Exploring a Signal-Based Trading Strategy with Linear Regression!

November 14, 2024

```
In [1]: import pandas as pd
        import statsmodels.formula.api as smf
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
In [21]: import warnings

         warnings.filterwarnings("ignore")
```

Import all stock market data into DataFrame

```
SP500, NASDAQ, DJI (U.S. markets)

DAXI, CAC40 (European markets)

Nikkei, HSI, ALL Ordinaries (Asia-Pacific markets)
```

```
In [4]: aord = pd.DataFrame.from_csv('../data/indice/ALLOrdinary.csv')
        nikkei = pd.DataFrame.from_csv('../data/indice/Nikkei225.csv')
        hsi = pd.DataFrame.from_csv('../data/indice/HSI.csv')
        daxi = pd.DataFrame.from_csv('../data/indice/DAXI.csv')
        cac40 = pd.DataFrame.from_csv('../data/indice/CAC40.csv')
        sp500 = pd.DataFrame.from_csv('../data/indice/SP500.csv')
        dji = pd.DataFrame.from_csv('../data/indice/DJI.csv')
        nasdaq = pd.DataFrame.from_csv('../data/indice/nasdaq_composite.csv')
        spy = pd.DataFrame.from_csv('../data/indice/SPY.csv')
```

Step 1: Data Munging
Due to the timezone issues, we extract and calculate appropriate stock market data for analysis "Indicepanel" is the DataFrame of our trading model

```
In [7]: indicepanel=pd.DataFrame(index=spy.index)

        indicepanel['spy']=spy['Open'].shift(-1)-spy['Open']
        indicepanel['spy_lag1']=indicepanel['spy'].shift(1)
```

```
indicepanel['sp500']=sp500["Open"]-sp500['Open'].shift(1)
indicepanel['nasdaq']=nasdaq['Open']-nasdaq['Open'].shift(1)
indicepanel['dji']=dji['Open']-dji['Open'].shift(1)

indicepanel['cac40']=cac40['Open']-cac40['Open'].shift(1)
indicepanel['daxi']=daxi['Open']-daxi['Open'].shift(1)

indicepanel['aord']=aord['Close']-aord['Open']
indicepanel['hsi']=hsi['Close']-hsi['Open']
indicepanel['nikkei']=nikkei['Close']-nikkei['Open']
indicepanel['Price']=spy['Open']
```

I use use method 'fillna()' from dataframe to forward filling the Nan values
Then I have dropped the reminding Nan values using 'dropna()'

```
In [8]: indicepanel = indicepanel.fillna(method='ffill')
        indicepanel = indicepanel.dropna()
```

