diversified_portfolio_finance

April 20, 2023

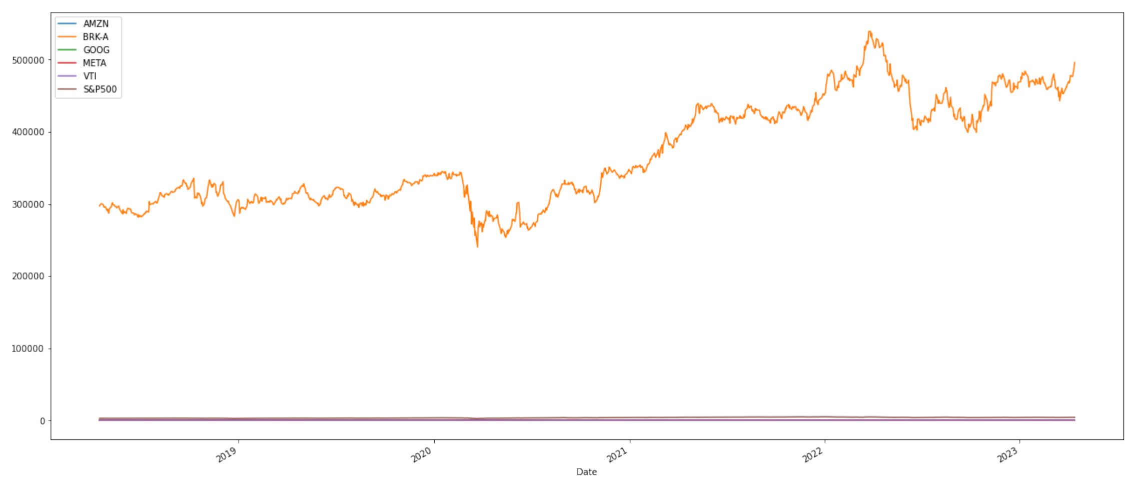
[ ]:
```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
from scipy.stats import norm
from numpy import isinf
from statsmodels.tsa.stattools import grangercausalitytests, adfuller
from statsmodels.tsa.statespace.varmax import VARMAX
from statsmodels.tsa.api import VAR
from sklearn.metrics import mean_squared_error
import math
from statistics import mean
from dateutil.relativedelta import relativedelta
import matplotlib.pyplot as plt
```

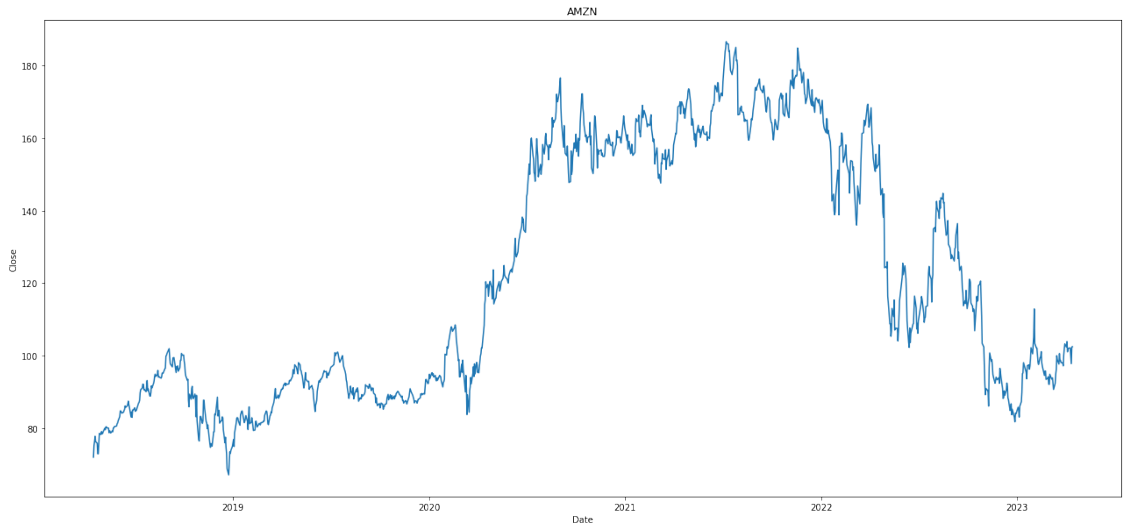
[ ]:
```python
# Step 1: Define the portfolio and gather data
portfolio = ['AMZN','BRK-A','GOOG','META','VTI','S&P500']  #portfolio
weights = [20,5,15,20,20,20]  #weights
```

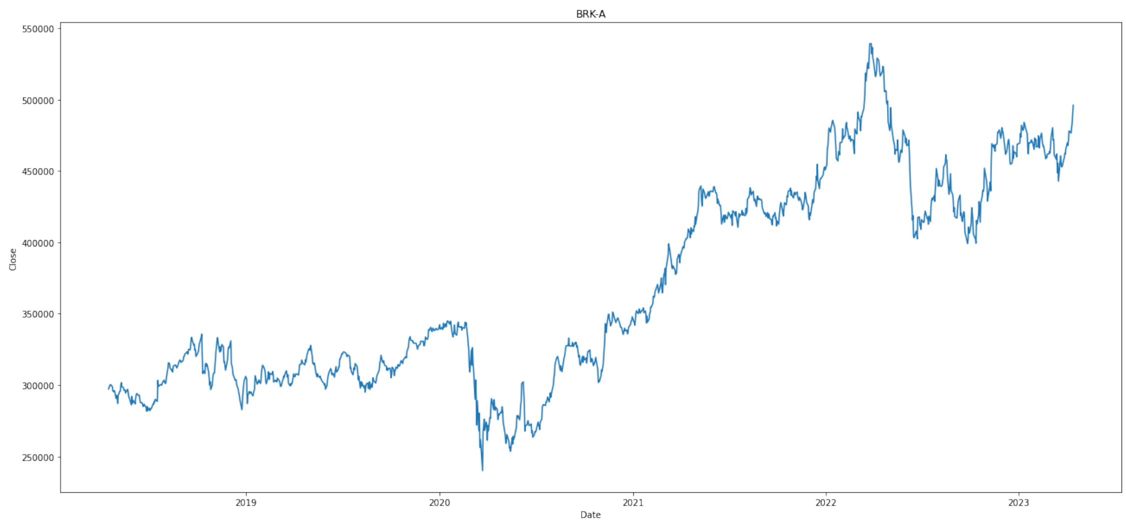
[ ]:
```python
# Retrieve historical prices of the portfolio assets
portfolio_data = pd.DataFrame()
for ticker in portfolio:
    portfolio_data[ticker] = pd.read_csv(f'./portfolio/{ticker}.csv',␣
 ↪index_col='Date', usecols=['Date', 'Close'],
                                 parse_dates=True)['Close']
print(portfolio_data)
portfolio_data.plot(figsize=(22,10))

# Loop through each column and plot a graph
for col in portfolio_data.columns:
    plt.figure(figsize=(22,10))
    plt.plot(portfolio_data[col])
    plt.title(col)
    plt.xlabel('Date')
    plt.ylabel('Close')
    plt.show()

ret_data=pd.DataFrame()
```

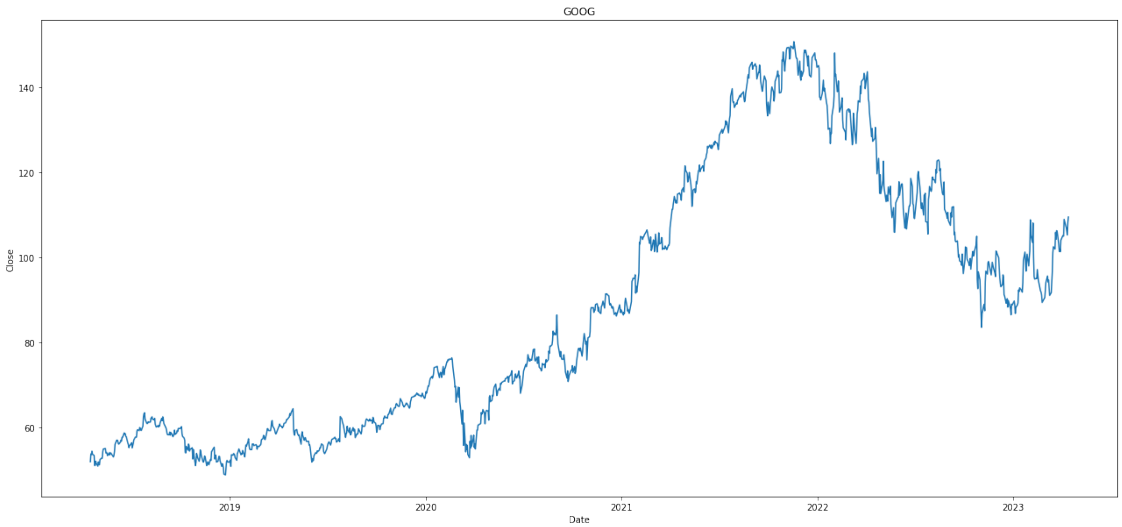
```python
ret_data=portfolio_data-portfolio_data.shift(30)

print(ret_data)
ret_data.plot(figsize=(22,10))

# Loop through each column and plot a graph
for col in ret_data.columns:
    plt.figure(figsize=(22,10))
    plt.plot(ret_data[col])
    plt.title(col)
    plt.xlabel('Date')
    plt.ylabel('Close')
    plt.show()
```

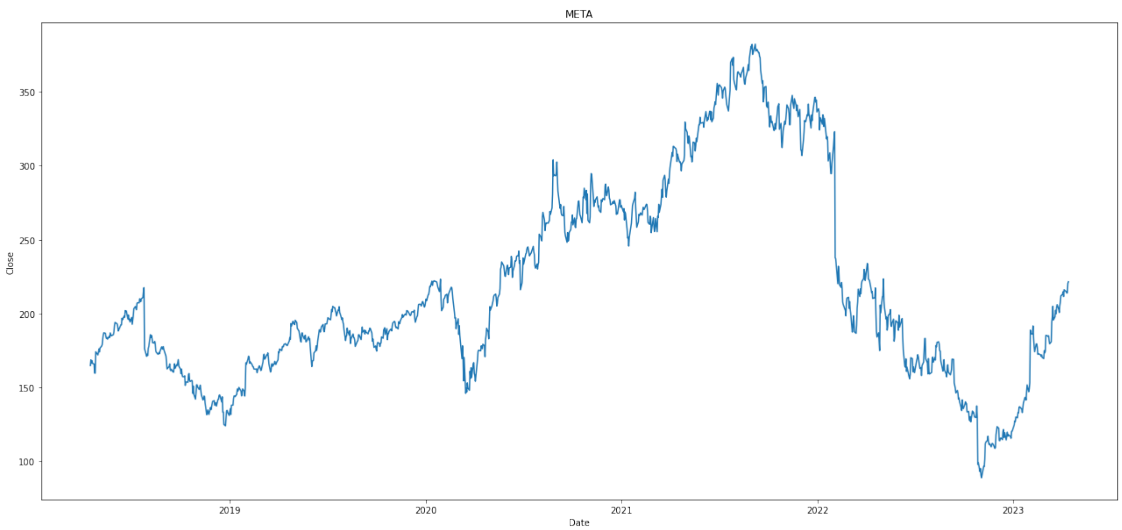
```
                  AMZN     BRK-A        GOOG        META         VTI    S&P500
Date
2018-04-16   72.074997  297181.0   51.898998  164.830002  137.720001  2677.84
2018-04-17   75.191498  298701.0   53.708000  168.660004  139.169998  2706.39
2018-04-18   76.391998  299205.0   53.604000  166.360001  139.360001  2708.64
2018-04-19   77.845497  300300.0   54.384998  168.100006  138.600006  2693.13
2018-04-20   76.374496  300140.0   53.647999  166.279999  137.500000  2670.14
…                  …        …           …          …           …        …
2023-04-10  102.169998  476500.0  106.949997  214.750000  203.660004  4109.11
2023-04-11   99.919998  480800.0  106.120003  213.850006  203.869995  4108.94
2023-04-12   97.830002  483500.0  105.220001  214.000000  203.009995  4091.95
2023-04-13  102.400002  490760.0  108.190002  220.350006  205.649994  4146.22
2023-04-14  102.510002  496000.0  109.459999  221.490005  205.080002  4137.64

[1259 rows x 6 columns]
```

