```

In [ ]:
```
# Print summary of the model
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    Val   No. Observations:                 1192
Model:                 ARIMA(2, 1, 1)   Log Likelihood              -11685.321
Date:                Sat, 10 Aug 2024   AIC                          23378.642
Time:                        08:25:16   BIC                          23398.972
Sample:                    06-01-2020   HQIC                         23386.303
                         - 09-05-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3456      0.022     15.409      0.000       0.302       0.390
ar.L2          0.1331      0.030      4.412      0.000       0.074       0.192
ma.L1         -0.9612      0.011    -90.940      0.000      -0.982      -0.940
sigma2      2.154e+07   2.24e-10   9.61e+16      0.000    2.15e+07    2.15e+07
===================================================================================
Ljung-Box (L1) (Q):                   0.31   Jarque-Bera (JB):               227.27
Prob(Q):                              0.58   Prob(JB):                         0.00
Heteroskedasticity (H):               7.33   Skew:                             0.14
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.12
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 4.14e+31. Standard errors may be
unstable.
```

In [ ]:
```
# Make predictions
predictions = model_fit.forecast(steps=len(test_Val))
predictions = pd.Series(predictions, index=test_Val.index)
```

In [ ]:
```
# Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(train_Val, label='Training Data')
plt.plot(test_Val, label='Actual Values')
plt.plot(predictions, label='Predicted Values', color='red')
plt.legend()
plt.show()
```



Using Predefined library which is able to find parameters od ARIMA model.

SARIMA(p,d,q)*(P,D,Q,S)

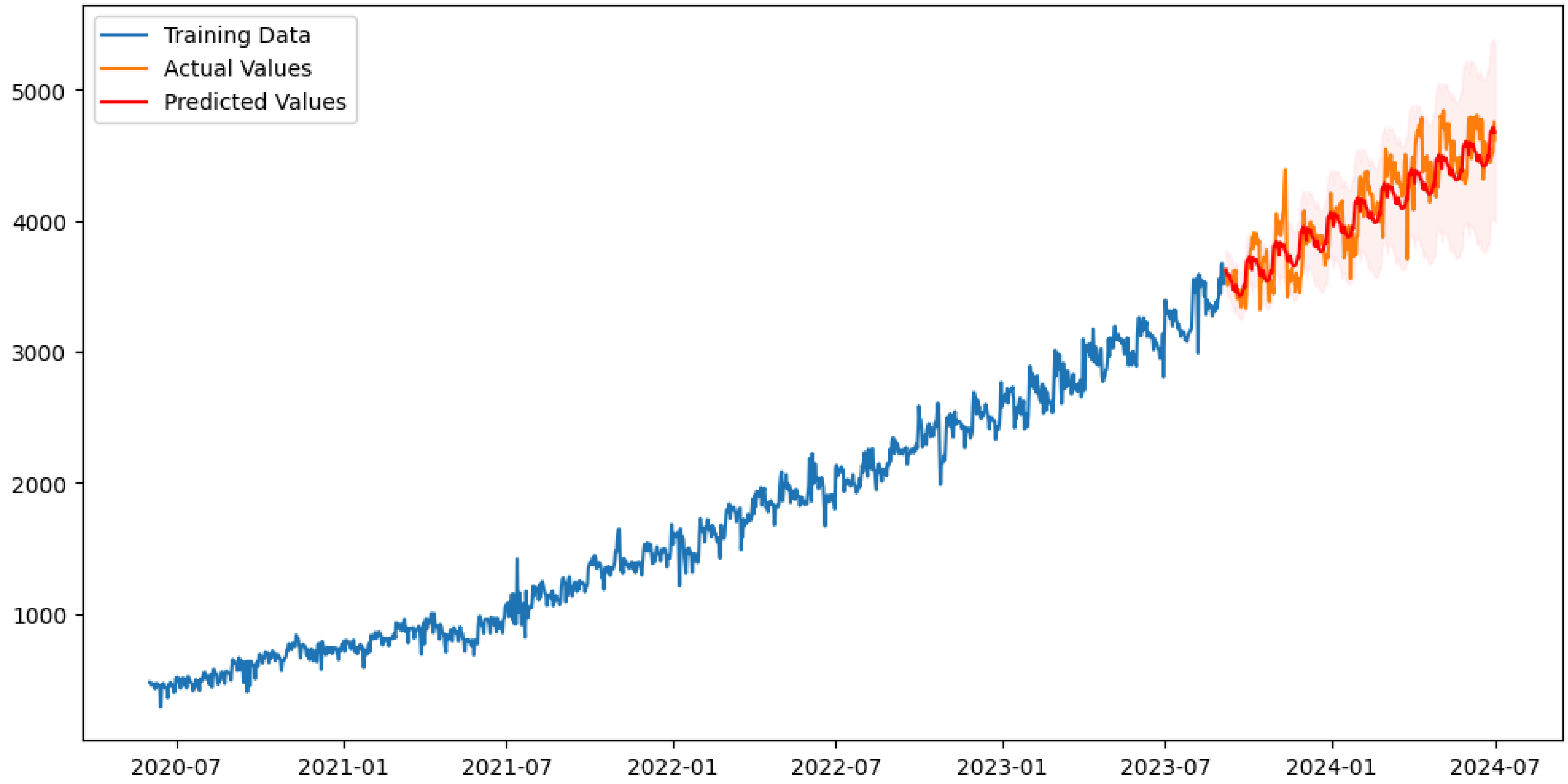
For Volume

In [ ]:
```
