In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import plotly.offline as py
import plotly.graph_objs as go
from sklearn.preprocessing import LabelEncoder
import warnings
# Ignore all warnings
warnings.filterwarnings('ignore')

from sklearn import linear_model
import statsmodels.api as sm
# pip install johansen
from statsmodels.tsa.vector_ar.vecm import coint_johansen
```

In [2]:

```
# data = pd.read_excel('jse all share index (333)(2).xlsx')
data = pd.read_excel('jse all share index liqudity(1).xlsx')
data.tail()
```

Out[2]:

	Date	Open	High	Low	Close	Volume	
1250	Monday, January 03, 2022	73638.720000	74290.930000	73638.720000	73722.600000	9.516390e+07	-0.(
1251	NaN	73730.157818	74263.606392	73092.368156	73621.908123	2.632627e+08	-0.(
1252	Thursday, December 29, 2023	73754.707447	74287.583683	73116.488387	73645.465651	2.633718e+08	0.(
1253	Wednesday, December 28, 2023	73779.257076	74311.560973	73140.608617	73669.023178	2.634810e+08	0.(
1254	Friday, December 23, 2023	73803.806705	74335.538264	73164.728847	73692.580705	2.635901e+08	0.(

In [3]:

```
data.columns
```

Out[3]:

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Return  ',
       'liquidity', 'volume in bln dollor', 'illiqudity'],
      dtype='object')
```

In [4]:

```
# Calculate daily percentage change in stock prices
data['Daily_Return'] = data['Close'].pct_change()
data.head(2)
```

Out[4]:

	Date	Open	High	Low	Close	Volume	Return	liquidity	
0	Monday, December 31, 2018	52729.14	52791.8	52444.89	52736.86	72328983.0	NaN	NaN	3
1	Friday, December 28, 2018	51548.65	52546.0	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06	6

In [5]:

```
# Drop the first row of the DataFrame
data = data.dropna()
data.head(2)
```

Out[5]:

	Date	Open	High	Low	Close	Volume	Return	liquidity
1	Friday, December 28, 2018	51548.65	52546.00	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06
2	Thursday, December 27, 2018	52252.80	52661.54	51178.51	51551.71	149747866.0	-0.017031	-4.594459e+05

In [6]:

```
daily_volatility = data['Daily_Return'].std()
daily_volatility
```

Out[6]:

```
0.01685777390437592
```

In [7]:

```
# Calculate the True Range (TR)
data['close_prev'] = data['Close'].shift(1)
data['volatility (ATR)'] = data.apply(lambda row: max(row['High'] - row['Low'],
                                        abs(row['High'] - row['close_prev']), abs(row['Lo
```

In [8]:

```
# Calculate annualized volatility
annualized_volatility = daily_volatility * np.sqrt(252)
annualized_volatility
```

Out[8]:

```
0.2676088644547982
```

In [9]:

```
# Print the daily and annualized volatility
print("Daily Volatility:", daily_volatility)
print("Annualized Volatility:", annualized_volatility)
```

```
Daily Volatility: 0.01685777390437592
Annualized Volatility: 0.2676088644547982
```

In [10]:

```
data.dtypes
```

Out[10]:

```
Date                       object
Open                      float64
High                      float64
Low                       float64
Close                     float64
Volume                    float64
Return                    float64
liquidity                 float64
volume in bln dollor      float64
illiqudity                float64
Daily_Return              float64
close_prev                float64
volatility (ATR)          float64
dtype: object
```

In [11]:

```
#rename column
data = data.rename(columns={'Volume':'volume','Date':'date','Open':'open','High':'high','
data.head()
```

Out[11]:

	date	open	high	low	close	volume	Return	liquidity
1	Friday, December 28, 2018	51548.65	52546.00	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06
2	Thursday, December 27, 2018	52252.80	52661.54	51178.51	51551.71	149747866.0	-0.017031	-4.594459e+05
3	Monday, December 24, 2018	51583.69	52200.83	51430.36	52081.11	63667185.0	0.010269	3.198064e+05
4	Friday, December 21, 2018	51654.27	52031.79	51347.57	51430.36	503502581.0	-0.012495	-2.081488e+06
5	Thursday, December 20, 2018	50998.01	51569.77	50698.06	51347.57	382348711.0	-0.001610	-1.211308e+07

In [12]:

```
# Calculate turnover ratio
data['market_cap'] = data['close'] * data['volume'] # Calculate market cap
data['turnover_ratio'] = data['volume'] / data['market_cap'] # Calculate turnover ratio
data.head(1)
```

Out[12]:

	date	open	high	low	close	volume	Return	liquidity
1	Friday, December 28, 2018	51548.65	52546.0	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06

In [13]:

```
# Calculate P/E ratio
data['earnings_per_share'] = (data['high'] + data['low'] + data['close']) / 3 # Calculate
data['price_per_share'] = data['close'] # Calculate price per share
data['pe_ratio'] = data['price_per_share'] / data['earnings_per_share'] # Calculate P/E r
data.head(2)
```

Out[13]:

	date	open	high	low	close	volume	Return	liquidity
1	Friday, December 28, 2018	51548.65	52546.00	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06
2	Thursday, December 27, 2018	52252.80	52661.54	51178.51	51551.71	149747866.0	-0.017031	-4.594459e+05

In [14]:

```
# Calculate return on investment
data['return_on_investment'] = (data['close'] - data['open']) / data['open'] # Calculate
data.head(2)
```

Out[14]:

	date	open	high	low	close	volume	Return	liquidity
1	Friday, December 28, 2018	51548.65	52546.00	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06
2	Thursday, December 27, 2018	52252.80	52661.54	51178.51	51551.71	149747866.0	-0.017031	-4.594459e+05

In [15]:

```
# Calculate volatility
data['daily_return'] = (data['close'] - data['open']) / data['open'] # Calculate daily re
data['volatility'] = np.std(data['daily_return']) # Calculate volatility
data.head(1)
```

Out[15]:

	date	open	high	low	close	volume	Return	liquidity
1	Friday, December 28, 2018	51548.65	52546.0	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06

1 rows × 21 columns

In [16]:

```
# Calculate market capitalization
data['market_cap'] = data['close'] * data['volume'] # Calculate market capitalization
data.head(2)
```

Out[16]:

	date	open	high	low	close	volume	Return	liquidity
1	Friday, December 28, 2018	51548.65	52546.00	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06
2	Thursday, December 27, 2018	52252.80	52661.54	51178.51	51551.71	149747866.0	-0.017031	-4.594459e+05

2 rows × 21 columns

In [17]:

```
# # Calculate the absolute return
# data["abs_return"] = abs(data["close"] - data["open"])

# # Calculate the dollar volume
# data["dollar_volume"] = data["volume"] * data["open"]

# # Calculate the Amihud liquidity
# data["amihud_liquidity"] = data["abs_return"] / data["dollar_volume"]
# data.head()
```

In [18]:

```
# data['abs_returns'] = data['close'].pct_change().abs()
# data.head()
```

In [19]:

```
# data['illiquidity'] = data['abs_returns']/(data['close']*data['volume'])
# data.head()
```

In [ ]:

```
```

In [ ]:

```
```

In [20]:

```
from datetime import datetime

date_format = "%A, %B %d, %Y"
data['date'] = pd.to_datetime(data['date'], format=date_format)
data.head(2)
```

Out[20]:

	date	open	high	low	close	volume	Return	liquidity	vo bl
1	2018-12-28	51548.65	52546.00	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06	6205
2	2018-12-27	52252.80	52661.54	51178.51	51551.71	149747866.0	-0.017031	-4.594459e+05	7824

2 rows × 21 columns

In [21]:

```
data.dtypes
```

Out[21]:

```
date                      datetime64[ns]
open                             float64
high                             float64
low                              float64
close                            float64
volume                           float64
Return                           float64
liquidity                        float64
volume in bln dollor             float64
illiqudity                       float64
Daily_Return                     float64
close_prev                       float64
volatility (ATR)                 float64
market_cap                       float64
turnover_ratio                   float64
earnings_per_share               float64
price_per_share                  float64
pe_ratio                         float64
return_on_investment             float64
daily_return                     float64
volatility                       float64
dtype: object
```

In [22]:

```
data.index
```

Out[22]:

```
Int64Index([   1,    2,    3,    4,    5,    6,    7,    8,    9,   10,
            ...
            1244, 1245, 1246, 1247, 1248, 1249, 1250, 1252, 1253, 1254],
           dtype='int64', length=1240)
```

In [23]:

```
print("The dataset is {} dataset".format(data.shape))
```

```
The dataset is (1240, 21) dataset
```

In [24]:

```
data.columns
```

Out[24]:

```
Index(['date', 'open', 'high', 'low', 'close', 'volume', 'Return  ',
       'liquidity', 'volume in bln dollor', 'illiqudity', 'Daily_Return',
       'close_prev', 'volatility (ATR)', 'market_cap', 'turnover_ratio',
       'earnings_per_share', 'price_per_share', 'pe_ratio',
       'return_on_investment', 'daily_return', 'volatility'],
      dtype='object')
```

In [25]:

```
data.size # This is the size of the dataframe
```

Out[25]:

```
26040
```

In [26]:

```
data.memory_usage() #The memory usage of each column in the dataframe in bytes
```

Out[26]:

```
Index                   9920
date                    9920
open                    9920
high                    9920
low                     9920
close                   9920
volume                  9920
Return                  9920
liquidity               9920
volume in bln dollor    9920
illiqudity              9920
Daily_Return            9920
close_prev              9920
volatility (ATR)        9920
market_cap              9920
turnover_ratio          9920
earnings_per_share      9920
price_per_share         9920
pe_ratio                9920
return_on_investment    9920
daily_return            9920
volatility              9920
dtype: int64
```

In [27]:

```
data.ndim #The number of axes/ array dimensions
```

Out[27]:

```
2
```

In [28]:

```
data.info() # prints a concise summary of the data frame
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1240 entries, 1 to 1254
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   date                  1240 non-null   datetime64[ns]
 1   open                  1240 non-null   float64
 2   high                  1240 non-null   float64
 3   low                   1240 non-null   float64
 4   close                 1240 non-null   float64
 5   volume                1240 non-null   float64
 6   Return                1240 non-null   float64
 7   liquidity             1240 non-null   float64
 8   volume in bln dollor  1240 non-null   float64
 9   illiqudity            1240 non-null   float64
 10  Daily_Return          1240 non-null   float64
 11  close_prev            1239 non-null   float64
 12  volatility (ATR)      1240 non-null   float64
 13  market_cap            1240 non-null   float64
 14  turnover_ratio        1240 non-null   float64
 15  earnings_per_share    1240 non-null   float64
 16  price_per_share       1240 non-null   float64
 17  pe_ratio              1240 non-null   float64
 18  return_on_investment  1240 non-null   float64
 19  daily_return          1240 non-null   float64
 20  volatility            1240 non-null   float64
dtypes: datetime64[ns](1), float64(20)
memory usage: 213.1 KB
```

In [29]:

```
#identifying unique data for each feature
def unique_value(data_set, column_name):
    return data_set[column_name].nunique()

print("Number of the Unique Values:")
print(unique_value(data,list(data.columns)))
```

```
Number of the Unique Values:
date                    1240
open                    1240
high                    1232
low                     1222
close                   1240
volume                  1235
Return                  1240
liquidity               1240
volume in bln dollor    1240
illiqudity              1240
Daily_Return            1240
close_prev              1239
volatility (ATR)        1238
market_cap              1240
turnover_ratio          1240
earnings_per_share      1240
price_per_share         1240
pe_ratio                1240
return_on_investment    1240
daily_return            1240
volatility                 1
dtype: int64
```

In [30]:

```
# Handling missing values

def missing_value_table(data):
    missing_value = data.isna().sum().sort_values(ascending=False)
    missing_value_percent = 100 * data.isna().sum()//len(data)
    missing_value_table = pd.concat([missing_value, missing_value_percent], axis=1)
    missing_value_table_return = missing_value_table.rename(columns = {0 : 'Missing Value
    cm = sns.light_palette("lightblue", as_cmap=True)
    missing_value_table_return = missing_value_table_return.style.background_gradient(cma
    return missing_value_table_return

missing_value_table(data)
```

Out[30]:

	Missing Values	% Value
close_prev	1	0
date	0	0
daily_return	0	0
return_on_investment	0	0
pe_ratio	0	0
price_per_share	0	0
earnings_per_share	0	0
turnover_ratio	0	0
market_cap	0	0
volatility (ATR)	0	0
Daily_Return	0	0
open	0	0
illiqudity	0	0
volume in bln dollor	0	0
liquidity	0	0
Return	0	0
volume	0	0
close	0	0
low	0	0
high	0	0
volatility	0	0

In [31]:

```
data[data.isnull().any(axis=1)].head(2)
```

Out[31]:

	date	open	high	low	close	volume	Return	liquidity	vol bln
1	2018-12-28	51548.65	52546.0	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06	6205.:

1 rows × 21 columns

In [32]:

```
data.loc[data['turnover_ratio'].isnull(), 'turnover_ratio'] = 0.0
```

In [33]:

```
# Handling missing values

def missing_value_table(data):
    missing_value = data.isna().sum().sort_values(ascending=False)
    missing_value_percent = 100 * data.isna().sum()//len(data)
    missing_value_table = pd.concat([missing_value, missing_value_percent], axis=1)
    missing_value_table_return = missing_value_table.rename(columns = {0 : 'Missing Value
    cm = sns.light_palette("lightblue", as_cmap=True)
    missing_value_table_return = missing_value_table_return.style.background_gradient(cma
    return missing_value_table_return

missing_value_table(data)
```

Out[33]:

