Utilizing Machine Learning in Stock Market Prediction
Master Colloquium, Summer Semester of 2019

Basem Sopeh - Salma Ragheb

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Table of Content

1. Understanding the Stock Market

Markets are like living organisms, constantly evolving. What is working out one day may not necessarily work out the next. For instance, every now and then papers are published to alert the financial world to the emerging of a phenomenon that results in a profitable anomaly. Usually this phenomenon is a subsequent effect of a real-world constraint.

For example, year-end tax loss sales: Tax selling is a type of sale where an investor, for income tax purposes, sells an asset with a capital loss to decrease or eliminate the capital gain realized by other investments. Tax selling allows the investor to avoid paying capital gains tax on recently sold assets.

Because of the tax laws, traders tend to sell their losses at the very end of the year, causing negative pressure on the losing stocks as the year concludes. These stocks are depreciated beyond their fair market value. Consequently, this negative pressure is gone by the beginning of the new year to be replaced by a positive pressure as money is invested towards these depreciated assets. When this phenomenon is made public, traders attempt to get ahead of it by buying stocks in late December and selling them to other traders in January. The losses that took place in December are now diluted in January as new traders walk into the market. The traders relieve the year-end selling pressure and decrease January's buying pressure. However, this scheme does not always work. Traders abandon the strategies that are no longer profitable, searching for new ones that are more effective. For that, traders must be quick at adapting to new strategies.

****Efficient Market Hypothesis***: It was developed by Eugene Fama, stating that: "All markets are rational and all available information is adequately reflected in stock prices." Consistently beating the market at a risk-adjusting basis is almost impossible to any investor. EMH discusses three forms:

- **Weak form:** The market is efficient in a sense that one cannot just use the past information of prices for future prices prediction. The *information is quickly reflected* in stocks. Sometimes, a fundamental or technical analysis could be effective.
- **Semi-Strong form**: The prices simply reflect all the *newly-emerged public information* in an unbiased manner. Hence, no fundamental or technical analysis would be effective.
- **Strong form**: The stock prices reflect all the *public and private information*.

Because of these theories, it is pretty much hopeless to exploit the patterns of the market. Fortunately, despite of the efficiency of the market running sometimes some inefficiencies occur. Some of them are sporadic, but others do persist.

Momentum Strategy: There are several variations of the strategy. But they are all centered around the same concept of that all stocks are ranked from highest to lowest relative to their returns over a given period of time. The top performers are bought and held for a certain time span. This process is repeated several times after some fixed holding periods. It is a pretty simple cycle, yet it has very powerful results. After a lot of research, it is concluded that something is systematically and inherently biased regarding the way humans deal with information. Humans tend to underact to news in the short term, yet overreact in the long term. Stocks' values behave the same way as humans perceive new. The effects are persistent over long periods of time.

2. Developing a Trading Strategy

First, the technical aspects of the strategy are analyzed. S&P 500 is observed over the past several years. S&P 500, also known as S&P, is an American stock market index. It tracks the stocks of 500 large American companies, representing the stock market's performance through reports of the risks and returns of these companies. It is used by the investors as the benchmark of the overall market, to which all other investments are compared. The market capitalization of the companies within its index is tracked using S&P. Market capitalization is the total value of all shares of stock that a company has issued, calculated by multiplying the number of shares issued by the stock price.

An example of a company which has a market cap of $100 billion receives 10 times the representation as a company whose market cap is $10 billion. The total market capitalization of the S&P 500 is $23.5 trillion. It holds 80% of the market capitalization of the stock market. The index is weighted by a float-adjusted market capitalization. It only measures the publicly available shares. Those held by control groups, other companies, or government agencies are not taken into account.

The Pandas functionality in Python is used to import the data. This will provide access to many stock data, including Yahoo and Google. Pandas data reader is installed to access the data. Jupyter notebook is used in implementing and running the program. SPY ETF is imported to gain access to the S&P 500 stock data. The period of observation of the data is from 01.01.2010 till 01.03.2019. As the code runs, the high price, low price, open price, close price, volume, and adjacent close of the stocks are displayed for each and every day in the given timeframe. A table of 1550 rows and 6 columns is displayed (the first 24 days and the last 24 days of the time period are displayed in Figures 2.1 and 2.2, respectively).

Out[4]:

Date	High	Low	Open	Close	Volume	Adj Close
2010-01-04	113.389999	111.510002	112.370003	113.330002	118944600.0	94.130867
2010-01-05	113.680000	112.849998	113.260002	113.629997	111579900.0	94.380074
2010-01-06	113.989998	113.430000	113.519997	113.709999	116074400.0	94.446495
2010-01-07	114.330002	113.180000	113.500000	114.190002	131091100.0	94.845207
2010-01-08	114.620003	113.660004	113.889999	114.570000	126402800.0	95.160805
2010-01-11	115.129997	114.239998	115.080002	114.730003	106375700.0	95.293701
2010-01-12	114.209999	113.220001	113.970001	113.660004	163333500.0	94.404961
2010-01-13	114.940002	113.370003	113.949997	114.620003	161822000.0	95.202347
2010-01-14	115.139999	114.419998	114.489998	114.930000	115718800.0	95.459831
2010-01-15	114.839996	113.199997	114.730003	113.639999	212283100.0	94.388382
2010-01-19	115.129997	113.589996	113.620003	115.059998	139172700.0	95.567795
2010-01-20	114.449997	112.980003	114.279999	113.889999	216490200.0	94.596008
2010-01-21	114.269997	111.559998	113.919998	111.699997	344859600.0	92.777000
2010-01-22	111.739998	109.089996	111.199997	109.209999	345942400.0	90.708870
2010-01-25	110.410004	109.410004	110.209999	109.769997	186937500.0	91.173965
2010-01-26	110.470001	109.040001	109.339996	109.309998	211168800.0	90.791885
2010-01-27	110.080002	108.330002	109.169998	109.830002	271863600.0	91.223801
2010-01-28	110.250000	107.910004	110.190002	108.570000	316104000.0	90.177261

Figure 2.1

2016-02-03	191.779999	187.100006	191.410004	191.300003	205054900.0	179.491806
2016-02-04	192.750000	189.960007	190.710007	191.600006	136318100.0	179.773300
2016-02-05	191.669998	187.199997	190.990005	187.949997	180788300.0	176.348541
2016-02-08	186.119995	182.800003	185.770004	185.419998	191526700.0	173.974731
2016-02-09	186.940002	183.199997	183.360001	185.429993	176478700.0	173.984131
2016-02-10	188.339996	185.119995	186.410004	185.270004	148214100.0	173.834000
2016-02-11	184.100006	181.089996	182.339996	182.860001	219058900.0	171.572784
2016-02-12	186.649994	183.960007	184.960007	186.630005	127632400.0	175.110077
2016-02-16	189.809998	187.630005	188.770004	189.779999	120250700.0	178.065643
2016-02-17	193.320007	191.009995	191.160004	192.880005	136009500.0	180.974274
2016-02-18	193.270004	191.720001	193.199997	192.089996	102343000.0	180.233032
2016-02-19	192.179993	190.449997	191.169998	192.000000	114793000.0	180.148605
2016-02-22	194.949997	193.789993	193.869995	194.779999	103640300.0	182.756973
2016-02-23	194.320007	192.179993	194.000000	192.320007	111455300.0	180.448868
2016-02-24	193.529999	189.320007	190.630005	193.199997	150812200.0	181.274536
2016-02-25	195.550003	192.830002	193.729996	195.539993	107512400.0	183.470047
2016-02-26	196.679993	194.899994	196.570007	195.089996	129833700.0	183.047852
2016-02-29	196.229996	193.330002	195.110001	193.559998	125918100.0	181.612305
2016-03-01	198.210007	194.449997	195.009995	198.110001	141799700.0	185.881454

1550 rows × 6 columns

Figure 2.2

Now, the close prices VS time are displayed in a plot individually. The results are displayed in Figure 2.3. We then start analyzing the fluctuations by studying the open price on the first day of the period which is 112.37point (Figure 2.4) and the close price of the last day of the period which is 198.11 point (Figure 2.5).

