Ke Gui

📈Python for finance series

# Feature Engineering & Feature Selection

## How to apply modern Machine Learning on Volume Spread Analysis (VSA)

Photo by Nong Vang on Unsplash

## 📈Python For Finance Series

- Identifying Outliers
- Identifying Outliers — Part Two
- Identifying Outliers — Part Three
- Stylized Facts
- Feature Engineering & Feature Selection
- Data Transformation
- Fractionally Differentiated Features
- Data Labelling
- Meta-labeling and Stacking

Following up the previous posts in these series, this time we are going to explore a real Technical Analysis (TA) in the financial market. For a very long time, I have been fascinated by the inner logic of TA called Volume Spread Analysis (VSA). I have found no articles on applying modern Machine learning on this time proving long-lasting technique. Here I am trying to throw out a minnow to catch a whale. If I could make some noise in this field, it was worth the time I spent on this article.

Especially, after I read David H. Weis's Trades About to Happen, in his book he described:

To closely listen to the market, as also well said from this quote below, just as it may not be possible to predict the future, it is also hard to neglect things about to happen. The key is to capture what is about to happen and follow the flow.

> For the gods perceive things in the future;
> Ordinary people things in the present;
> But the wise perceive things about to happen.
>
> —**Philostratos,** ***Life of Apollonies of Tyana***

But how to perceive things about to happen, a statement made long ago by Richard Wyckoff gives some clues:

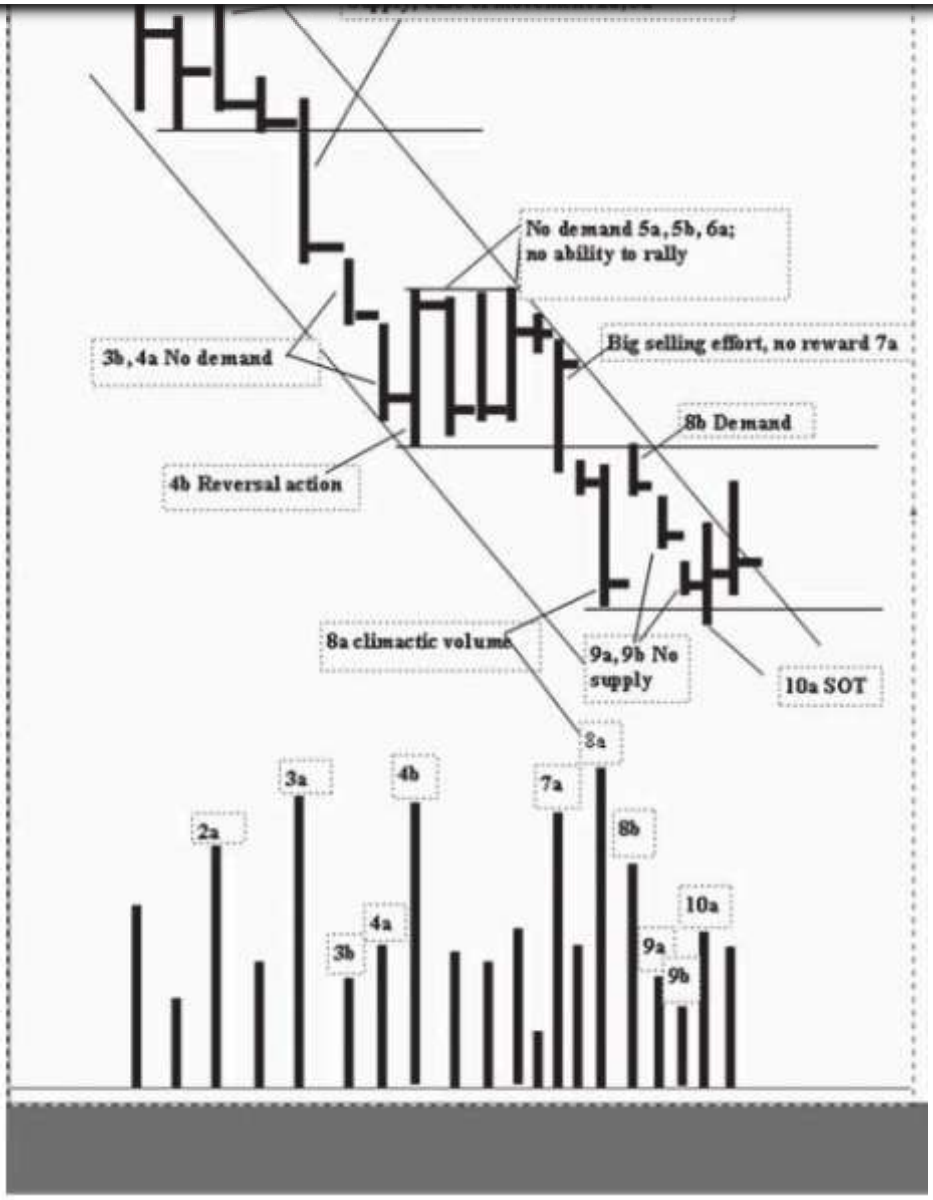
"Successful tape reading [chart reading] is a study of Force. It requires ability to judge which side has the greatest pulling power and one must have the courage to go with that side. There are critical points which occur in each swing just as in the life of a business or of an individual. At these junctures it seems as though a feather's weight on either side would determine the immediate trend. Any one who can spot these points has much to win and little to lose."[2]

But how to interpret the market behaviours? One of the eloquent description of market forces by Richard Wyckoff is very instructive:

"The market is like a slowly revolving wheel: Whether the wheel will continue to revolve in the same direction, stand still or reverse depends entirely upon

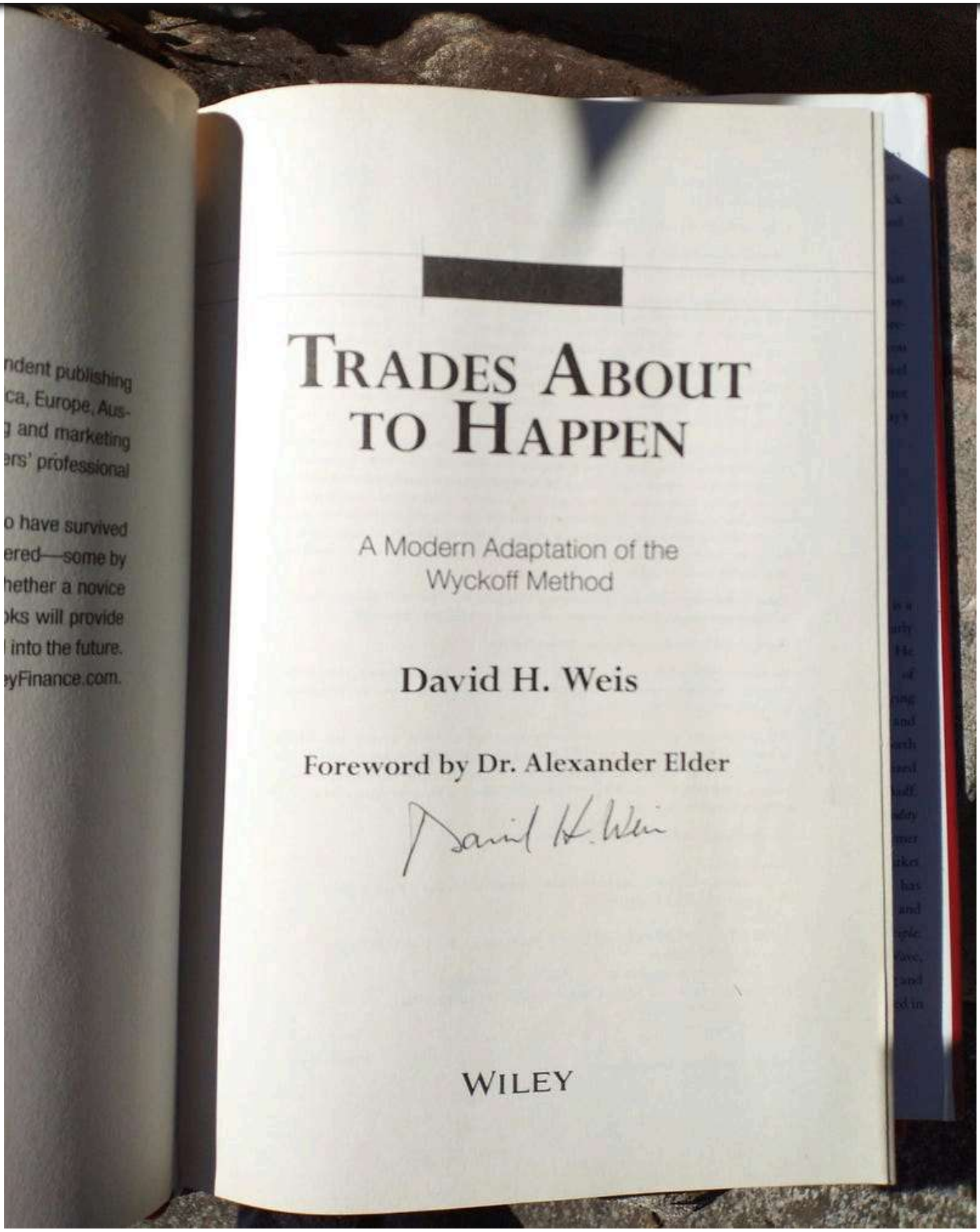
impulse from the most recent dominating force, and revolves until it comes to a standstill or is subjected to other influences."[2]

David H. Weis gives a marvellous example of how to interpret the bars and relate them to the market behaviours. Through his construction of a hypothetical bar behaviour, every single bar becomes alive and rushes to tell you their stories.




Hypothetical Behaviour

For all the details of the analysis, please refer to David's book.




Purchased this book before it officially released and got David's signature.

relationship between volume and price to predict market direction by following the professional traders, so-called market maker. All the interpretations of market behaviours follow 3 basic laws:

- The Law of Supply and Demand
- The Law of Effort vs. Results
- The Law of Cause and Effect

There are also three big names in VSA's development history.

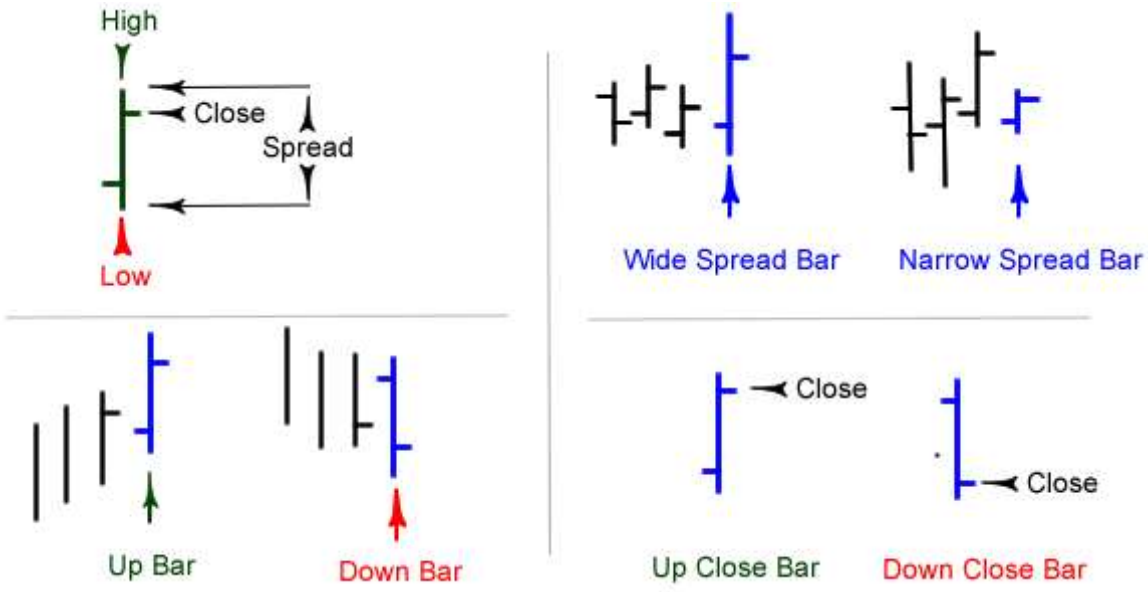
- Jesse Livermore
- Richard Wyckoff
- Tom Williams

And tons of learning materials can be found online. For a beginner, I would recommend the following 2 books.

- Master the Markets by Tom Williams
- Trades About to Happen by David H. Weis

Also, if you only want to have a quick peek on this topic, there is a nice article on VSA from here.

One of the great advantages of Machine learning / Deep learning lies on the no need for feature engineering. The basic of VSA is, as said in its name, volume, the spread of price range, location of the close related to the change of stock price in a bar.




Definition of bars

- Volume: pretty straight forward
- Range/Spread: Difference between high and close
- Closing Price Relative to Range: Is the closing price near the top or the bottom of the price bar?
- The change of stock price: pretty straight forward

There are many materials for VSA online. I have found these 7 live trading series of videos are pretty decent.

and this one.

More can be found on YouTube.

There are many terminologies created by Richard Wyckoff, like Sign of strength (SOS), Sign of weakness (SOW) etc.. However, most of those terminologies are purely the combination of those 4 basic features. I don't believe that, with deep learning, over-engineering features is a sensible thing to do. Considering one of the advantages of deep learning is that it completely automates what used to be the most crucial step in a machine-learning workflow: feature engineering. The thing we need to do is to tell the algorithm where to look at, rather than babysitting them step by step. Without further ado, let's dive into the code.

## 1. Data preparation

here, here and here.

```
#import all the libraries
import pandas as pd
import numpy as np
import seaborn as sns
import yfinance as yf  #the stock data from Yahoo Finance
```

```
import matplotlib.pyplot as plt #set the parameters for plotting
plt.style.use('seaborn')
plt.rcParams['figure.dpi'] = 300
```

```
#define a function to get data
def get_data(symbols, begin_date=None,end_date=None):
    df = yf.download('AAPL', start = '2000-01-01',
                     auto_adjust=True,#only download adjusted data
                     end= '2010-12-31')
    #my convention: always lowercase
    df.columns = ['open','high','low',
                  'close','volume']

    return df
prices = get_data('AAPL', '2000-01-01', '2010-12-31')
prices.head()
```

| | | | | | |
|---|---|---|---|---|---|
| 1999-12-31 | 3.115145 | 3.174940 | 3.070781 | 3.173011 | 40952800 |
| 2000-01-03 | 3.236664 | 3.471988 | 3.138292 | 3.454628 | 133949200 |
| 2000-01-04 | 3.340825 | 3.414122 | 3.122861 | 3.163368 | 128094400 |
| 2000-01-05 | 3.201946 | 3.412194 | 3.178799 | 3.209661 | 194580400 |
| 2000-01-06 | 3.275243 | 3.302247 | 2.931901 | 2.931901 | 191993200 |

### ✍Tip!

*The data we download this time is adjusted data from* `yfinance` *by setting the* `auto_adjust=True`*. If you have access to tick data, by all means. It would be much better with tick data as articulated from* Advances in Financial Machine Learning *by* Marcos Prado. Anyway, 10 years adjusted data only gives 2766 entries, which is far from “Big Data”.

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2766 entries, 1999-12-31 to 2010-12-29
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   open    2766 non-null   float64
 1   high    2766 non-null   float64
 2   low     2766 non-null   float64
 3   close   2766 non-null   float64
 4   volume  2766 non-null   int64  
dtypes: float64(4), int64(1)
```

## 2. Feature Engineering

The key point of combining VSA with modern data science is through reading and interpreting the bars' own actions, one (hopefully algorithm) can construct a story of the market behaviours. The story might not be easily understood by a human, but works in a sophisticated way.

- Volume: pretty straight forward
- Range/Spread: Difference between high and close

```
def price_spread(df):
    return (df.high - df.low)
```

- Closing Price Relative to Range: Is the closing price near the top or the bottom of the price bar?

```
def close_location(df):
    return (df.high - df.close) / (df.high - df.low)
```

```
#o indicates the close is the high of the day, and 1 means close
#is the low of the day and the smaller the value, the closer the #close price t
```

- The change of stock price: pretty straight forward

Now comes the tricky part,

“When viewed in a larger context, some of the price bars take on a new meaning.”

To do that, we need to reconstruct a High(H), Low(L), Close(C) and Volume(V) bar at varied time span.

```
def create_HLCV(i):
'''
#i: days
#as we don't care about open that much, that leaves volume,
#high,low and close
'''
```

```
    df = pd.DataFrame(index=prices.index)
```

```
    df[f'high_{i}D'] = prices.high.rolling(i).max()
    df[f'low_{i}D'] = prices.low.rolling(i).min()
    df[f'close_{i}D'] = prices.close.rolling(i).\
                        apply(lambda x:x[-1])
    # close_2D = close as rolling backwards means today is
    #literally, the last day of the rolling window.
    df[f'volume_{i}D'] = prices.volume.rolling(i).sum()

    return df
```

next step, create those 4 basic features based on a different time scale.

```
    df = create_HLCV(i)
    high = df[f'high_{i}D']
    low = df[f'low_{i}D']
    close = df[f'close_{i}D']
    volume = df[f'volume_{i}D']

    features = pd.DataFrame(index=prices.index)
    features[f'volume_{i}D'] = volume
    features[f'price_spread_{i}D'] = high - low
    features[f'close_loc_{i}D'] = (high - close) / (high - low)
    features[f'close_change_{i}D'] = close.diff()

    return features
```

The time spans that I would like to explore are 1, 2, 3 days and 1 week, 1 month, 2 months, 3 months, which roughly are [1,2,3,5,20,40,60] days. Now, we can create a whole bunch of features,

```
def create_bunch_of_features():
    days = [1,2,3,5,20,40,60]
    bunch_of_features = pd.DataFrame(index=prices.index)
    for day in days:
        f = create_features(day)
        bunch_of_features = bunch_of_features.join(f)

    return bunch_of_features
```

```
bunch_of_features = create_bunch_of_features()
bunch_of_features.info()
```

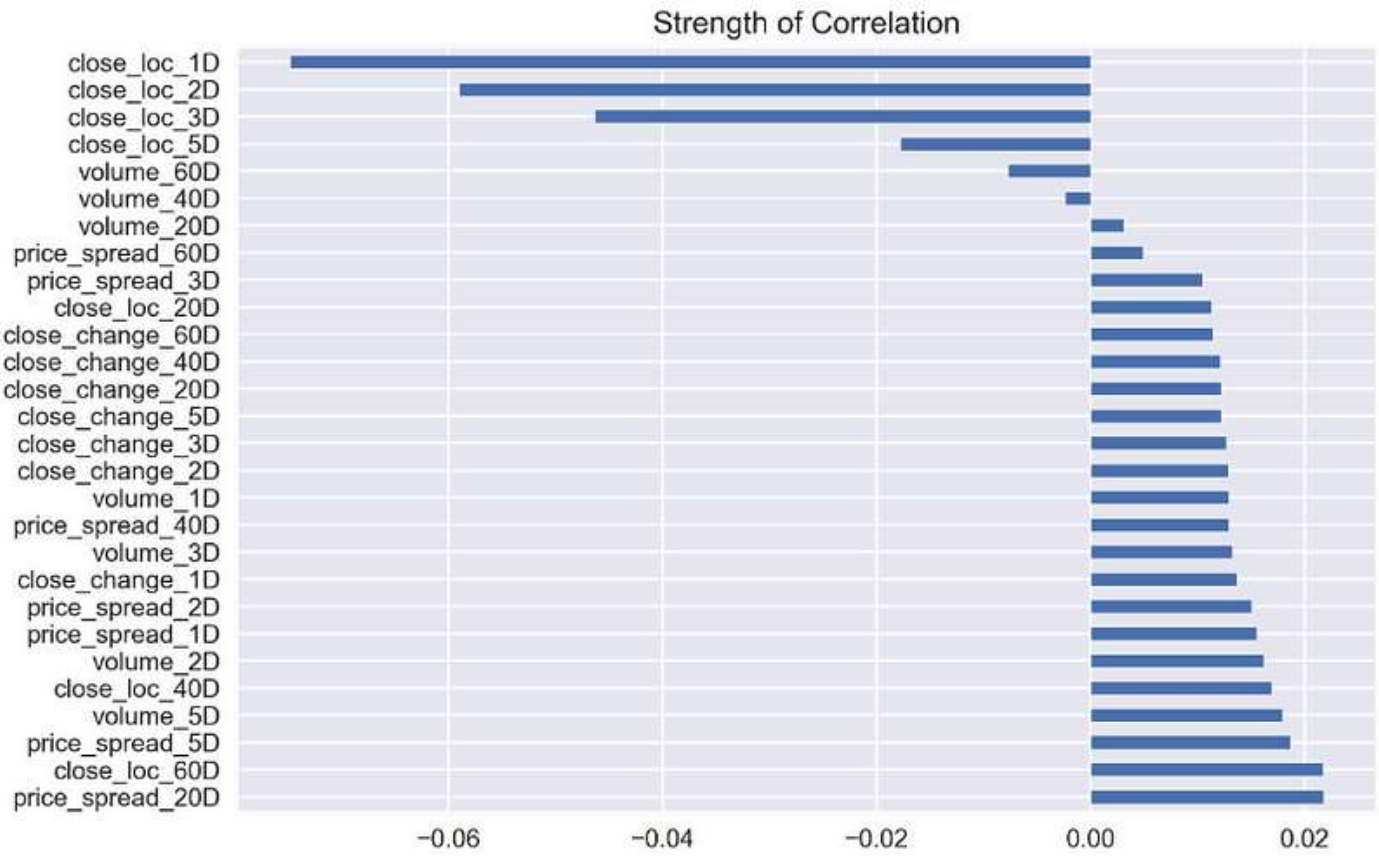
```
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   volume_1D         2766 non-null   float64
 1   price_spread_1D   2766 non-null   float64
 2   close_loc_1D      2766 non-null   float64
 3   close_change_1D   2765 non-null   float64
 4   volume_2D         2765 non-null   float64
 5   price_spread_2D   2765 non-null   float64
 6   close_loc_2D      2765 non-null   float64
 7   close_change_2D   2764 non-null   float64
 8   volume_3D         2764 non-null   float64
 9   price_spread_3D   2764 non-null   float64
 10  close_loc_3D      2764 non-null   float64
 11  close_change_3D   2763 non-null   float64
 12  volume_5D         2762 non-null   float64
 13  price_spread_5D   2762 non-null   float64
 14  close_loc_5D      2762 non-null   float64
 15  close_change_5D   2761 non-null   float64
 16  volume_20D        2747 non-null   float64
 17  price_spread_20D  2747 non-null   float64
 18  close_loc_20D     2747 non-null   float64
 19  close_change_20D  2746 non-null   float64
 20  volume_40D        2727 non-null   float64
 21  price_spread_40D  2727 non-null   float64
 22  close_loc_40D     2727 non-null   float64
 23  close_change_40D  2726 non-null   float64
 24  volume_60D        2707 non-null   float64
 25  price_spread_60D  2707 non-null   float64
 26  close_loc_60D     2707 non-null   float64
 27  close_change_60D  2706 non-null   float64
dtypes: float64(28)
```

To make things easy to understand, our target outcome will only be the next day's return.

```
# next day's returns as outcomes
outcomes = pd.DataFrame(index=prices.index)
outcomes['close_1'] = prices.close.pct_change(-1)
```

## 3. Feature Selection

```
corr = bunch_of_features.corrwith(outcomes.close_1)
corr.sort_values(ascending=False).plot.barh(title = 'Strength of Correlation');
```




It is hard to say there are some correlations, as all the numbers are well below 0.8.

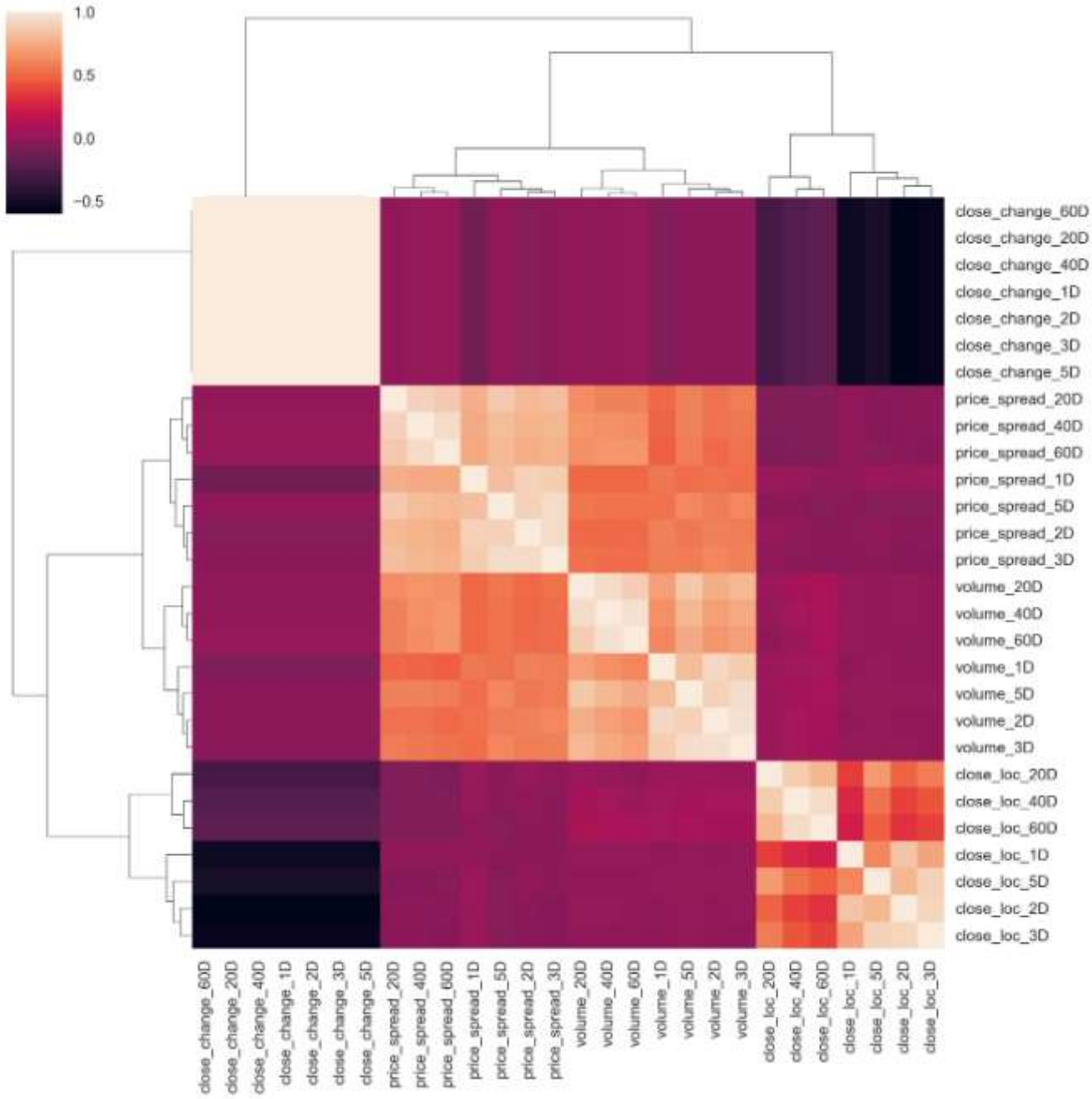
```
corr.sort_values(ascending=False)
```

```
volume_5D           0.017928
close_loc_40D       0.016927
volume_2D           0.016181
price_spread_1D     0.015564
price_spread_2D     0.015013
close_change_1D     0.013658
volume_3D           0.013248
price_spread_40D    0.012933
volume_1D           0.012911
close_change_2D     0.012846
close_change_3D     0.012706
close_change_5D     0.012185
close_change_20D    0.012182
close_change_40D    0.012126
close_change_60D    0.011414
close_loc_20D       0.011323
price_spread_3D     0.010494
price_spread_60D    0.004910
volume_20D          0.003072
volume_40D         -0.002367
volume_60D         -0.007679
close_loc_5D       -0.017735
close_loc_3D       -0.046320
close_loc_2D       -0.059042
close_loc_1D       -0.074800
dtype: float64
```

Next, let's see how those features related to each other.

```
corr_matrix = bunch_of_features.corr()
```

Instead of making heatmap, I am trying to use Seaborn's Clustermap to cluster row-wise or col-wise to see if there is any pattern emerges. Seaborn's Clustermap function is great for making simple heatmaps and hierarchically-clustered heatmaps with dendrograms on both rows and/or columns. This reorganizes the data for the rows and columns and displays similar content next to one another for even more depth of understanding the data. A nice tutorial about cluster map can be found here. To get a cluster map, all you need is actually one line of code.




If you carefully scrutinize the graph, some conclusions can be drawn:

- Price spread closely related to the volume, as clearly shown at the centre of the graph.

- From the pale colour of the top left corner, close price change does pair with itself, which makes perfect sense. However, it is a bit random as no cluster pattern at varied time scale. I would expect that 2Days change should be paired with 3Days change.

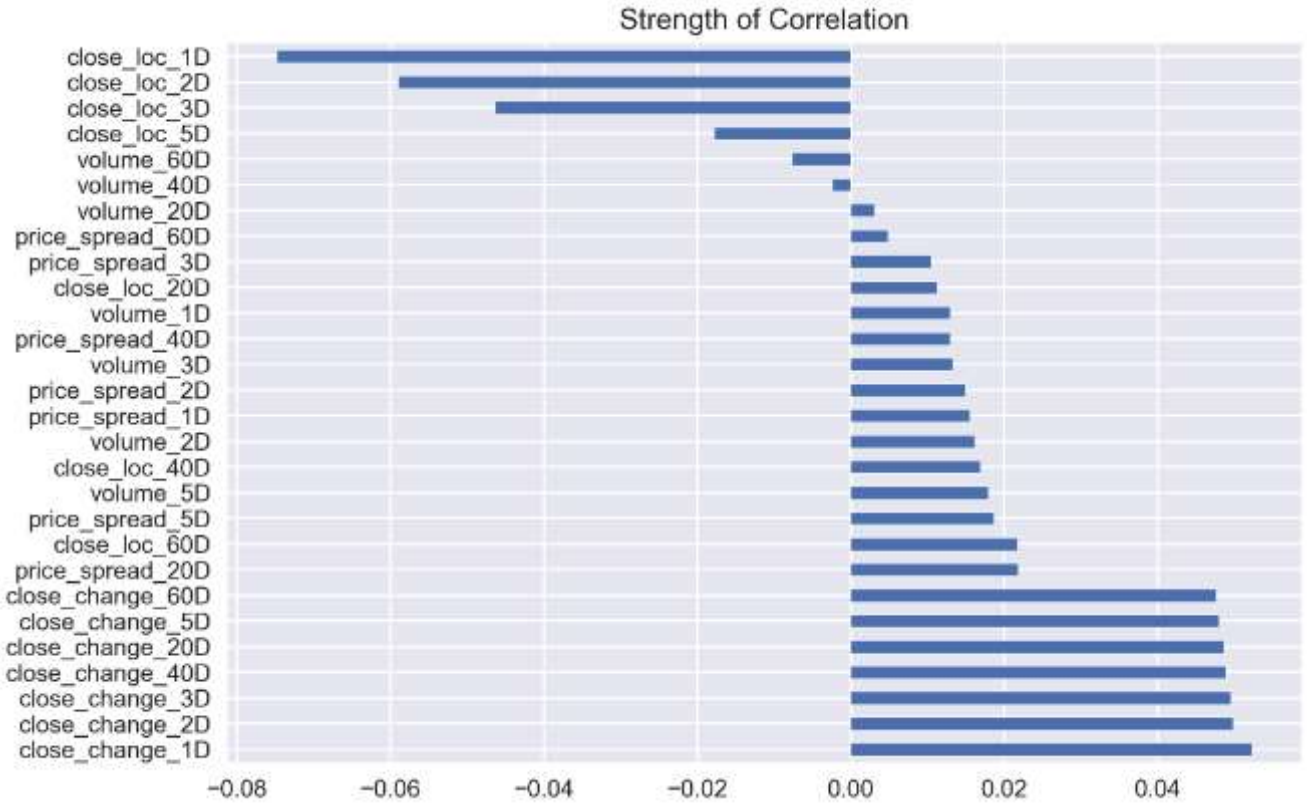
The randomness of the close price difference could thank to the characteristics of the stock price itself. Simple percentage return might be a better option. This can be realized by modifying the close `diff()` to close `pct_change()`.

```
def create_features_v1(i):
    df = create_HLCV(i)
    high = df[f'high_{i}D']
    low = df[f'low_{i}D']
    close = df[f'close_{i}D']
    volume = df[f'volume_{i}D']

    features = pd.DataFrame(index=prices.index)
    features[f'volume_{i}D'] = volume
    features[f'price_spread_{i}D'] = high - low
    features[f'close_loc_{i}D'] = (high - close) / (high - low)
    #only change here
    features[f'close_change_{i}D'] = close.pct_change()

    return features
```

and do everything again.

```
def create_bunch_of_features_v1():
    days = [1,2,3,5,20,40,60]
```

```
            bunch_of_features = bunch_of_features.join(f)

        return bunch_of_features




bunch_of_features_v1 = create_bunch_of_features_v1()




#check the correlation
corr_v1 = bunch_of_features_v1.corrwith(outcomes.close_1)
corr_v1.sort_values(ascending=False).plot.barh( title = 'Strength of Correlatic
```

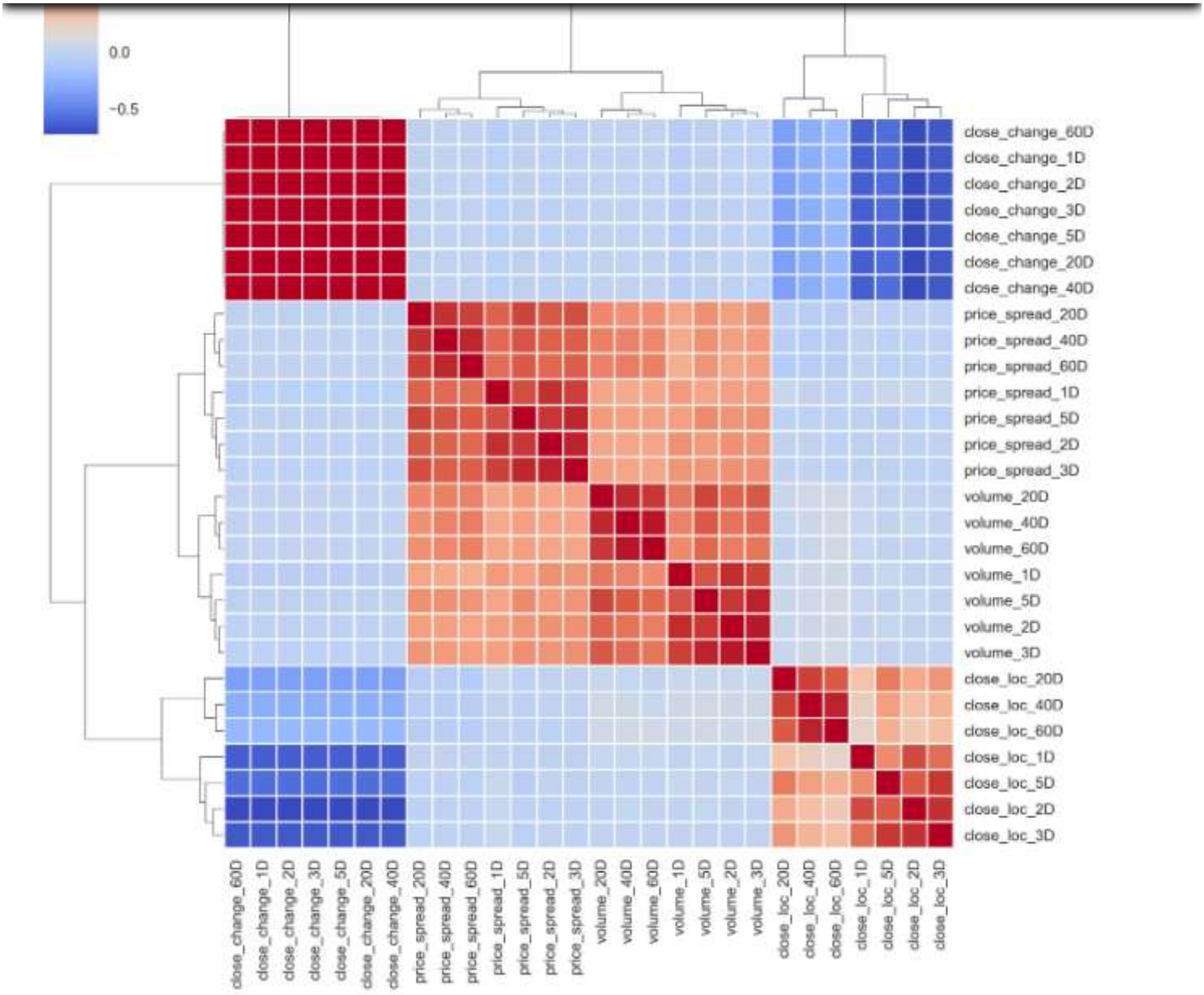


```
corr_v1.sort_values(ascending=False)
```

```
close_change_1D     0.052364
close_change_2D     0.049871
close_change_3D     0.049548
close_change_40D    0.048889
close_change_20D    0.048570
close_change_5D     0.047997
close_change_60D    0.047615
price_spread_20D    0.021786
close_loc_60D       0.021710
price_spread_5D     0.018695
volume_5D           0.017928
close_loc_40D       0.016927
volume_2D           0.016181
price_spread_1D     0.015564
price_spread_2D     0.015013
volume_3D           0.013248
price_spread_40D    0.012933
volume_1D           0.012911
close_loc_20D       0.011323
price_spread_3D     0.010494
price_spread_60D    0.004910
volume_20D          0.003072
volume_40D         -0.002367
volume_60D         -0.007679
close_loc_5D       -0.017735
close_loc_3D       -0.046320
close_loc_2D       -0.059042
close_loc_1D       -0.074800
dtype: float64
```

## What happens to the correlation between features?

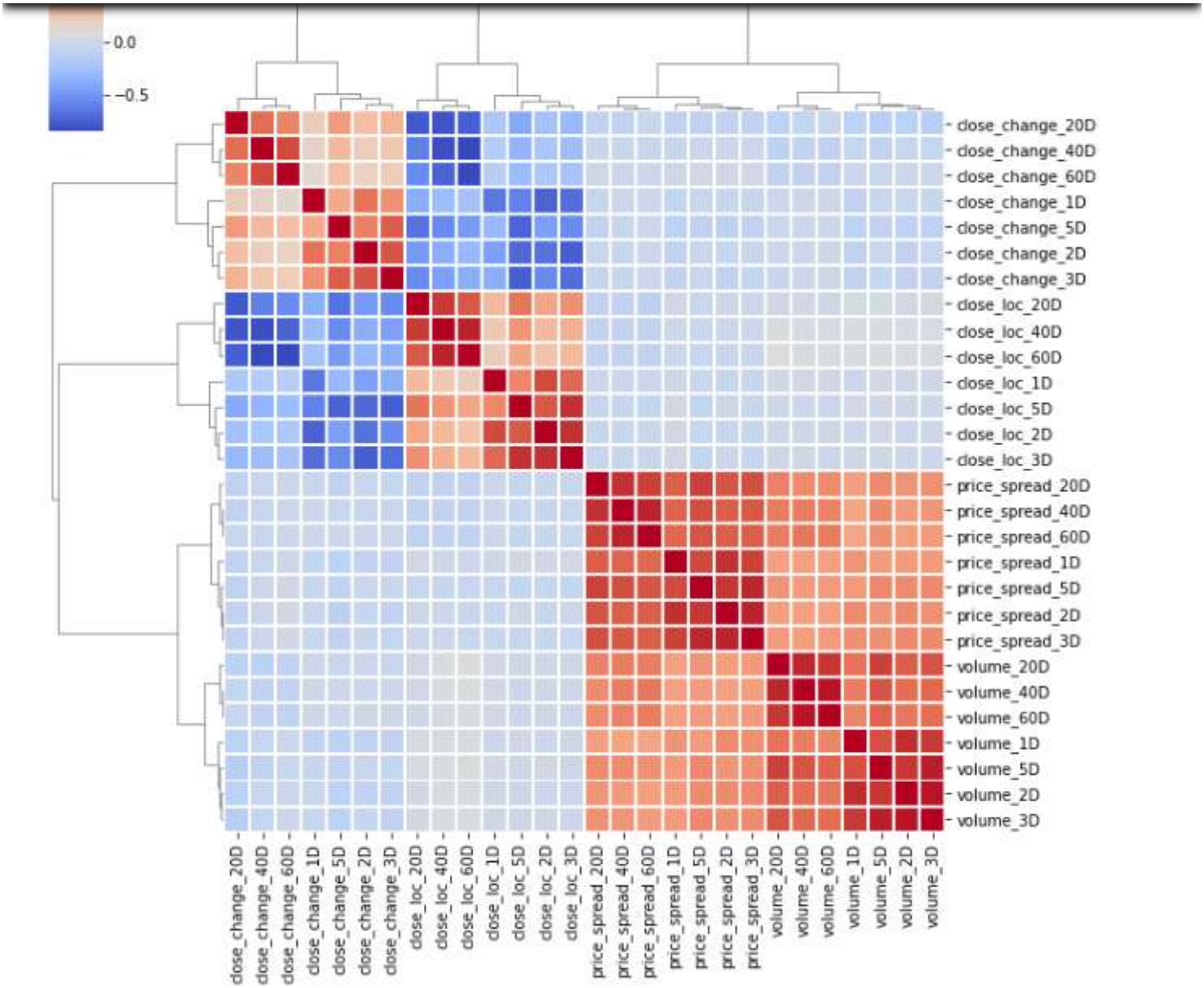
```
corr_matrix_v1 = bunch_of_features_v1.corr()
sns.clustermap(corr_matrix_v1, cmap='coolwarm', linewidth=1)
```




Well, the pattern remains unchanged. Let's change the default method from "average" to "ward". These two methods are similar, but "ward" is more like K-MEANs clustering. A nice tutorial on this topic can be found here.

Group Average WARDS method

```
sns.clustermap(corr_matrix_v1, cmap='coolwarm', linewidth=1,
                               method='ward')
```



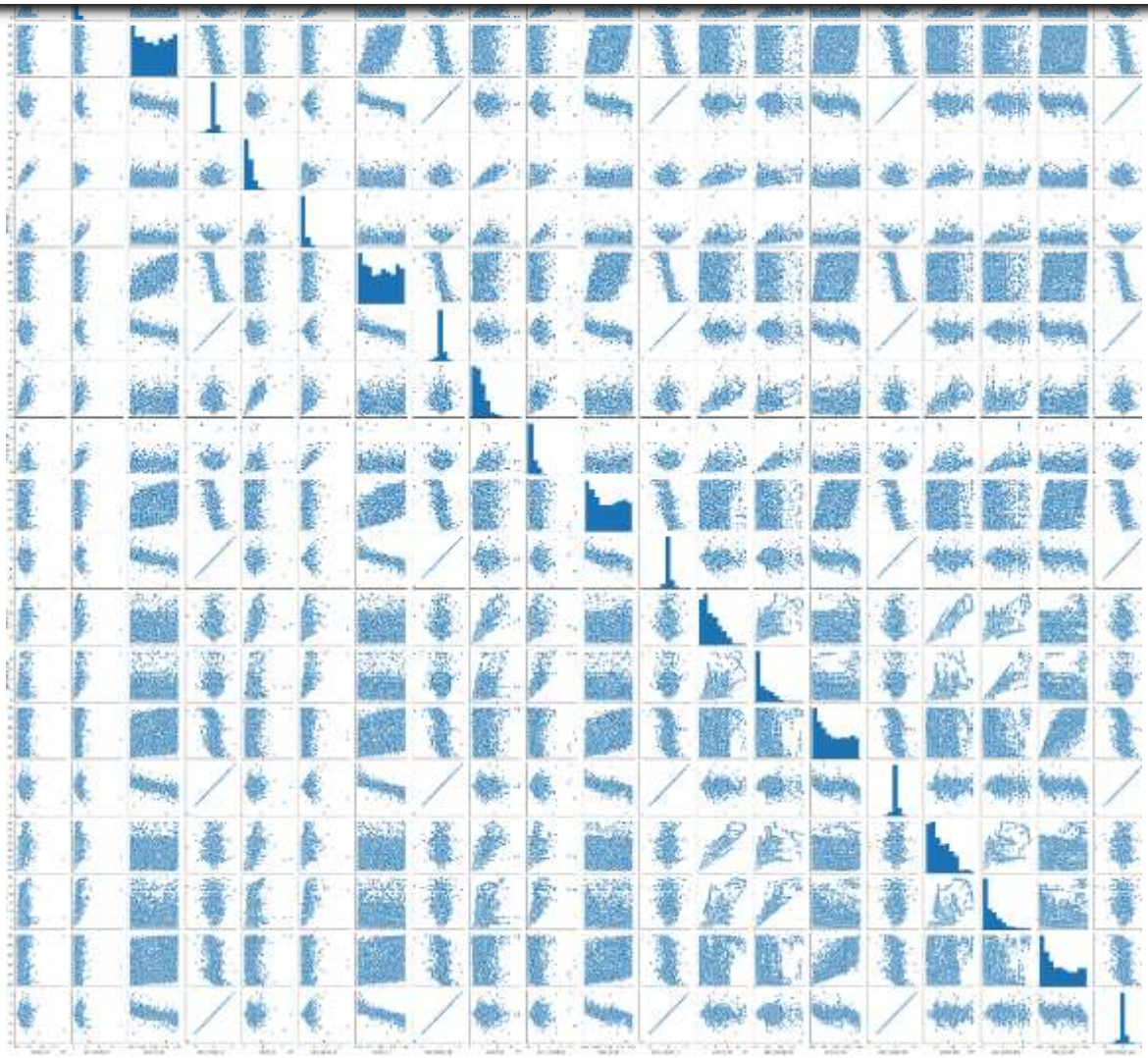

To select features, we want to pick those that have the strongest, most persistent relationships to the target outcome. At the meantime, to minimize the amount of overlap or collinearity in your selected features to avoid noise and waste of computer power. For those features that paired together in a cluster, I only pick the one that has a stronger correlation with the outcome. By just looking at the cluster map, a few features are picked out.

```
'volume_3D', 'volume_60D',
'price_spread_3D','price_spread_60D',
'close_change_3D','close_change_60D']
```

```
selected_features_v1 = bunch_of_features.drop \
(labels=deselected_features_v1, axis=1)
```

Next, we are going to take a look at pair-plot, A pair plot is a great method to identify trends for follow-up analysis, allowing us to see both distributions of single variables and relationships between multiple variables. Again, all we need is a single line of code.

```
sns.pairplot(selected_features_v1)
```

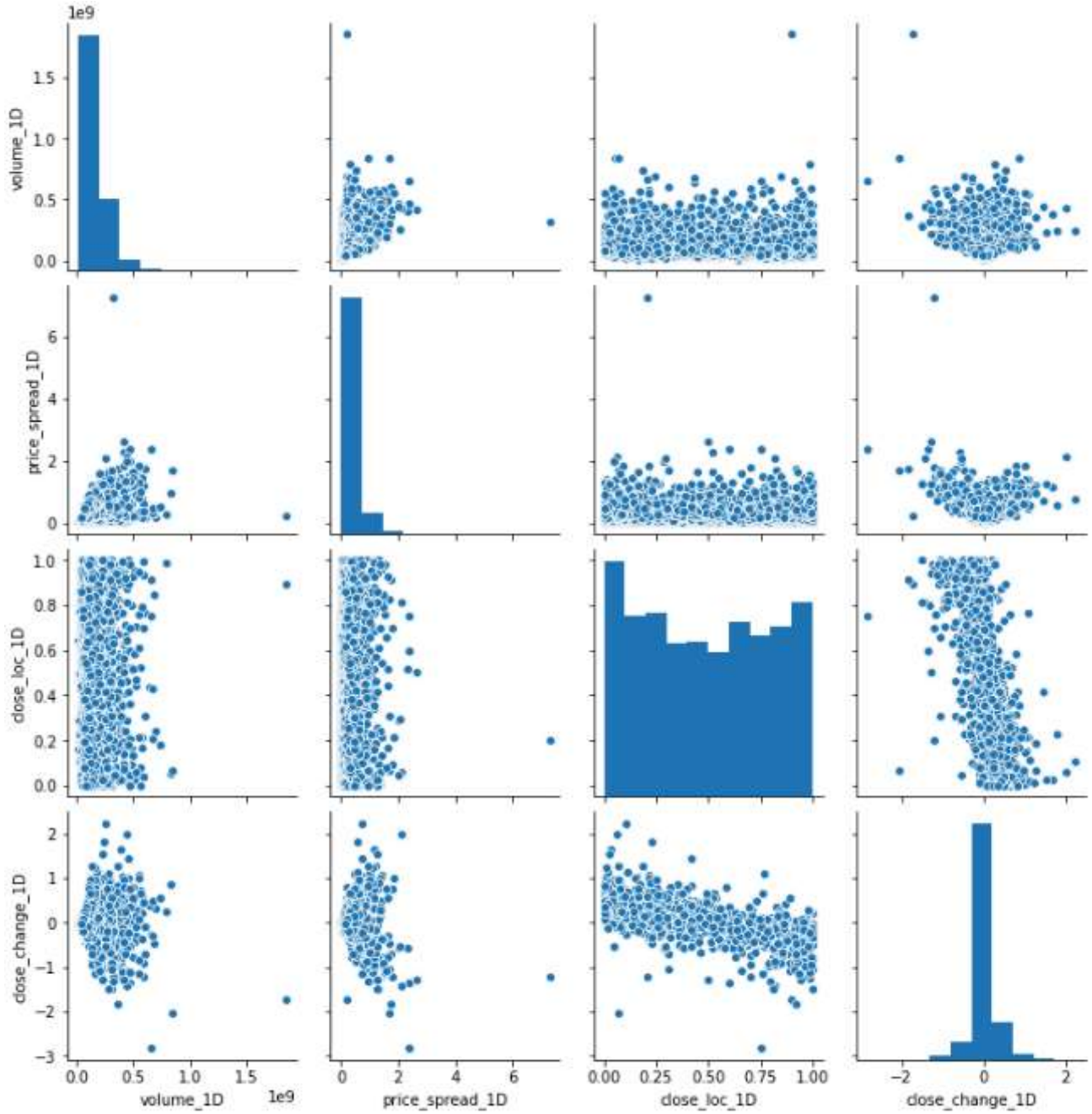


The graph is overwhelming and hard to see. Let's take a small group as an example.

```
selected_features_1D_list = ['volume_1D', 'price_spread_1D',\          'close_lo
```

[selected_features_1D_list]

```
sns.pairplot(selected_features_1D)
```



Let's deal with the outliers for now. In order to do everything in one go, I will join the outcome with features and remove outliers together.

```
features_outcomes = selected_features_v1.join(outcomes)
features_outcomes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2766 entries, 1999-12-31 to 2010-12-29
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   volume_1D         2766 non-null   float64
 1   price_spread_1D   2766 non-null   float64
 2   close_loc_1D      2766 non-null   float64
 3   close_change_1D   2765 non-null   float64
 4   volume_2D         2765 non-null   float64
 5   price_spread_2D   2765 non-null   float64
 6   close_loc_2D      2765 non-null   float64
 7   close_change_2D   2764 non-null   float64
 8   volume_5D         2762 non-null   float64
 9   price_spread_5D   2762 non-null   float64
 10  close_loc_5D      2762 non-null   float64
 11  close_change_5D   2761 non-null   float64
 12  volume_20D        2747 non-null   float64
 13  price_spread_20D  2747 non-null   float64
 14  close_loc_20D     2747 non-null   float64
 15  close_change_20D  2746 non-null   float64
 16  volume_40D        2727 non-null   float64
 17  price_spread_40D  2727 non-null   float64
 18  close_loc_40D     2727 non-null   float64
 19  close_change_40D  2726 non-null   float64
 20  close_1           2765 non-null   float64
dtypes: float64(21)
```

I will use the same method described here, here and here to remove the outliers.

```
def get_outliers(df, i=4):
    #i is number of sigma, which define the boundary along mean
    outliers = pd.DataFrame()

    for col in df.columns:
        mu = stats.loc['mean', col]
        sigma = stats.loc['std', col]
        condition = (df[col] > mu + sigma * i) | (df[col] < mu - sigma * i)
        outliers[f'{col}_outliers'] = df[col][condition]

    return outliers
```

```
outliers = get_outliers(features_outcomes, i=1)
outliers.info()
```

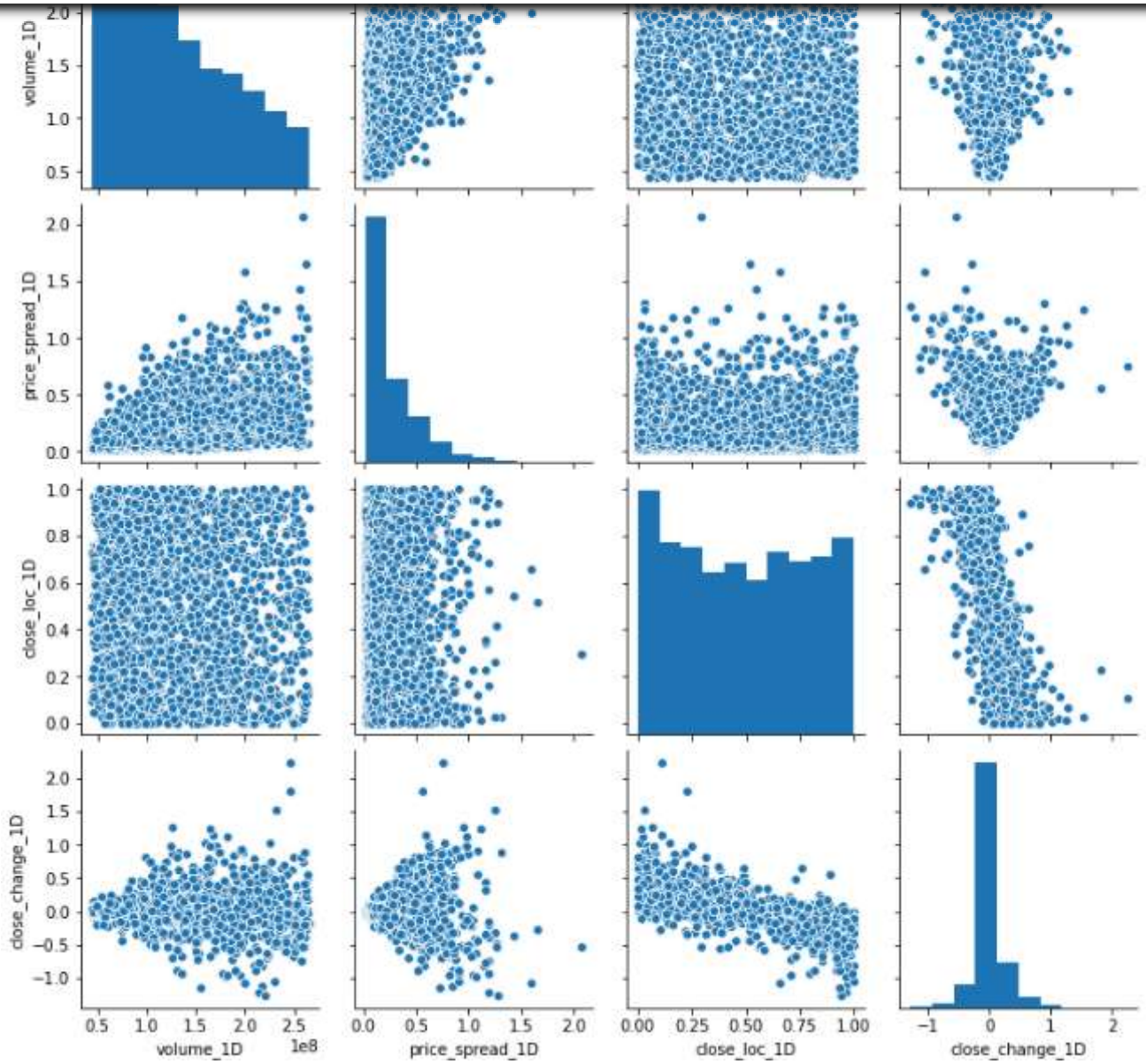
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 467 entries, 1999-12-31 to 2010-12-29
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   volume_1D_outliers        467 non-null    float64
 1   price_spread_1D_outliers  149 non-null    float64
 2   close_loc_1D_outliers     208 non-null    float64
 3   close_change_1D_outliers  171 non-null    float64
 4   volume_2D_outliers        387 non-null    float64
 5   price_spread_2D_outliers  158 non-null    float64
 6   close_loc_2D_outliers     234 non-null    float64
 7   close_change_2D_outliers  171 non-null    float64
 8   volume_5D_outliers        337 non-null    float64
 9   price_spread_5D_outliers  154 non-null    float64
 10  close_loc_5D_outliers     209 non-null    float64
 11  close_change_5D_outliers  171 non-null    float64
 12  volume_20D_outliers       306 non-null    float64
 13  price_spread_20D_outliers 131 non-null    float64
 14  close_loc_20D_outliers    202 non-null    float64
 15  close_change_20D_outliers 170 non-null    float64
 16  volume_40D_outliers       304 non-null    float64
 17  price_spread_40D_outliers 141 non-null    float64
 18  close_loc_40D_outliers    226 non-null    float64
 19  close_change_40D_outliers 170 non-null    float64
 20  close_1_outliers          121 non-null    float64
dtypes: float64(21)
```

```
features_outcomes_rmv_outliers = features_outcomes.drop(index = outliers.index)
features_outcomes_rmv_outliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2261 entries, 2000-02-29 to 2010-12-27
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   volume_1D         2261 non-null   float64
 1   price_spread_1D   2261 non-null   float64
 2   close_loc_1D      2261 non-null   float64
 3   close_change_1D   2261 non-null   float64
 4   volume_2D         2261 non-null   float64
 5   price_spread_2D   2261 non-null   float64
 6   close_loc_2D      2261 non-null   float64
 7   close_change_2D   2261 non-null   float64
 8   volume_5D         2261 non-null   float64
 9   price_spread_5D   2261 non-null   float64
 10  close_loc_5D      2261 non-null   float64
 11  close_change_5D   2261 non-null   float64
 12  volume_20D        2261 non-null   float64
 13  price_spread_20D  2261 non-null   float64
 14  close_loc_20D     2261 non-null   float64
 15  close_change_20D  2261 non-null   float64
 16  volume_40D        2261 non-null   float64
 17  price_spread_40D  2261 non-null   float64
 18  close_loc_40D     2261 non-null   float64
 19  close_change_40D  2261 non-null   float64
 20  close_1           2261 non-null   float64
dtypes: float64(21)
```

With the outliers removed, we can do the pair plot again.

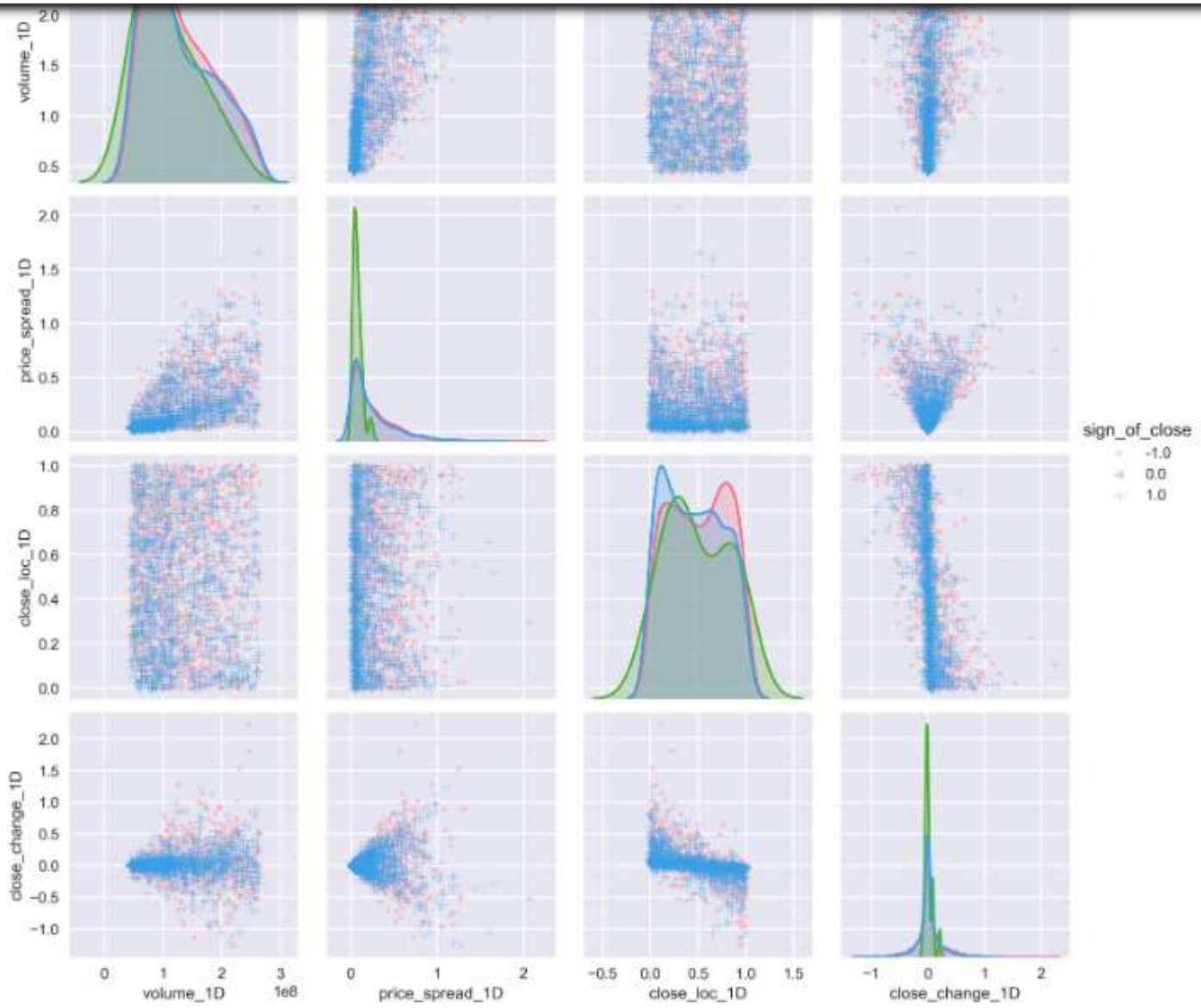
```
sns.pairplot(features_outcomes_rmv_outliers, vars=selected_features_1D_list);
```




Now, the plots are looking much better, but it is barely to draw any useful conclusions. It would be nice to see which spots are down moves and which are up moves in conjunction with those features. I can extract the sign of stock price change and add an extra dimension to the plots.

Now, let's re-plot the `pairplot()` again with a bit of tweak to make the graph pretty.

```
sns.pairplot(features_outcomes_rmv_outliers,
             vars=selected_features_1D_list,
             diag_kind='kde',
             palette='husl', hue='sign_of_close',
             markers = ['*', '<', '+'],
             plot_kws={'alpha':0.3});#transparence:0.3
```




Now, it looks much better. Clearly, when the prices go up, they (the blue spot) are denser and aggregate at a certain location. Whereas on down days, they spread everywhere.

***I would really appreciate it if you could shed some light on the pair plot and leave your comments below, thanks.***

Here is the summary of all the codes used in this article:

```
import pandas as pd
import numpy as np
import seaborn as sns
import yfinance as yf  #the stock data from Yahoo Finance
```

```
import matplotlib.pyplot as plt #set the parameters for plotting
plt.style.use('seaborn')
plt.rcParams['figure.dpi'] = 300
```

```
#define a function to get data
def get_data(symbols, begin_date=None,end_date=None):
    df = yf.download('AAPL', start = '2000-01-01',
                     auto_adjust=True,#only download adjusted data
                     end= '2010-12-31')
    #my convention: always lowercase
    df.columns = ['open','high','low',
                  'close','volume']

    return df
```

```
prices = get_data('AAPL', '2000-01-01', '2010-12-31')
```

```
#create some features
```

```
#as we don't care open that much, that leaves volume,
#high,low and close
```

```
df = pd.DataFrame(index=prices.index)
    df[f'high_{i}D'] = prices.high.rolling(i).max()
    df[f'low_{i}D'] = prices.low.rolling(i).min()
    df[f'close_{i}D'] = prices.close.rolling(i).\
                        apply(lambda x:x[-1])
    # close_2D = close as rolling backwards means today is
    # literly the last day of the rolling window.
    df[f'volume_{i}D'] = prices.volume.rolling(i).sum()

    return df
```

```
def create_features_v1(i):
    df = create_HLCV(i)
    high = df[f'high_{i}D']
    low = df[f'low_{i}D']
    close = df[f'close_{i}D']
    volume = df[f'volume_{i}D']

    features = pd.DataFrame(index=prices.index)
    features[f'volume_{i}D'] = volume
    features[f'price_spread_{i}D'] = high - low
    features[f'close_loc_{i}D'] = (high - close) / (high - low)
    features[f'close_change_{i}D'] = close.pct_change()

    return features
```

```
'''
the timespan that i would like to explore
are 1, 2, 3 days and 1 week, 1 month, 2 month, 3 month
which roughly are [1,2,3,5,20,40,60]
'''
    days = [1,2,3,5,20,40,60]
    bunch_of_features = pd.DataFrame(index=prices.index)
    for day in days:
        f = create_features_v1(day)
        bunch_of_features = bunch_of_features.join(f)

    return bunch_of_features
```

```
bunch_of_features_v1 = create_bunch_of_features_v1()
```

```
#define the outcome target
#here,  to make thing easy to understand, i will only try to predict #the next
outcomes = pd.DataFrame(index=prices.index)
```

```
# next day's returns
outcomes['close_1'] = prices.close.pct_change(-1)
```

```
#decide which features are abundant from cluster map
deselected_features_v1 = ['close_loc_3D','close_loc_60D',
```

```
selected_features_v1 = bunch_of_features_v1.drop(labels=deselected_features_v1,
```

```
#join the features and outcome together to remove the outliers
features_outcomes = selected_features_v1.join(outcomes)
stats = features_outcomes.describe()
```

```
#define the method to identify outliers
def get_outliers(df, i=4):
    #i is number of sigma, which define the boundary along mean
    outliers = pd.DataFrame()

    for col in df.columns:
        mu = stats.loc['mean', col]
        sigma = stats.loc['std', col]
        condition = (df[col] > mu + sigma * i) | (df[col] < mu -   sigma * i)
        outliers[f'{col}_outliers'] = df[col][condition]

    return outliers
```

```
outliers = get_outliers(features_outcomes, i=1)
```

```
#remove all the outliers and Nan value
```

I know this article goes too long, I am better off leaving it here. In the next article, I will do a data transformation to see if I have a way to fix the issue of distribution. Stay tuned!

## References


- Trades About to Happen by David H. Weis
- Rollo Tape [pseud.], Studies in Tape Reading (Burlington, VT: Fraser, 1910), 95.




Feature Engineering | Feature Selection | Trading | Python | Machine Learning

---

Recommended from ReadMedium


Sze Zhong LIM

**Exploring Pyspark.ml for Machine Learning: Crafting Optimal Feature Selection Strategies**

11 min read

Tim Sumner

## A New Coefficient of Correlation

What if you were told there exists a new way to measure the relationship between two variables just like correlation except possibly...

10 min read

Dr. Ernesto Lee

## Advanced Stock Pattern Prediction using LSTM with the Attention Mechanism in TensorFlow: A step by...

Introduction

15 min read

Ayrat Murtazin

## Citadel's Strategy Anyone Can Use—Moving Averages tell more than Strategists

In the investing world, accurately predicting macro market regimes, such as economic downturns, crisis turning points, or the duration of...

5 min read

EODHD APIs

## Practical Guide to Automated Detection Trading Patterns with Python

In order to discuss the topic of a Practical Guide to Automated Detection of Trading Patterns with Python, let's start with the basics...

Chris Kuo/Dr. Dataman

# Conformal Predictions for Time Series Probabilistic Forecasting

Real-world applications and planning require probabilistic forecasts rather than a point estimate. Probabilistic forecasts, also called...

7 min read