	Missing Values	% Value
close_prev	1	0
date	0	0
daily_return	0	0
return_on_investment	0	0
pe_ratio	0	0
price_per_share	0	0
earnings_per_share	0	0
turnover_ratio	0	0
market_cap	0	0
volatility (ATR)	0	0
Daily_Return	0	0
open	0	0
illiqudity	0	0
volume in bln dollor	0	0
liquidity	0	0
Return	0	0
volume	0	0
close	0	0
low	0	0
high	0	0
volatility	0	0

In [34]:

```
data[data.isnull().any(axis=1)].head(1)
```

Out[34]:

	date	open	high	low	close	volume	Return	liquidity	vol bln
1	2018-12-28	51548.65	52546.0	51548.65	52444.89	120376312.0	-0.005536	-1.120816e+06	6205.:

1 rows × 21 columns

In [35]:

```
data = data.drop(data.index[0])
data[data.isnull().any(axis=1)].head(1)
```

Out[35]:

date	open	high	low	close	volume	Return	liquidity	volume in bln dollor	illiqudity	...	close_prev

0 rows × 21 columns

In [36]:

```
# Handling missing values

def missing_value_table(data):
    missing_value = data.isna().sum().sort_values(ascending=False)
    missing_value_percent = 100 * data.isna().sum()//len(data)
    missing_value_table = pd.concat([missing_value, missing_value_percent], axis=1)
    missing_value_table_return = missing_value_table.rename(columns = {0 : 'Missing Value
    cm = sns.light_palette("lightblue", as_cmap=True)
    missing_value_table_return = missing_value_table_return.style.background_gradient(cma
    return missing_value_table_return

missing_value_table(data)
```

Out[36]:

	Missing Values	% Value
date	0	0
close_prev	0	0
daily_return	0	0
return_on_investment	0	0
pe_ratio	0	0
price_per_share	0	0
earnings_per_share	0	0
turnover_ratio	0	0
market_cap	0	0
volatility (ATR)	0	0
Daily_Return	0	0
open	0	0
illiqudity	0	0
volume in bln dollor	0	0
liquidity	0	0
Return	0	0
volume	0	0
close	0	0
low	0	0
high	0	0
volatility	0	0

In [37]:

```
data_copy = data.copy(deep=True)
```

In [38]:

```
data.columns
```

Out[38]:

```
Index(['date', 'open', 'high', 'low', 'close', 'volume', 'Return  ',
       'liquidity', 'volume in bln dollor', 'illiqudity', 'Daily_Return',
       'close_prev', 'volatility (ATR)', 'market_cap', 'turnover_ratio',
       'earnings_per_share', 'price_per_share', 'pe_ratio',
       'return_on_investment', 'daily_return', 'volatility'],
      dtype='object')
```

In [39]:

```
data.dtypes
```

Out[39]:

```
date                    datetime64[ns]
open                           float64
high                           float64
low                            float64
close                          float64
volume                         float64
Return                         float64
liquidity                      float64
volume in bln dollor           float64
illiqudity                     float64
Daily_Return                   float64
close_prev                     float64
volatility (ATR)               float64
market_cap                     float64
turnover_ratio                 float64
earnings_per_share             float64
price_per_share                float64
pe_ratio                       float64
return_on_investment           float64
daily_return                   float64
volatility                     float64
dtype: object
```

In [40]:

```
data.sort_values(by='date', ascending = True, inplace = True)
data.head(10)
```

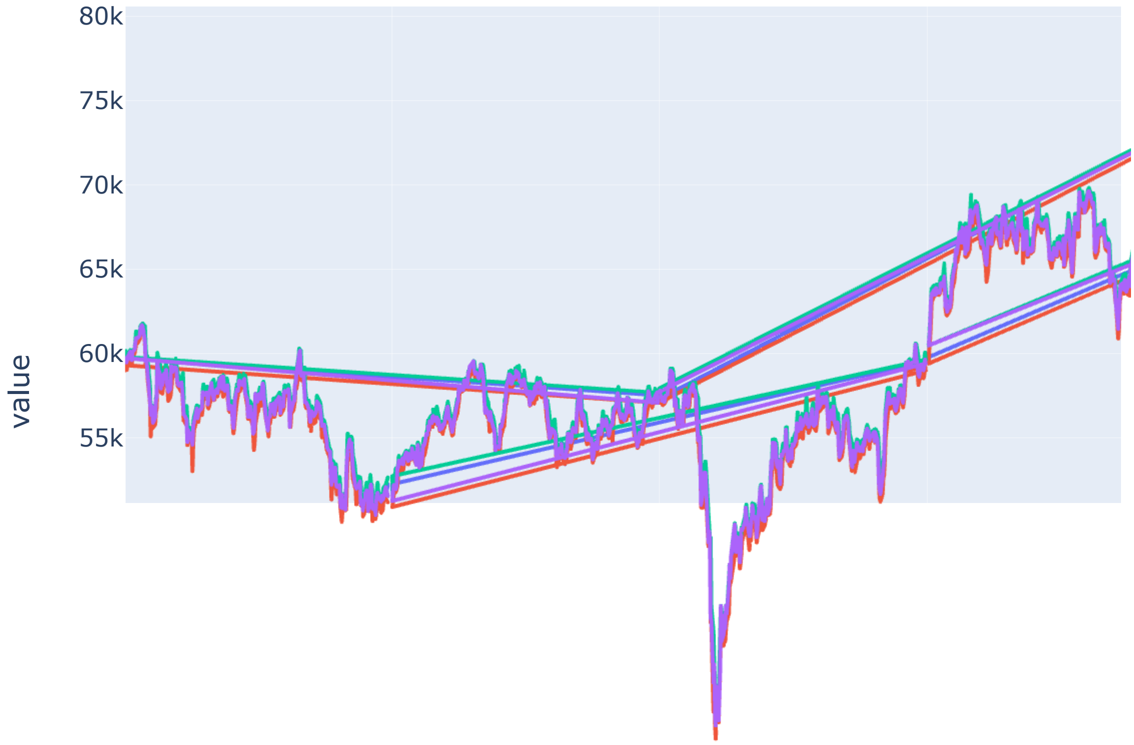
Out[40]:

	date	open	high	low	close	volume	Return	liquidity	
249	2018-01-02	59728.85	59790.28	59308.36	59731.16	149535140.0	0.001703	5.246117e+06	8
248	2018-01-03	60070.74	60150.69	59008.78	59629.64	165492764.0	0.002570	3.867828e+06	9
247	2018-01-04	59569.48	59820.59	59027.01	59476.77	221699935.0	-0.004026	-3.280199e+06	13
246	2018-01-05	59479.19	59835.12	59263.93	59717.20	156615075.0	-0.005350	-1.741268e+06	9
245	2018-01-08	59857.94	60082.44	59604.10	60038.39	173356696.0	-0.001252	-8.288411e+06	10
244	2018-01-09	60116.69	60206.59	59888.40	60113.65	204138679.0	0.002234	5.492304e+06	12
243	2018-01-10	60000.60	60182.70	59689.21	59979.63	483823776.0	0.006268	4.631423e+06	29
242	2018-01-11	59738.20	59789.08	59356.89	59606.02	394137494.0	-0.007941	-2.965063e+06	23
241	2018-01-12	59884.29	60083.13	59732.58	60083.13	165477368.0	-0.002620	-3.782281e+06	9
240	2018-01-15	60155.29	60320.54	60032.71	60240.96	190322679.0	-0.006738	-1.699227e+06	11

10 rows × 21 columns

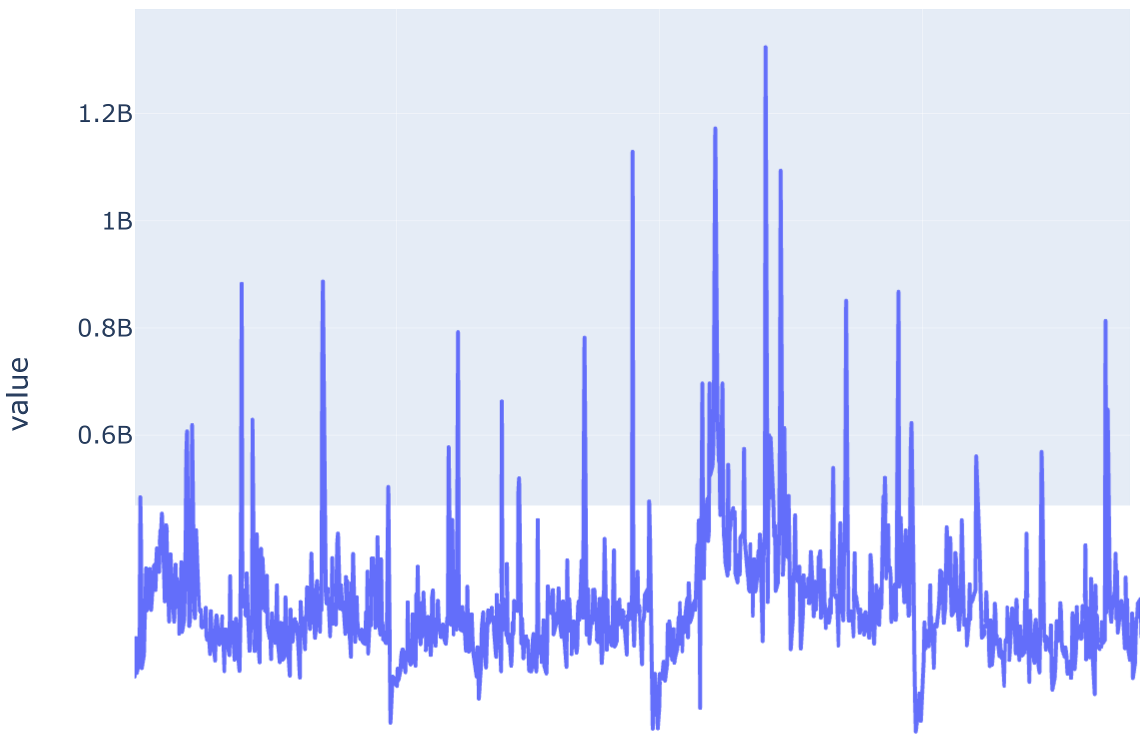
In [41]:

```
pd.options.plotting.backend = "plotly"
data_copy.plot(x='date', y=['open', 'low','high','close'])
```



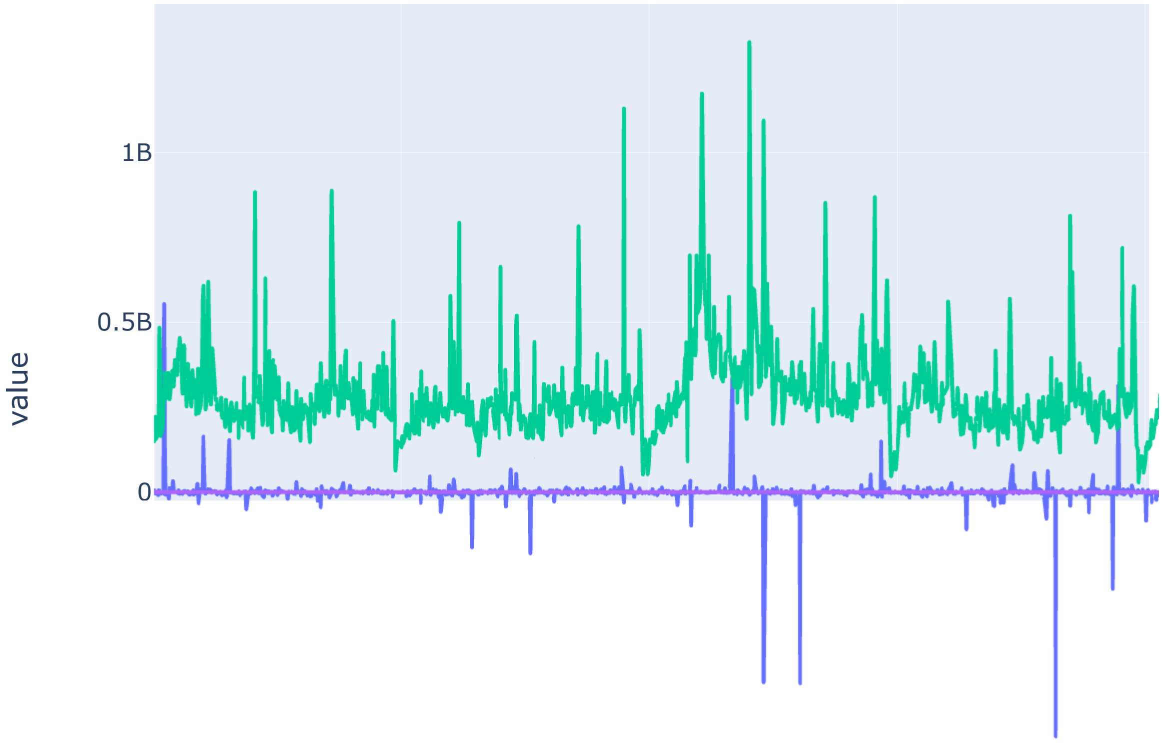
In [42]:

```
data.plot(x='date', y=['volume'], kind='line')
```