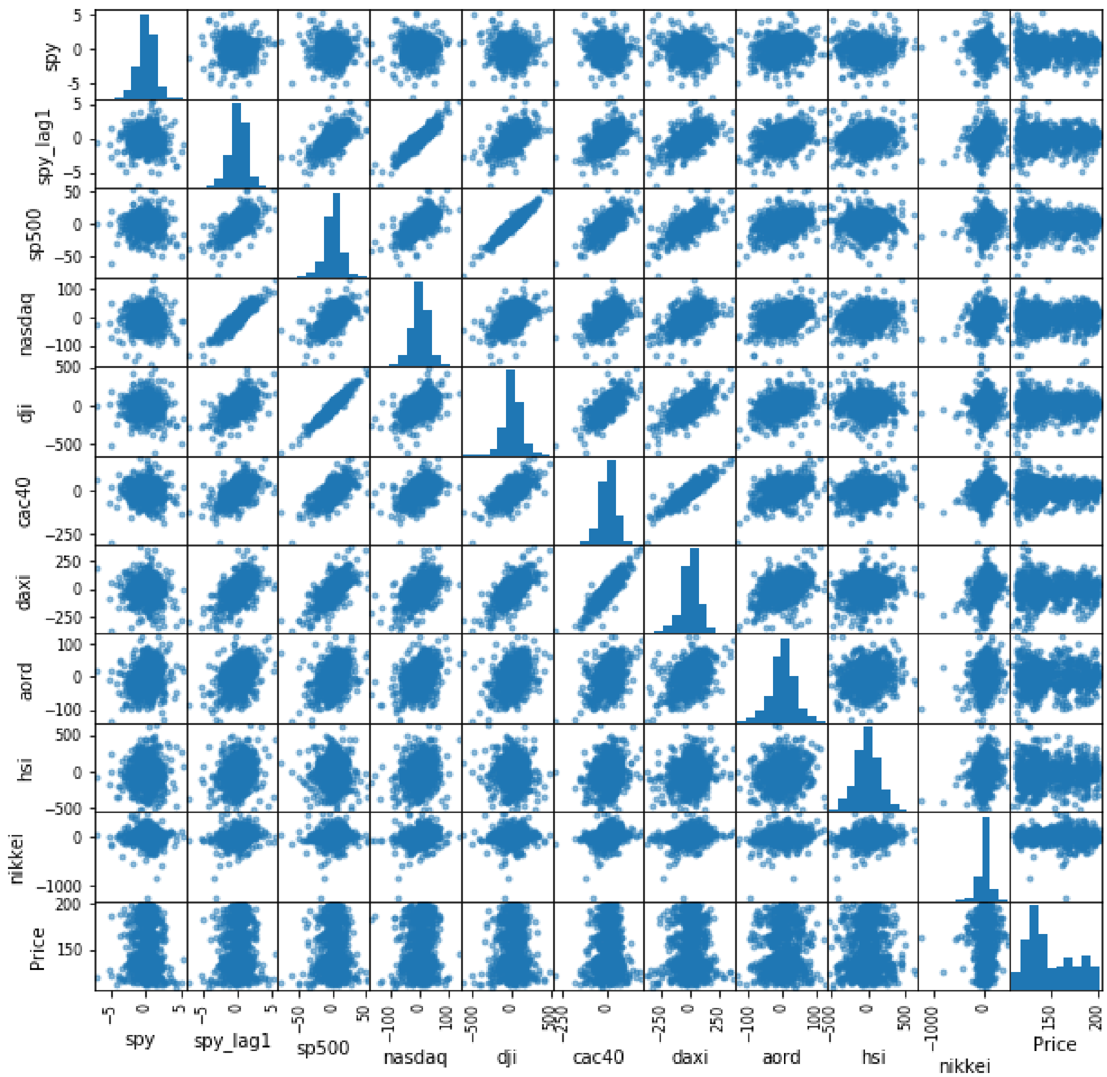
Step 2: Data Spliting
split the data into (1)train set and (2)test set

```
In [9]: Train = indicepanel.iloc[-2000:-1000, :]
        Test = indicepanel.iloc[-1000:, :]
        print(Train.shape, Test.shape)
```

```
(1000, 11) (1000, 11)
```

Step 3: Explore the train data set
Generate scatter matrix among all stock markets (and the price of SPY) to observe the association

```
In [10]: from pandas.tools.plotting import scatter_matrix
         sm = scatter_matrix(Train, figsize=(10, 10))
```



Step 4: Check the correlation of each index between spy

```
In [13]: corr_array = Train.iloc[:, :-1].corr()['spy']
         print(corr_array)
```

```
spy          1.000000
spy_lag1    -0.011623
sp500       -0.018632
nasdaq       0.012333
dji         -0.037097
cac40       -0.055304
daxi        -0.069735
aord         0.179638
hsi          0.031400
nikkei      -0.035048
```

```
Name: spy, dtype: float64
```

This code is implementing a multiple linear regression model to predict spy (the target variable) using several other variables (or "predictors") from the Train dataset, and then it applies the model to both the training and test datasets to generate predictions.

```
In [15]: formula = 'spy~spy_lag1+sp500+nasdaq+dji+cac40+aord+daxi+nikkei+hsi'
         lm = smf.ols(formula=formula, data=Train).fit()
         lm.summary()
```