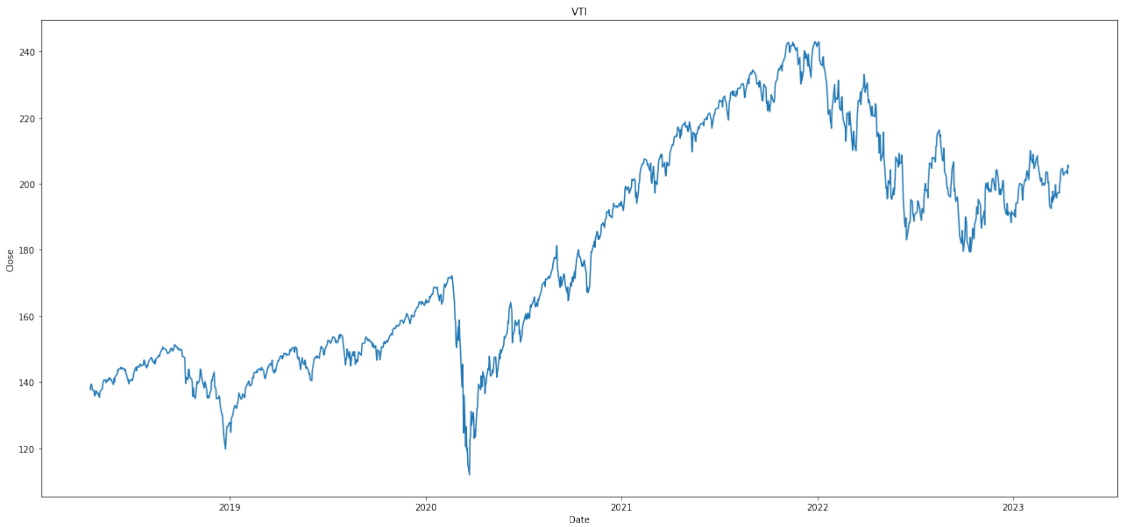


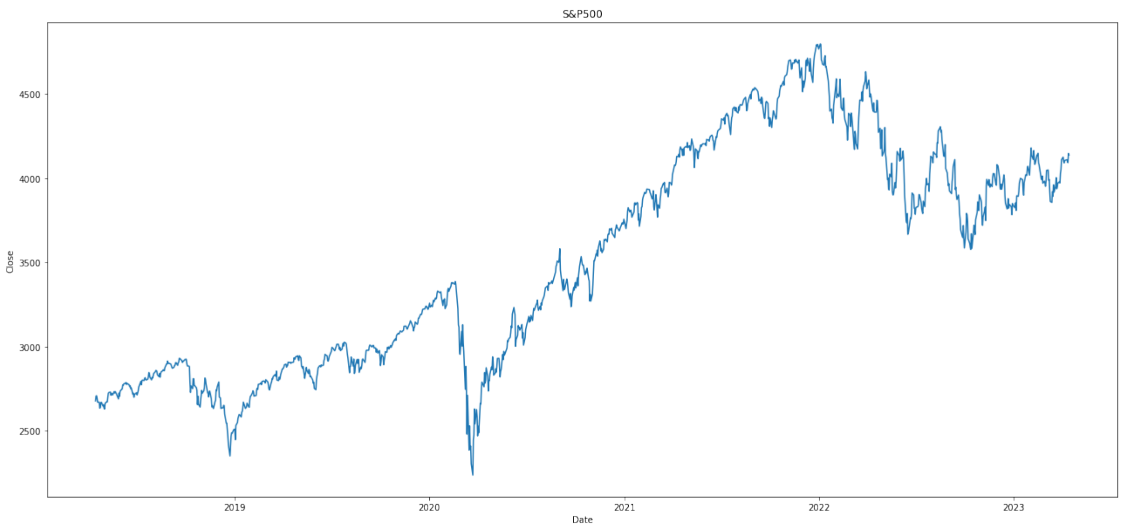

AMZN
180
160
140
120
100
80
Close
2019
2020
2021
2022
2023
Date


BRK-A
550000
500000
450000
400000
350000
300000
250000
Close
2019
2020
2021
2022
2023
Date


GOOG
Close
Date
140
120
100
80
60
2019
2020
2021
2022
2023

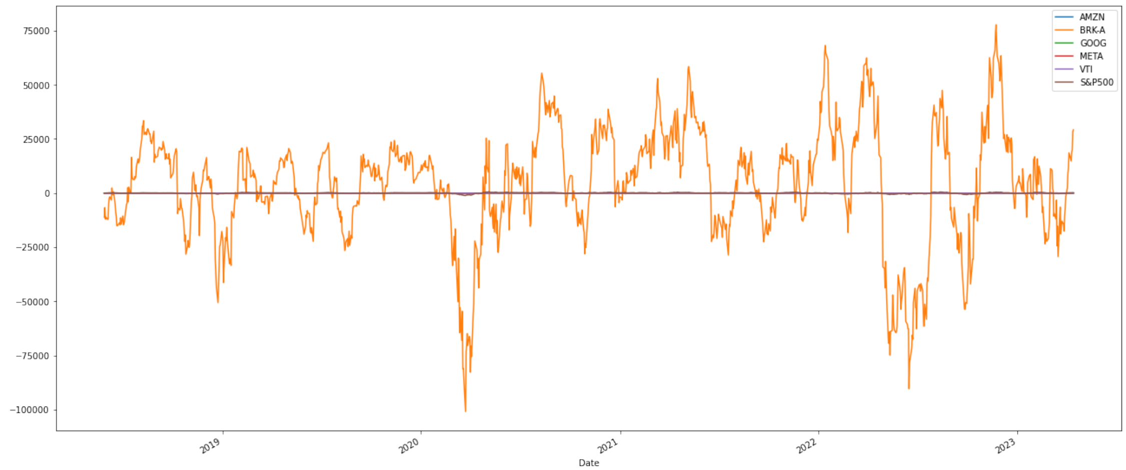

META
Close
Date
350
300
250
200
150
100
2019
2020
2021
2022
2023

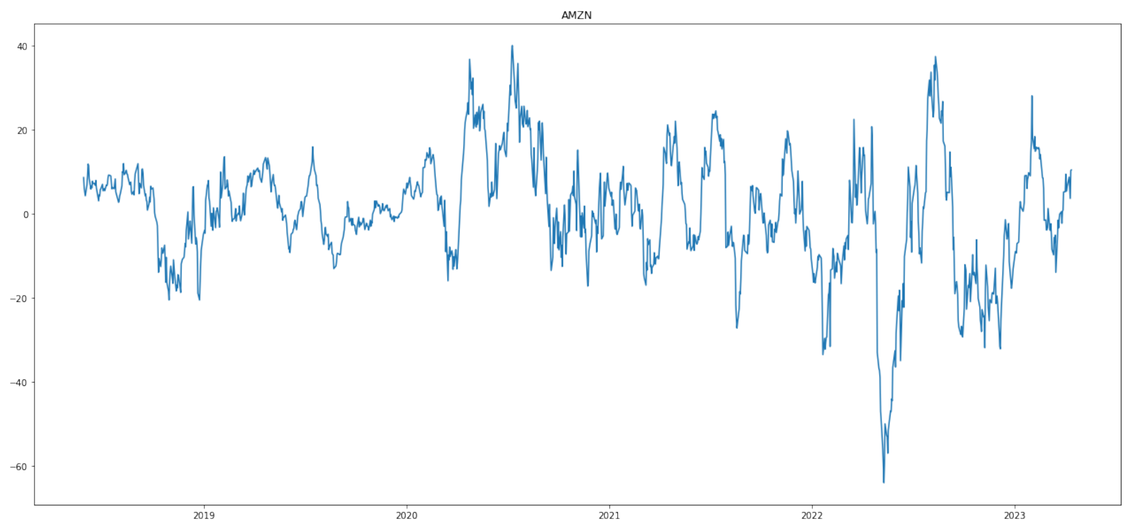


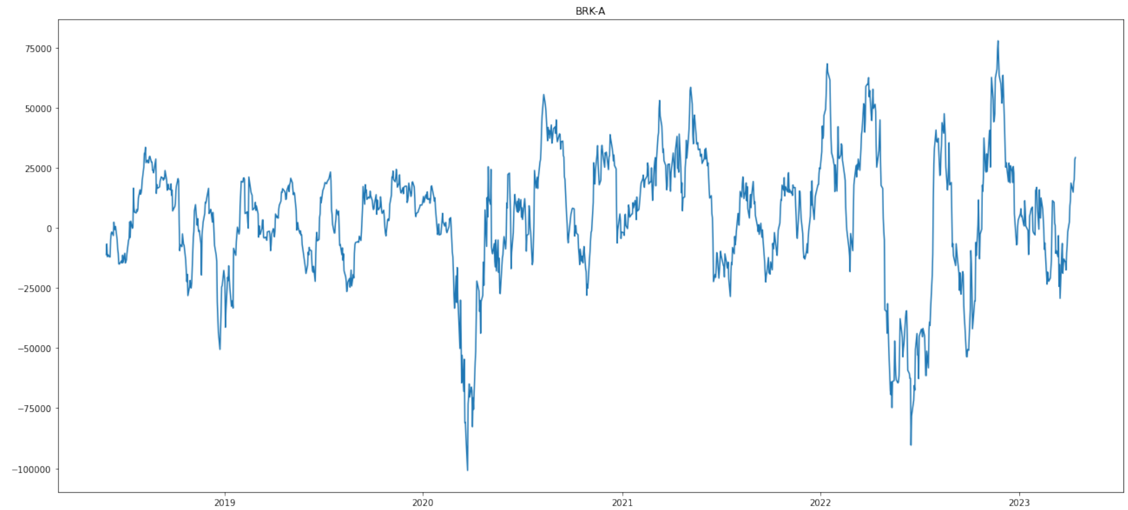


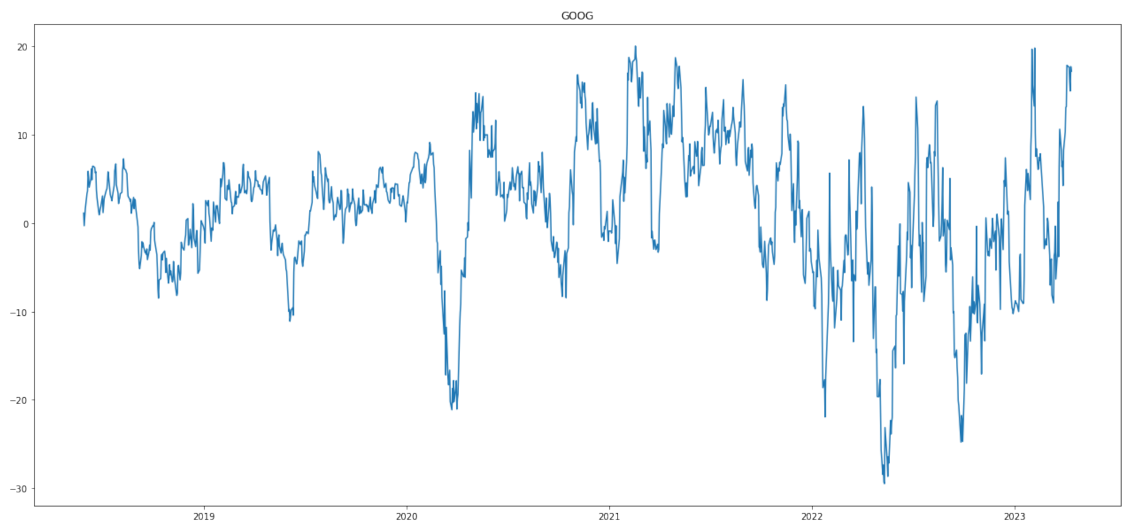
	AMZN	BRK-A	GOOG	META	VTI	S&P500
Date						
2018-04-16	NaN	NaN	NaN	NaN	NaN	NaN
2018-04-17	NaN	NaN	NaN	NaN	NaN	NaN
2018-04-18	NaN	NaN	NaN	NaN	NaN	NaN
2018-04-19	NaN	NaN	NaN	NaN	NaN	NaN
2018-04-20	NaN	NaN	NaN	NaN	NaN	NaN
…	…	…	…	…	…	…
2023-04-10	8.669998	14795.0	17.599999	44.360001	4.180008	139.07
2023-04-11	6.159996	18888.0	16.020005	44.310013	3.679993	126.70
2023-04-12	3.599999	19975.0	14.919998	39.059998	3.489991	121.80
2023-04-13	10.230004	28325.0	17.680000	46.930008	5.649994	194.83