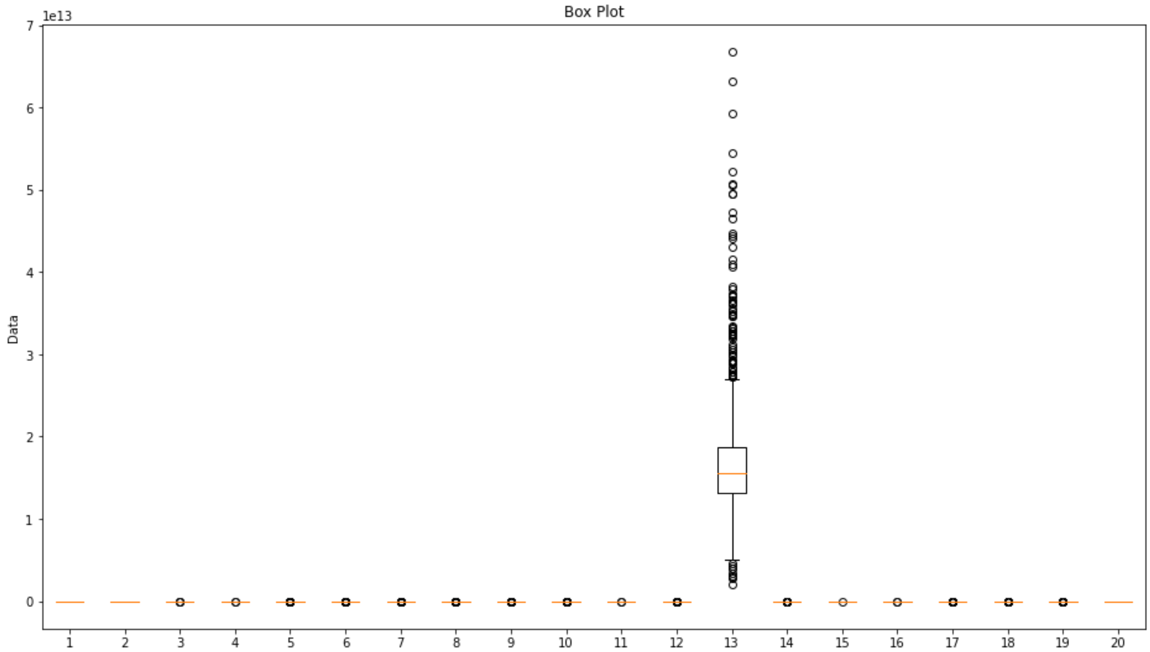
In [43]:

```
data.plot(x='date', y=['liquidity', 'Daily_Return',
       'volume', 'volatility (ATR)'], kind='line')
```



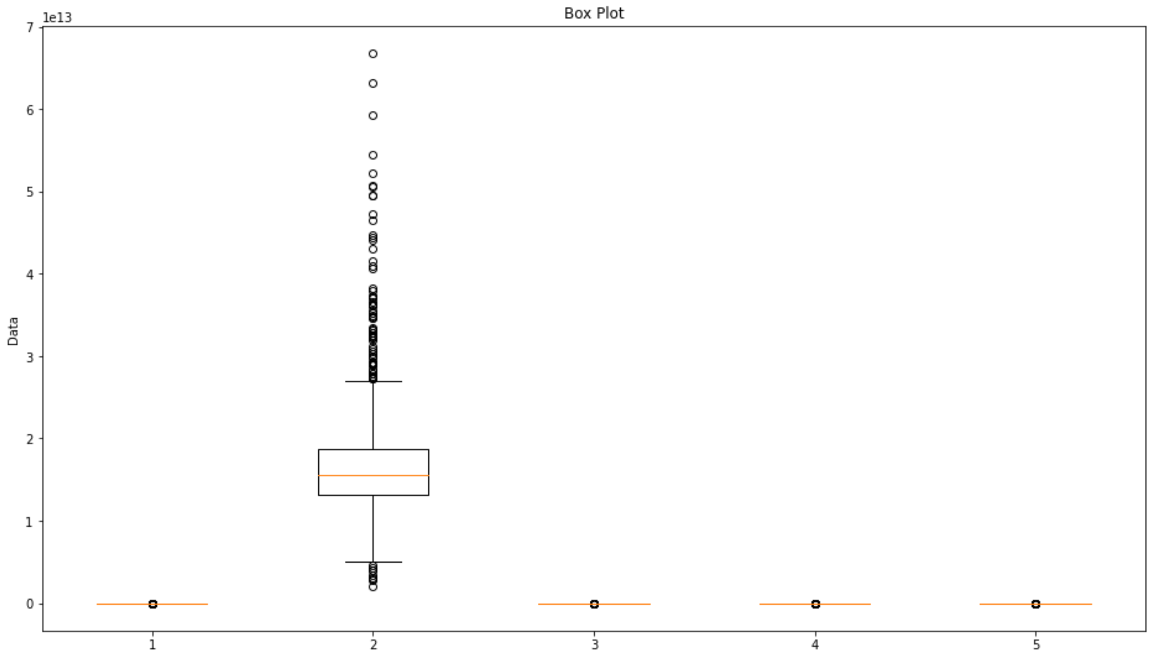
In [44]:

```
fig, ax = plt.subplots(figsize=(16, 9))
ax.boxplot(data.drop(['date'],axis=1))
ax.set_title('Box Plot')
ax.set_ylabel('Data')

plt.show()
```



In [45]:

```
fig, ax = plt.subplots(figsize=(16, 9))
ax.boxplot(data[['liquidity','market_cap','Daily_Return','volume','volatility (ATR)']])
ax.set_title('Box Plot')
ax.set_ylabel('Data')

plt.show()
```



In [46]:

```
data1 = data[['date','liquidity','market_cap','Daily_Return','volume','volatility (ATR)']]
```

In [47]:

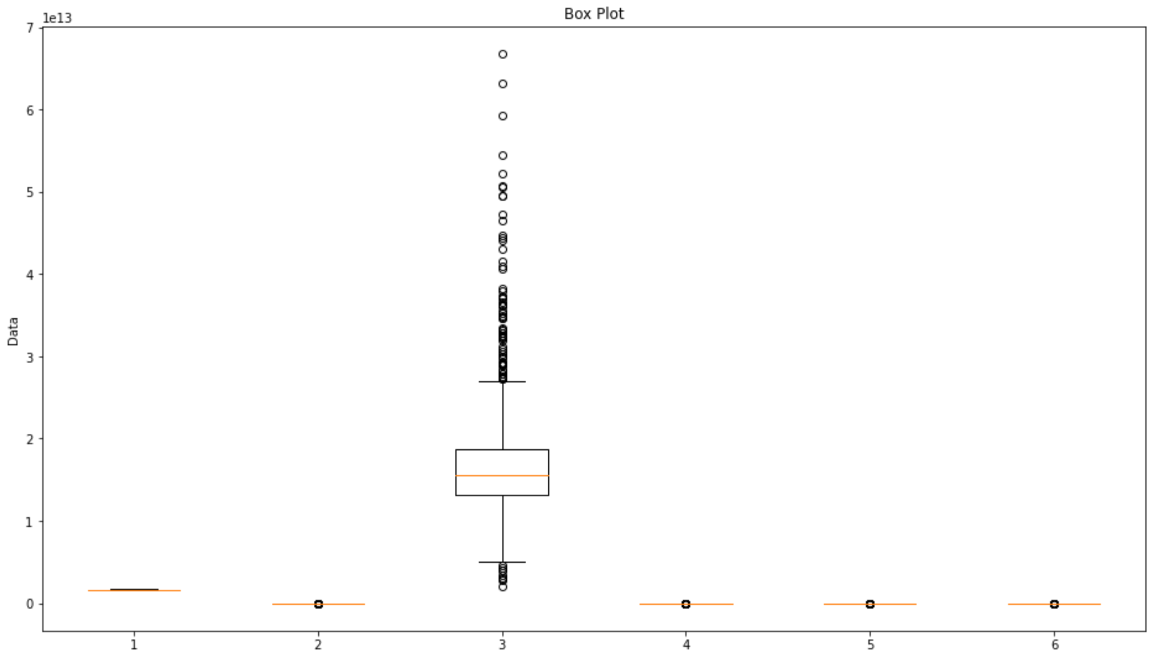
```
data1['date'] = data1['date'].apply(lambda x: int(x.timestamp() * 1000))
data1.head()
```

Out[47]:

	date	liquidity	market_cap	Daily_Return	volume	volatility (ATR)
249	1514851200000	5.246117e+06	8.931907e+12	0.001703	149535140.0	481.92
248	1514937600000	3.867828e+06	9.868274e+12	0.002570	165492764.0	1141.91
247	1515024000000	-3.280199e+06	1.318600e+13	-0.004026	221699935.0	793.58
246	1515110400000	-1.741268e+06	9.352614e+12	-0.005350	156615075.0	774.46
245	1515369600000	-8.288411e+06	1.040806e+13	-0.001252	173356696.0	509.55

In [48]:

```
fig, ax = plt.subplots(figsize=(16, 9))
ax.boxplot(data1)
ax.set_title('Box Plot')
ax.set_ylabel('Data')

plt.show()
```



In [49]:

```
data1 = data1.reset_index(drop=True)
```

In [50]:

```
import pandas as pd
from scipy import stats

# Calculate the Z-scores for each data point in the 'market_cap' column
z_scores = stats.zscore(data1['market_cap'])

# Find the indices of the outliers
outlier_indices = [i for i, z in enumerate(z_scores) if abs(z) > 3]

# Find the indices of the no outliers
nooutliers_indices = [i for i, z in enumerate(z_scores) if abs(z) <= 3]

# Print the outliers
outliers = data1.iloc[outlier_indices]

# dataframe with less outliers
nooutliers = data1.iloc[nooutliers_indices]
```

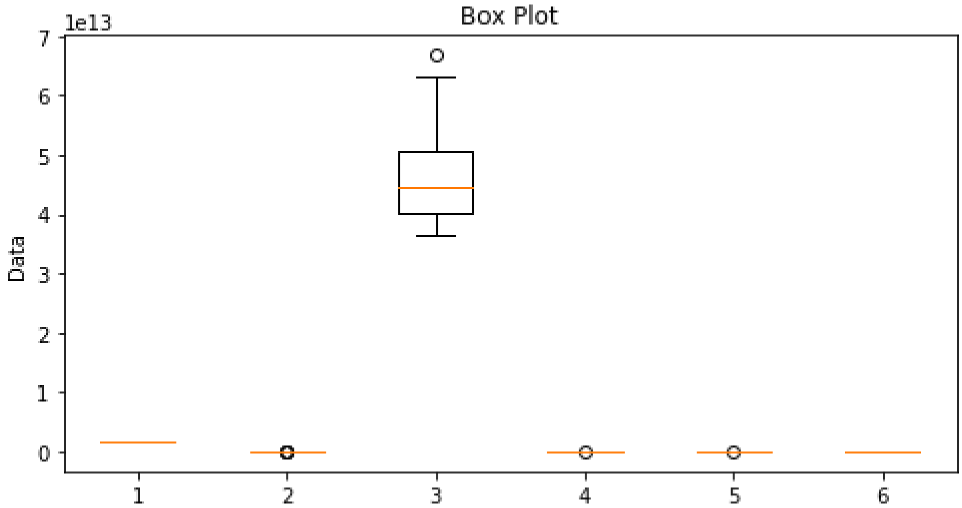
In [51]:

```
print(outliers.shape)
print(data1.shape)
print(nooutliers.shape)
```

```
(24, 6)
(1239, 6)
(1215, 6)
```

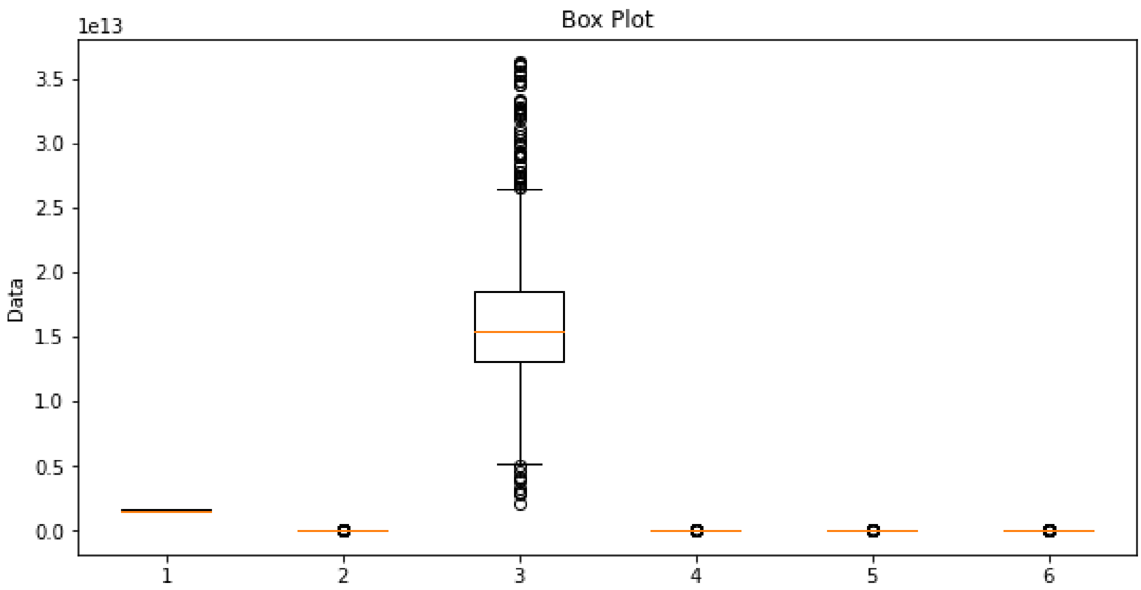
In [52]:

```
fig, ax = plt.subplots(figsize=(8, 4))
ax.boxplot(outliers)
ax.set_title('Box Plot')
ax.set_ylabel('Data')

plt.show()
```



In [53]:

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.boxplot(nooutliers)
ax.set_title('Box Plot')
ax.set_ylabel('Data')

plt.show()
```



In [54]:

```
nooutliers.columns
```

Out[54]:

```
Index(['date', 'liquidity', 'market_cap', 'Daily_Return', 'volume',
       'volatility (ATR)'],
      dtype='object')
```

In [55]:

```
# Calculate the Z-scores for each data point in the 'market_cap' column
z_scores = stats.zscore(data1['liquidity'])

# Find the indices of the outliers
outlier_amihud_liquidity = [i for i, z in enumerate(z_scores) if abs(z) > 3]

# Find the indices of the no outliers
nooutliers_amihud_liquidity = [i for i, z in enumerate(z_scores) if abs(z) <= 3]

# Print the outliers
outliers_amihud_liquidity = data1.iloc[outlier_amihud_liquidity]

# dataframe with less outliers
nooutliers_amihud_liquidity = data1.iloc[nooutliers_amihud_liquidity]
```

In [56]:

```
print(nooutliers_amihud_liquidity.shape)
print(data1.shape)
print(nooutliers_amihud_liquidity.shape)
```

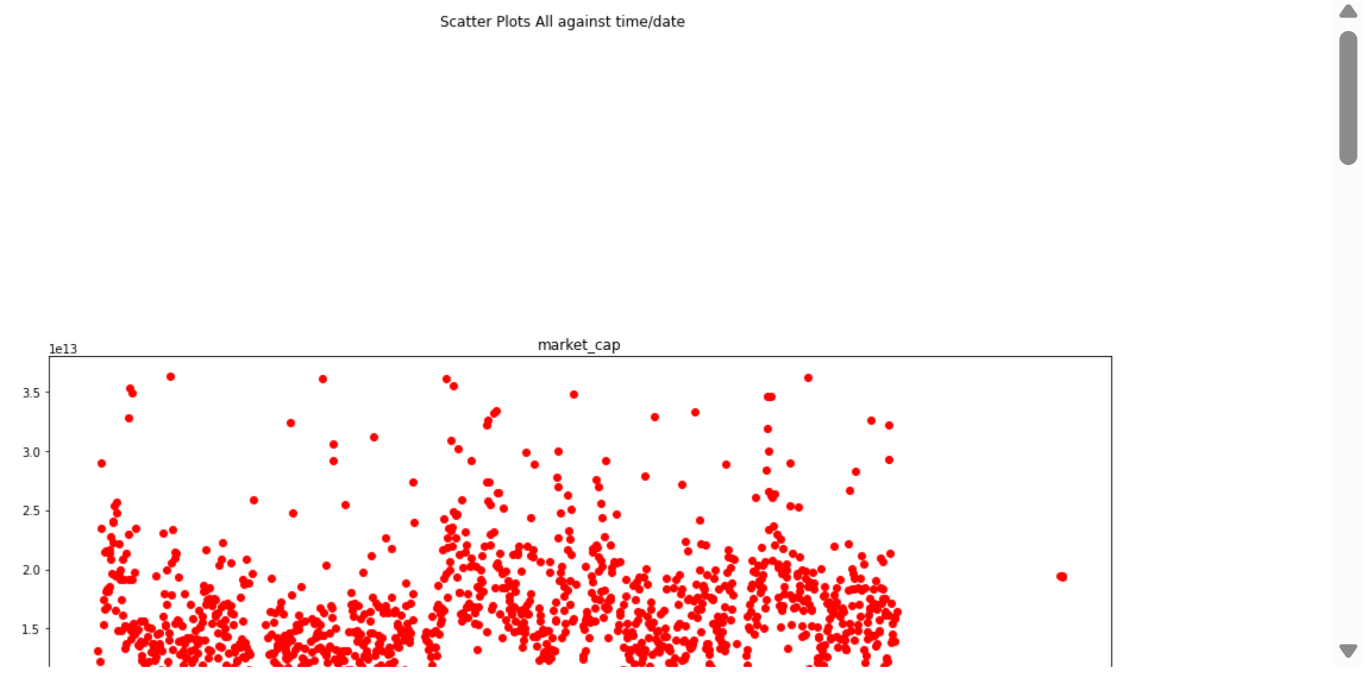
```
(1225, 6)
(1239, 6)
(1225, 6)
```

In [ ]:

```
```

In [57]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Create a 2x2 grid of subplots
fig, axs = plt.subplots(nrows=5, ncols=1, figsize=(16, 40))

# Create a scatter plot on each subplot
axs[0].scatter(nooutliers['date'], nooutliers['market_cap'], color='red')
axs[1].scatter(nooutliers['date'], nooutliers['Daily_Return'], color='blue')
axs[2].scatter(nooutliers['date'], nooutliers['volume'], color='green')
axs[3].scatter(nooutliers['date'], nooutliers['volatility (ATR)'], color='purple')
axs[4].scatter(nooutliers['date'], nooutliers['liquidity'], color='red')

# Add titles to each subplot
axs[0].set_title('market_cap')
axs[1].set_title('Daily_Return')
axs[2].set_title('volume')
axs[3].set_title('volatility (ATR)')
axs[4].set_title('liquidity')

# Add a title to the entire figure
fig.suptitle('Scatter Plots All against time/date')

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



In [58]:

```
# Handling missing values

def missing_value_table(data):
    missing_value = data.isna().sum().sort_values(ascending=False)
    missing_value_percent = 100 * data.isna().sum()//len(data)
    missing_value_table = pd.concat([missing_value, missing_value_percent], axis=1)
    missing_value_table_return = missing_value_table.rename(columns = {0 : 'Missing Value
    cm = sns.light_palette("lightblue", as_cmap=True)
    missing_value_table_return = missing_value_table_return.style.background_gradient(cma
    return missing_value_table_return

missing_value_table(data)
```

Out[58]:

	Missing Values	% Value
date	0	0
close_prev	0	0
daily_return	0	0
return_on_investment	0	0
pe_ratio	0	0
price_per_share	0	0
earnings_per_share	0	0
turnover_ratio	0	0
market_cap	0	0
volatility (ATR)	0	0
Daily_Return	0	0
open	0	0
illiqudity	0	0
volume in bln dollor	0	0
liquidity	0	0
Return	0	0
volume	0	0
close	0	0
low	0	0
high	0	0
volatility	0	0

In [59]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

independent_variables = data.drop(['date'], axis=1)

# compute the VIF for each independent variable
vif = pd.DataFrame()
vif['variable'] = independent_variables.columns
vif['VIF'] = [variance_inflation_factor(independent_variables.values, i) for i in range(i

# display the VIF values
print(vif)
```

```
                 variable           VIF
0                    open  1.753005e+03
1                    high  2.251800e+14
2                     low  3.160421e+13
3                   close  2.573486e+14
4                  volume  4.423641e+01
5                 Return   3.620609e+01
6               liquidity  1.007894e+00
7   volume in bln dollor  6.290487e+03
8              illiqudity  5.247944e+00
9            Daily_Return  3.620609e+01
10             close_prev  2.058113e+03
11       volatility (ATR)  2.044811e+00
12             market_cap  9.001468e+03
13         turnover_ratio  6.914720e+01
14     earnings_per_share  3.002400e+15
15        price_per_share  7.382950e+13
16               pe_ratio  5.227135e-03
17  return_on_investment  5.727958e+01
18           daily_return  5.727958e+01
19             volatility  4.663536e-25
```

In [60]:

```
independent_variables = nooutliers.drop(['date'], axis=1)

# compute the VIF for each independent variable
vif = pd.DataFrame()
vif['variable'] = independent_variables.columns
vif['VIF'] = [variance_inflation_factor(independent_variables.values, i) for i in range(i

# display the VIF values
print(vif)
```

```
           variable        VIF
0         liquidity   1.000568
1        market_cap  57.615716
2      Daily_Return   1.266364
3            volume   5.787186
4  volatility (ATR)   0.931166
```

A VIF of 57.6 for a column in a dataframe indicates that there is a high degree of multicollinearity between this column and the other columns in the dataframe.

In [61]:

```
## Droping market cap
nooutliers = nooutliers.drop(['market_cap'], axis=1)
```

In [62]:

```
independent_variables = nooutliers.drop(['date'], axis=1)

# compute the VIF for each independent variable
vif = pd.DataFrame()
vif['variable'] = independent_variables.columns
vif['VIF'] = [variance_inflation_factor(independent_variables.values, i) for i in range(i

# display the VIF values
print(vif)
```

```
           variable       VIF
0         liquidity  1.001486
1      Daily_Return  1.266290
2            volume  2.944972
3  volatility (ATR)  3.206219
```

In [63]:

```
nooutliers_copy = nooutliers.copy(deep=True)
```

In [64]:

```
nooutliers.drop('date',axis=1).describe()
```

Out[64]:

	liquidity	Daily_Return	volume	volatility (ATR)
count	1.215000e+03	1215.000000	1.215000e+03	1215.000000
mean	-4.278456e+05	0.000330	2.698729e+08	1182.001553
std	3.934045e+07	0.016858	9.173995e+07	880.710013
min	-7.179357e+08	-0.086511	2.912673e+07	267.000000
25%	-2.491595e+06	-0.006664	2.142787e+08	708.780000
50%	-5.224593e+05	-0.000451	2.548277e+08	987.240000
75%	2.419814e+06	0.006213	3.022808e+08	1422.605000
max	5.533378e+08	0.277061	9.360937e+08	16163.560000

In [65]:

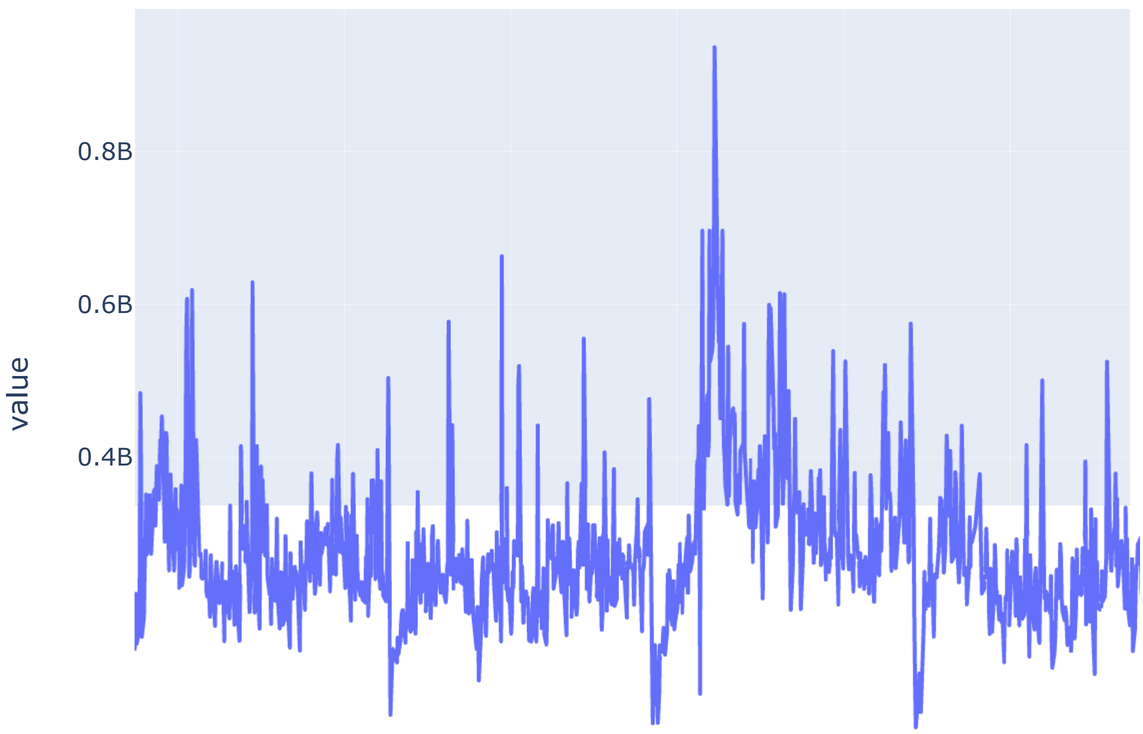
```
nooutliers.columns
```

Out[65]:

```
Index(['date', 'liquidity', 'Daily_Return', 'volume', 'volatility (ATR)'],
dtype='object')
```

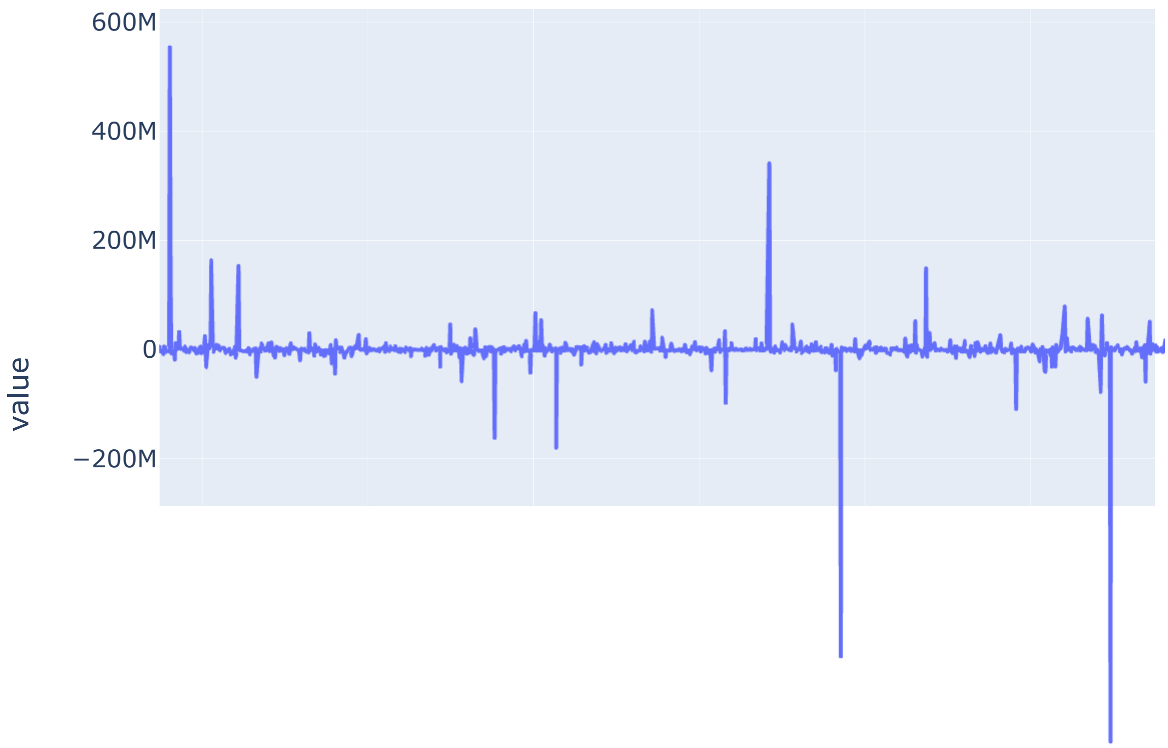
In [66]:

```
nooutliers.plot(x='date', y=['volume'], kind='line')
```