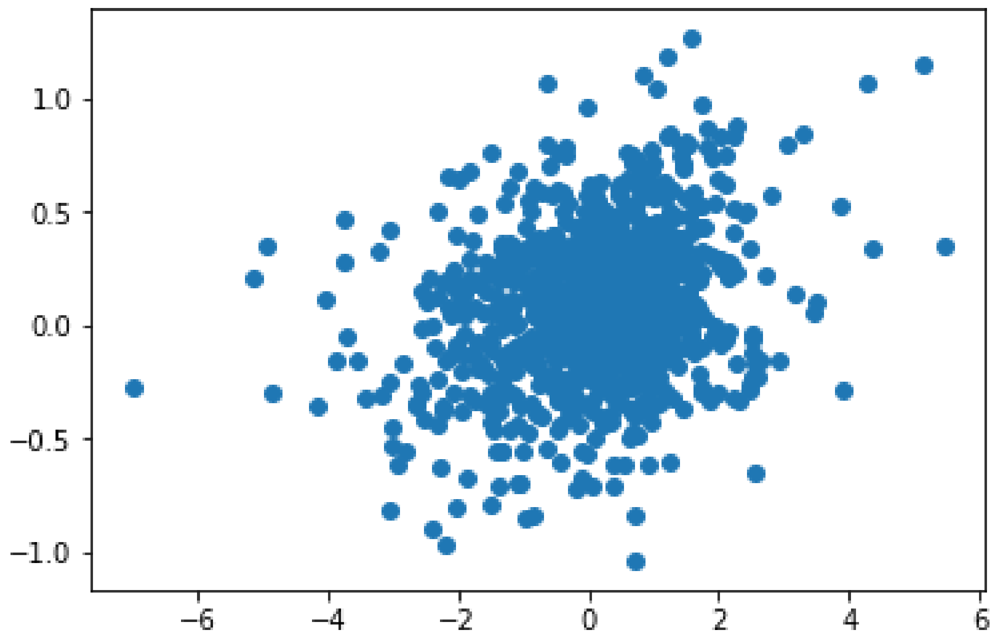
```
Out[15]: <class 'statsmodels.iolib.summary.Summary'>
         """
                                     OLS Regression Results
         ==============================================================================
         Dep. Variable:                    spy   R-squared:                       0.067
         Model:                            OLS   Adj. R-squared:                  0.059
         Method:                 Least Squares   F-statistic:                     7.962
         Date:                Thu, 14 Nov 2024   Prob (F-statistic):           1.97e-11
         Time:                        18:37:18   Log-Likelihood:                -1617.7
         No. Observations:                1000   AIC:                             3255.
         Df Residuals:                     990   BIC:                             3305.
         Df Model:                           9
         Covariance Type:            nonrobust
         ==============================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
         ------------------------------------------------------------------------------
         Intercept      0.0836      0.039      2.138      0.033       0.007       0.160
         spy_lag1      -0.1567      0.091     -1.730      0.084      -0.335       0.021
         sp500          0.0221      0.014      1.621      0.105      -0.005       0.049
         nasdaq         0.0040      0.004      1.066      0.287      -0.003       0.011
         dji           -0.0018      0.001     -1.248      0.212      -0.005       0.001
         cac40         -0.0003      0.002     -0.153      0.879      -0.004       0.004
         aord           0.0093      0.001      7.492      0.000       0.007       0.012
         daxi          -0.0025      0.001     -2.387      0.017      -0.005      -0.000
         nikkei        -0.0004      0.000     -1.264      0.207      -0.001       0.000
         hsi            0.0003      0.000      1.222      0.222      -0.000       0.001
         ==============================================================================
         Omnibus:                       91.018   Durbin-Watson:                   2.015
         Prob(Omnibus):                  0.000   Jarque-Bera (JB):              267.687
         Skew:                          -0.450   Prob(JB):                     7.45e-59
         Kurtosis:                       5.369   Cond. No.                         405.
         ==============================================================================

         Warnings:
         [1] Standard Errors assume that the covariance matrix of the errors is correctly specif
         """
```

Step 5: Make prediction

```
In [22]: Train['PredictedY'] = lm.predict(Train)
         Test['PredictedY'] = lm.predict(Test)
         plt.scatter(Train['spy'], Train['PredictedY'])
```

```
Out[22]: <matplotlib.collections.PathCollection at 0x7befaec76a20>
```



