LUT University

Rodrigo Antonio Melisan Amancio da Matta

Pairs Trading on High-Frequency Data using Machine Learning

Thesis for the degree of Master of Science (Technology) in Business Analytics

Abstract

Author:	Rodrigo Antonio Melisan Amancio da Matta
Title:	Pairs Trading on High-Frequency Data using Machine Learning
University:	Lappeenranta-Lahti University of Technology
Faculty:	School of Engineering Science
Degree:	Master of Science (Tech.) in Business Analytics
Thesis:	73 pages, 9 tables, and 21 figures
Year:	2020
Supervisor:	Prof. Mikael Collan, D.Sc.
Examiner:	Assoc.Prof. Sheraz Ahmed, D.Sc.
Keywords:	pairs trading, machine learning, high-frequency data, stock market, algorithmic trading

Pairs Trading is a well-known statistical arbitrage strategy where a couple of equities which prices have co-moved in the past is expected to do so in the future. The rationale behind it is simple: at some entry point, that means, when stocks' prices diverge, sell short the stock which outperforms and buy long the underperforming stock. Afterward, liquidate the position when stocks' prices converge (exit point). Many approaches are available to first screen pairs of stocks, and second to perform the trade. This work used the Augmented Engle-Granger two-step cointegration test to screen pairs of stocks and focused on using machine learning algorithms to support the trade phase. A Recurrent Neural Networks was deployed to model and predict the Z-Score of the stocks' spread. Then a Deep Q-Learning Network was used to predict trade actions. Results showed that the strategy is profitable most of the time when not accounting trading costs. Loss of cointegration between stocks is another issue that affects profitability. According to the outcomes, the maximum value of the portfolios formed by each pair was always higher than the final value which impels the use of optimization for an exit rule to improve profitability especially when considering trading costs.

Acknowledgements

I would like to start by thanking my thesis supervisor Professor Mikael Collan from the School of Business and Management at LUT University. First for accepting me as a supervisee, second for the opportunity to collaborate with the department as a TA.

I would also like to acknowledge Associate Professor Sheraz Ahmed from the School of Business and Management at LUT University as the second examiner of this thesis. I am gratefully indebted to him for his valuable comments and inputs on this work.

In closing, I should express my sincere gratitude for the continued support and encouragement my family has offered me throughout my life, and during the research and development process of this thesis. Without them, this accomplishment was not possible. Thank you.

Rodrigo Antonio Melisan Amancio da Matta
July 2020
Lappeenranta, Finland

Contents

List of symbols and abbreviations

In the present work, variables and constants are denoted using *slanted style*, vectors and matrices are denoted using **bold regular style**, and abbreviations are denoted using regular style.

Latin letters

a	Software agent's action
C	Trading cost
D	Cumulative discounted reward
I	Rounded stock price
L	Loss function
N	Number of stocks
P	Price of a stock
q	Trade order
Q	Value function
R	Portfolio's return
S	Sharpe ratio
s	Environment's state
V	Portfolio's value
X	Total of trade execution

Greek letters

(Note: This is listing used Greek symbols in alphabetical order including names of symbols.)

γ	(gamma)	
θ	(theta)	
π	(pi)	
$\mathit{\Sigma}$	(capital sigma)	often used for sum without slanting: Σ
σ	(sigma)	

Superscripts

$*$	Optimum
t	Time
T	Transpose

Subscripts

1	First stock of the pair
2	Second stock of the pair
30	30 frequency bars
a	Software agent action
c	Long-term state
f	Risk-free
g	Main gate
h	Short-term state
i	Recursive index
o	Output gate
p	Portfolio
π	Decision policy
pk	Peak
t	Time
tr	Trough
x	Input

Abbreviations

A	Agilent Technologies, Inc.
ABBV	AbbVie Inc.
AI	Artificial Intelligence
AMRC	Ameresco Inc.
ARIMA	Autoregressive Integrated Moving Average
BABA	Alibaba Group
BIP	Brookfield Infrastructure Partners
BRT	BRT Apartments Corp.
CNN	Convolutional Neural Network
CVS	CVS Caremark
D	Dominion Energy Inc.
DDQN	Double Deep Q-Network
DIS	The Walt Disney Company
DNN	Deep Neural Network
DQN	Deep Q-Learning Network
ETF	Exchange-Traded Fund
FC	Fully Connected
GA	Genetic Algorithm
GBT	Gradient Boosted Tree

GB	Gigabyte
GE	General Electric
HD	Home Depot
HFD	High-Frequency Data
IBM	IBM Corp.
JNJ	Johnson & Johnson
LSTM	Long Short-term Memory
MAPE	Mean Absolute Percentage Error
MAVG	Moving Average
MDD	Maximum Drawdown
ML	Machine Learning
MLP	Multi-Layer Percepteron
MNIST	Modified National Institute of Standards and Technology
MRK	Merck Sharp and Dohme
MSE	Mean Squared Error
NASDAQ	National Association of Securities Dealers Automated Quotations
NRM	Negative Rewards Multiplier
OLS	Ordinary Least Squares
PCC	Pearson Correlation Coefficient
PDF	Probability Density Function
PG	Procter & Gamble Company
RAF	Random Forest
reLU	Rectified Linear Units
RL	Reinforcement Learning
RMSE	Root-mean-square Error
RNN	Recurrent Neural Network
RSI	Relative Strengh Index
SD	Standard Deviation
SLB	Schlumberger
SMA	Simple Moving Average
SQ	Square Inc.
SSD	Euclidean Squared Distance
SVM	Support Vector Machine
T	AT&T Inc.
tanh	Hyperbolic tangent
TAQ	Trade and Quote
TLS	Total Least Squares
XOM	ExxonMobil
WT	Wavelet Transformation

List of figures and tables

1 Introduction

Financial markets are considered to be efficient in a way that all relevant information is widely accessible. This makes it unfeasible to beat the market since no undervalued nor overvalued securities are supposed to be available. Nevertheless, practitioners and researchers try to find arbitrage opportunities in the hope there is a momentary imbalance on securities prices.

One of these attempts is Pairs Trading. Pairs Trading is a statistical arbitrage strategy involving two equities which prices have co-moved in the past and is expected to do so in the future. But, due to market volatility, their prices may start to diverge, which produces an entry point for trade but converges afterward providing an exit point. The rationale behind is simple: at some entry point, short the stock which outperforms and long the underperforming stock. Afterward, liquidate the position at the exit point, which means, when the stocks' prices converge.

1.1 Background

Pairs Trading was previously mentioned in the academic literature by Gatev et al. (1999) when practitioners started to use it and remained a profitable arbitrage strategy until 2006 (Krauss, 2016) when same authors published their seminal paper about Pairs Trading. After these publications, it is pacified that Pairs Trading might have two phases: (1) formation, and (2) trade.

1.1.1 Formation phase

This phase consists of screening and selecting pairs of stocks to use in the trade phase. There are mainly two methods to achieve it:

1. Metric approach: screening stocks using this method is based on some metric, for instance, the sum of Euclidean squared distance – SSD (Gatev et al., 2006) or the Pearson correlation coefficient – PCC (Do and Faff, 2010). The idea is to identify potential comoving stocks using these metrics, that means, the minimum value in the case of SSD and the maximum absolute value in the case of PCC.

2. Cointegration approach: here the idea is to apply some kind of cointegration test (Engle-Granger's, Johansen's, or else) on the stocks in consideration in order to screen the most promising co-movers.

1.1.2 **Trade phase**

This is probably the most challenging part of Pairs Trading. There are many approaches to define the exact point in time to enter and exit the trade. It is an optimization problem that could be tackled in different forms:

1. Threshold approach: initially proposed by Gatev et al., (2006), the trade should occur when price deviates by some threshold from the mean (like 2 standard deviations) for the entry point and converges again to mean crossing down the threshold (exit point).
2. Modeling approach: this path utilizes econometric modeling assuming that securities prices evolution is a mean-reverting process. By this assumption (and after modeling), it may be possible to identify quasi-optimal entry and exit points.
3. Machine learning – ML approach: again, using mathematical modeling to solve the entry/exit point problem, but this time based in data mining i.e., there is not the strict requirement of using a model that describes some kind of phenomena.

1.1.3 **Historical evolution**

As mentioned, Pairs Trading dates back to the nineties with the work of Gatev et al. (1999) where authors used the distance approach on U.S. equities daily data from 1962 to 1997 to derive a trading strategy using pairs of stocks hedging each other. A few years later Vidyamurthy (2004) developed a theoretical framework for Pairs Trading using a univariate cointegration approach. Then, various authors like Elliot et al. (2005) started to use time series methods to model stocks spread as mean-reversion processes in Pairs Trading applications. After, some papers like *Dynamic portfolio selection in arbitrage* by Jurek and Yang (2007) analytically defined the limits of a "stabilization region" in which traders could open positions counter to divergences of the spread and benefit from it. This was one of the first works on Pairs Trading using a stochastic control approach. More recently, other approaches in the deployment of Pairs Trading like principal component analysis (PCA), copulas, and machine learning have been researched and published like Avellaneda and Lee (2010), Liew and Wu (2013), and Huck (2009) respectively.

1.1.4 **Trading costs**

Previous works like Do and Faff (2012) and Bowen et al. (2010) reported that Pairs Trading is quite sensitive to trading costs, becoming even an unprofitable strategy. Trading costs appear from a myriad of sources like exchange fees, taxes, execution, slippage, latency, and so on. They may be marginal for long-term strategies but could increase considerably in a high-frequency setting.

1.2 Research question, motivation, objective, and delimitation

Due to the deluge of data that is produced nowadays and the availability of increasing computational power, ML methods have been revamped and became widely available. From this perspective and according to the evolution of Pairs Trading, it became a natural path to develop applications using ML to solve this sort of problem. Therefore, we propose to answer the following research question: *Can the current ML algorithms benefit from financial high-frequency data and produce feasible applications in Pairs Trading?*

This work is motivated by the characteristics of financial high-frequency data especially in fine-grain format (few seconds) which to the best of our knowledge is applied to Pairs Trading for the first time. The coupling of such high-frequency data with ML algorithms provides a promising approach to capture long-term market microstructure dependencies in time series and take advantage of it when deploying and backtesting a statistical arbitrage strategy like Pairs Trading.

The objective of this work is to use up-to-date ML algorithms to support a trading strategy based on a pair of stocks in a high-frequency setting.

This research is delimited by the use of recent advances in terms of ML algorithms. The data utilized to train, and test models is of high-frequency type, that means, intra-day orders placed during a certain time and collected from real stock exchanges (TAQ data).

To achieve the objective, in the formation phase we utilized the cointegration approach for screening and selecting pairs of stocks for further use in the trade phase. A Recurrent Neural Network (RNN) was then be deployed to model the future behavior of the selected stocks to support the second ML algorithm, Reinforcement Learning (RL) when a software agent was trained to take actions like entering or exiting a trade position. The advantage of using RNNs is their capability to retain long-term sequence dependencies like in time series. In various tasks including natural language processing, sentiment analysis, and medical self-diagnostics, RNNs have proven to be successful (Du et al., 2017). Since the RNN algorithm is based on a recurring approach, the model can be updated progressively, so that it can adapt to new data emerging over time. On the other hand, RL is today one of the most promising areas of ML besides not being a novel one[1]. RL breakthrough occurred seven years ago when researchers from a British startup called DeepMind proposed the use of Deep Neural Networks (DNN) in conjunction with Q-Learning, a widely used method among the available RL algorithms, to train a software agent to play Atari video games (Mnih et al.,2013).

Results indicated that the method is largely successful when trading costs were not incurred. Another finding was that the strategy is affected by the loss of cointegration between stocks during the trade phase. The maximum value of each pair's portfolios was always, according to the results, higher than the final value which encourages the use of optimization to improve profitability in particular when taking account of trading costs.

[1] In the late 1980s, trial and error, optimal control, and temporal-difference methods combined to produce the modern field of reinforcement learning (Sutton and Barto, 2018).

1.3 Organization of this work

This text is divided into chapters, each having its own elements, but comprising parts of a whole manuscript.

Chapter 2 performs a literature review on recent advances of ML in quantitative finance particularly related to forecasting, classification, reinforcement learning, and Pairs Trading.

Chapter 3 offers a theoretical background to understand what is behind of RNN and RL algorithms and expose the characteristics and properties of financial high-frequency data.

Chapter 4 deals with the implementation of the proposed methodology. Starting from the data pre-processing, finding pairs of stocks, selecting and preparing features to feed the models, defining a strategy, and how to backtest it.

Chapter 5 provides an analysis of the results based on common performance measures used in quantitative finance and also compares the proposed method with an alternative one.

Chapter 6 concludes with the findings and limitations of this research and proposes future work.

2 Literature review

In this chapter we discuss the recent research in Machine Learning applied to quantitative finance and in special to Pairs Trading.

2.1 Forecasting

Forecasting is a tool that uses historical data as input to predict the course of future patterns in an informed way. Bao et al. (2017) proposed a novel framework using wavelet transforms (WT), autoencoders, and an LSTM network for stock price forecasting. The WT was used to eliminate noise on the time series, after, autoencoders generated high-level features for predicting stock prices. Finally, the result was input into the LSTM to perform the forecasting. The data used was a low-frequency type (six stock indices) within a 6-years' timeframe. As for performance criteria they used MAPE, R (correlation coefficient), and Theil U. MAPE calculates error size, R is the linear correlation between two variables, and Theil U is a relative measurement of two variables difference. A comparison was made with three other models: a combination of WT and LSTM, an LSTM, and a conventional RNN. The performance of the proposed model was significantly higher than of the other three models. Nonetheless, because the framework proposes to predict one-step ahead or the closing price of stock indices in the next day, it would seem unfeasible for high-frequency trading that requires a method to predict several steps forward due to the extremely large number of observations.

Fischer and Krauss (2018) utilized an LSTM network to predict directional movement on S&P 500 stocks. They formulated the problem as a binary classification one. A stock is labeled as class 0 if its one-period return is smaller than the cross-sectional median return of all stocks. Class 1 is inputted to stocks in which one period return is larger or equal than the cross-sectional median return of all stocks. Then they deployed an LSTM network with 3 layers, 1 feature, and 240 timesteps as input. Cross-entropy was used as the objective function. For comparison, they utilized 3 more classification methods: random forest, deep neural network, and logistic regression. Results were presented in 4 forms: (1) returns prior and (2) after trading costs, (3) screening of top and flop stocks through the patterns derived with the intention to identify profitability sources, (4) returns derived from a simple trading strategy based on the findings. Daily mean returns using a set of 10 stocks and prior trading costs for the LSTM network was 0.46% while 0.43% for the random forest, 0.32% for the deep neural network, and 0.26% for the logistic regression. Metrics like standard deviation (a measure of risk), Sharpe ratio (return per unit of risk), and classification accuracy were also better for the LSTM network compared with the other methods (this rank varies when altering the numbers of stocks, but in 70% of the cases, LSTM was better). Using a simple trading strategy, that is, short short-term winners and buy short-term losers, and hold the position for one day, after identifying top and flops stocks was also profitable using the LSTM network: 0.23% per day before trading costs. One of the contributions of the paper was to fulfill the lack of a large-scale empirical application on financial time series prediction using LSTM networks which makes these networks widely used nowadays by researchers and practitioners. However, it is not clear if those results are reproducible using more sophisticated strategies like statistical arbitrage.

2.2 Classification

A very common application of Machine Learning in quantitative finance is to create models that predict buy and sell signals. Thanks to their performance in image recognition problems, numerous researches centered on the use of Convolutional Neural Network (CNN) based models. Sezer and Ozbayoglu (2018) developed a model using a CNN to label 2-D images of technical indicators as buy, sell, or hold depending on the shape of the original time series. They utilized 15 technical indicators, each with 15-days data which produced a 15 ×15 sized 2-D image. To build the images they used stock prices of Dow 30 and daily Exchange-Traded Fund (ETFs) with a 5-years period for training and 1-year for testing but applying a 1-year sliding. Labeling was done manually as follows: all daily close prices are labeled as “buy” if they are the bottom points in a sliding window, “sell” if they are the top points in a sliding window and the remainders as “hold”. In the end, each stock had 1,250 images for training and 250 images for testing. The deep CNN was built with 9 layers and a structure similar to the ones used with the MNIST database[2]. The performance criteria were model accuracy and financial evaluation. For the stock data (Dow 30) the total accuracy was 58% and for the ETF data, 62%. For comparability, the method was confronted with the other 3 trading strategies, one based on the Relative Strength Index (RSI), other on the Simple Moving Average (SMA), and a simple buy-and-hold. Two other prediction models were used for performance measuring: a Multi-layer Perceptron (MLP) and LSTM neural network. On average, the proposed framework outperformed the other strategies and models by a large margin when using stock data (Dow 30) in a period of 10 years. However, the standard deviation (as a risk metric) was smaller for the MPL model (the proposed model was the fourth-best). Similar results were obtained with the ETF data. Statistical significance tests were also favorable to the proposed model. At the time of publication, the work brought a new perspective on the use of image recognition algorithms in the realm of algorithm trading. However, the trading strategy used was very simple (long-only) and there is no indication if the model would perform well with more intricate strategies like statistical arbitrage ones. Also, by producing and labeling 2-D images from a predetermined number of days data introduce selection bias even with the proposed sliding.

Tsantekidis et al. (2017) utilized a CNN to predict mid-price trend on 5 stocks traded at NASDAQ Nordic exchange for 10 days and utilizing high-frequency data. The network was fed with 100 vectors of most frequent limit orders. Those vectors were composed with the 10 highest bid and 10 lowest ask orders. Each of those orders with price and volume, that means, 40 values for each vector. Since the short-term changes between prices are very small and noisy, they developed a filter using the mean of *k* previous and next mid-prices. With such, they were able to label the training dataset in 3 classes: (trending) upward, (trending) downward, and stationary. Performance criteria were: Cohen’s kappa which is the agreement between two raters which classify elements into mutually exclusive categories, mean recall, precision, and F1-score. Comparing the CNN

[2] The Modified National Institute of Standards and Technology (MNIST) database is a large handwritten database commonly used for training various imaging systems.

results with two other models, MLP and SVM, and with the prediction horizon (*k*) of 5, the CNN outperformed the other models. CNN had slightly better results with $k = 10$ and also outperformed MLP and SVM. But with a longer prediction horizon (k = 50), the MLP had better precision than the CNN (67.4% against 55.6%) which is quite surprising due to the use of a high-frequency data (with lots of market microstructure patterns) and the shallowness of the MLP used (1 hidden layer). Also, by using a filter which utilizes *k* next mid-prices introduces a look-ahead bias[3] in the model.

2.3 Reinforcement learning

The three branches of Machine Learning are reinforcement learning (RL) along with supervised and unsupervised learning. Instead of having labeled data like in supervised learning or finding hidden structures in the data in the case of unsupervised learning, reinforcement learning is based on the idea of learning through trial and error. Reinforcement learning has been used in various financial and trading applications including portfolio optimization and optimum execution of trades. Chen et al. (2018) proposed an agent-based RL system to mimic professional trading strategies. The system was composed of a CNN and a DNN. The CNN was used to build a pre-trained model in which parameters were transferred to the DNN afterward. The inputs of CNN were market conditions while outputs were buy/sell signals (thirteen at total) based on the strategies of experts. Those signals represent thirteen actions in the futures market, that is, buy *k* futures contracts ($k = 1,2,3,4,5,6$), sell *k* futures contracts, or stay neutral. They used Taiwan stock index futures tick data for about 23 trading days. The RL algorithm used was a policy gradient in which input is an environment state and the output is an action or probability. The reward policy was adjusted as a PDF of a normal distribution with 0 mean and a standard deviation of 0.5. So, if the agent's action is the same as an expert's strategy, it will get the probability of 0.79 as a reward, if not, 0.1079. The model has 3 parameters to adjust: training period, exploration number, and episodes. The learning period was how long the policy network can be updated. How many times the

[3] Look-ahead bias is the use of available future data, resulting in incorrect simulation results.

agent plays in the same period was the exploration number. At the beginning of each episode, states were reset, which means, a randomly chosen moment in the training period. The performance criterion was accuracy or the comparison between the model's output and the expert's strategies. Out-of-sample results using 360 seconds as the training period were favorable to the parameter combination: 10 as exploration number and 200 episodes resulting an accuracy of 0.7401. However, this model depends on the expertise of knowledgeable traders to label the training data which would make many practical applications unfeasible. Furthermore, only 1D-CNN was used and the strong capacity of CNN to manage multi-dimensional data was not used in the best way.

2.4 **Pairs Trading**

Pairs Trading is a statistical arbitrage strategy that deploys two stocks with a strong relationship where long positions are balanced with short positions. There are many approaches to form those pairs of stocks and to perform the trading strategy itself. With more computational power available and the overall willingness to use the ever-increasing amount of accessible data, Machine Learning poses itself as an alternative approach. Brim (2020) presented an RL framework to learn a Pairs Trading strategy for cointegrated stocks. In the study, he used a Double Deep Q-Network (DDQN) to predict stocks' spread trends and then take actions like long, short, or hold. The use of a DDQN was due to the aim of decorrelating training samples, reduce errors, and improve performance. The dataset used was daily prices during four years of 38 stocks from the S&P 500 index. To form stocks pairs it was applied two statistics tests on stocks' prices: a p-value below 0.05 for the Augmented Dick-Fuller test (to check for cointegration), and a ratio between standard deviation and mean over 0.5 (to have enough variance and generate trading signals). To make the agent more risk-averse during training a negative reward booster was used which increased any negative spread returns to substantially higher negative rewards. The number of features feeding the network was 10: the current spread of the pair, daily returns of the spread, spread mean for 5,7,10, 15 days, and the spread/spread mean ratio for the same time intervals. A spread/mean ratio at equilibrium is 1.0 which recommends a hold position while a spread/mean ratio of 1.05 would be high suggesting the spread value will decline and a short position should be taken. A 0.95 spread/mean ratio would be an indication that the spread value will rise, and a long position should take place. The performance criterion was the total cumulative returns de depending on the NRM value. The total cumulative returns decreased as NRM value increases (up to 1000). However, the lack of comparison with other models had compromised the performance evaluation of the proposed framework. Curiously, the author seemed more interested in the sensitivity analysis of the model with regards to the NRM value. Besides the mention of the possibility of using the framework with different timeframes and financial markets, there is no indication of how to deploy it.

Kim and Kim (2019) proposed a framework to optimize trade and stop-loss boundaries on Pairs Trading strategies. They used a Deep Q-Learning Network (DQN) to train a software agent to learn those boundaries using the spread between pair's stocks and to maximize the expected sum of discounted future profits. Spread was computed both using OLS and TLS. The reward system was set as follows: the agent receives a positive reward if the spread exceeds a trading threshold and reverses to the mean. However, the agent earns a negative reward if it hits the thresholds of stop-loss or exceeds a trading threshold and does not reverse to the mean. The data used was the daily adjusted price of 50 stocks (S&P 500) from 1990 to 2018. Performance criteria were profit, maximum drawdown, and Sharpe ratio. The length of windows sizes for forming pairs and trading was respectively 30/15 days which was selected from the performance results of six possible options during model training. According to the authors, this was reflected by the number of closed positions compared to the number of open positions since a closed position means that the spread exceeds a trading threshold and reversed to the mean, i.e., it was profitable. Also, results with spread computed using TLS performed better than OLS. The proposed model was compared with six fixed trade and stop-loss boundaries models and, on average, performed better during the training stage in terms of profit and maximum drawdown, but not the Sharpe ratio. Using out-of-sample data, the proposed model performed better in terms of profit but the best results for maximum drawdown and Sharpe ration varied depending on the trading pair and comparison model. According to the authors, if they add Sharpe ratio (along with profit) in the objective function, they would get a more optimized system. The presented framework poses an interesting approach when setting a dynamic boundary for trading and stop-loss, however, it is not clear how the framework would perform using different kinds of datasets in different frequencies. For instance, it may be hard to implement a reward system like the proposed one, where several timesteps are needed to compute agent's reward, in a high-frequency setting and when the action space (with predefined boundaries) is large.

DNNs, gradient boosted trees (GBTs), random forests (RAFs), and certain ensembles of these techniques were implemented and analyzed for statistical arbitrage by Krauss et al. (2017). S&P 500 daily data from 1992 to 2015 was used in their work. They split the data set into 23 sub-sets (1,000 days each), having 750 days for training and 250 days for testing. As features, they used simple returns computed on the first 20 days, then in a multi-period covering the following 11 months. In total, the number of features was 31. The training task was to classify stocks according to their one-period return, which means if such return was larger than the cross-sectional median of all stocks or not. For such, they used a DNN with 5 layers including the input and output layers in a 31-31-10-5-2 configuration, that means, the input layer with 31 neurons, the first hidden-layer also with 31 neurons, and so on. For the GBT they set the number of trees to 100 and for the RAF 1000 trees. The ensemble learning part was composed by three models: (1) using the three previous mentioned models equally weighted, (2) also using the three previous models weighted according to the Gini index, and (3) the same three models which weights were based on a rank using the Gini index and the training periods. After training, it was possible to compute the probability of each stock to outperform the cross-sectional median and create a rank using k tops and k flops performers where k varied from 1 to 250. The trading strategy used was simply long the k top stocks and short the k flop stocks. Best results in terms of daily mean return, standard deviation, and directional accuracy were achieved with $k = 10$. Comparing the ensemble models, the equally-weighted one performed better. It also had the best results compared to the other models in terms of daily mean return, 0.0045 before transaction costs. The second best was RAF

(0.0043), followed by GBT (0.0037) and the DNN (0.0033). However, the maximum daily return was achieved by the DNN with 0.5474. The lower standard deviation (or volatility) was achieved by the RAF model. The best result for the MDD was also for the RAF model which made it the lowest risk option. However, part of the study was treated as a classification task, but some important measures like accuracy, specificity, sensitivity, precision, and recall were missing. Also, the use of 31 features incurred the missing of 240 data points just to compute them. The authors computed the importance of each feature for every model and founded out that simple returns regarding the last 4 days had the highest relative importance among all. Nonetheless, they did not use such findings to perform a feature selection. Additionally, it is not clear if that momentum-based strategy would be feasible with high-frequency data or with pairs of stocks (*k* =1).

Fallahpour et al. (2016) employed reinforcement learning to optimize Pairs Trading strategy parameters like formation and trade phase duration, and trade and stop-loss boundaries. They used intraday prices of the US equity market from June 2015 to January 2016. In the formation phase, they utilized the Johansen's cointegration test to identify potential pairs, but first, they segregated the stocks by industry, turnover rates, and size of deals. In the end, 14 pairs were identified. They sampled from data every minute and used the close price as the one to represent stock's price at each timestamp. Data set was then split into training and test sub-sets in a 75/25 proportion. The standardized spread of each pair was used as the input for the RL model which utilized the N-arm bandit algorithm. Thus, the software agent was supposed to learn the optimum duration for pairs' formation and trade phases, plus the thresholds to enter on a trade and stop-loss. Sortino ratio was used as the value function to be maximized according to the software agent actions. Parameters' space was discretized, so the formation phase duration varied from 60 to 600 minutes (with 5 minutes steps), trade phase duration from 5 to 120 minutes (with 5 minutes steps), trade threshold from 0 to 3 (with 0.5 steps), and stop-loss threshold from 0 to 5 (also with 0.5 steps). As for the learning policy, the ε-greedy option was utilized. By virtue of comparison, they performed a grid search and built a base case with the best values for the previous four parameters. Results showed that the proposed method outperformed the base case by far in practically all performance measures like returns (in and out of sample, annualized, and per trade), downside volatility, and Sortino rate. Only two pairs had worse results than the base case in terms of downside volatility. However, an important performance measure like daily returns was missing in their study (especially when using intraday data). Also, transaction costs were not computed to gauge the strategy's profitability. From the reinforcement learning point of view experimenting only 100 states from a universe of 149,040 possible states during the training is quite a modest setup which could have a significant impact on learning.

Cryptocurrencies are also prone to statistical arbitrage strategies. Fischer et al. (2019) studied how an RAF model would perform with portfolios of cryptocurrencies in a high-frequency setting. The data set was composed of minute-based prices and volume from January 2018 to September 2018 of 40 cryptocurrencies with a high market capitalization and in 6 exchanges. Two-thirds of data points were used for training and the remainder for testing. Features were composed of simple returns in a multi-period fashion: from 20 to 120 minutes (with 20 minutes steps), and from 240 to 1440 minutes (with $120k$ minutes steps where $k = \{2,3, \ldots,12\}$). In total, there were 17 features as input for the model. The RAF model itself had 1000 trees with a maximum depth of 15 branches. The goal was to classify those cryptocurrencies in a binary way: if the return over the next 120 minutes (after prediction) was at or above the cross-sectional median of all cryptocurrencies or not. More than a label, each cryptocurrency had a probability to outperform the cross-sectional median which made it possible to rank them. The trading strategy was simply long the tops and short the flops (3 each). Nonetheless, there were some execution conditions: (1) trade order was made one timestep after receiving the trading signal, (2) selected cryptocurrencies must have at least one trade at that point in time, (3) after opening a position it is closed automatically 120 minutes after, (4) a new portfolio is opened every minute t where $t = \{1,2,3,\ldots,120\}$ if condition 2 holds, and (5) trading costs were assumed as 15 bps (half turn). Condition 1 was set to eliminate the bid-ask bounce while condition 4 was to avoid starting point bias. As a benchmark, the authors deployed a logistic regression model. Results showed an advantage to the RAF model in terms of average return. Standard error, win ratio, and the standard deviation was better for the logistic regression. There was a draw between the models when considering maximum returns. After aggregation, daily returns were compared with a simple buy-and-hold strategy for Bitcoin and the general market. The RAF model still had the best result for the average return and win ratio followed by logistic regression. The Bitcoin strategy had the best result for the return's standard deviation and the market for the maximum return. The authors also analyzed the importance of each feature and found that returns over the past 20, 40, and 60 minutes had more impact in the predictions. Yet, they did not take advantage of this finding to perform a feature selection to avoid the curse of dimensionality especially present when using computation demanding models. Also, the decision to hold 120 portfolios in parallel to avoid starting point bias does not seem feasible in practical applications considering the limits of arbitrage in immature markets.

Huang et al. (2015) suggested a way of creating robust models to address the financial application's complex characteristics. They deployed a genetic algorithm (GA) model to optimize Pairs Trading strategy parameters like the moving average, the Bollinger bands, and the stock weighting coefficients. GA's chromosome was composed of four parts: the period parameter n of the moving average, the x and y of the Bollinger bands that governs the multiples of the standard deviations of the moving average to define entry and exit points, and the set of weighting coefficients β[4]. The GA model was set to use a binary tournament selection, one-point crossover, and mutation rates of 0.7 and 0.005. The fitness function of the chromosome was the annualized return of the portfolio. They used a generalized approach that uses more than two stocks to compute the spread of a synthetic asset and used it to produce trading signals. For instance, short the spread if it gets x standard deviations above its mean value. If the spread gets x standard deviations below its mean value, long the spread. In any case, the position is closed if the spread gets closer than y standard deviations to its mean. Two portfolios were built with a predefined number of stocks (10) traded on the Taiwan Stock Exchange. One with companies of the

[4] The β coefficient defines the quantity of each stock in the portfolio.

semiconductor industry and another with the largest capitalization companies in various industrial segments. Daily returns from the years 2003 to 2012 were used as input of the model. The output was the optimized set of parameters for trading. They divided the data set into training and testing sub-sets in a dynamic way depending on which quarter was used. As a benchmark, they utilized a simple buy-and-hold strategy using an equally weighted portfolio and during the same period. From the 39 quarters tested for the first portfolio, the annualized return using the GA was better than the benchmark on 30 occasions with values varying from 1.32% to 12.68%. For the second portfolio, results showed that the GA performance was better than the benchmark on 29 occasions. The precision of the method was also computed in a manner that if the GA outperformed the benchmark during the training and also during the testing, it was considered as a true positive. But if the GA outperformed the benchmark during the training, and not during testing, the case was labeled as a false positive. So, the precision was 0.7692 for the first portfolio and 0.7180 for the second portfolio. However, the dynamic method of splitting the data set ended up with cases with very few data points for training and much more for testing, and vice-versa (only a few data points for testing and much more for training) which is not a good practice. Also, the authors assumed that all processes were mean reverted and did not hedge the trading positions when they long (or short) the spread. This is quite problematic especially when there is no stop-loss rule.

3 Theoretical Background

With the abundance of produced data and the availability of computation power, Machine Learning methods have become pervasive in many industries. In this chapter, we explore two of them, namely, Recurrent Neural Networks and Reinforcement Learning which have been used with enthusiasm by AI practitioners to tackle a myriad of problems. After, we delve into the concept of High-Frequency Data, what it is, how is it gathered, organized, and displayed, its worthiness, and how trading within stock markets was transformed by its use.

3.1 Recurrent Neural Networks

RNN[5] is a type of neural network adequate to model time series and perform predictions. They can be deployed to analyze data from time series, like stock prices, and indicate when to purchase or sell. What differentiates an RNN from a feedforward neural network is a backward connection.

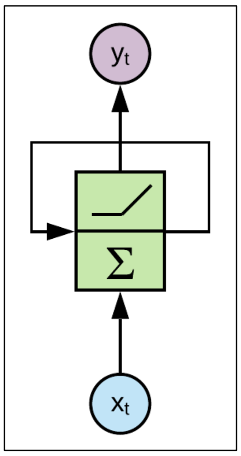


Figure 3.1: Recurrent neuron

The simplest RNN possible consists of a neuron receiving an input that produces an output and returns that output to itself like in Figure 3.1.

[5] The term "recurrent network" was explicitly mentioned first time in *Learning internal representations by error propagation* by Rumelhart et al. (1985).

At each timestep, this recurrent neuron receives the inputs x_t and also its outputs from previous timestep y_{t-1}. For better visualization, we can "unroll" this recurrent neuron as depicted in Figure 3.2.

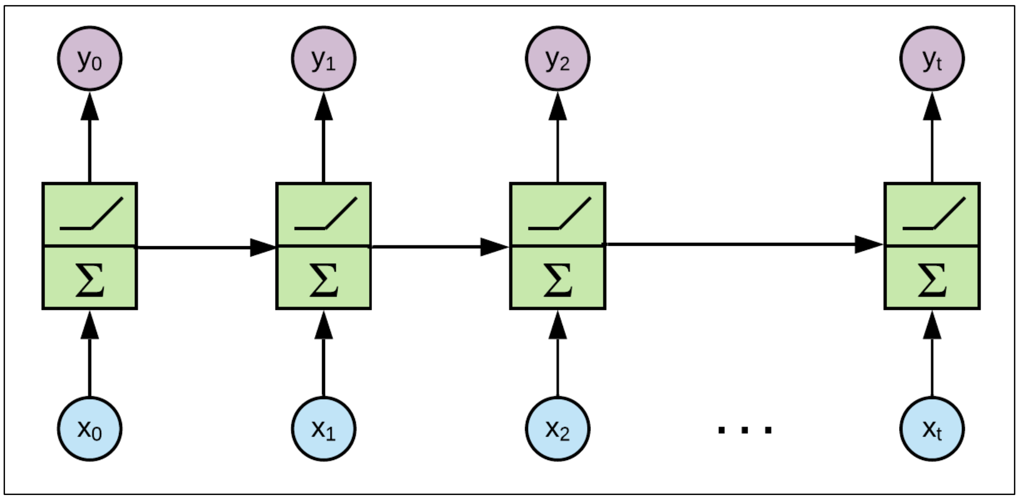


Figure 3.2: Unrolled recurrent neuron

It is possible to create layers of recurrent neurons and stack them to build a deep network composed of recurrent neurons (Deep RNN).

3.1.1 Memory cells

Because at timestep *t* the output of a recurring neuron depends on all inputs from past timesteps, it has a form of memory. A memory cell is the part of a neural network that maintains a state over time. The very basic memory cell is a single recurrent neuron or a layer of recurrent neurons.

Generally, the cell short-term state at time step t is equal to the cell output y_t, but sometimes not. Because data transformations are made during RNN propagations, there is almost no trace of the first inputs in the network state. Long short-term memory or LSTM cells can address this (Hochreiter and Schmidhuber, 1997). LSTM cells have a peculiarity, instead of having just one state like regular memory cells, it has two: (1) a short-term state, and (2) a long-term state, sometimes represented by h_t and c_t

respectively. The rationale of having an extra state, the long-term state in this case, is the ability to store in it, retrieve from it, keep or drop an input when it is not more necessary. This is a remarkable characteristic of LSTM cells which makes them able to learn what is important from the input and keep it for future use, and what is not simply drop it. This explains why these cells have managed to capture successfully longer-term patterns in time series.

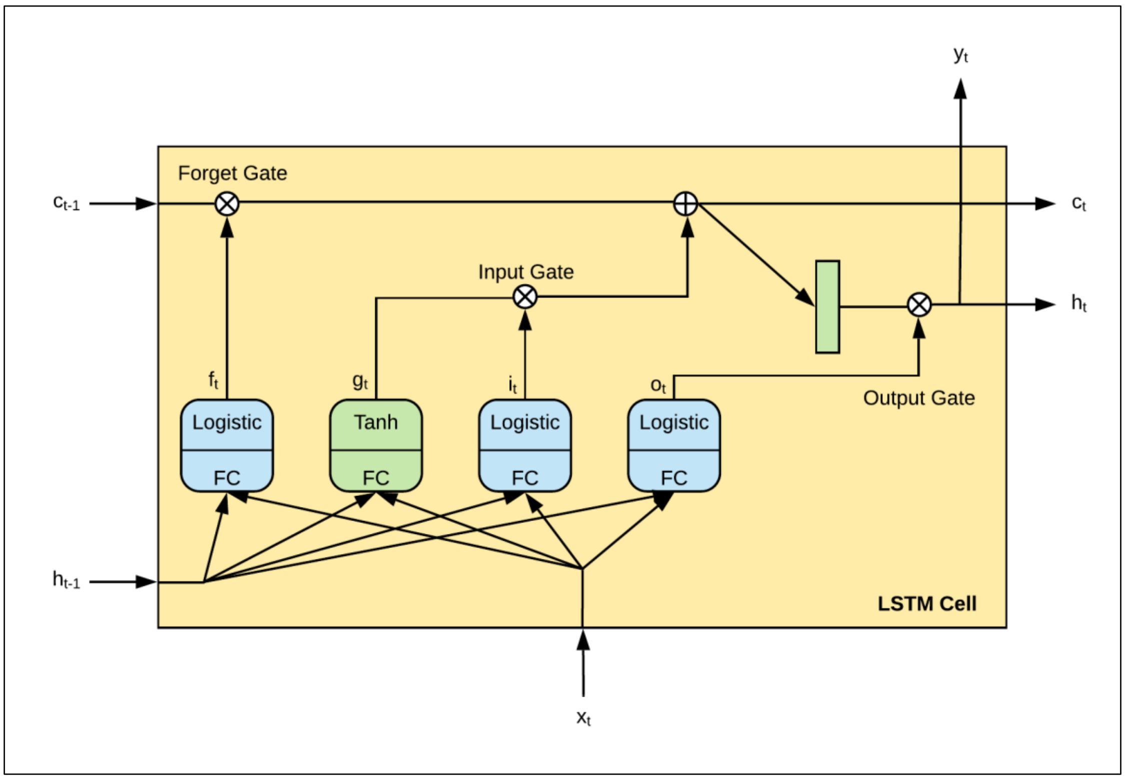


Figure 3.3: LSTM cell diagram

Figure 3.3 depicts the components and flow of an LSTM cell. Its state is divided into two vectors: c_t, the long-term state, and h_t, the short-term state. The main idea is that the network can learn what to store as well as what to retrieve (and discard) from the long-term state, for instance, while transversing the LSTM cell, the long-term state passes through the forget gate and may discard some memories. Then some new memories are introduced via the add process and incorporates the memories that the input gate has selected. Next, the long-term state is copied and passes through a *tanh* function. The output gate filters the result and creates the short-term state.

New memories come from layers f_t, g_t, i_t, and o_t.

g_t : this is the layer that analyzes the current x_t inputs and the previous short-term state h_{t-1}. The long-term state storages the most important parts of its output and removes the remainder.

f_t : controls the forget gate and handles the deletion from the long-term state.

i_t : input gate is controlled by this layer. It analyses which components of g_t are added to the long-term state.

o_t : controls the output gate and determines which long-term state parts are required to be read and output, both to h_t and y_t at each timestep.

As seen, LSTM cells were engineered to keep what is important from the input, store it whenever is useful, use it, and then discard it.

The following equations resume how the LSTM cells states and outputs can be computed:

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_{xi}^T x_t + \mathbf{W}_{hi}^T \mathbf{h}_{t-1} + \mathbf{b}_i\right) \tag{3.1}$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_{xf}^T x_t + \mathbf{W}_{hf}^T \mathbf{h}_{t-1} + \mathbf{b}_f\right) \tag{3.2}$$

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{xo}^T x_t + \mathbf{W}_{ho}^T \mathbf{h}_{t-1} + \mathbf{b}_o\right) \tag{3.3}$$

$$\mathbf{g}_t = \tanh\left(\mathbf{W}_{xg}^T x_t + \mathbf{W}_{hg}^T \mathbf{h}_{t-1} + \mathbf{b}_g\right) \tag{3.4}$$

$$\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \mathbf{g}_t \tag{3.5}$$

$$\mathbf{y}_t = \mathbf{h}_t = \mathbf{o}_t \otimes \tanh\left(\mathbf{c}_t\right) \tag{3.6}$$

Where $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xo}$, $\mathbf{W}_{xg}$ are the weight matrices for each layer connected to the input vector $\mathbf{x}_t$, while $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{ho}$, $\mathbf{W}_{hg}$ are the weight matrices for each layer connected to the short-term vector $\mathbf{h}_{t-1}$. The bias terms for each of these layers are $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$, and $\mathbf{b}_g$.

3.2 Reinforcement Learning

Reinforcement learning is not a single ML algorithm, but a group of algorithms which can deal with sequence-based problems. It is based on a software agent who learns from the environment and receives rewards for its actions.

Those algorithms have the following elements in common:

1. Agent
2. Environment
3. Policy
4. Reward system
5. Value function
6. Model of the environment

Agent

The software agent is the element that decides what action to perform to maximize the reward. It makes observations, acts in an environment, and receives rewards in return. The goal is to learn to act so that expected rewards can be maximized over time.

Environment

This could be a physical or virtual world with which the agent interacts and changes it.

Policy

The policy defines the behavior of the agent from state to action at a given time. It is essentially an algorithm that the software agent uses to define its actions.

Reward system

The reward system sets the objective for an RL problem and maps each pair of states/actions to a numerical reward.

Value function

In the long term, it reflects the cumulative future reward of the software agent.

Model of the environment

This is an optional element of RL. The model of the environment predicts its behavior.

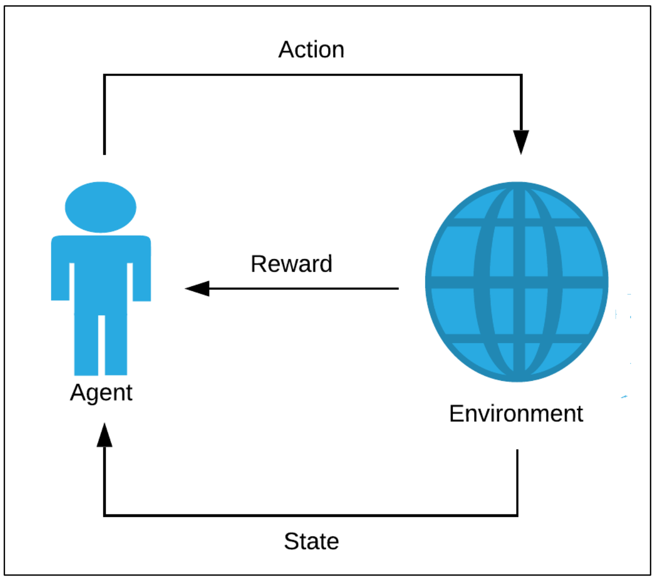


Figure 3.4: General reinforcement learning flow

3.2.1 Q-Learning

Q-learning is one of the model-free available RL algorithms, which means, it does not need a model for the environment. It aims to learn a policy to shape the action of the agent through trial and error.

However, it would take a quite long time to scan all feature space using a random policy. The solution is the deployment of an ε-greedy policy, which means, at each step agent acts randomly with probability ε, or greedily with probability 1–ε, whichever is greater.

At step *t*, the agent has a state from the environment to observe (s_t) and chooses an action (a_t). Next, it receives a reward (r_t) and the next state (s_{t+1}). The future reward is estimated by the value function Q.

$$Q_\pi(s,a) = \mathrm{E}\left[D_t \mid s_t = s, a_t = a, \pi\right] \tag{3.7}$$

Where E means expectation, D_t cumulative discounted reward at time *t*, and π decision policy. The cumulative discounted reward can be obtained by:

$$D_t = \sum_t \gamma^{t'-t} d_{t'} \tag{3.8}$$

Where γ means the discount factor for future rewards. The objective is to learn an optimum policy π^* such that the expected return is maximized.

$$Q^*(s,a) = \max_\pi (s,a) \tag{3.9}$$

The previous equation can be written recursively to obtain the Q-Learning algorithm equation.

$$Q^*(s,a) = d_t + \gamma \max_\pi Q_\pi(s,a) \tag{3.10}$$

It means that for each state/action pair, the Q-Learning algorithm keeps track of a running average of the rewards d_t the agent gets upon leaving the state *s* with action *a*, plus the sum of discounted future rewards it expects to get.

3.2.2 **Deep Q-Learning**

The key issue with Q-learning is that with many states and actions, it does not scale well, which means, it suffers from a lack of generality. A DNN can help with such by approximating Q-values for every state/action pair. This method is known as Deep Q-learning which combines RL and DNN to estimate the Q-values.

The DNN receives the current state s as input and gives the respective Q-value as output for each action a. The training is based on equation 1.10 and the loss function is written as:

$$L = \left[\left(d + \gamma \max_{a'} Q^* \left(s', a' ; \theta_{i-1} \right) \right) - Q \left(s, a ; \theta_i \right) \right]^2 \qquad (3.11)$$

Where i mean the current training epoch and θ is the DNN's set of parameters.

Experience replay

Due to nonlinearities, the use of neural networks to approximate Q-values is known to be unstable. The high correlation between contiguous states leads mainly to this instability, which biases the method itself and hinders its convergence.

The solution to tackle this problem is called experience replay. It consists of storing all the experiences, including state transitions, actions, and rewards. Then take randomly a few samples from the memory to update the DNN.

3.3 High-frequency data

Financial high-frequency data (HFD) are live market activity logs, also known as tick data. For instance, to buy a certain number of stocks at a determined price, when a person, dealer or an institution sends an order command, a bid quote with time, tick, price, and quantity is logged in exchange's information system.

When a new bid quote is the highest offer compared with all previously submitted bids, it is tagged as "the best bid." Similarly, if a new ask quote is below any of the existing quotes, it is regarded as "the best ask."

The set of time-based observations composed of ticks, latest quotes, order size, and volume is what composes financial HFD.

Financial HFD usually has the following features (Aldridge, 2013):

- Timestamp
- Security identifier
- Bid price
- Ask price
- Last trade price
- Last trade size

Timestamp

A timestamp indicates the date and time of the quote. The time could be when the quote has been released by the exchange (or broker) or when the quote is received by the trading system. The time lag between posting order and receiving feedback is extremely low nowadays, a few milliseconds.

Security identifier

The financial security identifier is another feature of high-frequency data. In stocks, the identifier can be a ticker, or a ticker plus an exchange symbol for stocks traded simultaneously on multiple exchanges.

Bid price

The best bid is the highest price on the market at some point in time to buy a security.

Ask price

The best ask is the lowest price to buy a security at some point in time.

Last trade price

The last trade price is the latest registered and broadcasted price of a security in a trade.

Last trade size

Like the previous feature, the last trade size is the size of the uttermost trade of a security.

Besides the mentioned features, some information systems also disseminate what is called "market depth". Depending on such depth, large orders may interfere with securities' prices. Market depth takes the general level and the number of open orders into consideration.

3.3.1 **Properties**

In the realms of financial applications and research, high-frequency data has unique properties compared to the commonly used daily (or monthly) data sets.

Volume

High-frequency data is voluminous. For instance, the number of observations equivalent to 30 years of daily data may be present on a single day for high-frequency data (Aldridge, 2013). This property carries some advantages and disadvantages. A great deal of information is available on a large number of observations, especially market microstructure info. On the other hand, it is hard to handle high-frequency data manually, making it feasible only with some kind of computational processing tool.

Bid-ask bounce

It is not possible to buy and sell a security simultaneously without a price gap, this is what bid-ask spread refers to.

Price changes are large enough in most low-frequency data to make bid-ask spread significant, but not in HFD: price changes may be less than the bid-ask spread. Also, continuous moves from bid to ask, a characteristic of high-frequency data, leads to a jump process that is difficult to manage via econometrics' models (Dacorogna, 2001). Still, this high-frequency characteristic conveys valuable information on the dynamics of the market.

Not log-normal (or normal) distributed

HDF returns do not follow a log-normal (or normal) distribution and this poses a difficulty to use traditional financial models, like Black-Scholes, and an opportunity to experiment with alternative ones like ML algorithms.

Irregularly spaced

Different time intervals separate high-frequency observations since tick data arrival is asynchronous. And this poses another difficulty in using traditional financial models, they require regularly spaced data (Aldridge, 2013). One way of dealing with the irregularities in the data is by sampling it every hour, minute, or second. These regular intervals are named "bars" of data. But, such irregularities on data arrival are not all bad since they also carry information.

Lack of buy-sell identifiers

Another feature of HFD is the absence of buy or ask labels on trades. However, this information is desirable in many situations like screening outliers, infer trade direction, compute liquidity measures, and so on.

Much of what has being discussed about high-frequency data make its relationship with big data consonant. Big data refers to the wide range of data growing at ever greater rates. The volume, velocity at which data is produced and collected, and the variability of the data points are characteristics of big data, as also of financial HFD. As mentioned, high-frequency data is voluminous. The velocity of high-frequency data gathering is in the magnitude of milliseconds (sometimes microseconds). Variability represents the opportunities that are available by interpreting the data. High-frequency data properties like being irregularly spaced can bring such variability by means of interpreting such irregularities as new information.

4 Implementation

In this chapter we describe the implementation of a strategy for Pairs Trading using high-frequency data with the support of Machine Learning.

Two machine learning algorithms were used, namely: Recurrent Neural Networks (RNN) and Reinforcement Learning (RL). The high-frequency data was composed of 2365 stocks traded during January 2018 on 3 different US exchanges (TAQ data).

4.1 Data pre-processing

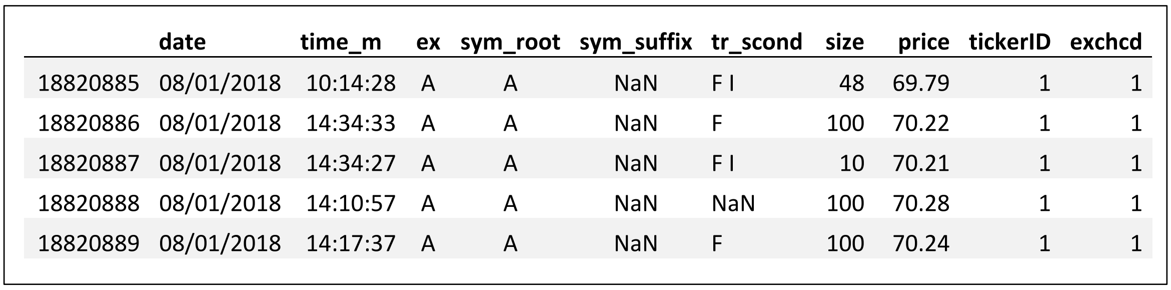
4.1.1 Load and split data

The raw data file with the high-frequency data was quite large (4,834,504,644 bytes) which easily overruns the memory of most workstations. The solution was to divide it into chunks while reading. The size of each chunk was defined as 1GB, and observations were filtered by trading days. In the end, 21 data frames, one for each trading day, were obtained and then saved in pickle[6] format

4.1.2 Filtering

Before starting any analysis, it is important to get acquainted with the data. The first five observations of the fifth trading day are displayed in Table 4.1.

Table 4.1: Data set before pre-processing



	date	time_m	ex	sym_root	sym_suffix	tr_scond	size	price	tickerID	exchcd
18820885	08/01/2018	10:14:28	A	A	NaN	F I	48	69.79	1	1
18820886	08/01/2018	14:34:33	A	A	NaN	F	100	70.22	1	1
18820887	08/01/2018	14:34:27	A	A	NaN	F I	10	70.21	1	1
18820888	08/01/2018	14:10:57	A	A	NaN	NaN	100	70.28	1	1
18820889	08/01/2018	14:17:37	A	A	NaN	F	100	70.24	1	1

[6] Python pickle format serializes Python objects into a byte stream. This byte stream can then be retrieved and de-serialized back to a Python object.

From the original data frame, the following features were kept for further analysis: date, time, stock symbol (sym_root) and price.

Another cleaning process performed was keeping only the trades within the core session of stock exchanges, which means, from 09:30:00 to 16:00:00 (Barndorff-Nielsen et al., 2009).

4.1.3 From ticks to frequency bars

Most typical financial models require regularly spaced data, but different time intervals separate observations on high-frequency data because the arrival of tick data is asynchronous (Aldridge, 2013). One way to address such irregularities is to sample from data every hour, minute, or second. These regular intervals are called *bars* of data.

Thus, the high-frequency data set was converted from ticks to regular 6 seconds intervals. In the end, there were 3,900 bars per trading day.

4.1.4 Selection of stocks regarding liquidity

Some stocks trade more frequently, which means, they present more liquidity. By plotting a null matrix, it was easy to check that some stocks have more bars populated than others. For instance, Figure 4.1 depicts the null matrix of the first 5 stocks of the fifth trading day where light shades represent the absence of observations (or null values). Clearly, stocks with symbols “AMRC”, “BIP”, and” BRT” have more liquidity than the others which present many more light shades.

In this work, liquidity selection was based on a threshold: stocks with at least 45% of the maximum number of trades per day were preserved, the ones which not reach the threshold were dropped. The rationale is that low liquidity stocks have lags during market corrections (Jasinski, 2020) and this could be exacerbated with high-frequency data.

Another drawback of keeping low liquidity stocks is due to strategy's expectation of entering and exiting trade positions many times a day which turns to be unfeasible if handling illiquid stocks.

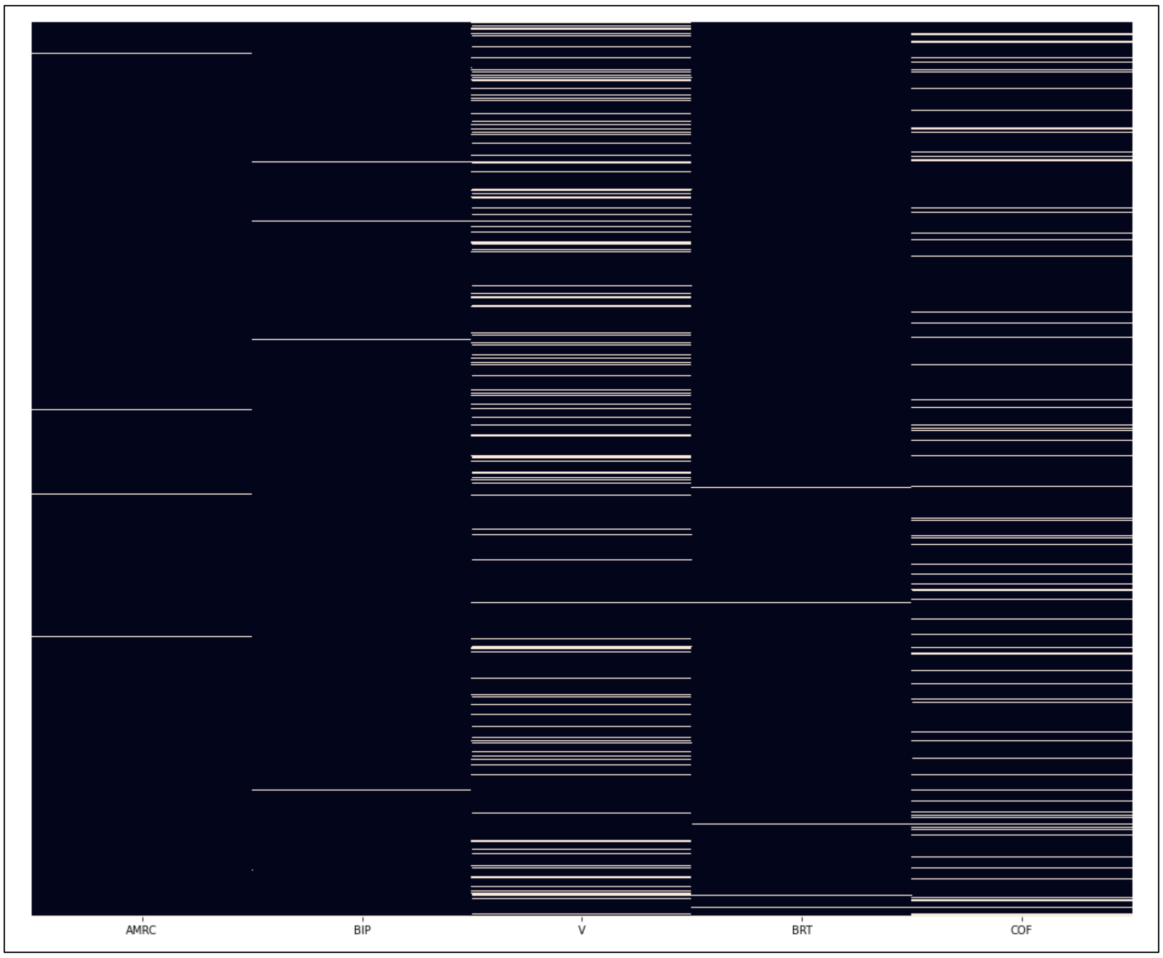


Figure 4.1: Null matrix of 5 stocks

4.1.5 **Filling bar gaps**

During the conversion from tick data to frequency bars, if no quotes arrived in a particular bar, then the price in the previous bar was taken as the price in the current bar, and so on. This procedure assumes that prices will remain stable in the absence of a new quote.

4.1.6 Dealing with outliers

Anomalies or outliers are observations that are very different from the expected. In a wide variety of applications, spotting outliers can be useful, such as in fraud detection, quality management in manufacturing, cybersecurity, etc. The interest here was to improve algorithm performance by removing outliers from the data set before training the model.

Entries for which price deviates by more than 10 mean absolute deviations from a rolling centered median of 50 observations (25 observations before and 25 after) were deleted. When computing the rolling centered median, the observation under consideration was not included (Barndorff-Nielsen et al., 2009).

4.2 Formation phase

The formation phase is the first stage of Pairs Trading after pre-processing data. In this work, the formation phase length was set twice the trade phase's as suggested by Gatev et al. (2006), that is, 2 days for formation, 1 day for trade.

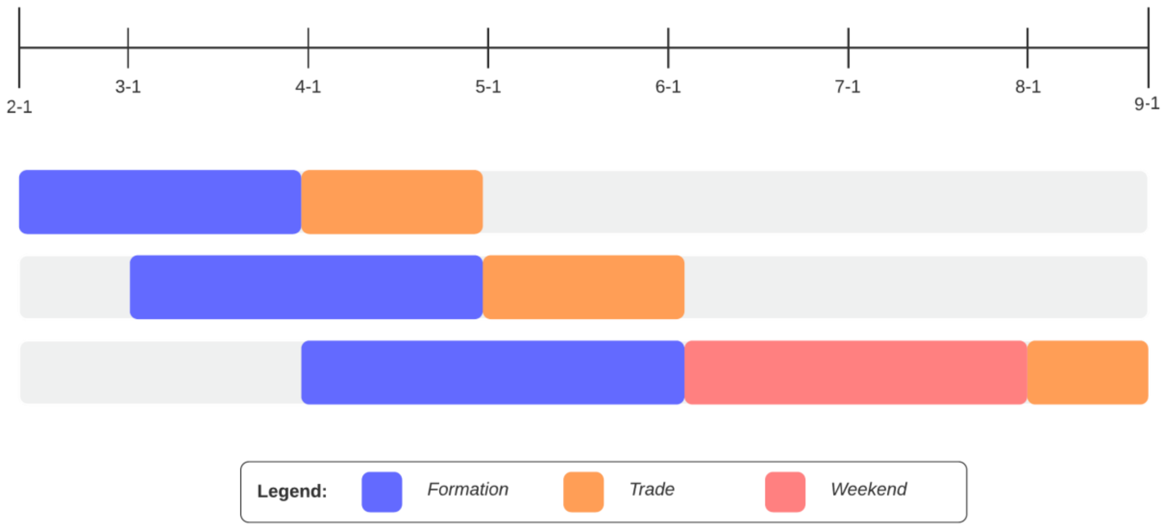


Figure 4.2: Formation/Trade phase schedule diagram example

4.2.1 Finding pairs

Finding pairs of stocks is the main task of the formation phase. There are many methods to achieve it.

This work used the cointegration method. The idea was to apply the Augmented Engle-Granger two-step cointegration test[7] on pairs of stocks to screen the most promising co-movers. The null hypothesis is that there is no cointegration.

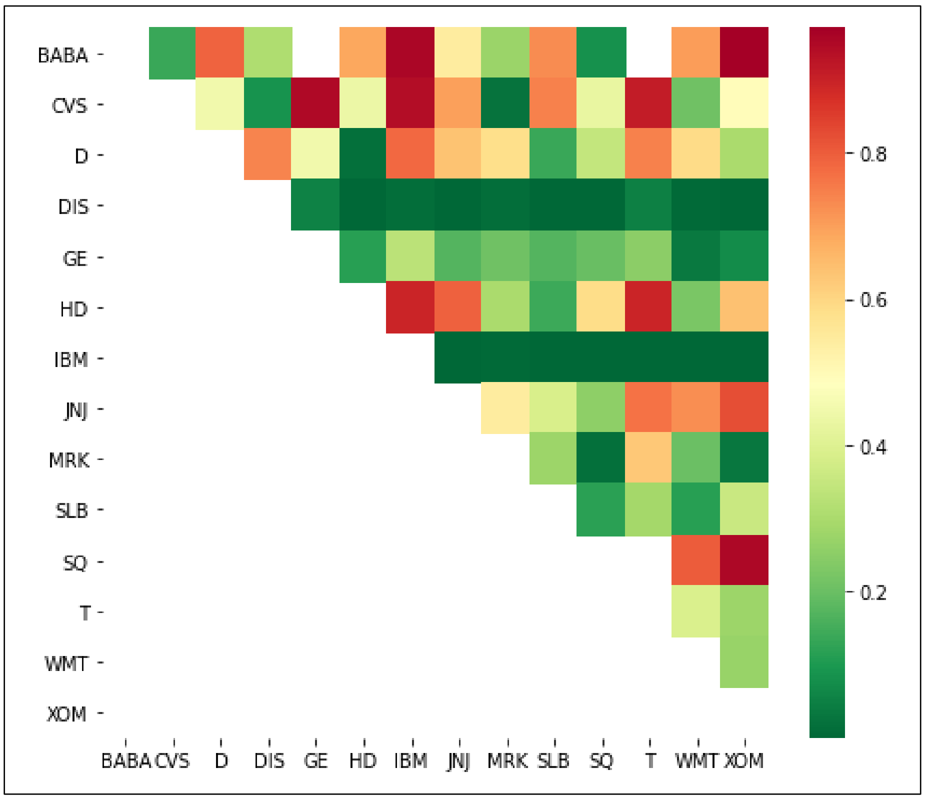


Figure 4.3: Cointegration-test heat graph

As an example, the third and fourth trading days were used for the formation phase. After pre-processing the data, the cointegration test on each pair was performed. The selection

[7] The Augmented Eagle-Granger test checks if two non-stationary time series are cointegrated, i.e. they move together, have a long-term equilibrium, and share a similar stochastic trend. The null hypothesis is that no cointegration exists.

criterion was set to a p-value of 0.02 or lower. The result was 17 potential pairs for the trade phase. Figure 4.3 depicts the heat graph of all pairs where the greener is the better.

One of the potential pairs was formed by stocks DIS and SQ. The p-value of the cointegration test (0.00331) was below the threshold which impels for the rejection of the null hypothesis, that is, stocks are not cointegrated. Their normalized prices during the very beginning of the third trading day (Figure 4.4) suggest a positive correlation. The situation then changes for a negative correlation, but overall, they seem to co-move during the first 2.5 hours of trade.

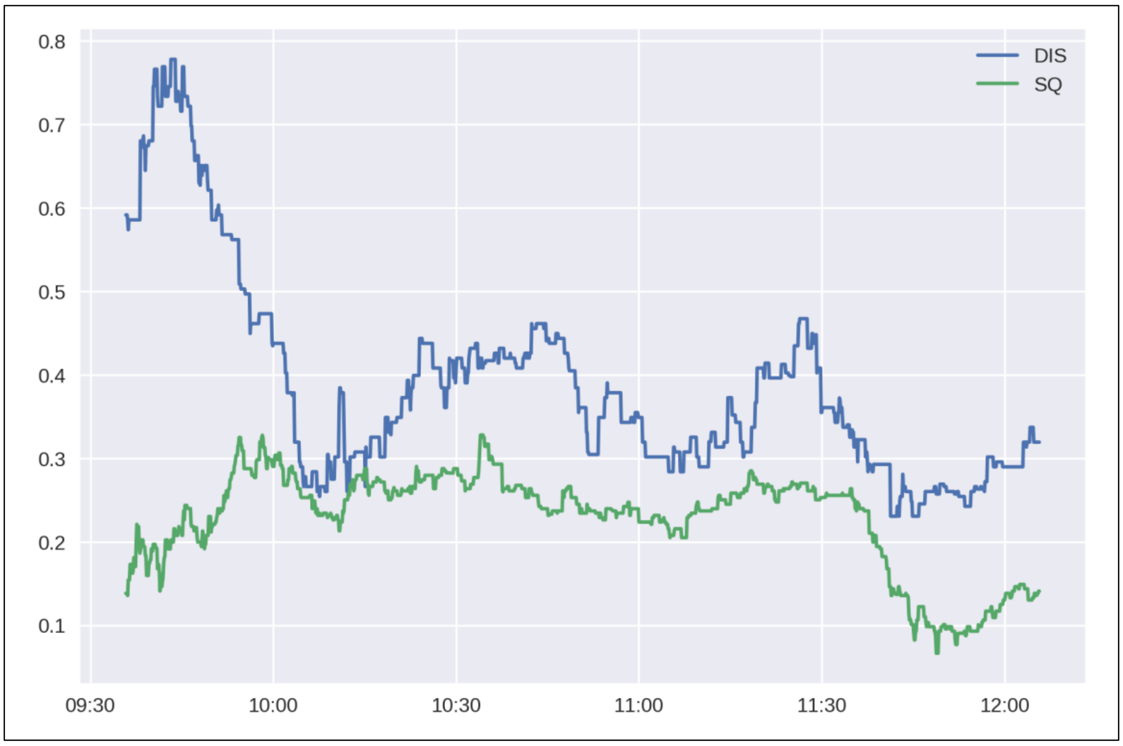


Figure 4.4: Normalized prices graph of a screened stock pair

4.3 Trade phase

The trade phase is the second stage of Pairs Trading. At this point, a strategy is deployed and checked. There are many methods to achieve such a trading strategy. This work approach is described in the following sections. It starts by calculating the pair's spread and its Z-score.

4.3.1 Calculating the spread

Synthetic is the term given to financial instruments designed to mimic other instruments' key features (Chen, 2019). The spread between the pair's stocks instead of their prices were used as a synthetic metric. Rather than compute the spread through the straight difference between stocks' prices, a 30-bars rolling OLS linear regression was deployed to compute the coefficient of the linear combination between the two stocks (Diba and Grossman, 1988). Such an approach to computing the spread is used to accommodate the distinction in magnitude between stocks' prices. Other rolling statistics[8] like moving averages of the spread are also used to avoid look-ahead bias.

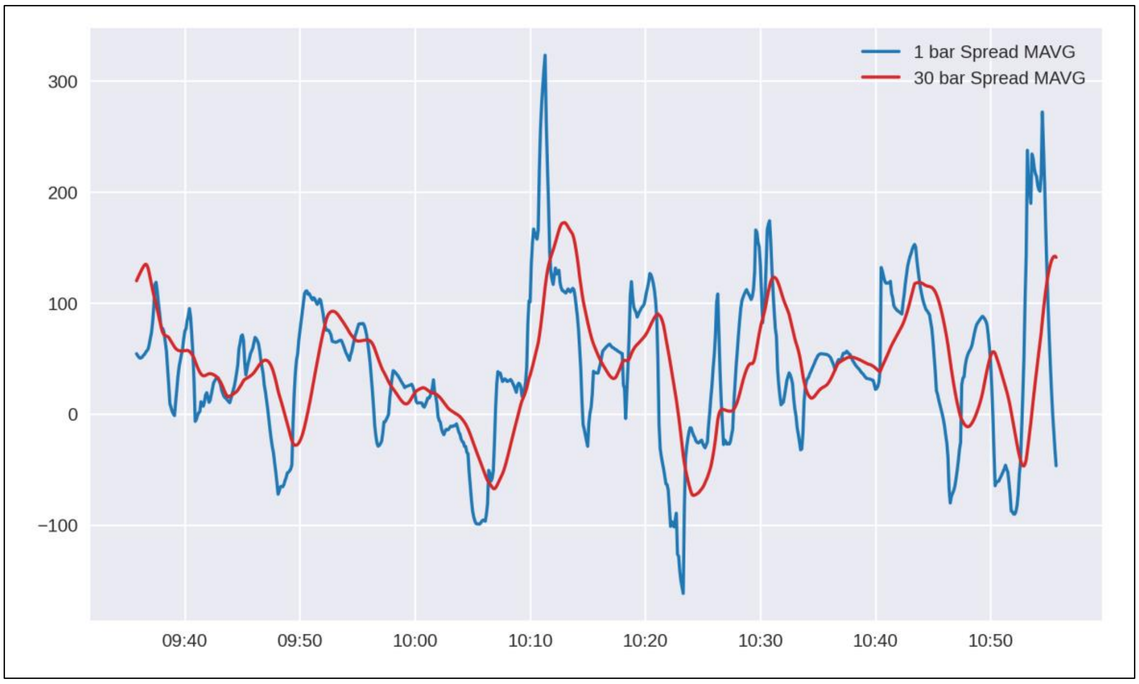


Figure 4.5: Spread moving averages graph

[8] Rolling statistics is a method used to compute statistics from a series of different subsets of the full data set obtained on a rolling basis.

Figure 4.5 depicts the spread moving averages of the spread during the first 80 minutes of the third trading day. As seen, the curve of the 30-bars moving average is smother than the curve of the 1-bar moving average. The process seems to be stationary around the mean. However, in statistical terms, the absolute spread is not very useful. It is more advantageous to normalize the spread and deal with its Z-score.

4.3.2 Computing Z-score

The Z-score shows how far a piece of data is from the population mean. Here, the Z-score will help to choose the direction of trade. For instance, if the Z-score value is positive, this means that the spread is higher than the average value and stocks' prices are diverging. In a mean reversion process, if the Z-score goes up, it is expected to go down after (towards the mean).

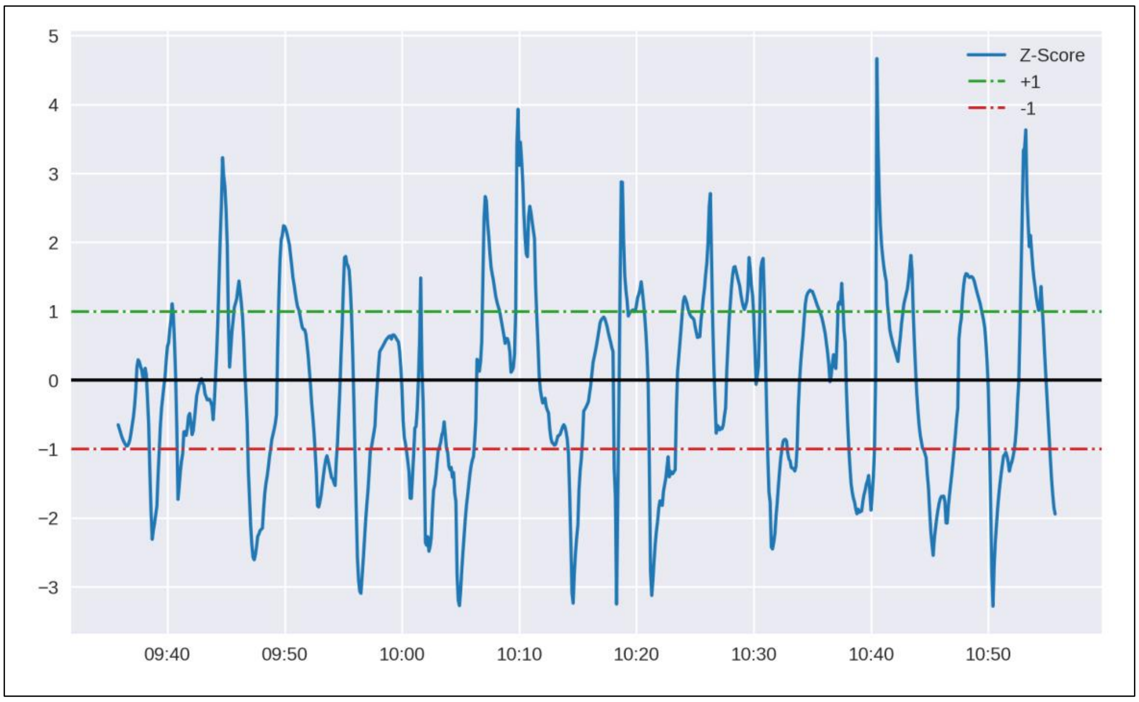


Figure 4.6: Graph of spread's Z-score with mean and SD boundaries

The Z-score was computed using the ratio between the difference of the moving averages and the 30-bars moving standard deviation.

$$z = \frac{SMA_1 - SMA_{30}}{SD_{30}} \tag{4.1}$$

Where: SMA_1 means 1-bar moving average

SMA_{30} means 30-bars moving average

SD_{30} means 30-bars moving standard deviation

The graph of Figure 4.6 shows a stationary behavior of the Z-score during the first 80 minutes of the third trading day which is an indication of a mean-reverting process.

4.3.3 **Z-score forecast**

Forecasting is a tool that uses historical data as input to predict the course of future observations in an informed way.

Here, the intention of using forecasting was to predict 15 timesteps ahead, each timestep as a 6 seconds interval, that means, predict the path of the Z-score in the next 1.5 minutes.

For such, a Recurrent Neural Network (RNN) was deployed but first, the missing bars (null observations) due to the previous rolling statistics computations need to be addressed (by dropping them). Also, the data must be scaled into the range of -1 and 1 using min-max normalization[9], and split into 3 subsets: train, validation, and test.

The division between train, validation, and test subsets had a 70:20:10 proportion. In machine learning, the segregation of data into subsets is standard practice. The reason is very simple: you obtain something unrealistic if you try to test the model on data that it has learned from since the goal is being able to predict with unknown data. It is important to properly train and test models to achieve a good generalization performance.

[9] Min-max normalization is one of the most common forms of data normalization. The minimum value is translated into -1 and the maximum value becomes 1. Any other value is transformed into a decimal between -1 and 1.

The deployed RNN had 4 LSTM layers with 50 neurons each. Mean square error[10] (MSE) was used as the cost function and Adam[11] (Adaptive moment estimation) as the optimization algorithm.

The process itself consists of successive iterations (or steps) organized in episodes where batches of 50 data points are used each time to predict the next 15 ones, which means, it used 50 timesteps backward to predict 15 timesteps ahead.

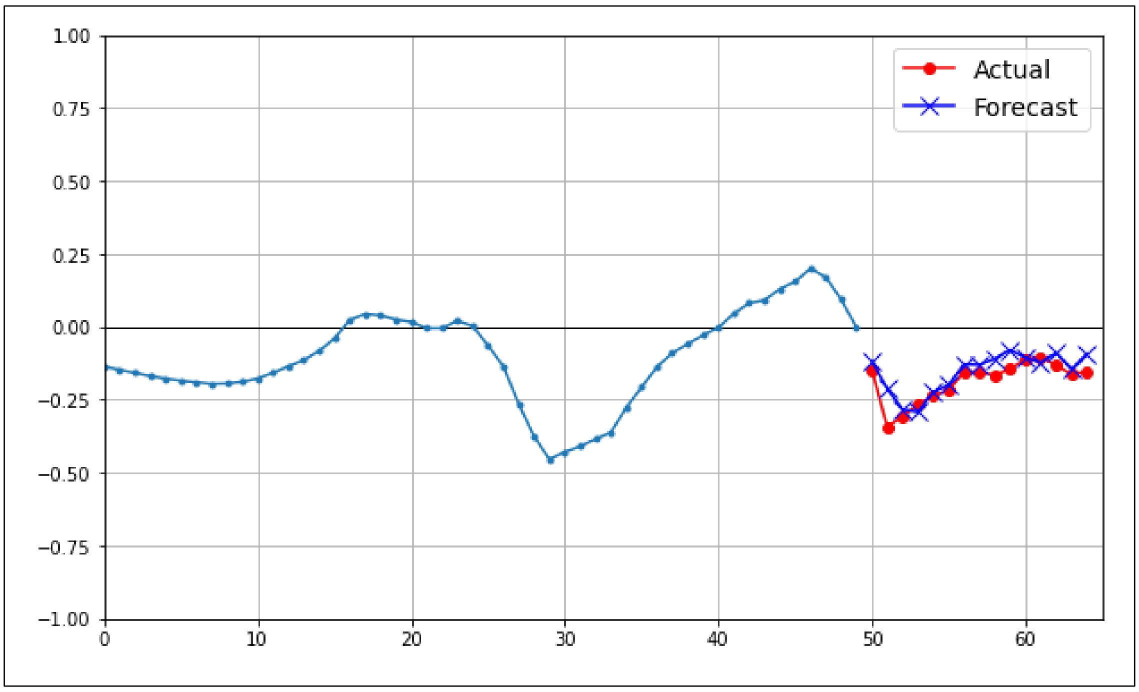


Figure 4.7: Graph of spread's Z-score with the forecast of 15 bars ahead

After each episode some metrics were obtained like elapsed time per step, train/validation average loss, and training/validation last step MSE. Figure 4.8 depicts an example of partial results.

[10] Mean squared error or MSE is a statistical estimator that calculates the average squared difference between the predicted and the measured values.
[11] Alternative optimization algorithm that calculates individual adaptive learning rates for different parameters from the estimates of first and second gradient moments, making it more computationally efficient, less dependent on large memory, and easier to implement.

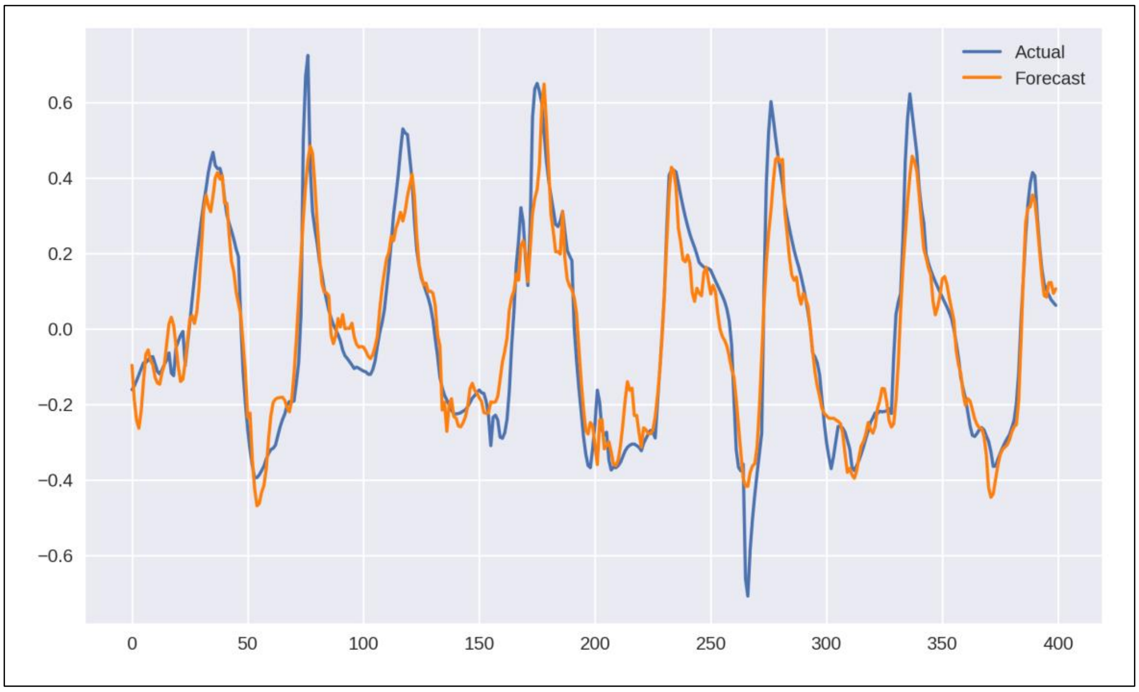


Figure 4.8: Graph of spread's Z-score with the forecast of 400 frequency bars

4.3.4 **Trading strategy**

The trading strategy was based on the previously computed Z-score and is detailed as follows:

- Enter in a **short** position for the **first stock** of the pair (and long the second one) when the spread deviates **+2** standards deviations from the mean and exit the position when the spread is +0.5 standard deviation from the mean.
- Enter a **long** position for the **first stock** of the pair (and short the second one) when the spread is **-2** standards deviations from the mean and exit the position when the spread reaches -0.5 standard deviations from the mean.
- Stop-loss limits for the short and long positions are respectively +3.0 and -3.0 standard deviations from the mean.

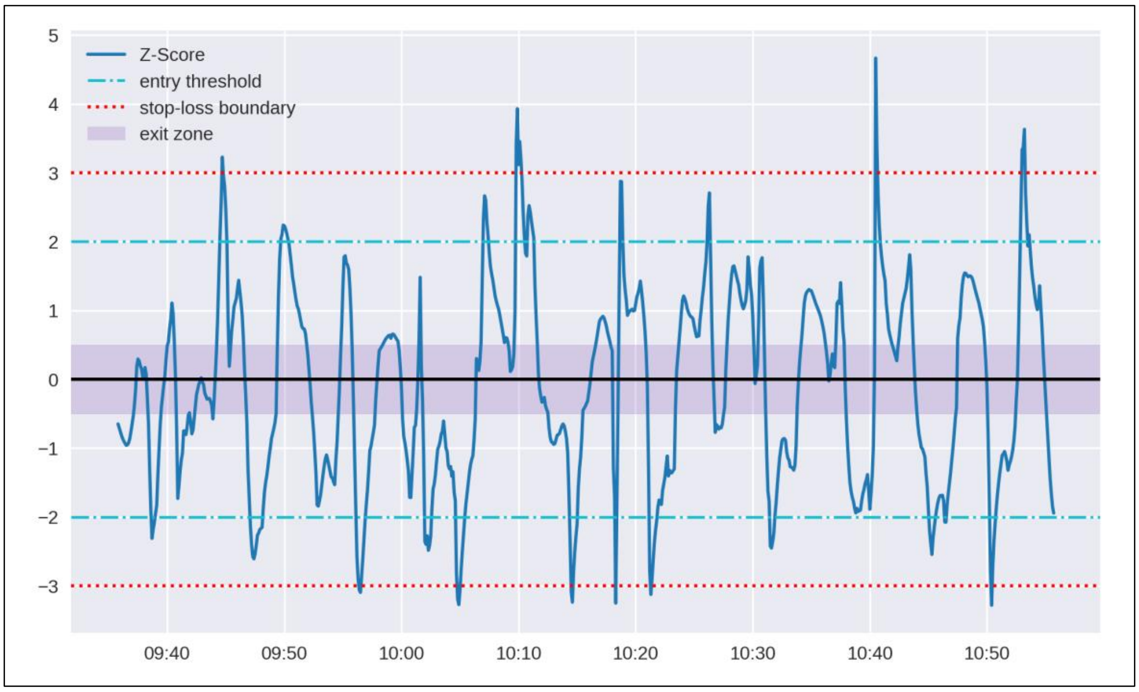


Figure 4.9: Graph of spread's Z-score with strategy's boundaries

According to Kim and Kim (2019), to optimize the Pairs Trading strategy it is necessary to set boundaries as criteria to decide if a Pairs Trading strategy should be executed or not. If a narrow boundary is established, many positions are taken, but profit would be low while a wide boundary will be highly rewarded by the execution of the strategy. Of course, these boundaries are used assuming there is a mean reversion process. But what if there is not? When stocks lose their cointegration relationship spread diverges from the mean and never reverse. Therefore, setting stop-loss boundaries could reduce this risk.

4.3.5 **Prepare to reinforcement learn**

Reinforcement learning (RL) is today one of the most appealing areas of Machine Learning. In RL there is a software agent that makes observations, performs actions, and receives awards in return. Its goal is to learn how to act so that its expected rewards can be maximized over time.

Using the Z-score as an explanatory variable, a response variable is generated as a result of the trading strategy. Possible values are as follows:

Table 4.2: Response variable values and descriptions

Value	Description
+1	enter a **short** position for the first stock of the pair and long for the second one
0	**hold**
-1	enter a **long** position for the first stock of the pair and short for the second one

Figure 4.10 shows an example of strategy execution. Short positions were assumed 2 times and long positions 6 times.

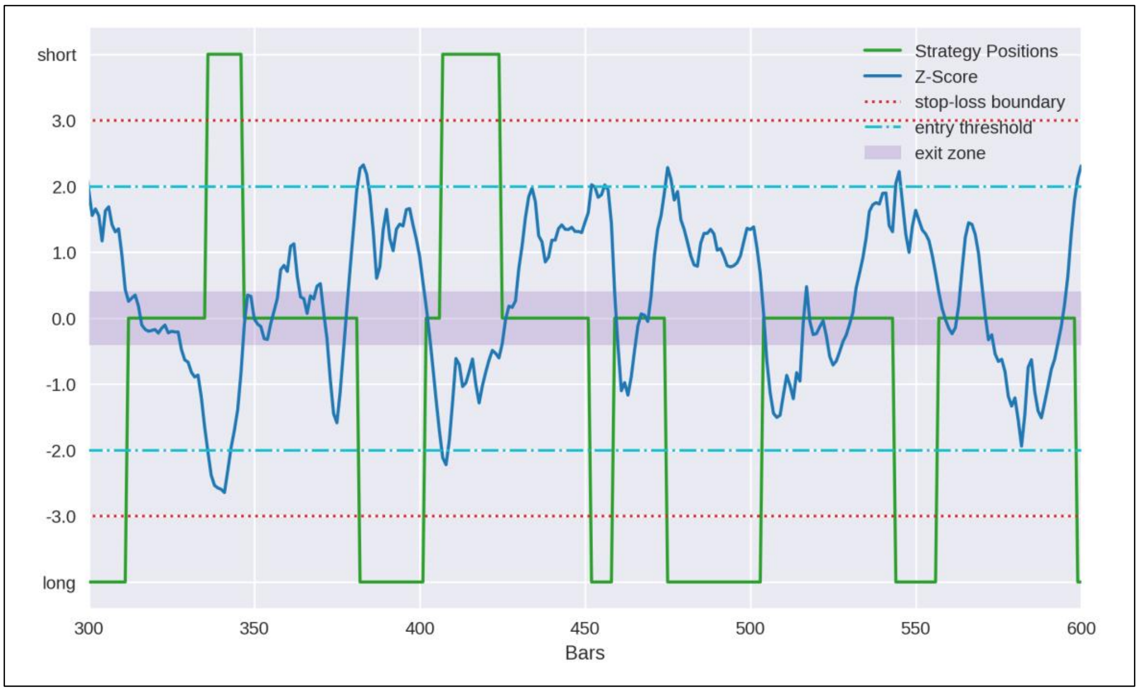


Figure 4.10: Graph of spread's Z-score with strategy's positions

The dataset was divided this time into train and test subsets in an 80:20 proportion. The exploratory variable was normalized using mix-max within the 0 to 1 range while the response variable was transformed to accommodate the RL model.

Table 4.3: Response variable before and after transformation

Before	After	Description
+1	**2**	enter a **short** position for the first stock of the pair and long for the second one
0	**1**	**hold**
-1	**0**	enter a **long** position for the first stock of the pair and short for the second one.

4.3.6 **Create and train agent**

The agent is a learning component that decides what to do to maximize the reward. The following is the sequence of steps of the RL algorithm which involves the agent:

1. The agent observes the environmental state.
2. The agent takes a policy-based action.
3. The agent receives the corresponding reward and the next environmental state.
4. Information on the reward is recorded for the given state/action pair and used to update the policy.

Over the process of creating an agent, some parameters need to be defined such as the number of timesteps backward to be considered at each step, in this work, 15 timesteps, the number of possible actions (3), and the number of features (1).

After, a Deep Q-learning network was built with 5 layers: the first three ones dense-type[12] with 30, 15, 5 neurons each (all using reLU activation function), then a flatten layer[13], and finally another dense layer using linear activation function[14]. MSE was used as the cost function and Adam as the optimization algorithm.

[12] Layer in a neural network that uses a non-linear activation function.
[13] Type of layer in a neural network that collapses the multi-dimensions of the input into a single array. It supports sequence input only.
[14] In this case the dense layer is reduced back to a linear layer.

Deep Q-learning is a widely used method among the available RL algorithms. The purpose is to learn a policy to model the selection of the agent through trial and error.

While a random policy makes it possible to reach every state/action combination, this could take an incredibly long time. An *ε*-greedy policy has then been applied, implying that the agent behaves arbitrarily at each point with the probability *ε*, or greedily with probability 1–*ε*, whichever is greater.

An important factor when creating the agent is the experience replay. This is to avoid instability during training which may prevent algorithm convergence. This factor was achieved using the following steps (Troiano et al., 2020):

1. Sample experiences from the memory (batch size: 32).
2. For each sampled experience: estimate future discounted reward and calculate Q-value associated with the current state.
3. Feed the network with the current state as input and Q-value as the target.
4. Decrease *ε* value while the agent gets experienced.

A memory list was created to support experience replay. It stored 3,000 experiences and included the current state, the action taken by the agent, the resulted reward, and the next state received from the environment.

To train the agent a function with a reward system was created. It works as follows: if the agent's action matches the previously computed one (according to Z-score strategy), the reward is +1. Otherwise, the reward is -1. The maximum possible reward is computed by the difference between the length of the input data set minus the lookback value plus one. Finally, the number of episodes for training was defined as 10.

Figure 4.11 shows an example of a rewarding path for the RL software agent. Rewards increased fast from the first to the second episode and then steadily grew until the last episode.

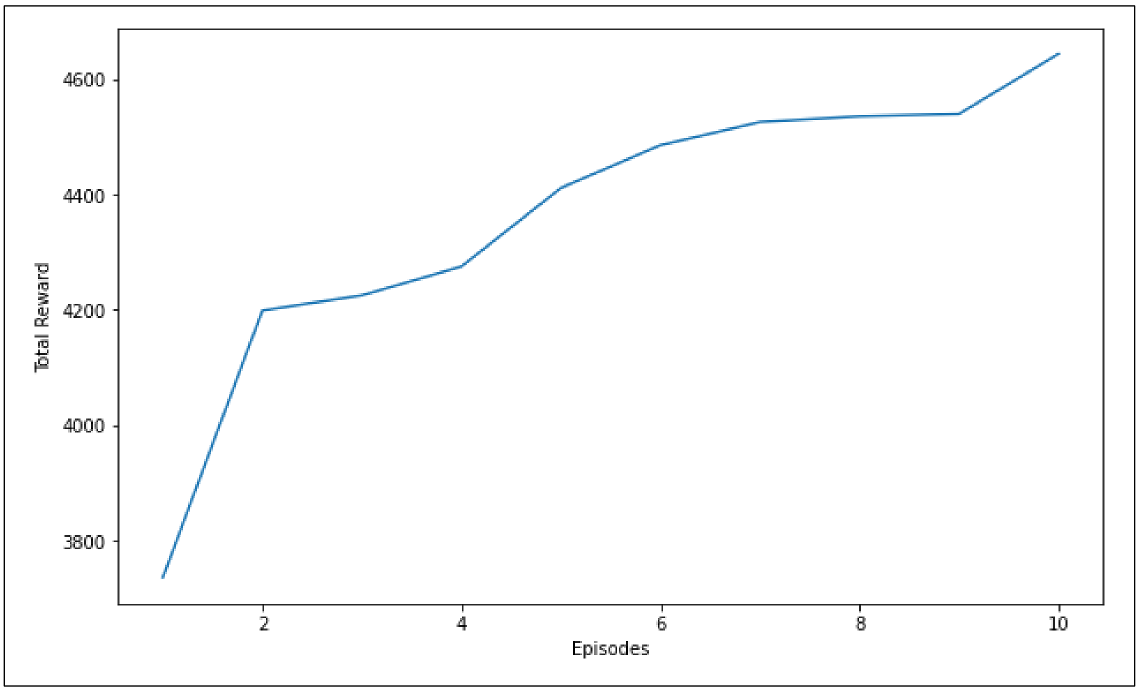


Figure 4.11: Graph of RL agent reward evolution

4.4 Backtesting

Backtesting is a common way to gauge performance of a strategy or model using simulation and historical data. When backtesting works, one can be more confident to deploy a strategy.

After training the RNN and RL algorithms with the formation phase data, we used the RL software agent to predict trade signals based on the trade phase data.

The following flowchart depicts this work's algorithm. Thus, backtesting involves using the RL software agent to predict trade signals using data not used for training it.

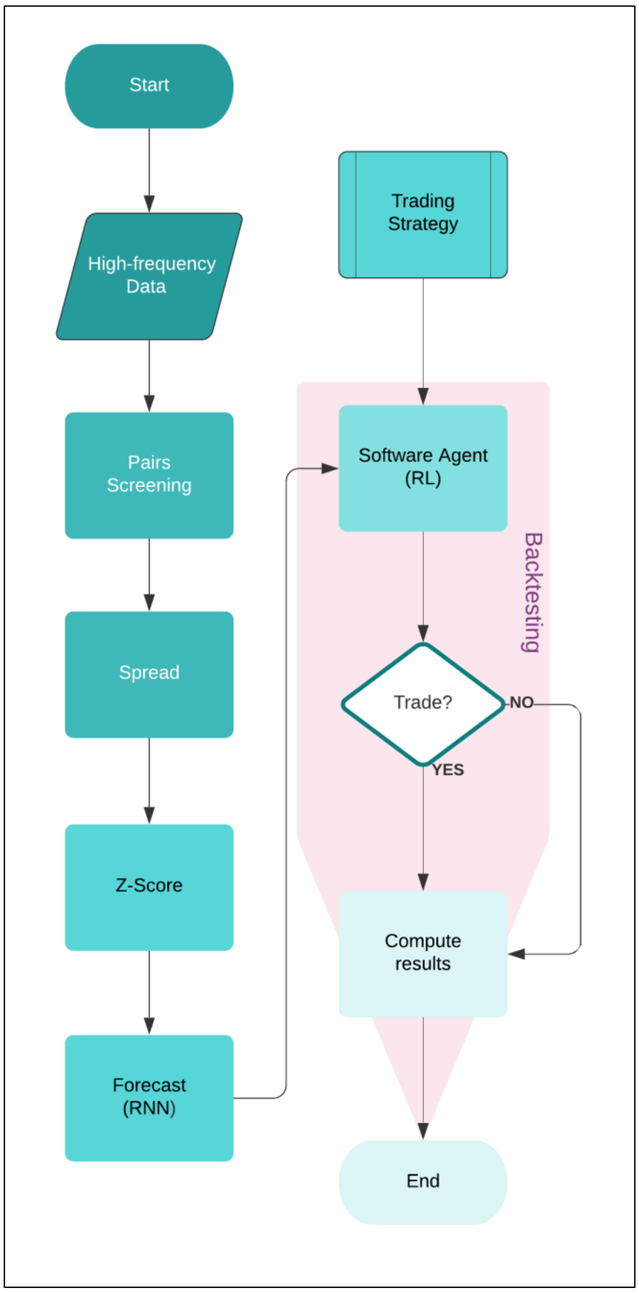


Figure 4.12: Methodology's flowchart

4.4.1 Outputs

Backtesting outputs were in ledger format where rows are timestamps and columns are respectively:

- Price of the first stock at timestamp
- Price of the second stock at timestamp
- Pair’s spread Z-score (normalized)
- Trade signal
- Order command
- Cost of the trade for the first stock
- Cost of the trade for the second stock
- Trade disbursement for the first stock
- Trade disbursement for the second stock
- Portfolio value (cash)

Price of the stock

Comprises the price of the first/second stock of the pair at some time *t* during the trade phase.

Pair’s spread Z-score

Amounts to the Z-score of the pair’s spread computed as previously and used as input for the models.

Trade signal

Constitutes one of three possible values (-1,0,+1) which indicates a trade action as described in Table 4.2.

Order command

The order command is computed in the way that if there is no open position and a -1 or +1 trade signal shows in the current timestamp it assumes the same value of the trade signal, which means, an order command is triggered as described in Table 4.2. But if there is an opened position, one of the following occurs:

Table 4.4: Order signals after an opened position

Position	Signal	Order	Description
+1	+1	**0**	maintain the **short** position for the first stock of the pair and long for the second one
+1	0	**-1**	exit the **short** position for the first stock of the pair and the long position for the second one
+1	-1	**-2**	exit the **short** position for the first stock of the pair and the long position for the second one and immediately enter a **long** position for the first stock of the pair and short for the second one
-1	-1	**0**	maintain the **long** position for the first stock of the pair and short for the second one
-1	0	**+1**	exit the **long** position for the first stock of the pair and the short position for the second one
-1	+1	**+2**	exit the **long** position for the first stock of the pair and the short position for the second one and immediately enter a **short** position for the first stock of the pair and long for the second one

The number of traded shares after a non-zero order command is computed as follows:

$$N_1^t = |q| I_2^t \tag{4.2}$$

Where: $I_2^t \approx P_2^t$

That means the number of shares of the first stock of the pair (N_1) at time t is equal to the integer obtained by rounding the price of the second stock price (P_2) at time t, times the absolute value of the order command (q) at that time. The number of shares of the second stock of the pair (N_2) was obtained using the same principle.

Notice that if any long/short position was still open at the end of the trade phase, it was automatically closed.

Cost of the trade

Comprises the costs for each trade performed. In this work 25 bps per transaction was used to account for those trade costs. Therefore, the trading cost was computed as follows:

$$C_1^t = 0.0025 N_1^t P_1^t \tag{4.3}$$

Where C_1^t means the cost to trade the first stock of the pair at time t. N_1^t is the number of traded shares of the first stock of the pair at time t, and P_1^t the price of the first stock of the pair at time t. The cost to trade the second stock of the pair (C_2^t) was obtained using the same principle.

Trade disbursement

Consists of the number of traded shares of the stock times its price.

Portfolio value

The cash or portfolio value at time t amounts to the previous portfolio value at time t-1 and the figures related to the execution of a trade order (if it exists). Therefore, the portfolio value can be obtained as follows:

$$V_t = V_{t-1} + X_t \tag{4.4}$$

Where V_t is the portfolio value at time t, V_{t-1} is the portfolio value at time t-1, and X_t is the total of the trade execution at time t obtained by:

$$X_t = \left(N_1^t P_1^t - C_1^t \right) + \left(N_2^t P_2^t - C_2^t \right) \tag{4.5}$$

It should be noticed that at time $t = 0$ portfolio value is zero, that means, there is no initial investment amount.

Table 4.5 shows an example of a ledger after backtesting for a pair of stocks.

Table 4.5: Backtesting ledger for a pair of stocks

	DIS	SQ	Z-score	signal	order	cost_DIS	cost_SQ	dis_DIS	dis_SQ	cash
08/01/2018 09:37	110.19	40.25	0.40048	1	1	-11.29	-11.17	-4407.40	4427.50	-2.36
08/01/2018 09:37	110.19	40.21	0.43384	1	0	0.00	0.00	0.00	0.00	-2.36
08/01/2018 09:37	110.19	40.23	0.42294	1	0	0.00	0.00	0.00	0.00	-2.36
08/01/2018 09:37	110.18	40.20	0.39752	1	0	0.00	0.00	0.00	0.00	-2.36
...	...	...	...	...	...	...	...	...	...	...
08/01/2018 09:37	110.18	40.20	0.38184	1	0	0.00	0.00	0.00	0.00	-2.36
08/01/2018 15:59	110.02	40.70	0.00888	1	0	0.00	0.00	0.00	0.00	-12071.76
08/01/2018 15:59	110.02	40.69	0.03162	1	0	0.00	0.00	0.00	0.00	-12071.76
08/01/2018 15:59	110.03	40.69	0.04533	1	0	0.00	0.00	0.00	0.00	-12071.76
08/01/2018 15:59	110.03	40.75	0.02729	1	0	0.00	0.00	0.00	0.00	-12071.76
08/01/2018 16:00	110.00	40.77	-0.0102	1	-1	-11.28	-11.21	4510.00	-4484.70	-12068.94

4.5 Performance measures

In this analysis, we considered each pair in a particular time window as a portfolio. To gauge the performance between portfolios, we employed the following performance measures.

4.5.1 Final position

Only the trades within the core session of stock exchanges, that is, from 09:30:00 to 16:00:00 were considered. The final position means the portfolio value at the end of the core session. If any long/short position was still open at that time, it was automatically closed.

4.5.2 Maximum portfolio value

At every frequency bar, the portfolio value was computed according to the previous portfolio value and the execution of a trade order (if it exists). The maximum portfolio value is therefore the highest amount computed.

4.5.3 **Return**

The return can be expressed as a price difference, but commonly it is computed as a percentual change in value. That makes it price level independent and more effective to compare different strategies.

Return equation can be illustrated as:

$$R_t = \frac{V_t}{V_{t-1}} - 1 \tag{4.6}$$

Where R_t means the return at time t, V_t portfolio's value at time t, and $P_{t\text{-}1}$ portfolio's value at time t-1.

4.5.4 **Average gain**

If a positive result is observed, the average gain measures the average profitability of the strategy. So, the average gain is the expected value of all positive results of the strategy.

4.5.5 **Average loss**

Similarly to the average gain, the average loss is the expected value of all negative results of the strategy.

4.5.6 **Win ratio**

The win ratio describes how profitable the portfolio ended. It is the ratio between the trading periods which experienced gains and the total trading periods. Win ratios help to evaluate signals precision since improved predictions lead to better win ratios.

4.5.7 **Volatility**

Return volatility measures how much the return increases and falls about its average value. Volatility is also considered a risk measure. The most common measure of volatility is the simple standard deviation which was used in this work.

4.5.8 **Sharpe ratio**

The Sharpe ratio is the average return per unit of volatility (or total risk) that exceeds the risk-free rate. The subtraction of risk-free rate from the expected return helps an investor to spot income from risk-taking activities more effectively.

The Sharpe ratio can be obtained from the equation:

$$S = \frac{R_p - R_f}{\sigma_p} \tag{4.7}$$

Where R_p means the expected return of the portfolio, R_f the risk-free rate, and σ_p the standard deviation of the portfolio's returns.

4.5.9 **Maximum drawdown**

The maximum drawdown or MDD is the average observed loss from a peak into a trough of the portfolio value before hitting a new peak. The maximum drawdown is a downside risk measure for a given time frame.

The MDD can be computed as follows:

$$MDD = \frac{V_{tr} - V_{pk}}{V_{pk}} \tag{4.8}$$

Where V_{tr} means a trough portfolio value and V_{pk} a peak portfolio value.

4.5.10 **Longs and shorts**

It counts how many long (or short) positions for the first stock of the pair were entered during the time frame. As a consequence of the Pairs Trading strategy, the algorithm enters the opposite position for the second stock.

5 Analysis of Results

The results using the proposed methodology are presented in this chapter. They are compared with another method widely used in time series analysis, the Autoregressive Integrated Moving Average (ARIMA) model.

5.1 Initial figures

As mentioned, the data set was composed of high-frequency data of intraday transactions for three US securities exchanges during the period of one month (January 2018). In total, there were 2,365 stocks.

The methodology utilized 2 days for the pairs' formation phase and 1 day for the trade phase on a rolling basis, so we ended up with 19 time windows to apply the ML algorithm. It must be noticed that some pairs identified during the formation phase were not feasible for analysis if the stocks were not present in the trade phase.

The number of feasible pairs on each time windows varied from 0 to 33. The average number of pairs per time window was 7.

5.2 Top performers

Considering the final portfolio value as the main performance criterion and not accounting trade costs, we built Table 5.1 with the top 5 performers where P_n means pair n within one of the 19 time windows. Note that the first time window did not form any feasible pair to use in the trade phase.

Table 5.1: Top 5 performers without trading costs

	Win	Final	Max	Return	Gain	Loss	WinRatio	Volatility	Sharpe	MDD	Longs	Shorts
P1	17	1315.70	1786.10	-0.0284	855.85	-99.13	0.85188	1.39511	-0.02039	-2.2076	201	215
P1	19	908.38	1129.56	0.36565	536.67	-160.71	0.84352	23.3189	0.01568	-1.7797	180	199
P4	12	856.93	1110.15	0.0323	592.42	-24.43	0.95533	1.42571	0.022655	-2.552	225	208
P1	11	788.96	1450.72	-0.0002	877.84	-83.92	0.9313	2.1605	-8.25E-05	-2.3728	275	283
P7	19	746.86	994.52	0.0047	679.96	-104.85	0.92294	0.27861	0.016871	-3.5507	183	194

From Table 5.1 it is evident that the strategy was profitable, and portfolios ended up with a positive value. Another notable aspect is that the maximum portfolio value was not reached at the end of the core session which makes room for some sort of optimized exit rule. Some of the returns were negative which is comprehensible since it was computed bar by bar and as mentioned, the last bar was not the one with the maximum portfolio value. The gains largely exceeded the losses and made the win ratios all above 0.84. Volatility results show an interesting aspect: most of them were in a range of 0.3 and 1.4 except one, 23.3 which could be considered an outlier. Nevertheless, that portfolio (P1W19) was not the one with the highest final and maximum values, as we would expect, nor is the one with the highest number of longs and shorts positions which is an indication that its Z-score had a great variability inside the buffer zone.

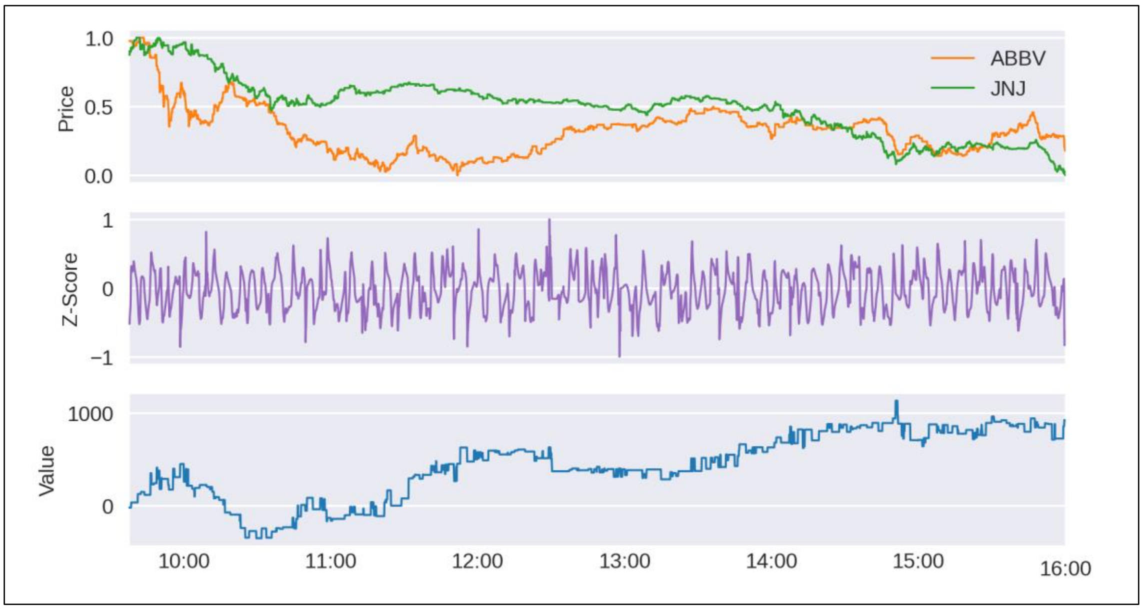


Figure 5.1: P1W19 normalized stock prices/Z-score plus portfolio value

The Sharpe ratios were predominant positive except two, one of them very low. This was due to the very low return and second-highest volatility of that portfolio (P1W11). The maximum drawdowns were in the range of -3.5 and -1.7 with the highest value to the portfolio with the highest volatility which indicates that besides the great variability the drawdowns were moderate. The number of longs/shorts varied from 180 to 283. The portfolio with the highest figures (P1W11) had the second-highest volatility which is a sign that such variability exceeded the buffer zone

5.2.1 Accounting costs

The costs for transactions may include the bid-offer spread, slippage, market impact, adverse selection, opportunity cost, and so on. We assumed the total cost of 25 bps per trade believing that it was sufficient to cover such charges in developed markets.

Table 5.2: Top 5 performers when accounting trading costs

	Win	Final	Max	Return	Gain	Loss	WinRatio	Volatility	Sharpe	MDD	Longs	Shorts
P6	14	-145.00	15.93	0.00203	15.93	-78.758	0.00026	0.0923	0.022044	-12.877	16	16
P2	10	-230.07	19.53	0.00094	19.53	-131.09	0.00026	0.04911	0.019222	-12.783	22	22
P2	9	-625.82	11.79	0.00069	11.79	-403.67	0.00026	0.09835	0.006998	-54.088	18	18
P3	16	-841.27	4.40	0.00096	4.40	-412.94	0.00235	0.08263	0.011582	-192.05	187	181
P4	13	-909.95	-25.15	0.00137	NaN	-487.91	0	0.03234	0.042346	0	99	100

As Table 5.2 depicts, when accounting costs the strategy was no longer profitable, at best case, the portfolio (P6W14) ended up with a negative value of -$145. However, all portfolios had positive maximum values (except one, P4W13) which also makes room for optimization of the strategy like exiting when a threshold is achieved. Besides all returns being positive, they were very low. Gains were substantially lower than losses which made the win ratios also very low. Volatilities varied from 0.03234 to 0.09835 and there was no outlier. They were significantly lower than the top 5 portfolios of Table 5.1 which was expected since volatility has a direct impact on the number of trades and consequently on profitability when accounting costs. Sharpe ratios were in the range of 0.007 to 0.042 making some portfolios better for this metric than the ones without costs which is reasonable since we had very low returns and also low volatilities. The maximum drawdowns were quite high for this group which indicates higher downside risk for these portfolios compared with the ones in Table 5.1. The number of longs/shorts was considerably lower than the top 5 portfolios without considering costs which confirms the impact of costs on the strategy.

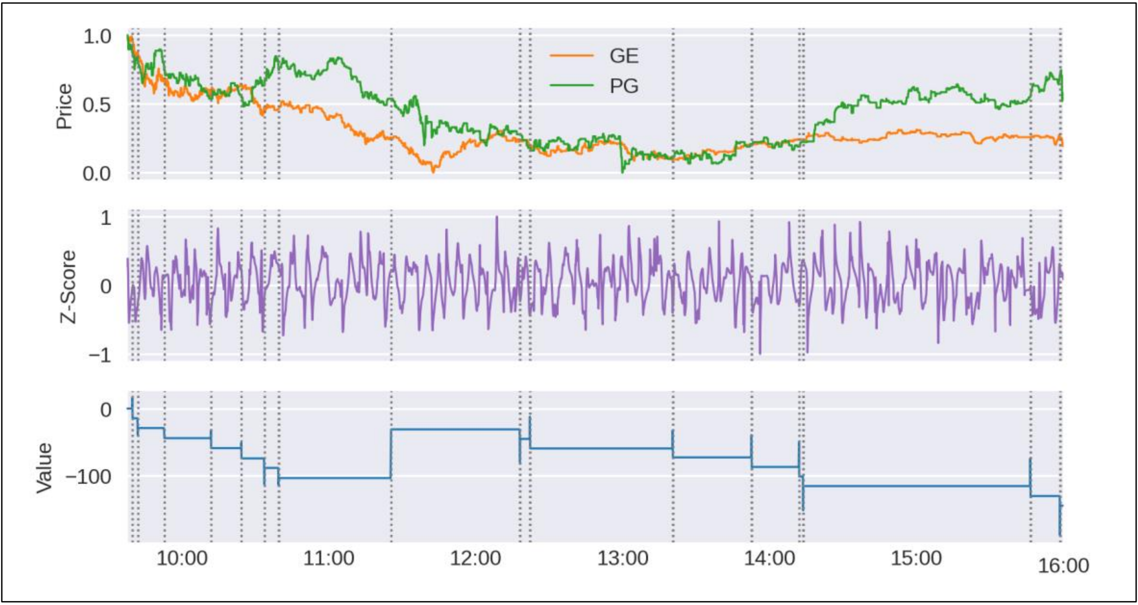


Figure 5.2: P6W14 normalized stock prices/Z-score including trade points

An interesting comparison is between portfolio P2W10 and P2W9. The latter had fewer longs/shorts than the former. Nevertheless, the MDD of the latter was 4 times higher which explains much higher drops on portfolio value and ending up with a much lower figure.

5.3 ARIMA comparison

ARIMA models are a family of mathematical models widely used for time series modeling and forecasting. For the sake of comparison, we deployed an alternative approach using ARIMA models which is described as follows:

1. Perform a grid search on the top 5 performers using their spread's Z-score from the formation phase data set to spot ARIMA hyperparameters[15].
2. Split the formation phase data set into train and test subsets.
3. Use the train subset to fit the ARIMA model and produce a forecast for each item in the test subset.
4. Train an RL software agent with ARIMA models' predictions.

[15] Grid search screened ARIMA(6,1,0) as the model with the lowest root-mean-square error (RMSE).

5. Set the RL software agent to predict signals using the trade phase data.
6. Gather the results and perform backtesting.

The same pairs and time windows were utilized in this comparative. Table 5.3 shows the results using the very same top performers of Table 5.1, that is, not considering trading costs.

Table 5.3: Top 5 performers using ARIMA (without trading costs)

	Win	Final	Max	Return	Gain	Loss	Win	Volatility	Sharpe	MDD	Longs	Shorts
P1	17	878.37	934.55	-0.6303	302.62	-30.666	0.8869	37.7923	-0.01668	-3.2068	106	118
P1	19	-117.17	52.51	0.00502	20.338	-75.666	0.0024	0.52844	0.009503	-11.765	27	28
P4	12	802.13	841.89	0.04238	382.33	-20.635	0.953	2.59333	0.016342	-2.4928	147	147
P1	11	-97.56	77.13	-0.009	51.148	-89.234	0.0091	0.26493	-3.41E-02	-4.6965	53	50
P7	19	44.84	413.99	-0.2636	214.35	-7.898	0.9794	17.3373	-0.01521	-1.2308	236	21

The strategy is profitable most of the time using ARIMA to model the spread's Z-score and deploying the software agent to predict signals during the trade phase. However, it is much less profitable than the ML algorithm and sometimes unprofitable. For instance, the best performer using ML ended up with a portfolio value of $1,315.70 against the final value of $878.37 using ARIMA. This was due to the much higher number of longs/shorts performed by the ML algorithm.

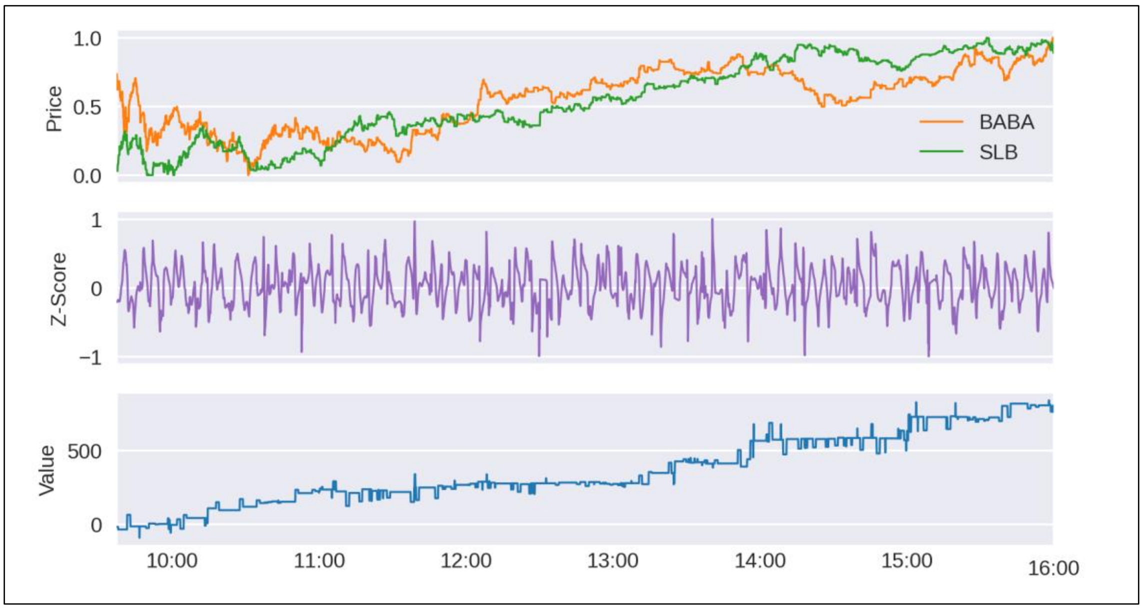


Figure 5.3: P4W12 normalized stock prices/Z-score plus portfolio value

The maximum portfolio value was different from the final portfolio value for all portfolios which also makes room for an optimized exit rule. Returns were all negative, except two, one of them was portfolio P4W12 which had a final value not far from the ML result. Most of the gains were much higher than losses which made the win ratios above 0.89 except for the two unprofitable portfolios. Volatilities ranged from 0.3 to 37.8 with the best performer (P1W17) having the largest value. Sharpe ratios lined up with returns and volatility, for example, the best performer had the highest volatility, making its Sharpe ratio the second-lowest. The best MDD value was for the portfolio with a final value not far from the ML algorithm (P4W12) which indicates the lower downside risk among the others. The number of longs/shorts were significantly lower than using the ML algorithm, except for one portfolio, P7W19, which seemed quite unbalanced (236 longs versus 21 shorts). Nevertheless, it had a positive final value.

When accounting trading costs the use of ARIMA to model spread's Z-Score during the formation phase (Table 5.4) performed worse most of the time compared to the same portfolios of Table 5.2.

Table 5.4: Top 5 performers using ARIMA when accounting trade costs

	Win	Final	Max	Return	Gain	Loss	Win	Volatility	Sharpe	MDD	Longs	Shorts
P6	14	-3958.25	15.64	0.00119	15.64	-1607.1	0.0005	0.0804	0.014841	-255.76	207	195
P2	10	-3140.06	15.21	0.00183	15.21	-1524.6	0.0005	0.10919	0.016792	-207.39	273	251
P2	9	-2454.59	-17.12	0.00183	NaN	-1386.3	0	0.04877	0.037517	0	56	57
P3	16	-687.79	4.40	-0.0128	1.87	-347.79	0.0128	0.89398	-0.01435	-157.19	137	115
P4	13	-581.52	-25.15	0.00147	NaN	-302.94	0	0.04519	0.032642	0	59	58

The P6W14 portfolio, for instance, it has the worst final value while it was the best performer when using the ML algorithm. Its number of longs/shorts was much higher which disfavor ARIMA's prediction accuracy. Maximum portfolio values were all higher than the final value and again makes room for an optimized exit rule. Returns were low, and losses were much higher than gains which made win ratios very low. Volatility varied on a scale from 0.04 to 0.89. Sharpe ratios were all positive except for one portfolio with a negative result. MDD was extremely high for all portfolios but two, which made them have very high downside risk. The number of longs/shorts were consistently higher than the ML results.

Comparing the accuracy of RNN and ARIMA when modeling and predicting spread's Z-score during the formation phase (Figure 5.4) we can check a clear advantage to the ML algorithm which reflected on the performance measures.

The highest accuracy gap was on portfolio P1W17, the best performer when trading costs were not accounted which was due to the much higher number of longs/shorts performed by the ML algorithm and its higher accuracy.

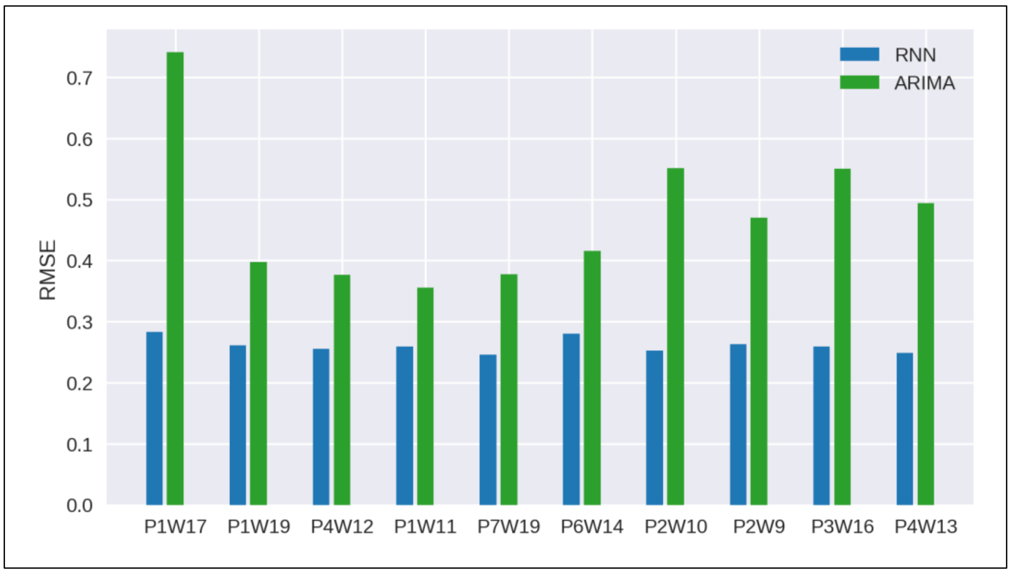


Figure 5.4: Models' accuracy comparison among top 5 portfolios

Regarding the RL software agent's precision, it was computed as follows:

$$\text{Precision} = \frac{tp}{tp + fp} \tag{5.1}$$

Where *tp* means the number of true positives and *fp* the number of false positives. Therefore, the precision of a classification algorithm is the ability to not mark a negative sample as positive.

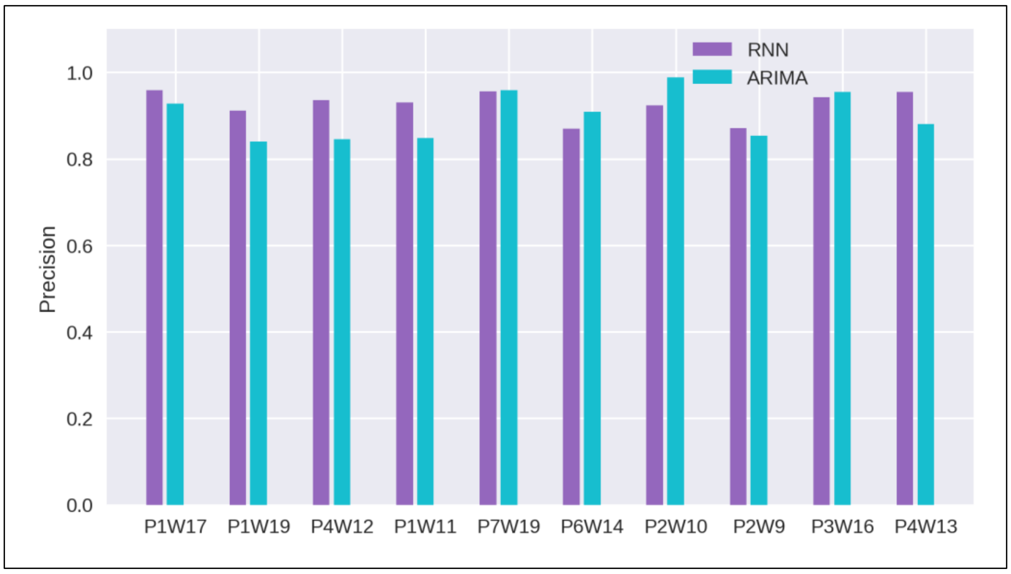


Figure 5.5: Models' precision comparison among top 5 portfolios

As we can see from Figure 5.5 there was no significant gap on RL software agent precision when predicting trade signals for the different models.

5.4 Final remarks

Not all portfolios had a positive final value, which means, were profitable, even disregarding trading costs. However, 63% of these portfolios presented pairs which lost cointegration during the trade phase according to the criterion used in this work: the p-value of the Augmented Engle-Granger two-step cointegration test below 0.02. Nonetheless, all portfolios (except two) presented the maximum value greater than zero when not considering trading costs.

6 Conclusion

In this work, we analyzed a well-known statistical arbitrage strategy, Pairs Trading, with the objective to use up-to-date ML algorithms to take advantage of the characteristics of high-frequency data and support an application of such a trading strategy.

The data was extracted from the book order of three different US stock exchanges – TAQ data for one month (January 2018).

We deployed a two-phased framework: one to screen stock pairs, and another to simulate trades with those pairs. To select pairs, we used the Augmented Engle-Granger two-step cointegration test and a threshold for the resulted p-value as a selection criterion. After, we used pairs' spread Z-score to feed one of the most well-regarded Machine Learning algorithms to model times series and perform predictions, the Recurrent Neural Network (RNN). Once trained, the RNN predictions were used to feed another ML algorithm, a Deep Q-Learning Network (DQN) in which a software agent creates signals from the spread's Z-score. As a trading strategy, we defined thresholds for the spread's Z-score to enter and exit positions. After trained, the software agent produced trading signals during the trade phase making it possible to create portfolios and measure the performance.

Results showed that the algorithms learned well, and the strategy was profitable most of the time (before trading costs) unless the cointegration between stocks is lost during the trade phase. As a matter of comparison, we deployed an ARIMA model in lieu of the RNN to predict the spread's Z-score and also trained the software agent. Results revealed that the RNN model was (on average) 45% more accurate than the ARIMA model which reflected on the profitability of the portfolios.

Trading costs had a great impact due to the high number of trades. Top 5 performers when not accounting trading costs had their win ratio dropped from 0.9 (on average) to 0.001.

According to results, the maximum value for the portfolios was often higher than the final value which promotes the use of optimization for an exit rule, particularly when considering trading costs.

6.1 Limitations of this research

There are many methods to screen cointegrated pairs of stocks during the formation phase and we chose one of them. However, we acknowledge that during research, experimenting with more than one method is desirable. Still, the focus of this work was on how ML algorithms would perform during the trade phase.

Due to constraints on computational power available, we were not able to experiment with different reward systems for the RL software agent which could have permitted alternative approaches like dynamic thresholds for the spread's Z-score to enter and exit trade positions.

Also, we utilized just one feature to feed the models, but we believe that feeding them with more features would improve their learning and therefore, their accuracy. But naturally, with the need for extra work like feature engineering.

6.2 Future work

From the results, the deployment of an optimized exit rule during the trade phase to increase the profitability is enticing for a future work. Further, the development of a method to easily spot the loss of cointegration between stocks would support such optimization. Finally, the designing of tools using the results of this research to integrate with real online trading platforms is promising.

References

Aldridge, I. (2013). *High-frequency trading : a practical guide to algorithmic strategies and trading systems*. Hoboken, New Jersey: John Wiley & Sons, Inc.

Avellaneda, M. and Lee, J.-H. (2010). Statistical arbitrage in the US equities market. *Quantitative Finance*, 10(7), pp.761–782.

Bao, W., Yue, J. and Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLOS ONE*, 12(7), p.e0180944.

Barndorff-Nielsen, O.E., Hansen, P.R., Lunde, A. and Shephard, N. (2009). Realized kernels in practice: trades and quotes. *The Econometrics Journal*, 12(3), pp.C1–C32.

Bowen, D., Hutchinson, M.C. and O'Sullivan, N. (2010). High-Frequency Equity Pairs Trading:Transaction Costs, Speed of Execution, and Patterns in Returns. *The Journal of Trading*, 5(3), pp.31–38.

Brim, A. (2020). Deep Reinforcement Learning Pairs Trading with a Double Deep Q-Network. *2020 10th Annual Computing and Communication Workshop and Conference (CCWC).*

Chen, C.-T., Chen, A.-P. and Huang, S.-H. (2018). Cloning Strategies from Trading Records using Agent-based Reinforcement Learning Algorithm. *2018 IEEE International Conference on Agents (ICA)*, pp.34–37.

Chen, J. (2019). *Synthetic*. [online] Investopedia. Available at: https://www.investopedia.com/terms/s/synthetic.asp#:~:text=Synthetic%20is%20the%20term%20given [Accessed 17 Jun. 2020].

Dacorogna, M.M. (2001). *An Introduction to High-Frequency Finance*. Academic Press.

Diba, B. and Grossman, H. (1988). Explosive Rational Bubbles in Stock Prices? *American Economic Review*, 78(3), pp.520–30.

Do, B. and Faff, R. (2010). Does Simple Pairs Trading Still Work? *Financial Analysts Journal*, 66(4), pp.83–95.

Do, B. and Faff, R. (2012). Are Pairs Trading Profits Robust to Trading Costs? *Journal of Financial Research*, 35(2), pp.261–287.

Du, M., Li, F., Zheng, G. and Srikumar, V. (2017). DeepLog. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*.

Elliott, R.J., Van Der Hoek , J. and Malcolm, W.P. (2005). Pairs trading. *Quantitative Finance*, [online] 5(3), pp.271–276.

Fallahpour, S., Hakimian, H., Taheri, K. and Ramezanifar, E. (2016). Pairs trading strategy optimization using the reinforcement learning method: a cointegration approach. *Soft Computing*, 20(12), pp.5051–5066.

Fischer, T. and Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2), pp.654–669.

Fischer, T., Krauss, C. and Deinert, A. (2019). Statistical Arbitrage in Cryptocurrency Markets. *Journal of Risk and Financial Management*, 12(1), p.31.

Gatev, E., Goetzmann, W. and Rouwenhorst, K.G. (1999). Pairs Trading: Performance of a Relative Value Arbitrage Rule. *NBER Working Papers*, 7032. National Bureau of Economic Research.

Gatev, E., Goetzmann, W.N. and Rouwenhorst, K.G. (2006). Pairs Trading: Performance of a Relative-Value Arbitrage Rule. *Review of Financial Studies*, [online] 19(3), pp.797–827. Available at: http://www-stat.wharton.upenn.edu/~steele/Courses/434/434Context/PairsTrading/PairsTradingGGR.pdf.

Jasinski, N. (2020). *During Market Stress, the Most-Liquid Stocks Perform Best, Goldman Sachs Says*. [online] www.barrons.com. Available at: https://www.barrons.com/articles/market-stress-most-liquid-stocks-illiquid-ratio-goldman-sachs-small-caps-51584478529 [Accessed 17 Jun. 2020].

Jurek, J.W. and Yang, H. (2007). Dynamic Portfolio Selection in Arbitrage. *SSRN Electronic Journal*.

Kim, T. and Kim, H.Y. (2019). Optimizing the Pairs-Trading Strategy Using Deep Reinforcement Learning with Trading and Stop-Loss Boundaries. *Complexity*, 2019, pp.1–20.

Krauss, C. (2016). Statistical Arbitrage Pairs Trading Strategies: Review and Outlook. *Journal of Economic Surveys*, 31(2), pp.513–545.

Krauss, C., Do, X.A. and Huck, N. (2017). Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. *European Journal of Operational Research*, 259(2), pp.689–702.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), pp.1735–1780.

Huang, C.-F., Hsu, C.-J., Chen, C.-C., Chang, B.R. and Li, C.-A. (2015). An Intelligent Model for Pairs Trading Using Genetic Algorithms. *Computational Intelligence and Neuroscience*, 2015, pp.1–10.

Huck, N. (2009). Pairs selection and outranking: An application to the S&P 100 index. *European Journal of Operational Research*, 196(2), pp.819–825.

Liew, R.Q. and Wu, Y. (2013). Pairs trading: A copula approach. *Journal of Derivatives & Hedge Funds*, 19(1), pp.12–30.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Workshop 2013*, arXiv: 1312.5602.

Rumelhart, D.E., Hinton, G.E., Williams, R.J. and Institute, D. (1985). *Learning internal representations by error propagation*. La Jolla, Calif.: Institute For Cognitive Science, University Of California, San Diego.

Sezer, O.B. and Ozbayoglu, A.M. (2018). Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, [online] 70, pp.525–538.

Sutton, R.S. and Barto, A. (2018). *Reinforcement learning : an introduction*. Cambridge, MA ; London: The MIT Press.

Troiano, L., Bhandari, A. and Villa, E.M. (2020). *Hands-On Deep Learning for Finance: implement deep learning techniques and algorithms to create ... powerful trading strategies.* Packt Publishing.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M. and Iosifidis, A. (2017). Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks. *2017 IEEE 19th Conference on Business Informatics (CBI).*

Vidyamurthy, G. (2004). *Pairs trading : quantitative methods and analysis*. Hoboken, N.J.: J. Wiley.