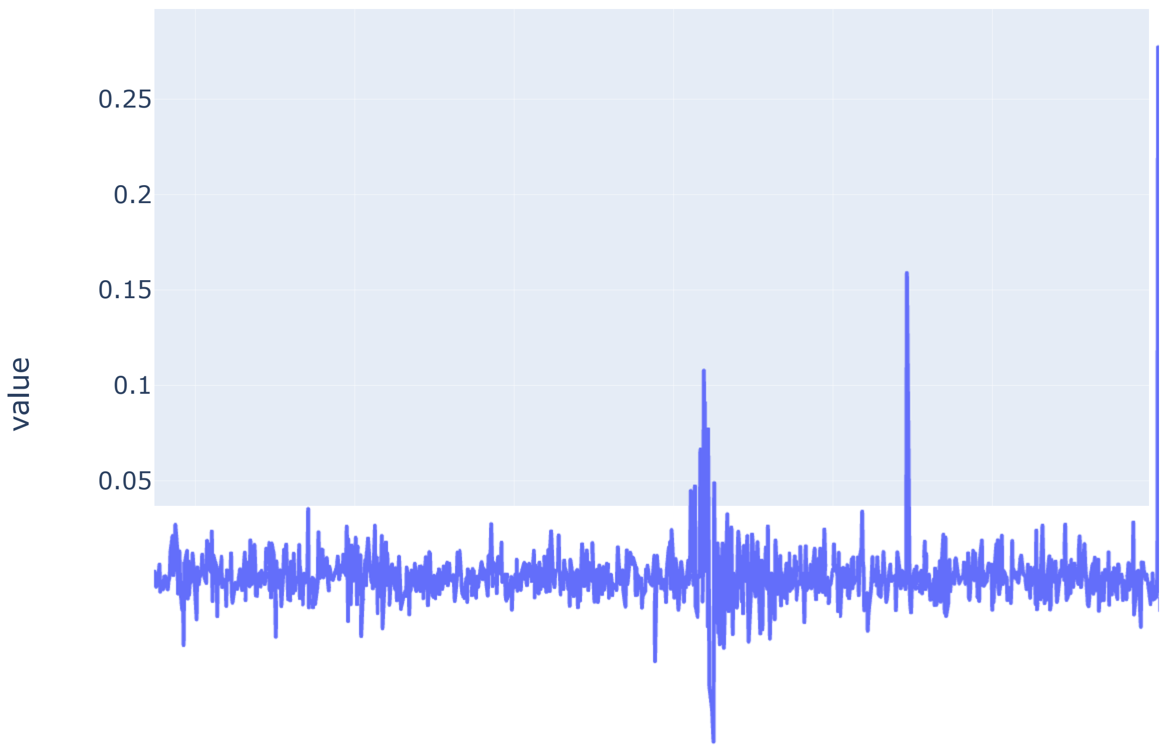
In [67]:

```
nooutliers.plot(x='date', y=['liquidity'], kind='line')
```



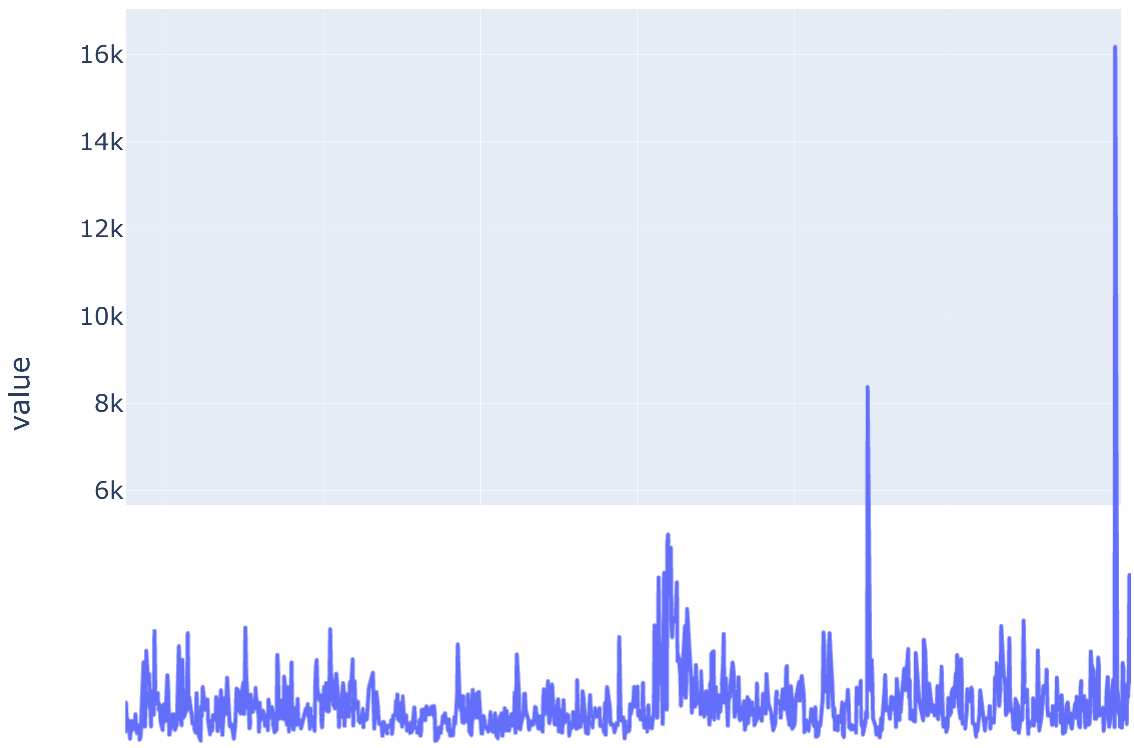
In [68]:

```
nooutliers.plot(x='date', y=['Daily_Return'], kind='line')
```



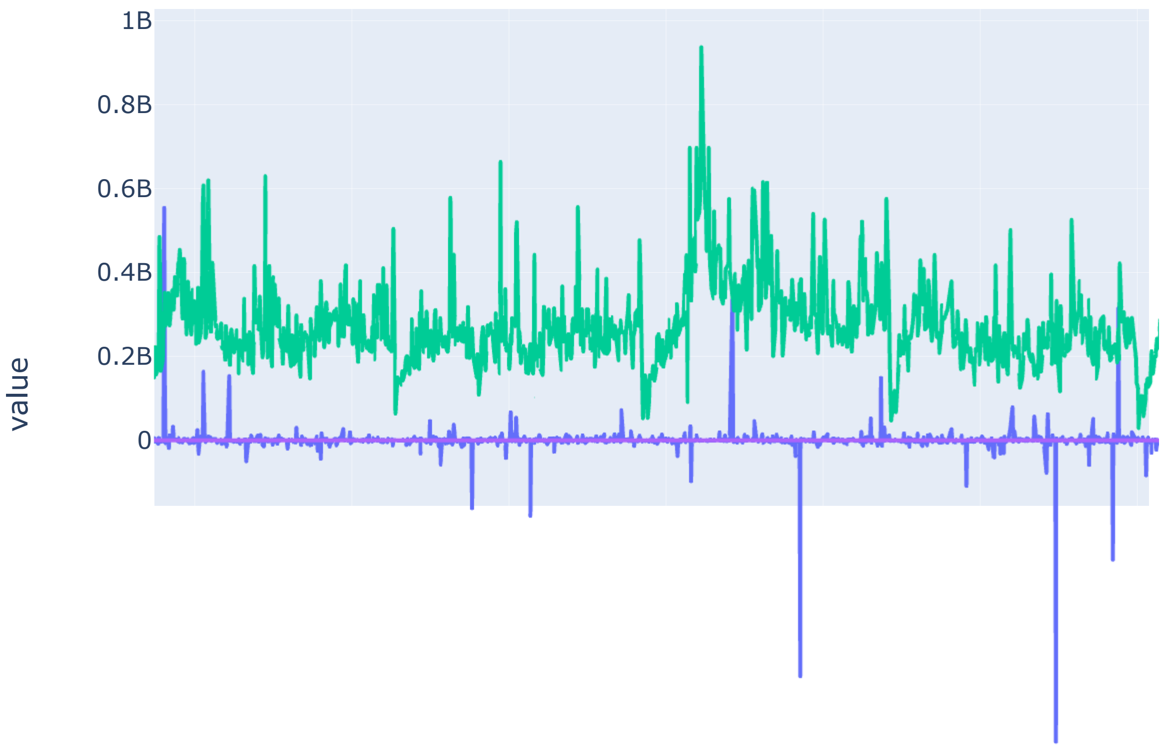
In [69]:

```
nooutliers.plot(x='date', y=['volatility (ATR)'], kind='line')
```



In [70]:

```
nooutliers.plot(x='date', y=['liquidity', 'Daily_Return',
        'volume', 'volatility (ATR)'], kind='line')
```

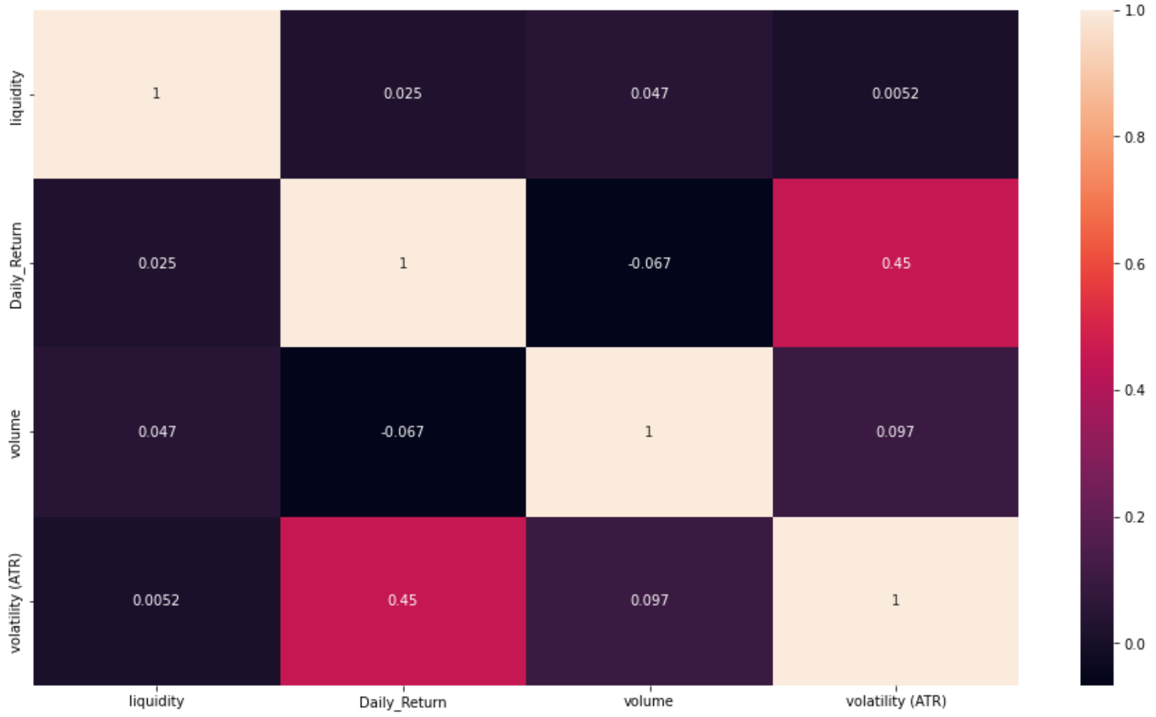


In [71]:

```
## Plotting a correlation table of the collumns in the data frame to check if there is a
plt.figure(figsize=(16,9))
sns.heatmap(nooutliers.drop('date',axis=1).corr(),annot=True)
```

Out[71]:

```
<AxesSubplot:>
```



In [ ]:

```

```

In [72]:

```
nooutliers.columns
```

Out[72]:

```
Index(['date', 'liquidity', 'Daily_Return', 'volume', 'volatility (ATR)'],
dtype='object')
```

In [73]:

```
# Importing additional libraries neccessary for the task at hand

import scipy.stats as ss
from collections import Counter
import math
from matplotlib import pyplot as plt
from scipy import stats
from IPython.display import display, Markdown, Latex
from sklearn.preprocessing import StandardScaler # for standadising the dataset
from statsmodels.tsa.stattools import adfuller # for testing for stationarity
```

In [74]:

```
## STANDADISING THE DATA Frame and normalising it so that the Dataframe will be in the sa

scaler = StandardScaler()
datanew= nooutliers[nooutliers.drop('date',axis=1).columns]
standardized_data = scaler.fit_transform(datanew)
standardized_df = pd.DataFrame(standardized_data, columns=datanew.columns)
# standardized_df['date'] = data['date']
data_new=standardized_df
data_new.head()
```

Out[74]:

	liquidity	Daily_Return	volume	volatility (ATR)
0	0.144287	0.081471	-1.312267	-0.795233
1	0.109237	0.132964	-1.138251	-0.045541
2	-0.072534	-0.258482	-0.525320	-0.441214
3	-0.033400	-0.337028	-1.235061	-0.462933
4	-0.199891	-0.093855	-1.052496	-0.763848

In [ ]:

```

```

In [75]:

```
res = adfuller(data_new[['volume']])
# Printing the statistical result of the adfuller test
print('Augmneted Dickey_fuller Statistic: %f' % res[0])
print('p-value: %f' % res[1])
# printing the critical values at different alpha levels.
print('critical values at different levels:')
for k, v in res[4].items():
    print('\t%s: %.3f' % (k, v))
```

```
Augmneted Dickey_fuller Statistic: -5.255231
p-value: 0.000007
critical values at different levels:
	1%: -3.436
	5%: -2.864
	10%: -2.568
```

The Augmented Dickey-Fuller (ADF) test is a statistical test used to determine if a time series is stationary. A stationary time series is one whose statistical properties do not change over time. The ADF test is a unit root test, which means that it tests for the presence of a unit root in a time series. A unit root is a value that causes the time series to trend over time.

The ADF test statistic is a negative number. The more negative the ADF test statistic, the stronger the evidence against the null hypothesis of a unit root. In this case, the ADF test statistic is -5.255231, which is very negative. This means that there is strong evidence against the null hypothesis of a unit root.

The p-value is the probability of obtaining the observed ADF test statistic if the null hypothesis is true. In this case, the p-value is 0.000007. This means that the probability of obtaining the observed ADF test statistic if the null hypothesis is true is very small.

The critical values are the values of the ADF test statistic that are used to determine if the null hypothesis can be rejected. In your case, the critical values at the 1%, 5%, and 10% levels are -3.436, -2.864, and -2.568, respectively. Since the ADF test statistic is more negative than all of the critical values, the null hypothesis can be rejected at all levels of significance.

The results of the ADF test suggest that the time series is stationary. This means that the statistical properties of the time series do not change over time.

In [76]:

```
res = adfuller(data_new[['liquidity']])
# Printing the statistical result of the adfuller test
print('Augmneted Dickey_fuller Statistic: %f' % res[0])
print('p-value: %f' % res[1])
# printing the critical values at different alpha levels.
print('critical values at different levels:')
for k, v in res[4].items():
    print('\t%s: %.3f' % (k, v))
```

```
Augmneted Dickey_fuller Statistic: -33.372760
p-value: 0.000000
critical values at different levels:
        1%: -3.436
        5%: -2.864
        10%: -2.568
```

In this case, the ADF test statistic is -33.372760, which is extremely negative. This means that there is very strong evidence against the null hypothesis of a unit root.

In this case, the p-value is 0.000000. This means that the probability of obtaining the observed ADF test statistic if the null hypothesis is true is essentially zero.

In this case, the critical values at the 1%, 5%, and 10% levels are -3.436, -2.864, and -2.568, respectively. Since the ADF test statistic is much more negative than all of the critical values, the null hypothesis can be rejected at all levels of significance.

The results of the ADF test suggest that the time series is stationary.

In [77]:

```
res = adfuller(data_new[['Daily_Return']])
# Printing the statistical result of the adfuller test
print('Augmneted Dickey_fuller Statistic: %f' % res[0])
print('p-value: %f' % res[1])
# printing the critical values at different alpha levels.
print('critical values at different levels:')
for k, v in res[4].items():
    print('\t%s: %.3f' % (k, v))
```

```
Augmneted Dickey_fuller Statistic: -34.311168
p-value: 0.000000
critical values at different levels:
        1%: -3.436
        5%: -2.864
        10%: -2.568
```

In this case, the ADF test statistic is -34.311168, which is extremely negative. This means that there is very strong evidence against the null hypothesis of a unit root.

In this case, the p-value is 0.000000. This means that the probability of obtaining the observed ADF test statistic if the null hypothesis is true is essentially zero.

In this case, the critical values at the 1%, 5%, and 10% levels are -3.436, -2.864, and -2.568, respectively.

The results of the ADF test suggest that the time series is stationary.

Specifically, the time series is not trending, and its variance is not changing over time. This means that the time series is a good candidate for forecasting.

In [78]:

```
res = adfuller(data_new[['volatility (ATR)']])
# Printing the statistical result of the adfuller test
print('Augmneted Dickey_fuller Statistic: %f' % res[0])
print('p-value: %f' % res[1])
# printing the critical values at different alpha levels.
print('critical values at different levels:')
for k, v in res[4].items():
    print('\t%s: %.3f' % (k, v))
```

```
Augmneted Dickey_fuller Statistic: -5.167802
p-value: 0.000010
critical values at different levels:
	1%: -3.436
	5%: -2.864
	10%: -2.568
```

In this case, the ADF test statistic is -5.167802, which is very negative. This means that there is strong evidence against the null hypothesis of a unit root.

In this case, the p-value is 0.000010. This means that the probability of obtaining the observed ADF test statistic if the null hypothesis is true is very small.

In this case, the critical values at the 1%, 5%, and 10% levels are -3.436, -2.864, and -2.568, respectively. Since the ADF test statistic is more negative than the critical value at the 1% level, the null hypothesis can be rejected at the 1% level.

The results of the ADF test suggest that the time series is stationary.

Specifically, the time series is not trending, and its variance is not changing over time. This means that the time series is a good candidate for forecasting.

All columns in the dataset are stationary

In [79]:

```
y = data_new['Daily_Return']
```

In [80]:

```
x = data_new.drop(['Daily_Return'],axis=1)
```

In [81]:

```
# with sklearn
### Logic of creating a model

regr = linear_model.LinearRegression()
regr.fit(x, y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)

# with statsmodels
x = sm.add_constant(x) # adding a constant

model = sm.OLS(y, x).fit()
# predictions = model.predict(x)

print(model.summary())
```

```
Intercept:
 -1.427341806553101e-17
Coefficients:
 [ 0.0275643  -0.11249656  0.46206662]
                            OLS Regression Results
==========================================================================
====
Dep. Variable:           Daily_Return   R-squared:
0.217
Model:                            OLS   Adj. R-squared:
0.215
Method:                 Least Squares   F-statistic:                     1
11.7
Date:                Tue, 16 May 2023   Prob (F-statistic):           7.67
e-64
Time:                        02:07:21   Log-Likelihood:                -15
75.6
No. Observations:                1215   AIC:                             3
159.
Df Residuals:                    1211   BIC:                             3
180.
Df Model:                           3
Covariance Type:            nonrobust
==========================================================================
==========
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
----------
const             1.041e-17      0.025   4.09e-16      1.000      -0.050
0.050
liquidity            0.0276      0.025      1.083      0.279      -0.022
0.078
volume              -0.1125      0.026     -4.398      0.000      -0.163
-0.062
volatility (ATR)     0.4621      0.026     18.083      0.000       0.412
0.512
==========================================================================
====
Omnibus:                      264.091   Durbin-Watson:
1.878
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             934
8.173
Skew:                           0.077   Prob(JB):
0.00
Kurtosis:                      16.588   Cond. No.
1.11
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

The intercept is the value of the dependent variable when all of the independent variables are equal to zero. In this case, the intercept is -1.427341806553101e-17, which is essentially zero. This means that if liquidity, volume, and volatility are all equal to zero, the daily return will also be equal to zero.

The coefficients are the slopes of the regression lines for each independent variable. The coefficient for liquidity is 0.0275643, which means that for every 1% increase in liquidity, the daily return is expected to increase by 0.0275643%. The coefficient for volume is -0.11249656, which means that for every 1% increase in volume, the daily return is expected to decrease by 0.11249656%. The coefficient for volatility (ATR) is 0.46206662, which means that for every 1% increase in volatility, the daily return is expected to increase by 0.46206662%.

The R-squared value is a measure of how well the regression line fits the data. In this case, the R-squared value is 0.217, which means that the regression line fits the data 21.7% of the time. This is not a very good fit, but it is not surprising given that the number of observations (1215) is much larger than the number of independent variables (3).

The F-statistic is a measure of the overall significance of the regression model. In this case, the F-statistic is 111.7, which is highly significant. This means that the regression model is significantly better than a model that does not include any independent variables.

The p-values are the probability of obtaining the observed results if the null hypothesis is true. The null hypothesis is that there is no relationship between the independent variables and the dependent variable. In this case, all of the p-values are less than 0.05, which means that we can reject the null hypothesis and conclude that there is a significant relationship between the independent variables and the dependent variable.

The results of this regression analysis suggest that there is a significant relationship between daily return and liquidity, volume, and volatility (ATR). However, the R-squared value is relatively low, which means that there

In [82]:

```
from scipy.stats import chi2_contingency
```

In [83]:

```
# Create a contingency table
table = pd.crosstab(y, [data_new['liquidity'], data_new['volume'],data_new['volatility (A

# Perform the chi-square test
stat, p, dof, expected = chi2_contingency(table)

# Print the results
print("Chi-square statistic:", stat)
print("P-value:", p)
print("Degrees of freedom:", dof)
print("Expected frequencies:", expected)
```

```
Chi-square statistic: 1475009.9999999984
P-value: 0.239689721742271
Degrees of freedom: 1473796
Expected frequencies: [[0.00082305 0.00082305 0.00082305 ... 0.00082305 0.
00082305 0.00082305]
 [0.00082305 0.00082305 0.00082305 ... 0.00082305 0.00082305 0.00082305]
 [0.00082305 0.00082305 0.00082305 ... 0.00082305 0.00082305 0.00082305]
 ...
 [0.00082305 0.00082305 0.00082305 ... 0.00082305 0.00082305 0.00082305]
 [0.00082305 0.00082305 0.00082305 ... 0.00082305 0.00082305 0.00082305]
 [0.00082305 0.00082305 0.00082305 ... 0.00082305 0.00082305 0.00082305]]
```

The "Chi-square statistic" value of 1475009.9999999984 represents the calculated test statistic for the Chi-Square test. This value is used to determine whether there is a significant association between the two variables under consideration.

The "Degrees of freedom" value of 1473796 represents the number of degrees of freedom associated with the test statistic. In this case, the degrees of freedom value is determined by subtracting 1 from the product of the number of levels or categories in each variable.

The "P-value" value of 0.239689721742271 represents the probability of observing a test statistic as extreme as the one obtained, assuming the null hypothesis is true. In other words, it represents the probability that the observed association between the two variables is due to chance.

In this case, since the P-value is greater than the commonly used alpha level of 0.05, we fail to reject the null hypothesis of independence. This means that we do not have sufficient evidence to conclude that there is a significant association between the two variables at the chosen alpha level.

Assuming that the Chi-Square test is used appropriately and all assumptions of the test were met.

In [84]:

```
from scipy.stats import ttest_ind
```

In [85]:

```
data_new.head()
```

Out[85]:

	liquidity	Daily_Return	volume	volatility (ATR)
0	0.144287	0.081471	-1.312267	-0.795233
1	0.109237	0.132964	-1.138251	-0.045541
2	-0.072534	-0.258482	-0.525320	-0.441214
3	-0.033400	-0.337028	-1.235061	-0.462933
4	-0.199891	-0.093855	-1.052496	-0.763848

In [86]:

```
group1 = data_new[data_new['Daily_Return'] > 0]['Daily_Return']
group2 = data_new[data_new['Daily_Return'] <= 0]['Daily_Return']

# Perform the t-test
stat, p = ttest_ind(group1, group2)

# Print the results
print("T-statistic:", stat)
print("P-value:", p)
```

```
T-statistic: 23.343963175658025
P-value: 7.585491690117294e-100
```

The output of a t-test of a single sample mean against a known or hypothesized value.

The "T-statistic" value of 23.343963175658025 represents the calculated test statistic for the t-test. It measures the difference between the sample mean and the known or hypothesized value, expressed in standard error units. This value is compared to a t-distribution with degrees of freedom equal to the sample size minus one to obtain the associated p-value.

The "P-value" value of 7.585491690117294e-100 represents the probability of observing a t-statistic as extreme or more extreme than the one obtained, assuming the null hypothesis is true. In other words, it represents the probability that the observed difference between the sample mean and the hypothesized value is due to chance.

In this case, since the P-value is much less than the commonly used alpha level of 0.05, we reject the null hypothesis and conclude that the sample mean is significantly different from the known or hypothesized value at the chosen alpha level

In [87]:

```
from scipy.stats import mannwhitneyu
```

In [88]:

```
data_new.shape
```

Out[88]:

```
(1215, 4)
```

In [89]:

```
1212/2
```

Out[89]:

```
606.0
```

In [90]:

```
# split the dataframe into two groups
group1 = data_new[data_new.index < 607]
group2 = data_new[data_new.index >= 607]

# perform Mann-Whitney U test
stat, p_value = mannwhitneyu(group1['Daily_Return'], group2['Daily_Return'])

# print the results
print(f"Mann-Whitney U statistic: {stat}")
print(f"P-value: {p_value}")
```

```
Mann-Whitney U statistic: 188711.0
P-value: 0.4940173842223965
```

The output of a Mann-Whitney U test, also known as the Wilcoxon rank-sum test.

The "Mann-Whitney U statistic" value of 188711.0 represents the test statistic for the Mann-Whitney U test. This value is used to determine whether there is a significant difference between two independent groups on a non-parametric measure of central tendency, such as the median.

The "P-value" value of 0.4940173842223965 represents the probability of observing a test statistic as extreme or more extreme than the one obtained, assuming the null hypothesis is true. In other words, it represents the probability that the observed difference between the two groups is due to chance.

In this case, since the P-value is greater than the commonly used alpha level of 0.05, we fail to reject the null hypothesis of no difference between the two groups. This means that we do not have sufficient evidence to conclude that there is a significant difference in the measure of central tendency between the two groups at the chosen alpha level.

Assuming that the Mann-Whitney U test is used appropriately and all assumptions of the test were met.

In [91]:

```
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

# Create Ridge model
ridge_model = Ridge(alpha=1)

# Fit the model
ridge_model.fit(x, y)

# Make predictions
y_pred = ridge_model.predict(x)

# Evaluate model performance
r2 = r2_score(y, y_pred)
mse = mean_squared_error(y, y_pred)
mae = mean_absolute_error(y, y_pred)

mse = mean_squared_error(y, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y, y_pred)
r2_adj = 1 - (1 - r2) * (len(y) - 1) / (len(y) - x.shape[1] - 1)
f_statistic = (r2 / x.shape[1]) / ((1 - r2) / (len(y) - x.shape[1] - 1))
p_value = 1 - stats.f.cdf(f_statistic, x.shape[1], len(y) - x.shape[1] - 1)

print('MSE:', mse)
print('RMSE:', rmse)
print("Mean absolute error:", mae)
print('R-squared:', r2)
print('R-squared adjusted:', r2_adj)
print('F-statistic:', f_statistic)
print('p-value:', p_value)
```

```
MSE: 0.7832955290362326
RMSE: 0.8850398460161173
Mean absolute error: 0.5880552122034969
R-squared: 0.21670447096376744
R-squared adjusted: 0.21411506425620963
F-statistic: 83.68885055069349
p-value: 1.1102230246251565e-16
```

"MSE" stands for "mean squared error" and measures the average squared difference between the actual values and the predicted values. A smaller MSE indicates a better fit of the model to the data.

"RMSE" stands for "root mean squared error" and is the square root of the MSE. This is a more interpretable metric, as it's on the same scale as the target variable. A smaller RMSE indicates a better fit of the model to the data.

"Mean absolute error" measures the average absolute difference between the actual values and the predicted values. It is also a measure of the accuracy of the model, and a smaller value indicates a better fit.

"R-squared" is a measure of the proportion of variance in the dependent variable that is explained by the independent variables. It ranges from 0 to 1, with higher values indicating a better fit of the model to the data.

"R-squared adjusted" is a version of R-squared that adjusts for the number of predictors in the model. It penalizes the addition of predictors that do not improve the fit of the model.

"F-statistic" is a measure of how well the model fits the data. It is the ratio of the mean square of the regression (explained variance) to the mean square of the residuals (unexplained variance). A larger F-statistic indicates a better fit of the model to the data.

"p-value" is the probability of observing a test statistic as extreme or more extreme than the one obtained, assuming the null hypothesis is true. In this case, the null hypothesis is that all the coefficients in the model are zero, indicating no relationship between the independent variables and the dependent variable. A small p-value indicates that the null hypothesis can be rejected, and there is evidence of a relationship between the independent variables and the dependent variable.

the MSE is 0.7832955290362326, which means that the average squared difference between the observed values and the predicted values is 0.7832955290362326. The RMSE is 0.8850398460161173, which means that the average absolute difference between the observed values and the predicted values is 0.8850398460161173. The mean absolute error is 0.5880552122034969, which means that the average absolute difference between the observed values and the predicted values is 0.5880552122034969. The R-squared is 0.21670447096376744, which means that 21.670447096376744% of the variation in the dependent variable is explained by the independent variables. The R-squared adjusted is 0.21411506425620963, which means that 21.411506425620963% of the variation in the dependent variable is explained by the independent variables after taking into account the number of independent variables in the model. The F-statistic is 83.68885055069349, which is highly significant. This means that the independent variables in the model are significantly different from zero. The p-value is 1.1102230246251565e-16, which is essentially zero. This means that the probability of obtaining the observed results if the null hypothesis is true is essentially zero.

The results of this regression analysis suggest that the independent variables in the model are significantly different from zero and that they explain a significant amount of the variation in the dependent variable.

In [92]:

```
data_new.head()
```

Out[92]:

	liquidity	Daily_Return	volume	volatility (ATR)
0	0.144287	0.081471	-1.312267	-0.795233
1	0.109237	0.132964	-1.138251	-0.045541
2	-0.072534	-0.258482	-0.525320	-0.441214
3	-0.033400	-0.337028	-1.235061	-0.462933
4	-0.199891	-0.093855	-1.052496	-0.763848

In [93]:

```
import statsmodels.api as sm

# Load dataset

# Create dependent and independent variables
y = data_new['Daily_Return']
x = data_new[['volume', 'volatility (ATR)','liquidity']]

# Add lagged variables to the independent variables
x_lag1 = x.shift(1)
y_lag1 = y.shift(1)

# Remove missing values
y = y[1:]
x = x[1:]
x_lag1 = x_lag1[1:]
y_lag1 = y_lag1[1:]

# Build ADL model
X = sm.add_constant(pd.concat([x, x_lag1, y_lag1], axis=1))
model = sm.OLS(y, X).fit()

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

```
                            OLS Regression Results
==========================================================================
====
Dep. Variable:           Daily_Return   R-squared:
0.249
Model:                            OLS   Adj. R-squared:
0.244
Method:                 Least Squares   F-statistic:                     5
7.06
Date:                Tue, 16 May 2023   Prob (F-statistic):           1.02
e-70
Time:                        02:07:22   Log-Likelihood:                -15
49.4
No. Observations:                1214   AIC:                             3
115.
Df Residuals:                    1206   BIC:                             3
156.
Df Model:                           7
Covariance Type:            nonrobust
==========================================================================
==========
                      coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
----------
const              -0.0003      0.025     -0.014      0.989      -0.049
0.049
volume             -0.0670      0.031     -2.192      0.029      -0.127
-0.007
volatility (ATR)    0.5236      0.027     19.741      0.000       0.472
0.576
liquidity           0.0224      0.025      0.893      0.372      -0.027
0.072
volume             -0.0228      0.030     -0.751      0.453      -0.082
0.037
volatility (ATR)   -0.2122      0.030     -7.050      0.000      -0.271
-0.153
liquidity          -0.0045      0.025     -0.179      0.858      -0.054
0.045
Daily_Return        0.0672      0.028      2.370      0.018       0.012
0.123
==========================================================================
====
Omnibus:                      233.601   Durbin-Watson:
2.020
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             577
1.618
Skew:                          -0.095   Prob(JB):
0.00
Kurtosis:                      13.680   Cond. No.
2.03
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
```

In [94]:

```
# data_new = data_new.dropna()
data_new.head()
```

Out[94]:

	liquidity	Daily_Return	volume	volatility (ATR)
0	0.144287	0.081471	-1.312267	-0.795233
1	0.109237	0.132964	-1.138251	-0.045541
2	-0.072534	-0.258482	-0.525320	-0.441214
3	-0.033400	-0.337028	-1.235061	-0.462933
4	-0.199891	-0.093855	-1.052496	-0.763848

In [95]:

```
data_new.shape
```

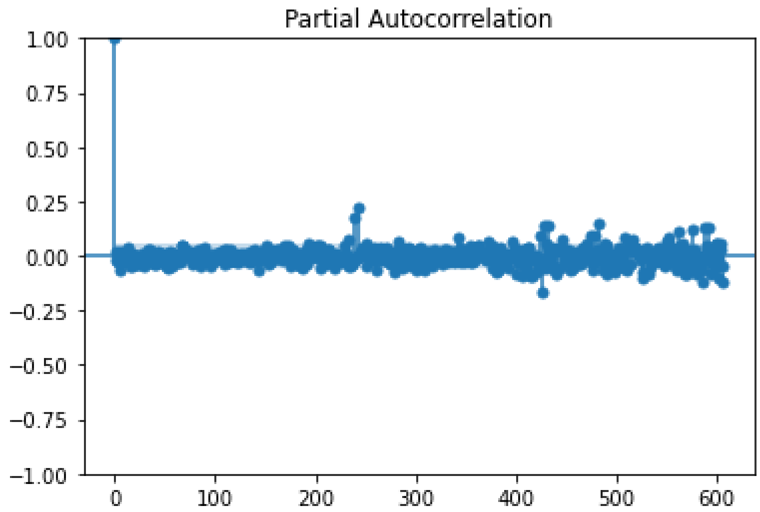
Out[95]:

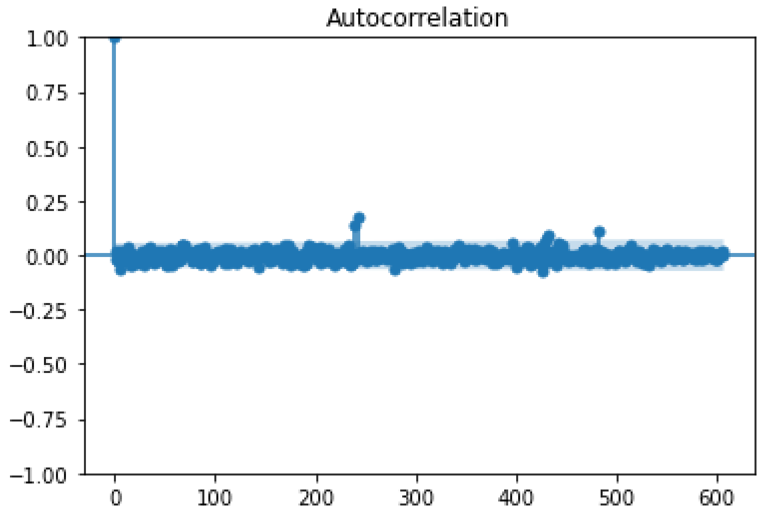
```
(1215, 4)
```

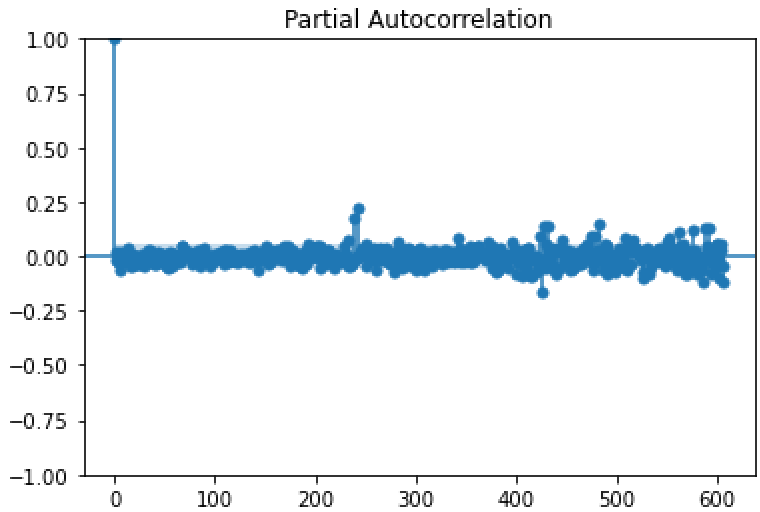
In [96]:

```
import statsmodels.graphics.tsaplots as tsaplots

tsaplots.plot_acf(data_new['Daily_Return'], lags=606)
tsaplots.plot_pacf(data_new['Daily_Return'], lags=606)
```

Out[96]:







In [97]:

```
# Identify the first significant spike in the ACF and the first significant drop in the P
first_significant_spike_acf = 1
first_significant_drop_pacf = 2

# The lag length is the number of lags between the first significant spike in the ACF and
lag_length = first_significant_drop_pacf - first_significant_spike_acf

print(lag_length)
```