Step 6: Model evaluation - Statistical standard

We can measure the performance of our model using some statistical metrics - RMSE, Adjusted R2

```
In [17]: # RMSE - Root Mean Squared Error, Adjusted R^2
         def adjustedMetric(data, model, model_k, yname):
             data['yhat'] = model.predict(data)
             SST = ((data[yname] - data[yname].mean())**2).sum()
             SSR = ((data['yhat'] - data[yname].mean())**2).sum()
             SSE = ((data[yname] - data['yhat'])**2).sum()
             r2 = SSR/SST
             adjustR2 = 1 - (1-r2)*(data.shape[0] - 1)/(data.shape[0] -model_k -1)
             RMSE = (SSE/(data.shape[0] -model_k -1))**0.5
             return adjustR2, RMSE
```

```
In [18]: def assessTable(test, train, model, model_k, yname):
             r2test, RMSEtest = adjustedMetric(test, model, model_k, yname)
             r2train, RMSEtrain = adjustedMetric(train, model, model_k, yname)
             assessment = pd.DataFrame(index=['R2', 'RMSE'], columns=['Train', 'Test'])
             assessment['Train'] = [r2train, RMSEtrain]
             assessment['Test'] = [r2test, RMSEtest]
             return assessment
```

```
In [23]: # Get the assement table for model
         assessTable(Test, Train, lm, 9, 'spy')
```

```
Out[23]:          Train      Test
         R2    0.059020  0.067248
         RMSE  1.226068  1.701291
```

Train ˆ2: 0.059, which means the model explains about 5.9% of the variance in spy on the training set.

Test Rˆ2: 0.067, which means the model explains about 6.7% of the variance in spy on the test set.

These low values suggest that the model is not very good at explaining the variation in spy and may not be capturing complex patterns

Train RMSE: 1.226, meaning the average error on the training set is about 1.226 units.

Test RMSE: 1.701, indicating that the average prediction error on the test set is slightly larger, at 1.701 units.

The higher RMSE on the test set compared to the train set suggests that the model performs slightly worse on unseen data, which is expected. However, the difference here is moderate, implying that overfitting is not severe.

Summary

The model explains only a small percentage of the variance in spy (low Rˆ2),which indicates that this model doesn't capture the majority of the patterns or relationships in the data.

The difference in RMSE between the train and test sets is modest, so the model is not significantly overfitted but may be too simplistic for accurate predictions in a highly noisy stock market environment.

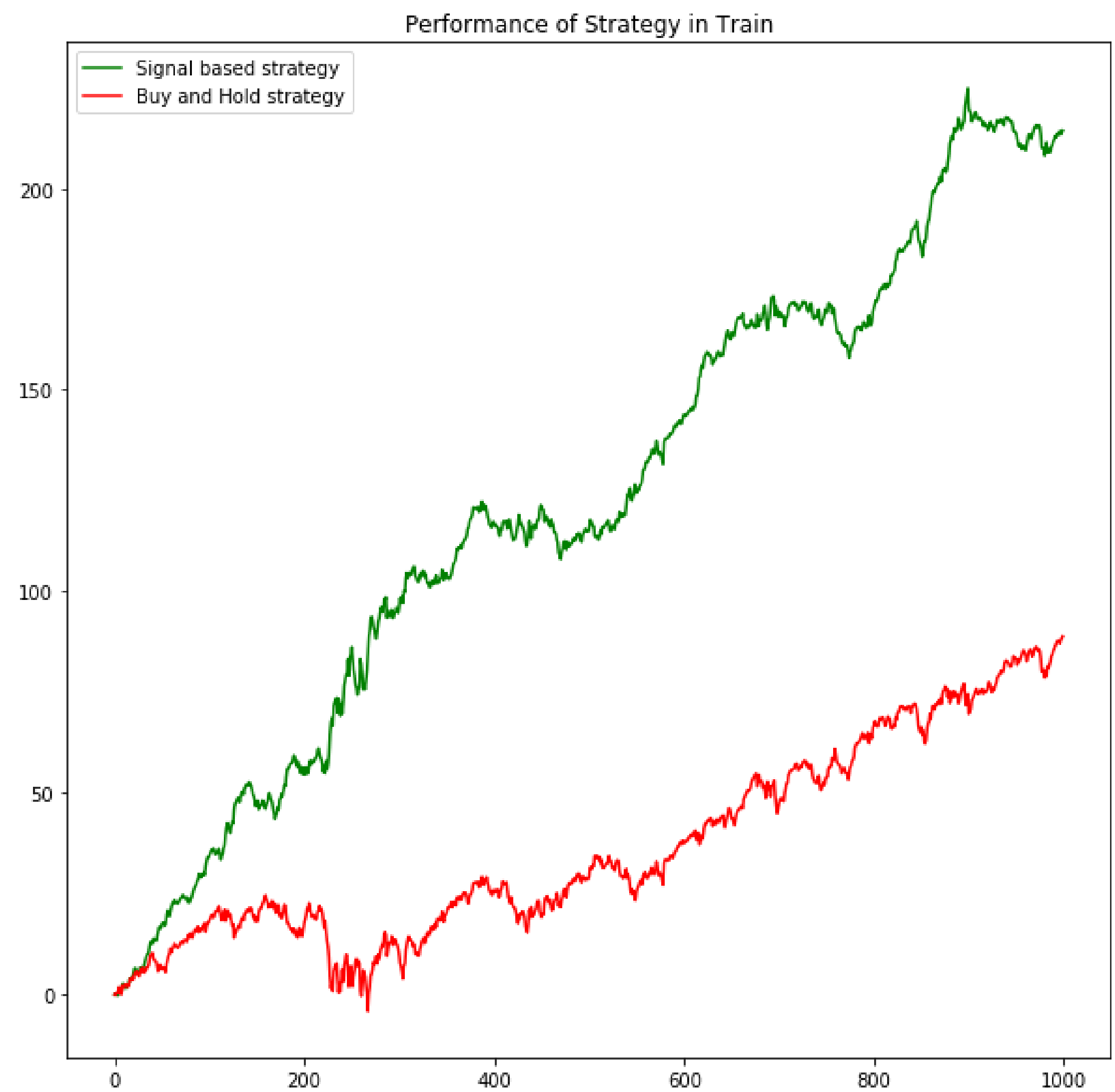
1 Profit of Signal-based strategy

```
In [24]: # Train
         Train['Order'] = [1 if sig>0 else -1 for sig in Train['PredictedY']]
         Train['Profit'] = Train['spy'] * Train['Order']

         Train['Wealth'] = Train['Profit'].cumsum()
         print('Total profit made in Train: ', Train['Profit'].sum())
```

```
Total profit made in Train:  214.340095
```

```
In [25]: plt.figure(figsize=(10, 10))
         plt.title('Performance of Strategy in Train')
         plt.plot(Train['Wealth'].values, color='green', label='Signal based strategy')
         plt.plot(Train['spy'].cumsum().values, color='red', label='Buy and Hold strategy')
         plt.legend()
         plt.show()
```