Figure 2.3

```
In [7]: first_open = spy['Open'].iloc[0]
        first_open

Out[7]: 112.37000274658203
```

Figure 2.4

```
In [8]: last_close = spy['Close'].iloc[-1]
        last_close

Out[8]: 198.11000061035156
```

Figure 2.5

```
In [9]: last_close - first_open

Out[9]: 85.73999786376953
```

Figure 2.6

After that, the daily change is calculated by subtracting the open price of each day from the close price of the very same day (Figure 2.7). The output is a 1550 x 2 (Figures 2.8 and 2.9). This is the list of daily changes from 04.01.2010 till 01.03.2016.

```
In [10]: spy['Daily Change'] = pd.Series(spy['Close'] - spy['Open'])
```

Figure 7

```
In [11]: spy['Daily Change']

Out[11]: Date
         2010-01-04     0.959999
         2010-01-05     0.369995
         2010-01-06     0.190002
         2010-01-07     0.690002
         2010-01-08     0.680000
         2010-01-11    -0.349998
         2010-01-12    -0.309998
         2010-01-13     0.670006
         2010-01-14     0.440002
         2010-01-15    -1.090004
         2010-01-19     1.439995
         2010-01-20    -0.389999
         2010-01-21    -2.220001
         2010-01-22    -1.989998
         2010-01-25    -0.440002
         2010-01-26    -0.029999
         2010-01-27     0.660004
         2010-01-28    -1.620003
         2010-01-29    -1.650002
         2010-02-01     0.909996
         2010-02-02     1.119995
         2010-02-03    -0.049995
         2010-02-04    -2.540001
         2010-02-05     0.100006
         2010-02-08    -0.849998
         2010-02-09     0.090004
         2010-02-10    -0.040001
         2010-02-11     1.259995
         2010-02-12     1.050003
         2010-02-16     0.879997
                          ...
```

Figure 2.8

```
                ...
2016-01-19   -1.900009
2016-01-20    0.619995
2016-01-21    0.479996
2016-01-22    0.740005
2016-01-25   -2.279999
2016-01-26    1.779999
2016-01-27   -1.449997
2016-01-28   -0.850006
2016-01-29    3.699997
2016-02-01    1.119995
2016-02-02   -1.800003
2016-02-03   -0.110001
2016-02-04    0.889999
2016-02-05   -3.040009
2016-02-08   -0.350006
2016-02-09    2.069992
2016-02-10   -1.139999
2016-02-11    0.520004
2016-02-12    1.669998
2016-02-16    1.009995
2016-02-17    1.720001
2016-02-18   -1.110001
2016-02-19    0.830002
2016-02-22    0.910004
2016-02-23   -1.679993
2016-02-24    2.569992
2016-02-25    1.809998
2016-02-26   -1.480011
2016-02-29   -1.550003
2016-03-01    3.100006
Name: Daily Change, Length: 1550, dtype: float64
```

Figure 2.9

Then, the sum of the changes is calculated which is 30.33 (Figure 2.10).

```
In [12]: spy['Daily Change'].sum()
Out[12]: 30.330169677734375
```

Figure 2.10

In conclusion, more than half of the market gains were achieved by holding onto the stocks overnight throughout the entire period. The overnight returns were better than the inter-day returns. Returns are always evaluated on a risk-adjusted basis. Now, the overnight trades are compared to the inter-day trades on the basis of their standard deviations. It is a quantity that expresses how much the daily changes differ from their mean. Numpy library offers a feature to calculate the standard deviation of the daily changes. The standard deviation is 1.16 (Figure 2.11).

```
In [13]: np.std(spy['Daily Change'])
Out[13]: 1.1673012238307021
```

Figure 2.11

Then, the standard deviation of the overnight change is calculated, which is 0.913 (Figure 2.12).

```
In [14]: spy['Overnight Change'] = pd.Series(spy['Open'] - spy['Close'].shift(1))
         np.std(spy['Overnight Change'])

Out[14]: 0.913886975033839
```

Figure 2.12

In conclusion, the overnight trading has lower volatility than the inter-day trading. Volatility is the measure of dispersion of a given market. Security and volatility are directly proportional. However, not all volatilities are created equal. Now, the mean of the downside days is compared to the mean of upside days for both strategies. First, the upside days are observed, which is the mean of the inter-day change (Figure 2.13). Then, the downside days are observed, which is the mean of the overnight change (Figure 2.14). The means are -0.92 and -0.63, respectively.

```
In [15]: spy[spy['Daily Change']<0]['Daily Change'].mean()

Out[15]: -0.9215800409731658
```

Figure 2.13

```
In [16]: spy[spy['Overnight Change']<0]['Overnight Change'].mean()

Out[16]: -0.6523316308353724
```

Figure 2.14

The average of the downside moves is less than the average of the upside moves. Since everything now is observed in terms of points. The returns are to be noted. This helps putting the losses and gains in a more realistic context. The daily return, inter-day return, and overnight return are observed. Pandas shift method is used to subtract each series from the previous day's series. That is, for the first line of code in Figure 15, close price from one day ago is subtracted from the close price of the current day, for each day. Same goes for the inter-day returns and overnight returns.

```
In [17]: daily_rtn = ((spy['Close'] -spy['Close'].shift(1))/spy['Close'].shift(1))*100
         id_rtn = ((spy['Close'] - spy['Open'])/spy['Open'])*100
         on_rtn = ((spy['Open'] - spy['Close'].shift(1))/spy['Close'].shift(1))*100
```

Figure 2.15

The corresponding tables for the daily returns (Figures 2.16 and 2.17), inter-day returns (Figures 2.18 and 2.19), and overnight returns (Figures 2.20 and 2.21) are displayed.

```
In [18]: daily_rtn

Out[18]: Date
         2010-01-04         NaN
         2010-01-05    0.264710
         2010-01-06    0.070406
         2010-01-07    0.422129
         2010-01-08    0.332776
         2010-01-11    0.139656
         2010-01-12   -0.932624
         2010-01-13    0.844623
         2010-01-14    0.270457
         2010-01-15   -1.122423
         2010-01-19    1.249558
         2010-01-20   -1.016859
         2010-01-21   -1.922910
         2010-01-22   -2.229183
         2010-01-25    0.512771
         2010-01-26   -0.419057
         2010-01-27    0.475715
         2010-01-28   -1.147229
         2010-01-29   -1.086857
         2010-02-01    1.555078
         2010-02-02    1.210343
         2010-02-03   -0.498275
         2010-02-04   -3.086588
         2010-02-05    0.206690
         2010-02-08   -0.721924
         2010-02-09    1.256022
         2010-02-10   -0.195858
         2010-02-11    1.046627
         2010-02-12   -0.083230
         2010-02-16    1.573488
                         ...
```

Figure 2.16

```
                         ...
2016-01-19    0.133113
2016-01-20   -1.281508
2016-01-21    0.560199
2016-01-22    2.051530
2016-01-25   -1.511655
2016-01-26    1.364313
2016-01-27   -1.088324
2016-01-28    0.520914
2016-01-29    2.437735
2016-02-01   -0.036138
2016-02-02   -1.802216
2016-02-03    0.599495
2016-02-04    0.156823
2016-02-05   -1.905015
2016-02-08   -1.346102
2016-02-09    0.005390
2016-02-10   -0.086280
2016-02-11   -1.300806
2016-02-12    2.061689
2016-02-16    1.687828
2016-02-17    1.633474
2016-02-18   -0.409586
2016-02-19   -0.046851
2016-02-22    1.447916
2016-02-23   -1.262959
2016-02-24    0.457565
2016-02-25    1.211178
2016-02-26   -0.230130
2016-02-29   -0.784253
2016-03-01    2.350694
Name: Close, Length: 1550, dtype: float64
```

Figure 2.17

```
In [19]: id_rtn

Out[19]: Date
         2010-01-04     0.854320
         2010-01-05     0.326678
         2010-01-06     0.167374
         2010-01-07     0.607932
         2010-01-08     0.597068
         2010-01-11    -0.304135
         2010-01-12    -0.271999
         2010-01-13     0.587982
         2010-01-14     0.384315
         2010-01-15    -0.950060
         2010-01-19     1.267378
         2010-01-20    -0.341267
         2010-01-21    -1.948737
         2010-01-22    -1.789566
         2010-01-25    -0.399240
         2010-01-26    -0.027436
         2010-01-27     0.604565
         2010-01-28    -1.470190
         2010-01-29    -1.513208
         2010-02-01     0.841420
         2010-02-02     1.025073
         2010-02-03    -0.045500
         2010-02-04    -2.330704
         2010-02-05     0.093850
         2010-02-08    -0.796326
         2010-02-09     0.084014
         2010-02-10    -0.037367
         2010-02-11     1.178997
         2010-02-12     0.981403
         2010-02-16     0.808375
                          ...
```

