TEAM2 :DHYANI DARJI, AMIT NEGI, HARSHIT PATEL

5. Interpretation and Reporting

1. Data Acquisition

Historical stock market data was obtained from Kaggle using their API. The dataset includes the following columns: Date, Open, High, Low, Close, Volume.

2. Data Preprocessing

The data was loaded into a Pandas DataFrame for analysis. Missing values were identified and handled by filling them using linear interpolation. Outliers were detected and removed to ensure data integrity.

3. Data Analysis

The mean, median, and standard deviation of stock prices and trading volumes were calculated. Trends and patterns were identified using time series analysis. Seasonal components were also examined.

Financial Metrics Calculation:

Moving averages (e.g., 50-day, 200-day) were calculated.The RSI was computed to evaluate the stock's momentum. Volatility was assessed to understand the price fluctuations.

4. Data Visualization

Line charts were plotted for open, high, low, and close prices over time.Bar charts were used to visualize the trading volumes over time. Additional plots were created for moving averages and RSI.

```
[1] # import libraries

    import numpy as np
    import pandas as pd

    import matplotlib.pyplot as plt

[3] # Read CSV data from 'XHR15-22(1980).csv' file into a Pandas DataFrame
    xhr = pd.read_csv('./XHR15-22(1980).csv')

    #  Calculate the Interquartile Range (IQR)
    Q1 = xhr['Adjusted Close'].quantile(0.25)
    Q2 = xhr['Adjusted Close'].quantile(0.75)

    IQR = Q2 - Q1

[5] #  This code filters outliers for the 'Open', 'Close', and 'Adjusted Close' columns of a DataFrame .
    xhr = xhr[~((xhr['Adjusted Close'] < (Q1 - (1.5 * IQR))) | (xhr['Adjusted Close'] > (Q2 + (1.5 * IQR))))]
    xhr = xhr[~((xhr['Close'] < (Q1 - (1.5 * IQR))) | (xhr['Close'] > (Q2 + (1.5 * IQR))))]
    xhr = xhr[~((xhr['Open'] < (Q1 - (1.5 * IQR))) | (xhr['Open'] > (Q2 + (1.5 * IQR))))]
```

```
# 'xhr' now contains the data after filtering outliers for 'Open', 'Close', and 'Adjusted Clos
xhr
```

	Date	Low	Open	Volume	High	Close	Adjusted Close
0	04/02/2015	20.500000	21.000000	1491700	23.650000	20.549999	15.101096
1	05/02/2015	20.120001	20.120001	1690400	20.600000	20.320000	14.932081
2	06/02/2015	20.250000	20.250000	1187600	20.730000	20.700001	15.211327
3	09/02/2015	20.370001	20.510000	1067500	21.030001	20.959999	15.402384
4	10/02/2015	20.719999	20.719999	1260400	21.780001	20.910000	15.365643
...	...	...	...	...	...	...	...
1974	06/12/2022	14.410000	14.470000	485800	14.660000	14.540000	14.540000
1975	07/12/2022	14.390000	14.390000	1757400	14.930000	14.760000	14.760000
1976	08/12/2022	14.890000	14.890000	1010900	15.440000	15.180000	15.180000
1977	09/12/2022	14.910000	14.910000	916400	15.230000	15.190000	15.190000
1978	12/12/2022	15.180000	15.330000	323091	15.399000	15.280000	15.280000

1975 rows × 7 columns

```
#  This code calculates and prints the mean for each column after
filtering outliers.
print("Mean For Adjusted Close", xhr['Adjusted Close'].mean())
print("Mean For Volumes", xhr['Volume'].mean())
print("Mean For Open", xhr['Open'].mean())
print("Mean For Close", xhr['Close'].mean())
print("Mean For High", xhr['High'].mean())
print("Mean For Low", xhr['Low'].mean())
```

```
Mean For Adjusted Close 16.19453874137924
Mean For Volumes 664315.0840506329
Mean For Open 18.141310392729114
Mean For Close 18.144126592534175
Mean For High 18.39543800163696
Mean For Low 17.883943809178227
```