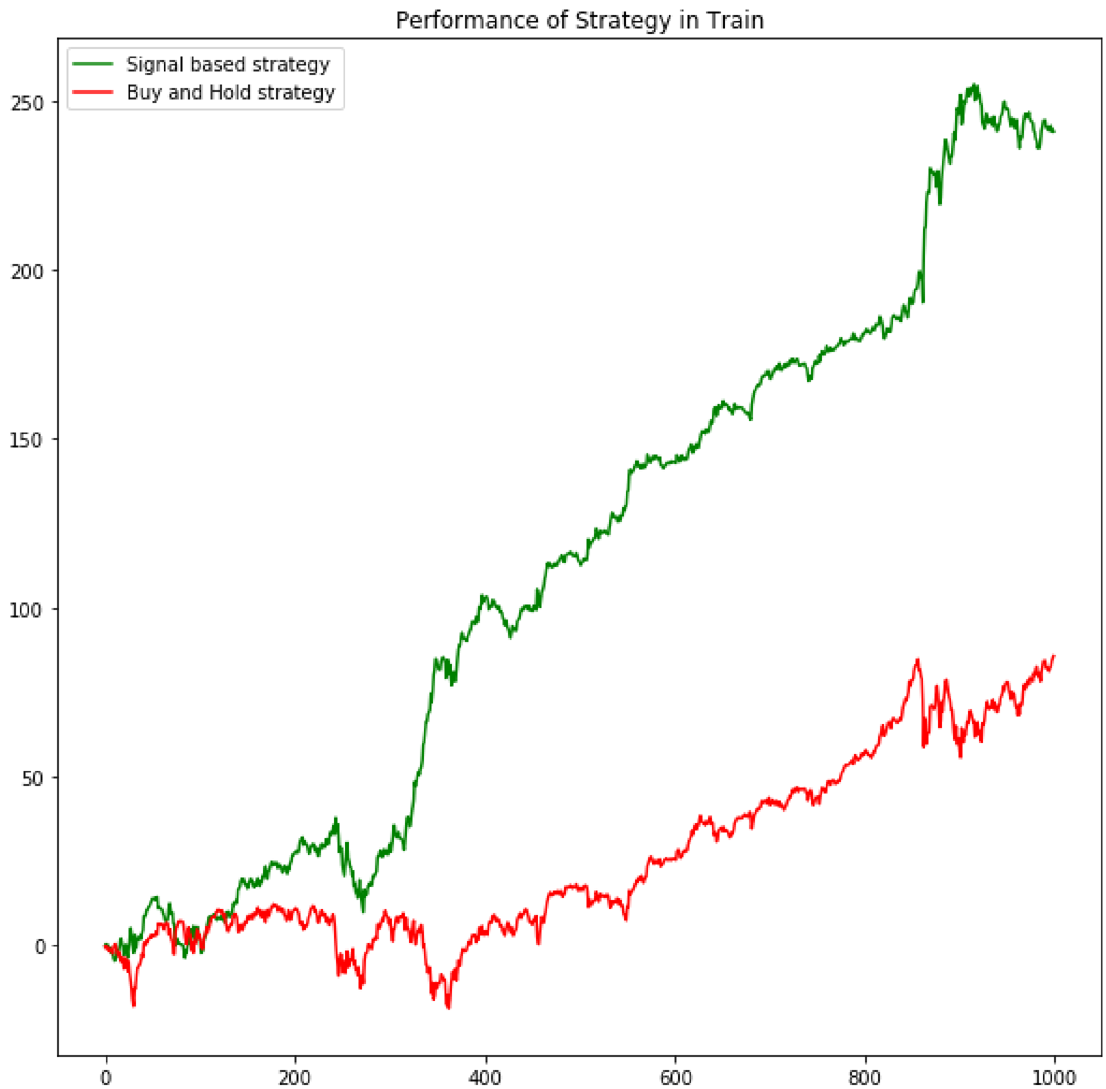

```
In [26]: # Test
         Test['Order'] = [1 if sig>0 else -1 for sig in Test['PredictedY']]
         Test['Profit'] = Test['spy'] * Test['Order']

         Test['Wealth'] = Test['Profit'].cumsum()
         print('Total profit made in Test: ', Test['Profit'].sum())
```

```
Total profit made in Test:  241.030088
```

```
In [27]: plt.figure(figsize=(10, 10))
         plt.title('Performance of Strategy in Train')
         plt.plot(Test['Wealth'].values, color='green', label='Signal based strategy')
         plt.plot(Test['spy'].cumsum().values, color='red', label='Buy and Hold strategy')
```

```
plt.legend()
plt.show()
```