Figure 2.18

```
                 ...
2016-01-19    -1.000215
2016-01-20     0.335078
2016-01-21     0.257771
2016-01-22     0.389928
2016-01-25    -1.200505
2016-01-26     0.944697
2016-01-27    -0.764847
2016-01-28    -0.447466
2016-01-29     1.947162
2016-02-01     0.581725
2016-02-02    -0.937697
2016-02-03    -0.057469
2016-02-04     0.466677
2016-02-05    -1.591711
2016-02-08    -0.188408
2016-02-09     1.128922
2016-02-10    -0.611555
2016-02-11     0.285184
2016-02-12     0.902897
2016-02-16     0.535040
2016-02-17     0.899770
2016-02-18    -0.574534
2016-02-19     0.434170
2016-02-22     0.469389
2016-02-23    -0.865976
2016-02-24     1.348157
2016-02-25     0.934289
2016-02-26    -0.752918
2016-02-29    -0.794425
2016-03-01     1.589665
Length: 1550, dtype: float64
```

Figure 2.19

```
In [*]: on_rtn

Out[20]: Date
         2010-01-04          NaN
         2010-01-05    -0.061766
         2010-01-06    -0.096806
         2010-01-07    -0.184680
         2010-01-08    -0.262723
         2010-01-11     0.445145
         2010-01-12    -0.662427
         2010-01-13     0.255141
         2010-01-14    -0.113423
         2010-01-15    -0.174016
         2010-01-19    -0.017596
         2010-01-20    -0.677906
         2010-01-21     0.026340
         2010-01-22    -0.447628
         2010-01-25     0.915667
         2010-01-26    -0.391728
         2010-01-27    -0.128076
         2010-01-28     0.327780
```

Figure 2.20

```
2016-02-04    -0.308414
2016-02-05    -0.318372
2016-02-08    -1.159879
2016-02-09    -1.110990
2016-02-10     0.528507
2016-02-11    -1.581480
2016-02-12     1.148423
2016-02-16     1.146653
2016-02-17     0.727160
2016-02-18     0.165902
2016-02-19    -0.478941
2016-02-22     0.973956
2016-02-23    -0.400451
2016-02-24    -0.878745
2016-02-25     0.274326
2016-02-26     0.526754
2016-02-29     0.010254
2016-03-01     0.749120
Length: 1550, dtype: float64
```

Figure 2.21

Now, the statistical values of all of the three strategies are observed. A function that takes in each series to return the desired results is used. The statistics under observation are: trades, wins, losses, break-evens, win-loss ratio, mean win, mean loss, standard deviation, maximum loss, maximum win, and sharp ratio (Figure 2.22). Each strategy is run individually.

```
In [21]: def get_stats(s, n=252):
             s = s.dropna()
             wins = len(s[s>0])
             losses = len(s[s<0])
             evens = len(s[s==0])
             mean_w = round(s[s>0].mean(), 3)
             mean_l = round(s[s<0].mean(), 3)
             win_r = round(wins/losses, 3)
             mean_trd = round(s.mean(), 3)
             sd = round(np.std(s), 3)
             max_l = round(s.min(), 3)
             max_w = round(s.max(), 3)
             sharpe_r = round((s.mean()/np.std(s))*np.sqrt(n), 4)
             cnt = len(s)
             print('Trades:', cnt,\
                 '\nWins:', wins,\
                 '\nLosses:', losses,\
                 '\nBreakeven:', evens,\
                 '\nWin/Loss Ratio', win_r,\
                 '\nMean Win:', mean_w,\
                 '\nMean Loss:', mean_l,\
                 '\nMean', mean_trd,\
                 '\nStd Dev:', sd,\
                 '\nMax Loss:', max_l,\
                 '\nMax Win:', max_w,\
                 '\nSharpe Ratio:', sharpe_r)
```

Figure 2.22

The buy-and-hold strategy has the highest mean return as well as the highest standard deviation among the three. It also has the largest daily draw-down loss. Although the overnight strategy has approximately the same mean as the inter-day strategy, it has much less volatility. In turn, it gives it a higher sharp ratio compared to inter-day strategy. A sharp ratio is a quantity that evaluates the return of an investment given a certain risk or volatility. With that, a solid view-point on how to compare the three strategies is obtained (Figure 2.23)

```
In [22]: get_stats(daily_rtn)

Trades: 1549
Wins: 846
Losses: 697
Breakeven: 6
Win/Loss Ratio 1.214
Mean Win: 0.69
Mean Loss: -0.745
Mean 0.041
Std Dev: 1.009
Max Loss: -6.512
Max Win: 4.65
Sharpe Ratio: 0.6475
```

```
In [23]: get_stats(id_rtn)

Trades: 1550
Wins: 849
Losses: 690
Breakeven: 11
Win/Loss Ratio 1.23
Mean Win: 0.517
Mean Loss: -0.598
Mean 0.017
Std Dev: 0.766
Max Loss: -4.196
Max Win: 3.683
Sharpe Ratio: 0.3547
```

```
In [24]: get_stats(on_rtn)

Trades: 1549
Wins: 823
Losses: 712
Breakeven: 14
Win/Loss Ratio 1.156
Mean Win: 0.419
Mean Loss: -0.432
Mean 0.024
Std Dev: 0.614
Max Loss: -2.936
Max Win: 4.09
Sharpe Ratio: 0.6284
```

Figure 2.23

Then the analysis is furtherly extended by pulling data from 01.01.2000 till 01.03.2016 (Figure 2.24). The corresponding outputs are a chart of 4065 rows and 6 columns (Figures 2.25 and 2.26) and a plot (Figure 2.27).

```
In [25]: start_date = pd.to_datetime('2000-01-01')
         stop_date = pd.to_datetime('2016-03-01')
         sp = pdr.data.get_data_yahoo('SPY', start_date, stop_date)
         sp
```

Figure 2.24

Out[25]:

Date	High	Low	Open	Close	Volume	Adj Close
2000-01-03	148.250000	143.875000	148.250000	145.437500	8164300.0	101.425385
2000-01-04	144.062500	139.640594	143.531204	139.750000	8089800.0	97.459068
2000-01-05	141.531204	137.250000	139.937500	140.000000	12177900.0	97.633377
2000-01-06	141.500000	137.750000	139.625000	137.750000	6227200.0	96.064301
2000-01-07	145.750000	140.062500	140.312500	145.750000	8066500.0	101.643333
2000-01-10	146.906204	145.031204	146.250000	146.250000	5741700.0	101.992004
2000-01-11	146.093704	143.500000	145.812500	144.500000	7503700.0	100.771645
2000-01-12	144.593704	142.875000	144.593704	143.062500	6907700.0	99.769150
2000-01-13	145.750000	143.281204	144.468704	145.000000	5158300.0	101.120308

Figure 2.25

2016-02-17	193.320007	191.009995	191.160004	192.880005	136009500.0	180.974274
2016-02-18	193.270004	191.720001	193.199997	192.089996	102343000.0	180.233032
2016-02-19	192.179993	190.449997	191.169998	192.000000	114793000.0	180.148605
2016-02-22	194.949997	193.789993	193.869995	194.779999	103640300.0	182.756973
2016-02-23	194.320007	192.179993	194.000000	192.320007	111455300.0	180.448868
2016-02-24	193.529999	189.320007	190.630005	193.199997	150812200.0	181.274536
2016-02-25	195.550003	192.830002	193.729996	195.539993	107512400.0	183.470047
2016-02-26	196.679993	194.899994	196.570007	195.089996	129833700.0	183.047852
2016-02-29	196.229996	193.330002	195.110001	193.559998	125918100.0	181.612305
2016-03-01	198.210007	194.449997	195.009995	198.110001	141799700.0	185.881454

4065 rows × 6 columns

Figure 2.26

Figure 2.27

In Figure 2.27, the SPY action in the beginning of year 2000 through the beginning of March 2016 is observed. There have been a lot of fluctuations during this period. The market has experienced both highly positive and highly negative regimes. Now, the statistics of our three main strategies (Daily return, inter-day return, and overnight return) are obtained in the same way they were obtained earlier (Figures 2.28 and 2.29). The differences between the three strategies are more pronounced as the period of time was extended. Should the money had been held overnight, the returns would have improved by more than 50%, assuming no trading costs or taxed filed. This can be confirmed by comparing the overnight mean loss to the sharpe ratio in Figure 2.29.

```
In [31]: get_stats(long_day_rtn)

Trades: 4064
Wins: 2170
Losses: 1879
Breakeven: 15
Win/Loss Ratio 1.155
Mean Win: 0.818
Mean Loss: -0.911
Mean 0.016
Std Dev: 1.275
Max Loss: -9.845
Max Win: 14.52
Sharpe Ratio: 0.1958
```

```
In [32]: get_stats(long_id_rtn)

Trades: 4065
Wins: 2126
Losses: 1909
Breakeven: 30
Win/Loss Ratio 1.114
Mean Win: 0.686
Mean Loss: -0.769
Mean -0.002
Std Dev: 1.054
Max Loss: -8.991
Max Win: 8.435
Sharpe Ratio: -0.0307
```

```
In [33]: get_stats(long_on_rtn)

Trades: 4064
Wins: 2154
Losses: 1870
Breakeven: 40
Win/Loss Ratio 1.152
Mean Win: 0.436
Mean Loss: -0.465
Mean 0.017
Std Dev: 0.691
Max Loss: -8.322
Max Win: 6.068
Sharpe Ratio: 0.3936
```

Figure 2.29

```
In [27]: long_day_rtn = ((sp['Close'] - sp['Close'].shift(1))/sp['Close'].shift(1))*100
         long_id_rtn = ((sp['Close'] - sp['Open'])/sp['Open'])*100
         long_on_rtn = ((sp['Open'] - sp['Close'].shift(1))/sp['Close'].shift(1))*100

In [28]: (sp['Close'] - sp['Close'].shift(1)).sum()

Out[28]: 52.67250061035156

In [29]: (sp['Close'] - sp['Open']).sum()

Out[29]: -49.94224548339844

In [30]: (sp['Open'] - sp['Close'].shift(1)).sum()

Out[30]: 99.80224609375
```

Figure 2.28

3. Building a Model and Evaluating its Performance

Now, a regression model will be built to evaluate its performance. Linear Regression is a technique commonly used for statistical.

Regression is a method of modelling a target value relative to independent predictors, by forecasting and finding out cause-and-effect relationship between variables. Regression techniques differences depend on the number of independent variables and the kind of relationship between them and the dependent variables.

The stock's close price of the prior day will be used to predict the close price of the next day. The model is built using a regression vector. The first step is setting up a data frame object containing the price history for each day. The preceding 20 closes of each day are included in the model. The code in Figure 3.1 is used to get the data. This code's output is the close price of each day with the prior 20 all along the same axis. A portion of the output is displayed in Figures 3.2 and 3.3. The entire output is a 4045 x 21 table that shows the 20 closes of each day from 01.01.2000 till 01.03.2016. This will provide the x-axis for the regression model.

```python
In [34]: for i in range(1, 21, 1):
             sp.loc[:,'Close Minus ' + str(i)] = sp['Close'].shift(i)
             sp20 = sp[[x for x in sp.columns if 'Close Minus' in x or x =='Close']].iloc[20:,]
         sp20
```

Figure 3.1

Out[34]:

Date	Close	Close Minus 1	Close Minus 2	Close Minus 3	Close Minus 4	Close Minus 5	Close Minus 6	Close Minus 7	Close Minus 8	Close Minus 9	...
2000-02-01	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704	144.437500	144.750000	147.000000	...
2000-02-02	141.062500	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704	144.437500	144.750000	...
2000-02-03	143.187500	141.062500	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704	144.437500	...
2000-02-04	142.593704	143.187500	141.062500	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704	...
2000-02-07	142.375000	142.593704	143.187500	141.062500	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	...
2000-02-08	144.312500	142.375000	142.593704	143.187500	141.062500	140.937500	139.562500	135.875000	140.250000	140.812500	...
2000-02-09	141.281204	144.312500	142.375000	142.593704	143.187500	141.062500	140.937500	139.562500	135.875000	140.250000	...
2000-02-10	141.562500	141.281204	144.312500	142.375000	142.593704	143.187500	141.062500	140.937500	139.562500	135.875000	...
2000-02-11	138.687500	141.562500	141.281204	144.312500	142.375000	142.593704	143.187500	141.062500	140.937500	139.562500	...
2000-02-14	139.500000	138.687500	141.562500	141.281204	144.312500	142.375000	142.593704	143.187500	141.062500	140.937500	...
2000-02-15	141.078094	139.500000	138.687500	141.562500	141.281204	144.312500	142.375000	142.593704	143.187500	141.062500	...
2000-02-16	139.000000	141.078094	139.500000	138.687500	141.562500	141.281204	144.312500	142.375000	142.593704	143.187500	...
2000-02-17	138.281204	139.000000	141.078094	139.500000	138.687500	141.562500	141.281204	144.312500	142.375000	142.593704	...
2000-02-18	135.312500	138.281204	139.000000	141.078094	139.500000	138.687500	141.562500	141.281204	144.312500	142.375000	...
2000-02-22	134.968704	135.312500	138.281204	139.000000	141.078094	139.500000	138.687500	141.562500	141.281204	144.312500	...

Figure 3.2

...	Close Minus 11	Close Minus 12	Close Minus 13	Close Minus 14	Close Minus 15	Close Minus 16	Close Minus 17	Close Minus 18	Close Minus 19	Close Minus 20
...	146.968704	145.000000	143.062500	144.500000	146.250000	145.750000	137.750000	140.000000	139.750000	145.437500
...	145.812500	146.968704	145.000000	143.062500	144.500000	146.250000	145.750000	137.750000	140.000000	139.750000
...	147.000000	145.812500	146.968704	145.000000	143.062500	144.500000	146.250000	145.750000	137.750000	140.000000
...	144.750000	147.000000	145.812500	146.968704	145.000000	143.062500	144.500000	146.250000	145.750000	137.750000
...	144.437500	144.750000	147.000000	145.812500	146.968704	145.000000	143.062500	144.500000	146.250000	145.750000
...	140.343704	144.437500	144.750000	147.000000	145.812500	146.968704	145.000000	143.062500	144.500000	146.250000
...	141.937500	140.343704	144.437500	144.750000	147.000000	145.812500	146.968704	145.000000	143.062500	144.500000
...	140.812500	141.937500	140.343704	144.437500	144.750000	147.000000	145.812500	146.968704	145.000000	143.062500
...	140.250000	140.812500	141.937500	140.343704	144.437500	144.750000	147.000000	145.812500	146.968704	145.000000
...	135.875000	140.250000	140.812500	141.937500	140.343704	144.437500	144.750000	147.000000	145.812500	146.968704
...	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704	144.437500	144.750000	147.000000	145.812500
...	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704	144.437500	144.750000	147.000000
...	141.062500	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704	144.437500	144.750000
...	143.187500	141.062500	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704	144.437500
...	142.593704	143.187500	141.062500	140.937500	139.562500	135.875000	140.250000	140.812500	141.937500	140.343704

Figure 3.3

After that, the columns of the table are reversed to get the axis from left to right. The code to reversing the table is in Figure 3.4. A portion of the output table is displayed in Figures 3.5 and 3.6.

```
In [35]: sp20 = sp20.iloc[:,::-1]
         sp20
```

Figure 3.4

Out[35]:

Date	Close Minus 20	Close Minus 19	Close Minus 18	Close Minus 17	Close Minus 16	Close Minus 15	Close Minus 14	Close Minus 13	Close Minus 12	Close Minus 11	...
2000-02-01	145.437500	139.750000	140.000000	137.750000	145.750000	146.250000	144.500000	143.062500	145.000000	146.968704	...
2000-02-02	139.750000	140.000000	137.750000	145.750000	146.250000	144.500000	143.062500	145.000000	146.968704	145.812500	...
2000-02-03	140.000000	137.750000	145.750000	146.250000	144.500000	143.062500	145.000000	146.968704	145.812500	147.000000	...
2000-02-04	137.750000	145.750000	146.250000	144.500000	143.062500	145.000000	146.968704	145.812500	147.000000	144.750000	...
2000-02-07	145.750000	146.250000	144.500000	143.062500	145.000000	146.968704	145.812500	147.000000	144.750000	144.437500	...
2000-02-08	146.250000	144.500000	143.062500	145.000000	146.968704	145.812500	147.000000	144.750000	144.437500	140.343704	...
2000-02-09	144.500000	143.062500	145.000000	146.968704	145.812500	147.000000	144.750000	144.437500	140.343704	141.937500	...
2000-02-10	143.062500	145.000000	146.968704	145.812500	147.000000	144.750000	144.437500	140.343704	141.937500	140.812500	...
2000-02-11	145.000000	146.968704	145.812500	147.000000	144.750000	144.437500	140.343704	141.937500	140.812500	140.250000	...
2000-02-14	146.968704	145.812500	147.000000	144.750000	144.437500	140.343704	141.937500	140.812500	140.250000	135.875000	...
2000-02-15	145.812500	147.000000	144.750000	144.437500	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	...
2000-02-16	147.000000	144.750000	144.437500	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500	...
2000-02-17	144.750000	144.437500	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500	141.062500	...
2000-02-18	144.437500	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500	141.062500	143.187500	...
2000-02-22	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500	141.062500	143.187500	142.593704	...