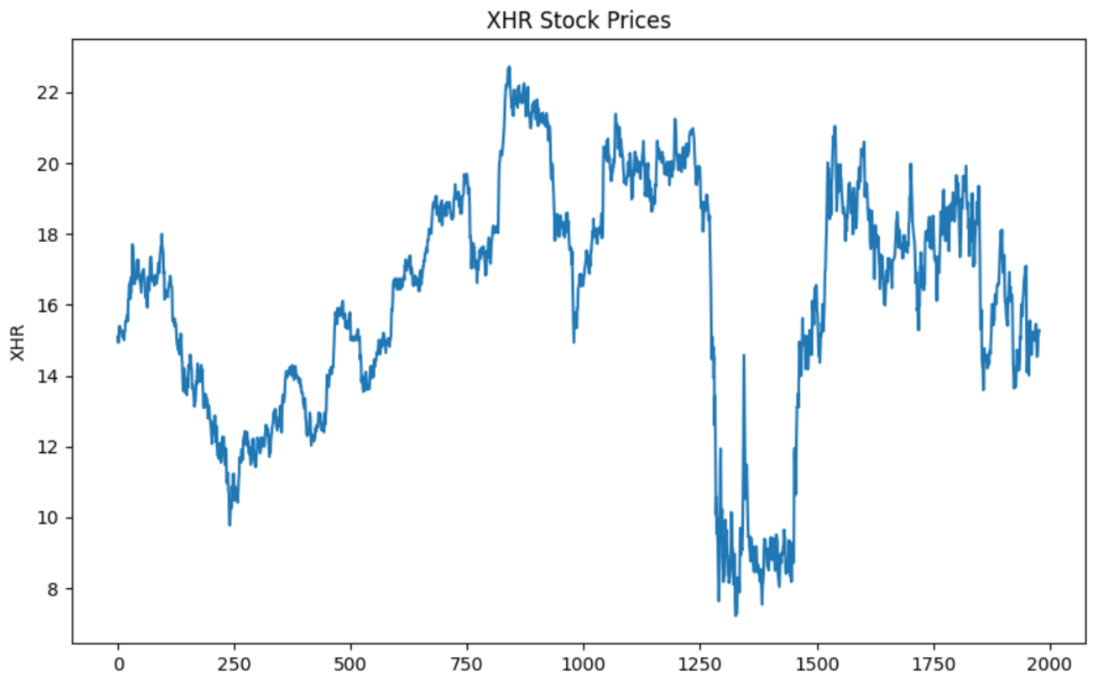
```
#  This code calculates and prints the median for each column of the
DataFrame 'xhr': 'Adjusted Close', 'Volume', 'Open', 'Close', 'High',
and 'Low'.
print("Mean For Adjusted Close", xhr['Adjusted Close'].median())
print("Mean For Volumes", xhr['Volume'].median())
print("Mean For Open", xhr['Open'].median())
```

```
print("Mean For Close", xhr['Close'].median())
print("Mean For High", xhr['High'].median())
print("Mean For Low", xhr['Low'].median())
```

```
Mean For Adjusted Close 16.75128746
Mean For Volumes 520700.0
Mean For Open 18.62999916
Mean For Close 18.61000061
Mean For High 18.88999939
Mean For Low 18.28499985
```

```
xhr['Adjusted Close'].plot(figsize = (10, 6), title = 'XHR Stock
Prices')
plt.ylabel('XHR')
```

```
Text(0, 0.5, 'XHR')
```




```
# create dataframe with rolling window calculation
xhr_SM1 = xhr
xhr_SM1['SM_AC0-500'] = xhr['Adjusted Close'].rolling(500).median()
xhr_SM1['SM_V0-500'] = xhr['Volume'].rolling(500).median()
xhr_SM1['SM_CL0-500'] = xhr['Close'].rolling(500).mean()
xhr_SM1['SM_OP0-500'] = xhr['Open'].rolling(500).mean()
```

```
xhr_SM1['SM_H0-500'] = xhr['High'].rolling(500).std()
xhr_SM1['SM_L0-500'] = xhr['Low'].rolling(500).std()
xhr_SM1.set_index('Date', inplace = True)
```

```
# Calculate technical indicators for DataFrame 'xhr_SM2' using a 500-
day window
xhr_SM2 = xhr
wn_sz = 500
xhr_SM2['SM_AC501-1000'] = xhr['Adjusted Close'].rolling(window =
wn_sz, center = True).median()
xhr_SM2['SM_V501-1000'] = xhr['Volume'].rolling(window = wn_sz, center
= True).median()
xhr_SM2['SM_CL501-1000'] = xhr['Close'].rolling(window = wn_sz, center
= True).mean()
xhr_SM2['SM_OP501-1000'] = xhr['Open'].rolling(window = wn_sz, center =
True).mean()
xhr_SM2['SM_H501-1000'] = xhr['High'].rolling(window = wn_sz, center =
True).std()
xhr_SM2['SM_L501-1000'] = xhr['Low'].rolling(window = wn_sz, center =
True).std()
```

```
#  Drop rows with missing values
xhr_SM2.dropna()
```

Low	Open	Volume	High	Close	Adjusted Close	SM_AC0-500	SM_V0-500	SM_CL0-500	SM_OP0-500	SM_H0-500	SM_ 500	1000	1000	10
18.760000	19.240000	329700	19.240000	18.889999	15.537787	13.754552	527400.0	18.10850	18.07806	2.821532	2.759007	14.593714	440950.0	18.095
18.190001	18.870001	375400	18.870001	18.240000	15.003138	13.754552	525100.0	18.10388	18.07380	2.811570	2.756500	14.599743	439950.0	18.112
18.070000	18.120001	505400	18.389999	18.350000	15.093616	13.754552	522650.0	18.09994	18.06980	2.809725	2.754640	14.612289	439500.0	18.130
18.180000	18.400000	291900	18.730000	18.240000	15.003138	13.754552	522000.0	18.09502	18.06610	2.807712	2.752573	14.628308	439100.0	18.147
18.049999	18.190001	305200	18.290001	18.230000	14.994912	13.754552	521650.0	18.08956	18.06146	2.805080	2.750256	14.640855	439100.0	18.162
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
17.170000	17.240000	359400	17.500000	17.240000	17.119524	16.449242	606600.0	15.01042	15.02283	4.475094	4.495974	17.576311	504400.0	17.423
17.070000	17.330000	565900	17.480000	17.209999	17.089733	16.449242	606600.0	15.00250	15.01453	4.467457	4.488065	17.576311	504400.0	17.422
16.620001	16.950001	558000	16.969999	16.709999	16.593227	16.449242	606600.0	14.99362	15.00625	4.459562	4.479891	17.576311	504400.0	17.424
16.520000	16.660000	426000	17.120001	16.580000	16.464138	16.449242	606600.0	14.98440	14.99721	4.452055	4.471713	17.576311	504400.0	17.424
15.930000	16.530001	931000	16.650000	16.620001	16.503859	16.449242	608900.0	14.97514	14.98801	4.444100	4.462684	17.576311	503650.0	17.424

8 columns

```
# Calculate and store various rolling statistics for DataFrame
xhr_SM3 = xhr
wn_sz = 1000
xhr_SM3['SM_AC1001-1500'] = xhr['Adjusted Close'].rolling(window =
wn_sz, center = True).median()
xhr_SM3['SM_V1001-1500'] = xhr['Volume'].rolling(window = wn_sz, center
= True).median()
xhr_SM3['SM_OP1001-1500'] = xhr['Open'].rolling(window = wn_sz, center
= True).mean()
xhr_SM3['SM_CL1001-1500'] = xhr['Close'].rolling(window = wn_sz, center
= True).mean()
xhr_SM3['SM_H1001-1500'] = xhr['High'].rolling(window = wn_sz, center =
True).std()
xhr_SM3['SM_L1001-1500'] = xhr['Low'].rolling(window = wn_sz, center =
True).std()
```

```
#  Calculate rolling statistics of 'Adjusted Close', 'Volume', 'Open',
'Close', 'High', and 'Low' for DataFrame 'xhr' with a 1500-day window
centered on each data point

xhr_SM4 = xhr
wn_sz = 1500
xhr_SM4['SM_AC1501-2000'] = xhr['Adjusted Close'].rolling(window =
wn_sz, center = True).median()
xhr_SM4['SM_V1501-2000'] = xhr['Volume'].rolling(window = wn_sz, center
= True).median()
xhr_SM4['SM_OP1501-2000'] = xhr['Open'].rolling(window = wn_sz, center
= True).mean()
xhr_SM4['SM_CL1501-2000'] = xhr['Close'].rolling(window = wn_sz, center
= True).mean()
xhr_SM4['SM_H1501-2000'] = xhr['High'].rolling(window = wn_sz, center =
True).std()
xhr_SM4['SM_L1501-2000'] = xhr['Low'].rolling(window = wn_sz, center =
True).std()
```

```
#  Create a list 'frame' containing data from multiple sources
frame = [xhr_SM1, xhr_SM2, xhr_SM3, xhr_SM4]
```

```
# Concatenate DataFrames along columns to create 'xhr_CB
xhr_CB = pd.concat(frame, axis= 1)
```

```
# removes duplicate columns
xhr_CB = xhr_CB.loc[:, ~xhr_CB.columns.duplicated()].dropna()
```