2 Evaluation of model - Practical Standard

Evaluate using two common practical standards - **Sharpe Ratio**, **Maximum Drawdown**, both Train and Test Data

```
In [28]: Train['Wealth'] = Train['Wealth'] + Train.loc[Train.index[0], 'Price']
         Test['Wealth'] = Test['Wealth'] + Test.loc[Test.index[0], 'Price']

         # Sharpe Ratio on Train data
         Train['Return'] = np.log(Train['Wealth']) - np.log(Train['Wealth'].shift(1))
```

```
dailyr = Train['Return'].dropna()

print('Daily Sharpe Ratio is ', dailyr.mean()/dailyr.std(ddof=1))
print('Yearly Sharpe Ratio is ', (252**0.5)*dailyr.mean()/dailyr.std(ddof=1))

# Sharpe Ratio in Test data
Test['Return'] = np.log(Test['Wealth']) - np.log(Test['Wealth'].shift(1))
dailyr = Test['Return'].dropna()

print('Daily Sharpe Ratio is ', dailyr.mean()/dailyr.std(ddof=1))
print('Yearly Sharpe Ratio is ', (252**0.5)*dailyr.mean()/dailyr.std(ddof=1))
```

```
Daily Sharpe Ratio is  0.179650763033
Yearly Sharpe Ratio is  2.85186745096
Daily Sharpe Ratio is  0.130351262086
Yearly Sharpe Ratio is  2.06926213537
```

In [29]:

```
# Maximum Drawdown in Train data
Train['Peak'] = Train['Wealth'].cummax()
Train['Drawdown'] = (Train['Peak'] - Train['Wealth'])/Train['Peak']
print('Maximum Drawdown in Train is ', Train['Drawdown'].max())

# Maximum Drawdown in Test data
Test['Peak'] = Test['Wealth'].cummax()
Test['Drawdown'] = (Test['Peak'] - Test['Wealth'])/Test['Peak']
print('Maximum Drawdown in Test is ', Test['Drawdown'].max())
```

```
Maximum Drawdown in Train is  0.0606901644364
Maximum Drawdown in Test is  0.117198995246
```

In [ ]: