```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving FINAL_USO.csv to FINAL_USO (3).csv
```

```
# LinearRegression is a machine learning library for linear regression
from sklearn.linear_model import LinearRegression

# pandas and numpy are used for data manipulation
import pandas as pd
import numpy as np

# matplotlib and seaborn are used for plotting graphs
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-darkgrid')
```

```
<ipython-input-67-067b9257ae8f>:11: MatplotlibDeprecationWarning: The
seaborn styles shipped by Matplotlib are deprecated since 3.6, as they
no longer correspond to the styles shipped by seaborn. However, they
will remain available as 'seaborn-v0_8-<style>'. Alternatively,
directly use the seaborn API instead.
  plt.style.use('seaborn-darkgrid')
```

```
print (uploaded['FINAL_USO.csv'][:200].decode('utf-8') + '...')
```

```
Date,Open,High,Low,Close,Adj
Close,Volume,SP_open,SP_high,SP_low,SP_close,SP_Ajclose,SP_volume,DJ_o
pen,DJ_high,DJ_low,DJ_close,DJ_Ajclose,DJ_volume,EG_open,EG_high,EG_lo
w,EG_close,EG_Ajclose,EG_volume...
```

```
import pandas as pd
import io

df = pd.read_csv(io.StringIO(uploaded['FINAL_USO.csv'].decode('utf-
8')))
df
```

```
           Date        Open        High         Low       Close   Adj
Close  \
0     2011-12-15  154.740005  154.949997  151.710007  152.330002
152.330002
1     2011-12-16  154.309998  155.369995  153.899994  155.229996
155.229996
2     2011-12-19  155.479996  155.860001  154.360001  154.869995
154.869995
3     2011-12-20  156.820007  157.429993  156.580002  156.979996
156.979996
```

```
4     2011-12-21  156.979996  157.529999  156.130005  157.160004
157.160004
...          ...         ...         ...         ...         ...
...
1713  2018-12-24  119.570000  120.139999  119.570000  120.019997
120.019997
1714  2018-12-26  120.620003  121.000000  119.570000  119.660004
119.660004
1715  2018-12-27  120.570000  120.900002  120.139999  120.570000
120.570000
1716  2018-12-28  120.800003  121.080002  120.720001  121.059998
121.059998
1717  2018-12-31  120.980003  121.260002  120.830002  121.250000
121.250000

        Volume     SP_open     SP_high      SP_low  ...    GDX_Low
GDX_Close  \
0     21521900  123.029999  123.199997  121.989998  ...  51.570000
51.680000
1     18124300  122.230003  122.949997  121.300003  ...  52.040001
52.680000
2     12547200  122.059998  122.320000  120.029999  ...  51.029999
51.169998
3      9136300  122.180000  124.139999  120.370003  ...  52.369999
52.990002
4     11996100  123.930000  124.360001  122.750000  ...  52.419998
52.959999
...        ...         ...         ...         ...  ...        ...
...
1713   9736400  239.039993  240.839996  234.270004  ...  20.650000
21.090000
1714  14293500  235.970001  246.179993  233.759995  ...  20.530001
20.620001
1715  11874400  242.570007  248.289993  238.960007  ...  20.700001
20.969999
1716   6864700  249.580002  251.399994  246.449997  ...  20.570000
20.600000
1717   8449400  249.559998  250.190002  247.470001  ...  20.559999
21.090000

      GDX_Adj Close  GDX_Volume   USO_Open   USO_High    USO_Low
USO_Close  \
0         48.973877    20605600  36.900002  36.939999  36.049999
36.130001
1         49.921513    16285400  36.180000  36.500000  35.730000
36.270000
2         48.490578    15120200  36.389999  36.450001  35.930000
36.200001
3         50.215282    11644900  37.299999  37.610001  37.220001
37.560001
```

```
4         50.186852     8724300  37.669998  38.240002  37.520000
38.110001
...             ...         ...        ...        ...        ...
...
1713      21.090000    60507000   9.490000   9.520000   9.280000
9.290000
1714      20.620001    76365200   9.250000   9.920000   9.230000
9.900000
1715      20.969999    52393000   9.590000   9.650000   9.370000
9.620000
1716      20.600000    49835000   9.540000   9.650000   9.380000
9.530000
1717      21.090000    53866600   9.630000   9.710000   9.440000
9.660000

      USO_Adj Close  USO_Volume
0         36.130001    12616700
1         36.270000    12578800
2         36.200001     7418200
3         37.560001    10041600
4         38.110001    10728000
...             ...         ...
1713       9.290000    21598200
1714       9.900000    40978800
1715       9.620000    36578700
1716       9.530000    22803400
1717       9.660000    28417400

[1718 rows x 81 columns]
```

```
#Basic information

df.info()

#Describe the data

df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1718 entries, 0 to 1717
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           1718 non-null   object
 1   Open           1718 non-null   float64
 2   High           1718 non-null   float64
 3   Low            1718 non-null   float64
 4   Close          1718 non-null   float64
 5   Adj Close      1718 non-null   float64
 6   Volume         1718 non-null   int64
```