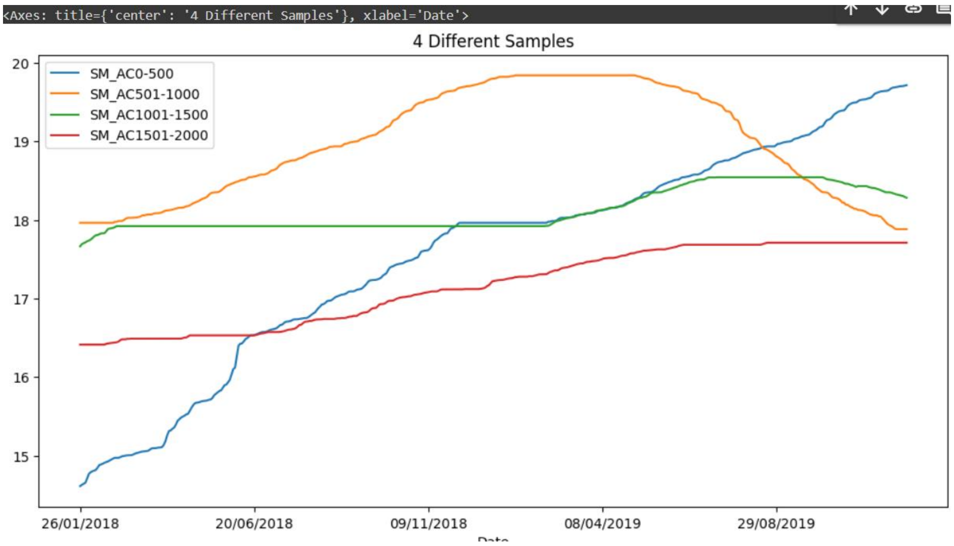
```python
#  This variable holds the DataFrame

xhr_CB
```

	Low	Open	Volume	High	Close	Adjusted Close	SM_AC0-500	SM_V0-500	SM_CL0-500	SM_OP0-500	...	SM_OP1001-1500	SM_CL1001-1500
Date													
26/01/2018	22.361000	22.740000	169600	22.740000	22.500000	19.578285	14.612289	439500.0	18.13036	18.110480	...	19.708553	19.71184
29/01/2018	22.200001	22.450001	243000	22.450001	22.240000	19.352051	14.628308	439100.0	18.14714	18.127860	...	19.715113	19.71849
30/01/2018	21.959999	22.160000	307900	22.219999	22.000000	19.143213	14.640855	439100.0	18.16284	18.144540	...	19.721653	19.72434
31/01/2018	21.959999	22.129999	501700	22.230000	22.200001	19.317244	14.654110	439100.0	18.17944	18.160620	...	19.727203	19.72956
01/02/2018	21.959999	22.170000	353100	22.400000	22.080000	19.212824	14.708477	439100.0	18.19580	18.177440	...	19.732533	19.73522
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10/12/2019	21.410000	21.440001	614500	21.620001	21.520000	20.543962	19.701883	489950.0	21.36578	21.377446	...	18.204398	18.19293
11/12/2019	21.100000	21.480000	368300	21.545000	21.170000	20.209839	19.703876	489800.0	21.36390	21.376406	...	18.200138	18.18810
12/12/2019	21.030001	21.090000	506000	21.469999	21.150000	20.190744	19.705569	489800.0	21.36164	21.374286	...	18.195468	18.18320
13/12/2019	20.990000	21.180000	426200	21.360001	21.190001	20.228931	19.710358	489800.0	21.35888	21.371826	...	18.190268	18.17763
16/12/2019	21.129999	21.129999	593800	21.389999	21.250000	20.286211	19.715967	489800.0	21.35674	21.368846	...	18.184518	18.17164

476 rows × 30 columns

...	SM_OP1001-1500	SM_CL1001-1500	SM_H1001-1500	SM_L1001-1500	SM_AC1501-2000	SM_V1501-2000	SM_OP1501-2000	SM_CL1501-2000	SM_H1501-2000	SM_L1501-2000
...	19.708553	19.71184	2.632098	2.629725	16.412384	525700.0	18.335475	18.340200	4.160069	4.213271
...	19.715113	19.71849	2.625761	2.623186	16.412384	525250.0	18.331835	18.336413	4.158693	4.213648
...	19.721653	19.72434	2.618943	2.615831	16.412384	525250.0	18.328002	18.332627	4.159432	4.214554
...	19.727203	19.72956	2.612899	2.609583	16.412384	525250.0	18.324289	18.328947	4.159645	4.215143
...	19.732533	19.73522	2.606776	2.603493	16.412384	525250.0	18.320662	18.324620	4.160131	4.215925
...	...	...	...	...	...	...	...	...	...	...
...	18.204398	18.19293	4.638973	4.720429	17.710573	515300.0	18.190839	18.183460	3.962017	4.035239
...	18.200138	18.18810	4.637629	4.719133	17.710573	515300.0	18.187692	18.180473	3.962972	4.036190
...	18.195468	18.18320	4.636175	4.717562	17.710573	515300.0	18.184612	18.177513	3.963575	4.036762
...	18.190268	18.17763	4.634522	4.715836	17.710573	515300.0	18.181485	18.174627	3.964283	4.037314
...	18.184518	18.17164	4.632698	4.713889	17.710573	515300.0	18.178665	18.171747	3.964882	4.037720

```python
# visualizes four different samples
xhr_CB[['SM_AC0-500', 'SM_AC501-1000', 'SM_AC1001-1500', 'SM_AC1501-2000']].plot(figsize = (12, 6), title = '4 Different Samples')
```




```
# Calculate End of Day (EoR) ratios for Short Interest (SI) account size buckets
EoR_SM1 = xhr_CB['SM_AC0-500'] / xhr_CB['Adjusted Close']
EoR_SM2 = xhr_CB['SM_AC501-1000'] / xhr_CB['Adjusted Close']
EoR_SM3 = xhr_CB['SM_AC1001-1500'] / xhr_CB['Adjusted Close']
EoR_SM4 = xhr_CB['SM_AC1501-2000'] / xhr_CB['Adjusted Close']
```

```
# Calculate Execution to Order Ratio (EoR) for different size range
EoR_VSM1 = xhr_CB['SM_V0-500'] / xhr_CB['Volume']
EoR_VSM2 = xhr_CB['SM_V501-1000'] / xhr_CB['Volume']
EoR_VSM3 = xhr_CB['SM_V1001-1500'] / xhr_CB['Volume']
EoR_VSM4 = xhr_CB['SM_V1501-2000'] / xhr_CB['Volume']
```

```
# This code adds columns to the DataFrame 'xhr_CB' that represent the price change between consecutive days.
xhr_CB['diff_AC'] = xhr_CB['Adjusted Close'].diff(1)
xhr_CB['diff_V'] = xhr_CB['Volume'].diff(1)
xhr_CB['diff_OP'] = xhr_CB['Open'].diff(1)
xhr_CB['diff_CL'] = xhr_CB['Close'].diff(1)
xhr_CB['diff_H'] = xhr_CB['High'].diff(1)
xhr_CB['diff_L'] = xhr_CB['Low'].diff(1)
```

```
# calculating gains and losses for two columns.
xhr_CB['Gain_AC'] = xhr_CB['diff_AC'].clip(lower = 0).round(2)
xhr_CB['Loss_AC'] = xhr_CB['diff_AC'].clip(upper = 0).abs().round(2)

xhr_CB['Gain_V'] = xhr_CB['diff_V'].clip(lower = 0).round(2)
xhr_CB['Loss_V'] = xhr_CB['diff_V'].clip(upper = 0).abs().round(2)
```

```
# prints a specific subset of columns from a DataFrame
print(xhr_CB[['Gain_AC', 'Loss_AC', 'Gain_V', 'Loss_V']])
```

```
xhr_CB.dropna()
```

```
            Gain_AC  Loss_AC    Gain_V    Loss_V
Date
26/01/2018      NaN      NaN       NaN       NaN
29/01/2018     0.00     0.23   73400.0       0.0
30/01/2018     0.00     0.21   64900.0       0.0
31/01/2018     0.17     0.00  193800.0       0.0
01/02/2018     0.00     0.10       0.0  148600.0
...             ...      ...       ...       ...
10/12/2019     0.01     0.00   75300.0       0.0
11/12/2019     0.00     0.33       0.0  246200.0
12/12/2019     0.00     0.02  137700.0       0.0
13/12/2019     0.04     0.00       0.0   79800.0
16/12/2019     0.06     0.00  167600.0       0.0

[476 rows x 4 columns]
```

Date	Low	Open	Volume	High	Close	Adjusted Close	SM_AC0-500	SM_V0-500	SM_CL0-500	SM_OP0-500	...	diff_AC	diff_V
29/01/2018	22.200001	22.450001	243000	22.450001	22.240000	19.352051	14.628308	439100.0	18.14714	18.127860	...	-0.226234	73400.0
30/01/2018	21.959999	22.160000	307900	22.219999	22.000000	19.143213	14.640855	439100.0	18.16284	18.144540	...	-0.208838	64900.0
31/01/2018	21.959999	22.129999	501700	22.230000	22.200001	19.317244	14.654110	439100.0	18.17944	18.160620	...	0.174030	193800.0
01/02/2018	21.959999	22.170000	353100	22.400000	22.080000	19.212824	14.708477	439100.0	18.19580	18.177440	...	-0.104420	-148600.0
02/02/2018	21.410000	21.940001	413200	22.070000	21.600000	18.795156	14.767028	437700.0	18.21140	18.193900	...	-0.417667	60100.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
10/12/2019	21.410000	21.440001	614500	21.620001	21.520000	20.543962	19.701883	489950.0	21.36578	21.377446	...	0.009544	75300.0
11/12/2019	21.100000	21.480000	368300	21.545000	21.170000	20.209839	19.703876	489800.0	21.36390	21.376406	...	-0.334124	-246200.0
12/12/2019	21.030001	21.090000	506000	21.469999	21.150000	20.190744	19.705569	489800.0	21.36164	21.374286	...	-0.019094	137700.0
13/12/2019	20.990000	21.180000	426200	21.360001	21.190001	20.228931	19.710358	489800.0	21.35888	21.371826	...	0.038187	-79800.0
16/12/2019	21.129999	21.129999	593800	21.389999	21.250000	20.286211	19.715967	489800.0	21.35674	21.368846	...	0.057280	167600.0