# Fit SARIMA model
# (p, d, q) are the parameters for the non-seasonal part of the model.
# (P, D, Q, s) are the parameters for the seasonal part of the model.
# 's' is the periodicity of the seasonality (e.g., 7 for weekly, 12 for monthly).
# You can adjust these parameters or use auto_arima to determine the best parameters.
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [ ]:
```
model = SARIMAX(train_Vol, order=(1, 0, 3), seasonal_order=(1, 1, 1, 30))
model_fit_vol = model.fit(disp=False)

# Forecast and get predictions for the training data to get residuals
train_predictions = model_fit_vol.fittedvalues
residuals = train_Vol - train_predictions

# Print summary of the model
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    Val   No. Observations:                 1192
Model:                 ARIMA(2, 1, 1)   Log Likelihood              -11685.321
Date:                Sat, 10 Aug 2024   AIC                          23378.642
Time:                        08:26:51   BIC                          23398.972
Sample:                    06-01-2020   HQIC                         23386.303
                         - 09-05-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3456      0.022     15.409      0.000       0.302       0.390
ar.L2          0.1331      0.030      4.412      0.000       0.074       0.192
ma.L1         -0.9612      0.011    -90.940      0.000      -0.982      -0.940
sigma2      2.154e+07   2.24e-10   9.61e+16      0.000    2.15e+07    2.15e+07
===================================================================================
Ljung-Box (L1) (Q):                   0.31   Jarque-Bera (JB):               227.27
Prob(Q):                              0.58   Prob(JB):                         0.00
Heteroskedasticity (H):               7.33   Skew:                             0.14
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.12
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 4.14e+31. Standard errors may be unstable.
```

In [ ]:
```
# Make predictions
predictions = model_fit_vol.get_forecast(steps=len(test_Vol))
predicted_mean = predictions.predicted_mean
conf_int = predictions.conf_int()

# Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(train_Vol, label='Training Data')
plt.plot(test_Vol, label='Actual Values')
plt.plot(predicted_mean, label='Predicted Values', color='red')
plt.fill_between(predicted_mean.index, conf_int.iloc[:, 0], conf_int.iloc[:, 1], color='pink', alpha=0.2)
plt.legend()
plt.show()
```


Training Data
Actual Values
Predicted Values
5000
4000
3000
2000
1000
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
2024-01
2024-07

In [ ]:

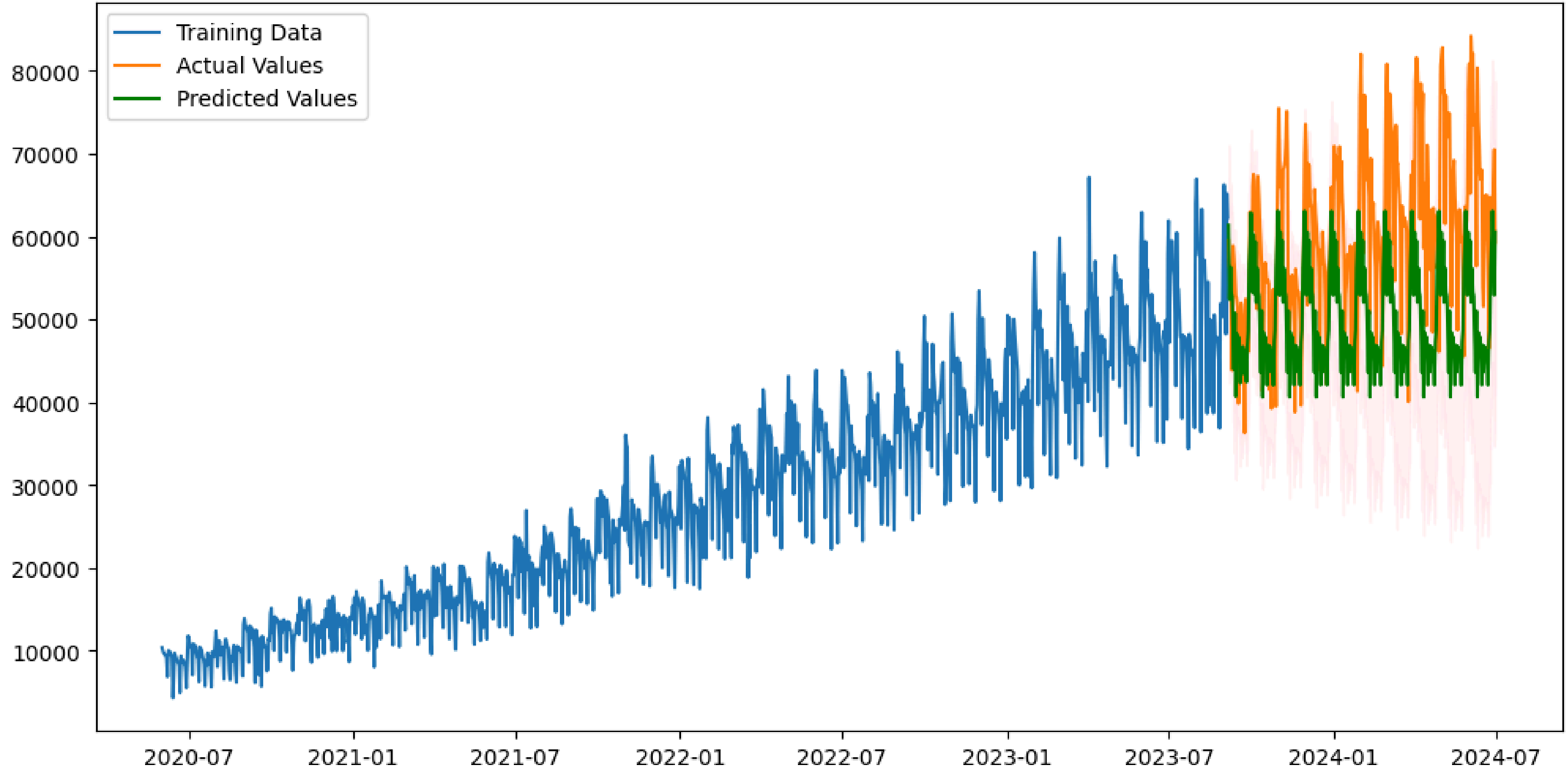
```

```

For Value

In [ ]:
```
model = SARIMAX(train_Val, order=(1,0, 0), seasonal_order=(1, 1, 1, 30))
model_fit_val = model.fit(disp=False)

# Forecast and get predictions for the training data to get residuals
train_predictions = model_fit_val.fittedvalues
residuals = train_Val - train_predictions

# Print summary of the model
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                    Val   No. Observations:                 1192
Model:                 ARIMA(2, 1, 1)   Log Likelihood              -11685.321
Date:                Sat, 10 Aug 2024   AIC                          23378.642
Time:                        08:27:32   BIC                          23398.972
Sample:                    06-01-2020   HQIC                         23386.303
                         - 09-05-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3456      0.022     15.409      0.000       0.302       0.390
ar.L2          0.1331      0.030      4.412      0.000       0.074       0.192
ma.L1         -0.9612      0.011    -90.940      0.000      -0.982      -0.940
sigma2      2.154e+07   2.24e-10   9.61e+16      0.000    2.15e+07    2.15e+07
===================================================================================
Ljung-Box (L1) (Q):                   0.31   Jarque-Bera (JB):               227.27
Prob(Q):                              0.58   Prob(JB):                         0.00
Heteroskedasticity (H):               7.33   Skew:                             0.14
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.12
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 4.14e+31. Standard errors may be unstable.
```

In [ ]:
```
# Make predictions
predictions = model_fit_val.get_forecast(steps=len(test_Val))
predicted_mean = predictions.predicted_mean
conf_int = predictions.conf_int()

# Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(train_Val, label='Training Data')
plt.plot(test_Val, label='Actual Values')
plt.plot(predicted_mean, label='Predicted Values', color='green')
plt.fill_between(predicted_mean.index, conf_int.iloc[:, 0], conf_int.iloc[:, 1], color='pink', alpha=0.2)
plt.legend()
plt.show()
```