7	SP_open	1718	non-null	float64
8	SP_high	1718	non-null	float64
9	SP_low	1718	non-null	float64
10	SP_close	1718	non-null	float64
11	SP_Ajclose	1718	non-null	float64
12	SP_volume	1718	non-null	int64
13	DJ_open	1718	non-null	float64
14	DJ_high	1718	non-null	float64
15	DJ_low	1718	non-null	float64
16	DJ_close	1718	non-null	float64
17	DJ_Ajclose	1718	non-null	float64
18	DJ_volume	1718	non-null	int64
19	EG_open	1718	non-null	float64
20	EG_high	1718	non-null	float64
21	EG_low	1718	non-null	float64
22	EG_close	1718	non-null	float64
23	EG_Ajclose	1718	non-null	float64
24	EG_volume	1718	non-null	int64
25	EU_Price	1718	non-null	float64
26	EU_open	1718	non-null	float64
27	EU_high	1718	non-null	float64
28	EU_low	1718	non-null	float64
29	EU_Trend	1718	non-null	int64
30	OF_Price	1718	non-null	float64
31	OF_Open	1718	non-null	float64
32	OF_High	1718	non-null	float64
33	OF_Low	1718	non-null	float64
34	OF_Volume	1718	non-null	int64
35	OF_Trend	1718	non-null	int64
36	OS_Price	1718	non-null	float64
37	OS_Open	1718	non-null	float64
38	OS_High	1718	non-null	float64
39	OS_Low	1718	non-null	float64
40	OS_Trend	1718	non-null	int64
41	SF_Price	1718	non-null	int64
42	SF_Open	1718	non-null	int64
43	SF_High	1718	non-null	int64
44	SF_Low	1718	non-null	int64
45	SF_Volume	1718	non-null	int64
46	SF_Trend	1718	non-null	int64
47	USB_Price	1718	non-null	float64
48	USB_Open	1718	non-null	float64
49	USB_High	1718	non-null	float64
50	USB_Low	1718	non-null	float64
51	USB_Trend	1718	non-null	int64
52	PLT_Price	1718	non-null	float64
53	PLT_Open	1718	non-null	float64
54	PLT_High	1718	non-null	float64
55	PLT_Low	1718	non-null	float64
56	PLT_Trend	1718	non-null	int64

```
 57  PLD_Price      1718 non-null   float64
 58  PLD_Open       1718 non-null   float64
 59  PLD_High       1718 non-null   float64
 60  PLD_Low        1718 non-null   float64
 61  PLD_Trend      1718 non-null   int64
 62  RHO_PRICE      1718 non-null   int64
 63  USDI_Price     1718 non-null   float64
 64  USDI_Open      1718 non-null   float64
 65  USDI_High      1718 non-null   float64
 66  USDI_Low       1718 non-null   float64
 67  USDI_Volume    1718 non-null   int64
 68  USDI_Trend     1718 non-null   int64
 69  GDX_Open       1718 non-null   float64
 70  GDX_High       1718 non-null   float64
 71  GDX_Low        1718 non-null   float64
 72  GDX_Close      1718 non-null   float64
 73  GDX_Adj Close  1718 non-null   float64
 74  GDX_Volume     1718 non-null   int64
 75  USO_Open       1718 non-null   float64
 76  USO_High       1718 non-null   float64
 77  USO_Low        1718 non-null   float64
 78  USO_Close      1718 non-null   float64
 79  USO_Adj Close  1718 non-null   float64
 80  USO_Volume     1718 non-null   int64
dtypes: float64(58), int64(22), object(1)
memory usage: 1.1+ MB
```

	Open	High	Low	Close	Adj Close
count	1718.000000	1718.000000	1718.000000	1718.000000	1718.000000
mean	127.323434	127.854237	126.777695	127.319482	127.319482
std	17.526993	17.631189	17.396513	17.536269	17.536269
min	100.919998	100.989998	100.230003	100.500000	100.500000
25%	116.220001	116.540001	115.739998	116.052502	116.052502
50%	121.915001	122.325001	121.369999	121.795002	121.795002
75%	128.427494	129.087498	127.840001	128.470001	128.470001
max	173.199997	174.070007	172.919998	173.610001	173.610001

	Volume	SP_open	SP_high	SP_low	SP_close	...
count	1.718000e+03	1718.000000	1718.000000	1718.000000		

```
1718.000000  ...
mean   8.446327e+06   204.490023   205.372637   203.487014
204.491222  ...
std    4.920731e+06    43.831928    43.974644    43.618940
43.776999  ...
min    1.501600e+06   122.059998   122.320000   120.029999
120.290001  ...
25%    5.412925e+06   170.392498   170.962506   169.577499
170.397500  ...
50%    7.483900e+06   205.464996   206.459999   204.430000
205.529999  ...
75%    1.020795e+07   237.292500   237.722500   236.147503
236.889996  ...
max    9.380420e+07   293.089996   293.940002   291.809998
293.579987  ...

           GDX_Low    GDX_Close  GDX_Adj Close    GDX_Volume
USO_Open  \
count  1718.000000  1718.000000    1718.000000  1.718000e+03
1718.000000
mean     26.384575    26.715012      25.924624  4.356515e+07
22.113417
std      10.490908    10.603110       9.886570  2.909151e+07
11.431056
min      12.400000    12.470000      12.269618  4.729000e+06
7.820000
25%      20.355000    20.585000      20.180950  2.259968e+07
11.420000
50%      22.870001    23.054999      22.677604  3.730465e+07
16.450000
75%      26.797500    27.317500      26.478154  5.697055e+07
34.419998
max      56.770000    57.470001      54.617039  2.321536e+08
41.599998

          USO_High      USO_Low    USO_Close  USO_Adj Close
USO_Volume
count  1718.000000  1718.000000  1718.000000    1718.000000
1.718000e+03
mean     22.307148    21.904657    22.109051      22.109051
1.922313e+07
std      11.478671    11.373997    11.432787      11.432787
1.575743e+07
min       8.030000     7.670000     7.960000       7.960000
1.035100e+06
25%      11.500000    11.300000    11.392500      11.392500
6.229500e+06
50%      16.635001    16.040000    16.345000      16.345000
1.613015e+07
75%      34.667499    34.110000    34.417499      34.417499
```

```
2.672375e+07
max      42.299999    41.299999    42.009998      42.009998
1.102657e+08

[8 rows x 80 columns]
```

```
#duplicate values
df.duplicated().sum()
```

```
0
```

```
df = df.dropna()
print(df)
```

```
            Date        Open        High         Low       Close   Adj
Close  \
0     2011-12-15  154.740005  154.949997  151.710007  152.330002
152.330002
1     2011-12-16  154.309998  155.369995  153.899994  155.229996
155.229996
2     2011-12-19  155.479996  155.860001  154.360001  154.869995
154.869995
3     2011-12-20  156.820007  157.429993  156.580002  156.979996
156.979996
4     2011-12-21  156.979996  157.529999  156.130005  157.160004
157.160004
...          ...         ...         ...         ...         ...
...
1713  2018-12-24  119.570000  120.139999  119.570000  120.019997
120.019997
1714  2018-12-26  120.620003  121.000000  119.570000  119.660004
119.660004
1715  2018-12-27  120.570000  120.900002  120.139999  120.570000
120.570000
1716  2018-12-28  120.800003  121.080002  120.720001  121.059998
121.059998
1717  2018-12-31  120.980003  121.260002  120.830002  121.250000
121.250000

        Volume     SP_open     SP_high      SP_low  ...    GDX_Low
GDX_Close  \
0     21521900  123.029999  123.199997  121.989998  ...  51.570000
51.680000
1     18124300  122.230003  122.949997  121.300003  ...  52.040001
52.680000
2     12547200  122.059998  122.320000  120.029999  ...  51.029999
51.169998
3      9136300  122.180000  124.139999  120.370003  ...  52.369999
52.990002