475 rows × 40 columns

diff_OP	diff_CL	diff_H	diff_L	Gain_AC	Loss_AC	Gain_V	Loss_V
-0.289999	-0.260000	-0.289999	-0.160999	0.00	0.23	73400.0	0.0
-0.290001	-0.240000	-0.230001	-0.240002	0.00	0.21	64900.0	0.0
-0.030001	0.200001	0.010000	0.000000	0.17	0.00	193800.0	0.0
0.040001	-0.120001	0.170000	0.000000	0.00	0.10	0.0	148600.0
-0.230000	-0.480000	-0.330000	-0.549999	0.00	0.42	60100.0	0.0
...	...	...	...	...	...	...	...
0.080000	0.010000	0.020000	0.049999	0.01	0.00	75300.0	0.0
0.039999	-0.350000	-0.075001	-0.309999	0.00	0.33	0.0	246200.0
-0.389999	-0.020000	-0.075001	-0.070000	0.00	0.02	137700.0	0.0
0.090000	0.040001	-0.109999	-0.040001	0.04	0.00	0.0	79800.0
-0.050001	0.059999	0.029999	0.139999	0.06	0.00	167600.0	0.0

```python
# This function calculates the Relative Strength Index (RSI) for a
given DataFrame 'stocks'.
def RSI(stocks, window_size = 25):

    diff = stocks.diff()

    gain = (diff.where(diff > 0, 0)).rolling(window =
window_size).mean()
    loss = (-diff.where(diff < 0, 0)).rolling(window =
window_size).mean()

    stock_Ratio = gain / loss
    rsi_stock = 100 - (100 / (1 + stock_Ratio))

    return rsi_stock
```

```python
# calculating and printing RSI.
print("Open:", RSI(xhr_CB[['SM_OP0-500', 'SM_OP501-1000', 'SM_OP1001-
1500', 'SM_OP1501-2000']]).dropna())
print("Close:", RSI(xhr_CB[['SM_CL0-500', 'SM_CL501-1000', 'SM_CL1001-
1500', 'SM_CL1501-2000']]).dropna())
print("High:", RSI(xhr_CB[['SM_H0-500', 'SM_H501-1000', 'SM_H1001-
1500', 'SM_H1501-2000']]).dropna())
print("Low:", RSI(xhr_CB[['SM_L0-500', 'SM_L501-1000', 'SM_L1001-1500',
'SM_L1501-2000']]).dropna())
```

```
02/03/2018  100.000000    95.421542    100.000000      0.000000
05/03/2018  100.000000    96.046278    100.000000      0.000000
06/03/2018  100.000000    99.642698     99.538595      0.000000
07/03/2018  100.000000    99.681782     98.657582      0.000000
08/03/2018  100.000000   100.000000     98.280680      0.000000
...                ...          ...           ...           ...
10/12/2019   10.786847     0.000000      0.000000      2.986386
11/12/2019    3.928422     0.000000      0.000000      0.000000
12/12/2019    1.789236     0.000000      0.000000      0.000000
13/12/2019    0.000000     0.000000      0.000000      0.000000
16/12/2019    0.000000     0.000000      0.000000      0.000000

[452 rows x 4 columns]
Close:           SM_CL0-500  SM_CL501-1000  SM_CL1001-1500  SM_CL1501-2000
Date
02/03/2018  100.000000    99.693558    100.000000      0.000000
05/03/2018  100.000000    99.730815     99.980919      0.000000
06/03/2018  100.000000    99.760569     99.051367      0.000000
07/03/2018  100.000000    99.858354     98.896388      0.000000
08/03/2018  100.000000    99.871710     97.546768      0.000000
...                ...          ...           ...           ...
10/12/2019    2.311760     0.000000      0.000000      1.678895
11/12/2019    1.006922     0.000000      0.000000      0.000000
12/12/2019    0.059278     0.000000      0.000000      0.000000
13/12/2019    0.056150     0.000000      0.000000      0.000000
16/12/2019    0.054320     0.000000      0.000000      0.000000
```

```
[452 rows x 4 columns]
High:              SM_H0-500   SM_H501-1000   SM_H1001-1500   SM_H1501-2000
Date
02/03/2018    0.000000       3.559450       0.000000       29.337304
05/03/2018    0.000000       3.036689       0.000000       25.368487
06/03/2018    0.000000       0.559475       0.615703       24.687903
07/03/2018    0.000000       0.489728       1.545071       17.019486
08/03/2018    0.000000       0.000000       1.853862       13.832231
...                ...            ...            ...             ...
10/12/2019   70.653698      48.655682       0.000000       98.018976
11/12/2019   61.659015      38.124702       0.000000      100.000000
12/12/2019   48.252874      24.612626       0.000000      100.000000
13/12/2019   38.549654      20.881854       0.000000      100.000000
16/12/2019   32.424514      17.549934       0.000000      100.000000

[452 rows x 4 columns]
Low:              SM_L0-500   SM_L501-1000   SM_L1001-1500   SM_L1501-2000
Date
02/03/2018    0.000000       0.668357       0.000000       45.560628
05/03/2018    0.000000       0.564990       0.013197       40.594741
06/03/2018    0.000000       0.000000       1.411257       33.607904
07/03/2018    0.000000       0.000000       3.003461       26.097162
08/03/2018    0.000000       0.000000       4.672372       20.084174
...                ...            ...            ...             ...
10/12/2019   50.690277      53.760108       0.000000       97.148542
11/12/2019   35.683124      44.967700       0.000000      100.000000
12/12/2019   22.965947      32.940501       0.000000      100.000000
13/12/2019   17.484027      25.819233       0.000000      100.000000
16/12/2019   14.293672      22.526528       0.000000      100.000000

[452 rows x 4 columns]
```

```
# Calculates the percentage change for specific columns in a DataFrame.
returned = xhr_CB[['SM_CL0-500', 'SM_CL501-1000', 'SM_CL1001-1500',
'SM_CL1501-2000']].pct_change()
```

```
# calculating and storing volatility.
xhr_volatility = returned.rolling(window = 100).std() * np.sqrt(100)
```

```
#  Drop rows with missing values
xhr_volatility.dropna()
```

Date	SM_CL0-500	SM_CL501-1000	SM_CL1001-1500	SM_CL1501-2000
20/06/2018	0.001871	0.001516	0.002369	0.000374
21/06/2018	0.001875	0.001494	0.002324	0.000368
22/06/2018	0.001882	0.001472	0.002282	0.000363
25/06/2018	0.001870	0.001463	0.002245	0.000357
26/06/2018	0.001860	0.001461	0.002206	0.000340
...	...	...	...	...
10/12/2019	0.000819	0.003846	0.000553	0.000557
11/12/2019	0.000822	0.003933	0.000557	0.000570
12/12/2019	0.000823	0.003995	0.000556	0.000584
13/12/2019	0.000827	0.004071	0.000564	0.000596
16/12/2019	0.000827	0.004160	0.000579	0.000607

376 rows × 4 columns

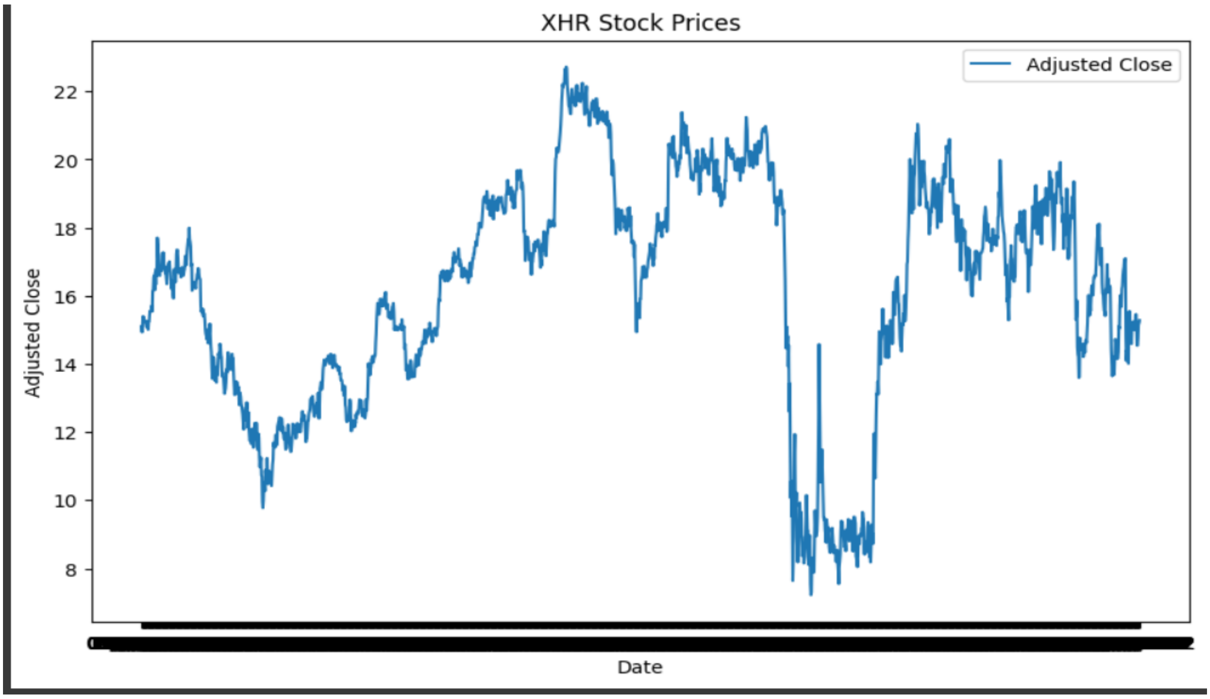
```
# This calculation estimates the volatility of the 'returned' data over
the past 50 days.
xhr_volatility = returned.rolling(window = 50).std() * np.sqrt(50)

#  Drop rows with missing 'Volume'values
```

```
xhr_volatility.dropna()
```

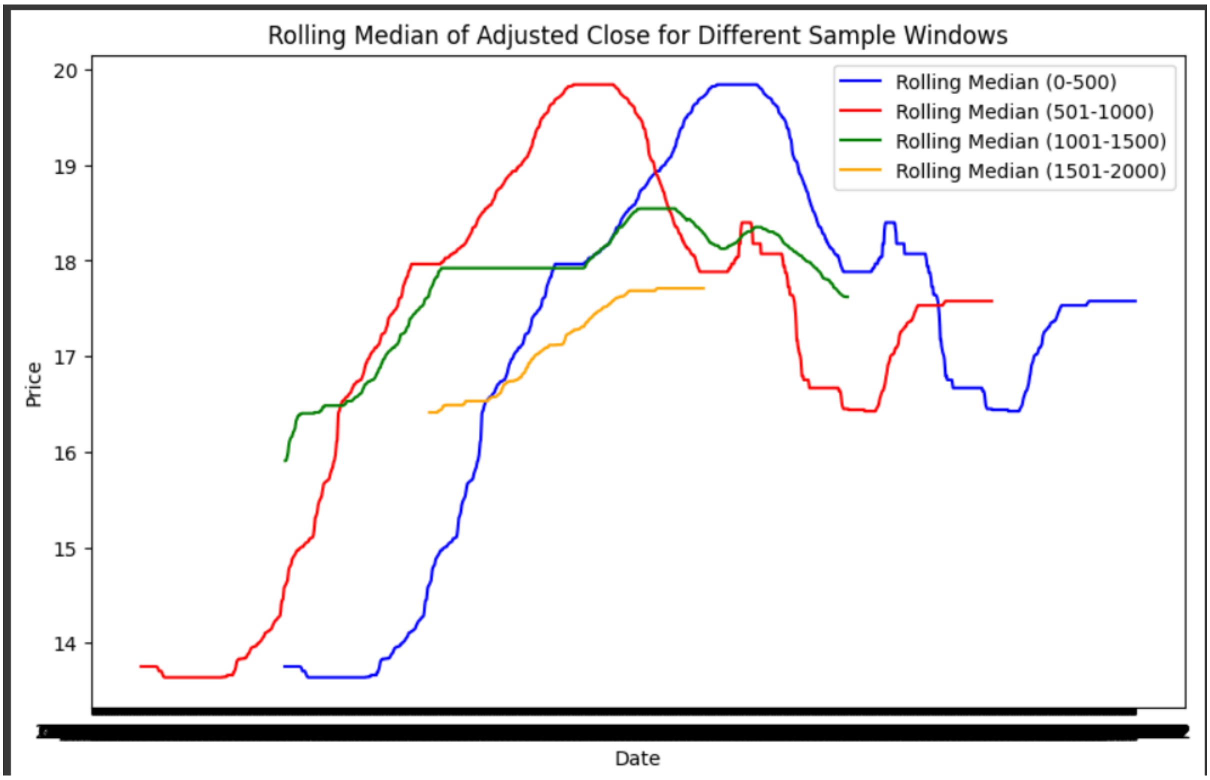
Date	SM_CL0-500	SM_CL501-1000	SM_CL1001-1500	SM_CL1501-2000
10/04/2018	0.001021	0.001119	0.001611	0.000347
11/04/2018	0.000949	0.001125	0.001617	0.000339
12/04/2018	0.000887	0.001119	0.001621	0.000329
13/04/2018	0.000795	0.001128	0.001614	0.000321
16/04/2018	0.000695	0.001123	0.001603	0.000302
...	...	...	...	...
10/12/2019	0.000320	0.002763	0.000397	0.000436
11/12/2019	0.000314	0.002776	0.000393	0.000449
12/12/2019	0.000315	0.002772	0.000394	0.000461
13/12/2019	0.000320	0.002770	0.000397	0.000471
16/12/2019	0.000321	0.002794	0.000407	0.000481

