HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

MACHINE LEARNING & DATA MINING

PROJECT REPORT

Pair selection for Statistical Arbitrage by Pair Trading using Clustering

NGUYEN KHANH TRUNG
trung.nk205133@sis.hust.edu.vn

NGUYEN KIM TUYEN
tuyen.nk205196@sis.hust.edu.vn

Lecturer: Ph.D. Nguyen Nhat Quang

Department: Computer Science

School: School of Information and Communications Technology

HANOI, 06/2023

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. ABSTRACT

Pair trading is one of the most popular trading strategies since 1980 for finding statistical arbitrage opportunities in the stock market. The logic behind pairs trading is to trade pairs of stocks belonging to the same industry or having similar characteristics, such that their historical returns move together and are expected to continue to do so in the future. In this project, we investigate the application of machine learning methods to find statistical arbitrage opportunities in the stock market using pair trading strategy. Overall, the project demonstrates the potential of machine learning techniques for improving the performance of pair trading strategies and provides a valuable contribution to the field of financial engineering.

CHAPTER 2. INTRODUCTION

Pairs trading is an investment strategy of constructing a portfolio with matching stocks in terms of market risk factors with a long/buy position in the stock we are optimistic and a short/sell position in the stock we are pessimistic. The approach is designed to eliminate market risk and exploit temporary discrepancies in the relative returns of stocks.

The first step in constructing the portfolio is to identify pairs of stocks based on fundamental analysis, i.e. having common factors such as being in the same industry and having similar market capitalization.

The second step is to track return spread between each pair of stocks. A pair of cointegrated assets is selected, that are known to historically move close to each other and are expected to continue to do so. Under the assumption that the spread, defined as the difference in price between the paired assets, is mean-reverting, deviations from the mean can be exploited.

When the spread is abnormally wide and deviation from the mean reaches some predetermined threshold, the outperforming stock is sold short, and the under-performing stock is purchased. As soon as the spread converges back to its mean, the investor liquidates both positions, resulting in a profit. Although this may sound intuitive and trivial, sophisticated machine learning techniques can be used at every step of the pairs trading process. The key to successful pairs trading is the ability to detect patterns in spreads and correctly identify when a spread has become abnormally large and is likely to converge back to it's mean.

In this project, we focus on the first part of the pair trading strategy mentioned above. Our approach will be to use K-Means and DBSCAN clustering algorithm on a large set of features for clustering stocks. Afterwards, we use cointegration test to extract all possible combinations of stocks in each cluster that are within 1% significance level.

CHAPTER 3. METHOD AND APPROACH

For constructing a Pairs Trading strategy, the first task is to find valid, eligible pairs which exhibit unconditional mean-reverting behavior. One simple approach is to iterate through all stock pairs in the universe of stocks. However, iterating through all stock pairs in the universe of stocks can be resource-consuming and not realistic, especially when dealing with a large number of stocks. To address this issue, we propose an alternative approach that leverages clustering techniques to group stocks with similar characteristics together. By clustering the stocks, we can significantly reduce the number of pairs that need to be considered, making the iteration process much more manageable and computationally efficient.

In our approach, we begin by incorporating feature engineering for each stock, combining the price movement features processed above with additional features derived from economic prior knowledge (in our case, Industry Sector and Market Capitalization). These features capture important characteristics of the stocks that are relevant for pair trading. Next, we apply the two different clustering algorithms: K-Means and DBSCAN (Density-Based Spatial Clustering of Applications with Noise) to cluster the stocks into different groups based on their feature similarities. In our envision, DBSCAN is a more suitable choice for this task compared to K-means clustering. This is because DBSCAN has the capability to automatically detect dense regions in the feature space and effectively handle noise and outliers. To test this theory, we experimented with both clustering algorithms.

Once the stocks are clustered, we focus on each cluster individually. Within each cluster, we calculate the p-values for all possible pairs of stocks by iterating through each pairs in a cluster and using the Engle-Granger two-step method on it. This involves running an ordinary least squares (OLS) regression on the two input time series to estimate the cointegrating vector. Then follows by conducting an Augmented Dickey-Fuller (ADF) statistical test to test for time series stationary. We select pairs with p-values less than 1%, indicating a strong likelihood of mean-reverting behavior, as our eligible pairs for the Pairs Trading strategy.

4.1 Data Collection

To start the pair selection process, we consider daily price data with time horizon, early $January$ 2015 to late $December$ 2018 (i.e. 4 years). This choice of timeframe, amounting to around 999 dimensions for each stock, strikes a balance between avoiding computational overload in the clustering models and ensuring comprehensive coverage of the entire time series.

```
import datetime
start_date = datetime.date(2015, 1, 1)
end_date = datetime.date(2018, 12, 31)
pricing = pricing.loc[start_date:end_date]
print(pricing.shape)
pricing.head()
```

```
(999, 750)
```

Ticker	AAA	AAM	AAT	AAV	ABS	ABT	ACB	ACC	ACL	ACM	...	VTJ	VTL	VTO	VTV	VXB	WCS	WSS	X20	YBM	YEG
DTYYYYMMDD																					
2015-01-05	6.223	7.485	NaN	NaN	NaN	31.261	4.991	14.233	4.147	NaN	...	NaN	11.92	4.526	6.860	11.637	82.363	5.2	NaN	NaN	NaN
2015-01-06	6.314	7.433	NaN	NaN	NaN	31.261	5.056	13.758	4.147	NaN	...	NaN	11.92	4.744	6.890	11.637	76.315	5.4	NaN	NaN	NaN
2015-01-07	6.539	7.485	NaN	NaN	NaN	30.968	5.121	13.370	4.328	NaN	...	NaN	11.92	4.799	6.978	11.637	73.488	5.3	NaN	NaN	NaN
2015-01-08	6.404	7.537	NaN	NaN	NaN	32.137	5.154	13.370	4.328	NaN	...	NaN	11.92	4.799	7.508	11.637	73.488	5.3	NaN	NaN	NaN
2015-01-09	6.269	7.069	NaN	NaN	NaN	33.890	5.187	14.060	4.292	NaN	...	NaN	11.92	4.908	7.508	11.637	73.431	5.4	NaN	NaN	NaN

5 rows × 750 columns

Figure 4.1: Daily prices dataframe

4.2 Data Preprocessing

4.2.1 Data Transforming

Calculating Returns

A return, also known as a financial return, in its simplest terms, is the money made or lost on an investment over some period of time. It is the change in price of an asset, investment, or project over time, which may be represented in terms of price change or percentage change.