4     11996100  123.930000  124.360001  122.750000  ...  52.419998
52.959999
```

```
...        ...         ...         ...         ...  ...        ...
...
1713   9736400  239.039993  240.839996  234.270004  ...  20.650000
21.090000
1714  14293500  235.970001  246.179993  233.759995  ...  20.530001
20.620001
1715  11874400  242.570007  248.289993  238.960007  ...  20.700001
20.969999
1716   6864700  249.580002  251.399994  246.449997  ...  20.570000
20.600000
1717   8449400  249.559998  250.190002  247.470001  ...  20.559999
21.090000

      GDX_Adj Close  GDX_Volume   USO_Open   USO_High    USO_Low
USO_Close  \
0         48.973877    20605600  36.900002  36.939999  36.049999
36.130001
1         49.921513    16285400  36.180000  36.500000  35.730000
36.270000
2         48.490578    15120200  36.389999  36.450001  35.930000
36.200001
3         50.215282    11644900  37.299999  37.610001  37.220001
37.560001
4         50.186852     8724300  37.669998  38.240002  37.520000
38.110001
...             ...         ...        ...        ...        ...
...
1713      21.090000    60507000   9.490000   9.520000   9.280000
9.290000
1714      20.620001    76365200   9.250000   9.920000   9.230000
9.900000
1715      20.969999    52393000   9.590000   9.650000   9.370000
9.620000
1716      20.600000    49835000   9.540000   9.650000   9.380000
9.530000
1717      21.090000    53866600   9.630000   9.710000   9.440000
9.660000

      USO_Adj Close  USO_Volume
0         36.130001    12616700
1         36.270000    12578800
2         36.200001     7418200
3         37.560001    10041600
4         38.110001    10728000
...             ...         ...
1713       9.290000    21598200
1714       9.900000    40978800
1715       9.620000    36578700
1716       9.530000    22803400
1717       9.660000    28417400
```

```
[1718 rows x 81 columns]
```

```
dfcr=df.corr()
dfcr
```

```
<ipython-input-73-bc6fb3df6887>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  dfcr=df.corr()
```

```
                    Open      High       Low     Close  Adj Close
Volume  \
Open            1.000000  0.999515  0.999442  0.998976   0.998976
0.251921
High            0.999515  1.000000  0.999262  0.999535   0.999535
0.261064
Low             0.999442  0.999262  1.000000  0.999532   0.999532
0.237031
Close           0.998976  0.999535  0.999532  1.000000   1.000000
0.246778
Adj Close       0.998976  0.999535  0.999532  1.000000   1.000000
0.246778
...                  ...       ...       ...       ...        ...
...
USO_High        0.634864  0.637208  0.633623  0.635311   0.635311
0.083064
USO_Low         0.634277  0.636538  0.633140  0.634732   0.634732
0.080475
USO_Close       0.635138  0.637483  0.633994  0.635675   0.635675
0.081642
USO_Adj Close   0.635138  0.637483  0.633994  0.635675   0.635675
0.081642
USO_Volume     -0.455920 -0.454913 -0.457628 -0.456193  -0.456193
0.069580

                 SP_open   SP_high    SP_low  SP_close  ...
GDX_Low  \
Open           -0.684314 -0.684597 -0.683464 -0.683998  ...  0.975479

High           -0.688118 -0.688365 -0.687325 -0.687817  ...  0.975650

Low            -0.680911 -0.681242 -0.679988 -0.680567  ...  0.975337

Close          -0.684618 -0.684904 -0.683750 -0.684284  ...  0.975561

Adj Close      -0.684618 -0.684904 -0.683750 -0.684284  ...  0.975561

...                  ...       ...       ...       ...  ...       ...
```

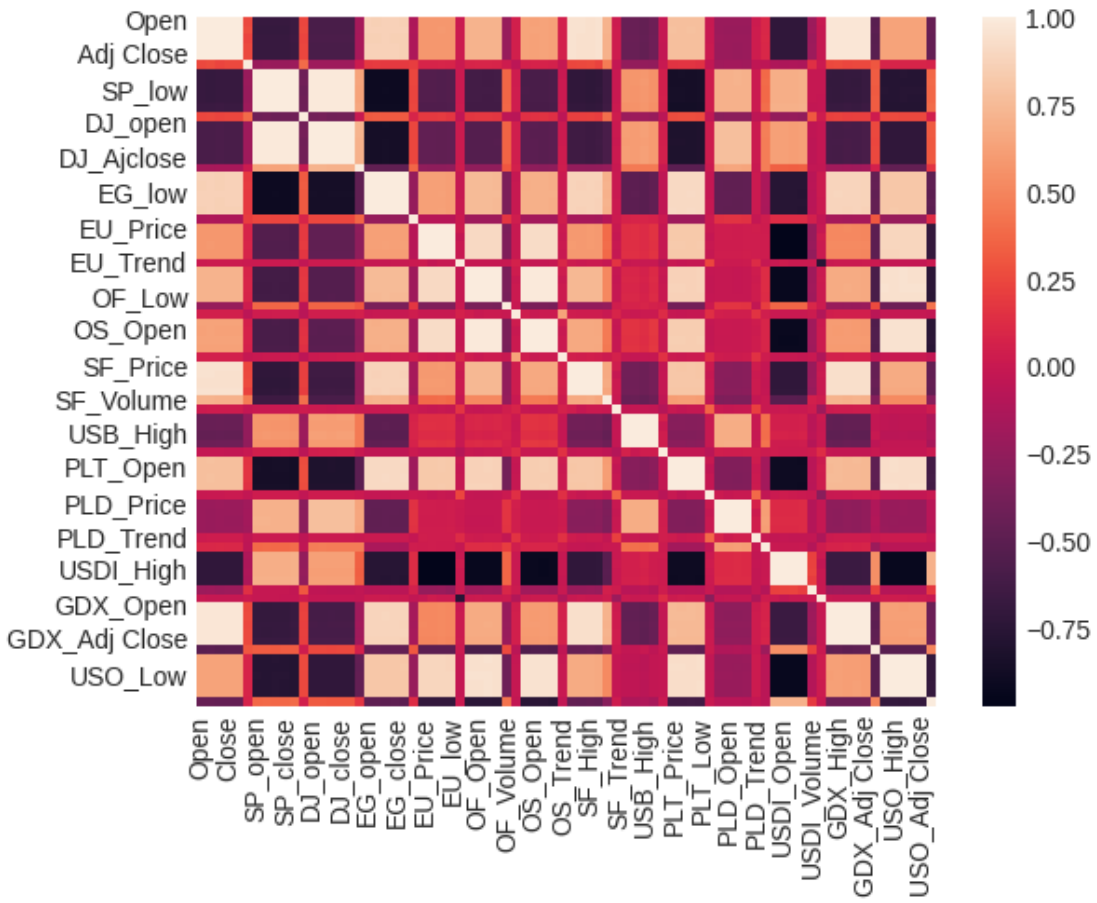
```
USO_High      -0.774626 -0.775482 -0.773550 -0.774799  ...  0.614587

USO_Low       -0.771235 -0.772154 -0.770087 -0.771396  ...  0.613844

USO_Close     -0.773099 -0.773957 -0.771928 -0.773159  ...  0.614733

USO_Adj Close -0.773099 -0.773957 -0.771928 -0.773159  ...  0.614733

USO_Volume     0.375568  0.377996  0.371390  0.374658  ... -0.429839


               GDX_Close  GDX_Adj Close  GDX_Volume  USO_Open
USO_High  \
Open            0.974596       0.974098   -0.514230  0.634872
0.634864
High            0.975341       0.974746   -0.508782  0.637101
0.637208
Low             0.974568       0.974182   -0.519988  0.633591
0.633623
Close           0.975459       0.974980   -0.514616  0.635197
0.635311
Adj Close       0.975459       0.974980   -0.514616  0.635197
0.635311
...                  ...            ...         ...       ...
..
.
USO_High        0.614766       0.600523   -0.522581  0.999857
1.000000
USO_Low         0.613931       0.599819   -0.523956  0.999848
0.999818
USO_Close       0.614915       0.600756   -0.523801  0.999699
0.999867
USO_Adj Close   0.614915       0.600756   -0.523801  0.999699
0.999867
USO_Volume     -0.426553      -0.421065    0.498816 -0.699000 -
0.695678

                USO_Low  USO_Close  USO_Adj Close  USO_Volume
Open           0.634277   0.635138       0.635138   -0.455920
High           0.636538   0.637483       0.637483   -0.454913
Low            0.633140   0.633994       0.633994   -0.457628
Close          0.634732   0.635675       0.635675   -0.456193
Adj Close      0.634732   0.635675       0.635675   -0.456193
...                 ...        ...            ...         ...
USO_High       0.999818   0.999867       0.999867   -0.695678
USO_Low        1.000000   0.999879       0.999879   -0.702665
USO_Close      0.999879   1.000000       1.000000   -0.699221
USO_Adj Close  0.999879   1.000000       1.000000   -0.699221
USO_Volume    -0.702665  -0.699221      -0.699221    1.000000
```

```
[80 rows x 80 columns]
```

```
import seaborn as sns
import plotly.graph_objects as go

plot_columns = df.drop('Date', axis=1).columns

#extract color palette, the palette can be changed
pal = list(sns.color_palette(palette='viridis',
n_colors=len(plot_columns)).as_hex())

fig = go.Figure()
for d,p in zip(plot_columns, pal):
    fig.add_trace(go.Scatter(x = df['Date'],
                             y = df[d],
                             name = d,
                             line_color = p,
                             fill=None))   #tozeroy

fig.show()
```

```
import seaborn as sns
sns.heatmap(df.corr())
```

```
<ipython-input-76-534f4f3c80b7>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In
a future version, it will default to False. Select only valid columns
or specify the value of numeric_only to silence this warning.


<Axes: >
```




```
# Select the two columns you want
df1 = df[['Date', 'Close']]

# Print the new dataframe
print(df1)
```

```
            Date       Close
0     2011-12-15  152.330002
1     2011-12-16  155.229996
2     2011-12-19  154.869995
3     2011-12-20  156.979996
4     2011-12-21  157.160004
...          ...         ...
1713  2018-12-24  120.019997
1714  2018-12-26  119.660004
1715  2018-12-27  120.570000
1716  2018-12-28  121.059998
1717  2018-12-31  121.250000

[1718 rows x 2 columns]
```

```
# set the 'Name' column as the index
df1.set_index('Date', inplace=True)
df1
```

```
                 Close
Date
2011-12-15  152.330002
2011-12-16  155.229996
2011-12-19  154.869995
2011-12-20  156.979996
2011-12-21  157.160004
...                ...
2018-12-24  120.019997
2018-12-26  119.660004
2018-12-27  120.570000
2018-12-28  121.059998
2018-12-31  121.250000

[1718 rows x 1 columns]
```

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.style.use('seaborn-darkgrid')
df1.Close.plot(figsize=(8, 5),color='r')
plt.ylabel("Gold ETF Prices")
plt.title("Gold ETF Price Series")
plt.show()
```

```
<ipython-input-83-00aebf5a883e>:4: MatplotlibDeprecationWarning:

The seaborn styles shipped by Matplotlib are deprecated since 3.6, as
they no longer correspond to the styles shipped by seaborn. However,
they will remain available as 'seaborn-v0_8-<style>'. Alternatively,
directly use the seaborn API instead.
```


```
df1['S_3'] = df1['Close'].rolling(window=3).mean()
df1['S_9'] = df1['Close'].rolling(window=9).mean()
df1['next_day_price'] = df1['Close'].shift(-1)

df1 = df1.dropna()
X = df1[['S_3', 'S_9']]

# Define dependent variable
y = df1['next_day_price']
df1
```

```
<ipython-input-85-5aa089d742b9>:1: 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

<ipython-input-85-5aa089d742b9>:2: 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

<ipython-input-85-5aa089d742b9>:3: 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


                Close         S_3         S_9  next_day_price
Date
2012-01-10  158.639999  157.446665  155.123333      159.669998
2012-01-11  159.669998  158.269999  156.083333      160.380005
2012-01-12  160.380005  159.563334  157.198890      159.259995
2012-01-13  159.259995  159.769999  158.006666      160.500000
2012-01-17  160.500000  160.046667  158.515556      161.600006
...                ...         ...         ...             ...
2018-12-20  119.239998  118.273333  117.810000      118.720001
2018-12-21  118.720001  118.463333  117.925556      120.019997
2018-12-24  120.019997  119.326665  118.201111      119.660004
2018-12-26  119.660004  119.466667  118.408889      120.570000
2018-12-27  120.570000  120.083334  118.746667      121.059998

[1700 rows x 4 columns]
```

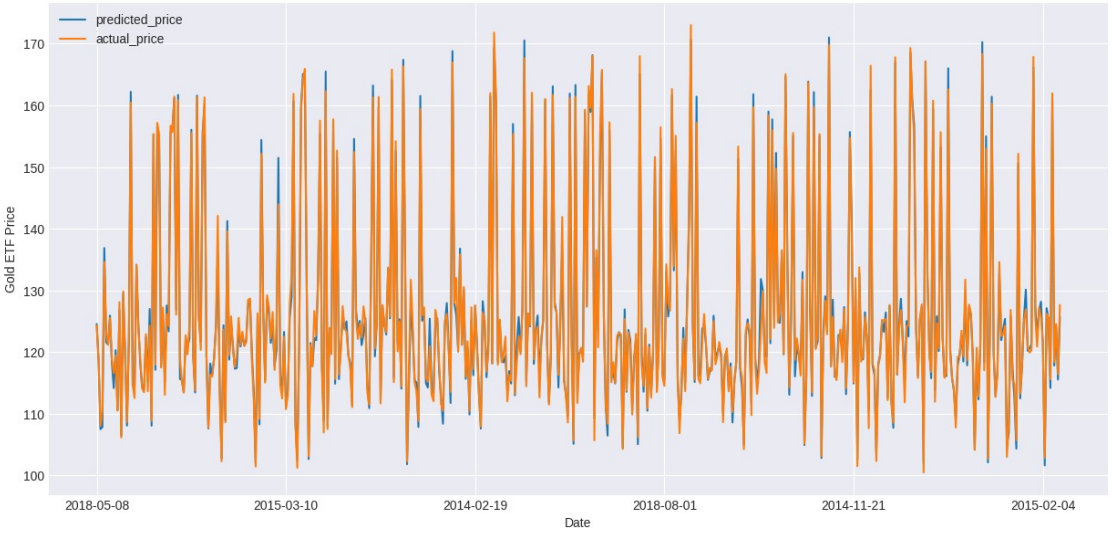
```
from sklearn.model_selection import train_test_split
import numpy as np

X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.3,random_state=80)

from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)

print('Error', np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
Error 1.47547959030051
```

```
Accuracy=r2_score(y_test,y_pred)*100
print(" Accuracy of the model is %.2f" %Accuracy)
```

```
 Accuracy of the model is 99.19
```

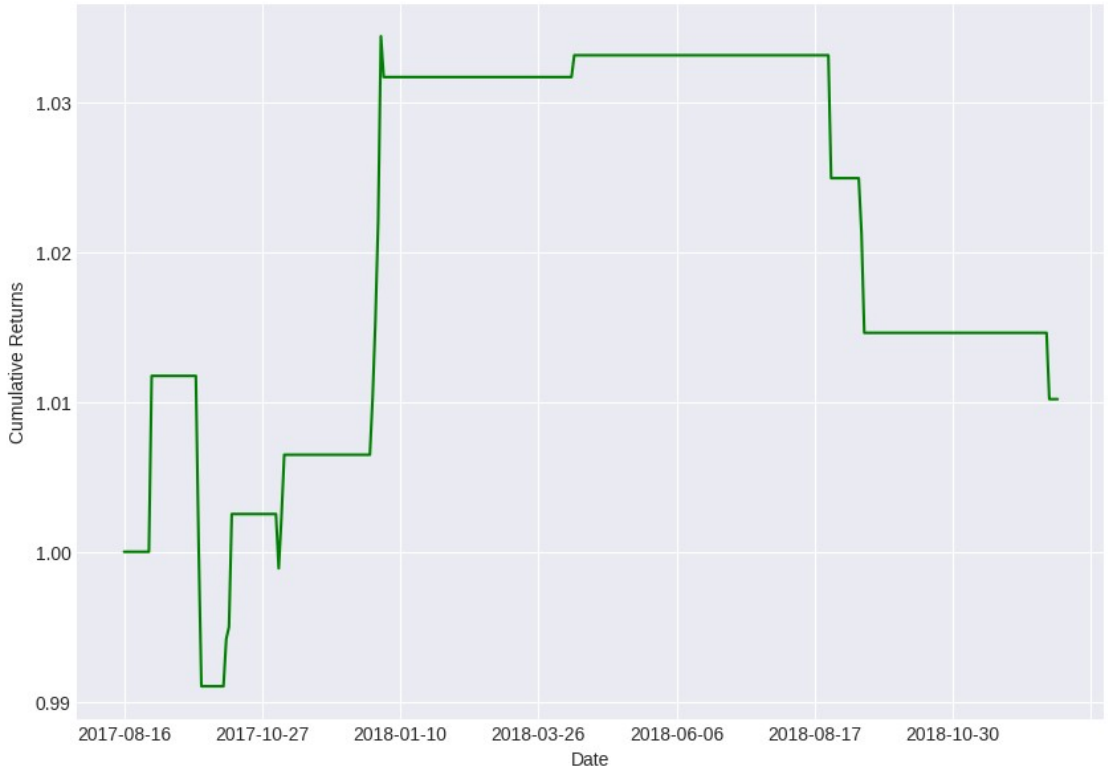
```
predicted_price = pd.DataFrame(
    y_pred, index=y_test.index, columns=['price'])
predicted_price.plot(figsize=(15, 7))
y_test.plot()
plt.legend(['predicted_price', 'actual_price'])
plt.ylabel("Gold ETF Price")
plt.show()
```




```
r2_score = lin_reg.score(X_test, y_test)*100
float("{0:.2f}".format(r2_score))
```

```
99.19
```

```
t = .8
t = int(t*len(df1))

gold = pd.DataFrame()

gold['price'] = df1[t:]['Close']
gold['predicted_price_next_day'] = predicted_price
gold['actual_price_next_day'] = y_test
gold['gold_returns'] = gold['price'].pct_change().shift(-1)

gold['signal'] = np.where(gold.predicted_price_next_day.shift(1) <
gold.predicted_price_next_day,1,0)

gold['strategy_returns'] = gold.signal * gold['gold_returns']
((gold['strategy_returns']
+1).cumprod()).plot(figsize=(10,7),color='g')
plt.ylabel('Cumulative Returns')
plt.show()
```



df

	Date	Open	High	Low	Close	Adj Close
0	2011-12-15	154.740005	154.949997	151.710007	152.330002	152.330002
1	2011-12-16	154.309998	155.369995	153.899994	155.229996	155.229996
2	2011-12-19	155.479996	155.860001	154.360001	154.869995	154.869995
3	2011-12-20	156.820007	157.429993	156.580002	156.979996	156.979996
4	2011-12-21	156.979996	157.529999	156.130005	157.160004	157.160004
...	...	...	...	...	...	...
1713	2018-12-24	119.570000	120.139999	119.570000	120.019997	120.019997
1714	2018-12-26	120.620003	121.000000	119.570000	119.660004	119.660004
1715	2018-12-27	120.570000	120.900002	120.139999	120.570000	120.570000
1716	2018-12-28	120.800003	121.080002	120.720001	121.059998	121.059998
1717	2018-12-31	120.980003	121.260002	120.830002	121.250000	121.250000

```
        Volume     SP_open     SP_high      SP_low  ...    GDX_Low
GDX_Close  \
0     21521900  123.029999  123.199997  121.989998  ...  51.570000
51.680000
1     18124300  122.230003  122.949997  121.300003  ...  52.040001
52.680000
2     12547200  122.059998  122.320000  120.029999  ...  51.029999
51.169998
3      9136300  122.180000  124.139999  120.370003  ...  52.369999
52.990002
4     11996100  123.930000  124.360001  122.750000  ...  52.419998
52.959999
...        ...         ...         ...         ...  ...        ...
...
1713   9736400  239.039993  240.839996  234.270004  ...  20.650000
21.090000
1714  14293500  235.970001  246.179993  233.759995  ...  20.530001
20.620001
1715  11874400  242.570007  248.289993  238.960007  ...  20.700001
20.969999
1716   6864700  249.580002  251.399994  246.449997  ...  20.570000
20.600000
1717   8449400  249.559998  250.190002  247.470001  ...  20.559999
21.090000

      GDX_Adj Close  GDX_Volume   USO_Open   USO_High    USO_Low
USO_Close  \
0         48.973877    20605600  36.900002  36.939999  36.049999
36.130001
1         49.921513    16285400  36.180000  36.500000  35.730000
36.270000
2         48.490578    15120200  36.389999  36.450001  35.930000
36.200001
3         50.215282    11644900  37.299999  37.610001  37.220001
37.560001
4         50.186852     8724300  37.669998  38.240002  37.520000
38.110001
...             ...         ...        ...        ...        ...
...
1713      21.090000    60507000   9.490000   9.520000   9.280000
9.290000
1714      20.620001    76365200   9.250000   9.920000   9.230000
9.900000
1715      20.969999    52393000   9.590000   9.650000   9.370000
9.620000
1716      20.600000    49835000   9.540000   9.650000   9.380000
9.530000
1717      21.090000    53866600   9.630000   9.710000   9.440000
9.660000
```

```
      USO_Adj Close  USO_Volume
0         36.130001    12616700
1         36.270000    12578800
2         36.200001     7418200
3         37.560001    10041600
4         38.110001    10728000
...             ...         ...
1713       9.290000    21598200
1714       9.900000    40978800
1715       9.620000    36578700
1716       9.530000    22803400
1717       9.660000    28417400

[1718 rows x 81 columns]
```

```
# Select the two columns you want
df2 = df[['Date', 'Close']]

df2.set_index('Date', inplace=True)
df2
```

```
                Close
Date
2011-12-15  152.330002
2011-12-16  155.229996
2011-12-19  154.869995
2011-12-20  156.979996
2011-12-21  157.160004
...                ...
2018-12-24  120.019997
2018-12-26  119.660004
2018-12-27  120.570000
2018-12-28  121.059998
2018-12-31  121.250000

[1718 rows x 1 columns]
```

```
import datetime as dt
current_date = dt.datetime.now()

# Get the data
data = df2
data['S_3'] = data['Close'].rolling(window=3).mean()
data['S_9'] = data['Close'].rolling(window=9).mean()
data = data.dropna()

# Forecast the price
data['predicted_gold_price'] = lin_reg.predict(data[['S_3', 'S_9']])
data['signal'] = np.where(data.predicted_gold_price.shift(1) <
```

```
data.predicted_gold_price,"Buy","No Position")

# Print the forecast
data.tail(1)[['signal','predicted_gold_price']].T
```

```
<ipython-input-96-ff91ed4a94ed>:6: 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

<ipython-input-96-ff91ed4a94ed>:7: 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

<ipython-input-96-ff91ed4a94ed>:11: 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

<ipython-input-96-ff91ed4a94ed>:12: 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


Date                 2018-12-31
signal                      Buy
predicted_gold_price  121.298118
```