Figure 3.5

...	Close Minus 9	Close Minus 8	Close Minus 7	Close Minus 6	Close Minus 5	Close Minus 4	Close Minus 3	Close Minus 2	Close Minus 1	Close
...	147.000000	144.750000	144.437500	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500
...	144.750000	144.437500	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500	141.062500
...	144.437500	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500	141.062500	143.187500
...	140.343704	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500	141.062500	143.187500	142.593704
...	141.937500	140.812500	140.250000	135.875000	139.562500	140.937500	141.062500	143.187500	142.593704	142.375000
...	140.812500	140.250000	135.875000	139.562500	140.937500	141.062500	143.187500	142.593704	142.375000	144.312500
...	140.250000	135.875000	139.562500	140.937500	141.062500	143.187500	142.593704	142.375000	144.312500	141.281204
...	135.875000	139.562500	140.937500	141.062500	143.187500	142.593704	142.375000	144.312500	141.281204	141.562500
...	139.562500	140.937500	141.062500	143.187500	142.593704	142.375000	144.312500	141.281204	141.562500	138.687500
...	140.937500	141.062500	143.187500	142.593704	142.375000	144.312500	141.281204	141.562500	138.687500	139.500000
...	141.062500	143.187500	142.593704	142.375000	144.312500	141.281204	141.562500	138.687500	139.500000	141.078094
...	143.187500	142.593704	142.375000	144.312500	141.281204	141.562500	138.687500	139.500000	141.078094	139.000000
...	142.593704	142.375000	144.312500	141.281204	141.562500	138.687500	139.500000	141.078094	139.000000	138.281204
...	142.375000	144.312500	141.281204	141.562500	138.687500	139.500000	141.078094	139.000000	138.281204	135.312500
...	144.312500	141.281204	141.562500	138.687500	139.500000	141.078094	139.000000	138.281204	135.312500	134.968704

Figure 3.6

Now, the support vector machine is imported. The training, matrices, and target vectors are set. There are around 4000 data points to work with. The last 2000 are chosen for testing. The model is fit and used to test the sample data as in Figures 3.7 and 3.8.

```
In [36]: from sklearn.svm import SVR
         clf = SVR(kernel='linear')
         X_train = sp20[:-1000]
         y_train = sp20['Close'].shift(-1)[:-1000]
         X_test = sp20[-1000:]
         y_test = sp20['Close'].shift(-1)[-1000:]
```

Figure 3.7

```
In [37]: model = clf.fit(X_train, y_train)
         preds = model.predict(X_test)
```

Figure 3.8

The output is a 1000 x 2 table showing the predicted close prices of the following day VS the actual ones (Figure 3.9). It can be inferred that most of the predictions were close to the actual prices.

Out[38]:

Date	Next Day Close	Predicted Next Close
2012-03-09	137.580002	137.603674
2012-03-12	140.059998	137.885118
2012-03-13	139.910004	139.951665
2012-03-14	140.720001	139.869231
2012-03-15	140.300003	140.642184
2012-03-16	140.850006	140.393783
2012-03-19	140.440002	140.722827
2012-03-20	140.210007	140.315668
2012-03-21	139.199997	140.052685
2012-03-22	139.649994	139.285878
2012-03-23	141.610001	139.812115
2012-03-26	141.169998	141.497342
2012-03-27	140.470001	141.142911
2012-03-28	140.229996	140.788757
2012-03-29	140.809998	140.628573
2012-03-30	141.839996	140.806655
2016-02-05	185.419998	187.905387
2016-02-08	185.429993	186.297637
2016-02-09	185.270004	185.368468
2016-02-10	182.860001	185.302749
2016-02-11	186.630005	183.284546
2016-02-12	189.779999	186.989269
2016-02-16	192.880005	189.949002
2016-02-17	192.089996	192.663220
2016-02-18	192.000000	192.173020
2016-02-19	194.779999	191.877234
2016-02-22	192.320007	194.727840
2016-02-23	193.199997	191.988886
2016-02-24	195.539993	192.493202
2016-02-25	195.089996	195.026926
2016-02-26	193.559998	194.976676
2016-02-29	198.110001	193.402064
2016-03-01	NaN	197.737472

1000 rows × 2 columns

Figure 3.9

Now, the performance of the model will be observed. The scenario presented is that the next day's open is bought if the close's prediction is higher than the open. Then, the stocks will be sold at the close of the very same day. Some extra data points will be added to the object, to calculate the results. The code is displayed in Figure 3.10. The corresponding output which is a 1000 x 4 table is displayed in Figures 3.11 and 3.12.

```
In [39]: cdc = sp[['Close']].iloc[-1000:]
         ndo = sp[['Open']].iloc[-1000:].shift(-1)
         tf1 = pd.merge(tf, cdc, left_index=True, right_index=True)
         tf2 = pd.merge(tf1, ndo, left_index=True, right_index=True)
         tf2.columns = ['Next Day Close', 'Predicted Next Close', 'Current Day Close', 'Next Day Open']
         tf2
```

Figure 3.10

Out[39]:

Date	Next Day Close	Predicted Next Close	Current Day Close	Next Day Open
2012-03-09	137.580002	137.603674	137.570007	137.550003
2012-03-12	140.059998	137.885118	137.580002	138.320007
2012-03-13	139.910004	139.951665	140.059998	140.100006
2012-03-14	140.720001	139.869231	139.910004	140.119995
2012-03-15	140.300003	140.642184	140.720001	140.360001
2012-03-16	140.850006	140.393783	140.300003	140.210007
2012-03-19	140.440002	140.722827	140.850006	140.050003
2012-03-20	140.210007	140.315668	140.440002	140.520004
2012-03-21	139.199997	140.052685	140.210007	139.179993
2012-03-22	139.649994	139.285878	139.199997	139.320007
2012-03-23	141.610001	139.812115	139.649994	140.649994
2012-03-26	141.169998	141.497342	141.610001	141.740005
2012-03-27	140.470001	141.142911	141.169998	141.100006
2012-03-28	140.229996	140.788757	140.470001	139.639999
2012-03-29	140.809998	140.628573	140.229996	140.919998
2012-03-30	141.839996	140.806655	140.809998	140.639999

Figure 3.11

2016-02-03	191.600006	191.210762	191.300003	190.710007
2016-02-04	187.949997	191.259473	191.600006	190.990005
2016-02-05	185.419998	187.905387	187.949997	185.770004
2016-02-08	185.429993	186.297637	185.419998	183.360001
2016-02-09	185.270004	185.368468	185.429993	186.410004
2016-02-10	182.860001	185.302749	185.270004	182.339996
2016-02-11	186.630005	183.284546	182.860001	184.960007
2016-02-12	189.779999	186.989269	186.630005	188.770004
2016-02-16	192.880005	189.949002	189.779999	191.160004
2016-02-17	192.089996	192.663220	192.880005	193.199997
2016-02-18	192.000000	192.173020	192.089996	191.169998
2016-02-19	194.779999	191.877234	192.000000	193.869995
2016-02-22	192.320007	194.727840	194.779999	194.000000
2016-02-23	193.199997	191.988886	192.320007	190.630005
2016-02-24	195.539993	192.493202	193.199997	193.729996
2016-02-25	195.089996	195.026926	195.539993	196.570007
2016-02-26	193.559998	194.976676	195.089996	195.110001
2016-02-29	198.110001	193.402064	193.559998	195.009995
2016-03-01	NaN	197.737472	198.110001	NaN

Figure 3.12

The columns in Figures 3.11 and 3.12 contain the next day close, predicted day close, current day close, and next day open. Now, a signal is created that shows 1 if predicted next day's close is greater than next day's open, otherwise it shows 0. The code for the signal's function is in Figure 3.13.

```
In [40]: def get_signal(r):
             if r['Predicted Next Close'] > r['Next Day Open']:
                 return 1
             else:
                 return 0
```

Figure 3.13

Now, the return is calculated by subtracting the next day open price from next day close price and dividing the difference by the next day open price. The code is in Figure 3.14. Then, the signals and returns are added to the previous table. The new 1000 x 6 table is displayed in Figures 3.15 and 3.16.

```
In [41]: def get_ret(r):
             if r['Signal'] == 1:
                 return ((r['Next Day Close'] - r['Next Day Open'])/r['Next Day Open']) * 100
             else:
                 return 0
```

Figure 3.14

Out[42]:

Date	Next Day Close	Predicted Next Close	Current Day Close	Next Day Open	Signal	PnL
2012-03-09	137.580002	137.603674	137.570007	137.550003	1	0.021809
2012-03-12	140.059998	137.885118	137.580002	138.320007	0	0.000000
2012-03-13	139.910004	139.951665	140.059998	140.100006	0	0.000000
2012-03-14	140.720001	139.869231	139.910004	140.119995	0	0.000000
2012-03-15	140.300003	140.642184	140.720001	140.360001	1	-0.042745
2012-03-16	140.850006	140.393783	140.300003	140.210007	1	0.456458
2012-03-19	140.440002	140.722827	140.850006	140.050003	1	0.278472
2012-03-20	140.210007	140.315668	140.440002	140.520004	0	0.000000
2012-03-21	139.199997	140.052685	140.210007	139.179993	1	0.014373
2012-03-22	139.649994	139.285878	139.199997	139.320007	0	0.000000
2012-03-23	141.610001	139.812115	139.649994	140.649994	0	0.000000
2012-03-26	141.169998	141.497342	141.610001	141.740005	0	0.000000
2012-03-27	140.470001	141.142911	141.169998	141.100006	1	-0.446495
2012-03-28	140.229996	140.788757	140.470001	139.639999	1	0.422512
2012-03-29	140.809998	140.628573	140.229996	140.919998	0	0.000000
2012-03-30	141.839996	140.806655	140.809998	140.639999	1	0.853240

Figure 3.15

2016-02-05	185.419998	187.905387	187.949997	185.770004	1	-0.188408
2016-02-08	185.429993	186.297637	185.419998	183.360001	1	1.128922
2016-02-09	185.270004	185.368468	185.429993	186.410004	0	0.000000
2016-02-10	182.860001	185.302749	185.270004	182.339996	1	0.285184
2016-02-11	186.630005	183.284546	182.860001	184.960007	0	0.000000
2016-02-12	189.779999	186.989269	186.630005	188.770004	0	0.000000
2016-02-16	192.880005	189.949002	189.779999	191.160004	0	0.000000
2016-02-17	192.089996	192.663220	192.880005	193.199997	0	0.000000
2016-02-18	192.000000	192.173020	192.089996	191.169998	1	0.434170
2016-02-19	194.779999	191.877234	192.000000	193.869995	0	0.000000
2016-02-22	192.320007	194.727840	194.779999	194.000000	1	-0.865976
2016-02-23	193.199997	191.988886	192.320007	190.630005	1	1.348157
2016-02-24	195.539993	192.493202	193.199997	193.729996	0	0.000000
2016-02-25	195.089996	195.026926	195.539993	196.570007	0	0.000000
2016-02-26	193.559998	194.976676	195.089996	195.110001	0	0.000000
2016-02-29	198.110001	193.402064	193.559998	195.009995	0	0.000000
2016-03-01	NaN	197.737472	198.110001	NaN	0	0.000000

1000 rows × 6 columns

Figure 3.16

Now, the price prediction strategy using only the price history is verified by calculating the points gained and the gain using the inter-day strategy during a 1000-

day interval. The codes and corresponding outputs for both strategies are displayed in Figure 3.17. It can be inferred that this strategy is not working well as the gain of the first strategy is completely different from the second one. To confirm the difference, the statistics of both strategies are obtained (Figures 3.18 and 3.19, respectively).

```
In [43]: (tf2[tf2['Signal']==1]['Next Day Close'] - tf2[tf2['Signal']==1]['Next Day Open']).sum()

Out[43]: -5.029960632324219

In [44]: (sp['Close'].iloc[-1000:] - sp['Open'].iloc[-1000:]).sum()

Out[44]: 17.53020477294922
```

Figure 3.17

```
In [45]: get_stats((sp['Close'].iloc[-1000:] - sp['Open'].iloc[-1000:])/sp['Open'].iloc[-1000:] * 100)

         Trades: 1000
         Wins: 544
         Losses: 449
         Breakeven: 7
         Win/Loss Ratio 1.212
         Mean Win: 0.459
         Mean Loss: -0.524
         Mean 0.014
         Std Dev: 0.671
         Max Loss: -4.196
         Max Win: 2.756
         Sharpe Ratio: 0.3354
```

Figure 3.18

```
In [46]: get_stats(tf2['PnL'])

         Trades: 1000
         Wins: 253
         Losses: 222
         Breakeven: 525
         Win/Loss Ratio 1.14
         Mean Win: 0.468
         Mean Loss: -0.538
         Mean -0.001
         Std Dev: 0.469
         Max Loss: -4.088
         Max Win: 2.756
         Sharpe Ratio: -0.0325
```

Figure 3.19

Since the resulting statistics fully prove that this strategy is failing, a modification can be done in an attempt to improve the results. Trades that were expected to be greater by a point or more instead of being an “amount” greater than the open price. The signal code will be re-executed with some modifications (Figure 3.20). Yet the return’s code will remain unchanged. The output is displayed in Figures 3.21 and 3.22. The sum and statistics of the modified strategy are displayed in Figure 3.23.

```
In [47]: def get_signal(r):
             if r['Predicted Next Close'] > r['Next Day Open'] + 1:
                 return 1
             else:
                 return 0

         def get_ret(r):
             if r['Signal'] == 1:
                 return ((r['Next Day Close'] - r['Next Day Open'])/r['Next Day Open']) * 100
             else:
                 return 0
```

Figure 3.20

Out[47]:

Date	Next Day Close	Predicted Next Close	Current Day Close	Next Day Open	Signal	PnL
2012-03-09	137.580002	137.603674	137.570007	137.550003	0	0.000000
2012-03-12	140.059998	137.885118	137.580002	138.320007	0	0.000000
2012-03-13	139.910004	139.951665	140.059998	140.100006	0	0.000000
2012-03-14	140.720001	139.869231	139.910004	140.119995	0	0.000000
2012-03-15	140.300003	140.642184	140.720001	140.360001	0	0.000000
2012-03-16	140.850006	140.393783	140.300003	140.210007	0	0.000000
2012-03-19	140.440002	140.722827	140.850006	140.050003	0	0.000000
2012-03-20	140.210007	140.315668	140.440002	140.520004	0	0.000000
2012-03-21	139.199997	140.052685	140.210007	139.179993	0	0.000000
2012-03-22	139.649994	139.285878	139.199997	139.320007	0	0.000000
2012-03-23	141.610001	139.812115	139.649994	140.649994	0	0.000000
2012-03-26	141.169998	141.497342	141.610001	141.740005	0	0.000000
2012-03-27	140.470001	141.142911	141.169998	141.100006	0	0.000000
2012-03-28	140.229996	140.788757	140.470001	139.639999	1	0.422512
2012-03-29	140.809998	140.628573	140.229996	140.919998	0	0.000000
2012-03-30	141.839996	140.806655	140.809998	140.639999	0	0.000000

Figure 3.21

2016-02-05	185.419998	187.905387	187.949997	185.770004	1	-0.188408
2016-02-08	185.429993	186.297637	185.419998	183.360001	1	1.128922
2016-02-09	185.270004	185.368468	185.429993	186.410004	0	0.000000
2016-02-10	182.860001	185.302749	185.270004	182.339996	1	0.285184
2016-02-11	186.630005	183.284546	182.860001	184.960007	0	0.000000
2016-02-12	189.779999	186.989269	186.630005	188.770004	0	0.000000
2016-02-16	192.880005	189.949002	189.779999	191.160004	0	0.000000
2016-02-17	192.089996	192.663220	192.880005	193.199997	0	0.000000
2016-02-18	192.000000	192.173020	192.089996	191.169998	1	0.434170
2016-02-19	194.779999	191.877234	192.000000	193.869995	0	0.000000
2016-02-22	192.320007	194.727840	194.779999	194.000000	0	0.000000
2016-02-23	193.199997	191.988886	192.320007	190.630005	1	1.348157
2016-02-24	195.539993	192.493202	193.199997	193.729996	0	0.000000
2016-02-25	195.089996	195.026926	195.539993	196.570007	0	0.000000
2016-02-26	193.559998	194.976676	195.089996	195.110001	0	0.000000
2016-02-29	198.110001	193.402064	193.559998	195.009995	0	0.000000
2016-03-01	NaN	197.737472	198.110001	NaN	0	0.000000

1000 rows × 6 columns

Figure 3.22

```
In [48]: (tf2[tf2['Signal']==1]['Next Day Close'] - tf2[tf2['Signal']==1]['Next Day Open']).sum()
Out[48]: -8.450103759765625

In [49]: get_stats(tf2['PnL'])

         Trades: 1000
         Wins: 48
         Losses: 46
         Breakeven: 906
         Win/Loss Ratio 1.043
         Mean Win: 0.581
         Mean Loss: -0.703
         Mean -0.004
         Std Dev: 0.249
         Max Loss: -2.032
         Max Win: 2.756
         Sharpe Ratio: -0.2828
```

Figure 3.23

As per the previous results, it can be inferred that: when the model predicts strong next day's gains, the market significantly underperforms for at least the given test period. It is unlikely that this response will hold true in all scenarios. Markets tend to turn over from mean reversion schemes to schemes of trend persistence. The model is run over a different period for further testing. The data set is now 2000 data samples and the range of observation is 1000 samples as shown in Figure 3.24. The new model returned an output of around 28 points. The output is compared to that of the inter-day strategy (Figure 3.25). It can be inferred that the new model outperformed the old model. However, more modifications should be done.

```
In [50]: X_train = sp20[:-2000]
         y_train = sp20['Close'].shift(-1)[:-2000]
         X_test = sp20[-2000:-1000]
         y_test = sp20['Close'].shift(-1)[-2000:-1000]

         model = clf.fit(X_train, y_train)
         preds = model.predict(X_test)

         tf = pd.DataFrame(list(zip(y_test, preds)), columns=['Next Day Close','Predicted Next Close'], index=y_test.index)
         cdc = sp[['Close']].iloc[-2000:-1000]
         ndo = sp[['Open']].iloc[-2000:-1000].shift(-1)
         tf1 = pd.merge(tf, cdc, left_index=True, right_index=True)
         tf2 = pd.merge(tf1, ndo, left_index=True, right_index=True)
         tf2.columns = ['Next Day Close', 'Predicted Next Close', 'Current Day Close', 'Next Day Open']

         def get_signal(r):
             if r['Predicted Next Close'] > r['Next Day Open'] + 1:
                 return 0
             else:
                 return 1
         def get_ret(r):
             if r['Signal'] == 1:
                 return ((r['Next Day Close'] - r['Next Day Open'])/r['Next Day Open']) * 100
             else:
                 return 0

         tf2 = tf2.assign(Signal = tf2.apply(get_signal, axis=1))
         tf2 = tf2.assign(PnL = tf2.apply(get_ret, axis=1))

         (tf2[tf2['Signal']==1]['Next Day Close'] - tf2[tf2['Signal']==1]['Next Day Open']).sum()
```

Figure 3.24

```
         (tf2[tf2['Signal']==1]['Next Day Close'] - tf2[tf2['Signal']==1]['Next Day Open']).sum()

Out[50]: 28.810020446777344

In [51]: (sp['Close'].iloc[-2000:-1000] - sp['Open'].iloc[-2000:-1000]).sum()

Out[51]: -7.090003967285156
```

Figure 3.25

4. *Modeling with Dynamic Time Warping (DTW)*

Comparing the statistical results in the previous section, it can be inferred that the regression model failed to match the inter-day return strategy. Therefore we are going to develop a new model using DTW algorithm trying to get better result.

Dynamic time warping (DTW) is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. For instance, as it can be seen in the figure 4.1, similarities in walking could be detected using DTW, even if one person was walking faster than the other, or if there were accelerations and decelerations during the course of an observation. DTW has been applied to temporal sequences of video, audio, and graphics data. A well-known application has been automatic speech recognition, to cope with different speaking speeds. Other applications include speaker recognition and online signature recognition. It can also be used in partial shape matching application.

In general, DTW is a method that calculates an optimal match between two given time series.

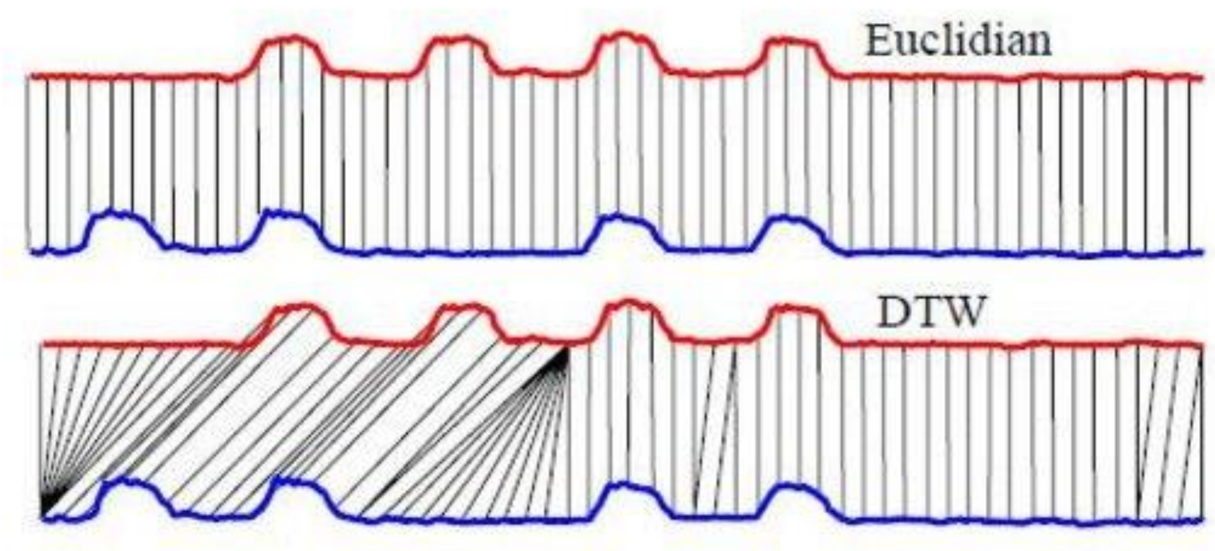


Figure 4.1

The following steps are followed in order to apply DTW on our data:

1- The extended data of close price are split into 5-day periods which is illustrated in the figure 4.2.

```
In [62]:  1  sp['Close'].iloc[0:5]

Out[62]: Date
         2000-01-03    145.4375
         2000-01-04    139.7500
         2000-01-05    140.0000
         2000-01-06    137.7500
         2000-01-07    145.7500
         Name: Close, dtype: float64
```

Figure 4.2

2- The percentage of change (Return) of the first 4-day of each period are calculated and used as time series (T.series), then each T.series is paired to the fifth day return (which is considered as label), that means the patterns of time series are compared to find out if the label is profitable or not. This is shown in the figure 4.3

```
In [63]:  1 tseries[0]

Out[63]: (array([-3.91061453,  0.17889088, -1.60714286,  5.8076225 ]),
          0.34305317324185847)
```

Figure 4.3

3- The distance between each T.series against all others are calculated using DTW algorithm, the figure 4.4 shows the distance between every two time series and the return of each of them.

```
In [73]:  1 sfe[805:815]

Out[73]:
```

	A	B	Dist	A Ret	B Ret
805	0	806	16.090497	0.343053	0.133113
806	0	807	12.297819	0.343053	1.364313
807	0	808	12.837923	0.343053	-1.802216
808	0	809	11.983774	0.343053	0.005390
809	0	810	12.184468	0.343053	1.633474
810	0	811	12.410562	0.343053	0.457565
812	1	2	8.309008	-0.786701	1.135638
813	1	3	5.839694	-0.786701	0.985222
814	1	4	5.801000	-0.786701	1.360843
815	1	5	4.674267	-0.786701	1.131251

Figure 4.4

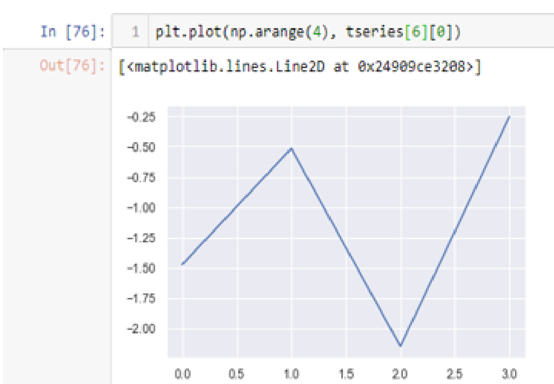
4- Some arrangement are done and shown in the figure 4.5 by removing the rows of zero distance (the same T.series), keeping only the optimal distance (less than one), and keeping only the profitable return of the series A (positive return), a sample are shown in the figure 4.5

Out[57]:

	A	B	Dist	A Ret	B Ret
3312	4	69	0.778629	1.360843	-1.696072
3439	4	196	0.608376	1.360843	0.410596
3609	4	366	0.973192	1.360843	0.040522
3790	4	547	0.832545	1.360843	-1.447712
3891	4	648	0.548912	1.360843	-0.510458
4035	4	792	0.740197	1.360843	0.819056
5463	6	598	0.678315	1.180863	2.896685
5489	6	624	0.897109	1.180863	0.757222
7769	9	471	0.932647	2.333028	-0.212983
13002	16	27	0.849448	0.754885	-0.571339
14269	17	483	0.841603	2.285035	0.793407
16369	20	150	0.164886	1.741904	-0.247414

Figure 4.5

5- To analyze the model, random T.series with positive return B is chosen and compared to T.series A by plotting them and comparing their curves, and that result in in identical curves as it is shown in then figure 4.6-a



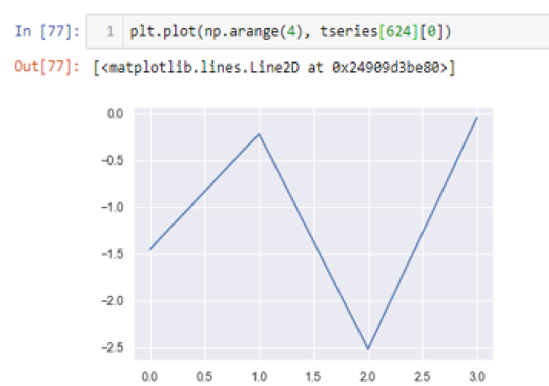


figure 4.6-a

On the other hand, if B T.series with negative return is chosen the curves will not be identical as the figure 4.6-b shows

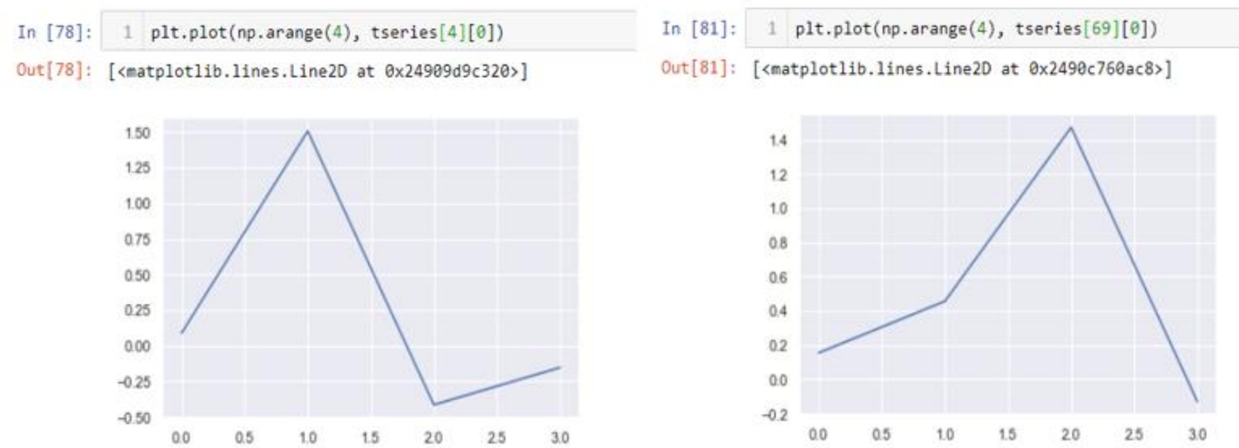


Figure 4.6-b

6- The previous steps lead us to predict tomorrow return by checking the curve of the current four days and comparing it to the curve of another T.series which has an optimal distance, if the curves are highly identical, then the return of tomorrow would be mostly profitable and then we buy the stock.

Out[57]:

	A	B	Dist	A Ret	B Ret
3312	4	69	0.778629	1.360843	-1.696072
3439	4	196	0.608376	1.360843	0.410596
3609	4	366	0.973192	1.360843	0.040522
3790	4	547	0.832545	1.360843	-1.447712
3891	4	648	0.548912	1.360843	-0.510458
4035	4	792	0.740197	1.360843	0.819056
	4	*	Optimal (<0)	1.360843	Mostly positive

Figure 4.7

The Statisctics of applying DTW are shown in the table 4.1 and the figure 4.7, and it can be seen the higher sharp value an d mean we got comparing to the regression model studied before.

Trades	729	**Standard Deviation**	**0.801**
Wins	439	Max Loss	-3.591
Losses	287	Max Win	3.454
Breakeven	3	**Sharpe Ratio**	**1.9731**
Win/Loss Ratio	1.53	**Mean**	**0.1**
Mean Win	0.566	Mean Loss	-0.613

Table 4.1

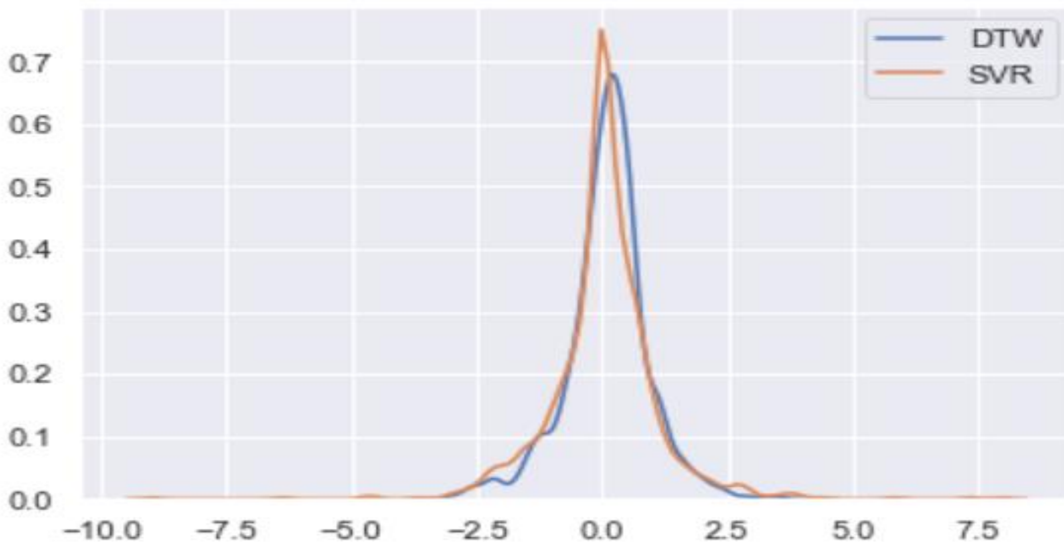


Figure 4.7

Comparing all startegies

	Mean Win	STD	Sharp ratio
Extened(Daily return)	0,769	1,212	0,274
Extened (inter-day)	0,643	1,004	0,0211
Extened(OVER-Night)	0,415	0,656	0,4619
SVR	0452	0,71	0,18
DTW	0,566	0,801	1,9731

Table 4.2