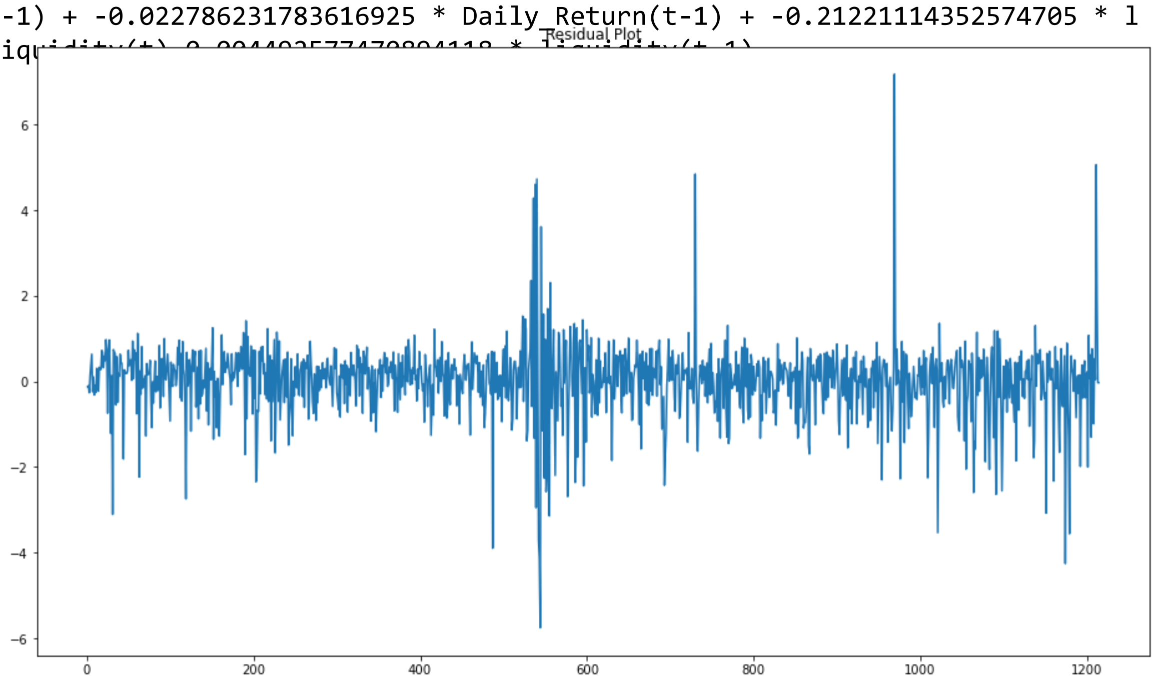
```
1
```

In [98]:

```
# Create dependent and independent variables
y = data_new['Daily_Return']
x = data_new[['volume', 'volatility (ATR)', 'liquidity']]

# Add lagged variables to the independent variables
x_lag1 = x.shift(1)
y_lag1 = y.shift(1)

# Remove missing values
y = y[1:]
x = x[1:]
x_lag1 = x_lag1[1:]
y_lag1 = y_lag1[1:]

# Build ADL model
X = sm.add_constant(pd.concat([x, x_lag1, y_lag1], axis=1))
model = sm.OLS(y, X).fit()

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

# Extract coefficients
beta0 = model.params[0]
beta1 = model.params[1]
beta2 = model.params[2]
beta3 = model.params[3]
beta4 = model.params[4]
beta5 = model.params[5]
beta6 = model.params[6]

# Create lag model equation
equation = "Daily_Return(t) = " + str(beta0) + " + " + str(beta1) + " * volume(t) + " +\
             str(beta2) + " * volatility(t) + " + str(beta3) + " * volume(t-1) + " + str(
print(equation)

residuals = model.resid

# Plot the residual plot
plt.figure(figsize=(16,9))
plt.plot(residuals)
plt.title('Residual Plot')
plt.show()
```

```
                           OLS Regression Results
==========================================================================
====
Dep. Variable:           Daily_Return   R-squared:
0.249
Model:                            OLS   Adj. R-squared:
0.244
Method:                 Least Squares   F-statistic:                     5
7.06
Date:                Tue, 16 May 2023   Prob (F-statistic):           1.02
e-70
Time:                        02:07:26   Log-Likelihood:                -15
49.4
No. Observations:                1214   AIC:                             3
115.
Df Residuals:                    1206   BIC:                             3
156.
Df Model:                           7
Covariance Type:            nonrobust
==========================================================================
==========
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
----------
const               -0.0003      0.025     -0.014      0.989      -0.049
0.049
volume              -0.0670      0.031     -2.192      0.029      -0.127
-0.007
volatility (ATR)     0.5236      0.027     19.741      0.000       0.472
0.576
liquidity            0.0224      0.025      0.893      0.372      -0.027
0.072
volume              -0.0228      0.030     -0.751      0.453      -0.082
0.037
volatility (ATR)    -0.2122      0.030     -7.050      0.000      -0.271
-0.153
liquidity           -0.0045      0.025     -0.179      0.858      -0.054
0.045
Daily_Return         0.0672      0.028      2.370      0.018       0.012
0.123
==========================================================================
====
Omnibus:                      233.601   Durbin-Watson:
2.020
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             577
1.618
Skew:                          -0.095   Prob(JB):
0.00
Kurtosis:                      13.680   Cond. No.
2.03
==========================================================================
====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is cor
rectly specified.
Daily_Return(t) = -0.00033726231540563006 + -0.06703469388058325 * volume
(t) + 0.5236004034283105 * volatility(t) + 0.022382501587492755 * volume(t
```

```
-1) + -0.022786231783616925 * Daily_Return(t-1) + -0.21221114352574705 * l
iquidity(t) 0.004492577470894118 * liquidity(t-1)
```



In [99]:

```python
from statsmodels.stats.stattools import durbin_watson

# Get the Durbin-Watson statistic
dw = durbin_watson(model.resid)

# Print the Durbin-Watson statistic
print('Durbin-Watson statistic:', dw)

# Interpret the Durbin-Watson statistic
if dw < 2:
    print('There is positive autocorrelation in the residuals.')
elif dw > 2:
    print('There is negative autocorrelation in the residuals.')
else:
    print('There is no autocorrelation in the residuals.')
```

```
Durbin-Watson statistic: 2.0204166786412743
There is negative autocorrelation in the residuals.
```

In [100]:

```python
#rename column
data_new = data_new.rename(columns={'volatility (ATR)':'volatility'})
data_new.columns
```

Out[100]:

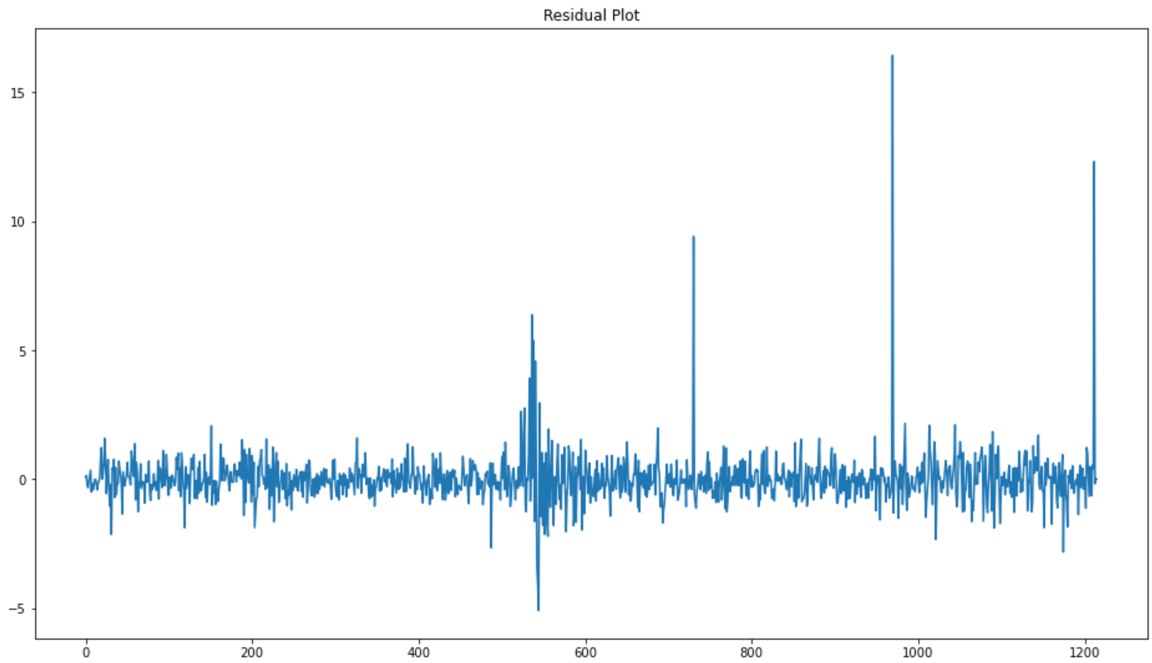
```
Index(['liquidity', 'Daily_Return', 'volume', 'volatility'], dtype='objec
t')
```

reject the null hypothesis of no autocorrelation and conclude that there is negative autocorrelation in the residuals.

In [101]:

```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(data_new['Daily_Return'], order=(1, 0, 0)).fit()
dw = durbin_watson(model.resid)

# Print the Durbin-Watson statistic
print('Durbin-Watson statistic:', dw)

# Interpret the Durbin-Watson statistic
if dw < 2:
    print('There is positive autocorrelation in the residuals.')
elif dw > 2:
    print('There is negative autocorrelation in the residuals.')
else:
    print('There is no autocorrelation in the residuals.')


residuals = model.resid

# Plot the residual plot
plt.figure(figsize=(16,9))
plt.plot(residuals)
plt.title('Residual Plot')
plt.show()
```

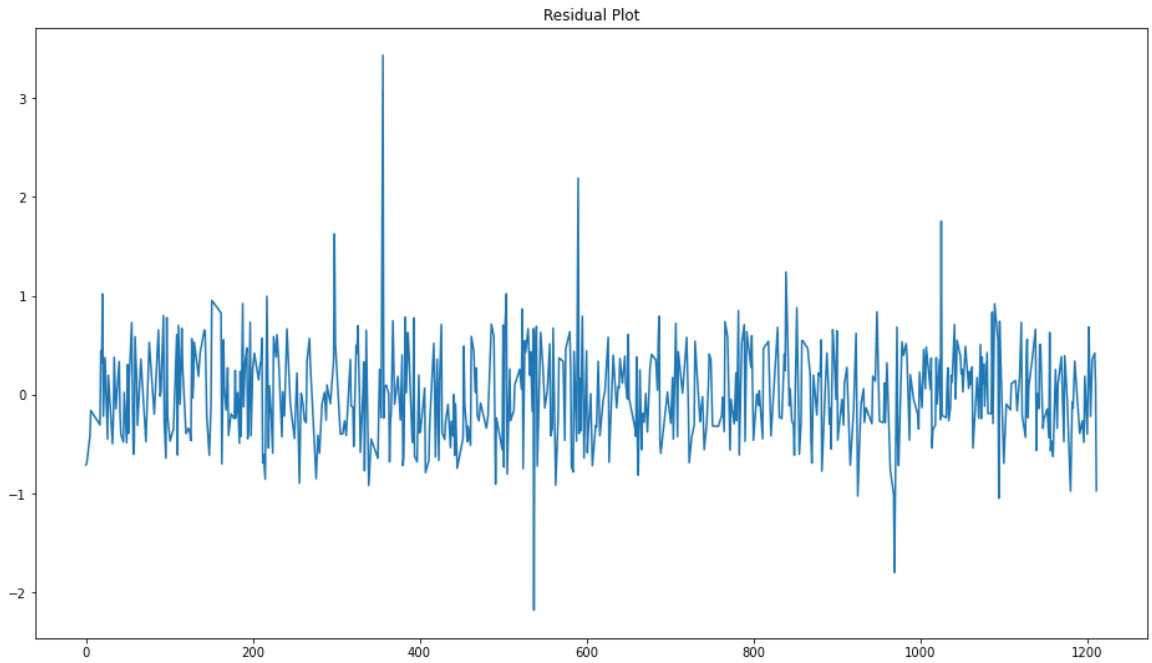
```
Durbin-Watson statistic: 1.9994934554289614
There is positive autocorrelation in the residuals.
```



In [102]:

```
from statsmodels.formula.api import ols

data_new['log_Daily_Return'] = np.log(data_new['Daily_Return'])
# Fit a model to the transformed data
model = ols('log_Daily_Return ~ liquidity + volume + volatility', data=data_new).fit()

# Re-fit the model and evaluate the results
dw = durbin_watson(model.resid)

# Print the Durbin-Watson statistic
print('Durbin-Watson statistic:', dw)

# Interpret the Durbin-Watson statistic
if dw < 2:
    print('There is positive autocorrelation in the residuals.')
elif dw > 2:
    print('There is negative autocorrelation in the residuals.')
else:
    print('There is no autocorrelation in the residuals.')


residuals = model.resid

# Plot the residual plot
plt.figure(figsize=(16,9))
plt.plot(residuals)
plt.title('Residual Plot')
plt.show()
```

```
Durbin-Watson statistic: 2.175442487369284
There is negative autocorrelation in the residuals.
```



In [103]:

```
data_new.columns
```

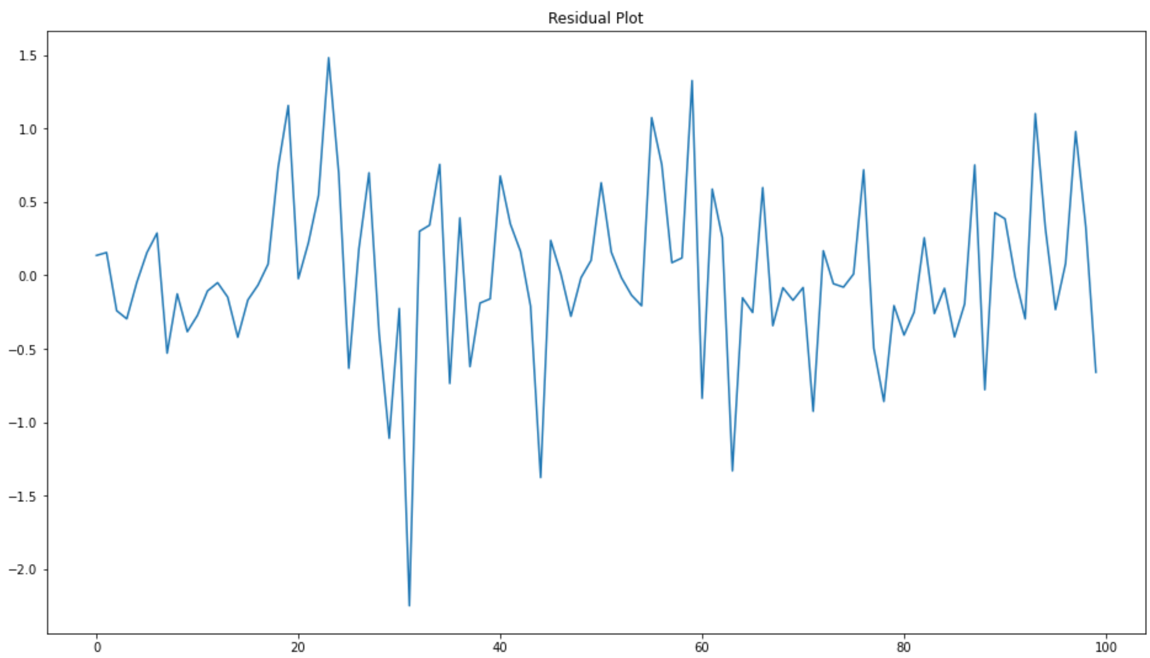
Out[103]:

```
Index(['liquidity', 'Daily_Return', 'volume', 'volatility',
       'log_Daily_Return'],
      dtype='object')
```

In [104]:

```
# Split the data into a training set and a holdout set
train_df = data_new.iloc[:100]
holdout_df = data_new.iloc[100:]

# Fit the ADL model to the training set
model = ols('Daily_Return ~ liquidity + volume + volatility', data=train_df).fit()

# Predict the values of the dependent variable using the fitted model
predicted_values = model.predict(holdout_df)

# Compare the predicted values to the actual values in the holdout set
actual_values = holdout_df['Daily_Return']

# Calculate the MSE
mse = ((predicted_values - actual_values)**2).mean()

# Print the MSE
print('MSE:', mse)

residuals = model.resid

# Plot the residual plot
plt.figure(figsize=(16,9))
plt.plot(residuals)
plt.title('Residual Plot')
plt.show()
```

```
MSE: 1.0326439582575089
```



The MSE of 1.0326439582575089 indicates that the ADL model is not very accurate. The model is able to predict the dependent variable with an average error of 1.0326439582575089. This error is relatively large, so the model may not be reliable for making predictions.

In [ ]:

```

```