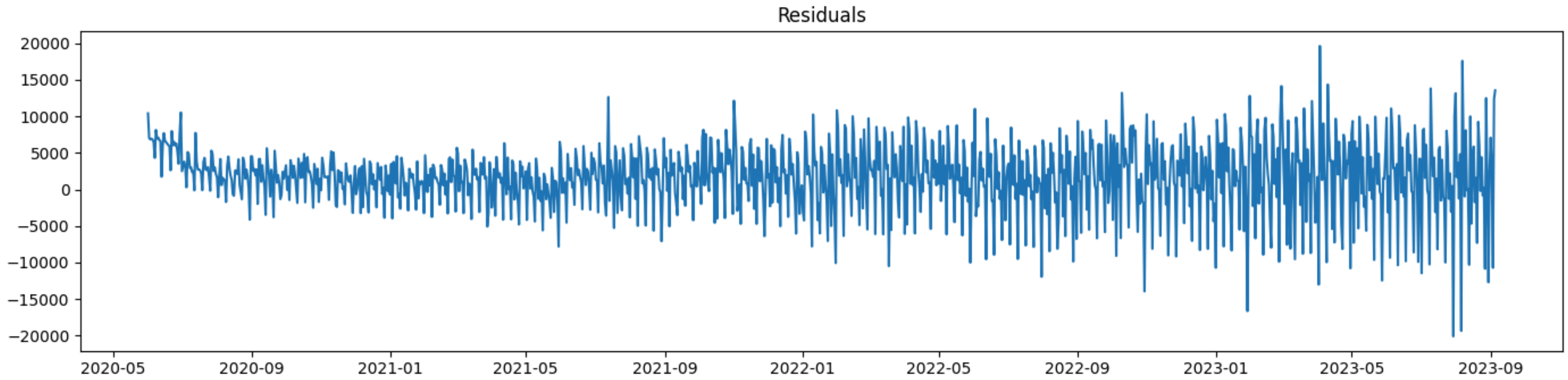
Type *Markdown* and LaTeX: α^2

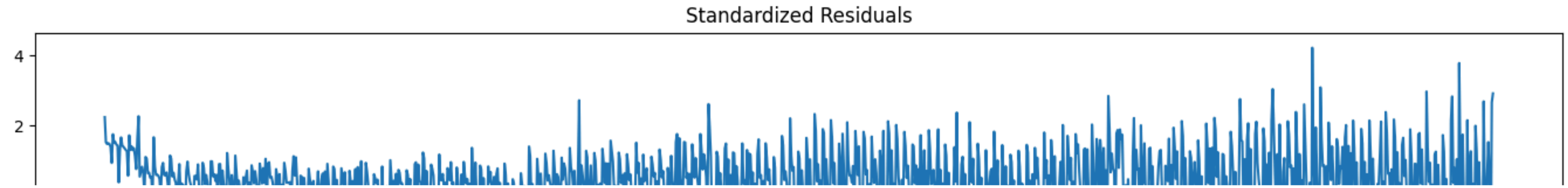
In [ ]:

```

```

In [ ]:
```
# Residuals
residuals = model_fit_val.resid
standardized_residuals = residuals / np.std(residuals)

# Plot the residuals and standardized residuals
plt.figure(figsize=(14, 7))

plt.subplot(2, 1, 1)
plt.plot(residuals)
plt.title('Residuals')

plt.subplot(2, 1, 2)
plt.plot(standardized_residuals)
plt.title('Standardized Residuals')

plt.tight_layout()
plt.show()

# Plot ACF and PACF of standardized residuals
plt.figure(figsize=(12, 6))
plt.subplot(121)
plot_acf(standardized_residuals, ax=plt.gca(), lags=40)
plt.title('ACF of Standardized Residuals')
plt.subplot(122)
plot_pacf(standardized_residuals, ax=plt.gca(), lags=40)
plt.title('PACF of Standardized Residuals')
plt.show()

# Perform Ljung-Box test
from statsmodels.stats.diagnostic import acorr_ljungbox
ljung_box_result = acorr_ljungbox(standardized_residuals, lags=[10], return_df=True)
print("Ljung-Box test result:")
print(ljung_box_result)

# Interpretation
if ljung_box_result['lb_pvalue'].values[0] > 0.05:
    print("Fail to reject the null hypothesis: Residuals are independently distributed.")
else:
    print("Reject the null hypothesis: Residuals are not independently distributed.")

# Plot the distribution of the standardized residuals
```

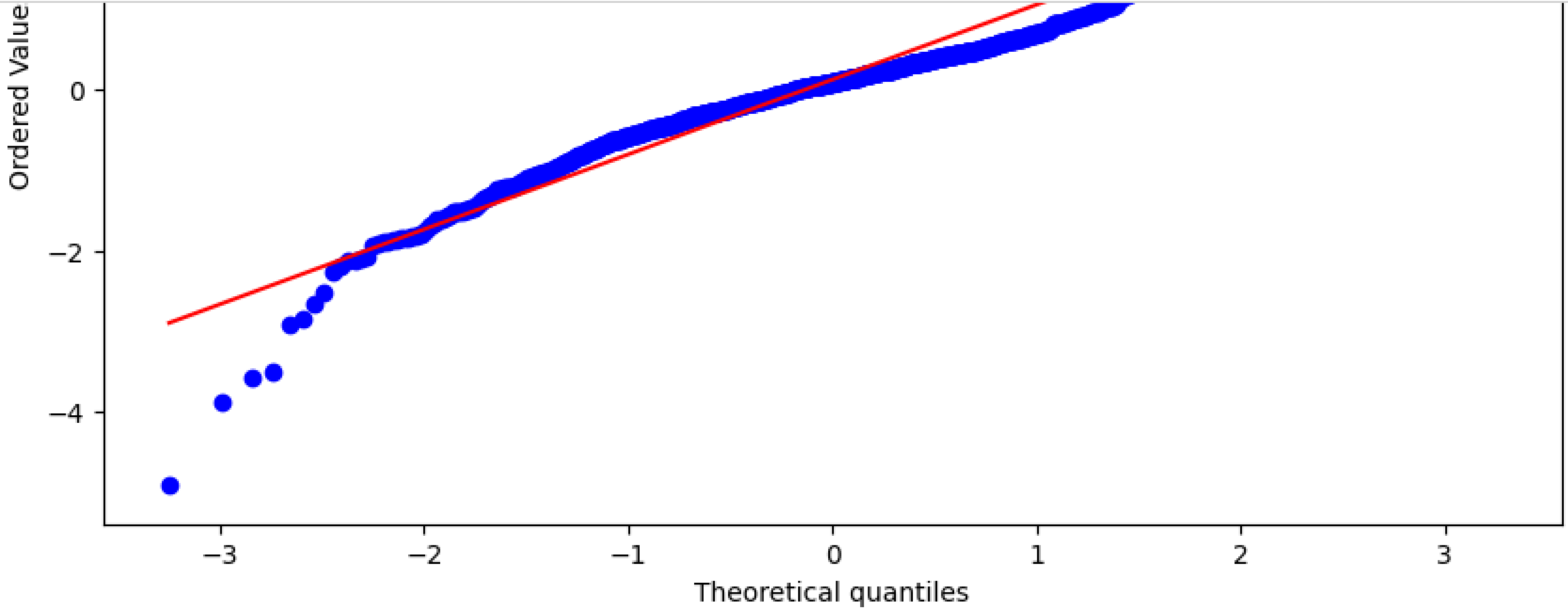
```
plt.figure(figsize=(10, 6))
sns.histplot(standardized_residuals, kde=True)
plt.title('Distribution of Standardized Residuals')
plt.show()

# Plot Q-Q plot
plt.figure(figsize=(10, 6))

import scipy.stats as stats

# Assuming standardized_residuals is a pandas Series or NumPy array
stats.probplot(standardized_residuals, dist="norm", plot=plt)



stats.probplot(standardized_residuals, dist="norm", plot=plt)
plt.title('Q-Q Plot of Standardized Residuals')
plt.show()
```





In [ ]:
```
# Residuals
residuals = model_fit_vol.resid
standardized_residuals = residuals / np.std(residuals)

# Plot the residuals and standardized residuals
plt.figure(figsize=(14, 7))

plt.subplot(2, 1, 1)
plt.plot(residuals)
plt.title('Residuals')

plt.subplot(2, 1, 2)
plt.plot(standardized_residuals)
plt.title('Standardized Residuals')

plt.tight_layout()
plt.show()

# Plot ACF and PACF of standardized residuals
plt.figure(figsize=(12, 6))
plt.subplot(121)
plot_acf(standardized_residuals, ax=plt.gca(), lags=40)
plt.title('ACF of Standardized Residuals')
plt.subplot(122)
plot_pacf(standardized_residuals, ax=plt.gca(), lags=40)
plt.title('PACF of Standardized Residuals')
plt.show()

# Perform Ljung-Box test
ljung_box_result = acorr_ljungbox(standardized_residuals, lags=[10], return_df=True)
print("Ljung-Box test result:")
print(ljung_box_result)

# Interpretation
if ljung_box_result['lb_pvalue'].values[0] > 0.05:
    print("Fail to reject the null hypothesis: Residuals are independently distributed.")
else:
    print("Reject the null hypothesis: Residuals are not independently distributed.")

# Plot the distribution of the standardized residuals
plt.figure(figsize=(10, 6))
```

```
sns.histplot(standardized_residuals, kde=True)
plt.title('Distribution of Standardized Residuals')
plt.show()

# Plot Q-Q plot
plt.figure(figsize=(10, 6))
stats.probplot(standardized_residuals, dist="norm", plot=plt)
plt.title('Q-Q Plot of Standardized Residuals')
plt.show()
```




```
In [ ]:
```