With the pricing dataframe, we calculate the daily returns on each asset for similarity analysis:

$$Returns_{daily} = \frac{close_i}{close_j} - 1$$

```
# Calculate returns and create a data frame
returns = pricing.pct_change()

returns.replace([np.inf, -np.inf], np.nan, inplace=True)
returns.dropna(inplace=True)

print(returns.shape)
returns.head()
```

```
(998, 556)
```

Ticker	AAA	AAM	ABT	ACB	ACC	ACL	ACM	ADC	AGM	AGR	...	VSH	VSI	VTB	VTH	VTL	VTO	VTV	VXB	WCS
DTYYYYMMDD																				
2015-01-06	0.015	-0.007	0.000	0.013	-0.033	0.000	0.0	-0.022	0.000	0.000	...	0.007	-0.062	-0.025	0.020	0.0	0.048	0.004	0.0	-7.343e-02
2015-01-07	0.036	0.007	-0.009	0.013	-0.028	0.044	0.0	0.000	0.000	0.016	...	0.000	0.057	-0.009	0.000	0.0	0.012	0.013	0.0	-3.704e-02
2015-01-08	-0.021	0.007	0.038	0.006	0.000	0.000	0.0	0.000	-0.044	0.000	...	-0.015	-0.063	0.069	0.000	0.0	0.000	0.076	0.0	0.000e+00
2015-01-09	-0.021	-0.062	0.055	0.006	0.052	-0.008	0.0	0.032	0.036	0.016	...	0.007	0.000	0.032	0.019	0.0	0.023	0.000	0.0	-7.756e-04
2015-01-12	-0.029	0.059	-0.034	0.013	0.000	-0.008	0.0	0.000	0.018	-0.016	...	-0.007	-0.067	0.000	-0.050	0.0	-0.033	0.000	0.0	-1.463e-02

5 rows × 556 columns

Figure 4.2: Returns dataframe

Fast Fourier Transform

Clustering algorithms typically operate on feature vectors rather than raw time series data. Therefore, it's important to transform the pricing time series data into a suitable feature representation that captures the relevant characteristics for clustering. Among common approaches for representing time series data as feature vectors for clustering, we choose the Discrete Fourier Transform.

The Discrete Fourier Transform (DFT) is a mathematical transformation that converts a sequence of time-domain samples into its frequency-domain representation.

By analyzing the frequency-domain representation obtained from the DFT, we can identify the dominant frequencies present in the signal. This is particularly useful in various applications such as audio processing, image processing, communication systems, and most important to us, financial analysis.

In the context of our problem, the DFT can be utilized to analyze the frequency characteristics of price movements. It can help identify periodic patterns or cyclic behaviors in the data, which can be valuable for detecting trading opportunities or understanding market dynamics.

However, directly computing the DFT can be computationally expensive, especially

for large datasets. Which is why the Fast Fourier Transform (FFT) was invented.

The FFT[1] algorithm exploits the symmetry and periodicity properties of the DFT to significantly reduce the number of computations required. It divides the input sequence into smaller subproblems and recursively computes their DFTs. By utilizing this divide-and-conquer approach, the FFT algorithm achieves a complexity of $O(n \log n)$, where n is the number of samples.

The FFT works by representing the input signal as a combination of sinusoidal waves with different frequencies. It decomposes the signal into a sum of complex exponential functions, each representing a different frequency component. The resulting frequency-domain representation provides information about the amplitudes and phases of these frequency components.

```
from scipy.fft import fft

# Apply Fourier Transform to each column (stock) in the DataFrame
fft_features = pd.concat([pd.Series(np.abs(fft(returns[column].values)), name=column) for column in returns.columns], axis=1)

print(fft_features.shape)
fft_features.head()
```

```
(998, 556)
```

	AAA	AAM	ABT	ACB	ACC	ACL	ACM	ADC	AGM	AGR	...	VSH	VSI	VTB	VTH	VTL	VTO	VTV	VXB	WCS	WSS
0	1.011	0.608	0.340	1.212	0.191	1.426	1.352	0.815	0.626	0.219	...	0.624	1.680	1.367	0.258	1.074	0.349	0.636	0.538	0.302	0.195
1	1.437	0.228	0.559	0.611	0.410	1.289	0.717	0.568	0.316	1.349	...	0.094	0.502	0.154	0.167	1.235	0.278	0.838	1.048	0.534	0.581
2	0.787	0.349	0.473	0.642	0.596	1.080	0.711	0.724	0.483	1.184	...	0.272	0.376	0.574	0.772	1.367	0.745	0.139	0.670	0.102	1.285
3	1.170	0.376	0.344	0.765	0.548	1.356	1.767	0.630	0.557	0.735	...	0.224	1.222	0.160	0.395	1.318	0.611	0.492	0.806	0.271	0.496
4	1.082	0.273	0.307	0.604	0.227	0.957	0.964	0.254	0.233	0.214	...	0.171	0.675	0.716	0.901	1.165	0.377	0.830	0.976	0.296	0.298

5 rows × 556 columns

Figure 4.3: FFT dataframe [2]

4.2.2 Integrating Additional Features

Mean Returns

Returns are often annualized for comparison purposes, while a holding period return calculates the gain or loss during the entire period an investment was held.

$$MeanReturns = Mean(Returns_{daily}) * T$$

Volatility

One way to measure an asset's variation is to quantify the daily returns (percent move on a daily basis) of the asset. Historical volatility is based on historical prices and represents the degree of variability in the returns of an asset. This number is without a unit and is expressed as a percentage.

While variance captures the dispersion of returns around the mean of an asset in general, volatility is a measure of that variance bounded by a specific period of time. Thus, we can report daily volatility, weekly, monthly, or annualized volatility. It is, therefore, useful to think of volatility as the annualized standard deviation.

$$\text{vol} = \sigma\sqrt{T}$$

where:

$$\begin{aligned} \text{vol} &: \text{volatility over some interval time} \\ \sigma &: \text{standard deviation of returns} \\ T &: \text{number of periods in the time horizon} \end{aligned}$$

```
#Calculate mean returns and create a data frame
mean_returns = pricing.pct_change().mean()*998
mean_returns = pd.DataFrame(mean_returns)
mean_returns.columns = ['mean returns']

#Calculate the volatility
mean_returns['volatility'] = pricing.pct_change().std()*np.sqrt(998)

print(mean_returns.shape)
mean_returns.head()
```

```
(556, 2)
```

Ticker	mean returns	volatility
AAA	1.011	0.748
AAM	0.608	0.898
ABT	0.340	0.710
ACB	1.212	0.601
ACC	0.191	0.727

Figure 4.4: The Mean Returns and Volatility Dataframe

Industry

Business analysts often classify stocks into industry groups (sector) primarily based

on similarity in revenue lines. Stocks of similar industries should be related (correlated) in the future. Considering this, we find it useful to crawl industrial sector data from MarketWatch Website, a free and open-source site for investing news and info.

Market Capitalization

Market capitalization represents the total value of a company's outstanding shares in the market, providing a measure of its size and relative importance in the market. We believe that stocks with similar market capitalization tend to exhibit similar characteristics and behaviors. To incorporate this information, we gather market capitalization data from a reliable and widely-used source, such as the VietstockFinance Website.

```
# Importing the sector
sector = pd.read_excel('./data/sector.xlsx', index_col=0)
print(sector.head())

# Importing the marketcap
marketcap = pd.read_excel('./data/marketcap.xlsx', index_col=0)
print(marketcap.head())
```

```
                          Sector
Ticker
AAA              industrial_goods
AAM                   agriculture
AAT                 consumer_goods
AAV       real_estate_construction
ABS      basic_materials_resources
         Marketcap
Ticker
AAA      Medium Cap
AAM       Micro Cap
AAT       Small Cap
AAV       Micro Cap
ABR       Micro Cap
```

Figure 4.5: The Industrial Sector and Market Capitalization Dataframe

4.2.3 Data Handling

Scale Numerical Data

As we know, most of the machine learning models learn from the data by the time the learning model maps the data points from input to output. And the distribution of the data points can be different for every feature of the data. Larger differences between the data points of input variables increase the uncertainty in the results of the model.

The machine learning models provide weights to the input variables according to their data points and inferences for output. In that case, if the difference between the data points is so high, the model will need to provide the larger weight to the points and in final results, the model with a large weight value is often unstable. This means the model can produce poor results or can perform poorly during learning.

So if the data in any conditions has data points far from each other, scaling is a technique to make them closer to each other or in simpler words, we can say that the scaling is used for making data points generalized so that the distance between them will be lower.

Normalization and Standardization are the two main methods for the scaling of the data. Which are widely used in the algorithms where scaling is required. In our case, $sklearn\ StandardScaler$ [3] is prefered as it is more useful in clustering problem and when the data have negative values.

```
# Create a merged numerical value dataframe
num_data = pd.concat([fft_features_transpose, mean_returns], axis=1)
print(num_data.shape)
num_data.head()
```

```
(556, 1000)
```

	0	1	2	3	4	5	6	7	8	9	...	990	991	992	993	994	995	996	997	mean returns	volatility
AAA	1.011	1.437	0.787	1.170	1.082	0.535	0.331	0.180	0.677	0.239	...	0.677	0.180	0.331	0.535	1.082	1.170	0.787	1.437	1.011	0.748
AAM	0.608	0.228	0.349	0.376	0.273	0.147	0.521	0.185	0.107	0.104	...	0.107	0.185	0.521	0.147	0.273	0.376	0.349	0.228	0.608	0.898
ABT	0.340	0.559	0.473	0.344	0.307	0.573	0.565	0.429	0.137	0.476	...	0.137	0.429	0.565	0.573	0.307	0.344	0.473	0.559	0.340	0.710
ACB	1.212	0.611	0.642	0.765	0.604	0.452	0.299	0.730	0.241	0.661	...	0.241	0.730	0.299	0.452	0.604	0.765	0.642	0.611	1.212	0.601
ACC	0.191	0.410	0.596	0.548	0.227	0.174	0.512	0.094	0.449	0.470	...	0.449	0.094	0.512	0.174	0.227	0.548	0.596	0.410	0.191	0.727

5 rows × 1000 columns

Figure 4.6: The Merged Numerical Dataframe

```
from sklearn import preprocessing
scaled_data = preprocessing.StandardScaler().fit_transform(num_data)

scaled_data_df = pd.DataFrame(scaled_data, columns=[f'SS{i+1}' for i in range(1000)], index = mean_returns.index)
print(scaled_data_df.shape)
scaled_data_df.head()
```

```
(556, 1000)
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureWarning: Feature names only support names th
at are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
C:\Users\Admin\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureWarning: Feature names only support names th
at are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

	SS1	SS2	SS3	SS4	SS5	SS6	SS7	SS8	SS9	SS10	...	SS991	SS992	SS993	SS994	SS995	SS996	SS997	SS998	SS999	SS1
Ticker																					
AAA	-0.057	-0.050	-0.058	-0.053	-0.054	-0.062	-0.064	-0.066	-0.059	-0.066	...	-0.059	-0.066	-0.064	-0.062	-0.054	-0.053	-0.058	-0.050	-0.055	-0.
AAM	-0.063	-0.066	-0.064	-0.064	-0.065	-0.067	-0.062	-0.066	-0.067	-0.067	...	-0.067	-0.066	-0.062	-0.067	-0.065	-0.064	-0.064	-0.066	-0.061	-0.
ABT	-0.066	-0.062	-0.062	-0.064	-0.064	-0.061	-0.061	-0.063	-0.067	-0.062	...	-0.067	-0.063	-0.061	-0.061	-0.064	-0.064	-0.062	-0.062	-0.064	-0.
ACB	-0.055	-0.061	-0.060	-0.058	-0.060	-0.063	-0.065	-0.059	-0.065	-0.060	...	-0.065	-0.059	-0.065	-0.063	-0.060	-0.058	-0.060	-0.061	-0.052	-0.
ACC	-0.068	-0.064	-0.060	-0.061	-0.065	-0.067	-0.062	-0.067	-0.062	-0.063	...	-0.062	-0.067	-0.062	-0.067	-0.065	-0.061	-0.060	-0.064	-0.066	-0.

5 rows × 1000 columns

Figure 4.7: The Scaled Numerical Dataframe

Encode Categorical Data

Most Machine Learning algorithms cannot work with categorical data and needs to be converted into numerical data. Our approach to this is using one-hot encoding [4], a technique used to represent categorical variables as numerical values in a machine learning model.

```
# Integrate the two categorical features to the dataframe
merged_df = pd.merge(scaled_data_df, sector, on='Ticker', how = 'left').merge(marketcap, on='Ticker', how = 'left')
print(merged_df.shape)
merged_df.head()
```

```
(556, 1002)
```

SS3	SS4	SS5	SS6	SS7	SS8	SS9	SS10	...	SS993	SS994	SS995	SS996	SS997	SS998	SS999	SS1000	Sector	Marketcap
-0.058	-0.053	-0.054	-0.062	-0.064	-0.066	-0.059	-0.066	...	-0.064	-0.062	-0.054	-0.053	-0.058	-0.050	-0.055	-0.061	industrial_goods	Medium Cap
-0.064	-0.064	-0.065	-0.067	-0.062	-0.066	-0.067	-0.067	...	-0.062	-0.067	-0.065	-0.064	-0.064	-0.066	-0.061	-0.059	agriculture	Micro Cap
-0.062	-0.064	-0.064	-0.061	-0.061	-0.063	-0.067	-0.062	...	-0.061	-0.061	-0.064	-0.064	-0.062	-0.062	-0.064	-0.062	consumer_goods	Small Cap
-0.060	-0.058	-0.060	-0.063	-0.065	-0.059	-0.065	-0.060	...	-0.065	-0.063	-0.060	-0.058	-0.060	-0.061	-0.052	-0.063	financial_services	Large Cap
-0.060	-0.061	-0.065	-0.067	-0.062	-0.067	-0.062	-0.063	...	-0.062	-0.067	-0.065	-0.061	-0.060	-0.064	-0.066	-0.062	real_estate_construction	Small Cap

Figure 4.8: The Merged Numerical and Unprocessed Categorical Dataframe

```
# Drop stocks with empty Sector and/or Marketcap
completed_df = merged_df.dropna(subset=['Sector', 'Marketcap'])
print(completed_df.shape)
completed_df.head()
```

```
(517, 1002)
```

Ticker	SS1	SS2	SS3	SS4	SS5	SS6	SS7	SS8	SS9	SS10	...	SS993	SS994	SS995	SS996	SS997	SS998	SS999	SS1000	
AAA	-0.057	-0.050	-0.058	-0.053	-0.054	-0.062	-0.064	-0.066	-0.059	-0.066	...	-0.064	-0.062	-0.054	-0.053	-0.058	-0.050	-0.055	-0.061	indu
AAM	-0.063	-0.066	-0.064	-0.064	-0.065	-0.067	-0.062	-0.066	-0.067	-0.067	...	-0.062	-0.067	-0.065	-0.064	-0.064	-0.066	-0.061	-0.059	
ABT	-0.066	-0.062	-0.062	-0.064	-0.064	-0.061	-0.061	-0.063	-0.067	-0.062	...	-0.061	-0.061	-0.064	-0.064	-0.062	-0.062	-0.064	-0.062	cons
ACB	-0.055	-0.061	-0.060	-0.058	-0.060	-0.063	-0.065	-0.059	-0.065	-0.060	...	-0.065	-0.063	-0.060	-0.058	-0.060	-0.061	-0.052	-0.063	finan
ACC	-0.068	-0.064	-0.060	-0.061	-0.065	-0.067	-0.062	-0.067	-0.062	-0.063	...	-0.062	-0.067	-0.065	-0.061	-0.060	-0.064	-0.066	-0.062	real_estate_

Figure 4.9: The Completed Numerical and Unprocessed Categorical Dataframe After Dropping Empty Record

```
# One hot encoded the Sector and Marketcap
X = pd.get_dummies(completed_df, columns=['Sector', 'Marketcap'])
print(X.shape)
X.head()
```

```
(517, 1020)
```

ctor_retail_wholesale	Sector_technology	Sector_transportation_logistics	Sector_utilities	Marketcap_Large Cap	Marketcap_Medium Cap	Marketcap_Micro Cap	Marketcap_Small Cap
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1

Figure 4.10: The One Hot Encoded Dataframe

4.3 Clustering

4.3.1 K-Means Clustering

Algorithm Details

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning data into distinct groups or clusters. The goal of k-means clustering is to assign each data point to one of K clusters, where K is a user-defined parameter representing the number of clusters. The algorithm's steps include:

1. Initialization: Select K initial cluster centers randomly or based on some heuristic. These cluster centers are often referred to as centroids.
2. Assignment: Assign each data point to the nearest centroid based on a distance metric, typically the Euclidean distance. Each data point belongs to the cluster whose centroid it is closest to.
3. Update: Recalculate the centroids of each cluster by taking the mean of all the data points assigned to that cluster. The centroid represents the center or average position of the data points in that cluster.
4. Iteration: Repeat the assignment and update steps iteratively until convergence. Convergence occurs when the centroids no longer change significantly, or a maximum number of iterations is reached.
5. Result: The algorithm outputs the final set of cluster assignments and the corresponding centroids.

The k-means algorithm aims to minimize the within-cluster sum of squares, also known as inertia or the sum of squared distances between data points and their respective cluster centroids. It optimizes the clustering by iteratively adjusting the cluster assignments and updating the centroids to minimize this objective function.

One limitation of k-means clustering is that it is sensitive to the initial selection of centroids, which can lead to different cluster assignments and results. To mitigate this, it is common to run the algorithm multiple times with different initializations and select the clustering with the lowest inertia.

Hypermeter Tuning

To demonstrate the hyper meter tuning of K, we draw the elbow plot:

```
from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline

K = range(3,15)
distortions = []

#Fit the method
for k in K:
    kmeans = KMeans(n_clusters = k)
    kmeans.fit(X)
    distortions.append(kmeans.inertia_)

#Plot the results
fig = plt.figure(figsize= (15,5))
plt.plot(K, distortions, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Distortion')
plt.title('Elbow Method')
plt.grid(True)
plt.show()
```

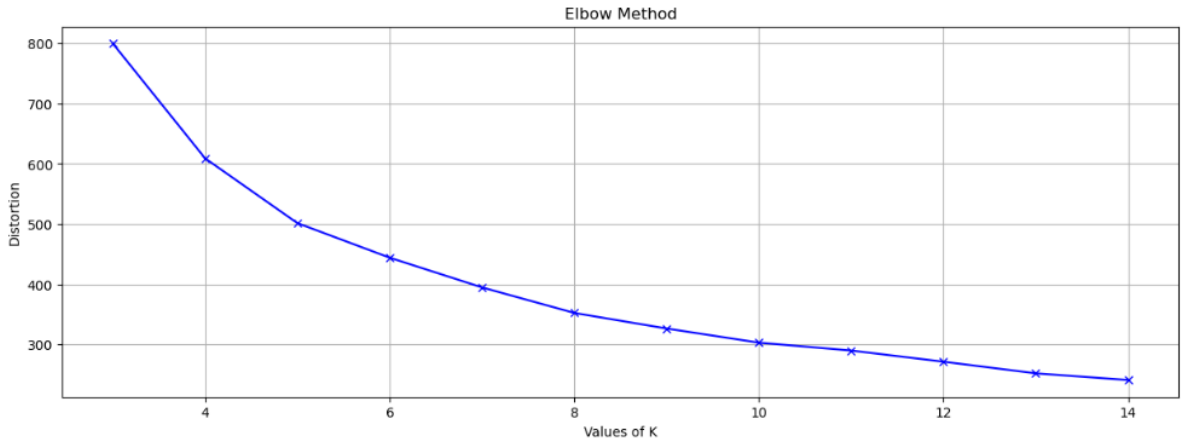

Figure 4.11: The Elbow Plot

The silhouette score is another metric commonly used to evaluate the quality of clustering. It measures how well each sample in a cluster is assigned to its own cluster compared to other clusters. A higher silhouette score indicates better-defined and well-separated clusters. The silhouette score is particularly useful when we want to evaluate the compactness and separation of clusters.

```python
from sklearn.metrics import silhouette_score

#For the silhouette method k needs to start from 2
K = range(2,15)
silhouettes = []

#Fit the method
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10, init='random')
    kmeans.fit(X)
    silhouettes.append(silhouette_score(X, kmeans.labels_))

#Plot the results
fig = plt.figure(figsize= (15,5))
plt.plot(K, silhouettes, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Silhouette score')
plt.title('Silhouette Method')
plt.grid(True)
plt.show()
```

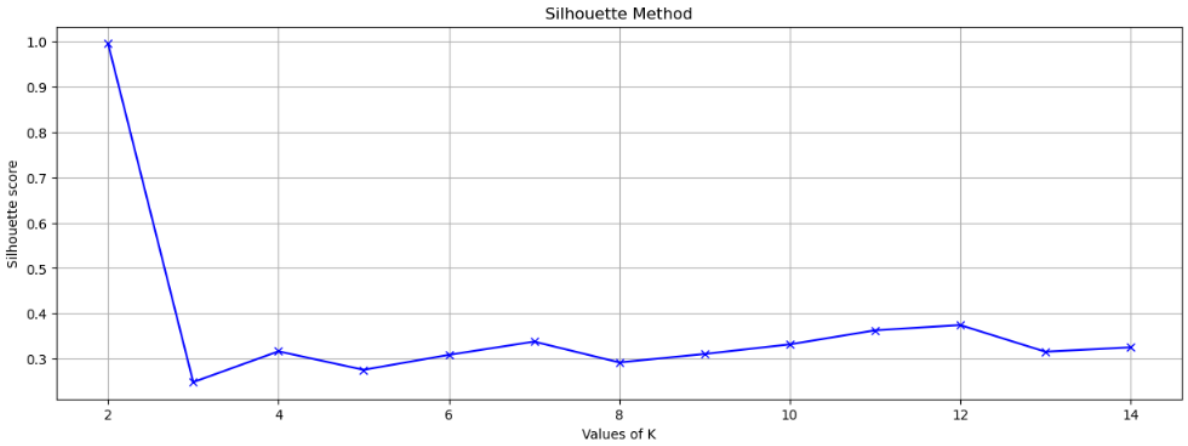

Figure 4.12: The Silhouette Plot

After evaluating the two plots, we decide to go with $k = 14$ as it balance out between finding out meaningful clusters and an acceptable distortion and silhouette score.

```python
c = 14
#Fit the model
k_means = KMeans(n_clusters=c)
k_means.fit(X)
prediction = k_means.predict(X)
labels = k_means.labels_
clustered = k_means.labels_
```

Figure 4.13: The K-Means Model

4.3.2 DBSCAN Clustering

Algorithm Details

Density-based spatial clustering of applications with noise ($DBSCAN$) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jorg Sander and Xiaowei Xu in 1996. It is a density-based clustering non-parametric algorithm which can be de-

scribed in the following steps:

1. Find the points in the ε neighborhood of every point, and identify the core points with more than $MinPts$ neighbors.

2. Find the connected components (Density Connected) of core points on the neighbor graph, ignoring all non-core points.

3. Assign each non-core point to a nearby cluster if the cluster is an ε neighbor, otherwise assign it to noise.

The advantages of DBSCAN include its ability to discover clusters of different shapes and handle noisy data effectively as points that are not density reachable from any core point are considered noise or outliers and they do not belong to any cluster. DBSCAN also does not require specifying the number of clusters in advance, making it suitable for exploratory data analysis. However, DBSCAN has some limitations, such as sensitivity to the choice of parameters (ε and $MinPts$) and difficulties in clustering datasets with varying densities.

Hypermeter Tuning

We set $MinPts = 3$ as we expected clusters to be as small as containing only 3 stocks.

To tune the ε parameter, we plot the silhouette graph with $MinPts = 3$ and ε varies from 0 to 5.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score

# Define the range of epsilon values to experiment with
eps_values = np.linspace(0.1, 5, num=50)

# Initialize lists to store epsilon values and silhouette scores
epsilon_list = []
score_list = []
```

```
# Iterate over each epsilon value
for eps in eps_values:
    # Create a new DBSCAN model with the current epsilon value
    clf = DBSCAN(eps=eps, min_samples=3)

    # Fit the model to the data
    clf.fit(X)

    # Evaluate the model using silhouette score
    labels = clf.labels_

    if len(set(labels)) > 1:  # Ensure at least two clusters are formed
        score = silhouette_score(X, labels)

        # Store the epsilon value and silhouette score
        epsilon_list.append(eps)
        score_list.append(score)

# Plot the silhouette scores against epsilon values
plt.plot(epsilon_list, score_list, marker='o')
plt.xlabel("Epsilon")
plt.ylabel("Silhouette Score")
plt.title("DBSCAN: Silhouette Score vs. Epsilon")
plt.grid(True)
plt.show()
```

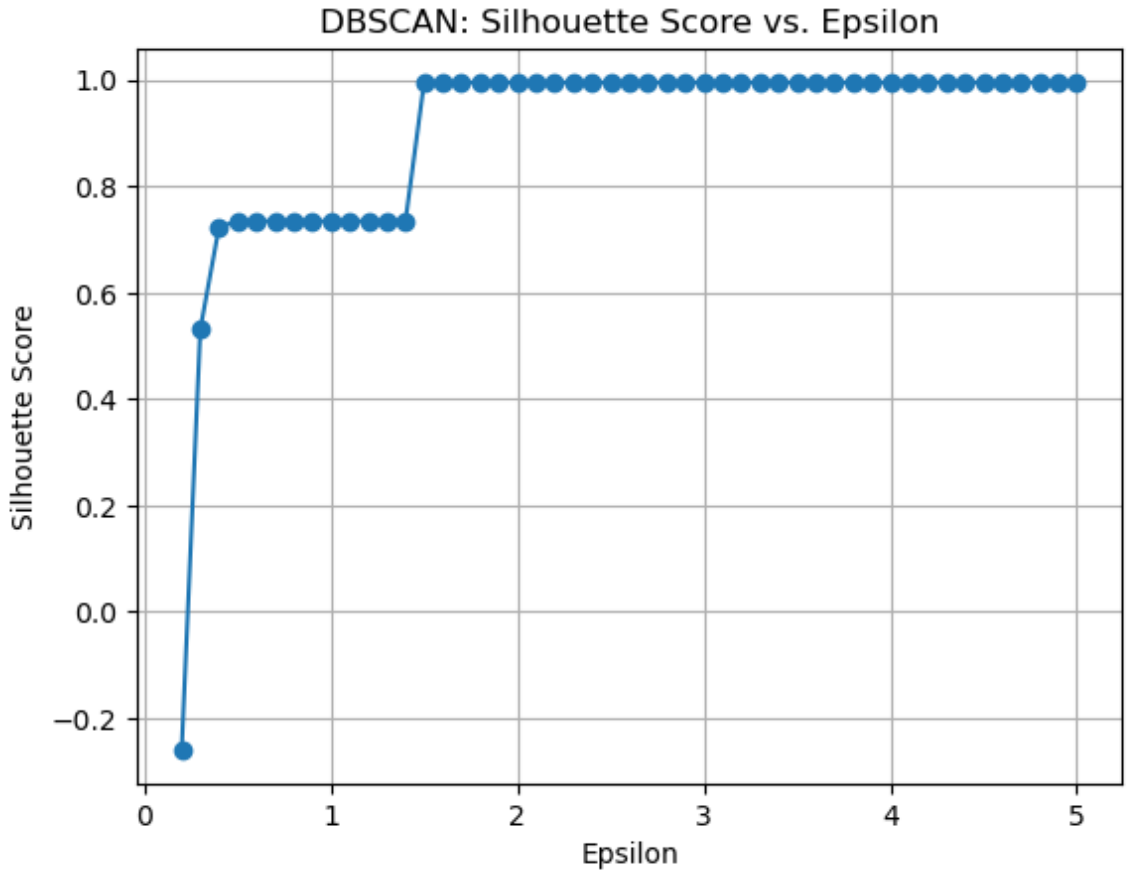

Figure 4.14: The Silhouette Plot

After evaluating the plot, we decide $\varepsilon = 1$ would be appropriate for our model as too large ε would result in every samples belonging to the same cluster and ultimately an unmeaningful clustering process.

```
clf = DBSCAN(eps=1.0, min_samples=3)
print(clf)

clf.fit(X)
labels = clf.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print ("\nClusters discovered: %d" % n_clusters_)

clustered = clf.labels_
```

Figure 4.15: The DBSCAN Model

4.3.3 Clustering Result Analysis

Cluster Evaluation

Many evaluation metrics come to mind when considering clustering result. Among them we choose: Silhouette Score, Calinski-Harabasz Index, Davies-Bouldin Index.

1. Silhouette Score [5]: The Silhouette Score measures the compactness and separation of clusters. It assigns a score to each data point based on its distance to other data points within its own cluster and to data points in neighboring clusters. The score ranges from -1 to 1, where a higher value indicates better-defined clusters. A Silhouette Score close to 1 suggests that the data point is well-matched to its own cluster and poorly matched to neighboring clusters.

2. Calinski-Harabasz Index [6]: The Calinski-Harabasz Index evaluates the ratio of between-cluster dispersion to within-cluster dispersion. It compares the dispersion between clusters with the dispersion within clusters. A higher index value indicates better-defined and more separated clusters.

3. Davies-Bouldin Index [7]: The Davies-Bouldin Index quantifies the similarity between clusters. It measures the average similarity between each cluster and its most similar cluster while considering their internal dispersion. The index ranges from 0 to infinity, where a lower value indicates better clustering. A value closer to 0 suggests well-separated and distinct clusters.

Table 4.1: Comparison of K-means and DBSCAN Clustering Results

Metric	K-means	DBSCAN
Silhouette Score	0.4566	0.7348
Calinski-Harabasz Index	88764.50	1.1090
Davies-Bouldin Index	0.9624	1.8308

Based on the provided results, DBSCAN seems to perform better than K-Means in terms of silhouette score, indicating well-separated clusters and compactness within each cluster. This can be explained as follows:

- K-means clustering is sensitive to the initial selection of centroids. The algorithm converges to a local optimum, meaning that the final clustering result can vary depending on the initial positions of the centroids. In some cases, different initializations can lead to different clustering outcomes, which makes the algorithm less reliable.

- The algorithm assumes that the clusters are spherical and of equal size. However, in the case of our dataset, clusters may have different shapes and sizes. This assumption can lead to suboptimal clustering results if the underlying clusters deviate significantly from the spherical shape or have different variances.

- K-means is not well-suited for handling outliers and noisy data points. Outliers can have a significant impact on the centroid calculation and can distort the clustering results. The algorithm assigns each point to the nearest centroid, which means that outliers or noise can be falsely assigned to a cluster, affecting the overall cluster structure.

Regarding the Calinski-Harabasz Index and Davies-Bouldin Index, eventhough they are both metrics commonly used to evaluate the quality of clustering algorithms, they have certain limitations when applied to DBSCAN, a density-based clustering algorithm. These indices may not be suitable for evaluating DBSCAN:

- The Calinski-Harabasz index measures the ratio of between-cluster dispersion to within-cluster dispersion. It assumes that clusters are well-separated and have similar sizes. However, DBSCAN does not generate clusters of equal sizes or assume a fixed number of clusters. The density-based nature of DBSCAN allows it to discover clusters of varying densities and shapes, which may not conform to the assumptions of the Calinski-Harabasz index.

- The Davies-Bouldin index calculates the average similarity between clusters based on their centroids and scatter. It assumes that clusters have convex shapes and that smaller values indicate better clustering. DBSCAN, on the other hand, can generate clusters of arbitrary shapes, including non-convex clusters. This violates the assumption of convexity made by the DB index.

Cluster Visualization

Once the data is clustered, we need a way to visualize the high dimensional data and its clusters into a two-dimensional graph. One approach to this visualization is using the nonlinear dimensionality technique known as *t-Distributed Stochastic Neighbor Embedding* $(t - SNE)$ [8].

```
from sklearn.manifold import TSNE
X_tsne = TSNE(init = "random", learning_rate=1000, perplexity=25, random_state=1337).fit_transform(X)
```

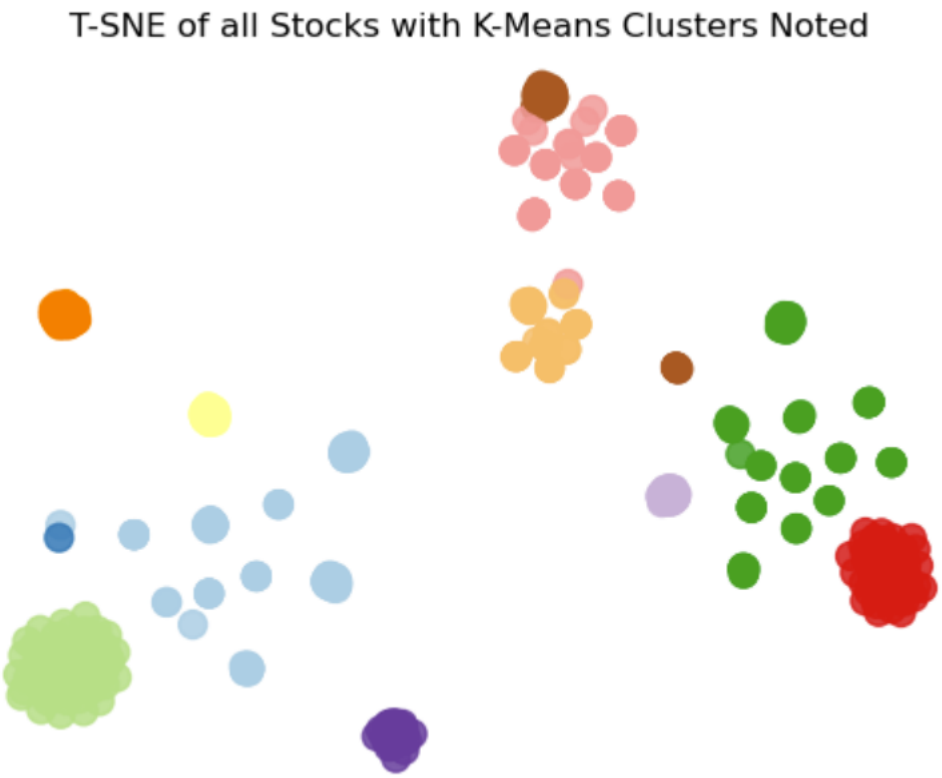

Figure 4.16: Visualization of K-Means Clusters

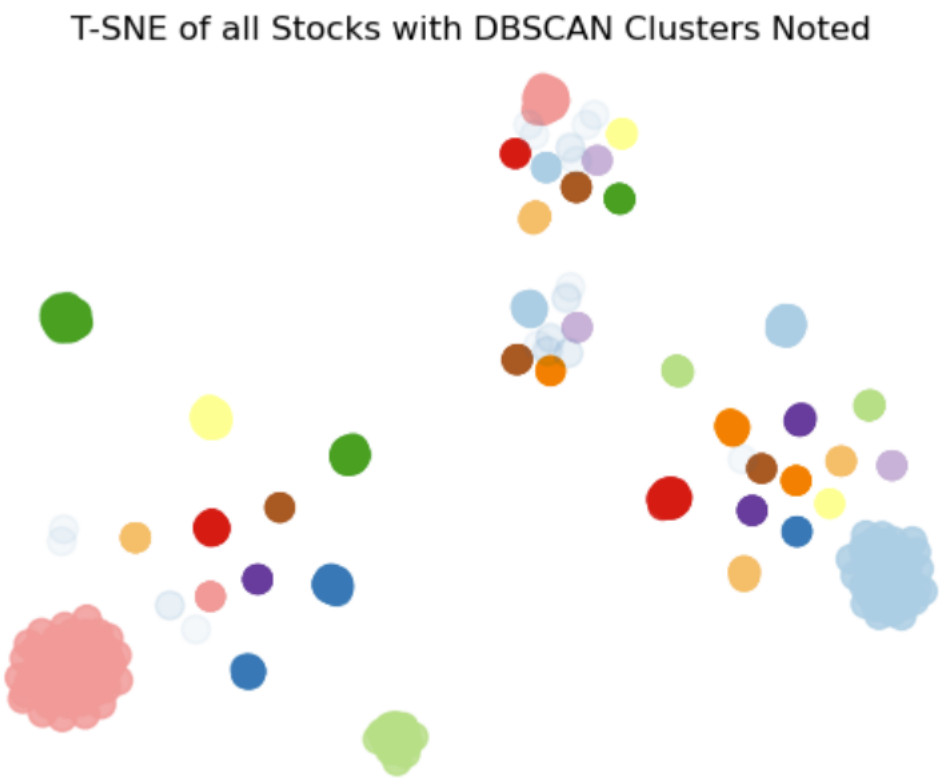

Figure 4.17: Visualization of DBSCAN Clusters

4.4 Pair Discovery

4.4.1 Cointegration Definition

A time series is stationary when the parameters of the underlying data do not change over time. While cointegration [9] implies that 2 time series $Y(t)$ and $X(t)$ share similar stochastic trends, and they never diverge too far from each other. The cointegrated variables exhibits a longterm equilibrium relationship defined by $Y(t) = \alpha + \beta X(t) + S(t)$, where $S(t)$ is the equilibrium error, which represents short-term deviations from the long-term relationship.

For pairs trading, the intuition is that if we find two stocks $Y(t)$ and $X(t)$ whose prices are cointegrated, then any short-term deviations from the spread mean, S, can be an opportunity to place trades accordingly, as we bet on the relationship to be mean reverting. Pairs are deemed as cointegrated when they aren't stationary and tend to move together.

4.4.2 The Engle-Granger two-step method

The statsmodels.tsa.stattools module in Python comes with $coint(S_1, S_2)$ [10], a handy function to verify cointegration between 2 time series S_1, S_2 , which implements the Engle-Granger two-step method.

The Engle-Granger two-step method [11] is a popular approach to test for cointegration between two time series. It involves two steps of estimation.

In the first step, the Engle-Granger method estimates the individual regressions of the two time series S_1 and S_2 on a constant and potentially some lagged values. This step is known as the estimation of the error correction model. The residuals obtained from these regressions are then saved for further analysis.

In the second step, the Engle-Granger method performs an augmented Dickey-Fuller (ADF) test on the residuals obtained from the first step. The ADF test is used to examine whether the residuals are stationary or contain unit roots. If the residuals are found to be stationary, it suggests that there is cointegration between the two time series.

By providing the two time series, $coint(S_1, S_2)$ returns the test statistic, p-value, and critical values for the ADF test on the residuals.

4.4.3 Pair Discovery

While trying to set up a function that finds the cointegrated pairs within a cluster, we make use of an already implemented function [12] from a platform known as Quantopian that's shutdown and not in use anymore.

```
from statsmodels.tsa.stattools import coint
def find_cointegrated_pairs(data, significance=0.01):
    # This function is from https://www.quantopian.com/lectures/introduction-to-pairs-trading
    n = data.shape[1]
    score_matrix = np.zeros((n, n))
    pvalue_matrix = np.ones((n, n))
    keys = data.keys()
    pairs = []
    for i in range(n):
        for j in range(i+1, n):
            S1 = data[keys[i]]
            S2 = data[keys[j]]
            result = coint(S1, S2)
            score = result[0]
            pvalue = result[1]
            score_matrix[i, j] = score
            pvalue_matrix[i, j] = pvalue
            if pvalue < significance:
                pairs.append((keys[i], keys[j]))
    return score_matrix, pvalue_matrix, pairs
```

Figure 4.18: Function to identify cointegrated pairs with 99% confidence level

4.4.4 Preview of Cointegrated Patterns

As a preview, among the pair that are deemed to be most optimar (having the highest confidence level), we choose 2 example pairs to plot the cointegrated patterns for visualization.

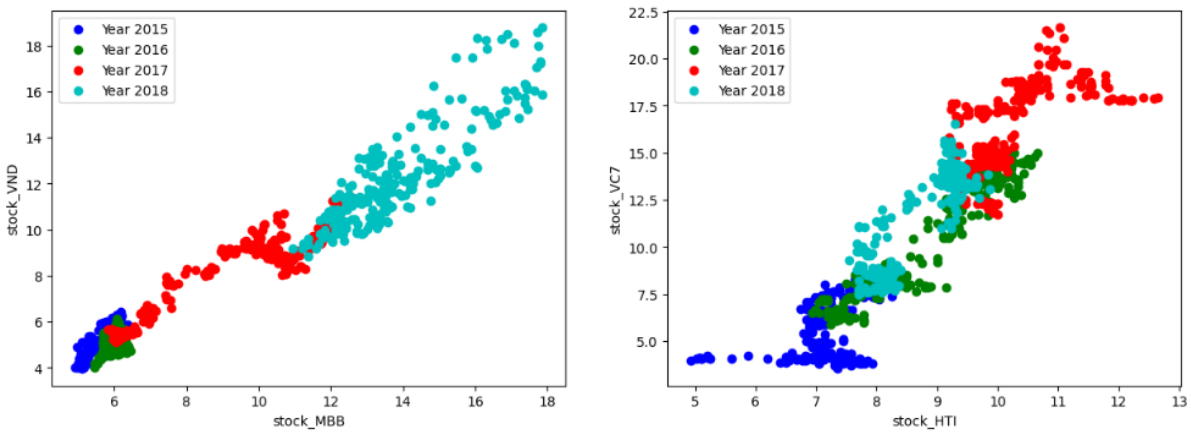

Figure 4.19: Scatter Plot of Pricing in 4 years used for Clustering

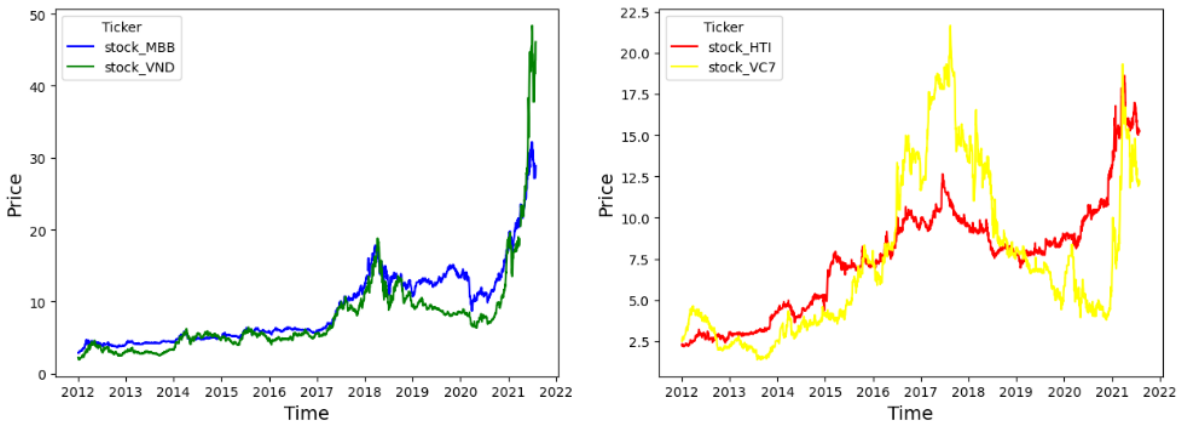

Figure 4.20: Pricing Plot in the whole time series

4.5 Link to project

Source to Github Repo: Pair-Trading-Analysis-Repo

CHAPTER 5. CONCLUSIONS

In conclusion, the machine learning pair trading project was successful in achieving its objective of identifying profitable trading pairs for using a pair trading strategy.

However, it's important to note that the strategy had some limitations and risks, including the potential for high transaction costs and the possibility of market disruptions or other unforeseen events. Therefore, further research and analysis would be necessary to determine the long-term viability of the strategy.

Further continuation on the project can include implementing the second step of the pair trading model: track return spread between each pair of stocks. When the spread is abnormally wide and deviation from the mean reaches some pre-determined threshold, the outperforming stock is sold short, and the under-performing stock is purchased. As soon as the spread converges back to its mean, the model liquidates both positions, resulting in a profit for the investor.

Overall, the project demonstrated the potential of using machine learning techniques to improve trading strategies and generate alpha in the financial markets. It also highlighted the importance of rigorous testing and risk management in developing successful trading strategies.

Bibliography

[1] Handling Stock Data using the Fast Fourier Transform. *Daryl Tng*. Date May 29, 2020. Available online: `https://medium.com/intuition/quantamental-approach-to-stock-trading-using-the-fourier-analysis-58f64792290`

[2] Fourier Transforms (`scipy.fft`). *Library Documentation*. scipy Version 1.10.1. Available online: `https://docs.scipy.org/doc/scipy/tutorial/fft.html`

[3] Standard Scaler (`sklearn.preprocessing.StandardScaler`). *Library Documentation*. scikit-learn Version 1.2.2. Available online: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

[4] One-Hot Encoder (`sklearn.preprocessing.OneHotEncoder`). *Library Documentation*. scikit-learn Version 1.2.2. Available online: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html`

[5] Silhouette Score (`sklearn.metrics.silhouette_score`). *Library Documentation*. scikit-learn Version 1.2.2. Available online: `https://scikit-learn.org/stable/modules/clustering.html#silhouette-coefficient`

[6] Calinski-Harabasz Index (`sklearn.metrics.calinski_harabasz_score`). *Library Documentation*. scikit-learn Version 1.2.2. Available online: `https://scikit-learn.org/stable/modules/clustering.html#calinski-harabasz-index`

[7] Davies-Bouldin Index (`sklearn.metrics.davies_bouldin_score`). *Library Documentation*. scikit-learn Version 1.2.2. Available online: `https://scikit-learn.org/stable/modules/clustering.html#davies-bouldin-index`

[8] t-distributed Stochastic Neighbor Embedding (t-SNE) (`sklearn.manifold.TSNE`). *Library Documentation*. scikit-learn Version 1.2.2. Available online: `https://scikit-learn.org/stable/modules/manifold.html#t-sne`

[9] Cointegration. *Niti Gupta*. Available online: https://www.wallstreetmojo.com/cointegration/

[10] Cointegration Test (statsmodels.tsa.stattools.coint). *Library Documentation*. statsmodels Version 0.15.0. Available online: https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.coint.html

[11] Cointegration Popular Methods [1/2]: The Engle-Granger Approach *Diego Barba*. Date Jun 16, 2022. Available online: https://towardsdatascience.com/cointegration-popular-methods-1-2-the-engle-granger-approach-82b6d270ddf2

[12] Cointegration Implementation. Now no longer available online: https://www.quantopian.com/lectures/introduction-to-pairs-trading