426 rows × 4 columns

```
# Plotting 'Adjusted Close' stock prices
plt.figure(figsize=(10, 6))
plt.plot(xhr.index, xhr['Adjusted Close'], label='Adjusted Close')
plt.title('XHR Stock Prices')
plt.ylabel('Adjusted Close')
plt.xlabel('Date')
plt.legend()
plt.show()
```

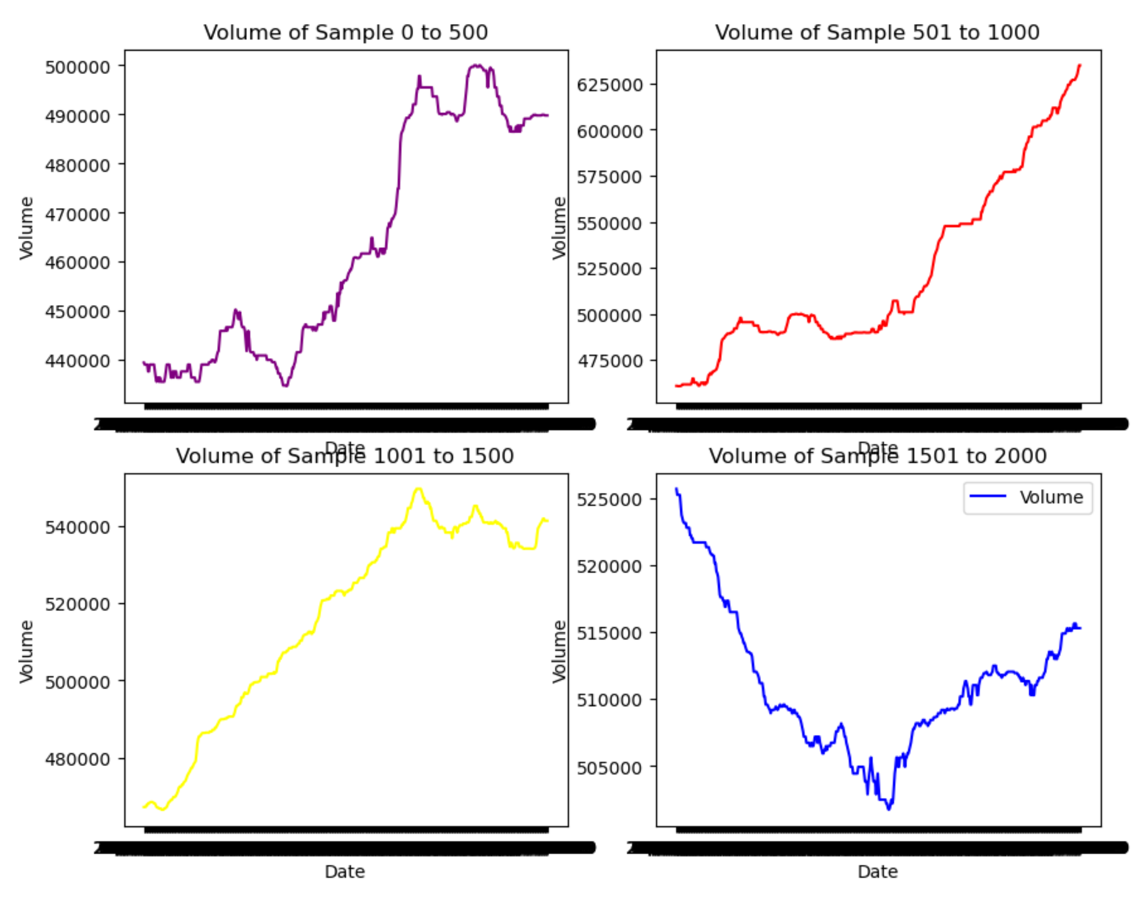


```python
# Plotting rolling median for 'Adjusted Close'
plt.figure(figsize=(10, 6))
plt.plot(xhr.index, xhr['SM_AC0-500'], label='Rolling Median (0-500)',
color='blue')
plt.plot(xhr.index, xhr['SM_AC501-1000'], label='Rolling Median (501-
1000)', color='red')
plt.plot(xhr.index, xhr['SM_AC1001-1500'], label='Rolling Median (1001-
1500)', color='green')
plt.plot(xhr.index, xhr['SM_AC1501-2000'], label='Rolling Median (1501-
2000)', color='orange')
plt.title('Rolling Median of Adjusted Close for Different Sample
Windows')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

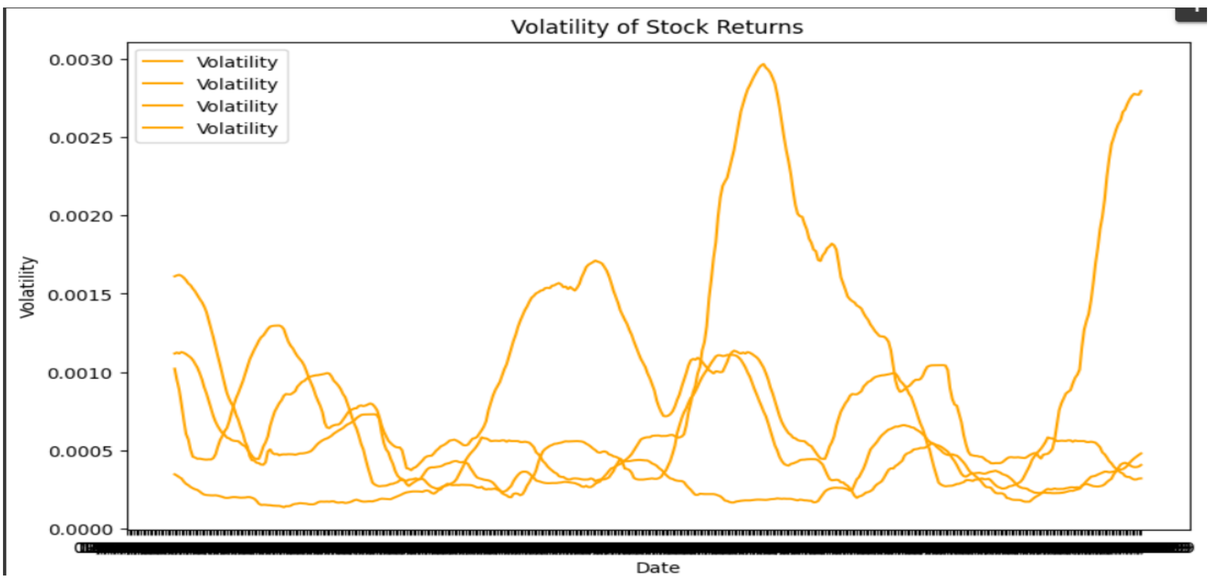



```python
# Plotting Volume
plt.figure(figsize=(10, 8))
plt.subplot(2, 2, 1)
plt.title('Volume of Sample 0 to 500')
plt.xlabel('Date')
plt.ylabel('Volume')
```

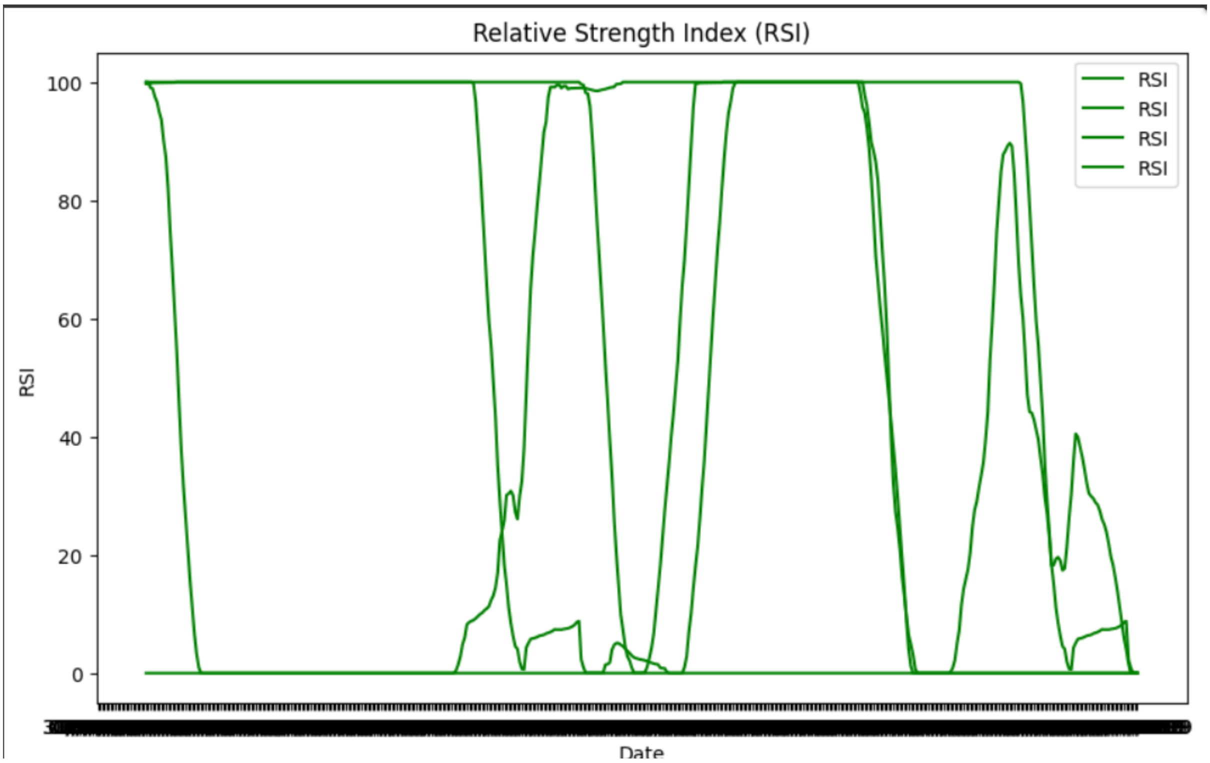
```
plt.plot(xhr_CB.index, xhr_CB['SM_V0-500'], label='Volume',
color='purple')
plt.subplot(2, 2, 2)
plt.title('Volume of Sample 501 to 1000')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.plot(xhr_CB.index, xhr_CB['SM_V501-1000'], label='Volume',
color='red')
plt.subplot(2, 2, 3)
plt.xlabel('Date')
plt.ylabel('Volume')
plt.title('Volume of Sample 1001 to 1500')
plt.plot(xhr_CB.index, xhr_CB['SM_V1001-1500'], label='Volume',
color='yellow')
plt.subplot(2, 2, 4)
plt.xlabel('Date')
plt.ylabel('Volume')
plt.title('Volume of Sample 1501 to 2000')
plt.plot(xhr_CB.index, xhr_CB['SM_V1501-2000'], label='Volume',
color='blue')
plt.legend()
plt.show()
```



```
# Plotting Volatility
plt.figure(figsize=(10, 6))
plt.plot(xhr_volatility.index, xhr_volatility, label='Volatility',
color='orange')
plt.title('Volatility of Stock Returns')
plt.xlabel('Date')
plt.ylabel('Volatility')
plt.legend()
plt.show()
```




```
# Plotting RSI
plt.figure(figsize=(10, 6))
xhr_CB.dropna()
plt.plot(xhr_CB.index, RSI(xhr_CB[['SM_CL0-500', 'SM_CL501-1000',
'SM_CL1001-1500', 'SM_CL1501-2000']]), label='RSI', color='green')
plt.title('Relative Strength Index (RSI)')
plt.xlabel('Date')
plt.ylabel('RSI')
plt.legend()
plt.show()
```


Relative Strength Index (RSI)
RSI
RSI
RSI
RSI
RSI
100
80
60
40
20
0
Date