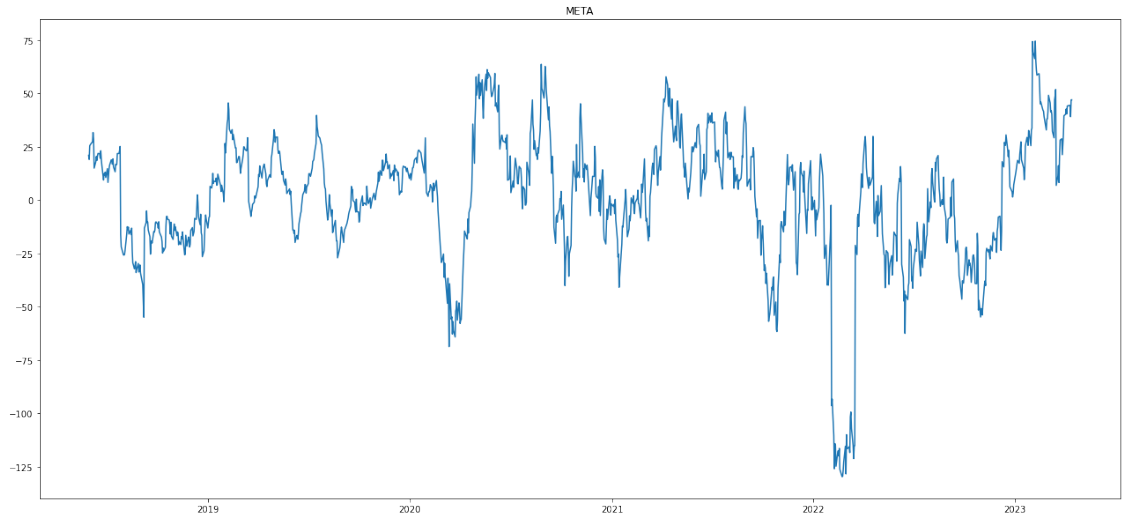
```
2023-04-14  10.380005  29210.0  17.150001  46.960006  4.770004  156.29

[1259 rows x 6 columns]
```

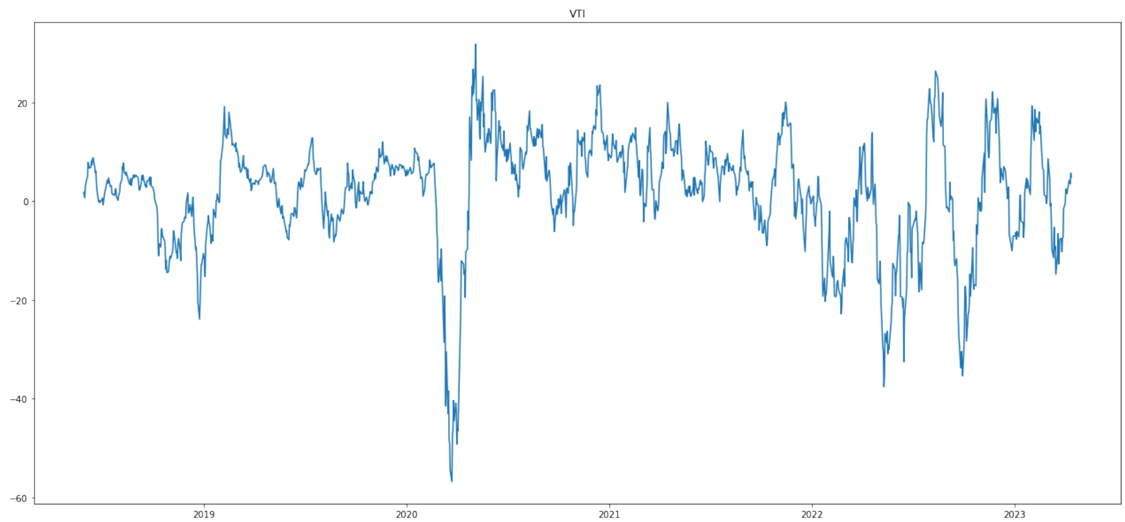


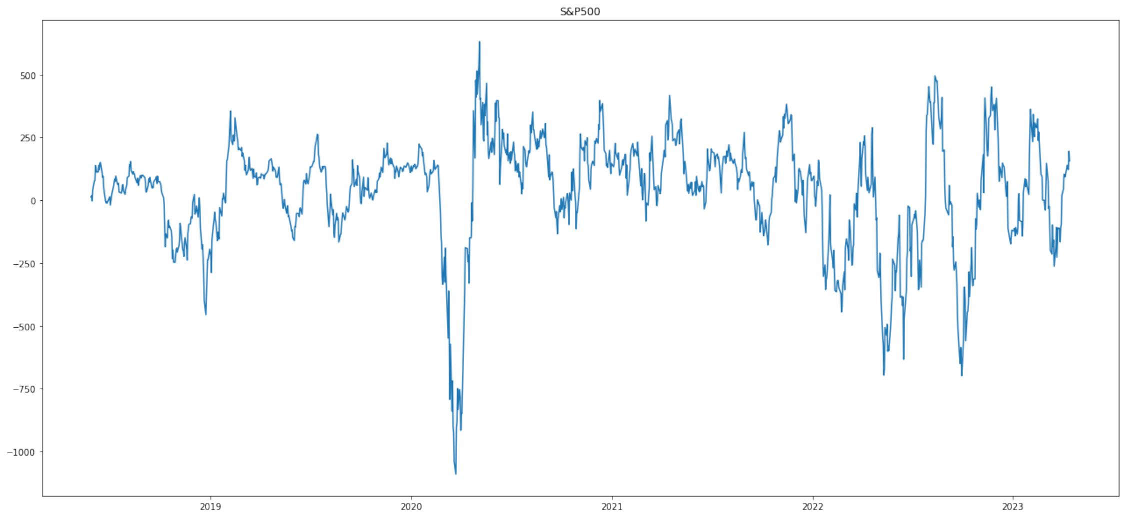



BRK-A
75000
50000
25000
0
−25000
−50000
−75000
−100000
2019
2020
2021
2022
2023


GOOG
20
10
0
−10
−20
−30
2019
2020
2021
2022
2023


META
75
50
25
0
−25
−50
−75
−100
−125
2019
2020
2021
2022
2023

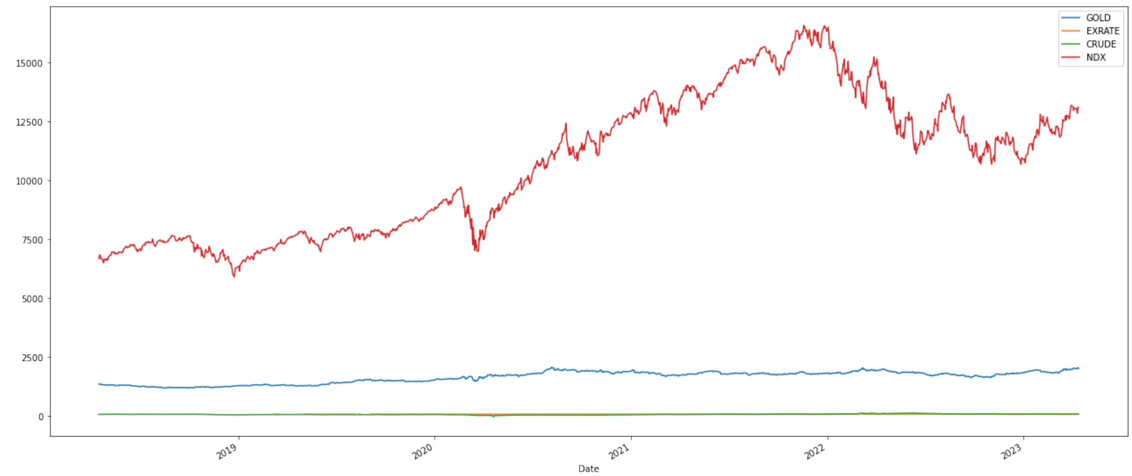

VTI
20
0
−20
−40
−60
2019
2020
2021
2022
2023

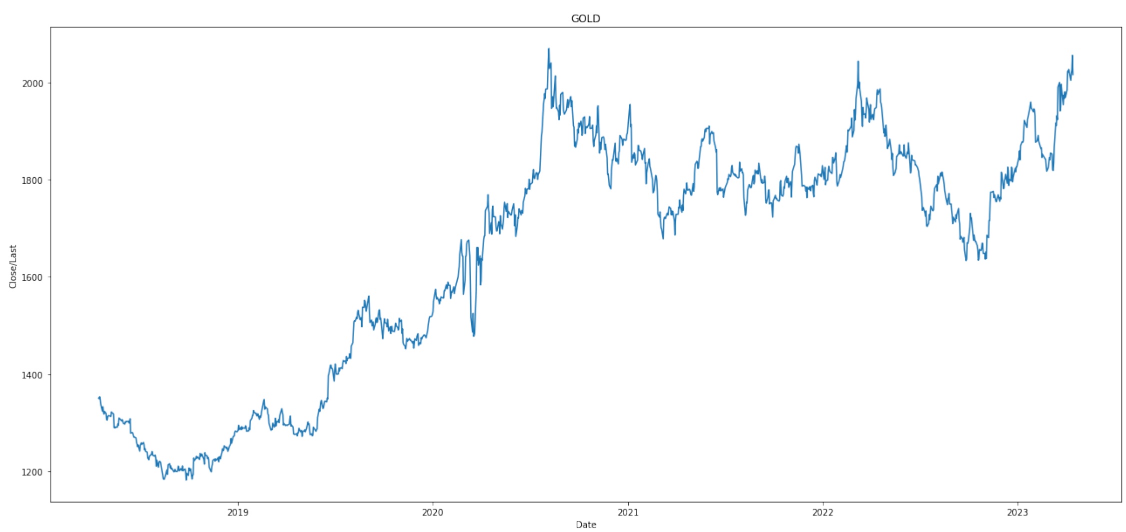


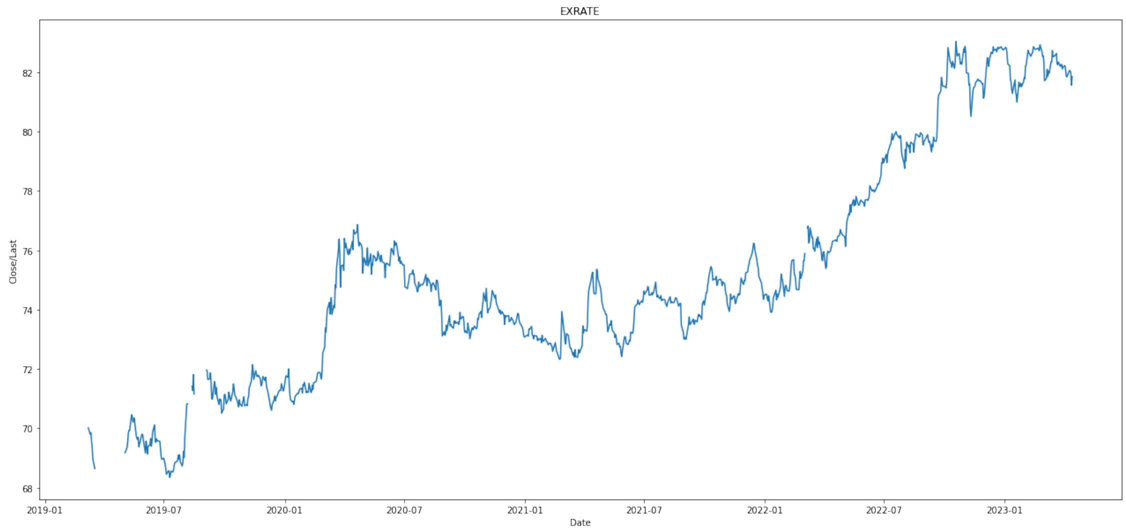

```
[ ]: variables = ['GOLD','EXRATE','CRUDE','NDX']  #variables
     variables_data = pd.DataFrame()

     for ticker in variables:
         variables_data[ticker] = pd.read_csv(f'./variables/{ticker}.csv',␣
      ↪index_col='Date', usecols=['Date', 'Close/Last'],
                                              parse_dates=True)['Close/Last']

     print(variables_data)
     variables_data.plot(figsize=(22,10))


     # Loop through each column and plot a graph
     for col in variables_data.columns:
         plt.figure(figsize=(22,10))
         plt.plot(variables_data[col])
         plt.title(col)
         plt.xlabel('Date')
         plt.ylabel('Close/Last')
         plt.show()
```

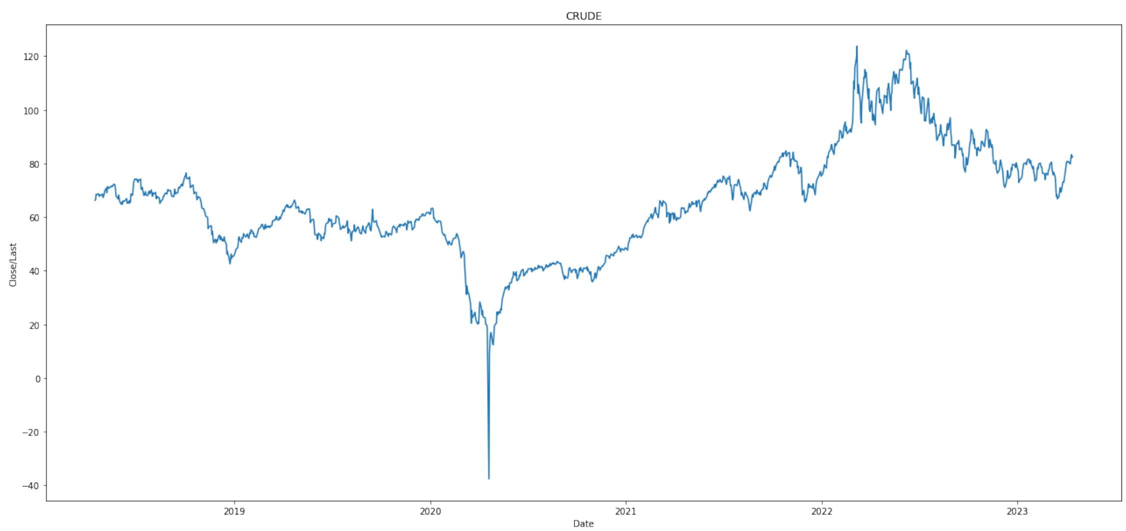
```
              GOLD   EXRATE  CRUDE           NDX
Date
2023-04-14  2015.8  81.8462  82.52  13079.519531
2023-04-13  2055.3  81.5600  82.16  13109.389648
2023-04-12  2024.9  81.9745  83.26  12848.349609
2023-04-11  2019.0  82.0464  81.53  12964.150391
2023-04-10  2003.8  82.0556  79.74  13051.230469
…              …       …      …            …
2018-04-20  1338.3      NaN  68.40   6667.750000
```

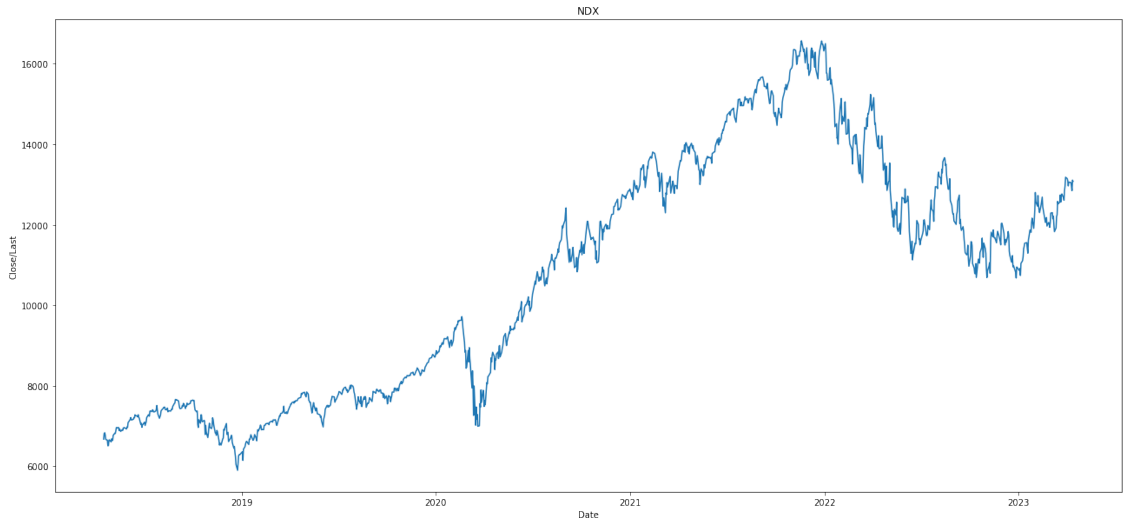
```
2018-04-19  1348.8      NaN  68.29   6774.890137
2018-04-18  1353.5      NaN  68.47   6833.209961
2018-04-17  1349.5      NaN  66.52   6816.370117
2018-04-16  1350.7      NaN  66.22   6675.180176

[1260 rows x 4 columns]
```






EXRATE
Close/Last
Date
82
80
78
76
74
72
70
68
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01


CRUDE
Close/Last
Date
120
100
80
60
40
20
0
−20
−40
2019
2020
2021
2022
2023



[ ]:
```
df=pd.DataFrame()
df=pd.merge(ret_data,variables_data,on='Date',how='inner')

df[['AMZN','BRK-A','GOOG','META','VTI','S&P500']] =␣
 ↪(df[['AMZN','BRK-A','GOOG','META','VTI','S&P500']] /␣
 ↪portfolio_data[['AMZN','BRK-A','GOOG','META','VTI','S&P500']].shift(30)) *␣
 ↪100
df.dropna(inplace=True)
df
```

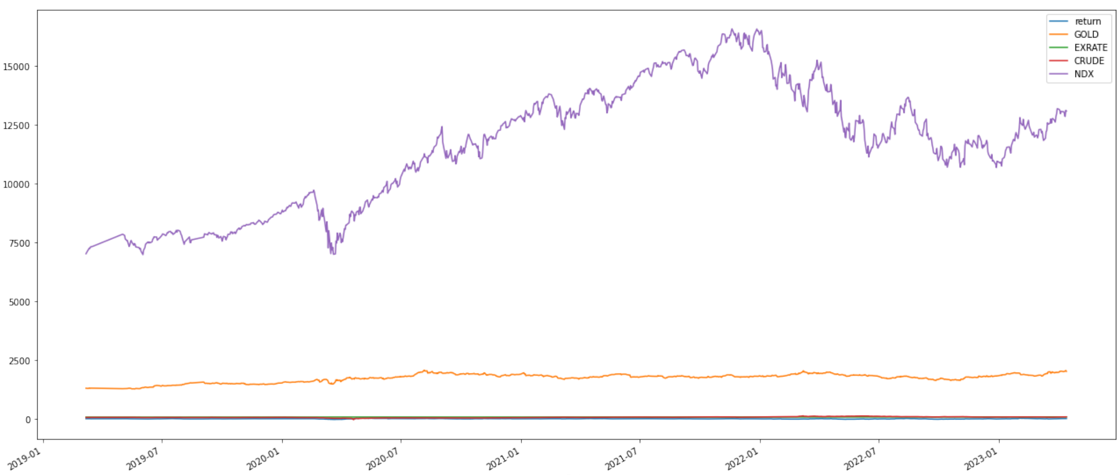
[ ]:
```
                 AMZN     BRK-A       GOOG       META       VTI    S&P500  \
Date
2019-03-08  -2.062319 -0.888359   6.371172  16.299804  4.282550  3.812544
2019-03-11   0.002987 -0.083023   7.770003  15.475480  4.828898  4.448431
2019-03-12   2.149715  1.074751  11.505678  16.579641  5.957033  5.585415
2019-03-13   6.081383  1.089109  12.511554  20.237182  6.785931  6.474242
2019-03-14   0.945262 -1.238761   8.859936  13.129903  5.112139  4.752989
…                …         …         …          …          …         …
2023-04-10   9.272725  3.204427  19.697817  26.034392  2.095452  3.502987
2023-04-11   6.569961  4.089091  17.780250  26.135434  1.838250  3.181626
2023-04-12   3.820438  4.309368  16.522699  22.327654  1.749194  3.067894
2023-04-13  11.099061  6.125185  19.533753  27.061474  2.824997  4.930670
2023-04-14  11.266694  6.257632  18.578704  26.906553  2.381311  3.925553

              GOLD   EXRATE  CRUDE          NDX
Date
2019-03-08  1299.3  70.0190  56.07  7015.689941
2019-03-11  1291.1  69.8003  56.79  7164.020020
2019-03-12  1296.3  69.8550  56.87  7201.279785
```

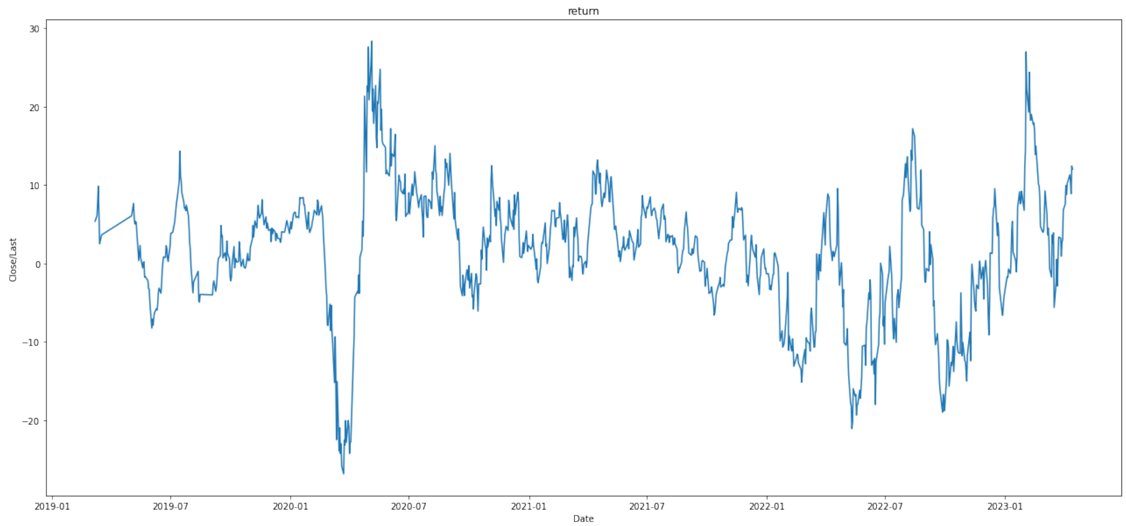
```
2019-03-13  1309.3  69.5672  58.26   7256.979980
2019-03-14  1293.4  69.3360  58.61   7243.009766
…              …       …     …            …
2023-04-10  2003.8  82.0556  79.74  13051.230469
2023-04-11  2019.0  82.0464  81.53  12964.150391
2023-04-12  2024.9  81.9745  83.26  12848.349609
2023-04-13  2055.3  81.5600  82.16  13109.389648
2023-04-14  2015.8  81.8462  82.52  13079.519531

[986 rows x 10 columns]
```

[ ]:
```
data = pd.DataFrame()
data = df.iloc[:, 6:10]

data['return'] = (np.dot(df.iloc[:, :6], weights)) / sum(weights)


data.dropna(inplace=True)

# get the list of column names
cols = list(data.columns)
# move the last column to the first position
cols = cols[-1:] + cols[:-1]
# reindex the dataset with the new order of columns
data = data.reindex(columns=cols)

print(data)
data.plot(figsize=(22,10))


# Loop through each column and plot a graph
for col in data.columns:
    plt.figure(figsize=(22,10))
    plt.plot(data[col])
    plt.title(col)
    plt.xlabel('Date')
    plt.ylabel('Close/Last')
    plt.show()
```

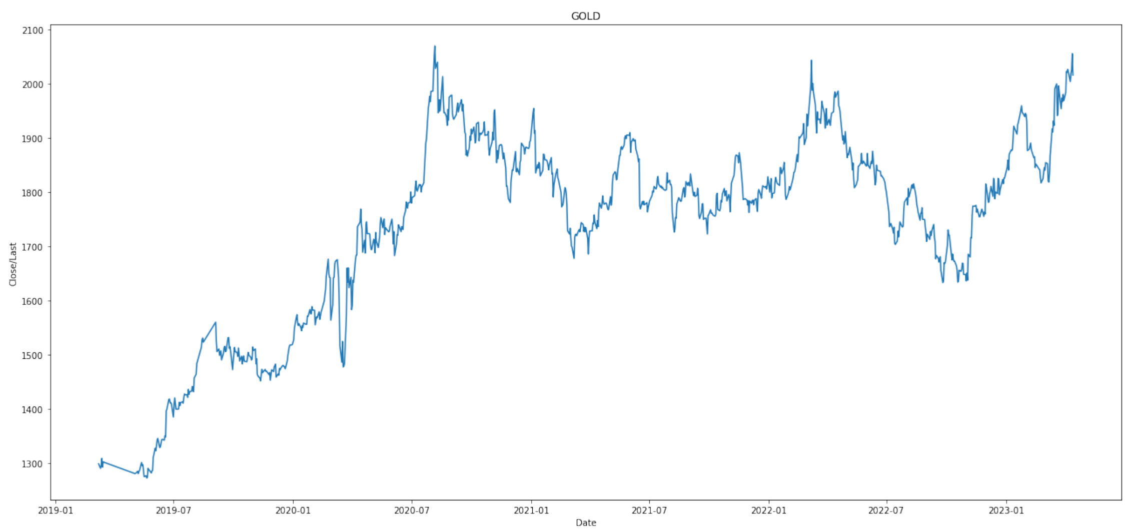
```
/tmp/ipykernel_86300/2139314171.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['return'] = (np.dot(df.iloc[:, :6], weights)) / sum(weights)
/tmp/ipykernel_86300/2139314171.py:7: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data.dropna(inplace=True)

               return    GOLD   EXRATE  CRUDE          NDX
Date
2019-03-08   5.377774  1299.3  70.0190  56.07   7015.689941
2019-03-11   6.112508  1291.1  69.8003  56.79   7164.020020
2019-03-12   7.833950  1296.3  69.8550  56.87   7201.279785
2019-03-13   9.846936  1309.3  69.5672  58.26   7256.979980
2019-03-14   6.055111  1293.4  69.3360  58.61   7243.009766
…                 …       …        …      …            …
2023-04-10  11.296005  2003.8  82.0556  79.74  13051.230469
2023-04-11  10.416546  2019.0  82.0464  81.53  12964.150391
2023-04-12   8.886909  2024.9  81.9745  83.26  12848.349609
2023-04-13  12.419563  2055.3  81.5600  82.16  13109.389648
2023-04-14  11.995709  2015.8  81.8462  82.52  13079.519531

[986 rows x 5 columns]
```

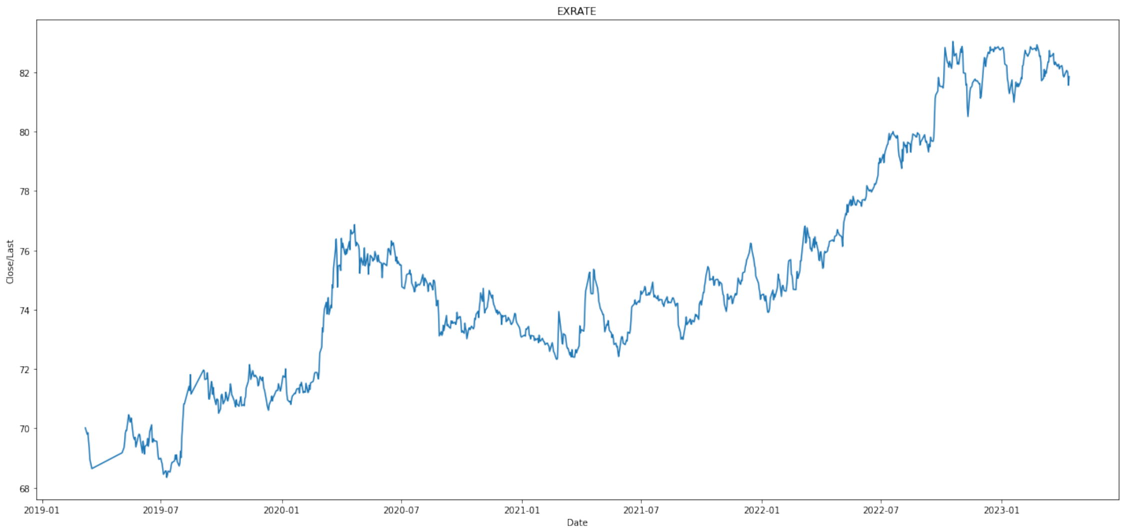


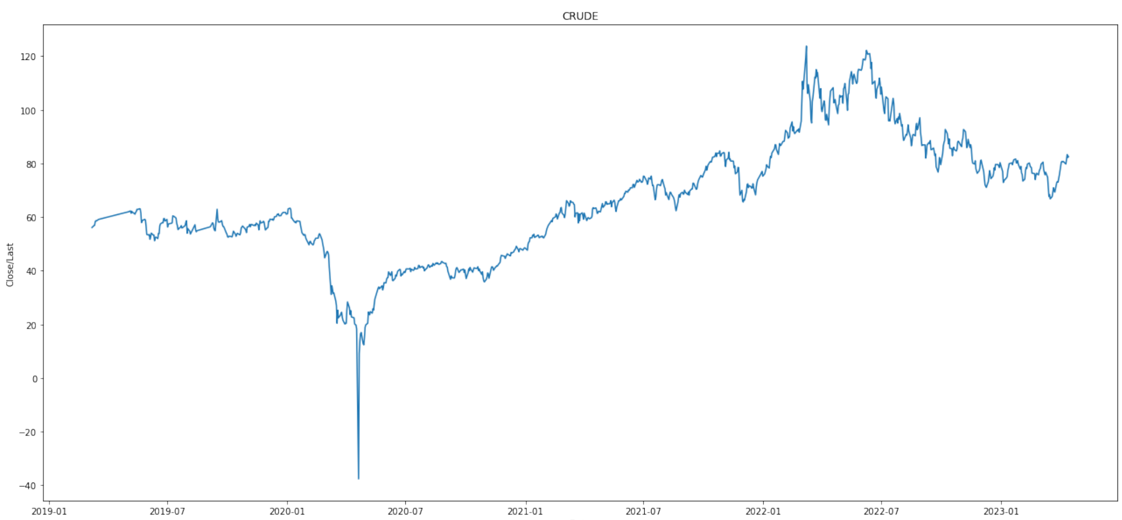


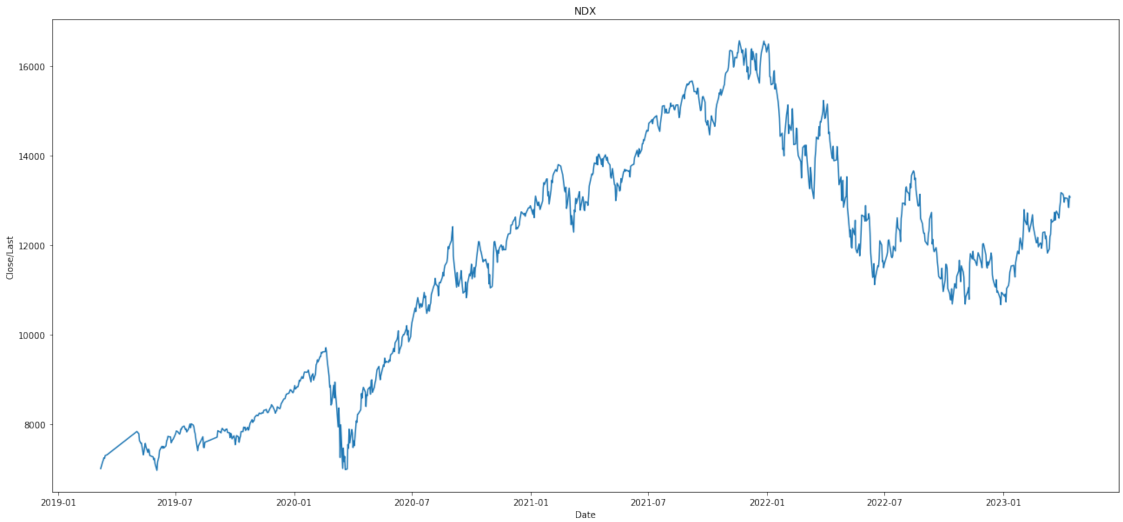
return
Close/Last
Date
30
20
10
0
−10
−20
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01



GOLD
Close/Last
Date
2100
2000
1900
1800
1700
1600
1500
1400
1300
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01


EXRATE
82
80
78
76
74
72
70
68
Close/Last
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
Date


CRUDE
120
100
80
60
40
20
0
−20
−40
Close/Last
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
Date



[ ]:
```
ad_fuller_result_1 = adfuller(data['return'])#.diff()[1:])

print(f'ADF Statistic: {ad_fuller_result_1[0]}')
print(f'p-value: {ad_fuller_result_1[1]}')
print()

ad_fuller_result_2 = adfuller(data['GOLD'].diff()[1:])

print(f'ADF Statistic: {ad_fuller_result_2[0]}')
print(f'p-value: {ad_fuller_result_2[1]}')
print()


ad_fuller_result_3 = adfuller(data['EXRATE'].diff()[1:])

print(f'ADF Statistic: {ad_fuller_result_3[0]}')
print(f'p-value: {ad_fuller_result_3[1]}')
print()

ad_fuller_result_4 = adfuller(data['CRUDE'].diff()[1:])

print(f'ADF Statistic: {ad_fuller_result_4[0]}')
print(f'p-value: {ad_fuller_result_4[1]}')
print()

ad_fuller_result_5 = adfuller(data['NDX'].diff()[1:])

print(f'ADF Statistic: {ad_fuller_result_5[0]}')
print(f'p-value: {ad_fuller_result_5[1]}')
```

```
ADF Statistic: -5.2795139309078705
p-value: 6.0368319685366406e-06

ADF Statistic: -14.790021111390425
p-value: 2.178466340554877e-27

ADF Statistic: -11.819283104270639
p-value: 8.490162006715022e-22

ADF Statistic: -21.951735164313426
p-value: 0.0

ADF Statistic: -8.901975223463445
p-value: 1.1647566456618446e-14
```

[ ]:
```python
print('GOLD causes return?\n')
granger_1 = grangercausalitytests(data[['return', 'GOLD']], 5)

print('\n\nEXRATE causes return?\n')
granger_2 = grangercausalitytests(data[['return', 'EXRATE']], 5)

print('\n\nCRUDE causes return?\n')
granger_3 = grangercausalitytests(data[['return', 'CRUDE']], 5)

print('\n\nNDX causes return?\n')
granger_4 = grangercausalitytests(data[['return', 'NDX']], 5)
```

```
GOLD causes return?


Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.9301  , p=0.3351  , df_denom=982, df_num=1
ssr based chi2 test:   chi2=0.9330  , p=0.3341  , df=1
likelihood ratio test: chi2=0.9325  , p=0.3342  , df=1
parameter F test:         F=0.9301  , p=0.3351  , df_denom=982, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=2.7337  , p=0.0655  , df_denom=979, df_num=2
ssr based chi2 test:   chi2=5.4953  , p=0.0641  , df=2
likelihood ratio test: chi2=5.4800  , p=0.0646  , df=2
parameter F test:         F=2.7337  , p=0.0655  , df_denom=979, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=4.1160  , p=0.0065  , df_denom=976, df_num=3
ssr based chi2 test:   chi2=12.4364 , p=0.0060  , df=3
```

```
likelihood ratio test: chi2=12.3584 , p=0.0063  , df=3
parameter F test:         F=4.1160  , p=0.0065  , df_denom=976, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=3.3735  , p=0.0094  , df_denom=973, df_num=4
ssr based chi2 test:   chi2=13.6188 , p=0.0086  , df=4
likelihood ratio test: chi2=13.5252 , p=0.0090  , df=4
parameter F test:         F=3.3735  , p=0.0094  , df_denom=973, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=2.7612  , p=0.0174  , df_denom=970, df_num=5
ssr based chi2 test:   chi2=13.9627 , p=0.0158  , df=5
likelihood ratio test: chi2=13.8642 , p=0.0165  , df=5
parameter F test:         F=2.7612  , p=0.0174  , df_denom=970, df_num=5


EXRATE causes return?


Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.1847  , p=0.6674  , df_denom=982, df_num=1
ssr based chi2 test:   chi2=0.1853  , p=0.6669  , df=1
likelihood ratio test: chi2=0.1853  , p=0.6669  , df=1
parameter F test:         F=0.1847  , p=0.6674  , df_denom=982, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.5189  , p=0.5954  , df_denom=979, df_num=2
ssr based chi2 test:   chi2=1.0430  , p=0.5936  , df=2
likelihood ratio test: chi2=1.0425  , p=0.5938  , df=2
parameter F test:         F=0.5189  , p=0.5954  , df_denom=979, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=2.3770  , p=0.0685  , df_denom=976, df_num=3
ssr based chi2 test:   chi2=7.1821  , p=0.0663  , df=3
likelihood ratio test: chi2=7.1560  , p=0.0671  , df=3
parameter F test:         F=2.3770  , p=0.0685  , df_denom=976, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=1.9842  , p=0.0948  , df_denom=973, df_num=4
ssr based chi2 test:   chi2=8.0101  , p=0.0912  , df=4
likelihood ratio test: chi2=7.9776  , p=0.0924  , df=4
parameter F test:         F=1.9842  , p=0.0948  , df_denom=973, df_num=4
```

```
Granger Causality
number of lags (no zero) 5
ssr based F test:         F=1.6429  , p=0.1459  , df_denom=970, df_num=5
ssr based chi2 test:   chi2=8.3075  , p=0.1401  , df=5
likelihood ratio test: chi2=8.2725  , p=0.1418  , df=5
parameter F test:         F=1.6429  , p=0.1459  , df_denom=970, df_num=5


CRUDE causes return?


Granger Causality
number of lags (no zero) 1
ssr based F test:         F=4.5415  , p=0.0333  , df_denom=982, df_num=1
ssr based chi2 test:   chi2=4.5553  , p=0.0328  , df=1
likelihood ratio test: chi2=4.5448  , p=0.0330  , df=1
parameter F test:         F=4.5415  , p=0.0333  , df_denom=982, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=1.9920  , p=0.1370  , df_denom=979, df_num=2
ssr based chi2 test:   chi2=4.0044  , p=0.1350  , df=2
likelihood ratio test: chi2=3.9963  , p=0.1356  , df=2
parameter F test:         F=1.9920  , p=0.1370  , df_denom=979, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=3.4550  , p=0.0161  , df_denom=976, df_num=3
ssr based chi2 test:   chi2=10.4393 , p=0.0152  , df=3
likelihood ratio test: chi2=10.3842 , p=0.0156  , df=3
parameter F test:         F=3.4550  , p=0.0161  , df_denom=976, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=4.0185  , p=0.0031  , df_denom=973, df_num=4
ssr based chi2 test:   chi2=16.2228 , p=0.0027  , df=4
likelihood ratio test: chi2=16.0903 , p=0.0029  , df=4
parameter F test:         F=4.0185  , p=0.0031  , df_denom=973, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=7.8849  , p=0.0000  , df_denom=970, df_num=5
ssr based chi2 test:   chi2=39.8715 , p=0.0000  , df=5
likelihood ratio test: chi2=39.0825 , p=0.0000  , df=5
parameter F test:         F=7.8849  , p=0.0000  , df_denom=970, df_num=5
```

```
NDX causes return?


Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.7624  , p=0.3828  , df_denom=982, df_num=1
ssr based chi2 test:   chi2=0.7647  , p=0.3818  , df=1
likelihood ratio test: chi2=0.7644  , p=0.3819  , df=1
parameter F test:         F=0.7624  , p=0.3828  , df_denom=982, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=1.1005  , p=0.3331  , df_denom=979, df_num=2
ssr based chi2 test:   chi2=2.2122  , p=0.3309  , df=2
likelihood ratio test: chi2=2.2097  , p=0.3313  , df=2
parameter F test:         F=1.1005  , p=0.3331  , df_denom=979, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.7139  , p=0.5438  , df_denom=976, df_num=3
ssr based chi2 test:   chi2=2.1571  , p=0.5405  , df=3
likelihood ratio test: chi2=2.1547  , p=0.5409  , df=3
parameter F test:         F=0.7139  , p=0.5438  , df_denom=976, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.6795  , p=0.6063  , df_denom=973, df_num=4
ssr based chi2 test:   chi2=2.7431  , p=0.6017  , df=4
likelihood ratio test: chi2=2.7392  , p=0.6024  , df=4
parameter F test:         F=0.6795  , p=0.6063  , df_denom=973, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=0.5952  , p=0.7037  , df_denom=970, df_num=5
ssr based chi2 test:   chi2=3.0097  , p=0.6985  , df=5
likelihood ratio test: chi2=3.0051  , p=0.6992  , df=5
parameter F test:         F=0.5952  , p=0.7037  , df_denom=970, df_num=5
```

[ ]:
```
# Remove column name 'CRUDE'
data.drop('NDX', axis=1,inplace=True)
data.drop('EXRATE', axis=1,inplace=True)

data
```

[ ]:

	return	GOLD	CRUDE
Date			
2019-03-08	5.377774	1299.3	56.07

```
2019-03-11   6.112508  1291.1  56.79
2019-03-12   7.833950  1296.3  56.87
2019-03-13   9.846936  1309.3  58.26
2019-03-14   6.055111  1293.4  58.61
…                 …       …     …
2023-04-10  11.296005  2003.8  79.74
2023-04-11  10.416546  2019.0  81.53
2023-04-12   8.886909  2024.9  83.26
2023-04-13  12.419563  2055.3  82.16
2023-04-14  11.995709  2015.8  82.52

[986 rows x 3 columns]
```

```
[ ]: train_df=data[:-30]
     test_df=data[-30:]
     print(train_df)
     print(test_df)
```

```
               return    GOLD  CRUDE
Date
2019-03-08   5.377774  1299.3  56.07
2019-03-11   6.112508  1291.1  56.79
2019-03-12   7.833950  1296.3  56.87
2019-03-13   9.846936  1309.3  58.26
2019-03-14   6.055111  1293.4  58.61
…                 …       …     …
2023-02-24   4.734287  1817.1  76.32
2023-02-27   4.069295  1824.9  75.68
2023-02-28   3.966830  1836.7  77.05
2023-03-01   4.229242  1845.4  77.69
2023-03-02   6.278340  1840.5  78.16

[956 rows x 3 columns]
                return    GOLD  CRUDE
Date
2023-03-03   9.242862  1854.6  79.68
2023-03-06   6.205897  1852.4  80.46
2023-03-07   3.652892  1820.0  77.58
2023-03-08   4.501504  1818.6  76.66
2023-03-09   2.988977  1834.6  75.72
2023-03-10  -0.622127  1867.2  76.68
2023-03-13  -1.731859  1916.5  74.80
2023-03-14   3.638291  1910.9  71.33
2023-03-15   2.559988  1931.3  67.61
2023-03-16   3.921017  1923.0  68.35
2023-03-17  -5.603373  1990.2  66.74
2023-03-20  -2.422152  1999.7  67.64
```

```
2023-03-21   0.533070  1941.1  69.33
2023-03-22  -2.853770  1949.6  70.90
2023-03-23   0.900307  1995.9  69.96
2023-03-24   3.366802  1983.8  69.26
2023-03-27   3.235196  1953.8  72.81
2023-03-28   0.931500  1973.5  73.20
2023-03-29   2.730583  1966.9  72.97
2023-03-30   3.028428  1980.3  74.37
2023-03-31   6.836645  1969.0  75.67
2023-04-03   7.563347  1983.9  80.42
2023-04-04   9.926627  2022.2  80.71
2023-04-05   8.783539  2020.9  80.61
2023-04-06  10.171852  2026.4  80.70
2023-04-10  11.296005  2003.8  79.74
2023-04-11  10.416546  2019.0  81.53
2023-04-12   8.886909  2024.9  83.26
2023-04-13  12.419563  2055.3  82.16
2023-04-14  11.995709  2015.8  82.52
```

[ ]:
```python
model = VAR(train_df.diff()[1:])
sorted_order=model.select_order(maxlags=10)
print(sorted_order.summary())
```

```
 VAR Order Selection (* highlights the minimums)
==================================================
       AIC         BIC         FPE         HQIC
--------------------------------------------------
0        9.965       9.981   2.127e+04       9.971
1        9.874      9.935*   1.941e+04       9.897
2        9.847       9.954   1.889e+04      9.888*
3        9.856       10.01   1.908e+04       9.915
4        9.834       10.03   1.867e+04       9.911
5        9.842       10.09   1.882e+04       9.936
6        9.827       10.12   1.853e+04       9.939
7        9.830       10.17   1.859e+04       9.959
8        9.834       10.22   1.865e+04       9.980
9       9.826*       10.26  1.851e+04*       9.990
10       9.837       10.31   1.871e+04       10.02
--------------------------------------------------

/home/kartik/.local/lib/python3.10/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  self._init_dates(dates, freq)
```

[ ]:
```python
var_model = VARMAX(train_df, order=(9,0),enforce_stationarity= True)
fitted_model = var_model.fit(disp=False)
```

```python
print(fitted_model.summary())
```

```
/home/kartik/.local/lib/python3.10/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/home/kartik/.local/lib/python3.10/site-packages/statsmodels/base/model.py:604:
ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check
mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "

                                Statespace Model Results
================================================================================
=======
Dep. Variable:     ['return', 'GOLD', 'CRUDE']   No. Observations:
956
Model:                                  VAR(9)   Log Likelihood
-8680.964
                                   + intercept   AIC
17541.928
Date:                         Thu, 20 Apr 2023   BIC
17979.576
Time:                                 08:21:26   HQIC
17708.628
Sample:                                      0
                                         - 956
Covariance Type:                           opg
================================================================================
==========
Ljung-Box (L1) (Q):       0.01, 0.01, 0.01   Jarque-Bera (JB):   236.08, 479.92,
885770.42
Prob(Q):                  0.92, 0.92, 0.91   Prob(JB):                    0.00,
0.00, 0.00
Heteroskedasticity (H):   1.79, 0.80, 0.55   Skew:                       0.12,
-0.48, -6.64
Prob(H) (two-sided):      0.00, 0.05, 0.00   Kurtosis:                   5.42,
6.33, 151.53
                         Results for equation return
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.6599      1.068     -0.618      0.536      -2.752       1.432
L1.return      0.8647      0.033     26.056      0.000       0.800       0.930
L1.GOLD        0.0067      0.005      1.367      0.172      -0.003       0.016
L1.CRUDE      -0.0174      0.042     -0.418      0.676      -0.099       0.064
L2.return      0.1535      0.039      3.926      0.000       0.077       0.230
L2.GOLD        0.0037      0.007      0.526      0.599      -0.010       0.017
```

L2.CRUDE	0.0540	0.053	1.029	0.304	-0.049	0.157
L3.return	-0.1096	0.038	-2.864	0.004	-0.185	-0.035
L3.GOLD	-0.0126	0.006	-2.009	0.045	-0.025	-0.000
L3.CRUDE	-0.0327	0.044	-0.746	0.456	-0.119	0.053
L4.return	0.0644	0.041	1.565	0.118	-0.016	0.145
L4.GOLD	0.0024	0.006	0.409	0.682	-0.009	0.014
L4.CRUDE	-0.1315	0.037	-3.584	0.000	-0.203	-0.060
L5.return	0.0230	0.040	0.572	0.567	-0.056	0.102
L5.GOLD	0.0028	0.006	0.459	0.646	-0.009	0.015
L5.CRUDE	0.1442	0.043	3.363	0.001	0.060	0.228
L6.return	-0.1010	0.039	-2.572	0.010	-0.178	-0.024
L6.GOLD	0.0029	0.006	0.506	0.613	-0.008	0.014
L6.CRUDE	-0.0792	0.034	-2.322	0.020	-0.146	-0.012
L7.return	0.1505	0.038	3.998	0.000	0.077	0.224
L7.GOLD	-0.0095	0.006	-1.466	0.143	-0.022	0.003
L7.CRUDE	-0.0034	0.041	-0.084	0.933	-0.084	0.077
L8.return	-0.0958	0.039	-2.472	0.013	-0.172	-0.020
L8.GOLD	0.0123	0.006	2.010	0.044	0.000	0.024
L8.CRUDE	0.0159	0.038	0.421	0.673	-0.058	0.090
L9.return	0.0006	0.031	0.020	0.984	-0.060	0.062
L9.GOLD	-0.0079	0.005	-1.688	0.091	-0.017	0.001
L9.CRUDE	0.0422	0.029	1.444	0.149	-0.015	0.100

Results for equation GOLD

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	17.5899	6.383	2.756	0.006	5.080	30.100
L1.return	0.3059	0.236	1.298	0.194	-0.156	0.768
L1.GOLD	0.9446	0.035	27.274	0.000	0.877	1.013
L1.CRUDE	0.4800	0.348	1.379	0.168	-0.202	1.162
L2.return	-0.1257	0.371	-0.339	0.735	-0.853	0.602
L2.GOLD	0.0367	0.045	0.825	0.409	-0.051	0.124
L2.CRUDE	-0.6472	0.496	-1.305	0.192	-1.619	0.325
L3.return	-0.2577	0.394	-0.654	0.513	-1.030	0.515
L3.GOLD	0.0270	0.045	0.600	0.549	-0.061	0.115
L3.CRUDE	0.2868	0.456	0.628	0.530	-0.608	1.181
L4.return	0.2212	0.375	0.590	0.555	-0.514	0.956
L4.GOLD	-0.0827	0.050	-1.652	0.099	-0.181	0.015
L4.CRUDE	0.2363	0.373	0.634	0.526	-0.494	0.966
L5.return	-0.1740	0.403	-0.432	0.666	-0.964	0.616
L5.GOLD	0.0244	0.048	0.508	0.612	-0.070	0.119
L5.CRUDE	-0.2501	0.391	-0.640	0.522	-1.016	0.516
L6.return	-0.3067	0.378	-0.811	0.417	-1.048	0.434
L6.GOLD	-0.0462	0.051	-0.906	0.365	-0.146	0.054
L6.CRUDE	-0.3848	0.442	-0.870	0.384	-1.251	0.482
L7.return	0.3490	0.362	0.965	0.334	-0.360	1.058
L7.GOLD	0.0648	0.046	1.396	0.163	-0.026	0.156
L7.CRUDE	0.2611	0.479	0.545	0.586	-0.678	1.201

L8.return	-0.6286	0.328	-1.918	0.055	-1.271	0.014
L8.GOLD	0.0618	0.054	1.151	0.250	-0.043	0.167
L8.CRUDE	0.0296	0.499	0.059	0.953	-0.948	1.007
L9.return	0.4773	0.270	1.769	0.077	-0.052	1.006
L9.GOLD	-0.0389	0.037	-1.053	0.292	-0.111	0.033
L9.CRUDE	-0.0559	0.343	-0.163	0.870	-0.728	0.616

Results for equation CRUDE

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-1.2236	1.848	-0.662	0.508	-4.845	2.397
L1.return	0.0207	0.055	0.375	0.707	-0.087	0.129
L1.GOLD	0.0168	0.007	2.314	0.021	0.003	0.031
L1.CRUDE	0.6965	0.017	41.509	0.000	0.664	0.729
L2.return	0.0887	0.075	1.180	0.238	-0.059	0.236
L2.GOLD	-0.0234	0.011	-2.207	0.027	-0.044	-0.003
L2.CRUDE	0.1324	0.024	5.531	0.000	0.085	0.179
L3.return	-0.1120	0.073	-1.536	0.125	-0.255	0.031
L3.GOLD	0.0058	0.011	0.528	0.598	-0.016	0.027
L3.CRUDE	0.0975	0.049	1.991	0.047	0.002	0.193
L4.return	0.0021	0.077	0.028	0.978	-0.148	0.153
L4.GOLD	-0.0057	0.010	-0.559	0.576	-0.026	0.014
L4.CRUDE	0.0117	0.072	0.163	0.870	-0.129	0.152
L5.return	-0.0158	0.077	-0.206	0.837	-0.166	0.135
L5.GOLD	0.0016	0.011	0.145	0.885	-0.020	0.023
L5.CRUDE	0.0124	0.056	0.223	0.824	-0.097	0.121
L6.return	-0.0461	0.071	-0.647	0.518	-0.186	0.094
L6.GOLD	-0.0056	0.010	-0.579	0.562	-0.025	0.013
L6.CRUDE	0.0330	0.069	0.479	0.632	-0.102	0.168
L7.return	0.1015	0.069	1.466	0.143	-0.034	0.237
L7.GOLD	0.0199	0.010	1.897	0.058	-0.001	0.040
L7.CRUDE	-0.0104	0.065	-0.160	0.873	-0.138	0.117
L8.return	-0.0022	0.078	-0.028	0.978	-0.155	0.151
L8.GOLD	-0.0047	0.011	-0.430	0.667	-0.026	0.017
L8.CRUDE	0.0124	0.060	0.207	0.836	-0.105	0.130
L9.return	-0.0312	0.062	-0.505	0.614	-0.152	0.090
L9.GOLD	-0.0036	0.009	-0.408	0.683	-0.021	0.014
L9.CRUDE	0.0087	0.038	0.229	0.819	-0.066	0.083

Error covariance matrix

	coef	std err	z	P>|z|	[0.025	0.975]
sqrt.var.return	2.3504	0.041	57.368	0.000	2.270	2.431
sqrt.cov.return.GOLD	0.7684	0.639	1.203	0.229	-0.483	

```
2.020
sqrt.var.GOLD           18.3857      0.361     50.932      0.000      17.678
19.093
sqrt.cov.return.CRUDE    0.2377      0.131      1.820      0.069      -0.018
0.494
sqrt.cov.GOLD.CRUDE      0.2521      0.139      1.818      0.069      -0.020
0.524
sqrt.var.CRUDE           2.8895      0.037     77.982      0.000       2.817
2.962
================================================================================
=========

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

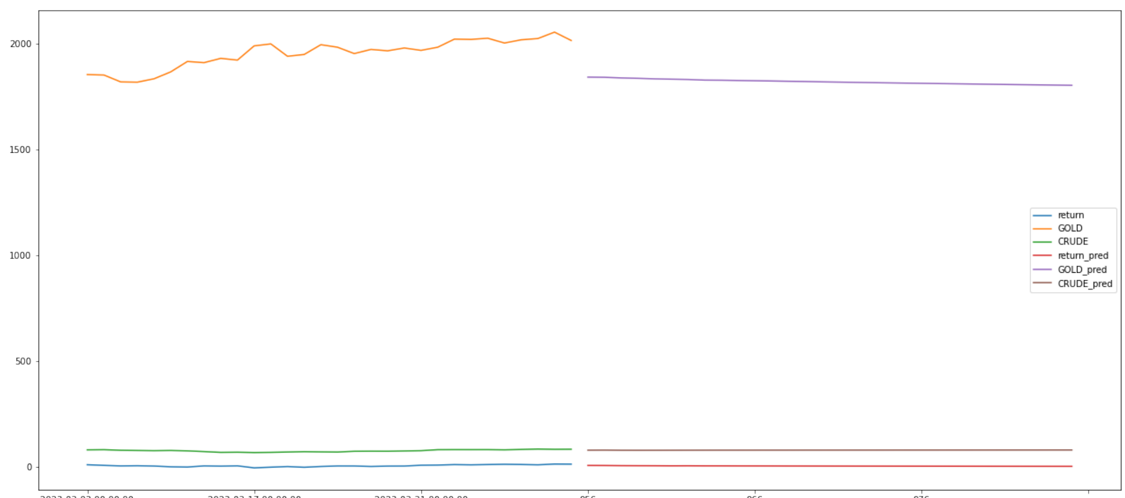
[ ]:
```
n_forecast = 30
predict = fitted_model.get_prediction(start=len(train_df),end=len(train_df) +
    n_forecast-1)

predictions=predict.predicted_mean
predictions.columns=['return_pred','GOLD_pred','CRUDE_pred']
print(predictions)
```

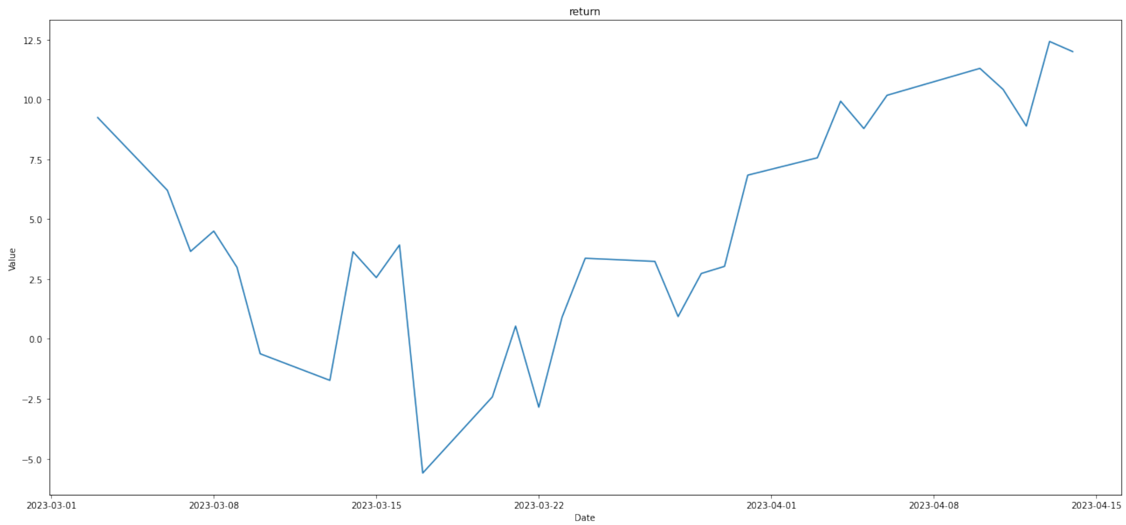
```
     return_pred    GOLD_pred  CRUDE_pred
956     5.837459  1842.468308   78.024502
957     5.596317  1841.660566   78.312804
958     4.888626  1838.644482   77.648642
959     4.638550  1836.759894   77.513984
960     4.458142  1834.302309   77.589998
961     4.073245  1832.651477   77.705151
962     4.289238  1831.198798   77.868730
963     3.952927  1828.283374   78.013783
964     3.888021  1827.697623   78.124545
965     3.667947  1826.432423   78.147612
966     3.547857  1825.383920   78.194681
967     3.401620  1824.266153   78.247081
968     3.186942  1822.829051   78.265323
969     3.085817  1821.759239   78.343331
970     2.913709  1820.202213   78.361434
971     2.791352  1818.959319   78.410543
972     2.662618  1817.671306   78.458737
973     2.547612  1816.491210   78.506012
974     2.448469  1815.350957   78.561213
975     2.333972  1814.189769   78.598614
976     2.239375  1813.140906   78.648374
977     2.140718  1812.037326   78.686925
978     2.050102  1810.969412   78.725963
```

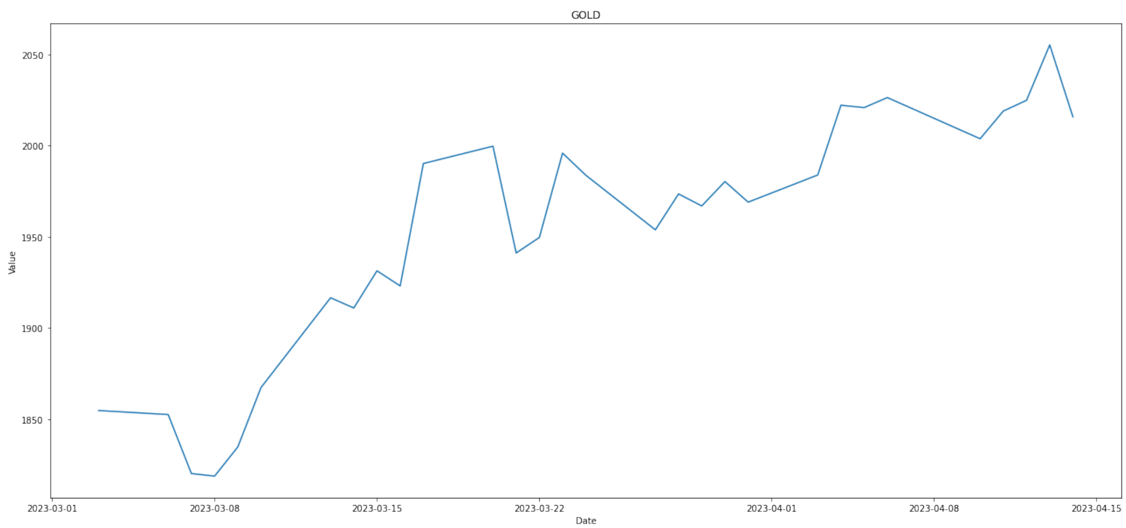
```
979     1.963525  1809.898694   78.764422
980     1.878085  1808.840052   78.801675
981     1.801794  1807.808041   78.840574
982     1.725504  1806.773418   78.876163
983     1.655274  1805.772472   78.913176
984     1.587447  1804.782190   78.948674
985     1.523070  1803.808769   78.983354
```

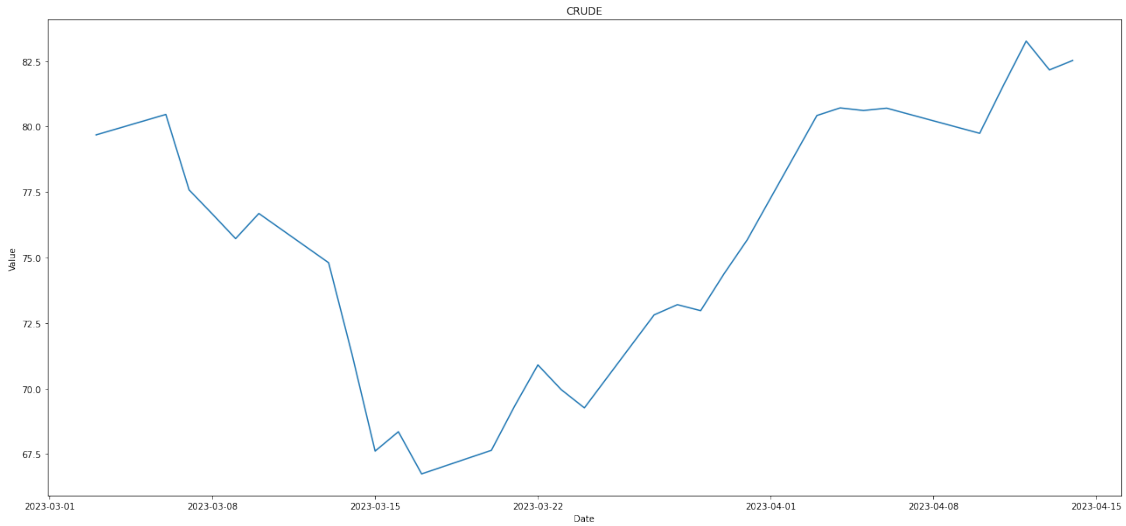
```
/home/kartik/.local/lib/python3.10/site-
packages/statsmodels/tsa/base/tsa_model.py:834: ValueWarning: No supported index
is available. Prediction results will be given with an integer index beginning
at `start`.
  return get_prediction_index(
```

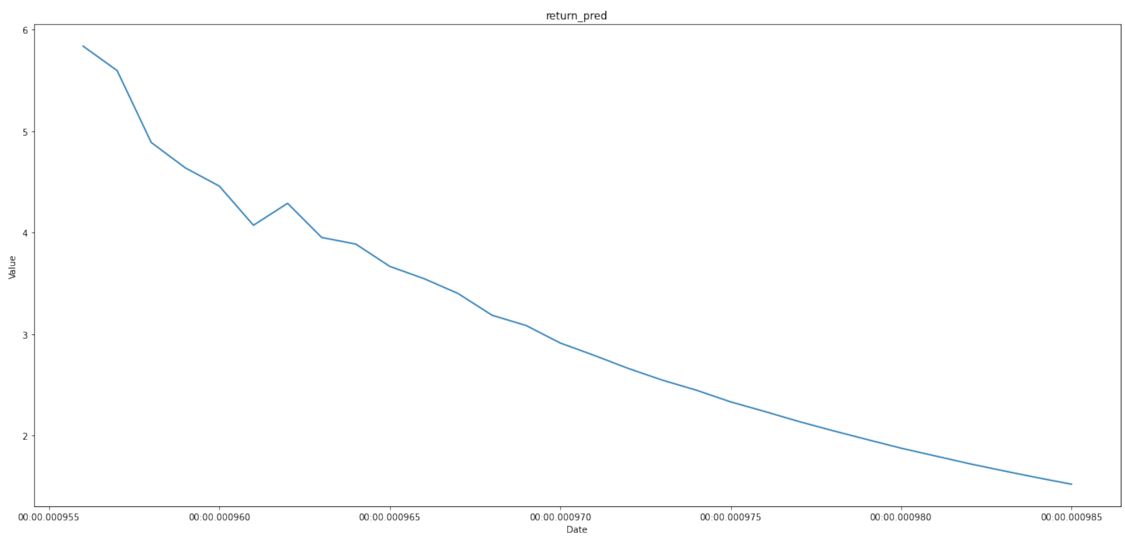
[ ]:
```python
test_vs_pred=pd.concat([test_df,predictions],axis=1)
test_vs_pred.plot(figsize=(22,10))
# Loop through each column and plot a graph
for col in test_vs_pred.columns:
    plt.figure(figsize=(22,10))
    plt.plot(test_vs_pred[col])
    plt.title(col)
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.show()
```

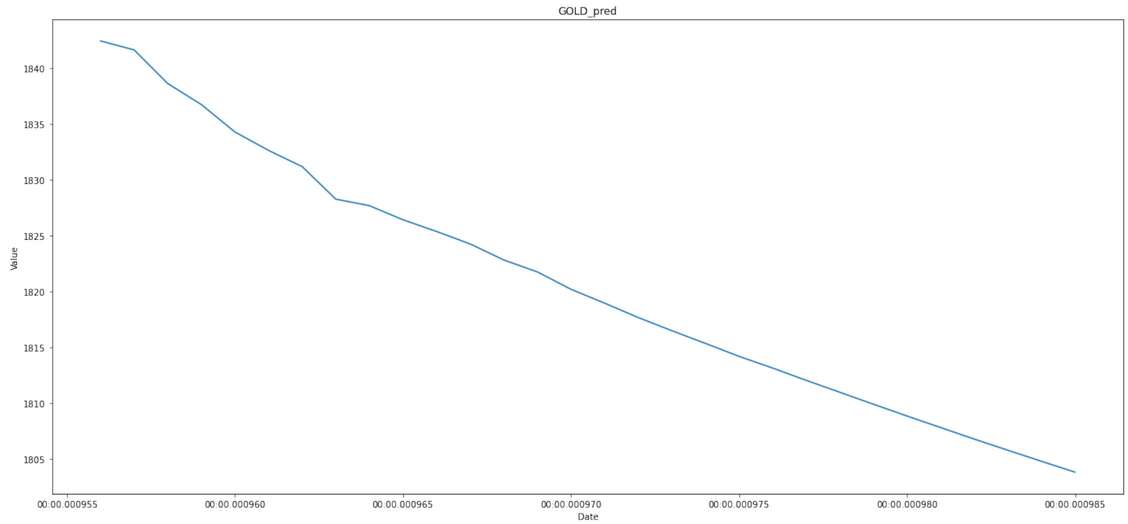


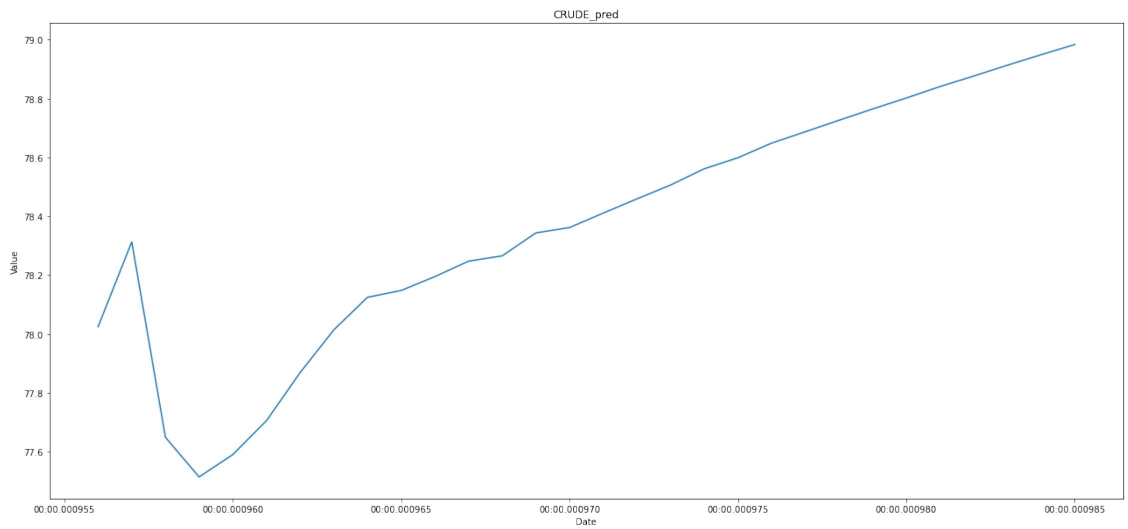

return
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
Value
2023-03-01
2023-03-08
2023-03-15
2023-03-22
2023-04-01
2023-04-08
2023-04-15
Date


GOLD
2050
2000
1950
1900
1850
Value
2023-03-01
2023-03-08
2023-03-15
2023-03-22
2023-04-01
2023-04-08
2023-04-15
Date


CRUDE
82.5
80.0
77.5
75.0
72.5
70.0
67.5
Value
2023-03-01
2023-03-08
2023-03-15
2023-03-22
2023-04-01
2023-04-08
2023-04-15
Date


return_pred
6
5
4
3
2
Value
00:00.000955
00:00.000960
00:00.000965
00:00.000970
00:00.000975
00:00.000980
00:00.000985
Date





[ ]:
```
rmse_return=math.
 ↪sqrt(mean_squared_error(predictions['return_pred'],test_df['return']))
print('Mean value of return is : {}. Root Mean Squared Error is :{}'.
 ↪format(mean(test_df['return']),rmse_return))
print()

rmse_exrate=math.
 ↪sqrt(mean_squared_error(predictions['CRUDE_pred'],test_df['CRUDE']))
print('Mean value of CRUDE is : {}. Root Mean Squared Error is :{}'.
 ↪format(mean(test_df['CRUDE']),rmse_exrate))
print()
```

```
rmse_gold=math.
 ↪sqrt(mean_squared_error(predictions['GOLD_pred'],test_df['GOLD']))
print('Mean value of GOLD is : {}. Root Mean Squared Error is :{}'.
 ↪format(mean(test_df['GOLD']),rmse_gold))
print()
```

```
Mean value of return is : 3.092511206109605. Root Mean Squared Error is
:5.5275953926818815

Mean value of CRUDE is : 75.44566666666667. Root Mean Squared Error is
:5.739175151643373

Mean value of GOLD is : 1953.5033333333333. Root Mean Squared Error is
:153.38642360946992
```